ISO-DONGLE-EV1Z Rev.D

Communications Dongle

Introduction

The manual provides an overview of the ISO-DONGLE-EV1Z Rev.D communications dongle bootloader and application software, and it describes the bootloader structure, operation, and programming. The manual contains a step-by-step guide for the application software post-processing and the updating of the device memory without using any external hardware tools. The description of the communication protocol facilitates communication with a Battery Front End (BFE) using the isolated dongle and a custom software that sends commands over a serial port.

Contents

1. Communications Dongle Overview

The ISO-DONGLE-EV1Z Rev.D isolated dongle serves primarily as a communications dongle between a BFE evaluation board and a Graphical User Interface (GUI) available on a workstation (Figure 1). It supports USB, I2C, and SPI communications and is compatible with multiple BFEs and GUIs. The dongle uses a powerful Renesas advanced family RA4 ARM-based microcontroller.

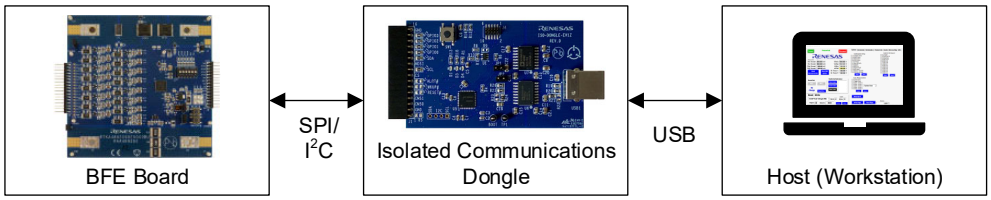

Figure 1. GUI, Communications Dongle, and Battery Front End Connection

1.1 Assumptions and Advisory Notes

- A basic understanding of microcontrollers, embedded systems hardware, battery management systems, and secondary battery cells is assumed.
- A prior experience working with Integrated Development Environments (IDEs) such as e2 studio, Flexible Software Package (FSP), and terminal emulation programs such as Tera Term is assumed.
- Renesas recommends reviewing the *ISO-DONGLE-EV1Z Rev.D Hardware Manual* before changing or developing any software for the dongle.
- Be familiar with the MCU in use, Renesas recommends reviewing the Renesas *RA4E1 Group 32-Bit MCU Datasheet* (see the RA4E1 product page).
- More information about the bootloader operation is provided in the Renesas *RA Family Secure Bootloader for RA2 MCU Series Application Note*.

2. Bootloader

2.1 Structure and Operation

The bootloader of the ISO-DONGLE-EV1Z Rev.D Isolated Communications Dongle enables upgrading or modifying the MCU firmware without requiring specialized programming hardware or tools. It features the MCUboot secure bootloader from the Renesas Flexible Software Package (FSP) to receive a binary file over the USB interface, verify its content integrity, and write the program memory with the application program. The bootloader uses the whole code flash memory of the MCU. Figure 2 shows the flash memory map. It is separated into three major parts: Bootloader, Primary, and Secondary Application Image Slot. The bootloader occupies 64KB of the code flash memory or the first eight blocks (8KB size), containing the MCUboot module, USB driver, HP Flash driver and GPIO driver for the LED indication. Each image slot is 96KB long or 3 blocks (32KB size), which is the maximum memory space available for the application software.

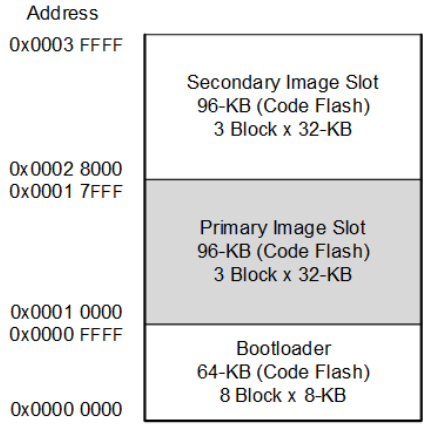

Figure 2. Bootloader Flash Memory Map

Figure 3 shows the operation flowchart of the bootloader. When the CPU is released from reset, a low-level initialization is performed. All three user LEDs activate simultaneously, ON for 500ms and OFF to both perform a LED test and indicate the beginning of the bootloader code execution. Next, the digital input corresponding to the BOOT Test Point is tested for low level. If this condition is true, the USB interface initializes to create a virtual COM port and activates the software image download process. The MCU waits for a connection with terminal emulator software and receives the start of download command, Carriage return' (Enter). The bootloader starts to store every byte received by the virtual serial connection in the Secondary Image Slot by portions of 32KB (1 block), and if the transferred image size is different than 96KB or any unexpected behavior is encountered, the operation is terminated and LED D4 is illuminated.

When the image download completes successfully, LED D3 is illuminated, and the MCU halts operation and waits for manual restart. This is also the moment when the level of the digital input (BOOT TP) must be changed to high (remove the jumper wire). Table 1 shows the various states indicated by the LEDs, and Table 4 shows the virtual COMM port settings.

Important: Do not set a higher baud rate because this can lead to buffer overrun and compromise the image transfer.

After a successful bootloader startup, and LED test and if the level of the digital input corresponding to the BOOT Test Point is high (default), the MCU proceeds to software image boot up. The bootloader checks the Secondary Image Slot memory space. If an image is available there, it is validated by checking integrity and authenticity. After successful authentication, the bootloader copies the new image using the overwrite update method where the entire content of the Primary Image Slot is overwritten with the contents of the Secondary Image Slot. The active image is always executed from the Primary Image Slot. This method is fail-safe, resistant to power-cut failures, and with less memory overhead compared to the other available update methods. Conversely, it does not support image pre-testing and automatic fallback mechanism. When the Primary Image Slot memory space is successfully overwritten, the bootloader executes the new image. If no new image is detected in the Secondary Image Slot, the bootloader directly authenticates and boots up the Primary Software Image.

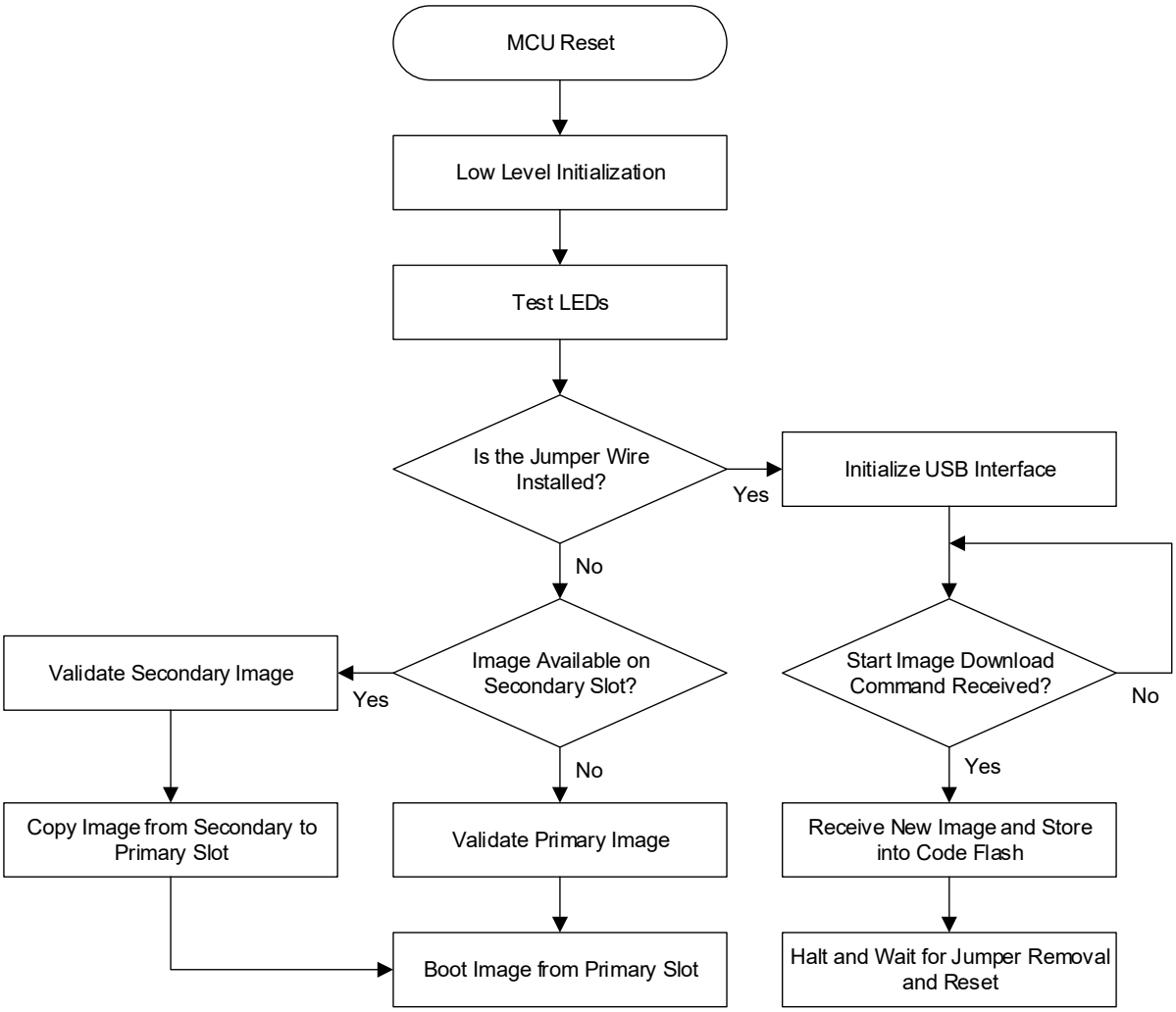

Figure 3. Bootloader Flowchart Diagram

Table 1. Bootloader LED States Description

Condition	LED States	
	D3 (Red)	D4 (Red)
LEDs blink test on startup.	ON	ON
Proceeding to normal software image boot.	OFF	OFF
Software image download mode is entered.	Blinking slow	OFF
Waiting to receive the software image by the virtual serial connection.	Blinking fast	OFF
Receiving software image in process.	Blinking fast	Blinking fast (antiphase with D3)
Software image download is completed.	ON	OFF
An error has occurred during image download.	OFF	ON

Table 2. Serial Connection (Terminal) Settings

Parameter	Value
New Line (Receive)	CR
New Line (Transmit)	CR
Terminal ID	VT100
Baud Rate	9600
Data Bits	8 bits
Parity	none
Stop Bits	1 bit
Flow Control	none

2.2 Bootloader Programming

The ISO-DONGLE-EV1Z Rev.D Isolated Communications Dongle is programmed by default with the bootloader software. However, if using any external debugging tools during custom software development, it is erased or overwritten in the code flash memory. This section describes how to restore the bootloader in the MCU memory.

Renesas E2 Emulator/ E2 Emulator Lite with 10-pin Cortex® Debug Connector and the following software available on a workstation running Windows 10 are required:

- Renesas Flash Programmer V3.11.00 or later
- **ra4e1_iso_dongle_mcuboot_v_1_0.srec** binary file v1.0 or later

Complete the following bootloader programming process steps:

1. On the dongle, ensure that jumper JP1 is installed.
2. Connect the Renesas E2 Emulator or E2 Emulator Lite to the JTAG connector J4 of the dongle board (Figure 4).

