RL78 Family

CTSU Module Software Integration System

Introduction

This application note describes the CTSU Module.

Target Device

RL78/G23 Group

RL78/G22 Group

RL78/G16 Group

Related Documents

RL78 Family TOUCH Module (R11AN0485)

Contents

1. Overview

The CTSU module is a CTSU driver for the Touch Module. The CTSU module is configured assuming access via the Touch middleware layer but can also be accessed from the user application.

There are 3types of CTSU peripherals, which are CTSUb, CTSU2L and CTSU2La.These are functionally different. Versions of CTSU peripherals which each device is equipped with are shown below.

CTSU2La：RL78/G22

CTSU2L：RL78/G23

CTSUb：RL78/G16

CTSUb is described as CTSU in this document. CTSU, CTSU2L and CTSU2La have different features and for the sake of explanation, CTSU2L and CTSU2La are described below.

- Common description for CTSU, CTSU2L and CTSU2La-> CTSU
- Description only for CTSU -> CTSU1
- Common description for CTSU2L and CTSU2La -> CTSU2L
- Description only for CTSU2La -> CTSU2La

1.1 Functions

The CTSU module supports the following functions.

1.1.1 QE for Capacitive Touch Usage

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

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

1.1.2 Measurements and Obtaining Data

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

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

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

Start the measurement with API function R_CTSU_ScanStart().

Obtain the measurement results with API function R_CTSU_DataGet().

1.1.3 Sensor CCO Correction function

The 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.

When temperature correction for CTSU2L is enabled, an external resistor connected to a TS terminal is used to periodically update the correction coefficient. By using an external resistor that is not dependent on temperature, you can even correct the temperature drift of the sensor CCO.

1.1.4 Initial Offset Adjustment

The CTSU 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. 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.

This function can be turned off in the configuration settings.

Default target value

Mode	Default target value (CTSU1)	Default target value (CTSU2L)
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 the 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 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 mode)

Target value	Target value (multi frequency)	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.

1.1.5 Random Pulse Frequency Measurement (CTSU1)

The CTSU1 peripheral measures at one drive pulse frequency.

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

The drive pulse frequency is calculated as below.

It is determined by f_{CLK} 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 f_{CLK} =32MHz, CTSUCLK = PLCK/2, and CTSUSDPA = 1/16, then drive pulse frequency is 0.5MHz. CTSUSDPA can change for each TS port.

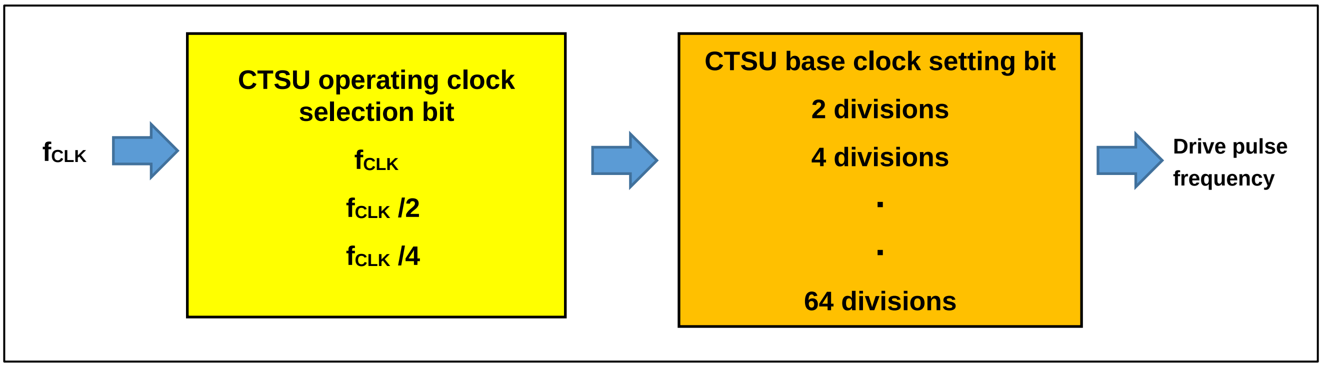


Figure 1 Drive Pulse Frequency Settings

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

CTSUSOFF = 0, CTSUSSMOD = 0, CTSUSSCNT = 3

1.1.6 Multi-frequency Measurements (CTSU2L)

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

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

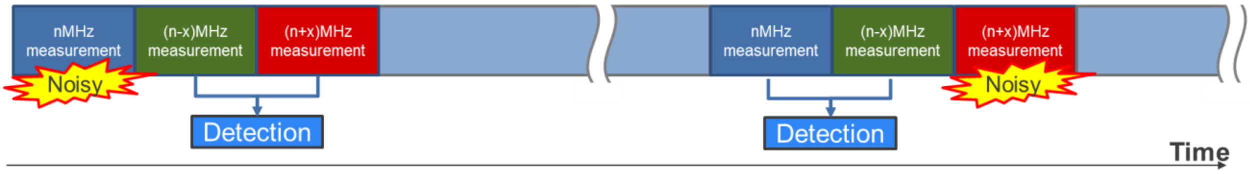


Figure 2 Multi-frequency Measurements

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

Figure 3 shows the flowchart of JMM and VMM with the Touch module.

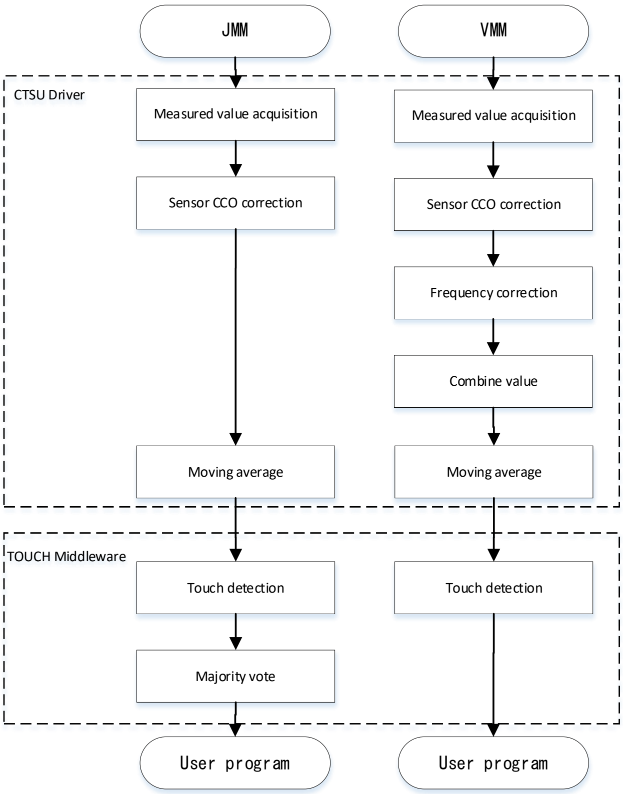


Figure 3 Flowchart of JMM and VMM

JMM makes a final touch judgment by using majority touch judgment results for each of the three CCO-corrected measurement values.

