

Renesas Synergy™ Platform

Secure Boot Manager

Introduction

This document instructs users how to evaluate the Synergy Boot Manager solution, including installing the Secure Bootloader, downloading a user application, and updating deployed applications. This installation uses the existing Renesas factory serial boot mode to install the Bootloader, it also includes tools for Mastering (digitally signing) the firmware and downloading the Boot Loader, certificates, keys, and the user application. At the end of the process, the flash access window that is set to the Secure Bootloader is locked from being modified — ensuring that it will function as an immutable root-of-trust, booting only trusted firmware. The Bootloader can be locked for the lifetime of the device by setting the FSPR bit, but for the purposes of this demonstration, the bit is not set; allowing development boards to be reset and re-flashed for running the demo multiple times or using the development board for other purposes. The Security MPU is also set to isolate sensitive data, such as keys, and the code that accesses the data into a separate memory segment. Note that some keys are wrapped, but are also stored, protected by the MPU.

Refer to the document *Synergy Secure Boot Manager Architecture* for an architectural description of the Secure Boot Manager.

Target Device

PK-S5D9

Installation requires:

1. 2 x USB Type-A to Micro-B cable for boot interface and power connection.
2. 2 x USB to RS232 Serial TTL converter adapter.

Contents

1. System Requirements.....	3
2. Extract Evaluation Software.....	3
3. PK-S5D9 Board Setup	3
4. Programming the Secure Bootloader and 1 st Receiver	4
4.1 Configuring the PK-S5D9 Development Board.....	4
4.2 Flashing the Secure Bootloader	5
5. Mastering and Programming the User Application (Secure Manufacturing).....	8
5.1 Configuring the PK-S5D9 Development Board.....	8
5.1.1 Connectivity	8
5.2 Secure Mastering: Preparation of Signed Application.....	9
5.3 Flashing Signed Application.....	11
6. Updating the Deployed Application (Secure Update).....	13
6.1 Preparation of Signed Update.....	14
6.2 Flashing the Updated Application	15
6.3 Device Reboot.....	15
7. Next Steps	16

7.1	Key Database	16
7.2	Reset the Evaluation	16
7.2.1	Resetting Application.....	16
7.2.2	Resetting Secure Boot Manager.....	17
7.2.3	Resetting the SMPU/FAW bits	17
7.3	Package and Apply Further Updates.....	17
7.4	Troubleshooting.....	18
8.	Building the Binaries.....	18
8.1	Embedded Programs.....	18
8.2	PC Programs	19
9.	Configuring an SSP Application.....	20
9.1.1	Updating the linker script.....	20
9.1.2	Adding Download Capability to the SSP Application.....	22
9.1.3	Adding OEM Certificate Challenge/Response Capability to the SSP Application.....	22
9.1.4	Debugging.....	22
10.	Project Configurations	23
10.1	Security Settings.....	23
10.1.1	Configuring the Security MPU area.....	23
10.1.2	Configuring the Flash Access Window	24
10.1.3	Configuring the FSPR.....	24
10.1.4	Configuring JTAG access.....	24
10.2	General Settings.....	24
10.2.1	Enabling usage of OEM Certificate.....	24
10.2.2	Enabling use of Hardware Crypto Engine	24
10.2.3	Enabling Debug Prints	25
10.2.4	Configuring the OFS registers.....	25
10.2.5	User-defined Functions.....	25
10.3	Configuration Summary	25
11.	Licenses	26
11.1	SHA256+ HMAC.....	26
11.2	Elliptic Curve.....	26
	Revision History.....	28

1. System Requirements

- PC Requirements: Microsoft® Windows® 10 with Intel® Core™ family processor running at 2.0 GHz or higher (or equivalent processor), 8 GB memory, 1TB hard disk or SSD, 4 free USB 2.0 ports.
- e² studio Integrated Solution Development Environment (ISDE) from Renesas.
- Renesas Synergy™ Software Package (SSP) version 1.4.0.
- Install VS2008 Runtime: It may be necessary to install Microsoft Visual C++ 2008 Redistributable package to run the Windows Desktop Utilities.

If these utilities do not run, you can download these packages from Microsoft in 32-bit (x86) and 64-bit (x64) Windows packages:

<https://www.microsoft.com/en-us/download/details.aspx?id=29>
<https://www.microsoft.com/en-us/download/details.aspx?id=15336>

- Before connecting Programming Interface – J5, download and install Renesas Flash Programmer V3 using <https://www.renesas.com/en-eu/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html#downloads>. It installs necessary drivers to load a program directly to the internal microcontroller flash through the USB device interface.
- Gearmo USB-TTL cables: <https://www.gearmo.com/shop/usb-to-3-3v-ttl-pin-header-cable-gm-ttl3vt/>.

2. Extract Evaluation Software

The Evaluation Software is provided in SecureBootManager v0.9.zip.

Extract the contents of the zip file to **C:\Renesas\Synergy** (the path of the folder “SecureBootManager” should be **C:\Renesas\Synergy\SecureBootManager**).

Notes:

- The binaries used on this walkthrough have been compiled with the following options enabled:
 - Generate Device Identity Certificate
 - Encrypt the update before sending to MCU
 - Support programming an app that uses QSPI as part of its application
 - Use QSPI as the Update Area for new images
- The Secure Boot Manager consists of the Secure Bootloader, the 1st Receiver plus several support modules and PC Tools
- Secure Bootloader is alternatively also referred to as the Bootloader.
- The term “SecurityKernel” is old terminology that refers to the Secure Bootloader; it has been deprecated, although the older name may remain in a few places in the code and files.

3. PK-S5D9 Board Setup

This section describes the pins that need to be connected to the board, and their functionality.

There are two distinct setups required to use the Secure Boot Manager. The first setup is used when programming the Secure Bootloader (SBL) and 1st Receiver. The second setup is required when programming in the user application. The following table summarizes the connections. Refer to the subsequent figure for a pictorial representation of the required connections.

PK-S5D9			Connect to	Used for
Label on PK-S5D9 Board	S5D9 Pins	Function Name		
J5		Factory Bootloader USB-CDC	PC USB port	SBM/1 st Receiver programming
P32	P302	SCI2-TX	USB-TTL Converter Pin - RX	User App Programming
P31	P301	SCI2-RX	USB-TTL Converter Pin - TX	
P15	P105	SCI8-TX	USB-TTL Converter Pin - RX	Console Debug Output
P14	P104	SCI8-RX	USB-TTL Converter Pin - TX	
J19		Board Power	PC USB port	Powering the board

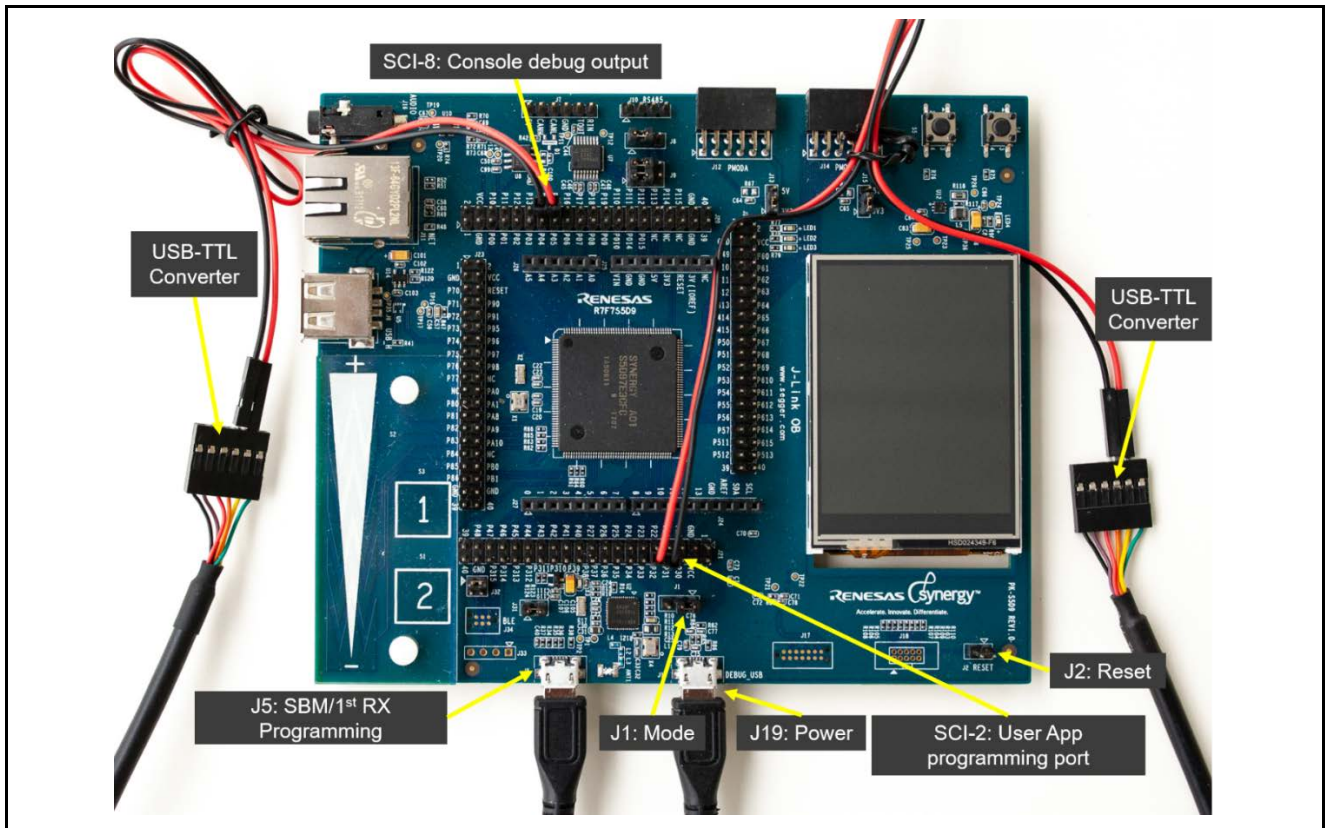


Figure 1. Wiring Diagram

A Gearmo USB-TTL converter is used here, but any other USB-TTL converter can also be used instead.

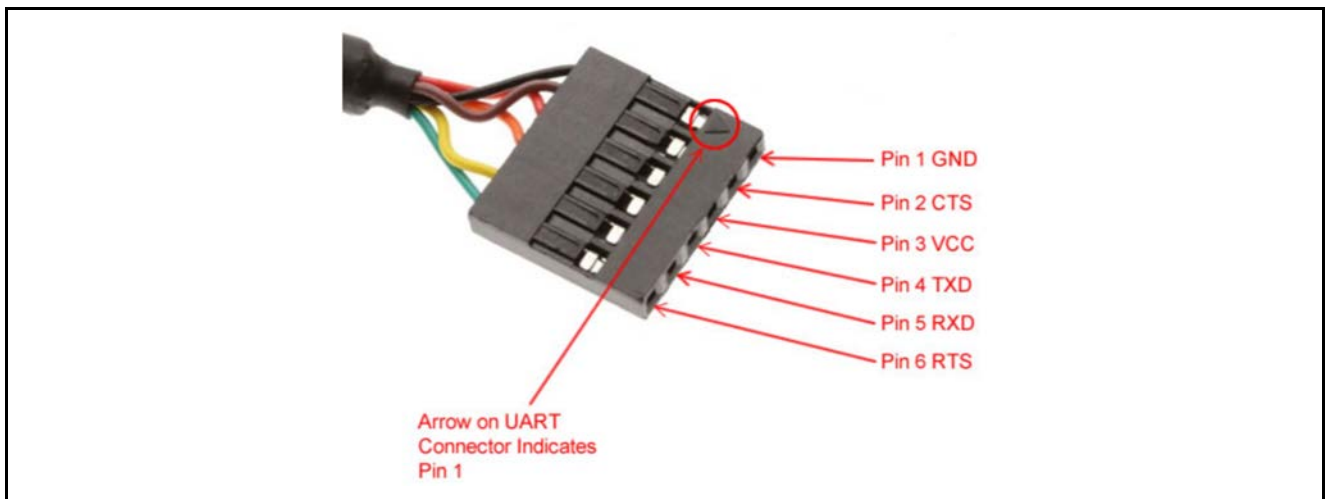


Figure 2. TTL Connector Pinout

4. Programming the Secure Bootloader and 1st Receiver

4.1 Configuring the PK-S5D9 Development Board

To configure the PK-S5D9 development board:

- Connect USB to Serial TTL converter adapter to SCI-8 on the board as shown in Figure 1. Plug the USB connector into the PC. This is used to display information on the console log. **Do not connect the second USB-TTL cable to the PC or board yet.**
- Connect Micro USB Cable from J19 connector to the PC for power connection.
- Connect Micro USB Cable from J5 connector to the PC for programming interface.
- Change the Jumper J1 position from 1-2 to position 2-3 **FACTORY BOOT LOADER MODE.**

Notes:

- **NORMAL BOOT:** If J1 is in **position 1-2**, S5D9 starts execution from internal flash (ROM). This is the normal mode of operation after initial flashing. This position will be switched back after flashing the Secure Boot Manager.
- **FACTORY BOOT LOADER MODE:** If J1 is in **position 2-3**, the S5D9 device boots with the factory bootloader in USB program mode, which enables you to load a program (the Secure Bootloader in this case) directly to the internal microcontroller flash through the USB device interface (CDC Class). This mode is used only for the initial programming of the device.

