RX Family

QE Touch module Firmware Integration Technology

Introduction

This application note describes the TOUCH Module.

Target Device

- RX113 Group
- RX130 Group
- RX230 Group
- RX231 Group
- RX23W Group
- RX671 Group
- RX140 Group
- RX260 Group
- RX261 Group

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

Related Documents

RX Family QE CTSU module Firmware Integration Technology (R01AN4469)
QE Touch Diagnostic API Users' Guide (R01AN4785EU)
Firmware Integration Technology User's Manual (R01AN1833)
Board Support Package Firmware Integration Technology Module (R01AN1685)
Adding Firmware Integration Technology to Projects (R01AN1723)
RX100 Series VDE Certified IEC60730 Self-Test Code (R01AN2061ED)
RX v2 Core VDE Certified IEC60730 Self-Test Code for RX v2 MCU (R01AN3364EG)

Contents

1. Overview 3

1.1 Functions 3

1.1.1 QE for Capacitive Touch Usage 3

1.1.2 Measurements and Data Processing 3

1.1.3 Button Touch Determination 3

1.1.4 Touch Position Detection of Slider/Wheel 6

1.1.5 Tuning the Touch Determination Threshold 7

1.2 API Overview 9

2. API Information 10

2.1 Hardware Requirements 10

2.2 Software Requirements 10

2.3 Supported Toolchains 10

2.4 Restrictions 10

2.5 Header File 10

2.6 Integer Type 10

2.7 Compilation Settings 11

2.8 Code Size 12

2.9 Arguments 13

2.10 Return Values 15

2.11 Adding the FIT Module to Your Project 16

2.11.1 Adding source tree and project include paths 16

2.11.2 Setting driver options when not using Smart Configurator 16

2.12 API compatibility mode 16

3. API Functions 17

3.1 RM_TOUCH_Open 17

3.2 RM_TOUCH_ScanStart 18

3.3 RM_TOUCH_DataGet 19

3.4 RM_TOUCH_CallbackSet 20

3.5 RM_TOUCH_Close 21

3.6 RM_TOUCH_ScanStop 22

3.7 RM_TOUCH_SensitivityRatioGet 23

3.8 RM_TOUCH_ThresholdAdjust 24

3.9 RM_TOUCH_DriftControl 25

3.10 RM_TOUCH_MonitorAddressGet 26

1. Overview

The TOUCH Module is middleware that uses the CTSU module to provide capacitive touch detection. The TOUCH module assumes access from the user application is possible.

1.1 Functions

The TOUCH module supports the following functions.

1.1.1 QE for Capacitive Touch Usage

Similar to the CTSU module, this module provides various capacitive touch detections based on configuration settings generated by QE for Capacitive Touch (referred to as QE)

As a part of the configuration settings, the touch interface configuration displays configuration information for the CTSU link information and buttons, sliders, and wheels. A multiple touch interface configuration is necessary when both self and mutual capacitance buttons are used in the same product or when using the active shield function.

This module also supports the QE monitor function. The monitor determines whether to use debugger or serial communications, determines the type of information from QE and sends only the necessary information.

This module also supports QE serial tuning function. Sensor drive pulse frequency, measurement time and threshold are determined by connecting to QE with UART communication.

1.1.2 Measurements and Data Processing

The module determines whether the button has been touched based on the change in capacitance and detects the position of the slider or wheel. This requires continued periodic measurements of capacitance. When developing your application, make sure to periodically call R_TOUCH_ScanStart() and R_TOUCH_DataGet(). For more details, refer to the sample application.

CTSU2L has two types of majority judgement mode, JMM (Judgement Majority Mode) and VMM (Value Majority Mode). JMM gathers three measured values to one button from CTSU module, and determines touch state for each measured value, then creates final touch state by majority vote of three touch state. VMM creates touch state from summed value out of three measured values at CTSU module.

VMM is only available for sliders and wheels.

Please refer to the APN of CTSU module to see the details of JMM and VMM.

1.1.3 Button Touch Determination

Figure 1shows a diagram of the button touch determination. More details are described below.

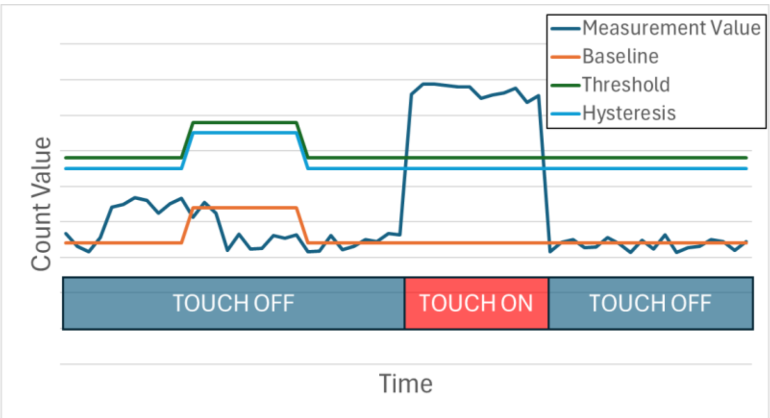

Figure 1 Button touch determination

(a) Baseline

The first measurement value after initial offset tuning is set as the baseline.

As a countermeasure for changes in the environment, the drift correction process refreshes the baseline. The drift correction process averages the measured values in the touch-off state over a certain period of time, and updates the baseline to the average value if the touch-off state is still in the touch-off state after a certain period of time. When the touch is turned on during the period, the average number of times and the average value up to that point are cleared. An example of operation is shown in Figure 2.

Set the period in the configuration settings (drift_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration. This allows you to adjust the ability to determine the touch state despite changes in the environment.

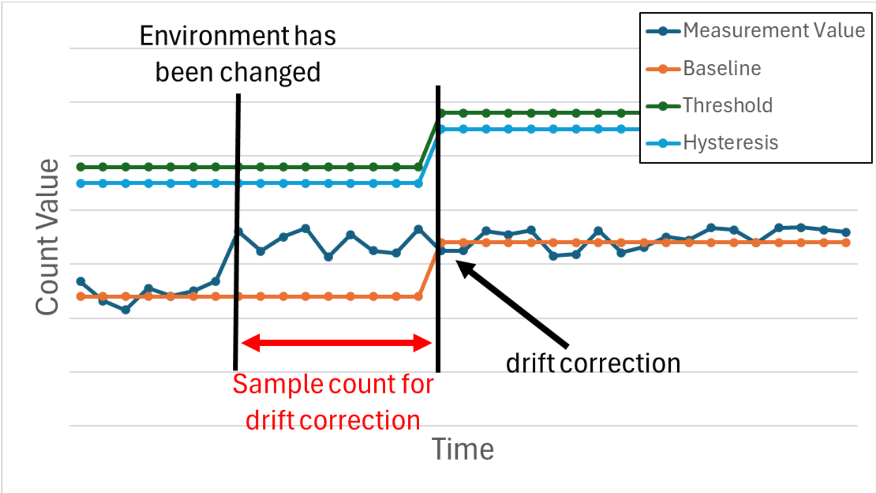

Figure 2 Button touch determination

(b) Touch Threshold

The touch threshold is set by adding an offset to the baseline value. Touch ON/OFF is determined depending on whether the measurement value exceeds the touch threshold. Self-capacitance buttons and mutual-capacitance buttons are processed in the same way, but with mutual capacitance buttons, the capacitance between the electrodes decreases when touched, so the touch threshold is set in the direction of the decreasing measurement value to determine ON/OFF. An example of operation is shown in Figure 3.

