NuMicro™ NANO100 Series Touch Key Library Reference Guide

V1.00.001

Publication Release Date: Aug.30.2012

Support Chips	Support Platform
NuMicro™ NANO100 series	Nuvoton

The information in this document is subject to change without notice.

Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Overview

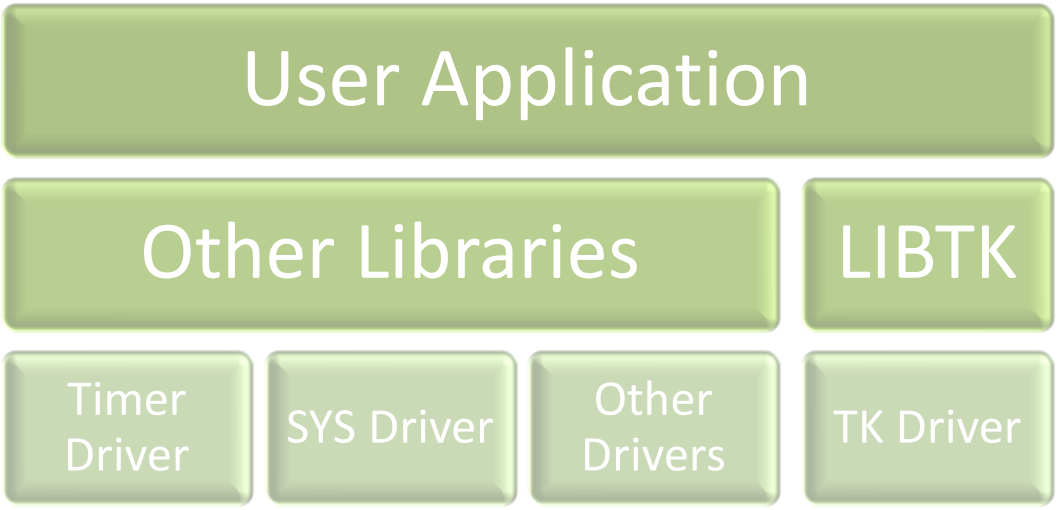
1.1. Features

The touch key library (LIBTK) provides APIs for the application to check touch key status. This library hides the run time calibration and status interpretation from application, and thus reduces the application complexity. Features of this library are listed below:

- Supports up to 16 touch key hardware channels in Nano100
- Supports three different types of components - ***key***, ***slider***, and ***rotor***; ***slider*** and ***rotor*** can be composed of 4~16 touch key channels; ***key*** is composed of 1 channel.
- Supports up to 16 components (all of which are ***key***)
- Supports configurable resolution for ***slider*** and ***rotor***.

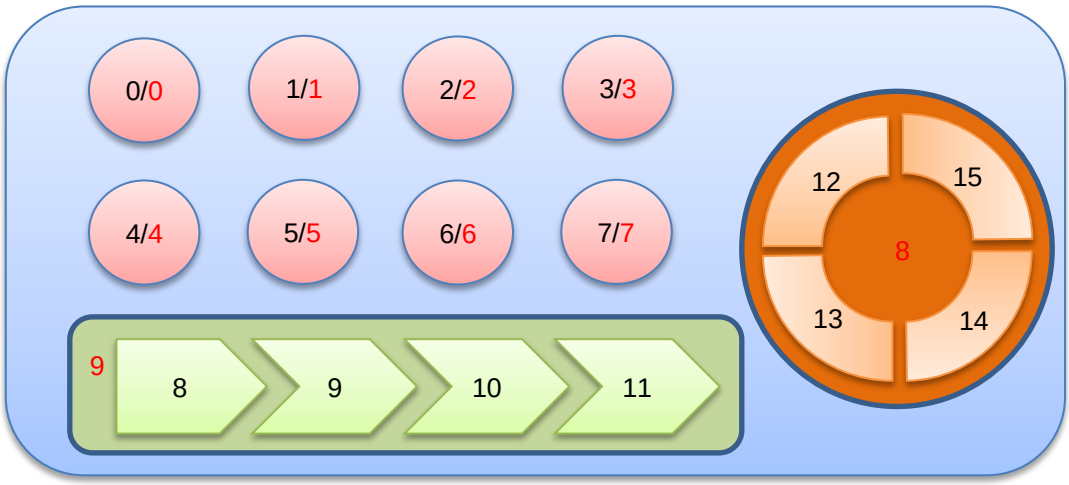
1.2. Software Architecture

The touch key library resides between user application and low level drivers and provides APIs for the application to check touch key component status in the system. The following figure illustrates the software architecture of an application using the touch key library.



