RX Family

GPIO Module Using Firmware Integration Technology

Introduction

This application note describes the General Purpose Input/Output (GPIO) module which uses Firmware Integration Technology (FIT). This module uses GPIO to implements a General Purpose Input/Output Driver. In this document, this module is referred to as the GPIO FIT module.

Target Devices

- RX110 Group
- RX111 Group
- RX113 Group
- RX130 Group
- RX230 Group
- RX231 Group
- RX23T Group
- RX24T Group
- RX24U Group
- RX64M Group
- RX651, RX65N Group
- RX66T Group
- RX71M Group
- RX72T Group

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

Target Compilers

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

For details of the confirmed operation contents of each compiler, refer to “6.1 Confirmed Operation Environment".

Contents

1. Overview

1.1 GPIO FIT Module

The GPIO FIT module can be used by being implemented in a project as an API. See section 2.12, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the GPIO FIT Module

This module provides an abstraction layer for reading, writing, and configuring General Purpose Input/Output (GPIO) pins on RX MCUs. By using this module's API functions the user can bypass the need to know of the available GPIO registers for each pin. For example, on the RX there are separate registers for controlling pin direction, reading the pin, writing to the pin, enabling internal pull-ups, configuring output mode, and assigning a pin for peripheral use.

1.3 API Overview

Table 1.1 lists the API functions included in this module.

Table 1.1 API Functions

Function	Description
R_GPIO_PortWrite()	Sets the output levels of all pins on a port.
R_GPIO_PortRead()	Reads the current levels of all pins on a port.
R_GPIO_PortDirectionSet()	Sets multiple pins on a port as inputs or outputs.
R_GPIO_PinWrite()	Sets the output level of a pin.
R_GPIO_PinRead()	Reads the current level of a pin.
R_GPIO_PinDirectionSet()	Sets the direction (input/output) of a pin.
R_GPIO_PinControl()	Changes various settings for a pin (e.g. internal pull-up, open drain).
R_GPIO_GetVersion()	Returns the current version of this module.

1.4 Limitations

Some MCU packages with reduced pin counts have a Port Switching feature that multiplexes some I/O ports to shared physical pins. This driver does not support the Port Switching feature. Port Switching may still be done outside of this driver by the user's own code.

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- GPIO

2.2 Software Requirements

This driver is dependent upon the following FIT module:

- Renesas Board Support Package (r_bsp) v5.20 or higher

2.3 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 6.1, Confirmed Operation Environment.

2.4 Interrupt Vector

None.

2.5 Header Files

All API calls and their supporting interface definitions are located in “r_gpio_rx_if.h”.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

The configuration option settings of this module are located in r_gpio_rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_gpio_rx_config.h	
GPIO_CFG_PARAM_CHECKING_ENABLE - Default value = “BSP_CFG_PARAM_CHECKING_ENABLE”	= 1: Include parameter checking in the build. = 0: Omit parameter checking from the build. = BSP_CFG_PARAM_CHECKING_ENABLE Note: Code size can be reduced by excluding parameter checking from the build.

2.8 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3, Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

ROM and RAM code sizes						
Device	Memory Used					
	Renesas Compiler		GCC		IARCompiler	
	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX110	ROM: 486 bytes code	ROM: 314 bytes code	ROM: 1320 bytes code	ROM: 960 bytes code	ROM: 758 bytes code	ROM: 536 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 6 bytes	RAM: 6 bytes
RX111, RX113	ROM: 459 bytes code	ROM: 314 bytes code	ROM: 1324 bytes code	ROM: 960 bytes code	ROM: 753 bytes code	ROM: 532 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 6 bytes	RAM: 6 bytes
RX130, RX230	ROM: 592 bytes code	ROM: 404 bytes code	ROM: 1608 bytes code	ROM: 1088 bytes code	ROM: 916 bytes code	ROM: 592 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 6 bytes	RAM: 6 bytes
RX231 RX64M, RX71M	ROM: 592 bytes code	ROM: 404 bytes code	ROM: 1588 bytes code	ROM: 1088 bytes code	ROM: 916 bytes code	ROM: 592 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 4 bytes	RAM: 4 bytes
RX23T, RX24T, RX24U	ROM: 576 bytes code	ROM: 404 bytes code	ROM: 1583 bytes code	ROM: 1088 bytes code	ROM: 900 bytes code	ROM: 592 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 6 bytes	RAM: 6 bytes
RX651, RX65N	ROM: 688 bytes code	ROM: 451 bytes code	ROM: 1888 bytes code	ROM: 1224 bytes code	ROM: 1072 bytes code	ROM: 652 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 4 bytes	RAM: 4 bytes
RX66T	ROM: 696 bytes code	ROM: 462 bytes code	ROM: 1892 bytes code	ROM: 1224 bytes code	ROM: 1080 bytes code	ROM: 652 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 4 bytes	RAM: 4 bytes
RX72T	ROM: 696 bytes code	ROM: 462 bytes code	ROM: 1892 bytes code	ROM: 1224 bytes code	ROM: 1076 bytes code	ROM: 652 bytes code
	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 0 bytes	RAM: 4 bytes	RAM: 4 bytes