4.2 Flashing the Secure Bootloader

The **Secure Bootloader** and the **1st Receiver** will be flashed to the MCU using the factory boot loader. The **1st Receiver** is a companion application that is used to download the application firmware to the MCU the very first-time during production (after which point it can also be deleted). The **1st Receiver** is configured to use UART to perform the download).

The following steps are performed:

1. A key database initializes with new unique keys, which are used as follows:
 - A. A Bootloader Signing key is used to self-sign the Bootloader (the public key is included in the image). This key currently is used to ensure that the bootloader has not been corrupted, similar to a CRC. The design can be extended to provide the public key separately, allowing the identity of the Secure Bootloader to also be validated (thus ensuring it comes from the OEM and has not been modified).
 - B. A Mastering key is generated to be used by the Mastering tool to sign the firmware. The public key is used to validate the firmware uploaded and is included in the Bootloader.
2. The Secure Bootloader and 1st Time Receiver gets flashed to the MCU through the Factor Boot Loader's "Frame Protocol."

The following steps are used to install/flash the Secure Bootloader and 1st Receiver onto your board:

- Connect SCI-8 to the PC and reset the MCU using J2 (short it, then release using jumper) on the board.
- Open the Windows Device Manager and open **Ports**. You should see a device called **Synergy USB Boot**; to determine the serial port it is enumerated as. This is the port used to program the Secure Bootloader over the USB cabled, which is plugged into the J5 USB connection. In this case, it is COM20; yours will likely be a different port.

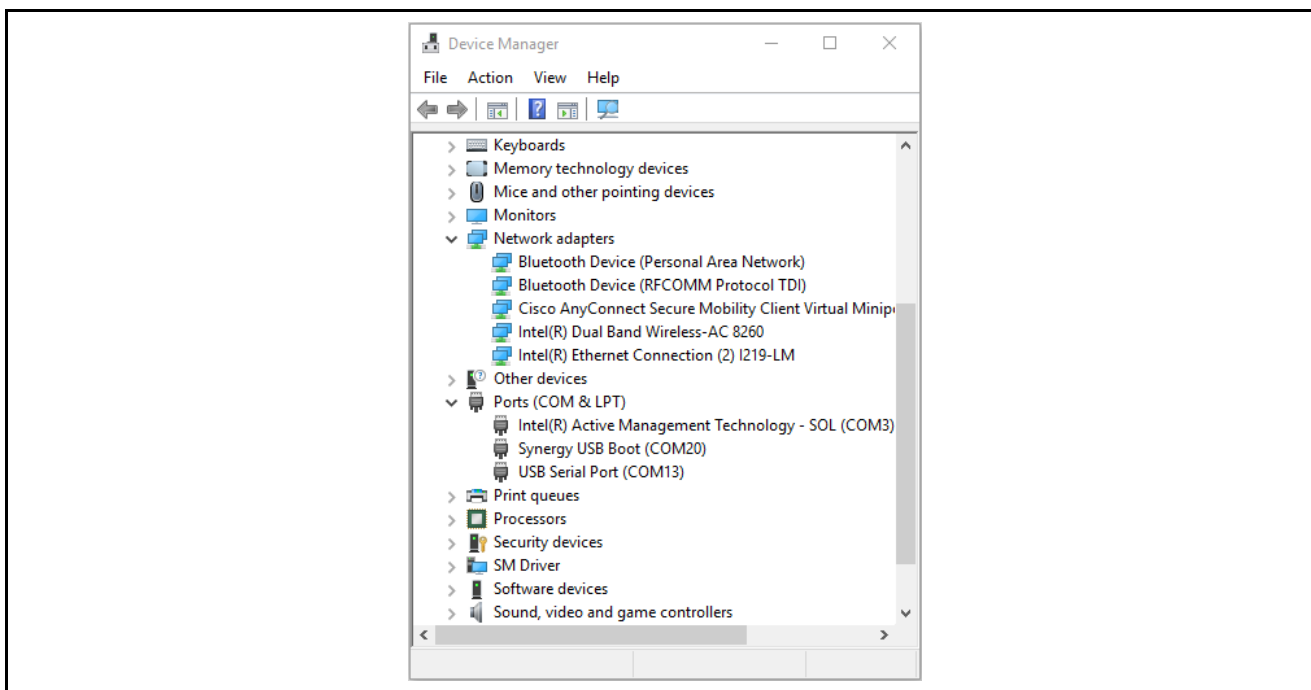


Figure 3. Device Manager

There should also be a USB Serial Port device, COM13 in this case. This is the port for the console log. For example, the cable shown with the blue and orange wires). This development version of the Security

Bootloader has logging enabled. The logging can be disabled in the code to reduce code footprint (see section 10.2.2).

Going forward, the instructions refer to COM20 (Synergy USB Boot) and COM13 (Console Log), but the actual port number may vary on the user's laptop and instructions must be interpreted accordingly.

- Open a Command Prompt, navigate to **C:\Renesas\Synergy\SecureBootManager\evaluation\bin**. Deploy the Secure Bootloader and 1st receiver to the board with the command `deploy_bootloader.bat <COMportNumber>` (for example, `deploy_bootloader.bat 20`).
- The window should show the progress of the kernel installation including key generation, Bootloader and 1st Receiver packaging, and Bootloader and 1st Receiver programming onto the MCU.

```

cmd <1> cmd
Mthomas01@SCL-MTHOMAS07 C:\Renesas\Synergy\SecureBootloader\evaluation\bin
> deploy_kernel.bat 15

Mthomas01@SCL-MTHOMAS07 C:\Renesas\Synergy\SecureBootloader\evaluation\bin
> REM script to package and deploy kernel

Mthomas01@SCL-MTHOMAS07 C:\Renesas\Synergy\SecureBootloader\evaluation\bin
> echo off
Successfully created MASTER SIGNING KEY
Successfully created FAB SIGNING KEY
Successfully created Secure Bootloader SIGNING KEY
Successfully created Secure Bootloader UPDATE KEY
Successfully created First Receiver SIGNING KEY
Successfully created First Receiver UPDATE KEY
MODULE_ID          VERSION          TYPE
    193              1              signing
    202              1              update
    202              1              signing
    192              1              signing
    203              1              update
    203              1              signing

Total number of SBM Keys = 6
Synergy Secure Bootloader Packager S5D9

Initialised
Securing the bootloader (..\embedded\SecureBootloader.srec)
  Read bootloader image from srec file
  Generated a random Seed for the device.
Inserted seed into the bootloader image.
  Generated a device certificate.
  Inserted device certificate into the bootloader image.
  Generated a device keystore.
  Populated and signed the bootloader module table.
  Written the Secured Bootloader image to disk in "..\embedded\SecureBootloader.ssrec".
Securing the 1st receiver (..\embedded\receiver1stTime.srec)
  Read 1stRx image from srec file.
  Populated and signed the 1st rx module table.
  Written the secured 1stRx image to disk in "..\embedded\receiver1stTime.ssrec".

Security packaged the Secure Bootloader + 1st Rx.
Synergy Secure Bootloader Packager (build=1)
parseSrec. bufferBaseAddress = 0x0
Parsed SREC file: ..\embedded\SecureBootloader.ssrec (0x0..0x28117)
parseSrec. bufferBaseAddress = 0x0
Parsed SREC file: ..\embedded\receiver1stTime.ssrec (0x18000..0x27FFE)
Initialising flash programming connection to port COM15
Established flash programming connection
Disabling flash access window (0x7FE000..0x7FDFFF)
Flash access window is now disabled
Erasing flash
Flash erased OK
Writing flash for ..\embedded\SecureBootloader.ssrec region (bufferOffset=0x0, 0x28118 bytes)
Flash region written OK
Erase flash for ..\embedded\receiver1stTime.ssrec region start=0x18000, end=0x27ffe
Flash region erased OK
Writing flash for ..\embedded\receiver1stTime.ssrec region (bufferOffset=0x18000, 0xFFFF bytes)
Flash region written OK
Erase flash for OEM_CERT_STORAGE region start=0x10000, end=0x17fff
Flash region erased OK

```

Figure 4. Bootloader and 1st Receiver Programming

- When the command window shows **Flash region written OK**, the Secure Bootloader and 1st Receiver have been programmed. If an OEM certificate is being used, then as the prior screenshot shows, an extra line is printed indicating that the area for the certificate was also erased.

Use the following steps to verify the installation:

- Connect a terminal program (TeraTerm, Putty, and Hyperterminal) to the console log port (for example, COM13) at **115200 baud 8N1**. This terminal is used to print the debug output from the Secure Bootloader.
- Switch to **NORMAL BOOT MODE** by changing the J1 position to 1-2 on the board.
- Reset the MCU using J2 (short using Jumper cable) on the board.
- The log on the terminal program should appear as follows:
 - The Secure Bootloader is valid – that is, has a valid self-signed signature (it has not been corrupted or modified).
 - The Main module table is invalid (because the main program/application has not yet been programmed).
 - The 1st Receiver is valid.
 - The Bootloader will run the 1st Receiver (which is the only valid module), which starts up the “Framed Data Protocol,” while waiting for the application image to be sent to it. The Framed Data Protocol is an extended version of the protocol the Factory Boot Loader uses.

```

COM13 - Tera Term VT
File Edit Setup Control Window Help

*****
[secBootloader] Synergy Secure Bootloader v0.9 (build=1)
*****

[MT] - Validating bootloader module table
[MT] - Validating bootloader module table signature
[MT] - bootloader module table signature VALID

[MT] - Validating bootloader code
[MT] - 0 - v1 - module validated - startAddr=0x0, length=65023
[MT] - bootloader module table VALID

[MT] - Validating main module table signature
[MT] - magic invalid
[MT] - main module table signature INVALID

[secBootloader] -Main Module Table Signature Verification failed
[secBootloader] - reinitialized Main Module Table.

[MT] - Validating main module table
[MT] - Validating main module table signature
[MT] - main module table signature VALID

[MT] - Validating main code
[MT] - 0 - empty module
[MT] - main module table VALID

[MT] - Validating 1stRx module table
[MT] - Validating 1stRx module table signature
[MT] - 1stRx module table signature VALID

[MT] - Validating 1stRx code
[MT] - 0 - v1 - module validated - startAddr=0x18000, length=31743
[MT] - 1stRx module table VALID

[MT] - Selecting program from 1st Rx
[secBootloader] Launching found program at addr=0x18000
[secBootloader] invokeAddr=0x18589, stack=0x20080000

*****
[1stRx] - 1st Time Receiver. (build=1)
*****

[1stRx] QSPI initialized
[1stRx] Enabling framed data protocol

```

The LED codes indicate success/failure:

- **SUCCESS:** Flashing Green and Yellow indicating that the bootloader was successfully installed. 1st Receiver is waiting for an image.
- **FAILURE:** Red indicating the bootloader is unable to find anything to invoke, or a bus/address/memory exception has occurred, or an assert has occurred.

