Renesas RA Family

RA0/RA2 MCUboot and USB-Based Firmware Updates

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 independent. Its operation across different hardware platforms relies on hardware porting layers. MCUboot is maintained by TrustedFirmware in the GitHub mcu-tools page https://github.com/mcu-tools/mcuboot. There is a /docs folder that holds the documentation for MCUboot in .md file format. This application note refers to those documents wherever possible.

The Renesas Flexible Software Package (FSP) integrates an MCUboot port across the entire RA MCU Family, starting from FSP v3.0.0. Implementing security into MCUboot requires the use of cryptographic algorithms. The memory footprint of Arm Mbed Crypto makes it unsuitable for smaller memory MCU such as the Renesas RA0 and RA2. To address this, Renesas integrates TinyCrypt—a lightweight cryptographic library—into the MCUboot module.

This application note offers clear, step-by-step instructions for building a secure bootloader using the MCUboot module with TinyCrypt, strengthening security on applications that target the Renesas RA0 and RA2 Series MCUs. It includes guidance on configuring application projects and performing system startup. Additionally, the document demonstrates how to implement an internal Flash rewrite program using MCUboot and USB Communications Device Class (CDC).

Required Resources

Development of tools and software

- e² studio IDE v2025-07
- RA Family Flexible Software Package (FSP) v6.1.0 or later
- SEGGER J-link® V8.58
- Terminal program which supports XMODEM file transfer protocol (ex. Tera-term v4.99).
- LLVM for Arm v18.1.3
- Python v3.13.7

The above 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

Hardware

- EK-RA2L2, with a Type-C USB cable, with a Micro USB cable, Evaluation Kit for RA2L2 MCU Group (http://www.renesas.com/ek-ra2l2)
- FPB-RA0E1, with a Type-C USB cable, Fast Prototyping Board for RA0E1 MCU Group (https://www.renesas.com/fpb-ra0e1)
- Host PC running Windows® 10/11 OS

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas e² studio IDE and RA Family Flexible Software Package (FSP). Before you perform the procedures in this application note, follow the procedure in the *FSP User Manual* to build and run the Blinky project. Doing so enables you to become familiar with the e² studio and the FSP and validates that the debug connection to your board functions properly. In addition, the application note assumes that you have some knowledge of cryptography. Prior knowledge of Python usage is also helpful. Prerequisite reading RA Flexible Software Package Documentation: MCUboot Port (rm_mcuboot_port) and chapter 3.2 in RA2 MCU Advanced Secure Bootloader Design using MCUboot Internal Code Flash and Memory Mirror Function. The goal of reading is to understand MCUboot in FSP and XMODEM file transfer protocol.

The intended audience is users who want to develop secure bootloader use TinyCrypt and firmware update via USB CDC (USB Communication Device Class) applications on Bare-metal environments using Renesas RA Family MCUs.

Contents

1. Overview of MCUboot 5
1.1 History of MCUboot 5
1.2 MCUboot Functionalities Overview 5
1.3 Overview of Application Booting Process 5
1.4 Application Update Strategies 5
2. Application Project Overview 7
2.1 Overview of using MCUboot with TinyCrypt on RA0E1 and RA2L2 devices 7
2.2 Overview of Hardware Connections 9
2.3 Overview of Software components 9
2.4 Internal Code Flash rewrite program via USB CDC overview 11
3. Application Project Package 15
4. Creating the Bootloader Project 19
4.1 Including the MCUboot Module in the Bootloader Project 19
4.2 Further Optimizing for the Bootloader Project Size 29
4.3 Compiling the Bootloader Project 33
4.4 Configuring the Python Signing Environment 33
5. Creating the User Application 35
5.1 Generate the Primary User Application Project 35
5.2 Implementing the Image Downloader sample for EK-RA2L2 Board 37
5.3 Signing the User Application Image 41
6. MCUboot Memory Configuration with Renesas FSP Solution Project 45
6.1 How to Set Up the Renesas FSP Solution Project 45
6.2 Managing the MCUboot Memory Configuration 47
6.3 Optimizing SRAM Allocation 50
7. Booting the Initial Application Project 52
7.1 Erase the MCU 52
7.1.1 Use the Renesas Flash Programmer 52
7.1.2 Use the J-Flash Lite 54
7.2 Configure the Debugger 56
7.3 Open the Tera Term Terminal for RA2L2 60
7.4 RA0E1 - Open the J-Link RTT Viewer 61
8. Mastering and Delivering a New Application 62
8.1 Prepare a Secondary Image 62
8.2 Update Existing Application to a New Application 64
8.3 Downloading and Booting the Secondary Application 66

8.3.1 Using the Load Ancillary File tool 66
8.3.2 Using the XMODEM-based Image Downloader 66

9. Appendix: Build and Execute the Provided Application Projects 70
9.1 Running the EK-RA2L2 Overwrite Update Mode sample project 70
9.2 Running the FPB-RA0E1 Overwrite Update Mode sample project 78

10. Known Issues and Limitations 81

11. Website and Support 82

Revision History 83

1. Overview of MCUboot

1.1 History of MCUboot

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 repository was later migrated from JuulLabs to the mcu-tools github project. In 2020, MCUboot was moved under the Linaro Community Project umbrella as an open-source project, and it now resides under TrustedFirmware (https://www.trustedfirmware.org/projects/mcuboot).

1.2 MCUboot Functionalities Overview

MCUboot handles the firmware integrity and authenticity check after startup and the firmware switch part of the firmware update process. The operation of switching the firmware from the original image to a new image depends on the image upgrade method. The image upgrade methods are described in section 1.4. Downloading a new firmware version is out of scope for MCUboot. This functionality is provided by the application project described in this document.

1.3 Overview of Application Booting Process

For applications utilizing MCUboot the MCU memory is partitioned into the following regions: MCUboot, Primary Application, Secondary Application, and Scratch Area. Below is an example of the single-image MCUboot memory map. For additional details on the MCUboot memory layout, refer to the Flash Map section on the MCUboot website.

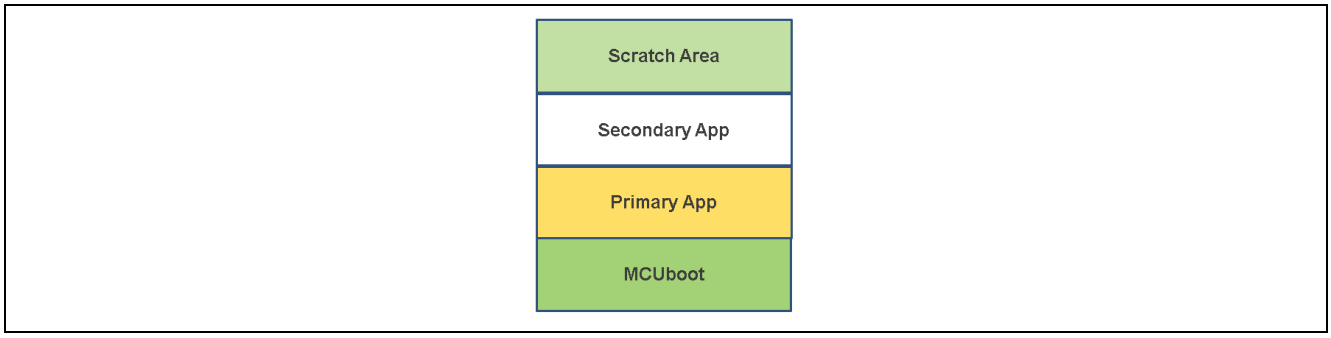


Figure 1. Single Image MCUboot Memory Flash Map

MCUboot's functionality during the boot and update phases adheres to the following process:

1. The bootloader starts when the MCU exists from reset state.
2. If there are images in the Secondary App memory are marked for update, the bootloader performs the following actions:
   A. The bootloader verifies the integrity and authenticity of the Secondary image.
   B. Upon successful authentication, the bootloader switches to the new image based on the update method selected.
   C. The bootloader boots the new image.
3. If there is no new image in the Secondary App memory region, the bootloader authenticates the Primary applications and boots the Primary image.

Application authentication is configurable in terms of the authentication methods and whether the authentication is performed by MCUboot. The firmware image can be authenticated by hash (SHA-256) and digital signature validation.

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.

1.4 Application Update Strategies

The following update strategies are supported by MCUboot. The Renesas FSP MCUboot Module supports one or more of the following strategies depending on the FSP version. The analysis of any advantages or disadvantages is based on the MCUboot functionality, not the FSP MCUboot Module functionality. In addition, this application note is not intended to provide all details on the MCUboot application update strategies. More details on these update strategies by referring to the MCUboot design page:

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

- **Overwrite**
  In the Overwrite update mode, the active firmware image is always executed from the Primary slot, and the Secondary slot is a staging area for new images. Before the new firmware image is executed, the entire contents of the Primary slot are overwritten with the contents of the Secondary slot (the new firmware image).
  - Advantages:
    - Fail-safe and resistant to power-cut failures.
    - Less memory overhead, with a smaller MCUboot trailer and no Scratch Area.
    - Encrypted image support is available when using external flash.
  - Disadvantages:
    - Does not support pre-testing of the new image prior to overwrite.
    - Does not support automatic application fallback mechanism.

  Overwrite upgrade mode is supported by Renesas RA FSP v3.0.0 or later.

- **Swap**
  In the Swap image upgrade mode, the active image is also stored in the Primary slot and is always started by the bootloader. If the bootloader finds a valid image in the Secondary slot that is marked for upgrade, then contents of the Primary slot and the Secondary slot are swapped. The new image then starts from the Primary slot.
  - Advantages:
    - The bootloader can revert the swapping as a fallback mechanism to recover the previous working firmware version after a faulty update.
    - The application can perform a self-test to mark itself permanently.
    - Fail-safe and resistant to power-cut failures.
    - Encrypted image support is available when using external flash.
  - Disadvantages:
    - Need to allocate a Scratch Area.
    - Larger memory overhead, due to a larger image trailer and additional Scratch Area.
    - Larger number of write cycles in the Scratch Area, wearing the Scratch sectors out faster.

  Swap upgrade mode is supported by Renesas RA FSP v3.0.0 or later.

- **Direct execute-in-place (XIP)**
  In the direct execute-in-place 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.
  - 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.

  Direct execute-in-place is supported by Renesas FSP v3.4.0 or later.

2. Application Project Overview

2.1 Overview of using MCUboot with TinyCrypt on RA0E1 and RA2L2 devices

Implementing security into MCUboot requires the use of cryptographic algorithms. The memory footprint of Arm Mbed Crypto makes it unsuitable for smaller memory MCU such as the Renesas RA0 and RA2. To address this, the Renesas Flexible Software Package (FSP) integrates TinyCrypt—a lightweight cryptographic library—into the MCUboot module.

- MCUboot is responsible for verifying the integrity and authenticity of firmware images before they are executed. It uses cryptographic methods such as SHA-256 hashing and ECDSA P-256 digital signature validation.
- TinyCrypt provides cryptographic primitives, ensuring that the verification process is both secure and efficient. For details on using TinyCrypt Cryptographic algorithms, refer to the /tinycrypt/documentation/ folder on Intel GitHub repository (https://github.com/intel/tinycrypt).

The bootloader checks firmware image stored in flash memory. If the image passes integrity and authenticity checks, it is executed; otherwise, the device remains in a safe state.

The application project describes Overwrite upgrade mode in a Bare-metal environment for the following devices:

- RA0E1: 64KB Code Flash, 12KB SRAM
- RA2L2: 128KB Code Flash, 16KB SRAM

Other upgrade modes can be configured through FSP MCUboot's properties and are discussed in other Secure Bootloader applications for RA4, RA6 and RA8. In addition to using mastering tools to flash new images to secondary slot, the RA2L2 application project provides a downloading program using USB CDC and XMODEM protocol. This program is integrated into application images for continuous rewriting without using mastering tools. Figure 2 illustrates overview operation of firmware update using Overwrite upgrade mode on RA0E1 and RA2L2.

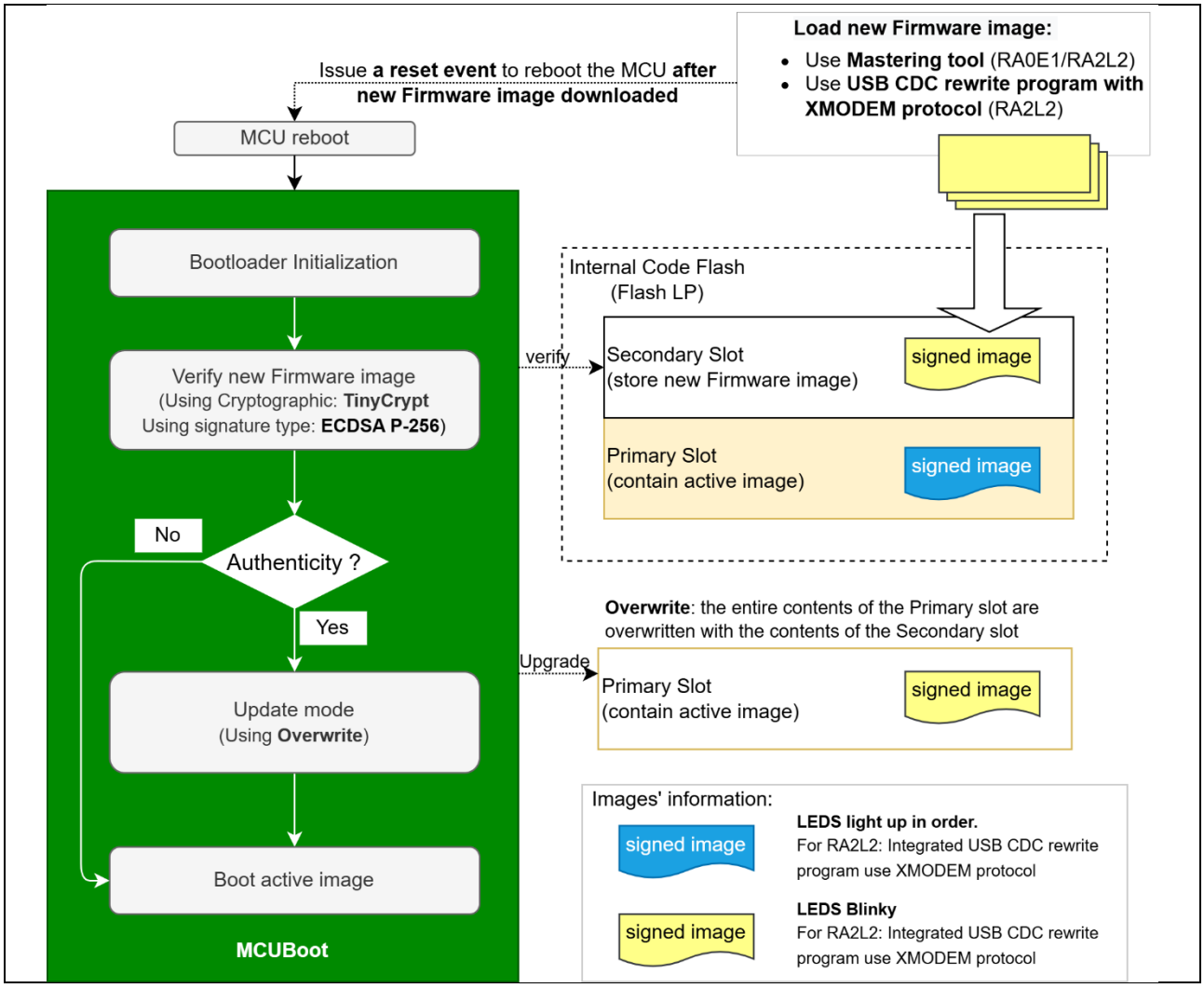


Figure 2. Firmware update using MCUboot Overwrite mode on RA0E1/RA2L2

The programmed image can be monitored by observing LED behavior. When the initial image (primary application) is booted, the LEDs illuminate sequentially. For example:

- On FPB-RA0E1, LED1 lights up first, followed by LED2.
- On EK-RA2L2, LED1 lights up first, followed by LED2 and 3 in sequence.

Once the new image (secondary application) has been authenticated and successfully booted, the LEDs begin blinking.

2.2 Overview of Hardware Connections

The following are hardware connections for FPB-RA0E1 and EK-RA2L2.

- FPB-RA0E1: Set J9 as open to use the on-board debug. Connect CN6 (USB Type-C™ 2.0 connector) with the Host PC.

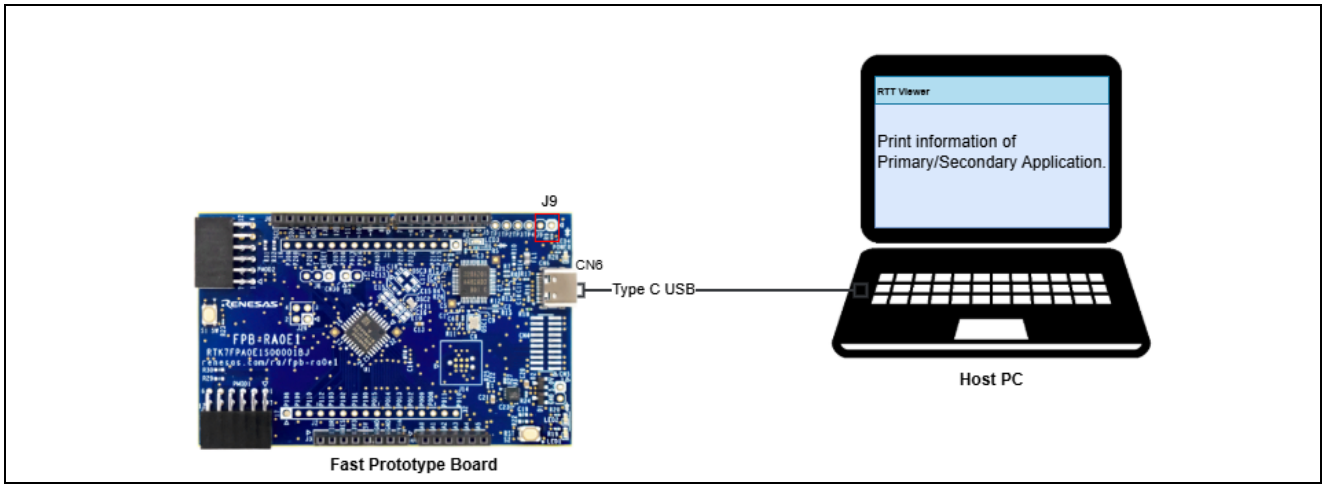


Figure 3. **Hardware connection in FPB-RA0E1**

- EK-RA2L2: Use the EK-RA2L2 with default jumper connections for normal operation and for on-board debug. Ensure that pins 2-3 of J16 are closed.
  For debug connect to the Host PC via J10 (Debug1 - Micro USB), and for application use connect to the Host PC via J11 (USB FS - USB Type-C).

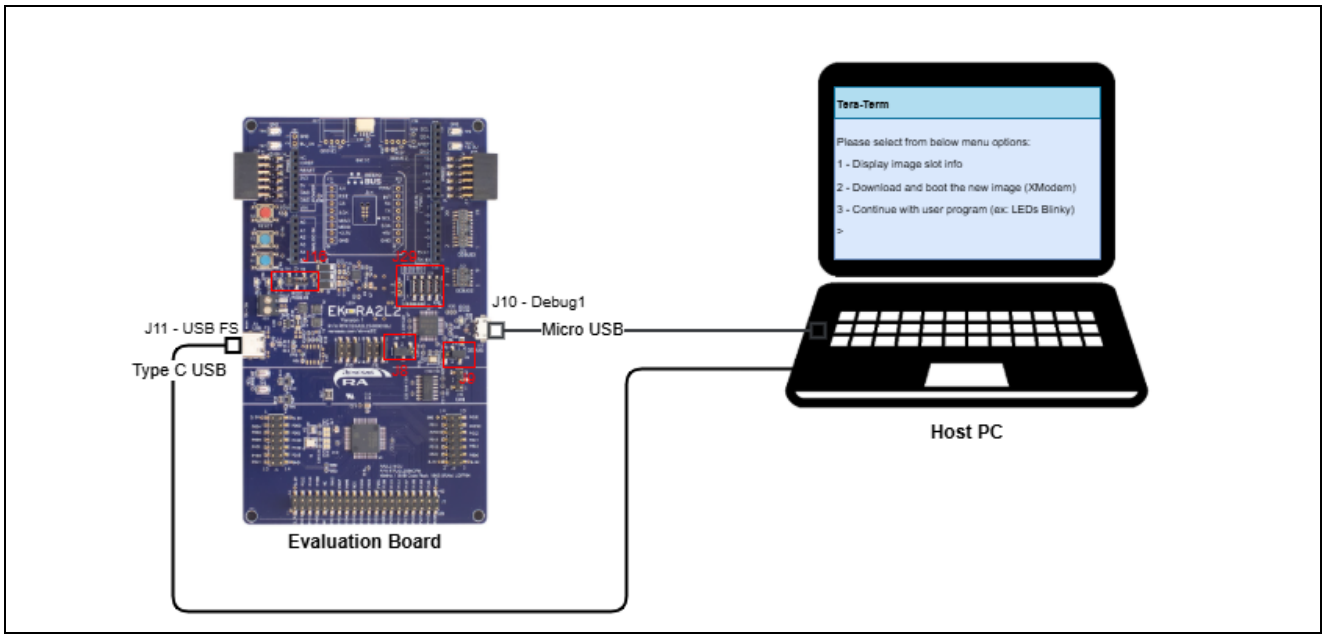


Figure 4. Hardware connection in EK-RA2L2

2.3 Overview of Software components

The following software stacks are used in the application project.

- Common software stacks for the bootloader projects include the middleware: MCUboot, MCUboot port for RA, MCUboot TinyCrypt (S/W Only) and Flash driver.

- I/O Port is used for LED control in both primary and secondary projects.
- Image downloading via USB PCDC Communication uses the middleware and drivers: USB basic, GPT and Flash.

For detailed information on each component, refer to FSP documentation (https://renesas.github.io/fsp/)

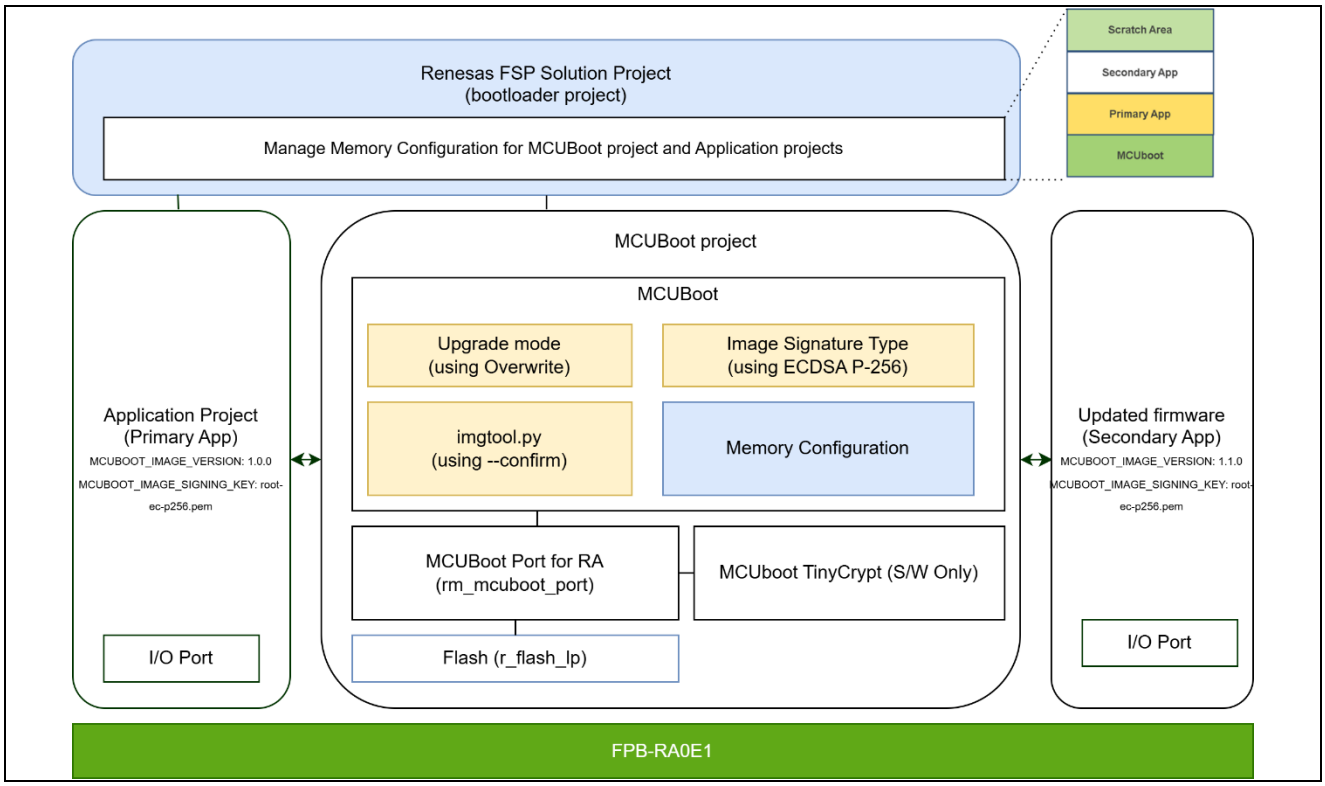


Figure 5. Software Stacks used in FPB-RA0E1 sample project

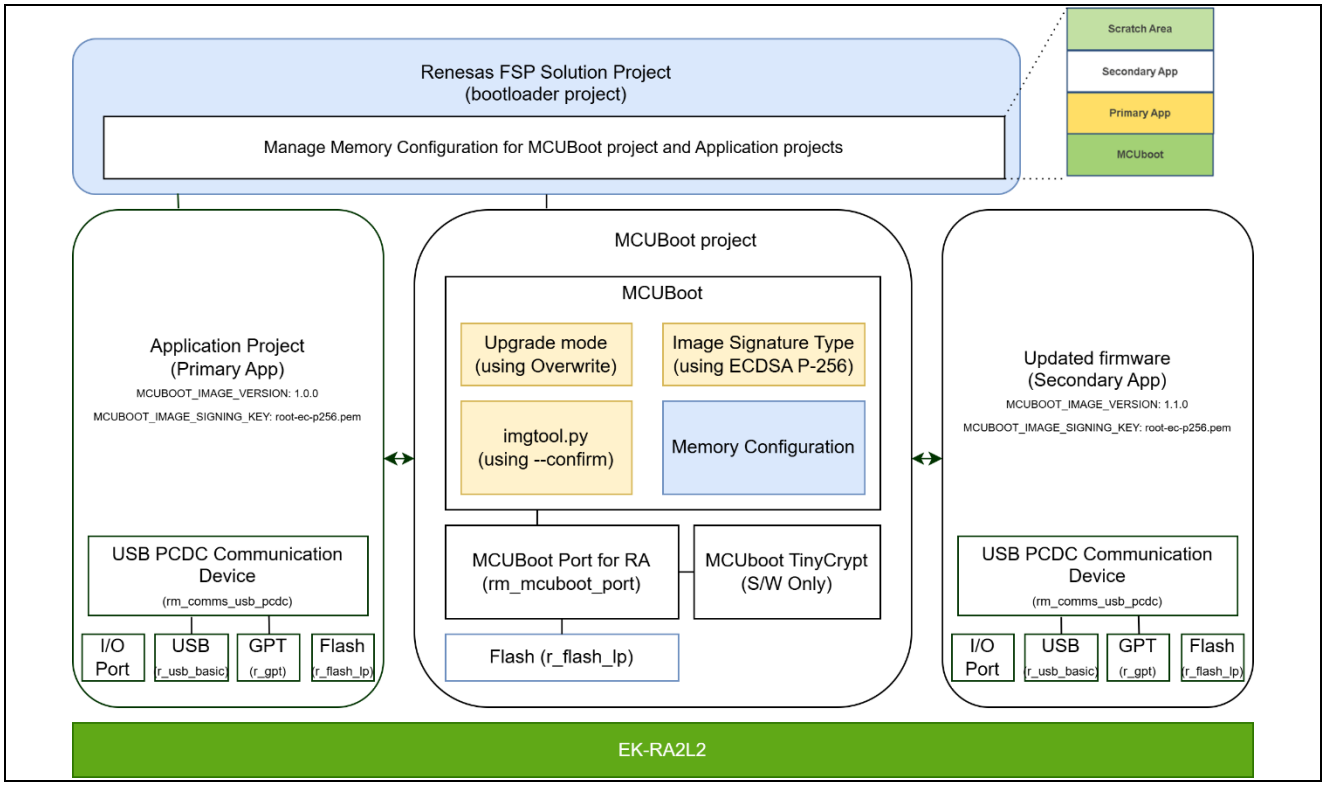


Figure 6. Software Stacks used in EK-RA2L2 sample project

2.4 Internal Code Flash rewrite program via USB CDC overview

Figures 7, 8, 9, 10, 11 & 12 describe internal Code Flash rewrite process for the EK-RA2L2 sample project. This process uses the XMODEM protocol to transfer a new signed image over USB.

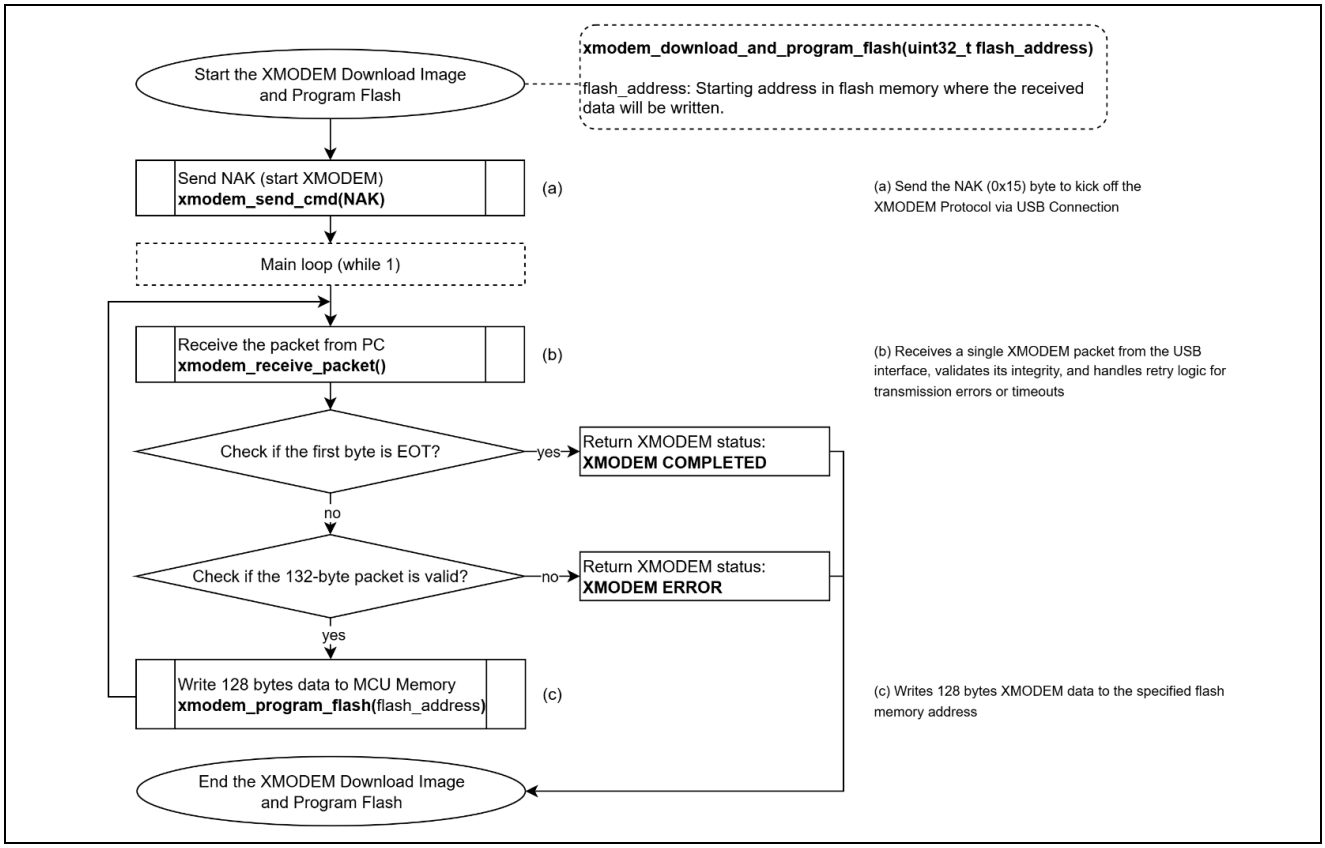


Figure 7. Overall Flow of internal code flash rewrite program

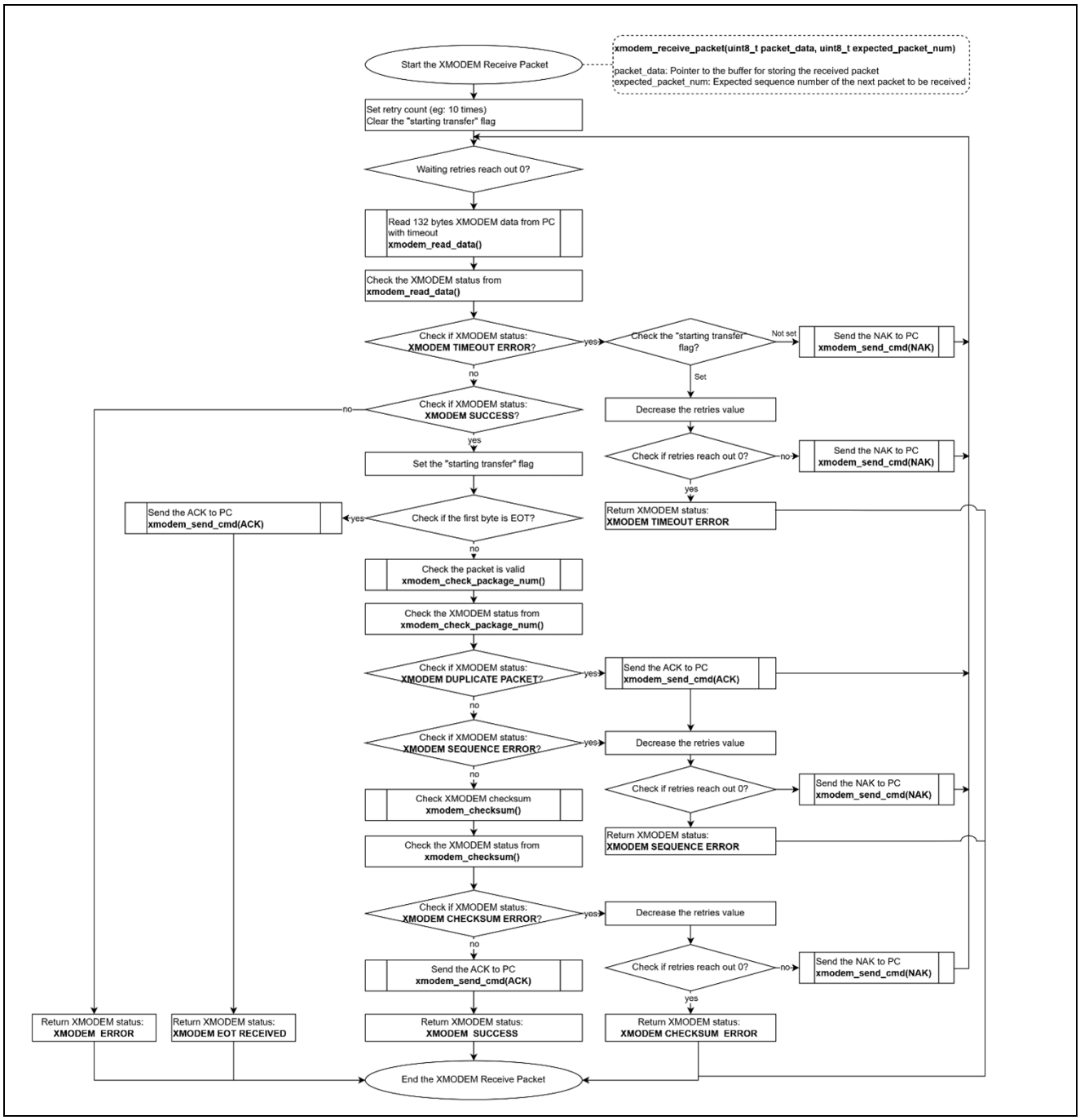


Figure 8. XMODEM Receive Packet Processing

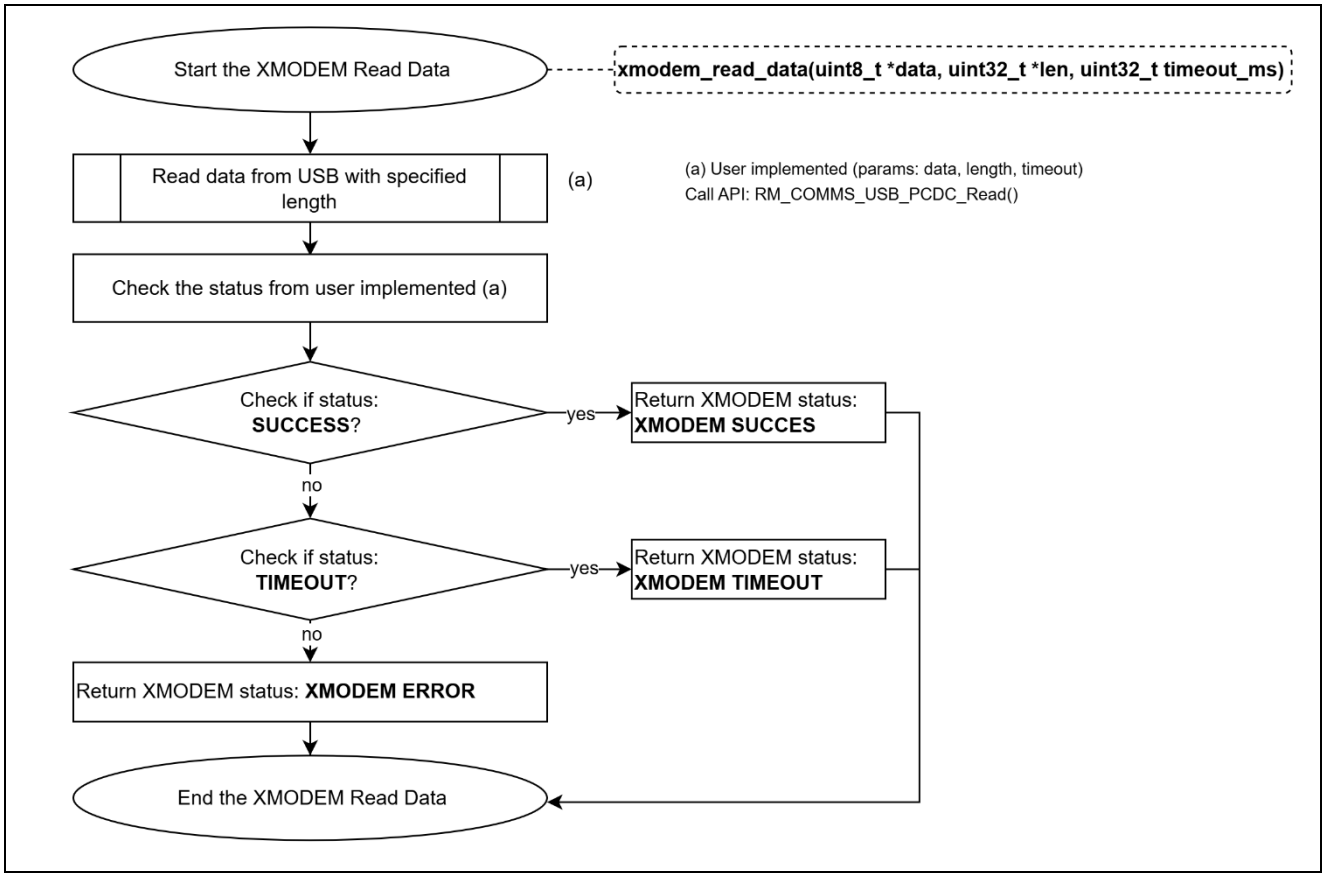


Figure 9. XMODEM Read Data Processing

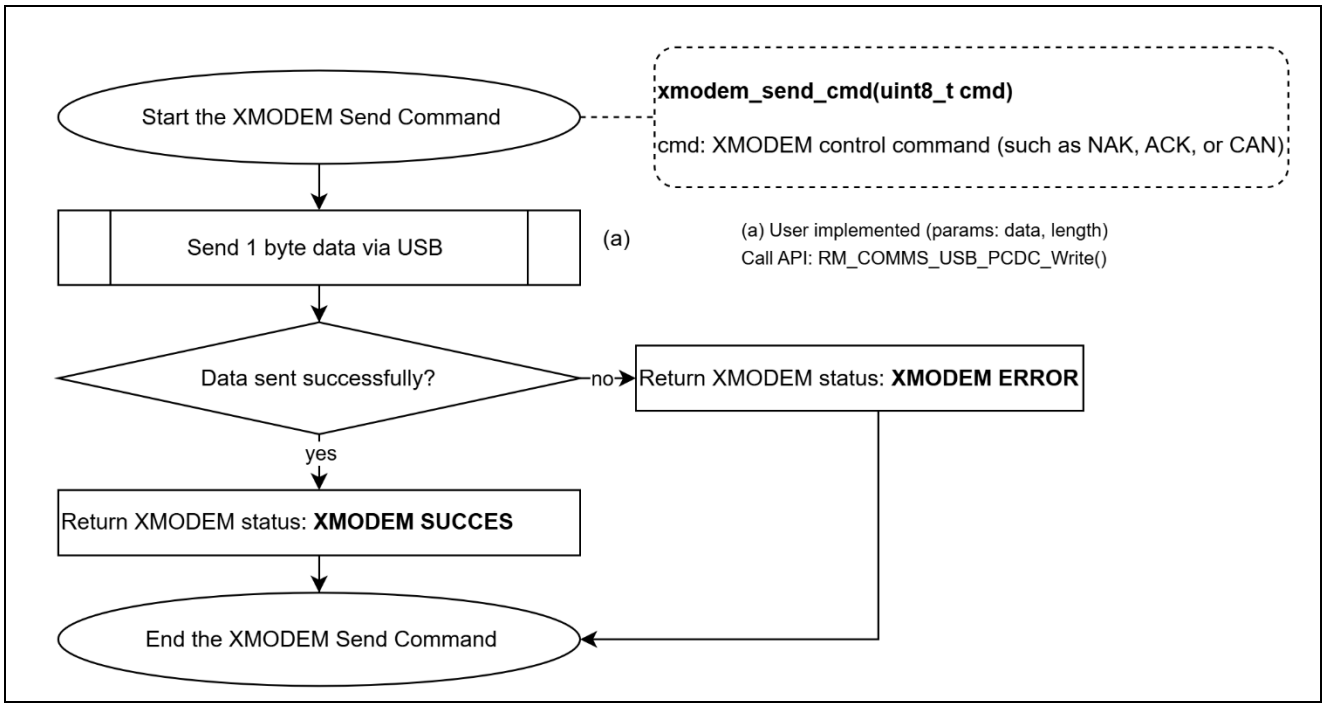


Figure 10. XMODEM Send Command Data Processing

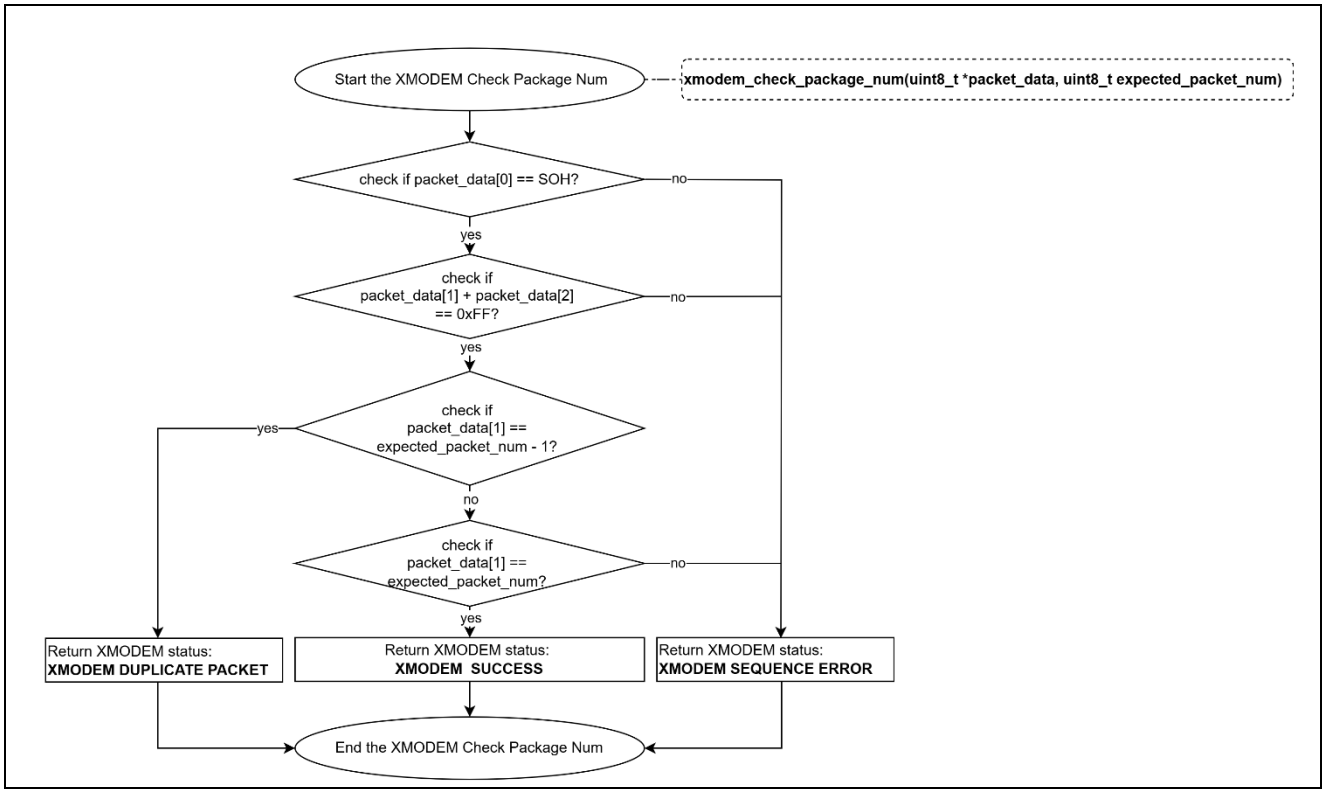


Figure 11. XMODEM Packet Validation

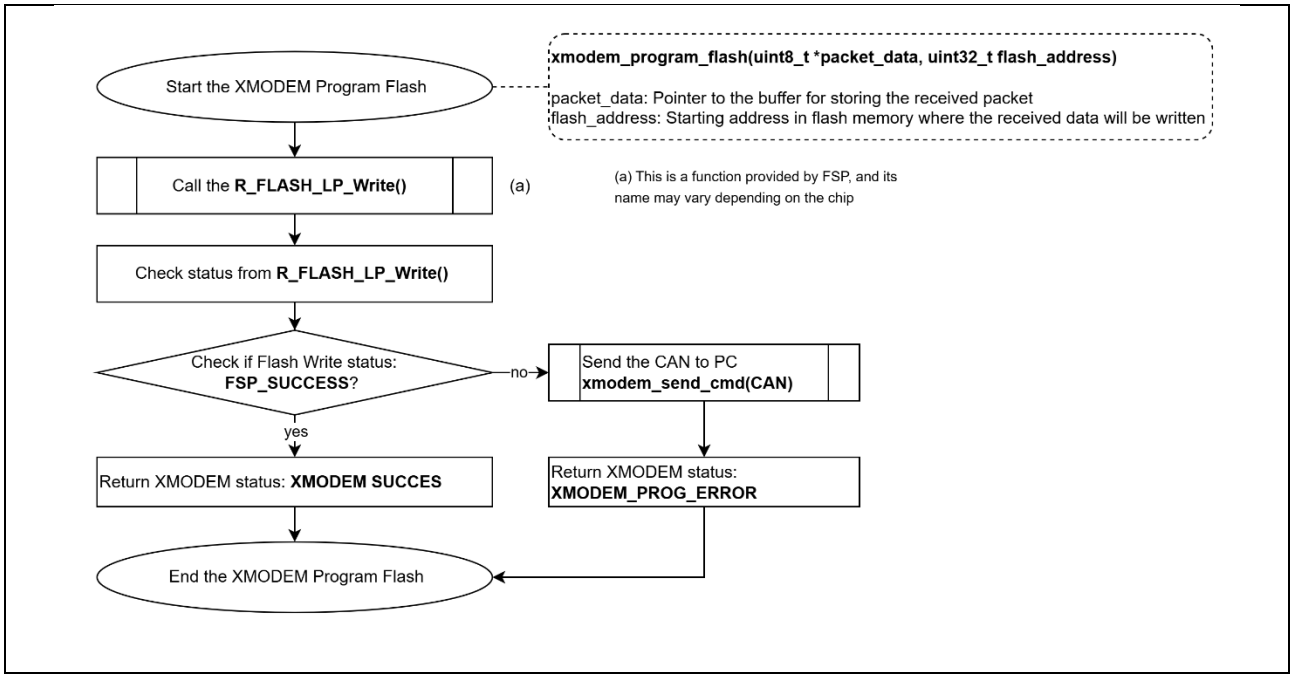


Figure 12. XMODEM Program Flash processing

3. Application Project Package

The Application Project Package includes 2 sample projects for both the EK-RA2L2 and FPB-RA0E1.

ek_ra2l2_firmware_update_via_usb_pcdc.zip	Sample application on EK-RA2L2
fpb_ra0e1_secure_bootloader.zip	Sample application on FPB-RA0E1

Folder Structure of EK-RA2L2 application project.

```
ek_ra2l2_firmware_update_via_usb_pcdc
    |_ mcuboot_overwrite_with_signature_ek_ra2l2
    |_ mcuboot_overwrite_solution_ek_ra2l2
    |_ primary_app_usb_ek_ra2l2
    |_ secondary_app_usb_ek_ra2l2
```

Folder Structure of FPB-RA0E1 application project.

```
fpb_ra0e1_secure_bootloader
    |_ mcuboot_overwrite_with_signature_fpb_ra0e1
    |_ mcuboot_overwrite_solution_fpb_ra0e1
    |_ primary_app_fpb_ra0e1
    |_ secondary_app_fpb_ra0e1
```

Bootloader project

- **mcuboot_overwrite_with_signature_ek_ra2l2** and **mcuboot_overwrite_with_signature_fpb_ra0e1:**
    - MCUboot TinyCrypt (S/W Only)
    - Upgrade mode: Overwrite Only
    - Signing: ECDSA P-256
    - Encryption: Disabled

  Refer to section “4.Creating the Bootloader Project” for creating the MCUboot projects.