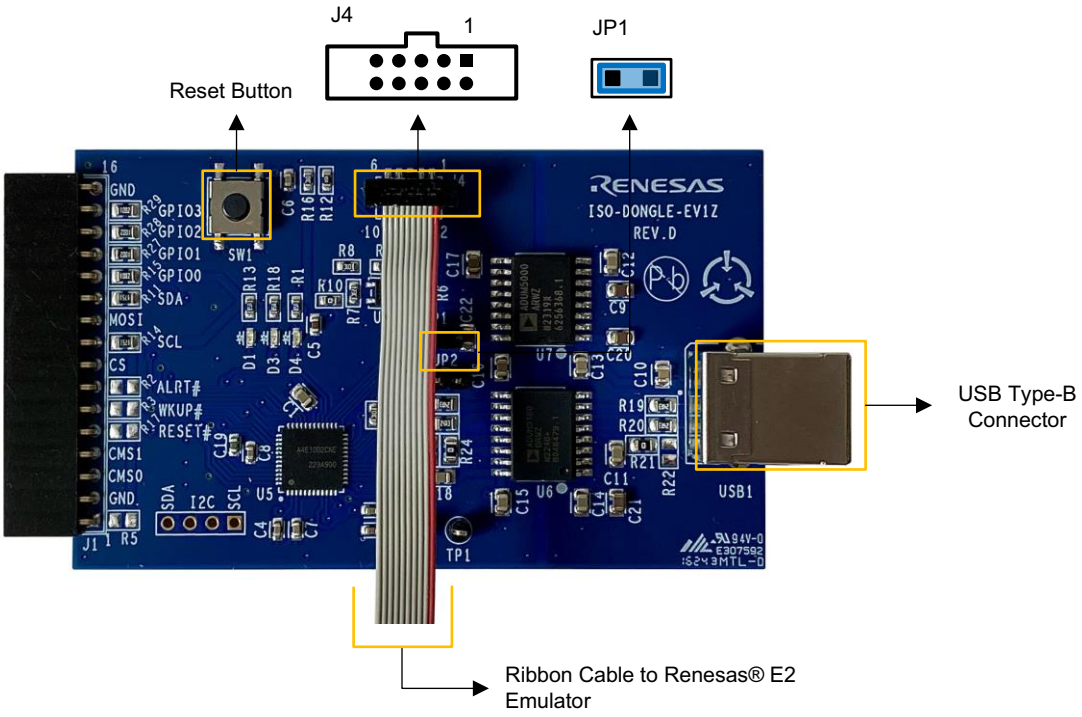

Figure 4. Isolated Communications Dongle with Attached Renesas E2 Emulator

3. Connect the USB cable between the PC and the USB Type-B connector of the board. Ensure the power ON indicator LED D1 is illuminated.
4. Run the Renesas Flash Programmer, and on the top menu, select **File** > **New Project...** from the **Microcontroller** dropdown menu select **RA** (Figure 5). In the **Project Name** field, enter RA4E1_Dongle. From the **Tool** dropdown menu, select **E2 emulator** or **E2 emulator Lite** and click the **Connect** button.

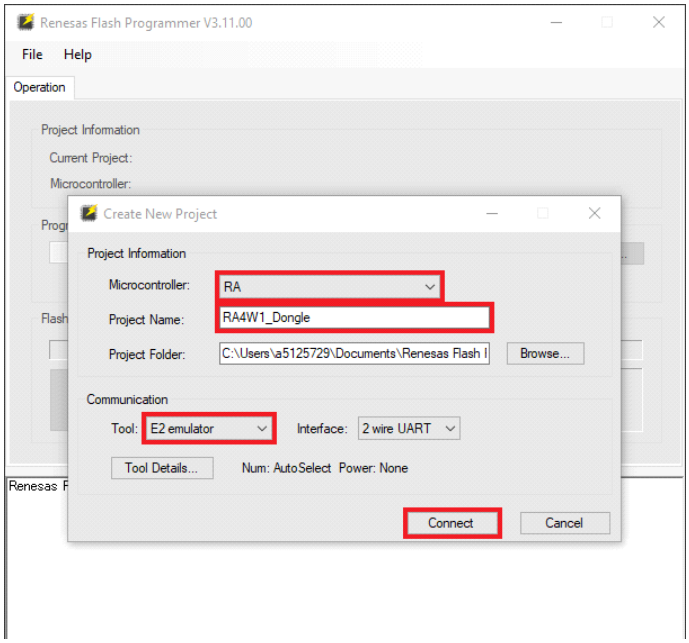

Figure 5. Creating a New Project in Renesas Flash Programmer

5. In the main window, select the **Operation Settings** tab and verify that in **Command** section that **Erase**, **Program**, and **Verify** are selected (Figure 6).

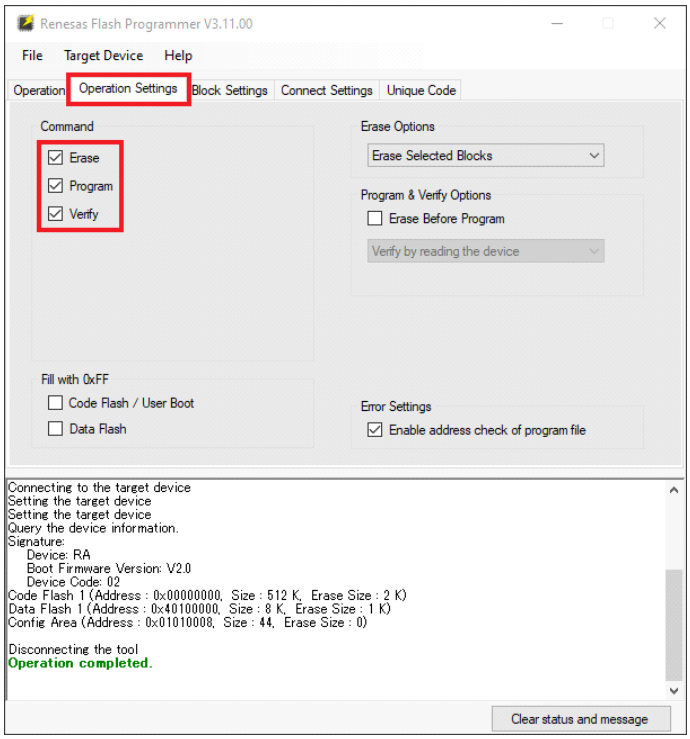

Figure 6. Operation Settings in Renesas Flash Programmer

6. Select the **Connect Settings** tab and click on the **Tool Details...** button (Figure 7). In the pop-up window, verify that option **None** is selected in the **Power Supply** section. Click on the **OK** button to close the pop-up window.

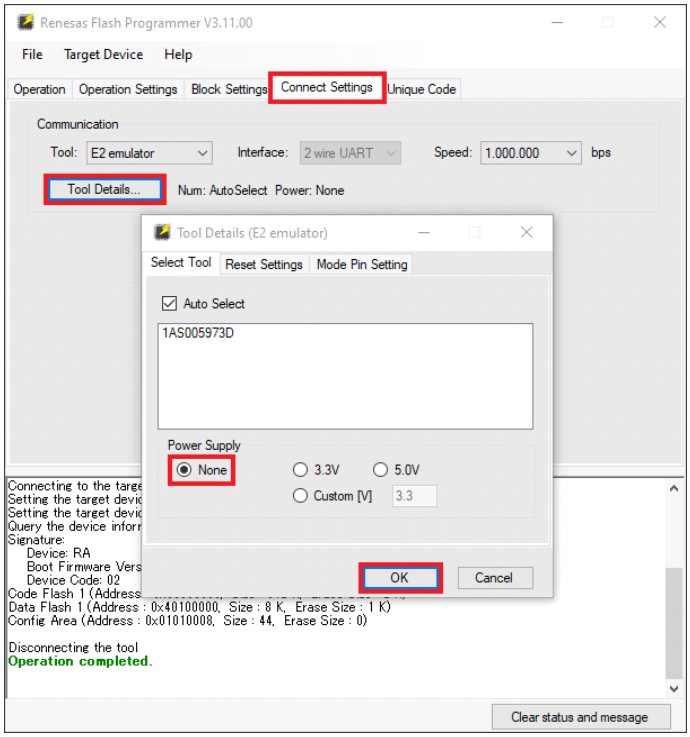

Figure 7. Connection Settings in Renesas Flash Programmer

7. Select the **Operation** tab and click on the **Browse…** button (Figure 8). In File Explorer, navigate to the location of the bootloader binary file (.srec), select it, and click the **Open** button. Click the **Start** button to download the file into the code flash memory.

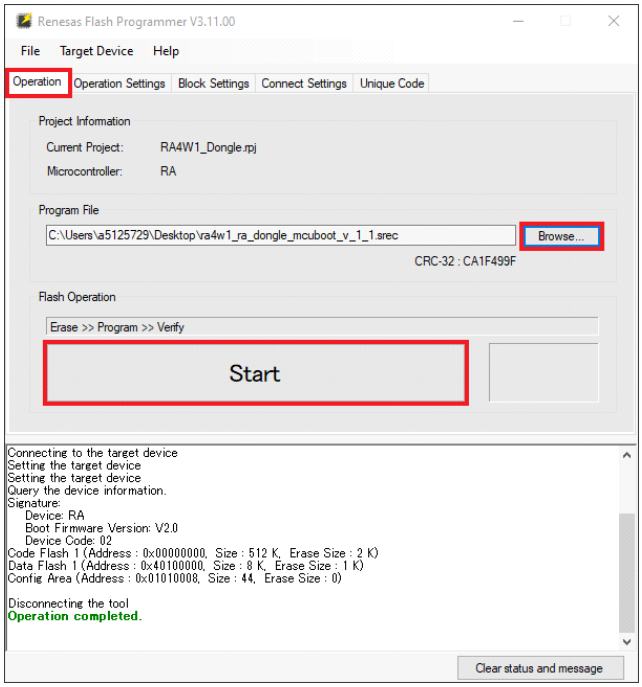

Figure 8. Selection of the Program File

8. When the process completes successfully, a green OK message in a rectangular window appears next to the **Start** button (Figure 9). Disconnect the USB cable. Disconnect the Renesas E2 Emulator or E2 Emulator Lite from the dongle and close the Renesas Flash programmer. The bootloader programming process is completed. ***Important***: Always first unplug the USB cable and after unplug the Renesas E2 Emulator or E2 Emulator Lite from the dongle.

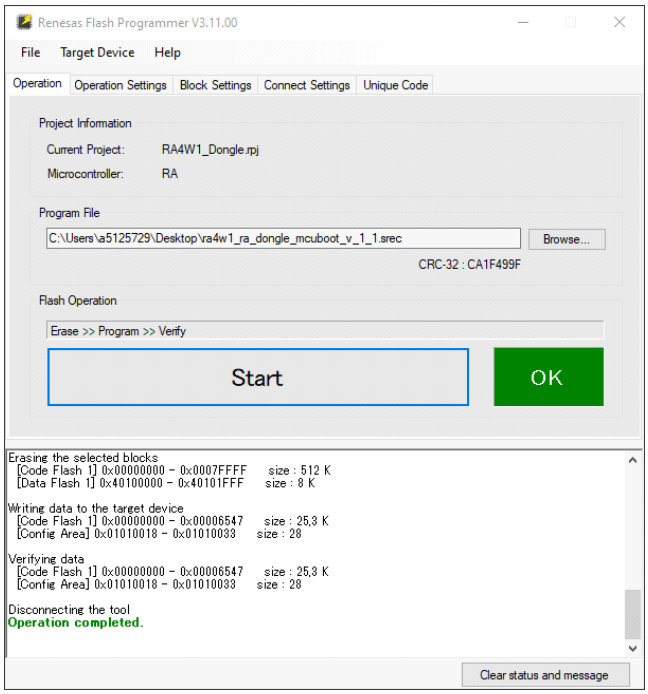

Figure 9. Successful Programming Win

3. Software Image Update

The software image (referred to as image) in the communication dongle can be updated or modified without using any additional external software programmers. To initiate the software image update process, first ensure that the required image is available on the PC and the terminal emulator program Tera Term is installed. Tera Term can be downloaded from the Tera Term Home Page.

