Renesas RA Family

RA2 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 with fail-safe application updates.

MCUboot is maintained at 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 the RA Family RA2 MCU Group, 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 starting with FSP v5.6.0. This application note demonstrates secure bootloader design using this dual bank feature for a Non-TrustZone environment based on RA2A2.

The included EK-RA2A2 application project demonstrates two firmware update methods: **Bank Swap with Reset** and **Bank Swap Instant**. **Bank Swap with Reset** is intended for applications where a full device reset and clean restart are acceptable after updating the firmware image. In contrast, **Bank Swap Instant** targets use cases that require continuous operation during the update – such as industrial control, building automation, or connected consumer devices - where downtime must be minimized. The project illustrates how to implement this fast image swap (Bank Swap Instant) in a practical application scenario.

Users can review the flash layout for RA2Users that and port the application to other MCUs. In addition, steps are provided for how to master an application to use with the bootloader and how to update to a new application. Users can follow these steps to recreate the reference bootloader and link the examples included in this application project to use the bootloader.

If you are interested in the basic secure bootloader design using the MCUboot module with RA2 internal code flash in linear mode, please refer Secure Bootloader for RA2 MCU Series – Application Project.

Required Resources

Development tools and software

- e² studio IDE v2025-04.1
- Renesas Flexible Software Package (FSP) v5.9.0
- SEGGER J-link® USB driver v8.44a

The three software components: the FSP, J-Link USB drivers and e² studio are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

- Python v3.11.0 or later- https://www.python.org/downloads/
- Renesas Flash Programming (RFP) v3.20.00 https://www.renesas.com/us/en/software-tool/renesas-flash-programmer-programming-gui

Hardware

- EK-RA2A2, Evaluation Kit for RA2A2 MCU Group http://www.renesas.com/ra/ek-ra2a2
- Workstation running Windows® 10
- One USB device cable (type-A male to micro-B male)

Prerequisites and Intended Audience

Users of this application 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.

It is recommended to start with **the Quick Start Example Project** on the EK-RA2A2 to verify that Renesas e^2 studio and Segger J-Link are set up properly and the driver versions are up to date. The Quick Start Example Project is available in the EK-RA2A2 Example Projects Bundle that is available in the Downloads tab of EK-RA2A2 webpage at http://www.renesas.com/ra/ek-ra2a2

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 the RA2A2 MCU. 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. Users 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 that 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 instructions on how to debug and boot the primary application project and update to a new image. Users that 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 production support considerations of provisioning a 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 the RA2A2 MCU group, the internal flash memory can operate in user mode (single bank) or dual bank mode. In user mode, the code flash memory is used as one area. In dual bank mode, the code flash memory is divided into two areas and the bank swap function can be used to boot into a new application for a system that includes a bootloader.

1.1 RA2A2 MCU Group Code Flash Configuration

Using the 512-Kbyte MCU as an example, the code flash memory in user mode for RA2A2 includes the blocks shown in Figure 1.

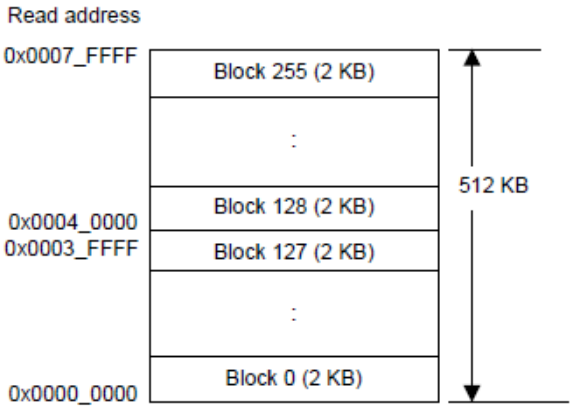


Figure 1. RA2A2 Code Flash Memory in User Mode

Upper Bank Address in Code Flash User Mode

In user mode, the upper bank starting address is half of the code flash size. For example, for the 512-Kbyte RA2A2 MCU used in this example project, the starting address of the upper bank address is 0x40000. The upper bank user mode address is used when downloading the upper bank bootloader using MCUboot in code flash dual bank mode.

Using the 512-Kbyte MCU as an example, the code flash memory in dual bank mode includes the blocks shown in Figure 2.

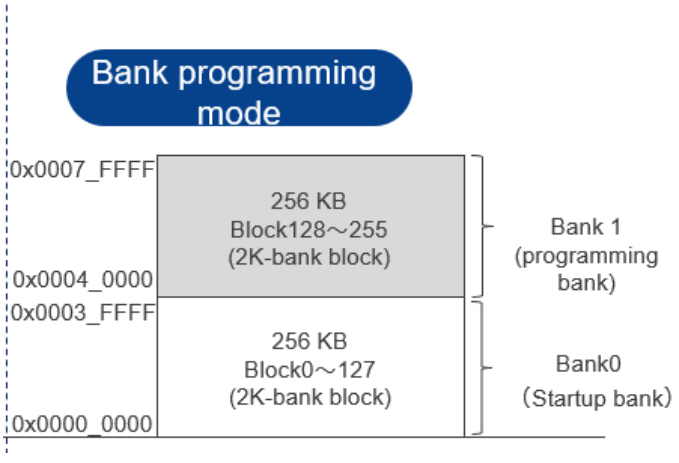


Figure 2. RA2A2 Code Flash Memory in Dual Bank Mode

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 0x40000. This address should be used with the application image downloader.

Table 1. RA2A2 Code Flash

Product	Code Flash Range Address	
	Linear	Dual
512 Kbyte MCU	0x0000_0000 to 0x0007_FFFF	Lower bank: 0x0000_0000 to 0x0003_FFFF
		Upper bank: 0x0004_0000 to 0x0007_FFFF

1.2 Option-Setting Memory

The description in this section applies to the RA2A2. The Option-Setting Memory of the MCU determines the state of the MCU after a reset. Several property settings that relate to the code flash mode are described in this section. See Figure 3. Option-Setting Memory.

When bank swap function under the bank programming mode is used, set the same value at 0x0004_0400 to 0x0004_043B as that at 0x0000_0400 to 0x0000_043B, because switching between bank0 (0x0000_0400 to 0x0000_043B) and bank1 (0x0004_0400 to 0x0004_043B) will proceed.

The Flash Control Flag (FCTLF) register is used to set the Bank Swap Startup Bank. The Flash Bank Program Control Register (FBKPGCR) sets the Bank Mode (User or Bank Programming). Both registers can be rewritten by the control registers in the flash software command. For details, see section 35, Flash Memory of the User's Manual Hardware (r01uh1005).

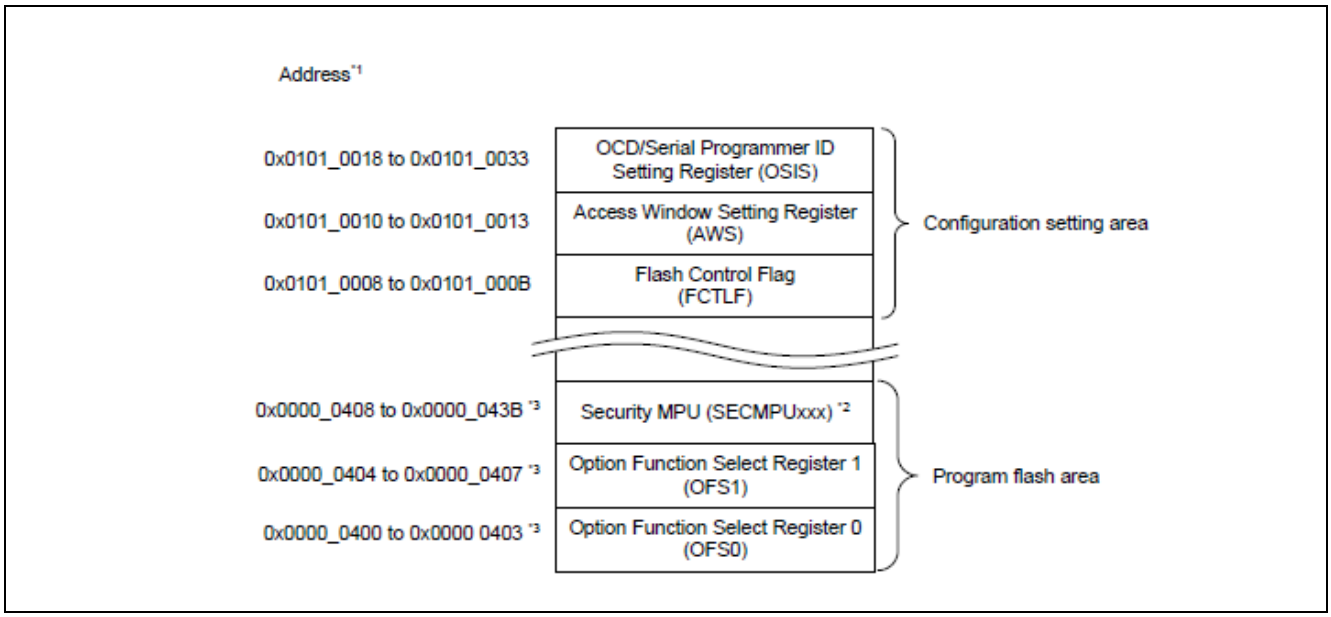


Figure 3. Option-Setting Memory

Note 1: The option-setting memory must be allocated to the user area of the flash memory.

Note 2: See Table 7.2 of the User's Manual Hardware (r01uh1005) for more details.

Note 3: The address of these registers will be changed when the boot swap is set. See section 7.2.1. OFS0 : Option Function Select Register 0, section 7.2.2. OFS1: Option Function Select Register 1 and section 7.2.3. MPU Registers of the User's Manual Hardware (r01uh1005) for details.

1.2.1 Code Flash Bank Mode

The register that configures the code flash bank mode is in the Flash Memory of the MCU.

For a blank MCU, the code flash is in user 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.

35.3.28 FBKPGCR : Flash Bank Program Control Register

Base address: FLCN = 0x407E_C000

Offset address: 0x3FD0

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

Bit	Symbol	Function	R/W
0	BKPGEN	Bank Programming Setting Enable 0: Bank programming is disabled (User mode) 1: Bank programming is enabled (Bank programming mode)	R/W
7:1	—	These bits are read as 0. The write value should be 0.	R/W
15:8	FEKEY[7:0]	Key Code	W

The FBKPGCR register controls the bank programming mode setting.

To set or clear this register, the FENTRYR register must be set to the Code flash P/E mode (FENTRYR.FENTRY0 = 1) and FPMCR register is not the Code flash P/E mode (FPMCR.FMS0 != 1). Additionally, data is written with FEKEY[7:0] bits = 0x6D by word access.

BKPGEN bit (Bank Programming Setting Enable)

Figure 4. Register Configuration for Code Flash Dual Bank Programming Mode

1.2.2 Startup Bank Selection

The description in this section applies to RA2A2 MCUs. Bank 0 is the lower bank for a blank RA2A2 MCU as defined by the Bank Select registers shown in Figure 5.

7.2.6 FCTLF : Flash Control Flag

Address: 0x0101_0008

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															

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															

Bit	Symbol	Function	R/W
2:0	BANKSWP[2:0]	Majority Value of These Bits Decides a Startup Bank 0: Bank 1 is operating. 1: Bank 0 is operating.	R
31:3	—	When read, these bits return the written value.	R

Note: This register can be rewritten by the flash control flag setting in the flash software command. For details, see section 35, Flash Memory.

Figure 5. Bank 0 is Default at Address 0x00000000

1.2.3 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.

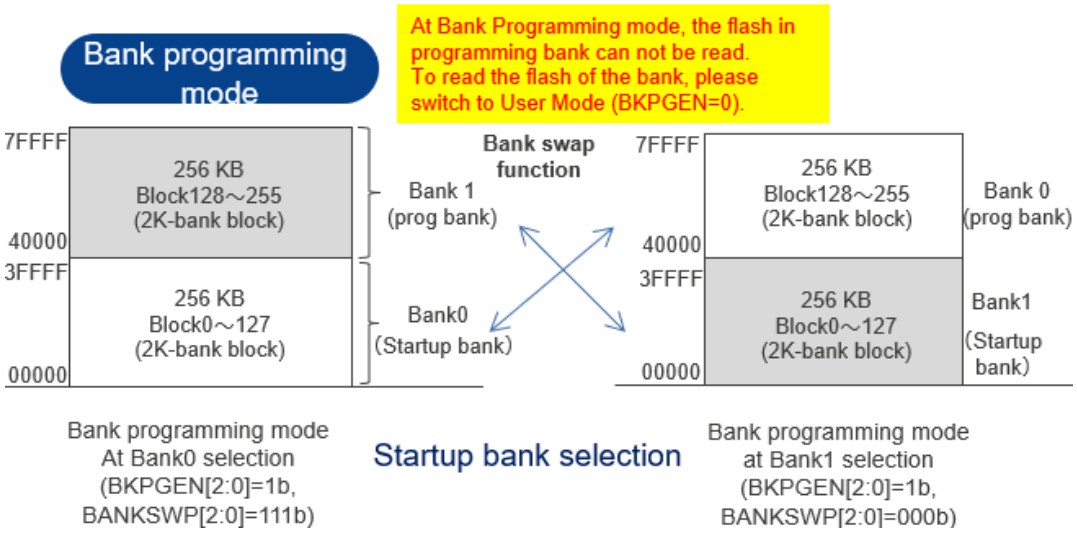


Figure 6. Example of Startup Bank Selection

Bank selection can be changed at runtime through the FSP API. The BANKSWP bits in the FCTLFR register can be changed at the application level. The FSP flash driver provides the `R_FLASH_LP_BankSwap()` API to facilitate this action. This API is automatically called from the FSP MCUboot module. The swap takes affect after the next reset or without reset depending on the configuration of FSP flash driver Instant Bank Swap setting.

1.2.4 Security Memory Protection Unit and Flash Access Window

The RA2A2 MCU incorporates a security MPU with four secure regions that include the code flash, SRAM, and two security functions. The secure regions can be protected from non-secure program access. A non-secure program cannot access a protected region. Refer to Figure 7 Security MPU Secure Regions for more information.

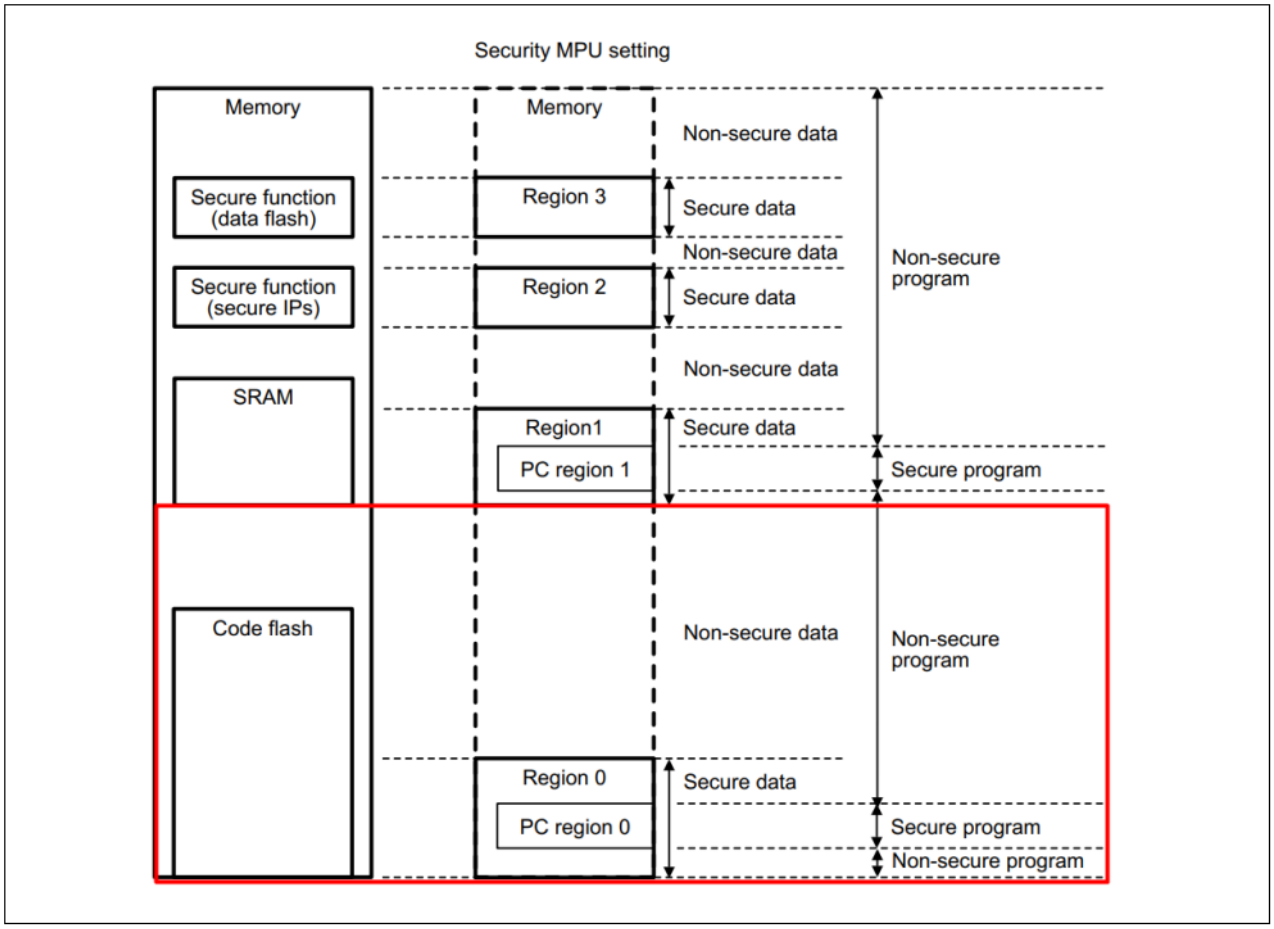


Figure 7. Security MPU Secure Regions

The Flash Access Window (FAW) in Figure 8 defines one contiguous flash region within the MCU flash space. Within this region, flash erase and write operations are allowed from both secure and non-secure programs. The access window is only valid in the code flash area.

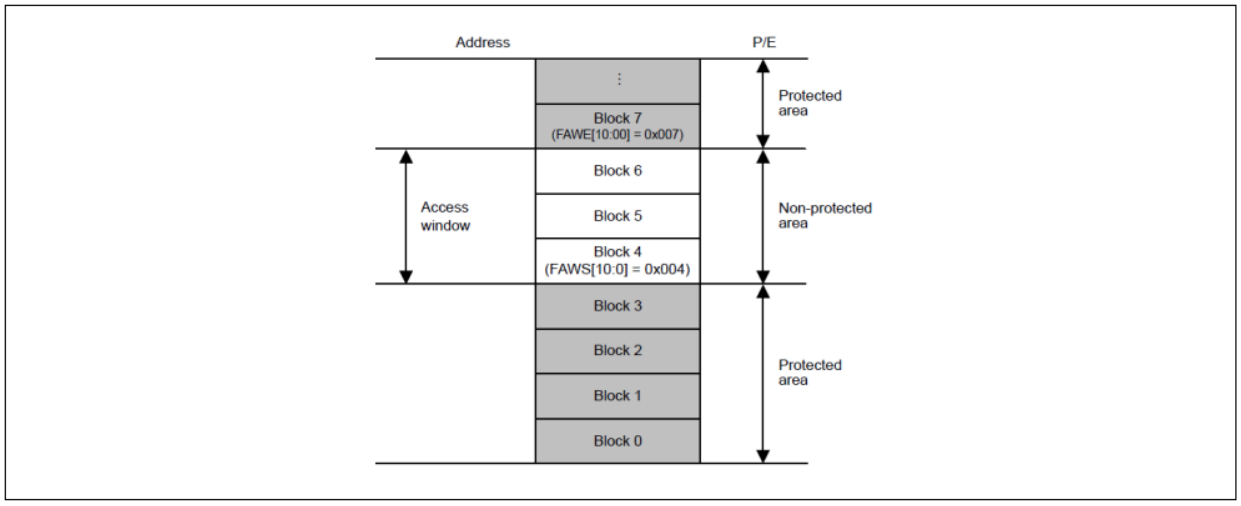


Figure 8. Flash Access Window

By combining the MPU and FAW, the bootloader's protection will be enhanced. To help users easily visualize the implementation, we provide the diagram as shown in Figure 9.

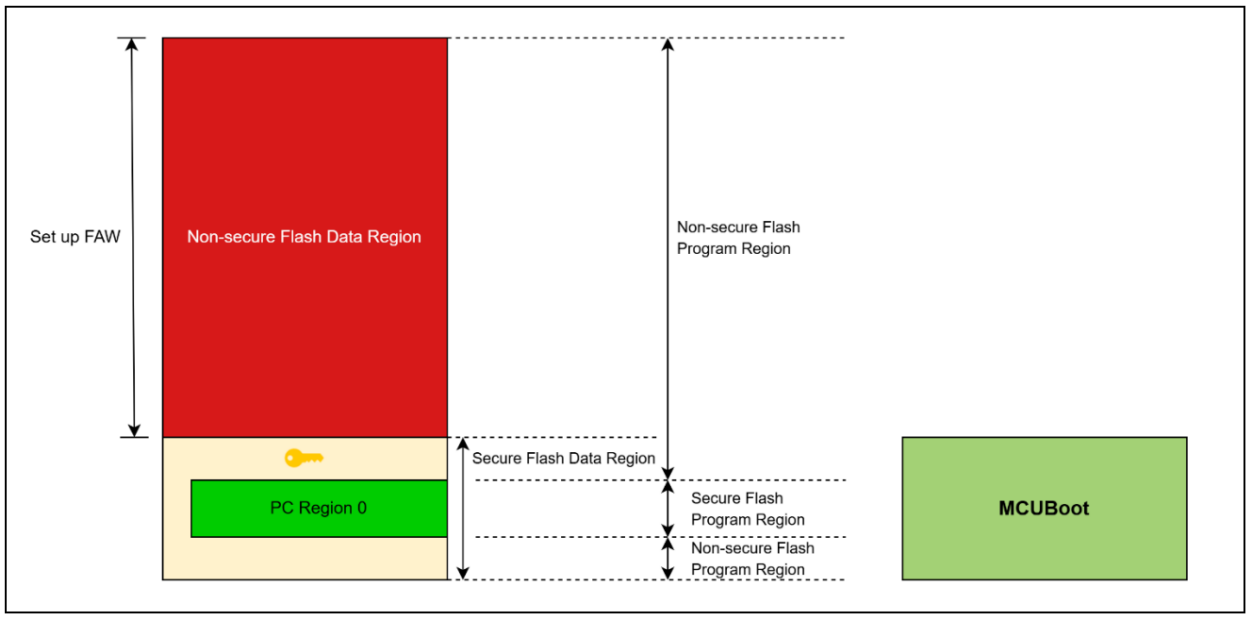


Figure 9. Combining the MPU and FAW to protect the bootloader

In addition, for manufacturing usage permanently locking the FAW region prevents a user from updating the FAW region. Users can refer to Permanent Locking of the FAW Region section in Application Project R11AN0416. Note: When permanently locking the FAW region, this action is irreversible.

