

PCT CONTROLLER SPECIFICATION AND PROGRAMMING GUIDE

Product Specification

| | |
|-----------------------|---|
| PRODUCT NUMBER | ITO PROJECTIVE CAPACITIVE TOUCH CONTROLLER |
|-----------------------|---|

| INTERNAL APPROVALS | | |
|--------------------|-------------|------------------|
| Product Manager | Engineering | Document Control |
| | | |

TABLE OF CONTENTS

| | | |
|---|--------------------------------|----|
| 1 | GENERAL DESCRIPTION | 3 |
| 2 | PACKAGING DRAWING | 4 |
| 3 | COMMUNICATION INTERFACE..... | 5 |
| 4 | REPORT PACKET | 7 |
| 5 | ILLUSTRATION OF GESTURES | 9 |
| 6 | SAMPLE CODE | 10 |

REVISION RECORD

| Rev. | Date | Page | Par. | Comment | ECN no. |
|------|----------|------|------|-----------------------------|---------|
| A | 11/04/08 | -- | -- | New DCA Specification | -- |
| B | 06/04/09 | -- | -- | Updated Gesture Description | -- |
| C | 08/06/09 | 10 | -- | Add sample code. | -- |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

1 GENERAL DESCRIPTION

This projected capacitive touch controller is designed for small scale touch panel applications in portable devices. The system block diagram is as shown in Fig. 1. has up to 22 channels to connect the touch screen and three communication pins connected between CPU and Controller, which are including external interrupt INT, I2C pins SCL and SDA. The INT is active low while the touch state is calculated by Controller and the touch information can be translated via I2C communication interface. The controller specifications are as shown in Table 1. The I2C data format, protocol, report packet and sample code are described as following.

