# RX Family

## QE CTSU module Firmware Integration Technology

### Introduction

This application note describes CTSU module.

### Target Device

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

CTSU peripheral has five versions: CTSU, CTSUa, CTSU2L, CTSU2SL, and CTSU2SLa. Each MCU device is equipped with the following version of CTSU peripherals.

| | |
|---|---|
| CTSU2SLa | : RX260, RX261 |
| CTSU2SL | : RX140-256KB, RX140-128KB |
| CTSU2L | : RX140-64KB |
| CTSUa | : RX130, RX671 |
| CTSU | : RX113, RX230, RX231, RX23W |

Since there is no difference in the explanation of the difference in function in this document, both CTSU and CTSUa are referred to as CTSU. CTSU and CTSU2 have different functions, so these are described in this application note as below.

- Common description for CTSU, CTSU2L, CTSU2SL, CTSU2SLa -> CTSU
- Description only for CTSU -> CTSU1
- Common description for CTSU2L, CTSU2SL and CTSU2SLa -> CTSU2L
- Common description for CTSU2SL, CTSU2SLa -> CTSU2SL
- Description only for CTSU2SLa -> CTSU2SLa

To understand this document, it is recommended to refer to “Capacitive Sensor Microcontrollers CTSU Capacitive Touch Introduction Guide” and “Capacitive Touch Software Overview” beforehand.

When developing an application using the module, it is recommended to use QE for Capacitive Touch

### 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)
Capacitive Sensor Microcontrollers CTSU Capacitive Touch Introduction Guide (R30AN0424)
Capacitive Touch Sensor Microcontrollers Overview of Capacitive Touch Software (R30AN0470)

## Contents

# 1. Overview

## 1.1 What is CTSU module

CTSU module is a driver for controlling CTSU peripheral.

By using configuration settings, CTSU module performs measurements corresponding to the electrodes connected to TS terminals and obtains the capacitance measurement values. Main functions are as follows:

- It controls CTSU peripheral, applies several correction processes to the measurement values obtained from CTSU, and calculates the final corrected measurement values. In addition, CTSU2L peripheral can measure current instead of capacitance.
- It obtains the button touch judgement result from CTSU2SL peripheral.
- It diagnoses the internal circuits to confirm that CTSU peripheral is operating correctly.

## 1.2 Position of CTSU module

The software configuration of a touch system is shown in Figure 1.1.

Normally, CTSU module controls CTSU peripheral and returns corrected measurement values. Therefore, to obtain button ON/OFF status or slider/wheel touch positions in an application, CTSU module should be used in combination with TOUCH module.

In that case, TOUCH module calls CTSU module and converts the corrected measurement values obtained from CTSU module into button ON/OFF status and touch positions. For details of the specific operation, refer to the application note for TOUCH module.

If you want to perform your own touch judgement based on the corrected measurement values obtained from CTSU module, refer to the subsequent chapters.

Furthermore, by combining the hardware judgement implemented in CTSU2SL peripheral (auto-judgement mode) with CTSU module, you can obtain touch ON/OFF results using CTSU module.

CTSU module can also be used to diagnose the internal circuits of CTSU peripheral and obtain the diagnosis results.

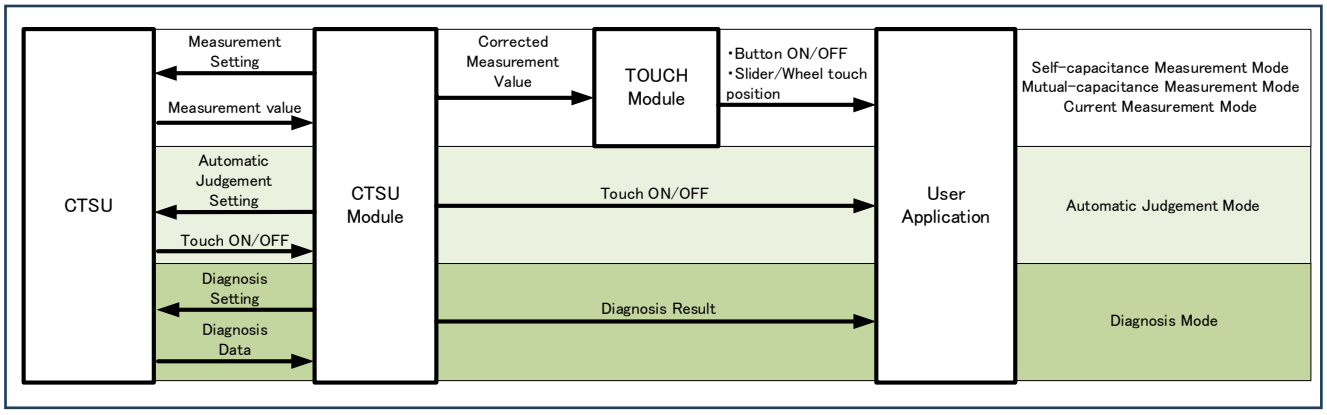


**Figure 1.1 Relationship between CTSU module and user application**

## 1.3 How to use CTSU module

Figure 1.2 shows an overview of the software system that illustrates how to use CTSU module.

CTSU module defines measurement modes using a combination of API call groups and the associated structure information. The API call groups used in self-capacitance / mutual-capacitance / current-measurement modes are treated as the basic operation model. For the operation models of auto-judgement and diagnosis modes, refer to Section 1.4.

In the basic operation model, you first call R_CTSU_Open() to initialize CTSU peripheral and to read the configuration settings and create the configuration information in the control structure. After that, by periodically calling R_CTSU_ScanStart() (measurement start processing) and R_CTSU_DataGet() (measurement result acquisition processing), you can perform periodic measurement processing.

During measurement, the following interrupts occur: INTCTSUWR (register setting request interrupt), INTCTSURD (measurement result readout request interrupt), and INTCTSUFN (measurement completion interrupt). These interrupts are generated by CTSU peripheral and processed by the module. After processing of INTCTSUFN completes, the module notifies the application of measurement completion via the callback function. After receiving the notification, perform the measurement result acquisition processing.

Note that INTCTSUWR and INTCTSURD processing can also use DTC. For details on using DTC, refer to Section 1.6.3.

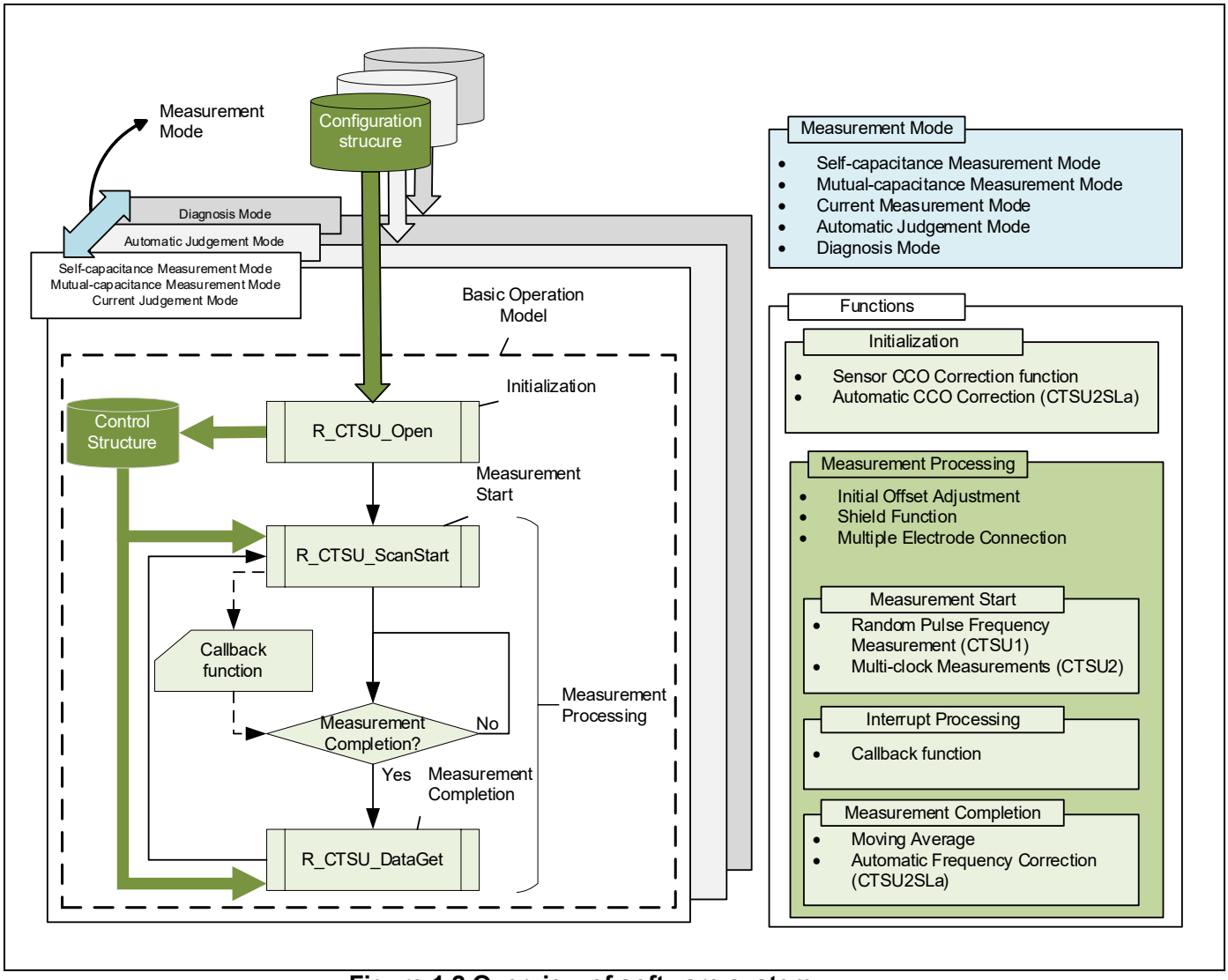


**Figure 1.2 Overview of software system**

In this module, data management is structured around two key data structures: the configuration structure and the control structure.

The configuration structure is used to set the measurement targets and measurement conditions. The target of a capacitance measurement is referred to as an element. In self-capacitance mode, TS terminal serves as the measurement target, while in mutual-capacitance measurement mode, the targets are the matrix of transmitter TS terminals and receiver TS terminals. The measurement conditions include various parameters required for executing measurements using CTSU peripheral, such as measurement mode, measurement time, measurement range, and trigger settings.

The measurement information defined in the configuration structure is copied into the control structure during the execution of R_CTSU_Open(). At the same time, buffers for storing measurement results are allocated, and internal flags and state information for managing measurement states are initialized. After initialization, only the control structure is referenced during the measurement process, enabling unified management of measurement settings, measurement states, and measurement results.

By preparing multiple sets of these structures, it becomes possible to run different measurement modes independently and in parallel, or to operate multiple measurement processes independently and in parallel even within the same measurement mode, provided the measurement targets and measurement conditions differ.

To simplify the configuration of these measurement settings, Renesas provides a development support tool called QE for Capacitive Touch. When creating applications that use this driver, it is recommended to use the touch interface configuration generated by QE for Capacitive Touch.

## 1.4 Measurement Mode

The term measurement mode collectively refers to the measurement processing determined by the measurement target, the measurement method, and the associated conditions. The available measurement modes differ depending on the functions of CTSU peripheral.

CTSU peripheral supports Self-capacitance Measurement Mode, mutual-capacitance measurement mode, and diagnosis mode.

In addition to these, CTSU2L peripheral supports current-measurement mode, and CTSU2SL peripheral supports auto-judgement mode.

Details of each measurement mode are described in Sections 1.4.1 and onward.

### 1.4.1 Self-capacitance Measurement Mode

Self-capacitance Measurement Mode is used to measure the capacitance of each terminal (TS).

CTSU peripheral measures the terminals in ascending order according to 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.

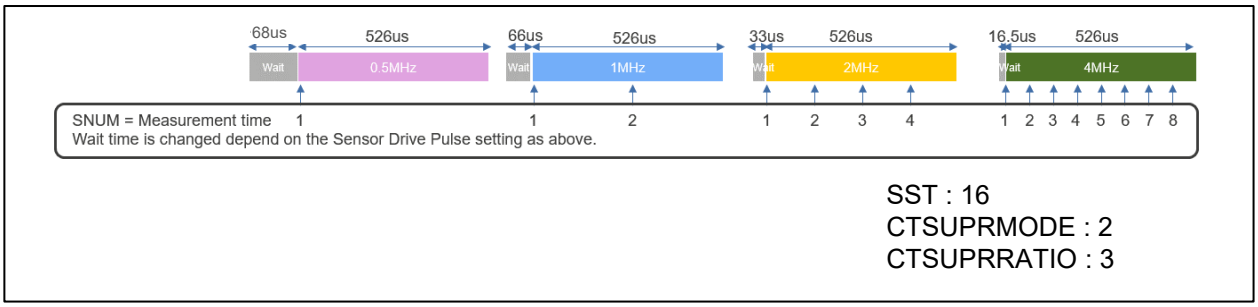


**Figure 1.3 Self-capacitance Measurement Period (CTSU1)**

[CTSU2L]

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

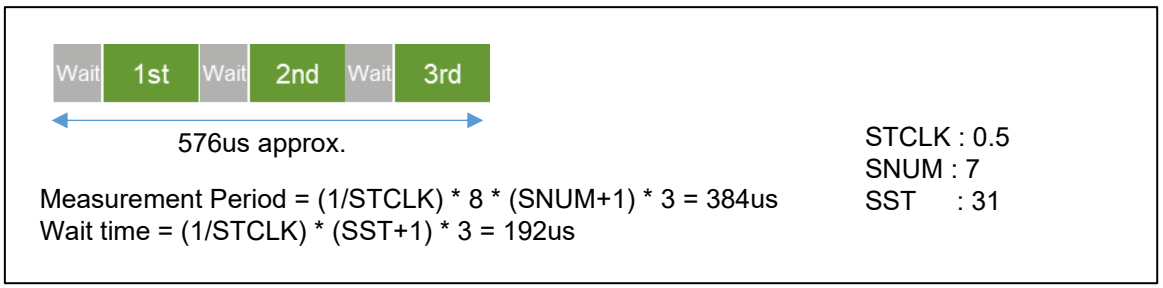


**Figure 1.4 Self-capacitance Measurement Period (CTSU2L)**

### 1.4.2 Mutual-capacitance Measurement Mode

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

CTSU2L peripheral measures all specified combinations of Rx and Tx. For example, when Rx is TS10 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 measurement generated between electrodes, CTSU2L peripheral performs the measurement process on the same electrode twice.

