RZ/A2M Group

RZ/A2M GPIO Driver

R01AN4395EG0100
Rev.1.0
Sept 25, 2018

Introduction

This application note describes the operation of the software GPIO driver for the RZ/A2 device on the RZ/A2M CPU board.

It provides a comprehensive overview of the driver. For further details please refer to the software driver itself.

The user is assumed to have knowledge of e^2 studio and to be equipped with an RZ/A2M CPU board.

Target Device

RZ/A2M Group

Driver Dependencies

This driver depends on:

- Drivers
  - STDIO

Referenced Documents

Document Type	Document Name	Document No.
User's Manual	RZ/A2M Hardware Manual	R01UH0746EJ

List of Abbreviations and Acronyms

Abbreviation	Full Form
ANSI	American National Standards Institute
GPIO	General Purpose Input Output
HLD	High Layer / Level Driver
LLD	Low Layer / Level Driver
MCU	Microcontroller Unit
OS	Operating System
PDR	Port Direction Register
PFS	Pin Function Control (Select) Register
PIDR	Port Input Data Register
PMR	Port Mode Register
PODR	Port Output Data Register
RTOS	Real Time Operating System
STDIO	Standard Input / Output

Table 1-1 List of Abbreviations and Acronyms

Contents

1. Outline of the GPIO Driver

This MCU provides a total of 22 sets of dedicated port pins and general-purpose input/output port pins P0 to PL, and JP00 and JP01. Each pin is also configurable as an I/O pin of a peripheral module or an input pin for an interrupt. All pins are set to nonuse immediately after a reset (Hi-Z input protection), and pin functions can be switched by register settings.

The general-purpose input/output port pins are multiplexed with the peripheral on-chip module functions. The functions multiplexed on these pins can be selected as desired by setting of the registers.

Each port has the port direction register (PDR) that selects non-use, input, or output, the port output data register (PODR) that holds data for output, the port input register (PIDR) that indicates the pin states, and the port mode register (PMR) that specifies the pin function of each port.

Each port has the control register (PFS) that is used to select the input/output function and interrupt pin from the multiplexed port pins, and to assign the function to the selected pin.

For further information regarding the hardware specifics of the GPIO peripheral please refer to the appropriate hardware manual.

The GPIO driver provides a layer of abstraction between these registers and the user application, allowing for simplified porting of the application to a different device.

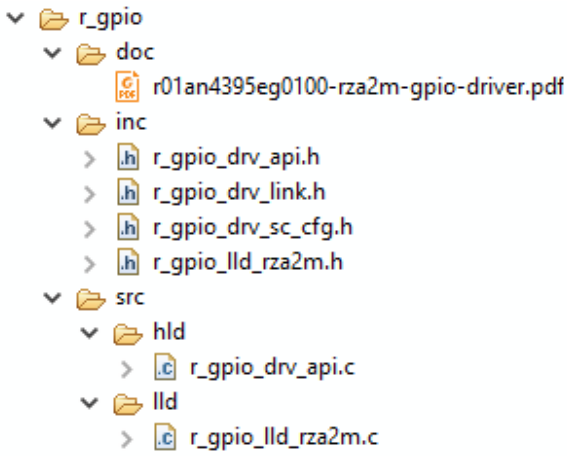
2. Description of the Software Driver

The key features of the driver include:

- Allocating individual pin function to GPIO or peripheral
- Setting GPIO pin to input / output / or Hi-Z (not used)
- Configuring GPIO pin to generate interrupts
- Writing the logic level on an output pin
- Reading the logic level of an input pin
- Setting the peripheral function for a pin allocated to a peripheral

2.1 Structure

The GPIO driver is split into two parts: The High Layer Driver (HLD) and the Low Layer Driver (LLD). The HLD includes platform independent features of the driver, implemented via the STDIO standard functions. The LLD includes all the hardware specific functions.



2.2 Description of each file

Each file's description can be seen in the following table.

