RX65N Group

AWS FreeRTOS 2nd MCU OTA Firmware Update Design Guide

Introduction

Many embedded systems that utilize the cloud service have the configuration shown in Figure 1.1, consisting of a primary MCU (1st MCU) that connects directly to internet and provides functionality for communicating with Amazon Web Services™ (AWS), and a secondary MCU (2nd MCU) that is connected to the 1st MCU via a local data transmission channel (such as a UART or BLE).

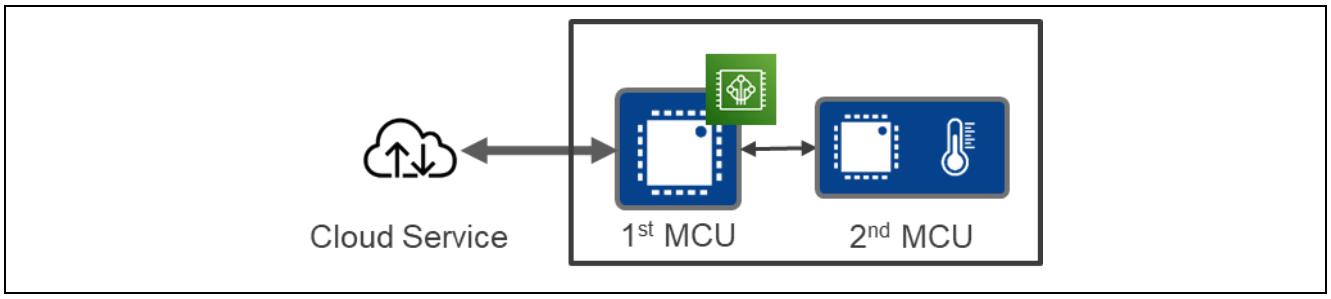


Figure 1.1 Conceptual Diagram of Hypothetical Embedded System Utilizing the Cloud Service

This application note describes a method for implementing OTA updating of the 2nd MCU's firmware via the 1st MCU (2nd MCU OTA update). By reading this application note you can learn the following.

- How to implement 2nd MCU OTA update using OTA Library in an environment comprising AWS and FreeRTOS™
- Example of the necessary software implementation for 2nd MCU OTA update on the device side
- How to switch between OTA update execution for the 1st MCU and 2nd MCU on the AWS cloud side

The above knowledge will enable you to make use of the OTA update mechanism provided by the AWS IoT Device Management service to implement 2nd MCU OTA update.

Refer to the link below for a 2nd MCU OTA update demo based on this design guide.

RX65N Group Sample Code for OTA Update of Secondary Device with Amazon Web Services Using FreeRTOS

Also, refer to the application note at the link below for basic information on the firmware update design policy for Renesas MCUs.

Renesas MCU Firmware Update Design Policy

Target Devices

1st MCU: RX65N Group

2nd MCU: RX23W Group

Operation of the code and procedures described in this application note and the 2nd MCU OTA update demo application note have been confirmed on the MCUs listed above. When applying the contents of this application note to other MCUs, you will need to make changes to match the specifications of the target MCU and to perform a thorough evaluation.

Related Documents

This application note refers to and explains the following documents. The chapter structure may change when documents are updated. Please keep this in mind when referencing these documents.

Document Title	Document No.
RX Family How to implement FreeRTOS OTA by using Amazon Web Services on RX65N	R01AN5549EJ0102
RX65N Group Sample Code for OTA Update of Secondary Device with Amazon Web Services Using FreeRTOS	R01AN6220EJ0110
Renesas MCU Firmware Update Design Policy	R01AN5548EJ0100
RX Family Firmware Update Module Using Firmware Integration Technology	R01AN5824EJ0103

Additional Links

- Secondary device OTA demonstration video
  Secondary Device OTA Update using FreeRTOS and Amazon Web Services
- RX Family Cloud Connectivity Solutions webpage
  https://www.renesas.com/rx-cloud

Contents

1. Overview

1.1 AWS IoT Services and 2nd MCU OTA Update

The following post on the AWS blog describes a method of implementing 2nd MCU OTA update by modifying a portion of the existing firmware.

Implementing OTA Update of a Secondary Processor Using FreeRTOS | Amazon Web Services Blog

In addition, the user guide to FreeRTOS issued by AWS and linked to below describes the use of the `fileType` parameter, which can be specified when creating an OTA update, as a way to apply an OTA update to a 2nd MCU, as follows:

> *Under File Type, enter an integer value in the range 0-255. The file type you enter will be added to the Job document that is delivered to the MCU. The MCU firmware/software developer has full ownership on what to do with this value. Possible scenarios include an MCU that has a secondary processor whose firmware can be updated independently from the primary processor. When the device receives an OTA update job, it can use the File Type to identify which processor the update is for.*

Creating an OTA update (AWS IoT console) - FreeRTOS (amazon.com)

Using this information as a basis, this application note summarizes the necessary steps to implement 2nd MCU OTA update on RX Family MCUs.

1.2 OTA Update Mechanism Provided by AWS IoT Services

AWS IoT services implement OTA update functionality using AWS IoT Jobs in the cloud and the AWS IoT Over-the-air Update Library (OTA Library) on a device running FreeRTOS.

AWS IoT Jobs provides functionality that enables remote execution of jobs on devices managed by AWS IoT services. A job is an operation defined by the user beforehand. The user defines the specifications of the job in a Job document. Job documents are created in JSON format and can be freely designed by the user.

When a job is created for a device, a message including the Job document is published to the AWS IoT Jobs topic for the device in the MQTT protocol. When the device is subscribed to its AWS IoT Jobs topic and receives the message, it parses the Job document contained in the message and executes the processing described. It is therefore necessary for the user to implement processing for parsing and executing the contents of the Job document beforehand.

Thus, in order to actualize the AWS IoT Jobs functionality, the user must define the Job document and implement on the device the processing for parsing and executing the contents of the Job document.

The OTA update provided by AWS IoT services includes Job documents for OTA update jobs predefined by AWS, and the OTA Library includes processing for parsing and executing Job documents, which must be implemented on the device. This provides a mechanism whereby the user can easily implement OTA update functionality.

1.3 OTA Library

The OTA Library is an OSS project developed by AWS on GitHub. It is publicly available under the MIT License.

aws/ota-for-aws-iot-embedded-sdk (github.com)

The following demo application is available in Version 202107.00 of FreeRTOS AWS Reference Integrations, which are FreeRTOS reference implementations created by AWS and utilizes OTA Library v3.0.0.

Over-the-air updates demo application - FreeRTOS (amazon.com)

When a message is published to the device's AWS IoT Jobs topic, the OTA Library parses the Job document and starts OTA update processing. The OTA update processing includes processing that performs operations on the hardware, such as self-programming of data to specific areas in ROM and execution of a software reset to update to the new firmware. To maintain portability of the library while including processing such as this, the OTA Library utilizes an API interface called the OTA Platform Abstraction Layer (OTA PAL). The OTA PAL allows abstraction of the hardware from the viewpoint of the library.

1.4 Extension to 2nd MCU OTA Update

To implement MCU OTA update functionality using AWS IoT Jobs and the OTA Library as the OTA update mechanism, it is necessary to distinguish the 1st MCU OTA update and 2nd MCU OTA update so that it is possible to switch between them.

First, the `fileType` parameter defined in the Job document of the OTA update job provided by AWS is used to distinguish the target of the OTA update. By default, a 1st MCU OTA update is executed when the OTA Library's `fileType` value is 0. Therefore, the OTA update target can be distinguished by specifying a value other than 0 (for example, 1).

Next, the OTA PAL is used for switching the OTA processing. Since the OTA PAL abstracts the hardware processing required for a firmware update, it is possible to switch the update target by preparing a set of functions composing an OTA PAL for the 2nd MCU OTA update (OTA PAL2) and then using the `fileType` value to switch the OTA PAL functions that are called.

1.5 Confirmed Operation Environment

The operation of the 2nd MCU OTA update functionality has been confirmed under the conditions listed below. Note that the description that follows assumes an environment conforming to these conditions.

Table 1.1 2nd MCU OTA Update Confirmed Operation Environment

Item	Description
1st MCU	RX65N
1st MCU board	Renesas Starter Kit+ for RX65N-2MB (RSK+RX65N-2MB board)
RTOS	FreeRTOS AWS Reference Integrations Version 202107.00
2nd MCU	RX23W
2nd MCU board	Target Board for RX23W (TB-RX23W board)
FWUP FIT module	V1.02
IDE	e2 studio 2022-01
Toolchain	CC-RX V3.04.00
Firmware concatenation tool	Renesas Secure Flash Programmer 1.01 Note: Make sure to use version 1.x.x. Version 2.x.x does not support Amazon FreeRTOS projects.
Firmware programming tool	Renesas Flash Programmer V3.09.00

2. System Configuration

The system configuration on which the 2nd MCU OTA update runs is described below.

2.1 Hardware Configuration

A schematic diagram of the hardware configuration is shown below.

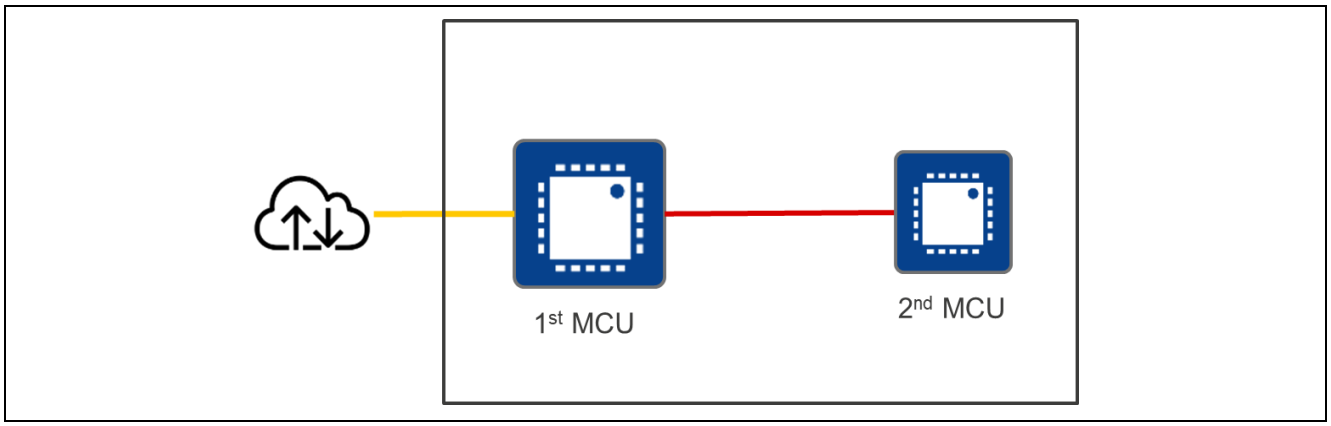


Figure 2.1 Schematic Diagram of Hardware Configuration

First, since the 1st MCU connects directly to the internet, it requires an appropriate interface, such as Ethernet or Wi-Fi (yellow line in Figure 2.1).

