RX Family

QE CTSU Module Using Firmware Integration Technology

R01AN4469EU0100
Rev. 1.00
Oct 4, 2018

Introduction

The QE CTSU Module Using Firmware Integration Technology (FIT) has been developed as a driver layer for the QE Touch Module. **The CTSU module is not meant to be accessed directly by the user application**, but rather only by the Touch middleware layer. This serves solely as a reference document.

Target Device

The following is a list of devices that are currently supported by this API:

- RX113 Groups
- RX130 Group
- RX231, RX230 Groups

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

Related Documents

- Using QE and FIT to Develop Capacitive Touch Applications (R01AN4516EU)
- Firmware Integration Technology User’s Manual (R01AN1833EU)
- Board Support Package Firmware Integration Technology Module (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)

Contents

1. Overview

The QE CTSU FIT module is provided as a driver layer in support of the QE Touch FIT module. Both Self and Mutual operational modes are supported. Scans may be triggered by software or an external trigger.

1.1 Features

Below is a list of the features supported by the QE CTSU FIT module.

- All arguments for Open() are generated by the QE for Capacitive Touch tool
- Sensors can be configured for Self or Mutual mode operation
- Scans may begin by a software trigger or an external trigger

1.2 Self Mode

The driver makes use of the following terminology and definitions when in self-capacitance mode:

Self Mode
In self-capacitance mode, only one CTSU TS sensor is necessary to functionally operate a touch button/key. A series of these sensors physically aligned can be used to create a **slider**. If the series of sensors are aligned such that they create a circular pattern, this is referred to as a **wheel**.

Scan Order
In Self mode, the hardware scans the specified number of sensors in ascending order. For example, if sensors 5, 8, 2, 3, and 6 are specified in your application, the hardware will scan them in the order: 2, 3, 5, 6 and 8.

Element
An element refers to the index of a sensor within the scan order. Using the previous example, sensor number 5 is element 2.

Buffer Contents
At the lowest level, both the CTSUSC sensor count and CTSURC reference count registers are loaded into the driver buffer for each sensor in the scan configuration. Though the RC is not used, both registers are placed into the buffer for two reasons. 1) Both registers must have their contents read for proper scan operation. 2) This allows for identical processing for both interrupt and DTC operation at a later point. Note, however, that API calls such as R_CTSU_ReadData() which access this buffer will load only the sensor values.

Scan Time
The scanning of sensors occurs in the background by the CTSU peripheral and does not utilize any main processor time. It takes approximately 500us to scan a single sensor. If DTC is not used, an additional 2.2us overhead (system clock 54MHz) is added for the main processor to transfer data to/from registers when each sensor is scanned.

Methods/Scan Configurations
These terms are used interchangeably. A single method (scan configuration) refers to the set of sensors to be scanned along with what mode they are to be scanned in (Self or Mutual). One to eight methods per application can be defined within the QE for Capacitive Touch tool suite. Typically there is only one method used per application. However, more advanced applications may require different scan configurations to be enabled as different features are enabled within the product, or when a combination of Self and Mutual sensor configurations are present.

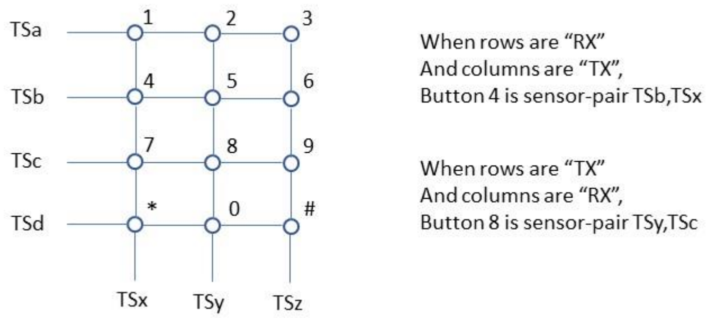
1.3 Mutual Mode

The driver makes use of the following terminology and definitions when in mutual-capacitance mode:

Mutual Mode
In mutual-capacitance mode, two CTSU TS sensors are necessary to functionally operate a single touch button/key. This mode is typically used when a matrix/keypad with more than four keys is present. This is because Mutual mode requires fewer sensors to operate than the equivalent in Self mode does.

Consider a standard phone keypad. You can think of it as a matrix of four rows and three columns. One TS sensor is shared for each row, and another sensor is shared for each column. Each button/key is identified by a **sensor-pair**. The

RX sensor is always listed first in the pair. (All row or all column sensors are designated as RX, and the others are designated TX. The peripheral scans both the RX and TX sensors in a sensor-pair, subtracts the difference, and the result is used to determine whether a button/key is touched or not.)