Filename	Usage	Description
Application-Facing Driver API		
r_gpio_drv_api.h	Application	The only API header file to include in application code
High Layer Driver (HLD) Source		
r_gpio_drv_api.c	Private (HLD only)	High Layer Driver (HLD) source code implementing the driver API functions
Abstraction Link between High and Low Layer Drivers (HLD/LLD Link)		
r_gpio_drv_link.h	Private (HLD/LLD only)	Header file intended as an abstraction between low and high layers. This header will include the device specific configuration file "r_gpio_lld_rza2m.h"
Low Layer Driver (LLD) Source		
r_gpio_lld_rza2m.c	Private (LLD only)	Provides the implementation of the Low Layer Driver interface
r_gpio_lld_rza2m.h	Private (LLD only)	Low Layer Driver header file. Provides the High Layer Driver with access to the LLD driver functions
Smart Configurator		
r_gpio_drv_sc_cfg.h	Private (HLD/LLD only)	This file is intended to be used by Smart Configurator to pass setup information to the driver. This is not for application use

2.3 High Level Driver

The High Layer Driver can be either used through STDIO or through direct access. It is recommended not to mix both access methods.

The driver layer functions can be seen in the below table:

Return Type	Function	Description	Arguments	Return
int_t	**gpio_hld_open(** st_stream_ptr_t p_stream**)**	Driver initialisation interface is mapped to open function called directly using the st_r_driver_t GPIO driver handle g_gpio_driver: i.e. **g_gpio_driver.open()**	[in] **p_stream** driver handle.	**DRV_SUCCESS** Open Success **DRV_ERROR** Open Error
void	**gpio_hld_close(** st_stream_ptr_t p_stream**)**	Driver close interface is mapped to close function called directly using the st_r_driver_t GPIO driver structure g_gpio_driver: i.e. **g_gpio_driver.close()**	[in] **p_stream** driver handle.	**None**
int_t	**gpio_hld_read(** st_stream_ptr_t p_stream, uint8_t *p_buffer, uint32_t count**)**	Driver close interface is mapped to read function called directly using the st_r_driver_t GPIO driver structure g_gpio_driver: i.e. **g_gpio_driver.read()**	[in] **p_stream** driver handle. [out] **p_buffer** buffer for returned data. [in] **count** size of buffer.	**Amount of Data Read** **DRV_ERROR** Write Error
int_t	**gpio_hld_write(** st_stream_ptr_t p_stream, uint8_t *p_buffer, uint32_t count**)**	Driver write interface is mapped to write function called directly using the st_r_driver_t GPIO driver structure g_gpio_driver: i.e. **g_gpio_driver.write()**	[in] **p_stream** driver handle. [in] **p_buffer** data to send. [in] **count** size of data to send.	**DRV_SUCCESS** Write Success **DRV_ERROR** Write Error
int_t	**gpio_hld_control(** st_stream_ptr_t p_stream**,** uint32_t ctl_code**,** void* p_ctl_struct**)**	Driver control interface function Maps to ANSI library low level control function called directly using the st_r_driver_t GPIO driver structure g_gpio_driver: i.e. **g_gpio_driver.control()**	[in] **p_stream** driver handle. [in] **ctl_code** the type of control function to use. [in/out] **p_ctl_struct** required parameter is dependent upon the control function.	**DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

Return Type	Function	Description	Arguments	Return
int_t	**gpio_get_version(** st_stream_ptr_t p_stream**,** st_ver_info_ptr_t p_ver_info**)**	Driver get_version interface function maps to extended non-ANSI library low level get_version function called directly using the st_r_driver_t GPIO driver structure g_gpio_driver: i.e. **g_gpio_driver.get_version()**	[in] **p_stream** handle to the (pre-opened) channel. [out] **p_ver_info** handle to the (pre-opened) channel.	**DRV_SUCCESS** Operation Success

These high layer functions can be accessed either executed directly or through STDIO.

2.4 Low Layer Driver

The Low Layer Driver provides the functions to configure the hardware.

