RX671 Group

Example of Program Execution from Serial ROM Using QSPIX XIP Mode

Introduction

This application note describes an example of the use of the XIP mode of the QSPIX module (the QSPIX) on the RX671 Group to execute a program located in the serial ROM.

The following three sample programs are provided with this application note as an example.

- Application program (an application program including program code allocated to the serial ROM)
- Writer program 1 (a program that copies a portion of the application program to the on-chip ROM for writer program 1 and then writes the copied data to the serial ROM)
- Writer program 2 (a program that receives a portion of the application program from the host PC by serial communication and then writes the received data to the serial ROM)

Target Device

RX671 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. XIP Mode and Prefetch Function on QSPIX

In the example described in this application note, the XIP mode and prefetch function of the QSPIX are used when reading instruction codes from the serial ROM.

XIP mode and the prefetch function are described below.

1.1 Overview of XIP Mode

Some serial ROM supports the speeding up of ROM read operations by omitting the reception of instruction codes. This function is controlled by mode data contained in dummy cycles within the preceding SPI bus cycle.

1.2 Enabling XIP Mode

XIP mode is available for use in memory map mode (AMOD bit in SPMR1 register set to 1). To enable XIP mode, specify the value for activating the XIP mode of the serial ROM to be used in the MODE[7:0] bits in the SPDCR register and set the XIPE bit in the SPDCR register to 1. The value specified in the MODE[7:0] bits is transmitted during the dummy cycles in the next SPI bus cycle, as shown in Figure 1 illustrating XIP mode control data. It is possible to determine if XIP mode is enabled by reading the XIPS flag in the SPDCR register after the above SPI bus cycle ends. Note that the mode data for activating XIP mode differs according to the type of serial ROM. It is therefore necessary to set the MODE[7:0] bits to match the serial ROM to be used.

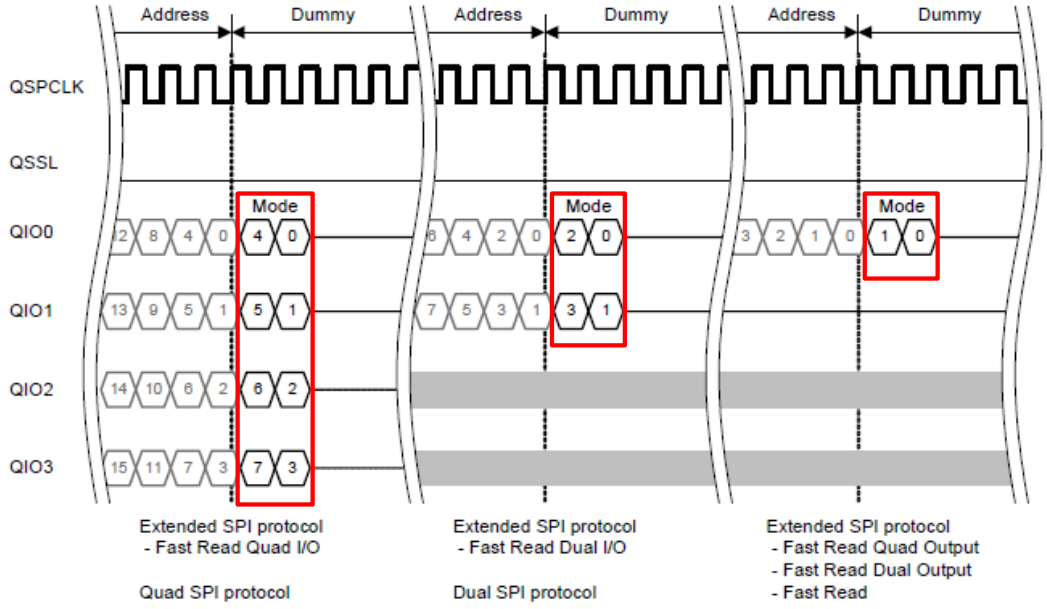


Figure 1 XIP Mode Control Data

1.3 Terminating XIP Mode

To terminate XIP mode, specify the value for terminating the XIP mode of the serial ROM used in the MODE[7:0] bits in the SPDCR register and clear the XIPE bit in the SPDCR register to 0. The value specified in the MODE[7:0] bits is transmitted during the dummy cycles in the next SPI bus cycle.

It is possible to determine if XIP mode has been terminated by reading the XIPS flag in the SPDCR register after the above SPI bus cycle ends.

1.4 Prefetch Function

The QSPIX has a prefetch function.

The serial ROM memory read command can be used to read an infinite amount of data in a single SPI bus cycle. However, if bus cycles issued by the CPU are individually converted into SPI bus cycles, the SPI bus cycle is divided and this feature of the serial ROM cannot be utilized.

The prefetch function makes use of this characteristic to speed up instruction execution.

The prefetch function is enabled when the PFE bit in the SPMR0 register is set to 1. When the prefetch function is enabled, the QSPIX continuously receives and buffers data without waiting for the next ROM read request. When the CPU next reads the ROM, the QSPIX compares the addresses and returns the data in the buffer to the CPU if the addresses match. If the addresses do not match, the data in the buffer is discarded and a new SPI bus cycle is generated.

2. Hardware Configuration

Figure 2 is diagram of the connections between the RX671 MCU mounted on the Renesas Starter Kit+ for RX671 (RSK) board and the serial ROM.

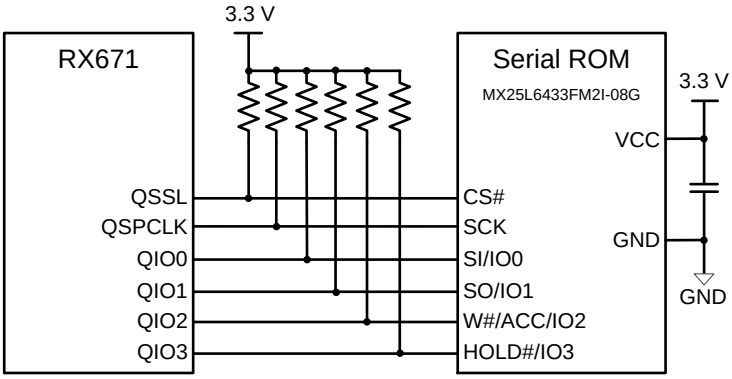


Figure 2 Connection Diagram of RX671 Mounted on Renesas Starter Kit+ for RX671 Board and Serial ROM

Writer program 2 receives a portion of the application program from the host PC by serial communication. Figure 3 shows "Diagram of Connection Between the RX671 and the Host PC".

The RSK has a USB serial conversion circuit. If the RSK is connected to the host PC by USB connection, the RSK can work as a virtual COM port, which can be used to send data to, and receive data from, the RX671 by serial communication.

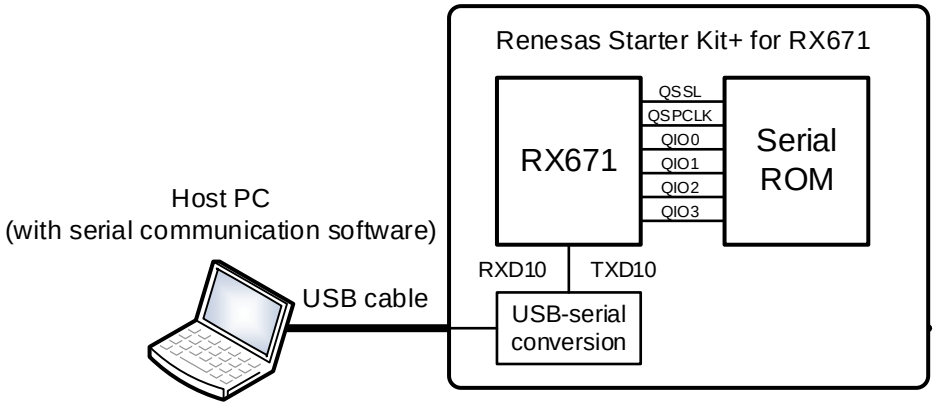


Figure 3 Diagram of Connection Between the RX671 and the Host PC

Table 2.1 lists the QSPIX pins that are used for connection between the RX671 and serial ROM.

Table 2.1 QSPIX Pins Used for Connection Between the RX671 and Serial ROM

Pin Name	I/O	Function
QSSL	Output	Slave select pin
QSPCLK	Output	Clock output pin
QIO0	Input/Output	Data 0 I/O
QIO1	Input/Output	Data 1 I/O
QIO2	Input/Output	Data 2 I/O
QIO3	Input/Output	Data 3 I/O

Table 2.2 lists the SCI pins that are used for connection between the RX671 and host PC.

Table 2.2 SCI Pins Used for Connection Between the RX671 and Host PC

Pin Name	I/O	Function
RXD10	Input	Receive data input pin
TXD10	Output	Send data output pin

The application program controls the LEDs mounted on the RSK board.

Table 2.3 lists the pins used for LED control.

Table 2.3 Pins Used for LED Control

Pin Name	Function
P17	LED0 control
PF5	LED1 control
P03	LED2 control
P05	LED3 control

3. Sample Programs

The sample programs described in this application note use the serial ROM module (Macronix MX25L6433FM2I-08G) mounted on the RSK board as external memory and use the QSPIX's XIP mode to read program code stored in the serial ROM.

The following three sample programs are provided.

Table 3.1 Sample Programs

Project Name	Description
xip_sample_rx671	Application program An application program including program code allocated to the serial ROM
serialROM_write1_direct_rx671	Writer program 1 A program that copies a portion of the application program to the on-chip ROM and then writes the copied data to the serial ROM
serialROM_write2_serial_rx671	Writer program 2 A program that receives a portion of the application program from the host PC by serial communication and then writes the received data to the serial ROM

The sample programs use e^2 studio and Smart Configurator (SC) as an integrated development environment. In addition, Firmware Integration Technology (FIT) modules are used as programs for configuring the settings of and controlling peripheral functions.

For details of the FIT modules and settings used, refer to 3.3, FIT Modules Used.

3.1 Application Program

3.1.1 Program Specifications

3.1.1.1 Software

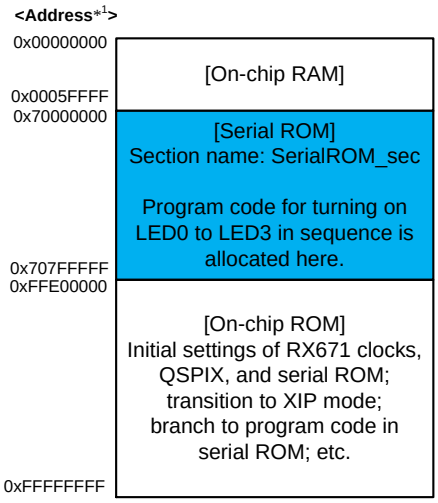
The application program is divided between the MCU's on-chip ROM and the serial ROM.

For a program that needs to run at a high speed, it is recommended that you allocate the program on the on-chip ROM. For a program that does not need to run at a high speed, it is recommended that you allocate the program on the serial ROM.

For the files generated when an application program is built, the files for the programs that are to be allocated to the on-chip ROM and the files for the programs that are to be allocated to the serial ROM are output separately.

(1) Address Allocation of Application Program

Figure 4 shows the address allocation of the application program. The program code allocated to the on-chip ROM includes the processing for configuring initial settings for the RX671's clocks, the QSPIX, and the serial ROM; for transitioning to XIP mode; and for branching to the program code located in the serial ROM. The program code allocated to the serial ROM turns on LED0 to LED3 on the RSK board in sequence. The section name of the portion of the program allocated to the serial ROM is SerialROM_sec.



Note: 1. The on-chip ROM and on-chip RAM addresses assume the use of products with a ROM/RAM capacity of 2 MB/384 KB. In addition, the serial ROM addresses assume a capacity of 8 MB.

Figure 4 Address Allocation of Application Program

(2) Divided Output of Files Generated when Building Application Program

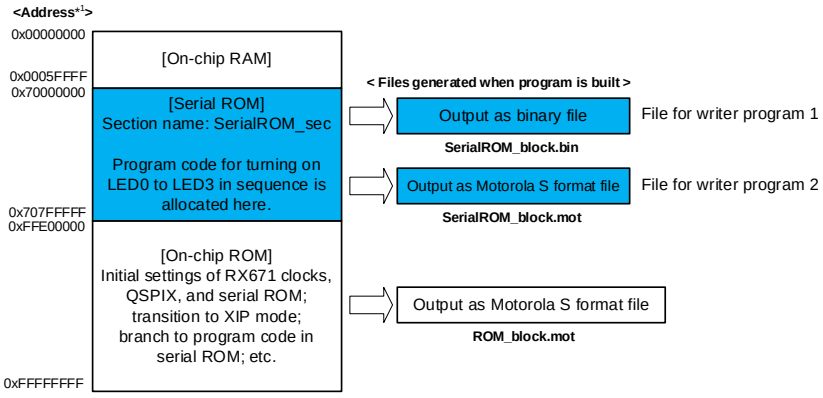
Figure 5 illustrates the divided output of the files generated the application program is built.

As shown in this figure, a program to be allocated to the on-chip ROM is output in only Motorola S format (ROM_block.mot).

A program to be allocated to the serial ROM is output in both binary and Motorola S formats.

The file in binary format (SerialROM_block.bin) is used for writer program 1.

The file in Motorola S format (SerialROM_block.mot) is used for writer program 2.



Note: 1. The on-chip ROM and on-chip RAM addresses assume the use of products with a ROM/RAM capacity of 2 MB/384 KB. In addition, the serial ROM addresses assume a capacity of 8 MB.

Figure 5 Divided Output of Files Generated when Building Application Program

3.1.1.2 Build Settings in e² studio

The option settings that need to be configured in e² studio are described below.

(1) Section Allocation of Program Code to be Assigned to Serial ROM

Proceed as follows to perform section allocation of the portion of program to be located in serial ROM.

Open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Linker** → **Section** to display the window shown in Figure 6, Section Allocation of Program Code to be Located in Serial ROM (1/2).

Click the [**…**] button to the right of **Sections (-start)**.

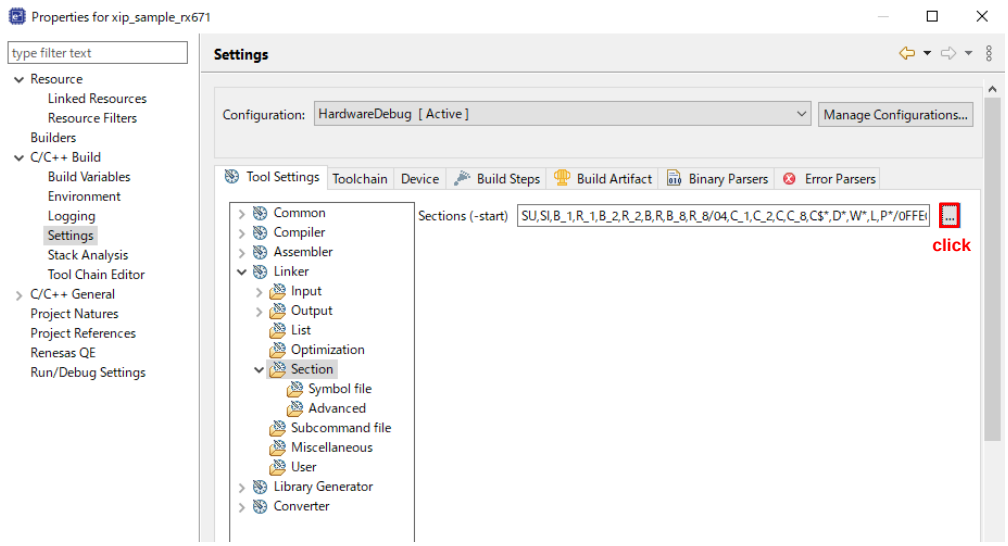


Figure 6 Section Allocation of Program Code to be Located in Serial ROM (1/2)

Next, as shown in Figure 7, click the **Add Section** button in **Section Viewer** and add the address and section name for the portion of the program to be allocated to the serial ROM.