The image update process must follow these steps:

1. Connect the BOOT Test Point to the TP1 (GND ISO) using a jumper wire as shown in Figure 10.

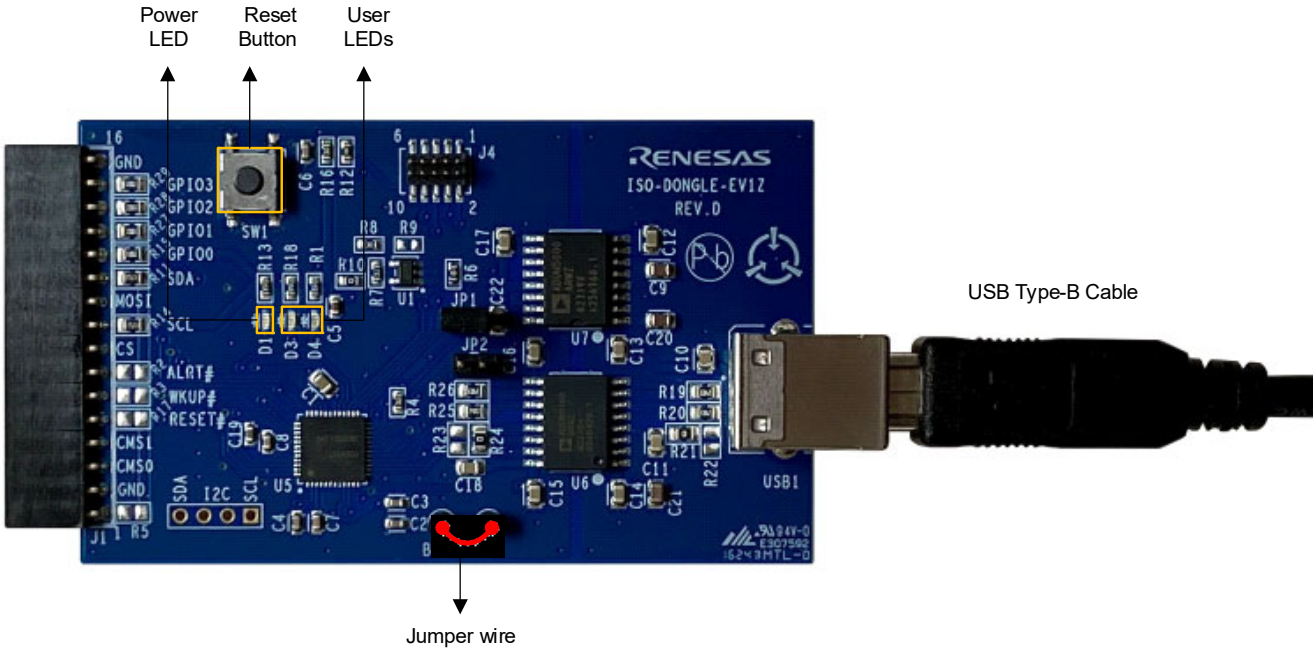

Figure 10. Connection of the ISO-DONGLE-EV1Z Rev.D Isolated Communications Dongle for the Software Image Update

2. Connect the USB cable between the PC and the USB Type-B connector of the board. Verify that LED D3 is blinking slowly indicating that the bootloader is in Image Update mode.
3. Open Tera Term. A new connection should automatically open (Figure 11), if not, select **File** > **New Connection**. From the window, select **Serial** and verify that the **USB Serial Device COM** port is selected from the dropdown list. Click the **OK** button to open the connection.

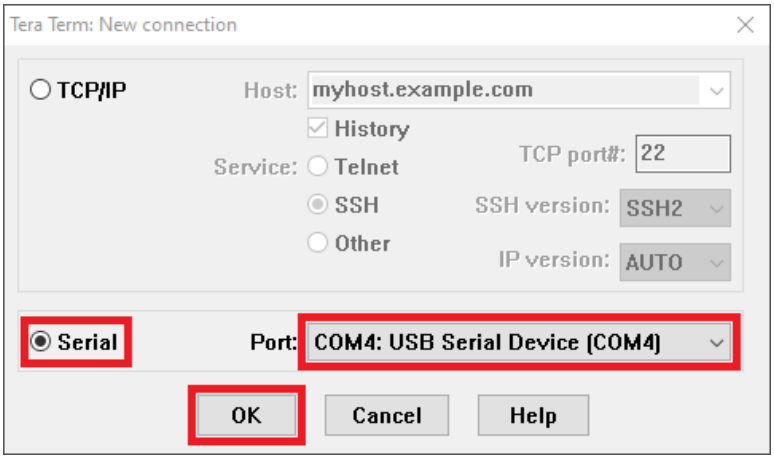

Figure 11. Tera Term New Connection Window

4. In the top menu of Tera Term, select **Setup** > **Serial Port**. In the window (Figure 12), verfiy that the connection settings match the values from Table 5 and click on the **New setting** button.

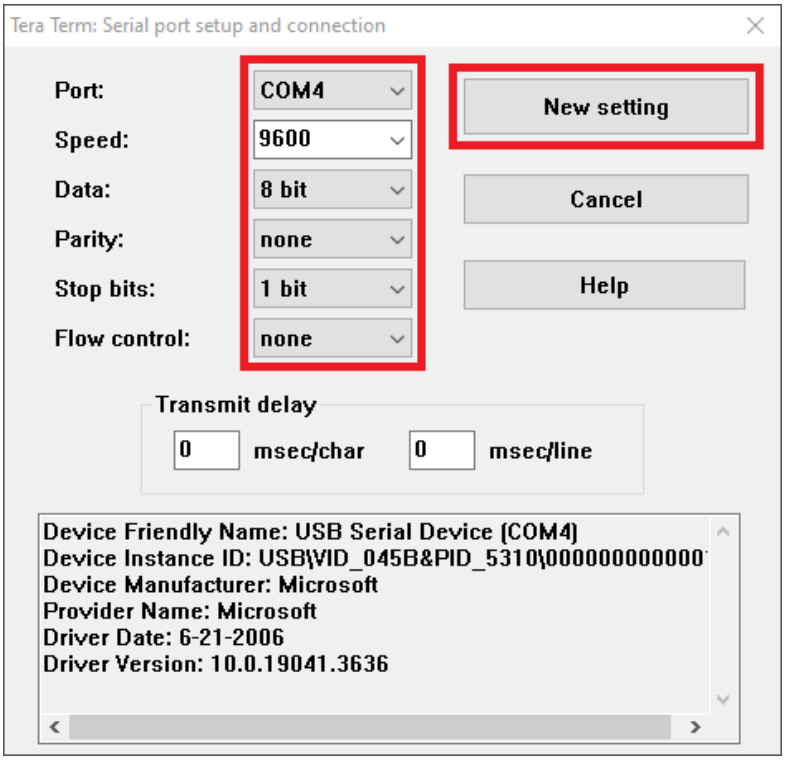

Figure 12. Tera Term Serial Port Setup and Connection Window

5. Press **Enter** to activate Image Download Mode. LED D3 starts blinking fast and a READY! message is displayed on the console screen (Figure 13). The bootloader waits to receive the image file by the USB serial interface. Each subsequent byte is interpreted as a part of the Image. ***Warning!***: Be careful not to press any other buttons or the process compromises and must be restarted.

Figure 13. Console Screen – Ready for Software Image Download

6. Select **File** > **Send file...** from the menu. On the opened window (Figure 14), select **Binary**, navigate to the location of the software image file, select it, and click the **Open** button. Now the terminal emulator program sends the whole file. LEDs D3 and D4 blink in antiphase, and a pop-up window displays the transfer progress, which takes approximately 10 seconds.

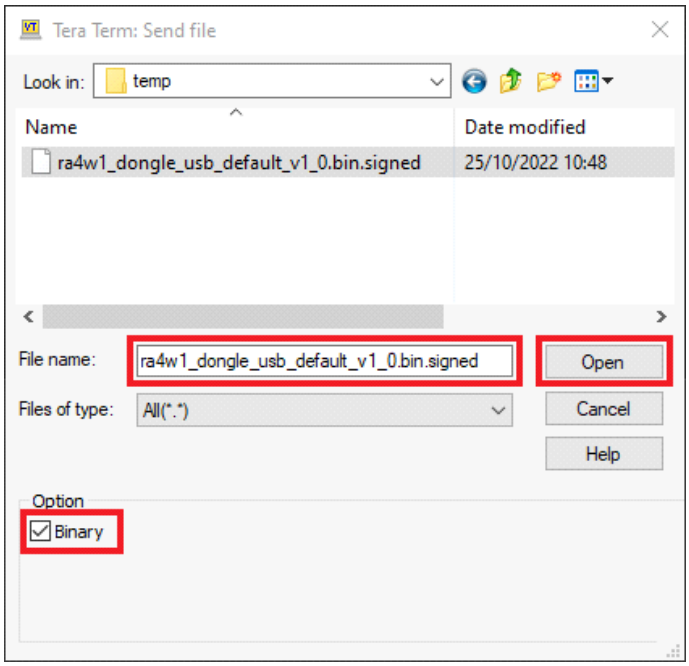

Figure 14. Selection of Software Image

7. When the software image download has finished, D3 remains illuminated and a DOWNLOAD COMPLETE! message is printed in the console screen (Figure 15). The USB cable must be unplugged and the jumper wire removed. Next time the MCU powers up, it copies the new software image, and boots from it. This completes the software image update process. If the red LED D4 is illuminated, either the bootloader becomes irresponsive or the image transfer sticks conveying that there is something wrong with the image update process. Unplug the USB cable and restart the process from Step 1.

