RX Family

QE CTSU module Firmware Integration Technology

Introduction

This application note describes the CTSU module.

Target Device

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

Firmware Integration Technology User's Manual (R01AN1833)
Board Support Package Firmware Integration Technology Module (R01AN1685)
Adding Firmware Integration Technology Module 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

The CTSU module is a CTSU driver for the Touch module. The CTSU module assumes the access from the Touch middleware layer, and it is also accessible from an user application.

RX140 supports CTSU2L and the other devices support CTSU.

CTSU and CTSU2L are functionally different, so CTSU and CTSU2L are described in this application note as below.

・Common description for CTSU and CTSU2L -> CTSU

・Description only for CTSU -> CTSU1

・Description only for CTSU2L -> CTSU2L

* Without mention, it means the common description for CTSU and CTSU2L.

1.1 Functions

The CTSU module supports the following functions.

1.1.1 QE for Capacitive Touch Usage

The module provides various capacitive touch measurements based on configuration settings generated by QE for Capacitive Touch.

As a part of the configuration settings, the touch interface configuration displays the combination of terminals to be measured (referred to as TS) and the corresponding measurement mode. Multi-touch interface configurations are necessary when the development product has a combination of different measurement modes or when the active shield is used.

1.1.2 Measurements and Obtaining Data

Measurements can be started by a software trigger or by an external event triggered by the Event Link Controller (ELCL).

As the measurement process is carried out by the CTSU2L peripheral, it does not use up main processor processing time.

The CTSU module processes INTCTSUWR and INTCTSURD if generated during a measurement. The data transfer controller (DTC) can also be used for these processes.

When the measurement complete interrupt (INTCTSUFN) process is complete, the application is notified in a callback function. Make sure you obtain the measurement results before the next measurement is started as internal processes are also executed when a measurement is completed.

Start the measurement with API function R_CTSU_ScanStart().

Obtain the measurement results with API function R_CTSU_DataGet().

1.1.3 Sensor ICO Correction function

The CTSU2L peripheral has a built-in correction circuit to handle the potential microvariations related to the manufacturing process of the sensor ICO MCU.

The module temporarily transitions to the correction process during initialization after power is turned on. In the correction process, the correction circuit is used to generate a correction coefficient (factor) to ensure accurate sensor measurement values.

When temperature correction is enabled, an external resistor connected to a TS terminal is used to periodically update the correction coefficient. By using an external resistor that is not dependent on temperature, you can even correct the temperature drift of the sensor ICO.

1.1.4 Initial Offset Adjustment

The CTSU2L peripheral was designed with a built-in offset current circuit in consideration of the amount of change in current due to touch. The offset current circuit cancels enough of the parasitic capacitance for it to fit within the sensor ICO dynamic range.

This module automatically adjusts the offset current setting. As the adjustment uses the normal measurement process, R_CTSU_ScanStart() and R_CTSU_DataGet() must be repeated several times after startup. Because the ctsu_element_cfg_t member “so” is the starting point for adjustments, you can set the appropriate value for “so” in order to reduce the number of times the two functions must be run to complete the adjustment. Normally, the value used for “so” is a value adjusted by QE for Capacitive Touch.

For CTSU2L, this feature can be turned off in the config.

1.1.5 Random Pulse Frequency Measurement (CTSU1)

The CTSU1 peripheral measures at one drive frequency.

The drive frequency determines the amperage to the electrode and generally uses the value tuned with QE for Capacitive Touch.

The drive frequency is calculated as below.

It is determined by PCLK frequency input to CTSU, CTSU Count Source Select bit(CTSUCLK), and CTSU Sensor Drive pulse Division Control bit(CTSUSDPA). For example, If it is set PCLK =32MHz, CTSUCLK = PLCK/2, and CTSUSDPA = 1/16, then drive frequency is 0.5MHz. CTSUSDPA can change for each TS port.

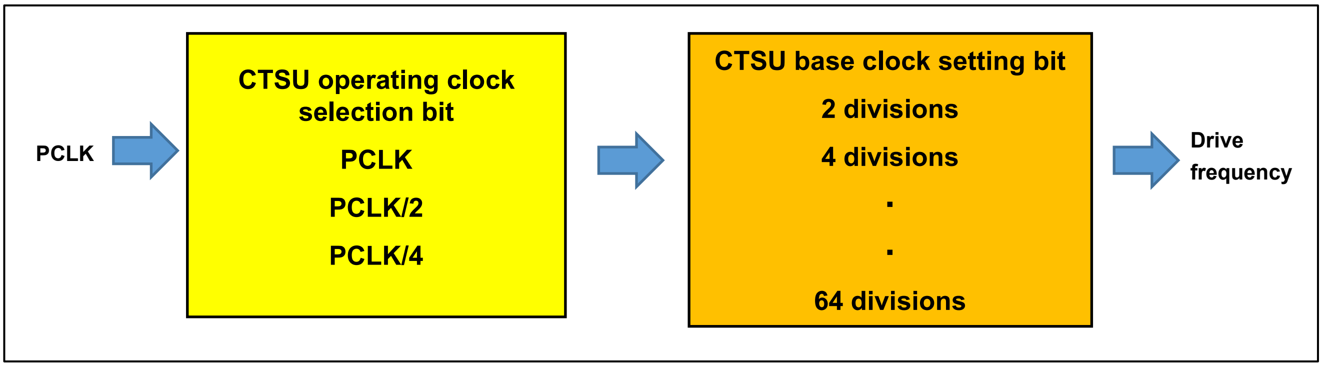

Figure 1 Drive Frequency Settings

The actual drive pulse is phase-shifted and frequency-spread with respect to the clock based on the drive frequency as a measure against external environmental noise. This module is fixed at initialization and sets the following.

CTSUSOFF = 0, CTSUSSMOD = 0, CTSUSSCNT = 3

1.1.6 Multi-frequency Measurements (CTSU2L)

The CTSU2L peripheral can measure in one of four drive frequencies to avoid synchronous noise.

With the default settings, the module takes measurements at three different frequencies. After standardizing the results obtained at the three frequencies in accordance with the first frequency reference value, the measured value is determined based on majority in a process referred to as “normalization.”

When this normalization is turned off in the config settings, the user can use the results of these three frequencies as noise filters. However, the three frequencies cannot be tied with the Touch module.