If the LEDs indicate success, but you do not see the log above, check that you have connected the USB-TTL pins correctly and the terminal is configured.

5. Mastering and Programming the User Application (Secure Manufacturing)

This section goes over the steps required to create a signed version of the user application that then flashes the signed user application to the MCU securely (the Bootloader will validate the signature).

During product manufacturing the following process applies:

1. The Secure Mastering tool is used to package the user application binary into a signed format. A pre-generated user application is used.
2. The Secure Manufacturing tool flashes the signed binary to the update area in the MCU via the 1st Receiver that uses the same protocol as the Factory Boot Loader (The "Framed Protocol"), but with extended commands.
3. The device is reset, and the Secure Boot loader detects that there is a new image in the update area. It validates the signature, and if the signature is valid, flashes the application to be executed upon system reboot.
4. Each time the system boots up, the Secure Boot Loader validates the image before booting.

The PC is used as the host for both the mastering and the manufacturing tool. Note that the GUIs invoked in the following steps are created simply for the ease of presentation. They invoke programs which can be driven by a command line for usage in a production environment. These programs can also be modified/customized for your production environment.

5.1 Configuring the PK-S5D9 Development Board

5.1.1 Connectivity

You should already have the connections setup as described in section 3, PK-S5D9 Board Setup.

Check the following:

- Connect the second USB to RS232 Serial TTL adapter to SCI-2 (see Figure 1). Plug the USB connector into the PC. This connector will be used to flash the application over serial.
- Disconnect the micro-USB cable from the J5 connector since we are not using the factory bootloader to program the user application.
- Ensure jumper J1 is NORMAL BOOT MODE (1-2 position) so that the Secure Bootloader, which was programmed in the previous section, can run.
- Reset the MCU using J2 (short and then release using Jumper) on the board.

Use the Windows Device manager to identify the Serial Port being used by the USB-to-Serial converter adapter that was just plugged in.

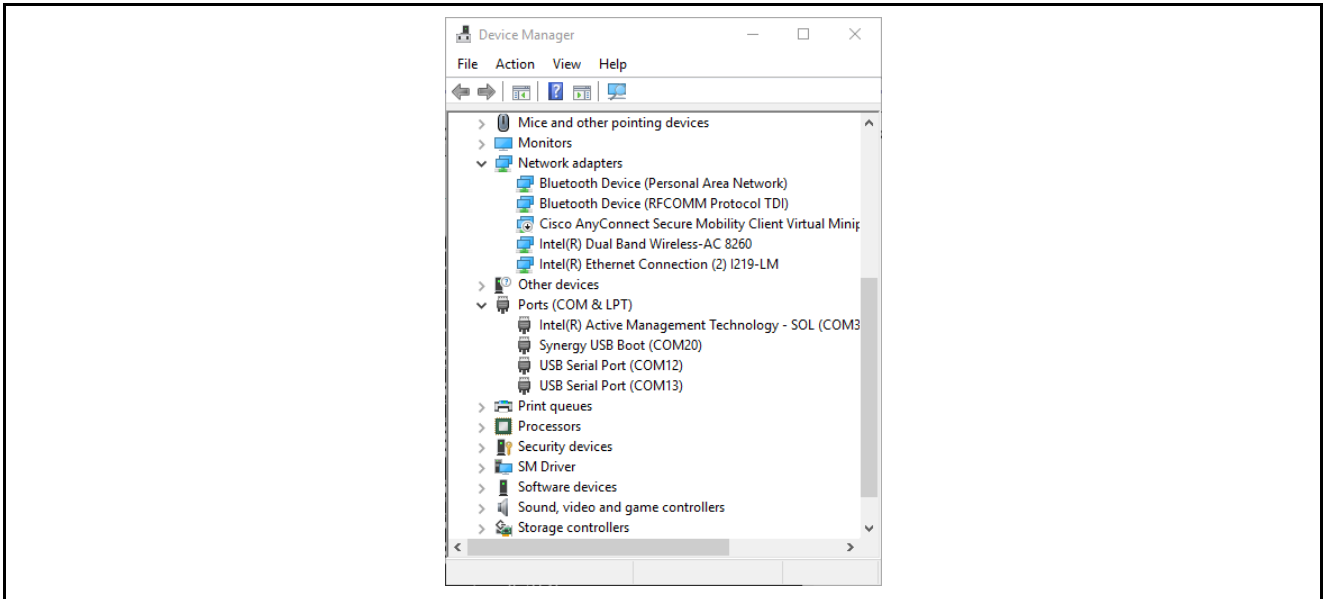


Figure 5. Identifying the Serial Port in Device Manager

In this example, the previous serial port used for the console log COM13 appears next to an additional Serial Port COM12, which is the port used to program the application. Going forward, the instructions will refer to COM13 and COM12, but the actual port number may vary on the user’s laptop and the instructions must be interpreted accordingly.

If you are not sure which port is used to flash the application image, unplug the USB cable you just connected, and see which COM port disappears from the device manager. This port is the one you should use for flashing the application.

5.2 Secure Mastering: Preparation of Signed Application

Note: If you have a display with small resolution, you might not be able to see the entire UI of the GUI applications (that is, if some of the GUI elements described in this document do not appear). If so, check the Windows Display setting in the Control Panel. Be sure **Smaller - 100%(default)** is selected.

Run **C:\Renesas\Synergy\SecureBootManager\evaluation\gui\masteringManufacturingGUI.exe** to bring up the following window:

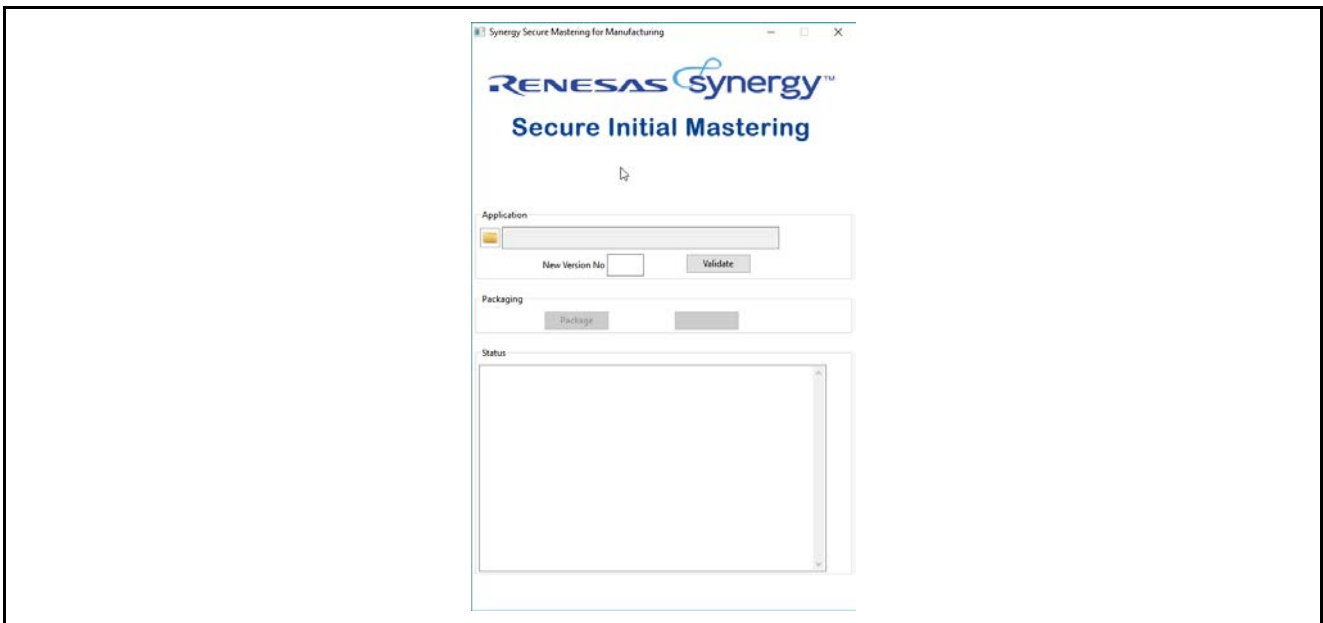


Figure 6. App Initial Secure Mastering GUI

1. Click on the application folder and browse to: **C:\Renesas\Synergy\SecureBootManager\evaluation\embedded** and select app1.srec. This file is the example application to be packaged.
2. Set the New version number to be 1; this is the initial version of the application being installed.
3. Click **Validate** to check that the srec file parses correctly and the Key Database is consistent with a new version number of 1 (that is, the Key Database maintains a key for each version of the firmware and ensures that only increasing version numbers are used. The Key Database is reset when the Secure Bootloader is installed). The **Package** button is enabled if the validation was successful.
4. Click **Package**. This will sign the application binary and write it out as **app1_YYYY_MM_DD_HH_mm_ss.ssrec** (proprietary binary format) in the same folder where the input srec file was located.
5. Click the **Complete** button — once it has turned green — to display the log file in a separate window.
6. Close the Mastering GUI.

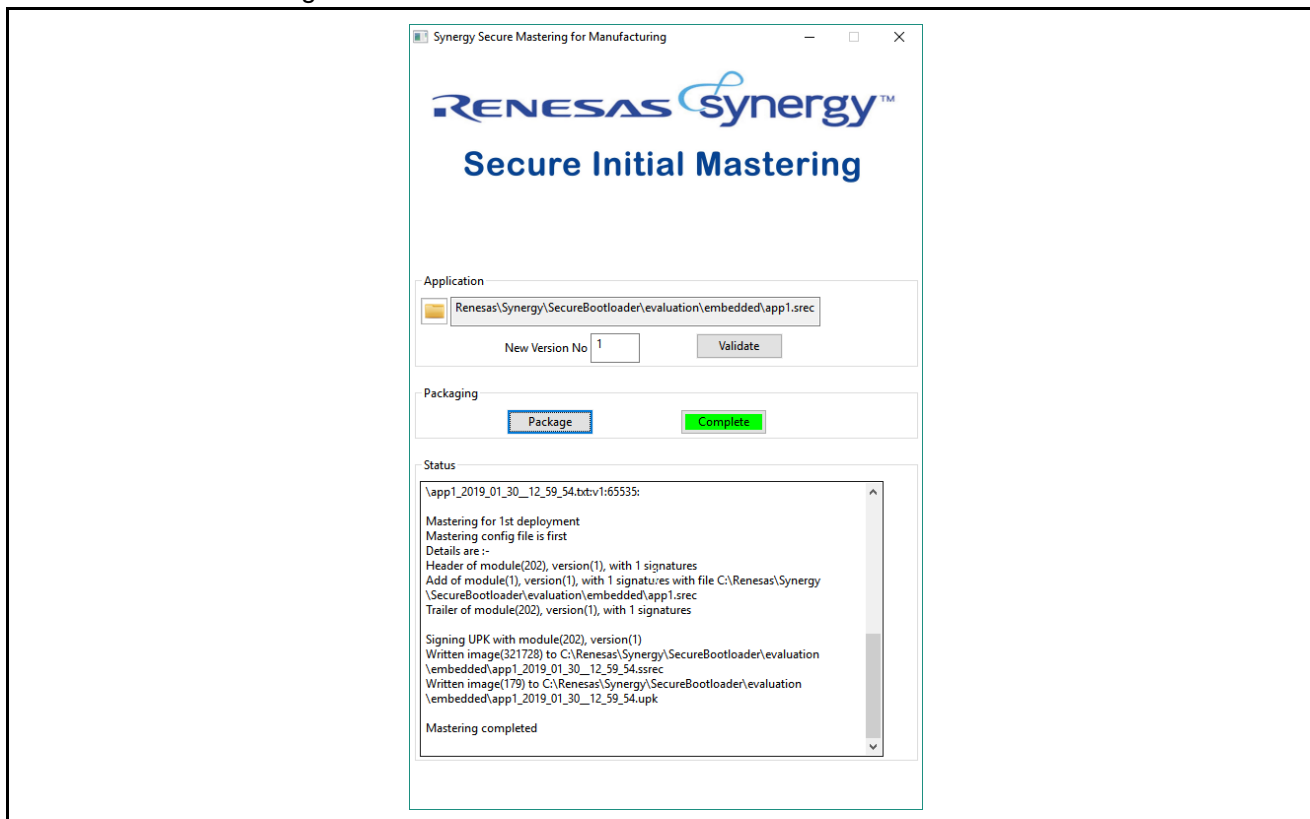


Figure 7. App Initial Mastering Successful

5.3 Flashing Signed Application

Run **C:\Renesas\SynergySecureBootloader\evaluation\gui\deployManufacturingGUI.exe**, shown in the following graphic and perform the steps:

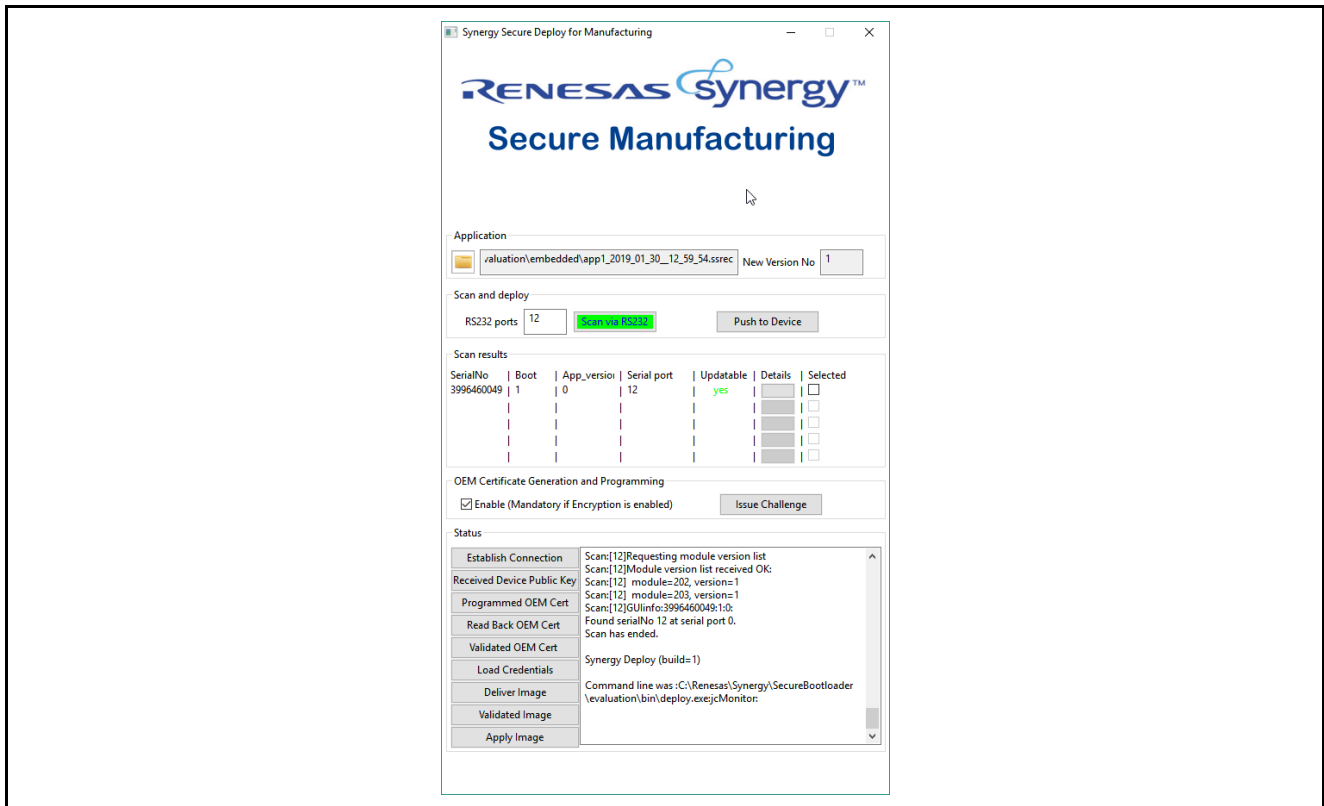


Figure 8. Secure Manufacturing GUI

1. Click on the application folder and browse to the ssrec file that was generated in the last step in **C:\Renesas\Synergy\SecureBootManager\evaluation\embedded**.
2. Enter the COM port number of the port for flashing the application (connected in step 4.1.1) into the RS232 ports box (where 12 indicates COM12 in the previous screenshot image).
3. Click the **Scan via RS232** button. In the example above, a board with device serial number 3996460049 has been found on Serial port 12. It has version 1 of the Bootloader (202,203) installed. App version = 0 indicates that no application is installed. The Updateable field is a green 'yes' indicating the application can be installed.
4. OEM Certificate Generation and Programming:
 - A. Click on the **Issue Challenge** button to generate a random number challenge to the device. Since no OEM certificate has been programmed into the device yet, the challenge will fail.
 - B. If you intend to generate and program in an OEM Certificate, check the Enable box under OEM Certificate Generation and Programming.
 - C. This is mandatory if encryption is enabled since the device identity is used to encrypt the image encryption key.

Note: Once the OEM cert is programmed in, it is not possible to erase/reprogram it without erasing everything including the SBM.

5. It is recommended to clear (or restart) the console log now.
6. Check the 'Selected' box for this board and then click **Push to Device** and the process of deploying the software (and the OEM Certificate if Enabled) will commence:
 - The 'Establish Connection' box will go green.
 - If OEM Certificate Generation and Programming is Enabled:
 - The **Received Device Public Key** button will go green as the device generated a key pair and sends the public key to the deploy tool.

- The **Programmed OEM Cert** button will go green as the deploy tool creates an X.509 certificate and programs that into the device.
 - The **Read Back OEM Cert** button will go green as the deploy tool reads back the OEM Certificate from the device.
 - The **Validated OEM Cert** button will go green as the deploy tool validates the OEM Certificate received from the device with the CA.
- There will be a 10 sec delay for the QSPI memory to be erased before loading the update image.
- The **Deliver Image** button goes green as the signed and encrypted srecord file is loaded into the GUI.
- A green bar progresses across the **Validated Image** and **Apply Image** buttons as the application is sent to the board, the app is validated on the board, and then applied to the execution area.
- The entry for the device in the “Scan Results” group will be updated to App_Version: 1 (from 0) to reflect that the application is installed.
7. The device reboots. Note that the application is now running. The orange LED stays on and the Green and Red LEDs alternate. The LCD Display indicates that the ‘Synergy Secure Bootloader demo app 1 (default version0)’ is installed, along with some status information. Note the line “module=1, version = 1”, this is the version number of the application (which is the 1st module).
 8. Scroll through the console log to see the sequence of events. The following partial screenshot shows the console log.

```

*****
[secBootloader] Synergy Secure Bootloader v0.9 (build=1)
*****

[MT] - Validating bootloader module table
[MT] - Validating bootloader module table signature
[MT] - bootloader module table signature VALID

[MT] - Validating bootloader code
[MT] - 0 - v1 - module validated - startAddr=0x0, length=65023
[MT] - bootloader module table VALID

[MT] - Validating main module table signature
[MT] - main module table signature VALID

[secBootloader] - found a pending update. Applying it
[secBootloader] - Pending update location is 0x60020000, 322048.
UPK - valid
decryptInitialise - decrypting first 32784 bytes at 0x60020128
count=0
decryptNextSegment - decrypting 176 bytes at 0x60028138
decodeHeader PASSED
decodeAddModify - module(1) is valid. Adding it to the module table
decryptNextSegment - decrypting 368 bytes at 0x600281e8
decodeAddModify - programming segment of size 32768 at 0x30000
decryptNextSegment - decrypting 32768 bytes at 0x60028358
decodeAddModify - programming segment of size 32768 at 0x38000
decryptNextSegment - decrypting 32768 bytes at 0x60030358
decodeAddModify - programming segment of size 32768 at 0x40000
decryptNextSegment - decrypting 32768 bytes at 0x60030358
decodeAddModify - programming segment of size 32768 at 0x48000
decryptNextSegment - decrypting 32768 bytes at 0x60040358
decodeAddModify - programming segment of size 32768 at 0x50000
decryptNextSegment - decrypting 32768 bytes at 0x60048358
decodeAddModify - programming segment of size 32768 at 0x58000
decryptNextSegment - decrypting 32768 bytes at 0x60050358
decodeAddModify - programming segment of size 32768 at 0x60000
decryptNextSegment - decrypting 32768 bytes at 0x60058358
decodeAddModify - programming segment of size 32768 at 0x68000
decryptNextSegment - decrypting 32768 bytes at 0x60060358
decodeAddModify - programming segment of size 32768 at 0x70000
decryptNextSegment - decrypting 26256 bytes at 0x60068358
decodeAddModify - programming segment of size 26112 at 0x78000
decodeAddModify - PASSED
decodeTrailer - PASSED
Decoded update.
[secBootloader] - Update has been applied. Rebooting.

*****
[secBootloader] Synergy Secure Bootloader v0.9 (build=1)
*****

[MT] - Validating bootloader module table
[MT] - Validating bootloader module table signature
[MT] - bootloader module table signature VALID

[MT] - Validating bootloader code
[MT] - 0 - v1 - module validated - startAddr=0x0, length=65023
[MT] - bootloader module table VALID

[MT] - Validating main module table signature
[MT] - main module table signature VALID

[MT] - Validating main module table
[MT] - Validating main module table signature
[MT] - main module table signature VALID

[MT] - Validating main code
[MT] - 0 - v1 - module validated - startAddr=0x30000, length=321024
[MT] - main module table VALID

[MT] - Validating 1stRx module table
[MT] - Validating 1stRx module table signature
[MT] - 1stRx module table signature VALID

[MT] - Validating 1stRx code
[MT] - 0 - v1 - module validated - startAddr=0x18000, length=31743
[MT] - 1stRx module table VALID

[MT] - Selecting program from main module table.
[secBootloader] Launching found program at addr=0x30000
[secBootloader] invokeAddr=0x36c25, stack=0x2000eb08
    
```

- 1st Receiver receives a request from the Secure Manufacturing tool to create a key, and the public key is then sent to the Secure Manufacturing Tool, which generates a certificate and sends it back to the 1st Receiver, which programs it into the OEM Certificate area. A challenge/response sequence is then done to verify that the certificate was properly programmed into the MCU.
 - 1st Receiver receives a request from the Secure Manufacturing tool for the Device Certificate and module list, which are then returned to the Secure Manufacturing tool.
 - 1st Receiver erases the application update area in QSPI (see Figure 14. Memory Layout) after receiving an erase command from the tool.
 - 1st Receiver programs the user appl into the application update area after receiving the write command.
 - 1st Receiver receives a command to apply the application (that is, program the application into the execution area) and invokes the Bootloader API to perform this function. The Bootloader API saves information about the update to data-flash and resets the MCU.
 - The Device is rebooted; the Bootloader comes up, sees an update available in the update area, validates and decrypts the image, then programs it into the execution area. The MCU is then reset.
 - The Device is rebooted again; the Bootloader comes up, validates the user app signature to determine authenticity and that it was successfully programmed. If successful, the Boot Manager then runs the application. **Note: This step occurs each time the device is booted.**
 - At this point, the device has been programmed with a signed user application, the secure bootloader has validated the application and executed it.
9. If OEM Certificate was generated and programmed. Click the **Issue Challenge** button to issue a challenge to the device to validate it as the owner of the certificate. This validation is achieved by the tool sending a random buffer of data to the device; the device signs it with its public key, and the tool validates the signature using the OEM Certificate from the device. For this validation to work, the user app (app1 in this case) should have the capability to respond to a challenge. The app1 has an example of how to use the crypto HAL calls to implement a response to a challenge.
 10. Clicking **Scan via RS232** again will show the Updateable field set to a red 'no'. (A version 1 application can't be 'updated' with itself.)
 11. Close the GUI.