For this application program, make sure that the following address is set:

Address: 0x70000000 (start address of QSPI area in serial ROM)

When you use writer program 1, do not change the preceding address.

When you use writer program 2, set a QSPI area address that is a multiple of 256 (the last byte is 0x00).

Section Name: SerialROM_sec

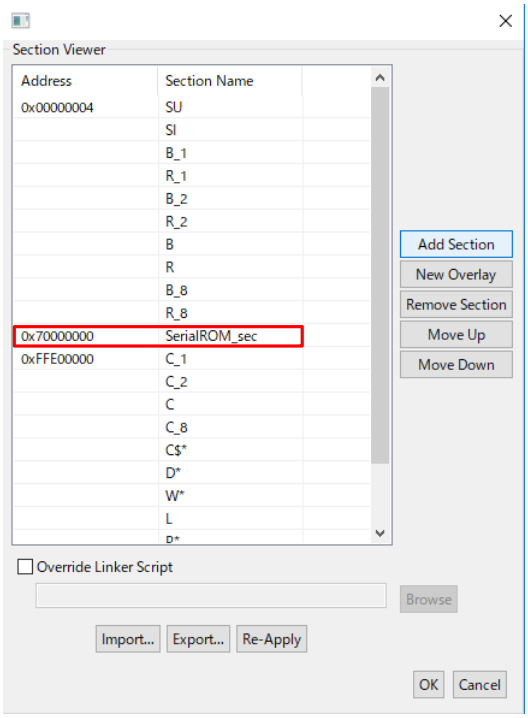


Figure 7 Section Allocation of Program Code to be Located in Serial ROM (2/2)

(2) Divided Output of Files Generated when Building Application Program

Open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Converter** → **Output** to display the window shown in Figure 8, Divided Output of Files Generated when Building Application Program in e2 studio (1/2).

Check the boxes next to **Motorola S format file (-form=stype)** and **Binary file (-form=binary)**.

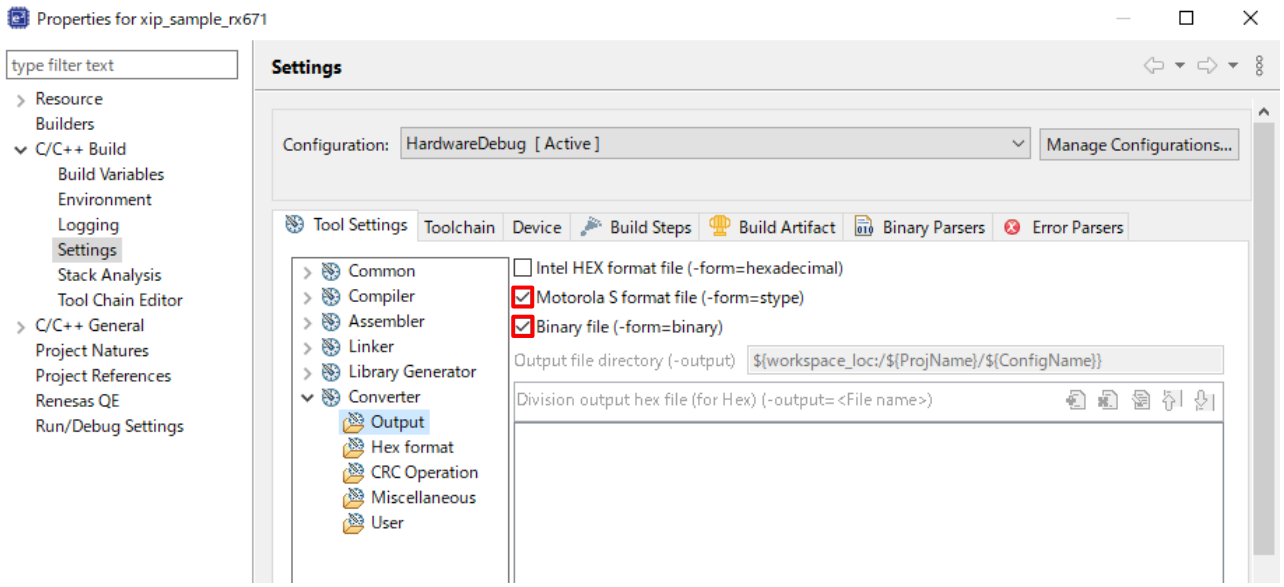


Figure 8 Divided Output of Files Generated when Building Application Program in e² studio (1/2)

Next, as shown in Figure 9, Divided Output of Files Generated when Building Application Program in e2 studio (2/2), use the scroll bar on the right to scroll down to the bottom, click the **Add** button next to **Division output mot (for Stype) (-output=<File name>)**, and add the value **ROM_block.mot=ffe00000-ffffffff** and the value **SerialROM_block.mot=SerialROM_sec**.

Next, click the **Add** button next to **Division output bin file (for Bin) (-output=<File name>)**, and add the value **SerialROM_block.bin=SerialROM_sec**.

Note that "SerialROM_sec" here indicates the section name of the program to be allocated to the serial ROM as described in "3.1.1.2(1) Section Allocation of Program Code to be Assigned to Serial ROM".

Click the **Apply and Close** button to close the project's **Properties** window. (It is not necessary to close the window if you wish to configure additional settings.)

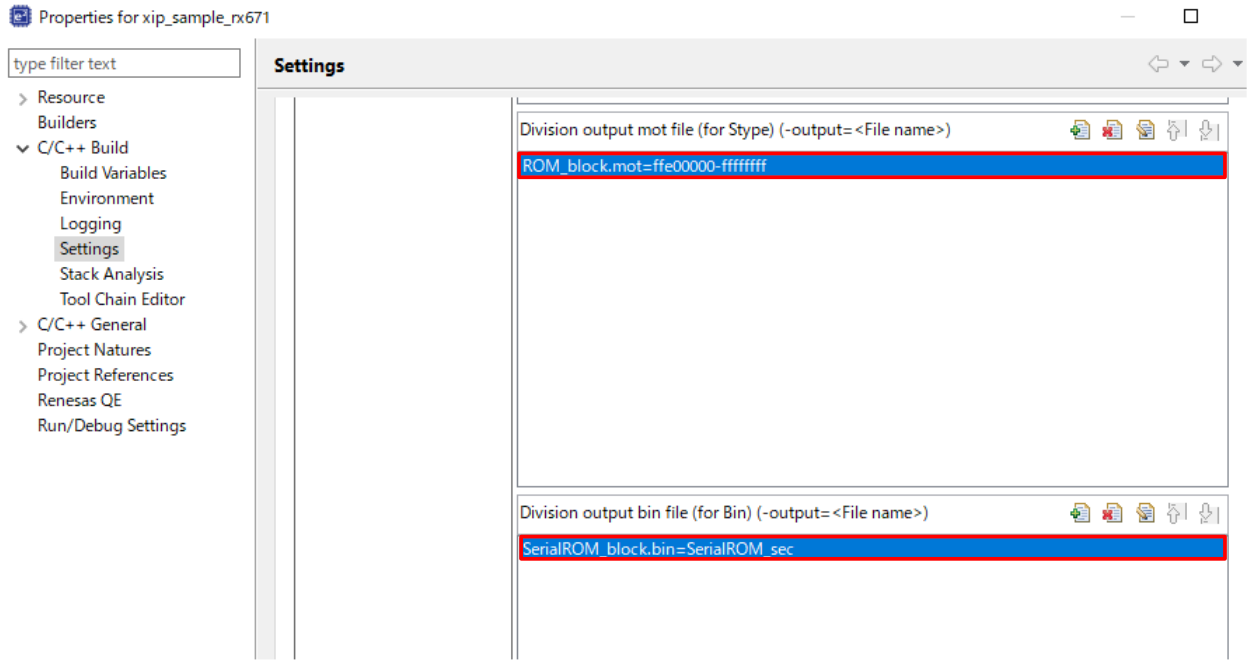


Figure 9 Divided Output of Files Generated when Building Application Program in e² studio (2/2)

(3) Branch width size (-branch) Option Setting

The address range of the QSPI area to be allocated in the serial ROM is 0x70000000 to 0x77FFFFFF, so a 24-bit branch width is insufficient for branching the program from the on-chip ROM to the serial ROM. It is therefore necessary to change the setting of the **Branch width size (-branch)** option.

To configure the **Branch width size (-branch)** option, open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Common** → **CPU** to display the window shown in Figure 10, Branch width size (-branch) Option Setting.

On the **Branch width size (-branch)** pulldown menu select **None**. Click the **Apply and Close** button to close the project's **Properties** window. (It is not necessary to close the window if you wish to configure additional settings.)

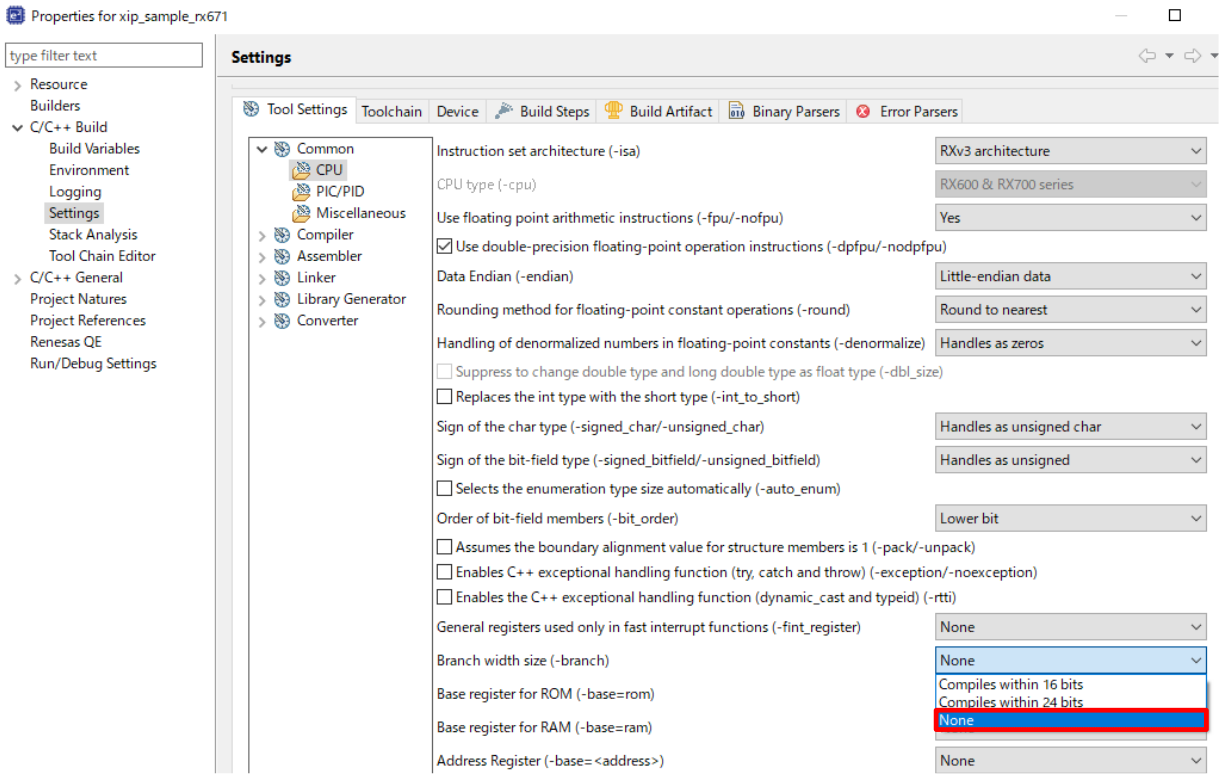


Figure 10 Branch width size (-branch) Option Setting

(4) Checks the section larger than the specified range of addresses (-cpu) Option Setting

If the default -cpu option setting is used, an error occurs when assigning the program to the QSPI area (address range 0x70000000 to 0x77FFFFFF) allocated in the serial ROM.

This is because the -cpu option enables checking of the address ranges to which sections are assigned, causing the QSPI area to be judged as outside the specified range of addresses.

The application program requires that **Checks the section larger than the specified range of addresses (-cpu)** be unselected.

Open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Linker** → **Section** → **Advanced** to display the window shown in Figure 11, Checks the section larger than the specified range of addresses (-cpu) Option Setting.

Uncheck the box next to **Checks the section larger than the specified range of addresses (-cpu)**.

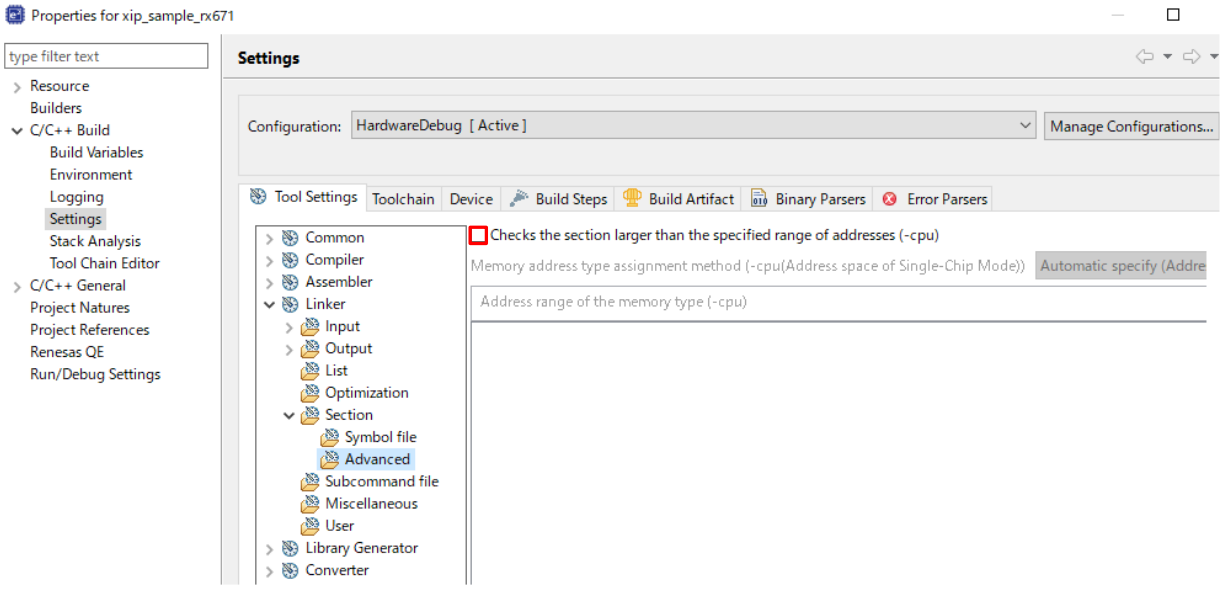


Figure 11 Checks the section larger than the specified range of addresses (-cpu) Option Setting

Alternatively, you can check the box next to **Checks the section larger than the specified range of addresses (-cpu)** and add the QSPI area to the -cpu option.

