RX Family

Low Power Capacitive Touch Demo for RSKRX130

R01AN5291EU0100
Rev. 1.00
Nov 11, 2019

Introduction

The Low Power Capacitive Touch Demo for RSKRX130 demonstrates capacitive touch operation in low and full power modes. It is intended as a model for battery powered systems using the RX130 MCU.

Target Device

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

- RX130 Group
- Renesas Starter Kit for RX130-128KB (YRTK5005130S00000BE)
- Renesas Starter Kit for RX130-512KB (YRTK5051308S00000BE)

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)
- QE CTSU Module Using Firmware Integration Technology (R01AN4469EU)
- QE Touch Module Using Firmware Integration Technology (R01AN4470EU)
- RX130 Group Renesas Start Kit User's Manual (R20UT3444EG)
- RX130 Group Renesas Start Kit User's Manual (R20UT3921EG)

Contents

1. Overview 3
1.1 RSKRX130 Sensor Usage 3
1.2 Software Requirements 3

2. Method State Flow Diagrams 4
2.1 Main Control Loop 4
2.2 process_full_pwr() 5
2.3 process_low_pwr() 6

3. Special Constants 8
3.1 Low Power Scan Interval and Button Response 8
3.2 Full Power Scan Interval 8

4. Application Code 9

Website and Support 18

Revision History 19

1. Overview

A typical low power application consists of two configuration methods: one for low power and one for full power. The low power configuration typically consists of a single button which serves as the “on-off” button. The full power configuration typically consists of all remaining sensors in the system as well as the “on-off” button. These two methods are defined and configured in this demo using the QE for Capacitive Touch tool.

The application itself consists of two different logic branches which handle processing for each of these methods. In the main processing loop a single function is called depending upon whether the system is operating in low or full power mode. No RTOS or scheduler is required.

The *application state low power* is quite simple as all it has to do is periodically check for the “on” condition and then go back to sleep if it did not occur. If it did occur, it changes to *application state full power wakeup* so the full power logic can run. The *application state full power wakeup* lasts only for one scan cycle and is where the wakeup operations such as activating an LCD or enabling another peripheral could be performed. When this completes, the system is changed to the normal processing *application state full power*. Upon detection of the “off” condition, “shutdown” operations are performed and the system changes back to *application state low power*.

1.1 RSKRX130 Sensor Usage

In this demo, button/key 1 serves as the “on-off” button. If running in *application state low power*, touching key1 will cause the application to switch to full power. If running in *application state full power*, touching key1 will cause the application to switch to low power.

When in *application state low power*, LED2 is lit, and key2 and the slider are inactive. These sensors are only active when in *application state full power* (LED1 lit). It is recommended to add the global variables “g_btn_states” and “g_sldr_pos” to the Expressions window with Real-time Refresh enabled. This will reflect touches to key2 and the slider when in *application state full power*.

1.2 Software Requirements

The following drivers are required for this application:

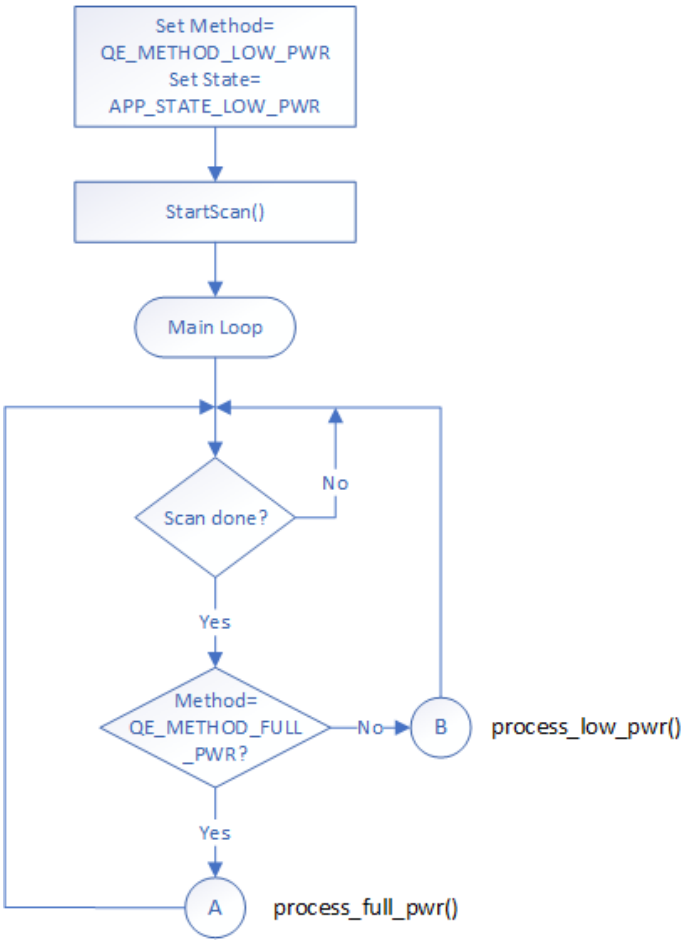
- QE CTSU FIT Module v1.11 (or later)
- QE Touch FIT Module v1.11 (or later)