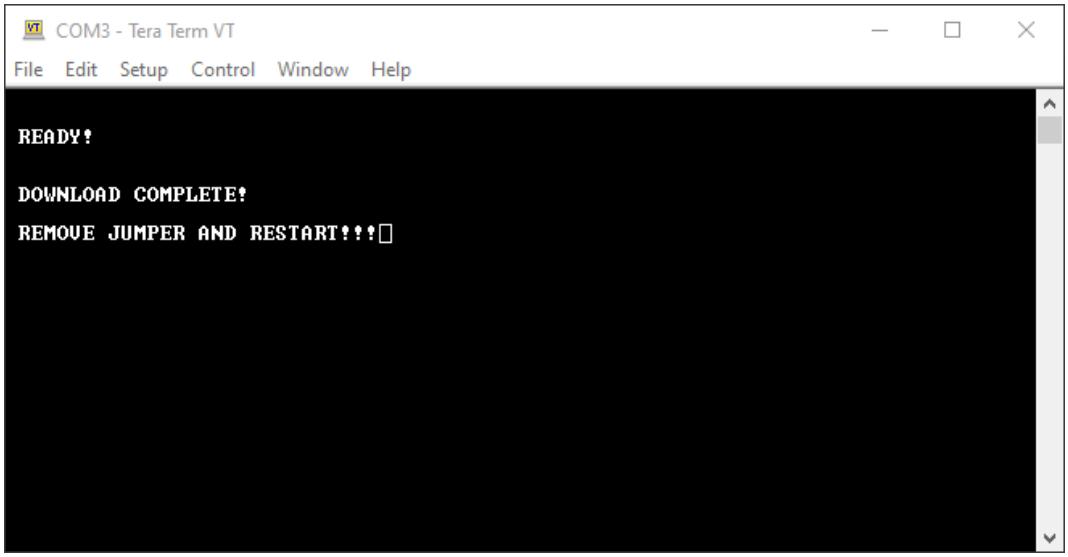

Figure 15. Console Screen – Software Image Download has Finished

4. Generation of Software Image

Developing and running customized software (application) on the dongle using the bootloader to avoid the need of any debugger tools requires the generation of a software image that meets the bootloader size and verification requirements. This section describes the process of generation of a software image from a conventional project in e^2 studio. Initially, the following development tools and software running on Windows® 10 are required:

- e^2 studio Development Environment (IDE) 2022-07 or greater with GCC Arm® Embedded toolchain.
- Renesas Flexible Software Package (FSP) 5.1.0 or later.
- Python 3.11.0 or later.
- **ra4e1_iso_dongle_mcuboot_v_1_0.zip** bootloader software project v1.0 or later.

Important: Ensure that Python is installed to all users and it is added to Windows PATH (**Control Panel** > **System** > **Advanced System Settings** > **Advanced** > **Environment Variables**). Otherwise, python scripts cannot run through e^2 studio IDE.

If this procedure has already been completed and the MCU boot software project and python environment are already set up, go directly to Step 10.

The image generation process must follow these steps:

1. Import the MCUboot software project that contains the python script and is used in the image generation process. *Note*: This is the bootloader project not the application software that runs from the target image. Verify that the **ra4e1_iso_dongle_mcuboot_v_1_0.zip** project archive file is already available. Right-click in the e^2 studio Project Explorer view and select **Import** from the menu to open the Import Wizard (Figure 16).

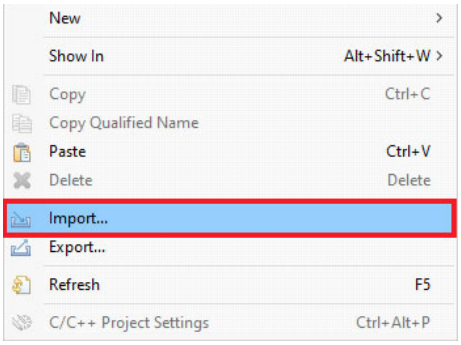

Figure 16. Right-Click Menu to Import the Bootloader Project

2. From the Import Wizard window, select **Existing Project into Workspace** and click the **Next >** button (Figure 17).

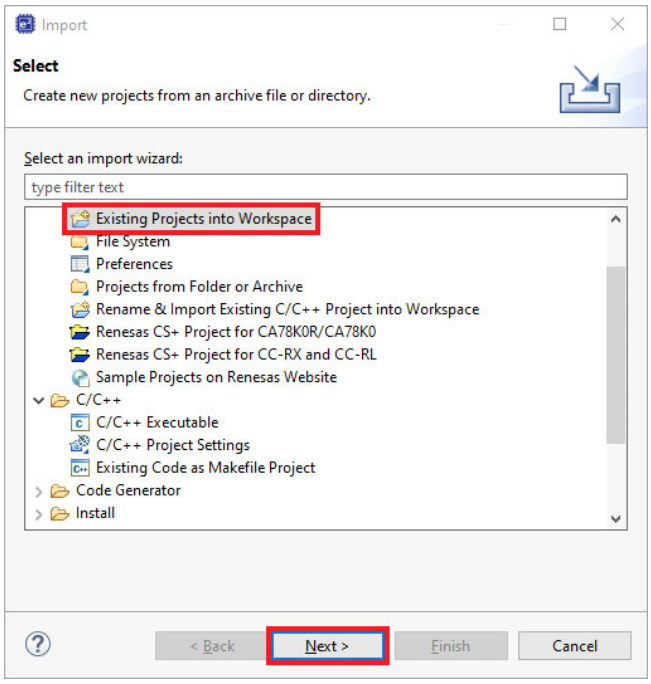

Figure 17. Selection of the Import Wizard Option

3. In the Import Projects window, select the **Select archive file** option and click on the **Browse…** button (Figure 18).
   a. In the File Explorer that appears, navigate to the location of the bootloader project archive file (.zip), select it, and click on the **Open** button.
   b. Click the **Finish** button to complete the import process and close the Import Wizard.

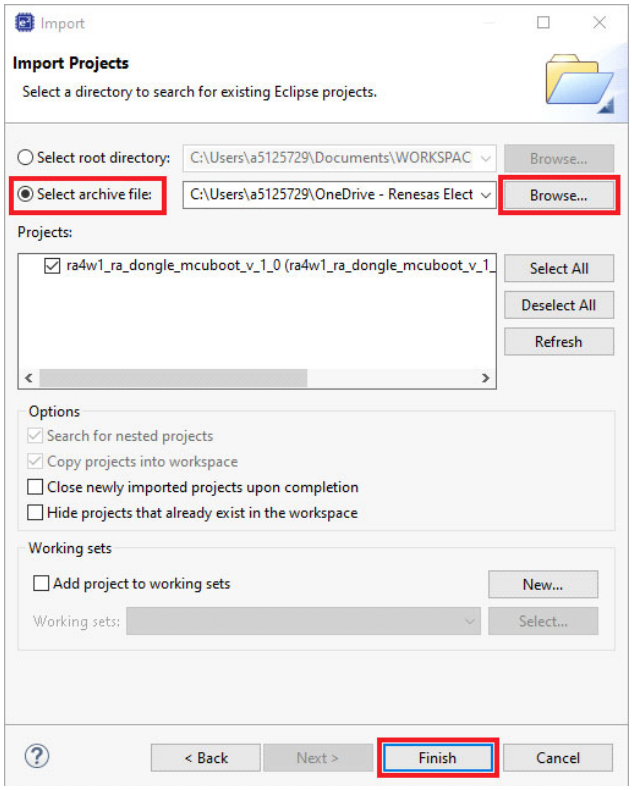

Figure 18. Selection of the Bootloader Project

4. The project is now imported into the e^2 studio workspace. Double-click on **configuration.xml** to open the FSP Configurator and click the **Generate Project Content** button to generate the hardware specific project files that are required to run the project on the selected hardware (Figure 19).

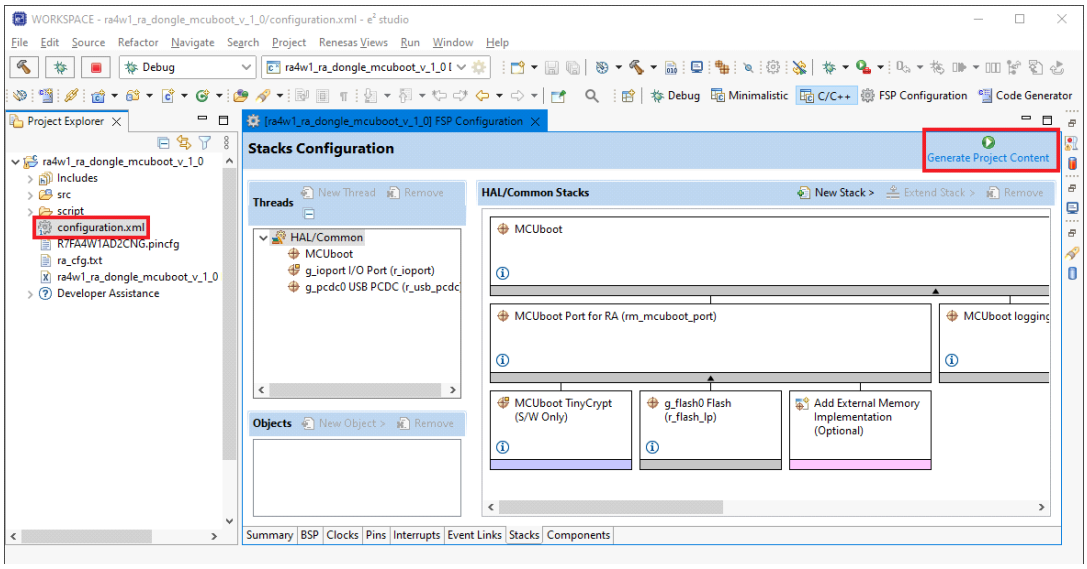

Figure 19. Generation of Hardware Specific Project Files

5. Configure the Python Signing Environment. Signing the application image is done using a post-build step in e^2 studio using the image signing tool **Imgtool.py**, which is included with MCUboot.
   c. In the e^2 studio Project Explorer, navigate to the **ra4e1_iso_dongle_mcuboot_v_1_0\ra\mcu-tools\MCUboot** folder (Figure 20).
   d. Right-click the MCUboot folder and select **Command Prompt**. This opens a command window with the path set to the **\mcu-tools\MCUboot** folder.

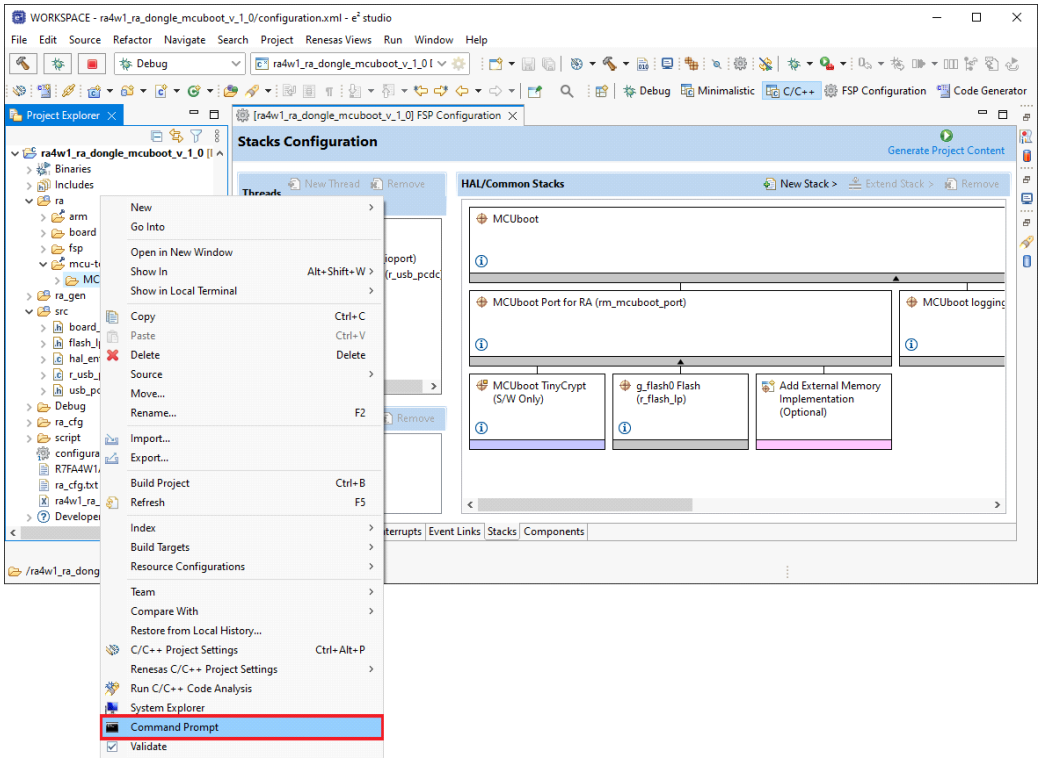

Figure 20. Opening Command Prompt in the MCUboot Folder

6. Enter the following command to update pip:

```
python -m pip install --upgrade pip
```

7. Next, in the command window, enter the following command line to verify and install all the MCUboot dependencies (Figure 21):