In this application note, Renesas Starter Kit+ for RX65N-2MB (RSK+RX65N-2MB board), which is an AWS qualified device up to and including the OTA update function, is used. It connects to the internet via Ethernet.

Renesas Starter Kit+ for RX65N-2MB | AWS Qualified (amazonaws.com)

Next, a data transmission channel to allow exchanging of data between the 1st MCU and the 2nd MCU is required (red line in Figure 2.1). In this application note a UART is used to implement this data transfer channel.

2.2 Software Configuration

The software configuration of the 1st MCU and 2nd MCU is shown below.

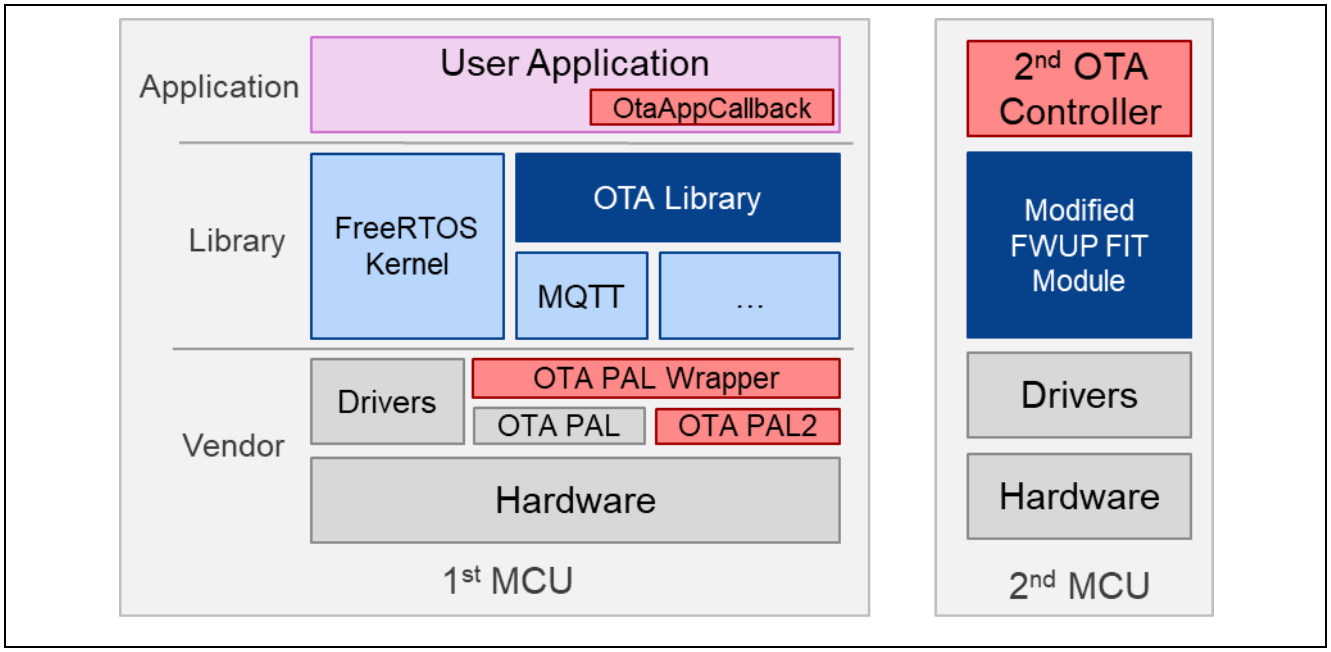


Figure 2.2 Software Configuration

2.2.1 1st MCU Software Configuration

This is created using FreeRTOS AWS Reference Integrations Version 202107.00 as the basis. In Figure 2.2, Software Configuration, *the portions in the red boxes are newly created or require additional code*.

The OTA update demo application `RunOtaCoreMqttDemo` is used as the User Application. OtaAppCallback is a callback function registered when the OTA Library is initialized. Processing that is executed when the update completion event occurs (*`OtaJobEventUpdateComplete`*) is added to this callback function.

OTA PAL2 implements processing to send commands to the 2nd MCU. OTA PAL Wrapper implements processing to switch between OTA PAL and OTA PAL2 based on the value of `fileType`.

2.2.2 2nd MCU Software Configuration

The 2nd MCU software comprises two projects: the bootloader and the user program. They are created using the firmware update (FWUP) FIT module demo project as a basis.

The bootloader runs first at startup and performs tasks such as verifying the code signature of the user program and switching to the new firmware.

The user program implements the 2nd OTA Controller as an event-driven state machine to which commands are sent from the 1st MCU via an SCI channel using UART data transfer. It controls the FWUP FIT module. Firmware update processing is performed by a partially modified version of the FWUP FIT module.

2.2.3 AWS Configuration

The environment that needs to be constructed on AWS is the same as that for OTA updates using regular AWS IoT services. Refer to the following application note when constructing the AWS environment.

RX Family How to implement FreeRTOS OTA by using Amazon Web Services on RX65N

3. 2nd MCU OTA Update Mechanism

3.1 General Operation Flow

The operation of the 2nd MCU OTA update is as follows.

① The User Application initializes the OTA Library and creates an OTA agent task. From this point forward the OTA agent performs OTA update processing.
② When the OTA agent receives a job, it parses the Job document and starts OTA update processing.
③ Within the OTA update processing, the OTA agent calls the OTA PAL Wrapper function.
④ The OTA PAL Wrapper function references the `fileType` value and switches execution to the appropriate OTA PAL function.
⑤ In the case of a 2nd MCU OTA update, the OTA PAL2 function is run and the UART is used to send commands to the 2nd MCU.
⑥ When the 2nd MCU receives a command from the 1st MCU, the 2nd OTA Controller runs an API function from the FWUP FIT module.
⑦ The FWUP FIT module executes firmware update processing.

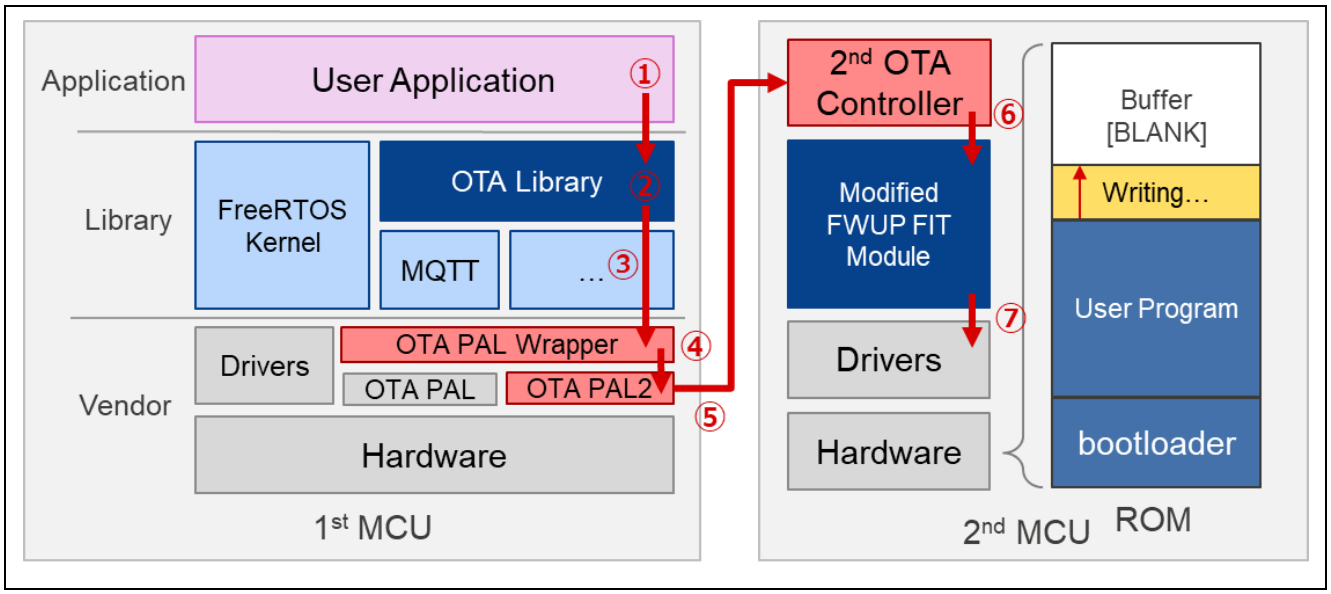


Figure 3.1 2nd MCU OTA Update Processing Flow

3.2 Relationship between `fileType` Value and OTA Agent Operation

The operation of the OTA Library differs according to whether or not the value of `fileType` matches `configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID`. The default value of `configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID` is defined as `0U`.

- When `fileType = configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID`
  The 1st MCU OTA update processing takes place. Figure 3.2 shows a general outline of the operation flow.

