AN12823

Enable High-Resolution Touch Slider Using CapTouch on LPC804

Rev. 0 — 7 April 2020

Application Note

Contents

1 Introduction

Touch slider is created by assembling multiple standalone touch pads together. When a finger touches any of the pads, the MCU can detect the touch event and then report the sensing position according to the hardware layout. Actually, in this basic usage case, the touch slider is considered as normal keyboard. The sensing position is just the key value of the touch pad, while the sensing resolution is limited by the total count of touch pads. In this paper, a simple algorithm is applied to the keyboard assembled with touch pads, to achieve a much higher sensing resolution. Even by using the same hardware layout of touch keyboard, the new method can sense more position, including the interval space between the pads. The low-cost MCU, LPC804 with mutual CapTouch module is used as the reference hardware platform. The algorithm is also available for other touch method, like using the self CapTouch module within KE1xZ MCU.

2 Hardware

A reference hardware layout is designed to show the usage of touch slider. The new designed "LPC804 Touch Slider EVK" board is using the LPC804M101JDH24 as the core MCU. It is built with five touch pads together, assembling a touch slider. 7 LEDs are integrated into the board to show the board's running status and the sensing position. A USB-HID-UART SOC (CH9329) is optional used on the board to communicate with the GUI panel PC through USB bus when necessary. The micro-USB socket is also used to fetch the 5 V power supply from external USB bus and convert it to 3.3 V for the whole board.

First of all, the single touchpad should be created. As in the reference hardware board, the touchpad's layout is designed as in Figure 1.

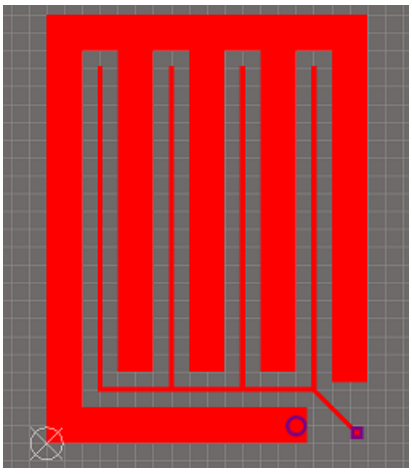

Figure 1. Touchpad layout for mutual CapTouch method

In Figure 1, the length of each grid edge is 0.5 mm. The width of X pad is 1 mm. The width of Y pad is 0.14 mm. The distance between X and Y in touchpad is 0.43 mm.

TIPS: When in the place the X wire and Y wire is closed or cross in different layer, it would become the "sensing area". In the touchpad, we want to create the "sensing area" as more as possible, to enhance the sensitivity. But in the connection routine,

we want to create the "sensing area" as less as possible, to reduce the unexpected sensing. Of course, we would like to reduce the unexpected "sensing area" in actual layout, but it is hard to kick them all. So, as a compromise, we can try to move the unexpected sensing area far away from the touchpad, so that we can reduce the mis-match as much as possible.

When creating the touch slider, we need to use the touchpad with the same size, and arrange them in the same internal space. The design sketch of layout for touch slider assembled by touchpads is showed in Figure 2.

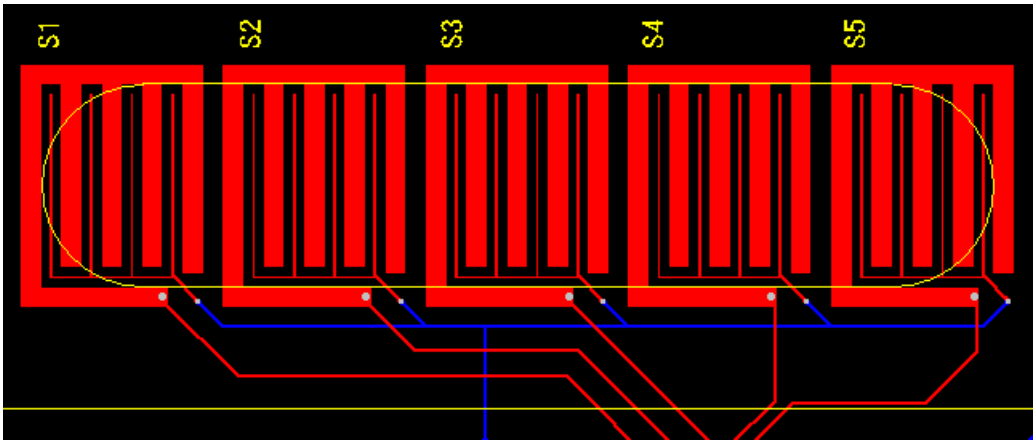

Figure 2. Touch slider layout for mutual CapTouch method

And finally, we got the real hardware board, shown in Figure 3.