2. Method State Flow Diagrams

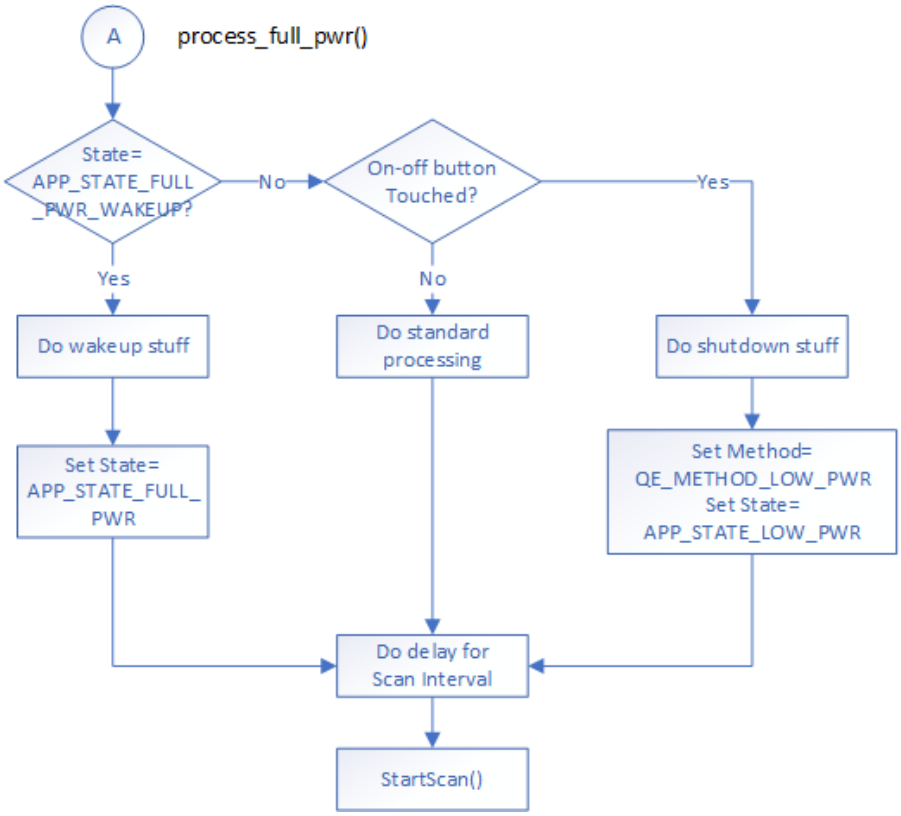
The code snippets found in this section come from the complete application code provided in Section 4.

2.1 Main Control Loop

The main loop waits for a scan to complete, then calls a function based upon whether the low or high power method is currently running. Note that even though this looks like the main loop is just spinning by polling the scan-done flag, the processor is usually sleeping in the low power function (connector “B”) or delaying at the end of the full power function. All full power processing is done within the full power function (connector “A”).