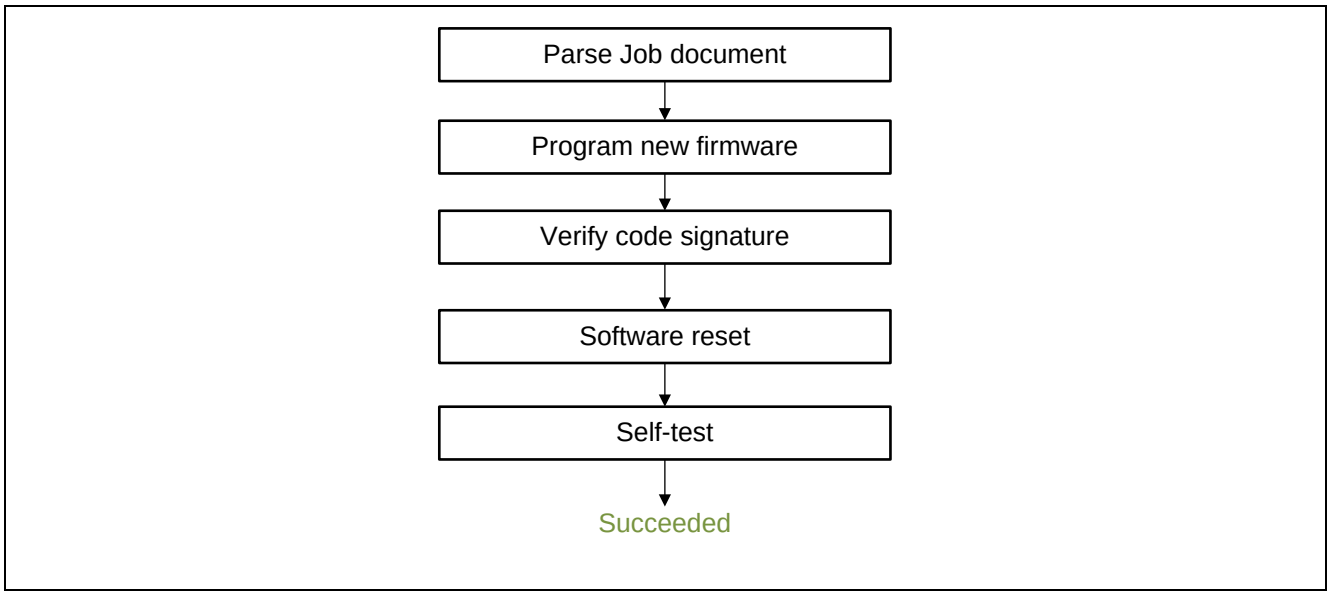


Figure 3.2 OTA Agent Operation when fileType = configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID

- When `fileType` ≠ `configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID`
  After programming of the new firmware completes and the code signature has been verified, a job SUCCEEDED/FAILED message is sent to the AWS IoT service and the job completes without the `OtaJobEventActivate` callback function being run. Then the `OtaJobEventUpdateComplete` callback function is run. Figure 3.3 shows a general outline of the operation flow.

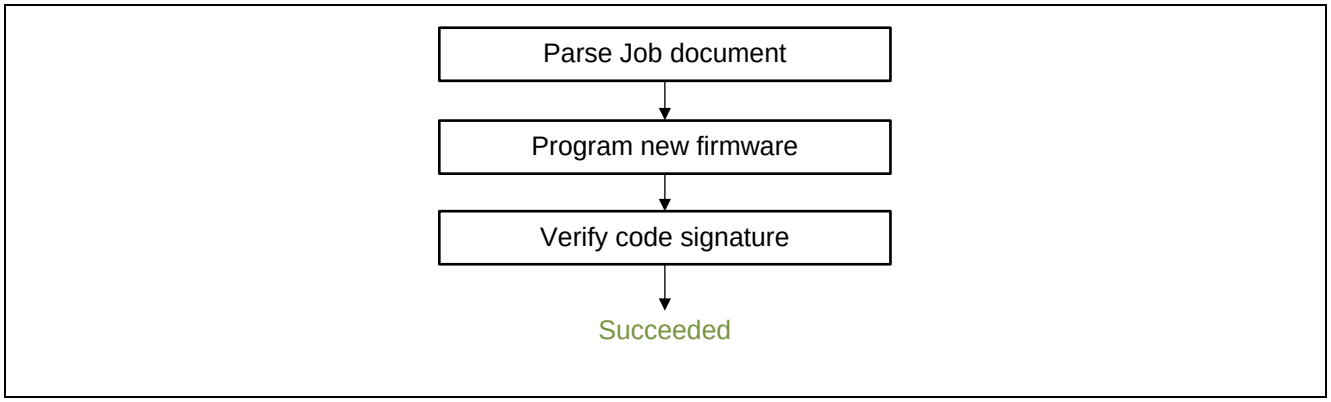


Figure 3.3 OTA Agent Operation when filetype ≠ configOTA_FIRMWARE_UPDATE_FILE_TYPE_ID

3.3 Communication between 1st MCU and 2nd MCU

The 1st MCU sends commands to the 2nd MCU when the OTA PAL2 function is called during the OTA processing. For this reason, the name of each command is the same as the name of the corresponding OTA PAL API interface. Figure 3.4 shows the communication sequence between the 1st MCU and 2nd MCU when the OTA update is proceeding normally.

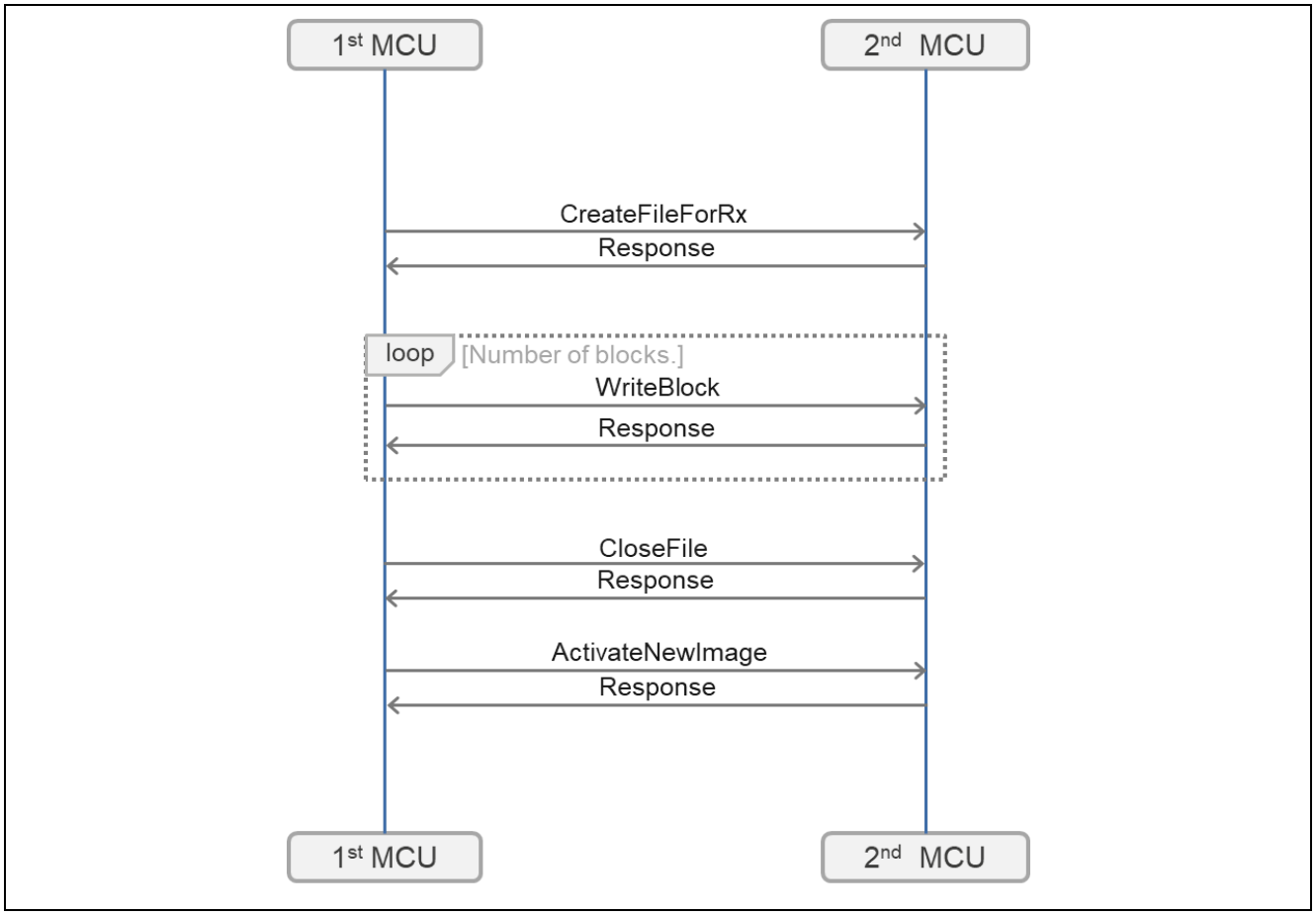


Figure 3.4 Communication Sequence between MCUs

Also, and separate from the above, GetPlatformImageState is called repeatedly during the OTA update processing.

Figure 3.5 Communication Querying Image State of 2nd MCU

In addition, if an error during the OTA update causes processing to terminate, Abort is called.

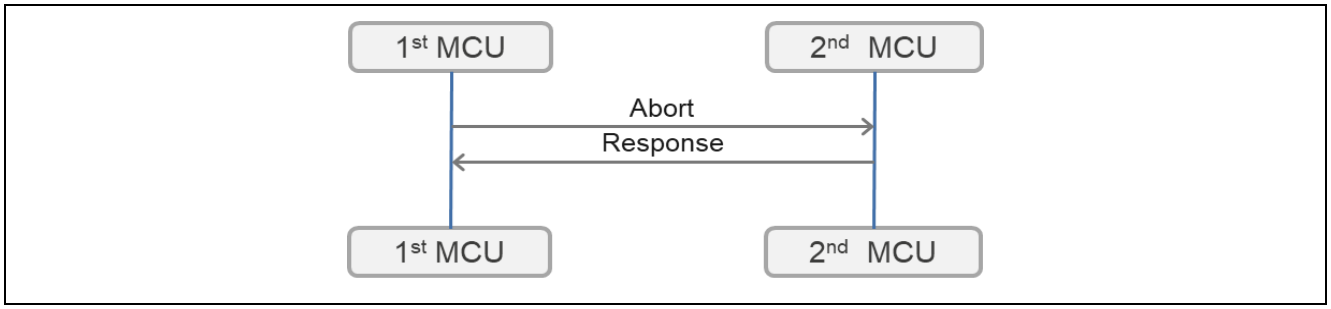


Figure 3.6 Communication when Abort Occurs

The roles of the commands used during the 2nd MCU OTA update are listed below.

Table 3.1 Description of `CreateFileForRx` Command

Command name	`CreateFileForRx`
Description	Notifies the 2nd MCU that OTA processing has started.
Response from 2nd MCU	Success or Failure

Table 3.2 Description of `WriteBlock` Command

Command name	`WriteBlock`
Description	Transfers the new firmware image sent from AWS to the 2nd MCU. A 1,024-bytes payload for the new firmware image is attached to this command.
Response from 2nd MCU	Success or Failure

Table 3.3 Description of `CloseFile` Command

Command name	`CloseFile`
Description	Directs verification of the code signature of the new firmware programmed to the 2nd MCU.
Response from 2nd MCU	Success or Failure

Table 3.4 Description of `ActivateNewImage` Command

Command name	`ActivateNewImage`
Description	Instructs the 2nd MCU to perform a software reset to update to the new firmware.
Response from 2nd MCU	Success or Failure

Table 3.5 Description of `Abort` Command

Command name	`Abort`
Description	Notifies the 2nd MCU that OTA processing has been aborted.
Response from 2nd MCU	Success or Failure

Table 3.6 Description of `GetPlatformImageState` Command

Command name	`GetPlatformImageState`
Description	Requests the firmware state of the 2nd MCU.
Response from 2nd MCU	Current firmware state

3.4 Partial Modification of FWUP FIT Module

Using the FWUP FIT module makes it possible to send the firmware for the update by UART communication via an SCI channel and execute the firmware update.

