RL78 Family

TOUCH Module Software Integration System

Introduction

This application note describes the TOUCH Module.

Target Device

RL78/G23 Group

RL78/G22 Group

RL78/G16 Group

Related Documents

RL78 Family CTSU Module Software Integration System (R11AN0484)

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
1.1.6 Automatic judgment measurement using SMS 9
1.2 API Overview 10

2. API Information 11
2.1 Hardware Requirements 11
2.2 Software Requirements 11
2.3 Supported Toolchains 11
2.4 Restrictions 11
2.5 Header File 12
2.6 Integer Type 12
2.7 Compilation Settings 13
2.8 Code Size 14
2.9 Arguments 15
2.10 Return Values 17

3. API Functions 18
3.1 RM_TOUCH_Open 18
3.2 RM_TOUCH_ScanStart 19
3.3 RM_TOUCH_DataGet 20
3.4 RM_TOUCH_CallbackSet 21
3.5 RM_TOUCH_SmsSet 22
3.6 RM_TOUCH_Close 25
3.7 RM_TOUCH_ScanStop 26
3.8 RM_TOUCH_SensitivityRatioGet 27
3.9 RM_TOUCH_ThresholdAdjust 28
3.10 RM_TOUCH_DriftControl 29
3.11 RM_TOUCH_MonitorAddressGet 30

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 UART communications, determines the type of the information from QE and sends only the necessary information.

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

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 is described below.

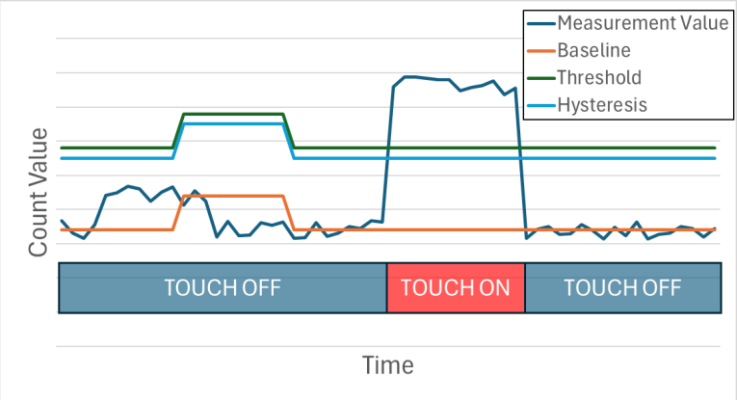


Figure 1 Button touch determination

(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.

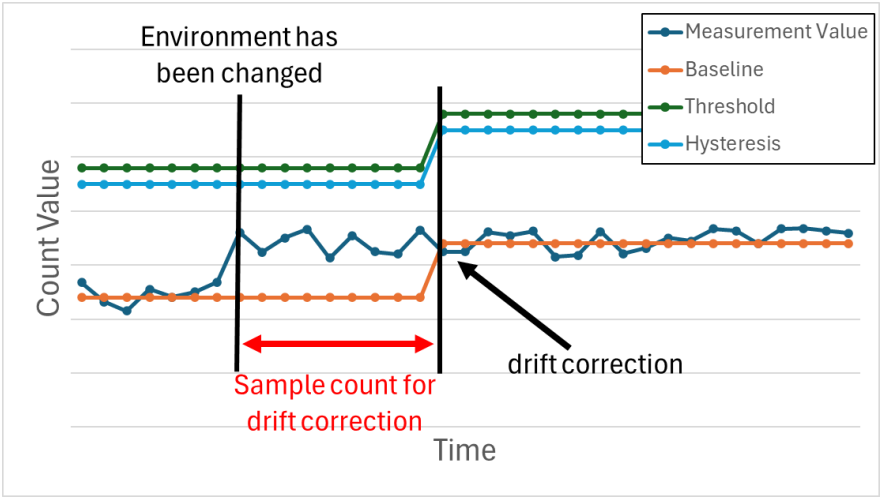


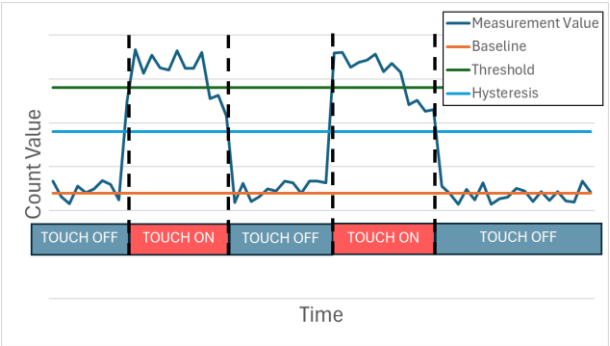
Figure 2 Button touch determination

(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).

