RX Family

Renesas Firmware Update Over the Air (FOTA) Sample Program With Bluetooth Low Energy DA14535/DA14531

Introduction

This application note describes a sample program that implemented MCU Firmware Update Over the Air on FPB-RX261 Fast Prototyping Board with Bluetooth® Low Energy (LE) wireless communication Pmod™ Board US159-DA14535EVZ/DA14531EVZ module, utilizing Renesas SUOTA Service of the Bluetooth LE DA14535/DA14531 module.

Target Device

RX Family
RX200 Series
RX261 Group

Related Documents

[1] Firmware Integration Technology User's Manual (R01AN1833)

[2] Firmware Update Module Using Firmware Integration Technology (R01AN6850)

[3] FPB-RX261 v1 – User's Manual (R20UT5363)

[4] US159-DA14535EVZ Evaluation Board Manual (R15UZ0014)

[5] US159-DA14531EVZ Evaluation Board Manual (R15UZ0004)

[6] DA1468x Software Developer's Guide User Manual (UM-B-056)

[7] Getting started with DA1453x and RX BLE Framework on Renesas Microcontrollers (UM-B-177)

Terminology/Abbreviation

Term	Description
FW Update Module	Firmware Update Module (FWUP FIT Module)
BLE Control Module	US159-DA1453xEVZ BLE Control Module (DA1453x BLE FIT Module) Reference: https://www.renesas.com/document/apn/rx-family-us159-da1453xevz-ble-control-module-using-firmware-integration-technology-application-note 4.6.10 ROM/RAM Usage also includes the r_ble_da1453x_rx module
FOTA	Firmware Update Over the Air
SUOTA Service	SUOTA GATT service
SUOTA Mobile Application (SUOTA Mobile App)	The mobile application used to perform SUOTA
SUOTA Application program	Firmware Update program (fwup_main program) refer to "5.1 Project Description"
SUOTA sample program	Include bootloader, fwup_leddemo, fwup_main programs refer to "5.1 Project Description"
SUOTA Reference Application	Refer to "4.5 Services" section
SUOTA Service Server	DA14535/DA14531 acting as the SUOTA server

Contents

Trademarks

Bluetooth® is a trademark of the Bluetooth SIG, Inc.

Pmod™ is a trademark of Digilent Inc.

1. Overview

This application note describes a sample program demonstrating the integration of Bluetooth LE connectivity for MCU Firmware Update Over-the-Air (FOTA) on the FPB-RX261 development board. This process is called Software Upgrade Over the Air (SUOTA) and is simple enough to be performed by the end user.

Key Steps in the Project:

- **Generate Key Pairs and Certificates:** Generate a public/private key pair for signing both the initial firmware and the firmware used for updates.
- **Generate and Upload the Firmware File:** Generate the new firmware to be programmed onto the board via OTA and upload it to the SUOTA Mobile Application.
- **Generate the initial firmware:** Configure and build the initial firmware image using the provided tool.
- **Execute Demonstration Project:** Execute the sample project to validate the FOTA process.

The following section provides a Bluetooth LE stack-related structure used in the OTA demonstration.

DA14535/DA14531 Bluetooth LE Module with SUOTA: The DA14535/DA14531 module connects to a mobile device via Bluetooth LE and receives firmware updates using the SUOTA Service. The firmware is transmitted from the mobile application and programmed into the MCU host for deployment.

Figure 1.1 illustrates the update procedure, which is initiated from SUOTA Mobile App, a new image is first transferred to FW Update Module. The image data is received over the communication interface, self-programming it into the on-chip flash memory of the target device through the FW Update Module and flash memory driver and then the device reboots to complete the update.

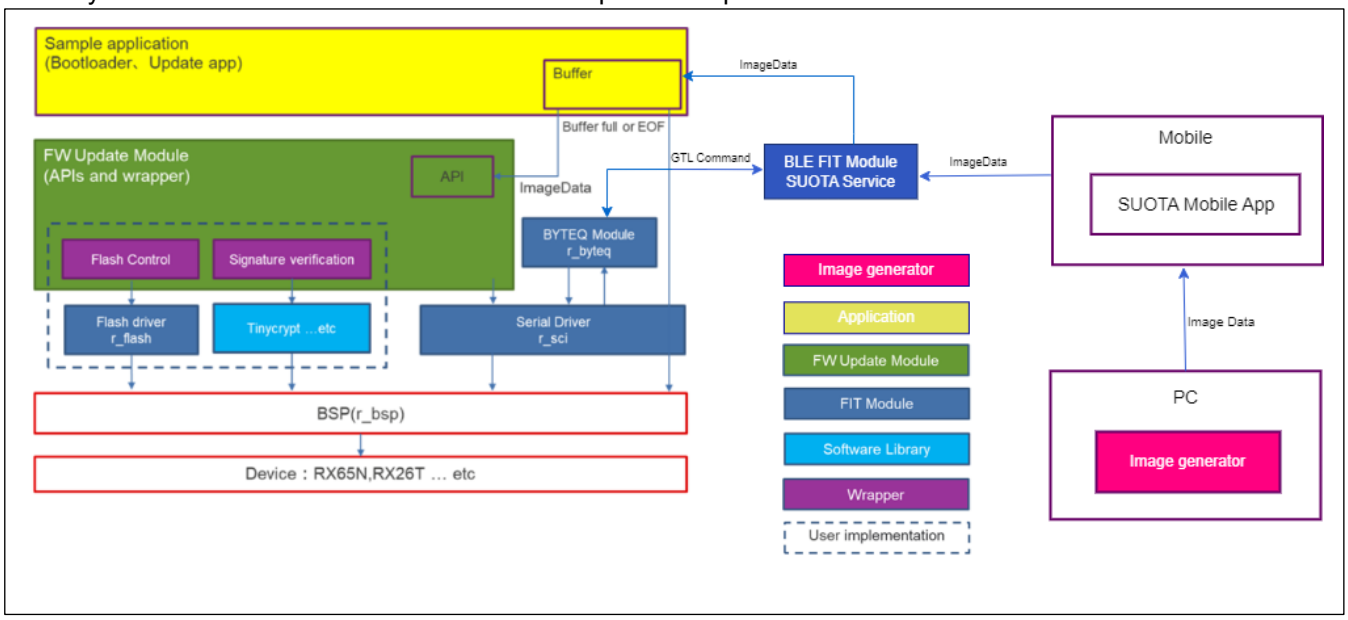


Figure 1.1 OTA Demonstrates with Bluetooth LE DA14535/DA14531

2. Operation Confirmation Conditions

Demo project operations have been confirmed in the following conditions.

Table 2.1 Operation Confirmation Conditions

Item	Description
MCU	R5F52618BGFP (RX261 Group)
Operating frequency	System Clock (ICLK) : 64MHz Peripheral Module Clock B (PCLKB) : 32MHz
Operating voltage	MCU: 3.3V Board: 5V
IDE (Integrated Development Environment)	Renesas Electronics e2 studio Version 2025-04.1 (25.4.1)
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Smart Configurator	Renesas Smart Configurator for RX 2.25.0
Endian order	Little endian
Firmware programming tool	Renesas Flash Programmer V3.20.00
Python	Python 3.12.7
Terminal software	Tera Term V5.0
Keygen tool	Win64 OpenSSL v3.0.12
SDK (Software Development Kit)	DA14535/DA14531 SDK V6.0.24
SUOTA Mobile Application	V3.190.24
Sample program version	V1.00
Emulator	E2 Emulator Lite on Board
Board used	Renesas Fast Prototyping Board FPB-RX261 v1 (Part number: RTK5FP2610S00001BE) (*) Bluetooth Low Energy Pmod Board (US159-DA14535EVZ Rev.C/US159-DA14531EVZ Rev.B)
Optimization level	Refer section “4.6.3 Project Optimization”

(*) Note: If your DA1453x device is an older revision that requires firmware updating via the SmartBond Flash Programmer (which typically requires a J-Link), we recommend obtaining the latest DA1453x device to ensure the latest firmware, compatibility, and avoid the need for additional programming tools.

3. Sample Program Preparation

3.1 Equipment List

The following lists the equipment required for the demo project.

Table 3.1 Equipment List

Item	Description
Board	FPB-RX261 v1 https://www.renesas.com/fpb-rx261
US159-DA14535EVZ module	Bluetooth Low Energy Pmod Board https://www.renesas.com/us159-da14535evz
US159-DA14531EVZ module	Bluetooth Low Energy Pmod Board https://www.renesas.com/us159-da14531evz
USB-UART conversion board	USB to UART Interface Module https://www.digikey.com/USB-UART-Conversion-Board
Micro USB Type-B cable	Connect another USB port on the base board to a PC for debugging, power supply and log output purposes.
Jumper pin x 2	Used to enable debugging mode
Jumper wire x 3	Used to connect the USB-UART conversion board to the MCU board
Smartphone	Android V4.4 or higher

3.2 Hardware Setup

This section shows how to set up the hardware to run this sample.

The following shows the overall configuration of hardware that makes up the sample project.

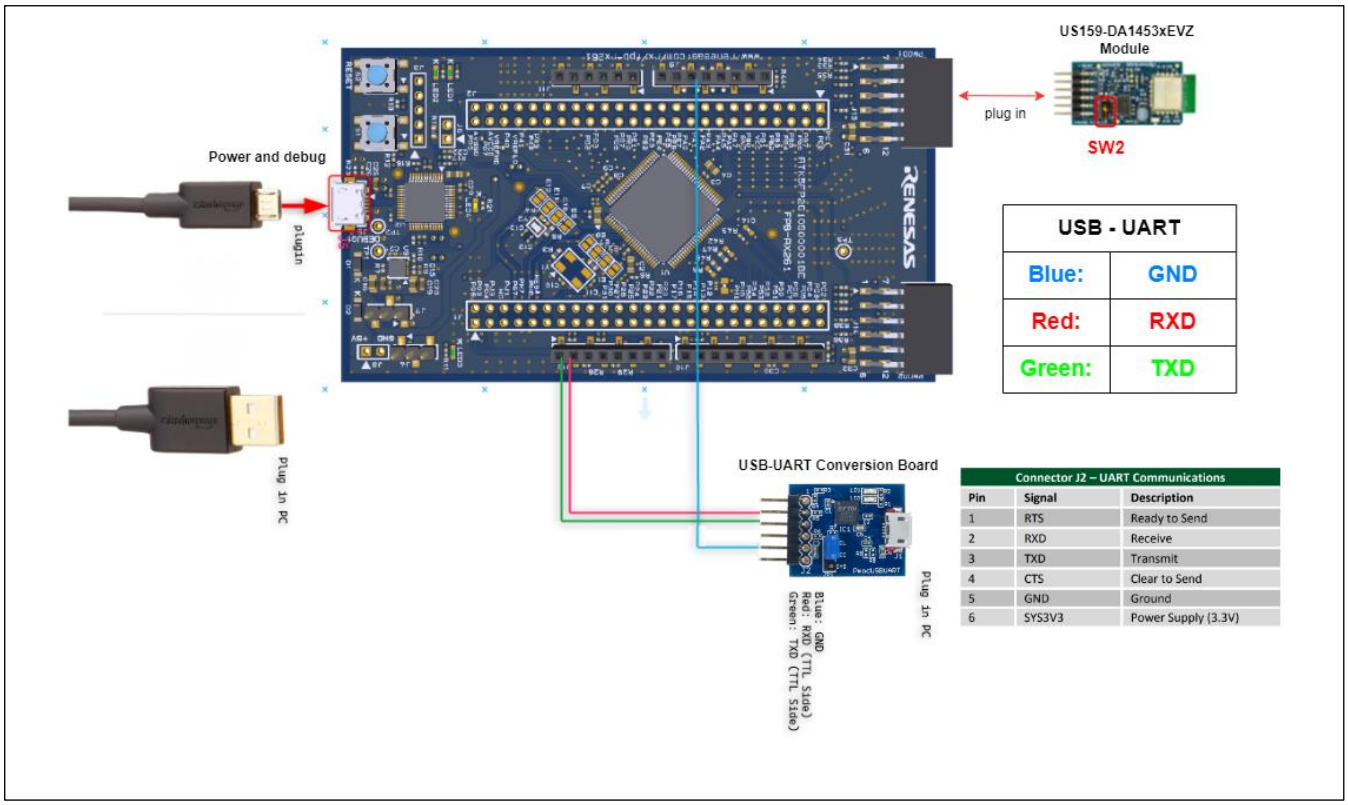


Figure 3.1 Hardware Setup

- Connect the US159-DA1453xEVZ module to the FPB-RX261 **PMOD1** connector.
- Boot from Host supports two modes for US159-DA1453xEVZ: **1-Wire** and **2-Wire**.
  - With DA14535:
    Set the SW2 on the module **OFF** to support 2-Wire UART mode.
  - With DA14531:
    Connect Tx/Rx with a **1K Ohm resistor** to support 1-Wire UART mode.
- Connect a Micro USB Type-B cable from the PC to the FPB-RX261 micro-USB connector (**J5**) for power supply and debugging.
- Connect a USB-UART Conversion Board from the PC to the FPB -RX261 SCI12 (**J12**) for serial log output.
- Set **JP1** of USB-UART Conversion Board to **LCL-VCC**. In this setting, the USB-UART Conversion Board is powered by USB bus.

3.2.1 Connection with DA14535

Connect the DA14535 BLE to PMOD 1 of FPB-RX261. The connection of DA14535 to FPB-RX261 is shown below.

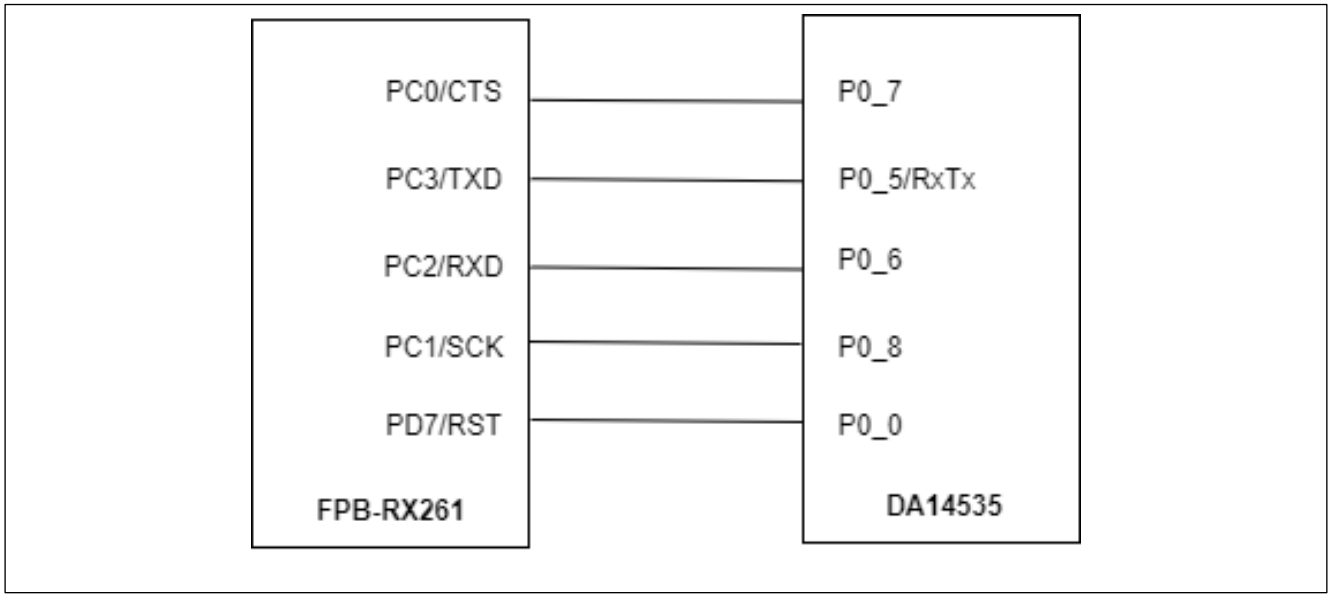


Figure 3.2 Connection with DA14535

3.2.2 Connection with DA14531

Due to the 1-Wire UART connection, the DA14531 cannot be connected directly to the PMOD1 port of the FPB-RX261 board. Instead, it must be connected by using external wires, swap the P0_5 and P0_6 pins of the DA14531, and add a 1 kΩ resistor between the RX/TX line to ensure stable signal operation. The connection to the DA14531 BLE with Boot from Host 1-wire UART is shown below.

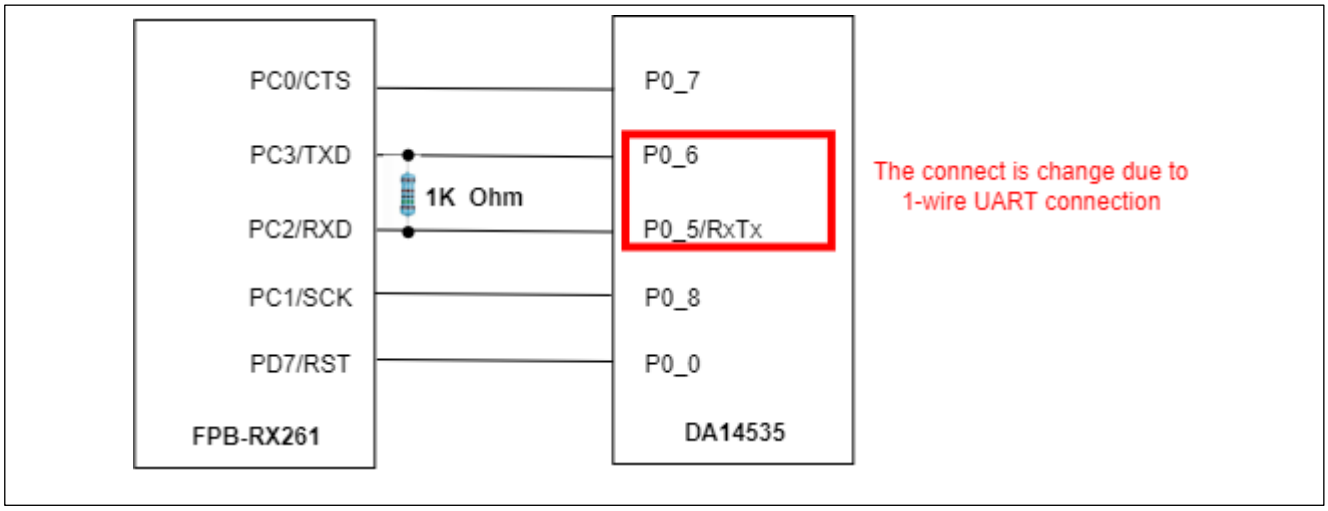


Figure 3.3 Connection with DA14531

3.3 Software Setup

3.3.1 Installing Tool

3.3.1.1 Install Python

Python generates initialization firmware from bootloader and application projects, and application firmware from the new application project.

Follow the steps below to install Python:

(1) Access the Python download web site.

https://www.python.org/downloads/

(2) Download the Python 3.12.7 installer.

Click the **Download** link for Python 3.12.7.

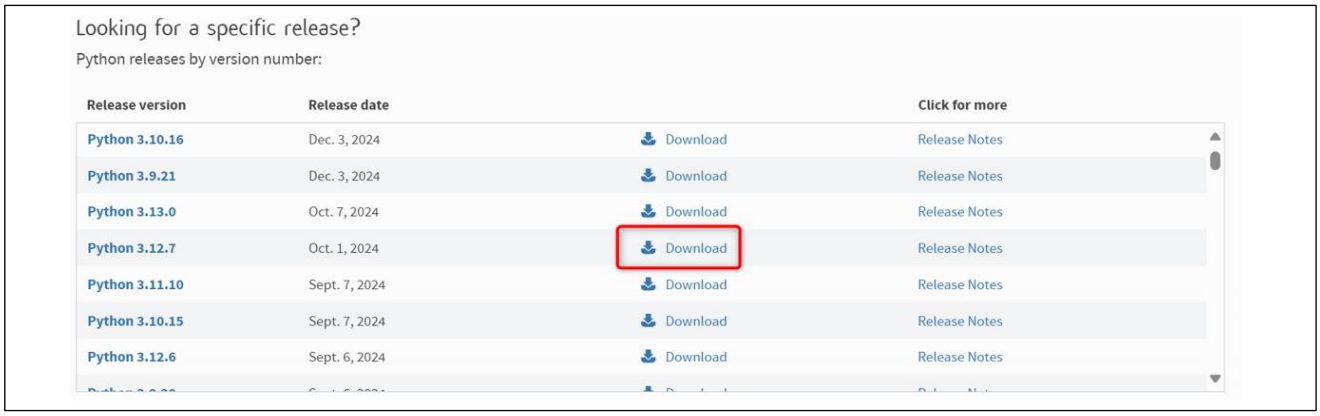


Figure 3.4 The Options for Installing the Release Version of Python

Download the installer for the operating system you are using.

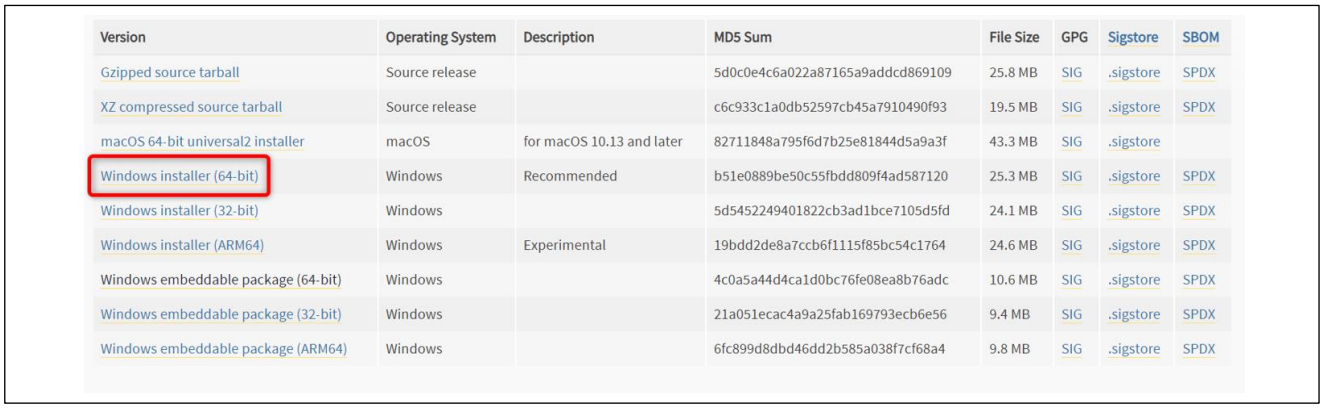
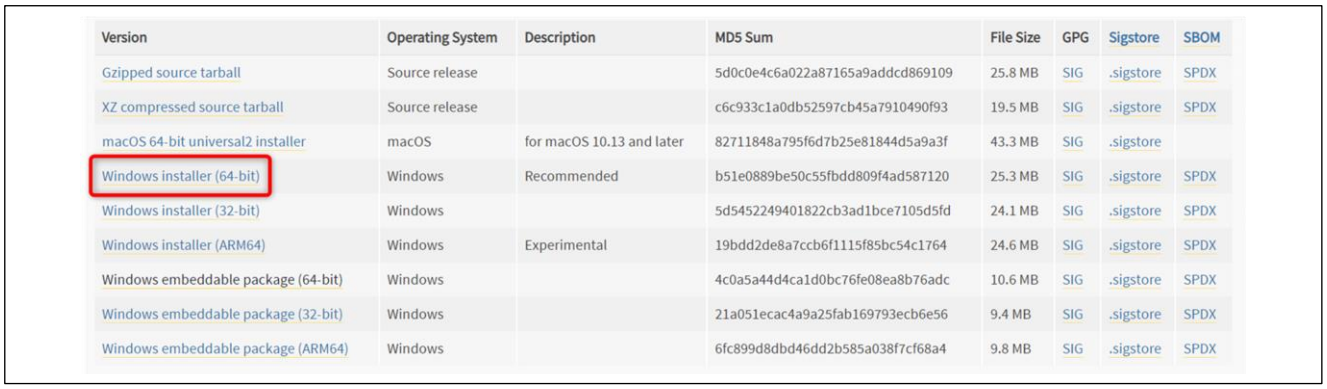


Figure 3.5 Python Windows Installer

(3) Run the installer and follow the prompts to install Python

On the installation screen, select the **Add python.exe to PATH** check box.



Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore	SBOM
Gzipped source tarball	Source release		5d0c0e4c6a022a87165a9addcd869109	25.8 MB	SIG	.sigstore	SPDX
XZ compressed source tarball	Source release		c6c933c1a0db52597cb45a7910490f93	19.5 MB	SIG	.sigstore	SPDX
macOS 64-bit universal2 installer	macOS	for macOS 10.13 and later	82711848a795f6d7b25e81844d5a9a3f	43.3 MB	SIG	.sigstore	
Windows installer (64-bit)	Windows	Recommended	b51e0889be50c55fbdd809f4ad587120	25.3 MB	SIG	.sigstore	SPDX
Windows installer (32-bit)	Windows		5d5452249401822cb3ad1bce7105d5fd	24.1 MB	SIG	.sigstore	SPDX
Windows installer (ARM64)	Windows	Experimental	19bdd2de8a7ccb6f1115f85bc54c1764	24.6 MB	SIG	.sigstore	SPDX
Windows embeddable package (64-bit)	Windows		4c0a5a44d4ca1d0bc76fe08ea8b76adc	10.6 MB	SIG	.sigstore	SPDX
Windows embeddable package (32-bit)	Windows		21a051ecac4a9a25fab169793ecb6e56	9.4 MB	SIG	.sigstore	SPDX
Windows embeddable package (ARM64)	Windows		6fc899d8dbd46dd2b585a038f7cf68a4	9.8 MB	SIG	.sigstore	SPDX

Figure 3.6 Python 3.12.7 Installer

(4) Install the Python encryption library (pycryptodome)

Install the encryption library by executing the following command: *$ pip install pycryptodome*

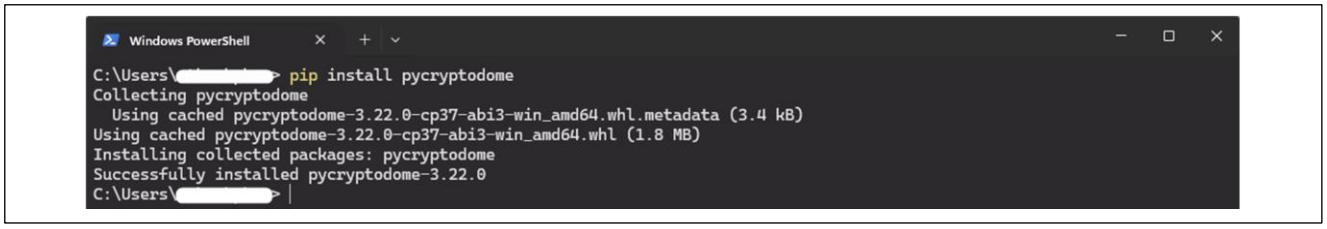


Figure 3.7 Installing Python Encryption Library

3.3.1.2 Installing OpenSSL

OpenSSL is a tool used to generate the cryptographic key pair required for firmware encryption and decryption during initialization and application firmware creation. OpenSSL can generate the following keys for use in the firmware update process:

- **Private key**: Used to encrypt the firmware and ensure its integrity.
- **Public key**: Used by the bootloader to decrypt and verify the firmware during update.

Follow the steps below to install and configure OpenSSL for this purpose.

(1) Access the Win32/Win64 Download Website for OpenSSL

https://slproweb.com/products/Win32OpenSSL.html

(2) Download the OpenSSL Installer

Download the installer for the operating system you are using.

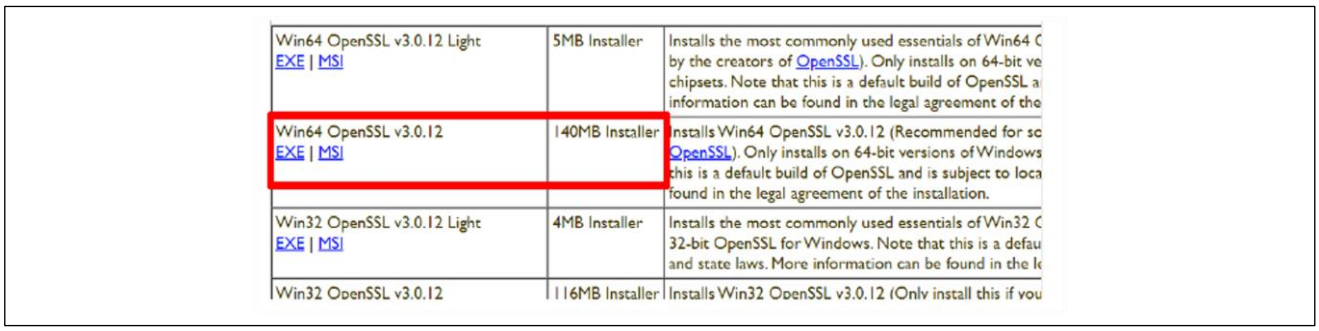


Figure 3.8 The Options for Installing the Release Version of OpenSSL

(3) Run the Installer and Follow the Prompts to Install OpenSSL.

Select the option to copy the OpenSSL DLLs to the OpenSSL binaries directory.

(4) From the Start Menu, Open the Win64 OpenSSL Command Prompt.

Figure 3.9 OpenSSL Windows (64-bit)

(5) Confirm the OpenSSL Command from the Command Prompt.

Execute the following command (“openssl version”) and confirm that version information appears.

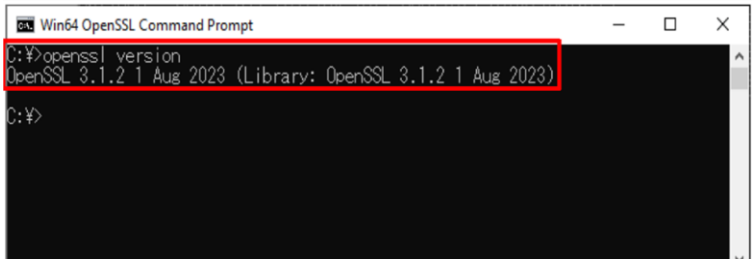


Figure 3.10 Checking OpenSSL Version

3.3.1.3 Installing Renesas Image Generator

Renesas Image Generator is a tool that generates the firmware images used by the Firmware Update Module.

Renesas Image Generator can generate the following images for use by the Firmware Update Module:

- Initial image: An image file containing the bootloader and application program written by flash writer during initial system configuration (extension: mot).
- Update image: An image file containing the updated firmware (extension: rsu).

Renesas Image Generator is provided as part of the Firmware Update Module.

(1) Download the Firmware Update Module:

https://www.renesas.com/document/scd/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes-rev204

(2) Extract the Downloaded Firmware Update Module

Extract the file RenesasImageGenerator.zip in the Firmware Update Module.

The **RenesasImageGenerator** folder contains the Renesas Image Generator script file (image-gen.py) and the parameter files for various devices (*_ImageGenerator_PRM.csv).

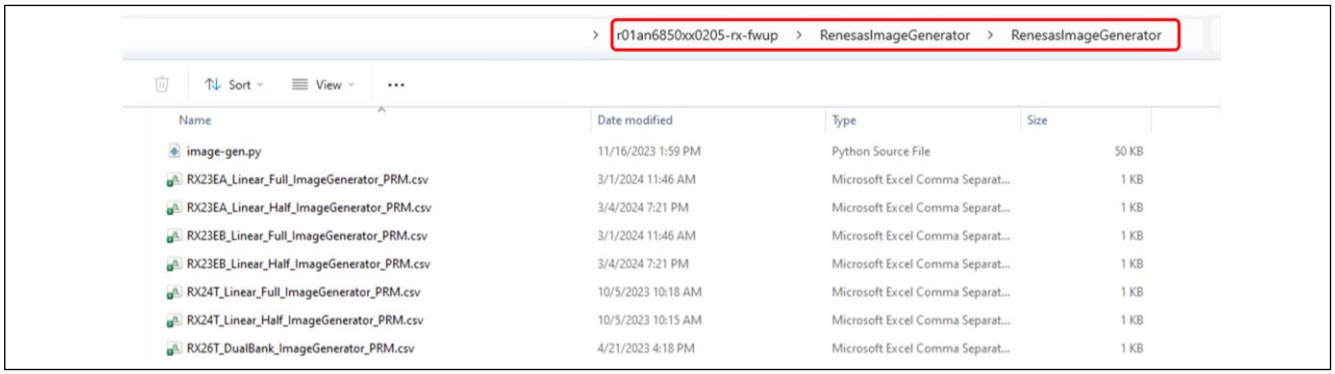


Figure 3.11 Renesas Image Generator Package

3.3.1.4 Installing Tera Term

Terminal software (example: Tera Term) is required to output demo project logs. The following show the serial port settings.

(1) Access the Tera Term Download Site.

https://github.com/TeraTermProject/osdn-download/releases

(2) Download the Tera Term Installer.

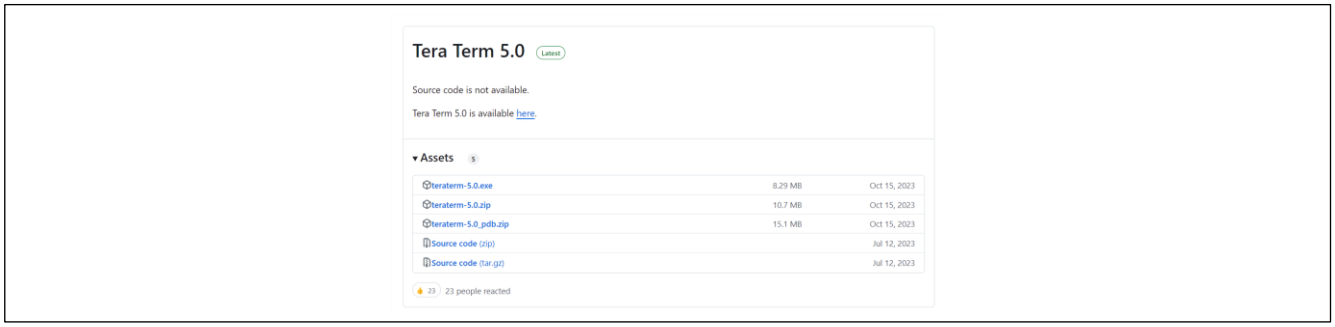


Figure 3.12 Tera Term Version 5.0

- Run the installer and follow the prompts to install Tera Term.
- Confirm that Tera Term starts when you click the Tera Term icon in the Start menu.

3.3.1.5 Installing Renesas Flash Programmer

Renesas Flash Programmer (RFP) is a utility provided by Renesas that allows users to write firmware to support Renesas MCUs via various interfaces such as USB, UART, or serial programming. It is an essential tool for programming both the initial firmware and subsequent updates during development and production. Follow the steps below to install Renesas Flash Programmer on your computer.

(1) Access the Renesas download web site.

https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui

Figure 3.13 Renesas Flash Programmer

3.3.1.6 Installing SUOTA Mobile App

Install the https://play.google.com/store/apps/details?id=com.dialog.suota

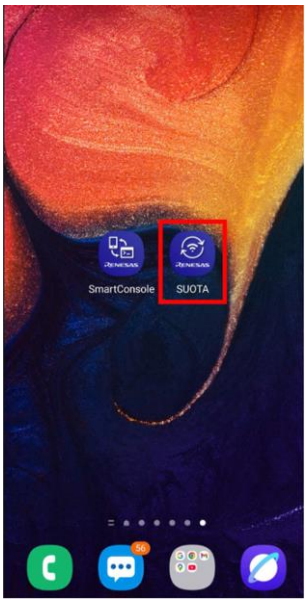


Figure 3.14 SUOTA Mobile Application

3.3.2 Terminal Software Setting

With the USB-UART conversion board connection port:

(1) Open Tera Term select **New connection** and select Serial and the appropriate COM port for your **UART-to-USB** adapter, and **click OK**

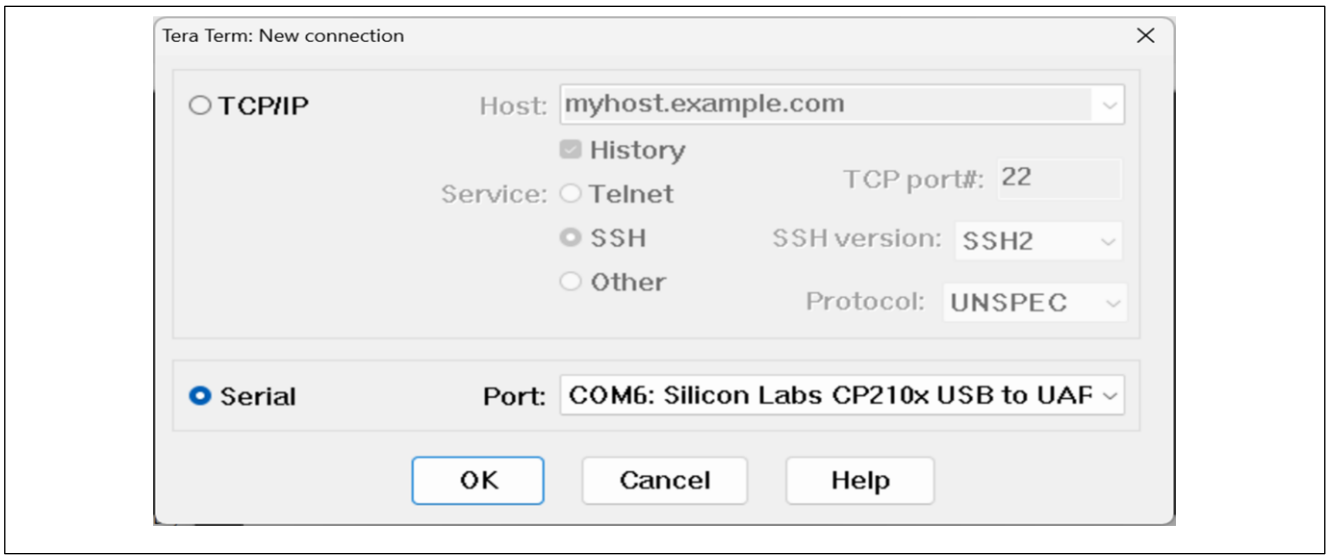


Figure 3.15 Tera Term Serial Connection

(2) Click **Setup > Terminal**…, in "**New-line"** section, set **"Receive**" as **AUTO.**

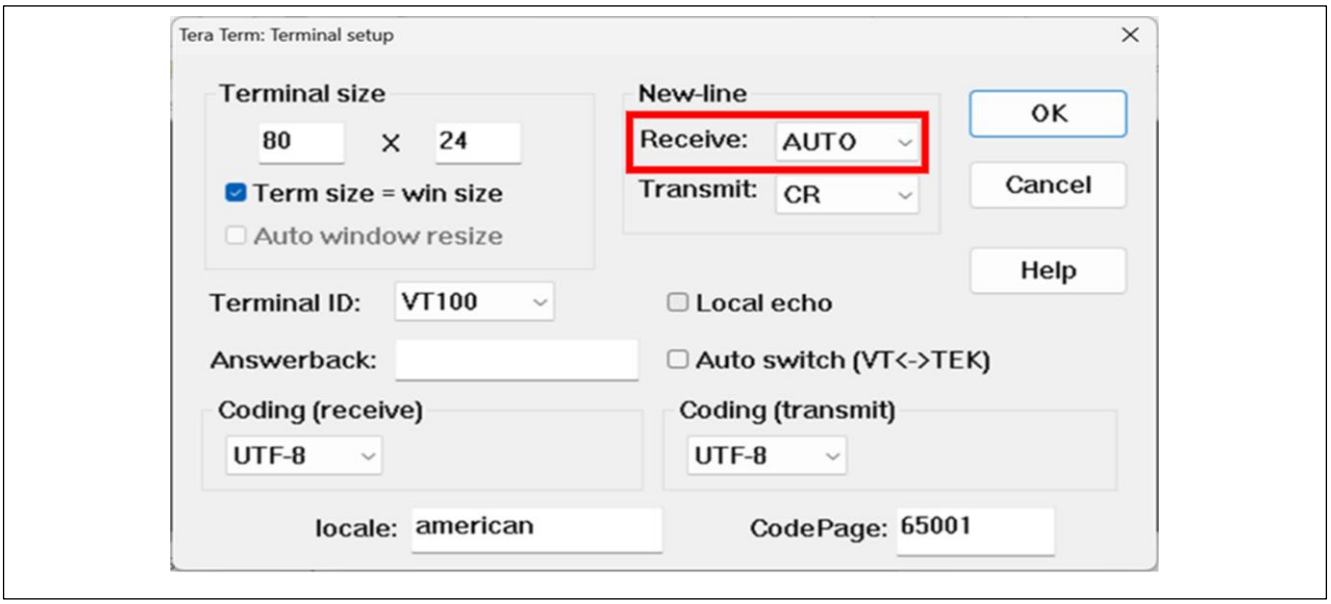


Figure 3.16 Terminal Setup for the USB-UART Conversion Board

(3) Click **Setup > Serial port…** and ensure that the speed is set to **115200**.

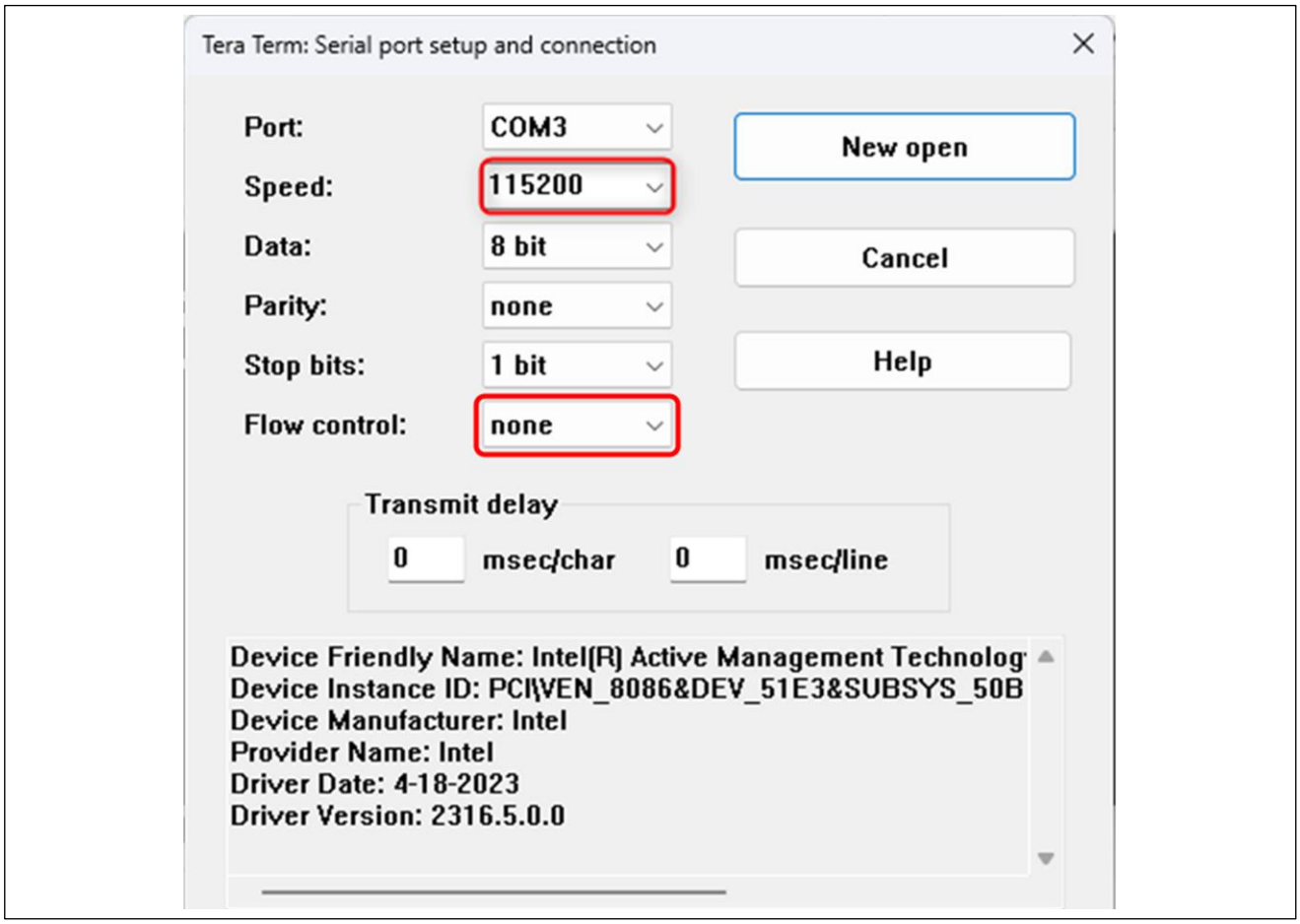


Figure 3.17 Serial Port Setup for USB-UART

3.3.3 Generate Key Pairs and Certificates

This section will generate a public key and a private key to create the initial firmware and the firmware used for updates.

To do this, open OpenSSL and enter the commands highlighted in yellow to generate the firmware verification keys.

```
openssl ecparam -genkey -name secp256r1 -out secp256r1.keypair

using curve name prime256v1 instead of secp256r1

openssl ec -in secp256r1.keypair -outform PEM -out secp256r1.privatekey

read EC key

writing EC key

openssl ec -in secp256r1.keypair -outform PEM -pubout -out secp256r1.publickey

read EC key

writing EC key
```

4. Sample Program

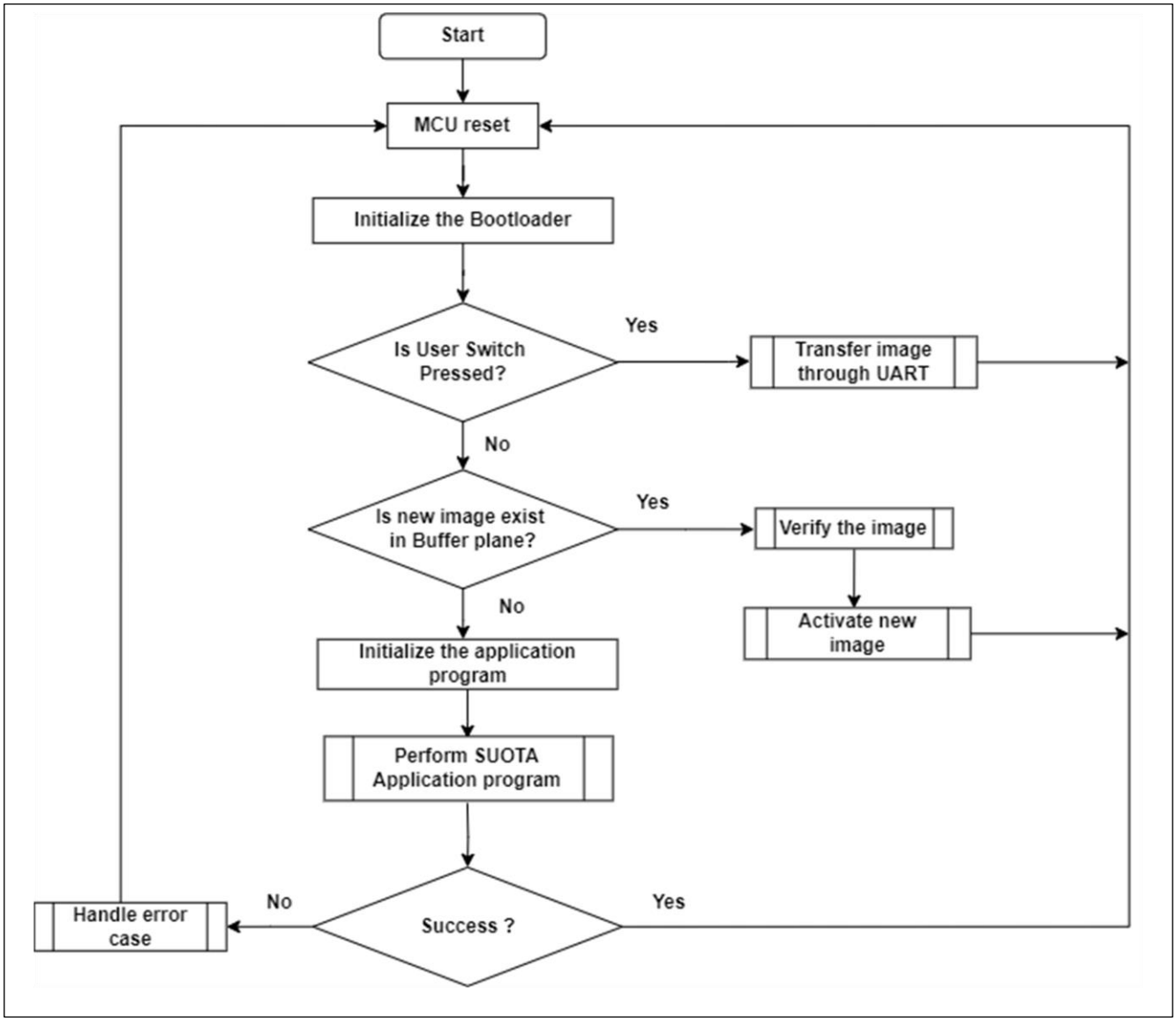


Figure 4.1 Sample Program Flowchart

Generally, a firmware update system comprises two programs: an application program providing SUOTA functionality to update firmware and a bootloader providing secure boot functionality used to validate the first program.

- **Bootloader**

The bootloader functionality is essential to the proper functioning of the firmware update. It guarantees that the sequence of processing that composes the firmware update, including validation of the update image, is legitimate.

- **SUOTA Application program**

The SUOTA process begins with the execution of the SUOTA Application program. As a result of this during the update, the new application image is stored in a separate location in Flash memory. After the new image is transferred and verified successfully, the device reboots to complete the update. After the reboot is completed, the bootloader transfers the image from the buffer plane to the main plane and executes it.

4.1 Section Layout

For details of section layout, please refer to “Firmware Update Module Using Firmware Integration Technology” (R01AN6850).

4.1.1 Operation of Linear Mode Partial Update Method

This method divides the on-chip flash memory into a main plane(*) and a buffer plane(*) and then temporarily stores the update image in the buffer plane, as shown in Figure 4.2. Firmware is updated by storing the update image on the buffer plane and copying it from the buffer plane to the main plane.

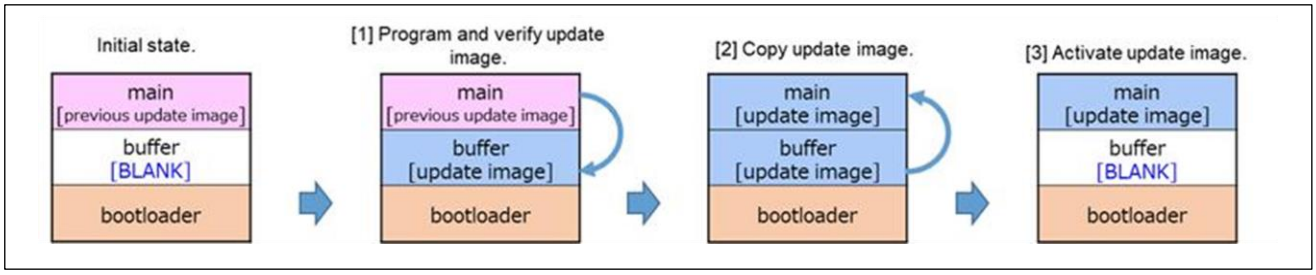


Figure 4.2 Operation of Linear Mode Partial Update Method

(*) Note:

- Main plane: Area for storing the image used for booting
- Buffer plane: Area for storing the image to be applied as an update

4.1.2 Memory Map of Demo Project in Linear Mode

Figure 4.3 shows the memory map of the FPB-RX261 sample project and the memory map of the configuration settings.