However, since the FWUP FIT module takes exclusive control of an SCI channel, the channel is not available to other applications (for transmitting sensor data, etc.).

To get around this limitation, the FWUP FIT module needs to be partially modified to allow other applications to use the data transmission channel between the 1st and 2nd MCUs.

3.5 Deploying a Data Communication Protocol for UART Communication

In order to enable two types of communication (communication for the firmware update and communication for other applications such as transmission of sensor data) on the same SCI channel, it is necessary to specify a data communication protocol. In the sample code, the protocol shown in Figure 3.7 is specified as an example. It permits transfer of both data for firmware updates and sensor data on the same channel.

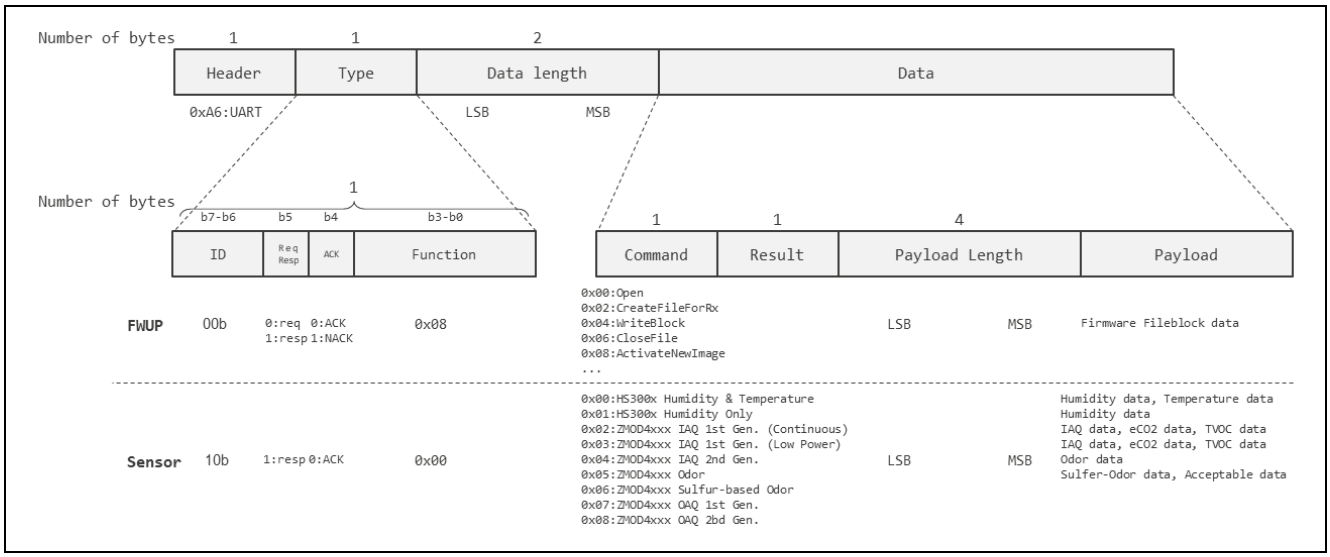


Figure 3.7 Example Data Communication Protocol

4. Example of Software Implementation on 1st MCU

Version 202107.00 of FreeRTOS AWS Reference Integrations is used as a basis. Implementation of new processing or modification of existing code to provide 2nd MCU OTA functionality is required in the following three locations.

1. vendors/renesas/boards/rx65n-rsk/ports/ota_pal_for_aws/ota_second_pal.c (newly created)
2. vendors/renesas/boards/rx65n-rsk/ports/ota_pal_for_aws/ota_pal_wrapper.c (newly created)
3. demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c (modified)

4.1 Creating New ota_second_pal.c File

Create a file named ota_second_pal.c in which the OTA PAL2 functions are defined.

The following OTA PAL2 functions implement the processing sequence:

1. Send command to the 2nd MCU.
2. Wait for a response from the 2nd MCU.
3. Check the response from the 2nd MCU and return the value to exit the function.

For an example implementation, refer to the sample code for the RSK+ RX65N-2MB board using OTA Library v3.x.x in the following application note.

RX65N Group Sample Code for OTA Update of Secondary Device with Amazon Web Services Using FreeRTOS

Here, the CreateFileForRx OTA PAL2 function `otaPal_Second_createFileForRx()` is shown as an example.

```
OtaPalStatus_t otaPal_Second_CreateFileForRx( OtaFileContext_t * const pFileContext )
{
    OtaPalStatus_t eResult     = OtaPalUninitialized;
    uint8_t   recv_pkt[ 8 ] = { 0 };

    create_send_packet( s_send_packet, FWUP_2nd_Command_LoadJob, 0, 0 );

    send_packet_to_secondary();

    if( SUCCESS_ARRIVE_PACKET == wait_arrive_packet( recv_pkt ) )
    {
        if( is_received_packet_intended_content( recv_pkt, FWUP_2nd_Command_LoadJob_Finish ) )
        {
            eResult = OtaPalSuccess;
        }
        else
        {
            eResult = OtaPalRxFileCreateFailed;
        }

        pFileContext->pFile = ( uint8_t * )pFileContext; /* Casting to uint8_t * type is valid */
        return eResult;
    }
    else
    {
        return OtaPalRxFileCreateFailed;
    }
}
```

4.1.1 OTA PAL2 Function Specifications

The processing performed by the various OTA PAL2 functions is summarized below.

Table 4.1 `otaPal_Second_CreateFileForRx` Function Specification

Function name		`otaPal_Second_CreateFileForRx`
Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalStatus_t`
	Success	`OtaPalSuccess`
	Failure	`OtaPalRxFileCreateFailed`
Note		It is necessary to replace the `pFileContext` member `pFile`, received as an argument, with a self-referencing pointer. Otherwise, an error occurs when the following line is processed. `C->pFile = (uint8_t *)C;`

Table 4.2 `otaPal_Second_WriteBlock` Function Specification

Function name		`otaPal_Second_WriteBlock`

Arguments		`OtaFileContext_t *const pFileContext,`
		`uint32_t offset,`
		`uint8_t *const pData,`
		`uint32_t blockSize`

Return values	Type	`OtaPalStatus_t`
	Success	`blockSize`
	Failure	`-2`
Note		`pData`, which is received as an argument, contains the start address in memory where the fragment of the new firmware image to be programmed is stored. Also, `blockSize`, also received as an argument, contains the file size (in bytes) of the fragment of the new firmware image to be programmed. Thus, a quantity of data equal to `blockSize` is extracted starting from `pData` and transmitted to the 2nd MCU. Upon successful completion, the file size (`blockSize`) of the fragment of the new firmware image that was programmed is returned. In case of failure, a negative integer value is returned.

Table 4.3 `otaPal_Second_CloseFile` Function Specification

Function name		`otaPal_Second_CloseFile`
Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalStatus_t`
	Success	`OtaPalSuccess`
	Failure	`OtaPalSignatureCheckFailed`

Table 4.4 `otaPal_Second_ActivateNewImage` Function Specification

Function name		`otaPal_Second_ActivateNewImage`
Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalStatus_t`
	Success	`OtaPalSuccess`
	Failure	`OtaPalActivateFailed`

Table 4.5 `otaPal_Second_Abort` Function Specification

Function name		`otaPal_Second_Abort`
Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalStatus_t`
	Failure	`OtaPalSuccess`
	Success	`OtaPalAbortFailed`
Note		Replace the `pFileContext` member `pFile`, received as an argument, with a value of `0`. `C->pFile = (uint8_t *)0;`

Table 4.6 `otaPal_Second_SetPlatformImageState` Function Specification

Arguments		`OtaFileContext_t *const pFileContext,` `OtaImageState_t eState`
Return values	Type	`OtaPalStatus_t`
	Success	`OtaPalSuccess`
	Failure	`OtaPalBadImageState`
Note		This function is not called during 2nd MCU OTA update processing, but it is provided because it is defined as part of PAL.

Table 4.7 `otaPal_Second_GetPlatformImageState` Function Specification

Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalImageState_t`
	Success	Current `ImageState`
	Failure	`OtaPalImageStateUnknown`
Note		This function always returns a value of `OtaPalImageStateValid` when 2nd MCU OTA update processing is proceeding normally.

Table 4.8 `otaPal_Second_ResetDevice` Function Specification

Arguments		`OtaFileContext_t *const pFileContext`
Return values	Type	`OtaPalStatus_t`
	Success	`OtaPalSuccess`
	Failure	`OtaPalAbortFailed`
Note		This function simply returns a value of `OtaPalSuccess` because the 1st MCU is not reset as part of 2nd MCU OTA update processing.

4.2 Creating New ota_pal_wrapper.c and ota_pal_wrapper.h Files

Create a file named ota_pal_wrapper.c in which wrapper functions for switching between the 1st MCU's own OTA PAL functions and the OTA PAL2 functions for 2nd MCU OTA update processing, based on the value of the `fileType` member of the `OtaFileContext_t` type argument passed to the OTA PAL function, are defined.

Here, the CreateFileForRx wrapper function `otaPalWrap_createFileForRx()` is shown as an example.

The OTA PAL function is run when the value of `fileType` is 0, and when the value is 1 the OTA PAL2 function is run.

```
OtaPalStatus_t otaPalWrap_CreateFileForRx(OtaFileContext_t * const pFileContext)
{
    OtaPalStatus_t eResult = OtaPalUninitialized;
    uint32_t otaTargetID = pFileContext->fileType;

    if (otaTargetID == 0)
    {
        eResult = otaPal_CreateFileForRx( pFileContext );
    }
    else if (otaTargetID == 1)
    {
        eResult = otaPal_Second_CreateFileForRx( pFileContext );
    }
    else
    {
        eResult = OtaPalRxFileCreateFailed;
    }
    return eResult;
}
```

In like manner, create wrapper functions for the other OTA PAL functions that reference `fileType` and switch to the appropriate function to be run.

Next, create a file named ota_pal_wrapper.h containing declarations for the wrapper functions defined as described above.

```
OtaPalStatus_t otaPalWrap_Abort( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_CreateFileForRx( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_CloseFile( OtaFileContext_t * const pFileContext );
int16_t otaPalWrap_WriteBlock( OtaFileContext_t * const pFileContext,
                               uint32_t ulOffset,
                               uint8_t * const pData,
                               uint32_t ulBlockSize );
OtaPalStatus_t otaPalWrap_ActivateNewImage( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_SetPlatformImageState( OtaFileContext_t * const pFileContext,
                                                 OtaImageState_t eState );
OtaPalImageState_t otaPalWrap_GetPlatformImageState( OtaFileContext_t * const pFileContext );
OtaPalStatus_t otaPalWrap_ResetDevice( OtaFileContext_t * const pFileContext );
```

4.3 Modifying ota_demo_core_mqtt.c File

Make the described modification and additions to the following three locations in the ota_demo_core_mqtt.c file.

