Renesas RA Family

RA6 Secure Firmware Update using MCUboot and Flash Dual Bank

Introduction

MCUboot is a secure bootloader for 32-bit MCUs. It defines a common infrastructure for the bootloader, defines system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software update. MCUboot is operating system and hardware independent and relies on hardware porting layers from the operating system it works with. The Renesas Flexible Software Package (FSP) integrates an MCUboot port starting from FSP v3.0.0. Users can benefit from using the FSP MCUboot Module to create a Root of Trust (RoT) for the system and perform secure booting and fail-safe application updates.

MCUboot is maintained by Linaro in the GitHub mcu-tools page https://github.com/mcu-tools/mcuboot. There is a `\docs` folder that holds the documentation for MCUboot in .md file format. This application note refers to the above-mentioned documents wherever possible and is intended to provide additional information that is related to using the Renesas FSP MCUboot Module.

For RA Family RA6M4, RA6M5, and RA6E1 MCU Groups, the internal code flash has a dual bank feature, which can be used to simplify and accelerate firmware update. This dual bank feature is supported from FSP v3.6.0. This application note demonstrates secure bootloader design using this dual bank feature.

Example projects using the EK-RA6M4 evaluation kit are provided in this application project. In addition, steps for how to master an application to use with the bootloader and how to update to a new application are provided. Users can follow these steps to recreate the reference bootloader and link the example application projects included in this application project to use the bootloader.

If you are interested in secure bootloader design using the MCUboot module with RA6 internal flash in linear mode, reference application project R11AN0497.

Required Resources

Development tools and software

- The e^2 studio ISDE v2022-10 or greater
- Renesas Flexible Software Package (FSP) v4.0.0 or later
- SEGGER J-link® USB driver

The above three software components: the FSP, J-Link USB drivers and e^2 studio are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

- Python v3.9 or later- https://www.python.org/downloads/
- Renesas Flash Programming (RFP) v3.10 or later
  https://www.renesas.com/us/en/software-tool/renesas-flash-programmer-programming-gui

Hardware

- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group http://www.renesas.com/ra/ek-ra6m4
- Workstation running Windows® 10
- Two USB device cables (type-A male to micro-B male)
- One USB to TTL Serial 3.3-V UART Converter

Prerequisites and Intended Audience

Users of this application note and project should have some experience with the Renesas e^2 studio. Users should read the MCUboot Port section of the FSP User's Manual as well as the MCU Hardware User's manual Flash Memory section prior to working with this application project. Users should also have some knowledge of cryptography. Prior knowledge of Python usage is also helpful.

The intended audience includes product developers, product manufacturers, product support, or end users who are involved with designing application systems involving usage of a secure bootloader.

Using this Application Note

Section 1 is an overview of the code flash dual bank feature of RA6M4 and RA6M5 MCUs. Users who are familiar with the MCU dual bank features can skip this section.

Section 2 covers the general flow of architecting a system using the FSP MCUboot module. For example, memory configuration for a code flash dual bank-based bootloader using MCUboot is introduced in this section.

Section 3 covers the introduction to the example projects included in this application project. User should review this section to understand how to use the example projects.

Section 4 covers the steps to create a secure bootloader using the code flash dual bank feature and MCUboot module. Users who will customize the bootloader should review this section to understand how the bootloader is structured.

Section 5 provides the steps to configure and sign an application to use the bootloader created in section 4. The included example projects are used in this section.

Section 6 provides the instructions on how to debug and boot the primary application project and update to a new image. Users who will use the dual bank feature for the first time should review this section as it includes information about:

- Debugging and booting the primary application
- Downloading a new image using the primary image downloader
- Booting the new image

Section 7 covers the production support of provisioning the new MCU with the bootloaders and the initial application.

Section 8 provides instructions on how to run the included example projects. Users who are familiar with bootloader design using MCUboot can go to this section for a quick evaluation of the included example projects.

Contents

1. Code Flash Dual Bank Feature

For RA6M4 and RA6M5 MCU groups, the internal flash memory can operate in linear mode or dual bank mode. In linear mode, the code flash memory is used as one area. In dual bank mode, the code flash memory is divided into two areas. In code flash dual bank mode, the bank swap function can be used to boot into a new application for a system that includes a bootloader.

1.1 RA6M4 and RA6E1 MCU Group Code Flash Configuration

Using the 1-Mbyte product as an example, the code flash memory in linear mode for RA6M4 includes the blocks shown in Figure 1.

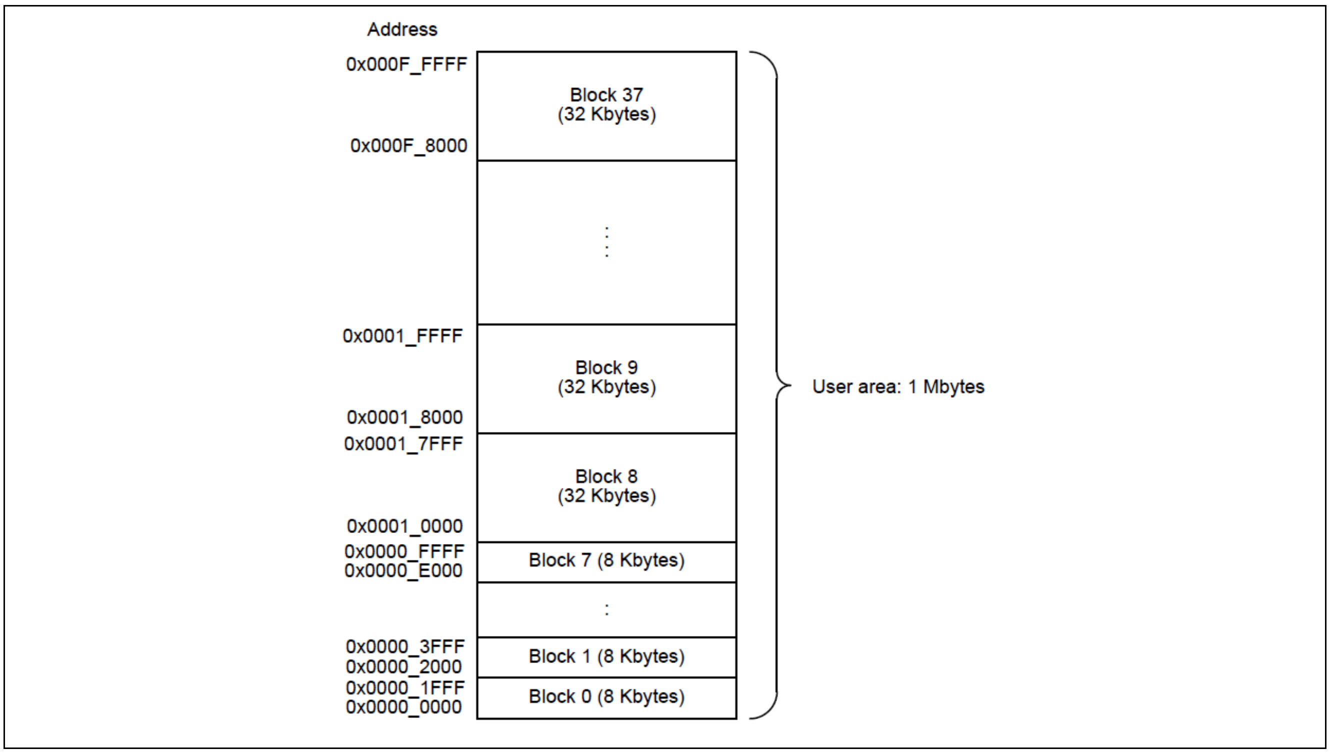


Figure 1. RA6M4 and RA6E1 Code Flash Memory in Linear Mode

Upper Bank Address in Code Flash Linear Mode

In code linear mode, the upper bank starting address is half of the code flash size. For example, for the 1-Mbyte RA6M4 and RA6E1 MCU used in this example project, the starting address of the upper bank address is 0x80000. The upper bank linear mode address is used when downloading the upper bank bootloader using MCUboot in code flash dual bank mode.

Using the 1-Mbyte product as an example, the code flash memory in dual bank mode includes the blocks shown in Figure 2.

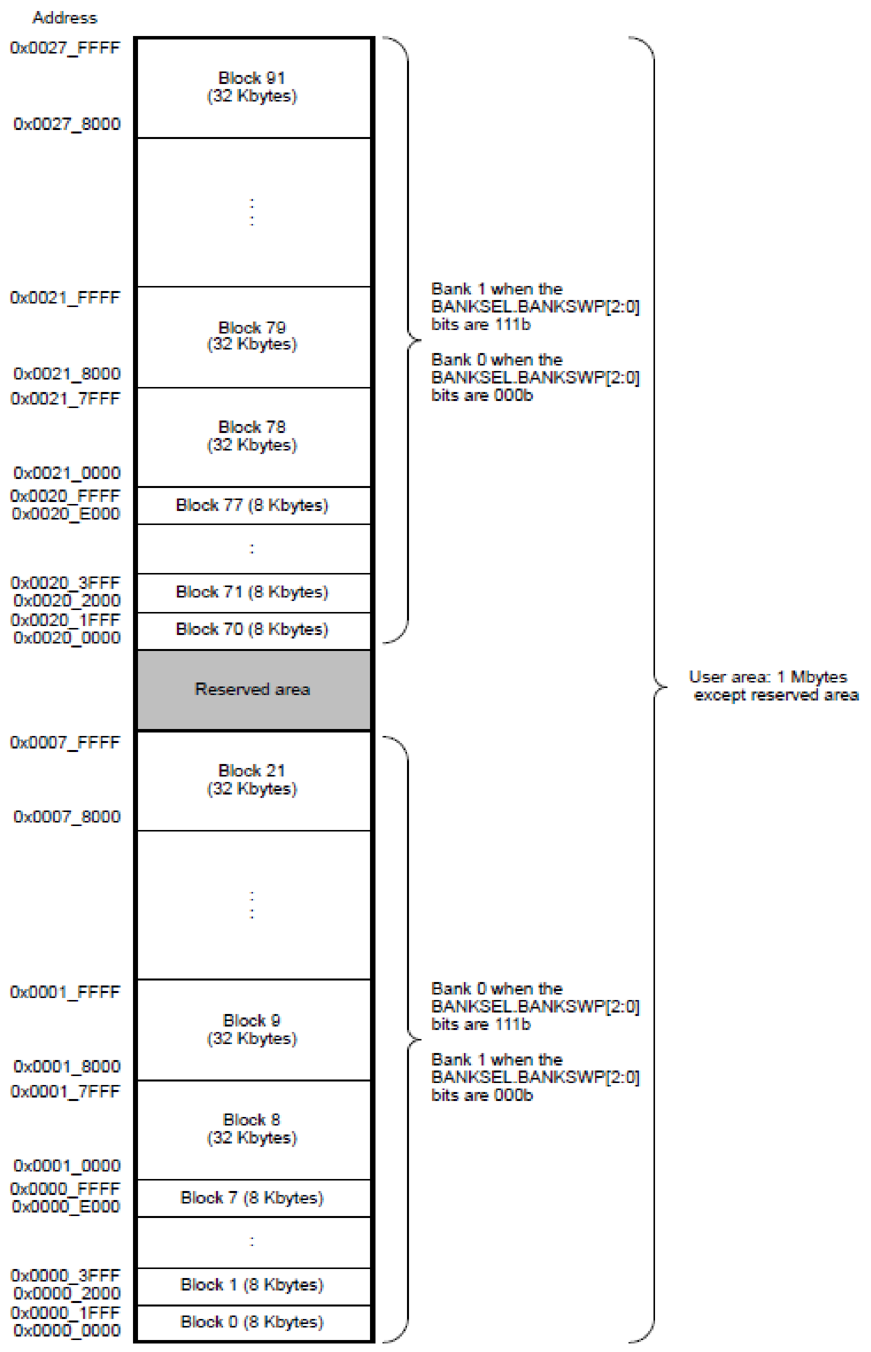


Figure 2. RA6M4 and RA6E1 Code Flash Memory in Dual Bank Mode

Table 1 is a summary of the code flash blocks in linear and dual bank mode. The upper bank address in dual bank mode is 0x200000 regardless of the code flash size. This address should be used with the application image downloader.

Table 1. RA6M4 and RA6E1 Code Flash

	Code Flash Range Address	
Product	**Linear**	**Dual**
1-Mbyte product	0x0000_0000 to 0x000F_FFFF	Lower side bank: 0x0000_0000 to 0x0007_FFFF
		Upper side bank: 0x0020_0000 to 0x0027_FFFF
768-Kbytes product	0x0000_0000 to 0x000B_FFFF	Lower side bank: 0x0000_0000 to 0x0005_FFFF
		Upper side bank: 0x0020_0000 to 0x0025 FFFF
512 Kbytes product	0x0000_0000 to 0x0007_FFFF	Lower side bank: 0x0000_0000 to 0x0003_FFFF
		Upper side bank: 0x0020_0000 to 0x0023_FFFF

Figure 3 is the code flash block structure for the RA6M4 and RA6E1. The code flash erase and programming minimum unit is the code flash block size. The block numbering scheme is used in the block protection design.

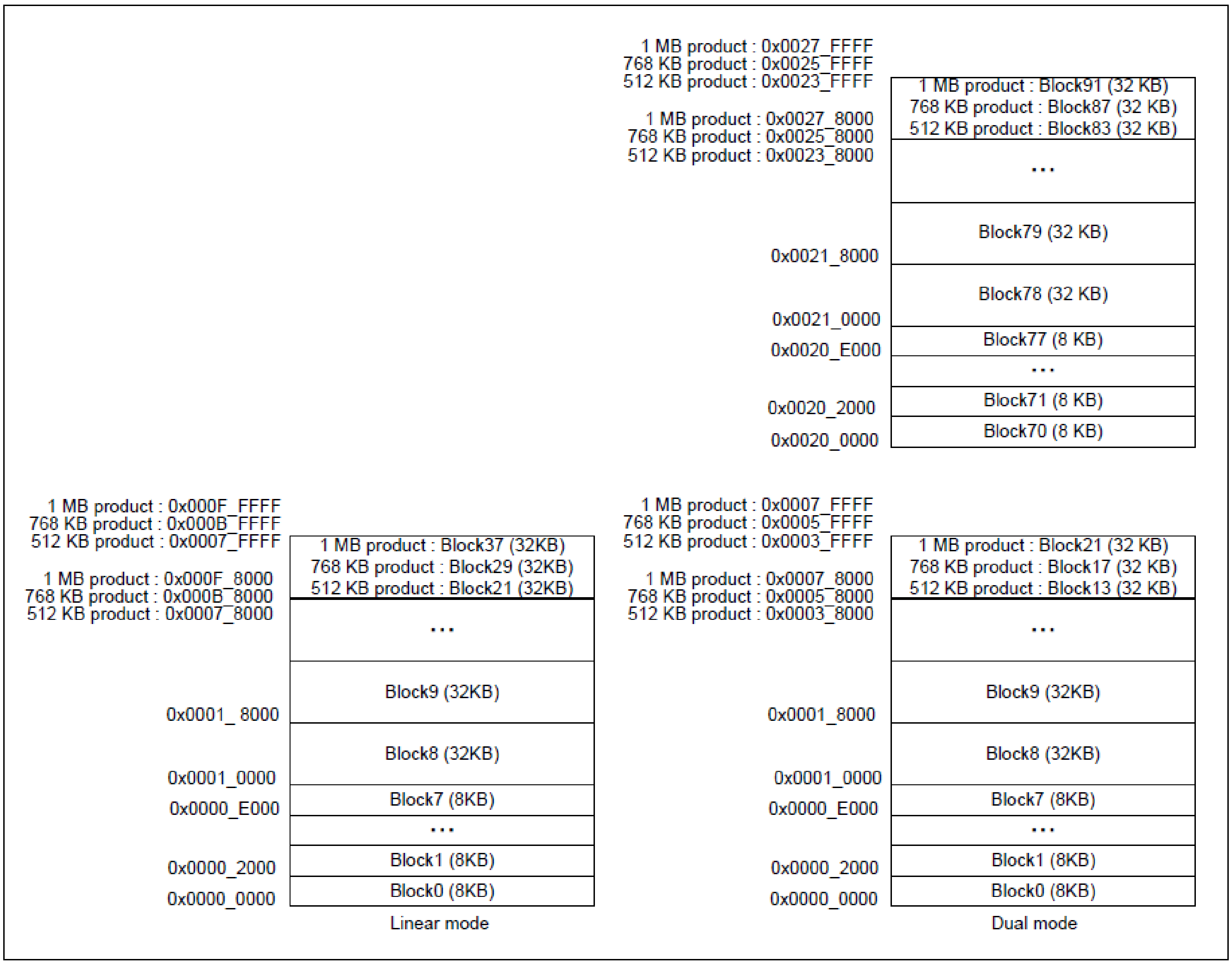

1 MB product : 0x0027_FFFF
768 KB product : 0x0025_FFFF
512 KB product : 0x0023_FFFF
1 MB product : 0x0027_8000
768 KB product : 0x0025_8000
512 KB product : 0x0023_8000
1 MB product : Block91 (32 KB)
768 KB product : Block87 (32 KB)
512 KB product : Block83 (32 KB)
...
Block79 (32 KB)
0x0021_8000
Block78 (32 KB)
0x0021_0000
Block77 (8 KB)
0x0020_E000
...
0x0020_2000
Block71 (8 KB)
0x0020_0000
Block70 (8 KB)
1 MB product : 0x000F_FFFF
768 KB product : 0x000B_FFFF
512 KB product : 0x0007_FFFF
1 MB product : 0x000F_8000
768 KB product : 0x000B_8000
512 KB product : 0x0007_8000
1 MB product : Block37 (32KB)
768 KB product : Block29 (32KB)
512 KB product : Block21 (32KB)
...
Block9 (32KB)
0x0001_ 8000
Block8 (32KB)
0x0001_0000
Block7 (8KB)
0x0000_E000
...
Block1 (8KB)
0x0000_2000
Block0 (8KB)
0x0000_0000
Linear mode
1 MB product : 0x0007_FFFF
768 KB product : 0x0005_FFFF
512 KB product : 0x0003_FFFF
1 MB product : 0x0007_8000
768 KB product : 0x0005_8000
512 KB product : 0x0003_8000
1 MB product : Block21 (32 KB)
768 KB product : Block17 (32 KB)
512 KB product : Block13 (32 KB)
...
Block9 (32KB)
0x0001_8000
Block8 (32KB)
0x0001_0000
Block7 (8KB)
0x0000_E000
...
Block1 (8KB)
0x0000_2000
Block0 (8KB)
0x0000_0000
Dual mode

Figure 3. RA6M4 and RA6E1 Code Flash Block Structure

1.2 RA6M5 MCU Group Code Flash Configuration

Using the 2-Mbyte product as example, the code flash memory in linear mode for the RA6M5 includes the blocks shown in Figure 4.

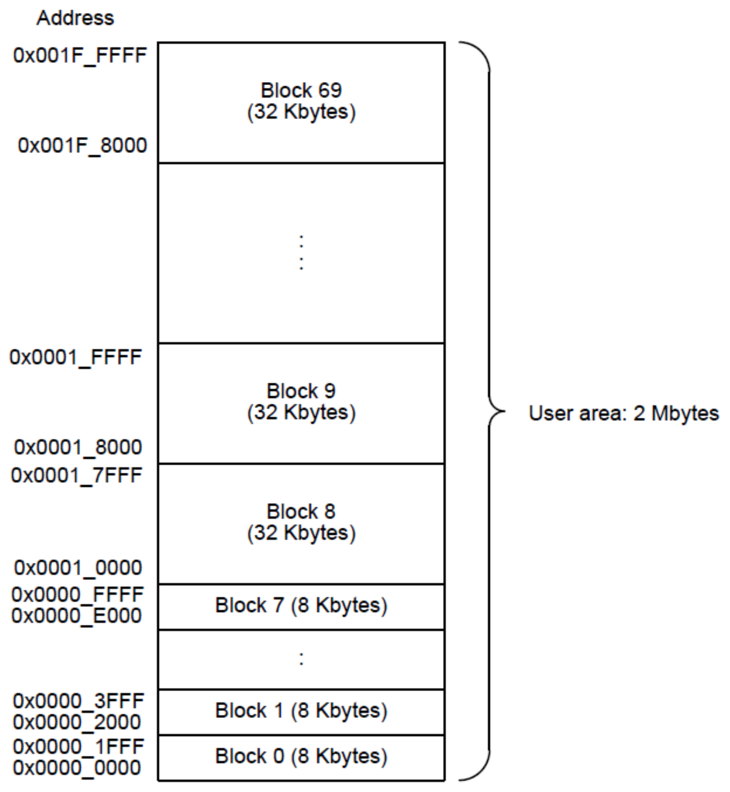


Figure 4. RA6M5 Code Flash Memory in Linear Mode

Upper Bank Address in Code Flash Linear Mode

In code linear mode, the upper bank starting address is half of the code flash size. For example, for the 2-Mbyte RA6M5 MCUs, the starting address of the upper bank address is 0x100000. The upper bank linear mode address is used when downloading the upper bank bootloader when using MCUboot in code flash dual bank mode.

Using the 2-Mbyte product as example, the code flash memory for the RA6M5 in dual bank mode includes the blocks shown in Figure 5.

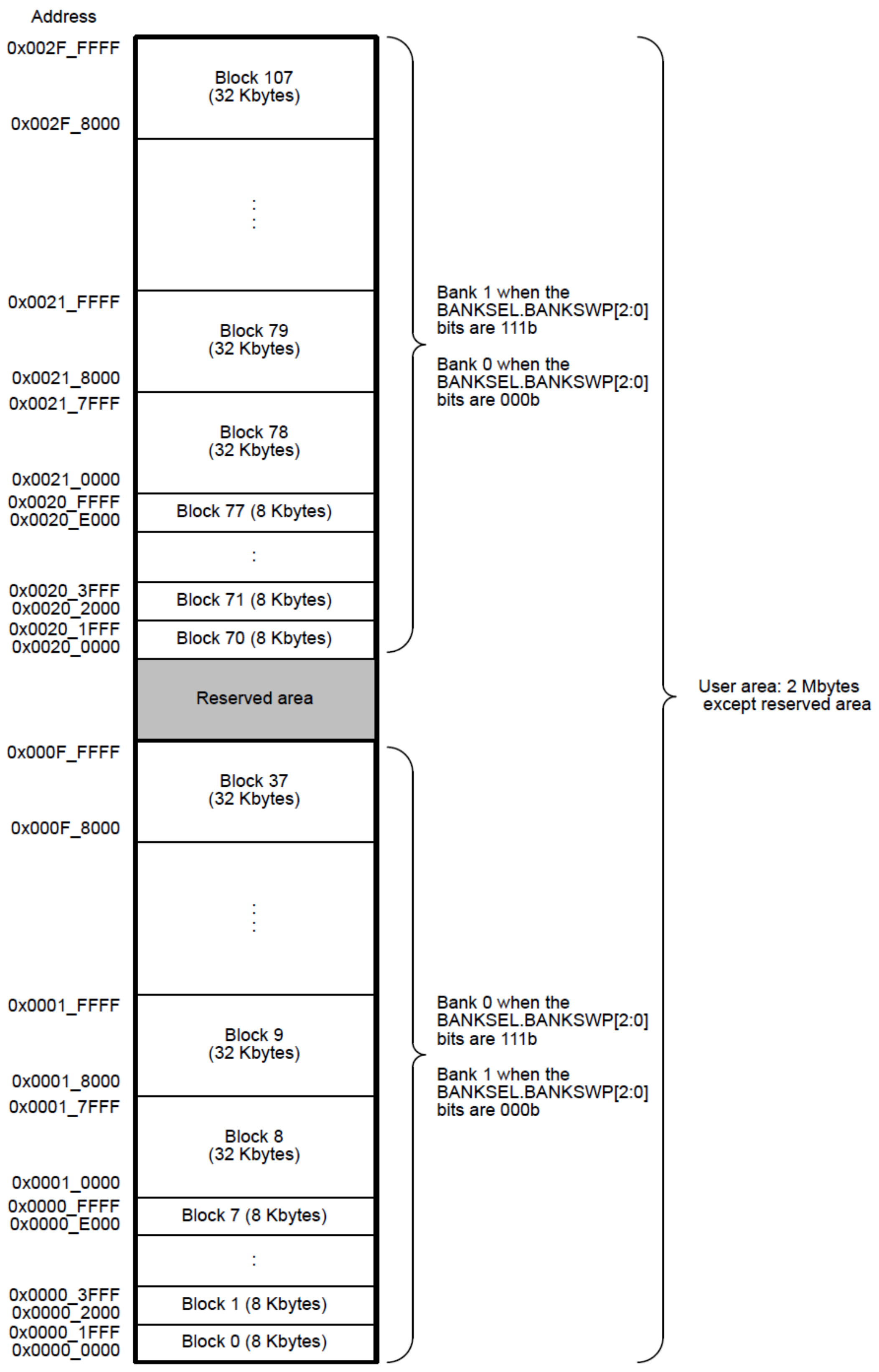


Figure 5. RA6M5 Code Flash Memory in Dual Bank Mode

Table 2 is a summary of the code flash blocks in linear and dual bank mode for the RA6M5. The upper bank address in dual bank mode is 0x200000 regardless of the code flash size. This address should be used with the application image downloader.

Table 2. RA6M5 Code Flash

Product	Code Flash Range Address	
	Linear	**Dual**
2-Mbytes product	0x0000_0000 to 0x001F_FFFF	Lower side bank: 0x0000_0000 to 0x000F_FFFF
		Upper side bank: 0x0020_0000 to 0x002F_FFFF
1-MByte product	0x0000_0000 to 0x000F_FFFF	Lower side bank: 0x0000_0000 to 0x0007_FFFF
		Upper side bank: 0x0020_0000 to 0x0027_FFFF

Figure 6 is the code flash block structure for RA6M5. The code flash erase and programming minimum unit is the code flash block size. The block numbering scheme is used in the block protection design.