For instructions on configuring -cpu option settings, refer to CC-RX Compiler User's Manual (R20UT3248).

3.1.1.3 Outline Flowchart

Figure 12 shows an outline flowchart of the application program (to be executed on the on-chip ROM).

Figure 13 shows an outline flowchart of the application program (to be executed on the serial ROM).

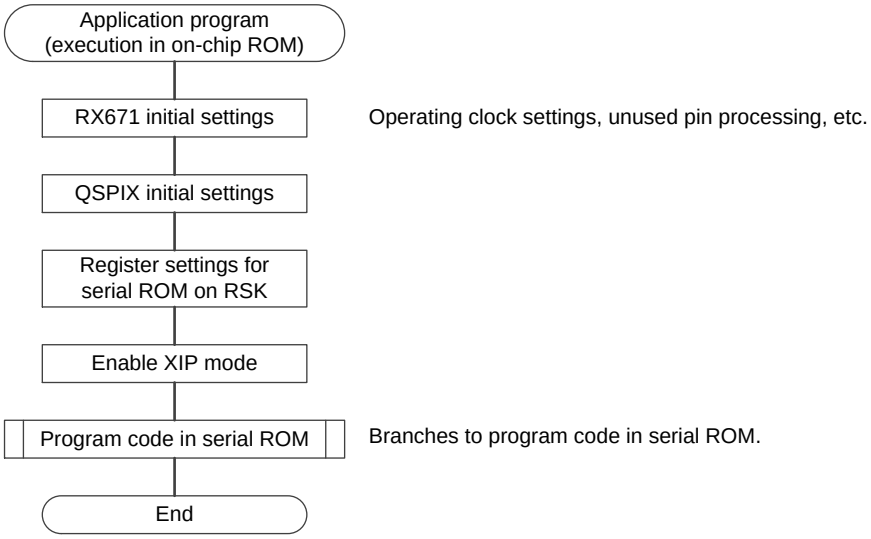
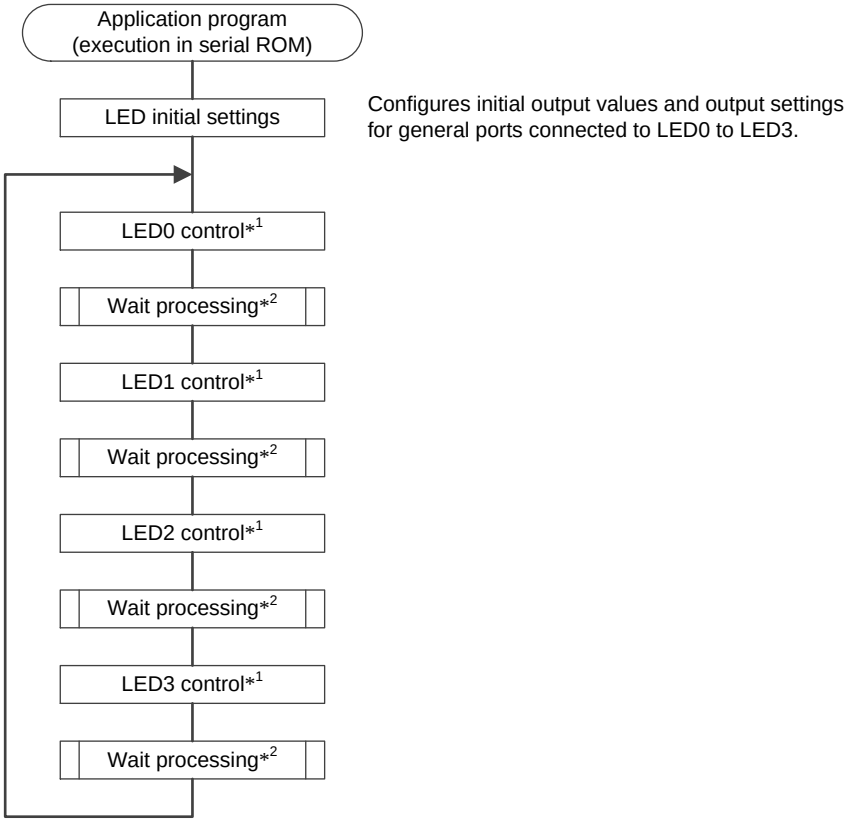


Figure 12 Outline Flowchart of the Application Program (to Be Executed on the On-chip ROM)



Notes: 1. The LEDs are toggled on/off each time the loop executes.
2. A software loop is used for wait processing, and the wait duration is approximately 0.5 seconds.

Figure 13 Outline Flowchart of the Application Program (to Be Executed on the Serial ROM)

3.1.2 Program Configuration

3.1.2.1 File Structure

The files used by the application program are listed below. Note that FIT module files and files generated automatically by SC are omitted.

Table 3.2 Files Used by Application Program

File Name	Overview
main.c	This is the main processing of the application program. Initializes the QSPIX and the serial ROM status register, transitions to XIP mode, and branches to the portion of the program in the serial ROM.
main_serial_rom.c	Program code located in the serial ROM
serial_rom.h	Serial ROM control command definitions

3.1.2.2 Option-Setting Memory

The option-setting memory setting used by the application program is shown below.

Table 3.3 Option-Setting Memory Setting Used by Application Program

Symbol	Address	Setting Value	Description
MDE	FE7F 5D00h to FE7F 5D03h	FFFF FFFFh	Little endian

3.1.2.3 Constants

The constants used by the application program are listed below.

Table 3.4 Constants Used by Application Program

Constant Name	Setting value	Description
LED_ON	(0)	LED on
LED_OFF	(1)	LED off
LED0	PORT1.PODR.BIT.B7	LED0 port output data storage bit
LED1	PORTF.PODR.BIT.B5	LED1 port output data storage bit
LED2	PORT0.PODR.BIT.B3	LED2 port output data storage bit
LED3	PORT0.PODR.BIT.B5	LED3 port output data storage bit
LED0_PDR	PORT1.PDR.BIT.B7	LED0 port direction control bit
LED1_PDR	PORTF.PDR.BIT.B5	LED1 port direction control bit
LED2_PDR	PORT0.PDR.BIT.B3	LED2 port direction control bit
LED3_PDR	PORT0.PDR.BIT.B5	LED3 port direction control bit
LED_INTERVAL	(0x16000)	Sets LED-on duration to 0.5 seconds.
CMD_WREN	(0x06)	Write Enable (WREN) command for serial ROM
CMD_WRSR	(0x01)	Write Status Register (WRSR) command for serial ROM
CMD_RDSR	(0x05)	Read Status Register (RDSR) command for serial ROM
SERIALROM_ENTER_QSPI_MODE	(0x40)	Serial ROM status register setting data (Quad mode enabled setting)
SERIALROM_CONFIG_REG	(0x00)	Serial ROM configuration register setting data

3.1.2.4 Functions

The functions of the application program are listed below.

Table 3.5 Functions of Application Program

Function Name	Overview
main	Main processing Initializes the QSPIX and the serial ROM status register; transitions to XIP mode; and branches to the program on the serial ROM.
rom_access_error_callback	Callback function for the QSPIX FIT module Performs confirmation when a ROM access error interrupt occurs.
main_serial_rom	Program code allocated to serial ROM Turns on LEDs 0 to 3 on the RSK in sequence.
led_wait	Software wait processing to maintain LED-on interval (wait time: approximately 0.5 seconds)

3.2 Writer Program

For this application note, two simple writer programs are provided. These programs are designed only to write data to the serial ROM installed on the RSK (Macronix MX25L6433FM2I-08G) for operation verification.

One writer program (writer program 1) copies a portion of the application program to the on-chip ROM and then writes the copied data to the serial ROM.

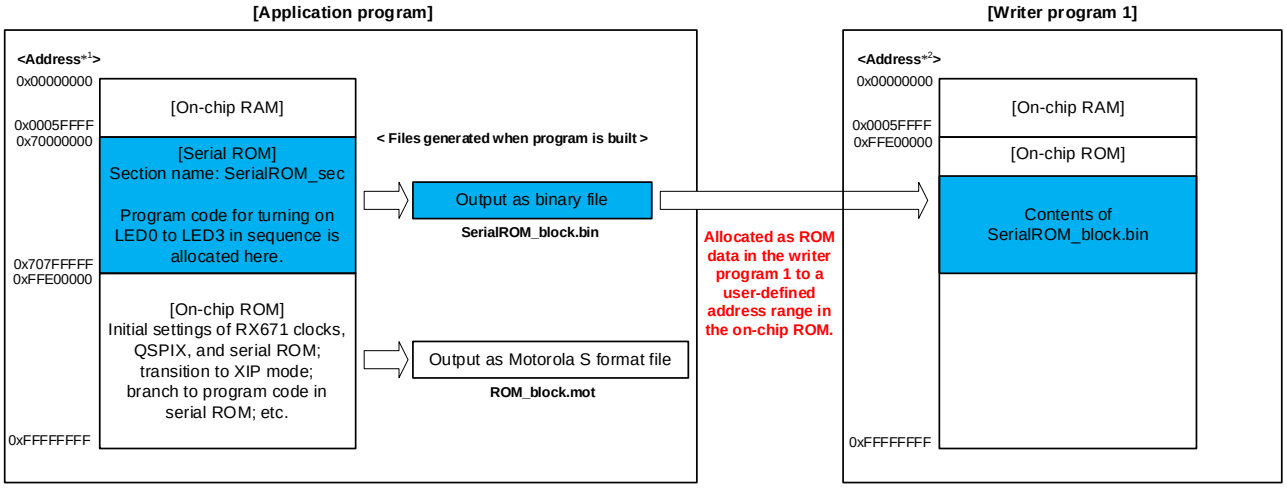
The other writer program (writer program 2) receives a portion of the application program from the host PC by serial communication and then writes the received data to the serial ROM.

3.2.1 Writer Program 1

3.2.1.1 Program Specifications

(1) Software

Figure 14 illustrates binary file input and address allocation by the writer program 1. As shown in the figure, a binary file (SerialROM_block.bin) resulting from divided output of the application program is input to the writer program 1. Then the input binary file is allocated as ROM data to a user-defined address range in the on-chip ROM. (The supplied writer program 1 allocates the binary file to a start address in the on-chip ROM.)



Notes: 1. The on-chip ROM and on-chip RAM addresses assume the use of products with a ROM/RAM capacity of 2 MB/384 KB. In addition, the serial ROM addresses assume a capacity of 8 MB.
2. On-chip ROM and on-chip RAM addresses for products with a ROM/RAM capacity of 2 MB/384 KB.

Figure 14 Binary File Input and Address Allocation by Writer Program 1

The writer program 1 uses LED0 to LED3 on the RSK board to indicate progress. It turns on LED0 when erasing of data in the first block in the serial ROM finishes, turns on LED1 when writing of data to the erased first block in the serial ROM finishes, and turns on LED2 when verification checking of the data written to the serial ROM finishes. If any of the above processes fails, LED3 turns on.

(2) Build Settings in e² studio

The option settings that need to be configured in e² studio are described below.

(a) Section Allocation of Addresses to be Assigned to Binary Data

Proceed as follows to perform section allocation of addresses to be assigned to binary data as preparation for allocating the input binary file to a user-defined address in the on-chip ROM as ROM data.

Open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Linker** → **Section** to display the window shown in Figure 15, Section Allocation of Addresses to be Assigned to Input Binary File (1/2).

Click the [**…**] button to the right of **Sections (-start)**.

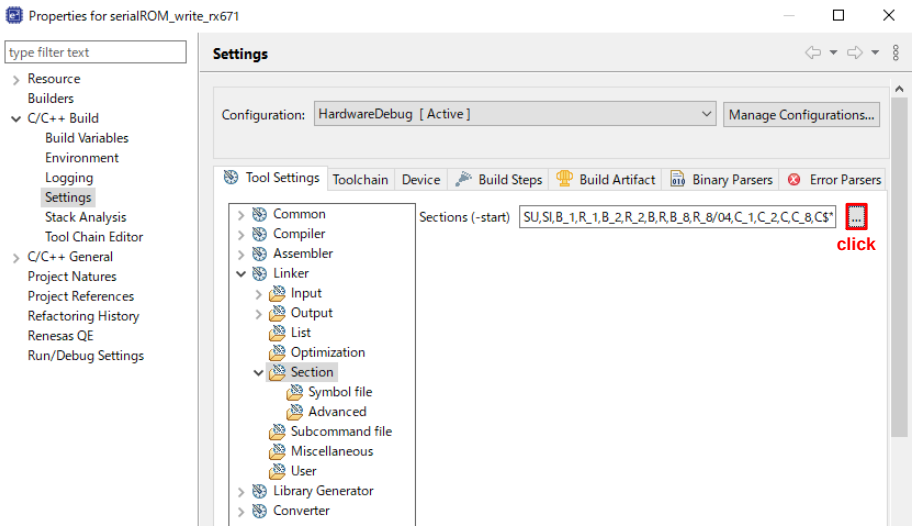


Figure 15 Section Allocation of Addresses to be Assigned to Input Binary File (1/2)

Next, as shown in Figure 16, click the **Add Section** button in **Section Viewer** to add a section at a user-defined address in the on-chip ROM. For the writer program 1, these values are as follows:

Address: 0xFFE00000 (start address in on-chip ROM)

Section Name: SerialROM_WriteData_sec

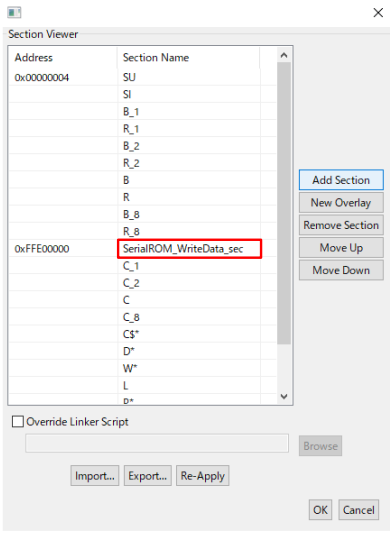


Figure 16 Section Allocation of Addresses to be Assigned to Input Binary File (2/2)

(b) Input Binary File Specification (-binary) Option Setting

The -binary option is used to input the binary file generated by the application program.

Open the project's **Properties** window, click **C/C++ Build** → **Settings**, and select **Tool Settings** from among the tabs displayed at right. Then select **Linker** → **User** to display the window shown in Figure 17, Binary File Input Setting.

Click the **Add** button next to **User-defined options (added after all specified options)** and add the -binary option setting.

