AN12933

Enable 2D Touch Sensing System Based on KE15Z MCU

Application Note

Contents

1 Introduction

The Touch Sensing Input (TSI) module on Kinetis E series MCU provides touch sensing detection on capacitive touch sensors. The external capacitive touch sensor is typically formed on Print Circuit Board (PCB), and the sensor electrodes are connected to the TSI input channels through the I/O pins in the MCU chips.

Usually, one touch channel is connected with one electrode, so that it can detect the on touch and no touch condition, which is called "touch button". Multiple touch buttons could be placed together in a queue to create a touch slider. The touch slider can be used to sense the touched position in the 1-dimension scale and the movement on it. Additionally, two sliders in orthogonal coordinate system can be used to build a 2-dimension (2D) touch sensing system.

This application note describes the hardware and software design of implementing a 2D touch sensing system based on the KE15Z64 MCU.

2 Hardware

2.1 X-DC-GESTURE board

During the implementation of 2D touch sensing system, the layout of touchpad is the most important thing. Figure 1 displays a sample touchpad layout for 2D touch sensing system.

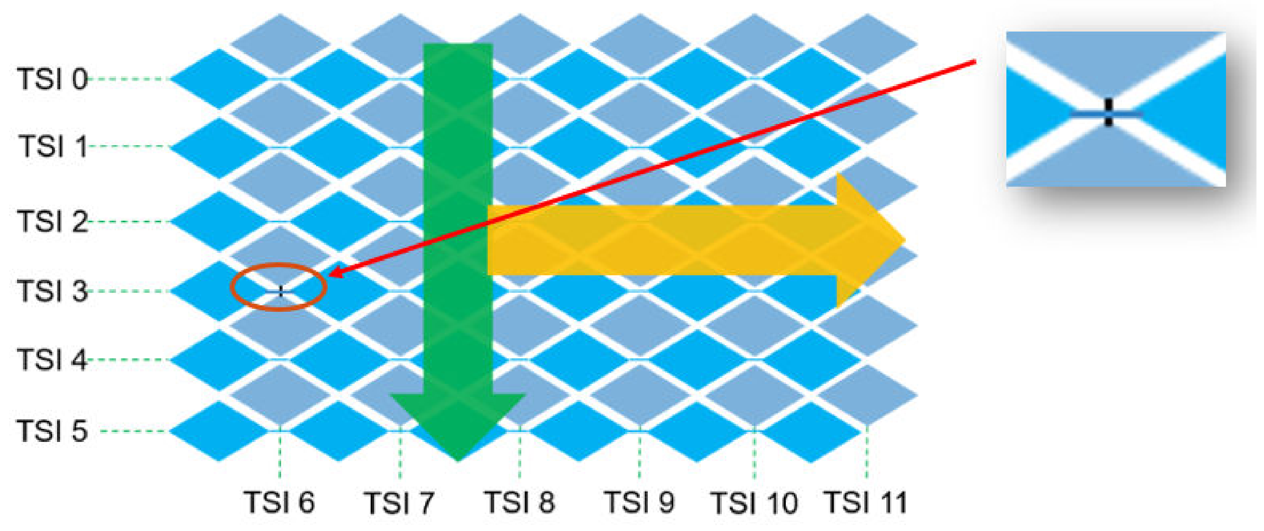


Figure 1. Sample touchpad layout for 2D touch sensing system

Each horizontal line with the diamond electrodes connected together is created as a big electrode for one TSI channel. All the horizontal line electrodes are placed in a vertical queue to create a 1-dimension touch slider. Each vertical line with the diamond electrodes connected together is created as a big electrode for one TSI channel as well, thereby creating a 1-dimension touch slider in the horizontal direction.

The vertical slider and horizontal slider co-exist in a whole touch panel. Once any point touches on this panel, both vertical slider and horizontal slider create their own position values for the touch position. These values can be put together as the coordinates of the touched point in the 2D touch panel.

According to the diagram in Figure 1, a hardware board X-DC-GESTURE is created (see Figure 2).