6. Updating the Deployed Application (Secure Update)

The Secure Update consists of two stages:

1. The Update Mastering tool is used to package an application binary, which is an update to the application previously installed on the device. Updates are only allowed if the newer version is higher than version which was previously installed. (For example, if you have version 2 installed, you can only install version 3, 4, and so forth.)
2. The Update Deployment tool is used to install the update onto the development board.

Note: The mechanism in which the update is deployed to the device is application-specific. It is the application's responsibility to provide application-level connectivity to download the updates. However, for the purposes of this reference design, the applications use the same mechanism that the 1st Receiver uses – the Frame Protocol running on UART.

6.1 Preparation of Signed Update

Run **C:\Renesas\Synergy\SecureBootManager\evaluation\gui\masteringUpdateGUI.exe** as shown in the following graphic and steps.

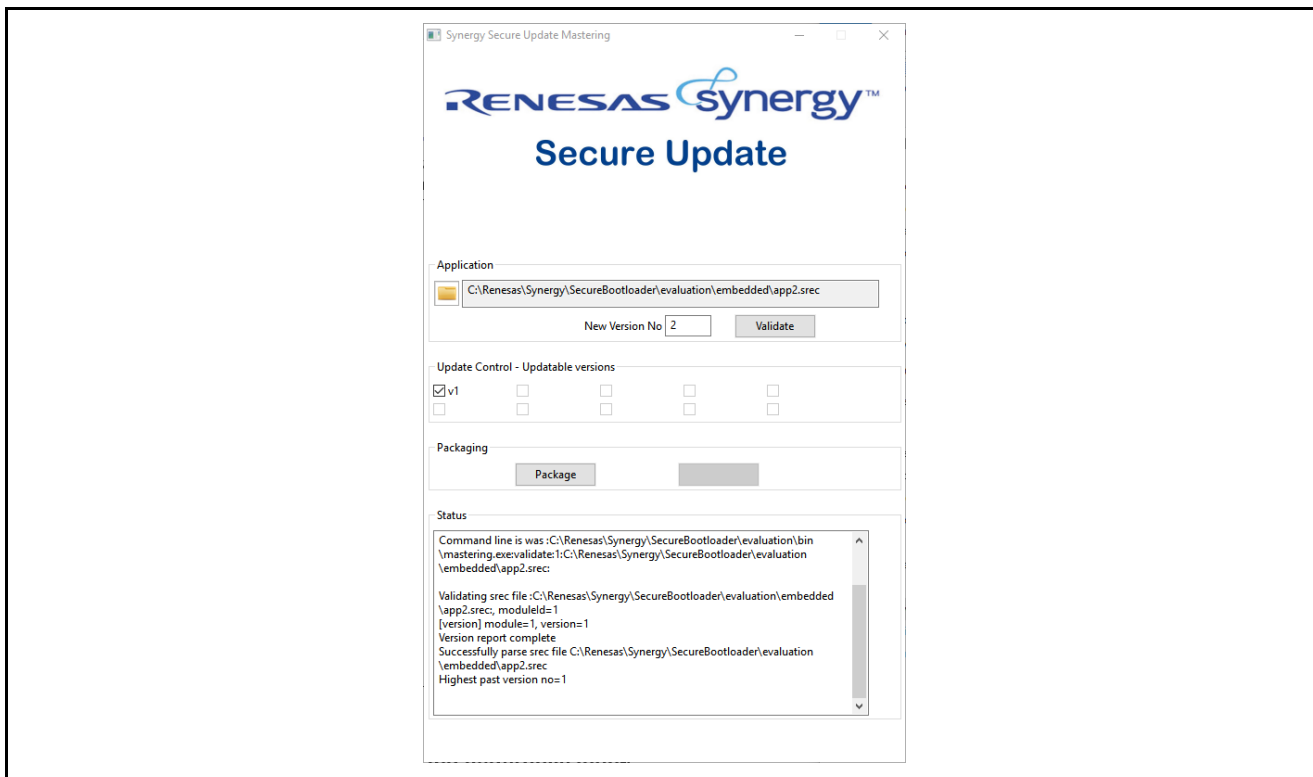


Figure 9. Signed Update Mastering GUI

1. Click the application folder and browse to **C:\Renesas\Synergy\SecureBootManager\evaluation\embedded** and select **app2.srec**. This is the example application update to be packaged – a slight variant on App1.
2. Set New Version number to 2 and select **Validate**. This checks that the .srec file can be parsed and that version 2 has not already been used (that is, there is not already an entry for that version number in the database).
3. In the Update Control box, check the versions that are already deployed in the field that you would like this update to be applicable to. Since v1 of the application code has been previously deployed, check the v1 box so that devices with version v1 can be updated with the new version being mastered.
4. Click **Package** to sign the application binary and write it out as app2_YYYY_MM_DD_HH_mm_ss.ssrec (proprietary binary format) to the same folder as the input srec file.
5. Close the GUI.

6.2 Flashing the Updated Application

Run `C:\Renesas\Synergy\SecureBootManager\evaluation\gui\deployUpdateGUI.exe`.

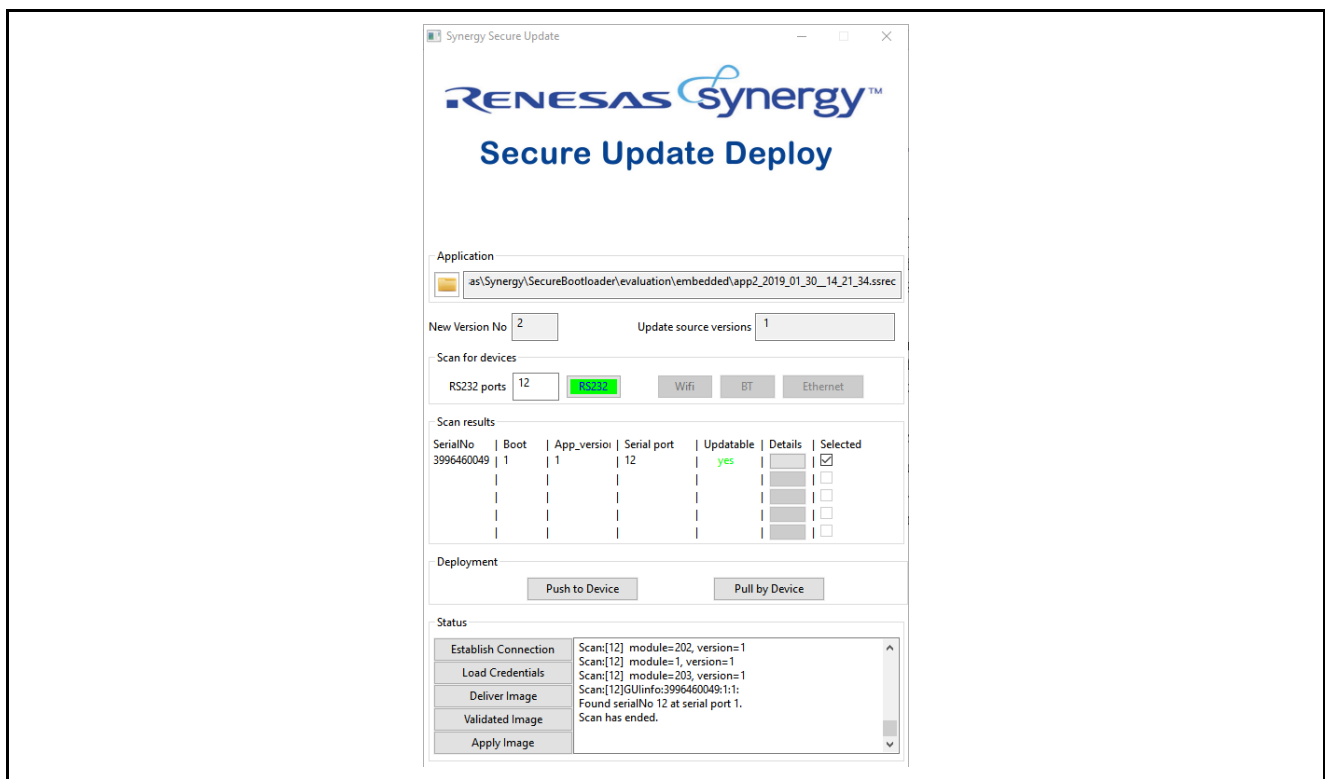


Figure 10. Signed Update Deployment GUI

1. Click the folder icon and select the file that was just created:
`C:\Renesas\Synergy\SecureBootManager\evaluation\embedded\app2_YYYY_MM_DD_HH_mm_ss.ssrec`.
2. The new version number is extracted from the .ssrec file and set to 2.
3. The Update Source versions field is extracted from the file. In this case it is set to 1. This is the only version of the application that can be updated to version 2. The device on the evaluation board will only install the update if it is one of the versions listed. If an attempt is made to apply the update to a board with the wrong version, then the device Secure Bootloader will refuse to install the update based on the signatures on the update.
4. Ensure that the board is connected as described in section 5.1.1. (No change if you've been following this document from the beginning).
5. Set RS232 Ports to the number of the COM port assigned to the programming port.
6. Click on the **RS232** button. The update tool then checks for valid devices connected to the PC and lists them in the scan results table. Up to five devices can be updated at the same time via this GUI.
7. One board should be found with App_version 1 and the Updateable field set to a green 'yes'. **Select this device by checking the "Selected" box.**
8. Select **Push to Device** button.
9. The update will be deployed to the device.
10. The debug terminal will also print out a log of the sequence. Note that the version of the main module will be shown as 2 in the debug log.
11. Close the GUI.

6.3 Device Reboot

Upon completion of the update installation, the device reboots. The secure boot code checks the integrity and authenticity of the application. If it is valid, it will reboot the device and the Green LED and Red LED will blink repeatedly three times each in alternate fashion to show that the app has been updated.

Note: The LCD "module=1, version = 2", shows that the version number has been incremented from 1 to 2.

7. Next Steps

The preceding steps have achieved the following:

1. Packaged the Secure Boot Manager and the 1st Receiver.
2. Deployed the Secure Boot Manager and the 1st Receiver.
3. Packaged/Mastered App1 for Installation on the development board as version 1 of the App.
4. Deployed App1 to the development board using the Secure Manufacturing Process.
5. Packaged/Mastered App2 to update boards that have v1 of the app programmed.
6. Updated the development board to App2 using the Secure Update Process.

7.1 Key Database

Open a command window in the folder **C:\Renesas\Synergy\SecureBootManager\evaluation\bin** and run `KeystoreManager.exe print` to display the keys in the database. Note the keys for Module 1 (the application). There is a signing and update key for each version of the application. You can run this utility throughout the steps in the demonstration to see how the key store changes.

This tool is also invoked by the mastering tools to generate and store keys.

The KeyStore uses Microsoft CAPI Next Generation for key generation and storage.

