# RL78/G22, RL78/G23

## Firmware Updating Communications Module

### Introduction

This application note describes a firmware updating communications module for the RL78/G22 and RL78/G23.

In a system consisting of a primary MCU and a secondary MCU, this module allows user updating of the firmware of the secondary MCU. This application note explains how to use this module, incorporate its API functions into user applications, and extend its functionality.

The release package associated with this application note includes two demonstration projects. You can confirm the basic operation of the functionality for updating the firmware of the secondary MCU with the use of this module by following the steps described in chapter 5, Demonstration Projects, to build an environment to run the demonstration.

### Operation Confirmation Devices

RL78/G22 (R7F102GGE)

RL78/G23 (R7F100GSN)

If you intend to use this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to suit the specifications of the alternative MCU.

### Related Application Notes

- RL78 Family Board Support Package Module Using Software Integration System (R01AN5522)
- RL78 Smart Configurator User's Guide: e2 studio (R20AN0579)
- Smart Configurator User's Guide: RL78 API Reference (R20UT4852)
- RL78/G22, RL78/G23, RL78/G24 Firmware Update Module (R01AN6374)

### Target Compilers

- CC-RL V1.15.00 from Renesas Electronics

For details of the environments in which operation has been confirmed, refer to section 6.1, Environments for Confirming Operation.

## Contents

# 1. Overview

## 1.1 About the Firmware Updating Communications Module

The firmware updating communications module is middleware which controls communications between MCUs in which the secondary MCU receives firmware for use in updating from the primary MCU and applies the firmware to updating in a system of the kind shown in Figure 1-1, consisting of the primary MCU and the secondary MCU. Users can easily update the firmware of the secondary MCU by embedding this module into the primary and secondary MCUs.

## 1.2 Supported Communications IP and Hardware Configuration

This module supports UART communications through serial communication interface (SCI) and synchronous serial communications using either SCI or the serial peripheral interface (RSPI) as the communications interfaces.

### 1.2.1 UART Communications

Figure 1-1 shows the hardware configuration for UART communications assumed for this module. The primary and secondary MCUs have one-to-one connections on the same bus via two-wire UART (TXD and RXD).

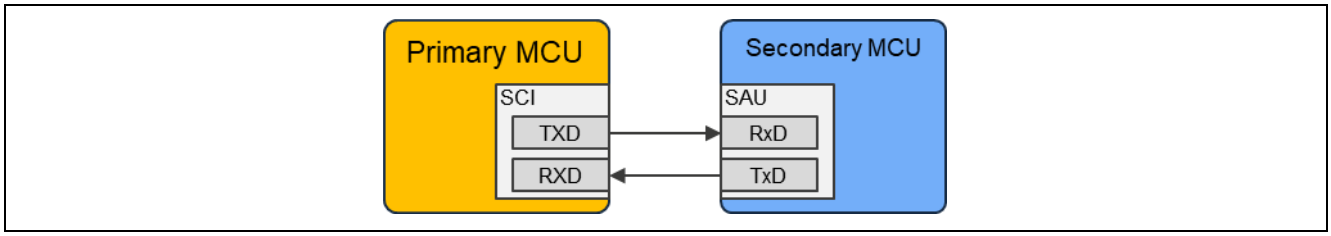


**Figure 1-1 Hardware Configuration(UART)**

### 1.2.2 SPI Communications

Figure 1-2 shows the hardware configuration for synchronous serial communication assumed for this module. The primary MCU and secondary MCUs are connected on the same bus via a three-wire bus (SI(hereinafter referred to as MOSI), SO(hereinafter referred to as MISO), SCK).

In SPI communication for this module, the MISO line is used to signal the primary MCU that the secondary MCU is in a busy state and cannot communicate. When the Secondary MCU is busy, it outputs MISO low. After returning from the busy state to a communicable state, the Secondary MCU changes MISO to open drain. Therefore, when using this module for SPI communication, pull up the MISO line.

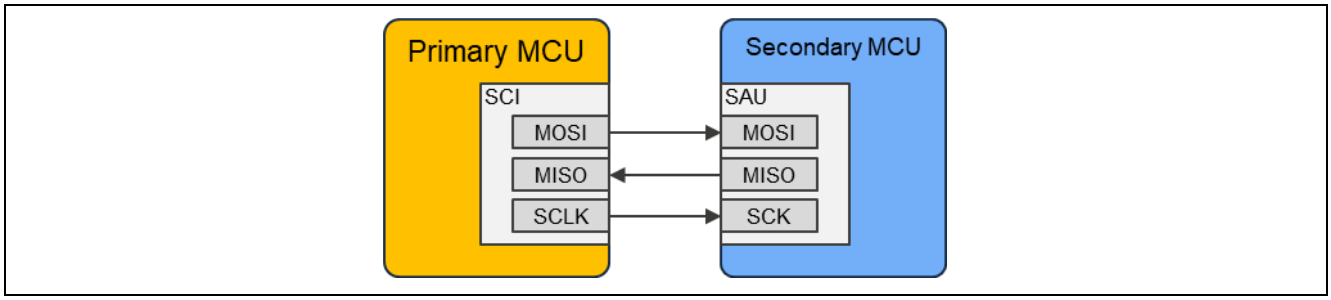


**Figure 1-2 Hardware Configuration(SPI)**

## 1.3 Software Configuration

Figure 1-3 show the configurations of the software modules.

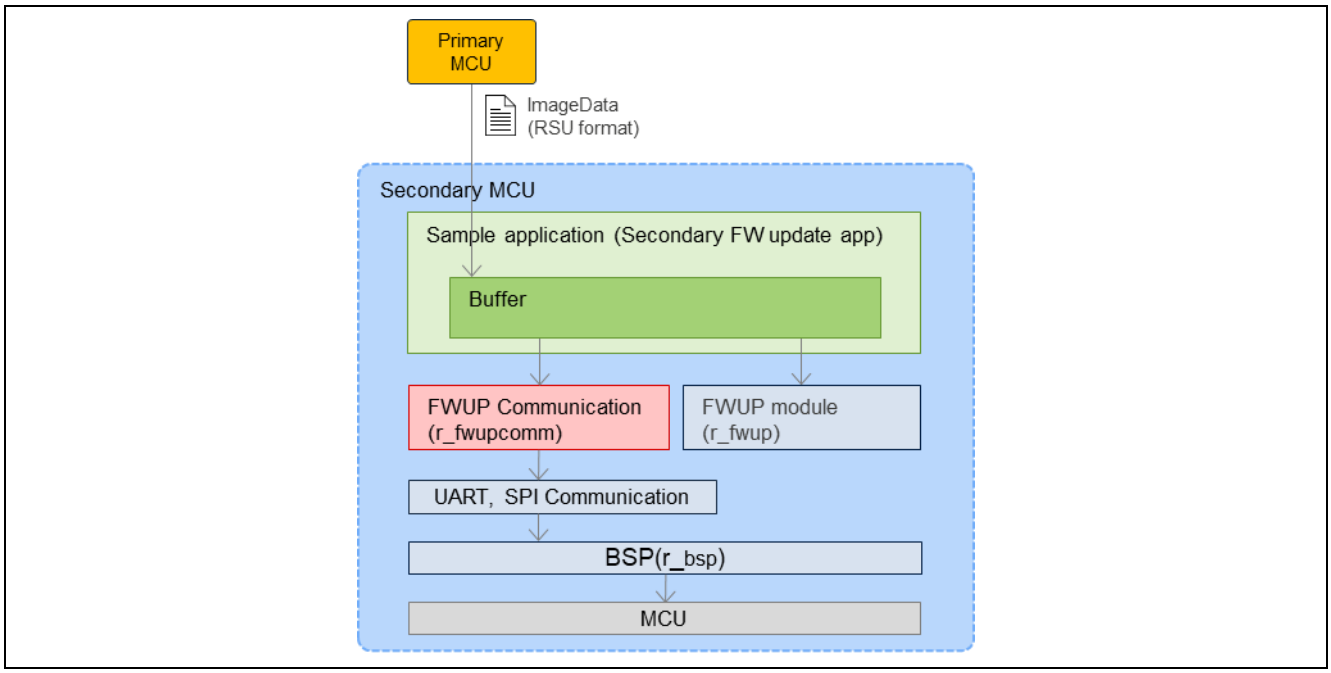


**Figure 1-3 Configuration of Software Modules in the Secondary MCU**

### 1.3.1 Setting UART Communications

The operation of this module has been confirmed with the settings for UART communications listed in section 6.2, Settings for UART Communications.

## 1.4 Packet Communications

Packet communications proceed between primary and secondary MCUs via the communications interface. The primary MCU sends request packets to the secondary MCU. When the secondary MCU receives a request packet, it processes the command and sends the results to the primary MCU as a response packet. Figure 1-4 shows the flow of packet communications.

| Primary MCU | | Secondary MCU |
|---|---|---|
| Sends a request packet. | | |
| | -----> | |
| | | Receives the request packet. |
| | | Processes the command. |
| | | Sends a response packet. |
| | <----- | |
| Receives the response packet. | | |

**Figure 1-4 Flow of Packet Communications**

All commands are classified according to their individual purposes, and the classification is called the command class.

## 1.5 Data Format

This section describes the specifications for packet communications between the primary and secondary MCUs. The specification of the data format is independent of the method of physical communications between the MCUs.

### 1.5.1 Data Format of Packets

Figure 1-5 shows the data format of command packets, each of which consists of a command header and command data.

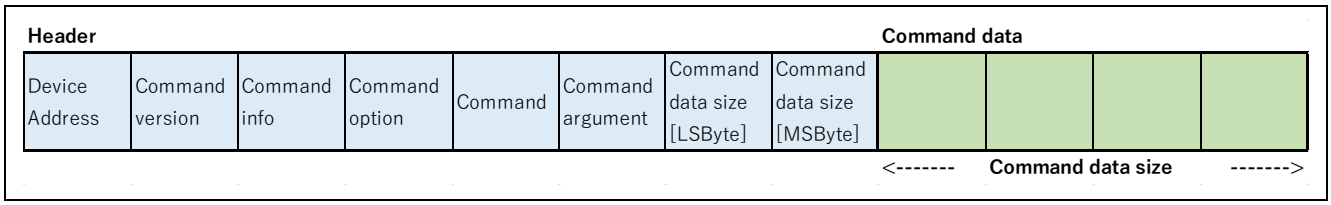


**Figure 1-5 Data Format of Command Packets**

Figure 1-6 shows the data format of response packets.