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.

2.1 MCUboot Functionalities Overview

MCUboot handles the firmware authenticity check after start-up and the runtime image selection step 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(s).
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 application 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 understand and use 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.

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 10. In addition, the memory allocation for the bootloader and application image must be identical.

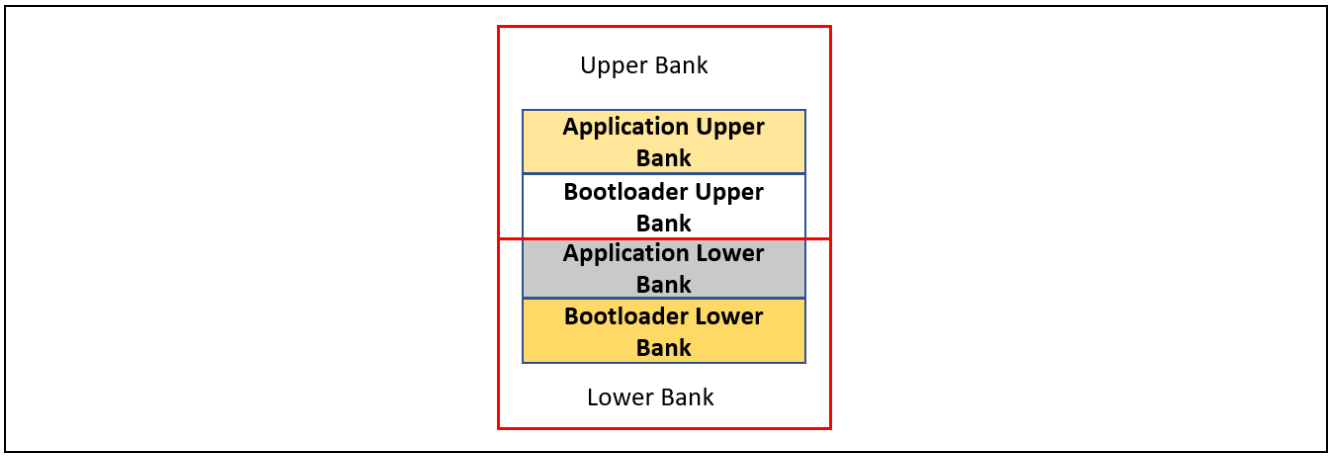


Figure 10. Memory Architecture Using Flash Dual Bank Mode and MCUboot

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 of the example projects in this application note.

3. Guidelines for Using the Example Projects

Unzip `RA2_Secure_Bootloader_DualBank.zip` to unpack the example projects included in this application project.

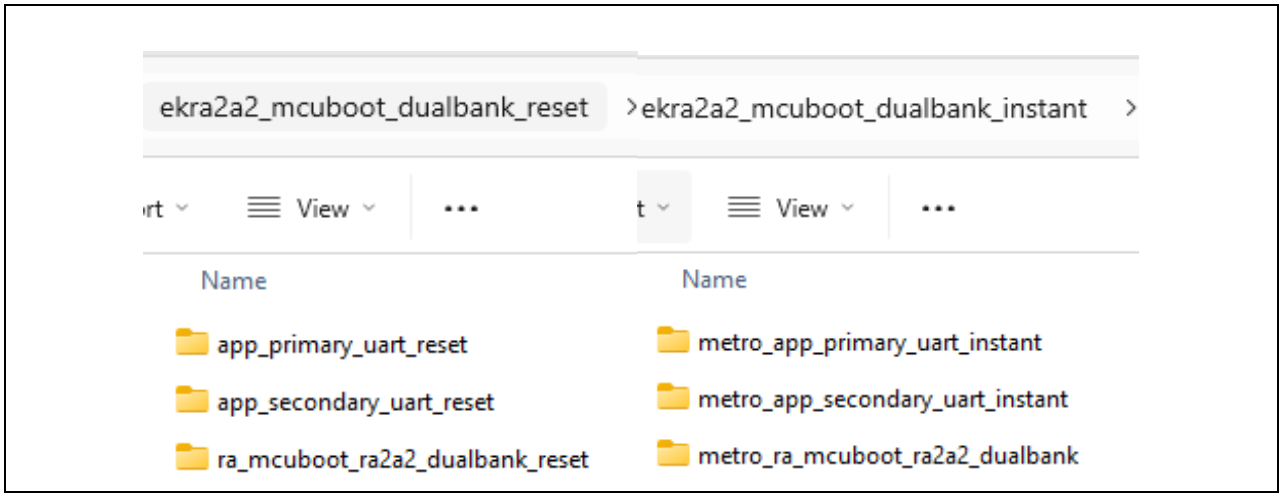


Figure 9. Example Projects Included

3.1 Dual Bank Flash Update with Bank Swap (with Reset)

Folder `\ekra2a2_mcuboot_dualbank_reset` includes a bootloader, which supports the flash dual bank swap with reset feature, as well as the applications that use XMODEM protocol over UART as the communication channel to download new application images 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_ra2a2_dualbank_reset` – Bootloader, which enables dual bank and direct XIP update mode.

`app_primary_uart_reset` – 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 thread blinks the red LED on EK-RA2A2 while the other thread downloads the new application image concurrently.

A metrology application monitors ICU interrupts generated from a square wave signal connected to input P306. The board Green LED is toggled on for each falling edge of the square wave signal connected to input pin P306 of Header J3.

`app_secondary_uart_reset` - Secondary application, which implements the same functionality as `app_primary_uart_reset` except only the blue LED is blinked.

- Note: the square wave signal connected to P306 should have a frequency of 5 Hz to 15 Hz

3.2 Dual Bank Flash Update with Bank Swap Instant (without Reset)

Folder `\ekra2a2_mcuboot_dualbank_instant` includes a bootloader, which supports the flash dual bank swap instant (without reset) feature, as well as example applications using XMODEM protocol over UART as the communication channel to download new application images which are configured to use the bootloader included in this folder. The subfolders are:

`metro_ra_mcuboot_ra2a2_dualbank` - Bootloader, which enables dual bank swap instant and direct XIP update mode.

`metro_app_primary_uart_instant` - 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 thread blinks the red LED on EK-RA2A2 while the other thread downloads the new application image concurrently.
A metrology application monitors ICU interrupts generated from a square wave signal connected to input P306. The board Green LED is toggled on for each falling edge of the square wave signal connected to input pin P306 of Header J3.

`metro_app_secondary_uart_reset` - Secondary application, which implements the same functionality as `metro_app_primary_uart_instant` except only the blue LED is blinked.

- Note: the square wave signal connected to P306 should have a frequency of 5 Hz to 15 Hz.

3.3 Discussion of Dual Bank Flash Update Projects

Both Dual Bank Flash Update projects contain a bootloader that is configured to put the device initially into Bank 0 of the program flash memory. Dual Bank mode is enabled in the bootloader at compile time. The user application will download the new update image to Bank 1. At the next reset (or in the case of bank swap instant at MCU soft startup), the bank will be swapped. After the update and bank swap, the debug symbol table will be invalid.

The Dual Bank Update projects use two methods for the programming flow. The two methods update the device with a programmed image; however, one is a startup bank change using Bank Swap with MCU Reset, and the other is Bank Swap without MCU Reset.

Figure 12 shows the two supported update methods with the pros and cons for each method.

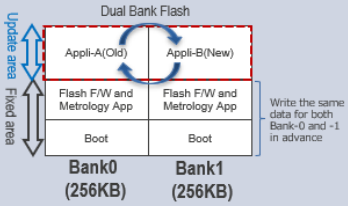
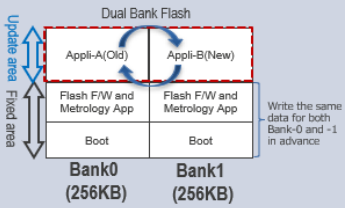
	(1-1) Dual Bank Flash with Bank Swap (Instant)	(1-2) Dual Bank Flash with Bank Swap (with Reset)
F/W Switching Method	Bank Swap Instant	Bank Swap with HW reset
Metrology app operation during Update	Continuous operation during Bank switching and image validation	Stopped during Bank switching and image validation (less than 1 second)
Configuration : Updatable Area		
Pros	✓ Non-stop Metrology app operation ✓ Easy address management	✓ Safety update by system reset ✓ Easy address management
Cons	✓ The usable flash area becomes small due to preparation two fixed areas. ✓ Software design is required so that the system can be started w/o full HW reset after switching to the updated F/W.	✓ Metrology operation stopped during bank switching and image validation. ✓ The usable flash area becomes small due to preparation of two fixed areas.

Figure 10. Supported Update Methods

Bank Swap Instant allows continuous app operation during the update Bank switching and image validation. This flow may be suitable for running a measuring process or metrology application. Careful software design is required so that the system can be started without a full hardware reset.

The projects in folder `ekra2a2_mcuboot_dualbank_instant` provides an example of how to create a bootloader with primary and secondary applications that operate a metrology application that runs continuously.

The metrology app samples a square wave signal connected to input P306 of Header J3. The application continues to operate during the image update process, bank swap, and MCU start up to run the secondary application image.

Key points for continuous app (metrology) operation:

- Interrupts cannot be disabled or critical sections entered during an MCU Soft Start.
- The metrology app must continue to use required interrupts. For a fast startup process, minimize the interrupts used during the Soft Start
- MCU Hard / NVIC resets cannot be used, or they will interrupt the continuous application
- Use a MCU Soft Start which jumps to reset vector address 0x0 and required HW peripherals are not initialized.
- NVIC reset does invoke HW initialization of registers
- The user must change I/O and Peripheral initialization behavior so that resources are not re-initialized after the image update and bank swap.
- Use RAM flags in Primary and Secondary App to check start-up execution to not initialize I/O and Peripherals once the Bank Swap and Soft Start complete.

Bank Swap with HW reset provides a method to update the new application image with a safe update by system reset and easy address management. The example projects in folder `ekra2a2_mcuboot_dualbank_reset` provides an example of how to create a bootloader with primary and secondary applications that operate a metrology application; however, the metrology app must be stopped during the bank switching and image validation. The metrology app is restarted after the secondary application is started after the MCU reset completes.

For more details of the startup bank selection flows, refer to section 35.5 Operating Modes Associated with the Flash Memory of the User's Manual Hardware (r01uh1005).

Figure 13 shows the operation flows of the two supported update methods.

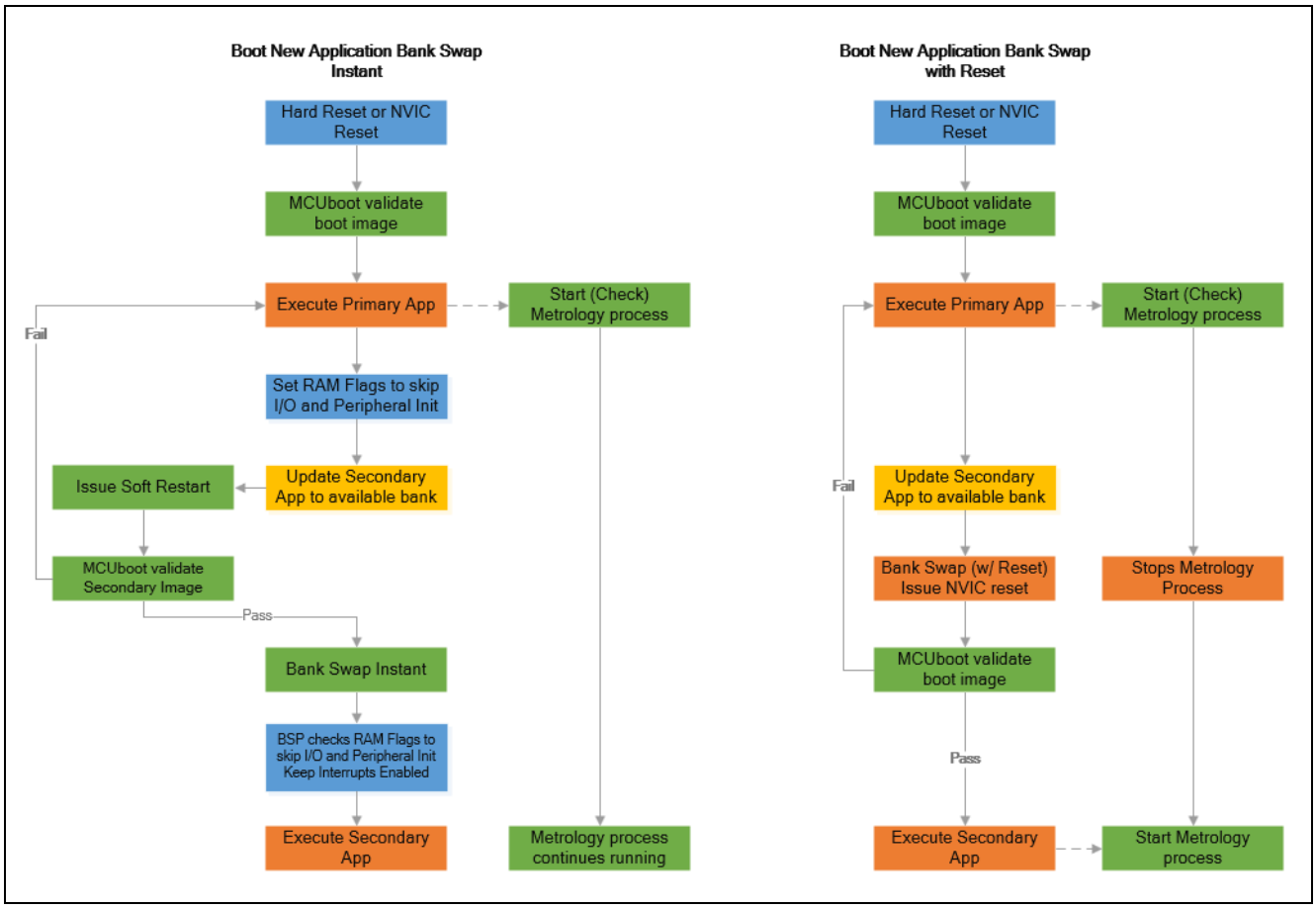


Figure 11. Image Update Operation Flows

Key Points for the Operation Flows:

- The Bank Swap Instant method uses an MCU soft re-start to preserve continuous operation of a metrology process.
- The Bank Swap Instant method sets RAM Flags after the Secondary Image is programmed to flash and checks the RAM flags to skip the initialization of Inputs/Outputs and Peripherals that are required by the metrology application.
- In Bank Swap Instant and Bank Swap Reset, MCUboot is used to validate the Secondary App Image
- If the Secondary App Image fails validation, then the Primary App is executed.
- In Bank Swap with Reset method, a metrology application is stopped during the Bank Swap operation and re-started when the Secondary App runs.

The next section provides software implementation highlights for the operation flows as detailed in Figure 13.

With Dual Bank Swap with Reset, the metrology process is operational at startup and the primary application downloader implements the XMODEM over UART protocol for the image transfer to the MCU program flash.

As shown in Figure 14, once the image transfer reaches end of transmission (EOT) then the download is complete and MCU is restarted with the call to NVIC_SystemReset.

The bootloader will then run, MCUboot validates the secondary application, re-starts the Metrology process, and then runs the secondary application as shown in Figure 15.

xmodem.c

```
179             // if first received byte is 'end of frame'
180             // Secondary Image File Xmodem Transfer has completed so return ACK to sender
181             if ( RxByteBuffer[0] == EOT )
182             {
183                 tx_byte = ACK;
184                 comms_send(&tx_byte, 1);
185                 // Perform the Bank Swap and reset MCU to run from Flash Bank 1 (Secondary Image)
186                 // Dual Bank mode must be supported by the MCU and enabled
187                 // with FSP Swap support.
188                 // MCUBoot performs the R_FLASH_BankSwap(gp_mcuboot_flash_ctrl)
189                 // which is called in bootloader startup RM_MCUBOOT_PORT_BootApp(..)
190                 sprintf((char *)tx_str, "Secondary Download completed.\r\n");
191                 comms_send(tx_str, strlen((char *)tx_str));
192                 sprintf((char *)tx_str, "Resetting the system to run MCUboot\r\n");
193                 comms_send(tx_str, strlen((char *)tx_str));
194                 NVIC_SystemReset();
195                 return( XM_OK );
196             }
```

Figure 12. Call NVIC Reset on Image Update Complete

hal_entry.c

```
2   * Copyright [2020-2024] Renesas Electronics Corporation and/or its affiliates.  All Rights Reserved.
20
21  #include "hal_data.h"
22
23
24  void R_BSP_WarmStart(bsp_warm_start_event_t event);
25
27  * This function is called at various points during the startup process.  This implementation uses the event
32  void R_BSP_WarmStart (bsp_warm_start_event_t event)
33  {
34      if (BSP_WARM_START_RESET == event)
35      {
36  #if BSP_FEATURE_FLASH_LP_VERSION != 0
37
38          /* Enable reading from data flash. */
39          R_FACI_LP->DFLCTL = 1U;
40
41          /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable here, before c
42           * C runtime initialization, should negate the need for a delay since the initialization will typical
43  #endif
44      }
45
46      if (BSP_WARM_START_POST_C == event)
47      {
48          /* C runtime environment and system clocks are setup. */
49
50          /* Configure pins. */
51          R_IOPORT_Open(&g_ioport_ctrl, g_ioport.p_cfg);
52          /* Open the Metrology ICU IRQ module */
53          R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl, &g_external_irq0_cfg);
54          /* Enable Metrology ICU IRQ channel */
55          R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl);
56      }
57  }
```

Figure 13. Start Metrology Process at System Reset

With Dual Bank Swap Instant (without reset), the metrology process is operational at startup and the primary application downloader implements the XMODEM over UART protocol for the image transfer to the MCU program flash.

