

# Renesas RA Family RA8 Basic Secure Bootloader Using MCUboot and Internal Code Flash

# 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 updates. MCUboot is independent of the operating system and hardware and relies on hardware porting layers from the operating system it works with. The Renesas Flexible Software Package (FSP) integrates an MCUboot port starting from FSP v3.0.0. Users can benefit from using the FSP MCUboot Module to create a Root of Trust (RoT) for the system and perform secure booting and fail-safe application updates.

MCUboot is maintained by Linaro on the GitHub MCU-tools page: <u>https://github.com/mcu-tools/mcuboot.</u> There is a \docs folder that holds the documentation for MCUboot in .md file format. This application note refers to the above-mentioned documents wherever possible and is intended to provide additional information that is related to using the MCUboot Module with Renesas RA FSP v3.0.0 or later.

This application note guides you through application project creation using the MCUboot Module on the Renesas EK-RA8M1 kit for internal flash usage using FSP v5.5.0. Example projects for the use case of designing with TrustZone® for multi-image support are provided for EK-RA8M1 internal flash. The MCUboot Module is supported across the entire RA MCU Family. Guidelines on how to adapt the example project configurations for other RA Family MCUs are provided.

# **Required Resources**

#### **Development tools and software**

- The e<sup>2</sup> studio IDE v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- SEGGER J-link<sup>®</sup> USB driver

Note: The above three software components are bundled in a downloadable platform installer available on the FSP webpage at <u>renesas.com/ra/fsp.</u>

• Python v3.9 or later (<u>https://www.python.org/downloads/</u>)

#### Hardware

- EK-RA8M1 Evaluation Kit for RA8M1 MCU Group (<u>http://www.renesas.com/ra/ek-ra8m1</u>)
- Workstation running Windows<sup>®</sup> 10 and Tera Term console or similar application
- One USB device cable (type-A male to micro-B male)

# **Prerequisites and Intended Audience**

This application note assumes that you have some experience with the Renesas e<sup>2</sup> studio IDE and Arm® TrustZone-based development models with e<sup>2</sup> studio. You also need to understand the device lifecycle management of Renesas RA TrustZone-based MCU groups. This knowledge can be acquired by reading the HW User's Manual section "Security Features" and Renesas Application Project R11AN0469. In addition, you should read the entire MCUboot Port section of the FSP User's Manual prior to moving forward with this application project. This application project also assumes that you have some knowledge of cryptography.

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



# Contents

1.	Overview of MCUboot	4
1.1	History of MCUboot	4
1.2	MCUboot Functionalities Overview	4
1.2.1	Validate Application before Booting and Updating	4
1.2.2	Applications Update Strategies	4
2.	Architecting an Application with MCUboot Module using FSP	6
2.1	MCU Memory Configuration using MCUboot Module with FSP	6
2.2	Overview of FSP MCUboot Module	6
2.2.1	General Configuration	7
2.2.2	Application Image Signature Type Options	8
2.2.3	Signing Options	9
2.2.4	MCU Memory Configuration	. 10
2.3	Designing Bootloader and the Initial Primary Application Overview	. 11
2.4	General Guidelines using the MCUboot Module Across RA Family MCUs	. 11
2.5	Customize the Bootloader	. 12
2.6	Production Support	. 12
2.6.1	Key Provisioning	. 12
2.6.2	Make the Bootloader Immutable for Enhanced Security	. 12
263	Advance the Device Lifecycle States Prior to the Deploving the Product to the Field	. 12
2.0.0		
3.	Running the Example Projects	.12
3. 3.1	Running the Example Projects	.12 .13
<ol> <li>3.</li> <li>3.1</li> <li>3.1.1</li> </ol>	Running the Example Projects Set Up the Hardware Set up EK-RA8M1	.12 .13 .13
3. 3.1 3.1.1 3.2	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment	.12 .13 .13 .13
3. 3.1 3.1.1 3.2 3.3	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone	.12 .13 .13 .14 .16
3. 3.1 3.1.1 3.2 3.3 3.3.1	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU	.12 .13 .13 .14 .16 .16
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1 overwrite with bootloader tz	.12 .13 .13 .14 .16 .16 .16
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.2 3.3.3	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects	.12 .13 .13 .14 .16 .16 .16 .16
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects Debug the Applications and Boot the Primary Applications	.12 .13 .13 .14 .16 .16 .16 .16 .17
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects Debug the Applications and Boot the Primary Applications Open the J-Link RTT Viewer	.12 .13 .13 .14 .16 .16 .16 .16 .16 .17 .18
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects Debug the Applications and Boot the Primary Applications Open the J-Link RTT Viewer Downloading and Running the Secondary Applications	.12 .13 .13 .14 .16 .16 .16 .16 .17 .18 .18
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7	Running the Example Projects Set Up the Hardware	.12 .13 .13 .14 .16 .16 .16 .16 .16 .17 .18 .18 .19
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects Debug the Applications and Boot the Primary Applications Open the J-Link RTT Viewer Downloading and Running the Secondary Applications Update the Non-Secure Secondary Image Running the EK-RA8M1 Swap Update Mode Example without Test Mode with TrustZone	.12 .13 .13 .14 .16 .16 .16 .16 .17 .18 .18 .18 .19 .20
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4 3.4.1	Running the Example Projects	.12 .13 .13 .14 .16 .16 .16 .17 .18 .17 .18 .19 .20 .21
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4 3.4.1 3.5	Running the Example Projects Set Up the Hardware Set up EK-RA8M1 Configure the Python Signing Environment Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone Initialize the RA8M1 MCU Import the Projects under \ra8m1_overwrite_with_bootloader_tz Compile All the Projects Debug the Applications and Boot the Primary Applications Open the J-Link RTT Viewer Downloading and Running the Secondary Applications Update the Non-Secure Secondary Image Running the EK-RA8M1 Swap Update Mode Example without Test Mode with TrustZone Downloading and Running the Secondary Applications Downloading and Running the Secondary Applications Running the EK-RA8M1 DXIP Update Mode Example	.12 .13 .13 .14 .16 .16 .16 .16 .16 .17 .18 .18 .19 .20 .21 .22
3. 3.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4 3.4.1 3.5 3.5.1	Running the Example Projects         Set Up the Hardware         Set up EK-RA8M1         Configure the Python Signing Environment         Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone         Initialize the RA8M1 MCU         Import the Projects under \ra8m1_overwrite_with_bootloader_tz         Compile All the Projects         Debug the Applications and Boot the Primary Applications	.12 .13 .14 .16 .16 .16 .16 .17 .18 .19 .20 .21 .22 .23
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4 3.4.1 3.5 3.5.1 3.6	Running the Example Projects.         Set Up the Hardware         Set up EK-RA8M1         Configure the Python Signing Environment         Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone         Initialize the RA8M1 MCU         Import the Projects under \ra8m1_overwrite_with_bootloader_tz         Compile All the Projects         Debug the Applications and Boot the Primary Applications         Open the J-Link RTT Viewer         Downloading and Running the Secondary Applications         Update the Non-Secure Secondary Image         Running the EK-RA8M1 DXIP Update Mode Example without Test Mode with TrustZone         Downloading and Running the Secondary Applications         Running the EK-RA8M1 DXIP Update Mode Example         Downloading and Running the Secondary Applications         Running the EK-RA8M1 DXIP Update Mode Example         Downloading and Running the Secondary Applications         Running the EK-RA8M1 DXIP Update Mode Example         Downloading and Running the Secondary Applications         Running the EK-RA8M1 DXIP Update Mode Example         Downloading and Running the Secondary Applications         Running the EK-R8M1 Overwrite Update Mode Example without TrustZone	.12 .13 .13 .14 .16 .16 .16 .16 .17 .18 .19 .20 .21 .22 .23 .24
3. 3.1 3.1.1 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.3.6 3.3.7 3.4 3.4.1 3.5 3.5.1 3.6 3.6.1	Running the Example Projects	.12 .13 .13 .14 .16 .16 .16 .16 .16 .17 .18 .18 .19 .20 .21 .22 .23 .24 .24



3.6.3 Debug the Applications and Boot the Primary Application	25
3.6.4 Open the J-Link RTT Viewer	
3.6.5 Downloading and Running the Secondary Applications	
3.7 Running the EK-RA8M1 Swap Test Update Mode Example without TrustZone	ə27
3.7.1 Import the Projects	27
3.7.2 Compile the Projects	
3.7.3 Debug the Applications and Boot the Primary Application	
3.7.4 Open the J-Link RTT Viewer	
3.7.5 Downloading and Running the Secondary Applications	
3.8 Troubleshooting	
4. Creating the Bootloader	
4.1 Creating a Bootloader Project for RA Family	
4.1.1 Start Bootloader Project Creation with e <sup>2</sup> studio	
4.1.2 Resolve the Configurator Dependencies	
4.1.3 Setting up the Booting Authentication Support	
4.1.4 Setting up the Application Authentication Signature Type	
4.1.5 Add MCUboot Initialization Code	
5. Using the Bootloader with Applications	41
5.1.1 Import the Standalone Application Projects	
5.1.2 Configure the Application Projects to Use the Bootloader	
5.2 Signing the Existing Application Projects to Use the Bootloader	
5.2.1 Click Generate Project Content and Compile All Four Application Projects	
5.2.2 Configure the Debug Configuration	
5.3 Mastering and Delivering a New Application	
5.4 Customize the Bootloader to other MCUs	
6. Appendix	48
6.1 Making the Bootloader for Cortex-M85 Immutable	
6.2 Making the Bootloader for Cortex-M33 Immutable	
6.3 Making the Bootloader for Cortex-M4 Immutable	
6.4 Device Lifecycle Management for Renesas RA Cortex-M85 MCUs	
6.5 Device Lifecycle Management for Renesas RA Cortex-M33 MCUs	
6.6 Device Lifecycle Management for Renesas RA Cortex-M4 MCUs	
7. References	49
8. Website and Support	49
Revision History	50



# 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 repo was later migrated from JuulLabs to the <u>mcu-tools github project</u>. In 2020, MCUboot was moved under the <u>Linaro Community Project</u> umbrella as an open-source project.

# **1.2 MCUboot Functionalities Overview**

MCUBoot handles the firmware authenticity check after startup and the firmware switch stage of the firmware update process. Downloading the new version of the firmware is out-of-scope for MCUboot. Typically, downloading the new version of the firmware is a functionality that is provided by the application project itself.

### 1.2.1 Validate Application Before Booting and Updating

For applications using MCUboot, the MCU memory is separated into MCUboot, Primary App, Secondary App, and the Scratch Area. Figure 1 is an example of the single-image MCUboot memory map. For more information on the MCUboot memory layout, refer to the <u>Flash Map section</u> of the MCUboot website.



Figure 1. Single Image MCUboot Memory Flash Map

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

The bootloader is started when the CPU is released from reset. For TrustZone-based MCUs, MCUboot is designed to run in secure mode with all access privileges available to it. If there are images in the Secondary App memory marked as to be updated, the bootloader performs the following actions:

- 1. The bootloader authenticates the Secondary App image.
- 2. Upon successful authentication, the bootloader switches to the new image based on the update method selected. Available update methods are introduced in section 1.2.2.
- 3. The bootloader boots the new image.

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

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

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

### 1.2.2 Applications Update Strategies

The following are the update strategies supported by MCUboot. The analysis of pros and cons is based on the MCUboot functionality but not the FSP MCUboot Module functionality. In addition, this application note is not intended to provide all details on the MCUboot application update strategies. We recommend acquiring 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).

- Pros
  - 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.
- Cons
  - Does not support pre-testing of the new image prior to overwriting.
  - Does not support automatic application fallback mechanism.

Overwrite upgrade mode is supported by Renesas RA FSP v3.0.0 or later. External flash memory support is supported by FSP v3.5.0 or later. The overwrite update mode is demonstrated in sections 3.3 and 3.6.

#### • 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 the contents of the Primary slot and the Secondary slot are swapped. The new image then starts from the Primary slot. Upgrading an old image with a new one by swapping can be a two-step process. In this process, MCUboot performs a "test" swap of image data in Flash and boots the new image. The new image can then update the contents of flash at runtime to mark itself "OK", and MCUboot will then still choose to run it during the next boot.

- Pros
  - 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.
  - This image upgrade mode is fail-safe and resistant to power-cut failures.
  - Encrypted image support is available when using external flash.
- Cons
  - 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, thus faster wearing out of Scratch sectors.

Swap upgrade mode is supported by Renesas RA FSP v3.0.0 or later. Runtime image testing is supported by FSP v3.4.0 or later, excluding v3.5.0. External flash memory support is supported by FSP v3.5.0 or later. The swap update mode without the test mode is demonstrated in section 3.4 and the swap update mode with test mode is demonstrated in section 3.7.

#### • Direct execute-in-place (DXIP)

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.

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

Direct execute-in-place mode is enabled in FSP for the code flash linear mode as well as code flash dual bank mode. The DXIP update mode is demonstrated in section 3.5.



Renesas RA Family	RA8 Basic Secure Bootloader Using MCUboot and Internal Code Flash
-------------------	---

#### • RAM loading firmware update

Like the direct-XIP mode, RAM loading firmware update mode selects the newest image by reading the image version numbers in the image headers. However, instead of executing it in place, the newest image is copied to RAM for execution. The load address (the location in RAM where the image is copied to) is stored in the image header. This upgrade method is not typically used in an MCU environment. Refer to the <u>RAM Loading section</u> on the MCUboot page for more information on this update strategy. This image update mode does not support encrypted images (see MCUboot documentation on encrypted image operation).

RAM loading update mode is not supported by the Renesas RA FSP.

# 2. Architecting an Application with MCUboot Module using FSP

This section provides an overview of the FSP MCUboot Module, which integrates MCUboot as a module into the FSP. The available upgrade modes and memory architecture design are discussed. In addition, signing and mastering new images are discussed.

