AN12306

Enable Touch 2-dimension Gesture Recognition based on KL16Z MCU

Rev. 0 — 10/2020 Application Note

Contents

1 Introduction

A kind of touch 2-dimension (2D) sensing system using self capacitive touch sensing channels was already described in *Enable 2D Touch Sensing System Based on KE15Z MCU* (document AN12933). Based on the existing touch 2D sensing algorithm, in this application note document, we do further development to implement a simple 2D gesture recognition algorithm, on KL16Z MCU with lower cost but similar self capacitive touch sensing hardware. In the attached demo project, some gestures on the 2D touchpad can be recognized, including:

- Move up/down/left/right
- Touch 1st/2nd/3rd click
- All cover

A compliable source code project for demo is also provided as attachment.

2 Touch 2D sensing algorithm overview

First of all, let us take an overview about the architecture of touch 2D sensing method based on the self capacitive touch sensing channels.

Enable 2D Touch Sensing System Based on KE15Z MCU (document AN12933) describes the method using a group of specially designed touch pads to build a touch 2D touch sensing hardware platform, as shown in Figure 1.

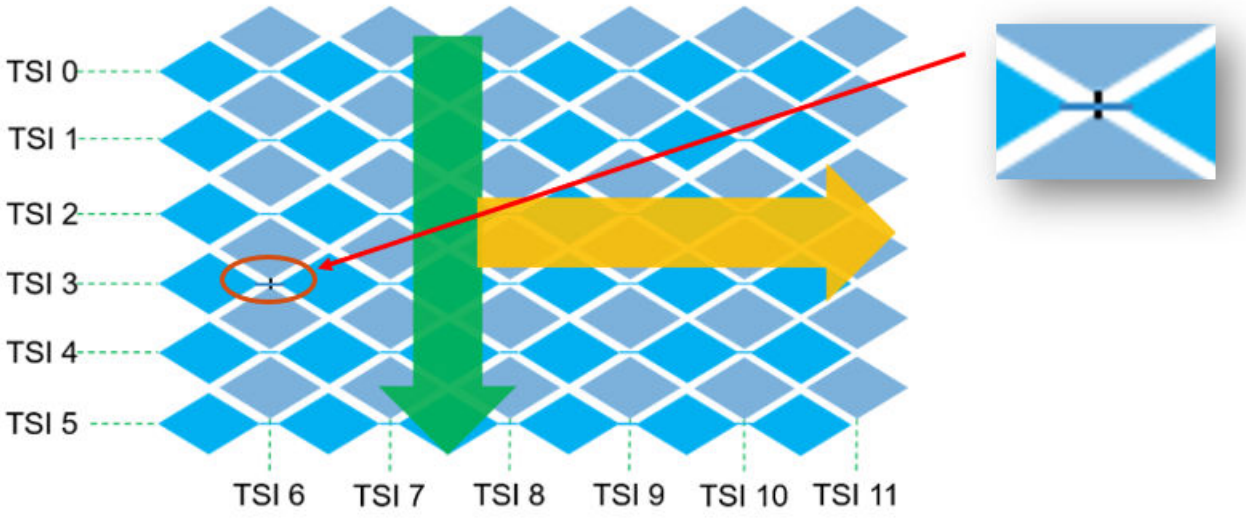

Figure 1. Touch 2D PCB diagram

Actually, they are two groups of touch electrodes organized as two sliders, interleaved with each other, to build a 2D coordinated position system. So, the method just treats the touch 2D sensing system as two touch sliders, and uses the touch slider position calculation algorithm for calculating the touched position separately inside each slider. Then, one slider brings the X value of pixel and the other slider brings the **Y** value of pixel.

A state machine about processing sensing values in the buffer is implemented to process the samples from the beginning of the application to the position detection cycles, as shows in Figure 2. When the application starts while the sample buffer is empty, the algorithm collects the samples into buffer first (Warm up), and wait for the samples are stable (Calibration), then run the process to check if the on touch or no touch event is detected and calculate the touch position.