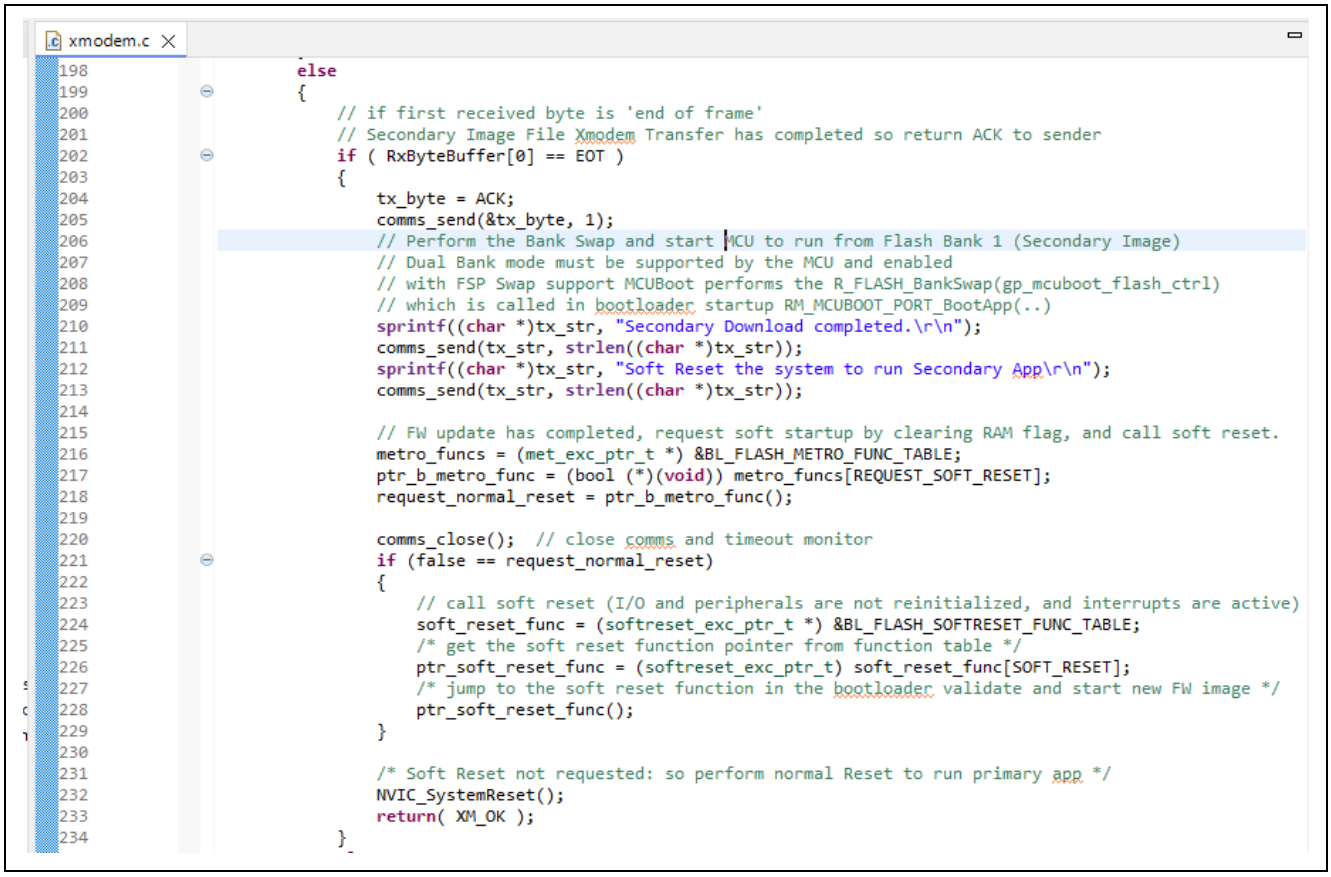


Figure 14. Call MCU Soft Reset on Image Update Complete

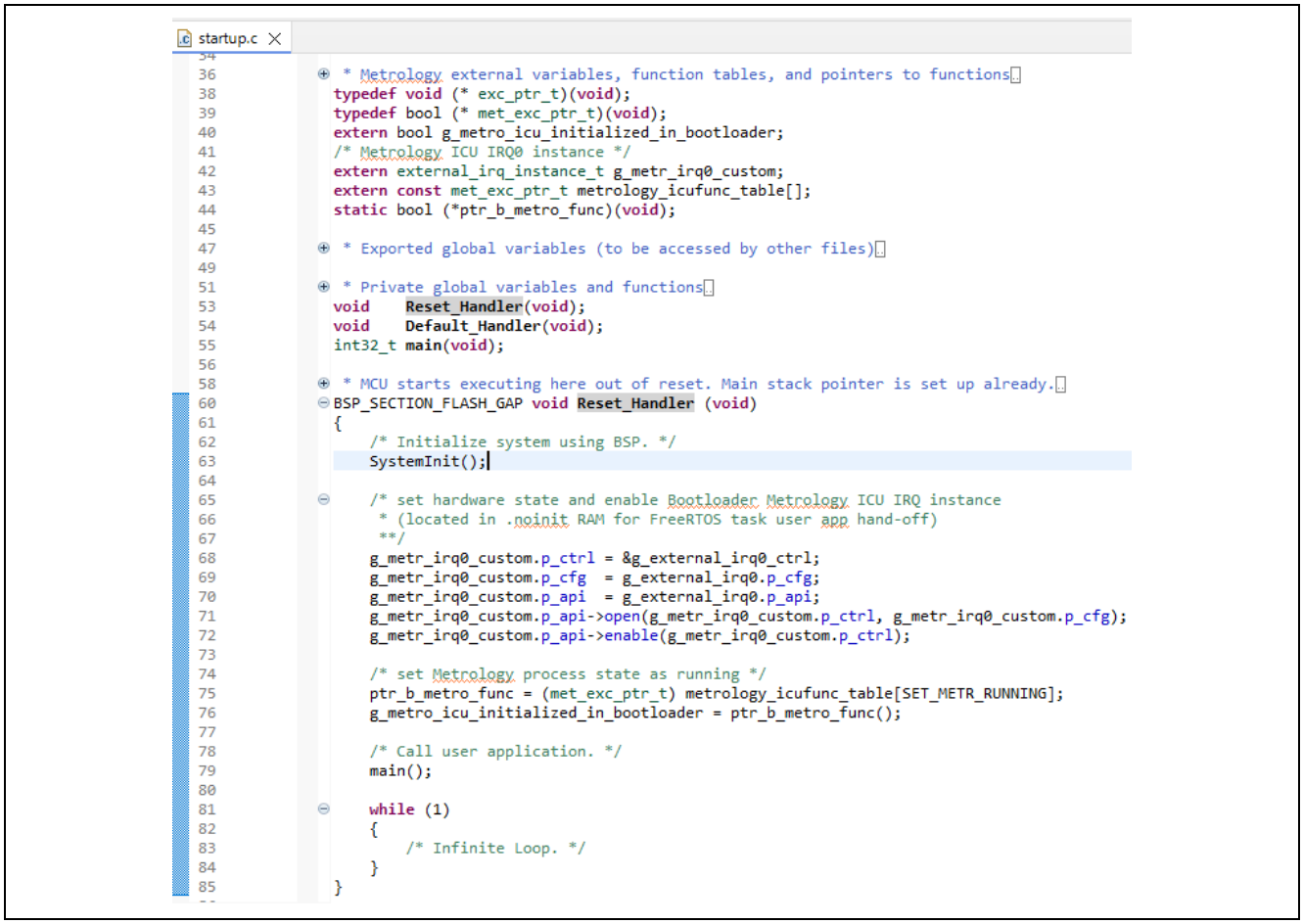
As shown in Figure 16, once the image transfer reaches end of transmission (EOT) then the download is complete and a MCU soft reset is executed to jump to the bootloader to validate the secondary application image. The soft reset is used to preserve (not re-initialize or re-configure) all the I/O and Interrupt resources used by metrology process.

hal_entry.c

```
28  void soft_cpu_restart(void)
29  {
30      /* Soft restart to keep interrupts running for metrology app */
31  #undef MEASURE_TIME
32  #ifdef  MEASURE_TIME
33      /* 946 uSec from soft start to Metrology ICU IRQ instance hand-off to user app
34      R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
35      R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_03, BSP_IO_LEVEL_HIGH);
36  #endif
37
38      /* Optional: Reset SysTick if used  */
39      SysTick->CTRL = 0;
40      SysTick->LOAD = 0;
41      SysTick->VAL  = 0;
42
43      /* flush data and instruction caches */
44      __asm volatile (
45          "DSB                                    \n"
46          "ISB                                    \n"
47          );
48
49      /* Get the address of the bootloader vector table */
50      vector_table = (uint32_t *)__Vectors;
51
52      /* Set the applications vector table. */
53      SCB->VTOR = vector_table[1];
54      /* Set the Main Stack Pointer */
55      __set_MSP(vector_table[0]);
56
57      /* Disable MSP monitoring. */
58      #if BSP_FEATURE_TZ_HAS_TRUSTZONE
59          __set_MSPLIM(0);
60      #elif BSP_FEATURE_BSP_HAS_SP_MON
61          R_MPU_SPMON->SP[0].CTL = 0;
62      #endif
63
64      /* Jump to the Bootloader Reset Handler */
65      pFunction resetHandler = (pFunction)vector_table[1];
66      resetHandler();
67
68      while (1); /* Just in case jump fails */
69  }
```

Figure 15. Bootloader Soft Restart

As shown in Figure 17, the bootloader soft reset does not perform a full hardware reset. It loads the bootloader vector table and sets up the MCU to jump to the bootloader reset handler.



```
startup.c
 * Metrology external variables, function tables, and pointers to functions
typedef void (* exc_ptr_t)(void);
typedef bool (* met_exc_ptr_t)(void);
extern bool g_metro_icu_initialized_in_bootloader;
/* Metrology ICU IRQ0 instance */
extern external_irq_instance_t g_metr_irq0_custom;
extern const met_exc_ptr_t metrology_icufunc_table[];
static bool (*ptr_b_metro_func)(void);

 * Exported global variables (to be accessed by other files)

 * Private global variables and functions
void    Reset_Handler(void);
void    Default_Handler(void);
int32_t main(void);

 * MCU starts executing here out of reset. Main stack pointer is set up already.
BSP_SECTION_FLASH_GAP void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* set hardware state and enable Bootloader Metrology ICU IRQ instance
     * (located in .noinit RAM for FreeRTOS task user app hand-off)
     **/
    g_metr_irq0_custom.p_ctrl = &g_external_irq0_ctrl;
    g_metr_irq0_custom.p_cfg  = g_external_irq0.p_cfg;
    g_metr_irq0_custom.p_api  = g_external_irq0.p_api;
    g_metr_irq0_custom.p_api->open(g_metr_irq0_custom.p_ctrl, g_metr_irq0_custom.p_cfg);
    g_metr_irq0_custom.p_api->enable(g_metr_irq0_custom.p_ctrl);

    /* set Metrology process state as running */
    ptr_b_metro_func = (met_exc_ptr_t) metrology_icufunc_table[SET_METR_RUNNING];
    g_metro_icu_initialized_in_bootloader = ptr_b_metro_func();

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

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

Figure 16. Bootloader Reset Handler

In Figure 18, the bootloader reset handler calls a special version of SystemInit( ) that only initializes I/O that is required by the bootloader. Here, the user must omit the I/O initialization for any resource that the metrology process will require in order to continue to operate. Observe in the reset handler, the metrology resources (ICU IRQ instance) are registered for hand-off to the user application.

```
    /* Initialize C runtime environment. */
    /* Zero out BSS */
    memset(&__bss_start__, 0U, ((uint32_t) &__bss_end__ - (uint32_t) &__bss_start__));
    /* Copy initialized RAM data from ROM to RAM. */
    memcpy(&__data_start__, &__etext, ((uint32_t) &__data_end__ - (uint32_t) &__data_start__));

    /* Initialize static constructors */
    int32_t count = __init_array_end - __init_array_start;
    for (int32_t i = 0; i < count; i++)
    {
        __init_array_start[i]();
    }
    // BSP_CFG_C_RUNTIME_INIT

    /* Initialize SystemCoreClock variable. */
    SystemCoreClockUpdate();

#if BSP_FEATURE_RTC_IS_AVAILABLE || BSP_FEATURE_RTC_HAS_TCEN || BSP_FEATURE_SYSC_HAS_VBTICTLR
    /* For TZ project, it should be called by the secure application, whether RTC module is to be configured as secure or not. */
 #if !BSP_TZ_NONSECURE_BUILD && !BSP_CFG_BOOT_IMAGE && !BSP_CFG_SKIP_INIT

    /* Perform RTC reset sequence to avoid unintended operation. */
    R_BSP_Init_RTC();
 #endif
#endif

    /* Instead of calling BSP_WARM_START_POST_C, call IO open with custom pin cfg that excludes metrology used I/O and peripherals */
    R_IOPORT_Open (&g_ioport_ctrl, &IOPORT_CFG_METROAPP);

    /* Initialize ELC events that will be used to trigger NVIC interrupts. */
    bsp_irq_cfg();

    /* Call any BSP specific code. No arguments are needed so NULL is sent. */
    bsp_init(NULL);
}
```

system.c

```
/**
 * The Metrology app custom pin configuration excludes Green LED P308, VCOM Debug port (SWDIO) pins P108, P300.
 * Remove any peripherals or port pins that are required to be used by the continuous operating Metrology App
 */
static const ioport_pin_cfg_t g_bsp_metroapp_pin_cfg_data[] =
{
//      { .pin = BSP_IO_PORT_01_PIN_08, .pin_cfg = ((uint32_t) IOPORT_CFG_PERIPHERAL_PIN | (uint32_t) IOPORT_PERIPHERAL_DEBUG) },
//      { .pin = BSP_IO_PORT_03_PIN_00, .pin_cfg = ((uint32_t) IOPORT_CFG_PERIPHERAL_PIN | (uint32_t) IOPORT_PERIPHERAL_DEBUG) },
//      { .pin = BSP_IO_PORT_03_PIN_06, .pin_cfg = ((uint32_t) IOPORT_CFG_IRQ_ENABLE | (uint32_t) IOPORT_CFG_PORT_DIRECTION_INPUT) },
//      { .pin = BSP_IO_PORT_03_PIN_08, .pin_cfg = ((uint32_t) IOPORT_CFG_PORT_DIRECTION_OUTPUT | (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW) },
//      { .pin = BSP_IO_PORT_03_PIN_09, .pin_cfg = ((uint32_t) IOPORT_CFG_PORT_DIRECTION_OUTPUT | (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW) },
        { .pin = BSP_IO_PORT_01_PIN_03, .pin_cfg = ((uint32_t) IOPORT_CFG_PORT_DIRECTION_OUTPUT | (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW) },
};

static const ioport_cfg_t g_bsp_metroapp_pin_cfg = { .number_of_pins =
        sizeof(g_bsp_metroapp_pin_cfg_data) / sizeof(ioport_pin_cfg_t), .p_pin_cfg_data =
        &g_bsp_metroapp_pin_cfg_data[0], };

#define IOPORT_CFG_METROAPP g_bsp_metroapp_pin_cfg
```

Figure 17. Bootloader SystemInit( )

In Figure 19, the custom version of SystemInit( ) initializes I/O and peripherals that are required by the bootloader. Note that Green LED P308, Pin P306 Signal Input, VCOM Debug Port Pins, and other resources are not re-configured. They are configured only on a NVIC / Hard Reset (not configured on the Soft Start requested by the Downloader).

As observed in Figure 18, once the metrology resources (in this case, the ICU IRQ instance) are registered for hand-off to the user application, the metrology process state is set as running and the user application reset handler is called.

startup.c

```
70
72          * MCU starts executing here out of reset. Main stack pointer is set up already.
74      BSP_SECTION_FLASH_GAP void Reset_Handler (void)
75      {
77           * Get Metrology operation state
79          metro_funcs = (met_exc_ptr_t *) &BL_FLASH_METRO_ICUFUNC_TABLE;
80          ptr_b_metro_func = (bool (*)(void)) metro_funcs[READ_METR_STATE];
81          g_metro_icu_initialized_in_bootloader = ptr_b_metro_func();
82          if (METR_RUNNING == g_metro_icu_initialized_in_bootloader)
83          {
84              /* Metrology is initialized in bootloader */
85              metro_icu = (met_icu_ptr_t *) &BL_FLASH_METRO_ICU_TABLE;
86              ptr_metro_icu = (external_irq_instance_t (*)(void)) metro_icu[GET_ICU_INSTANCE];
87              /* get the bootloader Metrology instance */
88              g_bl_external_metr_irq0 = ptr_metro_icu();
89          }
90
91          /* check for MCU soft reset request (firmware update occurred)*/
92          metro_funcs = (met_exc_ptr_t *) &BL_FLASH_METRO_FUNC_TABLE;
93          ptr_b_metro_func = metro_funcs[READ_RESET_MODE];
94          g_normal_reset = ptr_b_metro_func();
95          if (true == g_normal_reset)
96          {
97              /* Initialize system using BSP. */
98              SystemInit();
99          }
100         else
101         {
102             /* Initialize system with Soft Reset to keep metrology app running */
103             SoftSystemInit();
104         }
105
106         if ((METR_RUNNING == g_metro_icu_initialized_in_bootloader) &&
107              g_bl_external_metr_irq0.p_ctrl && g_bl_external_metr_irq0.p_cfg)
108         {
109             /* register the Metrology IRQ callback with the bootloader
110              * Metrology ICU instance and set the ICU hardware state that does not persist from the bootloader
111             err = metro_irq_instance_register(&g_bl_external_metr_irq0,
112                                               &g_external_irq0_ctrl,
113                                               &g_external_irq0_cfg,
114                                               irq0_callback);
115
116         }
117
118         /* Call user application. */
119         main();
```

Figure 18. User Application Reset Handler

In Figure 20, the user application `Reset_Handler()` determines whether a soft reset was requested by an image update. If a soft reset is in process then `SoftSystemInit()` is called to keep the metrology application running. Otherwise, `SystemInit()` is called to initialize all MCU I/O and Peripherals as required by the user application.

The function `SoftSystemInit()` uses a customized version of `g_ioport_ctrl`, IOPORT_CFG_METROAPP that excludes I/O and Peripherals that are resources used by the metrology process.

Next, the metrology process instance (ICU IRQ) is registered to set the hardware state and callback function that does not persist through the bootloader hand-off. Finally, the user application starts by calling the function `main()`.

metro_ra_mcuboot_ra2a2_dualbank.bld

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ddscBootloader>
    <symbols>
        <symbol name="FSP_BOOTABLE_IMAGE" value="0x00000001"/>
        <symbol name="RAM_NS_START" value="0x2000C000"/>
        <symbol name="BL_FLASH_SOFTRESET_FUNC_TABLE" value="0x00003004"/>
        <symbol name="BL_FLASH_METRO_ICU_TABLE" value="0x00003008"/>
        <symbol name="BL_FLASH_METRO_ICUFUNC_TABLE" value="0x0000300c"/>
        <symbol name="BL_FLASH_METRO_FUNC_TABLE" value="0x00003018"/>
        <symbol name="FLASH_NS_START" value="0x0003F800"/>
        <symbol name="FLASH_IMAGE_END" value="0x0003F800"/>
        <symbol name="FLASH_NS_IMAGE_START" value="0x0003F800"/>
        <symbol name="FLASH_NSC_START" value="0x0003F800"/>
        <symbol name="FLASH_IMAGE_START_FROM_MMF_REGION" value="0x00000000"/>
        <symbol name="MEMORY_MIRROR_REGION_START" value="0x02000000"/>
        <symbol name="RAM_NSC_START" value="0x2000C000"/>
        <symbol name="FLASH_IMAGE_LENGTH" value="0x0003B600"/>
        <symbol name="XIP_SECONDARY_FLASH_IMAGE_END" value="0x0007B000"/>
        <symbol name="XIP_SECONDARY_FLASH_IMAGE_START" value="0x0003FA00"/>
        <symbol name="FLASH_IMAGE_START" value="0x00004200"/>
        <symbol name="FLASH_IMAGE_START" value="0x0003F800" security="n"/>
        <symbol name="FSP_BOOTABLE_IMAGE" value="0x00000001" security="n"/>
        <symbol name="FLASH_IMAGE_LENGTH" value="0x0003F800" security="n"/>
    </symbols>
    <images>
        <image path="${BuildArtifactFileBaseName}.bin.signed">python ${workspace
        <image path="${BuildArtifactFileBaseName}.bin.signed" security="n">pytho
    </images>
</ddscBootloader>
```

Figure 19. Bootloader Metrology Function Table Symbols

As mentioned previously in Figure 13, Bank Swap Instant operation flow sets RAM flags after the Secondary Image is programmed to flash and checks the RAM flags to skip the initialization of Inputs/Outputs and Peripherals on MCU soft reset. The RAM flags are stored in a RAM Section ".noinit" of the MCU.

The metrology process functions access the flags in RAM by using function tables defined in the bootloader metrology_utils.c. The linker for the user applications (primary and secondary) uses the bootloader file `metro_ra_mcuboot_ra2a2_dualbank/Debug/metro_ra_mcuboot_ra2a2_dualbank.bld` to map the symbols and addresses of the metrology functions to be used by user applications.

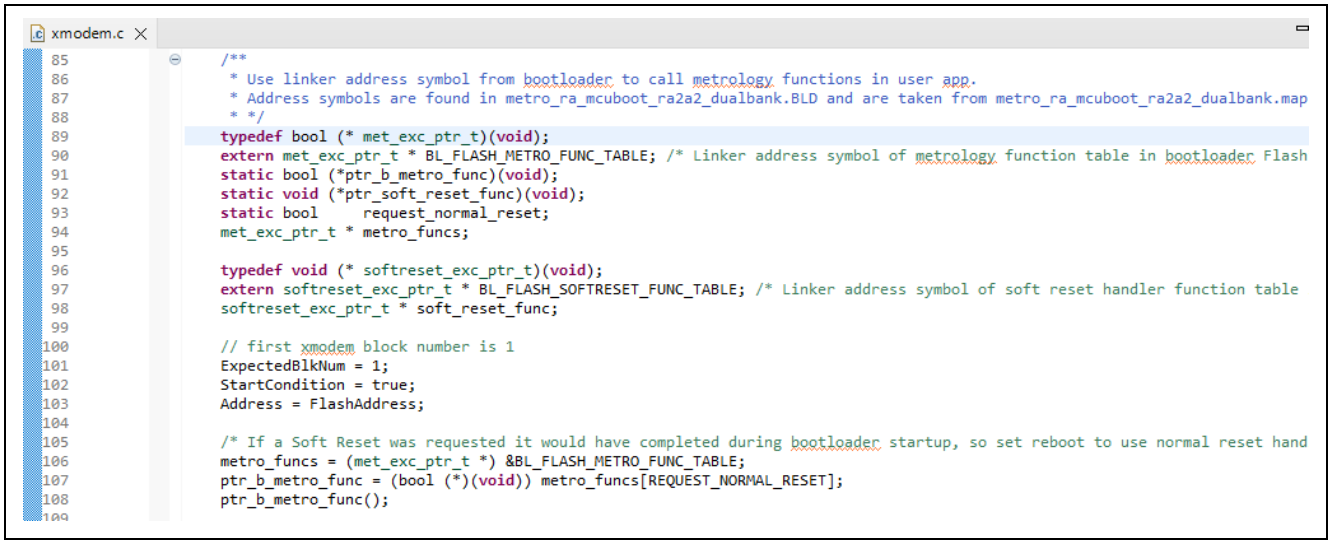


Figure 20. User Application Uses Metrology Functions

Observe in Figure 21 Bootloader Metrology Function Table Symbols and Figure 22 an example of how the user application uses the metrology functions by accessing through function tables.

For more details, see the source code implementation for the projects in `ekra2a2_mcuboot_dualbank_instant` which provides a bootloader, primary application, and secondary application.

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 RA2A2 running in Non-TrustZone mode. Here we will focus on the Dual Bank update with reset to detail the process. The other included application project Dual Bank update with Instant mode uses a similar process.

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 e² studio and start a new C/C++ Project. Click **File** > **New** > **C/C++ Project.**

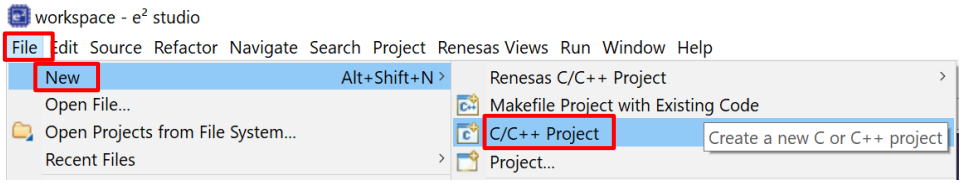


Figure 21. Start a New Project

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

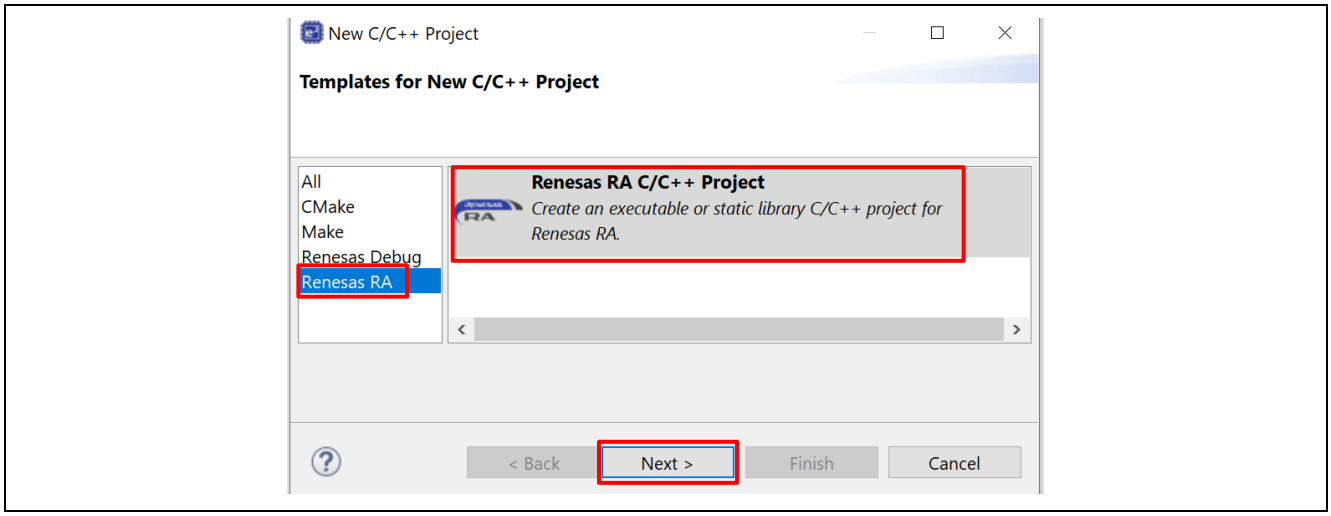


Figure 22. Choose Renesas RA C/C++ Project

3. Provide the project name **`ra_mcuboot_ra2a2_dualbank`** in the next screen. Click **Next**.
4. In the next screen, choose **EK-RA2A2** for **Board** and click **Next**.