2.2 process_full_pwr()



When the function process_full_pwr() is called from the main loop, the MCU is operating at full power even if the previous scan occurred during a low power state because the core would have wakened from a scan-done interrupt. This function basically operates as any non-low power application would. The main difference is that there is a two-state state machine to check. This serves the purpose of allowing the application to be aware of the first scan occurring after leaving the *application state low power* so any special one-time operations that may need to be performed can be. In this demo, all that is done is to change the LED states so that there is a visual indicator of what mode the system is running in. Other applications may require the enabling of another peripheral or activating an LCD.

```
        /* ADD CODE HERE FOR FULL POWER WAKE-UP... */

        FULL_PWR_LED_ON;
        LOW_PWR_LED_OFF;

        /* ...END FULL POWER WAKE-UP USER CODE */
```

If it is not the first scan after entering the full power application state, normal processing of all sensors occurs. If the on-off button (key1) is not touched, the remaining sensors (key2 and slider) are checked for touches. After the sensors are checked, any other system processing is performed.

```
            /* ADD CODE HERE FOR NORMAL FULL POWER PROCESSING... */

            /* see if key2 touched */
            if ((touch_changed(&last_state_key2,
                               (g_btn_states & FULL_PWR_MASK_KEY2),
                               &change) == true)
             && (change == CHANGE_TOUCH_DETECTED))
            {
                nop();  // do key2 processing
            }
```

```
            /* see if slider touched */
            R_TOUCH_GetSliderPosition(FULL_PWR_ID_SLIDER00, &g_sldr_pos);
            if (g_sldr_pos != 0xFFFF)
            {
                nop();  // do slider processing
            }

            /* do other non-touch full power processing here */

            /* ...END NORMAL FULL POWER PROCESSING USER CODE */
```

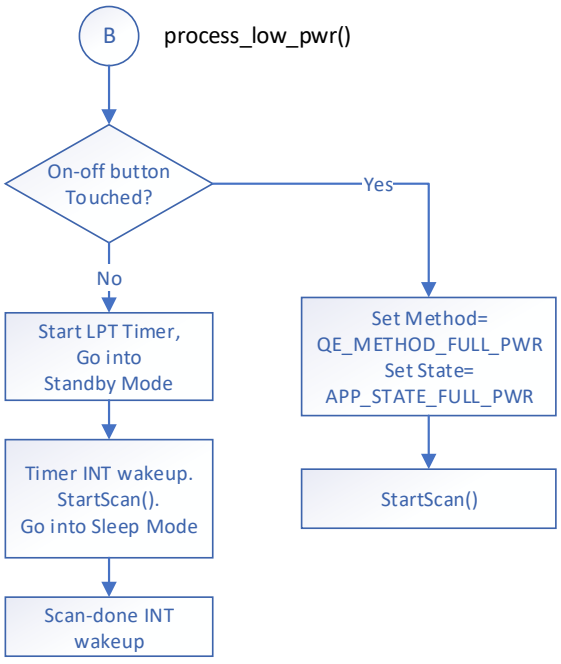
If the on-off button is touched, shutdown operations are performed, then the method and state information is set to go into low power. In this demo, all that is done is to reverse the LED values to show the new system mode. Other applications may involve shutting down of another peripheral or LCD.

```
            /* ADD CODE HERE FOR EXITING FULL POWER... */

            FULL_PWR_LED_OFF;
            LOW_PWR_LED_ON;

            /* ...END EXITING FULL POWER USER CODE */
```

2.3 process_low_pwr()



When the system is running in *application state low power*, the MCU goes in and out of low power modes as it periodically needs to start scans and check for button touches. When this function is called, the MCU is temporarily awake due to a scan-done interrupt. There is no user-specific code present in this function.

The MCU is capable of running in one of three low power modes: Sleep Mode, Deep Sleep Mode, and Software Standby Mode (see Table 11.2 in the RX130 Hardware Manual). Software Standby Mode consumes the least amount of power so it is desirable to spend as much time in this mode as possible.