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

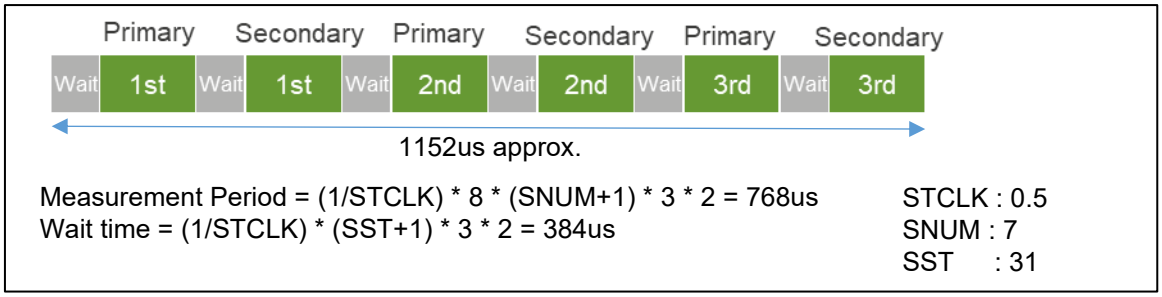


**Figure 1.5 Mutual-capacitance Measurement Period (CTSU2L)**

### 1.4.3 Current Measurement Mode (CTSU2L)

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

The order of measurement and data storage is the same as that of 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 SST is set to 63.

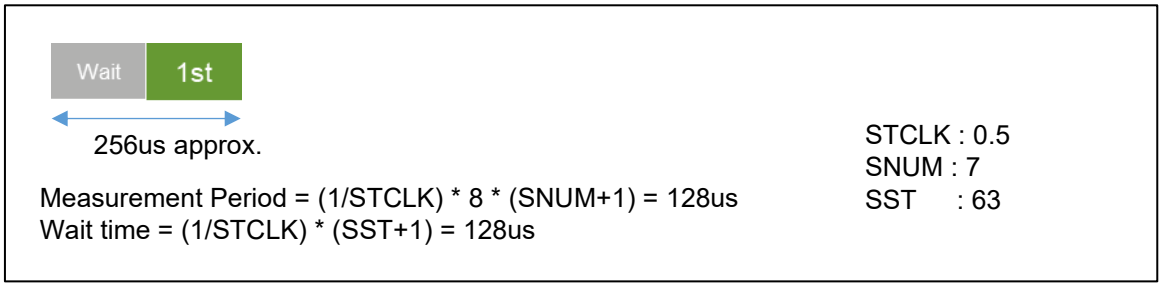


**Figure 1.6 Current Measurement Period**

### 1.4.4 Automatic Judgement Mode (CTSU2SL)

In automatic judgement mode, CTSU2SL peripheral is controlled to obtain touch-judgement results. Since touch-judgement processing is performed by CTSU2SL peripheral, touch judgement can be executed without using CPU processing. The DTC module is mandatory because DTC transfer are necessary to process the interrupts that occur during measurement. For details on DTC, refer to Section 1.6.3.

The operation flow of automatic judgement mode is shown below. First, initial offset tuning is performed using R_CTSU_ScanStart() and R_CTSU_OffsetTuning(). For details on initial offset tuning, refer to Section 1.5.4. After initial offset tuning completes, automatic judgement measurement processing is performed using R_CTSU_ScanStart() and R_CTSU_AutoJudgementDataGet(). Note that the processing for the first measurement is different from that for the second and subsequent measurements:

First measurement:

The baseline for touch judgement is set. The first corrected measurement value is used as the baseline, and the touch judgement result is OFF.

Second and subsequent measurements:

Measurements are performed according to the measurement settings in the control structure, and touch judgement results are obtained.

By combining automatic judgement mode with measurement started by external triggers, you can create applications that perform touch judgement with low-power consumption. For more details and notes in that case, refer to 1.4.4(1). If you need more details on auto-judgement, refer to 1.4.4(2).

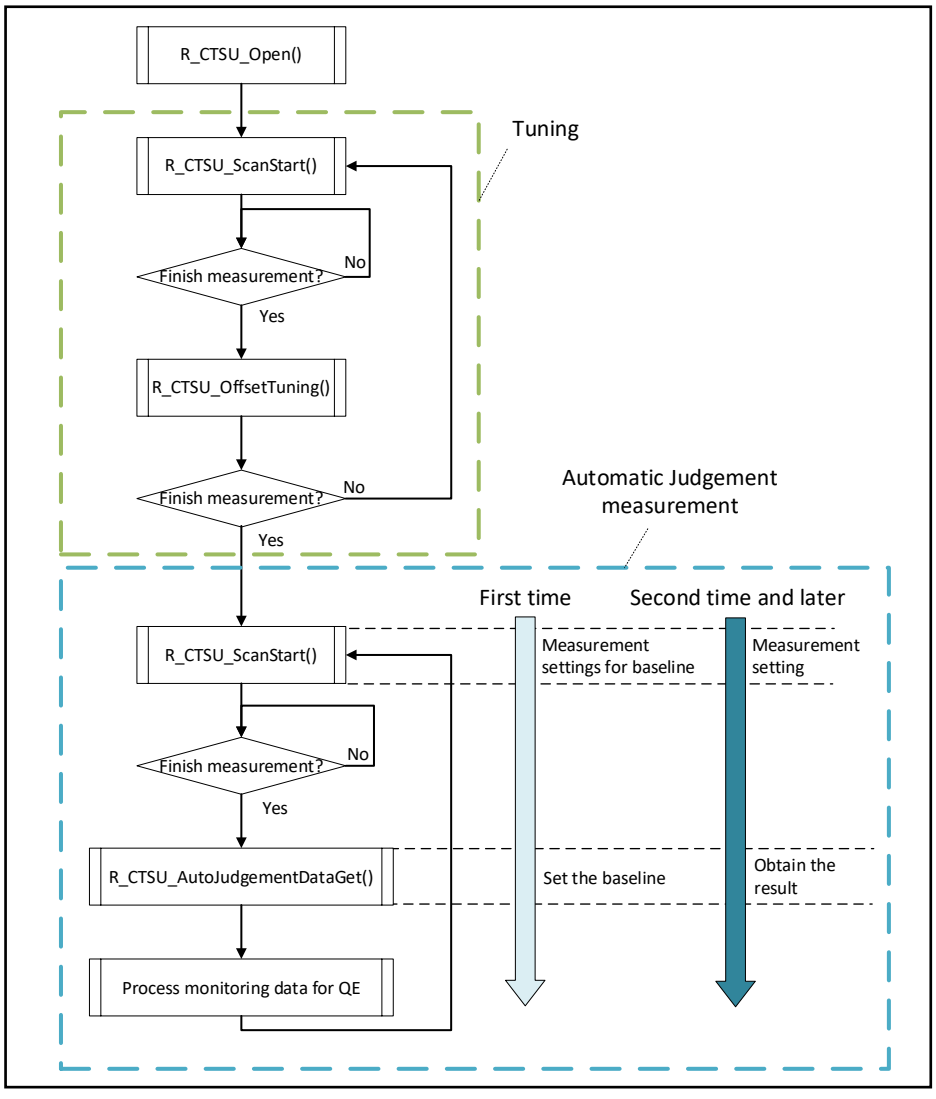


**Figure 1.7 Basic operation model of auto-judgement mode**

(1) Notes for low-power touch judgement

In low-power touch applications that combine CPU Snooze mode or Software Standby mode with automatic judgement mode, it is possible to keep CPU in a low-power state while performing touch judgement and wake up CPU only when a touch judgement result is ON.
However, CPU processing is required until the baseline is set. Therefore, switch to the low-power operation mode after executing the first R_CTSU_ScanStart() and R_CTSU_AutoJudgementDataGet().
For concrete operation examples, refer to “Smart Wakeup Solution” application note.

(2) Functions and settings

When using the automatic judgement mode, in addition to the standard measurement settings, it is necessary to modify certain macro definitions and configure the configuration structure for automatic judgement mode. These settings differ depending on the automatic judgement method, JMM or VMM.

First, to enable the automatic judgement feature, set CTSU_CFG_AUTO_JUDGE_ENABLE = 1. At the same time, enable the automatic CCO correction function by setting CTSU_CFG_AUTO_CORRECTION_ENABLE = 1. For VMM, also enable the automatic frequency correction function by setting CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE = 1.

Next, an example of configuration structure settings required in addition to the standard measurement settings is shown below for JMM. When using VMM, set jc = 0 and majirimd = 1.

```
    .tlot = 2,     // Non-touch  judgment continuous count : 3 times
    .thot = 2,     // Touch judgment continuous count : 3 times
    .jc = 1,       // Judgement by two or more frequency
    .ajmmat = 2,   // Moving average : 2^2times
    .ajbmat = 7,   // Baseline average count : 2^(7+1)times
    .majirimd = 0, // JMM
    .mtucfen = 1,  // Enable mutual-capacitance calculation
    .ajfen = 1,    // Enable automatic judgement
```

The following (a) to (e) describe the automatic judgment and its setting. In the case of JMM, (a) ~ (e) settings are set for each multi-clock measurement.

(a) Measurement mode

Select self-capacitance or mutual-capacitance measurement with “mtucfen” of ctsu_auto_button_cfg_t. Set the self-capacitance to 0. Set the Mutual-capacitance to 1.

(b) Baseline

Set the baseline from the corrected measurement result in the non-touch state. After completing the initial offset adjustment with R_CTSU_OffsetTuning (), the baseline is initially set (set BLINI bit) when R_CTSU_ScanStart () is called for the first time. After that, when R_CTSU_AutoJudgementDataGet () is called, the baseline initialization is canceled (clear BLINI bit) and the baseline update process is started.

The baseline is updated every set number of measurements to follow changes in the surrounding environment. If “non-touch” state continues for the set number of measurements, the baseline is updated to the average value. When judgement result is “touch”, the number of counts is cleared.

Set the number of measurements (baseline update interval) with “ajbmat” of ctsu_cfg_t. Common to all buttons in the touch interface configuration. Adjusts the ability to follow changes in the surrounding environment.

(c) Touch threshold

Judgment is made using a threshold with an arbitrary offset from the baseline.

The threshold is set by adding hysteresis. Chattering is prevented by giving hysteresis to the transition from “touch” to “non-touch”. Increasing the hysteresis value is more effective in preventing chattering, but be aware that it will be more difficult to transition from “touch” to “non-touch”.

Set the threshold and hysteresis for each button with threshold and hysteresis of ctsu_auto_button_cfg_t. This module calculates the upper threshold and the lower threshold from these and sets them in CTSUAJTHR register.

Figure 1.8 shows the self-capacitance judgement. Since the electrode capacitance of the self-capacitance button increases when touched, it is judged “touch” when the upper threshold is exceeded.

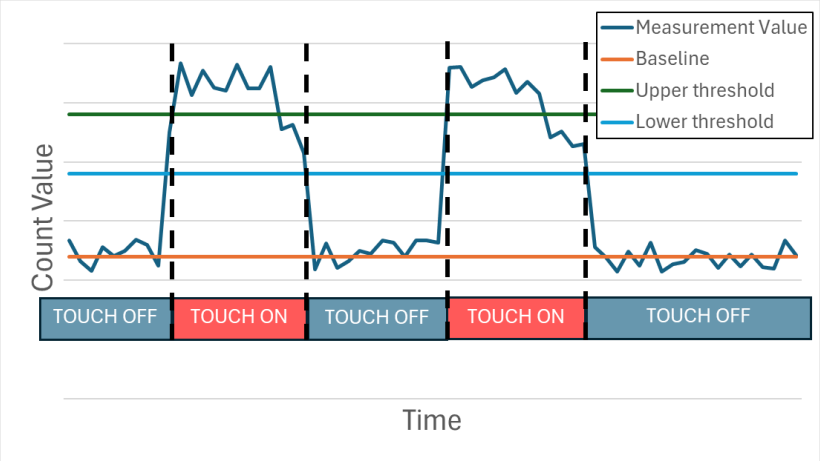


**Figure 1.8 Self-capacitance judgement**

Figure 1.9 shows the mutual-capacitance judgement. Since the mutual-capacitance button reduces the capacitance between electrodes when touched, it is judged as "touch" when the lower threshold is exceeded.