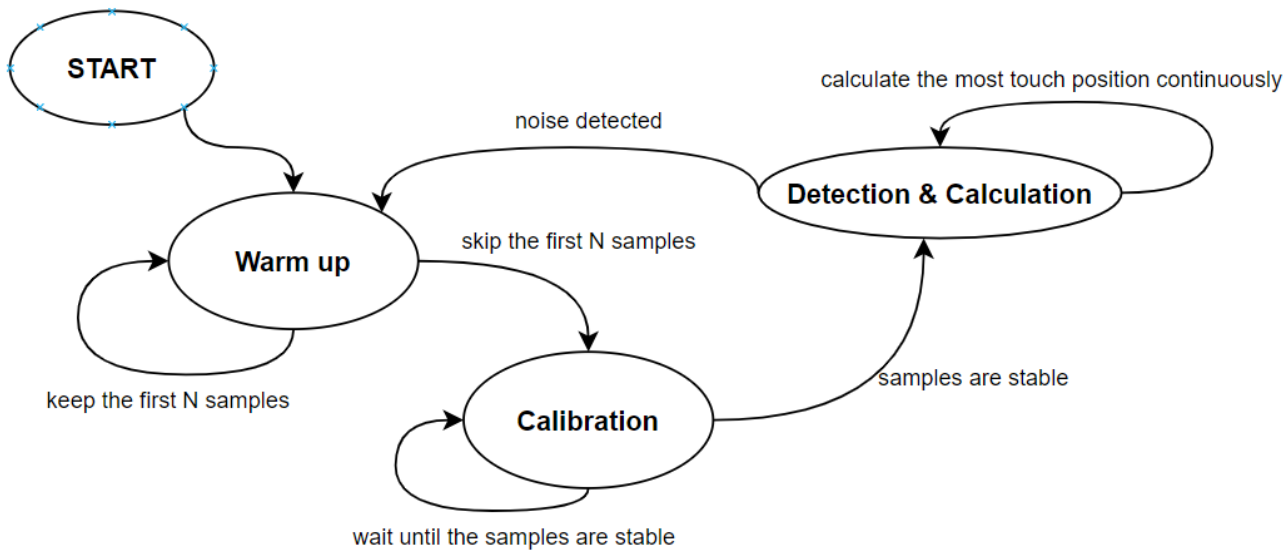

Figure 2. Touch 2D detection state machine

There are also a auto calibration and 2 level's difference during the position calculation. Once the noise is big enough, the algorithm resets the state machine and runs the calibration again. This makes the algorithm can be self-adapting without manual settings threshold for each sensing channels (even it still need only one manual setting threshold to set the sensitivity).

When scheduling the tasks of sampling and processing, there is an "ping-pang buffer" mechanism: the sampling service based on the hardware TSI interrupt would capture the current sensing values when the process is dealing with the previous sensing values, as the previous ones were copy to a buffer before starting the new sampling task. As the processing time is always shorter than the sampling time, this makes the sampling rate keeps stable and not stalls per the running the processing task. The sampling task and processing task were running parallelly, as shown in Figure 3.

