RL78 Family

Flash programmer with Raspberry Pi (RL78 Protocol A)

Introduction

This application note describes a sample program for a flash programmer that writes to the flash memory of a microcontroller that supports Protocol A.

Operation Confirmation Device

RL78/G13

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

Related Document

Documents related to this application note are listed below, refer to the following documents as well.

- RL78 Microcontrollers (RL78 Protocol A) Programmer Edition (R01AN0815)

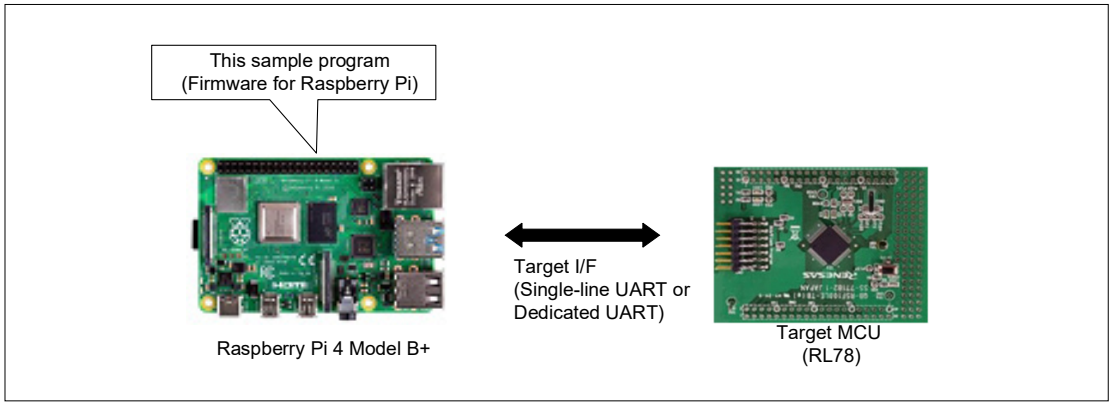
Contents

1. Overview

This sample program is a Raspberry Pi sample program for writing to the RL78 microcontroller's on-chip flash memory and has the following features.

- The writing target RL78 microcontroller (target MCU) supports the RL78 Protocol A.
- Serial programming of the RL78 Protocol A is used for writing.
- Raspberry Pi 4 Model B+ is used as the programmer hardware.
- Program files (written data) must conform to Motorola S format.

Figure 1-1 Image diagram



2. Development Environments

The operation of the sample program provided with this application note has been tested under the following conditions. There are several ways to connect the flash programmer (Raspberry Pi4 Model B+), including remote connection using a PC and standalone connection by directly connecting peripherals such as a monitor.

Table 2-1 Operation Confirmation Conditions

Development tools	Description
Flash programmer	Raspberry Pi4 Model B+ (On-chip RAM 4GB)
OS	Raspberry Pi OS 64-bit (version 5.10.17)
Language	C99
Software build environment	gcc：8.3.0 (Raspbian 8.3.0-6+rpi1)
Compiler	make：GNU Make 4.2.1
Shared library	ldd：2.28 (Debian GLIBC 2.28-10+rpi1)

Caution: It may not work with versions other than those listed above.

3. Raspberry Pi Settings

3.1 Raspberry Pi environment (config.txt) Setting

To be able to use UART2 to 5, add the following items under [all] in /boot/config.txt and reboot.

```
enable_uart=1
dtoverlay=uart1
dtoverlay=uart2
dtoverlay=uart3
dtoverlay=uart4
dtoverlay=uart5
```

3.2 Build Environment

If you want to update the build environment, execute the following command.

```
$sudo apt-get update
$sudo apt-get upgrade
```

You can check the versions of gcc and make with the following command.

```
$gcc -v
```

3.3 Way to Build

To build, execute the following command in the directory where the makefile is located.

Build:

```
$sudo male ALL
```

If you want to delete executable binary:

```
$sudo male clean
```

4. Specification

This sample program executes the executable file "fp_a" on the flash programmer (Raspberry Pi4 Model B+) and writes the Motorola S format file (write data) in the flash programmer to the target MCU.

4.1 Option Specifications

Perform initial settings and communicate with the target according to the specifications below.

- If execution is successful with the specified option settings, the flash programmer sends "OK" to the terminal.
- If execution fails with the specified option settings, the flash programmer sends "ERROR:XX" to the terminal. XX is displayed as a 2-digit hexadecimal number. See Table 4-2 for more information.

Table 4-1 shows details of the options, and Figure 4-1and Figure 4-2 show examples of how to use the options.

Table 4-1 Option Specifications

Long option	Short option	Setting	Description
--file=	-f	“file name”.mot	Specify the S-Record file.
--if=	-u	uart1	uart1: Performs communication with the target MCU using a single-line UART (TOOL0). uart2: Performs communication with the target MCU using a dedicated UART (TOOL0, TOOLTxD, TOOLRxD). When omitted: It is the same result as when the uart1 is specified.
		uart2	
--speed=	-b	115200	Specifies the transmission rate (bps) set by the Baud Rate Set command of the RL78 Protocol A. When omitted: It is the same result as when the 115200 is specified.
		250000	
		500000	
		1000000	
--vdd=	-d	x.x (1-digit decimal integer, first decimal place)	Specifies the VDD voltage (V) set by the Baud Rate Set command of the RL78 Protocol A. Set the VDD voltage supplied to the programmer board and the target MCU. When omitted: It is the same result as when the 3.3 is specified.
--verify	-v	-	If this option is specified, verification is performed additionally.
--checksum	-s	-	If this option is specified, a checksum value is acquired additionally.

