

Renesas RA Family

Arm® Secure Boot Solution for RA6M3 MCU Group

Introduction

This application note provides a detailed overview of the Arm Trusted Firmware M (TF-M) Secure Boot solution ported for Renesas RA6 Family Arm® Cortex®-M4 based MCU devices. The reference solution provided in this package is targeted for use cases where security is a critical component in the application design. This reference solution showcases how an application image is signature-authenticated and validated before being allowed to execute.

Upon completion of this application note, the user will gain a better understanding of the implementation of the Arm secure boot solution provided along with various boot options that can be selected by the user.

Currently, the secure boot reference solution is implemented and tested on the EK-RA6M3 kit.

Required Resources

To build and run the secure boot reference solution, you need the following resources:

Tools and Software

- e² studio ISDE v7.6.0 or later
- RA Family Flexible Software Package (FSP) v1.0.0 or later.
- SEGGER J-link® USB driver

Hardware

- Renesas EK-RA6M3 kit
- PC running Windows® 7 or 10; terminal emulation program (Tera Term or similar application)
- Micro USB cable USB-UART TTL cable

Prerequisites and Intended Audience

This application note assumes that the user has some experience with the Renesas e² studio ISDE and Flexible Software Package (FSP) for the Renesas RA Family. Before performing the procedures in this application note, follow the procedure in the FSP User's Manual to build and run the Blinky project. Doing so enables you to become familiar with the e² studio and the FSP and validates that the debug connection to your board functions properly.

The intended audience for this application note is users who want to use the Arm Secure Boot solution with the Renesas RA6M3 MCU Series.

Contents

1. Overview	3
1.1 Features.....	3
2. Arm® Secure Boot Implementation	4
2.1 Operational Flow	4
2.2 Memory Layout.....	5
2.2.1 TLV (Type-Length-Value) Section.....	6
2.2.2 Image Trailer	6
2.3 Application Image Integrity Check.....	7
2.4 Application Image Authentication	8
2.5 Security Counter	8
2.6 Firmware Upgrade Operation.....	8
2.6.1 Overwrite Operation	8
2.6.2 Swapping Operation.....	8
2.6.3 Non-Swapping Operation	9
2.6.4 RAM Loading.....	9
3. Build Time Configuration	9
4. PC Tools.....	10
4.1 Image Creation	10
4.1.1 Sign.bat	10
4.1.2 Imgtool.py	10
4.2 Dependency Installation	11
4.3 Downloader	11
4.4 Secure MPU and FAW Reset.....	11
5. RA Hardware Security Features	11
5.1 Secure MPU	11
5.2 Flash Access Window	12
5.3 FSPR (One Time Programmable Setting).....	12
6. Arm Secure Boot Reference Solution	12
6.1 Importing, Building the Project	13
6.2 Powering up the Board.....	13
6.3 Build Time Configuration Settings	13
6.4 Running the Demo.....	14
6.5 Creating and Signing Application	15
6.5.1 Application Image Creation	15
6.5.2 Application Image Signing.....	15
6.6 Known Issues and Limitations.....	16
Revision History	18

1. Overview

For secure devices, it is security-critical to enforce firmware authenticity to protect against execution of malicious software. This is implemented by building a “trust chain” where each step in the execution chain authenticates the next step before execution. The chain of trust is based on a “Root of Trust,” which is implemented using asymmetric cryptography. The Root of Trust in this case, is established through the combination of an immutable bootloader and a public key (ROTPK).

The bootloader runs when the CPU is released from reset. It runs in Secure mode. It authenticates the firmware image by performing a hash (SHA-256) and digital signature (RSA-2048) validation. The public key that is used for this can be built into the bootloader image or can be provisioned during manufacturing. Metadata of the image is delivered together with the image itself in a header and trailer section. In the case of successful authentication, the bootloader passes execution to the validated image. Execution never returns to the bootloader until the next reset.

1.1 Features

The Arm® Secure Boot reference solution currently supports the following features:

- Supported upgrade methods, configurable by user at build time:
 - Image swapping
 - No swap
 - Image overwrites
 - Execution from RAM
- Supported application authentication mechanism:
 - RSA 2048/3072
- Supported application integrity check mechanism:
 - SHA-256
- Rollback protection
- Fallback mechanism to stable version
- Integrity check of application in slot 0 on every boot, configurable at build time
- Update non-secure and secure application images separately using different keys for authentication.

2. Arm® Secure Boot Implementation

2.1 Operational Flow

This section describes the secure boot process of an RA MCU on power-up when using the Arm Secure Boot reference solution. Note that this flow shows the overwrite and swap option only.

