RX Family

Delta Firmware Update Using Firmware Update Module

Introduction

This application note explains how to perform Delta Firmware Update for the RX Family using the Firmware Update Module. Transferring only the differences from the current firmware reduces the update image size, the amount of data transferred, and the update time, enabling efficient firmware update.

This application note describes how Delta Firmware Update work, the update procedure, and how to verify the operation of the demo projects using the RX65N.

Target Device

RX65N Group

When applying this application note to another microcontroller, please modify it according to the specifications of that microcontroller and evaluate it thoroughly.

Related Application Notes

- RX Family Firmware Update Module Firmware Integration Technology (R01AN6850)
- RX Family Dual Mode Usage Guide (R01AN6894)
- RX65N Group CK-RX65N v2 User's Manual (R20UT5366)

(Please obtain the latest version from the Renesas Electronics website.)

Target Compiler

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX

For the confirmation environment of each compiler, please refer to "3 Operation Confirmation Environment".

Content

1. Overview

1.1 What is Delta Firmware Update?

A Delta Firmware Update is a method that includes only the data for address ranges that differ from the current firmware in the update image.

By adopting this method, the size of the update image can be reduced, enabling shorter firmware update times and decreased data transfer volume. This enables efficient firmware update in embedded systems with constraints on communication bandwidth and write times.

1.2 How Delta Firmware Update Works

This section explains how the Delta Firmware Update works in this application note.

Renesas offers a Firmware Update Module for RX microcontrollers.

The Firmware Update Module writes the update firmware to a pre-allocated buffer area in the code flash memory of the target MCU, verifies its integrity, and then activates the new firmware. Delta Firmware Update also operate within the framework of this firmware update mechanism (dual-bank system).

The update image generated by the Firmware Update Module contains all data of the target firmware, except for blank regions. On the other hand, the Delta Firmware Update compare the differences between the current firmware and the update firmware, generating an update image that includes only the parts that have changed.

When the target MCU receives the Delta Firmware Update image (hereafter referred to as the diff image), it writes the differential data (hereafter referred to as the diff data) to the appropriate addresses by using the APIs provided by the Firmware Update Module. Furthermore, for areas without differences, we supplement them by copying data from the current firmware and build the final update firmware.

1.3 How to Create a Diff Image

This section explains how to create a diff image.

The steps to create a diff image are as follows.

(1) Step 1: Prepare the current firmware and the update firmware.

(2) Step 2: Extract the difference between the current firmware and the update firmware.

(3) Step 3: Convert the extracted diff data and update firmware into update image files (RSU format) using Renesas Image Generator.

(4) Step 4: Merge the diff data and the update image file of the update firmware to create the diff image.

Details of each step will be explained below.

1.3.1 Step 1: Prepare the Current Firmware and the Update Firmware

Prepare the current firmware and update firmware.

The current firmware is the firmware that runs when the update starts on the MCU to be updated. Instead of the initial image file created with Renesas Image Generator, use the MOT file created immediately after the build. Similarly, the update firmware uses MOT files immediately after build.

1.3.2 Step 2: Extraction of the Diff Data

Extract the difference between the update firmware and the current firmware.

First, the current firmware and update firmware are divided into blocks by program unit for code flash of the MCU to be updated, and the differences between the two firmwares are compared at the block level (see the left figure below). For example, if the code flash program unit is 128 bytes on the MCU, the difference between the current firmware and update firmware is compared in blocks of 128 bytes.

Next, if a block contains a difference, the entire block is extracted as a diff block. (see the middle figure below)

Finally, the extracted differential blocks (hereafter, the diff blocks) are merged to form the final diff data.

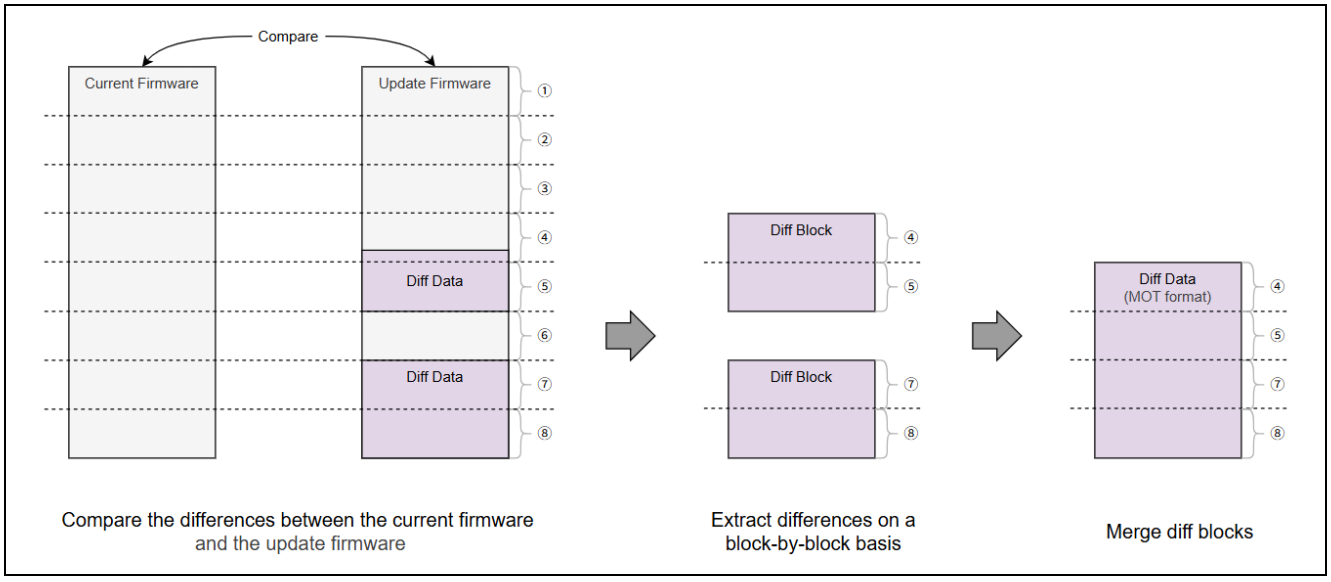


Figure 1-1 Procedure for Extracting Diff Data

1.3.3 Step 3: Convert to the Update Image File (RSU format)

Input the update firmware and the diff data created in Step 2 into the Renesas Image Generator to generate the update image file (RSU format).

During the Renesas Image Generator update image file generation process, firmware information is converted from MOT format to binary data, and RSU headers (signature and address information) are added.

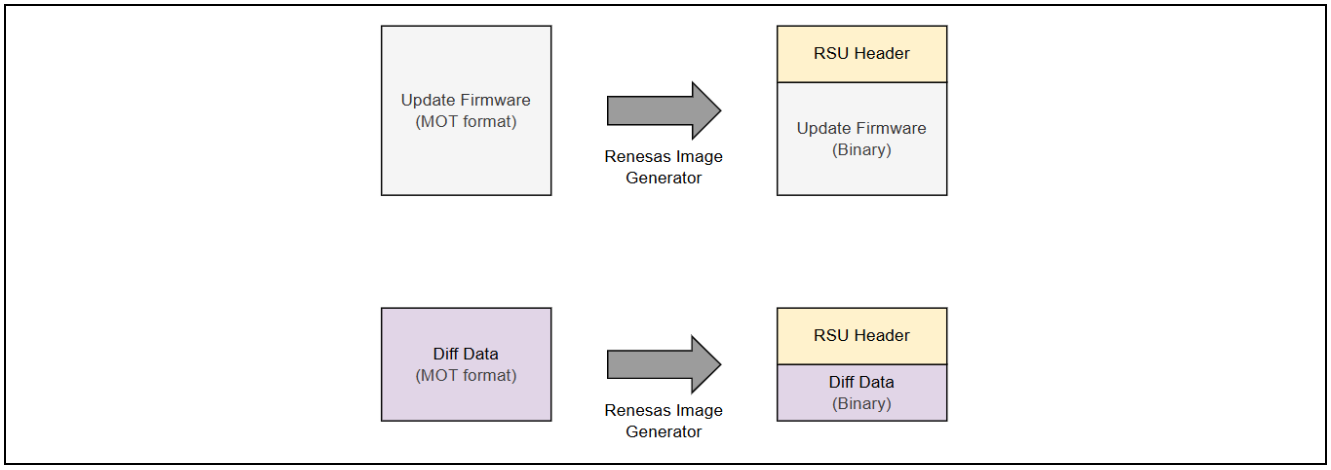


Figure 1-2 Conversion to Update Image File

1.3.4 Step 4: Create the Diff Image File

Merge the diff data generated in Step 3 with the update image file of the update firmware to create the diff image.