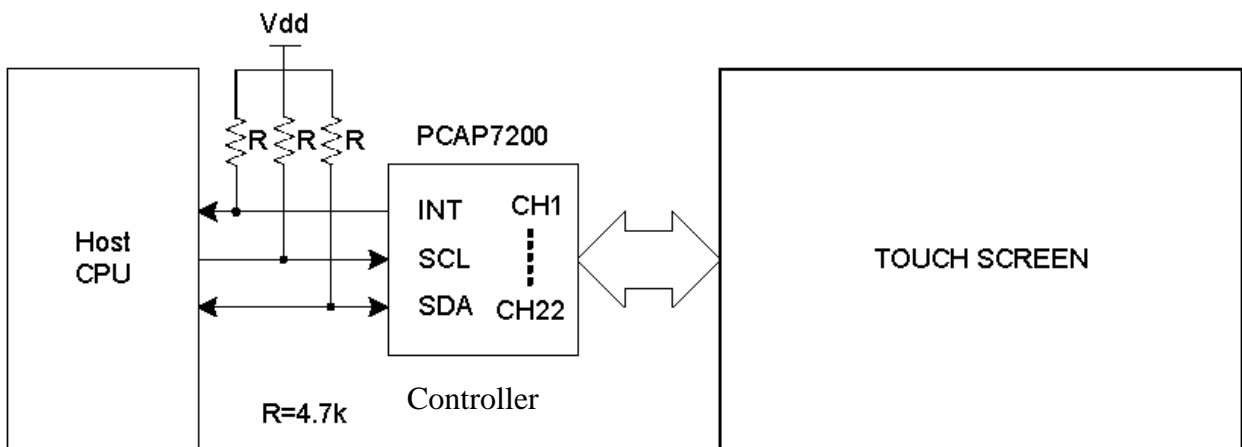


Figure 1. System Block Diagram

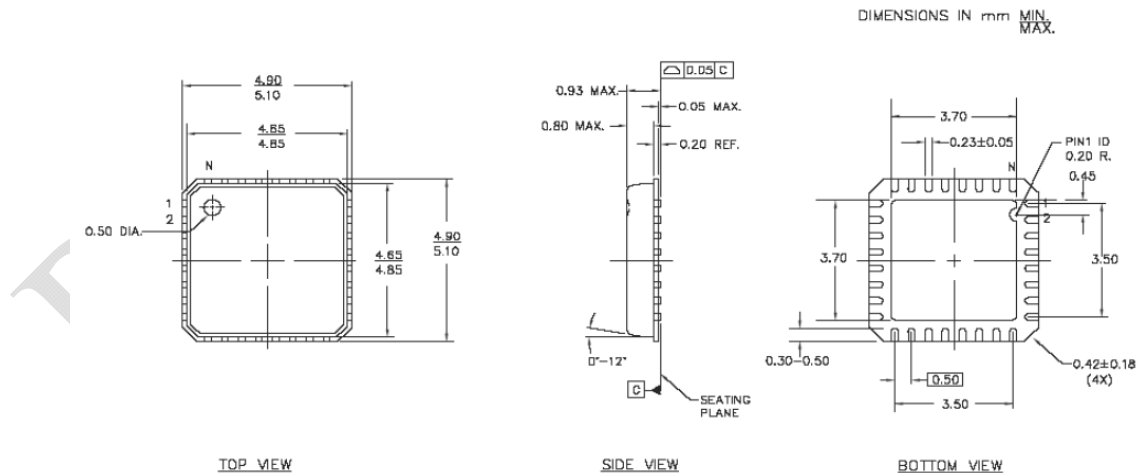
Table 1. Controller Specifications

| | |
|-----------------------|--|
| Channels | Up to 22 CH |
| Interface | I ² C Slave 100 KHz @ 5V ; 50 KHz @ 3.3V |
| Slave address | 0Ah |
| Resolution | Up to 2048 x 2048 (Configurable) |
| Report Rate | 40 ~ 100 points/sec |
| Response Time | 25 ms |
| Gesture | Zoom, 1 st direction Zoom, 2 nd direction Zoom, Rotate, 1 st direction Slide, 2 nd direction Slide |
| Power Supply Voltage | 5 / 3.3V DC |
| Power Supply Current | 10 / 0.06 mA @ 5V 5 / 0.06 mA @ 3.3V (Active / Sleep) |
| Operating Temperature | -40°C to 85°C |
| Package | 32 pins QFN |
| Package Dimension | 5 mm x 5 mm x 0.93 mm |

Notice:

1. This documentation is preliminary specification. So the detail specification may be changed without any notice.
2. NDA Confidential. Do Not Distribute.

2 PACKAGING DRAWING

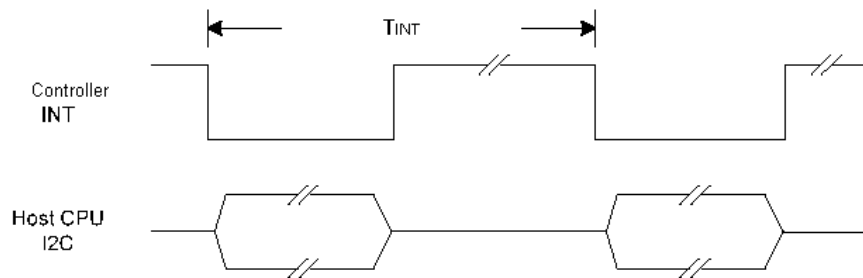
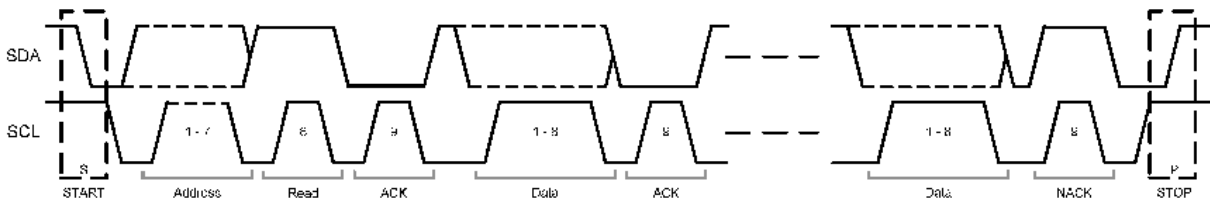