VMM performs frequency correction to standardize the three CCO-corrected measurements to the measured values at the first frequency and adds two measurements with similar values. This results in a measurement value that doubles the measurement time. Touch judgment is made with this measured value.

Example VMM Calculations

From the frequency-corrected 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 chattering, a combination of value 1 and value 2 is given weight to be selected. When comparing value 3, multiply the difference value 2 by 2 and multiply the difference value 3 by 1.5.

Value 1	Value 2	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_ 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 of each previous processes, use R_CTSU_SepcificDataGet(). These data can also be used to determine the data with its own noise filter in the Touch module. See Chapter 3.9 and 3.10 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 pulse frequencies.

Drive pulse frequency is calculated in the following equation:

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

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

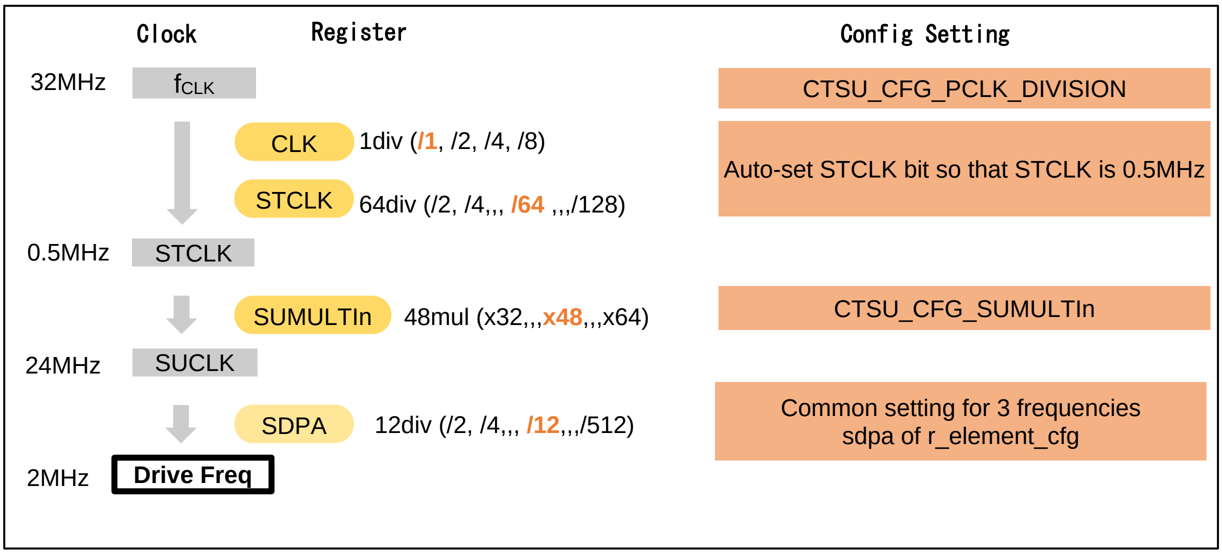


Figure 4 Drive Pulse Frequency Settings

1.1.7 Shield Function (CTSU2L)

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

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

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

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

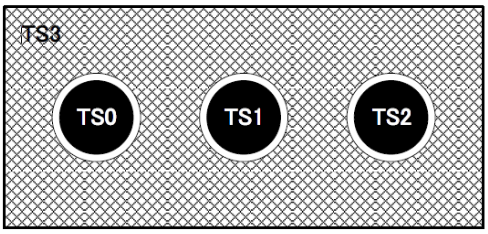


Figure 5 Example of Shield Electrode Structure

1.1.8 Measurement Error Message

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

In the measurement complete interrupt process, This module reads CTSUICOMP and CTSUSOVF for CTSU1, ICOMP0, ICOMP1, and SENSOVF for CTSU2L, and notifies the results in the callback function. The status register is reset after the contents are read. For more details on abnormal measurements, refer to “member event” in the ctsu_callback_args_t callback function argument.

1.1.9 Moving Average

This function calculates the moving average of the measured results. Set the number of times the moving average should be calculated in the config settings.

1.1.10 Diagnosis Function

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

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

CTSU1 and 9 types for CTSU2L.The diagnosis function is executed by calling the API function. This is executed independently from the other measurements.

To enable the diagnosis function, set CTSU_CFG_DIAG_SUPPORT_ENABLE to 1.

CTSU1 and CTSU2L use ADC.

For CTSU1, 27pF condenser should be connected externally. After diagnostic function measurement, wait about 1ms before starting touch scanning.

1.1.11 Automatic judgment measurement using SNOOZE Mode Sequencer (SMS) (RL78/G22, RL78/G23)

This function uses SMS to operate from measurement to touch judgment without CPU operation. Since the CPU operates only in STOP mode and SNOOZE mode, it can measure with low power consumption. Only external trigger setting and DTC setting is supported. Please use 32-bit interval timer with f_{sxp} selected as the operating clock for the external trigger.

For the touch interface for which you want to use this function, please call R_CTSU_SmsSet () and then start measurement with R_CTSU_ScanStart (). It is recommended to execute after the initial offset adjustment is completed.

Every time the CTSU peripheral measures with an external trigger and reads the result, SMS performs the processing equivalent to R_CTSU_DataGet () and the touch judgment processing.

When touch ON is determined, an INTSMSE interrupt is occurred and the same callback function as for normal measurement is called and cancel the SMS measurement setting. At that time the application can get the measurement result by calling R_CTSU_DataGet () as in the normal operation.

When using this function, SMS cannot be used for other processing of the system.

To enable this function, set the measurement setting by external trigger and CTSU_CFG_DTC_SUPPORT_ENABLE to 1 and CTSU_CFG_SMS_SUPPORT_ENABLE to 1. Since DTC repeat transmission is used, the lower 8 bits of the variable specified in the repeat area must be 00H. Therefore, set the address of the RAM area and the address where the lower 8 bits are 00H in CTSU_CFG_SMS_TRANSFER_ADDRESS and CTSU_CFG_SMS_CTSUWR_ADDRESS. Variables placed in CTSU_CFG_SMS_TRANSFER_ADDRESS use 544 bytes. The variable placed in CTSU_CFG_SMS_CTSUWR_ADDRESS uses (4 * number of elements * number of multi-frequency). For example, 36 bytes are used for 3 frequency measurement with 3 self-capacity buttons.