As you can see, only seven sensors are necessary to scan 12 buttons. In self mode, 12 sensors are required.

At this time, sliders and wheels are not supported in Mutual mode.

Scan Order

In Mutual mode, the hardware scans the sensor-pairs in ascending order, with sensors configured as RX being the **primary**, and the sensors configured as TX the **secondary**. For example, if sensors 10, 11, and 3 are specified as RX sensors, and sensors 2, 7, and 4 are specified as TX sensors, the hardware will scan them in the following sensor-pair order:

3,2 - 3,4 - 3,7
10,2 - 10,4 - 10,7
11,2 - 11,4 - 11,7

Element

In mutual-capacitance mode, an element refers to the index of a sensor-pair within the scan order. Using the previous example, sensor-pair 10,7 is element 5.

Buffer Contents

At the lowest level, both the CTSUSC sensor count and CTSURC reference count registers are loaded into the driver buffer for each sensor in a sensor-pair in the scan configuration. Though the RC is not used, both registers are placed into the buffer for two reasons. 1) Both registers must have their contents read for proper scan operation. 2) This allows for identical processing for both interrupt and DTC operation at a later point. Note, however, that API calls such as R_CTSU_ReadData() which access this buffer will load only the sensor values.

Scan Time

The scanning of sensors occurs in the background by the CTSU peripheral and does not utilize any main processor time. It takes approximately 1000us (1ms) to scan a single sensor-pair. If DTC is not used, an additional 4.4us overhead (system clock 54MHz) is added for the main processor to transfer data to/from registers when each sensor-pair is scanned.

Methods/Configurations

These terms are used interchangeably. A single method (scan configuration) refers to the set of sensors to be scanned along with what mode they are to be scanned in (self or mutual). One to eight methods per application can be defined within the QE for Capacitive Touch Tool. Typically there is only one method used per application. However, more advanced applications may require different scan configurations to be enabled as different features are enabled within the product, or when a combination of Self and Mutual sensor configurations are present.

1.4 Trigger Sources

Scanning of sensors may begin by either a software trigger or an external event initiated by the Event Link Controller (ELC). Typically, a software trigger is used.

For software triggers, a periodic timer such as the CMT is configured whose interval is large enough to allow for all sensors to be scanned and data to be updated (see *Scan Time* in sections 1.2 and 1.3). When the timer expires, the following sequence of events occurs:

- A flag is set in the timer interrupt routine
- A slight delay occurs until the main application detects that the flag has been set (red bar in Figure 1).
- Using a Touch Middleware driver, the main application makes an API call to load the scanned data, update internal values (such as moving averages), and issue a software trigger to begin another scan. Using the FIT QE Touch driver, this is done using the API function R_TOUCH_UpdateDataAndStartScan().