The diff image consists of the following four regions, in order from the beginning of the data:

- RSU Header (Signature Information)
- RSU Header (Address Information for Diff Data)
- Diff Data
- RSU Header (Address Information for Update Firmware)

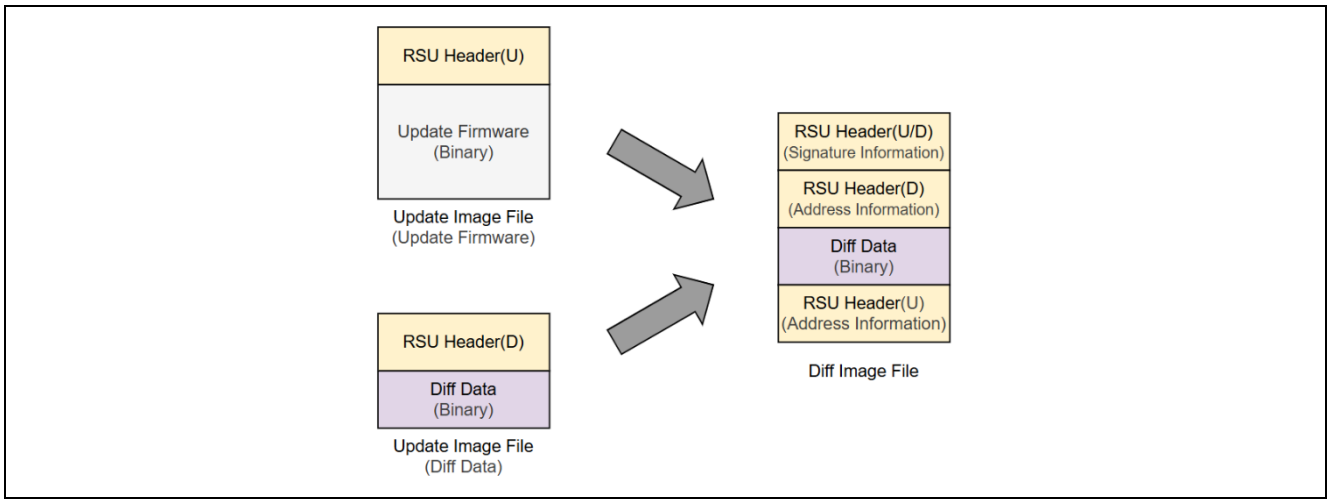


Figure 1-3 Creating Diff Image Files

(1) RSU Header (Signature Information) − 768 bytes

Each item in the RSU header of the diff image file is built in accordance with Table 1-1 below, using either the RSU header of the update firmware's update image file (RSU Header(U) in the figure above) or the RSU header of the diff data's update image file (RSU Header(D) in the figure above).

Table 1-1 How to Build RSU Headers for Diff Image Files

Offset	Item	Length (byte)	Use	Reason
0x00000000	Magic Code	7	-	-
0x00000007	Reserved	1	-	-
0x00000008	Firmware Verification Type	32	Update firmware	Code signature verification is performed for the entire area of the update image,
0x00000028	Signature size	4	Update firmware	Same as above
0x0000002C	Signature	64	Update firmware	Same as above
0x0000006C	RSU File Size	4	Diff Data	The RSU file size field specifies the size of the update image file provided during firmware update.
0x00000070	Reserved	400	-	-

(2) RSU Header (Address Information for Diff Data) − 256 bytes

During firmware update, the address information of the diff data is used to place binary diff data at the correct address on the code flash.

(3) Diff Data

The diff data in the diff image file uses binary data from the update image file generated in Step 3.

(4) RSU header (address information for update firmware) − 256 bytes

The address information of the RSU header (RSU Header(U)) of the update image file for the update firmware is used. This is because the update image area subject to final signature verification includes the RSU header address information.

1.4 Delta Firmware Update Processing on MCU

The procedure for executing the Delta Firmware Update using the diff image created above are shown in Figure 1-4 and Figure 1-5.

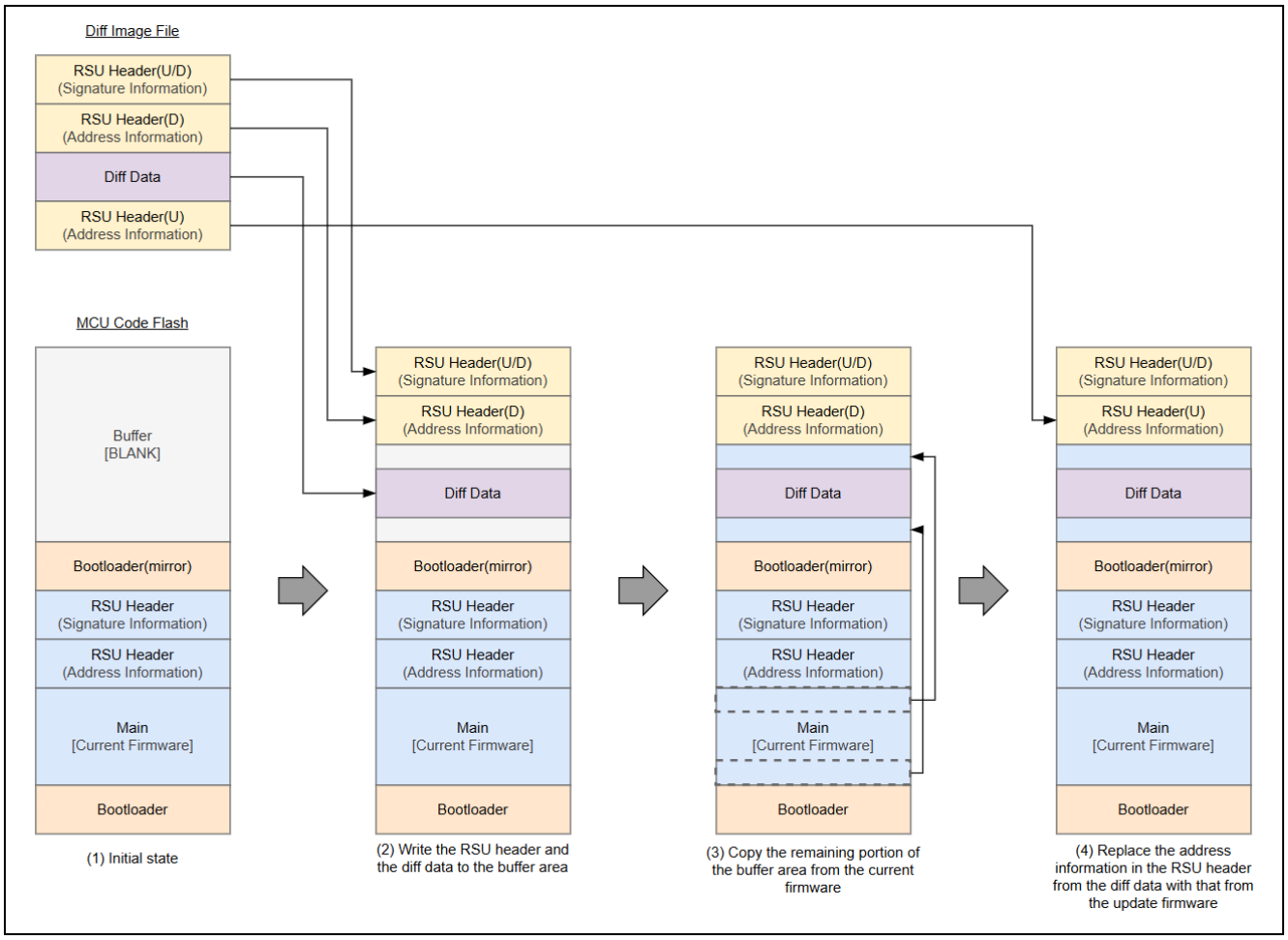


Figure 1-4 Procedure for Delta Firmware Update on MCU (1)

(1) This is the initial state of the dual-bank system. There is no special setup for the Delta Firmware Update.

(2) Use the R_FWUP_WriteImage function of the Firmware Update Module to write the RSU header and diff data of the diff image to the buffer area.

(3) Copy the current firmware data to areas that are not diffs of the update firmware.

(4) Replace the RSU header (address information for diff data) written in (2) with the RSU header (address information for update firmware) in the diff image.

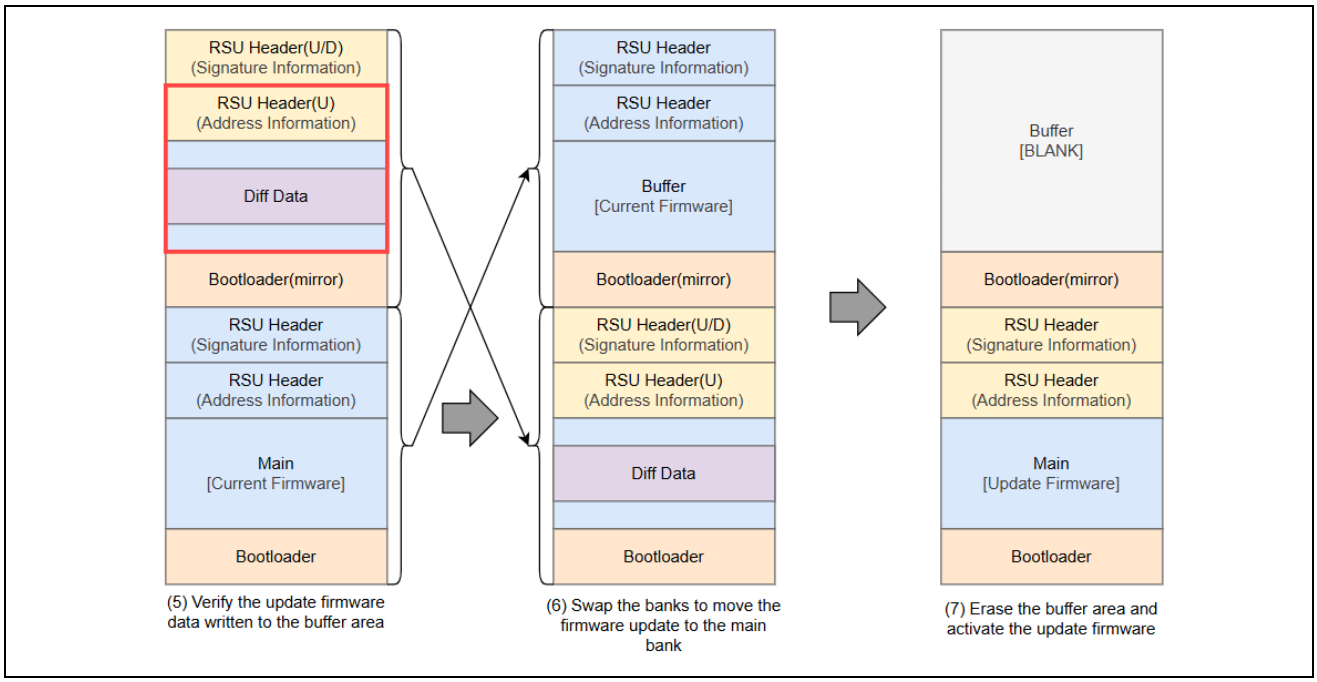


Figure 1-5 Procedure for Delta Firmware Update on MCU (2)

(5) Verify the validity of the update firmware data written to the buffer area.