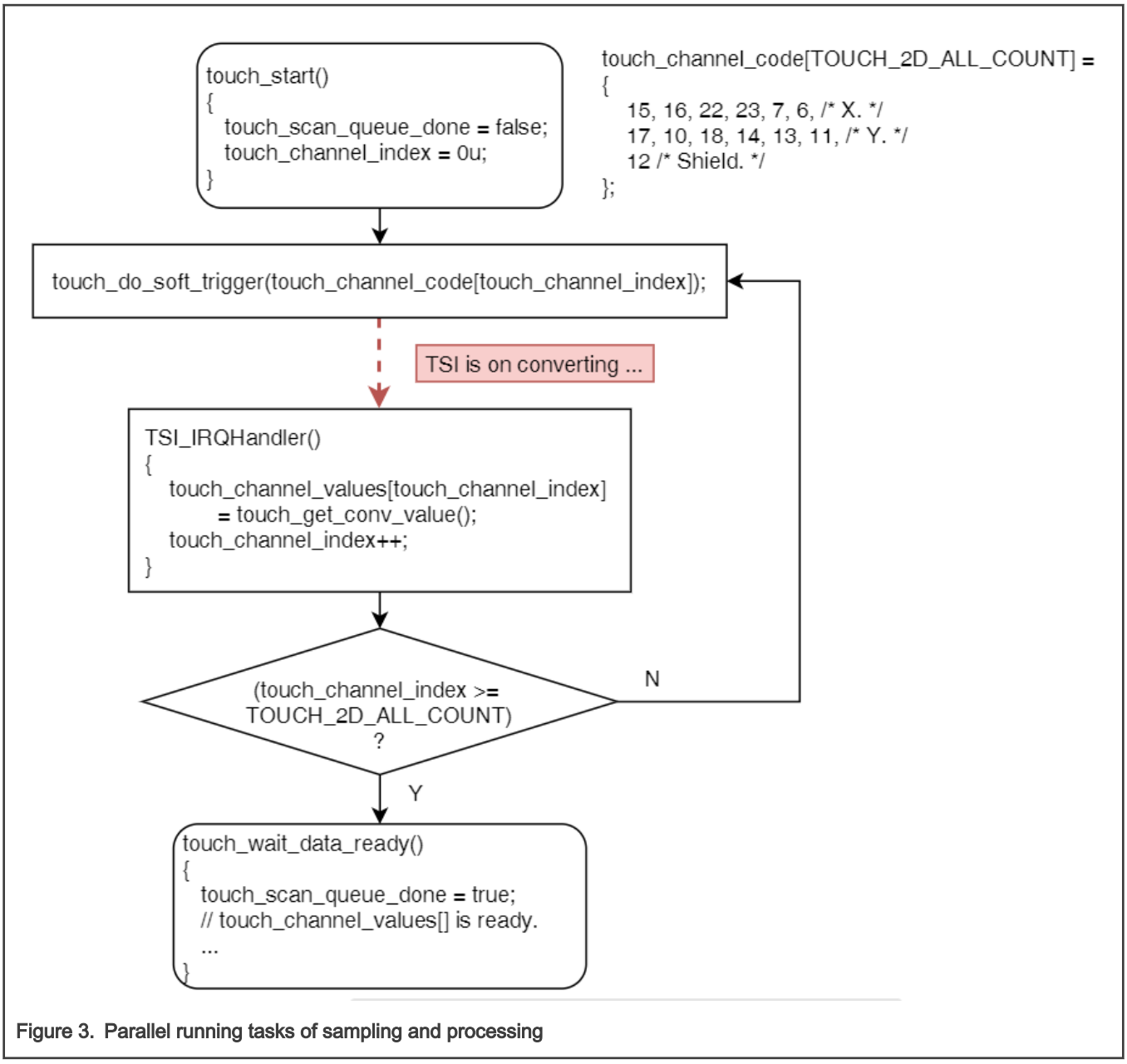

Figure 3. Parallel running tasks of sampling and processing

In the end of the processing, the position values of touched point would output through the function as follows:

```
app_touch_err = touch_2d_calc_position(app_touch_channel_values,
                                       &app_touch_2d_x_pos_raw,
                                       &app_touch_2d_y_pos_raw);
```

In this application note, the further development would be based the position values in previous algorithm, keeping the continuous calculated positions in a queue, and recognizing the gestures according the their movement of position. In the demo code, we would like to implement a function as follows:

```
app_touch_2d_gesture_event_raw = touch_2d_gesture_process( app_touch_2d_x_pos_raw,
                                                           app_touch_2d_y_pos_raw,
                                                          (app_touch_err == 0) );
```

The `touch_2d_gesture_process()` function would get the **X** and **Y** value of touched position, and whether the touch is available, then output the judgement of event as follows:

```
typedef enum
{
    eTouch_2d_gesture_event_no_touch_keep = 0, /* no touch. */
    //eTouch_2d_gesture_event_no_touch_just = 1,
    eTouch_2d_gesture_event_on_touch_just = 2, /* 1st click. */
    eTouch_2d_gesture_event_on_touch_move_0 = 3, /* no move, on touch. */
    eTouch_2d_gesture_event_on_touch_move_1 = 4, /* move up. */
    eTouch_2d_gesture_event_on_touch_move_2 = 5, /* move down. */
    eTouch_2d_gesture_event_on_touch_move_3 = 6, /* move left. */
    eTouch_2d_gesture_event_on_touch_move_4 = 7, /* move right. */

    /* to support continuous clicks. */
    eTouch_2d_gesture_event_no_touch_interval, /* short leave interval. */
    eTouch_2d_gesture_event_on_touch_just_2nd_click,  /* 2nd click. */
    eTouch_2d_gesture_event_on_touch_just_3rd_click,  /* 3rd click */

    eTouch_2d_gesture_event_on_touch_no_process,  /* other. */
} touch_2d_gesture_event_t;
```