You can set the threshold for each button separately in the configuration settings (threshold in touch_button_cfg_t).

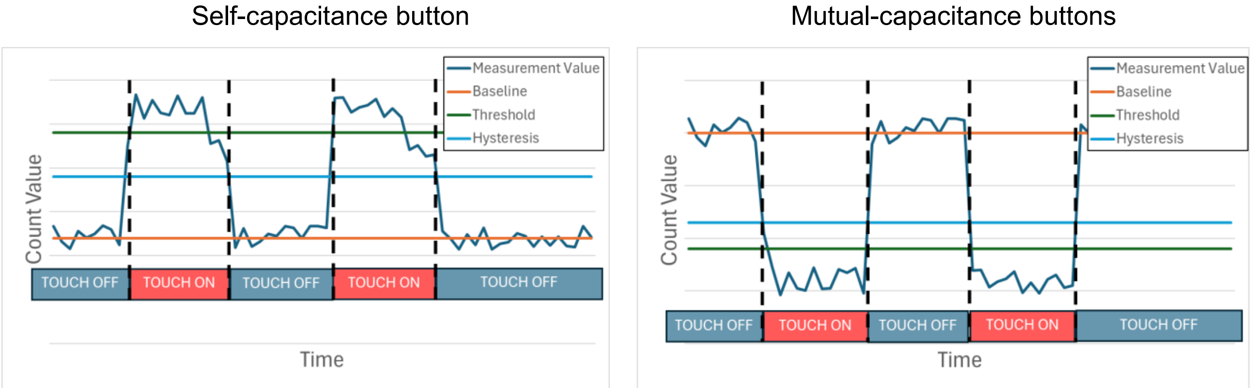

Figure 3 Touch threshold with Self-capacitance button and Mutual Capacitance button

Since touch detection requires preventing chattering and responding to environmental changes, this module also has the functions described below.

(c) Hysteresis

By offsetting the touch threshold value with a hysteresis value, chattering from touch ON to touch OFF can be prevented.

You can set the hysteresis value for each button in the configuration settings (hysteresis in touch_button_cfg_t). The larger the hysteresis value, the more effective it is at preventing chattering, but it also makes it more difficult to determine whether the touch is off.

(d) Debouncing count of touch-on filter/Debouncing count of touch-off filter

Touch ON is determined when the touch threshold is exceeded for a certain number of consecutive times. Touch OFF is determined when the touch threshold (hysteresis value offset) is not exceeded for a certain number of consecutive times. Figure 4 shows an example of operation.

In the configuration settings (on_freq and off_freq in touch_cfg_t) set the number of consecutive ON or OFF states. You can do this for all buttons in the touch interface configuration. Be aware that, although this is an effective solution to improving chattering, the greater the number of consecutive states, the slower the response to actual touch.

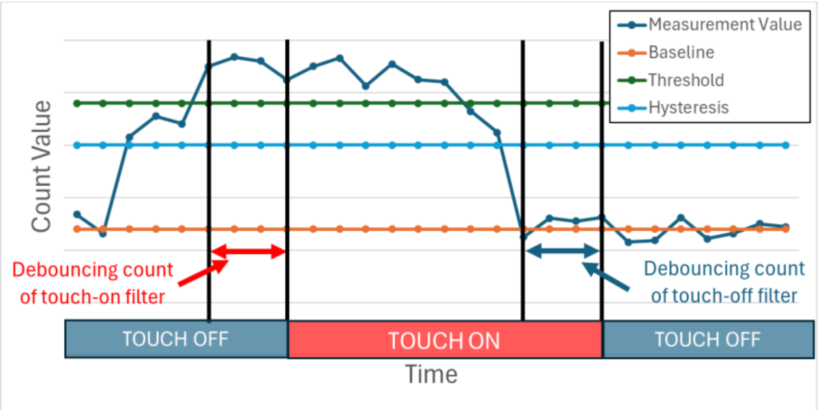

Figure 4 Chattering suppression process

(e) Chattering suppression type (Build option)

Select whether to use Type A or Type B for the touch judgment function (Hysteresis and Debouncing count of touch-on filter/Debouncing count of touch-off filter).

Type A: Within the hysteresis range, hold the touch ON counter.

Type B: Within the hysteresis range, the touch ON counter is reset.

In both cases, touch OFF judgment is not made within the hysteresis range.

Figure 5 shows an example of the operation of the chattering suppression type.

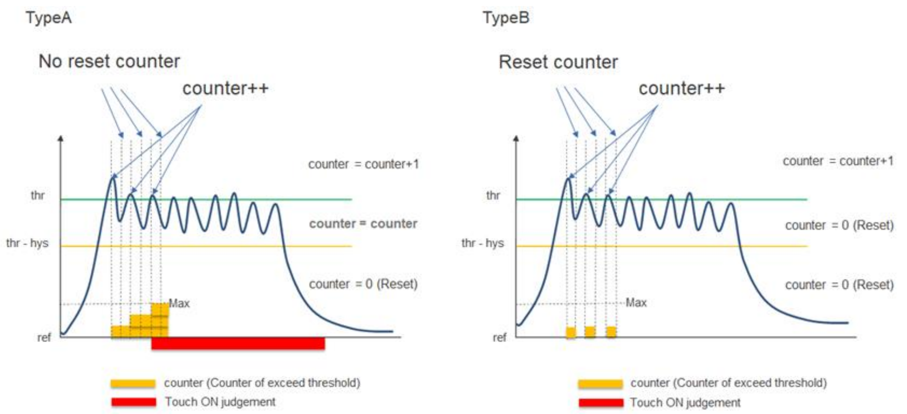

Figure 5 Chattering suppression type

(f) Continuous Touch Cancel

Strong noise or other sudden environment changes can disable the drift correction process, preventing return from the ON state. The Continuous Touch Cancel function implements the drift correction process and returns the button from the ON state by forcibly turning the state to OFF after a certain number of consecutive ON state periods.

Set the number of consecutive ON periods required for the Continuous Touch Cancel function to return the button to the OFF state in the configuration settings (cancel_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration.