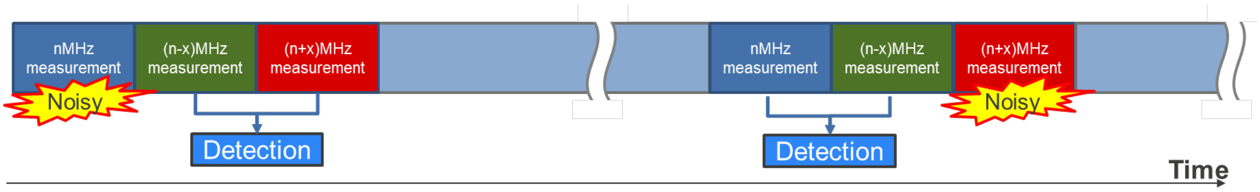

Figure 2 Multi-frequency Measurements

Drive frequency is determined based on the config settings. The module sets registers according to the config settings, and sets the three drive frequencies.

Drive frequency is calculated in the following equation:

(PCLKB frequency / CLK / STCLK) x SUMULTIn / 2 / SDPA : n = 0, 1, 2

The figure below shows the settings for generating a 2MHz drive frequency when the PCLKB frequency is 32 MHz. SDPA can be set for each touch interface configuration.

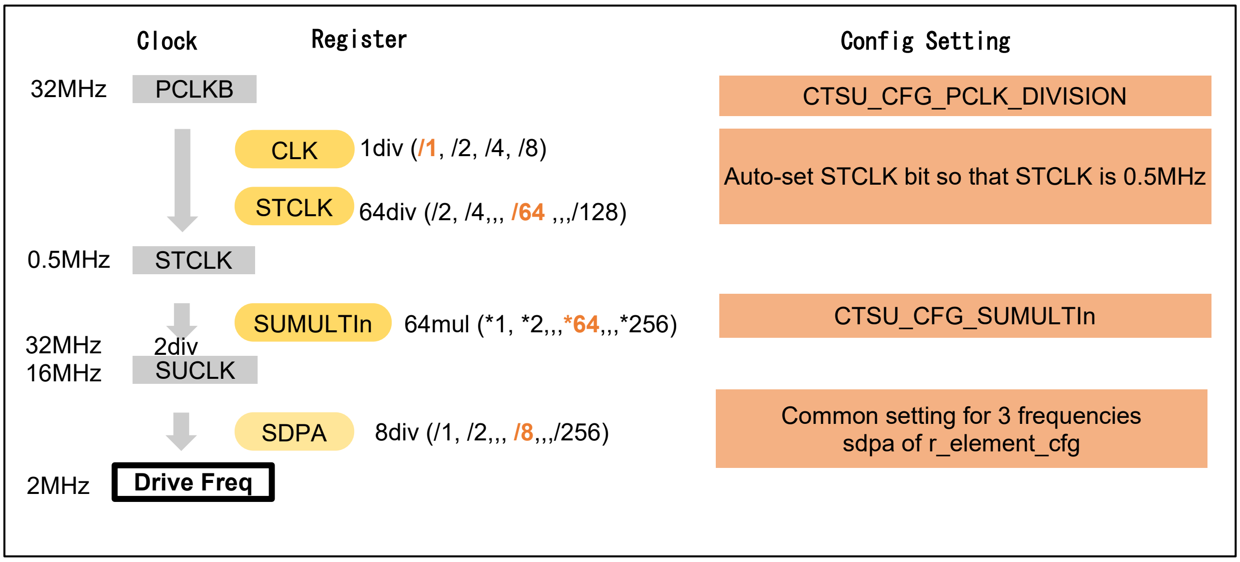

Figure 3 Drive Frequency Settings

1.1.7 Shield Function(CTSU2L)

The CTSU2L peripheral has a built-in function that outputs a shield signal in phase with the drive pulse from the shield terminal and the non-measurement terminal in order to shield against external influences while suppressing any increase in parasitic capacitance. This function can only be used during self-capacitance measurements.

This module allows the user to set a shield for each touch interface configuration.

For example, for the electrode configuration shown in , the members of ctsu_cfg_t should be set as follows. Other members have been omitted for the example.

```
.txvsel     = CTSU_TXVSEL_INTERNAL_POWER,
.txvsel2    = CTSU_TXVSEL_MODE,
.md         = CTSU_MODE_SELF_MULTI_SCAN,
.posel      = CTSU_POSEL_SAME_PULSE,
.ctsuchac0  = 0x0F,
.ctsuchtrc0 = 0x08,
```

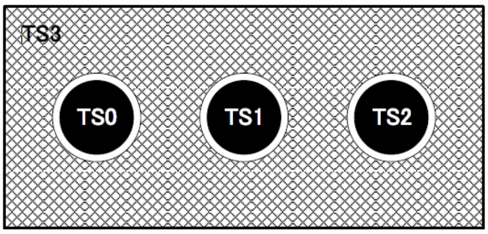

Figure 4 Example of Shield Electrode Structure

1.1.8 Measurement Error Message

When the CTSU2L peripheral detects an abnormal measurement, it sets the status register bit to 1.

In the measurement complete interrupt process, the module reads ICOMP1, ICOMP0, and SENSOVF of the status register and notifies the results in the callback function. The status register is reset after the contents are read. For more details on abnormal measurements, refer to “member event” in the ctsu_callback_args_t callback function argument.

1.1.9 Moving Average

This function calculates the moving average of the measured results.

Set the number of times the moving average should be calculated in the config settings.

1.1.10 Diagnosis Function

The CTSU peripheral has a built-in function that diagnoses its own inner circuit. This diagnosis function provides the API for diagnosing the inner circuit.

The diagnostic requirements are different for CTSU1 and CTSU2L providing 5 types of diagnosis for CTSU1 and 9 types for CTSU2L.

The diagnosis function is executed by calling the API function. This is executed independently from the other measurements and does not affect them.

To enable the diagnosis function, set CTSU_CFG_DIAG_SUPPORT_ENABLE to 1.

For CTSU1, a 27pF condenser should be connected externally.

For CTSU2L, use ADC FIT (r_s12ad_rx).

1.2 Measurement Mode

This module supports all three modes offered by the CTSU2L peripheral: self-capacitance, mutual capacitance, and current measurement modes. The temperature correction mode is also offered as a mode for updating the correction coefficient.

1.2.1 Self-capacitance Mode

The self-capacitance mode is used to measure the capacitance of each terminal (TS).