- **mcuboot_overwrite_solution_ek_ra2l2** and **mcuboot_overwrite_solution_fpb_ra0e1**:
  Configure for Flash layout which is supported from FSPv6.0.0. Manage for memory of MCUboot projects (mcuboot_overwrite_with_signature_*) and user application projects.

  Refer to section “6. MCUboot Memory Configuration with Renesas FSP Solution Project” for creating solution projects.

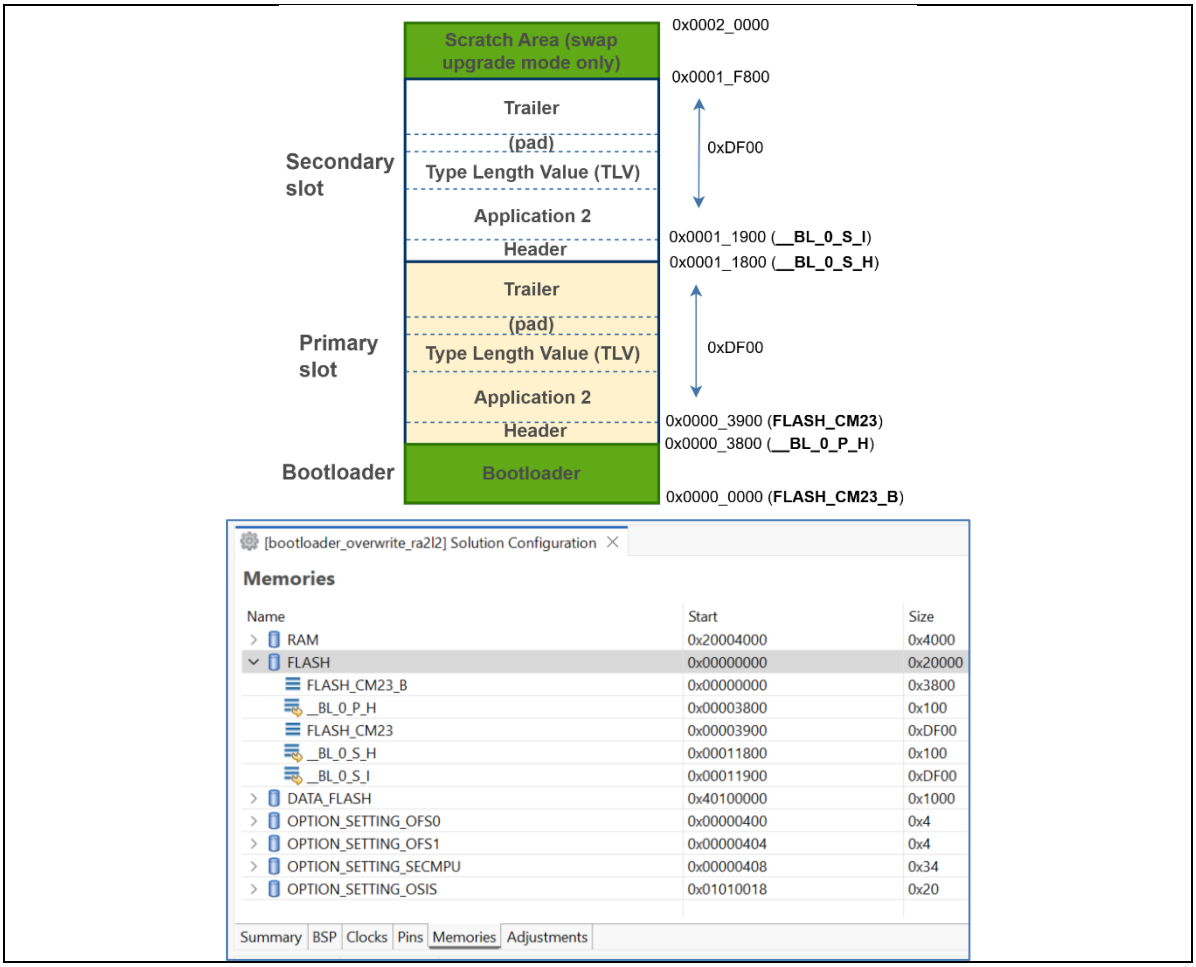
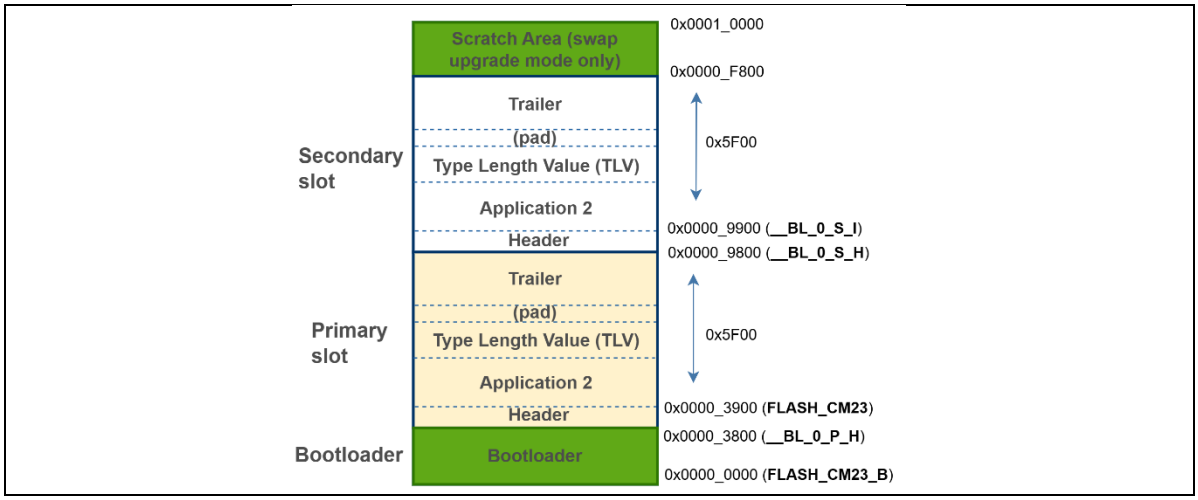


Figure 13. Flash Layout on EK-RA2L2 sample project



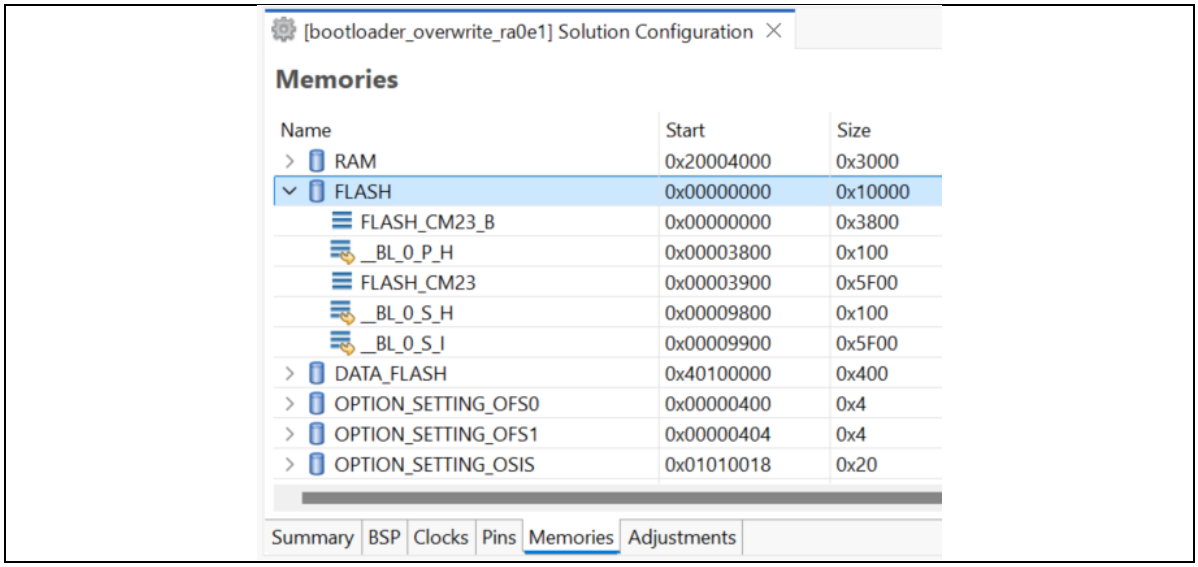


Figure 14. Flash Layout on FPB-RA0E1 sample project

Primary program

- XMODEM downloader via USB PCDC (only available on EK-RA2L2 sample project, refer to 2.3 Internal Code Flash rewrite program via USB CDC overview for implementation).
- LEDs light up in the order (LED1 > LED2 > LED3).

Secondary program

- XMODEM downloader via USB PCDC (only available on EK-RA2L2 sample project, refer to 2.3 Internal Code Flash rewrite program via USB CDC overview for implementation).
- LEDs blink on and off.

Refer to 5. Creating the User Application for creating Primary programs and refer to 8.1 Prepare a Secondary Image for Secondary programs.

Primary/secondary program's structure:		
src		
	xmodem	XMODEM downloader via USB PCDC. Only available on EK-RA2L2 sample codes.
	app_common.c	Implementation for Systick functions which are used as timer for LEDs operation.
	app_common.h	Header of systick functions
	app_led.c	LED implementation.
	app_led.h	Header of LED function.
	hal_entry.c	Default generated of project. hal_entry() is invoked in main(). Call downloader and LED functions.
	hal_warmstart.c	Default generated of project

Structure of xmodem		
xmodem		
	comms.c	Implementation of communication functions using USB PCDC. They will be called in downloader, menu and xmodem.
	comms.h	Headers of communication functions.
	downloader.c	Implement for downloader program.
	downloader.h	Headers of downloader functions.
	info_header.h	Contains macros for image slots, enum and function prototypes used for image slots.
	menu.c	Implementation for menu display in terminal program. Menu display function will be invoked in downloader program.
	r_usb_pcdc_descriptor.c	USB PCDC Descriptor definition.
	xmodem.c	Implementation for XMODEM protocol. They will be called in downloader program.
	xmodem.h	Contains XMODEM protocol macro definitions and function definitions.

4. Creating the Bootloader Project

4.1 Including the MCUboot Module in the Bootloader Project

1. Launch e² studio and start to create a new C/C++ Project.
   Click **File > New > Renesas C/C++ Project > Renesas RA.**

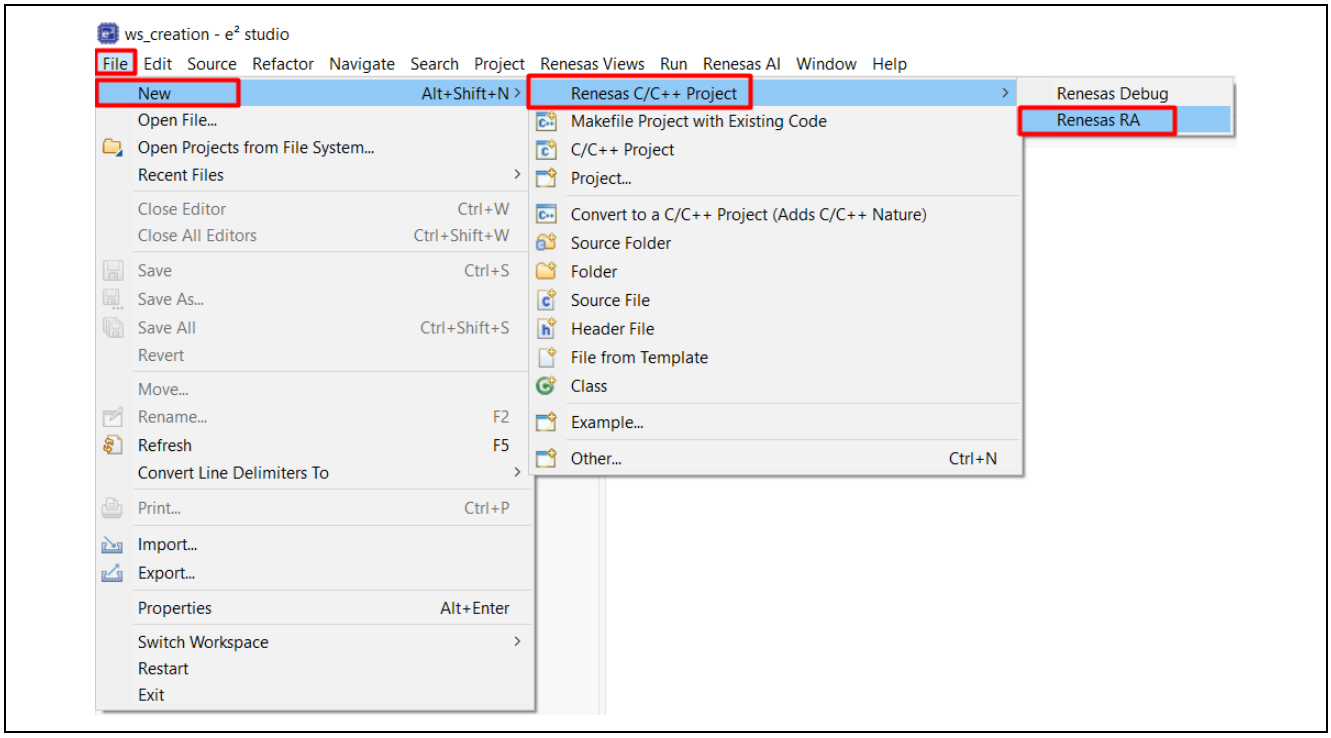


Figure 15. Start a New Project

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

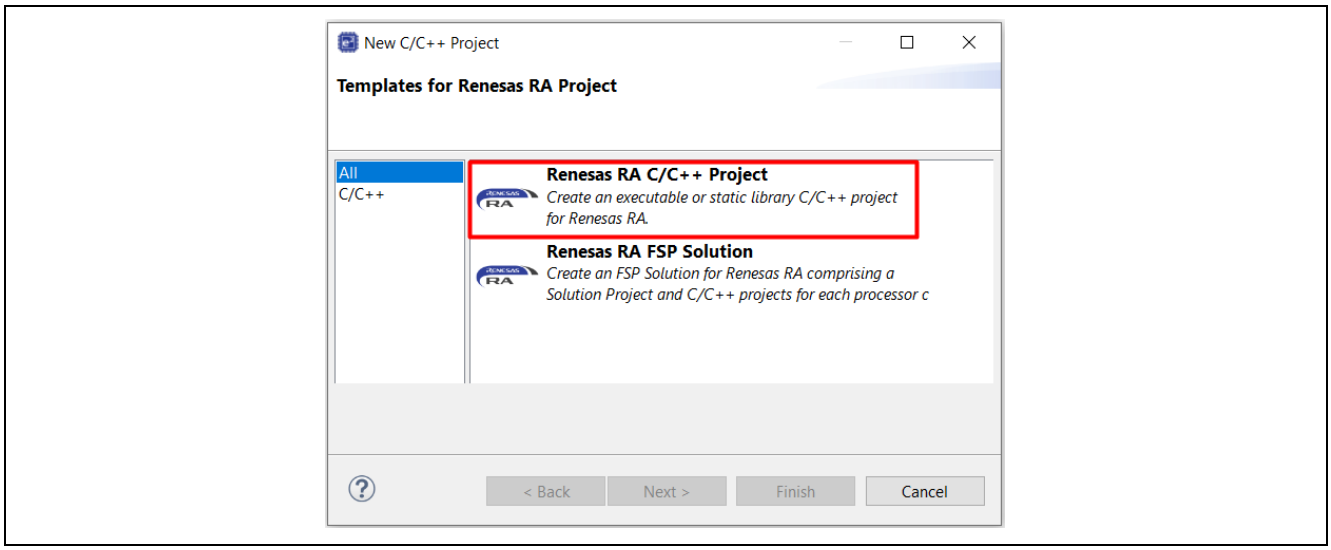


Figure 16. Choose Renesas RA C/C++ Project

3. Provide the project name **mcuboot_overwrite_with_signature_ek_ra2l2**. Click **Next**.
   Project name based on the upgrade mode and authentication method. The name will persist in the instructions used in this application note. Note that magic number and SHA256 integrity check are included in all the systems.

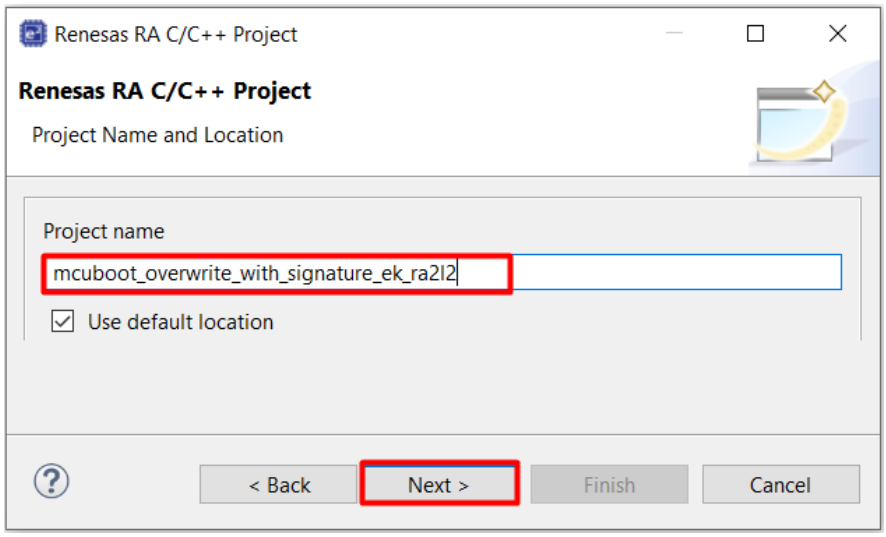


Figure 17. Name the Bootloader Project

If a different name for the bootloader is chosen, then adapt the corresponding instructions in this application note accordingly.

4. Select **FSP Version 6.1.0**, **Board EK-RA2L2** and **Toolchains LLVM.** Click **Next**.

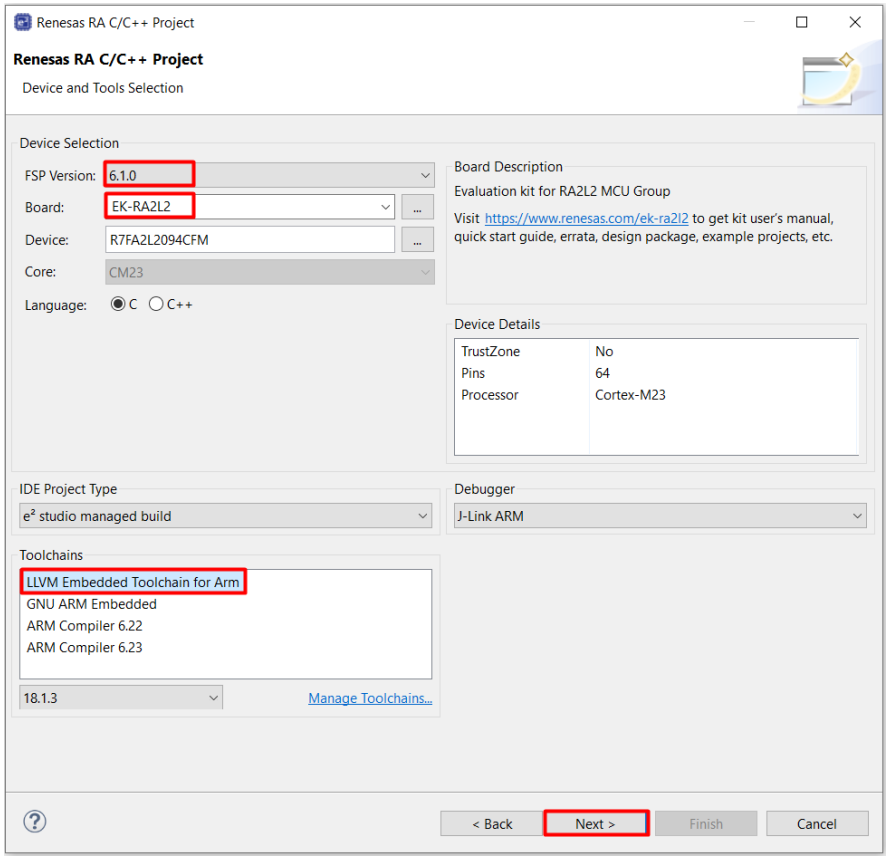


Figure 18. Select the FSP, Board & Toolchain

5. Select **None** for Preceding Project or Smart Bundle Selection screen. Click **Next**.

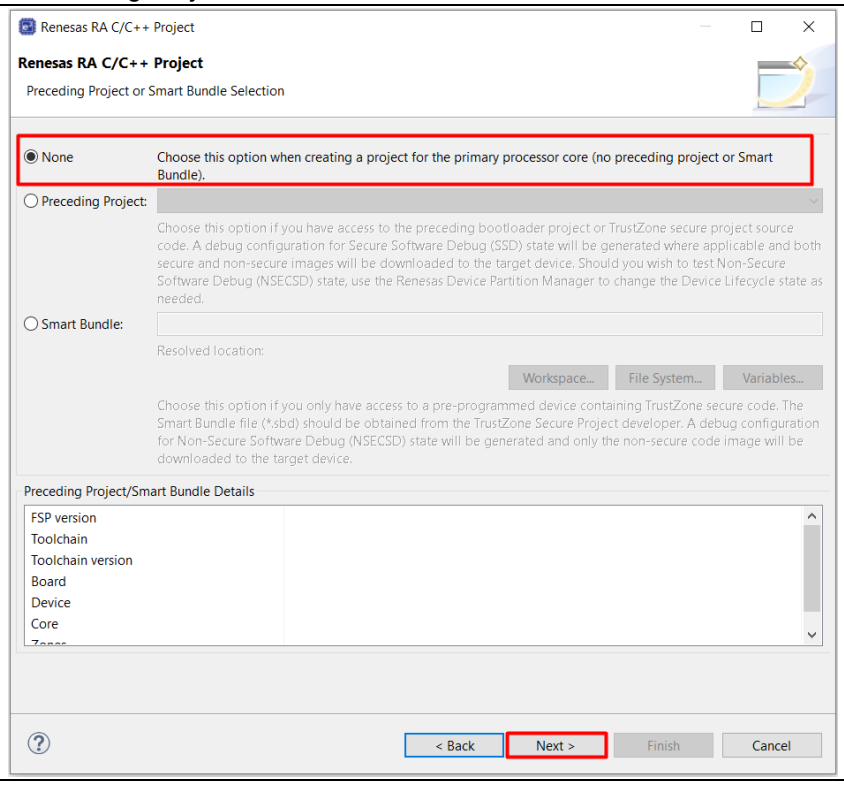


Figure 19. Select the None option for Preceding Project or Smart Bundle Selection screen

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

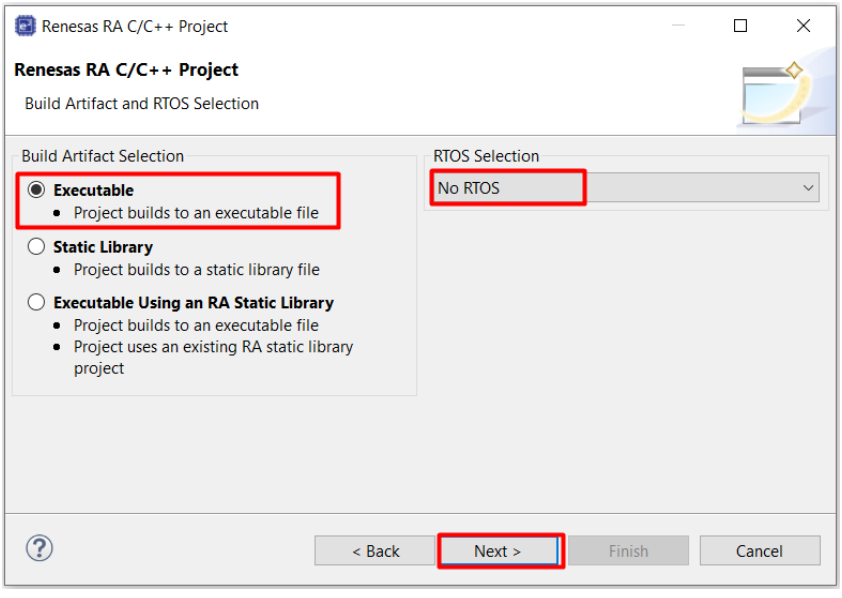


Figure 20. Choose to Build Executable and No RTOS

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

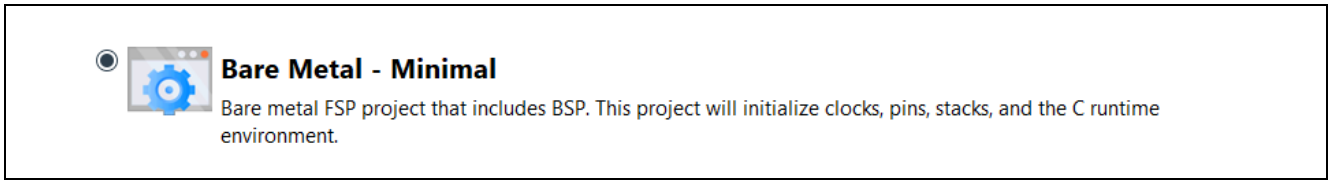


Figure 21. Choose the Project Template

8. If prompted, click **Open Perspective**.

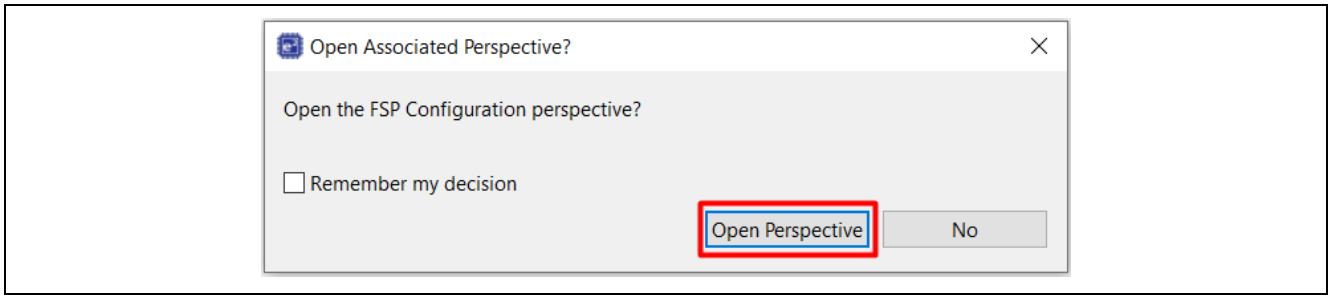


Figure 22. Choose Open the FSP Configuration Perspective

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

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

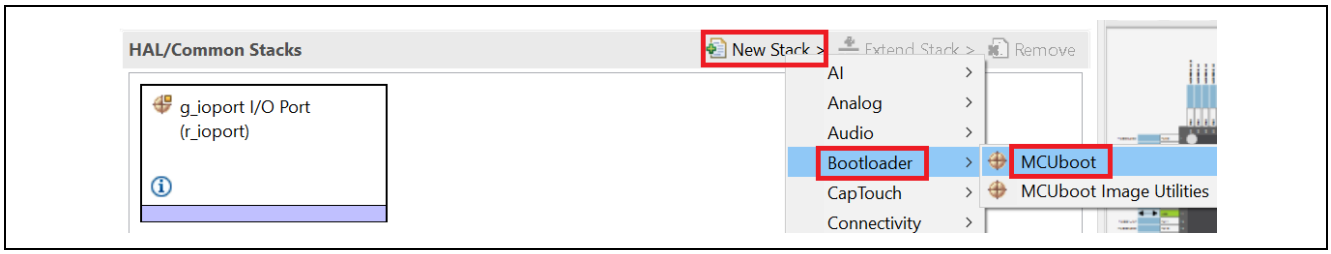


Figure 23. Add the MCUboot Port

10. Configure the **General** properties of **MCUboot**, as shown in Figure 24.

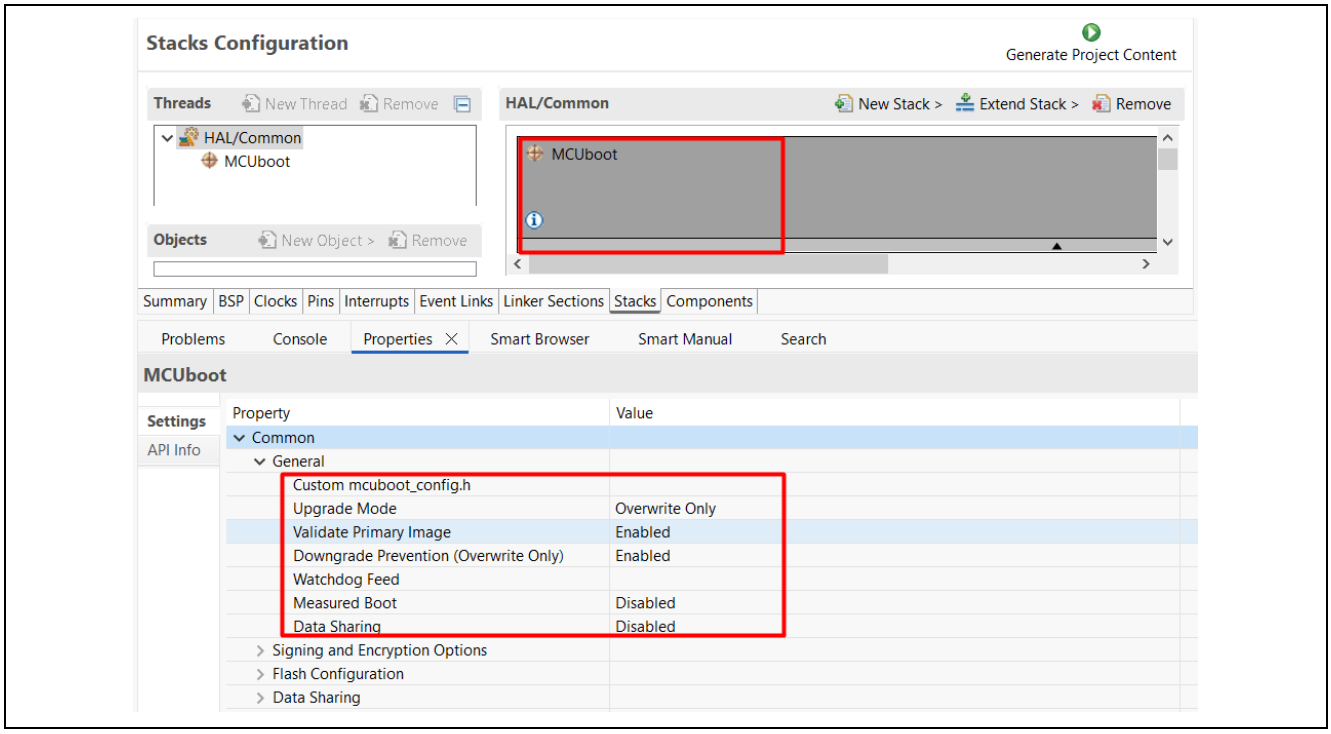


Figure 24. General Properties for MCUboot with Overwrite Upgrade Mode

Figure 25 shows a more detailed application image format that can be referenced to understand the various MCUboot property definitions.

- The header magic number is used for image validation sanity check (refer to the description of **Validate Primary Image).**
- The **image_ok** byte is a flag used by the bootloader for swap test mode confirmation.
- The trailer magic number is written after the image upgrade is finished.

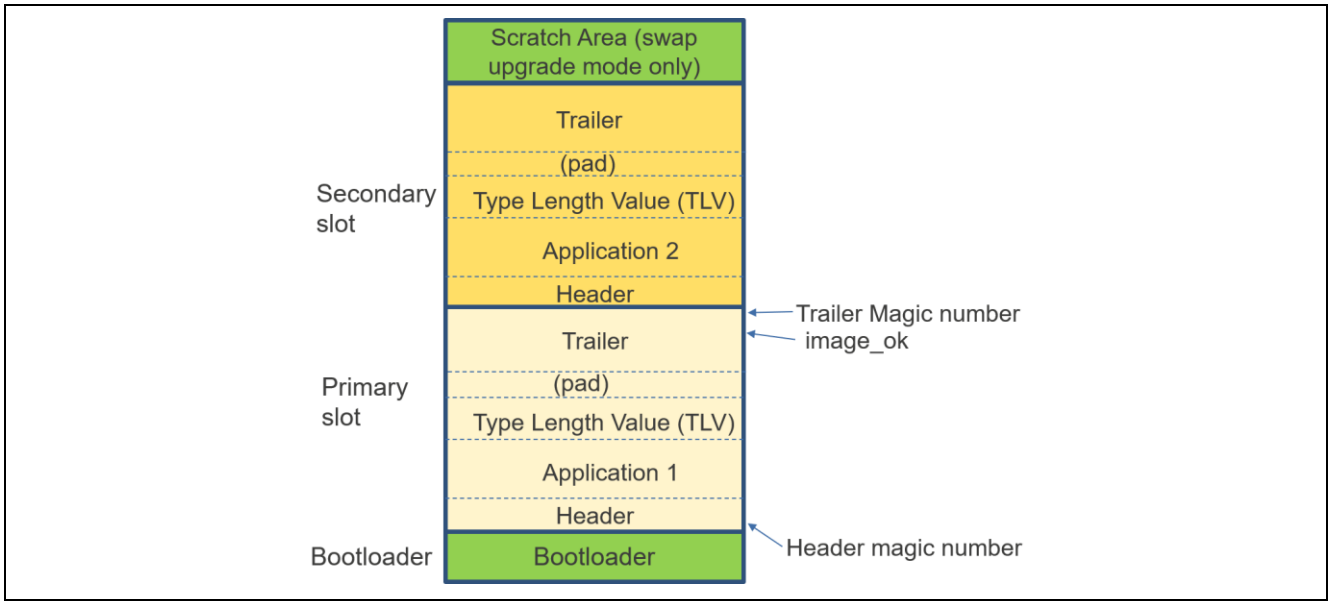


Figure 25. General Configuration for MCUboot Module

The MCUboot General properties are:

- **Custom mcuboot_config.h**: The default mcuboot_config.h file contains the MCUboot Module configuration that you select from the RA configurator. You 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.
- **Validate Primary Image**:
  When Enabled
  - The bootloader performs:
    - Hash or signature verification (depending on the selected verification method).
    - MCUboot sanity check (based on the image header magic number).
  - The header magic number is always checked as part of the sanity checking prior to the integrity checking and the signature verification.

  When Disabled
  - Only sanity check is performed based on the MCUboot header magic numbers.
- **Note**
  - It is **strongly recommended** to enable this property, as it adds critical security handling to the bootloader while requiring less than 32 bytes of additional code.
  - The image magic number is not part of the image validation.
  - It is a **reference value**, useful for sanity checks during the application upgrade debugging process.
  - The image magic number is written to flash only after a successful image upgrade.
- **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. To see how to set the version number of an image, refer to Figure 63.

11. Configure the **Signing and Encryption Options**

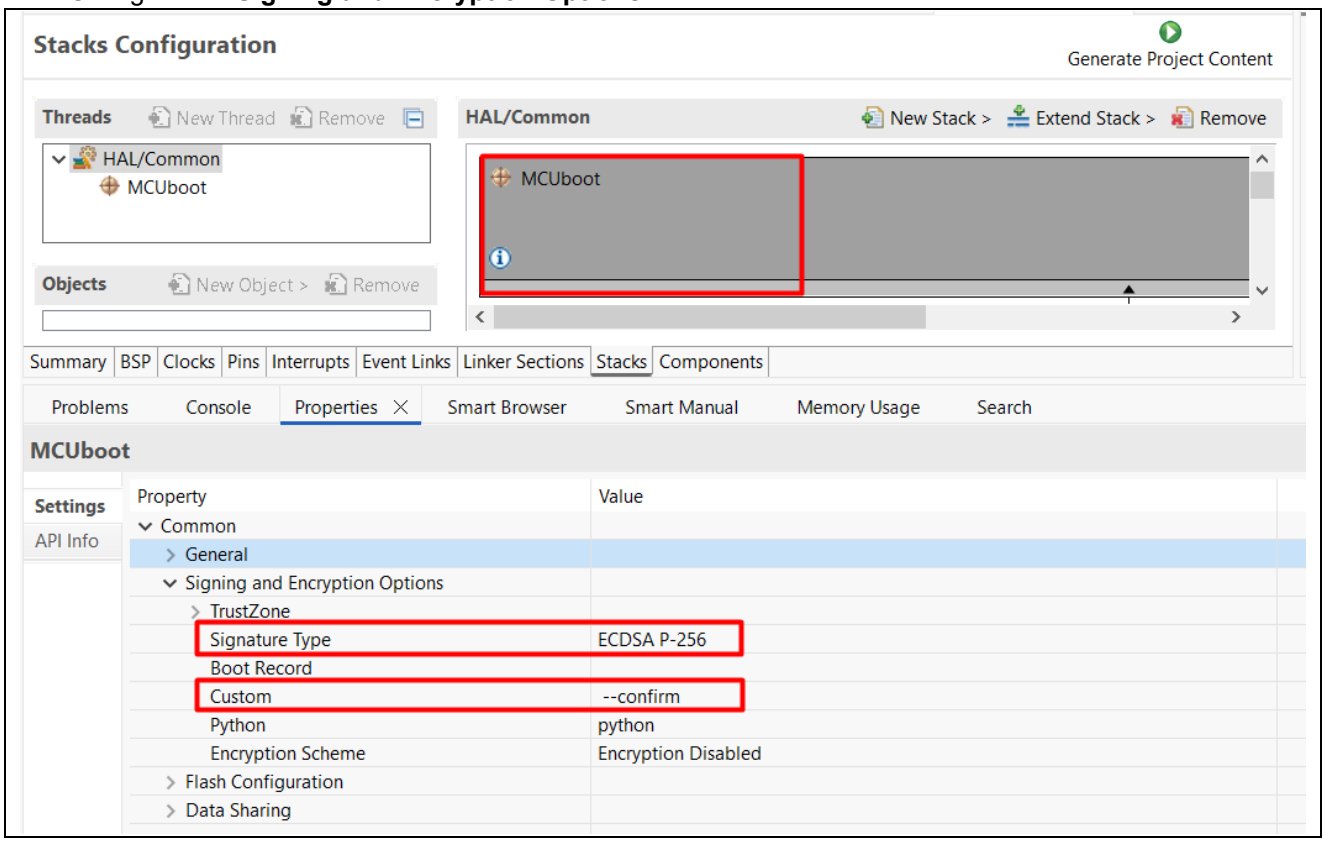


Figure 26. Application Image Signature Type and Signature Options

The MCUboot Signing and Encryption properties are:

For both single-image and two-image configurations, the following properties need to be defined:

- **Signature Type** is the signing algorithm selection. 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.
- Note that when using Ocrypt as the cryptographic support for MCUboot, RSA signature verification is not supported. The choices are:
  - **NONE:** This option is selected for the bootloaders that do not support signature verification
  - **ECDSA P-256**: This option is selected for the example bootloaders that support signature verification included in this application project.
  - **RSA 2048 and RSA 3072:** Not supported.
- **Custom:** This property allows you to input any specific arguments for the signing command. By default --confirm is set for this property, which has the following influence on the Secondary image:
  - For Overwrite upgrade mode, the new image will always overwrite the original application image upon successful verification.
  - For Swap upgrade mode, the Primary image slot will be marked as Confirmed after the swap update. No swap happens upon the next reset after the swap update.