3 Hardware overview

The Touch 2D KL16Z board is using 5+5 channels to build the touch 2D sensing system. There are also 11 LEDs on the board to show the status of application's running state. The MCU is MKL16Z64VFT4, with Arm® Cortex®-M0+ core @ 48MHz core clock, 64 KB FLASH, 8KB RAM, QFN48 package and TSI module, which is used to support touch sensing application. The board is as shown in Figure 4.

Figure 4. Touch 2D KL16Z board

Table 1. Pin connections

Touch 2D kl16z board	MKL16Z pin	Pin mux
R0	PTA1	ALT0, TSI0_7
R1	PTA2	ALT0, TSI0_6
R2	PTB0	ALT0, TSI0_0
R3	PTB1	ALT0, TSI0_3
R4	PTB2	ALT0, TSI0_2
C0	PTB3	ALT0, TSI0_8
C1	PTB16	ALT0, TSI0_9
C2	PTB17	ALT0, TSI0_10
C3	PTC0	ALT0, TSI0_13
C4	PTC1	ALT0, TSI0_14
Shield	PTC2	ALT0, TSI0_15
UART0_RX	PTD6	ALT3, UART0_RX
UART0_TX	PTD7	ALT3, UART0_TX
I2C0_SDA	PTE18	ALT4, I2C0_SDA
I2C0_SCL	PTE19	ALT4, I2C0_SCL

4 2D gesture recognition algorithm

The 2D gesture recognition algorithm starts from detecting the movement of touch position first. To detect the movement, a FIFO queue of the calculated position is used to keep the most recent two positions. Once a new position is pushed into the the queue, the oldest one is kicked out, so there is a previous position and a current position. We can make a difference from current position to previous one to check the movement. It was easy to do the judgement when the movement is in 1-dimension, as there are only two directions. Here for the 2D situation, we need to open a little mind and make extension.

When we do the difference between position in 2D position system, the **X** value minus **X** value, **Y** value minus **Y** value. Then comparison between the delta **X** and delta **Y** to find the one with bigger absolute value. The direction of the bigger one represents the position's movement direction in the available range of up/down/left/right. If the bigger one is still less than a given threshold, the behavior would be marked as staying the same position, with no movement.