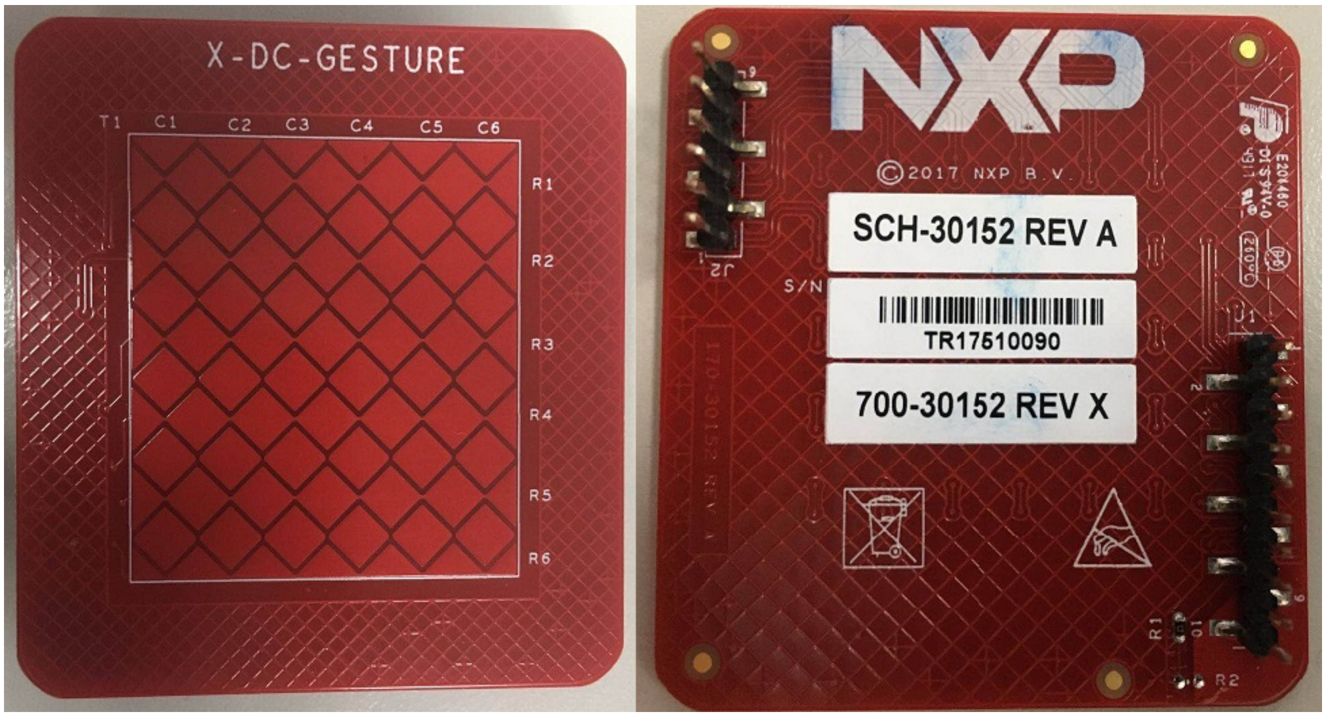


Figure 2. PCB board of X-DC-GESTURE

On this board, the surround grids of the touch panel can be connected to either the ground or another TSI channel, which can be used to suppress the environment noise actively. The resistors R1 and R2 are the switchers to the ground and a TSI channel. In the software package provided with the application note, R1 is disconnected and R2 is connected, and the surround grids are used as a TSI channel's sensing pad. It detects the boundary of touch panel, within a software method (self-offset and filters), to suppress the environment noise as well.

2.2 X-FRDM-KE15Z-QFN board

The X-FRDM-KE15Z-QFN board is based on the MCU of MKE15Z64VFP, with ARM Cortex-M0+ core with 48 MHz max core clock and the internal memory of 64 KB Flash, 8 KB SRAM. The KE1xZ64 MCU includes a TSI module supporting up to 6x6 mutual capacitive touch matrix and 25 self-capacitive touch channels. The X-FRDM-KE15Z-QFN board includes a group Arduino-like sockets to adapt to the X-DC-GESTURE board, see Figure 3.

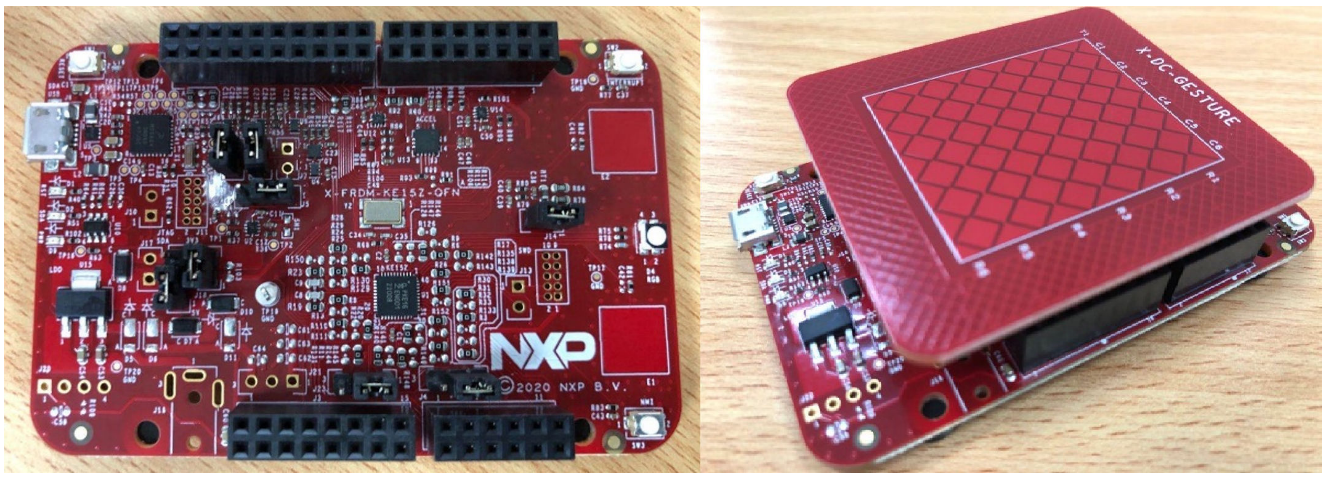


Figure 3. X-FRDM-KE15Z-QFN board

Once both boards, X-DC-GESTURE and X-FRDM-KE15Z-QFN, are assembled, the following connections are made between the TSI module on KE15Z64 and the electrodes on touch panel, as listed in Table 1.