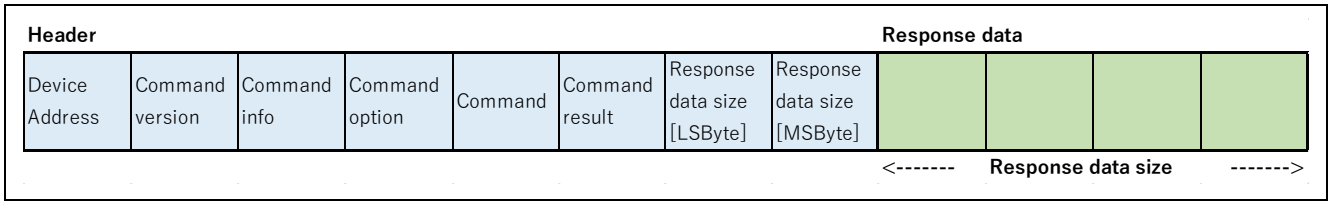


**Figure 1-6 Data Format of Response Packets**

Table 1-1 lists the specifications of the headers of packets.

**Table 1-1 Specifications of the Headers of Packets**

| Item | Description |
|---|---|
| Device address | Device address of the secondary MCU to which the command is sent.<br>The secondary MCU only processes a command when it receives the command with its own device address in the header.<br>• 0x00 – 0xFD: Device address of the secondary MCU<br>• 0xFE: Broadcast address<br>• 0xFF: Reserved. |
| Command version | Version of the command. The secondary MCU only processes a command when the version of the command is the same as that of the command on the secondary MCU.<br>0x00 – 0xFF |
| Command info | • b7: 0: Command, 1: Response<br>• b4 – b6: Command class. Refer to section 1.6, Specifications of Commands.<br>• b0 – b3: Command ID |
| Command option | • b7: 0: A response is to be sent. 1: No response is to be sent.<br>• b0 – b6: Reserved. |
| Command | Indicates the command. Refer to section 1.6, Specifications of Commands. |
| Command argument/result | Indicates an argument of the command when a command is being sent.<br>Indicates the result of executing the command when a response is being sent.<br>Refer to section 1.6, Specifications of Commands. |
| Command/Response data size | Size of the command data or response data.<br>The size must be in bytes and a multiple of 4. |

## 1.6 Specifications of Commands

This module has definitions of FWUP commands to control updating of firmware of the secondary MCU and common commands for general data communications.

**Table 1-2 List of Command Class**

| Command class | Description | Value |
|---|---|---|
| Common Commands | Commands for general data communications. | 0x00 |
| FWUP Commands | Commands for controlling updating of firmware of the secondary MCU. | 0x01 |

### 1.6.1 Common Commands

The common commands are a set of commands for general purpose use. Table 1-3 lists the commands.

**Table 1-3 List of Common Commands**

| Command | Description | Value |
|---|---|---|
| DATA_SEND: Sending data | Sends data with the desired size to the secondary MCU. | 0x01 |
| DATA_RECV: Receiving data | Requests sending of data with the desired size for the secondary MCU. | 0x02 |

(1) **DATA_SEND: Sending data**

This command sends data to the secondary MCU.

**Table 1-4 Specifications of the COMMON DATA_SEND Command**

| Item | Value |
|---|---|
| Command | 0x01 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | Desired data length which can be set according to section 2.6, Compiler Settings. |
| Response data size | 0 |
| Command data | Desired data |
| Response data | None |

(2) **DATA_RECV: Receiving data**

This command requests sending of data for the secondary MCU.

**Table 1-5 Specifications of the COMMON DATA_RECV Command**

| Item | Value |
|---|---|
| Command | 0x02 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | 0 |
| Response data size | Desired data length which can be set according to section 2.6, Compiler Settings. |
| Command data | None |
| Response data | Desired data |

### 1.6.2 FWUP Commands

FWUP commands are a group of commands used in updating of the firmware. Table 1-6 lists the FWUP commands.

**Table 1-6 List of FWUP Commands**

| Command | Description | Value |
|---|---|---|
| START: Starting of updating the firmware | Starts updating the firmware. | 0x01 |
| WRITE: Writing the updated firmware | Writes the updated firmware. | 0x02 |
| INSTALL: Installing the updated firmware | Installs and executes the updated firmware. | 0x03 |
| CANCEL: Canceling of updating the firmware | Cancels updating of the firmware. | 0x04 |
| VERSION: Confirming the Firmware Version | Confirms the currently running firmware version. | 0xF0 |

(1) **START: Starting of updating the firmware**

This command requests starting of updating the firmware of the secondary MCU.

The desired data length can be set for the command data. It is used for sending data which are required for initialization processing on the user side when updating of the firmware is started.

On reception of this command, the secondary MCU enables reception of the data for updating the firmware.

When starting of updating the firmware, send this command first.

**Table 1-7 Specifications of the FWUP START Command**

| Item | Value |
|---|---|
| Command | 0x01 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | Desired data length which can be set according to section 2.6, Compiler Settings. |
| Response data size | 0 |
| Command data | Desired data |
| Response data | None |

(2) **WRITE: Writing the updated firmware**

This command sends the data for the updated firmware to the secondary MCU and requests writing of the firmware.

The secondary MCU runs the processing for writing. It also runs signature verification processing when the data for the updated firmware are in the final block.

**Table 1-8 Specifications of the FWUP WRITE Command**

| Item | Value |
|---|---|
| Command | 0x02 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x01: Signature verification succeeded. / 0x02: Processing failed. |
| Command data size | An integer multiple of the ROM writing unit of the secondary MCU.<br>The data size can be set according to section 2.6, Compiler Settings. |
| Response data size | 0x04 |
| Command data | Data for the updated firmware |
| Response data | Size of data for the remaining updated firmware |

(3) **INSTALL: Installing the updated firmware**

This command requests installing and executing the updated firmware which has been written to the secondary MCU.

**Table 1-9 Specifications of the FWUP INSTALL Command**

| Item | Value |
|---|---|
| Command | 0x03 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | 0 |
| Response data size | 0 |
| Command data | None |
| Response data | None |

(4) **CANCEL: Canceling of updating the firmware**

This command requests canceling of updating the firmware for the secondary MCU.

The secondary MCU stops updating the firmware and erases the updated firmware that has been written.

**Table 1-10 Specifications of the FWUP CANCEL Command**

| Item | Value |
|---|---|
| Command | 0x04 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | 0 |
| Response data size | 0 |
| Command data | None |
| Response data | None |

### (5) VERSION: Confirming the Firmware Version

This command requests the current firmware version running on the secondary MCU.

**Table 1-11 Specifications of the FWUP VERSION Command**

| Item | Value |
|---|---|
| Command | 0xF0 |
| Command argument | 0x00 |
| Command result | 0x00: Processing succeeded. / 0x02: Processing failed. |
| Command data size | 0 |
| Response data size | 0 |
| Command data | None |
| Response data | Currently running firmware version |

### 1.6.2.2 Flow of Communications for the FWUP Commands

Figure 1-7 shows the flow of communications for the commands when the firmware of the secondary MCU is to be updated by using the FWUP commands.

| Primary MCU | | Secondary MCU |
|---|---|---|
| Sends the FWUP START command. | | |
| | -----> | |
| | | Receives the FWUP START command. |
| | | Makes the transition to the state for receiving updated firmware. |
| | | Sends the FWUP START response. |
| | <----- | |
| Receives the FWUP START response. | | |
| Sends the FWUP WRITE command. | | |
| | -----> | |
| | | Receives the FWUP WRITE command. |
| | | Writes the received data for the updated firmware to the ROM by using the API of FWUP FIT. |
| | | Sends the FWUP WRITE response. |
| | <----- | |
| Receives the FWUP WRITE response. | | |
| Repeats the above communications from the FWUP WRITE command until all data for the updated firmware have been received. | | |
| Sends the FWUP INSTALL command. | | |
| | -----> | |
| | | Receives the FWUP INSTALL command. |
| | | Installs the updated firmware and prepares for execution of the updated firmware after sending the response. |
| | | Sends the FWUP INSTALL response. |
| | <----- | |
| Receives the FWUP INSTALL response. | | |
| | | Executes the updated firmware. |

**Figure 1-7 Flow of Communications for the FWUP Commands**

## 1.7 Handling Errors

If the secondary MCU fails in attempting to analyze the header of a received command packet, it will send the received command header to the primary MCU. However, the command version is overwritten with that set in the secondary MCU. Also, the command data size is overwritten as 0. In this case, no processing for the command proceeds. Analyzing the header of the command packet will fail in the following cases.

- The header of the received command packet differs from the defined specifications.
- The command version of the received command packet differs from that which has been set in the secondary MCU.
- The command class or command has an undefined value.
- The size of command data corresponding to the specified command data size was not received.

The primary MCU side can detect the failure of the header analysis on the secondary MCU side by confirming that the most significant bit of command info in the received packet is “0: Command”.

## 1.8 Overview of API Functions

Table 1-12 lists the API functions included in this module.

**Table 1-12 List of API Functions**

| Function | Description |
|---|---|
| R_FWUPCOMM_Open() | Opens a communications channel for use by or within this module. |
| R_FWUPCOMM_Close() | Closes a communications channel for use by or within this module. |
| R_FWUPCOMM_ProcessCmdLoop() | Receives a command from the primary MCU, runs the corresponding handler, and sends the result of executing the command. |

# 2. API Information

Operation of this module was confirmed under the following conditions.

## 2.1 Hardware Requirements

The MCUs in use must support the following function.

- SAU

## 2.2 Software Requirements

This module depends on the following drivers.

- Board support package (r_bsp)
- UART Communication driver (Config_UART)
- SPI (CSI) Communication driver (Config_CSI)

## 2.3 Supported Toolchains

The module has been confirmed to work with the toolchains listed in section 6.1, Environments for Confirming Operation.

## 2.4 Header Files

All API calls and their supporting interface definitions are stated in r_fwupcomm_if.h.

Configuration options which can be set during building are defined in r_fwupcomm_config.h.

## 2.5 Integer Types

This module uses ANSI C99. The integer types for use are defined in stdint.h.

## 2.6 Compiler Settings

The file r_fwupcomm_config.h contains the configuration option settings for this module.

The names of the options and descriptions of their settings are listed in Table 2-1.