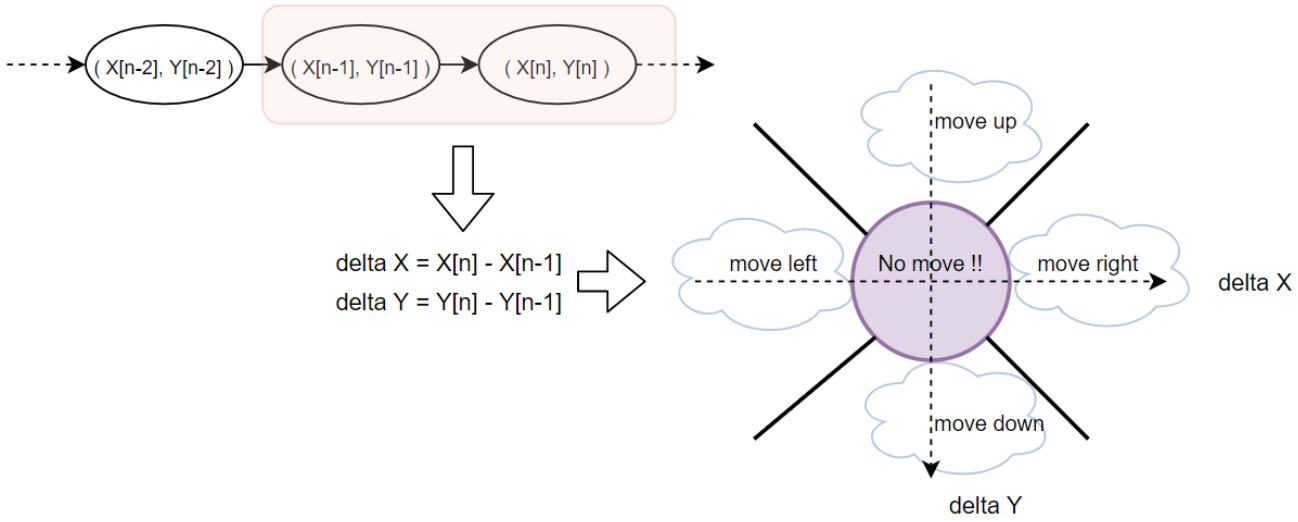

Figure 5. Algorithm to judge position movement in 2D coordination system

Till now, we have three features for a sample of position:

- On touch or no touch
- Movement direction if on touch
- Timing relationship of successive samples

These features are used to judge the events on the touch panel, including:

- Move up/down/left/right, or stay the same place (no move).
- Touch 1st/2nd/3rd click, and the interval between clicks is in a given short period.
- No touch keep and no touch interval are the state for no touch conditions.

Each event is triggered from the previous one under indicated conditions. The event transmit machine between are as shwon in Figure 6.

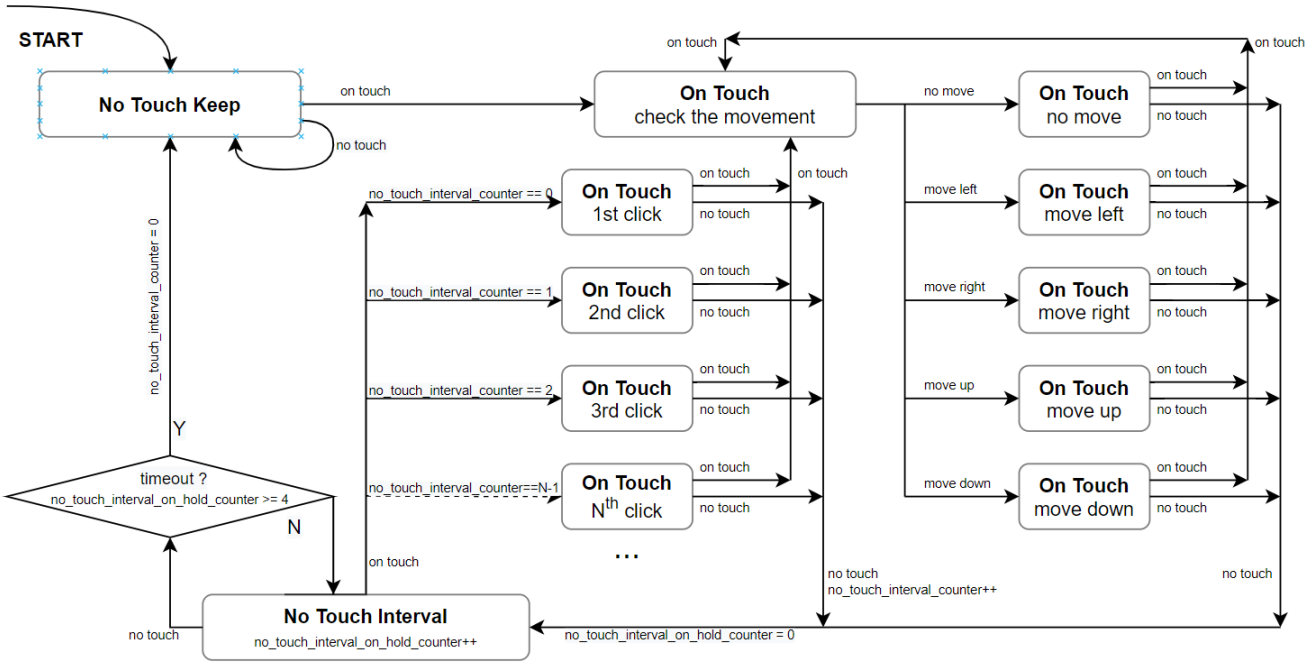

Figure 6. Event transmit machine for 2D touch gesture recognition

In the application project, the sampling rate for a group of all channels is 25 Hz, which is controlled by a LTPMR0 module. After each sensing period, the `touch_2d_calc_position()` and `touch_2d_gesture_process()` functions are called periodically for each group of new sensing values. The `touch_2d_calc_position()` function gets the information whether the panel is on touch and the position. The `touch_2d_gesture_process()` function fills these information into the event transmit machine and outputs the judgement of event for gestures.