Return Type	Function	Description	Arguments	Return
e_r_drv_gpio_err_t	**R_GPIO_HWInitialise**(void)	Initialises the GPIO hardware	**None**	**GPIO_SUCCESS**
e_r_drv_gpio_err_t	**R_GPIO_HWUninitialise**(void)	Uninitialises the GPIO hardware	**None**	**GPIO_SUCCESS**
e_r_drv_gpio_err_t	**R_GPIO_PinSetConfiguration(** const st_r_drv_gpio_sc_config_t * p_config**)**	Set single pin configuration	**p_config** [in] configuration data	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinGetConfiguration(** st_r_drv_gpio_sc_config_t * p_config**)**	Get single pin configuration	**p_config** [out] configuration data	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_InitByPinList(** const r_gpio_port_pin_t * p_pin_list, uint32_t count**)**	Set pin configuration by a pin list and marks the pins as in use. The configuration is retrieved from the Smart Configurator table	**p_pin_list** [in] the pins to initialise **count** [in] the number of pins in the list	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_ClearByPinList(** const r_gpio_port_pin_t * p_pin_list, uint32_t count**)**	Clear pins by a pin list and mark the pins as not in use	**p_pin_list** [in] the pins to clear **count** [in] the number of pins in the list	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_InitByTable(** const st_r_drv_gpio_sc_config_t * p_table, uint32_t count**)**	Configures pins from an array of pin configurations and marks the pins as in use	**p_table** [in] array of pin configuration data **count** [in] number of pins in the array	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_ClearByTable(** const st_r_drv_gpio_sc_config_t * p_table, uint32_t count**)**	Clears pins from an array of pin configurations and marks the pins as not in use	**p_table** [in] array of pin configuration data **count** [in] number of pins in the array	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PortWrite(** r_gpio_port_t port, uint8_t value**)**	Sets the logic levels on the specified port	**port** [in] the port to write to **value** [in] the value to write	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure

Return Type	Function	Description	Arguments	Return
e_r_drv_gpio_err_t	**R_GPIO_PortRead(** r_gpio_port_t port, uint8_t * value**)**	Reads the logic levels on the specified port	**port** [in] the port to read from **value** [out] receives the value read from the port	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PortDirectionSet(** r_gpio_port_t port, e_r_drv_gpio_dir_t dir, uint8_t mask**)**	Sets the direction on specified pins of the port	**port** [in] the port to set **dir** [in] the direction to set **mask** [in] specifies which bits to set	This function is not yet implemented and returns **GPIO_ERR_INVALID_CFG**
e_r_drv_gpio_err_t	**R_GPIO_PinWrite(** r_gpio_port_pin_t pin, e_r_drv_gpio_level_t level**)**	Sets the logic level on the specified pin	**pin** [in] the port / pin to set **level** [in] the logic level to set it to	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinRead(** r_gpio_port_pin_t pin, e_r_drv_gpio_level_t * level**)**	Gets the logic level on the specified pin	**pin** [in] the port / pin to read **level** [out] the logic level read from the pin	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinDirectionSet(** r_gpio_port_pin_t pin, e_r_drv_gpio_dir_t dir**)**	Sets the direction on the specified pin (input, output or Hi-Z)	**pin** [in] the port / pin to set **dir** [in] the direction to set	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinDirectionGet(** r_gpio_port_pin_t pin, e_r_drv_gpio_dir_t *dir**)**	Gets the direction on the specified pin (input, output or Hi-Z)	**port_pin** [in] the port / pin to read **dir** [out] receives the direction currently set	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinAssignmentSet(** r_gpio_port_pin_t pin, e_r_drv_gpio_assign_t assignment**)**	Assigns a pin to GPIO or to a peripheral	**pin** [in] the port / pin to assign **assignment** [in] how to assign the pin (GPIO or peripheral)	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinAssignmentGet(** r_gpio_port_pin_t pin, e_r_drv_gpio_assign_t * p_assignment**)**	Determines whether a pin is assigned to GPIO or a peripheral	**pin** [in] the port / pin to read **assignment** [out] how the pin is assigned (GPIO or peripheral)	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure

Return Type	Function	Description	Arguments	Return
e_r_drv_gpio_err_t	**R_GPIO_PinTintSet(** r_gpio_port_pin_t pin, e_r_drv_gpio_tint_t tint**)**	Turns interrupts on or off for a pin	**pin** [in] the port / pin to set **tint** [in] whether to enable or disable interrupts	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinTintGet(** r_gpio_port_pin_t pin, e_r_drv_gpio_tint_t * p_tint**)**	Reads interrupt on / off setting for a pin	**pin** [in] the port / pin to get the interrupts for **p_tint** [out] the interrupt setting	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinDscrSet(** r_gpio_port_pin_t pin, e_r_drv_gpio_current_t current**)**	Sets pin DSCR current (4mA or 8mA)	**pin** [in] the port / pin to set the DSCR for **current** [in] the current to set	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinDscrGet(** r_gpio_port_pin_t pin, e_r_drv_gpio_current_t * p_current**)**	Gets pin DSCR current setting (4mA or 8mA)	**pin** [in] the port / pin to get the DSCR for **current** [out] the current to set	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PeripheralFunctionWrite(** r_gpio_port_pin_t pin, uint8_t psel**)**	Sets the peripheral function for a pin	**pin** [in] the port / pin to set **psel** [in] the code for the required peripheral function	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PeripheralFunctionRead(** r_gpio_port_pin_t pin, uint8_t * p_psel**)**	Gets the peripheral function setting for a pin	**pin** [in] the port / pin to get **psel** [out] the code for the current peripheral function setting	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinSetFunction(** r_gpio_port_pin_t pin, e_r_drv_gpio_function_t function**)**	Set the function of the specified pin	**pin** [in] the port / pin to set **function** [in] the function to set it to	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_PinGetFunction(** r_gpio_port_pin_t pin, e_r_drv_gpio_function_t * p_function**)**	Get the function of the specified pin	**pin** [in] the port / pin to get **function** [out] the current function setting	**GPIO_SUCCESS** on success or a **GPIO_ERR_X** error code on failure
e_r_drv_gpio_err_t	**R_GPIO_GetVersion(** *st_drv_info_t* *pinfo**)**	Get the version information of the low level driver	**pinfo** [out] Receives the version information	**DRV_SUCCESS**: Success **DRV_ERROR**: Error