# 2.1 MCU Memory Configuration using MCUboot Module with FSP

For single-image projects, refer to Figure 1 from section 1.2.1 to see the default memory map layout. For applications with two separately updateable images, such as TrustZone<sup>®</sup> applications where the Secure and Non-Secure images can be updated separately, the default memory map layout is shown in Figure 2.



Figure 2. Two-Image MCUboot Module Memory Map (TrustZone)

Note that for RA8 MCUs, bit 28 of the address space indicates whether this region is defined as a secure or non-secure region. If bit 28 is 0, the corresponding region is a secure region. If bit 28 is 1, the corresponding region is non-secure. This is reflected in Figure 14 and Figure 23.

### 2.2 Overview of FSP MCUboot Module

This section provides a high-level overview of the MCUboot Module in the FSP. Currently, the FSP supports four firmware update methods:

- **Overwrite Only**: The entire Primary slot is overwritten with the Secondary slot.
- **Overwrite Only Fast**: Only sizeof(secondary\_image) is copied into Primary slot. Unused sectors are not copied.
- **Swap**: The entire Primary and Secondary slots are swapped. A Scratch region is required.
- **Direct XIP**: The new image is run directly from its flash partition.

We recommend reviewing MCUboot port section of the FSP User's Manual to understand the Build Time Configurations for MCUboot. This section is not meant to cover all the configurable properties. Only some of the most frequently used configuration options are introduced.



#### 2.2.1 General Configuration

Stacks	Configuration			Generate Project Cor	ntent
Threads	🚯 New Thread 🚯 Remove 📄	HAL/Common Stacks	🛃 Ne	w Stack >   🚢 Extend Stack > 🔬 Rem	ove
רא <u>א</u> יי ליי ליי	AL/Common g_ioport I/O Port (r_ioport) MCUboot	<ul> <li>MCUboot</li> <li>MCUboot</li> </ul>			Â
Objects	🔂 New Object > 🔞 Remove	<	<b></b>	\$	<b>•</b> ~
Summary	BSP Clocks Pins Interrupts Event Links	Stacks Components			
🛐 Probler	ns 📮 Console 🔲 Properties 🗙 🌸 S	Smart Browser 🚇 Smart Manual			
MCUboo	ot				
Settings	Property	Value			
API Info	V Common				
	Custom mcuboot_config.h				
	Upgrade Mode	Overwrite	Only		
	Validate Primary Image	Enabled			
	Downgrade Prevention (Over	write Only) Disabled			
	Number of Images Per Applic	ation 2 (TrustZ	ne)		

Figure 3. FSP MCUboot Module General Configuration Properties

General configuration properties include:

- **Custom mcuboot\_config.h**: The default mcuboot\_config.h file contains the MCUboot Module configuration that you selected from the RA configurator. You can create a custom version of this file to achieve additional bootloader functionalities that are available in MCUboot.
- **Upgrade Mode**: This property configures the application image update method selection explained at the beginning of section 2.2. The options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP, as shown in Figure 4. Overwrite Only is the default setting.

	Configuration				Generate Project Conte
Threads	😧 New Thread 🔹 Remove 📄	HAL/Common Sta	cks	된 New Stack >  🚢	: Extend Stack >  👔 Remov
ער אין	AL/Common g_ioport I/O Port (r_ioport) MCUboot	<ul><li>MCUboot</li><li>MCUboot</li></ul>			
Objects	🚯 New Object > 🔹 🔝 Remove				
		<			>
ummone (	DCD Clocks Ding Interrupts Event Links	Stacks Components			
summary   t	BSP Clocks Pins Interrupts Event Links	Stacks Components			
Problem	is 📮 Console 🔲 Properties 🗙 虆 S	Smart Browser 🛄 Sm	art Manual		
Problem	ns 📮 Console 🔲 Properties 🗙 🋶 S	Smart Browser 🛄 Sm	art Manual		
Problem	is 📮 Console 🔲 Properties 🗙 髞 S t	Smart Browser 🐺 Sm	art Manual		
Problem	IS Console Properties X S t Property	imart Browser 🔑 Sm	art Manual Value		
Problem	IS Console Properties X S t Property Common	imart Browser 😱 Sm	art Manual Value		
Problem MCUboot Settings API Info	Is Console Properties X S t Property Common General	imart Browser 🛄 Sm	art Manual Value		
Problem MCUboot Settings API Info	s Console Properties X S t Property Common General Custom mcuboot_config.h	imart Browser 😱 Sm	art Manual Value		
Problem MCUboot Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode	imart Browser 😱 Sm	Art Manual Value Overwrite Only	_	
Problem MCUboot Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode Validate Primary Image	imart Browser 😱 Sm	Value Overwrite Only Swap		
Problem MCUboot Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode Validate Primary Image Downgrade Prevention (Overv	imart Browser 😱 Sm. vrite Only)	Art Manual Value Overwrite Only Swap Overwrite Only		
Problem MCUboot Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode Validate Primary Image Downgrade Prevention (Overv Number of Images Per Applici	imart Browser 😱 Sm write Only) ation	Art Manual Value Overwrite Only Swap Overwrite Only Overwrite Only Fast		
Problem MCUboon Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode Validate Primary Image Downgrade Prevention (Overv Number of Images Per Application Watchdog Feed	imart Browser 😱 Sm write Only) ation	Value Value Overwrite Only Swap Overwrite Only Overwrite Only Fast Direct XIP		
Problem MCUboov Settings API Info	t Property Common General Custom mcuboot_config.h Upgrade Mode Validate Primary Image Downgrade Prevention (Overv Number of Images Per Applica Watchdog Feed Measured Boot	imart Browser 😱 Sm write Only) ation	Art Manual Value Overwrite Only Swap Overwrite Only Overwrite Only Fast Direct XIP Disabled		

Figure 4. Application Image Update Mode



Figure 5 is a more detailed application image format that can be referenced to understand the various MCUboot property definitions.



Figure 5. General Configuration for MCUboot Module

#### Validate Primary Image:

When Validate Primary Image is enabled, the bootloader performs a hash or signature verification, depending on the verification method chosen, in addition to the MCUboot sanity check based on the image header and TLV area magic numbers. The Header and TLV area magic numbers are always checked as part of the sanity checking prior to the integrity checking and the signature verification. When the Validate Primary Image is disabled, the integrity check based on the hash is performed, and the sanity check is performed as well. It is highly recommended that this property is always enabled if boot time is not a concern. Note that the image magic number is not part of the image validation; it is a reference value that can be used for sanity checks during the application upgrade debugging process. This image magic number is written to the flash after a successful image upgrade.

- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is enabled, new firmware with a lower version number will not overwrite the existing application.
- **Number of Images Per Application:** This property allows you to choose one image for Non-TrustZone<sup>®</sup>-based applications and two images for TrustZone-based applications.

### 2.2.2 Application Image Signature Type Options

Application images using MCUboot must also be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.

Figure 6 shows the signature types available for the application image signing methods supported by the MCUboot module. For memory-restricted devices, you can choose **None** for **Signature Type**, which will reduce the bootloader size. For example, the bootloader for the Overwrite update mode uses a flash area of 64 KB when using the ECDSA P-256 signature type, but when signature support is not used, the bootloader reduces to about 19 KB.



Image: Settings       Property       Value         Problems       Console       Properties         Settings       Property       Value         API Info       > General         Image: Signature Type       ECDSA P-256         Boot Record       None
Summary       BSP       Clocks       Pins       Interrupts       Event Links       Stacks       Components         Problems       Console       Properties       Image: Smart Browser       Smart Manual         MCUboot       Settings       Property       Value         Y       Common       Sental       Sental         Image: Signing and Encryption Options       TrustZone       ECDSA P-256         Boot Record       None       Context Part
Problems Console     Property     Value
MCUboot         Settings       Property       Value         API Info <ul> <li>General</li> <li>Signing and Encryption Options</li> <li>TrustZone</li> </ul> <ul> <li>ECDSA P-256</li> <li>Boot Record</li> </ul> Boot Record
Settings     Property     Value       API Info     > Common     >       > Signing and Encryption Options     >     >       > TrustZone
API Info
<ul> <li>Signing and Encryption Options</li> <li>TrustZone</li> <li>Signature Type</li> <li>Boot Record</li> <li>Control of the second</li> </ul>
TrustZone      Signature Type     ECDSA P-256 Boot Record None CODE A P 256
Signature Type ECDSA P-256 Boot Record None Control Co
Boot Record None
Custom BSA 2048
Francisco Schome RSA 3072
> Elach Lavout
> Data Sharing

Figure 6. Application Image Signature Type for FSP MCUboot Module

### 2.2.3 Signing Options

Figure 7 shows the default Custom signing option configuration provided by FSP.

<ul> <li>Signing and Encryption Options</li> </ul>	
✓ TrustZone	
Boot Record (Image 2)	
Custom (Image 2)	confirm
Signature Type	ECDSA P-256
Boot Record	
Custom	confirm
Python	python
Encryption Scheme	Encryption Disabled

#### Figure 7. FSP Default Signing Option

By default, FSP sets --confirm for the **Custom** property for both Image 1 and Image 2 when TrustZone is used. For TrustZone-based applications, the Secure Image (Image 1) and Non-Secure Image (Image 2) can have different configurations such that there is a different update policy for the Secure and Non-Secure Images. Some commonly used signing options are:

• Option --pad:

This option places a trailer on the image that indicates that the image should be considered for an upgrade. Writing this image in the Secondary slot causes the bootloader to upgrade to it. When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application image. On the next reset, the Primary application previously used will be swapped back and rebooted.

• Option --confirm:

When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application. At the next reset, there will be no swap between the Primary and Secondary application and the Secondary application will be booted. Confirm is the default Force Upgrade configuration.

No input:

If no option is put in this property, application images signed with the signing command generated from this setting will not be updated.



When Overwrite mode is selected, the **--pad** or **--confirm** option generates signing commands such that the overwrite will occur, and the Secondary application will overwrite the Primary application.

The image signing tool imgtool.py is included with MCUboot. It is integrated as a post-build tool in  $e^2$  studio to sign the application image. For detailed information about using this tool with  $e^2$  studio, refer to the application image signing information in section 5.2. For more information on the possible options available for this property setting, refer to the description in the <u>imagetool.py md file</u> and visit the MCUboot documentation page <u>https://docs.mcuboot.com/imgtool.html</u>.

### 2.2.4 MCU Memory Configuration

Figure 8 shows the default memory configuration options provided by the FSP configurator for RA8 MCU groups.

lash Layout	
_ TrustZone	
Non-Secure Callable Region Size (Bytes)	0x0
Non-Secure Flash Area Size (Bytes) (TrustZone Non-Secure)	0x0
Non-Secure Callable RAM Region Size (Bytes)	0x0
Non-Secure RAM Region Size (Bytes) (TrustZone Non-Secure)	0x0
Image 2 Header Size (Bytes)	0x200
Bootloader Flash Area Size (Bytes)	0x20000
Image 1 Header Size (Bytes)	0x200
Image 1 Flash Area Size (Bytes)	0x20000
Scratch Flash Area Size (Bytes)	0x0

Figure 8. MCU Memory Configuration Default Settings

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

- Bootloader Flash Area: Size of the flash area allocated for the bootloader.
- **Image 1 Header Size**: Size of the flash area allocated for the application header for single image configuration or the secure application image header size in the case of a TrustZone<sup>®</sup>-based application. This property should be set to 0x200 for RA8, RA6 and RA4 MCUs and 0x100 for RA2 MCUs.
- **Image 1 Flash Area Size**: Size of the flash area allocated for the application image for single image configuration or the secure application image in the case of a TrustZone-based application.
- Scratch Flash Area Size: This property is only needed for Swap mode. The Scratch area must be large enough to store the largest sector that is going to be swapped. For RA8M1, the Scratch area is set up to be 32KB (0x8000).

The properties under **TrustZone** are for TrustZone-based applications:

• Non-Secure Callable Region Size (Bytes): This area is used for the TrustZone Non-Secure Callable area plus the MCUboot trailer. This property needs to be set to a multiple of 1024 bytes. Each Non-Secure Callable function takes 8 bytes of flash area. The non-secure Callable function usage can be identified by referring to the section .sgstub in the secure application map file. For Swap mode, the MCUboot trailer size is calculated as 128\*(5+(3\*BOOT\_MAX\_IMG\_SECTORS)).

BOOT\_MAX\_IMG\_SECTORS is the number of flash sectors in either the secure or the non-secure image, whichever is larger.

For Overwrite mode, the image trailer is less than 256 bytes; for a typical application with a limited number of Non-Secure Callable APIs, it is recommended to set the Non-Secure Callable Region Size to 0x400.

- Non-Secure Flash Area Size: Size of the Non-Secure Flash region. You can compile the non-secure application to get the size of the image and set this value accordingly. This value must be a multiple of the flash block size.
- Non-Secure Callable RAM Region: This property is the size of the Non-Secure Callable RAM region of the Secure image. This property needs to be set to a multiple of 1024 bytes.
- Non-Secure RAM Region Size: Size of the Non-Secure RAM region. This property must be an integer multiple of 8192 bytes.
- **Image 2 Header Size**: The non-secure application header size. This property should be set up by following the same rule as explained for the **Image 1 Header Size**.



# 2.3 Designing Bootloader and the Initial Primary Application Overview

A bootloader is typically designed with the initial Primary application. The following are the general guidelines for designing the bootloader and the initial Primary application:

- Develop the bootloader and analyze the MCU memory resource allocation needed for the bootloader and the application. The bootloader memory usage is influenced by the application image update mode, signature type, and whether to validate the Primary Image. The bootloader maintains a memory map of all the different images shown in Figure 1 and Figure 2.
- Develop the initial Primary application, perform the memory usage analysis, and compare with the bootloader memory allocation for consistency and adjust as needed.
- Determine the bootloader configurations in terms of image authentication and new image update mode. This may result in the adjustment of the memory allocated definition in the bootloader project.
- Test the bootloader and the initial Primary application.