The touch key library is dependent on touch key (TK) driver (i.e. software project wishes to include the touch key library must include TK drivers in the project). Otherwise compilation will fail due to undefined reference error. If timer trigger TK is used, user application can either control timer 0 through Timer driver, or control its register directly.

The touch key library hides the complexity of touch key operation and calibration from user application. Instead of handling the raw data of each ***channel***, the application register callback functions handle state changes of ***components***. Three types of components are supported by this library. They are ***key***, ***slider***, and ***rotor***. ***Slider*** and ***rotor*** can be composed of 4~16 channels; ***key*** can only be composed of one channel. Each component will be assigned a unique ID by the library. The following figure illustrates a board containing 8 keys, 1 slider and 1 rotor. Four channels form the ***slider*** and four channels form the ***rotor***. Channel numbers are displayed in black, and component IDs are displayed in red.



2. Configuration Parameters

User application must include a set of configuration parameters for all touch key channels. Even if some channels are not used, a dummy value must be given. Below is an example of configuration array.

```
libtk_channel_config_s cfg[16] = {
      {0xab00, 0x0143, 3, 2},
      {0xb800, 0x00F6, 2, 4},
      {0xb770, 0x0100, 2, 3},
      {0xac60, 0x010E, 3, 2},
      {0xa970, 0x00DB, 4, 2},
      {0xa6c0, 0x00E4, 4, 2},
      {0x9c20, 0x0115, 4, 2},
      {0x9740, 0x0133, 4, 2},
      {0xa5a2, 0x0087, 4, 2},
      {0xb342, 0x00a6, 4, 3},
      {0x99f2, 0x00d2, 4, 2},
      {0x9927, 0x00ac, 4, 2},
      {0xa3d5, 0x00de, 4, 3},
      {0x95b2, 0x00cc, 4, 2},
      {0x9923, 0x00c6, 6, 2},
      {0x999a, 0x00dd, 6, 2},
};
```

Nuvoton provides a tool executed on a target board to gather necessary information on the target board and prints such array through UART. After the auto configure tool is downloaded and executed, user will be asked to input channels to be scanned.

```
Please select test channel mask 0001~ffff:
```

Each bit represents one touch key channel. The most significant bit represents channel 15, and the least significant bit represents channel 0. 1 means channel enabled and 0 means channel disabled. So if a system uses channel 0~6, the input should be 007F. After user inputs a valid mask (at least one channel selected), this tool starts to scan the enabled channels from the lowest channel number to the highest channel number.

```
########## Detect channel 1 configuration ##########

Keep finger away from channel 1 then press Enter……Done

Put finger on channel 1 then hit press……Done

########## Detect channel 2 configuration ##########

Keep finger away from channel 2 then press Enter……Done

Put finger on channel 2 then press Enter……Done

……
```

After a scan is completed, this tool will show the configuration array through UART, and the application that wants to use the touch key library can cut and paste this array in its source code.

3. Touch Key Library APIs

3.1. Definition of Enum, Structure, and Type

libtk_resolution_e

Enum indicates the resolution of specified slider and rotor. Each slider and rotor can have its own resolution setting. When the resolution of a component is set to *X* and the component is pressed by a finger, possible return values are from 0 to *X*.

Enumeration Identifier	Value	Description
LIBTK_RESOLUTION_4	2	Configure resolution to 4
LIBTK_RESOLUTION_8	3	Configure resolution to 8
LIBTK_RESOLUTION_16	4	Configure resolution to 16
LIBTK_RESOLUTION_32	5	Configure resolution to 32
LIBTK_RESOLUTION_64	6	Configure resolution to 64

libtk_channel_config_s

The best channel setting found by the touch key configuration tool.

Structure Element	Data Type	Description
base	uint16_t	Channel base value
diff	uint16_t	Threshold used by TK lib
current	uint8_t	Charge current level
div	uint8_t	Timer clock divider

libtk_callback

Prototype of callback functions for status change.

LIBTK calls registered callback function of the component(s) which status changed during last conversion or scan of all enabled component complete.

The first parameter is the current status, 0xFFFF means finger off. 0 means a key is touched for key component. 0~resolution for rotor and slider indicates the current finger position.

The second parameter is set by user application. Application can use this parameter to identify the component if multiple components share a single callback function.