Self-capacitance button



Mutual-capacitance buttons

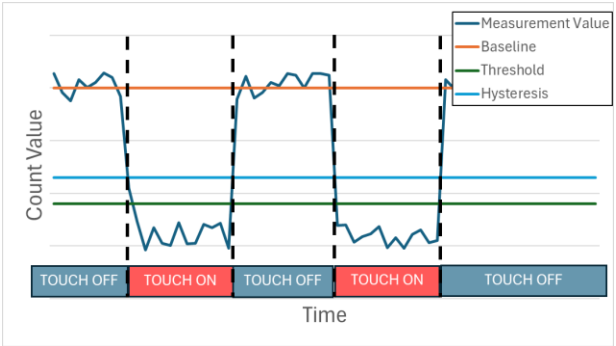


Figure 3 Touch threshold with Self-capacitance button and Mutual Capacitance button

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.

(b) 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.

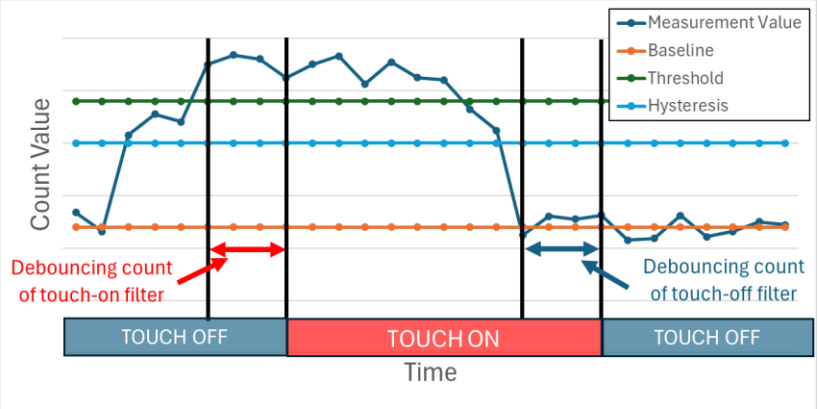


Figure 4 Chattering suppression process

(d) 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.

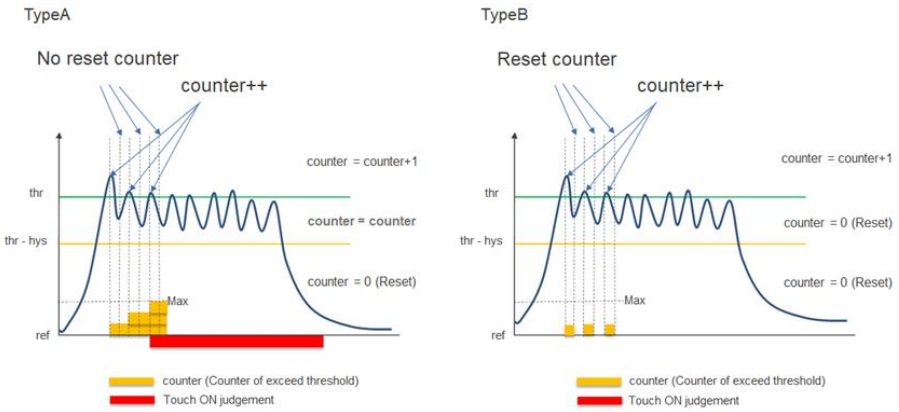


Figure 5 Chattering suppression type

(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 threshold, position calculation is initiated. If the total amount does not exceed the threshold, the position calculation process is ended.
4. 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 feature provides two API functions, RM_TOUCH_SensitivityRatioGet() and RM_TOUCH_ThresholdAdjust(). Set the touch_sensitivity_info_t structure to second arguments.