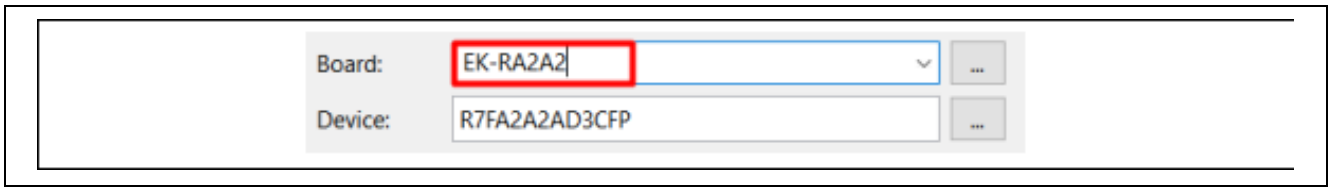


Figure 23. Select the Board

5. When the following screen appears, select Flat (Non-TrustZone) Project

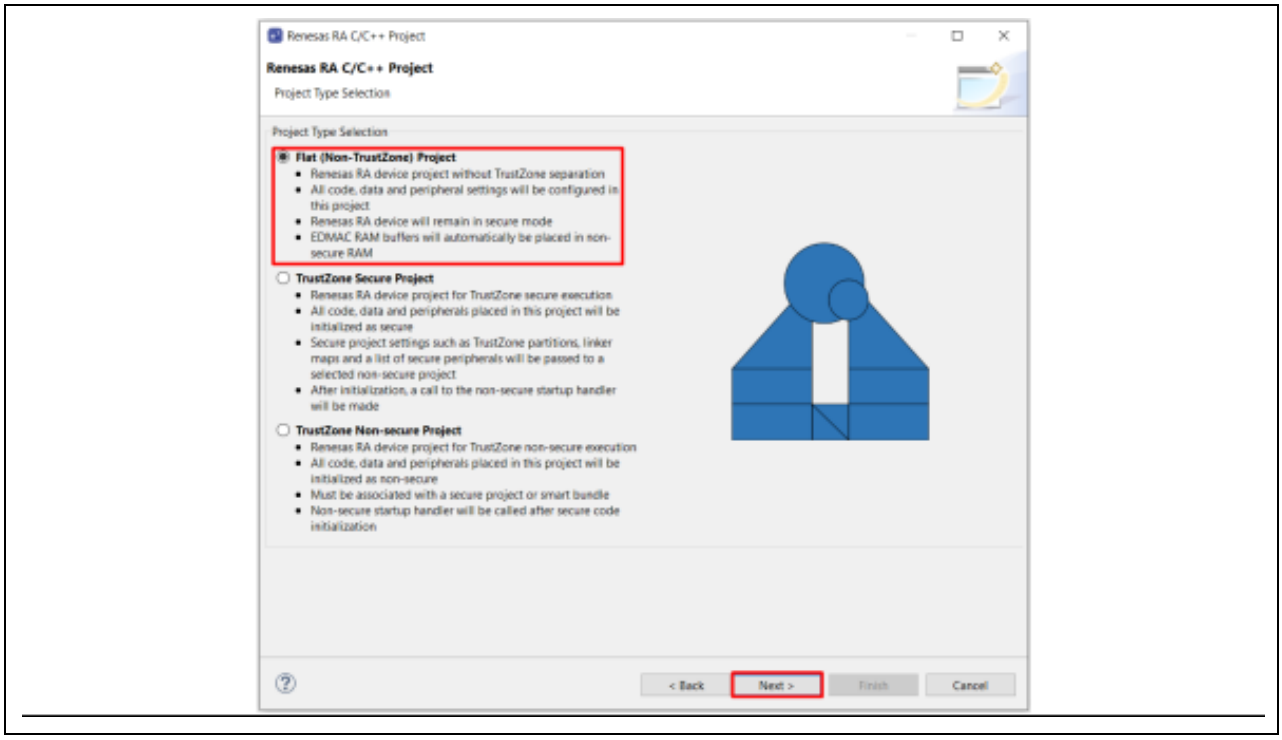


Figure 24. Choose Flat Project as Project Type

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

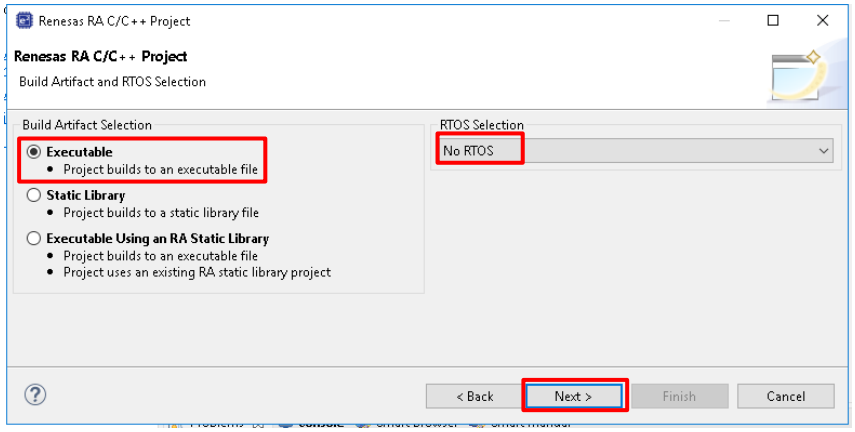


Figure 25. Choose to Build Executable and No RTOS

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

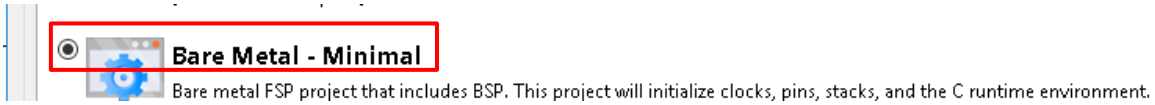


Figure 26. Choose the Project Template

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

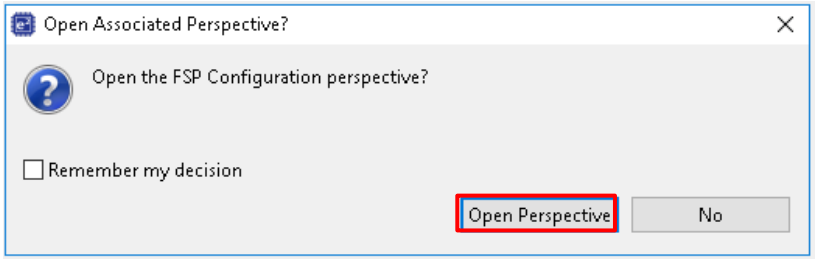


Figure 27. Choose Open the FSP Configuration Perspective

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

9. Select the **Pins** tab and uncheck **Generate data** for **RA2A2 EK.**

Figure 28. Uncheck Generate data for RA2A2 EK Pin Configuration

Use the pull-down menu to switch from **RA2A2 EK** to **R7FA2A2AD3CFP.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 **RA2A2 EK** configuration. This also reduces some memory usage for the bootloader project.

Figure 29. Select g_bsp_pin_cfg and Generate data g_bsp_pin_cfg

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

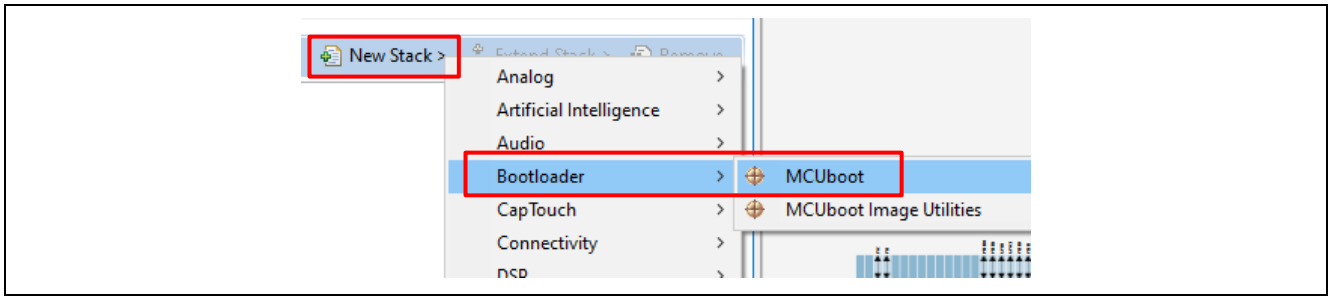


Figure 30. Add the MCUboot Port

11. 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**.

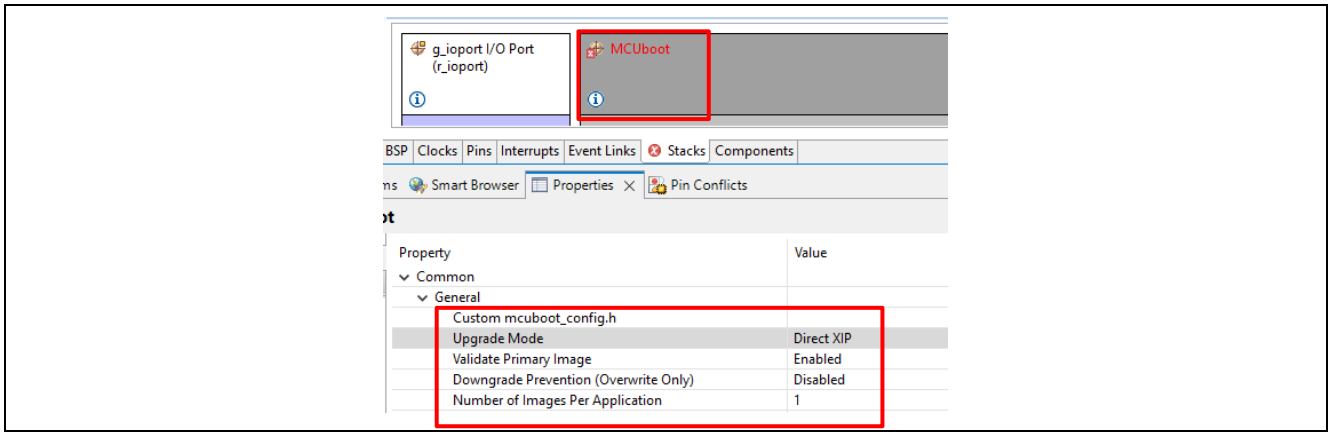


Figure 31. General Configuration for MCUboot Module

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-RA2A2, the default memory for the code flash dual bank mode is shown in Figure 32. This Bank Programming Mode memory map is used for the example bootloader design.

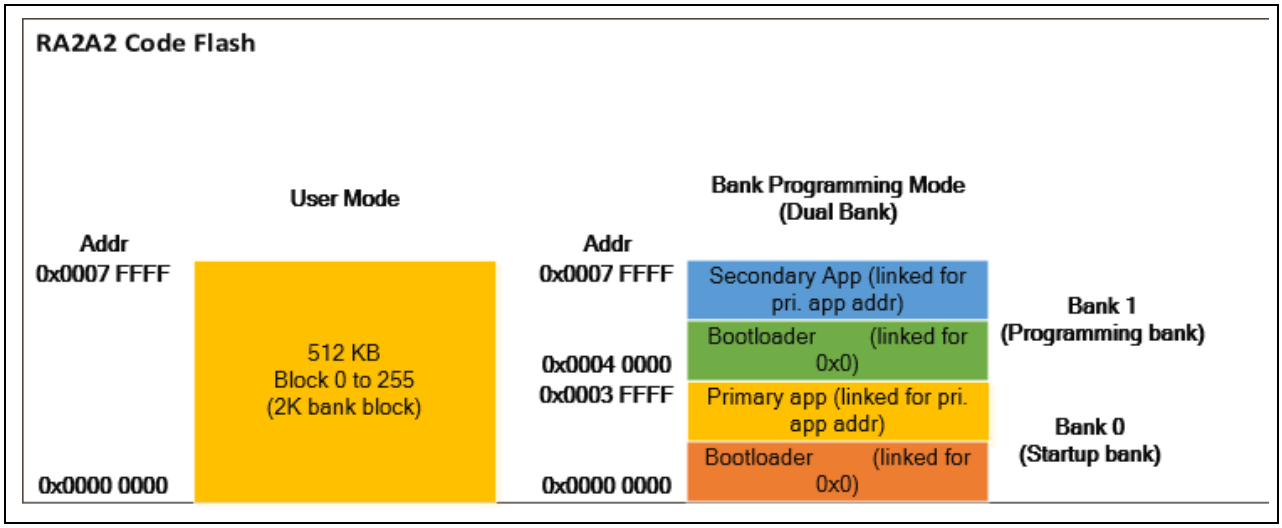


Figure 32. MCUboot Bank Programming Mode Memory Map

Follow Figure 34 to update the Properties for the Flash Layout to match with the MCUboot memory map, as shown in Figure 35.

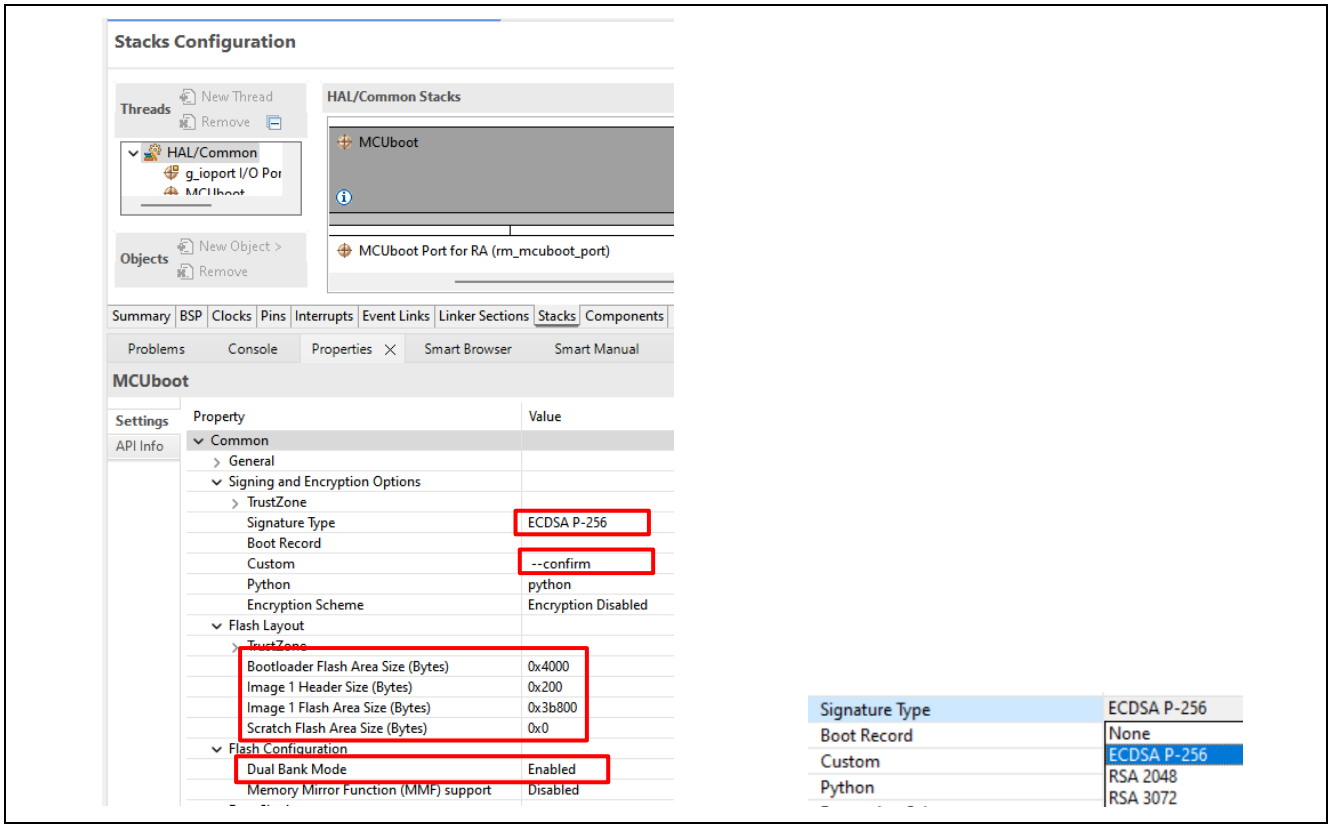


Figure 33. Configure the Flash Layout and Signing Options

Explanation of the Above Configurations:

- **Bootloader Flash Area**: Size of the flash area allocated for the bootloader, with a boundary of 0x4000.
- **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 RA2A2. For the RA2A2, 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 RA2A2, this size needs to be on a boundary of 0x800 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:** Typically, this option is not used as the time used in the authentication is much longer than the ECDSA P-256.
- 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.
- **Encryption Scheme:** Encryption is disabled in this example implementation.

4.3 Enable Dual Bank Swap Support

Click on the MCUboot stack > Properties > Flash Configuration. Then, enable the **Dual Bank Mode**, as shown in Figure 35.

4.4 Configure the TinyCrypt (S/W Only) Module and the Flash Driver

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

1. Click on **Add Crypto Stack** and select the **MCUboot TinyCrypt** (S/W Only) module..

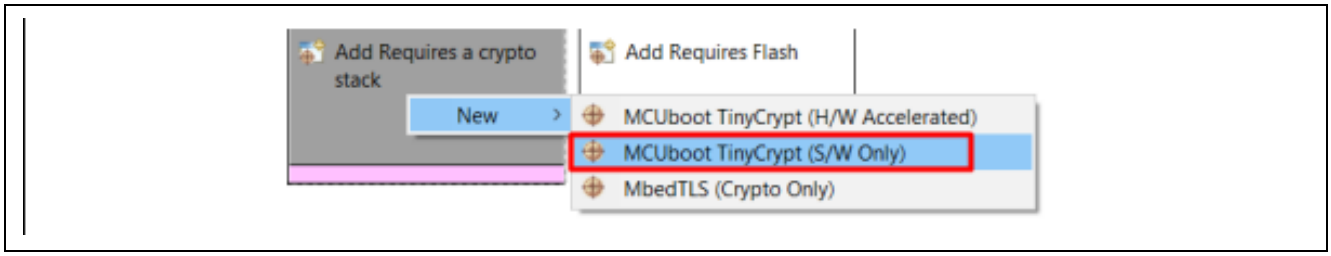


Figure 34. Select TinyCrypt S/W Only Module

2. If the user is creating a bootloader with signature verification support, then the **ASN.1 Parser** stack and the MCUboot Example Keys stack will be required.

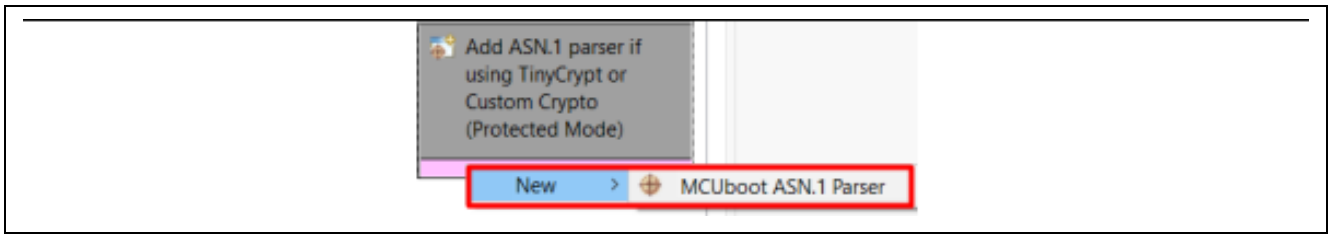


Figure 35. Add the ASN.1 Parser

Click on the Add [Optional] **Add Example Keys** stack and choose New > **MCUboot Example Keys** [NOT FOR PRODUCTION]

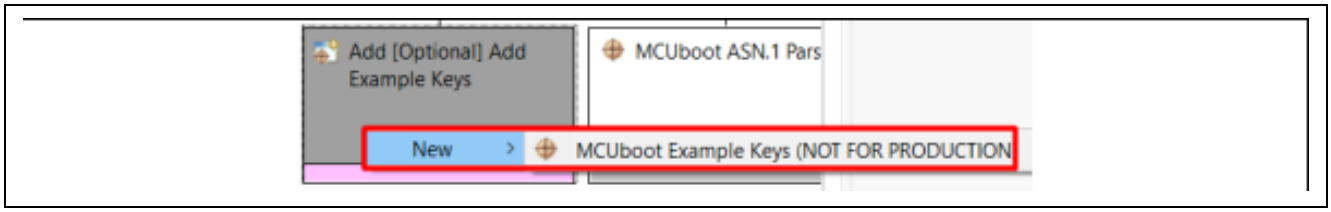


Figure 36. Add the Example Image Signing Key

Note: The example key is open to public access from MCUboot port, customers should not use them for production purposes. Customers can follow the procedure in section 3.6.1 in Application Project R11AN0516 to create and use customized signing key.

3. Click on **Add Requires Flash** stack and select **Flash (r_flash_lp)** stack to add the Flash Driver.

Click on the Flash (r_flash_lp) stack > Properties > **Instant Bank Swap** > Instant or After Reset.

Set as shown in Figure 39 as required for the bootloader project.

Set to **After Reset** means that an MCU reset is required for the Bank Swap to occur.

Instant means that the Bank Swap happens without an MCU reset.