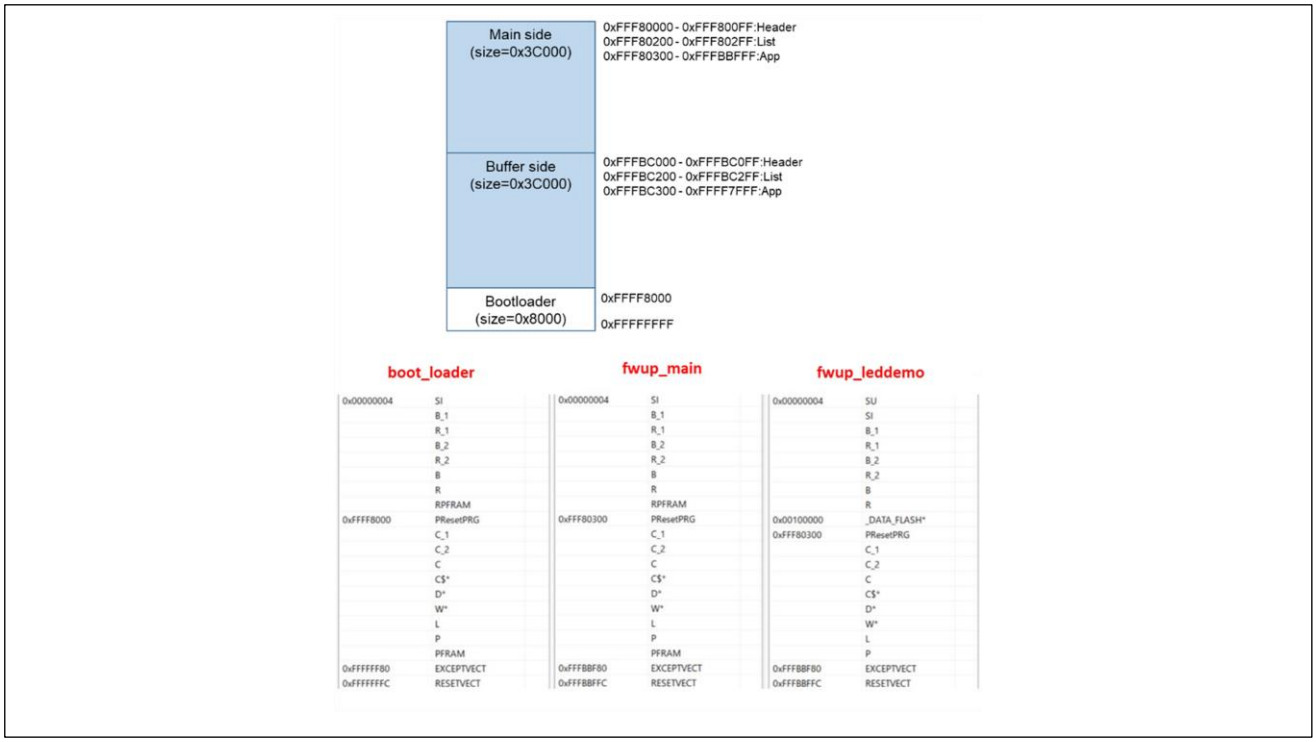


Figure 4.3 FPB-RX261 Project Memory Map

Note:

- boot_loader, fwup_main and fwup_leddemo project descriptions refer to “5.1 Project Description” section.

4.2 Package Contents

The SUOTA Sample Program package contains several files. These are listed in the Table 4.1 below

Table 4.1 Folder Structure of SUOTA Sample Program Package

Folder name	Description
r20an0825ea0100-rx261-suota.zip	
├ doc	Documentation
│ └r20an0825ea0100-rx261-suota.pdf	Sample program application note
├ modules	Sample program
│ ├ boot_loader.c	boot_loader.c
│ ├ fwup_main.c	fwup_main.c
│ └ my_flash.c	my_flash.c
├ DA14535	US159-DA14535EVZ module
│ └ rx261-fpb	Fast Prototyping Board for RX261 MCU
│ └ w_buffer	Linear Mode Partial Update Method
│ └ e2_ccrx	CC-RX version
│ ├ boot_loader	Bootloader
│ ├ fwup_leddemo	Update application
│ └ fwup_main	User application including SUOTA
└ DA14531	US159-DA14531EVZ module
└ rx261-fpb	Fast Prototyping Board for RX261 MCU
└ w_buffer	Linear Mode Partial Update Method
└ e2_ccrx	CC-RX version
├ boot_loader	Bootloader
├ fwup_leddemo	Update application
└ fwup_main	User application including SUOTA

4.3 Overall SUOTA Application Program Sequence Diagram

Figure 4.4 shows the client connects through the SUOTA Mobile App. The Mobile App queries the **Device Information Service (DIS)** to read the information of device.

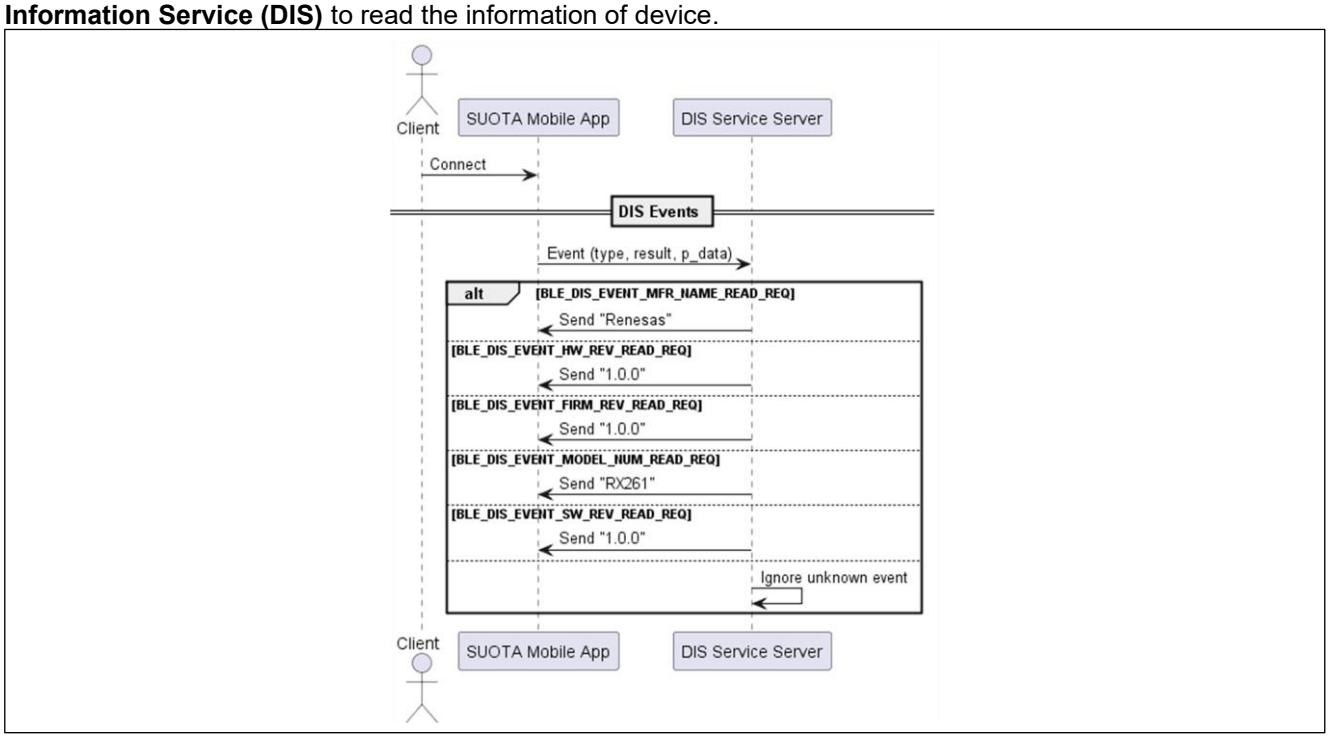


Figure 4.4 Device Information Service Sequence Diagram

Figure 4.5 illustrates at the beginning of the transfer process, the **SUOTA Mobile App** sends different event requests to the **SUOTA Service Server**. Depending on the request type:

- The server may restart the Firmware Update Module and return the SUOTA version.
- Initializes the SUOTA state and buffers, then returns the patch data characteristic length value.
- Handles MTU exchange and return the MTU value.
- Finally, the client sends the image data to the SUOTA Mobile App.

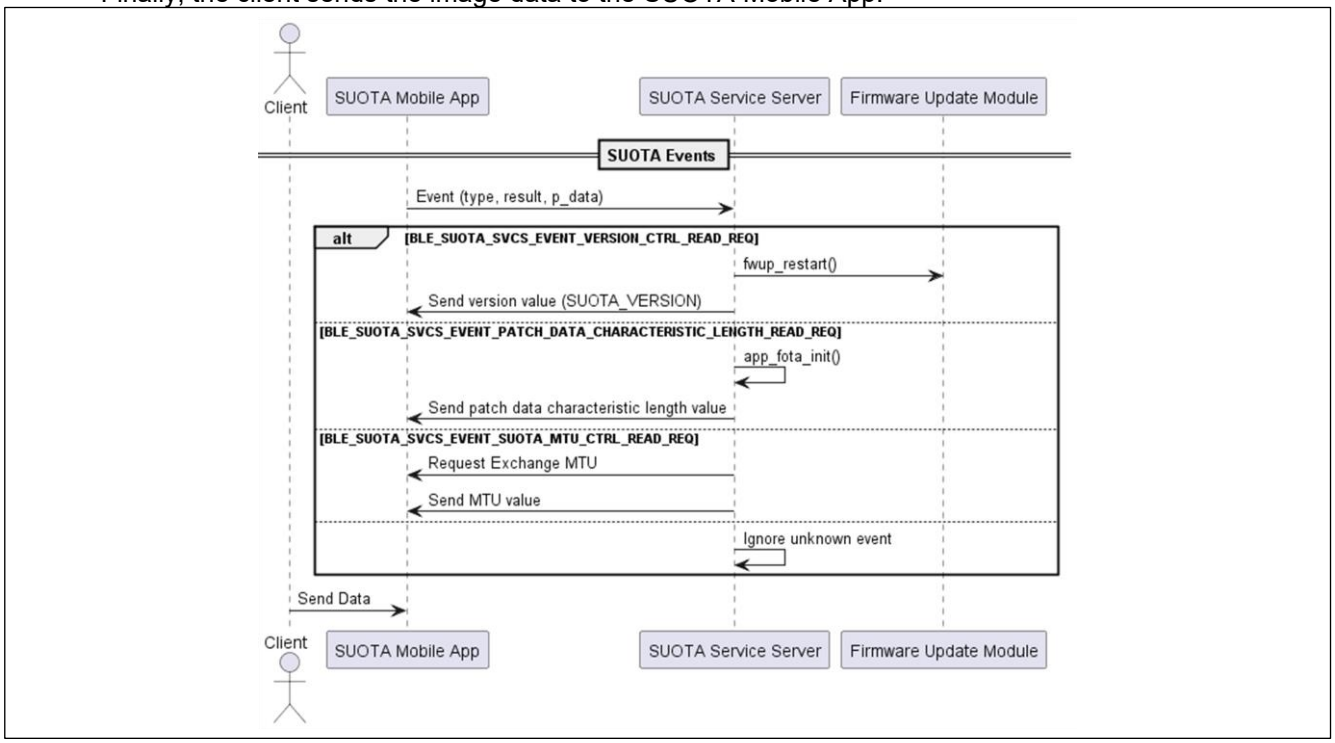


Figure 4.5 Initial Transfer Process Sequence Diagram

When the **SUOTA Service Server** receives a **memory device control write request** as shown in Figure 4.6, it ensures proper handling of the memory device operations, including initialization, CRC validation, reboot, and error reporting.

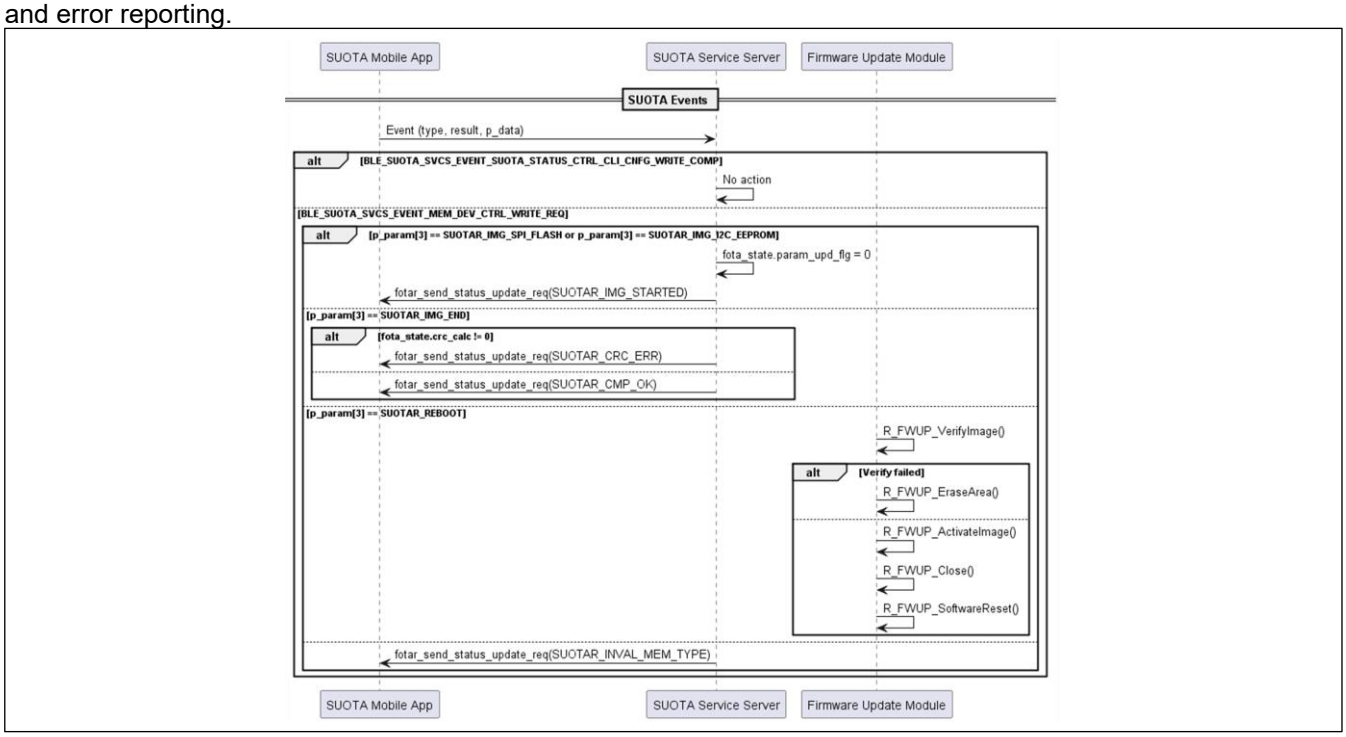


Figure 4.6 Memory Device Control Sequence Diagram

Figure 4.7 presents the process when the **SUOTA Mobile App** sends patch data, the **SUOTA Service Server** handles two events:

- **BLE_SUOTA_SVCS_EVENT_PATCH_LEN_CTRL_WRITE_REQ**: Handles the block size.
- **BLE_SUOTA_SVCS_EVENT_PATCH_DATA_CTRL_WRITE_REQ:** Handles incoming data chunks, calculates the CRC, writes the data to flash memory, and finally sends a SUOTAR_CMP_OK status back to the mobile app to proceed with the next block of the image.

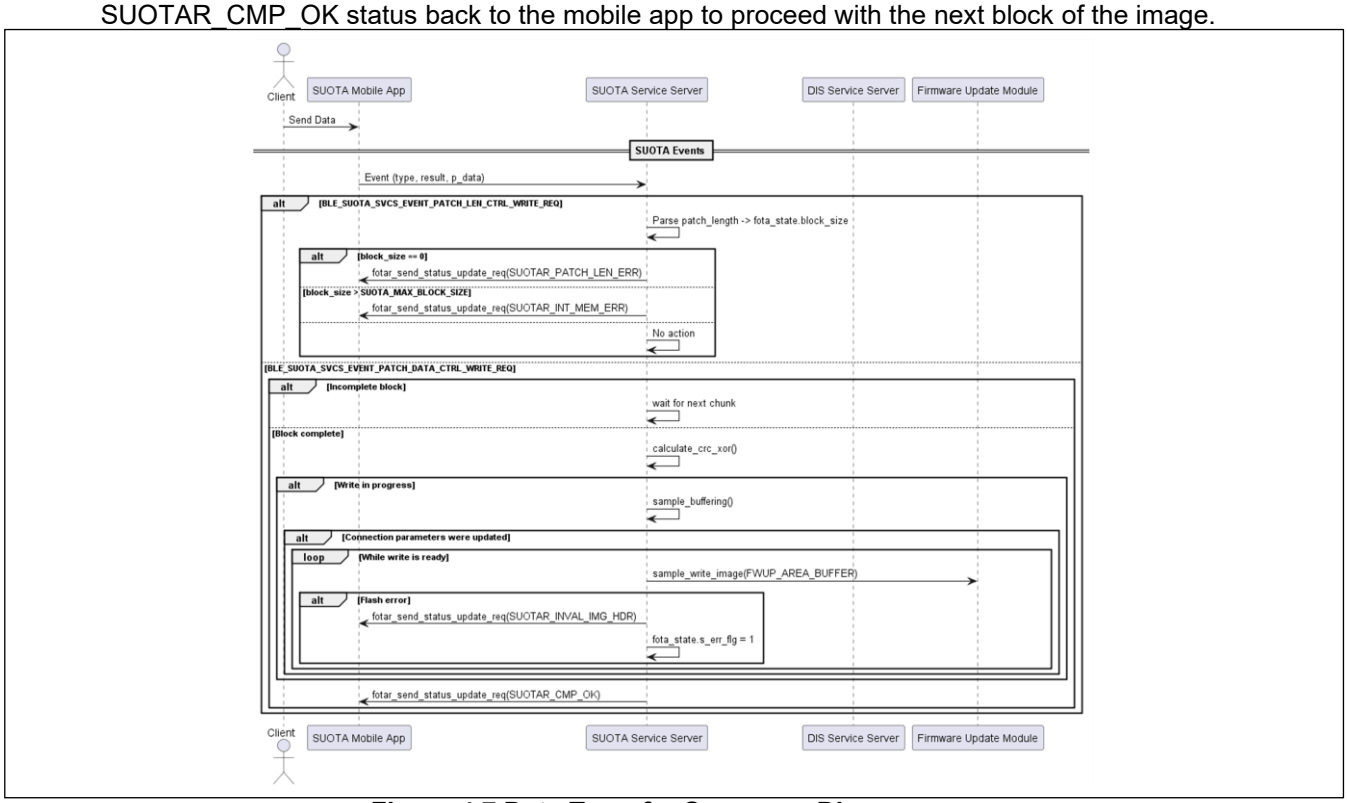


Figure 4.7 Data Transfer Sequence Diagram

4.4 Flow Chart of the SUOTA Application Program

4.4.1 Processing When Connecting to the SUOTA Mobile App

When the SUOTA Mobile App establishes a connection with the BLE device as shown in Figure 4.8, the phone queries several standard characteristics defined in the **Device Information Service (DIS)**.These read requests are sent by the phone to obtain device identification and firmware details.

On the DA14535/DA14531 side, each incoming read request event is handled within the event dis callback. For every received request, the application responds by calling app_read_cfm() with the corresponding handle, which sends the requested attribute value back to the phone.

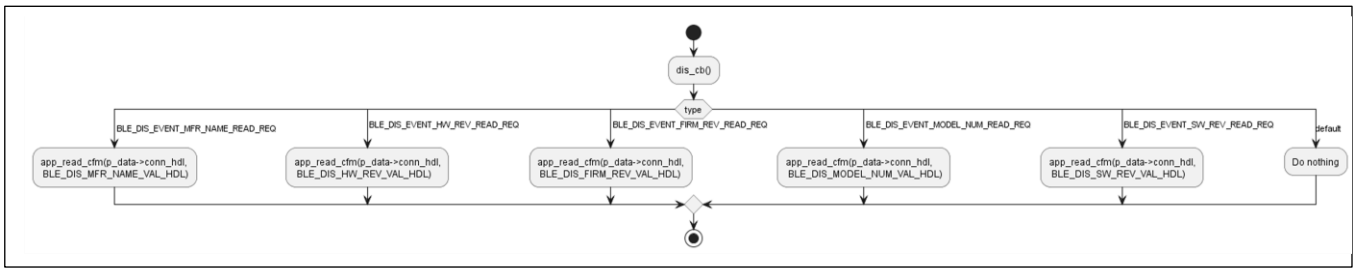


Figure 4.8 Device Information Service (DIS) Read Request Handling Flow Chart

- **Manufacturer Name:** handled by BLE_DIS_EVENT_MFR_NAME_READ_REQ, reply with BLE_DIS_MFR_NAME_VAL_HDL**.**
- **Hardware Revision**: handled by BLE_DIS_EVENT_HW_REV_READ_REQ, reply with BLE_DIS_HW_REV_VAL_HDL.
- **Firmware Revision**: handled by BLE_DIS_EVENT_FIRM_REV_READ_REQ, reply with BLE_DIS_FIRM_REV_VAL_HDL.
- **Model Number**: handled by BLE_DIS_EVENT_MODEL_NUM_READ_REQ, reply with BLE_DIS_MODEL_NUM_VAL_HDL.
- **Software Revision**: handled by BLE_DIS_EVENT_SW_REV_READ_REQ, reply with BLE_DIS_SW_REV_VAL_HDL.

4.4.2 Initial Transfer Events

Figure 4.9 describes the initial handling process when starting an image transfer after connecting SUOTA

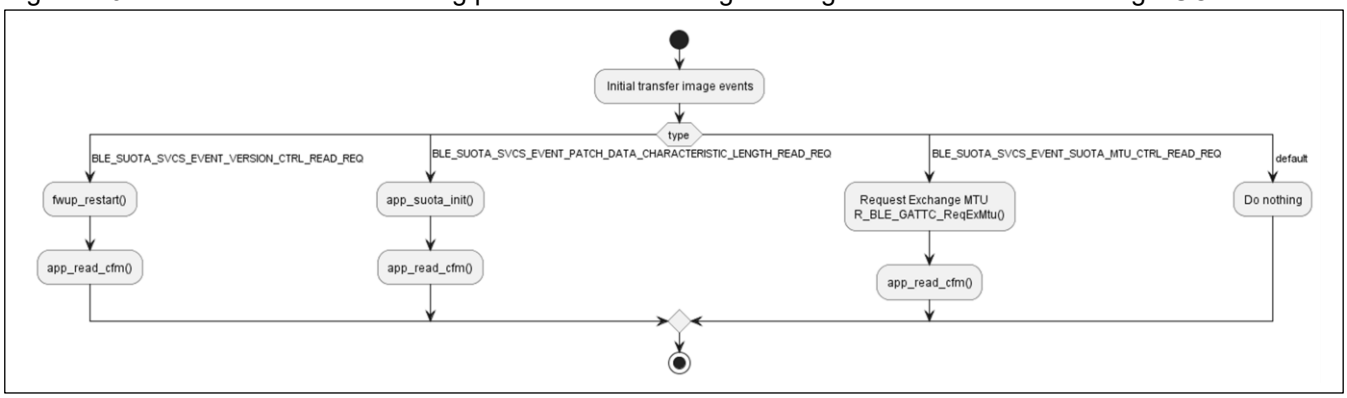


Figure 4.9 Initial Transfer Events

- The flow begins with these initial events during the transfer process.
- The application checks the **event type (type)** and executes the corresponding action:
  - **BLE_SUOTA_SVCS_EVENT_VERSION_CTRL_READ_REQ**: The firmware update process is restarted by calling **fwup_restart()**, followed by **app_read_cfm()** to confirm the version request.
  - **BLE_SUOTA_SVCS_EVENT_PATCH_DATA_CHARACTERISTIC_LENGTH_READ_REQ** : The SUOTA state and buffers is initialized using **app_SUOTA_init()**, then **app_read_cfm()** is called to confirm patch data length.
  - **BLE_SUOTA_SVCS_EVENT_SUOTA_MTU_CTRL_READ_REQ**: The application requests an **MTU exchange** with **R_BLE_GATTC_ReqExMtu()**, followed by **app_read_cfm()** for MTU confirmation.
  - **Default case**: If the event type does not match any of the above, the system performs no action and continues to process next step.

4.4.3 Processing When Writing Mem-Dev Info

Once the SUOTA Mobile App initiates the image transfer (Figure 4.10), it first writes to the **MEM_DEV** characteristic of the SUOTA Service to indicate the memory type to be used or to control the SUOTA session (start, end, or reboot).