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. The RM_TOUCH_SensitivityRatioGet() API function is provided for this purpose.

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. The RM_TOUCH_ThresholdAdjust() API function is provided for this purpose. 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%.

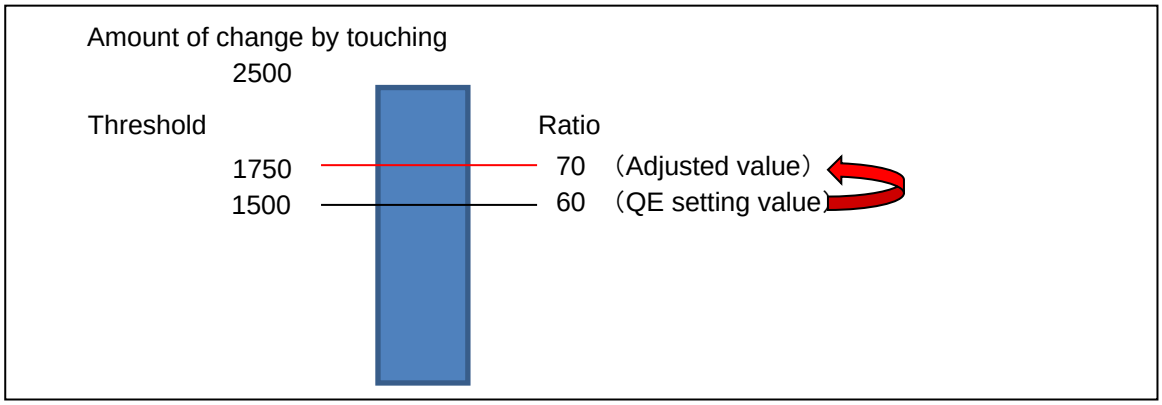


Figure 7 Example of Adjusting the Threshold by Changing the Ratios

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.

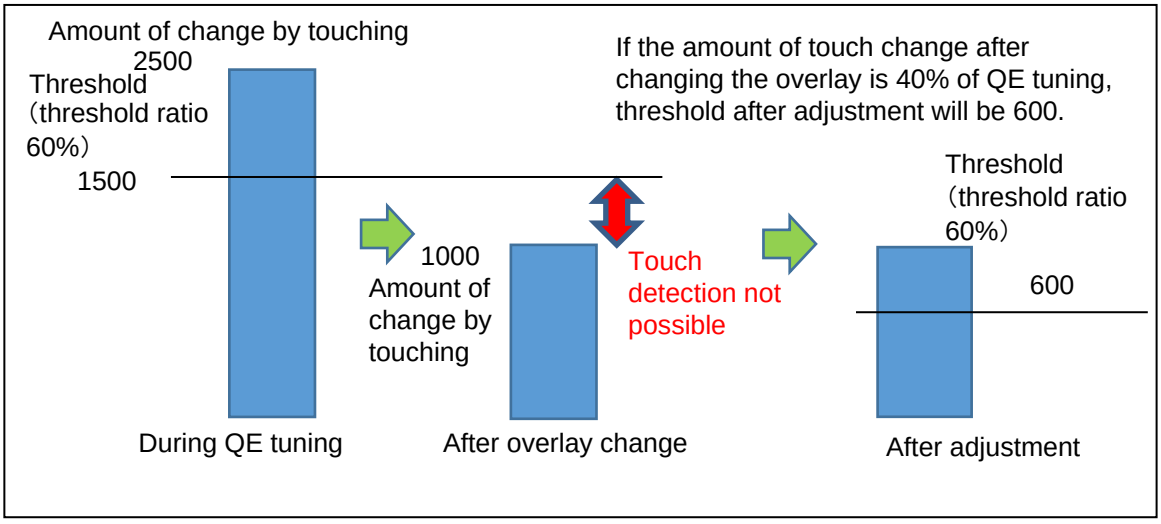


Figure 8 Example of Adjusting the Threshold according to the Touch Measurement Results

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 re-tuning 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 the 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.