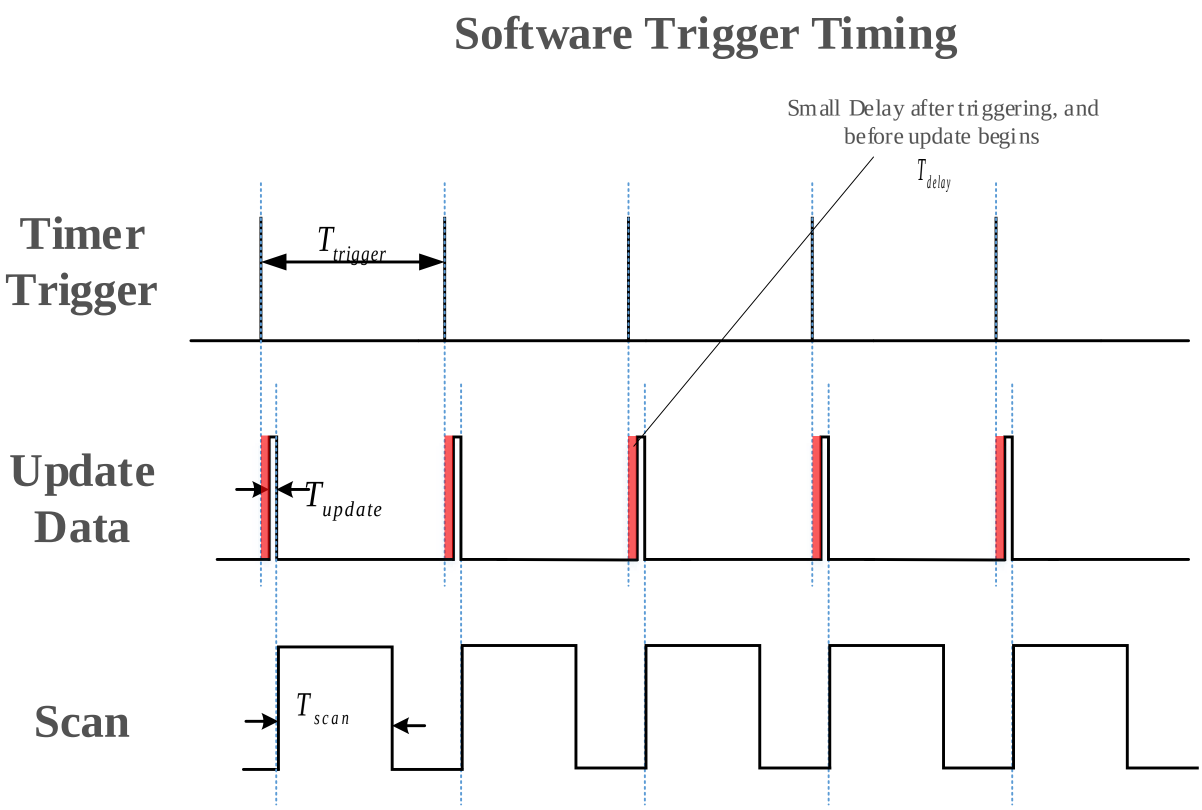


Figure 1: Software Trigger Timing Diagram (timing not to scale)

Using an external trigger is processed almost identically to using software triggers. Typically, a periodic timer such as the MTU is configured whose interval is large enough to allow for all sensors to be scanned and data to be updated (see *Scan Time* in sections 1.2 and 1.3). This timer is then linked to the ELC which in turn is linked to the CTSU. When the timer expires, the following sequence of events occurs:

- The ELC is triggered which triggers the CTSU to start a scan.
- When a scan completes, a flag is set in the CTSU scan-complete interrupt routine
- A slight delay occurs until the main application detects that the flag has been set (red bar in Figure 2.).
- Using a Touch Middleware driver, the main application makes an API call to load the scanned data and update internal values (such as moving averages). Using the FIT QE Touch driver, this is done using the API function R_TOUCH_UpdateData().

