AN12083

Capacitive Touch software and sensitivity test results on LPC800 CAPTouch Module

Rev.1.0 — 9 November 2017 Application note

Document information

Info	Content
Keywords	Capacitive Touch, Sensor, Calibration, Filter, Threshold
Abstract	Capacitive Touch module is newly integrated into the LPC800 series. It connects the mutual capacitance sensor when used in application. Software enables the Capacitive Touch application, with filters and algorithms for the touch key detection. Tests illustrate the sensitivity performance of the Capacitive Touch application.

Revision history

Rev	Date	Description
1.0	20171109	Initial version.

Contact information

For more information, please visit: http://www.nxp.com

1. Overview

The Capacitive Touch (CAPTouch) module is newly integrated into the NXP LPC8xx MCU. It measures the change in capacitance of an electrode plate when an earth-ground connected object (for example, the finger or stylus) is in close proximity. Based on the hardware of the Capacitive Touch module, the Capacitive Touch application implements the function to detect the touch event in an externally connected touch keyboard with more than one keypad. In the Capacitive Touch application, it senses the value of each channel connected to the touchpad, detects the touch event, and determines which touchpad is triggered.

To improve the performance and overcome unexpected noise, the application can introduce calibration and filters to process the sensor data to reduce the errors in analyzing the touch event. However, these procedures are optional in application. The developer has flexibility while implementing the features in software.

2. Hardware

2.1 Introduction

The Capacitive Touch module measures the change in capacitance of an electrode plate when an earth-ground connected object (for example, the finger or stylus) is in close proximity. The module delivers a small charge to an X capacitor (a mutual capacitance touch sensor), then transfers that charge to a larger Y capacitor (the measurement capacitor), and counts the number of iterations necessary for the voltage across the Y capacitor to cross a predetermined threshold.

The finger or stylus will impact the fringe capacitance field (between mutual electrodes of the X capacitor), effectively adding to the charge built-up on the X capacitor. Once the threshold is crossed, the Y capacitor is discharged, and the process is repeated for the next X sensor. The number of iterations necessary to cross the threshold with a no touch sensor is used as the baseline count. For a given system and its grounding characteristics, a touch sensor triggers at a higher count than a no touch sensor.

Once a calibrated and configured system operates normally, each X sensor generates either a ‘no touch’ or a ‘touch’ event every time that sensor is polled. These events can be used to generate interrupts, and/or DMA activity. The system requires re-calibration when environmental factors change, including temperature and humidity, which affect fringe capacitance fields.

2.2 Capacitive Touch working states

Capacitive Touch works on the principle of Switched Capacitor Integration circuit as shown in Fig 1. It consists of two capacitors (Cx-transfer and Cs-integration capacitor, controlled by two switches (S1 and S2) switched in non-overlapping fashion. When S1 is closed Cx charges to Vcc. Then S1 is opened and S2 is closed. This results in transfer of charge stored in Cx to Cs until both are at same potential. It is termed as one charge cycle where charge that is first stored in Cx is shared with Cs by alternate switching of S1 and S2. The value of Cs is chosen to be very large compared to Cx. Therefore, multiple charge cycles result in integration of charge stored on Cs (consequently increasing the voltage of Cs) and so, the name switched capacitor integrator circuit. After Cs is sufficiently charged to a measurable voltage, it is discharged using S3. One complete charge cycle of Cs forms an integration cycle (it is composed of multiple Cx charge cycles).

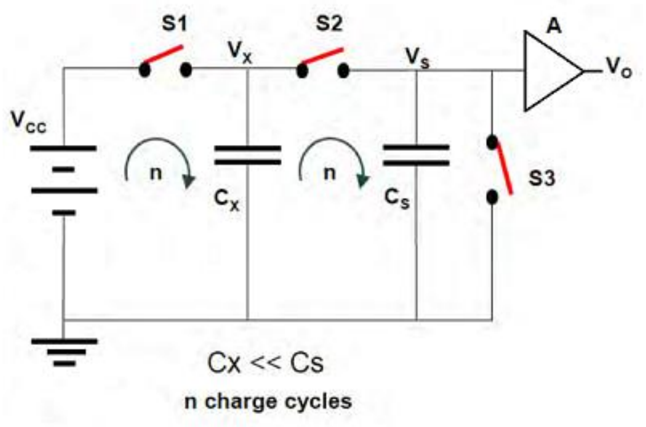


Fig 1. Switched capacity integrator

A touch sensor circuit can be realized using switched capacitor integrator circuit by replacing the transfer capacitor Cx by a touch sensor that changes its mutual capacitance on touch event. When the sensor is touched with finger, some amount of charge stored on it is lost. This results in less amount of charge being transferred to Cs in every charge cycle. Therefore, in case of a touch event, it would require more charge cycles to reach the required measurement voltage level on Cs. In other words, the touch circuit requires more charge cycles in case of touch, to reach the same voltage level as the no-touch case. By calculating the difference in charge cycles with and without touch event, it is possible to detect touch event.

The equivalent touch sensing circuit is shown in Fig 2. The Cx is represented by a touch sensor layout and Cs as discrete passive integration capacitor. The switching is realized using MCU GPIO pins.

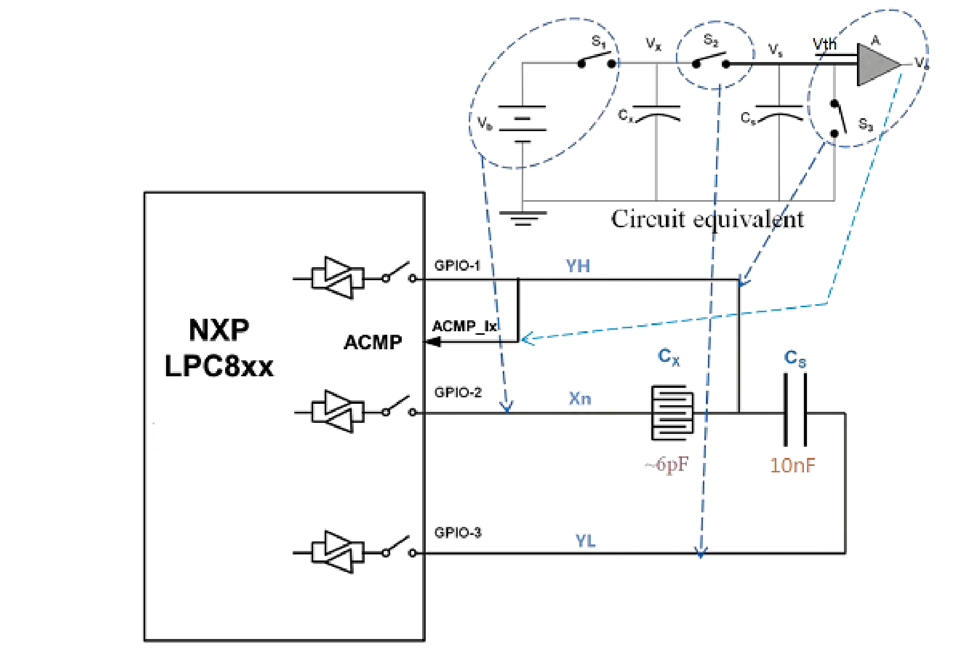


Fig 2. Touch sensing circuit

2.3 Touch sensor layout guideline

Print the electrodes on the PCB board to implement the touch sensor. The sensor is created by the conductor with a special shape. Additionally, to improve the sensitivity some parameter of the sensor's shape is considered. Per the experiment, a typical PCB sensor layout can be done with guidelines shown in Fig 3. Then, multiple sensors can be assembled as matrix, slider, or wheel, as examples.