Most of these design aspects are addressed in the walk-through in section 4.

# 2.4 General Guidelines using the MCUboot Module Across RA Family MCUs

The MCUboot Module is supported by all RA Family MCUs. The cryptographic support is provided via MbedTLS Crypto only module and Tiny Crypt module. Both crypto modules are supported on all RA MCUs either through software or MCU hardware. The MbedTLS Crypto Only module is supported by the MCU hardware if the corresponding algorithms are supported by the hardware crypto engine, otherwise MbedTLS software stack will be used. The MbedTLS offers more crypto algorithms, is generally faster, and has a larger memory footprint. On the other hand, the TinyCrypt module offers a smaller number of algorithms and is slower but has a much smaller memory footprint. TinyCrypt does not support the RSA algorithm.

For both algorithms, the image validation of the primary image prior to execution at MCU reset can be disabled to reduce the boot time. See explanations on the validation property in section 2.2.1.

Table 1 is the typical cryptographic selection recommendation when using MCUboot with RA MCUs. If memory footprint is a priority, users can choose the TinyCrypt module over the MbedTLS Crypto Only module for some of these use cases. To improve the verification speed and reduce boot time when using TinyCrypt, users can consider disabling image validation to improve verification and boot time performance.

Crypto Stack	RA2 No Encryption	RA2 with Encryption	RA4W1, RA4M1, RA6T2 No Encryption	RA4W1, RA4M1, RA6T2 with Encryption	RA4E1, RA6E1, RA6M1/M2/M3, RA6T1, RA4M2/M3, RA6M4/M5 with or without Encryption	RA8M1, RA8D1, RA8T1, with or without Encryption
MbedTLS (Crypto Only) HW					x	х
TinyCrypt (HW AES)		x		x		
TinyCrypt (SW Only)	х		x			

 Table 1. Typical Cryptographic Selection Recommendations for RA MCUs

For the Renesas RA Cortex<sup>®</sup>-M85 MCU series internal flash usage, refer to the RA8M1 example projects demonstrated in this application project.

For the Renesas RA Cortex<sup>®</sup>-M33 MCU and RA Cortex<sup>®</sup>-M4 MCUs RA6 MCU series internal flash usage, refer to the RA6M4 and RA6M3 example projects demonstrated in the RA6 Basic Secure Bootloader Using MCUboot and Internal Code Flash application project (R11AN0497).

For the Renesas RA Cortex<sup>®</sup>-M23 MCU series, refer to the RA2E1 example projects demonstrated in the Secure Bootloader for the RA2 MCUs application project (R11AN0516).

For RA6 MCUboot with encryption support, refer to the RA6M4 example projects demonstrated in the Booting Encrypted Image using MCUboot and QSPI application project (R11AN0567).



#### 2.5 Customize the Bootloader

The following aspects need to be considered when customizing the bootloader in a product design:

- Customize the image validation method.
- Customized method to download the application.
- Use various optimization methods to reduce bootloader and application image size. For example, compile the bootloader by Optimize size.

# 2.6 Production Support

#### 2.6.1 Key Provisioning

By default, the public key is embedded in the bootloader code, and its hash is added to the image manifest as a KEYHASH TLV entry. See section 4.1.3 for more details about the public key and private key that are used for testing purposes. For production support, follow the example shown in key.c to add the public key. In addition, you must update the private key for application image signing. Refer to Figure 65 and Figure 66 for the private key selection in the signing command.

As an alternative, the bootloader can be made independent of the included test keys by setting the MCUBOOT\_HW\_KEY option. In this case, the hash of the public key must be provided to the target device, and MCUboot must be able to retrieve the key-hash from there. For this reason, the target must provide a definition for the boot\_retrieve\_public\_key\_hash() function that is declared in

boot/bootutil/include/bootutil/sign\_key.h. The full option for the -public-key-format imgtool argument is also required to add the whole public key (PUBKEY TLV) to the image manifest instead of its hash (KEYHASH TLV).

During boot, the public key is validated before it is used for signature verification. MCUboot calculates the hash of the public key from the TLV area and compares it with the key hash that was retrieved from the device. This way, MCUboot is independent from the public key(s). The key(s) can be provisioned at any time and by different parties.

#### 2.6.2 Make the Bootloader Immutable for Enhanced Security

For a Cortex<sup>®</sup>-M85 MCU, refer to section 6.1 to make the bootloader immutable. For a Cortex<sup>®</sup>-M33 MCU, refer to section 6.2 to make the bootloader immutable. For a Cortex<sup>®</sup>-M4 MCU, refer to section 6.3 to make the bootloader immutable.

#### 2.6.3 Advance the Device Lifecycle States Prior to Deploying the Product to the Field

For a Cortex<sup>®</sup>-M85 MCU, refer to section 6.4 for the device lifecycle management of the MCU. For a Cortex<sup>®</sup>-M33 MCU, refer to section 6.4 for the device lifecycle management of the MCU. For a Cortex<sup>®</sup>-M4 MCU, refer to section 6.6 for the device lifecycle management of the MCU.

#### 3. Running the Example Projects

This section provides a walk-through of running the included example projects. To recreate the bootloader example projects demonstrated in this section, refer to section 4.1 for the Cortex<sup>®</sup>-M85 implementation.

The bootloader projects introduced have similar functionality, except that the memory map definition and application image update mode are different.

Unzip example\_projects\_with\_bootloader.zip, and you will see that there are five folders. Each folder contains example projects for the specific MCU, which include bootloader projects and example application projects.

example_projects_with_bootloader >	
Name	
ra8m1_overwrite_with_bootloader       non-trustzone examples         ra8m1_swap_test_with_bootloader_flat       non-trustzone examples         ra8m1_overwrite_with_bootloader_flat       trustzone examples         ra8m1_overwrite_with_bootloader_tz       trustzone examples	

Figure 9. Example Projects with Bootloader Support



Set up the Python development environment by following section 3.3 step 3.2. Note that this step only needs to be performed once.

# 3.1 Set Up the Hardware

#### 3.1.1 Set up EK-RA8M1

- Jumper setting: J12 is set to pins 2-3 and J15 is closed.
- Connect J10 using a USB micro to B cable from EK-RA8M1 to the development PC to provide power and debug connection using the onboard debugger.

Once the EK-RA8M1 is powered up, initialize the MCU prior to exercising the bootloader project.

Erase the entire MCU flash and ensure the MCU is in a Secure Software Development Device Lifecycle State. This can be achieved using the Renesas Device Partition Manager.

1. Power cycle the board, launch e<sup>2</sup> studio, and open the Renesas Device Partition Manager.

Run	Renesas Al Window Help				
	Renesas Debug Tools	>		Renesas Device Partition Manager	
Q,	Run	Ctrl+F11	1	TraceX	>
核	Debug	F11	Ð	Tracealyzer	>

Figure 10. Open Renesas Device Partition Manager

#### 2. Select Read current device information.

If the DLM state is OEM (PL2, PL1 or PL0), proceed to step 3. Otherwise, you must switch to a different kit to continue the rest of the operation. Below is an example of the readout from an RA8M1 MCU that is in the OEM state.

Reading the current Protection Level (PL) and Authentication Level (AL) of the deviceSUCCESSFUL!	^	
Reading the current secure/non-secure partition size of the deviceSUCCESSFUL!		
Current status of the device		
DLM state : Original Equipment Manufacturer (OEM)		-
Protection level (PL) : PL2		-
Authentication level (AL) : AL2		
Secure/NSC memory partition size :		
- Code Flash Secure (kB) : 32		
- Data Flash Secure (kB) : 0		
END of current status of the device.		
		-
Disconnecting		-
DISCONNECTED.		
SUMMARY OF RESULT		
Connection : SUCCESSFUL!		
Status display : SUCCESSFUL!		
END SUMMARY		-
		-
	~	-
< >	~	-
Show Command Line Run		
	se	
		1

Figure 11. Read the Device Lifecycle States



3. Select **Initialize device back to factory default**, choose **J-Link** as the connection method, and click **Run**.

Renesas Device Partition I	Manager					×
① Enter a value for Action a	ind Emulator typ	be				
Device Family: Renesas RA	~					^
Action						
Read current device inform	mation	Change	debug state			
Set TrustZone secure / no	n-secure bound	laries 🗹 Initialize	device back to	o factory o	default	
Target MCU connection:		J-Link	~			
Connection Type:		SCI	~			
Emulator Connection:		Serial No	~			
Serial No/IP Address:						
Debugger supply voltage (V)	:	0	~			
Connection Speed (bps for S	Cl, Hz for SWD)	9600	~			
Debug state to change to:		Secure Software	Development	· ·		
Memory partition sizes						- 11
Use Renesas Partition Dat	a file					
					Browse	
Code Flack Course (KD)	60					
Code Hash Secure (KB)	60					
Code Flash NSC (KB)	4					
Data Flash Secure (KB)	0					
SRAM Secure (KB)	10					
SRAM NSC (KB)	6					
Command line tool:						~
?	Import	Export	Run	ı	Close	

Figure 12. Initialize RA8M1 using Renesas Device Partition Manager

The entire flash will be erased if sections are not permanently locked down. In addition, if the device is in the PL1 or PL0 state, the RA8M1 will be initialized to the PL2 state.

### 3.2 Configure the Python Signing Environment

If this is **NOT** the first time you have used the Python script signing tool on your computer, you can skip this section. Note that section 3.3 to section 3.7 can be evaluated independently; it is not necessary to follow a particular sequence.

Download and Install Python v3.9 or later from https://www.python.org/downloads/.

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:

- From the included example project sets (refer to Figure 9), choose the set of projects you would like to do first.
- Import that set of projects into a workspace. In this example, we assume you have chosen to import the
  projects under the folder:
   Automatic and a project into a workspace of the set loader backspace.
  - \example\_projects\_with\_bootloader\ra8m1\_overwrite\_with\_bootloader\_tz.
- Navigate to folder \MCUboot in the bootloader project included, for example, ra\_mcuboot\_ra8ml>ra>mcu-tools>MCUboot, right click, and select Command Prompt.
   Depending on your PC policy, administrator privileges may be required when running the Command Prompt. This opens a command window with the path set to the \mcu-tools\MCUboot folder.





Figure 13. Open the Command Prompt

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

python -m pip install --upgrade pip

- Note that if you have multiple Python versions installed, make sure to check that the Python version is version 3.9.0 or later.
- Next, in the command window, enter the following command line to install all the MCUboot dependencies:

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

This will verify and install any required dependencies. Make sure this step runs successfully prior to moving to the following sections. If your project path contains special characters or spaces, an error may occur when executing the Python script.



# 3.3 Running the EK-RA8M1 Overwrite Update Mode Example with TrustZone

Follow the steps below to run the example projects for EK-RA8M1 using the MCUboot Module Overwrite Only Update mode with TrustZone.

#### 3.3.1 Initialize the RA8M1 MCU

Follow section 3.1.1 to initialize the RA8M1 MCU.

#### 3.3.2 Import the Projects under \ra8m1 overwrite with bootloader tz

New users should refer to the FSP User's Manual section on Importing Projects into the IDE for guidelines. Ensure the Python signing environment is set up referencing section 3.2.



#### Figure 14. Example Projects for RA8M1 Overwrite Update Mode

- **ra mcuboot ra8m1**: The bootloader project is configured with Overwrite update mode.
- app\_ra8m1\_s\_primary: The Primary Secure application project with FSP flash driver support with the flash driver configured as Non-Secure Callable.
- app\_ra8m1\_ns\_primary: The Primary Non-Secure application project calls the Non-Secure Callable flash driver to erase and write to a code flash region at the top of the code flash area. Upon successful flash operation, all three LEDs blink.
- app\_ra8m1\_s\_secondary: The Secondary Secure application project with FSP flash driver support with the flash driver configured as Non-Secure Callable. This application image has the same functionality as the Primary Secure application. You can use this project as a template to update the different functionalities and exercise the operation of updating the Secure image independently of the Non-Secure Image update.
- app\_ra8m1\_ns\_secondary: The Secondary Non-Secure application project, which is called the Non-Secure Callable flash driver, erases and writes to a code flash region at the top of the code flash area. Upon successful flash operation, only the blue and green LEDs blink.

#### 3.3.3 Compile All the Projects

The bootloader project must be compiled first prior to compiling the application projects. In addition, the secure project must be compiled first prior to compiling the corresponding non-secure project. For each project, open the configuration.xml file, click **Generate Project Contents**, and then click **S** to build the project. Compile the projects following the order listed below:

- 1. ra\_mcuboot\_ra8m1
- 2. app\_ra8m1\_s\_primary
- 3. app\_ra8m1\_ns\_primary
- 4. app\_ra8m1\_s\_secondary
- 5. app ra8m1 ns secondary

For the application projects, the post-build command will also sign the corresponding images. The signed image for the application project is located under the /Debug folder and is named



<application\_project\_name>\_bin.signed (For example, /app\_ra8m1\_s\_primary/Debug/app\_ra8m1\_s\_primary.bin.signed).