Prototype
void (*libtk_callback)(uint16_t status, uint16_t param)

3.2. Functions

Note: All functions are non-blocked.

tk_add_key

Prototype

int tk_add_key(uint8_t ch, libtk_callback cb, uint32_t param);

Description

Add a key component. This API disables the newly added component by default. Structure holds key component is allocate at run time. If heap is too small, allocation may fail.

Parameter

ch [in]

Channel composed of key components.

cb [in]

Callback function while component status changed.

param [in]

Parameter sends to callback function.

Include

libtk.h

Return Value

-1	Failed
Other value	Component ID of this key.

tk_add_rotor

Prototype

int tk_add_rotor(uint8_t *ch,
uint8_t num,
libtk_resolution_e res,
libtk_callback cb,
uint16_t param);

Description

Add a rotor component. This API disables the newly added component by default. The channels that form the ***rotor*** do not need to be consecutive channels. The

structure that holds rotor component is allocated at run time. If heap is too small, allocation may fail.

Parameter

ch [in]

Array of channel composite the rotor. The order should be clockwise starting from 12 o'clock position.

num [in]

Number of channels form this rotor. Minimum value is 4.

res [in]

Resolution of this rotor. The resolution could be 4, 8, 16, 32, or 64 as introduced in the description of libtk_resolution_e in previous section. Note that the number of channels cannot be larger than the resolution.

cb [in]

Callback function while component status changed.

param [in]

Parameter sends to callback function.

Include

libtk.h

Return Value

-1	Failed
Other value	Component ID of this rotor.

tk_add_slider

Prototype

```
int tk_add_slider(uint8_t *ch,
                  uint8_t num,
                  libtk_resolution_e res,
                  libtk_callback cb,
                  uint16_t param);
```

Description

Add a slider component. This API disables the newly added component by default. The channels form the rotor does not need to be consecutive channels. Structure holds slider component is allocate at run time. If heap is too small, allocation may fail

Parameter

ch [in]

Array of channel composite the slider follow the geometrical sequence on board.

num [in]

Number of channels form this slider. Minimum value is 4.

res [in]

Resolution of the rotor. The resolution could be 4, 8, 16, 32, or 64 as introduced in the description of libtk_resolution_e in previous section. Note that the number of channels cannot be larger than the resolution.

cb [in]

Callback function while component status changed.

param [in]

Parameter sends to callback function.

Include

libtk.h

Return Value

-1	Failed
Other value	Component ID of this slider.

tk_disable_component

Prototype

int tk_disable_component(uint8_t id);

Description

Disable specified component. If a component is disabled, touch key library does not scan the channel(s) of this component.

Parameter

id [in]

ID of the component to be disabled.

Include

libtk.h

Return Value

-1	Failed
0	Success

tk_enable_component

Prototype

int tk_enable_component(uint8_t id);

Description

Enable specified component.

Parameter

id [in]

ID of the component to be enabled.

Include

libtk.h

Return Value

-1	Failed
0	Success

tk_start_calibration

Prototype

void tk_start_calibration(void);

Description

Tell the touch key library to use scan data of next four rounds for calibration. It is recommended calling this function with all components enabled once after system power on or waking up from power down mode before collecting any scan result.

Parameter

None

Include

libtk.h

Return Value

None.

tk_start_sense

Prototype

int tk_start_sense(void);

Description

Trigger touch key library to scan channel(s) of all enabled component(s). After sensing all enabled component(s) is completed, the touch key library will notify user application about the component status change via registered callback function(s).

Parameter

None.

Include

libtk.h

Return Value

-1	Failed
0	Success

tk_timer_trigger

Prototype

int tk_timer_trigger(uint8_t enable, uint8_t flag);

Description

Enable/Disable timer trigger touch key.

Note:

1. *Only Timer 0 can trigger touch key on time out event. And must set Timer0's clock source from LXT or LIRC.*
2. *If timer trigger is enabled, subsequent tk_start_sense() function call will be ignored.*
3. *Please Note at most 8 channels can be enabled in timer trigger mode. Channel (0, 8) (1, 9) … (7, 15) are mutual exclusive*

Parameter

enable [in]

0: Disable timer trigger, 1: Enable timer trigger.

flag [in]

Reserved for future use. Must keep 0.

Include

libtk.h

Return Value

-1	Failed

0	Success

tk_check_state

Prototype

tk_chk_state_result tk_check_state(uint8_t id);

Description

IRQ handler shall call this API after all enabled channels are sensed.

