RX200, RX100 Series

Touch Module Using Firmware Integration Technology

R01AN3820EU0100
Rev 1.00
May 9, 2017

Introduction

This Firmware Integration Technology (FIT) Module implements a Capacitive Touch input detection middleware.

Target Device

RX231, RX230, RX130, RX113 with onboard capacitive touch sensing units.

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- CTSU Module Using Firmware Integration Technology (R01AN3821EU)
- Touch Button Module Using Firmware Integration Technology (R01AN3822EU)
- Touch Slider Module Using Firmware Integration Technology (R01AN3823EU)
- Workbench6 User Manual (R20UT3842EJ)
- Capacitive Touch for RX Devices (R11AN0139EU)

Contents

1. Overview of the Capacitive Touch Software Environment

The Capacitive Touch Sensing Unit (CTSU) measures the electrostatic capacitance of a touch sensor. The entirety of the driver stack is comprised of three different components: The CTSU layer, the touch layer, and the button/slider layer. The module covered in this documentation provides an abstraction layer for interpreting the data from the CTSU as touch events. This includes tunable parameters such as thresholds, hysteresis values, and debounces each sensor channel. It is intended for this FIT module to work with the button module to further enhance the touch functionality. It comes with python (2.7.13) scripts; these translate optimized parameters generated by the Renesas Workbench calibration tool to data that can be input to this driver. For more information on the translation process, using python refer to R11AN0139EU.

2. CTSU FIT Module

This module is intended for use along with the Renesas CTSU (r_ctsu_rx) lower level driver and button (r_touch_button) middleware as shown in the illustration below and is used by being incorporated into a FIT project when there is a need to interpret the digital counts from the CTSU layer into binary touch events.

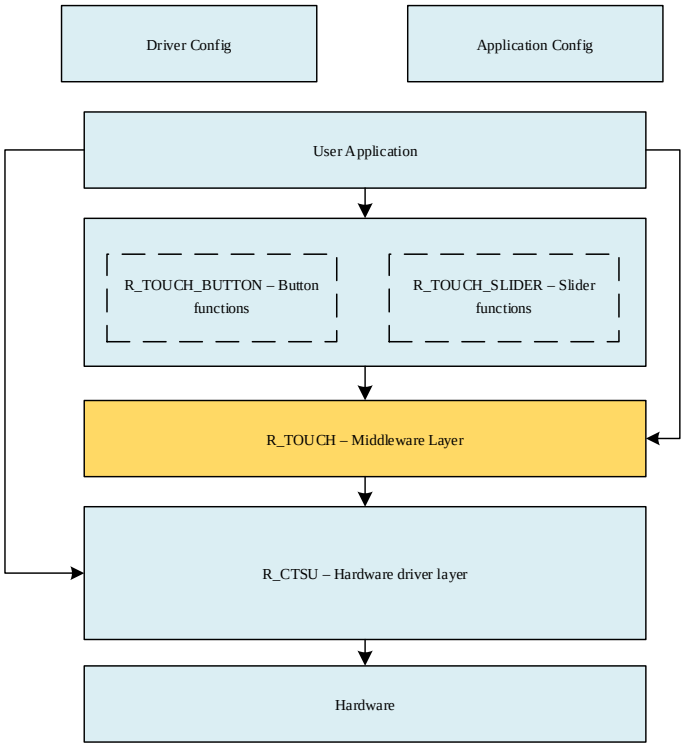

Figure 1: Overview of the capacitive touch solution and the various FIT modules

The block diagram describing how the middleware interacts with the user application is described in Figure 1, the highlighted part is the r_touch FIT module which is described as the touch layer. The output of this layer is a binary touch event that supplies the application and buttons layers with the means for interpreting a touch event. For API documentation on the CTSU, button and slider layers refer to the following documents respectively- R01AN3821EU, R01AN3822EU, and R01AN3823EU.

Similar to the configuration of other MCU peripherals, such as the analog to digital converter, the first step to configure the CTSU after power on reset is to configure the appropriate pins as touch sensing (TS) and TSCAP pins. After proper input/output (I/O) initialization, the user will then have to open the appropriate touch configuration provided by Workbench, referred to as a touch configuration block (TCB). The configuration block contains all the necessary information to operate a group of touch sensors using and estimate the presence of a user touch. The open function processes the TCB and provides an index called a handle to the user specified memory location, provided as an argument to the function. This command also invokes the R_CTSU_Open function initializing the underlying sensor configuration referenced in R01AN3821EU. This means that the firmware only needs to invoke R_TOUCH_Open if not using the CTSU API. When both functions are invoked, it only updates the information at the CTSU layer and does not result in a duplication of memory.

After the open function has been invoked, the software should update the touch control block, perform a scan, and read the newly scanned information into the control block for manipulation. An overview of the process is shown in Figure 2.

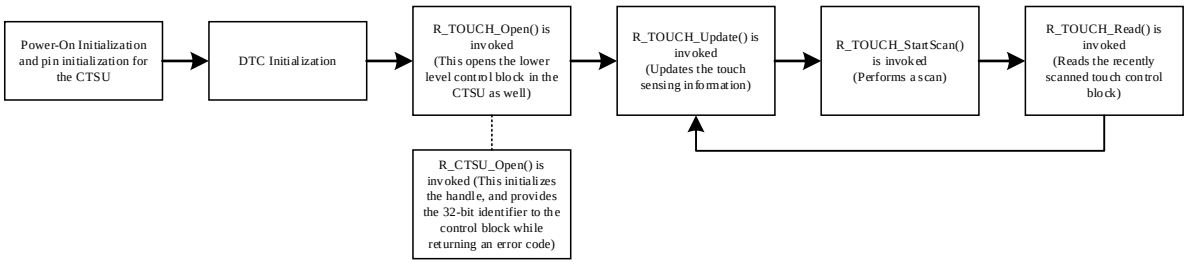

Figure 2: Example process of using the r_touch module

The value of this layer of the driver is the variety of tuning parameters, which can be modified to make the design more robust in different environments and physical layouts; more about custom tuning a Workbench application is found in, R30AN0291EU.