Figure 4-1 Example of using long options (executable file name: fp_a)

```
>  sudo ./fp_a --file=test.mot --if=uart1 --vdd=3.3 --verify --checksum
OK:connect
OK:erase
OK:program,verify
OK:checksum
 code flash:xxxx
 data flash:xxxx
```

Figure 4-2 Example of using short options (executable file name: fp_a)

```
>  sudo ./fp_a -ftest.mot -uuart1 -d3.3 -v -s
OK:connect
OK:erase
OK:program,verify
OK:checksum
 code flash:xxxx
 data flash:xxxx
```

4.2 Error Code Specifications

When the execution of the executable file fails, an error message in the format "Error:XX" is displayed on the terminal. If XX is a 2-digit hexadecimal number, see error code.
Table 4-2 show the error codes.

Table 4-2 List of Error Codes

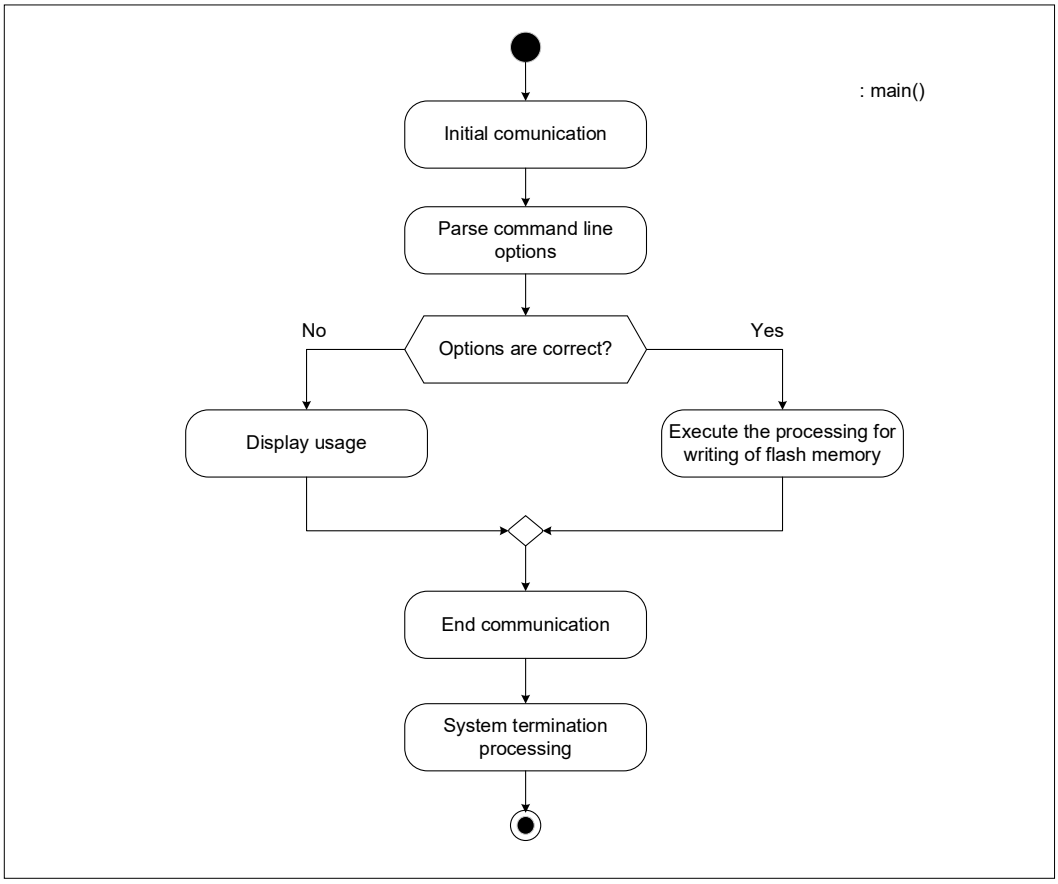
Error Code (Hexadecimal)	Description
04	Command number error This error occurs when a command number error of the RL78 Protocol A status code is received from the target MCU.
05	Parameter error This error occurs when a parameter error of the RL78 Protocol A status code is received from the target MCU.
07	Checksum error This error occurs when a checksum error of the RL78 Protocol A status code is received from the target MCU.
0F	Verification error This error occurs when a verification error of the RL78 Protocol A status code is received from the target MCU.
10	Protection error This error occurs when a protection error of the RL78 Protocol A status code is received from the target MCU.
15	NACK This error occurs when a NACK of the RL78 Protocol A status code is received from the target MCU.
1A	Erasure error This error occurs when an erasure error of the RL78 Protocol A status code is received from the target MCU.
1B	Blank error This error occurs when a blank error of the RL78 Protocol A status code is received from the target MCU.
1C	Write error This error occurs when a write error of the RL78 Protocol A status code is received from the target MCU.
FB	Invalid Motorola S-format data This error occurs if Motorola S-format data sent to the target MCU is invalid. This error occurs even when Motorola S-format data is not in ascending order of address.
FC	Target MCU communication timeout This error occurs if a timeout occurs during communication between the programmer board and the target MCU.
FE	Command communication data error This error occurs if an invalid packet format is received from the target MCU.
FF	System error This error occurs if the program does not work correctly.