The CTSU2L peripheral measures the terminals in ascending order according to the TS numbers, then stores the data. For example, even if you want to use TS5, TS8, TS2, TS3 and TS6 in your application in that order, they will still be measured and stored in the order of TS2, TS3, TS5, TS6, and TS8. Therefore, you will need to reference buffer indexes [2], [4], [0], [1], and [3].

[CTSU1]

In default settings, the measurement period for each TS is wait-time plus approximately 526us.

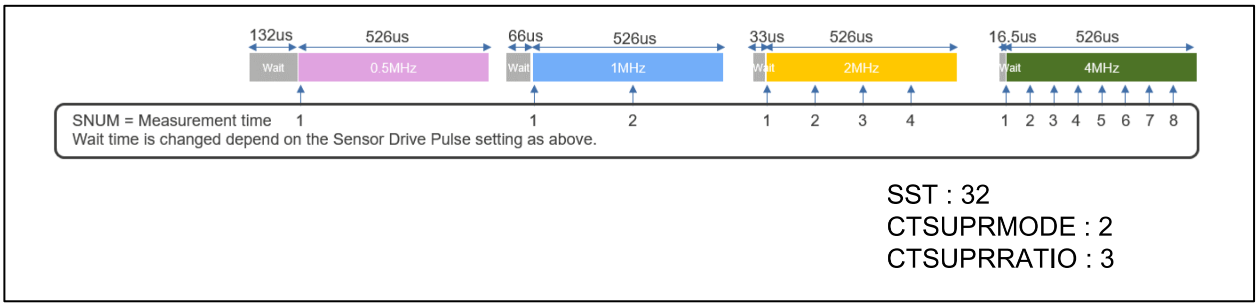

Figure 5 Self-capacitance Measurement Period (CTSU1)

[CTSU2L]

In default settings, the measurement period for each TS is approximately 576us.

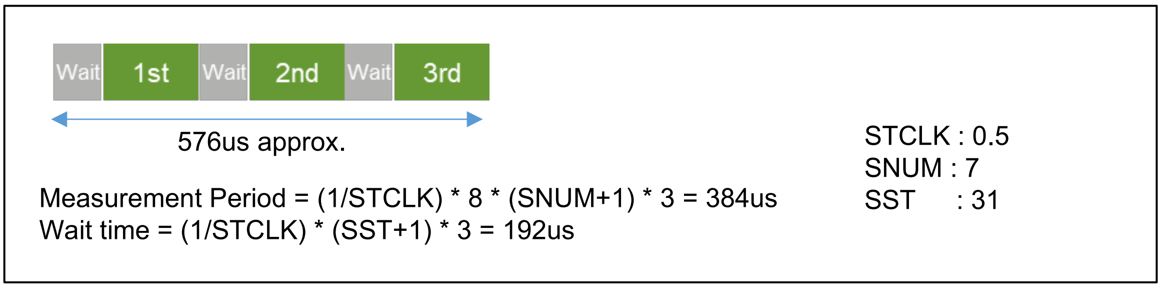

Figure 6 Self-capacitance Measurement Period (CTSU2L)

1.2.2 Mutual Capacitance Mode

The mutual capacitance mode is used to measure the capacitance generated between the receive TS (Rx) and transmit TS (Tx), and therefore requires at least two terminals.

The CTSU2L peripheral measures all specified combinations of Rx and Tx. For example, when Rx is TS1 and TS3, and Tx is TS2, TS7 and TS4, the combinations are measured in the following order and the data is stored.

TS3-TS2, TS3-TS4, TS3-TS7, TS10-TS2, TS10-TS4, TS10-TS7

To measure the mutual capacitance generated between electrodes, the CTSU2L peripheral performs the measurement process on the same electrode twice.

The mutual capacitance is obtained by inverting the phase relationship of the pulse output and switched capacitor in the primary and secondary measurements, and calculating the difference between the two measurements. This module does not calculate the difference, but outputs the secondary measured result.

[CTSU1]

In default settings, the measurement period for each TS is twice of wait-time plus approximately 526us.

[CTSU2L]

In default settings, the measurement period for each TS is approximately 1152us.

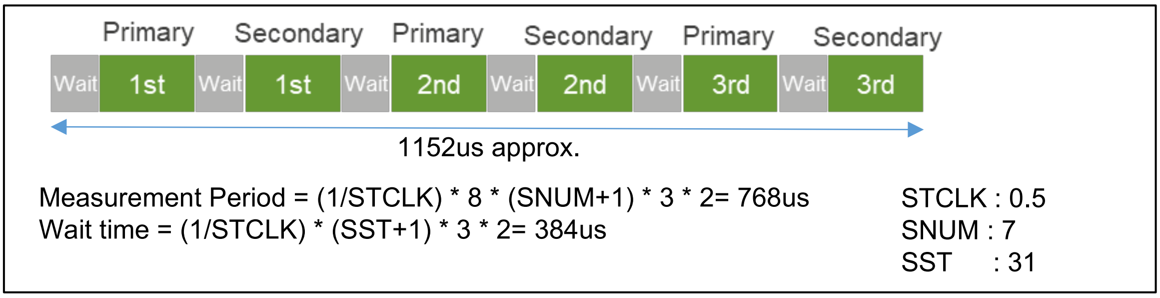

Figure 7 Mutual Capacitance Measurement Period (CTSU2L)

1.2.3 Current Measurement Mode(CTSU2L)

The current measurement mode is used to measure the minute current input to the TS terminal.

The order of measurement and data storage is the same as that of the self-capacitance mode.

As this does not involve the switched capacitor operation, the measurement is only performed once. The measurement period for one TS under default settings is approximately 256us. The current measurement mode requires a longer stable wait time than the other modes, so the SST is set to 63.

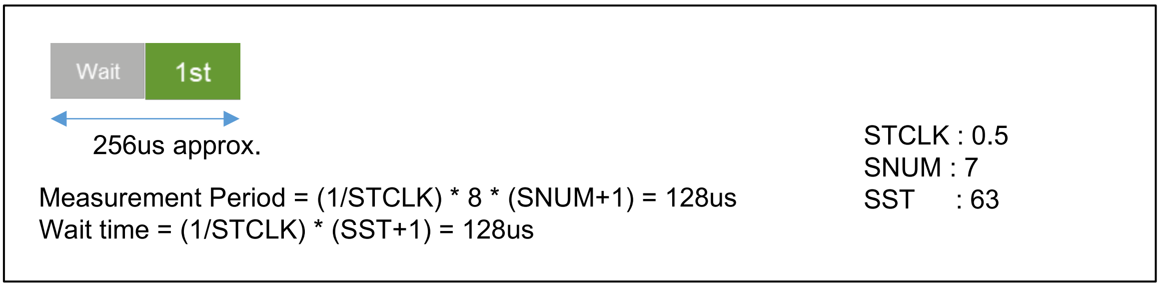

Figure 8 Current Measurement Period

1.2.4 Temperature Correction Mode(CTSU2L)

The temperature correction mode is used to periodically update the correction coefficient using an external resistor connected to a TS terminal. This involves three processes as described below. Also refer to the timing chart in Figure 9 Temperature Correction Measurement Timing Chart.