3. Accessing the High Layer Driver

3.1 STDIO

The HLD's API can be accessed through the ANSI 'C' library <stdio.h>. The following table details the operation of each function:

Operation	Return	Function Details
open	gs_stdio_handle, unique handle to driver	open (DEVICE_IDENTIFIER "gpio", O_RDWR);
close	DRV_SUCCESS successful operation, or driver specific error	close (gs_stdio_handle);
read	Number of characters read, -1 on error	read (gs_stdio_handle, buff, data_length);
write	Number of characters written, -1 on error	write (gs_stdio_handle, buff, data_length);
control	DRV_SUCCESS control was process, or driver specific error	control (gs_stdio_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	get_version (DEVICE_IDENTIFIER "gpio", &drv_info);

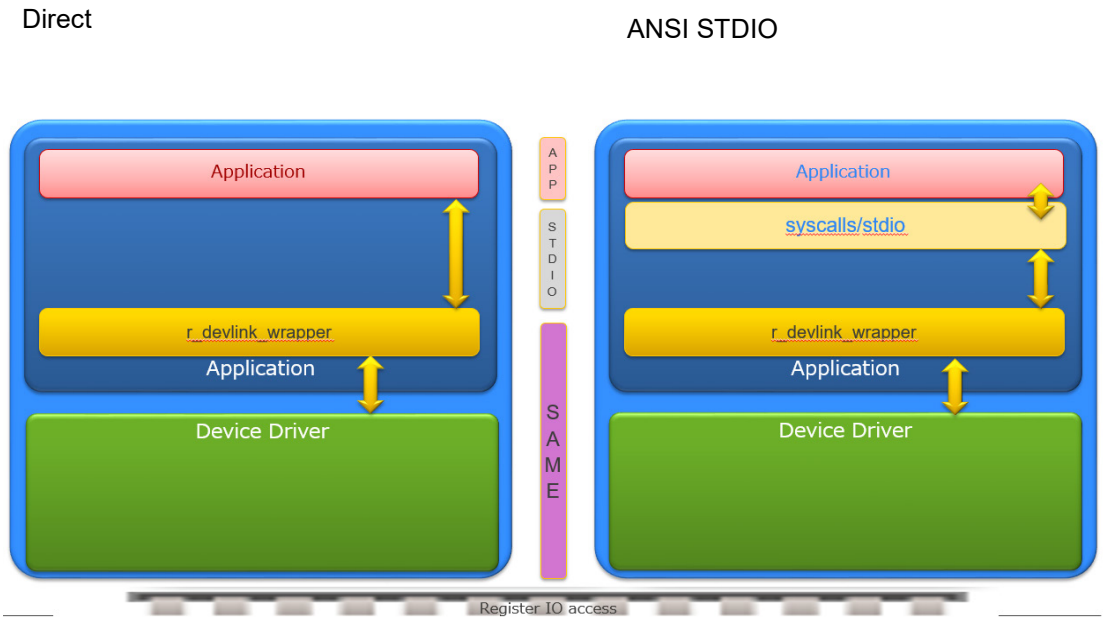
3.2 Direct

The following table shows the available direct functions.