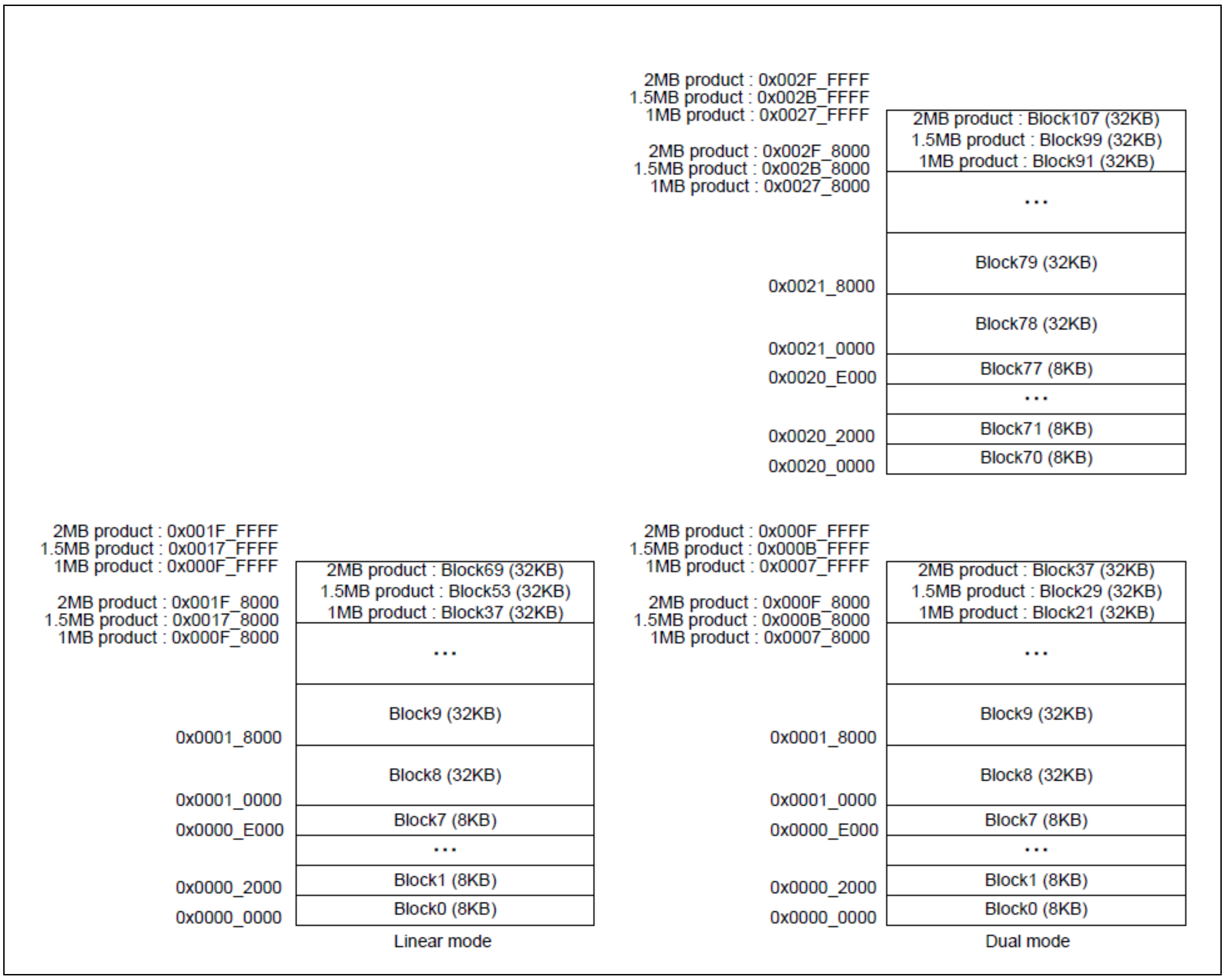

2MB product : 0x002F_FFFF
1.5MB product : 0x002B_FFFF
1MB product : 0x0027_FFFF
2MB product : 0x002F_8000
1.5MB product : 0x002B_8000
1MB product : 0x0027_8000
0x0021_8000
0x0021_0000
0x0020_E000
0x0020_2000
0x0020_0000
2MB product : Block107 (32KB)
1.5MB product : Block99 (32KB)
1MB product : Block91 (32KB)
...
Block79 (32KB)
Block78 (32KB)
Block77 (8KB)
...
Block71 (8KB)
Block70 (8KB)
2MB product : 0x001F_FFFF
1.5MB product : 0x0017_FFFF
1MB product : 0x000F_FFFF
2MB product : 0x001F_8000
1.5MB product : 0x0017_8000
1MB product : 0x000F_8000
0x0001_8000
0x0001_0000
0x0000_E000
0x0000_2000
0x0000_0000
2MB product : Block69 (32KB)
1.5MB product : Block53 (32KB)
1MB product : Block37 (32KB)
...
Block9 (32KB)
Block8 (32KB)
Block7 (8KB)
...
Block1 (8KB)
Block0 (8KB)
Linear mode
2MB product : 0x000F_FFFF
1.5MB product : 0x000B_FFFF
1MB product : 0x0007_FFFF
2MB product : 0x000F_8000
1.5MB product : 0x000B_8000
1MB product : 0x0007_8000
0x0001_8000
0x0001_0000
0x0000_E000
0x0000_2000
0x0000_0000
2MB product : Block37 (32KB)
1.5MB product : Block29 (32KB)
1MB product : Block21 (32KB)
...
Block9 (32KB)
Block8 (32KB)
Block7 (8KB)
...
Block1 (8KB)
Block0 (8KB)
Dual mode

Figure 6. RA6M5 Code Flash Block Structure

1.3 Option-Setting Memory

The description in this section applies to both RA6M4 and RA6M5. The Option-Setting Memory of the RA6M4 and RA6M5 MCUs determines the state of the MCU after a reset. Several property settings that relate to the code flash mode are described in this section.

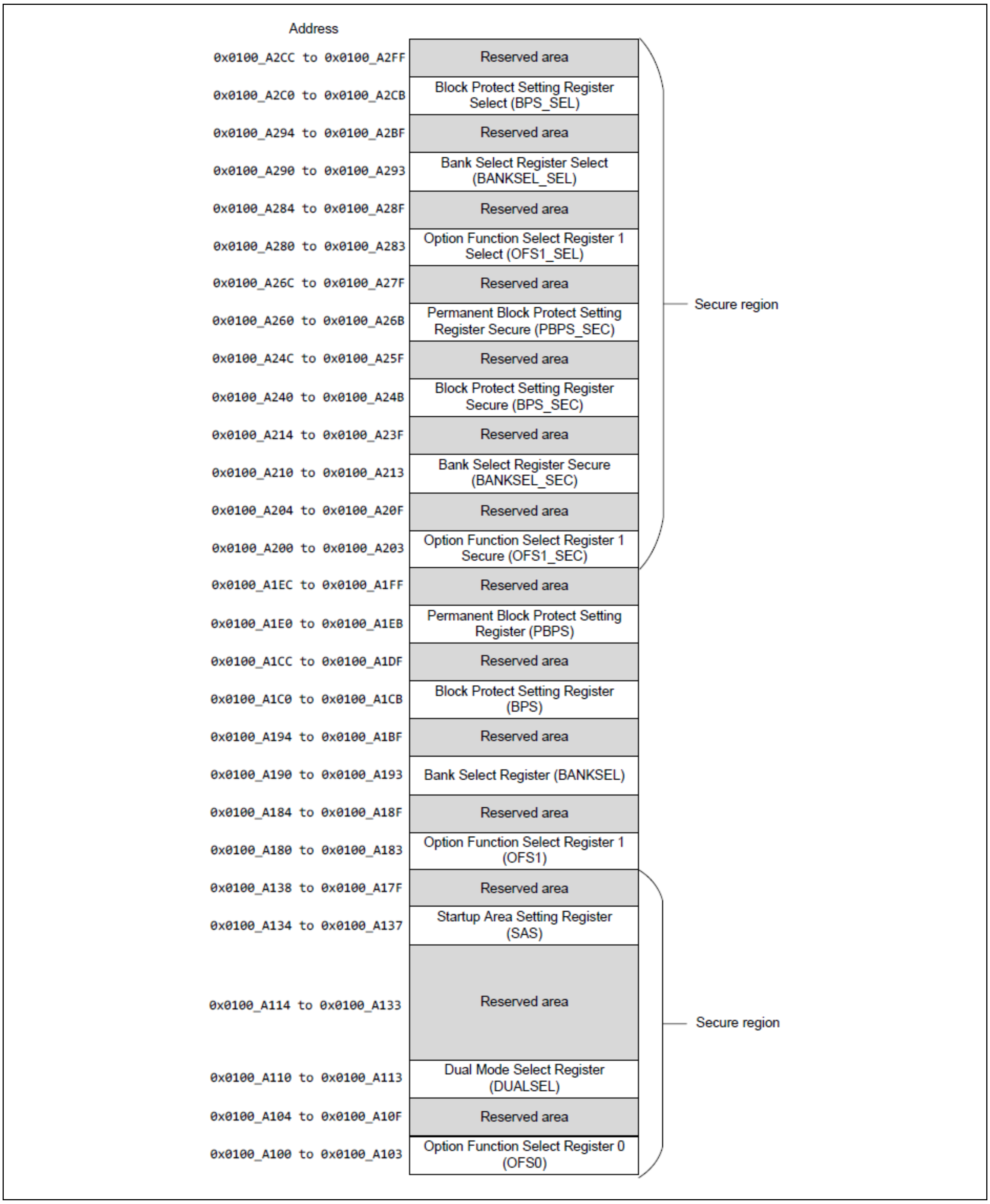
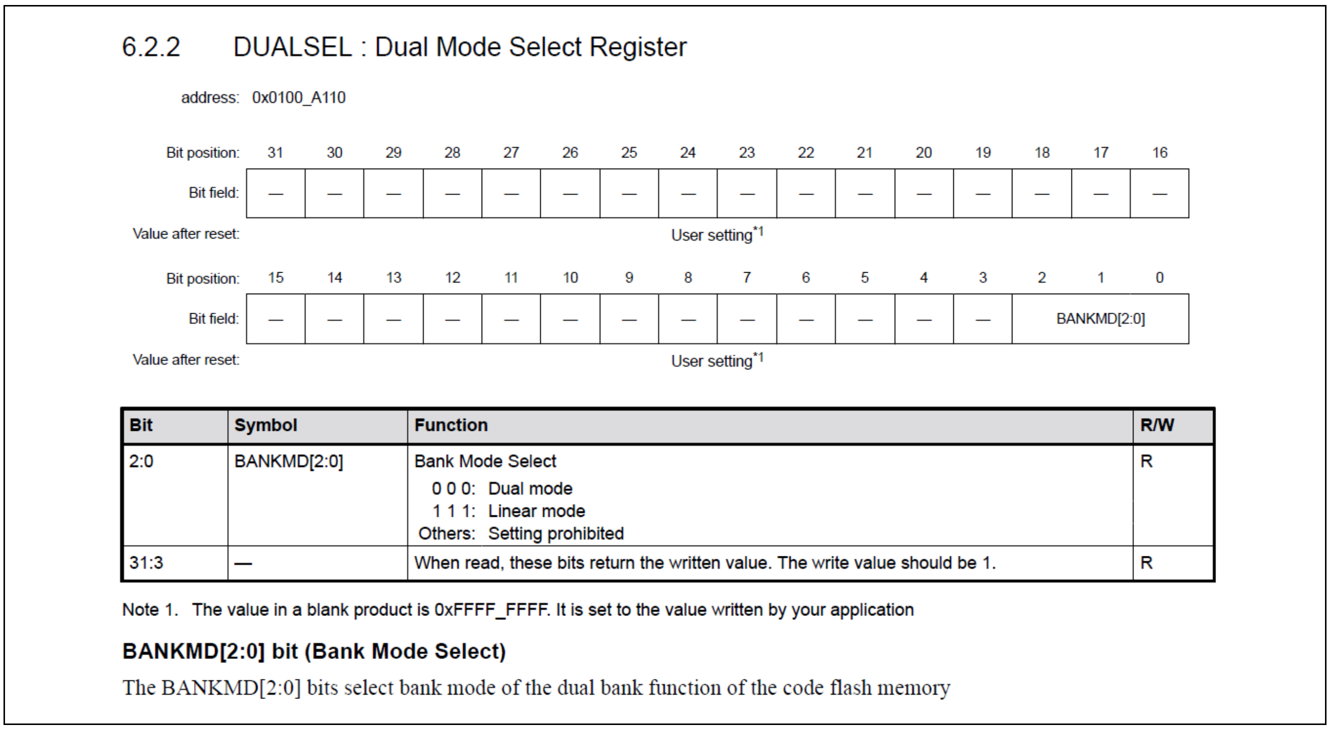

Address
0x0100_A2CC to 0x0100_A2FF Reserved area
0x0100_A2C0 to 0x0100_A2CB Block Protect Setting Register Select (BPS_SEL)
0x0100_A294 to 0x0100_A2BF Reserved area
0x0100_A290 to 0x0100_A293 Bank Select Register Select (BANKSEL_SEL)
0x0100_A284 to 0x0100_A28F Reserved area
0x0100_A280 to 0x0100_A283 Option Function Select Register 1 Select (OFS1_SEL)
0x0100_A26C to 0x0100_A27F Reserved area
0x0100_A260 to 0x0100_A26B Permanent Block Protect Setting Register Secure (PBPS_SEC)
0x0100_A24C to 0x0100_A25F Reserved area
0x0100_A240 to 0x0100_A24B Block Protect Setting Register Secure (BPS_SEC)
0x0100_A214 to 0x0100_A23F Reserved area
0x0100_A210 to 0x0100_A213 Bank Select Register Secure (BANKSEL_SEC)
0x0100_A204 to 0x0100_A20F Reserved area
0x0100_A200 to 0x0100_A203 Option Function Select Register 1 Secure (OFS1_SEC)
Secure region
0x0100_A1EC to 0x0100_A1FF Reserved area
0x0100_A1E0 to 0x0100_A1EB Permanent Block Protect Setting Register (PBPS)
0x0100_A1CC to 0x0100_A1DF Reserved area
0x0100_A1C0 to 0x0100_A1CB Block Protect Setting Register (BPS)
0x0100_A194 to 0x0100_A1BF Reserved area
0x0100_A190 to 0x0100_A193 Bank Select Register (BANKSEL)
0x0100_A184 to 0x0100_A18F Reserved area
0x0100_A180 to 0x0100_A183 Option Function Select Register 1 (OFS1)
0x0100_A138 to 0x0100_A17F Reserved area
0x0100_A134 to 0x0100_A137 Startup Area Setting Register (SAS)
0x0100_A114 to 0x0100_A133 Reserved area
0x0100_A110 to 0x0100_A113 Dual Mode Select Register (DUALSEL)
0x0100_A104 to 0x0100_A10F Reserved area
0x0100_A100 to 0x0100_A103 Option Function Select Register 0 (OFS0)
Secure region

Figure 7. Option-Setting Memory

1.3.1 Code Flash Bank Mode

The register that configures the code flash bank mode is in the Option-Setting Memory of the MCU. As shown in Figure 7, the Dual Mode Select dual bank select register DUALSEL is located at 0x0100A110.

The DUALSEL register defines whether the code flash is in linear or dual bank mode. For a blank MCU, the code flash is in linear mode. The user application can change this configuration. With current FSP support, this register is set up at compile time by configuring the property under the BSP tab.



6.2.2 DUALSEL : Dual Mode Select Register

address: 0x0100_A110

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset:	User setting[*1]															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKMD[2:0]		
Value after reset:	User setting[*1]															

Bit	Symbol	Function	R/W
2:0	BANKMD[2:0]	Bank Mode Select 0 0 0: Dual mode 1 1 1: Linear mode Others: Setting prohibited	R
31:3	—	When read, these bits return the written value. The write value should be 1.	R

Note 1. The value in a blank product is 0xFFFF_FFFF. It is set to the value written by your application

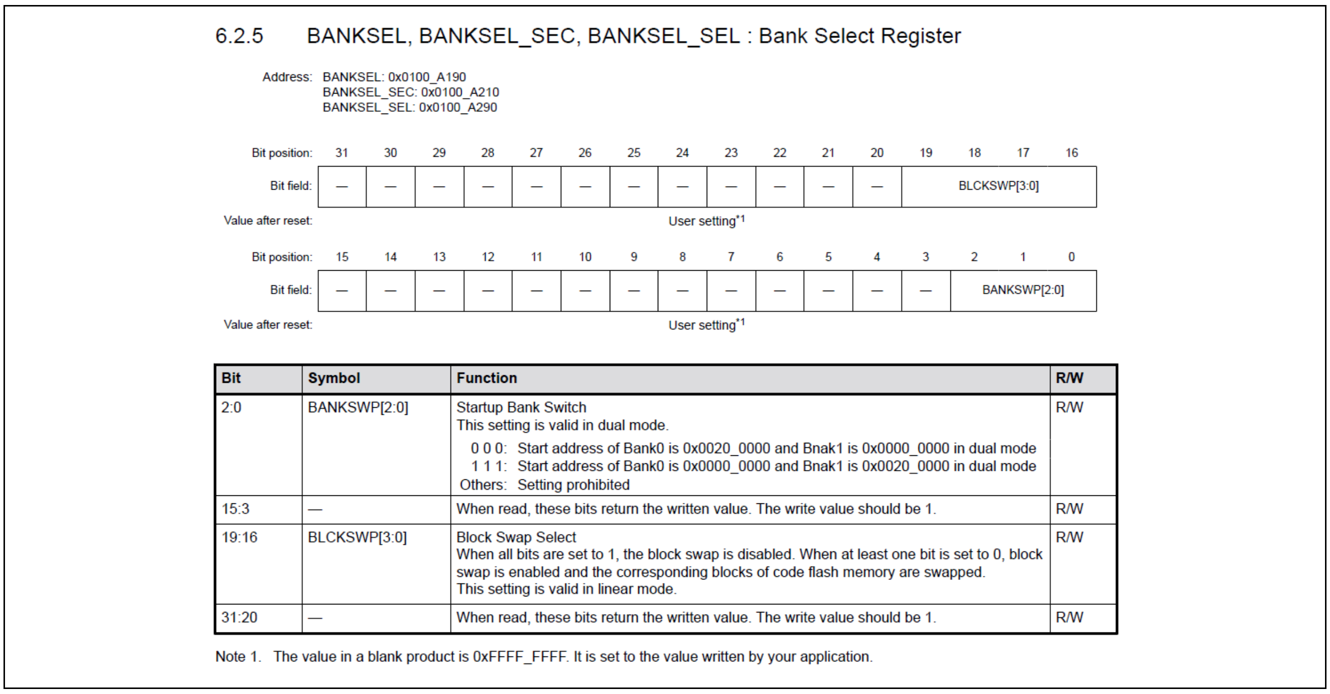
BANKMD[2:0] bit (Bank Mode Select)

The BANKMD[2:0] bits select bank mode of the dual bank function of the code flash memory

Figure 8. Register Configuration for Code Flash Dual Bank Mode

1.3.2 Startup Bank Selection

The description in this section applies to both RA6M4 and RA6M5 MCUs. Bank 0 is the lower bank for a blank RA6M4 or RA6M5 MCU as defined by the Bank Select registers shown in Figure 9.



6.2.5 BANKSEL, BANKSEL_SEC, BANKSEL_SEL : Bank Select Register

Address: BANKSEL: 0x0100_A190
BANKSEL_SEC: 0x0100_A210
BANKSEL_SEL: 0x0100_A290

Bit position:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	BLCKSWP[3:0]			
Value after reset:	User setting[*1]															

Bit position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit field:	—	—	—	—	—	—	—	—	—	—	—	—	—	BANKSWP[2:0]		
Value after reset:	User setting[*1]															

Bit	Symbol	Function	R/W
2:0	BANKSWP[2:0]	Startup Bank Switch This setting is valid in dual mode. 0 0 0: Start address of Bank0 is 0x0020_0000 and Bnak1 is 0x0000_0000 in dual mode 1 1 1: Start address of Bank0 is 0x0000_0000 and Bnak1 is 0x0020_0000 in dual mode Others: Setting prohibited	R/W
15:3	—	When read, these bits return the written value. The write value should be 1.	R/W
19:16	BLCKSWP[3:0]	Block Swap Select When all bits are set to 1, the block swap is disabled. When at least one bit is set to 0, block swap is enabled and the corresponding blocks of code flash memory are swapped. This setting is valid in linear mode.	R/W
31:20	—	When read, these bits return the written value. The write value should be 1.	R/W

Note 1. The value in a blank product is 0xFFFF_FFFF. It is set to the value written by your application.

Figure 9. Bank 0 is Default at Address 0x00000000

Only secure developers can program BANKSEL_SEC and BANKSEL_SEL registers. BANKSEL_SEC register is for secure developers, and BANKSEL register is for non-secure developers.

BANKSEL_SEL controls whether the BANKSEL or BANKSEL_SEC setting is applied. When BANKSEL_SEL is 0xFFFFFFF8, the setting in BANKSEL is used. When BANKSEL_SEL is 0xFFFFFFFF, the setting in BANKSEL_SEC is used. For Non-Trust Zone based Flat projects, BANKSEL_SEL selects the corresponding bits in the BANKSEL_SEC register.

1.3.3 Bank Swap

Startup bank selection provides a way to safely update the program by selecting a bank area to be started in dual mode during a reset.

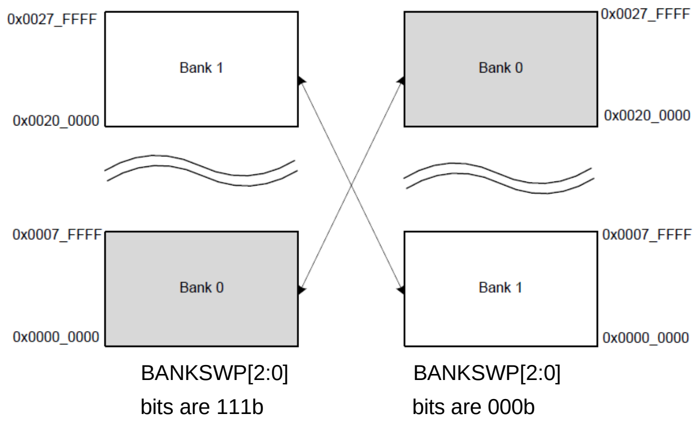


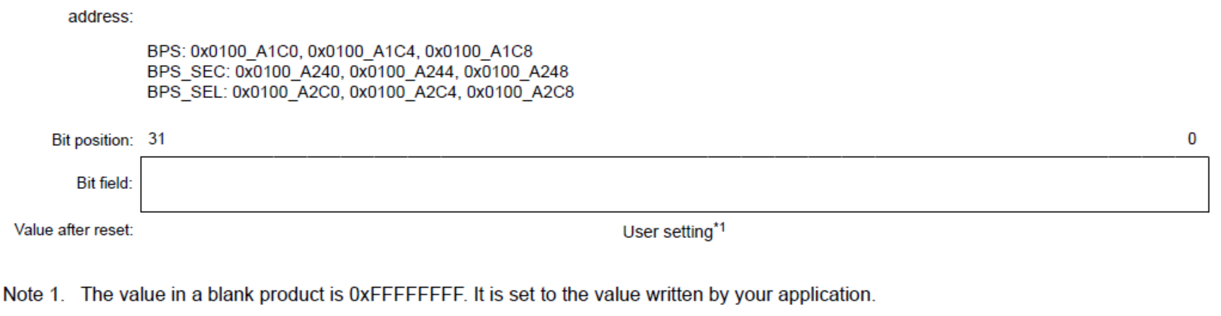
Figure 10. Example of Startup Bank Selection (For Products with 1 Mbyte of Code Flash Memory)

Bank selection can be changed at runtime through the FSP API. The BANKSWP bits in the BANKSEL register can be changed at the application level. The FSP flash driver provides the `R_FLASH_HP_BankSwap()` API to facilitate this action. This API is automatically called from the FSP MCUboot module. The swap takes affect after the next reset.

1.3.4 Code Flash Block Protection

The RA6M4 and RA6M5 MCUs implement a security function to protect the code flash against illicit tampering with or reading out of data in flash memory. The registers that define this security function reside in the Option-Setting Memory. The code flash memory can be temporally or permanently protected from programming/erasure operation.

The registers that support the temporary code flash block protection reside in the Option-Setting Memory:


address:

BPS: 0x0100_A1C0, 0x0100_A1C4, 0x0100_A1C8
BPS_SEC: 0x0100_A240, 0x0100_A244, 0x0100_A248
BPS_SEL: 0x0100_A2C0, 0x0100_A2C4, 0x0100_A2C8

Bit position: 31 0

Bit field:

Value after reset: User setting*1

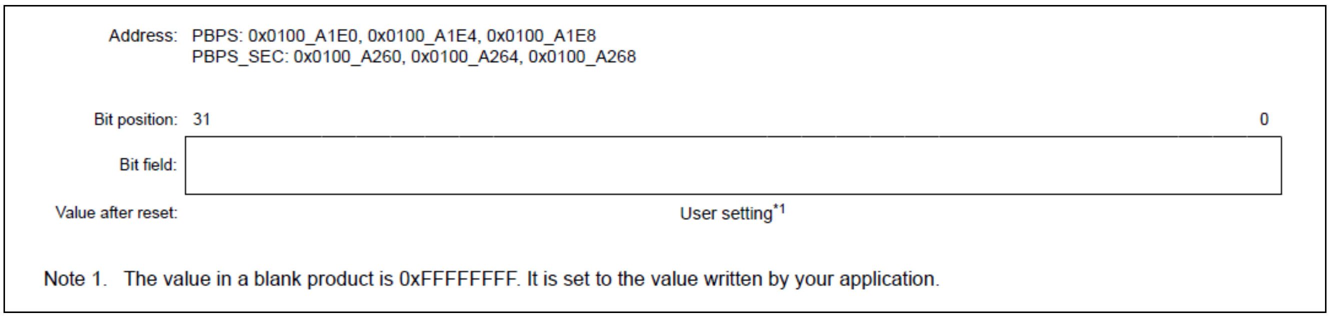
Note 1. The value in a blank product is 0xFFFFFFFF. It is set to the value written by your application.

Figure 11. Registers Related to Temporary Code Flash Block Protection

Only secure developers can program the BPS_SEC and BPS_SEL registers. The BPS_SEC register is for secure developers, and the BPS register is for non-secure developers. The applied setting value is determined by the value of the corresponding bit in BPS_SEL register. BPS_SEL controls whether the BPS or BPS_SEC setting is applied. When BPS_SEL is 0xFFFFFFF8, the setting in BPS is used. When BPS_SEL is 0xFFFFFFFF, the setting in BPS_SEC is used. For Non-Trust Zone based Flat projects, BSP_SEL selects the corresponding bits in the BSP_SEC register. The BPS and BPS_SEC registers invalidate the programming and erasure to the code flash memory. When a BPS/BPS_SEC bit is 0, the programming and erasure to the corresponding block are invalid.

These registers can be set by configuring the BSP stack in the RA configurator as shown in Figure 87 and Figure 88.