2.9 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_gpio_rx_if.h as are the prototype declarations of API functions.

2.9.1 Ports

This enum defines the available ports for this MCU. This enum is specific to a MCU Group and package and is stored in the target folder for that MCU Group. For example, to find the available ports for an RX111 the user would refer to the file *src/targets/rx111/r_gpio_rx111.h*. Below is an example from the RX111. Other MCUs will have different ports enumerations.

```
#if   (BSP_PACKAGE_PINS == 64)
typedef enum
{
    GPIO_PORT_0 = 0x0000,
    GPIO_PORT_1 = 0x0100,
    GPIO_PORT_2 = 0x0200,
    GPIO_PORT_3 = 0x0300,
    GPIO_PORT_4 = 0x0400,
    GPIO_PORT_5 = 0x0500,
    GPIO_PORT_A = 0x0A00,
    GPIO_PORT_B = 0x0B00,
    GPIO_PORT_C = 0x0C00,
    GPIO_PORT_E = 0x0E00,
    GPIO_PORT_H = 0x1100,
    GPIO_PORT_J = 0x1200,
} gpio_port_t;
```

2.9.2 Pins

This enum defines the available GPIO pins for this MCU. This enum is specific to an MCU Group and package. The user can find this enum in the target folder for that MCU Group. For example, to find the available GPIO pins for an RX111 the user would refer to the file *src/targets/rx111/r_gpio_rx111.h*. Below is an example from the RX111. Notice that the GPIO pins available in this enum are controlled by the BSP_PACKAGE_PINS macro which is automatically obtained from the r_bsp.

```
#if   (BSP_PACKAGE_PINS == 64)
typedef enum
{
    GPIO_PORT_0_PIN_3 = 0x0003,
    GPIO_PORT_0_PIN_5 = 0x0005,
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_0 = 0x0300,
    GPIO_PORT_3_PIN_1 = 0x0301,
    GPIO_PORT_3_PIN_2 = 0x0302,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_0 = 0x0400,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_3 = 0x0403,
    GPIO_PORT_4_PIN_4 = 0x0404,
    GPIO_PORT_4_PIN_6 = 0x0406,
    GPIO_PORT_5_PIN_4 = 0x0504,
    GPIO_PORT_5_PIN_5 = 0x0505,
    GPIO_PORT_A_PIN_0 = 0x0A00,
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
```

```
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_1 = 0x0B01,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_B_PIN_5 = 0x0B05,
    GPIO_PORT_B_PIN_6 = 0x0B06,
    GPIO_PORT_B_PIN_7 = 0x0B07,
    GPIO_PORT_C_PIN_0 = 0x0C00,
    GPIO_PORT_C_PIN_1 = 0x0C01,
    GPIO_PORT_C_PIN_2 = 0x0C02,
    GPIO_PORT_C_PIN_3 = 0x0C03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_C_PIN_5 = 0x0C05,
    GPIO_PORT_C_PIN_6 = 0x0C06,
    GPIO_PORT_C_PIN_7 = 0x0C07,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_E_PIN_5 = 0x0E05,
    GPIO_PORT_E_PIN_6 = 0x0E06,
    GPIO_PORT_E_PIN_7 = 0x0E07,
    GPIO_PORT_H_PIN_7 = 0x1107,
    GPIO_PORT_J_PIN_6 = 0x1206,
    GPIO_PORT_J_PIN_7 = 0x1207,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 48)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_0 = 0x0400,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_6 = 0x0406,
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_1 = 0x0B01,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_B_PIN_5 = 0x0B05,
    GPIO_PORT_C_PIN_0 = 0x0C00,
    GPIO_PORT_C_PIN_1 = 0x0C01,
    GPIO_PORT_C_PIN_2 = 0x0C02,
    GPIO_PORT_C_PIN_3 = 0x0C03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_C_PIN_5 = 0x0C05,
    GPIO_PORT_C_PIN_6 = 0x0C06,
    GPIO_PORT_C_PIN_7 = 0x0C07,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
```

```
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_E_PIN_7 = 0x0E07,
    GPIO_PORT_H_PIN_7 = 0x1107,
    GPIO_PORT_J_PIN_6 = 0x1206,
    GPIO_PORT_J_PIN_7 = 0x1207,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 40)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_2 = 0x0302,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_6 = 0x0406,
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_J_PIN_6 = 0x1306,
    GPIO_PORT_J_PIN_7 = 0x1307,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 36)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_J_PIN_6 = 0x1306,
    GPIO_PORT_J_PIN_7 = 0x1307
```

```
} gpio_port_pin_t;
#endif
```