3 COMMUNICATION INTERFACE

Pin assignment and Definitions

| Pin | Name | I/O | Description |
|-----|------|-----|---|
| 1 | GND | | Ground |
| 2 | SDA | I/O | I2C Data |
| 3 | SCL | I/O | I2C Clock |
| 4 | VDD | | Power Supply Voltage |
| 5 | INT | O | Interrupt active low, asserted to request Master start a new transaction. |
| 6 | XRES | I | Active high external reset with internal pull down. Minimum of Pulse Width is 10 μ s. |

I²C Data format



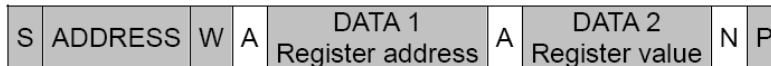
Report rate = $1 / T_{INT}$, it depends on properties of touch screen such as resistive value, channel number, thickness and material of cover lens, etc.

Protocol

Read mode: Master-receiver, slave-transmitter.



Write mode: Master-transmitter, slave-receiver.



| | |
|--|----------------------|
| | From Master to Slave |
| | From Slave to Master |

| | | |
|---------|---|--------------------------------|
| S | = | START condition |
| P | = | STOP condition |
| R | = | Data direction READ (SDA HIGH) |
| W | = | Data direction WRITE (SDA LOW) |
| A | = | Acknowledge (SDA LOW) |
| N | = | Not acknowledge (SDA HIGH) |
| ADDRESS | = | 7-bit (0Ah) |
| DATA | = | 8-bit |

4 REPORT PACKET

4.1 Single Touch

Each report packet contains 6 bytes. The single touch packet format is as follows:

| | MSB | | | | | | | LSB |
|--------|-----|----------|----|----|-----|----|----|--------|
| Byte 1 | 1 | Reserved | | | | | | Status |
| Byte 2 | 0 | 0 | 0 | 0 | A10 | A9 | A8 | A7 |
| Byte 3 | 0 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| Byte 4 | 0 | 0 | 0 | 0 | B10 | B9 | B8 | B7 |
| Byte 5 | 0 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| Byte 6 | 0 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Status: indicates the touch status: 1 for touch down and 0 for touch up.

A10 – A0: 11 bits of 1st direction raw data

B10 – B0: 11 bits of 2nd direction raw data

P6 – P0: 7 bits of finger pressure

Please be aware that A and B just represent 2 resolution directions of the touch panel. The reported coordinates are (0~2047, 0~2047), the bottom left is (0, 0).

4.2 Multi Touch – Gesture

The multi-touch gesture packet format is as follows:

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|--------|--------|--------|---------|--------|----------|
| 0x0A | 0x03 | 0x38 | Gesture | Value | Reserved |

There are six gesture events defined and the range of value is between -127 to +127. The Gesture events and value definition are as follows.

| Event | Gesture | Value | |
|---------------------------------|---------|-------|-------|
| | | + | - |
| ZOOM | 0x01 | In | Out |
| 1 st direction ZOOM | 0x02 | In | Out |
| 2 nd direction ZOOM | 0x03 | In | Out |
| ROTATE | 0x04 | CCW | CW |
| 1 st direction SLIDE | 0x05 | Left | Right |
| 2 nd direction SLIDE | 0x06 | Up | Down |

For example, the packet of zoom in is as follows:

0x0A, 0x03, 0x38, 0x01, 0x01, 0x00

The packet of rotate clockwise is as follows:

0x0A, 0x03, 0x38, 0x04, 0xFF, 0x00

4.3 Get Firmware Version

The firmware version packet format is as follows:

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|--------|--------|--------|--------|--------|--------|
| 0x0A | 0x04 | 0x44 | Major | Minor1 | Minor2 |

The firmware version is **v Major.Minor1 Minor2**. The Major, Minor1 and Minor2 are expressed as characters. For example, the packet of firmware version v1.04 is as follows:

0x0A, 0x04, 0x44, 0x31, 0x30, 0x34

4.4 Operating Mode Register

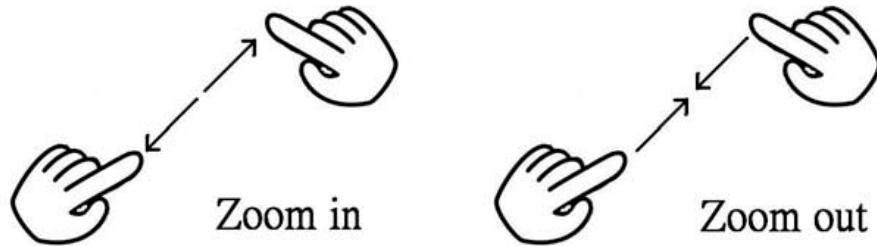
| Register Address | Register Name | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Default | |
|------------------|----------------|--------------------------------------|------|------|------|------|-------|--------|------|----------------|------|
| 0x07 | Operating Mode | Wakeup and Sleep | | | | | | | | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Wakeup / Sleep | 0x01 |
| | | Single touch and Multi touch gesture | | | | | | | | | |
| | | 0 | 0 | 0 | 1 | 0 | Slide | Rotate | Zoom | | 0x10 |
| | | Get firmware version | | | | | | | | | |
| | | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | | |

Sleep: Write (0x07,0x00), the chip sleep and power saving.

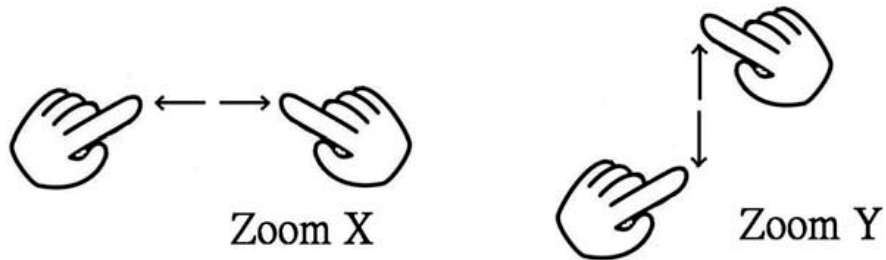
Wakeup: Enable the I2C using SDA falling edge when Controller is on sleep state, and write (0x07,0x01) to wakeup the chip.

5 ILLUSTRATION OF GESTURES

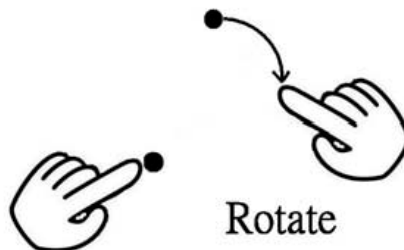
<Gesture 0x01>



<Gesture 0x02 / 0x03>



<Gesture 0x04>



<Gesture 0x05 / 0x06>



6 SAMPLE CODE

Firmware - Host CPU gets 6 bytes packet from Controller via I2C

Example 1

```
// Parameter define
#define TP_SLAVE_ADDR 0x0A
#define SCLK_HIGH sbIIC_SCLK = HIGH;
#define SCLK_LOW sbIIC_SCLK = LOW;
#define IIC_SDA_IN sbIIC_SDA = HIGH;
#define DELAY_TIME 6

// main.c while (1)
{
if ( IsGetPCAPData )
{
// Add user code here to proceed the DATA translated
// Report packet by I2C_Rx_Q[6]
}
}

// IsGetPCAPData function
BYTE IsGetPCAPData ( void )
{
// Interrupt active low; g_byGetI2CData = 1
if ( g_byGetI2CData )
{
if ( I2CReceive( TP_SLAVE_ADDR << 1 , byI2C_Rx_Q , 6 ))
{
g_byGetI2CData = 0;
return TRUE;
}
else return FALSE;
}
else return FALSE;
}

// I2CReceive function
BYTE I2CReceive( BYTE byAddr, char *pBuffer, char chBytes )
{
char chIter; BYTE byTemp;
```