The registers that support the permanent code flash block protection reside in the Option-Setting Memory:


Address: PBPS: 0x0100_A1E0, 0x0100_A1E4, 0x0100_A1E8
PBPS_SEC: 0x0100_A260, 0x0100_A264, 0x0100_A268

Bit position: 31 … 0

Bit field:

Value after reset: User setting*1

Note 1. The value in a blank product is 0xFFFFFFFF. It is set to the value written by your application.

Figure 12. Registers Related with Permanent Code Flash Block Protection

Only secure developers can program the PBPS_SEC and PBPS_SEL registers. The PBPS_SEC register is for secure developers, and the PBPS register is for non-secure developers. The applied setting value is determined by the set value of the corresponding bit in the PBPS_SEL register. PBPS_SEL controls whether the BPS or BPS_SEC setting is applied. When PBPS_SEL is 0xFFFFFFF8, the setting in PBPS is used. When PBPS_SEL is 0xFFFFFFFF, the setting in PBPS_SEC is used. For Non-Trust Zone based Flat projects, PBSP_SEL selects the corresponding bits in the PBSP_SEC register. The PBPS and PBPS_SEC registers invalidate the programming and erasure to the code flash memory. When a PBPS/PBPS_SEC bit is 0, the programming and erasure to the corresponding block are invalid.

Setting of these registers can be achieved by configuring the BSP Properties in the RA configurator as shown in Figure 89 and Figure 90.

2. Using the Code Flash Dual Bank Feature with MCUboot Overview

MCUboot evolved out of the Apache Mynewt bootloader, which was created by runtime.io. MCUboot was then acquired by JuulLabs in November 2018. The MCUboot github repo was later migrated from JuulLabs to the mcu-tools github project. In year 2020, MCUboot was moved under the Linaro Community Project umbrella as an open-source project.

2.1 MCUboot Functionalities Overview

MCUboot handles the firmware authenticity check after start-up and the firmware switch part of the firmware update process. Downloading the new version of the firmware is out-of-scope for MCUboot. Typically, downloading the new version of the firmware is functionality that is provided by the application project itself. This application project provides an example of downloading a new image using the XModem protocol from the application project.

The functionality of MCUboot during booting and updating follows the process below:

The bootloader starts when the CPU is released from reset. If there are images in the Secondary App memory marked as to be updated, the bootloader performs the following actions:

1. The bootloader authenticates the Secondary image.
2. Upon successful authentication, the bootloader switches to the new image based on the update method selected. Available update methods supported by FSP are overwrite, swap, and direct XIP.
3. The bootloader boots the new image.

If there is no new image in the Secondary App memory region, the bootloader authenticates the Primary applications and boots the Primary image.

The authentication of the application is configurable in terms of the authentication methods and whether the authentication is to be performed with MCUboot. If authentication is to be performed, the available methods are RSA or ECDSA. The firmware image is authenticated by hash (SHA-256) and digital signature validation. The public key used for digital signature validation can be built into the bootloader image or provisioned into the MCU during manufacturing. In the examples included in this application project, the public key is built into the bootloader images.

There is a signing tool included with MCUboot: `imgtool.py`. This tool provides services for creating Root keys, key management, and signing and packaging an image with version controls. Read the MCUboot documentation to use and understand these operations.

2.2 Using MCUboot for Code Flash Dual Bank Mode

The FSP supports overwrite, swap, and direct XIP (execute-in-place) update mode. For flash dual bank mode, only direct XIP mode is supported. The benefits of using code flash dual bank mode in a system including a bootloader are concurrent download of new image and faster switching to the new image, in addition to the safety features provided by the MCUboot module as explained in section 2.2.1.

2.2.1 Use Direct XIP Firmware Update Mode

When using direct XIP mode with code flash in linear mode, the active image slot alternates with each firmware update. If this update method is used, then two firmware update images must be generated: one of them is linked to be executed from the primary slot memory region, and the other is linked to be executed from the secondary slot. Direct XIP is supported in FSP versions 3.6.0 and later.

- Advantages:
  - Faster boot time, as there is no overwrite or swap of application images needed.
  - Fail-safe and resistant to power-cut failures.
- Disadvantages:
  - Added application-level complexity to determine which firmware image needs to be downloaded.
  - Encrypted image support is not available.

For overview and usage of other update modes, refer to R11AN0497 and the MCUboot design page:

https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md

When using direct XIP mode with code flash in dual bank mode, both primary and secondary images are linked to be executed from the primary slot memory region. When using the RA6M4 or RA6M5 MCU with flash dual bank mode, only Non-TrustZone based Flat project is currently supported by the FSP.

Note: For Direct XIP mode, downgrade prevention is supported from the MCUboot side. When using flash dual bank mode, the update image needs to have a version number higher than the current primary image.

2.2.2 Memory Configuration Overview with Dual Bank and MCUboot

The FSP MCUboot module with Flash Dual Bank mode needs a bootloader for both the lower bank and the upper bank as shown in Figure 13. In addition, the memory allocation for the bootloader and application image must be identical.

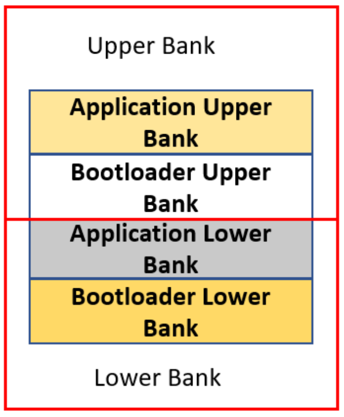


Figure 13. Memory Architecture Using Flash Dual Bank Mode and MCUboot

2.3 Designing Bootloader and Initial Primary Application Overview

A bootloader is typically designed with the initial primary application. The following general guidelines apply to designing the bootloader and the initial primary application:

- Develop the bootloader and analyze the MCU memory resource allocation needed for the bootloader and the application. The bootloader memory usage is influenced by the application image update mode, signature type, and whether to validate the Primary Image as well as the cryptographic library used.
- Develop the initial primary application, perform the memory usage analysis, and compare with the bootloader memory allocation for consistency and adjust as needed.
- Determine the bootloader configurations in terms of image authentication and new image update mode. This may result in adjustment of the memory allocated definition in the bootloader project.
- Sign the application image. The signing command is output to the <bootloader project>\Debug\>bootloader project>.bld file. The application image can use a Build Variable to access this .bld file. The IDE tools use the signing command to sign the application and generate a binary file for downloading to the MCU.
- Test the bootloader and the initial primary application.

The above guidelines are demonstrated in the walk-through sections in this application note.

2.4 Migrating an Existing Linear Mode MCUboot Based System

Users can follow the general steps below to migrate an MCUboot based application system from code flash linear mode to code flash dual bank mode:

1. Updates for the bootloader project:
   A. Update the code flash mode from linear mode to dual mode in the BSP tab, as shown in Figure 34.
   B. Update the application image code flash allocation if needed. See section 4.2 for details.
2. Updates for the application projects:
   A. For image downloader implementation, the image download address needs to be updated. Refer to the \src\Header.h in the example application project to understand where the updates need to happen.
   B. For development purpose, the debug configuration for the primary application needs to be updated. Refer to the debug configuration for the app_primary_usb project under the \example_projects_with_bootloader folder.
   C. For production support, the scripts to generate the .srec file using the signed image need to be updated. Refer to section 5.3 to understand the updates needed.

3. Guidelines for Using the Example Projects Included

Unzip `RA6_Secure_Bootloader_DualBank.zip` to unpack the example projects included in this application project.

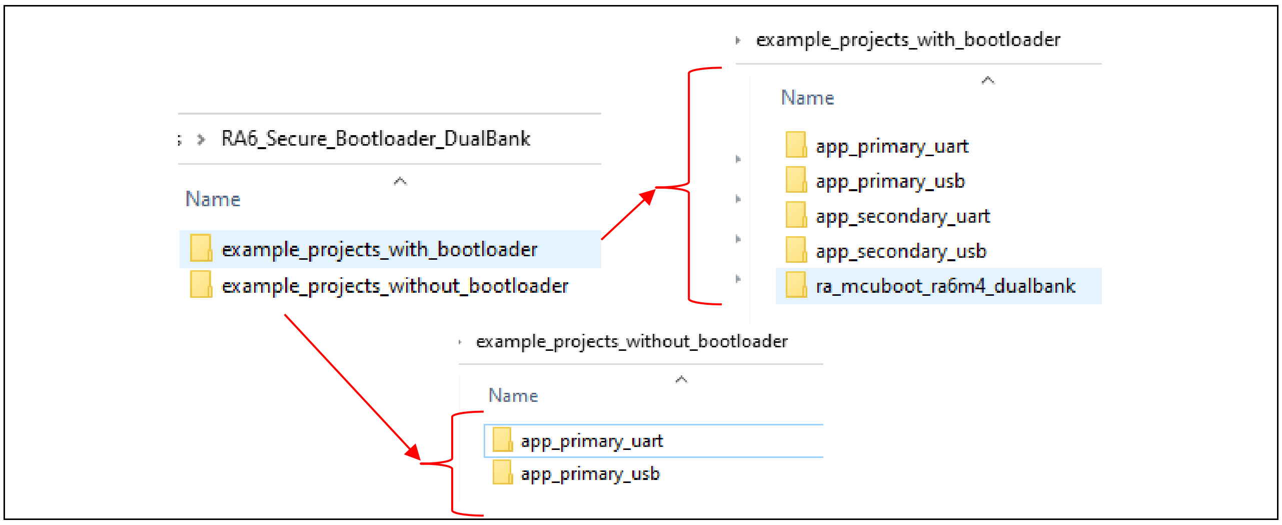

RA6_Secure_Bootloader_DualBank
Name
example_projects_with_bootloader
example_projects_without_bootloader
example_projects_with_bootloader
Name
app_primary_uart
app_primary_usb
app_secondary_uart
app_secondary_usb
ra_mcuboot_ra6m4_dualbank
example_projects_without_bootloader
Name
app_primary_uart
app_primary_usb

Figure 14. Example Projects Included

3.1 Example Projects with Bootloader

Folder `\example_projects_with_bootloader` includes a bootloader, which supports the flash dual bank feature, as well as example applications using USB or UART as the communication channel to download new application images which are configured to use the bootloader included in this folder. Users with experience working with MCUboot module can follow section 8 to directly exercise these example projects. The corresponding subfolders are:

- `ra_mcuboot_ra6m4_dualbank:` Bootloader, which enables dual bank and direct XIP update mode.
- `app_primary_usb:` Primary application, which is configured to work with the bootloader and implements XModem over USB VCOM to download a new application image. FreeRTOS is used with two threads, one threads blinks the three LEDs on EK-RA6M4 while the other thread downloads the new application image concurrently.
- `app_secondary_usb:` Secondary application, which implements the same functionality as `app_primary_usb` except only the blue and green LEDs are blinked.
- `app_primary_uart:` Primary application, which is configured to work with the bootloader and implements XModem over UART to download a new application image. FreeRTOS is used with two threads, one threads blinks the three LEDs on EK-RA6M4 while the other thread downloads the new application image concurrently.
- `app_secondary_uart:` Secondary application, which implements the same functionality as `app_primary_uart` except only the blue and green LEDs are blinked.

3.2 Example Projects without Bootloader

Folder `\example_projects_without_bootloader` includes standalone example projects that a user can configure to use the bootloader project, following section 5. Note that these application projects do not run correctly if the flash dual bank mode is not enabled because the image downloader routine included assumes the location of the new image is in the upper bank of the RA6M4 code flash. The subfolders are:

- `app_primary_usb:` Same functionality as `\example_projects_with_bootloader\app_primary_usb`, except it is not configured to work with the bootloader.
- `app_primary_uart:` Same functionality as `\example_projects_with_bootloader\app_primary_uart`, except it is not configured to work with the bootloader.

A user can also use a customized application project that implements image downloading and follow section 5 to use the bootloader.

4. Creating the Bootloader Project using Code Flash Dual Bank Mode

This section demonstrates the creation process of the bootloader project utilizing MCUboot and the Flash Dual Bank Mode with the RA6M4 running in Non-TrustZone mode.

4.1 Include the MCUboot Module in the Bootloader Project

Follow below steps to start the bootloader project creation and include the MCUboot module in the project:

1. Launch e^2 studio and start a new C/C++ Project. Click **File** > **New** > **C/C++ Project.**

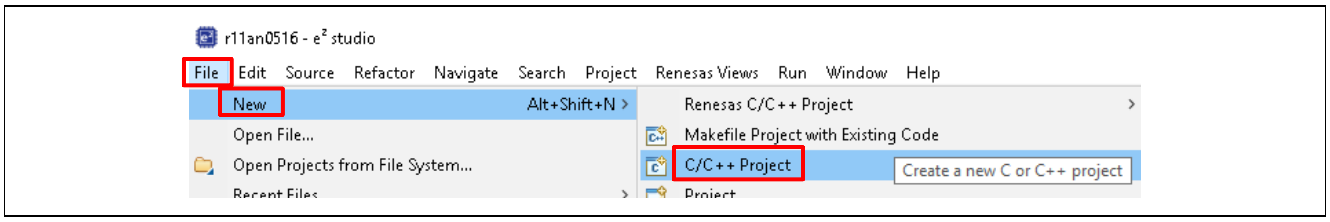
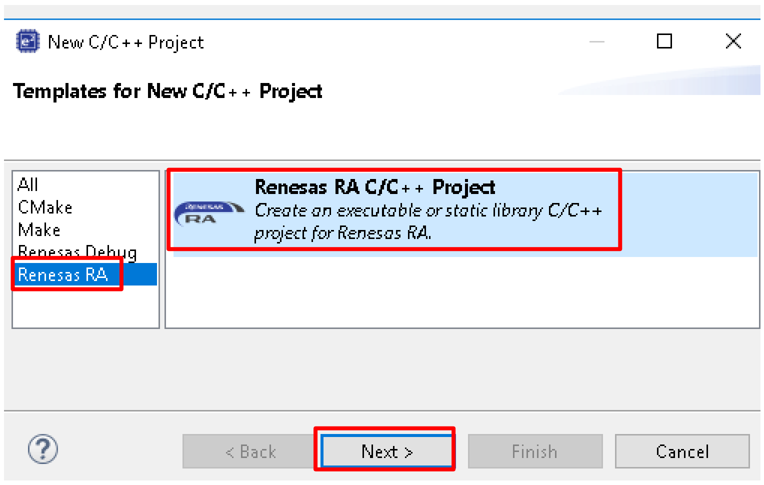


Figure 15. Start a New Project

2. Choose **Renesas RA->Renesas RA C/C++ Project**. Click **Next**.


New C/C++ Project

Templates for New C/C++ Project

All
CMake
Make
Renesas Debug
Renesas RA

Renesas RA C/C++ Project
Create an executable or static library C/C++ project for Renesas RA.

Figure 16. Choose Renesas RA C/C++ Project

3. Provide the project name **`ra_mcuboot_ra6m4_dualbank`** in the next screen. Click **Next**.
4. In the next screen, choose **EK-RA6M4** for **Board** and click **Next**.

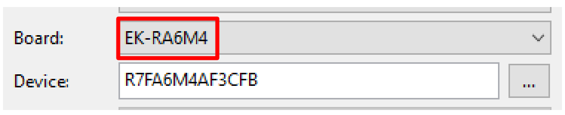
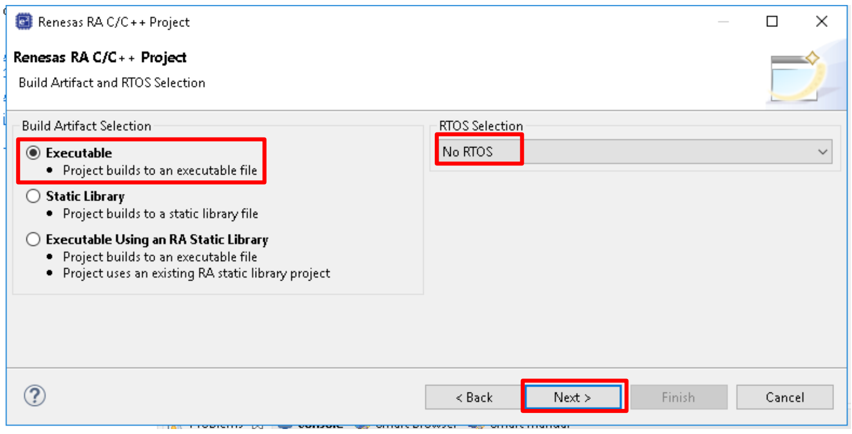

Board: EK-RA6M4
Device: R7FA6M4AF3CFB

Figure 17. Select the Board

5. Choose **Executable** for **Build Artifact Selection** and **No RTOS**. Click **Next**.


Renesas RA C/C++ Project

Renesas RA C/C++ Project
Build Artifact and RTOS Selection

Build Artifact Selection
- Executable
  - Project builds to an executable file
- Static Library
  - Project builds to a static library file
- Executable Using an RA Static Library
  - Project builds to an executable file
  - Project uses an existing RA static library project

RTOS Selection
No RTOS

Figure 18. Choose to Build Executable and No RTOS

6. Choose **Bare Metal – Minimal** for the Project Template in the next screen and click **Finish** to establish the initial project.

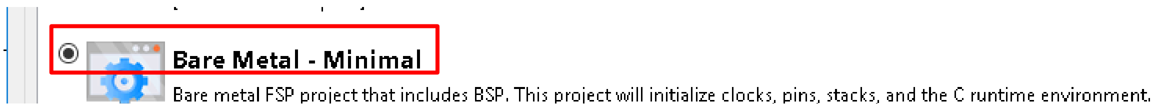

Bare Metal - Minimal
Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.

Figure 19. Choose the Project Template

7. When the following prompt opens, click **Open Perspective**.

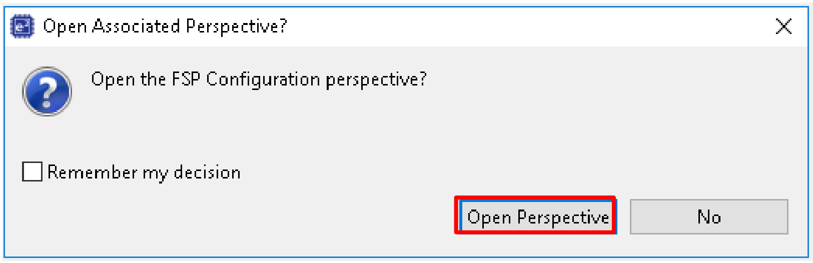

Open Associated Perspective?
Open the FSP Configuration perspective?
Remember my decision
Open Perspective
No

Figure 20. Choose Open the FSP Configuration Perspective

The project is created and the bootloader project configuration is displayed.

8. Select the **Pins** tab and uncheck **Generate data** for **RA6M4 EK.**

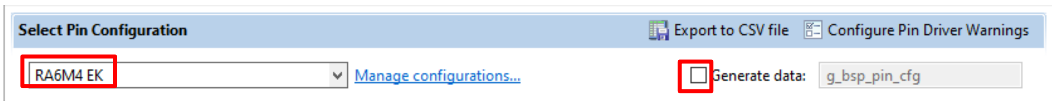


Figure 21. Uncheck Generate data for RA6M4 EK Pin Configuration

Use the pull-down menu to switch from **RA6M4 EK** to **R7FA6M4AF3CFB.pincfg** for the **Select Pin Configuration** option, then select the **Generate data** check box and enter **g_bsp_pin_cfg**. Note that here we choose to use this configuration which has fewer peripherals/pins configured since the bootloader does not use the extra peripheral or GPIO pins configured in the **RA6M4 EK** configuration. This also reduces some memory usage for the bootloader project.

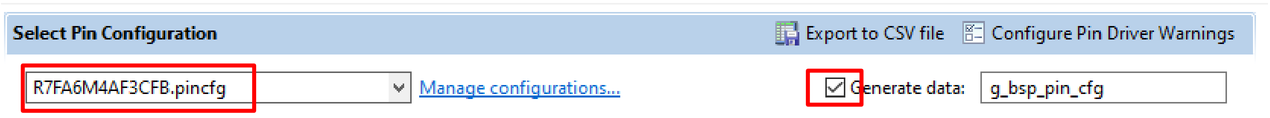


Figure 22. Select g_bsp_pin_cfg and Generate data g_bsp_pin_cfg

9. Once the project is created, click the **Stacks** tab on the RA configurator. Add **New Stack->Bootloader-> MCUboot**.

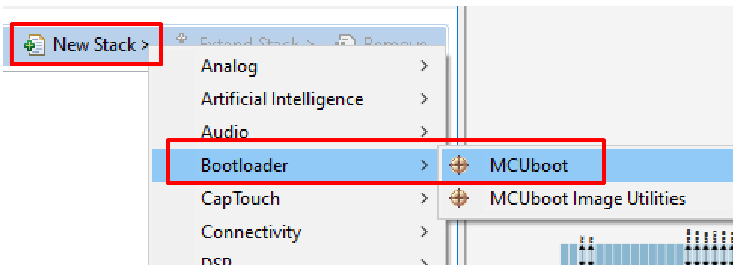
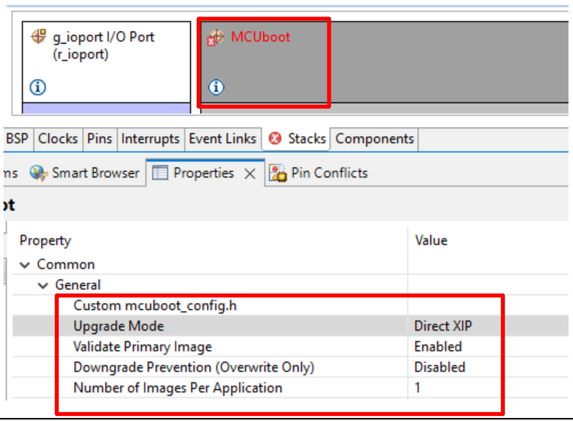


Figure 23. Add the MCUboot Port

10. Next, configure the **General** properties of **MCUboot**. We will resolve the errors in the configurator in the following steps.
    For the MCUboot module, configure the **Update Mode** to **Direct XIP** and **Number of Images Per Application** to **1**.



Property	Value
Common	
General	
Custom mcuboot_config.h	
Upgrade Mode	Direct XIP
Validate Primary Image	Enabled
Downgrade Prevention (Overwrite Only)	Disabled
Number of Images Per Application	1

Figure 24. General Configuration for MCUboot Module

The properties configured are:

- **Custom mcuboot_config.h**: The default `mcuboot_config.h` file contains the MCUboot Module configuration that the user selected from the RA configurator. The user can create a custom version of this file to achieve additional bootloader functionalities available in MCUboot.
- **Upgrade Mode**: This property configures the application image upgrade method. The available options are Overwrite Only, Overwrite Only Fast, Swap and Direct XIP. Only Direct XIP is supported for flash dual bank operation.
- **Validate Primary Image**: When enabled, the bootloader will perform a hash or signature verification,depending on the verification method chosen, in addition to the MCUboot magic number based sanity check. When disabled, only a sanity check is performed based on the MCUboot magic number.
- **Number of Images Per Application:** This property allows user to choose one image for Non-TrustZone-based applications and two images for TrustZone-based applications. Set this property to 1.
- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is **Enabled**, a new firmware with a lower version number will not overwrite the existing application.

Note: For Direct XIP mode, download grade prevention is supported from the MCUboot side. When using flash dual bank mode, the update image needs to have a version number higher than the current primary image.

4.2 Configure the Memory Configuration and Authentication Method

Configure the Signing Options and Flash Layout of the MCUboot module. For the EK-RA6M4, the default memory for the code flash dual bank mode is shown in Figure 25. This default memory map is used for the example bootloader design.

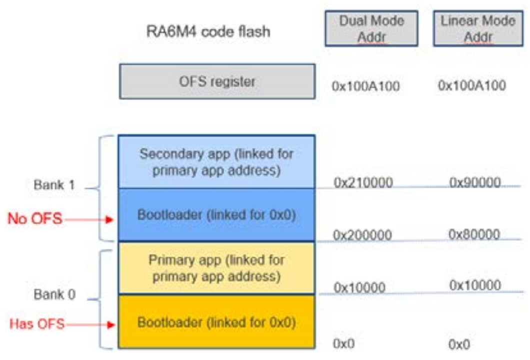
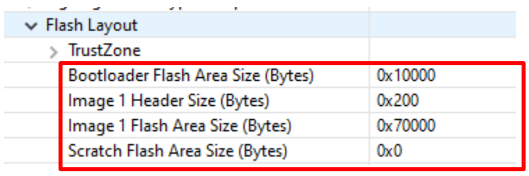


Figure 25. MCUboot Dual Bank Memory Map

From the configurator point of view, there is no need to update any of the properties for the Flash Layout as it already matches with the memory map shown in Figure 25.



Flash Layout	
TrustZone	
Bootloader Flash Area Size (Bytes)	0x10000
Image 1 Header Size (Bytes)	0x200
Image 1 Flash Area Size (Bytes)	0x70000
Scratch Flash Area Size (Bytes)	0x0

Figure 26. Configure the Flash Layout and Signing Options

Explanation of the Above Configurations:

- **Bootloader Flash Area**: Size of the flash area allocated for the bootloader, with a boundary of 0x8000 since 0x8000 is the minimum erase size for RA6M4 code flash.
- **Image 1 Header Size**: Size of the code flash reserved for the application image header. It must meet minimum VTOR alignment requirements based on the number of interrupts implemented on the RA6M4. For the RA6M4, this property should be set to a minimum of 0x200 to support all interrupts.
- **Image 1 Flash Area Size**: Size of application image 1, including the header and trailer. For the RA6M4, this size needs to be on a boundary of 0x8000 which is the smallest flash erase size.
- **Scratch Flash Area Size**: This property is only needed for Swap mode. This property is not used for the flash dual bank bootloader design.
- **Signature Type:** Signing algorithm selection. The choices are:
  - **NONE:** Select this option for bootloaders that do not support signature verification.
  - **ECDSA P-**256: Select this option for this example bootloader design.
  - **RSA 2048 and RSA 3072**
  - Application images using MCUboot must be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.
- **Custom:** Use the default `--confirm` for this bootloader design. Switching to a new image is always confirmed, and the new image will be booted after a subsequent system reset. Reverting the image with Direct XIP is not supported with the current FSP version.