When the on-off button is not touched, this function

- starts the low power timer (LPT- source is IWDT due to slightly better power savings than using Subclock)

- disables the DTC and puts the CTSU into Snooze mode for extra power savings
- puts the MCU into Standby Mode
- “waits” for the low power timer to expire and its interrupt to wake up the MCU (technically, the processor halts just before the “return” statement in the function called by R_LPC_LowPowerModeActivate(), then resumes processing at that “return” statement after the interrupt exits).

```
/* start one-shot low power timer for next scan start */
R_LPT_Control(LPT_CMD_START);

/* disable DTC to save power */
DTC.DTCST.BIT.DTCST = 0;

/* have CTSU snooze to save power */
R_CTSU_Control(CTSU_CMD_SNOOZE_ENABLE, 0 , NULL);

/* configure for standby mode */
R_LPC_LowPowerModeConfigure(LPC_LP_SW_STANDBY);

/* put main processor into standby mode */
R_LPC_LowPowerModeActivate(FIT_NO_FUNC);

/* ...main processor and CTSU "sleeping" here... */

/* LPT timer expired; processor awake; resume processing */
```

While the MCU is temporarily awake again, this function

- re-enables the DTC (The DTC handles processing of the CTSU register write and read interrupts which occur just before and after scanning each sensor. The core handles the scan-done interrupt which occurs only after all of the sensors have been scanned.)
- takes the CTSU out of snooze mode
- starts a scan
- puts the MCU into Sleep Mode
- “waits” for the scan-done interrupt to occur to wake up the MCU (technically, the processor halts just before the “return” statement in the function called by R_LPC_LowPowerModeActivate(), then resumes processing at the “return” statement after the interrupt exits).

```
/* enable DTC */
DTC.DTCST.BIT.DTCST = 1;

/* disable CTSU snooze */
R_CTSU_Control(CTSU_CMD_SNOOZE_DISABLE, 0 , NULL);

/* start scan */
R_CTSU_StartScan();

/* configure for sleep mode */
R_LPC_LowPowerModeConfigure(LPC_LP_SLEEP);

/* put main processor into sleep mode */
R_LPC_LowPowerModeActivate(FIT_NO_FUNC);

/* ...main processor sleeping here until interrupt occurs... */

/* scan done interrupt occurred; processor awake; resume processing */
```

3. Special Constants

3.1 Low Power Scan Interval and Button Response

This application sleeps approximately 100ms between scans while in *application state low power*. This is set using the #define LOW_PWR_SCAN_INTERVAL_US at the top of the main() application file and can be changed by the user if desired. This value is passed to the Low Power Timer routine R_LPT_Open().An additional 500us scan time plus minor processing time should be added to this value to calculate the actual total scan interval time..

The Touch driver by default does not report a touch condition immediately. It waits for three scans (QE generated default) after the moving average is calculated above the touch condition (threshold) before reporting it. This is meant to act like a noise filter. At faster scan speeds, this is not an issue. However, with a 100ms scan interval, the delay in reporting is a minimum 400ms (likely a half second or more) depending upon when the button is touched during the sleep interval and how large a threshold offset there is. This is too long to wait for most users.

To address this, the constants **LOW_PWR_TOUCH_ON** and **LOW_PWR_TOUCH_OFF** are **changed to 1** (default 3) in the file “qe_low_pwr.h”. This causes the touch condition to be reported as soon as the moving average crosses the touch threshold (200ms+).

The baseline value which the threshold offset is added to for detecting touch conditions should be updated every few seconds. This is to account for changes in environmental conditions like temperature and humidity. The baseline value is updated every **LOW_PWR_DRIFT_FREQ** number of scans. To account for the longer low power scan interval, this value is **changed to 50** (default 255) in the file “qe_low_pwr.h”.

Note that if the code is ever regenerated by the QE Tool (as would likely be done if retuning the system), these values will be overwritten and need to be changed manually again.

3.2 Full Power Scan Interval

This application pauses approximately 20ms between scans in process_full_pwr() at the following line:

```
R_BSP_SoftwareDelay(FULL_PWR_SCAN_INTERVAL_MS, BSP_DELAY_MILLISECS);
```

The #define FULL_PWR_SCAN_INTERVAL_MS is located at the top of the main() file and its value may be modified by the user. This is a software delay loop that occurs after the full power processing completes. Note that the application stays here until the delay loop completes. If additional processing is needed during this delay period, the delay may be broken into a sequence of smaller software delays with interspersed processing, or replaced with a hardware timer polling loop which contains the necessary system processing. An additional 3ms scan time (500us per sensor) plus app processing time should be added to this value to calculate the actual total scan interval time.