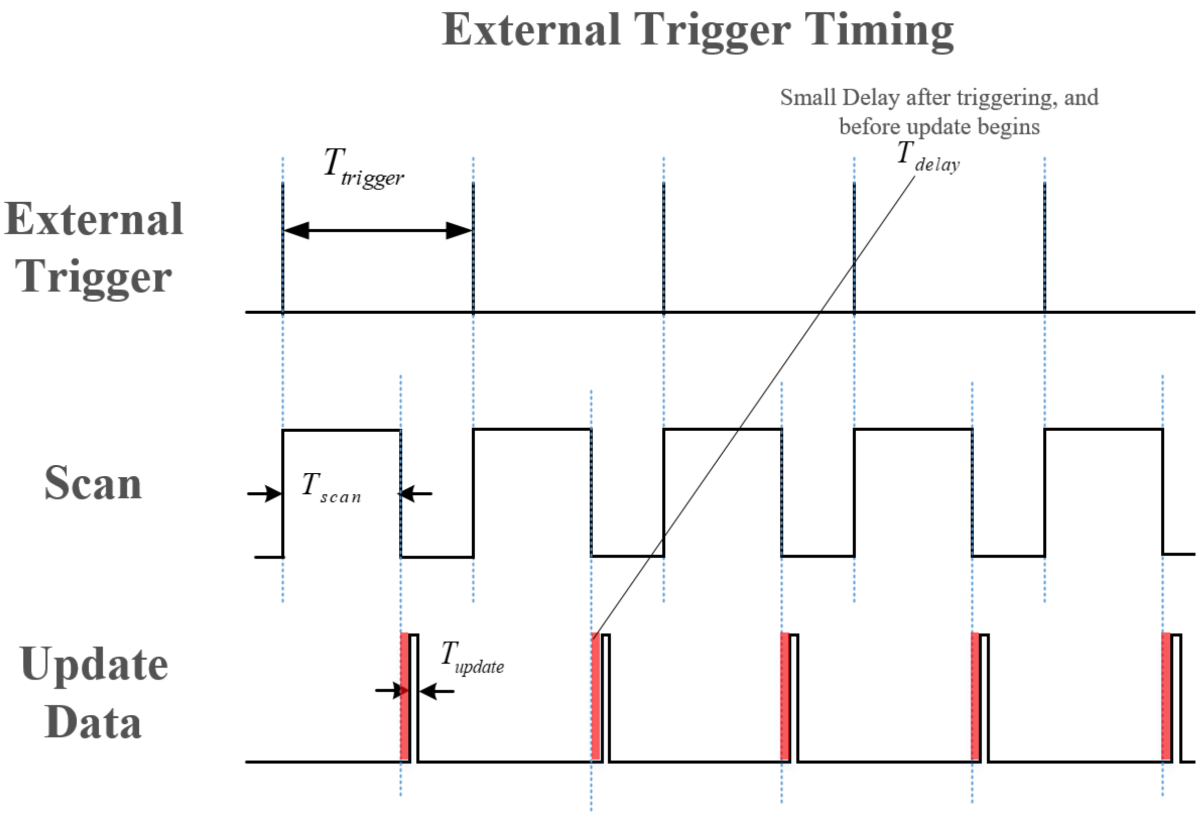


Figure 2: External Trigger Timing Diagram (timing not to scale)

With either trigger mechanism, the time it takes for the main application to detect a flag set in an interrupt routine can vary by a several microseconds with each scan depending upon the user's polling algorithm. Also, the time it takes to update the scanned data can vary by several microseconds depending upon such things as whether a sensor is touched or not (different paths through an else-if statement). In general, these subtle differences are minimal with each scan, and are extremely small compared to overall time spent performing a scan. For these reasons, the majority of users prefer the software triggering mechanism because of its simpler setup. But for those rare cases where a slight variation in scan interval is not acceptable, the external trigger mechanism can be used. **NOTE: DTC cannot be used when external triggers are used.**

2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires that your MCU supports the following peripheral(s):

- CTSU

2.2 Software Requirements

This driver is dependent upon the following FIT packages:

- Renesas Board Support Package (r_bsp) v3.71.

2.3 Limitations

- This code is not re-entrant and protects against multiple concurrent function calls.

2.4 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v2.08.00

2.5 Header Files

All API calls and their supporting interface definitions are located in “r_ctsu_qe_if.h”. This file should be included by the user’s application.

Build-time configuration options are selected or defined in the file "r_ctsu_qe_config.h”.

2.6 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in stdint.h.

2.7 Configuration Overview

Configuring this module is done through the supplied r_ctsu_qe_config.h header file. Each configuration item is represented by a macro definition in this file. Each configurable item is detailed in the table below.

Configuration options in r_flash_rx_config.h		
Equate	**Default Value**	**Description**
CTSU_CFG_PARAM_CHECKING_ENABLE	1	Setting to 0 omits parameter checking. Setting to 1 includes parameter checking.
CTSU_CFG_USE_DTC	0	Set the value to 1 to use the DTC instead of main processor for handling CTSU write and read interrupts. **Warnings:** If DTC is used elsewhere in the application, there may be conflicts with using this driver. If DTC is used, external triggers may not be used for scan trigger type.
CTSU_CFG_INT_PRIORITY_LEVEL	8	Sets the priority level of the CTSU interrupts (necessary even when using DTC). Range is 1-15, low to high.

Table 1: QE CTSU general configuration settings

2.8 Code Size

The code size is based on optimization level 2 and optimization type for size for the RXC toolchain in Section 2.4. The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options set in the module configuration header file.

ROM and RAM usage	
ROM usage: **CTSU_PARAM_CHECKING_ENABLE 1 > CTSU_PARAM_CHECKING_ENABLE 0** **CTSU_CFG_USE_DTC 1 > CTSU_CFG_USE_DTC 0**	
Minimum Size	ROM: 2836 bytes
	RAM: 410 bytes
Maximum Size	ROM: 3241 bytes
	RAM: 442 bytes

2.9 API Data Types

The API data structures are located in the file “r_typedefs_qe.h” and discussed in Section 3.

2.10 Return Values

This shows the different values API functions can return. This return type is defined in “r_typedefs_qe.h”.

```
/* Return error codes */
typedef enum e_qe_err
{
    QE_SUCCESS = 0,
    QE_ERR_NULL_PTR,            // missing argument
    QE_ERR_INVALID_ARG,
    QE_ERR_BUSY,
    QE_ERR_ALREADY_OPEN,
    QE_ERR_CHAN_NOT_FOUND,
    QE_ERR_TUNING_IN_PROGRESS,
    QE_ERR_TRIGGER_TYPE,        // function not available for trigger type
} qe_err_t;
```

2.11 Adding the QE CTSU FIT Module to Your Project

For detailed explanation of how to add a FIT Module to your project, see document R01AN1723EU "Adding FIT Modules to Projects".

