RZ/A3

Initial Program Loader Application Note (Octa Flash Boot Edition)

Introduction

This application note describes the following contents about the reference implementation of Initial Program Loader (called IPL), which is a boot loader for booting user programs from Octa Flash memory connected to RZ/A3 device.

- Description of IPL function
- Description of how to build the development environment
- Description of how to debug
- How to customize when supporting Octa Flash memory that is not supported by default IPL

Target Device

RZ/A3UL

When using this application note with other Renesas Microprocessors, sufficient evaluation is recommended after making modifications to comply with the alternate Microprocessor.

Contents

1. Overview 5
1.1 Glossary 5
1.2 IPL processing overview 6
2. Operating Confirmation Conditions 7
3. Reference Application Notes 8
4. Hardware 8
4.1 Peripheral Functions used 8
4.2 Pins Used List 9
5. Software 10
5.1 Operation overview 10
5.1.1 Operation image of boot process 10
5.1.2 Boot process to launch FSP application on Octa RAM 12
5.1.3 Boot process to launch FSP application on Octa Flash memory 14
5.2 Peripheral Function Setting 16
5.2.1 Settings of Octa Memory Controller Register 19
5.3 Memory Map 21
5.4 MMU setting 22
5.5 Interrupt used list 23
5.6 Exception level 23
5.7 Application Header 23
5.8 Data Types 24
5.9 Constants 25
5.10 Structures/Unions 28
5.11 Variables 36
5.12 Function 64
5.12.1 Function List 64
5.12.2 Function Specifications 67
5.12.2.1 IPL control functions 67
5.12.2.2 Octa Memory Controller driver functions 68
5.12.2.3 Octa Flash memory / Octa RAM driver functions 72
5.12.2.4 module control functions 78
5.13 Flow Chart 85
5.13.1 IPL (overall) flow 85
5.13.2 FSP Application Image copy process 88
5.13.3 Setting of Octa Flash memory, Octa RAM, and Octa Memory Controller 90
6. IPL development environment construction procedure 94
6.1 Procedure for building a development environment for Widows10 PC 94

6.1.1 Installation of IDE (e² studio) for Windows10 94
6.1.1.1 e² studio installation (Windows10) 95
6.1.1.2 Installation of RZ/A FSP packs (Windows10) 103
6.1.2 MSYS2 Installation 104
6.2 Procedure for building a development environment for Ubuntu 22.04 LTS 107
6.2.1 Installation of IDE (e² studio) for Ubuntu 22.04 LTS 107
6.2.1.1 e² studio installation (Ubuntu 22.04 LTS) 107
6.2.1.2 Installation of RZ/A FSP packs (ubuntu 22.04 LTS) 118
6.2.2 Additional Software Installation 119

7. IPL build environment construction procedure 120

8. Execution Environment of IPL 130
8.1 Create Debug Configuration 130
8.2 Connection to SMARC EVK Board 139
8.3 Setting Serial Terminal 141
8.4 Execution procedure of IPL by Octa Flash boot 145

9. Application Example 149
9.1 Execution procedure of IPL by On-chip RAM download 149
9.2 Procedure for writing an FSP application to Octa Flash memory 153
9.3 How to build IPL for custom board 158
9.3.1 How to customize IPL for your Octa Flash memory 158
9.3.1.1 flash_open 167
9.3.1.2 flash_close 171
9.3.1.3 flash_exec_op 172
9.3.1.4 flash_enter_xip 173
9.3.1.5 flash_exit_xip 176
9.3.1.6 flash_read 177
9.3.1.7 flash_write 178
9.3.1.8 flash_erase 183
9.3.1.9 flash_get_write_status 187
9.3.1.10 flash_write_enable_spi / flash_write_enable_dopi 189
9.3.1.11 flash_read_status_reister_spi / flash_read_status_register_dopi 192
9.3.1.12 flash_write_config_register2_spi 195
9.3.1.13 flash_erase_4k / flash_erase_64k 197
9.3.2 How to customize IPL for your Octa RAM 200
9.3.2.1 octaram_open 207
9.3.2.2 octaram_close 211
9.3.2.3 octaram_exec_op 212
9.3.2.4 octaram_enter_xip 213
9.3.2.5 octaram_exit_xip 217

9.3.2.6 octaram_read 218
9.3.2.7 octaram_write 218
9.3.2.8 octaram_erase 219
9.3.2.9 octaram_get_write_status 219
9.3.2.10 octaram_write_mode_register 220
9.3.3 How to add a board setting 222
9.3.4 How to add a Module for Clock Supply 225
9.3.5 How to add pin setting 227
9.3.6 How to change MMU setting 228

10. Precautions 229
10.1 About DDR SDRAM access prohibited area 229
10.2 About Octa RAM access prohibited area 229

11. Reference Documents 230

12. Trademarks 231

Revision History 232

1. Overview

IPL is a boot program for RZ/A3 device.

When starting the RZ/A3 device, IPL initializes the peripheral I/O modules, the memory installed on the board, etc., and launches the FSP application.

IPL consists of functions such as peripheral I/O module driver, memory control driver, and FSP application image load module.

1.1 Glossary

Table 1.1 shows the terms related to the operation of IPL explained in this application note.

Table 1.1 The term for IPL operation (1/2)

Term	Description
Initial Program Loader (IPL)	A loader program that launches an FSP Application for the RZ/A3 environment. It is called from the BootROM program to initialize the memory and the peripheral I/O modules installed in boards, then copies the FSP application and jumps to the copy destination according to the application header. IPL is developed based on Arm Trusted Firmware and it is equivalent to BL2 of Arm Trusted Firmware. When changing the serial flash memory to be used, the user is requested to create and add the initialization code according to the specifications of the Octa Flash memory to be used, referring to this application note. By default, optimal settings are made for using the Macronix Octa Flash memory(MX66UW1G45G).
BootROM Program	The BootROM program is the program that executes first after the device starts, and it copies the next boot program from the memory associated with the boot mode to the On-chip RAM and jumps to the next boot program. IPL is premised on operation in Boot Mode 3 (1.8-V serial Flash memory), and the IPL written to the serial flash memory is copied to the specified address (H'0001_2000) of the On-chip RAM and the boot process is executed. This program is stored in the RZ/A3's internal ROM, so it does not need to be created by the user.
FSP Application	This is a program consisting of BSP and sample programs to use the functions implemented on the board.
Application Header	A header placed at the beginning of the FSP application, used by the IPL to refer to the destination address or entry point address of the FSP application. Before executing IPL, the FSP application image including the application header must be written to the specified address of the serial flash memory. The write destination address of the Application Header can be changed in the IPL build option, and the default is H'2002_0000.
Octa Flash memory driver / Octa RAM driver	The Octa Flash memory driver and Octa RAM driver create parameters for executing operations that comply with the specifications of each memory and manage the execution order of operations. These drivers are implemented with specifications that depend on each memory. If you use the Octa Flash memory and Octa RAM other than the one supported by the default IPL, you need to change drivers according to the specifications.

Table 1.2 The term for IPL operation (2/2)

Term	Description
Configuration Mode	This is the memory access mode of the Octa Memory Controller. In this mode, data is read / written by sending Read and Write commands to the Octa Flash memory and Octa RAM via serial communication. Both Configuration Mode and Memory-map Mode can be used in Octa Memory Controller without switching the operation mode.
Memory-map Mode	This is the memory access mode of the Octa Memory Controller. By using this mode, the device can read / write data by performing read / write access to the address space where Octa Flash memory and Octa RAM are mapped. Both Configuration Mode and Memory-map Mode can be used in Octa Memory Controller without switching the operation mode.

1.2 IPL processing overview

The outline of IPL processing is described below.

1. Release the reset and supply the clock to the module used for the boot process.
2. Initialize the serial communication interface for debug message output and control the message input/output.
3. Change the settings of Octa Memory Controller.
4. Initialize DDR SDRAM.
5. Copy the FSP Application image from Octa Flash Memory to DDR SDRAM.
6. Jump to the entry point of FSP Application on DDR SDRAM.

2. Operating Confirmation Conditions

IPL has been confirmed to run in the following conditions.

Table 2.1 Operating Confirmation Conditions

Item	Contents
MPU	RZ/A3UL
Operating Frequency	CPU clock (Iϕ): 1.0GHz SRAM MCPU clock (I2ϕ): 200MHz Peripheral clock 0 (P0ϕ): 100MHz Peripheral clock 1 / SRAM ACPU clock (P1ϕ): 200MHz Octa controller main clock (OC0ϕ): 200MHz QSPI0_SPCLK (OC1ϕ): 100MHz
Operating Voltage	Power supply voltage (I/O): 3.3V Power supply voltage (1.8/3.3V switching I/O (SPI_PVDD)):1.8V
Host PC	Windows10 (64bit) installed PC or Ubuntu 22.04 LTS Desktop (64bit) installed PC
Debugger	J-Link Base J-Link Lite (*J-Link Software version: V8.10a)
IDE	e² studio 2025-01
Toolchain	Windows10 PC: gcc-arm-10.3-2021.07-mingw-w64-i686-aarch64-none-elf Ubuntu 22.04 LTS PC: gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf
	Compiler option (Excludes the addition of directory paths Release configuration: -march=armv8-a -mgeneral-regs-only -mstrict-align -ffunction-sections -fdata-sections -ffreestanding -fno-builtin -fno-common -Os -std=gnu99 -fno-stack-protector -Werror -Wall -Wmissing-include-dirs -Wunused -Wdisabled-optimization -Wvla -Wshadow -Wno-unused-parameter -Wredundant-decls -Wunused-but-set-variable -Wmaybe-uninitialized -Wpacked-bitfield-compat -Wshift-overflow=2 -Wlogical-op -Wno-error=deprecated-declarations -Wno-error=cpp Debug Configuration: -march=armv8-a -mgeneral-regs-only -mstrict-align -ffunction-sections -fdata-sections -ffreestanding -fno-builtin -fno-common -Os -std=gnu99 -fno-omit-frame-pointer -fno-stack-protector -Werror -Wall -Wmissing-include-dirs -Wunused -Wdisabled-optimization -Wvla -Wshadow -Wno-unused-parameter -Wredundant-decls -Wunused-but-set-variable -Wmaybe-uninitialized -Wpacked-bitfield-compat -Wshift-overflow=2 -Wlogical-op -Wno-error=deprecated-declarations -Wno-error=cpp
Operating Mode	Boot Mode3 (1.8-V serial Flash memory)
Board	RZ/A3UL-Evaluation-Board-Kit (Octal-SPI version) (In this document, it is called SMARC EVK board)
Device to use (Function used on the board)	• Octa Flash memory (Store the image of the program) Manufacture's name: Macronix Model Name: MX66UW1G45G • Octa RAM (Used as an execution area for FSP Application) Manufacture's name: AP Memory Model Name: APS51208N • DDR4 SDRAM (Used as an execution area for FSP Application) Manufacture's name: Micron Technology Model Name: MT40A512M16LY-062E

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RZ/A3 Initial Program Loader Application Note (QSPI Flash Boot Edition) (R01AN6413EJ)

4. Hardware

This chapter describes the hardware used by IPL.

4.1 Peripheral Functions used

IPL uses the peripheral features shown in Table 4.1.

Table 4.1 Peripheral functions and uses

Peripheral Function	Use
SPI Multi-I/O Bus Controller (SPIBSC)	Set to external space read mode and generate a signal for the CPU to read directly from the Octa Flash memory connected to the SPI multi-I/O bus space. IPL does not use the SPIBSC controller. On-chip ROM (Boot program) uses it.
Octa Memory Controller	Enable Memory-map Mode (memory-mapped reading and writing) to memory devices in separate flash and RAM. IPL uses the "Octa Controller Driver" to control the Octa Memory controller.
DDR3L/DDR4 SDRAM Memory Controller (MEMC)	Make settings to access DDR3L/DDR4 SDRAM connected to the external bus space.
Clock pulse generator (CPG)	Generate the operating frequency of the RZ/A3. Release the module standby mode of the RZ/A3's peripheral I/O modules.
General Purpose Input Output Port (GPIO)	Switch multiplexed pin functions for SCIFA channel 0 and set the SPI dedicated pins.
Serial communication interface with FIFO (SCIFA)	Used for message output to the host PC using SCIFA channel 0.
System Counter (SYC)	Generate the count value used by the generic timer built into the Cortex®-A55 core.
Memory Management Unit (MMU)	Generate a translation table that specifies the cache effective area and memory type for the RZ/A3 address space.
System Controller (SYSC)	System control of this LSI. Select the Octa Memory controller.

4.2 Pins Used List

Table 4.2 lists the pins used and their functions.

Table 4.2 Pins used and their functions

Pin name	I/O	Function
QSPI0_SPCLK	Output	Clock signal for an Octa Flash memory and an Octa RAM
QSPI0_IO0	Input/Output	IO0 signal for an Octa Flash memory and an Octa RAM
QSPI0_IO1	Input/Output	IO1 signal for an Octa Flash memory and an Octa RAM
QSPI0_IO2	Input/Output	IO2 signal for an Octa Flash memory and an Octa RAM
QSPI0_IO3	Input/Output	IO3 signal for an Octa Flash memory and an Octa RAM
QSPI0_SSL	Output	Chip select signal for an Octa Flash memory
QSPI_RESET#	Output	Reset signal for an Octa Flash memory and an Octa RAM
OM_CS1#	Output	Chip select signal for an Octa RAM
OM_DQS	Input/Output	Read data strobe input / Write mask signal for an Octa Flash memory and an Octa RAM
OM_SIO4	Input/Output	IO4 signal for an Octa Flash memory and an Octa RAM
OM_SIO5	Input/Output	IO5 signal for an Octa Flash memory and an Octa RAM
OM_SIO6	Input/Output	IO6 signal for an Octa Flash memory and an Octa RAM
OM_SIO7	Input/Output	IO7 signal for an Octa Flash memory and an Octa RAM
SCIF0_RXD	Input	Receive data signal for serial communication
SCIF0_TXD	Output	Transmit data signal for serial communication

5. Software

This chapter describes the IPL software configuration and functions.

5.1 Operation overview

When the SMARC EVK board is booted in Boot Mode 3 (1.8-V serial Flash memory), the Boot ROM program written in the internal ROM will start.

The IPL is copied from the Octa Flash memory located in the SPI multi I/O bus space to the On-chip RAM by the BootROM program and booted.

IPL gets the copy destination address and entry point address of the FSP application image from the application header placed at the beginning of the FSP application.

This chapter describes the case in RZ/A3UL as an example of the boot process.

After that, copy the FSP application to the obtained address and jump to the entry point address.
Please refer to 5.7 Application Header for application header configuration.

5.1.1 Operation image of boot process

Figure 5.1 shows an operation image of the basic boot process.

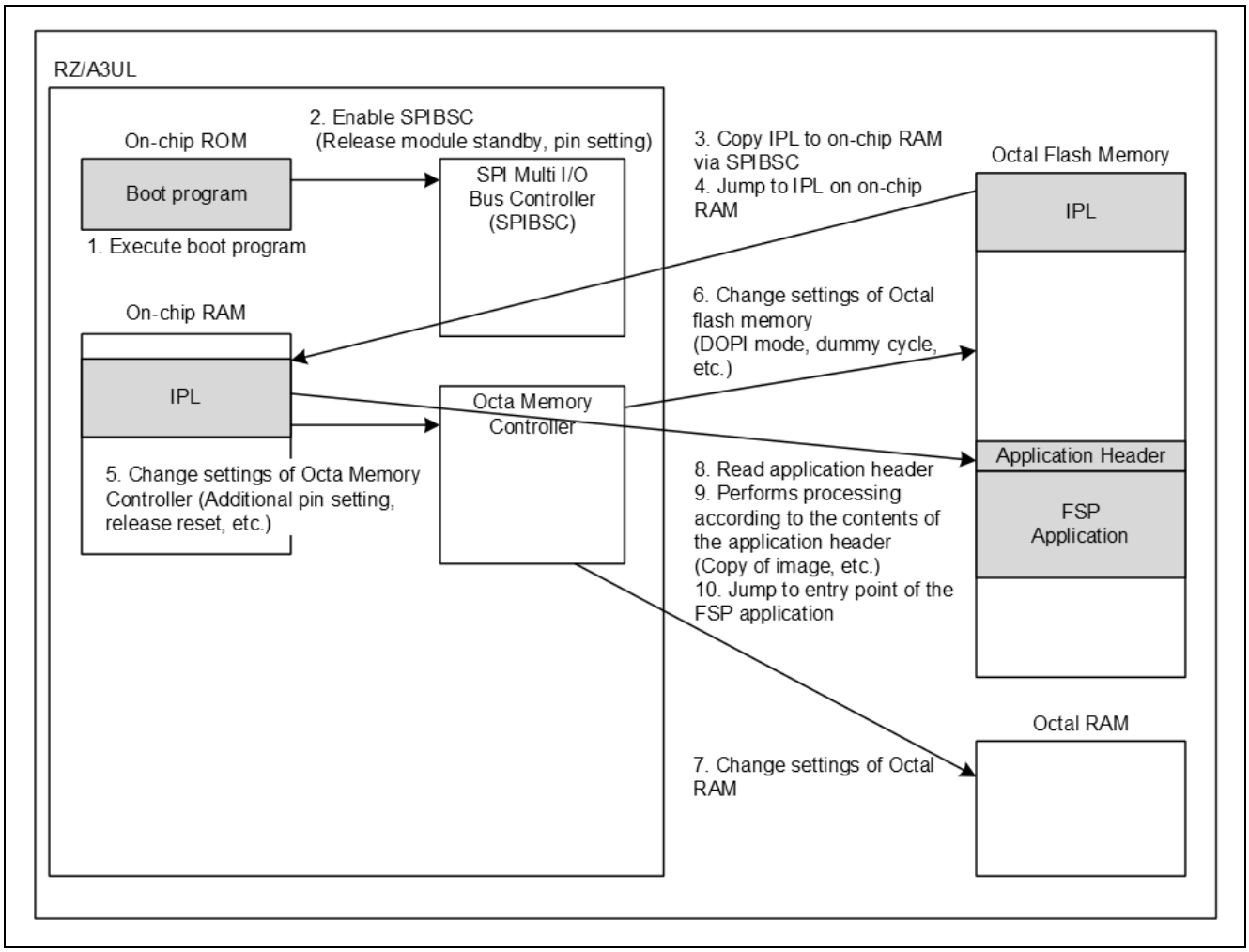


Figure 5.1 Operation image of boot process

Describes the IPL boot process.

1. RZ/A3UL executes the BootROM program after releasing the power-on reset.
2. When the RZ/A3UL operating mode is Boot mode 3, the BootROM program will enable SPIBSC and initialize it.
   The initial setting is that it can access independently of the model of Octa Flash memory.
3. The BootROM program reads the IPL stored in the start address (*1) of the Octa Flash memory and copies the IPL to the entry point address (*2) on the On-chip RAM.
   Then the SPIBSC is reset by the BootROM, and access to the Octa Flash memory will be prohibited.
4. The BootROM program jumps to the IPL entry point (* 2) on the On-chip RAM.
5. IPL makes additional pin settings and changes the settings to the optimum settings for the Octa Flash memory (MX66UW1G45G) and the Octa RAM (APS51208N) mounted on the SMARC EVK board.
6. A command is issued via the Octa Memory Controller to the Octa Flash memory to change the settings.
7. A command is issued via the Octa Memory Controller to the Octa RAM to change the settings.
8. IPL reads the application header.
9. If the application header specifies a copy destination address for the FSP application, IPL copies it.
   As an example of the process is shown in the following chapter.
10. The IPL follows the contents of the application header and jumps to the entry point of the FSP application.

    (*1) This address will be H'2000_0200 when direct access to Octa Flash memory is possible.
    (*2) The IPL entry point address is H'0001_2000.

5.1.2 Boot process to launch FSP application on Octa RAM

Figure 5.2 shows an operation image when an FSP application written to the Octa Flash memory is loaded into Octa RAM and executed.

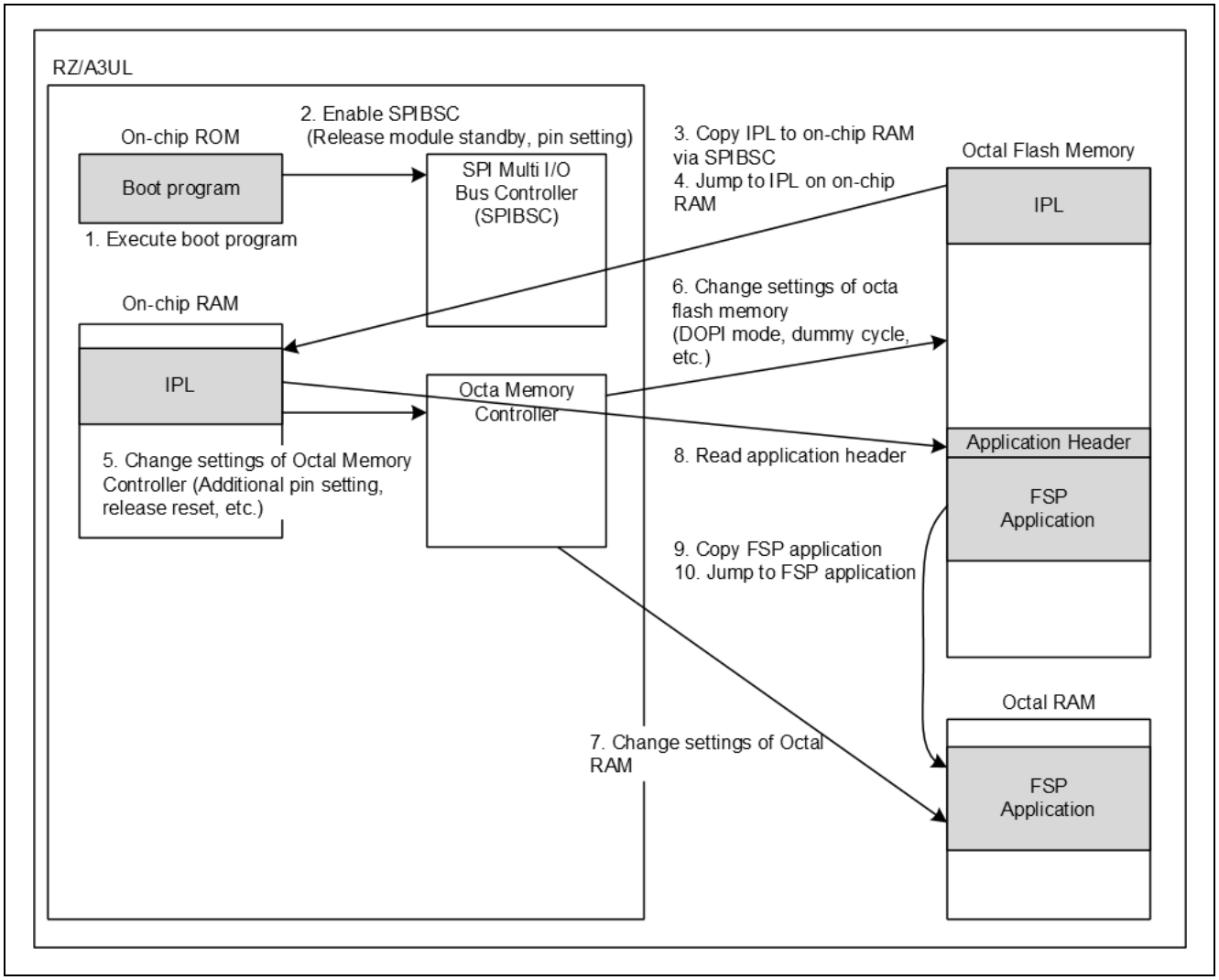


Figure 5.2 Operation image when launching an FSP application on Octa RAM

Describes the IPL boot process.

1. RZ/A3UL executes the BootROM program after releasing the power-on reset.
2. When the RZ/A3UL operating mode is Boot mode 3, the BootROM program will enable SPIBSC and initialize it.
   The initial setting is that it can access independently of the model of the Octa Flash memory.
3. The BootROM program reads the IPL stored in the start address of the Octa Flash memory and copies the IPL to the entry point address on the On-chip RAM.
   Then the SPIBSC is reset by the BootROM, and access to the Octa Flash memory will be prohibited.
4. The BootROM program jumps to the IPL entry point on the On-chip RAM.
5. IPL makes additional pin settings and changes the settings to the optimum settings for the Octa Flash memory (MX66UW1G45G) and the Octa RAM (APS51208N) mounted on the SMARC EVK board.
6. A command is issued via the Octa Memory Controller to the Octa Flash memory to change the settings.
7. A command is issued via the Octa Memory Controller to the Octa RAM to change the settings.
8. IPL reads the application header.
9. IPL will copy the FSP application to the Octa RAM according to the contents of the application header.
10. The IPL jumps to the entry point of the FSP application on the Octa RAM.

5.1.3 Boot process to launch FSP application on Octa Flash memory

Figure 5.3 shows an operation image when jumping directly to the FSP application written in the Octa Flash memory and executing on it.

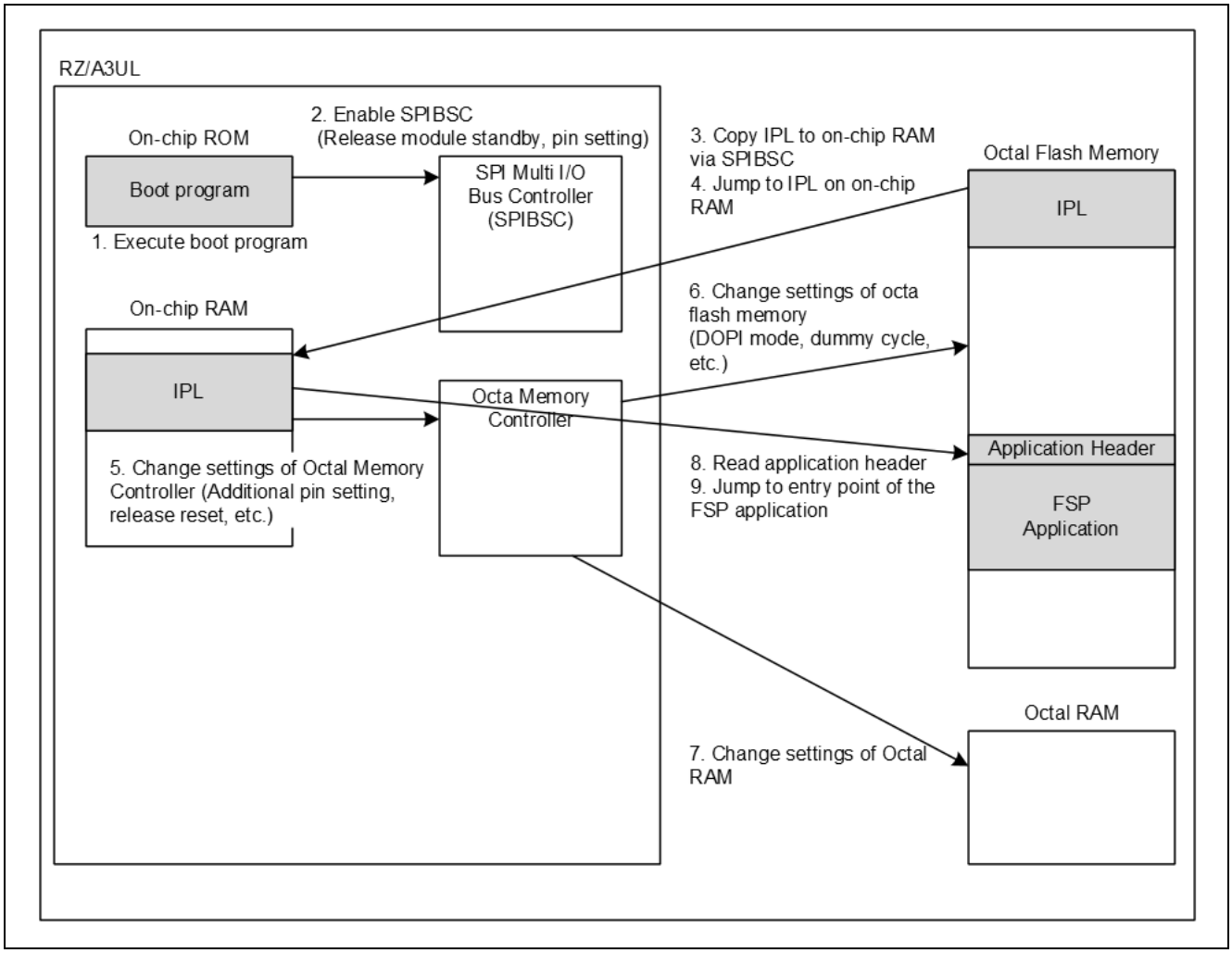


Figure 5.3 Operation image when booting FSP application on Octa Flash memory

Describes the IPL boot process.

1. RZ/A3UL executes the BootROM program after releasing the power-on reset.
2. When the RZ/A3UL operating mode is Boot mode 3, the BootROM program will enable SPIBSC and initialize it.
   The initial setting is that it can be accessed independently of the model of Octa Flash memory.
3. The BootROM program reads the IPL stored in the start address of the Octa Flash memory and copies the IPL to the entry point address on the On-chip RAM.
   Then the SPIBSC is reset by the BootROM, and access to the Octa Flash memory will be prohibited.
4. The BootROM program jumps to the IPL entry point on the On-chip RAM.
5. IPL makes additional pin settings and changes the settings to the optimum settings for the Octa Flash memory (MX66UW1G45G) and the Octa RAM (APS51208N)) mounted on the SMARC EVK board.
6. A command is issued via the Octa Memory Controller to the Octa Flash memory to change the settings.
7. A command is issued via the Octa Memory Controller to the Octa RAM to change the settings.
8. IPL reads the application header.
   If the copy destination address of the FSP application is the same as the stored address, IPL will skip the copy process.
9. IPL will jump to the entry point of the FSP application in Octa Flash memory according to the contents of the application header.

5.2 Peripheral Function Setting

Table 5.1 Lists the settings for Peripheral Functions during execution of IPL.

Table 5.1 Settings for Peripheral Functions (1/3)

Module	Settings
CPG	Set the following operating frequencies. • CPU clock (Iϕ): 1.0GHz Set to 1/1 times the frequency of the PLL1 (FOUT) • Peripheral clock 0 (P0ϕ): 100MHz Set to 1/8 times the frequency of the PLL2 (FOUT1PH0) • Peripheral clock 1 / SRAM ACPU clock (P1ϕ): 200MHz Set to 1/8 times the frequency of the PLL3 (FOUT1PH0) • SRAM MCPU clock (I2ϕ): 200MHz Set to 1/8 times the frequency of the PLL3 (FOUT1PH0) • SPI multi I/O main clock (SPI0ϕ): 33.313MHz Set to 1/8 times the frequency of the PLL3 (FOUT3) • Octa Memory Controller clock (OC0ϕ): 200MHz Set to 1/1 times the frequency of the PLL3 (FOUT4) • QSPI0_SPCLK (Before IPL): 16.656MHz 1/2 times of SPI0ϕ • QSPI0_SPCLK (After IPL): 100MHz 1/2 times of OC0ϕ Supply clock and release reset to the peripherals shown below Release the reset • AXI Bus • Peripheral Bus • On-chip RAM • Internal ROM • DDR SDRAM • OTP • GIC-600 • IA55 • SYSC • SYC • DMAC • WDT • GPIO • SPIBSC • Octa Memory Controller • SCIFA (ch0 only) • CoreSight™

Table 5.2 Settings for Peripheral Functions (2/3)

Module	Settings
Octa Memory Controller (Octa Flash)	• Command-Address-Data bits 8-8-8 (DOPI) • Opcode "8 I/O DT Read" EE11h • Address size 4-byte • Additional data 0-byte • Dummy cycles 10 QSPI0_SPCLK • SLCH 1.5 QSPI0_SPCLK (15ns) • CLSH 6.5 QSPI0_SPCLK (65ns) • SHSL 5 QSPI0_SPCLK (50ns)
Octa Memory Controller (Octa RAM Read)	• Command-Address-Data bits 8-8-8 (Octa RAM) • Opcode "Sync Read" 8000h • Address size 4-byte • Additional data 0-byte • Dummy cycles 4 QSPI0_SPCLK • SLCH 1.5 QSPI0_SPCLK (15ns) • CLSH 6.5 QSPI0_SPCLK (65ns) • SHSL 5 QSPI0_SPCLK (50ns)
Octa Memory Controller (Octa RAM Write)	• Command-Address-Data bits 8-8-8 (Octa RAM) • Opcode "Sync Write" 0000h • Address size 4-byte • Additional data 2-byte • Dummy cycles 4 QSPI0_SPCLK • SLCH 1.5 QSPI0_SPCLK (15ns) • CLSH 1.5 QSPI0_SPCLK (15ns) • SHSL 5 QSPI0_SPCLK (20ns)

Table 5.3 Settings for Peripheral Functions (3/3)

Module	Settings
GPIO	Set the following shared pin • P6_3: SCIF0_RXD • P6_4: SCIF0_TXD Set the following in P6_3 and P6_4 • Drive capacity: 4mA • Slew rate: Fast • Pull up: None • Pull down: None Set the following for the decided pins of QSPI0_IO0, QSPI0_IO1, QSPI0_IO2, QSPI0_IO3, QSPI0_SPCLK, QSPI0_SSL • Output impedance: 66Ω • Slew rate: Fast • Pull up: None Pull down: None
SCIFA	Set the following for Ch0 • Operation mode: asynchronous communication mode • Transfer rate: 115200bps • Data length: 8bit • Stop bit length: 1 bit • Parity: None • Flow control: disabled
SYC	Input 24MHz clock to Cortex®-A55 built-in Generic Timer

5.2.1 Settings of Octa Memory Controller Register

Table 5.2 shows the Octa Memory Controller register settings after the initial settings are completed.

These setting values include the results that reflect the settings described in the Octa Memory Controller section in Table 5.1 Peripheral Function Settings.

Table 5.4 Octa Memory Controller register setting value after initial setting (1/2)

Register	Settings
Device Size Register 0 (Octa Flash)	
DSR0.DV0TYP	B’00 (Octa Flash memory)
DSR0.DV0SZ	H’08000000 (128-Mbyte)
Device Size Register 1 (Octa RAM)	
DSR1.DV1TYP	B’01 (Octa RAM)
DSR1.DV1SZ	H’04000000 (64-Mbyte)
Address Extend Register	
AER.AAS	B’111 (64-Mbyte)
Memory Delay Trim Register	
MDTR.DQSEDOPI	12
MDTR.DQSESOPI	Not used in this IPL
MDTR.DQSERAM	5
MDTR.DV0DEL	Unknown (Calibrated value)
MDTR.DV1DEL	Unknown (Calibrated value)
Auto-Calibration Timer Register	
ACTR.CTP	H’10000000 (≈1.342-seconds)
Auto-Calibration Address Register 0	
ACAR0.CAD0	Unknown (Build-time dependent)
Auto-Calibration Address Register 1	
ACAR1.CAD1	H’00000000
Device Memory Map Read Chip Select Timing Setting Register	
DRCSTR.DVRDLO0	B’00 (1.5 QSPI0_SPCLK for Octa flash)
DRCSTR.DVRDHI0	B’101 (6.5 QSPI0_SPCLK for Octa flash)
DRCSTR.DVRDCMD0	B’001 (5 QSPI0_SPCLK for Octa flash)
DRCSTR.CTR0	B’1 (Enable continuous read for Octa flash)
DRCSTR.CTRW0	B’1111111 (≈1us or less for Octa flash)
DRCSTR.DVRDLO1	B’00 (1.5 QSPI0_SPCLK for Octa RAM)
DRCSTR.DVRDHI1	B’101 (6.5 QSPI0_SPCLK for Octa RAM)
DRCSTR.DVRDCMD1	B’001 (5 QSPI0_SPCLK for Octa RAM)
Device Memory Map Write Chip Select Timing Setting Register	
DWCSTR.DVWLO0	Not used in this IPL
DWCSTR.DVWHI0	Not used in this IPL
DWCSTR.DVWCMD0	Not used in this IPL
DWCSTR.DVWLO1	B’00 (1.5 QSPI0_SPCLK for Octa RAM)
DWCSTR.DVWHI1	B’000 (1.5 QSPI0_SPCLK for Octa RAM)
DWCSTR.DVWCMD1	B’001 (5 QSPI0_SPCLK for Octa RAM)

Table 5.5 Octa Memory Controller register setting value after initial setting (2/2)

Register	Settings
Controller and Device Setting Register	
CDSR.DLFT	B’1 (Disable deadlock free timer)
CDSR.ACMODE	B’01 (Enable automatic calibration)
CDSR.ACMEME	B’11 (Enable automatic calibration for both channel)
CDSR.DV0PC	B’0 (Disable pre-cycle for Octa Flash memory)
CDSR.DV0TTYP	B'10 (DOPI mode for Octa Flash memory)
CDSR.DV1PC	B’0 (Disable pre-cycle for Octa RAM)
CDSR.DV1TTYP	B’10 (DOPI mode for Octa RAM)
Memory Map Dummy Length Register	
MDLR.DV0WDL	0 (Not used in this IPL)
MDLR.DV0RDL	10 (For Octa flash)
MDLR.DV1WDL	4 (For Octa RAM)
MDLR.DV1RDL	4 (For Octa RAM)
Memory Map Read/Write Command Register 0	
MRWCR0.D0MWCMD1	0x00 (Not used in this IPL)
MRWCR0.D0MWCMD0	0x00 (Not used in this IPL)
MRWCR0.D0MRCMD1	0xEE (8 I/O DT Read)
MRWCR0.D0MRCMD0	0x11 (8 I/O DT Read)
Memory Map Read/Write Command Register 1	
MRWCR1.D1MWCMD1	0x00 (Sync Write)
MRWCR1.D1MWCMD0	0x00
MRWCR1.D1MRCMD1	0x80 (Sync Read)
MRWCR1.D1MRCMD0	0x00
Memory Map Read/Write Setting Register	
MRWCSR.MWCL0	0 (Not used in this IPL)
MRWCSR.MWAL0	0 (Not used in this IPL)
MRWCSR.MWO0	0 (Not used in this IPL)
MRWCSR.MRCL0	2 (2-byte command for Octa Flash memory)
MRWCSR.MRAL0	4 (4-byte address for Octa Flash memory
MRWCSR.MRO0	1 (1-0-3-2 order for Octa Flash memory)
MRWCSR.MWCL1	2 (2-byte command for Octa RAM)
MRWCSR.MWAL1	4 (4-byte address for Octa RAM
MRWCSR.MWO1	1 (1-0-3-2 order for Octa RAM)
MRWCSR.MRCL1	2 (2-byte command for Octa RAM)
MRWCSR.MRAL1	4 (4-byte address for Octa RAM
MRWCSR.MRO1	1 (1-0-3-2 order for Octa RAM)
Device Reset Register	
RSTCNT.RSTVAL	1 (O_MEM_RESET_N=H)

5.3 Memory Map

Figure 5.4 shows the RZ/A3UL address space and the memory map of the SMARC EVK board.

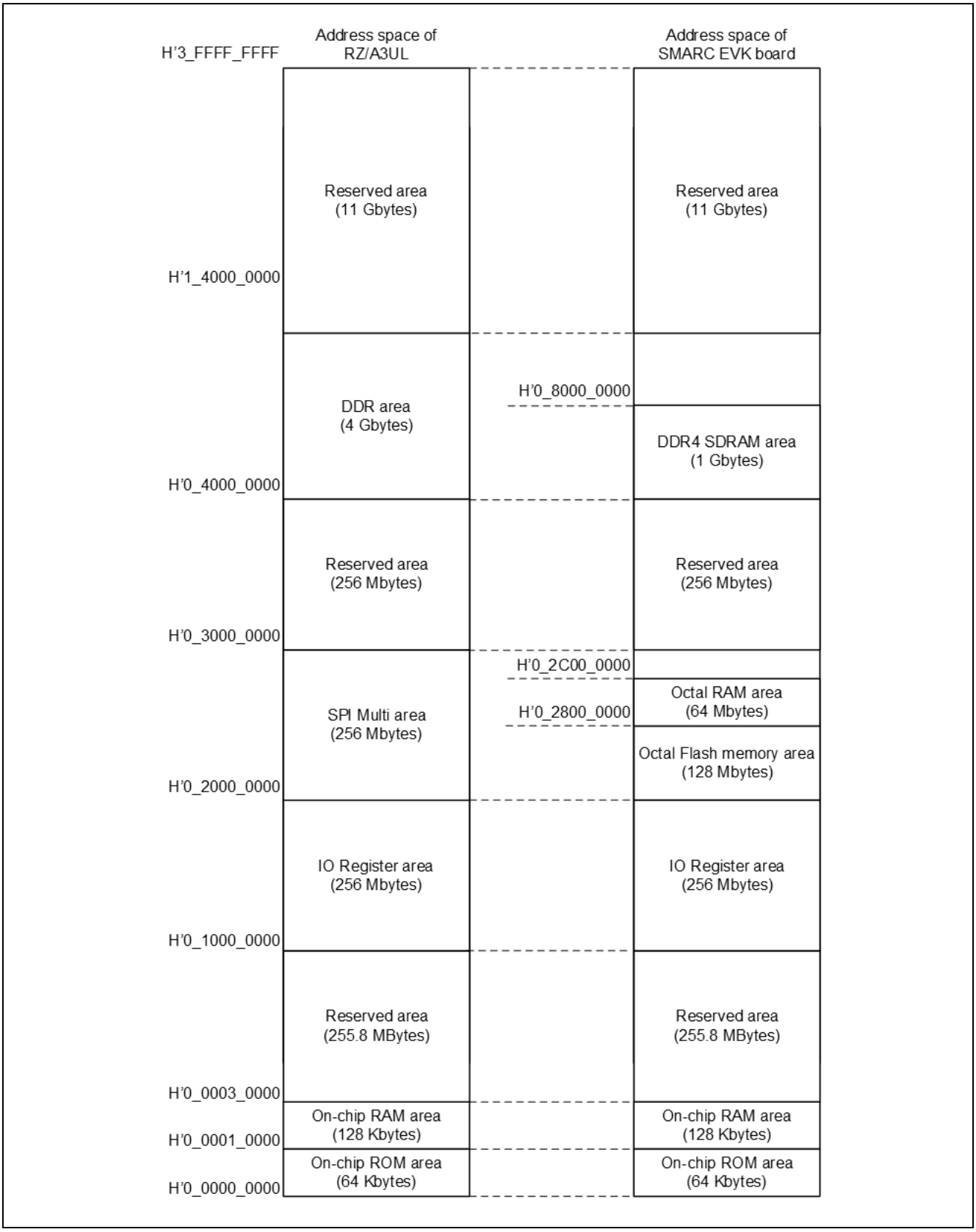


Figure 5.4 Memory MAP

5.4 MMU setting

Figure 5.5 shows the IPL MMU settings.

The MMU is disabled when jumping to the FSP Application.

Make the optimum settings in the FSP application if necessary.

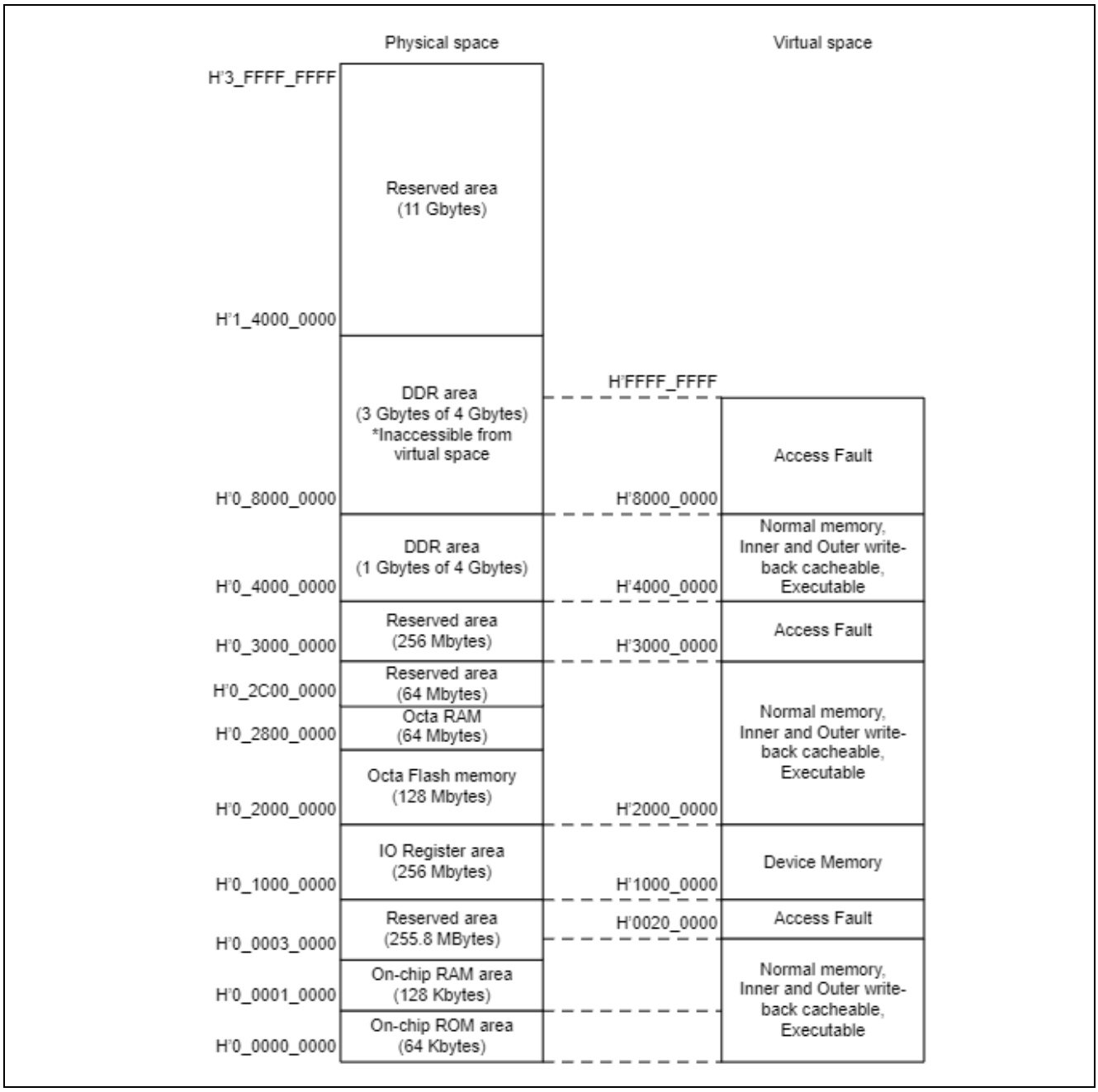


Figure 5.5 MMU setting

5.5 Interrupt used list

IPL does not use interrupt.

5.6 Exception level

IPL operates at exception level EL3.

It also jumps to the FSP Application in the state of EL3.

5.7 Application Header

IPL obtains the save address, copy destination address and entry point of the FSP Application image according to the application header written to the Octa Flash memory, copies the image, and jumps to the FSP Application.

IPL loads the application header from the address set in "FSP_BASE" in the make option.

Write the application header in advance in the format shown in the figure from the address set in "FSP_BASE" before executing IPL.

If "FSP_BASE" is not specified in the make command, H'2002_0000 is set as the default value. The header structure is shown below.

Table 5.6 Application Header format

offset	size	Description
+0h	QWORD(8bytes)	FSP entry address
+8h	QWORD(8bytes)	Inverted FSP entry point address This is used to check if the header data is correct
+10h	QWORD(8bytes)	Address to copy the FSP image payload If FSP image payload does not need to transfer, this field must be set FSP payload address (FSP_BASE + 200h)
+18h	QWORD(8bytes)	Inverted address to copy the FSP image payload This is used to check if the header data is correct
+20h	QWORD(8bytes)	FSP image payload size
+28h	QWORD(8bytes)	Inverted FSP image payload size This is used to check if the header data is correct
+30h	-	Reserved (as 0)
+1E0h	String (MAX 32 bytes)	Signature "RZ FSP Application" and fill gap as 0
+200h later	-	FSP image payload

5.8 Data Types

Table 5.7 lists the data types used in IPL.

Table 5.7 Data Types used in the IPL

Symbol	Description
bool	Boolean type. Value is true (1) or false (0)
char	8-bit character.
int	Fast integer signed, 32-bit integer in the IPL.
int8_t	8-bit integer, signed (defined in standard library stdint.h)
int16_t	16-bit integer, signed (defined in standard library stdint.h)
int32_t	32-bit integer, signed (defined in standard library stdint.h)
int64_t	64-bit integer, signed (defined in standard library stdint.h)
uint8_t	8-bit integer, unsigned (defined in standard library stdint.h)
uint16_t	16-bit integer, unsigned (defined in standard library stdint.h)
uint32_t	32-bit integer, unsigned (defined in standard library stdint.h)
uint64_t	64-bit integer, unsigned (defined in standard library stdint.h)
unsigned long long	64-bit integer, unsigned
unsigned int	32-bit integer, unsigned
size_t	64-bit integer, unsigned
uintptr_t	Unsigned integer type with pointer type size, 64-bit unsigned integer in the IPL.

5.9 Constants

Table 5.8 to Table 5.10 lists the constants used in IPL.
Table 5.8 to Table 5.10 lists only constants used in the RZ/A3UL platform boot process.

Please refer to https://trustedfirmware-a.readthedocs.io/en/latest/ for constants related to behavior of Arm Trusted Firmware.

Table 5.8 Constants used in the IPL (1/3)

Constant	Setting Value	Description
RZ_IPL_VERSION_STRING	"v2.0.0"	IPL version
CPG_OFF	(0)	Flag to set ON/OFF of CPG register Set the register to off (disable)
CPG_ON	(1)	Flag to set ON/OFF of CPG register Set the register to on (enable)
CPG_T_CLK	(0)	Flag that specifies the type of CPG register to set Set a value in the clock register
CPG_T_RST	(1)	Flag that specifies the type of CPG register to set Set a value in the reset register
CPG_SEL_PLL1_ON_OFF	(0)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL1
CPG_SEL_PLL2_1_ON_OFF	(1)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL2_1
CPG_SEL_PLL2_2_ON_OFF	(2)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL2_2
CPG_SEL_PLL3_1_ON_OFF	(3)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL3_1
CPG_SEL_PLL3_2_ON_OFF	(4)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL3_2
CPG_SEL_PLL3_3_ON_OFF	(5)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL3_3
CPG_SEL_PLL5_1_ON_OFF	(6)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL5_1
CPG_SEL_PLL5_3_ON_OFF	(7)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL5_3
CPG_SEL_PLL5_4_ON_OFF	(8)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL5_4
CPG_SEL_PLL6_1_ON_OFF	(9)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for PLL6_1
CPG_SEL_GPU1_1_ON_OFF	(10)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for GPU1_1

Table 5.9 Constants used in the IPL (2/3)

Constant	Setting Value	Description
CPG_SEL_GPU1_2_ON_OFF	(11)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for GPU1_2
CPG_SEL_GPU2_ON_OFF	(12)	Number for specifying the element of the source clock selector setting table of PLL Specify selector for GPU2
PFC_OFF	(0)	PFC disable setting flag
PFC_ON	(1)	PFC enable setting flag
XSPI_CLOCK_SPIM	(0)	Clock setting flag for Serial Flash Memory
XSPI_CLOCK_OCTA	(1)	Clock setting flag for Octa Flash memory/Octa RAM
CPG_CLOCK_SPIM	(20)	CPG clock ID for Serial Flash Memory
CPG_CLOCK_OCTA	(61)	CPG clock ID for Octa Flash memory/Octa RAM
DEFAULT_SPI_FREQUENCY	(1000000)	Default operating frequency of serial flash memory
RESET_DURATION_US	(10)	SPI device reset time (10usec)
WRITE_STATUS_OK	(0)	xSPI device write status Successful write to xSPI device
WRITE_STATUS_IN_PROGRESS	(1)	xSPI device write status While writing to the xSPI device
WRITE_STATUS_PROTECTED	(2)	xSPI device write status xSPI devices are write protected
WRITE_STATUS_FAIL	(3)	xSPI device write status Write failure to xSPI device
WRITE_STATUS_SUSPENDED	(4)	xSPI device write status Write suspend to xSPI device
SPI_FORM_1_1_1	(0)	Command executes at 1-1-1 form
SPI_FORM_1_1_2	(1)	Command executes at 1-1-2 form
SPI_FORM_1_2_2	(2)	Command executes at 1-2-2 form
SPI_FORM_2_2_2	(3)	Command executes at 2-2-2 form
SPI_FORM_1_1_4	(4)	Command executes at 1-1-4 form
SPI_FORM_1_4_4	(5)	Command executes at 1-4-4 form
SPI_FORM_4_4_4	(6)	Command executes at 4-4-4 form
SPI_FORM_1_1_8	(7)	Command executes at 1-1-8 form
SPI_FORM_1_8_8	(8)	Command executes at 1-8-8 form
SPI_FORM_8_8_8	(9)	Command executes at 8-8-8 form
XSPI_FEATURE_FORM112	(1 << 0)	The driver supports 1-1-2 form.
XSPI_FEATURE_FORM122	(1 << 1)	The driver supports 1-2-2 form.
XSPI_FEATURE_FORM222	(1 << 2)	The driver supports 2-2-2 form.
XSPI_FEATURE_DUAL	(0x07)	The driver supports 1-1-2, 1-2-2, and/or 2-2-2 form(s).
XSPI_FEATURE_FORM114	(1 << 3)	The driver supports 1-1-4 form.
XSPI_FEATURE_FORM144	(1 << 4)	The driver supports 1-4-4 form.
XSPI_FEATURE_FORM444	(1 << 5)	The driver supports 4-4-4 form.
XSPI_FEATURE_QUAD	(0x38)	The driver supports 1-1-4, 1-4-4, and/or 4-4-4 form(s).

Table 5.10 Constants used in the IPL (3/3)

Constant	Setting Value	Description
XSPI_FEATURE_FORM118	(1 << 6)	The driver supports 1-1-4 form.
XSPI_FEATURE_FORM188	(1 << 7)	The driver supports 1-8-8 form.
XSPI_FEATURE_FORM888	(1 << 8)	The driver supports 8-8-8 form.
XSPI_FEATURE_OCTA	(0x1C0)	The driver supports 1-1-8, 1-8-8, and/or 8-8-8 form(s).
XSPI_FEATURE_XIP_READ	(1 << 12)	The driver supports read access.
XSPI_FEATURE_XIP_WRITE	(1 << 13)	The driver supports write access.
XSPI_FEATURE_DDROP	(1 << 14)	The driver supports DDR for op code.
XSPI_FEATURE_DDR	(1 << 15)	The driver supports DDR for address and/or data.
XSPI_FEATURE_OCTARAM	(1 << 16)	Octa RAM addressing (RAS/CAS) supported.
XSPI_FLAGS_SEQUENTIAL_DDR	(1u << 0)	Enable inverted order for DDR transfer
XSPI_FLAGS_FORCE_STROBE	(1u << 1)	Force to use data strobe signal for timing reference
XSPI_FLAGS_DATA_ACCESS	(1u << 2)	Force to use data access method for exec_op
RZ_NOFIP	(0)	How to load the Application image Use the Application Header when loading the Application image
RZ_FIP	(1)	How to load the Application image: Use FIP format image when loading Application image
RZ_APP_PAYLOAD_OFFSET	(0x200)	Offset to payload data in Application header
RZA_MMU_ATTR_NON_CACHEABLE	(0x44)	MMU descriptor normal memory, non-cacheable memory attribute
RZA_MMU_ATTR_DEVICE	(0x04)	MMU descriptor device memory attribute
RZA_MMU_ATTR_IWBWA_OWBWA _NTR	(0xFF)	MMU descriptor normal memory, cacheable memory attribute
RZA_MMU_ATTRIBUTE_ACCESS_FAULT	(0x00000000)	MMU descriptor Access Fault
RZA_MMU_ATTRIBUTE_CONFIG _END	(0xFFFFFFFF)	End of MMU configuration table setting

5.10 Structures/Unions

Table 5.11 to Table 5.12 lists the structures/unions used in IPL.
Table 5.11 to Table 5.12 lists only structures/unions used in the RZ/A3UL platform boot process.

Please refer to https://trustedfirmware-a.readthedocs.io/en/latest/ for structures/unions related to behavior of Arm Trusted Firmware.

Table 5.11 List of structures used in the IPL (1/2)

Structure/Union	Description	Detail
fsp_app_header_t	Structure to store the Application Header	Refer to Table 5.13
octaflash_mx66uw_ctrl_t	Structure to store the control information of MX66UW1G45G	Refer to Table 5.14
octaflash_mx66uw_ext_t	Structure to store the extended information of MX66UW1G45G	Refer to Table 5.15
octaram_apsxx_ctrl_t	Structure to store the control information of APS51208N	Refer to Table 5.16
octaram_apsxx_ext_t	Structure to store the extended information of APS51208N	Refer to Table 5.17
octa_ctrl_t	Structure to store the control information of xSPI module	Refer to Table 5.18
octa_ext_t	Structure to store the extended information of xSPI module	Refer to Table 5.19
xspidevice_info_t	Structure to store the device information of xSPI device	Refer to Table 5.20
xspidevice_cfg_t	Structure to store the configure information of xSPI device	Refer to Table 5.21
xspidevice_api_t	Structure to store the API functions that control xSPI device	Refer to Table 5.22
xspidevice_instance_t	Structure to store the instance that control xSPI device	Refer to Table 5.23
xspi_cfg_t	Structure to store the configure information of xSPI interface	Refer to Table 5.24
xspi_op_t	Structure to store the operation of xSPI interface	Refer to Table 5.25 Refer to Table 5.26
xspi_api_t	Structure to store the API functions of the xSPI interface	Refer to Table 5.27
xspi_instance_t	Structure to store the instance of the xSPI interface	Refer to Table 5.28
CPG_SETUP_DATA	A structure that stores the startup setting data of the CPG register	Refer to Table 5.29
CPG_REG_SETTING	A structure that stores the set values for the CPG registers	Refer to Table 5.30
CPG_PLL_SETDATA_146	Structure to save setting parameters for PLL1/4/6	Refer to Table 5.31
CPG_PLL_SETDATA_235	Structure to save setting parameters for PLL2/3/5	Refer to Table 5.32
cpg_spi_clk_table	Structure to store clock setting parameters for serial flash memory	Refer to Table 5.33
PFC_REG_UINT8	A structure that stores the setting parameters for the 8byte width PFC register	Refer to Table 5.34
PFC_REG_UINT32	A structure that stores the setting parameters for the 32byte width PFC register	Refer to Table 5.35

Table 5.12 List of structures used in the IPL (2/2)

Structure/Union	Description	Detail
PFC_REG_UINT64	A structure that stores the setting parameters for the 64byte width PFC register	Refer to Table 5.36
PFC_REGS	A structure that stores the set values for the PFC registers	Refer to Table 5.37
rz_mmu_pgtbl_cfg_t	Structure that stores MMU page table configuration parameters	Refer to Table 5.38

Table 5.13 Application Header structure (fsp_app_header_t)

Member	Description
uint64_t entry_addr	FSP entry point address
uint64_t entry_addr_inv	Inverted FSP entry point address
uint64_t dest_addr	Address to copy the FSP image payload
uint64_t dest_addr_inv	Inverted address to copy the FSP image payload
uint64_t payload_size	FSP image payload size
uint64_t payload_size_inv	Inverted FSP image payload size
uint8_t padding[432]	Reserved
uint8_t signature[32]	Signature
uint8_t app_payload_start	Application image payload

Table 5.14 MX66UW1G45G control structure (octaflash_mx66uw_ctrl_t)

Member	Description
bool opened	Opened or not
const xspi_instance_t * xspi	xSPI control driver instance

Table 5.15 MX66UW1G45G extended information structure (octaflash_mx66uw_ext_t)

Member	Description
-	Member not defined because extended information is not used in default IPL

Table 5.16 APS51208N control structure (octaram_apsxx_ctrl_t)

Member	Description
bool opened	Opened or not
const xspi_instance_t * xspi	xSPI control driver instance

Table 5.17 APS51208Nextended information structure (octaram_apsxx_ext_t)

Member	Description
-	Member not defined because extended information is not used in default IPL

Table 5.18 xSPI module control structure (octa_ctrl_t)

Member	Description
bool opened	Opened or not
uint8_t device_type	Device type value (Species Octa Flash memory or Octa RAM) A3UL sticks this value for each channel, CH0 is Octa Flash memory and CH1 is Octa RAM. It may be variable in other processor or groups.
uintptr_t reg_base	Pointer to register base address
uintptr_t mmap_base	Pointer to base address of area allocated in Memory-map Mode
uint32_t calibration_base	Offset address for calibration pattern data based on the device top
size_t mmap_size	Size allocated in Memory-map Mode (as byte)
uint8_t channel	Channel number (0 based)

Table 5.19 xSPI module extended information structure (octa_ext_t)

Member	Description
uint32_t device_size	The target device size Default: XIP mapped size (128M) [Octa Flash memory] If this size is smaller than actual device size, it will not be able to access all regions of that device. [Octa RAM] If device size is not 128M, don't forget set this parameter. If this size is not equal to actual device size, it will not be able to access target RAM correctly.
uint32_t calibration_base	Offset address for calibration pattern data based on the device top Default: [Octa Flash memory] Automatic detect the pattern. If you want to keep the default, set UINT32_MAX. Default: [Octa RAM] Top of the device.

Table 5.20 xSPI device information structure (xspidevice_info_t)

Member	Description
uint32_t capacity	Amount of memory by bytes.
uint32_t minimum_erase_size	Minimum erase size by bytes.
char decide_vendor[32]	Device vendor name.
char device_product[32]	Device product name.

Table 5.21 xSPI device configure structure (xspidevice_cfg_t)

Member	Description
xspi_instance_t *xspi	Associated xSPI instance.
void const *extend	device specific configuration data.

Table 5.22 xSPI device API function structure (xspidevice_api_t)

Member	Description
int (* open)(xspidevice_ctrl_t * ctrl, xspidevice_cfg_t const * cfg)	Open function
int (* close)(xspidevice_ctrl_t * ctrl)	Close function
int (* exec_op)(xspidevice_ctrl_t * ctrl, xspi_op_t const * op, bool is_write)	Operation Execution function
int (* enter_xip)(xspidevice_ctrl_t * ctrl)	Entering in-place access mode function
int (* exit_xip)(xspidevice_ctrl_t * ctrl)	Exiting in-place access mode function
int (* get_info)(xspidevice_ctrl_t * ctrl, xspidevice_info_t * info)	Getting device information function
int (* read)(xspidevice_ctrl_t * ctrl, void * buffer, size_t address, size_t length)	Read function
int (* write)(xspidevice_ctrl_t * ctrl, void const * buffer, size_t address, size_t length)	Write function
int (* erase)(xspidevice_ctrl_t * ctrl, size_t address, size_t length)	Erase function
enum xspidevice_write_status (* get_write_status)(xspidevice_ctrl_t * ctrl)	Getting write status function

Table 5.23 xSPI device instance structure (xspidevice_instance_t)

Member	Description
xspidevice_ctrl_t * ctrl	Control parameter
xspidevice_cfg_t const * cfg	Configuration parameter
xspidevice_api_t const * api	API functions

Table 5.24 xSPI interface control structure (xspi_cfg_t)

Member	Description
uint8_t channel	Channel number
uintptr_t base	Pointer to register base address
void const * extend	IP specific configuration data

Table 5.25 xSPI interface operation structure (xspi_op_t) (1/2)

Member	Description
xspi_transfer_form_t form	Specify the form to send this operation It is possible to set the parameters shown in the following SPI_FORM_X_Y_Z format X, Y, and Z indicate the number of I/O data lines used when transferring the following information in operation X: Operation Code Y: Address Z: Data SPI_FORM_1_1_1 SPI_FORM_1_1_2 SPI_FORM_1_2_2 SPI_FORM_2_2_2 SPI_FORM_1_4_4 SPI_FORM_4_4_4 SPI_FORM_1_1_8 SPI_FORM_1_8_8 SPI_FORM_8_8_8
uint16_t op	Operation code (high side used as second byte)
uint8_t op_size	Operation code size (0 to 2)
bool op_is_ddr	If true, the op-code phase sends as DDR
uint32_t address	Address on Serial Flash Memory
uint8_t address_size	Address size (0 to 4)
bool address_is_ddr	If true, the address phase sends as DDR
uint8_t additional_size	Additional data size as bytes after the address phase If address_is_ddr is true, this cycle is also as DDR, and this size must be zero or even.
uint32_t additional_value	Additional data value • [0-byte] Not used • [1-byte] Use b0-b7 as value • [2-byte] Use b8-b15 as 1st value and b0-b7 as 2nd value • [3-byte] Use b16-b23 as 1st value, b8-b15 as 2nd value and b0-b7 as 3rd value • [4-byte] Use b24-b31 as 1st value, b16-b23 as 2nd value, b8-b15 as 3rd value and b0-b7 as 4th value
uint8_t dummy_cycles	Dummy cycles as the SPCLK count after additional data phase. This cycle does not include additional data phase size.
size_t transfer_size	Transfer size as bytes (ignored for Memory-map Mode configuration)
void * transfer_buffer	Transfer buffer pointer (ignored for Memory-map Mode configuration)
bool transfer_is_ddr	Whether the transfer phase is DDR.
uint8_t transfer_flag	Additional transfer flags These flags can be bitwise-or' d • b0: Enable inverted order for DDR transfer (0: w1 w0 w3 w2, 1: w0 w1 w2 w3) • b1: Force to use of data strobe for timing reference If not specified, operation executes as register access • b2: Force to use of data strobe signal for timing reference

Table 5.26 xSPI interface operation structure (xspi_op_t) (2/2)

uint8_t force_idle_level_mask	Whether or not to force each IO level during idle state Not all controllers support it The idle state does not include the dummy cycle. The dummy cycle is always Hi-Z • b0: Ignored • b1 – b7: 0: Always Hi-Z, 1: Forcibly outs the force_idle_level_value's state
uint8_t force_idle_level_value	IO level during idle state Only active if force_idle_mask of the same position is 1 • b0: Ignored • b1 – b7: 0: Low, 1: High
uint8_t slch_value	SSL Low to Clock High delay cycles as the SPCLK count (also known as CS# Active Setup Time)
uint8_t clsh_value	Clock Low to SSL High delay cycles as the SPCLK count (also known as CS# Active Hold Time)
uint8_t shsl_value	SSL High to SSL Low delay cycles as the SPCLK count (also known as CS# Deselect Time)

Table 5.27 xSPI interface API function structure (xspi_api_t)

Member	Description
int (* open)(xspi_ctrl_t * ctrl, xspi_cfg_t const * const cfg)	Open function
int (* close)(xspi_ctrl_t * const ctrl)	Close function
int (* post_init)(xspi_ctrl_t * const ctrl)	Post initialization function
int (* exec_op)(xspi_ctrl_t * const ctrl, xspi_op_t const * const op, bool is_write)	Operation Execution function
int (* configure_xip)(xspi_ctrl_t * const ctrl, xspi_op_t const * const rop, xspi_op_t const * const wop)	Configure for in-place access mode function
int (* start_xip)(xspi_ctrl_t * const ctrl)	Start in-place access mode function
int (* stop_xip)(xspi_ctrl_t * const ctrl)	Stop in-place access mode function
int (* run_manual_calibration)(xspi_ctrl_t * const ctrl)	Manual calibration function
int (* enable_auto_calibration)(xspi_ctrl_t * const ctrl)	Enable auto calibration function
int (* disable_auto_calibration)(xspi_ctrl_t * const ctrl)	Disable auto calibration function
int (* set_frequency)(xspi_ctrl_t * const ctrl, int frequency_hz)	Setting SPI frequency function
int (* inv_mmap)(xspi_ctrl_t * const ctrl)	Invalidate cache for entire MMAP function
int (* clean_mmap)(xspi_ctrl_t * const ctrl)	Clean cache for entire MMAP function
uintptr_t (* get_mmap_base)(xspi_ctrl_t * const ctrl)	Getting MMAP base address
size_t (* get_mmap_size)(xspi_ctrl_t * const ctrl)	Getting MMAP size
uint32_t (* get_features)(xspi_ctrl_t * const ctrl)	Getting supported features flag

Table 5.28 xSPI interface instance structure (xspi_instance_t)

Member	Description
xspi_ctrl_t * ctrl	Pointer to the control structure for this instance
xspi_cfg_t const * cfg	Pointer to the configuration structure for this instance
xspi_api_t const * api	Pointer to the API structure for this instance

Table 5.29 CPG setup data structure (CPG_SETUP_DATA)

Member	Description
uintptr_t reg	Pointer to CPG register
uintptr_t mon	CPG monitor register corresponding to the register specified by "reg"
uint32_t val	Setting value to CPG register
uint32_t type	Select the type of register to be set from the following CPG_T_CLK(clock operation) CPG_T_RST(reset operation)

Table 5.30 CPG register setting structure (CPG_REG_SETTING)

Member	Description
uintptr_t reg	Pointer to CPG register
uint32_t val	Setting value to CPG register

Table 5.31 PLL1/4/6 setting structure (CPG_PLL_SETDATA_146)

Member	Description
CPG_REG_SETTING clk1_dat	Register setting to CPG_SAMPLL1/4/6_CLK1
CPG_REG_SETTING clk2_dat	Register setting to CPG_SAMPLL1/4/6_CLK2
CPG_REG_SETTING stby_dat	Register setting to CPG_SAMPLL1/4/6_STBY

Table 5.32 PLL2/3/5 setting structure (CPG_PLL_SETDATA_235)

Member	Description
CPG_REG_SETTING clk1_dat	Register setting to CPG_SAMPLL5_CLK1 CPG_SAMPLL2/3_CLK1 register is not implemented in RZ/A3UL
CPG_REG_SETTING clk2_dat	Register setting to CPG_SAMPLL5_CLK2 CPG_SAMPLL2/3_CLK2 register is not implemented in RZ/A3UL
CPG_REG_SETTING clk2_dat	Register setting to CPG_SAMPLL5_CLK3 CPG_SAMPLL2/3_CLK3 register is not implemented in RZ/A3UL
CPG_REG_SETTING clk4_dat	Register setting to CPG_SAMPLL5_CLK4 CPG_SAMPLL2/3_CLK2 register is not implemented in RZ/A3UL
CPG_REG_SETTING clk5_dat	Register setting to CPG_SAMPLL5_CLK5 CPG_SAMPLL2/3_CLK2 register is not implemented in RZ/A3UL
CPG_REG_SETTING stby_dat	Register setting to CPG_SAMPLL5_STBY CPG_SAMPLL2/3_STBY register is not implemented in RZ/A3UL

Table 5.33 spi clock table structure (cpg_spi_clk_table)

Member	Description
int frequency	Frequency setting
uint8_t selector	Source clock selector setting
uint8_t divider	Clock divide rate setting

Table 5.34 PFC 8byte register setting structure (PFC_REG_UINT8)

Member	Description
int flag	PFC ON / OFF setting
uintptr_t reg	Pointer to PFC register
uint8_t val	Setting value to PFC register

Table 5.35 PFC 32byte register setting structure (PFC_REG_UINT32)

Member	Description
int flag	PFC ON / OFF setting
uintptr_t reg	Pointer to PFC register
uint32_t val	Setting value to PFC register

Table 5.36 PFC 64byte register setting structure (PFC_REG_UINT64)

Member	Description
int flag	PFC ON / OFF setting
uintptr_t reg	Pointer to PFC register
uint64_t val	Setting value to PFC register

Table 5.37 PFC register setting structure (PFC_REGS)

Member	Description
PFC_REG_UINT8 pmc	Setting to PMC register
PFC_REG_UINT32 pfc	Setting to PFC register
PFC_REG_UINT64 iolh	Setting to IOLH register
PFC_REG_UINT64 pupd	Setting to PUPD register
PFC_REG_UINT64 sr	Setting to SR register
PFC_REG_UINT64 ien	Setting to IEN register

Table 5.38 MMU page table setting structure (rza_mmu_pgtbl_cfg_t)

Member	Description
uint64_t vaddress	Virtual address (0x200000 aligned address)
uint64_t paddress	Physical address (0x200000 aligned address)
uint64_t size	Page size (0x200000 aligned address)
uint64_t attribute	Memory attribute

5.11 Variables

Table 5.39 lists the variables used in IPL.
Table 5.39 lists only structures/unions used in the RZ/A3UL platform boot process.