2.9.3 Port_Pin Masks

This enum is specific to the MCU Group and package. It defines port-pin masks for this MCU. These are bit-masks that may optionally be applied by the user when performing port-wide writes or reads to check for valid port-wide operations. For each bit location in a port that has an I/O pin available, these bit-masks will have the bit set to '1'. Bits for non-existent pins are set to '0'. In the interest of performance, the GPIO driver does not automatically check for invalid pin settings when the port-wide write function is called. It is up to the user's application to insure that only valid pins are written to. Use of these masks is not required; they are provided as a convenience. Below is an example from the RX111 MCU.

```
#if (BSP_PACKAGE_PINS == 64)
/* This enumerator has a bit mask for each available GPIO pin for the given port
on this MCU. */
typedef enum
{
GPIO_PORT0_PIN_MASK = 0x28,   /* Available pins: P03,P05                  */
GPIO_PORT1_PIN_MASK = 0xF0,   /* Available pins: P14, P15, P16,P17        */
GPIO_PORT2_PIN_MASK = 0xC0,   /* Available pins: P26,P27                  */
GPIO_PORT3_PIN_MASK = 0x27,   /* Available pins: P30 to P32, P35          */
GPIO_PORT4_PIN_MASK = 0x5F,   /* Available pins: P40 to P44, P46          */
GPIO_PORT5_PIN_MASK = 0x30,   /* Available pins: P54, P55                 */
GPIO_PORTA_PIN_MASK = 0x5B,   /* Available pins: PA0, PA1, PA3, PA4, PA6  */
GPIO_PORTB_PIN_MASK = 0xEB,   /* Available pins: PB0, PB1, PB3, PB5 to PB7*/
GPIO_PORTC_PIN_MASK = 0xFF,   /* Available pins: PC0 to PC7               */
GPIO_PORTE_PIN_MASK = 0xFF,   /* Available pins: PE0 to PE7               */
GPIO_PORTH_PIN_MASK = 0x80,   /* Available pins: PH7                      */
GPIO_PORTJ_PIN_MASK = 0xC0    /* Available pins: PJ6, PJ7                 */
} gpio_pin_bit_mask_t;
```

2.9.4 Pin Level

This enum defines the different options that can be returned when reading a GPIO pin.

```
/* Levels that can be set and read for individual pins. */
typedef enum
{
    GPIO_LEVEL_LOW = 0,
    GPIO_LEVEL_HIGH
} gpio_level_t;
```

2.9.5 Pin Direction

This enum defines the different options that can be used for configuring a GPIO pin's direction.

```
/* Options that can be used with the R_GPIO_PortDirectionSet() and
   R_GPIO_PinDirectionSet() functions. */
typedef enum
{
    GPIO_DIRECTION_INPUT = 0,
    GPIO_DIRECTION_OUTPUT
} gpio_dir_t;
```