  If the **Custom** property is set to --pad, the system behavior is:
  - For Overwrite upgrade mode, the system behavior is same as when --confirm is set.
  - For Swap upgrade mode, the system behavior depends on whether the application has routines to mark the Primary image slot as Confirmed.

  Primary image boot behavior is not influenced by the choice between --confirm or --pad.

Note that the **Signature Type** is set to **NONE**, the bootloader size will be decreased.

The properties under **TrustZone are not used for RA2 MCUs since they do not have TrustZone.** For other properties shown in this step, refer to the *FSP User's Manual* section on MCUboot port.

12. Add the **TinyCrypt** module under **MCUboot Port for RA**. **TinyCrypt (S/W Only)** is used. The **MbedTLS (Crypto Only)** module has a larger memory footprint compared with **TinyCrypt** and is not used in this bootloader application.

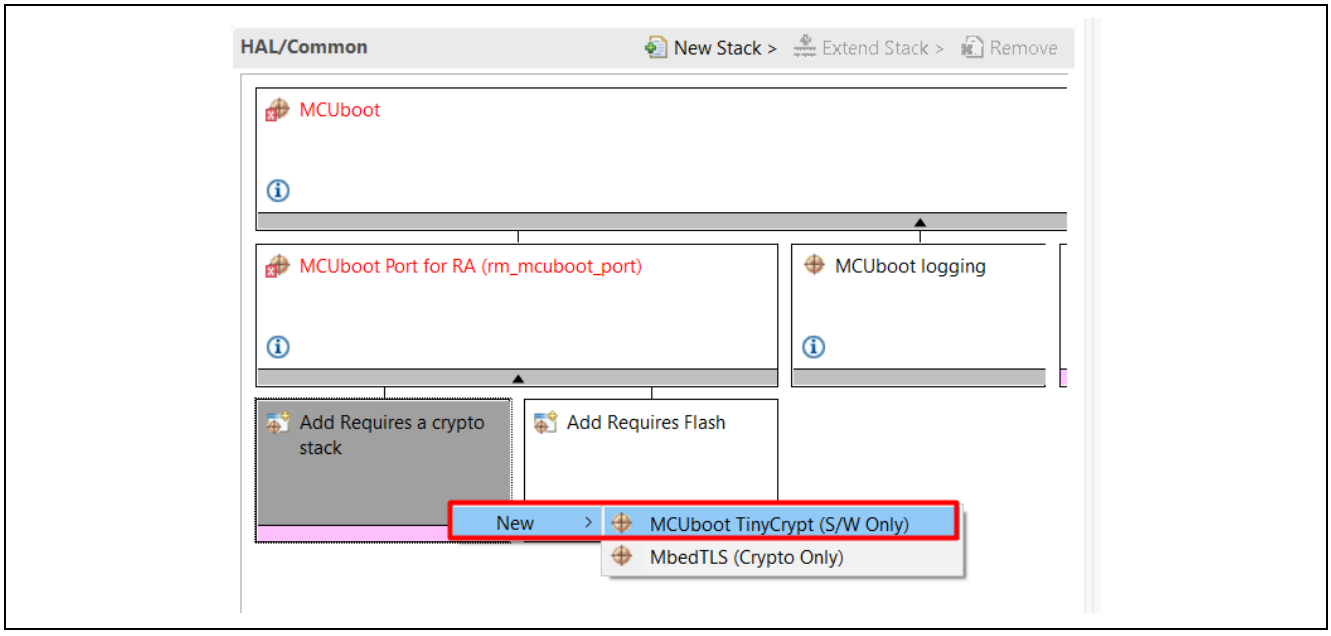


Figure 27. Select TinyCrypt Module

13. If creating a bootloader with signature verification support, then the **ASN.1 Parser** stack and the **MCUboot Example Keys** stack are required.

Click on the **Add ASN.1 parser** stack and select **New** to add the ASN.1 Parser.

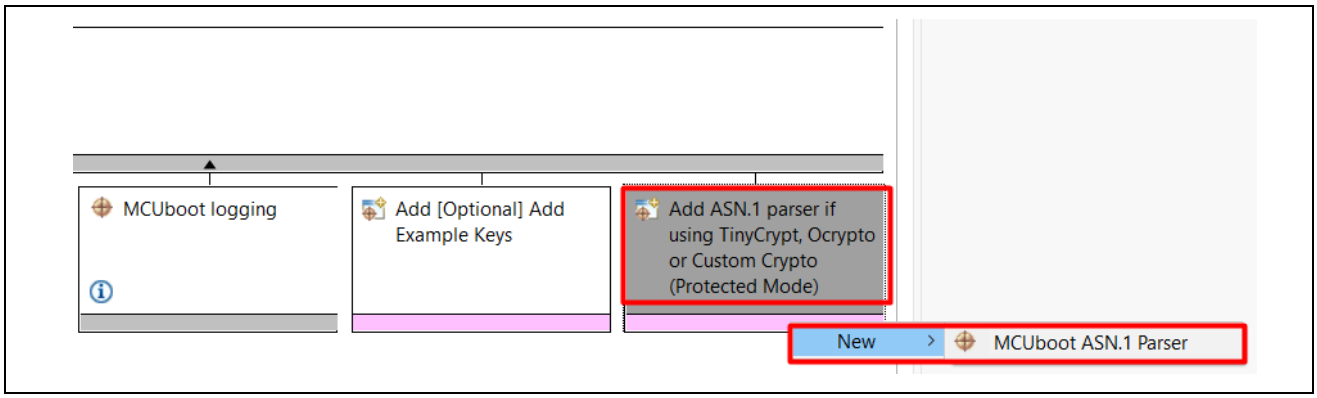


Figure 28. Add the ASN.1 Parser

Click on the **Add [Optional] Add Example Keys** stack and choose **New** to add **MCUboot Example Keys (NOT FOR PRODUCTION)**.

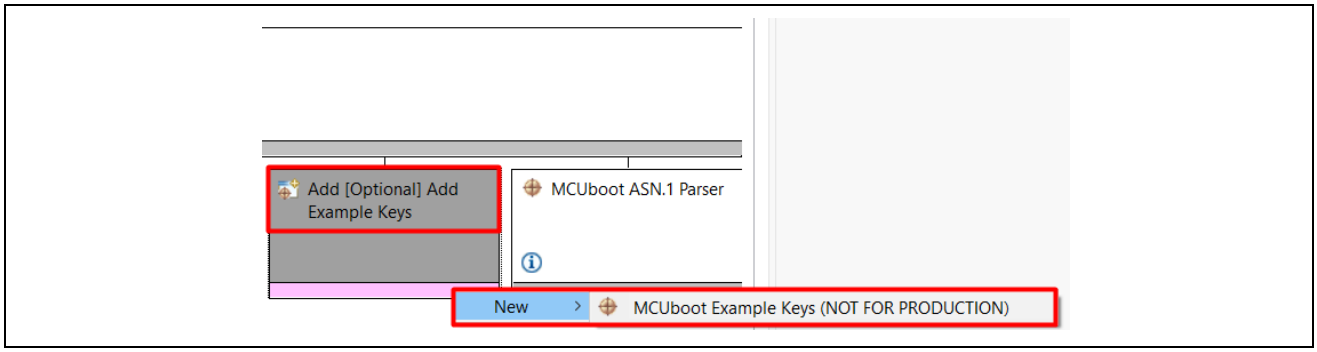


Figure 29. Add the Example Image Signing Key

Note: The MCUboot Example Keys are open to public access from the MCUboot port.

Users should not use them for production purposes.

14. Update the **BSP Main Stack size** to **0x800**.

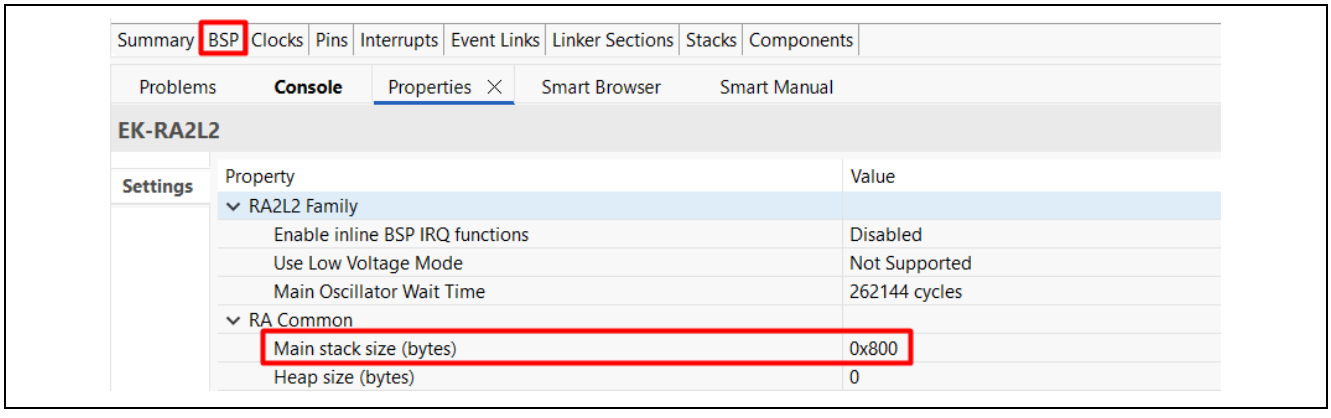


Figure 30. Update the BSP Main Stack Size

15. Click on **Add Required Flash** stack and add **Flash (r_flash_lp)**.
16. Click on the **Flash Driver** block and set the **Code Flash Programming** to **Enabled**. As **Data Flash Programming**, **Data Flash Background Operation Support,** and **Data Flash Background Operation**

are not used in the bootloader, select **Disabled** for these two properties to reduce the bootloader memory footprint.

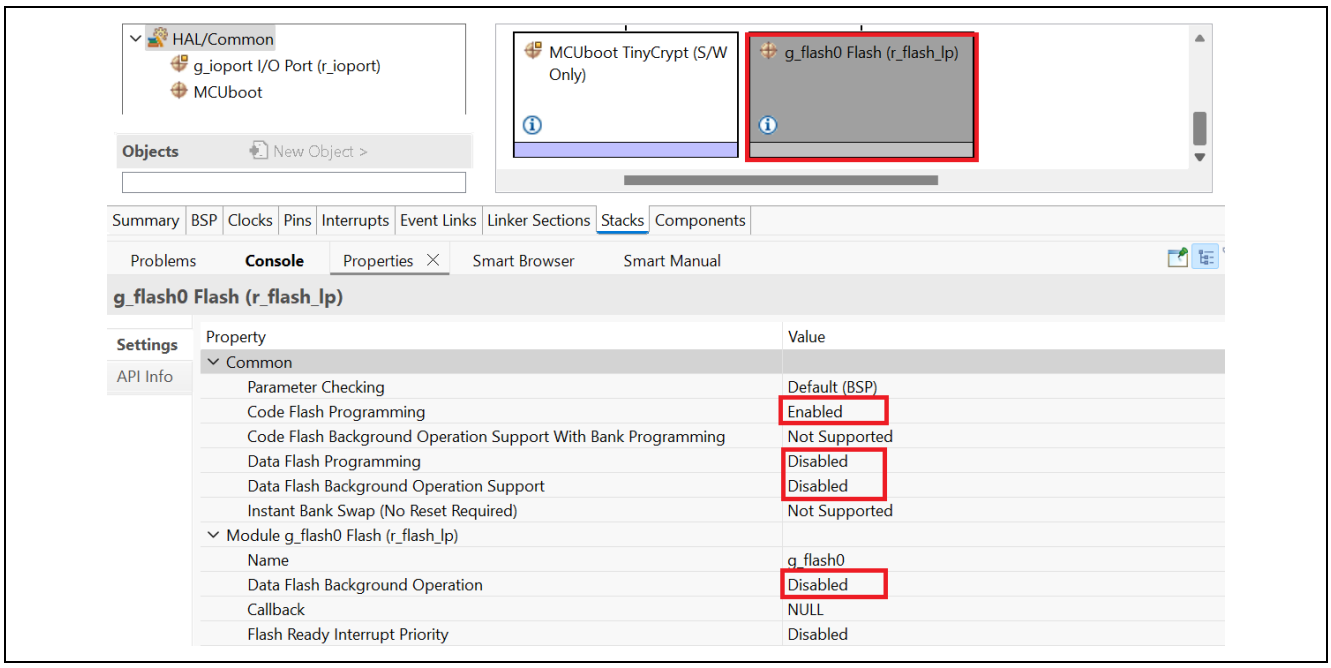


Figure 31. Enable Code Flash programming

17. Save configuration.xml and click **Generate Project Content**.
    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.

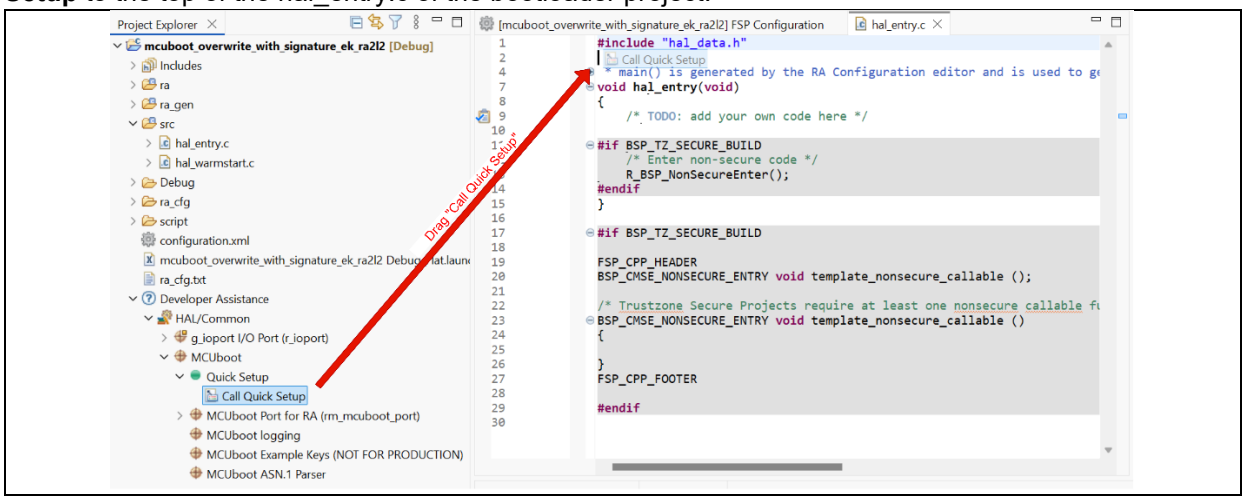


Figure 32. Add mcuboot_quick_setup() using Developer Assistance

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

Figure 33. Add function call to hal_entry()

18. By default, the **I/O Port** Driver is brought into the project when the project is created. As the **I/O Port Driver** is not used in the bootloader project, this stack can be removed to reduce the bootloader project size. Right click on the **I/O Port** stack and choose **Delete**.

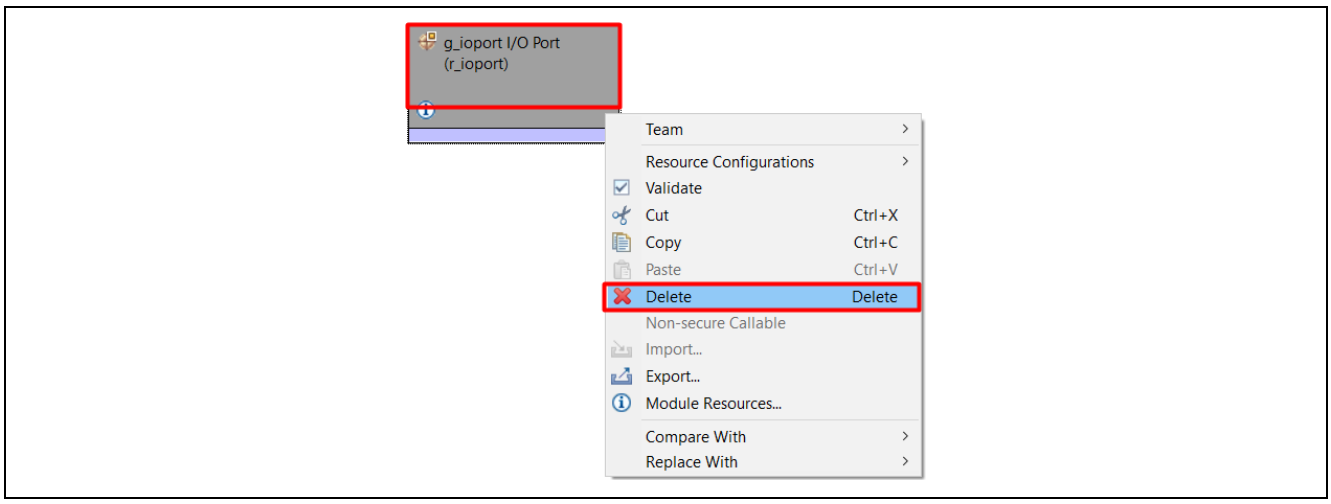


Figure 34. Remove the I/O Port Stack

After the I/O Port is deleted, remove all sections of code referencing the I/O Port API. Likewise, since data flash is unused, remove any code segments referencing data flash. For example, in this application remove the two sections of the code in the red boxes in the function R_BSP_WarmStart in hal_warmstart.c file, as shown in Figure 35.

```
/*
* Copyright (c) 2020 - 2025 Renesas Electronics Corporation and/or its affiliates
*
* SPDX-License-Identifier: BSD-3-Clause
*/

#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);

FSP_CPP_FOOTER

 * This function is called at various points during the startup process.  This implementation uses the event that is
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable here, before clock and
         * C runtime initialization, should negate the need for a delay since the initialization will typically take more than 6us. */
#endif
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open(&IOPORT_CFG_CTRL, &IOPORT_CFG_NAME);

#if BSP_CFG_SDRAM_ENABLED

        /* Setup SDRAM and initialize it. Must configure pins first. */
        R_BSP_SdramInit(true);
#endif
    }
}
```

Figure 35. Remove Unused Code in hal_entry.c

Note: All of the preceding steps are valid for FPB-RA0E1 board.

4.2 Further Optimizing for the Bootloader Project Size

To further optimize the bootloader project for code size, some of the application code can be placed in the gap area between the interrupt vector and the RA2L2/RA0E1 ROM registers (Option-Setting Memory Registers). The section (.flash_gap) in the linker script can be used to store some application code.

Note that the bootloader image size optimization methods introduced in this section apply to any application project, regardless of whether a bootloader is used. Users can use the methods described in this section to save code space for any RA2/RA0 application.

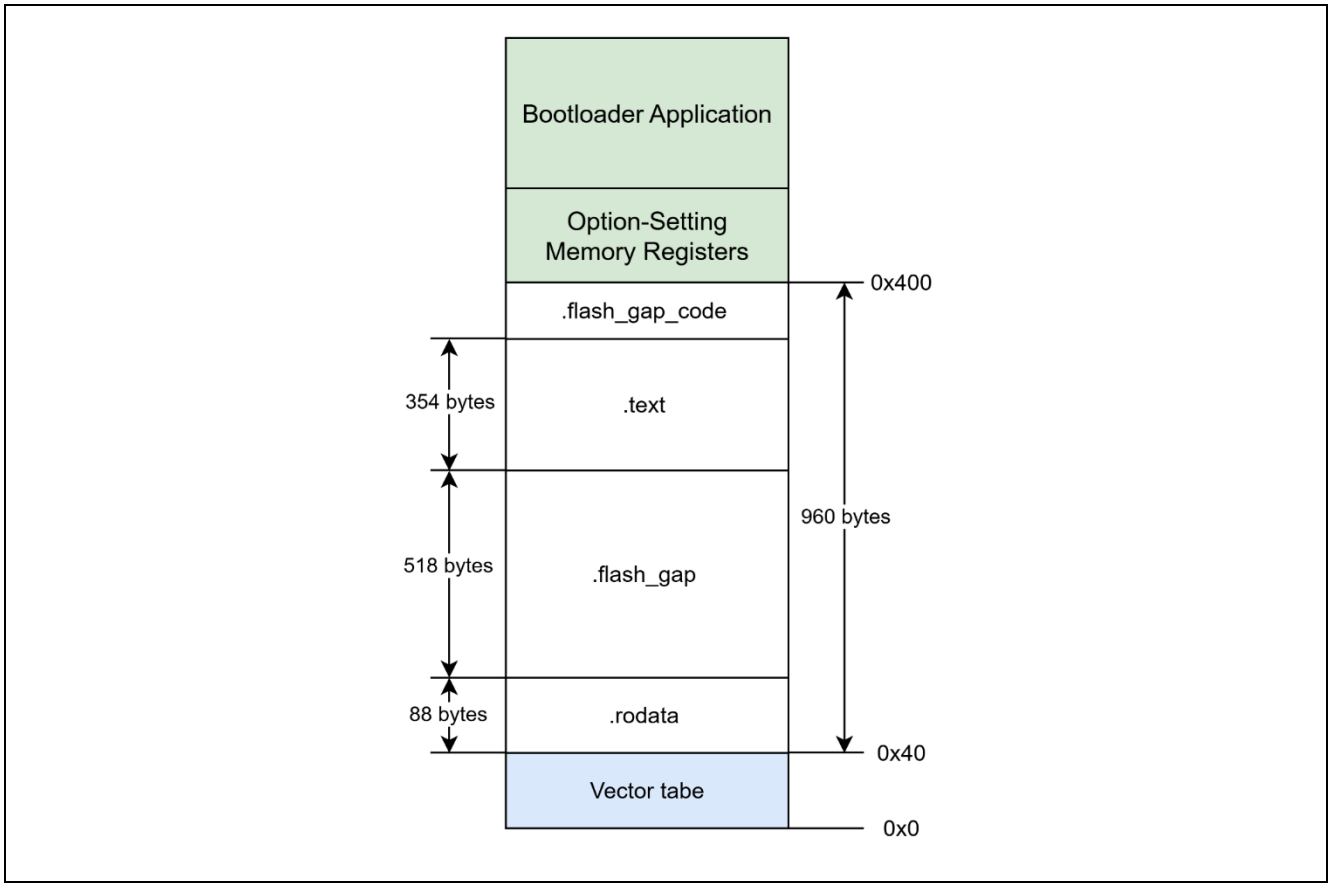


Figure 36. Example: First Flash Sector in Overwrite Update Mode for RA2L2

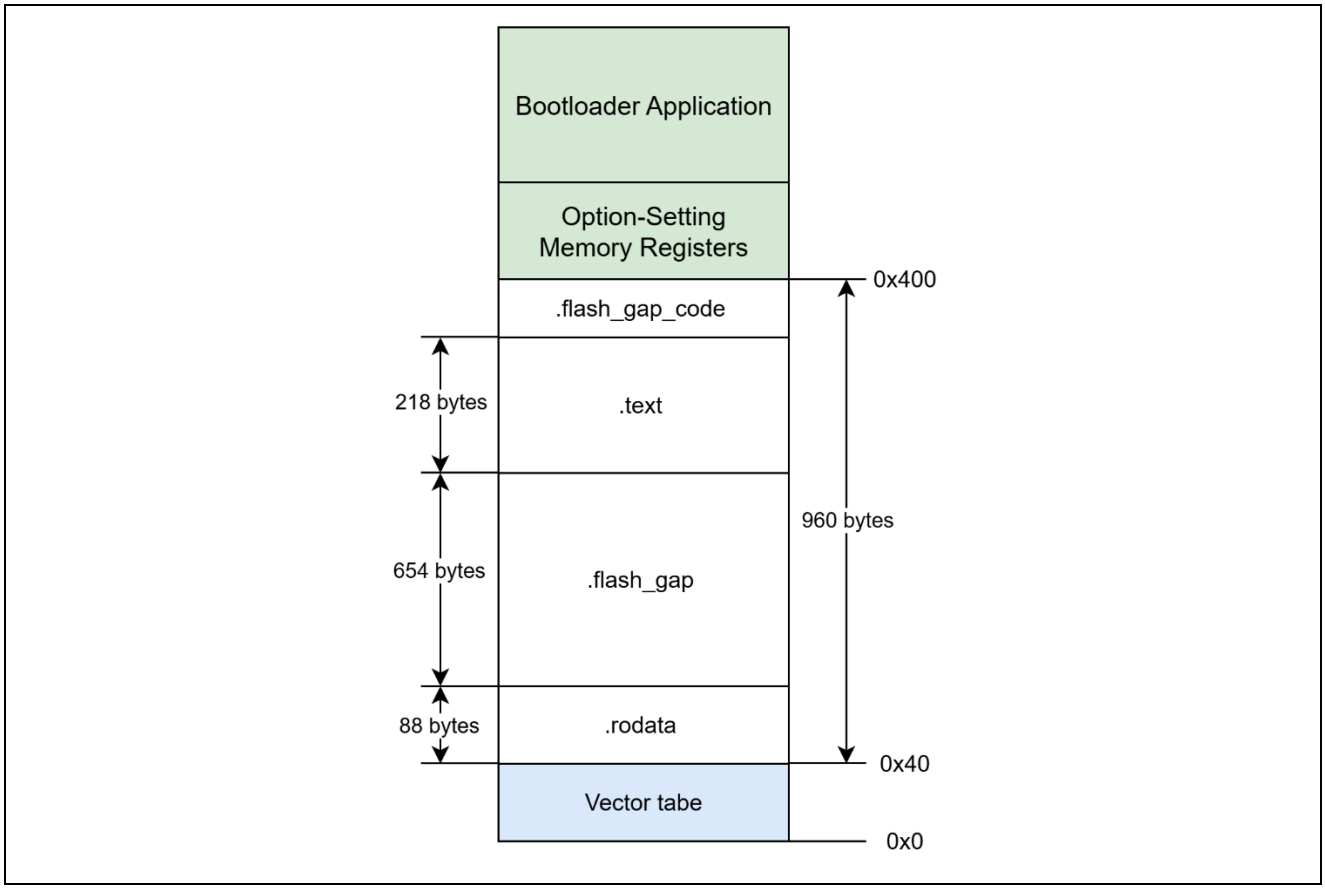


Figure 37. Example: First Flash Sector in Overwrite Update Mode for RA0E1

The following sections detail how the RA2L2 MCUboot project can be optimized for code size, utilizing the .flash_gap or .flash_gap_code section. The following steps can be used for the RA0E1 project.

Which functions are placed in the .flash_gap / .flash_gap_code section are specified by the developer.

It is recommended that only one section, either .flash_gap or .flash_gap_code, is used to make simplify the memory management.

Depending on which Toolchain and Toolchain settings (optimization) are made, the size of .flash_gap, is approximately 518 bytes. The size of the section can be seen using the **Memory Usage Window** in e^2 studio.

To launch the **Memory Usage Window**, click **Window > Show View > Other… > C/C++ > Memory Usage,** as shown in Figure 38.

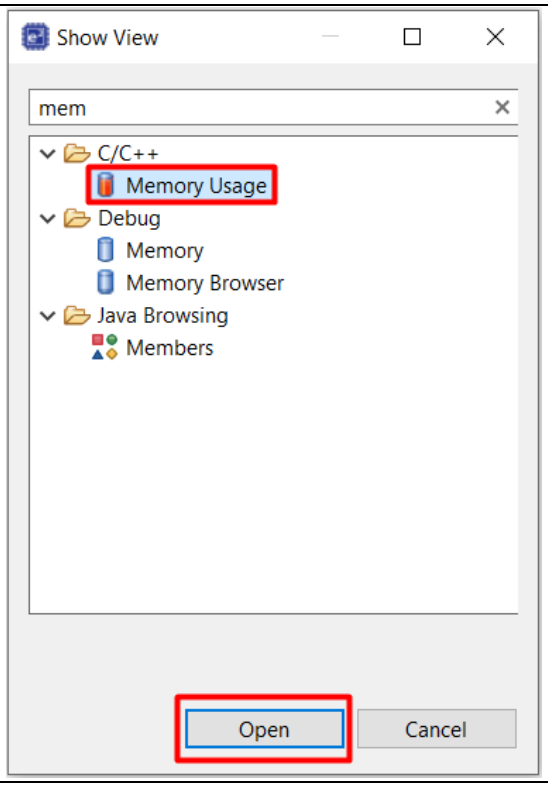


Figure 38. Open the Memory Usage

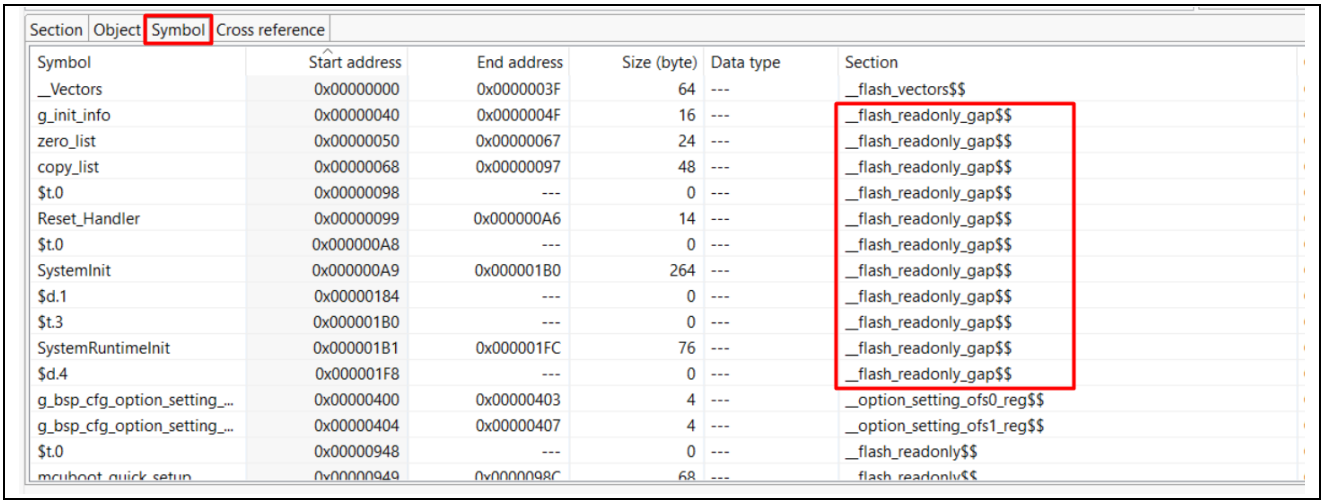


Figure 39. Memory Usage View

The **Memory Usage** view can be used to identify and select functions of a suitable size to place in the .flash_gap section. Based on the **Memory Usage** results, the following functions where placed into the .flash_gap section, as shown in Figure 40.

In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\bootutil.h

```
int bootutil_tlv_iter_begin(struct image_tlv_iter *it,
                            const struct image_header *hdr,
                            const struct flash_area *fap, uint16_t type,
                            bool prot) BSP_PLACE_IN_SECTION(".flash_gap");
```

In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\src\bootutil_priv.h

```
fih_ret bootutil_verify_sig(uint8_t *hash, uint32_t hlen, uint8_t *sig,
```

```
                size_t slen, uint8_t key_id) BSP_PLACE_IN_SECTION(".flash_gap");

In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\bootutil_public.h
int boot_read_swap_state_by_id(int flash_area_id, struct boot_swap_state *state) BSP_PLACE_IN_SECTION(".flash_gap");
```

Figure 40. Functions to put in the .flash_gap section in Overwrite Update Mode for RA2L2

```
In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\image.h
fih_ret bootutil_img_validate(struct enc_key_data *enc_state, int image_index,
                              struct image_header *hdr,
                              const struct flash_area *fap,
                              uint8_t *tmp_buf, uint32_t tmp_buf_sz,
                              uint8_t *seed, int seed_len, uint8_t *out_hash) BSP_PLACE_IN_SECTION(".flash_gap");

In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\src\bootutil_priv.h
int boot_copy_region(struct boot_loader_state *state,
                     const struct flash_area *fap_src,
                     const struct flash_area *fap_dst,
                     uint32_t off_src, uint32_t off_dst, uint32_t sz) BSP_PLACE_IN_SECTION(".flash_gap");

In \ra\mcu-tools\MCUboot\boot\bootutil\include\bootutil\bootutil_public.h
int boot_read_image_ok(const struct flash_area *fap, uint8_t *image_ok) BSP_PLACE_IN_SECTION(".flash_gap");
```

Figure 41. Functions to put in the .flash_gap section in Overwrite Update Mode for RA0E1

After placing functions into .flash_gap section, check the function location using the **Memory Usage** window.

Section | Object | Symbol | Cross reference

Symbol	Start address	End address	Size (byte)	Data type	Declarati...	Section
__Vectors	0x00000000	0x0000003F	64	---	---	__flash_vectors$$
g_init_info	0x00000040	0x0000004F	16	---	---	__flash_readonly_gap$$
zero_list	0x00000050	0x00000067	24	---	---	__flash_readonly_gap$$
copy_list	0x00000068	0x00000097	48	---	---	__flash_readonly_gap$$
$t.6	0x00000098	---	0	---	---	__flash_readonly_gap$$
boot_read_swap_state_by_id	0x00000099	0x000000C0	40	---	---	__flash_readonly_gap$$
$t.0	0x000000C0	---	0	---	---	__flash_readonly_gap$$
bootutil_verify_sig	0x000000C1	0x000001D4	276	---	---	__flash_readonly_gap$$
$d.1	0x000001C0	---	0	---	---	__flash_readonly_gap$$
$t.0	0x000001D4	---	0	---	---	__flash_readonly_gap$$
bootutil_tlv_iter_begin	0x000001D5	0x0000029C	200	---	---	__flash_readonly_gap$$
$d.1	0x00000298	---	0	---	---	__flash_readonly_gap$$
$t.0	0x0000029C	---	0	---	---	__flash_readonly_gap$$
Reset_Handler	0x0000029D	0x000002AA	14	---	---	__flash_readonly_gap$$
$t.0	0x000002AC	---	0	---	---	__flash_readonly_gap$$
SystemInit	0x000002AD	0x000003B4	264	---	---	__flash_readonly_gap$$
$d.1	0x00000388	---	0	---	---	__flash_readonly_gap$$
$t.3	0x000003B4	---	0	---	---	__flash_readonly_gap$$
SystemRuntimeInit	0x000003B5	0x00000400	76	---	---	__flash_readonly_gap$$

Figure 42. Memory Usage View showing functions placed in the .flash_gap section

It can be seen that **Flash Usage** decreases, while **Flash Gap Usage** increases, as shown in Figure 43.

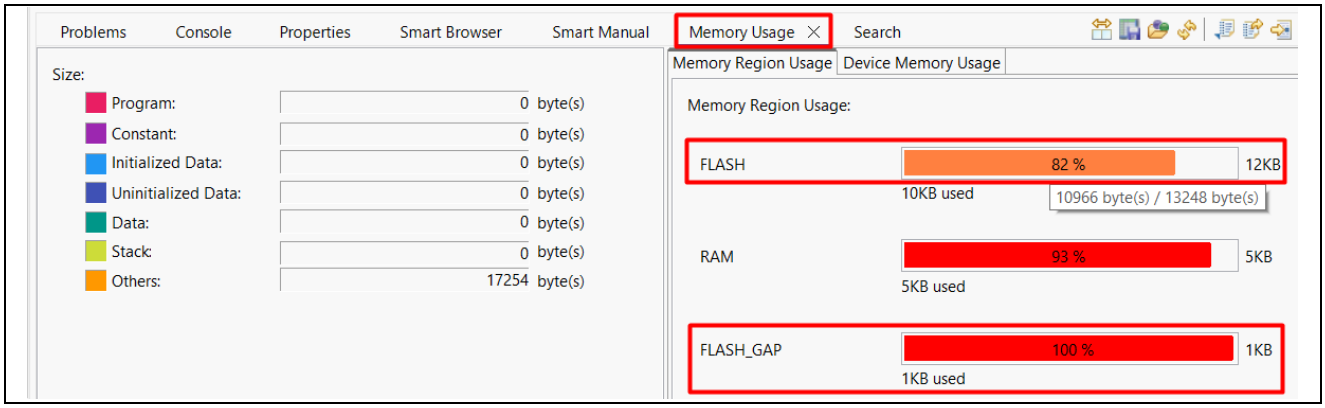


Figure 43. Effect of placing functions in the .flash_gap section on Flash and Flash Gap usage

4.3 Compiling the Bootloader Project

Depending on the upgrade mode selected, the size may differ slightly. In addition, migrating projects to a later FSP version may result in minor size variations.

```
llvm-size --format=berkeley "mcuboot_overwrite_with_signature_ek_ra2l2.elf"
   text    data     bss     dec     hex filename
  13246       0    4040   17286    4386 mcuboot_overwrite_with_signature_ek_ra2l2.elf

11:07:14 Build Finished. 0 errors, 0 warnings. (took 453ms)
```

Figure 44. Compilation Result in Overwrite Update Mode for RA2L2 with signature

```
llvm-size --format=berkeley "mcuboot_overwrite_with_signature_ek_ra2l2.elf"
   text    data     bss     dec     hex filename
   8094       0    4040   12134    2f66 mcuboot_overwrite_with_signature_ek_ra2l2.elf
llvm-objcopy "mcuboot_overwrite_with_signature_ek_ra2l2.elf" -O srec  "mcuboot_overwrite_with_signature_ek_ra2l2.srec"

11:55:13 Build Finished. 0 errors, 0 warnings. (took 5s.660ms)
```

Figure 45. Compilation Result in Overwrite Update Mode for RA2L2 without signature

```
llvm-size --format=berkeley "mcuboot_overwrite_with_signature_fpb_ra0e1.elf"
   text    data     bss     dec     hex filename
  12834       0    3768   16602    40da mcuboot_overwrite_with_signature_fpb_ra0e1.elf

10:44:40 Build Finished. 0 errors, 0 warnings. (took 2s.315ms)
```

Figure 46. Compilation Result in Overwrite Update Mode for RA0E1 with signature

```
llvm-size --format=berkeley "mcuboot_overwrite_with_signature_fpb_ra0e1.elf"
   text    data     bss     dec     hex filename
   7682       0    3768   11450    2cba mcuboot_overwrite_with_signature_fpb_ra0e1.elf
llvm-objcopy "mcuboot_overwrite_with_signature_fpb_ra0e1.elf" -O srec  "mcuboot_overwrite_with_signature_fpb_ra0e1.srec"

11:58:19 Build Finished. 0 errors, 0 warnings. (took 5s.848ms)
```

Figure 47. Compilation Result in Overwrite Update Mode for RA0E1 without signature

These outcomes are also related to the bootloader size configuration described in section 6.

4.4 Configuring 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 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 <bootloader_project>\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.

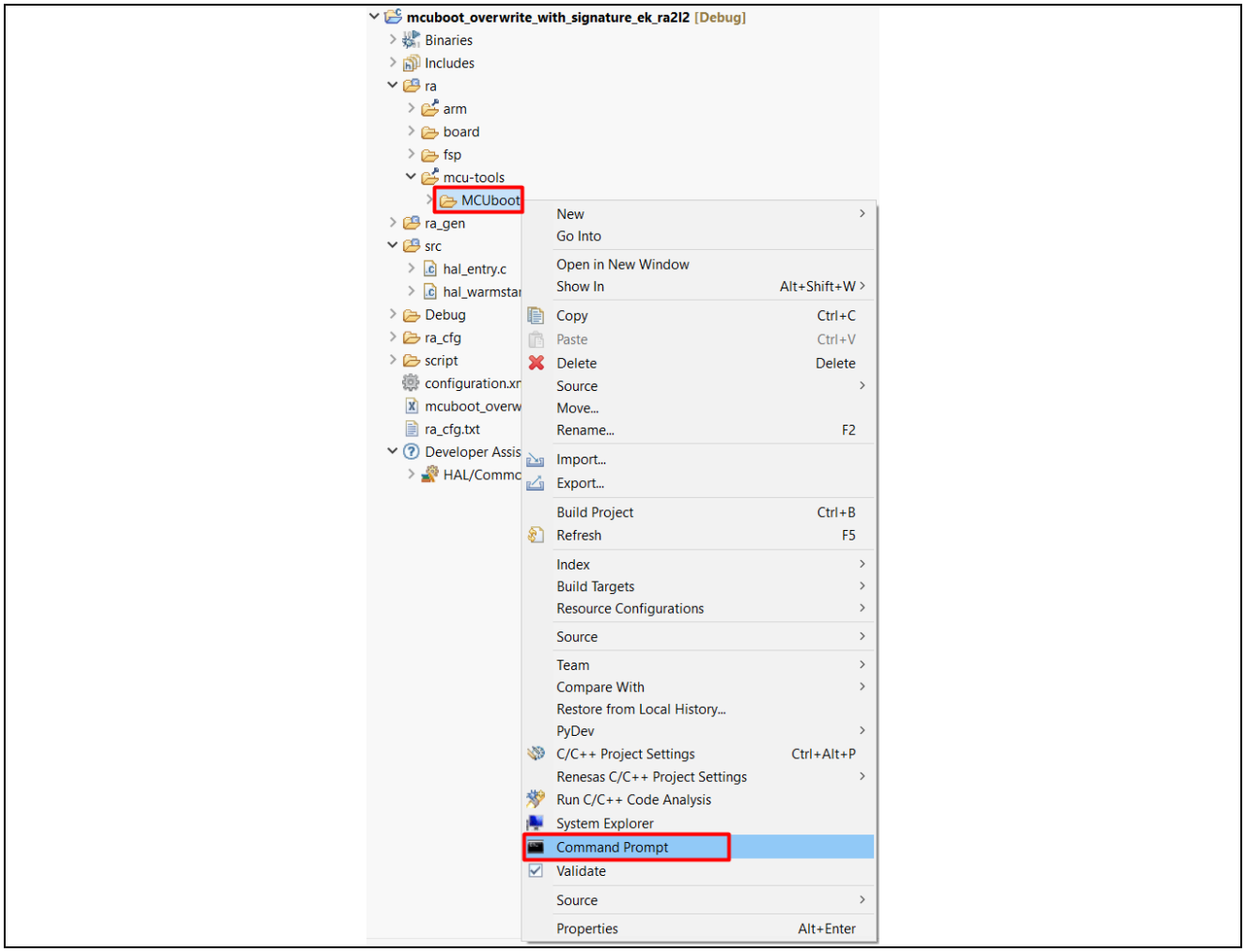


Figure 48. Open the Command Prompt

It is recommended to upgrade pip prior to installing the dependencies. Enter the following command to update pip:

python -m pip install --upgrade pip

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

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

This will verify and install any dependencies that are required.

5. Creating the User Application

Developing a user application to use MCUboot typically starts with developing and testing the application and the bootloader independently. For use with the bootloader, the user application must consider the following common steps:

- Adjust the memory map of the bootloader to allow the user application and bootloader to fit the available MCU memory area.
- Configure the application to use the bootloader.
- Signing the user application image.
- Adding the functionality to download a new application image.
  This is demonstrated in this application project, where the new application image is downloaded using XMODEM protocol via USB PCDC.