Table 5.39 List of structures used in the IPL (1/2)

variable name	Description	Detail
xspi_api_t octa_api	xSPI control driver API function table	Refer to Table 5.41
xspidevice_api_t octaflash_mx66uw_api	MX66UW1G45G driver API function table	Refer to Table 5.42
xspidevice_api_t octaram_apsxx_api	APS51208N driver API function table	Refer to Table 5.43
xspi_op_t op_write_enable_spi	Write Enable operation table for SPI (H'06)	Refer to Table 5.44
xspi_op_t op_write_enable_dopi	Write Enable operation table for DOPI (H'06F9)	Refer to Table 5.45
xspi_op_t op_read_status_spi	Read Status Register operation table for SPI (H'05)	Refer to Table 5.46
xspi_op_t op_read_status_dopi	Read Status Register operation table for DOPI (H'05FA)	Refer to Table 5.47
xspi_op_t op_write_config2_spi	Write Config Register-2 operation table for SPI (H'72)	Refer to Table 5.48
xspi_op_t rop	8 I/O Read operation table for DOPI (H'EE11)	Refer to Table 5.49
xspi_op_t op_page_program_dopi	Program 4B operation table for DOPI (H'12ED)	Refer to Table 5.50
xspi_op_t op_erase_4k_dopi	Erase 4k operation table for DOPI (H'21DE)	Refer to Table 5.51
xspi_op_t op_erase_64k_dopi	Erase 64k operation table for DOPI (H'DC23)	Refer to Table 5.52
xspi_op_t op_write_mode	Write Mode operation table (H'4000)	Refer to Table 5.53
xspi_op_t op_rop	Sync Read operation table (H'0x8000)	Refer to Table 5.54
xspi_op_t op_wop	Sync Write operation table (H'0x0000)	Refer to Table 5.55
xspi_op_t op_write_mode	Mode Register Write operation table (H'0x4000)	Refer to Table 5.56
CPG_PLL_SETDATA_146 cpg_pll4_setdata	PLL4 operation setting parameters	Refer to Table 5.57
CPG_PLL_SETDATA_146 cpg_pll6_setdata	PLL6 operation setting parameters	Refer to Table 5.58
CPG_SETUP_DATA early_setup_tbl[]	CPG setting table for modules used for IPL initialization	Refer to Table 5.59
CPG_SETUP_DATA cpg_clk_on_tbl[]	CPG setting table for clock supplying module	Refer to Table 5.60, Table 5.61, Table 5.62
CPG_SETUP_DATA cpg_reset_tbl[]	CPG setting table for modules to be released from reset	Refer to Table 5.63, Table 5.64, Table 5.65
CPG_REG_SETTING cpg_static_select_tbl[]	Clock selector register table that stops clock supply when register settings are changed	Refer to Table 5.66
CPG_REG_SETTING cpg_dynamic_select_tbl[]	Table of clock selector registers whose settings can be changed while the clock is being supplied	Refer to Table 5.67

Table 5.40 List of structures used in the IPL (2/2)

variable name	Description	Detail
PFC_REGS pfc_mux_reg_tbl[]	Pin setting table for modules used in boot processing	Refer to Table 5.67
PFC_REGS pfc_qspi_reg_tbl[]	Serial flash memory pin setting table	Refer to Table 5.68, Table 5.69
rza_mmu_pgtbl_cfg_t g_mmu_pagetable_array[]	MMU setting table	Refer to Table 5.70, Table 5.71

Table 5.41 xSPI control driver API function table (octa_api)

Member	Setting value	Description
open	octa_open	xSPI control driver open function
close	octa_close	xSPI control driver close function
post_init	octa_post_init	xSPI control driver post initialization function
exec_op	octa_exec_op	xSPI control driver operation function
configure_xip	octa_configure_xip	xSPI control driver Memory-map Mode setting function
start_xip	octa_start_xip	xSPI control driver Memory-map Mode start function
stop_xip	octa_stop_xip	xSPI control driver Memory-map Mode stop function
run_manual_calibration	octa_run_manual_calibration	xSPI control driver manual calibration function
enable_auto_calibration	octa_enable_auto_calibration	xSPI control driver auto calibration enable function
disable_auto_calibration	octa_disable_auto_calibration	xSPI control driver auto calibration disable function
set_frequency	octa_set_frequency	xSPI control driver operation frequency setting function
clean_mmap	octa_clean_mmap	xSPI control driver cache clean function
inv_mmap	octa_inv_mmap	xSPI control driver cache invalid function
get_mmap_base	octa_get_mmap_base	xSPI control driver memory map region address get function
get_mmap_size	octa_get_mmap_size	xSPI control driver memory map region size get function

Table 5.42 MX66UW1G45G driver API function table (octaflash_mx66uw_api)

Member	Setting value	Description
open	flash_open	MX66UW1G45G open function
close	flash_close	MX66UW1G45G close function
exec_op	flash_exec_op	MX66UW1G45G operation function
enter_xip	flash_enter_xip	MX66UW1G45G Memory-map Mode start function
exit_xip	flash_exit_xip	MX66UW1G45G Memory-map Mode stop function
get_info	flash_get_info	MX66UW1G45G information get function
read	flash_read	MX66UW1G45G read function
write	flash_write	MX66UW1G45G write function
erase	flash_erase	MX66UW1G45G erase function
get_write_status	flash_get_write_status	MX66UW1G45G write status get function

Table 5.43 APS51208N driver API function table (octaram_apsxx_api)

Member	Setting value	Description
open	octaram_open	APS51208N open function
close	octaram_close	APS51208N close function
exec_op	octaram_exec_op	APS51208N operation function
enter_xip	octaram_enter_xip	APS51208N Memory-map Mode start function
exit_xip	octaram_exit_xip	APS51208N Memory-map Mode stop function
get_info	octaram_get_info	APS51208N information get function
read	octaram_read	APS51208N read function
write	octaram_write	APS51208N write function
erase	octaram_erase	APS51208N erase function
get_write_status	octaram_get_write_status	APS51208N write status get function

Table 5.44 Operation table for SPI: "Write Enable" (op_write_enable_spi)

Member	Setting value	Description
form	SPI_FORM_1_1_1	Command bit width: 1bit Address bit width: 1bit Data bit width: 1bit
op	0x06	Command code for SPI mode "Write Enable"
op_size	1	Command size
op_is_ddr	false	Command transfer method (SDR)
address	0	Address
address_is_ddr	false	Address transfer method (SDR)
address_size	0	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.45 Operation table for DOPI: "Write Enable" (op_write_enable_dopi)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x06F9	Command code for DOPI mode "Write Enable "
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	0	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.46 Operation table for SPI: "Read Status Register" (op_read_status_spi)

Member	Setting value	Description
form	SPI_FORM_1_1_1	Command bit width: 1bit Address bit width: 1bit Data bit width: 1bit
op	0x05	Command code for SPI mode "Read Status Register"
op_size	1	Command size
op_is_ddr	false	Command transfer method (SDR)
address	0	Address
address_is_ddr	false	Address transfer method (SDR)
address_size	0	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer This parameter is updated to the address of the buffer from which the register value is read when communicating with Octa Flash memory
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.47 Operation table for DOPI: "Read Status Register" (op_read_status_dopi)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x05FA	Command code for SPI mode "Read Status Register"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	0	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer This parameter is updated to the address of the buffer from which the register value is read when communicating with Octa Flash memory
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.48 Operation table for SPI: "Write Config Register-2" (op_write_config2_spi)

Member	Setting value	Description
form	SPI_FORM_1_1_1	Command bit width: 1bit Address bit width: 1bit Data bit width: 1bit
op	0x72	Command code for SPI mode "Write Config Register2"
op_size	1	Command size
op_is_ddr	false	Command transfer method (SDR)
address	0	Address
address_is_ddr	false	Address transfer method (SDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer This parameter is updated to the address of the buffer that stores the set value to the register when communicating with Octa Flash memory
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	1	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.49 Operation table for DOPI: "8 I/O Read" (rop)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0xEE11	Command code for DOPI mode "8 I/O Read"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	10	Number of dummy cycles
transfer_buffer	NULL	Data buffer This parameter is not used in IPL because data is not read by serial communication
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.50 Operation table for DOPI: "Program 4B" (op_page_program_dopi)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x12ED	Command code for DOPI mode "Program 4B"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer This parameter is updated to the address of the buffer that stores the write data to the memory when communicating with Octa Flash memory
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.51 Operation table for DOPI: "Erase 4K" (op_erase_4k_dopi)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x21DE	Command code for DOPI mode "Sector Erase 4KB"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.52 Operation table for DOPI: "Erase 64K" (op_erase_64k_dopi)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0xDC23	Command code for DOPI mode "Sector erase 64KB"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.53 Operation table: "Write Mode" (op_write_mode)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x4000	Command code "Write Mode"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0x00040000	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	2	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	0	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.54 Operation table: "Sync Read" (rop)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x8000	Command code "Sync Read"
op_size	2	Command size
op_is_ddr	True	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	4	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	5	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.55 Operation table: "Sync Write" (wop)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x0000	Command code "Sync Write"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	4	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	0	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.56 Operation table: "Write Mode Register" (op_write_mode)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Command bit width: 8bit Address bit width: 8bit Data bit width: 8bit
op	0x4000	Command code "Write Mode Register"
op_size	2	Command size
op_is_ddr	true	Command transfer method (DDR)
address	0	Address
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	0	Number of dummy cycles
transfer_buffer	NULL	Data buffer
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	2	Data size
slch_value	0	Clock Delay DCSTR register DVSELLO bit value
clsh_value	0	SSL Negation Delay DCSTR register DVSELHI bit value
shsl_value	1	Next Access Delay DCSTR register DVSELCMD bit value

Table 5.57 PLL4 operation setting parameters cpg_pll4_setdata)

Member	Setting value	Description
clk1_dat.reg	CPG_PLL4_CLK1	CPG_PLL4_CLK1 register
clk1_dat.val	0xFAE13203	PLL4 = 1600MHz setting Please refer to chapter "PLLn (SSCG) Output Clock Setting Register 1 (CPG_SAMPLLn_CLK2) (n = 1, 4, or 6)" of A3UL Group User's Manual: Hardware for details
clk1_dat.reg	CPG_PLL4_CLK2	CPG_PLL4_CLK2 register
clk1_dat.val	0x00081000	PLL4 = 1600MHz setting Please refer to chapter "PLLn (SSCG) Output Clock Setting Register 2 (CPG_SAMPLLn_CLK2) (n = 1, 4, or 6)" of A3UL Group User's Manual: Hardware for details
stby_dat.reg	CPG_PLL4_STBY	CPG_PLL4_STBY register
stby_dat.val	0x00010001	PLL4 active setting

Table 5.58 PLL6 operation setting parameters (cpg_pll6_setdata)

Member	Setting value	Description
clk1_dat.reg	CPG_PLL6_CLK1	CPG_PLL6_CLK1 register
clk1_dat.val	0x00003Eb3	PLL4 = 500MHz setting Please refer to chapter "PLLn (SSCG) Output Clock Setting Register 1 (CPG_SAMPLLn_CLK2) (n = 1, 4, or 6)" of A3UL Group User's Manual: Hardware for details
clk1_dat.reg	CPG_PLL6_CLK2	CPG_PLL6_CLK2 register
clk1_dat.val	0x00082D02	PLL4 = 500MHz setting Please refer to chapter "PLLn (SSCG) Output Clock Setting Register 2 (CPG_SAMPLLn_CLK2) (n = 1, 4, or 6)" of A3UL Group User's Manual: Hardware for details
stby_dat.reg	CPG_PLL6_STBY	CPG_PLL6_STBY register
stby_dat.val	0x00010001	PLL6 active setting

Table 5.59 CPG setting table for modules used for IPL initialization (early_setup_tbl)

Member	Setting value	Description
early_setup_tbl[0].reg	CPG_CLKON_SYC	CPG_CLKON_SYC register
early_setup_tbl[0].mon	CPG_CLKMON_SYC	CPG_CLKMON_SYC register
early_setup_tbl[0].val	0x00010001	Clock supply setting
early_setup_tbl[0].type	CPG_T_CLK	Set the clock control register
early_setup_tbl[1].reg	CPG_RST_SYC	CPG_RST_SYC register
early_setup_tbl[1].mon	CPG_RSTMON_SYC	CPG_RSTMON_SYC register
early_setup_tbl[1].val	0x00010001	Reset release setting
early_setup_tbl[1].type	CPG_T_RST	Set the release control register

Table 5.60 (cpg_clk_on_tbl) (1/3)

Member	Setting value	Description
cpg_clk_on_tbl[0].reg	CPG_CLKON_CM33	CPG_CLKON_CM33 register
cpg_clk_on_tbl[0].mon	CPG_CLKMON_CM33	CPG_CLKMON_CM33 register
cpg_clk_on_tbl[0].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[0].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[1].reg	CPG_CLKON_MHU	CPG_CLKON_MHU register
cpg_clk_on_tbl[1].mon	CPG_CLKMON_MHU	CPG_CLKMON_MHU register
cpg_clk_on_tbl[1].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[1].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[2].reg	CPG_CLKON_DMAC_REG	CPG_CLKON_DMAC_REG register
cpg_clk_on_tbl[2].mon	CPG_CLKMON_DMAC_REG	CPG_CLKMON_DMAC_REG register
cpg_clk_on_tbl[2].val	0x00030003	Clock supply setting
cpg_clk_on_tbl[2].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[3].reg	CPG_CLKON_OSTM	CPG_CLKON_OSTM register
cpg_clk_on_tbl[3].mon	CPG_CLKMON_OSTM	CPG_CLKMON_OSTM register
cpg_clk_on_tbl[3].val	0x00060000	Clock supply setting
cpg_clk_on_tbl[3].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[4].reg	CPG_CLKON_MTU	CPG_CLKON_MTU register
cpg_clk_on_tbl[4].mon	CPG_CLKMON_MTU	CPG_CLKMON_MTU register
cpg_clk_on_tbl[4].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[4].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[5].reg	CPG_CLKON_POE3	CPG_CLKON_POE3 register
cpg_clk_on_tbl[5].mon	CPG_CLKMON_POE3	CPG_CLKMON_POE3 register
cpg_clk_on_tbl[5].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[5].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[6].reg	CPG_CLKON_WDT	CPG_CLKON_WDT register
cpg_clk_on_tbl[6].mon	CPG_CLKMON_WDT	CPG_CLKMON_WDT register
cpg_clk_on_tbl[6].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[6].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[7].reg	CPG_CLKON_DDR	CPG_CLKON_DDR register
cpg_clk_on_tbl[7].mon	CPG_CLKMON_DDR	CPG_CLKMON_DDR register
cpg_clk_on_tbl[7].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[7].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[8].reg	CPG_CLKON_SDHI	CPG_CLKON_SDHI register
cpg_clk_on_tbl[8].mon	CPG_CLKMON_SDHI	CPG_CLKMON_SDHI register
cpg_clk_on_tbl[8].val	0x00FF00FF	Clock supply setting
cpg_clk_on_tbl[8].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[9].reg	CPG_CLKON_ISU	CPG_CLKON_ISU register
cpg_clk_on_tbl[9].mon	CPG_CLKMON_ISU	CPG_CLKMON_ISU register
cpg_clk_on_tbl[9].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[9].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[10].reg	CPG_CLKON_CRU	CPG_CLKON_CRU register
cpg_clk_on_tbl[10].mon	CPG_CLKMON_CRU	CPG_CLKMON_CRU register
cpg_clk_on_tbl[10].val	0x001F0000	Clock supply setting
cpg_clk_on_tbl[10].type	CPG_T_CLK	Set the clock control register

Table 5.61 (cpg_clk_on_tbl) (2/3)

Member	Setting value	Description
cpg_clk_on_tbl[11].reg	CPG_CLKON_LCDC	CPG_CLKON_LCDC register
cpg_clk_on_tbl[11].mon	CPG_CLKMON_LCDC	CPG_CLKMON_LCDC register
cpg_clk_on_tbl[11].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[11].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[12].reg	CPG_CLKON_SSI	CPG_CLKON_SSI register
cpg_clk_on_tbl[12].mon	CPG_CLKMON_SSI	CPG_CLKMON_SSI register
cpg_clk_on_tbl[12].val	0x00FF0000	Clock supply setting
cpg_clk_on_tbl[12].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[13].reg	CPG_CLKON_SRC	CPG_CLKON_SRC register
cpg_clk_on_tbl[13].mon	CPG_CLKMON_SRC	CPG_CLKMON_SRC register
cpg_clk_on_tbl[13].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[13].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[14].reg	CPG_CLKON_USB	CPG_CLKON_USB register
cpg_clk_on_tbl[14].mon	CPG_CLKMON_USB	CPG_CLKMON_USB register
cpg_clk_on_tbl[14].val	0x000F0000	Clock supply setting
cpg_clk_on_tbl[14].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[15].reg	CPG_CLKON_ETH	CPG_CLKON_ETH register
cpg_clk_on_tbl[15].mon	CPG_CLKMON_ETH	CPG_CLKMON_ETH register
cpg_clk_on_tbl[15].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[15].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[16].reg	CPG_CLKON_I2C	CPG_CLKON_I2C register
cpg_clk_on_tbl[16].mon	CPG_CLKMON_I2C	CPG_CLKMON_I2C register
cpg_clk_on_tbl[16].val	0x000F0000	Clock supply setting
cpg_clk_on_tbl[16].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[17].reg	CPG_CLKON_SCIF	CPG_CLKON_SCIF register
cpg_clk_on_tbl[17].mon	CPG_CLKMON_SCIF	CPG_CLKMON_SCIF register
cpg_clk_on_tbl[17].val	0x001F0001	Clock supply setting
cpg_clk_on_tbl[17].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[18].reg	CPG_CLKON_SCI	CPG_CLKON_SCI register
cpg_clk_on_tbl[18].mon	CPG_CLKMON_SCI	CPG_CLKMON_SCI register
cpg_clk_on_tbl[18].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[18].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[19].reg	CPG_CLKON_IRDA	CPG_CLKON_IRDA register
cpg_clk_on_tbl[19].mon	CPG_CLKMON_IRDA	CPG_CLKMON_IRDA register
cpg_clk_on_tbl[19].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[19].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[20].reg	CPG_CLKON_RSPI	CPG_CLKON_RSPI register
cpg_clk_on_tbl[20].mon	CPG_CLKMON_RSPI	CPG_CLKMON_RSPI register
cpg_clk_on_tbl[20].val	0x00070000	Clock supply setting
cpg_clk_on_tbl[20].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[21].reg	CPG_CLKON_CANFD	CPG_CLKON_CANFD register
cpg_clk_on_tbl[21].mon	CPG_CLKMON_CANFD	CPG_CLKMON_CANFD register
cpg_clk_on_tbl[21].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[21].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[22].reg	CPG_CLKON_GPIO	CPG_CLKON_GPIO register
cpg_clk_on_tbl[22].mon	CPG_CLKMON_GPIO	CPG_CLKMON_GPIO register
cpg_clk_on_tbl[22].val	0x00010001	Clock supply setting
cpg_clk_on_tbl[22].type	CPG_T_CLK	Set the clock control register

Table 5.62 (cpg_clk_on_tbl) (3/3)

Member	Setting value	Description
cpg_clk_on_tbl[23].reg	CPG_CLKON_ADC	CPG_CLKON_ADC register
cpg_clk_on_tbl[23].mon	CPG_CLKMON_ADC	CPG_CLKMON_ADC register
cpg_clk_on_tbl[23].val	0x00030000	Clock supply setting
cpg_clk_on_tbl[23].type	CPG_T_CLK	Set the clock control register
cpg_clk_on_tbl[24].reg	CPG_CLKON_TSU	CPG_CLKON_TSU register
cpg_clk_on_tbl[24].mon	CPG_CLKMON_TSU	CPG_CLKMON_TSU register
cpg_clk_on_tbl[24].val	0x00010000	Clock supply setting
cpg_clk_on_tbl[24].type	CPG_T_CLK	Set the clock control register

Table 5.63 CPG setting table for modules to be released from reset (cpg_reset_tbl) (1/3)

Member	Setting value	Description
cpg_reset_tbl[0].reg	CPG_RSTON_CM33	CPG_RSTON_CM33 register
cpg_reset_tbl[0].mon	CPG_RSTMON_CM33	CPG_RSTMON_CM33 register
cpg_reset_tbl[0].val	0x00070000	Reset release setting
cpg_reset_tbl[0].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[1].reg	CPG_RSTON_MHU	CPG_RSTON_MHU register
cpg_reset_tbl[1].mon	CPG_RSTMON_MHU	CPG_RSTMON_MHU register
cpg_reset_tbl[1].val	0x00010000	Reset release setting
cpg_reset_tbl[1].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[2].reg	CPG_RSTON_DMAC_REG	CPG_RSTON_DMAC_REG register
cpg_reset_tbl[2].mon	CPG_RSTMON_DMAC_REG	CPG_RSTMON_DMAC_REG register
cpg_reset_tbl[2].val	0x00030003	Reset release setting
cpg_reset_tbl[2].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[3].reg	CPG_RSTON_OSTM	CPG_RSTON_OSTM register
cpg_reset_tbl[3].mon	CPG_RSTMON_OSTM	CPG_RSTMON_OSTM register
cpg_reset_tbl[3].val	0x00060000	Reset release setting
cpg_reset_tbl[3].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[4].reg	CPG_RSTON_MTU	CPG_RSTON_MTU register
cpg_reset_tbl[4].mon	CPG_RSTMON_MTU	CPG_RSTMON_MTU register
cpg_reset_tbl[4].val	0x00010000	Reset release setting
cpg_reset_tbl[4].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[5].reg	CPG_RSTON_POE3	CPG_RSTON_POE3 register
cpg_reset_tbl[5].mon	CPG_RSTMON_POE3	CPG_RSTMON_POE3 register
cpg_reset_tbl[5].val	0x00010000	Reset release setting
cpg_reset_tbl[5].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[6].reg	CPG_RSTON_WDT	CPG_RSTON_WDT register
cpg_reset_tbl[6].mon	CPG_RSTMON_WDT	CPG_RSTMON_WDT register
cpg_reset_tbl[6].val	0x00030000	Reset release setting
cpg_reset_tbl[6].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[7].reg	CPG_RSTON_DDR	CPG_RSTON_DDR register
cpg_reset_tbl[7].mon	CPG_RSTMON_DDR	CPG_RSTMON_DDR register
cpg_reset_tbl[7].val	0x007F0000	Reset release setting
cpg_reset_tbl[7].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[8].reg	CPG_RSTON_SDHI	CPG_RSTON_SDHI register
cpg_reset_tbl[8].mon	CPG_RSTMON_SDHI	CPG_RSTMON_SDHI register
cpg_reset_tbl[8].val	0x00030003	Reset release setting
cpg_reset_tbl[8].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[9].reg	CPG_RSTON_ISU	CPG_RSTON_ISU register
cpg_reset_tbl[9].mon	CPG_RSTMON_ISU	CPG_RSTMON_ISU register
cpg_reset_tbl[9].val	0x00030000	Reset release setting
cpg_reset_tbl[9].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[10].reg	CPG_RSTON_CRU	CPG_RSTON_CRU register
cpg_reset_tbl[10].mon	CPG_RSTMON_CRU	CPG_RSTMON_CRU register
cpg_reset_tbl[10].val	0x00070000	Reset release setting
cpg_reset_tbl[10].type	CPG_T_RST	Set the release control register

Table 5.64 CPG setting table for modules to be released from reset (cpg_reset_tbl) (2/3)

Member	Setting value	Description
cpg_reset_tbl[11].reg	CPG_RSTON_LCDC	CPG_RSTON_LCDC register
cpg_reset_tbl[11].mon	CPG_RSTMON_LCDC	CPG_RSTMON_LCDC register
cpg_reset_tbl[11].val	0x00010000	Reset release setting
cpg_reset_tbl[11].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[12].reg	CPG_RSTON_SSIF	CPG_RSTON_SSIF register
cpg_reset_tbl[12].mon	CPG_RSTMON_SSIF	CPG_RSTMON_SSIF register
cpg_reset_tbl[12].val	0x000F0000	Reset release setting
cpg_reset_tbl[12].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[13].reg	CPG_RSTON_SRC	CPG_RSTON_SRC register
cpg_reset_tbl[13].mon	CPG_RSTMON_SRC	CPG_RSTMON_SRC register
cpg_reset_tbl[13].val	0x00010000	Reset release setting
cpg_reset_tbl[13].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[14].reg	CPG_RSTON_USB	CPG_RSTON_USB register
cpg_reset_tbl[14].mon	CPG_RSTMON_USB	CPG_RSTMON_USB register
cpg_reset_tbl[14].val	0x000F0000	Reset release setting
cpg_reset_tbl[14].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[15].reg	CPG_RSTON_ETH	CPG_RSTON_ETH register
cpg_reset_tbl[15].mon	CPG_RSTMON_ETH	CPG_RSTMON_ETH register
cpg_reset_tbl[15].val	0x00030000	Reset release setting
cpg_reset_tbl[15].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[16].reg	CPG_RSTON_I2C	CPG_RSTON_I2C register
cpg_reset_tbl[16].mon	CPG_RSTMON_I2C	CPG_RSTMON_I2C register
cpg_reset_tbl[16].val	0x000F0000	Reset release setting
cpg_reset_tbl[16].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[17].reg	CPG_RSTON_SCIF	CPG_RSTON_SCIF register
cpg_reset_tbl[17].mon	CPG_RSTMON_SCIF	CPG_RSTMON_SCIF register
cpg_reset_tbl[17].val	0x001F0001	Reset release setting
cpg_reset_tbl[17].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[18].reg	CPG_RSTON_SCI	CPG_RSTON_SCI register
cpg_reset_tbl[18].mon	CPG_RSTMON_SCI	CPG_RSTMON_SCI register
cpg_reset_tbl[18].val	0x00030000	Reset release setting
cpg_reset_tbl[18].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[19].reg	CPG_RSTON_IRDA	CPG_RSTON_IRDA register
cpg_reset_tbl[19].mon	CPG_RSTMON_IRDA	CPG_RSTMON_IRDA register
cpg_reset_tbl[19].val	0x00010000	Reset release setting
cpg_reset_tbl[19].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[20].reg	CPG_RSTON_RSPI	CPG_RSTON_RSPI register
cpg_reset_tbl[20].mon	CPG_RSTMON_RSPI	CPG_RSTMON_RSPI register
cpg_reset_tbl[20].val	0x00070000	Reset release setting
cpg_reset_tbl[20].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[21].reg	CPG_RSTON_CANFD	CPG_RSTON_CANFD register
cpg_reset_tbl[21].mon	CPG_RSTMON_CANFD	CPG_RSTMON_CANFD register
cpg_reset_tbl[21].val	0x00030000	Reset release setting
cpg_reset_tbl[21].type	CPG_T_RST	Set the release control register

Table 5.65 CPG setting table for modules to be released from reset (cpg_reset_tbl) (3/3)

Member	Setting value	Description
cpg_reset_tbl[22].reg	CPG_RSTON_GPIO	CPG_RSTON_GPIO register
cpg_reset_tbl[22].mon	CPG_RSTMON_GPIO	CPG_RSTMON_GPIO register
cpg_reset_tbl[22].val	0x00070007	Reset release setting
cpg_reset_tbl[22].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[23].reg	CPG_RSTON_ADC	CPG_RSTON_ADC register
cpg_reset_tbl[23].mon	CPG_RSTMON_ADC	CPG_RSTMON_ADC register
cpg_reset_tbl[23].val	0x00030000	Reset release setting
cpg_reset_tbl[23].type	CPG_T_RST	Set the release control register
cpg_reset_tbl[24].reg	CPG_RSTON_TSU	CPG_RSTON_TSU register
cpg_reset_tbl[24].mon	CPG_RSTMON_TSU	CPG_RSTMON_TSU register
cpg_reset_tbl[24].val	0x00030000	Reset release setting
cpg_reset_tbl[24].type	CPG_T_RST	Set the release control register

Table 5.66 Clock selector register table that stops clock supply (cpg_static_select_tbl)

Member	**Setting value**	**Description**
cpg_static_select_tbl[0].reg	CPG_PL3A_DDIV	CPG_PL3A_DDIV register
cpg_static_select_tbl[0].val	0x01000100	PLL3 clock divide rate setting
cpg_static_select_tbl[1].reg	CPG_PL3_SSEL	CPG_PL3_SSEL register
cpg_static_select_tbl[1].val	0x01000000	PLL3 source clock setting

Table 5.67 Clock selector register table that continues clock supply (cpg_dynamic_select_tbl)

Member	**Setting value**	**Description**
cpg_dynamic_select_tbl[0].reg	CPG_PL4_DSEL	CPG_PL4_DSEL register
cpg_dynamic_select_tbl[0].val	0x00010001	PLL4 clock divide rate setting
cpg_dynamic_select_tbl[1].reg	CPG_PL2SDHI_DSEL	CPG_PL2SDHI_DSEL register
cpg_dynamic_select_tbl[1].val	0x00110022	PLL2 source clock setting

Table 5.68 Pin setting table for modules used in boot processing (pfc_mux_reg_tbl)

Member	Setting value	Description
pfc_mux_reg_tbl[0].pmc.flag	PFC_ON	PFC enable flag
pfc_mux_reg_tbl[0].pmc.reg	PFC_PMC10	PFC_PMC10 register
pfc_mux_reg_tbl[0].pmc.val	0x0F	Register setting value
pfc_mux_reg_tbl[0].pfc.flag	PFC_ON	PFC enable flag
pfc_mux_reg_tbl[0].pfc.reg	PFC_PFC10	PFC_PFC10 register
pfc_mux_reg_tbl[0].pfc.val	0x00001111	Register setting value
pfc_mux_reg_tbl[0].iolh.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[0].iolh.reg	PFC_IOLH10	PFC_IOLH10 register
pfc_mux_reg_tbl[0].iolh.val	0x0000000001010101	Register setting value
pfc_mux_reg_tbl[0].pupd.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[0].pupd.reg	PFC_PUPD10	PFC_PUPD10 register
pfc_mux_reg_tbl[0].pupd.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[0].sr.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[0].sr.reg	PFC_SR10	PFC_SR10 register
pfc_mux_reg_tbl[0].sr.val	0x0000000001010101	Register setting value
pfc_mux_reg_tbl[0].ien.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[0].ien.reg	NULL	PFC_IEN register NULL is defined for this member because PFC_IEN10 is not implemented
pfc_mux_reg_tbl[0].ien.val	0	Register setting value
pfc_mux_reg_tbl[1].pmc.flag	PFC_ON	PFC enable flag
pfc_mux_reg_tbl[1].pmc.reg	PFC_PMC16	PFC_PMC16 register
pfc_mux_reg_tbl[1].pmc.val	0x18	Register setting value
pfc_mux_reg_tbl[1].pfc.flag	PFC_ON	PFC enable flag
pfc_mux_reg_tbl[1].pfc.reg	PFC_PFC16	PFC_PFC16 register
pfc_mux_reg_tbl[1].pfc.val	0x00066000	Register setting value
pfc_mux_reg_tbl[1].iolh.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[1].iolh.reg	PFC_IOLH16	PFC_IOLH16 register
pfc_mux_reg_tbl[1].iolh.val	0x0000000101000000	Register setting value
pfc_mux_reg_tbl[1].pupd.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[1].pupd.reg	PFC_PUPD16	PFC_PUPD16 register
pfc_mux_reg_tbl[1].pupd.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[1].sr.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[1].sr.reg	PFC_SR16	PFC_SR16 register
pfc_mux_reg_tbl[1].sr.val	0x0000000101000000	Register setting value
pfc_mux_reg_tbl[1].ien.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[1].ien.reg	NULL	PFC_IEN register NULL is defined for this member because PFC_IEN16 is not implemented
pfc_mux_reg_tbl[1].ien.val	0	Register setting value

Table 5.69 Serial flash memory pin setting table (pfc_qspi_reg_tbl) (1/2)

Member	Setting value	Description
pfc_mux_reg_tbl[0].pmc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[0].pmc.reg	NULL	PFC_PMC register NULL is defined for this member because PFC_PMC0A is not implemented
pfc_mux_reg_tbl[0].pmc.val	0	Register setting value
pfc_mux_reg_tbl[0].pfc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[0].pfc.reg	NULL	PFC_PFC register NULL is defined for this member because PFC_PFC0A is not implemented
pfc_mux_reg_tbl[0].pfc.val	0	Register setting value
pfc_mux_reg_tbl[0].iolh.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[0].iolh.reg	PFC_IOLH0A	PFC_IOLH0A register
pfc_mux_reg_tbl[0].iolh.val	0x0000010101010101	Register setting value
pfc_mux_reg_tbl[0].pupd.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[0].pupd.reg	PFC_PUPD0A	PFC_PUPD0A register
pfc_mux_reg_tbl[0].pupd.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[0].sr.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[0].sr.reg	PFC_SR0A	PFC_SR0A register
pfc_mux_reg_tbl[0].sr.val	0x0000010101010101	Register setting value
pfc_mux_reg_tbl[0].ien.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[0].ien.reg	NULL	PFC_IEN register NULL is defined for this member because PFC_IEN0A is not implemented
pfc_mux_reg_tbl[0].ien.val	0	Register setting value
pfc_mux_reg_tbl[1].pmc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[1].pmc.reg	NULL	PFC_PMC register NULL is defined for this member because PFC_PMC0B is not implemented
pfc_mux_reg_tbl[1].pmc.val	0	Register setting value
pfc_mux_reg_tbl[1].pfc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[1].pfc.reg	NULL	PFC_PFC register NULL is defined for this member because PFC_PFC0B is not implemented
pfc_mux_reg_tbl[1].pfc.val	0	Register setting value
pfc_mux_reg_tbl[1].iolh.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[1].iolh.reg	PFC_IOLH0B	PFC_IOLH0B register
pfc_mux_reg_tbl[1].iolh.val	0x0000010101010101	Register setting value
pfc_mux_reg_tbl[1].pupd.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[1].pupd.reg	PFC_PUPD0B	PFC_PUPD0B register
pfc_mux_reg_tbl[1].pupd.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[1].sr.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[1].sr.reg	PFC_SR0B	PFC_SR0B register
pfc_mux_reg_tbl[1].sr.val	0x0000010101010101	Register setting value
pfc_mux_reg_tbl[1].ien.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[1].ien.flag	PFC_OFF	PFC disable flag

Table 5.70 Serial flash memory pin setting table (pfc_qspi_reg_tbl) (2/2)

Member	Setting value	Description
pfc_mux_reg_tbl[1].ien.reg	NULL	PFC_IEN register NULL is defined for this member because PFC_IEN0B is not implemented
pfc_mux_reg_tbl[1].ien.val	0	Register setting value
pfc_mux_reg_tbl[2].pmc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[2].pmc.val	0	Register setting value
pfc_mux_reg_tbl[2].pfc.flag	PFC_OFF	PFC enable flag
pfc_mux_reg_tbl[2].pfc.reg	NULL	PFC_PFC register NULL is defined for this member because PFC_PFC0C is not implemented
pfc_mux_reg_tbl[2].pfc.val	0	Register setting value
pfc_mux_reg_tbl[2].iolh.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[2].iolh.reg	PFC_IOLH0C	PFC_IOLH0C register
pfc_mux_reg_tbl[2].iolh.val	0x0000000000000101	Register setting value
pfc_mux_reg_tbl[2].pupd.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[2].pupd.reg	PFC_PUPD0C	PFC_PUPD0C register
pfc_mux_reg_tbl[2].pupd.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[2].sr.flag	PFC_ON	PFC disable flag
pfc_mux_reg_tbl[2].sr.reg	PFC_SR0C	PFC_SR0C register
pfc_mux_reg_tbl[2].sr.val	0x0000000000000000	Register setting value
pfc_mux_reg_tbl[2].ien.flag	PFC_OFF	PFC disable flag
pfc_mux_reg_tbl[2].ien.reg	NULL	PFC_IEN register NULL is defined for this member because PFC_IEN0C is not implemented
pfc_mux_reg_tbl[2].ien.val	0	Register setting value

Table 5.71 MMU setting table (g_mmu_pagetable_array[]) (1/2)

Member	Setting value	Description
g_mmu_pagetable_array[0].vaddress	0x00000000	Virtual address setting
g_mmu_pagetable_array[0].paddess	0x00000000	Physical address setting
g_mmu_pagetable_array[0].size	0x00200000	Virtual memory size
g_mmu_pagetable_array[0].attribute	RZA_MMU_ATTRIBUTE_NORMAL_CACHEABLE	Memory attribute setting: Normal Memory、Cache Enable (Write back)
g_mmu_pagetable_array[1].vaddress	0x00200000	Virtual address setting
g_mmu_pagetable_array[1].paddess	0x00200000	Physical address setting
g_mmu_pagetable_array[1].size	0x0FE00000	Virtual memory size
g_mmu_pagetable_array[1].attribute	RZA_MMU_ATTRIBUTE_ACCESS_FAULT	Memory attribute setting: Access Fault
g_mmu_pagetable_array[2].vaddress	0x10000000	Virtual address setting
g_mmu_pagetable_array[2].paddess	0x10000000	Physical address setting
g_mmu_pagetable_array[2].size	0x10000000	Virtual memory size
g_mmu_pagetable_array[2].attribute	RZA_MMU_ATTRIBUTE_DEVICE	Memory attribute setting: Device Memory
g_mmu_pagetable_array[3].vaddress	0x20000000	Virtual address setting
g_mmu_pagetable_array[3].paddess	0x20000000	Physical address setting
g_mmu_pagetable_array[3].size	0x10000000	Virtual memory size
g_mmu_pagetable_array[3].attribute	RZA_MMU_ATTRIBUTE_NORMAL_CACHEABLE	Memory attribute setting: Normal Memory、Cache Enable (Write back)
g_mmu_pagetable_array[4].vaddress	0x30000000	Virtual address setting
g_mmu_pagetable_array[4].paddess	0x30000000	Physical address setting
g_mmu_pagetable_array[4].size	0x10000000	Virtual memory size
g_mmu_pagetable_array[4].attribute	RZA_MMU_ATTRIBUTE_ACCESS_FAULT	Memory attribute setting: Access Fault
g_mmu_pagetable_array[5].vaddress	0x40000000	Virtual address setting
g_mmu_pagetable_array[5].paddess	0x40000000	Physical address setting
g_mmu_pagetable_array[5].size	0x40000000	Virtual memory size
g_mmu_pagetable_array[5].attribute	RZA_MMU_ATTRIBUTE_NORMAL_CACHEABLE	Memory attribute setting: Normal Memory、Cache Enable (Write back)

Table 5.72 MMU setting table (g_mmu_pagetable_array[]) (2/2)

g_mmu_pagetable_array[6].vaddress	0x80000000	Virtual address setting
g_mmu_pagetable_array[6].paddess	0x80000000	Physical address setting
g_mmu_pagetable_array[6].size	0x80000000	Virtual memory size
g_mmu_pagetable_array[6].attribute	RZA_MMU_ATTRIBUTE_ACCESS_FAULT	Memory attribute setting: Access Fault
g_mmu_pagetable_array[7].vaddress	0xC0000000	Virtual address setting
g_mmu_pagetable_array[7].paddess	0xC0000000	Physical address setting
g_mmu_pagetable_array[7].size	0xC0000000	Virtual memory size
g_mmu_pagetable_array[7].attribute	RZA_MMU_ATTRIBUTE_ACCESS_FAULT	Memory attribute setting: Access Fault
g_mmu_pagetable_array[8].vaddress	0xFFFFFFFF	Dummy address
g_mmu_pagetable_array[8].paddess	0xFFFFFFFF	Dummy address
g_mmu_pagetable_array[8].size	0xFFFFFFFF	Dummy size
g_mmu_pagetable_array[8].attribute	RZA_MMU_ATTRIBUTE_CONFIG_END	End of MMU configure table

5.12 Function

This chapter shows the functions implemented in IPL.

This chapter only describes the functions used in the RZ/A3UL platform boot process.

5.12.1 Function List

Table 5.73 - Table 5.76 list the functions used in the RZ/A3UL platform boot process.

Table 5.73 List of IPL control functions

Function name	Description
bl2_el3_early_platform_setup	Initial settings for Renesas platforms
bl2_el3_plat_arch_setup	MMU settings for Renesas platforms
bl2_platform_setup	Memory settings for Renesas platforms
rz_io_setup	Initialize the FSP application image storage destination memory
panic	Jump to panic handler

Table 5.74 List of Octa Memory Controller driver functions

Function name	Description
xspi_setup	Initialize Octa Flash memory/ Octa RAM
octa_open	Initialize Octa Memory Controller driver
octa_close	Close Octa Memory Controller driver
octa_post_init	Post-processing of Octa Memory Controller driver initialization
octa_exec_op	Execute operation to Octa Flash memory/ Octa RAM
octa_configure_xip	Octa Flash memory/ Octa RAM Memory-map Mode setting
octa_start_xip	Octa Flash memory/ Octa RAM Memory-map Mode start
octa_stop_xip	Octa Flash memory/ Octa RAM Memory-map Mode finish
octa_run_manual_calibration	Execute manual calibration for Octa Flash memory/ Octa RAM
octa_enable_auto_calibration	Enable auto calibration for Octa Flash memory/ Octa RAM
octa_disable_auto_calibration	Disable auto calibration for Octa Flash memory/ Octa RAM
octa_set_frequency	Sets the maximum operating frequency of the Octa Flash memory/ Octa RAM
octa_clean_mmap	Clean read cache of Octa Flash memory/ Octa RAM
octa_inv_mmap	Invalidate read cache of Octa Flash memory/ Octa RAM
octa_get_mmap_base	Get base address where Octa Flash memory/ Octa RAM is mapped
octa_get_mmap_size	Get the size of the region where the Octa Flash memory/ Octa RAM is mapped
octa_get_features	Get the operation format supported by the Octa Memory Controller driver

Table 5.75 List of Octa Flash memory / Octa RAM driver functions

Function name	Description
flash_open	Initialize the Octa Flash memory
flash_close	Close setting to the Octa Flash memory
flash_exec_op	Execute operation to Octa Flash memory
flash_enter_xip	External Address Space Read Mode start
flash_exit_xip	External Address Space Read Mode finish
flash_get_info	Get the Octa Flash memory information
flash_read	Data read from the Octa Flash memory
flash_write	Data write to the Octa Flash memory
flash_erase	Data erase of Octa Flash memory
flash_get_write_status	Get write status of the Octa Flash memory
flash_write_enable_spi	Set write enable to Octa Flash memory in SPI
flash_write_enable_dopi	Set write enable to Octa Flash memory in DOPI
flash_read_status_register_spi	Read the status register of Octa Flash memory in SPI
flash_read_status_register_dopi	Read the status register of Octa Flash memory in DOPI
flash_write_config_register2_spi	Write value to the configuration register2 of Octa Flash memory in SPI
flash_erase_4k	Erase Octa Flash memory in units of 4Kbytes
flash_erase_64k	Erase Octa Flash memory in units of 64Kbytes
octaram_open	Initialize the Octa RAM
octaram_close	Close setting to the Octa RAM
octaram_exec_op	Execute operation to Octa RAM
octaram_enter_xip	External Address Space Read Mode start
octaram_exit_xip	External Address Space Read Mode finish
octaram_get_info	Get the Octa RAM information
octaram_read	Data read from the Octa RAM
octaram_write	Data write to the Octa RAM
octaram_erase	Data erase of Octa RAM
octaram_get_write_status	Get write status of the Octa RAM
octaram_write_mode_register	Write to the mode register of Octa RAM

Table 5.76 List of module control functions

Function name	Description
syc_init	Initialize the Cortex®-A55 built-in Generic Timer
generic_delay_timer_init	Initialize timer management parameters
pfc_setup	Set up pin
pfc_mux_setup	Set up pin of SCIF
pfc_qspi_setup	Set up pin of SPIBSC
cpg_setup	Set up CPG
cpg_ctrl_clkrst	Write processing to the clock control register and reset control register
cpg_selector_on_off	Clock ON/OFF setting for modules whose clocks are managed by the source clock selector
cpg_pll_start_146	Start process of PLL1/PLL4/PLL6
cpg_pll_setup	Set up PLL
cpg_div_sel_setup	Write process to Division Ratio Setting register and Source Clock Setting register
cpg_div_sel_static_setup	Setting of Static Switching Frequency Dividers and Selector
cpg_div_sel_dynamic_setup	Setting of Dynamic Switching Frequency Dividers and Selector
cpg_clk_on_setup	Clock supply setting for each module
cpg_reset_setup	Reset release setting for each module
cpg_active_ddr	DDR PHY clock and reset release settings
cpg_reset_ddr_mc	DDR Memory Controller clock and reset release setting
cpg_early_setup	Clock supply and reset release to modules required for pre-processing of platform boot process
cpg_wdtrst_sel_setup	Watchdog Timer module reset release processing
cpu_cpg_setup	Clock division ratio setting for CPU
cpg_search_spi_clk_table	Search the register setting table corresponding to the clock frequency supplied to the serial flash memory
cpg_research_spi_clk_table	Search for clock frequency value supplied to serial flash memory
cpg_stop_xspi_clock	Clock stop processing for serial flash memory
cpg_start_xspi_clock	Clock supply processing to serial flash memory
cpg_set_xspi_clock	Serial flash memory clock frequency setting processing
cpg_get_xspi_clock	Get the clock frequency supplied to the serial flash memory
wait_until_32	Processing to wait for the end of clock setting
console_rzg2l_register	Console output setting
console_set_scope	Set enable state for console output
plat_get_bl_image_load_info	Get load information of the FSP Application image
plat_get_next_bl_params	Get parameter information of the FSP Application image
rz_check_fsp_header	Check the Application Header
rz_update_descs	Update descriptor parameters
rz_print_descs	Print descriptor parameters
ddr_setup	Initial setting of DDR SDRAM

5.12.2 Function Specifications

This chapter shows the specification of functions.

5.12.2.1 IPL control functions

The functions related to IPL control are listed below.

bl2_el3_early_platform_setup	
Outline	Perform initial settings for Renesas platform
Declaration	void bl2_el3_early_platform_setup (u_register_t arg1, u_register_t arg2, u_register_t arg3, u_register_t arg4)
Description	Initial settings related to hardware such as clocks, pins, general-purpose timers, console output, etc.
Arguments	arg1 : no use it arg2 : no use it arg3 : no use it arg4 : no use it
Return value	None

bl2_el3_plat_arch_setup	
Outline	Set MMU for Renesas platforms
Declaration	void bl2_el3_plat_arch_setup (void)
Description	Set MMU for Renesas platforms.
Arguments	None
Return value	None

bl2_platform_setup	
Outline	Performs memory settings for Renesas platforms
Declaration	void bl2_platform_setup (void)
Description	Initialize memory for Renesas platform and copy FSP application image.
Arguments	None
Return value	None

rz_io_setup	
Outline	Initialize the FSP application image storage destination memory
Declaration	void rz_io_setup (void)
Description	Initializes the serial flash memory where the FSP application image is stored. Additionally, this function checks the boot mode. If the boot mode is set to something other than boot mode 3, the IPL will start an infinite loop and stop the boot process.
Arguments	None
Return value	None

panic	
panic	Jump to panic handler
Outline	void panic (void)
Declaration	When an error occurs, jump to panic handler which stops processing by infinite loop. This function is called when an unrecoverable error occurs.
Description	None
Arguments	None

5.12.2.2 Octa Memory Controller driver functions

The Octa memory controller driver functions that control the Octa Memory Controller are shown below.

xspi_setup	
Outline	Initialize the Octa Flash memory / Octa RAM
Declaration	void xspi_setup (void)
Description	This function calls the API function corresponding to the Octa Flash memory/ Octa RAM and initializes the memory.
Arguments	None
Return value	None

octa_open	
Outline	Initialize the Octa Memory Controller driver
Declaration	int octa_open (xspi_ctrl_t * ctrl, xspi_cfg_t const * const cfg)
Description	Initialize the Octa Memory Controller driver.
Arguments	*ctrl : Pointer to the control structure for instance *cfg : Pointer to the configuration structure for instance
Return value	0 : Initialization is complete -1 : Initialization is failed

octa_close	
Outline	Close the Octa Memory Controller driver
Declaration	int octa_close (xspi_ctrl_t * ctrl)
Description	Close the Octa Memory Controller driver.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : Close process is complete -1 : Close process is failed

octa_post_init	
Outline	Perform post-processing of Octa Memory Controller driver initialization
Declaration	void octa_post_init (xspi_ctrl_t * ctrl)
Description	Perform post-processing of Octa Memory Controller driver initialization. Processing in this function is not implemented in this version.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octa_exec_op	
Outline	Issue the operation to Octa Flash memory / Octa RAM
Declaration	int octa_exec_op (xspi_ctrl_t * const ctrl, xspi_op_t const * const op, bool is_write)
Description	Issue the operation to Octa Flash memory/ Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance *op : Pointer to xSPI interface operation structure is_write : Specifies the transmission/reception direction of the operation(0=read、1=write)
Return value	0 : Issue the operation is complete -1 : Issue the operation is failed

octa_configure_xip	
Outline	Set the Memory-map Mode setting to memory
Declaration	int octa_configure_xip (xspi_ctrl_t * const ctrl, xspi_op_t const * const rop, xspi_op_t const * const wop)
Description	Set the Memory-map Mode setting to memory
Arguments	*ctrl : Pointer to the control structure for instance *rop : Pointer to operation structure for read *wop : Pointer to operation structure for write
Return value	0 : Setting is complete -1 : Setting is failed

octa_start_xip	
Outline	Start the Memory-map Mode to memory
Declaration	int octa_start_xip (xspi_ctrl_t * const ctrl)
Description	Start the Memory-map Mode to memory.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octa_stop_xip	
Outline	Stop the Memory-map Mode to memory
Declaration	int octa_stop_xip (xspi_ctrl_t * const ctrl)
Description	Stop the Memory-map Mode to memory.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octa_run_manual_calibration	
Outline	Execute calibration for Octa Flash memory/ Octa RAM
Declaration	int octa_run_manual_calibration (xspi_ctrl_t * const ctrl)
Description	Execute Calibration for Octa Flash memory/ Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : Calibration is complete -1 : Calibration is failed

octa_enable_auto_calibration	
Outline	Enable auto calibration for Octa Flash memory/ Octa RAM
Declaration	int octa_enable_auto_calibration (xspi_ctrl_t * const ctrl)
Description	Enable auto calibration for Octa Flash memory/ Octa RAM. Processing in this function is not implemented in this version.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	-1 : This value is always returned

octa_disable_auto_calibration	
Outline	Disable auto calibration for Octa Flash memory/ Octa RAM
Declaration	int octa_disable_auto_calibration (xspi_ctrl_t * const ctrl)
Description	Disable auto calibration for Octa Flash memory/ Octa RAM. Processing in this function is not implemented in this version.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	-1 : This value is always returned

octa_set_frequency	
Outline	Sets the maximum operating frequency of the Octa Flash memory/ Octa RAM
Declaration	int octa_set_frequency (xspi_ctrl_t * const ctrl, int frequency_hz)
Description	Sets the maximum operating frequency of the Octa Flash memory/ Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance frequency_hz : Maximum operating frequency of the Octa Flash memory/ Octa RAM
Return value	0 : This value is always returned

octa_clean_mmap	
Outline	Clean read cache of the Octa Flash memory/ Octa RAM
Declaration	int octa_clean_mmap (xspi_ctrl_t * const ctrl)
Description	Clean read cache of the Octa Flash memory/ Octa RAM. Processing in this function is not implemented in this version.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octa_inv_mmap	
Outline	Invalidate read cache of the Octa Flash memory/ Octa RAM
Declaration	int octa_inv_mmap (xspi_ctrl_t * const ctrl)
Description	Invalidate read cache of the Octa Flash memory/ Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octa_get_mmap_base	
Outline	Get the base address where the Octa Flash memory/ Octa RAM is mapped
Declaration	uintptr_t octa_get_mmap_base (xspi_ctrl_t * const ctrl)
Description	Get the base address to which the Octa Flash memory/ Octa RAM is mapped.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	Positive number: Base address where the Octa Flash memory/ Octa RAM is mapped

octa_get_mmap_size	
Outline	Get the size of the area where the Octa Flash memory/ Octa RAM is mapped
Declaration	uintptr_t octa_get_mmap_size (xspi_ctrl_t * const ctrl)
Description	Get the size of the area where the Octa Flash memory/ Octa RAM is mapped.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	Positive number: the size of the area where the Octa Flash memory/ Octa RAM is mapped

octa_get_features	
Outline	Get the operation format supported by the Octa Memory Controller driver
Declaration	uint32_t octa_get_features (xspi_ctrl_t * const ctrl)
Description	Get the operation format supported by the Octa Memory Controller driver
Arguments	*ctrl : Pointer to the control structure for instance
Return value	Positive number: Returns the OR of the following bit values
	XSPI_FEATURE_XIP_READ : The driver supports read access.
	XSPI_FEATURE_XIP_WRITE : The driver supports write access.
	XSPI_FEATURE_FORM112 : The driver supports 1-1-2 form.
	XSPI_FEATURE_FORM122 : The driver supports 1-2-2 form.
	XSPI_FEATURE_FORM222 : The driver supports 2-2-2 form.
	XSPI_FEATURE_DUAL : The driver supports 1-1-2, 1-2-2, and/or 2-2-2 form(s).
	XSPI_FEATURE_FORM114 : The driver supports 1-1-4 form.
	XSPI_FEATURE_FORM144 : The driver supports 1-4-4 form.
	XSPI_FEATURE_FORM444 : The driver supports 4-4-4 form.
	XSPI_FEATURE_QUAD : The driver supports 1-1-4, 1-4-4, and/or 4-4-4 form(s).
	XSPI_FEATURE_FORM118 : The driver supports 1-1-8 form.
	XSPI_FEATURE_FORM188 : The driver supports 1-8-8 form.
	XSPI_FEATURE_FORM888 : The driver supports 8-8-8 form.
	XSPI_FEATURE_OCTAL : The driver supports 1-1-8, 1-8-8, and/or 8-8-8 form(s).
	XSPI_FEATURE_DDROP : The driver supports DDR for op code.
	XSPI_FEATURE_DDR : The driver supports DDR for address and/or data.
	XSPI_FEATURE_OCTARAM : Octa RAM addressing (RAS/CAS) supported.