1.1.4 Touch Position Detection of Slider/Wheel

Configure a slider with multiple pins to be measured (TS) physically arranged in a straight line. Configure a wheel with multiple pins physically arranged in a circle.

The touch position is calculated from the measured values of the TS in the configuration. The calculation method for sliders and wheels is fundamentally the same.

1. Detect the maximum value (TS_MAX) among the pins in the configuration.
2. Calculate the difference (d1, d2) between TS_MAX and the pins on either side. (If the TS_MAX pin is at one end of the slider, use the values of the two pins to the right or left, accordingly.)
3. If the total of d1 and d2 exceeds the slider threshold or wheel threshold, position calculation is initiated. If the total amount does not exceed the threshold, the position calculation process is ended.

With TS_MAX as the middle position, the ratio of d1 to d2 is used to calculate the position. The slider has a range of 1 to 100, and the while has a range of 1 to 360.

1.1.5 Tuning the Touch Determination Threshold

In cases where the amount of change in measurement value in response to touching varies due to changes in the environment, this function allows the user program to dynamically adjust the thresholds for judging the state of touch instead of requiring re-tuning through the QE tool.

This function is set with the API function RM_TOUCH_SensitivityRatioGet() and RM_TOUCH_ThresholdAdjust(). The pointer member of the structure shown in Table 1 is passed through the second argument of these API functions.

Table 1 touch_sensitivity_info_t Structure

Data Type	Member Name	Description
uint16_t *	p_touch_sensitivity_ratio	Pointer to the array holding the ratios of changes in response to touching
uint16_t	old_threshold_ratio	Old threshold ratio
uint16_t	new_threshold_ratio	New threshold ratio
uint8_t	new_hysteresis_ratio	New hysteresis ratio

- Obtaining the Ratio of Change in Response to Tuning

The measurement value is compared with the baseline. If the measurement value is smaller, the ratio of change in response to touching is set to 0. If the measurement value is larger, the ratio of change in response to touching is calculated and output. This function is set with the API function RM_TOUCH_SensitivityRatioGet().

Ratio of change in response to touching
= (measurement value – baseline) × new threshold ratio ÷ old threshold

- Adjusting the Threshold and Hysteresis by Changing the Ratios

The middleware receives the old threshold ratio, a new threshold ratio, and a new hysteresis ratio from the user program and changes the threshold and hysteresis. This function is set with the API function RM_TOUCH_ThresholdAdjust().The default threshold determined by QE tuning is 60% of the ratio of change by touching. To change the ratios based on this value, set old_threshold_ratio to 60.

New threshold = old threshold × new threshold ratio ÷ old threshold ratio

New hysteresis = new threshold × new hysteresis ratio

In addition, the middleware has a function for adjustment by using the ratio of change in response to touching. Use of the result obtained by the function described in “Obtaining the Ratio of Change in Response to Tuning” is recommended. When not using this function, set the ratio of change in response to touching to 100.

The middleware receives the ratio of change in response to touching from the user program and changes the threshold accordingly.

New threshold = desired threshold × current ratio of change by touching

The new threshold is used to adjust the hysteresis.

New hysteresis = new hysteresis ratio × new threshold

[Example 1]
When preventing errors in judgement due to EMC noise is to given priority over the touch sensitivity

The default threshold determined by QE tuning is 60% of the amount of change in response to touching. This value is adjusted to 70%.

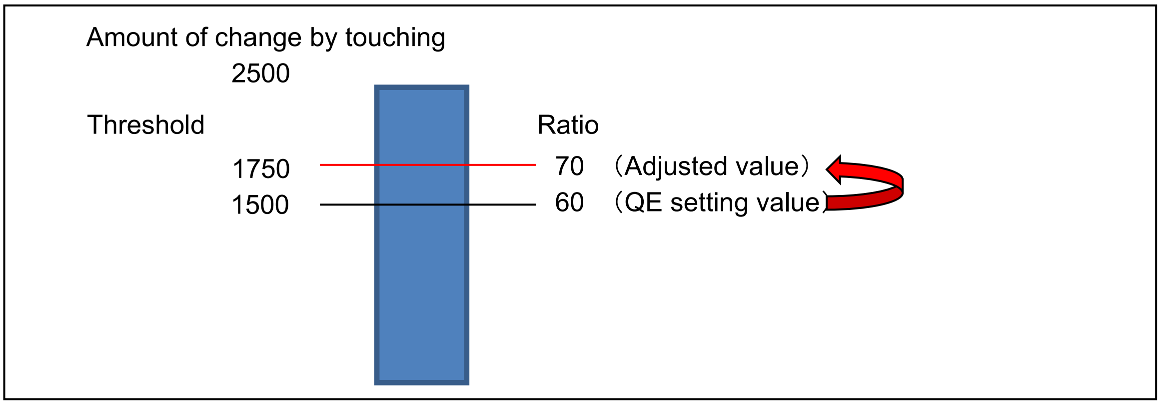

Figure 6 Example of Adjusting the Threshold by Changing the Ratios

In this case, set the members of the touch_sensitivity_info_t structure as follows.

*p_touch_sensitivity_ratio = 100, old_threshold_ratio = 60, new_threshold_ratio = 70 and new_hysteresis_ratio = 5.

[Example 2]
When the type of the overlay panel has been changed but re-tuning by the QE tool is not possible

When the overlay panel used is thicker than that used in tuning, the amount of change in response to touching becomes smaller. Therefore, if the software is used without re-tuning, judgement of the touched state may not be possible. In such cases, the threshold for judgement of the touched state is adjusted by using the ratio of the amount of change in response to touching after the overlay panel has been changed to the amount of change at the time of tuning.

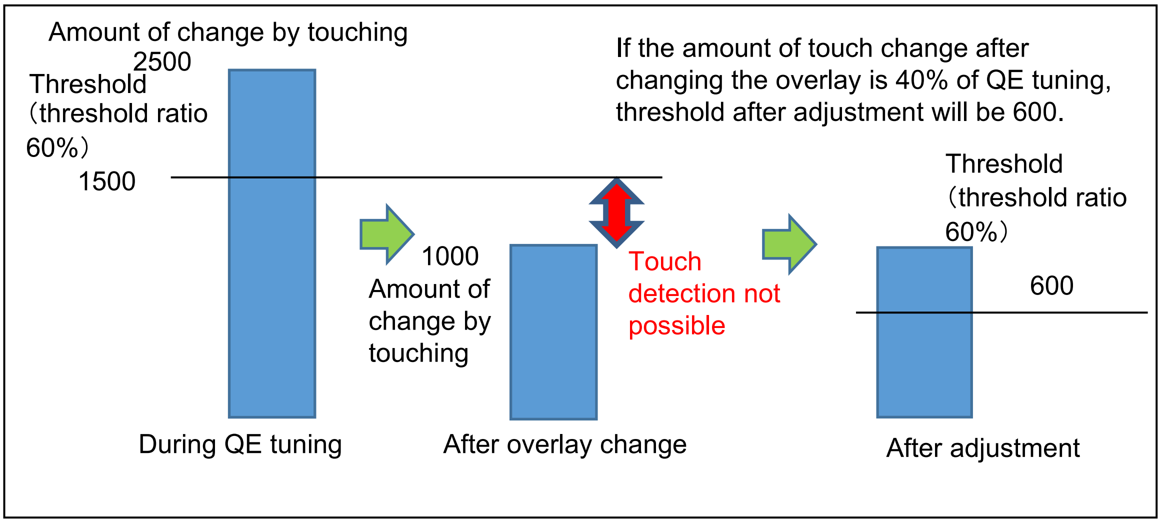

Figure 7 Example of Adjusting the Threshold according to the Touch Measurement Results