**Table 2-1 Configuration Settings (r_fwupcomm_config.h)**

| Configuration Option (r_fwupcomm_config.h) | |
|---|---|
| FWUPCOMM_CFG_PARAM_CHECKING_ENABLE<br>*The default setting is 0. | 0: Checking of parameters in the code at the time of building is omitted.<br>1: Checking of parameters in the code at the time of building is included.<br>Setting BSP_CFG_PARAM_CHECKING_ENABLE selects use of the default setting for the system. |
| FWUPCOMM_CFG_CH_INTERFACE<br>*The default setting is 0. | Sets the communication IP and communication method to use.<br>20: RL78/G23 CSI21<br>21: RL78/G22 CSI21<br>30: RL78/G23 UART3<br>32: RL78/G22 UART1 |
| FWUPCOMM_CFG_SPI_CSI_SO_PORTNO<br>*The default setting is “0”. | Sets the CSI SO port number for SPI communication. |
| FWUPCOMM_CFG_SPI_CSI_SO_BITNO<br>*The default setting is “7”. | Sets the pin number for the CSI SO port for SPI communication. |
| FWUPCOMM_CFG_SEND_PACKET_BUFFER_SIZE<br>*The default setting is 1048U. | Sets the size of the transmission buffer for commands. The size must be specified as 8 or greater and a multiple of 4. |
| FWUPCOMM_CFG_RECV_PACKET_BUFFER_SIZE<br>*The default setting is 1048U. | Sets the size of the reception buffer for commands. The size must be specified as 8 or greater and a multiple of 4. |
| FWUPCOMM_CFG_DEVICE_ADDRESS<br>*The default setting is 0xA0. | Sets a specific address for the device. |
| FWUPCOMM_CFG_CMD_SEND_TIMEOUT<br>*The default setting is 500U. | Sets the timeout time for sending in communications. Unit is milliseconds. |
| FWUPCOMM_CFG_CMD_RECV_TIMEOUT<br>*The default setting is 500U. | Sets the timeout time for receiving in communications. Unit is milliseconds. |
| FWUPCOMM_CFG_CMD_COMMON_ENABLE<br>*The default setting is 1. | Select whether to enable the Common command. |
| FWUPCOMM_CFG_CMD_HANDLER_COMMON<br>*The default setting is R_FWUPCOMM_CmdHandler_Common. | Sets the name of the handler function to be called when a Common command is received. |
| FWUPCOMM_CFG_CMD_HANDLER_FWUP<br>*The default setting is 1. | Select whether to enable the FWUP command. |
| FWUPCOMM_CFG_CMD_HANDLER_FWUP<br>*The default setting is R_FWUPCOMM_CmdHandler_FWUP. | Sets the name of the handler function to be called when an FWUP command is received. |
| FWUPCOMM_CFG_CMD_VER<br>*The default setting is 0x01. | Sets the version number of commands. |
| FWUPCOMM_CFG_CMD_FWUP_START_DATA_SIZE<br>*The default setting is 0U. | Sets the size of data to be included with the FWUP_START command. |
| FWUPCOMM_CFG_CMD_FWUP_WRITE_FW_BLOCK_SIZE<br>*The default setting is 1024U. | Sets the size of the block of firmware to be included with the FWUP_WRITE command. |
| FWUPCOMM_CFG_CMD_COMMON_MAX_DATA_SIZE<br>*The default setting is 12U. | Sets the maximum size of data to be included with a common command. |

## 2.7 Code Size of the Sample Projects

Table 2-2 lists the ROM and RAM sizes for the sample projects included in the package for this application note. The values in the table were confirmed under the following conditions.

Module revision: r_fwupcomm rev.1.10

Compiler versions: Renesas Electronics C/C++ Compiler for RL78 Family V1.15.00

CC-RL

- Optimization level: Code size and Speeed optimization (-Odefault)
- Delete variables or functions to which there is no reference (-optimize=symbol_delete)

**Table 2-2 ROM and RAM Sizes for the Sample Projects(Half Update Method)**

| ROM and RAM Code Sizes | | | |
|---|---|---|---|
| **Device** | **Category** | **Memory Used (Byte)** | **Project Name** |
| | | **CC-RL** | |
| RL78/G23 | ROM | 30067 | app_rl78g23_fpb_w_buffer |
| | | 20357 | bootloader_rl78g23_fpb_w_buffer |
| | RAM | 4189 | app_rl78g23_fpb_w_buffer |
| | | 1338 | bootloader_rl78g23_fpb_w_buffer |
| RL78/G22 | ROM | 13551 | app_rl78g22_fpb_w_buffer |
| | | 12596 | bootloader_rl78g22_fpb_w_buffer |
| | RAM | 2315 | app_rl78g22_fpb_w_buffer |
| | | 783 | bootloader_rl78g22_fpb_w_buffer |

**Table 2-3 ROM and RAM Sizes for the Sample Projects(Full Update Method)**

| ROM and RAM Code Sizes | | | |
|---|---|---|---|
| **Device** | **Category** | **Memory Used (Byte)** | **Project Name** |
| | | **CC-RL** | |
| RL78/G23 | ROM | 7263 | app_rl78g23_fpb_wo_buffer |
| | | 33134 | bootloader_rl78g23_fpb_wo_buffer |
| | RAM | 3596 | app_rl78g23_fpb_wo_buffer |
| | | 4187 | bootloader_rl78g23_fpb_wo_buffer |
| RL78/G22 | ROM | 9108 | app_rl78g22_fpb_wo_buffer |
| | | 15287 | bootloader_rl78g22_fpb_wo_buffer |
| | RAM | 2338 | app_rl78g22_fpb_wo_buffer |
| | | 2317 | bootloader_rl78g22_fpb_wo_buffer |

## 2.8 Arguments

This section shows the definitions of structures and enumerated types that are used as arguments of the API functions. The definitions of these types are described in r_fwupcomm_if.h, along with the prototype declarations of the API functions.

```
/* Structure used for registering a timer interface */
typedef struct r_fwupcomm_timer
{
   r_fwupcomm_start_timer_t start; // Pointer to the function to start counting by a timer
   r_fwupcomm_stop_timer_t stop;   // Pointer to the function to stop counting by a timer
} r_fwupcomm_timer_t;
```

```
/* Structure used as an argument of the Open function during initialization */
typedef struct r_fwupcomm_cfg
{
    r_fwupcomm_timer_t timer; // Timer interface
} r_fwupcomm_cfg_t;
```

```
/* Structure for specifying command information */
struct r_fwupcomm_cmd_info
{
    uint8_t device_address; // Address of the destination device for a command
    uint8_t class;          // Command class
    uint8_t type;           // Command
    uint8_t arg;            // Command argument
    uint16_t data_size;     // Command data size
    const void *data;       // Pointer to command data
    uint8_t id;             // Command ID
};
```

```
/* Structure for storing response information */
struct r_fwupcomm_resp_info
{
    int8_t result;       // Command result
    void *data;          // Pointer to the destination for storing response data
    uint16_t data_size;  // Size of the destination for storing response data
};
```

```
/* Structure used as an argument of the CmdSend function when a command is to be sent */
struct r_fwupcomm_cmd_instr
{
   uint16_t timeout_ms;         // Timeout time from sending the command to receiving the response
   r_fwupcomm_cmd_info_t cmd;    // Command information
   r_fwupcomm_resp_info_t resp;  // Destination for storing response information
};
```

```
/* Enumerated type for defining the command classes */
typedef enum
{
    FWUPCOMM_CMD_CLS_COMMON = (0),    // Common command
    FWUPCOMM_CMD_CLS_FWUP = (1),      // FWUP command
} r_fwupcomm_cmd_class_t;
```

```
/* Enumerated type for defining commands of the common command class */
typedef enum
{
    FWUPCOMM_CMD_COMMON_DATA_SEND = (0),  // DATA_SEND command
    FWUPCOMM_CMD_COMMON_DATA_RECV,        // DATA_RECV command
    FWUPCOMM_CMD_COMMON_NUM_COMMANDS      // Number of defined common commands
} r_fwupcomm_cmd_type_common_t;
```

```
/* Enumerated type for defining commands of the FWUP command class */
typedef enum
{
    FWUPCOMM_CMD_FWUP_START = (0),        // START command
    FWUPCOMM_CMD_FWUP_WRITE,              // WRITE command
    FWUPCOMM_CMD_FWUP_INSTALL,           // INSTALL command
    FWUPCOMM_CMD_FWUP_CANCEL,            // CANCEL command
    FWUPCOMM_CMD_FWUP_VERSION,           // VERSION command
    FWUPCOMM_CMD_FWUP_NUM_COMMANDS       // Number of defined FWUP commands
} r_fwupcomm_cmd_type_fwup_t;
```

## 2.9 Return Values

This section describes the return values of the API functions. The enumerated type is defined in r_fwupcomm_if.h, along with the prototype declarations of the API functions.

```
typedef enum
{
    FWUPCOMM_SUCCESS = 0,
    FWUPCOMM_ERR_INVALID_PTR,       // The pointer passed as an argument was NULL.
    FWUPCOMM_ERR_INVALID_ARG,       // The parameter passed as an argument was invalid.
    FWUPCOMM_ERR_NOT_OPEN,          // The module has not been opened.
    FWUPCOMM_ERR_ALREADY_OPEN,      // The module has already been initialized.
    FWUPCOMM_ERR_INVALID_CMD,       // An invalid command was received.
    FWUPCOMM_ERR_INVALID_RESP,      // The received response was invalid.
    FWUPCOMM_ERR_RECV_RESP_TIMEOUT, // A timeout occurred before a response was received.
    FWUPCOMM_ERR_NO_CMD,            // No command was received.
    FWUPCOMM_ERR_CH_ALREADY_OPEN,   // The communications channel has already been opened.
    FWUPCOMM_ERR_CH_SEND,           // Sending of data in the communications channel failed.
    FWUPCOMM_ERR_CH_SEND_BUSY,      // The communications channel was busy so sending of data failed.
    FWUPCOMM_ERR_CH_RECV,           // Receiving of data from the communications channel failed.
    FWUPCOMM_ERR_CH_RECV_NO_DATA,    // The communications channel does not have enough received data.
} fwupcomm_err_t;
```

## 2.10 “for”, “while” and “do while” Statements

In this module, “for”, “while”, and “do while” statements (loop processing) are used in processing to wait for registers to reflect written values and so on. For such loop processing, the comment “WAIT_LOOP” is written as a keyword. Therefore, if the user wishes to incorporate fail-safe processing into the loop processing, the user can search for the corresponding processing by using “WAIT_LOOP”.