5.12.2.3 Octa Flash memory / Octa RAM driver functions

This chapter shows the Octa Flash memory driver functions and Octa RAM driver functions for controlling the Octa Flash memory and Octa RAM driver for controlling the Octa RAM.

flash_open	
Outline	Initialize the Octa Flash memory
Declaration	int flash_open (xspidevice_ctrl_t * ctrl, xspidevice_cfg_t const * cfg)
Description	Initialize the Octa Flash memory.
Arguments	*ctrl : Pointer to xSPI device control structure
	*cfg : Pointer to xSPI device configure structure
Return value	0 : Initialization is complete
	-1 : Initialization is failed

flash_close	
Outline	Close the Octa Flash memory
Declaration	int flash_close (xspidevice_ctrl_t * ctrl)
Description	Close the Octa Flash memory.
Arguments	*ctrl : Pointer to xSPI device control structure
Return value	0 : Close process is complete
	-1 : Close process is failed

flash_exec_op	
Outline	Execute operation to Octa Flash memory
Declaration	void flash_exec_op (xspidevice_ctrl_t * ctrl, xspi_op_t const * op, bool is_write)
Description	Execute operation to Octa Flash memory.
Arguments	*ctrl : Pointer to xSPI device control structure
	*op : Pointer to xSPI interface operation structure
	is_write :Specifies the transmission/reception direction of the operation(0=read、1=write)
Return value	0 : Execute the operation is complete
	-1 : Execute the operation is failed

flash_enter_xip	
Outline	Set the Memory-map Mode to the Octa Flash memory
Declaration	int flash_enter_xip (xspidevice_ctrl_t * ctrl)
Description	Set the Memory-map Mode to the Octa Flash memory.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : Setting is complete
	-1 : Setting is failed

flash_exit_xip	
Outline	Finish the Memory-map Mode to Octa Flash memory
Declaration	int flash_exit_xip (xspi_ctrl_t * const ctrl)
Description	Finish the Memory-map Mode to Octa Flash memory.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

flash_get_info	
Outline	Get the information of the Octa Flash memory
Declaration	int flash_get_info (xspidevice_ctrl_t * ctrl, xspidevice_info_t * info)
Description	Get the information of the Octa Flash memory.
Arguments	*ctrl : Pointer to the control structure for instance * info : Information of the Octa Flash memory
Return value	0 : This value is always returned

flash_read	
Outline	Data read from the Octa Flash memory
Declaration	int flash_read (xspi_ctrl_t * const ctrl, void * buffer, size_t address, size_t length)
Description	Data read from the Octa Flash memory,
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer to store read data address: Data read source address Specify the offset address in the Octa Flash memory for this argument length : Read data size
Return value	0 : This value is always returned

flash_write	
Outline	Data write to the Octa Flash memory
Declaration	int flash_write (xspi_ctrl_t * const ctrl, void const * buffer, size_t address, size_t length)
Description	Data write to the Octa Flash memory.
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer where write data is stored address: Data write destination address Specify the offset address in the Octa Flash memory for this argument length : Write data size
Return value	0 : Data write is success -1 : Data write is failed

flash_erase	
Outline	Erase the Octa Flash memory
Declaration	int flash_erase (xspidevice_ctrl_t * ctrl, size_t address, size_t length)
Description	Data erase the Octa Flash memory. Data erase is performed with addresses and sizes in units of 4 Kbytes.。
Arguments	*ctrl : Pointer to the control structure for instance address: Start address for data erase Specify the offset address in the Octa Flash memory for this argument. If a value other than a multiple of 4Kbytes is set, the address obtained by aligning the specified address with 4Kbytes will be the start address of the erase area. length : erase size If a value other than 4Kbyte aligned size is set in this argument, it erases the data up to the 4Kbyte aligned address including the specified range.
Return value	0 : Data erase is success -1 : Data erase is failed.

flash_get_write_status	
Outline	Gets the write status of the Octa Flash memory
Declaration	int flash_get_write_status (xspidevice_ctrl_t * ctrl)
Description	Gets the write status of the Octa Flash memory.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	WRITE_STATUS_FAIL : Data write is failed WRITE_STATUS_IN_PROGRESS : Data write in progress WRITE_STATUS_OK : Data write is complete

flash_write_enable_spi	
Outline	Set write enable to Octa Flash memory in SPI
Declaration	int flash_write_enable_spi (octaflash_mx66uw_ctrl_t *myctrl)
Description	Set write enable to Octa Flash memory in SPI.
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation is success -1 : Operation is failed

flash_write_enable_dopi	
Outline	Set write enable to Octa Flash memory in DOPI
Declaration	int flash_write_enable_dopi (octaflash_mx66uw_ctrl_t *myctrl)
Description	Set write enable to Octa Flash memory in SPI.
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation is success -1 : Operation is failed

flash_read_status_register_spi	
Outline	Read status register of Octa Flash memory in SPI
Declaration	int read_status_register_spi (octaflash_mx66uw_ctrl_t *myctrl)
Description	Read status register of Octa Flash memory in SPI.
Arguments	*myctrl : Pointer to the control structure for instance
Return value	Positive number : Operation success, and status register value -1 : Operation is failed

flash_read_status_register_dopi	
Outline	Read status register of Octa Flash memory in DOPI
Declaration	int read_status_register_dopi (octaflash_mx66uw_ctrl_t *myctrl)
Description	Read status register of Octa Flash memory in DOPI.
Arguments	*myctrl : Pointer to the control structure for instance
Return value	Positive number : Operation success, and status register value -1 : Operation is failed

flash_write_config_register2_spi	
Outline	Write configuration register2 of Octa Flash memory in SPI
Declaration	int flash_write_config_register2_spi(octaflash_mx66uw_ctrl_t *myctrl, uint32_t add, unit8_t data)
Description	Write configuration register2 of Octa Flash memory in SPI
Arguments	*myctrl : Pointer to the control structure for instance add : Pointer to configuration register2 data : Setting value to configuration register2
Return value	0 : Operation is success -1 : Operation is failed

flash_erase_4k	
Outline	Erase data by 4Kbytes
Declaration	int flash_erase_4k (qspiflash_at25_ctrl_t * myctrl)
Description	Erase data by 4Kbytes
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation success -1 : Operation Failed

flash_erase_64k	
Outline	Erase data by 64Kbytes
Declaration	int flash_erase_64k (qspiflash_at25_ctrl_t * myctrl)
Description	Erase data by 64Kbytes
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation success -1 : Operation Failed

octaram_open	
Outline	Initialize the Octa RAM
Declaration	int octaram_open (xspidevice_ctrl_t * ctrl, xspidevice_cfg_t const * cfg)
Description	Initialize the Octa RAM.
Arguments	*ctrl : Pointer to xSPI device control structure *cfg : Pointer to xSPI device configure structure
Return value	0 : Initialization is complete -1 : Initialization is failed

octaram_close	
Outline	Close the Octa RAM
Declaration	int octaram_close (xspidevice_ctrl_t * ctrl)
Description	Close the Octa RAM.
Arguments	*ctrl : Pointer to xSPI device control structure
Return value	0 : Close process is complete -1 : Close process is failed

octaram_exec_op	
Outline	Execute operation to Octa RAM
Declaration	void octaram_exec_op (xspidevice_ctrl_t * ctrl, xspi_op_t const * op, bool is_write)
Description	Execute operation to Octa RAM.
Arguments	*ctrl : Pointer to xSPI device control structure *op : Pointer to xSPI interface operation structure is_write : Specifies the transmission/reception direction of the operation(0=read、1=write)
Return value	0 : Execute the operation is complete -1 : Execute the operation is failed

octaram_enter_xip	
Outline	Set the Memory-map Mode to Octa RAM
Declaration	int octaram_enter_xip (xspidevice_ctrl_t * ctrl)
Description	Set the Memory-map Mode to Octa RAM
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : Setting is complete -1 : Setting is failed

octaram_exit_xip	
Outline	Finish the Memory-map Mode to Octa RAM
Declaration	int octaram_exit_xip (xspi_ctrl_t * const ctrl)
Description	Finish the Memory-map Mode to Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned

octaram_get_info	
Outline	Get the information of the Octa RAM
Declaration	int octaram_get_info (xspidevice_ctrl_t * ctrl, xspidevice_info_t * info)
Description	Get the information of the Octa RAM.
Arguments	*ctrl : Pointer to the control structure for instance * info : Information of the Octa RAM
Return value	0 : This value is always returned

octaram_read	
Outline	Data read from the Octa RAM
Declaration	int octaram_read (xspi_ctrl_t * const ctrl, void * buffer, size_t address, size_t length)
Description	The processing of this function is undefined. Calling this function will return an error and exit.
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer to store read data address: Data read source address length : Read data size
Return value	-1 : This value is always returned

octaram_write	
Outline	Data write to the Octa RAM
Declaration	int octaram_write (xspi_ctrl_t * const ctrl, void const * buffer, size_t address, size_t length)
Description	The processing of this function is undefined. Calling this function will return an error and exit.
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer where write data is stored address: Data write destination address length : Write data size
Return value	-1 : This value is always returned

octaram_erase	
Outline	Erase the Octa RAM
Declaration	int octaram_erase (xspidevice_ctrl_t * ctrl, size_t address, size_t length)
Description	The processing of this function is undefined. Calling this function will return an error and exit.
Arguments	*ctrl : Pointer to the control structure for instance address: Start address for data erase length : erase size
Return value	-1 : This value is always returned

octaram_get_write_status	
Outline	Gets the write status of the Octa RAM
Declaration	int octaram_get_write_status (xspidevice_ctrl_t * ctrl)
Description	The processing of this function is undefined. Calling this function will return an error and exit.
Arguments	*ctrl : Pointer to the control structure for instance
Return value	-1 : This value is always returned

octaram_write_mode_register	
Outline	Write to the mode register of Octa RAM
Declaration	int octaram_write_mode_register (octaram_apsxx_ctrl_t * myctrl, uint16_t data)
Description	Write to the mode register of Octa RAM
Arguments	*myctrl : Pointer to the control structure for instance data : Setting value to mode register
Return value	0 : Operation success -1 : Operation failed

5.12.2.4 module control functions

The module control functions that control each module are shown below.

syc_init	
Outline	Initialize the Cortex®-A55 built-in Generic Timer
Declaration	void syc_init (unsigned int freq)
Description	Initialize the Cortex®-A55 built-in Generic Timer and start count
Arguments	freq : Frequency of the counter
Return value	None

generic_delay_timer_init	
Outline	Initialize timer management parameters
Declaration	void generic_delay_timer_init (void)
Description	Timer parameter is initialized. By executing the syc_init() function and this function, the following wait functions can be executed. functionudelay(): Wait in usec unit. functionmdelay(): Wait in msec unit.
Arguments	None
Return value	None

pfc_setup	
Outline	Set up the pin
Declaration	void pfc_setup (void)
Description	Set up pin setting of SCIF and Octa Memory Controller.
Arguments	None
Return value	None

pfc_mux_setup	
Outline	Set up pin setting of SCIF
Declaration	void pfc_mux_setup (void)
Description	Set up pin setting of SCIF and octa Memory Controller. Set the P6 pin as SCIF0.
Arguments	None
Return value	None

pfc_qspi_setup	
Outline	Set up pin setting of Octa Memory Controller
Declaration	void pfc_qspi_setup (void)
Description	Set up pin setting of Octa Memory Controller.
Arguments	None
Return value	None

cpg_setup	
Outline	Set up CPG
Declaration	void cpg_setup (void)
Description	Set up CPG.
Arguments	None
Return value	None

cpg_ctrl_clkrst	
Outline	Execute write processing to the clock control register and reset control register
Declaration	void cpg_ctrl_clkrst (CPG_SETUP_DATA const *array、uint32_t num)
Description	Execute write processing to the clock control register and reset control register.
Arguments	*array : Pointer to CPG setup data table num : Number of elements in CPG setup data table
Return value	None

cpg_selector_on_off	
Outline	Clock ON/OFF setting for modules whose clocks are managed by the source clock selector
Declaration	void cpg_selector_on_off (uint32_t sel, uint8_t flag)
Description	The source clock selector simultaneously turns ON/OFF the clocks of the modules to which the clock is to be supplied. Call this function to turn the clock on/off before and after changing the source clock selector.
Arguments	sel : Specifies the source clock selector to change flag : Set the clock ON/OFF
Return value	None

cpg_pll_start_146	
Outline	Start process of PLL1/PLL4/PLL6
Declaration	void cpg_pll_start_146 (CPG_PLL_SETDATA_146 *pdata)
Description	By operating the registers of PLL1/PLL4/PLL6, the frequency is set, standby is released, and PLL operation is started.
Arguments	*pdata : Pointer to PLL1/4/6 setting table
Return value	None

cpg_pll_setup	
Outline	Set up PLL setting
Declaration	void cpg_pll_setup (void)
Description	Reset the PLL4/PLL6 and set the frequency, start the PLL operation.
Arguments	None
Return value	None

cpg_div_sel_setup	
Outline	Write to Division Ratio Setting Register and Source Clock Setting Register.
Declaration	void cpg_div_sel_setup (CPG_REG_SETTING *tbl, uint32_t size)
Description	Write to Division Ratio Setting Register and Source Clock Setting Register.
Arguments	*tbl : Pointer to CPG register setting table size : Size of CPG register setting table
Return value	None

cpg_div_sel_static_setup	
Outline	Perform Static Switching Frequency Dividers and Selector settings.
Declaration	void cpg_div_sel_static_setup (void)
Description	Perform Static Switching Frequency Dividers and Selector settings. Set the registers registered in cpg_static_select_tbl.
Arguments	None
Return value	None

cpg_div_sel_dynamic_setup	
Outline	Perform Dynamic Switching Frequency Dividers and Selector settings.
Declaration	void cpg_div_sel_dynamic_setup (void)
Description	Perform Dynamic Switching Frequency Dividers and Selector settings. Set the registers registered in cpg_dynamic_select_tbl.
Arguments	None
Return value	None

cpg_clk_on_setup	
Outline	Execute clock supply settings for each module
Declaration	void cpg_clk_on_setup (void)
Description	Execute clock supply settings for each module. Set the registers registered in cpg_clk_on_tbl.
Arguments	None
Return value	None

cpg_reset_setup	
Outline	Execute reset release setting for each module
Declaration	void cpg_reset_setup (void)
Description	Execute reset release setting for each module. Set the registers registered in cpg_reset_tbl.
Arguments	None
Return value	None

cpg_active_ddr	
Outline	Execute DDR PHY clock and reset release settings
Declaration	void cpg_active_ddr (void (*disable_phy)(void))
Description	Execute DDR PHY clock and reset release settings.
Arguments	*disable_phy : Pointer to disable_phy_clk function
Return value	None

cpg_reset_ddr_mc	
Outline	Execute DDR memory controller clock and reset release settings
Declaration	void cpg_active_ddr (void (*disable_phy)(void))
Description	Execute DDR memory controller clock and reset release settings.
Arguments	*disable_phy : Pointer to disable_phy_clk function
Return value	None

cpg_early_setup	
Outline	Executes clock supply to modules required for preprocessing of platform settings and reset release settings
Declaration	void cpg_early_setup (void)
Description	Executes clock supply to modules required for preprocessing of platform settings and reset release settings. This function sets the registers registered in early_setup_tbl.
Arguments	None
Return value	None

cpg_wdtrst_sel_setup	
Outline	Execute reset release setting of Watchdog Timer module
Declaration	void cpg_wdtrst_sel_setup (void)
Description	Execute reset release setting of Watchdog Timer module.
Arguments	None
Return value	None

cpu_cpg_setup	
Outline	Execute clock division ratio setting to CPU
Declaration	void cpu_cpg_setup (void)
Description	Set the clock division ratio to the CPU to 1/1.
Arguments	None
Return value	None

cpg_search_spi_clk_table	
Outline	Search the clock frequency register setting table corresponding to the clock frequency supplied to the Octa Flash memory / Octa RAM
Declaration	int cpg_search_spi_clk_table (int frequency)
Description	Search the element number corresponding to the clock frequency supplied to the Octa Flash memory/ Octa RAM from the clock frequency register setting table.
Arguments	frequency : Clock frequency
Return value	Positive number: Element number of register setting table -1 : The element number corresponding to the clock frequency does not exist in the table

cpg_research_spi_clk_table	
Outline	Search for clock frequency value supplied to Octa Flash memory / Octa RAM
Declaration	int cpg_search_spi_clk_table (uint8_t selector, uint8_t divider)
Description	Searches for the clock frequency to be supplied to the Octa Flash memory / Octa RAM from the clock selector and divider settings set in the CPG register.
Arguments	selector : Source clock selector setting divider : Divider setting
Return value	Positive number: Clock frequency -1 : No parameters in table for clock selector and divider settings

cpg_stop_xspi_clock	
Outline	Stop supplying clock to Serial Flash memory
Declaration	void cpg_stop_xspi_clock (xspi_clock spi)
Description	Stop supplying clock to Serial Flash memory.
Arguments	spi : Specifies the type of Serial Flash memory XSPI_CLOCK_SPIM: Serial Flash memory XSPI_CLOCK_OCTA: Octa Flash memory
Return value	None

cpg_start_xspi_clock	
Outline	Start supplying clock to Serial Flash memory
Declaration	void cpg_start_xspi_clock (xspi_clock spi)
Description	Start supplying clock to Serial Flash memory.
Arguments	spi : Specifies the type of Serial Flash memory XSPI_CLOCK_SPIM: Serial Flash memory XSPI_CLOCK_OCTA: Octa Flash memory
Return value	None

cpg_set_xspi_clock	
Outline	Execute clock setting processing for Serial Flash memory
Declaration	void cpg_set_xspi_clock (xspi_clock spi, int frequency_hz)
Description	Execute clock setting processing for Serial Flash memory.
Arguments	spi : Specifies the type of Serial Flash memory XSPI_CLOCK_SPIM: Serial Flash memory XSPI_CLOCK_OCTA: Octa Flash memory frequency_hz : Clock frequency
Return value	None

cpg_get_xspi_clock	
Outline	Get the clock frequency supplied to the Serial Flash memory
Declaration	int cpg_get_xspi_clock (xspi_clock spi)
Description	Get the clock frequency supplied to the Serial Flash memory.
Arguments	spi : Specifies the type of Serial Flash memory XSPI_CLOCK_SPIM: Serial Flash memory XSPI_CLOCK_OCTA: Octa Flash memory)
Return value	positive number: Clock frequency -1 : Failed to get clock frequency

wait_until_32	
Outline	Wait for completion of clock setting
Declaration	int wait_until_32 (uintptr_t addr, uint32_t mask, uint32_t test)
Description	This function reads the CPG monitor register and waits until the clock setting is reflected and the clock stabilizes.
Arguments	addr : CPG monitor register mask : Bitmask to read test : Expected read value of CPG monitor register
Return value	None

console_rzg2l_register	
Outline	Configuring console output
Declaration	int console_rzg2l_register (uintptr_t baseaddr, uint32_t clock, uint32_t baud, console_t *console)
Description	Perform the SCIF settings used for console output. After setting, it is possible to output to the console with printf().
Arguments	baseaddr : SCIF register base address of the channel used clock : Input clock frequency for SCIF module baud : Baud rate setting to console output *console : Parameter of console output This parameter is automatically set when this function is executed
Return value	1 : Console output setting success 0 : Console output registration failure

console_set_scope	
Outline	Set the valid state of the console output
Declaration	void console_set_scope (console_t *console, unsigned int scope)
Description	Set the state in which console output processing is enabled.
Arguments	*console : Console output parameter Set parameters obtained by console_rzg2l_register() scope : Sets the state that enables console output processing Please set (CONSOLE_FLAG_BOOT | CONSOLE_FLAG_CRASH) in the IPL
Return value	None

plat_get_bl_image_load_info	
Outline	Get FSP application image load information
Declaration	bl_load_info_t *plat_get_bl_image_load_info(void)
Description	Get FSP application image load information.
Arguments	None
Return value	Positive number: Pointer to FSP application image load information

plat_get_next_bl_params	
Outline	Get FSP application image parameter information
Declaration	bl_params_t *plat_get_next_bl_params (void)
Description	Get FSP application image parameter information.
Arguments	None
Return value	Positive number: Pointer to FSP application image parameter information

rz_check_fsp_header	
Outline	Perform application header check
Declaration	int rz_check_fsp_header (fsp_app_header_t *header)
Description	Check the contents of the application header. This function will return an error if: - entry_addr != entry_addr_inv - dest_addr != dest_addr_inv - payload_size != payload_size_inv - The image copy area address is outside the On-chip RAM or DDR SDRAM area.
Arguments	*header : Pointer to Application Header
Return value	0 : Application header parameters are valid values -1 : Application header parameters are invalid values

rz_update_descs	
Outline	Update the descriptor parameter with the content obtained from the application header
Declaration	void rz_update_descs (fsp_app_header_t *header)
Description	Update the descriptor parameter with the content obtained from the application header. IPL controls the jump destination address of the FSP application by this descriptor.
Arguments	*header : Pointer to Application Header
Return value	None

rz_load_fsp	
Outline	Load FSP application image
Declaration	void rz_load_fsp (void)
Description	The application header is read and checked, and the FSP application is copied according to its contents and the descriptor is updated. If the application header check fails, the IPL will go into an infinite loop and stop the boot process.
Arguments	None
Return value	None

rz_print_descs	
Outline	Parameters of the descriptor are printed to the console
Declaration	void rz_print_descs (void)
Description	Parameters of the descriptor are printed to the console.
Arguments	None
Return value	None

ddr_setup	
Outline	Initialize DDR SDRAM
Declaration	void ddr_setup (void)
Description	Initializes DDR SDRAM and enables read/write access to DDR SDRAM.
Arguments	None
Return value	None

5.13 Flow Chart

5.13.1 IPL (overall) flow

Figure 5.6, Figure 5.7, Figure 5.8 show the flowchart of IPL (overall)

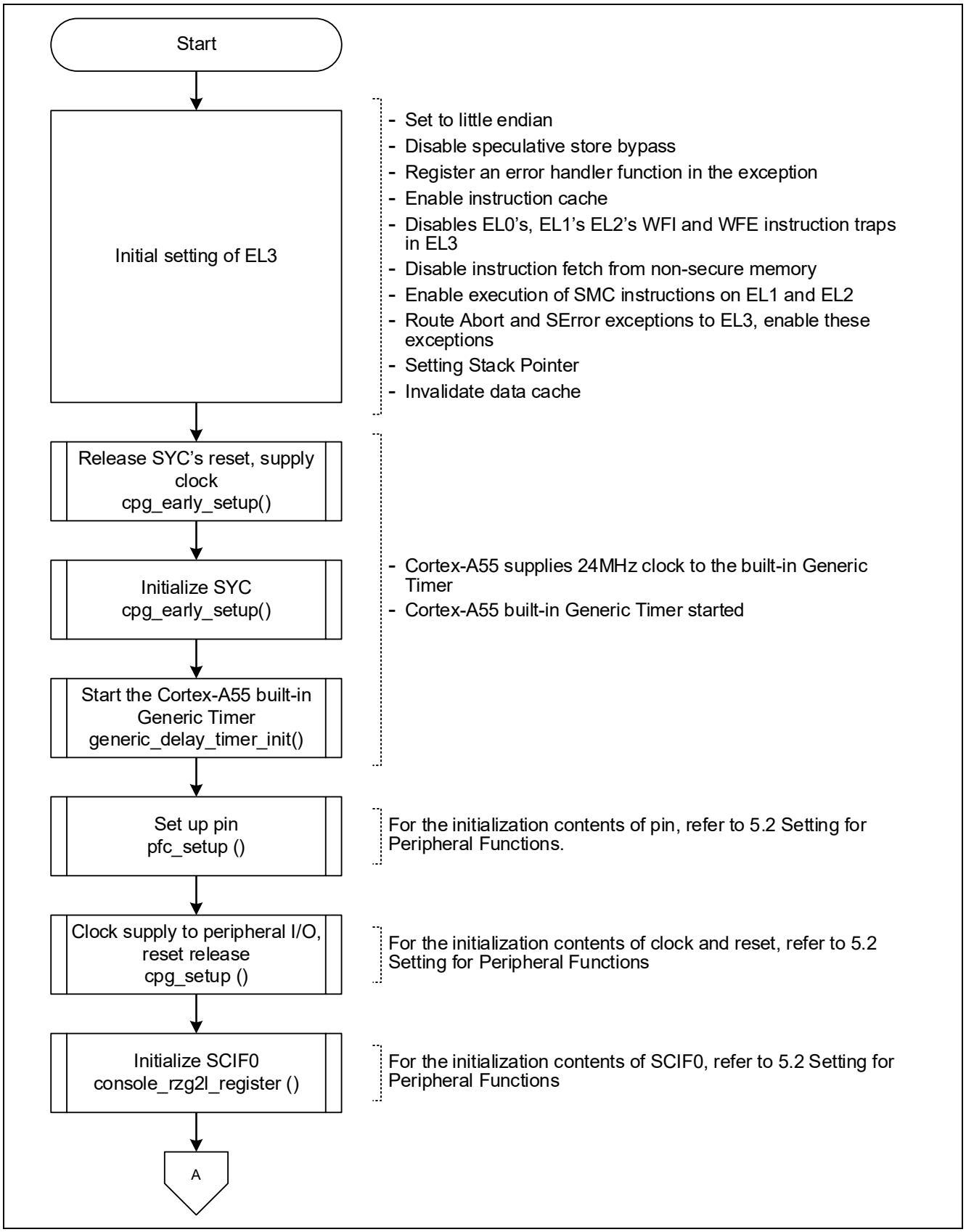


Figure 5.6 IPL Flow Chart part (1/3)

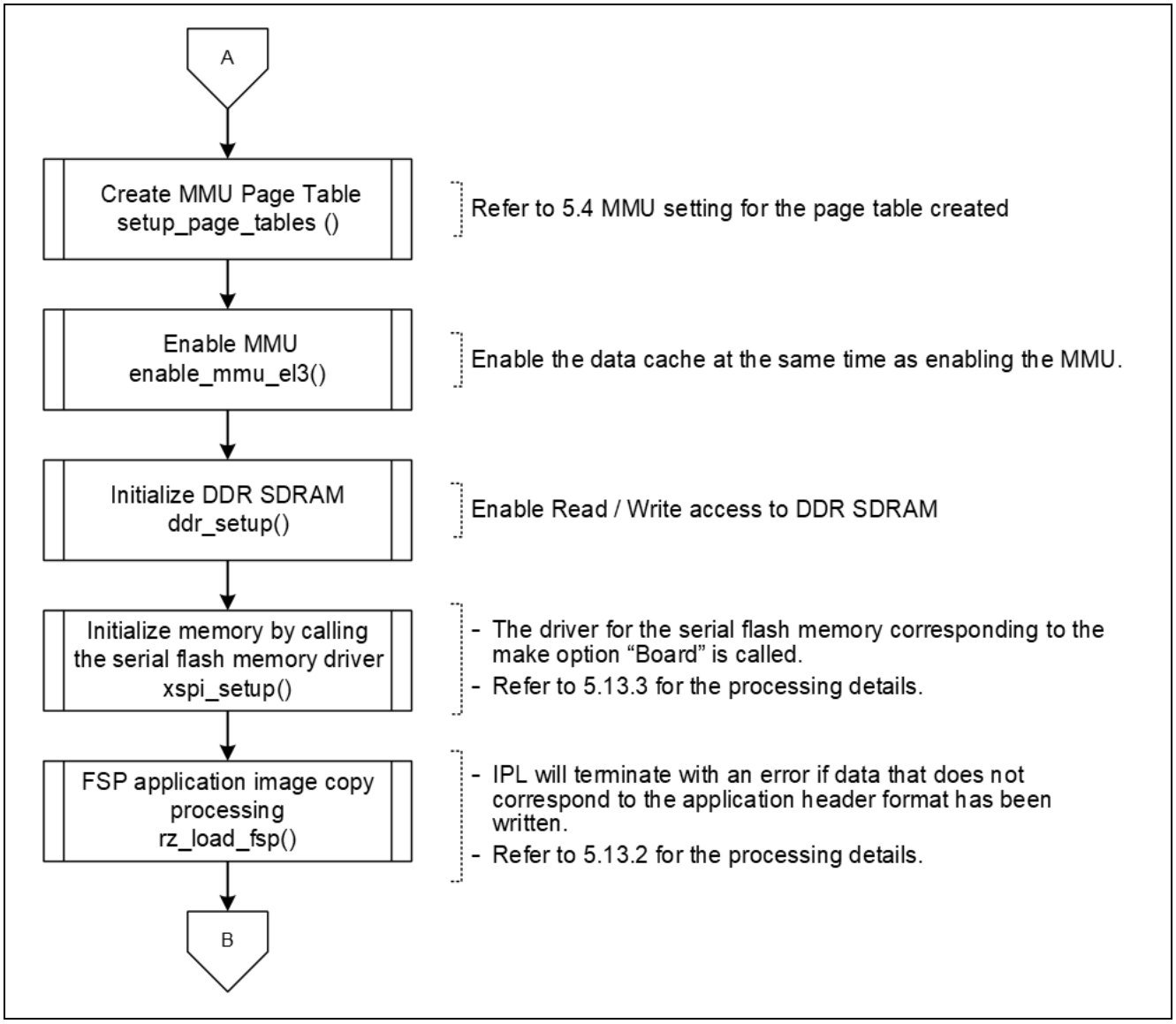


Figure 5.7 IPL Flow Chart part (2/3)

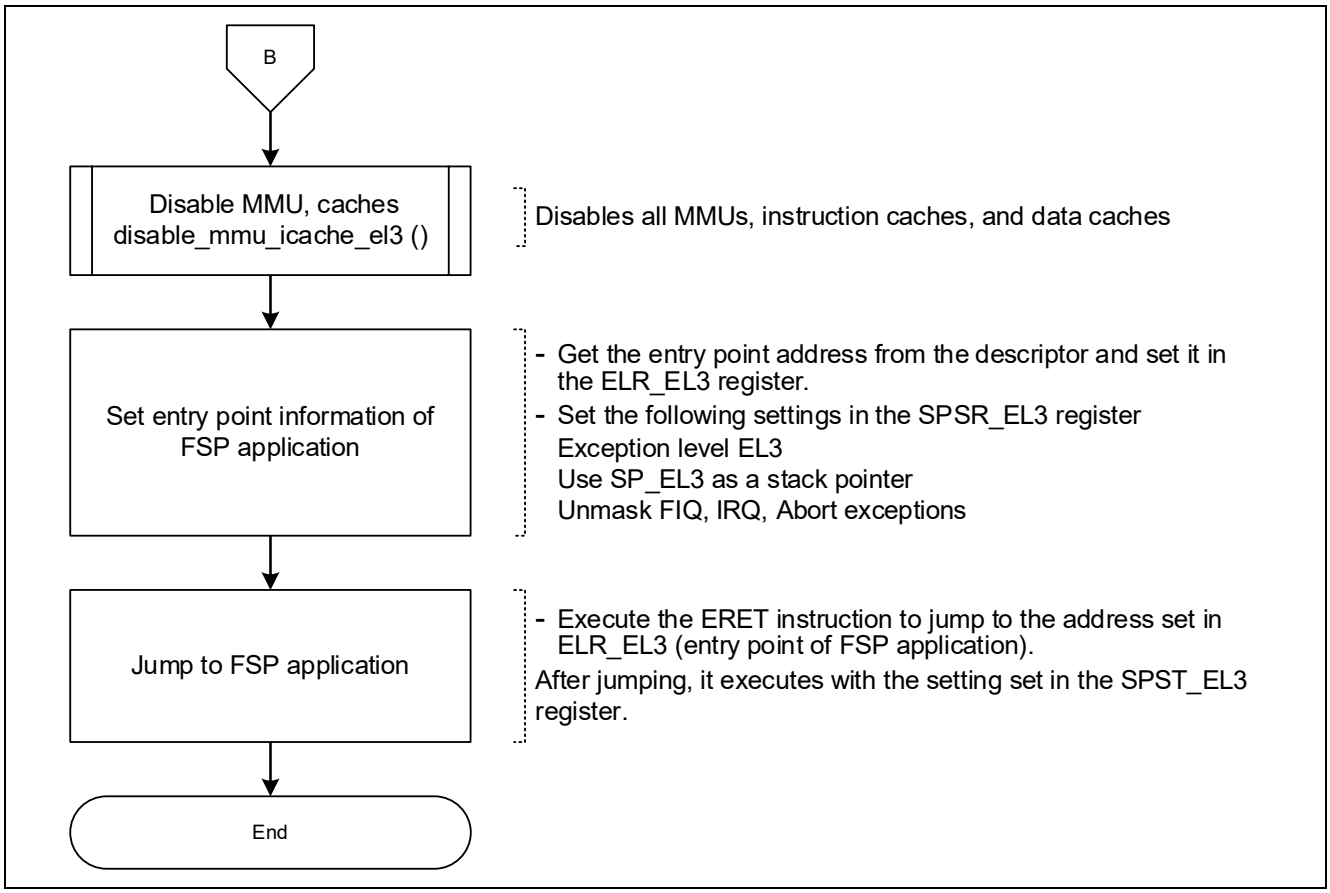


Figure 5.8 IPL Flow Chart part (3/3)

5.13.2 FSP Application Image copy process

Figure 5.9 and Figure 5.10 show the flow chart of the FSP Application image copy process.

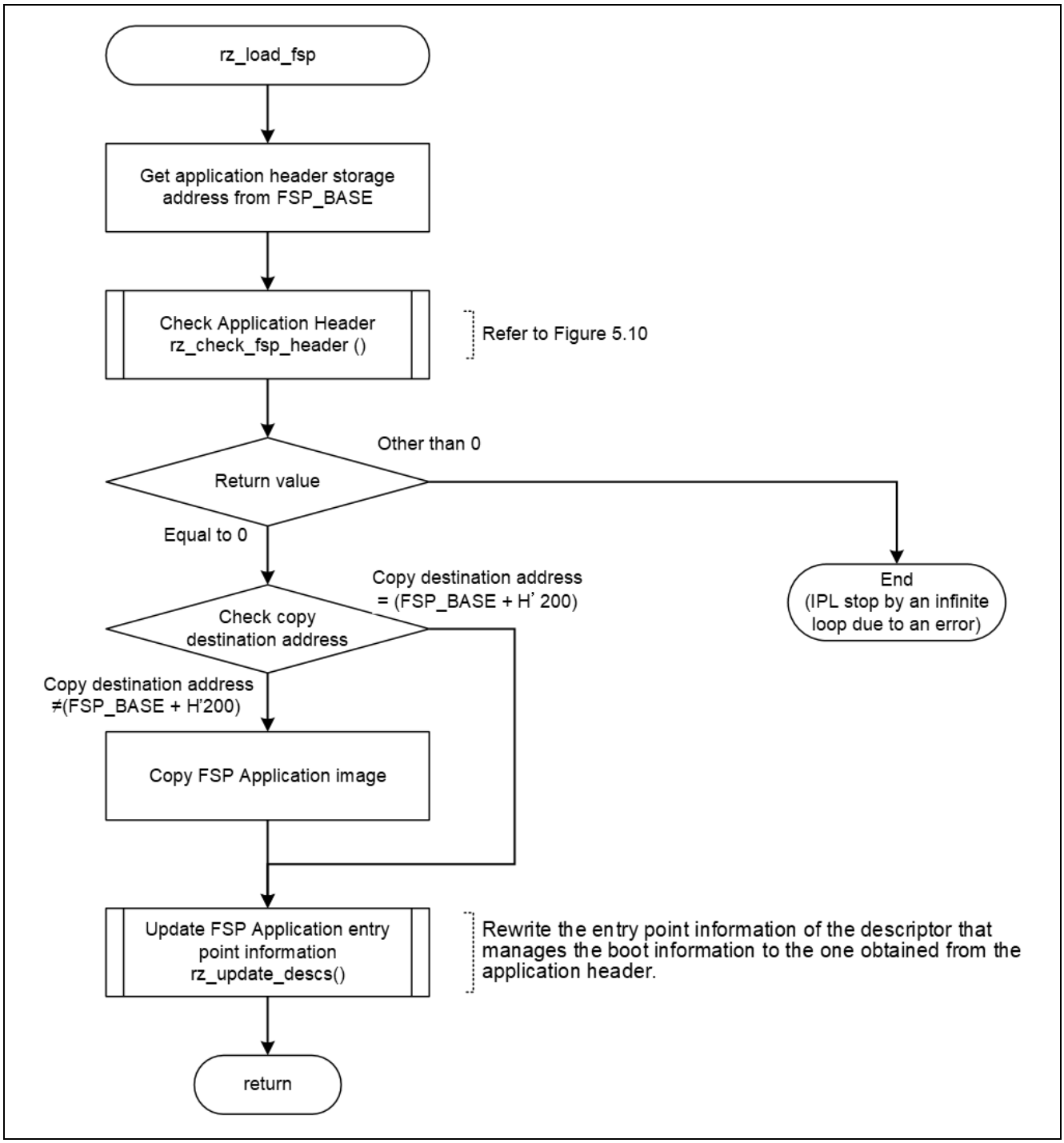


Figure 5.9 Flow Chart of rz_load_fsp()

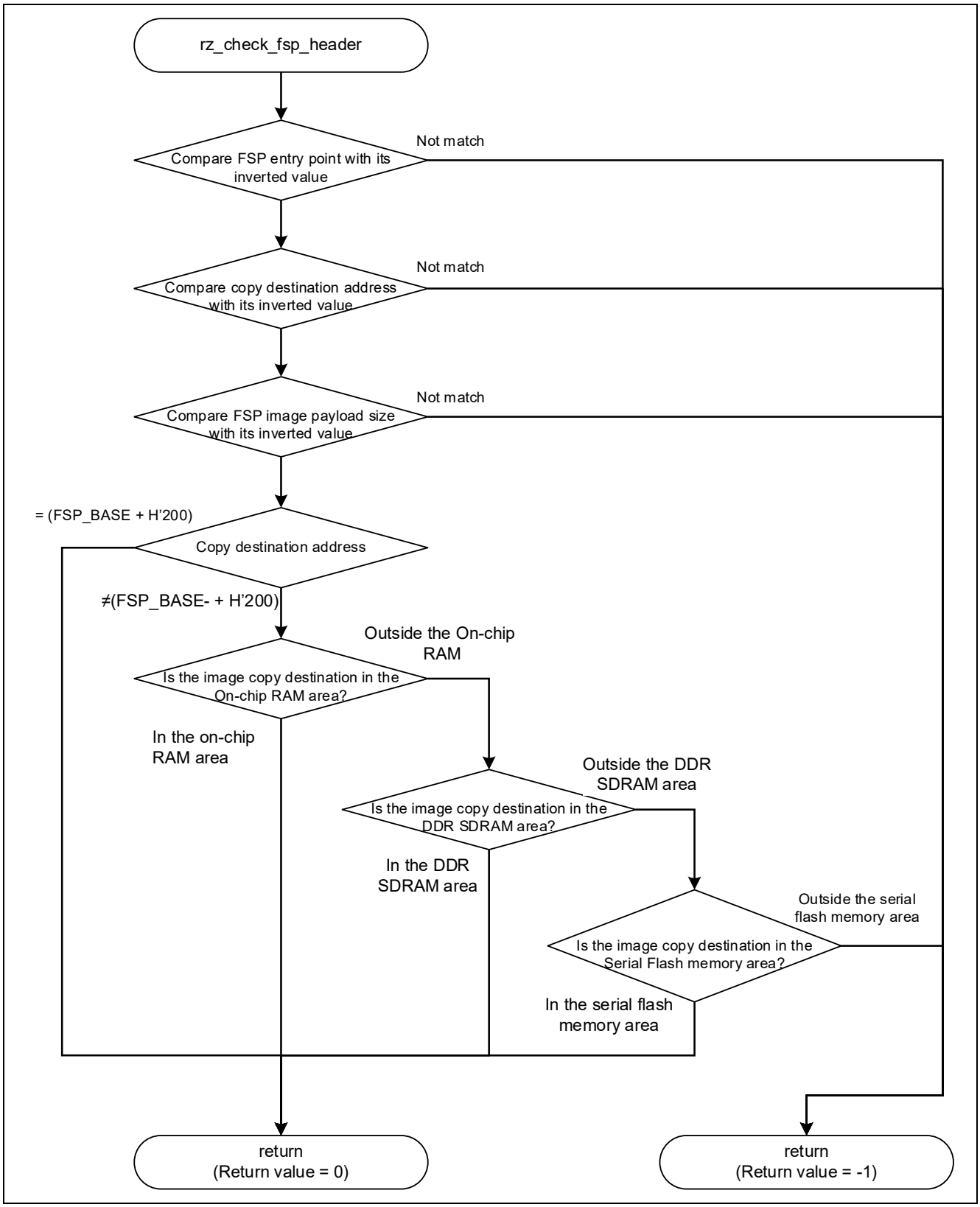


Figure 5.10 Flow Chart of rz_check_fsp_header()

5.13.3 Setting of Octa Flash memory, Octa RAM, and Octa Memory Controller

In this IPL, the Macronix Octa Flash memory (MX66UW1G45G) and the Octa RAM (APS51208N) is set so that it can be accessed by Memory-map Mode via Octa Memory Controller.

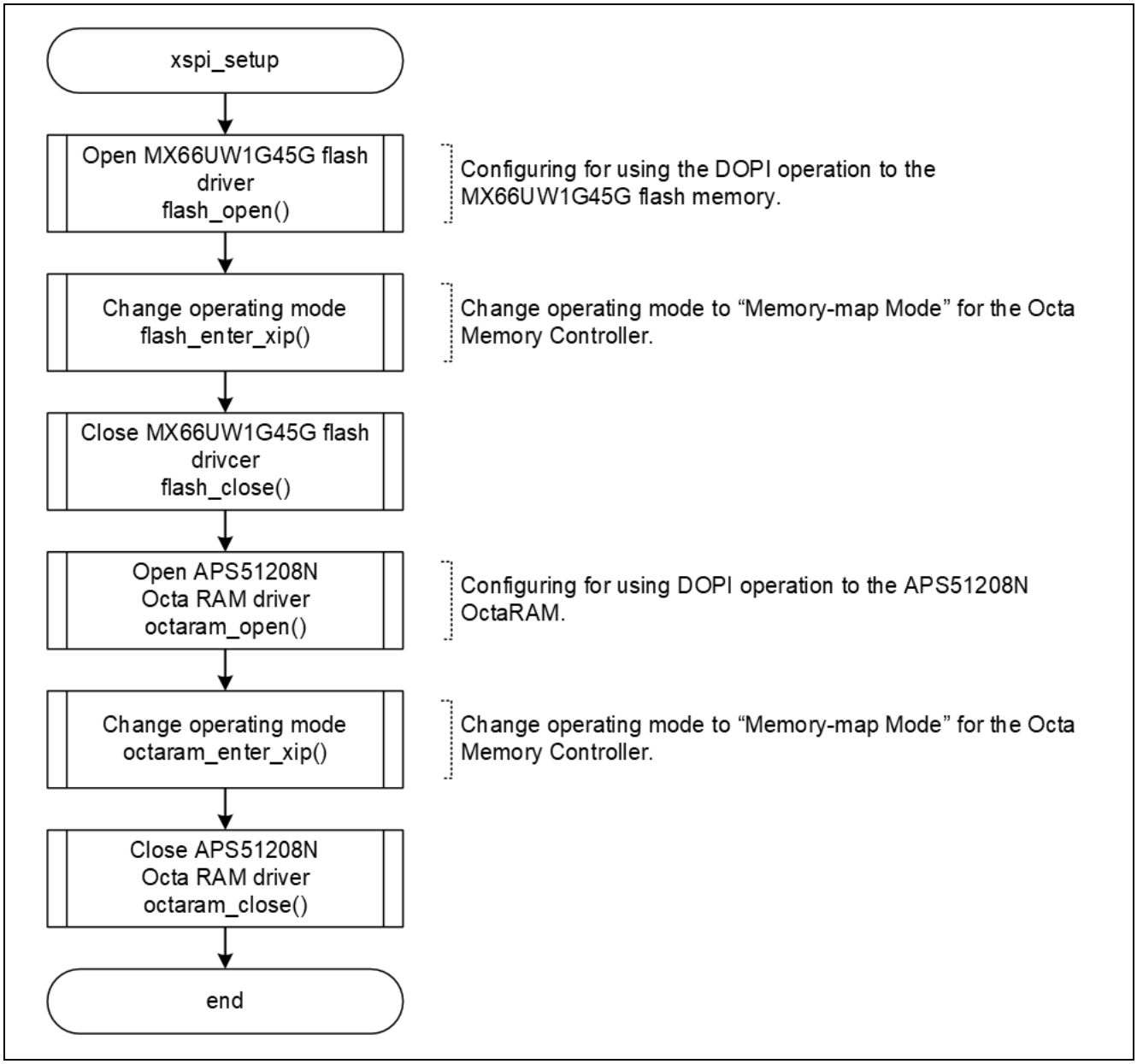


Figure 5.11 Flowchart of xspi_setup

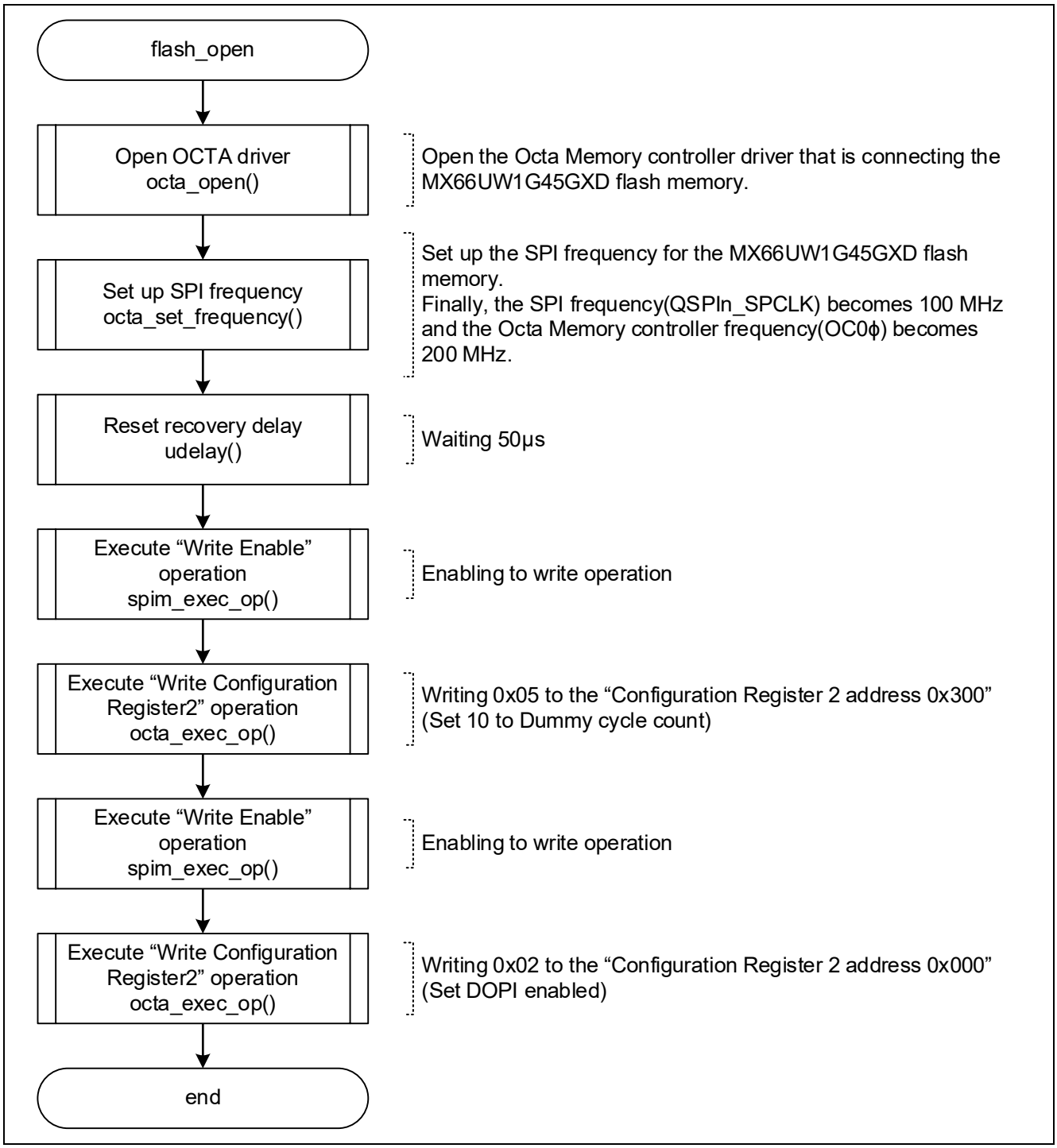


Figure 5.12 Flowchart of flash_open

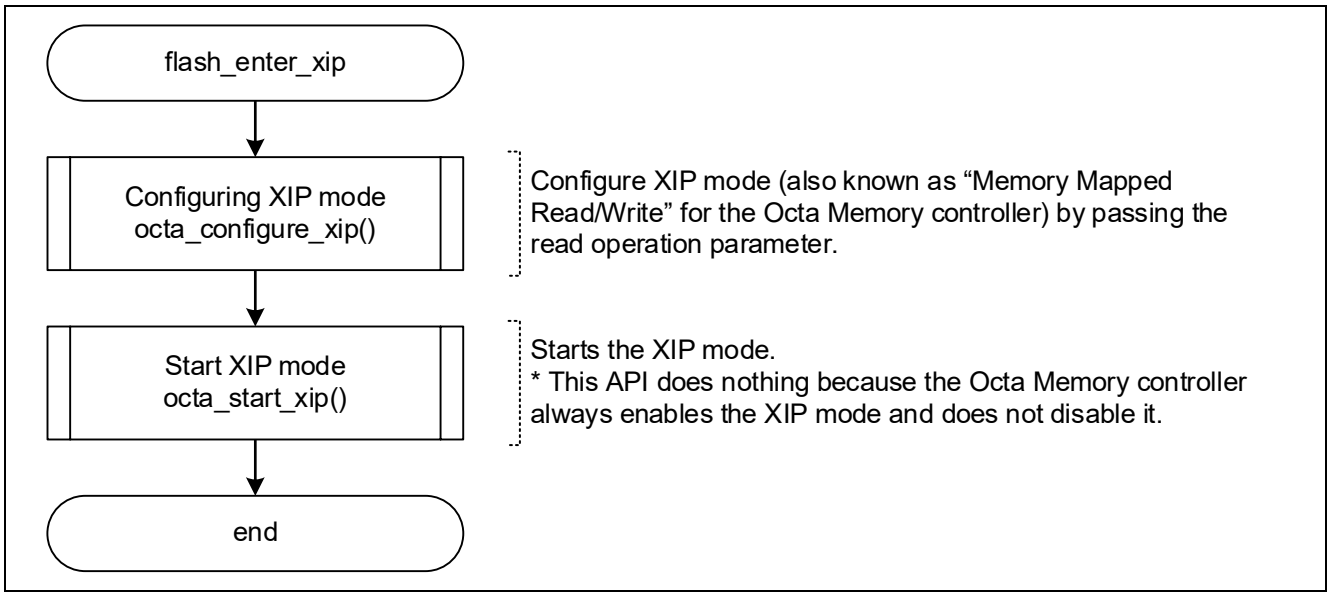


Figure 5.13 Flowchart of flash_enter_xip

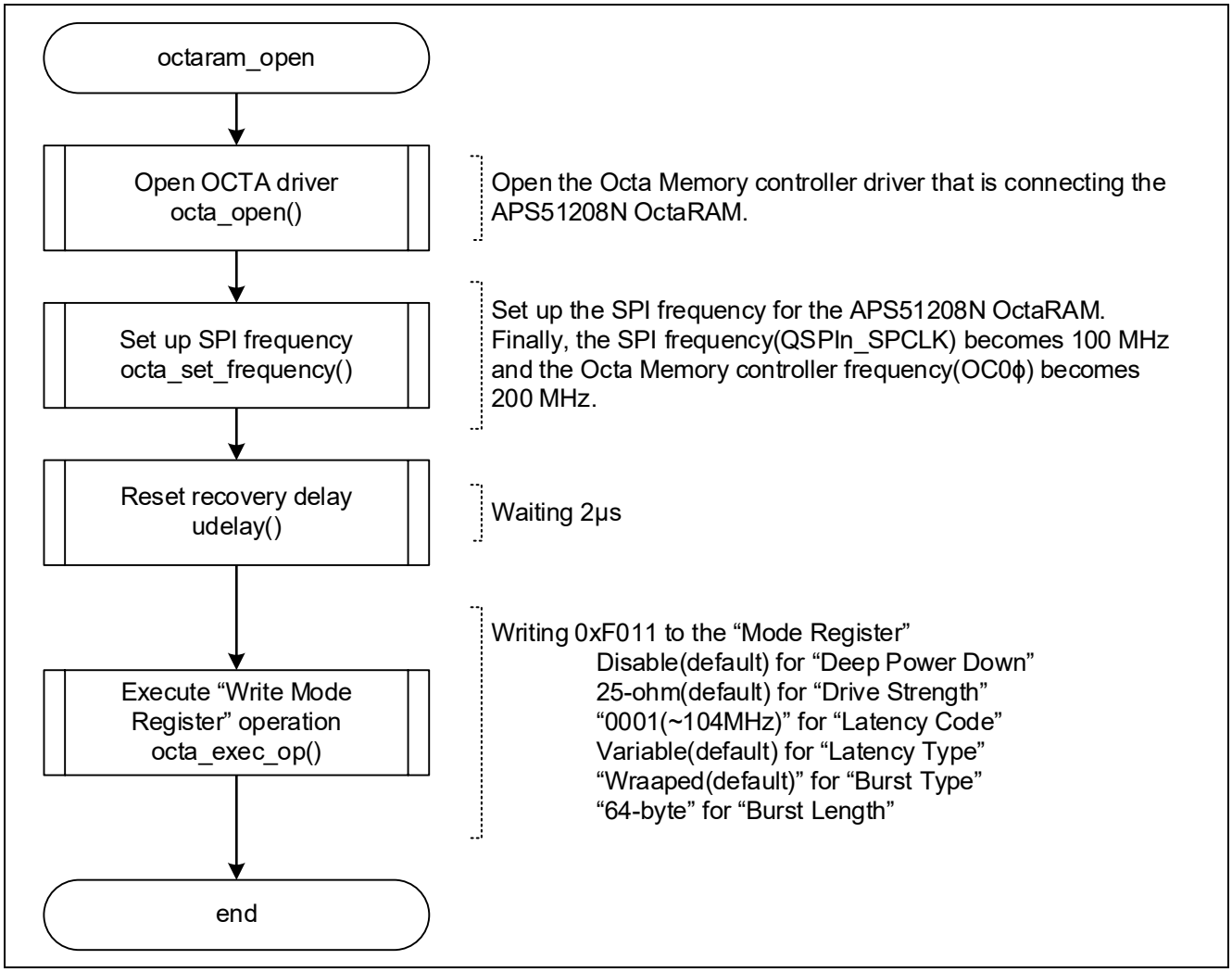


Figure 5.14 Flowchart of octaram_open

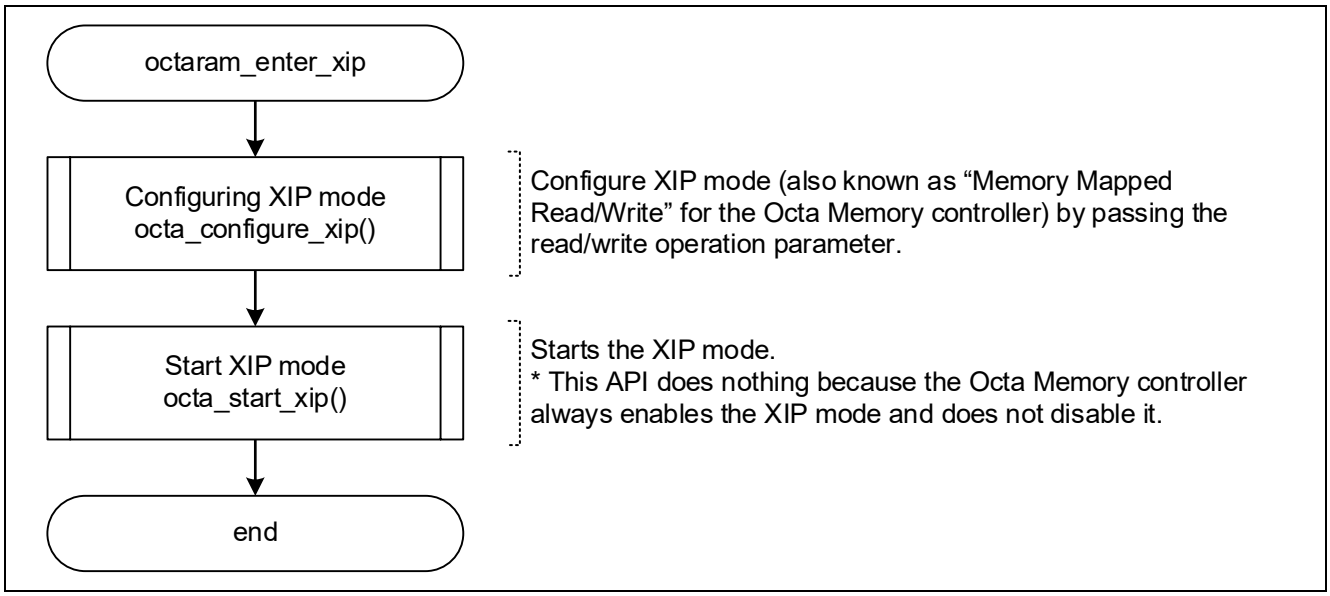


Figure 5.15 Flowchart of octaram_enter_xip

6. IPL development environment construction procedure

This chapter describes the procedure for building the IPL development environment.

6.1 Procedure for building a development environment for Widows10 PC

The procedure for building a development environment for the Windows10 environment is shown below.

6.1.1 Installation of IDE (e² studio) for Windows10

IPL uses e² studio as the development environment.

This chapter shows the installation procedure for Windows10.

Please download the e² studio installer from the Renesas Electronics web page below.

The toolchain is included in the e² studio installer.

The toolchain is installed while you install e² studio.

https://www.renesas.com/jp/ja/software-tool/e-studio

RZ/A3 FSP packs are required to build an IPL execution environment in e² studio. Please download from the following website.

https://github.com/renesas/rza-fsp/releases

6.1.1.1 e² studio installation (Windows10)

1. Double-click the e2 studio installer to launch the e2 studio installation wizard. Then, select the [Custom Install] option and click the [Next] button.

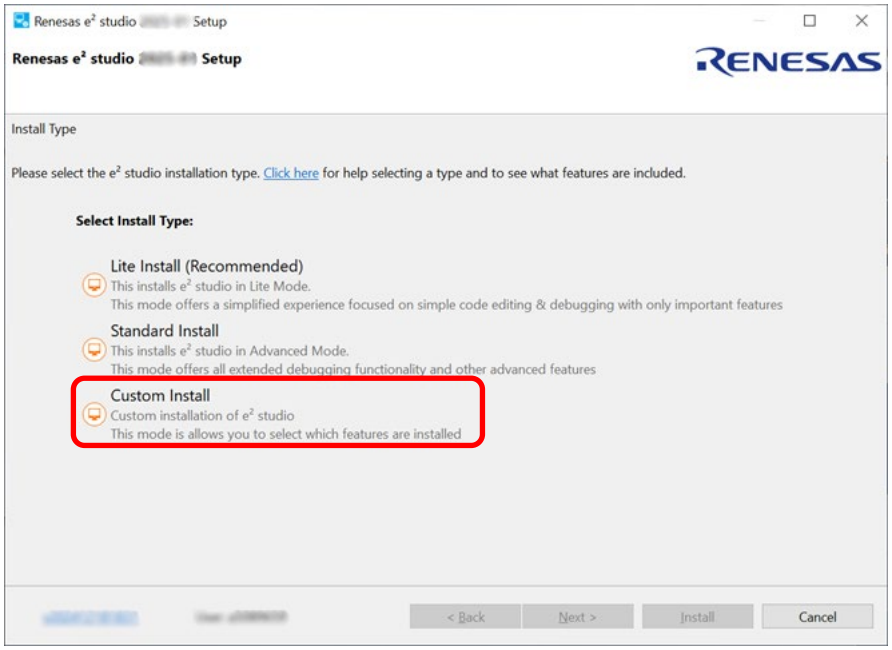


Figure 6.1 e2 studio installation wizard

Note: If you are using a multi-user environment, you may receive a prompt to confirm whether you want to install it for the current user only or for all users.

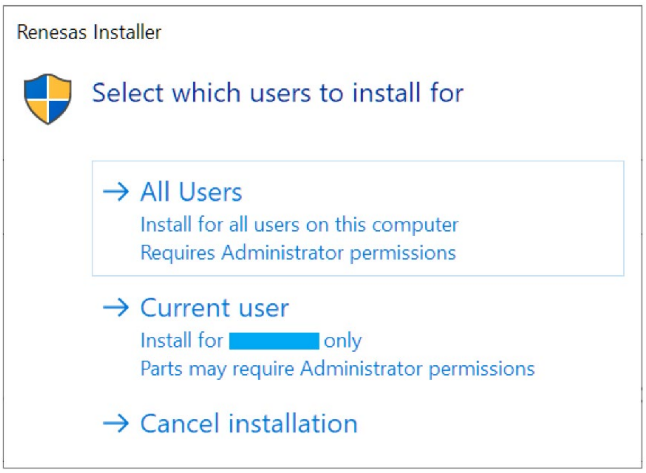


Figure 6.2: Select User for Installation

Note: If e² studio had been installed in your PC, the option to modify, remove the existing version or install e² studio to a different location will be displayed.
In this case, please select [Install].

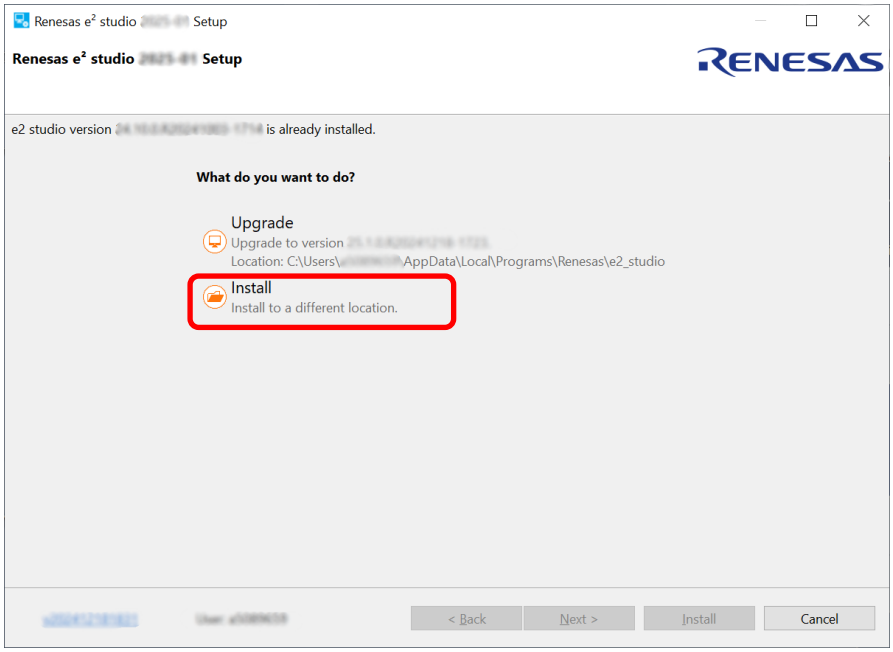


Figure 6.3 e² studio installation wizard

2. "Welcome" page
   User can change the install folder by clicking [Change...]. Click [Next] to continue.

Notes: 1. If you would like to have multiple versions of e² studio, please specify new folder here.
2. Multi-byte characters cannot be used for e² studio installation folder name.

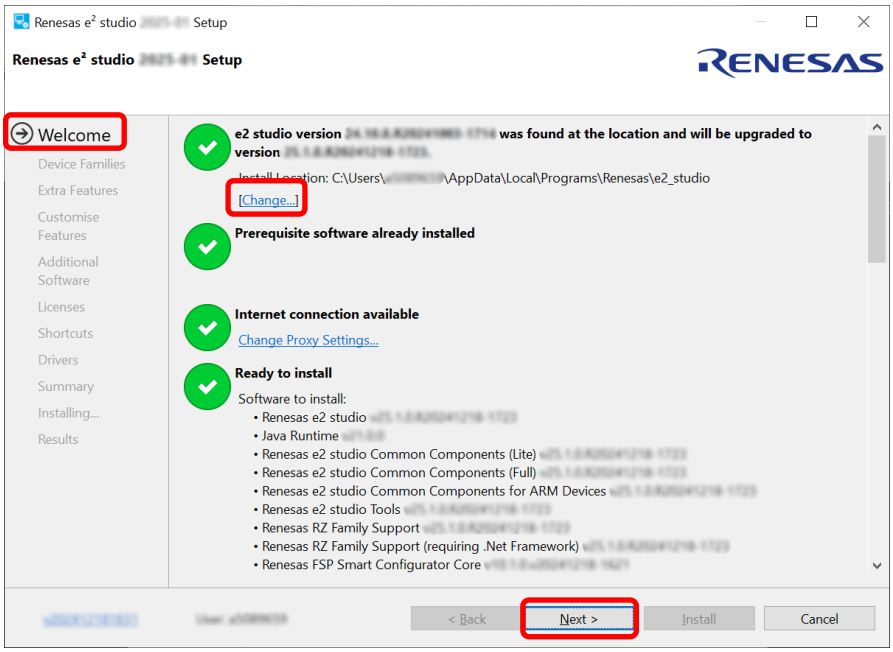


Figure 6.4 Installation of e² studio – Welcome page

3. "Device Families"
   You can select the device families to install. Select [RZ] Device Families to install.
   Click the [Next] button to continue.

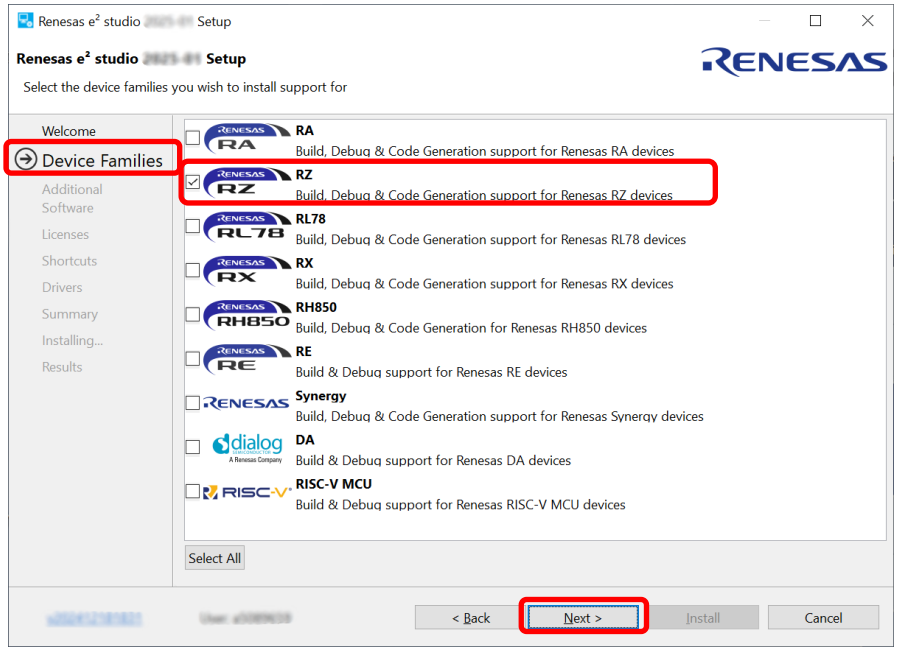


Figure 6.5 Installation of e² studio – Device Families

4. "Extra Features"
   Select Extra Features (i.e., Language packs, SVN & Git support, RTOS support...) to be installed.
   For non-English language users, please select Language packs at this step if needed.
   This document describes an example of using e² studio terminal, so please check [Terminals].
   Click the [Next] button to continue.

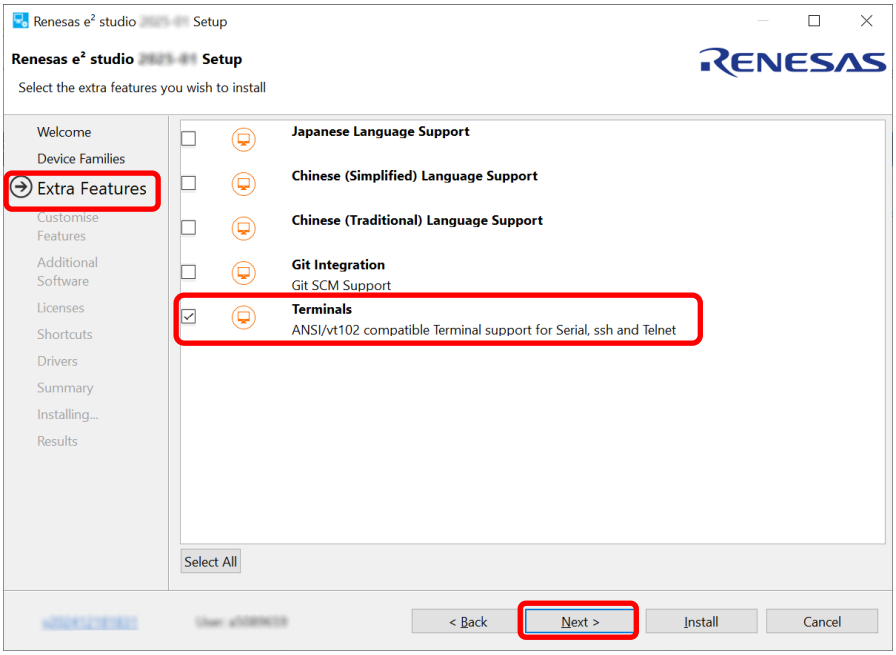


Figure 6.6 Installation of e² studio – Extra Features

5. "Customize Features"
   Be sure to select [Renesas FSP Smart Configurator Core].

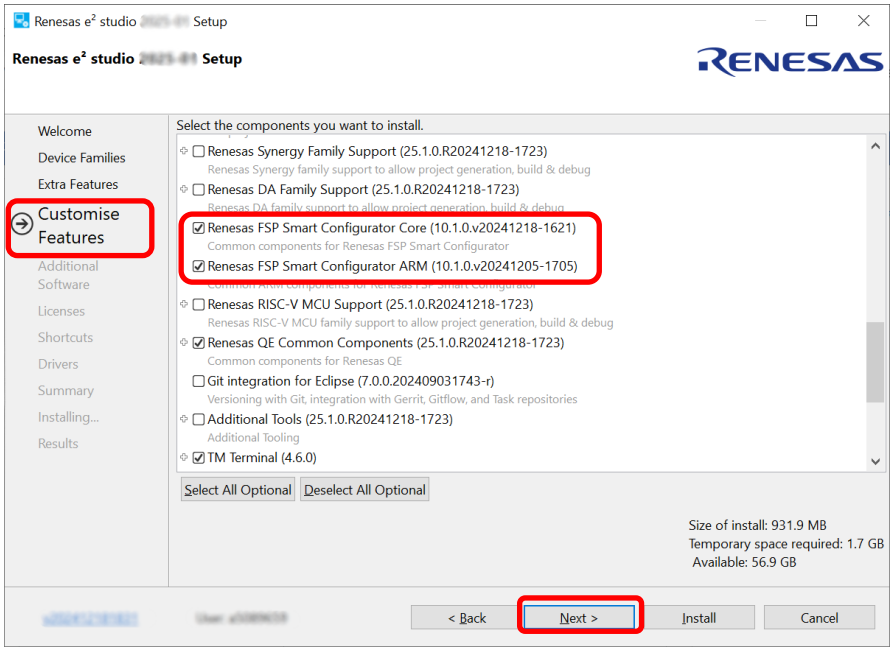


Figure 6.7 Installation of e² studio – Features

6. "Additional Software"
   Select additional software (i.e., compilers, utilities, QE…) to be installed.
   Add the following items if they are not checked, and click [Next] to continue:

   - GCC ARM A-Profile (AArch64 bare-metal) 10.3.2021.07

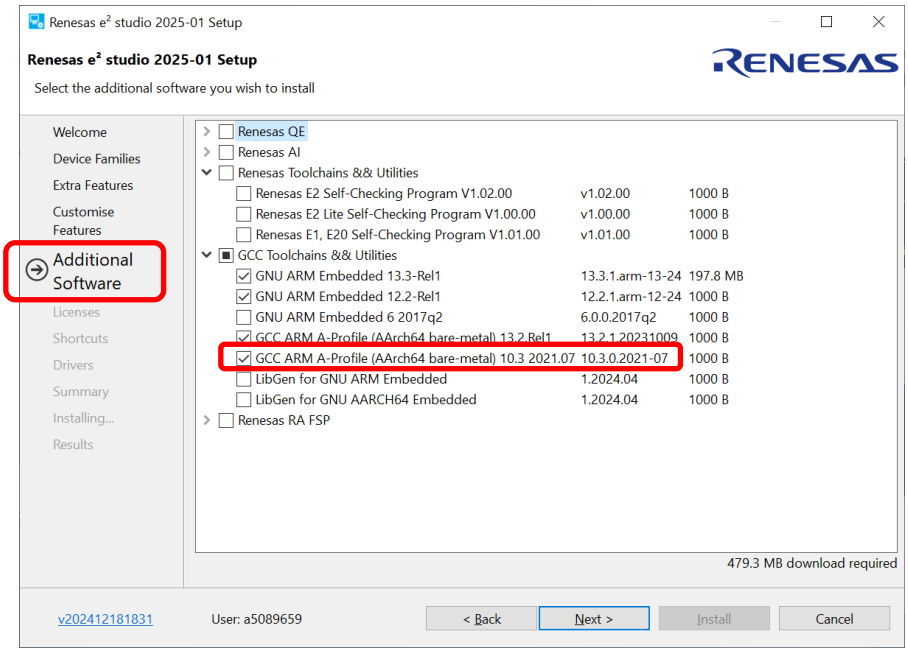


Figure 6.8 Installation of e² studio – Additional Software

7. "License Agreement"
   Please read and accept the software license agreement, then click the [Next] button. Note that acceptance of the license agreement is mandatory; without it, the installation process cannot proceed.

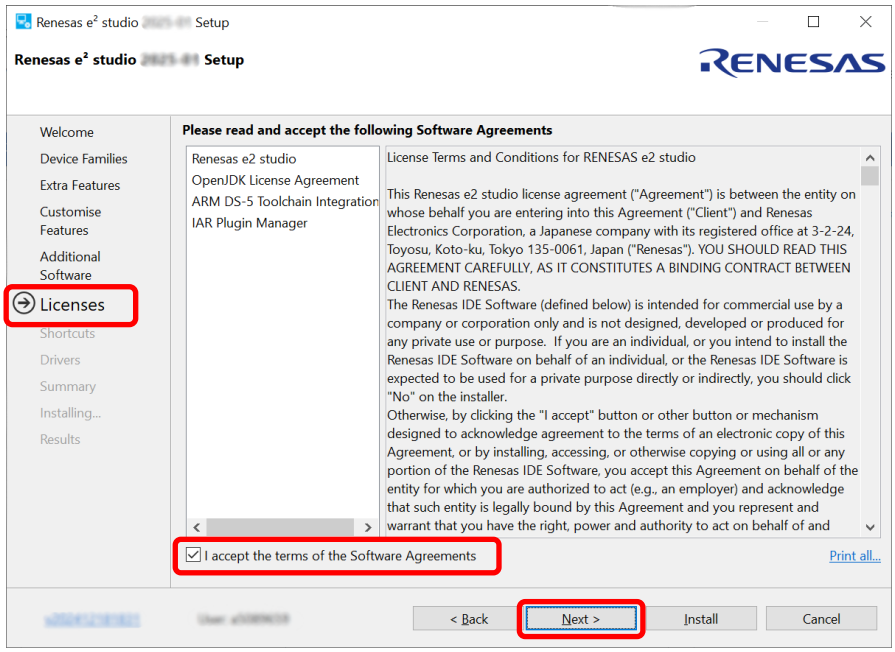


Figure 6.9 Installation of e² studio – Licenses

8. "Shortcuts"
   Select shortcut name for start menu and click [Next] button to continue.

Note:

If e2 studio was installed in another location, it is recommended to rename it to distinguish from the other e2 studio(s).

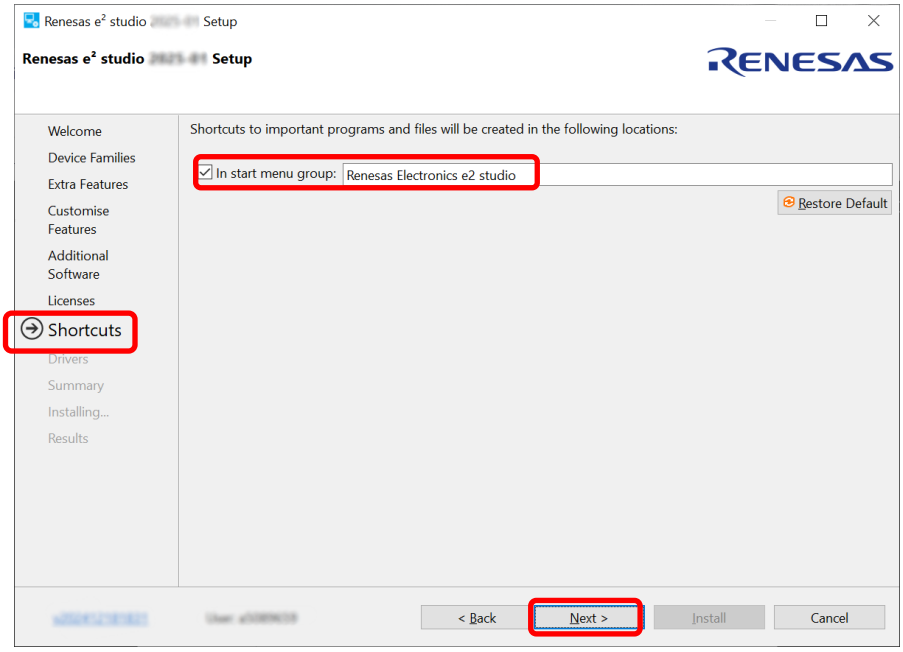


Figure 6.10 Installation of e² studio – Shortcuts

9. "Summary"
   Components list to be installed is shown.
   Please confirm the contents and click the [Install] button to install the Renesas e² studio IDE.