2.9.6 Control Commands

This enum defines the different commands that can be sent to the R_GPIO_PinControl() function.

```
/* Commands that can be used with the R_GPIO_PinControl() function. This list
   will vary depending on the MCU chosen. */
typedef enum
{
    GPIO_CMD_OUT_CMOS = 0,
```

```
    GPIO_CMD_OUT_OPEN_DRAIN_N_CHAN,
    GPIO_CMD_OUT_OPEN_DRAIN_P_CHAN,
    GPIO_CMD_IN_PULL_UP_DISABLE,
    GPIO_CMD_IN_PULL_UP_ENABLE,
    GPIO_CMD_ASSIGN_TO_PERIPHERAL,
    GPIO_CMD_ASSIGN_TO_GPIO,
    GPIO_CMD_DSCR_DISABLE,
    GPIO_CMD_DSCR_ENABLE,
    GPIO_CMD_DSCR2_DISABLE,
    GPIO_CMD_DSCR2_ENABLE
} gpio_cmd_t;
```

2.10 Return Values

This section describes return values of API functions. This enumeration is located in r_gpio_rx_if.h as are the prototype declarations of API functions.

Below are the available return values for the R_GPIO_PinControl() function.

```
/* Function return type. */
typedef enum
{
    GPIO_SUCCESS = 0,
    GPIO_ERR_INVALID_MODE,  // The mode specified cannot be applied to this pin
    GPIO_ERR_INVALID_CMD    // The input command is not supported
} gpio_err_t;
```

2.11 Callback Function

None.

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.13 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
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

R_GPIO_PortWrite

This function writes the levels of all pins on a port.

Format

```
void    R_GPIO_PortWrite (
        gpio_port_t     port,
        uint8_t         value
)
```

Parameters

gpio_port_t port

Which port to write to. See Section 2.9.1, Ports.

uint8_t value

The value to write to the port. Each bit corresponds to a pin on the port (e.g. bit 0 of value will be written to pin 0 on supplied port)

Return Values

None.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

The input value will be written to the specified port. Each bit in the value parameter corresponds to a pin on the port. For example, bit 7 of write value corresponds to pin 7, bit 6 corresponds to pin 6, and so forth.

Example

```
/* Write 0xAA to Port 5. */
R_GPIO_PortWrite(GPIO_PORT_5, 0xAA);
```

Special Notes:

In the interest of performance, this function does not automatically check for non-existent pins when the port-wide write function is called. It is up to the user’s application to insure that only valid pins are written to.

R_GPIO_PortRead

This function reads the levels of all pins on a port.

Format

```
uint8_t                 R_GPIO_PortRead (
                gpio_port_t     port
)
```

Parameters

gpio_port_t port

Which port to read. See Section 2.9.1, Ports.

Return Values

The value of the port.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 of read value corresponds to pin 7, bit 6 corresponds to pin 6, and so forth.

Example

```
uint8_t    port_5_value;

/* Read Port 5. */
port_5_value = R_GPIO_PortRead(GPIO_PORT_5);
```

Special Notes:

None.

R_GPIO_PortDirectionSet

This function sets multiple pins on a port to inputs or outputs at once.

Format

```
void    R_GPIO_PortDirectionSet (
            gpio_port_t     port,
            gpio_dir_t      dir,
            uint8_t         mask
)
```

Parameters

gpio_port_t port
Which port to use. See Section 2.9.1, Ports.

gpio_dir_t dir
Which direction to use. See Section 2.9.5, Pin Direction.

uint8_t mask
Mask of which pins to change. 1 = set direction, 0 = do not change.

Return Values

None.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 of mask corresponds to pin 7, bit 6 corresponds to pin 6, and so forth. If a bit is set to 1 then the corresponding pin will be changed to an input or output as specified by the dir parameter. If a bit is set to 0 then the direction of the pin will not be changed.