4.3 Flowchart

4.3.1 Main Routine (main function)

Figure 4-3 shows the operation of the main routine.

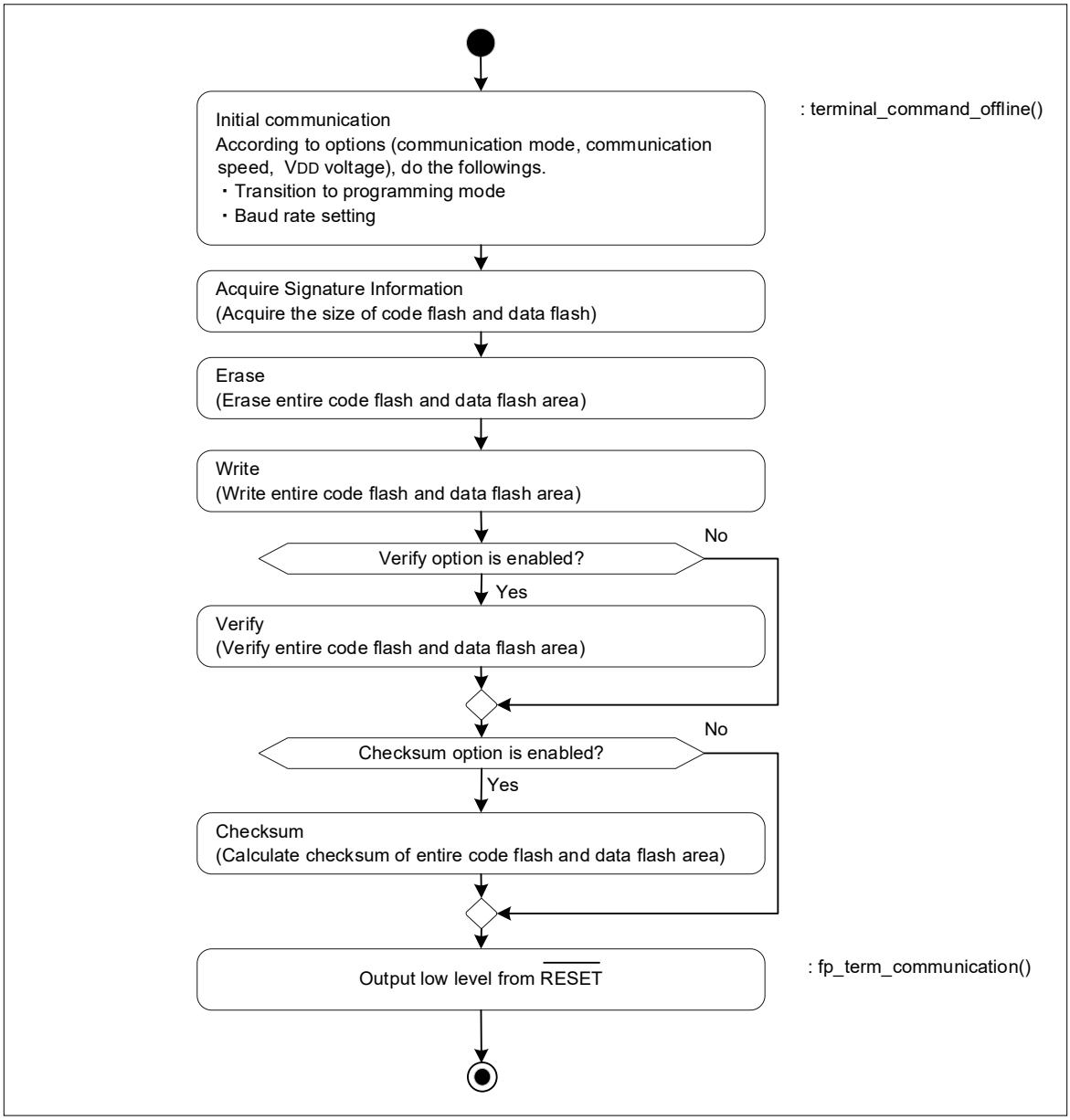
Figure 4-3 Main routine



4.3.2 Memory Write Processing

Figure 4-4 shows the operation of the memory write processing.