The following listings are examples of such loop processing.

```
while statement example:
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

```
for statement example:
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

```
do while statement example:
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET));
/* WAIT_LOOP */
```

# 3. API Functions

## 3.1 R_FWUPCOMM_Open Function

**Table 3-1 Specifications of the R_FWUPCOMM_Open Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t R_FWUPCOMM_Open(r_fwupcomm_hdl_t *hdl, void *cfg) | |
| Description | Opens a communications channel for use by or within this module. This function must be executed before other API functions are used. | |
| Parameters | hdl: Handler of the module<br>cfg: Structure variable with information required for initializing modules | |
| Return Values | FWUPCOMM_SUCCESS | The channel was successfully initialized. |
| | FWUPCOMM_ERR_INVALID_PTR | The pointer passed as an argument was NULL. |
| | FWUPCOMM_ERR_ALREADY_OPEN | Opening has already proceeded. |
| | FWUPCOMM_ERR_CH_ALREADY_OPEN | The communications channel has already been opened. |
| | FWUPCOMM_ERR_NOT_OPEN | Initializing the communications channel failed. |
| Special Notes | — | |

Example:

```
fwupcomm_err_t fwupcomm_err;
r_fwupcomm_hdl_t fwupcomm_hdl = {0};
r_fwupcomm_cfg_t fwupcomm_cfg;
fwupcomm_cfg.timer.start = demo_start_timer;
fwupcomm_cfg.timer.stop = demo_stop_timer;

fwupcomm_err = R_FWUPCOMM_Open(&fwupcomm_hdl, &fwupcomm_cfg);
```

## 3.2 R_FWUPCOMM_Close Function

**Table 3-2 Specifications of the R_FWUPCOMM_Close Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t R_FWUPCOMM_Close(r_fwupcomm_hdl_t *hdl) | |
| Description | Closes a communications channel for use by or within this module. | |
| Parameters | hdl: Handler of the module | |
| Return Values | FWUPCOMM_SUCCESS | Closing was successful. |
| | FWUPCOMM_ERR_NOT_OPEN | The module has not been opened. |
| | FWUPCOMM_ERR_INVALID_PTR | The pointer passed as an argument is NULL. |
| Special Notes | — | |

Example:

```
fwupcomm_err = R_FWUPCOMM_Close(&fwupcomm_hdl);
```

## 3.3 R_FWUPCOMM_ProcessCmdLoop Function

**Table 3-3 Specifications of the R_FWUPCOMM_ProcessCmdLoop Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t R_FWUPCOMM_ProcessCmdLoop(r_fwupcomm_hdl_t *hdl) | |
| Description | Receives a command from the primary MCU, runs the corresponding handler, and sends the result of executing the command. Periodically execute this function in the secondary MCU while it is waiting for commands. | |
| Parameters | hdl: Handler of the module | |
| Return Values | FWUPCOMM_SUCCESS | The channel was successfully initialized. |
| | FWUPCOMM_ERR_NOT_OPEN | The module has not been opened. |
| | FWUPCOMM_ERR_INVALID_PTR | The pointer passed as an argument was NULL. |
| | FWUPCOMM_ERR_INVALID_ARG | The parameter passed as an argument was invalid. |
| | FWUPCOMM_ERR_NO_CMD | No command was received. |
| | FWUPCOMM_ERR_INVALID_CMD | An invalid command was received. |
| | FWUPCOMM_ERR_CH_SEND | Sending of data in the communications channel failed. |
| | FWUPCOMM_ERR_CH_RECV | Receiving of data from the communications channel failed. |
| Special Notes | — | |

Example:

```
do
{
    fwupcomm_err = R_FWUPCOMM_ProcessCmdLoop(&fwupcomm_hdl);
}while((FWUPCOMM_SUCCESS == fwupcomm_err)||(FWUPCOMM_ERR_NO_CMD == fwupcomm_err));
```

# 4. Extending the Functionality of This Module

This chapter describes how to add commands to this module and change the method of communications.

## 4.1 Adding Commands

This section describes how to define desired commands in addition to the FWUP and common commands which have already been defined for this module. Here, ADDITIONAL1 and ADDITIONAL2 commands having the UserDefined command class name are added as an example.

(1) Create a source file such as r_fwupcomm_cmd_user_defined.c and a header file such as r_fwupcomm_cmd_user_defined.h.
Include the r_fwupcomm_if.h header file and also include the header file of the created UserDefined commands in the source file.

(2) Create an enumerated type for defining the UserDefined commands, such as r_fwupcomm_cmd_class_user_defined_t shown below, in the header file and define enumerators to indicate the ADDITIONAL1 and ADDITIONAL2 commands. Define an enumerator to indicate the number of elements as the last enumerator of the enumerated type.

```
typedef enum
{
    FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1,
    FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2,
    FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS
} r_fwupcomm_cmd_class_user_defined_t;
```

(3) Define an array of the r_fwupcomm_cmd_table_t type in the source file and place information on the ADDITIONAL1 and ADDITIONAL2 commands as the two elements of the array.

```
const r_fwupcomm_cmd_table_t
r_fwupcomm_user_defined_cmd_table[FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS] =
{
  { FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1, 0x01, 0U, 0U },
  { FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2, 0x02, 0U, 0U }
};
```

The r_fwupcomm_cmd_table_t type is a structure defined in r_fwupcomm_if.h. Each of the members is defined as follows.

```
typedef struct r_fwupcomm_cmd_table
{
    uint8_t type;                 // Value indicating this command (enumerator)
    uint8_t value;                // Actual value used for communications by this command
    uint16_t cmd_data_max_size;   // Maximum size of the command data of this command
    uint16_t resp_data_max_size;  // Maximum size of the response data of this command
} r_fwupcomm_cmd_table_t;
```

(4) Define the handler function which describes the processing to be executed when the secondary MCU receives the UserDefined command in the source file.
The pointer variable of the r_fwupcomm_cmd_info_t type contains the information on the received command such as pointers to the command arguments or command data. Refer to such command information to run the processing within the handler function. After that, store the information on responses to be sent to the primary MCU (command results, pointer to the response data, and response data size) in a pointer variable of the r_fwupcomm_resp_info_t type as the argument.

```
void R_FWUPCOMM_CmdHandler_UserDefined(r_fwupcomm_cmd_info_t *cmd,
                                       r_fwupcomm_resp_info_t *resp)
{
    if((NULL == cmd)||(NULL == resp))
    {
        return;
    }

    if(cmd->type >= FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS)
    {
        return;
    }

    switch(cmd->type)
    {
        case FWUPCOMM_CMD_USERDEFINED_ADDITIONAL1:
            /* Describe the processing to be executed upon receiving the ADDITIONAL1 command. */
            break;
        case FWUPCOMM_CMD_USERDEFINED_ADDITIONAL2:
            /* Describe the processing to be executed upon receiving the ADDITIONAL2 command. */
            break;
}
```

(5) Declare an array of the r_fwupcomm_cmd_table_t type for the UserDefined command, which was previously defined in the source file, in the header file as extern. Similarly, write a prototype declaration for the handler function of the UserDefined command.

```
extern const r_fwupcomm_cmd_table_t r_fwupcomm_user_defined_cmd_table
[FWUPCOMM_CMD_COMMON_NUM_COMMANDS];


#if FWUPCOMM_CFG_DEVICE_PRIMARY == (0)  // Macro which enables only the secondary MCU
void R_FWUPCOMM_CmdHandler_UserDefined (r_fwupcomm_cmd_info_t *cmd,
r_fwupcomm_resp_info_t *resp);
#endif
```

(6) Include the header file for the UserDefined command in the r_fwupcomm¥src¥commands¥r_fwupcomm_cmd.h file.

```
#include "r_fwupcomm_cmd_common.h"
#include "r_fwupcomm_cmd_fwup.h"
#include "r_fwupcomm_cmd_user_defined.h"
```

(7) Enter the total number of command classes after adding the UserDefined command into the FWUPCOMM_CMD_NUM_CLASS macro defined in the r_fwupcomm_cmd.h file.

```
#define FWUPCOMM_CMD_NUM_CLASS    (FWUPCOMM_CFG_CMD_COMMON_ENABLE +
FWUPCOMM_CFG_CMD_FWUP_ENABLE + 1)
```

(8) Add an enumerator indicating the UserDefined command to the r_fwupcomm_cmd_class_t enumerated type which is defined in the r_fwupcomm_cmd.h file.

```
typedef enum
{
    FWUPCOMM_CMD_CLS_COMMON = (0),
    FWUPCOMM_CMD_CLS_FWUP = (1),
    FWUPCOMM_CMD_CLS_USERDEFINED = (2)
} r_fwupcomm_cmd_class_t;
```

(9) Add the UserDefined command to the array of the r_fwupcomm_cmd_def_table_t type which is defined in the r_fwupcomm_cmd.c file.

```
const r_fwupcomm_cmd_def_table_t r_fwupcomm_cmd_def_table_list[] =
{
  [FWUPCOMM_CMD_CLS_COMMON] = {r_fwupcomm_common_cmd_table, FWUPCOMM_CMD_COMMON_NUM_COMMANDS},
  [FWUPCOMM_CMD_CLS_FWUP] = {r_fwupcomm_fwup_cmd_table, FWUPCOMM_CMD_FWUP_NUM_COMMANDS},
  [FWUPCOMM_CMD_CLS_USERDEFINED] = {r_fwupcomm_user_defined_cmd_table,
                                    FWUPCOMM_CMD_USERDEFINED_NUM_COMMANDS}
};
```

As stated, the r_fwupcomm_cmd_def_table_t type is defined in r_fwupcomm_cmd.h. Specify the array of the r_fwupcomm_cmd_table_t type defined in the source file as the table member. Specify the number of commands in that command class as the num_cmd member.

```
typedef struct
{
    const r_fwupcomm_cmd_table_t *table;
    uint8_t num_cmd;
} r_fwupcomm_cmd_def_table_t;
```

(10) Add the handler functions of the UserDefined command defined in the source file to the array of the R_FWUPCOMM_CmdHandler_t type which is defined in the r_fwupcomm_cmd.c file.

```
#if FWUPCOMM_CFG_DEVICE_PRIMARY == (0)    // Macro which enables only the secondary MCU
const R_FWUPCOMM_CmdHandler_t r_fwupcomm_cmd_handler_list[FWUPCOMM_CMD_NUM_CLS] =
{
    [FWUPCOMM_CMD_CLS_COMMON] = FWUPCOMM_CFG_CMD_HANDLER_COMMON,
    [FWUPCOMM_CMD_CLS_FWUP] = FWUPCOMM_CFG_CMD_HANDLER_FWUP,
    [FWUPCOMM_CMD_CLS_USERDEFINED] = R_FWUPCOMM_CmdHandler_UserDefined
};
#endif
```

The steps described above are used for adding commands. For further information, refer to the definition files for the FWUP commands (r_fwupcomm_cmd_fwup.c and r_fwupcomm_cmd_fwup.h) and for the common commands (r_fwupcomm_cmd_common.c and r_fwupcomm_cmd_common.h) in the r_fwupcomm¥src¥commands folder.

## 4.2 Changing the Method of Communications

This module only supports UART communications via the SAU. This section describes how to change to another method of communications.