1. Measure the correction circuit. One set comprises twelve measurements.

2. Measure the current when TSCAP voltage is applied to the external resistor to create a correction coefficient based on an external resistor that does not depend on temperature. Execute the next measurement after the previous measurement set is completed (as described in step 1).

3. Flow offset current to the external resistor and measure the voltage with the ADC. This will adjust the RTRIM register and handle the temperature drift of the internal reference resistor. In the config settings, set the number of times step 2 should be executed before carrying out this measurement.

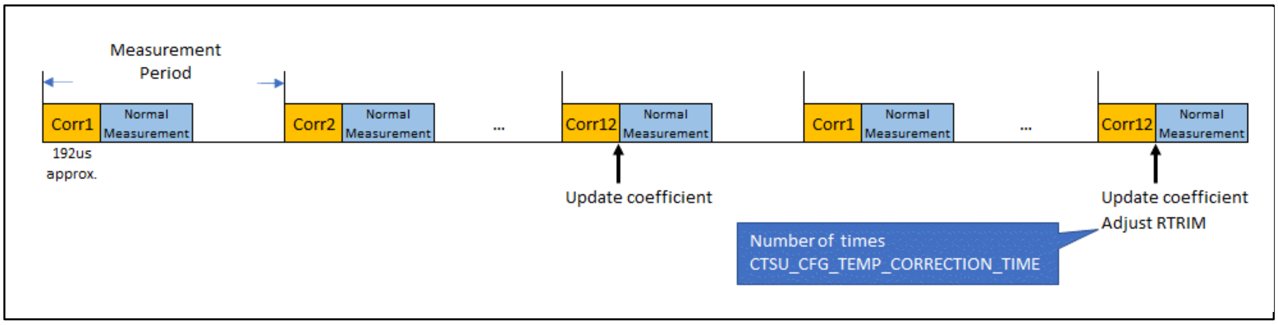

Figure 9 Temperature Correction Measurement Timing Chart

1.2.5 Diagnosis Mode

The diagnosis mode is a mode in which various internal measurement values are scanned by using this diagnosis function.

The details are described in 1.1.10.

1.3 Measurement Timing

As explained in section 1.1.2, measurements are initiated by a software trigger or an external event which is triggered by the Event Link Controller (ELCL).

The most common method is using a timer to carry out periodic measurements. Make sure to set the timer interval to allow the measurement and internal value update processes to complete before the next measurement period. The measurement period differs according to touch interface configuration and measurement mode. See section 1.2 for details.

The execution timing of software triggers and external triggers differ slightly.

Since a software trigger sets the start flag after setting the touch interface configuration with R_CTSU_ScanStart (), there is a slight delay after the timer event occurrence. However, as the delay is much smaller than the measurement period, a software trigger is recommended for most instances as it is easy to set.

An external trigger is recommended for applications in which this slight delay is not acceptable or that require low-power consumption operations. When using an external trigger with multiple touch interface configurations, use R_CTSU_ScanStart() to set another touch interface configuration after one measurement is completed.

1.4 API Overview

The CTSU module includes the following functions.

Function	Description
R_CTSU_Open()	Initializes the specified touch interface configuration.
R_CTSU_ScanStart()	Starts measurement of specified touch interface configuration.
R_CTSU_DataGet()	Gets measured values of specified touch interface configuration.
R_CTSU_CallbackSet()	Set callback function of specified touch interface configuration.
R_CTSU_Close()	Closes specified touch interface configuration.
R_CTSU_Diagnosis()	Executes diagnosis.

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:

- CTSU1
- CTSU2L

2.2 Software Requirements

This driver depends on the following FIT modules:

- Board support package module (r_bsp) v6.10 or newer

According to the configuration settings, the driver may also depend on the following modules:

- DTC module r_dtc v3.80 or newer (In case of using DTC transfer)
  When using DTC transfer, set the Heap size of the r_bsp property to 0x800 or more.
- ADC module r_s12ad_rx_v4.90 or newer (In case of using Temperature correction mode or diagnosis mode)

This driver also assumes the use of following tool:

- Renesas QE for Capacitive Touch V2.0.0 or nower

2.3 Supported Toolchains

Module operations have been confirmed on the following toolchains:

- Renesas CC-RX Toolchain v3.03.00
- IAR RX Toolchain v4.20.1
- GCC RX Toolchain v 8.3.0.201904

2.4 Restrictions

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

2.5 Header File

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

Select “r_ctsu_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 CTSU module.

r_ctsu_config.h Configuration Options	
CTSU_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. "1": Omit parameter check process from code. "2": Include parameter check process in code. "BSP_CFG_PARAM_CHECKING_ENABLE": Selection depends on BSP setting.
CTSU_CFG_USE_DTC *Default value: "0"	Select "1" to use the DTC, rather than the main processor, to run the CTSU2L's CTSUWR interrupt and CTSURD interrupt processes. Note: If the DTC is used elsewhere in the application, it may compete with the use of this driver.
CTSU_CFG_INTCTSUWR_PRIORITY_LEVEL *Default value: "2"	Sets the CTSUWR interrupt priority level (also necessary when using the DTC). The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_INTCTSURD_PRIORITY_LEVEL *Default value: "2"	Sets the CTSURD interrupt priority level (also necessary when using the DTC). The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_INTCTSUFN_PRIORITY_LEVEL *Default value: 2	Sets the CTSUFN interrupt priority level. The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_DTC_USE_SC	
The following configurations depend on the touch interface configuration and cannot be set using Smart Configurator. These configurations are set when using QE for Capacitive Touch. In this case, QE_TOUCH_CONFIGURATION is defined in the project. Although r_ctsu_config.h becomes invalid, qe_touch_define.h is defined instead.	Sets the total number of TS for self-capacitance, current measurement, and temperature correction.
CTSU_CFG_NUM_SELF_ELEMENTS	Sets the total number of matrixes for mutual capacitance
CTSU_CFG_NUM_MUTUAL_ELEMENTS	Enables/disables the low voltage mode. This value is set in the CTSUCRAL register's ATUNE0 bit.
CTSU_CFG_LOW_VOLTAGE_MODE	Sets the PCLK frequency division rate. This value is set in the CTSUCRAL register's CLK bit.
CTSU_CFG_PCLK_DIVISION	Sets the TSCAP port. Example: For P30, set 0x0300.
CTSU_CFG_TSCAP_PORT	Sets the VCC (voltage). Example: for 5.00V, set 5000.
CTSU_CFG_VCC_MV	Sets the number of multi-frequency measurements.
CTSU_CFG_NUM_SUMULTI	Sets the multiplication factor for the first frequency in a multi-frequency measurement. Recommended: 0x3F
CTSU_CFG_SUMULTI0	Sets the multiplication factor for the second frequency in a multi-frequency measurement. Recommended: 0x36
CTSU_CFG_SUMULTI1	Sets the multiplication factor for the third frequency in a multi-frequency measurement. Recommended: 0x48