In this case, set the members of the touch_sensitivity_info_t structure as follows.

*p_touch_sensitivity_ratio = 40, old_threshold_ratio = 60, new_threshold_ratio = 60 and new_hysteresis_ratio = 5

[Practical Example]
Tuning application without either returning or rewriting the software

This is an example of the application for adjustment using data flash without re-tuning or software rewriting. Enable UART communication to PC and 'tuning mode'. In tuning mode, the MCU transmits the ratio of touch sensitivity in the touch state to the PC in real time. A user sends a command to decide the ratio while monitoring on the PC. The MCU stores the received ratio in the data flash. Make sure that the ratio stored in the data flash is read at the software activation, and the touch determination threshold is adjusted based on this stored value.

1.2 API Overview

This module has the following API functions.

The first argument of all API functions must be a pointer to a control structure. If you pass pointers for other arguments, make sure that they are not NULL and that you have reserved the required size for each API. However, RM_TOUCH_DataGet() and RM_TOUCH_CallBackSet() are exception, so please refer to the detailed description of API functions 3.3 and 3.4.

Function	Description
RM_TOUCH_Open()	Initializes the specified touch interface configuration.
RM_TOUCH_ScanStart()	Starts measurement of specified touch interface configuration.
RM_TOUCH_DataGet()	Gets measured values of specified touch interface configuration.
RM_TOUCH_CallbackSet()	Sets callback function (*1) of specified touch interface configuration.
RM_TOUCH_Close()	Closes specified touch interface configuration.
RM_TOUCH_ScanStop()	Stops measuring the specified touch interface configuration.
RM_TOUCH_SensitivityRatioGet()	Gets the ratio of the current touch sensitivity compare to change in response to tuning to specified touch interface configuration.
RM_TOUCH_ThresholdAdjust()	Adjust the ratio of touch determination threshold and the hysteresis value to specified touch interface configuration.
RM_TOUCH_DriftControl()	Changes drift correction settings.
RM_TOUCH_MonitorAddressGet()	Gets the address of the variable used for the QE monitor.

*1 : For detail about callback function, refer to the application note for "QE CTSU module Firmware Integration Technology" in the related documents.

2. API Information

Operations of this FIT module have been confirmed under the following conditions.

2.1 Hardware Requirements

The MCU used in the development must support one of the following functions:

- CTSU
- CTSUa
- CTSU2L
- CTSU2SL
- CTSU2SLa

2.2 Software Requirements

This driver depends on the following FIT modules:

- Board support package (r_bsp) v7.50 or newer
- QE CTSU FIT Module (r_ctsu_qe) v3.20
- SCI module (r_sci_rx) v5.30 or newer
- RSCI module (r_rsci_rx) v2.60 or newer

The driver also assumes the use of the following tool:

- QE for Capacitive Touch V4.2.0 (capacitive touch sensor development support tool) or newer

2.3 Supported Toolchains

This FIT module has been confirmed with the development environment and compiler shown below.

Development environment

- Renesas e^2 studio 2025-07
- IAR Embedded Workbench for Renesas RX 5.10.1

Compiler

- Renesas CC-RX Toolchain v.3.07.00
- GCC RX Toolchain v14.2.0.202505
- IAR C/C++ Compiler for Renesas RX version 5.10.1

2.4 Restrictions

The module code is non-reentrant and protects simultaneous calls for multiple functions.

2.5 Header File

All interface definitions to be called and used in the API are defined in ”rm_touch_qe.h”.

Select “rm_touch_qe_config.h” as the configuration option in each build.

2.6 Integer Type

This driver uses ANSI C99. The types are defined in stdint.h.

2.7 Compilation Settings

The following table provides the names and setting values for the configuration option settings used the TOUCH module.

rm_touch_config.h Configuration Options	
TOUCH_CFG_PARAM_CHECKING_ENABLE *Default value: "BSP_CFG_PARAM_CHECKING_ENABLE"	Selects whether to include the parameter check process in the code. Selecting "0" allows the user to omit the parameter check process from the code to shorten the code size. "0": Omit parameter check process from code. "1": Include parameter check process in code. "BSP_CFG_PARAM_CHECKING_ENABLE": Selection depends on BSP setting.
TOUCH_CFG_MONITOR_ENABLE This option is not available for rm_touch_config.h. The option is defined in the qe_touch_define.h output by the QE; the default value is "1".	Select "1" to enable data generation for the QE monitor.
TOUCH_CFG_UART_MONITOR_SUPPORT *Default value: "0"	This option is used when TOUCH_CFG_MONITOR_ENABLE is enabled. Set to "1" to enable QE and serial communications. Note: When using the UART module, generate this option with the Smart Configurator.
TOUCH_CFG_UART_TUNING_SUPPORT	Set the use of UART tuning. 0: Disable, 1: Enable
TOUCH_CFG_UART_MODULE_TYPE *Default value: "0"	Select FIT module wich control UART. Set "0", UART use of SCI module (r_sci_rx). Set "1", UART use of RSCI module (r_rsci_rx).
TOUCH_CFG_UART_NUMBER	Set the UART channel number.
TOUCH_CFG_UART_BAUDRATE	Set the UART Baudrate.
TOUCH_CFG_UART_PRIORITY	Set the UART interrupt priority.
TOUCH_CFG_CHATTERING_SUPPRESSION_TYPE *Default value: "0"	Set the chattering suppression type. Set "0", it is set to TypeA. The counter of the number of times the threshold is exceeded is held within the hysteresis range. Set "1", it is set to TypeB. Resets the counter of the number of times the threshold is exceeded within the hysteresis range.
The following configurations depend on the touch interface configuration and cannot be set using Smart Configurator. These configurations are set when using QE. In this case, QE_TOUCH_CONFIGURATION is defined in the project. Although rm_touch_qe_config.h is invalid, qe_touch_define.h is defined instead.	
CTSU_CFG_NUM_BUTTONS	Sets the total number of buttons.
CTSU_CFG_NUM_SLIDERS	Sets the total number of slides.
CTSU_CFG_NUM_WHEELS	Sets the total number of wheels.