(6) Perform a bank swap and move the update firmware to the main area.

(7) Erase the buffer area and enable the update firmware.

1.5 Exception Handling

This explains the cases where the Delta Firmware Update cannot be performed.

1.5.1 In case of Differences from Non-FF Block to All-FF Block

If a block that contains at least one byte of non-0xFF data in the current firmware (a Non-FF block) becomes entirely 0xFF (an All-FF block) in the update firmware, that All-FF block is removed as a region with no data during RSU file generation by Renesas Image Generator and is not written to the RSU file. Therefore, in this Delta Firmware Update method, the difference from Non-FF blocks to All-FF blocks is not reflected.

The diff image generation script (Delta Image Generator) provided as a sample program in this application note does not output a diff image when it includes differences from Non-FF blocks to All-FF blocks.

1.5.2 When Number of Diff Blocks between Current Firmware and Update Firmware Is Zero

Since there are no differences, it is considered the same file. Therefore, the diff image file will not be output.

1.5.3 When Number of Diff Blocks between Current Firmware and Update firmware Is 32 or More

Due to the Renesas Image Generator specification, the maximum address information managed by the RSU header offset 0x200 exceeds 31, causing an error. Therefore, the diff image file will not be output.

2. Delta Image Generator

Delta Image Generator (delta-image-gen.py) is a Python script that generates firmware images for use in Delta Firmware Update in this application note.

The Delta Image Generator runs the Renesas Image Generator internally. Renesas Image Generator is a tool that generates initial and update images compatible with firmware update included with the Firmware Update Module. For more details and operating environment of the Renesas Image Generator, please refer to "RX Family Firmware Update Module Firmware Integration Technology (R01AN6850)".

2.1 How to Use

The command format for delta-image-gen.py is as follows.

```
python delta-image-gen.py <options>
```

Table 2-1 delta-image-gen.py Options

Options	Required	Description
--before, -b	Yes	Specify the MOT file of the current firmware with absolute/relative paths.
--after, -a	Yes	Specify the MOT file for the update firmware with absolute/relative paths.
--output, -o	Yes	Specify the file name of the diff image to be exported. The output file name will be "[Input String].delta_rsu".
--block-size, -bs	No	Specify the size (bytes) of the diff block. It must be a program-unit multiple of the ROM of the MCU to be updated. Default value: 128
--tool-dir, -t	No	Specify the folder where the Renesas Image Generator (image-gen.py) is located with absolute/relative paths. Default value: ./tool
-ip	Yes	Specify the -ip option when the Delta Image Generator runs the Renesas Image Generator internally. (Note)
-ff	No	Specify the -ff option when the Delta Image Generator runs the Renesas Image Generator internally. (Note) Default: BareMetal
-vt	No	Specify the -vt option when the Delta Image Generator runs the Renesas Image Generator internally. (Note) Default value: ecdsa
-key	No	Specify the -key option when the Delta Image Generator runs the Renesas Image Generator internally. (Note)

Note: For details, please refer to "RX Family Firmware Update Module Firmware Integration Technology (R01AN6850) 5.1 How to Generate an Image".

2.2 Usage Examples

Below is an example of a command input for generating diff image using the Delta Image Generator.

- Current firmware: ./fwup_main_v1.mot
- Update firmware: ./fwup_main_v2.mot
- Output diff image name: update_firm
- Diff block size: 128
- Private key used for code signing: ./key/secp256r1.privatekey
- Parameter file for Renesas Image Generator: ./tool/RX65N_DualBank_ImageGenerator_PRM.csv

Example of Command Input

```
python ./delta-image-gen.py -b ./fwup_main_v1.mot -a ./fwup_main_v2.mot -o
update_firm -bs 128 -key ./key/secp256r1.privatekey -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv
```

output file

- update_firm.delta_rsu: Diff image file
- fw_diff.mot: A MOT file extracting the differences in update firmware on a block basis compared to the current firmware
- update_output.rsu: RSU file for update firmware created using Renesas Image Generator
- diff_output.rsu: An RSU file of “fw_diff.mot” created using the Renesas Image Generator

3. Operation Confirmation Environment

The operating environment for this demo is shown below.

Table 3-1 Operation Confirmation Environment (CC-RX)

item	Contents
Integrated Development Environment	Renesas Electronics e^2 studio 2025-12
C Compiler	Renesas Electronics C/C++ Compiler for RX Family (CC-RX) V3.07.00 Compilation options: Added the following options to the default settings of the integrated development environment -lang = c99
Endian	Little Endian
FIT Modules Used	RX Driver Package V1.50 Included Items
Board Used	CK-RX65N v2 cloud kit (Model name: RTK5CK65N0S08001BE)
USB Serial Conversion Board	Pmod USBUART (made by DIGILENT) https://digilent.com/reference/pmod/pmodusbuart/start

Table 3-2 Operation Confirmation Environment (GCC)

item	Contents
Integrated Development Environment	Renesas Electronics e^2 studio 2025-12
C Compiler	GCC for Renesas RX 14.2.0.202505 Compilation options: Added the following options to the default settings of the integrated development environment -std=gnu99
Endian	Little Endian
FIT Modules Used	RX Driver Package V1.50 Included Items
Board Used	CK-RX65N v2 cloud kit (Product model: RTK5CK65N0S08001BE)
USB Serial Conversion Board	Pmod USBUART (made by DIGILENT) https://digilent.com/reference/pmod/pmodusbuart/start

Table 3-3 Operation Confirmation Environment (Other)

item	Contents
Python	3.12.6
Renesas Image Generator	V3.05 (bundled with FWUP FIT v2.06)
Renesas Flash Programmer	V3.22.00
Tera Term	5.5.0

4. Demo

In this demo, we will perform firmware update using the Delta Firmware Update method.

4.1 Demo Operation

The flow of this demo is as follows.

(1) Using the sample project provided in this application note, we will create initial images and update images for the Delta Firmware Update.

(2) Write the initial image onto the CK-RX65Nv2 board. In the initial image, the log output on Tera Term displays "ver 1.0.0," and LED 4 (blue) lights up on the board.

(3) Use Tera Term's serial communication function to send update images to the CK-RX65Nv2 board. When the RX65N receives an update image using the Delta Firmware Update method, it executes the update process.

(4) After the update process is complete, the update image is executed, and the log output on Tera Term changes to "ver 2.0.0". Also, the LED 4 (blue) on the board changes to a blinking action.

4.2 Demo Project Folder Structure

Below is the folder configuration for the demo project.

The Demos folder stores demo projects for CC-RX and GCC.

The DeltaImageGenerator folder stores Python scripts (delta-image-gen.py) for creating diff images.

The tool folder stores the Renesas Image Generator for creating initial and update images provided by FWUP FIT.

The key folder stores the sample private and public keys used for digital firmware signatures in the demo.

```
r01an8321xx0100-rx-delta-fwup
├─Demos
│   ├─ccrx
│   │   ├─boot_loader
│   │   └─fwup_main
│   └─gcc
│       ├─boot_loader
│       └─fwup_main
├─DeltaImageGenerator
│   ├─key
│   │   ├─secp256r1.privatekey
│   │   └─secp256r1.publickey
│   ├─tool
│   │   ├─image-gen.py
│   │   ├─RX65N_DualBank_ImageGenerator_PRM.csv
│   │   └─...
│   └─delta-image-gen.py
├─r01an8321ej0100-rx-delta-fwup.pdf
└─r01an8321jj0100-rx-delta-fwup.pdf
```

4.3 Demo Project Settings

You can perform the Delta Firmware Update using existing projects.

This application note offers projects that reduce the ROM area subject to update by adding sections to further leverage the advantages of the Delta Firmware Update.

The fwup_main.c file, which corresponds to the user application, is set as a single section and placed separately from other sections. This minimizes the difference information in the ROM area caused by code changes in fwup_main.c.