This section shows how to use the MCUboot bootloader with the LED blinky applications. After the initial blinky project is created, the project is configured to the use of the MCUboot bootloader project, generated in the previous section, and is also signed using the signing command from the bootloader project.

In addition, the Image Downloader functionality in added to the user application project, enabling the download of new application image.

Note: Users can also follow section 0 to exercise the example bootloader and application projects without going through the application creation and configuration process to use with the bootloader. This section provides references for users to understand how to customize the project for their specific application.

5.1 Generate the Primary User Application Project

Follow the steps below to create a blinky application project as the Primary User Application Project. The steps in section 5.1 are identical when generating a blinky project, whether the application uses a bootloader or not. Launch e^2 studio and open a Workspace, click **File > New > Renesas C/C++ Project > Renesas RA**, and choose **Renesas RA C/C++ Project**. Click **Next**.

1. Assign the project name based on Table 1.

Table 1. Name the Initial Application Project

Bootloader project name	User application project name
mcuboot_overwrite_with_signature_ek_ra2l2	primary_app_usb_ek_ra2l2
mcuboot_overwrite_with_signature_fpb_ra0e1	primary_app_fpb_ra0e1

2. Click **Next**, choose **EK-RA2L2** as the **Board** from the drop-down menu, and choose the **LLVM Embedded Toolchain**. Click **Next**.
3. Linking the Application Project to Bootloader Project.

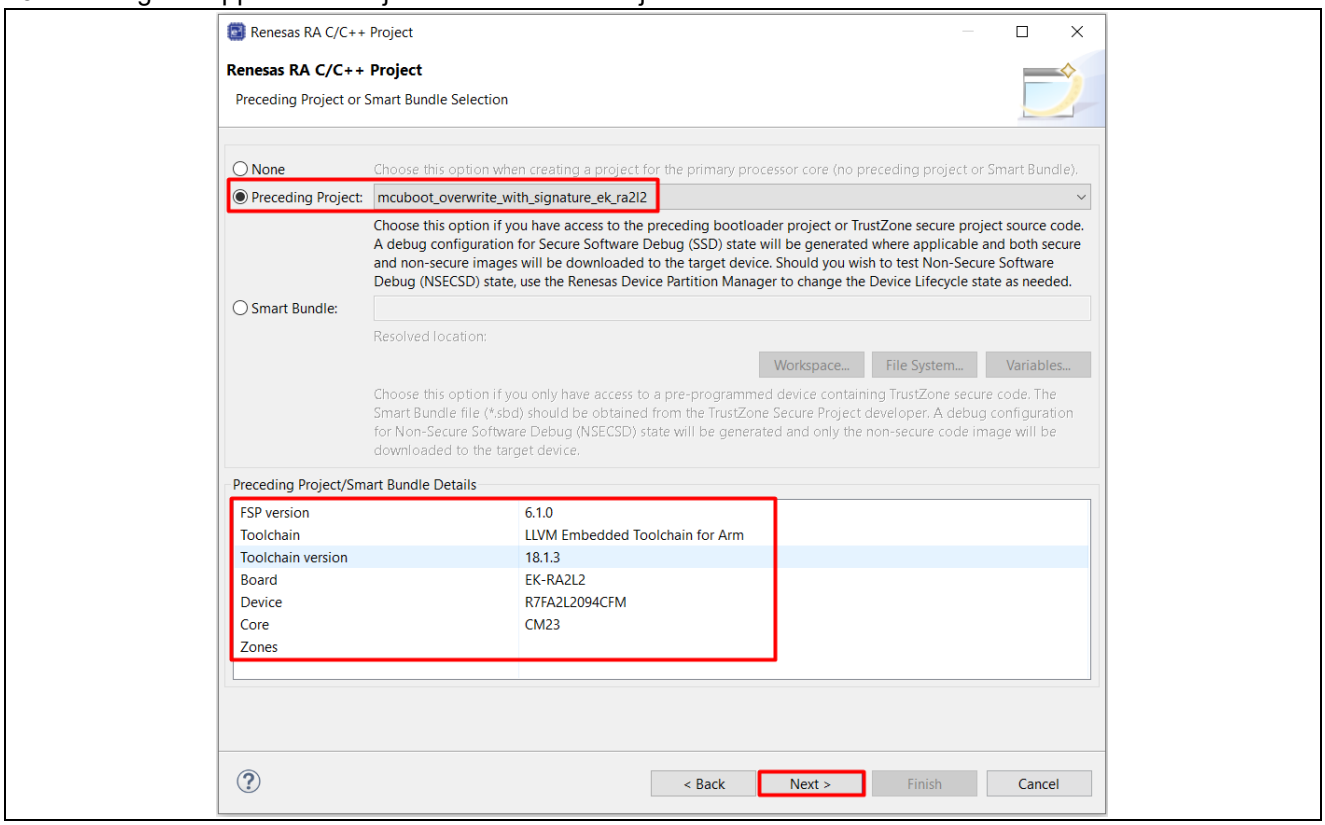


Figure 49. Selecting the Preceding Project

4. In the next screen, select **Executable** as the **Build Artifact** and **No RTOS** for the **RTOS Selection**. Click **Next**.

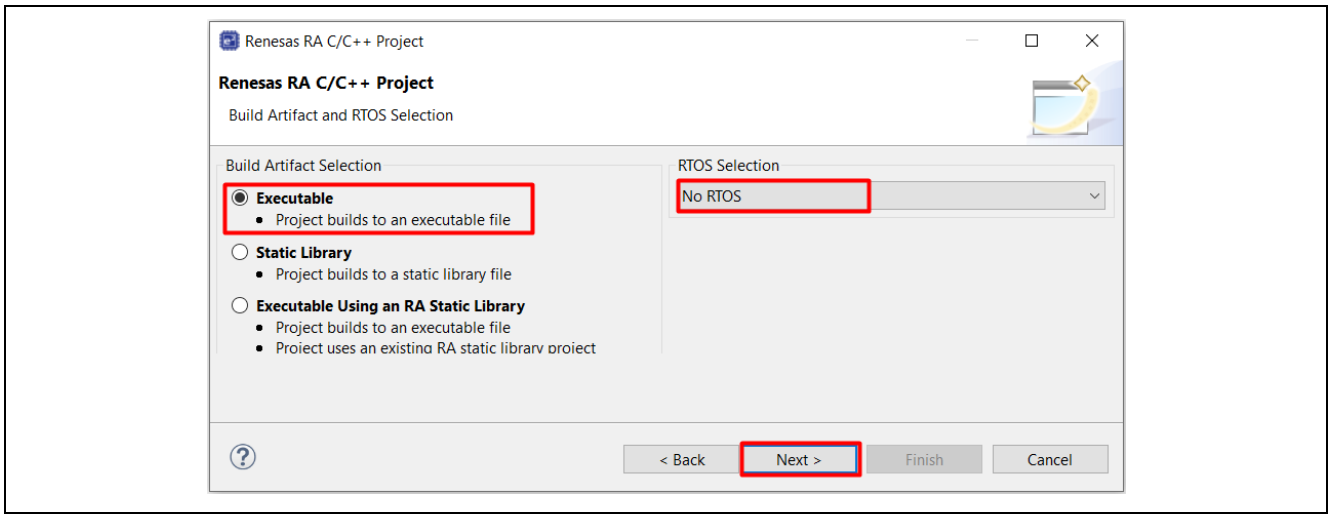


Figure 50. Choose to Build Executable with No RTOS

5. Select the **Bare Metal - Blinky** as the **Project Template** for the board and click **Finish**. The initial application project is now created.

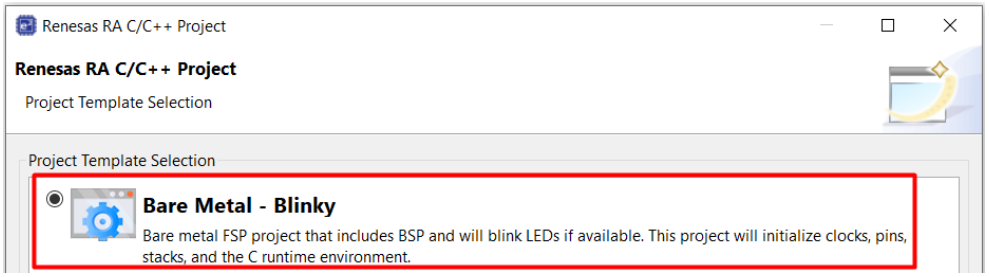


Figure 51. Choose Bare Metal – Blinky as Project Template

Note that all the steps described in this section are the same when developing for the FPB-RA0E1 board.

5.2 Implementing the Image Downloader sample for EK-RA2L2 Board

This section details how to add functionality for downloading a new image, using XMODEM protocol via USB PCDC to the application.

1. Open the configuration.xml file in the primary_app_usb_ek_ra2l2 project.
   In the **Stacks Configuration** window, select **New Stack > Connectivity > USB PCDC Communication Device (rm_comms_usb_pcdc)** stack.

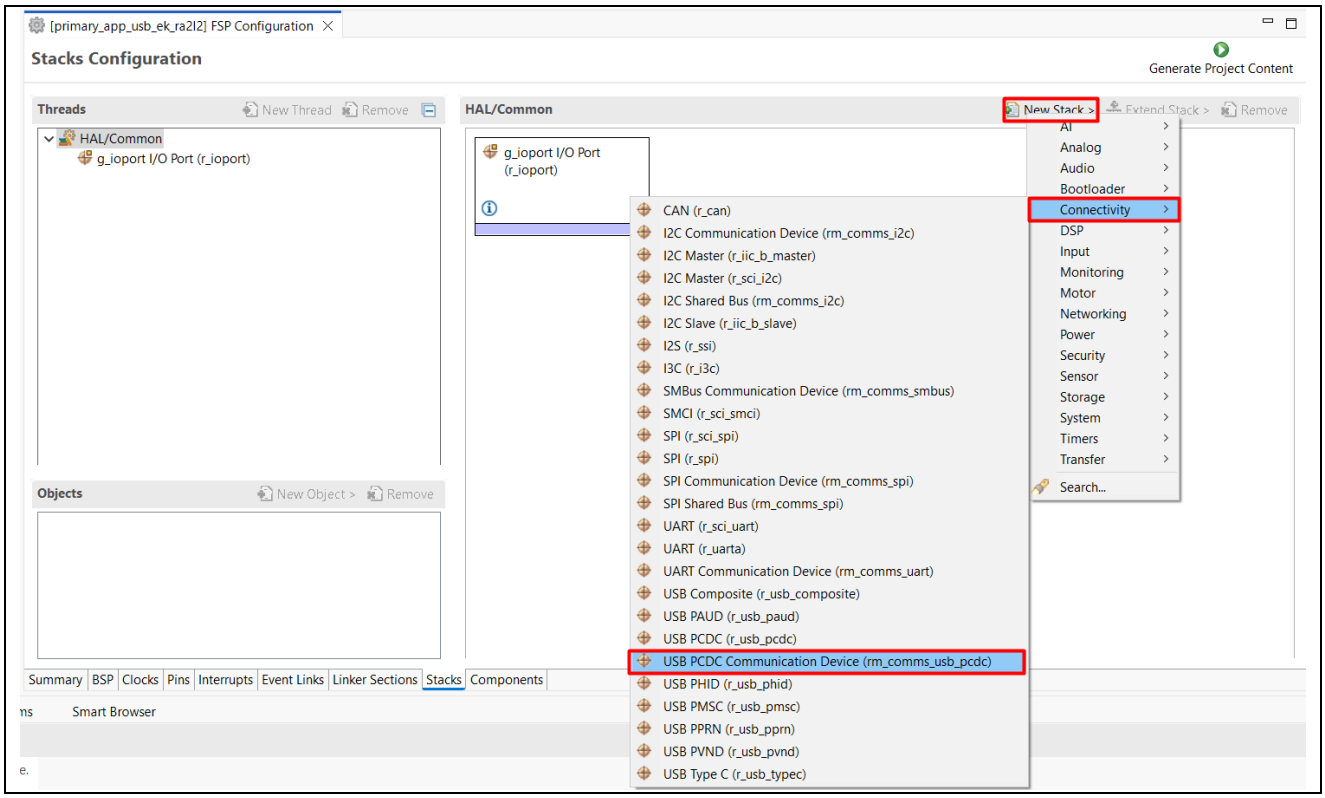


Figure 52. Choose USB PCDC Communication Device (rm_comms_usb_pcdc) stack

2. Configure the **r_gpt** under **USB PCDC Communication Device (rm_comms_usb_pcdc).**

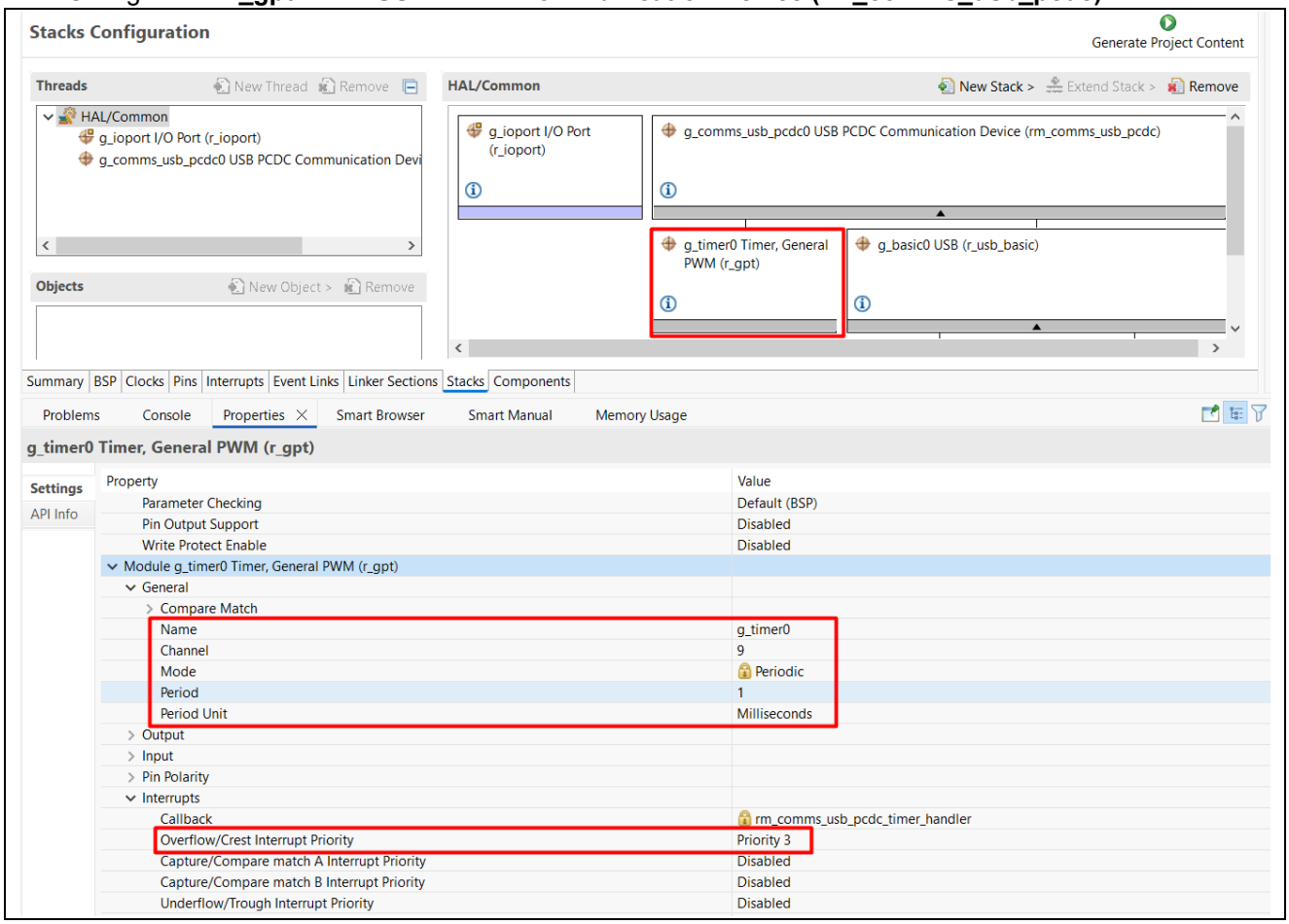


Figure 53. Configure the r_gpt

3. In the **Pins** tab, assign **P407** as **USB_VBUS** in **Connectivity: USB FS**, as shown in Figure 54.

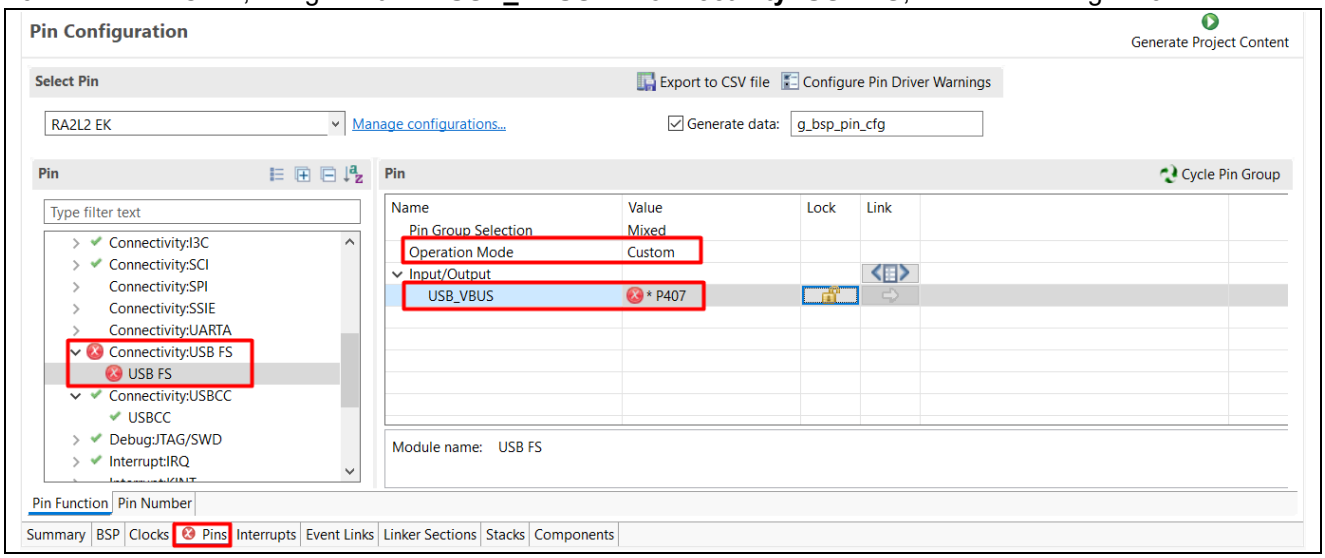


Figure 54. Enable the USB_VBUS pin in Connectivity: USB FS

4. To resolve the issue shown in Figure 54, disable **USB_VBUS** in **Connectivity: USBCC**

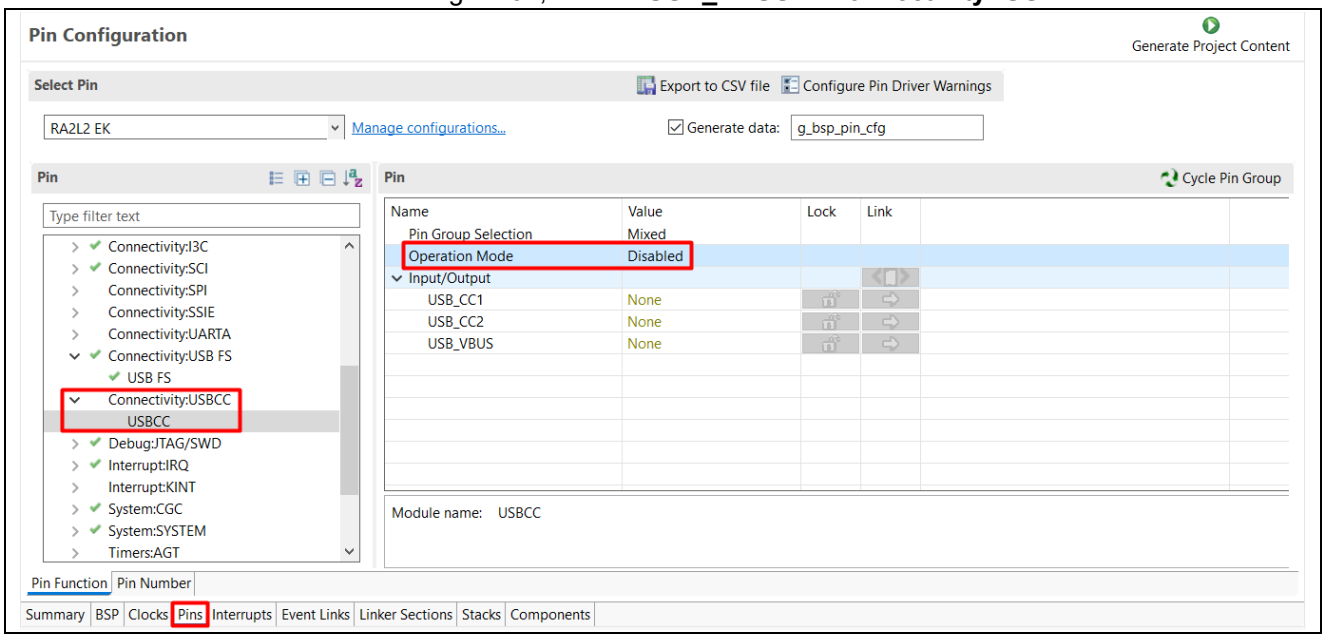


Figure 55. Disable the USB_VBUS pin in Connectivity: USBCC

After steps 3 and 4, the **USB_VBUS** pin is enabled in the **r_usb_basic** module, as shown in Figure 56.

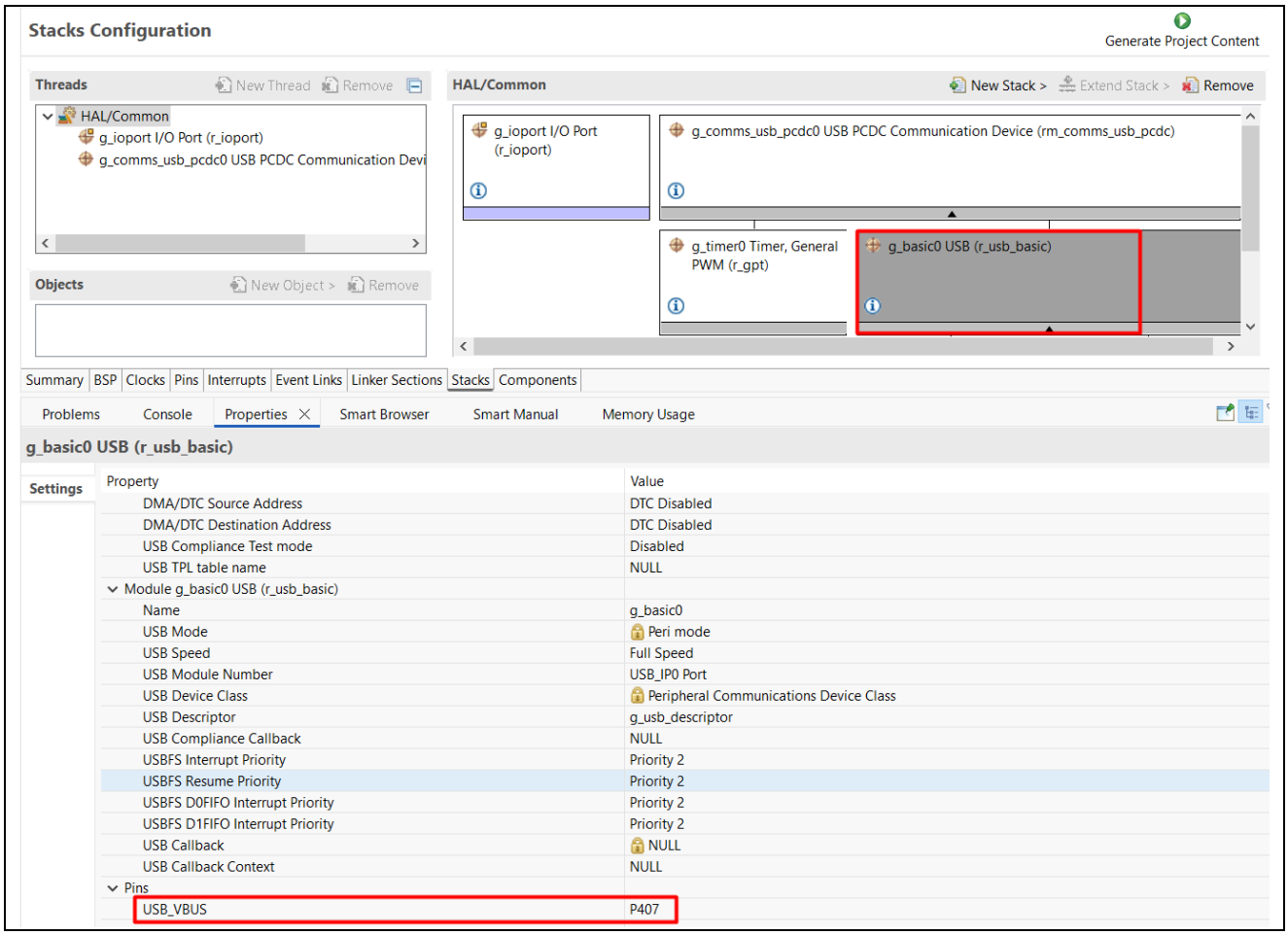


Figure 56. USB_VBUS enabled in the r_usb_basic module

5. Add the **Flash (r_flash_lp)** and set the **Code Flash Programming** to **Enabled**.

   Set **Data Flash Programming**, **Data Flash Background Operation Support**, and **Data Flash Background Operation** to **Disabled**, as shown in Figure 57.

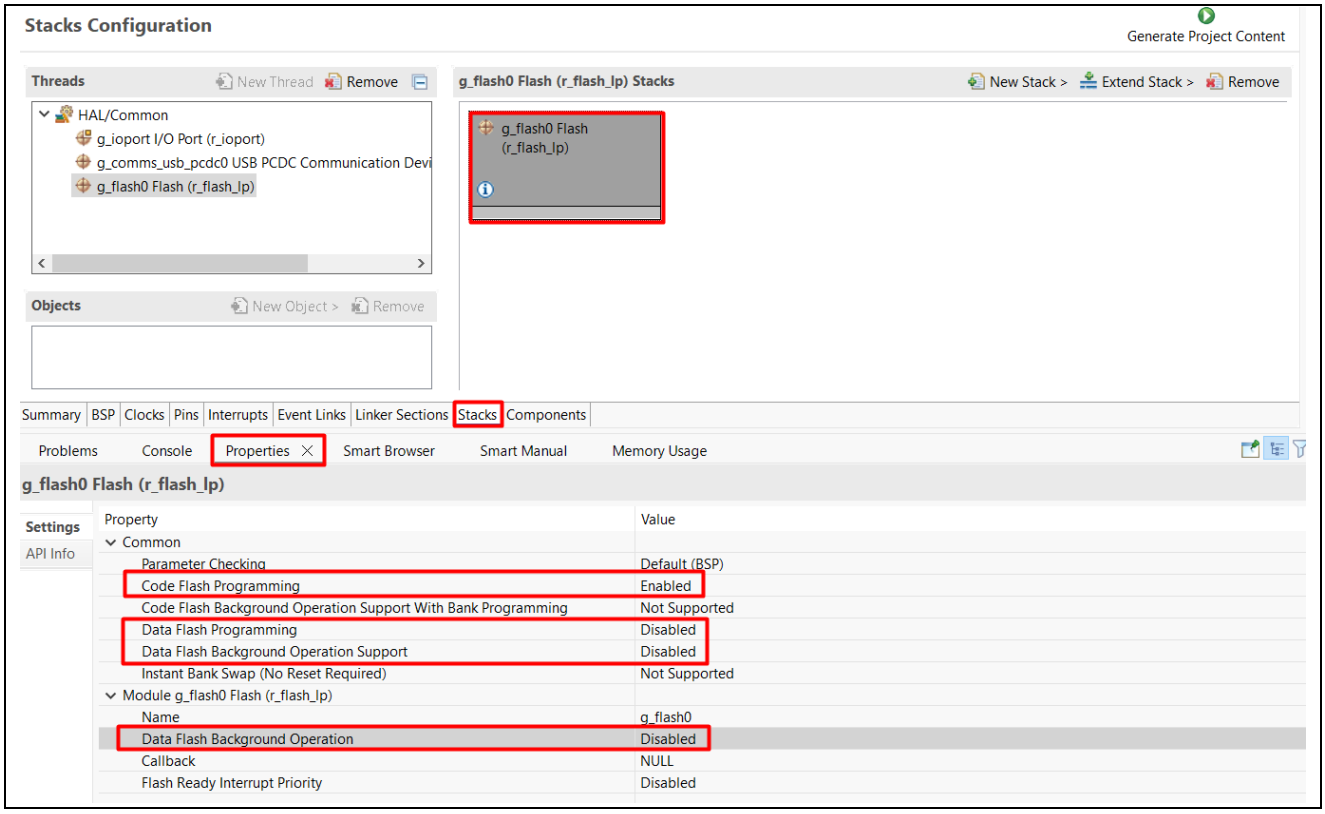


Figure 57. Add the Flash (r_flash_lp)

6. Click **Generate Project Content** to apply the configuration settings.
7. Unzip the file **ek_ra2l2_secure_bootloader_via_usb_pcdc.zip**.

   Open the folder **ek_ra2l2_secure_bootloader_via_usb_pcdc\primary_app_usb_ek_ra2l2\src** and copy all files under **\src** to the **\src** folder for the newly created project.

Since this application project was created as a **Bare Metal** environment (no RTOS is used), it is important to handle that the application can prevent blocking. To do this the **System Tick Timer** will be used to handle timing, rather than using software delay functions (e.g., **R_BSP_SoftwareDelay**) or using while() loops, waiting for an event too occur, which may cause the system to hang. The **System Tick Timer** will be used for the following purposes:

- Creating a timeout for USB read operations.
- Toggling the onboard LEDs to indicate that the application project is running.

5.3 Signing the User Application Image

Note: If the bootloader project is re-built after changing any of the signing and signature **Properties** of the MCUboot module, the developer must repeat **Generate Project Content** to update the .sbd file, because the application project is linked to the bootloader project through this file.

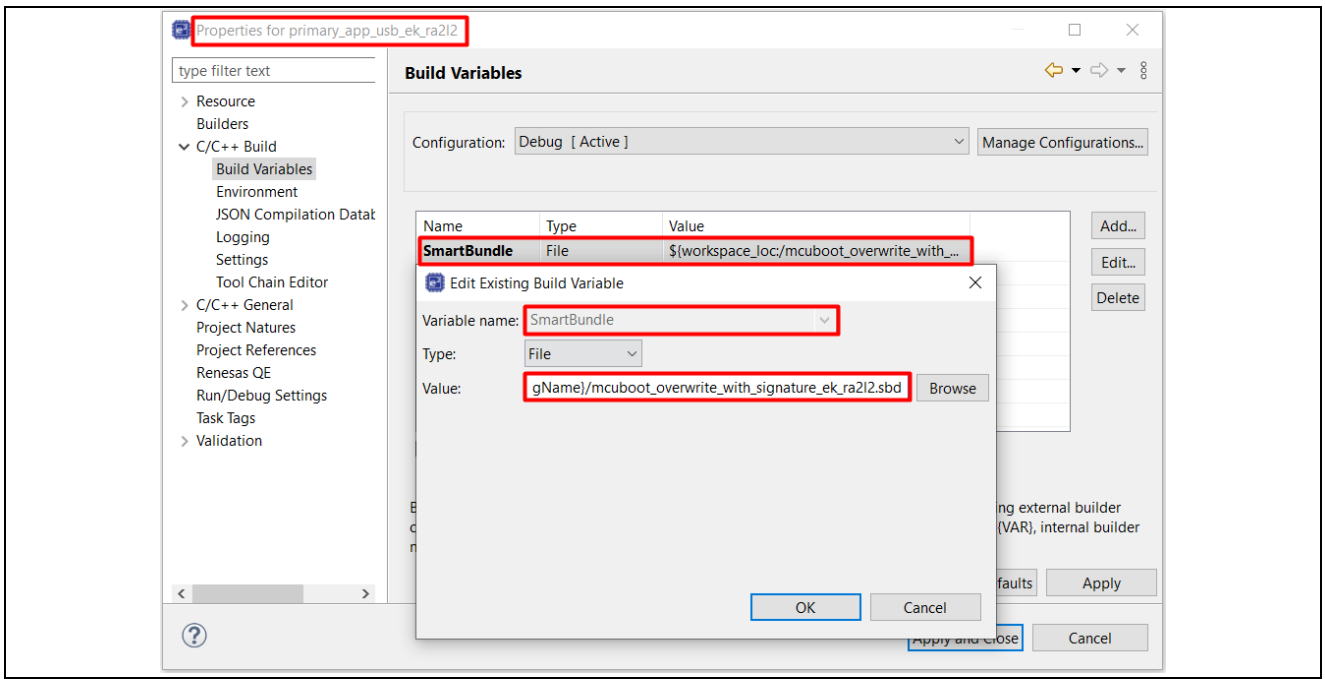


Figure 58. Access the .sbd file in the Application Project

Each application can have a defined version number. This version number can be used in the overwrite upgrade mode when **Downgrade Prevention** is **Enabled**. This is achieved by defining an Environment Variable: **MCUBOOT_IMAGE_VERSION**.

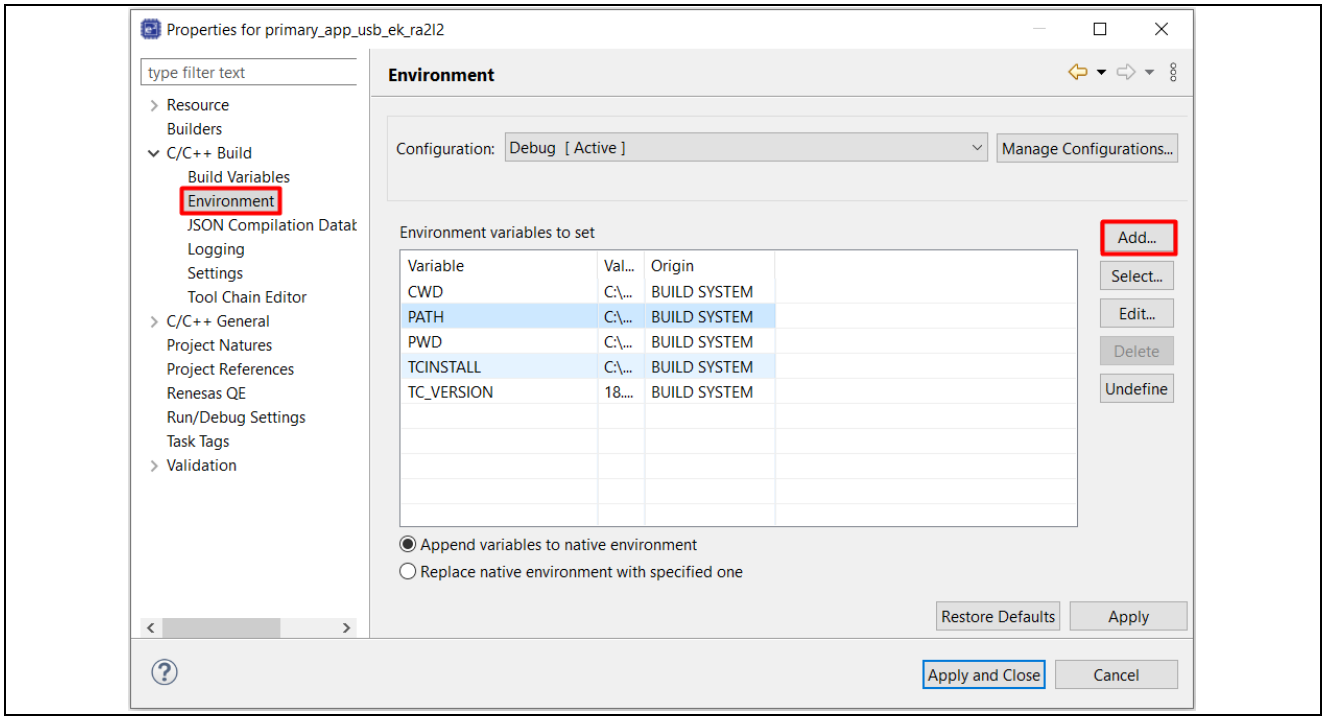


Figure 59. Add New Environment Variable

Add the Environment Variable: **MCUBOOT_IMAGE_VERSION** for the application image version.

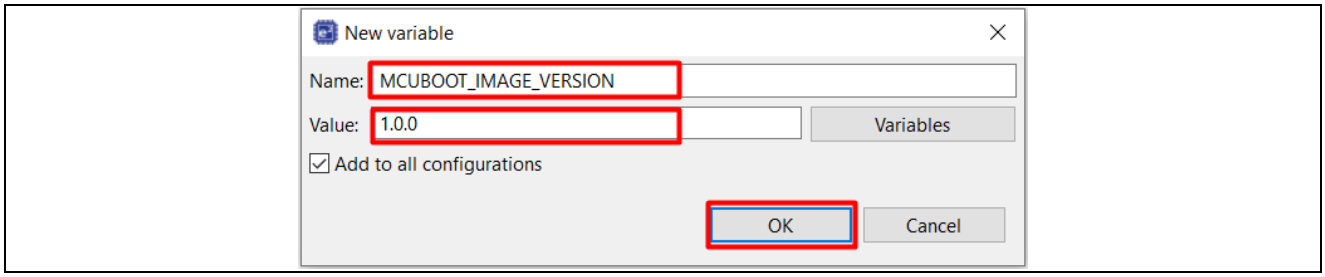


Figure 60. Add MCUBOOT_IMAGE_VERSION Variable

If there is signature verification, then it is necessary to set the Environment Variable: **MCUBOOT_IMAGE_SIGNING_KEY**

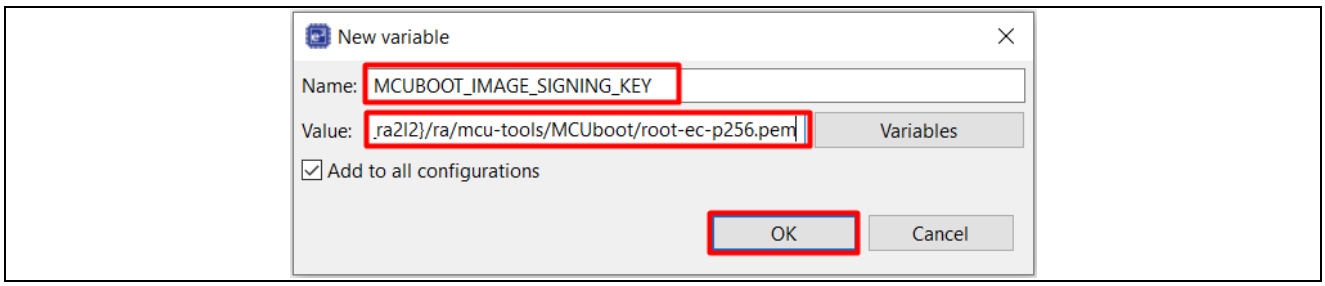


Figure 61. Add MCUBOOT_IMAGE_SIGNING_KEY Variable

When using the **LLVM Embedded Toolchain**, add the Environment Variable: **MCUBOOT_APP_BIN_CONVERTER** in order to sign the application image.

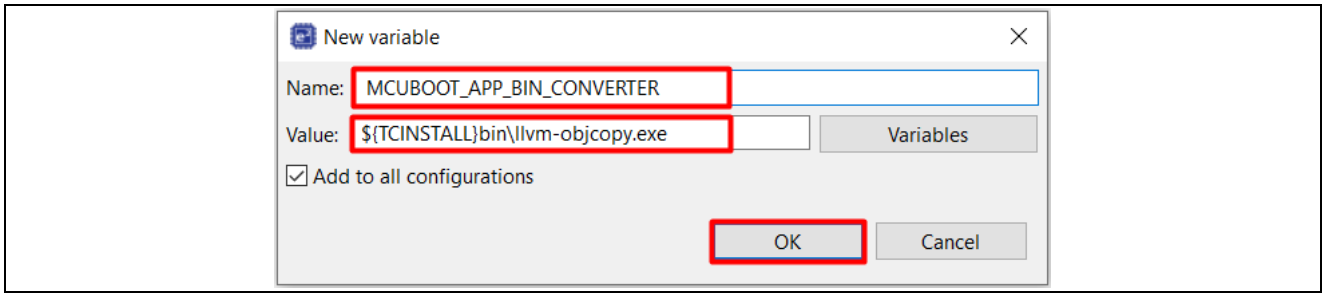


Figure 62. Add MCUBOOT_APP_BIN_CONVERTER Variable

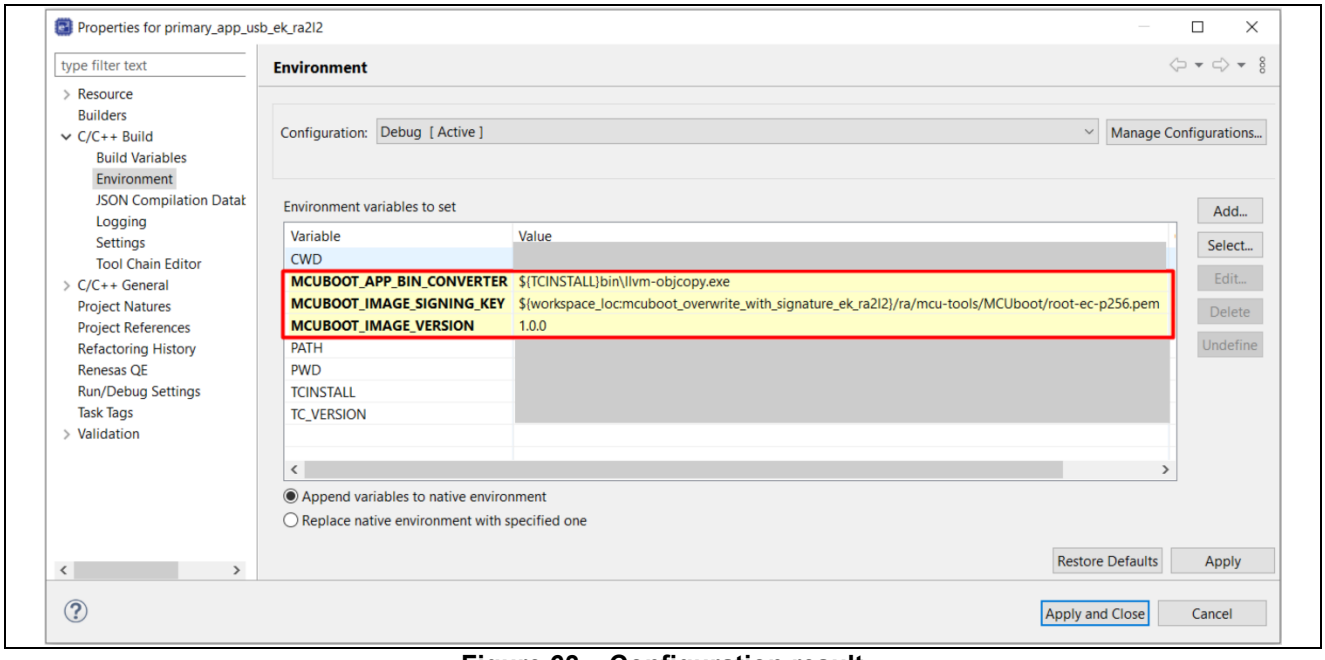


Figure 63. Configuration result

Note:

- The private key used for signing the application image is specified in the signing command. **/ra/mcu-tools/MCUboot/root-ec-p256.pem** is used in this application example. This key is used for testing purposes only. For real-world use cases and production support, the developer **MUST** change this to the private key of their choice.

- The value of the **MCUBOOT_APP_BIN_CONVERTER** variable corresponds to the installation path of the LLVM toolchain - specifically, ${TCINSTALL}bin\llvm-objcopy.exe.

To ensure that the project is re-built when the Environment Variables or the linker script are modified, it is recommended to add a **Pre-build** step that will delete the .elf file, as shown in Figure 64.