5 Customizing

There are some characters can be manually customized per user's experience and requirement.

5.1 Timeout period of continuous clicks interval

The time period of continuous click interval is about 200 ms now. This is controlled by the setting for LPTMR0's interrupt period, 40 ms, and the `no_touch_interval_on_holder_counter`. Every 40 ms, the `touch_2d_gesture_process()` function is called again, and the `no_touch_interval_on_holder_counter` variable is increased once when current state is for the event of **No Touch Interval**. It is used to compare with the number **4** (0 is the first one, 4 means the 5th period, for 5 × 40 ms = 200 ms), as shown in , to check whether the interval is timeout. If coming **on touch** condition caused by the click arrives before timeout, the new click is considered in a successive/continuous click sequence, and marked as a **Nth click event**. If no **on touch** condition comes before the timeout (no touch or later touch), the **No Touch Interval** state (causing the **No Touch Interval** event) is transmitted to the **No Touch Keep** state (causing the **No Touch Keep** event), which means the current state is the **No Touch** stably, and no continuous click detection is active for following samples.

5.2 Enable the detection of event for more continuous clicks

In the current example project, only the first three clicks in continuous click sequence are activated. However, more clicks can be recognized within the algorithm as well. It is controlled by the available values for the `no_touch_interval_counter` variable. This variable records the count of intervals in a continuous click sequence, and increases when a new click coming. Once the continuous click event happens, the `touch_2d_gesture_process()` function checks this variable's value and reports the related continuous click events, including for the first three clicks. The detection for more clicks is still available, as the **no touch**

interval_counter keeps increasing for the new click and does not reset until the continuous click sequence is break (the interval is timeout).

For example, when an 8th click is required to be picked out, only the following addition code is needed:

```
touch_2d_gesture_event_t touch_2d_gesture_process(int32_t x_pos, int32_t y_pos, bool on_touch)
{
    if (on_touch)
    {
        if (     (touch_2d_gesture_event_pre == eTouch_2d_gesture_event_no_touch_keep)
              || (touch_2d_gesture_event_pre == eTouch_2d_gesture_event_no_touch_interval)
            )
        {
            if (touch_2d_gesture_no_touch_interval_counter == 0u)
            {
                touch_2d_gesture_event = eTouch_2d_gesture_event_on_touch_just;
            }
            else if (touch_2d_gesture_no_touch_interval_counter == 1u)
            {
                touch_2d_gesture_event = eTouch_2d_gesture_event_on_touch_just_2nd_click;
            }
            else if (touch_2d_gesture_no_touch_interval_counter == 2u)
            {
                touch_2d_gesture_event = eTouch_2d_gesture_event_on_touch_just_3rd_click;
            }
            else if (touch_2d_gesture_no_touch_interval_counter == 7u)
            {
                touch_2d_gesture_event = eTouch_2d_gesture_event_on_touch_just_8th_click;
            }
            else
            {
                touch_2d_gesture_event = eTouch_2d_gesture_event_on_touch_no_process;
            }
            ...
```

5.3 Support "all cover" gesture

The current algorithm supports the gestures based on movement of one touch point. The **all cover** gesture is not just using one position. It is considered as a special case before calculating the position, as its judgement is not based on the current position algorithm.

We use the sum of different offsets for all channels in the touch panel as the critical character to check the **all cover** event. Once a channel is touched, its sensing value causes a offset against its baseline of no touch. This offset is the difference for the channel's touch event. The closer finger is near the touch pad, the bigger the difference value would be. In the self capacitive touch sensing method, which is used in KL16Z's touch module now, these differences for each channels are independent. So when multiple channels are touched together in the same time, they would have big difference together. Then the sum of these differences can be increase significantly than the one in no touch state. So, it can be used to judge multiple channels are touched at the same time, as more touch area would cover more channels.

Actually, its implementation is in the `touch_2d_calc_position()` function, and returns the code **3** when the all cover event is detected. For the details about the coding, see AN13026SW.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10/2020

Document identifier: AN12306

arm