2.11.1 Adding source tree and project include paths

In general, a FIT Module may be added in 3 ways:

1. Using an e2studio FIT tool, such as File>New>Renesas FIT Module (prior to v5.3.0), Renesas Views->e2 solutions toolkit->FIT Configurator (v5.3.0 or later), or projects created using the Smart Configurator (v5.3.0 or later). This adds the module and project include paths.
2. Using e2studio File>Import>General>Archive File from the project context menu.
3. Unzipping the .zip file into the project directory directly from Windows.

When using methods 2 or 3, the include paths must be manually added to the project. This is done in e2studio from the project context menu by selecting Properties>C/C++ Build>Settings and selecting Compiler>Source in the ToolSettings tab. The green "+" sign in the box to the right is used to pop a dialog box to add the include paths. In that box, click on the Workspace button and select the directories needed from the project tree structure displayed. The directories needed for this module are:

- ${workspace_loc:/${ProjName}/r_ctsu_qe
- ${workspace_loc:/${ProjName}/r_ctsu_qe/src
- ${workspace_loc:/${ProjName}/r_config

2.11.2 Setting driver options when not using Smart Configurator

The CTSU-specific options are found and edited in \r_config\r_ctsu_qe_config.h.

A reference copy (not for editing) containing the default values for this file is stored in \r_ctsu_qe\ref\r_ctsu_qe_config_reference.h.

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_CTSU_Open()	Initializes the module.
R_CTSU_StartScan()	Starts sensor scan with software trigger.
R_CTSU_ReadButton()	Reads sensor value for specified button.
R_CTSU_ReadSlider()	Reads all sensor values for specified slider or wheel.
R_CTSU_ReadElement()	Reads sensor value(s) for the specified sensor/sensor pair (mutual mode).
R_CTSU_ReadData()	Reads all sensor values.
R_CTSU_ReadReg()	Reads register.
R_CTSU_WriteReg()	Writes register.
R_CTSU_Control()	Performs special operation.
R_CTSU_Close()	Shuts down the module.
R_CTSU_GetVersion()	Returns software version of driver.

3.2 R_CTSU_Open

The function initializes the QE CTSU FIT module. This function must be called before calling any other API functions.

Format

```
qe_err_t R_CTSU_Open(ctsu_cfg_t *p_ctsu_cfgs[],
                     uint8_t    num_methods,
                     qe_trig_t  trigger);
```

Parameters

p_ctsu_cfgs

Pointer to array of scan configurations (gp_ctsu_cfgs[ ] generated by the QE for Capacitive Touch tool)

num_methods

Number of scan configurations in array (QE_NUM_METHODS generated by the QE for Capacitive Touch tool)

trigger

Scan trigger source (QE_TRIG_SOFTWARE or QE_TRIG_EXTERNAL)

Return Values

```
QE_SUCCESS:           CTSU initialized successfully.
QE_ERR_ALREADY_OPEN:  Open() called without intermediate call to Close()
QE_ERR_NULL_PTR:      Missing argument pointer.
QE_ERR_INVALID_ARG:   "num_methods" or "trigger" is invalid. Note: Configuration
                      array contents are not inspected (generated by tool).
QE_ERR_BUSY:          Cannot run because another CTSU operation is in progress.
                      Likely called Open() before Close() completed.
```

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

This function initializes the QE CTSU FIT module. This includes register initialization, enabling interrupts, and initializing the DTC if configured for operation in "r_ctsu_qe_config.h".

This function also runs an initial correction algorithm that is used for optimizing the QE Capacitive Touch tuning parameters. This is performed to account for small board-to-board variations as well as any minor environmental or physical differences in the Capacitive Touch application system.

Note that this function must be called before any other API function.

Reentrant

No.

Example

```
qe_err_t err;

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

/* Initialize the API. */
err = R_CTSU_Open(gp_ctsu_cfgs, QE_NUM_METHODS, QE_TRIG_SOFTWARE);

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

Special Notes:

Pins must be initialized prior to calling this function.

3.3 R_CTSU_StartScan

This function is used to start a sensor scan by software trigger.

Format

```
qe_err_t  R_CTSU_StartScan(void);
```

Parameters

None

Return Values

QE_SUCCESS:	*Scan started.*
QE_ERR_TRIGGER_TYPE:	*Open() specified external trigger.*
QE_ERR_BUSY:	*Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file “r_ctsu_qe_if.h”

Description

If DTC usage is enabled in “r_ctsu_qe_config.h”, this function first sets the DTC CTSU register transfer addresses, then initiates a hardware sensor scan.

Reentrant

No.

Example

```
qe_err_t err;

/* Initiate a sensor scan by software trigger */
err = R_CTSU_StartScan();

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

Special Notes:

This function should never be called when operating in external trigger mode.