3. FIT Module Specifications

- The Open function allows multiple instances of Touch configurations to be opened using control blocks for storing all relevant information.
- Provides a binary bit-string of data where each bit represents a channel (ascending order) being touched (logic 1) or not (logic 0). For e.g. if TS4, TS5, TS8, and TS10 are enabled, then the 0th bit set to logic 1 represents TS4 being touched. Likewise, 1st bit represents TS5, 2nd bit represents TS8, and 3rd bit represents TS10.
- Allows access to all result buffers maintained, such as filtered output, sensor baselines, etc.
- Data manipulation phases include filtering (User defined functions may be provided), Touch detection (User defined functions may be provided), and Drift compensation. These phases can be disabled and enabled on a per sensor basis.
- Drift compensation is implemented to maintain sensitivity with changing humidity/ temperature on the board. It can be enabled on a per channel basis, can perform a variable drift rate change (i.e. drift compensation interval speeds up as sensor is approaching touch and vice versa), and implements drift hold-time (i.e. disable drift compensation for all sensors for N scans after a sensor is touched).
- Implements debounces such as, delay-to-touch (detect integrator) and delay-to-release.
- Implements max-on limiter i.e. release all sensors if touched for too long.
- Each sensor maintains a min and max value while not touched. The difference between the min and max can be used for establishing a signal to noise ratio.
- Periodic run-time fine tuning iteration is triggered if input signal has deviated away from average by N counts for M scans. M and N can be specified.
- Error messages assertion to the Renesas Virtual Debug Console by setting BSP_CFG_IO_LIB_ENABLE == 1 in the Board Support Package and enabling parameter checking.

4. API Information

This driver follows the Renesas API naming standards.

4.1 Hardware Requirements

This driver requires that your MCU support the following features:

- CTSU peripheral
- (Optional) DTC

4.2 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_ctsu

4.3 Limitations to this driver include:

- This module requires the CTSU FIT module to be included, along with that modules dependencies.

4.4 Supported Toolchains

This driver is tested and works with the following toolchain:

- Renesas RX Toolchain v.2.05

4.5 Header Files

All API calls and their supporting interface definitions are located in r_ctsu_rx_if.h

4.6 Integer Types

This module uses ANSI C99. These types are defined in stdint.h

4.7 Configuration Overview

The configuration options in this module are specified in r_touch_rx_config.h. The option names and setting values are listed in the table below where applicable 0 represents disabled and 1 represents enabled:

Option	Description
TOUCH_CFG_PARAM_CHECKING_ENABLE	Specify whether to include code for API parameter checking. (Default = Enabled)
TOUCH_CFG_MAX_CONTROL_BLOCK_COUNT	Define maximum control blocks the user is allowed to open. (Default=1)
TOUCH_CFG_MAX_UPPER_LAYER_CALLBACKS	(0 to N)= Specifies the maximum number of callbacks for upper layers.
TOUCH_CFG_ENABLE_DRIFT_HOLD_TIME	0= Disabled (Default) 1= Drift Compensation for all sensors is paused for user specified iterations when a sensor is touched
TOUCH_CFG_FILTER_DEPTH	Length of the averaging filter used on touch counts
TOUCH_CFG_VARIABLE_DRIFT_RATES	0= Drift Compensation rate does not change (Default) 1= Drift Compensation rate decreases as sensor moves toward touch and increases as sensor moves away from touch.
MULTITOUCH_REJECTION_TYPE	0= Multi-touch rejection disabled (Default) 1= Multi-touch rejection is enabled when total number of touched sensors exceeds user specified amount 2= Multi-touch rejection is enabled when total number of simultaneous touched sensors exceeds user specified amount.

4.8 Code Size

The table lists values when the compile options are set to default values (with parameter checking disabled) and the RX130 Group is used. The required memory size varies depending on the C compiler version and compile options. Table does not reflect memory consumed by the configuration as it varies depending upon the sensor count.

Memory Type	Size (bytes)	Remarks
ROM	3725	Constant + Program + Initialized Data
RAM	173	Data + Uninitialized Data
Maximum User Stack usage	360	From R_TOUCH_Open
Maximum Interrupt Stack usage	–	Application dependent

4.9 Parameters and Header files

This section describes the structure instances, which are defined by the user while creating a touch configuration. This structure is located in r_ctsu_rx_if.h as are the prototype declarations of API functions. The structure that outlines the touch configuration is contained below.

```
typedef struct st_touch_cfg_t
{
    ctsu_cfg_t const * const p_ctsu_cfg;        // Underlying CTSU H/W
                                                // configuration.

    touch_common_parameter_t * const p_common;  // Parameters common to all
                                                // sensors

    touch_sensor_parameter_t * const p_sensor;  // Array containing parameters
                                                // related to touch.

    uint8_t * const p_binary_result;            // Pointer to location holding
                                                // binary results.

    fit_callback_t p_callback;                  // Pointer to function to use
                                                // as notification function.

    uint8_t num_ignored;                        // Count of sensors to ignore.

    touch_sensor_t const * const p_ignored;     // Sensor combinations to
                                                // ignore.

    touch_buffer_t buffer;                      // Buffer for maintain touch-
                                                // related parameters

    touch_custom_actions_t custom;              // User provided actions to
                                                // replace default actions
}touch_cfg_t;
```

Parameters that are common across multiple sensors are contained in the structure below.

```
typedef struct st_common_parameter
{
    uint16_t const drift_hold_limit;    // # of iterations the drift comp. is
                                       // disabled after touch.

    uint16_t const on_limit;            // Limiter for Touch-ON time
                                        // (Disabled =  65535)

    uint8_t const max_touched_sensors;  // Defines # of sensors allowed to be
                                       // touched simultaneously.
}touch_common_parameter_t;
```