Figure 4-4 Memory write processing

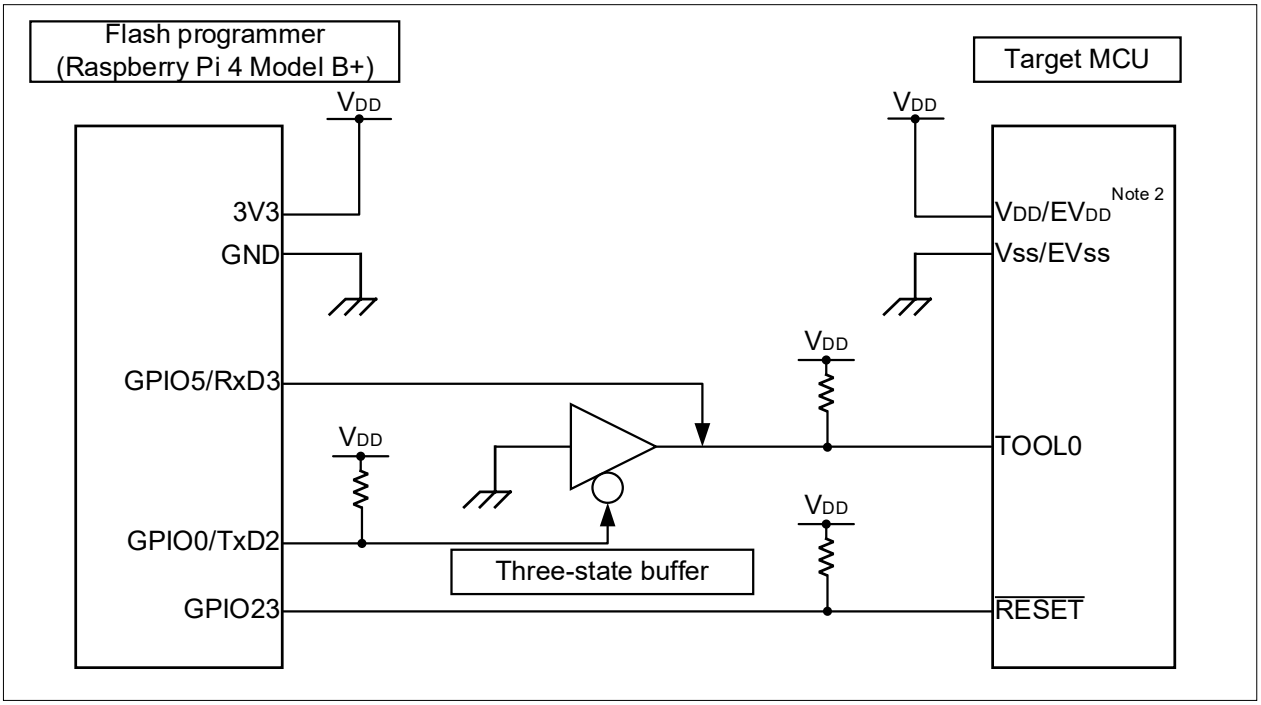


5. Hardware Descriptions

5.1 Target Interface Specifications

The following figures show how to connect the flash programmer to the target MCU.

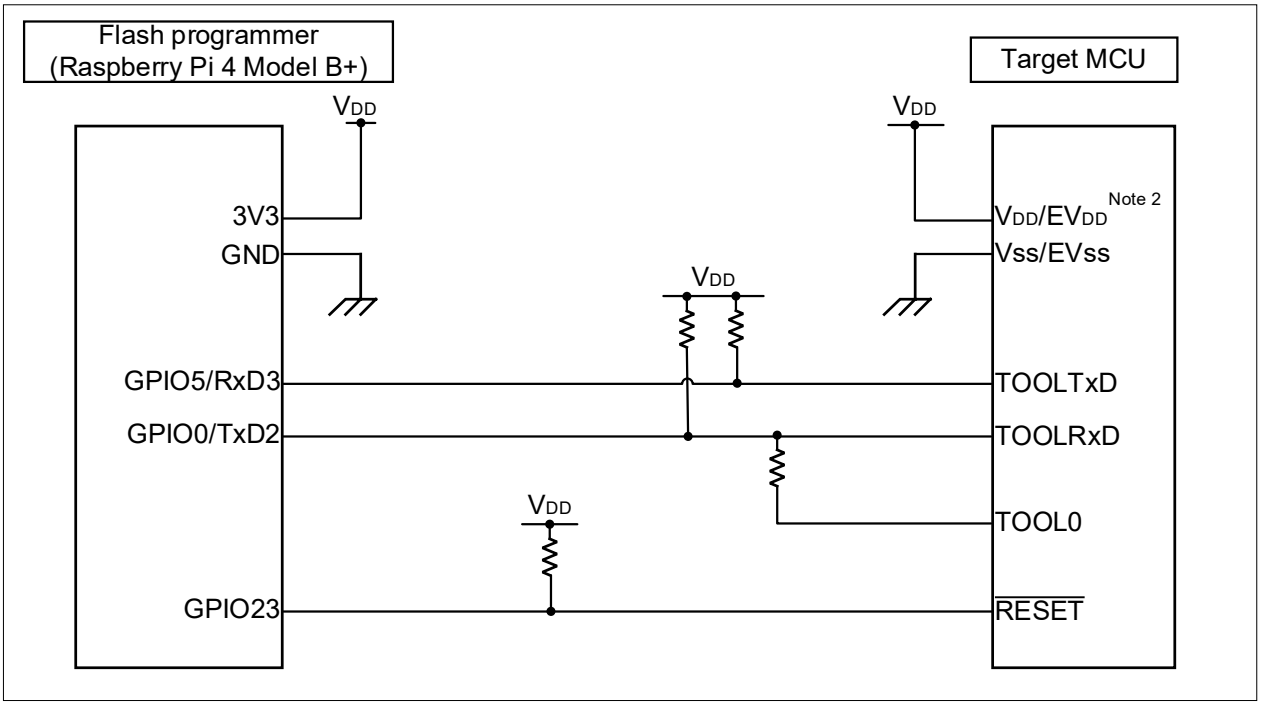
Figure 5-1 Single Line UART (V_{DD}=EV_{DD}) Note 1