2.8 Code Size

ROM (code and constants) and RAM (global data) size are determined according to the configuration options as described in "section 2.7 Compilation Setting" during a build. The values shown are baselines when the compile option is the default for C compiler listed in "section 2.3 Supported Toolchains". The default of compile options is as follows: the optimization level is 2, the optimization type is size priority, and the data-endian is a little endian. The code size varies according to the C compile version and the compile options.

Using Renesas CC-RX Toolchain v3.06.00, the following is the size at compilation settings.
Only settings related to size are shown.

- TOUCH_CFG_PARAM_CHECKING_ENABLE 0
- TOUCH_CFG_MONITOR_ENABLE 0
- TOUCH_CFG_UART_MONITOR_SUPPORT 0
- TOUCH_CFG_UART_TUNING_SUPPORT 0

The self-capacitance and mutual capacitance are shown as the size with one button, the size that increases with the addition of one button, and the size that increases with the addition of a slider and wheel. It also includes qe_touch_config.c output by QE.

[CTSU1]

- CTSU_CFG_NUM_SUMULTI 1

	Self-capacitance button 1	+Self-capacitance button	+Wheel	+Slider	Mutual-capacitance button 1	+Mutual-capacitance button
ROM	1252 bytes	5 bytes	+382 bytes	+434 bytes	1630 bytes	+6 bytes
RAM	154 bytes	24 bytes	+13 bytes	+15 bytes	174 bytes	+26 bytes

[CTSU2 VMM]

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 1

	Self-capacitance button 1	+Self-capacitance button	+Wheel	+Slider	Mutual-capacitance button 1	+Mutual-capacitance button
ROM	1252 bytes	+13 bytes	+382 bytes	+434 bytes	1665 bytes	+6 bytes
RAM	154 bytes	+65 bytes	+13 bytes	+15 bytes	190 bytes	+26 bytes

[CTSU2 JMM]

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 2

	Self-capacitance button 1	+Self-capacitance button	+Wheel	+Slider	Mutual-capacitance button 1	+Mutual-capacitance button
ROM	1648 bytes	+17 bytes	—	—	2022 bytes	+19 bytes
RAM	202 bytes	+64 byte	—	—	250 bytes	+70 bytes

2.9 Arguments

The following is the structures and enums used as arguments of the API functions. Many of the parameters used in the API functions are defined by the enums, which provides a way to check types and reduce errors.

These structures and enums are defined in rm_touch_qe.h, rm_touch_qe_api.h along with the prototype declaration.

The control structure for the touch interface configuration are shown in Table 2 touch_ctrl_t Structure. Please refer rm_touch_qe.h to see data type used in this structure. Using QE for Capacitive Touch allows the variables corresponding to the touch interface configuration to be output by qe_touch_config.c. Make sure to set qe_touch_config.c in the module's first API argument.

Table 2 touch_ctrl_t Structure

Data Type	Member	Description
uint32_t	open	Open flag
touch_button_info_t	binfo	Button information
touch_slider_info_t	sinfo	Slider information
touch_wheel_info_t	winfo	Wheel information
bool	serial_tuning_enable	Flag for enabling serial tuning
touch_cfg_t const *	p_touch_cfg	Pointer to the configuration structure
ctsu_instance_t const *	p_ctsu_instance	Pointer to the CTSU control structure
touch_mm_info_t	p_touch_mm_info	Pointer to the structure for majority judgment

The touch_button_info_t structure is shown below. It manages the results of touch judgement and the data required for touch judgement for each button.

Table 3 touch_button_info_t Structure

Data Type	Member	Description
uint64_t	status	Results of touch judgement for the button
uint16_t *	p_threshold	Pointer to the threshold buffer
uint16_t *	p_hysteresis	Pointer to the hysteresis buffer
uint16_t *	p_reference	Pointer to the baseline buffer
uint16_t *	p_on_count	Count of times the touch threshold was exceeded
uint16_t *	p_off_count	Count of times the touch threshold was not exceeded
uint32_t *	p_drift_buf	Pointer to the drift buffer
uint16_t *	p_drift_count	Pointer to the drift count buffer
uint8_t	on_freq	Number of Debouncing count of touch-on filtering
uint8_t	off_freq	Number of Debouncing count of touch-off filtering
uint16_t	drift_freq	Sample count for drift correction
uint16_t	cancel_freq	Continuous Touch Cancel Count

The touch_slider_info_t structure is shown below. It manages the results of position detection and threshold for each slider.

Table 4 touch_slider_info_t Structure

Data Type	Member	Description
uint16_t *	p_position	Pointer to the position result buffer
uint16_t *	p_threshold	Pointer to the threshold buffer

The touch_wheel_info_t structure is shown below. It manages the results of position detection and threshold for each wheel.

Table 5 touch_wheel_info_t Structure

Data Type	Member	Description
uint16_t *	p_position	Pointer to the position result buffer
uint16_t *	p_threshold	Pointer to the threshold buffer

Table 6 shows touch_cfg_t structure (configuration structure).
Using QE for Capacitive Touch allows the variables corresponding to the touch interface configuration to be output by qe_touch_config.c and set this structure to the second argument of RM_TOUCH_Open() function. The value of this structure is assumed to be set by Smart Configurator or QE for Capacitive Touch and no error check is conducted due to streamline the process. Be careful when setting the value of this structure manually.

Table 6 touch_cfg_t Structure

Data Type	Member Name	Description	Range of the Value
touch_button_cfg_t *	p_buttons	Pointer to a button configuration	—
touch_slider_cfg_t *	p_sliders	Pointer to a slider configuration	—
touch_wheel_cfg_t *	p_wheels	Pointer to a wheel configuration	—
touch_pad_cfg_t *	p_pad	Pointer to a pad configuration	—
uint8_t	num_buttons	Number of buttons	0 to 64
uint8_t	num_sliders	Number of sliders	0 or 7
uint8_t	num_wheels	Number of wheels	0 or 7
uint8_t	on_freq	Accumulated number of touch-ON judgements	0 to 255 (0 disable Debouncing count of touch-on filter)
uint8_t	off_freq	Accumulated number of touch-OFF judgements	0 to 255 (0 disable Debouncing count of touch-off filter)
uint16_t	drift_freq	Number of cycles for drift correction of the baseline	0 to 65535 (0 disables drift correction.)
uint16_t	cancel_freq	Maximum number of consecutive touch-ON judgements	0 to 65535 (0 disables long-press cancellation)
uint8_t	number	Configuration number for QE monitoring	0 to 255
ctsu_instance_t *	p_ctsu_instance	CTSU instance pointer	—
void *	p_context	Context pointer	—
void *	p_extend	Extended configuration pointer	—