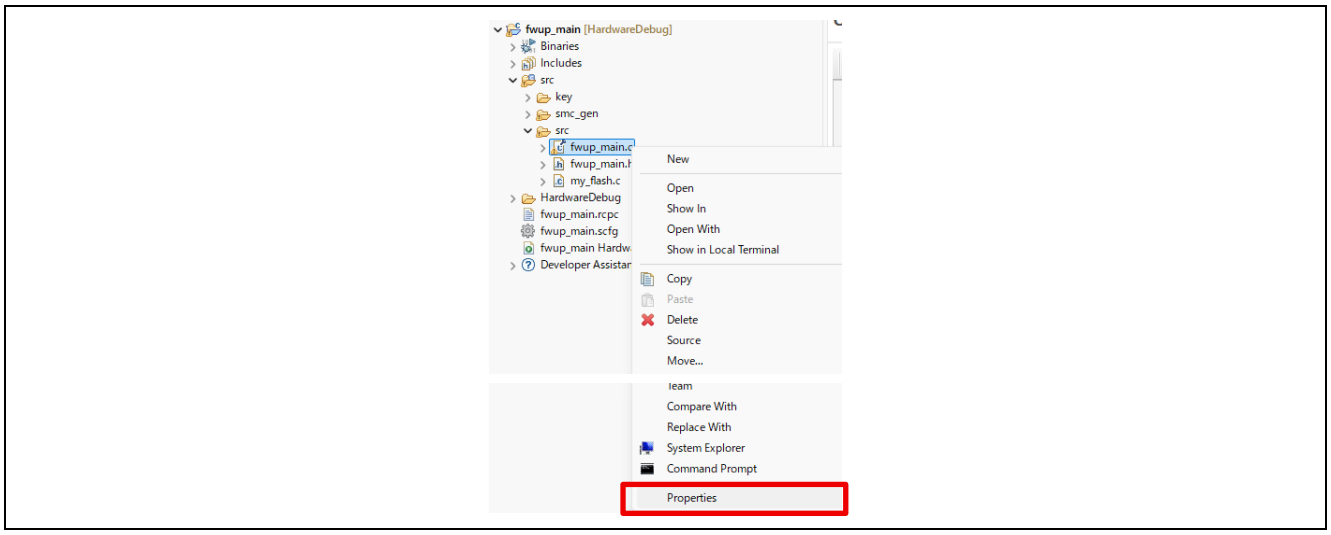
Both demo projects place the fwup_main.c section in the 0xFFF40000.

4.3.1 CC-RX Project

4.3.1.1 Procedure for Changing Sections by File

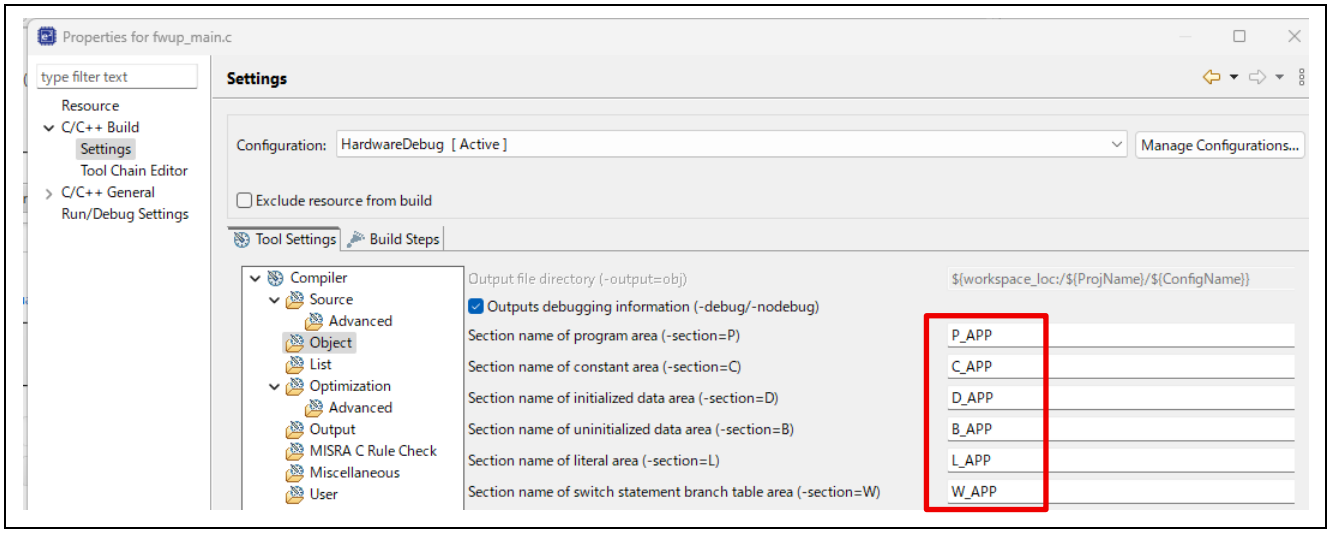
You can change sections by file by following the steps below.

(1) Right-click the "fwup_main.c" file in Project Explorer and select "Properties"



(2) Select "Settings" in "C/C++ Build"

(3) In "Tool Settings", click "Compiler" → "Source" → "Object" and change the section name to "X_APP" (X = P, C, D, B, L, W)



4.3.1.2 Procedures for Placing Sections at the Project Level

Set the section placement address added above by following the steps below.

(1) Right-click the "fwup_main" project in Project Explorer and select "Properties"

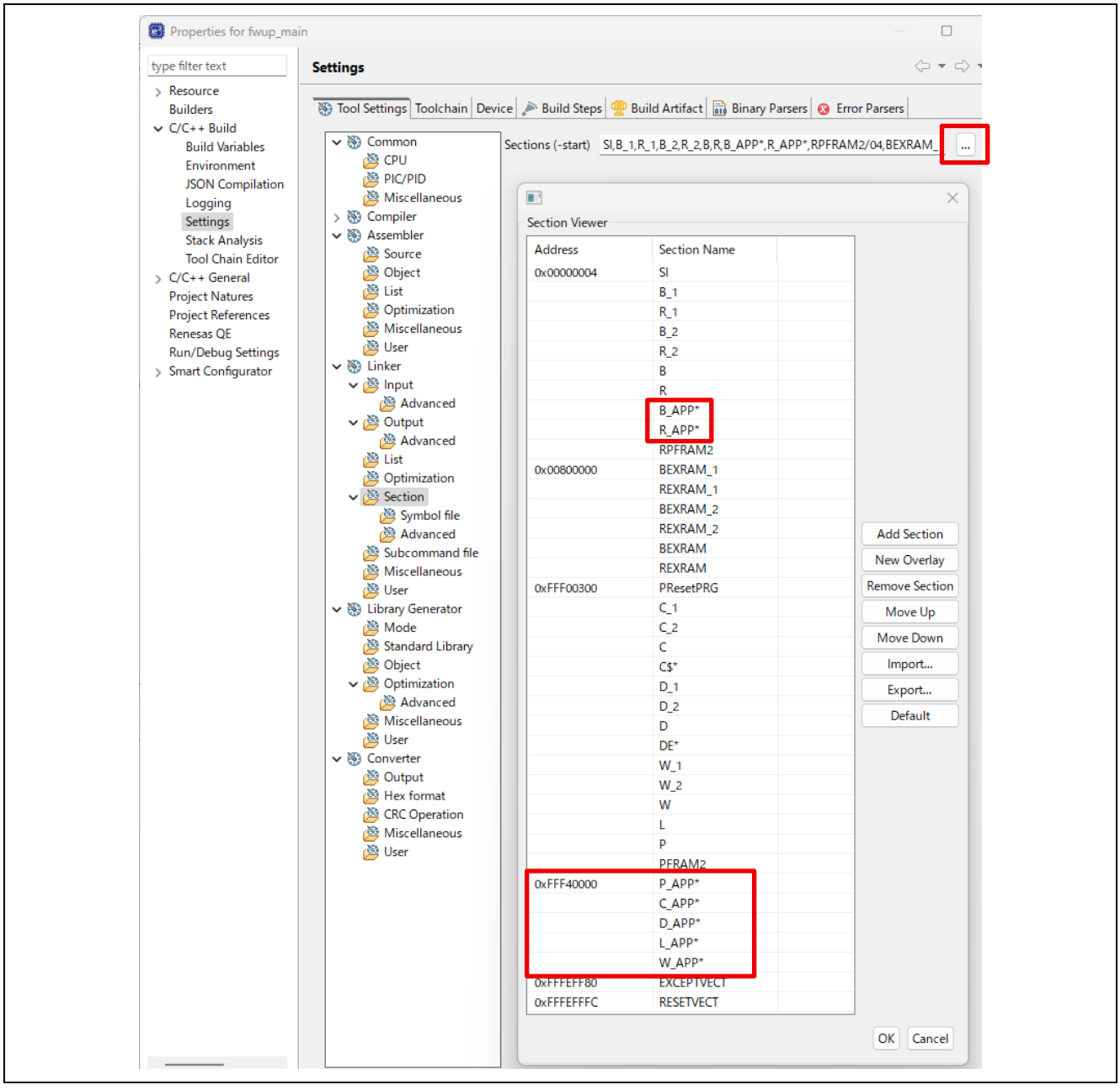
(2) Select "Settings" in "C++ Build"

(3) Click "Linker" → "Section" in "Tool Settings." Furthermore, click "..." to the right of "Sections (-start)".

(4) Add/Change sections

Below is the setup for the demo project.