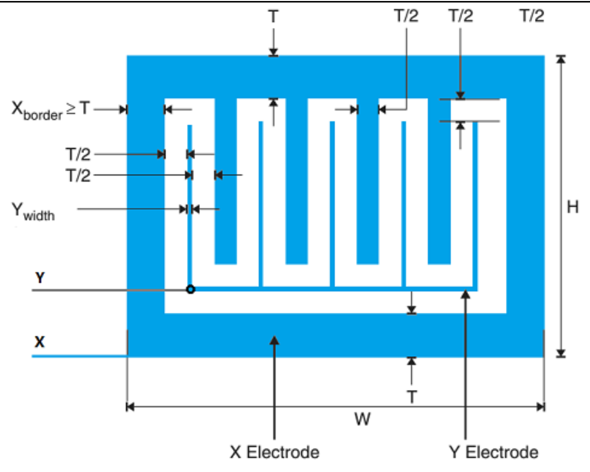


Fig 3. Planar touch pad

For this PCB sensor pad design:

- T: The thickness of front panel, which is usually an acrylic board. The thinner the front panel, the lower the sensitivity. Normally, the available range is 1mm ~ 3mm.
- Y_width: The width of Y electrode inside the sensor. The thinner the Y_width, the better the sensitivity. Normally, the available range is 0.1mm ~ 0.5mm.
- W, H: The area of finger touch, per the size of user touch area. It is better to fit the finger print, which is about 15 mm^2.
- N_fingers: The count of X fingers. The higher the count, the bigger is the Cx. Normally, the count is larger than 3.

There are also some guidelines for tracing:

- X routing is trivial as long as RC time constant rules are observed. Nearby foreign signals that have large kHz frequency switching transients should be routed well away from X traces as they can disturb the change transfer.
- X routing is not touch sensitive, and so X traces can be routed with ease on any layer of a PCB, including the side nearest to touch.
- Avoid routing Y traces over or close to ground planes (or other power planes).
- False key detection can occur wherever there is an interaction of the Y traces with the X traces. Anywhere the X and Y get close (< 10mm), the field between them can be influenced by touch. There would be a potential key or at least a touch sensitive zone that should be avoided.
- Keep the X and Y traces thin where possible.
- Route the Y traces farthest from touch where possible.
- Keep all the X traces together.

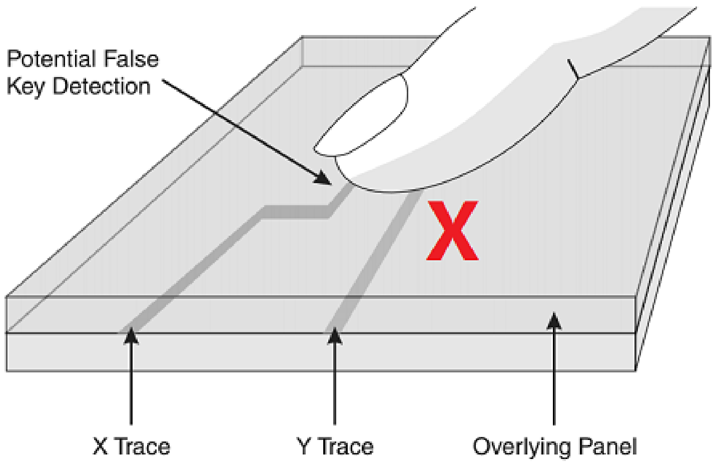


Fig 4. False key detection

Fig 5 shows a Capacitive Touch demo board.

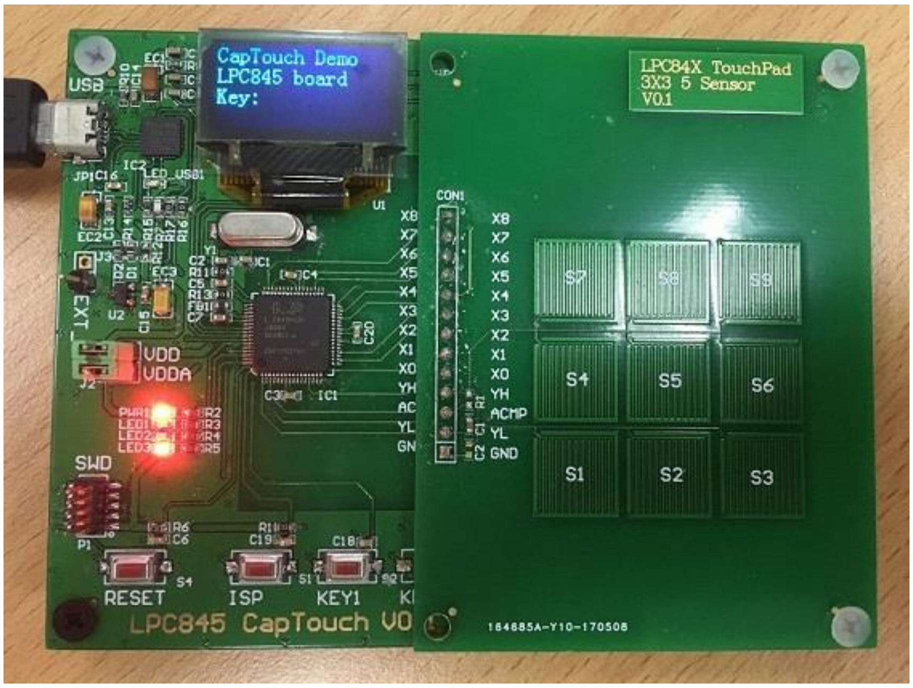


Fig 5. Capacitive Touch demo board

3. MCU Software

3.1 Introduction

Like any other HMI (Human Machine Interface) device, noise and jitter exist along with the original sensing data when reading the sensor count to detect the touch event. In the Capacitive Touch on the LPC84x device, software executes the algorithm to find out which key is touched, reduces the noise and unexpected trigger.

In this Capacitive Touch demo application, there are software for basic detection method for touch event and filters (Median Filter, Symmetric/Asymmetric Debounce Filter) to decrease the noise of sensor counts.

In some specific use cases, all the touch pads are designed uniformly and assembled together as a keyboard. Then, a simplified method can be applied to the application. The simplified method benefits from the hardware more and has better response to the external trigger.

3.2 Basic detection method

The basic usage of Capacitive Touch application is to detect the keys by considering each touch pad individually, with each sensing channel having different trigger thresholds. Therefore, in the software, the threshold values for each sensing channel are kept in memory, while there is only one threshold register (CAPT_TCNT[TCNT]) in hardware. Also, the filters for each sensing channel are maintained separately in the software.

The basic detection method is designed based on the hardware feature of the Capacitive Touch. All the nine X pins are connected to their own touch pads. The sensor count for all the X pins drops when any one touch pad is touched. The sensor count of the no touch channels is much lower than that of the touched. See Fig 6.

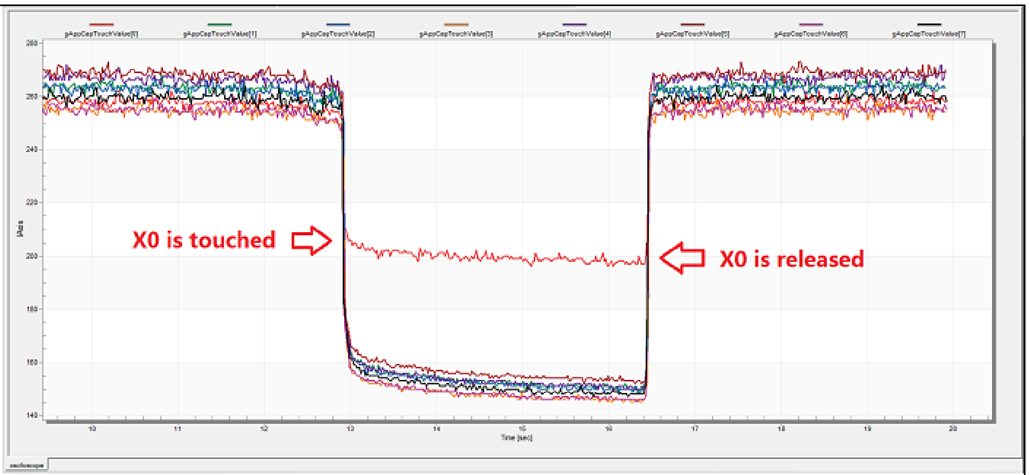


Fig 6. Waveform of sensor counts

Therefore, the software sets up the threshold value for each sensor of X pin. The no touch sensor count drops below its previous threshold value. The threshold is updated as the application condition changes. The touched channel sensor count keeps above its own threshold value. To detect a touch event, the software collects all the channel sensor counts of the available touch pads, compares the sensor counts to their own threshold values, and determines the touched pad. See Fig 7.