1. Include ota_pal_wrapper.h.

```
#include "ota_pal_wrapper.h"
```

2. The OTA PAL interfaces called in the OTA Library are initialized in the `prvSetOtaInterfaces` function. Configure these settings for the OTA PAL Wrapper functions created earlier.

```
    /* Initialize the OTA library PAL Interface.*/
    pxOtaInterfaces->pal.getPlatformImageState = otaPalWrap_GetPlatformImageState;
    pxOtaInterfaces->pal.setPlatformImageState = otaPalWrap_SetPlatformImageState;
    pxOtaInterfaces->pal.writeBlock = otaPalWrap_WriteBlock;
    pxOtaInterfaces->pal.activate = otaPalWrap_ActivateNewImage;
    pxOtaInterfaces->pal.closeFile = otaPalWrap_CloseFile;
    pxOtaInterfaces->pal.reset = otaPalWrap_ResetDevice;
    pxOtaInterfaces->pal.abort = otaPalWrap_Abort;
    pxOtaInterfaces->pal.createFile = otaPalWrap_CreateFileForRx;
```

3. In the `prvOtaAppCallback` function, add the following code after case in the switch statement. `OtaJobEventUpdateComplete` is called when the OTA update job completes if `fileType` ≠ `0`. Here, the `OTA_ActivateNewImage` function runs and a software reset command is sent to update the firmware of the 2nd MCU.

```
        case OtaJobEventUpdateComplete:
            if(pData != NULL)
            {
                const OtaJobDocument_t *pJobDoc = (const OtaJobDocument_t *)pData;
                if(pJobDoc->fileTypeId == 1)
                {
                    OTA_ActivateNewImage();
                }
            }

            break;
```

5. Example of Software Implementation on 2nd MCU

The 2nd MCU programs comprise two projects: the bootloader and the user program. The FWUP FIT module demo project is used as a basis for both projects.

5.1 Creating the Bootloader Project

First, follow the steps below to create the bootloader project.

1. From the FWUP FIT module sample code, import the boot_loader project for rx231-rsk.
2. Use Smart Configurator to change the target device to the TB-RX23W board.
3. Since the boot_loader/src/src/tinycrypt folder in the project folder is empty, obtain the Tinycrypt library from the link below and add the lib folder to boot_loader/src/src/tinycrypt.
   amazon-freertos/libraries/3rdparty/tinycrypt at master · renesas/amazon-freertos (github.com)
4. In `CODE_SIGNER_PUBLIC_KEY_PEM` in the boot_loader/src/key/code_signer_public_key.h file, enter the public key information for code-signer verification. For the method of entering the public key information, refer to the following steps at the link below:
   4. Creating a key for firmware verification using OpenSSL
   5. Inserting secp256r1.publickey into the bootloader as the public key for signature verification in order to use ECDSA+SHA256 for firmware verification

   Utilizing OTA renesas/amazon-freertos Wiki (github.com)

5.2 Creating the User Program Project

Next, follow the steps below to create the user program project.

1. From the FWUP FIT module sample code, import the fwup_main project for rx231-rsk.
2. Follow the same steps used when creating the boot_loader project as described above to change the target device, add Tinycript, and enter public key information.
3. Make the following partial modifications to the FWUP FIT module source code to make it suitable for the 2nd MCU OTA update processing.
   ① Create a function to enable the FWUP FIT module to accept firmware data received via UART communication. Add the following function in the r_fwup.c file.

```
void fwup_receive_fileblock(uint8_t received_packet[])
{
    if (s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag ==
FWUP_SCI_RECEIVE_BUFFER_EMPTY)
    {
        for (uint16_t i = 0; i < 1024; ++i)
        {
            s_sci_receive_control_block.p_sci_buffer_control->buffer[i] = received_packet[8 +
i];
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size++;
        }
#if (FLASH_CFG_CODE_FLASH_BGO == 0)
                /* RTS HIGH */
        FWUP_CFG_PORT_SYMBOL.PODR.BIT.FWUP_CFG_BIT_SYMBOL = 1;  /* Set RTS to HIGH */
        FWUP_CFG_PORT_SYMBOL.PDR.BIT.FWUP_CFG_BIT_SYMBOL = 1;
        FWUP_CFG_PORT_SYMBOL.PMR.BIT.FWUP_CFG_BIT_SYMBOL = 0;   /* Change to general I/O port
*/
#endif /* FLASH_CFG_CODE_FLASH_BGO == 0 */
        s_sci_receive_control_block.total_byte_size +=
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size;
        s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size = 0;
        s_sci_receive_control_block.p_sci_buffer_control->buffer_full_flag =
FWUP_SCI_RECEIVE_BUFFER_FULL;

        if (FWUP_SCI_CONTROL_BLOCK_A == s_sci_receive_control_block.current_state)
```

```
        {
            s_sci_receive_control_block.current_state = FWUP_SCI_CONTROL_BLOCK_B;
            s_sci_receive_control_block.p_sci_buffer_control =
&s_sci_buffer_control[FWUP_SCI_CONTROL_BLOCK_B];
        }
        else
        {
            s_sci_receive_control_block.current_state = FWUP_SCI_CONTROL_BLOCK_A;
            s_sci_receive_control_block.p_sci_buffer_control =
&s_sci_buffer_control[FWUP_SCI_CONTROL_BLOCK_A];
        }

    }
}
```

The following lines of the source code shown above

```
        for (uint16_t i = 0; i < 1024; ++i)
        {
            s_sci_receive_control_block.p_sci_buffer_control->buffer[i] = received_packet[8 +
i];
            s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size++;
        }
```

store the new firmware fragment (1,024 bytes) that is received as a payload attached to the WriteBlock command in s_sci_receive_control_block.p_sci_buffer_control->buffer and increment s_sci_receive_control_block.p_sci_buffer_control->buffer_occupied_byte_size.

② Create a wrapper function for the Abort function without arguments to allow it to be called from outside the FWUP FIT module.

```
fwup_err_t R_FWUP_Abort_global( void )
{
    return (fwup_err_t)R_FWUP_Abort( &g_file_context );
}
```

③ Add declarations to the r_fwup_if.h file for the two functions added in ① and ②.

```
void fwup_receive_fileblock(uint8_t []);
fwup_err_t R_FWUP_Abort_global(void);
```

5.2.1 Implementing the 2nd OTA Controller

An example design of state transitions of the 2nd OTA Controller in which FWUP FIT module API functions are called using commands from the 1st MCU as events is shown below.

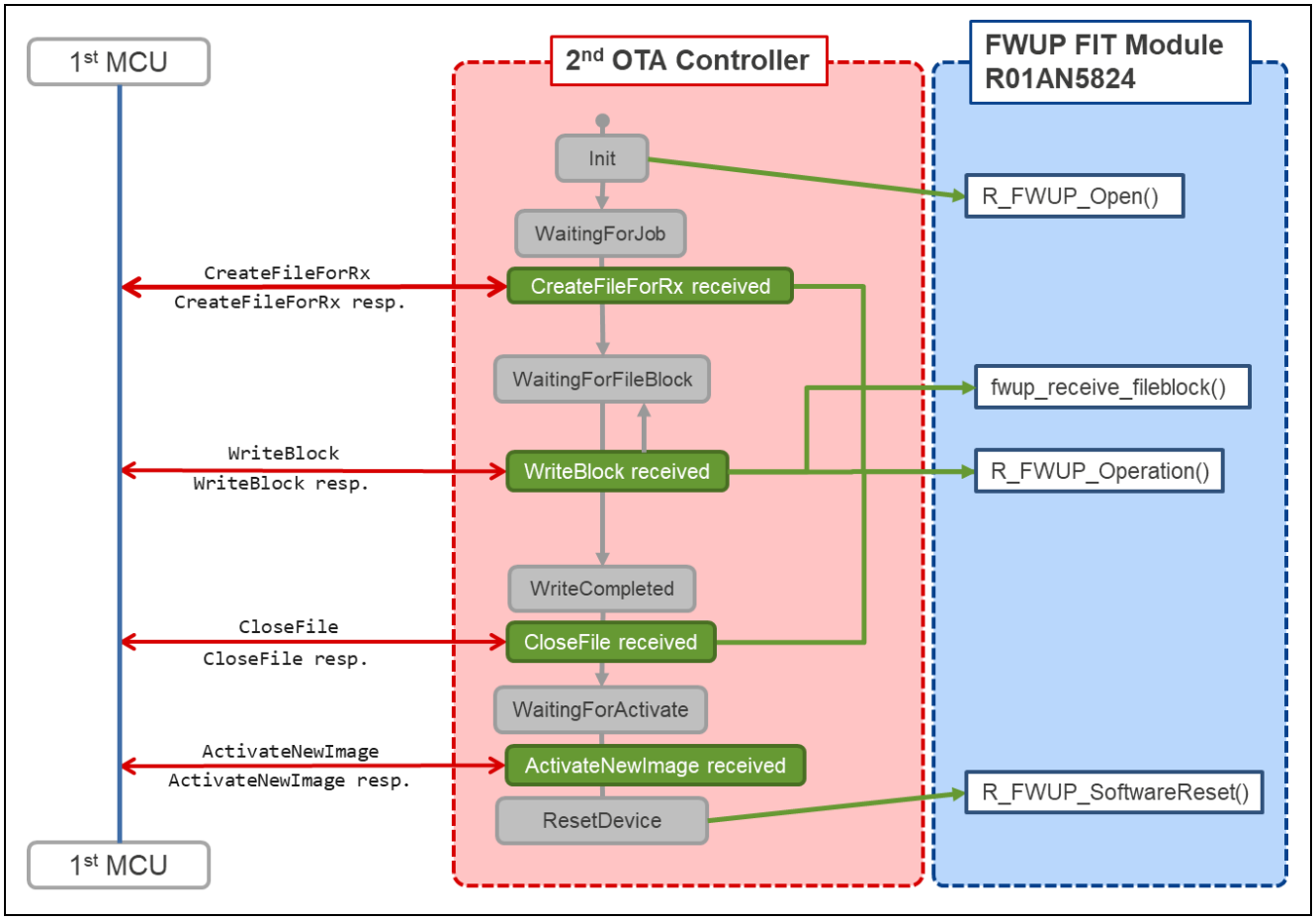


Figure 5.1 Example Design of 2nd OTA Controller State Transitions

In the figure above, FWUP FIT module API functions appear in the blue rectangle at the right, and a diagram of state transitions of the 2nd OTA Controller is shown within the red rectangle. In the diagram of 2nd OTA Controller state transitions, the items in gray boxes represent states, the items in green boxes represent events, and the green arrows indicate the FWUP FIT module API functions called when events occur.