To tuning with the QE for Capacitive Touch, set CTSU_CFG_SMS_TRANSFER_ADDRESS to value other than 0xFEF00 to 0xFC800, and CTSU_CFG_SMS_CTSUWR_ADDRESS to value other than 0xFF200 to 0xFCB00.

Figure 6 shows the flow of modules used for SMS measurement with RL78/G22.

Port output using DTC from CTSU2L. An interrupt signal is generated using the signal output from the port. An interrupt signal triggers the ELC to start SMS processing.

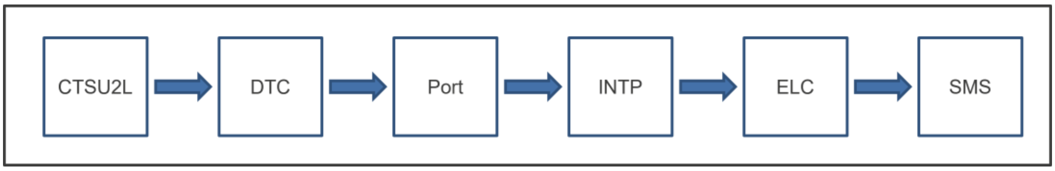


Figure 6 Module flow used for SMS measurement with RL78/G22

To perform automatic judgment measurement using SMS in the RL78/G22, set the following.

i. Setting Port

Set the port register to CTSU_CFG_SMS_EXTRIGGER_PORT, and set the bit used by the register to CTSU_CFG_SMS_EXTRIGGER_BIT. For example, when using P140 register, set CTSU_CFG_SMS_EXTRIGGER_PORT to P14 and CTSU_CFG_SMS_EXTRIGGER_BIT to 1. Also set the used port to output and set to low before starting measurement.

Note: DTC transfer rewrites the port register (Pxx) in 8-bit units. Therefore, in automatic judgment measurement processing using SMS, the port register (Pxx) that is the DTC transfer destination cannot be used for other functions. Select the port register (Pxx) to be used to avoid conflicts in system.

ii. External interrupt

Set the interrupt number to CTSU_CFG_SMS_ELC_INTP. For example, when using INTP1, set CTSU_CFG_SMS_ELC_INTP to 1. Also, set the external interrupt to be used to enable interrupt before starting measurement.

Connect the port and the external interruption on the board.

Majority judgment mode is set to JMM while using SMS.

This feature does not perform with the self-capacitance to 9 elements or more and the mutual capacitance to 8 elements or more.

1.1.12 Multiple Electrode Connection (MEC) Function (CTSU, CTSU2La)

The CTSU and CTSU2La peripheral have MEC function that connects multiple electrodes and measures them as a single electrode. This feature is only available in self-capacitance mode.

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

Figure 7 shows a compare of normal measurement time and MEC measurement. Since multi channels are measured at the same time, the measurement time is shortened.

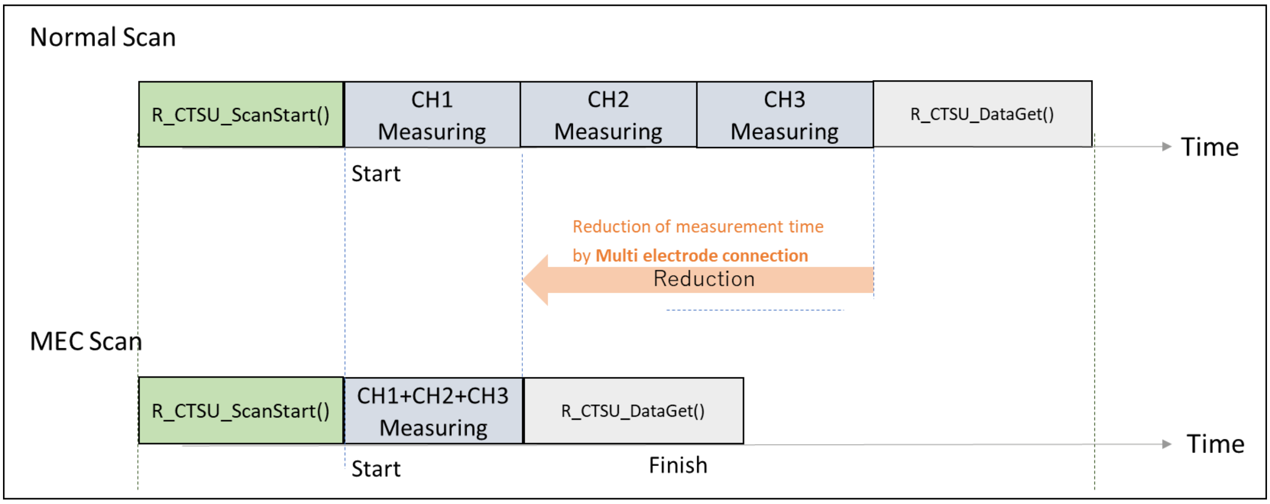


Figure 7 Compare of time between normal measurement and MEC measurement

To enable the code for the 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 the TS numbers to be measured.