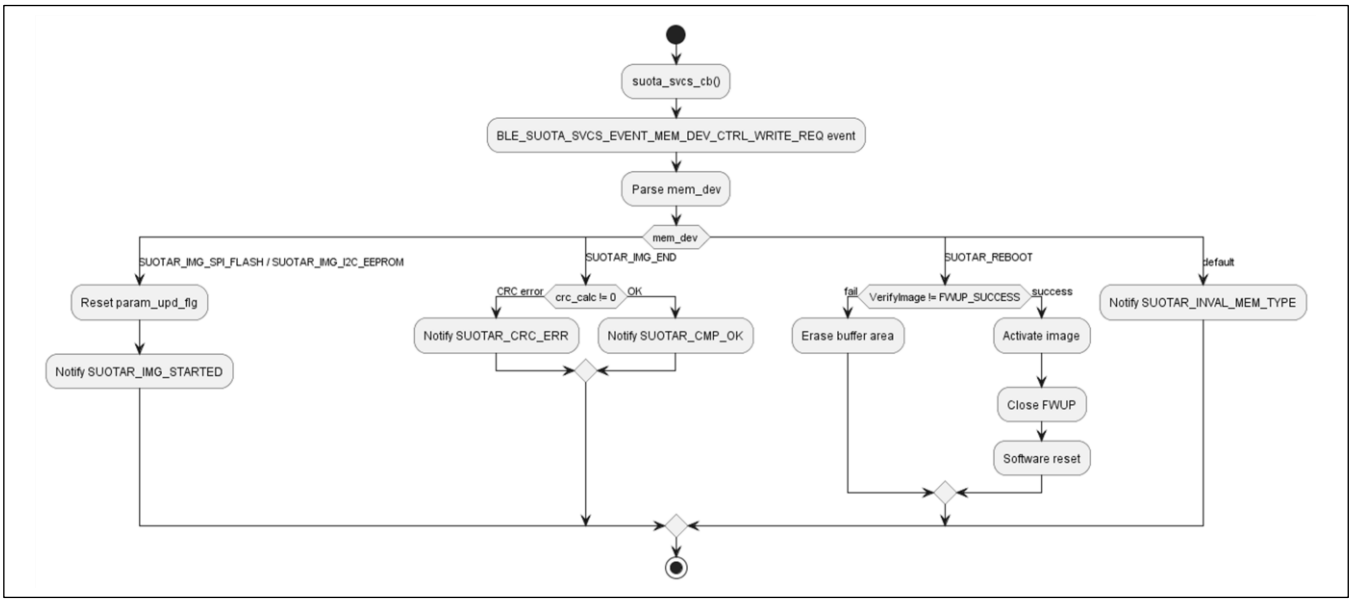


Figure 4.10 BLE_SUOTA_SVCS_EVENT_MEM_DEV_CTRL_WRITE_REQ Event Flow Chart

SUOTAR_IMG_SPI_FLASH or SUOTAR_IMG_I2C_EEPROM:

- Memory type info is **not used** here. This case is only for sending the status SUOTAR_IMG_STARTED back to the phone.

SUOTAR_IMG_END:

- This indicates that the image has been successfully transferred.
- The device then verifies the CRC of the updated image.

SUOTAR_REBOOT:

- This indicates that the phone requests the device to **reboot and apply the new firmware**.

Invalid case (default):

- If the MEM_DEV value is not recognized, the firmware responds with the status SUOTAR_INVAL_MEM_TYPE.

4.4.4 Processing When Writing GPIO Info

After receiving the SUOTAR_IMG_STARTED status from DA14535/DA14531, the peer device sends back the **GPIO map**. However, this GPIO information is not necessary, so it does not need to process it or send any notification in return.

4.4.5 Processing When Writing Patch Length

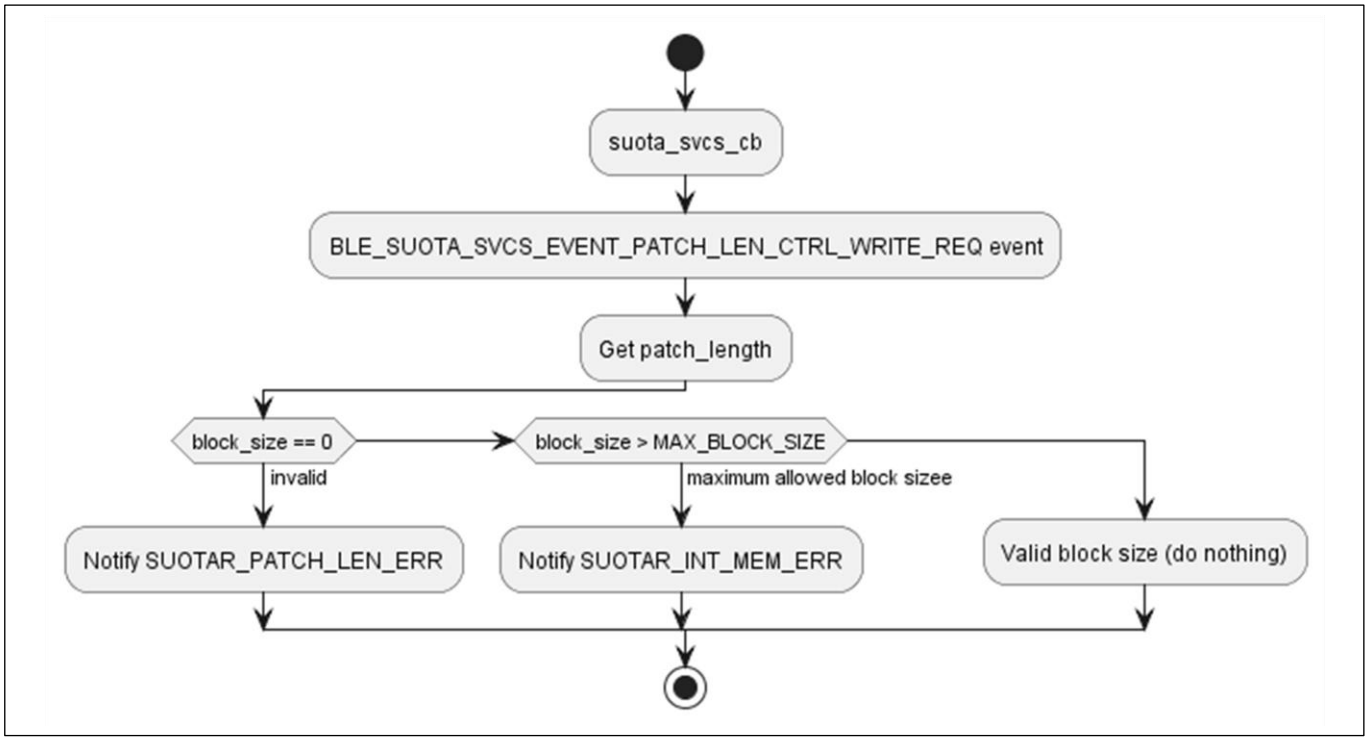


Figure 4.11 BLE_SUOTA_SVCS_EVENT_PATCH_LEN_CTRL_WRITE_REQ Event Flow Chart

Figure 4.11 shows this event occurs when the peer device sends the **PATCH_LEN_CTRL** value, which specifies the size of each data block to be transferred during the SUOTA process.

- The firmware first extracts the block length from the received parameter.
- If the block size is **0**, it is considered invalid, and the DA14535/DA14531 responds with the status SUOTAR_PATCH_LEN_ERR.
- If the block size exceeds the maximum allowed (SUOTA_MAX_BLOCK_SIZE), the device returns the status SUOTAR_INT_MEM_ERR.
- If the block size is valid, no additional response is needed at this step, and the process continues with the next transfer.

4.4.6 Processing When Writing Patch Data

This event is triggered when the peer device writes patch data into the **PATCH_DATA** characteristic as shown in Figure 4.12. The device (DA14535/DA14531) will then process the received patch data.

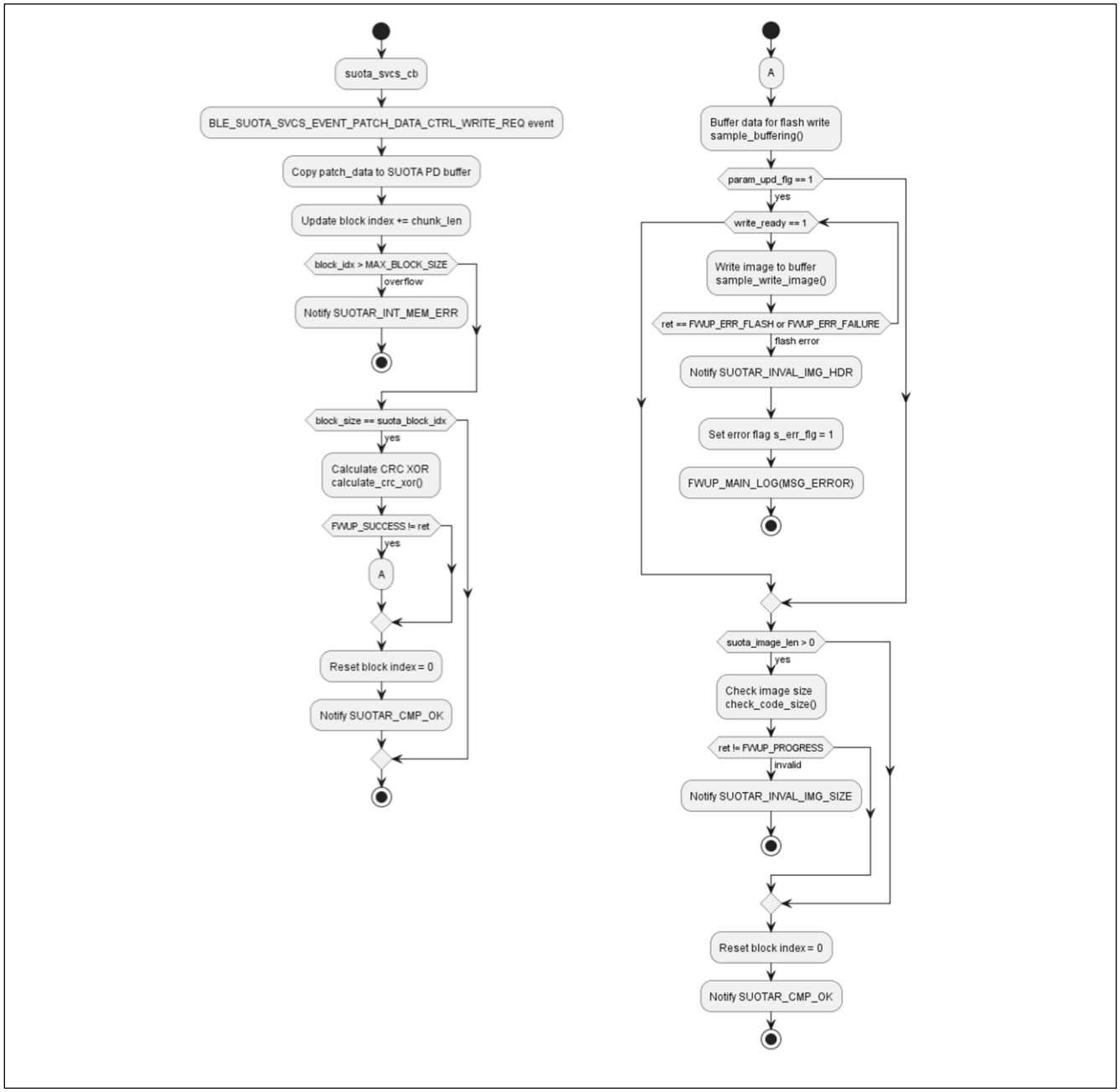


Figure 4.12 BLE_SUOTA_SVCS_EVENT_PATCH_DATA_CTRL_WRITE_REQ Event Flow Chart

- Incoming patch data chunks are stored into a buffer.
- When a full block is received, CRC is calculated, the block is buffered, written to flash memory (if ready), and validated.
- If any error occurs (invalid patch length, memory overflow, flash write failure, or image size mismatch), the corresponding error status is notified. Otherwise, the block is accepted, index reset, and a success status SUOTAR_CMP_OK is sent back to the peer.

4.5 Services

While the mandatory service to run the SUOTA Reference Application is the SUOTA Service, an additional service required to enable Software Update over-the-air is the Device Information Service.

4.5.1 SUOTA Service

This section gives a brief description of the SUOTA Service, responsible for performing software updates over Bluetooth LE. Detailed service characteristic description is given on Table 4.2.

Table 4.2 SUOTA Service Characteristics (1/2)

Characteristic	Since Version	Properties	Size (Byte)	Description
MEM_DEV	v1.3+	READ WRITE	4	Used by the client to send commands to the SUOTA Service, such as: SUOTAR_IMG_SPI_FLASH(0x13) or SUOTAR_IMG_I2C_EEPROM(0x12): Prepare for SUOTA. Memory type info is not used here SUOTAR_REBOOT(0xFD): Verify image and reboot the device. SUOTAR_IMG_END(0xFE): Client sent the whole image. SUOTA Service is allowed to perform CRC calculations.
GPIO MAP		READ WRITE	4	Used to specify GPIO map of external FLASH device. Currently not applicable.
MEM_INFO		READ	4	Stores the total number of bytes received until now. Currently not applicable
PATCH_LEN	v1.3+	READ WRITE	2	Specifies the number of bytes after which they are received. The exact value is set by the GATT client during SUOTA.
PATCH_DATA	v1.3+	READ WRITE WRITE_NO_RESP	Exact size is specified by PATCH_DATA_CHAR_SIZE, plus 2 additional bytes for the current chunk length.	This is the endpoint to which SUOTA image data are sent. The exact size is specified by the “PATCH_DATA CHAR_SIZE” characteristic for patch data, and different values (20 – 244) can be used depending on the throughput requirements

Table 4.3 SUOTA Service Characteristics (2/2)

SUOTA_STATUS	v1.3+	READ NOTIFY	1	This characteristic is used to notify the client of the status of the SUOTA process. Status notifications are sent to indicate error conditions (for example bad command, or CRC) or to allow flow control during SUOTA process.
VERSION	v1.3+	READ	1	Indicates the version of the SUOTA Service. The value is retrieved from the “SUOTA_VERSION” definition.
PATCH_DATA CHAR_SIZE	v1.3+	READ	2	Specifies the size of the “PATCH_DATA ” characteristic.
MTU	v1.3+	READ	2	Stores the current value of the MTU, which is going to be either 23 (default), or a bigger value if MTU exchange took place. This value can be used by the client to retrieve the MTU size (if such API is not available on its side) and write with optimal rate to the “PATCH_DATA ” characteristic.
CCC		READ WRITE	2	Client Characteristic Configuration. Allows the client to enable notifications from the “SUOTA_STATUS” source.

Once the SUOTA Service is discovered on the remote device and the GATT client has enabled notifications by writing the CCC characteristic, the SUOTA procedure can be started by issuing the SUOTAR_IMG_SPI_FLASH or SUOTAR_IMG_I2C_EEPROM command. The write command executes successfully only if:

- No more than one device is currently connected to the SUOTA enabled device
- The application on the SUOTA-enabled device allows the upgrade to take place
- There is enough memory to allocate the internal working buffers

If any of the above restrictions are violated, then command fails and an error notification is sent back to the GATT client. After a successful command execution, the service can receive data either over GATT

The GATT client can use the value of the characteristic MTU to perform ATT write commands to the characteristic PATCH_DATA with optimal size if the GATT Client (mobile phone) has no API to find the optimal packet size. It is also possible for the GATT client to retrieve the size of the PATCH_DATA characteristic by reading the PATCH_DATA_CHAR_SIZE characteristic.

Following this, the GATT client should specify the value of the patch_length variable by writing the PATCH_LEN characteristic. PATCH_LEN specifies the number of bytes that once received, will trigger a notification back to the GATT client. This kind of flow control could be used to avoid flooding the SUOTA enabled device with too much image data. The bigger the value, the better the throughput, since notifications are going to be generated less frequently and therefore the number of missed connection events (where flow has stopped waiting for the notification) is decreased.

For example, if patch_length is set to 500 bytes, notifications will be sent to the GATT client when byte ranges 1-500, 501-1000, 1001 – 1500 etc. are received. Following the Bluetooth low energy specification, the maximum number of bytes that can be written to the PATCH_DATA characteristic with a single ATT write command (244 bytes) is the minimum of MTU – 3 and the size of the PATCH_DATA characteristic.

When the entire image has been transferred, the GATT client should issue the SUOTAR_IMG_END command. The SUOTA Service will perform CRC, then generate a status notification (SUOTA_CMP_OK on success, SUOTA_CRC_ERR on error).

Finally, the GATT client could issue the SUOTAR_REBOOT command to verify new image and force a device reboot. This step is optional, but it is highly recommended.

4.5.2 Device Information Service

Device Information Service (DIS) is a GATT primary service that exposes manufacturer and/or vendor information about the device. These may be manufacturer name, model number, serial number, hardware, firmware, and software revisions and so on.

The SUOTA sample program populates these fields with values configured via QE for BLE. Upon establishing a Bluetooth LE connection, the mobile application automatically performs service discovery to identify available services—including DIS—and reads the characteristic values to gain insight into the remote device's identity and version information.

4.5.3 Pins Used

The following shows list of pins used in this sample program.

Table 4.4 List of Pins and Functions

Pin Name	Input/Output	Function / Connection
PC2/RXD5	input	Connected to DA14535/DA14531MOD UTX pin
PC3/TXD5	output	Connected to DA14535/DA14531MOD URX pin
PD7/RESET	output	Connected to DA14535/DA14531MOD RST pin
PC1/SCK5	output	Connected to DA14535/DA14531MOD UCTS pin
PE2/RXD12	input	Connected to USB-to-UART converter TXD pin
PE1/TXD12	output	Connected to USB-to-UART converter RXD pin

4.6 Sample Program Structure

4.6.1 Peripheral Functions Used

The following shows lists peripheral functions used in this sample program.

Table 4.5 List of Peripheral Functions Used and Functions

Peripheral Functions	Function
SCI5	Asynchronous communication with DA14535/DA14531
SCI12	Asynchronous communication with USB-to-UART converter

4.6.2 Peripheral Function Settings

The Smart Configurator settings used in this sample program are shown below. The items and settings in each table in the Smart Configurator settings are described in the notation on the configuration screen.

Settings not listed are assumed to be default settings.

Table 4.6 Parameters of Smart Configurator (2/3)

Category	Item	Setting/Description
Smart Configurator >> Clock		Default settings are used.
Smart Configurator >> System		Debugging interfaces setting: FINE
Smart Configurator >> Components >> r_bsp		Other than the changes listed below, default settings are used
	User stack setting	1 stack
	Interrupt stack size	0x800
	Heap size	0x2000
	Enable user stdio charput function	Use user charput() function.
	User stdio charput function name	my_sw_charput_function
Smart Configurator >> Components >> r_flash		Other than the changes listed below, default settings are used
	Parameter check	Disable parameter checks
	Enable code flash programming	Includes code to program ROM area
Smart Configurator >> Components >> r_sci_rx		Other than the changes listed below, default settings are used
	Include software support for channel 1	Not
	Include software support for channel 5	Include
	Include software support for channel 12	Include
	ASYNC mode TX queue buffer size for channel 5	1024
	ASYNC mode RX queue buffer size for channel 5	1024
	Transmit end interrupt	Enable
	Resources >> SCI >> SCI5 >>RXD5 Resources >> SCI >> SCI5 >>TXD5	Used
	Resources >> SCI >> SCI12 >> RXD12 Resources >> SCI >> SCI12 >> TXD12	Used
Smart Configurator >> Components >> r_byteq		Other than the changes listed below, default settings are used
	Number of static queue control blocks	16
Smart Configurator >> Components >> r_fwup		Other than the changes listed below, default settings are used
	Select the function mode	use for user program
	Main area start address	0xFFF80000
	Buffer area start address	0xFFFBC000
	Install area	0x3C000
	Code Flash block size	0x800
	Code Flash write unit size	128
	Data Flash block size	128
	Data Flash blocks	32
	Select the algorithm of signature verification	ECDSA + SHA256

Table 4.7 Parameters of Smart Configurator (3/3)

Smart Configurator >> Components >> r_ble_da1453x_rx		Other than the changes listed below, default settings are used
	SCI Channel number for DA1453x Serial Port for GTL command communication	5
	General-purpose port PDR register connected to DA1453x reset port	PORTD
	General-purpose port PODR register connected to DA1453x reset pin	7
	General-purpose port PDR register connected to DA1453x SCK port	PORTC
	General-purpose port PODR register connected to DA1453x SCK pin	1
	Boot from Host	2-Wire UART (for DA14535) 1-Wire UART (for DA14531)
	DA1453X_DEVICE	DA14535/DA14531

4.6.3 Project Optimization

The following shows the project optimization level.

Table 4.8 Project Optimization

Project	Project properties >> C/C++ Build >> Settings >> Compiler >> Optimization		Other than the changes listed below, default settings are used
boot_loader		Optimization Level	Level 2: Performs whole module optimization
fwup_leddemo		Optimization Level	Level 2: Performs whole module optimization
fwup_main		Optimization Level	Level 0: Do not perform optimization

4.6.4 File Structure

The following shows file structure by SUOTA Application program.

Table 4.9 File Structure

Folder name, File name	Outline
src	Folder for program source
├ src	
│ ├ fwup_main.c	Source file for main processing
│ └ fwup_main.h	Header file for main processing
└ smc_gen	Smart Configurator generation
├ general	
├ r_ble_da1453x_rx	
├ r_bsp	
├ r_byteq	
├ r_config	
├ r_flash_rx	
├ r_fwup	
├ r_pincfg	
└ r_sci_rx	
qe_gen	QE-BLE generation
└ ble	

4.6.5 Variables

The following shows the variables that are used in this sample program.

Table 4.10 List of Variables Used in the Sample Program

Variable name	Type	Contents
suota_state	app_suota_state	SUOTAR state
suota_all_pd[SUOTA_MAX_BLOCK_SIZE]	uint8_t	SUOTAR PD array

4.6.6 Constants

The following shows the constants that are used in this sample program

Table 4.11 List of Constants Used in the Sample Program

Constant Name	Setting Value	Contents
FWUP_UPDATE_DATA_FLASH	(1)	Enable updating data flash area
SUOTA_MAX_BLOCK_SIZE	(0x400)	Maximum block size
SUOTA_OVERALL_PD_SIZE	(SUOTA_MAX_BLOCK_SIZE*4)	Overall patch data size
ADDITINAL_CRC_SIZE	(1)	CRC size

4.6.7 SUOTA Parameters & Definitions

This section describes the SUOTA-related parameter structures and definitions used in the sample program.

Table 4.12 Flash Buffer Structure

struct st_flash_buf_t		
Data Fields		
uint8_t	buf	Flash buffer
uint32_t	cnt	Number of data in flash buffer
uint32_t	total	Total number of samples buffered

Table 4.13 SUOTA Status

struct app_suota_state		
Data Fields		
uint8_t	mem_dev	Memory device
uint32_t	block_size	Size of the patch data to be transferred
uint32_t	suota_block_idx	Block index
uint8_t	crc_calc	CRC
uint32_t	suota_image_len	Image length
void (*)(uint16_t conn_hdl, uint8_t suota_status)	status_ind_func	Status indication function
e_fwup_err_t	ret	Firmware update result
uint8_t	write_ready	Data ready to be written to flash
uint8_t	s_err_flg	Error Flag
uint8_t	param_upd_flg	Check param update
st_flash_buf_t	s_flash_buf	Flash buffer

Table 4.14 SUOTA Status Values

Name	Value	Description
SUOTAR_RESERVED	0x00	Value zero must not be used!! Notifications are sent when status changes.
SUOTAR_SRV_STARTED	0x01	Valid memory device has been configured by initiator. No sleep state while in this mode
SUOTAR_CMP_OK	0x02	SUOTA process completed successfully.
SUOTAR_SRV_EXIT	0x03	Forced exit of SUOTAR service
SUOTAR_CRC_ERR	0x04	Overall Patch Data CRC failed
SUOTAR_PATCH_LEN_ERR	0x05	Received patch Length not equal to PATCH_LEN characteristic value
SUOTAR_EXT_MEM_WRITE_ERR	0x06	External Mem Error (Writing to external device failed)
SUOTAR_INT_MEM_ERR	0x07	Internal Mem Error (not enough space for Patch)
SUOTAR_INVAL_MEM_TYPE	0x08	Invalid memory device
SUOTAR_APP_ERROR	0x09	Application error
SUOTAR_IMG_STARTED	0x10	SUOTA started to download image (SUOTA Application program)
SUOTAR_INVAL_IMG_BANK	0x11	Invalid image bank
SUOTAR_INVAL_IMG_HDR	0x12	Invalid image header
SUOTAR_INVAL_IMG_SIZE	0x13	Invalid image size
SUOTAR_INVAL_PRODUCT_HDR	0x14	Invalid product header
SUOTAR_SAME_IMG_ERR	0x15	Same Image Error
SUOTAR_EXT_MEM_READ_ERR	0x16	Failed to read from external memory device

Table 4.15 SUOTAR Physical Memory Device Selection and Commands

Name	Value	Description
SUOTAR_IMG_I2C_EEPROM	0x12	Value written to mem_dev when I2C memory is selected in the SUOTA Mobile App.
SUOTAR_IMG_SPI_FLASH	0x13	Value written to mem_dev when SPI memory is selected in the SUOTA Mobile App.
SUOTAR_MEM_INVAL_DEV	0x14	Default value of mem_dev when no memory device is selected.
SUOTAR_REBOOT	0xFD	Value written to mem_dev when image transfer is completed.
SUOTAR_IMG_END	0xFE	Value written to mem_dev when "Reboot" is pressed on the SUOTA Mobile App.

4.6.8 Functions

The following shows the user-defined functions added to this sample program.