3.4 R_CTSU_ReadButton

This function is used to read the button sensor value(s) from the previous scan.

Format

```
qe_err_t  R_CTSU_ReadButton(btn_ctrl_t *p_btn_ctrl,
                            uint16_t   *p_value1,
                            uint16_t   *p_value2);
```

Parameters

p_btn_ctrl

Pointer to button control structure (entry in g_buttons_xxx[ ] array generated by QE tool).

p_value1

Pointer to word to load primary sensor value into.

p_value2

Pointer to word to load secondary sensor value into. Used only when in Mutual mode.

Return Values

QE_SUCCESS: *Button value(s) read.*
QE_ERR_NULL_PTR: *Missing argument pointer.*
QE_ERR_INVALID_ARG: *Invalid element index specified in button control structure.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

This function loads the sensor value(s) for the specified button.

Reentrant

No.

Example: For configuration named PANEL, button named ONOFF, mode is SELF

```
qe_err_t err;
uint16_t sensor_val;

/* Load sensor value for button */
err = R_CTSU_ReadButton(&g_buttons_panel[PANEL_INDEX_ONOFF],
                        &sensor_val, NULL);
```

Example: For configuration named PANEL, button named ONOFF, mode is MUTUAL

```
qe_err_t err;
uint16_t primary,secondary,value;

/* Load sensor values for button */
err = R_CTSU_ReadButton(&g_buttons_panel[PANEL_INDEX_ONOFF],
                        &primary, &secondary);

value = secondary - primary;
```

Special Notes:

None.

3.5 R_CTSU_ReadSlider

This function is used to read the slider or wheel sensor values from the previous scan.

Format

```
qe_err_t  R_CTSU_ReadSlider(sldr_ctrl_t *p_sldr_ctrl,
                            uint16_t    *p_buf1,
                            uint16_t    *p_buf2);
```

Parameters

p_slider_ctrl

Pointer to slider control structure (entry in g_sliders_xxx[ ] array generated by QE tool).

p_buf1

Pointer to buffer to load primary sensor values into.

p_buf2

Pointer to buffer to load secondary sensor values into. Reserved for Mutual mode (unsupported at this time).

Return Values

QE_SUCCESS: *Slider values read*
QE_ERR_NULL_PTR: *Missing argument pointer*
QE_ERR_INVALID_ARG: *Invalid element index specified in slider control structure.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

This function loads the sensor values for the specified slider or wheel.

Reentrant

No.

Example: For configuration named PANEL, slider named DIMMER, mode is SELF

```
qe_err_t err;
uint16_t data[QE_MAX_SLDR_ELEM_USED];

/* Load sensor values from low to high for slider */
err = R_CTSU_ReadSlider(&g_sliders_panel[PANEL_INDEX_DIMMER], data, NULL);
```

Example: For configuration named PANEL, wheel named SERVO, mode is SELF

```
qe_err_t err;
uint16_t data[QE_MAX_WHEEL_ELEM_USED];

/* Load sensor values from low to high for wheel */
err = R_CTSU_ReadSlider(&g_wheels_panel[PANEL_INDEX_SERVO], data, NULL);
```

Special Notes:

Mutual mode for sliders is not supported at this time.

3.6 R_CTSU_ReadElement

This function is used to read a specific element from the previous scan.

Format

```
qe_err_t  R_CTSU_ReadElement(uint8_t     element,
                             uint16_t    *p_value1,
                             uint16_t    *p_value2);
```

Parameters

element

Index of sensor/sensor pair (mutual mode) to read.

p_value1

Pointer to word to load primary sensor value into.

p_value2

Pointer to word to load secondary sensor value into. Used only when in Mutual mode.

Return Values

QE_SUCCESS: *Element values read*
QE_ERR_NULL_PTR: *Missing argument pointer*
QE_ERR_INVALID_ARG: *Invalid element index.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

This function loads the sensor values for the specified element index.

Reentrant

No.

Example: Mode is SELF

```
qe_err_t err;
uint16_t sensor_val;

/* Load sensor value for element 4 */
err = R_CTSU_ReadElement(4,&sensor_val, NULL);
```

Example: Mode is MUTUAL

```
qe_err_t err;
uint16_t primary,secondary,value;

/* Load sensor values for element 7 */
err = R_CTSU_ReadElement(7, &primary, &secondary);

value = secondary - primary;
```

Special Notes:

None.

3.7 R_CTSU_ReadData

This function is used to read all sensor data from the previous scan.