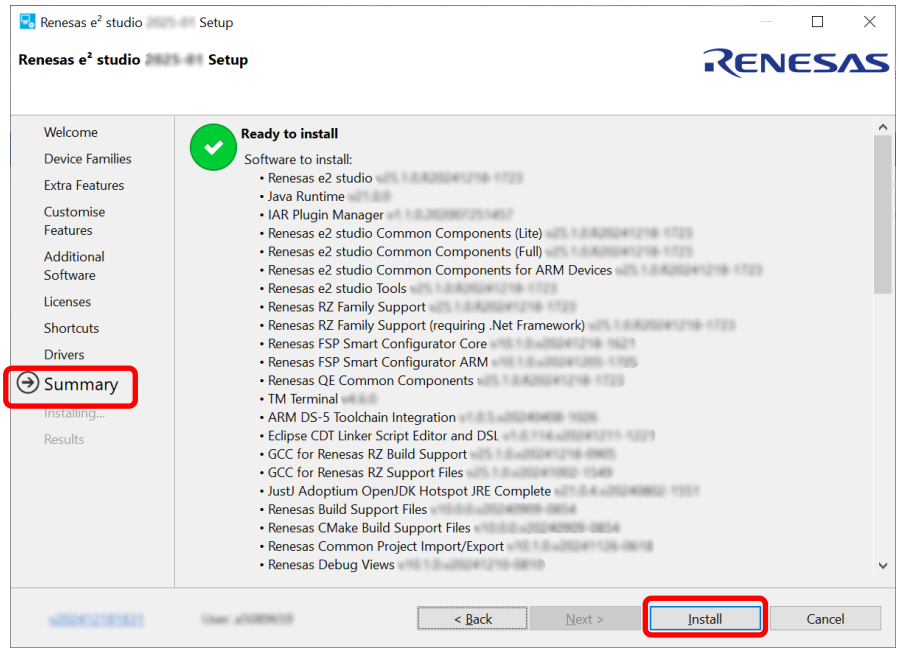


Figure 6.11 Installation of e² studio – Summary

10. "Installing"
    The installation is performed. Depending on selected items of additional software, new dialog prompts may appear during the installation process.

11."Results"

Click the [OK] button to complete the installation.

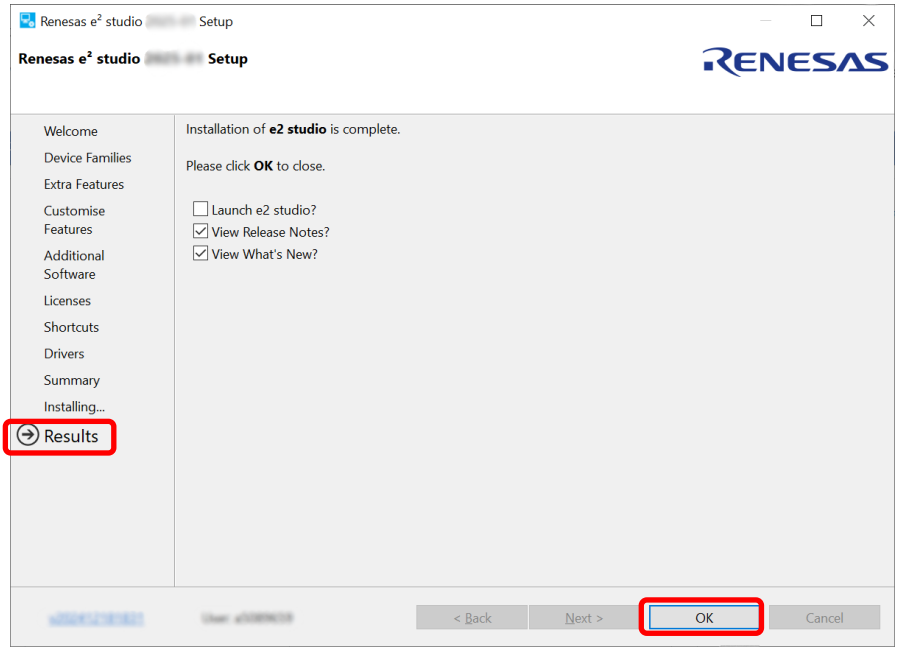


Figure 6.12 Result Page

6.1.1.2 Installation of RZ/A FSP packs (Windows10)

Install the FSP packs for use in the RZ/A3 IPL environment.

1. Exit e² studio
2. Extract FSP Packs.
3. Copy the extracted FSP Packs (named "internal") to the e² studio installation folder as shown below:
   "C:\Renesas\e2_studio\internal\projectgen\rz_fsp\rza\packs"

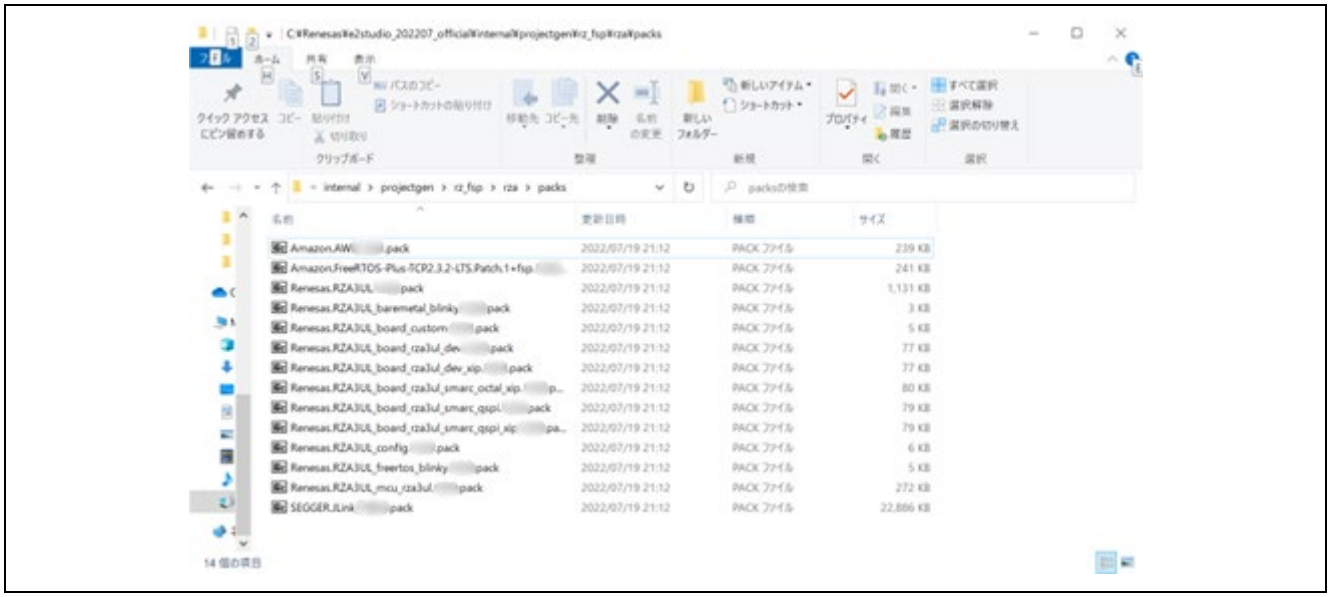


Figure 6.13 Installed packs folder (Windows10)

4. Please re-launch e² studio
   Then packs will be reflected in the development environment.

6.1.2 MSYS2 Installation

You must install a tool called MSYS2 to build the IPL in Windows10 environment.

1. Download MSYS2 from "Installation" on the following website.
   Please install by following the procedure described in "Installation".
   https://www.msys2.org/
   This document describes the case where MSYS2 is installed in the following folder.
   c:\msys64

2. Install the make command on MSYS2.
   Open the start menu and start "MSYS2 MSYS".

Figure 6.14 Launch MSYS2

3. A command window will appear.
   Enter "pacman -S make" on command window the make command on MSYS2.

Figure 6.15 Make command installation (1)

4. Enter ‘y’ when "Proceed with installation" [Y / n] "is output.

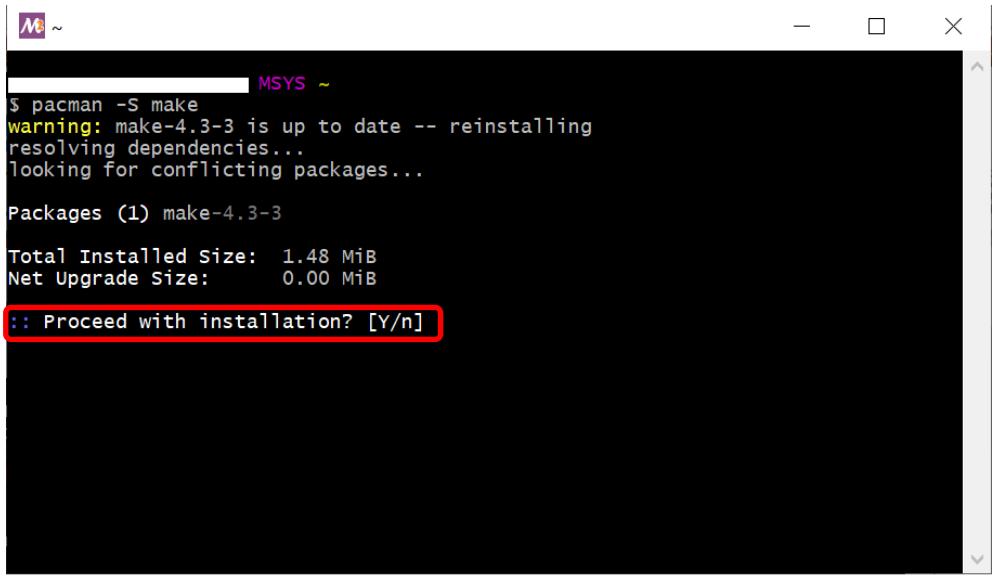


Figure 6.16 Make command installation (2)

5. The make installation is completed.

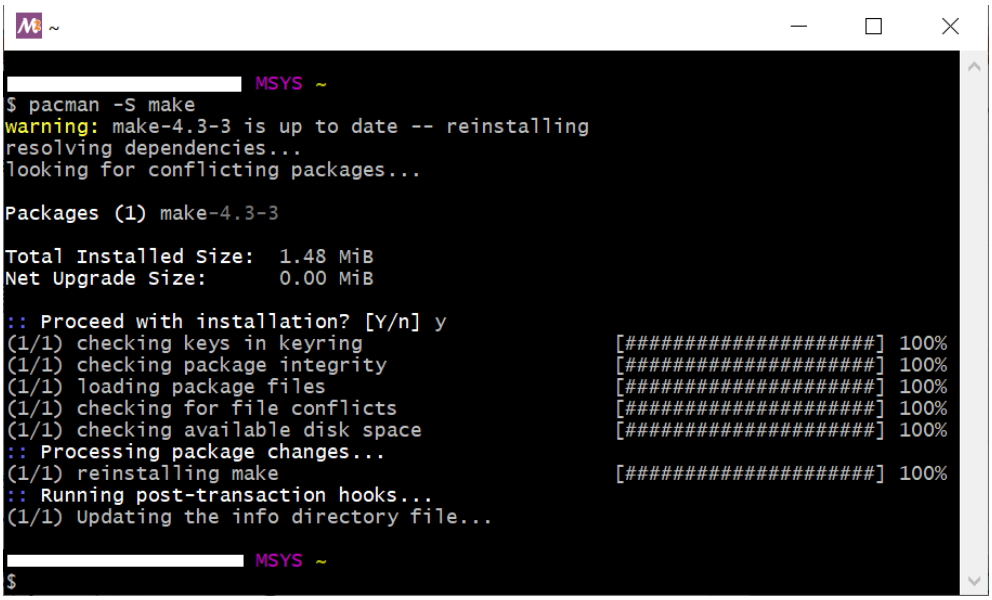


Figure 6.17 Make command installation (3)

6.2 Procedure for building a development environment for Ubuntu 22.04 LTS

The procedure for building a development environment for Ubuntu 22.04 LTS Desktop environment is shown below.

6.2.1 Installation of IDE (e² studio) for Ubuntu 22.04 LTS

IPL uses e² studio as the development environment.

This chapter shows the installation procedure for Ubuntu LTS 22.04.

Please download the e² studio installer from the Renesas Electronics web page below.

The toolchain is included in the e² studio installer.

The toolchain is installed while you install e² studio.

https://www.renesas.com/jp/ja/software-tool/e-studio

RZ/A3 FSP packs are required to build an IPL execution environment in e² studio. Please download from the following website.

https://github.com/renesas/rza-fsp/releases

6.2.1.1 e² studio installation (Ubuntu 22.04 LTS)

1. Execute the following command to give execute permission to the e² studio installer.
   Then, run the installer.
   The command is an example when the e² studio installer
   (file name: "e2studio_installer-2025-01_linux_host.run") is saved in the "~/Downloads" folder.
   Please change the command according to your environment.

```
$ cd ~/Downloads
$ sudo chmod +x e2studio_installer-2025-01_linux_host.run
$ ./e2studio_installer-2025-01_linux_host.run
```

2. User needs to select Install Type as shown below.
   In this material, it is expected that [Custom Install] is selected.
   Then, click [Next >] to continue.

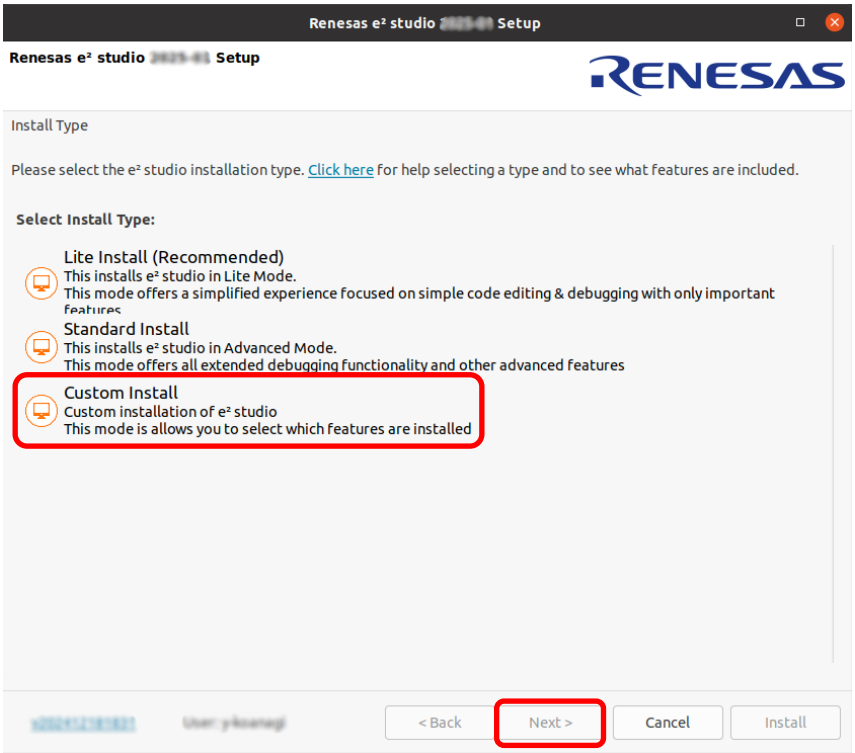


Figure 6.18 Installation of e² studio – Welcome page

3. "Welcome" page
   User can change the install folder by clicking [Change...]. Click [Next] to Click [Next] to continue.

Notes: 1. If you would like to have multiple versions of e² studio, please specify new folder here.
2. Multi-byte characters cannot be used for e² studio installation folder name.

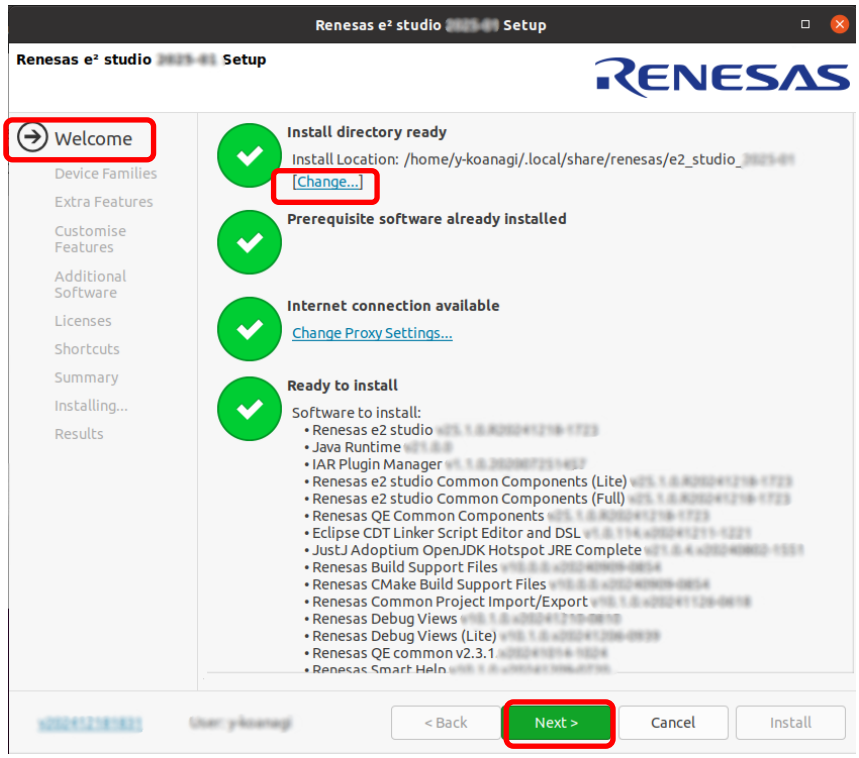


Figure 6.19 Installation of e² studio – Welcome page

4. "Device Families"
   You can select the device families to install.
   Select [RZ] Device Families to install.
   Click the [Next] button to continue.

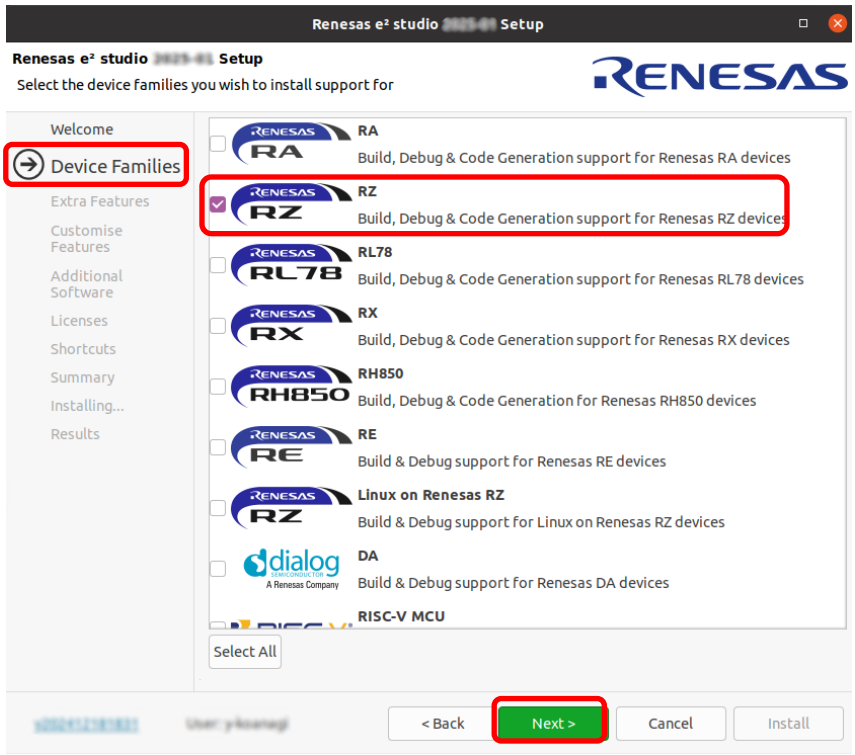


Figure 6.20 Installation of e² studio – Device Families

5. "Extra Features"
   Select Extra Features (i.e., Language packs, SVN & Git support, RTOS support...) to be installed.
   For non-English language users, please select Language packs at this step if needed.
   This document describes an example of using e² studio terminal, so please check [Terminals].
   Click the [Next] button to continue.

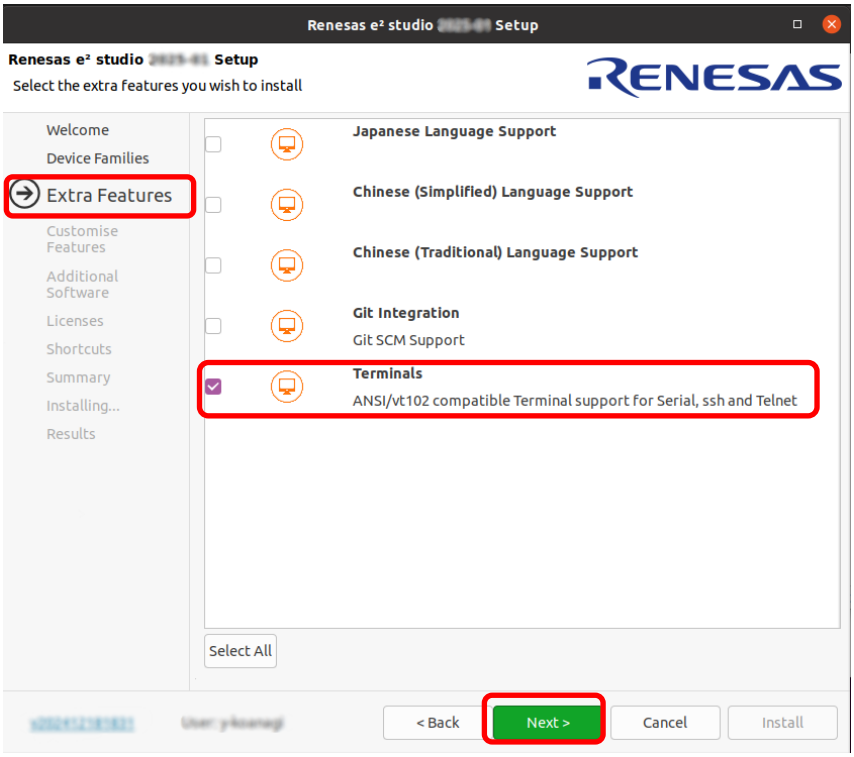


Figure 6.21 Installation of e² studio – Extra Features

6. "Customize Features"
   Select the components to install and click the [Next] button to continue. Be sure that [Renesas FSP Smart Configurator Core] is certainly selected.

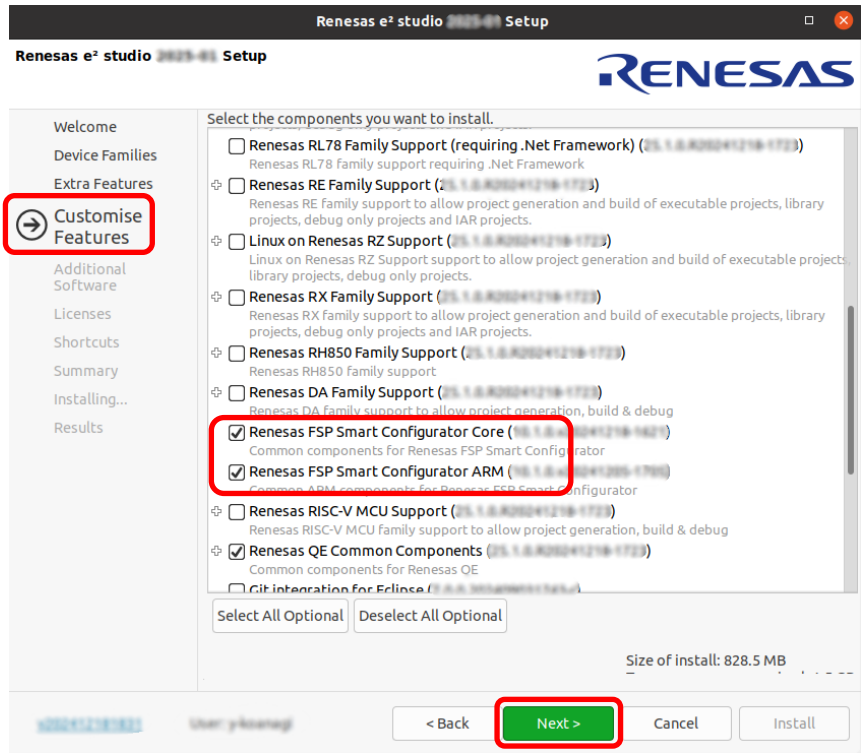


Figure 6.22 Installation of e² studio – Features

7. "Additional Software"
   Select additional software (i.e., compilers, utilities…) to be installed.
   Add the following items if they are not checked, and click [Next] to continue:

   - GCC ARM A-Profile (AArch64 bare-metal) 10.3.2021.07

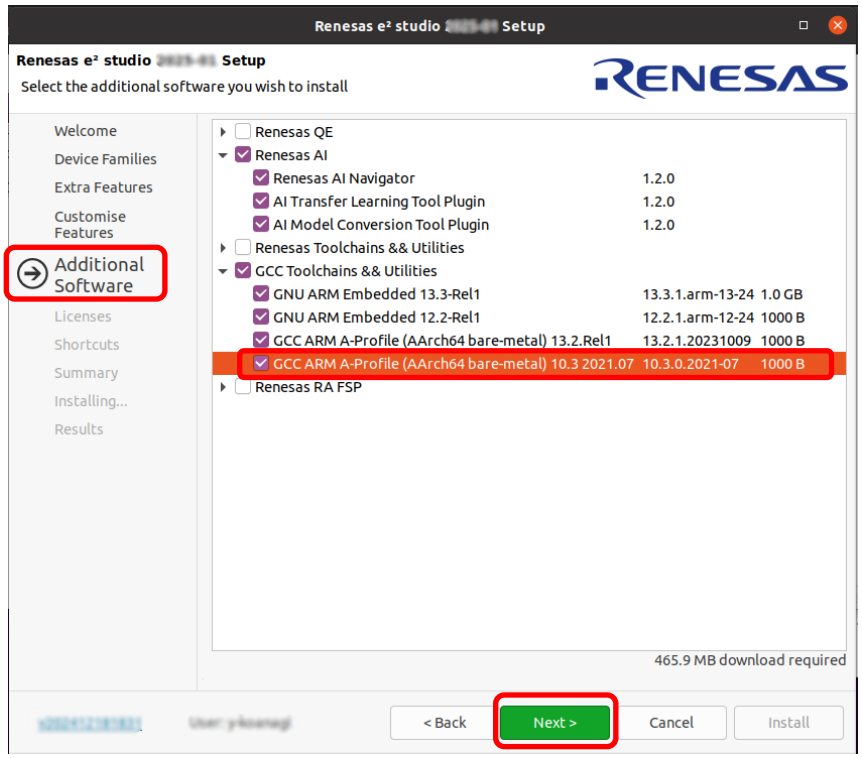


Figure 6.23 Installation of e² studio – Additional Software

8. "License Agreement"
   Please read and accept the software license agreement, then click the [Next] button. Note that acceptance of the license agreement is mandatory; without it, the installation process cannot proceed.

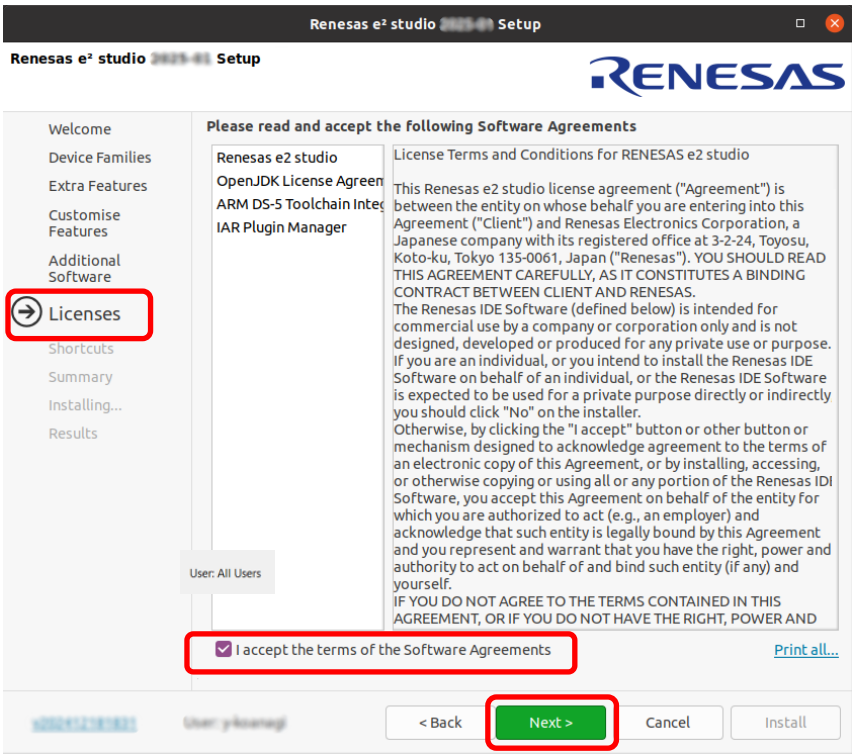


Figure 6.24 Installation of e² studio – Licenses

9. "Shortcuts"
   Check [In the Application launcher] to register for the launcher and click [Next] button to continue.

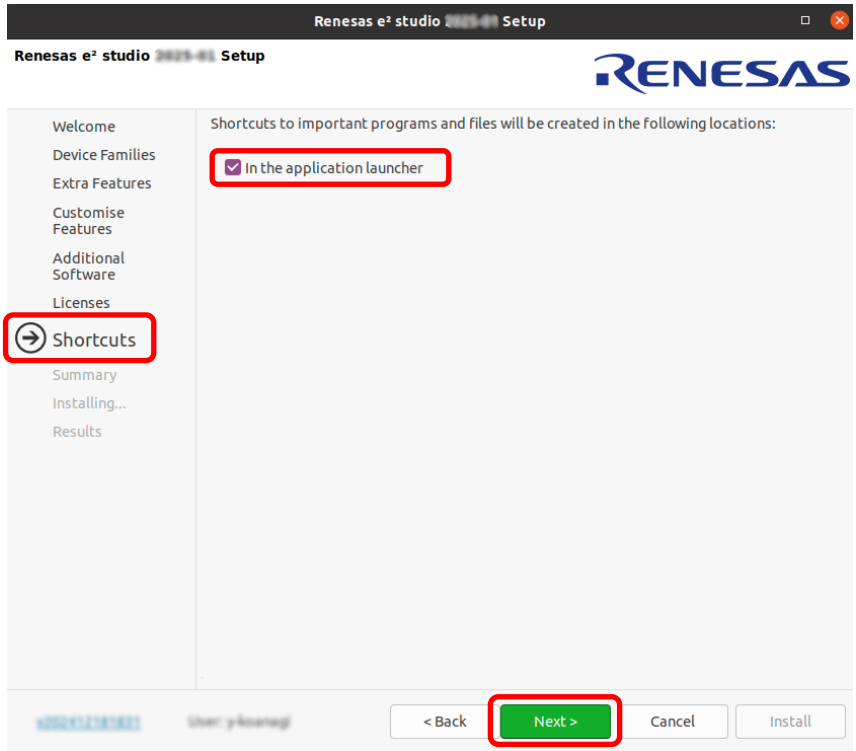


Figure 6.25 Installation of e² studio – Shortcuts

10."Summary"

Components list to be installed is shown.

Please confirm the contents and click the [Install] button to install the Renesas e² studio IDE.

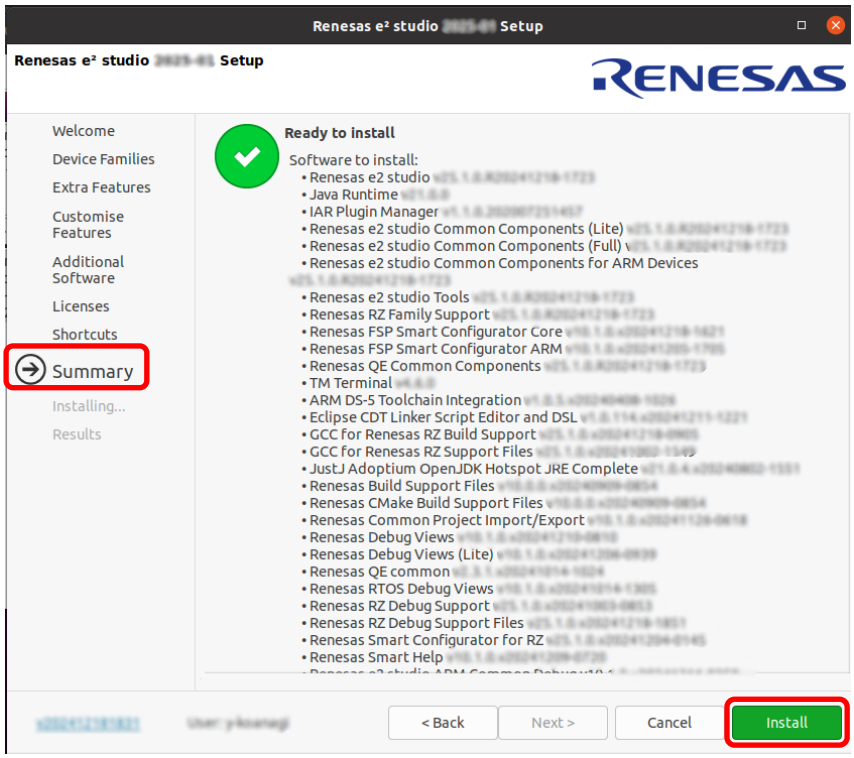


Figure 6.26 Installation of e² studio – Summary

11."Installing"

The installation is performed. Depending on selected items of additional software, new dialog prompts may appear during the installation process.

12."Results"

Click the [OK] button to complete the installation.

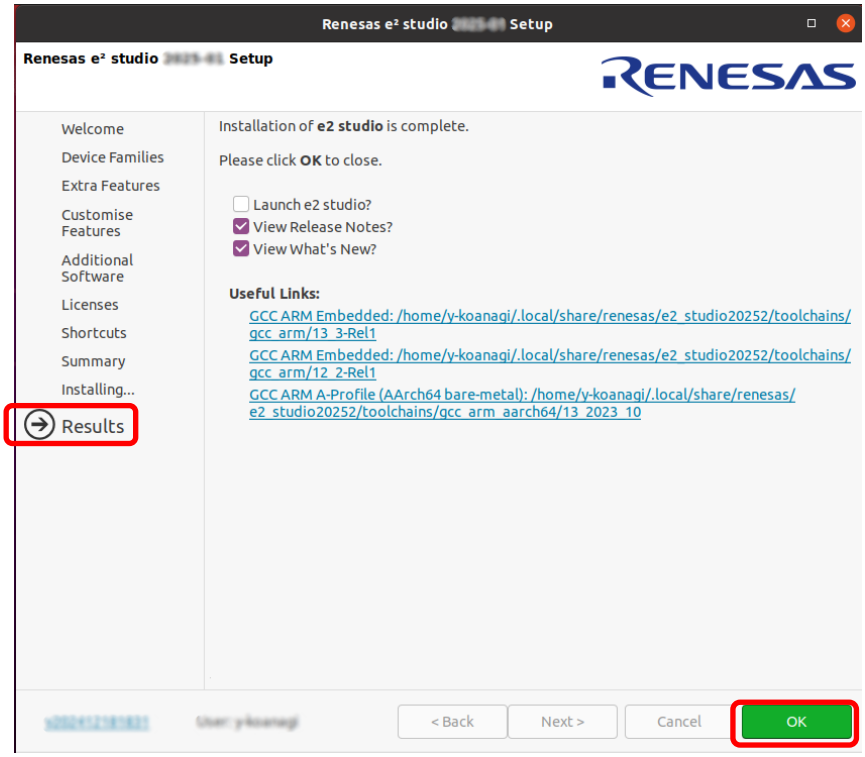


Figure 6.27 Results Page

6.2.1.2 Installation of RZ/A FSP packs (ubuntu 22.04 LTS)

Install the FSP packs for use in the RZ/A3 IPL environment.

1. Exit e² studio
2. Extract FSP Packs.
3. Copy the extracted FSP Packs (named "internal") to the e² studio installation folder as shown below:
   "~/.local/share/Renesas/e2_studio/internal/projectgen/rz_fsp/rza/packs"

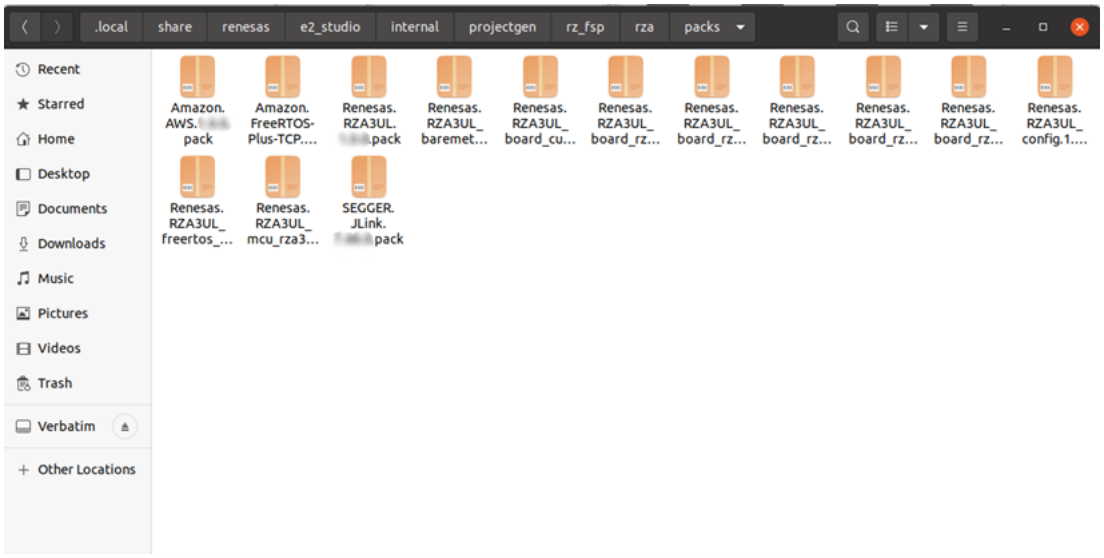


Figure 6.28 Installed packs folder (Ubuntu 22.04 LTS)

4. Please re-launch e² studio
   Then packs will be reflected in the development environment.

6.2.2 Additional Software Installation

You must install some additional software to build the IPL in ubuntu 22.04 LTS.

1. Execute the following command to install libncurses5.

```
$ sudo apt-get install libncurses5 libncurses5:i386
```

2. Download the Segger J-Link driver file from the "Linux" -> "64-bit DEB installer" link on the following website.

   https://www.segger.com/downloads/jlink/

3. Execute the following command to install driver file.
   The command is an example when the name of the driver file is "JLink_Linux_V810a_x86_64 .deb ".
   Change the argument according to the file name.

```
$ sudo dpkg -i JLink_Linux_v810a_x86_64.deb
```

   If you get unmet dependencies error, please do the following.

```
$ sudo apt-get -f install
$ sudo dpkg -i JLink_Linux_v810a_x86_64.deb
```

4. If your PC does not have libpython3.10 installed, install it.

```
$ sudo apt-get install libpython3.10
```

7. IPL build environment construction procedure

This chapter describes the procedure for building the IPL build environment for RZ/A3UL.

The procedure is the same for both Windows 10 version and Ubuntu 22.04 LTS version.

1. Get the IPL source code from the GitHub page shown below.
   This document shows an example when the obtained source code is saved in "C:\work\RZA3UL_IPL".
   (Please notice that the build might fail if the folder's path is too long.)
   renesas/rza-initial-program-loader (github.com)

2. Launch e² studio.
   Click [Browse] and specify "C:\work" for the workspace on e² studio launcher

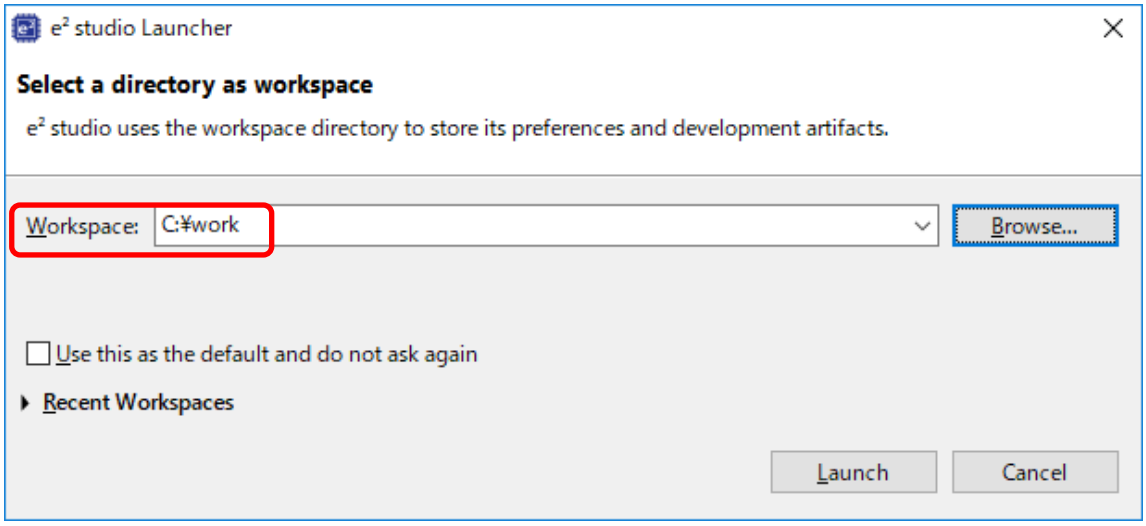


Figure 7.1 Launch e² studio (1)

3. In the Logging/Reporting dialog that is displayed when e² studio is started.
   Please confirm the descriptions, put a check to description that match your privacy policy, and click [Apply].

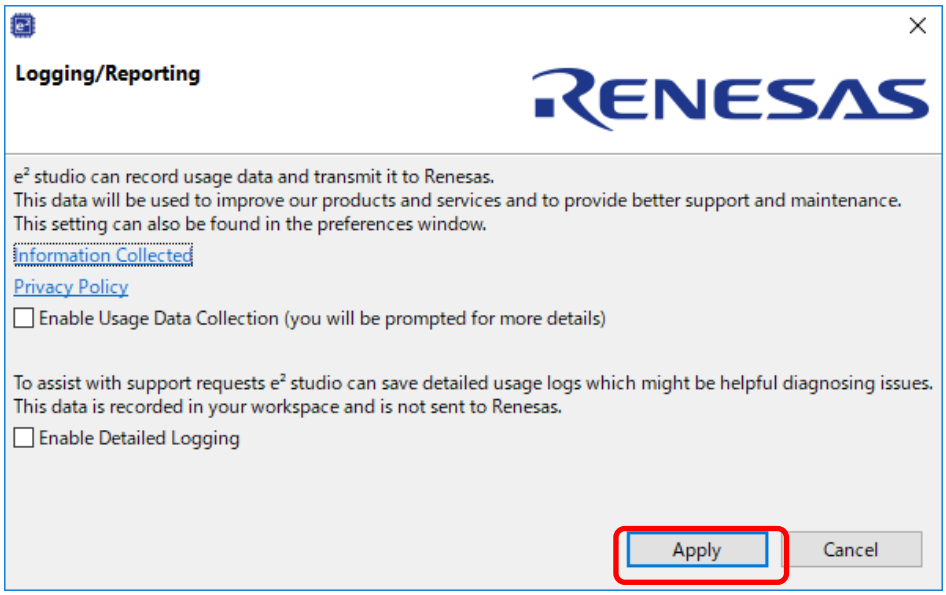


Figure 7.2 Launch e² studio (2)

4. When e² studio starts, click [Hide] to bring up the main window.

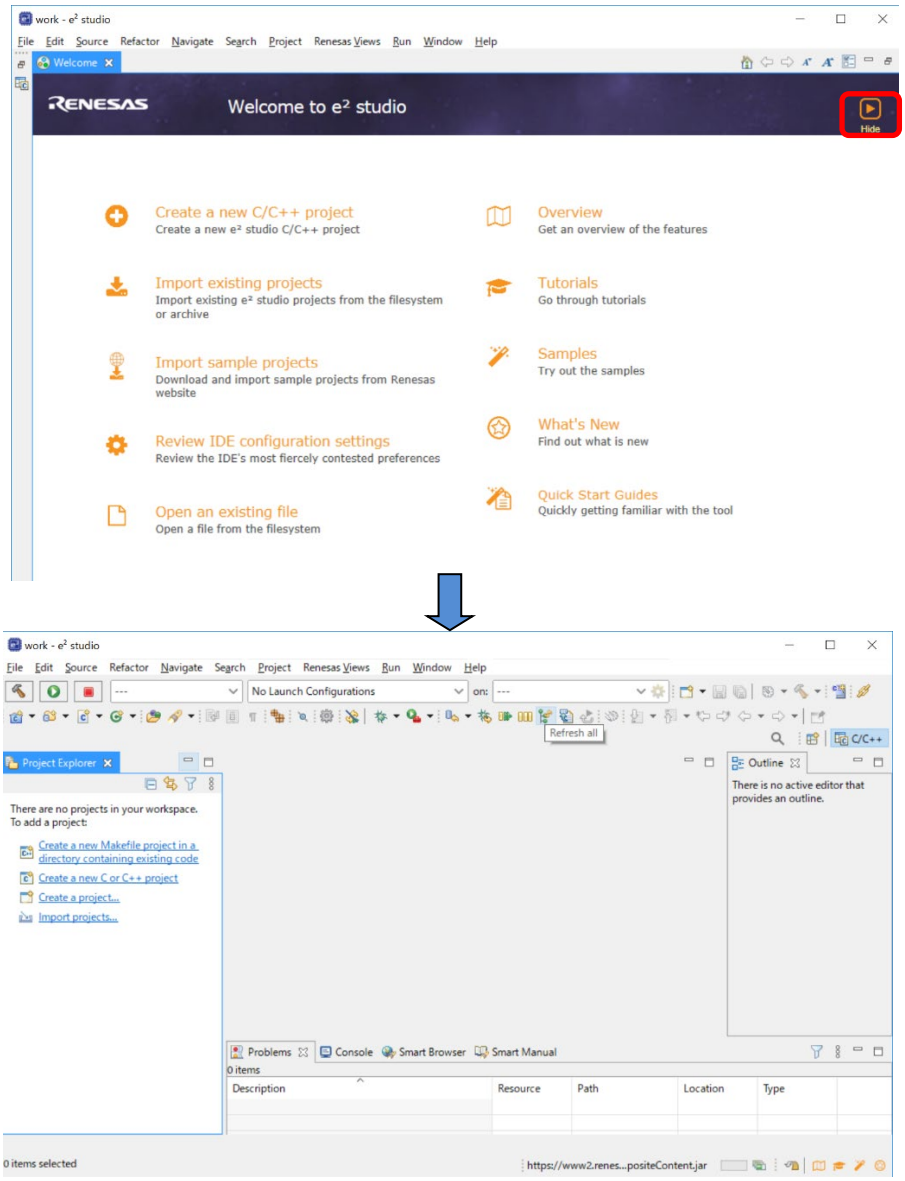


Figure 7.3 Launch e² studio (3)

5. Click [File] -> [New] -> [Makefile Project with Existing Code] in main window.

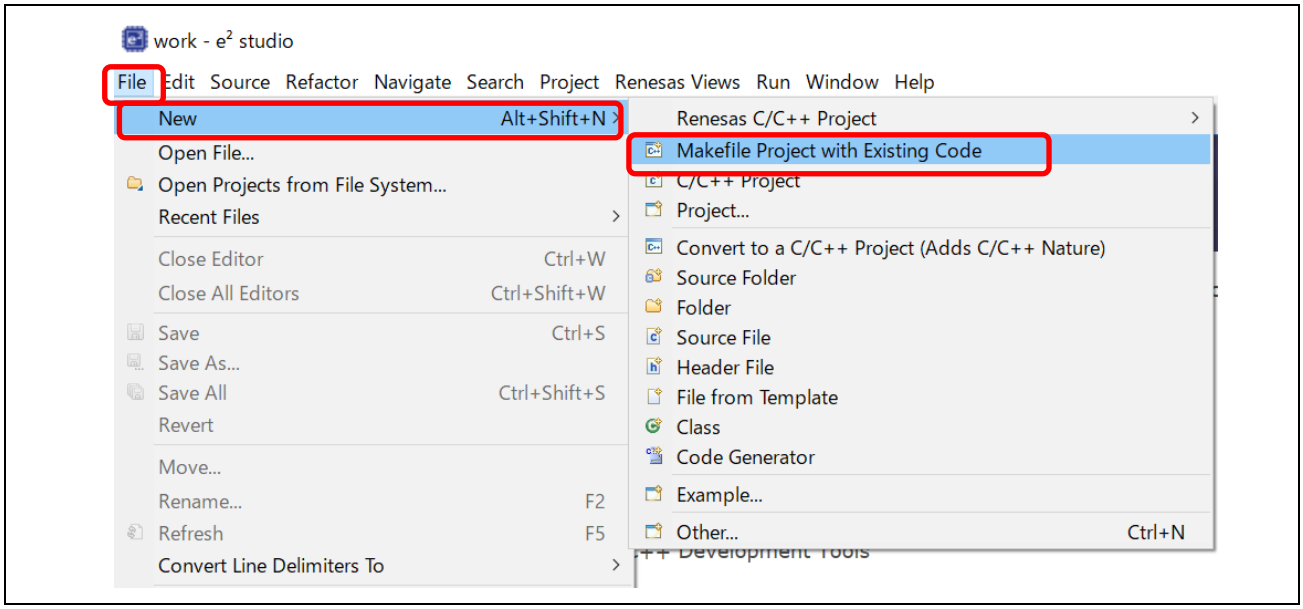


Figure 7.4 Create project (1)

6. Click Browse and choose "C:\work\RZA3UL_IPL", the folder where you unzipped the IPL.
   Then the folder path will be registered in the Existing Code Location field.
7. Select [GCC for Renesas RZ (A-Profile AArch64 bare-metal)] in the Toolchain for Indexer Settings and click [Finish].

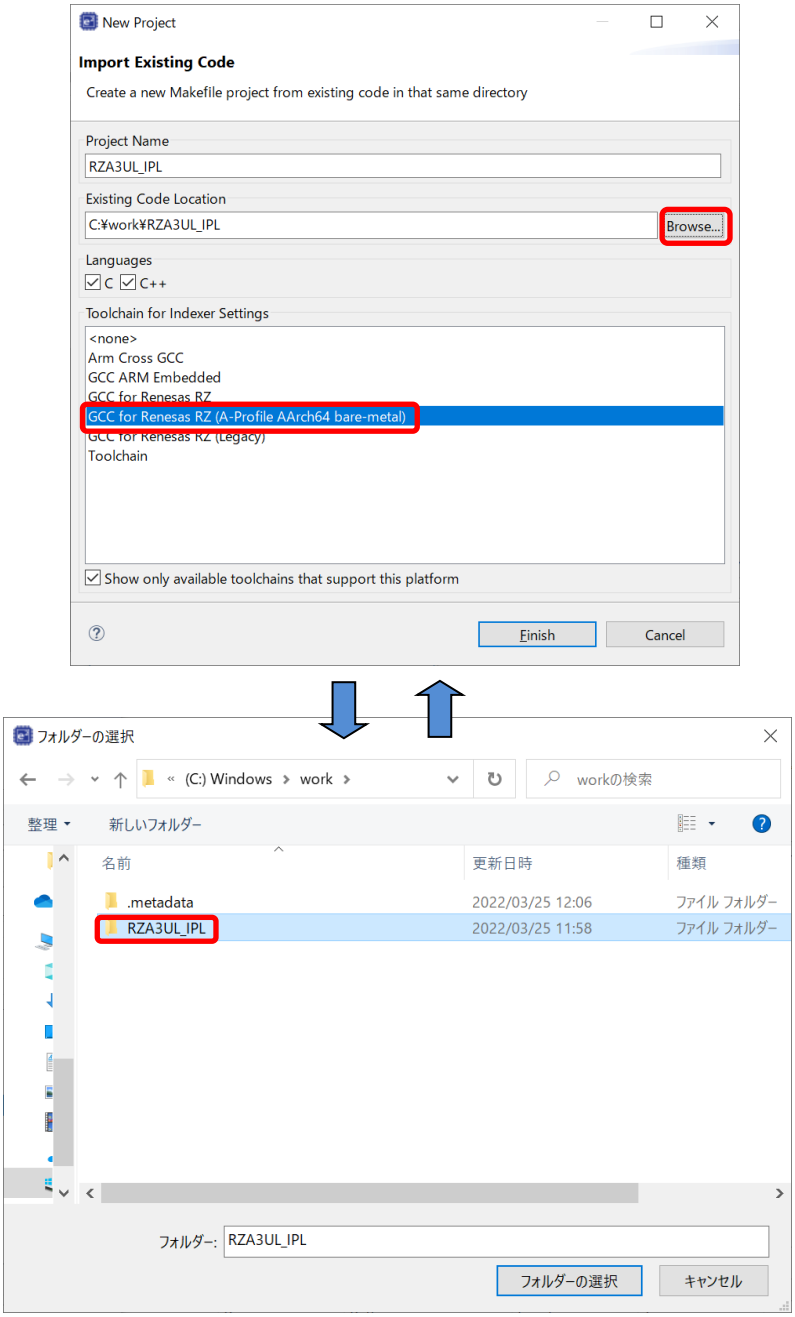


Figure 7.5 Create project (2)

8. Right-click the [RZA3UL_IPL] project generated in Project Explorer and click [Properties].

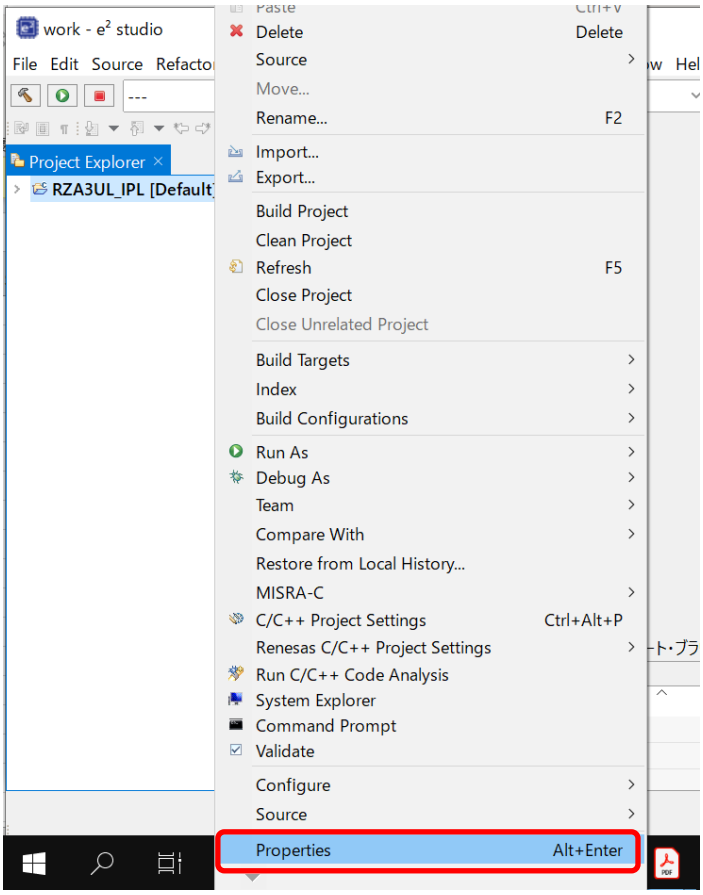


Figure 7.6 Create project (3)

9. Open [C / C ++ Build] -> [Tool Chain Editor] in Properties for [RZA3UL_IPL] and enter the build command.
   Select [GNU Make Builder] in "Current Builder", then click [Apply and Close].

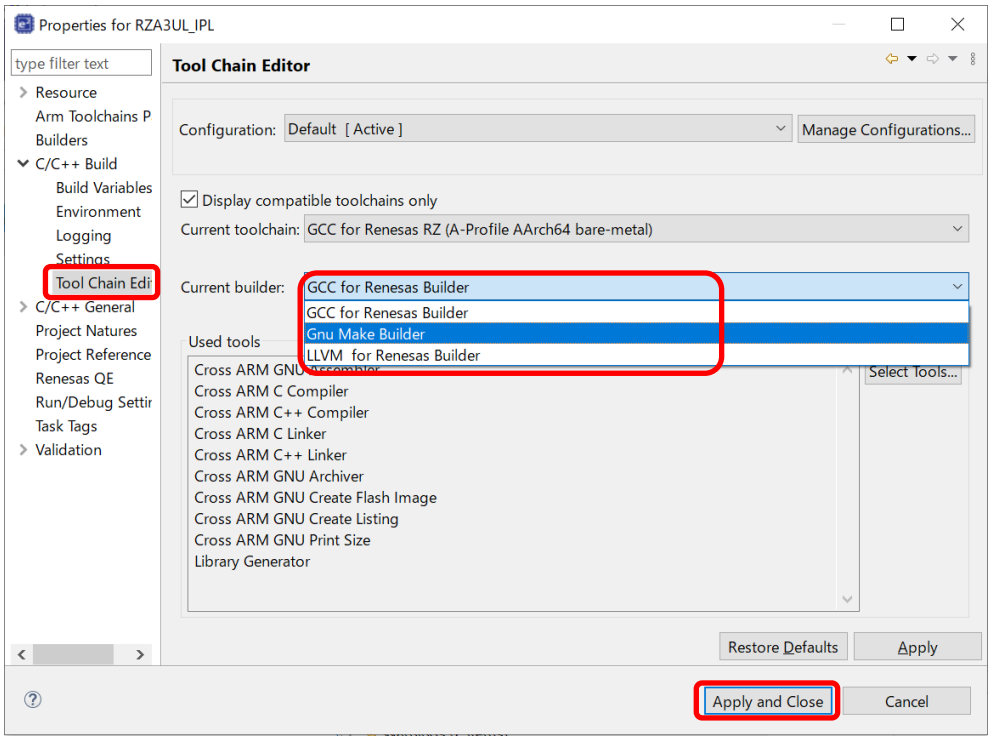


Figure 7.7 Create project (4)

10. Open [C/C ++ Build] -> [Behavior] in Properties for RZA3UL_IPL and enter the build command.
    Check "Use custom build arguments" and set the make option "PLAT=a3ul BOARD=a3ul_smarc_octal" in the "Build arguments" field, so build and clean will execute with the same make option.
    If you want to use different make options for build and clean, set the make options in the "Build" and "Clean" fields respectively.
    Then click [Apply].

Notes: 1. Options can be added and changed.
They are explained in the Table 7.1.
2. This description is the setting when the same make option is specified for Build and Clean.
If you want to use different make options for build and clean, set the make options in the "Build" and "Clean" fields respectively.

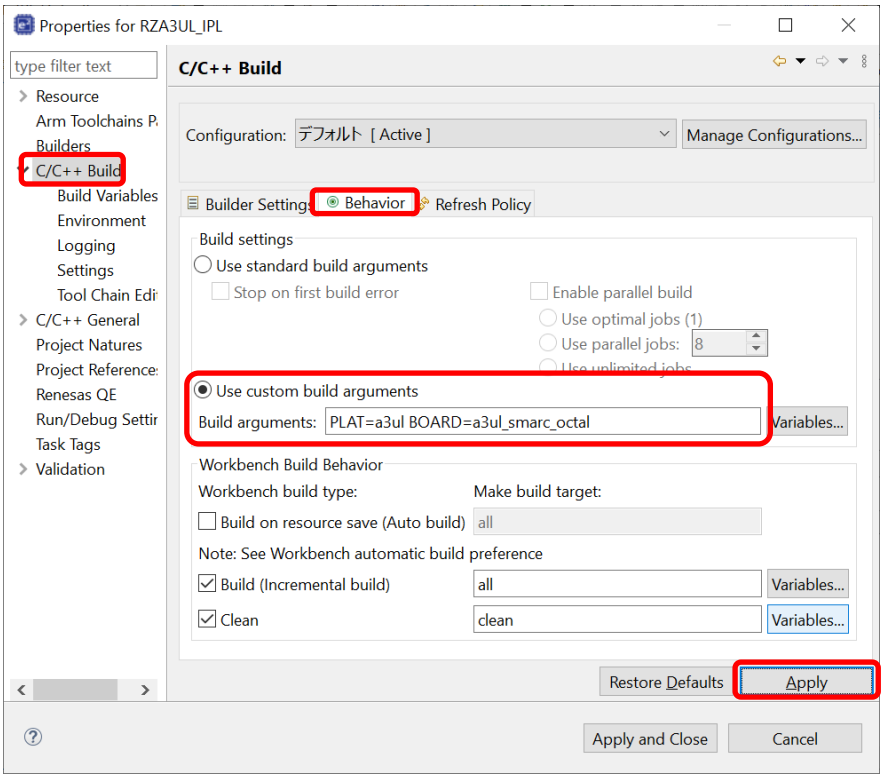


Figure 7.8 Build Command

Table 7.1 Make Option

Name	Explanation	Setting Example
PLAT	Set the platform This option cannot be omitted	a3ul: for RZ/A3UL This option can't be changed.
BOARD	Set the board This option cannot be omitted	a3ul_smarc_octal: RZ/A3UL-Evaluation-Board-Kit (Octal-SPI version)
DEBUG	Select debug/release build	1: debug build 0: release build This option can be omitted. In that case a release build is executed.
FSP_BASE	Indicates the start address of the application header	0x20020000 If you do not set this option, 0x20020000 is automatically set as the default value.

11. Open [C / C ++ Build]-> [Environment] in Properties for [RZA3UL_IPL] and enter the Environment variables.
    When you click [Add], the new variable window will appear.
    Set the environment variables of the combinations shown in the Table 7.2 (Windows10) or Table 7.3 (Ubuntu 22.04 LTS) in the Name and Value fields, then click [Apply].

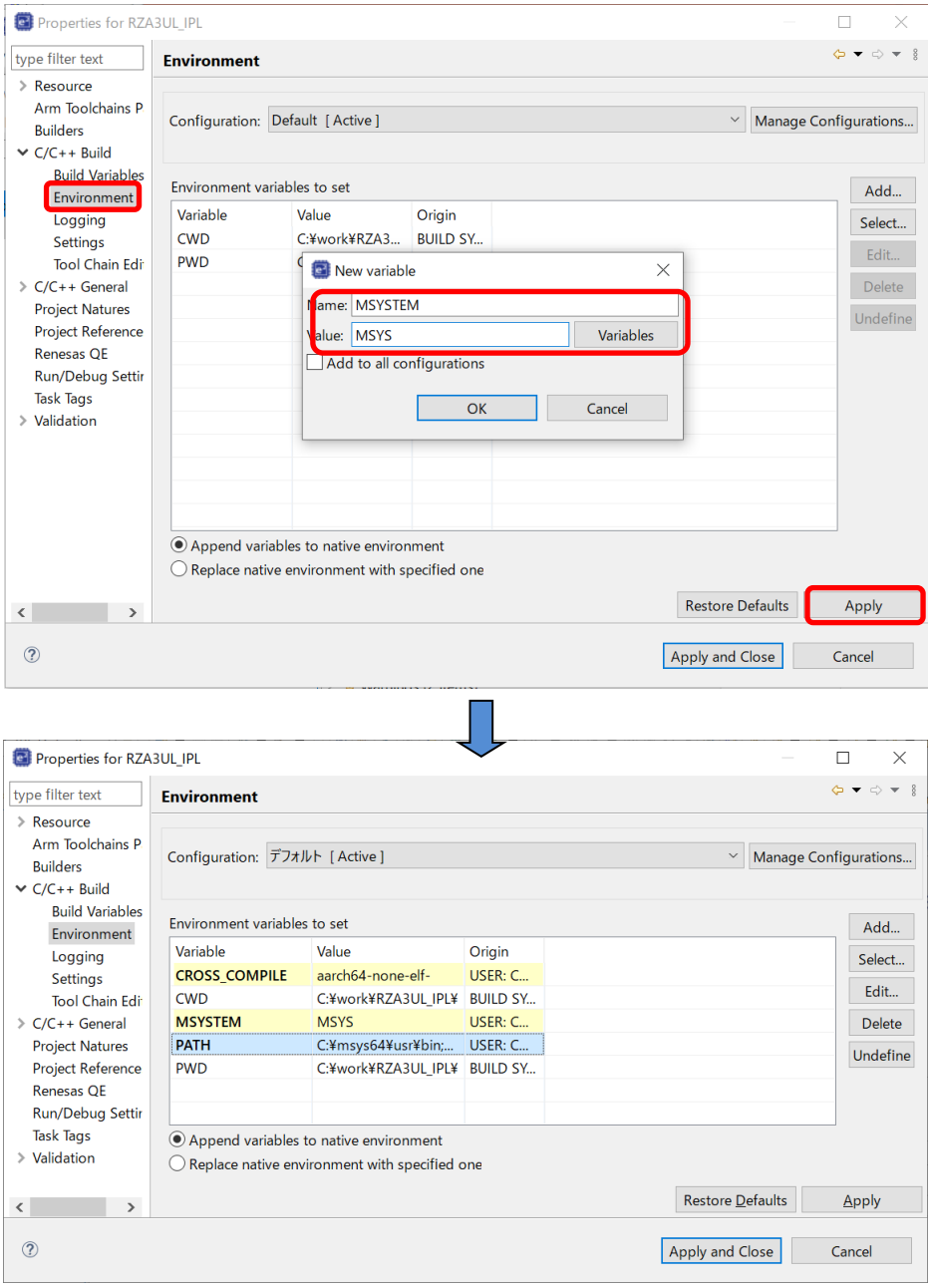


Figure 7.9 Build Environment

Table 7.2 Environment Variables (Windows10 PC)

Name	Value
CROSS_COMPILE	aarch64-none-elf-
MSYSTEM	MSYS
PATH	C:\msys64\usr\bin;C:\Renesas\e2studio\toolchains\gcc_arm_aarch64\10_2021_07\gcc-arm-10.3-2021.07-mingw-w64-i686-aarch64-none-elf\bin (Notes1, 2)

Notes: 1. Chane value to match the MSYS2 installation folder.
2. Toolchain will be installed under the e² studio installation folder.
Please set the path to that folder.

Table 7.3 Environment Variables (Ubuntu 22.04 LTS PC)

Name	Value
CROSS_COMPILE	aarch64-none-elf-
PATH	/home/username/.local/share/renesas/e2_studio/toolchains/gcc_arm_aarch64/10_2021_07/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/bin (Note)

Note: Toolchain will be installed under the e² studio installation folder.
Please set the path to that folder.

12.Right-click [RZA3UL_IPL] and click [Build Project] to run the build.
If you want to clean it, click [Clean Project].