If you want to use the shield function at the same time, set the 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 8, 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,
```

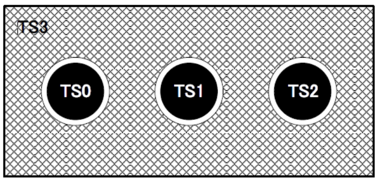


Figure 8 Example of MEC and shield electrode configuration

1.2 Measurement Mode

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

1.2.1 Self-capacitance Mode

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

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

[CTSU1]

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

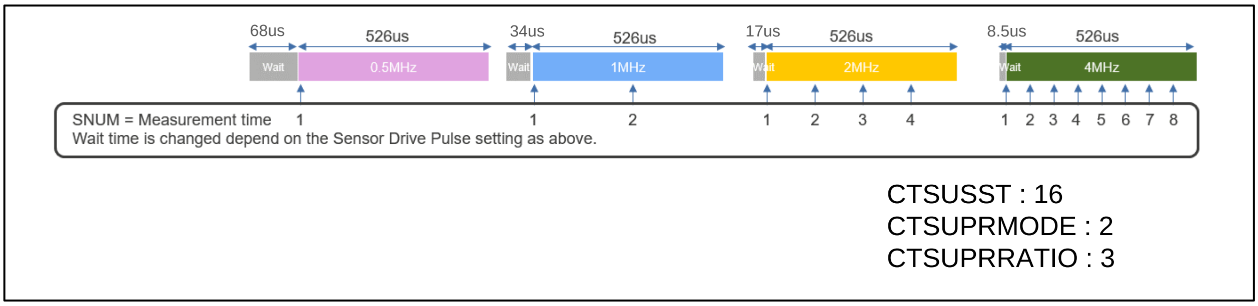


Figure 9 Self-capacitance Measurement Period (CTSU1)

[CTSU2L]

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

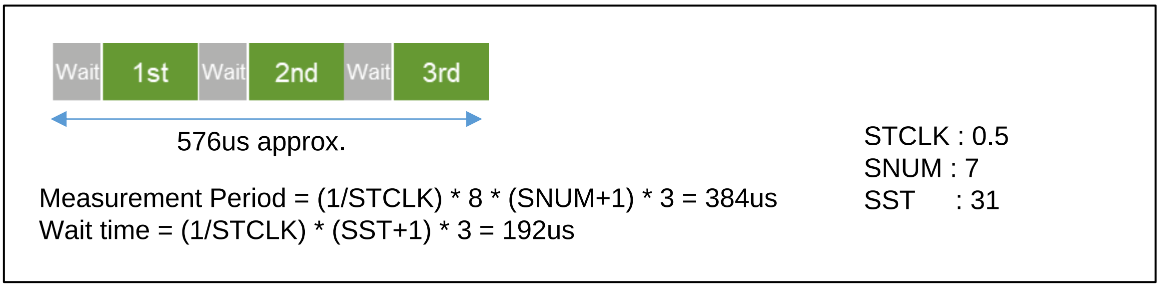


Figure 10 Self-capacitance Measurement Period

1.2.2 Mutual Capacitance Mode

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

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

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

To measure the mutual capacitance generated between electrodes, the CTSU peripheral performs the measurement process on the same electrode twice. Therefore, when the first electrode’s measurement time is set to default, the total measurement time will be about 1152us.

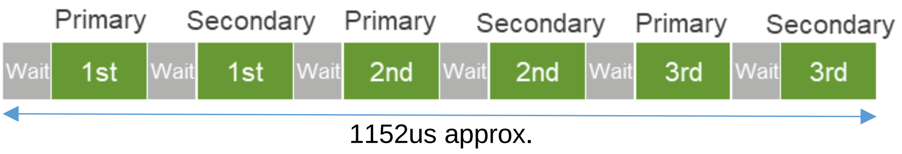
The mutual capacitance is obtained by inverting the phase relationship of the pulse output and switched capacitor in the primary and secondary measurements, and calculating the difference between the two measurements. This module does not calculate the difference, but outputs the secondary measured result.

[CTSU1]

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

[CTSU2L]

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



Measurement Period = (1/STCLK) * 8 * (SNUM+1) * 3 * 2= 768us
Wait time = (1/STCLK) * (SST+1) * 3 * 2= 384us

STCLK : 0.5
SNUM : 7
SST : 31

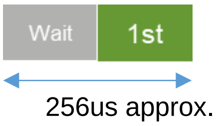
Figure 11 Mutual Capacitance Measurement Period

1.2.3 Current Measurement Mode (CTSU2L)

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

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

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



STCLK : 0.5
SNUM : 7
SST : 63

Measurement Period = (1/STCLK) * 8 * (SNUM+1) = 128us
Wait time = (1/STCLK) * (SST+1) = 128us

Figure 12 Current Measurement Period

1.2.4 Temperature Correction Mode (CTSU2L)

The temperature correction mode offered by CTSU2L is used to periodically update the correction coefficient using an external resistor connected to a TS terminal. This involves three processes as described below. Also refers to the timing chart in Figure 13.

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

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

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

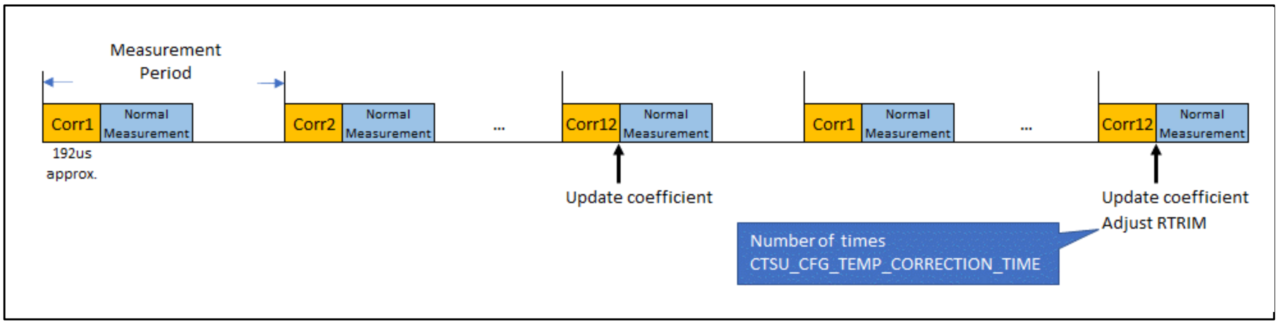


Figure 13 Temperature Correction Measurement Timing Chart

1.2.5 Diagnosis Mode

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

The details are described in 1.1.10.

1.3 Measurement Timing

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

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

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

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

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

1.4 API Overview

The CTSU module includes the following functions.

Function	Description
R_CTSU_Open()	Initializes the specified touch interface configuration.
R_CTSU_StartScan()	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_SmsSet()	Makes settings for automatic judgment measurement using SMS of the specified touch interface configuration.
R_CTSU_Close()	Closes specified touch interface configuration.
R_CTSU_Diagnosis()	Executes diagnosis.
R_CTSU_StartStop()	Stops measurement of the 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.

2. API Information

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

2.1 Hardware Requirements

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

- CTSUb
- CTSU2L
- CTSU2La

2.2 Software Requirements

This driver depends on the following module:

- Board Support Package (r_bsp) v1.70 or newer.

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

- Code generator DTC v1.00 or newer

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

- QE for Capacitive Touch V4.0.0 or newer.

2.3 Supported Toolchains

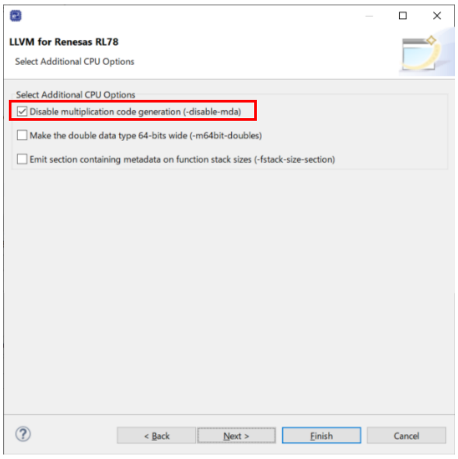
Module operations have been confirmed on the following toolchains.

- Renesas CC-RL Toolchain v1.14.00
- IAR Embedded Workbench for Renesas RL78 v5.10.3
- LLVM for Renesas RL78 v17.0.1.202406

2.4 Restrictions

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

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



LLVM for Renesas RL78

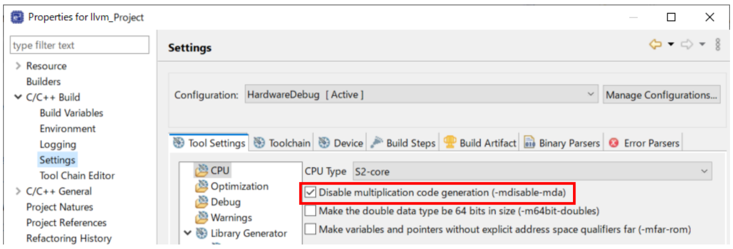
Select Additional CPU Options

Select Additional CPU Options

☑ Disable multiplication code generation (-disable-mda)

☐ Make the double data type 64-bits wide (-m64bit-doubles)

☐ Emit section containing metadata on function stack sizes (-fstack-size-section)



Properties for llvm_Project

Configuration: HardwareDebug [Active]

CPU Type S2-core

☑ Disable multiplication code generation (-mdisable-mda)

☐ Make the double data type be 64 bits in size (-m64bit-doubles)

☐ Make variables and pointers without explicit address space qualifiers far (-mfar-rom)

2.5 Header File

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

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

2.6 Integer Type

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

2.7 Compilation Settings

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

r_ctsu_config.h Configuration Options	
CTSU_CFG_PARAM_CHECKING_ENABLE *Default value: “BSP_CFG_PARAM_CHECKING_ENABLE”	Selects whether to include the parameter check process in the code. Selecting “0” allows the user to omit the parameter check process from the code to shorten the code size. “1”: Omit parameter check process from code. “2”: Include parameter check process in code. “BSP_CFG_PARAM_CHECKING_ENABLE”: Selection depends on BSP setting.
CTSU_CFG_USE_DTC *Default value: “0”	Select “1” to use the DTC, rather than the main processor, to run the CTSU2L’s CTSUWR interrupt and CTSURD interrupt processes. Note: If the DTC is used elsewhere in the application, it may compete with the use of this driver.