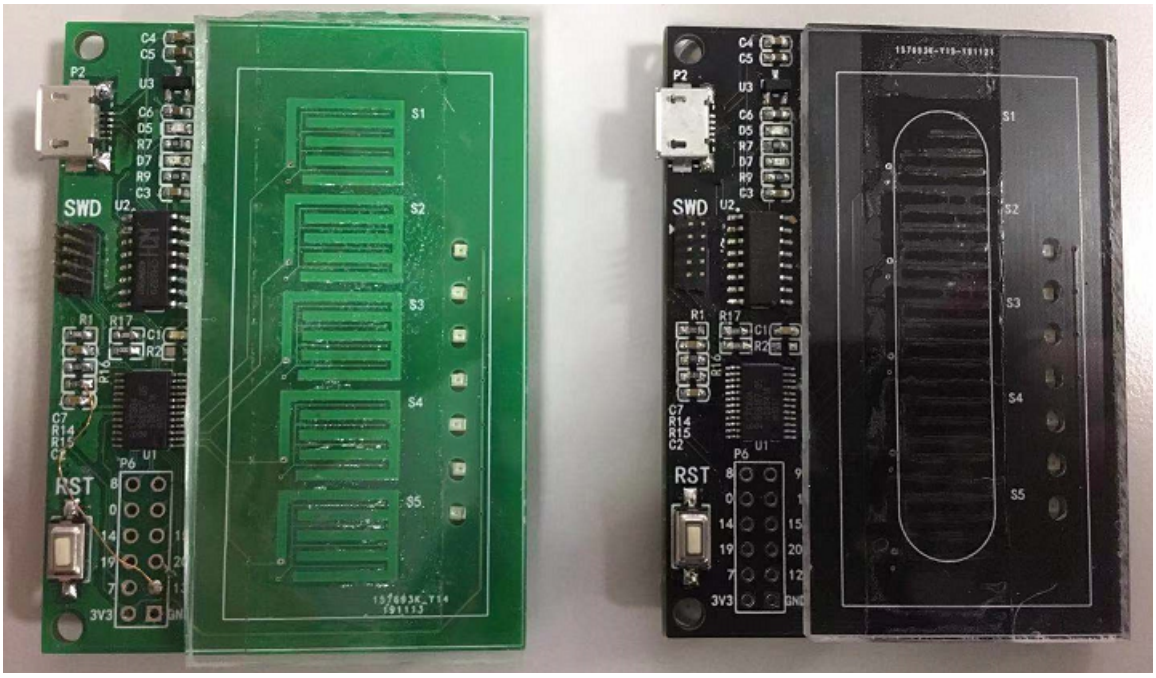

Figure 3. LPC804 TouchSlider EVK board

There is also an acrylic board covering the PCB, with about 3 mm thickness, as shown in Figure 4.

Figure 4. Use acrylic board with 3 mm thickness

3 Software

3.1 Basic function of fetching sensing value from touchpads

To make the touch sensing process running automatically in the background of the main application (the main application might run other algorithm with previous buffered sensing value during the current sensing process), a hardware timer trigged auto-sense framework is designed in software.

First, the MRT and CapTouch module should be initialized, enabling there interrupts. Then, all the work of starting the sensing and gathering the data would be done in their interrupt service routine. The touch sensing work would be done silently within CapTouch module.

```
volatile int16_t touch_val_raw_index;
const uint16_t touch_channel_code[TOUCH_USER_CHANNEL_NUM] =
{
    0x1, 0x2, 0x4, 0x8, 0x10
};

/* ISR to trigger the conversion periodically.
 */
void MRT0_IRQHandler(void)
{
    /* channel sensing. */
    if (0u != (MRT_GetStatusFlags(MRT0, kMRT_Channel_0) & kMRT_TimerInterruptFlag) )
    {
        MRT_ClearStatusFlags(MRT0, kMRT_Channel_0, kMRT_TimerInterruptFlag);
```

```
        /* periodical task. */
        touch_hal_init_pins();/* setup pins for captouch. */
        touch_hal_start_conv(touch_channel_code[touch_val_raw_index]);
    }
}
```