Parameter

id [in]

ID of the component to check state.

Include

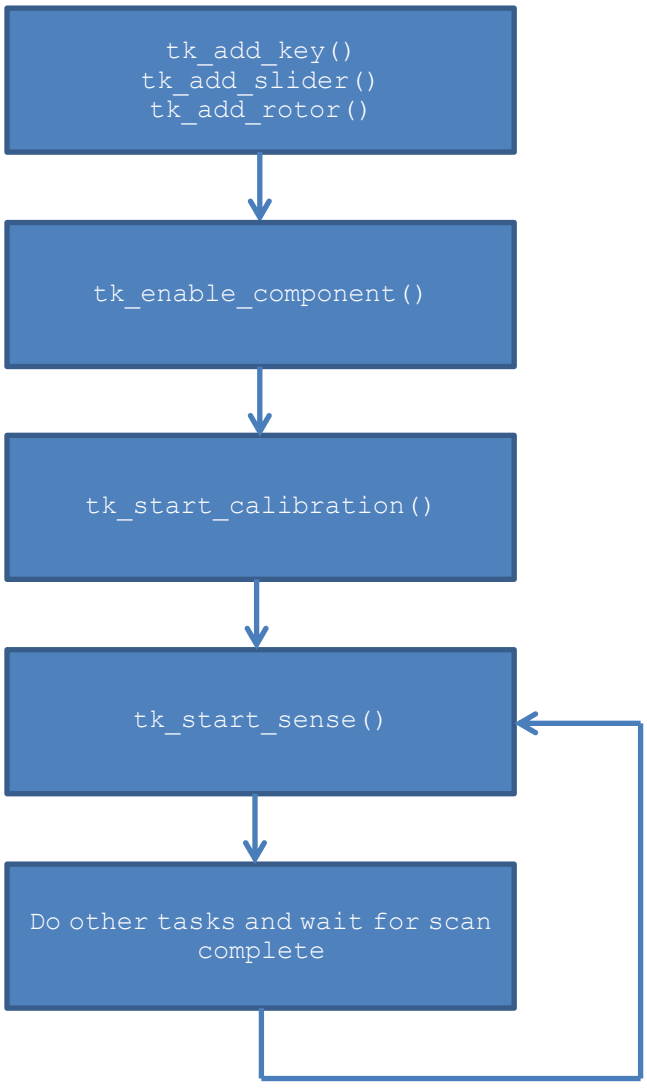
libtk.h

Return Value

LIB_TK_NOT_ENABLED	Component not enabled
LIB_TK_NOT_INIT	No component enabled
LIB_TK_SENSE_FAILED	Component sense failed
LIB_TK_CHECKED	Checked complete
LIB_TK_COMPLETE	No such component

3.3. API Calling Sequence

The following is a flow chart showing a typical touch key library API calling sequence in an application. Sample code in the next chapter also follows this procedure. The first step is to add all components in the system, and then enable the components just registered since they are disabled by default. After calibration is enabled, the application can trigger the library to scan touch key status periodically.

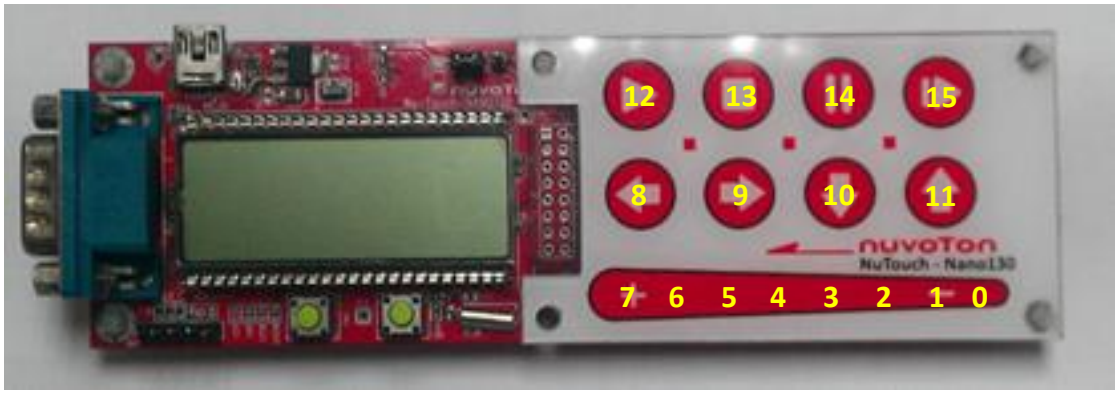


4. Sample Code

4.1. Sample Code – Software Trigger

Sample code below demonstrates the touch key function on a TK demo board. It supports 8 keys and a slider, and prints output state changes of these components to UART 0. Except the driver and library, there are 3 source files in the project. One is the main function main.c, and another is config.c which contains the configuration parameters acquired by the configuration tool mentioned in *section 1.3*. The last file is nano1xx_isr.c, which contains the IRQ handler.