Parameters that are custom to the application, such as a custom touch filter that operates on the CTSU counts are shown below.

```
typedef struct st_custom_actions
{
    void (*p_filter)(void*);            // Pointer to function to use for
                                       //  filtering CTSU data.

    bool (*p_touch_detect)(void*)       // Pointer to function to use for
                                       // Touch Detection.

    void ** const p_filter_instance;    // Array containing addresses of
                                        // filter control blocks.

    uint16_t num_filter_instances;      // Number of filter blocks.
}touch_custom_actions_t;
```

Parameters that are specific to each touch sensing channel are shown below.

```
typedef struct st_touch_parameter
{
    uint16_t const threshold;          // Threshold count of sensor

    uint16_t const hysteresis;         // Hysteresis count of sensor

    uint8_t  const dt_limit;           // Delay to touch limit (Immediate = 0)

    uint8_t  const dr_limit;         // Delay to release limit (Immediate = 0)

    uint16_t const drift_rate;        // # Scans between drift compensation
                                      //(Disabled = 0)

    uint8_t  const drift_rate_plus;     // drift rate accelerator (+ve rate
                                      // change disabled = 0)

    uint8_t  const drift_rate_minus;    // drift rate decelerator (-ve rate
                                      // change disabled = 0)

    uint32_t const recalib_delay;       // Delay to perform tuning of un-
                                       // touched
                                      // sensor (Disabled = 65535)

    uint16_t  const recalib_threshold;  // Expressed as raw counts beyond
                                      // which sample will trigger increase in
                                      // recalib_counter.
}touch_sensor_parameter_t;
```

The structure used to identify touch sensors via their individual rx and tx channels is shown below.

```
typedef struct st_sensor_ignored
{
    uint8_t rx;                         // Touch Sensor set as receive
                                        // electrode

    uint8_t tx;                         // Touch Sensor set as transmit
                                       // electrode (0xff = Not used)
}touch_sensor_t;
```

The following structure is used to provide buffer memory for a touch configuration.

```
typedef struct st_buffer_memory
{
    volatile uint8_t * const p_start;   // Pointer to start of memory

    volatile uint8_t * p_end;           // Pointer to end of memory (optional)

    size_t size;                        // Size of memory
}touch_buffer_t;
```

4.10 Return Values

This section describes the return values (error codes) of API functions. This enumeration is located in r_touch_rx_if.h as are the prototype declarations of API functions. Each touch function has the ability to return an error code to aid in debugging.

```
typedef enum e_touch_err
{
    TOUCH_SUCCESS,                  // No Errors.

    TOUCH_ERR_INVALID_PARAM,        // Received null pointer for required
                                   // argument.

    TOUCH_ERR_INSUFFICIENT_MEMORY,  // No more control blocks available.

    TOUCH_ERR_LOCKED,               // Control block is in use.

    TOUCH_ERR_CTSU_LOCKED,          // Attempted operation while CTSU is
                                    // scanning.

    TOUCH_ERR_INVALID_CMD,          // Command requested is invalid.

    TOUCH_ERR_CTSU_BAD_SCAN,        // Error in CTSU scan.

    TOUCH_ERR_AUTO_TUNE_FAILED,     // Calibration algorithm failed.
} touch_err_t;
```

4.11 Events

The touch layer receives events from the lower CTSU layer. The touch layer also generates events in response to some of the lower level events and adds additional event conditions. Upper layers are notified when a callback function is specified. Some events require specific actions in response to the condition. This layer generates the following events.

Event Type	Notes
TOUCH_EVENT_REQUEST_DELAY	The user must generate a 1 millisecond delay in response to this event. The information content provided in this callback contains the number of sensors being scanned.
TOUCH_EVENT_SCAN_STARTING	Generated before a hardware scan starts using R_TOUCH_StartScan. The information content provided in this callback contains the number of sensors being scanned.
TOUCH_EVENT_SCAN_STARTED	Generated after a scan has been started in hardware. The information content provided in this callback contains the number of sensors being scanned.
TOUCH_EVENT_SCAN_COMPLETE	Generated when hardware finishes measuring all sensors. The information content provided in this callback contains the touch error codes.
TOUCH_EVENT_PARAMETERS_UPDATED	Generated when all parameters related to touch detection have been updated. The information content provided in this callback contains the number of sensors determined as touched.

When using the touch layer, the callback function provided to the CTSU layer will not be invoked. When the callback function to the upper layer is invoked, a single argument pointer is provided. This pointer should be cast to the following structure type to access the event code:

```
typedef struct st_touch_callback_arg
{
    int handle_num;     // Handle which generated this event.
    uint32_t event;     // Event code.
    uint32_t info;          // Error code or additional information
}touch_callback_arg_t;
```

As seen above, the event code is located in the second long word.

4.12 Adding the FIT module to your project

This section outlines the process for integrating the generated FIT style drivers into an already created project, after the board has been tuned using Workbench6. For information regarding installation for the entirety of the driver stack, please refer to (R11AN0138EU).

1. Ensure that the User Project uses FIT, and is based on r_bsp v3.40 or later.
2. Copy the modules r_touch_rx to the FIT project from the Base Project.
3. Add the following locations to the compile include paths: "${workspace_loc:/${ProjName}/r_touch_rx}"

   And then copy the following files from the Base Project's r_config folder to the User Project: r_touch_rx_config.h
4. Ensure that the Parameter Checking preprocessor is enabled for all layers using the files mentioned in item 4. Note: This can be turned off once all layers are verified to be operate correctly.
5. Copy the folder and src/ctsu from the Base Project and place them in the User Project.
6. Ensure that the clock settings in r_bsp_config.h match for both the Base and User Project.
7. Initialize all CTSU pins using the r_mpc_rx and r_gpio_rx FIT modules.

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the "Board Support Package Module Using Firmware Integration Technology" application note (R01AN1685EU).

5. API Functions

5.1 Summary

Table below lists the API functions.