1.1.6 Automatic judgment measurement using SMS

This function uses SMS to operate from measurement to touch judgment without CPU operation.

For the touch interface configuration to use this function, call RM_TOUCH_SmsSet () and then start the measurement with RM_TOUCH_ScanStart (). Since the CPU operates only in STOP mode and SNOOZE mode until the touch is judged to be ON, measurement can be performed with low power consumption. In RM_TOUCH_SmsSet (), the Debouncing count of touch-on filter is set to on_freq value and the Debouncing count of touch-off filter is set to 0. Therefore In application, call RM_TOUCH_DataGet () to get the button judgment result as in normal operation.

See Chapter 3.5 and the RL78 Family CTSU Module (R11AN0484) for more information.

1.2 API Overview

The TOUCH module includes the following functions.

Function	Description
RM_TOUCH_Open()	Initializes the specified touch interface configuration.
RM_TOUCH_StartScan()	Starts measurement of specified touch interface configuration.
RM_TOUCH_DataGet()	Gets measured values of specified touch interface configuration.
RM_TOUCH_CallbackSet()	Sets callback function of specified touch interface configuration.
RM_TOUCH_SmsSet()	Makes settings for automatic judgment measurement using SMS of the specified touch interface configuration.
RM_TOUCH_Close()	Closes specified touch interface configuration.
RM_TOUCH_ScanStop()	Stops measurement of specified touch interface configuration.
RM_TOUCH_SensitivityRatioGet ()	Get the ratio of the current touch sensitivity compared 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.

2. API Information

Operations of this module has been confirmed under the following conditions.

2.1 Hardware Requirements

The MCU used in the development must support the following function:

- CTSUb
- CTSU2L
- CTSU2La

2.2 Software Requirements

This driver depends on the following modules:

- Board Support Package (r_bsp) v1.70 or newer
- CTSU module (r_ctsu) v2.10

This driver assumes use of the capacitive touch sensor development support tool:

- QE for Capacitive Touch V4.1.0 or newer

2.3 Supported Toolchains

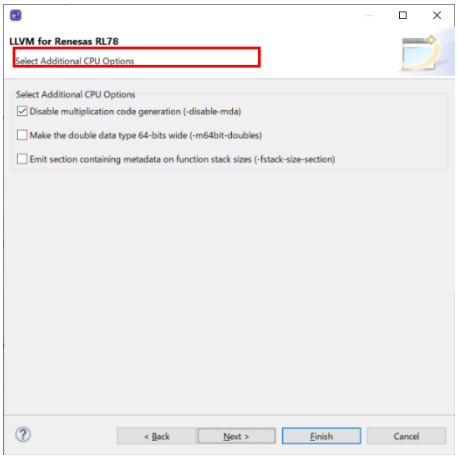
Module operations have been confirmed on the following toolchains.

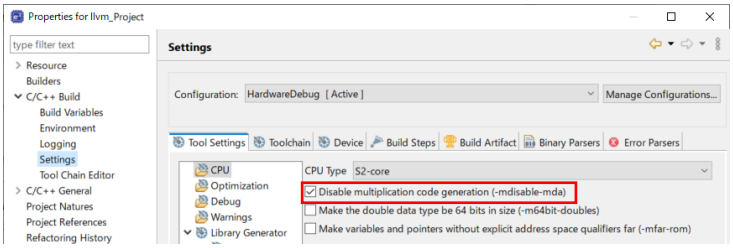
- Renesas CC-RL Toolchain v1.14.00
- IAR Embedded Workbench for Renesas RL78 v5.10.3
- LLVM for RL78 17.0.1.202412

2.4 Restrictions

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

When using the LLVM compiler in the RL78/G16 group, please check the following CPU Options when creating a project. After the project is created, it can be set from the project properties.





2.5 Header File

All interfaces definitions to be called and used in the API are defined in “rm_touch_api.h”.

Select ”rm_touch_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 *Default value: “0”	Set the use of UART tuning. 0: Disable, 1: Enable
TOUCH_CFG_UART_NUMBER	Set the UART channel number.
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_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 Settings”. Compilation Setting” during a build. The values shown are baselines when the compile option is the default for the CC-RL C compiler listed in “section 2.3 Supported Toolchains”. The code size varies according to the C compile version and compile options.

