# RX200, RX100 Series

## Touch Button Module using Firmware Integration Technology

R01AN3822EU0100
Rev 1.00
May 9, 2017

### Introduction

This Firmware Integration Technology (FIT) Module implements the capacitive touch button middleware.

### Target Device

RX231, RX230, RX130, RX113.

### Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Touch Module Using Firmware Integration Technology (R01AN3820EU)
- CTSU Module Using Firmware Integration Technology (R01AN3821EU)
- Touch Slider Module Using Firmware Integration Technology (R01AN3823EU)
- Workbench6 User Manual (R20UT3842EJ)
- Capacitive Touch for RX Devices (R11AN0139EU)

# Contents

# 1. Overview of the Capacitive Touch Software Environment

The Capacitive Touch Sensing Unit (CTSU) measures the electrostatic capacitance of a touch sensor. The entirety of the driver stack is comprised of three different components: The CTSU layer, the touch layer, and the button/slider layer. The module covered in this documentation provides an abstraction layer for interpreting the data from the touch layer as button events. This includes tunable parameters such as interpreting different button pressed states, multitouch-events, debounces, and individualized callbacks for each buttons. For users implementing buttons into their system, this is the intended application program interface (API) to work through.

This module comes with python (2.7) scripts, which translate optimized parameters generated by the Renesas Workbench calibration tool to data that can be input to this driver. For more information on the translation process, using python refer to R11AN0139EU.

# 2. CTSU FIT Module

This module is intended for use along with the Renesas CTSU (r_ctsu_rx) lower level driver, touch (r_touch) middleware as shown in the illustration below, and is used by being incorporated into a FIT project when there is a need to interpret the digital counts from the CTSU layer as buttons.

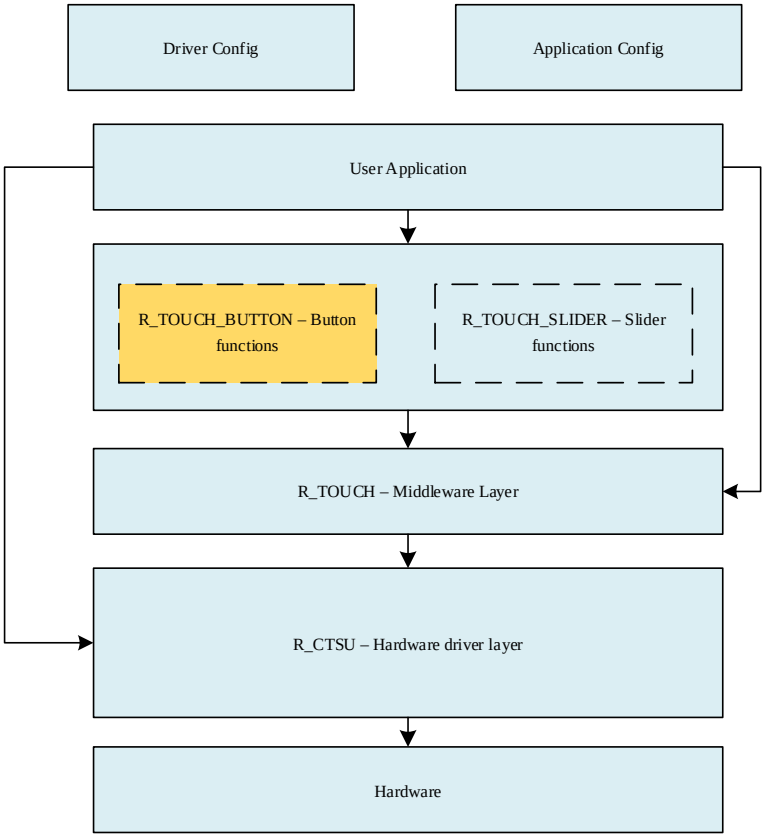


**Figure 1: Overview of the capacitive touch sensing solution and its various FIT modules**

The block diagram describing how the middleware interacts with the user application is described in Figure 1, the highlighted part is the r_touch_button FIT module which is described as the button layer. The button layer depends upon the binary provided by the touch layer to determine if a particular touch channel is touched or not. For API documentation the CTSU, touch and sliders layers refer to the following documents respectively- R01AN3821EU, R01AN3820EU, and R01AN3823EU.