The setting for the writer program 1 is as follows:

-binary="${WorkspaceDirPath}/xip_sample_rx671/HardwareDebug/SerialROM_block.bin" (SerialROM_WriteData_sec:4/DATA)

Note: The above setting is split into two lines for explanatory purposes in this document, but the actual option setting should not contain a line break.

This setting will cause the input binary file to be assigned as follows:

Assigned section: SeriarlROM_WriteData_sec section (boundary alignment: 4)

Assigned section property: DATA

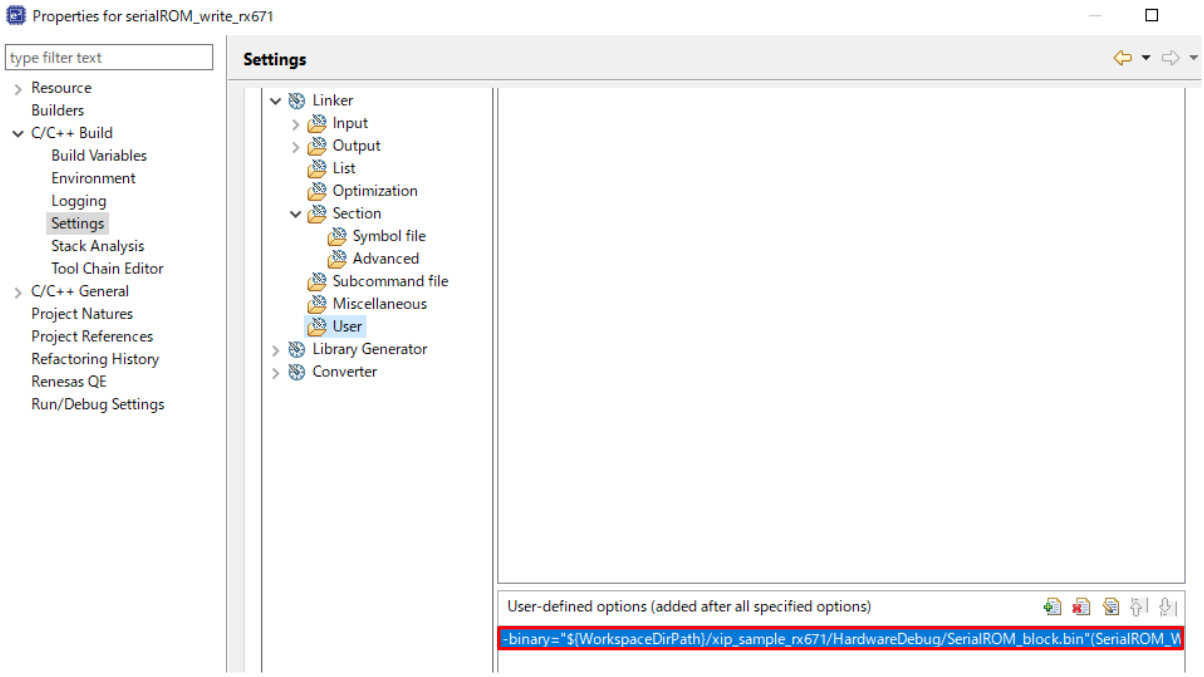


Figure 17 Binary File Input Setting

(3) Outline Flowchart

Figure 18 is an outline flowchart of the writer program 1.

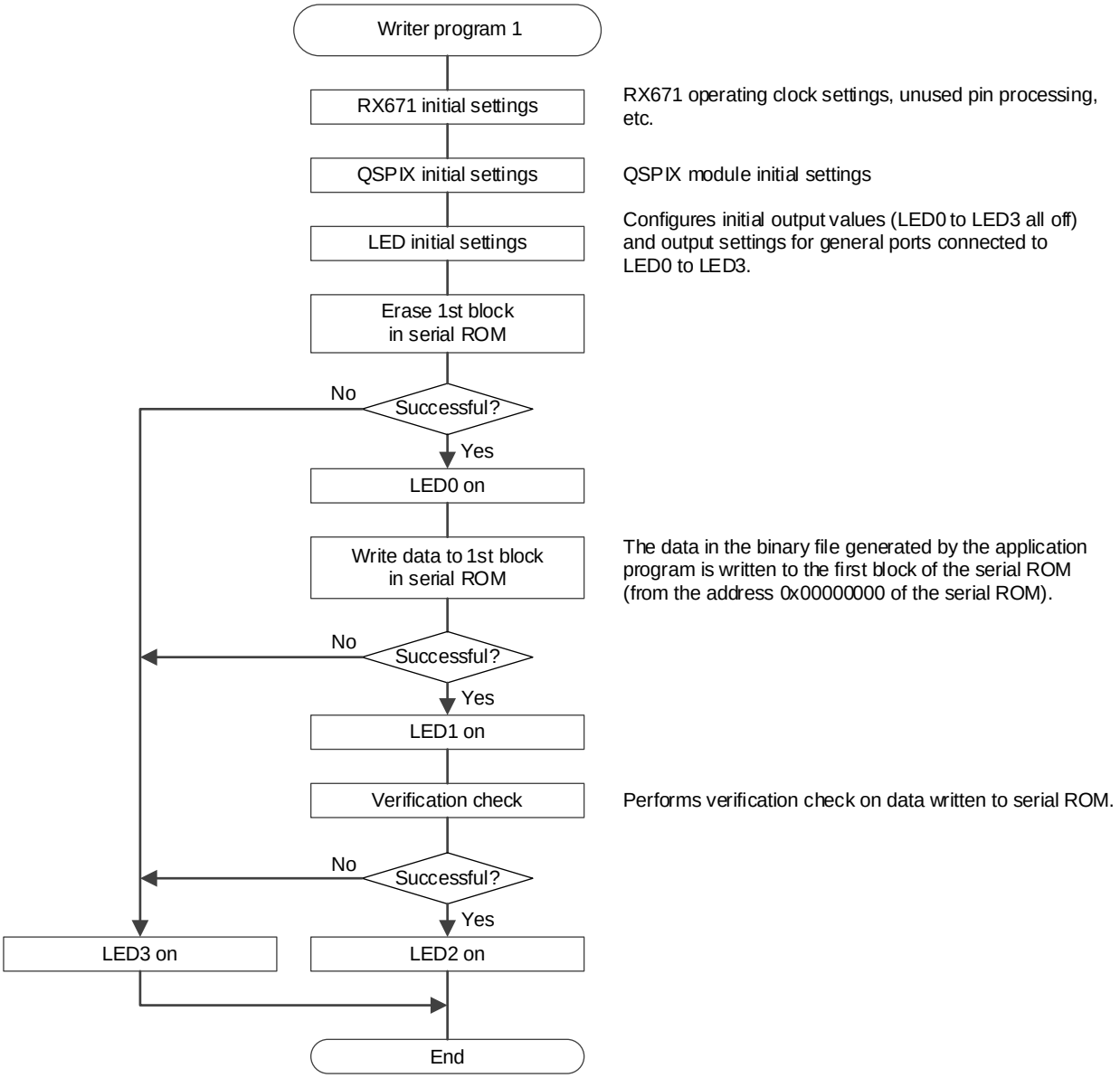


Figure 18 Outline Flowchart of Writer Program 1

3.2.1.2 Program Configuration

(1) File Structure

The files used by the writer program 1 are listed below. Note that FIT module files and files generated automatically by SC are omitted.

Table 3.6 Files Used by Writer Program 1

File Name	Overview
serial_rom_write1_direct_rx671.c	This is the main processing of the writer program 1. Initializes the QSPIX and the serial ROM status register; performs block erase, writes data, and verifies the data written to the serial ROM.
serial_rom.h	Serial ROM control command definitions

(2) Option-Setting Memory

The option-setting memory setting used by the writer program 1 is shown below.

Table 3.7 Option-Setting Memory Setting Used by Writer Program 1

Symbol	Address	Setting Value	Description
MDE	FE7F 5D00h to FE7F 5D03h	FFFF FFFFh	Little endian

(3) Constants

The constants used by the writer program 1 are listed below.

Table 3.8 Constants Used by Writer Program 1

Constant Name	Setting value	Description
LED_ON	(0)	LED on
LED_OFF	(1)	LED off
LED0	PORT1.PODR.BIT.B7	LED0 port output data storage bit
LED1	PORTF.PODR.BIT.B5	LED1 port output data storage bit
LED2	PORT0.PODR.BIT.B3	LED2 port output data storage bit
LED3	PORT0.PODR.BIT.B5	LED3 port output data storage bit
LED0_PDR	PORT1.PDR.BIT.B7	LED0 port direction control bit
LED1_PDR	PORTF.PDR.BIT.B5	LED1 port direction control bit
LED2_PDR	PORT0.PDR.BIT.B3	LED2 port direction control bit
LED3_PDR	PORT0.PDR.BIT.B5	LED3 port direction control bit
CMD_WREN	(0x06)	Write Enable (WREN) command for serial ROM
CMD_WRSR	(0x01)	Write Status Register (WRSR) command for serial ROM
CMD_RDSR	(0x05)	Read Status Register (RDSR) command for serial ROM
CMD_RDSCUR	(0x2B)	Read Security Register (RDSCUR) command for serial ROM
CMD_BE	(0x52)	Block Erase (BE) command for serial ROM
CMD_PP	(0x02)	Page Program (PP) command for serial ROM
SERIALROM_EXIT_QSPI_MODE	(0x00)	Serial ROM status register setting data (Quad mode disabled setting)
SERIALROM_CONFIG_REG	(0x00)	Serial ROM configuration register setting data

(4) Functions

The functions of the writer program 1 are listed below.

Table 3.9 Functions of Writer Program 1

Function Name	Overview
main	Main processing Initializes the QSPIX and the serial ROM status register; performs block erase of the serial ROM; programs the serial ROM; and verifies the data written to serial ROM.
rom_access_error_callback	Callback function for the QSPIX FIT module Performs confirmation when a ROM access error interrupt occurs.
write_data_func	Processing of writing data to serial ROM

3.2.2 Writer Program 2

3.2.2.1 Program Specifications

(1) Software

Writer program 2 uses the terminal software of the host PC to receive the SerialROM_block.mot file by serial communication (via the XMODEM/SUM protocol) and then writes the file to the serial ROM.

Note that the SerialROM_block.mot file here is the SerialROM_block.mot file that was generated in Motorola S format when the program (in the application program) to be allocated to the serial ROM is built.

For details, see "3.1.1.1(2) Divided Output of Files Generated when Building Application Program".

Table 3.10 shows the specifications of serial communication between the RX671 and host PC. For details about how to set up the terminal software, see the documentation for the terminal software.

Table 3.10 Specifications of Serial Communication Between the RX671 and Host PC

Item	Description
Communication method	Asynchronous communication
Communication protocol	XMODEM/SUM
Bit rate	115200 bps
Data length	8 bits
Parity	None
Stop bit	1 bit
Flow control	None

(2) Outline flowchart

Figure 19 shows Outline Flowchart for the Main Processing of Writer Program 2.

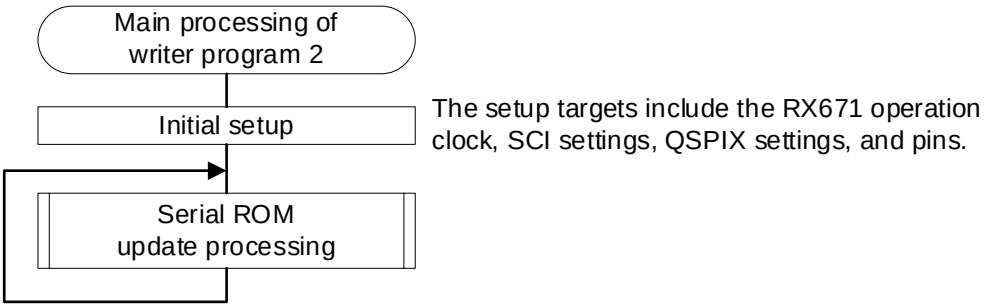
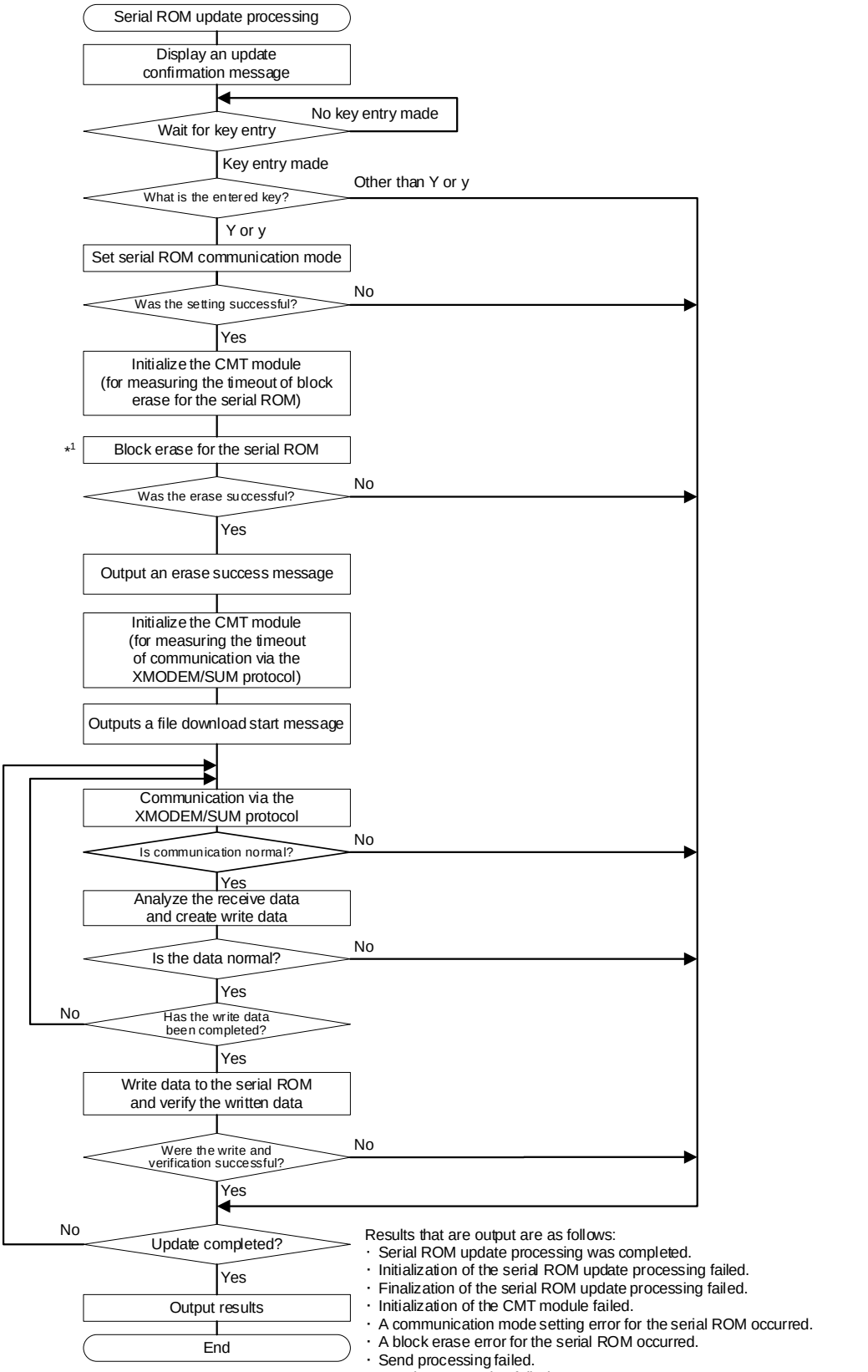


Figure 19 Outline Flowchart for the Main Processing of Writer Program 2

Figure 20 shows Outline Flowchart for the Processing to Update the Serial ROM for Writer Program 2.