As seen in the source code, the touch sensing for each channel is trigged by the MRT interrupt periodically one by one, while the internal of MRT interrupts can be programable in the application. It means, user can configure the sample rate of touch by programming the MRT's interrupt interval period. For each MRT's ISR running:

- touch_hal_init_pins(). Enable the pin function of CapTouch module. These pins are available during the sensing process, and would be grounded (configured as GPIO and output low voltage level) when the CapTouch module is idle during the sample interval.
- touch_hal_start_conv(). Enable the CapTouch converter with indicated channel number coded in constant table touch_channel_code[] one by one. The index would be updated later when the conversion is done in CapTouch ISR function.

Once any touch conversion is triggered. The CapTouch module would run the conversion silently with no CPU interaction. Then, it raises the CapTouch interrupt when the conversion is done.

```
volatile int16_t touch_val_raw[TOUCH_USER_CHANNEL_NUM];

/* ISR for capt conversion done.
 * read the sensing value on conversion done.
 */
__weak void touch_one_channel_scan_done_hook(uint8_t channel_index)
{
}
__weak void touch_all_channels_scan_done_hook(void)
{
}
void CMP_CAPT_IRQHandler(void)
{
    touch_val_raw[touch_val_raw_index] = touch_hal_read_conv();
    /* reset the pins after conversion done. */
    touch_hal_reset_pins();

    if (touch_val_raw_index == (TOUCH_USER_CHANNEL_NUM-1u))
    {
        touch_pause(); /* a round is done. */
        touch_scan_round_done = true;
        touch_val_raw_index = 0;
        touch_all_channels_scan_done_hook();
    }
    else
    {
        touch_one_channel_scan_done_hook(touch_val_raw_index);
        touch_val_raw_index++;
    }
}
```

In the CapTouch ISR function, the framework would collect the sensing value into the buffer touch_val_raw[], then reset the pins used by CapTouch, to make sure the energy on the touchpad would be released cleanly. Also, two callback functions are defined here with "weak" attribute, which can be re-defined optionally by user in application code to cover the default implementation:

- touch_one_channel_scan_done_hook() would be called when a single conversion is done.
- touch_all_channels_scan_done_hook() would be called when a sequence of conversions is done.

A touch_wait_date_ready() function is to sync the sequence scan done event, and copy the sensing values to user indicated memory.

```
void touch_wait_data_ready(int16_t *output)
 {
     while (!touch_scan_round_done)
     {}
     for (int i = 0u; i < TOUCH_USER_CHANNEL_NUM; i++)
     {
         output[i] = touch_val_raw[i];
     }
}
```

Then in application level, user code would wait for each conversion round, and copy the sensing array from internal buffer to user buffer.

```
int16_t app_touch_slider_val_raw[TOUCH_USER_CHANNEL_NUM] = {0};

int main(void)
{
    /* initialize the board ... */

    touch_init(); /* initialize all used pins. */
    touch_start(); /* start the scan. */

    while (1)
    {
        touch_wait_data_ready(app_touch_slider_val_raw);
        touch_start(); /* start new scan for next process. */

        /* TODO: process the previous round of sensing data. */
    }
}
```

For the full version's code, please refer to the attachment software package.

FreeMASTER tool is used to show the sensing values in the variable array of app_touch_slider_val_raw[] in run-time.

Figure 5. FreeMASTER waveforms for each sensing channels

In Figure 5, once any touch pad is touched, almost all the sensing values would fall down, but the responding sensing value would fall most. Also, it can be seen that, the closer to the touch position, the falling value is bigger.

Now, as we consider the sensing values are not changing when the MCU scans the whole sequence of touchpads, then we have the sample time slices. The sensing values in each slice can be used to make the judgement of touch position.

3.2 Enable high resolution slider using analog sensing values

When the slider is touched, with the basic judging method, in the ideal condition, there would one most sensitive channel can be used to identify the touched position. However, this way cannot describe the position more precise, as there would be only 5 available touched positions in the touch slider (assembled with 5 touchpad in this demo case). Actually, we would like to identify the position between the pads as well. Then we have a new algorithm.