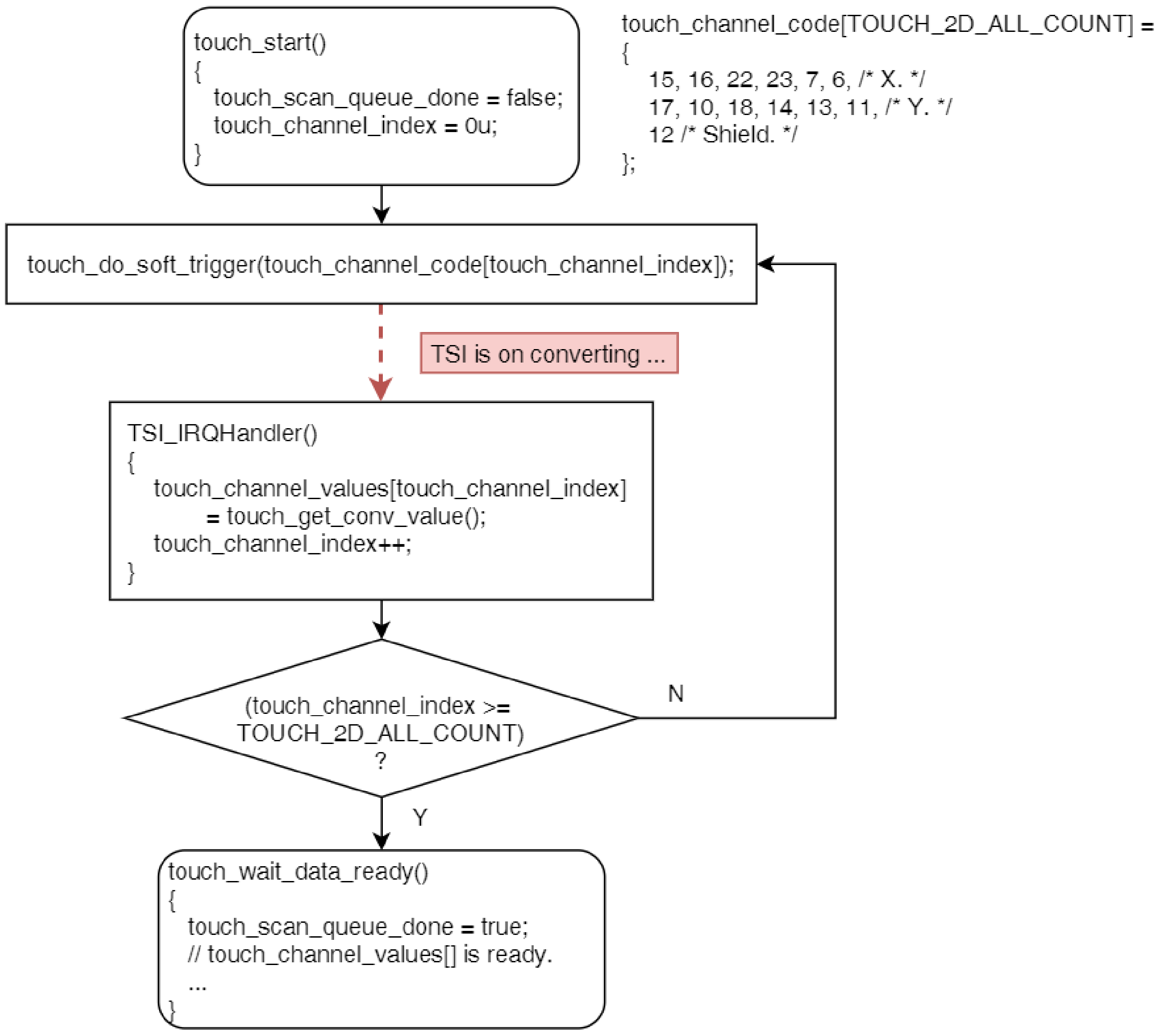
Table 1. Connections between TSI module on KE15Z64 and electrodes

X-DC-GESTURE pins	X-FRDM-KE15Z-QFN pins	KE15Z64 pins	Pin Mux (ALT0)
J1-2, TSI_R0	J2-3, PTC6/GES_R0	PTC6	TSI0_CH15, ALT0
J1-3, TSI_R1	J2-5, PTC7/GES_R1	PTC7	TSI0_CH16, ALT0
J1-4, TSI_R2	J2-7, PTC0/GES_R2	PTC0	ADC0*SE8/TSI0*CH22
J1-5, TSI_R3	J2-9, PTC1/GES_R3	PTC1	ADC0*SE9/TSI0*CH23
J1-6, TSI_R4	J2-11, PTD6/TOUCH*MRX0/GES*R4	PTD6	TSI0_CH7
J1-7, TSI_R5	J2-13, PTD5/TOUCH*MRX1/GES*R5	PTD5	TSI0_CH6
J1-8, TSI_SHIELD	J2-15, PTC5/GES_SHIELD, R117 on	PTC5	TSI0_CH12
J1-9, GND	J2-17, GND, R115 on	GND	
J1-10, GND	J2-19, GND	GND	
J2-1, TSI_C5	J4-1, PTE8/TOUCH*S*0/GES_C5	PTE8	ACMP0*IN3/TSI0*CH11
J2-2, TSI_C4	J4-3, PTE0/TOUCH*S*1/GES_C4	PTE0	TSI0_CH13
J2-3, TSI_C3	J4-5, PTE1/TOUCH*S*2/GES_C3	PTE1	TSI0_CH14
J2-4, TSI_C2	J4-7, PTA1/TOUCH*S*3/GES_C2	PTA1	ADC0*SE1/ACMP0*IN1/TSI0_CH18
J2-5, TSI_C1	J4-9, PTD7/TOUCH*S*4/GES_C1	PTD7	TSI0_CH10
J2-6, TSI_C0	J4-11, PTA0/TOUCH*S*5/GES_C0	PTA0	ADC0*SE0/ACMP0*IN0/TSI0_CH17