Note 1:
Block erase is performed from the first block of the serial ROM. The number of blocks to be erased is set by using the SEL_ERASE_BLOCK_NUM constant in the serial_rom_config.h file.

Figure 20 Outline Flowchart for the Processing to Update the Serial ROM for Writer Program 2

(3) Screen Output of the Terminal Software and Operation of Writer Program 2

(a) Confirmation for the Update

Writer program 2 first performs initial setup for the RX671 operation clock, SCI, QSPIX, pins, etc., as the main processing, and then uses the SCI to output a message (Figure 21) to the terminal software of the host PC. Writer program 2 then waits for key entry from the terminal software.

```
RX671 Serial ROM Update ver1.00
Erase and write (Y/N)?
```

Figure 21 Update Confirmation Message Output on the Screen

(b) Start of Downloading the SerialROM_block.mot File

If writer program 2 receives Y or y from the terminal software, it performs block erase for the serial ROM, waits for the file to be sent, and outputs the message shown in Figure 22.

Send the .mot file (the SerialROM_block.mot file generated when the application program was built) from the terminal software via the XMODEM/SUM protocol.

For details about how to send the file from terminal software via the XMODEM/SUM protocol, see the documentation for the terminal software.

```
Erasing has been done.
Start XMODEM download…
```

Figure 22 Message Output When File Download Starts

(c) Completion of Updating the Serial ROM

When the write to the serial ROM is completed, the message shown in Figure 23 is output.

```
Updating has been done.
>
```

Figure 23 Message Output When the Update of the Serial ROM Is Completed

(d) Error Messages That Can Be Output

If an error occurs, a message about the error is output. Table 3.11 lists the messages that can be output.

Table 3.11 Error Messages

Error Message	Description
Initialize update error.	Initialization of the update processing failed.
Finalize update error.	Finalization of the update processing failed.
CMT module error.	Initialization of the CMT module failed.
Serial ROM mode setting error.	A communication mode setting error for the serial ROM occurred.
Serial ROM Erasing error.	A block erase error for the serial ROM occurred.
Send error	Send processing failed.
Receive error.	Receive processing failed.
Timeout.	A timeout occurred in communication via the XMODEM/SUM protocol.
Data error.	A data error occurred in communication via the XMODEM/SUM protocol.
Block processing error.	A data analysis error or serial ROM write error occurred.

(e) Cancellation of the Update

If writer program 2 receives a command other than Y or y in "(a) Confirmation for the Update", it outputs the message shown in Figure 24 and cancels the update.

```
Command canceled.
>
```

Figure 24 Message Output When the Update Is Canceled

3.2.2.2 Program Configuration

(1) File Configuration

The following table lists the files used by writer program 2. Note that this list does not include files that are automatically generated by the FIT module and SC.

Table 3.12 Files Used by Writer Program 2

File Name	Summary
serial_rom_write2_serial_rx671.c	Main processing of writer program 2 Performs processing such as initialization of the SCI and QSPIX, output of an update confirmation message, and invocation of update processing.
r_xmodem.c	Processing of XMODEM/SUM communication
r_xmodem_if.h	Interface file for the processing of XMODEM/SUM communication
r_fw_up_rx.c	Processing of updating the serial ROM
r_fw_up_rx_if.h	Interface file for the processing of updating the serial ROM
r_fw_up_rx_private.h	Header file for the processing of updating the serial ROM
r_fw_up_buf.c	Processing of buffering the serial ROM update data
r_fw_up_buf.h	Header file for the processing of buffering the serial ROM update data
serial_rom.h	Definition of the serial ROM control commands
serial_rom_config.h	File for setting the number of blocks to be erased by the block erase for the serial ROM Use the SEL_ERASE_BLOCK_NUM constant in this file to set the number of blocks to be erased from the first block of the serial ROM.

(2) Option-Setting Memory

The following table shows the option-setting memory settings that can be used for writer program 2.

Table 3.13 Option-Setting Memory Settings That Can Be Used for Writer Program 2

Symbol	Address	Setting Value	Description
MDE	FE7F 5D00h to FE7F 5D03h	FFFF FFFFh	Little endian

(3) Constants

Table 3.14 to Table 3.20 list the constants that can be used for writer program 2.

Table 3.14 Constants Used for Writer Program 2 (serial_rom_write2_serial_rx671.c)

Constant Name	Setting value	Description
RECV_BYTE_SIZE	(1)	Number of bytes of receive data to request for the SCI FIT module
SEND_BYTE_SIZE	(1)	Number of bytes of send data to request for the SCI FIT module
COMMAND_YES_UPPER	('Y')	Character code for "Y" as an entry command
COMMAND_YES_LOWER	('y')	Character code for "y" as an entry command
COMMAND_CR	('\r')	Character code for the carriage return as an entry command
CMT_FREQUENCY_HZ	(2)	CMT frequency (for measuring the timeout of communication via the XMODEM/SUM protocol)
STRING_MAX_SIZE	SCI_CFG_CH10_TX_BUFSIZ	Maximum size of the character string to be output

Table 3.15 Constants Used for Writer Program 2 (r_xmodem.c)

Constant Name	Setting value	Description
XM_SOH	(0x01)	XMODEM/SUM control code "SOH"
XM_EOT	(0x04)	XMODEM/SUM control code "EOT"
XM_ACK	(0x06)	XMODEM/SUM control code "ACK"
XM_NAK	(0x15)	XMODEM/SUM control code "NAK"
XM_CAN	(0x18)	XMODEM/SUM control code "CAN"
XM_HEADER_SIZE	(1+1+1)	Header size (in bytes) of the XMODEM/SUM data block
XM_DATA_SIZE	(128)	Data size (in bytes) of the XMODEM/SUM data block
XM_SUM_SIZE	(1)	Check sum size (in bytes) of the XMODEM/SUM data block
XM_BLOCK_SIZE	(XM_HEADER_SIZE + XM_DATA_SIZE + XM_SUM_SIZE)	Size (in bytes) of the XMODEM/SUM data block
XM_RETRY_COUNT	(10)	Number of retries before determining a timeout of communication via the XMODEM/SUM protocol
UINT8T_0	(0)	0 of the uint8_t type
UINT8T_1	(1)	1 of the uint8_t type

Table 3.16 Constants Used for Writer Program 2 (r_fw_up_rx.c)

Constant Name	Setting value	Description
FW_UP_FIRM_EN_8MB_ADDRESS	(0x707FFFFF)	Last address of the first 8 MB of the QSPI area
CMT_FOR_ERASE_FREQUENCY_HZ	(2)	CMT frequency (for measuring the timeout of the block erase for the serial ROM)

Table 3.17 Constants Used for Writer Program 2 (r_fw_up_rx_private.h)

Constant Name	Setting value	Description
FW_UP_BINARY_BUF_SIZE	(256)	Buffer size for the data to be written to the serial ROM
FW_UP_BINARY_BUF_NUM	(2)	Number of buffers for the data to be written to the serial ROM
FW_UP_BUF_NUM	(60)	Number of buffers for the Motorola S record data (number of buffers that store the information about each field of records in Motorola S format based on record analysis)

Table 3.18 Constants Used for Writer Program 2 (r_fw_up_buf.h)

Constant Name	Setting value	Description
MOT_S_CHECK_SUM_FIELD	(0x02)	Number of characters of the check sum field of the Motorola S format
ADDRESS_LENGTH_S1	(0x04)	Number of characters of the address field of the Motorola S format (S1 type)
ADDRESS_LENGTH_S2	(0x06)	Number of characters of the address field of the Motorola S format (S2 type)
ADDRESS_LENGTH_S3	(0x08)	Number of characters of the address field of the Motorola S format (S3 type)
BUF_LOCK	(1)	Value for locking the Motorola S record data buffers
BUF_UNLOCK	(0)	Value for unlocking the Motorola S record data buffers

Table 3.19 Constants Used for Writer Program 2 (serial_rom.h)

Constant Name	Setting value	Description
CMD_WREN	(0x06)	Write Enable (WREN) command for the serial ROM
CMD_WRSR	(0x01)	Write Status Register (WRSR) command for the serial ROM
CMD_RDSR	(0x05)	Read Status Register (RDSR) command for the serial ROM
CMD_RDSCUR	(0x2B)	Read Security Register (RDSCUR) command for the serial ROM
CMD_BE	(0x52)	Block Erase (BE) command for the serial ROM
CMD_PP	(0x02)	Page Program (PP) command for the serial ROM
SERIALROM_EXIT_QSPI_MODE	(0x00)	Data for setting the status register for the serial ROM (setting that disables the Quad mode)
SERIALROM_CONFIG_REG	(0x00)	Data for setting the configuration register for the serial ROM

Table 3.20 Constants Used for Writer Program 2 (serial_rom_config.h)

Constant Name	Setting value	Description
SEL_ERASE_BLOCK_NUM	(1)	Number of blocks to be erased from the serial ROM

The user can set the value of SEL_ERASE_BLOCK_NUM.

Writer program 2 erases blocks from the first block of the serial ROM. Therefore, set SEL_ERASE_BLOCK_NUM to the number of blocks to be erased from the first block. The default is 1.

For SEL_ERASE_BLOCK_NUM, the user can specify a value in the range from 1 to 128.

(4) Type Definitions

Figure 25 to Figure 28 show the type definitions used for writer program 2.

```
typedef enum e_xmodem_proc_stage
{
    XMODEM_PROC_END = 0,
    XMODEM_PROCESSING,
    XMODEM_SOH_RECEIVED
} e_xmodem_proc_stage_t;

typedef struct st_xmodem_states
{
    uint8_t retry_counter;
    uint8_t expected_block_number;
    uint8_t recv_buf_index;
    uint8_t can_counter;
    uint8_t *precv_buf;
    e_xmodem_proc_stage_t proc_stage;
    xm_recv_func_t recv_func;
    xm_send_func_t send_func;
    xm_exec_func_t exec_func;
} st_xmodem_states_t;
```

Figure 25 Type Definitions Used for Writer Program 2 (r_xmodem.c)

```
typedef enum e_xmodem_err
{
    XMODEM_SUCCESS,
    XMODEM_SEND_ERR,
    XMODEM_RECV_ERR,
    XMODEM_TIMEOUT,
    XMODEM_PROC_BLOCK_ERR,
    XMODEM_RECV_CAN,
    XMODEM_DATA_ERR
} e_xmodem_err_t;

typedef e_xmodem_err_t (*xm_recv_func_t)(uint8_t* p_arg);
typedef e_xmodem_err_t (*xm_send_func_t)(uint8_t arg);
typedef e_xmodem_err_t (*xm_exec_func_t)(const uint8_t* p_buf, uint16_t size);
```

Figure 26 Type Definitions Used for Writer Program 2 (r_xmodem_if.h)

```
typedef enum e_fw_up_return_t
{
    FW_UP_SUCCESS,
    FW_UP_ERR_OPENED,
    FW_UP_ERR_NOT_OPEN,
    FW_UP_ERR_NULL_PTR,
    FW_UP_ERR_INVALID_RECORD,
    FW_UP_ERR_BUF_FULL,
    FW_UP_ERR_BUF_EMPTY,
    FW_UP_ERR_INITIALIZE,
    FW_UP_ERR_ERASE,
      FW_UP_ERR_CMT_FOR_ERASE,
    FW_UP_ERR_WRITE,
    FW_UP_ERR_VERIFY,
    FW_UP_ERR_INVALID_ADDRESS,
    FW_UP_ERR_INVALID_WRITE_SIZE,
    FW_UP_ERR_INTERNAL
} fw_up_return_t;

typedef struct st_fw_up_fl_data_t
{
    uint32_t src_addr;
    uint32_t dst_addr;
    uint32_t len;
    uint16_t count;
} fw_up_fl_data_t;
```

Figure 27 Type Definitions Used for Writer Program 2 (r_fw_up_rx_if.h)

```
typedef enum fw_up_mot_s_cnt_t
{
    STATE_MOT_S_RECORD_MARK = 0,
    STATE_MOT_S_RECORD_TYPE,
    STATE_MOT_S_LENGTH_1,
    STATE_MOT_S_LENGTH_2,
    STATE_MOT_S_ADDRESS,
    STATE_MOT_S_DATA,
    STATE_MOT_S_CHKSUM_1,
    STATE_MOT_S_CHKSUM_2
} fw_up_mot_s_cnt_t;

typedef struct MotSBufS
{
    uint8_t addr_length;
    uint8_t data_length;
    uint8_t *paddress;
    uint8_t *pdata;
    uint8_t type;
    uint8_t act;
    struct MotSBufS *pnext;
} fw_up_mot_s_buf_t;

typedef struct WriteDataS
{
    uint32_t addr;
    uint32_t len;
    uint8_t data[FW_UP_BINARY_BUF_SIZE];
    struct WriteDataS *pnext;
    struct WriteDataS *pprev;
} fw_up_write_data_t;
```

Figure 28 Type Definitions Used for Writer Program 2 (r_fw_up_buf.h)

(5) Variables

Table 3.21 to Table 3.24 list the static-type variables used for writer program 2.

Table 3.25 lists the const-type variables used for writer program 2.