### 3.3.4 Debug the Applications and Boot the Primary Applications

Right-click on project app\_ra8m1\_s\_primary, select **Debug As > Debug Configurations**, and confirm the following configuration information:

- The bootloader is downloaded using the .elf format (which includes image and symbol).
- The Primary secure and non-secure images (app\_ra8m1\_s\_primary.bin.signed, app ra8m1 ns primary.bin.signed) are downloaded using the signed binary as Raw Binary.
- The Primary secure and non-secure image symbols are included using the left files.

reate, manage, and ran configurations		ř
š 🖻 🖚 🗎 🗶 🖻 🏹 🗸	Name: app_ra8m1_s_primary Debug	
type filter text	🗙 📄 Main 🏇 Debugger 🕟 Startup 🤤 Source 🔲 Common	
C (C++ Application     EASE Script     GDB Hardware Debugging     GDB Simulator Debugging (RH850)     Launch Group     C Renesas GDB Hardware Debugging	Reset and Delay (seconds): 3	^
app_ra8m1_ns_primary Debug_SSD app_ra8m1_ns_secondary Debug_SSD	Load image and symbols	
app_ra8m1_ns_primary Debug_SSD     app_ra8m1_ns_secondary Debug_SSD     app_ra8m1_s_primary Debug	Load image and symbols           Filename         Load type         Offset (hex)         On connect	
app_ra8m1_ns_primary Debug_SSD     app_ra8m1_ns_secondary Debug_SSD     app_ra8m1_s_primary Debug     app_ra8m1_s_primary Debug     ap_ra8m1_s_secondary Debug	Load image and symbols       Filename     Load type     Offset (hex)     On connect       Program Binary [app_ra     Symbols only     Yes	
app_ra8m1_ns_primary Debug_SSD     app_ra8m1_ns_secondary Debug_SSD     app_ra8m1_s_primary Debug     app_ra8m1_s_secondary Debug     ra_mcuboot_ra8m1 Debug     ra_mcuboot_ra8m1 Debug	Load image and symbols         Filename       Load type       Offset (hex)       On connect         Program Binary [app_ra       Symbols only       Yes       Add         ra_mcuboot_ra8m1.elf [       Image and Symbols       0       Yes       Edit	
app_ra8m1_ns_primary Debug_SSD     app_ra8m1_ns_secondary Debug_SSD     app_ra8m1_s_primary Debug     app_ra8m1_s_secondary Debug     ra_mcuboot_ra8m1 Debug     Renesas Simulator Debugging (RX, RL78)	Load image and symbols         Filename       Load type       Offset (hex)       On connect         Program Binary [app_ra       Symbols only       Yes       Add         ra_mcuboot_ra8m1.elf [       Image and Symbols       0       Yes       Edit         app_ra8m1_s_primary.bi       Raw Binary       2020000       Yes       Parnove	
app_ra8m1_ns_primary Debug_SSD     app_ra8m1_ns_secondary Debug_SSD     app_ra8m1_s_primary Debug     app_ra8m1_s_secondary Debug     ra_mcuboot_ra8m1 Debug     Renesas Simulator Debugging (RX, RL78)	Load image and symbols         Filename       Load type       Offset (hex)       On connect	

Figure 15. Debug Configuration RA8M1 Overwrite

#### Click Debug.

The debugger should hit the reset handler in the bootloader. Note the address is in the bootloader image.



Figure 16. Start the Application Execution

Click Resume twice IP and boot the Primary image. All three LEDs should be blinking. Pause the execution and confirm that the execution is in the Non-secure Primary slot.

Click line to run again.



#### 3.3.5 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown below. Set up the search range as 0x32000000 0xE0000.

J-Link RTT Viewer V7.98b   Configuration	×
USB     Serial No	
О ТСР/ІР	
C Existing Session	
Specify Target Device	
R7FA8M1AH	
Force go on connect	
Script file (optional)	
Target Interface & Speed	
SWD - 4000 kH:	z 🔻
RTT Control Block	
○ Auto Detection ○ Address	nge
Enter one or more address range(s) the RTT Control block Syntax: <rangestart [hex]=""> <rangesize>[, <range1star Example: 0x10000000 0x1000, 0x2000000 0x1000</range1star </rangesize></rangestart>	c can b rt [Hex
0x32000000 0xE0000	
OK Can	cel

Figure 17. Configure the RTT Viewer

Click **OK** and observe the output on the RTT Viewer. This repeated output shows that the Primary application is being executed and all three LEDs are blinking.



#### Figure 18. Execution of Primary Non-Secure Application for Overwrite Mode

### 3.3.6 Downloading and Running the Secondary Applications

During development, you can use the ancillary loading capability to load the new secure image to the intended location. You can use the example of the new secure application provided in this project and follow the steps below to perform an application upgrade:



- 1. Press the <sup>III</sup> button to pause the program.
- 2. On the top of the e<sup>2</sup> studio toolbar, click the Coad Ancillary File button to load the new application images to the Secondary slot region. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

8	X
Load A Select	Ancillary File an ancillary file for loading
File:	\${workspace_loc:\app_ra8m1_s_secondary\Debug\app_ra8m1_s_secondary.bin.signed} ~ V Workspace File System
✓ Load Address	as raw binary image           0x02010000
	OK Cancel

Figure 19. Load the Secondary Secure Application Image for Overwrite Update Mode

<b>3</b> ×
Load Ancillary File Select an ancillary file for loading
File:       \$(workspace_loc:\app_ra8m1_ns_secondary\Debug_app_ra8m1_ns_secondary.bin.signed)       V       Workspace       File System
✓ Load as raw binary image Address: 0x12050000
OK Cancel

Figure 20. Load the Secondary Non-Secure Application Image for Overwrite Update Mode

- 3. Click **Resume** I. The overwrite occurs, and the new image is executed. The blue and green LEDs will be blinking instead of all three LEDs.
- 4. On the RTT Viewer output, confirm that the following messages are printed and that only the blue and green LEDs are blinking.



#### Figure 21. Executing the Secondary Non-Secure image for Overwrite Update Mode

### 3.3.7 Update the Non-Secure Secondary Image

This step is provided as a reference for the implementation of individual image updates when designing in a TrustZone environment. The Non-Secure Secondary Image can be updated independently of the Non-Secure Primary Image.

Click **Pause** again and download the Primary Non-Secure application to the Secondary Non-Secure slot

using the Load Ancillary File 1001. Click OK. Click Resume again. The three LEDs start to blink again, and the RTT Viewer shows the same message as Figure 36.

- For Overwrite update mode, if the Secondary image is marked for update, overwrite always occurs.
- It is possible to update the Secure and Non-Secure applications individually with proper application design.



G	×
Load A	ncillary File
Select	an ancillary file for loading
File:	{\workspace_loc:\app_ra8m1_ns_secondary\Debug app_ra8m1_ns_secondary.bin.signed} \view Workspace File System
⊡_oad	as raw binary image
Address:	0x12050000
	OK Cancel

Figure 22. Load the Secondary Non-Secure Image to the Second Slot

# 3.4 Running the EK-RA8M1 Swap Update Mode Example without Test Mode with TrustZone

The process of running the EK-RA8M1 Swap Update mode is similar to the Overwrite Update mode with TrustZone. This section focuses on the difference in the operation:

- 1. Follow section 3.1.1 to initialize the RA8M1 MCU.
- 2. Import the project under folder \ra8m1\_swap\_with\_bootloader\_tz to a workspace.



Figure 23. Example Projects for RA8M1 Swap Update Mode



- The bootloader project ra mcuboot ra8m1 has similar functionality as the bootloader with Overwrite Update mode introduced in section 3.3 step 3.3.3 except that the memory map definition and application image update mode are different.
- The functionalities of the application projects are the same as the Overwrite Update mode.
- 3. Configure the Python Signing Environment by following section 3.2 if this is the first time you have signed the application image.
- 4. Compile the example projects in the same order as the Overwrite update mode by referencing section 3.3 step 3.3.3. Ensure the signed image for the application project is located under the /Debug folder and is named <application project name> bin.signed.
- Review the Debug Configuration and boot the Primary applications by referencing section 3.3.4. 5.

	ations				
📑 🖻 闷 🗎 🗶 🖻 🏹 🗕	Name: app_ra8m1_s_primary De	bug			
type filter text ×	📄 Main 🕸 Debugger 🕟 Sta	rtup 🔲 Common 🍹	Source		
C/C++ Application	Initialization Commands				
C/C++ Remote Application	Reset and Delay (seconds)	3			
EASE Script					
GDB Hardware Debugging					
GDB Simulator Debugging (RH8:					
Renesas GDB Hardware Debuggi					
app ra8m1 ns primary Debug					
<pre>app_ra8m1_ns_secondary Deb</pre>	Load image and symbols				
app_ra8m1_s_primary Debug	Filename	Load type	Offset (hex)	On connect	
	Program Binary [app_ra	Symbols only		Yes	
app_ra8m1_s_secondary Debu	√ ra meuboot ra9m1 olf [	Image and Symbols	0	Yes	
app_ra8m1_s_secondary Debu     ra_mcuboot_ra8m1 Debug     Papasas Simulator Debugging (R			2030000	Yes	
app_ra8m1_s_secondary Debu     ar_ncuboot_ra8m1 Debug     Renesas Simulator Debugging (R	✓ app_ra8m1_s_primary.bi	Raw Binary			
app_ra8m1_s_secondary Debu     ra_mcuboot_ra8m1 Debug     Renesas Simulator Debugging (R	<ul> <li>✓ app_ra8m1_s_primary.bi</li> <li>✓ app_ra8m1_ns_primary</li> </ul>	Raw Binary Raw Binary	12040000	Yes	

Figure 24. Debug Configuration RA8M1 Swap Update Mode

- 6. Open the J-Link RTT Viewer and set up the same configuration as Figure 17.
- 7. Click **OK** and observe the following output on the RTT Viewer. This output shows that the Primary application is being executed, and all three LEDs are blinking.

00>	Running the Primary non-secure application with swap update mode.
00>	
00>	flash write successful!
00>	Flash Operation is successful. The Red, Blue and Green LEDs should be blinking.
00>	
00>	Running the Primary non-secure application with swap update mode.
00>	
00>	flash write successful!
00>	Flash Operation is successful. The Red, Blue and Green LEDs should be blinking.

#### Figure 25. Execution of Primary Non-Secure Application for Swap Update Mode

#### **Downloading and Running the Secondary Applications** 3.4.1

During development, you can use the Ancillary loading capability to load the new Secure image to the intended location. You can use the example new Secure application provided in this application and follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

- Press the <u>u</u> button to pause the program.
- 2. Load the secure new application images to the Secondary slot region using the Ancillary loading

capability is from the top of the e<sup>2</sup> studio toolbar in a similar way as Figure 19 except use address 0x02020000.



- 3. Load the non-secure new application image to the Secondary slot region using the Ancillary loading capability from the top of the e<sup>2</sup> studio toolbar in a similar way as Figure 20 except use address 0x12060000.
- 4. Click **Resume** . The swap occurs, and the new image is executed. Only the blue and green LEDs should be blinking.
- 5. Confirm the execution result.



Figure 26. Executing the Secondary Non-Secure Image for Swap Update Mode

# 3.5 Running the EK-RA8M1 DXIP Update Mode Example

The process of running the EK-RA8M1 DXIP Update Modes is similar to the Overwrite Update mode. This section will focus on the difference in the operation:

- 1. Follow section 3.1.1 to initialize the RA8M1 MCU.
- 2. Import the project under folder \ra8m1\_dxip\_with\_bootloader\_flat to a workspace and see the following set of example projects.



Figure 27. Example Projects for RA8M1 Direct XIP Update Mode

The functionalities of the application projects are blinking the LEDs and providing RTT viewer outputs.Configure the Python signing environment by following section 3.2 if this is the first time you have signed the application image.

- 4. The bootloader needs to be compiled first. For each project, open the configuration.xml file, click Generate Project Contents, and then click so to build the project. Compile the example projects following the order below. Ensure the signed image for the application project is located under the /Debug folder and is named <application project name> bin.signed
  - 1. ra\_mcuboot\_ra8m1\_dxip
  - 2. app\_ra8m1\_primary
  - 3. app\_ra8m1\_secondary
- 5. Verify the debug configuration and follow section 3.3 step 3.3.4 to start debugging the application.



Create, manage, and run computation	ns				Ť.
📑 🖻 闷 🐹 🖻 🏹 🗝	Name: app_ra8m1_primary Deb	oug_Flat			
	📄 Main 🕸 Debugger 🕟 Sta	artup 🦆 Source 🔲 C	ommon		
C/C++ Application	Initialization Commands				^
C/C++ Remote Application	Reset and Delay (seconds)	3			
GDB Hardware Debugging	Halt				
GDB Simulator Debugging (RH850)					^
Launch Group					
Renesas GDB Hardware Debugging Renesas GDB Hardware Debugging Renesas GDB Hardware Debugging					×
app_ra8m1_secondary Debug_Flat	Load image and symbols				
ra_mcuboot_ra8m1_dxip Debug_Fla	Filename	Load type	Offset (hex)	On connect	Add
Reflesas Simulator Debugging (KA, KL	Program Binary [app_ra	Symbols only		Yes	Add
	✓ ra_mcuboot_ra8m1_dxi	Image and Symbols	0	Yes	Edit
	Image: App_ra8m1_primary.bin	. Raw Binary	2010000	Yes	Remove
					Move up

Figure 28. Debug Configuration DXIP Update Mode

- 6. Open the J-Link RTT Viewer and set up a configuration similar to Figure 17, except change the search range to 0x22000000 0x8000.
- 7. Click **OK** and observe the following output on the RTT Viewer. This output shows that the Primary application is being executed, and all three LEDs are blinking.





#### 3.5.1 Downloading and Running the Secondary Applications

Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

During development, you can use the Ancillary loading capability from the top of the e<sup>2</sup> studio toolbar to load the new image to the intended location. You can use the example new application provided in this application and follow the steps below to perform an application upgrade:

- 1. Press the <sup>III</sup> button to pause the program.
- 2. Load the new application images to the Secondary slot region using the Ancillary loading capability from the top of the e<sup>2</sup> studio toolbar.



	×
Load A	ncillary File
Select a	an ancillary file for loading
File:	\${workspace_loc:\app_ra8m1_secondary\Debug\app_ra8m1_secondary.bin.signed} ~ Workspace File System
Load a	as raw binary image
Address:	0x02020000



- 3. Click **Resume** I . The swap occurs, and the new image is executed. Only the blue and green LEDs should be blinking.
- 4. Confirm the same configuration as shown in Figure 17, then click OK.





#### 3.6 Running the EK-R8M1 Overwrite Update Mode Example without TrustZone

Follow the steps below to run the example projects for EK-RA8M1 using the MCUboot Overwrite Only Update mode without TrustZone.