4.3 Configure the MbedTLS Crypto Only Module and the Flash Driver

Follow steps below to configure the MbedTLS module and the flash driver:

1. Right-click on **Add Crypto Stack** and choose to add the **MbedTLS (Crypto Only)** module.

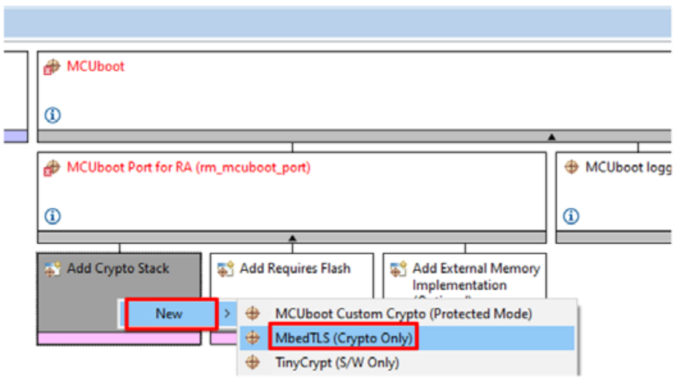

MCUboot
MCUboot Port for RA (rm_mcuboot_port)
MCUboot logg
Add Crypto Stack
Add Requires Flash
Add External Memory Implementation
New
MCUboot Custom Crypto (Protected Mode)
MbedTLS (Crypto Only)
TinyCrypt (S/W Only)

Figure 27. Select MbedTLS Crypto Only Module

2. Click on **Add Requires Flash** stack and select Flash (r_flash_hp) stack.

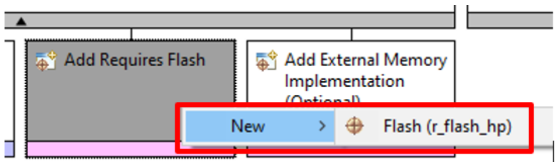
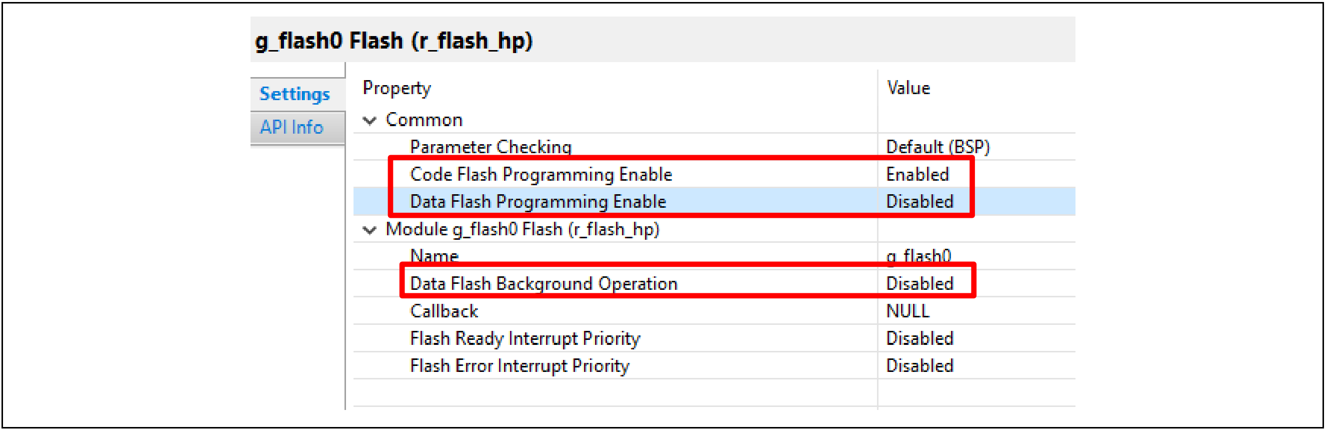


Figure 28. Add the Flash Driver

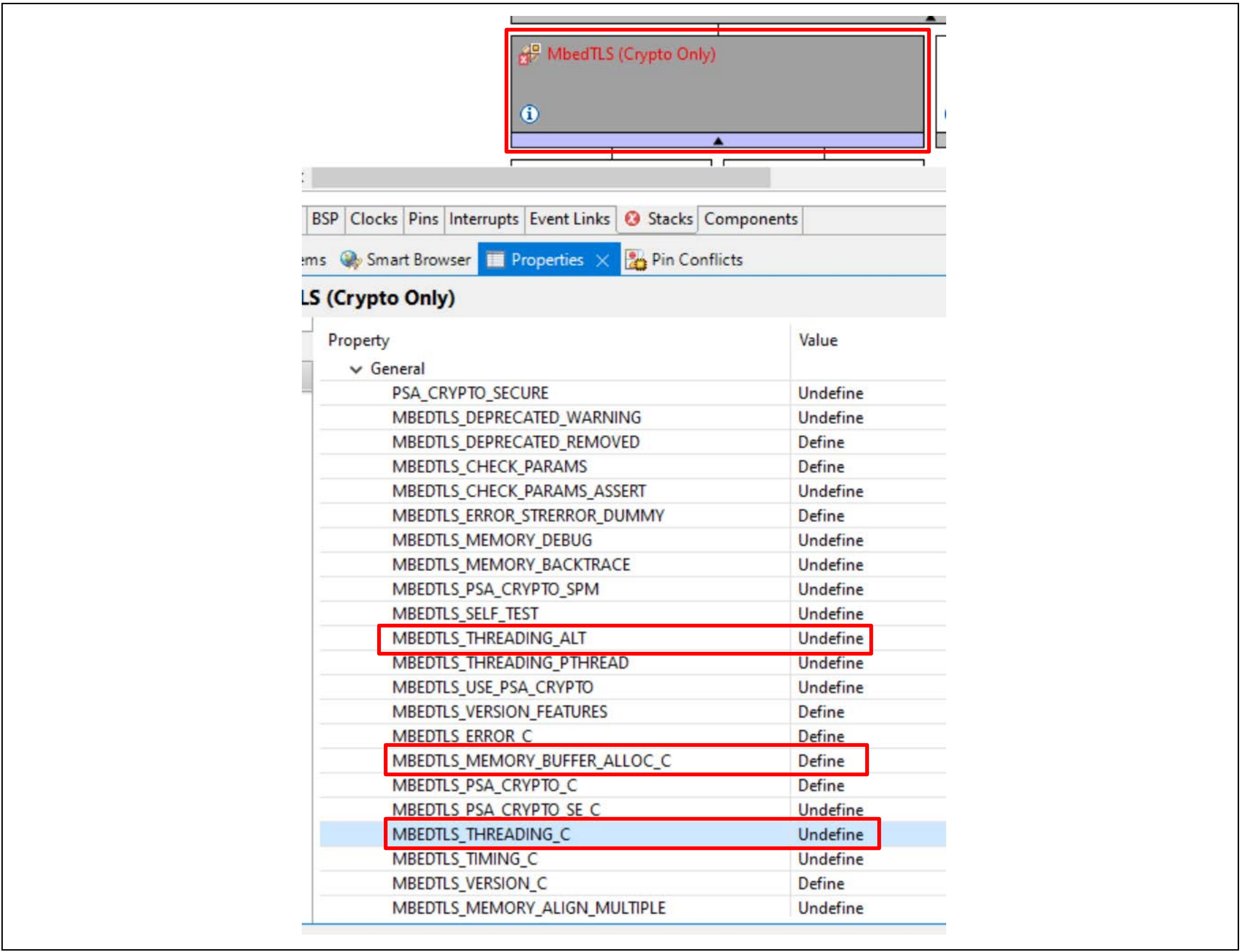
3. Next, set the **Code Flash Programming** to **Enabled**. As **Data Flash Programming** is not used in the bootloader, select **Disabled** for the **Data Flash Programming** to reduce the bootloader memory footprint.


g_flash0 Flash (r_flash_hp)

Property	Value
Common	
Parameter Checking	Default (BSP)
Code Flash Programming Enable	Enabled
Data Flash Programming Enable	Disabled
Module g_flash0 Flash (r_flash_hp)	
Name	g_flash0
Data Flash Background Operation	Disabled
Callback	NULL
Flash Ready Interrupt Priority	Disabled
Flash Error Interrupt Priority	Disabled

Figure 29. Configure the Flash Driver

4. Configure the following properties of the MbedTLS (Crypto Only) module:

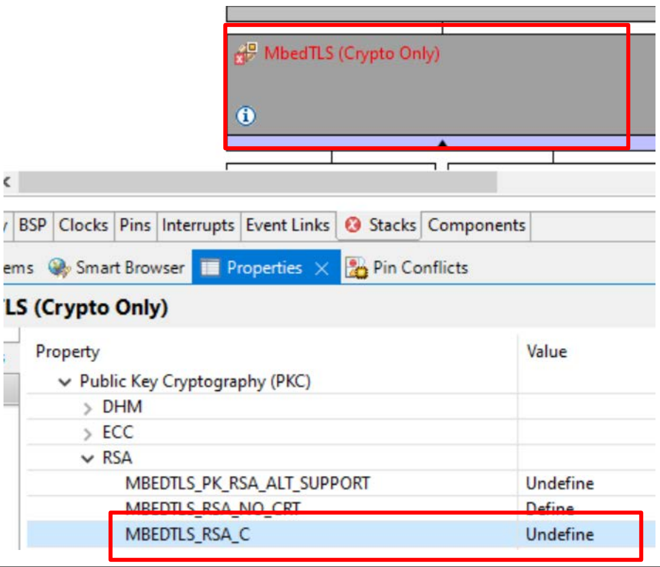


LS (Crypto Only)

Property	Value
General	
PSA_CRYPTO_SECURE	Undefine
MBEDTLS_DEPRECATED_WARNING	Undefine
MBEDTLS_DEPRECATED_REMOVED	Define
MBEDTLS_CHECK_PARAMS	Define
MBEDTLS_CHECK_PARAMS_ASSERT	Undefine
MBEDTLS_ERROR_STRERROR_DUMMY	Define
MBEDTLS_MEMORY_DEBUG	Undefine
MBEDTLS_MEMORY_BACKTRACE	Undefine
MBEDTLS_PSA_CRYPTO_SPM	Undefine
MBEDTLS_SELF_TEST	Undefine
MBEDTLS_THREADING_ALT	Undefine
MBEDTLS_THREADING_PTHREAD	Undefine
MBEDTLS_USE_PSA_CRYPTO	Undefine
MBEDTLS_VERSION_FEATURES	Define
MBEDTLS_ERROR_C	Define
MBEDTLS_MEMORY_BUFFER_ALLOC_C	Define
MBEDTLS_PSA_CRYPTO_C	Define
MBEDTLS_PSA_CRYPTO_SE_C	Undefine
MBEDTLS_THREADING_C	Undefine
MBEDTLS_TIMING_C	Undefine
MBEDTLS_VERSION_C	Define
MBEDTLS_MEMORY_ALIGN_MULTIPLE	Undefine

Figure 30. Configure the MbedTLS (Crypto Only) Module

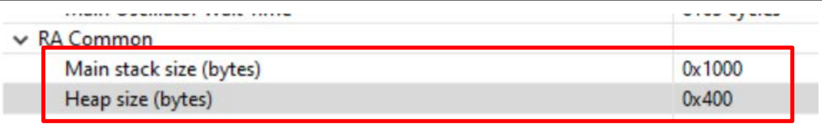
5. Disable RSA to save some memory usage.


MbedTLS (Crypto Only)
BSP Clocks Pins Interrupts Event Links Stacks Components
Smart Browser Properties Pin Conflicts
LS (Crypto Only)

Property	Value
Public Key Cryptography (PKC)	
DHM	
ECC	
RSA	
MBEDTLS_PK_RSA_ALT_SUPPORT	Undefine
MBEDTLS_RSA_NO_CRT	Define
MBEDTLS_RSA_C	Undefine

Figure 31. Disable RSA

6. Set up the Stack and Heap used by the bootloader based on the authentication mode. Set the following values in the BSP tab:



Property	Value
RA Common	
Main stack size (bytes)	0x1000
Heap size (bytes)	0x400

Figure 32. Configure the BSP Stack and Heap Usage

7. Add the Example Production Key module.

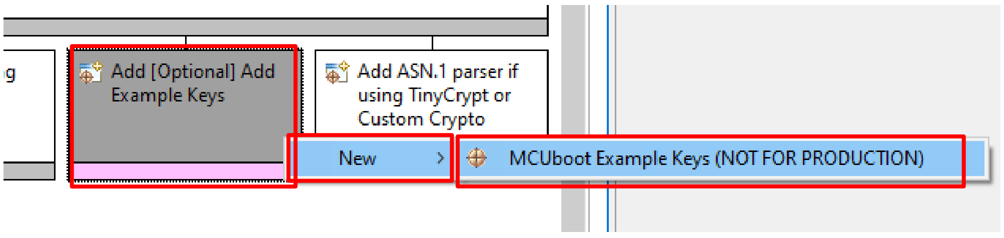
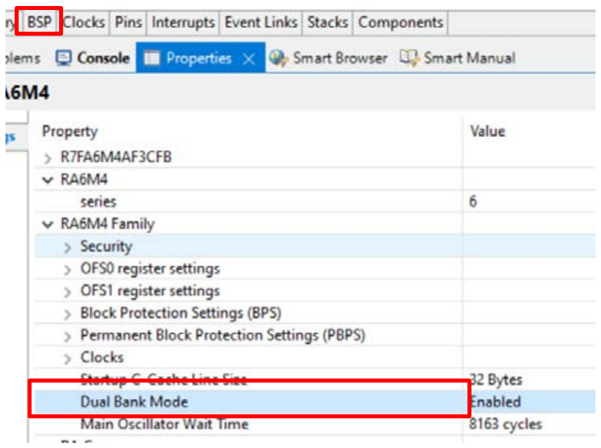

Add [Optional] Add Example Keys
Add ASN.1 parser if using TinyCrypt or Custom Crypto
New
MCUboot Example Keys (NOT FOR PRODUCTION)

Figure 33. Add the Example Production Key module

8. Enable the **Dual Bank Mode** under the **BSP** tab.


BSP Clocks Pins Interrupts Event Links Stacks Components
Console Properties Smart Browser Smart Manual
6M4

Property	Value
R7FA6M4AF3CFB	
RA6M4	
series	6
RA6M4 Family	
Security	
OFS0 register settings	
OFS1 register settings	
Block Protection Settings (BPS)	
Permanent Block Protection Settings (PBPS)	
Clocks	
Startup C-Cache Line Size	32 Bytes
Dual Bank Mode	Enabled
Main Oscillator Wait Time	8163 cycles

Figure 34. Enable Flash Dual Bank Mode

4.4 Add the Boot Code

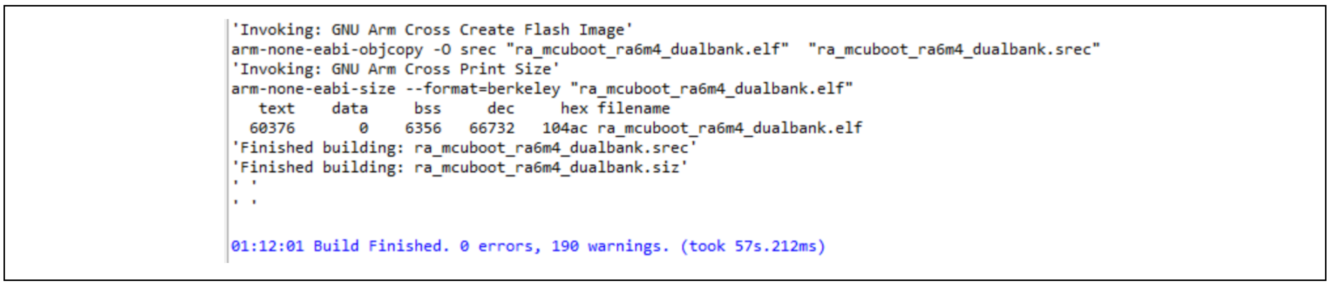
Save `Configuration.xml` and click **Generate Project Content**. Then, expand the `Developer Assistance->HAL/Common->MCUboot->Quick Setup` and drag `Call Quick Setup` to the top of the `hal_entry.c` of the bootloader project.

Add the following function call to the top of the `hal_entry()` function:

```
mcuboot_quick_setup();
```

4.5 Compile the Bootloader Project

In the RA configurator, click **Generate Project Content**, then compile the project.



```
'Invoking: GNU Arm Cross Create Flash Image'
arm-none-eabi-objcopy -O srec "ra_mcuboot_ra6m4_dualbank.elf"  "ra_mcuboot_ra6m4_dualbank.srec"
'Invoking: GNU Arm Cross Print Size'
arm-none-eabi-size --format=berkeley "ra_mcuboot_ra6m4_dualbank.elf"
   text    data     bss     dec     hex filename
  60376       0    6356   66732   104ac ra_mcuboot_ra6m4_dualbank.elf
'Finished building: ra_mcuboot_ra6m4_dualbank.srec'
'Finished building: ra_mcuboot_ra6m4_dualbank.siz'
' '
' '

01:12:01 Build Finished. 0 errors, 190 warnings. (took 57s.212ms)
```

Figure 35. Compile the Bootloader ra_mcuboot_ra6m4_dualbank

There are warnings from third-party code.

4.6 Configure the Python Signing Environment

Signing the application image can be done using a post-build step in e^2 studio, using the image signing tool `Imgtool.py`, which is included with MCUboot. This tool is integrated as a post-build tool in e^2 studio to sign the application image. If this is **NOT** the first time you have used the Python script signing tool on your computer, you can skip to section 5.

If this is the first time you are using the Python script signing tool on your system, you will need to install the dependencies required for the script to work. Navigate to the **ra_mcuboot_ra6m4_dualbank>ra>mcu-tools>MCUboot** folder in the **Project Explorer**, right click and select **Command Prompt**. This will open a command window with the path set to the `\mcu-tools\MCUboot` folder.

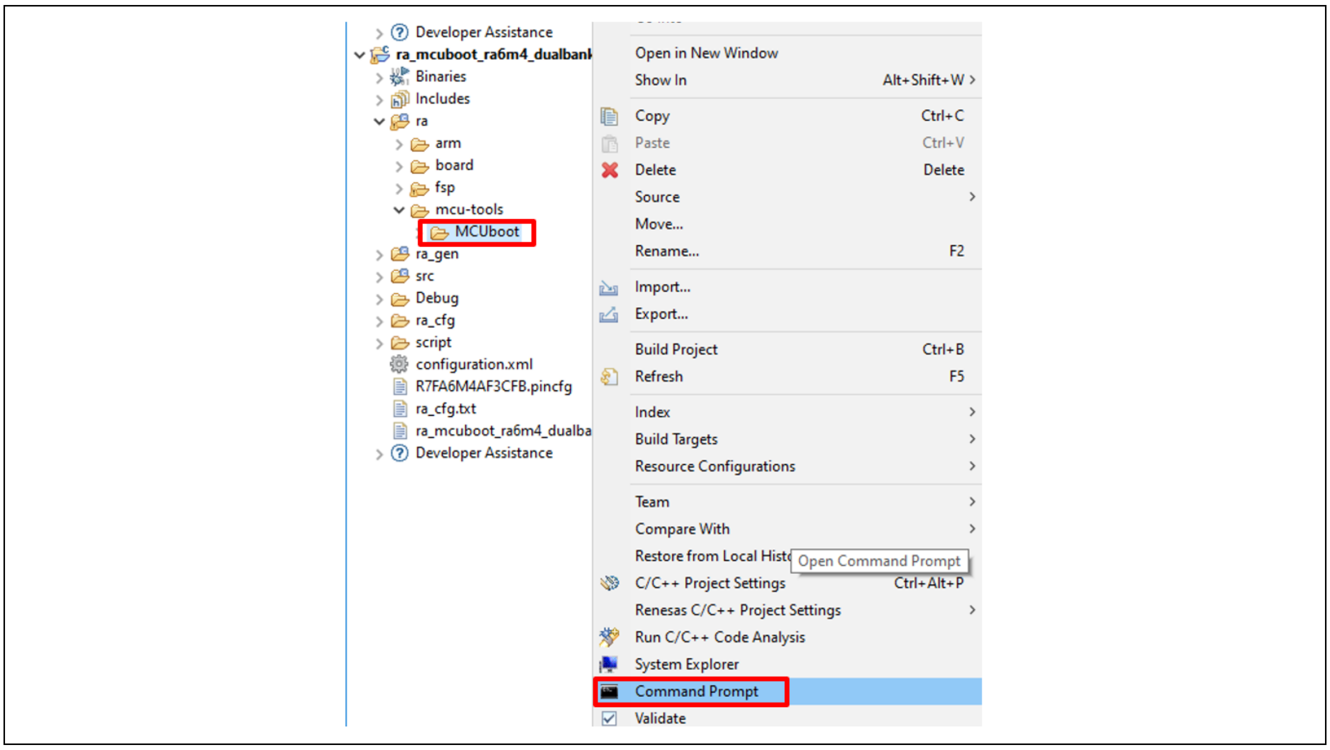


Figure 36. Open the Command Prompt

We recommend upgrading pip prior to installing the dependencies. Enter the following command to update pip:

```
python -m pip install --upgrade pip
```

Next, in the command window, enter the following command line to install all the MCUboot dependencies:

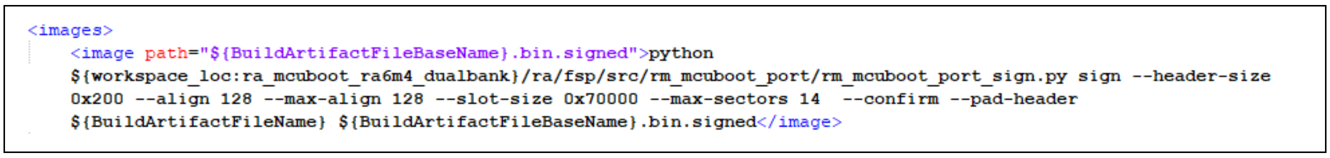
```
pip3 install --user -r scripts/requirements.txt
```

This will verify and install any dependencies that are required.

Review the Signing Command

The signing command for the application image will be automatically generated when the booloader is compiled. In the **Project Explorer**, open the `ra_mcuboot_ra6m4_dualbank\debug\ra_mcuboot_ra6m4_dualbank.bld` file. The signing command is under the section <image>.

The application image uses a **Build Variable** to link with the .bld file. This process is explained in detail in the section 5.1. The application image has access to the .bld file, and the signing command will be automatically executed when the application image is compiled.



```
<images>
    <image path="${BuildArtifactFileBaseName}.bin.signed">python
    ${workspace_loc:ra_mcuboot_ra6m4_dualbank}/ra/fsp/src/rm_mcuboot_port/rm_mcuboot_port_sign.py sign --header-size
    0x200 --align 128 --max-align 128 --slot-size 0x70000 --max-sectors 14  --confirm --pad-header
    ${BuildArtifactFileName} ${BuildArtifactFileBaseName}.bin.signed</image>
```

Figure 37. Signing Command in the .bld File

4.7 Prepare for Production Support

For production support, generate a `.srec` file of the bootloader to be loaded to the upper bank. This can be done by configuring a custom **Builder** within e^2 studio for the bootloader project.

This application project includes a bat file, `process_bootloader.bat`, which runs a script using `srec_cat.exe` to generate a .srec file, `ra_mcuboot_ra6m4_dualbank_offset.srec`, which offsets the bootloader offset to the RA6M4 flash linear mode upper bank address at 0x80000.

Note that for MCUs with different code flash size, the upper bank address needs to be updated accordingly. As explained in sections 1.1 and 1.2, this address is at half of the code flash size.

Since the option-setting memory is located outside of the bank range, this process also truncates the bootloader to the bank size, which is 0x80000.

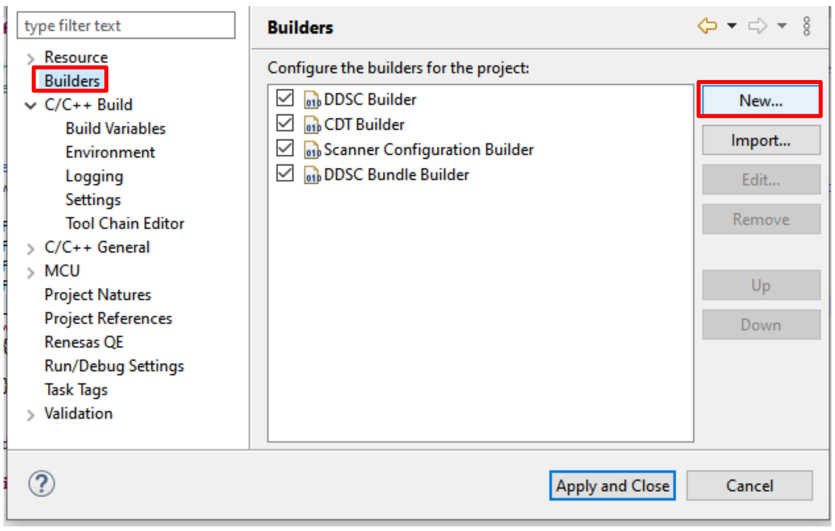
```
srec_cat Debug\ra_mcuboot_ra6m4_dualbank.srec -crop 0 0x80000 -offset 0x80000 -o
ra_mcuboot_ra6m4_dualbank_offset.srec
```

Figure 38. Process the Bootloader to Load to the Upper Bank: process_bootloader.bat

Follow the steps below to configure the custom **Builder** in the bootloader project just created:

1. Unzip `RA6_Secure_Bootloader_Dualbank.zip` and copy `\ra_mcuboot_ra6m4_dualbank\process_bootloader.bat` as well as `srec_cat.exe`, located in the same folder, to the project root folder of the bootloader project just created.

2. Right-click on the bootloader project, open the **Properties** page, and navigate to **Builders** page**.** Click **New** to start creating the customized Builder.