Example

```
/* Set Pins 0, 1, and 5 as inputs on Port A. */
R_GPIO_PortDirectionSet(GPIO_PORT_A, GPIO_DIRECTION_INPUT, 0x23);

/* Set Pins 2, 3, 4, 6, and 7 as outputs on Port A. */
R_GPIO_PortDirectionSet(GPIO_PORT_A, GPIO_DIRECTION_OUTPUT, 0xDC);
```

Special Notes:

This function does not allow the user to specify the use of special modes such as input pull-up resistors or open-drain outputs. To enable these modes use the R_GPIO_PinControl() function.

R_GPIO_PinWrite

This function sets the level of a pin.

Format

```
void    R_GPIO_PinWrite (
        gpio_port_pin_t         pin,
        gpio_level_t            level
)
```

Parameters

gpio_port_pin_t pin

Which pin to use. See Section 2.9.2, Pins.

gpio_level_t level

What level to set the pin to.

Return Values

None.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

Pins can either be set as high (‘1’) or low (‘0’).

Example

```
/* Set Port E Pin 0 high. */
R_GPIO_PinWrite(GPIO_PORT_E_PIN_0, GPIO_LEVEL_HIGH);

/* Set Port 3 Pin 2 low. */
R_GPIO_PinWrite(GPIO_PORT_3_PIN_2, GPIO_LEVEL_LOW);
```

Special Notes:

None.

R_GPIO_PinRead

This function reads the level of a pin.

Format

```
gpio_level_t      R_GPIO_PinRead (
        gpio_port_pin_t           pin
)
```

Parameters

gpio_port_pin_t pin

Which pin to use. See Section 2.9.2, Pins.

Return Values

The level of the specified pin.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

The specified pin will be read and the level returned.

Example

```
/* Check level of Port 5 Pin 4. */
if (R_GPIO_PinRead(GPIO_PORT_5_PIN_4) == GPIO_LEVEL_HIGH)
{
    …
}
else
{
    …
}
```

Special Notes:

None.

R_GPIO_PinDirectionSet

This function sets the direction (input/output) of a pin.

Format

```
void    R_GPIO_PinDirectionSet (
        gpio_port_pin_t         pin,
        gpio_dir_t              dir
)
```

Parameters

gpio_port_pin_t pin

Which pin to use. See Section 2.9.2, Pins.

gpio_dir_t dir

Which direction to use for this pin. See Section 2.9.5, Pin Direction.

Return Values

None.

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

This function sets pins as inputs or outputs. For enabling other settings such as open-drain outputs or internal pull-ups see the R_GPIO_PinControl() function.

Example

```
/* Set Port E Pin 0 as an output. */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_0, GPIO_DIRECTION_OUTPUT);

/* Set Port 3 Pin 2 as an input. */
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_2, GPIO_DIRECTION_INPUT);
```

Special Notes:

None.

R_GPIO_PinControl

This function allows the user to control various settings of a pin.

Format

```
gpio_err_t      R_GPIO_PinControl (
        gpio_port_pin_t         pin,
        gpio_cmd_t              cmd
)
```

Parameters

gpio_port_pin_t pin
Which pin to use. See Section 2.9.2, Pins.

gpio_cmd_t cmd
Which command to execute for this pin. See Section 2.9.6, Control Commands for available commands.

Return Values

```
[GPIO_SUCCESS]          /* Successful; pin modified as specified by command.    */
[GPIO_ERR_INVALID_MODE] /* Error; this pin does not support the specified option. */
[GPIO_ERR_INVALID_CMD]  /* Error; the input command is not supported.           */
```

Properties

Prototyped in file “r_gpio_rx_if.h”

Description

Depending on the MCU, pins have various settings that can be configured other than the direction and output level. Some examples include enabling open-drain outputs, internal pull-ups, and changing drive capacity levels. These features vary per chip which means that the options for this function will also vary.