Format

```
qe_err_t  R_CTSU_ReadData(uint16_t   *p_buf,
                          uint16_t   *p_cnt);
```

Parameters

p_buf

Pointer to buffer to load all sensor counter values into (does not include reference counter values).

p_cnt

Pointer to word to load number of words read. Mutual mode will read twice as many words as Self mode.

Return Values

QE_SUCCESS: *Data values read*
QE_ERR_NULL_PTR: *Missing argument pointer*
QE_ERR_INVALID_ARG: *Invalid element index.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file “r_ctsu_qe_if.h”

Description

This function loads into the buffer all sensor values from the previous scan, and provides the number of words read. This count will equal the number of sensors used in Self mode, or equal twice the number of sensor-pairs when in Mutual mode.

Reentrant

No.

Example: For configuration named PANEL, mode is SELF

```
qe_err_t err;
uint16_t buf[PANEL_NUM_ELEMENTS];
uint16_t cnt;

/* Load all sensor values */
err = R_CTSU_ReadData(buf,&cnt);
```

Example: For configuration named PANEL, mode is MUTUAL

```
qe_err_t err;
uint16_t buf[PANEL_NUM_ELEMENTS*2];
uint16_t cnt;

/* Load all sensor values */
err = R_CTSU_ReadData(buf, &cnt);
```

Special Notes:

None.

3.8 R_CTSU_ReadReg

This function is used to read the current register value.

Format

```
qe_err_t  R_CTSU_ReadReg(ctsu_reg_t reg, uint8_t element, uint16_t *p_value);
```

Parameters

reg

ID of register to read.

element

For CTSCUSSC, SO0, SO1 only; element associated with desired sensor/sensor pair (mutual mode).

p_value

Pointer to word to load register value into.

Return Values

QE_SUCCESS: Register successfully read
QE_ERR_NULL_PTR: Missing argument pointer
QE_ERR_INVALID_ARG: Invalid register ID or element index.
QE_ERR_BUSY: Cannot run because another CTSU operation is in progress.

Properties

Prototyped in file “r_ctsu_qe_if.h”

Description

In general, this function loads the word pointed to by “p_value” with the contents of the CTSU hardware register specified by “reg”.

In the case of the CTSUSSC, SO0, and SO1 registers, a saved configuration value specific to each sensor/sensor-pair is loaded (the hardware registers contain only the value for last sensor/sensor-pair scanned). The “element” argument specifies which stored configured value to load. See *Element* description in Sections 1.2 and 1.3.

Reentrant

No.

Example: Common register read

```
qe_err_t err;
uint16_t value;

/* Read the CTSU Error Status Register into “value” (“element” unused) */
err = R_CTSU_ReadReg(CTSU_REG_ERRS, 0, &value );
```

Example: Sensor-specific register read

```
qe_err_t err;
uint16_t value;

/* Read the CTSUSSC configured value associated with element 4 into “value” */
err = R_CTSU_ReadReg(CTSU_REG_SSC, 4, &value );
```

Special Notes:

None.

3.9 R_CTSU_WriteReg

This function writes the passed in value to the specified hardware register, except for the CTSUSSC, SO0, and SO1 cases where the value is written to the configuration variables associated with the specified key instead.

Format

```
qe_err_t  R_CTSU_WriteReg(ctsu_reg_t reg, uint8_t element, uint16_t value);
```

Parameters

reg

ID of register to write.

element

For CTSCUSSC, SO0, SO1 only; element associated with desired sensor/sensor pair (mutual mode).

value

Value to write to register

Return Values

QE_SUCCESS: *Register successfully written*
QE_ERR_INVALID_ARG: *Invalid register ID or element index.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

In general, this function writes "value" to the CTSU hardware register specified by "reg".

In the case of the CTSUSSC, SO0, and SO1 registers, "value" is saved internally by the driver for the sensor/sensor-pair specified by "element" (see *Element* description in sections 1.2 and 1.3). This value is later written to the hardware register just before the sensor/sensor-pair is scanned.

Reentrant

No.

Example: Common register write

```
qe_err_t err;
uint16_t value = 0x04;

/* Write "value" to CTSU_REG_CR0 ("element" unused) */
err = R_CTSU_WriteReg(CTSU_REG_CR0, 0, value );
```

Example: Sensor-specific register write

```
qe_err_t err;
uint16_t value = 0x1D21;

/* Update CTSUSO0 value associated with element 4 */
err = R_CTSU_WriteReg(CTSU_REG_SO0, 4, value );
```

Special Notes:

None.

3.10 R_CTSU_Control

This function processes driver special-operation commands.