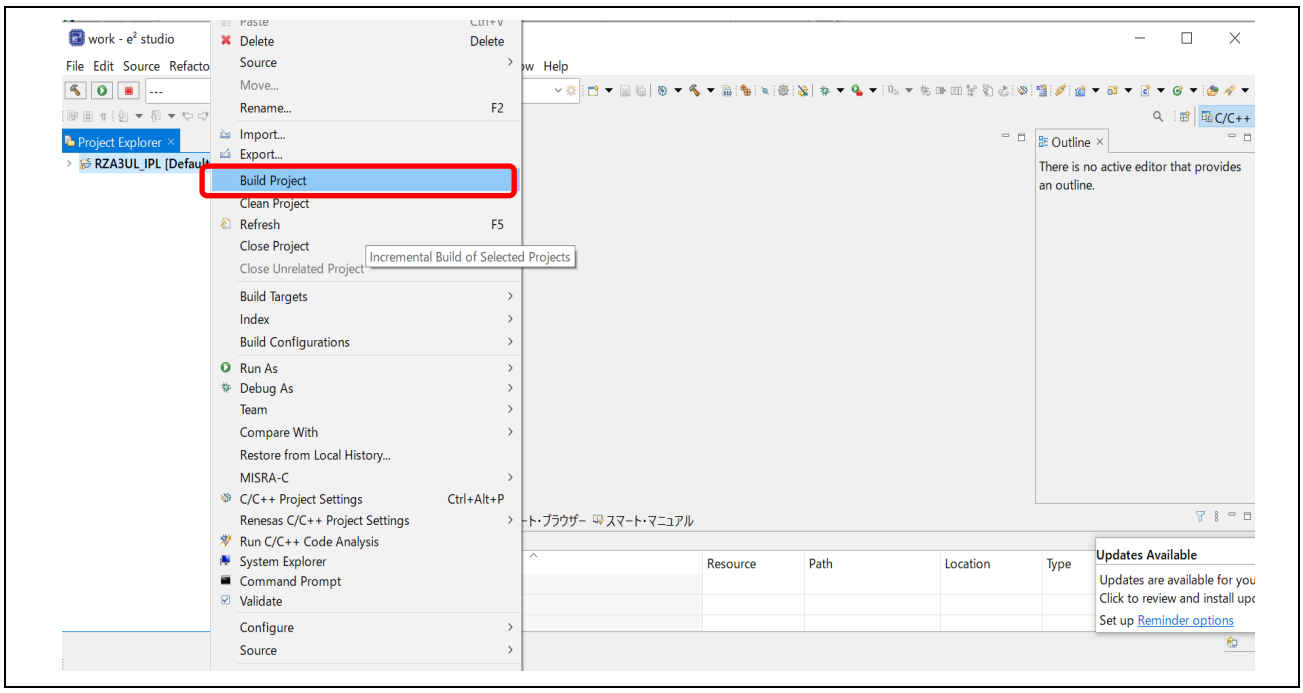


Figure 7.10 Build Environment

If the build is successful, the following files will be generated in "RZA3UL_IPL\build\a3ul\release".

rza3ul_smarc_octal_ipl.dump:
This file shows the disassembled data of the IPL code.

rza3ul_smarc_octal_ipl.map:
Map file of IPL.

rza3ul_smarc_octal_ipl.bin:
Raw Binary image of IPL.

rza3ul_smarc_octal_ipl.elf:
Executable file of IPL.
It is used to download and run the IPL to On-chip RAM in e² studio.

rza3ul_smarc_octal_ipl.srec:
Motorola S format file of IPL.
By writing to the Serial Flash Memory, automatic boot is performed when the SMARC EVK board is reset.
The IPL write destination address (H'2000_0000) is registered in this file itself.
When using the download function of e² studio, the registered address is referenced and written to the memory, so the offset address specified in e² studio is H'0000_0000.
When writing an IPL to the board using Flash Writer, specify the write destination offset address according to specification of it.

bl2.bin、rz_bl2_xspi.env、rz_bl2_xspi_def.c:
Temporary file when building

.

8. Execution Environment of IPL

8.1 Create Debug Configuration

This chapter shows how to create a Debug Configuration to run IPL for RZ/A3UL.

The procedure is the same for both Windows 10 version and Ubuntu 22.04 LTS version.

1. Right-click on RZA3UL_IPL and select [Debug AS] -> [Debug Configurations].

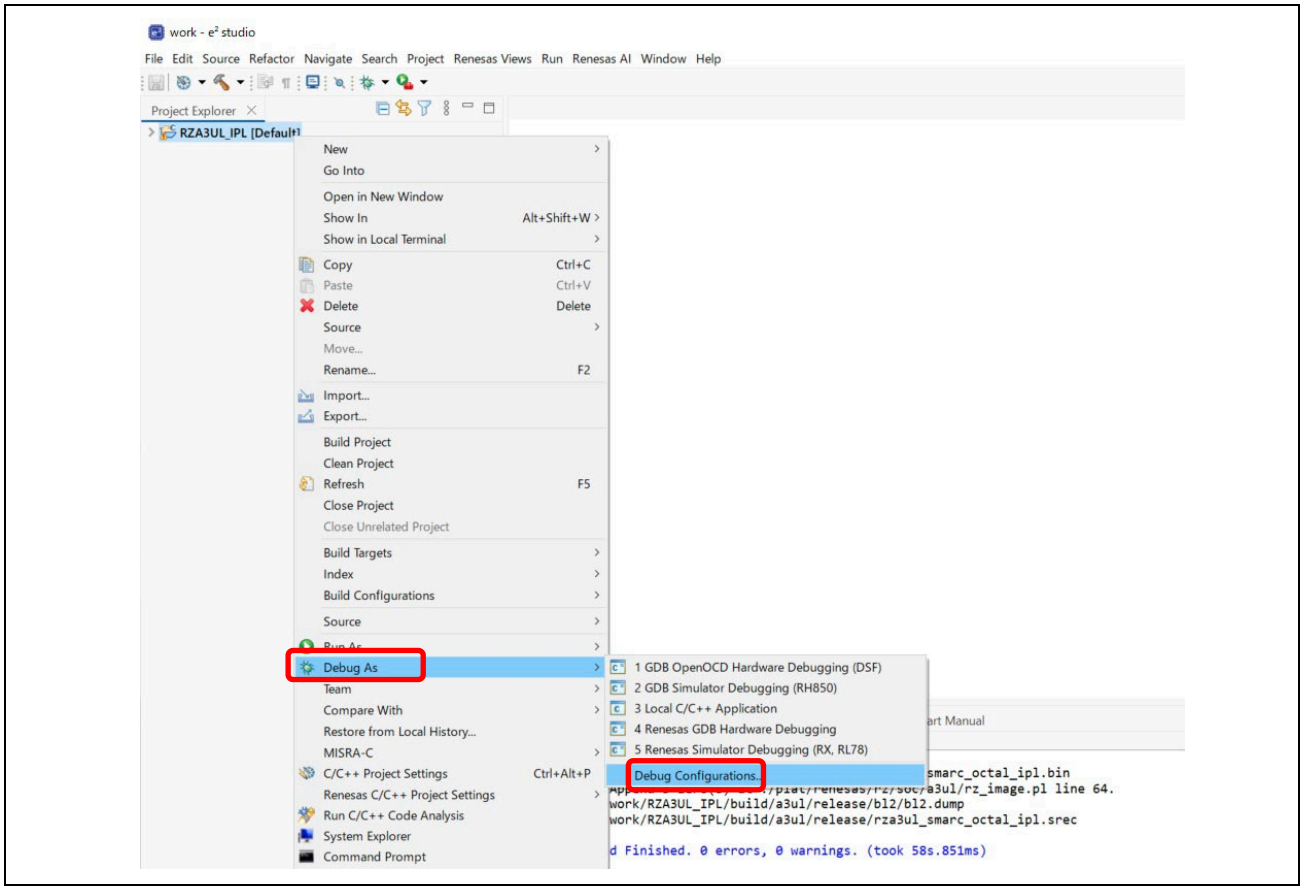
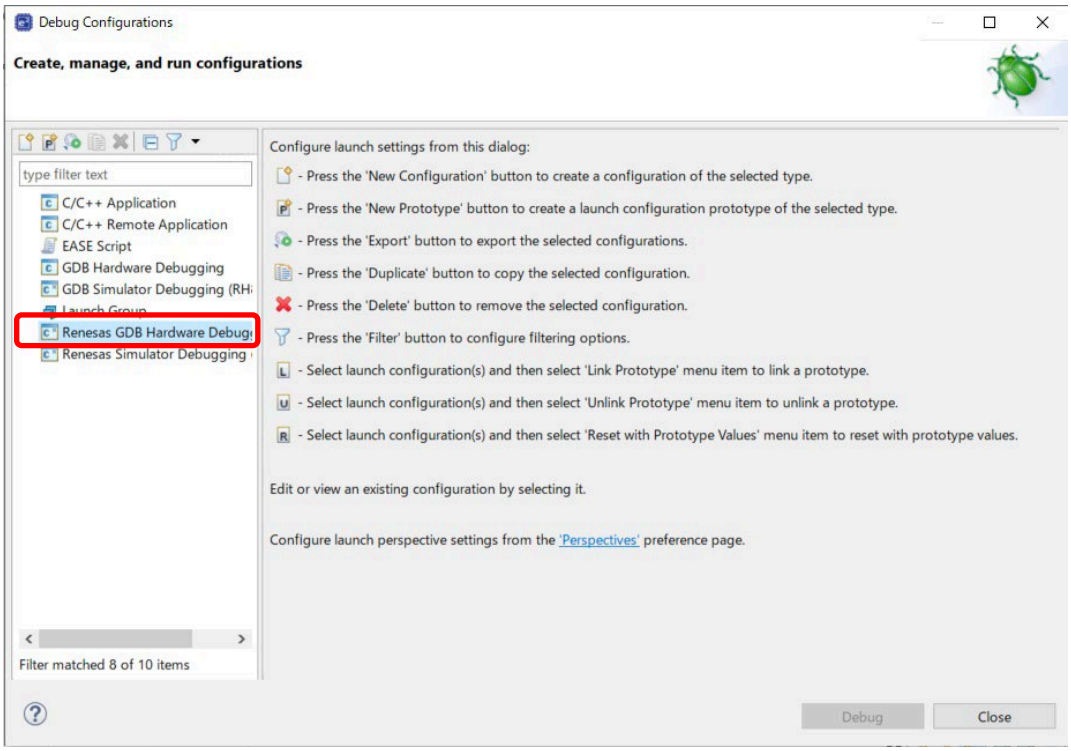


Figure 8.1 Create Debug Configuration (1)

2. In the "Debug Configurations" window, double-click [Renesas GDB Hardware Debugging] to create a Debug Configuration template for the RZA3UL_IPL.



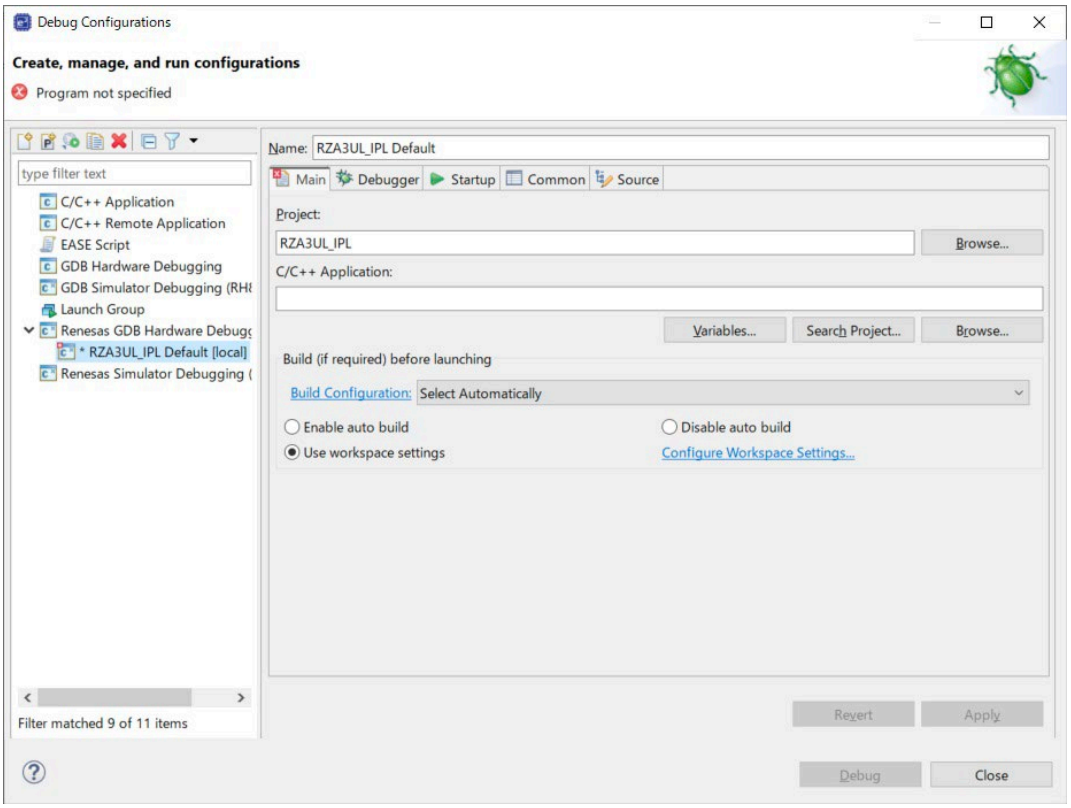


Figure 8.2 Create Debug Configuration (2)

3. Add device connection settings to the debug configuration template for RZA3UL_IPL.
   Click [Search Project] on the Main tab and select [rza3ul_smarc_octal_ipl.elf] on the Program Selection window.
   Furthermore, after selecting "Use Active" in [Build Configuration], click [Apply].

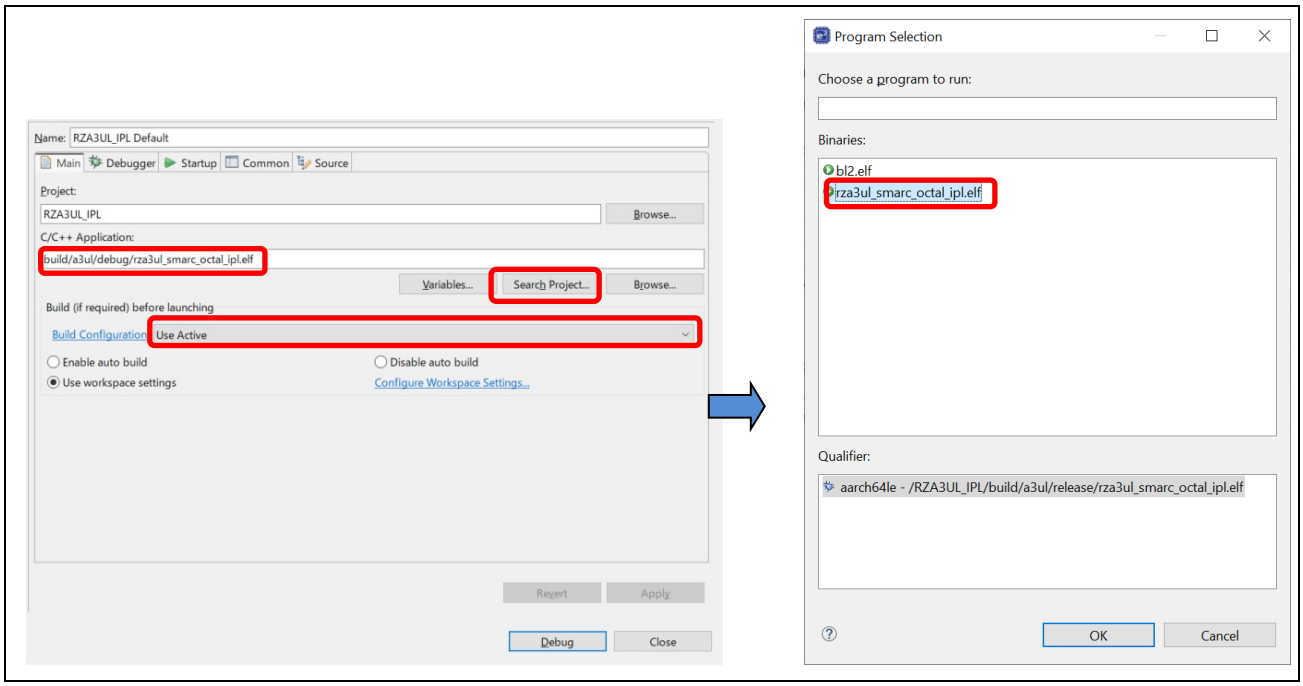


Figure 8.3 Create Debug Configuration (3)

4. Go to the [Debugger] tab.
   Select [J-Link ARM] in the Debug Hardware field.
   Click […] next to Target Device and select [RZ/A3UL] -> [R9A07G063U02GBG] on the Device Selection window.
   Also, set "aarch64-elf-gdb" in the GDB command field of the GDB settings tab.

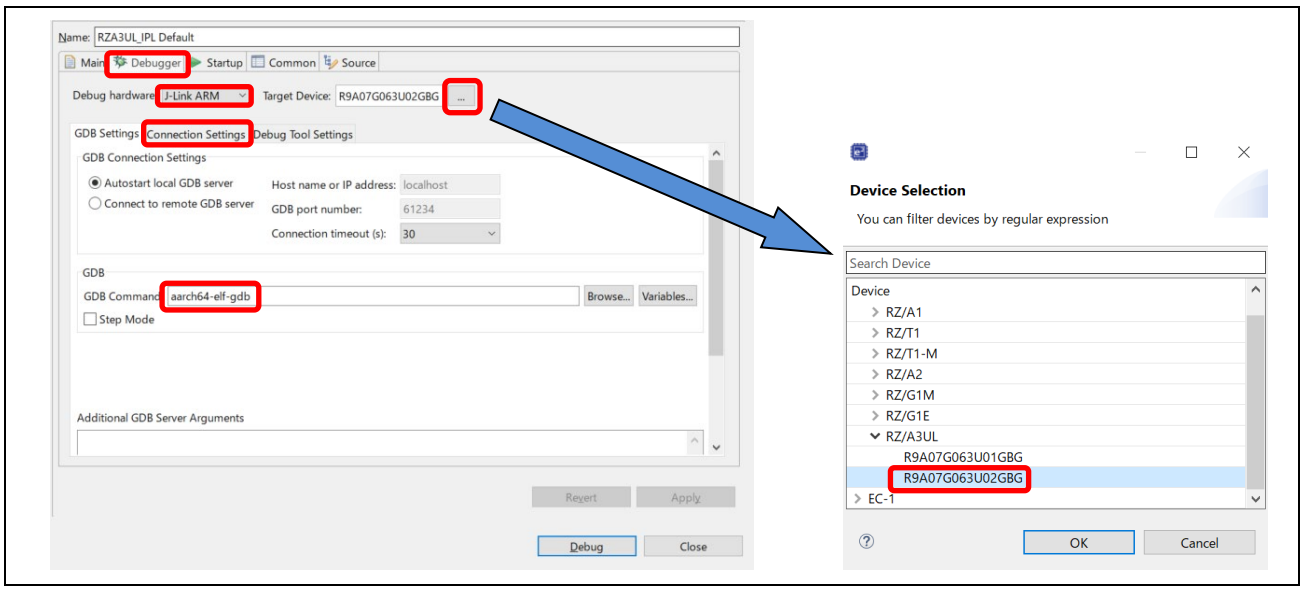


Figure 8.4 Create Debug Configuration (4)

5. Go to the "Debugger" -> "Connection Settings" tab.
   Set " Reset at the beginning of connection" to [Yes].
   If you use J-Link Lite, set SWD to [Type].
   If you use other J-Link models, please set [JTAG] or [SWD] to "Type" according to the specification.
   Then, set "Reset at the beginning of connection" to [Yes].

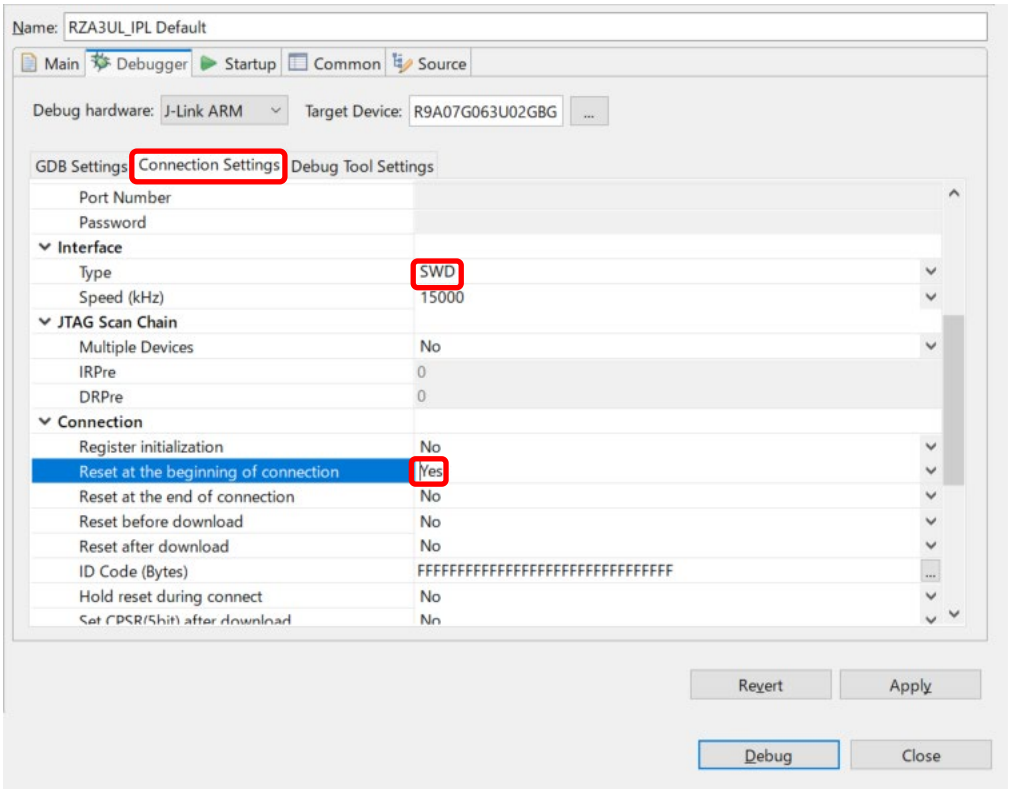


Figure 8.5 Create Debug Configuration (5)

6. Go to the "Debugger"-> "Debug Tool Settings" tab.
   Set "Reset After Reload" to [No].
   Set "Flash Bus Type" to [OctaBus] and "Flash Memory Type" to [OctaFlash].
   Set "Run Break Time Measurement" to [No].

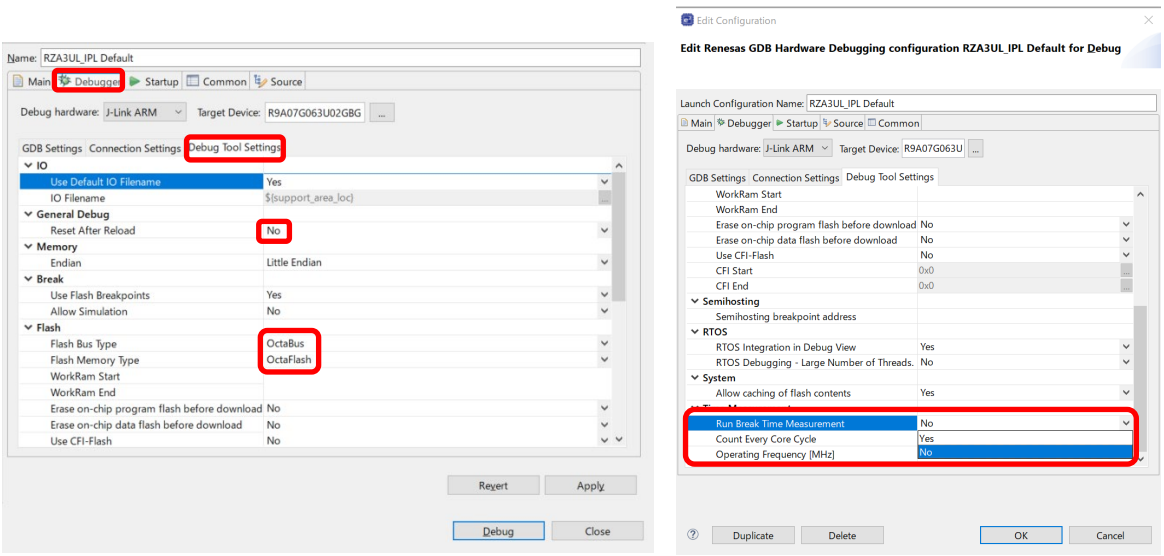


Figure 8.6 Create Debug Configuration (6)

7. Go to "Startup" tab
   Set "On connect" of "Program Binary" to [No].

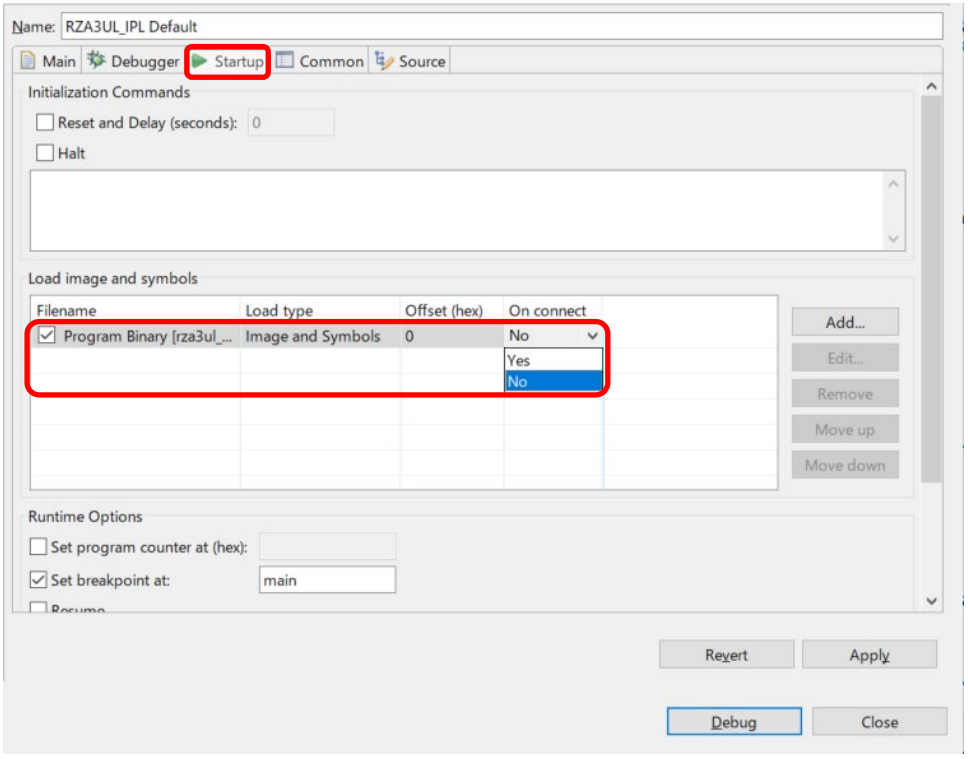


Figure 8.7 Create Debug Configuration (7)

8. Click [Add] to bring up the Add download module window.
   Next click [Workspace], select [RZA3UL_IPL\build\a3ul\release\rza3ul_smarc_octal_ipl.srec], and then click [OK] to return.
   Then, rza3ul_smarc_octal_ipl.srec is registered in Load image and symbols.

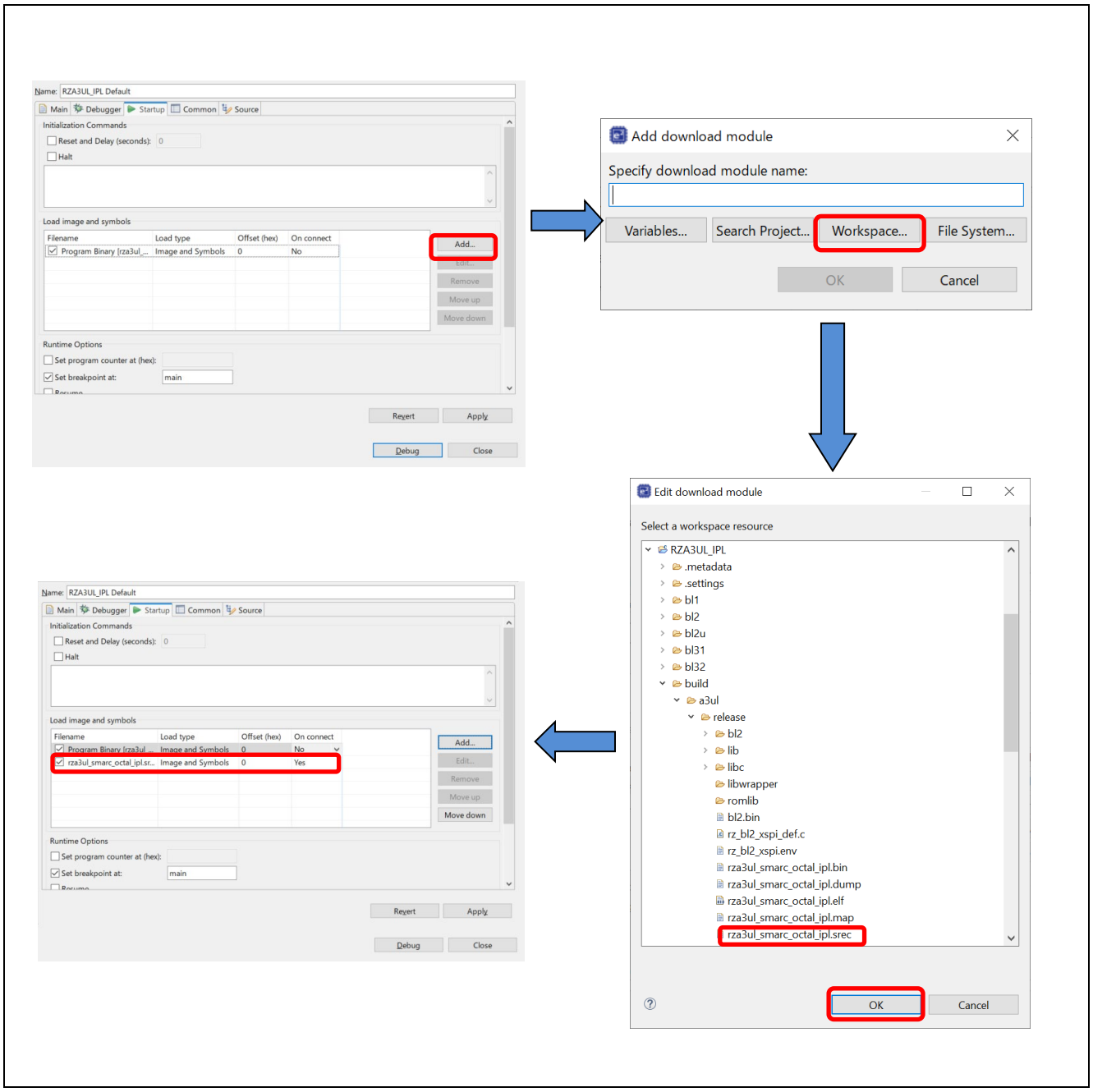


Figure 8.8 Create Debug Configuration (8)

9. Since the IPL write destination address H'2000_0000 is set in the registered rza3ul_smarc_octal_ipl.srec itself, set [0] to "Offset(hex)".
   With this setting, the IPL image will be written to H'2000_0000 at the start of debugging.
   Also, set" On connect" to [No] and uncheck "Set breakpoint at".

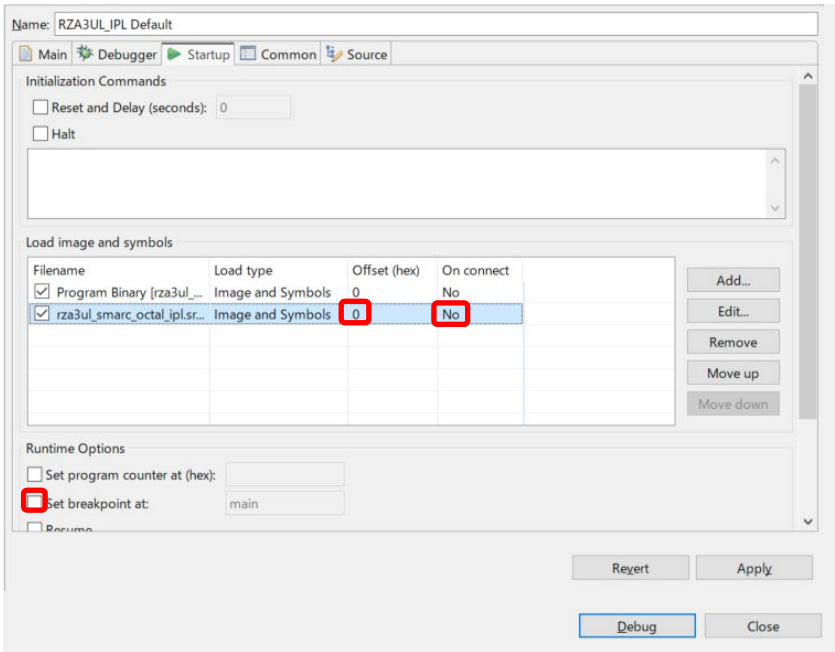


Figure 8.9 Create Debug Configuration of Octa Flash (9)

10.Go to the Common tab and set Save as to Share file:, and set the path to [\RZA3UL_IPL].
Also, uncheck "Launch in background".
After completing these settings, click [Apply] -> [Close] to complete the settings.

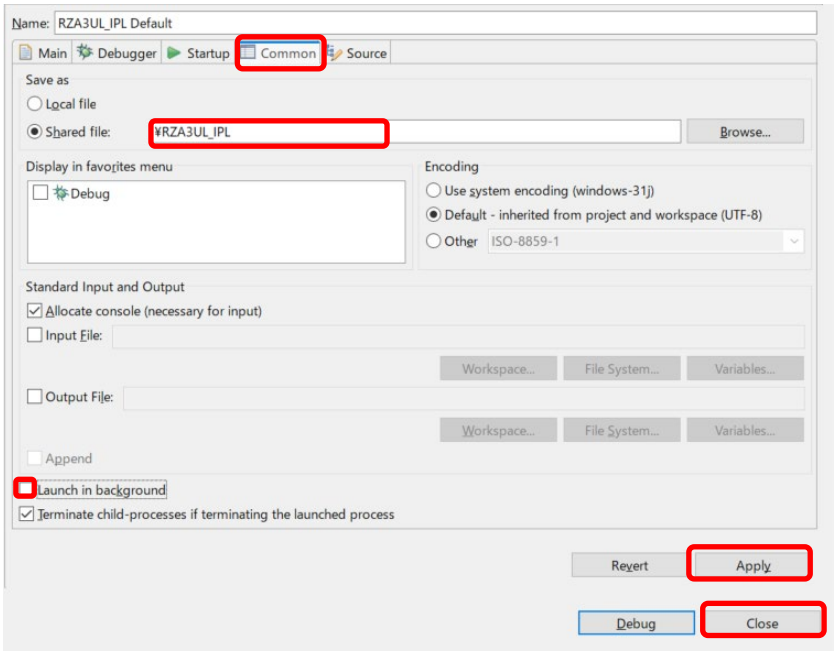


Figure 8.10 Create Debug Configuration (10)

8.2 Connection to SMARC EVK Board

Describes how to connect to the SMARC EVK board.

Figure 8.11 shows the appearance of the SMARC EVK board.

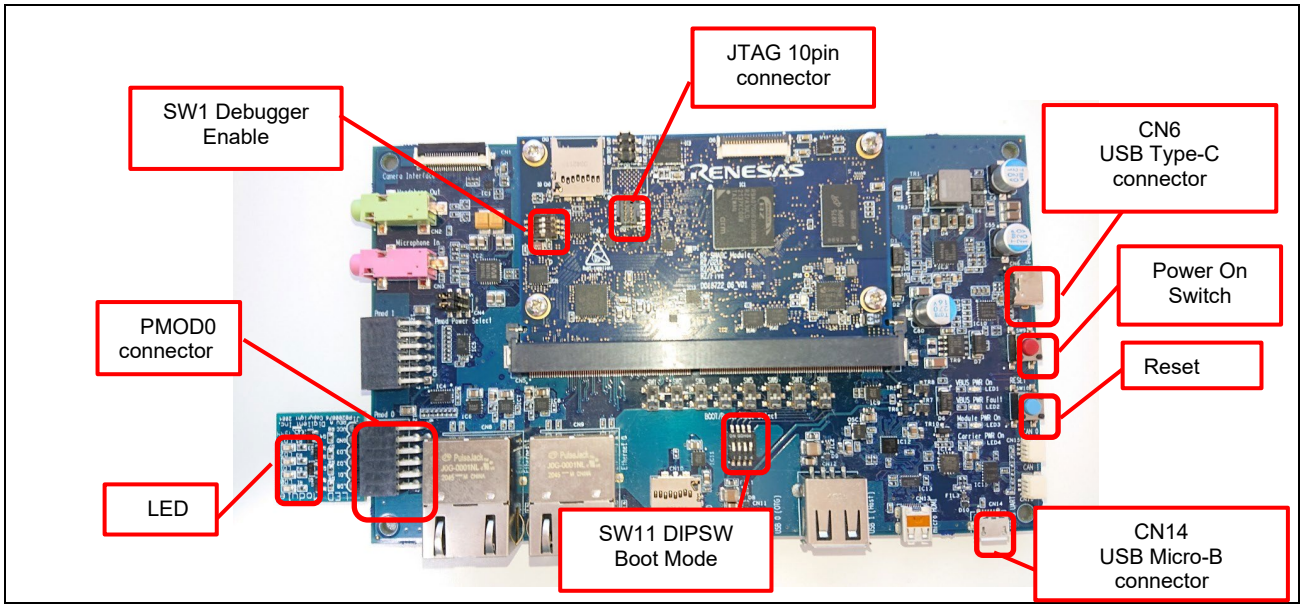


Figure 8.11 SMARC EVK Board

1. Connect the LED board to the PMOD0 connector.
   As shown in Figure 8.12, connect to terminals 7 to 12.

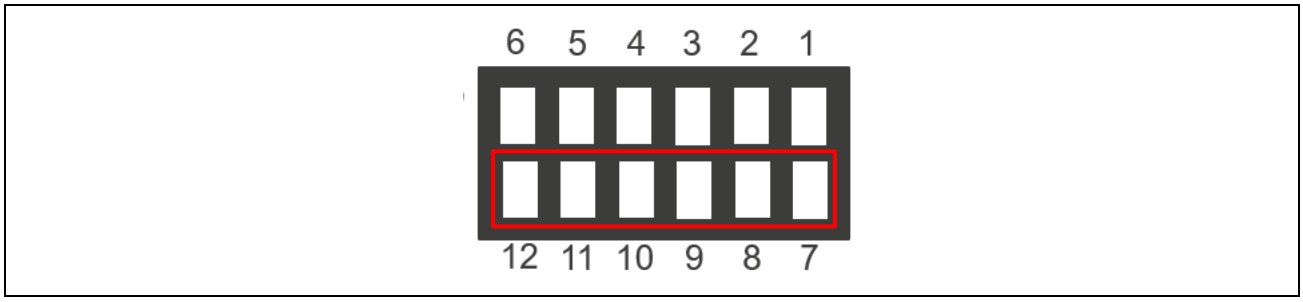


Figure 8.12 PMOD Connector

2. Connect CN14 and Windows 10 PC with a USB cable.
3. Connect J-Link Base 11.0 to the JTAG 10pin connector.
   In addition, connect J-Link Base 11.0 to the host PC with a USB cable.
4. Connect CN6 and USB-PD Power Charger.

5. Set SW1-1 to OFF as shown in Figure 8.13 to enable connectivity with the debugger.
   If you start the board without connecting to J-Link Base, set SW1-1 to ON.

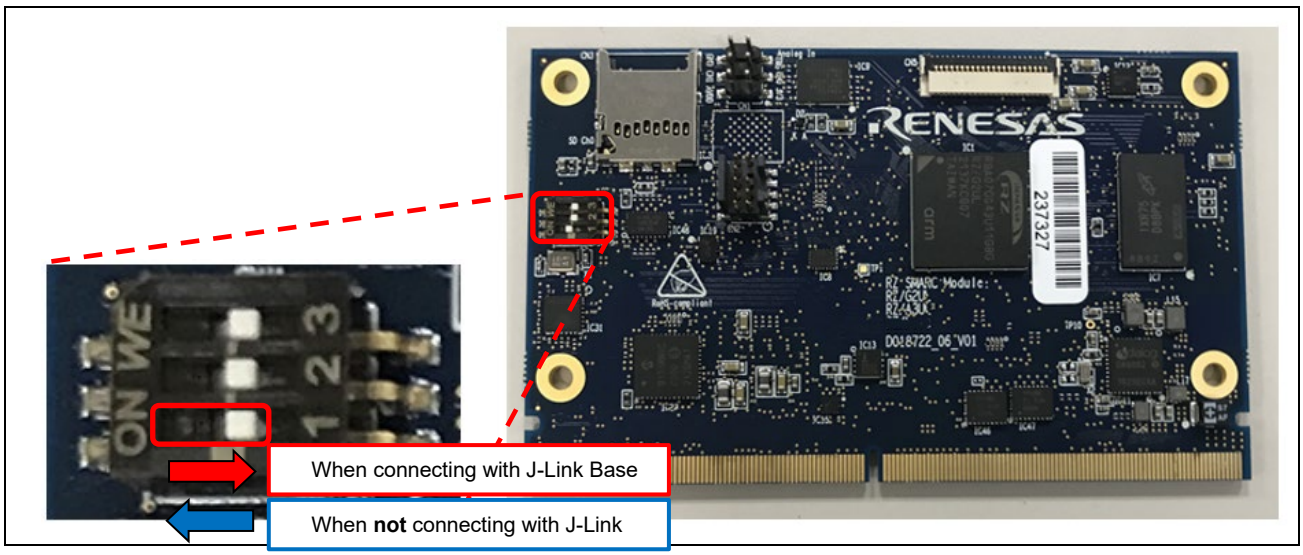


Figure 8.13 SW1 setting

6. Set Boot Mode to Boot Mode3 (1.8-V serial Flash memory) by setting SW11-1, SW11-2, SW11-3 as shown in Figure 8.14.

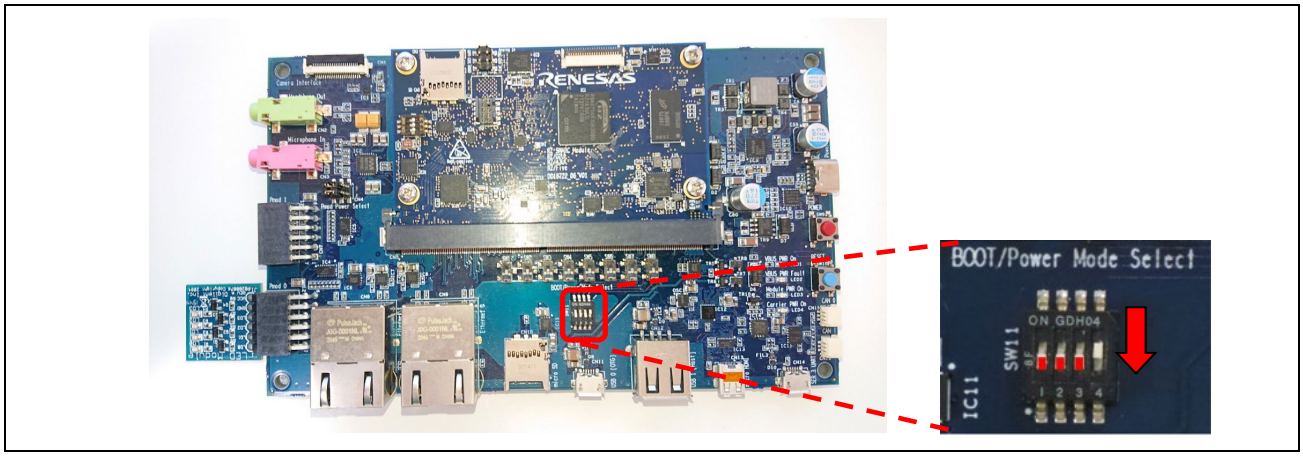


Figure 8.14 SW11 setting

8.3 Setting Serial Terminal

This section describes the procedure for setting the serial terminal to check the message output by IPL.

The procedure is the same for both Windows 10 version and Ubuntu 22.04 LTS version.

1. Set the SMARC EVK board operating mode to Boot Mode3 (1.8-V serial Flash memory) and turn on the power.
2. Launch e² studio.
3. Click [Window] -> [Show View] -> [Terminal].

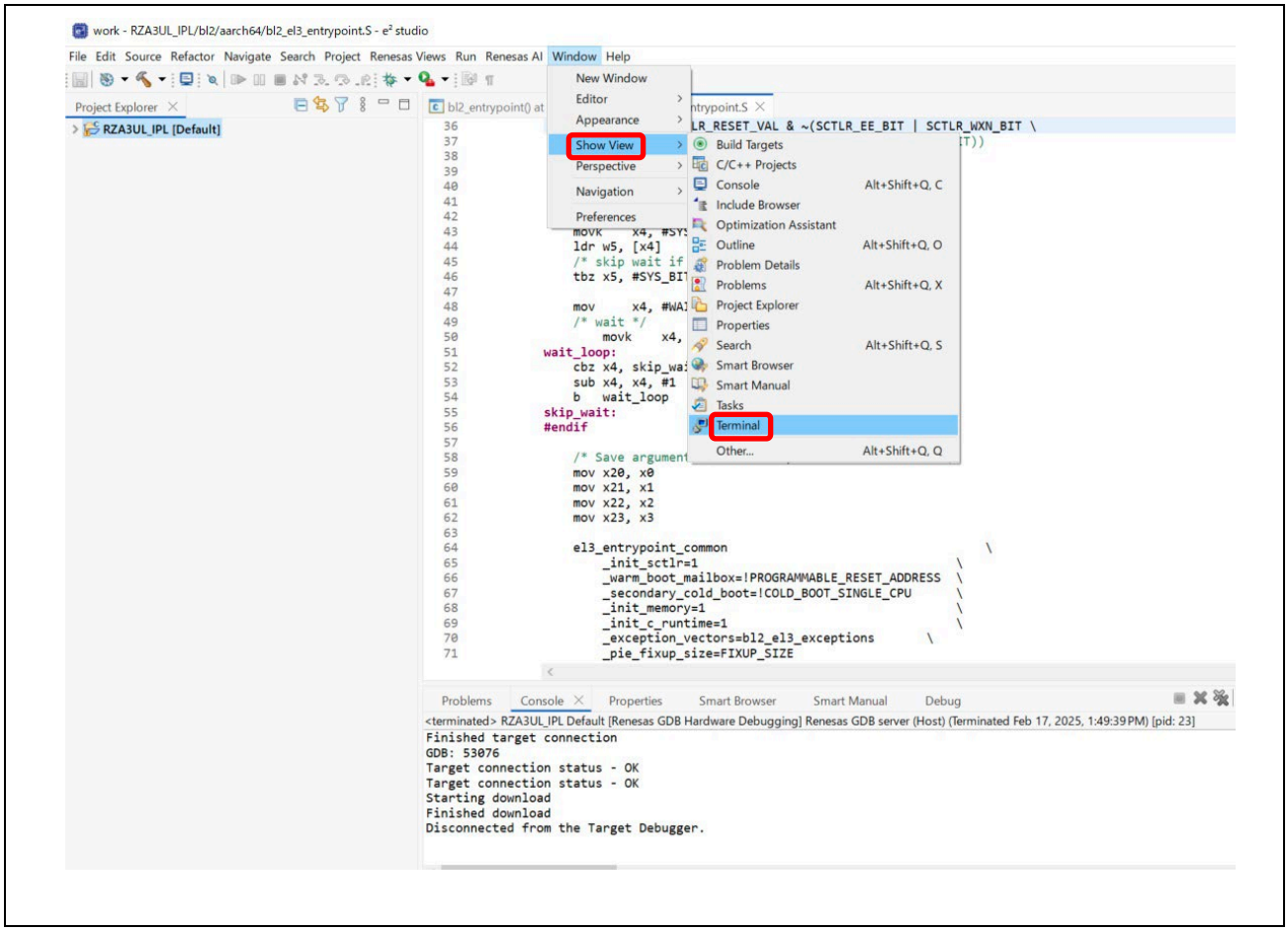


Figure 8.15 Setting Serial Terminal (1)

4. The Terminal window will open.
   Please click the mark.

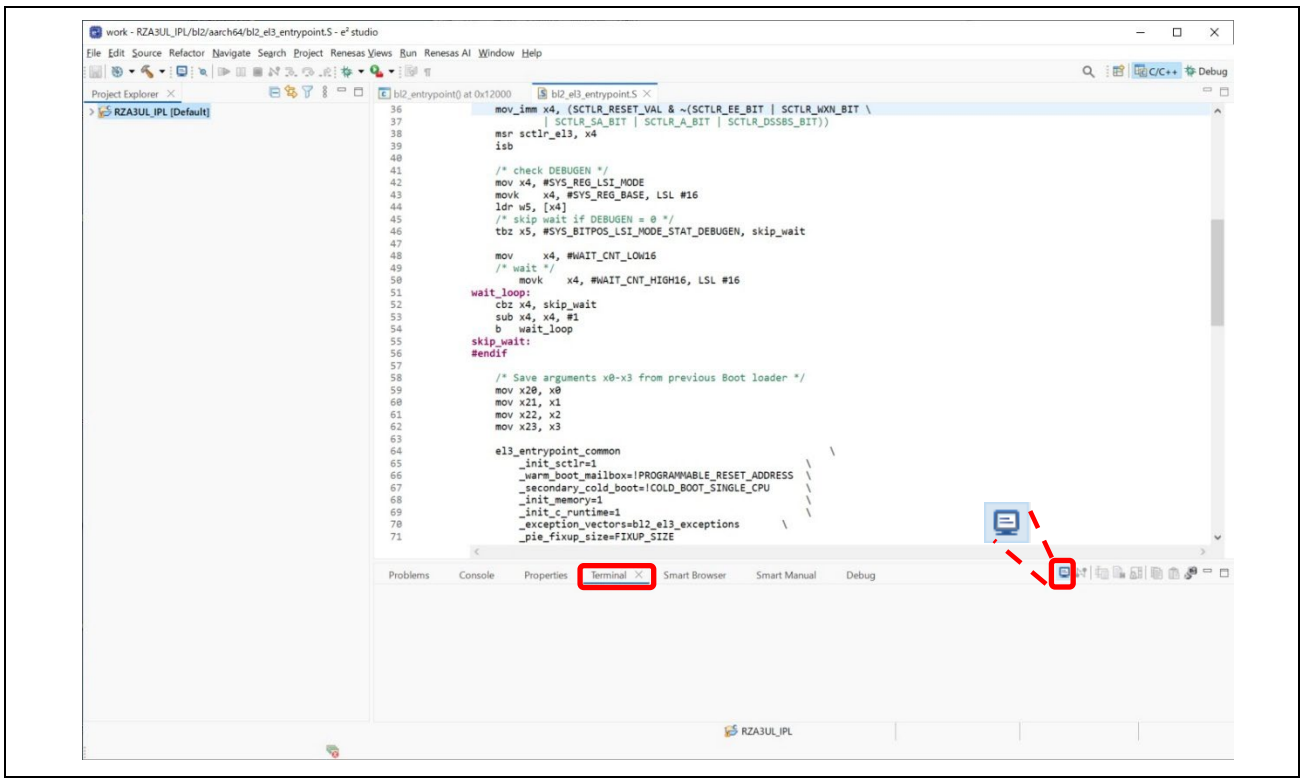


Figure 8.16 Setting Serial Terminal (2)

5. The Lunch Terminal window will bring up.
   Set the parameters as shown in Figure 8.17 on a Windows 10 PC and Figure 8.18 on an Ubuntu 22.04 LTS PC and click OK.
   "Serial port" is a PC environment-dependent port.
   Please configure the settings according to your environment.

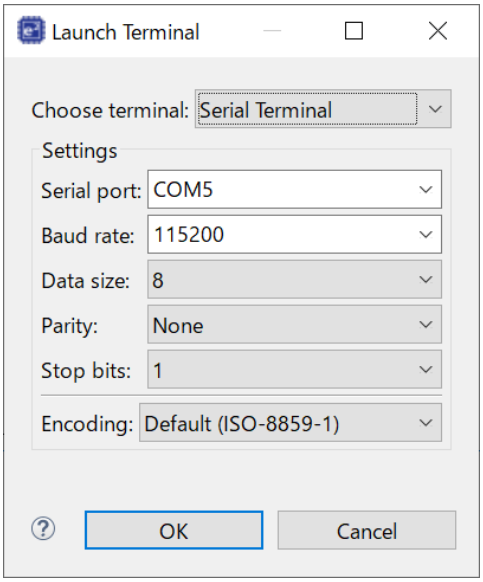


Figure 8.17 Parameter of Serial Terminal (Windows10)

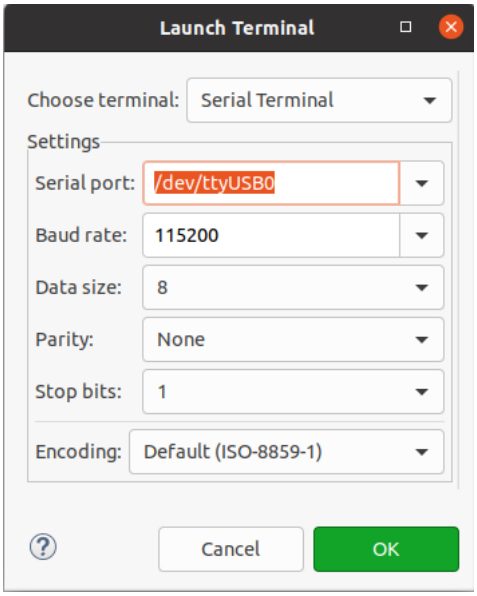


Figure 8.18 Parameter of Serial Terminal (Ubuntu 22.04 LTS)

6. Start the Serial Terminal.

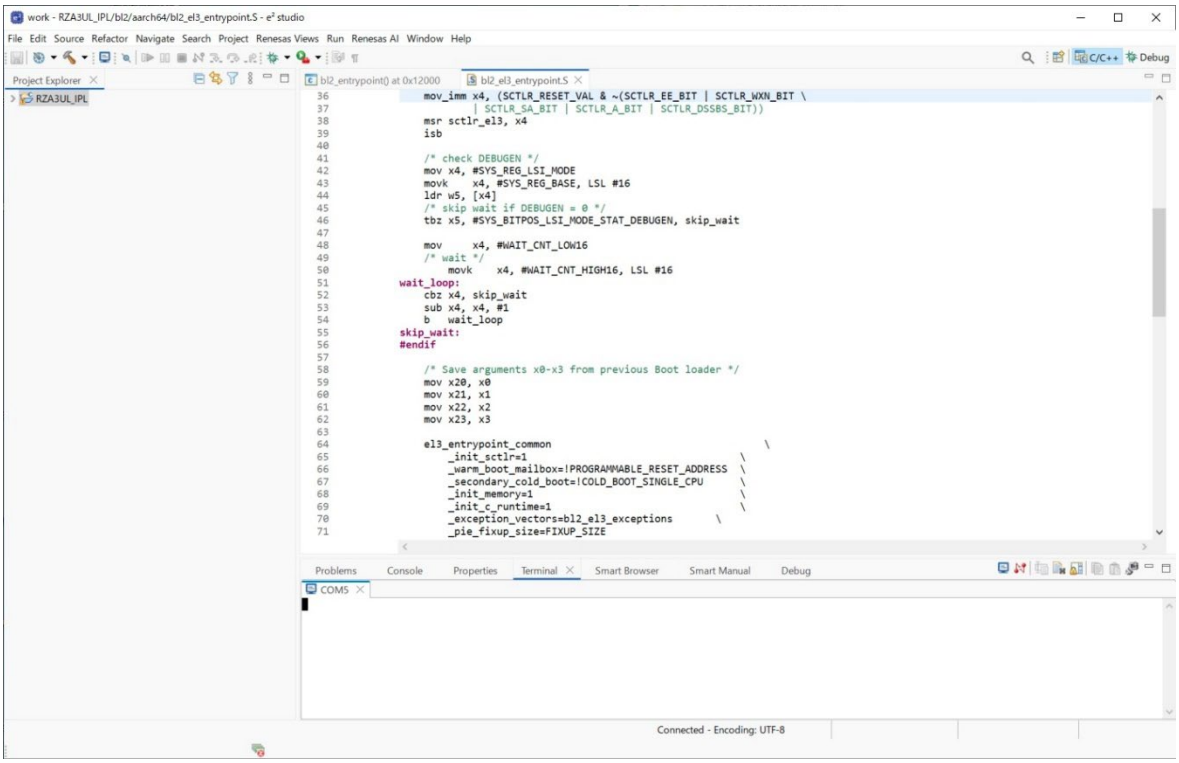


Figure 8.19 Setting Serial Terminal (3)

8.4 Execution procedure of IPL by Octa Flash boot

The procedure for writing the IPL to the Octa Flash memory and executing the IPL is shown below.

1. As the FSP application must be written to the Octa Flash memory to confirm the boot operation by IPL, if the FSP application is not written to the Octa Flash memory, write it.
   For the write procedure, refer to 9.2 Procedure for writing an FSP application to Octa Flash memory.
2 Set the SMARC EVK board operating mode to Boot Mode3 (1.8-V serial Flash memory) and turn on the power.
3. Right-click on RZA3UL_IPL and select [Debug AS] -> [Debug Configurations].
   Then, configuration generated in 8.1 Create Debug Configuration (hereinafter referred to as RZA3UL_IPL Default configuration) to bring up the setting window.

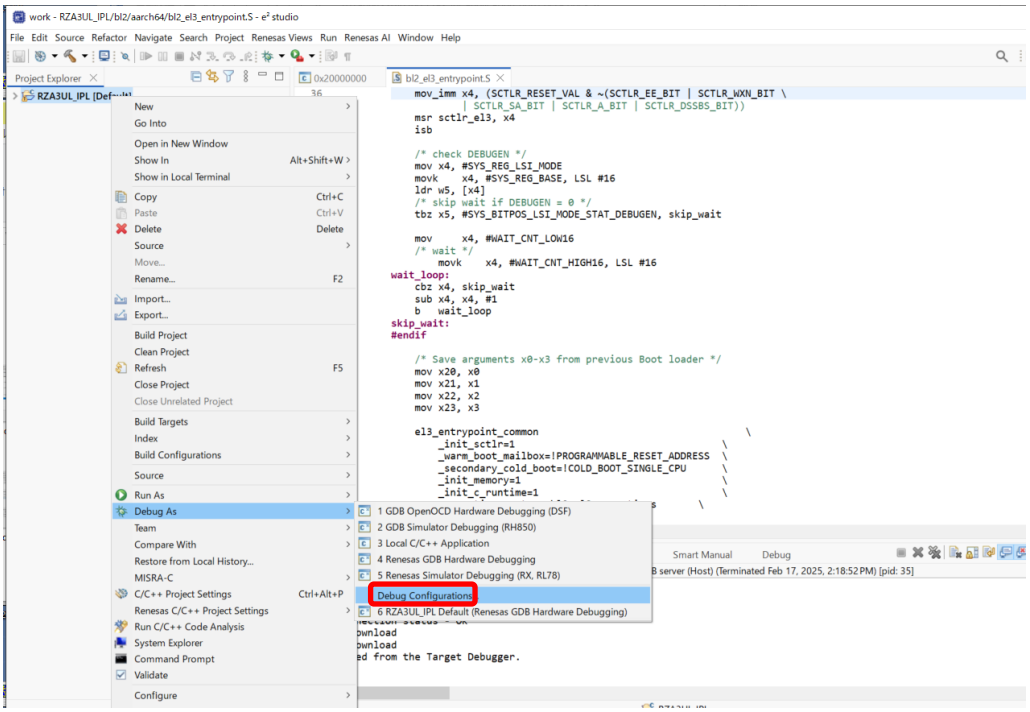


Figure 8.20 Change Debug Configuration of Octa Flash (1)

4. Go to the "Startup" tab and set "On connect" for "rza3ul_smarc_octal_ipl.srec" to Yes.
   Also, set "On connect" in "Program Binary" to [No].
   Then, click [Apply] to complete the setting, and click [Debug] to start the device connection.

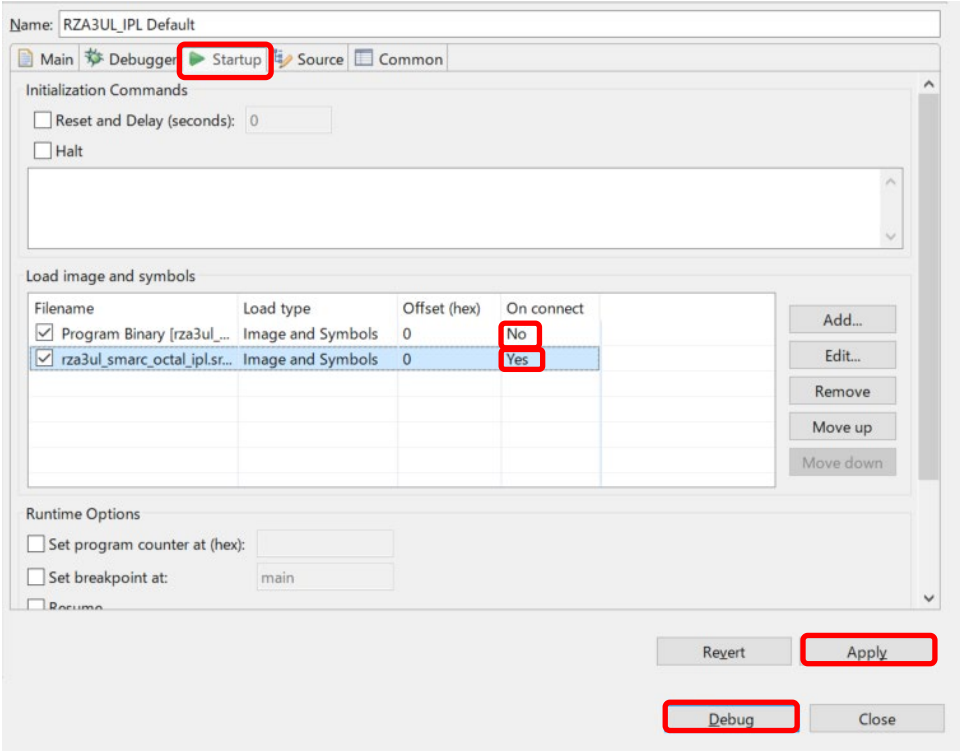


Figure 8.21 Change Debug Configuration of Octa Flash (2)

5. Writing to the Octa Flash memory is performed.
   After the write is complete, the perspective for the debug control will be launched.

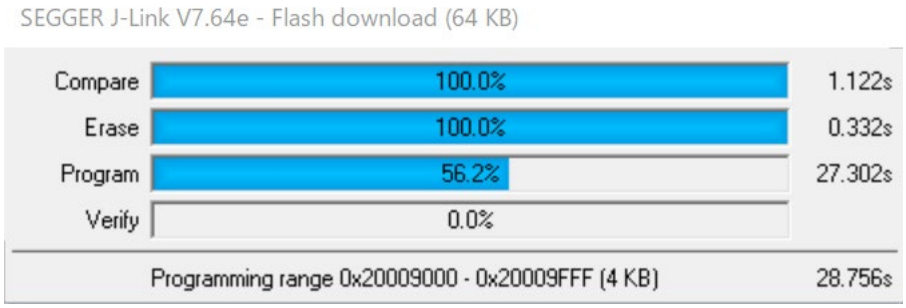


Figure 8.22 Execution of IPL on Octa Flash memory (1)

6. If "Confirm Perspective Switch" is displayed during the launch of the perspective, press [Switch] button.

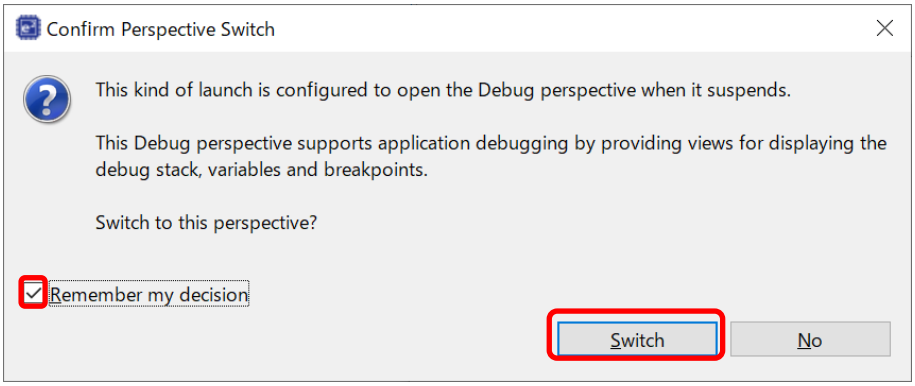


Figure 8.23 Execution of IPL on Octa Flash memory (2)

7. After writing to the Octa flash, click ■ mark in the perspective to disconnect from the SMARC EVK board. After that, press the RESET button on the SMARC EVK board to execute the IPL.
   At this time, the log shown in Figure 8.25 is displayed on Terminal.
   In addition, the program written in the Octa Flash memory starts and the LED of the SMARC EVK board blinks as shown in Figure 8.26.

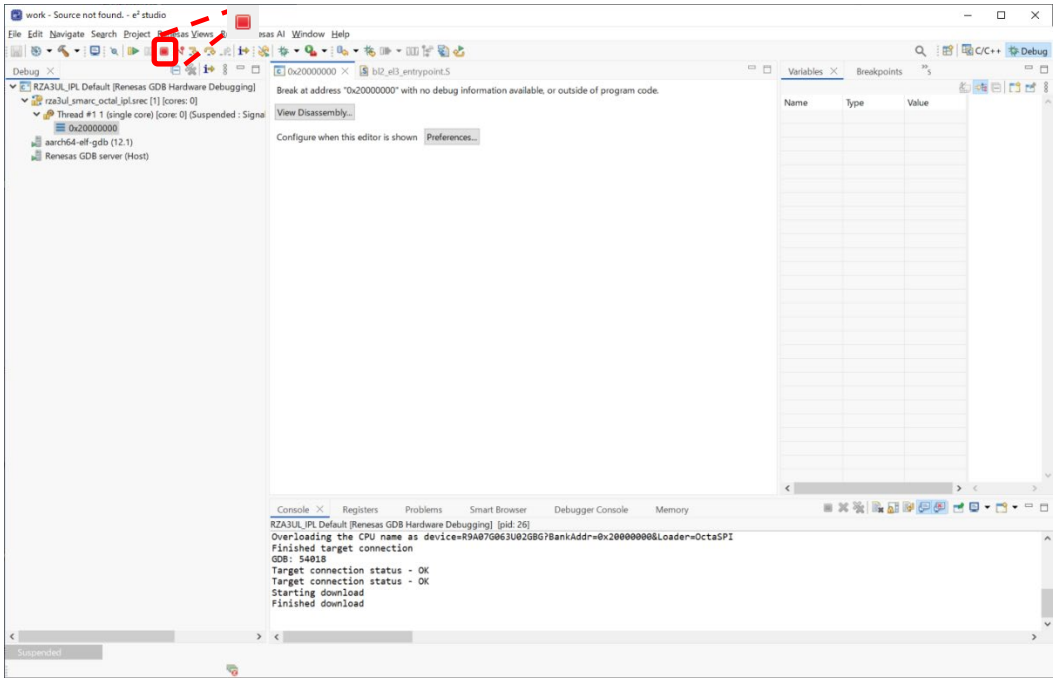


Figure 8.24 Execution of IPL (6)

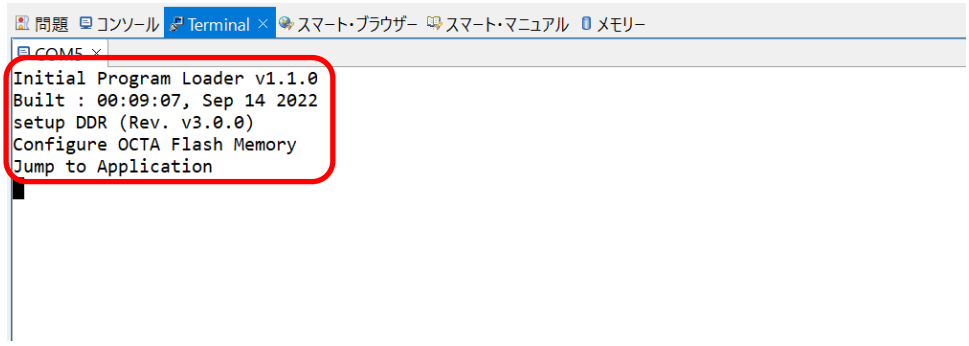


Figure 8.25 IPL execution log

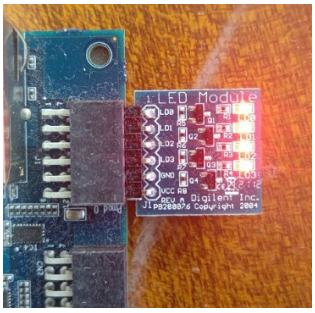


Figure 8.26 LED blinking

9. Application Example

This chapter shows application examples for applying IPL to various environments.

9.1 Execution procedure of IPL by On-chip RAM download

This chapter describes the procedure for writing and executing the IPL to the On-chip RAM for the purpose of facilitating the operation check when customizing the IPL for RZ/A3UL.