CTSU_CFG_SUMULTI2	Enables/disables temperature correction.
CTSU_CFG_TEMP_CORRECTION_SUPPORT	Sets the temperature correction terminal number.
CTSU_CFG_TEMP_CORRECTION_TS	Sets the update interval for the correction coefficient of the temperature correction. Assuming 13 measurements per set in the temperature correction mode, indicate the number of sets per update.
CTSU_CFG_TEMP_CORRECTION_TIME	Enables/disables RTRIM correction for temperature correction. The ADC must be selected to operate with RTRIM correction enabled.
CTSU_CFG_DIAG_SUPPORT_ENABLE	Enables/disables diagnosis function.
CTSU_CFG_DIAG_DAC_TS	Sets the number of TS pin to be used for diagnosis in CTSU1.

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 reference values 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 compiler version or the compile options.

[CTSU1]

ROM and RAM Usage the configuration options with Self-capacitance 1element	
CTSU_CFG_PARAM_CHECKING_ENABLE 0 CTSU_CFG_DTC_SUPPORT_ENABLE 0	ROM: 1817 bytes RAM: 57 bytes

ROM and RAM Usage Size of each mode, amount of increase by adding elements				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	1817 bytes	+0 bytes	1939 bytes	+0 bytes
RAM	57 bytes	+17 bytes	63 bytes	+23 bytes

[CTSU2L]

ROM and RAM Usage the configuration options with Self-capacitance 1element	
CTSU_CFG_PARAM_CHECKING_ENABLE 0 CTSU_CFG_DTC_SUPPORT_ENABLE 0	ROM: 3379 bytes RAM: 189 bytes

ROM and RAM Usage Size of each mode, amount of increase by adding elements				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	3379 bytes	+0 bytes	3426 bytes	+0 bytes
RAM	189 bytes	+29 bytes	201 bytes	+41 bytes

2.9 Arguments

The following are 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 inr_ctsu_qe.h, r_ctsu_qe_api.h.

The following is the control structure for the touch interface configuration. This does not need to be set in the application. Using QE 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.

```
typedef struct st_ctsu_instance_ctrl
{
    uint32_t              open;             ///< Whether or not driver is open.
    volatile ctsu_state_t state;            ///< CTSU run state.
    ctsu_cap_t            cap;              ///< CTSU Scan Start Trigger Select
    ctsu_tuning_t         tuning;           ///< CTSU Initial offset tuning status.
    uint16_t              num_elements;     ///< Number of elements to scan
    uint16_t              wr_index;         ///< Word index into ctsuwr register array.
    uint16_t              rd_index;         ///< Word index into scan data buffer.
    uint8_t             * p_tuning_count;   ///< Pointer to tuning count of each element. g_ctsu_tuning_count[]
is set by Open API.
    int32_t             * p_tuning_diff;    ///< Pointer to difference from base value of each element.
g_ctsu_tuning_diff[] is set by Open API.
    uint16_t              average;          ///< CTSU Moving average counter.
    uint16_t              num_moving_average; ///< Copy from config by Open API.
    uint8_t               ctsucr1;          ///< Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW,
and PON is set by HAL driver.
    ctsu_ctsuwr_t       * p_ctsuwr;         ///< CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API.
    ctsu_self_buf_t     * p_self_raw;       ///< Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API.
    uint16_t            * p_self_data;      ///< Pointer to Self moving average data. g_ctsu_self_data[] is set
by Open API.
    ctsu_mutual_buf_t   * p_mutual_raw;     ///< Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open
API.
    uint16_t            * p_mutual_pri_data; ///< Pointer to Mutual primary moving average data.
g_ctsu_mutual_pri_data[] is set by Open API.
    uint16_t            * p_mutual_snd_data; ///< Pointer to Mutual secondary moving average data.
g_ctsu_mutual_snd_data[] is set by Open API.
    ctsu_correction_info_t * p_correction_info; ///< Pointer to correction info

#if (BSP_FEATURE_CTSU_VERSION == 1)
 #if (CTSU_CFG_DIAG_SUPPORT_ENABLE == 1)
    ctsu_diag_info_t * p_diag_info;         ///< pointer to diagnosis info
 #endif
#endif

#if (BSP_FEATURE_CTSU_VERSION == 2)
    ctsu_range_t range;                     ///< According to atune12. (20uA : 0, 40uA : 1, 80uA : 2, 160uA : 3)
    uint8_t      ctsucr2;                   ///< Copy from (posel, atune1, md) by Open API. FCMODE and SDPSEL and
LOAD is set by HAL driver.
 #if (CTSU_CFG_NUM_CFC != 0)
    uint64_t            cfc_rx_bitmap;      ///< Bitmap of CFC receive terminal.
    ctsu_corrcfc_info_t * p_corrcfc_info;   ///< pointer to CFC correction info
 #endif
 #if (CTSU_CFG_DIAG_SUPPORT_ENABLE == 1)
    ctsu_diag_info_t * p_diag_info;         ///< pointer to diagnosis info
 #endif
#endif
    ctsu_cfg_t const * p_ctsu_cfg;          ///< Pointer to initial configurations.
    void (* p_callback)(ctsu_callback_args_t *); ///< Callback provided when a CTSUFN occurs.
    ctsu_event_t     error_status;   ///error status variable.
    void const          * p_context;        ///< Placeholder for user data.
} ctsu_instance_ctrl_t;
```

The following is the configuration setting structure for the touch interface configuration.