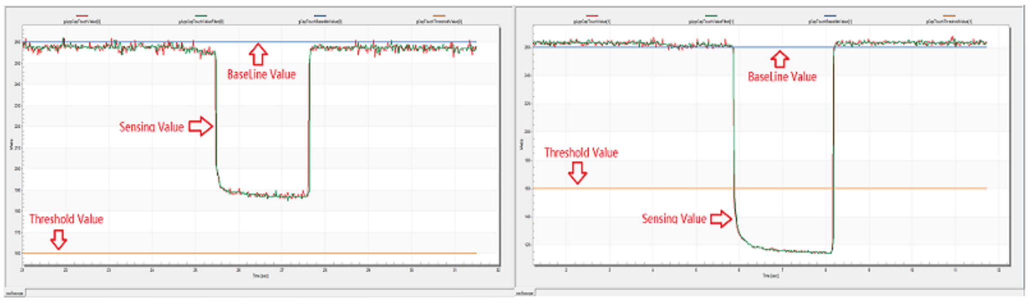


Fig 7. Waveform of sensor counts touch and no touch

Note that the current application software detects the touch for one pad. When more than one pad is touched, it requires additional procedure to recognize the touched keys.

3.2.1 Self adaptation of Artificial Intelligence

In the idle state, while none of the X channel sensor counts drop below the threshold value, the base line can be updated in run-time according to the changes in the environment. This is called “Self adaptation Artificial Intelligence”.

3.2.2 User API

The software is designed to hide the complex implementation of the procedure and enable the touch feature quickly. Therefore, the APIs (Application Program Interface) are used to access the Capacitive Touch feature. Use the following steps in the user application code:

1. Insert the API of “App_CapTouch_ProcessHook()” function into a hardware timer interrupt routine, so that it can be called periodically in about 10ms.
2. Call the API of “App_CapTouch_Calibration()” function when initializing the application to prepare the CapTouch feature.
3. Fetch the touch key value from the message queue with “App_CapTouch_GetTouchKey()”. It can be called anywhere where the touch value is used. Then, this API would return an available value (the ID of sensing channel) if any available touch event is buffered. No touch event would be missed even the application did not process it in time, since there is a FIFO inside to kept the touch key information in a timely sequence.

The sample code of the architecture would be:

```
/* main.c */


...
#include "app_captouch.h"


#define APP_CAPTOUCH_X_PIN_COUND 9U


/* Hardware will fill the sensing values of each X pin into this array. */
extern uint32_t gAppCapTouchValue[APP_CAPTOUCH_X_PIN_COUND];


int main(void)
{
    uint8_t touchKeyId;


    ...
    App_CapTouch_Calibration();
    SysTick_Config(BOARD_SYSTEM_CLOCK_HZ/100UL); /* Setup a periodical
timer. */


    while (1)
    {
        ...
        touchKeyId = App_CapTouch_GetTouchKey();
        if (0xFF != touchKeyId) /* OxFF means empty. */
        {
            /* Todo: Process the touch key value. */
            printf("TouchKey: %d\r\n", touchKeyId);
        }
        ...
    }
}

void SysTick_Handler(void)
{
    App_CapTouch_ProcessHook();
}

/* EOF. */
```

With the message queue, the Capacitive Touch module can be easily ported to multiple-task running environment (RTOS).

3.2.3 Median filter

A median filter is a nonlinear digital filtering technique, usually used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of post processing. In the Capacitive Touch demo application, it is used to pre-process the original sensor count for each X pin to reduce the glitches.

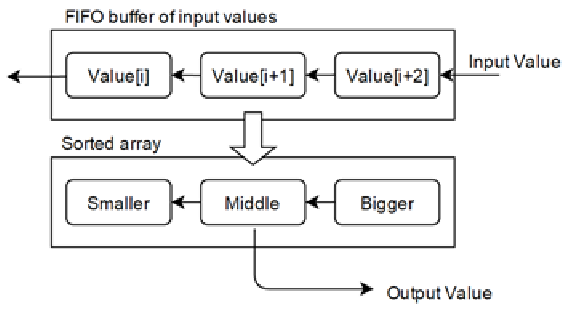


Fig 8. Median filter diagram

In the demo application code, the median filter is designed for pre-processing the sensor count. There is a group of FIFOs in software, and the FIFO contains several values (the count is defined by the value of ‘APP_CAPTOUCH_VALUE_BUFFER_COUNT’) for each X pin. In the code, the “gAppCapTouchValueBuffer[APP_CAPTOUCH_X_PIN_COUND][APP_CAPTOUCH_VALUE_BUFFER_COUNT]” is used as the FIFO to hold the original sensor counts for each sensor, sort them, and select the middle value.

With the median filter, the sensor counts are transferred to the array “gAppCapTouchValueFilted[APP_CAP_TOUCH_X_PIN_COUND]”, which keeps the values for future process.

3.2.4 Symmetric debounce filter

A symmetric de-bounce filter is designed in Capacitive Touch application to process the threshold-crossing event.

Similar to the mechanical key that generates glitches when pressed, the sensor count of the touch pad bounces around the threshold value. To get rid of glitches when crossing the threshold value, the symmetric debounce filter allows the output value to change only when the new value is kept stable for some time. There is a cache inside the filter and it changes when the input is different from the stable value. Only when the cache value becomes stable, the new value can be the output, otherwise, the output is always the previous stable value.

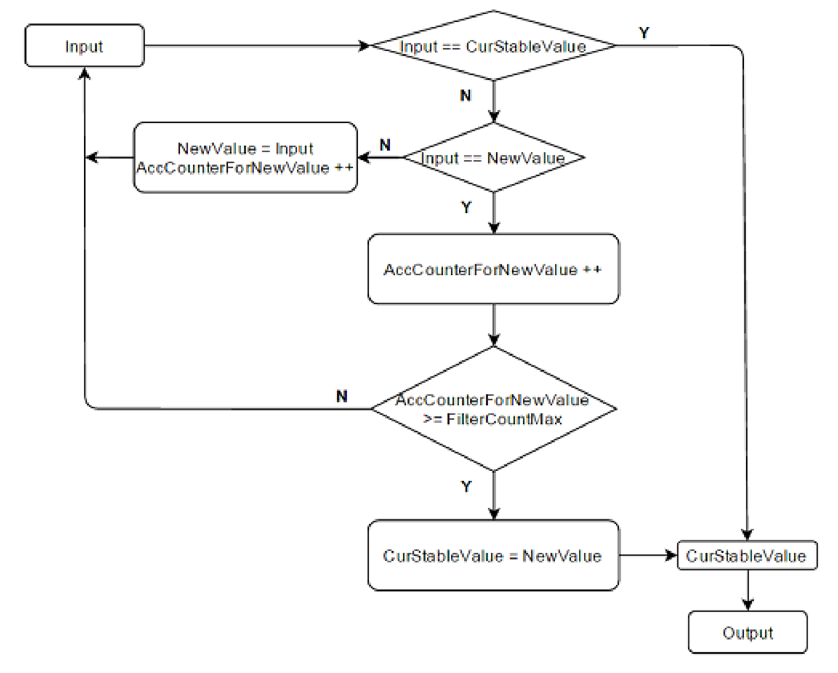


Fig 9. Symmetric debounce filter diagram

The structure type “KeyFilterHandler_T” is defined in the demo code. This keeps track of the internal state of the filter. The filter is implemented with an API: “Key_Filter_GetSymmetricDebounceOutput”. The following is an application code example:

```
/* main.c */

#include "key.h"

#define APP_KEY_FILTER_INIT_VALUE   0U
#define APP_KEY_FILTER_COUNT_MAX    10U

extern uint32_t inputValue;

int main(void)
{
    Key_FilterHandler_T keyFilterStruct;
    uint32_t outputValue;

    /* Initialize the filter. */
    Key_Filter_Init(&keyFilterStruct,
                    APP_KEY_FILTER_INIT_VALUE,
                    APP_KEY_FILTER_COUNT_MAX);

    while (1)
    {
        /* Filter: Input the value and get output. */
        outputValue = Key_Filter_GetSymmetricDebounceOutput(
                          &keyFilterStruct, inputValue);
    }
}

/* EOF. */
```

With the symmetric debounce filter in this Capacitive Touch demo, the glitches are reduced significantly during the threshold crossing and unexpected touch triggers are removed.