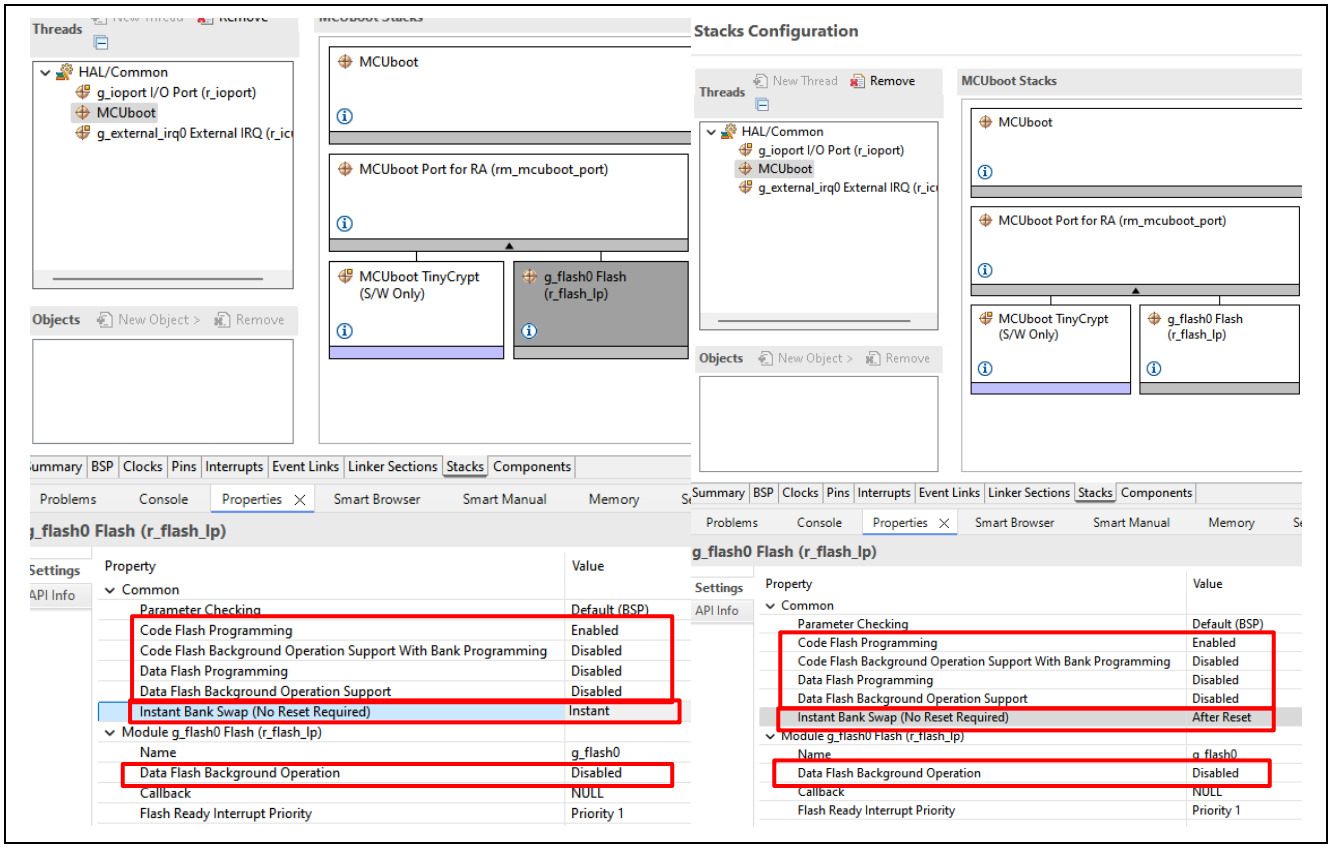


Figure 37. Set Flash Bank Swap Mode

4. 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.

5. Set up the Stack and Heap used by the bootloader based on the authentication mode. Set the following values in the **BSP** tab. Update the BSP > **Main Stack Size** to 0x2000 and **Heap Size** to 0x400.

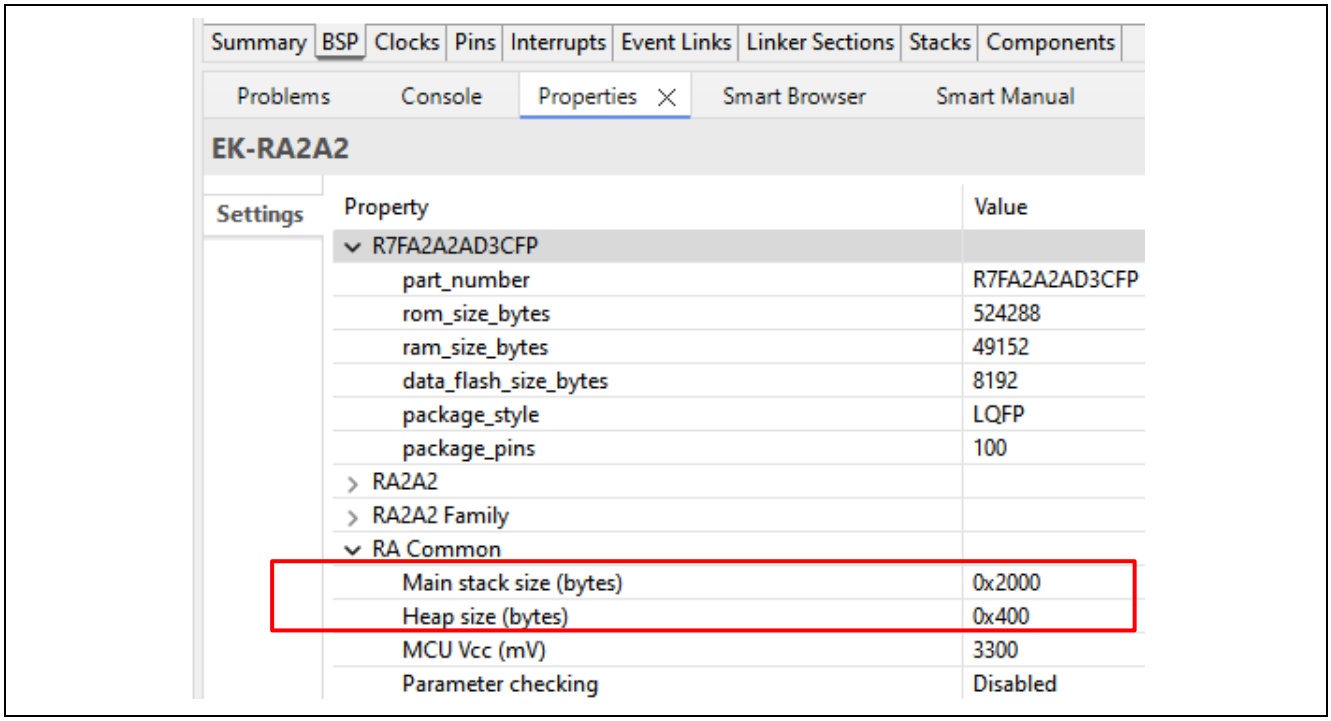


Figure 38. Configure the BSP Stack and Heap Usage

4.5 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();
```

4.6 Compile the Bootloader Project

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

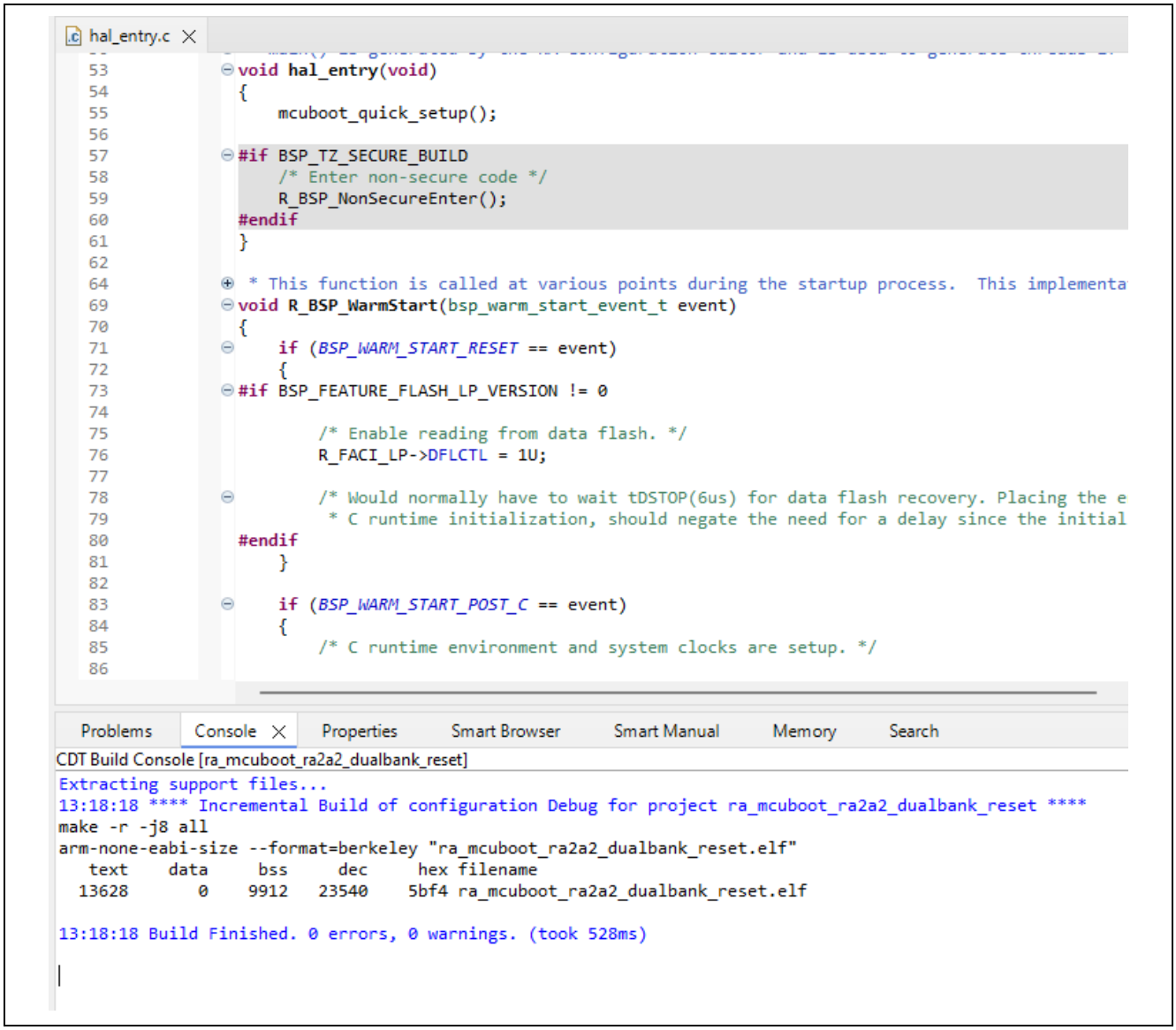


Figure 39. Compile the Bootloader ra_mcuboot_ra2a2_dualbank_reset

4.7 Configure the Python Signing Environment

Signing the application image can be done using a post-build step in e² studio, using the image signing tool `Imgtool.py`, which is included with MCUboot. This tool is integrated as a post-build tool in e² 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_ra2a2_dualbank_reset>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.

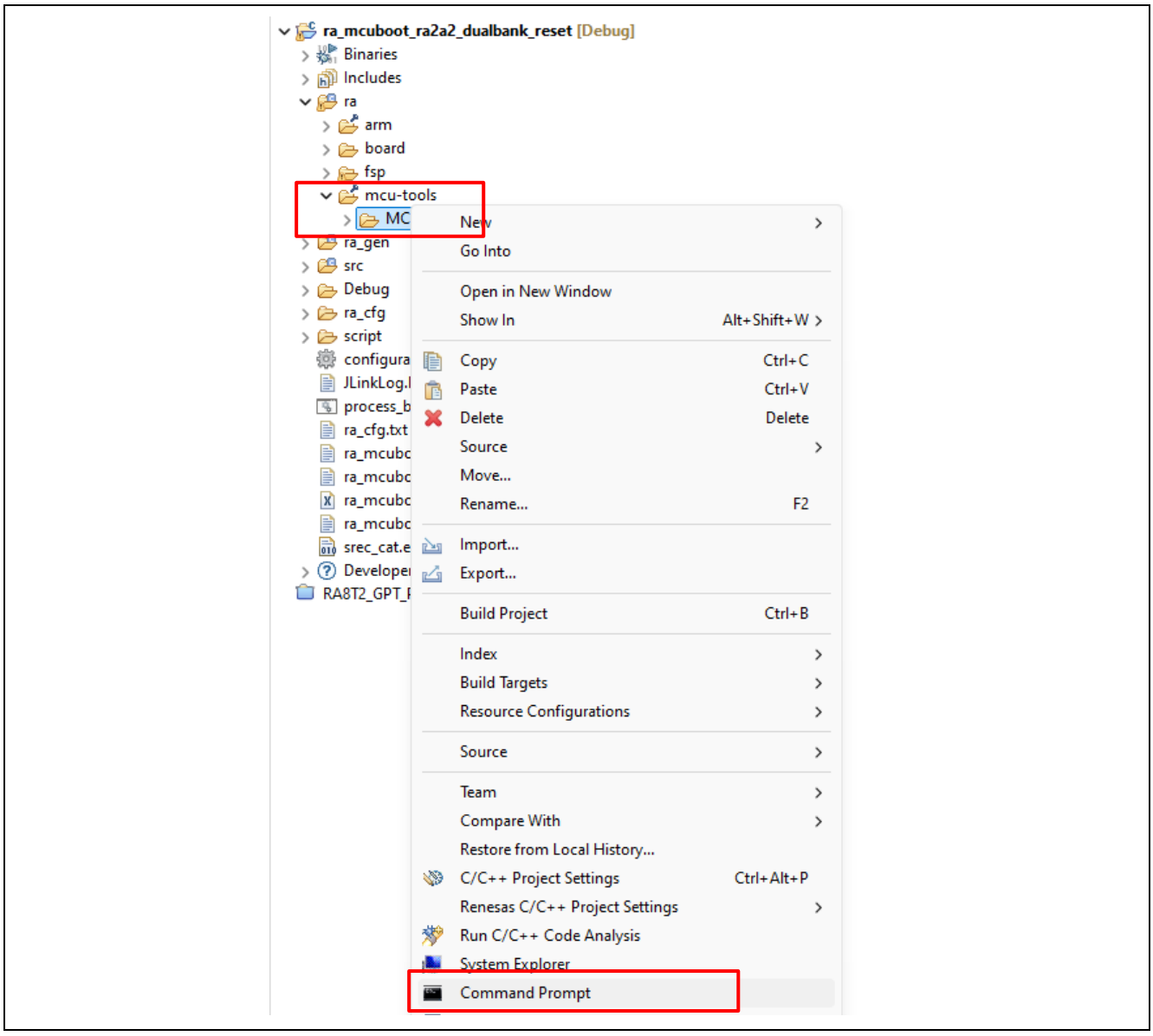


Figure 40. Open the Command Prompt

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
```

or use the alternative command below when the command above does not install the dependencies:

python -m pip install --user -r scripts/requirements.txt

This will verify and install any dependencies that are required.

Review of the Signing Command

The signing command for the application image will be automatically generated when the bootloader is compiled. In the **Project Explorer**, open the `ra_mcuboot_ra2a2_dualbank_reset\debug\ra_mcuboot_ra2a2_dualbank_reset.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.

```
<image path="${BuildArtifactFileBaseName}.bin.signed">python
${workspace_loc:ra_mcuboot_ra2a2_dualbank_reset}/ra/fsp/src/rm_mcuboot_port/rm_mcuboot_port_sign.py
sign --header-size 0x200 --align 8 --max-align 8 --slot-size 0x3b800 --max-sectors 119
--confirm --pad-header ${BuildArtifactFileName} ${BuildArtifactFileBaseName}.bin.signed</image>
```

Figure 41. Signing Command in the .bld File

4.8 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_ra2a2_dualbank_reset_offset.srec`, which offsets the bootloader offset to the RA2A2 flash linear mode upper bank address at 0x40000.

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 sections 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 0x40000.

```
srec_cat Debug\ra_mcuboot_ra2a2_dualbank_reset.srec -crop 0 0x40000 -offset 0x40000 -o
ra_mcuboot_ra2a2_dualbank_reset_offset.srec
```

Figure 42. 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 `RA2_Secure_Bootloader_Dualbank.zip` and copy `\ra_mcuboot_ra2a2_dualbank_reset\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.

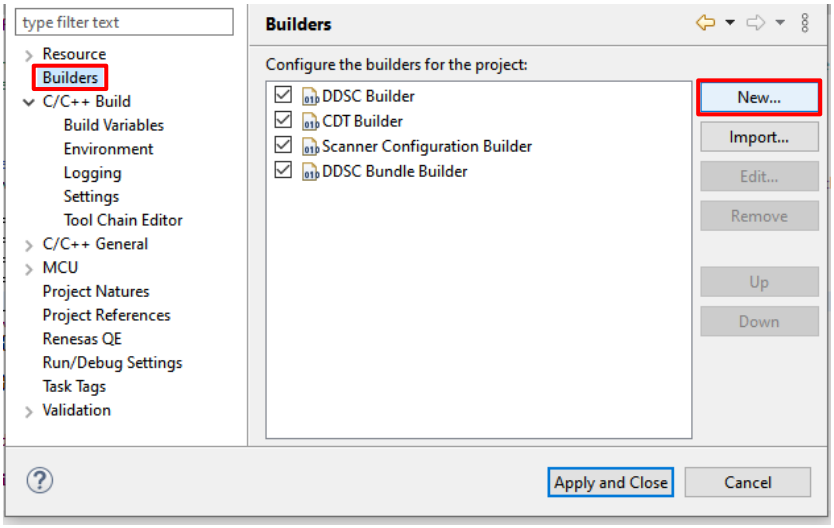


Figure 43. Create a New Custom Builder Entry

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

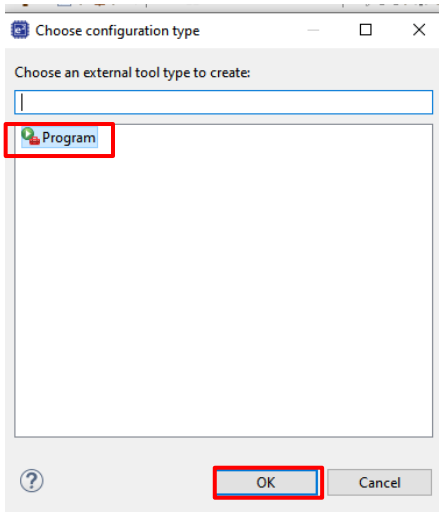


Figure 44. Select the Type of the Builder as Program

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**.

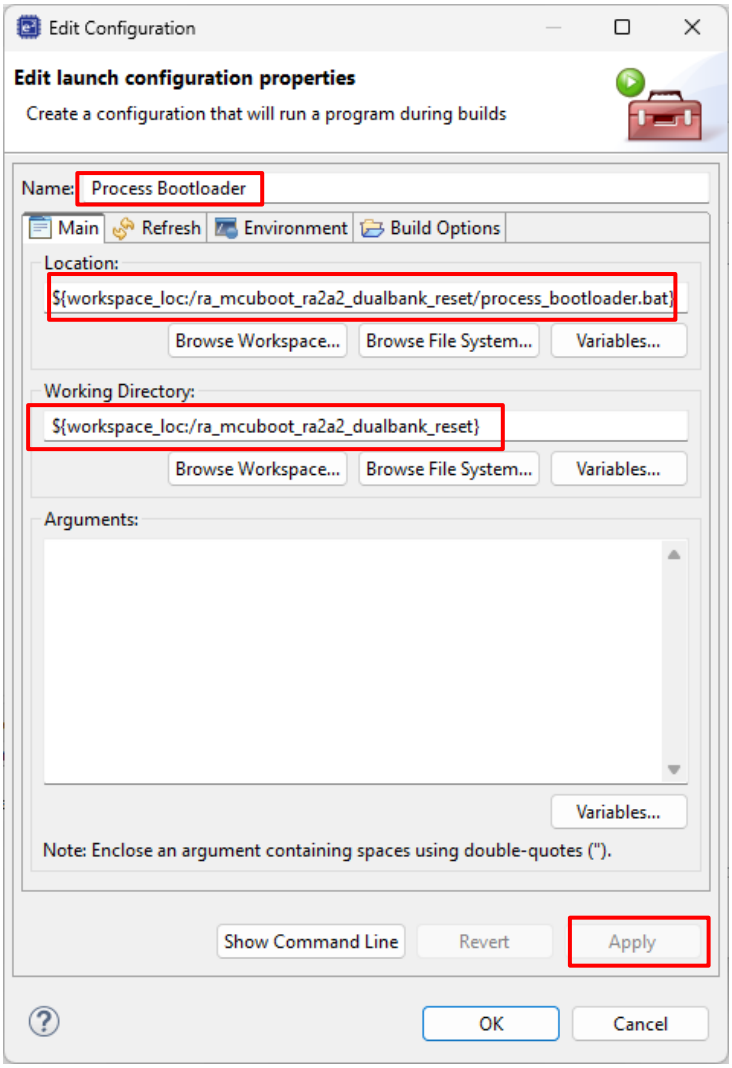


Figure 45. Configure the Custom Builder

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

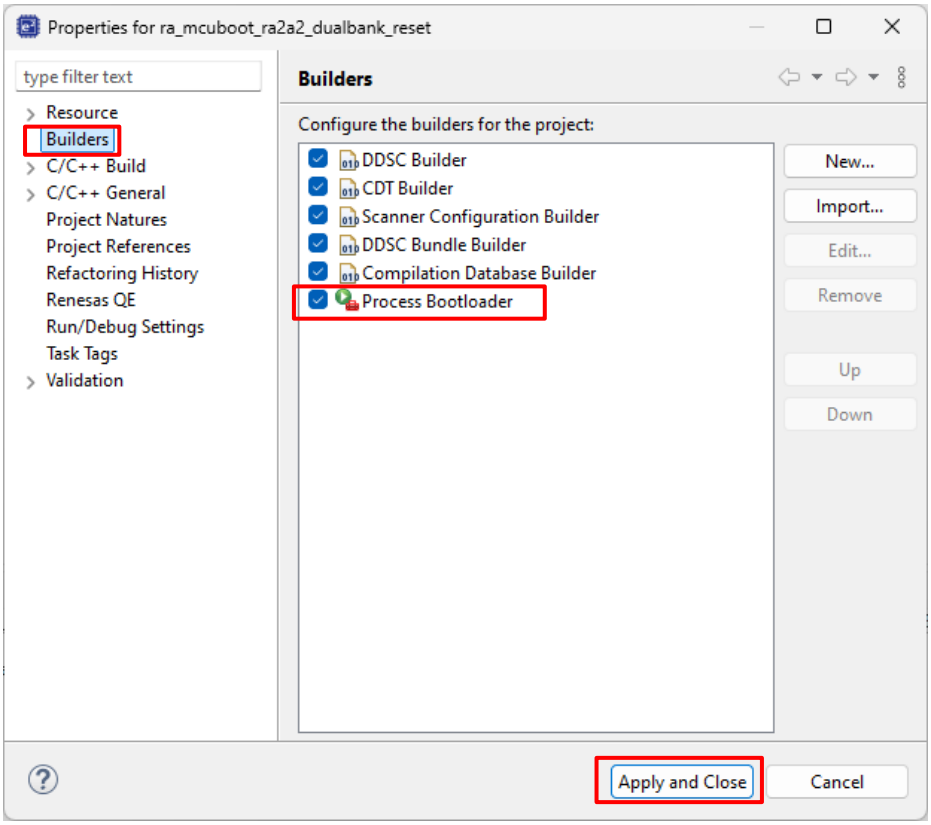


Figure 46. Custom Builder

6. Recompile the bootloader project and notice that `ra_mcuboot_ra2a2_dualbank_reset_offset.srec` is created under the bootloader project root directory.