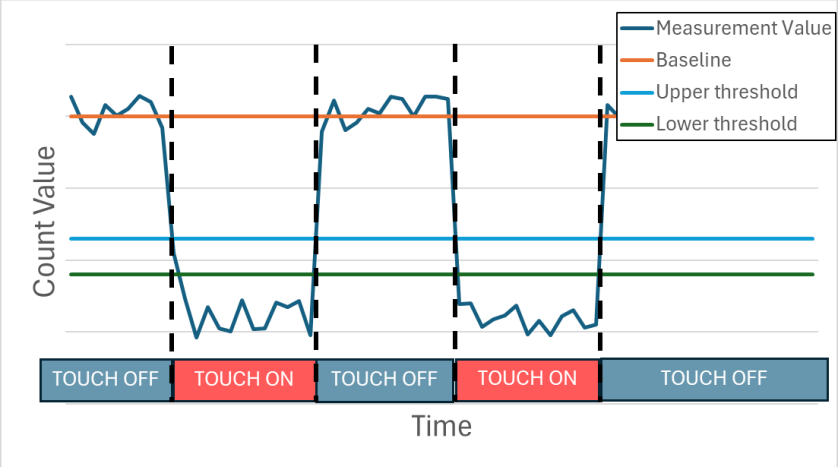


**Figure 1.9 Mutual-capacitance judgement**

(d) The number of consecutive “non-touch” and “touch” detections

This is a filter function to judge"touch" or "non-touch" when "touch" or "non-touch" state continues for a certain number of times.

Set the number of times with “tlot” and “thot” of ctsu_cfg_t. Common to all buttons in the touch interface configuration. Increasing the number of consecutive times will be more effective against chattering, but be aware that the reaction speed will decrease.

(e) Moving average

With the automatic judgment function, Set the number of moving averages with “ajmmat” of ctsu_cfg_t. Common to all buttons in the touch interface configuration.

Figure 1.10 shows the button judgment operation described above.

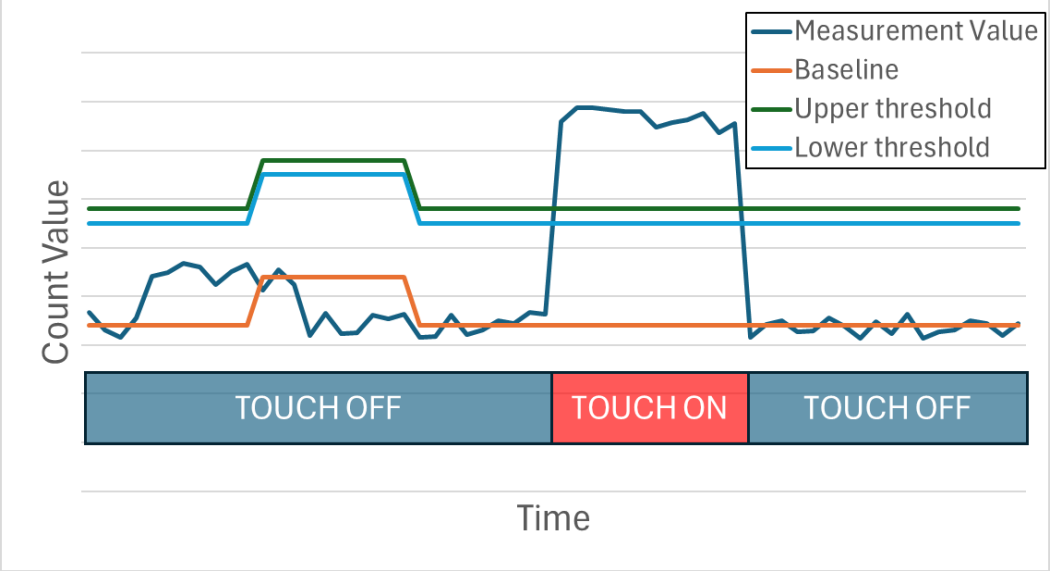


**Figure 1.10 Button judgement**

### 1.4.5 Diagnosis Mode

CTSU peripheral has a function to diagnose its internal circuits. Diagnosis mode provides APIs that diagnose whether the internal circuits are operating correctly.

In this mode, CTSU internal circuits are diagnosed and, if an abnormality occurs, the result can be obtained as an error.

The diagnosis contents differ between CTSU1 and CTSU2, and therefore the operation model (API call sequence) also differs.

To enable diagnosis mode, set CTSU_CFG_DIAG_SUPPORT_ENABLE = 1.

(1) CTSU1

(a) Diagnosis items

The diagnosis items of CTSU1 are shown in Table 1.1.

**Table 1.1 Diagnosis items (CTSU1)**

| Order | Diagnosis items |
|---|---|
| 1 | Over Voltage Detection Diagnosis |
| 2 | CCO High Diagnosis |
| 3 | CCO Low Diagnosis |
| 4 | SSCG Oscillator Diagnosis |
| 5 | Current Offset Diagnosis |

(b) Use case

For CTSU1, measurement and measurement-data processing are performed for a diagnosis instance using the basic model. If the return value of R_CTSU_DataGet() indicates normal completion, the diagnosis API is called to perform diagnosis processing, as shown in Figure 1.11.

If you want to use other measurement modes after diagnosis mode, start touch measurements after a wait time of about 1 ms following diagnosis mode measurement.

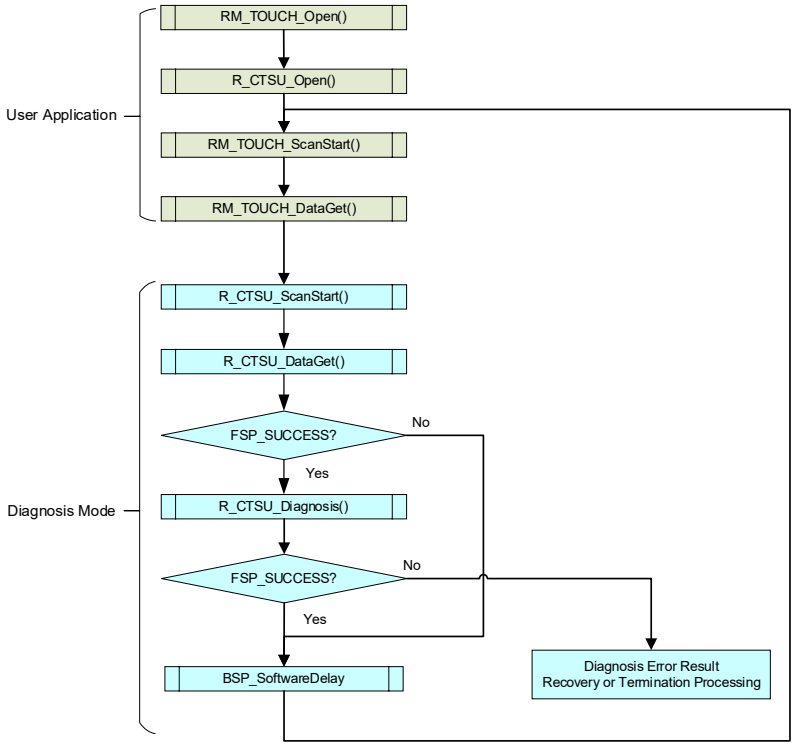


**Figure 1.11 Example flow when using diagnosis mode (CTSU1)**

(c) Configuration settings

In the configuration structure for this mode, set the member md to CTSU_MODE_DIAGNOSIS_SCAN.

Also connect a capacitor to any TS terminal (27 pF recommended) and assign TS terminal number you connected to the macro definition CTSU_CFG_DIAG_DAC_TS.

(2) CTSU2L

(a) Diagnosis items

The diagnosis items of CTSU2L are shown in Table 1.2.

**Table 1.2 Diagnosis items (CTSU2L)**

| Order | Diagnosis items | Onetime / Repeat (※1) |
|---|---|---|
| 1 | Output Voltage Diagnosis | Repeat |
| 2 | Over Voltage Detection Diagnosis | One-time |
| 3 | Over Current Detection Diagnosis | One-time |
| 4 | Load Resistance Diagnosis | Repeat |
| 5 | Current Offset Diagnosis | One-time |
| 6 | SENSCLK Frequency Diagnosis | Repeat |
| 7 | SUCLK Frequency Diagnosis | Repeat |
| 8 | SUCLK Clock Recovery Diagnosis | Repeat |

※1 “One-time” assumes diagnosis is performed once after reset. “Repeat” assumes periodic measurements.

(b) Use case

In CTSU2L diagnosis mode, the diagnosis is typically executed once at system startup or reset, before starting any other measurement modes. Even while other measurement modes are in use, you can periodically diagnose internal circuits by using this mode.

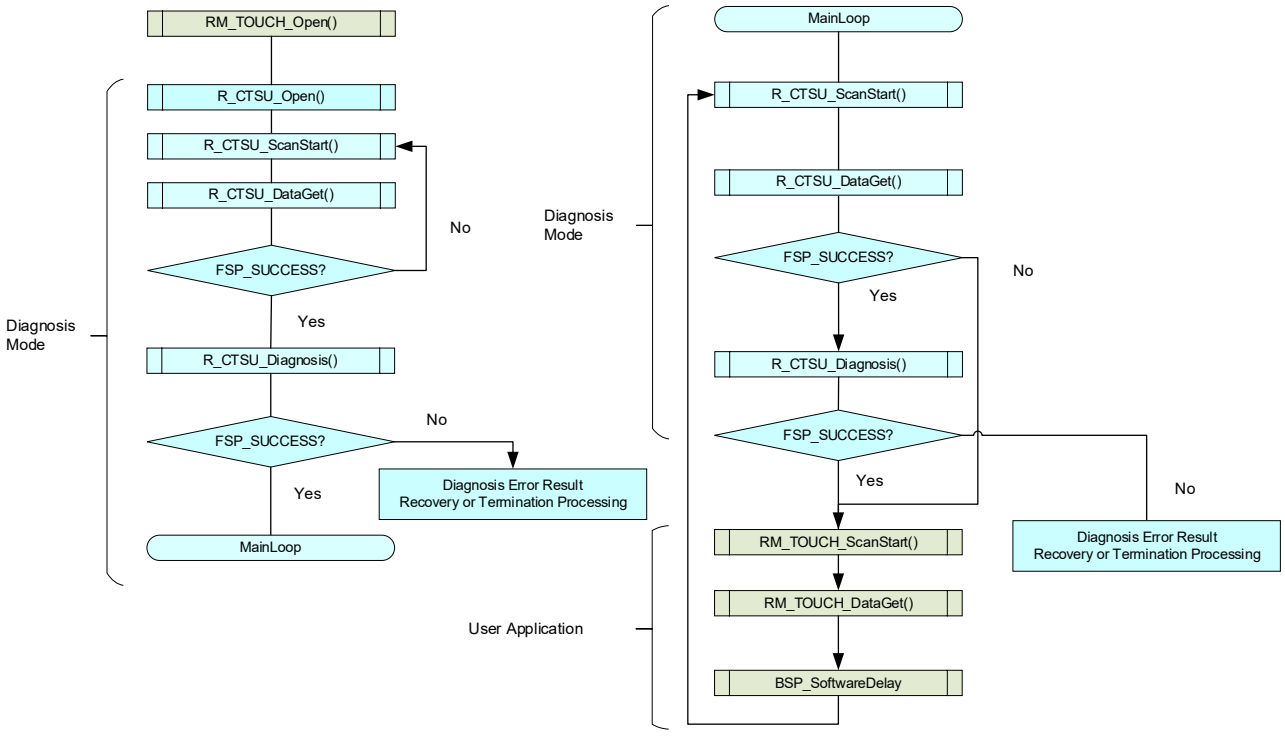


**Figure 1.12 Example flow when using diagnosis mode (CTSU2)**

(c) Configuration Settings

The required settings for the members of the diagnosis mode configuration structure are shown below.

These settings are automatically generated when using QE for Capacitive Touch. If not using QE for Capacitive Touch, refer to section 1.4.5(2)(d).

**Table 1.3 Configuration Settings for Diagnosis Mode (CTSU2L)**

| Member Name | Setting |
|---|---|
| ctsu_element_cfg_t | |
| so | 0x00 |
| snum | 0x07 |
| sdpa | Measurement setting for overcurrent detection |
| ctsu_cfg_t | |
| md | CTSU_MODE_DIAGNOSIS_SCAN |
| ctsuchacN (N=0～4) | Measurement terminal for overcurrent detection |

(d) Notes on configuration

In diagnosis mode, to verify overcurrent-detection operation, one TS terminal is selected from TS terminals used for normal measurement, and that TS terminal is set to ctsuchacN in the configuration structure. After this setting, the sensor-drive pulse frequency is adjusted according to the procedure below so that a current level that enables overcurrent detection is generated inside CTSU2SL peripheral connected to TS terminal.

1. Calculate the current flowing in TS terminal

   A measurement is performed for the selected TS terminal with an arbitrary sensor-drive pulse frequency $f$ and TSCAP voltage $V$, and the parasitic capacitance $C$ of the terminal is obtained.

   The current $I$ flowing inside CTSU2SL peripheral is calculated using the following formula:

$$I = fCV$$

2. Set SDPA for diagnosis

   The sensor-drive pulse is configured to meet the condition in which overcurrent can be detected.

   Let $SDPA_0$ be SDPA value used during parasitic-capacitance measurement and $I_0$ be the current calculated in step 1.

   Using these values, the diagnosis SDPA ($SDPA_1$) is determined as follows:

$$SDPA_1 = \frac{(SDPA_0 + 1)I_0}{100} - 1$$

Make sure that the sensor-drive pulse frequency set by $SDPA_1$ provides enough time to charge and discharge the target TS terminal.

## 1.5 Functions

CTSU module supports the following functions.

### 1.5.1 Random Pulse Frequency Measurement (CTSU1)

The drive pulse used for the measurement is a pulse with phase shifting and frequency spreading applied to the configured base clock.

The base-clock setting generally uses the value tuned with QE for Capacitive Touch.

This module is fixed at initialization and sets the following.