Operation	Return	Function details
open	gs_direct_handle unique handle to driver	direct_open ("gpio", 0);
close	DRV_SUCCESS successful operation, or driver specific error	direct_close (gs_direct_handle);
read	Number of characters read, -1 on error	direct_read (gs_direct_handle, buff, data_length);
write	Number of characters written, -1 on error	direct_write (gs_direct_handle, buff, data_length);
control	DRV_SUCCESS control was process, or driver specific error	direct_control (gs_direct_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	direct_get_version ("gpio", &drv_info);

3.3 Comparison

The diagram below illustrates the difference between the direct and ANSI STDIO methods.



4. Example of Use

This section illustrates a simple example of opening the driver, configuring a port, writing to a pin, reading from a pin, and then closing the driver.

4.1 Open

```
int_t gs_gpio_handle;
char_t *driver_name = "gpio";

gs_gpio_handle = open(driver_name, O_RDWR);
```

4.2 Control – Set Configuration Settings

```
st_r_drv_gpio_config_t pin_configuration;
int_t result;

pin_configuration.config.pin = GPIO_PORT_0_PIN_2;
pin_configuration.config.configuration.function = GPIO_FUNC_OUT;
pin_configuration.config.configuration.tint = GPIO_TINT_DISABLE;
pin_configuration.config.configuration.current = GPIO_CURRENT_8mA;

result = control(handle, CTL_GPIO_SET_CONFIGURATION,
                                    (void *) &pin_configuration);
```

4.3 Control – Get Configuration Settings

```
st_r_drv_gpio_config_t pin_configuration;
int_t result;

result = control(handle, CTL_GPIO_GET_CONFIGURATION,
                                    (void *) &pin_configuration);
```

4.4 Write the Logic Level to a Pin

```
st_r_drv_gpio_pin_rw_t pin_read_write;
int_t result;

pin_read_write.pin = GPIO_PORT_0_PIN_6;
pin_read_write.level = GPIO_LEVEL_HIGH;
result = control(handle, CTL_GPIO_PIN_WRITE,
                                    (void *) &pin_read_write);
```

4.5 Read the Logic Level from a Pin

```
st_r_drv_gpio_pin_rw_t pin_read_write;
e_r_drv_gpio_level_t logic_level;
int_t result;

pin_read_write.pin = GPIO_PORT_0_PIN_6;
result = control(handle, CTL_GPIO_PIN_READ,
                                    (void *) &pin_read_write);
logic_level = pin_read_write.level;
```

4.6 Close

```
close(gs_gpio_handle);
```

4.7 Get Version

```
st_ver_info_t info;
result = get_version(gs_gpio_handle, &info);
```

5. OS Support

This driver supports any OS through using the OS Abstraction module. For more details about the abstraction module please refer to the OS abstraction module application note.

6. How to Import the Driver

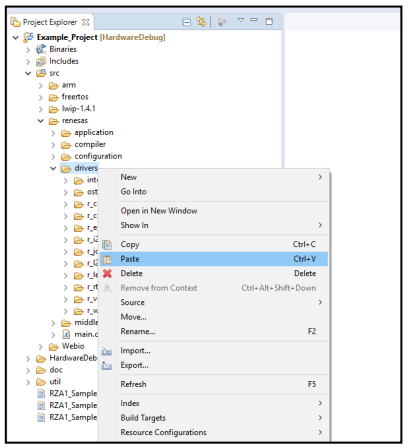
This section describes how to import the driver into your project. Generally, there are two steps in any IDE:

1) Copy the r_gpio driver to the location in the source tree that you require for your project.
2) Add the link to where you copied your driver to the compiler.

6.1 e² studio

To import the driver into your project please follow the instructions below.

1) In Windows Explorer, right-click on the r_gpio folder, and click **Copy**.
2) In e² studio Project Explorer view, select the folder where you wish the driver project to be located; right-click and click **Paste**.
3) Right-click on the parent project folder (in this case 'Example_Project') and click **Properties ...**
4) In 'C/C++ Build → Settings → Cross ARM Compiler → Includes', add the include folder of the newly added driver, e.g. '${ProjDirPath}\generate\drivers\r_gpio\inc'



Website and Support

Renesas Electronics website
https://www.renesas.com/

Inquiries
https://www.renesas.com/contact/

All trademarks and registered trademarks are the property of their respective owners.

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sept 25, 2018	All	Created document.

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. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   ¾ 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   ¾ The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   ¾ The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   ¾ 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   ¾ The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

© 2018 Renesas Electronics Corporation. All rights reserved.

Colophon 7.2