4. Application Code

```
/***********************************************************************
* File Name : touch_demo_rskrx130_low_pwr.c
* Description  : This program serves as a template for a typical low power
*                application using the RSKRX130.
*                v1.11 or later of the CTSU and Touch drivers is required for
*                proper operation.
*
*                HAND MODIFICATIONS MADE TO QE GENERATED FILE "qe_low_pwr.h":
*                   #define LOW_PWR_TOUCH_ON       1
*                   #define LOW_PWR_TOUCH_OFF      1
*                   #define LOW_PWR_DRIFT_FREQ     50
*
*                This program uses button "1" as the on/off button. After
*                initialization, the program goes into low power mode (LED2 on).
*                When button 1 is touched, the program goes into full power mode
*                (LED1 on) and button 2 and the slider become active. The values
*                for these sensors can be monitored in the Expressions window by
*                putting Real-time Refresh on the variables "g_btn_states" and
*                "g_sldr_pos". Touching button 1 again returns the program to
*                low power mode. The variables are not updated while in low
*                power.
***********************************************************************/
/***********************************************************************
* History : DD.MM.YYYY Version Description
*           09.10.2019 1.00    Initial version.
***********************************************************************/

#include "r_smc_entry.h"
#include "qe_common.h"
#include "qe_low_pwr.h"
#include "qe_full_pwr.h"
#include "r_touch_qe_if.h"
#include "r_ctsu_qe_if.h"
#include "r_elc_rx_if.h"
#include "r_lpt_rx_if.h"
#include "r_lpc_rx_if.h"
#include "r_ctsu_qe_pinset.h"


/***********************************************************************
* Macro definitions
***********************************************************************/
#define FULL_PWR_LED_ON     (PORT0.PODR.BIT.B4 = 0) // LED1
#define LOW_PWR_LED_ON      (PORT0.PODR.BIT.B6 = 0) // LED2
#define FULL_PWR_LED_OFF    (PORT0.PODR.BIT.B4 = 1)
#define LOW_PWR_LED_OFF     (PORT0.PODR.BIT.B6 = 1)

#define LOW_PWR_SCAN_INTERVAL_US    (100000)        // 100ms
/* full pwr scan interval = delay time + 3ms scan time + app processing time */
#define FULL_PWR_SCAN_INTERVAL_MS   (17)            // 17ms delay time


/***********************************************************************
* Typedef definitions
***********************************************************************/
typedef enum e_change
{
    CHANGE_NONE,
    CHANGE_TOUCH_DETECTED,
```

```
    CHANGE_UNTOUCH_DETECTED
} change_t;


typedef enum e_sys_state
{
    APP_STATE_FULL_PWR_WAKEUP,  // first scan after on-btn touched state
    APP_STATE_FULL_PWR,         // normal full power processing state
    APP_STATE_LOW_PWR           // low power processing state
} APP_STATE_t;


/***********************************************************************
* Private global variables and functions
***********************************************************************/
uint64_t        g_btn_states;
uint16_t        g_sldr_pos;
uint8_t         g_method;
volatile bool   g_scan_done_flg=false;
APP_STATE_t     g_app_state;

void main(void);
void process_full_pwr(void);
bool touch_changed(uint64_t *p_last_state,
                   uint64_t  current_state,
                   change_t *p_change);
void process_low_pwr(void);
void elc_lpt_trigger_callback(void * pdata);
void scan_done_callback(ctsu_isr_evt_t event, void *p_args);
void init_elc_and_lpt(void);


void main(void)
{
    qe_err_t    qe_err;
    lpc_err_t   lpc_err;


    /* Setup pins */
    R_CTSU_PinSetInit();
    FULL_PWR_LED_OFF;
    LOW_PWR_LED_OFF;


    /* Set operating mode (implies max clock frequencies allowed) */
    lpc_err = R_LPC_OperatingModeSet(LPC_OP_HIGH_SPEED);
    if (LPC_SUCCESS != lpc_err)
    {
        while(1)
        {
            nop();
        }
    }

    /* Open Touch driver (opens CTSU driver as well).
     * Setup special callback function.
     */
    qe_err = R_TOUCH_Open(gp_ctsu_cfgs, gp_touch_cfgs,
                          QE_NUM_METHODS, QE_TRIG_SOFTWARE);
    if (QE_SUCCESS != qe_err)
    {
```

```
        while(1)
        {
            nop();
        }
    }

    qe_err = R_CTSU_Control(CTSU_CMD_SET_CALLBACK, 0, scan_done_callback);
    if (QE_SUCCESS != qe_err)