Example

```
gpio_err_t gpio_err;

/* Set Port E Pin 0 as a CMOS output (default). */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_0, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_OUT_CMOS);

/* Configure Port E Pin 0 as a high-current output */
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_DSCR_ENABLE);

/* Configure Port E Pin 0 as a high-speed interface high-drive output */
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_DSCR2_ENABLE);

/* Set Port E Pin 1 as a P-channel open-drain output. */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_1, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_1,GPIO_CMD_OUT_OPEN_DRAIN_P_CHAN);

/* Set Port E Pin 2 as an N-channel open-drain output. */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_2, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_2,GPIO_CMD_OUT_OPEN_DRAIN_N_CHAN);
```

```
/* Set Port 3 Pin 2 as input with pull-up disabled (default). */
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_2, GPIO_DIRECTION_INPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_3_PIN_2, GPIO_CMD_IN_PULL_UP_DISABLE);

/* Set Port 3 Pin 3 as input with pull-up enabled. */
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_3, GPIO_DIRECTION_INPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_3_PIN_3, GPIO_CMD_IN_PULL_UP_ENABLE);

/* Port 2 Pin 6 will be used as TXD1 for SCI peripheral. */
R_GPIO_PinDirectionSet(GPIO_PORT_2_PIN_6, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_2_PIN_6, GPIO_CMD_ASSIGN_TO_PERIPHERAL);

/* Port 5 Pin 4 will be used as GPIO. */
gpio_err |= R_GPIO_PinControl(GPIO_PORT_5_PIN_4, GPIO_CMD_ASSIGN_TO_GPIO);

/* GPIO_SUCCESS is set to 0 so if gpio_err is not 0 then an error occurred
above. You could check gpio_err after every function call if needed. */
if (GPIO_SUCCESS != gpio_err)
{   /* Handle the error.  */}
```

R_GPIO_GetVersion

Returns the current version of this API.

Format
uint32_t R_GPIO_GetVersion (void)

Parameters
None.

Return Values
Version of this API.

Properties
Prototyped in file “r_gpio_rx_if.h”

Description
This function will return the version of the currently running API. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number. For example, Version 4.25 would be returned as 0x00040019.

Example
```
uint32_t cur_version;

/* Get version of running r_gpio_rx API. */
cur_version = R_GPIO_GetVersion();

/* Check to make sure version is new enough for this application’s use. */
if (MIN_VERSION > cur_version)
{
    /* This r_gpio_rx version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ….
}
```

Special Notes:
None.

4. Pin Setting

GPIO FIT module don’t use pin setting.

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

5.1 gpio_demo_rskrx113

The **gpio_demo_rskrx113** program demonstrates how to set the direction of an IO port as an input or output and how to read or write it. Once the demo code has been compiled and down-loaded to the target board and is running, the demo will flash LED2 three times to show that the demo is running then, wait for key-presses on SW1. LED2 is turned on while SW1 is pressed and off when it is released.

5.2 gpio_demo_rskrx231, gpio_demo_rskrx64m, gpio_demo_rskrx71m, gpio_demo_rskrx65n, gpio_demo_rskrx65n_2m

The **gpio_demo_rskrx231, gpio_demo_rskrx64m**, **gpio_demo_rskrx71m, gpio_demo_rskrx65n and gpio_demo_rskrx65n_2m** demo programs operate the same. They demonstrate how to set up a port pin as an input or output and how to read or write it. They also demonstrate how to configure an output pin for high-current drive.
Once the code has been compiled and down-loaded to the target board and is running, LED3 will flash three times to show that the demo is running then, wait for key-presses on SW1. LED3 is turned on while SW1 is pressed and off when it is released.

5.3 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo
project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.4 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes confirmed operation environment for the GPIO FIT module.

Table 6.1 Confirmed Operation Environment (Rev.3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
Endian	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.00
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxx)

Table 6.2 Confirmed Operation Environment (Rev.2.50)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.50
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 6.3 Confirmed Operation Environment (Rev.2.41)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.41
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.4 Confirmed Operation Environment (Rev.2.40)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.40
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.5 Confirmed Operation Environment (Rev.2.31)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.31
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.6 Confirmed Operation Environment (Rev.2.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.30
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- Using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_gpio_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_gpio_rx_config.h” may be wrong. Check the file “r_gpio_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7, Configuration Overview for details.