3.2.5 Self-adapted baseline and asymmetric debounce filter

In the application, when the environment changes, the sensor count may drift. For example, water drop, wet air, temperature, or electromagnetic interference can affect the sensor count.

A self-adaptable algorithm is integrated into the application software. It can trace the minor change of sensor count, and update the baseline and threshold value in idle state automatically for each sensing channel. The idle state here for the keyboard means none of touch pads is touched, while all the sensor count of channels are above their own threshold values. Fig 10 shows the diagram of the waveform.

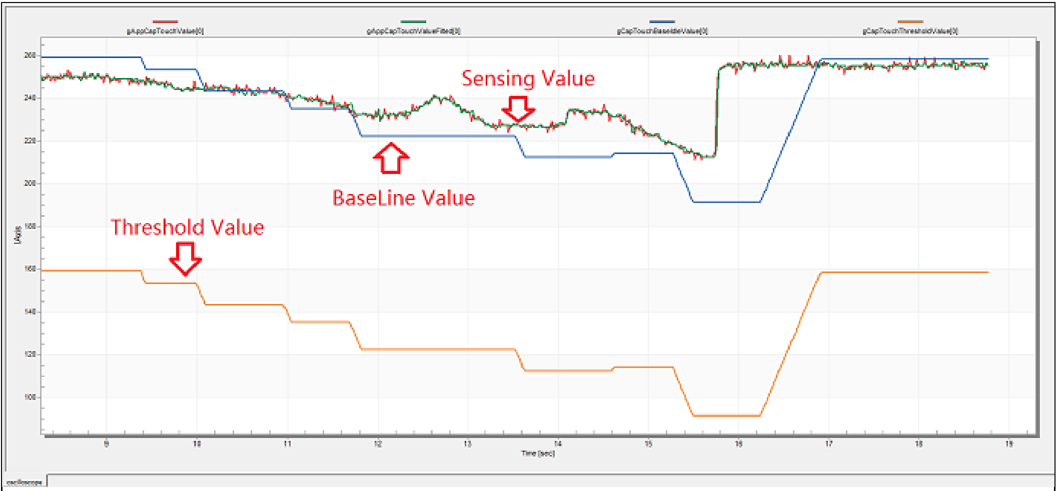


Fig 10. Self-adapted baseline and asymmetric debounce filter diagram

An asymmetric debounce filter is designed in CapTouch application here. It is a common filter based on the symmetric debounce filter, but it promotes the priority for indicated state. There would be three kinds of states: below the idle range, inside the idle range, and above the idle range. The state of inside the idle range has higher priority. Once sensing event comes, the output of the filter would change to idle state immediately, while the transfer to other states would have to wait for a few periods to make sure the state is stable.

```
uint32_t Key_Filter_GetAsymmetricDebounceOutput(
                Key_FilterHandler_T *handler, uint32_t curUnfiltedValue)
{
    if (curUnfiltedValue == handler->InitValue)
    {   /* everything turns to the initial ones */
        Key_Filter_Init(handler, handler->InitValue, handler->FilterCountMax);
    }
    else
    {
        handler->CurStableValue =
                Key_Filter_GetSymmetricDebounceOutput(handler, curUnfiltedValue);
    }
    return handler->CurStableValue;
}
```

When running the self-adapted base algorithm, the software must know how to change the baseline of the threshold value. Changes in the threshold value depends on the comparison between the stable sensor count in the idle state and the current sensor count.

The software for the threshold crossing compares the threshold value and sensor count. A debounce filter is used to get rid of the glitches when crossing the threshold. However, the idle state is a range and not a value. When the sensor count is within the idle range, the base line does not change. The debounce filter for the idle range is asymmetric, while the output stops immediately when the baseline falls into the idle range.

To assign the numeric code for different states, the input to the filter is the state code in addition to the boundary value.

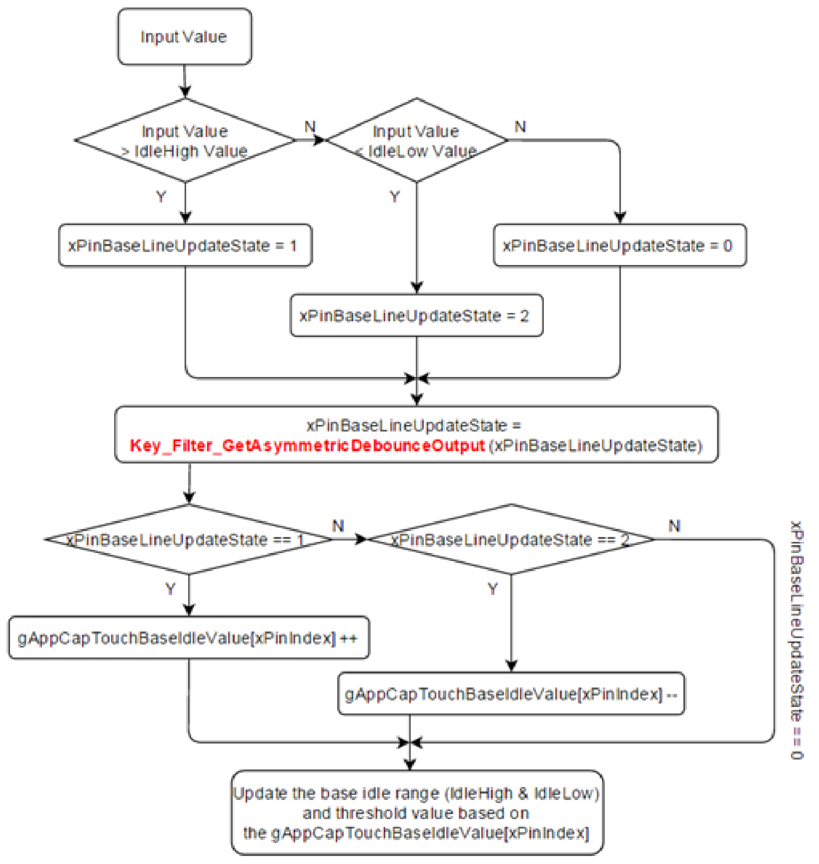

Input Value
Input Value > IdleHigh Value
N
Input Value < IdleLow Value
N
Y
Y
xPinBaseLineUpdateState = 1
xPinBaseLineUpdateState = 0
xPinBaseLineUpdateState = 2
xPinBaseLineUpdateState = Key_Filter_GetAsymmetricDebounceOutput (xPinBaseLineUpdateState)
xPinBaseLineUpdateState == 1
N
xPinBaseLineUpdateState == 2
N
Y
Y
gAppCapTouchBaseIdleValue[xPinIndex] ++
gAppCapTouchBaseIdleValue[xPinIndex] --
xPinBaseLineUpdateState == 0
Update the base idle range (IdleHigh & IdleLow) and threshold value based on the gAppCapTouchBaseIdleValue[xPinIndex]

Fig 11. Diagram of asymmetric debounce filter

The auto-tracing algorithm in the Capacitive Touch application supports anti-interference.

3.2.6 The periodical process hook and message queue

The Capacitive Touch demo is designed that the touch event is processed in background periodically (triggered by a hardware timer). User would never see the process routine in main loop. The only interface to application in main loop is just an API of message queue, which keeps the detected events so that the user can fetch and process them in their application. With this design of the message queue for touch events, the user application integrated with Capacitive Touch is simple and easy.