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

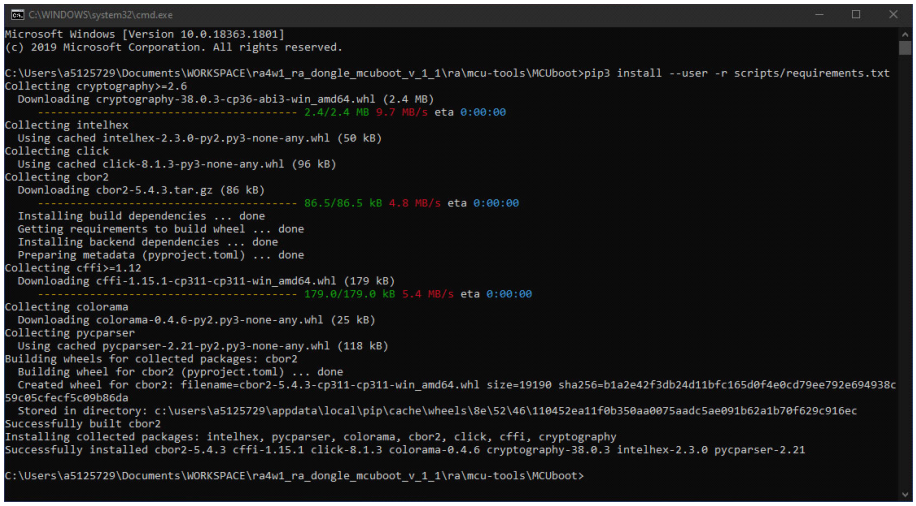

Figure 21. Installing the MCUboot Dependencies

8. Close the Command Prompt.
9. Ensure that the **ra4e1_iso_dongle_mcuboot_v_1_0** project is active (selected in Project Explorer) and on the top menu, select **Project** > **Build Project** and wait until the process completes.
10. Configure the existing application to use the bootloader project. The software project that runs on the dongle is duplicated and renamed so that the original can remain debugged using an external debugging tool during code development. ***Caution***: The binary size must be smaller or equal to 96KB. This is the fixed image size and dedicated memory space in the code flash. The binary size cannot be exceeded.
    Next, right-click in the e^2 studio Project Explorer and select **Import** from the menu to open the Import Wizard (Figure 16).
11. From the Import Wizard window, select **Rename & Import Existing C/C++ Project** into Workspace and click on the **Next >** button (Figure 22).

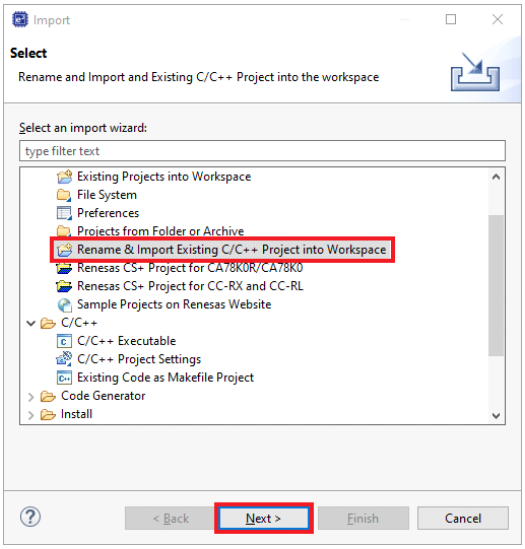

Figure 22. Selection of the Import Wizard Option

12. In the Rename & Import Project window, enter a project name (Figure 23). Renesas recommends using the same name as the original project with a suffix ***_app**.
    a. Select the **Select root directory** option and click on the **Browse…** button.
    b. In the File Explorer, navigate to the location of the existing application project, select it, click the **Select Folder** button, and select the desired project in the **Projects** section.
    c. Click the **Finish** button to complete the import process and close the Import Wizard.

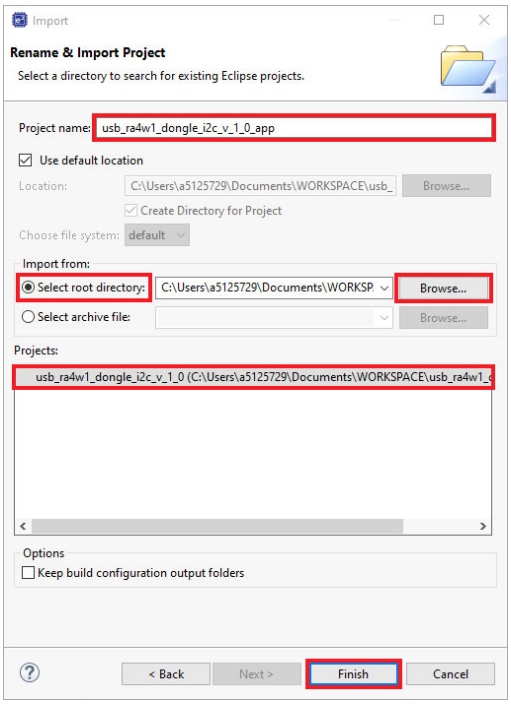

Figure 23. Selection of the Sample Project

13. The project is now duplicated into the e^2 studio workspace and must be configured. In the Project Explorer, right-click on the application project folder and select **Properties** to open the project properties window (Figure 24).

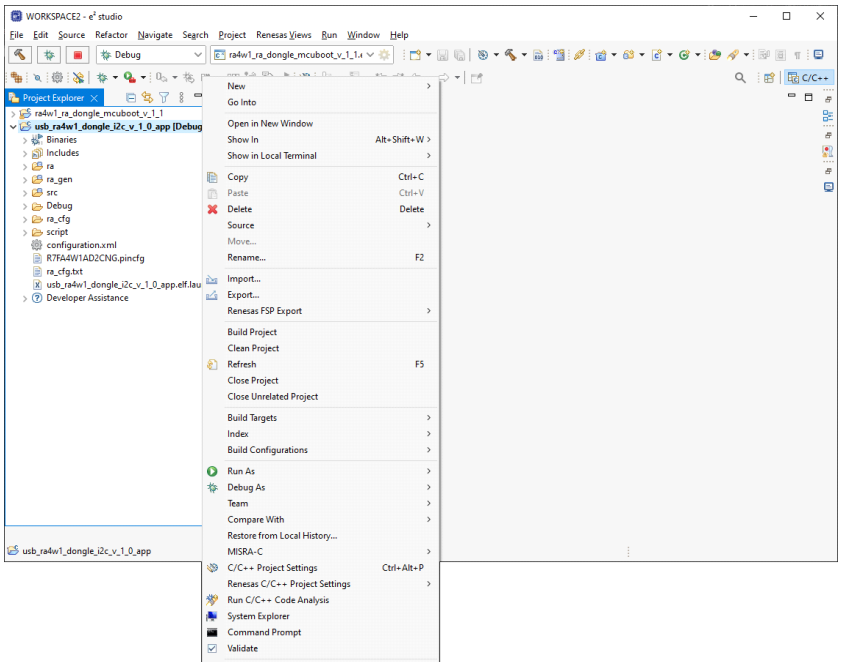

Figure 24. Opening Project Properties

14. Select **C/C++ Build** > **Build Variables** and click on the **Add...** button.
    a. In the **Variable name** field, enter BootloaderDataFile.
    b. Check the **Apply to all configurations** box (Figure 25).
    c. Change the **Type** to File, and for **Value**, enter the following relative path to the *.bld file:
    ${workspace_loc:ra4e1_iso_dongle_mcuboot_v_1_0}/Debug/ra4e1_iso_dongle_mcuboot_v_1_0.bld
    Note: If the bootloader project name or location are different, the value format is as follows:
    ${workspace_loc:<boot_project_name>}/Debug/<boot_project_name>.bld.
    d. Click the **OK** button and the **Apply** button to save the change of the properties.

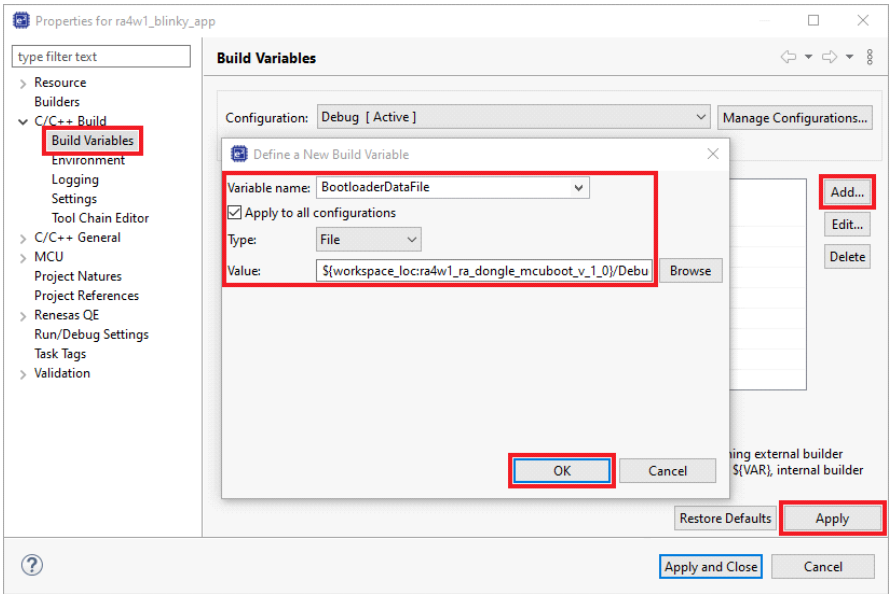

Figure 25. Adding a Build Variable to Use the Bootloader

15. Select **C/C++ Build** > **Environment**, click on the **Add...** button.
    a. In the **Name** field, enter MCUBOOT_IMAGE_VERSION and set **Value** to the actual application project version number (Figure 26).
    b. Click the **OK** button and the **Apply** button to save the change of the properties.