The following picture shows a touch key demo board with a channel number shown on each pad.




```
/*---------------------------------------------------------------------------*/
/*                                                                           */
/* Copyright(c) 2012 Nuvoton Technology Corp. All rights reserved.           */
/*                                                                           */
/*---------------------------------------------------------------------------*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "nano1xx.h"
#include "nano1xx_tk.h"
#include "libtk.h"


uint8_t volatile complete;

// Callbak function for status change of all key components
void key_callback(uint16_t status, uint16_t param)
{

    printf("Key %d status :%s\n", param, status == 0 ? "on" : "off");
}

// Callbak function for slider states change
void slider_callback(uint16_t status, uint16_t param)
```

```
{

    if(status == 0xFFFF)
        printf("slider status :off\n");
    else
        printf("slider position :%d\n", status);

    return;
}


int32_t main(void)
{

    uint8_t slider_ch[] = {0, 1, 2, 3, 4, 5, 6, 7};
    int id0, id1, id2, id3, id4, id5, id6, id7, id8;
    int volatile i;

    printf("TK demo code begins\n");

    GCR->PA_L_MFP = (GCR->PA_L_MFP & ~(PA0_MFP_MASK | PA1_MFP_MASK)) |
                    PA0_MFP_TK8 | PA1_MFP_TK9;      // TK8, 9
    GCR->PA_H_MFP = (GCR->PA_H_MFP & ~(PA12_MFP_MASK | PA13_MFP_MASK)) |
                    PA12_MFP_TK10 | PA13_MFP_TK11;      // TK10, 11
    GCR->PC_H_MFP = (GCR->PC_H_MFP &
            ~(PC8_MFP_MASK | PC9_MFP_MASK | PC10_MFP_MASK | PC11_MFP_MASK)) |
            PC8_MFP_TK12 | PC9_MFP_TK13 | PC10_MFP_TK14 | PC11_MFP_TK15;    //
12~15
    GCR->PD_L_MFP = (GCR->PD_L_MFP &
            ~(PD0_MFP_MASK | PD1_MFP_MASK | PD2_MFP_MASK | PD3_MFP_MASK | PD4_MFP_MASK
| PD5_MFP_MASK)) |
            PD0_MFP_TK0 | PD1_MFP_TK1 | PD2_MFP_TK2 | PD3_MFP_TK3 | PD4_MFP_TK4 |
PD5_MFP_TK5;    // 0~5
    GCR->PF_L_MFP = (GCR->PF_L_MFP & ~(PF4_MFP_MASK | PF5_MFP_MASK)) |
                    PF4_MFP_TK6 | PF5_MFP_TK7;    // 6, 7

    id0 = tk_add_key(8, key_callback, 0);
    id1 = tk_add_key(9, key_callback, 1);
    id2 = tk_add_key(10, key_callback, 2);
    id3 = tk_add_key(11, key_callback, 3);
    id4 = tk_add_key(12, key_callback, 4);
    id5 = tk_add_key(13, key_callback, 5);
    id6 = tk_add_key(14, key_callback, 6);
    id7 = tk_add_key(15, key_callback, 7);

    id8 = tk_add_slider(slider_ch,
                        sizeof(slider_ch)/sizeof(uint8_t),
                        LIBTK_RESOLUTION_32,
                        slider_callback,
                        0);


    // Components are disabled by default. Enable them
```