Note 1. This is a connection example when V_{DD} is 3.3V.
Note 2. If V_{DD} and EV_{DD} are different, EV_{DD} must be powered externally.

5.1.1 Dedicated UART

Figure 5-2 Dedicated UART (V_{DD}=EV_{DD}) Note 1



Note 1. This is a connection example when V_{DD} is 3.3V.
Note 2. If V_{DD} and EV_{DD} are different, EV_{DD} must be powered externally.

5.2 List of Pins Used

Table 5-1 shows the flash programmer pins and functions used in the sample program.

Table 5-1 List of Error Codes

Pin Name	I/O	Function
TxD2	Output	Target interface communication transmit pin (UART2) Note 1
RxD3	Input	Target interface communication receive pin (UART3) Note 1
GPIO23	Output	Target MCU $\overline{\text{RESET}}$ control pin

Caution In this application note, only the used pins are handled. When creating your circuit, apply appropriate handling to the pins and design the circuit to meet the electrical characteristics.

Note 1. In the RL78 protocol A communication specification, the stop bits for transmission and reception are different, but in the UART communication settings of the Raspberry Pi OS, the stop bits for transmission and reception are the same, so the UARTs used for transmission and reception are separated.

6. Software Descriptions

6.1 List of files

The followings are list of files used in the sample program.
Table 6-1 shows the files provided by the Raspberry Pi OS, and Table 6-2 shows the files provided by the sample program.

Table 6-1 List of files provided by Raspberry Pi OS

Directory	File name	Description
/dev/	mem	Memory mapped I/O file
/dev/	ttyAMA1 or ttyAMA2 Note 1	Serial communication port file For UART2
/dev/	ttyAMA2 or ttyAMA3 Note 1	Serial communication port file For UART3
/boot/	config.txt firmware/config.txt	config file for RPi4

Note 1. It depends on the OS version.

Table 6-2 List of files provided by sample program

Directory	File name	Description
./	fp_a	Executable file of the program created with make (This file was created in the environment described in 2 Development Environments.) If you copy and use it from another location, you may need to grant execute permission using the following command: $ chmod a+x <file name>"
./	makefile	Sample makefile (Text file describing the procedure to be executed with the make command)
./	main.c	main function processing
common/	protocol_a.c protocol_a.h	Protocol A command processing
common/	terminal_com.c terminal_com.h	Terminal command processing
common/	utility.h	Utility functions processing
driver/	config_driver.c config_driver.h	System initialization function processing
driver/	config_gpio.c config_gpio.h	Device driver for GPIO
driver/	config_systemtimer.c config_systemtimer.h	Device driver for System Timer
driver/	config_uart.c config_usrt.h	Device driver for UART

6.2 List of Functions

Table 6-3 and Table 6-4 show major functions used in the sample program.

Table 6-3 List of functions (1/2)

Function name	Description	Source file
main	main function	main.c
read_arguments	Parsing arguments of option	main.c
system_init	System initialization processing	config_driver.c
system_term	System termination processing	config_driver.c
config_gpio_create	Acquiring memory mapped I/O of GPIO register	config_gpio.c
config_gpio_destroy	Discarding memory mapped I/O of GPIO register	config_gpio.c
config_gpio_p23_output_start	Setting GPIO23 to output	config_gpio.c
config_gpio_p23_output_stop	Returning GPIO23 to initial setting	config_gpio.c
config_gpio_p0_txd2_start	Setting GPIO0 to TXD2	config_gpio.c
config_gpio_p0_txd2_stop	Setting GPIO0 to output	config_gpio.c
config_gpio_control_reset	HI/LO control of GPIO23($\overline{\text{RESET}}$)	config_gpio.c
config_gpio_control_tool0	HI/LO control of GPIO0(TOOL0)	config_gpio.c
config_systemtimer_create	Acquiring memory mapped I/O of SystemTimer register	config_systemtimer.c
config_systemtimer_destroy	Discarding memory mapped I/O of SystemTimer register	config_systemtimer.c
config_systemtimer_get_count	Acquiring count value of SystemTimer	config_systemtimer.c
config_systemtimer_wait_ms	Waiting in ms unit	config_systemtimer.c
config_systemtimer_wait_us	Waiting in us unit	config_systemtimer.c
config_uart_create	Acquiring memory mapped I/O of UART register	config_uart.c
config_uart_destroy	Discarding memory mapped I/O of UART register	config_uart.c
config_uart2_start	Initial setting for UART2	config_uart.c
config_uart2_stop	Discarding UART2 setting	config_uart.c
config_uart3_start	Initial setting for UART3	config_uart.c
config_uart3_stop	Discarding UART3 setting	config_uart.c
config_uart2_send	Sending data from TXD2	config_uart.c
config_uart2_send_with_wait	Sending data from TXD2 (with waiting time between data transmission)	config_uart.c
config_uart3_receive	Receiving data from RXD3	config_uart.c
config_uart23_set_baudrate	Baud rate setting of UART2, UART3	config_uart.c
config_uart2_set_send_wait_time	Setting wait time for config_uart2_send_with_wait	config_uart.c
fp_cmd_reset_a	Reset command sending processing	protocol_a.c
fp_cmd_verify_a	Verify command sending processing	protocol_a.c
fp_cmd_erase_a	Block Erase command sending processing	protocol_a.c
fp_cmd_program_a	Programming command sending processing	protocol_a.c
fp_cmd_baudrate_a	Baud Rate Set command sending processing	protocol_a.c