A table summarizing the 2nd OTA Controller state transitions illustrated in Figure 5.1 is shown below.

Table 5.1 2nd OTA Controller State Transitions

Current State	Event	Function Executed	Next State	
Init	—	R_FWUP_Open()	WaitingForJob	
WaitingForJob	CreateFileForRx command received	R_FWUP_Operation()	WaitingForFileBlock	
WaitingForFileBlock	WriteBlock command received	fwup_receive_fileblock() R_FWUP_Operation() R_FWUP_Operation()	*1	*2
			WriteCompleted	WaitingForFileBlock
WriteCompleted	CloseFile command received	R_FWUP_Operation()	WaitingForActivate	
WaitingForActivate	ActivateNewImage command received	R_FWUP_SoftwareReset()	—	
—	Abort command received	R_FWUP_Abort_global() fwup_communication_close() fwup_state_monitoring_close() fwup_flash_close() R_FWUP_Close() R_FWUP_Open()	WaitingForJob	
	Error during processing by 2nd OTA Controller			
—	GetPlatformImageState command received	R_FWUP_GetPlatformImageState()	—	

Notes: 1. When fwup_get_status() == FWUP_STATE_CHECK_SIGNATURE
2. When fwup_get_status() == FWUP_STATE_DATA_RECEIVE

For an example implementation, refer to the sample code for the TB-RX23W board using OTA Library v3.x.x in the following application note.

RX65N Group Sample Code for OTA Update of Secondary Device with Amazon Web Services Using FreeRTOS

6. Creating Initial Firmware and Firmware Update for 2nd MCU

6.1 Creating Initial Firmware

Build the bootloader and user program projects created as described above and generate mot files. Use Renesas Secure Flash Programmer to concatenate the two .mot files to create the initial firmware mot file. The parameters are listed below.

Table 6.1 Parameter Settings when Creating Initial Firmware File in Renesas Secure Flash Programmer

Select MCU	RX23W(ROM 512KB)/Secure Bootloader=64KB
Select Firmware Verification Type	sig-sha256-ecdsa
Private Key Path(PEM Format)	Path to private key generated above
Select Output Format	Bank 0 User Program + Boot Loader (Motorola S Format)
Boot Loader File Path (Motorola Format)	Path to bootloader mot file
Bank0 User program Firmware Sequence Number	1
Bank0 File Path (Motorola Format)	Path to user program mot file

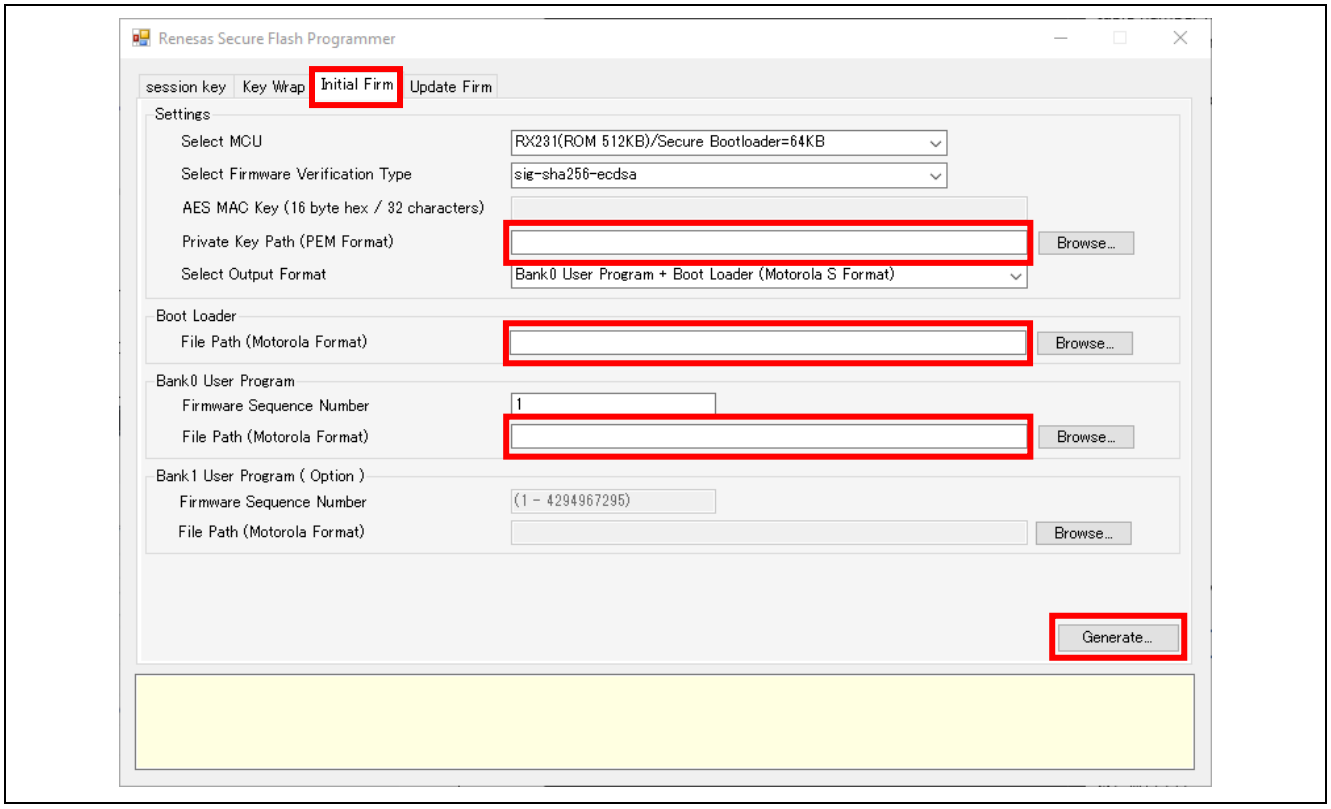


Figure 6.1 Renesas Secure Flash Programmer Settings Screen when Creating Initial Firmware File

6.2 Creating Firmware Update File

Build the updated user program project created as described above and generate a mot file. Use Renesas Secure Flash Programmer to convert the mot file to a rsu file in Renesas' proprietary binary firmware data format. For details of the Renesas Secure Update (RSU) format, refer to section 7 of Renesas MCU Firmware Update Design Policy, Rev.1.00 (Renesas.com).

Table 6.2 Parameter Settings when Creating Firmware Update File in Renesas Secure Flash Programmer

Select MCU	RX23W(ROM 512KB)/Secure Bootloader=64KB
Select Firmware Verification Type	sig-sha256-ecdsa
Private Key Path(PEM Format)	Path to private key generated above
Bank0 User program Firmware Sequence Number	1
Bank0 File Path (Motorola Format)	Path to firmware update mot file