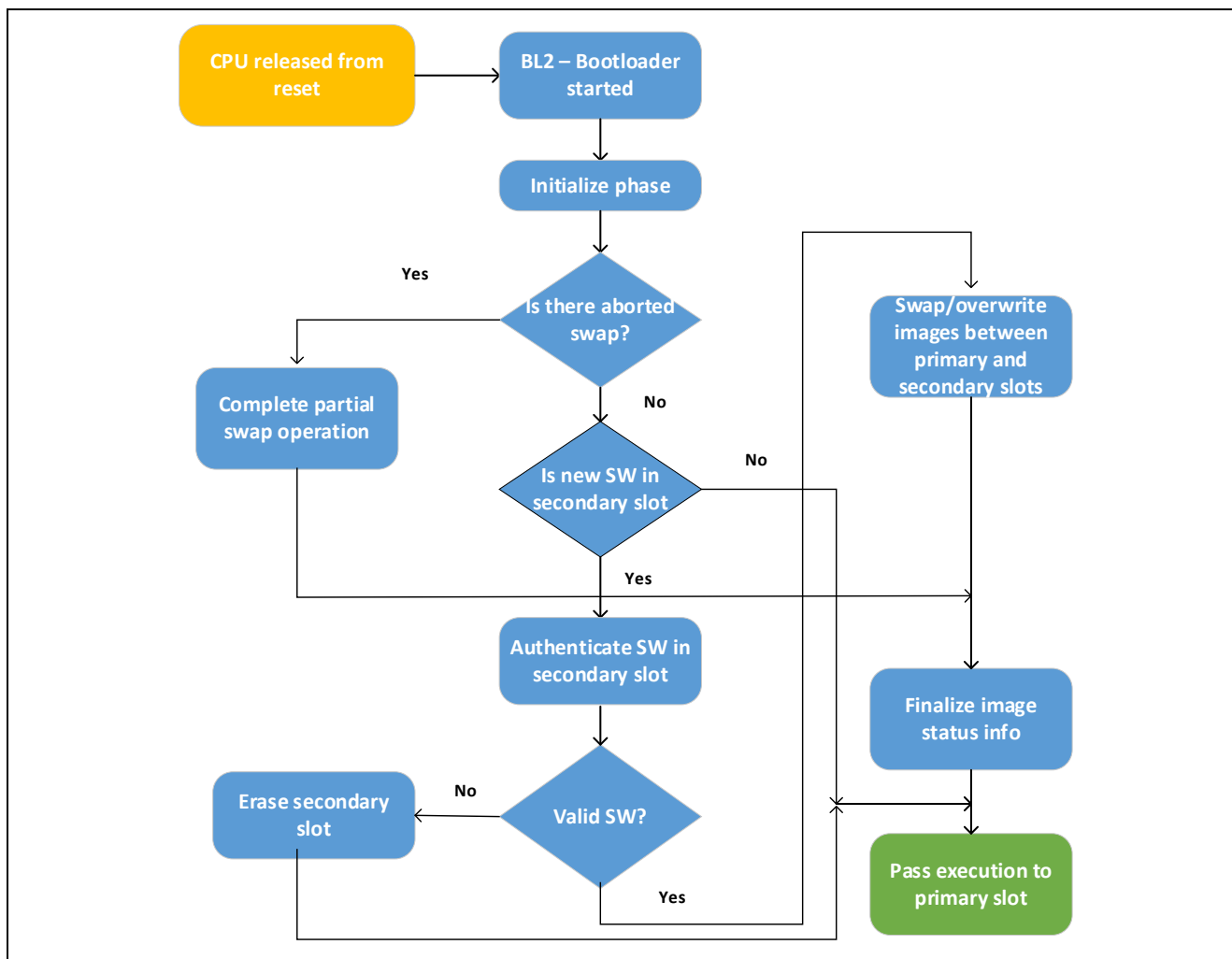


Figure 1. Operational Flow

Procedure

1. Inspect swap status region. Is an interrupted swap being resumed?
 - Yes: Complete the partial swap operation; skip to step 3.
 - No: Proceed to step 2.
2. Inspect image trailers. Is a swap requested?
 - Yes:
 - A. Is the requested image valid (integrity and security check)?
 - Yes.
 - a. Perform swap operation.
 - b. Persist completion of swap procedure to image trailers.
 - c. Proceed to step 3.
 - No.
 - a. Erase invalid image.

b. Persist failure of swap procedure to image trailers.

c. Proceed to step 3.

— No: Proceed to step 3.

3. Boot into image in primary slot.

2.2 Memory Layout

This section provides the detailed memory layout of code flash used in our reference Arm Secure Boot implementation. The memory layout can be customized depending upon the Arm core (M4/M33) and also based on the firmware upgrade mechanism chosen by the end user.