Table 7 touch_button_cfg_t Structure

Data Type	Member Name	Description	Range of the Value
uint8_t	elem_index	Index of a button element	0 to 63
uint16_t	threshold	Touch threshold	1 to 65535
uint16_t	hysteresis	Hysteresis value for debouncing	0 to 65534

Table 8 touch_slider_cfg_t Structure

Data Type	Member Name	Description	Range of the Value
uint8_t *	p_elem_index	Pointer to index of a slider element	-
uint8_t	num_elements	Number of elements used in the slider	1 to 10
uint16_t	threshold	Threshold for position calculation	1 to 65535

Table 9 touch_wheel_cfg_t Structure

Data Type	Member Name	Description	Range of the Value
uint8_t *	p_elem_index	Pointer to index of a wheel element	-
uint8_t	num_elements	Number of elements used in the wheel	1 to 8
uint16_t	threshold	Threshold for position calculation	1 to 65535

2.10 Return Values

The following provides return values for the API functions. The enum is defined in fsp_common_api.h.

```
/** Common error codes */
typedef enum e_fsp_err
{
    FSP_SUCCESS = 0,

    FSP_ERR_ASSERTION           = 1,                  ///< A critical assertion has failed
    FSP_ERR_INVALID_POINTER      = 2,                  ///< Pointer points to invalid memory location
    FSP_ERR_INVALID_ARGUMENT     = 3,                  ///< Invalid input parameter
    FSP_ERR_NOT_OPEN            = 7,                  ///< Requested channel is not configured or API not open
    FSP_ERR_ALREADY_OPEN         = 14,                 ///< Requested channel is already open in a different
configuration
    FSP_ERR_INVALID_HW_CONDITION  = 27,                 ///< Detected hardware is in invalid condition


    /* Start of CTSU Driver specific */
    FSP_ERR_CTSU_SCANNING           = 6000,      ///< Scanning.
    FSP_ERR_CTSU_NOT_GET_DATA        = 6001,      ///< Not processed previous scan data.
    FSP_ERR_CTSU_INCOMPLETE_TUNING    = 6002,      ///< Incomplete initial offset tuning.
    FSP_ERR_CTSU_DIAG_NOT_YET        = 6003,      ///< Diagnosis of data collected no yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,      ///< Diagnosis of LDO over voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH       = 6005,      ///< Diagnosis of CCO into 19.2uA failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW        = 6006,      ///< Diagnosis of CCO into 2.4uA failed.
    FSP_ERR_CTSU_DIAG_SSCG           = 6007,      ///< Diagnosis of SSCG frequency failed.
    FSP_ERR_CTSU_DIAG_DAC            = 6008,      ///< Diagnosis of non-touch count value failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE   = 6009,      ///< Diagnosis of LDO output voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE    = 6010,      ///< Diagnosis of over voltage detection circuit failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT    = 6011,      ///< Diagnosis of over current detection circuit failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE  = 6012,      ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE   = 6013,      ///< Diagnosis of Current source value failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN    = 6014,      ///< Diagnosis of SENSCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN      = 6015,      ///< Diagnosis of SUCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY   = 6016,      ///< Diagnosis of SUCLK clock recovery function failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN       = 6017,      ///< Diagnosis of CFC oscillator gain failed.


} fsp_err_t;
```

2.11 Adding the FIT Module to Your Project

2.11.1 Adding source tree and project include paths

This module must be added to each project in which it is used. Renesas recommends using “Smart Configurator” described in (1) or (3). However, “Smart Configurator” only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using “Smart Configurator” in e2 studio
By using the “Smart Configurator” in e2 studio, the FIT module is automatically added to your project. Refer to “Renesas e2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using “FIT Configurator” in e2 studio
By using the “FIT Configurator” in e2 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 “Smart Configurator” on CS+
By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “Renesas e2 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.

2.11.2 Setting driver options when not using Smart Configurator

The Touch-specific options are found and edited in r_config¥r_touch_qe_config.h.

2.12 API compatibility mode

The API function has been completely revised in Rev.2.00.
An API compatibility mode is provided to use the Rev.1.11 API.
Please output the code of API compatibility mode when outputting the code in QE. Check the code output option "Use API compatibility mode" in QE and click the "File output" button to output the code for API compatibility mode.
Please refer to QE Touch Module Firmware Integration Technology Rev.1.11 (R01AN4470JU0111) for the API of Rev.1.11.
API compatibility mode does not support the error detection of QE_ERR_OT_WINDOW_SIZE that was detected by R_TOUCH_Open ().

3. API Functions

3.1 RM_TOUCH_Open

This function initializes the module and must be executed before using any of the other API functions. Please execute this function for each touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_Open (touch_ctrl_t * const p_ctrl,
                         touch_cfg_t const * const p_cfg)
```

Parameters

p_ctrl	[in] Pointer to the control structure
p_cfg	[in] Pointer to the config structure

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_ALREADY_OPEN	*/* Open() is called without calling Close() */*
FSP_ERR_INVALID_ARGUMENT	*/* Configuration parameters are invalid */*

Properties

Prototype is declared in rm_touch_api.h .

Description

This function enables control structure initialization, calls R_CTSU_Open(), and initializes the CTSU module according to the argument p_cfg.
By setting TOUCH_CFG_MONITOR_ENABLE, the monitor buffer is initialized. By setting TOUCH_CFG_UART_MONITOR_SUPPORT, the UART monitor and UART module are initialized.

Example

```
fsp_err_t err;

/* Initialize pins (function created by Smart Configurator) */
R_CTSU_PinSetInit();

/* Initialize the API. */
err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);

/* Check for errors. */
if (err != FSP_SUCCESS)
{
    . . .
}
```

Special Notes:

The port must be initialized before calling this function. We recommend using the R_CTSU_PinSetInit() function generated by SmartConfigurator as the port initialization function.
This function calls the CTSU module's R_CTSU_Open(). Please refer R_CTSU_Open() for more details.

3.2 RM_TOUCH_ScanStart

This function starts measurement of the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_ScanStart (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl [in] Pointer to the control structure

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*
FSP_ERR_CTSU_SCANNING	*/* Now scanning */*
FSP_ERR_CTSU_NOT_GET_DATA	*/* Did not obtain previous results */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function calls R_CTSU_ScanStart() and starts the measurement.

Example

```
fsp_err_t err;

/* Initiate a sensor scan by software trigger */
err = RM_TOUCH_ScanStart(&g_touch_ctrl);

/* Check for errors. */
if (err != FSP_SUCCESS)
{
    . . .
}
```

Special Notes:

This function calls the CTSU module's R_CTSU_ScanStart(). Refer the R_CTSU_ScanStart() document for more details.

3.3 RM_TOUCH_DataGet

This function reads the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_DataGet (touch_ctrl_t * const p_ctrl,
                           uint64_t        * p_button_status,
                           uint16_t        * p_slider_position,
                           uint16_t        * p_wheel_position)
```

Parameters

p_ctrl	[in] Pointer to the control structure
p_button_status	[out] Pointer to the buffer that stores button state.
p_slider_position	[out] Pointer to the buffer that stores slider position.
p_wheel_position	[out] Pointer to the buffer that stores wheel position.