### 4.2.1 Communications Interface

This module specifies the communications interface for packet communications. It is defined in r_fwupcomm¥src¥connectivity¥r_fwupcomm_ch.h as follows.

```
typedef struct r_fwupcomm_ch_api
{
    fwupcomm_err_t (*open)(void);
    void (*close)(void);
    fwupcomm_err_t (*send)(uint8_t *src, uint16_t size);
    fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size);
    void (*rx_reset)(void);
} r_fwupcomm_ch_api_t;
```

#### 4.2.1.1 fwupcomm_err_t (*open)(void)

**Table 4-1 Specifications of the open Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t (*open)(void) | |
| Description | Opens a communications channel. | |
| Parameters | — | |
| Return Values | FWUPCOMM_SUCCESS | The channel was successfully initialized. |
| | FWUPCOMM_ERR_CH_ALREADY_OPEN | The communications channel has already been opened. |
| | FWUPCOMM_ERR_NOT_OPEN | Initializing the communications channel failed. |
| Special Notes | — | |

#### 4.2.1.2 void (*close)(void)

**Table 4-2 Specifications of the close Function**

| | |
|---|---|
| Format | void (*close)(void) |
| Description | Closes a communications channel. |
| Parameters | — |
| Return Values | — |
| Special Notes | — |

#### 4.2.1.3 fwupcomm_err_t (*send)(uint8_t *src, uint16_t size)

**Table 4-3 Specifications of the send Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t (*send)(uint8_t *src, uint16_t size) | |
| Description | Sends data by using a communications channel. | |
| Parameters | src: Pointer to the destination for storing data to be sent<br>size: Size of data to be sent | |
| Return Values | FWUPCOMM_SUCCESS | The channel was successfully initialized. |
| | FWUPCOMM_ERR_INVALID_PTR | The src pointer is NULL. |
| | FWUPCOMM_ERR_INVALID_ARG | size is 0. |
| | FWUPCOMM_ERR_NOT_OPEN | The communications channel has not been opened. |
| | FWUPCOMM_ERR_CH_SEND_BUSY | The communications channel was busy so sending of data failed. |
| | FWUPCOMM_ERR_CH_SEND | Sending of data in the communications channel failed. |
| Special Notes | — | |

#### 4.2.1.4 fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size)

**Table 4-4 Specifications of the recv Function**

| | | |
|---|---|---|
| Format | fwupcomm_err_t (*recv)(uint8_t *dest, uint16_t size) | |
| Description | Receives data by using a communications channel. | |
| Parameters | dest: Pointer to the buffer for storing received data<br>size: Required size of received data | |
| Return Values | FWUPCOMM_SUCCESS | The channel was successfully initialized. |
| | FWUPCOMM_ERR_INVALID_PTR | The dest pointer is NULL. |
| | FWUPCOMM_ERR_INVALID_ARG | size is 0. |
| | FWUPCOMM_ERR_NOT_OPEN | The communications channel has not been opened. |
| | FWUPCOMM_ERR_CH_RECV_NO_DATA | The communications channel does not have enough received data. |
| Special Notes | — | |

#### 4.2.1.5 void (*rx_reset)(void)

**Table 4-5 Specifications of the rx_reset Function**

| | |
|---|---|
| Format | void (*rx_reset)(void) |
| Description | Enables the communication channel for reception. |
| Parameters | — |
| Return Values | — |
| Special Notes | — |

### 4.2.2 How to Change the Method of Communications

(1) Implement the functions for communications interfaces described in section 4.2.1 by using the method of communications you wish to use.

(2) Define the r_fwupcomm_ch_api variable of the const r_fwupcomm_ch_api_t type, and initialize the functions which have been created for the communications interface as shown below.

```
const r_fwupcomm_ch_api_t r_fwupcomm_ch_api =
{ .open = r_fwupcomm_rx_sci_uart_open,          // open
  .close = r_fwupcomm_rx_sci_uart_close,        // close
  .send = r_fwupcomm_rx_sci_uart_send,          // send
  .recv = r_fwupcomm_rx_sci_uart_recv,          // recv
  .rx_reset = r_fwupcomm_rx_sci_uart_rx_reset   // rx_reset
};
```

(3) Create a header file with a name such as r_fwupcomm_ch_user_defined.h to declare the r_fwupcomm_ch_api variable as extern.

```
extern r_fwupcomm_ch_api_t const r_fwupcomm_ch_api;
```

(4) Add the definition of the communications interface to the r_fwupcomm¥src¥r_fwupcomm_private.h file in such a way that the newly created header file is included instead of the one that has been previously created.

```
#define FWUPCOMM_CFG_CH_INTERFACE          (2)

#if   (FWUPCOMM_CFG_CH_INTERFACE == 1)  /* RX SCI UART */
#define FWUPCOMM_CH_RX_SCI_UART            (FWUPCOMM_CFG_CH_INTERFACE)
#define FWUPCOMM_COMM_IF                  (FWUPCOMM_COMM_IF_UART)
#elif (FWUPCOMM_CFG_CH_INTERFACE == 2) /* USERDEFINED */
#define FWUPCOMM_CH_USERDEFINED            (FWUPCOMM_CFG_CH_INTERFACE)
...
#endif

#define FWUPCOMM_USE_CH                    (FWUPCOMM_CFG_CH_INTERFACE)

#if (FWUPCOMM_USE_CH == FWUPCOMM_CH_RX_SCI_UART)
  #include "r_fwupcomm_rx_sci_uart.h"
#elif (FWUPCOMM_USE_CH == FWUPCOMM_CH_USERDEFINED)
  #include "r_fwupcomm_ch_user_defined.h"
#endif
```

That ends the description of how to change the method of communications.

# 5. Demonstration Projects

This demonstration projects are sample programs for updating the firmware of the secondary MCU, as shown in Figure 5-1. The primary MCU is connected to a PC and receives the firmware for use in updating that of the secondary MCU via serial communications from the PC. The primary MCU then transfers that firmware to the secondary MCU by using the FWUP Comm module.

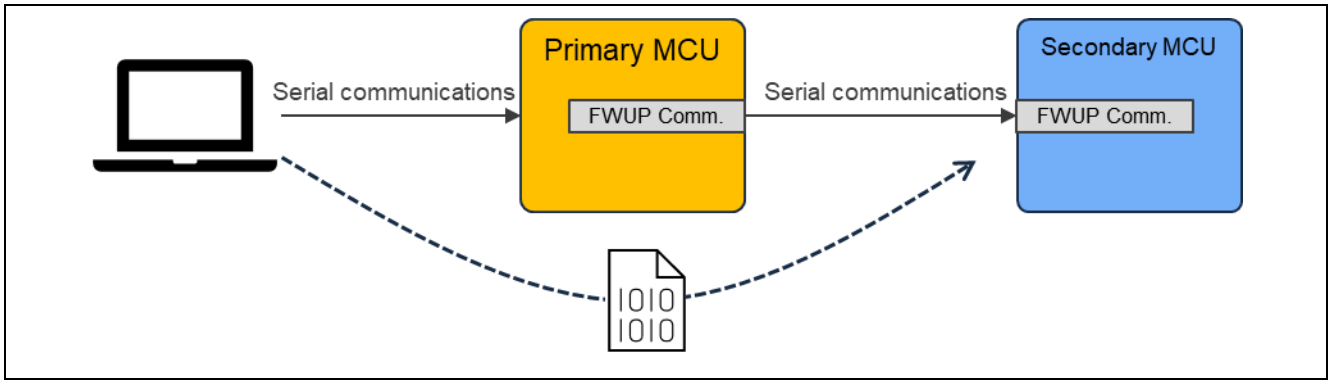


**Figure 5-1 Configuration of the Demonstrations**

## 5.1 Configuration for the Demonstration Projects

### 5.1.1 Primary MCU

The primary MCU demo project is only for the RX65N and is organized into folders as follows.

Demos¥rx65n-ck¥(compiler name)¥(project name)

Boot loader projects:

- Partial update method in linear mode : bootloader_rx65n_ck_w_buffer

Application projects:

Partial update method in linear mode: app_rx65n_ck_primary

### 5.1.2 Secondary MCU

Demonstration projects in the secondary MCU are classified into folders for each of the supported device groups.

- Partial update method in linear mode: Demos¥(board name)¥w_buffer¥(project name)
- Full update method in linear mode: Demos¥(board name)¥wo_buffer¥(project name)

Boot loader projects:

- Partial update method in linear mode: bootloader_(board name)_w_buffer
- Full update method in linear mode: bootloader_(board name)_wo_buffer

Application projects:

- Partial update method in linear mode: app_(board name)_w_buffer
- Full update method in linear mode: app_(board name)_wo_buffer

## 5.2 Preparing an Operating Environment

To update the firmware of the secondary MCU, use the firmware updating module. To run the demonstration projects, you need to install certain tools on your Windows PC.

### 5.2.1 Installing TeraTerm

TeraTerm is used to transfer the firmware updating image via serial communications from a Windows PC to the primary MCU. For the demonstration project, the operation was confirmed with TeraTerm 5.5.0.

After installation, make the serial port communications settings listed in Table 5-1.

**Table 5-1 Specifications for Communications**

| Item | Description |
|---|---|
| Communications system | Asynchronous |
| Bit rate | 115200 bps |
| Data length | 8 bits |
| Parity | None |
| Stop bit | 1 bit |
| Flow control | RTS/CTS |

### 5.2.2 Installing the Python Execution Environment

The Python execution environment is used by Renesas Image Generator (image-gen.py) to create the initial and updating images.

Renesas Image Generator uses ECDSA to generate signature data. For the demonstration project, the operation was confirmed with Python 3.10.4.

The Python encryption library (pycryptodome) is also used. Accordingly, after installing Python, execute the following pip command from the command prompt to install the library.

```
pip install pycryptodome
```

### 5.2.3 Installing the Flash Writer

A flash writer is required to write the initial image.

Renesas Flash Programmer V3.21.00 is used with the demonstration projects.

Renesas Flash Programmer (Programming GUI) | Renesas

## 5.3 Procedure for Executing a Demonstration Project

This section describes an example of the procedure for executing a demonstration project with the use of an RL78/G23 device. The procedure for executing the demonstration project is common to other MCU products; however, only the environment for confirming the operation differs with the MCU. Confirm the environment (section 6.1 Environments for Confirming Operation) for the MCU product you intend to use.

The following describes the procedure for executing UART communication using the FWUPCOMM FIT module. For SPI communication settings, refer to “5.5 Settings for the Demo Project When Using SPI Communication Between MCUs.”

### 5.3.1 Execution Environment

Prepare the environment for confirming the operation with an RL78/G23 (6.3.1).

### 5.3.2 Building the Demonstration Projects

Follow the steps below to build the projects for the primary and secondary MCUs.

#### 5.3.2.1 Creating Initial and Updating Images for the Primary MCU

The procedure for creating the initial and updating images, using initial_firm_rx65n.mot as the name of the initial image and update_firm_rx65n.rsu as the name of the updating image, is described below.

(1) Import the bootloader_rx65n_ck_w_buffer and app_rx65n_ck_primary project into the e$^2$ studio and build the project. For the full update method, change the “APP_COMM_CONFIG_FWUP_FULL_UPDATE” macro definition to (1) in app_rx65n_ck_primary¥src¥fwup¥app_fwup_config.h before the build.

```
17          /*
18           * Update Method for Secondary MCU
19           *  0: Half Update
20           *  1: Full Update
21           */
22          #define APP_COMM_CONFIG_FWUP_FULL_UPDATE        (1)
```

(2) Confirm that the following MOT file has been generated in the HardwareDebug folder for each project.
- bootloader_rx65n_ck_w_buffer.mot
- app_rx65n_ck_primary.mot

(3) Store the MOT files created by building the demonstration project in the bootloader_rx65n_ck_w_buffer¥src¥smc_gen¥r_fwup¥tool folder. Also store the FITDemos¥keys¥fwup¥secp256r1.privatekey file there as well.

```
image-gen.py
RX65N_Linear_Half_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rx65n_ck_w_buffer.mot
app_rx65n_ck_primary.mot
```

(4) Execute the following command in the bootloader_rx65n_ck_w_buffer¥src¥smc_gen¥r_fwup¥tool folder to create the initial image.

```
python .¥image-gen.py -iup ".¥app_rx65n_ck_primary.mot" -
ip .¥RX65N_Linear_Half_ImageGenerator_PRM.csv -o initial_firm_rx65n -ibp
".¥bootloader_rx65n_ck_w_buffer.mot" -vt ecdsa -key ".¥secp256r1.privatekey"
```

(5) app_rx65n_ck_primary¥src¥app_rx65n_ck_primary.h file. Change the definition of DEMO_VER_MAJOR from (1) to (2) and rebuild the app_rx65n_ck_primary project. After that, store the MOT files created by building the project in the tool folder.

```
41      /* FW Version definition */
42      #define DEMO_VER_MAJOR          (1)
43      #define DEMO_VER_MINOR          (0)
44      #define DEMO_VER_BUILD          (0)
```

(6) Execute the following command to create the updating image.

```
python .¥image-gen.py -iup ".¥app_rx65n_ck_primary.mot" -
ip .¥RX65N_Linear_Half_ImageGenerator_PRM.csv -o update_firm_rx65n -vt ecdsa -key
".¥secp256r1.privatekey"
```

Confirm that the initial and updating images have been generated in the tool folder.