Function	Description
R_TOUCH_Open	Initialize memory local to the module, i.e. a control block, and save argument parameters. Argument acting as buffer memory must remain valid until control block is closed.
R_TOUCH_Close	Close a control block that was previously opened.
R_TOUCH_Read	Read parameter(s) maintained by the control block.
R_TOUCH_StartScan	Request lower level driver to start a hardware scan.
R_TOUCH_Update	Update parameters for touch detection using the latest values generated by the CTSU.
R_TOUCH_Control	Change parameter values of a control block.

5.2 R_TOUCH_Open

This function is required to be invoked first before any other API calls with this module are used.

Format

```
touch_err_t R_TOUCH_Open (uint32_t * p_handle, touch_cfg_t const *const
p_touch_cfg)
```

Parameters

- p_handle - Pointer to a user specified location where the control block index (handle) will be stored. The user should use this control block number to perform all associated actions.
- p_touch_cfg - Pointer to a Touch configuration data and structures created by the calibration process.

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_INSUFFICIENT_MEMORY – Control block not available for use
- TOUCH_ERR_INVALID_PARAMETER – Found invalid parameter in arguments

Properties

Prototyped in r_touch_rx_if.h.

Description

Saves the configuration information into an empty control block and returns a handle at user specified location. Enables lower level CTSU hardware module and initializes necessary data in the handles.

1. Check the configuration block provided.
2. Open the lower level hardware driver and get lower level identifier.
3. Populate offsets for transmit and receive electrodes.
4. Get resource control block.
5. Copy information from configuration block to resource block.
6. Initialize buffers for drift compensation and sensitivity optimization.

Re-entrant

Function is re-entrant for different configurations passed.

Example

```
void my_func(void)
{
    uint32_t hdl_idx = UINT32_MAX;
    touch_cfg_t *    p_touch_cfg;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;
    p_ctsu_cfg = &g_ctsu_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
        // Actions on a successful open
    }
}
```

Special Notes

None.

5.3 R_TOUCH_Close

Close the user specified control block.

Format

```
touch_err_t R_TOUCH_Close (uint32_t hdl_num)
```

Parameters

- hdl_num - Handle returned by a successful call to R_TOUCH_Open

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_ERR_LOCKED – Control block is currently in use

Properties

Prototyped in r_touch_rx_if.h.

Description

1. Check the handle identifier provided and select resource block (exit if resource block is locked).
2. Close the lower level hardware driver.
3. Release resource control block.

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t hdl_idx = UINT32_MAX;
    touch_err_t touch_err;
    touch_cfg_t *    p_touch_cfg;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;
    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
        /*The hdl_idx holds the value passed back by the Open() function*/
        touch_err  = R_TOUCH_Close(hdl_idx);
    }
}
```

Special Notes

None.

5.4 R_TOUCH_StartScan

Perform a scan with the provided control block.

Format

```
touch_err_t R_TOUCH_StartScan (uint32_t hdl_num)
```

Parameters

- hdl_num - Handle assigned by the R_TOUCH_Open.

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_ERR_LOCKED – Control block is currently in use

Properties

Prototyped in r_touch_rx_if.h.

Description

1. Check the handle identifier provided and select resource (exit if resource block is locked).
2. Start scan with the lower level hardware driver.
3. Return immediately.

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t hdl_idx = UINT32_MAX;
    touch_err_t touch_err;
    touch_cfg_t *    p_touch_cfg;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;
    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
            touch_err  = R_TOUCH_StartScan(hdl_idx);
            delay();
    }
}
```

Special Notes

Common Callback releases the lock in the control block and will allow the next scan to be performed.

5.5 R_TOUCH_Update

Update Touch Sensing information after a control block has been successfully scanned.

Format

```
touch_err_t R_TOUCH_Update (uint32_t hdl_num)
```

Parameters

- hdl_num - Handle returned by a successful call to R_TOUCH_Open.

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_ERR_LOCKED – Control block is currently in use

Properties

Prototyped in r_touch_rx_if.h.

Description

Upon invocation of this function, it checks the handle identifier provided and selects a resource (exit if resource block is locked). It then performs the following:

1. Filter data.
2. Perform touch detection
3. Multi-Touch Rejection
4. (Optional) Maximum Touch-On Limiting.
5. (Optional) Drift Hold-time
6. Upper level Callback notification
7. Perform drift compensation
8. Perform sensitivity optimization

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t hdl_idx = UINT32_MAX;
    touch_err_t touch_err;
    touch_cfg_t *    p_touch_cfg;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;
    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
            touch_err  = R_TOUCH_StartScan(hdl_idx);
            delay();
           touch_err  = R_TOUCH_Update(hdl_idx);
    }
}
```

Special Notes

None.

5.6 R_TOUCH_Read

Read results updated with R_TOUCH_Update. Used for creating copies of data locally maintained by the driver to memory location provided by user.

Format

```
touch_err_t R_TOUCH_Read (uint32_t id, touch_read_t const * const p_arg)
```

Parameters

- hdl_num - Handle returned by a successful call to R_TOUCH_Open.
- p_arg - Structure specifying information to read.

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_ERR_LOCKED – Control block is currently in use
- TOUCH_ERR_INVALID_CMD – Read command not recognized
- TOUCH_ERR_INVALID_PARAM – Invalid parameter found in argument
- TOUCH_ERR_INSUFFICIENT_MEMORY – Memory of size provided at p_dest is not sufficient to copy results.

Properties

Prototyped in r_touch_rx_if.h.

Description

The additional argument taken by this function has a structure as shown below:

```
typedef struct st_touch_results
{
    touch_data_t read_cmd;          // Result type to read
    touch_sensor_t * const sensor;  // Channels to read when @ref count is
non-
                                  // zero
    uint8_t sensor_count;           // Number of sensors
    void * p_dest;                  // Location where results must be copied
    size_t size;                    // Amount of memory available at location
}touch_read_t;
```

The read function creates a copy of touch parameters maintained by the CTSU at the location p_dest. The type of result to read must always be provided by the user. If sensor combination information pointer is provided as FIT_NO_PTR, then all combinations for the results are read. The channel array has the definition as shown below:

```
typedef struct st_sensor_ignored
{
    uint8_t rx; // Touch Sensor set as receive electrode
    uint8_t tx; // Touch Sensor set as transmit electrode (0xff = Not used)
}touch_sensor_t;
```

The amount of data returned will be of (data type * sensor_count). The following read commands are supported by this function:

Command	Data Type	Description
TOUCH_DATA_BINARY	N x UINT8	Read binary string of touch determination for selected channels only.
TOUCH_DATA_CTSU_HW_IDENTIFIER	UINT32	Read CTSU Hardware Identifier returned by lower layer.
TOUCH_DATA_OPERATION_FLAGS	UINT32	Read operation enable/disable flags
TOUCH_DATA_COMMON_DRIFT_HOLD_COUNTER	UINT16	Read counter measuring drift hold time since last channel touched.
TOUCH_DATA_COMMON_DRIFT_HOLD_INTERVAL	UINT16	Read interval for disabling drift compensation for all channels.
TOUCH_DATA_COMMON_CONTINUOUS_ON_COUNTER	UINT16	Read counter measuring the amount of time a sensor is touched.
TOUCH_DATA_COMMON_CONTINUOUS_ON_INTERVAL	UINT16	Read the interval after which all touched sensors are released.
TOUCH_DATA_FILTERED_COUNT	UINT16	Read sensor count values output from filter for provided channels.
TOUCH_DATA_DELTA_COUNT	UINT16	Read out the difference between sensor and baseline count to p_dest array for selected channels.
TOUCH_DATA_BASELINE_COUNT	UINT16	Read the current sensor baseline used to determine if a channel is being touched for selected channels.
TOUCH_DATA_DELAY_TO_TOUCH_COUNTER	UINT16	Read value of delay to touch counter
TOUCH_DATA_DELAY_TO_RELEASE_COUNTER	UINT16	Read value of delay to release counter
TOUCH_DATA_AVERAGE_INPUT	UINT16	Read average input value during drift interval
TOUCH_DATA_MINIMUM_INPUT	UINT16	Read minimum value input during drift interval
TOUCH_DATA_MAXIMUM_INPUT	UINT16	Read maximum value input during drift interval
TOUCH_DATA_DRIFT_COUNTER	UINT16	Read value of counter counting up to drift interval
TOUCH_DATA_DRIFT_INTERVAL	UINT16	Read value of drift interval
TOUCH_DATA_AVG_SEN_COUNTER_PRI	UINT16	Read average value of primary sensor counter
TOUCH_DATA_AVG_REF_COUNTER_PRI	UINT16	Read average value of primary reference counter
TOUCH_DATA_RECALIBRATION_COUNTER	UINT16	Read value of re-calibration counter

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t    hdl_idx = UINT32_MAX;
    touch_err_t  touch_err;
    touch_cfg_t *    p_touch_cfg;
    touch_read_t read_arg;
    uint16_t    touch_result;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;

    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;

    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
            // More actions on successful open

            touch_err  = R_TOUCH_StartScan(hdl_idx);
            delay();
            touch_err = R_TOUCH_Update(hdl_idx);
            read_arg.p_dest = &touch_result;
            read_arg.size = sizeof(sensor_result);
            read_arg.sensor = CHANNEL_0;
            read_arg.sensor_count = 1;
            read_arg.cmd = TOUCH_DATA_BINARY;
            touch_err = R_TOUCH_Read(hdl_idx, &read_arg);
    }
}
```

Special Notes

None.

5.7 R_TOUCH_Calibrate

Function to use for performing calibration of a Touch Handle. Calibration involves performing offset adjustment to make the sensor counter equal to the reference counter for each sensor in the handle and updating the baselines to the current value of the sensor output.

Format

```
R_TOUCH_Calibrate (uint32_t hdl_num)
```

Parameters

- hdl_num - Handle assigned by a successful call to R_Touch_Open.

Return Values

- TOUCH_SUCCESS – Operation Successful
- TOUCH_ERR_AUTO_TUNE_FAILED – Too many errors occurred when scanning/tuning
- TOUCH_ERR_INVALID_PARAMETER – Invalid parameter found in control block

Properties

Prototyped in r_touch_rx_if.h.

Description

1. Check the handle identifier provided and select resource (exit if resource block is locked).
2. If no channels are ignored, use lower level hardware function to calibrate the CTSU.
3. Calibrate all sensors.

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t hdl_idx = UINT32_MAX;
    touch_err_t touch_err;
    touch_cfg_t *    p_touch_cfg;
    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;
    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
        touch_err  = R_TOUCH_Calibrate(hdl_idx);
        // More actions after calibration
    }
}
```

Special Notes

This function will call the scan function and will block (not return) until calibration succeeds (or fails).

5.8 R_TOUCH_Control

Change (write) parameters of associated with a control block.

Format

```
touch_err_t R_TOUCH_Control (uint32_t hdl_num, r_touch_control_arg_t *const
p_arg)
```

Parameters

- hdl_num - Handle being addressed.
- p_arg - Command and destination of information to read/write.

Return Values

- TOUCH_SUCCESS – Operation Successful.
- TOUCH_ERR_LOCKED – Control block is currently in use.
- TOUCH_ERR_INVALID_CMD – Control command not recognized.
- TOUCH_ERR_INVALID_PARAM – Invalid parameter found in argument.

Properties

Prototyped in r_touch_rx_if.h.

Description

The additional argument taken by this function has a structure as shown below:

```
typedef struct st_touch_control
{
    touch_data_t cmd;               // Control Command
    void * p_dest;                  // Location where results must be copied
    touch_sensor_t sensor;          // sensor channel to get/set information
} touch_control_arg_t;
```

The following is a list of actions to be performed in this function:

1. Check the handle identifier provided and select resource (exit if resource block is locked).
2. Write to the data location identified by the command and change the data to value pointed by p_dest.

The following Control Commands are supported by the driver:

Command	Data Type	Description
TOUCH_DATA_CTSU_HW_IDENTIFIER	UINT32	Writes CTSU Hardware Identifier returned by lower layer.
TOUCH_DATA_OPERATION_FLAGS	UINT32	Writes operation enable/disable flags
TOUCH_DATA_COMMON_DRIFT_HOLD_COUNTER	UINT16	Writes counter measuring drift hold time since last channel touched.
TOUCH_DATA_COMMON_DRIFT_HOLD_INTERVAL	UINT16	Writes interval for disabling drift compensation for all channels.
TOUCH_DATA_COMMON_CONTINUOUS_ON_COUNTER	UINT16	Writes counter measuring the amount of time a sensor is touched.
TOUCH_DATA_COMMON_CONTINUOUS_ON_INTERVAL	UINT16	Writes the interval after which all touched sensors are released.
TOUCH_DATA_FILTERED_COUNT	UINT16	Writes sensor count values output from filter for provided channels.
TOUCH_DATA_DELTA_COUNT	UINT16	Writes out the difference between sensor and baseline count to p_dest array for selected channels.
TOUCH_DATA_BASELINE_COUNT	UINT16	Writes the current sensor baseline used to determine if a channel is being touched for selected channels.
TOUCH_DATA_DELAY_TO_TOUCH_COUNTER	UINT16	Writes value of delay to touch counter
TOUCH_DATA_DELAY_TO_RELEASE_COUNTER	UINT16	Writes value of delay to release counter
TOUCH_DATA_AVERAGE_INPUT	UINT16	Writes average input value during drift interval
TOUCH_DATA_MINIMUM_INPUT	UINT16	Writes minimum value input during drift interval
TOUCH_DATA_MAXIMUM_INPUT	UINT16	Writes maximum value input during drift interval
TOUCH_DATA_DRIFT_COUNTER	UINT16	Writes value of counter counting up to drift interval
TOUCH_DATA_DRIFT_INTERVAL	UINT16	Writes value of drift interval
TOUCH_DATA_AVG_SEN_COUNTER_PRI	UINT16	Writes average value of primary sensor counter
TOUCH_DATA_AVG_REF_COUNTER_PRI	UINT16	Writes average value of primary reference counter
TOUCH_DATA_RECALIBRATION_COUNTER	UINT16	Writes value of re-calibration counter

Re-entrant

Function is reentrant for different identifiers passed.

Example

```
void my_func(void)
{
    uint32_t    hdl_idx = UINT32_MAX;
    touch_err_t touch_err;
    touch_cfg_t *    p_touch_cfg;
    touch_control_arg_t arg;
    uint16_t dt_value = EXAMPLE_DELAY_VALUE;

    extern touch_cfg_t g_touch_cfg_ctsensor_rx130_self;

    p_touch_cfg = &g_touch_cfg_ctsensor_rx130_self;
    if(TOUCH_SUCCESS == R_TOUCH_Open(&hdl_idx, p_touch_cfg))
    {
        arg.p_dest = &dt_value;
        arg.sensor = CHANNEL_0;
        arg.cmd = TOUCH_DATA_DELAY_TO_TOUCH_COUNTER;
        touch_err = R_TOUCH_Control(hdl_idx, &read_arg);
    }
}
```

Special Notes

This function does not check the locks in the control block. It is recommended to ensure the control block is not being used when this function call is invoked.

6. Appendices

6.1 Sample: Button/Slider/Wheel Board (20 Sensor Configuration)

An example configuration for 20 Sensors in total including a button, wheel and slider; below are the sensor drive settings.

```
static ctsu_sensor_setting_t sensor_drive_setting_self01[] = {
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x181)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1A1)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x156)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x185)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1A1)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1B4)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1B2)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x273)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x277)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x232)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x212)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x204)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1F9)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x208)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1A8)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x205)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x22B)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x230)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x287)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x282)),  .ctsuso1
= ((1) << 13) | ((0x03) << 8) | (0x5F), },
};
```

The structure that holds the SFRs.

```
static uint16_t
self_p_sensor_datas_self01[sizeof(sensor_drive_setting_self01)/sizeof(ctsu_sen
sor_setting_t)*2] = {0};
static ctsu_const_sfrs_t ctsu_const_sfrs_self01 = {
    .CTSUCR0.BYTE = (0x00),
    .CTSUCR1.BYTE = (((1)<<6)|((0)<<4)|(((0))<<3)|((0)<<2)|((1)<<1)|((1)<<0)),
    .CTSUSDPRS.BYTE = ((((0))<<6)|(((2))<<4)|((3)<<0)),
    .CTSUSST.BYTE = ((16)),
    .CTSUCHAC0.BYTE = ((((1) << 0) | ((1) << 1) | ((1) << 2) | ((1) << 3) |
((1) << 4) | ((1) << 5) | ((1) << 6) | ((1) << 7))),
    .CTSUCHAC1.BYTE = ((((0) << 0) | ((0) << 1) | ((0) << 2) | ((0) << 3) |
((0) << 4) | ((0) << 5) | ((0) << 6) | ((1) << 7))),
    .CTSUCHAC2.BYTE = ((((1) << 0) | ((1) << 1) | ((1) << 2) | ((1) << 3) |
((1) << 4) | ((1) << 5) | ((1) << 6) | ((0) << 7))),
    .CTSUCHAC3.BYTE = ((((0) << 0) | ((0) << 1) | ((0) << 2) | ((1) << 3) |
((0) << 4) | ((0) << 5) | ((1) << 6) | ((0) << 7))),
    .CTSUCHAC4.BYTE = ((((0) << 0) | ((1) << 1) | ((0) << 2) | ((1) << 3))),
    .CTSUDCLKC.BYTE = (((3)<<4)|((0)<<0)),
};
```

An example of the CTSU sensor configuration.