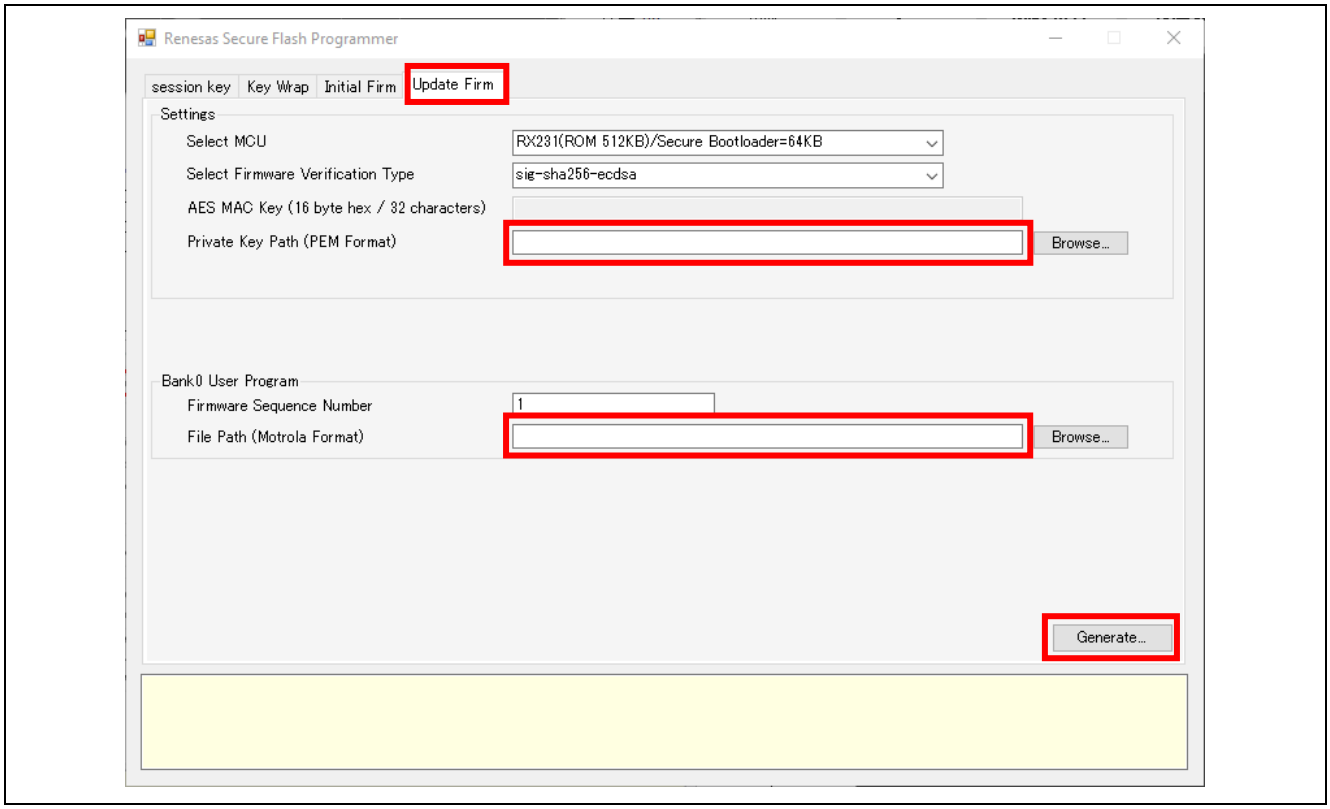
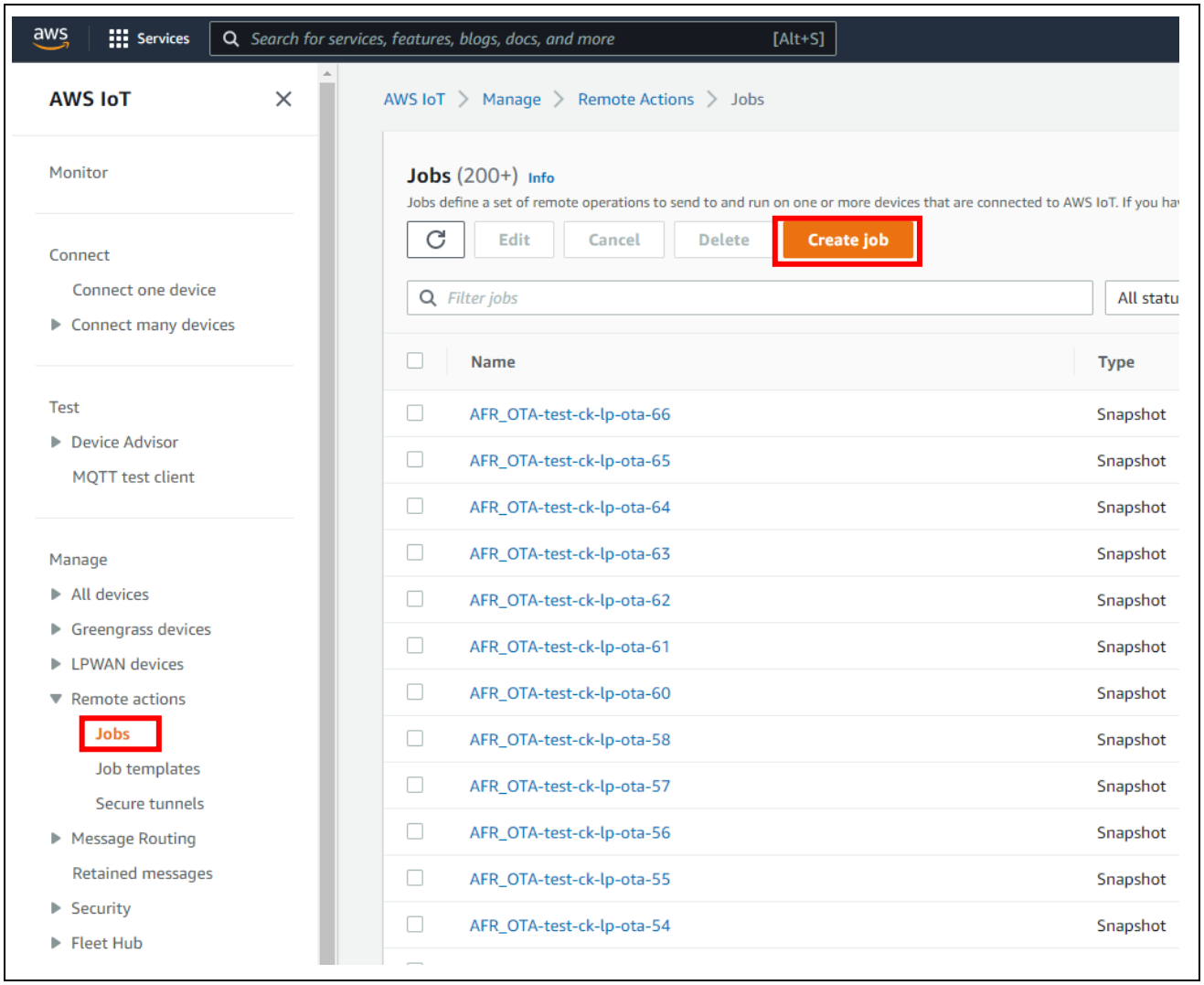


Figure 6.2 Renesas Secure Flash Programmer Settings Screen when Creating Firmware Update File

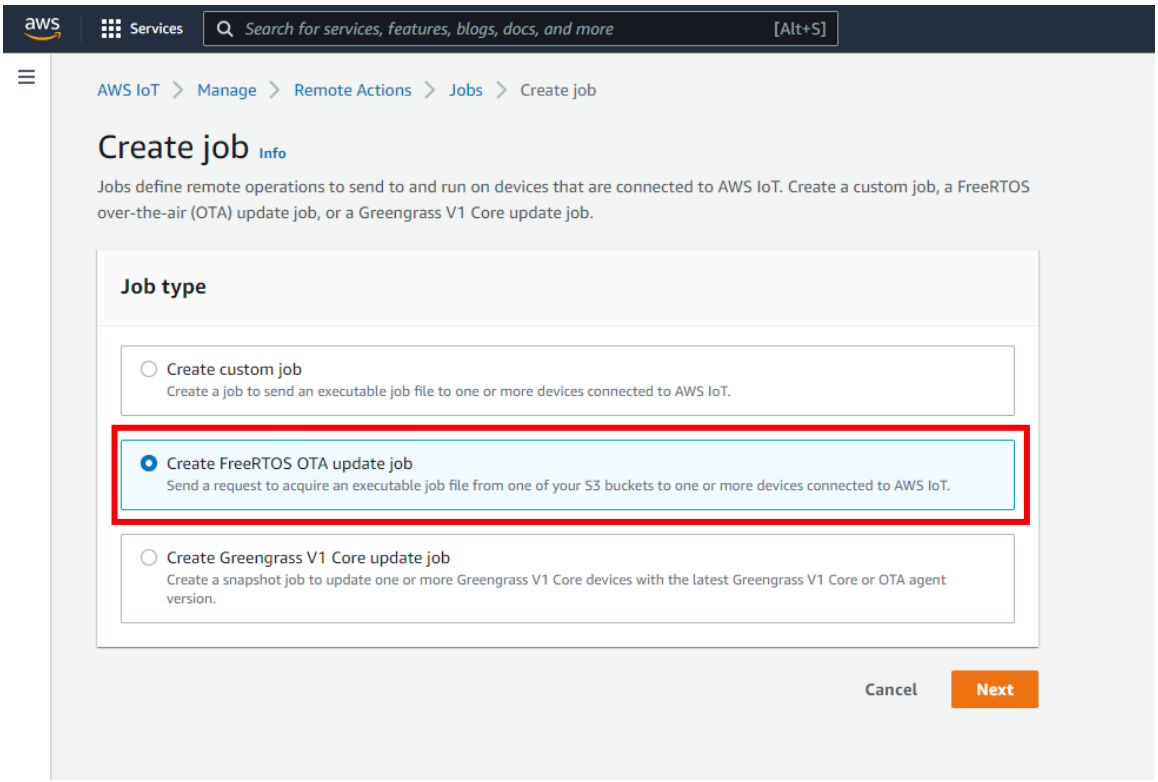
7. Executing 2nd MCU OTA Update

The procedure for creating a 2nd MCU OTA update job from the AWS IoT Core browser console is as follows.

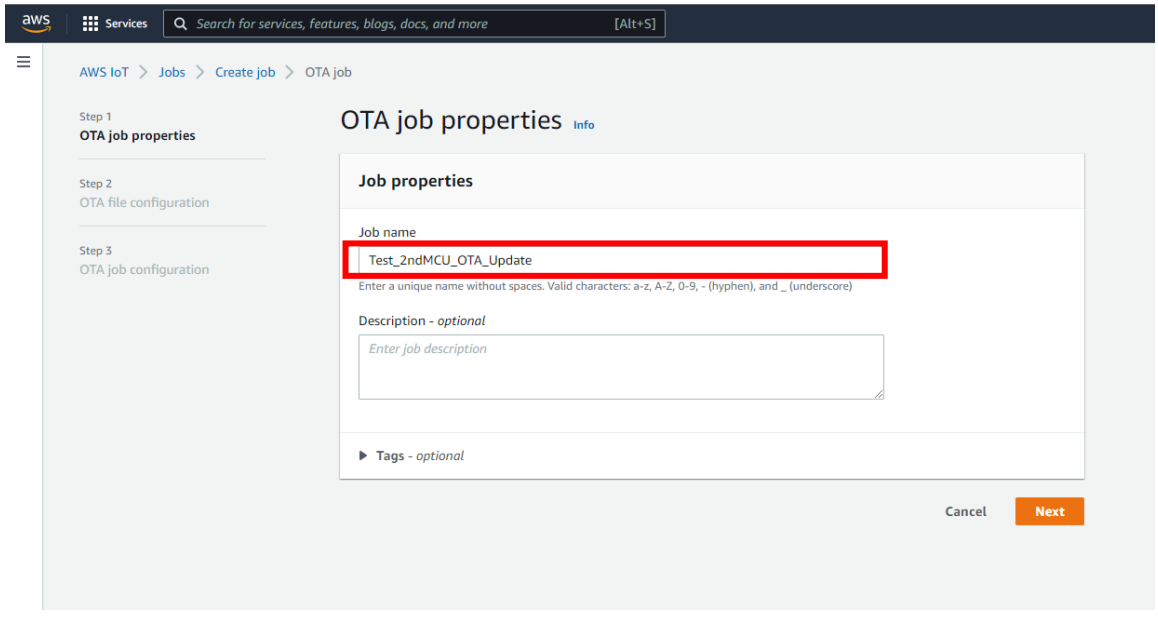
1. Open the AWS IoT Core browser console, select **Manage** → **Jobs** from the menu bar at left, and click the **Create job** button.