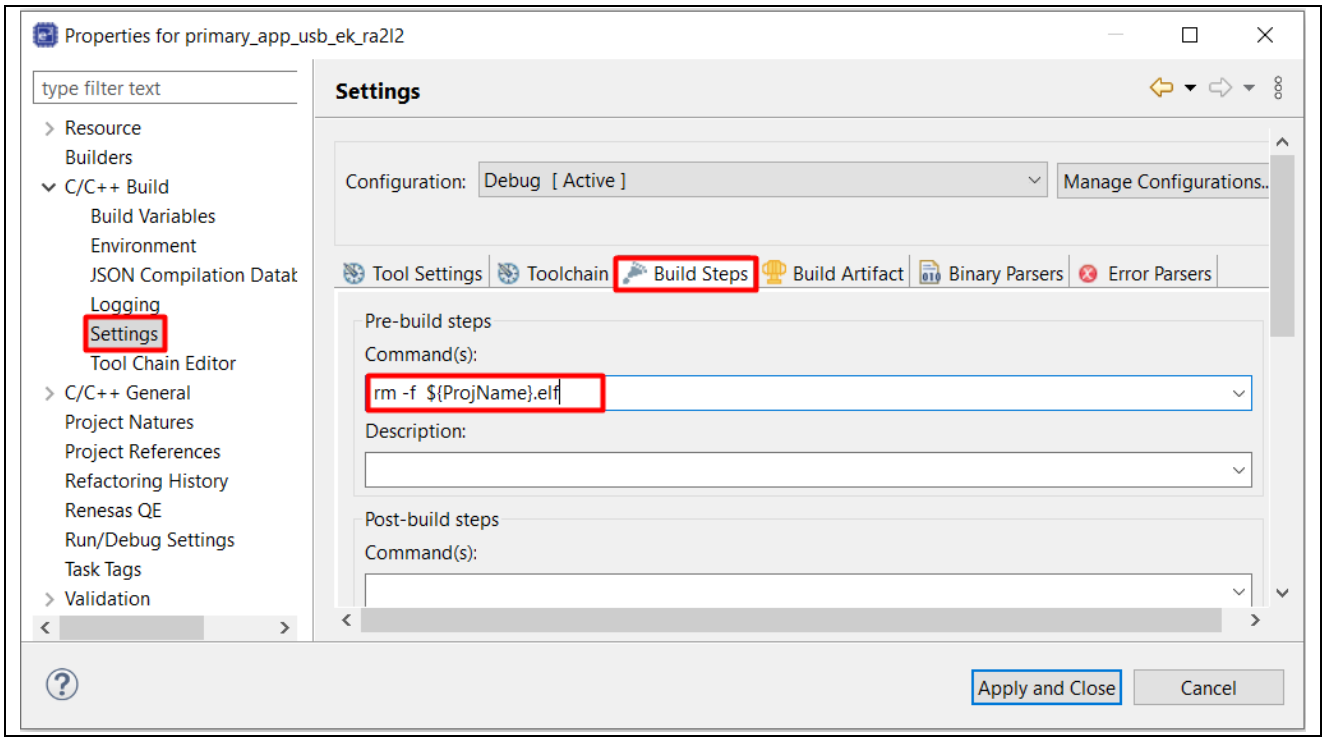


Figure 64. Configuration the Pre-build Command

When building the application project, an error will appear in the **Console** tab, as shown in Figure 65.

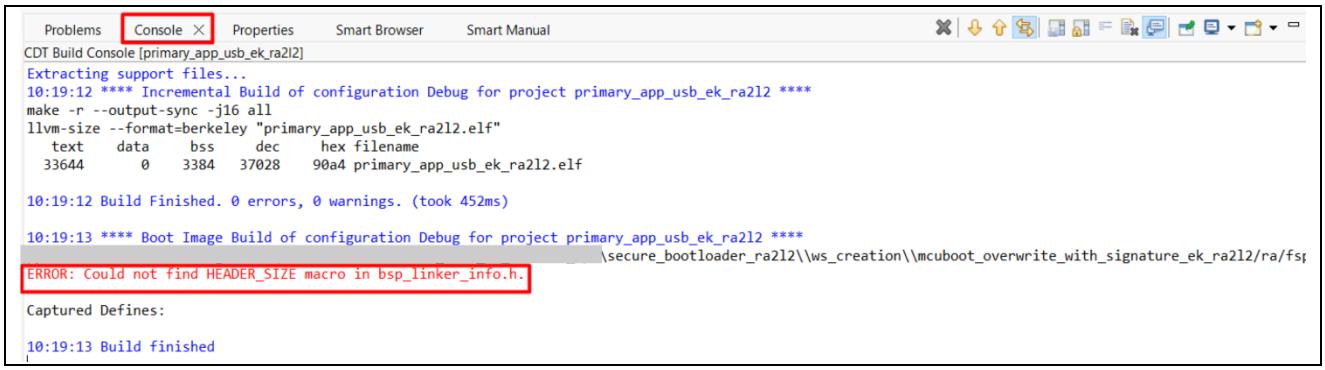


Figure 65. Error message when building the application project

To resolve this issue, users should refer to section 6 for the detailed procedure.

6. MCUboot Memory Configuration with Renesas FSP Solution Project

Starting from FSP v6.0.0 with e²studio v2025-04.1, when creating a Bootloader Project, developers must use the **Renesas FSP Solution Project (Advanced)** project template to merge the MCUboot and Application projects together. This provides a more effective and intuitive bootloader memory management process by allowing users to edit the “solution.xml” file within the **Solution Project**.

The **FSP Solution Project (Advanced)** is a chain of the projects, specifically consisting of the MCUboot bootloader and the application project that have are created in the previous steps.

6.1 How to Set Up the Renesas FSP Solution Project

Click **File** > **New** > **C/C++ Project** and select **Renesas FSP Solution Project (Advanced)**, as shown in Figure 66.

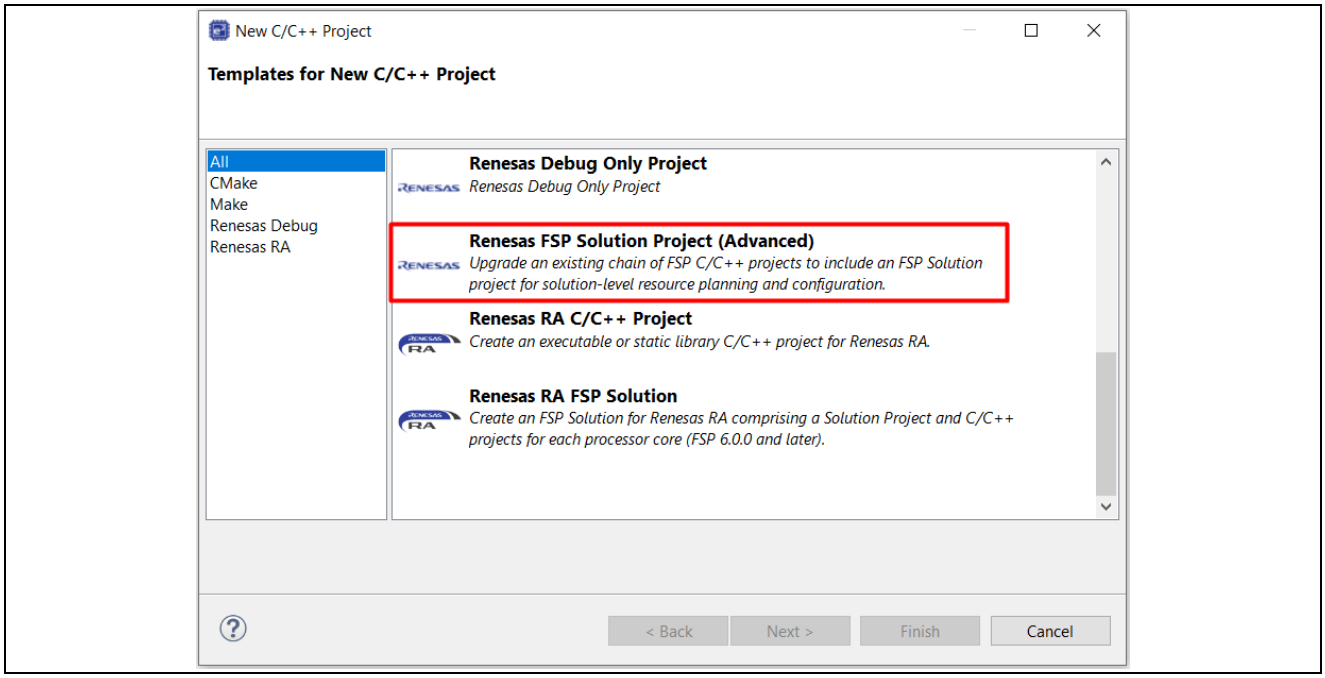


Figure 66. Renesas FSP Solution Project (Advanced)

1. Assign the project name based on Table 2.

Table 2. Renesas Solution Project names

Bootloader project name	Primary application project name	Renesas Solution Project name
mcuboot_overwrite_with_signature_ek_ra2l2	primary_app_usb_ek_ra2l2	mcuboot_overwrite_solution_ek_ra2l2
mcuboot_overwrite_with_signature_fpb_ra0e1	primary_app_fpb_ra0e1	mcuboot_overwrite_solution_fpb_ra0e1

2. Select the final project in a chain of projects.
In the **Project** option, select the primary_app_usb_ek_ra2l2, which is created in section. Click **Finish**.

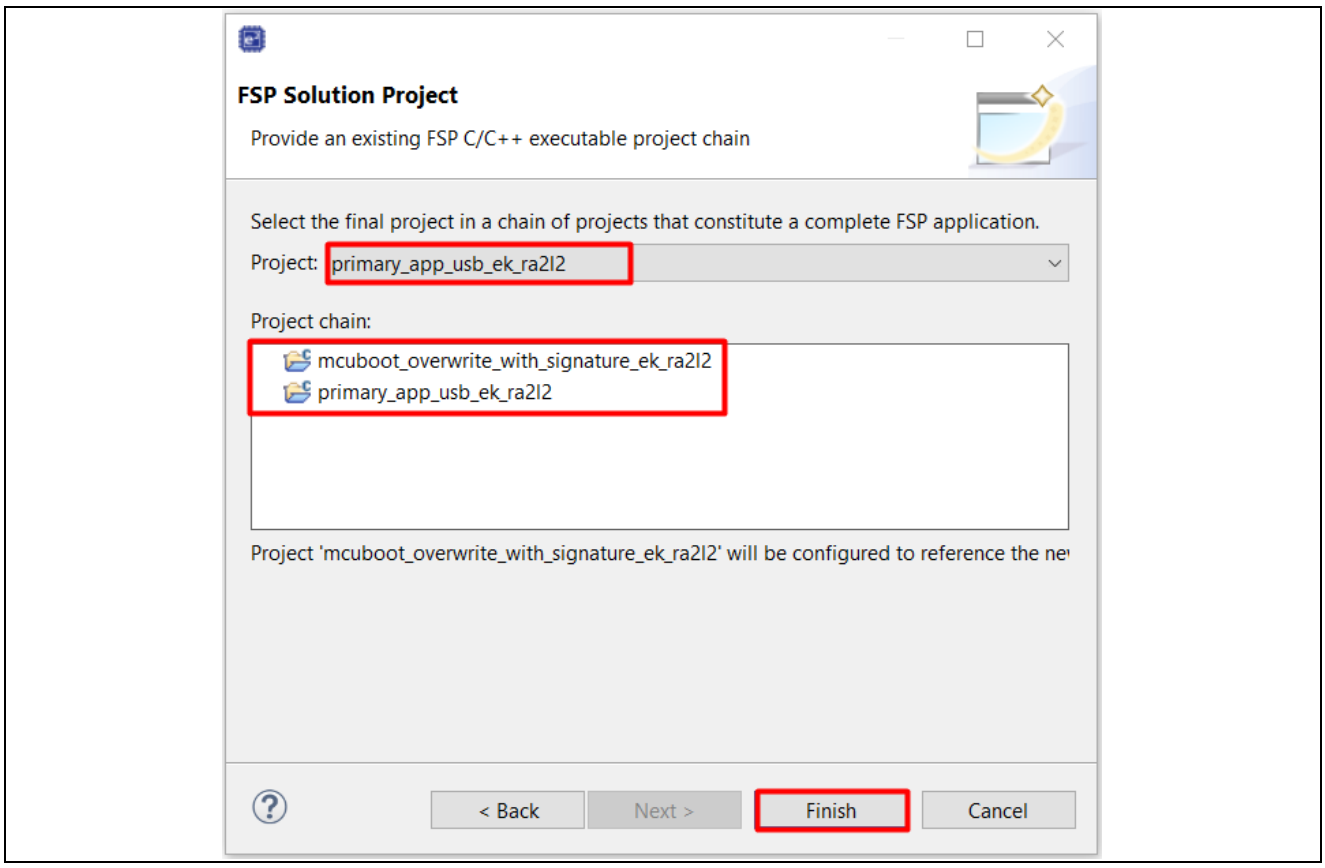


Figure 67. Select the application project in chain of projects

6.2 Managing the MCUboot Memory Configuration

Figure 68 and Figure 69 show the Flash memory map of the two applications. These can be referenced to when configuring the MCUboot memory settings within the **Solution project**.

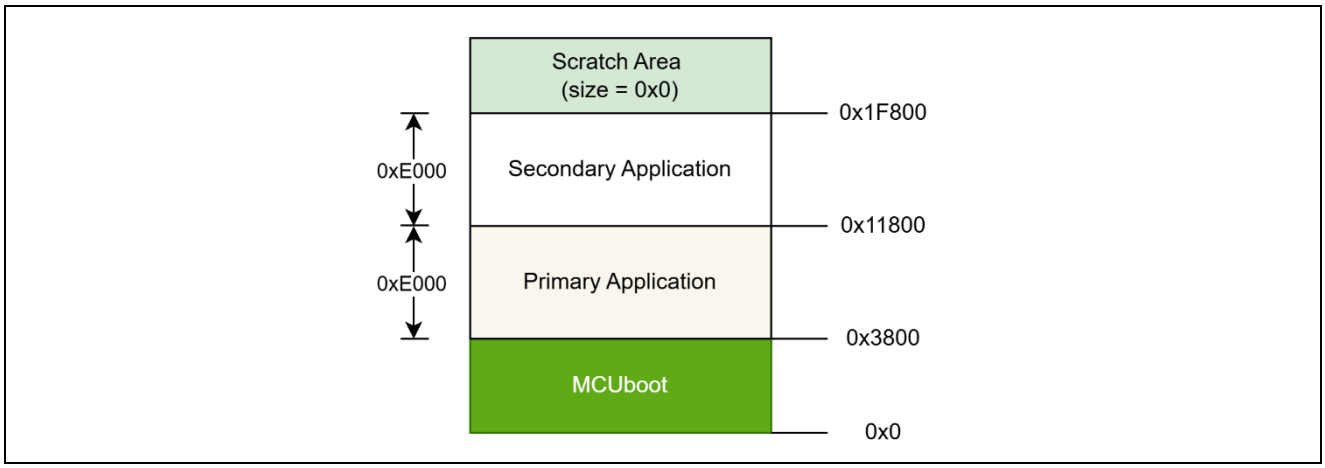


Figure 68. MCUboot Flash Map in Overwrite Update Mode for RA2L2 sample project

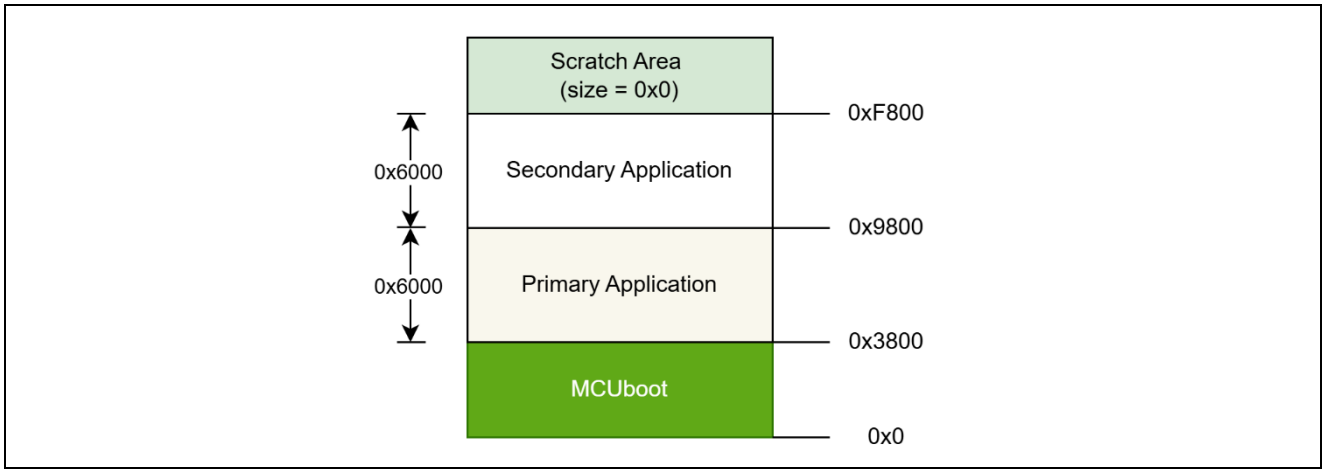


Figure 69. MCUboot Flash Map in Overwrite Update Mode for RA0E1 sample project

1. Memory Partition in Solution Project

Double-click solution.xml in the **overwrite_mode_solution** project.

Navigate to the **Memories** tab to configure the MCUboot flash map.

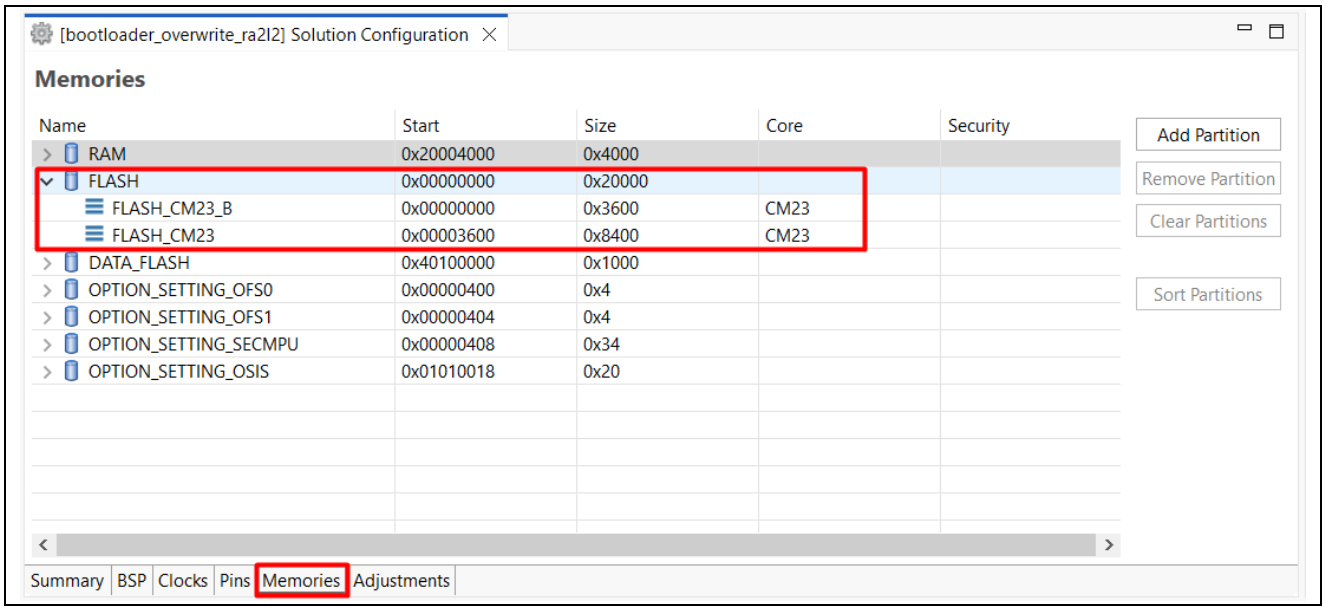


Figure 70. Memory Partition in Solution Project

Table 3 specified the labels that must be used when creating the memory partitions of the **Solution Project.** For more details about the MCUboot Memory Partition Labels, users can refer to the [renesas.github.io: MCUboot Port]

Table 3. Memory Partition Labels for MCUboot Flash Map

Memory Partition Labels	Flash Map	Definition of Memory Partition Labels
__BL_S	Scratch Area	Scratch area
__BL_0_S_I	Trailer	Image 0 Secondary Image
	TLV	
	app_secondary.bin	
__BL_0_S_H	Header	Image 0 Secondary Header
FLASH_CM23	Trailer	Image 0 Primary Image
	TLV	
	app_primary.bin	
__BL_0_P_H	Header	Image 0 Primary Header
FLASH_CM23_B	MCUboot	Bootloader area

Using the information from Figure 68 and Figure 71, configure the MCUboot Flash Map in the memory partition of the **Solution Project** as shown in figures 73, 74 & 75.

Add the MCUboot Memory Partition Labels

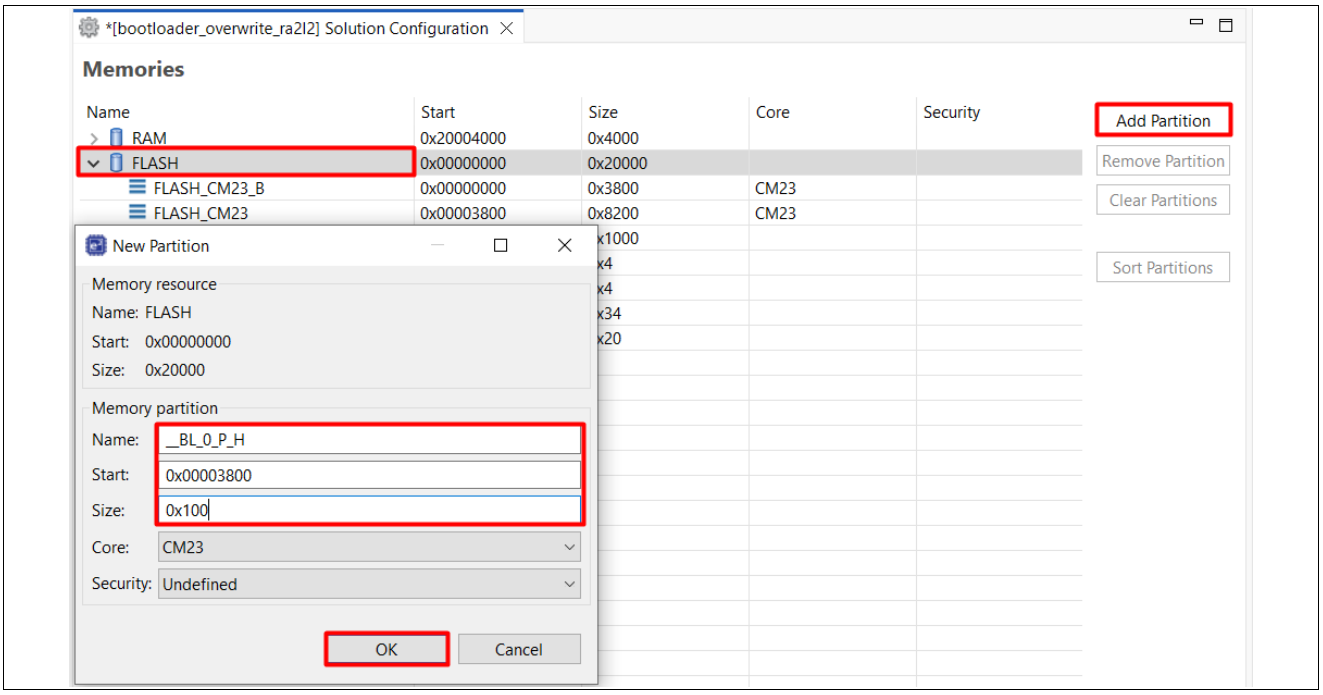


Figure 71. Add the MCUboot Memory Partition Labels

Sequentially copy the labels from Table 3 into the Memory Partition of the **Solution Project**, and then use **Ctrl + S** to save the settings.

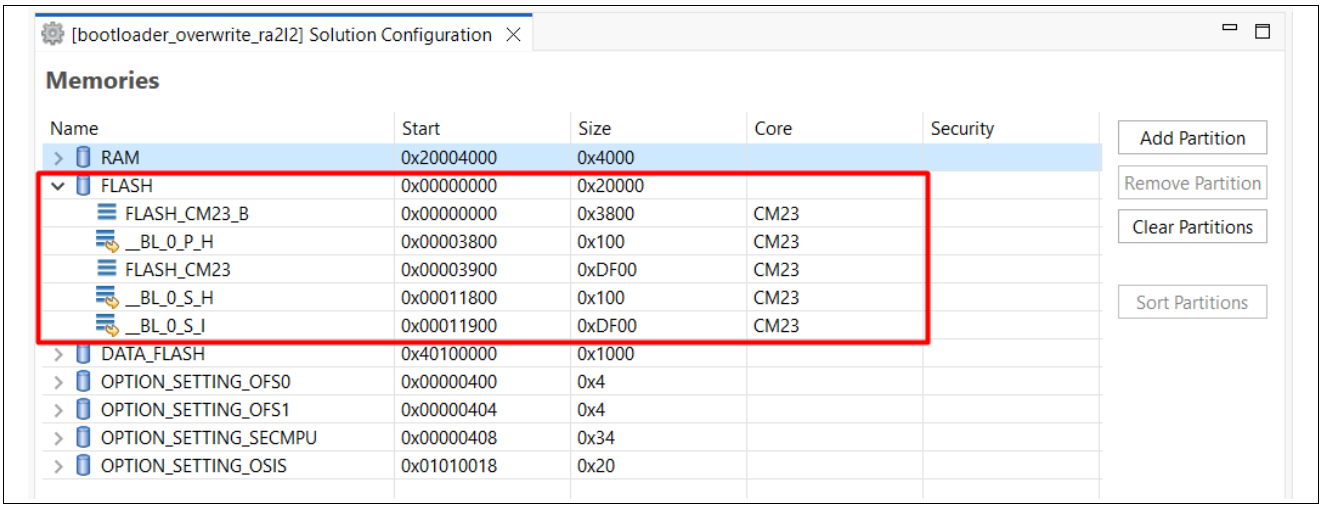


Figure 72. The MCUboot Memory Partition in Overwrite Update Mode for the RA2L2 project

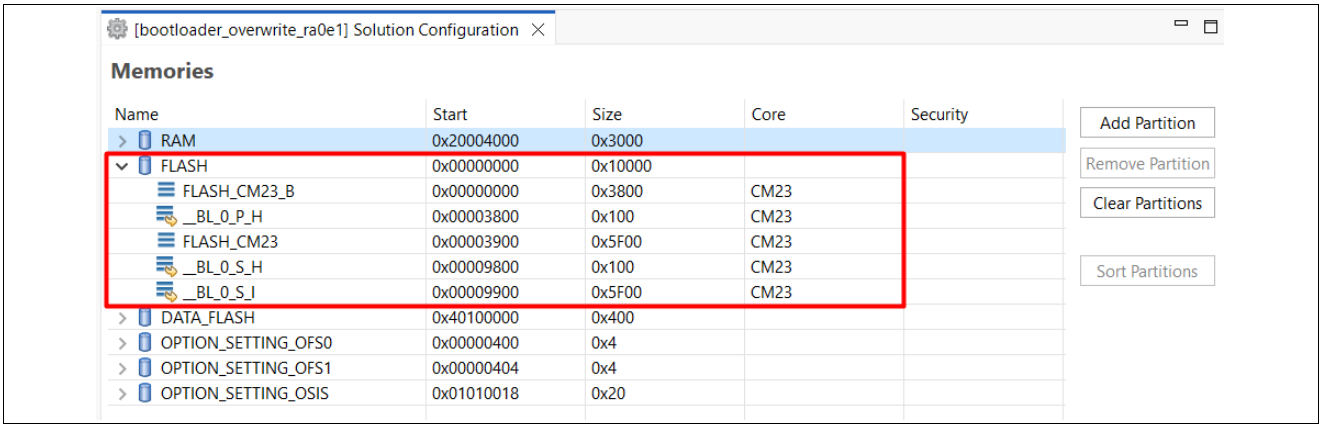


Figure 73. The MCUboot Memory Partition in Overwrite Update Mode for the RA0E1 project

2. Build the **Solution Project**

Right-click on the **Solution Project** and select **Build Project**.

This command will build all projects within the **Solution Project**.

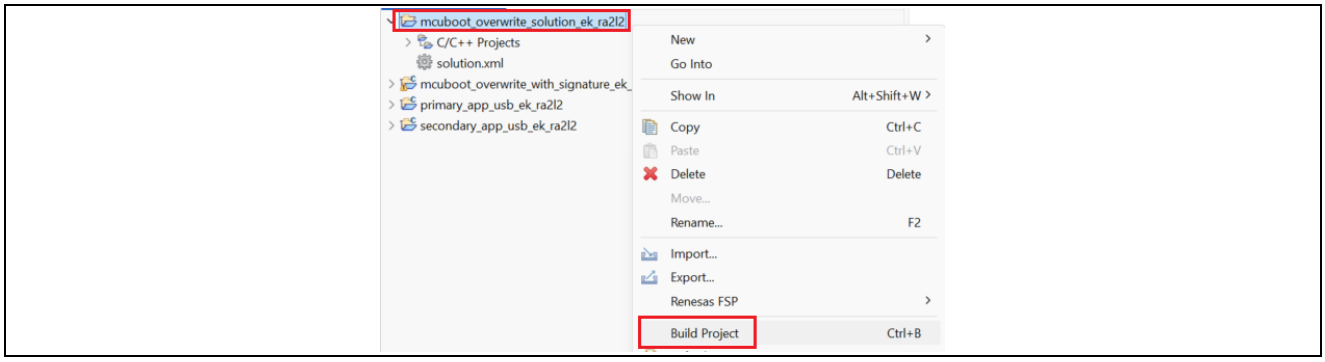


Figure 74. Build all projects within the Solution Project

This will generate the files **\debug\primary_app_usb_ek_ra2l2.bin.signed,**

or **\debug\ primary_app_fpb_ra0e1.bin.signed**

Note: If an error related to RAM region overflow appears in the **Console** tab, the RAM region in the **Memory Partition** settings of the **Solution Project** will have to be adjusted.

- For the Bootloader Project, adjust the **RAM_CM23_B** size.
- For the Application Project, adjust the **RAM_CM23** size.

6.3 Optimizing SRAM Allocation

The **Renesas FSP Solution Project (Advanced)** project template is also used when developing applications for Renesas Dual Core MCU. These applications have a unified SRAM that is used by both cores, but these are typically separate. As such, any RAM areas are defined as separate areas. In a MCUboot application, the RAM areas can overlap as the bootloader and application are never running at the same time and both applications can access the full RAM area.

Maximize the MCU SRAM by the following steps:

1. Navigate to the **Solution Project**: mcuboot_overwrite_solution
2. In the **RAM Memory Partition**, configure the **Start address** and **maximum RAM size** for the application project:

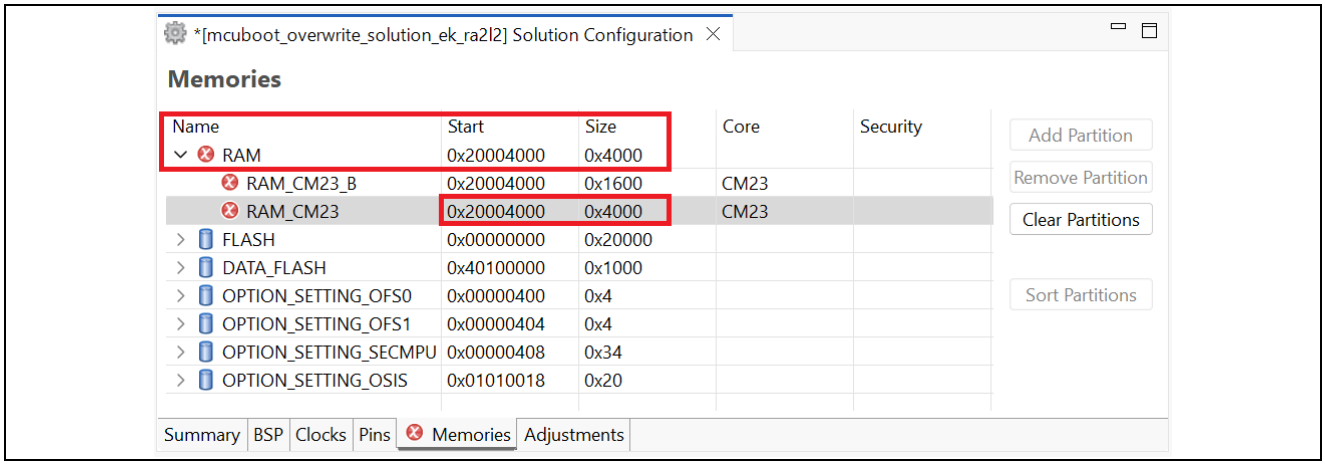


Figure 75. Adjust the maximum SRAM for the application in EK_RA2L2

The error that appears is a **Partition Overlapping** error, but this can be ignored.

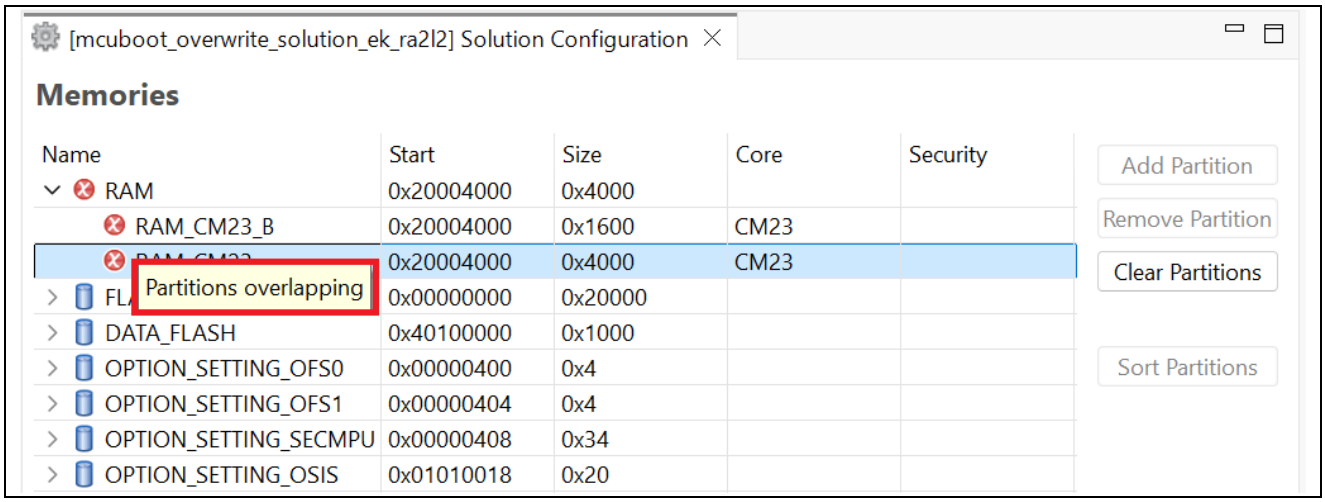


Figure 76. Partition Overlapping Error

3. Press Ctrl + S to save the configuration

7. Booting the Initial Application Project

7.1 Erase the MCU

To create a clean environment for starting the initial application project it is recommended that MCU should be completely erased using the Renesas Flash Programmer (RFP) or third-party tools like J-Flash Lite.

7.1.1 Use the Renesas Flash Programmer

Connect the EK-RA2L2 to the PC through J10 USB Debug. Launch **RFP** and create a new RFP project. Click **File** > **New Project**.

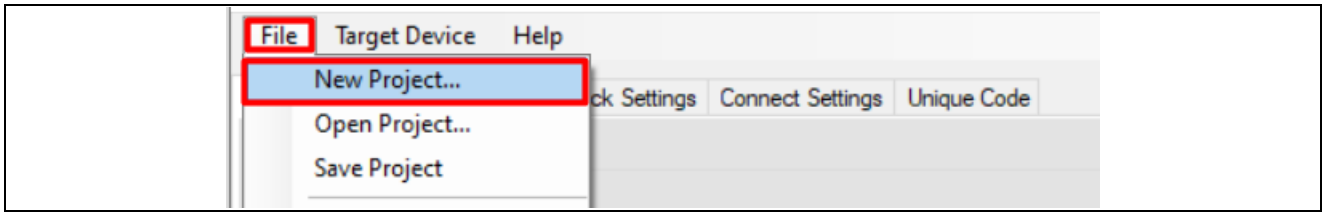


Figure 77. Create a New RFP Project

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

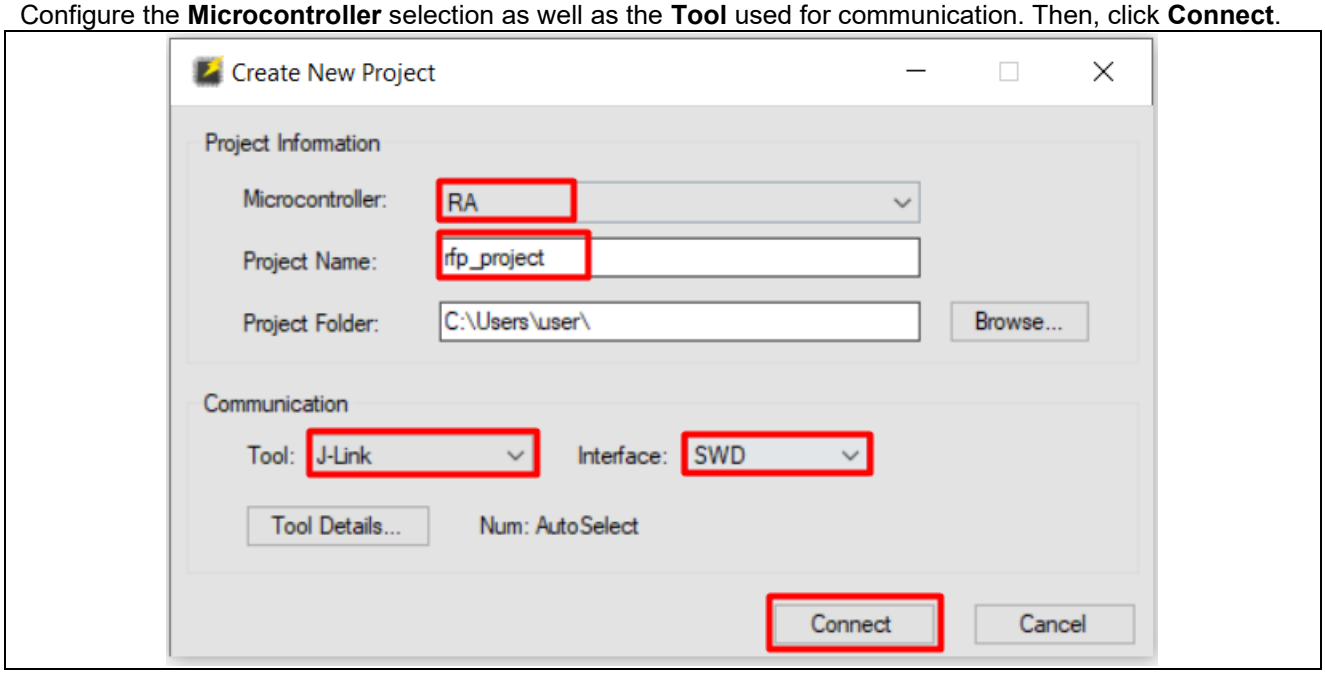


Figure 78. Configure the New Project

On successful connection, open the **Block Settings** tab to check the Code Flash configurations.

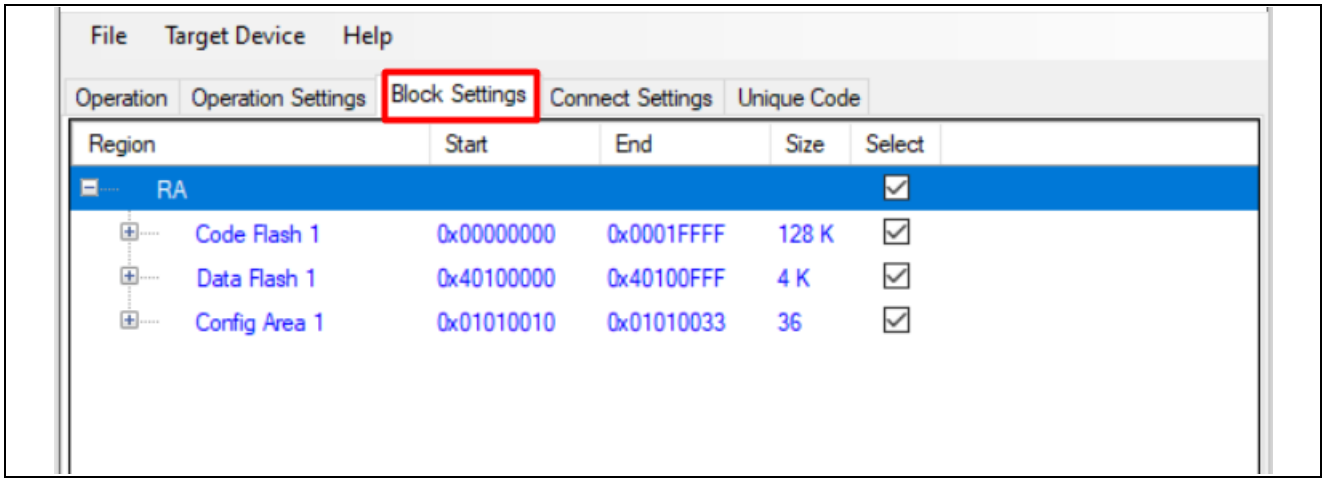


Figure 79. Code Flash configurations

Navigate to the **Operation Settings** tab and choose the **Erase** command.

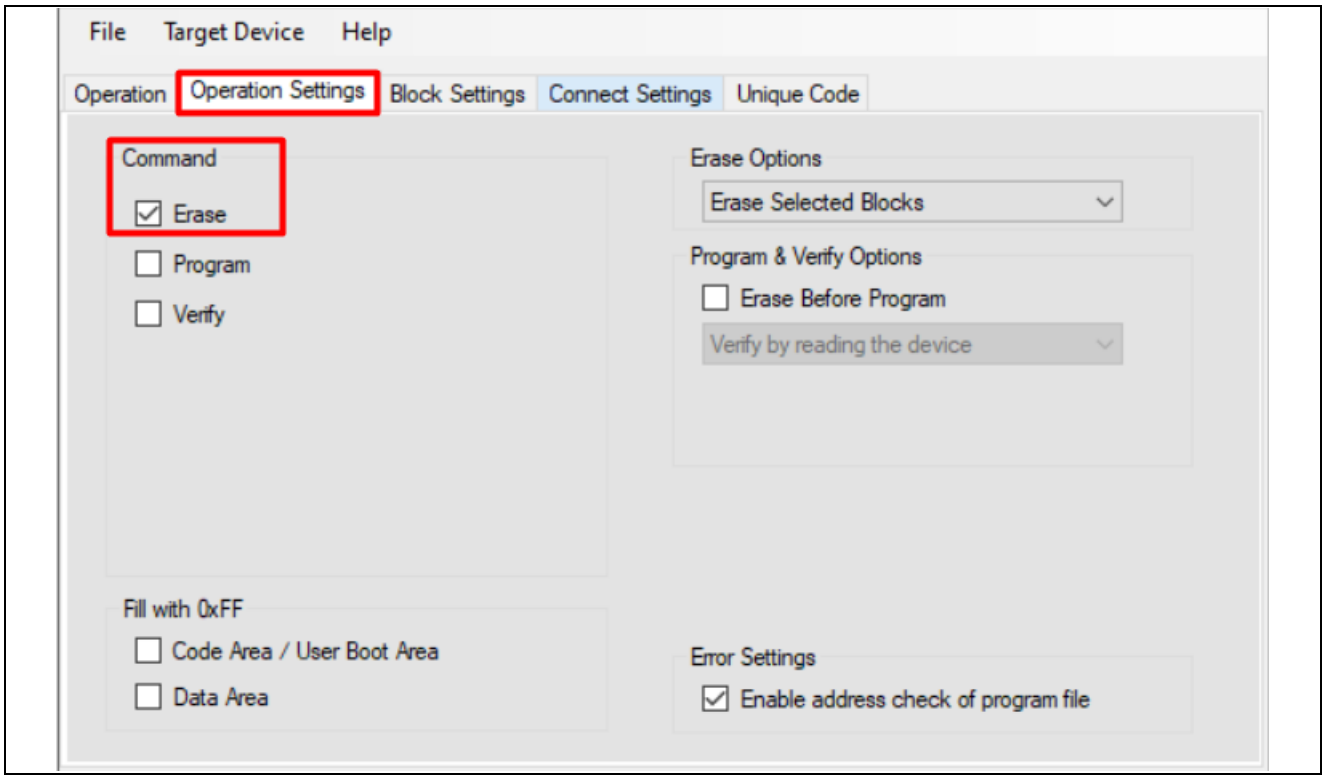


Figure 80. Choose Erase Command

Choose the **Operation** tab and select **Start**.