To simplify the user's operation, all the algorithm and operations are packed inside the "App_CapTouch_ProcessHook()". User only needs to call it in any timer interrupt routine so it can be executed in every about 10ms. It would fill the touch code into the message queue when any touch event happens, so that user can fetch the touch code through the message queue in application.

The "App_CapTouch_ProcessHook()" is the main process thread of the CapTouch module. The diagram in the Fig-x.x would show its workflow.

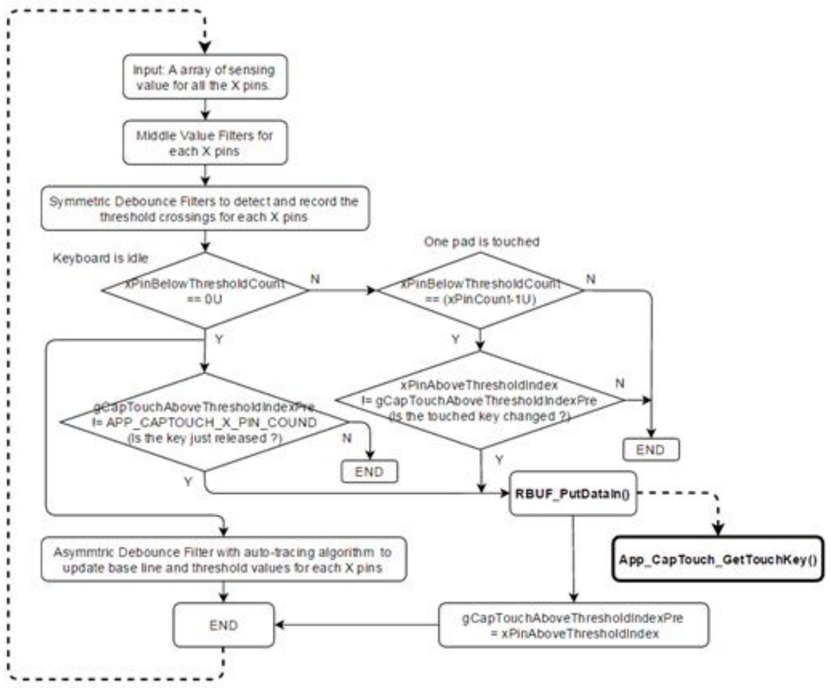

Input: A array of sensing value for all the X pins.
Middle Value Filters for each X pins
Symmetric Debounce Filters to detect and record the threshold crossings for each X pins
Keyboard is idle
xPinBelowThresholdCount == 0U
N
Y
One pad is touched
xPinBelowThresholdCount == (xPinCount-1U)
N
Y
gCapTouchAboveThresholdIndexPre != APP_CAPTOUCH_X_PIN_COUND (Is the key just released ?)
N
Y
END
xPinAboveThresholdIndex != gCapTouchAboveThresholdIndexPre (Is the touched key changed ?)
N
Y
END
RBUF_PutDataIn()
App_CapTouch_GetTouchKey()
Asymmtric Debounce Filter with auto-tracing algorithm to update base line and threshold values for each X pins
END
gCapTouchAboveThresholdIndexPre = xPinAboveThresholdIndex

Fig 12. Periodical process hook workflow

In the demo code, the "App_CapTouch_ProcessHook()" function is called in the SysTick's interrupt routine service, with the period of 10ms.

3.3 A simplified detection method

When all the touch pads are of the same shape, the threshold values are almost the same, therefore, it is unnecessary to keep separated threshold values for each channel.

The software can use the hardware feature of the register CAPT_TCNT[TCNT], which stores the threshold value. When using the CAPT_TCNT[TCNT] register, software does not have to compare the sensor counts of each channel because the hardware compares them automatically after each scan and launches the hardware trigger (the trigger event is configured in CAPT_CTRL[TRIGGER]). When the interrupt is enabled for this hardware trigger, the ISR (Interrupt Service Routine) of the Capacitive Touch is executed. Read the register CAPT_TOUCH[XVAL] in the ISR to get the X channel index of the touched pad.

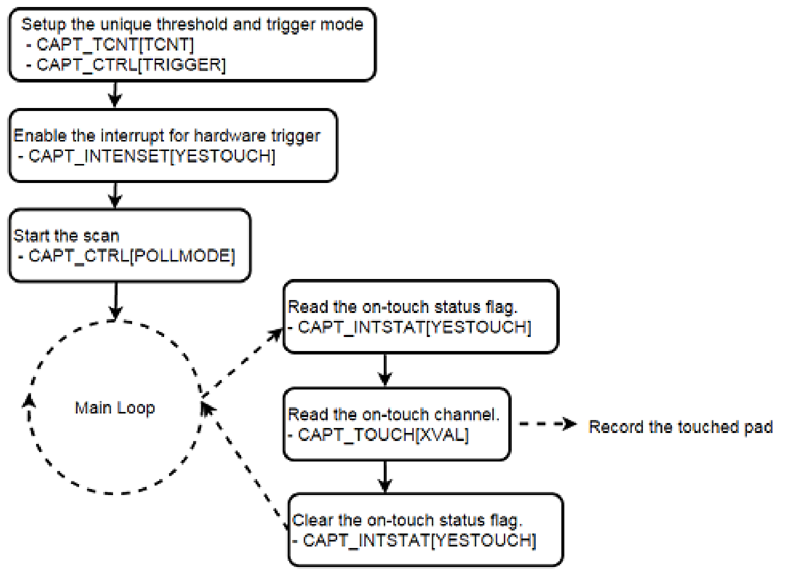

Setup the unique threshold and trigger mode
- CAPT_TCNT[TCNT]
- CAPT_CTRL[TRIGGER]
Enable the interrupt for hardware trigger
- CAPT_INTENSET[YESTOUCH]
Start the scan
- CAPT_CTRL[POLLMODE]
Main Loop
Read the on-touch status flag.
- CAPT_INTSTAT[YESTOUCH]
Read the on-touch channel.
- CAPT_TOUCH[XVAL]
Record the touched pad
Clear the on-touch status flag.
- CAPT_INTSTAT[YESTOUCH]

Fig 13. Workflow of the simplified detection

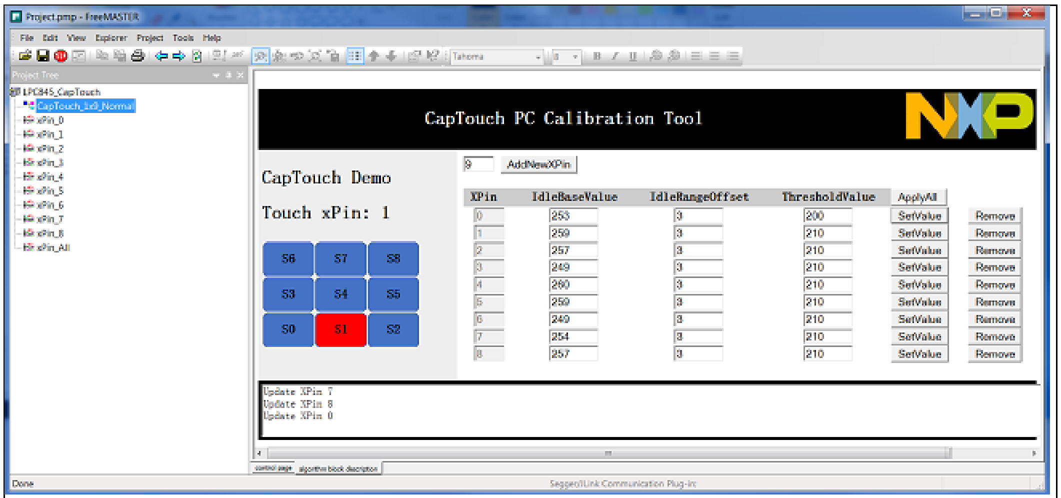
The detection method reduces the CPU load since the hardware automatically scans, compares, and records the touched channel. It also helps to reduce the power consumption.