3 Software

3.1 Fetch the capacitive touch sensing values

You need to enable a "background" service to trigger the TSI's conversion and read the sensing values in a queue automatically, so that the CPU core can perform calculation concurrently with the TSI's conversion ongoing at the same time.


Code in the figure:

```
touch_start()
{
  touch_scan_queue_done = false;
  touch_channel_index = 0u;
}
```

```
touch_channel_code[TOUCH_2D_ALL_COUNT] =
{
   15, 16, 22, 23, 7, 6, /* X. */
   17, 10, 18, 14, 13, 11, /* Y. */
   12 /* Shield. */
};
```

```
touch_do_soft_trigger(touch_channel_code[touch_channel_index]);
```

```
TSI_IRQHandler()
{
   touch_channel_values[touch_channel_index]
          = touch_get_conv_value();
   touch_channel_index++;
}
```

```
(touch_channel_index >=
TOUCH_2D_ALL_COUNT)
?
```

```
touch_wait_data_ready()
{
  touch_scan_queue_done = true;
  // touch_channel_values[] is ready.
  ...
}
```

Figure 4. Diagram of getting conversion values by hardware automatically

In the application level code, the previous conversion values are buffered in the *app_touch_channel_values[]* array and start the conversion for next round. While the conversions are ongoing, the calculation using *app_touch_channel_values[]* are working simultaneously. The two threads would be working together to make full use of chip's working load.

```
while (1)
{
    touch_wait_data_ready(app_touch_channel_values);
    touch_start();
    /* todo: using app_touch_channel_values for calculation.*/
    ...
}
```

A FreeMASTER project is created to show the waveforms of runtime values for the variables. The original sensing values of *app_touch_channel_values[]* are shown in figure below.