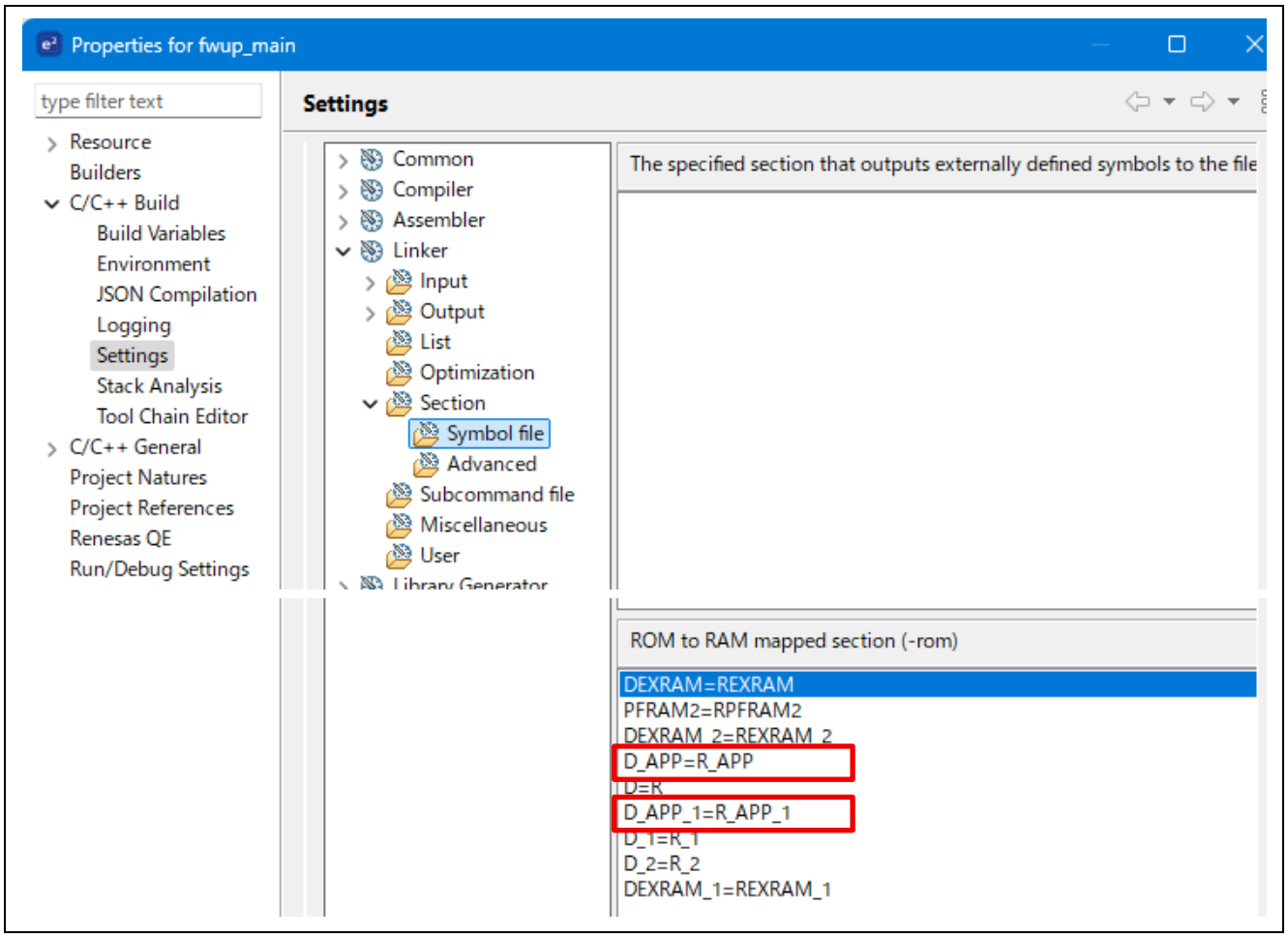
— Added B_APP/R_APP : Located behind the existing B/R section area
— Added P_APP/C_APP/D_APP/L_APP/W_APP : Placed in a new 0xFFF40000



(5) Add a copy section to ROM→RAM in the ROM to RAM Mapped section

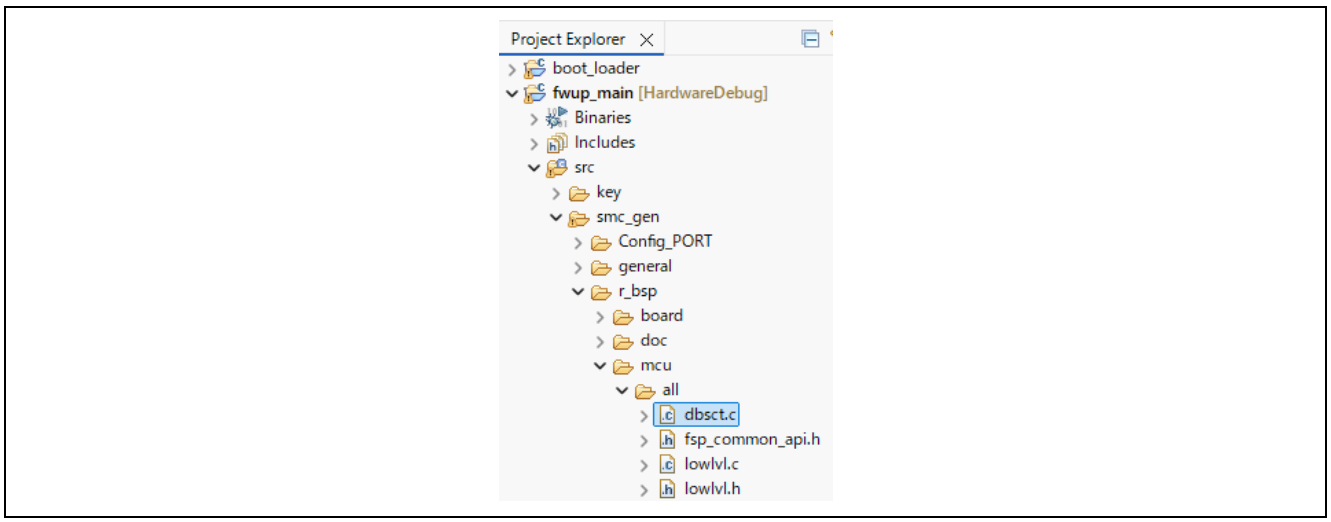
The demo project adds the following:

— D_APP = R_APP
— D_APP_1 = R_APP_1



(6) Add a section to dbsct.c

Add both copy processing to ROM→RAM and initialization of uninitialized areas in dbsct.c.



```
/* Section start */
#pragma section C C$DSEC

extern st_dtbl_t const _DTBL[] = {
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
    { __sectop("D_8"), __secend("D_8"), __sectop("R_8") },
#endif
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
    { __sectop("D_APP"), __secend("D_APP"), __sectop("R_APP") },
    { __sectop("D_APP_1"), __secend("D_APP_1"), __sectop("R_APP_1") }
#if (defined(BSP_CFG_EXPANSION_RAM_ENABLE) & (BSP_CFG_EXPANSION_RAM_ENABLE == 1))
#ifdef BSP_EXPANSION_RAM
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
   ,{ __sectop("DEXRAM_8"), __secend("DEXRAM_8"), __sectop("REXRAM_8") }
#endif
   ,{ __sectop("DEXRAM"), __secend("DEXRAM"), __sectop("REXRAM") },
    { __sectop("DEXRAM_2"), __secend("DEXRAM_2"), __sectop("REXRAM_2") },
    { __sectop("DEXRAM_1"), __secend("DEXRAM_1"), __sectop("REXRAM_1") }
#endif
#endif
#if (BSP_CFG_RTOS_USED == 4) & & (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600PX)
   ,{ __sectop("DRI_ROM"), __secend("DRI_ROM"), __sectop("RRI_RAM") }
#endif /* Renesas RI600PX */
};

/* Section start */
#pragma section C C$BSEC

extern st_btbl_t const _BTBL[] = {
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
    { __sectop("B_8"), __secend("B_8") },
#endif
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") },
    { __sectop("B_APP"), __secend("B_APP") },
    { __sectop("B_APP_1"), __secend("B_APP_1") }
#if (defined(BSP_CFG_EXPANSION_RAM_ENABLE) & (BSP_CFG_EXPANSION_RAM_ENABLE == 1))
#ifdef BSP_EXPANSION_RAM
#ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
   ,{ __sectop("BEXRAM_8"), __secend("BEXRAM_8") }
#endif
   ,{ __sectop("BEXRAM"), __secend("BEXRAM") },
    { __sectop("BEXRAM_2"), __secend("BEXRAM_2") },
    { __sectop("BEXRAM_1"), __secend("BEXRAM_1") }
#endif
#endif
#if (BSP_CFG_RTOS_USED == 4) & & (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600V4)
   ,{ __sectop("BRI_RAM"), __secend("BRI_RAM") }
#endif /* Renesas RI600V4 */
};
```

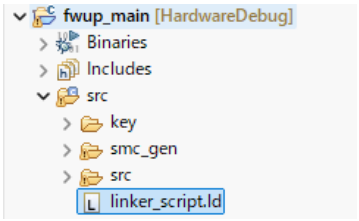
4.3.1.3 About Warnings During Build

The following warning is output for the added W_APP section. This warning does not affect operation and can be ignored.

```
W0561100:Cannot find "W_APP*" specified in option "start"
```

4.3.2 GCC Project

Edit the Linker_script.ld file and add sections.



(1) MEMORY area: No changes required

Because sections are configured on a per-file basis without editing the MEMORY area, no changes are required.

```
MEMORY
{
        RAM : ORIGIN = 0x4, LENGTH = 0x3fffc
        RAM2 : ORIGIN = 0x00800000, LENGTH = 393216
        ROM : ORIGIN = 0xFFF00000, LENGTH = 960K
        OFS : ORIGIN = 0xFE7F5D00, LENGTH = 128
}
```

(2) SECTIONS: Setting the text section in the ROM area

EXCLUDE_FILE (* fwup_main.o) setting excludes the text sections and the RX-specific P, P_1, P_2 sections of fwup_main.c from the text section of the ROM area.

```
SECTIONS
{
        .exvectors 0xFFFEFF80 : AT(0xFFFEFF80)
        {
                "_exvectors_start" = .;
                KEEP(*(.exvectors))
                "_exvectors_end" = .;
        } >ROM
        .fvectors 0xFFFEFFFC : AT(0xFFFEFFFC)
        {
                KEEP(*(.fvectors))
        } >ROM
        .text 0xFFF00300 : AT(0xFFF00300)
        {
                /* Exclude fwup_main.o */
                *(EXCLUDE_FILE(*fwup_main.o) .text*)
                /* P* is not used because there are PFRAM* sections */
                *(EXCLUDE_FILE(*fwup_main.o) P)
                *(EXCLUDE_FILE(*fwup_main.o) P_1)
                *(EXCLUDE_FILE(*fwup_main.o) P_2)
                KEEP(*(.text.*_isr))
                KEEP(*(.text.*isr))
                KEEP(*(.text. Excep_*))
                KEEP(*(.text.*ISR))
                KEEP(*(.text.*interrupt))
                etext = .;
        } >ROM
```

(3) SECTIONS: ROM area rodata section settings

EXCLUDE_FILE (* fwup_main.o) setting excludes the rodata section and the RX-specific C section of fwup_main.c from the rodata in the ROM area.

```
.rodata :
{
        /* Exclude fwup_main.o */
        *(EXCLUDE_FILE(*fwup_main.o) .rodata*)
        *(EXCLUDE_FILE(*fwup_main.o) C*)
        _erodata = .;
} >ROM
```

(4) SECTIONS: Arrangement settings for text and rodata sections for fwup_app

Add a dedicated fwup_main.c app section to the ROM area.