# 3.6.1 Import the Projects under Folder \ra8m1\_overwrite\_with\_bootloader to a Workspace

The following example projects are included in this folder:



Figure 32. Example Projects for RA8M1 Overwrite Update Mode

- Project ra mcuboot ra8m1 is the bootloader project.
- Project app\_ra8m1\_primary is the initial Primary application project. This project blinks the three
  LEDs on the EK-RA8M1 kit.
- Project app\_ra8m1\_secondary is the Secondary application project. This project blinks the blue LED on the EK-RA8M1 kit.

Follow section 3.2 to set up the Python signing environment if this is the first time you have signed the application image.

#### 3.6.2 Compile the Projects

The bootloader needs to be compiled first. For each project, open the configuration.xml file, click **Generate Project Contents**, and then click s to build the project. For the application projects, the post-build command will also sign the corresponding images. The signed image is located under the \Debug



Renesas RA Family

folder and is named <project\_name>.bin.signed (for example, /app\_ra8m1\_primary/Debug/app\_ra8m1\_primary.bin.signed)

- 1. ra\_mcuboot\_ra8m1
- 2. app\_ra8m1\_primary
- 3. app\_ra8m1\_secondary

# 3.6.3 Debug the Applications and Boot the Primary Application

Right-click on project app\_ra8m1\_primary and select Debug As > Debug Configuration.

reate, manage, and run configuration	ns				-	ì
C/C++ Application C/C++ Application C/C++ Remote Application C/C++ Remote Application C/C++ Remote Application C/C GDB Hardware Debugging C/C GDB Jamulator Debugging C/C app ra8m1 primary Debug flat C/C app ra8m1 primary Debug fla	Name: app_ra8m1_primary Debu Main 参 Debugger Stat Initialization Commands Reset and Delay (seconds): Halt	ig_Flat tup 🖅 Source 🔲 Co	ommon		×	^
<ul> <li>app_aom_secondary beoug_name</li> <li>accurate a secondary beoug_name</li> <li>accurate a secondary beought</li> <li>Renesas Simulator Debugging (RX, R</li> </ul>	<ul> <li>Filename</li> <li>Filename</li> <li>Program Binary (app_ra</li> <li>✓ ra_mcuboot_ra8m1.elf [</li> <li>✓ app_ra8m1_primary.bin</li> </ul>	Load type Symbols only Image and Symbols Raw Binary	Offset (hex) 0 2010000	On connect Yes Yes Yes	Add Edit Remove Move up	

Figure 33. Debug Configuration RA8M1 Overwrite Update

#### Click Debug.

The debugger should be at the reset handler in the bootloader. Note the address is in the bootloader image.

🎋 Debug 🛛 📄 🦌 📴 🗖	😼 startup.c 🗙
✓ ☑ app_ra8m1_primary Debug_Flat [Renesas GDB Hardware Debugging]	50 02008dd0 SystemInit();
✓ Papp_ra8m1_primary.elf [1] [cores: 0]	51
✓ IP Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:Trace/breakpoint trap	52 /* Call user application. */
Reset_Handler() at startup.c:50 0x2008dd0	53 02008dd6 main();
	34

Figure 34. Start the RA8M1 Application Execution

Click Resume twice IP and boot the Primary image. All three LEDs should be blinking.

#### 3.6.4 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown below. Configure the address search range as 0x22000000 0x8000.



J-Link RTT Viewer V7.98b   Configuration X
Connection to J-Link USB Serial No TCP/IP
C Existing Session
Specify Target Device       R7FA8M1AH
Script file (optional)
Target Interface & Speed
SWD • 4000 kHz •
RTT Control Block         Auto Detection       Address         Enter one or more address range(s) the RTT Control block can be located in.         Syntax: <rangestart [hex]=""> <rangesize>[, <range1start [hex]=""> <range1size>,]         Example: 0x10000000 0x1000, 0x2000000 0x1000         0x22000000 0x8000</range1size></range1start></rangesize></rangestart>
OK Cancel

Figure 35. Configure the RTT Viewer for RA8M1 Project

Click **OK** and observe the following output on the RTT Viewer. This output shows that the Primary application is being executed and all three LEDs are blinking.



#### Figure 36. Execution of Primary Application for Overwrite Mode

#### 3.6.5 Downloading and Running the Secondary Applications

During development, you can use the Ancillary loading capability to load the new Secure image to the intended location. Follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.



- 1. Press <sup>III</sup> to pause the program.
- Load the new application images to the Secondary slot region using the Ancillary loading capability from the top of the e<sup>2</sup> studio toolbar. Select Load as raw binary image and configure the Address to 0x02020000.

•		×
Load A	ncillary File	
Select	an ancillary file for loading	
File:	\${workspace_loc:\app_ra8m1_secondary\Debus \app_ra8m1_secondary.bin.signed}	File System
.oad	as raw binary image	
Address:	0x02020000	]
	ОК	Cancel

Figure 37. Load the Secondary Application Image for Overwrite Mode

- 3. Click Resume . The overwrite occurs, and the new image is executed. Now, only the Blue LED should be blinking.
- 4. Confirm the same configuration as shown in Figure 35, then click **OK**. The following output is printed, and only the blue LED blinks.



Figure 38. Executing the Secondary Application Image for Overwrite Update Mode

# 3.7 Running the EK-RA8M1 Swap Test Update Mode Example without TrustZone

Follow the steps below to run the example projects for EK-RA8M1 using the MCUboot Swap Test Update mode without TrustZone.

### 3.7.1 Import the Projects

Import the projects under Folder \ra8m1\_swap\_test\_with\_bootloader to a Workspace.

The following example projects are included in this folder:



#### Figure 39. Example Projects for RA8M1 Swap Test Update Mode

- **Project** ra\_mcuboot\_ra8m1\_swap\_testmode is the bootloader project.
- Project app\_ra8m1\_primary is the initial Primary application project. This project blinks the three
  LEDs on the EK-RA8M1 kit.
- Project app\_ra8m1\_secondary is the Secondary application project. This project blinks the blue LED on the EK-RA8M1 kit.

Follow section 3.2 to set up the Python signing environment if this is the first time you have signed the application image.



#### 3.7.2 Compile the Projects

The bootloader project needs to be compiled first. For each project, open the configuration.xml file, click Generate Project Contents, and then click so build the project. Compile the projects in the following order:

- 1. ra\_mcuboot\_ra8m1\_swap\_testmode
- 2. app\_ra8m1\_primary
- 3. app\_ra8m1\_secondary

For the application projects, the post-build command will also sign the corresponding images. The signed image is located under the \Debug folder and is named <project\_name>.bin.signed (for example, /app\_ra8m1\_primary/Debug/app\_ra8m1\_primary.bin.signed)

#### 3.7.3 Debug the Applications and Boot the Primary Application

Right-click on project **app\_ra8m1\_primary** and select **Debug As > Debug Configurations**.

bebug comgatations		, ,
reate, manage, and run configu	ations	TO T
ype filter text C/C++ Application C/C++ Remote Application C/C++ Seciet	Name:       app_ra8m1_primary Debug_Flat         Image: Startup       Image: Startup         Image: Startup       Image	^
<ul> <li>CASE Script</li> <li>C GDB Hardware Debugging</li> <li>G GDB Simulator Debugging (RH</li> <li>Launch Group</li> <li>Renesas GDB Hardware Debug</li> <li>app_ra8m1_primary Debug_</li> <li>app_ra8m1_secondary Debu</li> </ul>	Halt     Load image and symbols	<u>`</u>
ra_mcuboot_ra8m1_swap_te Renesas Simulator Debugging (	Filename         Load type         Offset (hex)         On connect           Program Binary Tapp ra         Symbols only         Yes	Add
	✓ ra_mcuboot_ra8m1_swa Image and Symbols 0 Yes	Edit
	✓ app_ra8m1_primary.bin Raw Binary 2010000 Yes	Remove
		Move up
		Move down
		WOVE GOWIT

Figure 40. Debug Configuration RA8M1 Overwrite Update

Click Debug.

Click Resume twice III and boot the Primary image. All three LEDs should be blinking.

#### 3.7.4 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown in Figure 35. Observe the following output on the RTT Viewer. This output shows that the Primary application is being executed and all three LEDs are blinking.

00> Running the Primary application with swap (test mode) update mode. 00> The Red, Blue and Green LEDs should be blinking.

Figure 41. Execution of Primary Application for Swap Test Mode



#### 3.7.5 Downloading and Running the Secondary Applications

During development, you can use the Ancillary loading capability to load the new Secure image to the intended location. Follow the steps below to perform an application upgrade. Refer to section 3.8 for troubleshooting when using the Load Ancillary File function.

- 1. Press <sup>III</sup> to pause the program.
- 2. Load the new application images to the Secondary slot region using the Ancillary loading capability from the top of the e<sup>2</sup> studio toolbar in a similar way as Figure 37. Select **Load as raw binary image** and configure the **Address** to 0x02020000.
- 3. Click Resume . A swap occurs, and the new image is executed. Now, only the blue LED should be blinking.
- 4. Confirm the same configuration as shown in Figure 35, then click **OK**. The following output is printed and only the blue LED should blink.

00> Running the Secondary application with swap (test mode) update mode. 00> The blue LED should be blinking.

#### Figure 42. Executing the Secondary Application Image for Swap Test Update Mode

5. Pause and reset the application from the debugger.

#### 3.8 Troubleshooting

When running the example projects, you may experience a USB Debug connection or the RTT Viewer

connection issue when using the "Load Ancillary File" button 🔊 to download the Secondary image. To recover from these failures:

- If the USB Debug connection disconnects, the recommendation is to try out another available USB port on the development PC for the USB Debug connection. If failure persists, contact Renesas support.
- If the RTT Viewer disconnects, the recommendation is to power cycle the board and restart the debug session.

### 4. Creating the Bootloader

This section provides a walk-through of the bootloader creation of the example projects, as well as how to link the standalone application with the bootloader. For most of the steps, the considerations and configurations in creating a bootloader with the different upgrade modes are common. Whenever there is a difference in the implementation of the different update modes, the difference will be addressed.

The walk-through of the bootloader creation in this section targets the bootloader used in section 3.3 for the TrustZone<sup>®</sup> enabled system. Wherever there is a need to address the Non-TrustZone-enabled implementation, it will be addressed.

# 4.1 Creating a Bootloader Project for RA Family

The screen captures used in these sections are based on the RA8M1-based bootloader projects used in section 3.3, 3.4, and 3.5. Follow this section to establish the bootloader projects used in section 3.3, which uses Overwrite Only as the application update mode. Updates needed for the bootloader projects used in the section 3.4 and 3.5 are addressed.



#### 4.1.1 Start Bootloader Project Creation with e<sup>2</sup> studio

Follow the steps below to create the initial bootloader project based on EK-RA8M1:

1. From the e<sup>2</sup> studio Workspace, navigate to the File > New > Renesas C/C++ Project > Renesas RA and then select Renesas RA C/C++ Project and press Next.

Provide the project name ra\_mcuboot\_ra8m1 and click **Next**. The exact name needs to be provided to follow the default instructions in this section. If a different name is provided, all instructions related to the name of the bootloader project need to be updated accordingly.

In the next screen, select FSP version 5.5.0 and the EK-RA8M1 board. Use the default Debugger setting J-Link Arm and click Next.

Note that if the creation process is using other newer FSP versions, some details on the error messages shown when the MCUboot module is initially added may be different. Adapt the actions accordingly to satisfy the dependencies.

- 2. When the following screen appears.
- For a Bootloader Project using TrustZone, select TrustZone Secure Project.



Figure 43. Choose TrustZone Secure Project as Project Type



#### • For a Bootloader Project not using TrustZone, select Flat (Non-TrustZone) Project.



Figure 44. Choose Flat Project as Project Type



#### 3. The following screen appears.

**For a Bootloader Project using TrustZone**, choose the project template. As shown in Figure 45, there are two Secure project templates. You can choose which templates to use based on whether an RTOS is used in the Non-secure project.

#### • Bare Metal – Minimal

Secure project with MCU Initialization functions with support for transitioning to Non-secure partition. This application note uses the **Bare Metal – Minimal** project template as an example to explain the general steps of creating a secure project.

#### • TrustZone Secure RTOS – Minimal

- Secure projects will add the required RTOS context in the Secure region for the Thread that needs to access the NSC APIs in an RTOS-enabled project. When this project type is selected, the Arm TrustZone Context RA Port will be added, as shown in Figure 45.
- The RTOS kernel and user tasks will reside in the Non-secure partition.

Renesas RA C/C++ Project       Renesas RA C/C++ Project <ul> <li>×</li> <li>×</li> <li>Project Template Selection</li> </ul>	
Project Template Selection	
Code Generation Settings           Use Renesas Code Formatter	

Figure 45. Choose the Project Template

Click Finish to allow the Project Generator to populate the project template.



For a Bootloader Project not using TrustZone, choose Executable as the Build Artifact Selection and No RTOS. Click Next.

Renesas RA C/C++ Project      Renesas RA C/C++ Project      Build Artifact and RTOS Selection      Build Artifact Selection	RTOS Selection	- • ×	
<ul> <li>Executable         <ul> <li>Project builds to an executable file</li> </ul> </li> <li>Static Library         <ul> <li>Project builds to a static library file</li> </ul> </li> <li>Executable Using an RA Static Library         <ul> <li>Project builds to an executable file</li> <li>Project uses an existing RA static library project</li> </ul> </li> </ul>	No RTOS	×	
?	ack Next > Finish	Cancel	

Figure 46. Choose Executable and No RTOS

In the next screen, select the project template.

Choose Bare-Metal – Minimal as the Project Template Selection and click Next.
4. For a Bootloader Project using TrustZone. In the clocks tab, set Clock as Security.