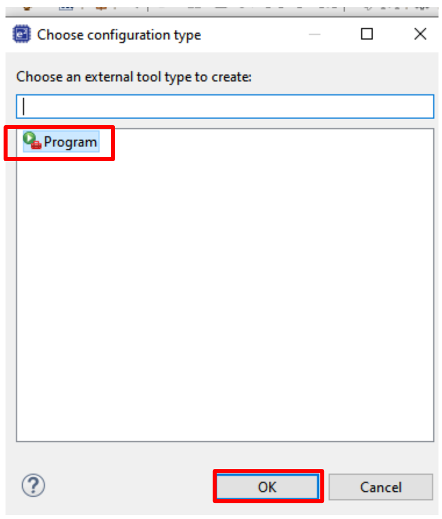


Configure the builders for the project:

- ☑ DDSC Builder
- ☑ CDT Builder
- ☑ Scanner Configuration Builder
- ☑ DDSC Bundle Builder

Figure 39. Create a New Custom Builder Entry

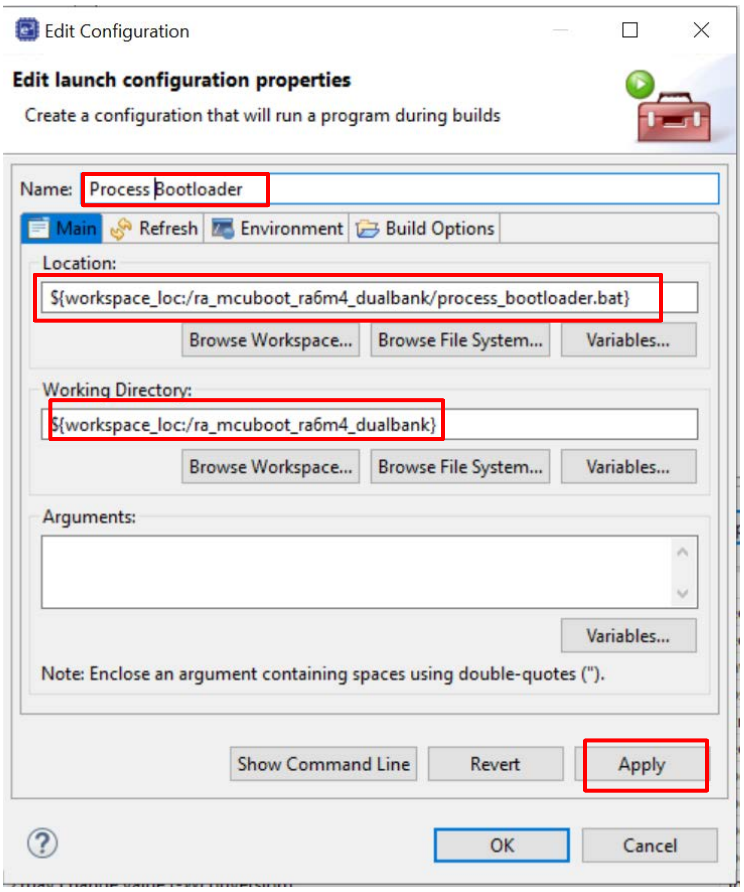
3. Select **Program** in the next screen, then click **OK**:



Choose an external tool type to create:

Figure 40. Select the Type of the Builder as Program

4. Next, provide the new **Builder** name **Process Bootloader** and click **Browse Workspace** to select `process_bootloader.bat` file as the **Location** of the Builder. Also, click **Browse Workspace** to set the **Working Directory** as shown below. Then, click **Apply**.


Edit Configuration

Edit launch configuration properties

Create a configuration that will run a program during builds

Name: Process Bootloader

Location: ${workspace_loc:/ra_mcuboot_ra6m4_dualbank/process_bootloader.bat}

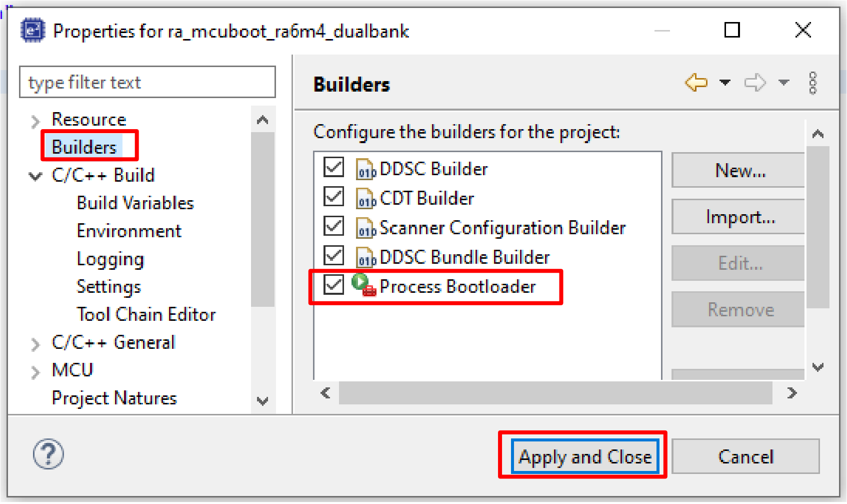
Working Directory: ${workspace_loc:/ra_mcuboot_ra6m4_dualbank}

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").

Figure 41. Configure the Custom Builder

5. Click **OK**, then **Apply and Close** at the next screen.


Properties for ra_mcuboot_ra6m4_dualbank

Builders

Configure the builders for the project:

- ☑ DDSC Builder
- ☑ CDT Builder
- ☑ Scanner Configuration Builder
- ☑ DDSC Bundle Builder
- ☑ Process Bootloader

Figure 42. Custom Builder

6. Recompile the bootloader project and notice that `ra_mcuboot_ra6m4_dualbank_offset.srec` is created under the bootloader project root directory.

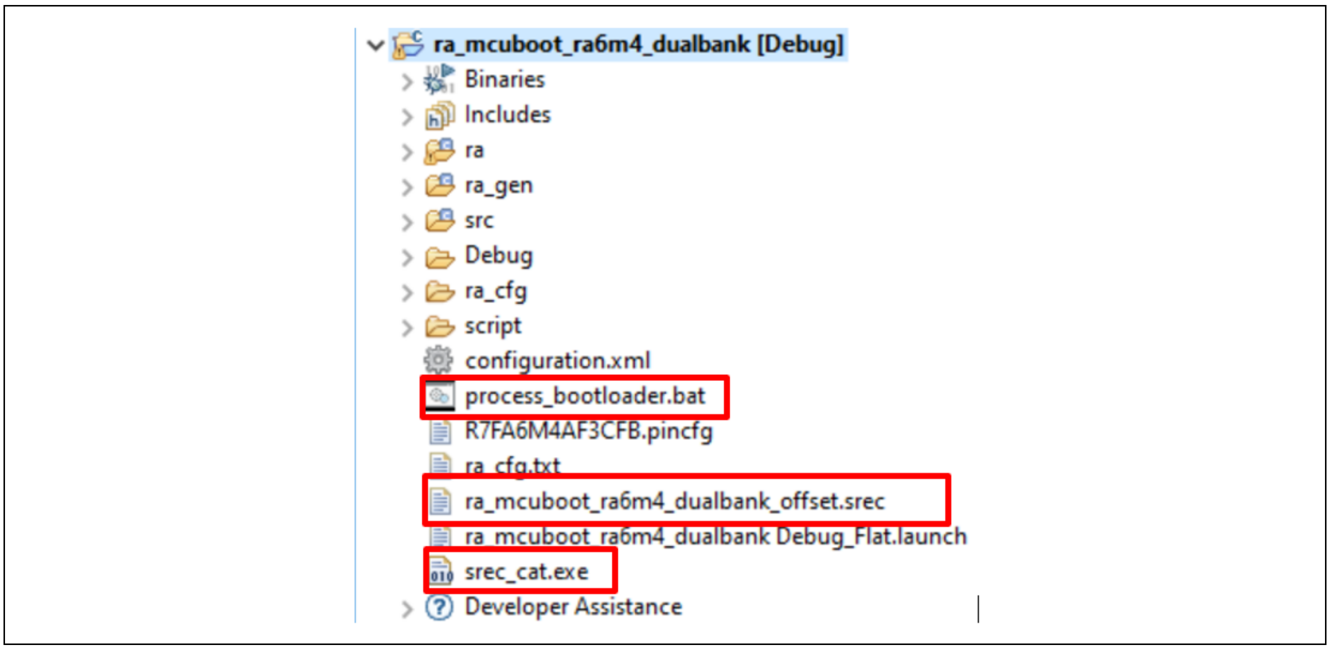


Figure 43. Rebuild the Bootloader with the Custom Builder

5. Configuring and Signing an Application Project

Developing an initial application to use a bootloader starts with developing and testing the application and the bootloader independently. Using the bootloader with an existing application or developing a new application to use the bootloader involves the following common steps:

- Adjust the memory map of the bootloader to allow the application and bootloader to fit the available MCU memory area.
- Configure the application to use the bootloader.
- Sign the application image.
- Developing an application to use a bootloader typically requires the application to have the capability to download a new application. This application project demonstrates how to download a new application using the USB and UART interfaces as examples. Users typically have custom methods to download new application images.

5.1 Configure the Application Project to Use the Bootloader

Users can follow *FSP User's Manual* section Tutorial: Your First RA MCU Project – Blinky to establish a new project. This application note uses the included example project as the initial application project and guides the user through the procedures to configure the example project to use the bootloader established in section 4.

Note that the steps described in this section can be applied to other existing application projects to configure the application project to use the bootloader. Be sure to consider the size the application project. When using the bootloader with a different application project, the **Image 1 Flash Area Size** property should be adjusted accordingly.

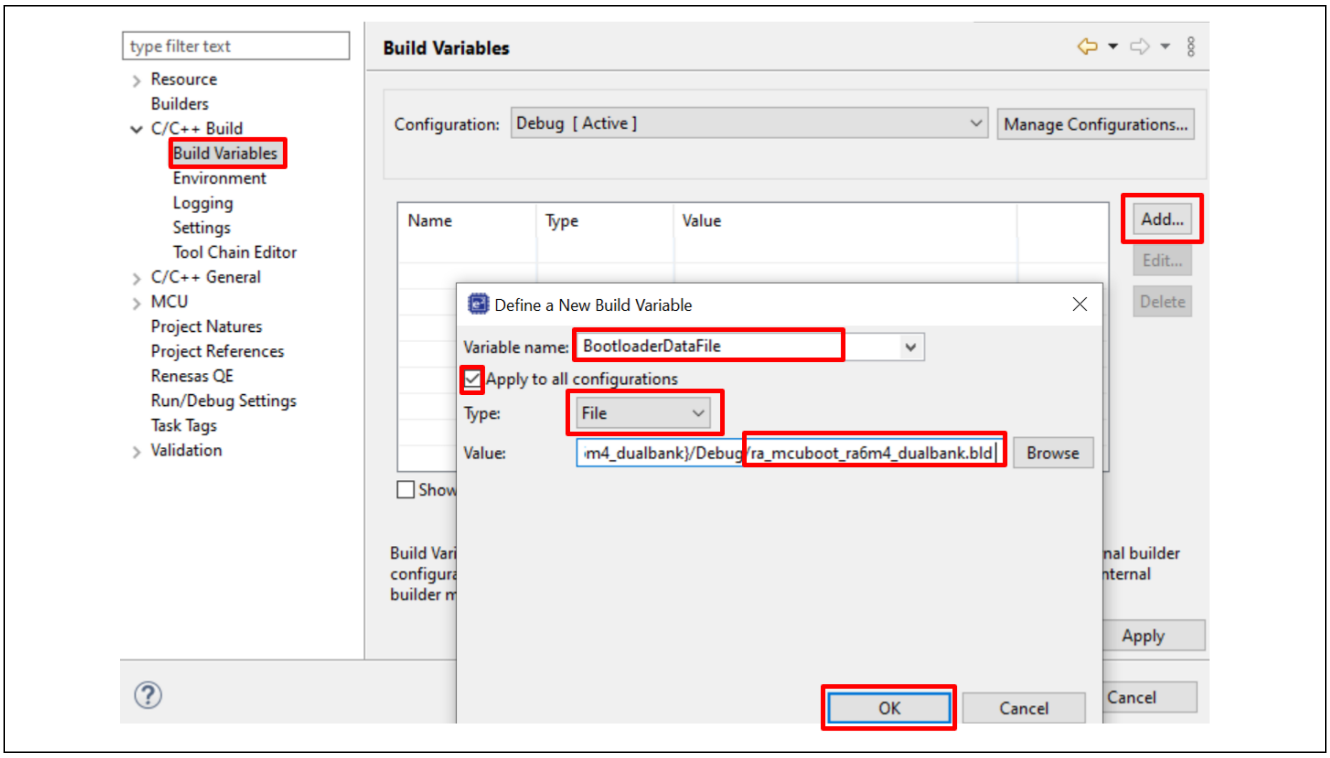
Import the desired application projects under folder `\example_projects_without_bootloader` to the workspace where the bootloader is created. For example, if the intended firmware update channel is USB, import `app_primary_usb` into the workspace.

Note: In this section's illustrations, the USB interface is used. The procedure for using the UART interface is similar to using USB.

Right-click on the application project folder `app_primary_usb` in the **Project Explorer** and select **Properties**. Select **C/C++ Build > Build Variables**, click **Add** and set the **Variable name** to

BootloaderDataFile, and check the **Apply to all configurations** box. Change the **Type** to **File** and enter the path to the `.bld` file for the bootloader project **`ra_mcuboot_ra6m4_dualbank`**:

- Set `${workspace_loc:ra_mcuboot_ra6m4_dualbank}/Debug/ra_mcuboot_ra6m4_dualbank.bld` for the value.


Define a New Build Variable

Variable name: BootloaderDataFile

[✓] Apply to all configurations

Type: File

Value: m4_dualbank}/Debug/ra_mcuboot_ra6m4_dualbank.bld

Figure 44. Configure the Build Variable to Use the Bootloader

Click **OK**, then **Apply** and **Apply and Close** in the next screen.

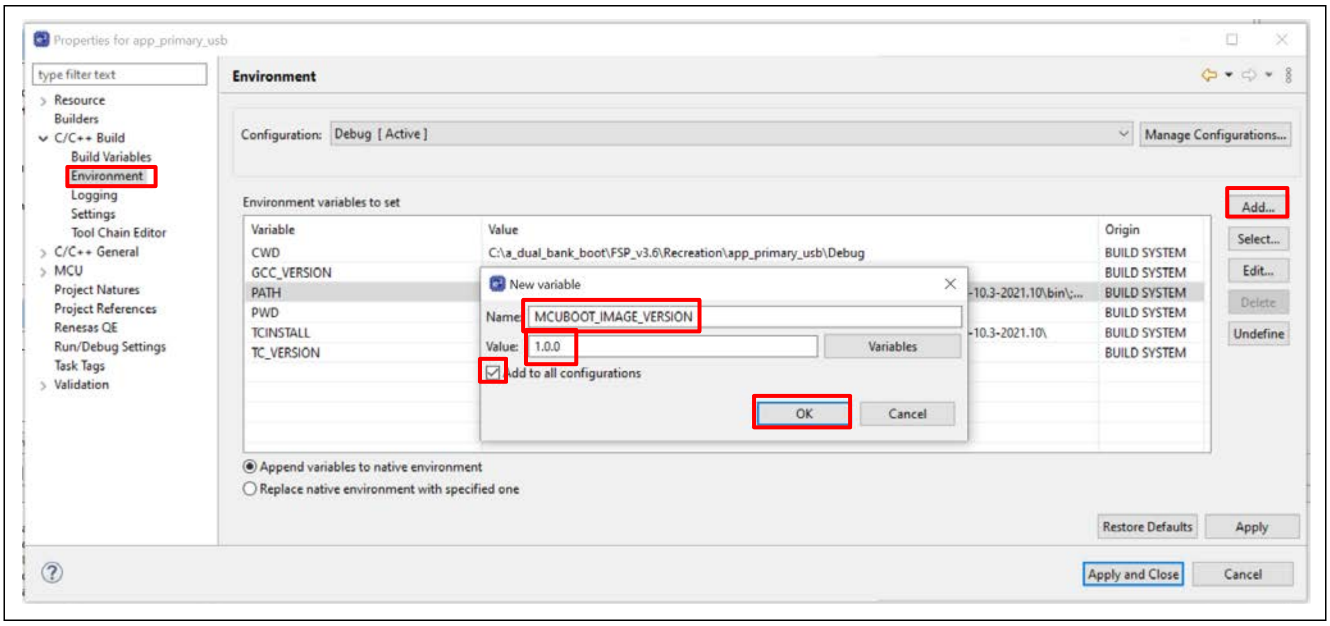
5.2 Signing the Application Image

Note: If you rebuild the bootloader project after changing any of the signing and signature **Properties** of the MCUboot module, you will need to **Generate Project Content** again to bring in the updated `.bld` file.

When using Direct XIP mode, each application can define a version number. This is achieved by defining an Environment Variable: MCUBOOT_IMAGE_VERSION.

For applications that support signature verification, the signing key can be configured using Environment Variable MCUBOOT_IMAGE_SIGNING_KEY. If there is no signature verification, then it is not necessary to set Environment Variable MCUBOOT_IMAGE_SIGNING_KEY.

Open the **Properties** page of the project `app_primary_usb`, under **Environment**, click **Add** and configure MCUBOOT_IMAGE_VERSION.



Environment

Configuration: Debug [Active] Manage Configurations...

Environment variables to set

Variable	Value	Origin
CWD	C:\a_dual_bank_boot\FSP_v3.6\Recreation\app_primary_usb\Debug	BUILD SYSTEM
GCC_VERSION		BUILD SYSTEM
PATH	-10.3-2021.10\bin\;...	BUILD SYSTEM
PWD		BUILD SYSTEM
TCINSTALL	-10.3-2021.10\	BUILD SYSTEM
TC_VERSION		BUILD SYSTEM

New variable

Name: MCUBOOT_IMAGE_VERSION

Value: 1.0.0

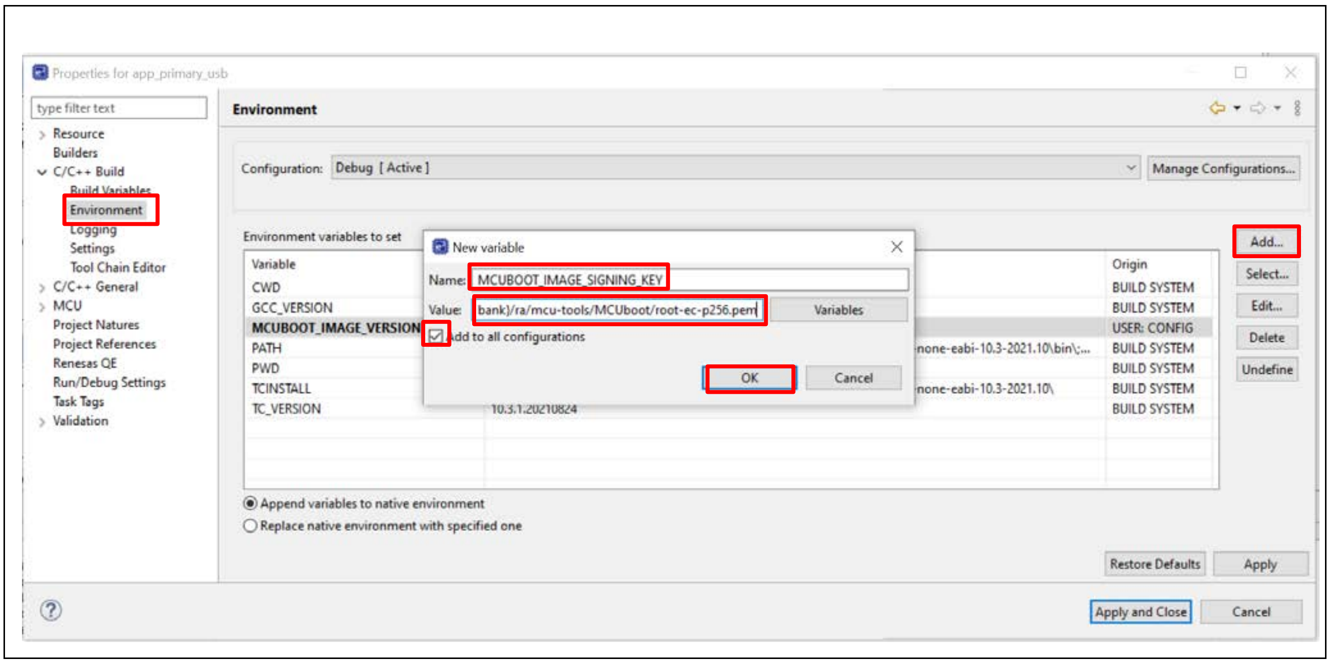
[✓] Add to all configurations

(●) Append variables to native environment

( ) Replace native environment with specified one

Figure 45. Configure the Application Version

Similarly, add the new variable for MCUBOOT_IMAGE_SIGNING_KEY.



Environment

Configuration: Debug [Active] Manage Configurations...

Environment variables to set

Variable	Value	Origin
CWD		BUILD SYSTEM
GCC_VERSION		BUILD SYSTEM
MCUBOOT_IMAGE_VERSION		USER: CONFIG
PATH	none-eabi-10.3-2021.10\bin\;...	BUILD SYSTEM
PWD		BUILD SYSTEM
TCINSTALL	none-eabi-10.3-2021.10\	BUILD SYSTEM
TC_VERSION	10.3.1.20210824	BUILD SYSTEM

New variable

Name: MCUBOOT_IMAGE_SIGNING_KEY

Value: bank}/ra/mcu-tools/MCUboot/root-ec-p256.pem

[✓] Add to all configurations

(●) Append variables to native environment

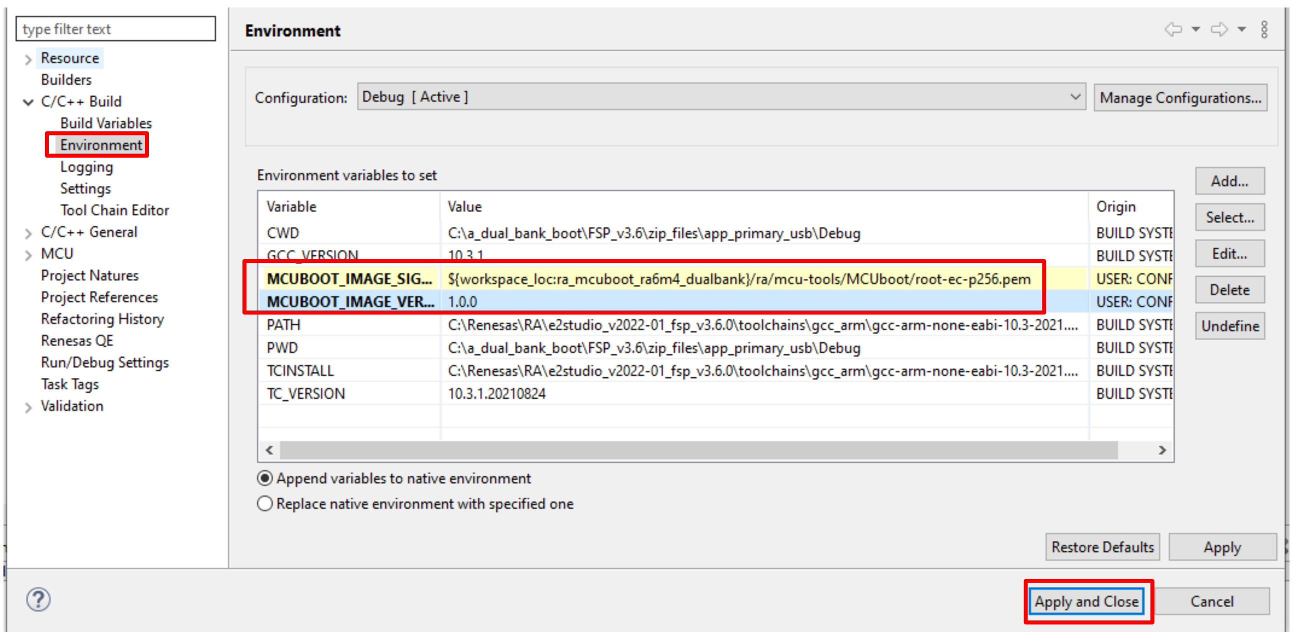
( ) Replace native environment with specified one

Figure 46. Configure the Private Signing Key

Note that the private key used for signing the application image is indicated in the signing command.

`/ra/mcu-tools/MCUboot/root-ec-p256.pem` is used for the example bootloader. This key is used for testing purpose only. For real world use case and production support, users MUST change this to the private key of their choice.

Figure 47 is the result of the above configuration. Click **Apply and Close**.



Configuration: Debug [Active]

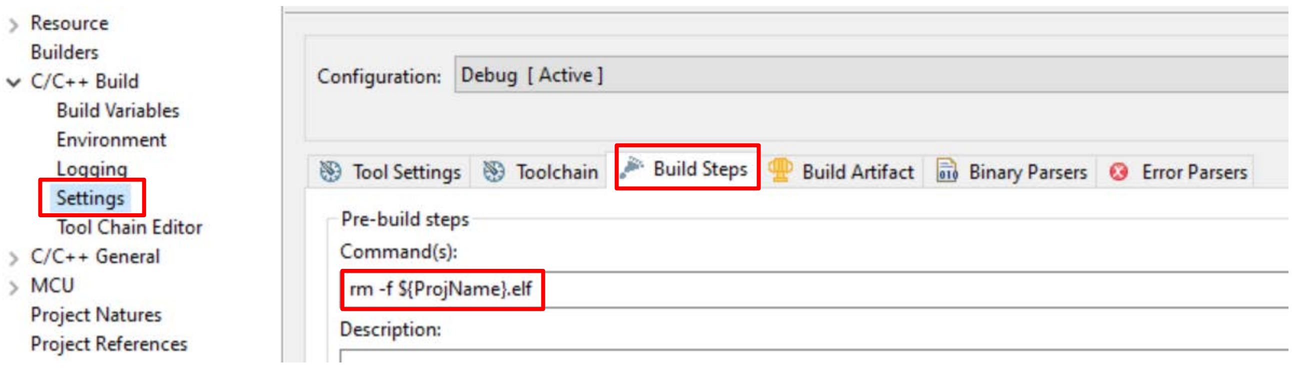
Environment variables to set

Variable	Value	Origin
CWD	C:\a_dual_bank_boot\FSP_v3.6\zip_files\app_primary_usb\Debug	BUILD SYSTE
GCC_VERSION	10.3.1	BUILD SYSTE
MCUBOOT_IMAGE_SIG...	${workspace_loc:ra_mcuboot_ra6m4_dualbank}/ra/mcu-tools/MCUboot/root-ec-p256.pem	USER: CONF
MCUBOOT_IMAGE_VER...	1.0.0	USER: CONF
PATH	C:\Renesas\RA\e2studio_v2022-01_fsp_v3.6.0\toolchains\gcc_arm\gcc-arm-none-eabi-10.3-2021....	BUILD SYSTE
PWD	C:\a_dual_bank_boot\FSP_v3.6\zip_files\app_primary_usb\Debug	BUILD SYSTE
TCINSTALL	C:\Renesas\RA\e2studio_v2022-01_fsp_v3.6.0\toolchains\gcc_arm\gcc-arm-none-eabi-10.3-2021....	BUILD SYSTE
TC_VERSION	10.3.1.20210824	BUILD SYSTE

[✓ Append variables to native environment]
Replace native environment with specified one

Figure 47. Configure the Application Image version number and Signing Key

To be able to recompile the project whenever the Environment Variables are updated, it is recommended add a Pre-build step to always delete the `.elf` file, as shown in Figure 48, so the application project is always recompiled.



Configuration: Debug [Active]

Pre-build steps

Command(s): rm -f ${ProjName}.elf

Description:

Figure 48. Configure the Pre-build Command

At this point, a user can click **Generate Project Content** and compile the newly created application project and ensure that `\Debug\app_primary_usb.bin.signed` is generated.