The settings are shown below.

- Set the app section start address to 0xFFF40000
- To store continuously on ROM, sections related to text, P, rodata, and C are set up

```
.pfram2 ALIGN(4) :
{
        _PFRAM2_start = .;
        . += _RPFRAM2_end - _RPFRAM2_start;
        _PFRAM2_end = .;
} >ROM
/* Add the app section for fwup_main.c in ROM */
.app 0xFFF40000 : AT(0xFFF40000)
{
        . = ALIGN(4);
        _fwup_main_start = .;

        KEEP(*fwup_main.o(.text*))
        KEEP(*fwup_main.o(P*))

        *fwup_main.o(.rodata*)
        *fwup_main.o(C*)

        . = ALIGN(4);
        _fwup_main_end = .;
} >ROM
```

(5) SECTIONS: ROM to RAM copy processing and initialization of uninitialized areas: No changes required

ROM to RAM copying and initializing of uninitialized areas are processed in __INITSCT of reset_program.S.

(a) ROM to RAM Copy processing

In copy processing, address “_mdata” is treated as the source for copying variables with initial values.

Since the setting includes all sections of the initial valued variable from address “_data” to “_edata”, no changes are needed.

```
.data : AT(_mdata)
{
        _data = .;
        *(.data)
        *(.data.*)
        *(D)
        *(D_1)
        *(D_2)
        _edata = .;
} >RAM
```

Set the ROM area for the initial value variable from the address “_mdata”. Since the ROM for the data section of “_edata - _data” size from the address “_mdata” is already set up, no changes are needed.

```
.tors :
{
        __CTOR_LIST__ = .;
                …
        . = ALIGN(2);
        _mdata = .;
        . += _edata - _data;
} >ROM
```

(b) Initializing uninitialized areas

In initialization, address “_bss” is treated as the initial initialization address.

Since the setting includes all sections of the variable without initial values from address “_bss” to “_ebss”, no changes are needed.

```
.bss :
{
        _bss = .;
        *(.bss)
        *(.bss.**)
        *(COMMON)
        *(B)
        *(B_1)
        *(B_2)
        _ebss = .;
        . = ALIGN(128);
        _end = .;
} >RAM AT>RAM
```

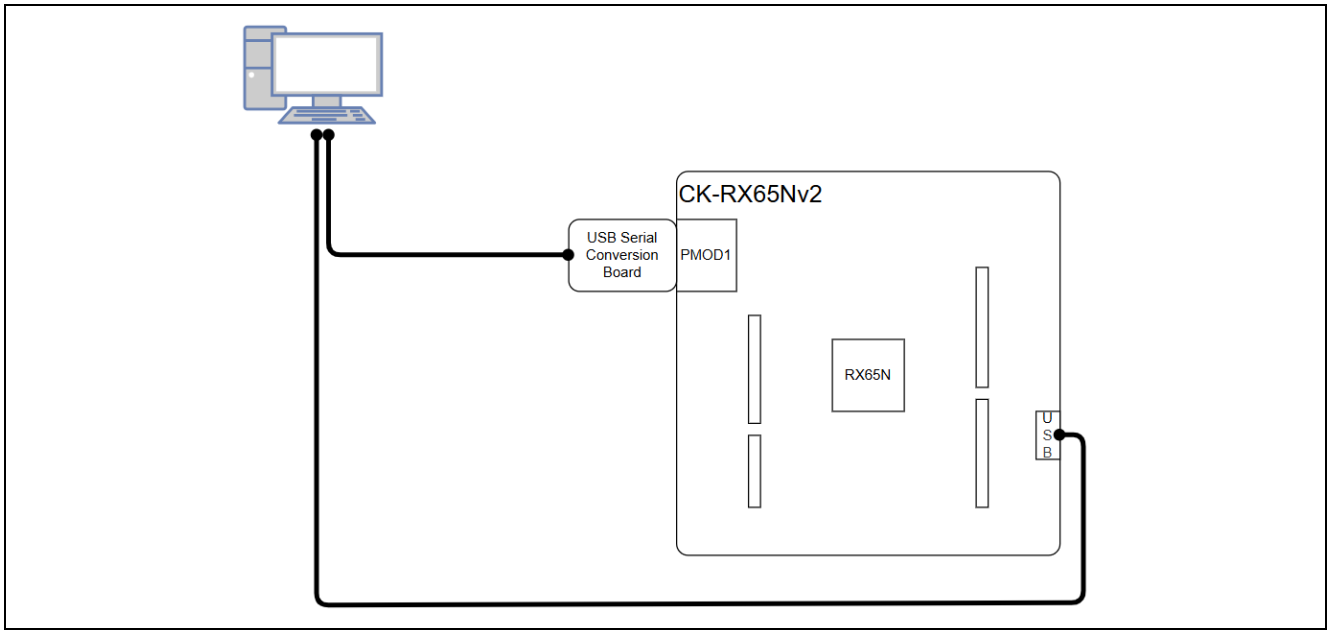
4.3.2.2 About Build Warnings

A total of 7 warnings related to stack usage are output, as shown below. The current project has secured sufficient stack size. Review stack sizes as needed.

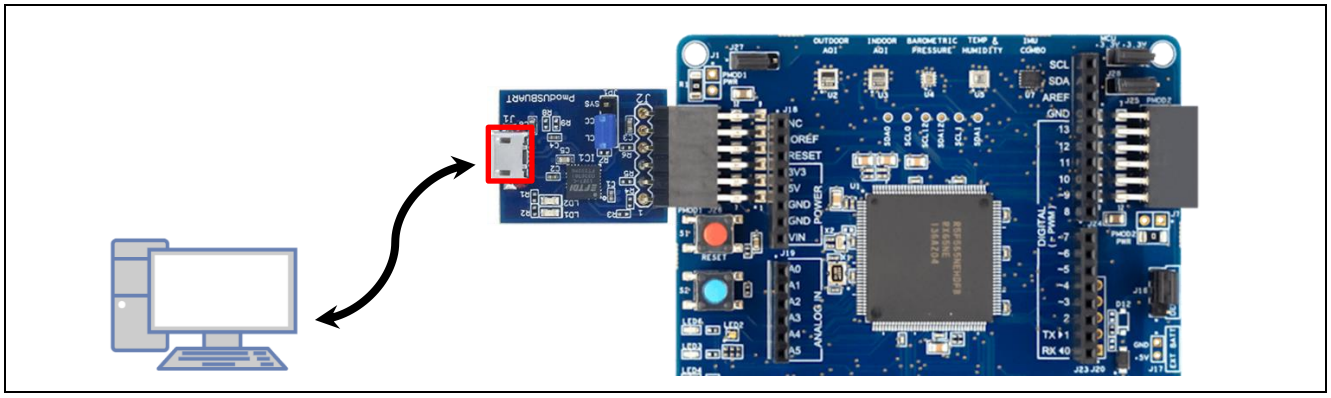
```
warning: stack usage is XXX bytes [-Wstack-usage=]
warning: stack usage computation not supported for this target
```

4.4 Hardware Setup

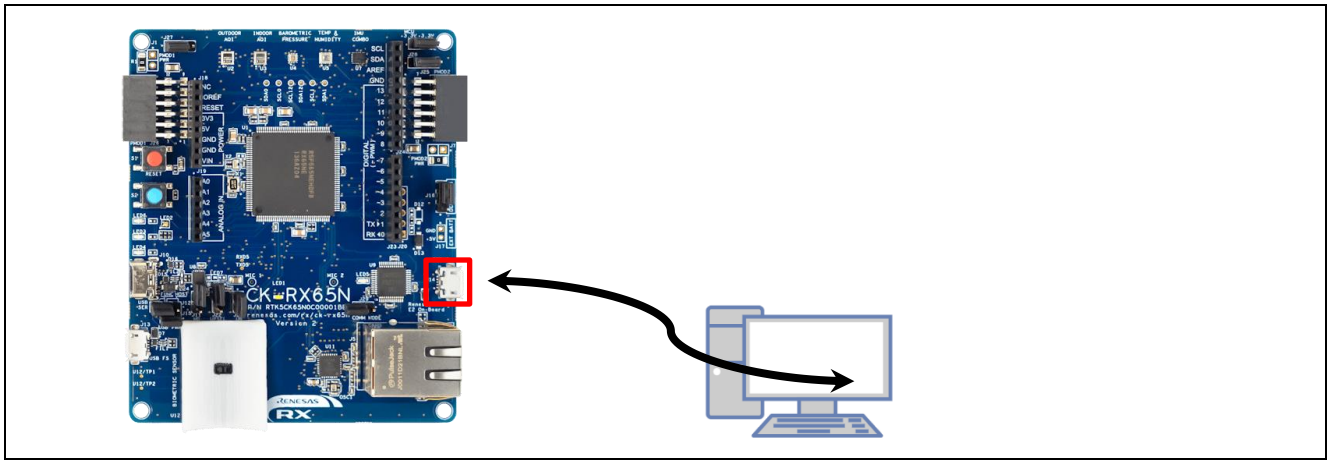
Below is the hardware configuration for the demo.