CTSUSOFF = 0, CTSUSSMOD = 0, CTSUSSCNT = 3

The base clock 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 base clock is 0.5MHz. CTSUSDPA can change for each TS port.

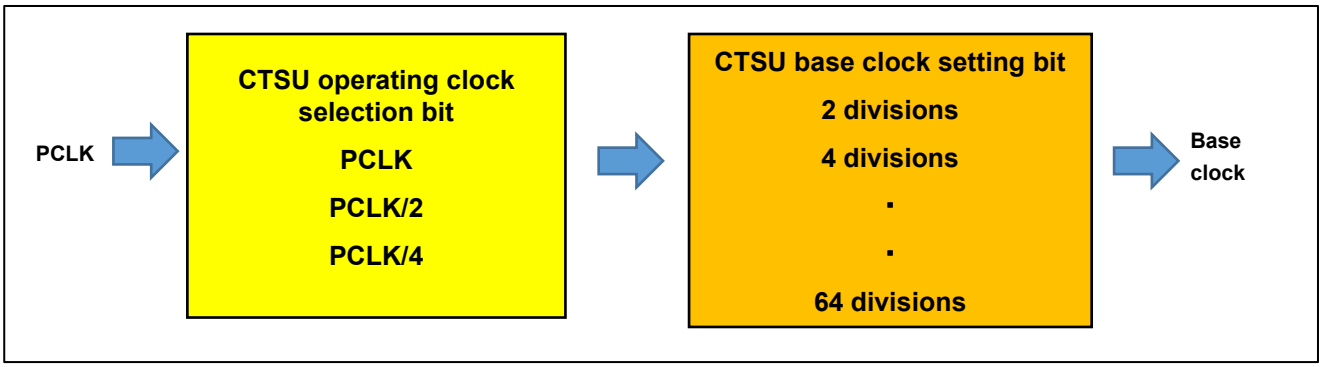


**Figure 1.13 Base clock settings**

### 1.5.2 Multi-clock Measurements (CTSU2L)

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

By default, this module measures at three different frequencies and makes a majority judgement on the three measurement results obtained.

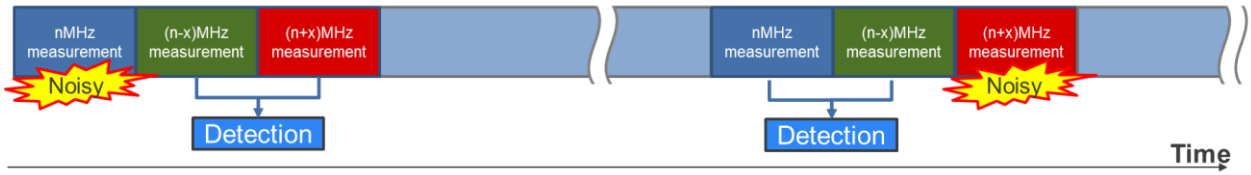


**Figure 1.14 Multi-clock Measurements**

There are two types of majority judgement modes for the three measurement results: JMM (Judgement Majority Mode) and VMM (Value Majority Mode). JMM only supports self-capacitance buttons and mutual-capacitance buttons.

Figure 1.15 shows the flowchart of JMM and VMM with Touch module.

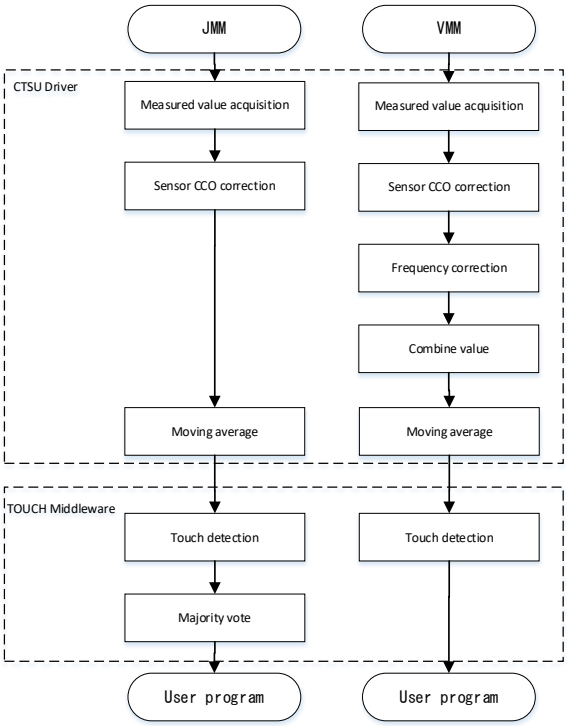


**Figure 1.15 Flowchart of JMM and VMM**

JMM performs CCO correction on the three measurement values, then makes a touch judgment for each of them, and determines the final touch result by majority vote of the three judgments.

VMM performs CCO correction on the three measurement values, applies frequency correction to convert them to the measurement value at the first frequency, and then selects the two values that are close to each other. These two values are added together, and the result is treated as a measurement value with double the measurement time. Touch judgment is made based on this corrected measurement value.

**Example VMM Calculations**

From the frequency-corrected measurement values 1, 2, and 3, the difference values 1, 2, and 3 for each pair are calculated, and the smaller pair is selected by comparing the absolute values of the difference values. To prevent variation in the corrected measurement values, a combination of value 1 and value 2 is given a weight to be selected. When comparing value 3, multiply the difference value 2 by 2 and multiply the difference value 3 by 1.5.

| Corrected Measurement Value 1 | Corrected Measurement Value 2 | Corrected Measurement Value 3 | Difference value 1 | Difference value 2 | Difference value 3 | Result | Added Value |
|---|---|---|---|---|---|---|---|
| 7734 | 7734 | 7663 | 0 | 71 | 71 | Value 1+2 | 15468 |
| 7689 | 7739 | 7666 | 50 | 23 | 73 | Value 1+3 | 15355 |
| 7734 | 7679 | 7664 | 55 | 70 | 15 | Value 2+3 | 15343 |
| 7721 | 7719 | 7694 | 2 | 27 | 25 | Value 1+2 | 15440 |
| 7716 | 7747 | 7693 | 31 | 23 | 54 | Value 1+2 | 15463 |

You can set JMM or VMM for each touch interface configuration. If the ctsu_cfg_t member "majority_mode" is set to 1, it works in JMM, and if it is set to 0, it works in VMM.

R_CTSU_DataGet () can get the data after conducting the moving average. To retrieve the data for each of the previous processes R_CTSU_SpecificDataGet use (). These data can also be used to determine the data with its own noise filter in Touch module. See Chapters 3.8 and 3.9 for more information.

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

Drive pulse frequency is calculated in the following equation:

$$f_{DrivePulse} = \frac{F_{PCLKB}}{2^{\mathrm{CLK}} \times 2(STCLK + 1)} \frac{\mathrm{SUMULTIn} + 1}{2(\mathrm{SDPA} + 1)},\ \mathrm{n} = 0, 1, 2$$

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

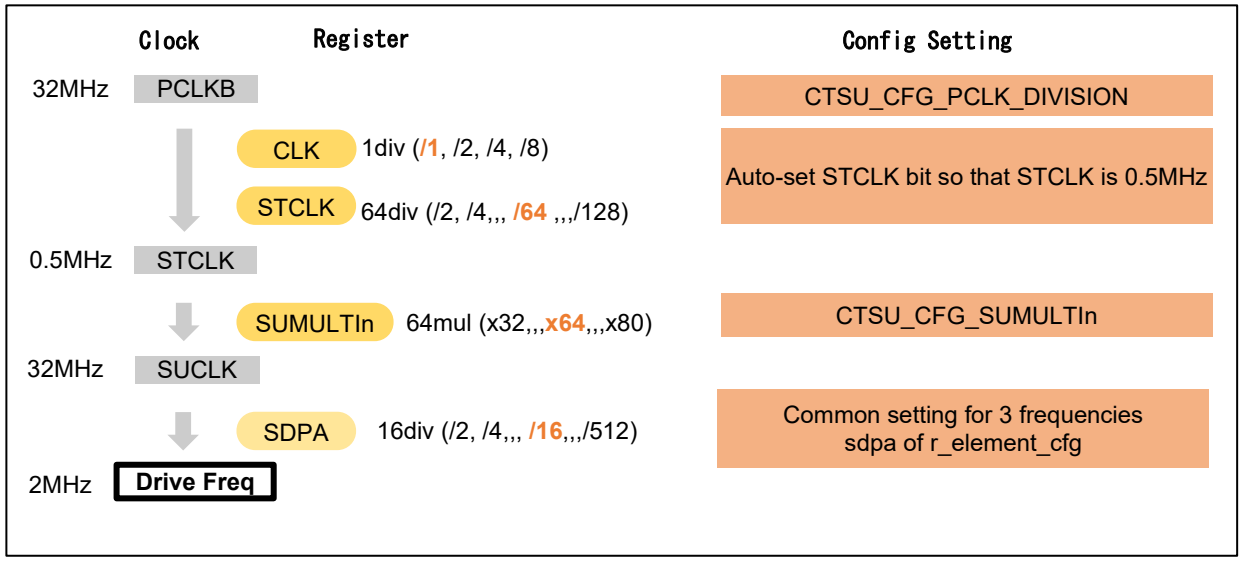


**Figure 1.16 Drive Pulse Frequency Settings**

### 1.5.3 Sensor CCO Correction function

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

This module uses the correction circuit during initialization after power-on to generate a correction coefficient to ensure accurate sensor measurement values. This correction coefficient is used to correct the measurement value.

### 1.5.4 Initial Offset Adjustment

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 CCO dynamic range.

This module adjusts the offset current setting so that the corrected measurement value reaches the target value. 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.

**Default target value (CTSU)**

| Mode | CTSU1 target value | CTSU2L target value |
|---|---|---|
| Self-capacitance | 15360 (37.5%) | 11520 (37.5%) |
| Self-capacitance using active shield | - | 4608 (15%) |
| Mutual-capacitance | 10240 (25%) | 7680 (25%) |

The percentage is based on 100% being the maximum input current applied to CCO.
CTSU1 : 100% is the measured value 40960 when the measurement time is 526us(base time).
CTSU2L : 100% is the measured value 30720 when the measurement time is 256us(base time).
When the measurement time is changed, the target value is adjusted by the ratio with the base time.

**Example of target value in combination of CTSUSNUM and CTSUSDPA**

CTSU1 (CTSU clock = 32MHz、Self-capacitance Measurement Mode)

| Target value | CTSUSNUM | CTSUSDPA | Measurement time |
|---|---|---|---|
| 15360 | 0x3 | 0x7 | 526us |
| 30720 | 0x7 | 0x7 | 1052us |
| 30720 | 0x3 | 0xF | 1052us |
| 7680 | 0x1 | 0x7 | 263us |
| 7680 | 0x3 | 0x3 | 263us |

The measurement time changes depending on the combination of CTSUSNUM and CTSUSDPA. Recommended CTSUPRRTIO, CTSUPRMODE are used. Changing this value is deprecated. For details, refer to the hardware manual of each capacitive touch sensor.

・CTSU2L (Self-capacitance Measurement Mode)

| Target value | Target value (multi-clock) | CTSUSNUM | Measurement time |
|---|---|---|---|
| 5760 | 11520 (128us + 128us) | 0x7 | 128us |
| 11520 | 23040 (256us + 256us) | 0xF | 256us |
| 2880 | 5760 (64us + 64us) | 0x3 | 64us |

The measurement time changes depending on CTSUSNUM. If STCLK cannot be set to 0.5MHz, it will not support the table above. Regarding STCLK, refer to the hardware manual.

### 1.5.5 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.

### 1.5.6 Callback function

When an abnormal condition is detected during measurement, the CTSU peripheral sets the corresponding status register. In the measurement-completion interrupt handler, the module reads CTSUSOVF from the status register and CTSUICOMP from the error status register for CTSU1, and reads ICOMP1, ICOMP0, and SENSOVF from the status register for CTSU2L. After the registers are read, the status registers are cleared. The details of the abnormal condition can be checked by referring to the event member of the ctsu_callback_args_t structure passed to the callback function.

### 1.5.7 Shield Function (CTSU2L)

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 Measurement.