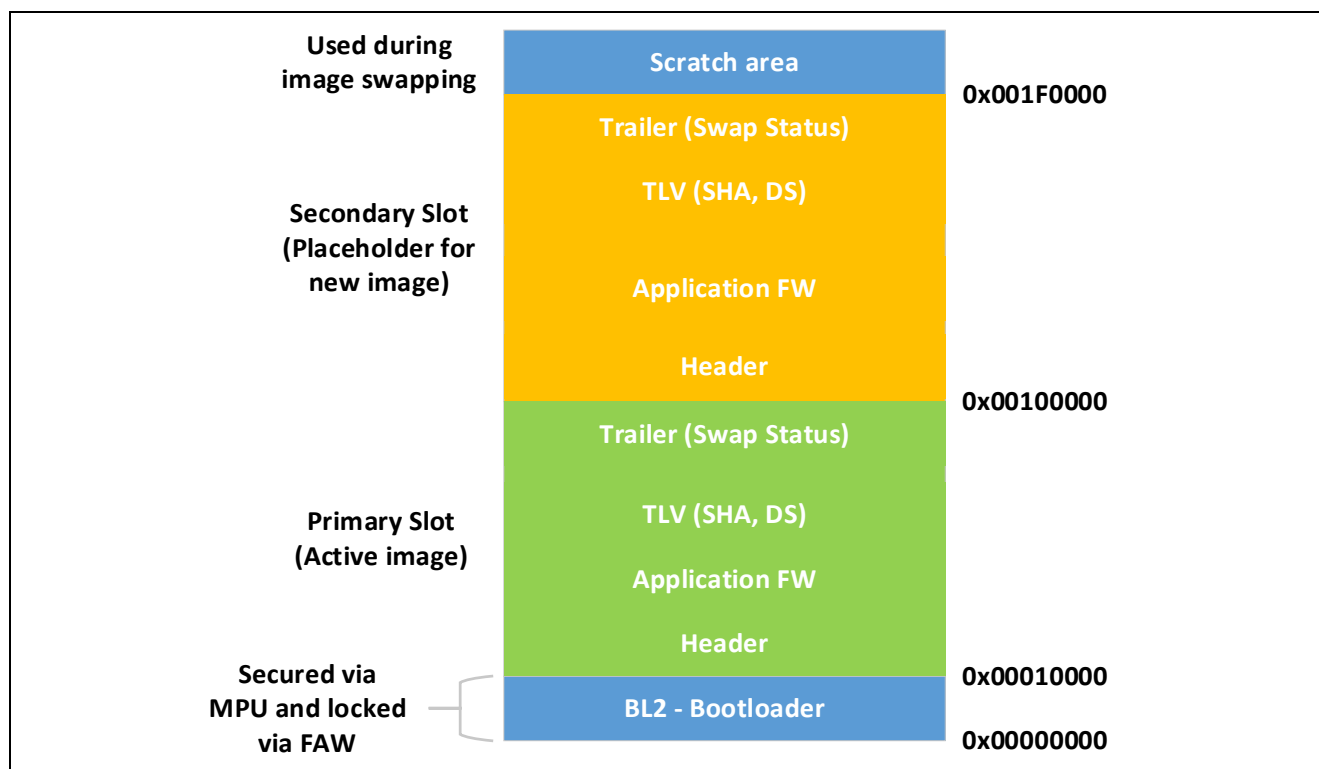


Figure 2. Arm Secure Boot Code Flash Memory Layout

The Flash memory is partitioned in 4 regions:

1. Bootloader section
2. Primary slot
3. Secondary slot
4. Scratch area

The bootloader supports swap or overwrite or non-swap or RAM loading image upgrades but must be configured at build time.

In addition to the slots of image areas, the bootloader requires a scratch area for the image swap operation to allow for reliable image swapping. The scratch area must be large enough to store at least the largest sector that is going to be swapped. The scratch area is only ever used when swapping firmware, which means only when doing an upgrade. An important consideration for opting to use a larger size for the scratch area is to ensure that the flash memory wears out evenly. By using a swap area that is large enough to hold two or more sectors, the number of erase-write cycles on each sector is reduced.

Image header is prepended to the beginning of the image. It is integrity protected.

```

struct image_header {
    uint32_t  ih_load_addr;
    uint16_t  ih_hdr_size;
    uint16_t  _pad1;
    uint32_t  ih_img_size;
    uint32_t  ih_flags;
    struct image_version  ih_ver;
    uint32_t  _pad2;
};

```

2.2.1 TLV (Type-Length-Value) Section

Image metadata is appended to the end of the image. It is not integrity protected. It contains following:

1. Hash of public key
2. Hash of image
3. Signature of image

2.2.2 Image Trailer

The bootloader uses metadata information stored in the image flash areas to determine the current state of an update process and activities to perform during the current boot operation. This metadata is located at the end of the image flash areas and is called an image trailer. An image trailer has the following structure.

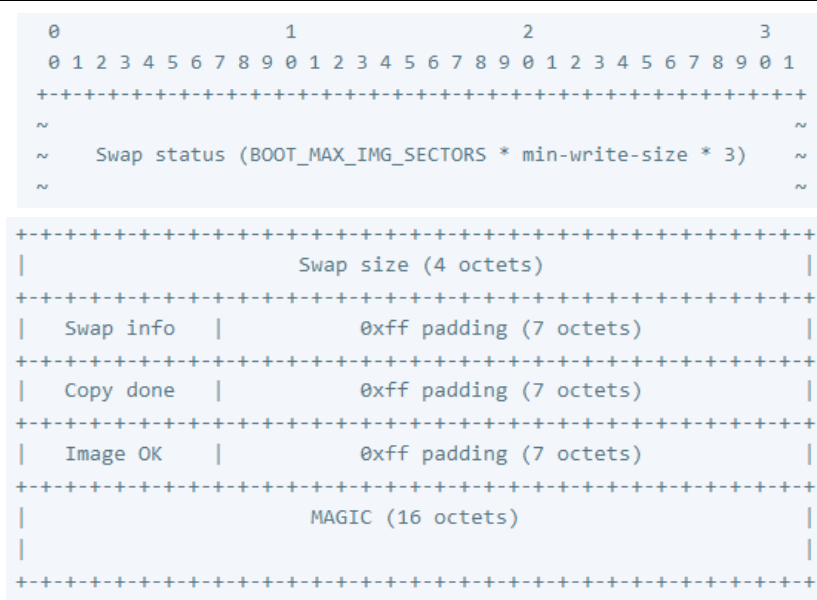


Figure 3. Image Trailer Structure

An image trailer contains the following fields:

Swap status

This is a series of records which records the progress of an image swap. To swap entire images, data are swapped between the two image areas one or more sectors at a time, using the following steps:

- Sector data in the primary slot is copied into scratch, then erased
- Sector data in the secondary slot is copied into the primary slot, then erased
- Sector data in scratch is copied into the secondary slot

As it swaps images, the bootloader updates the swap status field in a way that allows it to compute how far this swap operation has progressed for each sector. The swap status field can thus be used to resume a swap operation if the bootloader is halted while a swap operation was ongoing and later reset. The BOOT_MAX_IMG_SECTORS value is the maximum number of sectors the bootloader supports for each image; its default value is 128. This value can be decreased to limit RAM usage, or increased for devices

that have massive amounts of Flash or very small sized sectors, thereby requiring a bigger configuration to allow for the handling of all slot's sectors. The factor of min-write-sz is due to the behavior of flash hardware.

Swap size

At the beginning of a new swap operation, the total size that needs to be swapped (based on the slot with largest image + TLVs) is written to this location for easier recovery in case a reset occurs while performing the swap.

Swap Info

A single byte which encodes the following information:

- **Swap type:** Stored in bits 0-3. Indicating the type of swap operation in progress. When bootloader resumes an interrupted swap, it uses this field to determine the type of operation to perform. This field contains one of the following values in the table below.

Name	Value
BOOT_SWAP_TYPE_TEST	2
BOOT_SWAP_TYPE_PERM	3
BOOT_SWAP_TYPE_REVERT	4

Figure 4. Swap Info

- **Image number:** Stored in bits 4-7. It always has 0 value at single image boot. In case of a multi-image boot, it indicates which image was swapped when the interrupt happened. The same scratch area is used for all image swap operations. Therefore, this field is used to determine which image the trailer belongs to if boot status is found on scratch area when the swap operation is resumed.