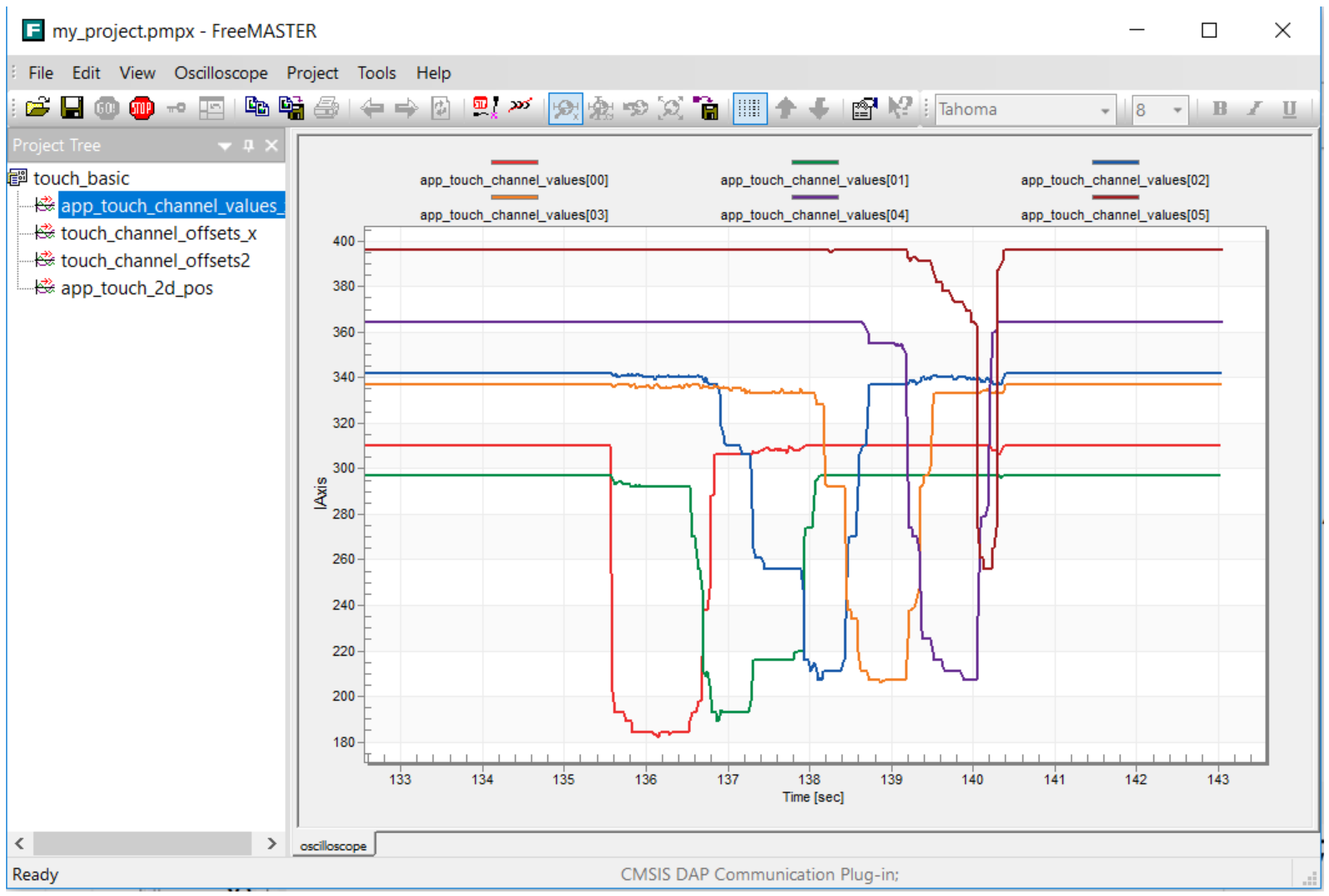

my_project.pmpx - FreeMASTER
File Edit View Oscilloscope Project Tools Help
Tahoma
8
Project Tree
touch_basic
app_touch_channel_values_
touch_channel_offsets_x
touch_channel_offsets2
app_touch_2d_pos
app_touch_channel_values[00]
app_touch_channel_values[01]
app_touch_channel_values[02]
app_touch_channel_values[03]
app_touch_channel_values[04]
app_touch_channel_values[05]
lAxis
Time [sec]
oscilloscope
Ready
CMSIS DAP Communication Plug-in;

Figure 5. Runtime waveforms of *app_touch_channel_values[]*

From the waveforms, it can be seen that once the electrode is touched or closed, the sensing value would drop down. The closer the sending value is between the finger and electrode, the more it drops down.

3.2 Calibration

You can use various methods of calibration for pre-processing to the original sensing values. This application note shows how to cut off the offsets of original baselines for each channels themselves. The common mode changes would be reduced in the position calculation algorithm later. So the application ignores the first 10 rounds of samples after the reset, uses the next round of samples as the calibration baseline, and considers it as the stable idle sensing value. So during the startup time, it is advised to not touch the panel and make it clean, so that the application can learn the right values.

In the application, the on-board red LED is used to indicate the calibration is done. It is switched on at the beginning of the program. After the calibration is complete, the red LED switches off, and the touch panel is ready to detect the touching position.

During the calibration, the application learns the baselines and keeps them in the array of *touch_channel_baselines[]*.

3.3 Position calculation algorithm

The process of position calculation is to find out the most significant changed channels, convert the changes to weights, and assemble them to generate precise position in the whole position range. In the application code, all the processes are implemented in the *touch_2d_calc_position()* function. The steps of the position calculation process are as follows:

1. Cut off the baselines of each channel, and get the offset caused by the changes from touch event.

```
/* cut off the baseline for each channel. */
for (uint32_t i = 0u; i < TOUCH_2D_ALL_COUNT; i++)
{
    touch_channel_offsets[i] = touch_channel_baselines[i] - inputs[i];
}
```

The waveforms of *touch_channel_offsets* are created, as shown in figure below.