Table 4.16 List of Additional Functions Used in the Sample Program

Function name	Outline
fwup_main_open	Initializes the firmware update module and related peripherals.
sample_sci_open	Initializes the configuration data structure for asynchronous (UART) operation to enable log printing
my_sw_charput_function	Char data output API
app_suota_init	Initializes the SUOTA (Software Update Over The Air) state and buffers.
fwup_restart	Closes and reopens the Firmware Update Module.
app_read_cfm	Sends a read confirmation with attribute value to the Bluetooth LE stack.
sample_buffering	Buffers SUOTA patch data into internal flash buffer.
sample_write_image	Writes buffered SUOTA image data to flash memory.
sample_buf_init	Initializes (clears) the internal flash buffer used for SUOTA.
suotar_send_status_update_req	Sends a SUOTA status update notification to the SUOTA Mobile App.
calculate_crc_xor	Calculates the XOR-based CRC for a given image.
check_code_size	Checks whether the size of the SUOTA image is within allowed area size.

4.6.9 Function Specifications

This section describes the specifications of user-defined functions that were added to the sample program.

Table 4.17 Firmware Update Module Initialization Function

static void fwup_main_open(void)
Description: Initializes the Firmware Update Module and related peripherals.
Arguments: None
Return values: None

Table 4.18 UART Initialization for Log Output

static void sample_sci_open(void)
Description: Initializes the configuration data structure for asynchronous (UART) operation to enable log printing
Arguments: None
Return values: None

Table 4.19 Character Output Utility Function

void my_sw_charput_function(uint8_t data)		
Description: char data output API		
Arguments:		
Type	**Parameter**	**Description**
uint8_t	data	Send data with SCI
Return values: None		

Table 4.20 SUOTA Initialization Function

static void app_suota_init(void)
Description: Initializes the SUOTA (Software Update Over The Air) state and buffers.
Arguments: None
Return values: None

Table 4.21 Restart Firmware Update Module

static void fwup_restart(void)
Description: Closes and reopens the firmware update module
Arguments: None
Return values: None

Table 4.22 Send BLE Read Confirmation

static void app_read_cfm(uint16_t conn_hdl, uint16_t attr_hdl)		
Description: Sends a read confirmation with attribute value to the BLE stack.		
Arguments:		
Type	**Parameter**	**Description**
uint16_t	conn_hdl	Handle of connection on which request should be made
uint16_t	attr_hdl	Handle of attribute to be read
Return values: None		

Table 4.23 Buffer SUOTA Data into Internal Flash Buffer

static void sample_buffering(uint8_t *rx_data, uint32_t length)		
Description: Buffers SUOTA patch data into internal flash buffer.		
Arguments:		
Type	**Parameter**	**Description**
uint8_t *	rx_data	Pointer to received data
uint32_t	length	Length of received data
Return values: None		

Table 4.24 Write Buffered Image to Flash

static e_fwup_err_t sample_write_image(e_fwup_area_t area)		
Description: Writes buffered SUOTA image data to flash memory.		
Arguments:		
Type	**Parameter**	**Description**
e_fwup_area_t	area	Flash area to write to
Return values:		
FWUP_SUCCESS	Updated image program	
FWUP_PROGRESS	Need more data	
FWUP_ERR_FLASH	Flash write error	

Table 4.25 Initialize SUOTA Flash Buffer

static void sample_buf_init(void)
Description: Initializes (clears) the internal flash buffer used for SUOTA.
Arguments: None
Return values: None

Table 4.26 Send SUOTA Status Update Notification

static void suotar_send_status_update_req(uint16_t conn_hdl, uint8_t suota_status)		
Description: Sends a SUOTA status update notification to the SUOTA Mobile App.		
Arguments:		
Type	**Parameter**	**Description**
uint16_t	conn_hdl	Handle of connection on which request should be made
uint8_t	suota_status	Status to send
Return values: None		

Table 4.27 Calculate XOR-Based CRC

static void calculate_crc_xor(const uint8_t *p_data, uint32_t length, uint8_t *p_crc)		
Description: Calculates the XOR-based CRC for a given image.		
Arguments:		
Type	**Parameter**	**Description**
const uint8_t*	p_data	Pointer to the data buffer
uint32_t	length	Number of bytes to process
uint8_t *	p_crc	Pointer to the CRC
Return values : None		

Table 4.28 Validate SUOTA Image Size

static e_fwup_err_t check_code_size(uint32_t suota_image_length)		
Description: Checks whether the size of the SUOTA image is within allowed area size.		
Arguments:		
Type	**Parameter**	**Description**
uint32_t	suota_image_length	Image length
Return values:		
FWUP_PROGRESS	Code size is within allowed area size	
FWUP_ERR_FAILURE	Code size is out of allowed area size	

4.6.10 ROM/RAM Usage

ROM/RAM usage for fwup_main of DA14535 sample program is shown below.

Table 4.29 ROM Usage

Size (KByte)	Description
10.9	QE-BLE
30.95	r_ble_da1453x_rx module
12.6	r_fwup, r_flash_rx modules
2.5	Demo program
19.2	Other
Total 77KByte	MAX 512Kbyte (15% Used)

Table 4.30 RAM Usage

Size (KByte)	Description
13.8	DA1453x work area
8	Heap area
0.15	Firmware update work area
15.29	Other
Total 29KByte	MAX 128Kbyte (22% Used)

ROM/RAM usage for fwup_main of DA14531 sample program is shown below.

Table 4.31 ROM Usage

Size (KByte)	Description
11.09	QE-BLE
37.25	r_ble_da1453x_rx module
12.6	r_fwup, r_flash_rx modules
2.5	Demo program
19.2	Other
Total 82.66KByte	MAX 512Kbyte (17% Used)

Table 4.32 RAM Usage

Size (KByte)	Description
14.01	DA1453x work area
8	Heap area
0.15	Firmware update work area
15.29	Other
Total 29KByte	MAX 128Kbyte (22% Used)

5. Start Demonstration

5.1 Project Description

This demo project supports two modules DA14531/DA14535.

- **da14531:** DA14531 project

  This program supports DA14531 module.

- **da14535:** DA14535 project

  This program supports DA14531 module.

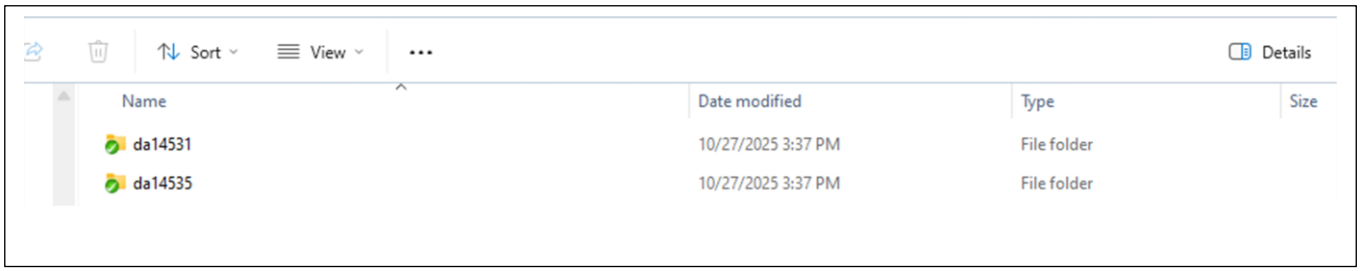


Figure 5.1 DA14531/DA14535 Project

The demo project is structured into multiple components to support the firmware update process. Each component plays a specific role in demonstrating the OTA update mechanism on the FPB-RX261 platform.

- **boot_loader:** Bootloader

  This program runs first after a reset. It verifies that the user program has not been tampered with and then, if verification is successful, launches the user program.

- **fwup_main:** SUOTA Application program

  SUOTA Application program (initial firmware) that transfers updated firmware and performs signature verification.

- **fwup_leddemo:** Application program (for update)

  This is an application program (for updating) that blinks an LED and updates firmware version.

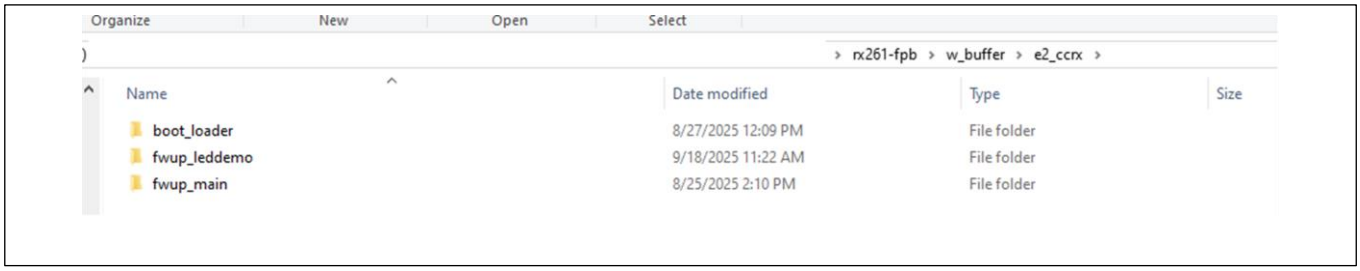


Figure 5.2 The Demo Project Structure Divided into Three Main Components

5.2 Importing the SUOTA Sample Project

(1) Download sample project on the Renesas website
SUOTA sample program: r01anxxxxeuxxx-rx261-suota.zip

(2) Extract the demo project

(3) Start e2 studio

(4) From the **File** menu , select **Import**

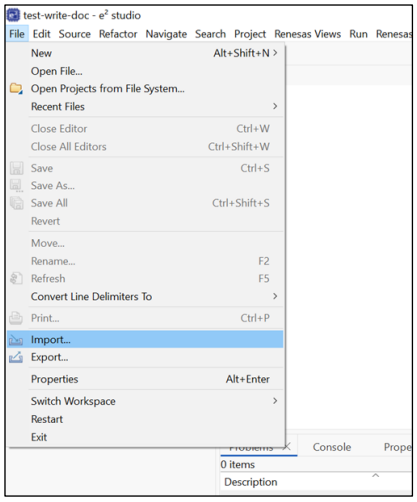


Figure 5.3 Importing Project

(5) Select **Existing Project into Workspace**

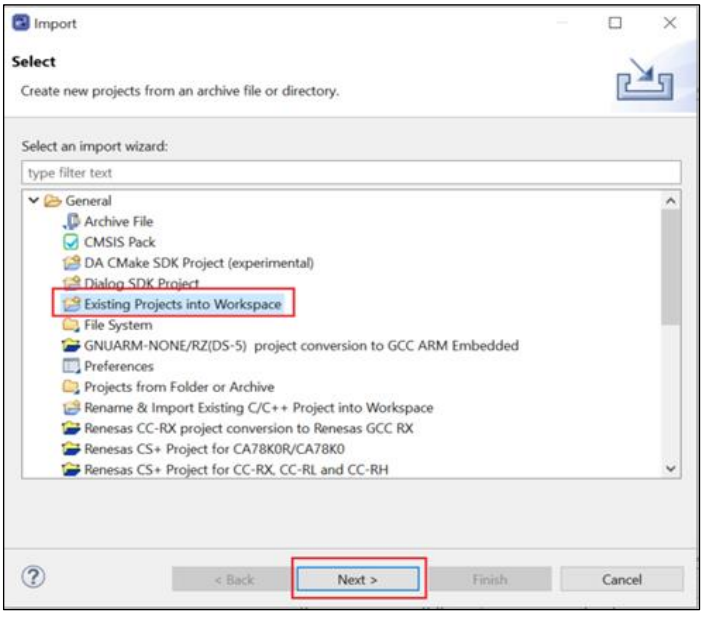


Figure 5.4 Select Existing Projects Into Workspace

(6) In **Select** root directory, select the folder extracted, select the check boxes for the following projects, and then click **Finish**

5.3 Creating Firmware Initialization

5.3.1 Building Project

(1) **Update the public key**

The public key must be pasted into both the bootloader and fwup main projects to match the private key located in the Renesas Image Generator folder. This ensures that the initial firmware can be generated successfully. Follow the instructions below to complete this setup.

1. Copy the contents of the secp256r1.publickey file you created in Section 3.3.3
2. Paste the public key into CODE_SIGNER_PUBLIC_KEY in \bootloader\src\key\code_signer_public_key.h

   \fwup main\src\key\code_signer_public_key.h

   \fwup leddemo\src\key\code_signer_public_key.h

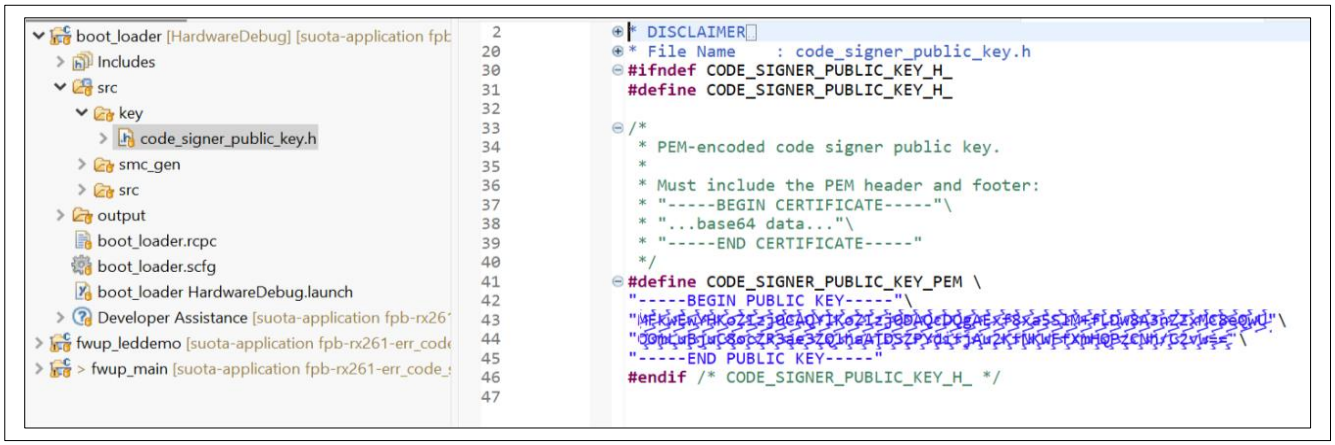


Figure 5.5 Replace a Public Key to Bootloader Project

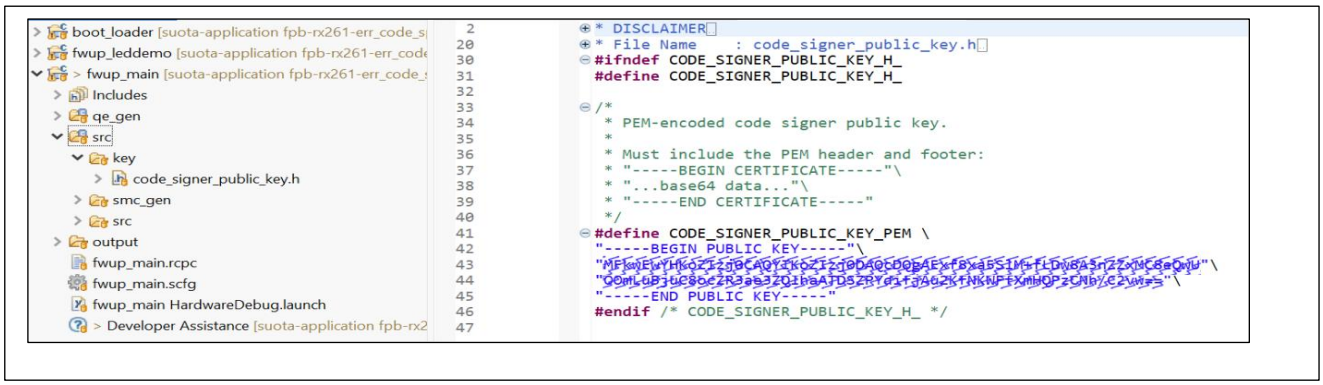


Figure 5.6 Replace a Public Key to fwup_main Project

Figure 5.7 Replace a Public Key to fwup_leddemo Project

(2) **Building the project**

Figure 5.8 Building the Bootloader Project Successfully

Figure 5.9 Building the fwup_main Project Successfully

Figure 5.10 Building the fwup_leddemo Project Successfully

5.3.2 Creating the Initial and Updating Firmware

This section is used to create the initial firmware(**initial_firm.mot**), which is executed on the MCU before the Over-The-Air process begins. Follow the setup steps below to generate the initial firmware correctly:

(1) **Place the following files in the Renesas Image Generator folder:**

- The results of the building process in Section 5.3.1: **fwup_main.mot**
- The results of building the bootloader in Section 5.3.1: **boot_loader.mot**
- The results of building the fwup_leddemo in Section 5.3.1: **fwup_leddemo.mot**
- The private key created in Section 3.3.3: **secp256r1.privatekey**

(2) **Use Renesas Image Generator to generate the initial firmware**

Renesas Image Generator has the bootloader file name(.mot) generated by build, application program(.mot), parameter file name(.csv), output file name (no extension), image verification method in Firmware Update Module. Specify(ecdsa/sha256) as a command line option to generate an initial image file(.mot)

Open a command prompt, navigate to the Renesas Image Generator folder, and execute the following command to generate the file **initial_firm.mot**.

```
> python image-gen.py -iup fwup_main.mot -ip RX261_Linear_Half_ImageGenerator_PRM.csv -o initial_firm -ibp boot_loader.mot -vt ecdsa -key secp256r1.privatekey
```

Parameter explanation:

- **-iup**: Input user program (.mot file for application firmware)
- **-ip:** Input parameter file (.csv with image generation settings)
- **-o:** Output file prefix (e.g., userprog.mot)
- **-ibp:** Input bootloader program (.mot file)
- **-vt:** Verification type (e.g., ecdsa for digital signature)

(3) **Use Renesas Image Generator to generate the updated firmware**

The Renesas Image Generator uses the update application program(.mot) generated by the build, parameter file name(.csv), output file name (no extension), image verification method(ecdsa/sha256) for the Firmware Update Module. Set the command line options to generate an update image file(.rsu)

Open a command prompt, navigate to the Renesas Image Generator folder, and execute the following command to generate the file **fwup_leddemo.mot**.

```
> python image-gen.py -iup fwup_leddemo.mot -ip RX261_Linear_Half_ImageGenerator_PRM.csv -o fwup_leddemo -vt ecdsa -key secp256r1.privatekey
```

5.4 New Connection Configuration for the MCU Board

(1) **Start the Renesas Flash Programmer**

- Microcontroller: RX200
- Project Name: Any (Example: FPB-RX261)
- Project Folder: Any
- Tool: E2 emulator Lite
- Interface: FINE
- Tool Details…: Choose -> Reset Settings -> Reset Pin as Hi-Z
- Click “Connect”

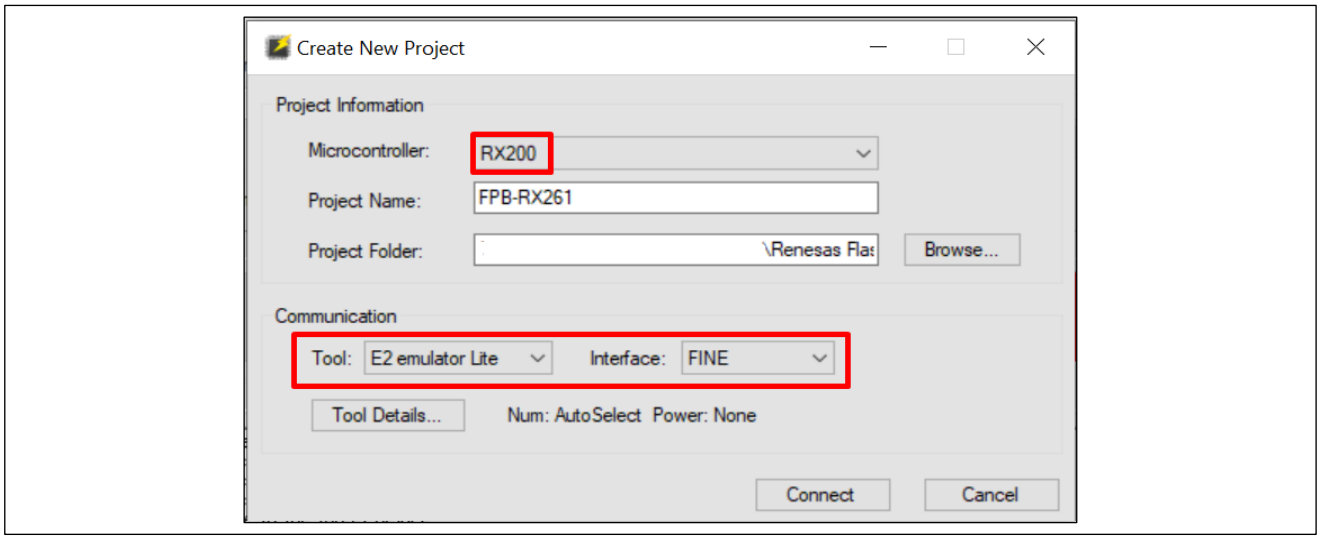


Figure 5.11 New Connection Configuration for the MCU Board

The connection is successful if the following window appears.

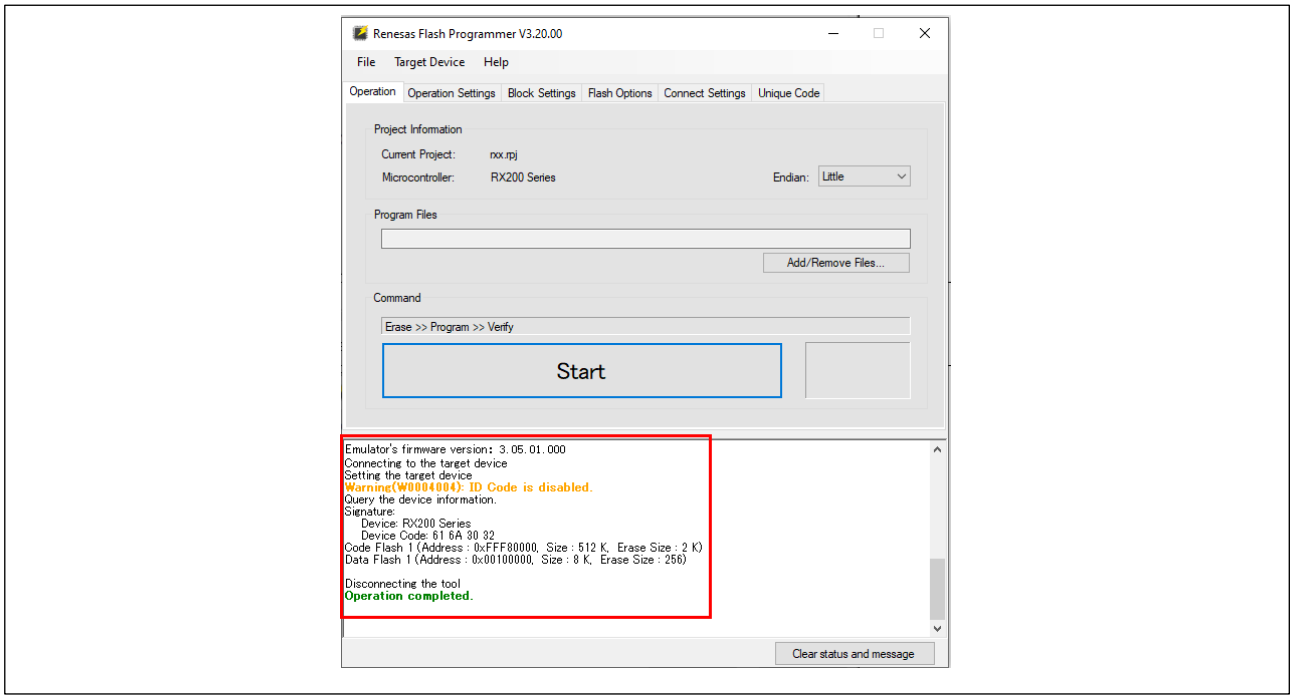


Figure 5.12 Connection Completed

5.5 Programming a MOT File to the MCU Board

(1) In the Program File field, enter the path to the MOT file to be programmed, and then click “Start”.

- Program File: MOT file to be programmed (Example: initial_image.mot, fwup_main.mot)
- Click “Start”

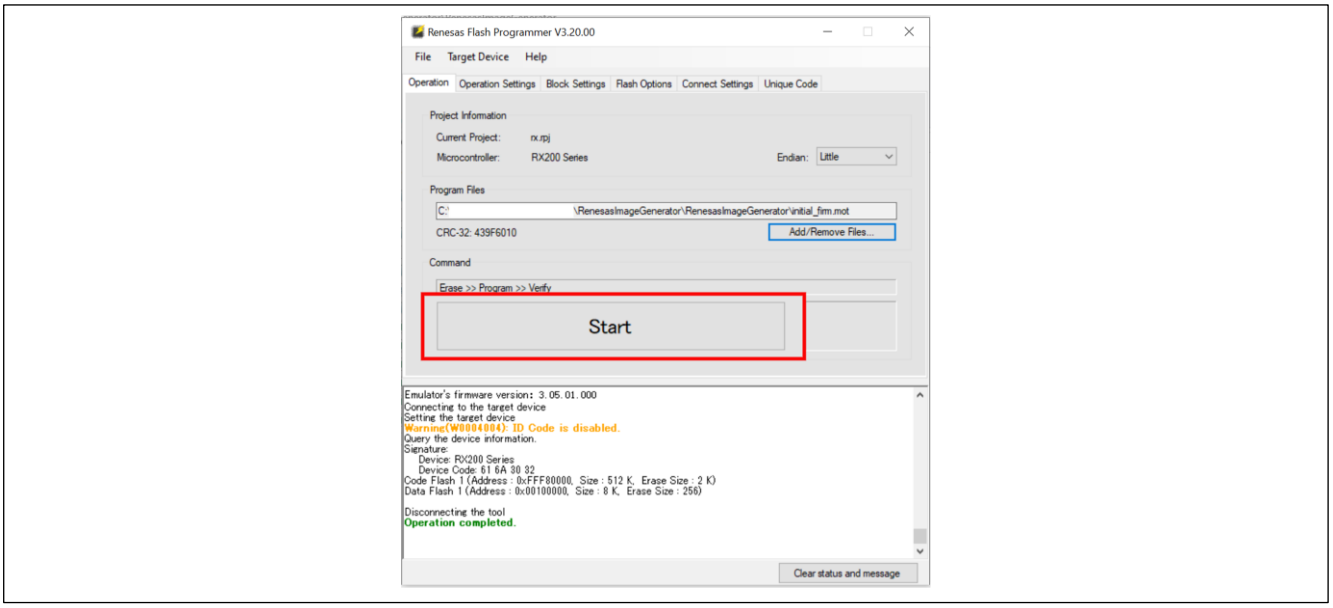


Figure 5.13 Programming a MOT File to the MCU Board

(2) Make sure that programming is successful.

