Application Note: AZD071
Low Cost Slider Implementation

1. Introduction

This application note provides design guidelines for a capacitive touch slider. A slider using only three channels can be implemented by taking advantage of the high Signal-to-Noise ratio offered by Azoteq's devices. A capacitive slider can be designed using a 2 or 3 channel self or projected capacitance electrodes. A self-capacitance 3 channel slider will offer a better resolution than a 2-channel slider. The self-capacitance slider can be designed on a single sided ITO film whereas projected capacitance sliders typically requires two layers of ITO or a double sided PCB. This application note focuses on a 3-channel self-capacitance slider because it offers a better performance and ease of design/manufacturing.

The designer has to be familiar with Azoteq's technology before going through this application note. Please follow the application note AZD004 "Azoteq Capacitive Sensing" to learn about Azoteq's technology.

2. Design and implementation

In order to create an ideal slider design, keep the following criteria in mind:

- A fast Sample rate
- Low power consumption
- Good Signal-to-Noise Ratio
- Minimum external components

For this application note, we will focus on using the IQS213.

IQS213 characteristics:

- Low Power: 5 µA considered as a “no current”
- Speed: Normal operation sample and process time is at 3.9mS
- SNR (Signal-to-Noise Ratio): 1000:1

All of these features make the IQS213 an ideal choice for a slider application. The performance of the slider mainly depends on the PCB layout. In order to get a good resolution, it is very important to get a strong signal from the neighbouring channels. Figure 1 below shows how to interleave channels for a slider layout. More interleaving can give a better linearity. In addition, it is important to make the width of the slider equal to or smaller than the tip of a finger. The layout shown in Figure 1 shows the ideal layout for a slider with a smaller width.

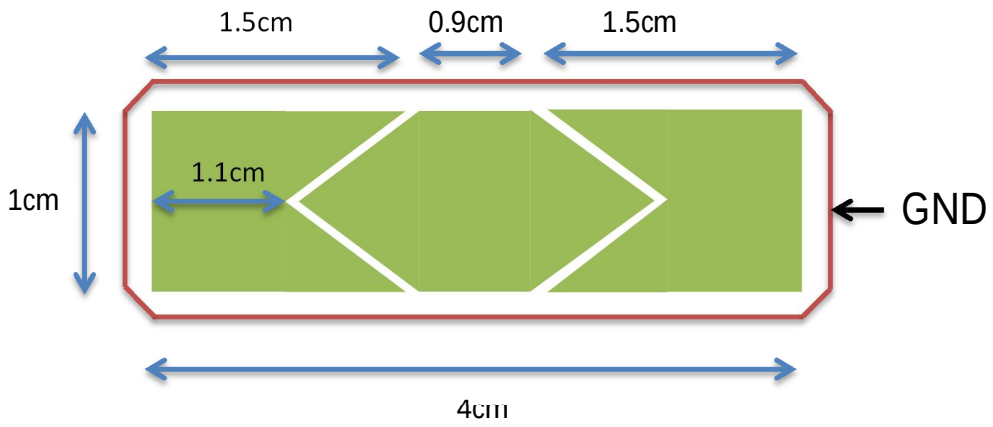


Figure 1: 3 Channel Slider Layout

It is recommended to put a ground trace around the slider to shield the electrodes from external noise. A 2mm of gap should be maintained between the ground trace and the outside edge of the slider.

It is also recommended to design a ground shield at the bottom side of PCB. The ground shield should be slightly bigger than the slider area with a hatched pattern of about 50%. This will block parasitic capacitance from interfering from the bottom side. This is very important in the case of

hand held devices. The ProxSense device should be placed as close as possible to the slider with the shortest routing traces possible. A long trace increases the touch area and therefore the noise.

The IQS213 offers two modes for communication, event mode (default) and streaming mode. In the case of event mode, an interrupt is generated for proximity, touch, swipe, and ATI events. A communication window is available after each interrupt. The user can select which events will generate an interrupt. In streaming mode, a communication window is available after each conversion. A low power slider can be implemented using event mode with the low power settings. Figure 2 shows how the interrupt should be configured for proper communication.