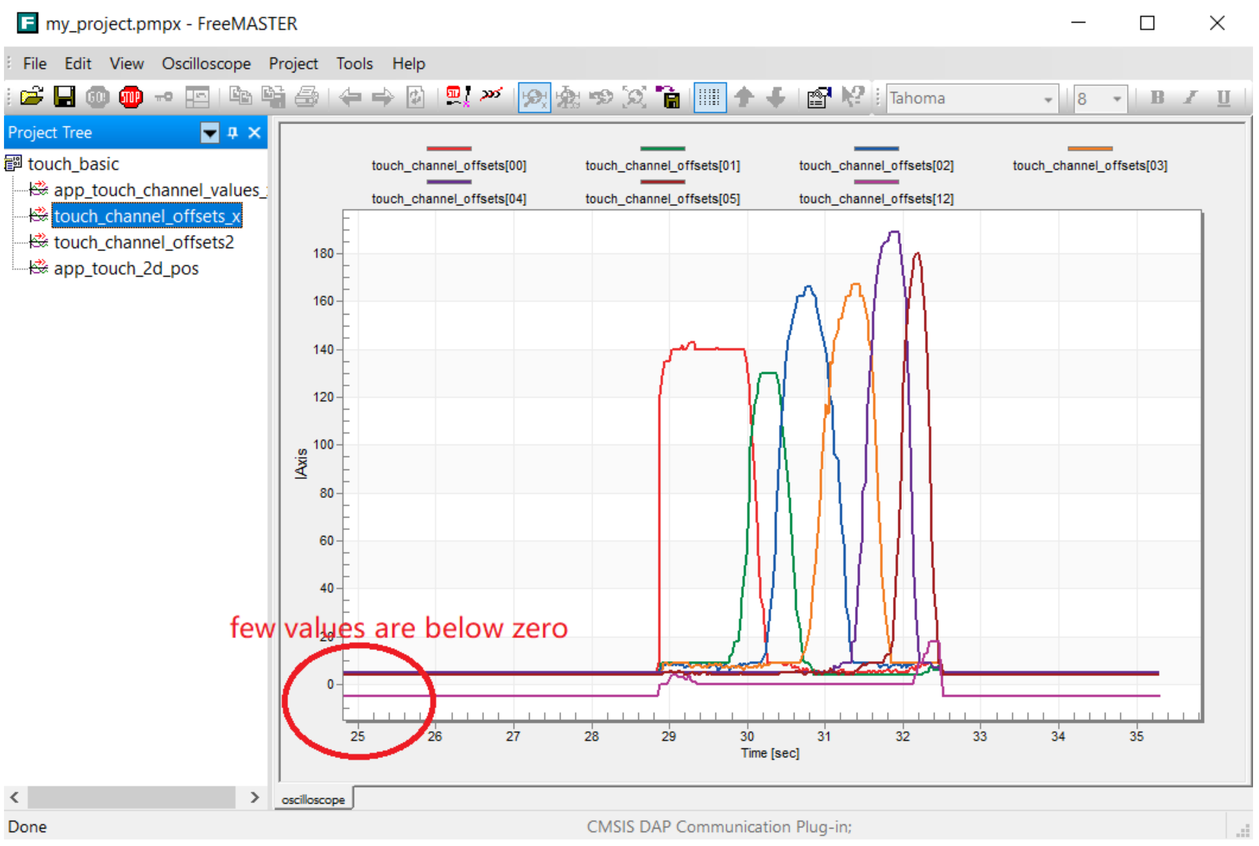

my_project.pmpx - FreeMASTER
File
Edit
View
Oscilloscope
Project
Tools
Help
Tahoma
8
Project Tree
touch_basic
app_touch_channel_values_
touch_channel_offsets_x
touch_channel_offsets2
app_touch_2d_pos
touch_channel_offsets[00]
touch_channel_offsets[01]
touch_channel_offsets[02]
touch_channel_offsets[03]
touch_channel_offsets[04]
touch_channel_offsets[05]
touch_channel_offsets[12]
IAxis
Time [sec]
few values are below zero
oscilloscope
Done
CMSIS DAP Communication Plug-in;

Figure 6. Runtime waveforms of *touch_channel_offsets*

Figure 6 displays the touched values, which are converted, are going high when touched, and the values start from around zero. Some values are even lower than zero.

2. Use the smallest value of the offset values as the baseline of all the channels. This process helps reduce all the common-mode noise caused by the environment condition during the runtime and also makes all the values positive. Then we get the *touch_channel_offsets2[]* array. The values in *touch_channel_offsets2[]* represent the changes caused by touch event. The bigger the change is, closer it is to the touch position, and all the changes are positive which would be easily processed. Actually, it would help to get rid of the zero-divided error in the following calculation.

```
uint32_t m_touch_val_max_idx = 0;
uint32_t m_touch_val_min_idx = 0;
```

```
/* cut off the baseline for each channel. */

for (uint32_t i = 0u; i < TOUCH_2D_ALL_COUNT; i++)
{
    if (touch_channel_offsets[i] > touch_channel_offsets[m_touch_val_max_idx])
    {
        m_touch_val_max_idx = i;
    }
    if (touch_channel_offsets[i] < touch_channel_offsets[m_touch_val_min_idx])
    {
        m_touch_val_min_idx = i;
    }
}
/* cut off the baseline in system. */
for (uint32_t i = 0u; i < TOUCH_2D_ALL_COUNT; i++)
{
    touch_channel_offsets2[i] = touch_channel_offsets[i] -
touch_channel_offsets[m_touch_val_min_idx];
}
```

The waveforms of *touch_channel_offsets2* are created, as shown in below figure.