1. As the FSP application must be written to the Octa Flash memory to confirm the boot operation by IPL, if the FSP application is not written to the Octa Flash memory, write it.
   For the write procedure, refer to 9.2 Procedure for writing an FSP application to Octa Flash memory.
2. Set the SMARC EVK board operating mode to Boot Mode3 (1.8-V serial Flash memory) and turn on the power.
3. Right-click on RZA3UL_IPL and select [Debug AS] -> [Debug Configurations].
   Then, open RZA3UL_IPL Default configuration.

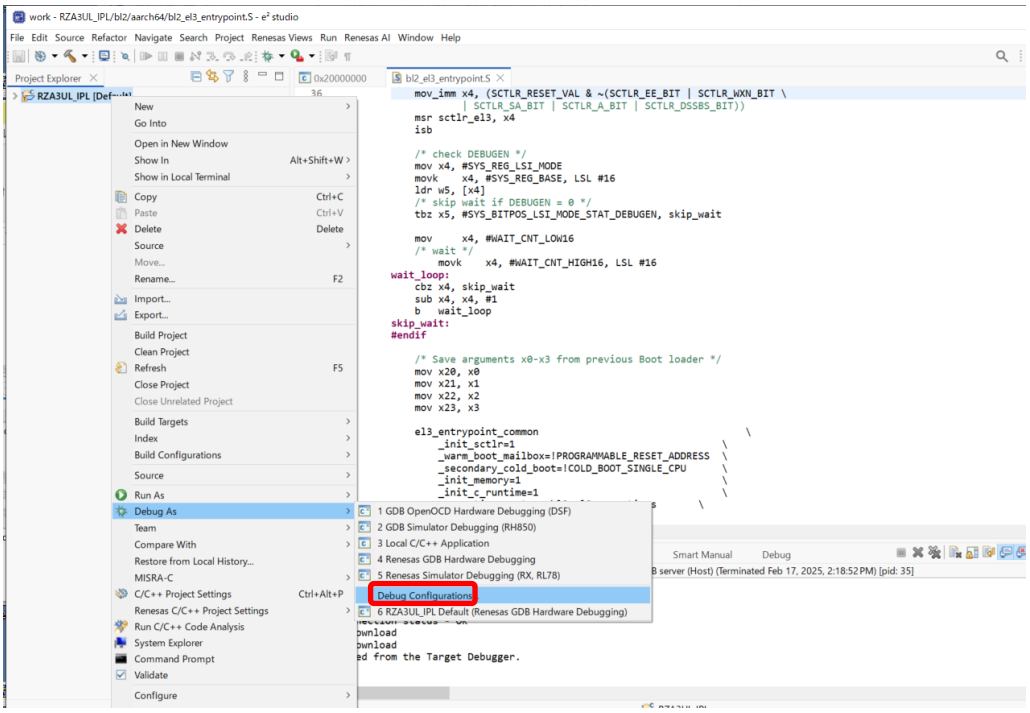


Figure 9.1 Change Debug Configuration of On-chip RAM (1)

4. Go to the "Debugger" -> "Connection Settings" tab and set "Reset at the beginning of connection" to [No].

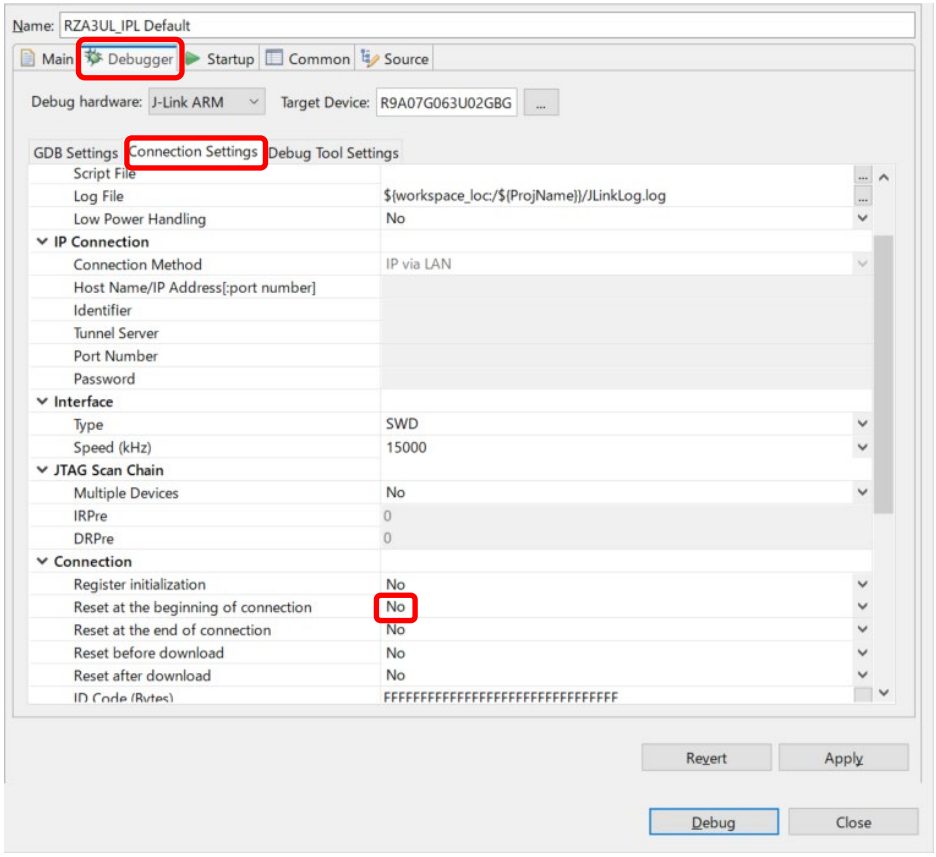


Figure 9.2 Change Debug Configuration of On-chip RAM (2)

5. Go to the "Startup" tab and set "On connect" for "Program Binary" to [Yes].
   Also, set "On connect" in "rza3ul_smarc_octal_ipl.srec" to [No].
   Then, click [Apply] to complete the setting, and click [Debug] to launch the program.
   This will bring up a perspective for the debug control.

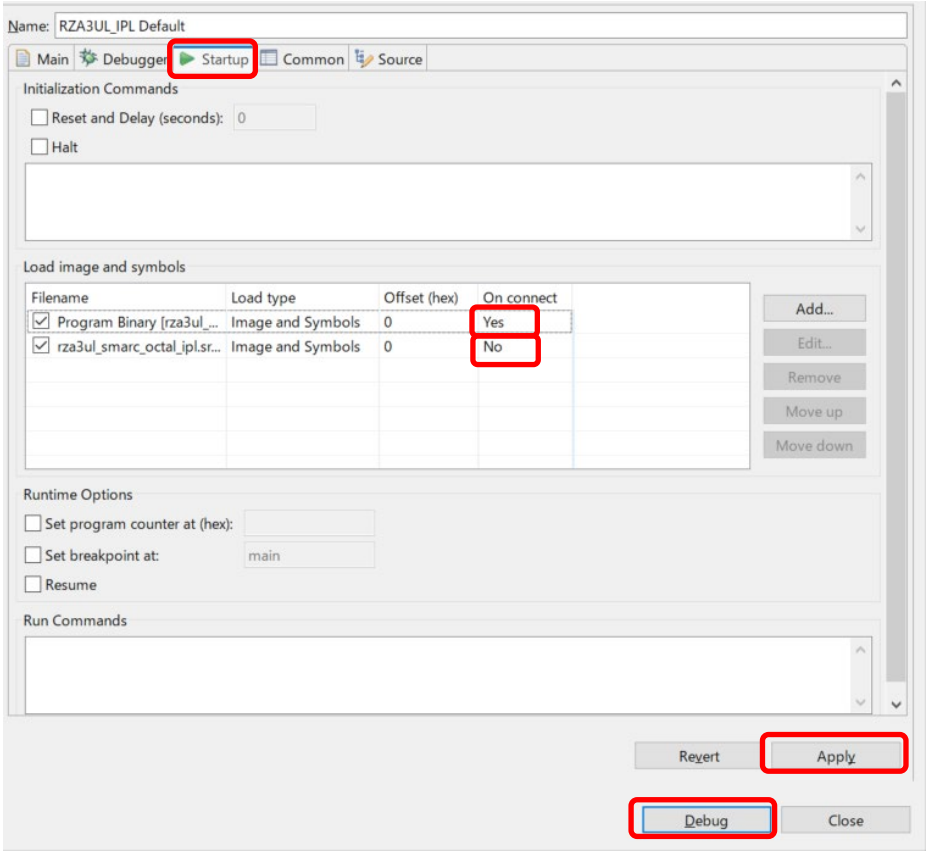


Figure 9.3 Change Debug Configuration of On-chip RAM (3)

6. Click mark in the perspective to perform the IPL.
   The IPL will start executing and the log shown in Figure 9.5 will be displayed in the terminal.
   At this time, the FSP application written on the Octa Flash memory starts, and the LED of the SMARC EVK board blinks as shown in Figure 9.6.

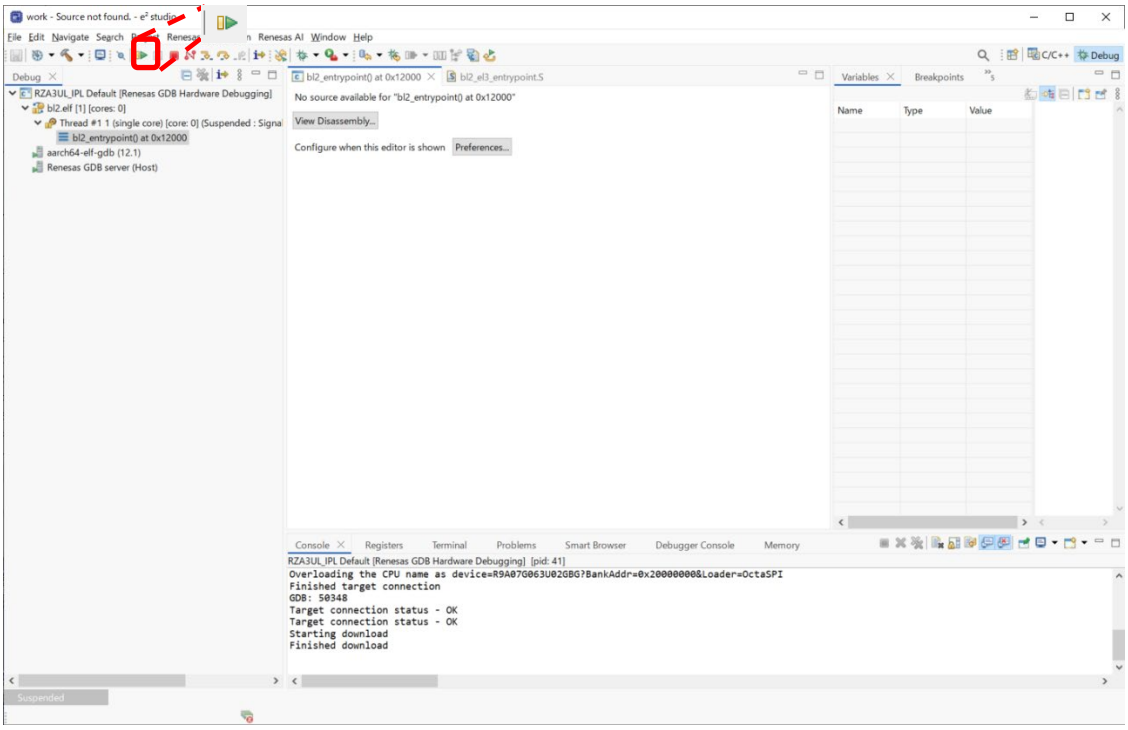


Figure 9.4 Execution of IPL (4)

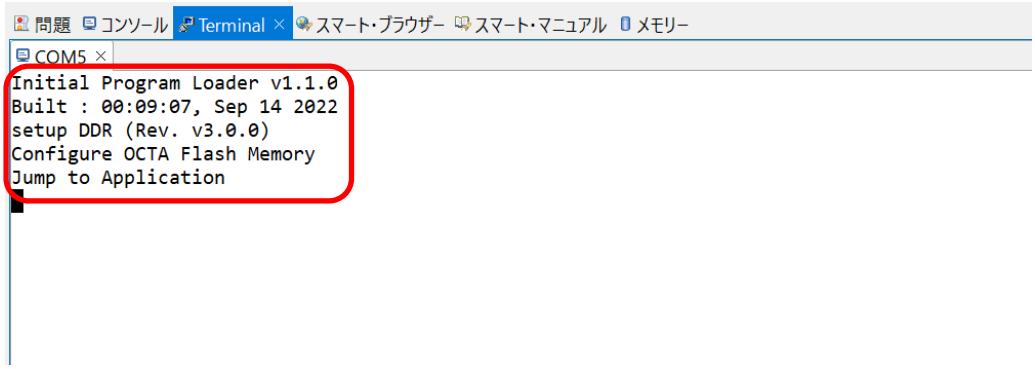


Figure 9.5 IPL execution log

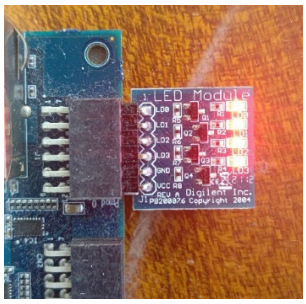


Figure 9.6 LED blinking

9.2 Procedure for writing an FSP application to Octa Flash memory

This chapter describes the procedure for writing an FSP application to Octa Flash memory.

Refer to the Getting Started with RZ/A Flexible Software Package (R01AN6499EJ) for building the FSP application.

This chapter shows an example of saving the FSP application binary file in C:\work.

1. Set the SMARC EVK board operating mode to Boot Mode3 (1.8-V serial Flash memory) and turn on the power.
2. Right-click on RZA3UL_IPL and select [Debug AS] -> [Debug Configurations].
   Then, open RZA3UL_IPL Default configuration.

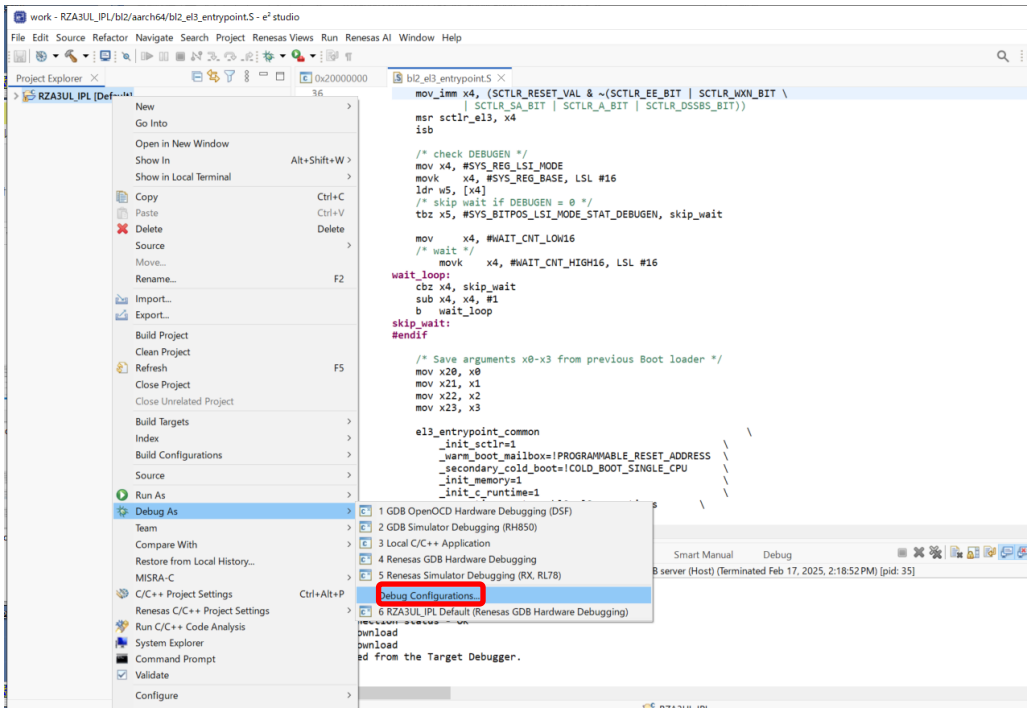


Figure 9.7 Procedure of writing FSP Application (1)

3. Go to the "Debugger" -> "Connection Settings" tab.
   Set the "On connect" of all registered images to [No], set the image not to be downloaded when connecting to the SMARC EVK board.
   Then, click [Apply] to complete the setting, and click [Debug] to launch the program.

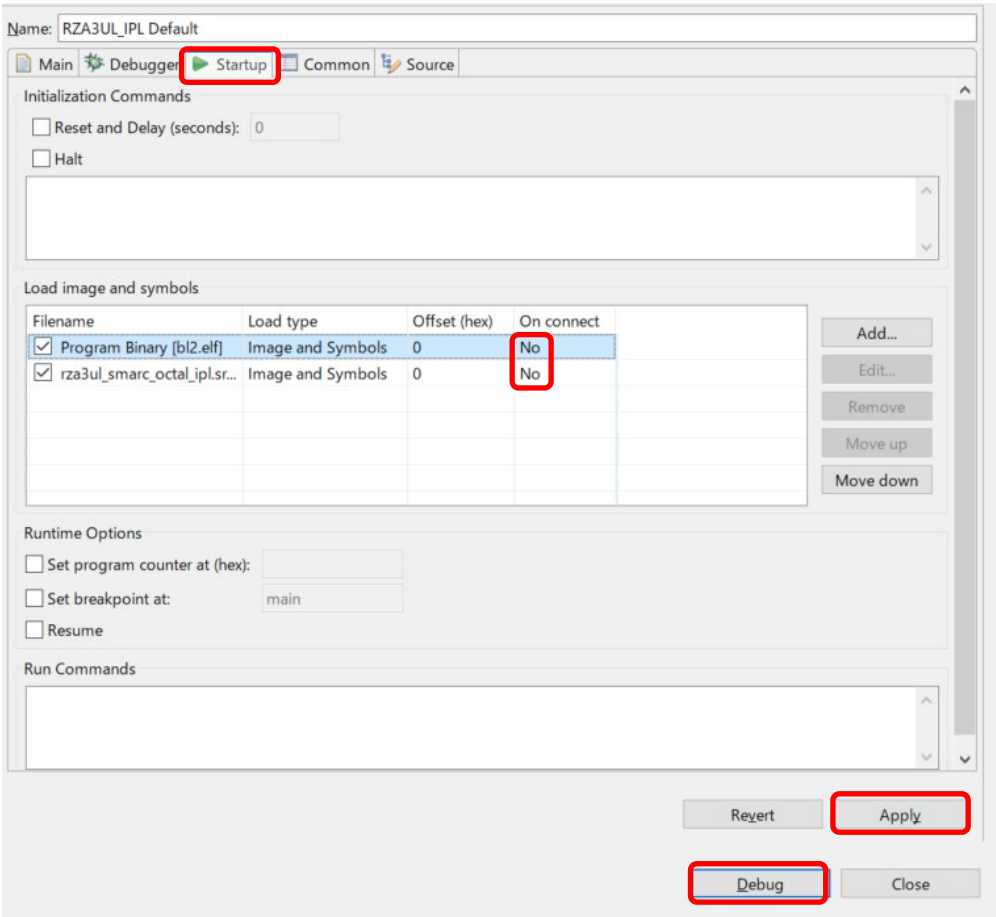


Figure 9.8 Procedure of writing FSP Application (2)

4. If "Confirm Perspective Switch" is displayed during the perspective launch, press [Switch] button.

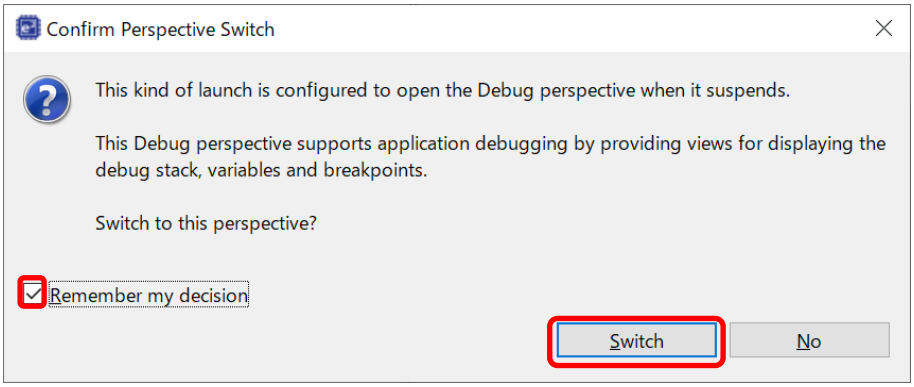


Figure 9.9 Procedure of writing FSP Application (3)

5. The board is connected, and the perspective is launched.
   Click Load Ancillary file .

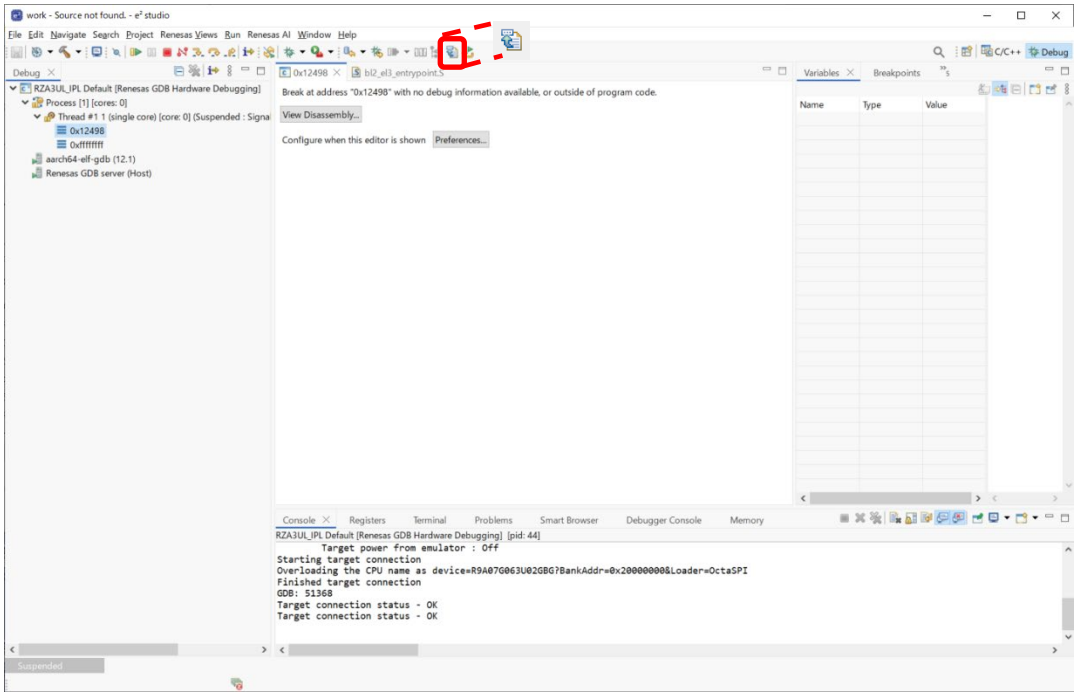


Figure 9.10 Procedure of writing FSP Application (4)

6. The Load Ancillary File window will be launched.
   Click [File System] and select C:\work\RZA3UL_CA55_BLINKY.srec.
   Also, uncheck "Load as raw binary image".
   Click [OK] to write to the Octa Flash memory.

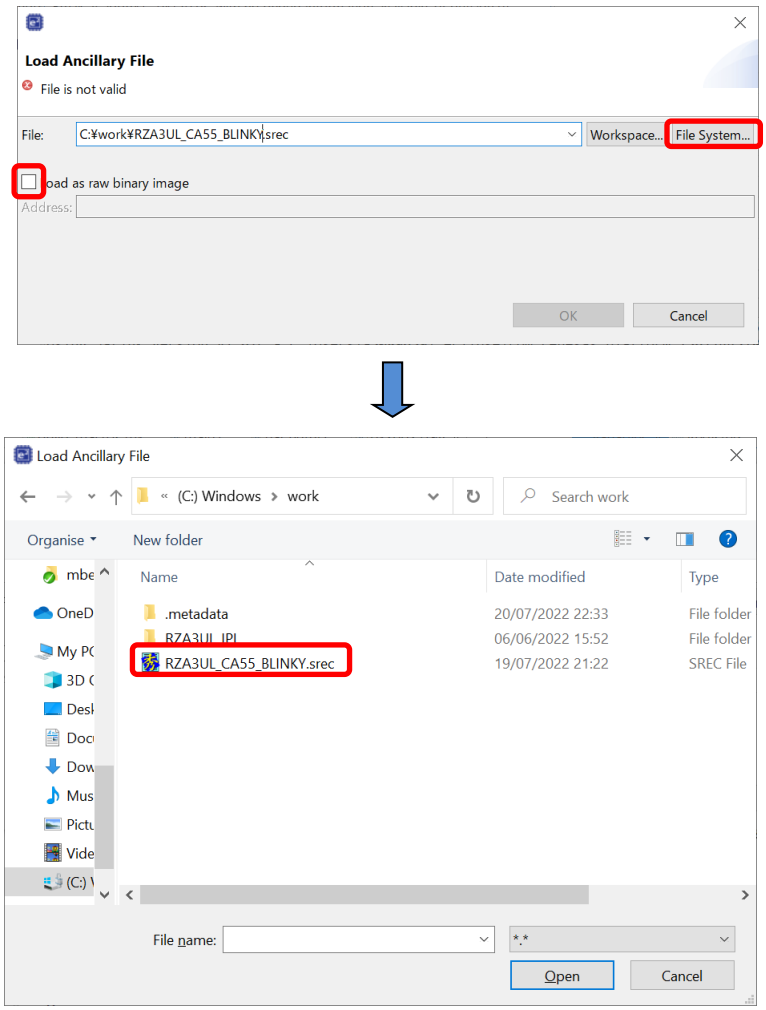


Figure 9.11 Procedure of writing FSP Application (5)

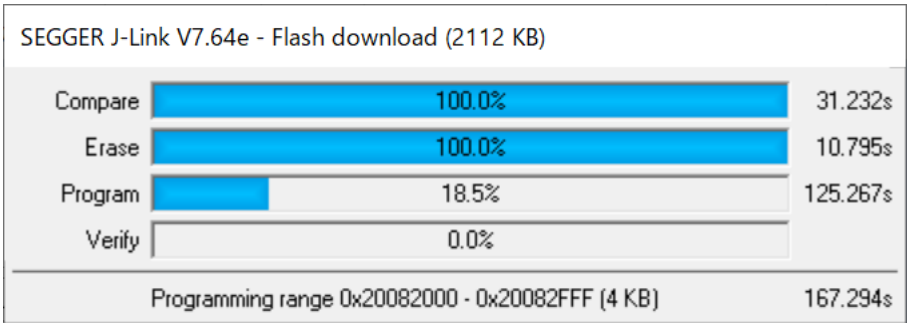


Figure 9.12 Procedure of writing FSP Application (6)

8. After writing, click ■ mark to disconnect from the SMARC EVK board.

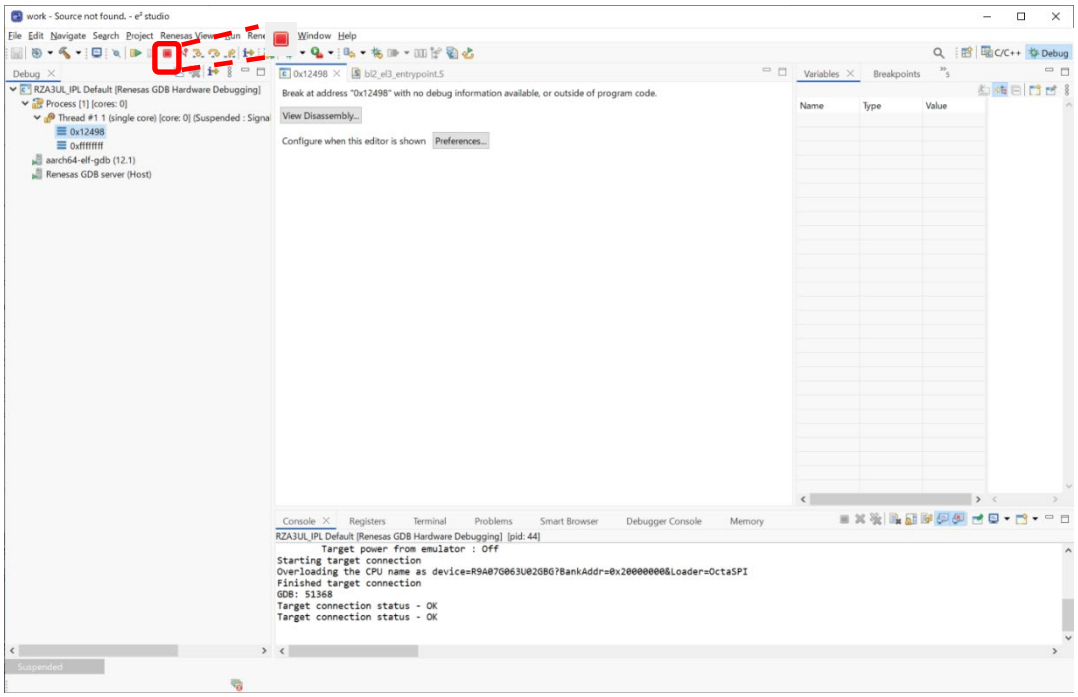


Figure 9.13 Procedure of writing FSP Application (7)

9.3 How to build IPL for custom board

This chapter describes the IPL build environment construction procedure for boards other than SMARC EVK boards (hereafter referred to as custom boards).

The IPL build environment for custom boards is created by modifying the SMARC EVK build environment.

The chapters listed below will show you how to do that.

- How to add Octa Flash memory used by IPL (Chapter 9.3.1)
- Specifications of functions that control Octa Flash memory and how to change functions when memory is changed (Chapter 9.3.1.1 to Chapter 9.3.1.13)
- How to add Octa RAM used by IPL (Chapter 9.3.2)
- Specifications of functions that control Octa RAM and how to change functions when memory is changed (Chapter 9.3.2.1 to Chapter 9.3.2.10)
- How to add settings for boards other than the SMARC EVK board to IPL (Chapter 9.3.3)
- How to add clock supply settings for mounted modules (Chapter 9.3.4)
- How to add pin settings (Chapter 9.3.5)
- How to add MMU settings (Chapter 9.3.6)

9.3.1 How to customize IPL for your Octa Flash memory

This chapter describes how to customize the IPL according to your Octa Flash memory.

First, the method of controlling the Octa Flash memory in IPL is shown.

The IPL is implemented the Octa Flash memory driver and the Octa Memory Controller driver to control the Octa Flash memory.

The Octa Flash memory driver manages the execution of operations of the Octa Flash memory.

The Octa Memory Controller driver is a driver for controlling serial communication between RZ/A3 and Octa Flash memory.

The Octa Flash memory driver creates an operation table consisting of parameters for executing Octa Flash memory operations and passes it to the Octa Memory Controller driver.

The Octa Memory Controller driver controls HW according to the operation table passed as an argument from the Octa Flash memory driver and performs serial communication.

Since the Octa Flash memory driver is an Octa Flash memory-dependent implementation, it must be changed according to your Octa Flash memory.

The Octa Memory Controller driver implementation, on the other hand, does not rely on Octa Flash memory.

Therefore, there is no need to change even when changing the Octa Flash memory.

Figure 9.14 shows the operation image diagram of each driver.

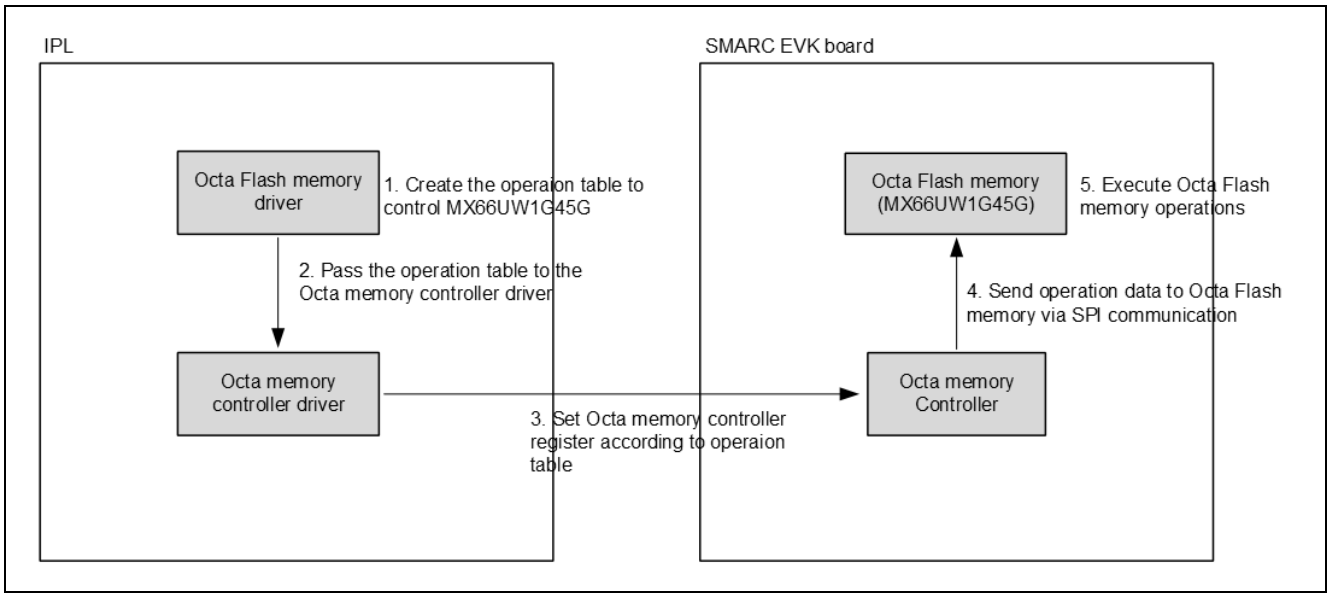


Figure 9.14 Operating image of Octa Flash memory driver and Octa Memory Controller driver

1. The Octa Flash memory driver creates an operation table corresponding to your memory.
2. The Octa Flash memory driver calls the Octa Memory Controller driver.
   At this time, the operation table is passed to the Octa Memory Controller driver.
3. The Octa Memory Controller driver sets the Octa Memory Controller register according to the received operation table parameters.
4. Octa Memory Controller performs serial communication with the Octa Flash memory according to the settings made in step3 and sends operation data to the Octa Flash memory.
5. The Octa Flash memory controls itself according to the received operation data.

Figure 9.15 shows the relationship between operation table parameters and transfer formats.

As an example of an operation table, Figure 9.16 shows MX66UW1G45G read command operation table setting values.

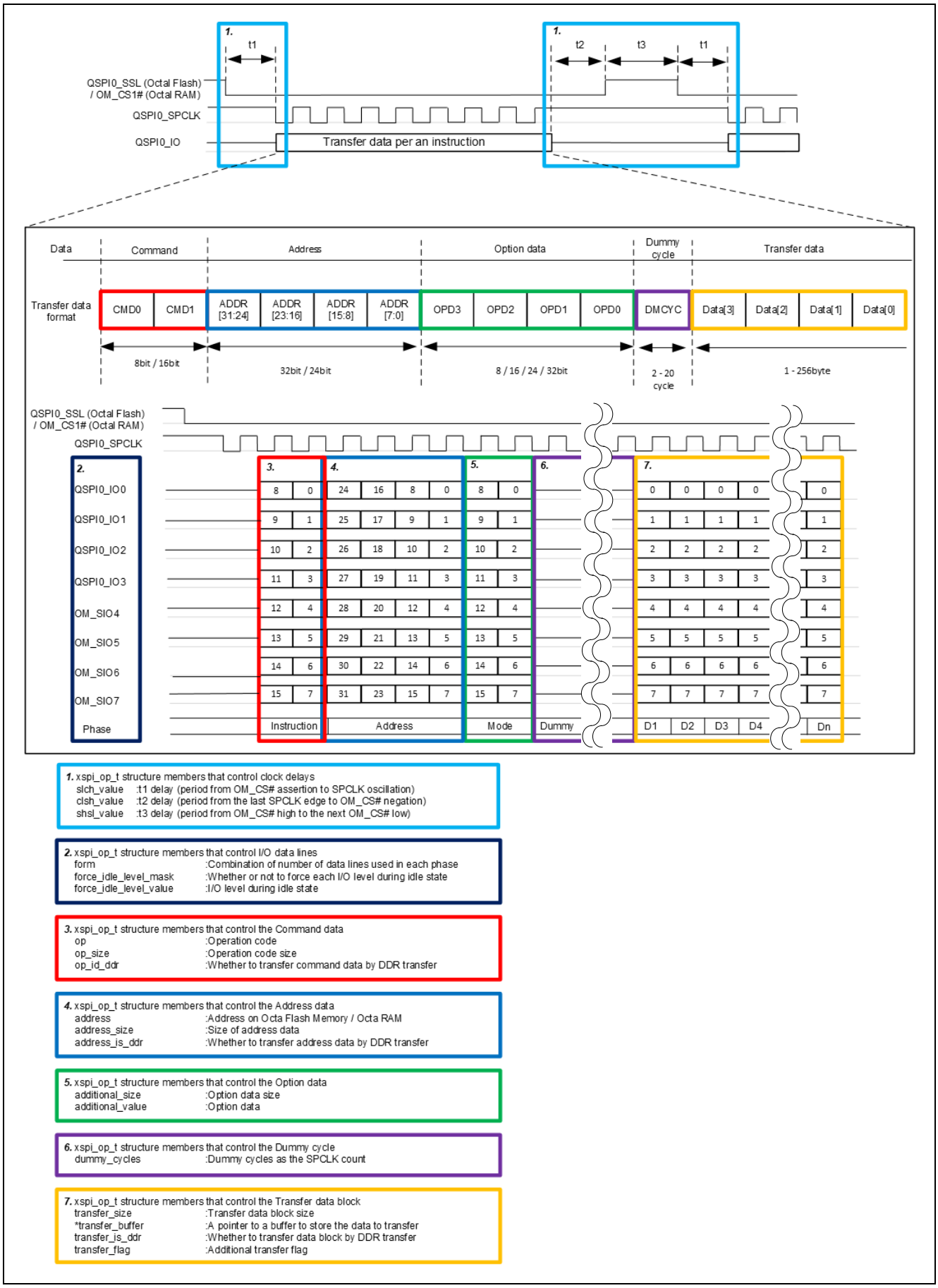


Figure 9.15 Correspondence between the member of xspi_op_t structure and the transfer format

Setting example
8 I/O DT Read command (0xEE11) of the MX66UW1G45G
This is defined in the source code as variable "rop".

form	:SPI_FORM_8_8_8
op	:0xEE11
op_size	:2
op_id_ddr	:true
address	:0
address_size	:4
address_is_ddr : true	
additional_size	:0
additional_value	:0
dummy_cycles	:10
transfer_size	: user-specified size
*transfer_buffer	: user-specified address
transfer_is_ddr	:true
transfer_flag	:0
force_idle_level_mask	:0
force_idle_level_value	:0
slch_value	:0
clsh_value	:5
shsl_value	:1

Setting memober description.
Similar to the one in Figure 9.17

1. xspi_op_t structure members that control clock delays

slch_value	:t1 delay (period from OM_CS# assertion to SPCLK oscillation)
clsh_value	:t2 delay (period from the last SPCLK edge to OM_CS# negation)
shsl_value	:t3 delay (period from OM_CS# high to the next OM_CS# low)

2. xspi_op_t structure members that control I/O data lines

form	:Combination of number of data lines used in each phase
force_idle_level_mask	:Whether or not to force each I/O level during idle state
force_idle_level_value	:I/O level during idle state

3. xspi_op_t structure members that control the Command data

op	:Operation code
op_size	:Operation code size
op_id_ddr	:Whether to transfer command data by DDR transfer

4. xspi_op_t structure members that control the Address data

address	:Address on Octal Flash Memory / Octa RAM
address_size	:Size of address data
address_is_ddr	:Whether to transfer address data by DDR transfer

5. xspi_op_t structure members that control the Option data

additional_size	:Option data size
additional_value	:Option data

6. xspi_op_t structure members that control the Dummy cycle

dummy_cycles	:Dummy cycles as the SPCLK count

7. xspi_op_t structure members that control the Transfer data block

transfer_size	:Transfer data block size
*transfer_buffer	:A pointer to a buffer to store the data to transfer
transfer_is_ddr	:Whether to transfer data block by DDR transfer
transfer_flag	:Additional transfer flag

Figure 9.16 Setting example of parameters corresponding to transfer format

Shows how to add the Octa Flash memory driver for the new Octa Flash memory.

1. Open "c:\work\RZA3UL_IPL\plat\renesas\rz\common\drivers\xspidevice" folder, copy the "octaflash_mx66uw" folder and rename it to any name.
   From step2 onwards, the user will change the file name and structure name to correspond to the folder name decided in this step.
   This time, rename the folder name to "octaflash_custom".

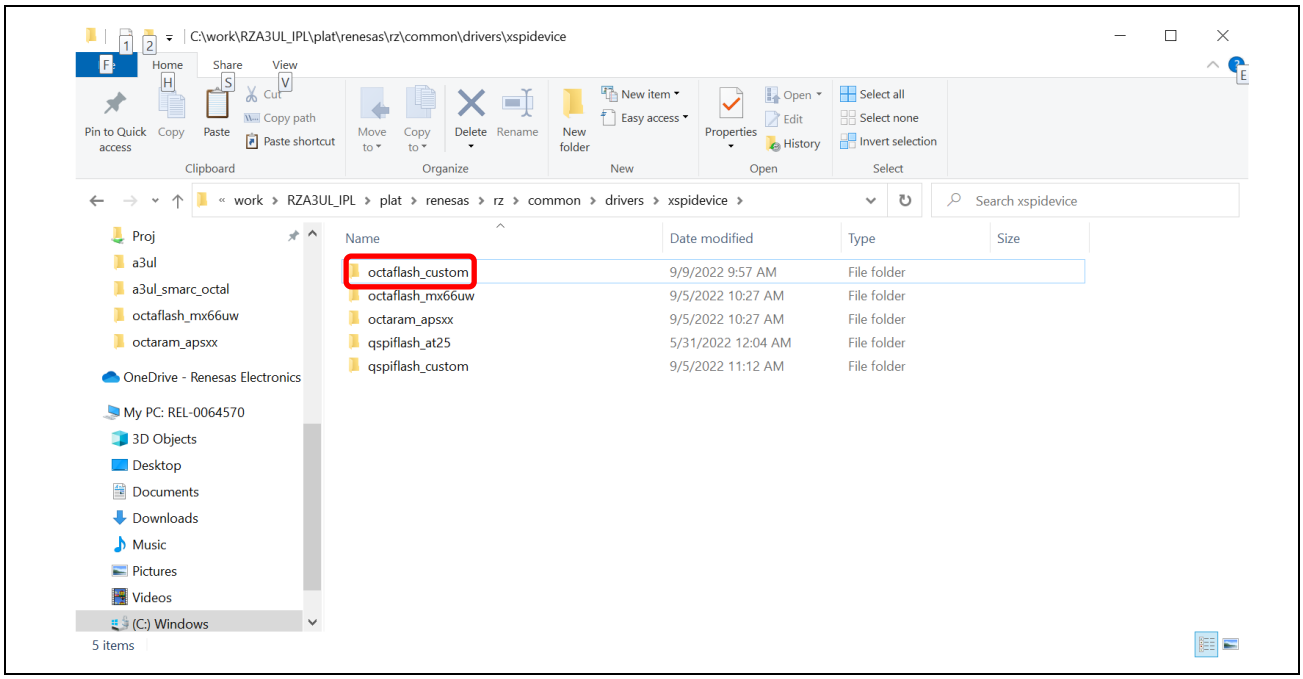


Figure 9.17 Rename Octa Flash memory driver folder

2. Open "octaflash_custom" folder and change "octaflash_mx66uw.c", "octaflash_mx66uw_api.h" to "octaflash_custom.c", "octaflash_custom_api.h".

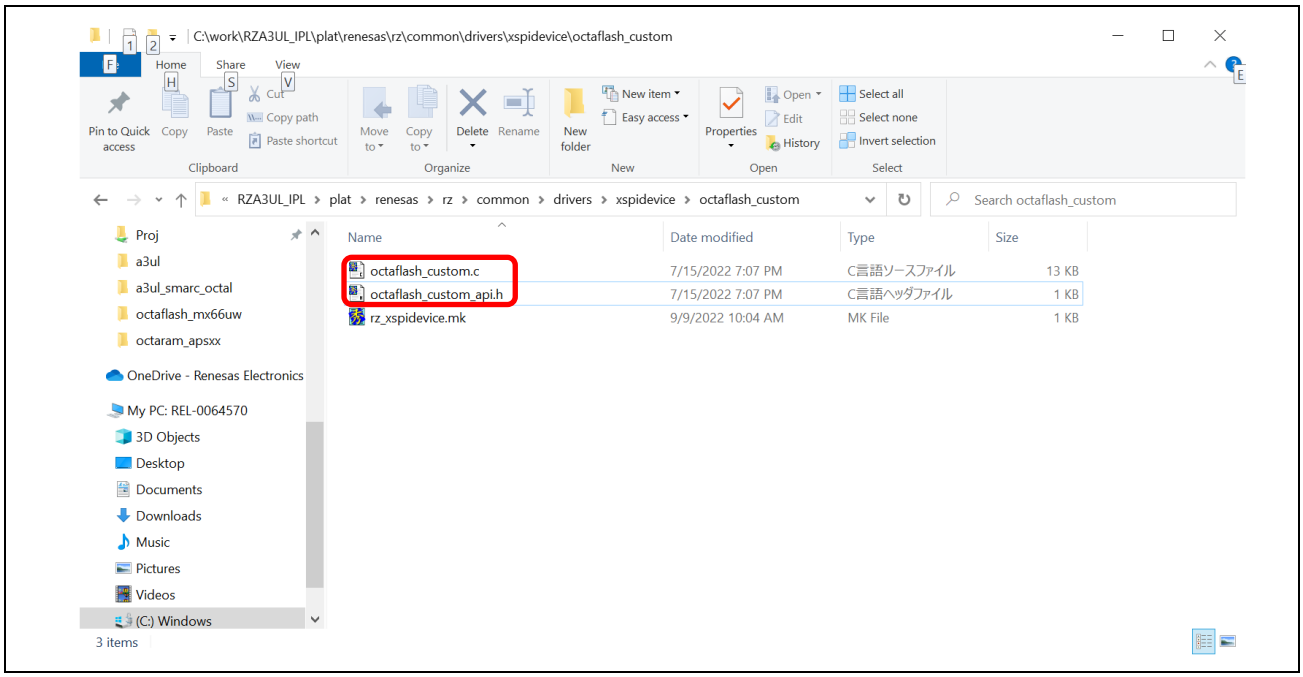


Figure 9.18 Rename Octa Flash memory driver files

3. Open "rz_xspidevice.mk" and change "_XSPI_DEVICE:=octaflash_mx66uw" on line 6 to "_XSPI_DEVICE:=octaflash_custom".

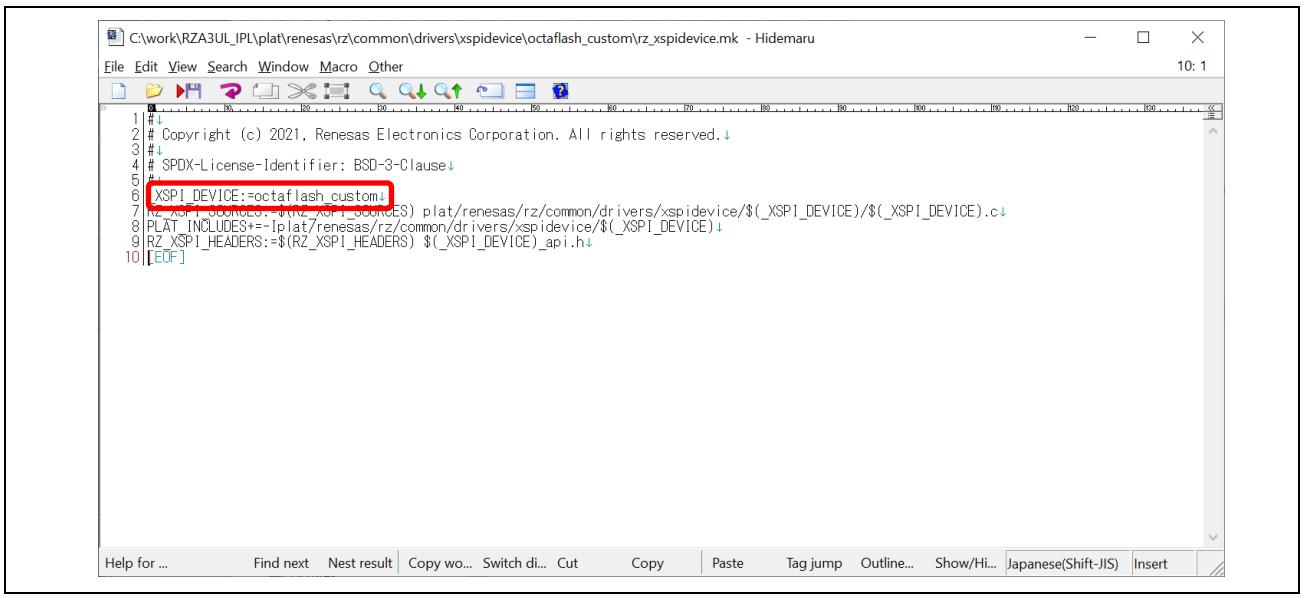
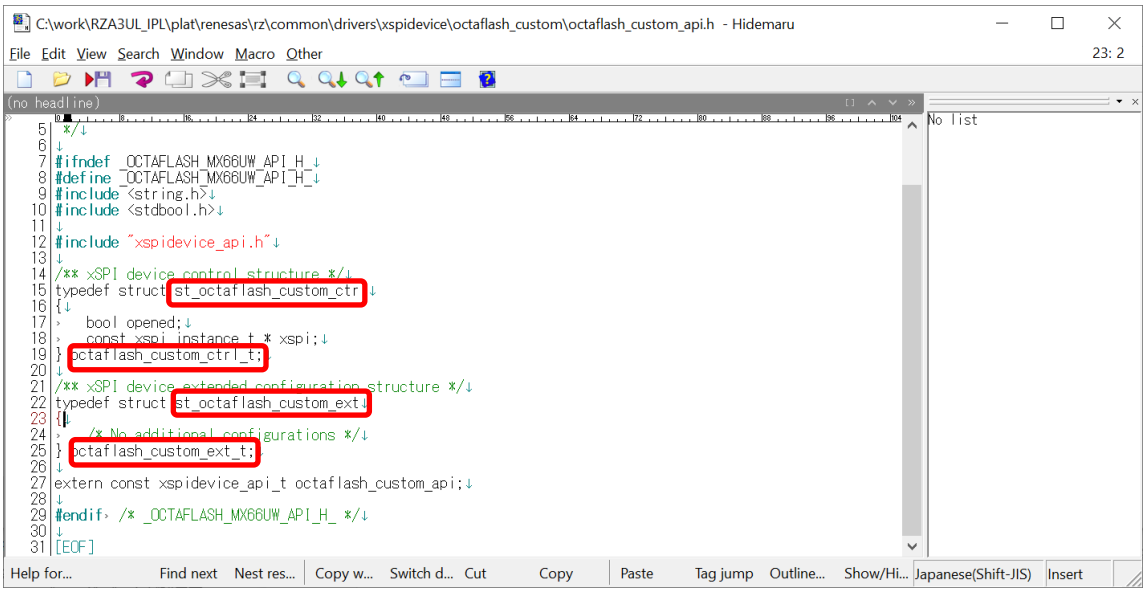


Figure 9.19 Change Octa Flash memory driver Makefile

4. Open "octaflash_custom_api.h" and replace all structure definitions of "xx_octaflash_mx66uw_xx" with "xx_octaflash_custom_xx".



Note: The octaflash_custom_ext_t structure is for extended settings.
If the members corresponding to the parameters required for your Octa Flash memory are not provided in the octaflash_custom_t structure, add those members as extended settings.

Figure 9.20 Change Octa Flash memory driver structs name

5. Open octaflash_custom.c and replace the included header with "octaflash_custom_api.h".
   Also, reflect the structure name replaced in step 4.

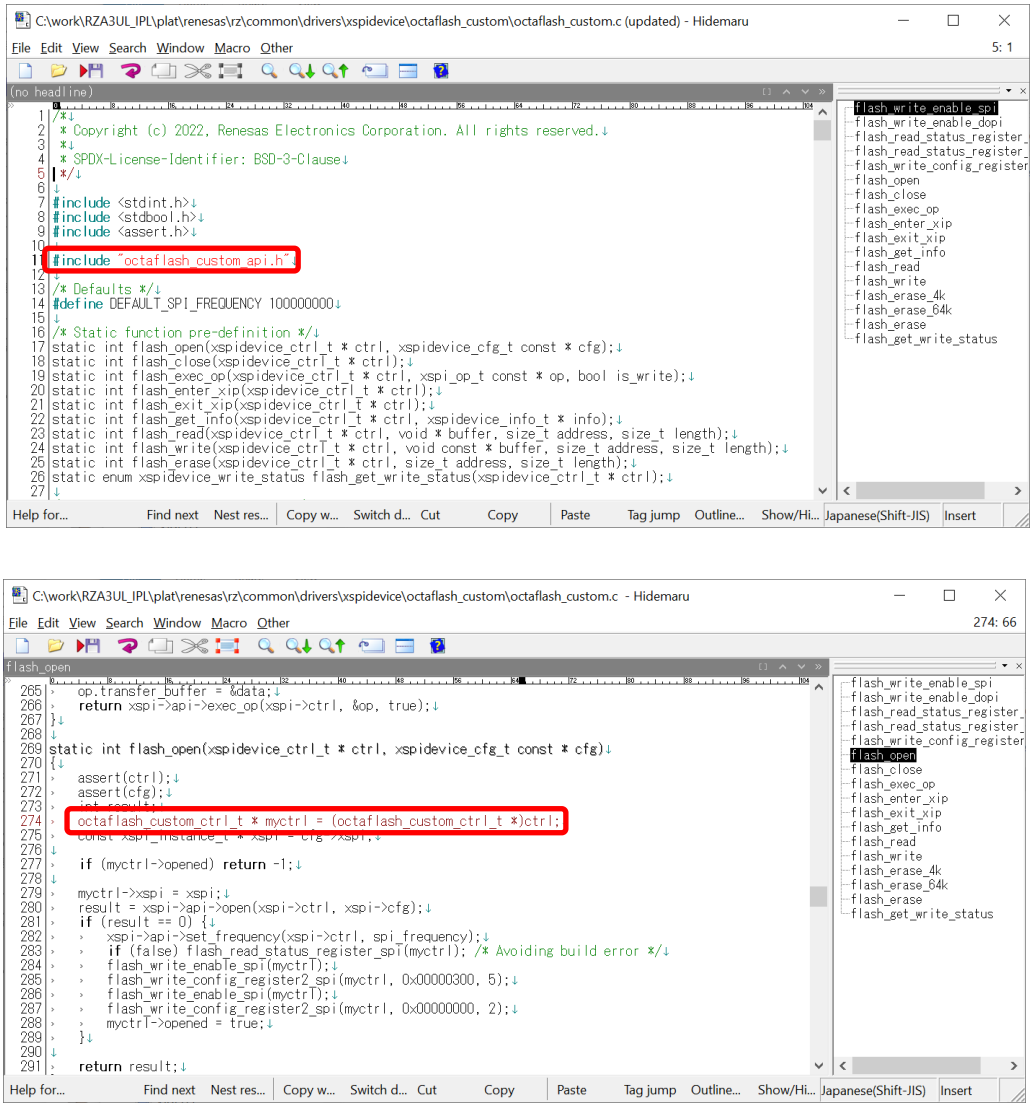


Figure 9.21 Change Octa Flash memory driver source code

6. Change the implementation of API functions in Table 9.1 and Sub functions in Table 9.2 Table 9.1according to your Octa Flash memory.
   9.3.1.1 to 9.3.1.13 show the specifications of each function implemented in the default IPL and the change points when changing the Octa Flash memory.
   Please refer to these and change the implementation of each function.

Table 9.1 Octa Flash memory driver API Function

Function Name	Description
flash_open	Octa Flash memory driver open function
flash_close	Octa Flash memory driver close function
flash_exec_op	Octa Flash memory driver operation function
flash_enter_xip	Octa Flash memory driver Memory-map Mode start function
flash_exit_xip	Octa Flash memory driver Memory-map Mode stop function
flash_get_info	Octa Flash memory driver information get function
flash_read	Octa Flash memory driver read function
flash_write	Octa Flash memory driver write function
flash_erase	Octa Flash memory driver erase function
flash_get_write_status	Octa Flash memory driver write status get function

Table 9.2 Octa Flash memory Sub Function

Function Name	Description
flash_write_enable_spi	Set write enable to Octa Flash memory in SPI mode
flash_write_enable_dopi	Set write enable to Octa Flash memory in DOPI mode
flash_read_status_register_spi	Read the status register of Octa Flash memory in SPI mode
flash_read_status_register_dopi	Read the status register of Octa Flash memory in DOPI mode
flash_write_config_register2_spi	Write value to specified address in the Configuration Register2 in SPI mode
flash_erase_4k	Erase Octa Flash memory in units of 4Kbytes
flash_erase_64k	Erase Octa Flash memory in units of 64Kbytes

7. Add the added Octa Flash memory driver to the board settings.
   Refer to Chapter 9.3.3 for how to add.

9.3.1.1 flash_open

flash_open() is a function that initializes the Octa Flash memory so that it can be accessed.

flash_open() implemented in default IPL initializes to access MX66UW1G45G mounted on the SMARC EVK board.

flash_open	
Outline	Initialize the Octa Flash memory
Declaration	int flash_open (xspidevice_ctrl_t * ctrl, xspidevice_cfg_t const * cfg)
Description	- Set SPCLK frequency - Release write protect and set value to Configuration register - Set dummy cycle - DTR OPI mode is enabled - Set opened flag
Arguments	*ctrl : Pointer to xSPI device control structure *cfg : Pointer to xSPI device configure structure
Return value	0 : Initialization is complete -1 : Initialization is failed
Note	-

Figure 9.22 shows a flowchart of the flash_open() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Clock frequency setting change
- P2 Release Write Protect
- P3 Dummy Cycle setting
- P4 Enable DTR OPI mode

These are shown as P1 to P4 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

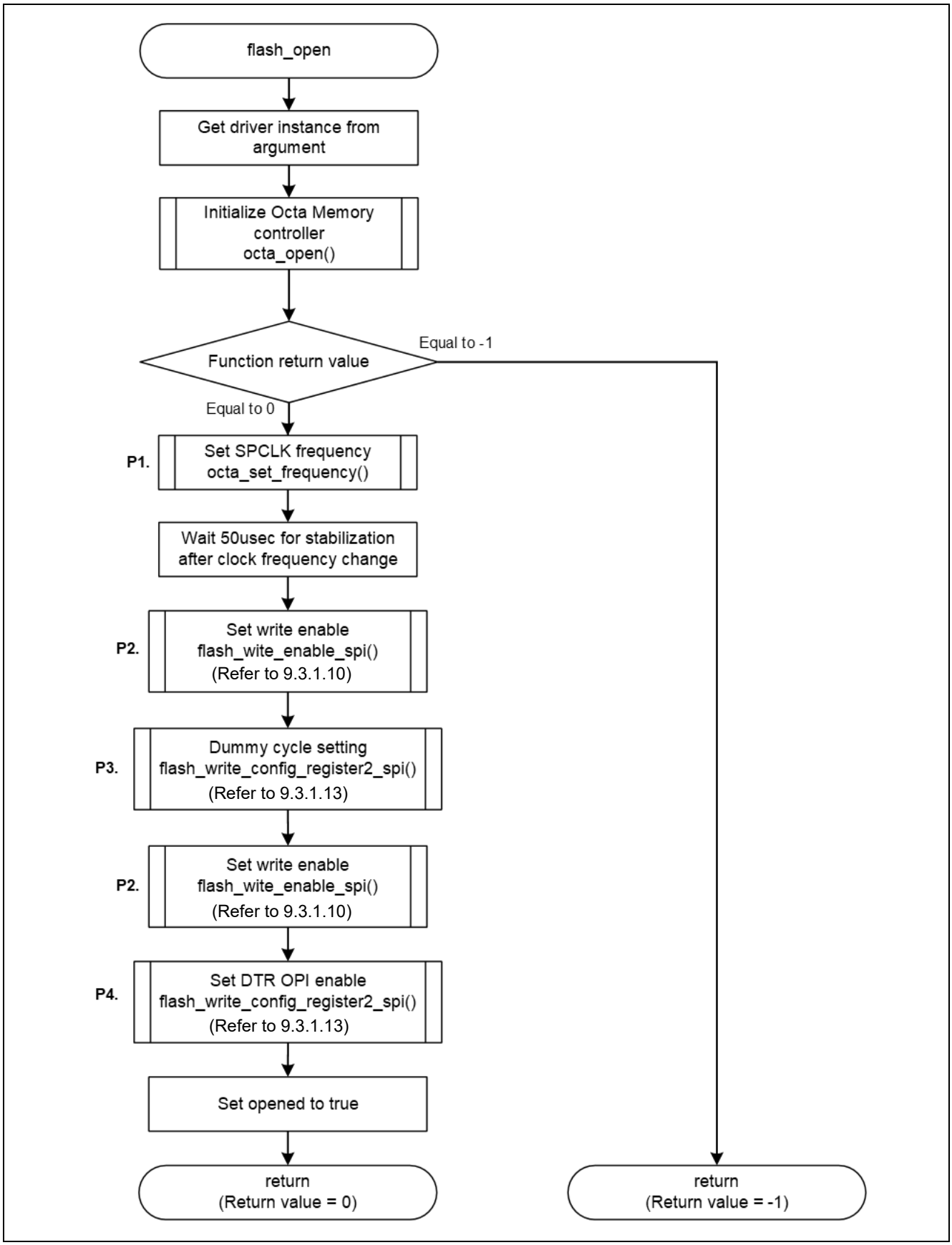


Figure 9.22 Flowchart of flash_open

P1. Clock Frequency setting change

flash_open sets the operating clock frequency of the Octa Flash memory.

The defined value of DEFAULT_SPI_FREQUENCY in L14 of octalflash_custom.c shown in Figure 9.23 is used for the operating clock frequency.

Set this define value to the optimum value for your Octa Flash memory.

The maximum value that can be set is 100000000 (100MHz) and the minimum value is 3125000 (3.125MHz).

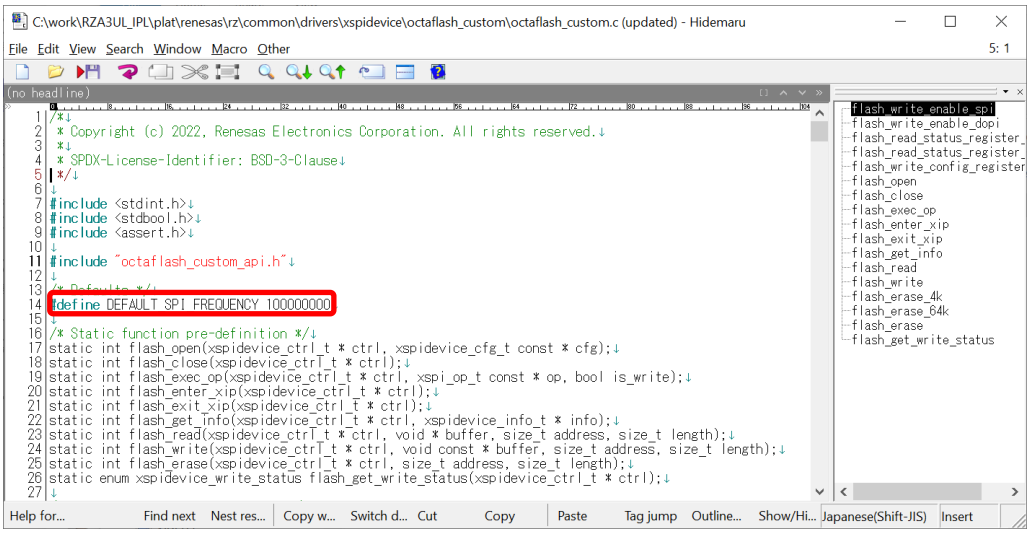


Figure 9.23 Change operation clock frequency

P2. Release Write protect

Most of Octa Flash memories require release of write protect when writing data.

The default Octa Flash memory driver provides a sub function flash_write_enable_spi() that release Write Protect for MX66UW1G45G.

Please refer to 9.3.1.10 for change of Write protect release method.

P3. Dummy Cycle setting

In the MX66UW1G45G, the dummy cycle and operating frequency settings must be set in Configuration Register2 as initial settings.

Configuration Register setting processing is performed by the sub function flash_write_config_register2_spi() function.

Refer to 9.3.1.12 for the writing procedure to Configuration Register2.

Configuration Register2 of MX66UW1G45G consists of multiple registers, and the addresses and register settings shown in Table 9.3 are passed as arguments to flash_write_config_register2_spi() to perform serial communication.

This time, since the operating frequency is 100MHz, set 101b in the Configuration Register (Address: 0x00000300)

Table 9.3 Configuration Register (Address: 0x00000300)

Address	Bit	Bit Name	Description	Setting
0x00000300	[7:3]	-	Reserved	Reserved
	[2:0]	DC	Dummy cycle	Refer to follow dummy cycle and frequency setting

	dummy cycle	Octa I/O STR, DTR (MHz)
000b	20	200
001b	18	173
010b	16	166
011b	14	155
100b	12	133
101b	10	104
110b	8	84
111b	6	66

P4. Enable DTR OPI mode

Enable DTR OPI Mode (DOPI).

To enable DOPI mode, use Configuration Register2 (Address: 0x00000000), so pass the address and set value shown in Table 9.4 to the flash_write_config_register2_spi() function as in P3.

In this case, set 10b to Configuration Register2 (Address: 0x00000000).

If you need to change the operation mode setting for your memory, refer to this case and implement the process that makes the optimal settings for the registers according to the specifications of the memory.

A DQS signal is required when using STR OPI Mode.

In that case, set it so that the DQS signal can be output.

Table 9.4 Configuration Register2 (Address: 0x00000000)

Address	Bit	Bit Name	Description	Setting
0x00000000	[7:4]	-	Reserved	Reserved
	[3]	CRCEN	Parity check Enable	1: parity check Enable 0: parity check Disable
	[2]	-	Reserved	Reserved
	[1]	DOPI	DTR OPI Enable	00: SPI 01: STR OPI enable 10: DTR OPI enable 11: inhibit
	[0]	SOPI	STR OPI Enable	

9.3.1.2 flash_close

flash_close() is a function to terminate the Octa Flash memory driver.

flash_close() implemented in the default IPL only clears the opened flag, which indicates the open state of the driver.

Therefore, it is possible to access Octa Flash memory even after executing flash_close().

flash_close		
Outline	Close the Octa Flash memory	
Declaration	int flash_close (xspidevice_ctrl_t * ctrl)	
Description	- Clear opened flag	
Arguments	*ctrl	: Pointer to xSPI device control structure
Return value	0	: Close process is complete
	-1	: Close process is failed
Note	-	

Figure 9.24 shows a flowchart of the flash_close() implementation in the default IPL.

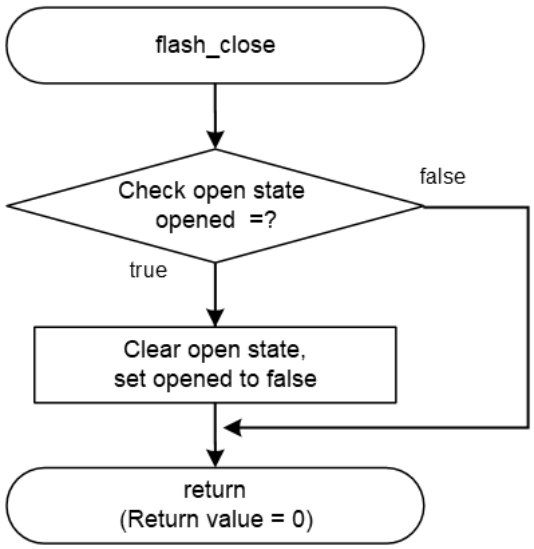


Figure 9.24 Flowchart of flash_close

9.3.1.3 flash_exec_op

flash_exec_op() is a function to send operation information to be executed by Octa Flash memory.