Using QE for Capacitive Touch allows the variables and initialization values corresponding to the touch interface configuration to be output by qe_touch_config.c. Make sure to set qe_touch_config.c in the second argument of R_CTSU_Open().

```
typedef struct st_ctsu_cfg
{
    ctsu_cap_t              cap;                ///< CTSU Scan Start Trigger Select
    ctsu_txvsel_t           txvsel;             ///< CTSU Transmission Power Supply Select
    ctsu_txvsel2_t          txvsel2;            ///< CTSU Transmission Power Supply Select 2 (CTSU2 Only)
    ctsu_atune1_t           atune1;             ///< CTSU Power Supply Capacity Adjustment (CTSU Only)
    ctsu_atune12_t          atune12;            ///< CTSU Power Supply Capacity Adjustment (CTSU2 Only)
    ctsu_md_t               md;                 ///< CTSU Measurement Mode Select
    ctsu_posel_t            posel;              ///< CTSU Non-Measured Channel Output Select (CTSU2 Only)
    uint8_t                 ctsuchac0;          ///< TS00-TS07 enable mask
    uint8_t                 ctsuchac1;          ///< TS08-TS15 enable mask
    uint8_t                 ctsuchac2;          ///< TS16-TS23 enable mask
    uint8_t                 ctsuchac3;          ///< TS24-TS31 enable mask
    uint8_t                 ctsuchac4;          ///< TS32-TS39 enable mask
    uint8_t                 ctsuchtrc0;         ///< TS00-TS07 mutual-tx mask
    uint8_t                 ctsuchtrc1;         ///< TS08-TS15 mutual-tx mask
    uint8_t                 ctsuchtrc2;         ///< TS16-TS23 mutual-tx mask
    uint8_t                 ctsuchtrc3;         ///< TS24-TS31 mutual-tx mask
    uint8_t                 ctsuchtrc4;         ///< TS32-TS39 mutual-tx mask
    ctsu_element_cfg_t const * p_elements;      ///< Pointer to elements configuration array
    uint8_t                 num_rx;             ///< Number of receive terminals
    uint8_t                 num_tx;             ///< Number of transmit terminals
    uint16_t                num_moving_average; ///< Number of moving average for measurement data
    bool tunning_enable;                        ///< Initial offset tuning flag
    bool judge_multifreq_disable;               ///< Disable to judge multi frequency
    void (* p_callback)(ctsu_callback_args_t * p_args); ///< Callback provided when CTSUFN ISR occurs.
    void const * p_context;                     ///< User defined context passed into callback function.
    void const * p_extend;                      ///< Pointer to extended configuration by instance of
interface.
} ctsu_cfg_t;
```

The followings are the enums used for the above listed structures.

```
/** CTSU Events for callback function */
typedef enum e_ctsu_event
{
    CTSU_EVENT_SCAN_COMPLETE = 0x00,    ///< Normal end
    CTSU_EVENT_OVERFLOW      = 0x01,    ///< Sensor counter overflow (CTSUST.CTSUSOVF set)
    CTSU_EVENT_ICOMP         = 0x02,    ///< Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set)
    CTSU_EVENT_ICOMP1        = 0x04     ///< Abnormal sensor current (CTSUSR.ICOMP1 set)
} ctsu_event_t;

/** CTSU Scan Start Trigger Select */
typedef enum e_ctsu_cap
{
    CTSU_CAP_SOFTWARE,                  ///< Scan start by software trigger
    CTSU_CAP_EXTERNAL                   ///< Scan start by external trigger
} ctsu_cap_t;

/** CTSU Transmission Power Supply Select */
typedef enum e_ctsu_txvsel
{
    CTSU_TXVSEL_VCC,                    ///< VCC selected
    CTSU_TXVSEL_INTERNAL_POWER          ///< Internal logic power supply selected
} ctsu_txvsel_t;

/** CTSU Transmission Power Supply Select 2 (CTSU2 Only) */
typedef enum e_ctsu_txvsel2
{
    CTSU_TXVSEL_MODE,                   ///< Follow TXVSEL setting
    CTSU_TXVSEL_VCC_PRIVATE,            ///< VCC private selected
} ctsu_txvsel2_t;

/** CTSU Power Supply Capacity Adjustment (CTSU Only) */
typedef enum e_ctsu_atune1
{
    CTSU_ATUNE1_NORMAL,                 ///< Normal output (40uA)
```

```
        CTSU_ATUNE1_HIGH                  ///< High-current output (80uA)
    } ctsu_atune1_t;

    /** CTSU Power Supply Capacity Adjustment (CTSU2 Only) */
    typedef enum e_ctsu_atune12
    {
        CTSU_ATUNE12_80UA,                ///< High-current output (80uA)
        CTSU_ATUNE12_40UA,                ///< Normal output (40uA)
        CTSU_ATUNE12_20UA,                ///< Low-current output (20uA)
        CTSU_ATUNE12_160UA                ///< Very high-current output (160uA)
    } ctsu_atune12_t;

    /** CTSU Measurement Mode Select */
    typedef enum e_ctsu_mode
    {
        CTSU_MODE_SELF_MULTI_SCAN  = 1,   ///< Self-capacitance multi scan mode
        CTSU_MODE_MUTUAL_FULL_SCAN = 3,   ///< Mutual capacitance full scan mode
        CTSU_MODE_MUTUAL_CFC_SCAN  = 7,   ///< Mutual capacitance cfc scan mode (CTSU2 Only)
        CTSU_MODE_CURRENT_SCAN     = 9,   ///< Current scan mode (CTSU2 Only)
        CTSU_MODE_CORRECTION_SCAN  = 17,  ///< Correction scan mode (CTSU2 Only)
        CTSU_MODE_DIAGNOSIS_SCAN   = 33   ///< Diagnosis scan mode
    } ctsu_md_t;

    /** CTSU Non-Measured Channel Output Select (CTSU2 Only) */
    typedef enum e_ctsu_posel
    {
        CTSU_POSEL_LOW_GPIO,              ///< Output low through GPIO
        CTSU_POSEL_HI_Z,                  ///< Hi-Z
        CTSU_POSEL_LOW,                   ///< Output low through the power setting by the TXVSEL[1:0] bits
        CTSU_POSEL_SAME_PULSE             ///< Same phase pulse output as transmission channel through the power setting
by the TXVSEL[1:0] bits
    } ctsu_posel_t;

    /** CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) */
    typedef enum e_ctsu_ssdiv
    {
        CTSU_SSDIV_4000,                  ///< 4.00 <= Base clock frequency (MHz)
        CTSU_SSDIV_2000,                  ///< 2.00 <= Base clock frequency (MHz) < 4.00
        CTSU_SSDIV_1330,                  ///< 1.33 <= Base clock frequency (MHz) < 2.00
        CTSU_SSDIV_1000,                  ///< 1.00 <= Base clock frequency (MHz) < 1.33
        CTSU_SSDIV_0800,                  ///< 0.80 <= Base clock frequency (MHz) < 1.00
        CTSU_SSDIV_0670,                  ///< 0.67 <= Base clock frequency (MHz) < 0.80
        CTSU_SSDIV_0570,                  ///< 0.57 <= Base clock frequency (MHz) < 0.67
        CTSU_SSDIV_0500,                  ///< 0.50 <= Base clock frequency (MHz) < 0.57
        CTSU_SSDIV_0440,                  ///< 0.44 <= Base clock frequency (MHz) < 0.50
        CTSU_SSDIV_0400,                  ///< 0.40 <= Base clock frequency (MHz) < 0.44
        CTSU_SSDIV_0360,                  ///< 0.36 <= Base clock frequency (MHz) < 0.40
        CTSU_SSDIV_0330,                  ///< 0.33 <= Base clock frequency (MHz) < 0.36
        CTSU_SSDIV_0310,                  ///< 0.31 <= Base clock frequency (MHz) < 0.33
        CTSU_SSDIV_0290,                  ///< 0.29 <= Base clock frequency (MHz) < 0.31
        CTSU_SSDIV_0270,                  ///< 0.27 <= Base clock frequency (MHz) < 0.29
        CTSU_SSDIV_0000                   ///< 0.00 <= Base clock frequency (MHz) < 0.27
    } ctsu_ssdiv_t;

    /** Callback function parameter data */
    typedef struct st_ctsu_callback_args
    {
        ctsu_event_t event;               ///< The event can be used to identify what caused the callback.
        void const * p_context;           ///< Placeholder for user data. Set in ctsu_api_t::open function
in ::ctsu_cfg_t.
    } ctsu_callback_args_t;

    /** CTSU Control block. Allocate an instance specific control block to pass into the API calls.
     * @par Implemented as
     * - ctsu_instance_ctrl_t
     */
    typedef void ctsu_ctrl_t;

    /** CTSU Configuration parameters. */
    /** Element Configuration */
    typedef struct st_ctsu_element
```

```
{
    ctsu_ssdiv_t ssdiv;              ///< CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)
    uint16_t     so;                 ///< CTSU Sensor Offset Adjustment
    uint8_t      snum;               ///< CTSU Measurement Count Setting
    uint8_t      sdpa;               ///< CTSU Base Clock Setting
} ctsu_element_cfg_t;
```