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

For example, for the electrode configuration shown in Figure 1.17, 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,
```

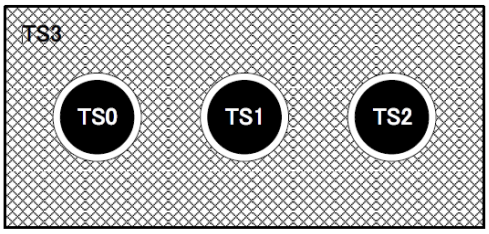


**Figure 1.17 Example of Shield Electrode Structure**

### 1.5.8 MEC Function (CTSU2SL)

CTSU2SL peripheral has MEC (Multiple Electrode Connection) function that connects multiple electrodes and measures them as a single electrode. This feature is only available in Self-capacitance Measurement Mode.

This is an example when using three electrodes. In normal times, normal measurement is performed, and 3 channels are measured to get each corrected measured value. In power saving, MEC measurement is performed, and one channel is measured by combining three channels to acquire one corrected measured value.

Figure 1.18 shows a compare of time of normal measurement and MEC measurement. Since multi channels are measured at the same time, the measurement time is shortened.

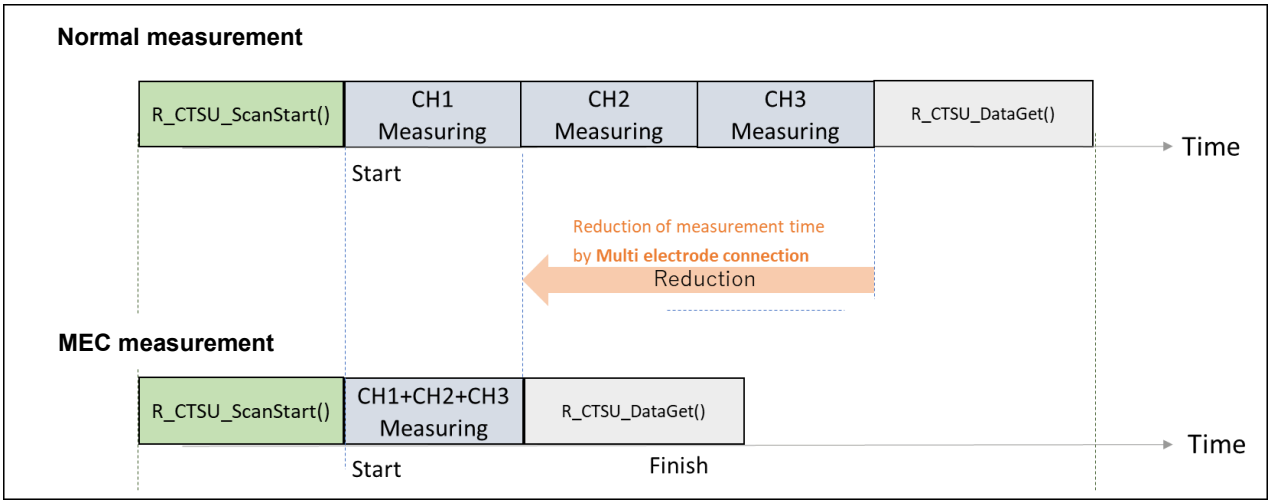


**Figure 1.18 Compare of time between normal measurement and MEC measurement**

To enable the code for MEC feature, set CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE to 1.

When using MEC, create a touch interface configuration different from the normal touch interface configuration for the same TS. The following settings are required for the touch interface configuration for MEC measurement.

To enable MEC for touch interface configurations by setting tsod in ctsu_cfg_t to 1.

Set mec_ts of ctsu_cfg_t to one of TS numbers to be measured.

If you want to use the shield function at the same time, set TS number of the shield terminal in mec_shield_ts of ctsu_cfg_t. In this case, only one TS can be used as a shield terminal.

Set num_rx of ctsu_cfg_t to 1.

For example, in the case of the electrode configuration shown in Figure 1.19, set the members of ctsu_cfg_t as shown below. Other members are omitted here.