Similar to the other modules used in this driver stack, the MCU must first be configured properly at the pin level by configuring the CTSU after power on reset by enabling the touch sensing (TS) and TSCAP pins. After proper input/output (I/O) initialization, the user will then have to open a button using the open function. Similar to the other APIs in this driver stack this function returns an index called a handle and a pointer to where the control block is stored in memory.

The open function ties to the lower levels of the driver to call both the open functions of the touch and CTSU layers and configures the firmware in such a way to create callbacks each time a button is pressed. An example of this process is shown in Figure 2.

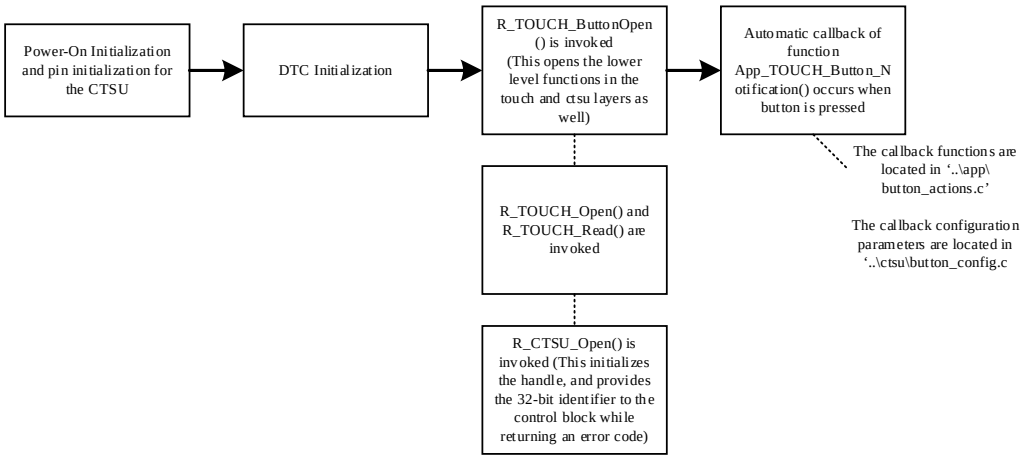


**Figure 2: Example initialization process**

The button actions are configured to match the configuration from Workbench; an example configuration is shown below.

```
touch_button_cfg_t const Button_RX30_TX05_on_g_touch_cfg_on_g_ctsu_cfg_mutual0
= {
      .button.rx = 30,
      .button.tx = 5,
      .debounce = 20,
      .hold_max = 1000,
      .p_callback = App_TOUCH_Button_Notification,
      .enable.press = true,
      .enable.release = true,
      .enable.hold =false,
      .p_touch_cfg = &g_touch_cfg_on_g_ctsu_cfg_mutual0,
};
```

The action the application takes when a button is interacted with is called via the function pointer (p_callback) and the function being called is shown below.

```
void App_TOUCH_Button_Notification(void * p_args)
{
   touch_button_callback_arg_t * p_arg = p_args;
   uint32_t button_id = p_arg->id;
   touch_button_event_t button_event = (touch_button_event_t)p_arg->event;

   (void)button_id;
   (void)button_event;

   if(TOUCH_BUTTON_EVENT_REQUEST_DELAY==button_event)
   {
     R_BSP_SoftwareDelay(1, BSP_DELAY_MILLISECS);
   }
}
```

# 3. FIT Module Specifications

- Transmit and Receive channels for each Capacitive Touch Button can be defined. Maximum of one combination consisting of two channels.
- Button generates events when sensor is Touched, Released from Touch, Touched for a specified amount of time, and Multi-touch i.e. another touch sensor was touched along with this button.
- Events generate a callback with a unique identifier and trigger type as an argument.
- All active buttons are automatically updated when the lower touch layer parameters are updated.
- Operates independently of other middle-ware layers interpreting the touch layer data.