Image-gen.py
RX65N_Linear_Half_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rx65n_ck_w_buffer.mot
app_rx65n_ck_w_primary.mot
initial_firm_rx65n.mot
update_firm_rx65n.rsu

### 5.3.2.2 Creating Initial and Updating Images for the Secondary MCU

The procedure for creating the initial and updating images, using initial_firm_rl78g23.mot as the name of the initial image and update_firm_rl78g23.mot as the name of the updating image, is described below. This is the procedure for the partial update method, but the procedure is the same for the full update method, so please replace projects used with those for the full update method.

(1) Import the bootloader_rl78g23_fpb_w_buffer and app_rl78g23_fpb_w_buffer projects into the e$^2$ studio and build the projects.
(2) Confirm that the following MOT files have been generated in the HardwareDebug folder for each project.
   — bootloader_rl78g23_fpb_w_buffer.mot
   — app_rl78g23_fpb_w_buffer.mot
(3) Store the MOT files created by building the demonstration project in the Demos¥RenesasImageGenerator folder. Also store the Demos¥keys¥secp256r1.privatekey file there as well.

image-gen.py
RL78_G23_Full_ImageGenerator_PRM.csv
RL78_G23_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rl78g23_fpb_w_buffer.mot
app_rl78g23_fpb_w_buffer.mot

(4) Execute the following command in the Demos¥RenesasImageGenerator folder to create the initial image. For the full update method, use RL78_G23_ImageGenerator_PRM.csv instead of RL78_G23_Full_ImageGenerator_PRM.csv.

```
python .¥image-gen.py -iup ".¥app_rl78g23_fpb_w_buffer.mot" -
ip .¥RL78_G23_ImageGenerator_PRM.csv -o initial_firm_rl78g23 -ibp
".¥bootloader_rl78g23_fpb_w_buffer.mot" -vt ecdsa -key ".¥secp256r1.privatekey"
```

(5) Open the app_rl78g23_fpb_w_buffer¥src¥fwupcomm_demo_main.h file. Change the definition of DEMO_VER_MAJOR from (1) to (2) and rebuild the app_rl78g23_fpb_w_buffer project. After that, store the MOT files created by building the project in the tool folder.

```
51  #define DEMO_VER_MAJOR      (2)
52  #define DEMO_VER_MINOR      (0)
53  #define DEMO_VER_BUILD      (0)
```

(6) Execute the following command to create the updating image. For the full update method, use RL78_G23_Full_ImageGenerator_PRM.csv instead of RL78_G23_ImageGenerator_PRM.csv.

```
python .¥image-gen.py -iup ".¥app_rl78g23_fpb_w_buffer.mot" -
ip .¥RL78_G23_ImageGenerator_PRM.csv -o update_firm_rl78g23 -vt ecdsa -key
".¥secp256r1.privatekey"
```

Confirm that the initial and updating images have been generated in the RenesasImageGenerator folder.

```
image-gen.py
RL78_G23_Full_ImageGenerator_PRM.csv
RL78_G23_ImageGenerator_PRM.csv
secp256r1.privatekey
bootloader_rl78g23_fpb_w_buffer.mot
app_rl78g23_fpb_w_buffer.mot
initial_firm_rl78g23.mot
update_firm_rl78g23.rsu
```

### 5.3.3 Programming the Initial Image

Use the flash writer to program initial_firm_rx65n.mot to the MCU on the CK-RX65Nv2 board.

Similarly, use the flash writer to program the initial image (initial_firm_rl78g23.mot) to the MCU on the RL78/G23-128p FPB board. After programming is finished, turn off the power to the board.

### 5.3.4 Executing a Firmware Update

Once the initial image firmware has been activated, it waits for the transfer of the updating image through the primary MCU. The received updating image is programmed to the flash memory, and after the transfer is completed, the signature of the updating image is verified and the firmware is activated.

Follow the steps below to execute a firmware update.

(1) Launch two TeraTerm windows on the PC, select the serial COM ports for the primary MCU (CK-RX65Nv2) and the secondary MCU (RL78/G23-128p FPB) in the respective windows, and configure the connection settings.

(2) Turn on the board. The following messages will be output to the TeraTerm windows.

Primary MCU side:

```
==== RX65N : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : FWUPCOMM DEMO [Primary][with buffer]  ver 1.0.0 ====
Please select the target MCU to update firmware.
  0: Primary MCU
  1: Secondary MCU

>
```

Secondary MCU side:

```
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...
==== RL78/G23 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 1.0.0 ====
```

(3) On the primary MCU's TeraTerm screen, enter the number of the MCU to be updated.
(4) Send the updating image via TeraTerm.
Click on [Send file] from the [File] menu of TeraTerm for the primary MCU side. Select update_firm_rl78g23.rsu then [Binary] as the option and click on [OK].
The following messages are output during the transfer of the updating image, a software reset is applied after installation and signature verification are completed, and the firmware from the updating image is executed.
The version number output in the last message from the targeted MCU having been incremented indicates that the upedate was successful.

The following is an example of log output when the secondary MCU (RL78/G23-128p FPB) is the target for the firmware update.

Primary MCU side:

```
Please send the firmware for secondary MCU
[S]Received 512 bytes. total 512 bytes.
Confirmed the Secondary ready state.
Send FWUP_START command... OK.
Confirmed the Secondary ready state.
Send FWUP_WRITE command... OK. (512 bytes sent, remaining 4294966784 bytes.)
[S]Received 512 bytes. total 1024 bytes.
Send FWUP_WRITE command... OK. (512 bytes sent, remaining 8448 bytes.)
...
[S]Received 512 bytes. total 9216 bytes.
Send FWUP_WRITE command... OK. (512 bytes sent, remaining 256 bytes.)
[S]Received 256 bytes. total 9472 bytes.
Send FWUP_WRITE command... OK. All data sent. Verification succeeded.
Send FWUP_INSTALL command... OK.
Firmware update for the device(0xA0) is successful.
```

Secondary MCU side:

```
Received FWUPCOMM_CMD_FWUP_START command.
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x1000, 512 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x1200, 256 ... OK
W 0x1300, 256 ... OK
...
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x4C00, 512 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=256
W 0x4E00, 256 ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
Received FWUPCOMM_CMD_FWUP_INSTALL command.
software reset...

==== RL78G23 : BootLoader [with buffer] ====
verify install area buffer [sig-sha256-ecdsa]...OK
copy to main area ... OK
software reset...
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...
==== RL78/G23 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 2.0.0 ====
```

## 5.4 Procedure for Executing the Demo Project When the Communication Method Between the PC and Primary MCU is XMODEM

This demo project uses UART binary data communication between the PC and primary MCU as the default communication method.

The following describes the procedure for using XMODEM.

(1) Connect the primary MCU CK-RX65Nv2 as described in “6.3.3.1 Connection Configuration When the Communication Method Between the PC and Primary MCU is XMODEM”.

(2) Set APP_COMM_CONFIG_PROTOCOL defined in src/comm/app_comm_config.h of the primary MCU's application project to (2).

(3) Perform the procedures in “5.3.2 Building the Demonstration Projects” and “5.3.3 Programming the Initial Image”.

(4) In “5.3.4 Executing a Firmware Update,” launch TeraTerm in three windows. In addition to the serial COM ports for the primary MCU (CK-RX65Nv2) and secondary MCU (RL78/G23-128p FPB), select the serial COM port for the additionally connected USB terminal and configure the connection settings.

(5) Power on the board. The following messages will be displayed in TeraTerm.

Primary MCU side:

```
==== RX65N : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : FWUPCOMM DEMO [Primary][with buffer]  ver 1.0.0 ====
Please select the target MCU to update firmware.
  0: Primary MCU
  1: Secondary MCU

>
```

Secondary MCU side:

```
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...
==== RL78/G23 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 1.0.0 ====
```

(6) On the primary MCU's TeraTerm window, enter the number of the MCU to be updated with the new firmware.

(7) Send the update image from TeraTerm.
From the [File] menu in TeraTerm on the primary MCU, click [Transfer] → [XMODEM] → [Send]. Select update_firm_rx65n.rsu for updating the primary MCU's firmware, or update_firm_rl78g23.rsu for updating the secondary MCU's firmware, then click [Open]. It may take several seconds for the update firmware transmission to begin. Note: This demo project does not support transfers with a 1K byte block size.
During the update image transfer, progress is output to the TeraTerm on the additionally connected serial COM port. Once installation and signature verification complete, a software reset occurs, and the firmware from the update image begins execution.
If the version number output in the message on the MCU targeted for the firmware update has incremented, the update was successful.

The following is an example of log output during XMODEM transfer when updating the firmware on the secondary MCU (RL78G23-128p FPB).

Primary MCU side (1):

```
==== RX65N : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute image ..
==== RX65N : FWUPCOMM DEMO [Primary][with buffer]  ver 1.0.0 ====
Please select the target MCU to update firmware.
  0: Primary MCU
  1: Secondary MCU

> 1
Please send the firmware for secondary MCU
```

Primary MCU side (2) (Additional connection for XMODEM):

```
[S]Received 128 bytes. total 128 bytes.
Send FWUP_START command... OK.
[S]Received 128 bytes. total 256 bytes.
[S]Received 128 bytes. total 384 bytes.
[S]Received 128 bytes. total 512 bytes.
Send FWUP_WRITE command... OK. (512 bytes sent, remaining 4294966784 bytes.)
[S]Received 128 bytes. total 640 bytes.
[S]Received 128 bytes. total 768 bytes.
[S]Received 128 bytes. total 896 bytes.
[S]Received 128 bytes. total 1024 bytes.
Send FWUP_WRITE command... OK. (512 bytes sent, remaining 8448 bytes.)
...
[S]Received 128 bytes. total 9344 bytes.
[S]Received 128 bytes. total 9472 bytes.
Send FWUP_WRITE command... OK. All data sent. Verification succeeded.
Send FWUP_INSTALL command... OK.
Firmware update for the device(0xA0) is successful.
```

Secondry MCU side:

```
Received FWUPCOMM_CMD_FWUP_START command.
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x1000, 512 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x1200, 256 ... OK
W 0x1300, 256 ... OK
...
Received FWUPCOMM_CMD_FWUP_WRITE command. size=512
W 0x4C00, 512 ... OK
Received FWUPCOMM_CMD_FWUP_WRITE command. size=256
W 0x4E00, 256 ... OK
verify install area buffer [sig-sha256-ecdsa]...OK
Received FWUPCOMM_CMD_FWUP_INSTALL command.
software reset...

==== RL78G23 : BootLoader [with buffer] ====
verify install area buffer [sig-sha256-ecdsa]...OK
copy to main area ... OK
software reset...
==== RL78G23 : BootLoader [with buffer] ====
verify install area main [sig-sha256-ecdsa]...OK
execute new image ...
==== RL78/G23 : FWUPCOMM DEMO [Secondary][with buffer]  ver. 2.0.0 ====
```

## 5.5 Settings for the Demo Project When Using SPI Communication Between MCUs

This demo project uses UART communication between MCUs by default.