Table 6-4 List of functions (2/2)

Function name	Description	Source file
fp_cmd_checksum_a	Checksum command sending processing	protocol_a.c
fp_cmd_signature_a	Silicon Signature command sending processing	protocol_a.c
fp_initial_communication	Initial processing for starting communication (Entry to Flash Memory Programming Modes)	protocol_a.c
fp_get_signature	Executing Silicon Signature command and acquiring each parameter	protocol_a.c
terminal_command_init	Initialization of each parameter	terminal_com.c
terminal_command_init_dev	Initialization of device-dependent parameters	terminal_com.c
terminal_command_offline	Executing Flash rewriting processing	terminal_com.c

6.3 Specification of Functions

This section shows the specifications of major functions used in the sample program.

read_arguments	
Outline	Parsing arguments of option
Declaration	static uint8_t read_arguments(st_command_data_t * cmd, int argc, char * argv[])
Argument	st_command_data_t * cmd: Option setting information int argc: Number of optional arguments char * argv[]: Optional arguments
Return Value	0: Normal end 1: Abnormal end (Option is not correct.)
Description	Read and parse the argument “com_data”.

system_init	
Outline	System initialization processing
Declaration	void system_init (void)
Argument	-
Return Value	-
Description	Initialize system settings.

system_term	
Outline	System termination processing
Declaration	void system_term (void)
Argument	-
Return Value	-
Description	Terminate the system.

config_gpio_create	
Outline	Acquiring memory mapped I/O of GPIO register
Declaration	void config_gpio_create (int32_t mem_fd)
Argument	int32_t mem_fd: MMIO file descriptor
Return Value	-
Description	Acquire the memory mapped I/O of the GPIO register and set the GPIO ports.

config_gpio_destroy	
Outline	Discarding memory mapped I/O of GPIO register
Declaration	void config_gpio_destroy (void)
Argument	-
Return Value	-
Description	Discard the memory mapped I/O of the GPIO register to return the GPIO port to its state before program execution.

config_gpio_p23_output_start	
Outline	Setting GPIO23 to output pin
Declaration	void config_gpio_p23_output_start (void)
Argument	-
Return Value	-
Description	Set the GPIO23 of Raspberry Pi to output.

config_gpio_p23_output_stop	
Outline	Returning GPIO23 to initial setting
Declaration	void config_gpio_p23_output_stop (void)
Argument	-
Return Value	-
Description	Return the GPIO23 of Raspberry Pi to its initial state.

config_gpio_p0_txd2_start	
Outline	Setting GPIO0 to TXD2
Declaration	void config_gpio_p0_txd2_start(void)
Argument	-
Return Value	-
Description	Set the GPIO0 of Raspberry Pi to the TxD2.

config_gpio_p0_txd2_stop	
Outline	Setting GPIO0 to output
Declaration	void config_gpio_p0_txd2_stop(void)
Argument	-
Return Value	-
Description	Set the GPIO0 of Raspberry Pi to output.

config_gpio_control_reset	
Outline	HI/LO control of GPIO23($\overline{\text{RESET}}$)
Declaration	void config_gpio_control_reset(uint8_t enabled)
Argument	uint8_t enabled: Reset information (0: Release of a reset 1: Reset)
Return Value	-
Description	Control HI/LO of the GPIO23($\overline{\text{RESET}}$) of Raspberry Pi. The reset is released by executing “config_gpio_control_reset(0)” and $\overline{\text{RESET}}$ signal is changed to HI. The target MCU is in the reset state by execution “config_gpio_control_reset(1)” and $\overline{\text{RESET}}$ signal is changed to LO.

config_gpio_control_tool0	
Outline	HI/LO control of GPIO0(TOOL0)
Declaration	void config_gpio_control_tool0(uint8_t enabled)
Argument	uint8_t enabled: HI/LO information of GPIO0 (0: LO 1: HI)
Return Value	-
Description	Control HI/LO of GPIO0(TOOL0) of Raspberry Pi.