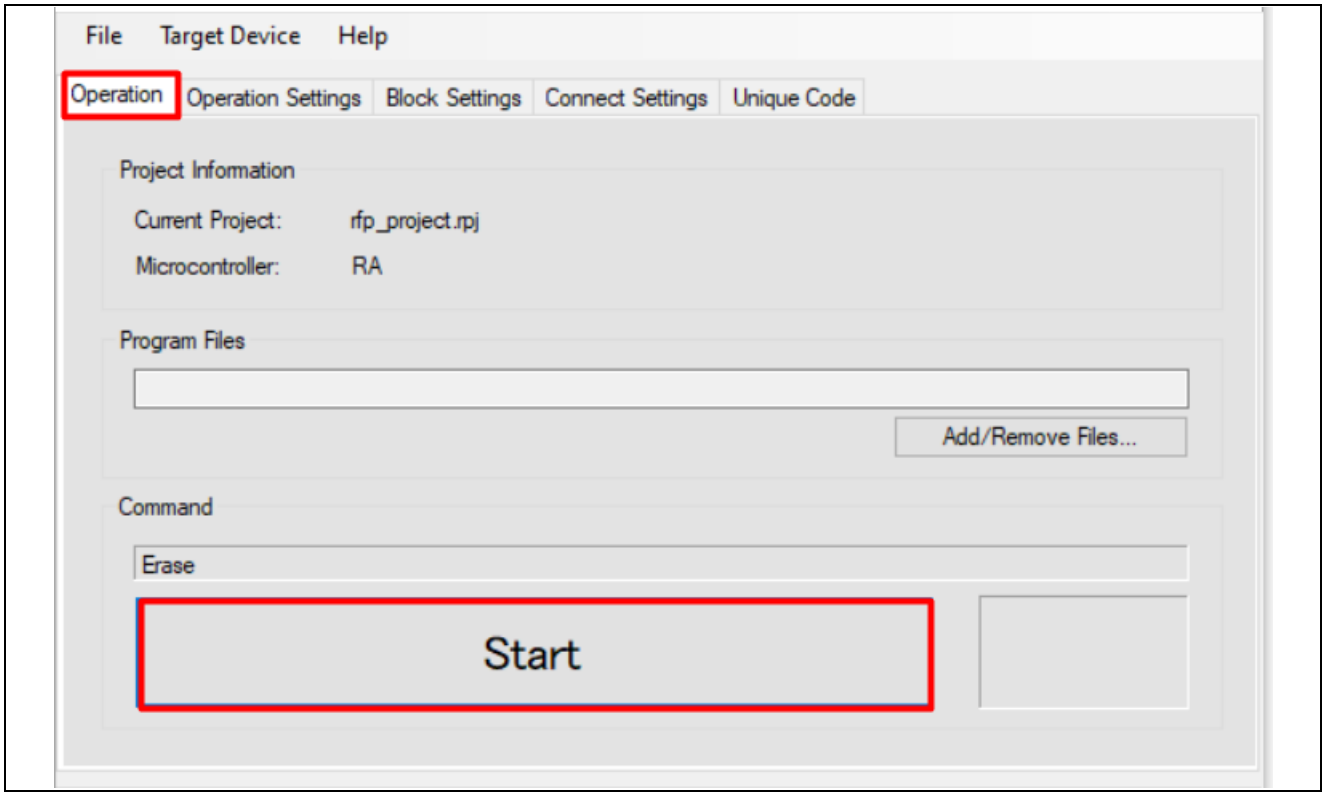


Figure 81. Start Erase Chip

When the chip is erased, **Operation completed** will be displayed, as shown in Figure 82.

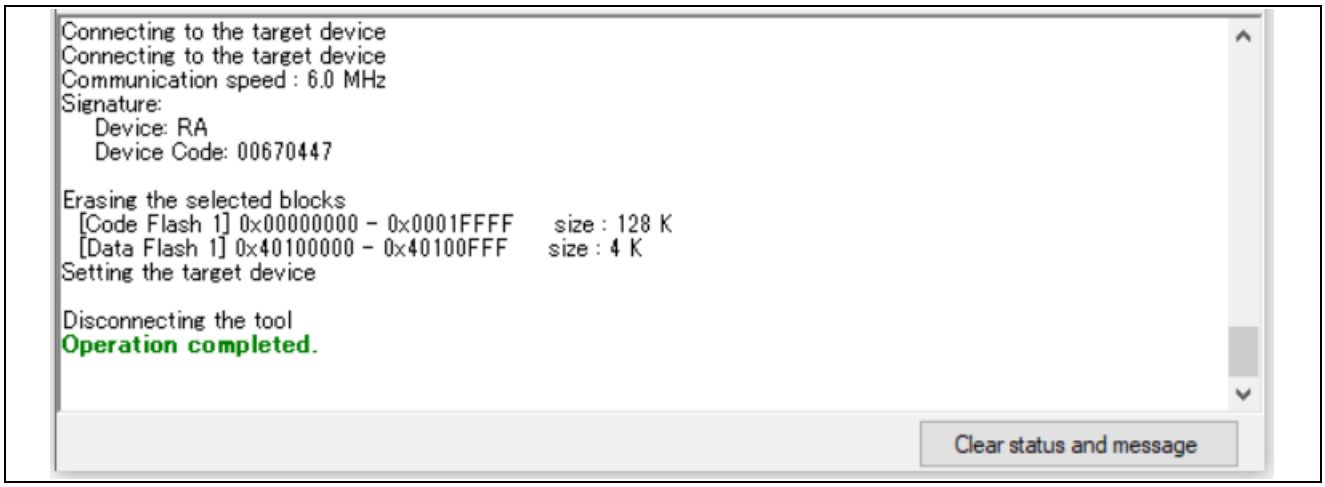


Figure 82. Erase Chip Succeeded

Repeat the same procedure for FPB-RA0E1.

7.1.2 Use the J-Flash Lite

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

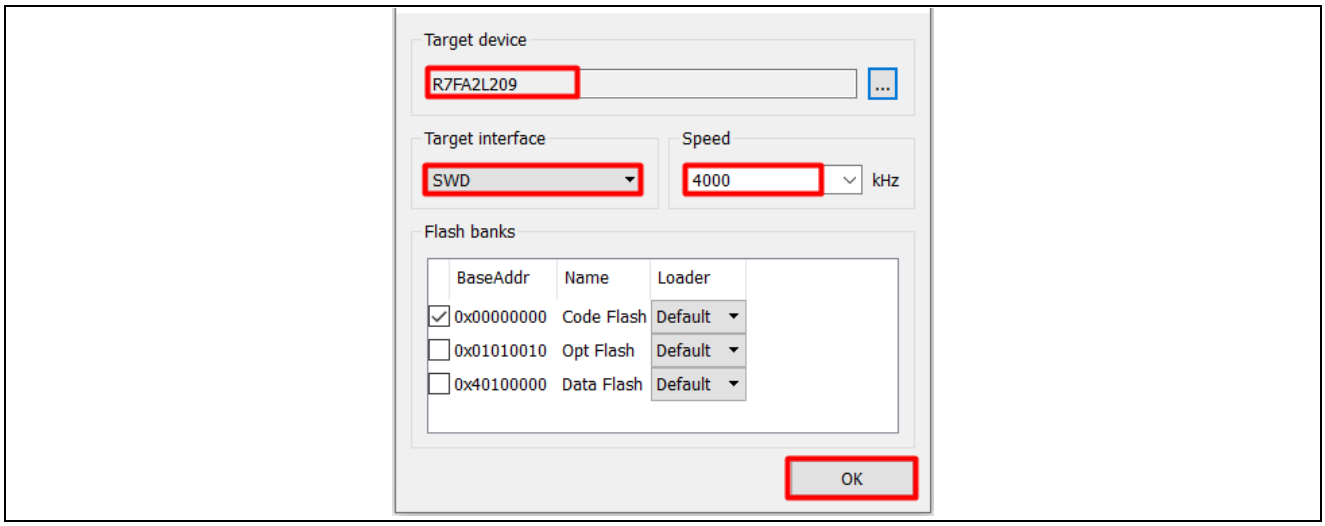


Figure 83. Launch J-Flash Lite

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

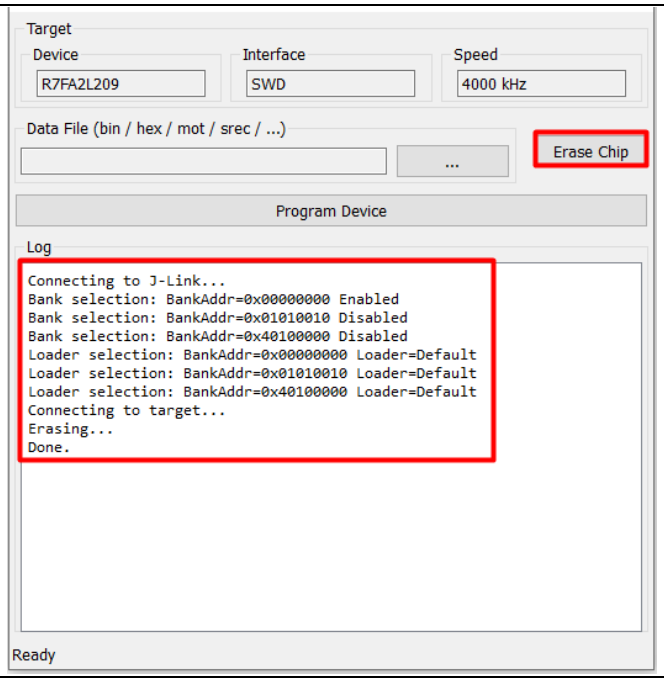


Figure 84. Erase MCU Succeeded

Repeat the same procedure for FPB-RA0E1 by selecting the device **R7FA0E107**, as shown in Figure 83.

7.2 Configure the Debugger

Right-click on the **primary_app_usb_ek_ra2l2** project, and **open the** Debug Configurations**: primary_app_usb_ek_ra2l2 > Debug As > Debug Configurations**

Navigate to **Debugger > Debug Tool Settings** and **Set Allow caching of flash contents to** No**, as shown in** Figure 85**.**

If set to yes, when debugging the bootloader applications, the memory window information may show the wrong information.

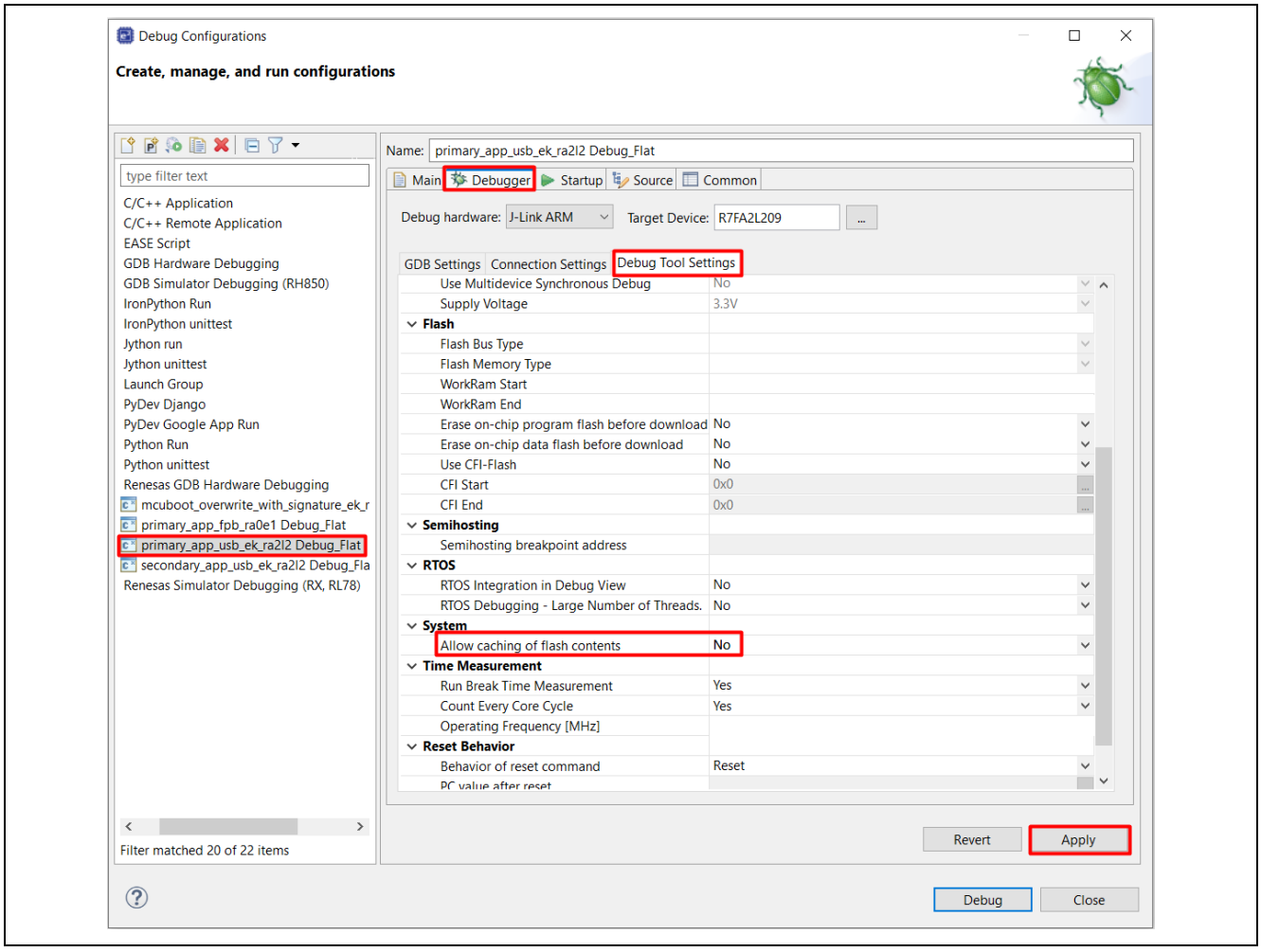


Figure 85. Disable Flash Content Caching

Make sure the **primary_app_usb_ek_ra2l2 Debug_Flat** is selected and select the **Startup** tab.

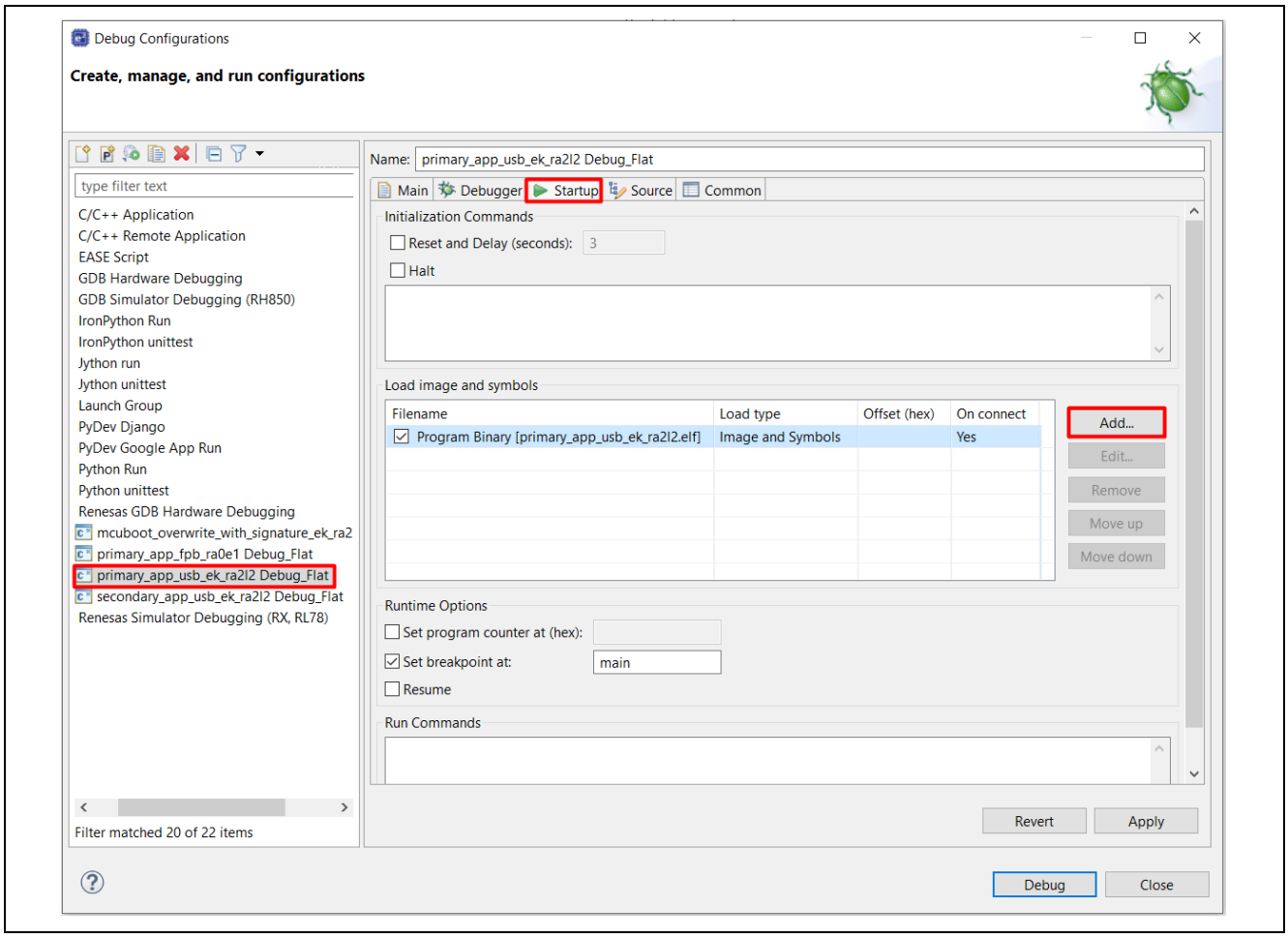


Figure 86. Configure the Primary Project Debug Startup

Click **Add…** then **Workspace** and navigate to and select

mcuboot_overwrite_with_signature_ek_ra2l2.elf

from the debug folder. Click **OK**.

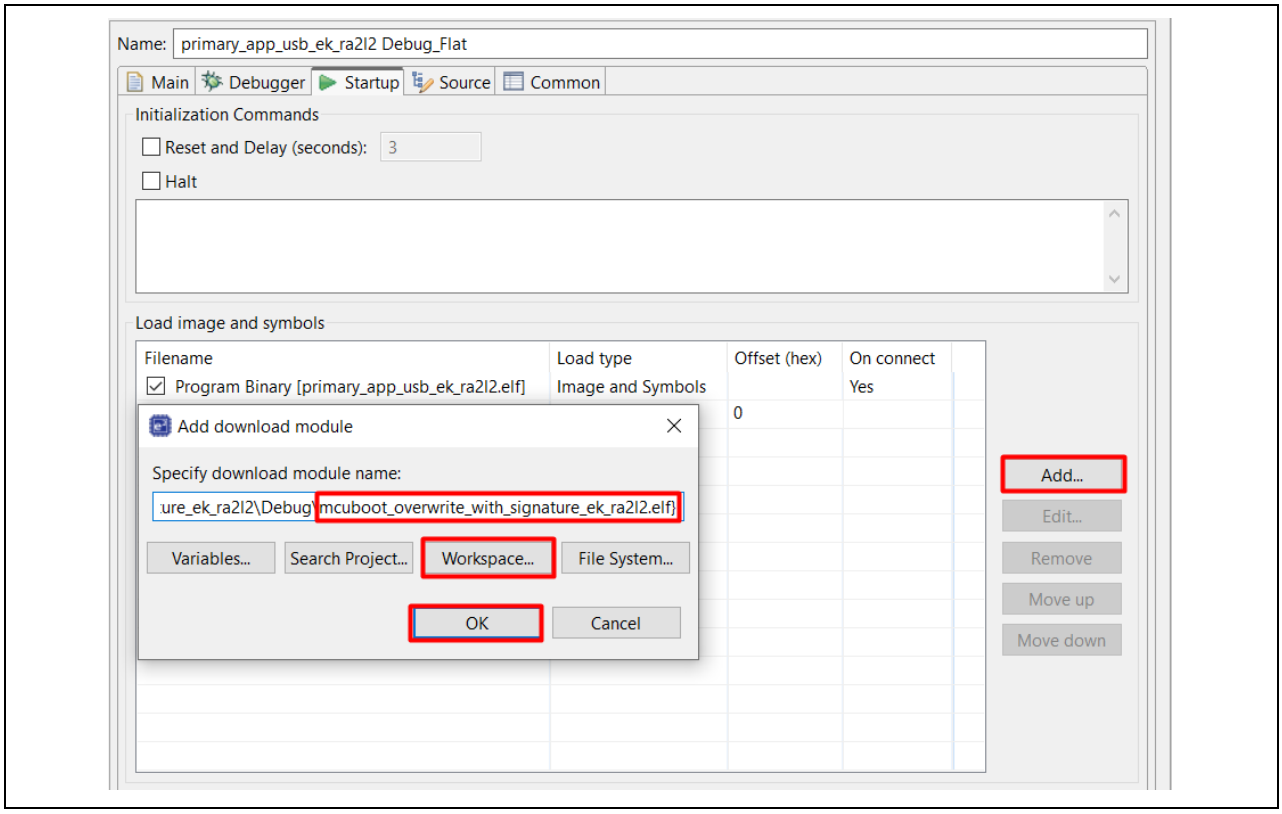


Figure 87. Add the Bootloader Project to Debug Configuration

Change the load type of the Program Binaries for the **primary_app_usb_ek_ra2l2** project to **Symbols only** by clicking on the cell for load type and selecting **Symbols only** from the drop-down menu.

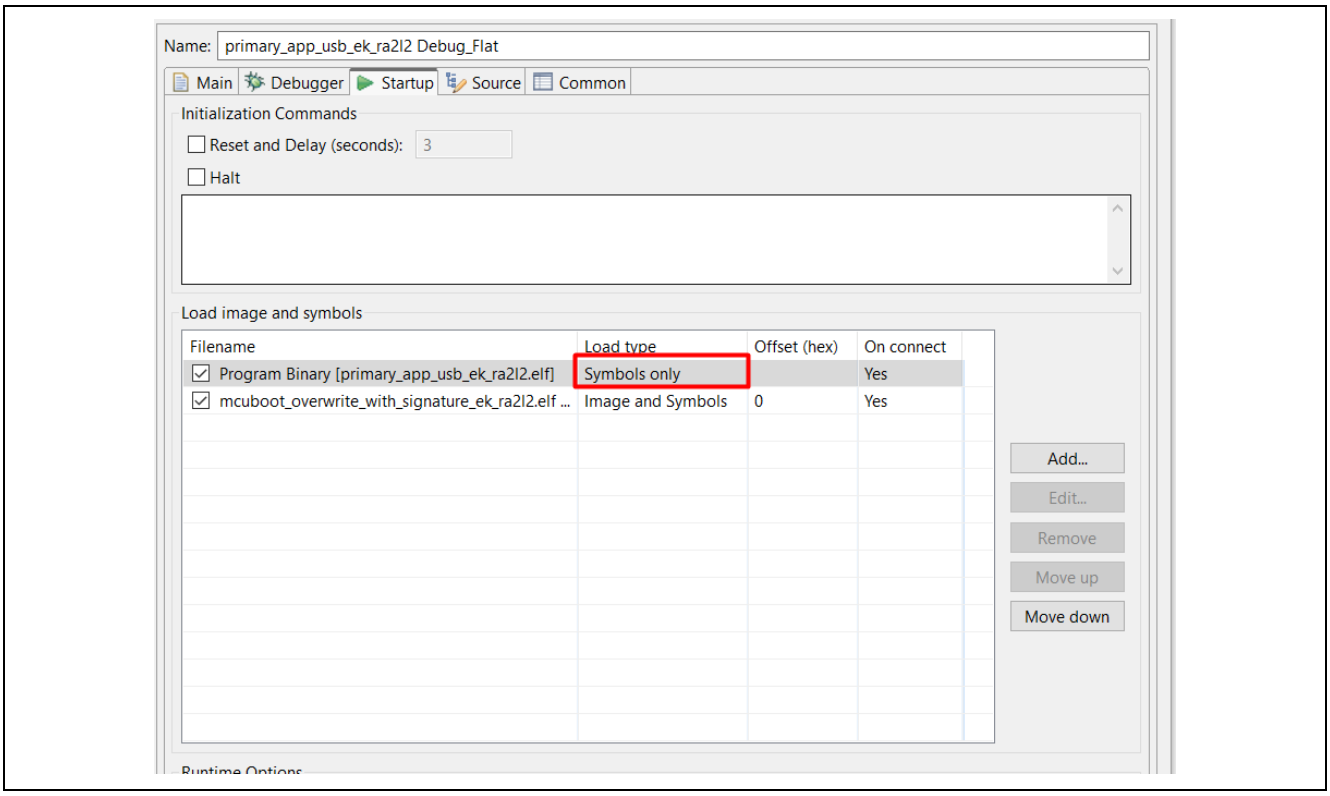


Figure 88. Select to Load Symbols Only for the Application Project

Next, configure the Debug Configuration to include the **Raw Binary** of the signed primary application for download.

Click **Add…** then **Workspace** and navigate to and select

primary_app_usb_ek_ra2l2.bin.signed

from the debug folder. Click **OK**.

Then, change the **Load type** to **Raw Binary**.

Note that the **Offset (hex)** setting of the signed primary image is the size of the bootloader (refer to Figure 68). Figure 89 is an example of downloading the signed primary image for the overwrite project.

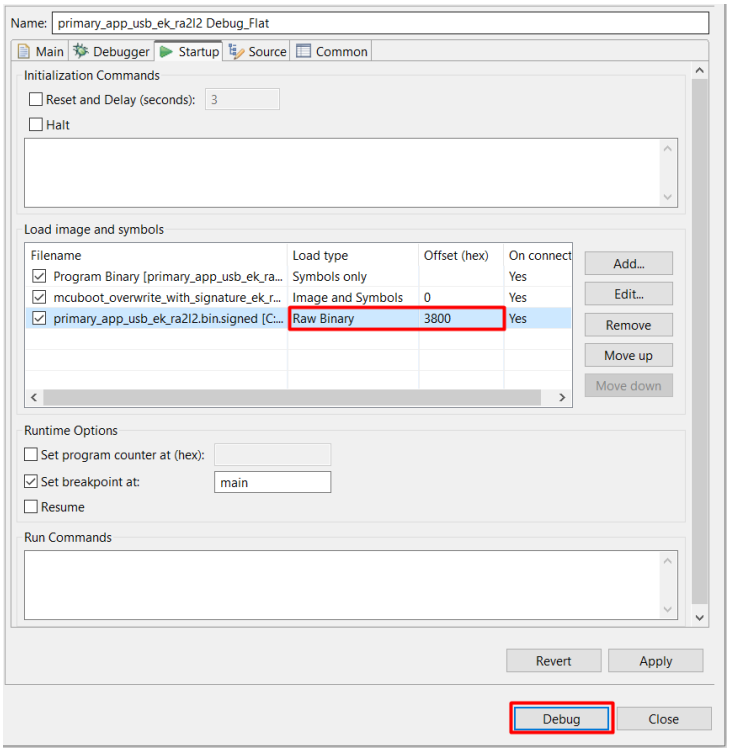


Figure 89. Include the Raw Binary of the signed image in the download

Click **Debug**. The debugger should hit the Reset_Handler() in the bootloader project.

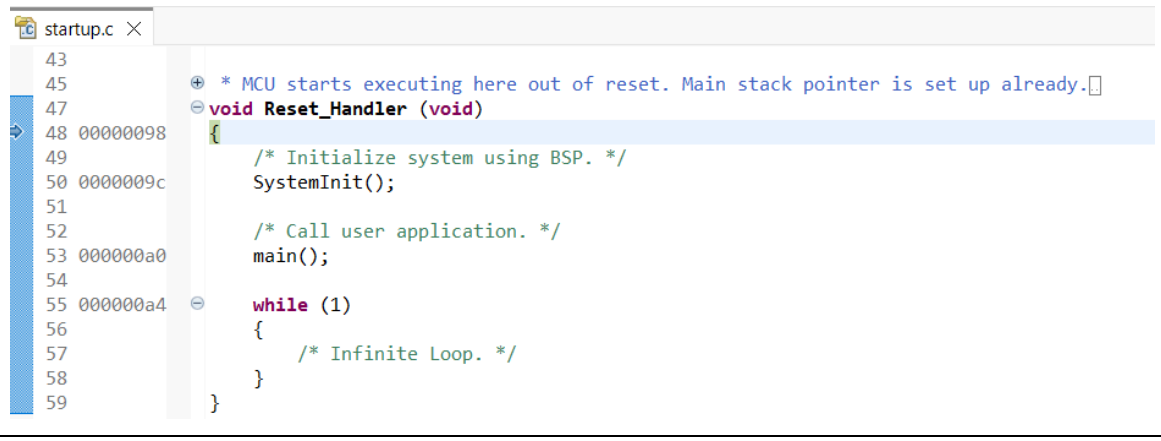


Figure 90. Start the Application Execution

Click **Resume** twice to run the project. The bootloader and the primary application project will be programmed and then the primary application project will be booted.

For the EK-RA2L2: The Red, Blue, and Green LEDs light up in consecutive order.

For the FPB-RA0E1: The two Green LEDs light up in consecutive order.

All actions described in this section should be performed similarly for RA0E1.

7.3 Open the Tera Term Terminal for RA2L2

Open the Tera Term Terminal and set up the following configurations.

1. Launch Tera Term and choose the USB Serial Port (COM number may be different for your setup), as shown in Figure 91. Click **OK.**

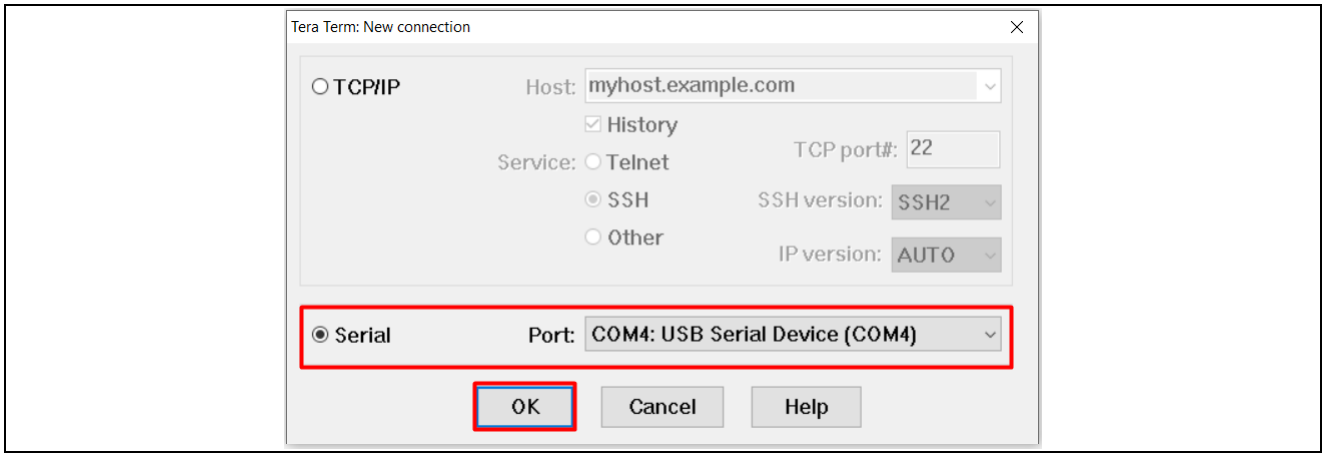


Figure 91. Open the USB COM Port

2. The following text will be displayed.

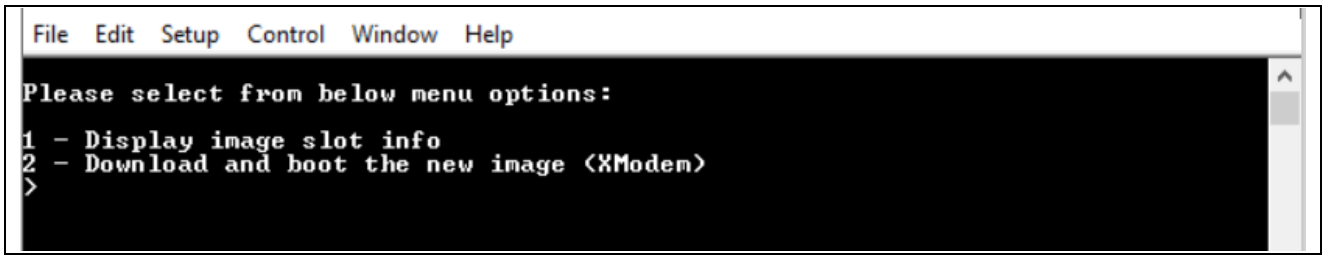


Figure 92. Tera Term Menu

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

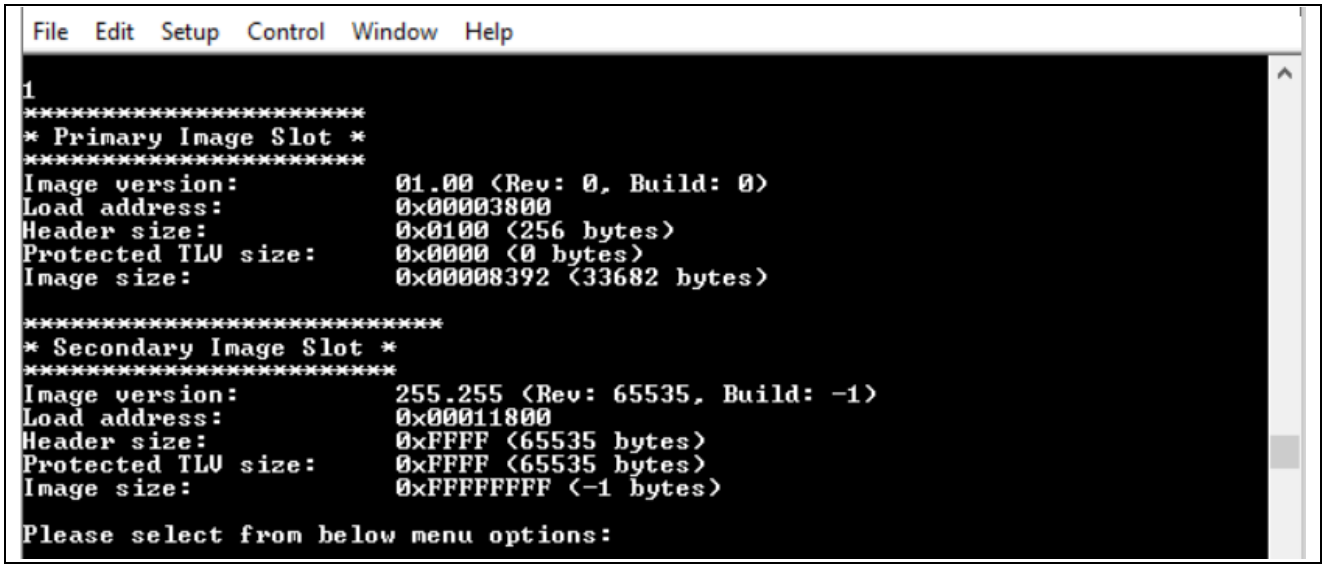


Figure 93. Primary and Secondary Slot Status

7.4 RA0E1 - Open the J-Link RTT Viewer

Open the JLink RTT Viewer and set up the following configurations. Set up the search range as 0x20000000 0x8000

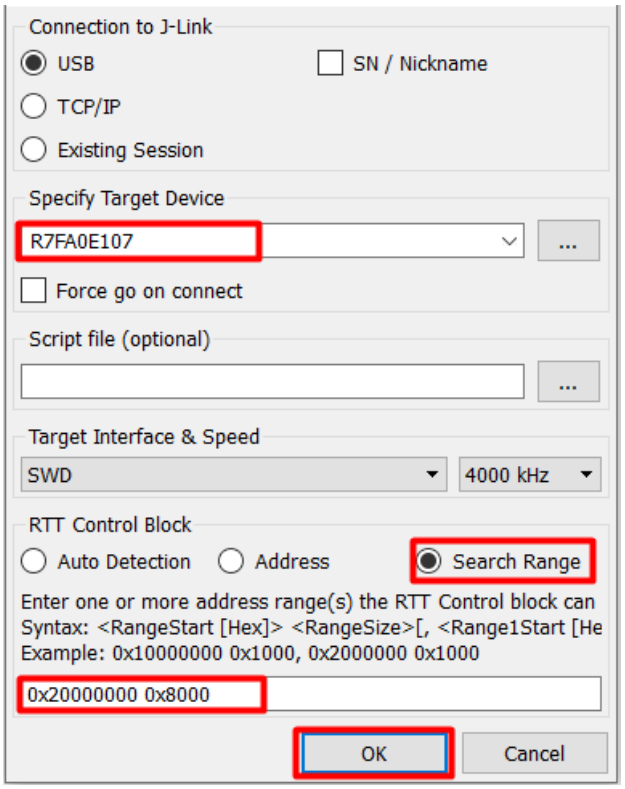


Figure 94. Configure the RTT Viewer

Click **OK** and observe the following output on the RTT Viewer. This output shows that the Primary application is being executed. The message displayed indicates the upgrade mode and the Primary Image is running.

```
00>
00> Running the Primary Application with Overwrite Update Mode. All two LEDs light up in consecutive order.
```

Figure 95. RTT Viewer Output from the Primary Application

8. Mastering and Delivering a New Application

This section explains how to prepare the Secondary Application from the Primary Application and how to download the Secondary Application to the target board.

8.1 Prepare a Secondary Image

The secondary application can be created by modifying the existing application. Import the initial project to the same workspace and rename the new project.

Click **File** > **Import…** and select **General > Rename & Import Existing C/C++ Project into Workspace**.

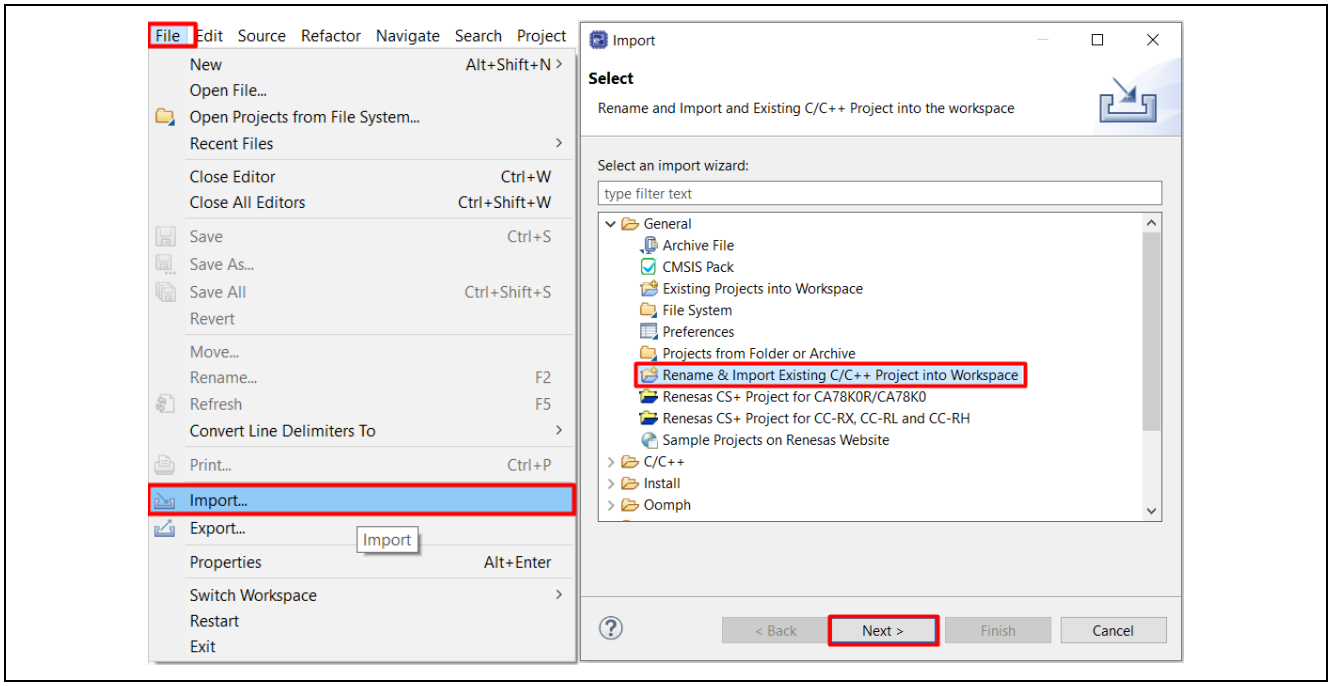


Figure 96. Select Rename and Import the Primary Application

Click **Next**. When the **Import** window opens, name the project and click **Browse** for **Select root directory** as shown in Figure 97.

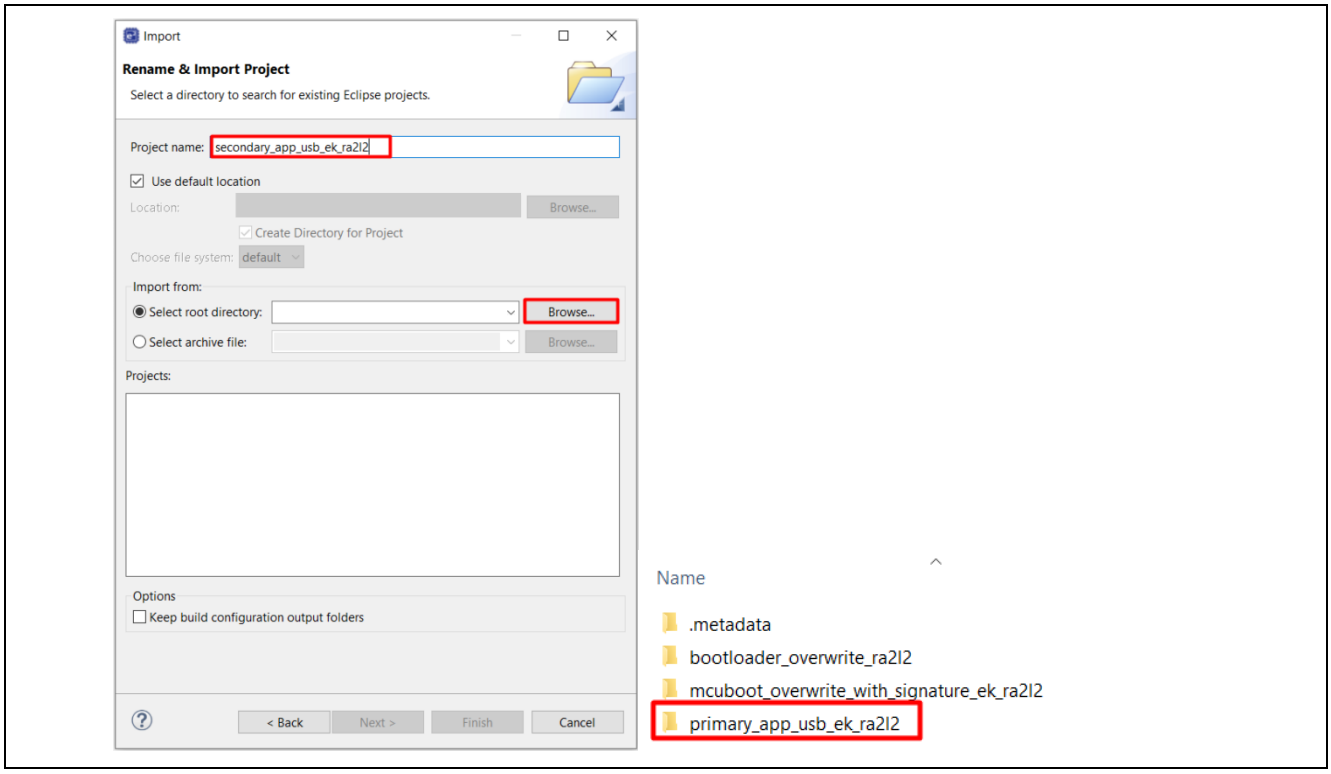


Figure 97. Rename the Project

Name the new project based on Table 4.