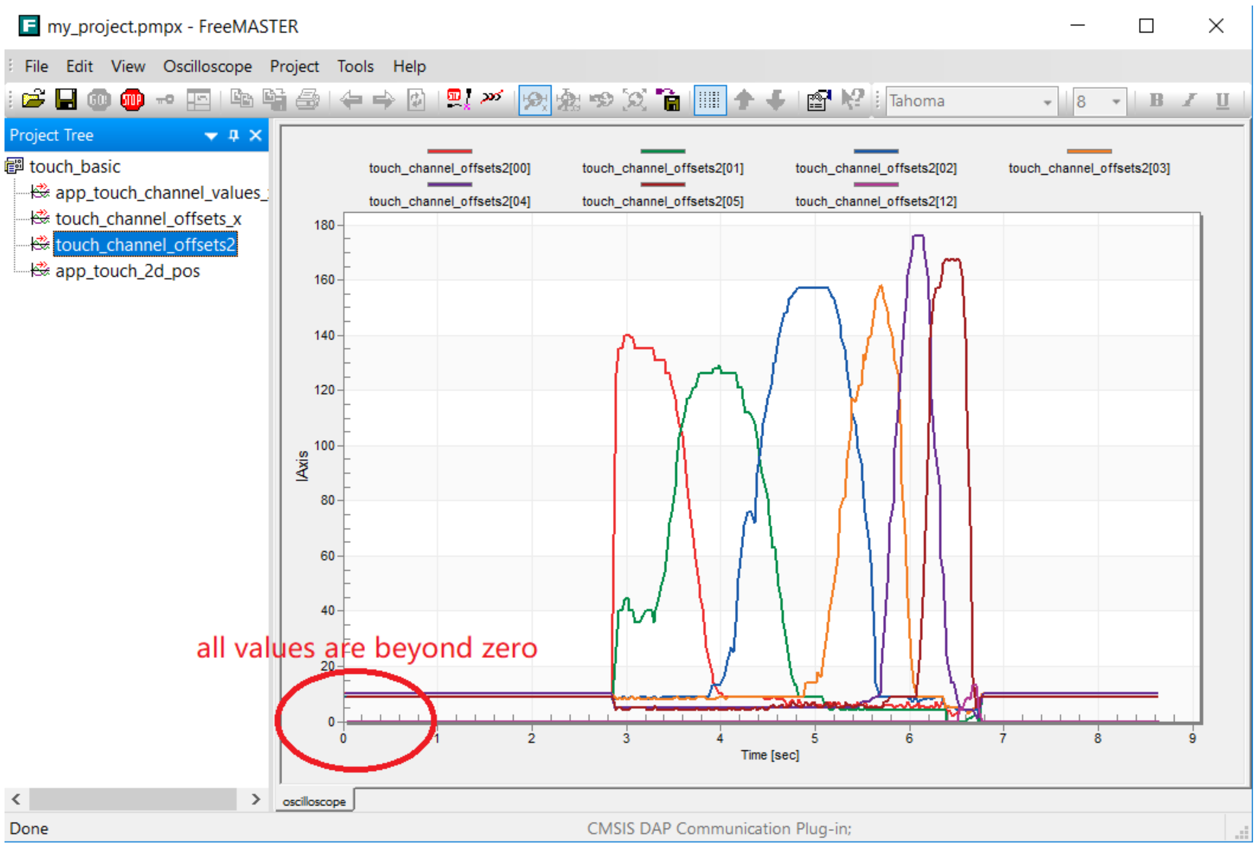

my_project.pmpx - FreeMASTER
File Edit View Oscilloscope Project Tools Help
Tahoma
8
Project Tree
touch_basic
app_touch_channel_values_
touch_channel_offsets_x
touch_channel_offsets2
app_touch_2d_pos
touch_channel_offsets2[00]
touch_channel_offsets2[01]
touch_channel_offsets2[02]
touch_channel_offsets2[03]
touch_channel_offsets2[04]
touch_channel_offsets2[05]
touch_channel_offsets2[12]
lAxis
Time [sec]
all values are beyond zero
oscilloscope
Done
CMSIS DAP Communication Plug-in;

Figure 7. Figure 7 Runtime waveforms of *touch_channel_offsets2*

Figure 7 displays that all the values are positive now.

3. Use weights to describe the touching position in the system. As the values in *touch_channel_offsets2[]* go higher, they are closer to the touched position. The values can be used as weights to describe the distance between the electrodes

and the touched point. Then the sum of weight multiplied with the anchor position is divided by the sum of weight to normalize the output into the range between anchors.

```
/* x. */
int32_t *calc_base = touch_channel_offsets2;
int32_t m_touch_calc_val[2] = {0, 0};
for (uint32_t i = 0u; i < TOUCH_2D_X_COUNT; i++)
{
    m_touch_calc_val[0] += (*calc_base) * i;
    m_touch_calc_val[1] += (*calc_base);
    calc_base++;
}
m_touch_calc_val[0] = m_touch_calc_val[0] * TOUCH_2D_X_POS_RANGE / m_touch_calc_val[1];
/* cut off the boundary offset. */
m_touch_calc_val[0] -= TOUCH_BOUNDARY_X_OFFSET;
*x_pos = (m_touch_calc_val[0] > 0) ? m_touch_calc_val[0] : 0;
```

Finally, the memory that x*_pos* points to would keep the calculated result for X axis. Similarly, the Y position, using the pointer *y_pos* in the application, can be calculated.

4 Run the demo

In the software package along with this application note, the *touch_2d_position* project is available, which can be compiled and downloaded into the board. After binary download and board reset is complete, the red LED switches on and off in a quick time, which is an indication that the calibration for touch panel is done. Then you can touch the panel with your finger and move it. The green LED would switch on when the panel is on touch and would switch off once the finger leaves. The FreeMASTER page also shows the position both on X axis and Y axis. See Figure 8.