Depending on the application and the number of b2.2uttons, your program may exceed the RAM size. Please note that the RL78/G16 group has 2KB of RAM.

Using Renesas CC-RL Toolchain v1.14.00, 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	2128 bytes	+ 6 bytes	+ 592 bytes	+ 621 bytes	2302 bytes	+ 6 bytes
RAM	97 bytes	+ 24 bytes	+ 13 bytes	+ 15 bytes	99 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	2283 bytes	+ 6 bytes	+582 bytes	+601 bytes	2459 bytes	+6 bytes
RAM	97 bytes	+24 bytes	+13 bytes	+15 bytes	99 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	2866 bytes	+18 bytes	—	—	2989 bytes	+18 bytes
RAM	137 bytes	+64 byte	—	—	143 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_api.h along with the prototype declaration.

The controls structure for the touch interface configuration are shown in Table 2. 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 configuration constructure
ctsu_instance_t const *	p_ctsu_instance	Pointer to CTSU control constructure
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, along with the API function prototype declaration.

```
  /** 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;
```

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 : Pointer to the control structure
p_cfg : 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 initialize CTSU module by calling R_CTSU_Open() after initializing control structure 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(). Refer the R_CTSU_Open() document 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 : 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 : Pointer to the control structure
p_button_status: Pointer to the buffer that stores button state.
p_slider_position: Pointer to the buffer that stores slider position.
p_wheel_position: 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_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 and 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.

The buffer that stores the slider position contains a value of 1~100 for touch and 65535 for non-touch.

The buffer that stores the wheel position contains a value of 1~360 for touch and 65535 for non-touch.

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: Pointer to the control structure

p_callback: Pointer to callback function

p_context: Pointer to send to callback function

p_callback_memory: 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_SmsSet

This function makes settings for automatic judgment measurement using SMS to the specified touch interface configuration.

Format

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

Parameters

p_ctrl : 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 configure the specified touch interface to automatic judgment measurement using SMS. To start automatic judgment measurement, call RM_TOUCH_ScanStart () for the same touch interface after calling this function. When the touch is judged to be ON, the callback function is called. Call RM_TOUCH_DataGet () to get the status of the button.

Example:

```
    fsp_err_t err;
    uint64_t button_status;

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

    /* for ExternalTrigger */
    ELISEL7 = 0x17;
    ELL1SEL0 = 0x08;
    ELL1LNK0 = 0x01;
    ELOSEL6 = 0x01;
    ELOENCTL = 0x40;

    /* Open Touch middleware */
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    /* Offset tuning cannot be performed by SMS measurement, so it should be
performed in advance. */

    /* for ExternalTrigger */
    err = RM_TOUCH_ScanStart(&g_touch_ctrl);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }
    R_ITL_Start_Interrupt();
    R_Config_ITL000_Start();

    /* Measurement loop */
    while (true)
    {
        /* for [CONFIG01] configuration */
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            R_Config_ITL000_Stop();
            break;
        }
    }

    /* Start SMS measurement */
    if (0 == button_status)
    {
        err = RM_TOUCH_SmsSet(&g_touch_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
    }
    err = RM_TOUCH_ScanStart(&g_touch_ctrl);
    if (FSP_SUCCESS != err)
    {
```

```
    while (true) {}
}

R_Config_ITL000_Start();

/* Measurement loop for low power consumption */
__stop();

err = RM_TOUCH_DataGet(&g_touch_ctrl, &button_status, NULL, NULL);
if (FSP_SUCCESS == err)
{
   if (button_status)
   {
      /* LED ON  */
   }
   else
   {
      /* LED OFF */
   }
}
```

Special Notes:

Please call this function after confirming touch OFF. If this function is called with touch ON, the baseline will be set with touch ON, and touch detection will not be possible until it is updated with the baseline drift function.

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

3.6 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 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

3.7 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: 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

3.8 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: Pointer to the control structure

p_touch_sensitivity_info: 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

This function estimates 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.