2. On the **Create job** page, select **Create FreeRTOS OTA update job** and click the **Next** button.

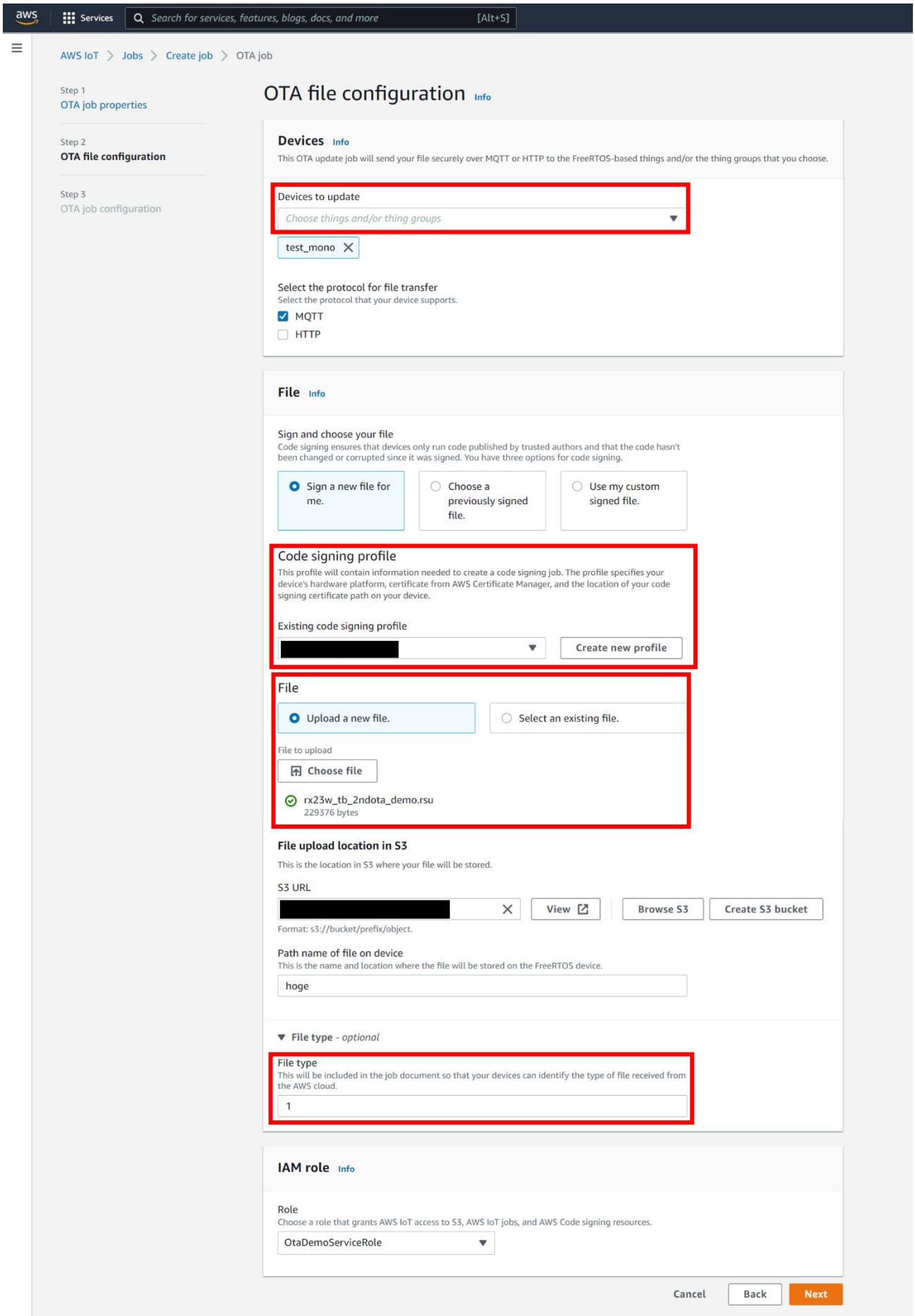


3. On the **OTA job Properties** page, enter a name for **Job name** and click the **Next** button.

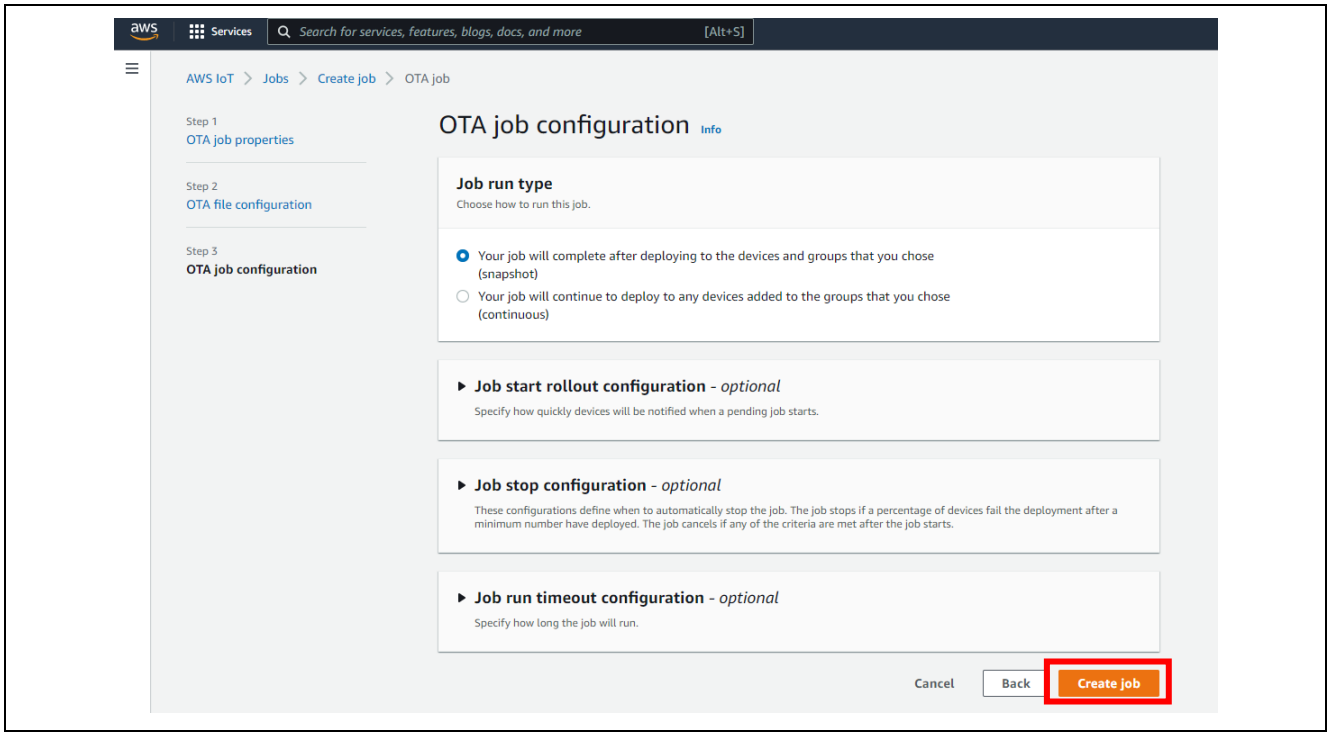


4. On the **OTA file configuration** page, enter information for each item as follows.
   ① For **Device to update**, enter the thing name registered on AWS IoT services for the 1st MCU.
   ② For **Code signing profile**, enter a profile of your choice.
   Note: The code signing profile specified here is not used for code-signer verification of the 2nd MCU firmware. The code signature is designated in the file when the firmware update rsu file is created using Renesas Secure Flash Programmer. It is therefore not necessary to create a code signing profile using the certificate and private key used when creating the new firmware for the 2nd MCU.
   ③ For **File to upload**, select the new firmware file (rsu format) for the 2nd MCU.
   ④ For **File type**, enter the value assigned to 2nd MCU OTA update when the firmware was created.

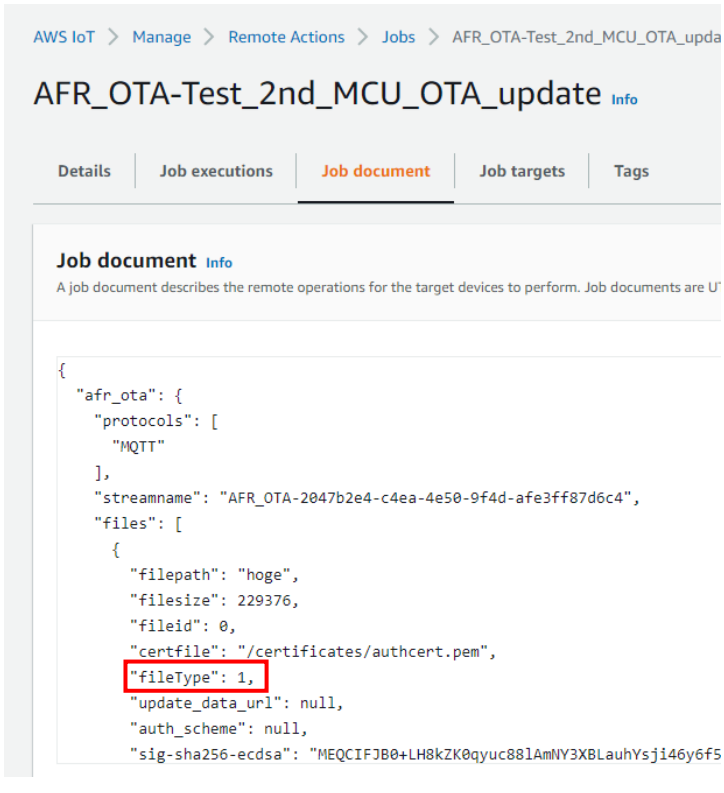
   For the other items, enter the same values used when creating ordinary OTA jobs, then click the **Next** button.


aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
AWS IoT > Jobs > Create job > OTA job
Step 1
OTA job properties
Step 2
OTA file configuration
Step 3
OTA job configuration
OTA file configuration Info
Devices Info
This OTA update job will send your file securely over MQTT or HTTP to the FreeRTOS-based things and/or the thing groups that you choose.
Devices to update
Choose things and/or thing groups
test_mono
Select the protocol for file transfer
Select the protocol that your device supports.
MQTT
HTTP
File Info
Sign and choose your file
Code signing ensures that devices only run code published by trusted authors and that the code hasn't been changed or corrupted since it was signed. You have three options for code signing.
Sign a new file for me.
Choose a previously signed file.
Use my custom signed file.
Code signing profile
This profile will contain information needed to create a code signing job. The profile specifies your device's hardware platform, certificate from AWS Certificate Manager, and the location of your code signing certificate path on your device.
Existing code signing profile
Create new profile
File
Upload a new file.
Select an existing file.
File to upload
Choose file
rx23w_tb_2ndota_demo.rsu
229376 bytes
File upload location in S3
This is the location in S3 where your file will be stored.
S3 URL
View
Browse S3
Create S3 bucket
Format: s3://bucket/prefix/object.
Path name of file on device
This is the name and location where the file will be stored on the FreeRTOS device.
hoge
File type - optional
File type
This will be included in the job document so that your devices can identify the type of file received from the AWS cloud.
1
IAM role Info
Role
Choose a role that grants AWS IoT access to S3, AWS IoT jobs, and AWS Code signing resources.
OtaDemoServiceRole
Cancel
Back
Next

5. There is no need to make changes on the **OTA job configuration** page, so simply click the **Create job** button.



The procedure described above will create an OTA job for the 2nd MCU OTA update and deliver it to the specified device.
If you check the Job document of the newly created OTA job, you can confirm that the member **`"filetype": 1`** has been added. This means that when the Job document is received by the 1st MCU, control will branch to processing for the 2nd MCU OTA update.



Revision History

Rev.	Date	Description	
		Page	Summary
1.01	Jun. 30, 2022	—	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 promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
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/.