config_systemtimer_create	
Outline	Acquiring memory mapped I/O of SystemTimer register
Declaration	void config_systemtimer_create(int32_t mem_fd)
Argument	int32_t mem_fd: MMIO file descriptor
Return Value	-
Description	Acquire the memory mapped I/O of the SystemTimer register.

config_systemtimer_destroy	
Outline	Discarding memory mapped I/O of SystemTimer register
Declaration	void config_systemtimer_destroy(void)
Argument	-
Return Value	-
Description	Discard the memory mapped I/O of the SystemTimer register.

config_systemtimer_get_count	
Outline	Acquiring count value of SystemTimer
Declaration	uint64_t config_systemtimer_get_count(void)
Argument	-
Return Value	The count value of SystemTimer
Description	Acquire the count value of the SystemTimer.

config_systemtimer_wait_ms	
Outline	Waiting in ms unit
Declaration	void config_systemtimer_wait_ms(const uint16_t wait_count)
Argument	const uint16_t wait_count: Wait time [ms]
Return Value	-
Description	Wait for the specified time (ms unit).

config_systemtimer_wait_us	
Outline	Waiting in us unit
Declaration	void config_systemtimer_wait_us(const uint16_t wait_count)
Argument	const uint16_t wait_count: Wait time [us]
Return Value	-
Description	Wait for the specified time (us unit).

config_uart_create	
Outline	Acquiring memory mapped I/O of UART register
Declaration	void config_uart_create(int32_t mem_fd)
Argument	int32_t mem_fd: MMIO file descriptor
Return Value	-
Description	Acquire the memory mapped I/O of the UART register.

config_uart_destroy	
Outline	Discarding memory mapped I/O of UART register
Declaration	void config_uart_destroy(void)
Argument	-
Return Value	-
Description	Discard the memory mapped I/O of the UART register.

config_uart2_start	
Outline	Initial setting for UART2
Declaration	void config_uart2_start(void)
Argument	-
Return Value	-
Description	Initialize the UART2. (In the Raspberry Pi4, the stop bit set by this function is common to both transmission and reception, so UART2 is used for transmission in this sample.)

config_uart2_stop	
Outline	Discarding UART2 setting
Declaration	void config_uart2_stop(void)
Argument	-
Return Value	-
Description	Discard the UART2 settings.

config_uart3_start	
Outline	Initial setting for UART3
Declaration	void config_uart3_start(void)
Argument	-
Return Value	-
Description	Initialize the UART3. (In the Raspberry Pi4, the stop bit set by this function is common to both transmission and reception, so UART3 is used for reception in this sample.)

config_uart3_stop	
Outline	Discarding UART3 setting
Declaration	void config_uart3_stop(void)
Argument	-
Return Value	-
Description	Discard the UAR3 settings.

config_uart2_send	
Outline	Sending data from TXD2
Declaration	e_md_status_t config_uart2_send(uint8_t * const tx_buf, uint16_t tx_num)
Argument	uint8_t * const tx_buf: Data for transmission uint16_t tx_num; Size of data for transmission
Return Value	MD_OK: Normal end MD_ARGERROR: Argument error MD_TXERROR: Transmission error
Description	Send data from the TXD2. If the data is too long, write as much as possible and then write the remaining data again. Repeat this until all data is sent. If there is a transmission error, the write() function returns a negative value, so this is used to detect errors.

config_uart2_send_with_wait	
Outline	Sending data from TXD2 (with waiting time between data transmission)
Declaration	e_md_status_t config_uart2_send_with_wait(uint8_t * const tx_buf, uint16_t tx_num)
Argument	uint8_t * const tx_buf: Data for transmission uint16_t tx_num; Size of data for transmission
Return Value	MD_OK: Normal end MD_ARGERROR: Argument error MD_TXERROR: Transmission error
Description	Send data from the TXD2. Add a wait for each 1 byte transmission and repeat until all data is sent.

config_uart3_receive	
Outline	Receiving data from RXD3
Declaration	e_md_status_t config_uart3_receive(uint8_t * const rx_buf, uint16_t rx_num, uint16_t timeout_ms, uint8_t is_echobacked, uint16_t * p_top_pos)
Argument	uint8_t * const rx_buf: Data for reception uint16_t rx_num: Size of data for reception uint16_t timeout_ms: Timeout time uint8_t is_echobacked: Eliminating single-line UART echoback uint16_t * p_top_pos: First data of received data
Return Value	MD_OK: Normal end MD_ARGERROR: Argument error MD_RXERROR: Reception error MD_RXTIMEOUT: Receive timeout error
Description	Receive data from the RXD3. If the received data is insufficient (for example, not all data has been received), read as much as possible and read it again. Repeat this until all data is read. If it cannot be received to the end, select() will fail at some stage and a timeout will occur.

config_uart23_set_baudrate	
Outline	Baud rate setting of UART2, UART3
Declaration	void config_uart23_set_baudrate(e_uart_baudrate_t baudrate)
Argument	e_uart_baudrate_t baudrate: Baud rate
Return Value	-
Description	Set the baud rate for UART2 and UART3.