Note: Do not use the executable for anything other than viewing the database information with the `print` and `printall` commands. If the key database is modified, it may cause future updates to fail.

7.2 Reset the Evaluation

Because of the Secure Manufacturing and Secure Update processes, it is not possible to simply re-run the Secure Manufacturing process on a board where an application is already installed. A script has been provided for this evaluation that uses the update process to remove the application installed on the board and reset the key database held on the PC. This allows the walk-through to be re-run.

7.2.1 Resetting Application

To reset the flow back to the point where the Secure Boot Manager was installed, but no applications were mastered or installed, (see section 5, Mastering and Programming the User Application (Secure Manufacturing)), then perform the following steps:

1. Make sure J1 is in position 1-2.
2. Reset the MCU using J2 (short, then release using the Jumper cable) on the board.
3. Open a command window to **C:\Renesas\Synergy\SecureBootManager\evaluation\resetEvaluation**.
4. Run `resetEvaluation.bat <COMportNo>`; where `<COMportNo >` is the programming port number (for example, `resetEvaluation.bat 12`).

Note: Step will not work if the user app is using USB (for example, `app2 usb.srec`) due to the port changing partially through the reset. If this is the case, go to the next section to reset the system back to the beginning.

Once the script is run:

- The key database has been updated to remove all application version keys.
- The application has been removed from the device.
- All the previously mastered files are removed from the evaluation/embedded folder.
- The key database still has the keys used to master the Secure Boot Manager and the 1st receiver.
- The Secure Boot Manager is still on the device and the 1st receiver has been re-installed.
- The device has rebooted and the 1st receiver is waiting for a new application image to be sent to it. This can be verified by looking at the debug log that should show that the first receiver has booted and enabled the frame protocol. The LEDs will also be flashing Green and Yellow.
- You can now re-run the steps from Section 5, Mastering and Programming the User Application (Secure Manufacturing).

Note: This step does not clear the OEM Certificate if you had programmed it in the first place. To be able to reprogram the OEM Certificate, follow the next section to Reset the SBM.

5. If this step does not work, follow the procedure in the next section to reset the entire system.

7.2.2 Resetting Secure Boot Manager

To reset the demonstration back to the point at the beginning — before the Security Boot Manager was installed (see section 3) — run the following steps:

1. Open a command window at **C:\Renesas\Synergy\SecureBootManager\evaluation\bin**.
2. Run **keystoreManager.exe clear**.

At this point,

- The key database has been flushed of all keys.
- Re-run the steps from Section 3.

If you have the SMPU and FAW bits enabled (default setting for the demonstration), then follow the next section to clear these bits.

7.2.3 Resetting the SMPU/FAW bits

If the Secure MPU was enabled in the SBM configuration, then it is not possible to connect the debugger via JTAG (J5) anymore. Similarly, if the FAW (Flash Access Window Setting) bits have been set, they cannot be erased via normal programming.

To recover from such a situation, follow these steps:

1. Set the MCU into Boot Mode but changing J1 to **position 2-3**.
2. Reset the board using J2.
3. Run the “S5D9_ERASE_SMPU_OSIS_AWS.bat” file that is provided under **\SecureBootManager\evaluation\resetEvaluation\Erase_SMPU_OSIS_AWS**. You may need to edit the J-Link path in the batch file to match your J-Link installation path.
4. Reset the board using J2.

This script first clears the JTAG ID, FAW settings, and then erases the first flash block (and thus the Secure MPU bits).

Note: If the FSPR bit is set, it is not possible to clear the Secure MPU or FAW bits. The FSPR cannot be modified once set. See 10.1 Security Settings and the MCU Hardware Manual for details.

7.3 Package and Apply Further Updates

At the end of the walk-through, you can apply further updates via app4.srec. The srec files are also found in **C:\Renesas\Synergy\SecureBootManager\evaluation\embedded\bin**.

- Check the boxes of all previous versions installed.

You can also apply selective updates by choosing any specific ones out of the previous version listed in the Signed Update Mastering GUI.

Note: You can also use app_usb.srec. This project uses USB-CDC instead of UART in the user application. Ensure J5 is connected so the app can communicate over USB. The deploy tool expects the updated code to respond on the same port over which the download was sent; since the USB switches to a different port, the deploy tool will not get the confirmation it was expecting. This lack of confirmation does not cause any problem with the update process, but the GUI will wait indefinitely for a response. You can close the GUI once the updated device enumerates as a USB-CDC port on the MCU.

7.4 Troubleshooting

Common problems that occur and solutions are as follows:

- Communication fails
 - Check to see that the wired are connected as described.
 - Check that the serial connection is set to the specified baud date.
- Programming fails
 - Reset the board such that the 1st receiver runs.
 - Make sure you have the log port connected to see the status.
- The walk-through cannot be re-run without resetting the applications installed on the board and the key database stored on the PC. When packaging is attempted for a second time, the red **FAILED** button appears. The status window indicates 'failed to make new keys for add of module.
 1. Open a command window in **C:\Renesas\Synergy\SecureBootManager\evaluation\resetEvaluation.**
 2. Run `resetEvaluation.bat <COMportNo>`.
 3. Where `<COMportNo >` is the USB-to-Serial converter adapter port number.
 4. Restart the masteringprocess.

8. Building the Binaries

The embedded projects are included in the folder

C:\Renesas\Synergy\SecureBootManager\renesas\src\embedded. The following instructions are only needed if you wish to recompile the code.

8.1 Embedded Programs

1. In e² studio, select **File -> Import... -> Existing Projects into Workspace** and browse to the above folder in the **Select Root Directory** section:

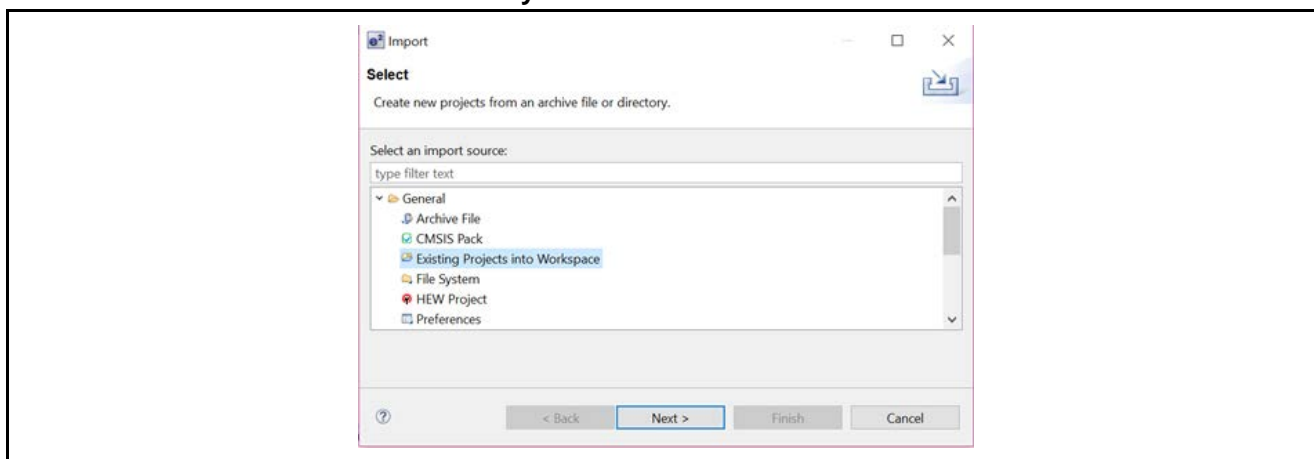


Figure 11. Importing Project

2. Select to import all projects as shown in the following figure. **DO NOT CHECK** the “Copy projects into workspace” box.

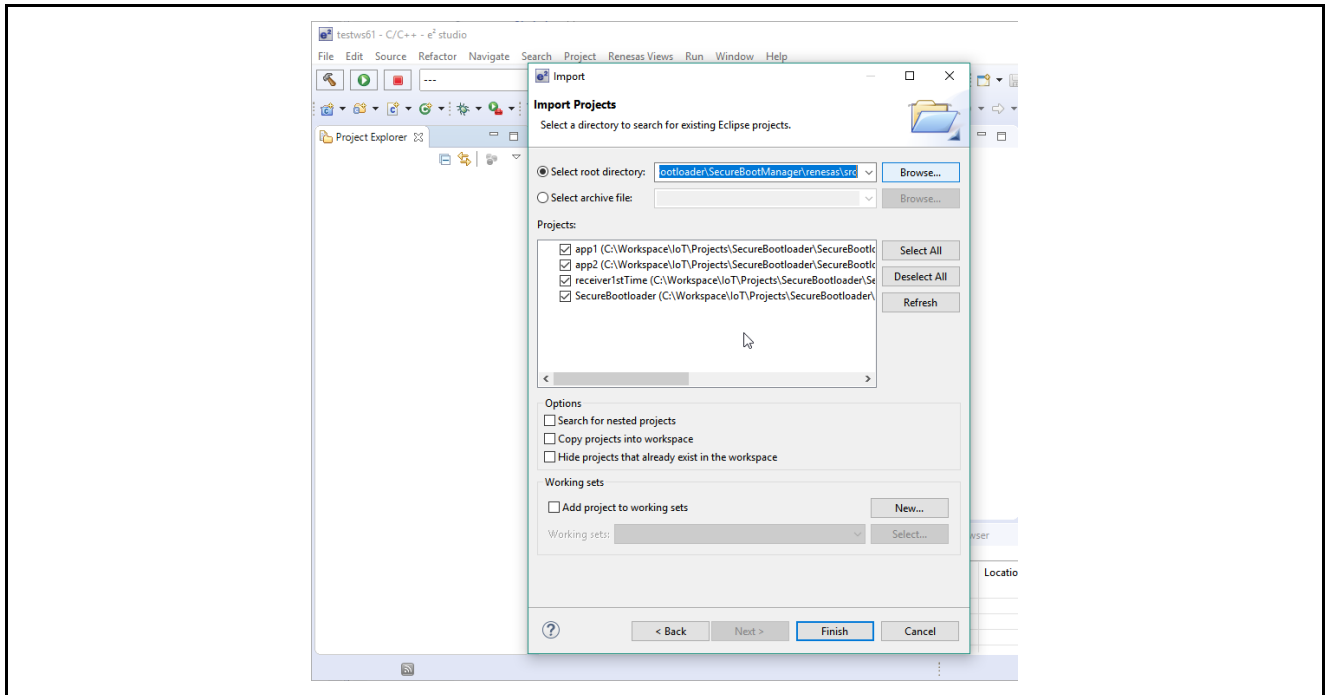


Figure 12. List of Projects

- Build each of the projects in e² studio. The project should build with zero errors and zero. Secure Boot Manager has approximately 52 Kbytes ROM size at O2 optimization with debug prints enabled and 32 Kbytes with debug prints disabled (see section 10.2.2).

8.2 PC Programs

- All the PC code is compiled in Visual Studio® 2017.
- In Windows Explorer, go to path **C:\Renesas\Synergy\SecureBootManager\renesas\src\pc\apps** and search for the *.sln file.
- Double-click each .sln file in turn to load them in VS2017.
- Build each project in VS2017. The project should build with zero errors. The deploy project has some debug warnings which are related to the openssl library debug symbols and has no effect on the code functionality. In this case, deploy project has being compiled. To minimize the dependencies on Microsoft Libraries, build the programs in Release mode.