# 4. API Information

This driver follows the Renesas API naming standards

## 4.1 Hardware Requirements

This driver requires that your MCU support the following features:

- CTSU peripheral
- (Optional) Data Transfer Controller (DTC)

## 4.2 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_ctsu_rx
- r_touch_rx

## 4.3 Limitations to this driver include:

- This module does not support CTSU operation in self-capacitance single scan mode.
- Port pins to use as TSCAP and TS Pins must be configured using the general purpose input output (GPIO) and multi-function pin controller (MPC) FIT modules.
- When DTC is used, the DTC vector table (DTCVBR) indexes 60 and 61 should be mutable.

## 4.4 Supported Toolchains

This driver is tested and works with the following toolchain:

- Renesas RX Toolchain v.2.05

## 4.5 Header Files

All API calls and their supporting interface definitions are located in r_ctsu_rx_if.h

## 4.6 Integer Types

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

## 4.7 Configuration Overview

The configuration options in this module are specified in r_touch_button_if.h. The option names and setting values are listed in the table below where applicable 0 represents disabled and 1 represents enabled:

| Configuration option | Description |
|---|---|
| BUTTON_CFG_PARAM_CHECKING_ENABLE | Specify whether to include code for API parameter checking. (Default = Enabled) |
| BUTTON_CFG_MAX_CONTROL_BLOCK_COUNT | Define maximum control blocks the user is allowed to open. (Default=1) |

## 4.8 Code Size

The table lists values when the compile options are set to default values (with parameter checking disabled) and the RX130 Group is used. The required memory size varies depending on the C compiler version and compile options. Table does not reflect memory consumed by the configuration as it varies depending upon the sensor count.

| Memory Type | Size (bytes) | Remarks |
|---|---|---|
| ROM | 849 | Constant + Program + Initialized Data |
| RAM | 64 | Data + Uninitialized Data |
| Maximum User Stack usage | 496 | From R_TOUCH_ButtonOpen |
| Maximum Interrupt Stack usage | 280 | excep_ctsu_ctsufn + touch_common_callback + touch_button_callback |

## 4.9 Parameters and Header files

This section describes the structure instances, which need to be defined by the user while creating a touch button configuration. This structure is located in r_touch_button_if.h as are the prototype declarations of API functions.

```
/** Definition of a button and it's behavior in software provided by the
Application. */
typedef struct st_touch_button
{
    touch_sensor_t button;              // Define the channel pairs which make
                                        // up a button.

    touch_cfg_t * const p_touch_cfg;    // Pointer to lower level Touch
                                       // configuration with which this button
                                       // operates.
    union
    {
        struct{
            uint8_t  release : 1;       // enable release events
            uint8_t  press   : 1;       // enable press events
            uint8_t  hold    : 1;       // enable hold events
        };
        uint8_t byte;                   // Byte representation of events
                                        // enabled.
    } enable;
    uint16_t    debounce;               // Debounce threshold for the sensor
    uint16_t hold_max;                  // Maximum duration of a valid touch
                                       // (for reporting a HOLD event)

    fit_callback_t p_callback;          // Function invoked when an button
                                       // event occurs
} touch_button_cfg_t;
```

## 4.10 Return Values

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

```
typedef enum e_touch_button_err
{
    TOUCH_BUTTON_SUCCESS = TOUCH_SUCCESS,   // Operation Successful
    TOUCH_BUTTON_ERR_INVALID_PARAM,         // Invalid Parameter Found in
                                           // arguments.

    TOUCH_BUTTON_ERR_INSUFFICIENT_MEMORY,   // Insufficient memory (consider
                           // increasing BUTTON_CFG_MAX_CONTROL_BLOCK_COUNT)

    TOUCH_BUTTON_ERR_LOCKED,     // Button Control block is currently in use.
}touch_button_err_t;
```

## 4.11 Events

The touch button layer receives events from the lower touch layer and generates events in response to data interpreted from the lower layers. Upper layers (usually the application) are notified by means of a callback function specified as part of the button configuration.