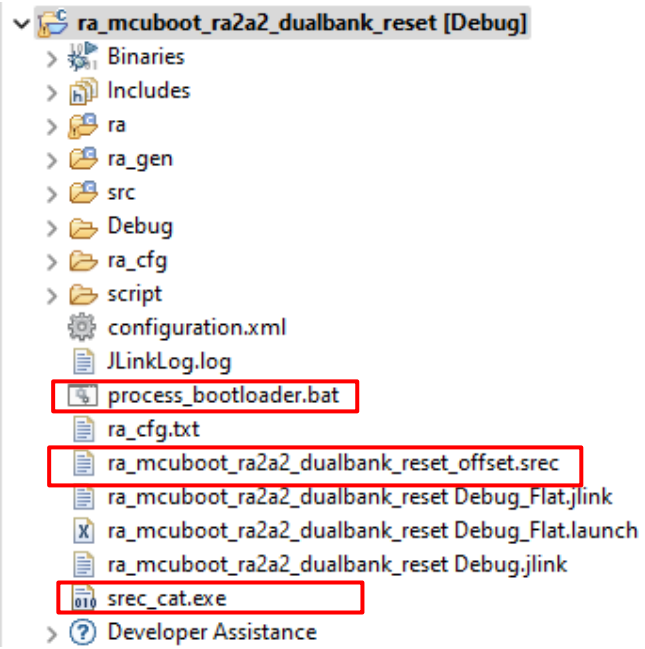


Figure 47. Rebuild the Bootloader with the Custom Builder

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. The accompanying application projects demonstrate how to download a new application using XMODEM over the UART interface as an example. 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 of the application project. When using the bootloader with a different application project, the **Image 1 Flash Area Size** property should be adjusted accordingly.

It is beyond the scope of this application note to fully detail how to integrate a bootloader. Instead, the steps to configure the primary application to use the bootloader for the included project `ekra2a2_mcuboot_dualbank_reset` will be covered. Import the three projects under folder `\ekra2a2_mcuboot_dualbank_reset` to the workspace.

Right-click on the application project folder `app_primary_uart_reset` in the **Project Explorer** and select **Properties**. Select **C/C++ Build > Build Variables**, click **Edit** and for the **Variable name**: **BootloaderDataFile**. Notice that the **Type** is **File** and the path to the `.bld` file for the bootloader project **`ra_mcuboot_ra2a2_dualbank_reset`**:

- The Bootloader data file path is: `${workspace_loc:ra_mcuboot_ra2a2_dualbank_reset}/Debug/ra_mcuboot_ra2a2_dualbank_reset.bld` for the build variable.

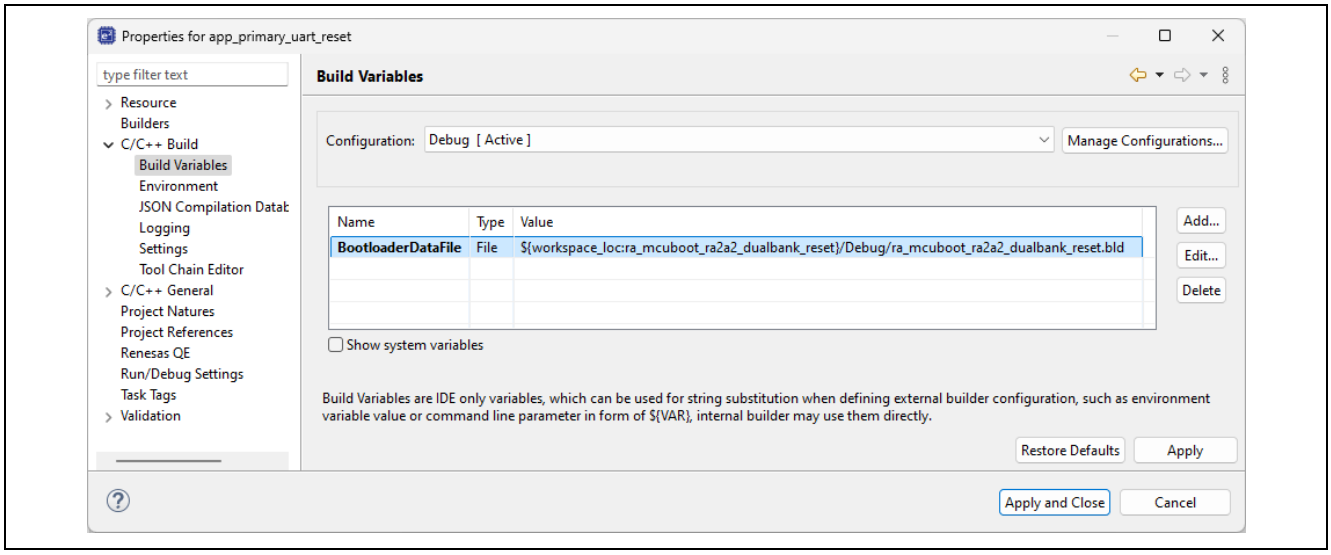


Figure 48. The Build Variable configured to Use the Bootloader

After examining the settings then Click **Cancel**, at the bottom of the screen to exit the Build Variables.

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 should 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_uart_reset`, under **Environment**, click **Edit** and for MCUBOOT_IMAGE_VERSION. Note for the primary application the image version is 1.0.0.

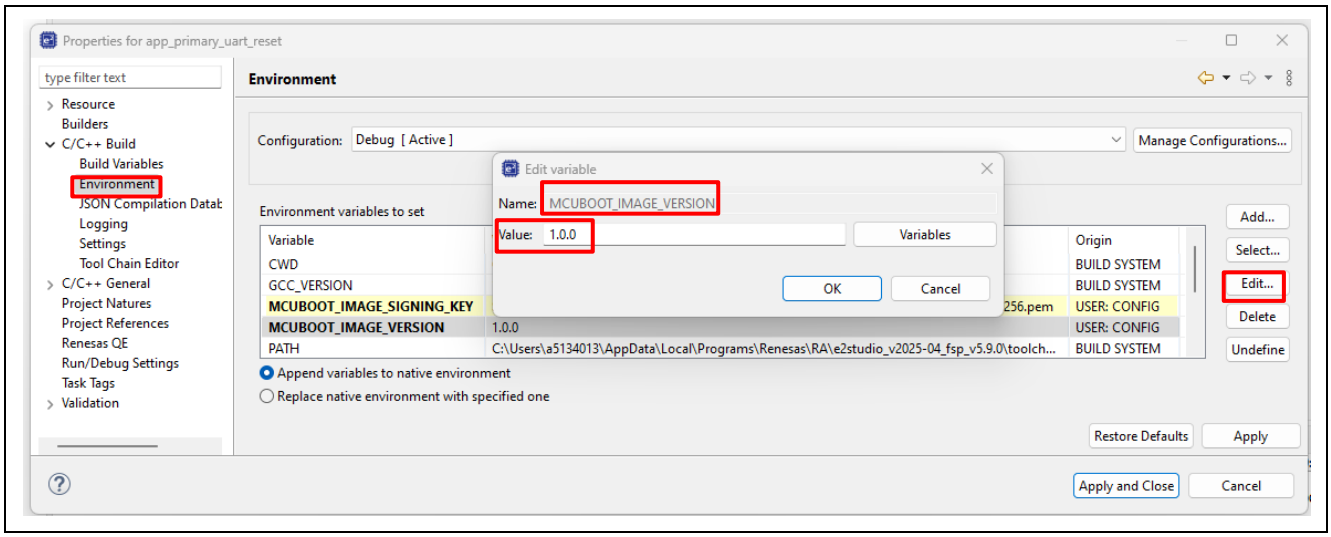


Figure 49. Configure the Application Version

Similarly, edit the variable for MCUBOOT_IMAGE_SIGNING_KEY.

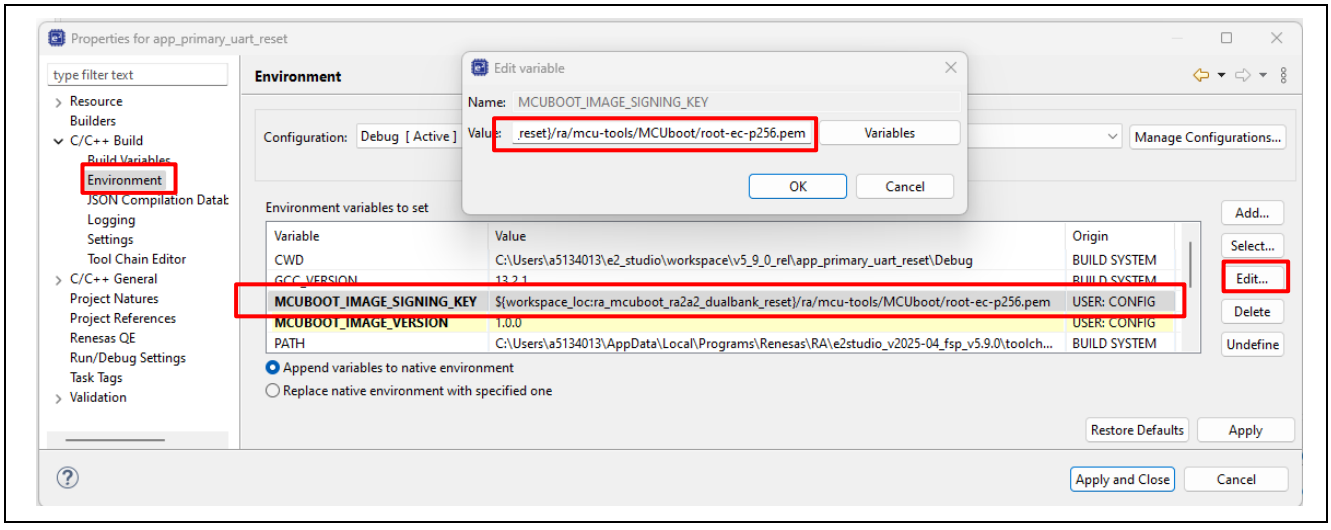


Figure 50. Configure the Private Signing Key

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 projects. 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 53 is the result of the above configuration. If any changes are made then Click **Apply and Close, else Click Cancel to exit**.

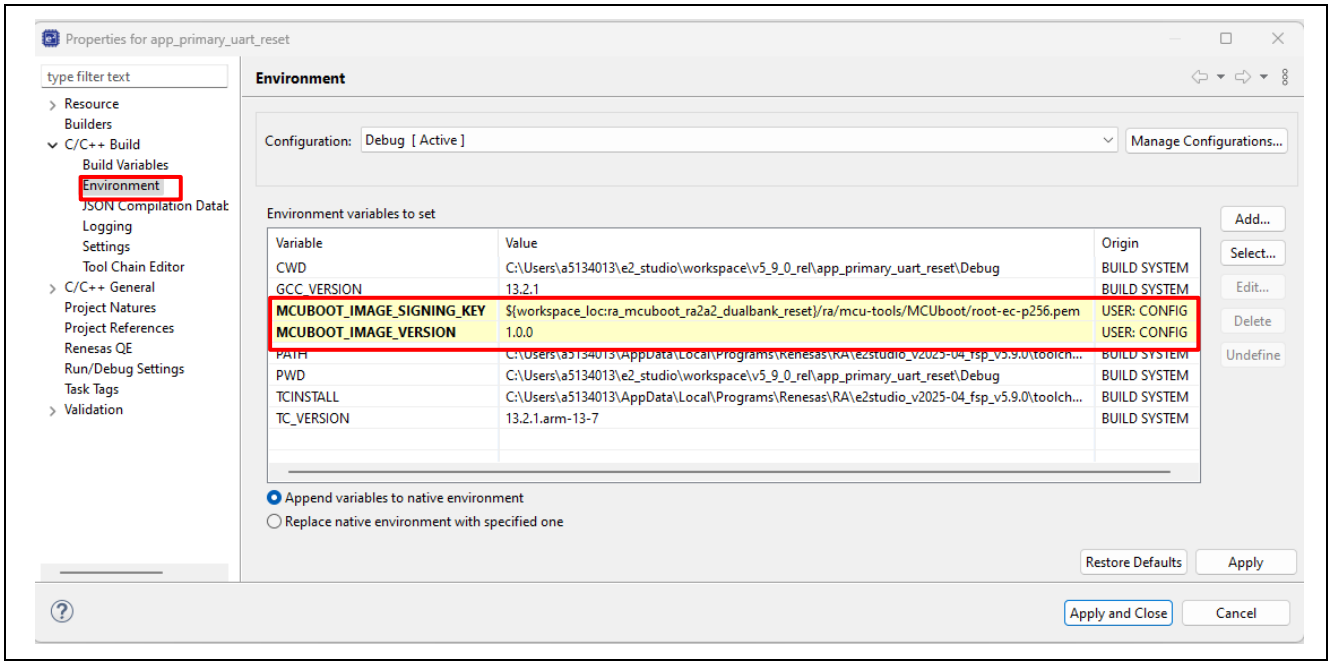


Figure 51. Configure the Application Image Version Number and Signing Key

To be able to re-compile 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 52, so the application project is always recompiled.

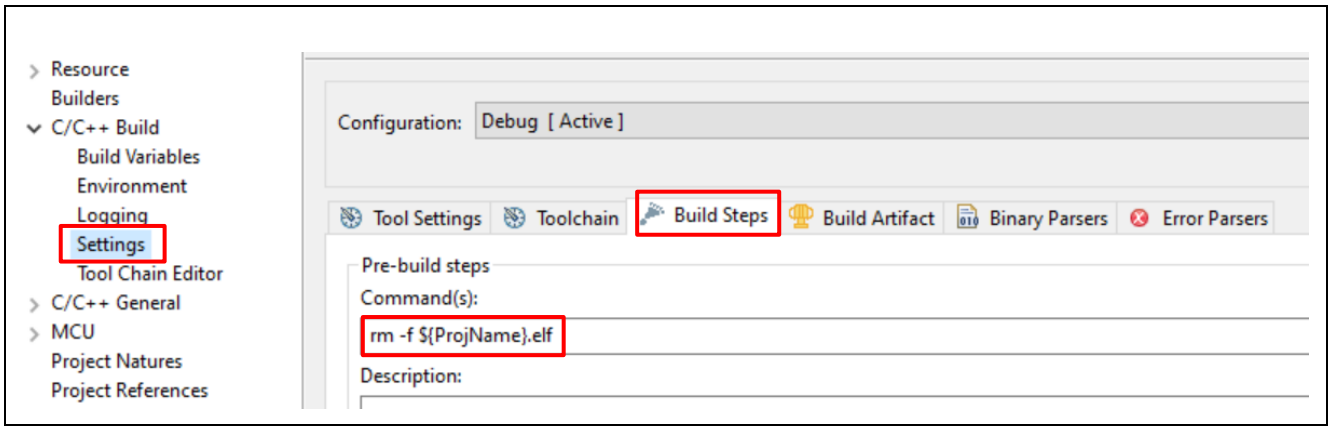


Figure 52. Configure the Pre-build Command

At this point if any Environment variable values were changed, a user can click **Generate Project Content** and compile the newly created application project and ensure that `\Debug\app_primary_uart_reset.bin.signed` is generated.

5.3 Preparation for Production Support

For production support, an `.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, 0x4000 based on Figure 34.

```
srec_cat Debug\app_primary_uart_reset.bin.signed -binary -offset 0x4000 -o
app_primary_uart_reset_signed_offset.srec
```

Figure 53. Create app_primary_uart_reset_signed_offset.srec

Follow steps similar to Section 4.7 to edit the custom **Builder** and compile the primary application:

1. Observe `\ekra2a2_mcuboot_dualbank_reset\app_primary_uart_reset\srec_cat.exe` and `process_signed_binary_primary.bat` in the root of project `app_primary_uart_reset`.
2. Follow Section 4.7 to exam the **Builder**. The configuration should look like Figure 56.

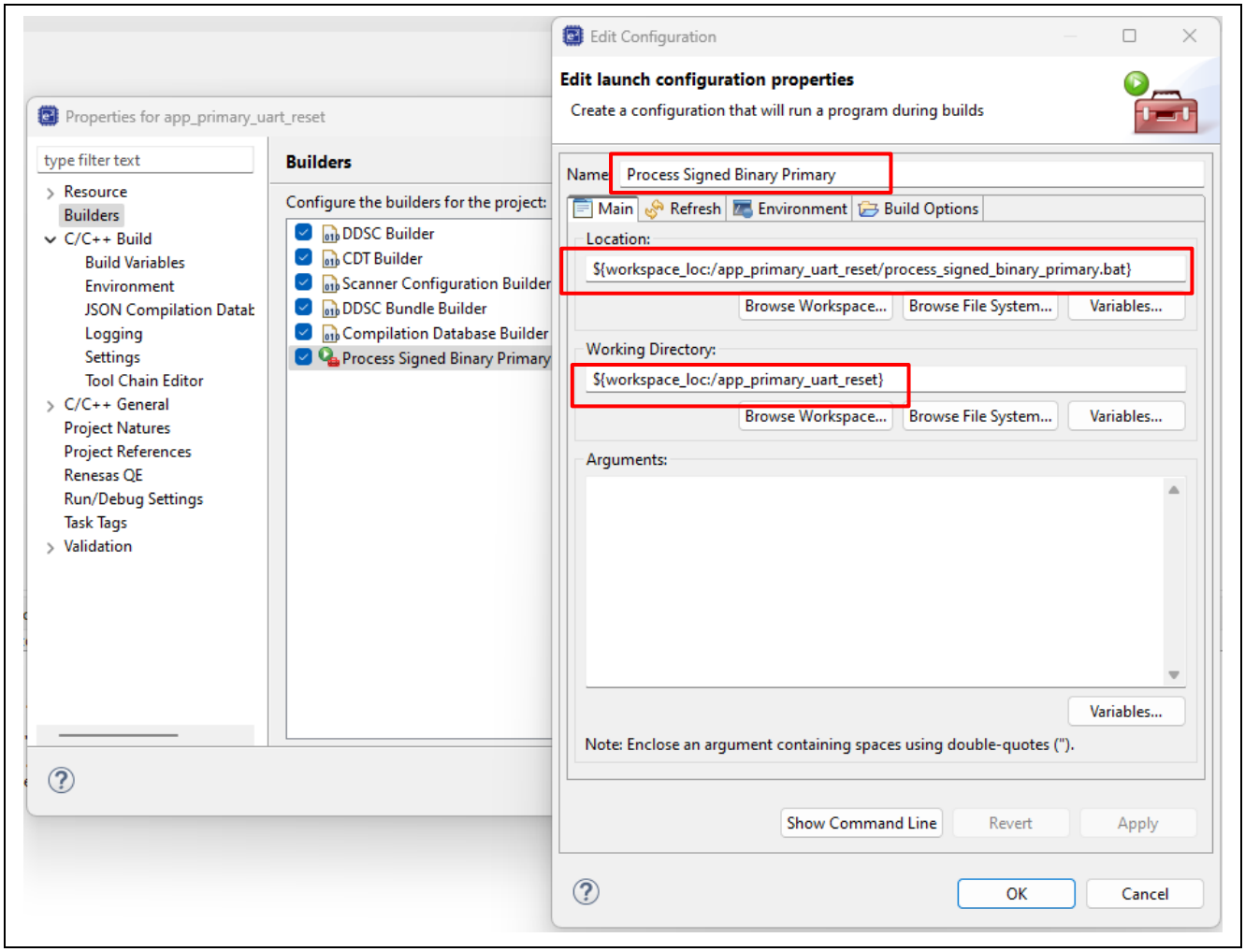


Figure 54. The Custom Builder for the Primary Application

3. If any changes are made to the project, Click **Generate Project Content** and compile the `app_primary_uart_reset` project. Otherwise, ensure that `app_primary_uart_reset_signed_offset.srec` is present under the root of the `app_primary_uart_reset` project.

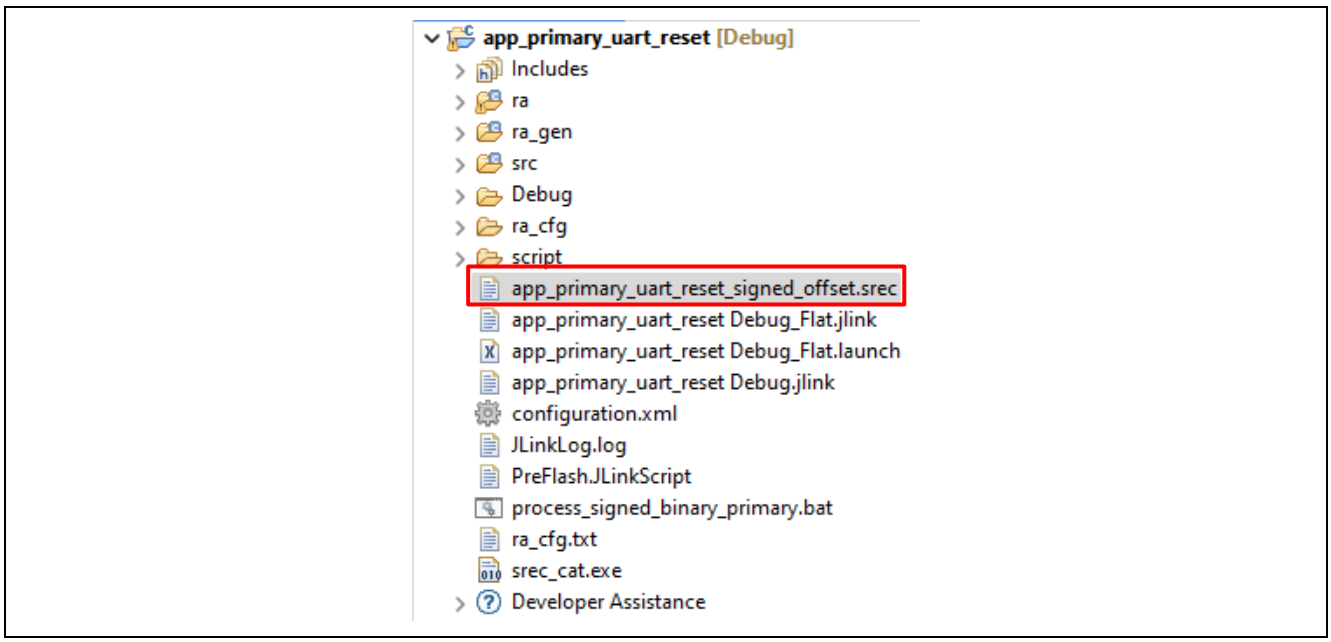


Figure 55. Signed Primary Image Offset to the Primary Slot

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 is prepared to test the image downloader function.