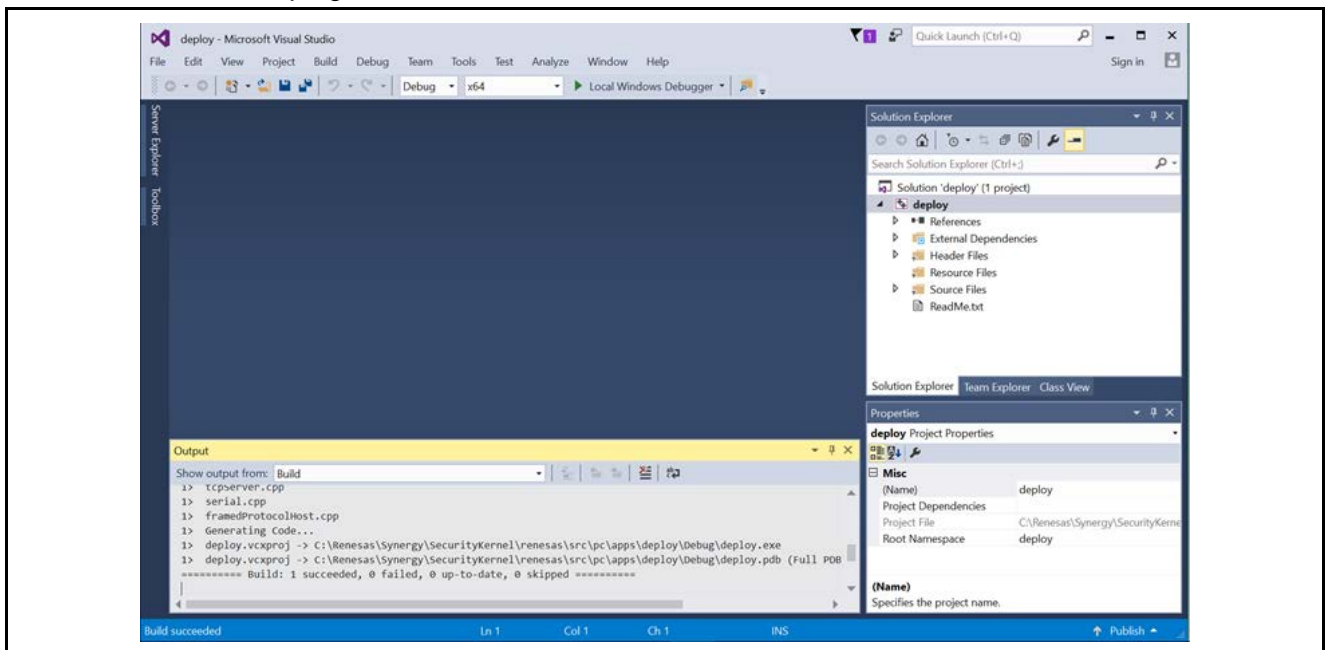


Figure 13. Compilation of Project

9. Configuring an SSP Application

To be able to use an SSP application with the secure bootloader solution, a clear understanding of the shared system elements is required. The following items need to be taken into consideration:

- The SSP user-application linker sections must be updated so as not overlap with the rest of the code.
- Only ROM registers (FAW, OFS, S-MPU) can be configured in the bootloader, even though the SSP configurator will show options to configure them in the application.
- To be able to perform application updates, the application must be able to communicate with the external world to download updates and with the flash API to program updates to the internal update area.

The following steps take you through the linker modifications needed for a new SSP application and the steps to add the Frame Protocol implementation to the SSP application in order to download updates.

9.1.1 Updating the linker script

- Refer to Figure 14 to get an understanding of the memory layout.
- For this section, use the linker script file of the provided app1 project as a reference as it is structured similar to what is being done below. This file will be the S5D9.ld file in the **script** folder. If opening in e² studio, then once the file is opened, click the s5d9.ld tab at the bottom to view the file in text mode.
- Create a new SSP project and add any functionality you would like (for example, Blinky with a specific sequence).
- In your SSP project, open the s5d9.ld linker script file, located in the scripts folder in the project. If opening in e² studio, once the file is opened, click the s5d9.ld tab at the bottom to view the file in text mode. At the very top of the file, the MEMORY areas are laid out for the SSP project.
- The memory layout that is used in the Bootloader project is defined in **C:\Renesas\Synergy\SecureBootManager\evaluation\src\embedded\common\loaderScript\memoryMap.ld**.
- Replace the MEMORY definition in your application's linker script file by including the **memoryMap.ld** file at the top of the S5D9.ld file; except for the ID Code section. You may need to include the whole path of the memoryMap.ld file.


```

INCLUDE
C:\Renesas\Synergy\SecureBootManager\evaluation\src\embedded\common\loaderScript\
memoryMap.ld

MEMORY
{
    ID_CODES (rx)      : ORIGIN = 0x0100A150, LENGTH = 0x10    /* 16 bytes */
}
      
```
- Read through the memoryMap.ld file to understand the memory layout; particularly the area reserved for the user application and temporary holding area for updates (Update Area). Refer image that follows:

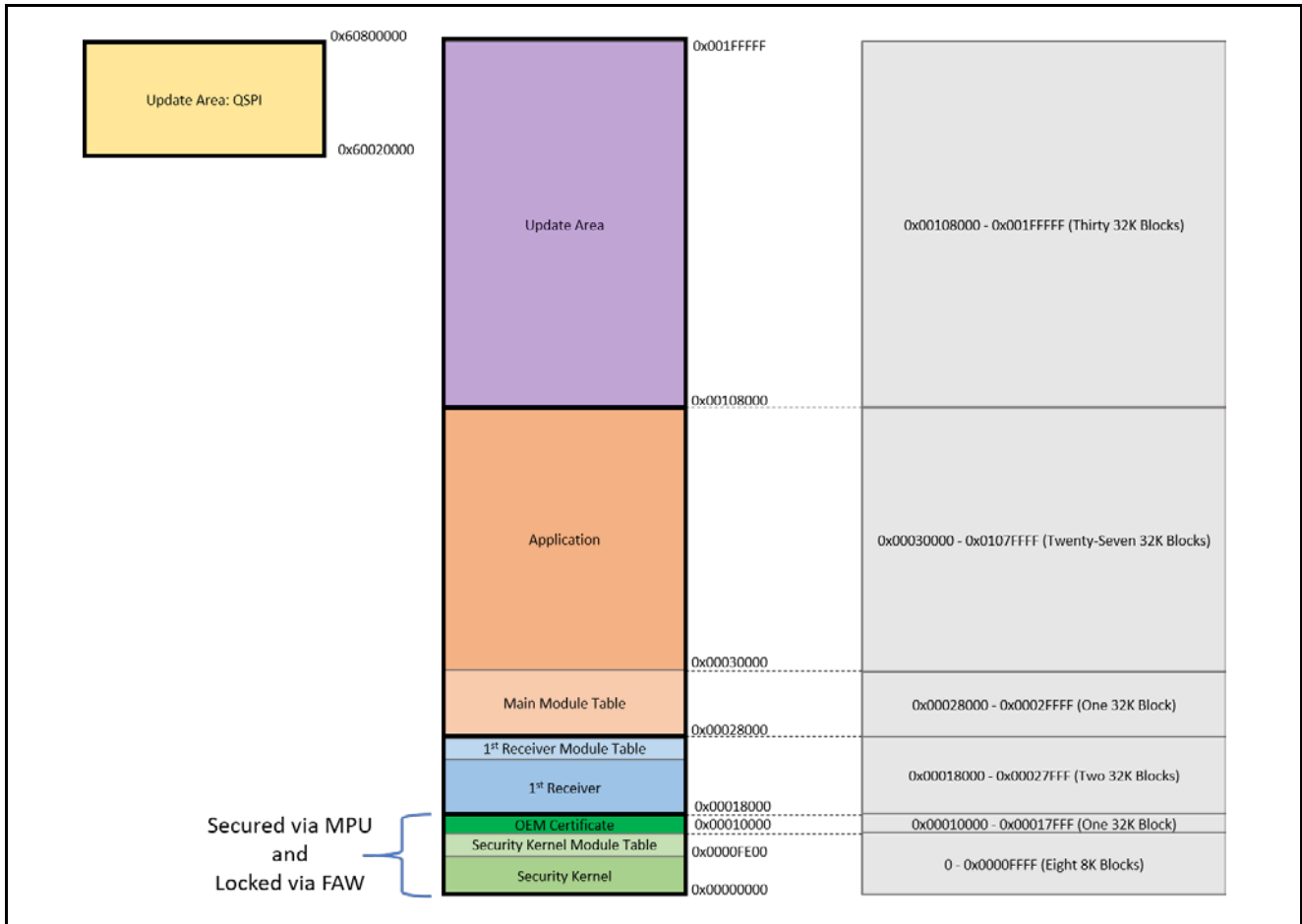


Figure 14. Memory Layout

- Add a line “__Vectors_Start = .;” before the “KEEP(* (. vectors))” line. This line is required to create the definition for the “__Vectors_Start” symbol that is used by the SSP.
- Comment out the DTC vector table entry, which forces the DTC vector table to a fixed RAM location. The DTC vector table will be auto-located in RAM if used in the SSP project:

```

/*
.ssp_dtc_vector_table :
{
    . = ORIGIN(RAM);
    *(.ssp_dtc_vector_table)
} > RAM
*/
    
```

- Build the project. You can now use the generated srec file instead of the app1.srec for the Secure Manufacturing or Secure Update process.
- Note that even though the ID Code and ROM register sections are maintained in the SSP application, they will not be written to the memory. Those locations can only be set in the bootloader code itself. They are only maintained since SSP expects those sections to be present; this will be fixed in future.

9.1.2 Adding Download Capability to the SSP Application

If a deployed application is expected to be updated after field deployment. The application should have H/W communication capability as well as a software download protocol implemented. The current PC side update deployment tool uses the Frame Protocol implemented over a serial channel to send updates. The provided app1 project shows how the firmware can receive the data sent over the Frame Protocol via serial port. Refer to that project for details on how to add download capability to your SSP application. The included app2 project shows an example of how to add download capability over USB to the user app.

The primary steps involved to add download-over-UART capability to your project are:

- Add the `r_flash_hp` module to support writing to flash.
- Add the `r_sci_uart` module to support data transfer. You can replace the UART functionality with any other communications channel so long as both the transmitting side and receiving side are updated to use the same medium.
- Add a semaphore and a queue to the application thread where the update will be downloaded.
- Add the Frame Protocol files to the SSP project:
 - `framedProtocolCommon.c`
 - `framedProtocolCommon.h`
 - `framedProtocolTarget.c`
 - `framedProtocolTarget.h`
- Refer to the `PMOD_thread_entry.c` file in `app1/src` folder for information on how to initialize the modules and process received data.

9.1.3 Adding OEM Certificate Challenge/Response Capability to the SSP Application.

The 1st Receiver application in conjunction with the deploy PC tool shows an example of how a host can validate the identity of a device certificate.

The sequence is as follows:

1. The deploy tool on the PC side requests the OEM certificate from the 1st Receiver on the MCU.
2. The 1st Receiver calls an SBM API to retrieve the certificate and then sends that to the deploy tool.
3. The deploy tool validates that certificate with the local CA to determine that the certificate is valid. This only validates the certificate and not the device which is sending it.
4. The deploy tool then generates a random buffer of data and sends that to the 1st Receiver.
5. The 1st Receiver signs that data with the wrapped private key via calls to the SCE hardware and sends that signature to the deploy tool.
6. The deploy tool verifies the signature using the public key from the X.509 certificate this validating the device identity.

The code in the 1st Receiver and deploy tool are provided as an example.

Since the challenge/response mechanism requires usage of the crypto HAL modules, if the SSP user app is expected to perform this action, it needs to use the crypto HAL modules provided by SSP. The challenge request is received via the frame protocol in the file `frameProtocolTarget.c` via the function `handleHrkCertChallengeResp()` which makes a call to `hwSignData()`. That function needs to be implemented in the user application. It is implemented in 1st Receiver as an example, but that uses the crypto modules that are part of the 1st Receiver project. There is also an example in the user app1 project in the file `challengeResponse.c` which uses SSP HAL calls to implement the same functionality.

9.1.4 Debugging

The user application can be developed and debugged as a normal standalone project with the default linker settings. However, once the application functionality is completed and is to be tested with the Secure Bootloader, then it is necessary to make some debugger changes to allow this.