When an event is passed to the upper layer, the argument structure used is as shown below:

```
typedef struct st_touch_button_callback_arg
{
    uint32_t id;                      // Button Identifier (matches identifier
                                     // returned by successful call to
                                     // R_Touch_Button_Open)

    uint32_t event;                    // Type of event occurred

    void const * p_context;           // Placeholder for user data (Future use)

}touch_button_callback_arg_t;
```

A button generates the following event types:

| Event Type | Notes |
|---|---|
| TOUCH_BUTTON_EVENT_PRESSED | Touch Sensor has been debounced and has reached a “Touched” state. |
| TOUCH_BUTTON_EVENT_RELEASED | Touch sensor is no longer in “Touched” state. |
| TOUCH_BUTTON_EVENT_HOLD | Touch sensor was held down for the “valid press” timing specified by the user. |
| TOUCH_BUTTON_EVENT_MULTI_TOUCH | Touch sensor has been debounced and has reached a “Touched” state, but another Touch sensor is also detected as touched. |
| TOUCH_BUTTON_EVENT_REQUEST_DELAY | Lower layer is requesting a 1 ms delay. |

## 4.12 Adding the FIT module to your project

This section outlines the process for integrating the generated FIT style drivers into an already created project, after the board has been tuned using Workbench6. For information regarding instillation for the entirety of the driver stack, please refer to (R11AN0138EU).

1. Ensure that the User Project uses FIT, and is based on r_bsp v3.40 or later.
2. Copy the module r_touch_button to the FIT project from the Base Project.

   Add the following locations to the compile include paths: "${workspace_loc:/${ProjName}/r_touch_button}"

   Copy the following files from the Base Project's r_config folder to the User Project: r_touch_button_config.h
3. Ensure that the Parameter Checking preprocessor is enabled for all layers using the files mentioned in item 4. Note: This can be turned off once all layers are verified to be operate correctly.
4. Copy the folders src/app and src/ctsu from the Base Project and place them in the User Project.
5. Ensure that the clock settings in r_bsp_config.h match for both the Base and User Project.
6. Initialize Multi-function Pin Controller and General Purpose Input Output for all CTSU TS Pins used.
7. Acquire the Touch handle from the Button handle via the R_TOUCH_ButtonOpen() function.
8. Call the touch Scan/Update function to operate the CTSU and perform touch detection.

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the "Board Support Package Module Using Firmware Integration Technology" application note (R01AN1685EU).

# 5. API Functions

## 5.1 Summary

Table below lists the API functions.

| Function | Description |
|---|---|
| R_TOUCH_ButtonOpen | Initialize memory local to the module for a TOUCH Button. |
| R_TOUCH_ButtonClose | Close a control block that was previously opened. |
| R_TOUCH_ButtonControl | Change parameter values of a button control block. |
| R_TOUCH_ButtonGetVersion | Returns the driver version number at runtime |

## 5.2 R_TOUCH_ButtonOpen

This function is required to be invoked first before any other API calls with this module are used, this initializes memory local to the module for a TOUCH button.

### Format

```
touch_button_err_t R_TOUCH_ButtonOpen(uint32_t * const p_hdl,
touch_button_cfg_t const * const p_cfg)
```

### Parameters

- p_hdl - Pointer to location where user wants the index of the Button Control block to be returned.
- p_cfg - Pointer to button configuration information.

### Return Values

- TOUCH_BUTTON_SUCCESS - Operation Successful
- TOUCH_BUTTON_ERR_INVALID_PARAM - Invalid Parameter Found in arguments.
- TOUCH_BUTTON_ERR_INSUFFICIENT_MEMORY - Insufficient memory (consider increasing BUTTON_CFG_MAX_CONTROL_BLOCK_COUNT)

### Properties

Prototyped in r_touch_button_if.h.

### Description

This function will initialize a Button control block. The lower layers for Capacitive Touch Detection/Sensing will be opened and initialized.

### Re-entrant

Function is re-entrant for different configurations passed.