Table 4. Project Naming

Bootloader Project Name	Primary Application Project Name	Secondary Application Project Name
mcuboot_overwrite_with_signature_ek_ra2l2	primary_app_usb_ek_ra2l2	secondary_app_usb_ek_ra2l2
mcuboot_overwrite_with_signature_fpb_ra0e1	primary_app_fpb_ra0e1	secondary_app_fpb_ra0e1

Figure 98 is an example screenshot when importing the secondary project as secondary_app_usb_ek_ra2l2.

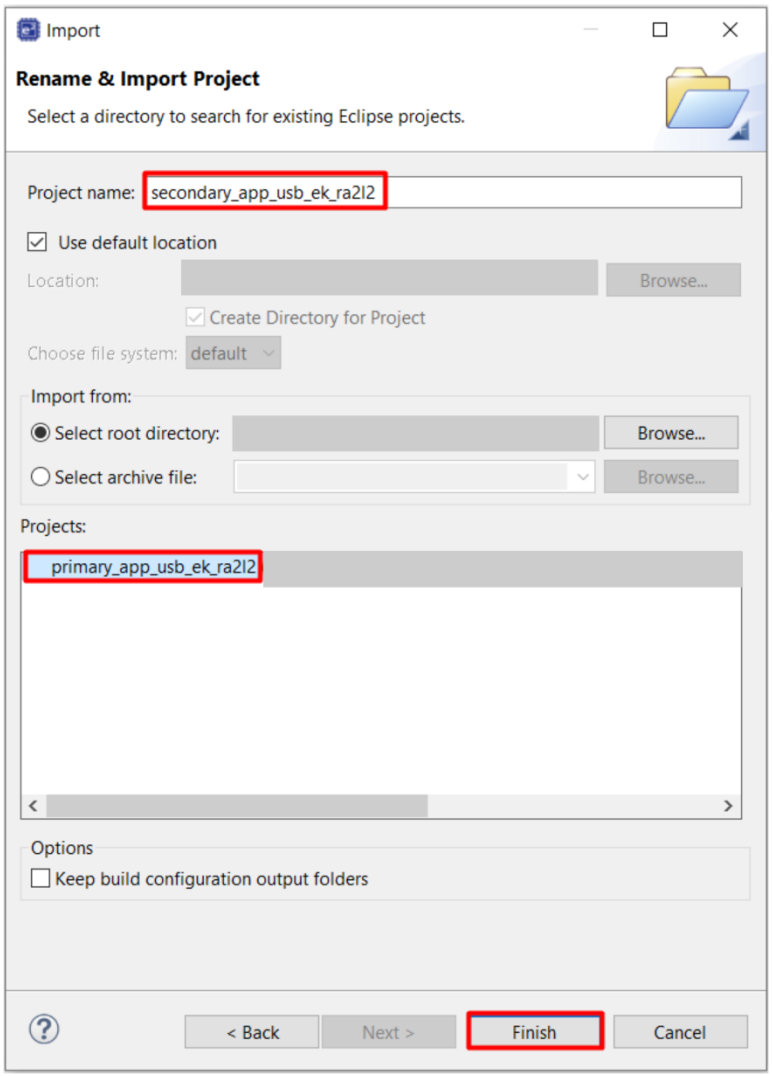


Figure 98. Import Secondary Project as secondary_app_usb_ek_ra2l2

Click **Finish,** and the new application project will be created.

Repeat the same procedure for the FPB-RA0E1 board.

8.2 Update Existing Application to a New Application

To demonstrate that the application is updated, the code can be updated to show new behavior:

- Update the application so that LEDs (LED1, LED2, LED3) on the board light up simultaneously.
- Update the RTT Viewer message to show this is the update image for FPB-RA0E1.
- Change the Environment Variable for the Secondary Image Version.

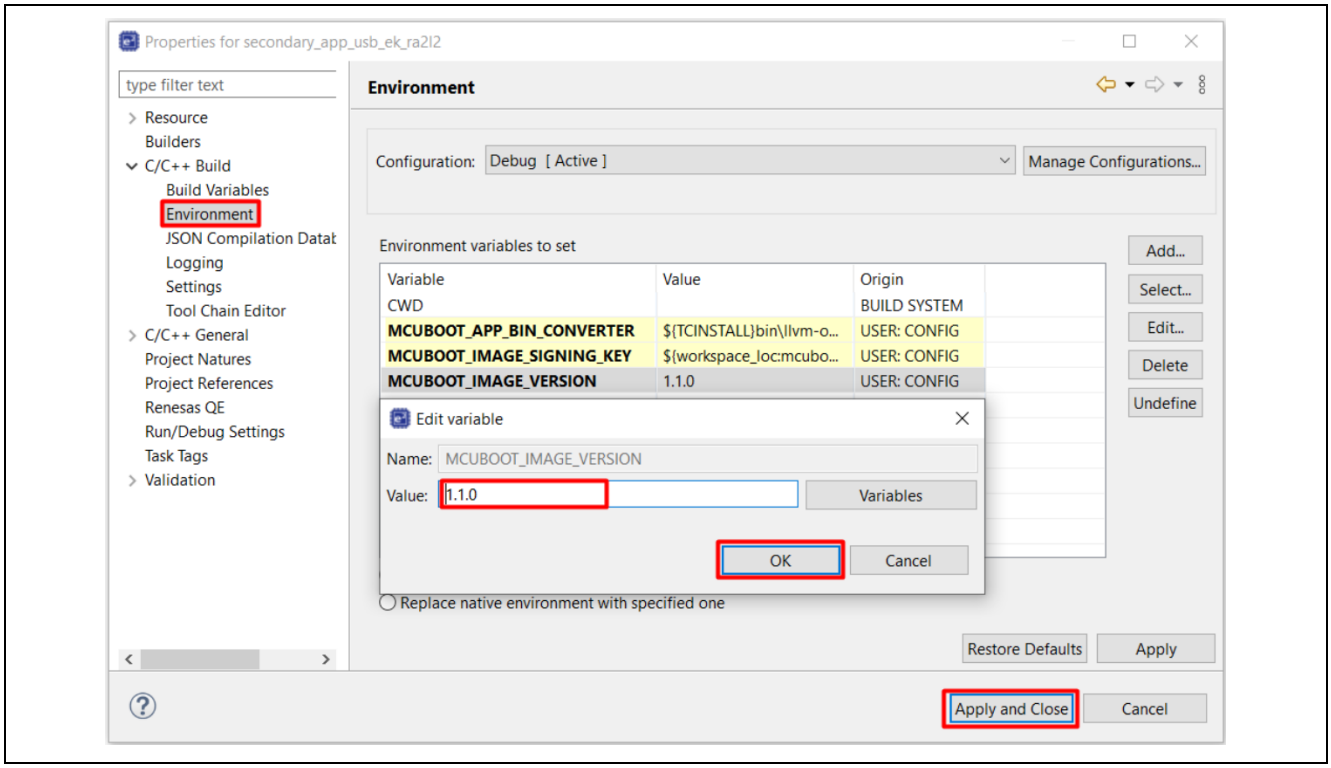


Figure 99. Change MCUBOOT_IMAGE_VERSION Variable

For simplicity, developers can use the supplied files.

- For EK-RA2L2, unzip the file **ek_ra2l2_secure_bootloader_via_usb_pcdc.zip**

  Copy all files under **\src** to the **\src** folder for the newly established project.

- For FPB-RA0E1, unzip the file **fpb_ra0e1_secure_bootloader.zip**

  Copy all files under **\src** to the **\src** folder for the newly established project.

When importing the primary application, the **Build Variable** and the **Environment Variables** as well as the Debug configurations are automatically imported.

Click **Generate Project Content** and compile the new application. The signed binary for the new application is now created.

- For EK-RA2L2, the **secondary_app_usb_ek_ra2l2.bin.signed** will be created in Debug folder.
- For FPB-RA0E1, the **secondary_app_fpb_ra0e1.bin.signed** will be created in Debug folder.

For completeness, delete the .jlink file of the old project under the root of the newly created project structure.

8.3 Downloading and Booting the Secondary Application

This section will detail two methods for downloading the new application:

- Using the **Load Ancillary File** tool, available in e2 studio, for FPB-RA0E1 board.
- Using the **XMODEM-based Image Downloader**, implemented in Primary Application, for EK-RA2L2 board.

The **Load Ancillary File** method can be used for testing purposes, while the **XMODEM-based Image Downloader** can be used for the production process.

8.3.1 Using the Load Ancillary File tool

Click **Pause** and use the **Ancillary Download** button (which is available under the e2studio Debug view) to load the compiled Secondary Application secondary_app_fpb_ra0e1.bin.signed.

Select the new application image and set the download address. The download address depends on the bootloader flash memory allocation (refer to Figure 69).

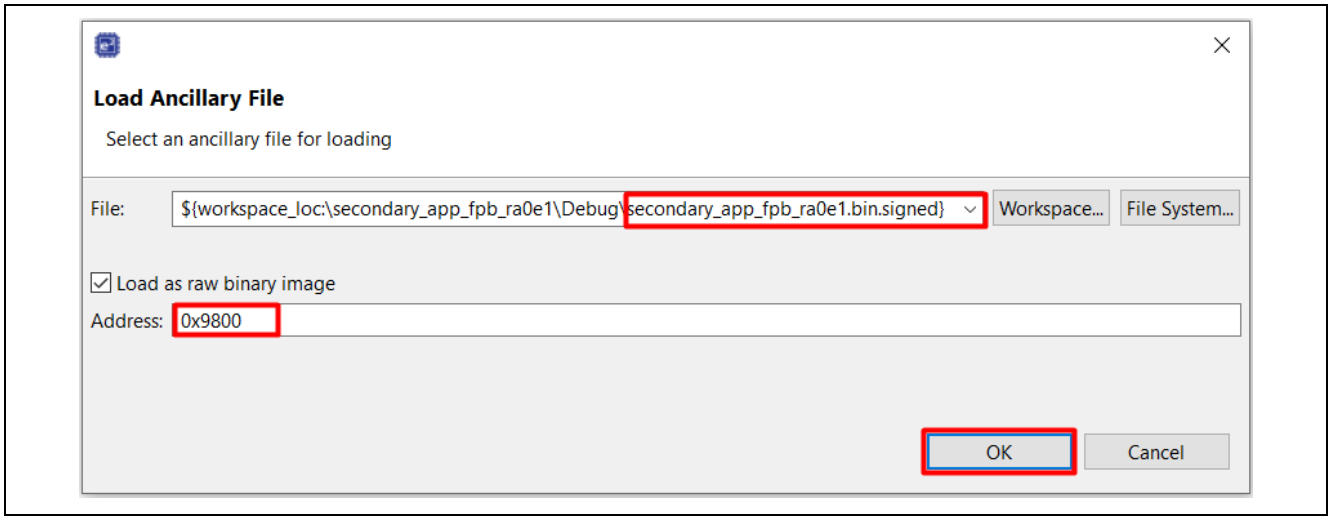


Figure 100. Download the Secondary Application Image in Overwrite Mode for RA0E1

Click OK, and then click **Resume** to allow the system to perform image overwritten and the new image will be booted. The two Green LEDs on board light up simultaneously.

```
00>
00> Running the Secondary Application with Overwrite Update Mode. All two LEDs light up simultaneously.
```

Figure 101. RTT Viewer Output from the Secondary Application

Note that for user-created customized applications, the download address needs to be adjusted by referencing the specific flash layout. Users can refer to Figure 68 to learn how to come up the download address.

8.3.2 Using the XMODEM-based Image Downloader

Refer to Figure 68 to identify the address where the secondary image will be loaded. Define the addresses of the primary image and secondary image in the **/src/xmodem/info_header.h** file, as shown in Figure 102.

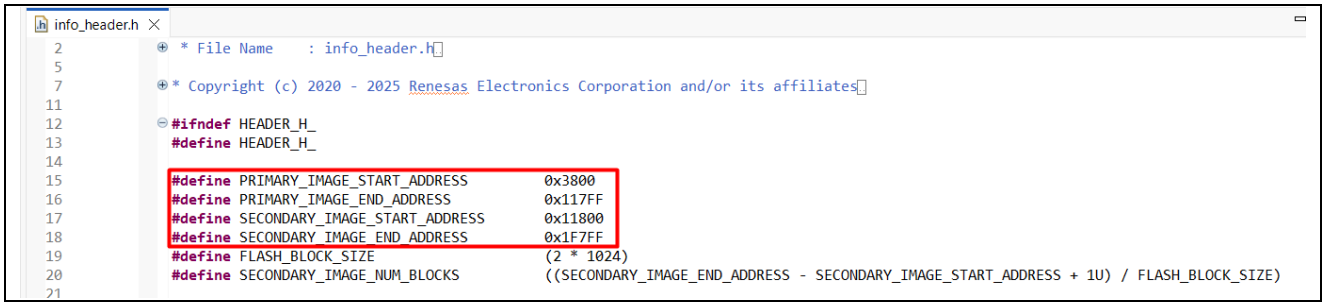


Figure 102. Image Address Configuration

To use the image downloader to load the secondary application image:

1. Choose option **2** from the Tera Term Menu as shown in Figure 92.

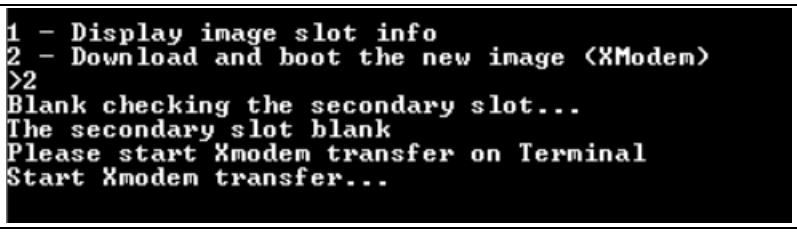


Figure 103. Choose Option 2 to Download the Secondary Image using Xmodem

2. Open the **Transfer** interface of the Tera Term by selecting **File** > **Transfer** > **XMODEM** > **Send**

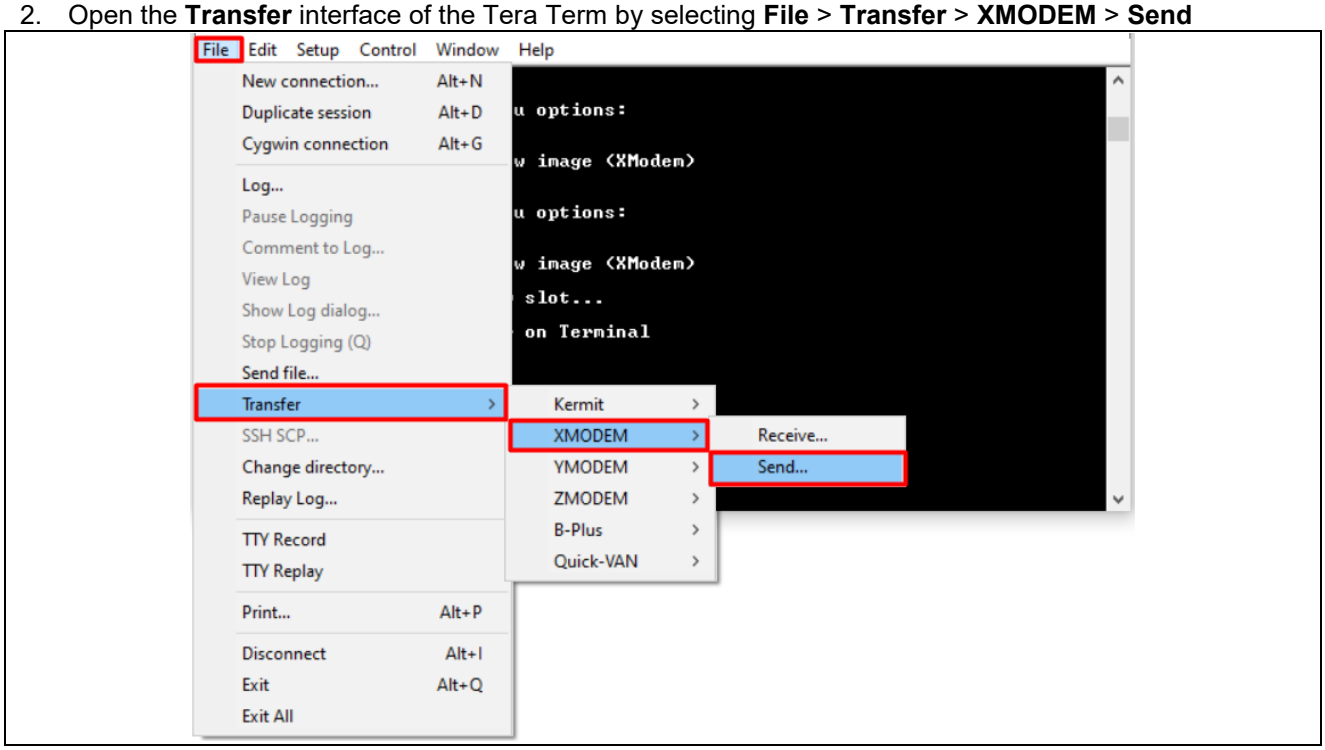


Figure 104. Start Transfer from Tera Term

3. Choose **secondary_app_usb_ek_ra2l2.bin.signed**, then click **Open**.

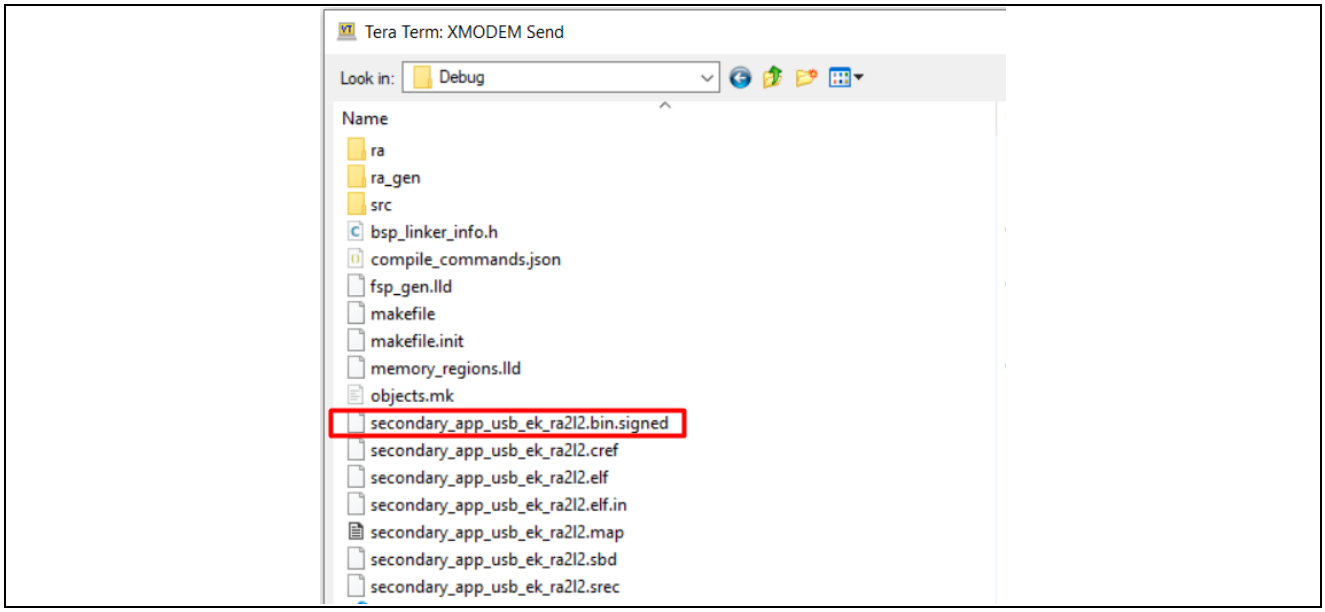


Figure 105. Choose the Signed Secondary Image

4. The secondary image is downloaded and programmed into the secondary slot. The download process takes approximately four seconds.

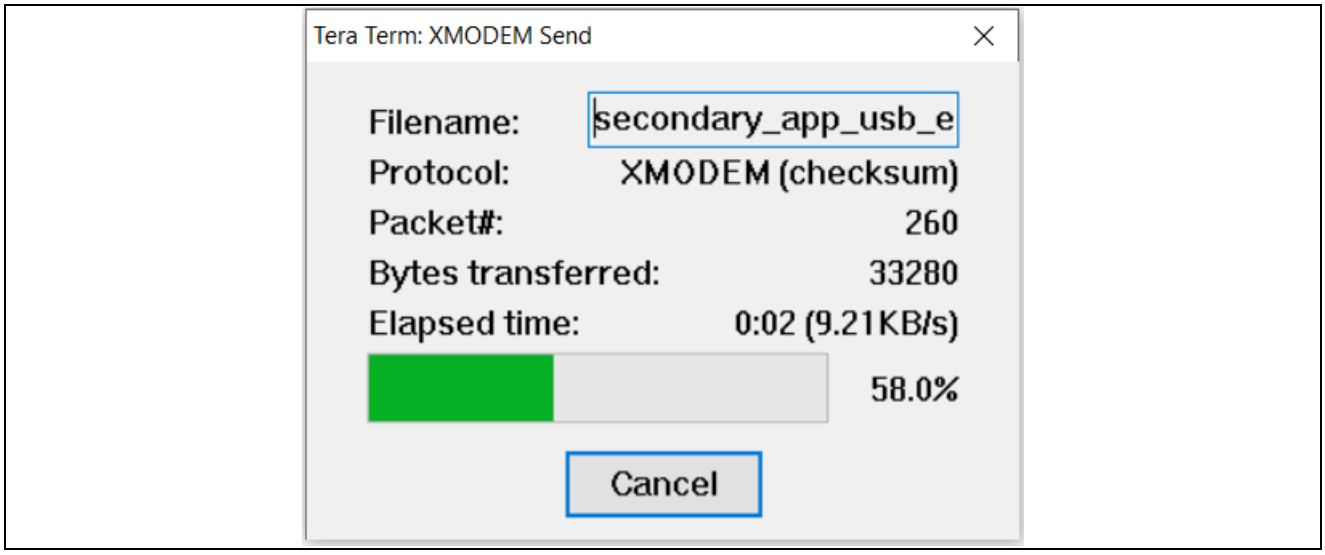


Figure 106. Download the Secondary Image using XMODEM

5. When the secondary application is downloaded, the primary application will reset the system by calling the NVIC_SystemReset() function. Wait for about five seconds, and the menu from the secondary application will be displayed, as shown in Figure 107. The Red, Blue, and Green LEDs light up simultaneously.

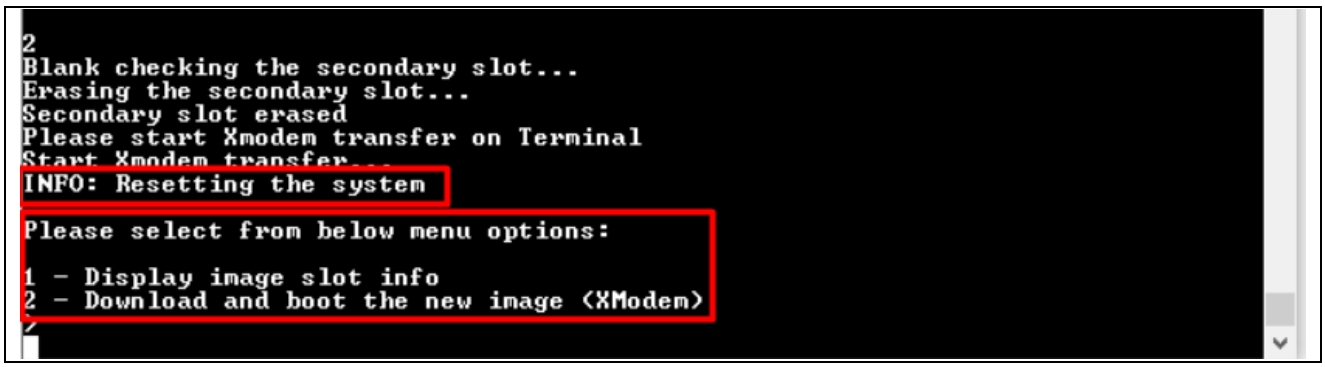


Figure 107. Secondary Image is booted

6. Select option **1** to read the secondary image information.

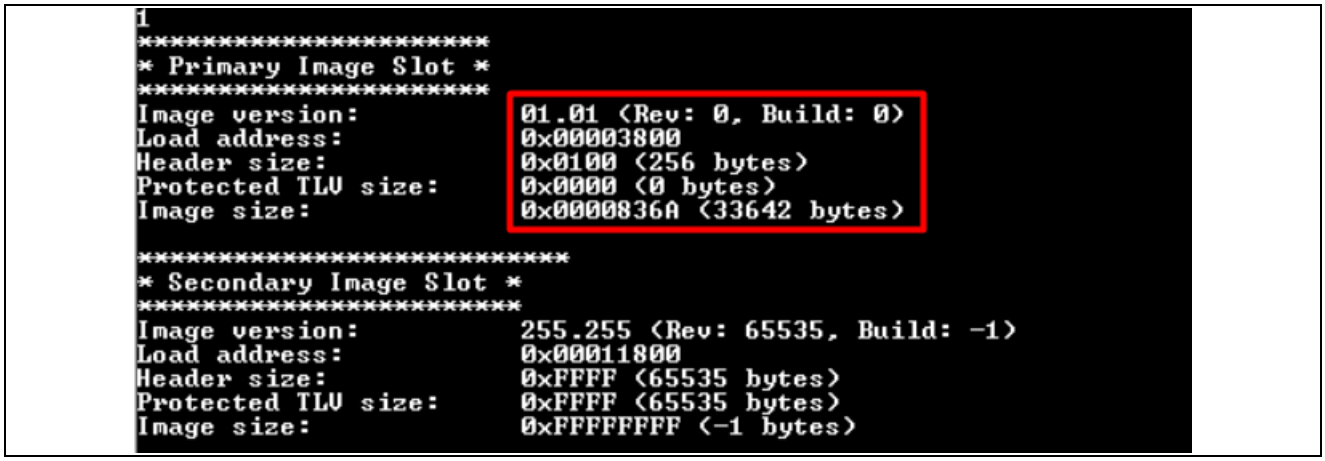


Figure 108. Secondary Image Information

Note - to perform further updates, the new image must have a higher version than the current image in the primary slot. This is because the **Downgrade Prevention (Overwrite Only)** option is used, as shown in Figure 24.

9. Appendix: Build and Execute the Provided Application Projects

9.1 Running the EK-RA2L2 Overwrite Update Mode sample project

Perform the following steps to execute the example projects located in the \ek_ra2l2_secure_bootloader_via_usb_pcdc directory:

1. Import projects to the workspace.

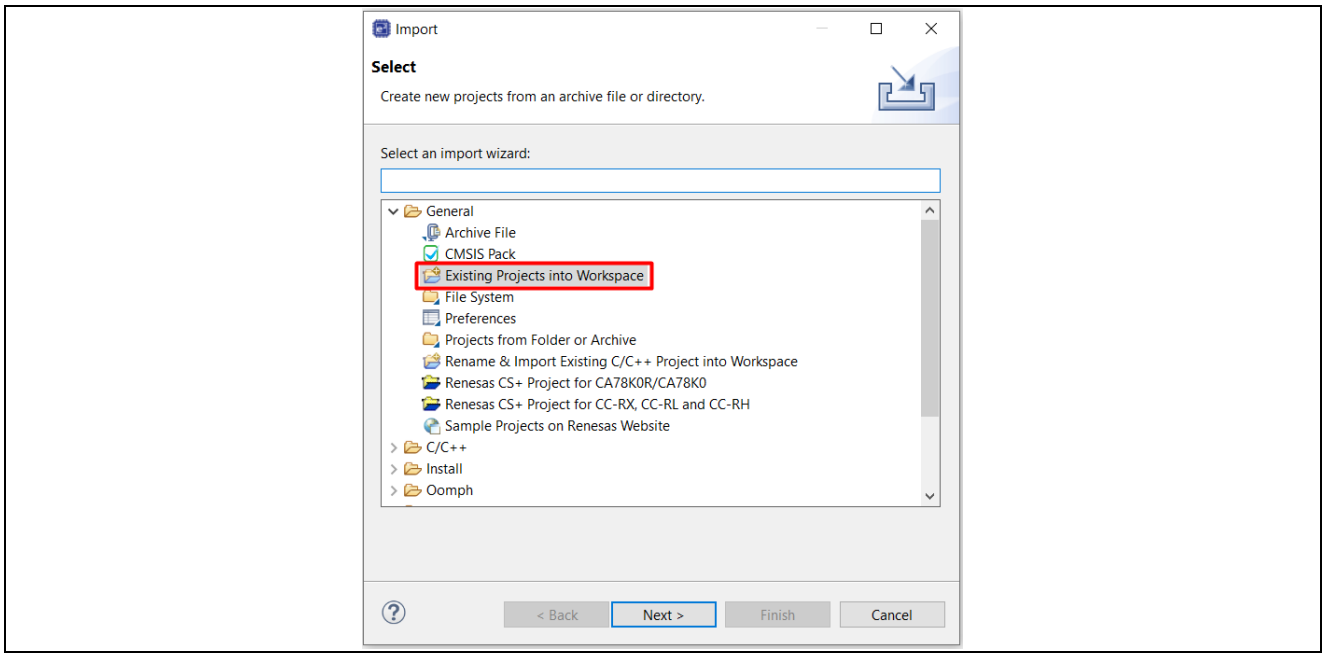


Figure 109. Choose to import “Existing Projects”

2. Install necessary libraries for MCUboot’s scripts. From ra/mcu-tools/MCUboot, open **Command Prompt**. Then run following commands:

   python -m pip install --upgrade pip

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

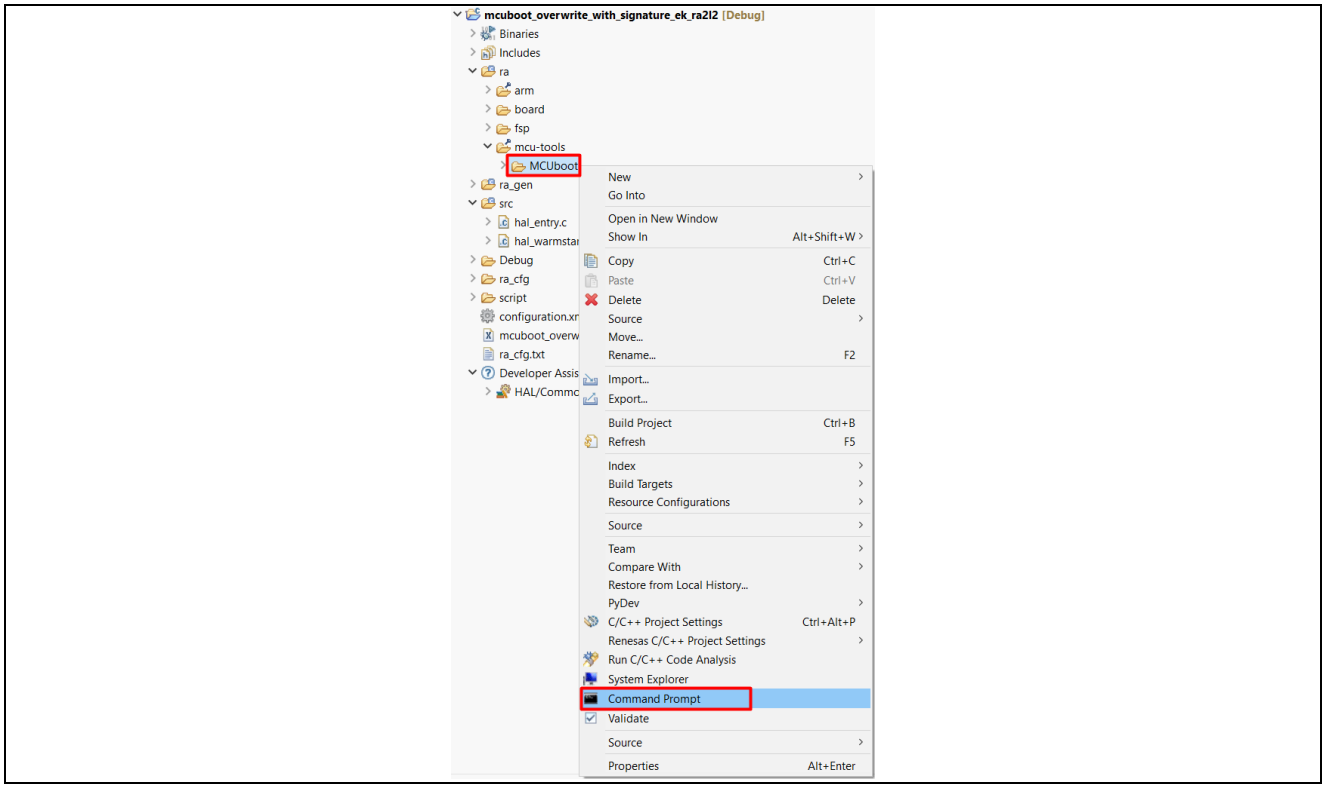


Figure 110. Open the Command Prompt

3. Build the **Solution Project**:

- Right-click on the **Solution Project**: mcuboot_ overwrite_solution_ek_ra2l2.

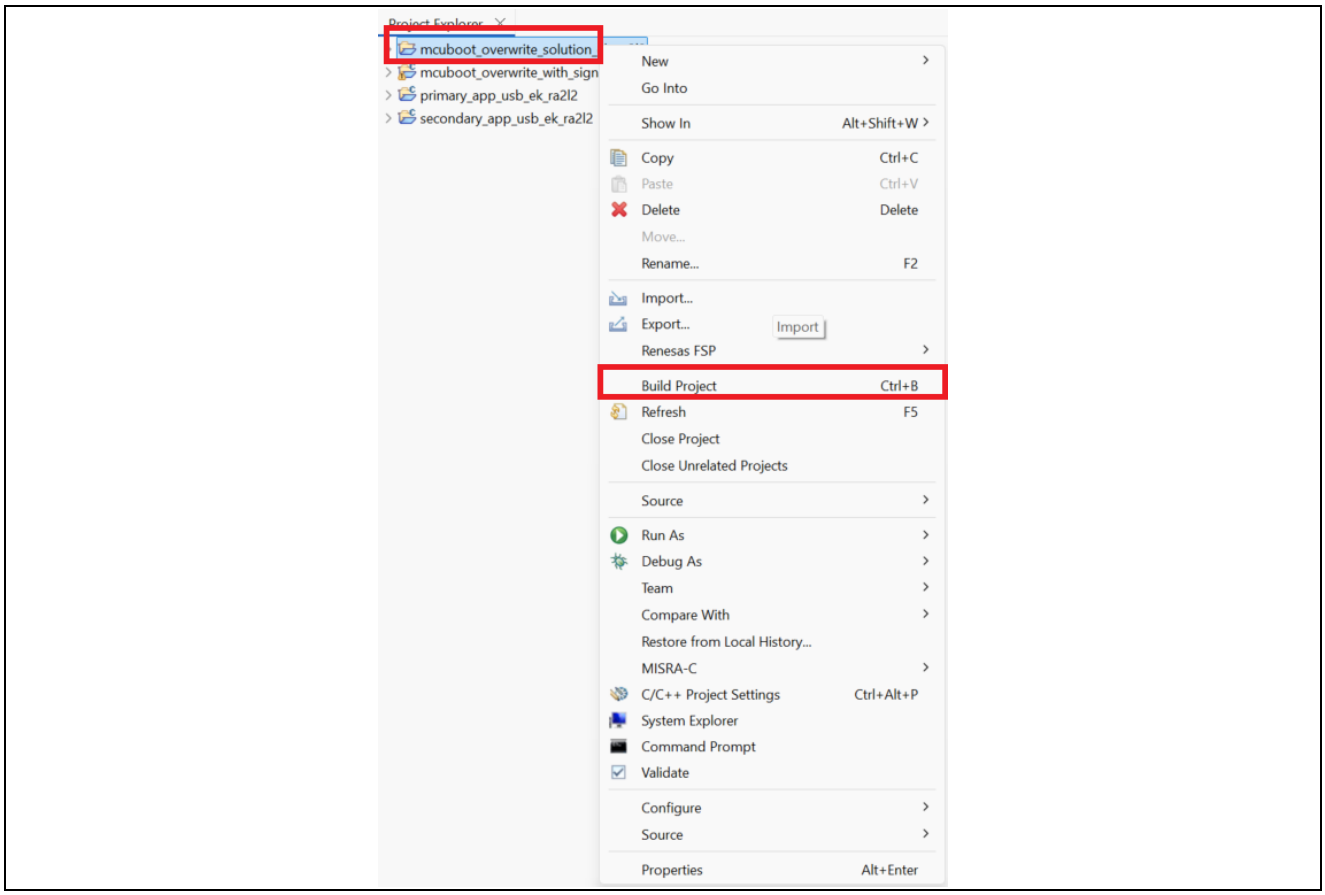


Figure 111. Build the Solution Project

- Select **Build Project**. This command will build all projects within the **Solution Project**, as shown in **Figure** 111. At this point the mcuboot_overwrite_with_signature_ek_ra2l2.elf and primary_app_usb_ek_ra2l2.bin.signed files are generated in the Debug folder.

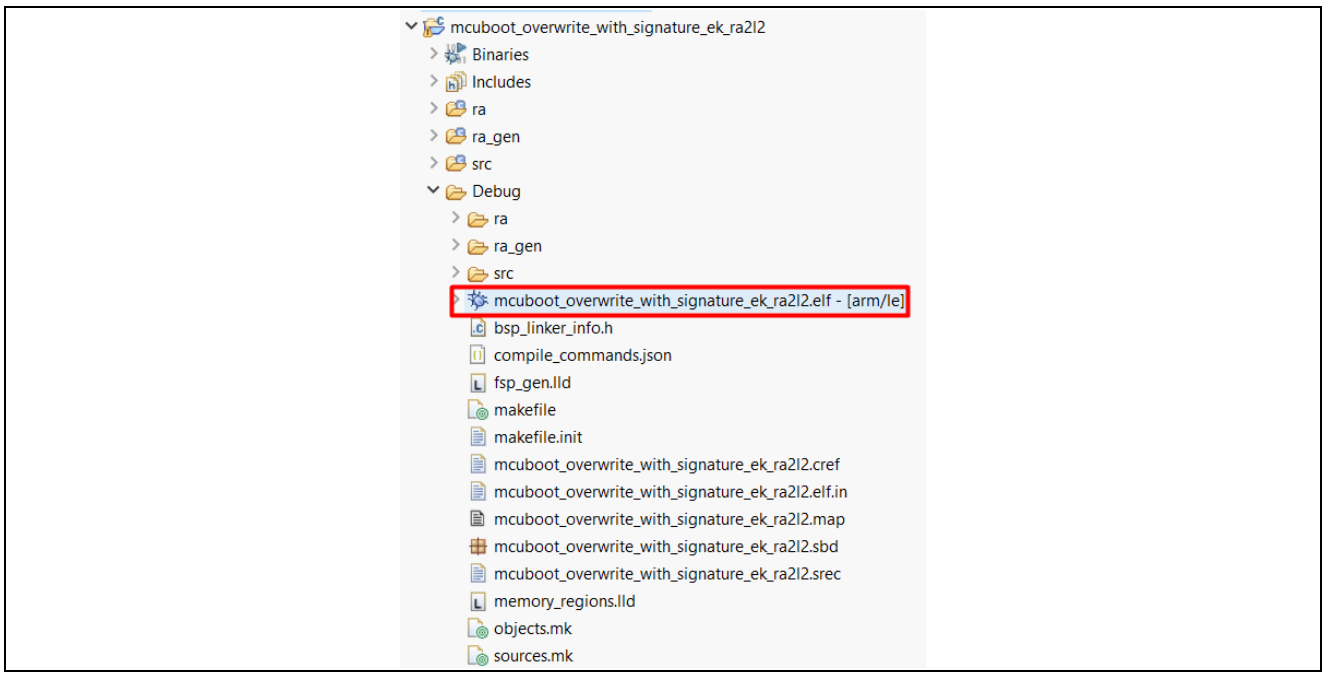


Figure 112. Bootloader Image is generated

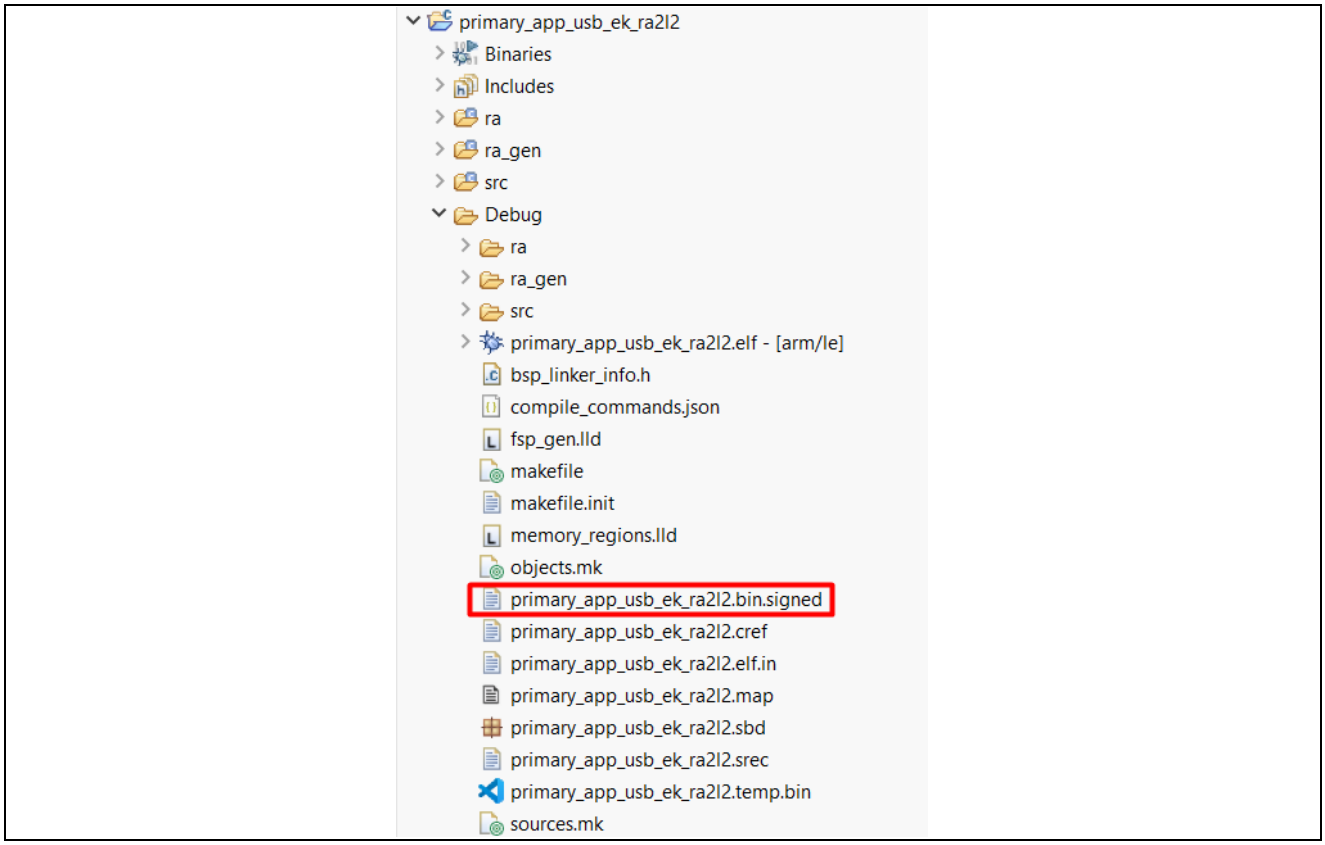


Figure 113. Primary Application Image is generated

- Select **Build Project** for the Secondary Application: secondary_app_usb_ek_ra2l2.