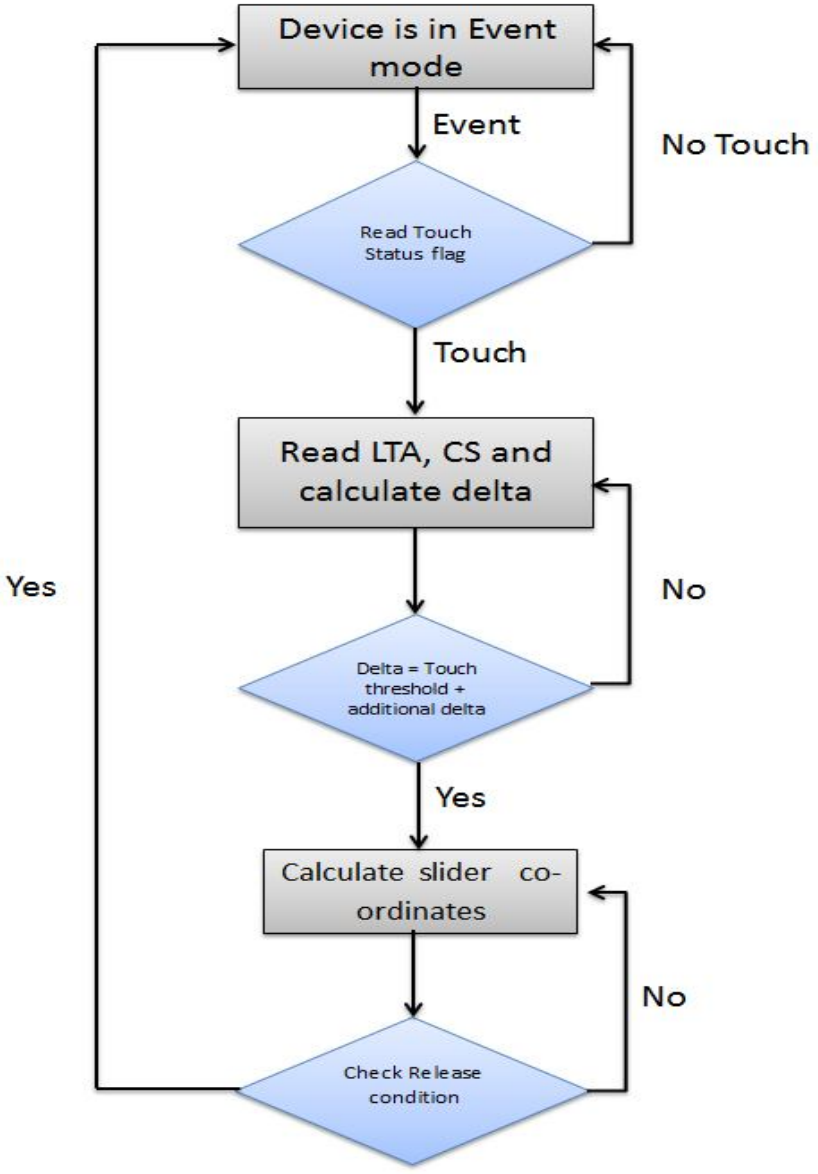
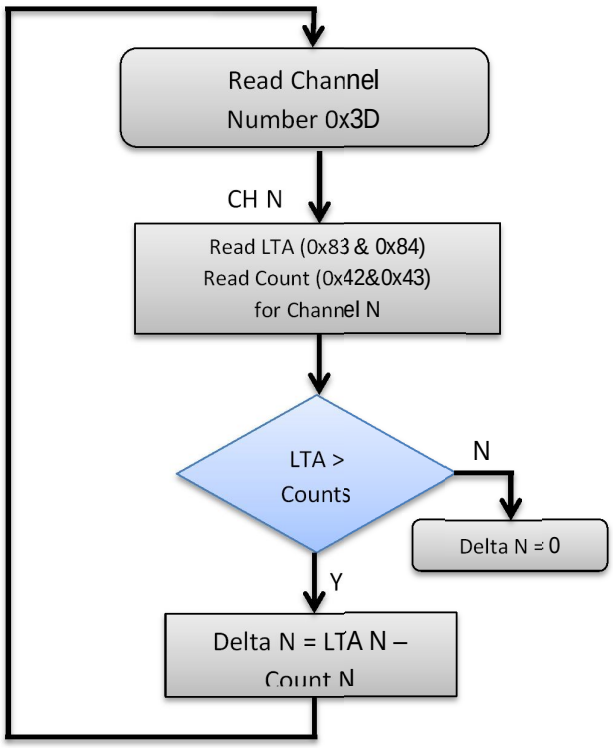


Figure 2: Flow chart for switching modes

After detecting a touch, it is good to debounce 2 or 3 deltas (Touch Threshold – Long Term Averaging) before calculating the co-ordinates. This is the best way to detect whether it is a real touch, a false trigger, or a non-sliding touch. The release condition should be the touch+additional delta. This will show a precise end co-ordinate.

At low power mode, the device samples channel 0 at a slower rate. As soon as a proximity event is detected on channel 0, the IC will start sampling all the channels at a normal rate. Channel 0 cannot be disabled while using low power mode.

Calculating the co-ordinate requires interpolation of channel data. The change in capacitance is presented as deltas by reading the difference between the Long Term Average (LTA) and the counts. Moving the finger towards the sensor will reduce the count value. The LTA filter halts on a proximity event, setting a very high proximity threshold will affect LTA halt filter and hence delta counts. The figure below shows the processing of channel data and calculated delta.