To use SPI communication, change the r_fwupcomm setting in src/src/r_config/r_fwupcomm_config.h as follows.

Primary MCU (CK-RX65Nv2) side:

**Table 5-2 Changes to r_fwupcomm settings on the primary MCU side**

| Property | Macro definition | Vale |
|---|---|---|
| Communication Interface | FWUPCOMM_CFG_CH_INTERFACE | SCI SPI (Primary MCU Only) |

Secondary MCU(RL78 MCU) side:

**Table 5-3 Changes to r_fwupcomm settings on the secondary MCU side**

| Property | Macro definition | Vale |
|---|---|---|
| Communication Interface | FWUPCOMM_CFG_CH_INTERFACE | 20 (RL78/G23)<br>21 (RL78/G22) |

# 6. Appendices

## 6.1 Environments for Confirming Operation

This section describes environments in which the operation of this module has been confirmed.

**Table 6-1 Environment for Confirming Operation (CC-RL)**

| Item | Description |
|---|---|
| Integrated development environment | e$^2$ studio 2025-10 from Renesas Electronics |
| C compiler | C/C++ Compiler for RL Family V1.15.00 from Renesas Electronics<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| Endian | Little endian |
| Revision of the module | Rev. 1.10 |
| Board used | RL78/G23-128p Fast Prototyping Board (product No.: RTK7RLG230CSN000BJ)<br>RL78/G22 Fast Prototyping Board (product No.: RTK7RLG220C00000BJ) |
| USB-to-serial conversion board | Pmod USBUART (from DIGILENT)<br>https://digilent.com/reference/pmod/pmodusbuart/start |

**Table 6-2 Environment for Confirming Operation (CC-RX)**

| Item | Description |
|---|---|
| Integrated development environment | e$^2$ studio 2025-10 from Renesas Electronics |
| C compiler | C/C++ Compiler for RX Family V3.07.00 from Renesas Electronics<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| Endian | Little endian |
| Revision of the module | Rev. 1.10 |
| Board used | Cloud Kit for RX65N Microcontroller Group (product No.: RTK5CK65N0S08001BE) |
| USB-to-serial conversion board | Pmod USBUART (from DIGILENT)<br>https://digilent.com/reference/pmod/pmodusbuart/start |

**Table 6-3 Environment for Confirming Operation (GCC)**

| Item | Description |
|---|---|
| Integrated development environment | e$^2$ studio 2025-10 from Renesas Electronics |
| C compiler | GCC for Renesas RX 14.2.0.202505<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| Endian | Little endian |
| Revision of the module | Rev. 1.10 |
| Board used | Cloud Kit for RX65N Microcontroller Group (product No.: RTK5CK65N0S08001BE) |
| USB-to-serial conversion board | Pmod USBUART (from DIGILENT)<br>https://digilent.com/reference/pmod/pmodusbuart/start |

## 6.2 Settings for UART Communications

Table 6-4 lists the settings for UART communications by this module.

**Table 6-4 Settings for UART Communications**

| Item | Description |
|---|---|
| Data length | 8 bits |
| Parity | None |
| Stop bit | 1 bit |
| Flow control | None |
| Bit rate | 1 Mbps |

## 6.3 Operating Environment for the Demonstration Projects

This section shows the configurations of connections of each device for the demonstration projects.

For the PMOD pins of the evaluation board and the USB-to-serial conversion board in the figure, pins 1 to 6 of the PMOD interface are connected to pins 1 to 6 of the USB-to-serial conversion board (Pmod USBUART).

### 6.3.1 Environment for Confirming Operation with an RL78/G23

The configuration of connections is shown below.

#### 6.3.1.1 Connection Configuration for UART Communications

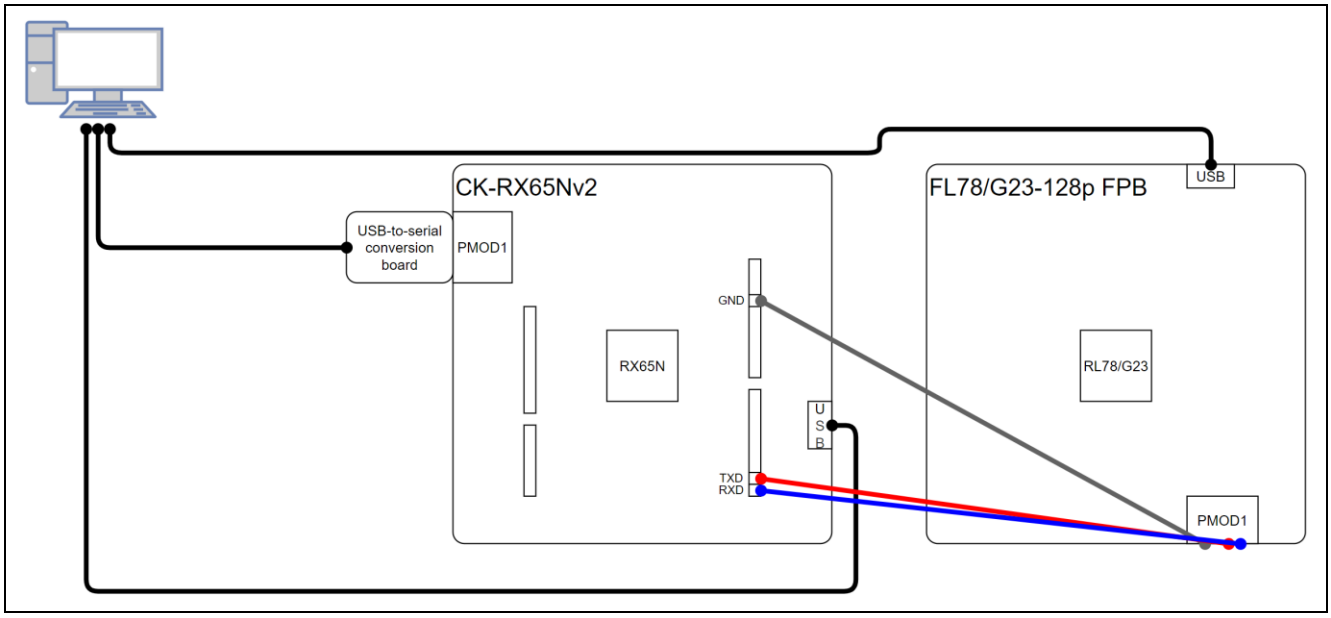


**Figure 6-1 Configuration of Connections on the RL78/G23-128p FPB(UART)**

**Table 6-5 Correspondence of Connected Pins for UART Communications between the CK-RX65Nv2 and RL78/G23-128p FPB**

| CK-RX65Nv2 | | RL78/G23-128p FPB |
|---|---|---|
| J24 Pin7: GND | ⇔ | PMOD1 Pin5 |
| J23 Pin2: D1/TX | ⇔ | PMOD1 Pin3 |
| J23 Pin1: D0/RX | ⇔ | PMOD1 Pin2 |

### 6.3.1.2 Connection Configuration for SPI Communication

For this demo project, pull up the MISO line.

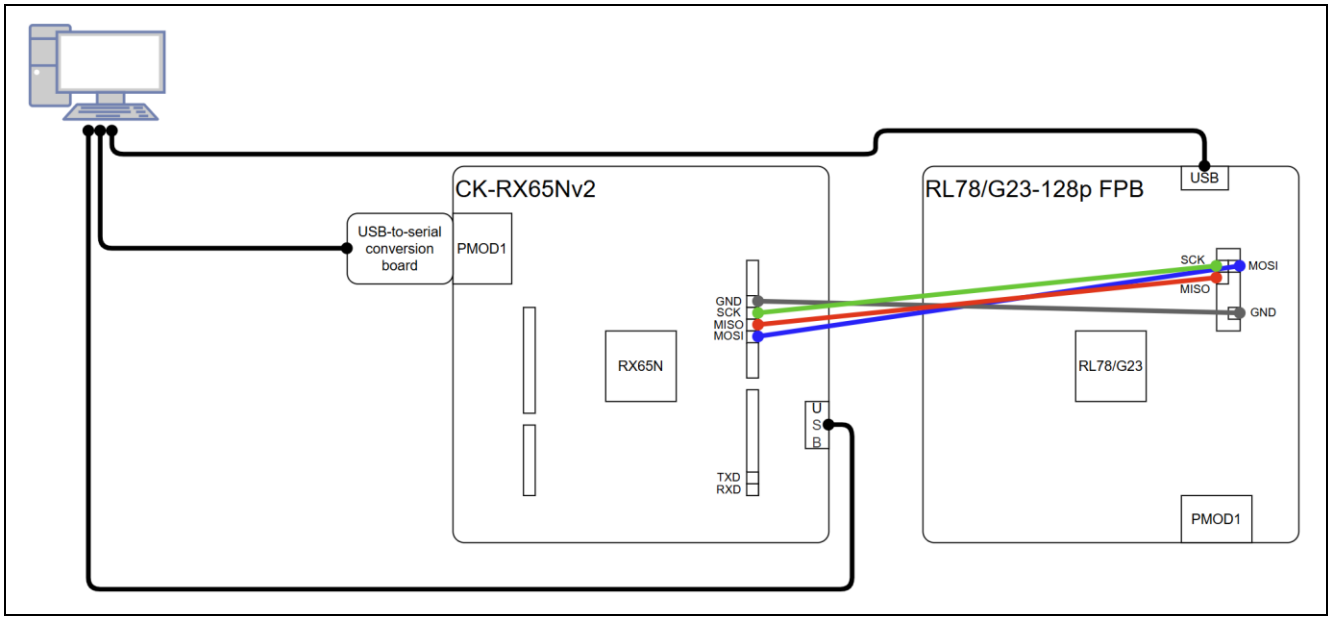


**Figure 6-2 Configuration of Connections on the RL78/G23-128p FPB(SPI)**

**Table 6-6 Correspondence of Connected Pins for SPI Communications between the CK-RX65Nv2 and RL78/G23-128p FPB**

| CK-RX65Nv2 | | RL78/G23-128p FPB |
|---|---|---|
| J24 Pin7: GND | ⇔ | J3 Pin12 |
| J24 Pin6: SPI_SCK | ⇔ | J3 Pin3 |
| J24 Pin5: SPI_MISO | ⇔ | J3 Pin5 |
| J24 Pin4: SPI_MOSI | ⇔ | J3 Pin4 |

### 6.3.2 Environment for Confirming Operation with an RL78/G22

The configuration of connections is shown below.