Copy done

A single byte indicating whether the image in this slot is complete (0x01=done; 0xff=not done).

Image OK

A single byte indicating whether the image in this slot has been confirmed as working by the user (0x01=confirmed; 0xff=not confirmed).

MAGIC

The following 16 bytes, written in host-byte-order:

```
const uint32_t boot_img_magic[4] = {  
    0xf395c277,  
    0x7fed260,  
    0x0f505235,  
    0x8079b62c,  
};
```

2.3 Application Image Integrity Check

An image is checked for integrity immediately before it gets copied into the primary slot. If the bootloader does not perform an image swap, it can perform an optional integrity check of the image in the primary slot if MCUBOOT_VALIDATE_PRIMARY_SLOT is set; otherwise it does not perform an integrity check.

During the integrity check, the bootloader verifies the following aspects of an image:

- 32-bit magic number must be correct (IMAGE_MAGIC).
- Image must contain all fields defined in the image_tlv_info struct, identified by its magic value (IMAGE_TLV_PROT_INFO_MAGIC or IMAGE_TLV_INFO_MAGIC) exactly following the firmware (hdr_size + img_size). If IMAGE_TLV_PROT_INFO_MAGIC is found, then, after ih_protect_tlv_size bytes, another image_tlv_info with magic value equal to IMAGE_TLV_INFO_MAGIC must be present.
- Image must contain a SHA-256 TLV.
- Calculated SHA-256 must match SHA-256 TLV contents.
- Image may contain a signature TLV. If it does, it must also have a KEYHASH TLV with the hash of the key that was used to sign. The list of keys will then be iterated over, looking for the matching key, which will then be used to verify the image contents.

2.4 Application Image Authentication

The final step of the integrity check process is signature verification. The bootloader can have one or more public keys embedded as part of it at build time. During signature verification, the bootloader verifies that an image was signed with a private key that corresponds to the embedded KEYHASH TLV.

2.5 Security Counter

Each signed image contains a security count in its manifest. It is used by the bootloader and its aim is to have an independent (from the image version) counter to ensure rollback protection by comparing the new image's security counter against the original (currently active) image's security counter during the image upgrade process. It is added to the manifest (to the TLV area that is appended to the end of the image) by one of the Python scripts, when signing the image. The value of the security counter is security critical data and it is in the integrity protected part of the image. The last valid security counter should always be stored in a non-volatile and trusted component of the device and its value should always be increased if a security flaw was fixed in the current image version.

2.6 Firmware Upgrade Operation

Arm Secure Boot handles only the firmware authenticity check after start-up and the firmware switch part of the firmware update process. Downloading the new version of the firmware is out-of-scope for Arm Secure Boot. It supports three different strategies to switch to the new firmware and it is assumed that firmware images are executed-in-place (XIP).

2.6.1 Overwrite Operation

This operation can be enabled by setting the MCUBOOT_OVERWRITE_ONLY macro as part of build time configuration. This is the default option used in the Arm Secure Boot solution for upgrading the application firmware.

In this configuration, the active image is stored in the primary slot, and this image is always started by the bootloader. Therefore, a developer must ensure that all linkage within the image is restricted to address within the primary slot. If the bootloader finds a valid image in the secondary slot, which is marked for upgrade, then the content of the primary slot will be simply overwritten with the content of the secondary slot, before starting the new image from the primary slot. After the content of the primary slot has been successfully overwritten, the header and trailer of the new image in the secondary slot is erased to prevent the triggering of another unnecessary image upgrade after a restart. The overwrite operation is fail-safe and resistant to power failures.

2.6.2 Swapping Operation

This operation can be enabled by setting the MCUBOOT_SWAP macro as part of build time configuration. This operation is not tested in the released Secure Boot reference solution.

With the swapping image upgrade strategy, the active image is also stored in the primary slot and it will always be started by the bootloader. If the bootloader finds a valid image in the secondary slot, which is marked for upgrade, then contents of the primary slot and the secondary slot will be swapped, before starting the new image from the primary slot. Scratch area is used as a temporary storage place during image swapping. Update mark from the secondary slot is erased when the swapping is successful. The bootloader can revert the swapping as a fallback mechanism to recover the previous working firmware version after a faulty update. The swap operation is fail-safe and resistant to power failures.

After a successful image upgrade, the firmware can mark itself as “OK” at runtime by setting the `image_ok` flag in the flash. When this happens, the swap is made “permanent” and the bootloader will then choose to run it during the next boot.

2.6.3 Non-Swapping Operation

This operation can be enabled by setting the `MCUBOOT_NO_SWAP` macro as part of build time configuration. This operation is not tested in the released secure boot reference solution.

When non-swapping operation is enabled, the active image flag is moved between slots during firmware upgrade. If firmware is executed-in-place (XIP), then two firmware images must be generated. One of them is linked to be executed from the primary slot memory region and the other from the secondary slot. The firmware upgrade client, which downloads the new image, must be aware as to which slot hosts the active firmware, and which acts as a staging area. It is responsible for downloading the proper firmware image. At boot time, Arm Secure Boot inspects the version number in the image header and passes execution to the newer firmware version. The new image must be marked for upgrade, which is automatically done by Python scripts at compile time. Image verification is done the same way in all operational modes. If the new image fails during authentication, then Arm Secure Boot erases the memory slot and starts the other image after successful authentication.

2.6.4 RAM Loading

RAM loading operation can be enabled by setting the `MCUBOOT_RAM_LOADING` macro as part of build time configuration. This operation is not tested in the released secure boot reference solution.

Like the non-swapping mode, this selects the newest image by reading the image version numbers in the image headers, but 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.

Note: Only single image boot is supported with RAM loading upgrade mode.

3. Build Time Configuration

The following description is meant to provide background on various build time configuration settings.