3.9 RM_TOUCH_ThresholdAdjust

This function adjust the touch determination threshold and hysteresis value to 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: Pointer to the control structure

p_touch_sensitivity_info: 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.

3.10 RM_TOUCH_DriftControl

This function changes the settings of drift correction to 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: Pointer to the control structure

input_drift_freq: 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);
```

Special Notes:

None

3.11 RM_TOUCH_MonitorAddressGet

This function returns the address of buffer used for QE monitor of specified touch interface configurations.

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: Pointer to the control structure

pp_monitor_buf : Pointer to the variable storing starting address of monitor buffer

pp_monitor_id : Pointer to the variable storing address of monitor ID variables

pp_monitor_size : Pointer to the variable storing starting address of monitor size

Return Values

FSP_SUCCESS	*/* Successfully get QE monitor variable address */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified*/*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*
FSP_ERR_NOT_ENABLED	*/* Requested operation is not enabled */*

Properties

Prototype is declared in rm_touch_api.h.

Description

This function is used for QE monitoring features when there are two touch interface configurations, software judgement and auto judgement. It returns the starting address of monitor buffer as second arguments, address of monitor ID variables with third arguments and starting address of monitor size with fourth arguments.

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 used for QE monitoring features.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.13.21	-	First edition issued
1.10	Aug.31.21	3	Updated 1.1.1 QE for Capacitive Touch Usage
		5	Added 1.1.5 Tuning the Touch Determination Threshold
		6	Added 1.1.6 Automatic judgment measurement using SMS
		9	Updated 2.7 Compilation settings
		10	Updated 2.8 Code Size
		10	Updated 2.9 Arguments
		11	Updated 2.10 Return Values
		-	Deleted RM_TOUCH_VersionGet
		19	Added 3.5 RM_TOUCH_SmsSet
		23	Added 3.7 RM_TOUCH_ScanStop
		24	Added 3.8 RM_TOUCH_SensitivityRatioGet
		26	Added 3.9 RM_TOUCH_Threshold Adjust
		28	Added 3.10 RM_TOUCH_DriftControl
1.11	Jan.18.22	8	Updated 2.2 Software Requirements
		8	Updated 2.3 Supported Toolchains
		10	Updated 2.8 Code size
1.20	Apr.20.22	6	Updated 1.1.6 Automatic judgment measurement using SMS
		24	Fixed Example: in 3.5 RM_TOUCH_SmsSet
1.30	Feb.14.23	1	Added RL78/G22 to Target Device
		4	Updated 1.1.3 Button Touch Determination
		9	Updated 2.2 Software Requirements
		9	Updated 2.3 Supported Toolchains
		11	Fixed 2.8 Code Size
		13	Updated 2.10 Return Values
		17	Updated 3.3 RM_TOUCH_DataGet
1.40	Jun.14.23	1	Added RL78/G16 group to Target Device
		9	Updated 2.1 Hardware Requirements
		9	Updated 2.2 Software Requirements
		9	Updated 2.4 Restrictions
		12	Fixed 2.8 Code Size
1.50	May.31.24	9	Updated 2.2 Software Requirements
		9	Updated 2.3 Supported Toolchains
		22	Updated 3.5 RM_TOUCH_SmsSet
2.00	Oct.15.24	3	Updated 1.1.2 Measurements and Data Processing
		7	Updated 1.1.5 Tuning the Touch Determination Threshold
		11	Updated 2.1 Hardware Requirements
		11	Updated 2.2 Software Requirements
		11	Updated 2.3 Supported Toolchains
		13	Updated 2.7 Compilation Settings
		14	Updated 2.8 Code Size
		15	Updated 2.9 Arguments
		27	Updated Description of 3.8 RM_TOUCH_SensitivityRatioGet
		28	Updated Description of 3.9 RM_TOUCH_ThresholdAdjust
		30	Added 3.11 RM_TOUCH_MonitorAddressGet
2.10	Feb.19.25	3	Updated 1.1.3 Button Touch Determination
		11	Updated 2.2 Software Requirements
		11	Updated 2.3 Supported Toolchains
		14	Updated 2.8 Code Size

		20	Updated Description of 3.3 RM_TOUCH_DataGet

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.