6.1 Prepare a Secondary Image

In the example projects, a secondary image is provided 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 import the secondary application project to the same workspace.

Right-click in the white space in the **Project Explorer** area and select **Import** and choose **Existing Projects into Workspace**.

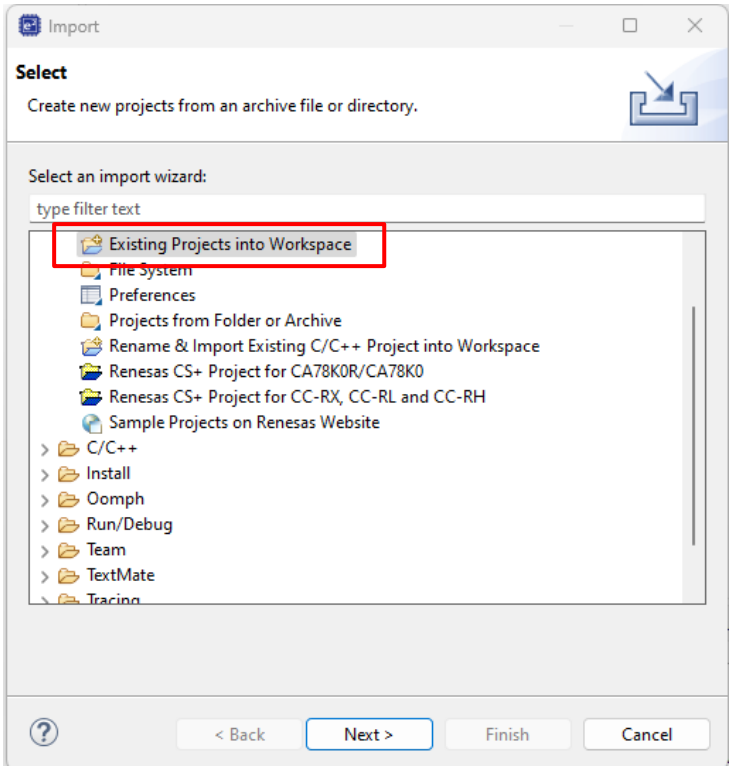


Figure 56. Import the Initial Application

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

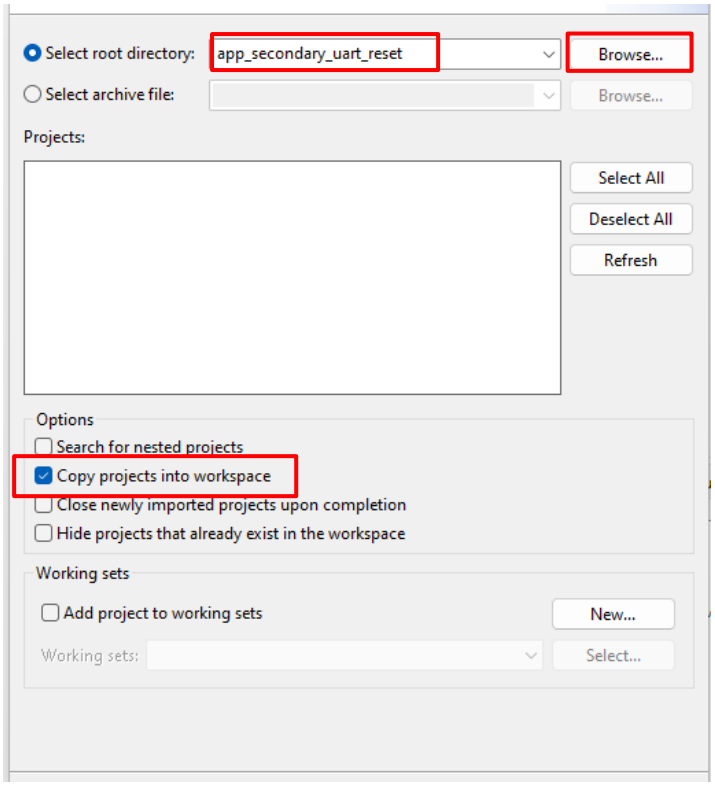


Figure 57. Select the Secondary Application

Browse into the Workspace folder and select `app_secondary_uart_reset`.

Click **Finish**. The secondary application project will be imported with the following attributes:

- The **Build Variable** and **Environment Variables** are automatically imported.
- The secondary app project does not contain the custom Builder “Process Signed Binary Primary
- 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.

6.2 Set Up the Hardware

When using `app_primary_uart_reset` as the initial application project:

- Connect J10 using a USB micro to B cable from the EK-RA2A2 to the development PC to provide power and debug connection using the on-board debugger. The board Green LED is toggled on for each falling edge of a square wave signal connected to input pin P306 of Header J3.

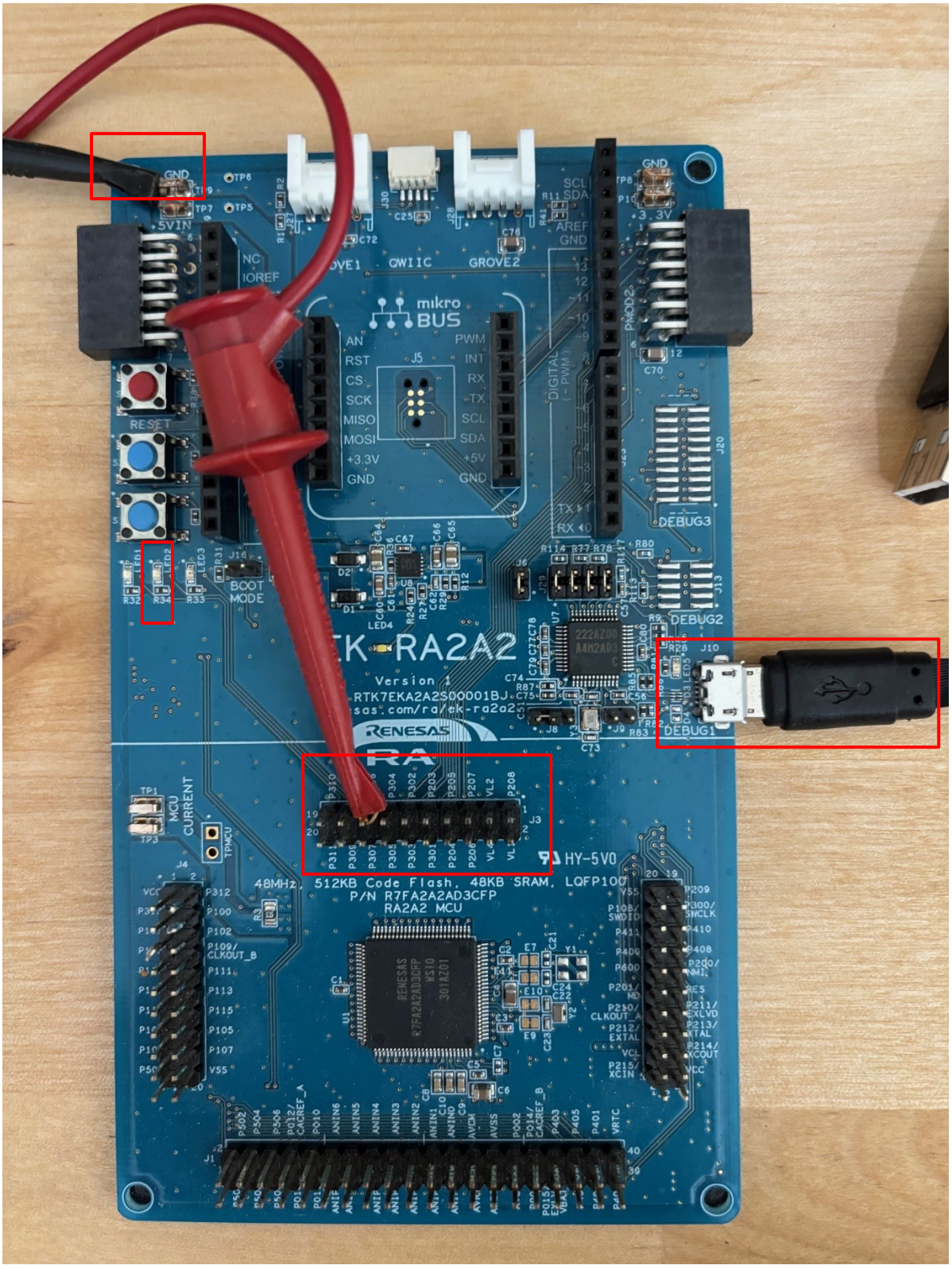


Figure 58. EK-RA2A2 Board Connections

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 J-Flash Lite.

Note: If the MCU is in code flash dual bank mode, make sure to restore it 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 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.

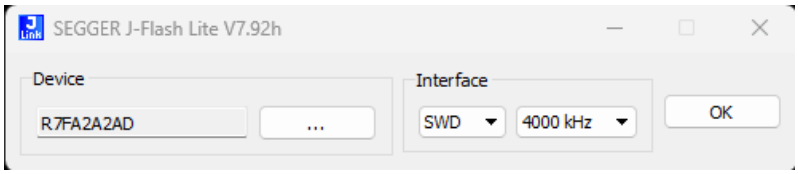


Figure 59. Launch the J-Flash Lite

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

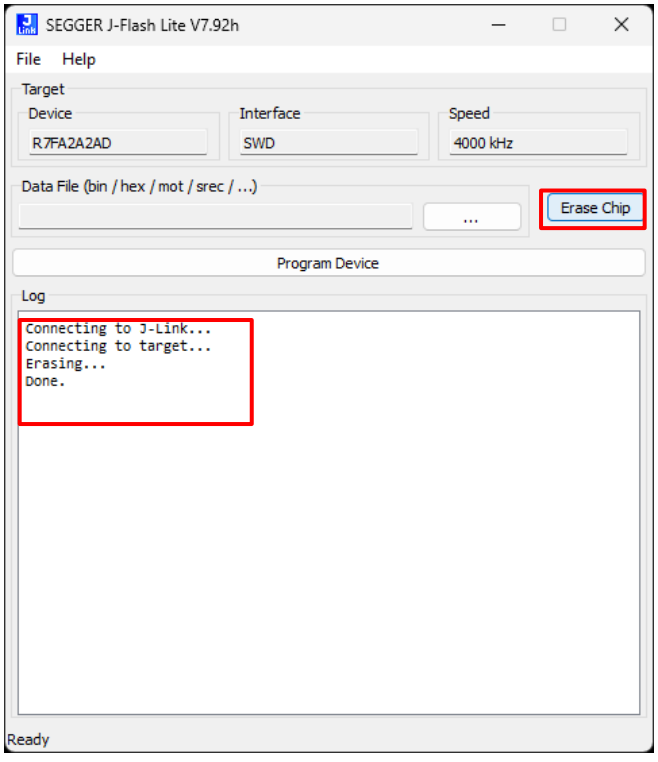


Figure 60. Erase the MCU using J-Flash Lite

Note that when using Segger J-Flash Lite 7.92j 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.

6.4 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_uart_reset->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.

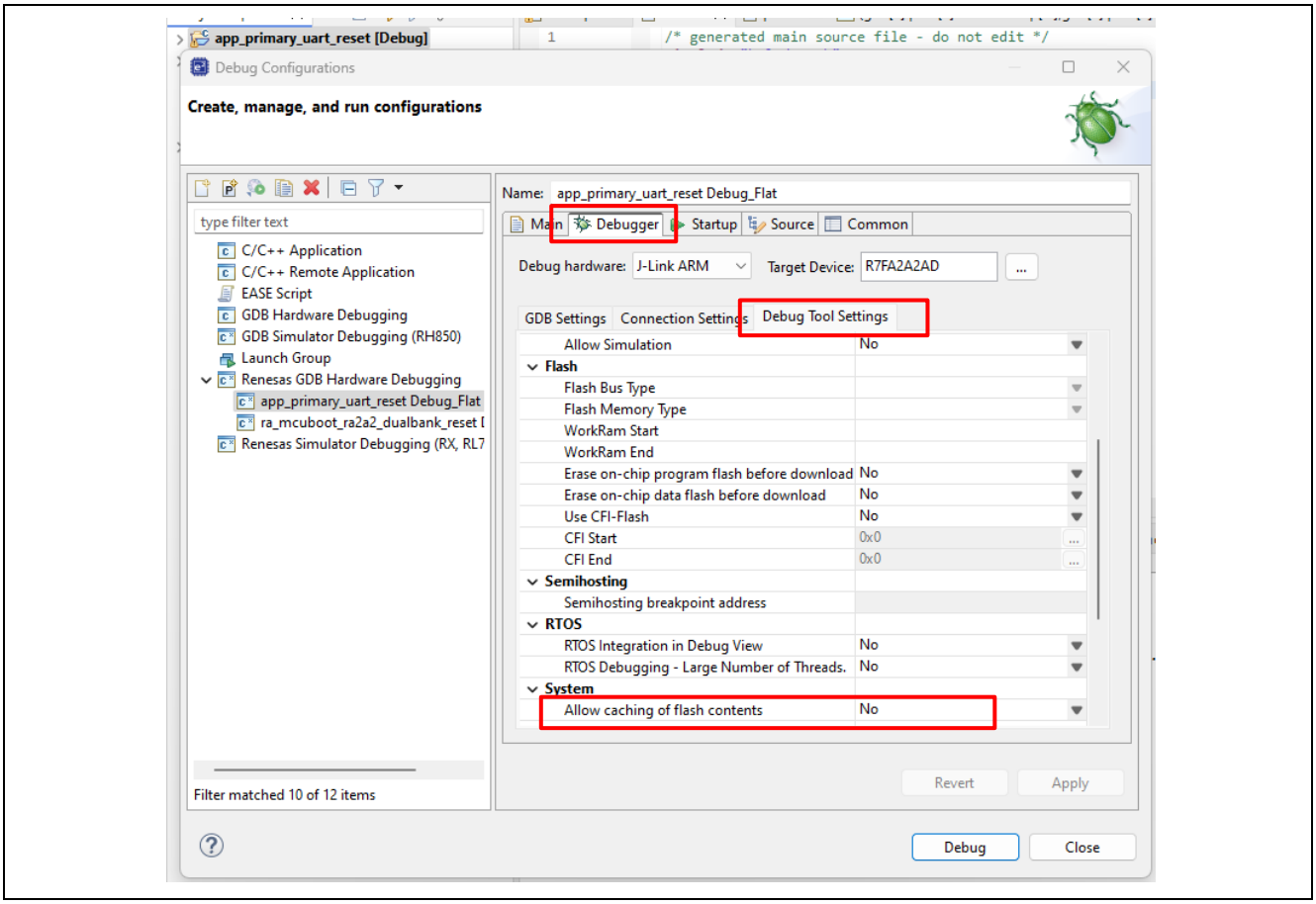


Figure 61. Disable Flash Content Caching

2. Configure the load image and symbols properties.
   Open the **Debug Configurations**: **app_primary_uart_reset->Debug As->Debug Configurations**.
   Make sure **app_primary_uart_reset Debug_Flat** is selected and select the **Startup** tab.
   Click **Add…**, then **Workspace**, navigate to the **ra_mcuboot_ra2a2_dualbank_reset** project, and select the **ra_mcuboot_ra2a2_dualbank_reset.elf** file from the debug folder. Click **OK**.

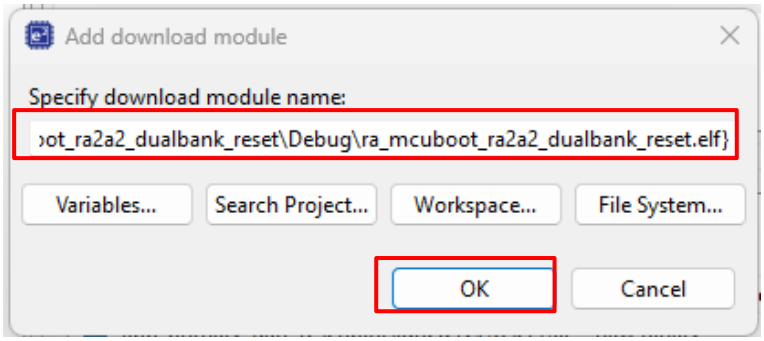


Figure 62. Add the Bootloader Project to Debug Configuration

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

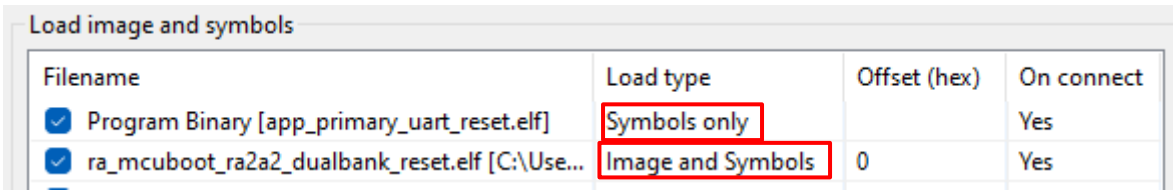


Figure 63. Select to load Symbols only for the Application Project

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.

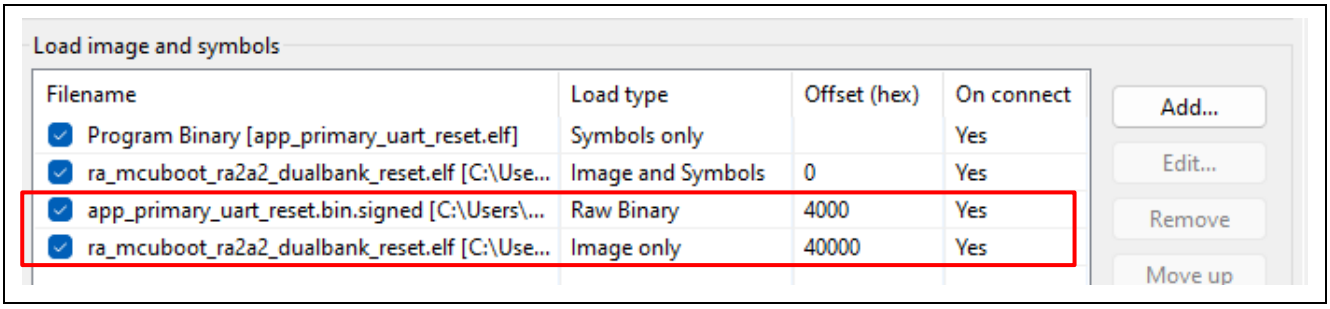


Figure 64. Add the Signed Primary Image and Upper Bank Bootloader

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

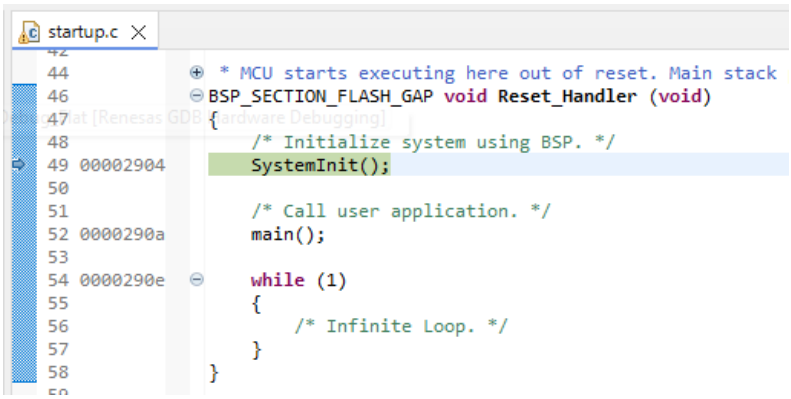


Figure 65. Start the Application Execution

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

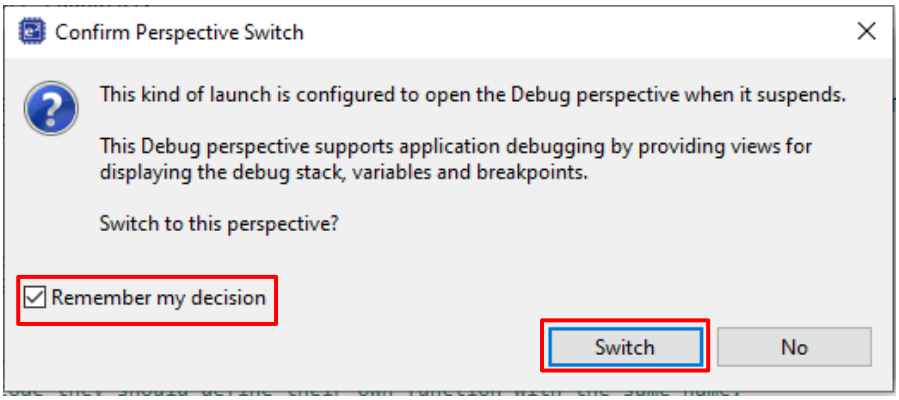


Figure 66. Switch the Perspective

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

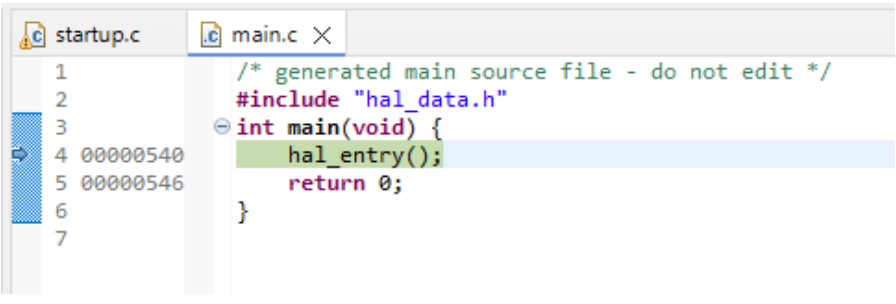


Figure 67. Start the Application Execution

8. Click to run again.
   The red LED on the EK-RA2A2 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**.

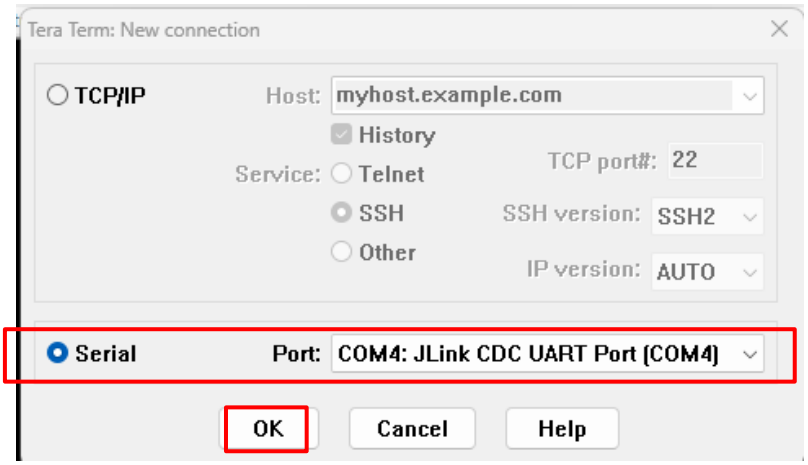


Figure 68. Open the COM Port

Note: When using the UART interface, configure the Terminal New-line settings and select the Serial Terminal and set the **Speed** to 115200 as observed in Figure 70 and 71.