	🔒 Security 💀 Restore Defaults	
Clock Src: PLL1P → CPUCLK Div /1 → CPUCL	Sets whether the clock circuit is secure or non-	secure (currently secure; override disabled)
>ICLK Div /2 ~	→ ICLK 240MHz	RENESAS

Figure 47. Security Clock Security

5. Update the Pin configuration file.

The project will now be created, and the bootloader project configuration will be displayed. Select the **Pins** tab and deselect the **Generate data** check box. Use the pull-down menu to switch from **RA8M1 EK** to **R7FA8M1AHECBD.pincfg** for the **Select Pin Configuration** option, select the **Generate data** check box, and enter **g\_bsp\_pin\_cfg**.

elect Pin Configuration		📑 Export to CSV file  🔚 Configure Pin Driver Warnings
RA8M1 EK	✓ Manage configurations	Generate data; g_bsp_pin_cfg

#### Figure 48. Uncheck Generate Data for RA8M1 EK Pin Configuration

Select Pin Configuration		🔛 Export to CSV file
R7FA8M1AHECBD.pincfg v	Manage configurations	Generate data: g_bsp_pin_cfg

### Figure 49. Select R7FA8M1AHECBD.pincfg and Generate data g\_bsp\_pin\_cfg

Note that when we select the Flat Project model, the I/Os are configured as Secure by default. Updating the pin configuration, as shown above, selects the pin configuration with the minimal number of pins defined because any I/O that is defined in the Flat project will not be available for use in the Non-Secure application and can only be accessed by the Secure application.



#### 6. Add the MCUboot module.

Change to the Stacks tab and select New Stack > Bootloader > MCUboot.

HAL/Common Stacks	New Stack Stack Stack Stack Stack
	Analog >
4 a japart 1/0 Part	Artificial Intelligence
(r isport)	Audio
(1_104017)	Bootloader > 🕀 MCUboot
	CapTouch > 🕀 MCUboot Image Utilities
<b>U</b>	Connectivity >
	DSP

Figure 50. Add the MCUboot Module

#### 4.1.2 **Resolve the Configurator Dependencies**

After the MCUboot module is brought into the configurator, follow the steps in this section to resolve the dependencies:

1. Resolve the following dependency of the MCUboot by adding the MbedTLS (Crypto Only) stack.



Figure 51. MCUboot Module Dependency

Left-click on Add Crypto Stack, choose New and add the **MbedTLS (Crypto Only)** stack.

Stacks Configuration Genera			
Threads	HAL/Common Stacks	🔊 New Stack > 🏯 Extend Stack > 🔬 Remove	
<ul> <li>✓ R HAL/Common</li> <li> <i>⊕</i> g_ioport I/O Port (r_ioport) <i>⊕</i> MCUboot      </li> </ul>	MCUboot	MCUboot logging	
	Add Requires a crypto     stack     New     MCUboot Custom Crypto (Protect     McUboot Custom Crypto (RW Only)     McUboot Custom Crypto (WO Only)     ModrLS (Crypto Only)	Add External Memory Implementation (Optional) ted Mode)	

Figure 52. Add MbedTLS (Crypto Only) Module



 Configure the Mbed Crypto dependencies.
 Follow the prompt in Figure 53 to update the corresponding properties for the MCUboot Port for the RA Module.

Al /Common Stacks	New Stack >
MCUboot	
(1)	
HCUboot Port for RA (rm_mcu	poot_port) 🗘 MCUboot log
MbedTLS (Crypto Only)     MbedTLS (Crypto Only)     Mbed Crypto H/W     Acceleration     Mm.psa.crypto)     SCE Compatibility Mode	A stack element with a bar of this color indicates 'MCUboot Port for RA (rm_mcuboot_port)' is a Module instance. This instance may be referenced by one other module instance only. Error: Requires Flash Driver Error: MBEDTLS_THREADING_C in MbedTLS (Crypto Only) must not be defined under MbedTLS (Crypto Only) Common General MBEDTLS_THREADING_C. Error-related property: MbedTLS (Crypto Only) > General > MBEDTLS_THREADING_C Error: MBEDTLS_THREADING_ALT in MbedTLS (Crypto Only) must not be defined under MbedTLS (Crypto Only) Common General MBEDTLS_THREADING_ALT. Error: MBEDTLS_THREADING_ALT in MbedTLS (Crypto Only) must not be defined under MbedTLS (Crypto Only) Common General MBEDTLS_THREADING_ALT. Error: MBEDTLS_THREADING_ALT in MbedTLS (Crypto Only) must not be defined under MbedTLS (Crypto Only) Common General MBEDTLS_THREADING_ALT. Error-related property: MbedTLS (Crypto Only) > General > MBEDTLS_THREADING_ALT Error: MBEDTLS_MEMORY_BUFFER_ALLOC_C in MbedTLS (Crypto Only) must be defined under MbedTLS (Crypto Only) Common General MBEDTLS_MEMORY_BUFFER_ALLOC_C. Error-related property: MbedTLS (Crypto Only) > General > MBEDTLS_MEMORY_BUFFER_ALLOC_C Error: Code Flash Programming must be Enabled under Flash Common Code Flash Programming Enable Error: Tota flash bgo must be disabled under Flash Module Data Flash Background Operation
Add Key Injection for PSA Crypto (Optional)	

Figure 53. Dependencies of MCUboot Module for RA Stack

Configure the following properties:

d MbedTLS (Crypto Only)	The second secon
6	
L <u> </u>	
<	
SP Clocks Pins Interrupts Event Links 🧕 Stacks Com	ponents
Smart Browser Properties ×	
(Crypto Only)	
Property	Value
✓ General	
MBEDTLS_PSA_CRYPTO_DRIVERS	Undefine
MBEDTLS_DEPRECATED_WARNING	Undefine
MBEDTLS_DEPRECATED_REMOVED	Define
MBEDTLS_CHECK_RETURN_WARNING	Undefine
MBEDTLS_ERROR_STRERROR_DUMMY	Define
MBEDTLS_MEMORY_DEBUG	Undefine
MBEDTLS_MEMORY_BACKTRACE	Undefine
MBEDTLS_PSA_CRYPTO_CLIENT	Undefine
MBEDTLS_PSA_CRYPTO_SPM	Undefine
MBEDTLS_SELF_TEST	Undefine
MBEDTLS_THREADING_ALT	Undefine
MBEDTLS_THREADING_PTHREAD	Undefine
MBEDTLS_USE_PSA_CRYPTO	Undefine
MBEDTLS_VERSION_FEATURES	Define
MBEDTLS ERROR C	Define
MBEDTLS_MEMORY_BUFFER_ALLOC_C	Define
MBEDTLS_PSA_CRYPTO_C	Define
MBEDTLS PSA CRYPTO SE C	Undefine
MBEDTLS THREADING C	Undefine
MBEDTLS_TIMING_C	Undefine
MBEDTLS_VERSION_C	Define
MBEDTLS_MEMORY_ALIGN_MULTIPLE	Undefine
MBEDTLS_MEMORY_ALIGN_MULTIPLE value	4
MBEDTLS_CHECK_RETURN	Define
MBEDTLS_IGNORE_RETURN	Undefine

Figure 54. Configure Highlighted Properties for the MbedTLS (Crypto Only) Stack



**Renesas RA Family** 

Add the r\_flash\_hp module:



#### Figure 55. Add the r\_flash\_hp Stack

#### Configure the r\_flash\_hp stack:

		-	<b>A</b>
á	MbedTLS (Crypto Only	)	<pre>     g_flash0 Flash     (r_flash_hp)     ① </pre>
	5	•	•
			7
	Mbed Crypto HW	Add Persistent	
BSP Clocks Pins Interrupts	Event Links 🔞 Stacks	Components	
ms 👒 Smart Browser 🏽 🍡 F	Pin Conflicts 🔲 Properti	es 🛛	
Flash (r_flash_hp)			
Property		Value	
✓ Common			
Parameter Checking		Default (BSP)	
Code Flash Program	ming Enable	Enabled	
Data Flash Programm	ning Enable	Disabled	
<ul> <li>Module g_flash0 Flash (</li> </ul>	r_flash_hp)		
Name		g_flash0	
Data Flash Backgrou	nd Operation	Disabled	
Callback		NULL	
Flash Ready Interrup	t Priority	Disabled	
Flash Error Interrupt	Priority	Disabled	

#### Figure 56. Configure the r\_flash\_hp Stack

- 3. Hover the cursor over MbedTLS (Crypto Only) stack. You will see warnings as shown in Figure 57.
- 4. Under the BSP tab, set up the stack and heap size to support ECC:
  - RA Common > (set Main stack size to 0x1000 and Heap size to 0x400)

MbedTLS (Crypto Only)	g_flash0 Flash (r_flash_hp)	Add External Memory Implementation (Optional)		
A stack element with a bar of this color indicates 'MbedTLS ( This instance may be referenced by multiple other module in	Crypto Only)' is a common module stances across multiple stacks.	instance.		
Error: A minimum heap of 0x400 is required to use ECC. To d	isable ECC, under Common PKC EC	C, undefine MBEDTLS_ECP_C and	anything else that uses ECC (MBEDTLS_ECDSA_C).	
Error-related property:				
MbedTLS (Crypto Only) > Public Key Cryptography (PKC	) > ECC >			
Error: A minimum heap of 0x1500 is required to use RSA. To	disable RSA, under Common PKC R	SA undefine MBEDTLS_RSA_C, a	nd under Common PKC, undefine MBEDTLS_PK_C, MBEDTLS_PK_F	PARSE_C, MBEDTLS_PK_WRITE_C.
Error-related property:				
MbedTLS (Crypto Only) > Public Key Cryptography (PKC	) > RSA > MBEDTLS_RSA_C			
Error: A minimum heap of 0x200 is required to use AES. AES	cannot be disabled.			
Error-related property:				
MbedTLS (Crypto Only) > Cipher > MBEDTLS AES C				
Error: A minimum stack of 4K (0x1000) is required. If used in	an RTOS thread, the thread stack sl	hould instead be at least 0x1000.		
			11.1	

#### Figure 57. Dependencies of Mbed TLS (Crypto Only) Stack

5. Disable RSA following the prompt in Figure 58. This bootloader design uses ECC for signature generation. Disable the RSA algorithm to save the BSP Heap size.

Mbed LS (Crypto Only)	g_flash0 Flash (r_flash_hp)	Add External Memory		
stack element with a bar of this color indicates 'MbedTLS (C his instance may be referenced by multiple other module ins	rypto Only)' is a common module in tances across multiple stacks.	nstance.		
rror: A minimum heap of 0x1500 is required to use RSA. To d	lisable RSA, under Common PKC RS	5A undefine MBEDTLS_RSA_C, an	d under Common PKC, undefine MBED	'LS_PK_C, MBEDTLS_PK_PARSE_C, MBEDTLS_PK_WRITE_C.
MbedTLS (Crypto Only) > Public Key Cryptography (PKC)	> RSA > MBEDTLS RSA C			

Figure 58. Dependencies of RSA



MbedTLS (Crypto Only	↓ ↓ ↓ ■ ● • • • • • • • • • • • • • • • • • • •		
BSP Clocks Pins Interrupts Event Links Stacks Components The Second Stacks Components Stacks Stacks Components Stacks Stack			
.S (Crypto Only)			
<b>.S (Crypto Only)</b> Property	Value	ie	
-S (Crypto Only) Property V Public Key Cryptography (PKC)	Value	ie	
-S (Crypto Only) Property ✓ Public Key Cryptography (PKC) → DHM	Value	ie	
S (Crypto Only) Property Problec Key Cryptography (PKC) DHM ECC	Value	le	
S (Crypto Only) Property Public Key Cryptography (PKC) DHM ECC RSA	Value	ie	
S (Crypto Only)  Property  Public Key Cryptography (PKC)  DHM  ECC  RSA MBEDTLS_PK_RSA_ALT_SUPPORT	Value	le	
S (Crypto Only)  Property  Public Key Cryptography (PKC)  DHM  ECC  RSA  MBEDTLS_PK_RSA_ALT_SUPPORT  MBEDTLS_RSA_NO_CRT	Value Unde Defir	lefine	

Figure 59. Disable RSA

At this point, the error message in the stack window should have been resolved.

- 6. Decide the number of application images.
  - For MCUs with TrustZone support:
    - If the application uses TrustZone, there will be two application images in each slot: secure and nonsecure applications. In this case, set the **Number of images per Application** to 2.
    - If the application does not use TrustZone, there will be one application image in each slot. In this case, set the **Number of images per Application** to 1.
    - The bootloader used in section 3.3 uses TrustZone, so for this example bootloader, set the Number of images per Application to 2. MCUboot > Common > General > Number of images per Application (change from 1 to 2).

For MCUs without TrustZone support, set this property to 1.

7. Configure the **Flash Layout** for RA8M1 Overwrite Update mode with TrustZone as shown below based on the standalone application projects described in section 5. For your application projects, you can follow the guidelines in section 2.3 to design the bootloader memory allocation. This configuration matches the bootloader used in section 3.3.



Figure 60. Memory Configuration of Overwrite Update Mode RA8M1 with TrustZone



Configure the MCUboot module and application memory allocation based on RA8M1 Swap Update mode with TrustZone as shown below based on the standalone application projects described in section 5. This configuration matches the bootloader used in section 3.4.