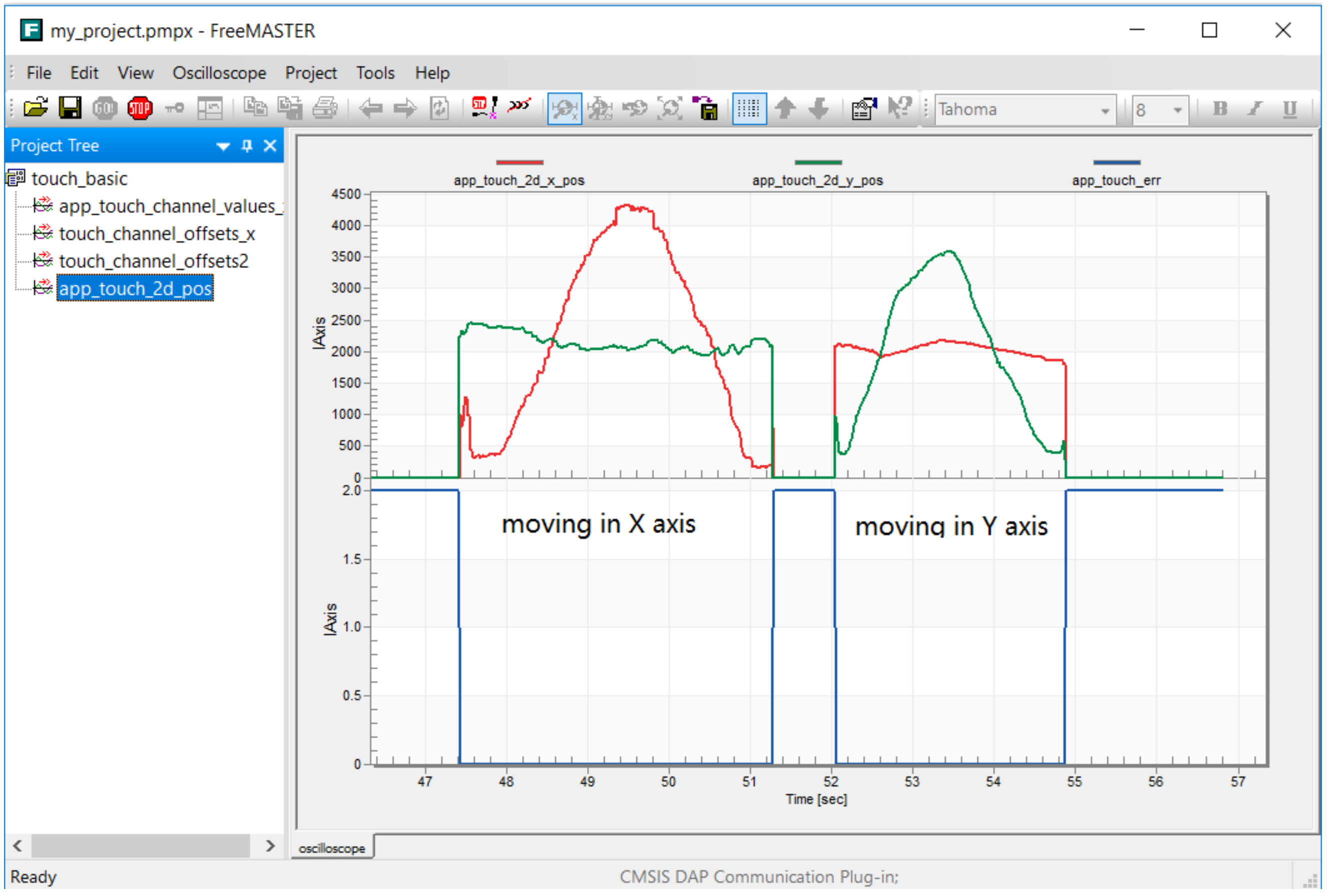


Figure 8. Figure 8 Runtime waveforms of calculated positions on X axis and Y axis

Figure 8 shows the process calculated position values on X axis and Y axis when a finger is moved horizontally and vertically. The "app_*touch*_err" value in the figure is showing the status of on touch or not touched.

5 Performance measurement

The following performance features are measured in the software package project.

5.1 Timing

- Scan a round of touch channels: 7873 us.
- Calculate the position (run *touch_2d_calc_position()*): 21 us for no touch, 32 us for on touch.

So, there is still more than 7800 us to use the position doing other work in user application.

5.2 Code Size

Here is the output log after building the demo project (using IAR for ARM).

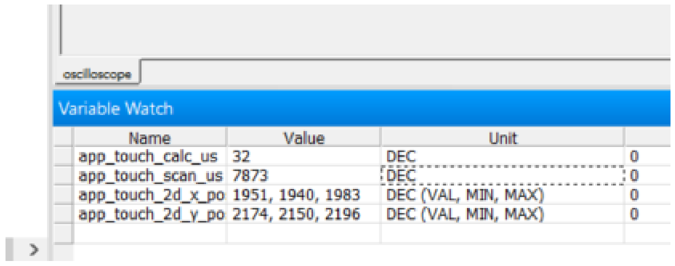
```
3 852 bytes of readonly  code memory
452 bytes of readonly  data memory
590 bytes of readwrite data memory
```

It means the whole demo project would need 3852+452+590 = 4894 bytes FLASH and 590 bytes RAM on the MCU.

5.3 Precision

Since it is difficult to move pixel by pixel, another way to measure the DPI of this touch panel is by using the floating error of calculation result while resting your finger on the touch panel without moving it.

According to the measurement record showing in FreeMASTER, as shown in below figure, the X value's floating error is 43, the Y value's floating error is 46, while the whole available range is 0 - 4095. It means the count of recognizing points for X is 95 and Y is about 89.



Name	Value	Unit	
app_touch_calc_us	32	DEC	0
app_touch_scan_us	7873	DEC	0
app_touch_2d_x_po	1951, 1940, 1983	DEC (VAL, MIN, MAX)	0
app_touch_2d_y_po	2174, 2150, 2196	DEC (VAL, MIN, MAX)	0

Figure 9. Measuring the floating error of calculation position

The available area of sensing panel is about 1.5 inch (X) x 1.3 inch (Y).Then the DPI (dot per inch) values would be 63 for X axis and 68 for Y axis.