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_INVALID_CHANNEL        = 4,                      ///< Selected channel does not exist
    FSP_ERR_INVALID_MODE           = 5,                      ///< Unsupported or incorrect mode
    FSP_ERR_UNSUPPORTED            = 6,                      ///< Selected mode not supported by this API
    FSP_ERR_NOT_OPEN               = 7,                      ///< Requested channel is not configured or API not open

    /* 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 IEC 60730 Compliance

This module complies with both R.1 (IEC 60335-1) and software class B (IEC 60730-1). For the latest information on the support status, refer to the web page Functional Safety Solutions for Home Appliances (IEC/UL 60730).

3. API Functions

3.1 R_CTSU_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.

Format

```
fsp_err_t R_CTSU_Open (ctsu_ctrl_t * const p_ctrl,
                        ctsu_cfg_t const * const p_cfg)
```

Parameters

p_ctrl	Pointer to the control structure (normally generated by QE for Capacitive Touch)
p_cfg	Pointer to the config structure (normally generated by QE for Capacitive Touch)

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 r_ctsu_api.h

Description

This function enables control structure initialization, register initialization, and interrupt setting according to the argument p_cfg.

Also, the correction coefficient generation process is executed while processing the first touch interface structure. The process takes approximately 120ms.

The DTC is initialized if CTSU_CFG_USE_DTC is enabled when the first touch interface configuration is processed.

Reentrant

This function is non-reentrant

Example

```
fsp_err_t err;

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

/* Initialize the API. */
err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_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

3.2 R_CTSU_ScanStart

This function starts measurement of the specified touch interface configuration.

Format

```
fsp_err_t R_CTSU_ScanStart (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl Pointer to the control structure (normally generated by QE for Capacitive Touch)

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 r_ctsu_api.h.

Description

When a software trigger occurs, this function sets and starts the measurement based on the touch interface configuration. With an external trigger, the function sets the measurement and goes to the trigger wait state.

If CTSU_CFG_USE_DTC is enabled, the function also sets the DTC.

The resulting value is notified in the callback generated from the INTCTSUFN interrupt handler.

Reentrant

This function is non-reentrant.

Example

```
fsp_err_t err;

/* Initiate a sensor scan by software trigger */
err = R_CTSU_ScanStart(&g_ctsu_ctrl);

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

Special Notes:

None

3.3 R_CTSU_DataGet

This function reads all the values previously measured in the specified touch interface configuration.

Format

```
fsp_err_t R_CTSU_DataGet (ctsu_ctrl_t * const p_ctrl, uint16_t * p_data)
```

Parameters

p_ctrl Pointer to the control structure (normally generated by QE for Capacitive Touch)

p_data Pointer to the buffer that stores the measured value.

Return Values

FSP_SUCCESS */* CTSU initialization successfully completed */*

FSP_ERR_ASSERTION */* Argument pointer not specified */*

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

FSP_ERR_CTSU_SCANNING */* scanning */*

FSP_ERR_CTSU_INCOMPLETE_TUNING */*Tuning initial offset */*

Properties

Prototype is declared in r_ctsu_api.h.

Description

This function reads all previously measured values into the specified buffer. The required buffer size varies depending on the measurement mode. Prepare twice the number of TS for the self-capacitance and current measurement modes, and twice the number of matrixes for the mutual-capacitance mode. If normalization (majority frequency) is turned off, prepare multiple CTSU_CFG_NUM_SUMULTI terminals for each mode. The value measured in the temperature correction mode is not stored. When RTRIM adjustment is performed, the RTRIM value is stored. At this time, the ADC settings have been changed in this function, so perform the process to return to the ADC settings you are using. Otherwise, store 0xFFFF.

When initial offset adjustment is on, FSP_ERR_INCOMPLETE_TUNING is returned several times until the adjustment is complete. Measured values are not stored in the buffer at this time. For more details on initial offset adjustment, refer to section 1.1.6.

The measured value is the value resulting from the sensor ICO correction, normalization (when on), and moving average processes executed in this function.

Reentrant

This function is non-reentrant.

Example:

```
fsp_err_t err;
uint16_t buf[CTSU_CFG_NUM_SELF_ELEMENTS];

/* Get all sensor values */
err = R_CTSU_DataGet(&g_ctsu_ctrl, buf);
```

Special Notes:

None

3.4 R_CTSU_CallbackSet

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

Format

```
fsp_err_t R_CTSU_CallbackSet (ctsu_ctrl_t * const p_api_ctrl,
                              void (* p_callback)(ctsu_callback_args_t *),
                              void const * const p_context,
                              ctsu_callback_args_t * const p_callback_memory)
```

Parameters

p_api_ctrl	Pointer to the control structure (normally generated by QE for Capacitive Touch)
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 r_ctsu_api.h.

Description

This function sets the function specified for the measurement completion callback function. By default, the callback function is set to the function of member p_callback of ctsu_cfg_t, so use it when you want to change to another function during operation.

You can also set the context pointer. If not used, set p_context to NULL. Set p_callback_memory to NULL.

Reentrant

This function is non-reentrant.

Example:

```
fsp_err_t err;

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

Special Notes:

None

3.5 R_CTSU_Close

This function closes the specified touch interface configuration.

Format

```
fsp_err_t R_CTSU_Close (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl Pointer to the control structure (normally generated by QE for Capacitive Touch)

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 r_ctsu_api.h.

Description

This function closes the specified touch interface configuration.

Reentrant

This function is non-reentrant.

Example:

```
fsp_err_t err;

/* Shut down peripheral and close driver */
err = R_CTSU_Close(&g_ctsu_ctrl);
```

Special Notes:

None

3.6 R_CTSU_Diagnosis

This is the API function providing the function for diagnosis of the CTSU inner circuit.

Format

```
fsp_err_t R_CTSU_Diagnosis (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl Pointer to the control structure (normally, generated by QE for Capacitive Touch)

Return Values

FSP_SUCCESS */* All diagnoses are normal */*
FSP_ERR_ASSERTION */* Missing argument pointer */*
FSP_ERR_NOT_OPEN */* Called without calling Open() */*
FSP_ERR_CTSU_NOT_GET_DATA */*Not processed previous scan data. */*
FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE */* Diagnosis of LDO over voltage failed */*
FSP_ERR_CTSU_DIAG_CCO_HIGH */* Diagnosis of CCO into 19.2uA failed. */*
FSP_ERR_CTSU_DIAG_CCO_LOW */* Diagnosis of CCO into 2.4uA failed.*/*
FSP_ERR_CTSU_DIAG_SSCG */* Diagnosis of SSCG frequency failed. */*
FSP_ERR_CTSU_DIAG_DAC */* Diagnosis of non-touch count value failed. */*
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE */*Diagnosis of LDO output voltage failed. */*
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE */*Diagnosis of over voltage detection circuit failed.*/*
FSP_ERR_CTSU_DIAG_OVER_CURRENT */*Diagnosis of over current detection circuit failed. */*
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE */*Diagnosis of LDO internal resistance value failed.*/*
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE */*Diagnosis of Current source value failed.*/*
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN */*Diagnosis of SENSCLK frequency gain failed.*/*
FSP_ERR_CTSU_DIAG_SUCLK_GAIN */*Diagnosis of SUCLK frequency gain failed.*
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY */*Diagnosis of SUCLK clock recovery function failed.*/*

Properties

Prototyped in file “r_ctsu_qe.h

Description

This is the API function providing the function for diagnosis of the CTSU inner circuit
Call when the return value of the function R_CTSU_DataGet is FSP_SUCCESS.

Reentrant

No.

Example:

```
fsp_err_t err;
uint16_t  dummy;

/* Open Diagnosis function */
R_CTSU_Open(g_qe_ctsu_instance_diagnosis.p_ctrl,
g_qe_ctsu_instance_diagnosis.p_cfg);

/* Scan Diagnosis function */
R_CTSU_ScanStart(g_qe_ctsu_instance_diagnosis.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;

err = R_CTSU_DataGet(g_qe_ctsu_instance_diagnosis.p_ctrl,&dummy);
if (FSP_SUCCESS == err)
{
    err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
    if ( FSP_SUCCESS  == err )
    {
       /* Diagnosis was succssed. */
    }
}
```

Special Notes:

3.7 R_CTSU_ScanStop

This function stops measuring the specified touch interface configuration.

Format

```
fsp_err_t R_CTSU_ScanStop (ctsu_ctrl_t * const p_ctrl)
```

Parameters

p_ctrl	Pointer to the control structure (normally, generated by QE for Capacitive Touch)

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 r_ctsu_api.h.

Description

This function stops measuring the specified touch interface configuration.

Reentrant

This function is non-reentrant.

Example:

```
fsp_err_t err;

/* Stop CTSU module */
err = R_CTSU_ScanStop(&g_ctsu_ctrl);
```

Special Notes:

None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.04.18	—	First edition issued
1.10	Jul.09.19	1	Added RX23W support.
		3-5	Added definitions for “correction” and “offset tuning”.
		9,12	Updated API return values.
		21-22	Added CTSU_CMD_GET_METHOD_MODE and CTSU_CMD_GET_SCAN_INFO Control() commands.
		8, 10-14	Added #pragma section macros and configuration option to driver for Safety Module support (includes GCC/IAR support).
		1,14	Added IEC 60730 Compliance section.
1.11	Jan.09.20	4,5	Added definition for “baseline” (Touch layer).
		26,27	Added CTSU_CMD_SNOOZE_ENABLE and CTSU_CMD_SNOOZE_DISABLE Control() commands.
		—	Fixed bug where a custom callback function was called twice after a scan completes.
		—	Fixed compile error for RX231 when PLL had multiplier of 13.5.
2.00	Jul.30.21	-	Full-fledged revision

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

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/.

Trademarks

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