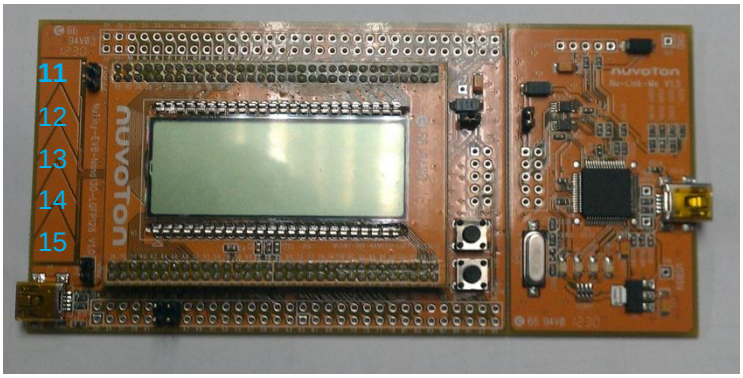
```
    tk_enable_component(id0);
    tk_enable_component(id1);
    tk_enable_component(id2);
    tk_enable_component(id3);
    tk_enable_component(id4);
    tk_enable_component(id5);
    tk_enable_component(id6);
    tk_enable_component(id7);
    tk_enable_component(id8);

    tk_start_calibration();

    while(1) {
        complete = 0;
        tk_start_sense();
        // Don't start sense before previous sense complete.
        while(complete == 0);

        // Do other job here…


    }

}
```

4.2. Sample Code – Timer Trigger

Sample code below demonstrates the touch key function on a Tiny board. It supports 1 slider composed of 5 channels. Except the driver and library, there are 3 source files in the project. One is the main function main.c, and another is config.c which contains the configuration parameter acquired by the configuration tool mentioned in *section 1.3*. The last file is nano1xx_isr.c, which contains the IRQ handler.

If slider status is unchanged for a while, the system enters Power-down mode and uses timer 0 to trigger touch key periodically. The system is only woken up if slider status is changed.

The following picture shows a Tiny board with a channel number shown on each pad.




```
/*---------------------------------------------------------------------------*/
/*                                                                           */
/* Copyright(c) 2012 Nuvoton Technology Corp. All rights reserved.           */
/*                                                                           */
/*---------------------------------------------------------------------------*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "nano1xx.h"
#include "nano1xx_timer.h"
#include "nano1xx_tk.h"
#include "nano1xx_sys.h"
#include "libtk.h"

uint8_t volatile complete;
uint32_t offcount = 0;

void slider_callback(uint16_t status, uint16_t param)
{
    char str[8];
    if(status == 0xFFFF) {
        printf("OFF\n");
    } else {
        printf("%d\n", status);
```

```
    }
    offcount = 0;
    return;
}


int32_t main(void)
{
    uint8_t slider_ch[] = {11, 12, 13, 14, 15};
    uint8_t id;
    int volatile i;

    UNLOCKREG();

    CLK->PWRCTL |= CLK_PWRCTL_LXT_EN;
    while(!(CLK->CLKSTATUS & CLK_CLKSTATUS_LXT_STB));  // wait 'til LXT stable

    LOCKREG();


    printf("TK demo code begins\n");

    GCR->PA_H_MFP = (GCR->PA_H_MFP & ~0xF00000) | 0x600000;   // TK11
    GCR->PC_H_MFP = (GCR->PC_H_MFP & ~0xFFFF) | 0x6666;   // TK12~15

    id = tk_add_slider(slider_ch, 5, LIBTK_RESOLUTION_32, slider_callback, 0);

    tk_enable_component(id);
    tk_start_calibration();

    for(i = 0; i < 20; i++) {    // for calibration

        complete = 0;
        tk_start_sense();
        while(complete == 0);
    }

    tk_timer_trigger(1, 0);

    CLK->CLKSEL1 = ((CLK->CLKSEL1) & ~CLK_CLKSEL1_TMR0_MASK ) | CLK_CLKSEL1_TMR0_LXT;
    CLK->APBCLK_BITS.TMR0_EN = 1;
    TIMER0->CMPR = 0x8000/5;         // Trigger 5 timer per second (32768/5)
    TIMER0->CTL = 0x8011;            // Periodic mode, enable trigger TK.

    while(1) {
        tk_timer_trigger(0, 0);
        CLK->APBCLK_BITS.TMR0_EN = 0;
        if(offcount++ < 20) {
            complete = 0;
            tk_start_sense();
            while(complete == 0);

        } else {        // Goes to power down if input not change for a while
```

```
            CLK->APBCLK_BITS.TMR0_EN = 1;
            tk_timer_trigger(1, 0);
            printf("SLEEP\n");
            SYS_SetUpPowerDown(0);
            __WFI();
            printf("WAKE\n");
            offcount = 0;
        }
    }
}
```

5. Revision History

Revision	Date	Description
V1.00.001	Aug.30.2012	Initially issued.

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.