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*
FSP_ERR_CTSU_SCANNING	*/* Now scanning */*
FSP_ERR_CTSU_NOT_GET_DATA	*/* Did not obtain previous results */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/* Tuning initial offset */*
FSP_ERR_INVALID_HW_CONDITION	*/ * Values scanned by CTSU show abnormal values */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function calls R_CTSU_DataGet() and reads all measured values from the previous measurement to determine the touch/non-touch state or position. By setting TOUCH_CFG_MONITOR_ENABLE, data is stored in the monitor buffer. By setting TOUCH_CFG_UART_MONITOR_SUPPORT, the data in the monitor buffer is sent to the UART module.

The buffer that stores button state holds the button status in each bit and the bits are assigned tin decreasing order of the number of TS pins assigned to the buttons in the specified touch interface configuration.

If no buttons are assigned to touch interface configuration, the second argument, p_button_status can be set as NULL.

The buffer that stores the slider position contains a value of 1~100 for touch and 65535 for non-touch. If no sliders are assigned to touch interface configuration, the third argument, p_slider_position can be set as NULL.

The buffer that stores the wheel position contains a value of 1~360 for touch and 65535 for non-touch. If no wheels are assigned to touch interface configuration, the fourth argument, p_wheel_position can be set as NULL.

Example:

```
  fsp_err_t err;
  uint64_t button_status;
  uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
  uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];

  /* Get all sensor values */
  err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, slider_position,
wheel_position);
```

Special Notes:
This function calls the CTSU module's R_CTSU_DataGet(). Refer the R_CTSU_DataGet() document for more details.

3.4 RM_TOUCH_CallbackSet

This function sets the function specified for the measurement completion callback function.

Format
```
fsp_err_t RM_TOUCH_CallbackSet (touch_ctrl_t * const p_api_ctrl,
                                void (* p_callback)(touch_callback_args_t *),
                                void const * const p_context,
                                touch_callback_args_t * const p_callback_memory)
```

Parameters

p_api_ctrl	[in] Pointer to the control structure
p_callback	[in] Pointer to callback function
p_context	[in] Pointer to send to callback function
p_callback_memory	[in] Set to NULL

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function calls R_CTSU_CallbackSet() and sets the callback function.

Example:
```
fsp_err_t err;

/* Set callback function */
err = RM_TOUCH_CallbackSet(&g_ctsu_ctrl, ctsu_callback, NULL, NULL);
```

Special Notes:

This function calls the CTSU module's R_CTSU_CallbackSet(). Refer the R_CTSU_CallbackSet() document for more details.

3.5 RM_TOUCH_Close

This function closes the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_Close (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl [in] Pointer to the control structure

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function closes the specified touch interface configuration.

Example:

```
fsp_err_t err;

/* Shut down peripheral and close driver */
err = RM_TOUCH_Close(&g_touch_ctrl);
```

Special Notes:

This function calls the CTSU module's R_CTSU_Close(). Refer the R_CTSU_Close() document for more details

3.6 RM_TOUCH_ScanStop

This function stops measuring the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_ScanStop (touch_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl	[in] Pointer to the control structure

Return Values

FSP_SUCCESS	*/* Successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function stops measuring the specified touch interface configuration.

Example:

```
fsp_err_t err;

/* Stop CTSU module */
err = RM_TOUCH_ScanStop(&g_touch_ctrl);
```

Special Notes:

This function calls the CTSU module's R_CTSU_ScanStop(). Refer the R_CTSU_ScanStop() document for more details

3.7 RM_TOUCH_SensitivityRatioGet

This function returns the ratio of the current touch sensitivity to the touch sensitivity during tuning for the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_SensitivityRatioGet (touch_ctrl_t * const p_ctrl,
                       touch_sensitivity_info_t * p_touch_sensitivity_info);
```

Parameters

p_ctrl [in] Pointer to the control structure

p_touch_sensitivity_info [in/out] Pointer to the variable storing table information of touch sensitivity ratio calculation

Return Values

FSP_SUCCESS / Successfully got the ratio of touch sensitivity */*

FSP_ERR_ASSERTION / Argument pointer not specified */*

FSP_ERR_NOT_OPEN / Called without calling Open() */*

FSP_ERR_CTSU_SCANNING / Now scanning */*

FSP_ERR_CTSU_INCOMPLETE_TUNING / Tuning initial offset */*

Properties

Prototyped in file rm_touch_api.h.

Description

Estimate change of touch by threshold and threshold ratio and returns the ratio of change in response to tuning by assuming the change of touch is 100%.
Create the buffer for the number of button elements because this function outputs the ratio of change in current measurement to all selected touch interfaces, so set the start address of created buffer to p_touch_sensitibity_info->p_touch_sensitivity_ratio.
Also, call this function after obtain the measurement value of touch-on state by RM_TOUCH_DataGet().

Example:

```
  qe_err_t err;
  touch_sensitivity_info_t touch_sensitivity_table[QE_NUM_METHODS];
  uint16_t touch_sensitivity_first[CONFIG01_NUM_BUTTONS ] = { 100 };

  touch_sensitivity_table[QE_METHOD_CONFIG01].p_touch_sensitivity_ratio =
touch_sensitivity_first;
  touch_sensitivity_table[QE_METHOD_CONFIG01].old_threshold_ratio = 60;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_threshold_ratio = 70;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_hysteresis_ratio = 5;

  err = RM_TOUCH_SensitivityRatioGet(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);
```

Special Notes:

None.

3.8 RM_TOUCH_ThresholdAdjust

This function adjusts the thresholds for judging the state of touch and hysteresis for the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_ThresholdAdjust  (touch_ctrl_t * const p_ctrl,
                              touch_sensitivity_info_t * p_touch_sensitivity_info);
```

Parameters

p_ctrl [in] Pointer to the control structure

p_touch_sensitivity_info [in] Pointer to the variable storing table information of touch sensitivity ratio calculation

Return Values

FSP_SUCCESS */* Successfully changed touch determination threshold. */*

FSP_ERR_ASSERTION */* Argument pointer not specified */*

FSP_ERR_NOT_OPEN */* Called without calling Open() */*

Properties

Prototyped in file rm_touch_api.h.