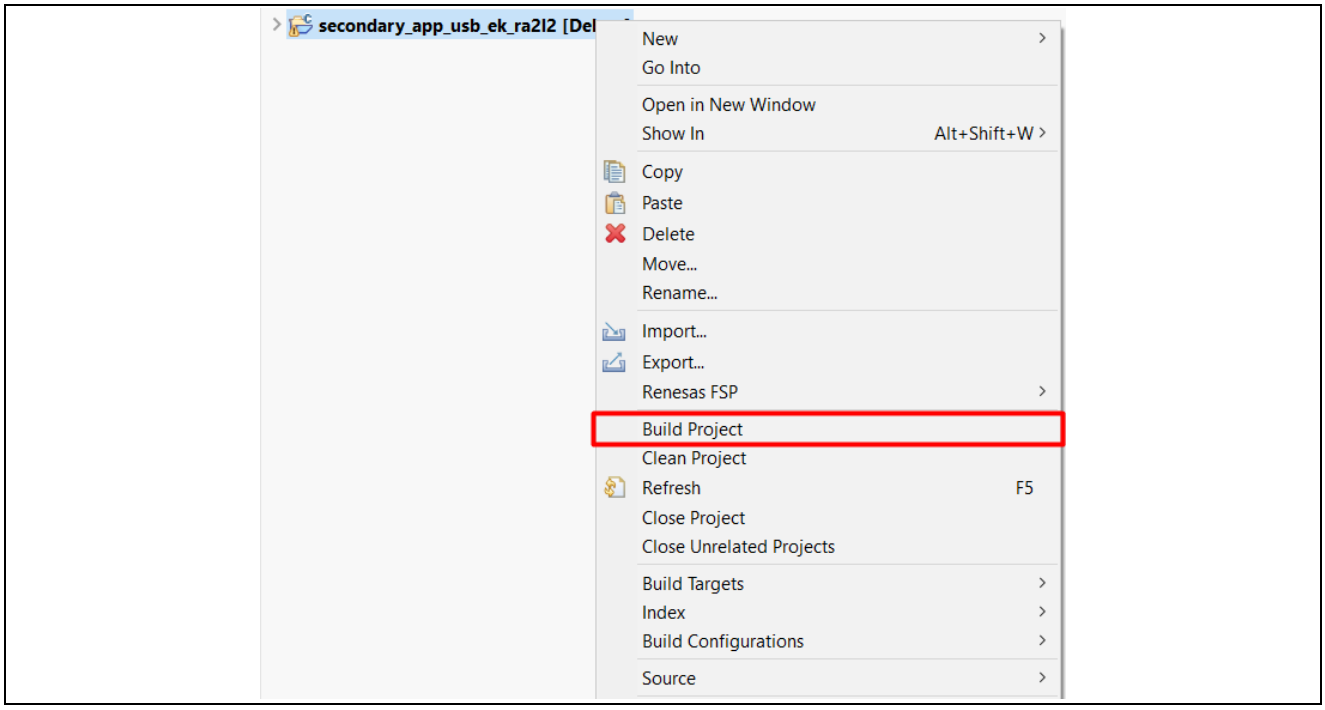


Figure 114. Build the Secondary Application

- Make sure that secondary_app_usb_ek_ra2l2.bin.signed file is generated in the Debug folder.

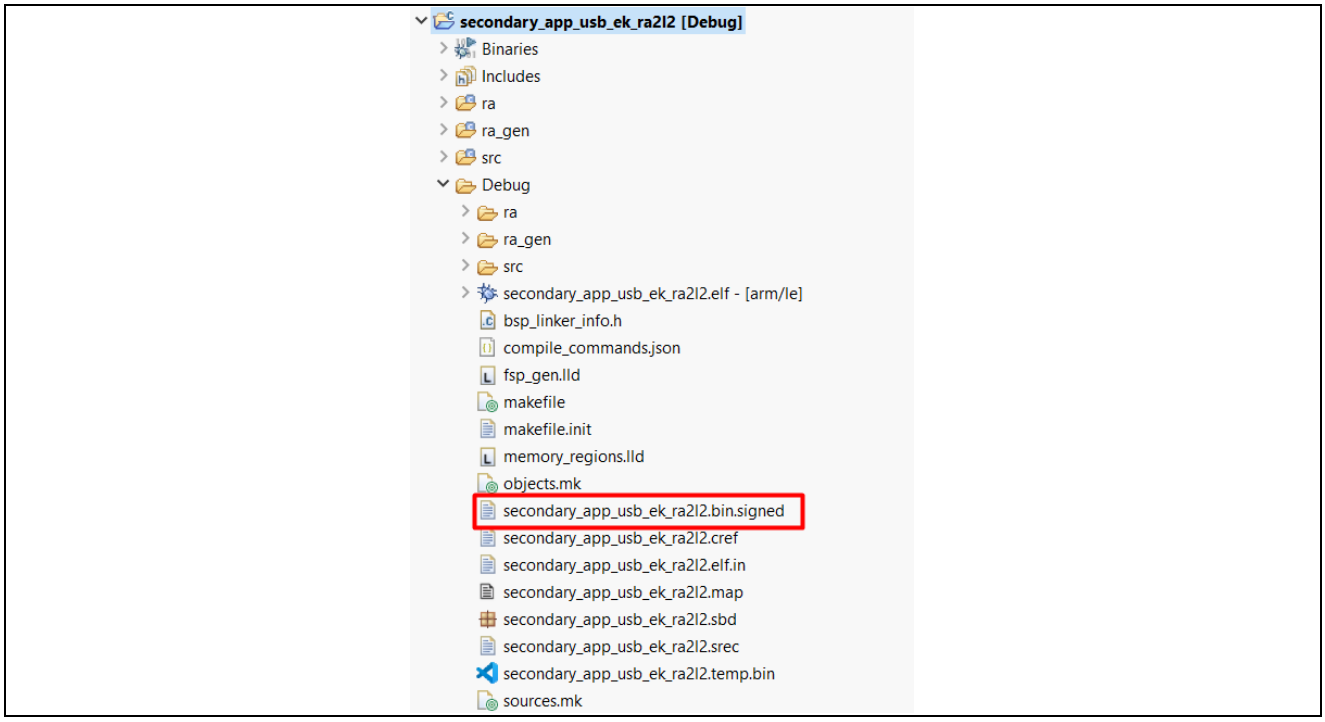


Figure 115. Secondary Application Image is generated

4. Erase the entire chip following instructions in section 7.1.
5. Debug the application from project primary_app_usb_ek_ra2l2.

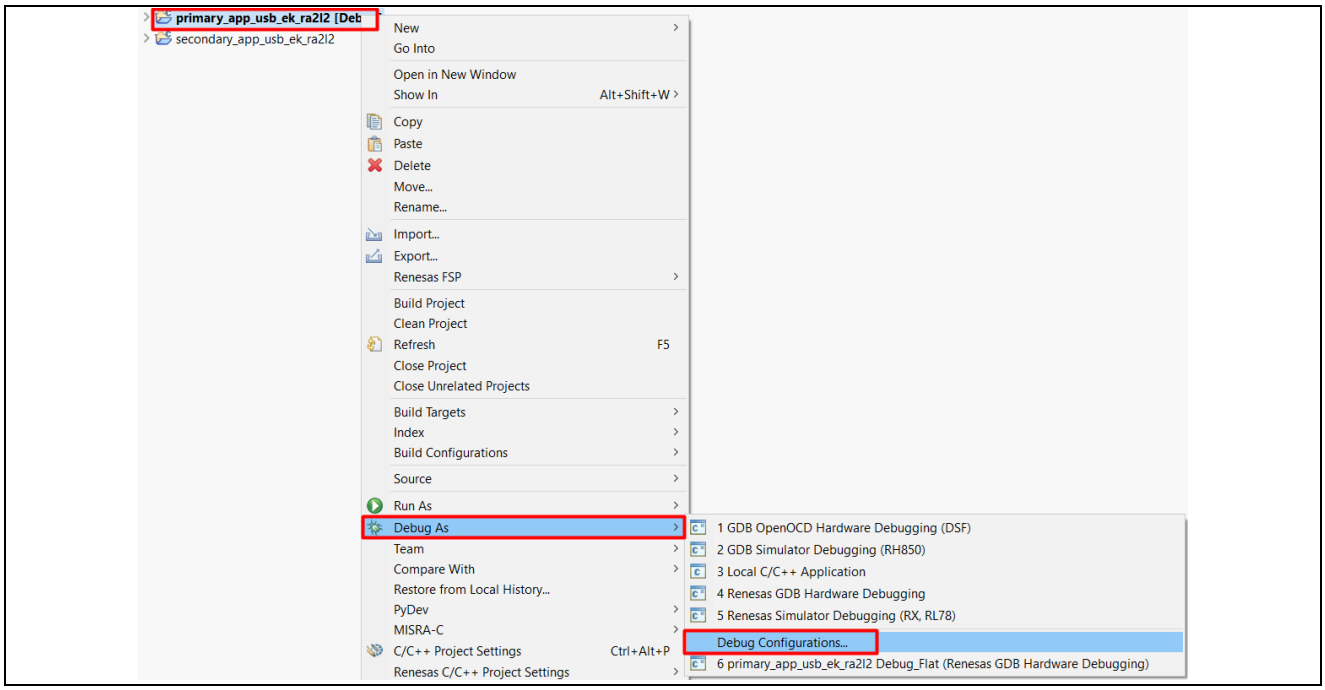


Figure 116. Open the Debug Configurations

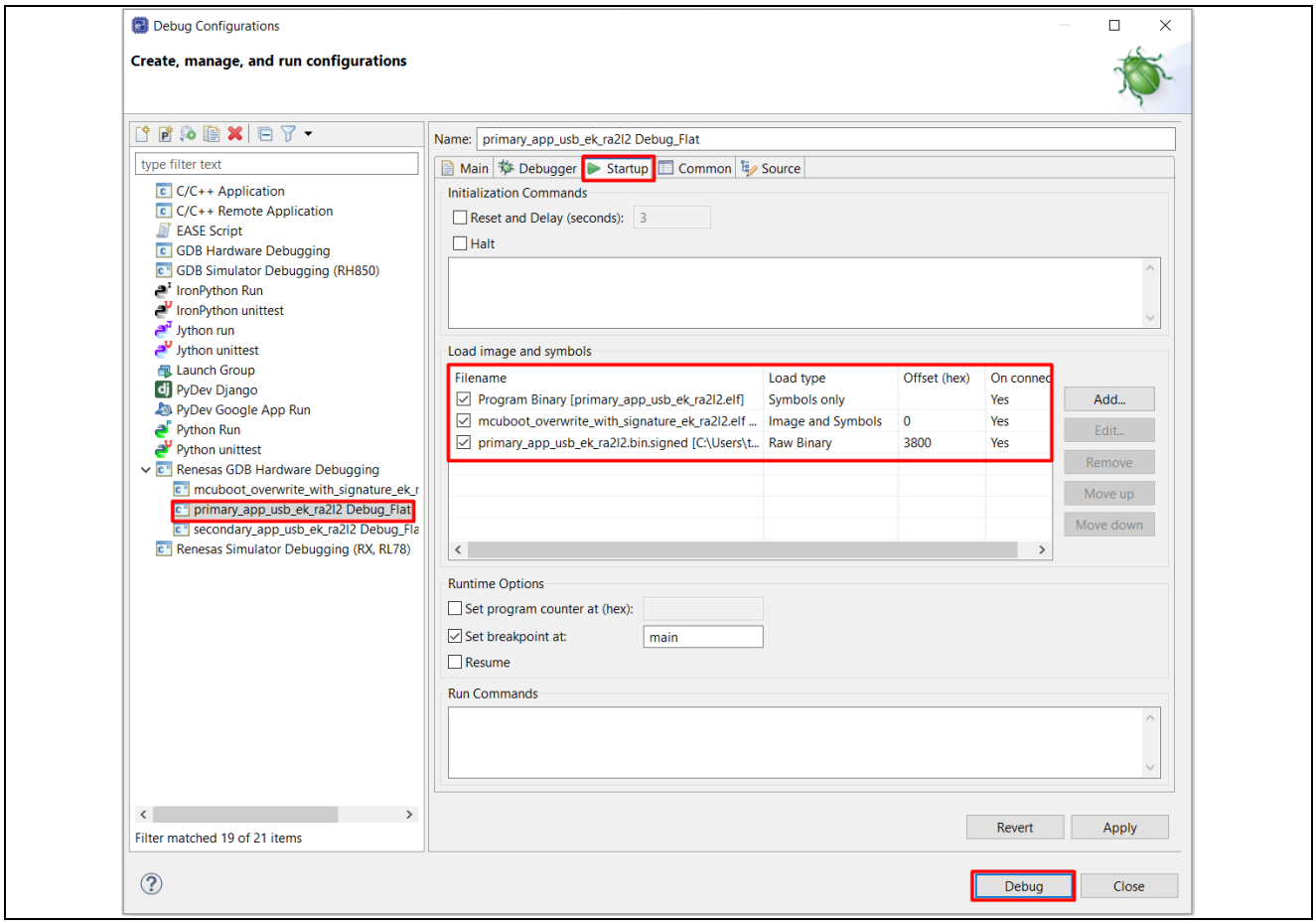


Figure 117. Debug Configurations Results

6. Resume the program execution twice. The Red, Blue, and Green LEDs light up in consecutive order.
7. The target kit will now appear as a **USB Serial Device** in the **Device Manager**. Note the COM port number of the target kit from the **Device Manager**.

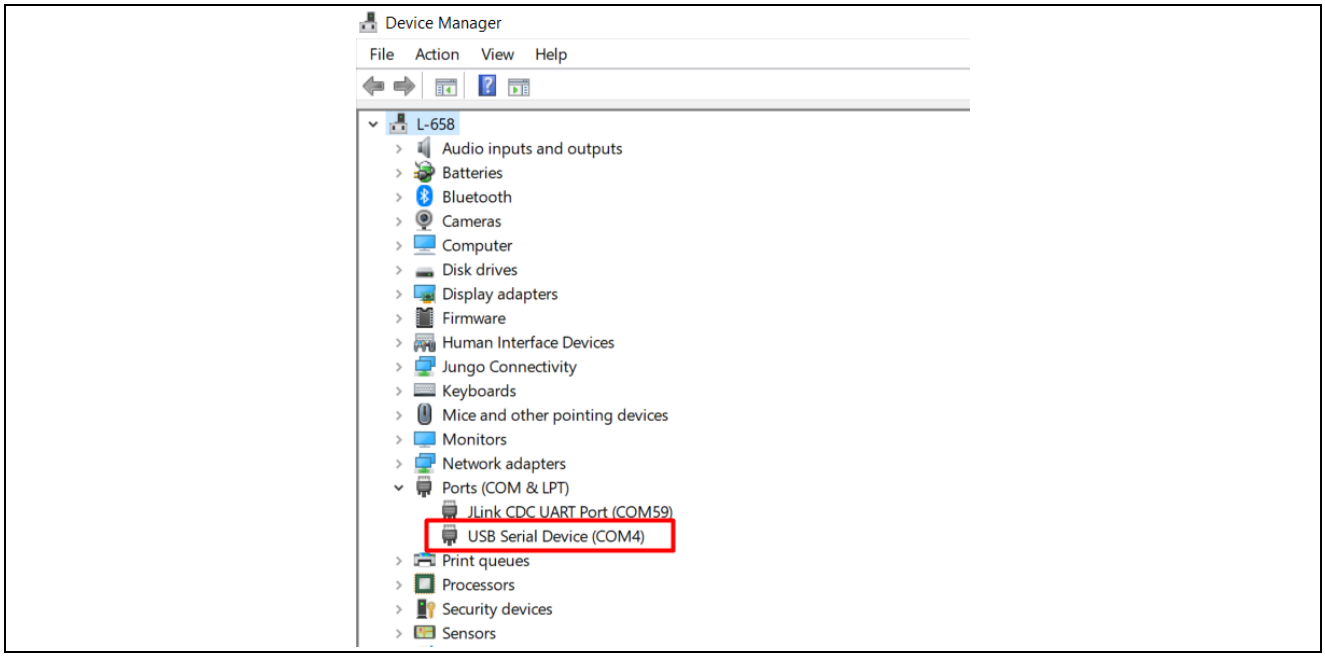


Figure 118. USB Serial Device COM Port for downloader

8. Open Tera Term with COM port (USB Serial Device) (COM number may be different). Then click **OK**.

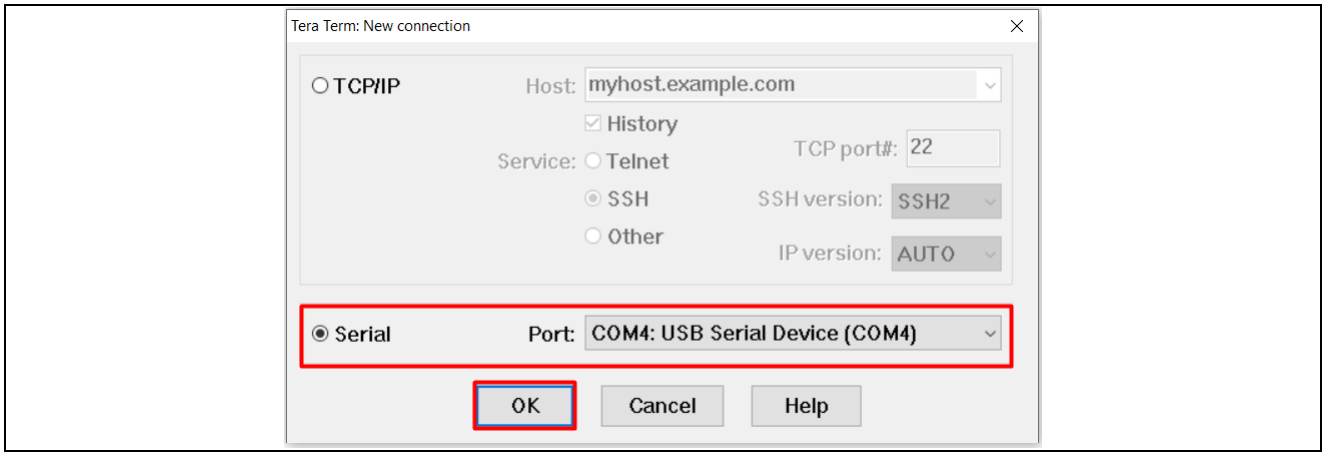


Figure 119. Open the USB COM Port

9. Below message will be printed.

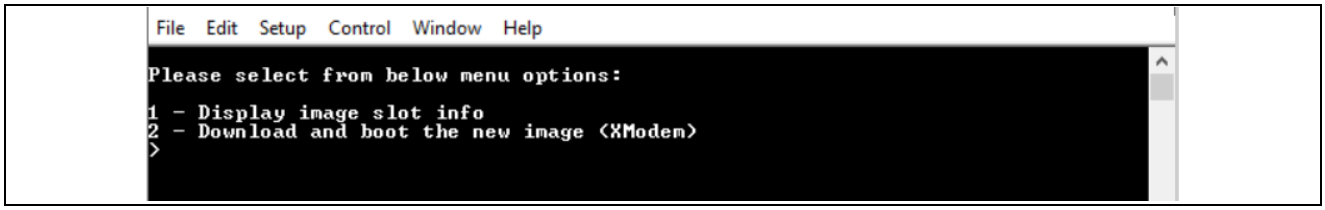


Figure 120. Tera Term Menu

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

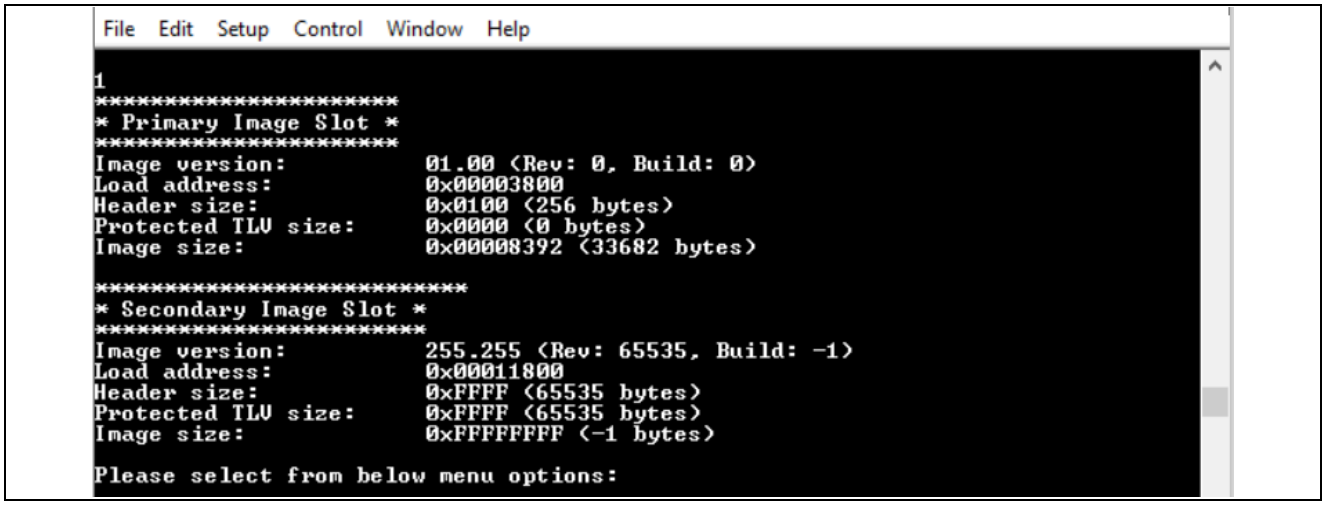


Figure 121. Primary and Secondary Slot Status

11. Use the XMODEM Tera Term to send the secondary_app_usb_ek_ra2l2\Debug\secondary_app_usb_ek_ra2l2.bin.signed to the MCU::

- Choose option **2** from the Tera Term Menu to download the secondary image.

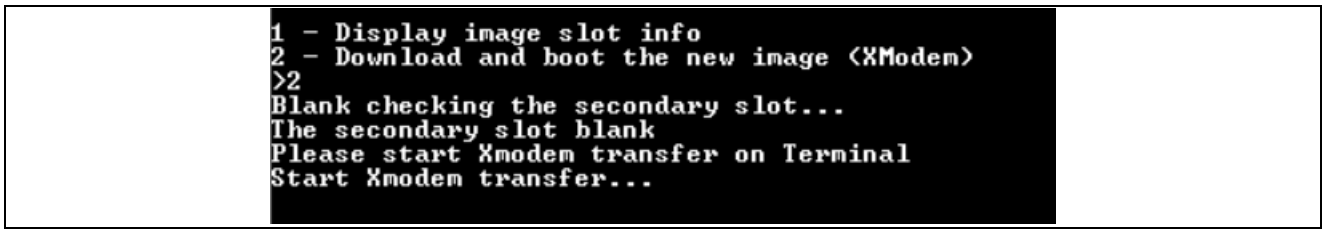


Figure 122. Choose Option 2 to Download the Secondary Image using XMODEM

- Open the **Transfer** interface of the Tera Term by selecting **File** > **Transfer** > **XMODEM** > **Send**

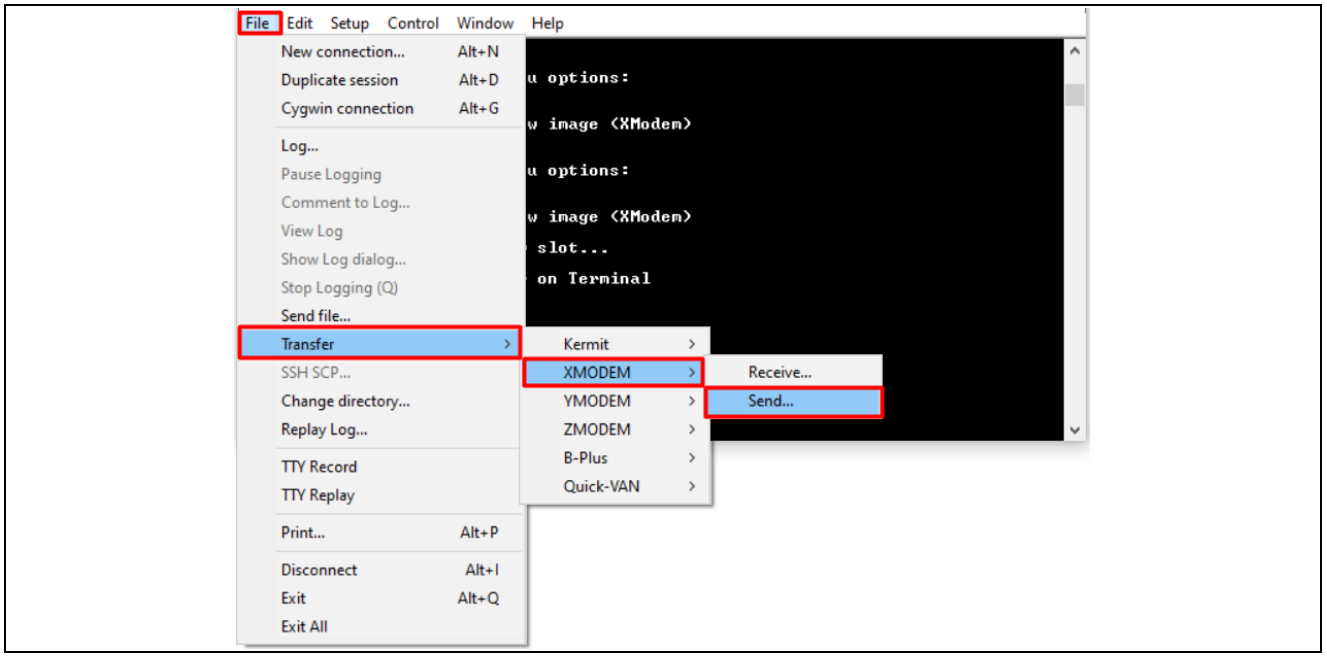


Figure 123. Start Transfer from Tera Term

- Choose secondary_app_usb_ek_ra2l2.bin.signed, then click **Open**

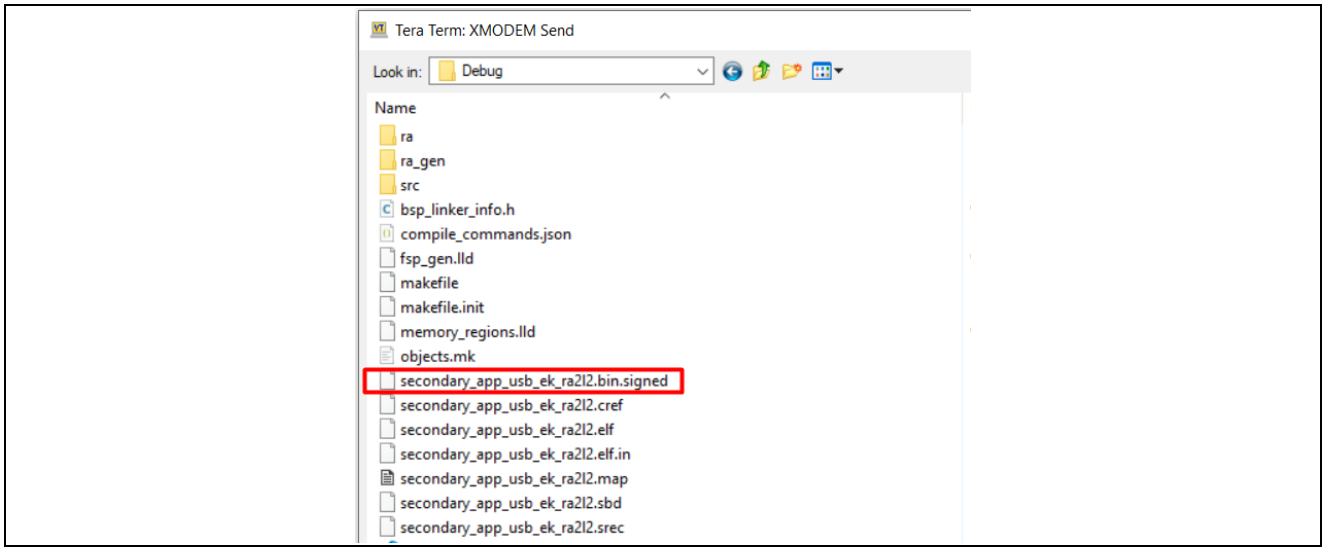


Figure 124. Start Transfer from Tera Term

- The secondary image is downloaded and programmed into the secondary slot. The download process takes approximately four seconds.

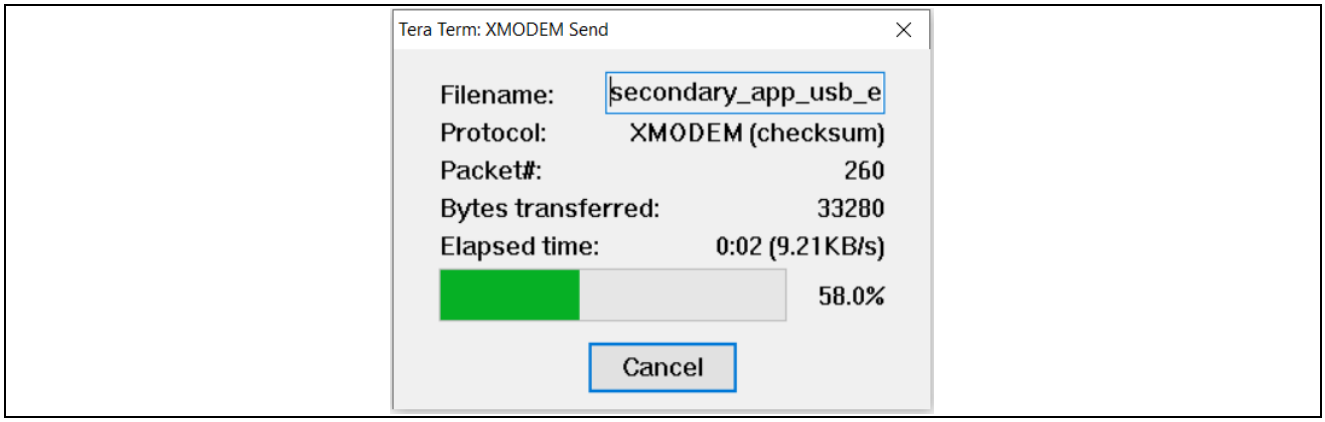


Figure 125. File transferring

12. System will reset automatically after the signed secondary image is successfully downloaded.

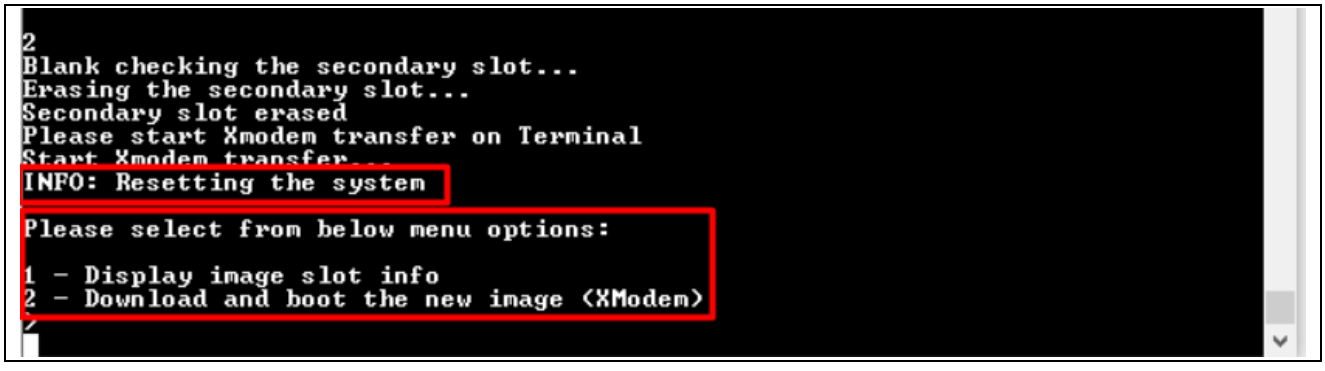


Figure 126. Secondary Image is booted

13. At this point, the Red, Blue, and Green LEDs light up simultaneously.
14. Enter option **1** from the menu to confirm that the version of Primary Image is 1.1.0..

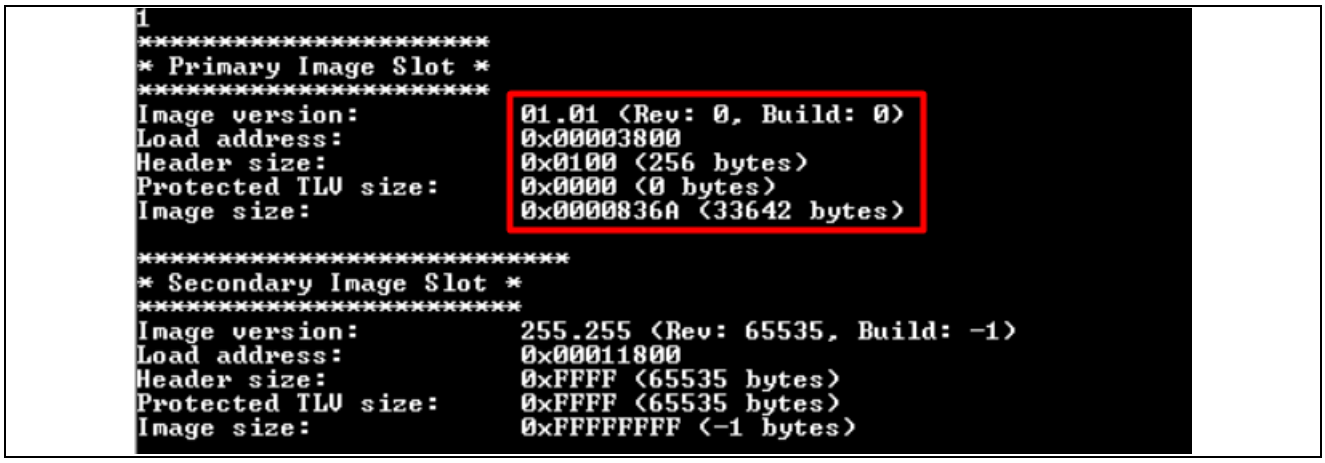


Figure 127. Updated Image Information

9.2 Running the FPB-RA0E1 Overwrite Update Mode sample project

The following procedure describes how to execute the example projects located in the \fpb_ra0e1_secure_bootloader:

1. Import projects to the workspace.
2. Install necessary libraries for MCUboot's scripts. From ra/mcu-tools/MCUboot, open **Command Prompt**. Then run following commands:

   python -m pip install --upgrade pip

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

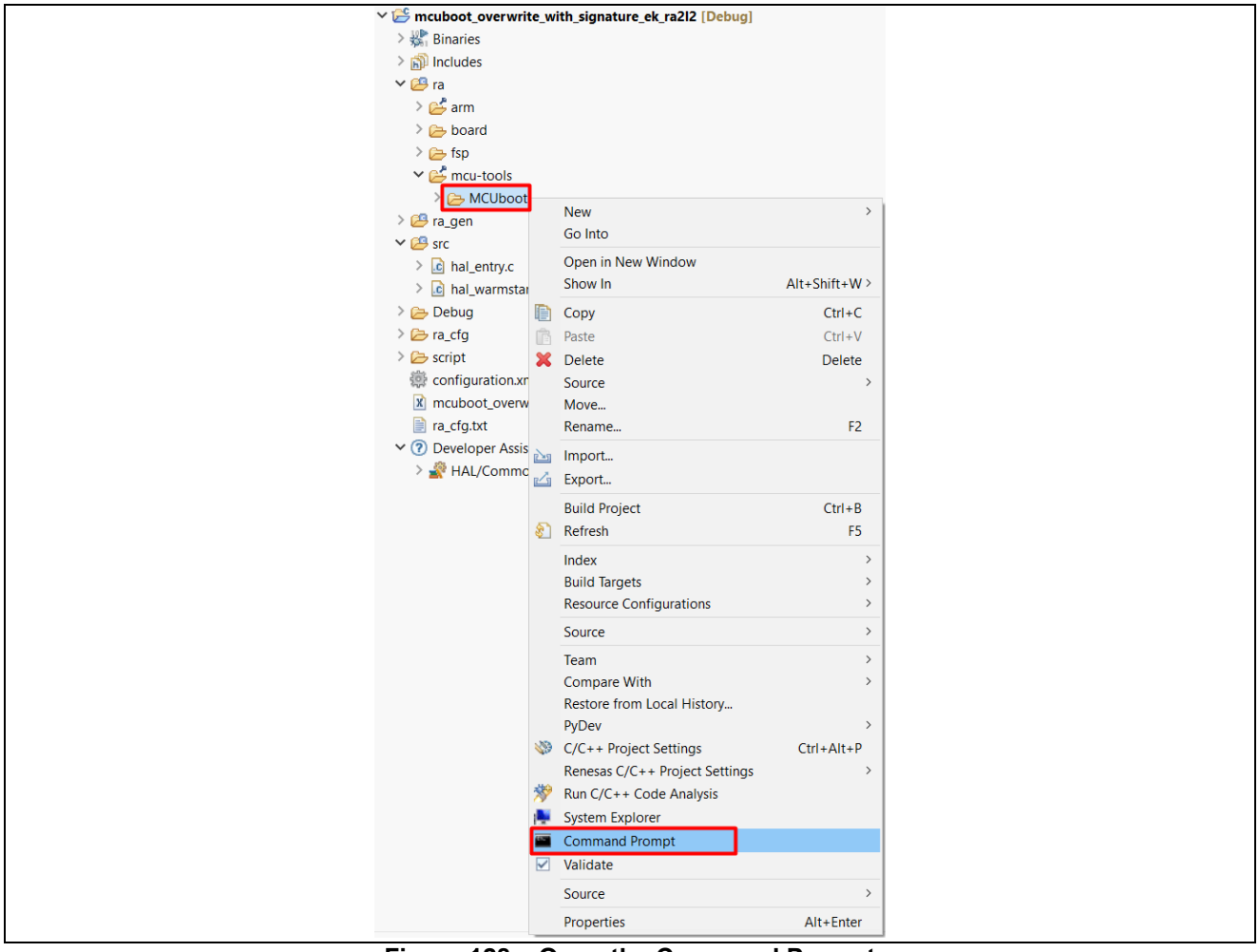


Figure 128. Open the Command Prompt

3. Build the **Solution Project**:

- Right-click on the **Solution Project**: mcuboot_overwrite_solution_fpb_ra0e1.
- Select **Build Project**. This command will build all projects within the **Solution Project**.
- Select **Build Project** for the Secondary Application: secondary_app_fpb_ra0e1.
- Make sure that primary_app_fpb_ra0e1.bin.signed and secondary_app_fpb_ra0e1.bin.signed files are generated in the Debug folder.

4. Erase the entire chip following instructions in section 7.1.
5. Debug the application from project primary_app_fpb_ra0e1.
6. Resume the program execution twice. The two Green LEDs light up in consecutive order.

```
00>
00> Running the Primary Application with Overwrite Update Mode. All two LEDs light up in consecutive order.
```

Figure 129. RTT Viewer Output from the Primary Application

7. Pause the execution.
8. Download the secondary_app_fpb_ra0e1.bin.signed using **Load Ancillary File** to address **0x9800**.

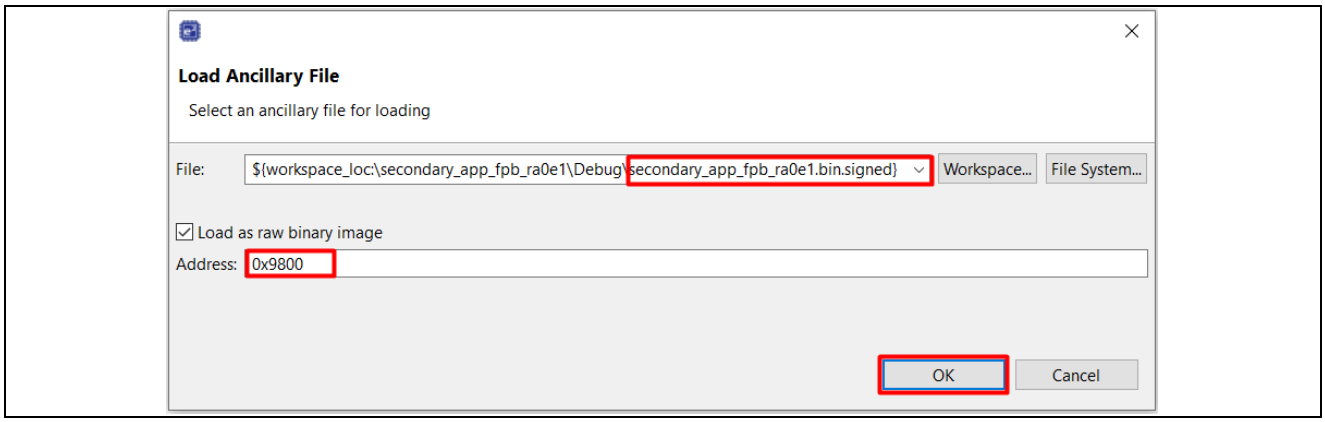


Figure 130. Download the Secondary Application Image in Overwrite Mode

9. Resume the program execution. The two Green LEDs on board light up simultaneously.

```
00>
00> Running the Secondary Application with Overwrite Update Mode. All two LEDs light up simultaneously.
```

Figure 131. RTT Viewer Output from the Secondary Application

10. Known Issues and Limitations

1. The host application is tested only on Windows® 10/11 PC.
2. Issue: Cannot transfer file from host PC. The transferring file is failed as following figure:

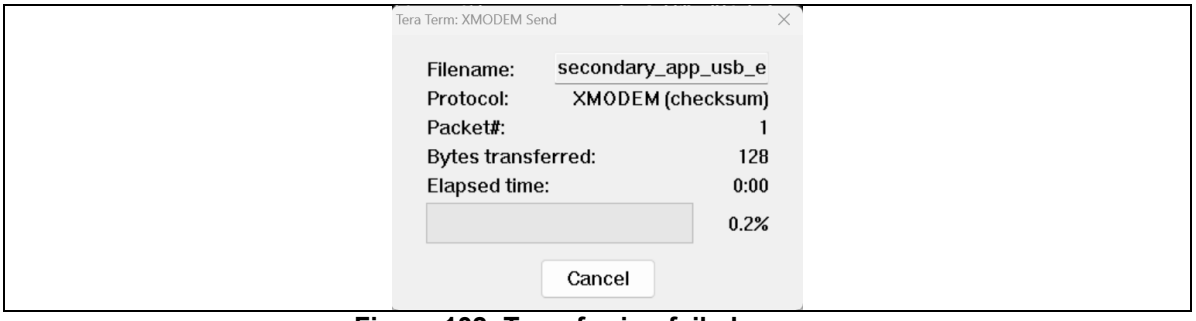


Figure 132: Transferring failed.

Reason: The Host PC did not respond in time when MCU sent 1st NAK to activate for XMODEM transferring.
Solution: Verify the USB cable connection and ensure that XMODEM is being used on the host PC.

3. Issue: Opening primary program's configuration.xml is fail:

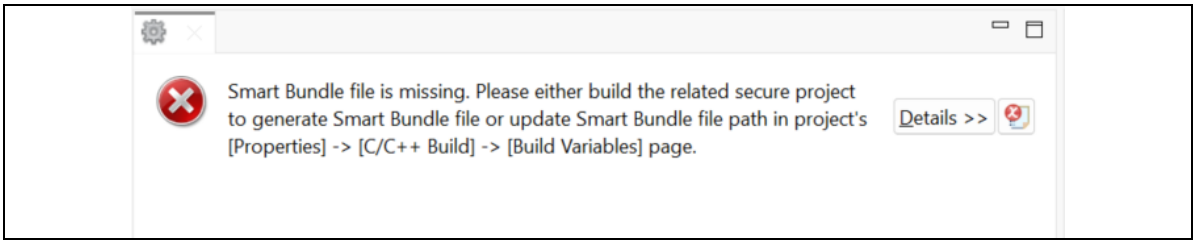


Figure 133. Error in configuration of Primary program

Reason: The MCUboot project (mcuboot_overwrite_with_signature*) has not been built yet.
Solution: Build the MCUboot project, then reopen the configuration of the primary program.

11. Website and Support

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

EK-RA2L2 Resources	renesas.com/ek-ra2l2
FPB-RA0E1 Resources	renesas.com/fpb-ra0e1
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	Jan.07.26	-	Initial version

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.r2. Processing at power-onГThe 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.n3. Input of signal during power-off stateПDo 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.n5. Clock signalsПAfter 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.c6. Voltage application waveform at input pinCWaveform 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.).Г7. Prohibition of access to reserved addressesГAccess 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.t2. 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.p4. 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.r6. 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.D8. 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.o9. 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.t10. 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.s11. 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.n12. 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

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/.

Trademarks

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