```
    .tsod = 1,
    .mec_ts = 0,
    .mec_shield_ts = 3,
    .num_rx = 1,
```

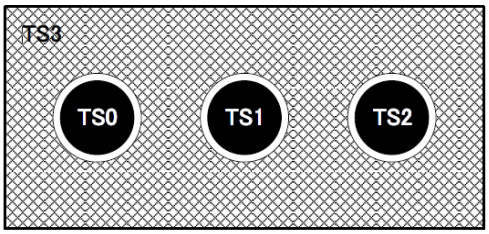


**Figure 1.19 Example of MEC and shield electrode configuration**

### 1.5.9 Automatic CCO Correction (CTSU2SL)

The automatic CCO correction function is a feature that performs sensor CCO correction calculations inside CTSU2SL peripheral. For details on sensor CCO correction, refer to section 1.5.3.

Because CTSU2SL peripheral processes the sensor CCO correction calculation internally, the corrected data can be obtained without using software-based correction processing, and no processing time is consumed on the main processor.

To enable the automatic CCO correction function, set CTSU_CFG_AUTO_CORRECTION_ENABLE to 1.

### 1.5.10 Automatic Frequency Correction (CTSU2SLa)

The automatic frequency-correction function is a feature that performs multi-clock correction inside CTSU2SLa peripheral.

Because CTSU2SLa peripheral processes the multi-clock correction calculation internally, the corrected data can be obtained without using software-based correction processing, and no processing time is consumed on the main processor.

To enable the automatic frequency correction function, set CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE to 1.

When using the automatic frequency-correction function, also enable the automatic CCO correction function.

## 1.6 Cooperation with other FIT modules

Some functions of CTSU module require cooperation with other FIT modules.

Table 1.4 FIT modules used in cooperation

| FIT Module | Application | Integration Method |
|---|---|---|
| Timer (CMT, LPT, etc.) | Start measurement via external trigger | Add FIT module for the timer used (such as CMT). Configure it in the application as appropriate. |
| ELC | Start measurement via external trigger | Add ELC FIT module. Configure it in the application according to the timer used. |
| LPC | Low-power touch operation | Add LPC FIT module. |
| DTC | Data transfer for INTCTSUWR and INTCTSURD during measurement | Add DTC FIT module. CTSU module already uses DTC FIT APIs internally. |
| S12ADC | Diagnose Output Voltage in CTSU2L diagnosis function | Add S12ADC FIT module. CTSU module already uses r_s12ad_rx APIs internally. |

### 1.6.1 Trigger for Measurement Start

Measurement can be started by either a software trigger or an external trigger. Select according to your system requirements.

Software trigger

Measurement starts when R_CTSU_ScanStart() sets the measurement-start bit to 1. Since configuration for the touch interface is performed before setting this bit, there is a small delay between the API call and the actual start of measurement. However, this delay is very small compared with the measurement time, so the software trigger, which is simpler to configure, is generally recommended.

External trigger

After R_CTSU_ScanStart() sets the measurement-start trigger-selection bit and the measurement-start bit to 1, CTSU enters a trigger-wait state. Measurement starts upon receiving an event. Generally, timers are used to measure periodically. Use this method when the small delay of the software trigger cannot be tolerated or when low-power operation is required. When using multiple measurement modes with external triggers, configure the next measurement mode by calling R_CTSU_ScanStart() after measurement of one mode completes.

### 1.6.2 Low-power consumption

LPC module reduces power consumption by managing transitions between operating control modes and controlling low-power states. By combining LPC module with the auto-judgement mode of CTSU2SL, low-power touch operation can be achieved. For details, refer to Smart Wakeup APN for each MCU.

### 1.6.3 Data Transfer

In CTSU module, DTC can be used instead of the CPU to handle INTCTSUWR and INTCTSURD processing during measurement. In auto-judgement mode, DTC module must be used. In other measurement modes, DTC is optional.

When using DTC module, configure the following in Smart Configurator:

- Import r_dtc_rx from Components tab.
- In the r_dtc_rx settings, set “DMAC FIT check” to “DMAC FIT module is not used with DTC FIT module”.
- In the r_ctsu_qe settings, set “Data transfer of INTCTSUWR and INTCTSURD” to “DTC”.
- In the r_bsp settings, set the heap size to 0x1000 or larger.

### 1.6.4 Diagnosis

In CTSU2L diagnosis mode, ADC measurements are required in some steps. Therefore, add r_s12ad_rx from Components tab in Smart Configurator. No additional per-module settings are required inside r_s12ad_rx for CTSU diagnosis.

If you also use ADC FIT in the application outside diagnosis mode, release ADC resources by closing the ADC module at the following timings:

- Before calling R_CTSU_ScanStart() for diagnosis mode for the first time after reset.
- Before calling R_CTSU_ScanStart() for diagnosis mode for the first time after executing R_CTSU_Diagnosis().

ADC module can be closed by calling R_ADC_Close().

If you do not close ADC module, R_CTSU_Diagnosis() returns FSP_ERR_ABORTED. In that case, close ADC FIT and ensure that ADC measurement by CTSU module can be performed in the next diagnosis mode execution.

## 1.7 API Overview

This module has the following API functions.

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

Table 1.5 API function list

| 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. |
| R_CTSU_ScanStop() | Stops measurement of specified touch interface configuration. |
| R_CTSU_SpecificDataGet() | Read the measurements for the specified data type for the specified touch interface. |
| R_CTSU_DataInsert() | Inserts the specified data in buffer of touch measurement results for the specified touch interface configuration. |
| R_CTSU_OffsetTuning() | Adjusts the offset register (SO) for the specified touch interface configuration. |
| R_CTSU_AutoJudgementDataGet() | Use the automatic judgement function to get all the button judgment results of the specified touch interface configuration. |

# 2. API Information

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

## 2.1 Hardware Requirements

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

- CTSU
- CTSUa
- CTSU2L
- CTSU2SL
- CTSU2SLa

## 2.2 Software Requirements

This driver depends on the following FIT modules:

- Board support package module (r_bsp) v7.70

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

- DTC module r_dtc v4.50 (In case of using DTC transfer)
  When using DTC transfer, set the Heap size of the r_bsp property to 0x1000 or more.
  Heap size of 0x1600 is recommended when using GCC compiler.
- ADC module r_s12ad_rx_v5.41 (In case of using diagnosis mode)

This driver also assumes the use of following tool:

- Renesas QE for Capacitive Touch V4.3.0

## 2.3 Supported Toolchains

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

Development environment

- Renesas e$^2$ studio 2025-12
- IAR Embedded Workbench for Renesas RX 5.20.1

Compiler

- Renesas CC-RX Toolchain v3.07.00
- GCC RX Toolchain v14.2.0.202511
- IAR C/C++ Compiler for Renesas RX version 5.20.1

## 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 module 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 CTSU module.

| r_ctsu_config.h Configuration Options | |
|---|---|
| CTSU_CFG_PARAM_CHECKING_ENABLE<br>*Default value: “BSP_CFG_PARAM_CHECKING_ENABLE” | Selects whether to include the parameter check process in the code.<br>Selecting “0” allows the user to omit the parameter check process from the code to shorten the code size.<br>“0”: Omit parameter check process from code.<br>“1”: Include parameter check process in code.<br>“BSP_CFG_PARAM_CHECKING_ENABLE”: Selection depends on BSP setting. |
| CTSU_CFG_USE_DTC_SUPPORT_ENABLE<br>*Default value: “0” | Select “1” to use DTC, rather than the main processor, to run CTSU2L’s CTSUWR interrupt and CTSURD interrupt processes.<br>Note:<br>If DTC is used elsewhere in the application, it may compete with the use of this driver. |
| CTSU_CFG_AUTO_JUDGE_ENABLE<br>*Default value: “0” | Set to “1” to enable the automatic judgment code. |
| CTSU_CFG_INTCTSUWR_PRIORITY_LEVEL<br>*Default value: “2” | Sets CTSUWR interrupt priority level (also necessary when using DTC). The priority level range is from 0 (high) to 15 (low). |
| CTSU_CFG_INTCTSURD_PRIORITY_LEVEL<br>*Default value: “2” | Sets CTSURD interrupt priority level (also necessary when using DTC). The priority level range is from 0 (high) to 15 (low). |
| CTSU_CFG_INTCTSUFN_PRIORITY_LEVEL<br>*Default value: “2” | Sets CTSUFN interrupt priority level. The priority level range is from 0 (high) to 15 (low). |
| 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 | |
| QE_TOUCH_VERSION | QE version |
| CTSU_CFG_NUM_SELF_ELEMENTS | Sets the total number of TS for self-capacitance, current measurement, and diagnosis function. |
| CTSU_CFG_NUM_MUTUAL_ELEMENTS | Sets the total number of matrixes for mutual-capacitance. |
| CTSU_CFG_NUM_AUTOJUDGE_SELF_ELEMENTS | Sets the total number of TS for self-capacitance with automatic judgement. |
| CTSU_CFG_NUM_AUTOJUDGE_MUTUAL_ELEMENTS | Sets the total number of matrixes for mutual-capacitance with automatic judgement. |
| CTSU_CFG_LOW_VOLTAGE_MODE | Enables/disables the low voltage mode. This value is set in CTSUCRAL register’s ATUNE0 bit.<br>Note:<br>This software does not support Low Voltage Mode on CTSU1, please set 0 using CTSU1. |
| CTSU_CFG_PCLK_DIVISION | Sets PCLK frequency division rate. This value is set in CTSUCR1 register’s CTSUCLK bit for CTSU1 and CTSURAL register’s CLK bit for CTSU2L. |
| CTSU_CFG_TSCAP_PORT | Sets TSCAP port.<br>Example: For P30, set “0x0300”. |

| | |
|---|---|
| CTSU_CFG_VCC_MV | Sets VCC (voltage).<br>Example: for 5.00V, set “5000”. |
| CTSU_CFG_NUM_SUMULTI | Sets the number of multi-clock measurements. |
| CTSU_CFG_SUMULTI0 | Sets the multiplication factor for the second frequency in a multi-clock measurement.<br>Recommended for RX260, RX261: 0x2F<br>Other recommended: 0x3F |
| CTSU_CFG_SUMULTI1 | Sets the multiplication factor for the second frequency in a multi-clock measurement.<br>Recommended for RX260, RX261: 0x28<br>Other recommended: 0x36 |
| CTSU_CFG_SUMULTI2 | Sets the multiplication factor for the third frequency in a multi-clock measurement.<br>Recommended for RX260, RX261: 0x36<br>Other recommended: 0x48 |
| CTSU_CFG_DIAG_SUPPORT_ENABLE | Enables/disables diagnosis function. |
| CTSU_CFG_DIAG_DAC_TS | Sets the number of TS terminal to be used for diagnosis in CTSU1. |
| CTSU_CFG_AUTO_CORRECTION_ENABLE | Select whether to enable or disable the automatic CCO correction process. |
| CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE | Select whether to enable or disable the automatic frequency correction process. |
| CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE | Select to enable or disable MEC feature. |
| CTSU_CFG_MAJORITY_MODE | Bitmap of majority judgement mode processing. The first bit is VMM, and the second bit is JMM. Set according to the touch interface configuration.<br>1 : VMM<br>2 : JMM<br>3 : VMM and JMM |

## 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 C compiler version or the compile options.

Using Renesas CC-RX Toolchain v3.07.00, the following is the size at compilation settings.
Only settings related to size are shown.

- CTSU_CFG_PARAM_CHECKING_ENABLE 0
- CTSU_CFG_DTC_SUPPORT_ENABLE 0
- CTSU_CFG_AUTO_JUDGE_ENABLE 0
- CTSU_CFG_LOW_VOLTAGE_MODE 0
- CTSU_CFG_TEMP_CORRECTION_SUPPORT 0
- CTSU_CFG_CALIB_RTRIM_SUPPORT 0
- CTSU_CFG_AUTO_CORRECTION_ENABLE 0
- CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE 0
- CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE 0

The size of the self-capacitance and the mutual-capacitance are shown in one element, and the size is increased by adding one element. It also includes qe_touch_config.c output by QE.

**[CTSU1]**

- CTSU_CFG_NUM_SUMULTI 1

| Mode and element num | Self-capacitance 1 element | + 1 element | Mutual-capacitance 1 element | +1 element |
|---|---|---|---|---|
| ROM | 2678 bytes | +24 bytes | 2995 bytes | +22 bytes |
| RAM | 223 bytes | +23 bytes | 241 bytes | +35 bytes |

**[CTSU2L] VMM**

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 1

| Mode and element num | Self-capacitance 1 element | + 1 element | Mutual-capacitance 1 element | +1 element |
|---|---|---|---|---|
| ROM | 4320 bytes | +24 bytes | 4693 bytes | +22 bytes |
| RAM | 457 bytes | +41 bytes | 491 bytes | +63 bytes |

**[CTSU2L] JMM**

- CTSU_CFG_NUM_SUMULTI 3
- CTSU_CFG_MAJORITY_MODE 2

| Mode and element num | Self-capacitance 1 element | + 1 element | Mutual-capacitance 1 element | +1 element |
|---|---|---|---|---|
| ROM | 4453 bytes | +24 bytes | 4809 bytes | +22 bytes |
| RAM | 465 bytes | +49 bytes | 507 bytes | +79 bytes |

* The code measurement method has been changed in APN of FIT QE CTSU Module Firmware Integration Technology Rev.3.20.

## 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.

Table 2.1 shows the ctsu_ctrl_t structures (control structures). For information about the data types used in this structure r_ctsu_qe.h. Manage the measurement settings and measurement results for each touch interface configuration. By using QE for Capacitive Touch, the variables of the control structure according to the touch interface configuration are output to qe_touch_config.c, so set them as the first argument of the API of this module.

**Table 2.1 ctsu_ctrl_t Structure**

| Data Type | Member | Description |
|---|---|---|
| uint32_t | open | Open flag |
| volatile ctsu_state_t | state | Measurement state |
| ctsu_cap_t | cap | Measurement trigger |
| ctsu_md_t | md | Measurement mode |
| ctsu_tuning_t | tuning | Initial offset tuning flag |
| uint16_t | num_elements | Number of elements |
| uint16_t | wr_index | Index of CTSUWR interrupt |
| uint16_t | rd_index | Index of CTSURD interrupt |
| uint8_t * | p_element_complete_flag | Pointer to the flag indicating the completion of offset tuning for the element |
| int32_t * | p_tuning_diff | Pointer to the difference from the target value |
| uint16_t | average | Number of moving average operations |
| uint16_t | num_moving_average | Number of samples used for moving average operation |
| uint8_t | ctsucr1 | CTSUCR1 setting |
| ctsu_ctsuwr_t * | p_ctsuwr | CTSUWR setting |
| ctsu_self_buf_t * | p_self_raw | Pointer to the Self-capacitance Measurement raw value buffer |
| uint16_t * | p_self_corr | Pointer to the self-capacitance corrected value buffer |
| uint16_t * | p_self_mfc | Pointer to the self-capacitance multi-clock corrected value buffer |
| ctsu_data_t * | p_self_data | Pointer to the self-capacitance measurement value buffer |
| ctsu_mutual_buf_t * | p_mutual_raw | Pointer to the mutual-capacitance measurement raw value buffer |
| uint16_t * | p_mutual_pri_corr | Pointer to the mutual-capacitance primary corrected value buffer |
| uint16_t * | p_mutual_snd_corr | Pointer to the mutual-capacitance secondary corrected value buffer |
| uint16_t * | p_mutual_pri_mfc | Pointer to the mutual-capacitance primary multi-clock corrected value buffer |
| uint16_t * | p_mutual_snd_mfc | Pointer to the mutual-capacitance secondary multi-clock corrected value buffer |
| ctsu_data_t * | p_mutual_pri_data | Pointer to the mutual-capacitance primary corrected measurement value buffer |
| ctsu_data_t * | p_mutual_snd_data | Pointer to the mutual-capacitance secondary corrected measurement value buffer |
| ctsu_correction_info_t * | p_correction_info | Pointer to the correction information |
| ctsu_txvsel_t | txvsel | TXVSEL setting |
| ctsu_txvsel2_t | txvsel2 | TXVSEL2 setting |
| uint8_t | ctsuchac0 | CHAC0 setting |
| uint8_t | ctsuchac1 | CHAC1 setting |

| Data Type | Member | Description |
|---|---|---|
| uint8_t | ctsuchac2 | CHAC2 setting |
| uint8_t | ctsuchac3 | CHAC3 setting |
| uint8_t | ctsuchac4 | CHAC4 setting |
| uint8_t | ctsuchtrc0 | CHTRC0 setting |
| uint8_t | ctsuchtrc1 | CHTRC1 setting |
| uint8_t | ctsuchtrc2 | CHTRC2 setting |
| uint8_t | ctsuchtrc3 | CHTRC3 setting |
| uint8_t | ctsuchtrc4 | CHTRC4 setting |
| uint16_t | self_elem_index | Index of the self-capacitance element |
| uint16_t | mutual_elem_index | Index of the mutual-capacitance element |
| uint16_t | ctsu_elem_index | Element index |
| ctsu_cfg_t const * | p_ctsu_cfg | Pointer to the configuration structure |
| void | (* p_callback) (ctsu_callback_args_t *) | Pointer to the callback function |
| uint8_t | interrupt_reverse_flag | Flag for indicating reversal of the order of interrupts |
| ctsu_event_t | error_status | Error state |
| ctsu_callback_args_t * | p_callback_memory | Callback function stored (for TrustZone) |
| void const * | p_context | Context pointer |
| bool | serial_tuning_enable | Flag for enabling serial tuning |
| uint16_t | serial_tuning_mutual_cnt | Serial tuning |
| uint16_t | tuning_self_target_value | Target value for self-capacitance offset tuning |
| uint16_t | tuning_mutual_target_value | Target value for mutual-capacitance offset tuning |
| uint8_t | tsod | TSOD setting |
| uint8_t | mec_ts | TS terminal number to be used for MEC |
| uint8_t | mec_shield_ts | TS terminal number to be used for MEC shield |
| **CTSU_CFG_DIAG_SUPPORT_ENABLE == 1** | | |
| ctsu_diag_info_t * | p_diag_info | Pointer to the diagnostic information |
| **BSP_FEATURE_CTSU_VERSION == 2** | | |
| uint8_t * | p_frequency_complete_flag | Pointer to the flag for indicating the completion of offset tuning for a multi-clock scan |
| uint8_t * | p_selected_freq_self | Pointer to the selected frequency number (self-capacitance) |
| uint8_t * | p_selected_freq_mutual | Pointer to the selected frequency number (mutual-capacitance) |
| ctsu_range_t | range | Current range |
| uint8_t | ctsucr2 | CTSUCR2 setting |
| **(BSP_FEATURE_CTSU_VERSION == 2 && CTSU_CFG_AUTO_JUDGE_ENABLE == 1)** | | |
| ctsu_auto_judge_t * | p_auto_judge | Pointer to the auto-judgement information |
| uint32_t | address_auto_judge | Address of p_auto_judge |
| uint32_t | address_ctsuwr | Address of p_ctsuwr |
| uint32_t | address_self_raw | Address of p_self_raw |
| uint32_t | address_mutual_raw | Address of p_mutual_raw |
| uint32_t | count_auto_judge | Number of DTC transfers in auto-judgement |
| uint32_t | count_ctsuwr_self_mutual | Number of CTSUWR interrupts in auto-judgement |
| uint8_t | blini_flag | BLINI setting flag |
| uint8_t | ajmmat | AJMMAT setting |
| uint8_t | ajbmat | AJBMAT setting |
| **(BSP_FEATURE_CTSU_VERSION == 2 && CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE == 1)** | | |
| uint32_t | p_mcact1 | Pointer to MCACT1 settings |
| uint32_t | p_mcact2 | Pointer to MCACT2 settings |
| uint8_t | mcact_flag | Automatic Frequency Correction Setting Flag |

Table 2.2 shows the ctsu_cfg_t structure (config structure).

By using QE for Capacitive Touch, variables according to the touch interface configuration are output to "qe_touch_config.c", so set it as the second argument of "R_CTSU_Open()". The configuration value is assumed to be set by "Smart Configurator" or "QE for Capacitive Touch", and this software does not check for errors to improve processing efficiency. Be careful if you want to modify the configs manually.

**Table 2.2 ctsu_cfg_t Structure**

| Data Type | Member Name | Description | Range of the Value |
|---|---|---|---|
| ctsu_cap_t | cap | Selects CTSU scan start trigger. | CTSU_CAP_SOFTWARE: software trigger.<br>CTSU_CAP_EXTERNAL: external trigger. |
| ctsu_txvsel_t | txvsel | Selects the transmission power. | CTSU_TXVSEL_VCC: VCC is selected.<br>CTSU_TXVSEL_INTERNAL_POWER: VDD is selected. |
| ctsu_txvsel2_t | txvsel2 | Selects the transmission power 2. (only for CTSU2) | CTSU_TXVSEL_MODE: Power is selected by TXVSEL setting.<br>CTSU_TXVSEL_VCC_PRIVATE: Dedicated VCC is selected. |
| ctsu_atune1_t | atune1 | Adjusts the power capability. (only for CTSU) | CTSU_ATUNE1_NORMAL: Normal output<br>CTSU_ATUNE1_HIGH: Large-current output |
| ctsu_atune12_t | atune12 | Adjusts the power capability. (only for CTSU2) | CTSU_ATUNE12_80UA：80uA mode<br>CTSU_ATUNE12_40UA：40uA mode<br>CTSU_ATUNE12_20UA：20uA mode<br>CTSU_ATUNE12_160UA：160uA mode |
| ctsu_md_t | md | Selects CTSU measurement mode. | CTSU_MODE_SELF_MULTI_SCAN: Self multi-scan mode<br>CTSU_MODE_MUTUAL_FULL_SCAN: Mutual full-scan mode<br>CTSU_MODE_MUTUAL_CFC_SCAN: Mutual simultaneous scan mode (only for CTSU2)<br>CTSU_MODE_CURRENT_SCAN: Current-scan mode (only for CTSU2)<br>CTSU_MODE_DIAGNOSIS_SCAN: Diagnosis scan mode |
| ctsu_posel_t | posel | Selects the output from non-measurement terminals. | CTSU_POSEL_LOW_GPIO: Low level is output (GPIO).<br>CTSU_POSEL_HI_Z: Hi-Z state<br>CTSU_POSEL_LOW: Low level is output (TXVSEL or TXVSEL2 setting)<br>CTSU_POSEL_SAME_PULSE: In-phase (transmission) pulses are output (TXVSEL or TXVSEL2 setting) |
| uint8_t | tsod | Selects measurement or fixed output from TS terminals. | 0: Electrostatic capacitance measurement mode<br>1: A fixed level (high or low) is output from TS terminals. |
| uint8_t | mec_ts | TS terminal number to be used for MEC function | 0 to 35 |
| uint8_t | mec_shield_ts | TS terminal number of the active shield to be used for MEC function | 0 to 35 |
| uint8_t | tlot | Number of consecutive judgements of a value exceeding the low threshold in auto-judgement | 0 to 255 |
| uint8_t | thot | Number of consecutive judgements of a value exceeding the high threshold in auto-judgement | 0 to 255 |

| Data Type | Member Name | Description | Range of the Value |
|---|---|---|---|
| uint8_t | jc | Criteria for auto-judgement | 0: Touch-ON is detected when the result of judgement is that the high threshold has been exceeded once.<br>1: Touch-ON is detected when the result of judgement is that the high threshold has been exceeded twice.<br>2: Touch-ON is detected when the result of judgement is that the high threshold has been exceeded three times.<br>3: Touch-ON is detected when the result of judgement is that the high threshold has been exceeded four times. |
| uint8_t | ajmmat | Number of moving average operations for the corrected measurement values in auto-judgement | 0 to 11 (2^ set value) |
| uint8_t | ajbmat | Number of average calculations for the baseline values in auto-judgement | 0 to 15<br>(2^ (set value + 1). 0 indicates that updating of the baseline value is stopped.) |
| uint8_t | mtucfen | Calculation of mutual-capacitance in auto-judgement | 0: No subtraction<br>1: The first measurement value is subtracted from the second measurement value. |
| uint8_t | ajfen | Enables or disables auto-judgement. | 0: Auto-judgement is disabled.<br>1: Auto-judgement is enabled. |
| uint8_t | autojudge_monitor_num | QE monitoring configuration number for auto-judgement | 0 to 7 |
| uint8_t | ctsuchac0 | Mask for enabling TS00 to TS07 | 0x00 to 0xFF |
| uint8_t | ctsuchac1 | Mask for enabling TS08 to TS15 | 0x00 to 0xFF |
| uint8_t | ctsuchac2 | Mask for enabling TS16 to TS23 | 0x00 to 0xFF |
| uint8_t | ctsuchac3 | Mask for enabling TS24 to TS31 | 0x00 to 0xFF |
| uint8_t | ctsuchac4 | Mask for enabling TS32 to TS39 | 0x00 to 0xFF |
| uint8_t | ctsuchtrc0 | Mask for mutual-capacitance transmission TS00 to TS07 | 0x00 to 0xFF |
| uint8_t | ctsuchtrc1 | Mask for mutual-capacitance transmission TS08 to TS15 | 0x00 to 0xFF |
| uint8_t | ctsuchtrc2 | Mask for mutual-capacitance transmission TS16 to TS23 | 0x00 to 0xFF |
| uint8_t | ctsuchtrc3 | Mask for mutual-capacitance transmission TS24 to TS31 | 0x00 to 0xFF |
| uint8_t | ctsuchtrc4 | Mask for mutual-capacitance transmission TS32 to TS39 | 0x00 to 0xFF |
| ctsu_element_cfg_t * | p_elements | Element configuration pointer | — |
| uint8_t | num_rx | Number of receiving terminals | 0 to 36 |
| uint8_t | num_tx | Number of transmitting terminals | 0 to 36 |
| uint16_t | num_moving_average | Number of moving average operations for measured data | 0 to 65535 |
| bool | tunning_enable | Initial offset tuning flag | true: Enable<br>false: Disable |
| void * | p_callback | CTSUFN interrupt callback | — |

| Data Type | Member Name | Description | Range of the Value |
|---|---|---|---|
| void * | p_context | Context pointer | — |
| void * | p_extend | Extended configuration pointer | — |
| uint16_t | tuning_self_target_value | Target value of self-capacitance initial offset | 0 to 65535 |
| uint16_t | tuning_mutual_target_value | Target value of mutual-capacitance initial offset | 0 to 65535 |
| ctsu_auto_button_cfg_t * | p_ctsu_auto_buttons | Pointer to the array of button settings for use in auto-judgement | — |

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;

/** CTSU select data type for slect data get */
typedef enum e_ctsu_specific_data_type
{
   CTSU_SPECIFIC_RAW_DATA,
   CTSU_SPECIFIC_CORRECTION_DATA,
   CTSU_SPECIFIC_SELECTED_FREQ,
} ctsu_specific_data_type_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;

/** Configuration of each automatic judgement button */
typedef struct st_ctsu_auto_button_cfg
{
   uint8 t  elem index;             ///< Element number used by this button for automatic judgement.
   uint16_t threshold;              ///< Touch/non-touch judgement threshold for automatic
judgement.
   uint16_t hysteresis;             ///< Threshold hysteresis for chattering prevention for
automatic judgement.
} ctsu_auto_button_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 is not supported by this API
    FSP_ERR_NOT_OPEN            = 7,       ///< Requested channel is not configured or API not open
    FSP_ERR_ABORTED             = 18,      ///< An operation was aborted

    /* 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 not yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,      ///< Diagnosis of Output Voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH        = 6005,       ///< Diagnosis of CCO High failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW         = 6006,       ///< Diagnosis of CCO Low failed.
    FSP_ERR_CTSU_DIAG_SSCG           = 6007,        ///< Diagnosis of SSCG Oscillator failed.
    FSP_ERR_CTSU_DIAG_DAC            = 6008,        ///< Diagnosis of Sensor Offset failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE   = 6009,      ///< Diagnosis of Output Voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE     = 6010,      ///< Diagnosis of Over Voltage Detection failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT     = 6011,      ///< Diagnosis of Over Current Detection failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE  = 6012,      ///< Diagnosis of Load Resistance failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE   = 6013,      ///< Diagnosis of Current Offset failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN     = 6014,      ///< Diagnosis of SENSCLK Frequency failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN      = 6015,       ///< Diagnosis of SUCLK Frequency failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY   = 6016,      ///< Diagnosis of SUCLK Clock Recovery failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN        = 6017,       ///< Diagnosis of CFC oscillator failed.
} fsp_err_t;
```