CTSU_CFG_DTC_USE_SC *Default value: “0”	When using DTC, select whether to use the DTC settings of Smart Configurator. “0”: DTC setting inside the CTSU module is used. “1”: DTC setting in Smart Configurator. When using the DTC setting in the RL78/G23 group, assign CTSUWR to No.22 and CTSURD to No.23, and set normal mode and 16-bit transfer. When using the DTC setting in the RL78/G22 group, assign CTSUWR to 21 and CTSURD to 22, set normal mode, 16-bit transfer, and DTC base address to 0xFFB00. Note: When the automatic judgment measurement function using SMS is enable, please set 0.
CTSU_CFG_SMS_SUPPORT_ENABLE *Default value: “0”	Select whether to enable the automatic judgment measurement function using SMS.
CTSU_CFG_SMS_TRANSFER_ADDRESS *Default value: “0xFF800”	This is the address setting of the repeat area used for DTC repeat transfer for SMS. See Section 1.1.11.
CTSU_CFG_SMS_CTSUWR_ADDRESS *Default value: “0xFFB00”	This is the address setting of the repeat area used for DTC repeat transfer for SMS. See Section 1.1.11.
CTSU_CFG_INTCTSUWR_PRIORITY_LEVEL *Default value: “2”	Sets the CTSUWR interrupt priority level (also necessary when using the DTC). The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_INTCTSURD_PRIORITY_LEVEL *Default value: “2”	Sets the CTSURD interrupt priority level (also necessary when using the DTC). The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_INTCTSUFN_PRIORITY_LEVEL *Default value: 2	Sets the CTSUFN interrupt priority level. The priority level range is from 0 (high) to 3 (low).
CTSU_CFG_SMS_EXTRIGGER_PORT *Default value: “P14”	Set the output port for calling SMS. The range is P0 ~ P7, P12 ~ P14 that can be set as an output port.
CTSU_CFG_SMS_EXTRIGGER_BIT *Default value: “1”	Sets the output port bitmap for calling SMS.

CTSU_CFG_SMS_ELC_INTP *Default value: “1”	Set the number of the interrupt input function to call SMS. The range is 0 to 7.
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 temperature correction.
CTSU_CFG_NUM_MUTUAL_ELEMENTS	Sets the total number of matrixes for mutual capacitance
CTSU_CFG_LOW_VOLTAGE_MODE	Enables/disables the low voltage mode. This value is set in the CTSUCRAL register’s ATUNE0 bit. Note: This software does not support Low Voltage Mode on CTSU1, please set 0 using CTSU1.
CTSU_CFG_PCLK_DIVISION	Sets the f_{CLK} frequency division rate. This value is set in the CTSUCRAL register’s CLK bit.
CTSU_CFG_TSCAP_PORT	Sets the TSCAP port. For example, P30 : 0x0300.
CTSU_CFG_VCC_MV	Sets the V_{DD} voltage. For example, 5.00V : 5000.
CTSU_CFG_NUM_SUMULTI	Sets the number of multi-frequency measurements.
CTSU_CFG_SUMULTI0	Sets the multiplication factor for the first frequency in a multi-frequency measurement. Recommended: 0x2F
CTSU_CFG_SUMULTI1	Sets the multiplication factor for the second frequency in a multi-frequency measurement. Recommended: 0x28
CTSU_CFG_SUMULTI2	Sets the multiplication factor for the third frequency in a multi-frequency measurement. Recommended: 0x36
CTSU_CFG_TEMP_CORRECTION_SUPPORT	Enables/disables temperature correction.
CTSU_CFG_TEMP_CORRECTION_TS	Sets the temperature correction terminal number.
CTSU_CFG_TEMP_CORRECTION_TIME	Sets the update interval for the correction coefficient of the temperature correction. Assuming 13 measurements per set in the temperature correction mode, indicate the number of sets per update.
CTSU_CFG_CALIB_RTRIM_SUPPORT	Enables/disables RTRIM correction for temperature correction. The ADC must be selected to operate with RTRIM correction enabled.
CTSU_CFG_DIAG_SUPPORT_ENABLE	Enables/disables diagnosis function.
CTSU_CFG_SMS_ELEMENT_NUM	Set the total number of elements to be measured by SMS.
CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE	Select to enable or disable the 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. 1 : VMM 2 : JMM 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 the CC-RL C compiler listed in “section 2.3 Supported Toolchains”. The code size varies according to the C compile version and compile options.

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

[CTSU1]

ROM and RAM Usage **Self-capacitance 1element**	
CTSU_CFG_PARAM_CHECKING_ENABLE 0	ROM: 3400 bytes
CTSU_CFG_DTC_SUPPORT_ENABLE 0	RAM: 146 bytes

ROM and RAM Usage **Size of each mode, amount of increase by adding elements**				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	3400 bytes	+20 bytes	3749 bytes	+30 bytes
RAM	146 bytes	+22 bytes	156 bytes	+28 bytes

[CTSU2L] VMM

ROM and RAM Usage **Self-capacitance 1element**	
CTSU_CFG_PARAM_CHECKING_ENABLE 0	ROM: 7006 bytes
CTSU_CFG_DTC_SUPPORT_ENABLE 0	RAM: 192 bytes

ROM and RAM Usage **Size of each mode, amount of increase by adding elements**				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	7006 bytes	+6 bytes	7102 bytes	+6 bytes
RAM	192 bytes	+34 bytes	317 bytes	+50 bytes

[CTSU2L] JMM

ROM and RAM Usage **Self-capacitance 1element**	
CTSU_CFG_PARAM_CHECKING_ENABLE 0	ROM: 7306 bytes
CTSU_CFG_DTC_SUPPORT_ENABLE 0	RAM: 297 bytes

ROM and RAM Usage **Size of each mode, amount of increase by adding elements**				
Mode and element num	Self-capacitance 1 element	+ 1 element	Mutual capacitance 1 element	+1 element
ROM	7306 bytes	+6 bytes	7999 bytes	+6 bytes
RAM	297 bytes	+42 bytes	333 bytes	+66 bytes

2.9 Arguments

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

These structures and enums are defined in r_ctsu_api.h along with the prototype declaration.

Table 1 shows ctsu_ctrl_t structure (control structure) for the touch interface configuration. Please refer to r_ctsu_qe.h to see more about the data type used in this structure. This structure manages measurement settings and results of each touch interface configuration. Using QE for Capacitive Touch allows the variables corresponding to the touch interface configuration to be output by qe_touch_config.c. Make sure to set qe_touch_config.c in the module's first API argument.

Table 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 the CTSUWR interrupt
uint16_t	rd_index	Index of the 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 raw value buffer
uint16_t *	p_self_corr	Pointer to the self-capacitance 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 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