## Example

```
void my_button_initializer(void)
{
    uint32_t btn_hdl = UINT32_MAX;
    touch_button_err_t button_err = TOUCH_BUTTON_SUCCESS;

    touch_button_cfg_t Button_RX00_on_g_touch_cfg_on_g_ctsu_cfg_self = {
         .button.rx = 0,
         .button.tx = 255,
         .debounce = 20,
         .hold_max = 1000,
         .p_callback = g_touch_cfg_on_g_ctsu_cfg_self_button_callback,
         .enable.press = true,
         .enable.release = true,
         .enable.hold = false,
         .p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self01,
    };

    button_err = R_TOUCH_ButtonOpen(&btn_hdl,
&Button_RX00_on_g_touch_cfg_on_g_ctsu_cfg_self);

    if(TOUCH_BUTTON_SUCCESS==button_err)
    {
        /* Things to do on a successful open */
    }
}
```

## Special Notes

None.

## 5.3 R_TOUCH_ButtonClose

Close a Touch Button Control block.

### Format

```
touch_button_err_t R_TOUCH_ButtonClose(uint32_t hdl)
```

### Parameters

- hdl - Control block identifier for the button. (Returned by successful call to R_TOUCH_ButtonOpen)

### Return Values

- TOUCH_BUTTON_SUCCESS - Operation Successful
- TOUCH_BUTTON_ERR_INVALID_PARAM - Invalid Parameter Found in arguments.
- TOUCH_BUTTON_ERR_LOCKED - Button Control block is currently in use

### Properties

Prototyped in r_touch_button_if.h.

### Description

This function un-initializes a button control block. If all buttons using common lower layers are closed, then the lower layers will also be un-initialized.

### Re-entrant

Function is reentrant for different identifiers passed.

## Example

```
void my_button_uninitializer(void)
{
    uint32_t btn_hdl = UINT32_MAX;
    touch_button_err_t button_err = TOUCH_BUTTON_SUCCESS;

    touch_button_cfg_t Button_RX00_on_g_touch_cfg_on_g_ctsu_cfg_self = {
         .button.rx = 0,
         .button.tx = 255,
         .debounce = 20,
         .hold_max = 1000,
         .p_callback = g_touch_cfg_on_g_ctsu_cfg_self_button_callback,
         .enable.press = true,
         .enable.release = true,
         .enable.hold = false,
         .p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self01,
    };

    button_err = R_TOUCH_ButtonOpen(&btn_hdl,
&Button_RX00_on_g_touch_cfg_on_g_ctsu_cfg_self);

    if(TOUCH_BUTTON_SUCCESS==button_err)
    {
        /* Things to do on a successful open */
    }

    button_err = R_TOUCH_ButtonClose(btn_hdl);

    if(TOUCH_BUTTON_SUCCESS==button_err)
    {
        /* Things to do on a successful close */
    }
}
```

## Special Notes

None.

## 5.4 R_TOUCH_ButtonControl

Perform a single scan with a parameters stored in a successfully opened control block.

### Format

```
touch_button_err_t R_TOUCH_ButtonControl(uint32_t hdl,
touch_button_control_arg_t * const p_arg)
```

### Parameters

- hdl - Button Control block being addressed.
- p_arg - Command and destination of information to read/write.

### Return Values

- TOUCH_BUTTON_SUCCESS - Operation Successful
- TOUCH_BUTTON_ERR_INVALID_PARAM - Invalid Parameter Found in arguments.
- TOUCH_BUTTON_ERR_LOCKED - Button Control block is currently in use.

### Properties

Prototyped in r_touch_button_if.h.

### Description

The additional argument taken by this function has a structure as shown below:

```
/**
 * Argument structure passed to the R_Touch_Button_Control function.
 */
typedef struct st_touch_button_control_arg
{
    touch_control_cmd_t cmd;       // Control Command type
    void * p_dest;                 // Pointer to location where data is stored

    size_t size;                   // Amount of memory available at the
                                  // location pointed by
                                  // st_touch_button_control_arg::p_dest.
}touch_button_control_arg_t;
```

The following is a list of actions to be performed in this function:

1. Check the handle identifier provided and selects resource (exit if resource block is locked).
2. Write to the data location identified by the command and change the data to value pointed by p_dest.
3. The user must ensure there is sufficient memory for the API to copy the requested information.