				0x1208_0000
		Secondary slot	Secondary Non-Secure App	
✓ Flash Layout		Primary slot	Primary Non-Secure App	0x1206_0000
✓ TrustZone				0.0004.00004.0.4004.0000
Non-Secure Callable Region Size (Bytes)	0xC00			0x0204_000070x1204_0000
Non-Secure Flash Area Size (Bytes) (TrustZone Non-Secure)	0x20000	Primary slot	Primary Secure App	
Non-Secure Callable RAM Region Size (Bytes)	0x0			0x0203_0000
Non-Secure RAM Region Size (Bytes) (TrustZone Non-Secure)	0x2000	Secondary	Secondary Secure App	
Image 2 Header Size (Bytes)	0x200	slot		0x0202_0000
Bootloader Flash Area Size (Bytes)	0x18000		Scratch Area (size=0x8000)	
Image 1 Header Size (Bytes)	0x200			0x0201_8000
Image 1 Flash Area Size (Bytes)	0x10000		MCUboot	
Scratch Flash Area Size (Bytes)	0x8000			0x0200 0000

Figure 61. Memory Configuration of Swap Update Mode RA8M1 with TrustZone

Configure the MCUboot module and application memory allocation based on RA8M1 Direct XIP mode based on the example projects presented in section 3.5. This configuration matches the bootloader used in section 3.5.



Figure 62. Memory Configuration of Direct XIP Update Mode RA8M1

Configure the MCUboot module and application memory allocation based on the RA8M1 Overwrite Update mode without TrustZone based on the example projects presented in section 3.6. This configuration matches the bootloader used in section 3.6.

TrustZone		-	
Non-Secure Callable Region Size (Bytes)	0x0	Secondary	Secondary App
Non-Secure Flash Area Size (Bytes) (TrustZone Non-Secure)	0x0	slot	eccondary rep
Non-Secure Callable RAM Region Size (Bytes)	0x0	Primary	
Non-Secure RAM Region Size (Bytes) (TrustZone Non-Secure)	0x0	slot	Primary App
Image 2 Header Size (Bytes)	0x200		
Bootloader Flash Area Size (Bytes)	0x18000		Scratch Area (size=0x0)
Image 1 Header Size (Bytes)	0x200		
Image 1 Flash Area Size (Bytes)	0x20000		MCUboot
Scratch Flash Area Size (Bytes)	0x0	-	

Figure 63. Memory Configuration of Overwrite Update Mode RA8M1 without TrustZone



Configure the MCUboot module and application memory allocation based on RA8M1 Swap Test Update mode without TrustZone based on the example projects presented in section 3.7. This configuration matches the bootloader used in section 3.7.

✓ TrustZone				0x0203_800
Non-Secure Callable Region Size (Bytes)	0x0		Scratch Area (size=0x8000)	
Non-Secure Flash Area Size (Bytes) (TrustZone Non-Secure)	0x0	Secondary		0x0203_000
Non-Secure Callable RAM Region Size (Bytes)	0x0	slot	Secondary App	
Non-Secure RAM Region Size (Bytes) (TrustZone Non-Secure)	0x0			0x0202_000
Image 2 Header Size (Bytes)	0x200	Primary	Primary App	
Bootloader Flash Area Size (Bytes)	0x10000	SIDE	,	0~0201_000
Image 1 Header Size (Bytes)	0x200		MOULE	0x0201_0000
Image 1 Flash Area Size (Bytes)	0x10000		TOODODIM	
Scratch Flash Area Size (Bytes)	0x8000			0x0200_000

#### Figure 64. Memory Configuration of Swap Test Update Mode RA8M1 without TrustZone

For the configuration of the swap test mode run-time support, refer to application note R11AN0516 to understand the operation.

#### 4.1.3 Setting up the Booting Authentication Support

You can choose to use the default pair of public/private keys included in MCUboot for testing purposes:

- The default public keys are defined in /ra\_mcuboot\_ra8m1/ra/mcutools/MCUboot/sim/mcuboot-sys/csupport/keys.c.
- The default private keys are included in folder /ra\_mcuboot\_ra8m1/ra/mcu-tools/MCUboot/sim.



Figure 65. Example Public Keys and Private Keys Included in MCUboot Port Stack

To use the example keys, select Add Example Keys > New > MCUboot Example Keys (NOT FOR PRODUCTION).



	MCUboot logging	Add [Optional] Add Example Keys Crypto (Protected Mode)
Add External Memory Implementation (Optional)	0	New       MCUboot Example Keys (NOT FOR PRODUCTION)



Note: The example public key and private key used in the MCUboot is for testing purposes only. Refer to section 2.6 for guidelines on selecting the public key and private key for production support. Application Project R11AN0567 includes procedures to create customized key pair preparation. Refer to R11AN0567 to create customized key pairs.

### 4.1.4 Setting up the Application Authentication Signature Type

There are three signature types supported in FSP, as shown below. Open the **Property** page of stack **MCUboot** > **Common** > **Signing and Encryption Options** to look at the signing options. In this example implementation, ECDSA P-256 is used for all the example bootloaders demonstrated in section 3.

#### 4.1.5 Add MCUboot Initialization Code

Follow the steps below to add the MCUboot activation code and compile the bootloader:

1. Add the source code and compile the bootloader.

Follow the steps below to add the source code to the bootloader project and compile the project.

- **Open** hal entry.c.
- Open Developer Assistance.
- Go to HAL/Common > MCUboot > Quick Setup. Drag Call Quick Setup to the top of the hal\_entry.c file before the hal\_entry() function call.
- Call this function at the top of the hal\_entry() function
  - mcuboot\_quick\_setup();

#### Notes on the ${\tt mcuboot\_quick\_setup}$ function

- The main functionality established in the bootloader project is established by the function mcuboot\_quick\_setup, which performs the following functions:
  - The boot\_go function does most of the functions of a bootloader except the final step of jumping to the main image. This function returns a structure pointer (rsp for return structure pointer) from which the image is booted.
  - The RM\_MCUBOOT\_PORT\_BootApp function cleans up resources used by the bootloader and jumps to the application image.
- 2. Compile the bootloader project.
  - Save the project (save the source code and the configuration.xml file), click Generate Project Content and then compile the project.



#### 5. Using the Bootloader with Applications

A set of existing non-bootloader-based projects is used to demonstrate how to configure existing application projects to use the bootloader. General guidelines are also provided for adapting to other existing applications. Unzip example\_projects\_no\_bootloader.zip.

These projects have the same functionality as the projects demonstrated in section 3.3 except these projects are not configured to use the bootloader. Follow the steps below to configure the standalone application projects to use the bootloader and sign the application.

#### 5.1.1 Import the Standalone Application Projects

Import the RA8M1 standalone example project to the same workspace as the bootloader project you created in the previous section. In this section, we will update these existing projects to use the bootloader created in the previous section.

example_projects_no_bootloader >	
Name	
app_ra8m1_ns_secondary	
app_ra8m1_ns_primary	
app_ra8m1_s_secondary	
app_ra8m1_s_primary	

#### Figure 67. Standalone Example Projects for RA8M1 with No Bootloader support

#### 5.1.2 Configure the Application Projects to Use the Bootloader

We will now alter the project **Properties** configuration to allow it to use the bootloader. Right-click on the app\_ra8m1\_s\_primary folder in the Project Explorer and select **Properties**. Select **C/C++ Build>Build Variables**, click **Add**, set the **Variable name** to **BootloaderDataFile**, and check the **Apply to all configurations** box. Change the **Type** to **File** and enter

**\${workspace\_loc:ra\_mcuboot\_ra8m1}/Debug/ra\_mcuboot\_ra8m1.bld** for the value. Click **OK** to save the changes.

builders	Configuration Debug (Asting)	N
$\sim C/C++$ Build	Configuration: Debug [ Active ]	Manage Configurations
Build Variables		
Environment		
Logging	Name Type Value	Add
Settings Teol Chain Editor	Edit Evicting Build Variable	
> C/C++ General		Edit
Project Natures	Variable name BootloaderDataFile v	Delete
Project References	Type: File Y	
Renesas QE		
Run/Debug Settings	Value: e_loc:ra_mcuboot_ra8m1}/Debug/ra_mcuboot_ra8m1.bld Browse	
Task Tags		
> Validation		
	OK Cancel	
	Show system variables	
	Build Variables are IDE only variables, which can be used for string substitution when defining external builde	r configuration, such as
	environment variable value or command line parameter in form of \${VAR}, internal builder may use them dire	ectly.
	Porto	a Dafaulta – Apply
	Resto	e Delaults Apply

Figure 68. Configure the Build Variable to Use the Bootloader



Follow the same procedure and settings as shown in Figure 68 to configure the other three projects:

app\_ra8m1\_ns\_primary

Г

- app\_ra8m1\_s\_secondary
- app\_ra8m1\_ns\_secondary

# 5.2 Signing the Existing Application Projects to Use the Bootloader

The signing command for the application image will be automatically generated when the bootloader is compiled. In the **Project Explorer**, navigate to the <boot\_project > debug > <boot\_project > .bld file. The signing command is under the section <image >.

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

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. If there is signature verification, then it is necessary to set the Environment Variable: MCUBOOT\_IMAGE\_SIGNING\_KEY.

type filter text X	Environment			<	> + <> + §
<ul> <li>Resource Builders</li> <li>C/C++ Build Build Variables Environment</li> </ul>	Configuration: Debug	J [ Active ]		<ul> <li>✓ Manage Co</li> </ul>	onfigurations
Logging	Environment variables	to set			Add
Settings Tool Chain Editor > C/C++ General Project Natures Project References Renesas QE Run/Debug Settings Task Tags > Validation	Variable CWD GCC_VERSION PATH PWD TCINSTALL TC_VERSION	Value C:\Users\ 13.2.1 C:\e2_04_2024\toolchain C:\Users\ C:\e2_04_2024\toolchain 13.2.1.arm-13-7	Origin BUILD SYSTEM BUILD SYSTEM BUILD SYSTEM BUILD SYSTEM BUILD SYSTEM		Select Edit Delete Undefine
	Append variables to     Replace native envi	o native environment ironment with specified one		Restore Defaults	Apply

Figure 69. Add New Environment Variable

Add the Environment variable for the application image version.

🕲 New variable	×
Name: MCUBOOT_IMAGE_VERSION	
Value: 1.0.0	Variables
Add to all configurations	
	OK Cancel

Figure 70. Add MCUBOOT\_IMAGE\_VERSION Variable

Add an Environment variable to configure the application image signing key.



Renesas RA Family

New variable	×
Name: MCUBOOT_IMAGE_SIGNING_KEY	
Value: a8m1}/ra/mcu-tools/MCUboot/root-ec-p256.pem	Variables
Add to all configurations	
ОК	Cancel

Figure 71. Add MCUBOOT\_IMAGE\_SIGNING\_KEY Variable

type filter text	Environment					() ▼ () ▼
> Resource						
Builders	Configuration: Debug [Ac	tive 1		~	Manage Co	onfigurations
V C/C++ Build	configuration. Debug [//c	uve j			manage ee	Jingurations.
Environment						
Logging	Environment variables to set	+				
Settings						Add
Tool Chain Editor	Variable	value		Origin		Select
> C/C++ General		C:\Users\	\Documents\example_projects_with_bootloader\ra8m1_o	BUILD S	YSTEM	Edit
Project Natures		13.2.1	meuhaat ra?m1)/ra/meu-tools/MCI boot/root-ac-p256 pam	BUILD S		Cortan
Project References	MCUBOOT IMAGE VER	100		USER: CO	ONFIG	Delete
Run/Debug Settings	PATH	C:\e2 04 2024\too	Ichains\gcc_arm\13.2.rel1\bin\:\${renesas.build.utilsPath}:C:/e2_0	BUILD S	YSTEM	Undefine
Task Tags	PWD	C:\Users\	\Documents\example_projects_with_bootloader\ra8m1_o	BUILD S	YSTEM	
> Validation	TCINSTALL	C:\e2_04_2024\too	Ichains\gcc_arm\13.2.rel1\	BUILD S	YSTEM	
	TC_VERSION	13.2.1.arm-13-7		BUILD S	YSTEM	
	Append variables to nation	ve environment				
	Replace native environm	ent with specified or	ne -			
				Restore	e Defaults	Apply

Figure 72. Configure the Signing Key and Application Version

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

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

To be able to always recompile the project when the environment variables or the linker script are updated, we recommend adding a **Pre-build step** to always delete the .elf file, as shown in Figure 73.

type filter text	Settings
<ul> <li>▶ Resource</li> <li>Builders</li> <li>✓ C/C++ Build</li> <li>Build Variables</li> </ul>	Configuration: Debug [ Active ]
Environment Logging Settings Tool Chain Editor	<ul> <li>Tool Settings</li> <li>Toolchain</li> <li>Build Steps</li> <li>Pre-build steps</li> <li>Command(s):</li> </ul>

Figure 73. Configure the Pre-build Command



Follow the same procedure to configure the other three projects:

- app\_ra8m1\_ns\_primary
- app\_ra8m1\_s\_secondary
- app\_ra8m1\_ns\_secondary

# 5.2.1 Click Generate Project Content and Compile All Four Application Projects

For both Primary and Secondary applications, compile the Secure application first and then the Non-Secure application.

# 5.2.2 Configure the Debug Configuration

1. Open the **Debug Configurations**: app\_ra8m1\_s\_primary > Debug As > Debug Configurations Make sure that app\_ra8m1\_s\_primary Debug is selected and select the Startup tab.

] 🖻 🐢 🗎 🗶 🖻 🏹 🗸	Name: app_ra8m1_s_primary De	bug			
ype filter text © C/C++ Application © C/C++ Remote Application @ EASE Script © GDB Hardware Debugging © GDB Simulator Debugging (RH850) @ Launch Group © Renesas GDB Hardware Debugging © app_ra8m1_ns_primary Debug_SSD © app_ra8m1_ns_secondary Debug_SSD © app_ra8m1_s_primary Debug © Renesas Simulator Debugging (RX, RL78)	Main State Debugger State Initialization Commands Reset and Delay (seconds): Halt Load image and symbols Filename Program Binary [app_ra	Common Common Load type Image and Symbols	Source Source Offset (hex)	On connect Yes	Add
					Remove Move up Move down
	Runtime Options          Runtime Options         Set program counter at (hex         Set breakpoint at:         Resume         Run Commands	): main			

Figure 74. Configure the Primary Secure Project Debug Startup



2. Set up the **Debug Configurations**.

Click Add... and then Workspace. Navigate to the ra\_mcuboot\_ra8m1 project and select the ra\_mcuboot\_ra8m1.elf file from the debug folder. Click OK.