    {
        while(1)
        {
            nop();
        }
    }


    /* Setup Event Link Controller (ELC) and Low Power Timer (LPT)
     * for low power scan interval
     */
    init_elc_and_lpt();


    /* Begin processing with low power method (do not cycle through methods) */
    g_method = QE_METHOD_LOW_PWR;
    R_TOUCH_Control(TOUCH_CMD_SET_METHOD, &g_method);
    g_app_state = APP_STATE_LOW_PWR;
    LOW_PWR_LED_ON;
    R_CTSU_StartScan(); // MCU still in full power until initial scan processed


    while(1)
    {
        if (true == g_scan_done_flg)
        {
            /* Scan complete. Reset flag and update touch data */
            g_scan_done_flg = false;
            qe_err = R_TOUCH_UpdateData();

            /* process touch data */
            if (g_method == QE_METHOD_FULL_PWR)
            {
                process_full_pwr();
            }
            else // QE_METHOD_LOW_PWR
            {
                process_low_pwr();
            }
        }

    } // end while
}


/***********************************************************************************
* Function Name: process_full_pwr
* Description  : This is the high-level function which handles full power
*                processing. There are three parts to this:
*                1) Handle "wake-up" condition
*                2) Do normal processing
*                3) Detect off-button touch and handle "shut down" condition
*                The pause time between scans occurs at the end of this
```

```
*                function. Note that the full time between scans includes
*                this delay time plus the variable time it takes to process
*                the preceding code in this function.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void process_full_pwr(void)
{
    change_t        change;
    static uint64_t last_state_key2=CHANGE_NONE;
    static uint64_t last_state_on_off=CHANGE_NONE;


    /* get touch data */
    R_TOUCH_GetAllBtnStates(QE_METHOD_FULL_PWR, &g_btn_states);

    /* see if on/off button changed state */
    touch_changed(&last_state_on_off,
                  (g_btn_states & FULL_PWR_MASK_ON_OFF), &change);

    /* process based upon system state */
    if ((g_app_state == APP_STATE_FULL_PWR_WAKEUP)
     && (change == CHANGE_TOUCH_DETECTED))             // detected on-btn touch
    {
        /* ADD CODE HERE FOR FULL POWER WAKE-UP... */
        FULL_PWR_LED_ON;
        LOW_PWR_LED_OFF;

        /* ...END FULL POWER WAKE-UP USER CODE */
        g_app_state = APP_STATE_FULL_PWR;
    }
    else if (g_app_state == APP_STATE_FULL_PWR)
    {
        if (change == CHANGE_TOUCH_DETECTED)           // detected off-btn touch
        {
            /* ADD CODE HERE FOR EXITING FULL POWER... */
            FULL_PWR_LED_OFF;
            LOW_PWR_LED_ON;

            /* ...END EXITING FULL POWER USER CODE */

            g_method = QE_METHOD_LOW_PWR;
            R_TOUCH_Control(TOUCH_CMD_SET_METHOD, &g_method);
            g_app_state = APP_STATE_LOW_PWR;
        }
        else
        {
            /* ADD CODE HERE FOR NORMAL FULL POWER PROCESSING... */

            /* see if key2 touched */
            if ((touch_changed(&last_state_key2,
                               (g_btn_states & FULL_PWR_MASK_KEY2),
                               &change) == true)
             && (change == CHANGE_TOUCH_DETECTED))
            {
                nop();  // do key2 processing
            }

            /* see if slider touched */
            R_TOUCH_GetSliderPosition(FULL_PWR_ID_SLIDER00, &g_sldr_pos);
            if (g_sldr_pos != 0xFFFF)
```

```
            {
                nop();  // do slider processing
            }

            /* do other non-touch full power processing here */

            /* ...END NORMAL FULL POWER PROCESSING USER CODE */
        }
    }

    /* start next scan after interval expires */
    R_BSP_SoftwareDelay(FULL_PWR_SCAN_INTERVAL_MS, BSP_DELAY_MILLISECS);
    R_CTSU_StartScan();
}


/***********************************************************************
* Function Name: touch_changed
* Description  : This function determines if a single button has changed its
*                state or not. This function works with masked values from
*                R_TOUCH_GetAllBtnStates().
* Arguments    : p_last_state
*                   Pointer to variable containing the last state of a single
*                   button. This variable is updated in this function.