The following Control Commands are supported by the driver:

| Command | Data Type | Description |
|---|---|---|
| TOUCH_BUTTON_GET_CHANNELS | UINT8x2 | Get TS pins (touch_sensor_t) used by button |
| TOUCH_BUTTON_GET_ENABLE_MASK | UINT8 | Get enabled events mask |
| TOUCH_BUTTON_GET_CALLBACK | UINT32 | Get the callback function for the button |
| TOUCH_BUTTON_GET_TOUCH_HANDLE | UINT32 | Get the identifier of lower TOUCH layer. |
| TOUCH_BUTTON_GET_STATE | UINT32 | Get the state of the button |
| TOUCH_BUTTON_GET_DEBOUNCE_COUNTER | UINT16 | Get the value of the debounce counter |
| TOUCH_BUTTON_GET_HOLD_COUNTER | UINT16 | Get the value of the hold counter for the button. |
| TOUCH_BUTTON_SET_CHANNELS | UINT8x2 | Set TS pins (touch_sensor_t) used by button |
| TOUCH_BUTTON_SET_ENABLE_MASK | UINT8 | Set enabled events mask |
| TOUCH_BUTTON_SET_CALLBACK | UINT32 | Set the callback function for the button |
| TOUCH_BUTTON_SET_TOUCH_HANDLE | UINT32 | Set the identifier of lower TOUCH layer. |
| TOUCH_BUTTON_SET_STATE | UINT32 | Set the state of the button |
| TOUCH_BUTTON_SET_DEBOUNCE_COUNTER | UINT16 | Set the value of the debounce counter |
| TOUCH_BUTTON_SET_HOLD_COUNTER | UINT16 | Set the value of the hold counter for the button. |

## Re-entrant

Function is reentrant for different identifiers passed.

## Example

```
void my_button_info(uint32_t btn_hdl)
{
    touch_button_err_t button_err = TOUCH_BUTTON_SUCCESS;
    fit_callback_t btn_callback = FIT_NO_PTR;
    touch_button_control_arg_t change_arg = {
        .cmd = TOUCH_BUTTON_GET_CALLBACK
        .p_dest = &btn_callback
        .size = sizeof(btn_callback),
    };

    button_err = R_TOUCH_ButtonControl(btn_hdl, &change_arg);

    if(TOUCH_BUTTON_SUCCESS==button_err)
    {
        /* Things to do on a successful call */
    }
}
```

## Special Notes

None.

## 5.5 R_TOUCH_ButtonGetVersion

This function returns the driver version number at runtime

### Format

```
uint32_t R_TOUCH_ButtonGetVersion (void)
```

### Parameters

None.

### Return Values

Version of this module in 32-bits.

### Properties

Prototyped in r_touch_button_if.h.

### Description

Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

### Re-entrant

Yes.

### Example

```
void my_func(void)
{
    uint32_version;
    version = R_TOUCH_ButtonGetVersion();
}
```

### Special Notes

This function is inlined using the #pragma inline directive.

## Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://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 | May 9, 2017 | - | Initial Version Released |

## 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.

   — 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.

   — 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.

   — 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.

   — 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.

   — 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 disputes 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, drawing, chart, program, algorithm, 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 otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
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; and 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.

   Renesas Electronics products are neither intended nor 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 damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond 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. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate 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. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use 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 not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
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 majority-owned subsidiaries.

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

(Rev.3.0-1 November 2016)

RENESAS

## SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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


**Renesas Electronics America Inc.**
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**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: +44-1628-585-100, Fax: +44-1628-585-900

**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, P. R. China 200333
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 II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

**Renesas Electronics Korea Co., Ltd.**
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141



© 2017 Renesas Electronics Corporation. All rights reserved.


Colophon 6.0