Table 3.21 Variables of the "static" Type Used for Writer Program 2 (serial_rom_write2_serial_rx671.c)

Type	Variable Name	Description	Functions Supporting the Variable
static sci_hdl_t	s_sci_handle	SCI module control handle	main send_string_sci recv_byte_xm send_byte_xm update_serial_rom exec_firmware
static volatile bool	s_sci_send_end_flag	Flag for judging whether SCI send has ended	sci_callback send_string_sci
static volatile int32_t	s_timeout_count	Counter for judging whether communication via the XMODEM/SUM protocol has timed out	cmt_callback recv_byte_xm
static volatile bool	s_timeout_flag	Flag for detecting a timeout of communication via the XMODEM/SUM protocol	cmt_callback recv_byte_xm
static volatile bool	s_start_timer_flag	Flag for starting judgment of whether communication via the XMODEM/SUM protocol has timed out	cmt_callback recv_byte_xm

Table 3.22 Variables of the "static" Type Used for Writer Program 2 (r_xmodem.c)

Type	Variable Name	Description	Functions Supporting the Variable
static uint8_t	recv_buf[XM_BLOCK_SIZE]	Buffer for the data received via the XMODEM/SUM protocol	exec_xmodem

Table 3.23 Variables of the "static" Type Used for Writer Program 2 (r_fw_up_rx.c)

Type	Variable Name	Description	Functions Supporting the Variable
static bool	is_opened	Flag indicating that initial setup of the serial ROM update is complete	fw_up_open fw_up_close fw_up_put_data fw_up_get_data disable_quad_mode_serial_rom erase_serial_rom write_serial_rom
Static volatile int32_t	s_timeout_count_for_erase	Counter for judging whether the block erase for the serial ROM has timed out	erase_serial_rom cmt_callback_for_erase
static volatile bool	s_timeout_flag_for_erase	Flag for detecting a timeout of the block erase for the serial ROM	erase_serial_rom cmt_callback_for_erase
static volatile bool	s_start_timer_flag_for_erase	Flag for starting judgment of whether the block erase for the serial ROM has timed out	erase_serial_rom cmt_callback_for_erase

Table 3.24 Variables of the "static" Type Used for Writer Program 2 (r_fw_up_buf.c)

Type	Variable Name	Description	Functions Supporting the Variable
static fw_up_mot_s_buf_t	mot_s_buf [FW_UP_BUF_NUM]	Motorola S record data buffer	fw_up_buf_init fw_up_memory_init
static fw_up_mot_s_buf_t	*papp_put_mot_s_buf	Pointer to the Motorola S record data buffer that is currently used for analysis of the Motorola S format	fw_up_buf_init fw_up_put_mot_s
static fw_up_mot_s_buf_t	*papp_get_mot_s_buf	Pointer to the Motorola S record data buffer that is currently used for creating the data to be written to the serial ROM	fw_up_buf_init fw_up_get_binary
static fw_up_write_data_t	write_buf [FW_UP_BINARY_BUF_NUM]	Data buffer for the write to the serial ROM	fw_up_buf_init
static fw_up_write_data_t	*papp_write_buf	Pointer to the data buffer that is currently used for the write to the serial ROM	fw_up_buf_init fw_up_get_binary
static fw_up_mot_s_cnt_t	mot_s_data_state	Record analysis status of the Motorola S format	fw_up_buf_init fw_up_put_mot_s
static uint32_t	write_current_address	Current write-destination address of the serial ROM (address in the QSPI area)	fw_up_buf_init fw_up_get_binary
static bool	detect_terminal_flag	Termination record detection flag	fw_up_buf_init fw_up_put_mot_s fw_up_get_binary

Table 3.25 Variables of the "const" Type Used for Writer Program 2 (serial_rom_write2_serial_rx671.c)

Type	Variable Name	Description	Functions Supporting the Variable
static const uint8_t	s_string_menu0[]	"RX671 Serial ROM Update ver1.00\r\n"	update_serial_rom
static const uint8_t	s_string_update[]	"Erase and Write (Y/N)?"	update_serial_rom
static const uint8_t	s_string_erase_success[]	“Erasing has been done.\r\n”	update_serial_rom
static const uint8_t	s_string_download[]	“Start XMODEM download…\r\n”	update_serial_rom
static const uint8_t	s_string_finish_xmodem[]	“Updating has been done.\r\n”	update_serial_rom
static const uint8_t	s_string_cancel[]	“Command canceled.\r\n”	update_serial_rom
static const uint8_t	s_string_input[]	“> ”	update_serial_rom
static const uint8_t	s_string_crlf[]	“\r\n”	main update_serial_rom
static const_uin8_t	s_string_cmt_err[]	"CMT module error.\r\n"	update_serial_rom
static const uint8_t	s_string_mode_setting_err[]	"Serial ROM mode setting error.\r\n"	update_serial_rom
static const uint8_t	s_string_erase_err[]	“Serial ROM Erasing error.\r\n”	update_serial_rom
static const uint8_t	s_string_send_err[]	“Send error.\r\n”	update_serial_rom
static const uint8_t	s_string_recv_err[]	“Receive error.\r\n”	update_serial_rom
static const uint8_t	s_string_timeout[]	“Timeout.\r\n”	update_serial_rom
static const uint8_t	s_string_block_err[]	“Block processing error.\r\n”	update_serial_rom
static const uint8_t	s_string_data_err[]	“Data error.\r\n”	update_serial_rom
static const uint8_t	s_string_init_update_err[]	“Initialize update error.\r\n”	update_serial_rom
static const uint8_t	s_string_fin_update_err[]	“Finalize update error.\r\n”	update_serial_rom

(6) Functions

Table 3.26 to Table 3.29 list the functions for writer program 2.

Table 3.26 Functions for Writer Program 2 (serial_rom_write2_serial_rx671.c)

Function Name	Summary
main	Main processing Initializes the SCI and QSPIX, and invokes the serial ROM update function.
update_serial_rom	Serial ROM update function Performs operations such as outputting a message or inputting a command to the terminal software on the host PC and invoking a function that changes the communication mode of the serial ROM, function that performs the block erase for the serial ROM, and function that processes communication via the XMODEM/SUM protocol.
send_byte_xm	Callback function for the XMODEM/SUM protocol Sends 1-byte data.
recv_byte_xm	Callback function for the XMODEM/SUM protocol Receives 1-byte data.
block_proc_xm	Callback function for the XMODEM/SUM protocol Processes the data of 1 data block.
send_string_sci	Character string send processing
rom_access_error_callback	Callback function for the QSPIX FIT module Performs confirmation when a ROM access error interrupt occurs.
sci_callback	Callback function for the SCI FIT module Confirms completion of SCI send processing.
cmt_callback	Callback function for the CMT FIT module Detects a timeout of communication via the XMODEM/SUM protocol.

Table 3.27 Functions for Writer Program 2 (r_modem.c)

Function Name	Summary
exec_xmodem	Processes communication via the XMODEM/SUM protocol.
xmodem_recv_soh	Receives the header of the data block of the XMODEM/SUM protocol.
xmodem_check_eot	Checks the header of the data block of the XMODEM/SUM protocol.
xmodem_recv_block	Receives 1 data block of the XMODEM/SUM protocol.
xmodem_analyze_block	Analyzes data blocks of the XMODEM/SUM protocol.
xmodem_proc_data	Processes the data of 1 data block of the XMODEM/SUM protocol.
xmodem_send_response	Processes response of the XMODEM/SUM protocol.

Table 3.28 Functions for Writer Program 2 (r_fw_up_rx.c)

Function Name	Summary
fw_up_open	Initializes the update of the serial ROM.
fw_up_close	Finalizes the update of the serial ROM.
analyze_and_write_data	Invokes functions such as the receive data analysis function, serial ROM write data acquisition function, and serial ROM write function.
fw_up_put_data	Analyzes the receive data.
fw_up_get_data	Acquires the serial ROM write data.
disable_quad_mode_serial_rom	Disables the QUAD mode of the serial ROM.
erase_serial_rom	Block erase for the serial ROM Performs block erase for the serial ROM from its first block. The number of blocks to be erased is specified for SEL_ERASE_BLOCK_NUM.
write_serial_rom	Writes data to the serial ROM.
write_serial_rom_send_command	Sends a command to write data to the serial ROM.
cmt_callback_for_erase	CMT FIT module callback function Detects a timeout of the block erase for the serial ROM.

Table 3.29 Functions for Writer Program 2 (r_fw_up_buf.c)

Function Name	Summary
fw_up_buf_init	Initializes the buffer that is used to update the serial ROM.
fw_up_memory_init	Initializes the pointer to the buffer.
fw_up_put_mot_s	Analyzes the records of the Motorola S format.
fw_up_get_binary	Acquires the serial ROM write data.
fw_up_ascii_to_hexbyte	Converts the data format from ASCII to binary.

3.3 FIT Modules Used

This section shows the FIT modules that are used by the application program, writer program 1, and writer program 2. This section also describes the settings of each FIT module.

3.3.1 List of FIT Modules Used

Table 3.30 lists the FIT modules that are used.

Table 3.30 List of FIT Modules Used

FIT Module	Document Title	Application Program	Writer Program 1	Writer Program 2
BSP	RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)	Used	Used	Used
QSPIX	RX Family QSPIX Module Using Firmware Integration Technology (R01AN5685)	Used	Used	Used
CMT	RX Family CMT Module Using Firmware Integration Technology (R01AN1856)	-	-	Used
SCI	RX Family SCI Module Using Firmware Integration Technology (R01AN1815)	-	-	Used
BYTEQ	RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)	-	-	Used

3.3.2 FIT Module Settings

The FIT module and e^2 studio SC settings used are listed below. For SC settings, the items and setting details match those displayed on the setting menu. For details of the FIT modules, refer to the associated FIT module documents.

Table 3.31 BSP Module Settings

(Settings Common to the Application Program, Writer Program 1, and Writer Program 2)

Category	Item	Setting/Description
Smart Configurator >> Components >> r_bsp		Properties are left in the default settings.
Smart Configurator >> Clocks		**Clock** tab: Default settings
	VCC settings	3.3 (V)
	Main clock settings	Operation: Checked. Oscillation source: Resonator Frequency: 24 MHz Oscillation waittime: 9980 (µs) (actual value: 10000)
	PLL circuit settings	Frequency Division: ×1 Frequency Multiplication: ×10.0
	System clock settings	Clock source: PLL circuit System clock (ICLK): ×1/2 120 (MHz) Peripheral module clock (PCLKA): ×1/2 120 (MHz) Peripheral module clock (PCLKB): ×1/4 60 (MHz) Peripheral module clock (PCLKC): ×1/4 60 (MHz) Peripheral module clock (PCLKD): ×1/4 60 (MHz) External bus clock (BCLK): ×1/4 60 (MHz) FlashIF clock (FCLK): ×1/4 60 (MHz)
	Sub-clock settings	Operation: Checked. (The sub-clock is not used, but the default setting is left unchanged.)
	HOCO clock settings	Stopped: Unchecked.
	LOCO clock settings	Stopped: Unchecked.
	IWDT dedicated clock settings	Stopped: Unchecked.

Table 3.32 QSPIX Module Settings

(Settings Common to the Application Program, Writer Program 1, and Writer Program 2)

Category	Item	Setting/Description
Smart Configurator >> Components >> r_qspix_rx		Use the default settings other than the settings shown below:
	Resources >> QSPIX	QSPIX0: Checked. QSPCLK pin: Used: Checked. QSSL pin: Used: Checked. QIO0 pin: Used: Checked. QIO1 pin: Used: Checked. QIO2 pin: Used: Checked. QIO3 pin: Used: Checked.
Smart Configurator >> Pins		Use the default settings other than the changes shown below:
	Function: QIO0	The following pin assignments are selected: PD6/D6/MTIC5V/MTIOC8A/POE4#/SSLC2-A/ SDHI_D0-B/QIO0-B/IRQ6/AN101
	Function: QIO1	The following pin assignments are selected: PD7/D7/MTIC5U/POE0#/SSLC3-A/SDHI_D1-B/ QIO1-B/IRQ7/AN100
	Function: QIO2	The following pin assignments are selected: PD2/D2/MTIOC4D/TIC2/CRX0/MISOC-A/ SDHI_D2-B/QIO2-B/IRQ2/AN105
	Function: QIO3	The following pin assignments are selected: PD3/D3/MTIOC8D/POE8#/TOC2/RSPCKC-A/ SDHI_D3-B/QIO3-B/IRQ3/AN104
	Function: QSPCLK	The following pin assignments are selected: PD5/D5/MTIC5W/MTIOC8C/POE10#/SSLC1-A/ SDHI_CLK-B/QSPCLK-B/IRQ5/AN102
	Function: QSSL	The following pin assignments are selected: PD4/D4/MTIOC8B/POE11#/SSLC0-A/ SDHI_CMD-B/QSSL-B/IRQ4/AN103

Table 3.33 CMT Module Settings (Settings for Only Writer Program 2)

Category	Item	Setting/Description
Smart Configurator >> Components >> r_cmt_rx		Use the default settings.

Table 3.34 SCI Module Settings (Settings for Only Writer Program 2)

Category	Item	Setting/Description
Smart Configurator >> Components >> r_sci_rx		Use the default settings other than the changes shown below:
	Configurations	Include software support for channel1：Not Include software support for channel10：Include Transmit end interrupt：Enable
	Resources >> SCI	SCI10: Checked. SCK10 pin: Used: Checked. RXD10/SMISO10/SSCL10 pin: Used: Checked. TXD10/SMOSI10/SSDA10 pin: Used: Checked.
Smart Configurator >> Pins		Use the default settings other than the changes shown below:
	Function: RXD10	The following pin assignments are selected: P86/MTIOC4D/TIOCA0/SMISO10/SSCL10/ RXD10/SMISO010/ SSCL010/RXD010/IRQ14
	Function: TXD10	The following pin assignments are selected: P87/MTIOC4C/TIOCA2/SMOSI10/SSDA10/TXD10/ SMOSI010/SSDA010/TXD010/SDHI_DS-C/IRQ15

Table 3.35 BYTEQ Module Settings (Settings for Only Writer Program 2)

Category	Item	Setting/Description
Smart Configurator >> Components >> r_byteq		Use the default settings.

3.4 Operation Confirmation Conditions

This section shows the conditions under which operations of the application program, writer program 1, and writer program 2 were verified.

Table 3.36 Operation Confirmation Conditions

Item	Description
MCU	R5F5671EHDFB (RX671 Group)
Operating frequency	• Main clock: 24 MHz • PLL circuit output clock: 240 MHz • System clock (ICLK): 120 MHz (PLL circuit output clock divided by 2) • Peripheral module clock A (PCLKA): 120 MHz (PLL circuit output clock divided by 2) • Peripheral module clock B (PCLKB): 60 MHz (PLL circuit output clock divided by 4) • Peripheral module clock C (PCLKC): 60 MHz (PLL circuit output clock divided by 4) • Peripheral module clock D (PCLKD): 60 MHz (PLL circuit output clock divided by 4) • Bus clock (BCLK): 60 MHz (PLL circuit output clock divided by 4) • FlashIF clock (FCLK): 60 MHz (PLL circuit output clock divided by 4)
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio Version 2022-04
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.04.00
	Compiler options -lang = c99 For other settings, refer to 3.1.1.2, Build Settings in e2 studio, and 3.2.1.1(2) Build Settings in e2 studio.
iodefine.h version	V1.00
Endian order	Little endian
Operating mode	Single-chip mode
Processor mode	Supervisor mode
Sample program version	Version 2.00
Emulator	E2 emulator Lite
Board used	Renesas Starter Kit+ forRX671 (product No.: RTK55671EHSxxxxxxx)

3.5 Sample Program Operation Confirmation

Figure 29 shows "Procedure for Verifying the Operation of the Application Program When Using Writer Program 1".