1. **BL2** (default: True)

- **True:** TF-M is built together with the bootloader. Arm Secure Boot is executed after reset and it TF-M and starts secure code. This is the default option used in the Arm Secure Boot solution for upgrading the application firmware.
- **False:** TF-M is built without the bootloader. Secure image is linked to the beginning of the device memory and executed after reset. If set to false, then, using any of the further compile time switches invalid.

2. **MCUBOOT_UPGRADE_STRATEGY** (default: “OVERWRITE_ONLY”)

- **OVERWRITE_ONLY:** Default firmware upgrade operation with overwrite. This is the default option used in the Arm Secure Boot solution for upgrading the application firmware.
- **SWAP:** Activate swapping firmware upgrade operation.
- **NO_SWAP:** Activate non-swapping firmware upgrade operation.
- **RAM_LOADING:** Activate RAM loading firmware upgrade operation, where the latest image is copied to RAM and runs from there instead of being executed in-place.

3. **MCUBOOT_SIGNATURE_TYPE** (default: “RSA-3072”)

- **RSA-3072:** Image is signed with RSA-3072 algorithm
- **RSA-2048:** Image is signed with RSA-2048 algorithm. This is the default option used in the Arm Secure Boot solution for upgrading the application firmware.

4. **MCUBOOT_IMAGE_NUMBER** (default: 2)

1. Single image boot, secure and non-secure images are signed and updated together. This is the default option used in the Arm Secure Boot solution for upgrading the application firmware
2. Multiple image boot, secure and non-secure images are signed and updatable independently.

5. **MCUBOOT_HW_KEY** (default: True)

- **True:** The hash of public key is provisioned to the MCU and the image manifest contains the whole public key. Bootloader validates the key before using it for firmware authentication; it calculates the hash of the public key from the manifest and compares it against the retrieved key-hash from the hardware. Thus, the bootloader is independent from the public key(s). Key(s) can be provisioned any time and by different parties.
- **False:** The whole public key is embedded in the bootloader code and the image manifest contains only the hash of the public key. Bootloader validates the key before using it for firmware authentication; it calculates the hash of built-in public key and compares against the retrieved key-hash from the image manifest. After this, the bootloader can verify that the image was signed with a private key that corresponds to the retrieved key-hash (it can have more public keys embedded in it and it may have to look for the matching one). All the public key(s) must be known at bootloader build time.

This is the default option used in the Arm Secure Boot solution for upgrading the application firmware.

6. **MCUBOOT_LOG_LEVEL** (Default: LOG_LEVEL_INFO)

Can be used to configure the level of logging in Arm Secure bootloader. The possible values are the following:

- LOG_LEVEL_OFF
- LOG_LEVEL_ERROR
- LOG_LEVEL_WARNING
- LOG_LEVEL_INFO
- LOG_LEVEL_DEBUG

Logging can be disabled and thus the code size can be reduced by setting it to LOG_LEVEL_OFF. Its value depends on the build type. If the build type is Debug and a value has been provided, then that value will be used, otherwise it is LOG_LEVEL_INFO by default.

4. **PC Tools**

This section describes the following utilities provided as part of the package. It explains what the functionality of each tool is, and how to use them in the reference project.

4.1 **Image Creation**

The folder `Image_Creation` provided as part of the Secure Boot release package contains python scripts to generate signed application binary. Two important files under this folder are:

4.1.1 **Sign.bat**

The `sign.bat` batch file invokes the `imgtool.py` python file to generate signed application binary. To run these python scripts, we need to install python3. Python3 can be downloaded from the python.org website: <https://www.python.org/>

The following fields need to be updated to generate new application image.

- **Infile**

Set this variable to the path that contains your application project binary file.

- **Python_path**

Set this variable to the python 3 folder in your PC.

4.1.2 **Imgtool.py**

The Python program `scripts/imgtool.py` can be used to perform the operations that are necessary to manage keys and sign images. This Python script is provided by Arm. It is in the `bl2/ext/mcuboot/scripts` directory. Issue the Python `imgtool.py sign --help` command in the directory for more information about the mandatory and optional arguments.

This tool takes an image in binary format and adds a header and trailer that the bootloader is expecting. `sign.bat` run the following command to generate a signed binary image that can be run on Renesas RA MCU devices.

4.2 Dependency Installation

This folder contains `requirements.txt` file which contains the dependency modules that need to be installed in your PC to perform cryptographic operations. These can be installed using the command line below:

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

4.3 Downloader

Bootloader is not responsible for downloading of the application image into the target hardware platform. It is the responsibility of the application to provide a utility to handle this. However, in this release package, Renesas provided reference utilities to program the bootloader, primary/secondary application images in their corresponding memory slots in the RA6M3 MCU Series internal flash memory regions using JTAG interface. Support for additional interfaces such as USB or Wi-Fi can be added by the end user based on their design.

BL2_download.bat

This file downloads and programs the bootloader image in the internal flash at offset 0x0. This file can be found under the `scripts/downloader/BL2_download` folder.

Note: In case of updating the secure bootloader, run the `RA6M3_ERASE_SMPU_FAW.bat` file found under the `scripts/reset_kit/SMPU_Reset` folder. This script resets the Secure MPU and FAW settings, and erases the flash completely.

EKRA6M3_App_primary.bat

This file downloads and programs the primary application image in the internal flash primary slot at offset 0x10000 (Primary slot start address). This file can be found under the `scripts/downloader/EKRA6M3_App_primary` folder.

EKRA6M3_App_upgrade_3leds.bat

This file downloads and programs the application upgrade image in the internal flash secondary slot at offset 0x100000 (Secondary slot start address). This file can be found under the `scripts/downloader/EKRA6M3_App_upgrade1` folder.

EKRA6M3_App_upgrade_blueled.bat

This file downloads and programs the application upgrade image in the internal flash secondary slot at offset 0x100000 (Secondary slot start address). This file can be found under the `scripts/downloader/EKRA6M3_App_upgrade2` folder.