7. Reference Documents

User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
RX Family Compiler CC-RX User's Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov 15, 2013	—	First Release
1.20	April 23 ,2014	1	Additional supported MCUs listed. List of documents revised.
		3	Updated list of required BSP versions.
		3	Added note on limitation: Port Switching feature not supported.
		4-7	Corrected value assigned in references to PORT_J
		7	Added section 2.9.3 Port_Pin Masks
		11	Added note regarding writing to port bits for non-existent pins
1.30	June 3, 2014	1	Added mention of RX64M in the list of supported MCUs.
		3	Updated toolchain version.
		19	Updated formatting of section 3.9.
1.40	Nov 28, 2014	—	Added support for the RX113 Group.
		5	Added a Code Size section.
1.50	Mar 06, 2015	—	Added support for the RX71M Group.
		5	Updated the Code Size table for RX71M.
		10	Updated “Control Commands” to include DSCR.
1.60	June 30, 2015	—	Added support for the RX231 Group.
		5	Updated the Code Size table for RX231.
1.70	Sep 30, 2015	—	Added support for the RX23T Group.
		5	Updated the Code Size table for RX23T.
1.80	Oct 1, 2015	—	Added support for the RX130 Group.
		5	Updated the Code Size table for RX130.
1.90	Dec 1, 2015	—	Added support for the RX24T Group.
		1, 10	Changed the document number for the “Board Support Package Firmware Integration Technology Module” application note.
		4	Changed the description in section 2.
		5	Updated the Code Size table for RX24T.
		20	Added “4. Demo Projects”.
2.00	Feb 1, 2016	—	Added support for the RX230 Group.
		5	Updated the Code Size table for RX230.
		21	Added “Related Technical Updates”.
2.01	June 15, 2016	20	Added RSKRX64M to “4. Demo Projects”.
2.10	Oct 1, 2016	—	Added support for the RX65N Group.
		1	Changed the document number for the “Board Support Package Firmware Integration Technology Module” application note.
			Added RX65N Group to the target device.
		4	Added RX65N to “2.5 Supported Toolchains”.
		5	Updated the Code Size table for RX65N.
		10	Updated “Control Commands” to include DSCR2.
		19	Added DSCR2 command example to “3.8 R_GPIO_PinControl”.
		22	Added “5. Provided Modules”.
			Added “6. Reference Documents”.
		23	Updated Inquiries.

Rev.	Date	Description	
		Page	Summary
2.20	Feb 28, 2017	—	Added support for the RX24U Group.
		4	Added RXC v2.06.00 to “2.5 Supported Toolchains”.
		5	Updated the Code Size table for RX24U.
		Program	The following specifications have been additionally supported for the RX24T Group. - P36 and P37 - 64-pin package
2.30	Jul 21, 2017	—	Added support for the RX130-512KB and RX65N-2MB.
		4	Added RXC v2.07.00 to “2.5 Supported Toolchains”.
		10	Updated “2.12 Adding the FIT Module to Your Project”.
2.31	Oct 31, 2017	21	Added RSKRX65N, RSKRX65N-2MB to “4. Demo Projects” Added 4.4 Downloading Demo Project
		22	Added 5. Appendices
2.40	Sep 28, 2018	1	Added support for the RX66T.
		6	Added code size corresponding to RX66T
		24	6.1 Confirmed Operation Environment: Added table for Rev.2.40
2.41	Nov 16, 2018	—	Added document number in XML
		24	Changed Renesas Starter Kit Product No for RX66T. Added table for Rev.2.41
2.50	Feb 01, 2019	Program	Added support for RX72T.
		1	Added support for RX72T.
		6	Added code size corresponding to RX72T
		13-21	Removed ‘Reentrant’ description in each API function.
		24	6.1 Confirmed Operation Environment: Added Table for Rev 2.50
3.00	May.20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Deleted the RX210, RX631 in Target Devices for end of update these devices. Added the section of Target compilers. Deleted related documents.
		5	2.2 Software Requirements Requires r_bsp v5.20 or higher
		6	Updated the section of 2.8 Code Size
		24	Table 6.1 Confirmed Operation Environment: Added table for Rev.3.00
		28	Deleted the section of Website and Support.
		Program	Changed below for support GCC and IAR compiler: Deleted the inline expansion of the R_GPIO_GetVersion function.

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

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/.