4. PC calibration tool

4.1 Introduction

In the Capacitive Touch demo, the self-adapted baseline algorithm (filters) can update the baseline along with the environmental changes. The idle range (a range of baseline value during idle state) and the threshold values must be pre-defined in MCU software. These values depend on the user board and sensors.

To help the user find the optimal settings in debug mode, the PC calibration Tool with user-friendly GUI is provided. With the PC calibration tool, the user can monitor the baseline, idle range (“IdleRangeOffset”), and the threshold (“ThresholdValue”) in runtime. Find the optimal value and hardcode it in to the software.



CapTouch PC Calibration Tool

CapTouch Demo

Touch xPin: 1

9 AddNewXPin

XPin	IdleBaseValue	IdleRangeOffset	ThresholdValue	ApplyAll	
0	253	3	200	SetValue	Remove
1	259	3	210	SetValue	Remove
2	257	3	210	SetValue	Remove
3	249	3	210	SetValue	Remove
4	260	3	210	SetValue	Remove
5	259	3	210	SetValue	Remove
6	249	3	210	SetValue	Remove
7	254	3	210	SetValue	Remove
8	257	3	210	SetValue	Remove

```
Update XPin 7
Update XPin 8
Update XPin 0
```

Fig 14. PC calibration tool panel

4.2 Features

The PC Calibration Tool has the following features:

- Monitors the baseline, idle range, threshold values for each X pin.
- Modifies the base, idle range, threshold values in runtime.
- Shows the waveforms for each X pin.
- 1x9 Normal scan keyboard.
- Shows the touched pad in visual keyboard.
- Activates and deactivates the programmable X pins.

4.3 Operating steps

Once the PC Calibration Tool is launched (with the Freemaster already installed):

- Click the “AddNewXPin” button to activate the settings for new X Pin in the main screen (‘KeyPads_1x9_Normal’ page). The edited text block appears as the new row inside the table. “SetValue” button is used to load the setting value into board. “Remove” button is used to deactivate the row for responding X Pin. For each X Pin in a row, there are editable text blocks, including:
    - XPinIdx: The index of each X Pin. It represents the pin number inside the hardware Capacitive Touch module. The value in the respective X Pin index is set when adding the new X pin using the “AddNewXPin” button.
    - IdleBaseValue: Represents the sensor count when in idle state, where none of the keys are touched. This value for each X pin is automatically with the self adaptation algorithm. During the calibration, the user manually edits the value inside the editable text block.

  The IdleBaseValue maps the variable of "gAppCapTouchBaseIdleValue[<xPinIndex>]" in the software.

  - IdleRangeOffset: Used to generate the boundary of the idle range. Considering the noise in the sensor count, the high and low values inside the idle range are:
    IdleRangeLow = IdleBaseValue value - IdleRangeOffset value
    IdleRangeHigh = IdleBaseValue value + IdleRangeOffset.
    The user sets the new value into this text block during calibration.
    The IdleRangeOffset maps the variable of "gAppCapTouchIdleRangeOffset[<xPinIndex>]" in the software.
  - ThresholdValue: Used to generate the offset of the threshold value: (ThresholdValue = ThresholdOffset + IdleBaseValue).
    The offset value is stored in the software. The offset determines the touch or no touch event.
    The ThresholdValue maps the variable of "gAppCapTouchThresholdOffset[<xPinIndex>]" in the software.

- In the scope pages for each X Pin, the user can observe the waveform of IdleBaseValue, IdleRangeRange with low/high boundaries, threshold values, and the most recent sensor counts in run-time.

  To calibrate the touch keyboard, the user must tune the variable values for each X pin in "KeyPads_1x9_Normal" page to ensure the following:

  - The corresponding sensor count for the touched pad sensor is higher than the threshold value.
  - The no touch pad sensor counts are lower than the threshold values.
  - "IdleRangeOffset" value is used to prevent forced detection caused by sensor noise.

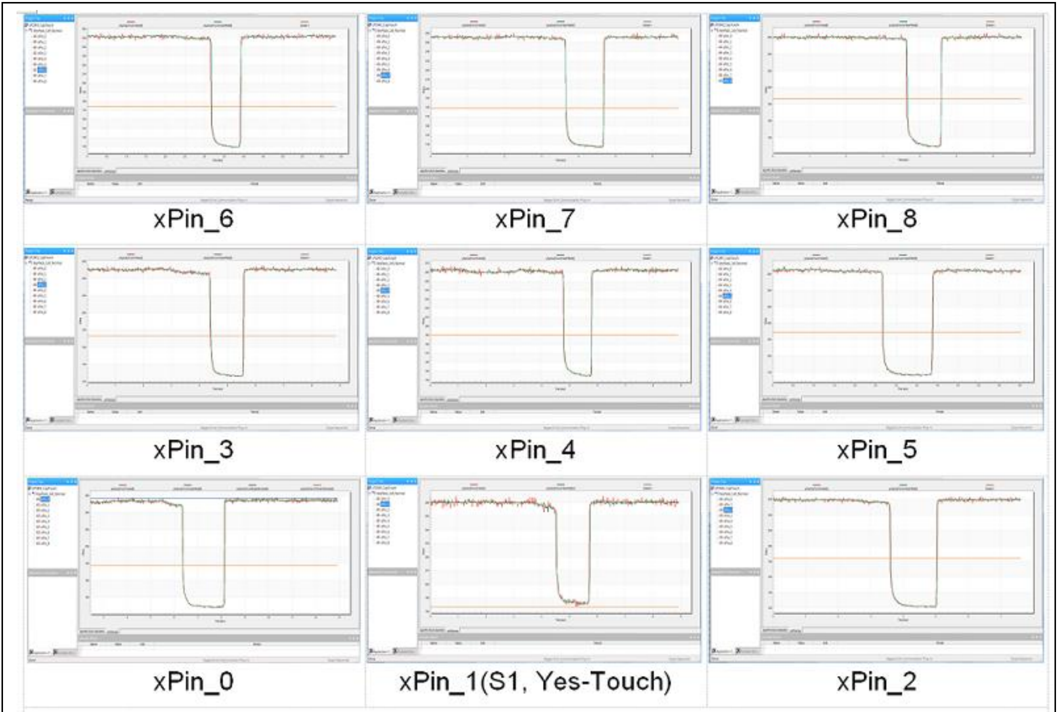


Fig 15. Calibration for xPin_1

When xPin_1's pad is touched (S1), the sensor count for X pin 1 is dropped but still above its threshold, while all the other 8 pins' sensor counts are all below their own threshold values. See Fig 15.

5. Sensitivity oerformance

5.1 Introduction

Capacitive Touch application can be used in situations with various kinds of interference affecting the sensor performance. This section describes a few test cases to simulate situations that can affect the performance of the Capacitive Touch.

5.2 Test record

The algorithms for Capacitive Touch are based on the difference between touch and no-touch sensor counts.

Note: The experiments described in this section are for qualitative analysis to help the user to determine if the test condition affects the Capacitive Touch. In the final product the results will depend on the working environment.

5.2.1 Thin Film

The thin film is a polyester film. It is soft and waterproof, and can be used as a protective material.

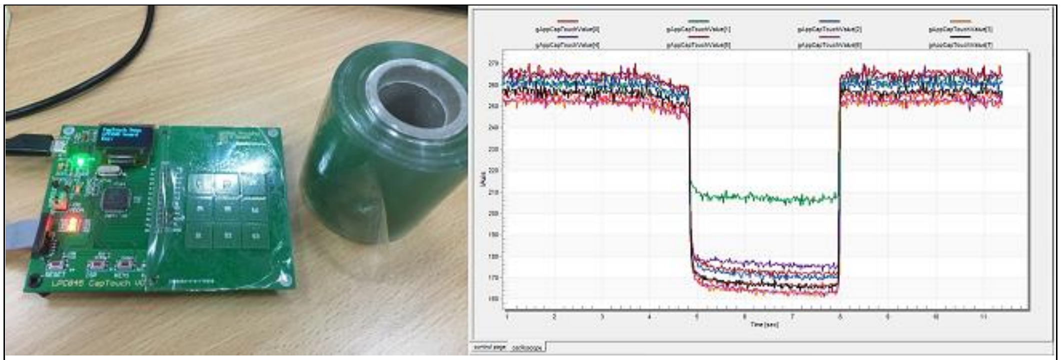


Fig 16. Touch with thin film covered

In the experiment, the waveforms show that the touched key can be easily recognized.