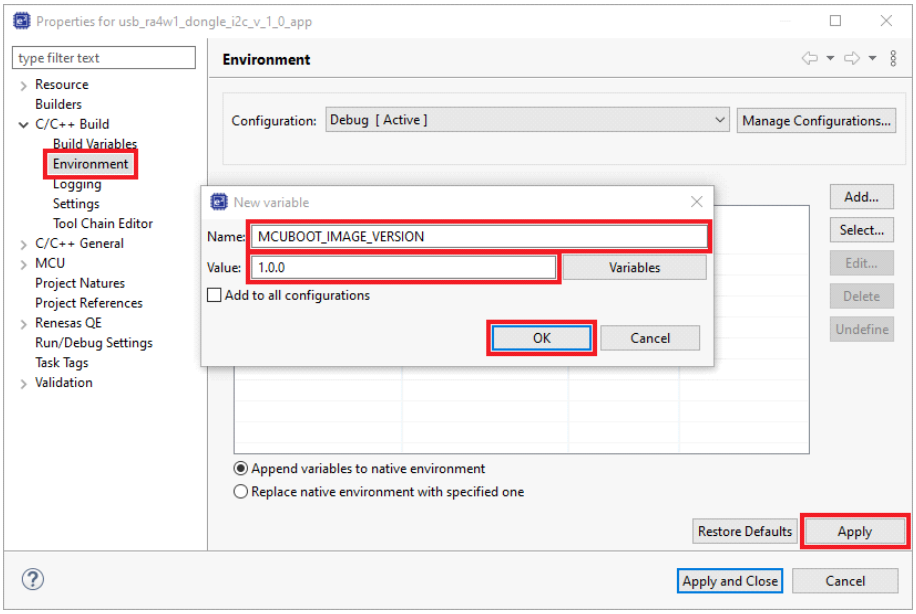

Figure 26. Adding a Project Version Number

16. Select **C/C++ Build** > **Settings** and select the **Build Steps** tab (Figure 27).
    a. In the **Command(s)** field, enter the following pre-build command: **del ${ProjName}.elf** to always delete the *.elf file.
    b. Click the **Apply** and **Close** button to save the change of properties and close the window.

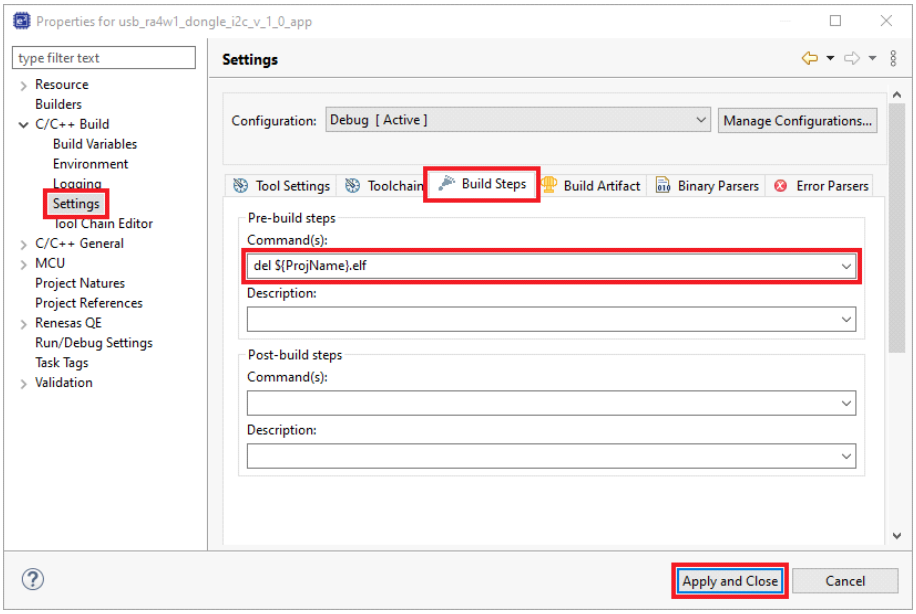

Figure 27. Adding a Pre-Build Command

17. Verify the project is active (selected in the Project Explorer). From the top menu, select **Project** > **Build Project** and wait until the process completes (Figure 28). In the Project Explorer, expand the **Debug** folder. The generated application image binary file is under the name format **<app_project_name>.bin.signed**. From this location, it can be directly copied to a known location.

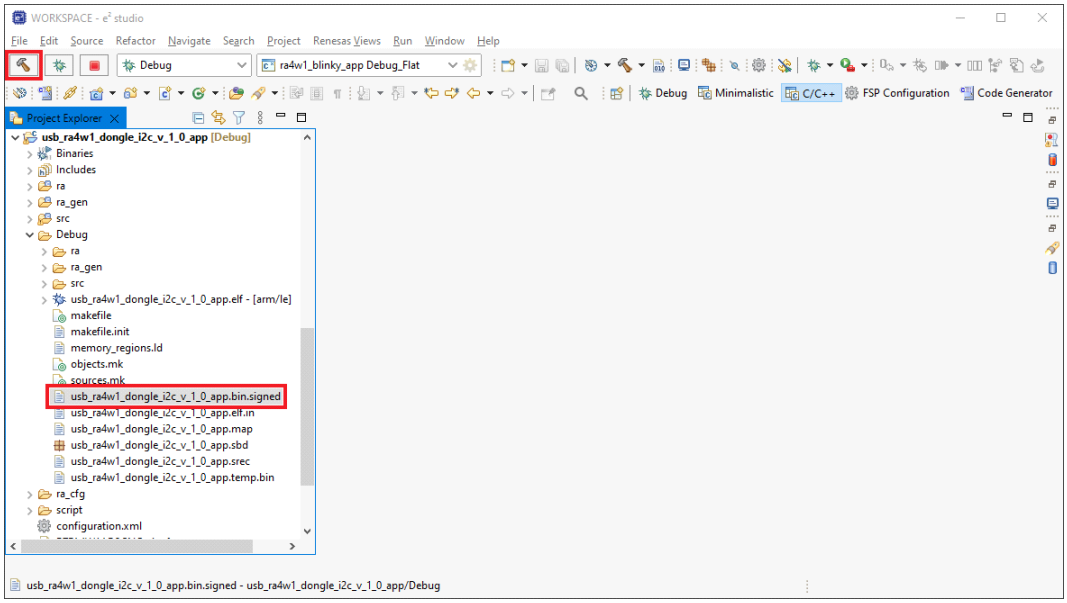

Figure 28. Generating the Application Image

For more information about the image generation process, refer to the Renesas *RA Family Secure Bootloader for RA2 MCU Series Application Note*.

5. Communication Protocol

The following section describes the communication protocol supported by the firmware of the ISO-DONGLE-EV1Z Rev. D (ra4e1_iso_dongle_ev1z_rev_d_fw_v_2_1_app.bin.signed version 2.1 or later). The communication protocol runs across a serial port and allows sending direct commands to the BFE and receiving responses, but it also reconfigures the dongle peripherals. This communication protocol is used by the GUI software. It can also be used to develop custom applications that communicate with the BFE using the isolated dongle. Each transmission follows the same sequence: the application software on the PC sends a command and the dongle sends back a response. The length of the command and response packets vary, and the packet contains obligatory elements like begin indicator (0x0A), command code, end indicator (0x0D), error code, and others. The maximum transmission length is limited to 64B and the maximum payload inside a response is 32B. Longer responses are divided into several response packets but by no more than eight. Every incorrect or unrecognized command results in an error response (Table 3). Table 4 provides more information about the reason for the errors and facilitates the debugging.

Table 3. Description of a Command Error

Command Packet Structure									
Byte	0	1	2	3	4	5	6	7	8
Filed Name	-	-	-	-	-	-	-	-	-
Value Range	Any unrecognized command or incorrect packet!								
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x02, 0x03	\r [0x0D]	-	-	-	-	-	-

Table 4. List of Error Codes

Error Code	Error Name	Description
0x00	Success (no error)	No error was detected during execution
0x01	Communication	Error in communication with the BMIC
0x02	Command	The command is incorrect.
0x03	Configuration	Error in configuration
0x04	Pins	Error in GPIO and special pins control

The following sections describe all supported commands in detail. For practical examples how to use the commands, refer to the Examples section.

5.1 Connect Command

The connect command tests the communication between the application software on the PC and the ISO-DONGLE-EV1Z Rev.D dongle. It does not have any effect on the BFE. This is the command that must always be sent first to the dongle after power on and before any other. Without it you will not be able to communicate with the BFE. List of ALERT Pin States shows the content of the connect command and the expected response.

Table 5. Description of the Connect Command

Command Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Command	End	-	-	-	-	-	-
Value Range	\n [0x0A]	c [0x63]	\r [0x0D]	-	-	-	-	-	-
Response Packet Structure									
Byte	0	1	2	3	4	5	6	7	8
Filed Name	Begin	-	-	-	-	Version number (might differ)			End
Value Range	\n [0x0A]	B [0x49]	M [0x53]	S [0x4F]	v [0x76]	2 [0x32]	. [0x2E]	1 [0x31]	\r [0x0D]

5.2 Read/Write Commands

The communication protocol supports three different types of read/write commands:

- I^2C commands (Table 6 and Table 7)
- SPI commands without CRC (Table 9 and Table 10)
- SPI commands with CRC (Table 11 and Table 12)

All these commands share the same structure with small differences. The read commands include a slave address, register address, and byte length fields.

Important: The LSB of the slave address in a read command must always be set to 1 (which is 0x34 | 0x01 for I^2C) according to the device specifics.

When the target register is longer than 1B, this must be considered in the requested Length byte. This protocol also supports reading multiple consecutive registers. The number of the register is controlled using the Length byte, and the first register is the one indicated by the Register Address byte. When using SPI with CRC, there are two additional bytes before the packet-end indicator that contains the checksum (Table 11). It is not automatically calculated by the dongle and the PC application must take care. The response to the read command contains Error Code, ALERT pin state, and packet number. The ALERT pin state byte indicates the logic state of the ALERT pin (Table 7). When the Length byte exceeds 32B, the payload is divided into multiple response packets. The number of the register is denoted by the packet byte number. When using SPI with CRC, the last response packet contains two additional bytes with the total checksum (Table 11).

The write commands include a slave address, register address, byte length, and (data) Byte 1. These commands support writing of only one byte at a time. Therefore, the Length byte must always be set to 0x01. If the target register is longer, each byte must be addressed in a separate transmission.

Important: The LSB of the slave address in a write command must always be set to 0 (which is 0x34 for I^2C) according to the device specifics.

When using SPI with CRC, the write command contains two more bytes before the end indicator with the checksum. The response of a write command always contains the Error code and ALERT pin status.