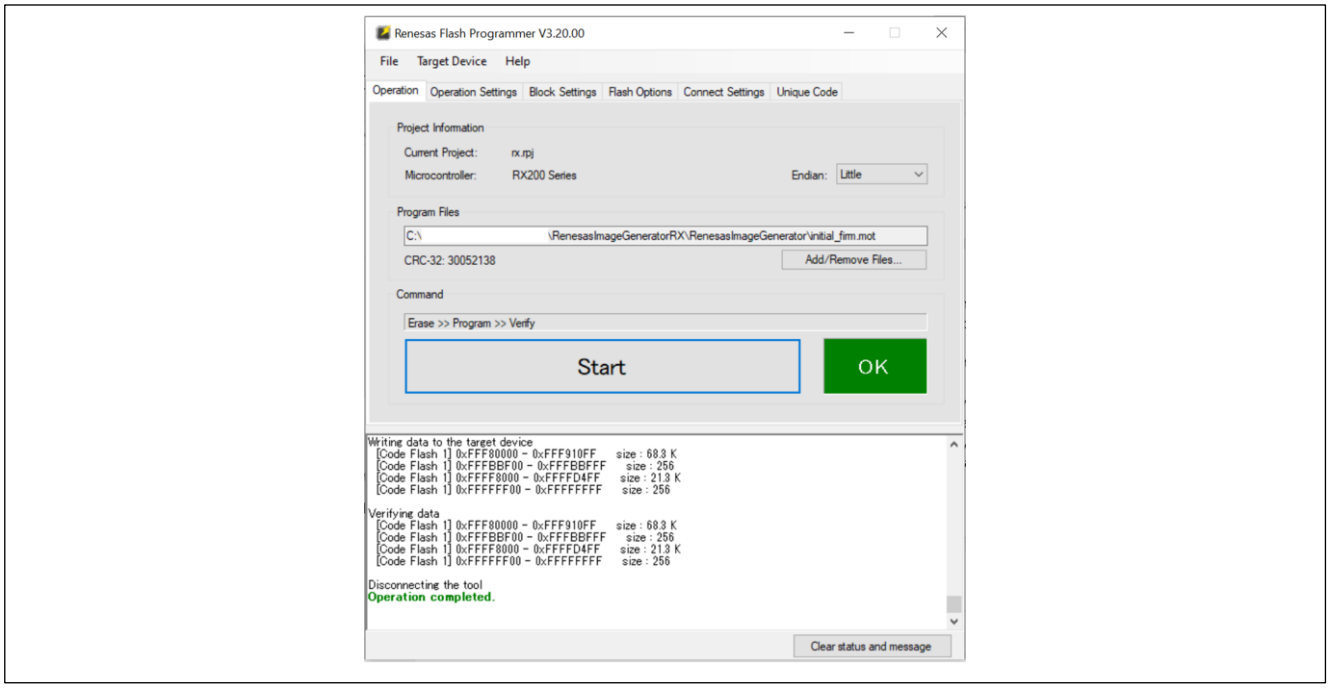


Figure 5.14 Successful Programming into MCU

Figure 5.15 After Flashing the Initial Image

5.6 Requesting to Update the Firmware

In the current SUOTA implementation, the block size for transferring firmware data over Bluetooth LE is limited to 1024 bytes. If the block size exceeds 1024, the SUOTA Mobile App will report an error.

To ensure correct operation:

- Always configure the Mobile App to use a block size (Example: 256 bytes)

Failure to comply with this limitation may lead to firmware update instability or failure.

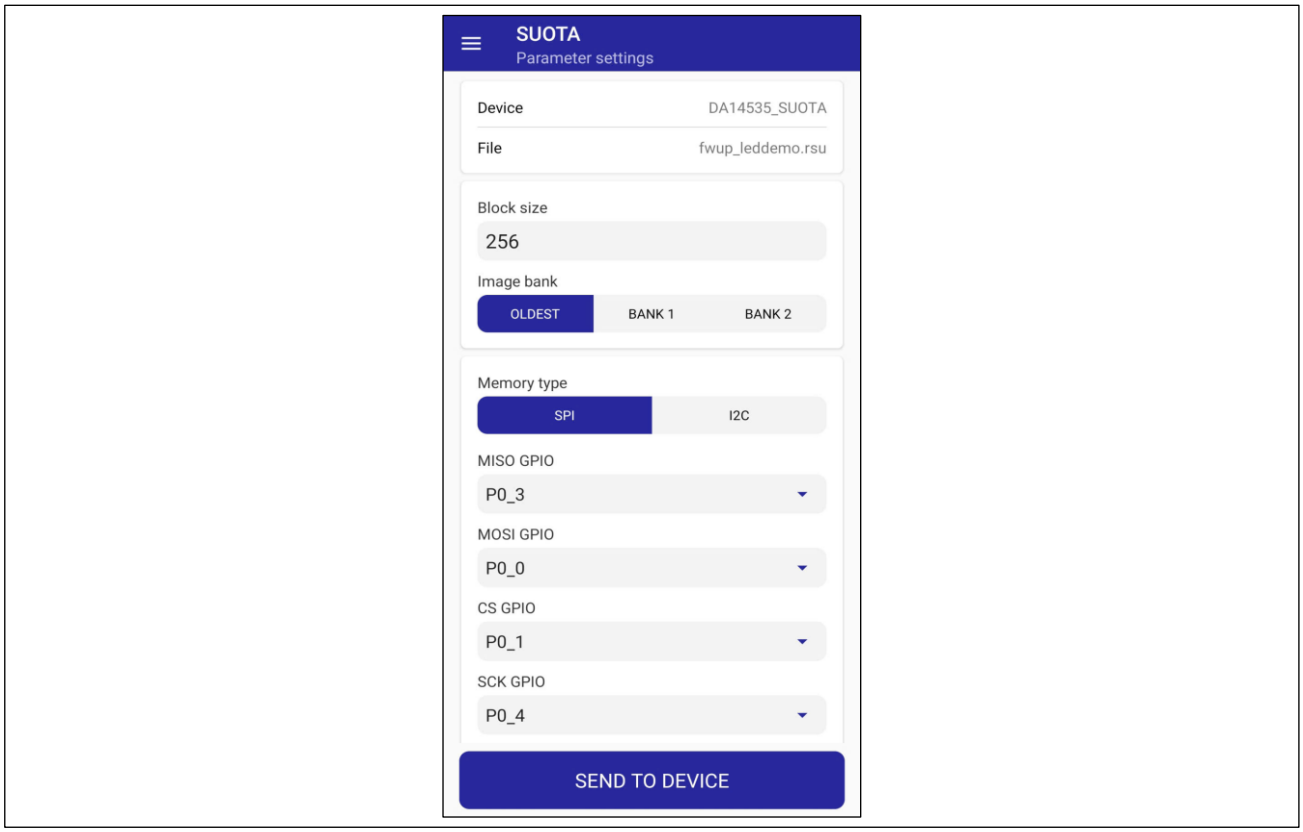


Figure 5.16 Block Size

5.6.1 Updating Firmware by SUOTA Service

(1) Waiting for the Bluetooth LE connection to be successfully established.

Figure 5.17 Successfully Initialize Bluetooth LE in SUOTA Sample Program

(2) Launch the Renesas SUOTA Mobile App on the Android phone and select the “DA14535_SUOTA” device name to connect Bluetooth LE.

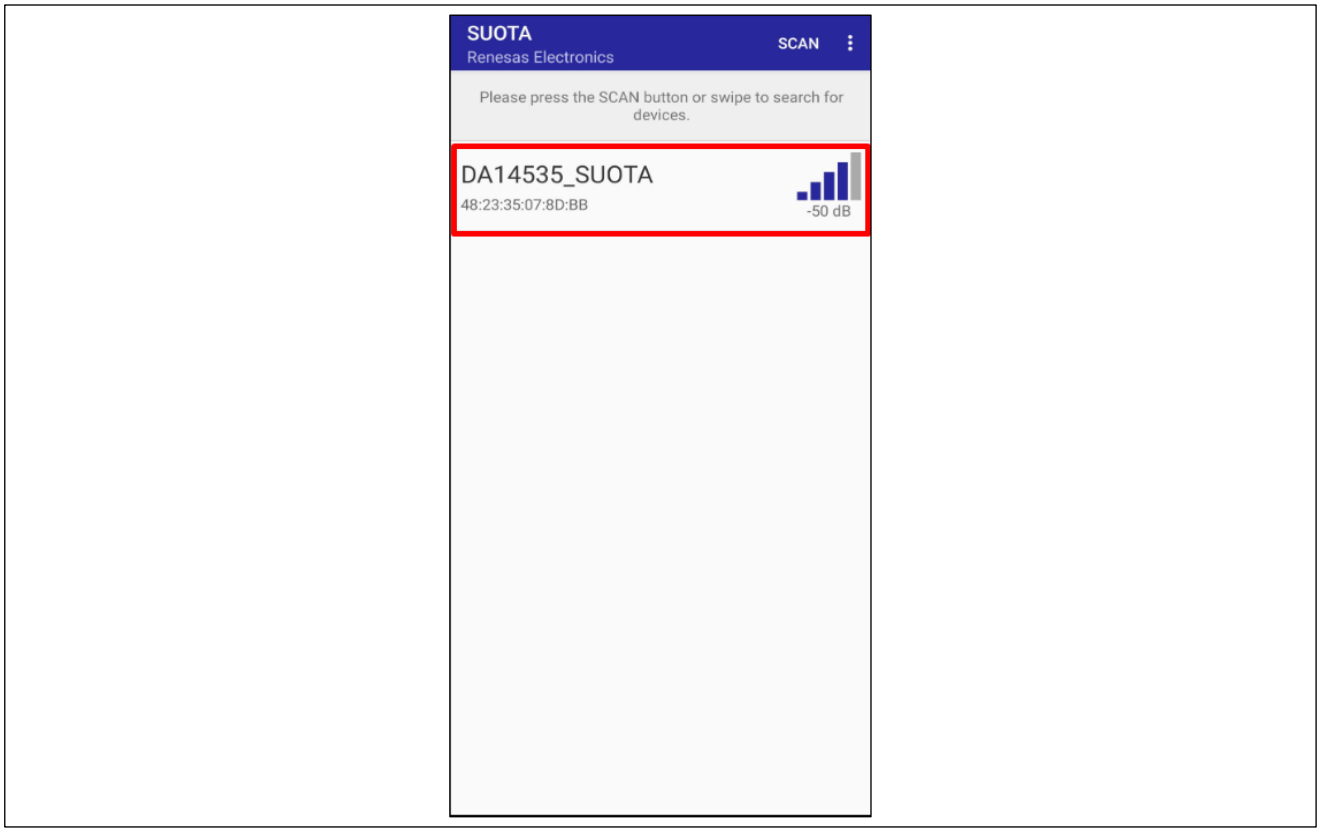


Figure 5.18 SUOTA Mobile App Interface

(3) After a connection successfully Bluetooth LE, click “Update Device”

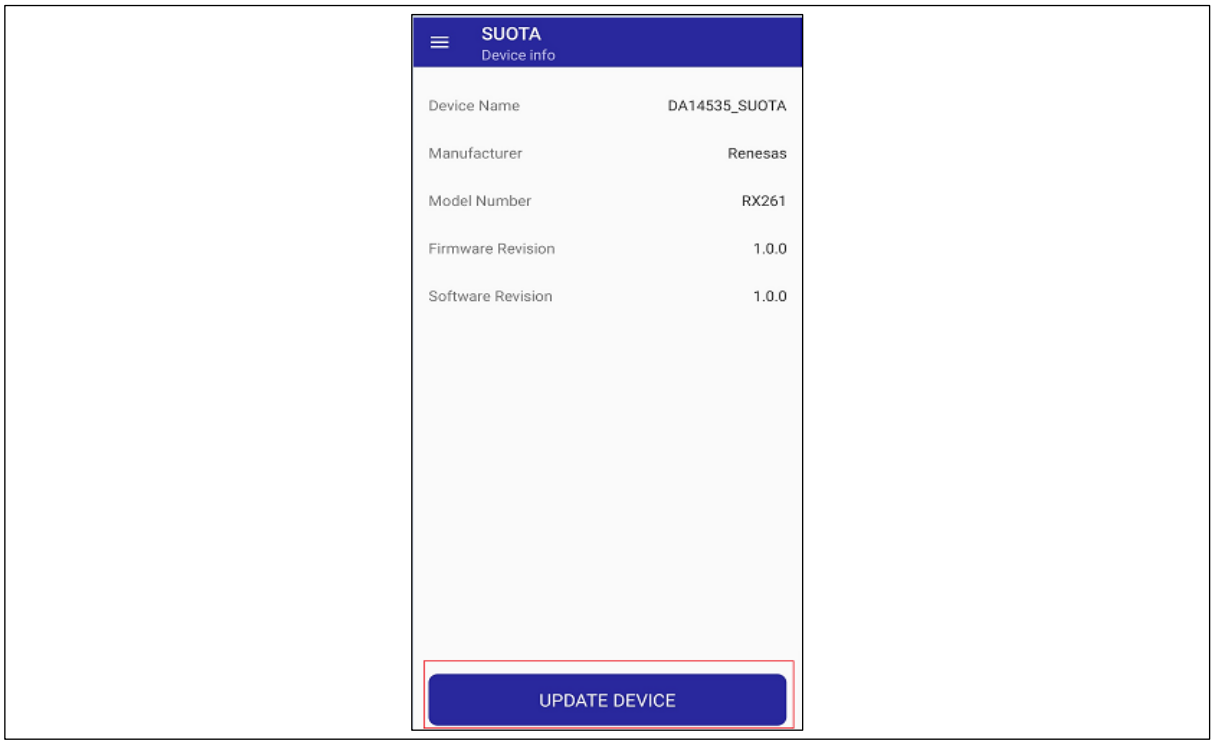


Figure 5.19 Device Information

(4) Configure file streaming

- Select the update firmware that is generated in Section 5.3.2
- Image bank : OLDEST
- Memory type: SPI
- MISO GPIO: P0_3
- MOSI GPIO: P0_0
- CS GPIO: P0_1
- SCK GPIO: P0_4
- Select “SEND TO DEVICE” option to begin firmware transfer

Note: These settings are arbitrary and not used in this sample. You may configure them with any values, as they do not affect the firmware writing process.

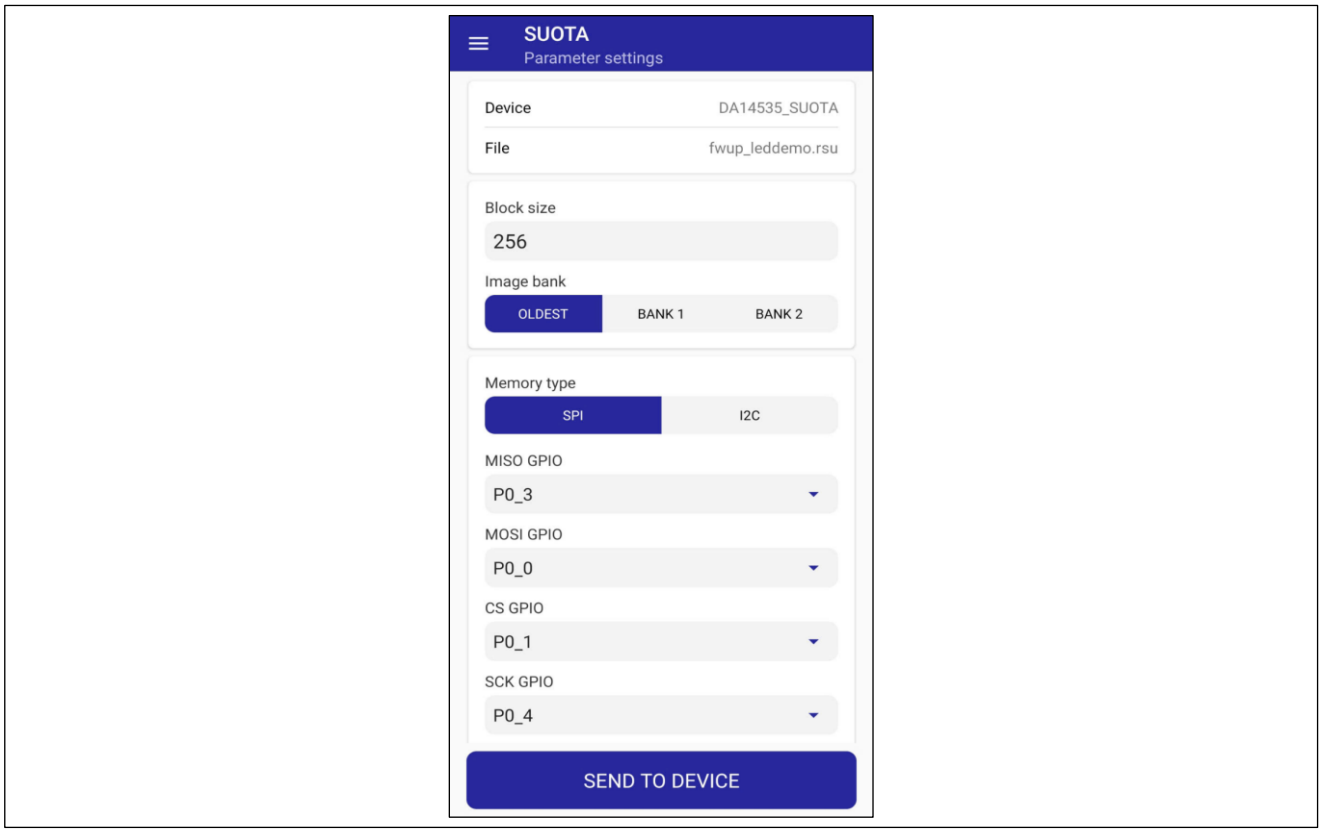


Figure 5.20 Configure SUOTA Mobile App

(5) Observing the SUOTA firmware update process

The process starts from 0% to 100%.

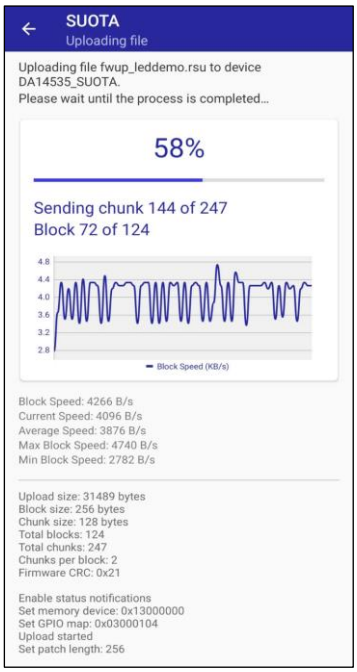


Figure 5.21 Transfer in-Progress

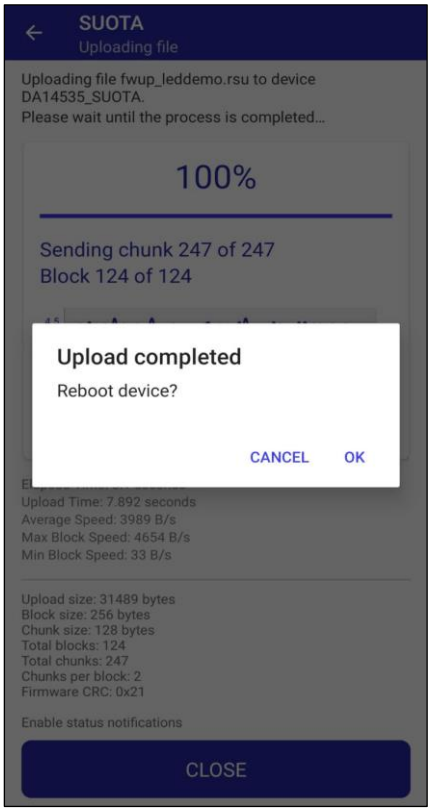


Figure 5.22 Upload Completed

(6) Press “OK” button to update new firmware

The firmware update process starts with progress messages such as download progress, “W 0xFFFBC000, 512” (indicating that address of flash memory and write size that was written new firmware) displayed in the terminal. Upon successful completion, the firmware is updated from **1.0.0** to **1.1.0**. If an error occurs during the update, the MCU automatically resets, and the previous stable firmware version is retained.

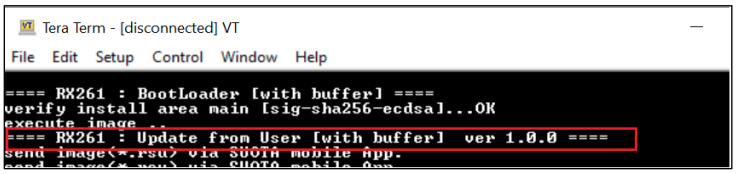


Figure 5.23 Version Firmware Before Update

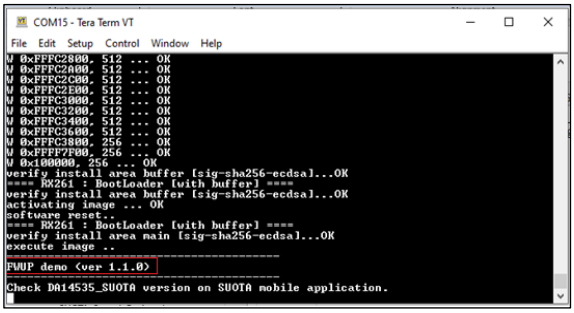


Figure 5.24 Check New Firmware Version Over Log UART

Figure 5.25 Check New Firmware Version on SUOTA Mobile App

(7) The updated firmware (fwup_leddemo) can continue updating the new firmware. If you want to update firmware version, follow these steps.

The version of firmware can be updated in “R_BLE Custom Profile RA, RE, RX (QE)”
Step 1: Move to “Renesas Views”, choose “Renesas QE” and select “R_BLE Custom Profile RA, RE, RX (QE)”

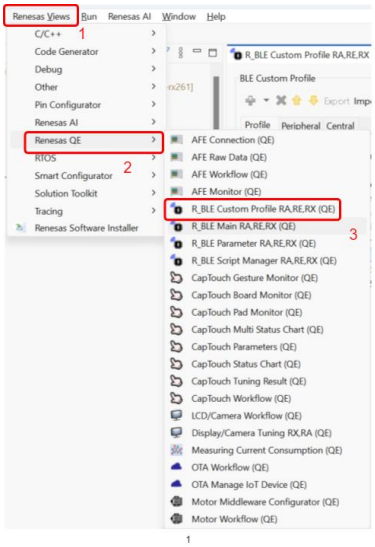


Figure 5.26 R_BLE Custom Profile

Step 2: Select “fwup_leddemo” and “DA1453x”, choose “Device Information Service” and then the list of characteristics will show.

The version of device can be configured in the option “Value” of the revision characteristics.

Step 3: Click “Generate Code” and build “fwup_leddemo” project again to generate mot file.

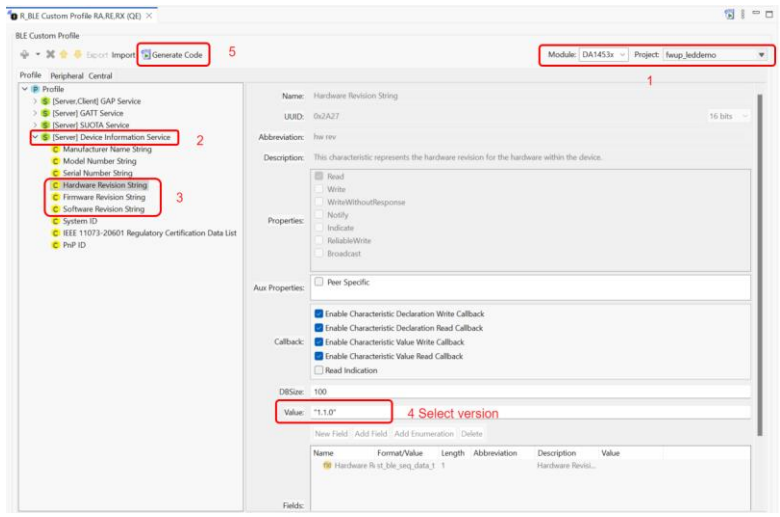


Figure 5.27 Select Firmware Version

5.7 How to debug the demo project

If you wish to debug this project (bootloader + application program) in the e2 studio environment, the following procedure can be used.

This demo project is set to be powered by the emulator in the debugger (E2 Lite). If you want to connect with other debuggers or supply power from the target board, change the debugger settings.

(1) Build the bootloader and application program without optimization.

Build the bootloader (boot_loader) and application program (fwup_main) with the e2 studio optimization level set to no optimization.

(2) Generate the initial image.

The Renesas Image Generator generates an initial image file (.mot) consisting of a bootloader (boot_loader) and an application program (fwup_main).

(3) Debug settings for the application program (fwup_main).

Follow the steps below to configure debugging settings for the application program (fwup_main).

(a) Open Run->Debug Configuration and select fwup_main_HardwareDebug.