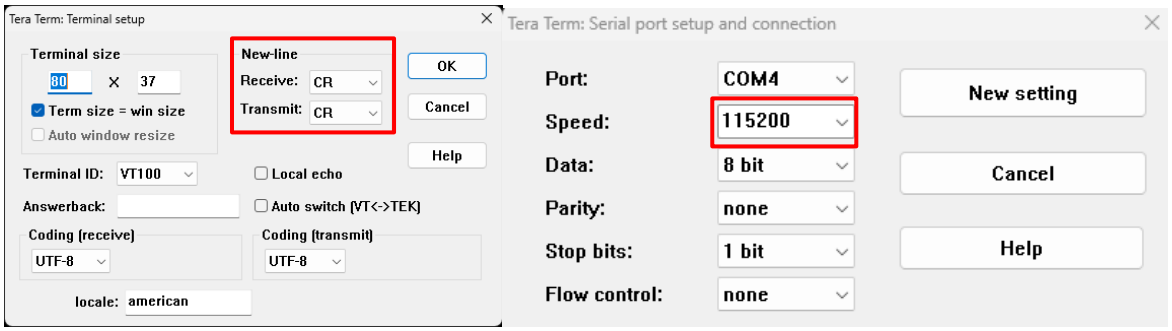


Figure 69. Configure the New-Line and Baud Rate for UART Interface

The menu in Figure 72 will be displayed on the Tera Term.

Figure 70. 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:          0x00004000
Header size:            0x0200 (512 bytes)
Protected TLV size:     0x0000 (0 bytes)
Image size:             0x00006830 (26672 bytes)

*************************
* Secondary Image Slot *
*************************
Image version:          255.255 (Rev: 65535, Build: -1)
Secondary image start address:          0x00044000
Header size:            0xFFFF (65535 bytes)
Protected TLV size:     0xFFFF (65535 bytes)
Image size:             0xFFFFFFFF (-1 bytes)
```

Figure 71. Print the Image Slot Information

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

```
EK-RA2A2 MCUBoot Dual Bank (Swap with Reset) App v1.0.0

Please select from below menu options:

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 run Secondary App after successful download...
```

Figure 72. Choose Option 2 to Download the New Image using XModem

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

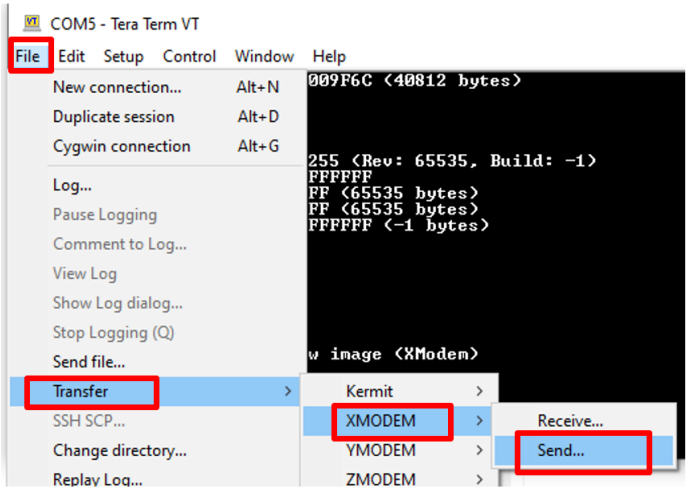


Figure 73. Start Transfer from Tera Term

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

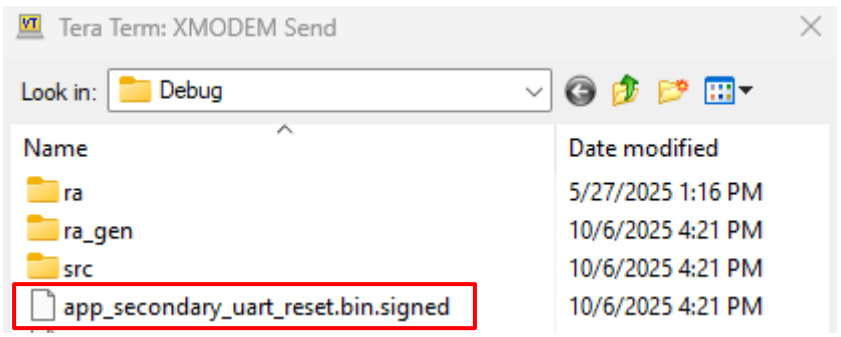


Figure 74. Choose the Signed Secondary Image

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

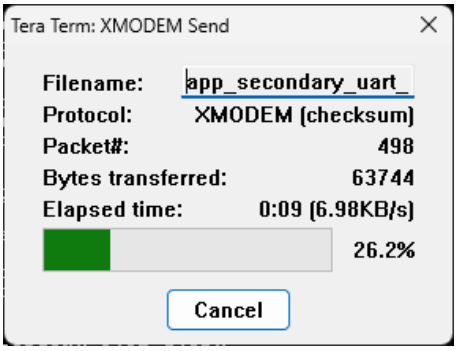


Figure 75. Download the New Image via XModem

6.6 Boot the New Application

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

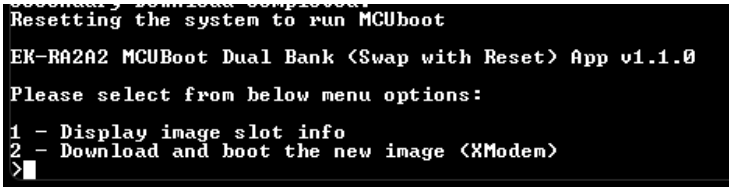


Figure 76. The New Image is Booted

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

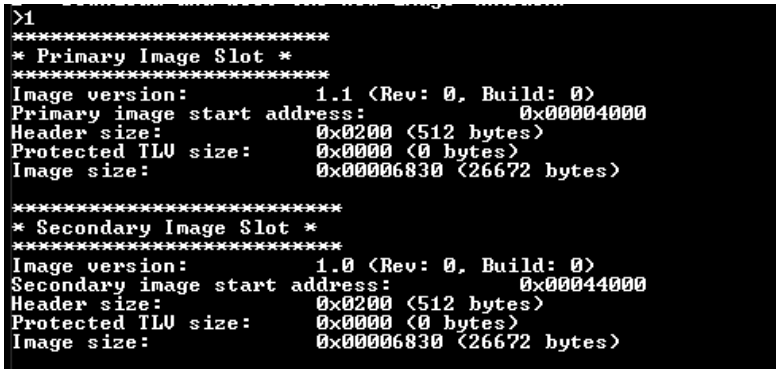


Figure 77. The Slot Layout After New Image is Booted

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 updates, the new image must have a version number higher than the current image in the primary slot.

7. Production Support Considerations

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

7.1 Protect the Bootloader using Memory Protection Unit and Flash Access Window

In this application, we only need to focus on the secure flash program and data regions. Users need to determine the bootloader size and set the boundaries for both the secure flash data and program regions within this area, as shown in Figure 9 Combining the MPU and FAW to protect the bootloader. For the secure flash program region, users can configure the Security MPU Regions in the *ra_mcuboot_ra2a2_dualbank_reset* and *metro_ra_mcuboot_ra2a2_instant* projects under the BSP tab.

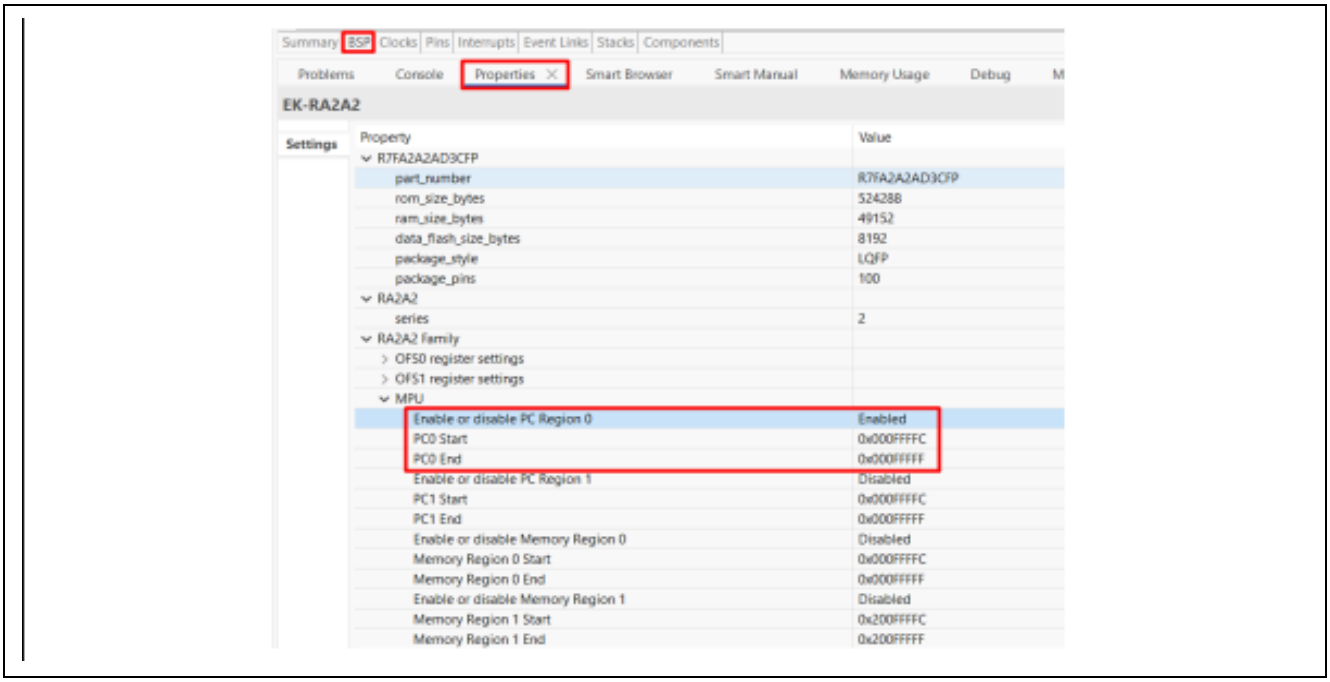


Figure 78. Security MPU Configuration

• Secure flash program

– **Enable or disable PC Region 0**: enable or disable the secure flash program.

– **PC0 Start** and **PC0 End**: program counter region for the secure flash program.

For the secure flash data region, users need to create a customized linker script to define it, as shown in Figure 81. For more details, users can refer to Section 5 in Application Project R11AN0416.

```
/* Linker script to configure memory regions. */
MEMORY
{
        VECTOR_TABLE  (rx)      : ORIGIN = 0x00000000, LENGTH = 0x00000400   /*1024 bytes */
        SECURE_PROGRAM (rx)     : ORIGIN = 0x00000400, LENGTH = 0x0007FC00   /*  512K - 1024 bytes  */
        SECURE_DATA   (rw)      : ORIGIN = 0x00080000, LENGTH = 0x00080000    /*  512K bytes */
        FLASH (rx)              : ORIGIN = 0x00100000, LENGTH = 0x00100000   /*  1MB */
        SECURE_RAM_PROGRAM (rwx)    : ORIGIN = 0x1FFE0000, LENGTH = 0x00010000 /*   64K     */
        SECURE_RAM (rw)         : ORIGIN = 0x1FFF0000, LENGTH = 0x00040000   /*      256K    */
        RAM (rwx)               : ORIGIN = 0x20030000, LENGTH = 0x00050000   /*      320K    */
        DATA_FLASH (rx)         : ORIGIN = 0x40100000, LENGTH = 0x00008000   /*  32K         */
        QSPI_FLASH (rx)         : ORIGIN = 0x60000000, LENGTH = 0x04000000   /*  64M         */
        SDRAM (rwx)             : ORIGIN = 0x90000000, LENGTH = 0x02000000   /*  32M         */

}
```

Figure 79. Customized Linker Script for Secure Data Region

Note that this is an example of the customized linker script, users need to calculate the memory regions to fit their application project.

For the FAW region, users only need to call the FSP FAW API:

err = R_FLASH_LP_AccessWindowSet(&g_flash0_ctrl, FAW_START, FAW_END);

where:

• g_flash0_ctrl is the instance of this flash HAL driver. • FAW_START is the start address of the FAW window.

• FAW_END is the address of the next block acceptable for programming and erasure defined by the access window.

Note:

• If the FAW is permanently locked before running this API, the FAW region cannot be updated using this API.

• It is recommended to set up the FAW region outside of Security MPU Regions, as shown in Figure 9.

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

8.1 Using Dual Bank Swap with Reset

For ekra2a2_mcuboot_dualbank_reset, Dual Bank Swap with Reset: three projects are needed:

- `ra_mcuboot_ra2a2_dualbank_reset`
- `app_primary_uart_reset`
- `app_secondary_uart_reset`

Users can follow the steps below to run the example projects. Follow the instructions in section 6.2 to set the hardware.

1. Import the above-mentioned three projects to a workspace.
2. Open the `Configuration.xml` file from project `ra_mcuboot_ra2a2_dualbank`.
3. Click **Generate Project Content**.
4. Compile the project `ra_mcuboot_ra2a2_dualbank`.
5. Open the `Configuration.xml` file from project `app_primary_uart_reset`.
6. Click **Generate Project Content**.
7. Compile `app_primary_uart_reset`.
8. Open the `Configuration.xml` file from project `app_secondary_uart_reset`.
9. Click **Generate Project Content**.
10. Compile the `app_secondary_uart_reset` project.
11. Erase the entire chip following the instructions in section 6.3.
12. Debug the application from project `app_primary_uart_reset` in the e^2 studio environment.
13. Resume the program execution twice. The Red LED should be blinking and the Metrology application Green LED blinks at a rate that tracks a square wave signal applied to input P306. Set the square wave signal frequency to 2 to 15 Hz range. During the firmware download and after MCU reset, the application continues to monitor the signal input on P306.
14. Open the Tera Term with the enumerated COM port and set up the baud rate as 115200.
15. Use Tera Term to send file: `\app_secondary_uart_reset\Debug\app_secondary_uart_reset.bin.signed` to the MCU by following section 6.6. This will take about 50 seconds.
16. The system will reset automatically after download.
17. Blue LED should be blinking and the Green LED will blink for an applied square wave signal on P306.
18. 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 Dual Bank Swap Instant (without reset)

For ekra2a2_mcuboot_dualbank_instant, Dual Bank Swap Instant: three projects are needed:

- `metro_ra_mcuboot_ra2a2_dualbank`
- `metro_app_primary_uart_instant`
- `metro_app_secondary_uart_instant`

Users can follow the steps below to run the example projects. Follow the instructions in section 6.2 to set the hardware.

1. Import the above-mentioned three projects to a workspace.
2. Open the `Configuration.xml` file from project `metro_ra_mcuboot_ra2a2_dualbank`.
3. Click **Generate Project Content**.
4. Compile the project `metro_ra_mcuboot_ra2a2_dualbank`.
5. Open the `Configuration.xml` file from project `metro_app_primary_uart_instant`.

6. Click **Generate Project Content**.
7. Compile `metro_app_primary_uart_instant`.
8. Open the `Configuration.xml` file from project `metro_app_secondary_uart_instant`.
9. Click **Generate Project Content**.
10. Compile the `metro_app_secondary_uart_instant` project.
11. Erase the entire chip following the instructions in section 6.3.
12. Debug the application from project `metro_app_primary_uart_instant` in the e^2 studio environment.
13. Resume the program execution twice. The Red LED should be blinking.
14. Open the Tera Term with the enumerated COM port and set up the baud rate as 115200.
15. Use Tera Term to send file: `\metro_app_secondary_uart_instant\Debug\metro_app_secondary_uart_instant.bin.signed` to the MCU by following section 6.6. This will take about 50 seconds.
16. The system will start automatically after download and a soft start is performed on the MCU. The MCU is not reset and continues to run the Metrology application indicated by the Green LED which blinks at a rate that tracks a square wave signal; which is applied to input P306 and set for a frequency in 2 to 15Hz range. During the firmware download and MCU soft start the application continues to monitor the signal input on P306. Interrupts continue to operate during the firmware download and soft start phase of the application.
17. Blue LED should be blinking indicating the Secondary App is running and the Green LED will continue to blink for the applied square wave signal on P306.
18. 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).

    Observe in Figure 82, during the firmware transfer and download phase you may notice some jitter on the Green LED blink rate. This is due to the programming of flash sectors as the image is programmed into the flash memory. The FSP flash subsystem writes the sector blocks with critical sections used to protect against external interrupts until the sector writes complete.

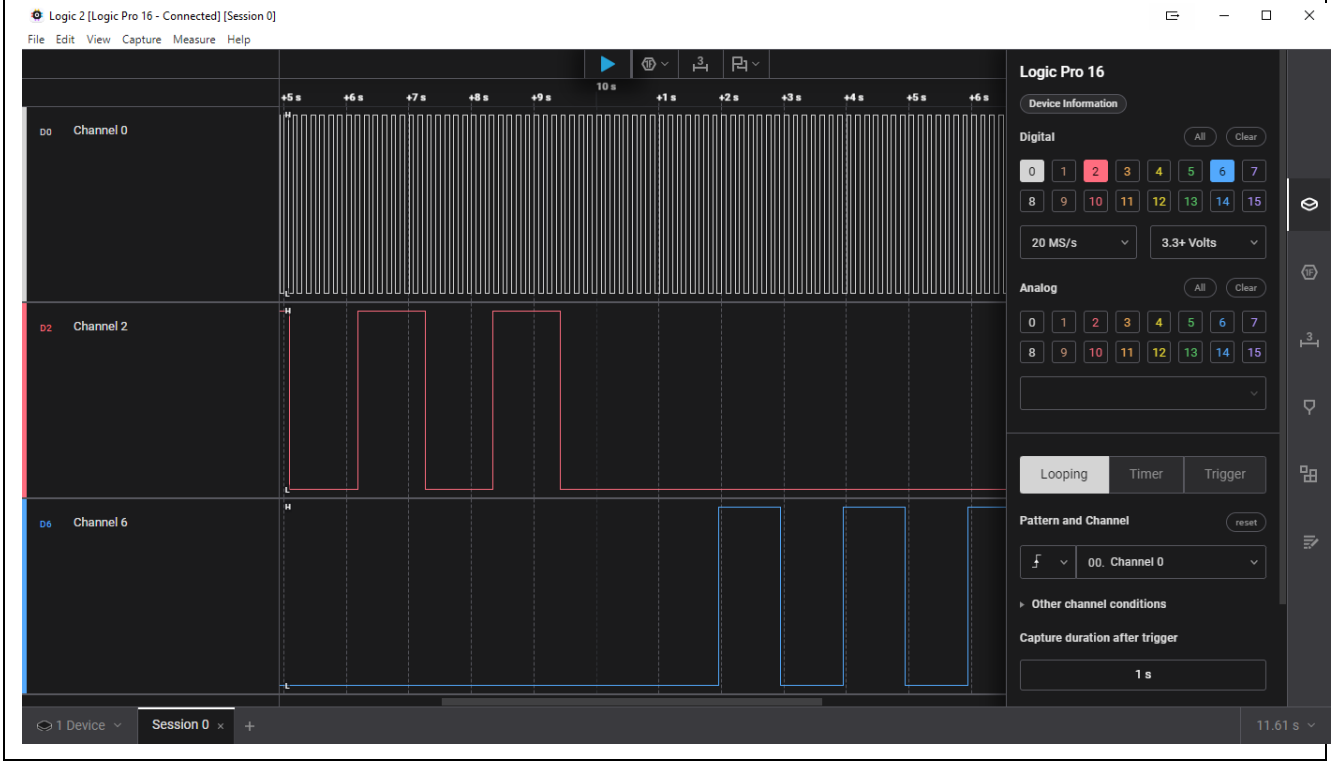


Figure 80. Dual Bank Swap Instant at MCU Soft Start

In addition, in Figure 82, the analyzer shows the board LEDs blinking during the firmware download transfer complete and soft start phase when the secondary firmware image runs.
Channel 2 is the board Red LED blinking when primary app is running.
Channel 6 is the Blue LED blinking after Bank Swap and MCU soft start runs the secondary app and notice that Channel 0 (Green LED) continues to operate during the firmware download, Dual Bank Swap Instant (without reset), MCU soft start, and secondary application startup.

9. References

1. RA2A2 User's Manual Hardware (R01UH1005) RA2A2 Group User's Manual: Hardware
2. Renesas RA2 MCU Secure Bootloader for RA2 MCU Series (R11AN0516) Secure Bootloader for RA2 MCU Series – Application Project
3. Renesas RA2 MCU Advanced Secure Bootloader Design using MCUboot Internal Code Flash and Memory Mirror Function (R01AN7766) RA2 MCU Advanced Secure Bootloader Design using MCUboot Internal Code Flash and Memory Mirror Function
4. Renesas RA Family RA6 Series MCU Basic Secure Bootloader Design using MCUboot with Code Flash Linear Mode Application Project (R11AN0497) RA6 Basic Secure Bootloader Using MCUboot and Internal Code Flash
5. RA6 Secure Firmware Update using MCUboot and Flash Dual Bank (R11AN0570) RA6 Secure Firmware Update using MCUboot and Flash Dual Bank – Application Project

10. Website and Support

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

EK-RA2A2 Resources	renesas.com/ra/ek-ra2a2
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	Nov.21.25	-	Initial release
		-	
		-	
		-	

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.