## 2.11 Callback function

This FIT module calls the registered callback function when the processing of the measurement completion interrupt is completed. Set it to the member p_callback of the config structure. It has already been set in the output code of QE. It can also be set with R_CTSU_CallbackSet (). Please refer to 3.4.

The callback function should be provided by the application. When the tuning result is output using QE, the sample code of the callback function below is also output. The output function changes depending on the software judgment and the automatic judgment. If both configurations are present, both are output.

**Software Judgement**

```
void qe_touch_callback(touch_callback_args_t * p_args)
{
    g_qe_touch_flag = 1;
    g_qe_ctsu_event = p_args -> event;
}
```

**Automatic Judgement**

```
void qe_ctsu_auto_callback(ctsu_callback_args_t * p_args)
{
    g_qe_touch_flag = 1;
    g_qe_ctsu_event = p_args -> event;
}
```

As shown below, g_qe_touch_flag is typically polled between R_CTSU_ScanStart() and R_CTSU_DataGet().

```
R_CTSU_ScanStart(g_qe_ctsu_instance.p_ctrl);

while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;

R_CTSU_DataGet(g_qe_ctsu_instance.p_ctrl, &data);
```

For information about the arguments of the callback function, see the ctsu_callback_arg_t in Chapter 2.9. touch_call_back_arg_t is a typedef of ctsu_callback_args_t in Touch module. As explained in Chapter 1.5.6, you can check whether there is an error in the measurement by using the structure member event.

## 2.12 API Processing Time

This section provides reference values for API processing time.
The conditions under which the processing time was measured are shown below:

- **Compiler** : Renesas CC-RX v3.07.00
- **Compiler optimization option** : Level 2
- **Main clock** : 32 MHz

The API processing time for the basic-operation model is shown below.
For example, the table below shows the processing time for each API when measuring a single element in Self-capacitance Measurement Mode. Note that for R_CTSU_Open(), initialization operations such as reset processing are performed, including the creation of CCO correction coefficient table, so the processing time is relatively long.

**Table 2.3 API Processing Time in Basic-Operation Mode**

| Function Name | Processing Time (µs) |
|---|---|
| R_CTSU_Open（1st） | 102251 |
| R_CTSU_Open（2nd and later） | 52 |
| R_CTSU_ScanStart | 8 |
| R_CTSU_DataGet | 24 |

The API processing time when using diagnosis mode is shown below. R_CTSU_Open() is the same as in the basic-operation model.

In diagnosis mode, since the processing time differs between the first measurement and subsequent measurements, check both R_CTSU_ScanStart() and R_CTSU_DataGet() processing times. Since the diagnosis of Output Voltage does not use the R_CTSU_DataGet function, it is excluded from the measurement.

In addition, the diagnosis item in Table 2.5 correspond to those in Table 1.2.

**Table 2.4 API Processing Time During Execution in Diagnosis Mode**

| Function Name | Processing Time (µs) |
|---|---|
| R_CTSU_Diagnosis | 1 |

**Table 2.5 API Processing Time During Execution in Diagnosis Mode**

| Diagnosis Item | R_CTSU_ScanStart | R_CTSU_DataGet |
|---|---|---|
| Output Voltage Diagnosis | 849 | |
| Over Voltage Detection Diagnosis | 16 | 7 |
| Over Current Detection Diagnosis | 15 | 7 |
| Load Resistance Diagnosis | 16 | 14 |
| Sensor Offset Diagnosis | 17 | 12 |
| SENSCLK Frequency Diagnosis | 17 | 7 |
| SUCLK Frequency Diagnosis | 16 | 7 |
| SUCLK Clock Recovery Diagnosis | 18 | 7 |

## 2.13 Adding the FIT Module to Your Project

### 2.13.1 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.

### 2.13.2 Setting module options when not using Smart Configurator

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

# 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 configuration.

### Format

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

### Parameters

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

### Return Values

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

### Properties

Prototype is declared in 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 100ms.
DTC is initialized if CTSU_CFG_DTC_SUPPORT_ENABLE is enabled when the first touch interface configuration is processed.

### 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.
When the touch interface configuration is in diagnosis mode, execute the R_CTSU_Open () of the other touch interface configuration first.

## 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 [in] Pointer to the control structure

### Return Values

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

### Properties

Prototype is declared in 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_DTC_SUPPORT_ENABLE is enabled, the function also sets DTC.
The resulting value is notified in the callback generated from the INTCTSUFN interrupt handler.
When using the automatic judgement function, the measurement settings are initialized when this function is called for the first time after offset tuning is completed.

### 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 | [in] Pointer to the control structure |
| p_data | [out] 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 */* |
| *FSP_ERR_ABORTED* | */* Operate error of ADC data collection */* |
| *FSP_ERR_CTSU_DIAG_NOT_YET* | */* The collection of diagnostic data has not been completed */* |

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

This function reads all previously corrected measured values into the specified buffer(p_data).

CTSU1: The value passed through sensor CCO correction and moving average.

CTSU2L JMM: The value passed through sensor CCO correction and moving average.

CTSU2L VMM: Sensor passed through sensor CCO correction, frequency correction and moving average.

The required buffer size varies depending on the measurement mode. Prepare the number of TS for the self-capacitance measurement and current measurement modes, and twice the number of matrixes for the mutual-capacitance measurement mode. In the case of CTSU2 JMM, data of 3 frequencies is stored, so prepare 3 times more.

In diagnosis mode, if data collection has not been completed, the function returns FSP_ERR_CTSU_DIAG_NOT_YET. If data collection is completed, it will return FSP_SUCCESS, so please call R_CTSU_Diagnosis() to check the diagnosis result. In addition, if an error occurs in ADC measurement during the output-voltage diagnosis, the function returns FSP_ERR_ABORTED.

**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 [in] Pointer to the control structure

p_callback [in] Pointer to callback function

p_context [in] Pointer to send to callback function

p_callback_memory [in] Set to NULL

### Return Values

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

### Properties

Prototype is declared in 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.

### 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 [in] Pointer to the control structure

### Return Values

*FSP_SUCCESS* */* Successfully completed */*

*FSP_ERR_ASSERTION* */* Argument pointer not specified */*

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

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

This function closes the specified touch interface configuration.

### Example:

```
fsp_err_t err;

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

### Special Notes:

When using this API together with the diagnosis mode, please note the following:

・ Do not call this API between executing R_CTSU_ScanStart() and calling R_CTSU_DataGet().

## 3.6 R_CTSU_Diagnosis

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

### Format

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

### Parameters

p_ctrl [in] Pointer to the control structure

### Return Values

| | |
|---|---|
| *FSP_SUCCESS* | */* All diagnoses are success */* |
| *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 Output Voltage failed */* |
| *FSP_ERR_CTSU_DIAG_CCO_HIGH* | */* Diagnosis of CCO High failed */* |
| *FSP_ERR_CTSU_DIAG_CCO_LOW* | */* Diagnosis of CCO Low failed */* |
| *FSP_ERR_CTSU_DIAG_SSCG* | */* Diagnosis of SSCG Oscillator failed */* |
| *FSP_ERR_CTSU_DIAG_DAC* | */* Diagnosis of Sensor Offset failed */* |
| *FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE* | */* Diagnosis of Output Voltage failed */* |
| *FSP_ERR_CTSU_DIAG_OVER_VOLTAGE* | */* Diagnosis of Over Voltage Detection failed */* |
| *FSP_ERR_CTSU_DIAG_OVER_CURRENT* | */* Diagnosis of Over Current Detection failed */* |
| *FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE* | */* Diagnosis of Load Resistance failed */* |
| *FSP_ERR_CTSU_DIAG_CURRENT_SOURCE* | */* Diagnosis of Current Offset failed */* |
| *FSP_ERR_CTSU_DIAG_SENSCLK_GAIN* | */* Diagnosis of SENSCLK Frequency failed */* |
| *FSP_ERR_CTSU_DIAG_SUCLK_GAIN* | */* Diagnosis of SUCLK Frequency failed */* |
| *FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY* | */* Diagnosis of SUCLK Clock Recovery failed */* |

### Properties

Prototyped in file r_ctsu_api.h

### Description

This is the API function providing the function for diagnosis of CTSU inner circuit. Please call the function when the return value of R_CTSU_DataGet is FSP_SUCCESS. If an abnormality is detected in any of the diagnosis items, the corresponding diagnosis error is returned. If all diagnosis is complete normally, FSP_SUCCESS is returned

### Example:

```
/* For CTSU1 */
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 succeeded. */
   }
}
```

```
/* For CTSU2 */
void R_CTSU_PinSetInit(void);
void qe_touch_main(void);
uint16_t  dummyD;

void qe_touch_main(void)
{
    fsp_err_t err;
    uint8_t initial_diag_flag = 0;

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

    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);

    /* Open CTSU driver for [Diagnosis] */
    err = R_CTSU_Open(g_qe_ctsu_instance_diagnosis.p_ctrl, g_qe_ctsu_instance_diagnosis.p_cfg);

    /* Initial Diagnosis loop */
    while (0 == initial_diag_flag)
    {
        /* for [Diagnosis] configuration */
        err = 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, &dummyD);
        if (FSP_SUCCESS == err)
        {
            err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
            if (FSP_SUCCESS == err)
            {
                initial_diag_flag = 1;
            } else {
                while(1);
            }
        }
    }

    /* Main loop */
    while (true)
    {
        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);

        /* for [Diagnosis] configuration */
        err = 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, &dummyD);
        if (FSP_SUCCESS == err)
        {
            err = R_CTSU_Diagnosis(g_qe_ctsu_instance_diagnosis.p_ctrl);
            if (FSP_SUCCESS != err)
            {
                while(1);
            }
        }

        /* FIXME: Since this is a temporary process, so re-create a waiting process yourself. */
        R_BSP_SoftwareDelay(TOUCH_SCAN_INTERVAL_EXAMPLE, BSP_DELAY_MILLISECS);
    }
}
```

**Special Notes:**

None.

## 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 [in] Pointer to the control structure

### Return Values

*FSP_SUCCESS* */* Successfully completed */*

*FSP_ERR_ASSERTION* */* Argument pointer not specified */*

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

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

This function stops measuring the specified touch interface configuration.

### Example:

```
fsp_err_t err;

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

### Special Notes:

When using this API together with the diagnosis mode, please note the following:

・ Do not call this API between executing R_CTSU_ScanStart() and calling R_CTSU_DataGet().