ctsu_data_t *	p_mutual_pri_data	Pointer to the mutual-capacitance primary measurement value buffer
ctsu_data_t *	p_mutual_snd_data	Pointer to the mutual-capacitance secondary 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
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

Data Type	Member	Description
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 pin number to be used for MEC
uint8_t	mec_shield_ts	TS pin 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-frequency 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 mode
uint8_t	ctsucr2	CTSUCR2 setting

Table 2 shows the ctsu_cfg_t structure (configuration setting structure).

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

Data Type	Member Name	Description	Range of the Value
ctsu_cap_t	cap	Selects the CTSU scan start trigger.	CTSU_CAP_SOFTWARE: software trigger. CTSU_CAP_EXTERNAL: external trigger.
ctsu_txvsel_t	txvsel	Selects the transmission power.	CTSU_TXVSEL_VCC: VCC is selected. 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 the TXVSEL setting. CTSU_TXVSEL_VCC_PRIVATE: Dedicated VCC is selected.
ctsu_atune1_t	atune1	Adjusts the power capability. (only for CTSU)	CTSU_ATUNE1_NORMAL: Normal output CTSU_ATUNE1_HIGH: Large-current output
ctsu_atune12_t	atune12	Adjusts the power capability. (only for CTSU2)	CTSU_ATUNE12_80UA：80uA mode CTSU_ATUNE12_40UA：40uA mode CTSU_ATUNE12_20UA：20uA mode CTSU_ATUNE12_160UA：160uA mode

Data Type	Member Name	Description	Range of the Value
ctsu_md_t	md	Selects the CTSU measurement mode.	CTSU_MODE_SELF_MULTI_SCAN: Self multi-scan mode CTSU_MODE_MUTUAL_FULL_SCAN: Mutual full-scan mode CTSU_MODE_MUTUAL_CFC_SCAN: Mutual simultaneous scan mode (only for CTSU2) CTSU_MODE_CURRENT_SCAN: Current-scan mode (only for CTSU2) CTSU_MODE_CORRECTION_SCAN: Correction scan mode (only for CTSU2) CTSU_MODE_DIAGNOSIS_SCAN: Diagnosis scan mode
ctsu_posel_t	posel	Selects the output from non-measurement pins.	CTSU_POSEL_LOW_GPIO: Low level is output (GPIO). CTSU_POSEL_HI_Z: Hi-Z state CTSU_POSEL_LOW: Low level is output (TXVSEL or TXVSEL2 setting) CTSU_POSEL_SAME_PULSE: In-phase (transmission) pulses are output (TXVSEL or TXVSEL2 setting)
uint8_t	tsod	Selects measurement or fixed output from the TS pins.	0: Electrostatic capacitance measurement mode 1: A fixed level (high or low) is output from the TS pins.
uint8_t	mec_ts	TS pin number to be used for the MEC function	0 to 35
uint8_t	mec_shield_ts	TS pin number of the active shield to be used for the MEC function	0 to 35
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 pins	0 to 36
uint8_t	num_tx	Number of transmitting pins	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 false: Disable
void *	p_callback	CTSUFN interrupt callback	—
void *	p_context	Context pointer	—

Data Type	Member Name	Description	Range of the Value
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

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

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

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

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

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

/** CTSU Power Supply Capacity Adjustment (CTSU Only) */
typedef enum e_ctsu_atune1
{
    CTSU_ATUNE1_NORMAL,                ///< Normal output (40uA)
    CTSU_ATUNE1_HIGH                   ///< High-current output (80uA)
} ctsu_atune1_t;

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

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

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

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

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

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

2.10 Return Values

The following provides return values for the API functions. The enum is defined in fsp_common_api.h, along with the API function prototype declaration.

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

      FSP_ERR_ASSERTION              = 1,                       ///< A critical assertion has failed
      FSP_ERR_INVALID_POINTER        = 2,                       ///< Pointer points to invalid memory location
      FSP_ERR_INVALID_ARGUMENT       = 3,                       ///< Invalid input parameter
      FSP_ERR_INVALID_CHANNEL        = 4,                       ///< Selected channel does not exist
      FSP_ERR_INVALID_MODE           = 5,                       ///< Unsupported or incorrect mode
      FSP_ERR_UNSUPPORTED            = 6,                       ///< Selected mode not supported by this API
      FSP_ERR_NOT_OPEN               = 7,                       ///< Requested channel is not configured or API not open
      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 no yet.
    FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE = 6004,       ///< Diagnosis of LDO over voltage failed.
    FSP_ERR_CTSU_DIAG_CCO_HIGH         = 6005,       ///< Diagnosis of CCO into 19.2uA failed.
    FSP_ERR_CTSU_DIAG_CCO_LOW          = 6006,       ///< Diagnosis of CCO into 2.4uA failed.
    FSP_ERR_CTSU_DIAG_SSCG             = 6007,       ///< Diagnosis of SSCG frequency failed.
    FSP_ERR_CTSU_DIAG_DAC              = 6008,       ///< Diagnosis of non-touch count value failed.
    FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE   = 6009,       ///< Diagnosis of LDO output voltage failed.
    FSP_ERR_CTSU_DIAG_OVER_VOLTAGE     = 6010,       ///< Diagnosis of over voltage detection circuit failed.
    FSP_ERR_CTSU_DIAG_OVER_CURRENT     = 6011,       ///< Diagnosis of over current detection circuit failed.
    FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE  = 6012,       ///< Diagnosis of LDO internal resistance value failed.
    FSP_ERR_CTSU_DIAG_CURRENT_SOURCE   = 6013,       ///< Diagnosis of Current source value failed.
    FSP_ERR_CTSU_DIAG_SENSCLK_GAIN     = 6014,       ///< Diagnosis of SENSCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_SUCLK_GAIN       = 6015,       ///< Diagnosis of SUCLK frequency gain failed.
    FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY   = 6016,       ///< Diagnosis of SUCLK clock recovery function failed.
    FSP_ERR_CTSU_DIAG_CFC_GAIN         = 6017,       ///< Diagnosis of CFC oscillator gain failed.
} fsp_err_t;
```