Format

```
qe_err_t  R_CTSU_Control(ctsu_cmd_t cmd, uint8_t method, uint16_t *p_arg);
```

Parameters

cmd

Command to perform.

method

Method to perform the command on.

p_arg

Pointer to arguments specific to the command, can also be NULL.

Return Values

```
QE_SUCCESS:          Command Successfully Completed
QE_ERR_NULL_PTR:     Missing required argument pointer (p_arg)
QE_ERR_INVALID_ARG: Invalid command or unknown method
```

Properties

Prototyped in file “r_ctsu_qe_if.h”

Description

This function is used to perform special operations with the CTSU driver. The “cmd” commands are as follows:

CTSU_CMD_SET_METHOD. Used to change the method/scan configuration currently in use. Equates for “method” are defined in “qe_common.h” and have the form QE_METHOD_xxx. “p_arg” is unused.

CTSU_CMD_SET_CALLBACK. Used with external triggers to override the default QE generated function *qetouch_timer_callback()* which is called after each scan completes. “method” argument is unused. “p_arg” is name of new callback function.

CTSU_CMD_GET_STATE. Indicates the current state of the driver (where it is in scan process). “method” argument is unused. “p_arg” is a pointer to a variable of type “ctsu_state_t”.

CTSU_CMD_GET_STATUS. Provides error and processing-complete flags for specified method. Equates for “method” are defined in “qe_common.h” and have the form QE_METHOD_xxx. “p_arg” is pointer to a variable of type “ctsu_status_t”.

CTSU_CMD_CLR_UPDATE_FLG. Used to notify the CTSU driver that scanned data has been processed by the Touch layer. Equates for “method” are defined in “qe_common.h” and have the form QE_METHOD_xxx. “p_arg” is unused.

Reentrant

No.

Example: CTSU_CMD_SET_METHOD

```
qe_err_t err;

/* Change method/scan configuration (methods defined in qe_common.h) */
err = R_CTSU_Control(CTSU_CMD_SET_METHOD, QE_METHOD_FULLPOWER, NULL );
```

Example: CTSU_CMD_SET_CALLBACK

```
qe_err_t err;
```

```
/* Change default callback function to ext_trig_scan_complete_cb() */
err = R_CTSU_Control(CTSU_CMD_SET_CALLBACK, 0, ext_trig_scan_complete_cb)
```

Example: CTSU_CMD_GET_STATE

```
qe_err_t     err;
ctsu_state_t current_state;

/* Get the current scan state of the driver */
err = R_CTSU_Control(CTSU_CMD_GET_STATE, 0, &current_state);
```

Example: CTSU_CMD_GET_STATUS

```
qe_err_t      err;
ctsu_status_t current_status;

/* Get status flags for method (methods defined in qe_common.h) */
err = R_CTSU_Control(CTSU_CMD_GET_STATUS, QE_METHOD_PANEL2, &current_status);
```

Example: CTSU_CMD_CLR_UPDATE_FLG

```
qe_err_t      err;
ctsu_status_t current_status;

/* Clear touch-processing-complete update-flag for method */
err = R_CTSU_Control(CTSU_CMD_CLR_UPDATE_FLG, QE_METHOD_PANEL1, NULL);
```

Special Notes:

None.

3.11 R_CTSU_Close

This function shuts down the CTSU peripheral.

Format

```
qe_err_t  R_CTSU_Close(void);
```

Parameters

None.

Return Values

QE_SUCCESS: *The CTSU peripheral is successfully closed.*
QE_ERR_BUSY: *Cannot run because another CTSU operation is in progress.*

Properties

Prototyped in file “r_ctsu_qe_if.h”

Description

This function closes the CTSU driver and shuts down the peripheral. It disables interrupts associated with the CTSU and disables the clock to the peripheral.

Reentrant

No.

Example:

```
qe_err_t err;

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

Special Notes:

None.

3.12 R_CTSU_GetVersion

Returns the current version of the QE CTSU FIT module.

Format

```
uint32_t R_CTSU_GetVersion(void);
```

Parameters

None.

Return Values

Version of the CTSU FIT module.

Properties

Prototyped in file "r_ctsu_qe_if.h"

Description

This function returns the version number of the currently installed QE CTSU driver. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number. For example, Version 4.25 would be returned as 0x00040019.

Reentrant

Yes.

Example

```
uint32_t cur_version;

/* Get version of installed CTSU API. */
cur_version = R_CTSU_GetVersion();

/* Check to make sure version is new enough for this application's use. */
if (MIN_VERSION > cur_version)
{
    /* This QE CTSU API version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ...
}
```

Special Notes:

This function is specified to be an inline function.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Oct.04.18	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0