4.4 Secure MPU and FAW Reset

The script to reset the SMPU and FAW settings can be found under the `scripts/reset_kit/SMPU_Reset` folder. The user needs to run this script only when updating the secure bootloader.

5. RA Hardware Security Features

This section describes the Renesas RA hardware features that are elemental to the Arm Secure Boot implementation. For more information on these hardware features, refer to the hardware user's manual.

The Secure Crypto Engine (SCE) is a RA hardware peripheral that is used by the Arm Secure Boot to do the following:

1. Perform SHA-256 HASH calculation
2. Perform RSA signature validation.

5.1 Secure MPU

The Secure MPU on the RA devices can be configured to allow only secure code to access specific areas in ROM/RAM. In addition, the secure code also cannot be read by non-secure code (that is, portions of the firmware can be protected from being stolen).

The Secure MPU registers do the following:

1. Allow configuration of up to 2 Secure Data regions – one in Code Flash and one in SRAM.
2. Allow configuration of up to 2 Secure Code regions – one in Code Flash and one in SRAM. Only code in this region can access the Secure Data regions. An attempt by other code to access the Secure Data region will fail.

The Secure MPU registers do not prevent the user from erasing or reprogramming the secure areas. This is achieved by the Flash Access Window described in the following section.

5.2 Flash Access Window

The Flash Access Window registers are used to set the Code Flash address range, which can be erased/programmed. The addresses that are outside this range, referred to as outside the Flash Access Window, cannot be modified once programmed, as long as the flash access window is not cleared.

This feature is used to prevent the Arm Secure bootloader from being erased or reprogrammed.

Note: The FAW is set to the area of Flash that CAN be written, so the area of memory that is LOCKED is the inverse of the FAW address range.

5.3 FSPR (One Time Programmable Setting)

The FAW and security MPU registers can be permanently set using the FSPR bit. This bit is one-time programmable and thus must only be set once all the settings are confirmed and the device is ready to leave the production floor.

It is important that this bit is set to prevent the FAW and security MPU registers themselves from being modified in the field. For example, when the security MPU is used, the FAW and FSPR bit must be set to lock the security MPU settings from being erased. This feature can be enabled to achieve immutability of the secure boot to create a root of trust for the system.

6. Arm Secure Boot Reference Solution

This bootloader reference solution demonstrates secure booting of the application image on the Renesas RA Cortex M4-based MCU Family, using its hardware security features. In this reference solution, the EK-RA6M3 kit is used as the hardware. Support for other Renesas RA MCUs will be added in future releases.

This solution authenticates the user application using SHA-256 and validates its signature using RSA-2048 to avoid malicious software getting executed on Renesas RA Family.

The following sections talk about memory layout used in the release package, how to import, build and run the Arm Secure Boot application project. It also talks about how to verify that the demo is running correctly or not and what is expected to happen when the demo is running on the RA kit.

Memory Layout

The Secure Boot solution in the release package supports the OVERWRITE option only. Support for other boot options are not tested in this release.

For the OVERWRITE option, the application image after validation is copied from the secondary slot and overwrites on the primary slot. To maintain the flash partition consistently across multiple boot strategies, the scratch area which starts at offset 0x1F0000 is left unused during memory partitioning.