5.3 Preparation for Production Support

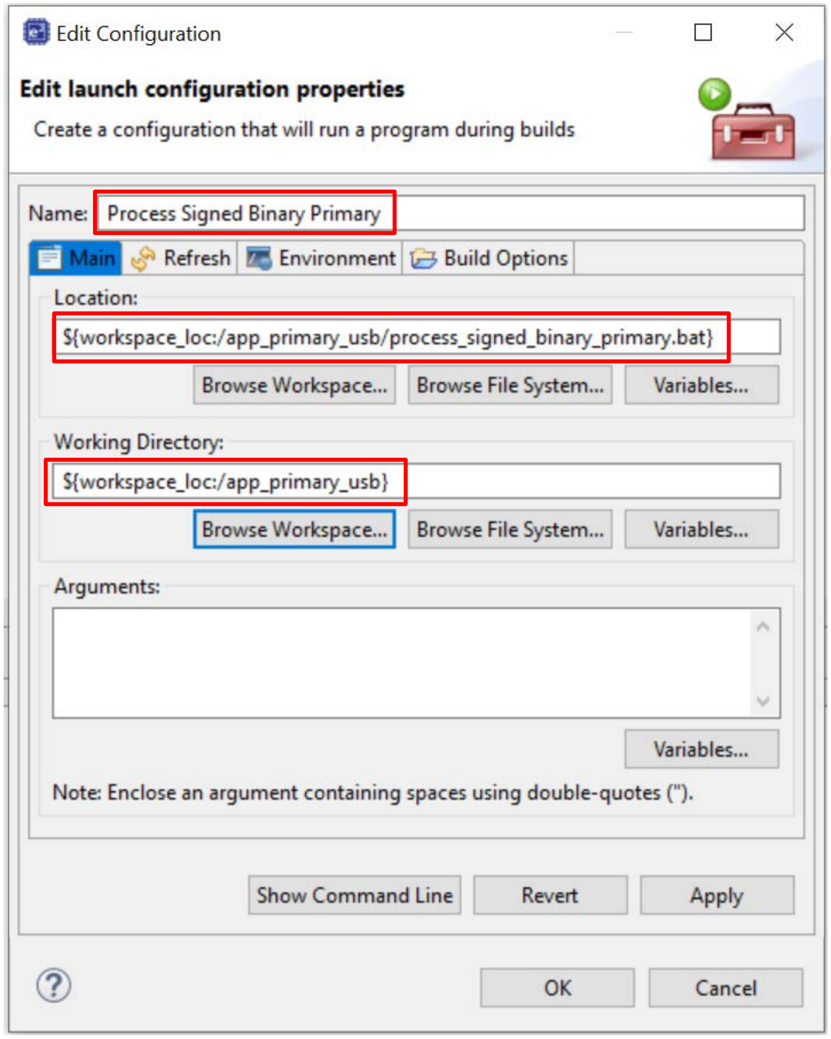
For production support, a `.srec` file based on the signed application image needs to be generated. This `.srec` file offsets the application to the start address of the primary application, 0x10000 based on Figure 25.

```
srec_cat Debug\app_primary_usb.bin.signed -binary -offset 0x10000 -o
app_primary_usb_singed_offset.srec
```

Figure 49. Create app_primary_usb_signed_offset.srec

Follow steps similar to section 4.7 to add the custom **Builder** and compile the primary application:

1. Copy `\example_projects_with_bootloader\app_primary_usb\srec_cat.exe` and `process_signed_binary_primary.bat` to the root of project `app_primary_usb`.
2. Follow section 4.7 to create the new **Builder**. The finished configuration should look like Figure 50.


Edit Configuration

Edit launch configuration properties

Create a configuration that will run a program during builds

Name: Process Signed Binary Primary

Location:
${workspace_loc:/app_primary_usb/process_signed_binary_primary.bat}

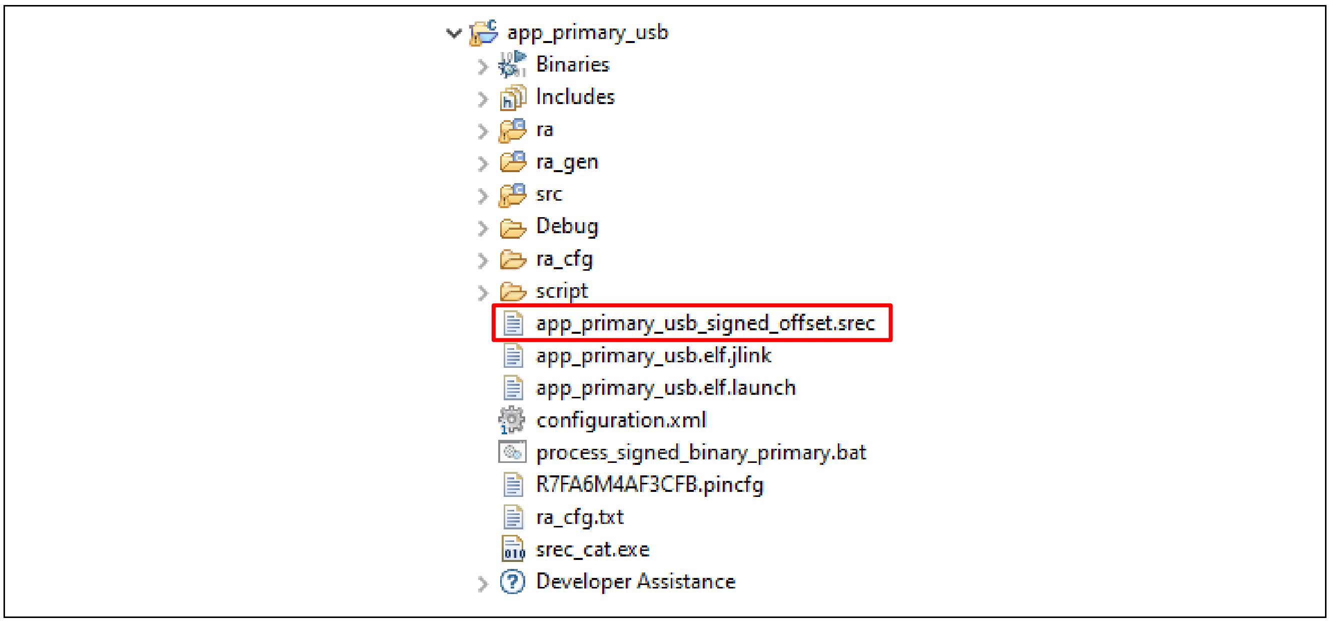
Working Directory:
${workspace_loc:/app_primary_usb}

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").

Figure 50. Configure the Custom Builder for the Primary Application

3. Click **Generate Project Content** and compile the `app_primary_usb` project. Ensure that `app_primary_usb_signed_offset.srec` is generated under the root of the `app_primary_usb` project.



```
app_primary_usb
  Binaries
  Includes
  ra
  ra_gen
  src
  Debug
  ra_cfg
  script
  app_primary_usb_signed_offset.srec
  app_primary_usb.elf.jlink
  app_primary_usb.elf.launch
  configuration.xml
  process_signed_binary_primary.bat
  R7FA6M4AF3CFB.pincfg
  ra_cfg.txt
  srec_cat.exe
  Developer Assistance
```

Figure 51. Signed Primary Image Offset to the Primary Slot

6. Booting the Primary Application and Updating to a New Image

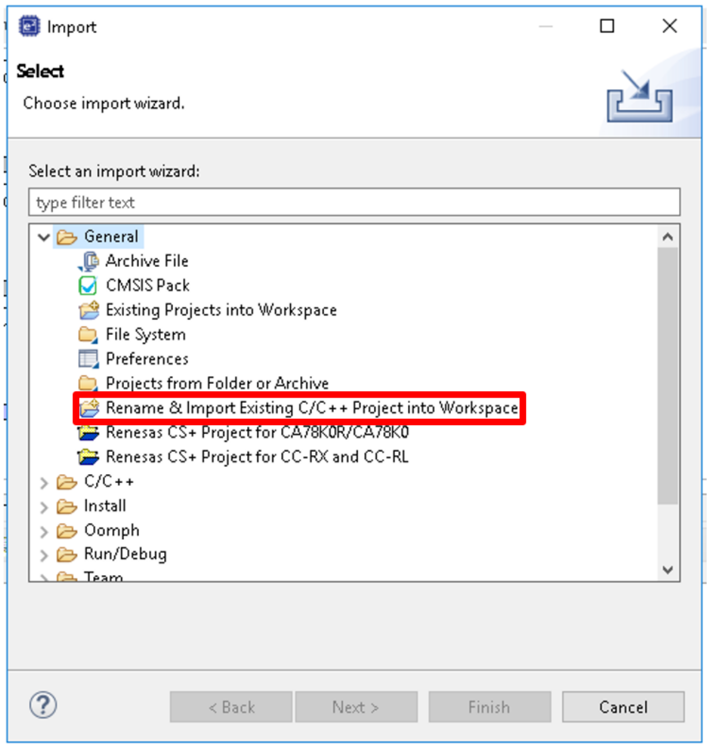
To update the application, the primary application needs to provide an image downloader. A new image will also need to be prepared to test the image downloader function.

6.1 Prepare a Secondary Image

In this project, a secondary image is created to test the downloading functionality of the primary application. The new application can be created by either modifying the existing application or creating a new application project. If a new application project is used, the user needs to establish the linkage to the bootloader by following section 5. The newly created application project must also provide a method to download the new application to the upper bank.

In this application project, we will import the initial application project to the same workspace, rename the new project, and perform minor updates.

Right-click in the white space in the **Project Explorer** area and select **Import** and choose **Rename & Import Existing C/C++ Project into Workspace**.


Import

Select

Choose import wizard.

Select an import wizard:

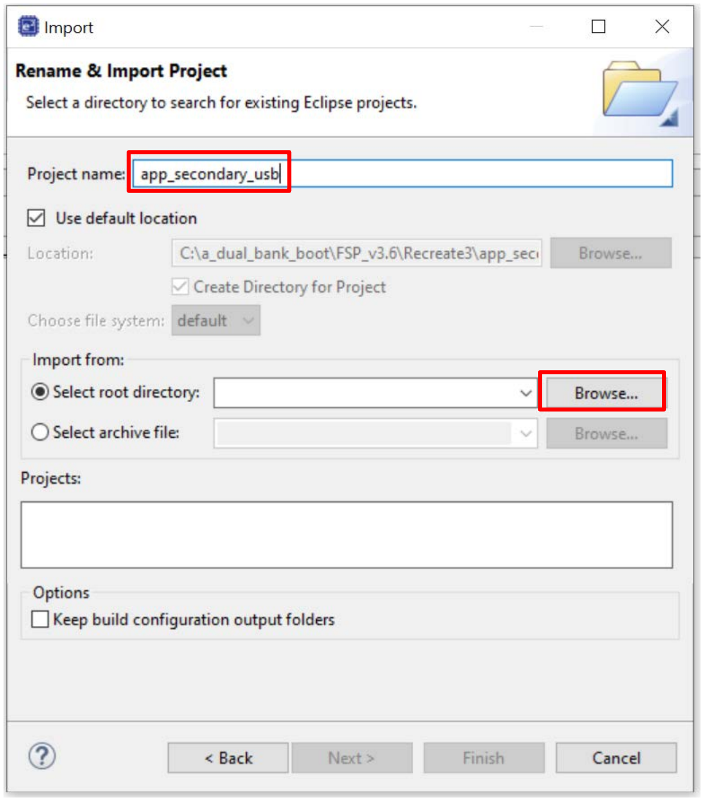
type filter text

- General
  - Archive File
  - CMSIS Pack
  - Existing Projects into Workspace
  - File System
  - Preferences
  - Projects from Folder or Archive
  - Rename & Import Existing C/C++ Project into Workspace
  - Renesas CS+ Project for CA78K0R/CA78K0
  - Renesas CS+ Project for CC-RX and CC-RL
- C/C++
- Install
- Oomph
- Run/Debug
- Team

< Back | Next > | Finish | Cancel

Figure 52. Import the Initial Application

Once the **Import** window opens, name the project `app_secondary_usb`, check **Select root directory**, and click **Browse**:


Import

Rename & Import Project

Select a directory to search for existing Eclipse projects.

Project name: app_secondary_usb

☑ Use default location

Location: C:\a_dual_bank_boot\FSP_v3.6\Recreate3\app_seco Browse...

☑ Create Directory for Project

Choose file system: default

Import from:

● Select root directory: Browse...

○ Select archive file: Browse...

Projects:

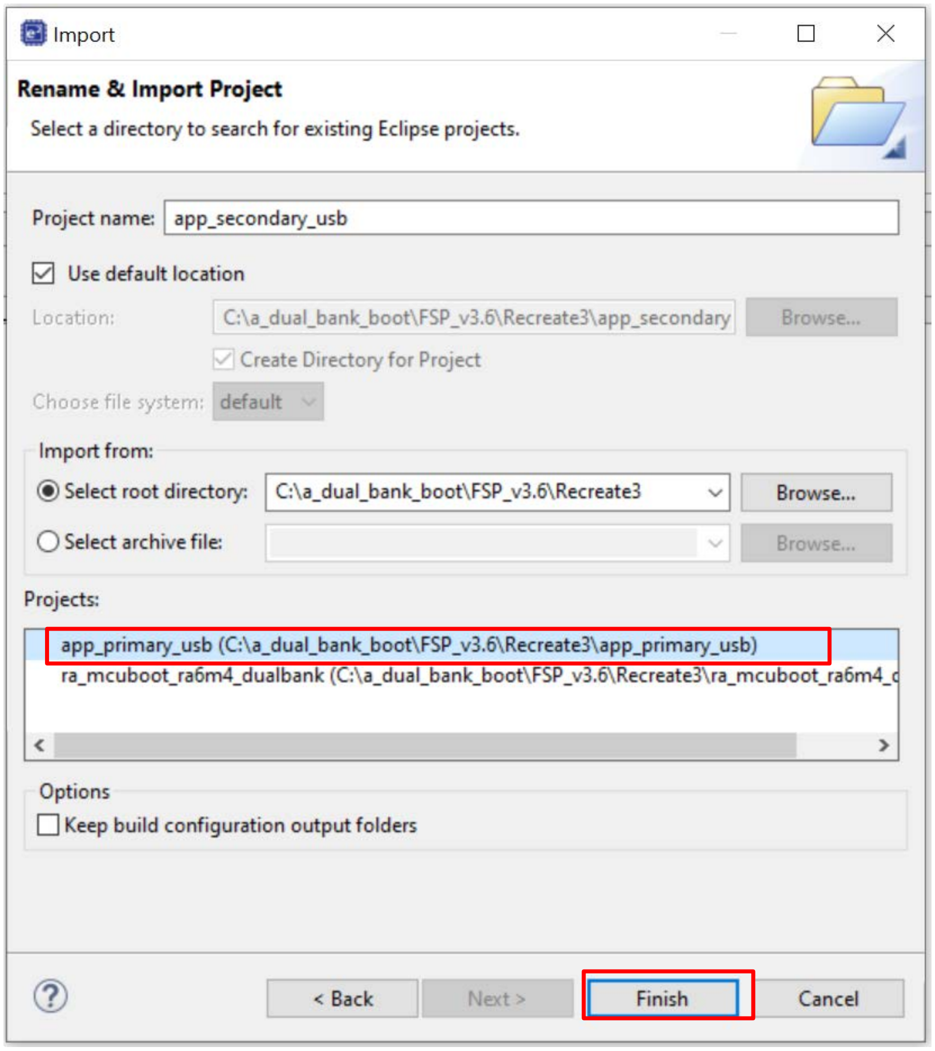
Options

☐ Keep build configuration output folders

< Back | Next > | Finish | Cancel

Figure 53. Name the New Application

Browse into the Workspace folder and select `app_primary_usb`.



Select a directory to search for existing Eclipse projects.

Project name: app_secondary_usb

☑ Use default location

Location: C:\a_dual_bank_boot\FSP_v3.6\Recreate3\app_secondary

☑ Create Directory for Project

Choose file system: default

Import from:

◉ Select root directory: C:\a_dual_bank_boot\FSP_v3.6\Recreate3

○ Select archive file:

Projects:

app_primary_usb (C:\a_dual_bank_boot\FSP_v3.6\Recreate3\app_primary_usb)

ra_mcuboot_ra6m4_dualbank (C:\a_dual_bank_boot\FSP_v3.6\Recreate3\ra_mcuboot_ra6m4_d

Options

☐ Keep build configuration output folders

Figure 54. Select to Initial Primary Application

Click **Finish**. The new application project will be created with the following attributes:

- When importing the primary application, the **Build Variable** and **Environment Variables** are automatically imported.
- The custom Builder “Process Signed Binary Primary” is also imported. For a clean project, a user must manually remove this Builder and the corresponding support files from the secondary project.
- Unlike in normal XIP Mode operation, the linker script symbol **XIP_SECONDARY_SLOT_IMAGE** must be undefined in Dual Bank mode. By default, **XIP_SECONDARY_SLOT_IMAGE** is undefined in the linker script symbol, so no action needs to be taken here.

Update Existing Application to a New Application

To demonstrate the application update, update the application to blink the blue and green LED only.

Perform the following code updates in `blinky_thread_entry.c`:

Change below section of code in blinky_thread_entry:

```
        /* Update all board LEDs */
         for (uint32_t i = 0; i < leds.led_count; i++)
         {
             /* Get pin to toggle */
             uint32_t pin = leds.p_leds[i];

             /* Write to this pin */
             R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
        }
```

To:

```
        /* update the blue led */
         R_BSP_PinWrite(leds.p_leds[0], pin_level);

         /* update the green led */
         R_BSP_PinWrite(leds.p_leds[1], pin_level);
```

Figure 55. Update the LED Control

Save the updated source file, click **Generate Project Content**, then compile the new project**.**

If you create a new application project and would like to debug the new project with the bootloader, follow the instructions in section 6.3 to section 6.7. When debugging an update image with the bootloader, you can treat the update image as the primary application.

6.2 Set Up the Hardware

If using `app_primary_usb` as the initial application project:

- Connect J10 (USB Debug) using a USB micro to B cable from the EK-RA6M4 to the development PC to provide power and debug connection using the on-board debugger.
- Connect J11 (USB FS) using a USB micro to B cable from the EK-RA6M4 to the development PC to provide USB Device connection.

If using `app_primary_uart` as the initial application project:

- Connect J10 using a USB micro to B cable from the EK-RA6M4 to the development PC to provide power and debug connection using the on-board debugger.
- Connect the three pins in Table 3 on the UART to USB converter to the EK-RA6M4.

Table 3. Connection through the UART Interface

UART to USB Converter	RA6M4
RX	P101 (TX)
TX	P100 (RX)
GND	GND

6.3 Erase the MCU

When MCUboot is used in flash dual bank mode, the code flash mode needs to start in linear mode. Erasing the MCU Option-Setting Memory settings will configure the code flash mode to linear mode. Erasing the entire MCU memory is recommended. The MCU can be erased through a variety of methods. A user can erase the MCU flash using the Renesas Device Partition Manager, Renesas Flash Programmer, or third-party tools like JFlash Lite.

Note: If the MCU is in code flash dual bank mode, make sure to restore to linear mode prior to proceeding to the rest of the application note sections. The rest of the operations assume the device starts in code flash linear mode. They will not work if the device is already in code flash dual bank mode.

6.3.1 Use the Renesas Flash Programmer

The Renesas Flash Programmer (RFP) can detect the flash mode when a new RFP project is created.

Note: Prior to connecting with the RFP, power cycle the development board.

Connect the EK-RA6M4 to the PC through J10 USB Debug. Launch RFP and create a new RFP project. Click **File** -> **New Project**.

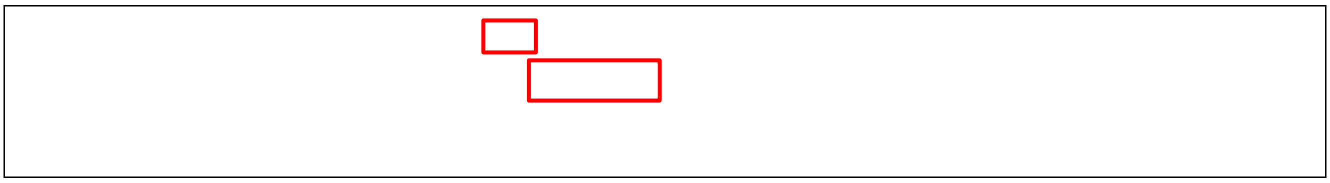
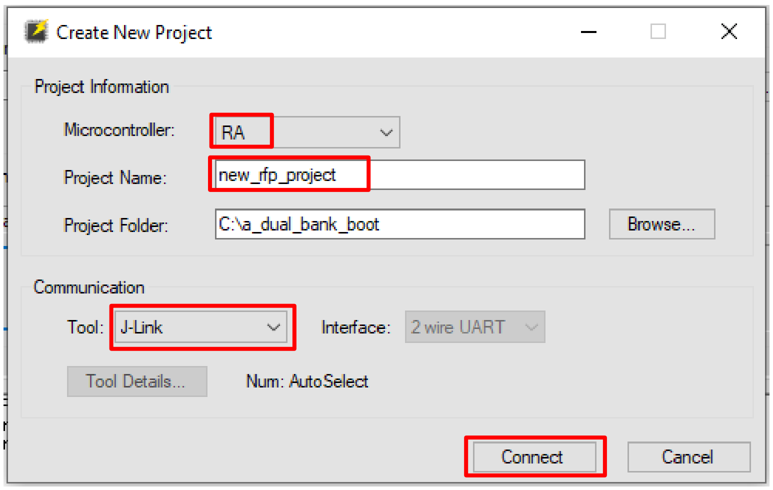


Figure 56. Create a New RFP Project

Configure the **Microcontroller** selection as well as the **Tool** used for communication. Then, click **Connect**.



Create New Project

Project Information

Microcontroller: RA

Project Name: new_rfp_project

Project Folder: C:\a_dual_bank_boot

Communication

Tool: J-Link

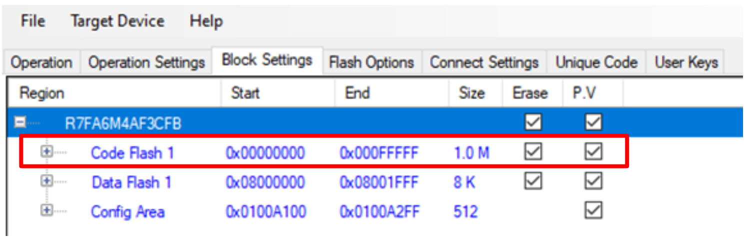
Interface: 2 wire UART

Num: AutoSelect

Figure 57. Configure the New Project

Once the connection is successfully established, the user can open the **Block Settings** page to check the Code Flash configurations.

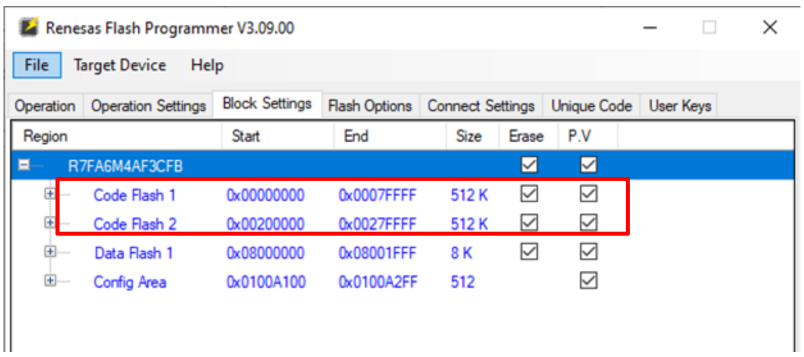
If RA6M4 flash is in code flash linear mode, **Blocks Settings** are presented as in Figure 58.



Region	Start	End	Size	Erase	P.V
R7FA6M4AF3CFB				☑	☑
Code Flash 1	0x00000000	0x000FFFFF	1.0 M	☑	☑
Data Flash 1	0x08000000	0x08001FFF	8 K	☑	☑
Config Area	0x0100A100	0x0100A2FF	512		☑

Figure 58. Flash in Linear Mode

If the RA6M4 flash is in flash dual bank mode, **Block Settings** are presented as in Figure 59.



Region	Start	End	Size	Erase	P.V
R7FA6M4AF3CFB				☑	☑
Code Flash 1	0x00000000	0x0007FFFF	512 K	☑	☑
Code Flash 2	0x00200000	0x0027FFFF	512 K	☑	☑
Data Flash 1	0x08000000	0x08001FFF	8 K	☑	☑
Config Area	0x0100A100	0x0100A2FF	512		☑

Figure 59. Flash in Dual Bank Mode

Whether the MCU is in flash dual bank mode or flash linear mode, the **Initialize Device** command can erase the entire flash, including the Config Area, and thus return the MCU to code flash linear mode.

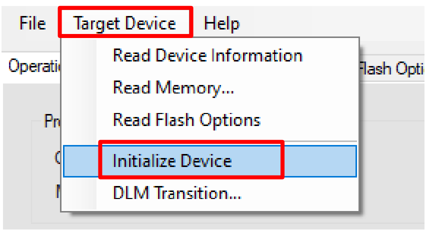
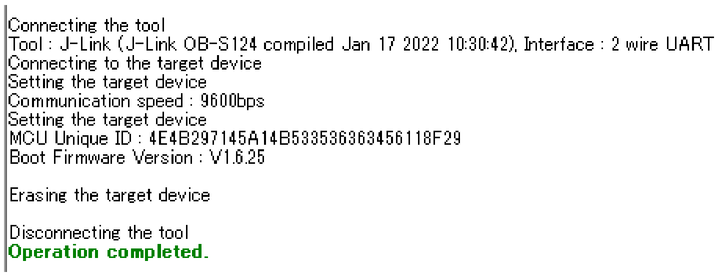


Figure 60. Initialize Device Command

If the **Initialize Device** is successful, the message in Figure 61 will be presented in the status window.



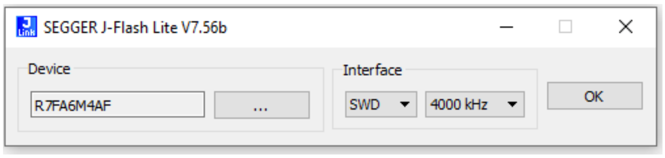
```
Connecting the tool
Tool : J-Link (J-Link OB-S124 compiled Jan 17 2022 10:30:42), Interface : 2 wire UART
Connecting to the target device
Setting the target device
Communication speed : 9600bps
Setting the target device
MCU Unique ID : 4E4B297145A14B533536363456118F29
Boot Firmware Version : V1.6.25

Erasing the target device

Disconnecting the tool
Operation completed.
```

Figure 61. Initialize Device Succeeded

6.3.2 Use the SEGGER J-Flash Lite

J-Flash Lite is a free, simple graphical user interface which allows downloading into flash memory of target systems. J-Flash Lite is part of the J-Link Software and Documentation package that is installed when the J-Link software & documentation pack is installed.