Once the project linker settings are changed as described in the previous section, compile the project in Debug mode so that all the debug symbols are generated. Now, the srec file needs to be mastered and downloaded to a device with an installed Secure Bootloader as described in section 5.2.

Once the mastered user application has been downloaded to the device, update the debug session settings so that the code flash is not updated when the debugger connects to the device. This is done by modifying the `<debug_session_name>.jlink` file in the project such that the flag `EnableFlashDL` is set to 0.

The provided Jlink Debug sessions have has flash download disabled for the bootloader and 1stRx.

Now when the debugger connects to the device, it will load the debug symbols but not download any code to the device.

Note that if you want to update the code now, you will have to go through the update mastering process again. Thus, it is recommended that all application development and testing be done without the Secure Boot Manager until the system integration stage.

Remember to change that flag back to 2 if you want to download code to the device during debugging.

10. Project Configurations

10.1 Security Settings

This section describes the various registers on the MCU that are used to control access to data as part of the security strategy.

10.1.1 Configuring the Security MPU area

The security MPU registers are used to control read access to memory regions. The hardware allows the user to configure three secure regions and three (not corresponding) PC (program counter) ranges. The secure regions may be in ROM, RAM, or Data Flash.

The secure regions can only be read when the PC is in any of the defined PC ranges (that is, by code that is in those same ranges).

The Security Boot Manager firmware and Installer PC tool are configured to set the Security MPU registers to cover the Security Boot Manager, Secure Boot Manager Module Table, and OEM Certificate area of code. This prevents reading of the Security Boot Manager code base.

The registers are configured in SecureBootloader\sbmCfg\mcu_rom_cfg.h. See the hardware manual on how to modify the registers.

The current settings are:

```
#define SECMPUAC_DISPC0_VAL (0U) //SECMPU Program Counter region 0 Enabled
#define MCU_CFG_SECMPU_PC0_START 0x00000000U // Security MPU Program counter region 0 Start address
#define MCU_CFG_SECMPU_PC0_END ((ADDR_BOOTLOADER_MT -1) | 0x3) // * Security MPU Program counter region 0 End address

#define SECMPUAC_DISPC1_VAL (1U) // * SECMPU Program Counter region 1 Disabled
#define MCU_CFG_SECMPU_PC1_START 0x00000000U // Security MPU Program counter region 1 Start address
#define MCU_CFG_SECMPU_PC1_END 0xFFFFFFFFU // Security MPU Program counter region 1 End address

#define SECMPUAC_DIS0_VAL (0U) // SECMPU Region 0 Enabled
#define MCU_CFG_SECMPU_S0_START 0x00000000U // Security MPU Region 0 (code flash) Start address
#define MCU_CFG_SECMPU_S0_END ((ADDR_1ST_RX -1) | 0x3) // Security MPU Region 0 (code flash) End address

#define SECMPUAC_DIS1_VAL (1U) // SECMPU Region 1 Disabled
#define MCU_CFG_SECMPU_S1_START 0x1FF00000U // Security MPU Region 1 (SRAM) start address
#define MCU_CFG_SECMPU_S1_END 0x200FFFFFU // Security MPU Region 1 (SRAM) end address

#define SECMPUAC_DIS2_VAL (1U) // SECMPU Region 2 Disabled
#define MCU_CFG_SECMPU_S2_START 0x400C0000U // Security MPU Region 2 (security function 1) start address
#define MCU_CFG_SECMPU_S2_END 0x400DFFFFU //Security MPU Region 2 (security function 1) end address

#define SECMPUAC_DIS3_VAL (1U) // SECMPU Region 3 Disabled
```



```
#define MCU_CFG_SECMPU_S3_START 0x400C0000U // Security MPU Region 3 (security function 2) start address
```

```
#define MCU_CFG_SECMPU_S3_END 0x400DFFFFU //Security MPU Region 3 (security function 1) end address
```

After successful installation of the Secure Boot Manager with Security MPU enabled, **programming interface J5 will not be accessible** (that is, you cannot re-flash the application through normal JTAG interface). To re-gain access to programming interface J5, you would need disable Security MPU settings by erasing all of block 0. To do this, run the script as described in section 7.2.3 Resetting the SMPU/FAW bits.

10.1.2 Configuring the Flash Access Window

The Flash Access Window is used to prevent the Secure Boot Manager and Receiver 1st Time from being overwritten / modified. It works by setting the address range which the user can erase and reprogram. The Flash Access window is set in the Kernel Installer Visual Studio application to cover addresses from the end of SBM Module Table to the top of Program Flash.

The FAW is enabled by default. However, during development, it is recommended to disable it to allow the Secure Boot Manager to be reprogrammed (that is, to NOT lock the board each time it is programmed).

To enable the FAW, define `ENABLE_FLASH_ACCESS_WINDOW` in the Kernel Installer Visual Studio project. Do this by opening the project properties and under **C/C++>Preprocessor**, add a Preprocessor definition for `ENABLE_FLASH_ACCESS_WINDOW`. To disable it, remove the definition from the project.

Once these bits are programmed on the MCU, special methods are required to erase them. To erase these bits, run the script as described in section 7.2.3 Resetting the SMPU/FAW bits.

10.1.3 Configuring the FSPR

The FSPR bit is used to prevent modification of the Secure MPU and Flash Access Window registers. This bit must only be set when the Secure Boot Manager is being programmed into the device during manufacturing since **it cannot be modified once set**.

To program the FSPR, define `ACTIVATE_THE_FSPR` in the Kernel Installer Visual Studio project. Do this by opening the project properties and under **C/C++>Preprocessor**, add a Preprocessor definition for `ACTIVATE_THE_FSPR`.

It is disabled by default in the project

Care must be taken to ensure that the device kernels and applications are correctly constructed as it may not be possible to recover the device in case of programming mistakes.

10.1.4 Configuring JTAG access

The ID Code in the OSIS Register has not been set. This permits JTAG access to the device. The Certificate Installer tool can also set the ID Code register and limit JTAG access to the Secure Bootloader provisioned device.

The JTAG ID is disabled by default to allow the Secure Boot Manager and 1st Receiver to be reprogrammed in development (that is, to NOT lock the board for life). To enable the JTAG ID, define “`ENABLE_JTAG_LOCKING`” in the Certificate Installer Visual Studio project. Do this by opening the project properties, and under **C/C++>Preprocessor**, add a Preprocessor definition for “`ENABLE_JTAG_LOCKING`”.

Then recompile the project and invoke the executable with the desired JTAG locking ID.

10.2 General Settings

10.2.1 Enabling usage of OEM Certificate

To enable the usage of OEM Certificates in the system, add the symbol `ENABLE_SBM_OEM_CERTIFICATE` to the compiler pre-processor by right clicking on the **SecureBootloader** project in e² studio and under **Settings>Cross ARM C Compiler**, add the symbol. Do this for the 1st Receiver project as well and the user app (if you are using the `frameProtocolTarget.c` file in the user app to download new updates).

10.2.2 Enabling use of Hardware Crypto Engine

The software allows the user to determine if hardware or software elements are to be used for the cryptographic operations. Since the S5D9 supports all the hardware cryptographic elements needed, it is recommended to leave all of them enabled for best performance.

The file `crypto_cfg.h` contains the macros that need to be defined to enable hardware support for SHA256, and ECC (signing and verification).

10.2.3 Enabling Debug Prints

To disable debug printouts on the SCI8 serial port, add the macro `DISABLE_PRINTF` to the compiler pre-processor by right clicking the **SecureBootloader** project in e² studio. Under **Settings>Cross ARM C Compiler**, add the macro. Doing this will significantly reduce the code footprint.

10.2.4 Configuring the OFS registers

The OFS register bits can be configured in the `mcu_rom_cfg.h` file. For details on these registers, see the hardware manual.

10.2.5 User-defined Functions

The **SecureBootloader** project has hooks the user can define to control the flow of the secure bootloader. The user can replace these functions as required by their application.

The functions are listed in `securityDebug.h` and `securityUser.h` with example implementations in `securityDebug.c` and `securityUser.c`.

10.3 Configuration Summary

Macro	Description	Configurable via	PC Project	e ² studio project
RX_1ST_BUILD	Used in the frameprotocol embedded implementation to determine that the currently running code is the 1stRX. Do not modify.	Project define	N/A	1ST_RX
ENABLE_WINDOWS_KEYSTORE	Replace the Excel sheet based key management with Microsoft CAPI. Do not modify.	Project define	KernelPackager, KernelInstaller, Mastering, Deploy	N/A
ENABLE_FLASH_ACCESS_WINDOW	Used to set the Flash Access Window.	Project define	KernelInstaller OR CertificateInstaller	N/A
ADDR_FAW_START ADDR_FAW_END	Configure the FAW range	memory Map.h	N/A	SecureBootloader
ACTIVATE_THE_FSPR	Lock the FSPR bit	Project define	KernelInstaller OR CertificateInstaller	N/A
ENABLE_QSPI_FLASH_DOWNLOAD_SUPPORT	Allow using QSPI as Update Area	Project define	Mastering, Deploy	All
ENABLE_QSPI_FLASH_USERDATA_SUPPORT	Support programming an app that uses QSPI as part of its application.	Project define	N/A	SecureBootloader
ENABLE_SBM_OEM_CERTIFICATE	Support Device Identity creation and programming	Project define	N/A	SecureBootloader, 1ST_RX
ENABLE_ENCRYPTION	Support Update-Image encryption/decryption	Project define	Deploy	SecureBootloader
ENABLE_JTAG_LOCKING	Set the JTAG locking ID	Project define	CertificateInstaller	N/A
ADDR_BOOTLOADER BOOTLOADER_FLASH	Start Address of the bootloader	memory Map.h memory Map.Id	All	All
ADDR_BOOTLOADER_MT MT_BOOTLOADER	Start Address of the bootloader module table	memory Map.h memory Map.Id	All	All
ADDR_1ST_RX FIRST_RX	Start Address of the 1 st Rx Code	memory Map.h memory Map.Id	All	All
ADDR_1ST_RX_MT FIRST_RX_MT	Start Address of the 1 st RXModule Table	memory Map.h memory Map.Id	All	All
ADDR_OEM_CERT OEM_CERT	Start Address of the area that holds the OEM Certificate	memory Map.h memory Map.Id	All	All
SIZE_OEM_CERT	Size of the OEM Cert Area. This must be of one flash block size.	memory Map.h	All	All
FLASH	Start of the Application	memory Map.Id	All	All
ADDR_MAIN_MT MT_MAIN	Start of the Application Module Table	memory Map.h memory Map.Id	All	All

ADDR_PENDING_UPDATE PENDING_UPDATE	Start of the location where the updates will be downloaded to	memory Map.h memory Map.Id	All	All
BOOTLOADER_RAM_START_ADDR BOOTLOADER_RAM	Start of the RAM used by the bootloader	memory Map.h memory Map.Id	All	All
APP_RAM_START_ADDR RAM	Start of the RAM available to the user app	memory Map.h memory Map.Id	All	All
BOOTLOADER_DF_START_ADDR BOOTLOADER_DF_END_ADDR BOOTLOADER_DATA_FLASH	Data Flash used by the Bootloader	memory Map.h memory Map.Id	All	All

11. Licenses

Third-party cryptographic code is used in this demonstration and is subject to the following licenses.

11.1 SHA256 + HMAC

Copyright © IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Internet Society, IETF or IETF Trust, nor the names of specific contributors, may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11.2 Elliptic Curve

Copyright © 2014, Kenneth MacKay. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Website and Support

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

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.16.18	—	Initial release
1.01	Oct.29.18	—	The Secure Boot Manager software included with this document has been updated to version v0.8.1. The content of this document has not changed.
1.02	Mar.01.19	—	Updated for v0.9.

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/.