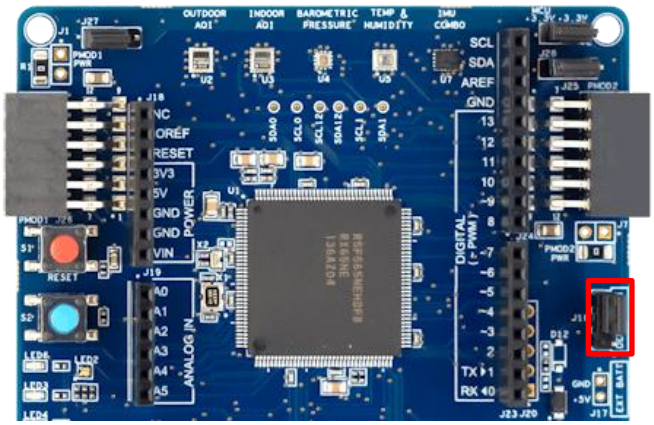
(1) Connect the USB serial conversion board to pin 1-6 of the PMOD1 terminal of the CK-RX65Nv2 board. Connect the USB connector of the USB serial conversion board to your PC.



(2) Connect the E2OB Debugger connector (microUSB Type-B) of the CK-RX65Nv2 board to your PC.



(3) Short-circuit the jumper J16 of the CK-RX65Nv2 board to the DEBUG side (pins 1-2).



4.5 Software Setup

4.5.1 Preparing System Environment

Install the necessary application on your PC to run the demo.

4.5.1.1 Installing Tera Term

It is used to transfer images of firmware update via serial communication from a Windows PC to the primary MCU. In the demo project, we are conducting operational checks using Tera Term 5.5.0.

After installation, please configure the serial port communication settings as shown in Table 4-1.

Table 4-1 Serial Communication Specifications

Item	Contents
Communication Methods	Asynchronous communication
Bitrate	115200bps
Data Length	8-bit
Parity	None
Stop-bit	1 bit
Flow Control	RTS/CTS

4.5.1.2 Installing Python Runtime Environment

Used to create diff images with the Delta Image Generator (delta-image-gen.py). In the demo project, we are conducting a test of operation using Python 3.12.6.

Also, since the Renesas Image Generator (image-gen.py) that Delta Image Generator runs internally uses Python's encryption library (pycryptodome), after installing Python, run the following pip command from the command prompt to install the library.

```
pip install pycryptodome
```

4.5.1.3 Installing Flash Programmer

A flash programmer is needed to write the initial image.

For the demo project, Renesas Flash Programmer (RFP) V3.22.00 is used.

Renesas Flash Programmer (Programming GUI) | Renesas

In the demo project, the dual-bank method is used. Therefore, if firmware update are repeated, it may become unclear which bank is currently active. In addition, for MOT-format files generated by e^2 studio, the bank boot setting is fixed to Bank 0. As a result, a Bank 0/Bank 1 mismatch may occur when writing a MOT file using RFP.

To resolve this issue, refer to "RX Family Dual Mode Usage Guide (R01AN6894)", Section 2.1, "IF Bank Position Are Unclear", and return the MCU to its factory-shipped state.

The flow for programming the initial image using RFP is shown below:

1. Program the initial image using an RFP project for Linear mode while the MCU is in the factory-shipped state.
   Since the setting is switched to "Dual/Bank0 Boot", firmware update using the demo project can be performed.
2. If you need to program the initial image again using RFP, execute "Chip Erase" of “Erase Options” using either: (Note)
   - an RFP project newly created while the firmware has already been updated, or
   - an RFP project configured for “Dual” mode.

   Since the setting is switched back to "Linear" mode and the MCU returns to the factory-shipped state, the initial image can then be programmed.

Note: For details, refer to "RX Family Dual Mode Usage Guide (R01AN6894)".

4.6 Demo Execution Procedure

This chapter describes the execution steps using the demo project for CC-RX as an example. The procedures are also common for the demo project for GCC.

4.6.1 Building Demo Project

4.6.1.1 Creating Initial Image

Set the initial image name as “initial_firm.mot” and explain the creation procedure.

(1) Import both boot_loader and fwup_main projects into e^2 studio and build them.

(2) Make sure that the built MOT file has been generated in each project's HardwareDebug folder and copy it to the DeltaImageGenerator folder. The DeltaImageGenerator folder contains the following file structure.

```
key
tool
delta-image-gen.py
boot_loader.mot
fwup_main.mot
```

(3) Create the initial image using Renesas Image Generator. Run the following command in the DeltaImageGenerator folder.

```
python ./tool/image-gen.py -iup "./fwup_main.mot" -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv -o initial_firm -ibp
"./boot_loader.mot" -vt ecdsa -key "./key/secp256r1.privatekey"
```

A file named “initial_firm.mot” will be generated inside the DeltaImageGenerator folder.

4.6.1.2 Creating Diff Image

The diff image is named “update_firm.delta_rsu”, and the creation procedure is explained.

(1) Rename the “fwup_main.mot” file that be created by building the fwup_main project in “4.6.1.1 Creating Initial Image” to "fwup_main_v1.mot" file.

(2) Open the fwup_main\src\src\fwup_main.h file, change the DEMO_VER_MAJOR definition from (1) to (2), and rebuild the fwup_main project.

```
58  /* FW version definition */
59  #define DEMO_VER_MAJOR          (1)
60  #define DEMO_VER_MINOR          (0)
61  #define DEMO_VER_BUILD          (0)
```

(3) Rename the generated MOT file to “fwup_main_v2.mot” and copy it to the DeltaImageGenerator folder. The DeltaImageGenerator folder contains the following file structure.

```
key
tool
delta-image-gen.py
initial_firm.mot
boot_loader.mot
fwup_main_v1.mot
fwup_main_v2.mot
```

(4) Use the DeltaImageGenerator to create diff images. Run the following command in the DeltaImageGenerator folder.

```
python ./delta-image-gen.py -b ./fwup_main_v1.mot -a ./fwup_main_v2.mot -o
update_firm -bs 128 -key ./key/secp256r1.privatekey -
ip ./tool/RX65N_DualBank_ImageGenerator_PRM.csv
```

The execution log will be output as shown below, and a “update_firm.delta_rsu” file will be generated in the DeltaImageGenerator folder.

```
=== Step 1: Parsing S-record files ===
Parsing: ./fwup_main_v1.mot
  Address range: 0xFE7F5D00 - 0xFFFEFFFF
  Data bytes: 37184
Parsing: ./fwup_main_v2.mot
  Address range: 0xFE7F5D00 - 0xFFFEFFFF
  Data bytes: 37184

=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
Diff blocks: 1
Block size: 128 bytes
Total diff size: 128 bytes
Addresses:
  0xFFF40680 - 0xFFF406FF
Generated diff MOT file: C:\delta_fwup\DeltaImageGenerator\fw_diff.mot

=== Step 3: Generating RSU files ===
Using parameter file:
C:\delta_fwup\DeltaImageGenerator\tool\RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : C:\delta_fwup\DeltaImageGenerator\key\secp256r1.privatekey

Generating RSU for diff data...
Running: C:\Users\taro\.pyenv\pyenv-win\versions\3.12.6\python.exe
C:\delta_fwup\DeltaImageGenerator\tool\image-gen.py -iup
C:\delta_fwup\DeltaImageGenerator\fw_diff.mot -ip
C:\delta_fwup\DeltaImageGenerator\tool\RX65N_DualBank_ImageGenerator_PRM.csv -vt
ecdsa -ff BareMetal -o diff_output -key
C:\delta_fwup\DeltaImageGenerator\key\secp256r1.privatekey


Successfully generated the diff_output.rsu file.


Generating RSU for update firmware...
Running: C:\Users\taro\.pyenv\pyenv-win\versions\3.12.6\python.exe
C:\delta_fwup\DeltaImageGenerator\tool\image-gen.py -iup
C:\delta_fwup\Demos\ccrx\fw\fwup_main_v2.mot -ip
C:\delta_fwup\DeltaImageGenerator\tool\RX65N_DualBank_ImageGenerator_PRM.csv -vt
ecdsa -ff BareMetal -o update_output -key
C:\delta_fwup\DeltaImageGenerator\key\secp256r1.privatekey


Successfully generated the update_output.rsu file.


=== Step 4: Merging RSU files ===
Merged file created: update_firm.delta_rsu
File size: 1152 bytes

=== Complete ===
Output file: update_firm.delta_rsu
Total file size: 1152 bytes
Diff blocks: 1
Total diff size: 128 bytes
```

4.6.1.3 Special Cases where Diff Image File (“update_firm.delta_rsu”) Is Not Generated

When the diff image file is not generated as shown in “1.5 Exception Handling", the execution results of DeltaImageGenerator are shown below. You can check the execution results by looking at the logs.