To use J-Flash Lite, connect the USB Debug port J10 to the PC and launch J-Flash Lite. Select the **Device** and debug **Interface** and communication speed.



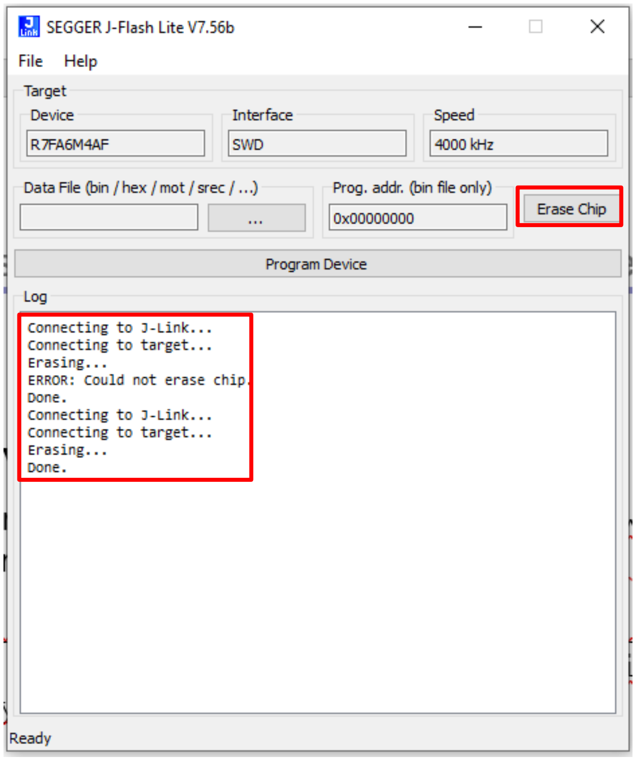
SEGGER J-Flash Lite V7.56b

Device: R7FA6M4AF

Interface: SWD, 4000 kHz

Figure 62. Launch the J-Flash Lite

Click **OK**. In the next screen, select **Erase Chip**.


SEGGER J-Flash Lite V7.56b
File
Help
Target
Device
Interface
Speed
Data File (bin / hex / mot / srec / ...)
...
Prog. addr. (bin file only)
Erase Chip
Program Device
Log
Ready

Device: R7FA6M4AF
Interface: SWD
Speed: 4000 kHz
Prog. addr. (bin file only): 0x00000000

```
Connecting to J-Link...
Connecting to target...
Erasing...
ERROR: Could not erase chip.
Done.
Connecting to J-Link...
Connecting to target...
Erasing...
Done.
```

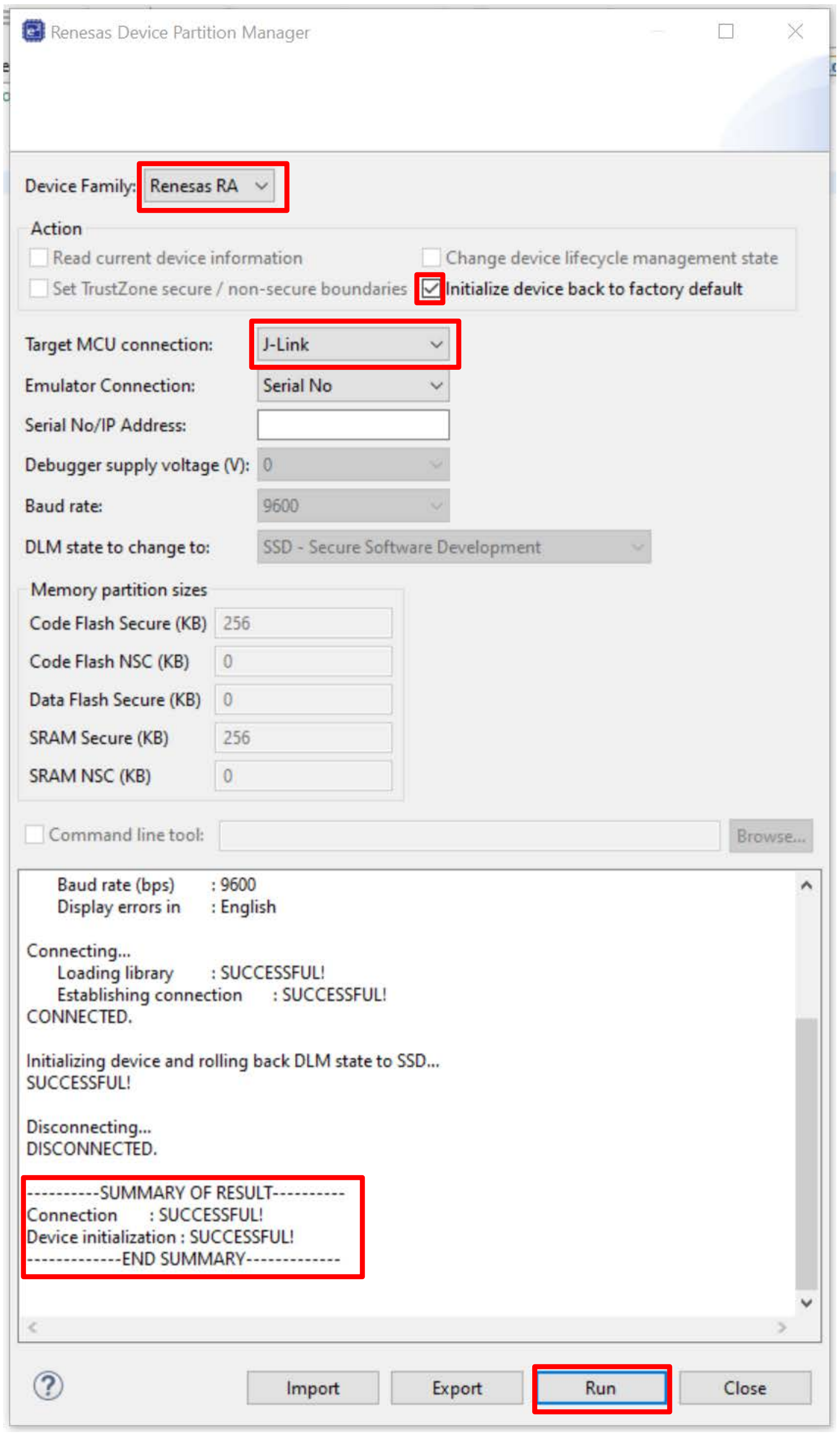
Figure 63. Erase the MCU using J-Flash Lite

Note that when using Segger J-Flash Lite 7.68b or earlier, the Erase operation needs to be performed twice if the device is already in dual bank mode. This may be fixed in later J-Flash Lite versions.

6.3.3 Use Renesas Device Partition Manager

Power cycle the evaluation board EK-RA6M4 after a debug session to use the Renesas Device Partition Manager. Within e^2 studio, navigate to **Run** -> **Renesas Debug Tools** -> **Renesas Device Partition Manager**. Select J-Link as the connection method and select action **Initialize device back to factory default**.

Click **Run**. The MCU will be erased.



Device Family: Renesas RA

- [ ] Read current device information
- [ ] Change device lifecycle management state
- [ ] Set TrustZone secure / non-secure boundaries
- [✓] Initialize device back to factory default

Target MCU connection: J-Link
Emulator Connection: Serial No
Serial No/IP Address:
Debugger supply voltage (V): 0
Baud rate: 9600
DLM state to change to: SSD - Secure Software Development

Code Flash Secure (KB)	256
Code Flash NSC (KB)	0
Data Flash Secure (KB)	0
SRAM Secure (KB)	256
SRAM NSC (KB)	0

- [ ] Command line tool:

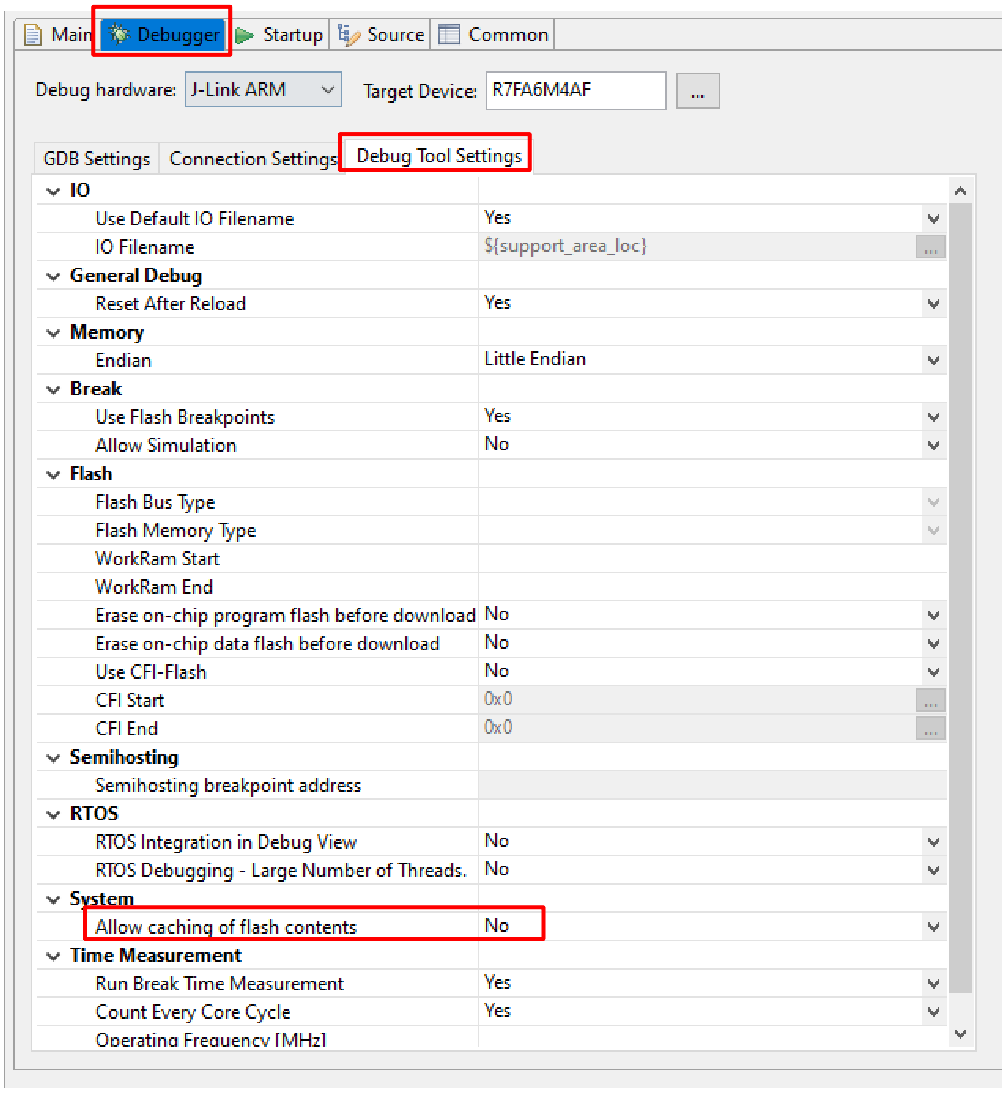
```
    Baud rate (bps)      : 9600
    Display errors in    : English

Connecting...
    Loading library      : SUCCESSFUL!
    Establishing connection     : SUCCESSFUL!
CONNECTED.

Initializing device and rolling back DLM state to SSD...
SUCCESSFUL!

Disconnecting...
DISCONNECTED.

----------SUMMARY OF RESULT----------
Connection      : SUCCESSFUL!
Device initialization : SUCCESSFUL!
-------------END SUMMARY-------------
```

Figure 64. Erase the MCU using Renesas Device Partition Manager

6.4 Start the Debug Session

Follow the steps below to start the debug session:

1. Disable flash content caching from the **Debugger** setting.
   Right-click on project **app_primary_usb**->**Debug As**->**Debug Configurations**, navigate to **Debugger** -> **Debug Tool Settings**, and uncheck **Allow caching of flash contents**. Otherwise, when debugging bootloader applications, the memory window may show wrong information.

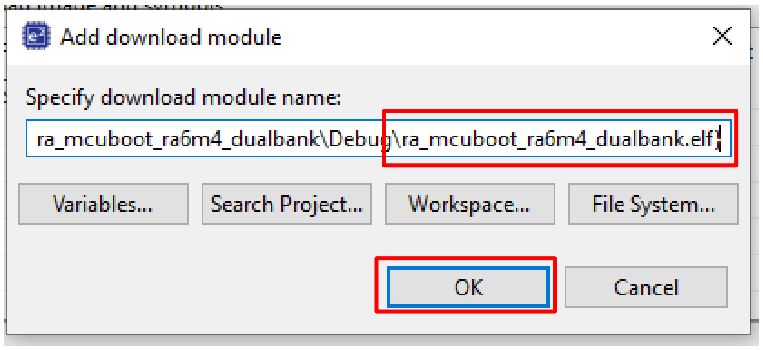


Debug hardware: J-Link ARM   Target Device: R7FA6M4AF

IO	
Use Default IO Filename	Yes
IO Filename	${support_area_loc}
General Debug	
Reset After Reload	Yes
Memory	
Endian	Little Endian
Break	
Use Flash Breakpoints	Yes
Allow Simulation	No
Flash	
Flash Bus Type	
Flash Memory Type	
WorkRam Start	
WorkRam End	
Erase on-chip program flash before download	No
Erase on-chip data flash before download	No
Use CFI-Flash	No
CFI Start	0x0
CFI End	0x0
Semihosting	
Semihosting breakpoint address	
RTOS	
RTOS Integration in Debug View	No
RTOS Debugging - Large Number of Threads.	No
System	
Allow caching of flash contents	No
Time Measurement	
Run Break Time Measurement	Yes
Count Every Core Cycle	Yes
Operating Frequency [MHz]	

Figure 65. Disable Flash Content Caching

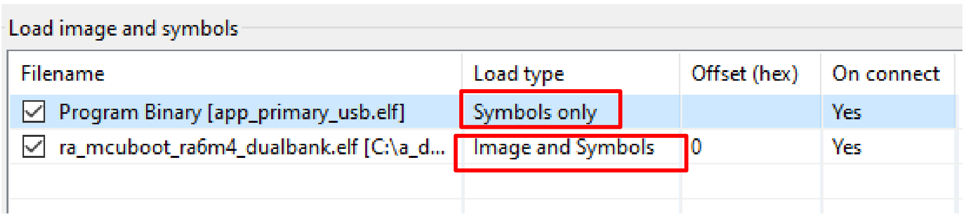
2. Configure the load image and symbols properties.
   Open the **Debug Configurations**: **app_primary_usb->Debug As->Debug Configurations**.
   Make sure **app_primary_usb Debug_Flat** is selected and select the **Startup** tab.
   Click **Add…**, then **Workspace**, navigate to the **ra_mcuboot_ra6m4_dualbank** project, and select the **ra_mcuboot_ra6m4_dualbank.elf** file from the debug folder. Click **OK**.



ra_mcuboot_ra6m4_dualbank\Debug\ra_mcuboot_ra6m4_dualbank.elf

Figure 66. Add the Bootloader Project to Debug Configuration

3. Change the **Load type** of the **Program Binaries** for the **app_primary_usb** project to **Symbols only** by clicking on the cell for **Load type** and selecting **Symbols only** from the drop-down menu.

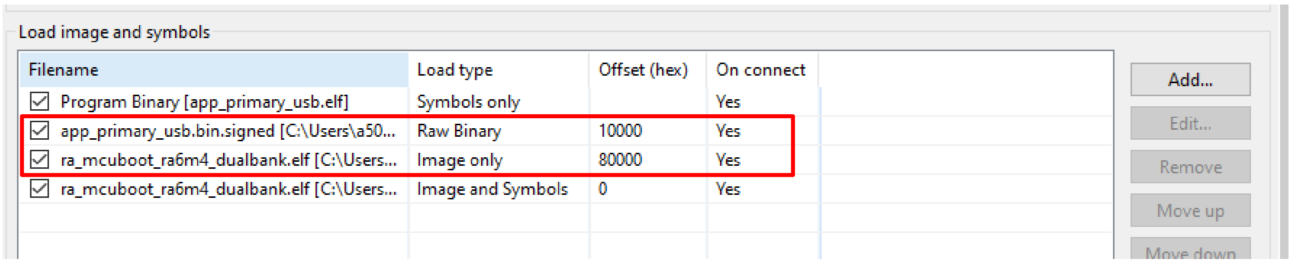


Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [app_primary_usb.elf]	Symbols only		Yes
☑ ra_mcuboot_ra6m4_dualbank.elf [C:\a_d...	Image and Symbols	0	Yes

Figure 67. Select to load Symbols only for the Application Project

4. Follow similar steps to add the signed primary image and the upper bank bootloader. Choose **Image only** as the **Load type** for the upper bank bootloader and choose **Raw Binary** as the **Load type** for the primary application image.

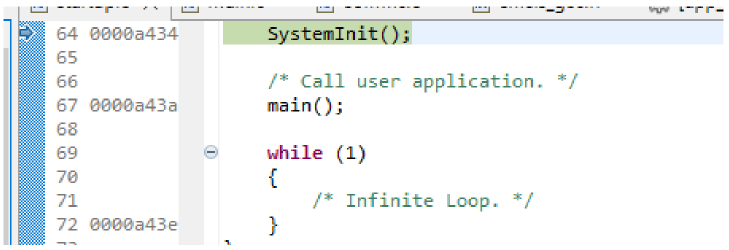


Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Program Binary [app_primary_usb.elf]	Symbols only		Yes
☑ app_primary_usb.bin.signed [C:\Users\a50...	Raw Binary	10000	Yes
☑ ra_mcuboot_ra6m4_dualbank.elf [C:\Users...	Image only	80000	Yes
☑ ra_mcuboot_ra6m4_dualbank.elf [C:\Users...	Image and Symbols	0	Yes

Figure 68. Add the Signed Primary Image and Upper Bank Bootloader

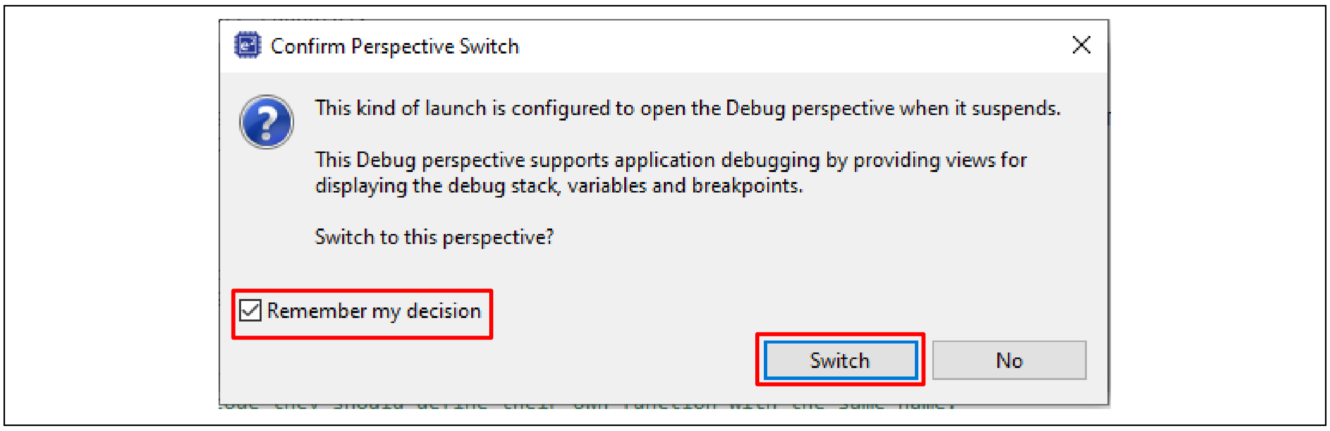
5. Click **Debug**. The debugger should hit the reset handler in the bootloader.



```
0000a434    SystemInit();

            /* Call user application. */
0000a43a    main();

            while (1)
            {
                /* Infinite Loop. */
0000a43e    }
```

Figure 69. Start the Application Execution

6. Choose **Remember my decision** and click **Switch** if prompted to switch the perspective.



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

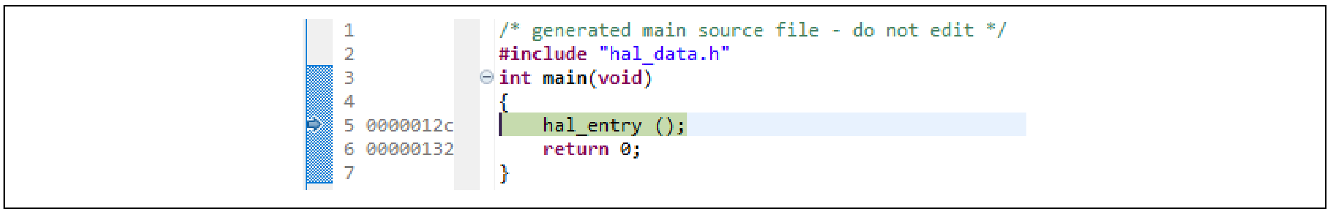
This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

☑ Remember my decision

Figure 70. Switch the Perspective

7. Click **Resume** to run the project.
   The program should now be paused in `main` at the `hal_entry()` call in the bootloader.



```
  0000012c
  00000132
/* generated main source file - do not edit */
#include "hal_data.h"
int main(void)
{
    hal_entry ();
    return 0;
}
```

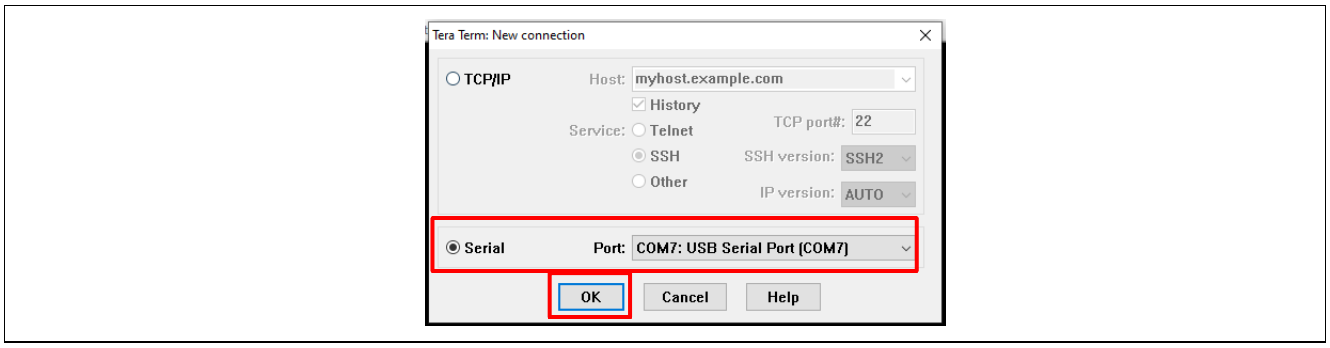
Figure 71. Start the Application Execution

8. Click to run again.
   The red, blue, and green LEDs on the EK-RA6M4 should now be blinking while the blinky application is running.

6.5 Program the New Application Using the Primary Application Downloader

Follow the steps below to program the new application created in section 6.1:

1. Open Tera Term and choose the USB Serial Port (COM number may be different for your setup). Then click **OK**.



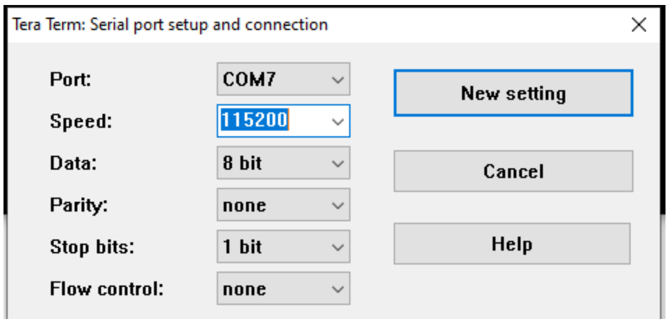
Tera Term: New connection

○ TCP/IP Host: myhost.example.com
☑ History
Service: ○ Telnet TCP port#: 22
◉ SSH SSH version: SSH2
○ Other IP version: AUTO

◉ Serial Port: COM7: USB Serial Port (COM7)

Figure 72. Open the COM Port

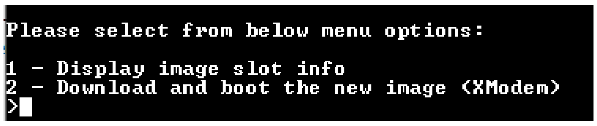
Note: When using the UART interface, select the Serial Terminal and set the **Speed** to 115200. Skip this step if using the USB interface.