Table 6. Description of the I²C Register Read Command

Command Packet Structure									
Byte #	0	1	2	3	4	5	-	-	-
Filed Name	Begin	Command	Slave Address	Register Address	Length	End	-	-	-
Value Range	\n [0x0A]	0x06	0x00… 0xFF	0x00… 0xFF	0x01... 0xFF	\r [0x0D]	-	-	-
Response Packet Structure									
Byte #	0	1	2	3	4	5	…	(Length + 3) or 35	(Length + 4) or 36
Filed Name	Begin	Error Code	ALERT pin state	Packet #	Byte 1	Byte 2	…	Byte (Length)	End
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	0x00… 0x07	0x00… 0xFF	0x00… 0xFF	…	0x00… 0xFF	\r [0x0D]

Table 7. List of ALERT Pin States

Value	Pin State	Description
0x00	De-asserted	The ALERT pin is de-asserted (HIGH level)
0x01	Asserted	The ALERT pin is asserted (LOW level)

Table 8. Description of the I²C Register Write Command

Command Packet Structure									
Byte #	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Command	Slave Address	Register Address	Length	Byte 1	End	-	-
Value Range	\n [0x0A]	0x07	0x00… 0xFF	0x00… 0xFF	0x01	0x00… 0xFF	\r [0x0D]	-	-
Response Packet Structure									
Byte #	0	1	2	3	-	-	-	-	-
Filed Name	Begin	Error Code	ALERT pin state	End	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-	-	-	-

Table 9. Description of the SPI without CRC Register Read Command

Command Packet Structure									
Byte #	0	1	2	3	4	5	-	-	-
Filed Name	Begin	Command	Slave Address	Register Address	Length	End	-	-	-
Value Range	\n [0x0A]	0x08	0x00… 0xFF	0x00… 0xFF	0x01... 0xFF	\r [0x0D]	-	-	-
Response Packet Structure									
Byte #	0	1	2	3	4	5	…	(Length + 3) or 35	(Length + 4) or 36
Filed Name	Begin	Error Code	ALERT pin state	Packet #	Byte 1	Byte 2	…	Byte (Length)	End
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	0x00… 0x07	0x00… 0xFF	0x00… 0xFF	…	0x00… 0xFF	\r [0x0D]

Table 10. Description of the SPI without CRC Register Write Command

Command Packet Structure									
Byte #	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Command	Slave Address	Register Address	Length	Byte 1	End	-	-
Value Range	\n [0x0A]	0x09	0x00… 0xFF	0x00… 0xFF	0x01	0x00… 0xFF	\r [0x0D]	-	-
Response Packet Structure									
Byte #	0	1	2	3	-	-	-	-	-
Filed Name	Begin	Error Code	ALERT pin state	End	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-	-	-	-

Table 11. Description of the SPI with CRC Register Read Command

Command Packet Structure											
Byte #	0	1	2	3	4	5	6	7	-	-	-
Filed Name	Begin	Command	Slave Address	Register Address	Length	CRC MSB	CRC LSB	End	-	-	-
Value Range	\n [0x0A]	0x11	0x00… 0xFF	0x00… 0xFF	0x01... 0xFF	0x00… 0xFF	0x00… 0xFF	\r [0x0D]	-	-	-
Response Packet Structure											
Byte #	0	1	2	3	4	5	…	35	36	-	-
Filed Name	Begin	Error Code	ALERT pin state	Packet #	Byte 1	Byte 2	…	Byte (Length)	End	-	-
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	0x00… 0x07	0x00… 0xFF	0x00… 0xFF	…	0x00… 0xFF	\r [0x0D]	-	-
Byte #	0	1	2	3	4	5	…	(Length+ 3) or 35	(Length+ 4) or 36	(Length+ 5) or 37	(Length+ 6) or 38
Filed Name	Begin	Error Code	ALERT pin state	Packet #	Byte 1	Byte 2	…	Byte (Length)	CRC MSB	CRC LSB	End
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	0x00… 0x07	0x00… 0xFF	0x00… 0xFF	…	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	\r [0x0D]

Table 12. Description of the SPI with CRC Register Write Command

Command Packet Structure									
Byte	0	1	2	3	4	5	6	7	8
Filed Name	Begin	Command	Slave Address	Register Address	Length	Byte 1	CRC MSB	CRC LSB	End
Value Range	\n [0x0A]	0x12	0x00… 0xFF	0x00… 0xFF	0x01	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	\r [0x0D]
Response Packet Structure									
Byte	0	1	2	3	-	-	-	-	-
Filed Name	Begin	Error Code	ALERT pin state	End	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-	-	-	-

Note: Switching from SPI to I^{2}C and vice versa requires sending a configuration command to the dongle, which is described in the following section.

The drivers for the SPI and I^{2}C communication interfaces of the ISO-DONGLE-EV1Z Rev.D dongle are configured to comply with the inter-byte timing requirements of the ISL94216A and RAA489206. During a read, there is more than 7µs of delay between the data bytes (see Figure 29 and Figure 31. On other hand the write commands does not have any additional delay (Figure 30 and Figure 32).