*Read channel 0 for proximity, CH0 needs to be enabled for LP mode

Figure 3: Delta calculation

A strong touch on one channel provides a large delta on that particular channel but smaller deltas on the neighbouring channels. The non-charging channels should be grounded by setting the configuration register (0xCC: bit 2). A small averaging filter is recommended for steady delta counts.

$$\textbf{Delta Average} = \frac{(\mathbf{N}-\mathbf{1}) * \textbf{Delta Average} + \textbf{Delta}}{\mathbf{N}}$$

*For example n = 128

There are several algorithms to calculate slider co-ordinates. One of the easiest algorithms is to use a weighted average. An example code is included in this application note; it is set-up to calculate 8 bits of resolution considering the size of the layout. The equation below shows an implemented weighted average, which gives 255 positions using three channels.

$$Co-ordinates = \frac{x1a1 + x2a2 + x3a3}{x1 + x2 + x3} \times \frac{2^8}{(N-1)}$$

$\{x1, x2, x3\} = \{\mathbf{delta1}, \mathbf{delta2}, \mathbf{delta3}\}$

$\{\mathbf{a1}, \mathbf{a2}, \mathbf{a3}\} = \{0, 1, 2\}$

N = No. of channels used for slider

3. Conclusion

This application note explains the design and implementation of a 3-channel self-capacitance slider. 8-bit resolution can be achieved with using only three channels. The combination of Event and Low Power modes makes the slider power efficient without affecting the response time. The current consumption can be as low as 5µA when in sleep mode. A capacitive slider with 256-steps can replace a conventional volume control or any step-up step-down form of input keys.

4. Example code

```
void main(void)
{
  CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);    /* Configures clock */

  delay_ms(1);
  delay_ms(1);

  init_i2c();    //Initiate i2c driver

  Init_IQS();    //Initialize IQS213

  for (;;)
  {
    if(!(RDY) && Cont_touch == 0)
    {
      Cont_touch = 1;
      i2c_start();
      i2c_write_register(PROX_SETTINGS1,0x60);    //Set IQS213 in streaming mode for duration of touch
      i2c_stop();
    }
```

```
    while(RDY)    //Wait for RDY to go low again this loop can be removed
    {
      _asm("nop");
    }

    if(!(RDY))
    {
      read_stream(read_buffer,10);    //Read the first 11 bytes from IQS213 starting from address 0x00

      read_buffer[PROX_Set_idx] = i2c_read_register(PROX_SETTINGS0, 1); //Check for RESET
      i2c_stop();
    }

    if((read_buffer[PROX_Set_idx]&0x80) == 0x80)
    {
      Init_IQS();    //If RESET reinitialize IQS213
    }

    if((read_buffer[Status_byte_idx]&0x04) == 0x00)    //Check if the ATI is finished
    {
      Process_data(read_buffer);    //Calculate the delta & add filtering

      if ((read_buffer[Touch_byte_idx]&0x0E) != 0x00)    //Check for touch on any channel
      {
        Slide_event();
      }

      else if((read_buffer[Touch_byte_idx]&0x0F) == 0x00)     //If no touch and no prox then clear all
    the flag
      {
        Y_pos = 0xFF;    // 0xFF represents not a valid Y positions
        First_sample = 1;   //Clear the very first byte stored in averaging filter, restore it on next
      touch
        Cont_touch = 0;
        i2c_start();
        i2c_write_register(PROX_SETTINGS1, 0x20);      //Set the device back in event mode
        i2c_stop();
      }
    }
  }
}


void Process_data(unsigned char *read_buffer)
{
  channel = read_buffer[Chan_num_idx];     // Active Channel

  // Read count values from IQS 213 of channel
  count[channel]=((unsignedint)(read_buffer[Count_High_idx])<<8)|(unsigned int)
(read_buffer[Count_Low_idx]);

  // Read LTA from IQS 213 of active channel
  lta[channel]=((unsigned int)(read_buffer[LTA_High_idx])<<8)|(unsigned
int)(read_buffer[LTA_Low_idx]);

  if(lta[channel] > count[channel])
  {
    CH_delta[channel] = lta[channel] - count[channel];    //Calculate delta for each channel
  }
  else
  {
    CH_delta[channel] = 0;    //Ignore the reverse delta
  }
  if(First_sample == 1)
  {
    CH_delta_avg[channel] = CH_delta[channel];     //Store the very first CH delta
    First_sample = 0;
  }

  CH_delta_avg[channel] =(CH_delta_avg[channel]*3+ CH_delta[channel])/4;    //Averaged delta from
last 4 samples
}
```

```
void Slide_event(void)
{
  if (channel == 3) //Wait until channel 3 data is available to avoid repeated data on channel 1 & 2
  {

    if((CH_delta[1] > 105) || (CH_delta[2] > 105) || (CH_delta[3] > 105))     //set the release
  condition
    {
      Y_long = (CH_delta_avg[2] + CH_delta_avg[3] *2);
      Y_long =  Y_long *127;    // Change the multiplier to change resolution
      Y_pos = (Y_long)/(CH_delta_avg[3]+CH_delta_avg[2]+CH_delta_avg[1]);    //Calculate Sliding
    positions 8bit

      if(Y_pos >= 255)
      {
        Y_pos = 254;
      }
    }
  }
}



#define Product_byte_idx     0

#define Version_byte_idx     1

#define Status_byte_idx      2

#define Swipe_byte_idx       3

#define Touch_byte_idx       4

#define Chan_num_idx         5

#define Count_High_idx       6

#define Count_Low_idx        7

#define LTA_High_idx         8

#define LTA_Low_idx          9

#define PROX_Set_idx         10
```