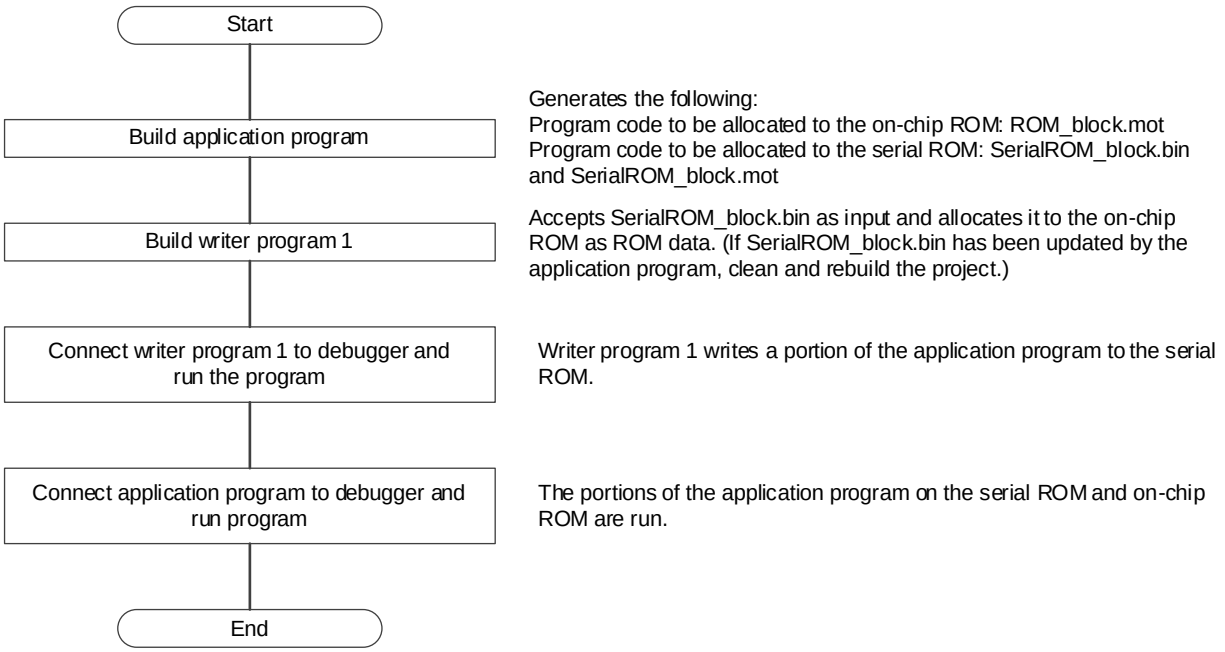


Figure 29 Procedure for Verifying the Operation of the Application Program When Using Writer Program 1

Figure 30 shows the Procedure for Verifying the Operation of the Application Program When Using Writer Program 2.

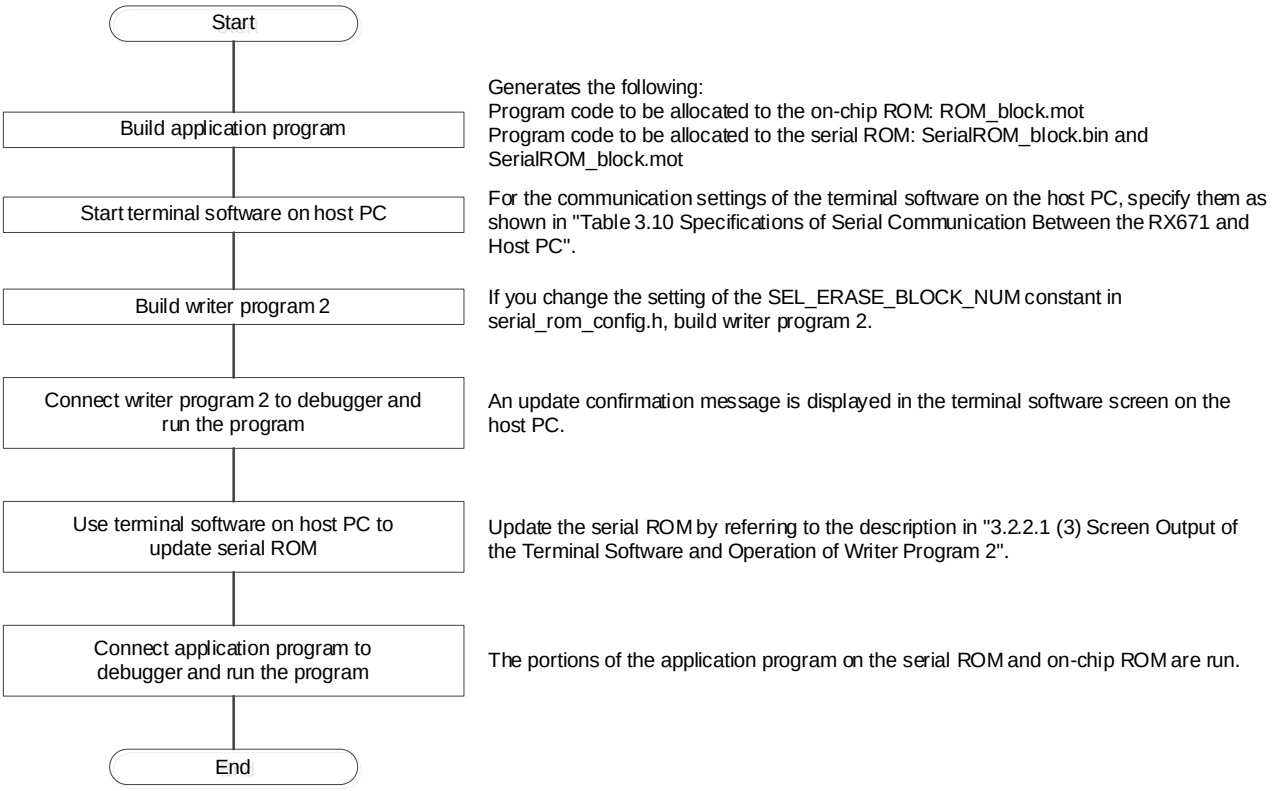


Figure 30 Procedure for Verifying the Operation of the Application Program When Using Writer Program 2

3.5.1 Debugger Connection Settings for Application Program

The settings necessary for connecting the application program to the debugger in e^2 studio are described below.

Figure 31 to Figure 36 show the debugger connection settings for the application program.

From the **Run** menu, select **Debug Configurations…** to display the **Debug Configurations** dialog box.

From **Renesas GDB Hardware Debugging** select **xip_sample_rx671 HardwareDebug** ([1] in the figure), select the **Startup** tab ([2] in the figure), and click the **Add…** button ([3] in the figure).

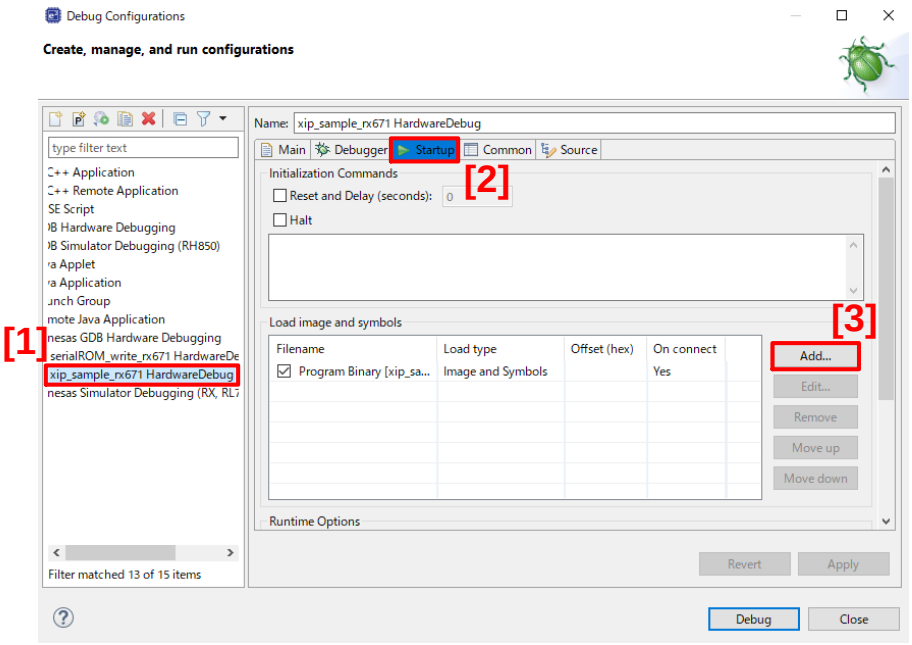


Figure 31 Debugger Connection Settings for Application Program (1/6)

When the **Add download module** dialog box appears, click the **Workspace…** button.

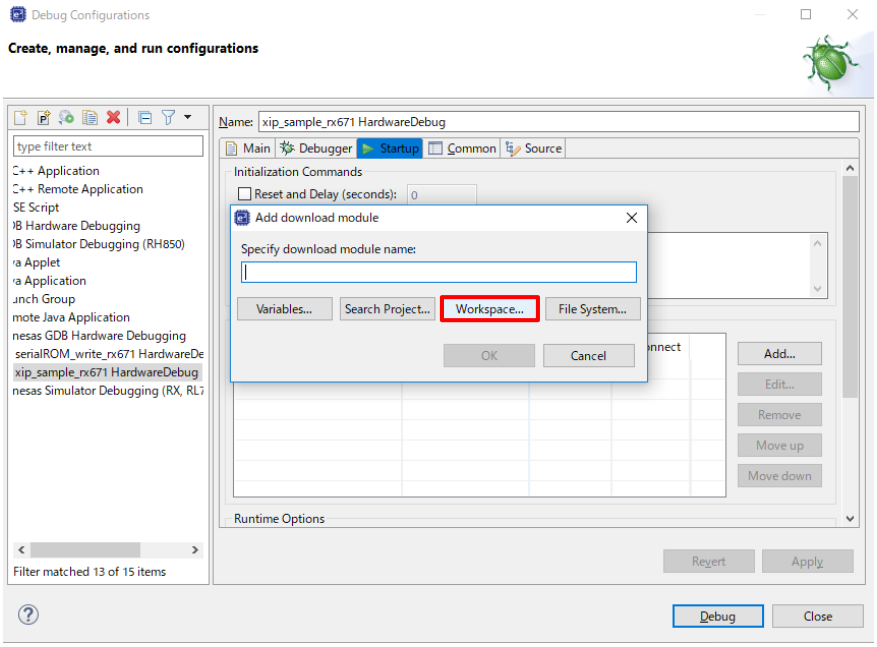


Figure 32 Debugger Connection Settings for Application Program (2/6)

Select **xip_sample_rx671** → **HardwareDebug** → **ROM_block.mot**, then click the **OK** button.

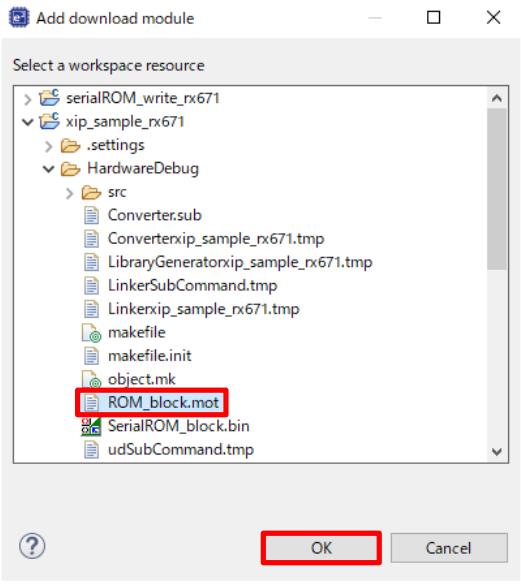


Figure 33 Debugger Connection Settings for Application Program (3/6)

In the **Add download module** dialog box, click the **OK** button.

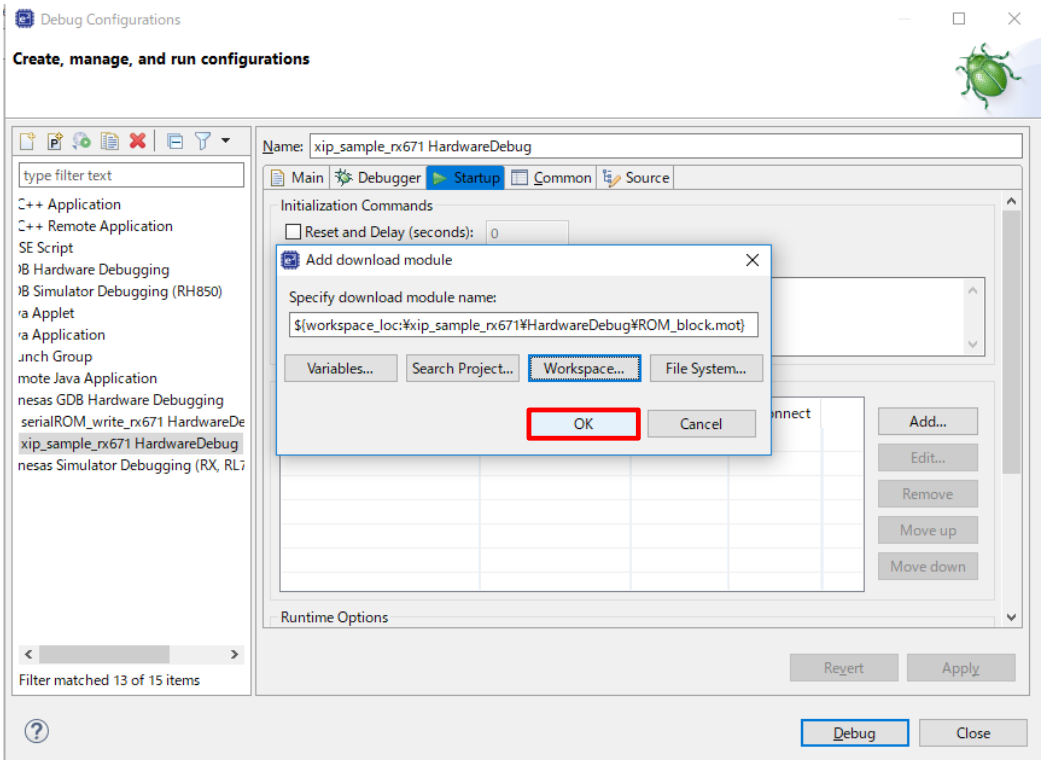


Figure 34 Debugger Connection Settings for Application Program (4/6)

From the **Load type** pulldown menu next to the filename **Program Binary [xip_sample_rx671.x]**, select **Symbols only**.