Figure 29. Inter-byte Timing of the SPI Read Command

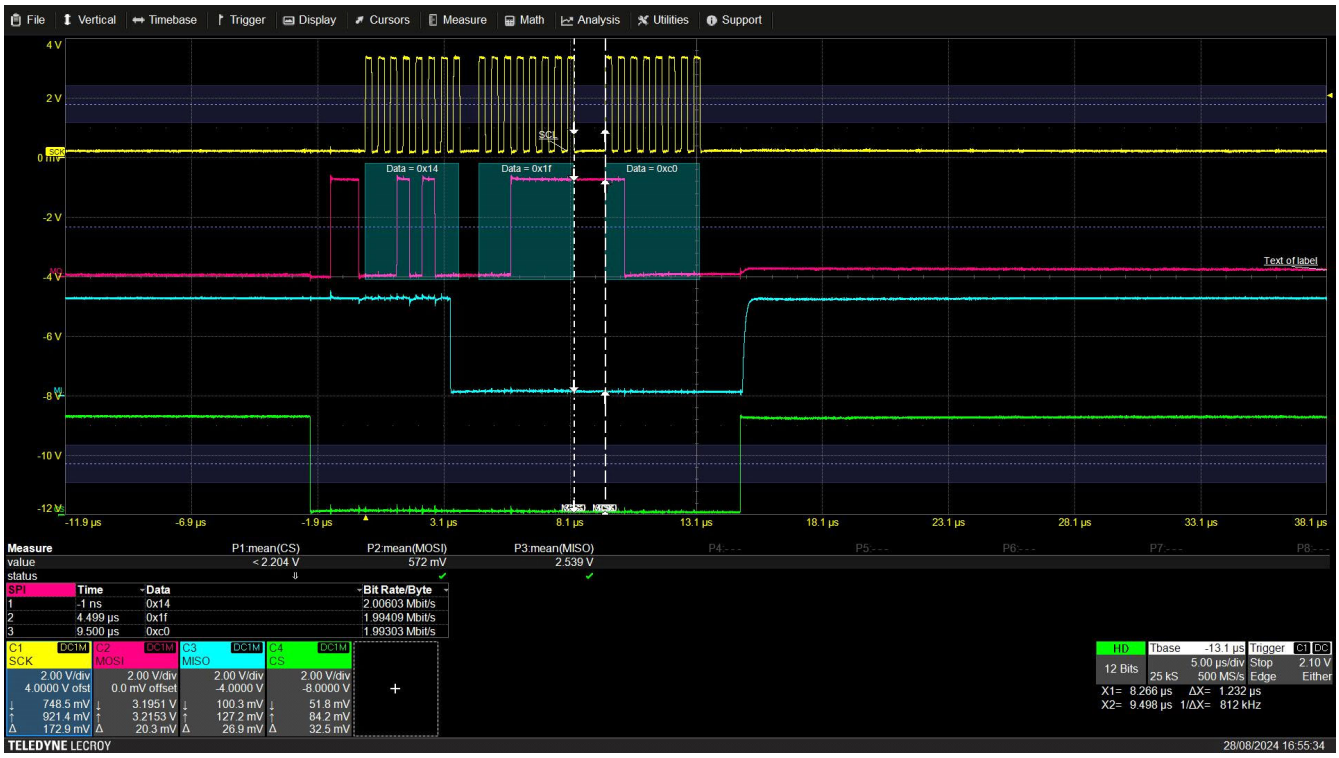

Figure 30. Inter-Byte Timing of the SPI Write Command

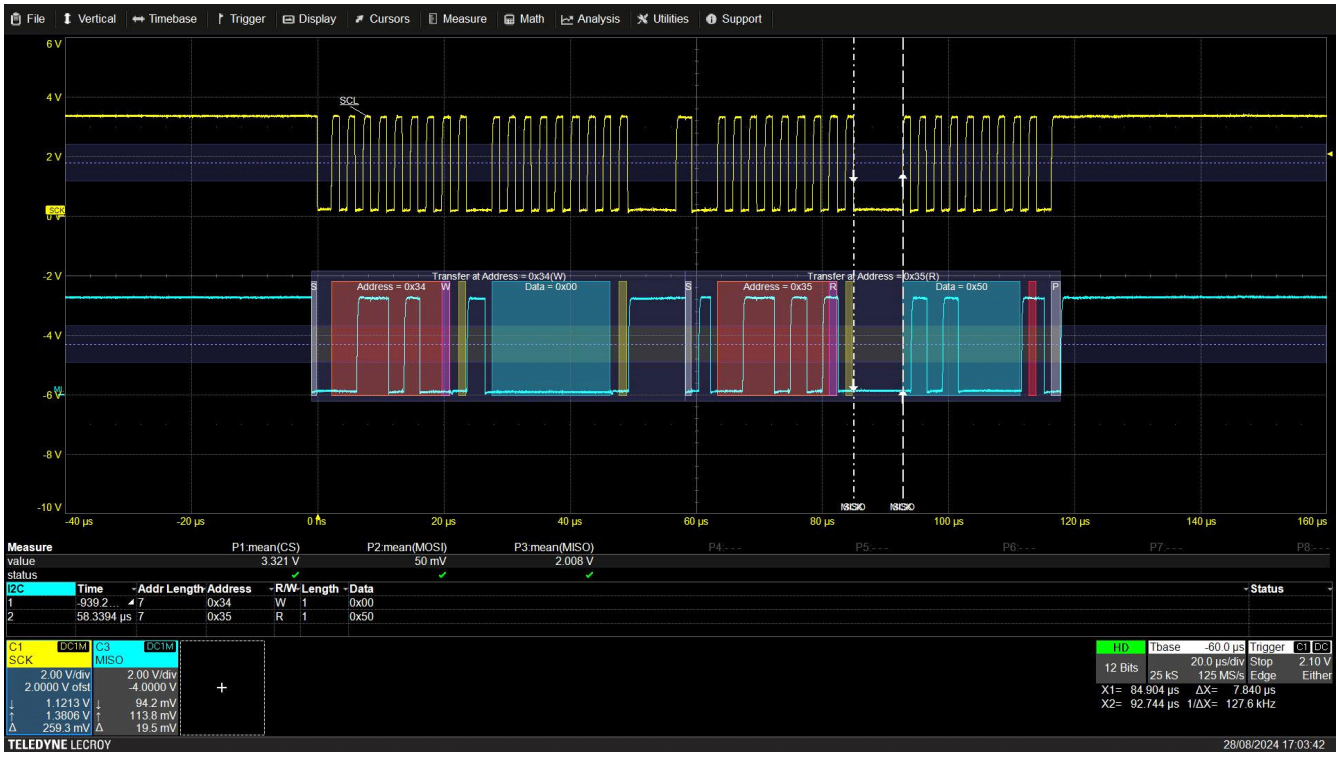

Figure 31. Inter-Byte Timing of the I²C Read Command

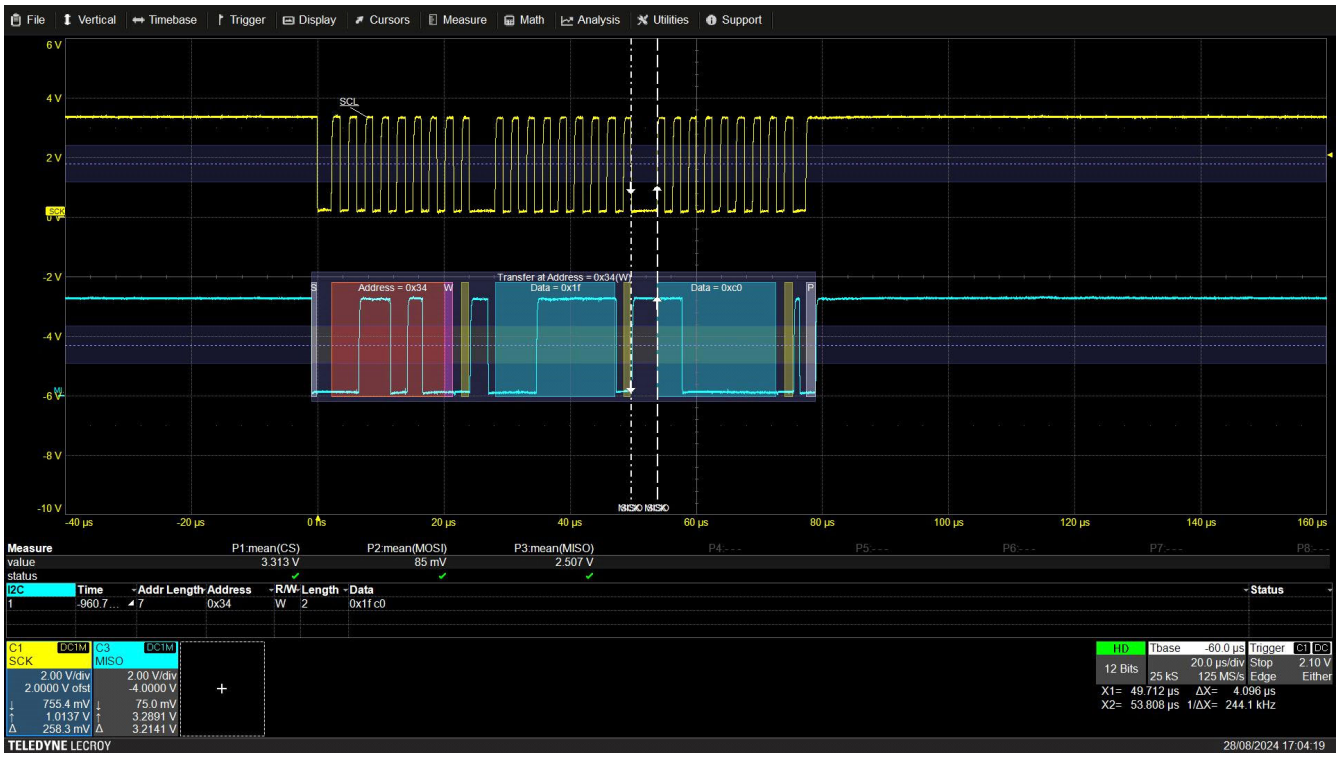

Figure 32. Inter-Byte Timing of the I²C Write Command

5.3 Configuration Commands

The configuration commands affect directly the communications and indirectly the BFE. They can change the communication interface, data rate or any pin state, or read back the state.

The serial communication interface can be changed using the Set Serial Interface Command (see Table 13). The options for the Serial Interface byte are provided in Table 15. The SPI option works both for commands with or without CRC. Sending a write/read command for the unselected serial protocol returns a non-zero error code. The Read Serial Interface Command returns the currently selected interface (see Table 14).

Table 13. Description of the Set Serial Interface Command

Command Packet Structure									
Byte	0	1	2	3	-	-	-	-	-
Filed Name	Begin	Command	Serial Interface	End	-	-	-	-	-
Value Range	\n [0x0A]	P [0x50]	0x01, 0x02	\r [0x0D]	-	-	-	-	-
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x02, 0x03	\r [0x0D]	-	-	-	-	-	-

Table 14. Description of the Read Serial Interface Command

Command Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Command	End	-	-	-	-	-	-
Value Range	\n [0x0A]	p [0x70]	\r [0x0D]	-	-	-	-	-	-
Response Packet Structure									
Byte	0	1	2	3	-	-	-	-	-
Filed Name	Begin	Error Code	Serial Protocol	End	-	-	-	-	-
Value Range	\n [0x0A]	0x00	0x00, 0x01, 0x02	\r [0x0D]	-	-	-	-	-

Table 15. List of Serial Interface Options

Value	Direction Option	Description
0x01	SPI	SPI communication Interface (with or without CRC)
0x02	I^2C	I^2C communication Interface

Table 16 and Table 17 describe the commands for changing the I^2C and SPI data rates. The supported data rates for I^2C can be 100kbps or 400kbps, and for SPI, they can vary between 100kbps and 2.5Mbps. The heximal equivalent is written in the Data Rate Byte fields, which is 100kbps for 0x00 0x01 0x86 0xA0.

Table 16. Description of the Set I2C Data Rate Command

Command Packet Structure									
Byte	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Command	Data Rate Byte 3	Data Rate Byte 1	Data Rate Byte 2	Data Rate Byte 0	End	-	-
Value Range	\n [0x0A]	f [0x66]	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	\r [0x0D]	-	-
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x03	\r [0x0D]	-	-	-	-	-	-

Table 17. Description of the Set SPI Data Rate Command

Command Packet Structure									
Byte	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Command	Data Rate Byte 3	Data Rate Byte 1	Data Rate Byte 2	Data Rate Byte 0	End	-	-
Value Range	\n [0x0A]	F [0x46]	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	0x00… 0xFF	\r [0x0D]	-	-
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x03	\r [0x0D]	-	-	-	-	-	-

The communications dongle has four GPIO pins that can be configured as digital inputs or open-drain outputs (pull-up resistors are available on the board) using the Set MCU GPIO Direction Command (see Table 18). The pin options can be selected from Table 19. Each pin can be individually configured.

Table 18. Description of the Set MCU GPIO Pins Direction Command

Command Packet Structure											
Byte	0	1	2	3	4	5	6	7	8	9	10
Filed Name	Begin	Command	GPIO3 Option	GPIO2 Option	GPIO1 Option	GPIO0 Option	End	-	-	-	-
Value Range	\n [0x0A]	D [0x44]	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-	-	-
Response Packet Structure											
Byte	0	1	2	3	4	5	6	7	8	9	10
Filed Name	Begin	Error Code	End	-	-	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x03	\r [0x0D]	-	-	-	-	-	-	-	-

Table 19. List of GPIO Direction Options

Value	Direction Option	Description
0x00	Input	The GPIO pin is an input.
0x01	Open Drain Output	The GPIO pin is an open-drain output.

The GPIO states can be read using the Read MCU GPIO Pins Command (see Table 20). This option works both for the input and output configuration.

Table 20. Description of the Read MCU GPIO Pins Command

Command Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Command	End	-	-	-	-	-	-
Value Range	\n [0x0A]	g [0x67]	\r [0x0D]	-	-	-	-	-	-
Response Packet Structure									
Byte	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Error Code	GPIO3 Level	GPIO2 Level	GPIO1 Level	GPIO0 Level	End	-	-
Value Range	\n [0x0A]	0x00	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-

Table 21. List of MCU Pin Levels

Value	MCU Pin Levels	Description
0x00	Low	The MCU pin level is high.
0x01	High	The MCU pin level is low.

The Write MCU GPIO Pins Command is used for changing the level of the open-drain outputs. When the pins are configured as an input, the setting is ignored.

Table 22. Description of the Write MCU GPIO Pins Command

Command Packet Structure									
Byte	0	1	2	3	4	5	6	-	-
Filed Name	Begin	Command	GPIO3 Level	GPIO2 Level	GPIO1 Level	GPIO0 Level	End	-	-
Value Range	\n [0x0A]	G [0x47]	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	\r [0x0D]	-	-
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x04	\r [0x0D]	-	-	-	-	-	-

Similarly, the GPIO pins, the special-purpose pins of the isolated dongle, are controlled using the Read and Write MCU Special Purpose Pins Commands (see Table 23 and Table 24).

Table 23. Description of the Read MCU Special Purpose Pins Command

Command Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Command	End	-	-	-	-	-	-
Value Range	\n [0x0A]	s [0x73]	\r [0x0D]	-	-	-	-	-	-
Response Packet Structure									
Byte	0	1	2	3	4	5	6	7	8
Filed Name	Begin	Error Code	RESET Level	WAKEUP Level	ADDR Level	CMS1 Level	CMS0 Level	ALERT Level	End
Value Range	\n [0x0A]	0x00	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	\r [0x0D]

Table 24. Description of the Write MCU Special Purpose Pins Command

Command Packet Structure									
Byte	0	1	2	3	4	5	6	7	8
Filed Name	Begin	Command	RESET Level	WAKEUP Level	ADDR Level	CMS1 Level	CMS0 Level	ALERT Level	End
Value Range	\n [0x0A]	s [0x53]	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	0x00, 0x01	\r [0x0D]
Response Packet Structure									
Byte	0	1	2	-	-	-	-	-	-
Filed Name	Begin	Error Code	End	-	-	-	-	-	-
Value Range	\n [0x0A]	0x00, 0x04	\r [0x0D]	-	-	-	-	-	-

5.4 Examples

The following examples demonstrate how to use the commands described in the previous section.

Table 25. Examples of the Communication Protocol Commands

#	Sent	Received	Description
1	0x0A 0x63 0x0D	0A 49 53 4F 76 32 2E 30 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Test Connection
2	0x0A 0x06 0x35 0x00 0x01 0x0D	0A 00 00 01 F2 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	I²C Read (Slave Address: 0x34 | 0x01): 0x00 Product ID Register
3	0x0A 0x06 0x35 0x63 0x07 0x0D	0A 00 00 01 00 00 00 00 08 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	I²C Read (Slave Address: 0x34 | 0x01): 0x63 – 0x69 Faults and Status Registers
4	0x0A 0x07 0x34 0x1F 0x01 0xC0 0x0D	0A 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	I²C Write (Slave Address: 0x34): 0x1F Vbat1 Operation Register
5	0x0A 0x08 0x15 0x00 0x01 0x0D	0A 00 00 01 F2 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SPI Read w/o CRC (Address: 0x14 | 0x01): 0x00 Product ID Register
6	0x0A 0x08 0x15 0x63 0x07 0x0D	0A 00 00 01 00 00 00 00 08 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SPI Read w/o CRC (Address: 0x14 | 0x01): 0x63 – 0x69 Faults and Status
7	0x0A 0x11 0x9D 0x9F 0x07 0xCA 0x1C 0x0D	0A 00 00 01 00 00 00 00 08 00 00 4A B2 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SPI Read with CRC (Address: 0x9C | 0x01): 0x9F Faults Command
8	0x0A 0x09 0x14 0x1F 0x01 0xC0 0x0D	0A 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SPI Write w/o CRC (Address: 0x14): 0x1F Vbat1 Operation Register
9	0x0A 0x12 0x9C 0x02 0x01 0x81 0x4C 0x27 0x0D	0A 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SPI Write with CRC (Address: 0x9C): 0x02 Vcell Operation Register
10	0x0A 0x70 0x0D	0A 00 01 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Read the serial protocol
11	0x0A 0x50 0x02 0x0D	0A 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Set I²C as a serial protocol
12	0x0A 0x66 0x00 0x01 0x86 0xA0 0x0D	0A 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Change the I²C data rate to 100 kbps
13	0x0A 0x46 0x00 0x0F 0x42 0x40 0x0D	0A 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Change the SPI data rate to 1 Mbps
14	0x0A 0x44 0x01 0x01 0x01 0x01 0x0D	0A 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Set all GPIO pins of the MCU as outputs
15	0x0A 0x67 0x0D	0A 00 01 00 01 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Read the GPIO pins of the MCU
16	0x0A 0x73 0x0D	0A 00 01 00 01 00 01 01 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Read MCU Special Purpose Pins
17	0x0A 0x53 0x00 0x01 0x01 0x00 0x01 0x0D	0A 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Write MCU Special Purpose Pins: Select SPI BFE configuration and Reset BFE

6. Revision History

Revision	Date	Description
1.01	Jul 24, 2025	Corrected board number throughout.
1.00	May 22, 2025	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

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

Contact Information

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

Trademarks

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