Description

This function adjusts the threshold and hysteresis as ratio of change in ratio of touching(each button), new threshold ratio, old threshold ratio and new hysteresis ratio are inputted.

Example:

```
  qe_err_t err;
  touch_sensitivity_info_t touch_sensitivity_table[QE_NUM_METHODS];
  uint16_t touch_sensitivity_first[CONFIG01_NUM_BUTTONS ] = { 100 };

  touch_sensitivity_table[QE_METHOD_CONFIG01].p_touch_sensitivity_ratio =
touch_sensitivity_first;
  touch_sensitivity_table[QE_METHOD_CONFIG01].old_threshold_ratio = 60;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_threshold_ratio = 70;
  touch_sensitivity_table[QE_METHOD_CONFIG01].new_hysteresis_ratio = 5;

  err = RM_TOUCH_SensitivityRatioGet(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);

  err = RM_TOUCH_ThresholdAdjust(g_qe_touch_instance_config01.p_ctrl,
&touch_sensitivity_table[QE_METHOD_CONFIG01]);
```

Special Notes:

None.

3.9 RM_TOUCH_DriftControl

This function changes the settings of drift correction for the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_DriftControl(touch_ctrl_t * const p_ctrl,
                               uint16_t input_drift_freq);
```

Parameters

p_ctrl	[in] Pointer to the control structure
input_drift_freq	[in] Enables / disables interval of drift correction

Return Values

FSP_SUCCESS	*/* Successfully changed drift correction */*
FSP_ERR_ASSERTION	*/* Missing required argument pointer */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*

Properties

Prototyped in file rm_touch_api.h.

Description

Set the drift correction to the number of times set in input_drift_freq. When a value other than 0 is set, the drift correction function is enabled. If set to 0, the drift correction function is disabled.
As an example of using this API, when calculating the ratio of the touch change amount using RM_TOUCH_SensitivityRatioGet (), the touch change amount decreases due to the thick overlay, and the threshold value is not exceeded even if touched. Prevents the baseline from drifting.

Example:

```
qe_err_t  err;

err = RM_TOUCH_DriftControl(g_qe_touch_instance_config01.p_ctrl, 0);
```

3.10 RM_TOUCH_MonitorAddressGet

This function gets the address of the variable used by the QE monitor for the specified touch interface configuration.

Format

```
fsp_err_t RM_TOUCH_MonitorAddressGet (touch_ctrl_t * const p_ctrl,
                                      uint8_t ** pp_monitor_buf,
                                      uint8_t ** pp_monitor_id,
                                      uint16_t ** pp_monitor_size)
```

Parameters

p_ctrl	[in] Pointer to the control structure
pp_monitor_buf	[out] Pointer to the variable storing starting address of monitor buffer
pp_monitor_id	[out] Pointer to the variable storing address of monitor ID variables
pp_monitor_size	[out] Pointer to the variable storing starting address of monitor size

Return Values

FSP_SUCCESS	*/* Successfully QE monitor variable address was got */*
FSP_ERR_ASSERTION	*/* Null pointer passed as a parameter */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*
FSP_ERR_NOT_ENABLED	*/* Requested operation is not enabled */*

Properties

Prototyped in file rm_touch_api.h.

Description

Use this feature for the QE monitor when you have both automatic judgement and software judgement touch interface configurations. Get the start address of the monitor buffer with the second argument, the address of the monitor ID variable with the third argument, and the start address of the monitor size with the fourth argument.

Example:

```
qe_err_t  err;
uint8_t * gp_monitor_buf;
uint8_t * gp_monitor_id;
uint16_t * gp_monitor_size;

err =  RM_TOUCH_MonitorAddressGet(g_qe_touch_instance_config01.p_ctrl,
                                  &gp_monitor_buf,
                                  &gp_monitor_id,
                                  &gp_monitor_size);
```

Special Notes:

This function is for QE monitoring features.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.04.18	—	First edition issued
1.10	Jul.09.19	1,39 4-6 15, 18, 22, 30, 31 24, 26 * 14, 16-20 1,20 15,22	Added RX23W support Added definitions for “correction” and “offset tuning”. Updated API return values Added TOUCH_CMD_GET_FAILED_SENSOR and TOUCH_CMD_GET_LAST_SCAN_METHOD Control() commands Moved offset tuning processing into R_TOUCH_Open(). Added #pragma section macros and configuration option to driver for Safety Module support (includes GCC/IAR support). Added IEC 6730 Compliance section. Added error code QE_ERR_UNSUPPORTED_CLK_CFG.
1.11	Jan.09.20	30-31 29,32 5,6,22,35 — —	Updated example code. Added TOUCH_CMD_CLEAR_TOUCH_STATES for low power applications. Added API function R_TOUCH_GetBtnBaselines(). Fixed bug (CTSU) where a custom callback function was called twice after a scan completes. Fixed compile error (CTSU) for RX231 when PLL had multiplier of 13.5.
2.00	Jul.30.21	-	Full-fledged revision
2.01	Dec.17.21	7	Fixed 1.2 API Overviews
		10	Fixed 2.8 Code Size
2.10	Apr.20.22	7	Fixed 1.2 API Overviews
		29,30	Added 3.10 RM_TOUCH_MonitorAddressGet
2.20	Dec.28.22	4	Added to 1.13 Button touch detection
		9	Updated 2.2 Software requirements
		9	Updated 2.3 Supported toolchains
		11	Fixed 2.8 Code size
		13	Updated 2.10 return value
		18	Updated 3.3 R_TOUCH_DataGet
3.00	Oct.15.24	1	Added RX260, RX261 support
		3	Updated 1.1.2 Measurements and Data Processing
		6	Updated 1.1.5 Tuning the Touch Determination Threshold
		8	Updated 2.1 Hardware requirements
		8	Updated 2.2 Software requirements
		8	Updated 2.3 Supported toolchains
		9	Updated 2.7 Compilation Settings
		10	Updated 2.8 Code Size
		13	Updated 2.9 Arguments
		23	Updated Description of 3.7 RM_TOUCH_SensitivityRatioGet
		24	Updated Description of 3.8 RM_TOUCH_ThresholdAdjust
		26	Updated Description of 3.10 RM_TOUCH_MonitorAddressGet
3.10	Feb.19.25	3	Updated 1.1.3 Button Touch Determination
		10	Updated 2.2 Software requirements
		10	Updated 2.3 Supported toolchains
		12	Updated 2.8 Code Size
		19	Updated Description of 3.3 RM_TOUCH_DataGet

3.11	Mar.21.25	-	Updated with changes to disclaimer comment in code file. No changes to the content of this APN.
3.20	Jul.31.25	9	Added description to 1.2 API Overview
		10	Updated 2.2 Software requirements
		10	Updated 2.3 Supported toolchains
		12	Updated 2.8 Code Size
		17-26	Added input and output information to API function arguments.

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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

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/.