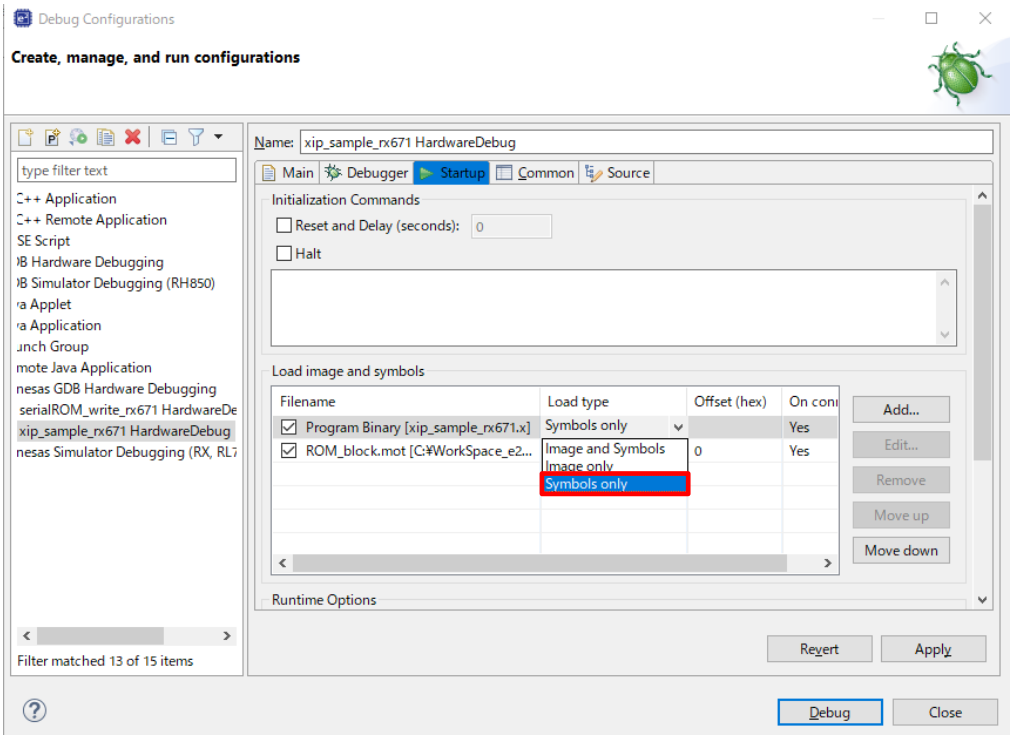


Figure 35 Debugger Connection Settings for Application Program (5/6)

From the **Load type** pulldown menu next to the filename **ROM_block.mot**, select **Image only**. Finally, click the **Apply** button.

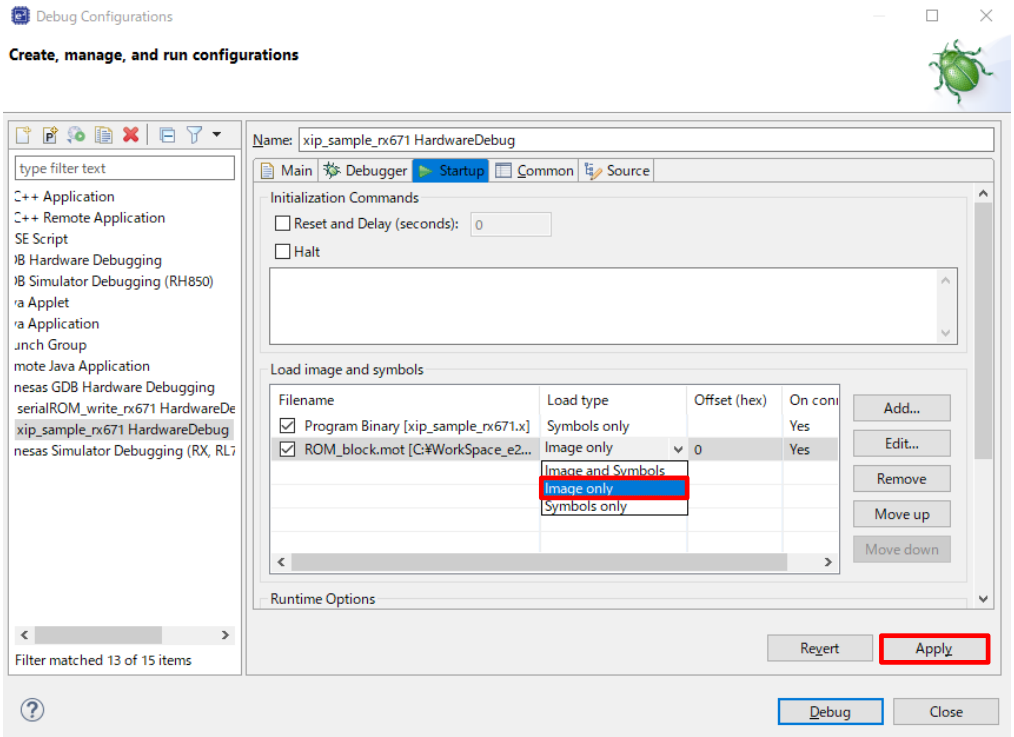


Figure 36 Debugger Connection Settings for Application Program (6/6)

3.5.2 Notes

3.5.2.1 Address of the Application Program to Be Allocated to the Serial ROM

To use writer program 1, make sure that the application program code (the SerialROM_sec section) is written to the serial ROM from the address 0x70000000.

Use writer program 1 to receive the binary data of the application program code to be allocated to the serial ROM and write the received data to the first block (at address 0x00000000) in the serial ROM.

To use writer program 2, make sure that the address in the serial ROM at which the application program code is to be allocated is a multiple of 256 of the QSPI area (the last byte is 0x00).

For the serial ROM installed on the RSK, data cannot be written across multiple pages and written data is wrapped around at a page (256 bytes) boundary. Therefore, to write data with the maximum size (256 bytes) by issuing a write command (Page Program command), the data must be written from an address that is a multiple of 256.

3.5.2.2 Project Configuration

Place the application program project, xip_sample_rx671, and the writer program 1 project, serialROM_write1_direct_rx671, in the same workspace.

When you use writer program 1, do not change the project name of the application program.

If you have to change the project name, you must change the SerialROM_block.bin file storage location (red portion of the following path) specified for the -binary option in "3.2.1.1(2)(b) Input Binary File Specification (-binary) Option Setting".

-binary="${WorkspaceDirPath}/xip_sample_rx671/HardwareDebug/SerialROM_block.bin" (SerialROM_WriteData_sec:4/DATA,_g_SerialROM_WriteData)

Note: The above option setting is split into two lines for explanatory purposes in this document, but the actual option setting should not contain a line break.

3.5.2.3 Note on Building the Writer Program 1

If changes are made to the application program, clean and rebuild the writer program 1 project.

3.5.2.4 Debugging the Portion of the Program in Serial ROM

When the application program is connected to the debugger and the portion of the program in the serial ROM is being debugged, it is not possible to set software breaks in the portion of the program in the serial ROM.

3.5.2.5 Using Renesas Flash Programmer to Write the Application Program to the RX671

When using Renesas Flash Programmer (RFP) to write the application program to the RX671, use the file ROM_block.mot generated when building the application program.

For information on using RFP, refer to Renesas Flash Programmer: Flash Memory Programming Software User's Manual (R20UT5038).

4. Importing a Project

The sample programs are distributed in e² studio project format. This section shows how to import a project into e² studio. After importing a project, check the build and debug settings.

4.1 Procedure in e² studio

To use sample programs in e² studio, follow the steps below to import them into e² studio. (Note that depending on the version of e² studio you are using, the interface may appear somewhat different from the screenshots below.)

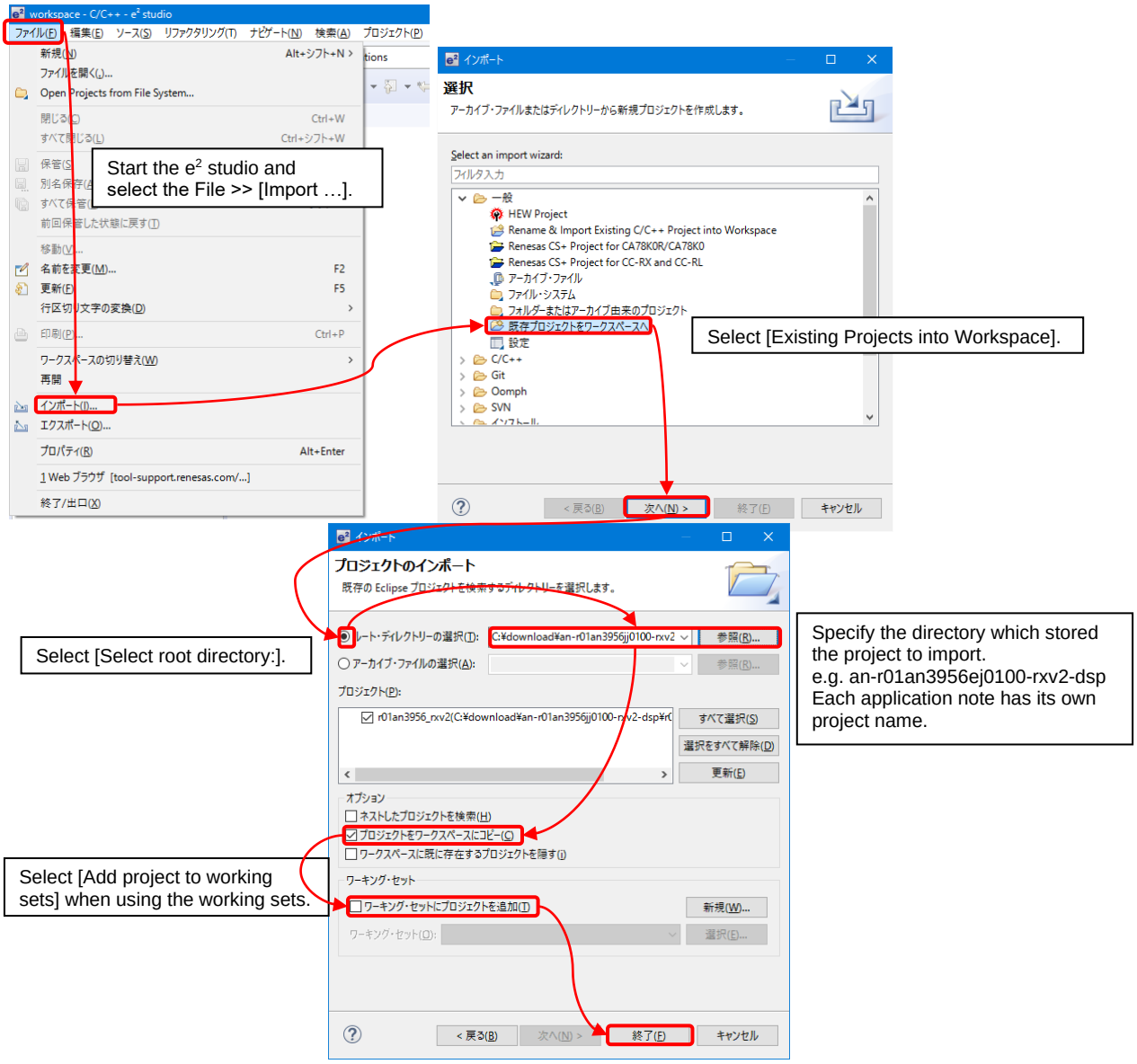


Figure 4.1 Importing a Project into e² studio

5. Obtaining the Development Environment

5.1 e² studio

Visit the following URL and download e² studio.

https://www.renesas.com/jp/ja/software-tool/e-studio

Note that this document assumes that the version of e² studio is the same as or later than the version indicated in "Table 3.36 Operation Confirmation Conditions". If the version is earlier than the indicated version, some e² studio functions might be unavailable. Make sure to download the latest version of e² studio on the website.

5.2 Compiler Package

Visit the following URL and download the RX Family C/C++ Compiler Package.

https://www.renesas.com/jp/ja/software-tool/cc-compiler-package-rx-family

6. Additional Information

6.1 Notes on Using the Evaluation Version of C/C++ Compiler Package for RX Family

The evaluation version of C/C++ Compiler Package for RX Family can only be used for a limited duration and other usage limitations apply. When the evaluation period expires, the size of linkable objects is reduced to 128 KB or less, which may cause incorrect generation of the load module.

For details, refer to the following software tool page for evaluation versions on the Renesas website:

https://www.renesas.com/jp/ja/software-tool/evaluation-software-tools

7. Reference Documents

- RX671 Group User's Manual: Hardware (R01UH0899)
- CC-RX Compiler User's Manual (R20UT3248)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family QSPIX Module Using Firmware Integration Technology (R01AN5685)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
- RX Smart Configurator User's Guide: e² studio (R20AN0451)
- Renesas Flash Programmer: Flash Memory Programming Software User's Manual (R20UT5038)
- Renesas Starter Kit+ for RX671 User's Manual (R20UT4879)
- Renesas Starter Kit+ for RX671 CPU Board Schematics (R20UT4878)

The latest version can be downloaded from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan. 21. 22	—	First edition issued
2.00	Jun. 30. 22	Global	The information in "Function Specifications" was moved to "Functions", and the "Function Specifications" section was deleted. The writer program in revision 1.00 was renamed to "writer program 1". The project name of the writer program in revision 1.00 was changed from "serialROM_write_rx671" to "serialROM_write1_direct_rx671".
		p6	"Figure 3 Diagram of Connection Between the RX671 and the Host PC" was added. The title of Table 2.1 was changed. "Table 2.2 SCI Pins Used for Connection Between the RX671 and Host PC" was added.
		p8	The cross-reference to the section that shows the FIT modules used was corrected.
		p10 and p13	In Figure 5, output of a Motorola S format file was added as a file that is generated when the program code to be allocated to the serial ROM is built. The description in the body text was also changed accordingly. With these changes, in Figure 9, "SerialROM_block.mot=SerialROM_sec" was added under [Division output mot file (for Stype)]. The description in the body text was also changed accordingly.
		p11	For the program code to be allocated to the serial ROM, under the address that is to be set for the application program, a supplementary explanation about the address to be set was added.
		p17	In "Table 3.2 Files Used by Application Program", the file name "cmd_serial_rom.h" was changed to "serial_rom.h".
		p19	Because writer program 2 was added in revision 2.00, the section "3.2 Writer Program" was reorganized to have subsections for writer program 1 (writer program in revision 1.00) and for writer program 2.
		p20 and p21	With a change to the project name of writer program 1, Figure 15 and Figure 17 were updated.
		p22	In the flowchart in Figure 18, in the supplementary explanation about the processing "Write data to 1st block in serial ROM", the address representation was changed from a QSPI-area-based address (0x70000000) to a serial-ROM-based address (0x00000000).
		p23	In "Table 3.6 Files Used by Writer Program 1", the file name "cmd_serial_rom.h" was changed to "serial_rom.h".
		p25 to p39	Section "3.2.2 Writer Program 2" was added.
		p40	The information in "Table 3.30 List of FIT Modules Used" was changed.

2.00	Jun. 30. 22	p41	In "3.3.2 FIT Module Settings", the FIT module settings for the CMT, SCI, and BYTEQ were added.
		p44	In "Table 3.36 Operation Confirmation Conditions", the required versions of the integrated development environment, compiler, and sample program were changed. Also, the cross-reference indicated in the "Compiler options" column was changed.
		p45	Figure 29 was modified because the writer program was renamed to "writer program 1" and the number of files generated when the application program is built increased. Also, Figure 30 was added because writer program 2 was added.
		p46 to p48	Figure 31 to Figure 36 were updated because project names were changed or added.
		p49	"3.5.2.1 Address of the Application Program to Be Allocated to the Serial ROM" was added to "3.5.2 Notes". With the reorganization of the section, a cross-reference in "3.5.2.2 Project Configuration" was changed.
		p51	Information about the FIT modules for the CMT, SCI, and BYTEQ was added in "7 Reference Documents".

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/.