#### 6.3.2.1 Connection Configuration for UART Communications

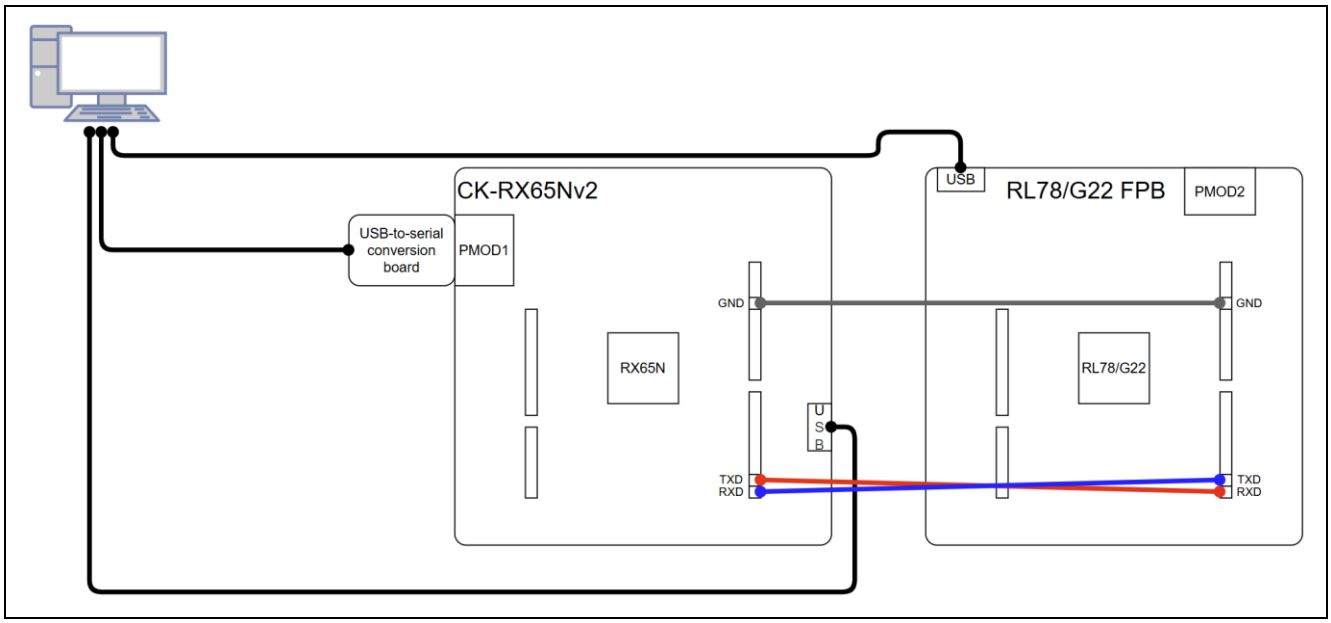


**Figure 6-3 Configuration of Connections on the RL78/G22 FPB(UART)**

**Table 6-7 Correspondence of Connected Pins for UART Communications between the CK-RX65Nv2 and RL78/G22 FPB**

| CK-RX65Nv2 | | RL78/G22 FPB |
|---|---|---|
| J24 Pin7: GND | ⇔ | J8 Pin7 |
| J23 Pin2: D1/TX | ⇔ | J7 Pin1 RX/0 |
| J23 Pin1: D0/RX | ⇔ | J7 Pin2 TX/1 |

### 6.3.2.2 Connection Configuration for SPI Communications

For this demo project, pull up the MISO line.

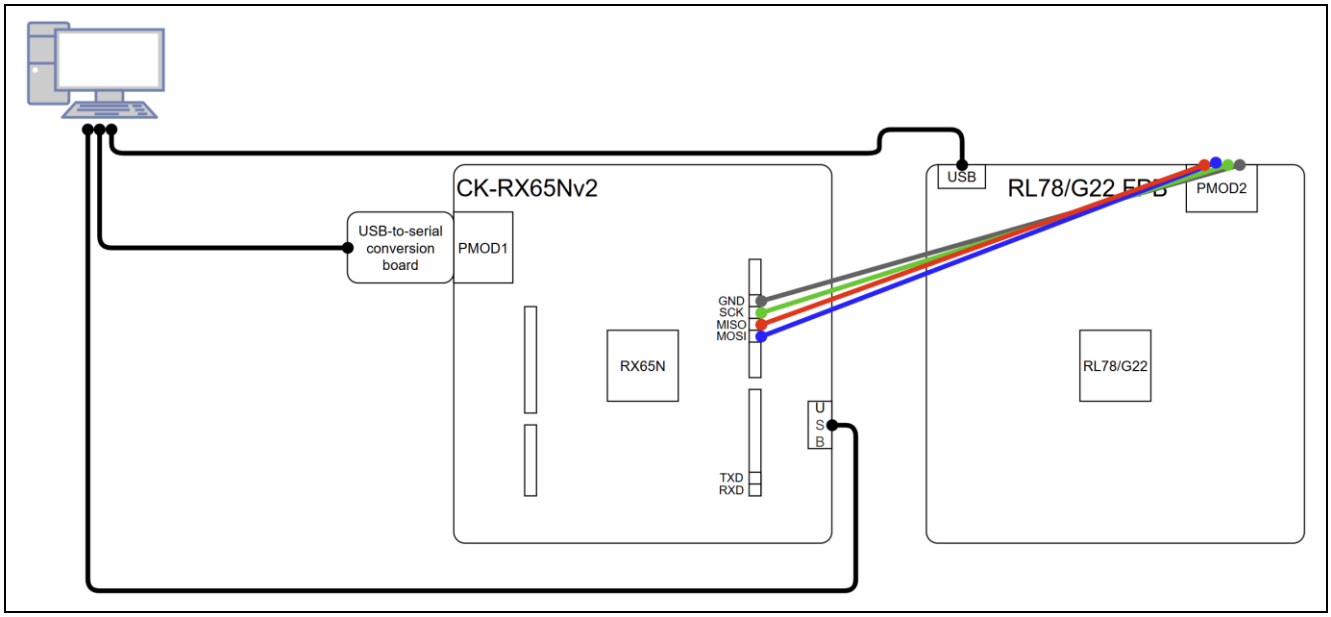


**Figure 6-4 Configuration of Connections on the RL78/G22 FPB(SPI)**

**Table 6-8 Correspondence of Connected Pins for SPI Communications between the CK-RX65Nv2 and RL78/G22 FPB**

| CK-RX65Nv2 | | RL78/G22 FPB |
|---|---|---|
| J24 Pin7: GND | ⇔ | PMOD2 Pin5 |
| J24 Pin6: SPI_SCK | ⇔ | PMOD2 Pin4 |
| J24 Pin5: SPI_MISO | ⇔ | PMOD2 Pin2 |
| J24 Pin4: SPI_MOSI | ⇔ | PMOD2 Pin3 |

### 6.3.3 Environment for Confirming Operation with an RX65N

The configuration of connections is shown below.

#### 6.3.3.1 Connection Configuration When the Communication Method Between the PC and Primary MCU is XMODEM

When the communication method between the PC and primary MCU is XMODEM, the connection between the CK-RX65Nv2 and the PC requires connecting the CK-RX65Nv2's USB Type-C port to the PC, in addition to the connection required for UART RAW mode.

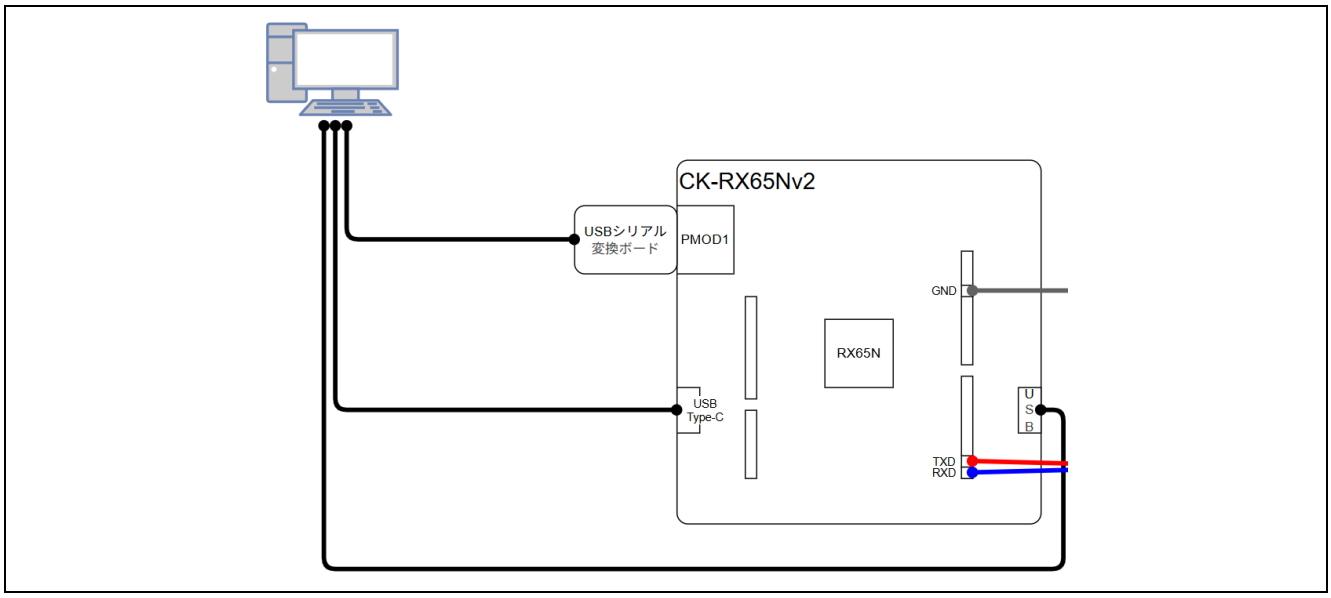


**Figure 6-5 Configuration of Connection on the CK-RX65Nv2 for XMODEM Communication**

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | May.20.25 | — | First edition issued |
| 1.10 | Dec.24.25 | — | • Module<br>— Added SPI communication functionality<br>— Added broadcast address<br>— Added FWUP_VERSION command<br>— Renamed communication interface function from rx_flush to rx_reset<br>• Demo project<br>— Added XMODEM to the firmware update data transfer method from PC<br>— Supported firmware updates for the primary MCU |

# 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 2020.10)

## 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/.