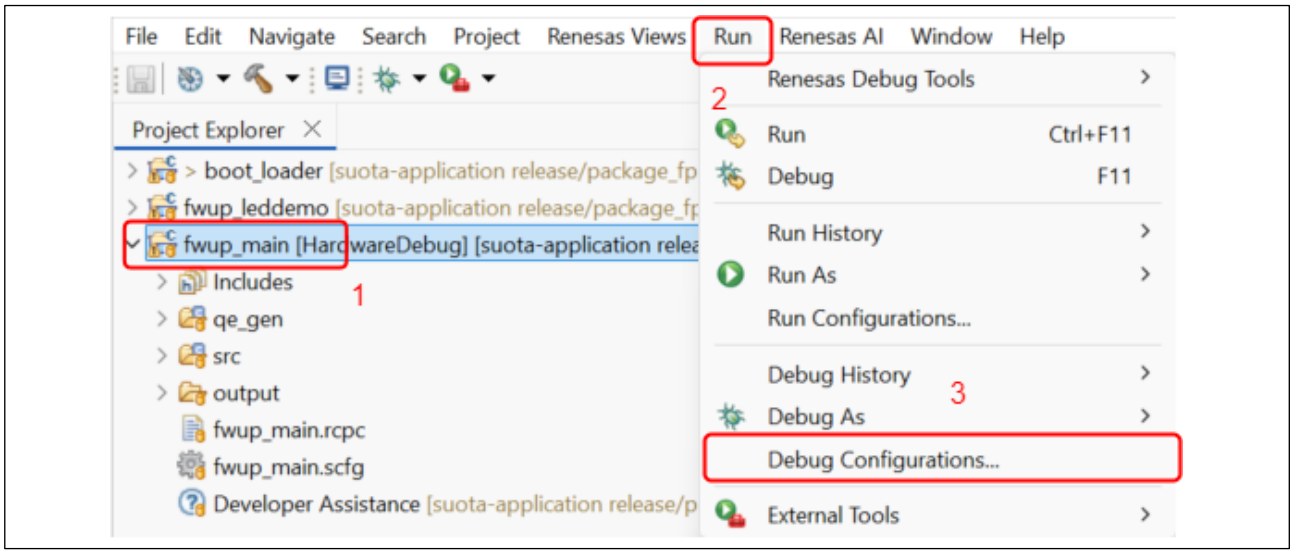


Figure 5.28 Debug Configuration

(b) Select fwup_main_HardwareDebug and click Startup.

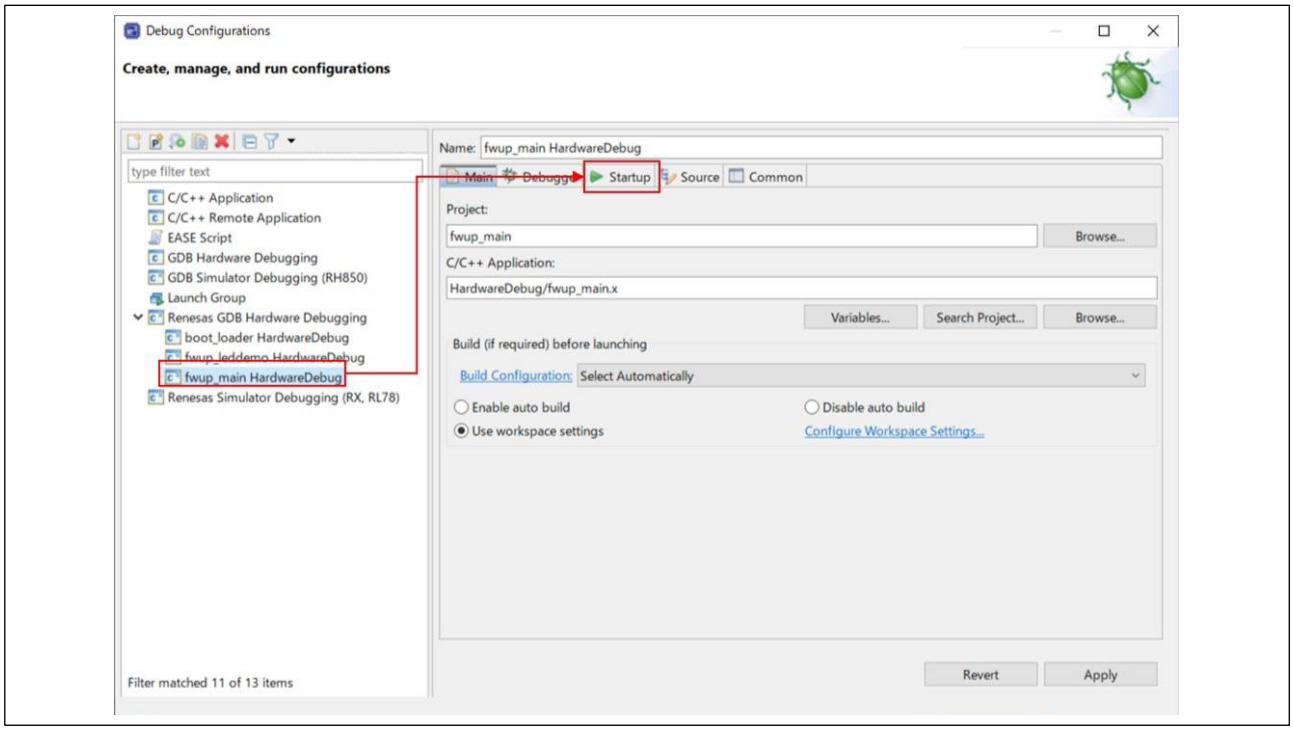


Figure 5.29 Debug Configuration

(c) Change the load type of the program binary [fwup_main.x] from "Image and Symbol" to "Symbol Only". In the GCC environment, this will be fwup_main.elf.

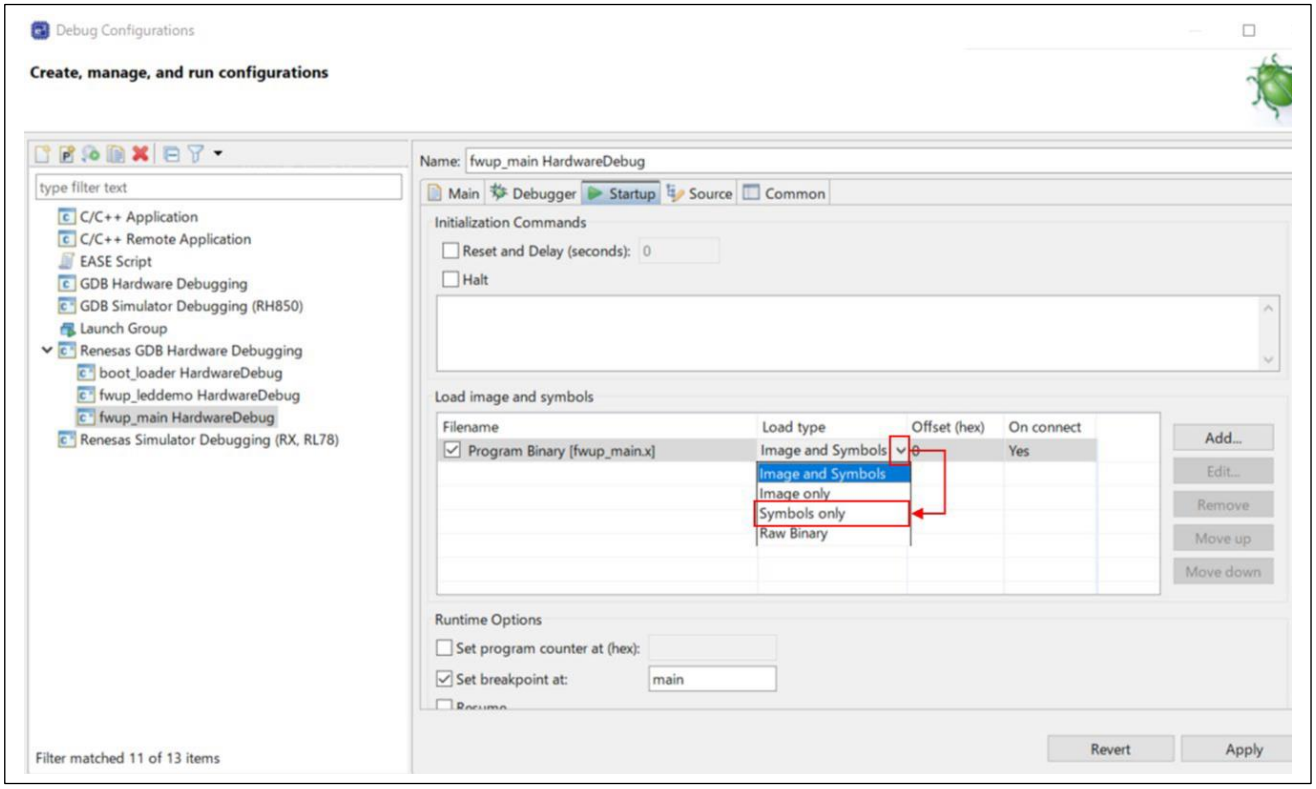


Figure 5.30 Change Load Type

(4) Add the bootloader (boot_loader) symbol.

Follow the procedure below to add the boot loader (boot_loader) symbol built in step (1).

(a) Click "Add".

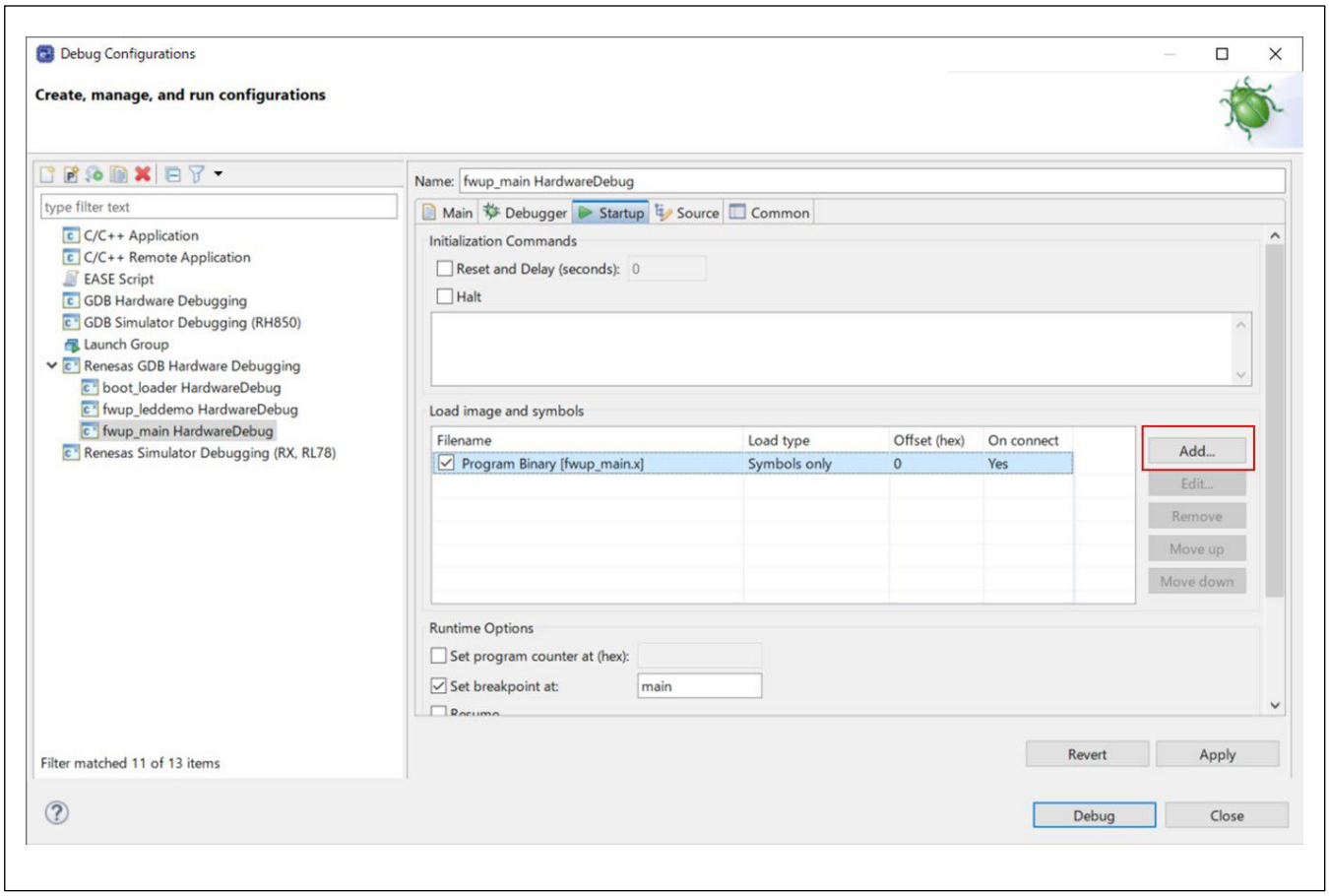


Figure 5.31 Add Symbols

(b) Click Workspace.

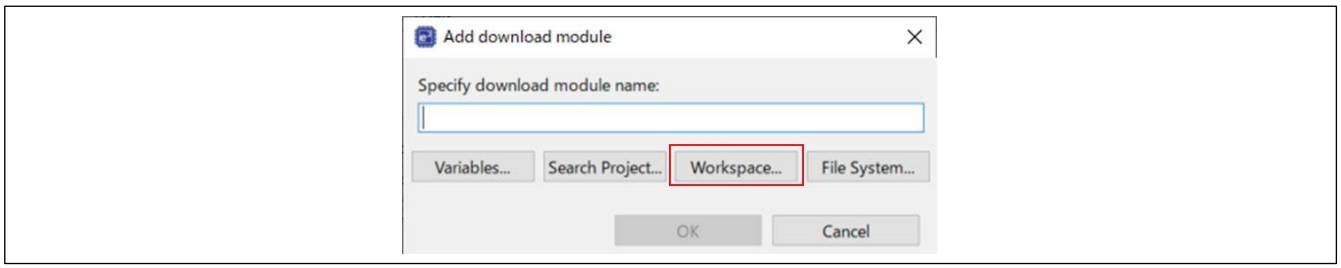


Figure 5.32 Click "Workspace"

(c) Select the bootloader (boot_loader.x) and click "OK." In the GCC environment, the symbol is boot_loader.elf.

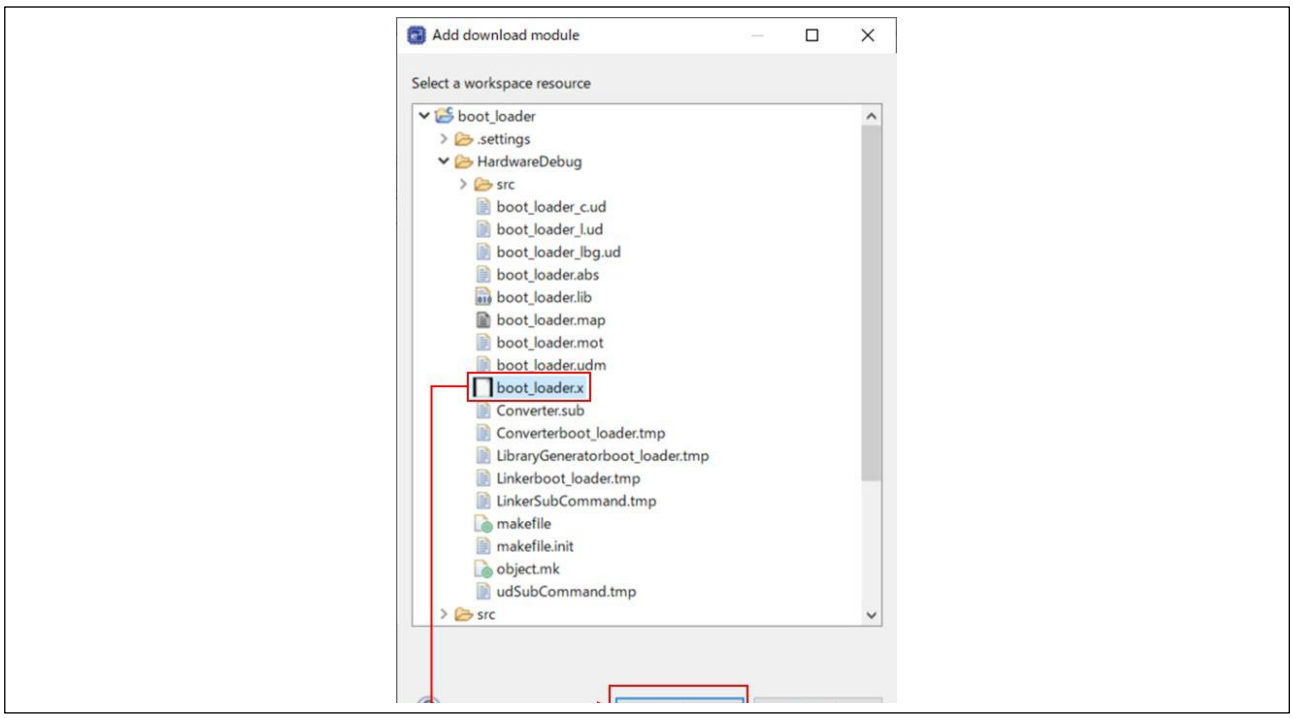


Figure 5.33 Add boot_loader.x

(d) Confirm that the download module name is set to bootloader (boot_loader.x) and click "OK.

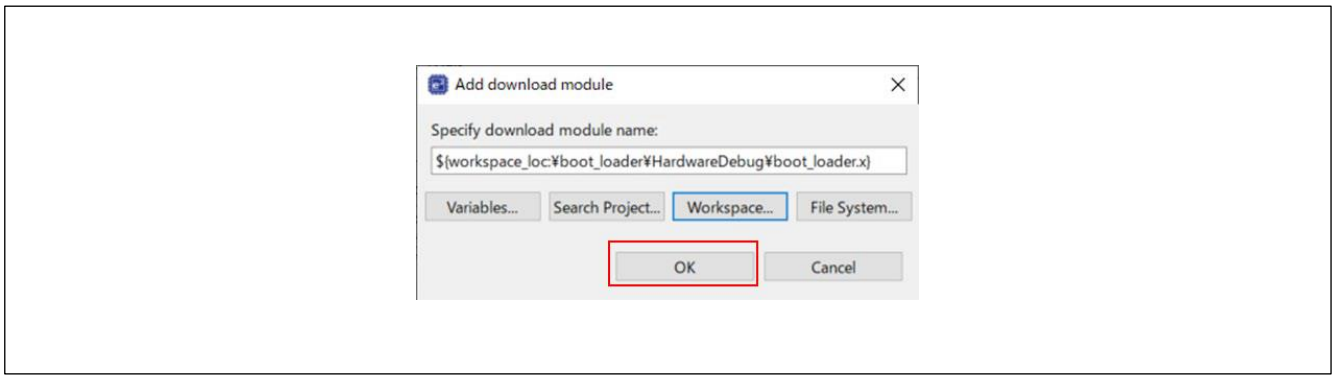


Figure 5.34 Click "OK"

(e) Change the load type of the bootloader (boot_loader.x) from "Image and Symbol" to "Symbol only".

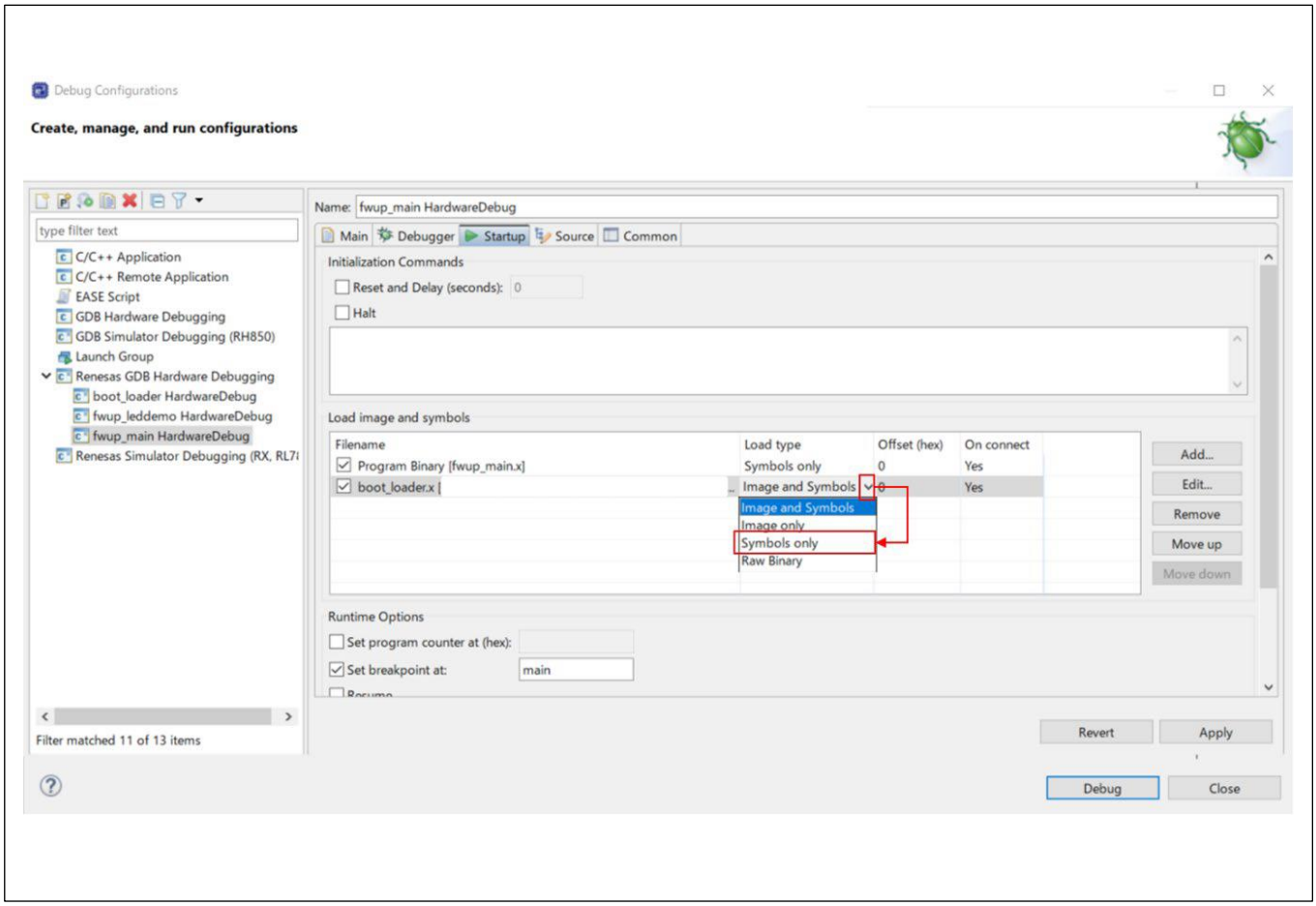


Figure 5.35 Change Load Type

(5) Add the image of the initial image (initial_firm.mot).

Add the image of the initial image (initial_firm.mot) generated in step (2) according to the following procedure.

(a) Click "Add".

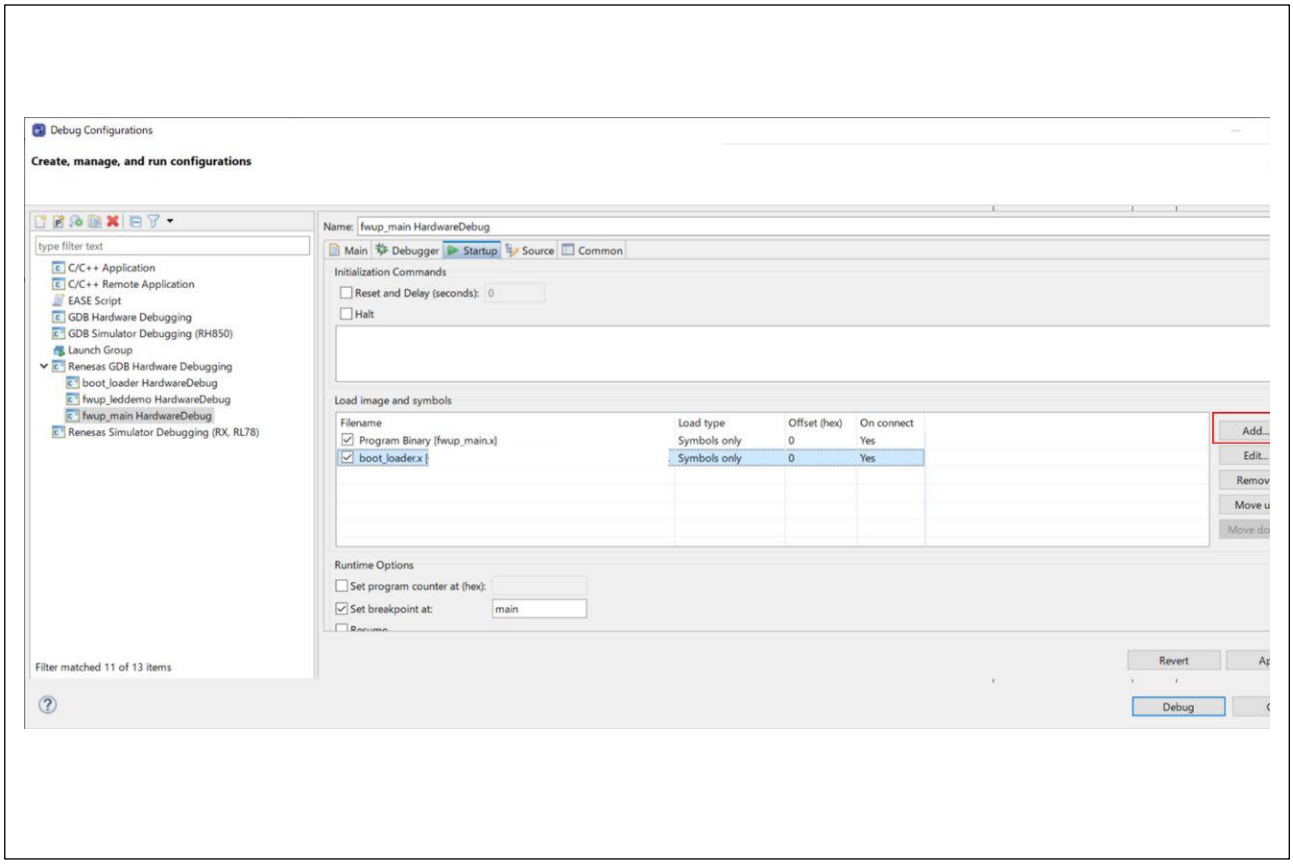


Figure 5.36 Click "Add"

(b) Click File System.