The flash_exec_op() implemented in the default IPL is called from the functions shown below and sends the operation data to be executed by the MX66UW1G45G mounted on the SMARC EVK board.

- flash_write_enable_spi()
- flash_write_enable_dopi()
- flash_read_status_register_spi()
- flash_read_status_register_dopi()
- flash_write_config_register2_spi()
- flash_erase_4k()
- flash_erase_64k()

flash_exec_op	
Outline	Execute operation to Octa Flash memory
Declaration	void flash_exec_op (xspidevice_ctrl_t * ctrl, xspi_op_t const * op, bool is_write)
Description	- Pass operation table *op to octa_exec_op() to transfer operation data
Arguments	*ctrl : Pointer to xSPI device control structure *op : Pointer to xSPI interface operation structure is_write : Specifies the transmission/reception direction of the operation(0=read、1=write)
Return value	0 : Execute the operation is complete -1 : Execute the operation is failed
Note	-

Figure 9.25 shows a flowchart of the flash_exec_op() implementation in the default IPL.

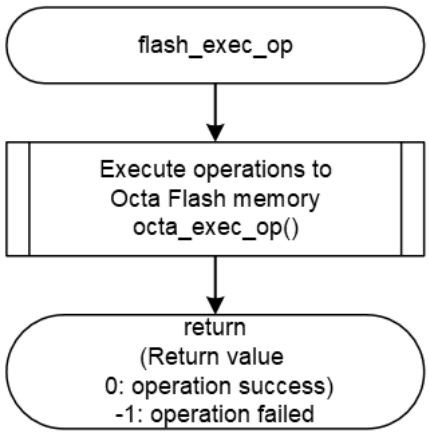


Figure 9.25 Flowchart of flash_exec_op

9.3.1.4 flash_enter_xip

flash_enter_xip() is a function to set Memory-map Mode

By executing this function, data can be read by read access to the Octa Flash memory area (after H'20000000) in the memory space shown in Figure 5.4.

When Memory-map Mode is enabled, Octa Memory Controller automatically performs serial communication and reads data when read access is made to Octa Flash memory area.

At this time, the Read operation registered in Octa Memory Controller is used for serial communication.

flash_enter_xip() implemented in the default IPL registers the Read operation of MX66UW1G45G mounted on the SMARC EVK board to Octa Memory Controller.

flash_enter_xip		
Outline	Set the Memory-map Mode to Octa Flash memory	
Declaration	int flash_enter_xip (xspidevice_ctrl_t * ctrl)	
Description	- Register Read operation to Octa Memory Controller	
	- Adjust memory access timing (memory calibration)	
Arguments	*ctrl	: Pointer to the control structure for instance
Return value	0	: Setting is complete
	-1	: Setting is failed
Note	-	

Figure 9.26 shows a flowchart of the flash_enter_xip() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Register Read operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

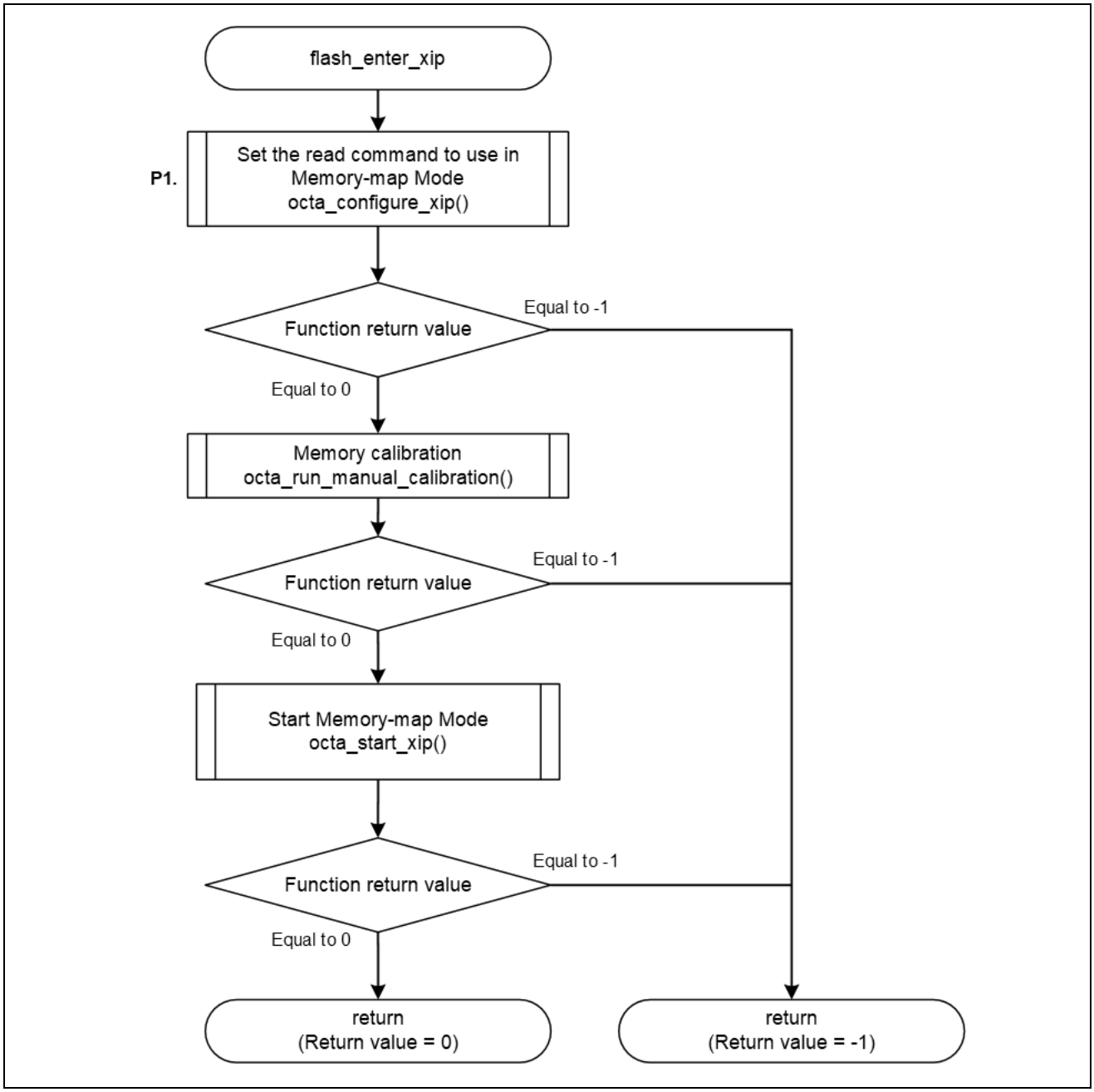


Figure 9.26 Flowchart of flash_enter_xip

P1. Register Read operation

Register the Read operation used in Memory-map Mode in the argument rop of octa_config_xip().

The Octa Flash memory driver implemented in the default IPL uses the 8 I/O DT Read operation (code: 0xEE) as the Read operation.

Table 9.5 shows the rop settings when using the 8 I/O DT Read operation.

When changing the Octa Flash memory, rewrite rop to the contents corresponding to the Read operation of your Octa Flash memory.

Table 9.5 rop setting contents when using 8 I/O DT Read for DOPI Mode (0xEE11)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0xEE11	Command code for DOPI mode "8 I/O DT Read"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	Address is not set because bus access is translated
address_is_ddr	true	Address transfer method (DDR)
address_size	4	32bit address
additional_value	0	Option data
additional_size	0	Option data size
dummy_cycles	10	Dummy cycles = 10cycles
transfer_buffer	NULL	Buffer pointer is not set because bus access is translated
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Transfer data size is not set because bus access is translated
force_idle_level_mask	0	IO3/HOLD is Hi-z during idle state
force_idle_level_value	0	IO3/HOLD is Hi-z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.1.5 flash_exit_xip

flash_exit_xip() is a function to exit Memory-map Mode.

The Octa Memory Controller implemented in RZ/A3 always enables Memory-map Mode, so executing flash_exit_xip() implemented in the default IPL does not actually end Memory-map Mode.

flash_exit_xip	
Outline	Finish the Memory-map Mode to Octa Flash memory
Declaration	int flash_exit_xip (xspi_ctrl_t * const ctrl)
Description	- Call function for stopping the Memory-map Mode
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned
Note	-

Figure 9.27 shows a flowchart of the flash_exit_xip() implementation in the default IPL.

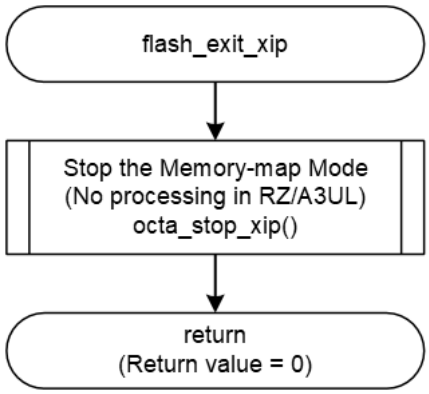


Figure 9.27 Flowchart of flash_exit_xip

9.3.1.6 flash_read

flash_read() is a function to read data from Octa Flash memory.

At IPL, Octa Flash memory operates in Memory-map mode.

Therefore, flash_read() implemented in the default IPL directly reads data from the space where Octa Flash memory is mapped (0x20000000 or later) without serial communication processing of the Read operation.

flash_read	
Outline	Data read from the Octa Flash memory
Declaration	int flash_read (xspi_ctrl_t * const ctrl, void * buffer, size_t address, size_t length)
Description	- Get address information where Octa Flash memory is mapped - Copy data from the Octa Flash memory to the buffer prepared by the application that called this function
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer to store read data address: Data read source address Specify the offset address in the Octa Flash memory for this argument length : Read data size
Return value	0 : This value is always returned
Note	-

Figure 9.30 shows a flowchart of the flash_read() implementation in the default IPL.

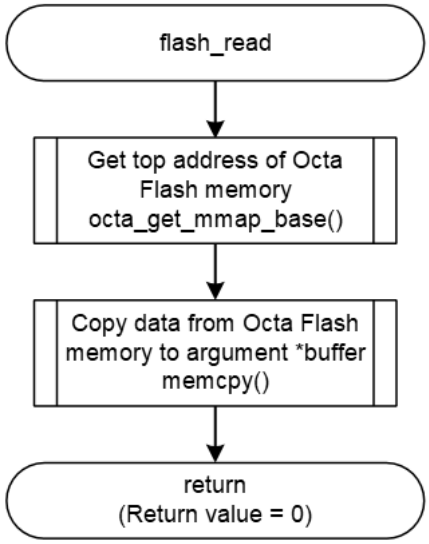


Figure 9.28 Flowchart of flash_read

9.3.1.7 flash_write

flash_write() is a function to write data to Octa Flash memory.

flash_write() implemented in the default IPL writes data by sending the Write operation of the MX66UW1G45G mounted on the SMARC EVK board.

flash_write	
Outline	Data write to the Octa Flash memory
Declaration	int flash_write (xspi_ctrl_t * const ctrl, void const * buffer, size_t address, size_t length)
Description	- Checking the write status - Waiting for the end of the write process - Release write protection - Updating the write operation table - Write data by sending write operation data to Octa Flash memory
Arguments	*ctrl : Pointer to the control structure for instance * buffer : Pointer to buffer where write data is stored address: Data write destination address Specify the offset address in the Octa Flash memory for this argument length : Write data size
Return value	0 : Data write is success -1 : Data write is failed
Note	Processing of this function is disabled at the default IPL. To enable it, refer to Figure 9.29 and rewrite RZ_XSPIFLASH_DISABLE_WRITE:=1 in rz_xspi_common.mk L13 to RZ_XSPIFLASH_DISABLE_WRITE:=0 and rebuild.

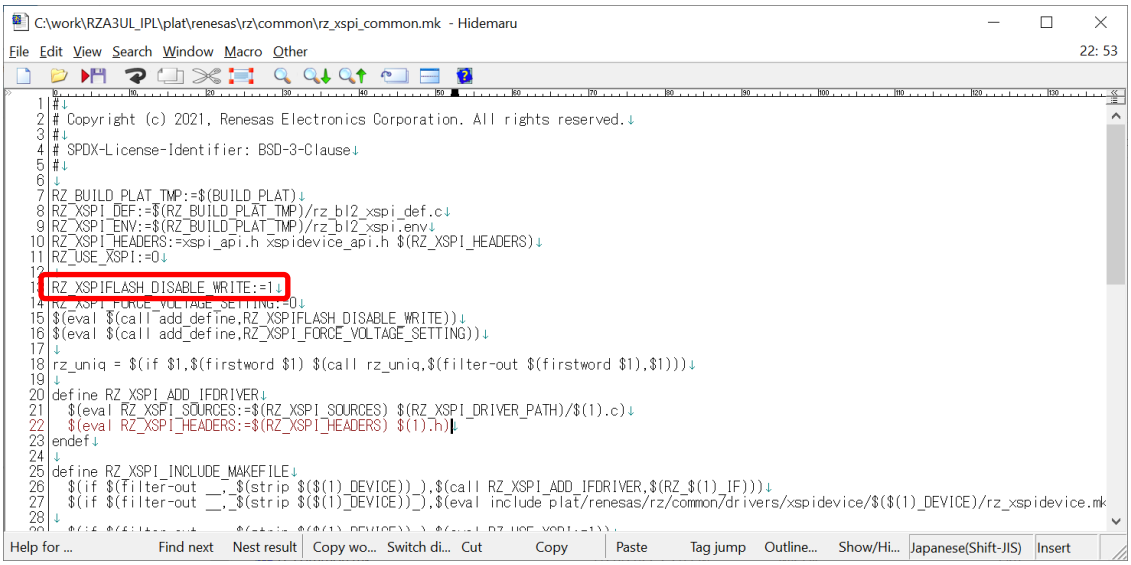


Figure 9.29 Change build switch for write function

Figure 9.30 and Figure 9.31 show the flowchart of the flash_write() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1. Wait process of write completion
- P2. Release Write Protect
- P3. Setting of Write operation table

These are shown as P1 to P3 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

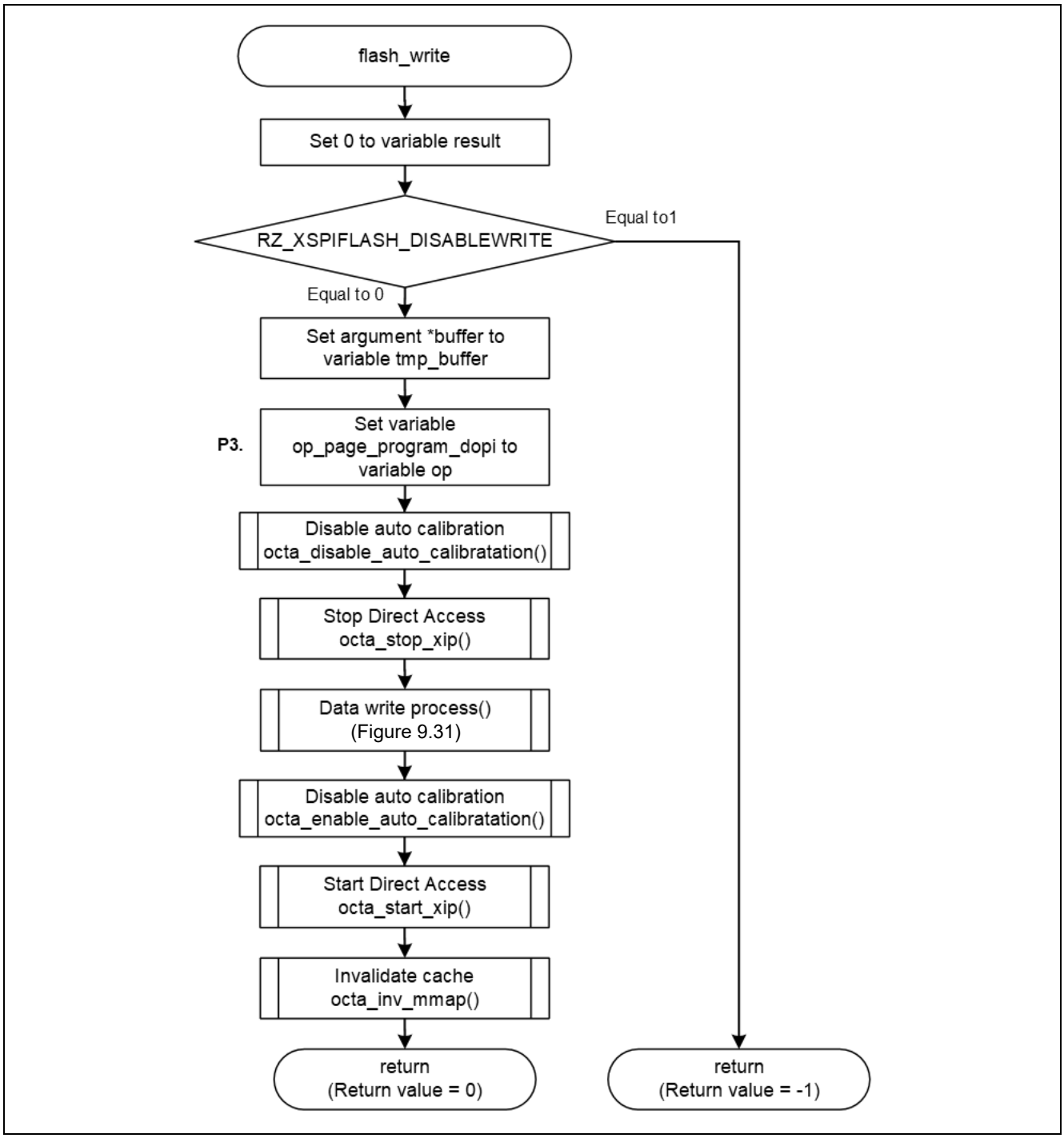


Figure 9.30 Flowchart of flash_write

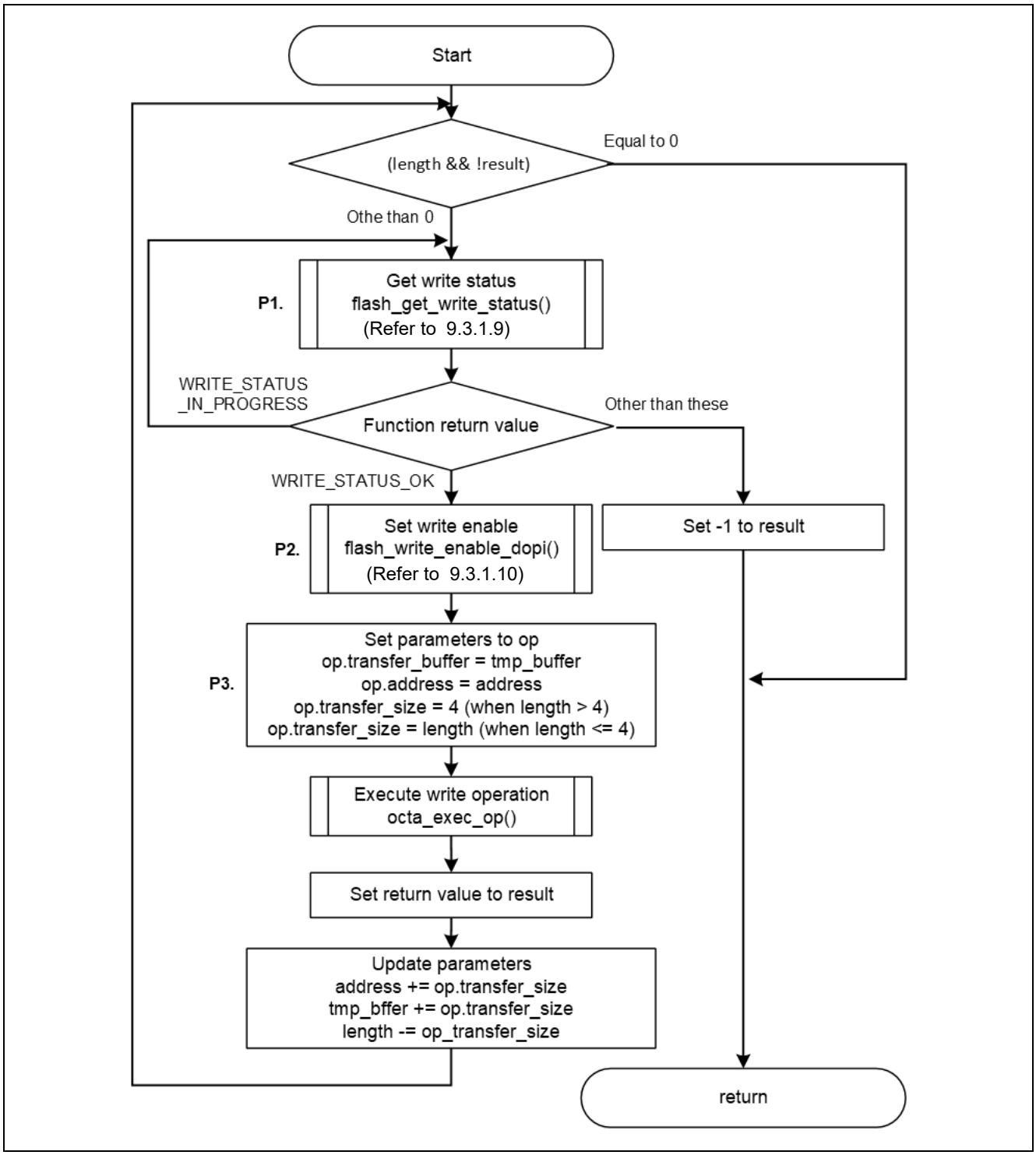


Figure 9.31 Flowchart of Data write process

P1. Wait process of write completion

Since the next data cannot be received while the Octa Flash memory is writing the data received by serial communication or erasing the data, IPL must check the write status and wait for the end.

The sub function flash_get_write_status() for getting the write status for the MX66UW1G45G is implemented in the Octa Flash memory driver implemented in the default IPL.

Since the method of checking the write status depends on the Octa Flash memory specifications, it is necessary to change the implementation of the function when changing the memory.

Please refer to 9.3.1.9 for change of checking method of write status.

P2. Release Write Protect

Most of Octa Flash memories require release of write protect when writing data or erasing data.

The Octa Flash memory driver implemented in the default IPL provides a sub function flash_write_enable_spi() / flash_write_enable_dopi() that release Write Protect for MX66UW1G45G.

Since the write protect release method depends on the specifications of the Octa Flash memory, it is necessary to change the implementation of the function when changing the memory.

Please refer to 9.3.1.10 for change of write protect release method.

P3. Setting Write operation table

When writing data to Octa Flash memory, data is written by sending a Write command.

The Octa Flash memory driver implemented in the default IPL sets the MX66UW1G45G's Page program 4B operation (code: 0x12ED) to the variable op_page_program_dopi as the Write operation and sends it.

Table 9.6 shows the settings of op_page_program_dopi when using Page program 4B.

When changing the Octa Flash memory, refer to Table 9.6 and rewrite the setting contents of op_page_program_dopi to the contents corresponding to the Write operation of your Octa Flash memory.

Note that the write destination address and the size of data that can be written with one write operation are limited by the operations implemented in the Octa Flash memory.

The Page program 4B operation requires writing to start from a 4byte aligned address.

Also, with this operation, the write data size per write is up to 4byte, so flash_write() repeatedly transfers data in units of 4 byte.

Similarly, adjust the write destination address and transfer data size per transfer according to the specifications of the Write operation that uses it.

Table 9.6 op_page_program_dopi setting contents using Page program 4B for DOPI Mode (0x12ED)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0x12ED	Command code "Page program (4B) (DOPI)"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	address	Address is serial flash memory address to data write It is updated in the function
address_is_ddr	true	Address transfer method (DDR)
address_size	4	32bit address
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	tmp_buffer	tmp_buffer is buffer address of write data It is updated in the function
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	length	length is remaining size of write data It is updated in the function
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.1.8 flash_erase

flash_erase() is a function to erase data from Octa Flash memory.

flash_erase() implemented in the default IPL erase data by sending the Erase operation of the MX66UW1G45G mounted on the SMARC EVK board.

flash_erase	
Outline	Erase the Octa Flash memory
Declaration	int flash_erase (xspidevice_ctrl_t * ctrl, size_t address, size_t length)
Description	- Checking the write status - Waiting for the end of the write process - Release write protection - Updating the erase operation table - Erase data by sending erase operation data to Octa Flash memory
Arguments	*ctrl : Pointer to the control structure for instance address: Start address for data erase Specify the offset address in the Octa Flash memory for this argument. If a value other than a multiple of 4Kbytes is set, the address obtained by aligning the specified address with 4Kbytes will be the start address of the erase area. length : erase size If a value other than 4Kbyte aligned size is set in this argument, it erases the data up to the 4Kbyte aligned address including the specified range.
Return value	0 : Data erase is success -1 : Data erase is failed.
Note	Processing of this function is disabled at the default IPL. To enable it, refer to Figure 9.29 and rewrite RZ_XSPIFLASH_DISABLE_WRITE:=1 in rz_xspi_common.mk L13 to RZ_XSPIFLASH_DISABLE_WRITE:=0 and rebuild.

Figure 9.32 and Figure 9.33 show the flowchart of the flash_erase() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1. Wait process of write completion
- P2. Release Write Protect
- P3. Send Erase operation

These are shown as P1 to P3 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

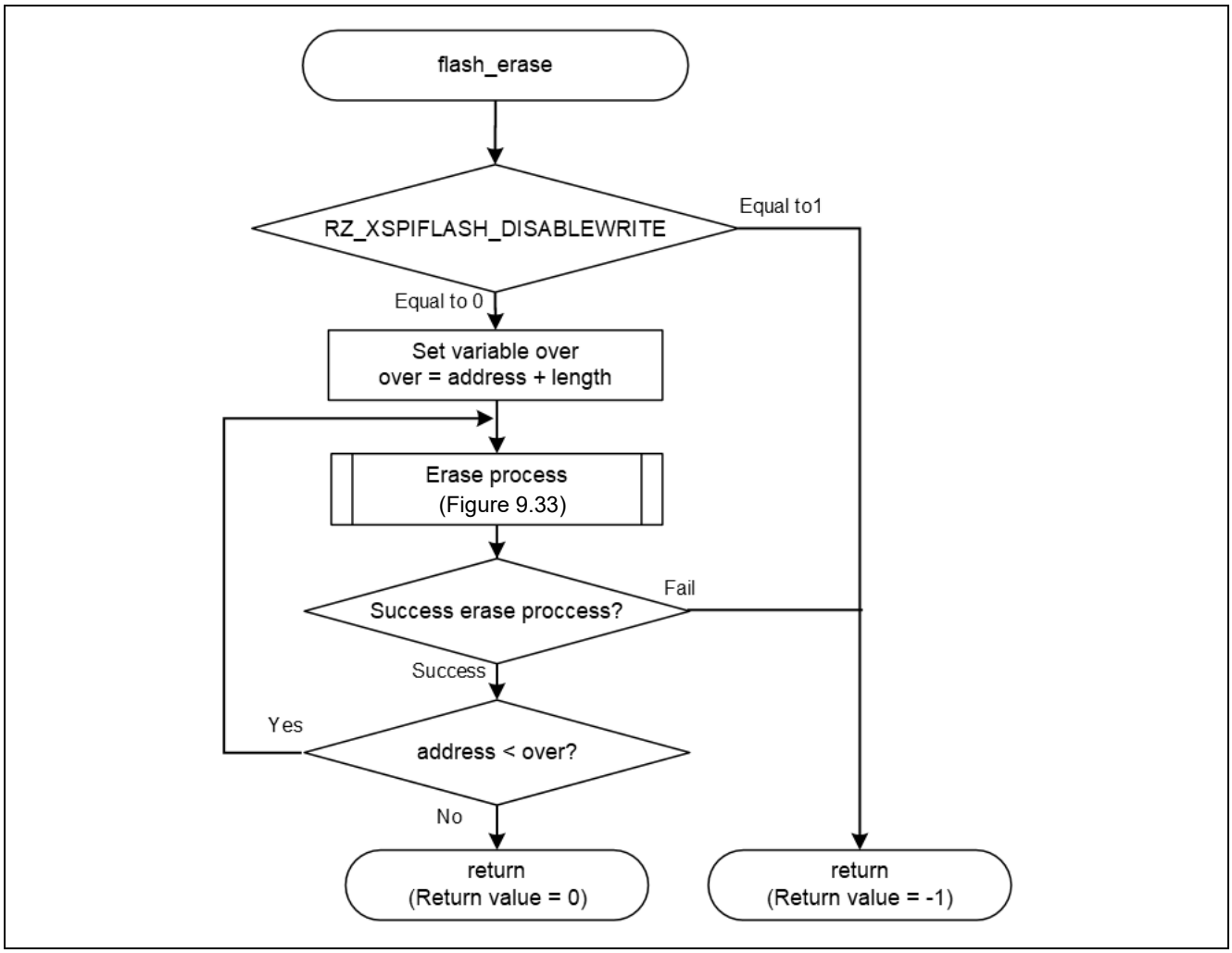


Figure 9.32 Flowchart of flash_erase

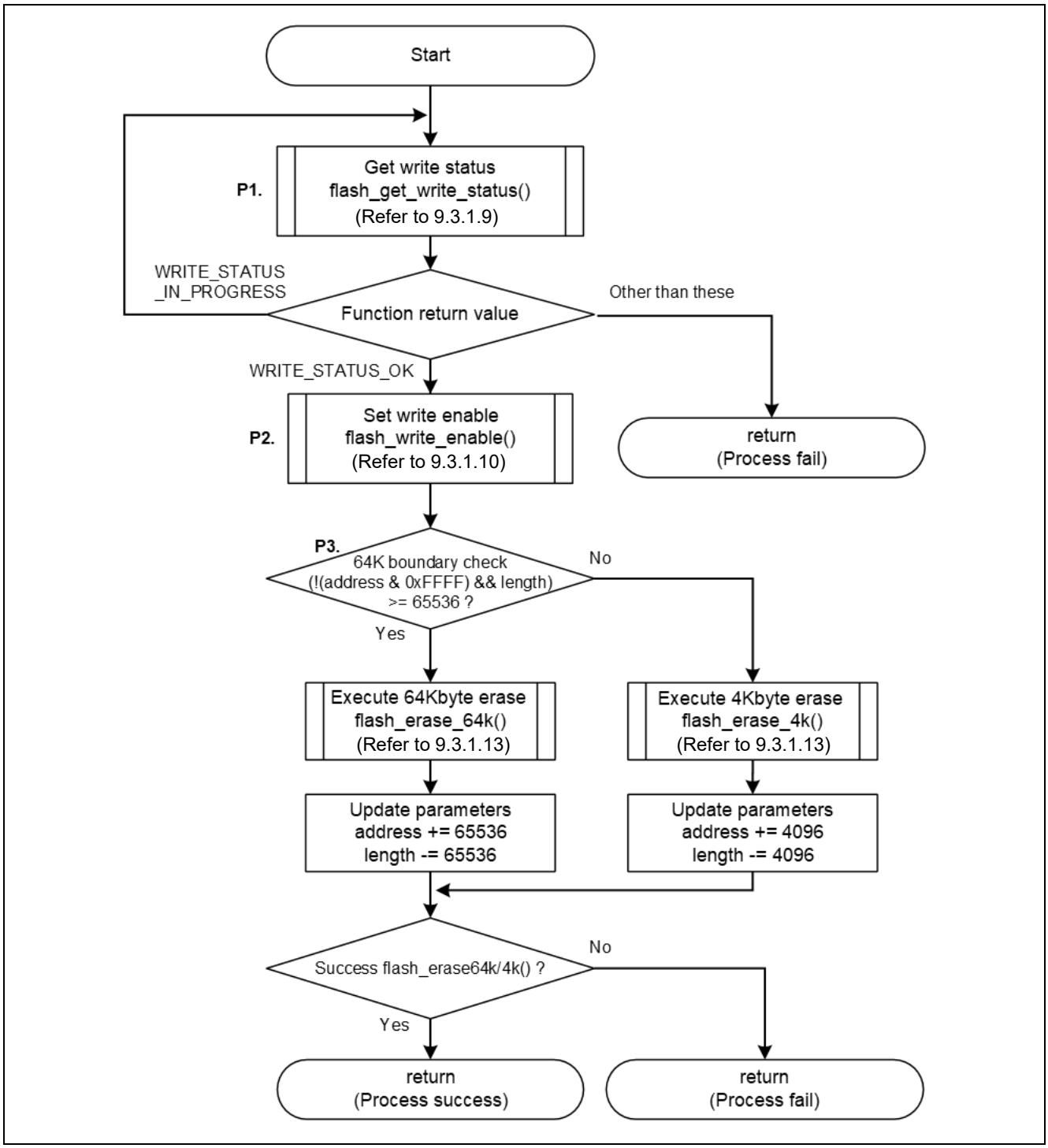


Figure 9.33 Flowchart of Erase process

P1. Wait process of write completion

Since the next data cannot be received while the Octa Flash memory is writing the data received by serial communication or erasing the data, IPL must check the write status and wait for the end.

The sub function flash_get_write_status() for getting the write status for the MX66UW1G45G is implemented in the Octa Flash memory driver implemented in the default IPL.

Since the method of checking the write status depends on the Octa Flash memory specifications, it is necessary to change the implementation of the function when changing the memory.

Please refer to 9.3.1.9 for change of checking method of write status.

P2. Release Write Protect

Most of Octa Flash memories require release of write protect when writing data or erasing data.

The Octa Flash memory driver implemented in the default IPL provides a sub function flash_write_enable_spi() / flash_write_enable_dopi() that release Write Protect for MX66UW1G45G.

Since the write protect release method depends on the specifications of the Octa Flash memory, it is necessary to change the implementation of the function when changing the memory.

Please refer to 9.3.1.10 for change of write protect release method.

3. Send Erase operation

When erasing data to Octa Flash memory, data is erased by sending a Erase command.

The MX66UW1G45G provides an Erase operation in 4Kbyte units (Sector erase 4KB (4kbytes)) and an Erase operation in 64Kbyte units (Sector erase 64KB (64kbytes)).

Therefore, the Octa Flash memory driver implemented in the default IPL performs a 64Kbyte erase when the remaining erase size is 64Kbyte or more, and a 4Kbyte erase when it is less than 64Kbyte.

These erase operations are executed by the sub functions flash_erase_4k() / flash_erase_64k() corresponding to each operation.

Refer to 9.3.1.13 for how to change the erase operation execution process.

The size unit that can be erased depends on the operations implemented in the Octa Flash memory.

Adjust the erase unit of the program according to the erase unit of the erase operation provided by your Octa Flash memory.

9.3.1.9 flash_get_write_status

flash_get_write_status() is a function to get the write status of the Octa Flash memory.

flash_get_write_status() implemented in the default IPL checks the write status by reading the Status Register of the MX66UW1G45G mounted on the SMARC EVK board.

flash_get_write_status	
Outline	Gets the write status of the Octa Flash memory
Declaration	int flash_get_write_status (xspidevice_ctrl_t * ctrl)
Description	- Read Status Register - Return from a function with a return value that depends on the value of the Status Register
Arguments	*ctrl : Pointer to the control structure for instance
Return value	WRITE_STATUS_FAIL : Data write is failed WRITE_STATUS_IN_PROGRESS : Data write in progress WRITE_STATUS_OK : Data write is complete
Note	-

Figure 9.34 shows a flowchart of the flash_get_write_status() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Read Status Register

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

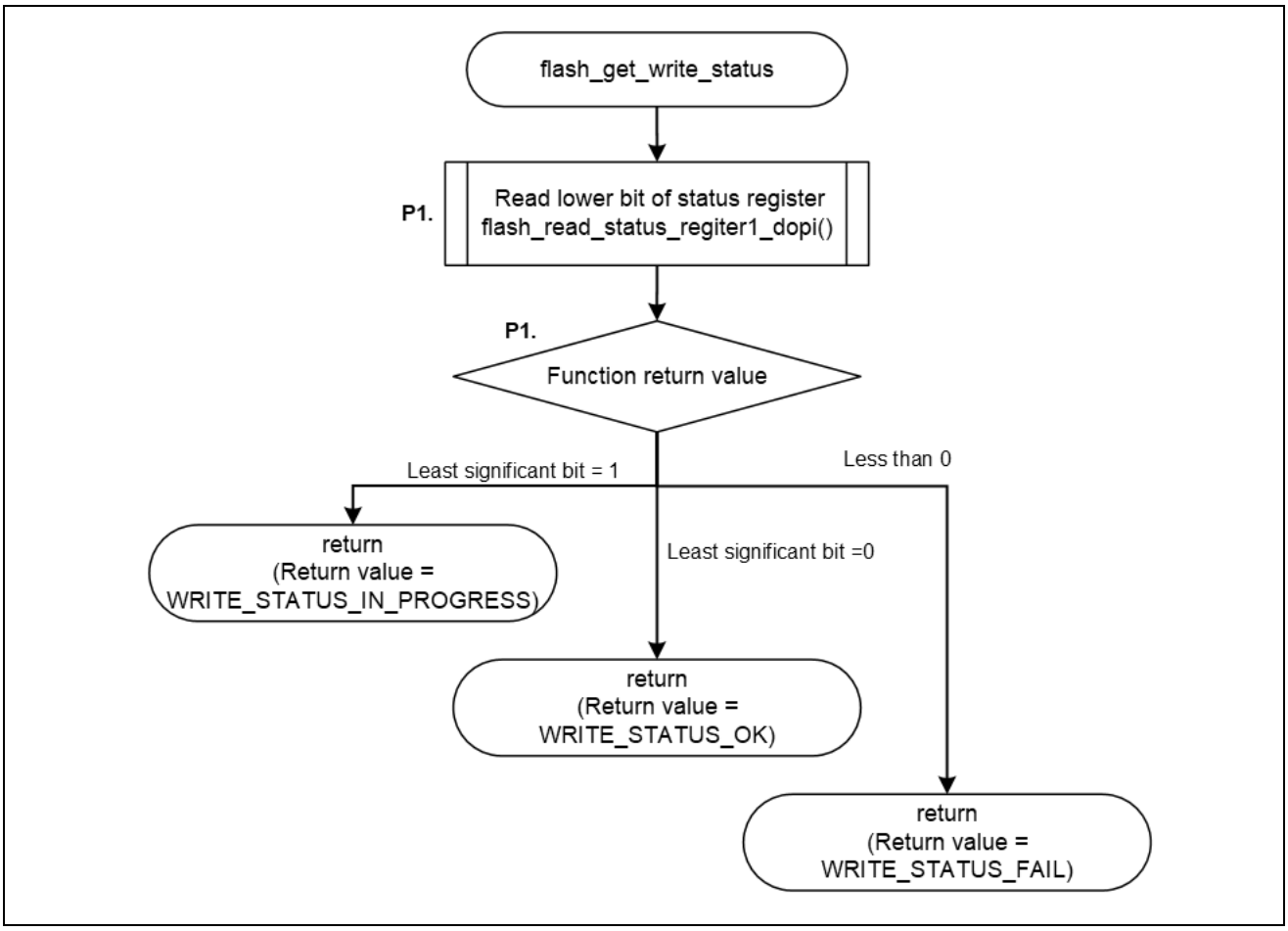


Figure 9.34 Flowchart of flash_get_write_status

P1. Read Status Register

MX66UW1G45G has Status Register shown in Table 9.7.

The WIP bit in this register indicates whether it is in Write (Erase) or idle.

Therefore, the Octa Flash memory driver implemented at default IPL reads the Status Register.

If the value of the Status Register is checked and the WIP bit is 1, a return value indicating that processing is in progress is returned, and if it is 0, a return value indicating that processing has ended is returned.

The default IPL provides a sub-function flash_read_status_register1_dopi() that reads the Status Register of the MX66UW1G45G.

Refer to 9.3.1.9 for how to read the Status Register.

When changing the Octa Flash memory, read the register corresponding to the MX66UW1G45G Status Register and change the source code so that the bit indicating the Erase or Write status is checked to determine the status.

Table 9.7 BUSY bit of AT25QL128A Status Register (BIT7 – Bit 0)

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Reserved	Reserved	BP3 (level of protected block)	BP2 (level of protected block)	BP1 (level of protected block)	BP0 (level of protected block)	WEL (write enable latch)	WIP (write in progress bit)
Reserved	Reserved	Set the ratio of protect block and non-protect block				1=write enable 0=not write enable	1=write operation 0=not in write operation

9.3.1.10 flash_write_enable_spi / flash_write_enable_dopi

flash_write_enable_spi() / flash_write_enable_dopi() are the function to release Write Protect of Octa Flash memory.

These functions implemented in the default IPL release Write Protect by sending the Write Enable operation of the MX66UW1G45G mounted on the SMARC EVK board.

flash_write_enable_spi() sends a Write Enable command in SPI mode (1bit data bus width communication), and flash_write_enable_dopi() sends a Write Enable command in DOPI mode (DDR communication and 8bit data bus width communication).

The only difference between these functions is the instructions used, and the function definition and flow are the same.

flash_write_enable_spi / flash_write_enable_dopi	
Outline	Set Write enable to MX66UW1G45G
Declaration	int flash_write_enable_spi (octaflash_mx66uw_ctrl_t* myctrl) Int flash_write_enable_dopi (octaflash_mx66uw_ctrl_t* myctrl)
Description	- Send Write Enable operation to Octa Flash memory
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation success -1 : Operation Failed
Note	-

Figure 9.35 shows a flowchart of the flash_write_enable_spi() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Send Write Enable operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

For flash_write_enable_dopi (), replace spi with dopi and refer to it.

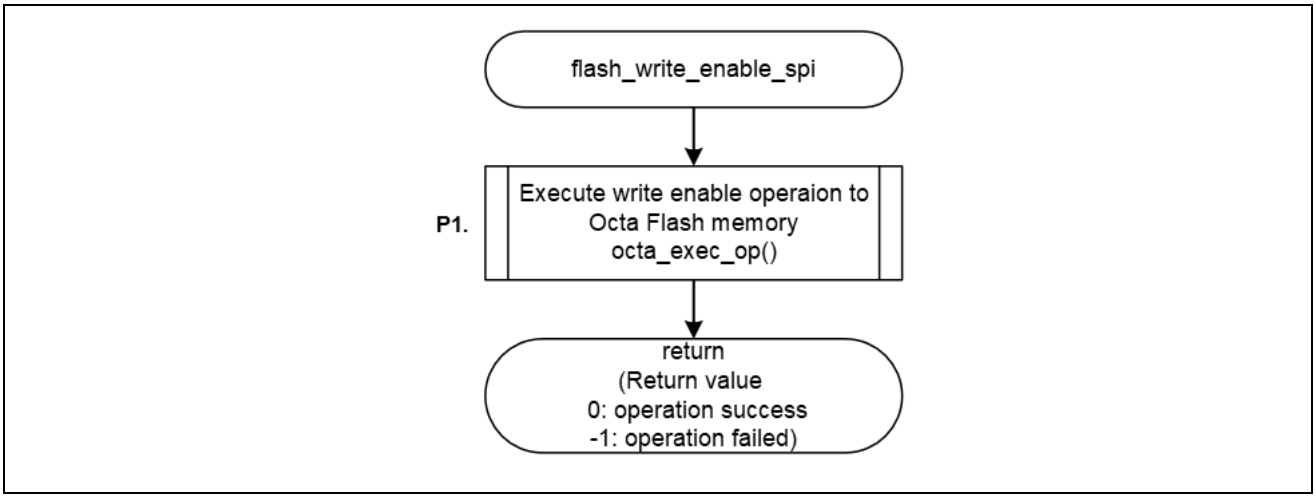


Figure 9.35 Flowchart of flash_write_enable_spi

P1. Send Write Enable operation

The Write Enable setting to the Octa Flash memory is performed by sending the Write Enable command.

The Octa Flash memory driver implemented in the default IPL sets the MX66UW1G45G Write Enable command (code: 0x06) to the variable op_write_enable_spi as the Write Enable operation for SPI mode, and the Write Enable information is sent by serial communication.

Table 9.8 shows the op_write_enable_spi settings when using the Write Enable operation.

When changing the Octa Flash memory, refer to Table 9.8 and rewrite the setting contents of op_write_enable to the contents equivalent to the Write Enable instruction of your Octa Flash memory.

In addition, there are Octa Flash memories that require register access in a specific procedure when executing Write Enable.

In such cases, execute the operation transfer flow according to the determined procedure.

When using the Write Enable command (code: 0x06F9) in DOPI mode, set the variable op_write_enable_dopi and send the Write Enable information.

Table 9.9 shows the settings for op_write_enable_dopi.

Table 9.8 op_write_enable_spi setting contents when using Write Enable (0x06) for SPI Mode

Member	Setting value	Description
form	SPI_FORM_1_1_1	Number of I/O line used for command: 1 line Number of I/O line used for address: 1 line Number of I/O line used for data: 1 line
op	0x06	Command code for SPI mode "Write Enable"
op_size	1	Command size = 1byte
op_is_ddr	false	Command transfer method (SDR)
address	0	No address
address_is_ddr	false	Address transfer method (SDR)
address_size	0	No address
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	NULL	No data
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	0	No data
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 2cycle
clsh_value	5	SSL Negation Delay = 7cycle
shsl_value	1	Next Access Delay = 5cycle

Table 9.9 op_write_enable_dopi setting contents when using Write Enable (0x06F9) for DOPI Mode

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0x06F9	Command code for DOPI mode "Write Enable"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	No address
address_is_ddr	true	Address transfer method (DDR)
address_size	0	No address
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	NULL	No data
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	No data
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.1.11 flash_read_status_reister_spi / flash_read_status_register_dopi

flash_read_status_register_spi() / flash_read_status_register_dopi() are the function to read Status Register value of MX66UW1G45G mounted on the SMARC EVK board.

These functions implemented in the default IPL read value from Status Register by sending the Read Status Register operation of the MX66UW1G45G.

flash_read_status_register_spi () sends a Write Enable command in SPI mode (1bit data bus width communication), and flash_read_status_register_dopi() sends a Write Enable command in DOPI mode (DDR communication and 8bit data bus width communication).

The only difference between these functions is the instructions used, and the function definition and flow are the same.

flash_read_status_register_spi / flash_read_status_register_dopi	
Outline	Gets the status register of the MX66UW1G45G
Declaration	int flash_read_status_register_spi (octaflash_mx66uw_ctrl_t * myctrl) int flash_read_status_register_dopi (octaflash_mx66uw_ctrl_t * myctrl)
Description	- Send Read Status Register operation to Octa Flash memory and get register value
Arguments	*myctrl : Pointer to the control structure for instance
Return value	Positive number : Operation success, and status register value -1 : Operation Failed
Note	-

Figure 9.36 shows a flowchart of the flash_read_status_register_spi() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Send Read Status Register operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

For flash_write_enable_dopi(), replace spi with dopi and refer to it.

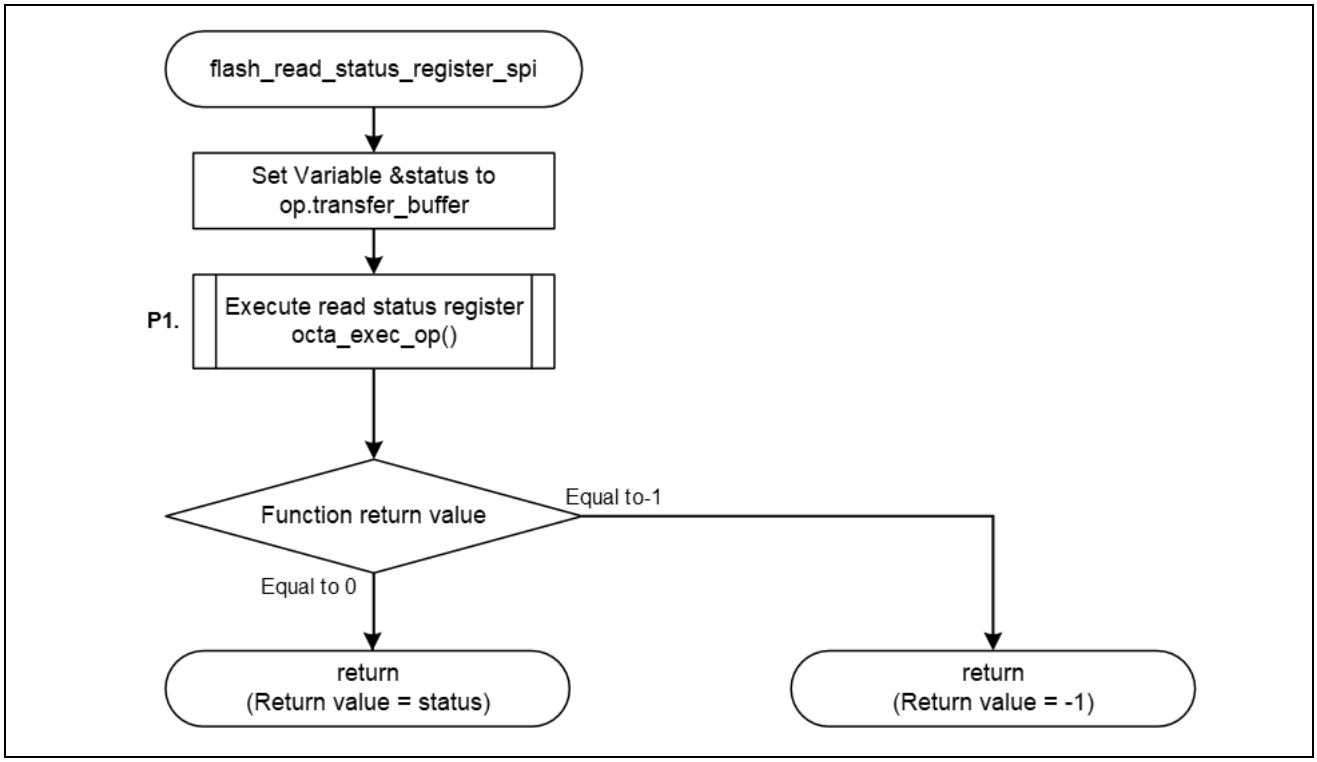


Figure 9.36 Flowchart of flash_read_status_register_spi

P1. Send Read Status Register operation

Send the Read Status Register operation to get the Status Register value from the Octa Flash memory.

The flash_read_status_register_spi() function sets the Read Status Register command (0x05) for SPI Mode to the variable op_read_status_spi and passes it as an argument to octa_exec_op() to execute the command.

Table 9.10 shows the settings of op_read_status_spi.

When changing the Octa Flash memory, rewrite the variable members according to the specifications of the command equivalent to the MX66UW1G45G's Read Status Register operation implemented in your Octa Flash memory.

When using the Read Status Register command (code: 0x05FA) for DOPI Mode, set the variable op_write_enable_dopi and send the Read Status Register operation.

Table 9.11 shows the settings of op_write_enable_dopi.

Table 9.10 op_read_status_spi setting contents when using Read Status Register (0x05) for SPI Mode

Member	Setting value	Description
form	SPI_FORM_1_1_1	Number of I/O line used for command: 1 line Number of I/O line used for address: 1 line Number of I/O line used for data: 1 line
op	0x05	Command code for SPI mode "Read Status Register"
op_size	1	Command size = 1byte
op_is_ddr	false	Command transfer method (SDR)
address	0	No address
address_is_ddr	false	Address transfer method (SDR)
address_size	0	No address
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	&status	Set pointer of argument status in function
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	1	Status Register size = 1byte
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 2cycle
clsh_value	5	SSL Negation Delay = 7cycle
shsl_value	1	Next Access Delay = 1cycle

Table 9.11 op_read_status_dopi setting contents when using Read Status Register (0x05FA) for DOPI Mode

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0x05FA	Command code for DOPI mode "Read Status Register"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	Register address = 0
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Address size = 4 byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	4	Dummy cycles = 4 cycles
transfer_buffer	&status	Set pointer of argument status in function
transfer_is_ddr	false	Data transfer method (DDR)
transfer_size	1	Status register size = 1byte
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.1.12 flash_write_config_register2_spi

flash_write_config_reister2_spi() is a function that writes value to the MX66UW1G45G Configuration Register2.

flash_write_config_reister2_spi() implemented in the default IPL write value to the Configuration Register2 by sending the Write Configuration Register2 operation of the MX66UW1G45G mounted on the SMARC EVK board.

flash_get_write_status	
Outline	Set value to Configuration Register2
Declaration	int flash_write_config_register2_spi (octaflash_mx66uw_ctrl_t * myctrl, uint32_t add, uint8_t status)
Description	- Send Write Configuration operation and write value to Register
Arguments	*myctrl : Pointer to the control structure for instance add : Address of Configuration Register status : Setting value to Configuration Register
Return value	0 : Operation success -1 : Operation Failed
Note	-

Figure 9.37 shows a flowchart of the flash_write_config_reister2_spi() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Send Write Configuration Register operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

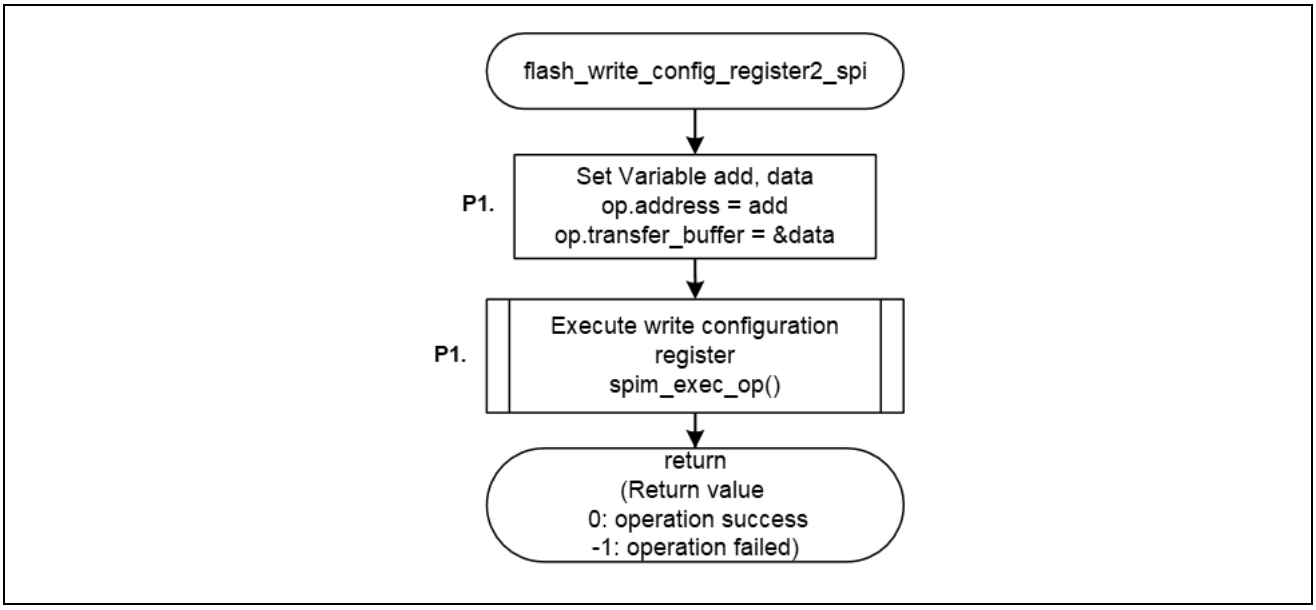


Figure 9.37 Flowchart of flash_write_config_register2_spi

P1. Send Write Configuration Register operation

Send a Write Configuration Register2 command to write the value to the Configuration Register2.

The flash_write_config_regster2_spi() function sets the Write Configuration Register2 command (0x72) to the variable op_write_config2_spi and sends the Write Configuration Register2 operation.

Table 9.12 shows the settings of op_write_config2_spi when using the Write Configuration Register2 operation.

When changing the Octa Flash memory, rewrite the variable members according to the specifications of the operation equivalent to the MX66UW1G45G Write Configuration Register2 operation implemented in your Octa Flash memory.

Table 9.12 op_write_config2_spi setting contents when using Write Configuration Register2 (0x72) for SPI Mode

Member	Setting value	Description
form	SPI_FORM_1_1_1	Number of I/O line used for command: 1line Number of I/O line used for address: 1line Number of I/O line used for data: 1line
op	0x72	Command code for SPI mode "Write Configuration Register2"
op_size	1	Command size = 1byte
op_is_ddr	false	Command transfer method (SDR)
address	add	Set add of argument in function
address_is_ddr	false	Address transfer method (SDR)
address_size	4	4byte address
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	&data	Pointer of argument data in function
transfer_is_ddr	false	Data transfer method (SDR)
transfer_size	1	Write data = 1byte
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 2cycle
clsh_value	5	SSL Negation Delay = 7cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.1.13 flash_erase_4k / flash_erase_64k

flash_erase_4k() / flash_erase_64k() are functions to erase data in units of 4Kbyte and 64Kbyte respectively from MX66UW1G45G implemented in SMARC EVK board.

flash_erase_4k() / flash_erase_64k() implemented in the default IPL send Sector Erase 4KB, Sector Erase 64KB commands of MX66UW1G45G to erase data.

These functions are similar in function definition and process flow.

flash_erase_4k / flash_erase_64k	
Outline	Erase data by 4Kbytes / 64Kbytes
Declaration	int flash_erase_4k (octaflash_mx66uw_ctrl_t * myctrl) int flash_erase_64k (octaflash_mx66uw_ctrl_t * myctrl)
Descrition	- Check size of data to be erased - Updating the erase operation table - Erase data by sending erase operation
Arguments	*myctrl : Pointer to the control structure for instance
Return value	0 : Operation success -1 : Operation Failed
Note	-

Figure 9.38 shows a flowchart of the flash_erase_4k() / flash_erase_64k() implementation in the default IPL.

The processing that changes the implementation when changing the Octa Flash memory is shown below.

- P1 Change parameter of Erase operation table

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

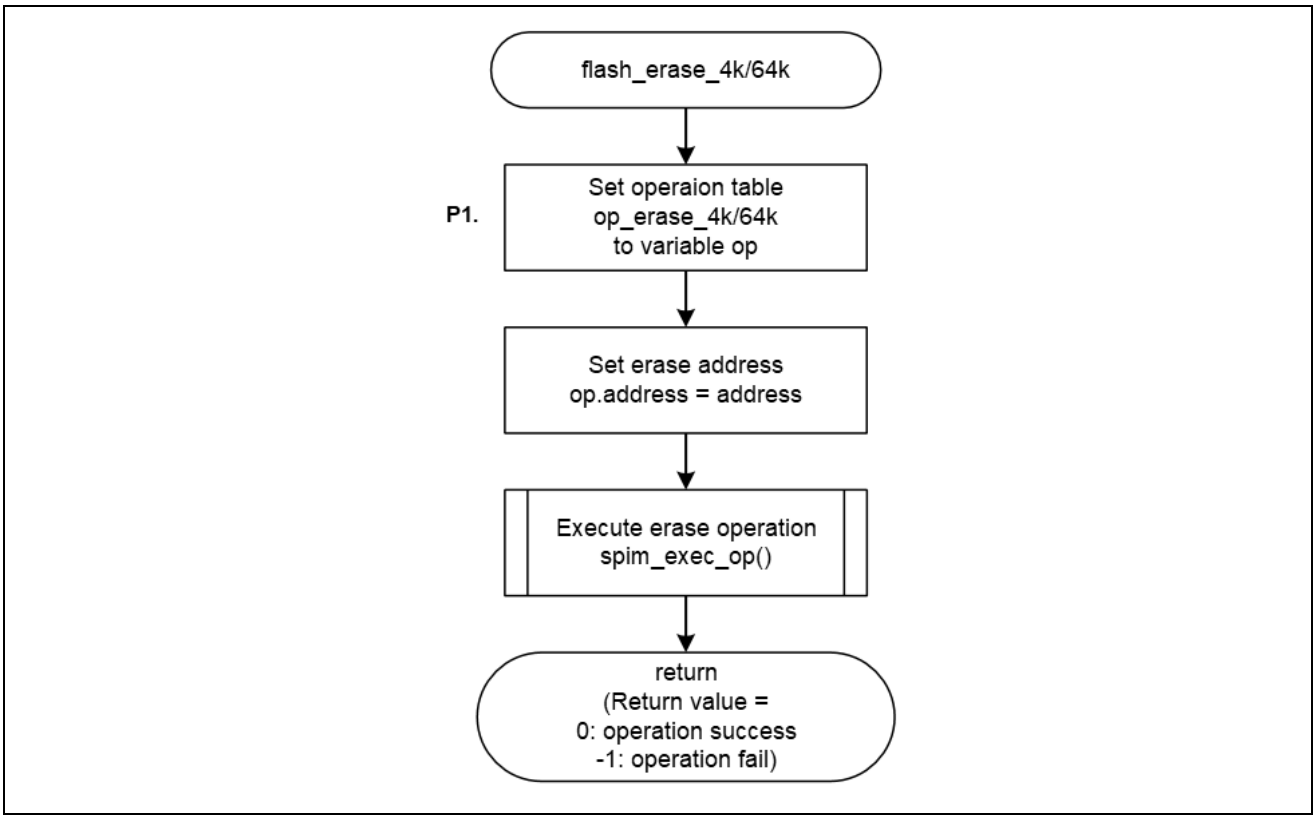


Figure 9.38 Flowchart of flash_erase_4k/64k

P1. Change parameter of Erase operation table

The flash_erase_4k() function implemented in the default IPL sets the Sector Erase 4KB command (0x21DE) to the variable op_erase_4k and sends the operation.

Similarly, in the flash_erase_64k() function, set the Sector Erase 64KB command (0xDC23) to the variable op_erase_64k and send the operation.

By doing this, the data is erased by sending the Erase command in each function.

Table 9.13 shows the op_erase_4k settings when using the Sector Erase 4KB operation, and Table 9.14 shows the op_erase_64k settings when using the Sector Erase 64KB operation.

When changing the Octa Flash memory, rewrite each variable to the contents corresponding to the Erase operation implemented in that memory.

Table 9.13 op_erase_4k setting contents when using Sector Erase 4KB (0x21DE) for DOPI Mode

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0x21DE	Command code for DOPI mode "Sector Erase 4KB"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	address	Set address of argument
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Erase address size = 4byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	NULL	No data
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	No data
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

Table 9.14 op_erase_64k setting contents when using Sector Erase 64KB (0xDC23) for DOPI Mode

Member	**Setting value**	**Description**
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8line
op	0xDC23	Command code for DOPI mode "Sector Erase 64KB"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	address	Set address of argument
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Erase address size = 4byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	NULL	No data
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	No data
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	5	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.2 How to customize IPL for your Octa RAM

This chapter describes how to customize the IPL according to your Octa RAM.

First, the method of controlling the Octa RAM in IPL is shown.

The IPL is implemented the Octa RAM driver and the Octa Memory Controller driver to control the Octa RAM.

The Octa RAM driver manages the execution of operations of the Octa RAM.

The Octa Memory Controller driver is a driver for controlling serial communication between RZ/A3 and Octa RAM.

The Octa RAM driver creates an operation table consisting of parameters for executing Octa RAM operations and passes it to the Octa Memory Controller driver.

The Octa Memory Controller driver controls HW according to the received operation table and performs serial communication.

Since the Octa RAM driver is an Octa RAM-dependent implementation, it must be changed according to your Octa RAM.

The Octa Memory Controller driver implementation, on the other hand, does not rely on Octa RAM.

Therefore, there is no need to change even when changing the Octa RAM.

Figure 9.39 shows the operation image diagram of each driver.

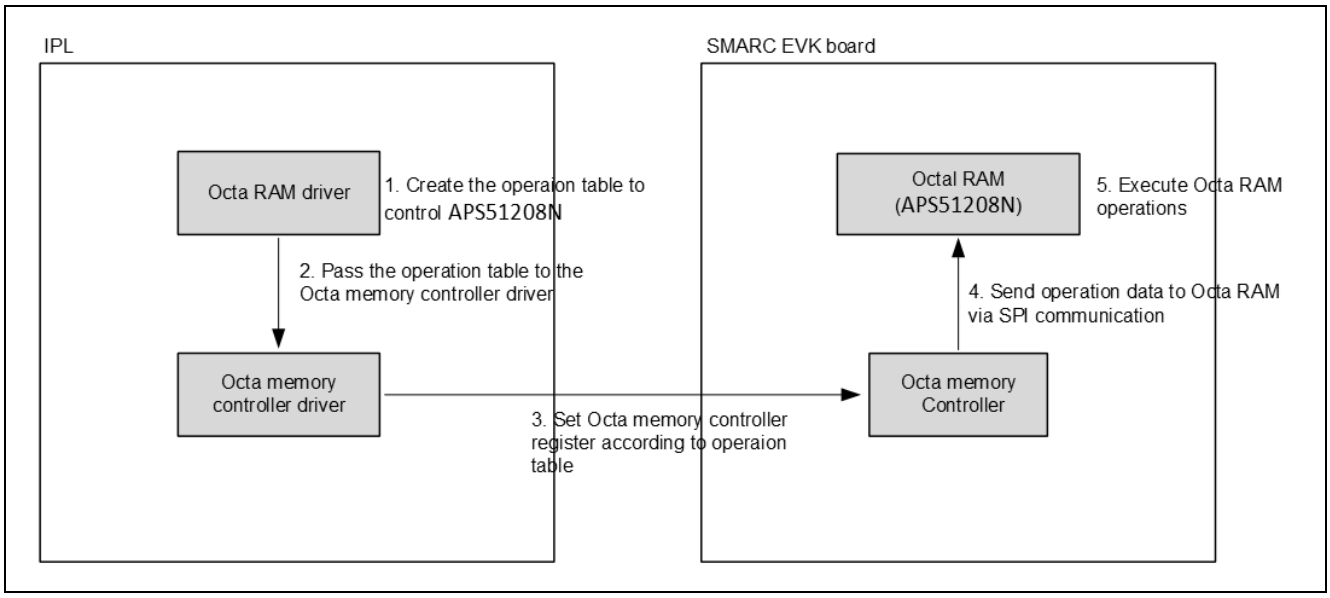


Figure 9.39 Operating image of Octa RAM driver and Octa Memory Controller driver

1. The Octa RAM driver creates an operation table corresponding to your memory.
2. The Octa RAM driver calls the Octa Memory Controller driver.
   At this time, the operation table is passed to the Octa Memory Controller driver.
3. The Octa Memory Controller driver sets the Octa Memory Controller register according to the received operation table parameters.
4. Octa Memory Controller performs serial communication with the Octa RAM according to the settings made in step3 and sends operation data to the Octa RAM.
5. The Octa RAM controls itself according to the received operation data.

The relationship between operation table parameters and transfer formats is the same as for Octa Flash memory.

The relationship is shown in Figure 9.15.

Shows how to add the Octa Flash memory driver for the new Octa Flash memory.

1.Open "c:\work\RZA3UL_IPL\plat\renesas\rz\common\drivers\xspidevice" folder, copy the "octaram_apsxx" folder and rename it to any name.
This time, rename the folder name to "octaram_custom".