5.2.2 Acrylic board

The acrylic board is a typical insulation material and can be used to isolate the circuit board. In this experiment, the thickness of the acrylic board is about 2mm.

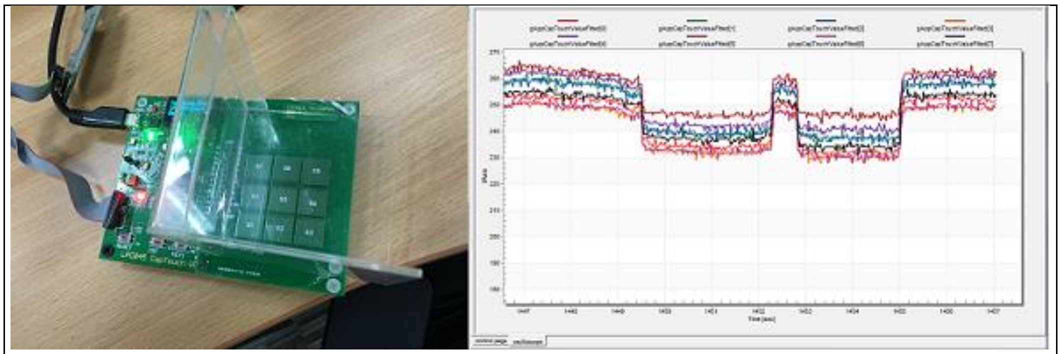


Fig 17. Touch with acrylic board covered

In this experiment, the waveforms of sensor count changes when the finger is close to the covered acrylic board. The sensor count value of the touched key is not recognized. It did not pass the test.

The failure is caused by the thickness of the acrylic board. The test passes with a thinner acrylic board (about 0.5mm).

5.2.3 Plastic glove

In this experiment, the waveform of the touch sensor count can be recognized. It passes the test.

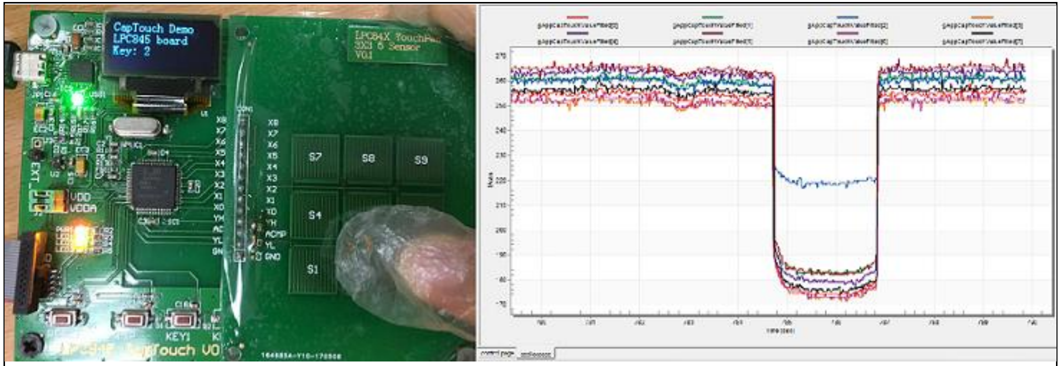


Fig 18. Touch with plastic glove

5.2.4 Rubber glove

Rubber gloves have better anti-chemical characteristics.

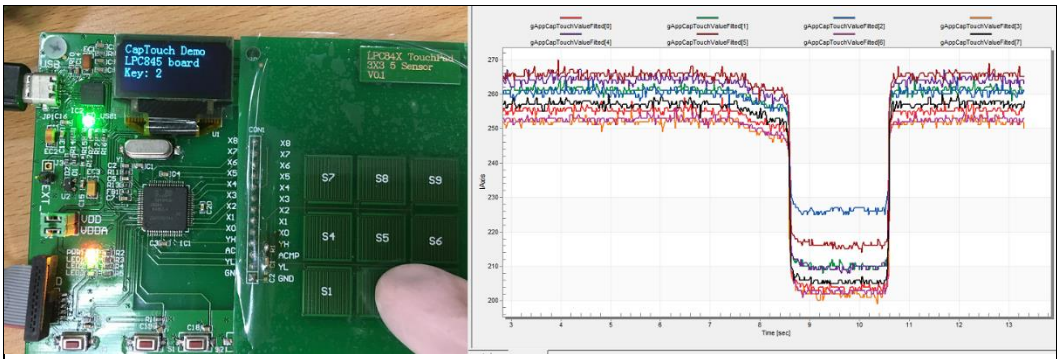


Fig 19. Touch with rubber glove

The rubber glove affects the sensor counts. It decreases the IdelBaseValue. However, the waveforms for touched keys can be recognized. By tuning the threshold value with PC Calibration Tool it passes the test.

.

5.2.5 Water / salty water

In this experiment, water drops are placed on the touch keyboard.

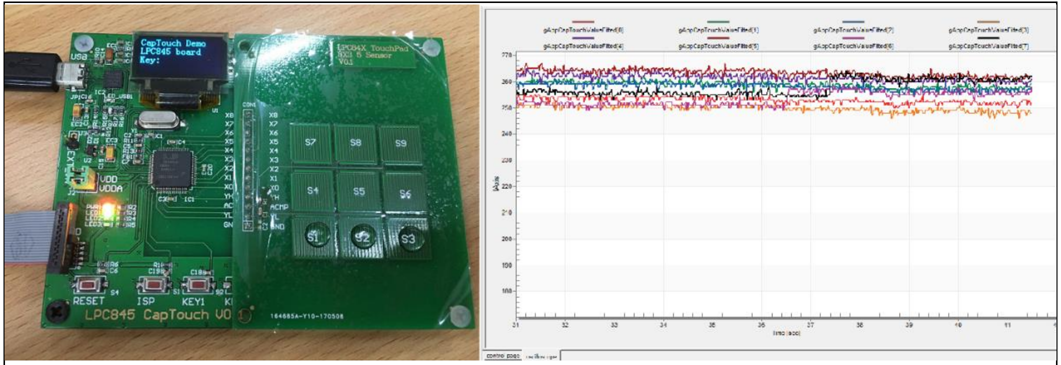


Fig 20. Idle with water on keyboard

When the water drops on the touch pads, the corresponding sensor count values change slightly. The algorithm can track the minor change of the base line automatically, so the water drop does not trigger a false touch.

When touching with finger while the water is on the touch keyboard, the demo passes the test.

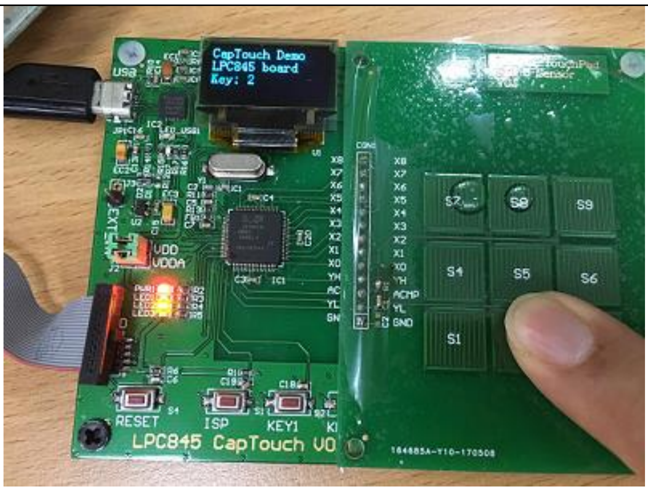


Fig 21. Touch with water

The salty water experiment also passes the test.

5.2.6 Oil

In this experiment, when oil drops are placed on the touch keyboard, the corresponding sensor count values do not change.

When touching with finger while the oil is on the touch keyboard, the demo passes the test.