```
static ctsu_cfg_t g_ctsu_cfg_ctsensor_rx130_self = {
    .p_ctsu_settings = &ctsu_const_sfrs_self01,
    .p_sensor_settings = (ctsu_sensor_setting_t*)sensor_drive_setting_self01,
    .p_sensor_data = (uint16_t*)self_p_sensor_datas_self01,
    .pclkb_hz = 32000000,
    .p_callback = 0,
};
```

An example of parameters common across multiple touch sensors in one configuration.

```
static touch_common_parameter_t g_touch_common_parameter_rx130_self01 = {
    .drift_hold_limit = 1000,
    .on_limit = 2000,
    .max_touched_sensors = 1,
};
```

An example of touch related parameters that describe tunable parameters touch thresholds and hysteresis values specific to each touch sensor are contained below.

```
static touch_sensor_parameter_t const g_touch_sensor_parameter_rx130_self01[]
= {
    [0] =  {.threshold = (3172), .hysteresis = (158), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(3172)/3},
    [1] =  {.threshold = (2858), .hysteresis = (142), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(2858)/3},
    [2] =  {.threshold = (1002), .hysteresis = (52), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1002)/3},
    [3] =  {.threshold = (1003), .hysteresis = (53), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1003)/3},
    [4] =  {.threshold = (1004), .hysteresis = (54), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1004)/3},
    [5] =  {.threshold = (1005), .hysteresis = (55), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1005)/3},
    [6] =  {.threshold = (1006), .hysteresis = (56), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1006)/3},
    [7] =  {.threshold = (1007), .hysteresis = (57), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1007)/3},
    [8] =  {.threshold = (1015), .hysteresis = (65), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1015)/3},
    [9] =  {.threshold = (1016), .hysteresis = (66), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1016)/3},
    [10] = {.threshold = (1017), .hysteresis = (67), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1017)/3},
    [11] = {.threshold = (1018), .hysteresis = (68), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1018)/3},
    [12] = {.threshold = (1019), .hysteresis = (69), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1019)/3},
    [13] = {.threshold = (1020), .hysteresis = (70), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1020)/3},
```

```
    [14] = {.threshold = (1847), .hysteresis = (92), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1847)/3},
    [15] = {.threshold = (1022), .hysteresis = (72), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1022)/3},
    [16] = {.threshold = (1027), .hysteresis = (77), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1027)/3},
    [17] = {.threshold = (1030), .hysteresis = (80), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1030)/3},
    [18] = {.threshold = (1033), .hysteresis = (83), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1033)/3},
    [19] = {.threshold = (1035), .hysteresis = (85), .dt_limit =
100, .dr_limit = 100, .drift_rate = 100, .drift_rate_plus =
0, .drift_rate_minus = 0, .recalib_delay = 200, .recalib_threshold =
(1035)/3},
};
```

An example of a touch configuration is contained below.

```
static uint8_t g_touch_binary_rx130_self01[3];
static uint8_t buffer[640];
touch_cfg_t g_touch_cfg_ctsensor_rx130_self01 = {
    .p_ctsu_cfg = &g_ctsu_cfg_ctsensor_rx130_self,
    .p_common = &g_touch_common_parameter_rx130_self01,
    .p_sensor =
(touch_sensor_parameter_t*)&g_touch_sensor_parameter_rx130_self01[0],
    .p_binary_result = &g_touch_binary_rx130_self01[0],
    .p_callback = 0,
    .num_ignored = 0,
    .p_ignored = 0,
    .buffer = {
        .p_start = buffer,
        .size = sizeof(buffer),
    },
    .custom ={
        .p_filter = 0,
        .p_touch_detect = 0,
        .p_filter_instance = 0,
        .num_filter_instances = 0,
    },
};
```

6.2 Sample: Mutual Capacitance Button board (20 Sensor Configuration)

An example configuration for 20 Sensors in total in a mutual capacitance configuration; below are the sensor drive settings.

```
static ctsu_sensor_setting_t sensor_setting_mutual0[] = {
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1BE)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1CC)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1CF)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1CE)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1CE)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1E3)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1F5)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1FB)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x1F9)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x201)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x23E)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x256)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x255)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x260)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x268)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x01) << 8),  .ctsuso0 = (((3) << 10) | (0x0B8)),  .ctsuso1
= (((1) << 13) | ((7) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x21D)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x226)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x22A)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
    { .ctsussc = ((0x00) << 8),  .ctsuso0 = (((7) << 10) | (0x237)),  .ctsuso1
= (((1) << 13) | ((3) << 8) | (0x3F)), },
};
```

The structure that holds the SFRs.