(1) In case of Differences from Non-FF Block to All-FF Block

When using the DeltaImageGenerator, Step 2 outputs logs for the following All-FF block detections.
A diff file is not generated; instead, an “update_firm.rsu” files (Note) is generated. You can use this file to update FW.
Note: “update_firm.rsu” file: identical to the RSU file generated using RenesasImageGenerator

```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
WARNING: Erased block (all 0xFF) detected at 0xFFFXXXXX - 0xFFFXXXX
Aborting diff generation. Will generate RSU from update firmware directly.

=== Erased block detected: generating RSU from update firmware directly ===
Using parameter file: D:\DeltaImageGenerator\tool\RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : D:\DeltaImageGenerator\key\secp256r1.privatekey

Generating RSU for update firmware...
Running: C:\Program Files\Python313\python.exe D:\DeltaImageGenerator\tool\image-gen.py
-iup D:\DeltaImageGenerator\fwup_main_v2.mot -ip D:\DeltaImageGenerator\tool\RX65N_
DualBank_ImageGenerator_PRM.csv -vt ecdsa -ff BareMetal -o update_output -key
D:\DeltaImageGenerator\key\secp256r1.privatekey
Successfully generated the update_output.rsu file.

=== Complete ===
Output file: update_firm.rsu
Note: Diff generation was aborted due to erased block detection in the update firmware.
The output file contains the entire update firmware RSU.
```

(2) When Number of Diff Blocks between Current Firmware and the Update firmware is Zero (identical)

When using the DeltaImageGenerator, Step 2 outputs the following logs and finishes.
In this case, no updates are necessary.

```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
No differences found between the two firmware files.
```

(3) When Number of Diff Blocks between Current and Update Firmware Is 32 or More

When using the DeltaImageGenerator, the following logs are output.
In Step 2, the following logs of "Diff block: NN" (NN > 31) detections
In Step 3, the following "The number of program data exceeds 31."and the logs of "Error:"

Diff files and “update_firm.rsu” files are not generated.

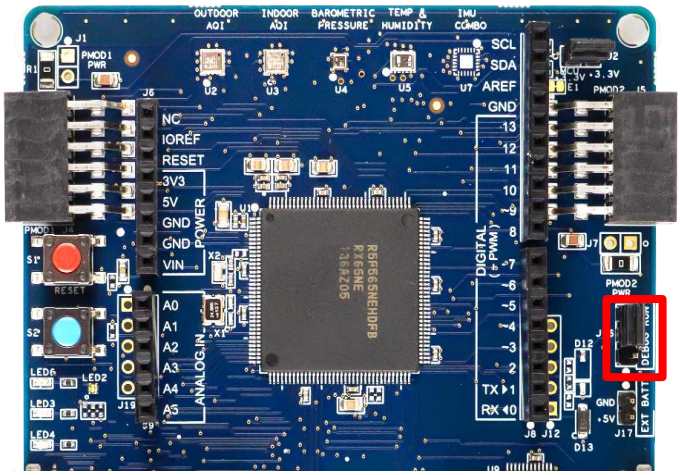
```
=== Step 2: Extracting differences ===
Extracting differences: fwup_main_v2.mot compared to fwup_main_v1.mot...
Diff blocks: 32    <- Detected Number of Blocks
Block size: 128 bytes
Total diff size: XXXX bytes
Addresses:
  0xFFFXXXXX - 0xFFFXXXX
  …

=== Step 3: Generating RSU files ===
Using parameter file: D:\DeltaImageGenerator\tool\RX65N_DualBank_ImageGenerator_PRM.csv
Using private key   : D:\DeltaImageGenerator\key\secp256r1.privatekey

Generating RSU for diff data...
Running: C:\Program Files\Python313\python.exe D:\DeltaImageGenerator\tool\image-gen.py
-iup D:\DeltaImageGenerator\fw_diff.mot -ip D:\DeltaImageGenerator\tool\RX65N_
DualBank_ImageGenerator_PRM.csv -vt ecdsa -ff BareMetal -o diff_output -key
D:\DeltaImageGenerator\key\secp256r1.privatekey
The number of program data exceeds 31.

Error: RSU file not created: D:\DeltaImageGenerator\diff_output.rsu
Traceback (most recent call last):
  File "D:\DeltaImageGenerator\delta-image-gen.py", line 739, in main
    diff_rsu_path = runner.generate_rsu(
        diff_mot_path,
    ... <2 lines>...
        args.ff
    )
  File "D:\DeltaImageGenerator\delta-image-gen.py", line 453, in generate_rsu
    raise RuntimeError(f"RSU file not created: {rsu_path}")
RuntimeError: RSU file not created: D:\DeltaImageGenerator\diff_output.rsu
```

4.6.2 Writing and Executing Initial Image

(1) Write the “initial_firm.mot” created with "4.6.1.1 Creating Initial Image" onto the CK-RX65Nv2 board using a flash writer.

(2) Start Tera Term, select the serial COM port of the USB serial conversion board connected to the CK-RX65Nv2 board, and configure the connection. For the serial communication settings, see “Table 4-1 Serial Communication Specifications”.

Short-circuit the jumper J16 from the CK-RX65Nv2 board to the RUN side (pins 2-3).



(3) The initial image (ver 1.0.0) log is output in Tera Term as shown below. Additionally, LED4 (blue) on the CK-RX65Nv2 board lights up.

```
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]... OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 1.0.0 ====
send image(*.rsu) via UART.
```

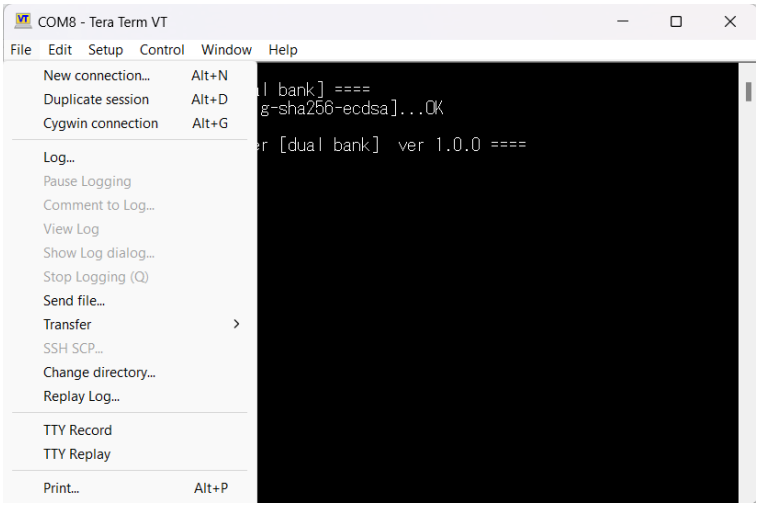
If **"ver 1.0.0" is not displayed** as shown below, it indicates that the system has not transitioned from the "BootLoader" to "Update from User (Main)", and a Bank0/Bank1 mismatch has occurred.

Please refer to Section "4.5.1.3 Installing Flash Programmer" and reprogram initial_firm.mot.

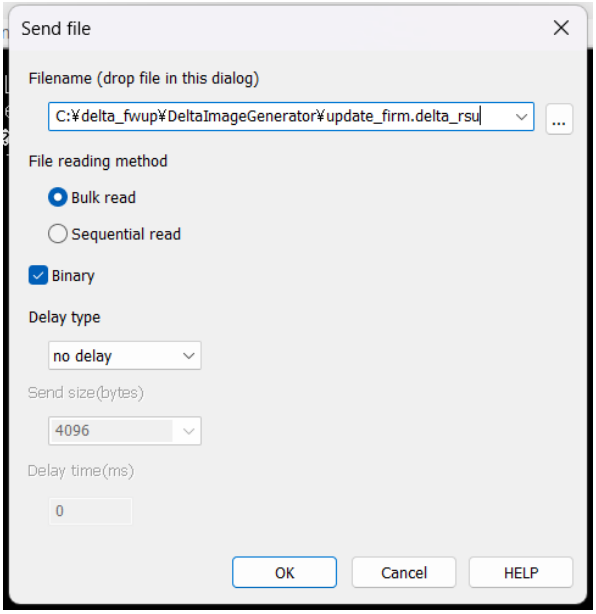
```
==== RX65N : BootLoader [dual bank] ====
send image(*.rsu) via UART.
```

4.6.3 Running a Firmware Update

(1) From the “File” menu in Tera Term, click “Send file…”.



(2) In Filename, Select the diff image file created with "4.6.1.2 Creating Diff Image" and check the "Binary" option and click “OK”.



(3) During the transfer and update process of the update image, progress logs are output as shown below, and once the update process is complete, the software is automatically reset and the firmware for the update image is executed.

(4) If the firmware version output from the logs is "ver 2.0.0", it is successful. Also, the LED 4 (blue) on the CK-RX65Nv2 board changes from lighting up to blinking.

```
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]... OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 1.0.0 ====
send image(*.rsu) via UART.
W 0xFFE00000, 256 ... OK
W 0xFFE00100, 256 ... OK
W 0xFFE00200, 256 ... OK
W 0xFFE40680, 128 ... OK
copy current data ... OK
rewrite addr info ... OK
verify install area buffer [sig-sha256-ecdsa]... OK
bank swap ...
software reset...
==== RX65N : BootLoader [dual bank] ====
verify install area main [sig-sha256-ecdsa]... OK
execute image ..
==== RX65N : Update from User [dual bank]  ver 2.0.0 ====
send image(*.rsu) via UART.
```

That concludes the explanation of the demo execution steps.

4.7 Characteristics of Diff Image Generated in This Demo

Below are the characteristics of the diff image created with "4.6.1.2Creating Diff Image" in this demo.

Table 4-2 Characteristics of Diff Image in This Demo Program

Item	Contents
Behavior changes before and after update	• Changes to the version of logs output (1.0.0 → 2.0.0) • Changed LED lighting pattern (always-on → blinking)
Diff Image File Size	1152 bytes
Number of diff blocks	1
Diff Data Size (Note)	128 bytes

Note: (Diff block size) x (Number of diff blocks)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.03.26	-	First released

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/.