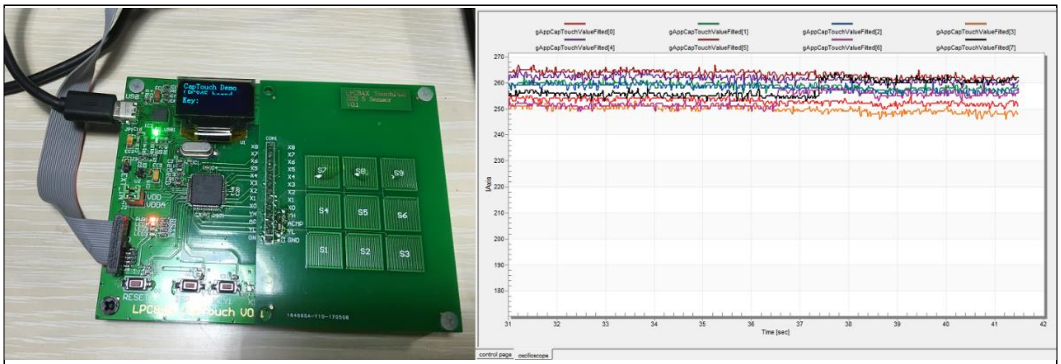


Fig 22. Idle with oil on keyboard

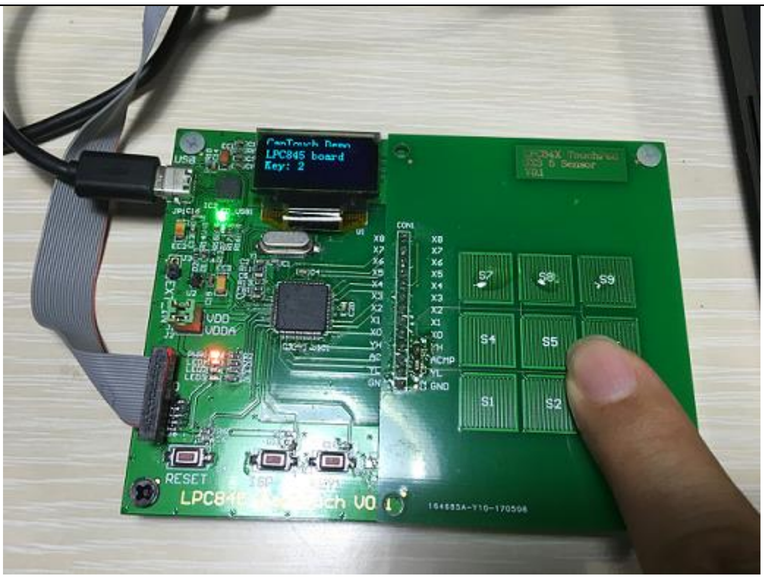


Fig 23. Touch with oil

5.2.7 Electromagnetic interference

Electromagnetic interference exists in some harsh industrial environment. This experiment simulates a harsh environment to check the tolerance of Capacitive Touch against the electromagnetic interference. A walkie is used as the source for the electromagnetic noise.

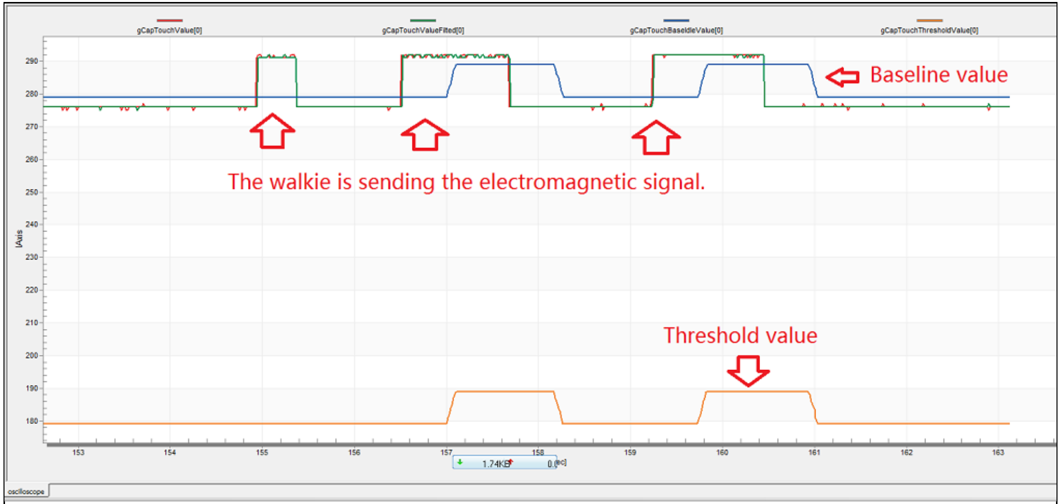


Fig 24. Touch with electromagnetic interference

Fig 24 shows the sensor count of X1 increases when the walkie sends the electromagnetic signal. When the interference is less, the algorithm of Capacitive Touch filters tracks the baseline and recognizes the touch key.

However, the radiation strength depends on the distance of the noise source. Additional experiments did show the impact of different radiation strengths.

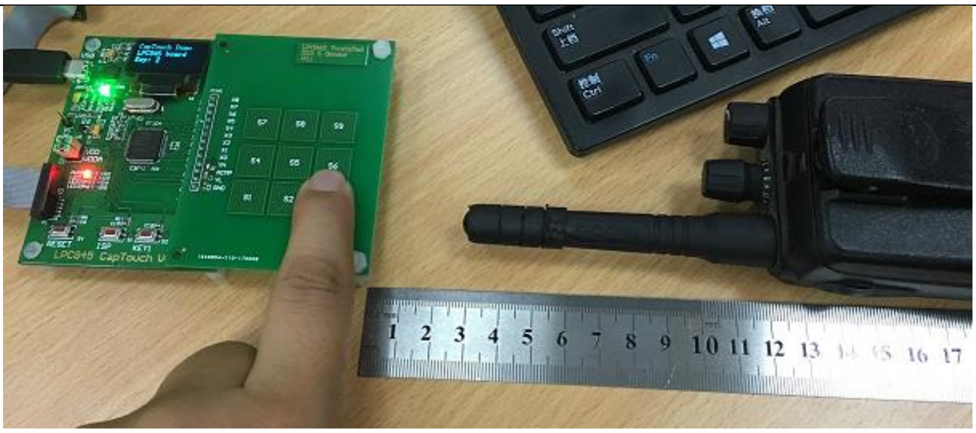


Fig 25. Place walkie near demo board

In Fig 25, when the antenna is placed about 3 cm away from the touch keyboard, the Capacitive Touch demo board recognizes the touch key. It fails when the walkie is placed less than 3 cm away. When the interference source is removed, the demo board can resume its functionality.

5.3 Conclusion

Table 1 is a summary of the Capacitive Touch sensitivity to noise and environmental changes.

Table 1. Performance test summary of Capacitive Touch demo

Condition	Result	Comment
Thin film	Pass	NO sensor count change. No addition calibration needed.
Acrylic board	No Pass	The thickness of about 2mm would make the sensing lines no distinguish of touched and no-touched key pads. Need to update the PCB sensor for better performance.
Plastic glove	Pass	No additional calibration needed.
Rubber glove	Pass	Calibrated by increase the threshold values (decrease the offset between baseline value and threshold value).
Plain water	Pass	The sensor count for the touch pad with water increases a little, but can be adopted by the algorithm in the demo program. The demo can distinguish the correct key touched.
Salty Water	Pass	Results are same as plain water.
Oily water	Pass	NO sensor count change. No addition calibration needed. The demo can detect the right touched key.
Electromagnetic interference	Pass	The sensor counts for all the touch pads increase when the walkie is sending the electromagnetic signal. The demo cannot work well when the walkie is too close, but can return to be well when the walkie is away.

6. Legal information

6.1 Definitions

6.2 Disclaimers

6.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

6.4 Patents

6.5 Trademarks

7. Index

8. List of figures

9. List of tables

10. Contents