*                current_state
*                   Current state of the button
*                p_change
*                   Pointer to variable which will be set to CHANGE_NONE,
*                   CHANGE_TOUCH_DETECTED, or CHANGE_UNTOUCH_DETECTED.
* Return Value : false - no change in state detected
*                true - a change was detected
***********************************************************************/
bool touch_changed(uint64_t *p_last_state,
                   uint64_t  current_state,
                   change_t *p_change)
{
    bool    changed=true;


    /* NOTE: FUNCTION CHECKS SINGLE BUTTON STATE ONLY!!! */

    if (current_state > *p_last_state)
    {
        *p_change = CHANGE_TOUCH_DETECTED;
    }
    else if (current_state < *p_last_state)
    {
        *p_change = CHANGE_UNTOUCH_DETECTED;
    }
    else // current state == last state
    {
        *p_change = CHANGE_NONE;
        changed = false;
    }

    *p_last_state = current_state;

    return changed;
}
```

```
/***********************************************************************************
* Function Name: process_low_pwr
* Description  : This is the high-level function which handles low power
*                processing. Note that while doing this, the MCU will go in and
*                out of different power levels. There are two main purposes of
*                this function:
*                1) Detect an on-button touch (go to full power mode)
*                2) Go back to MCU standby/sleep mode otherwise.
*                The processor wakes up whenever an interrupt occurs. This
*                includes the low power timer interrupt and the scan-done
*                interrupt. When this function is called, a scan-done interrupt
*                has already occurred and the processor is awake.
*
*                In the beginning of this function, if it is determined that the
*                on-button is not touched, the function puts the MCU into
*                standby mode until the low power timer expires. When that timer
*                expires, a new scan is started and this function then puts the
*                MCU into sleep mode where it remains until a scan completes.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void process_low_pwr(void)
{
    change_t        change;
    static uint64_t last_state_on_off=CHANGE_NONE;


    /* CALLING FUNCTION DETERMINED SCAN FINISHED. CORE PROCESSOR NOW AWAKE. */

    /* get touch data */
    R_TOUCH_GetAllBtnStates(QE_METHOD_LOW_PWR, &g_btn_states);

    /* see if on/off button changed state */
    touch_changed(&last_state_on_off,
                  (g_btn_states & LOW_PWR_MASK_ON_OFF), &change);

    /* see if button touched */
    if ((g_app_state == APP_STATE_LOW_PWR)
     && (change == CHANGE_TOUCH_DETECTED))
    {
        /* go to full power */
        g_method = QE_METHOD_FULL_PWR;
        R_TOUCH_Control(TOUCH_CMD_SET_METHOD, &g_method);
        R_TOUCH_Control(TOUCH_CMD_CLEAR_TOUCH_STATES, &g_method);
        g_app_state = APP_STATE_FULL_PWR_WAKEUP;
        R_CTSU_StartScan();
    }


    /* GO BACK TO STANDBY, THEN START ANOTHER SCAN */

    if (g_app_state != APP_STATE_FULL_PWR_WAKEUP)
    {
        /* start one-shot low power timer for next scan start */
        R_LPT_Control(LPT_CMD_START);


        /* Go into standby mode. Will awake when timer expires.
         * Most power saving done here.
         */
```

```
        /* disable DTC to save power */
        DTC.DTCST.BIT.DTCST = 0;

        /* have CTSU snooze to save power; RX100 SERIES ONLY! */
        R_CTSU_Control(CTSU_CMD_SNOOZE_ENABLE, 0 , NULL);

        /* configure for standby mode */
        R_LPC_LowPowerModeConfigure(LPC_LP_SW_STANDBY);

        /* put main processor into standby mode */
        R_LPC_LowPowerModeActivate(FIT_NO_FUNC);

        /* ...main processor and CTSU "sleeping" here... */

        /* timer expired; processor awake; resume processing */

        /* stay awake just long enough to start scan,
         * then go into sleep mode while scan is running
         */

        /* enable DTC */
        DTC.DTCST.BIT.DTCST = 1;

        /* disable CTSU snooze */
        R_CTSU_Control(CTSU_CMD_SNOOZE_DISABLE, 0 , NULL);

        /* start scan */
        R_CTSU_StartScan();

        /* configure for sleep mode */
        R_LPC_LowPowerModeConfigure(LPC_LP_SLEEP);

        /* put main processor into sleep mode */
        R_LPC_LowPowerModeActivate(FIT_NO_FUNC);