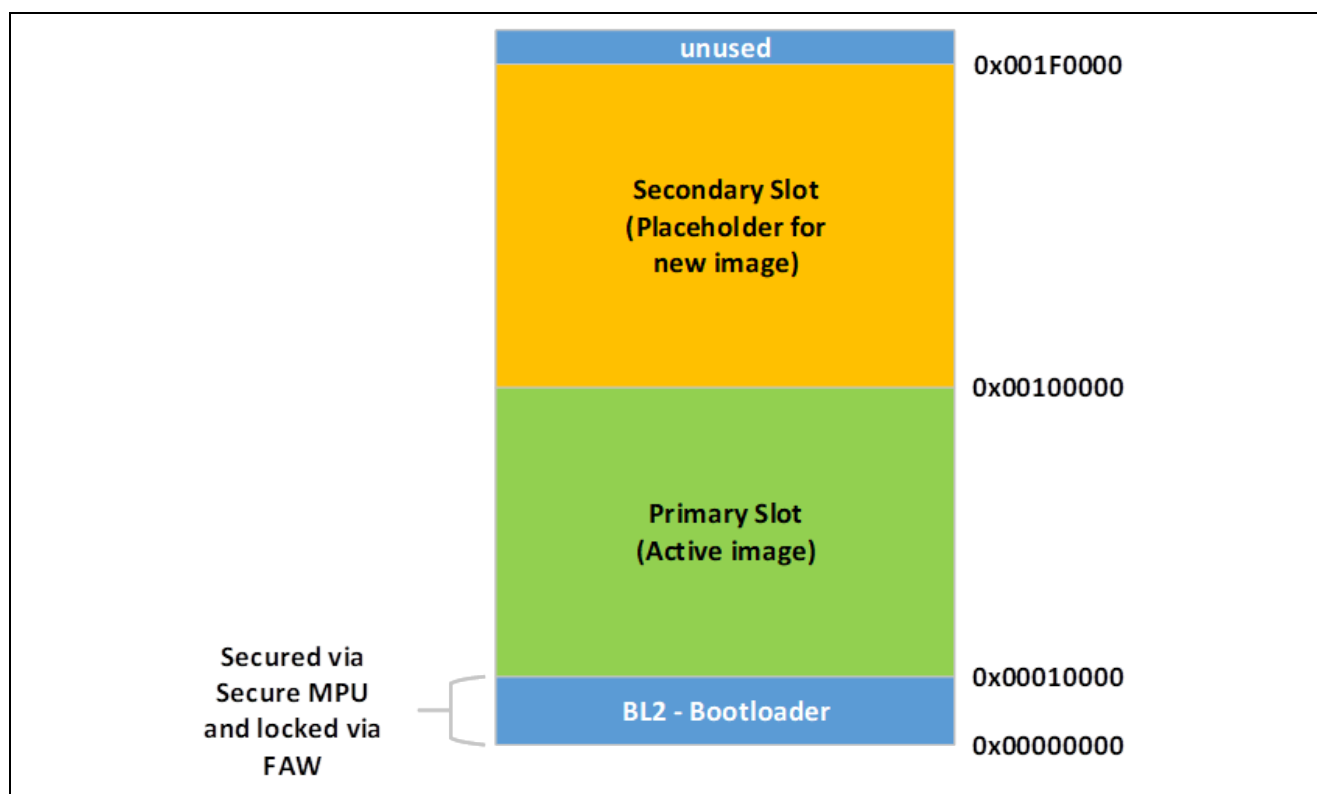


Figure 5. Arm Secure Boot Reference Solution Memory Layout

6.1 Importing, Building the Project

Refer to the FSP User's Manual for details on how to import and build a project using Renesas e² studio ISDE.

6.2 Powering up the Board

To connect power to the board, use the following instructions:

- Connect the micro USB end of the supplied USB cable to the EK-RA6M3 kit J10 connector (DEBUG_USB).

Note: This kit contains a SEGGER J-Link® On-board debugger. J-Link offers full debug and programming for the EK-RA6M3 kit.

- Connect the other end of the USB cable to the USB port on your workstation.
- Connect the USB UART TTL cable Rx, Tx, GND lines to the corresponding pins in the EK-RA6M3 kit Arduino connector. This will enable the user to monitor UART debugging messages in a serial console.

6.3 Build Time Configuration Settings

In the Secure Boot reference solution provided as part of this release package, the following required build time configuration settings are already configured in the bootloader project properties.

- BL2
- MCUBOOT_OVERWRITE_ONLY
- MCUBOOT_VALIDATE_PRIMARY_SLOT
- MCUBOOT_SIGN_RSA
- MCUBOOT_SIGN_RSA_LEN=2048
- MCUBOOT_IMAGE_NUMBER=1
- MBEDTLS_PLATFORM_MEMORY

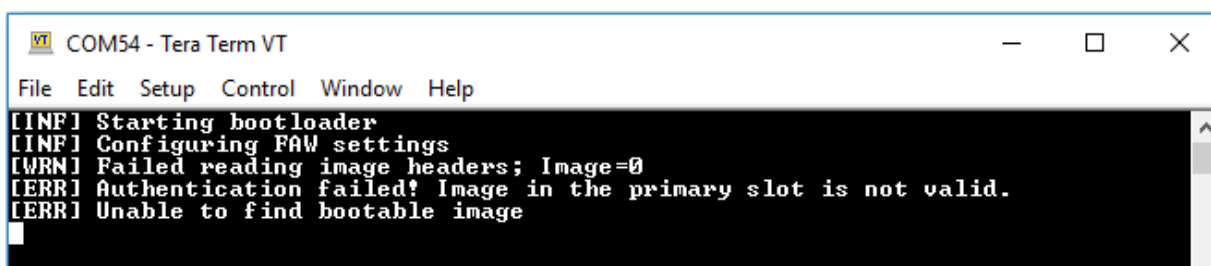
To get the background on each of the build time configuration settings, refer to section 3, Build Time Configuration.

6.4 Running the Demo

The following instructions show how to run the Secure Boot solution on Renesas RA Family using the pre-built binaries shared in the secure boot release package.

Note: At this stage, it is assumed that the user completed the instructions in previous sections on how to import, build, load the project and powering up the hardware.

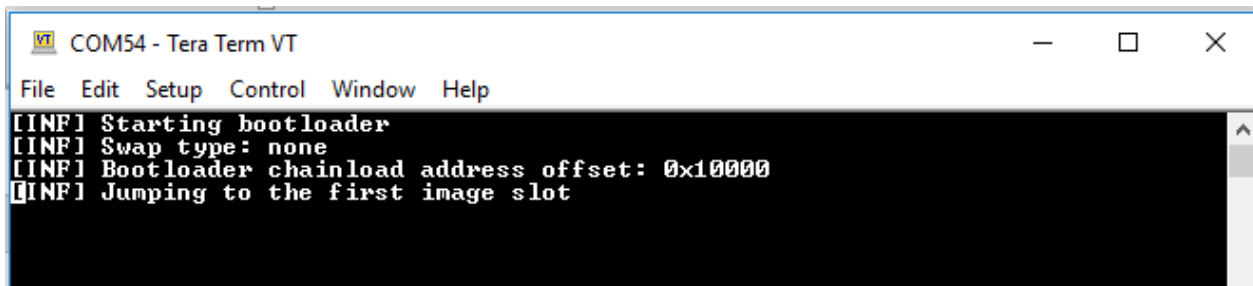
1. Open the serial console application such as Tera Term, to connect it to the EK-RA6M3 kit. The default Tera Term settings are 8-N-1, and the baud rate is 115200.
2. Run the `BL2_download.bat` file to program the secure bootloader in the EK-RA6M3 kit. This file can be found under the `scripts/downloader/BL2_download` folder. On successful download, find the serial message on the serial console as shown in the following screen capture.



A screenshot of a Tera Term VT window titled 'COM54 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The terminal output shows the following messages: '[INF] Starting bootloader', '[INF] Configuring FAW settings', '[WRN] Failed reading image headers; Image=0', '[ERR] Authentication failed! Image in the primary slot is not valid.', and '[ERR] Unable to find bootable image'. The cursor is at the end of the last line.

The above message indicates that the bootloader fails to locate the application image to boot. This is valid since at this stage; the application image is not yet programmed.