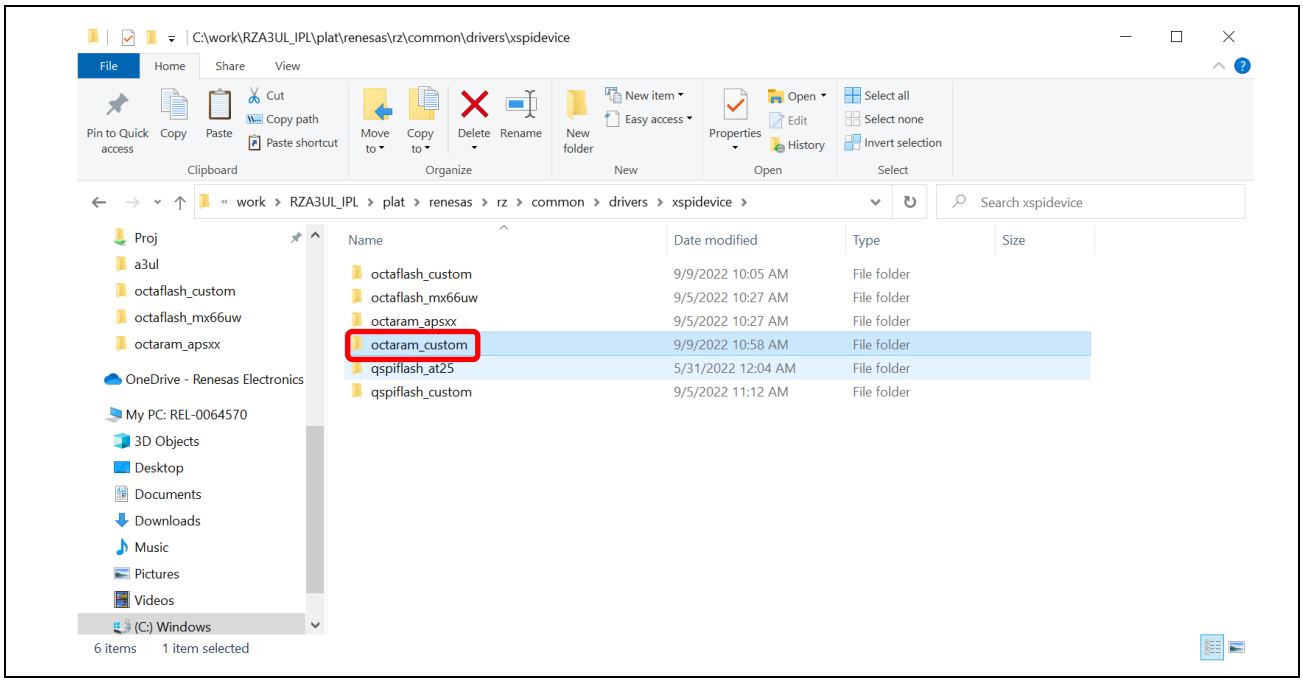


Figure 9.40 Rename Octa RAM driver folder

2. Open "octaram_custom" folder and change "octaram_apsxx.c", "octaram_apsxx_api.h" to "octaram_custom.c", "octaram_custom_api.h".

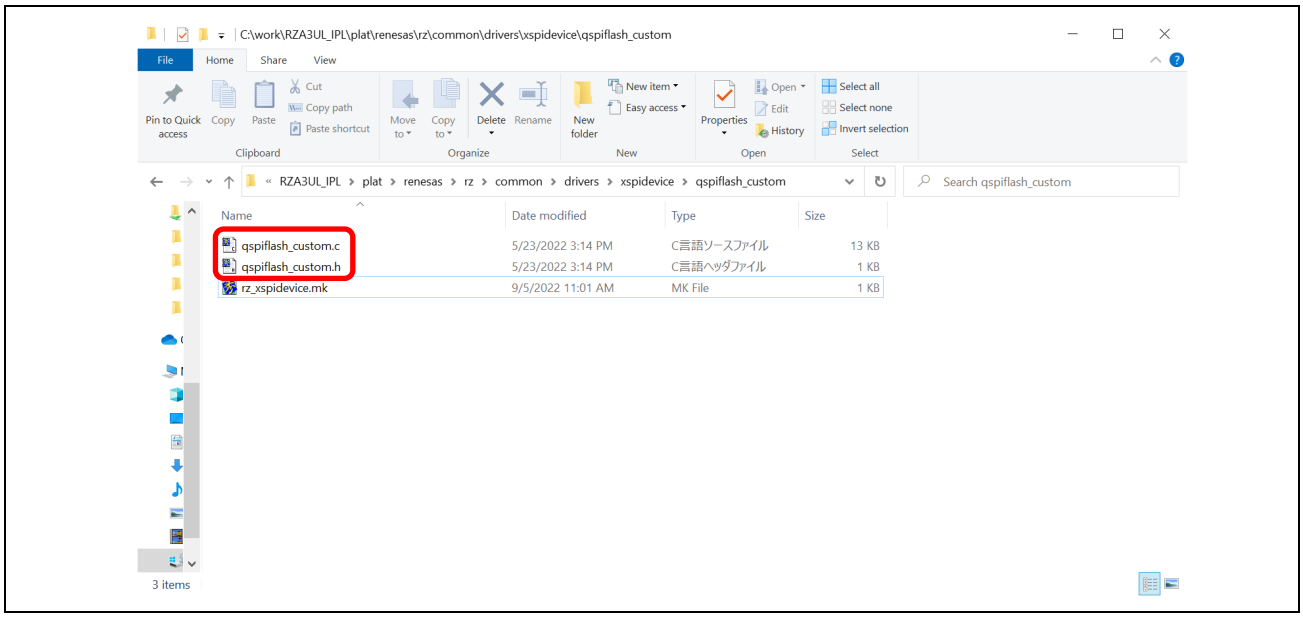


Figure 9.41 Rename Octa RAM driver files

3. Open "rz_xspidevice.mk" and change "_XSPI_DEVICE:=octaram_apsxx" on line 6 to "_XSPI_DEVICE:=octaram_custom".

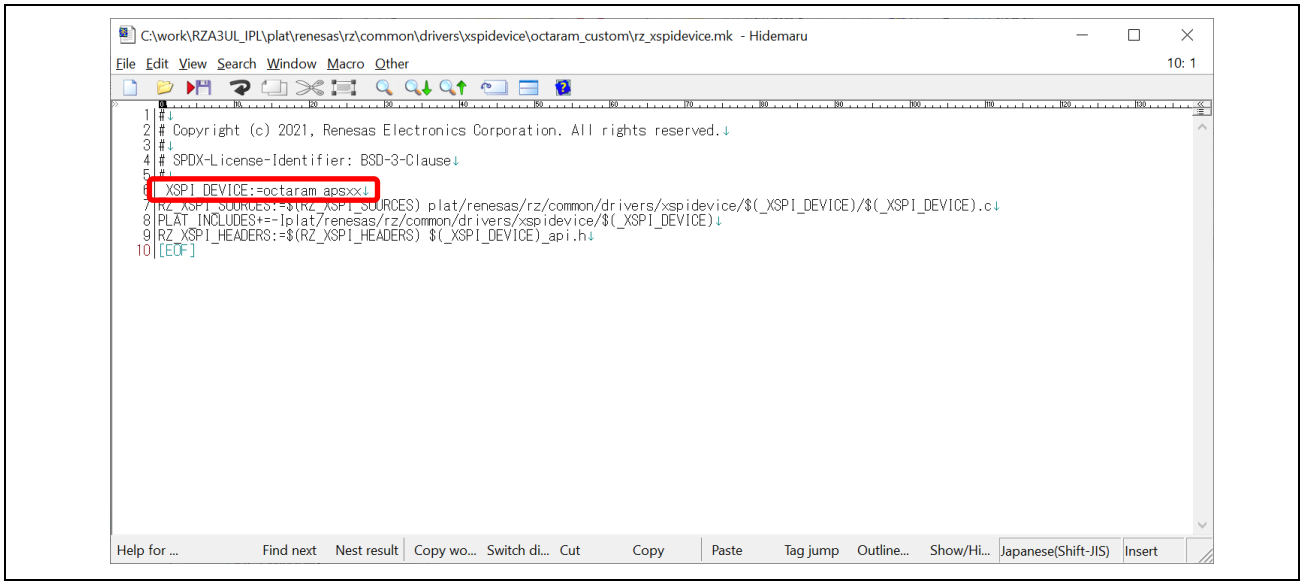
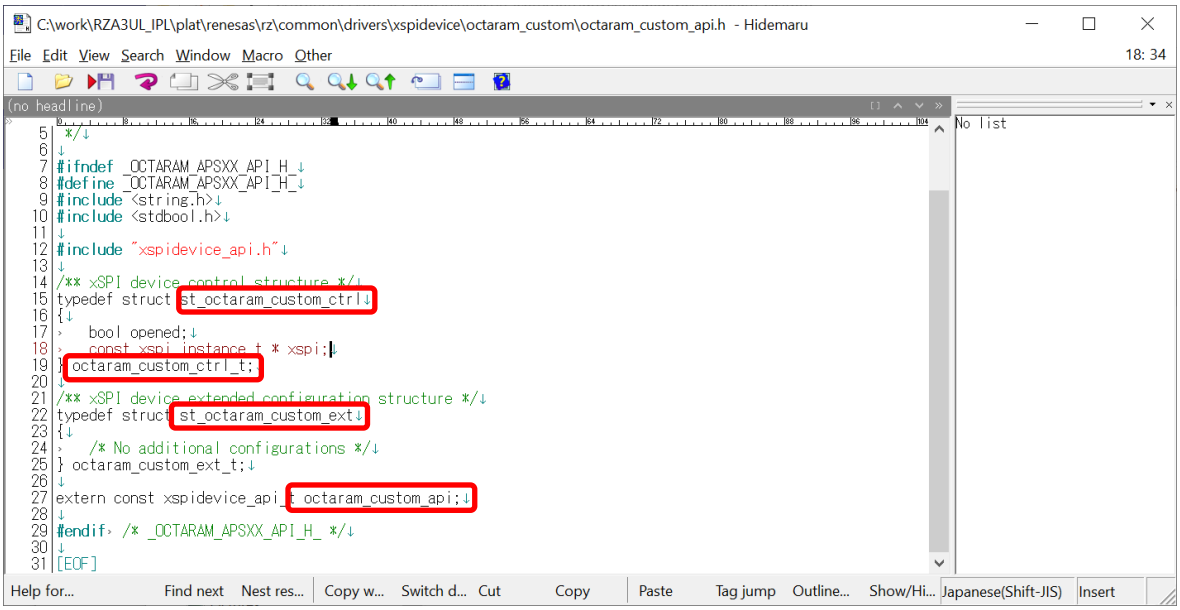


Figure 9.42 Change Octa RAM driver Makefile

4. Open "octaram_custom_api.h" and replace all structure definitions of "xx_octarm_apsxx_xx" with "xx_octaram_custom_xx".



Note: The octaram_custom_ext_t structure is for extended settings.
If the members corresponding to the parameters required for your Octa RAM are not provided in the octaram_custom_t structure, add those members a sextended settings.

Figure 9.43 Change Octa RAM driver structs name

5. Open octaram_custom.c and replace the included header with "octaram_custom_api.h". Also, reflect the structure name replaced in step 4.

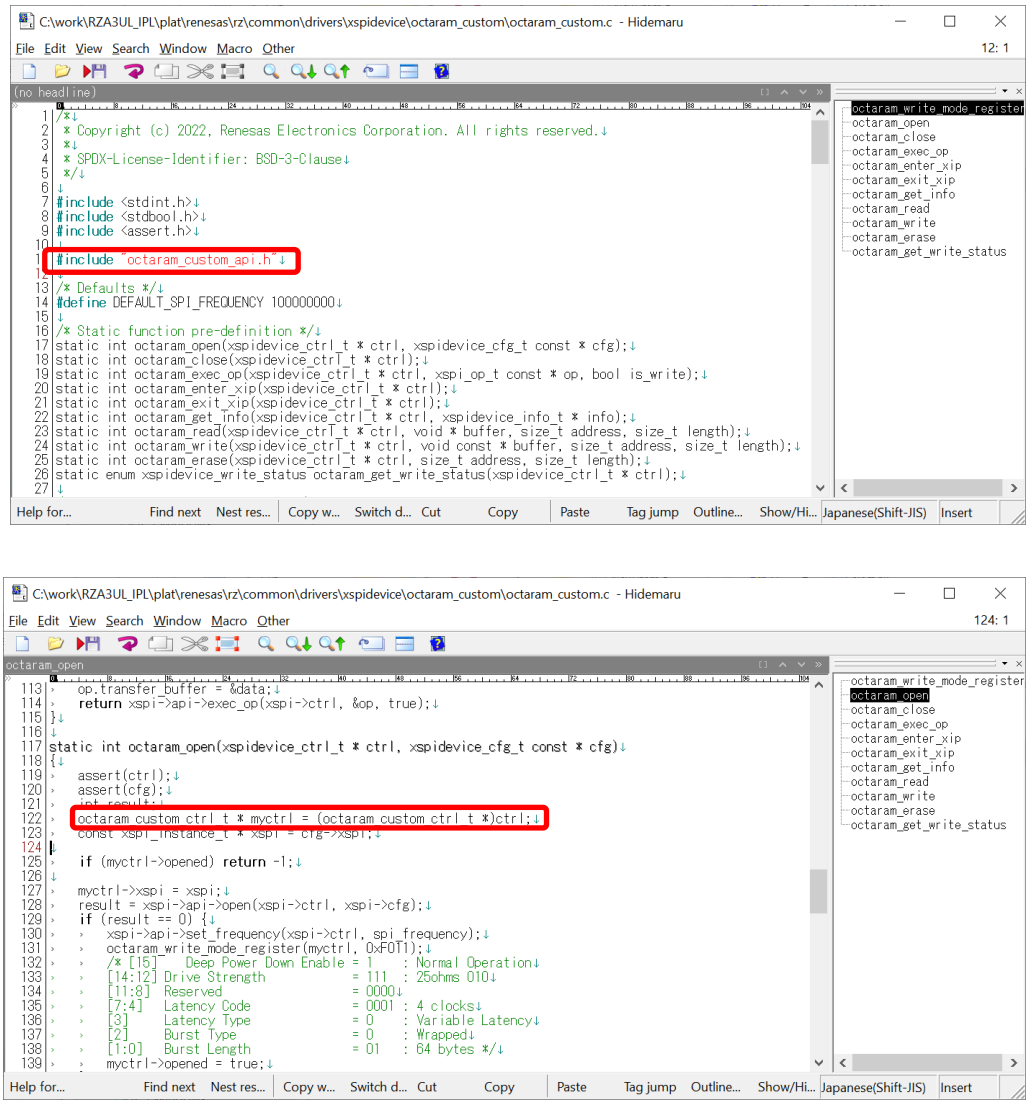


Figure 9.44 Change Octa RAM driver source code

6. Change the implementation of API functions in Table 9.15 and Sub functions in Table 9.16 Table 9.1 according to your Octa RAM.
   9.3.2.1 to 9.3.2.10 show the specifications of each function implemented in the default IPL and the change points when changing the Octa RAM.
   Please refer to these and change the implementation of each function.

Table 9.15 Octa RAM driver API Function

Function Name	Description
octaram_open	Octa RAM driver open function
octaram_close	Octa RAM driver close function
octaram_exec_op	Octa RAM driver operation function
octaram_enter_xip	Octa RAM driver Memory-map Mode to memory start function
octaram_exit_xip	Octa RAM driver Memory-map Mode to memory stop function
octaram_get_info	Octa RAM driver information get function
octaram_read	Octa RAM driver read function
octaram_write	Octa RAM driver write function
octaram_erase	Octa RAM driver erase function
octaram_get_write_status	Octa RAM driver write status get function

Table 9.16 Octa RAM Sub Function

Function Name	Description
octaram_write_mode_register	Set Write Mode function

7. Add the added Serial Flash Memory driver to the board settings.
   Refer to Chapter 9.3.3 for how to add.

9.3.2.1 octaram_open

octaram_open() is a function that initializes the Octa RAM so that it can be accessed.

octaram_open() implemented in default IPL initializes to access APS51208N mounted on the SMARC EVK board.

octaram_open		
Outline	Initialize Octa RAM	
Declaration	int octaram_open (xspidevice_ctrl_t * ctrl, xspidevice_cfg_t const * cfg)	
Description	- Set SPCLK frequency - Set the operating parameters to mode register of Octa RAM - Set opened flag	
Arguments	*ctrl	: Pointer to xSPI device control structure
	*cfg	: Pointer to xSPI device configure structure
Return value	0	: Initialization is complete
	-1	: Initialization is failed
Note	-	

Figure 9.45 shows a flowchart of the octaram_open() implementation in the default IPL.

The processing that changes the implementation when changing the Octa RAM is shown below.

- P1 Clock frequency setting change
- P2 Change operation setting parameters of Octa RAM

These are shown as P1, P2 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

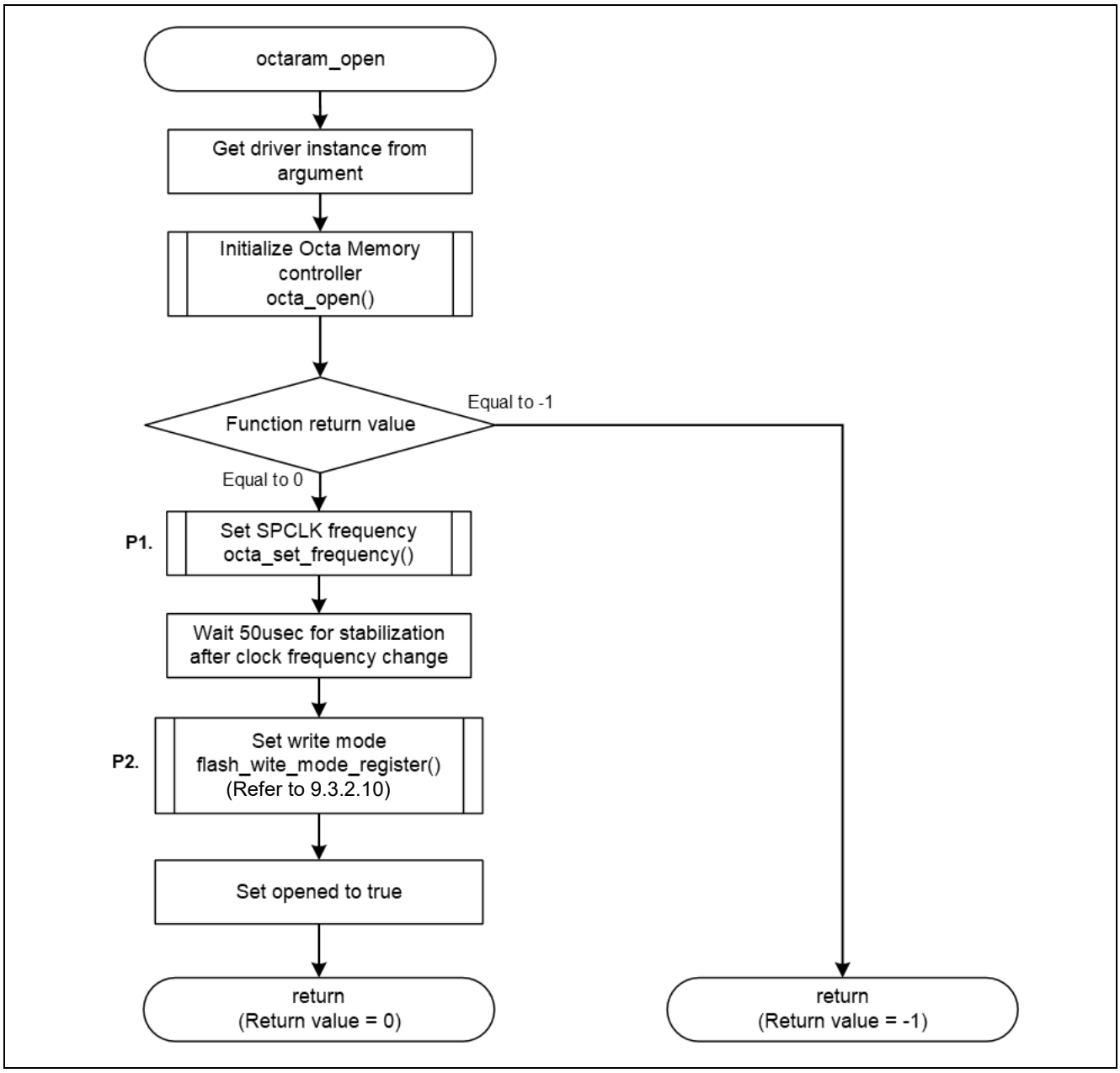


Figure 9.45 Flowchart of octaram_open

P1. Change the clock frequency setting

octaram_open sets the operating clock frequency of the Octa RAM.

The defined value of DEFAULT_SPI_FREQUENCY in L14 of octaram_custom.c shown in Figure 9.46 is used for the operating clock frequency.

Set this define value to the optimum value for your Octa RAM.

The maximum value that can be set is 100000000 (100MHz) and the minimum value is 3125000 (3.125MHz).

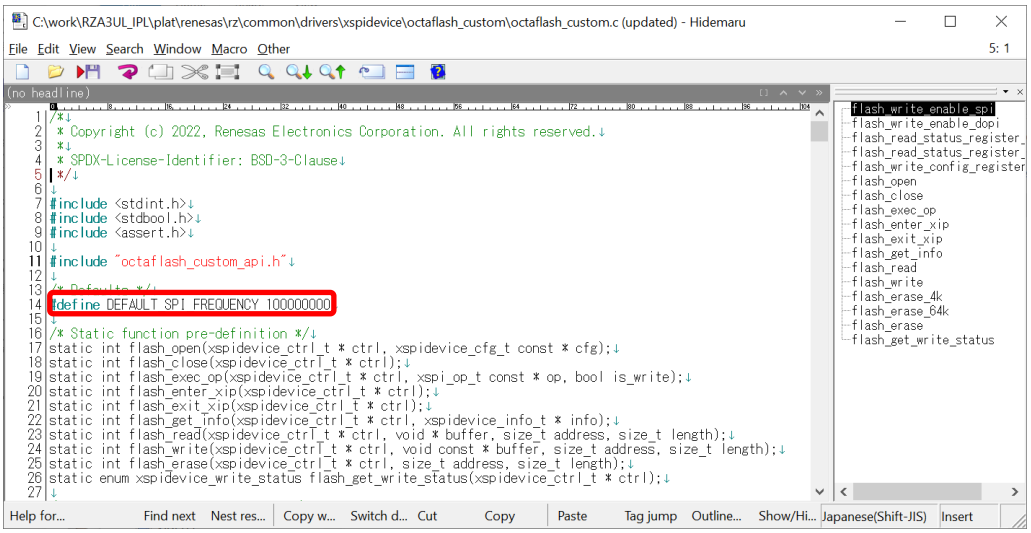


Figure 9.46 Change operation clock frequency

P2. Change operation setting parameters of Octa RAM

octaram_open() implemented by the default IPL performs mode setting, drive capacity setting, latency setting, and burst access setting as the initial setting of APS51208N.

These are set by writing values to the Mode Register of the APS51208N.

octaram_open() constructs the Mode Register setting value and passes it as an argument to the subfunction octaram_write_mode_register() function, which executes the register rewriting instruction.

Table 9.17 shows setting value to Mode Register.

When using the APS501208N at IPL, set the settings other than Latency Code to the values shown in Table 9.17.

If the same processing is required when changing the Octa RAM, please refer to this processing and make the optimal settings.

Table 9.17 Mode Register of APS51208N Setting

<table>
<tr><th>BIT</th><th>Description</th><th>Settings</th></tr>
<tr><td>15</td><td>Deep Power Down Enable</td><td>0: Deep Power Down Entry
1: Normal Operation</td></tr>
<tr><td>14 – 12</td><td>Drive Strength</td><td>000b: 100ohms
001b: 66ohms
010b: 50ohms
011b: 40ohms
100b: 33ohms
101b: 33ohms
110b: 25ohms
111b: 25ohms</td></tr>
<tr><td>11 - 8</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>7 - 4</td><td>Latency Code</td><td><table><tr><th></th><th>Max Freq</th></tr><tr><td>0000b</td><td>66MHz</td></tr><tr><td>0001b</td><td>104MHz</td></tr><tr><td>0010b</td><td>133MHz</td></tr><tr><td>0011b</td><td>166MHz</td></tr><tr><td>0100b</td><td>200MHz</td></tr><tr><td>0101b</td><td>200MHz</td></tr></table></td></tr>
<tr><td>3</td><td>Latency Type</td><td>0: Variable Latency
1: Fixed Latency</td></tr>
<tr><td>2</td><td>Burst Type</td><td>0: Wrapped
1: Hybrid Continuous</td></tr>
<tr><td>1 - 0</td><td>Burst Length</td><td>00b: 128byte
01b: 64byte
10b: 32byte
11b: 16byte</td></tr>
</table>

9.3.2.2 octaram_close

octaram_close() is a function to terminate the Octa RAM driver.

octaram_close() implemented in the default IPL only clears the opened flag, which indicates the open state of the driver.

Therefore, it is possible to access Octa RAM even after executing octaram_close().

octaram_close		
Outline	Close the Octa RAM	
Declaration	int octaram_close (xspidevice_ctrl_t * ctrl)	
Description	- Clear opened flag	
Arguments	*ctrl	: Pointer to xSPI device control structure
Return value	0	: Close process is complete
	-1	: Close process is failed
Note	-	

Figure 9.47 shows a flowchart of the octaram_close() implementation in the default IPL.

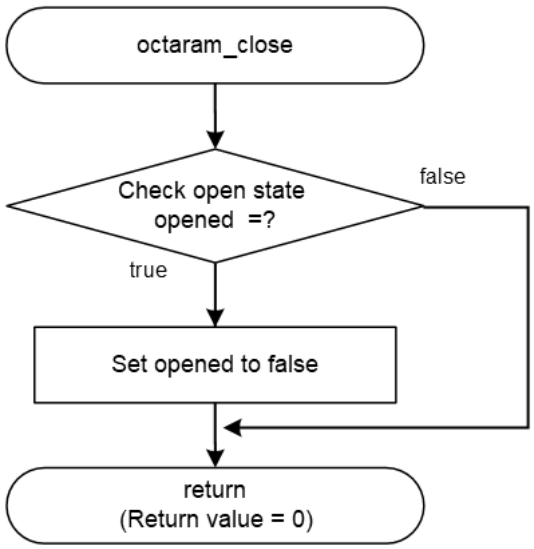


Figure 9.47 Flowchart of octaram_close

9.3.2.3 octaram_exec_op

octaram_exec_op() is a function to send operation information to be executed by Octa RAM.

This function is unused in the default IPL implementation of the Octa RAM driver.

octaram_exec_op	
Outline	Execute operation to Octa Flash memory
Declaration	void octaram_exec_op (xspidevice_ctrl_t * ctrl, xspi_op_t const * op, bool is_write) - Pass operation table *op to octa_exec_op() to transfer operation data
Arguments	*ctrl : Pointer to xSPI device control structure *op : Pointer to xSPI interface operation structure is_write : Specifies the transmission/reception direction of the operation(0=read、1=write)
Return value	0 : Execute the operation is complete -1 : Execute the operation is failed
Note	-

Figure 9.48 shows a flowchart of the octaram_exec_op() implementation in the default IPL.

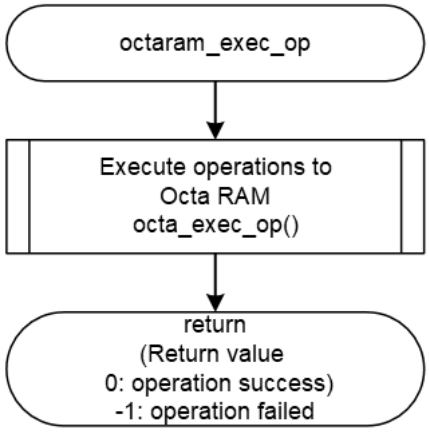


Figure 9.48 Flowchart of octaram_exec_op

9.3.2.4 octaram_enter_xip

octaram_enter_xip() is a function to set Memory-map Mode

By executing this function, data can be read and written by read / write access to the Octa RAM area (after H'28000000) in the memory space shown in Figure 5.4.

When Memory-map Mode is enabled, Octa Memory Controller automatically performs serial communication to read / write data when read / write access is made to Octa RAM area.

At this time, the Read operation and Write operation registered in Octa Memory Controller is used for serial communication.

octaram_enter_xip() implemented in the default IPL registers the Read operation and Write operation of APS51208N mounted on the SMARC EVK board to Octa Memory Controller.

octaram_enter_xip	
Outline	Set the Memory-map Mode to Octa RAM
Declaration	int octaram_enter_xip (xspidevice_ctrl_t * ctrl)
Description	- Register Read operation and Write operation to Octa Memory Controller - Change the Read access method to Memory-map Mode - Adjust memory access timing (memory calibration)
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : Setting is complete -1 : Setting is failed
Note	-

Figure 9.49 shows a flowchart of the octaram_enter_xip() implementation in the default IPL.

The processing that changes the implementation when changing the Octa RAM is shown below.

- P1 Register Read operation and Write operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

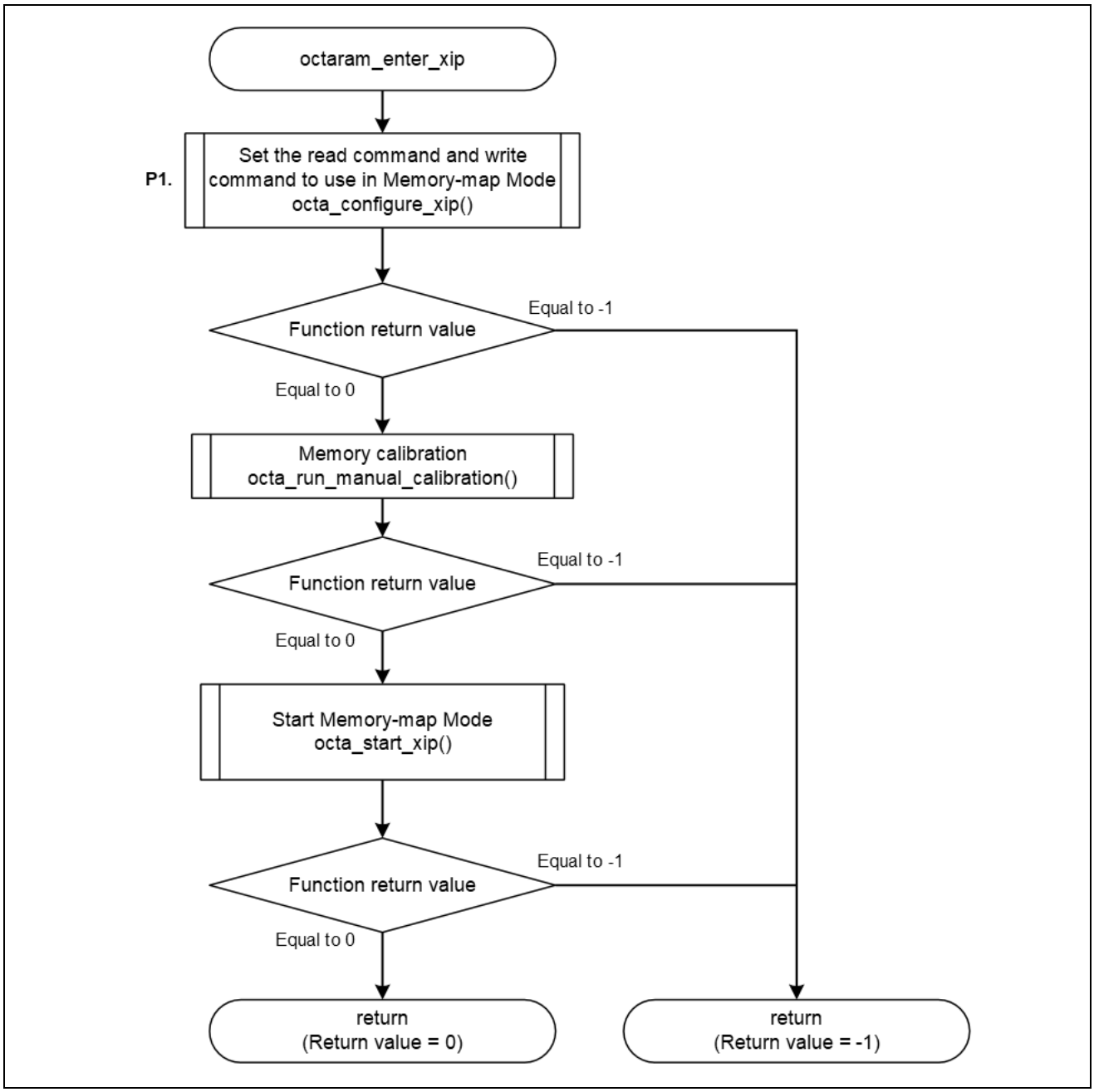


Figure 9.49 Flowchart of octaram_enter_xip

P1. register Read operation and Write operation

Register Read instructions to rop and Write instructions to wop to be used in Memory-map Mode.

rop and wop are passed as arguments to octa_config_xip

The Octa RAM driver implemented in the default IPL uses the Sync Read command (code: 0x8000) as the Read operation and the Sync Write command (code: 0x0000) as the Write operation.

Table 9.18 shows the rop settings when using the Sync Read command and Table 9.19 shows the wop setting when using the Sync Write command.

When changing the Octa RAM, rewrite rop and wop to the contents corresponding to the Read operation and Write operation of your Octa RAM.

Table 9.18 rop setting contents when using Sync Read (0x8000)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8ine
op	0x8000	Command code "Sync Read"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	Address is not set because bus access is translated
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Read address size = 4byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	4	Dummy cycles = 4 cycles
transfer_buffer	NULL	Buffer pointer is not set because bus access is translated
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Transfer data size is not set because bus access is translated
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 0 cycle
clsh_value	5	SSL Negation Delay = 5cycle
shsl_value	1	Next Access Delay = 1cycle

Table 9.19 wop setting contents when using Sync Write (0x8000)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8ine
op	0x0000	Command code "Sync Write"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	Address is not set because bus access is translated
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Write address size = 4byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	4	Dummy cycles = 4 cycles
transfer_buffer	NULL	Buffer pointer is not set because bus access is translated
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	0	Transfer data size is not set because bus access is translated
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	0	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.2.5 octaram_exit_xip

octaram_exit_xip() is a function to exit Memory-map Mode.

The Octa Memory Controller implemented in RZ/A3 always enables Memory-map Mode, so executing octaram_exit_xip() implemented in the default IPL does not actually end Memory-map Mode.

octaram_exit_xip	
Outline	Stop the Memory-map Mode to Octa RAM
Declaration	int octaram_exit_xip (xspi_ctrl_t * const ctrl)
Description	- Change the Read access method and Write access method to Manual Mode
Arguments	*ctrl : Pointer to the control structure for instance
Return value	0 : This value is always returned
Note	-

Figure 9.50 shows a flowchart of the octaram_exit_xip() implementation in the default IPL.

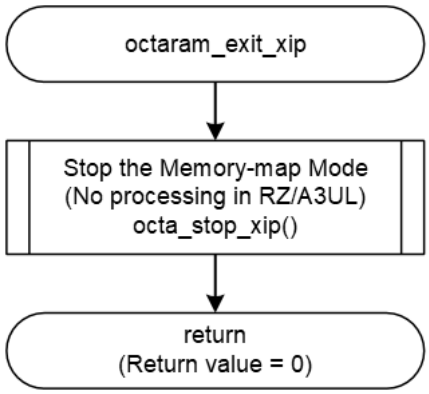


Figure 9.50 Flowchart of octaram_exit_xip

9.3.2.6 octaram_read

Since serial communication of operation is not required for Octa RAM read processing, only API functions are defined and processing is not implemented in functions.

Calling this function always returns an error.

To read data from the Octa RAM, access the Octa RAM area (after H'28000000) directly.

octaram_read		
Outline	The processing of this function is undefined. Calling this function will return an error and exit.	
Declaration	int octaram_read (xspi_ctrl_t * const ctrl, void * buffer, size_t address, size_t length)	
Description	- Return error	
Arguments	*ctrl	: Pointer to the control structure for instance
	* buffer	: Pointer to buffer to store read data
	address	: Data read source address
	length	: Read data size
Return value	-1	: This value is always returned
Note	-	

9.3.2.7 octaram_write

Since serial communication of operation is not required for Octa RAM write processing, only API functions are defined and processing is not implemented in functions.

Calling this function always returns an error.

To write data to the Octa RAM, access the Octa RAM area (after H'28000000) directly.

octaram_write		
Outline	The processing of this function is undefined. Calling this function will return an error and exit.	
Declaration	int octaram_write (xspi_ctrl_t * const ctrl, void const * buffer, size_t address, size_t length)	
Description	- Return error	
Arguments	*ctrl	: Pointer to the control structure for instance
	* buffer	: Pointer to buffer where write data is stored
	address	: Data write destination address
	length	: Write data size
Return value	-1	: This value is always returned
Note	-	

9.3.2.8 octaram_erase

Since serial communication of operation is not required for Octa RAM Erase processing, only API functions are defined and processing is not implemented in functions.

Calling this function always returns an error.

octaram_erase		
Outline	The processing of this function is undefined. Calling this function will return an error and exit.	
Declaration	int octaram_erase (xspidevice_ctrl_t * ctrl, size_t address, size_t length)	
Description	- Return error	
Arguments	*ctrl	: Pointer to the control structure for instance
	address	: Start address for data erase
	length	: erase size
Return value	-1	: This value is always returned
Note	-	

9.3.2.9 octaram_get_write_status

Since status acquisition processing is not required for Octa RAM, only API functions are defined and processing is not implemented in functions.

Calling this function always returns an error.

octaram_get_write_status		
Outline	The processing of this function is undefined. Calling this function will return an error and exit.	
Declaration	int octaram_get_write_status (xspidevice_ctrl_t * ctrl)	
Description	- Return error	
Arguments	*ctrl	: Pointer to the control structure for instance
Return value	-1	: This value is always returned
Note	-	

9.3.2.10 octaram_write_mode_register

octaram_write_mode_register() is a function that writes value to the APS51208N Mode Register.

octaram_write_mode_register() implemented in the default IPL write value to the Mode Register by sending the Write Mode Register operation of the APS51208N mounted on the SMARC EVK board.

octaram_write_mode_register	
Outline	Set Mode Register of APS51208N
Declaration	int octaram_write_mode_register(octaram_apsxx_ctrl_t * myctrl, uint16_t data)
Description	- Send Mode Register Write operation and write value to register
Arguments	*myctrl : Pointer to the control structure for instance
	data : Setting value to Mode Register
Return value	0 : Operation success
	-1 : Operation Failed
Note	-

Figure 9.51 shows a flowchart of the octaram_write_mode_register() implementation in the default IPL.

The processing that changes the implementation when changing the Octa RAM is shown below.

- P1 Send Mode Register Write operation

This is shown as P1 in the flow chart as implementation change points.

The details of the processing are explained after the flow chart.

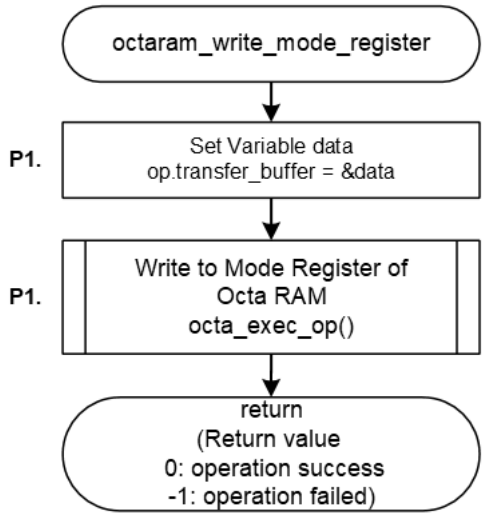


Figure 9.51 Flowchart of octaram_write_mode_register

P1. Send Mode Register Write operation

Register the Mode Register Write operation (Code: 0x4000) in the argument op_write_mode to octa_exec_op().

Also, register the pointer of the argument data that indicates the set value to the Mode Register in the transfer_buffer member.

octaram_write_mode_register() implemented in the default IPL sets the Mode Register Write command (code: 0x4000)setting of op_write_mode.

Table 9.20 shows the settings of op_write_mode.

When changing the Octa RAM, rewrite the variable members according to the specifications of the operation equivalent to the APS51208N Mode Register Write operation implemented in your Octa RAM.

Table 9.20 op_write_mode setting contents when using Mode Register Write (0x4000)

Member	Setting value	Description
form	SPI_FORM_8_8_8	Number of I/O line used for command: 8line Number of I/O line used for address: 8line Number of I/O line used for data: 8ine
op	0x4000	Command code "Mode Register Write"
op_size	2	Command size = 2byte
op_is_ddr	true	Command transfer method (DDR)
address	0	Read Address is set in function
address_is_ddr	true	Address transfer method (DDR)
address_size	4	Read address size = 4byte
additional_value	0	No option data
additional_size	0	No option data
dummy_cycles	0	Dummy cycles = 0 cycles
transfer_buffer	&data	Set pointer of argument data in function
transfer_is_ddr	true	Data transfer method (DDR)
transfer_size	2	Data size = 2byte
force_idle_level_mask	0	IO pins to Hi-Z during idle state
force_idle_level_value	0	IO pins to Hi-Z during idle state
slch_value	0	Clock Delay = 1.5cycle
clsh_value	0	SSL Negation Delay = 6.5cycle
shsl_value	1	Next Access Delay = 5cycle

9.3.3 How to add a board setting

1. Open the c:\work\RZA3UL_IPL\plat\renesas\rz\board folder, copy the "a3ul_smarc_octal" folder and rename it to any name.
   This time, rename the folder name to "a3ul_custom_board_octal".

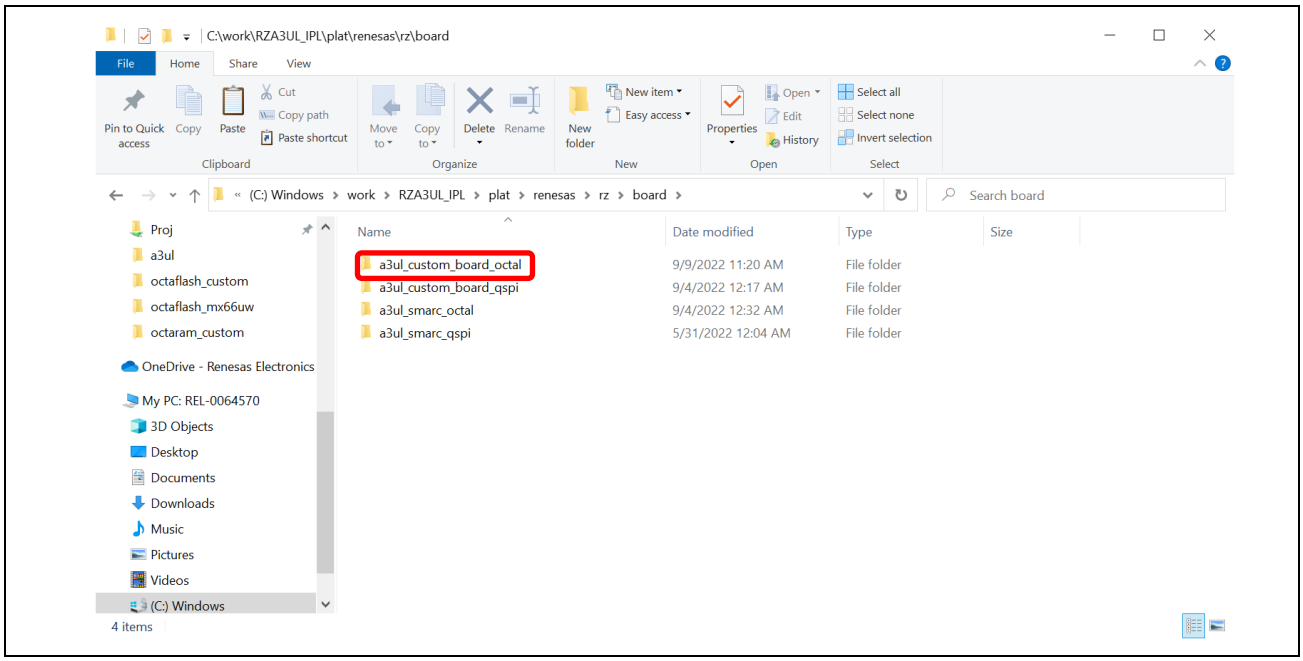
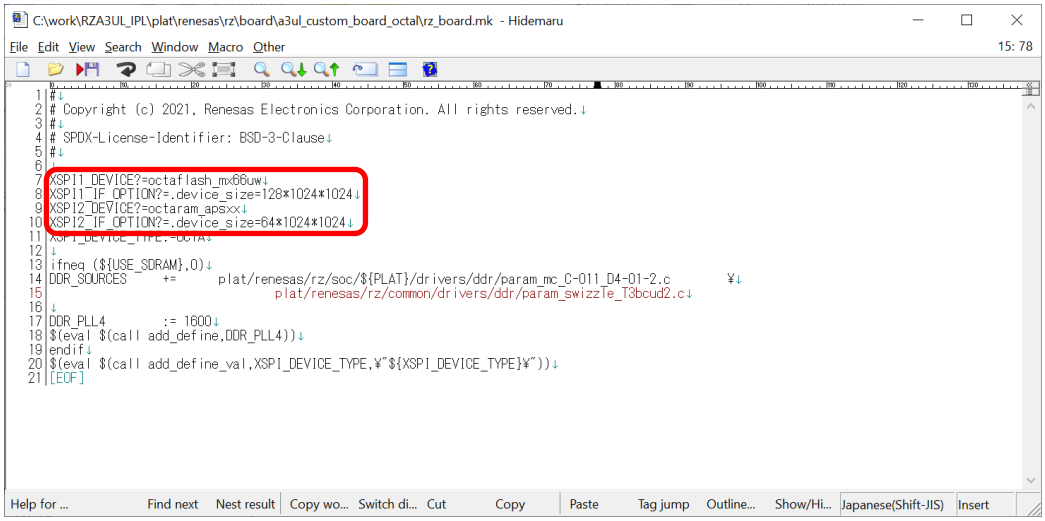


Figure 9.52 Rename board setting folder

2. Open "a3ul_custom_board_octal" folder and open "rz_board.mk".
   Register the folder name of the Octa Flash memory driver and Octa RAM driver added in Chapter 9.3.1 and Chapter 9.3.2.
   Rewrite "_XSPI1_DEVICE:=octaflash_mx66uw" on line 7 to "_XSPI1_DEVICE:=octaflash_custom" and rewrite "_XSPI2_DEVICE:=octaram_apsxx" on line 9 to "_XSPI2_DEVICE:=octaram_custom".
   Then, rewrite the device size value on lines 8 and 10 with the memory size of your Octa Flash memory and your Octa RAM.



Note: For RZ/A3UL, register the Octa Flash memory driver in XSPI1_DEVICE and the Octa RAM driver in XSPI2_DEVICE.
Channels cannot be swapped.
Also, by removing the definition of L7 and L8, the Octa Flash memory driver will not be included in the IPL.
Similarly, removing the L9 and L10 definitions will prevent the Octa RAM driver from being included in the IPL.

Figure 9.53 Change board setting Makefile

3. Launch e² studio.
   Then refer to step 10 of chapter 7. IPL build environment construction procedure to launch Properties for RZA3UL_IPL and move to the [Behavior] tab.
   Change the BOARD value set in the "Build arguments" field to "a3ul_custom_board_octal".
   After this, running the build as usual will generate an IPL for your custom board.

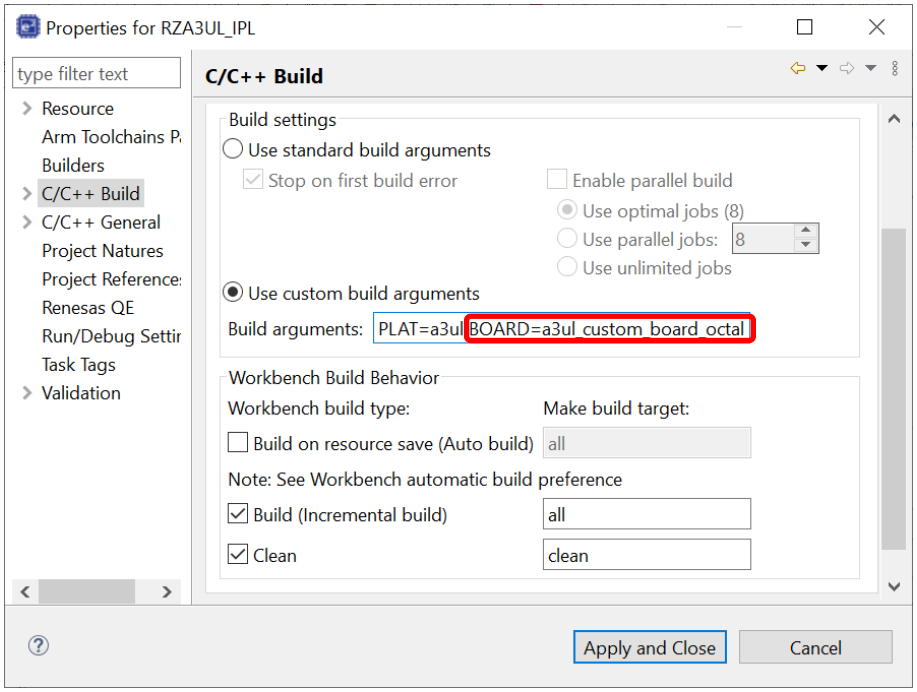
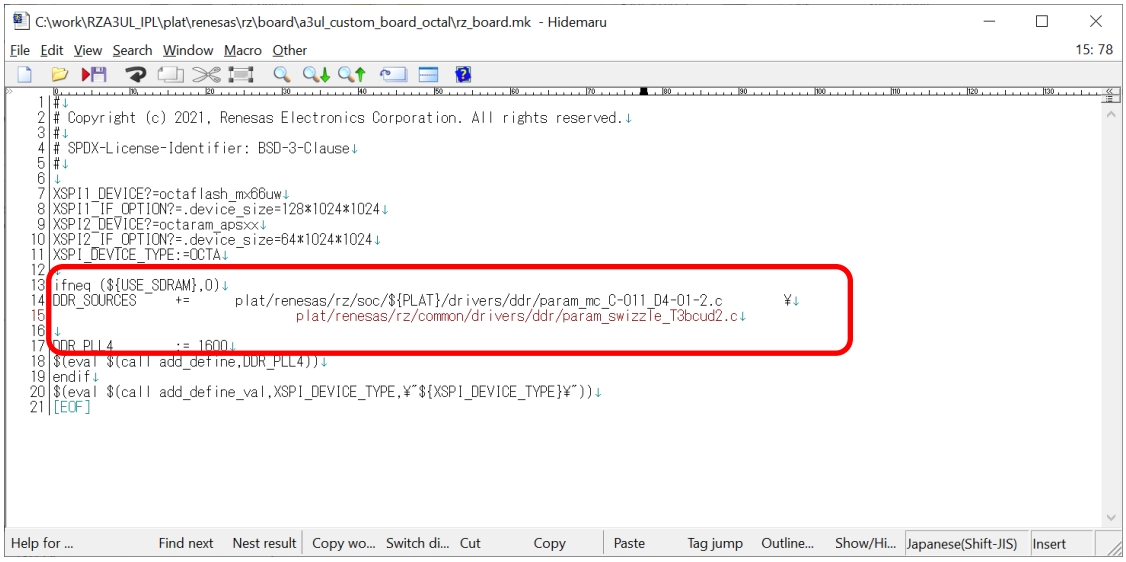


Figure 9.54 Change project setting



Note: When changing DDR SDRAM, set the parameter file corresponding to DDR SDRAM to DDR_SOURCES in rz_board.mk, and set the operating clock frequency of DDR SDRAM to DDR_PLL4.
Please contact Renesas Electronics for the parameter file corresponding to your DDR SDRAM.

Figure 9.55 Information of DDR SDRAM setting

9.3.4 How to add a Module for Clock Supply

In IPL, clocks are supplied to the modules shown in the CPG column of Table 5.1, resets are released, and the modules are made available.

This chapter describes how to add the settings when adding a module that supplies clocks and resets.

1. Open "c:\work\RZA3UL_IPL\plat\renesas\rz\common\drivers\cpg.c".
2. Add the setting table of the module to be supplied with the clock to "cpg_clk_on_tbl[]".

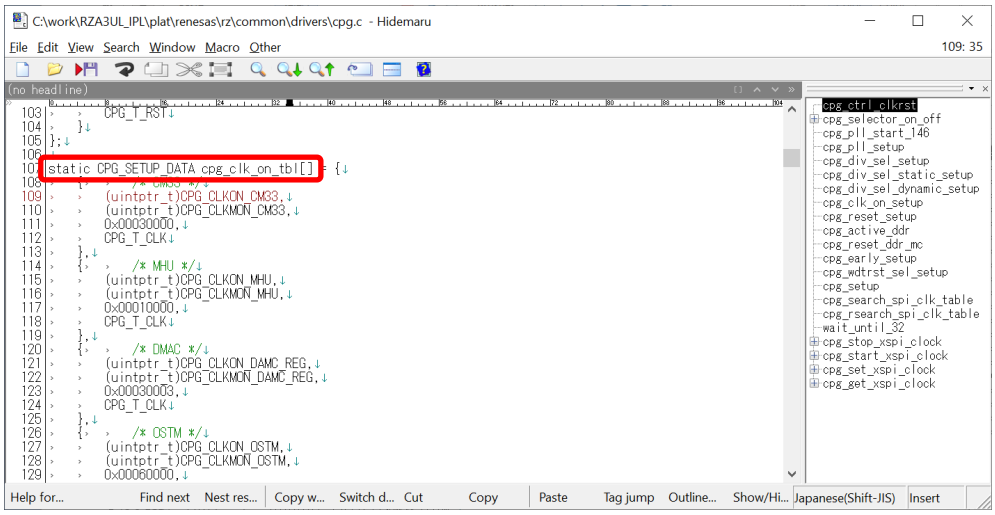


Figure 9.56 Information of cpg_clk_om_tbl[]

3. The setting table has the following format.

```
{
    reg,
    mon,
    val,
    type
}
```

reg: Sets the Clock Control Register address of the module supplying the clock.
mon: Sets the Clock Monitor Register address corresponding to the register set in reg.
val: Sets the value to write to the register set in reg.
type: Set CPG_T_CLK.
For details of Clock Control Register and Clock Monitor Register, refer to RZ/A3UL Group User's Manual: Hardware (R01UH0973EJ).

4. Add the setting table of the module to be reset to "cpg_reset_tbl[]".

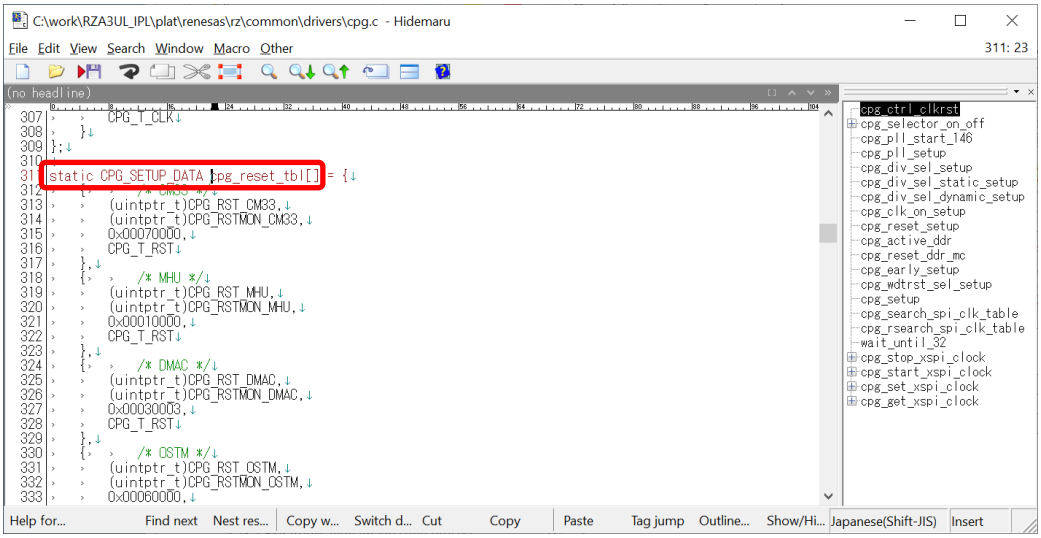


Figure 9.57 Information of cpg_reset_tbl[]

5. The format of the setting table is the same as "cpg_clk_on_tbl[]".
The values to be set are as follows.
reg: Sets the Reset Control Register address of the module to be reset.
mon: Sets the Reset Monitor Register address corresponding to the register set in reg.
val: Sets the value to write to the register set in reg.
type: Set CPG_T_RST.
For details of Reset Control Register and Reset Monitor Register, refer to RZ/A3UL Group User's Manual: Hardware (R01UH0973EJ).

9.3.5 How to add pin setting

In IPL, the pin setting shown in the GPIO column of Table 5.1 is performed.

This chapter shows how to add pin settings.

1. Open "c:\work\RZA3UL_IPL\plat\renesas\rz\common\drivers\pfc.c".
2. Add the pin setting table to "pfc_mux_reg_tbl[]".

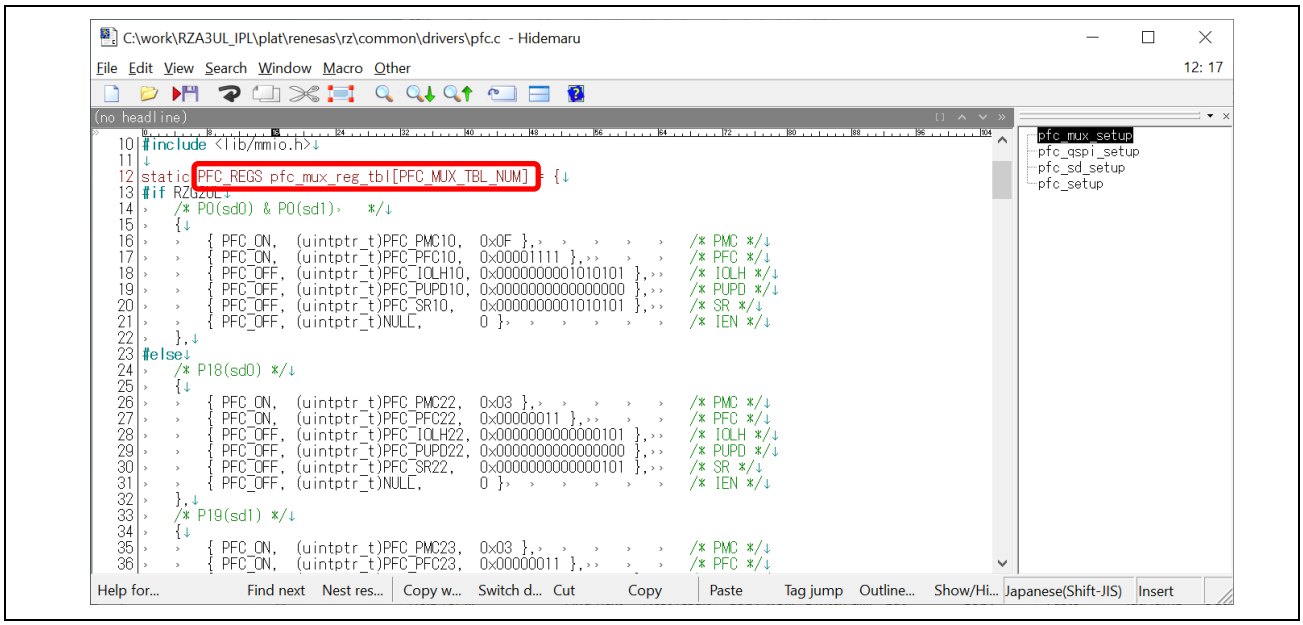


Figure 9.58 Information of pfc_mux_reg_tbl[]

3. The setting table has the following format.

```
{
    {pmc.flag,   pmc.reg,   pmc.val },
    {pfc.flag,   pfc.reg,   pfc.val },
    {iolh.flag,  iolh.reg,  iolh.val },
    {pupd.flag,  pupd.reg,  pupd.val },
    {sr.flag,    sr.reg,    sr.val },
    {ien.flag,   ien.reg,   ien.val }
}
```

"XXX" below indicates PMCn, PFCn, IOLHn, PUPDn, SRn, IENn registers.
XXX.flag: Set PFC_ON to change the register value registered in the table. Set PFC_OFF if you do not want to change the register value.
XXX.reg: Set the address of the register to which the parameter is to be set.
XXX.val: Sets the value to write to the register set in reg.
XXX indicates PMCn, PFCn, IOLHn, PUPDn, SRn, IENn registers.
For details of each register, refer to RZ/A3UL Group User's Manual: Hardware (R01UH0973EJ).

4. Open pfc_regs.h and increase the value of PFC_MUX_TBL_NUM by the number of added tables.

9.3.6 How to change MMU setting

This chapter shows how to change MMU settings.

1. Open "c:\work\RZA3UL_IPL\plat\renesas\rz\common\bl2_plat_setup.c".
2. Rewrite the MMU setting table of "g_mmu_pagetable_array[]".

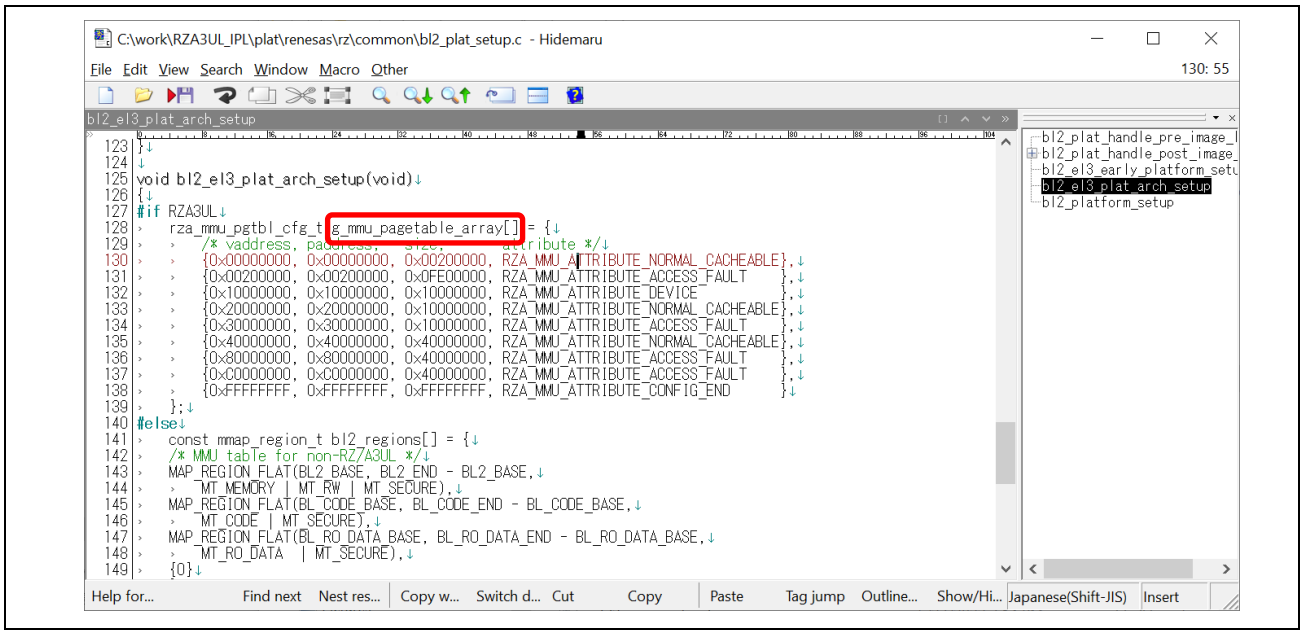


Figure 9.59 Information of g_mmu_pagetable_array[]

3. Each line in "g_mmu_pagetable_array[]" is the setting for each block in the virtual address space, and consists of the following format.

```
{
    {vaddress,  paddress,  size, attribute },
    ...
    {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, RZA_MMU_ATTRIBUTE_CONFIG_END }
}
```

vaddress: Sets the starting address of the virtual address space. Sets a 2Mbyte aligned address.
paddress: Sets the starting address of the physical address space. Sets a 2Mbyte aligned address.
size: Sets the size of block. Sets a 2Mbyte aligned size.
Attribute: Memory attribute. The following parameters can be specified.

- RZA_MMU_ATTRIBUTE_NORMAL_CACHEABLE: Normal Memory, Cache enable (Write-back)
- RZA_MMU_ATTRIBUTE_NORMAL_UNCACHE: Normal Memory, Cache disable
- RZA_MMU_ATTRIBUTE_DEVICE: Device Memory
- RZA_MMU_ATTRIBUTE_ACCESS_FAULT: Areas where access is prohibited
  This attribute is set for areas that are not set in the MMU setting table.
- RZA_MMU_ATTRIBUTE_CONFIG_END: Indicates the end of MMU configuration
  This attribute is set to the last element of the MMU setting table.

10. Precautions

10.1 About DDR SDRAM access prohibited area

Test data for DDR SDRAM initialization is stored in the DDR SDRAM area shown below.

If you overwrite the test data, you may not be able to access DDR SDRAM correctly.

Therefore, access to this area is prohibited.

- Access prohibited address:
- 0x40000000 - 0x4000003F

10.2 About Octa RAM access prohibited area

Test data for Octa RAM initialization is stored in the Octa RAM area shown below.

If you overwrite the test data, you may not be able to access Octa RAM correctly.

Therefore, access to this area is prohibited.

- Access prohibited address:
- 0x28000000 - 0x2800000F

11. Reference Documents

User's Manual: Hardware

RZ/A3UL Group User's Manual: Hardware (R01UH0973EJ)
(Please get the latest version from the Renesas Electronics website)

RZ/G2UL, RZ/A3UL, RZ/FIVE SMARC Module Board User's Manual: Hardware (R01UH0990EJ)
(Please get the latest version from the Renesas Electronics website)

RZ SMARC Series Carrier Board User's Manual: Hardware (R01UH0990EJ)
(Please get the latest version from the Renesas Electronics website)

Getting Started with RZ/A Flexible Software Package (R01AN6499EJ)
(Please get the latest version from the Renesas Electronics website)

Arm® Architecture Reference Manual for A-profile architecture
(Please get the latest version from the Arm website)

Arm® Cortex®-A Series Programmer's Guide for ARMv8-A Version:1.0
(Please get the latest version from the Arm website)

Arm® Cortex®-A55 Core Technical Reference Manual Revision; r2p0
(Please get the latest version from the Arm website)

Technical Update/Technical News
(Please get the latest information from the Renesas Electronics website)

User's Manual: Integrated Development Environment
(Please get the latest version from the Renesas Electronics website)

12. Trademarks

Company names and product names in this document are trade names, trademarks or registered trademarks of their respective companies.

- Arm, Cortex and CoreSight are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Revision History

Rev.	Date	Description	
		Page	Summary
1.0.0	Sep.30.22	-	First edition issued.
1.1.0	Dec.26.22	-	Changed from the document for RZ/A3UL IPL to the document for RZ/A3 IPL.
		1	Introduction Added a list of document summaries.
		5	Table 1.1 Added description of Octa Flash memory Driver / Octa RAM driver.
		6	Table 1.2 Added description of Configuration Mode and Memory-map Mode.
		7	Table 2.1 Added QSPI0_SOCLK information. Updated information for IPL v1.2.0.
		22	Figure 5.4 Added address information of Octa Flash memory area and Octa RAM area.
		52	Table 5.56 Add a new operation table structure information.
		56	Table 5.75 Added sub function of Octa Flash memory driver and Octa RAM driver.
		73	5.12.2.3 Added sub function of Octa Flash memory driver and Octa RAM driver.
		86	Figure 5.7 Fixed chapter number on the flowchart.
		95	6.1.1.1 Update installation procedure of e² studio 2022-10.
		107	6.2.1.1 Update installation procedure of e² studio 2022-10.
		130	Figure 8.5 Added setting of "Reset Before Run". Added information about J-Link interface type.
		131	Figure 8.6 Added informatin about Flash memory. Change Reset setting after reloading to No.
		147	Figure 9.2 Added setting of "Reset Before Run".
		157	9.3 Added the outline of the contents explained in Chapter 9.3.
		157	9.3.1 Added a description of the serial transfer method and transfer.format to the Octa Flash memory.
		166	9.3.1.1 to 9.3.1.13 Added the new chapters describing the Octa Flash memory driver functions.

Rev.	Date	Description	
		Page	Summary
1.1.0	Dec.26.22	199	9.3.2 Added a description of the serial transfer method to Octa RAM.
		206	9.3.2.1 to 9.3.1.13 Added the new chapters describing the Octa RAM driver functions.
		229	10.1 Removed chapter 10.1 from the previous version.
2.0.0	Feb.28.25	9	Table 2.1 Update information of Operation Confirmation Conditions.
		96	6.1.1.1 Update installation procedure of e² studio for Windows.
		109	6.2.1.1 Update installation procedure of e² studio for Linux.
		131	8.1 Updated the procedure for creating Debug Configuration using new e² studio GUI.
		142	8.3 Updated setup procedure to use the new e² studio GUI.
		146	8.4 Updated execution procedure to use the new e² studio GUI.
		150	9.1 Updated execution procedure to use the new e² studio GUI.
		154	9.2 Updated writing procedure to use the new e² studio GUI.

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/.

© 2025 Renesas Electronics Corporation. All rights reserved.