As known that, when the finger is closer to the touch pad, the sensing value offset against the no touch state would be bigger. It means that the shorter sensing distance would make bigger available sensing value. Once the finger is touching the slider, all the 5 touched, each channel has its own sensing value. Then we would not use only the biggest one to describe the position, but use all the sensing values as the weights to describe which touchpads the touch point would be like. The sensing value of pads near the touch point would be bigger than the ones far from it.

To get the offset values, which are also the available sensing values caused by the touch behavior, a calibration function is created to get the stable baseline for each touch sensing channel. Before the calculation, the user application would sample several time slices of sensing values. The touchcalib() function would update the internal touch_calib_baseline[] according to the recent samples. Once the values are stable enough, the application would use these baselines to apply the following sensing values and get the available sensing values for calculation.

```
volatile int16_t touch_calib_baseline[TOUCH_USER_CHANNEL_NUM] = {0};

uint16_t touch_calib(int16_t * input)
{
    int16_t err;
    uint16_t err_max = 0u;

    for (int i = 0; i<TOUCH_USER_CHANNEL_NUM; i++)
    {
        err = input[i] - touch_calib_baseline[i];
        touch_calib_baseline[i] += (err / 4); /* update the baseline. */
        if (err >= err_max)
```

```
        {
            err_max = err;
        }
    }
    return err_max;
}
```

Then, the accumulated sum value of each touch pad's position value multiplied with their sensing weight value, can be used to identify the position of touch point. However, the common sensitivity would affect the accumulated sum value. When the finger is far from the touchpads, the available sensing values would be small, and the sum is small. When the finger is close to the touchpads, the sum would be big. So the sum value here is still not enough to describe the position.

Usually, the weight-evaluation method goes with the normalization step. Then the normalization is applied to the sum value. Then, a result value of descripting the position distribute in the position system is available.

```
volatile int16_t touch_val[TOUCH_USER_CHANNEL_NUM]; /* unified value. */

int32_t touch_get_pos(int16_t * val_raw)
{
    int32_t sum = 0, sum2 = 0;

    for (int i = 1u; i<TOUCH_USER_CHANNEL_NUM; i++)
    {
        touch_val[i] = touch_calib_baseline[i] - val_raw[i];
        sum += touch_val[i] * i;
        sum2 += touch_val[i];
    }
    return sum / sum2;
}
```

In the example application code, there are also some skills to defense the noise and reduce the exception:

- Ignore the condition with small available sensing values to check the available touch behavior.
- Apply the second level's offset to normalize the sensing value again.
- Use the nearest 3 pads of 5 to calculate the position.
- Scale the range of output position value.

These processes help to increase the stability and smoothness of output position. The final implementation of calculation is:

```
touch_get_pos_err_t touch_get_pos(int16_t * val_raw, uint32_t pos_range, int32_t * pos)
{
    touch_get_pos_err_t err = eTouch_GetPosErr_UnKnown;
    uint16_t m_touch_val_max_idx = 0, m_touch_val_min_idx = 0;
    int32_t m_touch_calc_val[2] = {0};
    int16_t len = TOUCH_USER_CHANNEL_NUM;
    uint16_t m_touch_val_calc_idx_start = 0;

    /* apply the calib offset, and find the max and min sensing channel. */
    touch_val[0] = touch_calib_baseline[0] - val_raw[0];
    for (int i = 1u; i < len; i++)
    {
        touch_val[i] = touch_calib_baseline[i] - val_raw[i];
        if (touch_val[i] > touch_val[m_touch_val_max_idx])
        {
            m_touch_val_max_idx = i;
        }
```

```
        if (touch_val[i] < touch_val[m_touch_val_min_idx])
        {
            m_touch_val_min_idx = i;
        }
    }

    /* iguore the small sensing case. */
    if (abs(touch_val[m_touch_val_max_idx] - touch_val[m_touch_val_min_idx]) <
TOUCH_YES_TOUCH_SHRESHOLD_LOW)
    {
        err = eTouch_GetPosErr_NoTouch;
        return err;
    }

/* only use the nearest 3 channels. */
#define TOUCH_POS_CALC_CHANNEL_NUM 3
    if (m_touch_val_max_idx == 0)
    {
        m_touch_val_calc_idx_start = 0;
    }
    else if ( m_touch_val_max_idx >= (len-1) )
    {
        m_touch_val_calc_idx_start = (len-TOUCH_POS_CALC_CHANNEL_NUM);
    }
    else
    {
       m_touch_val_calc_idx_start = m_touch_val_max_idx-(TOUCH_POS_CALC_CHANNEL_NUM/2);
    }

    /* rebase the sensing values. */
    m_touch_val_min_idx = m_touch_val_calc_idx_start;
    for (int i = m_touch_val_calc_idx_start+1;
         i < m_touch_val_calc_idx_start+TOUCH_POS_CALC_CHANNEL_NUM;
         i++)
    {
        if (touch_val[i] < touch_val[m_touch_val_min_idx])
        {
            m_touch_val_min_idx = i;
        }
    }

    /* normalization. */
    m_touch_calc_val[0] = 0;
    m_touch_calc_val[1] = 0;
    for (int i = m_touch_val_calc_idx_start;
         i < m_touch_val_calc_idx_start+TOUCH_POS_CALC_CHANNEL_NUM;
         i++)
    {
        touch_val[i] -= touch_val[m_touch_val_min_idx];
        m_touch_calc_val[0] += touch_val[i] *i;
        m_touch_calc_val[1] += touch_val[i];
    }
    *pos = m_touch_calc_val[0] * (pos_range / (len-1)) / m_touch_calc_val[1];

    err = eTouch_GetPosErr_YesTouch;
    return err;
}
```