        /* ...main processor sleeping here until interrupt occurs... */

        /* scan done interrupt occurred; processor awake; resume processing */
    }

    return;
}


/***********************************************************************
* Function Name: lpt_trigger_callback
* Description  : Dummy function in this application. Just need the interrupt
*                to occur so processor wakes up.
* Arguments    : None
* Return Value : None
***********************************************************************/
void elc_lpt_trigger_callback(void * pdata)
{
    nop();
}


/***********************************************************************
* Function Name: scan_done_callback
* Description  : This function executes at interrupt level when a scan
*                completes. It sets a flag for the main application loop so
```

```
*                it knows to get the latest sensor data and process.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void scan_done_callback(ctsu_isr_evt_t event, void *p_args)
{
    ctsu_isr_evt_t evt;

    if (CTSU_EVT_SCAN_COMPLETE != evt)
    {
        nop();  // handle scan error
    }

    g_scan_done_flg = true;
}


/***********************************************************************************
* Function Name: init_elc_and_lpt
* Description  : This function initializes the Event Link Controller (ELC) and
*                Low Power Timer (LPT) drivers. The LPT is the only timer
*                (other than IWDT) which can run while the processor is in
*                standby mode. It outputs a signal to the ELC when it expires.
*                The ELC causes an interrupt to occur which causes the processor
*                to wake up from standby mode.
*
*                NOTES:
*                - Lower power consumption occurs when the IWDT is used as the
*                  LPT source instead of the Sbclock.
*                - Any LPT and ELC error conditions which are detected within
*                  this function should only occur during the development phase.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void init_elc_and_lpt(void)
{
    lpt_err_t   lpt_err;
    elc_err_t   elc_err;
    elc_event_signal_t  ev_signal;
    elc_link_module_t   ev_module;


    lpt_err = R_LPT_Open(LOW_PWR_SCAN_INTERVAL_US);
    if (LPT_SUCCESS != lpt_err)
    {
        while(1)
        {
            nop();
        };
    }

    elc_err = R_ELC_Open();
    if (ELC_SUCCESS != elc_err)
    {
        while(1)
        {
            nop();
        }
    }
```

```
    /* Setup ELC for LPT timer to trigger ISR */

    ev_signal.event_signal = ELC_LPT_CMP0;
    ev_module.link_module = ELC_ICU_LPT;
    ev_module.link_module_callbackfunc = &elc_lpt_trigger_callback;
    ev_module.link_module_interrupt_level = 15;
    elc_err = R_ELC_Set(&ev_signal, &ev_module);
    if (ELC_SUCCESS != elc_err)
    {
        while(1)
        {
            nop();
        }
    }

    elc_err = R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);
    if (ELC_SUCCESS != elc_err)
    {
        while(1)
        {
            nop();
        }
    }

    return;
}
```

Website and Support

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

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov.11.19	—	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD) A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (Max.) and VIH (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (Max.) and VIH (Min.).
7. Prohibition of access to reserved addresses Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a systemevaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
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; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc. Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
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. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
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 comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
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 directly or indirectly controlled subsidiaries. (Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

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

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

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

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

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 7.2