Port: COM7
Speed: 115200
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none

Figure 73. Configure the Baud Rate if using UART Interface

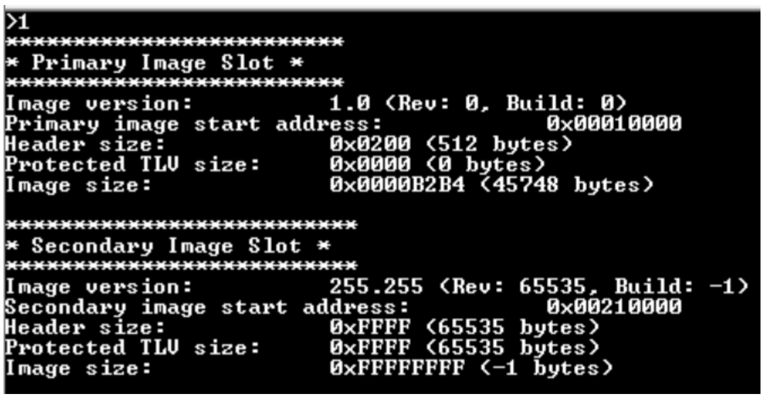
The menu in Figure 78 will be displayed on the Tera Term.



```
Please select from below menu options:

1 - Display image slot info
2 - Download and boot the new image (XModem)
>
```

Figure 74. Tera Term Menu

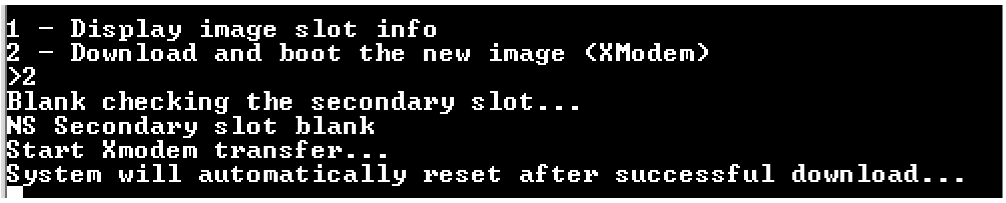
2. Select option 1 to print the image slot information**.**



```
>1
**************************
* Primary Image Slot *
**************************
Image version:          1.0 (Rev: 0, Build: 0)
Primary image start address:       0x00010000
Header size:            0x0200 (512 bytes)
Protected TLV size:     0x0000 (0 bytes)
Image size:             0x0000B2B4 (45748 bytes)

***************************
* Secondary Image Slot *
***************************
Image version:          255.255 (Rev: 65535, Build: -1)
Secondary image start address:     0x00210000
Header size:            0xFFFF (65535 bytes)
Protected TLV size:     0xFFFF (65535 bytes)
Image size:             0xFFFFFFFF (-1 bytes)
```

Figure 75. Print the Image Slot Information

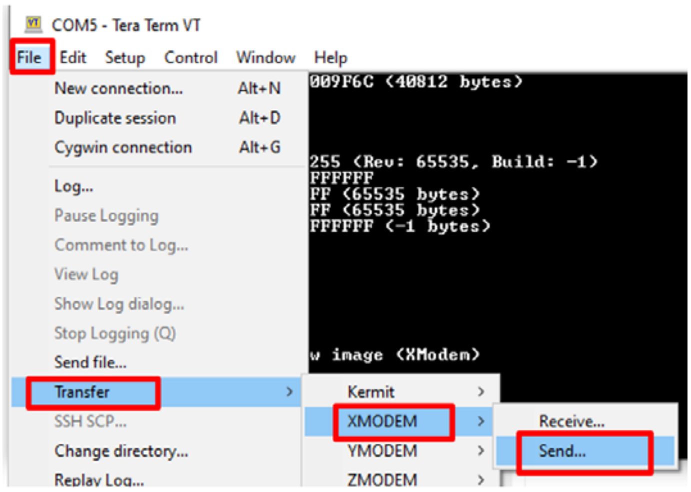
3. Select option **2** to download the secondary image using the primary image downloader.



```
1 - Display image slot info
2 - Download and boot the new image (XModem)
>2
Blank checking the secondary slot...
NS Secondary slot blank
Start Xmodem transfer...
System will automatically reset after successful download...
```

Figure 76. Choose Option 2 to Download the New Image using XModem

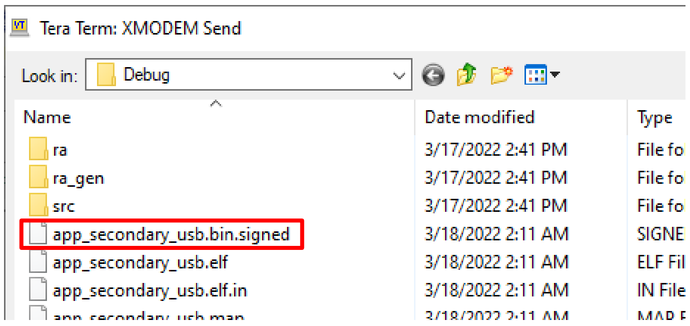
4. Open the **Transfer** interface of the Tera Term.



```
009F6C (40812 bytes)

255 (Rev: 65535, Build: -1)
FFFFFF
FF (65535 bytes)
FF (65535 bytes)
FFFFFF (-1 bytes)

w image (XModem)
```

Figure 77. Start Transfer from Tera Term

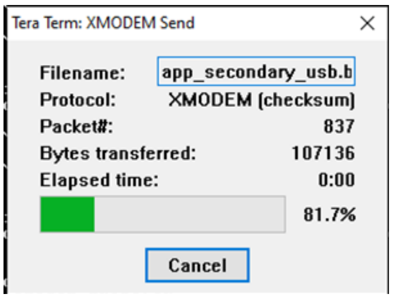
5. Choose \app_secondary_usb\Debug\app_secondary_usb.bin.signed, then click **Open**.



Name	Date modified	Type
ra	3/17/2022 2:41 PM	File fo
ra_gen	3/17/2022 2:41 PM	File fo
src	3/17/2022 2:41 PM	File fo
app_secondary_usb.bin.signed	3/18/2022 2:11 AM	SIGNE
app_secondary_usb.elf	3/18/2022 2:11 AM	ELF Fil
app_secondary_usb.elf.in	3/18/2022 2:11 AM	IN File

Figure 78. Choose the Signed Secondary Image

The secondary image is then downloaded and programmed to the upper bank.

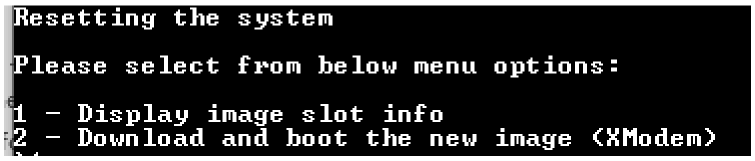


Filename: app_secondary_usb.b
Protocol: XMODEM (checksum)
Packet#: 837
Bytes transferred: 107136
Elapsed time: 0:00
81.7%

Figure 79. Download the New Image via XModem

6.6 Boot the New Application

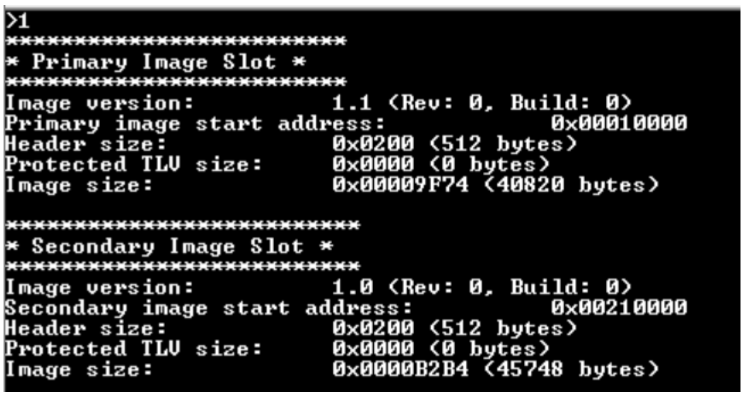
The system will automatically reboot after the new image is downloaded.



```
Resetting the system

Please select from below menu options:

1 - Display image slot info
2 - Download and boot the new image (XModem)
```

Figure 80. The New Image is Booted

Select option **1** to read the swapped memory layout.



```
>1
*************************
* Primary Image Slot *
*************************
Image version:              1.1 (Rev: 0, Build: 0)
Primary image start address:        0x00010000
Header size:                0x0200 (512 bytes)
Protected TLV size:         0x0000 (0 bytes)
Image size:                 0x00009F74 (40820 bytes)

**************************
* Secondary Image Slot *
**************************
Image version:              1.0 (Rev: 0, Build: 0)
Secondary image start address:      0x00210000
Header size:                0x0200 (512 bytes)
Protected TLV size:         0x0000 (0 bytes)
Image size:                 0x0000B2B4 (45748 bytes)
```

Figure 81. The Slot Layout After New Image is Booted

Note that even though the secondary image is booted, it cannot be debugged as the symbol downloaded to the debugger is for the primary image.

Also, if you want to perform further update, the new image must have a version of higher than the current image in the primary slot.

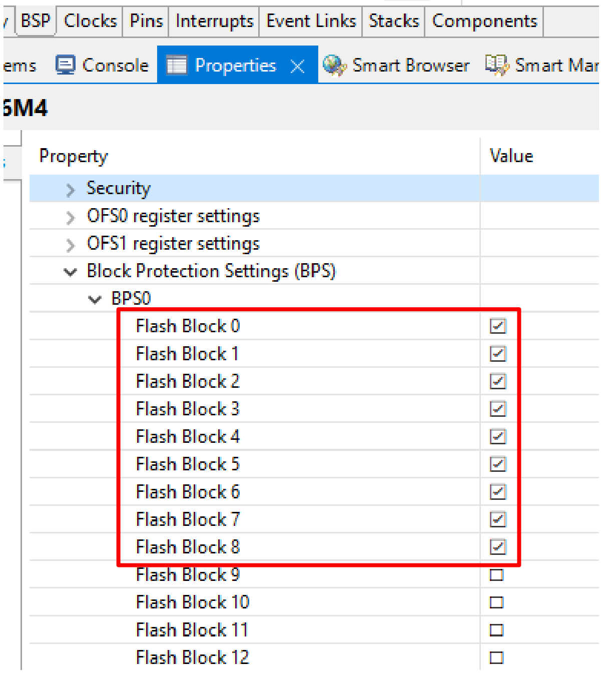
7. Production Support Considerations

This section describes one possible flow of production flow. Users may adapt this procedure to their own needs wherever possible.

7.1 Protect the Bootloader using Flash Block Protection

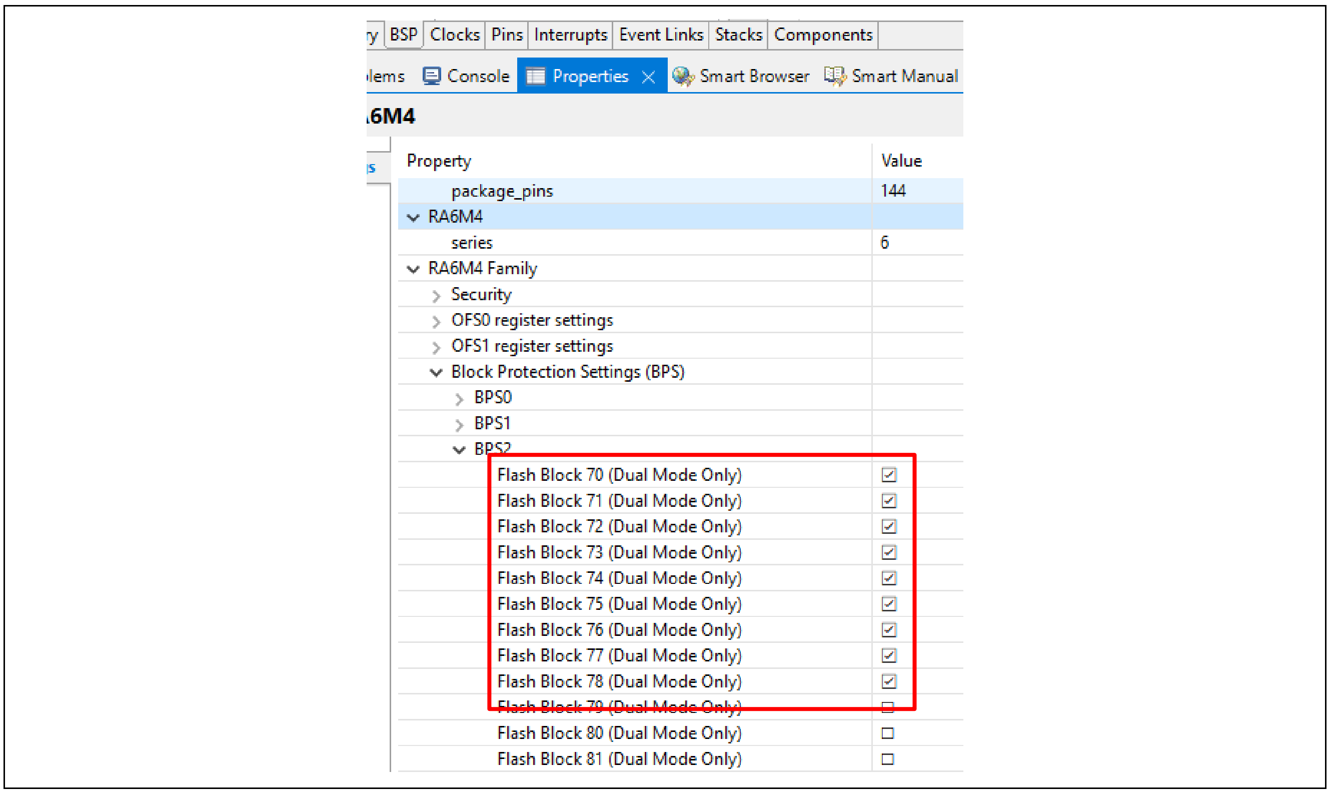
The secure bootloader protects the Root of Trust of the system. It should be protected from alteration by the application. Based on Figure 35, the bootloader is located in the first 64-KB region. Based on Figure 3, the blocks that need to be protected are blocks 0 to 8 for the lower bank and 70 to 78 for the upper bank.

Users can set up these blocks to be temporarily protected in the `ra_mcuboot_ra6m4_dualbank` project under the BSP tab. If these blocks are protected temporarily, the block protection setting can be reset by performing the MCU erase operations described in section 6.3.



Property	Value
› Security	
› OFS0 register settings	
› OFS1 register settings	
⌄ Block Protection Settings (BPS)	
⌄ BPS0	
Flash Block 0	☑
Flash Block 1	☑
Flash Block 2	☑
Flash Block 3	☑
Flash Block 4	☑
Flash Block 5	☑
Flash Block 6	☑
Flash Block 7	☑
Flash Block 8	☑
Flash Block 9	☐
Flash Block 10	☐
Flash Block 11	☐
Flash Block 12	☐

Figure 82. Temporary Protection of the Lower Bank Bootloader Area



y | BSP | Clocks | Pins | Interrupts | Event Links | Stacks | Components

lems | Console | Properties | Smart Browser | Smart Manual

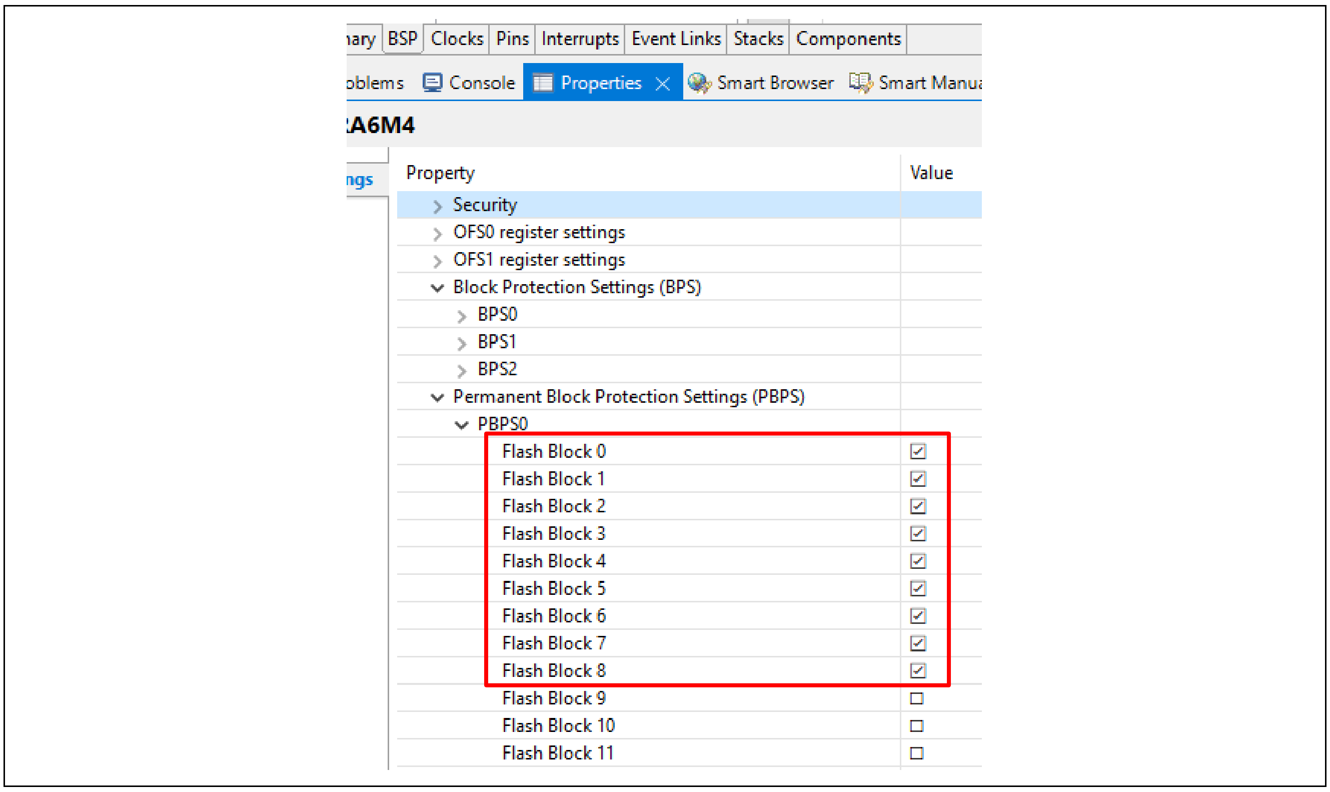
.6M4

Property	Value
package_pins	144
RA6M4	
series	6
RA6M4 Family	
Security	
OFS0 register settings	
OFS1 register settings	
Block Protection Settings (BPS)	
BPS0	
BPS1	
BPS2	
Flash Block 70 (Dual Mode Only)	☑
Flash Block 71 (Dual Mode Only)	☑
Flash Block 72 (Dual Mode Only)	☑
Flash Block 73 (Dual Mode Only)	☑
Flash Block 74 (Dual Mode Only)	☑
Flash Block 75 (Dual Mode Only)	☑
Flash Block 76 (Dual Mode Only)	☑
Flash Block 77 (Dual Mode Only)	☑
Flash Block 78 (Dual Mode Only)	☑
Flash Block 79 (Dual Mode Only)	☐
Flash Block 80 (Dual Mode Only)	☐
Flash Block 81 (Dual Mode Only)	☐

Figure 83. Temporary Protection of the Upper Bank Bootloader Area

Users can set up these blocks to be permanently protected in the `ra_mcuboot_ra6m4_dualbank` project under the BSP tab.

Note: If these blocks are protected permanently, these areas cannot be erased and reprogrammed through the lifetime of the MCU. Users need to be very cautious when setting up the permanent protection. The MCU erase operations described in section 6.3 will not be able to erase these blocks.



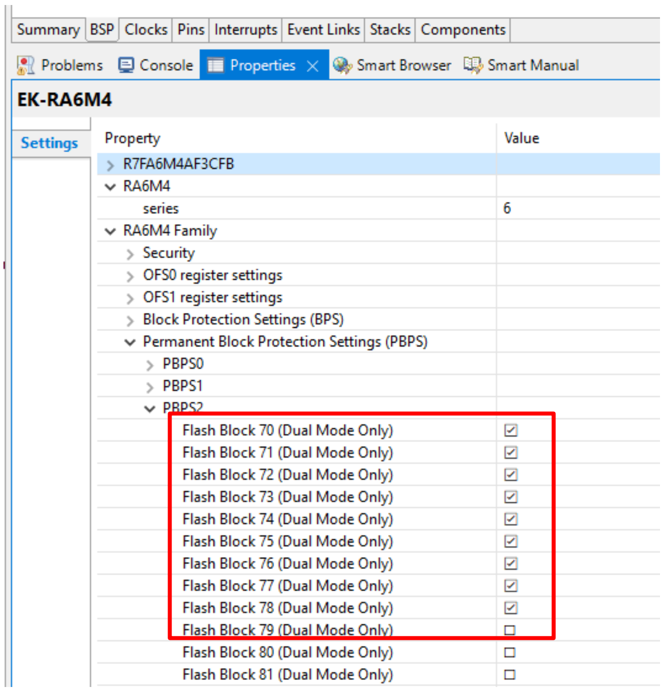
ary | BSP | Clocks | Pins | Interrupts | Event Links | Stacks | Components

oblems | Console | Properties | Smart Browser | Smart Manua

A6M4

Property	Value
Security	
OFS0 register settings	
OFS1 register settings	
Block Protection Settings (BPS)	
BPS0	
BPS1	
BPS2	
Permanent Block Protection Settings (PBPS)	
PBPS0	
Flash Block 0	☑
Flash Block 1	☑
Flash Block 2	☑
Flash Block 3	☑
Flash Block 4	☑
Flash Block 5	☑
Flash Block 6	☑
Flash Block 7	☑
Flash Block 8	☑
Flash Block 9	☐
Flash Block 10	☐
Flash Block 11	☐

Figure 84. Permanent Protection of the Lower Bank Bootloader Area



Property	Value
R7FA6M4AF3CFB	
RA6M4	
series	6
RA6M4 Family	
Security	
OFS0 register settings	
OFS1 register settings	
Block Protection Settings (BPS)	
Permanent Block Protection Settings (PBPS)	
PBPS0	
PBPS1	
PBPS2	
Flash Block 70 (Dual Mode Only)	☑
Flash Block 71 (Dual Mode Only)	☑
Flash Block 72 (Dual Mode Only)	☑
Flash Block 73 (Dual Mode Only)	☑
Flash Block 74 (Dual Mode Only)	☑
Flash Block 75 (Dual Mode Only)	☑
Flash Block 76 (Dual Mode Only)	☑
Flash Block 77 (Dual Mode Only)	☑
Flash Block 78 (Dual Mode Only)	☑
Flash Block 79 (Dual Mode Only)	☐
Flash Block 80 (Dual Mode Only)	☐
Flash Block 81 (Dual Mode Only)	☐

Figure 85. Permanent Protection of the Upper Bank Bootloader Area

The included example bootloader does not include the block settings to enable block protection. Users can enable them prior to field deployment.

7.2 Provision the Bootloaders and the Initial Application to MCU

Users can combine the `.srec` files generated from the above sections into one `.srec` file and program it to the MCU during production.

The three images to be combined are:

- Bootloader for the Lower Bank: `ra_mcuboot_ra6m4_dualbank.srec`
- Bootloader for the Upper Bank: `ra_mcuboot_ra6m4_dualbank_offset.srec`
- Application for the Lower Bank: `app_primary_usb_signed_offset.srec`

Use the following command to generate one combined `.srec` from the above three `.srec` files:

```
srec_cat ra_mcuboot_ra6m4_dualbank.srec ra_mcuboot_ra6m4_dualbank_offset.srec
app_primary_usb_signed_offset.srec -o combined.srec
```

Download `combined.srec` to the MCU using RFP or J-Flash Lite in the same way as programming the `ra_mcuboot_ra6m4_dualbank_offset` as explained in section 6.4.

Once the device is deployed to the field, the application update can be achieved using the image downloader implemented in the application project.

8. Compile and Exercise the Included Example Bootloader and Application Projects

8.1 Using USB as the Download Interface

For the USB interface, three projects are needed:

- `ra_mcuboot_ra6m4_dualbank`
- `app_primary_usb`
- `app_secondary_usb`

Users can follow the steps below to run the example projects in the folder `\RA6_Secure_Bootloader_Dualbank\example_projects_with_bootloader`:

1. Follow the instructions in section 6.2 to set up the hardware.

2. Import the above-mentioned three projects to a Workspace.
3. Open the `Configuration.xml` file from project `ra_mcuboot_ra6m4_dualbank`.
4. Click **Generate Project Content.**
5. Compile the project `ra_mcuboot_ra6m4_dualbank`.
6. Open the `Configuration.xml` file from project `app_primary_usb`.
7. Click **Generate Project Content**.
8. Compile the `app_primary_usb`.
9. Open the `Configuration.xml` file from project `app_secondary_usb`.
10. Click **Generate Project Content**.
11. Compile the `app_secondary_usb` project.
12. Erase the entire chip following instructions in section 6.3.
13. Debug the application from project `app_primary_usb` in the e^2 studio environment.
14. Resume the program execution twice. All three LEDs should be blinking.
15. Stop the debug session and power cycle the EK-RA6M4.
16. Open Tera Term with the enumerated COM port (USB Serial Device).
17. Use Tera Term to send the `\app_secondary_usb\Debug\app_secondary_usb.bin.signed` to the MCU following the instructions in section 6.6. This will take about 30 seconds.
18. System will be reset automatically after download.
19. Blue and green LEDs should be blinking.
20. Enter menu item 1 to confirm the image with version 1.1.0 is located in the primary slot (lower bank) and the image with version 1.0.0 is located in the secondary slot (upper bank).

8.2 Using the UART as the Download Interface

For the UART interface, three projects are needed:

- `ra_mcuboot_ra6m4_dualbank`
- `app_primary_uart`
- `app_secondary_uart`

Users can follow the steps below to run the example projects in the folder `\RA6_Secure_Bootloader_Dualbank\example_projects_with_bootloader`:

1. Follow the instructions in section 6.2 to set the hardware.
2. Import the above-mentioned three projects to a workspace.
3. Open the `Configuration.xml` file from project `ra_mcuboot_ra6m4_dualbank`.
4. Click **Generate Project Content**.
5. Compile the project `ra_mcuboot_ra6m4_dualbank`.
6. Open the `Configuration.xml` file from project `app_primary_uart`.
7. Click **Generate Project Content**.
8. Compile `app_primary_uart`.
9. Open the `Configuration.xml` file from project `app_secondary_uart`.
10. Click **Generate Project Content**.
11. Compile the `app_secondary_uart` project.
12. Erase the entire chip following the instructions in section 6.3.
13. Debug the application from project `app_primary_uart` in the e^2 studio environment.
14. Resume the program execution twice. All three LEDs should be blinking.
15. Stop the debug session and power cycle the EK-RA6M4.
16. Open the Tera Term with the enumerated COM port and set up the baud rate as 115200.
17. Use Tera Term to send the `\app_secondary_uart\Debug\app_secondary_uart.bin.signed` to the MCU by following section 6.6. This will take about 50 seconds.
18. System will be reset automatically after download.
19. Blue and green LEDs should be blinking.
20. Enter menu item 1 to confirm the image with version 1.1.0 is located in the primary slot (lower bank) and the image with version 1.0.0 is located in the secondary slot (upper bank).

9. References

1. Renesas RA Family MCU Securing Data at Rest using Security MPU Application Project (R11AN0416)
2. Using MCUboot with RA6 Family MCUs Application Project (R11AN0497)
3. Using MCUboot with RA2 Family MCUs Application Project (R11AN0516)
4. Using MCUboot with Encrypted Image and QSPI (R11AN0567)

10. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA6M4 Resources	renesas.com/ra/ek-ra6m4
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	March 21, 2022	-	First release document
1.10	Nov. 11, 2022	-	Updated Operation Flow based on e^{2}studio 2022-10 or later. Used FSP v4.0.0. Document title changed from “RA6 Secure Bootloader Update using MCUboot and Flash Dual Bank” to “RA6 Secure Firmware Update using MCUboot and Flash Dual Bank”
1.1.1	Nov. 23, 2022	-	Corrected typo, added Figure 56 and included RA6E1.

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/.