config_uart2_set_send_wait_time	
Outline	Set wait time for config_uart2_send_with_wait
Declaration	void config_uart2_set_send_wait_time(uint8_t freq)
Argument	uint8_t freq : Operating frequency of Target MCU
Return Value	-
Description	Set the waiting time after sending.

fp_cmd_reset_c	
Outline	Reset command sending processing
Declaration	uint8_t fp_cmd_reset_c(void)
Argument	-
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Reset command of RL78 protocol A.

fp_cmd_verify_c	
Outline	Verify command sending processing
Declaration	uint8_t fp_cmd_verify_c(const uint32_t start, const uint32_t end, const uint8_t * data)
Argument	const uint32_t start: Start address for verify const uint32_t end: End address for verify const uint8_t * data: Comparison data for verify
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Verify command of RL78 protocol A.

fp_cmd_erase_c	
Outline	Block Erase command sending processing
Declaration	uint8_t fp_cmd_erase_c(const uint32_t addr)
Argument	const uint32_t addr: Start address of block for erase
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Block Erase command of RL78 protocol A.

fp_cmd_program_c	
Outline	Programming command sending processing
Declaration	uint8_t fp_cmd_program_c(const uint32_t start, const uint32_t end, const uint8_t * data)
Argument	const uint32_t start: Start address for rewriting const uint32_t end: End address for rewriting const uint8_t * data: Data for rewriting
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Programming command of RL78 protocol A.

fp_cmd_baudrate_c	
Outline	Baud Rate Set command sending processing
Declaration	uint8_t fp_cmd_baudrate_c(const e_uart_speed_t baudrate, const uint16_t vdd, uint8_t * frq, uint8_t * fpm)
Argument	const UART_SPEED baudrate: Communication baud rate UART_SPEED_DEFAULT: 115200 bps UART_SPEED_250000: 250000 bps UART_SPEED_500000: 500000 bps UART_SPEED_1000000: 1000000 bps const uint16_t vdd: V_{DD} voltage to be applied (in 100-mV units, truncate the digits after the decimal point.) uint8_t * frq: CPU operating frequency [MHz] (Acquire it from target MCU) uint8_t * fpm: Flash memory rewriting mode (Acquire it from target MCU)
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Baud Rate Set command of RL78 protocol A.

fp_cmd_checksum_c	
Outline	Checksum command sending processing
Declaration	uint8_t fp_cmd_checksum_c(const uint32_t start, const uint32_t end, uint16_t * checksum)
Argument	const uint32_t start: Start address for checksum const uint32_t end: End address for checksum const uint16_t * checksum: Acquired checksum data
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Checksum command of RL78 protocol A.

fp_cmd_signature_c	
Outline	Silicon Signature command sending processing
Declaration	uint8_t fp_cmd_signature_c(uint8_t * dvc, uint8_t * dev,uint32_t * cfe, uint32_t * dfe, uint8_t * fwv)
Argument	uint8_t * dvc: Device function code (Acquire it from target MCU) uint8_t * dev: Device name (Acquire it from target MCU) uint32_t * cfe: Last address of code flash memory area (Acquire it from target MCU) uint32_t * dfe: Last address of data flash memory area (Acquire it from target MCU) uint8_t * fwv: Boot firmware version (Acquire it from target MCU)
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Execute the Silicon Signature command of RL78 protocol A.

fp_initial_communication	
Outline	Initial processing for starting communication (Entry to Flash Memory Programming Modes)
Declaration	uint8_t fp_initial_communication(st_command_data_t * command)
Argument	st_command_data_t * command: Setting information for command
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Perform initial communication with the target MCU according to the RL78 protocol A. Release a reset, send the mode information, and execute the Baud Rate Set command and the Reset command.

fp_get_signature	
Outline	Executing Silicon Signature command and acquiring each parameter
Declaration	uint8_t fp_get_signature(st_command_data_t * command)
Argument	st_command_data_t * command: Setting information for command
Return Value	0: Normal end Other than 0: Abnormal end (refer to 4.2 Error Code Specifications)
Description	Acquire the signature information of the target MCU and set it to the argument “com_data”.

terminal_command_init	
Outline	Initialization of each parameter
Declaration	void terminal_command_init(st_command_data_t * com_data)
Argument	st_command_data_t * com_data: Setting information for command
Return Value	-
Description	Initialize the argument “com_data”.

terminal_command_init_dev	
Outline	Initialization of device-dependent parameters
Declaration	void terminal_command_init_dev(st_command_data_t * com_data)
Argument	st_command_data_t * com_data: Setting information for command
Return Value	-
Description	Initialize the signature information of the argument “com_data”.

terminal_command_offline	
Outline	Executing Flash rewriting processing
Declaration	void terminal_command_init(st_command_data_t * com_data)
Argument	st_command_data_t * com_data: Setting information for command
Return Value	-
Description	Read the S-Record data from the file and rewrite the flash memory.

7. Reference Documents

RL78 Microcontrollers (RL78 Protocol A) Programmer Edition (R01AN0815)

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb 29, 2024	-	First edition

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/.