```
static uint16_t
mutual01_p_sensor_datas[sizeof(sensor_setting_mutual0)/sizeof(ctsu_sensor_sett
ing_t)*4] = {0};
static ctsu_const_sfrs_t ctsu_const_sfrs_mutual01 = {
    .CTSUCR0.BYTE = (((0))<<7),
    .CTSUCR1.BYTE = (((3)<<6)|((0)<<4)|(((1))<<3)|((0)<<2)|((1)<<1)|((1)<<0)),
    .CTSUSDPRS.BYTE = ((((0))<<6)|(((2))<<4)|((3)<<0)),
    .CTSUSST.BYTE = ((16)),
    .CTSUCHAC0.BYTE = ((((0) << 0) | ((0) << 1) | ((0) << 2) | ((0) << 3) |
((0) << 4) | ((1) << 5) | ((1) << 6) | ((1) << 7))),
    .CTSUCHAC1.BYTE = ((((1) << 0) | ((1) << 1) | ((0) << 2) | ((0) << 3) |
((0) << 4) | ((0) << 5) | ((0) << 6) | ((0) << 7))),
    .CTSUCHAC2.BYTE = ((((0) << 0) | ((0) << 1) | ((0) << 2) | ((0) << 3) |
((0) << 4) | ((0) << 5) | ((0) << 6) | ((0) << 7))),
    .CTSUCHAC3.BYTE = ((((0) << 0) | ((0) << 1) | ((0) << 2) | ((1) << 3) |
((0) << 4) | ((0) << 5) | ((1) << 6) | ((0) << 7))),
    .CTSUCHAC4.BYTE = ((((0) << 0) | ((1) << 1) | ((0) << 2) | ((1) << 3))),
    .CTSUCHTRC0.BYTE = (((((0)) << 0) | (((0)) << 1) | (((0)) << 2) | (((0))
<< 3) | (((0)) << 4) | (((1)) << 5) | (((1)) << 6) | (((1)) << 7))),
    .CTSUCHTRC1.BYTE = (((((1)) << 0) | (((1)) << 1) | (((0)) << 2) | (((0))
<< 3) | (((0)) << 4) | (((0)) << 5) | (((0)) << 6) | (((0)) << 7))),
    .CTSUCHTRC2.BYTE = (((((0)) << 0) | (((0)) << 1) | (((0)) << 2) | (((0))
<< 3) | (((0)) << 4) | (((0)) << 5) | (((0)) << 6) | (((0)) << 7))),
    .CTSUCHTRC3.BYTE = (((((0)) << 0) | (((0)) << 1) | (((0)) << 2) | (((0))
<< 3) | (((0)) << 4) | (((0)) << 5) | (((0)) << 6) | (((0)) << 7))),
    .CTSUCHTRC4.BYTE = (((((0)) << 0) | (((0)) << 1) | (((0)) << 2) | (((0))
<< 3))),
    .CTSUDCLKC.BYTE = (((3)<<4)|((0)<<0)),
};
```

An example of the CTSU sensor configuration.

```
static ctsu_cfg_t g_ctsu_cfg_ctsensor_rx130_mutual01 = {
    .p_ctsu_settings = &ctsu_const_sfrs_mutual01,
    .p_sensor_settings = (ctsu_sensor_setting_t*)sensor_setting_mutual0,
    .p_sensor_data = (uint16_t*)mutual01_p_sensor_datas,
    .pclkb_hz = 32000000,
    .p_callback = 0,
};
```

An example of parameters common across multiple touch sensors in one configuration.

```
static touch_common_parameter_t g_touch_common_parameter_rx130_mutual01 = {
    .drift_hold_limit = 50,
    .on_limit = 500,
    .max_touched_sensors = 1,
};
```

An example of touch related parameters that describe tunable parameters touch thresholds and hysteresis values specific to each touch sensor are contained below.

```
touch_sensor_parameter_t const g_touch_sensor_parameter_rx130_mutual01[] = {
    [0] =  {.threshold = (1442), .hysteresis = (72), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1442)/3},
    [1] =  {.threshold = (1271), .hysteresis = (63), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1271)/3},
    [2] =  {.threshold = (1603), .hysteresis = (80), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1603)/3},
    [3] =  {.threshold = (1738), .hysteresis = (86), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1738)/3},
    [4] =  {.threshold = (1512), .hysteresis = (75), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1512)/3},
    [5] =  {.threshold = (1869), .hysteresis = (93), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1869)/3},
    [6] =  {.threshold = (1736), .hysteresis = (86), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1736)/3},
    [7] =  {.threshold = (1301), .hysteresis = (65), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1301)/3},
    [8] =  {.threshold = (1511), .hysteresis = (75), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1511)/3},
    [9] =  {.threshold = (1831), .hysteresis = (91), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1831)/3},
    [10] = {.threshold = (2202), .hysteresis = (110), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (2202)/3},
    [11] = {.threshold = (1601), .hysteresis = (80), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1601)/3},
    [12] = {.threshold = (1756), .hysteresis = (87), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1756)/3},
    [13] = {.threshold = (1109), .hysteresis = (55), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1109)/3},
    [14] = {.threshold = (1263), .hysteresis = (63), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1263)/3},
    [15] = {.threshold = (762), .hysteresis = (38), .dt_limit = 1, .dr_limit =
1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (762)/3},
    [16] = {.threshold = (1211), .hysteresis = (60), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1211)/3},
    [17] = {.threshold = (1195), .hysteresis = (59), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1195)/3},
    [18] = {.threshold = (1168), .hysteresis = (58), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1168)/3},
```

```
    [19] = {.threshold = (1260), .hysteresis = (63), .dt_limit = 1, .dr_limit
= 1, .drift_rate = 100, .drift_rate_plus = 0, .drift_rate_minus =
0, .recalib_delay = 200, .recalib_threshold = (1260)/3},
};
```

An example of a touch configuration is contained below.

```
static uint8_t g_touch_binary_rx130_mutual01[3];
static uint8_t buffer_mutual01[640];
touch_cfg_t g_touch_cfg_ctsensor_rx130_mutual01 = {
    .p_ctsu_cfg = &g_ctsu_cfg_ctsensor_rx130_mutual01,
    .p_common = &g_touch_common_parameter_rx130_mutual01,
    .p_sensor =
(touch_sensor_parameter_t*)&g_touch_sensor_parameter_rx130_mutual01[0],
    .p_binary_result = &g_touch_binary_rx130_mutual01[0],
    .p_callback = 0,
    .num_ignored = 0,
    .p_ignored = 0,
    .buffer = {
        .p_start = buffer_mutual01,
        .size = sizeof(buffer_mutual01),
    },
    .custom = {
        .p_filter = 0,
        .p_touch_detect = 0,
        .p_filter_instance = 0,
        .num_filter_instances = 0,
    },
};
```

Website and Support

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

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	May 9, 2017	-	Initial Version Released

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. 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 Microprocessing unit or Microcontroller unit products 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.

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 disputes 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, drawing, chart, program, algorithm, 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 not alter, modify, copy, or otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
5. 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; and 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.

   Renesas Electronics products are neither intended nor 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 damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. 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. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use 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.
9. 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 not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

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: +44-1628-585-100, Fax: +44-1628-585-900

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, P. R. China 200333
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 II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0