3. API Functions

3.1 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 : Pointer to the control structure

p_cfg : Pointer to the config structure

Return Values

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

Properties

Prototype is declared in r_ctsu_api.h

Description

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

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

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

Example

```
fsp_err_t err;

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

/* Initialize the API. */
err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);

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

Special Notes:

The port must be initialized before calling this function. We recommend using the R_CTSU_PinSetInit() function generated by SmartConfigurator as the port initialization function.

3.2 R_CTSU_ScanStart

This function starts measurement of the specified touch interface configuration.

Format

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

Parameters

p_ctrl : Pointer to the control structure

Return Values

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

Properties

Prototype is declared in r_ctsu_api.h.

Description

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

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

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

Example

```
fsp_err_t err;

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

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

Special Notes:

None

3.3 R_CTSU_DataGet

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

Format

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

Parameters

p_ctrl : Pointer to the control structure

p_data: Pointer to the buffer that stores the measured value.

Return Values

FSP_SUCCESS	*/* CTSU initialization successfully completed */*
FSP_ERR_ASSERTION	*/* Argument pointer not specified */*
FSP_ERR_NOT_OPEN	*/* Called without calling Open() */*
FSP_ERR_CTSU_SCANNING	*/* scanning */*
FSP_ERR_CTSU_INCOMPLETE_TUNING	*/*Tuning initial offset */*

Properties

Prototype is declared in r_ctsu_api.h.

Description

This function reads all previously measured values into the specified buffer by 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 twice the number of TS for the self-capacitance and current measurement modes, and twice the number of matrixes for the mutual-capacitance mode. In the case of CTSU2L JMM, data of 3 frequencies is stored, so prepare 3 times more.

The value measured in the temperature correction mode is not stored. When RTRIM adjustment is performed, the RTRIM value is stored. At this time, the ADC settings have been changed in this function, so perform the process to return to the ADC settings you are using. Otherwise, store 0xFFFF.

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

Example:

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

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

Special Notes:

None

3.4 R_CTSU_CallbackSet

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

Format

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

Parameters

p_api_ctrl : Pointer to the control structure

p_callback: Pointer to callback function

p_context: Pointer to send to callback function

p_callback_memory: Set to NULL

Return Values

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

Properties

Prototype is declared in r_ctsu_api.h.

Description

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

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

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_SmsSet

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

Format

```
fsp_err_t R_CTSU_SmsSet (ctsu_ctrl_t * const p_ctrl,
                         uint16_t * p_threshold,
                         uint16_t * p_hysteresis,
                         uint16_t count_filter)
```

Parameters

p_ctrl : Pointer to the control structure

p_threshold: Pointer to the touch judgment threshold

p_context: Pointer to the touch judgment hysteresis

count_filter: Touch count match filter value (upper 8 bits are for OFF, lower 8 bits are for ON)

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 following: Use the argument setting for touch judgment processing.

- Disable CTSUFN interrupts
- Enable SMS module
- SMS settings
- ELCL setting (when using RL78/G23 group)
- ELC settings (when using the RL78/G22 group)
- Start SMS

To start automatic judgment measurement, call R_CTSU_ScanStart () for the same touch interface after calling this function. When touch ON is determined, INTSMSE occurs, and the following settings are made in the interrupt handler of the CTSU module.