## 3.8 R_CTSU_SpecificDataGet

This function reads the measurements for the specified data type for the specified touch interface configuration.

### Format

```
fsp_err_t R_CTSU_SpecificDataGet (ctsu_ctrl_t * const      p_ctrl,
                                  uint16_t               * p_specific_data,
                                  ctsu_specific_data_type_t specific_data_type)
```

### Parameters

| | |
|---|---|
| p_ctrl | [in] Pointer to the control structure |
| p_specific_data | [out] Pointer to specific data array. |
| specific_data_type | [in] Specific data type to get |

### 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 */* |
| *FSP_ERR_NOT_ENABLED* | */* Specify unsupported types */* |

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

When CTSU_SPECIFIC_RAW_DATA is set to specific_data_type, the measurement value is stored in p_specific_data. Prepare a buffer that is the number of elements multiplied by the number of elements in CTSU1 and the number of elements multiplied by the number of frequencies in CTSU2.
When CTSU_SPECIFIC_CCO_CORRECTION_DATA is set to specific_data_type, the sensor CCO correction data is stored in p_specific_data. Prepare a buffer that is the number of elements multiplied by the number of elements in CTSU1 and the number of elements multiplied by the number of frequencies in CTSU2.
When CTSU_SPECIFIC_CORRECTION_DATA is set to specific_data_type, the p_specific_data stores multi-clock correction data. Only VMM of CTSU2 is valid. Prepare a buffer for the elements.
When CTSU_SPECIFIC_SELECTED_FREQ is set specific_data_type, p_specific_data contains a bitmap of the frequencies used in the majority vote. The first frequency corresponds to bit 0, the second frequency corresponds to bit 1, and the third frequency corresponds to bit 2. For example, if the first and third frequencies were used, store the 0x05. Only VMM of CTSU2 is valid.

### Example:

```
fsp_err_t err;
uint16_t specific_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI]

/* Get Specific Data */
err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, &specific_data[0],
CTSU_SPECIFIC_RAW_DATA );
```

### Special Notes:

When the specific_data_type is set to something other than CTSU_SPECIFIC_RAW_DATA, execute this API after calling R_CTSU_DataGet().

## 3.9 R_CTSU_DataInsert

This function inserts the specified data in buffer of touch measurement results for the specified touch interface configuration.

### Format

```
fsp_err_t R_CTSU_DataInsert (ctsu_ctrl_t * const p_ctrl,
                             uint16_t * p_insert_data)
```

### Parameters

| | |
|---|---|
| p_ctrl | [in] Pointer to the control structure |
| p_insert_data | [in] Pointer to insert data array. |

### 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 is supposed to process the data acquired by R_CTSU_SpecificDataGet () in the user application, such as noise suppression, and store the data in this function. Set the start address of the data array to be stored in p_insert_data. The data is stored in the corrected measurement buffer. (p_ctrl->p_self_data for Self-capacitance Measurement Mode, p_ctrl->p_mutual_pri_data and p_crtl->p_mutual_snd_data for mutual-capacitance)

### Example:

```
fsp_err_t err;
uint16_t specific_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI]

/* Get Specific Data */
err = R_CTSU_DataGet(&g_ctsu_ctrl, &specific_data[0],
CTSU_SPECIFIC_CORRECTION_DATA);

/* Noise filter process */

/* Insert data */
err = R_CTSU_DataInsert(&g_ctsu_ctrl, &specific_data[0]);
```

### Special Notes:

None

## 3.10 R_CTSU_OffsetTuning

This function adjusts the offset register (SO) for the specified touch interface configuration.

### Format

```
fsp_err_t R_CTSU_OffsetTuning (ctsu_ctrl_t * const p_ctrl);
```

### Parameters

p_ctrl [in] Pointer to the control structure

### Return Values

| | |
|---|---|
| *FSP_SUCCESS* | */* CTSU successfully configured */* |
| *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* | */* Initial offset tuning in progress */* |

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

This function adjusts the offset using all the previously corrected measured values. Call this function after the measurement is complete. Execute this function once, it returns FSP_ERR_CTSU_INCOMPLETE_TUNING until the offset adjustment is completed. Return FSP_SUCCESS when the offset adjustment is complete. Repeat the measurement and this function call until the offset adjustment is completed. See Chapter 1.5.4 for offset adjustment.
If automatic judgement is enabled, set the baseline initialization bit flag after offset adjustment is complete.

### Example:

```
fsp_err_t err;
err = R_CTSU_ScanStart (g_qe_ctsu_instance_config01.p_ctrl);
while (0 == g_qe_touch_flag) {}
g_qe_touch_flag = 0;
err = R_CTSU_OffsetTuning (g_qe_ctsu_instance_config01.p_ctrl);
```

### Special Notes:

None

## 3.11 R_CTSU_AutoJudgementDataGet

This function gets the result of the automatic judgement button for the specified touch interface configuration.

### Format

```
fsp_err_t R_CTSU_AutoJudgementDataGet (ctsu_ctrl_t * const p_ctrl,
                                       uint64_t * p_button_status)
```

### Parameters

| | |
|---|---|
| p_ctrl | [in] Pointer to the control structure |
| p_button_status | [out] Pointer to a buffer that stores the button status |

### Return Values

| | |
|---|---|
| *FSP_SUCCESS* | */* CTSU successfully configured */* |
| *FSP_ERR_ASSERTION* | */* Null pointer passed as a parameter */* |
| *FSP_ERR_NOT_OPEN* | */* Called without calling Open() */* |
| *FSP_ERR_CTSU_SCANNING* | */* Scanning this instance */* |
| *FSP_ERR_INVALID_MODE* | */* The mode of automatic judgement off is invalid */* |

### Properties

Prototype is declared in r_ctsu_api.h.

### Description

This function gets the result of the automatic judgement button. Call this function after the measurement is completed. The result is a 64-bit bitmap, stored in the order of TS numbers for the specified touch interface configuration.

When this function is called for the first time after offset tuning is completed, it is set to start the baseline average calculation.

### Example:

```
fsp_err_t err;
uint64_t button_status;

/* Open CTSU Driver  */
err = R_CTSU_Open (&g_ctsu_ctrl, &g_ctsu_cfg);

/* Initial Offset Tuning */
while (true)
{
   err = R_CTSU_ScanStart (&g_ctsu_ctrl);
   while (0 == g_qe_touch_flag) {}
   g_qe_touch_flag = 0;

   err = R_CTSU_OffsetTuning (&g_ctsu_ctrl);
}

/* Main loop */
while (true)
{
   /* for [CONFIG01] configuration */
   err = R_CTSU_ScanStart (&g_ctsu_ctrl);
   while (0 == g_qe_touch_flag) {}
   g_qe_touch_flag = 0;

   /* Get all sensor values */
   err = R_CTSU_AutoJudgementDataGet(&g_ctsu_ctrl, &button_status);
}
```

### Special Notes:

This function is only supported by CTSU2SL.

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Oct.04.18 | — | First edition issued |
| 1.10 | Jul.09.19 | 1<br>3-5<br>9,12<br>21-22<br><br>8, 10-14<br><br>1,14 | Added RX23W support.<br>Added definitions for “correction” and “offset tuning”.<br>Updated API return values.<br>Added CTSU_CMD_GET_METHOD_MODE and CTSU_CMD_GET_SCAN_INFO Control() commands.<br>Added #pragma section macros and configuration option to driver for Safety Module support (includes GCC/IAR support).<br>Added IEC 60730 Compliance section. |
| 1.11 | Jan.09.20 | 4,5<br>26,27<br><br>—<br><br>— | Added definition for “baseline” (Touch layer).<br>Added CTSU_CMD_SNOOZE_ENABLE and CTSU_CMD_SNOOZE_DISABLE Control() commands.<br>Fixed bug where a custom callback function was called twice after a scan completes.<br>Fixed compile error for RX231 when PLL had multiplier of 13.5. |
| 2.00 | Jul.30.21 | - | Full-fledged revision |
| 2.01 | Dec.17.21 | 4 | Added description to 1.1.4 Initial offset adjustment |
| | | 5 | Added description to 1.1.6 multi-measurement frequency (CTSU2L) |
| | | 6 | Added description to 1.1.7 shield function (CTSU2L) |
| | | 9 | Added description to 1.2.4 temperature compensation mode (CTSU2L) |
| | | 10 | Added API to 1.4 API overview |
| | | 14 | Fixed 2.8 Code size |
| | | 15~18 | Update to 2.9 Arguments |
| | | 28 | Added description to 3.6 R_CTSU_Diagnosis |
| | | 31~32 | Create a new 3.8 R_CTSU_SpecificDataGet |
| | | 33 | Create a new 3.9 R_CTSU_DataInsert |
| 2.10 | Apr.20.22 | 3 | Add content to the overview |
| | | 7 | Added 1.1.11 MEC function (CTSU2SL) |
| | | 7 | Added 1.1.12 Automatic judgment function (CTSU2SL) |
| | | 7,8,9 | Added 1.1.13 Automatic function (CTSU2SL) |
| | | 16 | Added contents to 2.7 Compile settings |
| | | 19,20 | Added content to 2.9 Argument |
| | | 37 | Added 3.10 R_CTSU_OffsetTuning |
| | | 38 | Added 3.11R_CTSU_AutoJudgmentDataGet |
| 2.20 | Dec.28.22 | 3 | Update 1 Overview |
| | | 7 | Added to 1.1.10 Diagnosis Function |
| | | 12 | Replaced figure at 1.2.1 Self-capacitance Mode |
| | | 14 | Added to 1.2.4 Temperature Compensation Mode (CTSU2L) |
| | | 16 | Updated 2.2 Software Requirements |
| | | 16 | Updated 2.3 Supported Toolchains |
| | | 19 | Updated 2.8 Code Size |
| | | 24 | Updated 2.9 Arguments |
| | | 29 | Updated 2.10 Return Value |
| 3.00 | Oct.15.24 | 1 | Added RX260/RX261 support. |
| | | 3 | Updated 1 Overview. Added CTSU2SLa |
| | | 4 | Updated 1.5.4 Default target value (CTSU2L) |
| | | 15~7 | Added 1.5.2 Majority Judgement Mode(JMM/VMM) |

| | | | |
|---|---|---|---|
| | | | Updated Figure 1.16 |
| | | 20 | Added 1.5.10 Automatic Frequency Correction (CTSU2SLa) |
| | | 24 | Updated 2.1 Hardware Requirements |
| | | 24 | Updated 2.2 Software Requirements |
| | | 24 | Updated 2.3 Supported Toolchains |
| | | 25 | Updated 2.7 Compilation Settings |
| | | 27 | Updated 2.8 Code Size |
| | | 28 | Updated 2.9 Arguments |
| | | 40 | Updated 3.3 R_CTSU_DataGet |
| | | 46 | Updated 3.8 R_CTSU_SpecificDataGet |
| 3.10 | Feb.19.25 | 15 | Updated 1.5.1 Random Pulse Frequency Measurement (CTSU1) |
| | | 8 | Updated 1.4.4<br>Automatic Judgement Mode (CTSU2SL) |
| | | 24 | Updated 2.2 Software Requirements |
| | | 24 | Updated 2.3 Supported Toolchains |
| | | 25 | Updated 2.7 Compilation Settings |
| | | 27 | Updated 2.8 Code Size |
| | | 28 | Updated 2.9 Arguments |
| 3.11 | Mar.21.25 | - | Updated with changes to disclaimer comment in code file.<br>No changes to the content of this APN. |
| 3.20 | Jul.31.25 | 23 | Added Description to 1.7 API Overview |
| | | 24 | Updated 2.2 Software Requirements |
| | | 24 | Updated 2.3 Supported Toolchains |
| | | 27 | Updated 2.8 Code Size |
| | | 35 | Added 2.11 Callback function |
| | | 38 - 49 | Added input and output information to API function arguments. |
| 3.30 | Mar.24.26 | 3 | Added 1.1 What is CTSU module |
| | | 3 | Added 1.2 Position of CTSU module |
| | | 4 | Added 1.3 How to use CTSU module |
| | | 23 | Moved the API overview to section 1.7 |
| | | 8~9 | 1.4.4 Auto-Judgement Mode (CTSU2SL)” |
| | | 12~14 | Added 1.4.5 Diagnosis Mode |
| | | 15 | Reorganized and reordered 1.5 functions |
| | | 21 | Added 1.6 Cooperation with other FIT modules |
| | | - | Deleted the chapter on Temperature Compensation Mode (CTSU2L) |
| | | 24 | Updated 2.2 Software Requirements |
| | | 24 | Updated 2.3 Supported Toolchains |
| | | 27 | Updated 2.8 Code Size |
| | | 36 | Added 2.12 API Processing Time |
| | | - | Deleted the chapter on IEC60730 compliance |
| | | 40 | Added explanation of return values and Diagnosis Mode for 3.3 R_CTSU_DataGet |
| | | 43 | Added explanation for 3.6 R_CTSU_Diagnosis according to the module update |
| | | - | Included sdpx file and json file in the package. |

# 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/.