,		) ( r
type filter text C/C++ Application C/C++ Application G C/C++ Remote Application E ASE Script C GDB Hardware Debugging G GDB Simulator Debugging (RH850) C Renesas GDB Hardware Debugging C app_ra8m1_ns_primary Debug_SSD C app_ra8m1_ns_primary Debug C app_ra8m1_s_primary Debug C Renesas Simulator Debugging (RX, RL78)	Name:       app_ra8m1_s_primary Debug         Main * Debugge       Startup         Initialization Commands         Reset and Delay (seconds):         3         Hait         Specify download module         \$ f(workspace_loc\ra_mcuboot_ra8m1)\Debug (ra_mcuboot_ra8m1.elf)         Variables         Search Project         Workspace         File System         OK         Cancel         Runtime Options         Set program counter at (hex):         Set breakpoint at:         main         Resume         Run Commands	Add Edit Remove Move up Move down
Filter matched 12 of 14 items	Rever	t Apply

Figure 75. Add the Bootloader Project

Click Add again and add the app\_ra8m1\_ns\_primary project binary app\_ra8m1\_ns\_primary.elf as in the prior step. Click OK.

Add download module	×
Specify download module name: pace_loc:\app_ra8m1_ns_primary\Deb	ug <mark>.app_ra8m1_ns_primary.elf}</mark>
Variables Search Project	OK Cancel

Figure 76. Add the Non-Secure Project



Change the load type of the Program Binaries for the **app\_ra8m1\_ns\_primary** and **app\_ra8m1\_s\_primary** to **Symbols only** by clicking on the cell for load type and selecting **Symbols only** from the drop-down menu.

✓ Program Binary [app_ra8m1_s_prim       Symbols only       Yes       Add         ✓ ra_mcuboot_ra8m1.elf [C:\Users\tru       Image and Symbols       0       Yes       Edit         ✓ app_ra8m1_ns_primary.elf [C:\Users\       Symbols only       0       Yes       Move up         Move up       Move down       Move down       Move down         Runtime Options	✓ Program Binary [app_ra8m1_s_prim       Symbols only       Yes       Iddu         ✓ ra_mcuboot_ra8m1.elf [C:\Users\tru       Image and Symbols       0       Yes       Edit         ✓ app_ra8m1_ns_primary.elf [C:\Users\       Symbols only       0       Yes       Move up         Move up       Move down       Move down       Move down         Runtime Options
✓ ra_mcuboot_ra8m1.elf [C\Users\tru       Image and Symbols       0       Yes       Edit         ✓ app_ra8m1_ns_primary.elf [C\Users\       Symbols only       0       Yes       Move up         Move up       Move down       Move down       Move down       Move down         Runtime Options	✓ ra_mcuboot_ra&m1.elf [C:\Users\tru       Image and Symbols       0       Yes       Edit         ✓ app_ra&m1_ns_primary.elf [C:\Users\       Symbols only       0       Yes       Move up         Move up       Move down       Move down       Move down       Move down         Runtime Options
app_ra8m1_ns_primary.elf [C:\Users\ Symbols only 0 Yes Remove   Move up Move up   Move down    Runtime Options   Set program counter at (hex):   Set breakpoint at:   main   Resume	☑ app_ra8m1_ns_primary.elf [C:\Users\ Symbols only       0       Yes       Remove         Move up       Move up       Move down         Runtime Options       Set program counter at (hex):
Move up   Move down     Runtime Options     Set program counter at (hex):     Set breakpoint at:     main     Resume	Move up       Move down
Runtime Options   Set program counter at (hex):   Set breakpoint at:   main   Resume	Runtime Options Set program counter at (hex): Set breakpoint at: main
Runtime Options         Set program counter at (hex):         Set breakpoint at:         main         Resume	Runtime Options
	Resume

#### Figure 77. Select to load Symbols only for the Secure and Non-Secure Project

3. Add the signed binary image to the download options using the Raw Binary Load type.

✓ Program Binary [app_ra8m1_s_prim       Symbols only       Yes         ✓ ra_mcuboot_ra8m1.elf [C:\Users\tru       Image and Symbols       0       Yes         ✓ app_ra8m1_ns_primary.elf [C:\Users       Symbols only       0       Yes         ✓ app_ra8m1_s_primary.bin.signed [C:       Raw Binary       2020000       Yes         ✓ app_ra8m1_ns_primary.bin.signed [L:       Raw Binary       12030000       Yes	Filename	Load type	Offset (hex)	On connect
✓ ra_mcuboot_ra8m1.elf [C:\Users\tru       Image and Symbols       0       Yes         ✓ app_ra8m1_ns_primary.elf [C:\Users       Symbols only       0       Yes         ✓ app_ra8m1_s_primary.bin.signed [C:       Raw Binary       2020000       Yes         ✓ app_ra8m1_ns_primary.bin.signed [C:       Raw Binary       12030000       Yes	Program Binary [app_ra8m1_s_prim	Symbols only		Yes
✓ app_ra8m1_ns_primary.elf [C:\Users         Symbols only         0         Yes           ✓ app_ra8m1_s_primary.bin.signed [C:         Raw Binary         2020000         Yes           ✓ app_ra8m1_ns_primary.bin.signed [C:         Raw Binary         12030000         Yes	✓ ra_mcuboot_ra8m1.elf [C:\Users\tru	Image and Symbols	0	Yes
app_ra8m1_s_primary.bin.signed [C Raw Binary 2020000 Yes     app ra8m1 ns primary.bin.signed [ Raw Binary 12030000 Yes	app_ra8m1_ns_primary.elf [C:\Users	Symbols only	0	Yes
app ra8m1 ns primary,bin.signed [ Raw Binary 12030000 Yes	app_ra8m1_s_primary.bin.signed [C:	Raw Binary	2020000	Yes
	app_ra8m1_ns_primary.bin.signed [	Raw Binary	12030000	Yes

Figure 78. Load the Signed Images

Note that for different update modes and different application images, the load address needs to be updated. For the example projects included in this application project, you can reference the memory configuration images included in Figure 60 to Figure 64 to set up the load address.

4. After the above is set up, follow section 3.3 to run the projects if Overwrite Update mode is used or follow section 3.4 to run the projects if Swap Update mode is used.



### 5.3 Mastering and Delivering a New Application

Mastering and delivering a new application involves similar steps described above in section 4.2 and section 5.2. Typically, the following aspects must be considered in the design of delivering new applications:

- 1. Create the new application and sign the latest application by following the steps below:
  - A. Refer to the Renesas RA Family Security Design with Arm<sup>®</sup> TrustZone<sup>® -</sup> IP Protection for new project creation with TrustZone support.
  - B. Refer to section 4 to configure the new application to use the bootloader and sign the new application.
- 2. Download the new application to the Secondary slots.

This step varies based on the downloading method selected by each user. In this application project, the Ancillary file download capability from e<sup>2</sup> studio is used for demonstration purposes. You can use this method as a testing tool when developing a customized new image downloader. Application Projects R11AN0570 and R11AN0576 include image downloader examples using XModem over COM port. They can be used as references.

# 5.4 Customize the Bootloader to other MCUs

When customizing to other MCUs, a recreation of the bootloader project is recommended. Changing the board BSP selection is not a recommended path. Recreating the bootloader project is needed, particularly when moving to a different FSP version.

- Users need to adjust the memory configuration based on the bootloader size and the application size (refer to).
- When customizing to other MCUs using the same FSP version, users can use the stack export and import functionality to save some steps in the bootloader project recreation.
  - Note that there can be errors after exporting and importing the stack; the user needs to review the imported stack by comparing it with the original bootloader to correct any errors.

To export a stack, right-click on the module, select Export, and save the stack with .xml as the file extension.

(1)			Team	>		
• · · · · · · · · · · · · · · · · · · ·	1		Resource Configurations	>		
MCUboot Port for RA (rm_mcuboot_port)			Validate		Add [Optional] Add Logging Support	Keys (NO
-		ď	Cut	Ctrl+X		PRODUCT
(i)			Сору	Ctrl+C		<b>(i)</b>
I	<b>A</b>	Ē	Paste	Ctrl+V		
HbedTLS (Crypto Only)	g_flash0	×	Delete	Delete		
	(r_tiasn_	ľ	Non-secure Callable			
<ol> <li>(i)</li> </ol>	í	è	Import			
▲ 		ß	Export 🔓			
Mbed Crypto H/W Add Persistent Storage		í	Module Resources			
Acceleration on LittleFS (Optional)		0	Run As	>		
(rm_psa_crypto)		*	Debug As	>		
▲			Compare With	>		
4. SCE Compatibility	_		Replace With	>		
Mode		-			7	
(i)						
Add Key Injection for						
Add Key Injection for PSA Crypto (Optional)						

#### Figure 79. Export a Stack

To import a stack, right-click in the open area of the Stack window, select Import, and choose the exported .xml file.





Figure 80. Import a Stack

# 6. Appendix

# 6.1 Making the Bootloader for Cortex-M85 Immutable

To make the bootloader immutable, you must lock the flash blocks containing the bootloader from being programmed and erased.

The RA8M1 features two sets of registers that facilitate flash block locking. Block Protect Setting (BPS) Registers feature bits that map to individual flash blocks. When a bit is set to zero, the corresponding flash block cannot be erased or programmed. The Permanent Block Protect Setting (PBPS) Registers have a similar bit mapping to flash blocks. When a bit is set in one of these registers, the corresponding flash block is permanently locked from being erased and programmed if the same bit in the Block Protect Setting Register is also cleared to zero. This process is irreversible. Once a flash block is permanently locked, it cannot be unlocked again.

Based on the example bootloaders provided in this application project, the flash blocks used by the bootloader are:

- RA8M1 Overwrite Mode: block 0-7
- RA8M1 Swap Mode: block 0-8

Refer to the Renesas RA Family MCU Securing Data at Rest Using the Arm<sup>®</sup> TrustZone<sup>®</sup> Application Project to understand the operational flow of setting up the Flash Block Protection.

Note that ticking the BSP0 and PBPS0 Flash Block settings will permanently lock the flash blocks. This CANNOT be reversed. Further details can be found in sections 6.2.9 and 6.2.10 of the RA8M1 Hardware User's Manual.

# 6.2 Making the Bootloader for Cortex-M33 Immutable

Refer to the Renesas RA Family MCU Securing Data at Rest Using the Arm<sup>®</sup> TrustZone<sup>®</sup> Application Project to understand the operational flow of setting up the Flash Block Protection.

### 6.3 Making the Bootloader for Cortex-M4 Immutable

Refer to the *Renesas RA MCU Family Securing Data at Rest Utilizing the Renesas Security MPU* application project section Permanent Locking of the Flash Access Window (FAW) Region to understand how to make the bootloader for Cortex-M4 Immutable. Section PC Application to Permanently Lock the FAW in the same application note describes how to handle Flash locking in production mode.



# 6.4 Device Lifecycle Management for Renesas RA Cortex-M85 MCUs

Once the bootloader development is finished, you can transition the Device Lifecycle State of the RA Cortex-M85 MCU to lock down the debugger and the serial programming interface.

We recommend referring to the Device Lifecycle State Transitions in the Production Flow section in the *Renesas RA Family Device Lifecycle Management Key Injection Application Note* to understand the device lifecycle management options during production.

The operational overview of how to use Renesas Flash Programmer to perform these transitions is explained in the Overview of Device Lifecycle State Transitions using the Renesas Flash Programmer section.

#### 6.5 Device Lifecycle Management for Renesas RA Cortex-M33 MCUs

Once the bootloader development is finished, you can transition the Device Lifecycle State of the RA Cortex-M33 MCU to lock down the debugger and the serial programming interface.

We recommend referring to the Device Lifecycle State Transitions in the Production Flow section in the *Renesas RA Family Device Lifecycle Management Key Injection Application Note* to understand the device lifecycle management options during production.

The operational overview of how to use Renesas Flash Programmer to perform these transitions is explained in the Overview of Device Lifecycle State Transitions using the Renesas Flash Programmer section.

#### 6.6 Device Lifecycle Management for Renesas RA Cortex-M4 MCUs

Once the bootloader development is finished, set up the ID Code protection on the Renesas RA Cortex-M4 MCU to lock down the debugger and the serial programming interface.

You can refer to the *Securing Data at Rest Utilizing the Renesas Security MPU Application Project* section Setting up the Security Control for Debugging for the desired setting to control the device lifecycle management of the RA Cortex-M4 MCUs using the ID Code protection method.

#### 7. References

- Renesas RA Family Securing Data at Rest Using the Arm<sup>®</sup> TrustZone<sup>®</sup> (R11AN0468)
- Renesas RA Family Device Lifecycle Management for RA8 MCUs (R11AN0785)
- Renesas RA Family Security Design with RA8 MCU using Arm® TrustZone® IP Protection (R11AN0897)
- Renesas RA Family Secure Bootloader for RA2 MCU Series (R11AN0516)
- Renesas RA Family RA6 Secure Firmware Update using MCUboot and Flash Dual Bank (R11AN0570)
- Renesas RA Family RA6 Booting Encrypted Image using MCUboot and QSPI (R11AN0567)

### 8. Website and Support

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

EK-RA8M1 Resources RA Product Information Flexible Software Package (FSP) RA Product Support Forum Renesas Support renesas.com/ra/ek-ra8m1 renesas.com/ra renesas.com/ra/fsp renesas.com/ra/forum renesas.com/support



# **Revision History**

		Description	
Rev.	Date	Page	Summary
1.00	Sep.06.24	-	First release document.



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

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

#### 1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

#### 6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

#### Notice

- 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.
- Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
- 3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
- 4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
- 5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
- 6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

- 7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
- 8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
- 12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
   Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.
- (Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
- (Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

# **Corporate Headquarters**

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

### Trademarks

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

# Contact information

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