The FreeMASTER tool is also used to show the waveform of output position.

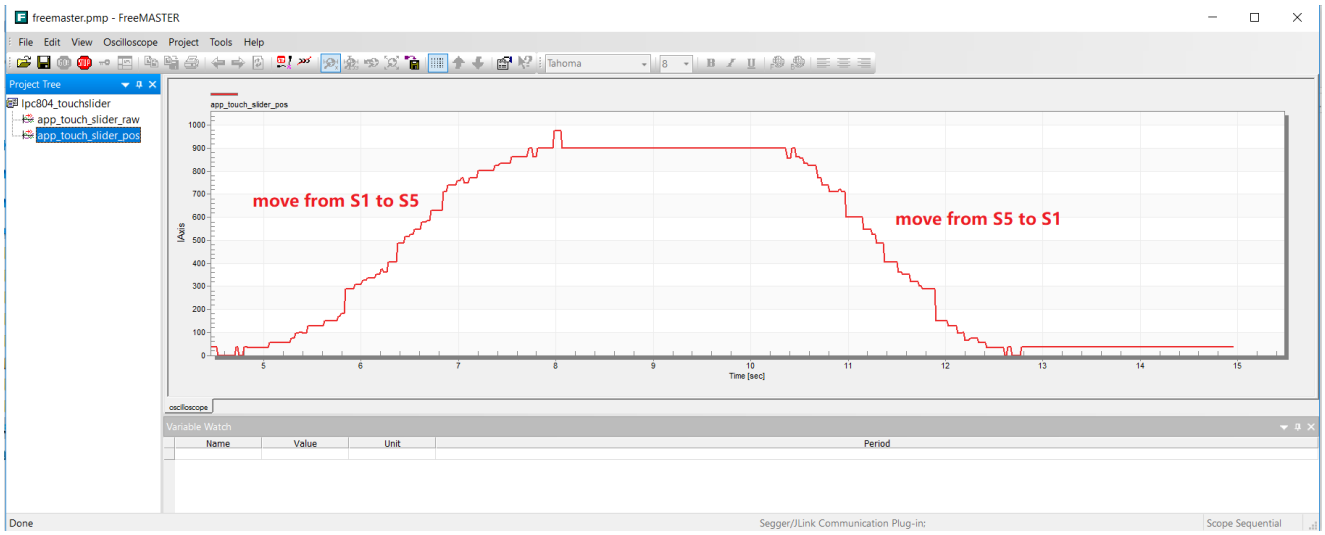

Figure 6. FreeMASTER waveform for sensing position in touch slider

4 Conclusion

The method used to detect the touch key only can provide very few available position. A new calculation method with weights and normalization described in this paper helps to get more precious position based on the whole touch slider (assembled with multiple channels). Actually, as the touch position is not depends on the one value, but due to the more values, it belongs to the way of using the redundancy information to trade for detail description.

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: 7 April 2020

Document identifier: AN12823

arm