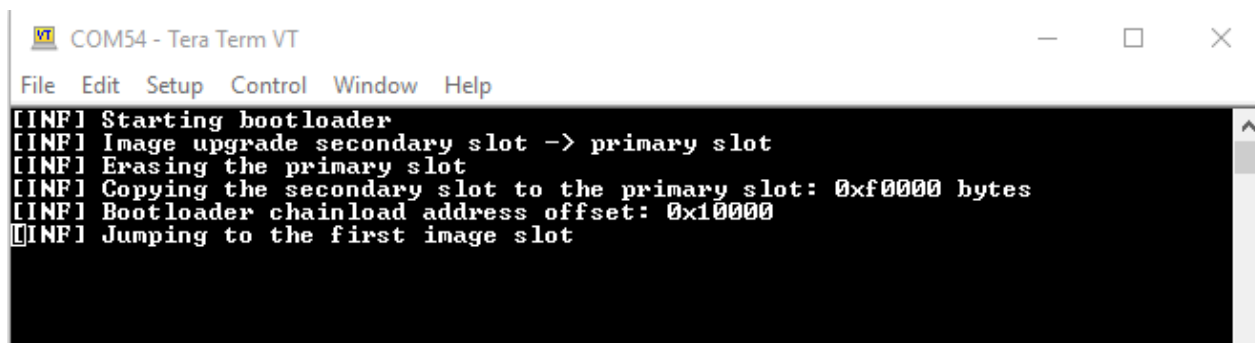
3. Run the `EKRA6M3_App_primary.bat` file to program the application image in the primary slot. This file can be found under the `scripts/downloader/EKRA6M3_App_primary` folder. At the end of download, the script resets the kit. This application functionally blinks the red LED periodically. On successful download of application image, find the following serial message on the serial console. Also, the red LED in the EK-RA6M3 kit will be blinking periodically.



A screenshot of a Tera Term VT window titled 'COM54 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The terminal output shows the following messages: '[INF] Starting bootloader', '[INF] Swap type: none', '[INF] Bootloader chainload address offset: 0x10000', and '[INF] Jumping to the first image slot'. The cursor is at the end of the last line.

The above message indicates that the bootloader successfully authenticated and validated the application image stored in the primary slot. On successful validation, the bootloader executes the application.

4. Run `EKRA6M3_App_upgrade_3leds.bat` to program the application upgrade image update in the secondary slot. This file can be found under the `scripts/downloader/EKRA6M3_App_upgrade1` folder. At the end of download, the script resets the kit. On successful download of application image, find the following serial message on the serial console. Also, the 3 LEDs in the EK-RA6M3 kit will be blinking periodically.



```

COM54 - Tera Term VT
File Edit Setup Control Window Help
[INF] Starting bootloader
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0xf0000 bytes
[INF] Bootloader chainload address offset: 0x10000
[INF] Jumping to the first image slot

```

The above message indicates that the bootloader parsed the secondary slot, authenticated and validated the application image. On successful authentication and validation, it copies the application image from secondary slot to primary slot and then executes the application from the primary slot.

Optional

Renesas has provided an additional application upgrade image that can be programmed in the secondary slot for testing. Run the `EKRA6M3_App_upgrade_blueled.bat` file found under the `scripts/downloader/EKRA6M3_App_upgrade2` folder. On successful download of the application image, the blue LED in the EK-RA6M3 kit will be blinking periodically.

6.5 Creating and Signing Application

6.5.1 Application Image Creation

The following changes are needed to be done before creating a new application image.

- Update the linker script.

The e² studio project linker script can be found under the script folder in your project.

Set the FLASH (rx) start address as 0x00010400 (This includes base address + header size of 0x400).

Update the `.id_code` section as follows.

```

.id_code (NOLOAD):
{
    __ID_Code_Start = .;
    KEEP(*(.id_code*))
    __ID_Code_End = .;
} > ID_CODE

```

- Update the project properties to generate raw binary output file. In the e² studio ISDE, this can be done by going to option **Project -> Properties -> C/C++ Build -> Settings -> GNU ARM Cross Create Flash Image-> General -> Output file format -> Raw binary**.

The above two steps are needed to be done before generating the new application project image.

6.5.2 Application Image Signing

At this stage, it is assumed the user followed instructions described in section 6.5.1 and generated the new application binary image.

Now update the `sign.bat` file found `scripts/image_creation` folder. The following components are needed to be updated in `sign.bat` before running the batch file.

- **Infile**

Set this variable to the path that contains your application project binary file.

- **Python_path**

Set this variable to the python 3 folder in your PC.

Once the `sign.bat` file is updated, install the python dependency modules in your PC.

```

pip3 install --user -r requirements.txt

```


Now run the `sign.bat` file to generate signed application image that can be programmed to the hardware kit using the downloader scripts. To run the `sign.bat` file, double click the `sign.bat` file.

Once the application image is signed, `sign.bat` will generate signed application binary such as `blinky_red_led_image.bin` (when using `blinky_red_led` Application Project from the release package). The signed binary file can be found under the `debug` folder of the corresponding application project.

Copy the signed binary image to the corresponding downloader folder. In case, the signed application binary is a primary image, copy it to the `scripts/downloader/EKRA6M3_App_update1` folder. Make sure to update the `s1.jlink` file found inside the above folder to match with the signed application binary file just generated.

6.6 Known Issues and Limitations

- In the latest release, only the `OVERWRITE_ONLY` boot option is validated.
- The Arm Secure Boot from Arm TF-M code base need to be modified to support Renesas RA MCU devices. The changes include:
 - Temporary workaround to fix `OVERWRITE_ONLY` booting. The fix is expected to be rolled into the ARM TF-M mainline release.
 - Explicit call to `psa_crypto_init()` API to initialize crypto services.

Website and Support

Visit the following vanity URLs to learn about key elements of the RA family, download components and related documentation, and get support.

RA Product Information	www.renesas.com/ra
RA Product Support Forum	www.renesas.com/ra/forum
RA Flexible Software Package	www.renesas.com/FSP
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.21.20	—	Initial document release
1.10	May.13.20		Updated secure boot to TFM 1.0 and FSP 1.0.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall 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.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

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

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

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

Contact information

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