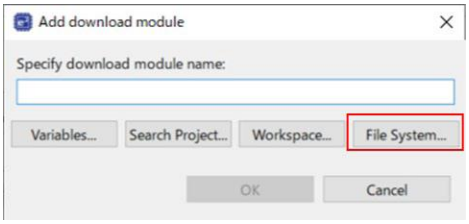


Figure 5.37 File System

(c) Select the initial image (initial_firm.mot) and click "OK".

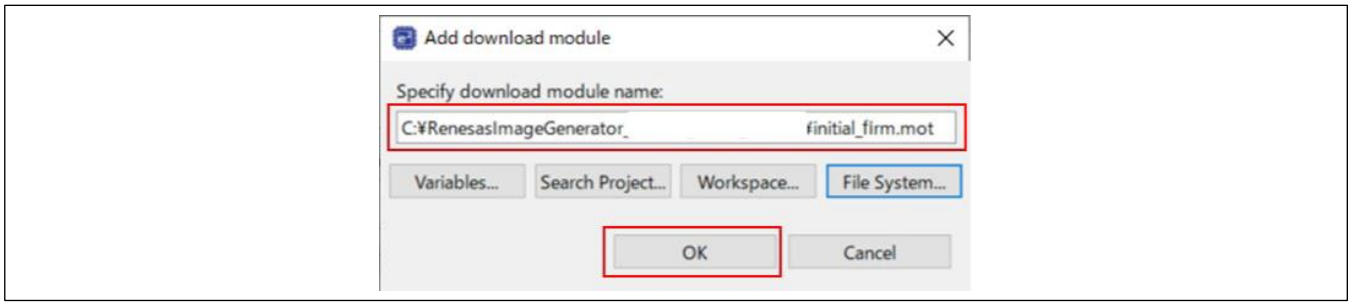


Figure 5.38 Choose Initial Image

(d) Change the load type of the initial image (initial_firm.mot) from "Image and Symbol" to "Image only" and click "OK".

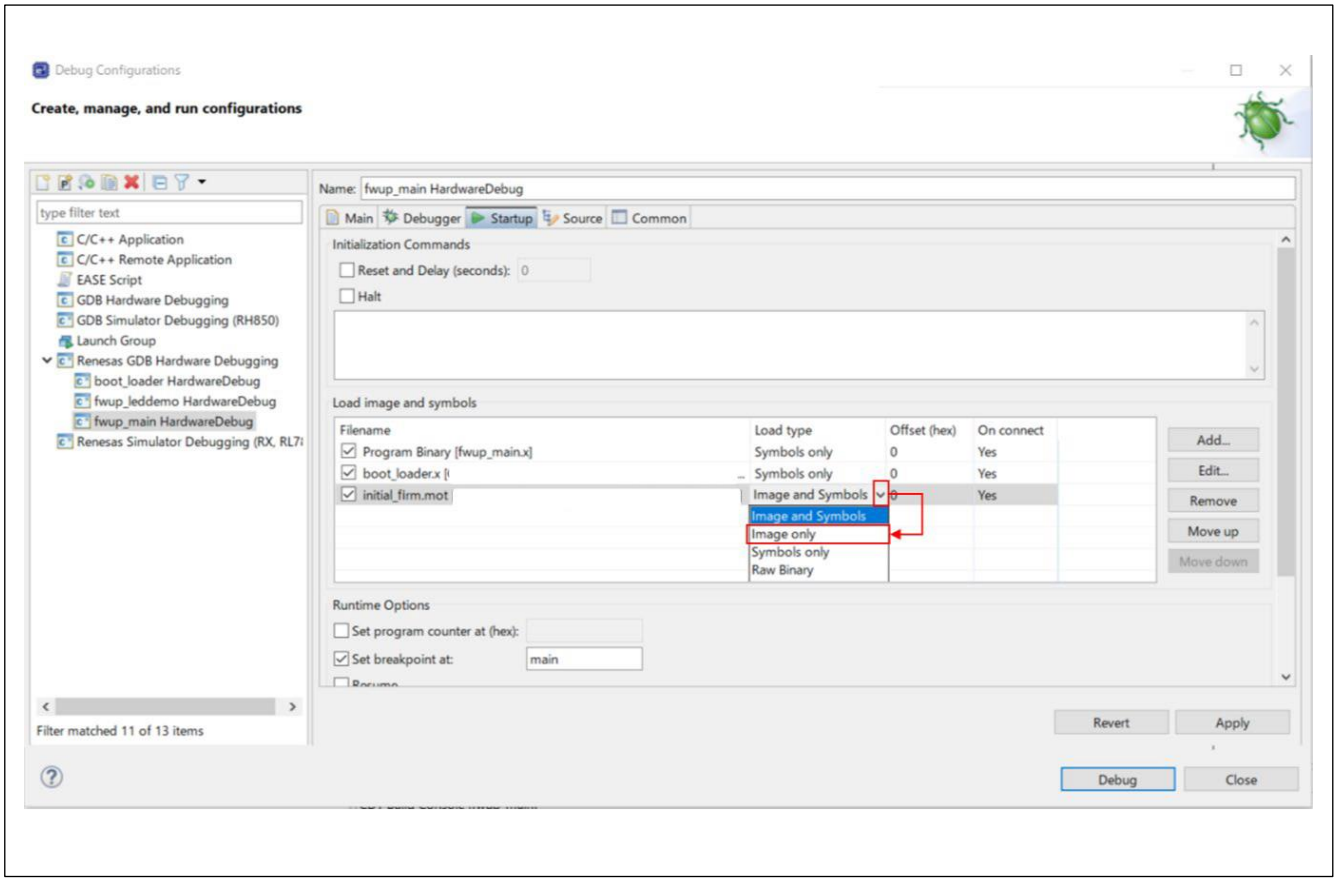


Figure 5.39 Change Load Type

(e) Click "Apply" and click "Debug".

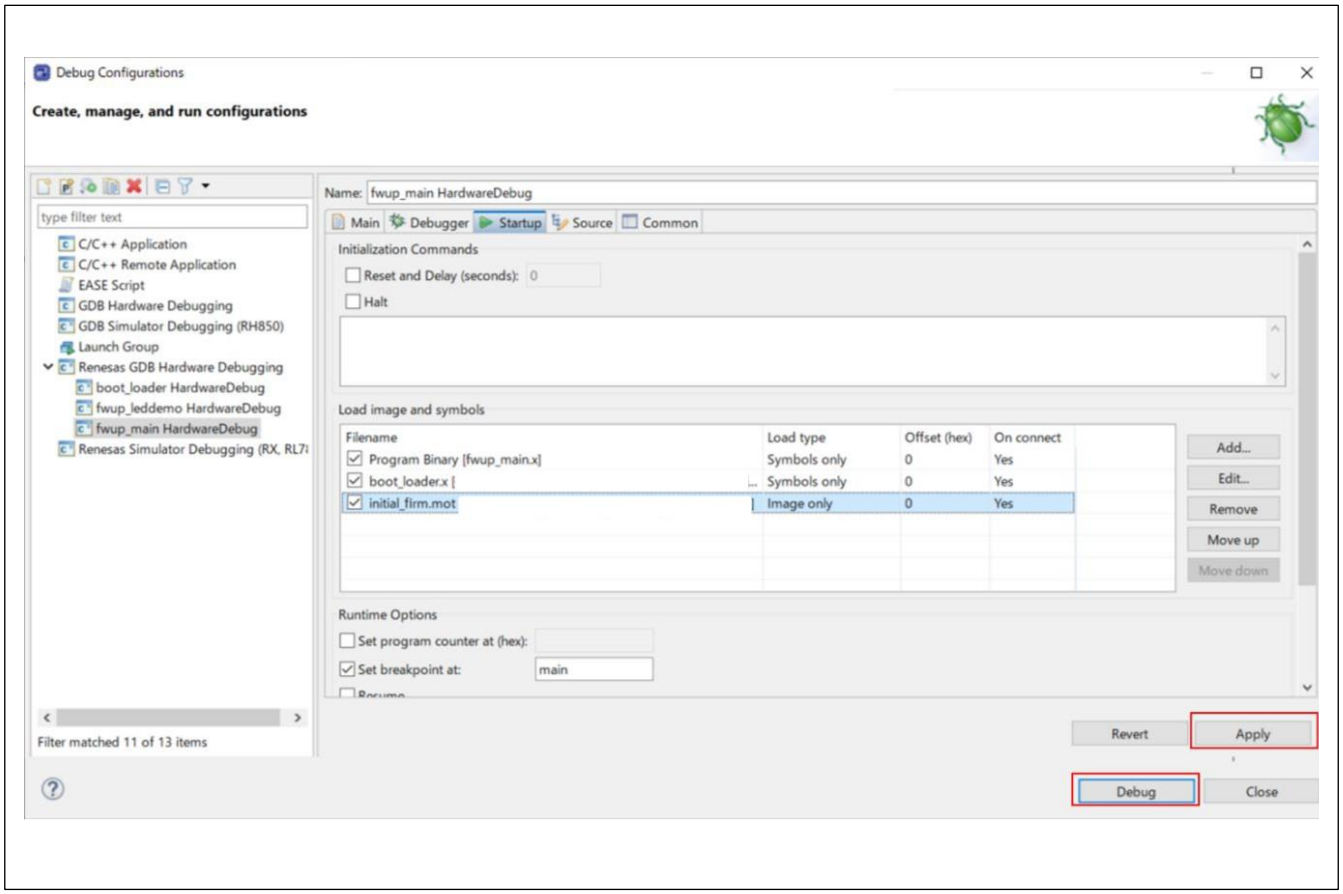


Figure 5.40 Apply and Debug

(6) Start the debugger.

When the debugger starts, it jumps to resetprg.c in the boot_loader project.

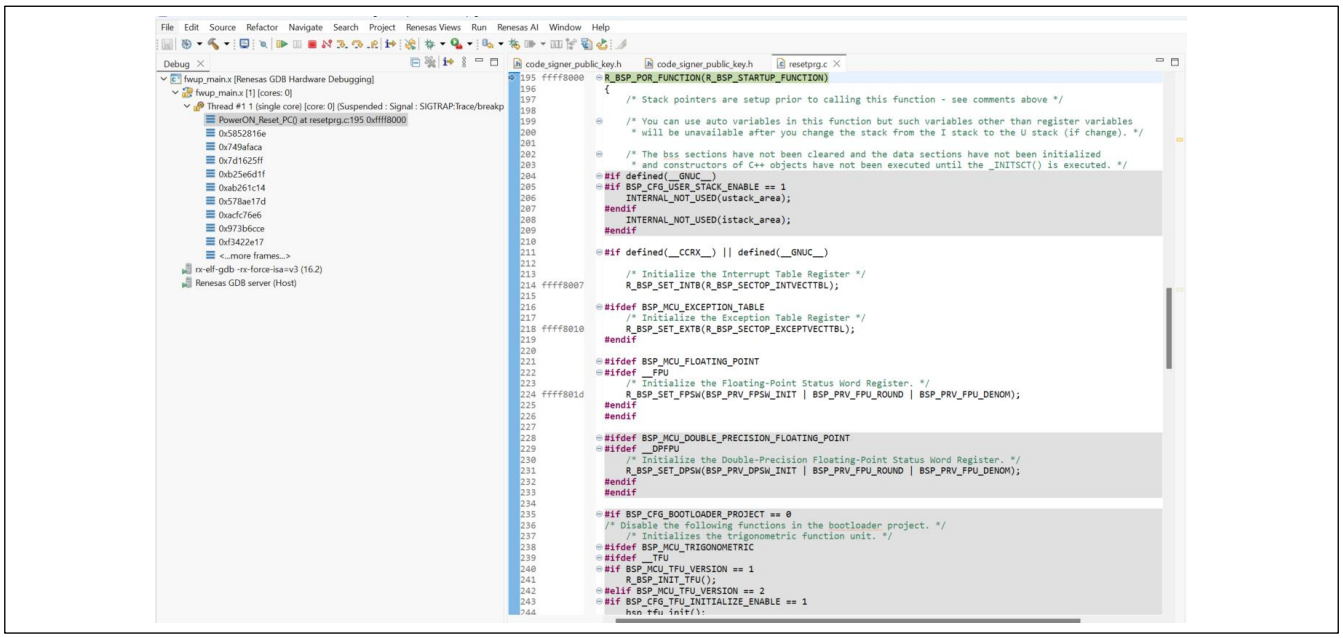


Figure 5.41 Debugger Start

(7) Resume the program.

When you click Resume, the program stops at the beginning of the main() function in boot_loader.c. project.

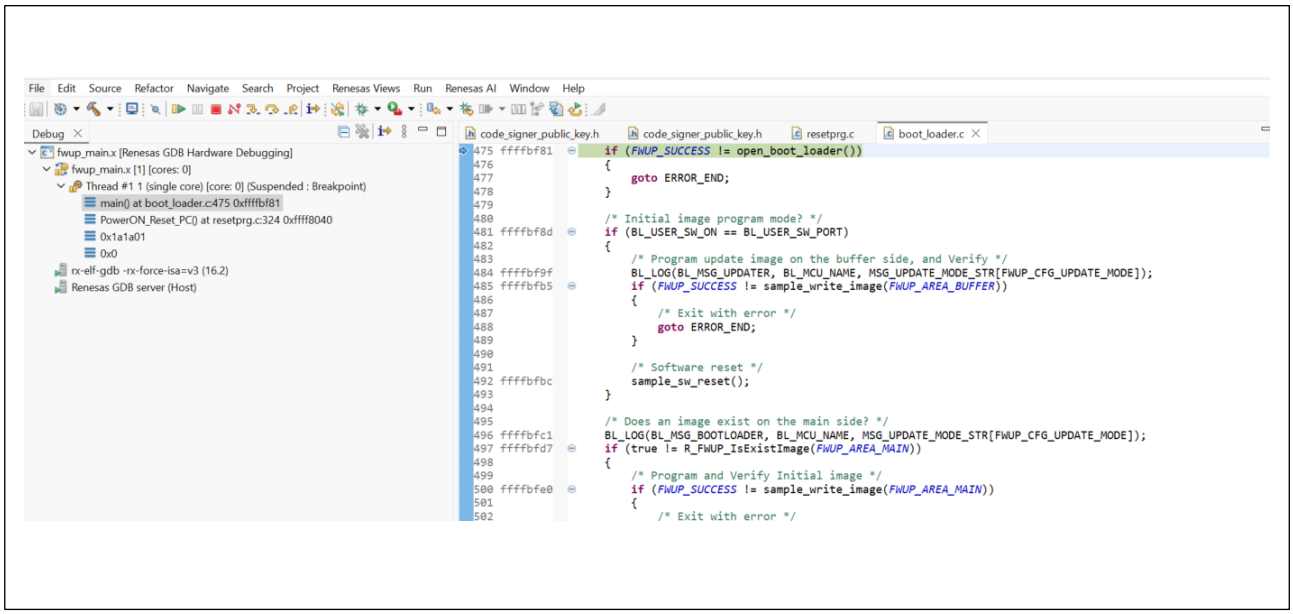


Figure 5.42 Click "Resume"

(8) Set a breakpoint in main() of the fwup_main project.

Set a breakpoint in the following red frame in main() of the fwup_main project.

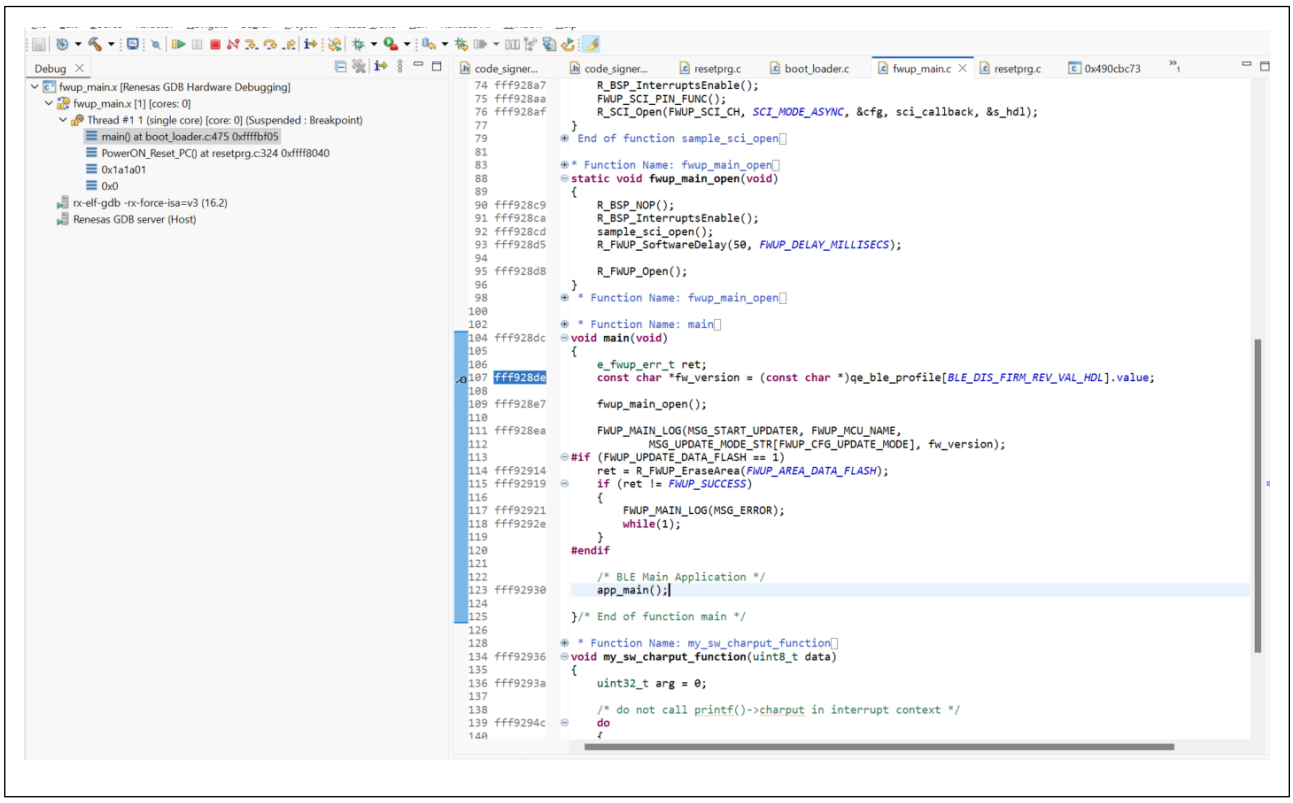


Figure 5.43 Set Breakpoint in fwup_main Project

(9) Resume the program.

Click restart and stop at the breakpoint set in (8).

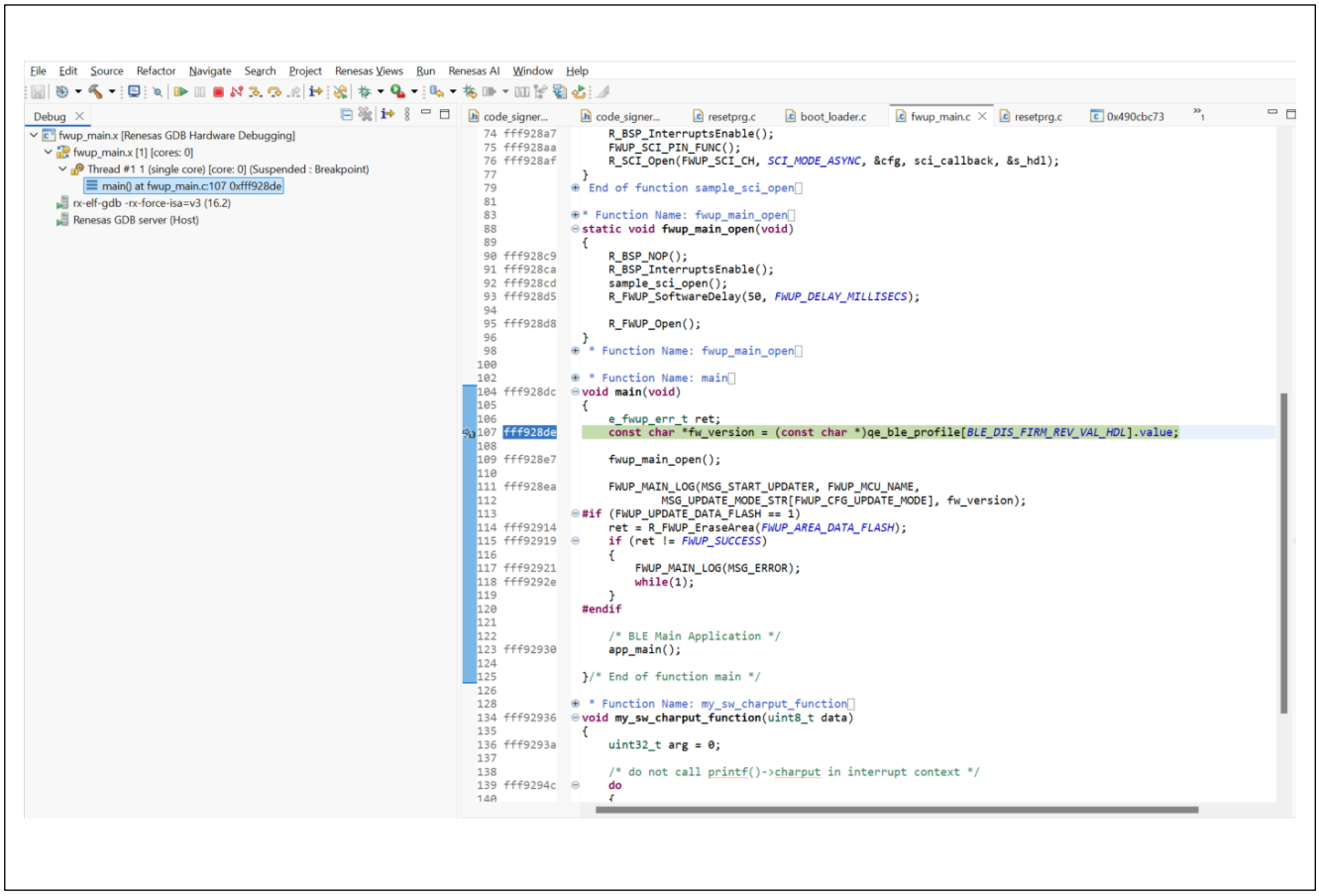


Figure 5.44 Click "Restart"

6. Limitation

Due to hardware connection differences between **DA14531** and **DA14535**, the default GTL UART and POR pin assignments used in r_ble_da1453x_rx module are not compatible across DA14531 devices. Additionally, when **DA14531 is operated in 1-Wire boot mode**, the default pin configuration inside the source code is not valid and must be manually updated to match the required hardware wiring.

(1) Move to this file:

- fwup_main

```
da14531\rx261fpb\w_buffer\e2_ccrx\fwup_main\src\smc_gen\r_ble_da1453x_rx\src\rm_abs\rm_ble_abs_api.h
```

- fwup_leddemo

```
da14531\rx261fpb\w_buffer\e2_ccrx\fwup_leddemo\src\smc_gen\r_ble_da1453x_rx\src\rm_abs\
rm_ble_abs_api.h
```

(2) The previous implementation:

```
/* Configure 2 wire UART */
#if (2 == BLE_CFG_HOST_BOOT_MODE) && (DA14531_DEVICE == BLE_CFG_DA1453X_DEVICE)
#define DA1453X_GTL_UART_RX_PIN     1
#define DA1453X_GTL_UART_TX_PIN     0
#define DA1453X_GTL_POR_PIN         9
#else
#define DA1453X_GTL_UART_RX_PIN     5
#define DA1453X_GTL_UART_TX_PIN     6
#define DA1453X_GTL_POR_PIN         0
#endif
```

(3) The updated implementation:

```
#if (2 == BLE_CFG_HOST_BOOT_MODE) && (DA14531_DEVICE == BLE_CFG_DA1453X_DEVICE)
#define DA1453X_GTL_UART_RX_PIN     1
#define DA1453X_GTL_UART_TX_PIN     0
#define DA1453X_GTL_POR_PIN         9

#elif (2 == BLE_CFG_HOST_BOOT_MODE) && (DA14535_DEVICE == BLE_CFG_DA1453X_DEVICE)
#define DA1453X_GTL_UART_RX_PIN     5
#define DA1453X_GTL_UART_TX_PIN     6
#define DA1453X_GTL_POR_PIN         0

#else
#define DA1453X_GTL_UART_RX_PIN     6
#define DA1453X_GTL_UART_TX_PIN     5
#define DA1453X_GTL_POR_PIN         0
#endif
```

7. Reference Documents

User's Manal: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
RX Family CC-RX Compiler User's Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Website and Support
Renesas Electronics Website
http://www.renesas.com/
Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Revision History	
		Page	Summary
1.00	Dec. 17, 2025	-	First edition issued

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

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

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

Notice

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

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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