- Set the measurement status as an argument of the callback function
- Set the measured value as a variable
- Callback function call
- Allow CTSUFN interrupts
- Disable SMS module

Example:

```
fsp_err_t err;
uint16_t threshold[3] = {1000, 1500, 2000};
uint16_t hysteresis[3] = {50, 75, 100};
uint16_t buf[3];

    /* Start SMS measurement */
    err = R_CTSU_SmsSet(&g_ctsu_ctrl, threshold, hysteresis[3], 0x0303);
    err = R_CTSU_ScanStart(&g_ctsu_ctrl);

    __stop();


    err = R_CTSU_DataGet(&g_ctsu_ctrl, buf);
```

Special Notes:

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

3.6 R_CTSU_Close

This function closes the specified touch interface configuration.

Format

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

Parameters

p_ctrl : Pointer to the control structure

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:

None

3.7 R_CTSU_Diagnosis

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

Format

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

Parameters

p_ctrl : Pointer to the control structure

Return Values

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

Properties

Prototype is declared in r_ctsu_api.h.

Description

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

Call when the return value of the function R_CTSU_DataGet is FSP_SUCCESS.

Example:

```
fsp_err_t err;
uint16_t  dummy;

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

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

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

Special Notes:

None

3.8 R_CTSU_ScanStop

This function stops measuring the specified touch interface configuration.

Format

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

Parameters

p_ctrl : Pointer to the control structure

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:

None

3.9 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 : Pointer to the control structure

p_specific_data: Pointer to specific data array.

specific_data_type: 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	*/* CTSU_SPECIFIC_SELECTED_FREQ for CTSU1 */*

Properties

Prototype is declared in r_ctsu_api.h.

Description

When CTSU_SPECIFIC_RAW_DATA is set to specific_data_type, the RAW 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_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 fequencies in CTSU2.

When CTSU_SPECIFIC_CORRECTION_DATA is set to specific_data_type, the p_specific_data stores multifrequency correction data. Only the VMM of CTSU2 is valid. Prepare a buffer for the number of elements.

When CTSU_SPECIFIC_SELECTED _DATA is set specific_data_type, p_specific_data contains a bitmap of the frequencies used in the majority vote. Only the VMM of CTSU2 is valid. For example, if the first and third frequencies were used, store the 0x05.

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_CORRECTION_DATA);
```

Special Notes:

None

3.10 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 : Pointer to the control structure

p_insert_data : 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. For self-capacity mode, store in p_ctrl-> p_self_data. For mutual capacity, store in p_ctrl-> p_mutual_pri_data and p_crtl-> p_mutual_snd_data.

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

3.11 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: 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	*/*Tuning initial offset */*

Properties

Prototype is declared in r_ctsu_api.h.

Description

This function adjusts the offset using all the previously 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 1.1.4 for offset adjustment.

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.13.21	-	First edition issued
1.10	Aug.31.21	5	Added 1.1.9 Diagnosis Function
		5	Added 1.1.10 Automatic judgment measurement using SMS
		8	Added 1.2.5 Diagnosis Mode
		9	Updated 1.4 API overview
		11	Updated 2.7 Compilation settings
		13	Updated 2.8 Code size
		13	Updated 2.9 Arguments
		16	Updated 2.10 Return Values
		-	Deleted R_CTSU_VersionGet
		24	Added 3.5 R_CTSU_SmsSet
		27	Added 3.7 R_CTSU_Diagnosis
		29	Added 3.8 R_CTSU_ScanStop
1.11	Jan.18.22	3,4	Added 1.1.4 Initial offset adjustment
		5	Added 1.1.6 multi-measurement frequency (CTSU2L)
		9	Updated 1.4 API overview
		10	Updated 2.2 Software Requirements Updated 2.3 Supported Toolchains
		13	Updated 2.8 Code size
		13-14	Updated 2.9 Arguments
		30-31	Added 3.8 R_CTSU_SpecificDataGet
		31-32	Added 3.9 R_CTSU_DataInsert
1.20	Apr.20.22	6	Added 1.1.10 Automatic judgment measurement using SMS
		4,5	Fixed PCLKB to f_{CLK}
		9	Updated 1.4 API overview
		10	Update 2.2 Software Requirement Update 2.3 Supported Toolchain
		33	Added 3.11 R_CTSU_OffsetTuning
		24	Fixed Example: in 3.5 R_CTSU_SmsSet
1.30	Feb.14.23	1	Added RL78/G22 to Target Device
		6	Updated 1.1.11 Automatic judgment measurement using SNOOZE Mode Sequencer (SMS
		7	Added 1.1.12 Multiple Electrode Connection (MEC) Function (CTSU, CTSU2La)
		12	Updated 2.2 Software Requirements
		12	Updated 2.3 Supported Toolchains
		13,14	Updated 2.7 Compilation Settings
		13	Updated 2.8 Code Size
		15,16	Updated 2.9 Arguments
		18	Updated 2.10 Return Values
		24	Updated 3.5 R_CTSU_SmsSet
1.40	Jun.14.23	1	Added RL78/G16 group to Target Device
		3	Added CTSU description to 1 Overview
		3	Updated 1.1.2 Measurements and Obtaining Data
		4	Added CTSU1 function description to 1.1.4 Initial Offset Adustment
		4	Added 1.1.5 Random Pulse Frequency Measurement (CTSU1)

		7	Added CTSU1 function description to 1.1.10 Diagnosis Function
		10,11	Added CTSU1 function description to 1.2 Measurement Mode
		14	Updated 2.1 Hardware Requirements
		14	Updated 2.2 Software Requirements
		14	Updated 2.4 Restrictions
		18	Updated 2.8 Code Size
		18	Updated 2.9 Arguments
		31	Updated 3.7 R_CTSU_Diagnosis
1.50	May.31.24	4	Updated 1.1.4 Initial offset adjustment
		6	Updated 1.1.6 multi-measurement frequency (CTSU2L)
		8	Updated 1.1.11 Automatic judgment measurement using SNOOZE Mode Sequencer (SMS
		14	Updated 2.2 Software Requirements
		14	Updated 2.3 Supported Toolchains
		18	Updated 2.7 Compilation Settings
		18	Updated 2.8 Code Size
		29	Updated 3.5 R_CTSU_SmsSet
2.00	Oct.15.24	3	Changed “ICO” to “CCO” at 1.1.3 (applied to all documents)
		5	Added majority measurement (JMM/VMM) to 1.1.6
		15	Updated 2.2 Software Requirements
		15	Updated 2.3 Supported Toolchains
		17	Updated 2.7 Compilation Settings
		19	Updated 2.8 Code Size
		20	Updated 2.9 Arguments
		29	Updated description of 3.3 R_CTSU_DataGet
		38	Updated description of 3.9 R_CTSU_SpecificDataGet

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/.