| | |
|-------------|--------|
| Product No. | REV. C |
|-------------|--------|

| | |
|------|---------|
| Page | 10 / 23 |
|------|---------|

```
byAddr |= 0x01;    // set R/W bit = 1
IICStartBit();
IICSendData( byAddr );    // set Address byTemp = IICWaitACK();
```

```
for ( chIter = 0; chIter < chBytes; chIter++ )
{
if ( byTemp )
*pBuffer++ = IICReceiveData();
```

```
else
```

```
break;
```

```
if ( chIter < ( chBytes - 1 ) ) IICSendACK();
```

```
else
{
```

```
}
}
```

```
SCLK_LOW;
sbIICSDA = LOW; PusedoDelay( DELAY_TIME ); sbIICSDA = HIGH;
SCLK_HIGH;
PusedoDelay( DELAY_TIME ); SCLK_LOW;
```

```
IICStopBit();
return byTemp;
}
```

```
// I2CSend function
void I2CSend( BYTE byAddr, char *pBuffer, BYTE chBytes )
{
char chIter; BYTE byTemp;
```

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 11 / 23 |
|------|---------|

```

byAddr &= 0xFE;    // set R/W bit = 0
IICStartBit();
IICSendData( byAddr );    // set Address byTemp = IICWaitACK();

for ( chIter = 0; chIter < chBytes; chIter++ )
{
if ( byTemp )
IICSendData( *pBuffer++ );

else

break;

byTemp = IICWaitACK();
}

byTemp = IICWaitNACK;

if ( byTemp )
IICStopBit();
}

// IICStartBit function
void IICStartBit( void )
{
SCLK_HIGH;
sbIICSDA = HIGH; PusedoDelay( DELAY_TIME ); sbIICSDA = LOW; PusedoDelay(
DELAY_TIME );
}

// IICSendData function
void IICSendData( BYTE byChar )
{
BYTE chIter;
for ( chIter = 0; chIter < 8; chIter++ )
{
SCLK_LOW;
PusedoDelay( DELAY_TIME );
sbIICSDA = ( byChar & BYTEMsb ) ? HIGH : LOW; PusedoDelay( DELAY_TIME );
SCLK_HIGH;
PusedoDelay( DELAY_TIME );
byChar <<= 1;
}
}

```

```

}
SCLK_LOW;
}

// IICWaitACK function
bit IICWaitACK( void )
{
IIC_SDA_IN;
PusedoDelay( DELAY_TIME ); SCLK_LOW;
PusedoDelay( DELAY_TIME );
SCLK_HIGH;
PusedoDelay( DELAY_TIME );
if ( !sbIICSDA )
{

}
else
{

}
}

SCLK_LOW;
return TRUE;

SCLK_LOW;
return FALSE;

// IICWaitNACK function
bit IICWaitNACK( void )
{
IIC_SDA_IN;
PusedoDelay( DELAY_TIME ); SCLK_LOW;
PusedoDelay( DELAY_TIME ); SCLK_HIGH;

```

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 13 / 23 |
|------|---------|

```
PusedoDelay( DELAY_TIME );
if ( sbIICSDA )
{
```

```

}
else
{
```

```

}
}
```

```
SCLK_LOW;
return TRUE;
```

```
SCLK_LOW;
return FALSE;
```

```
// IICReceiveData function
BYTE IICReceiveData( void )
{
  BYTE byIter;
  BYTE chIIC_RX_TEMP = 0; IIC_SDA_IN;
  for( byIter = 0; byIter < 8; byIter++ )
  {
    SCLK_LOW;
    PusedoDelay( DELAY_TIME ); SCLK_HIGH;
    PusedoDelay( DELAY_TIME );
    if( sbIICSDA )
      chIIC_RX_TEMP |= BYTELSB;
    if ( byIter < 7 )
      chIIC_RX_TEMP <<= 1;
  }
  SCLK_LOW;
  return chIIC_RX_TEMP;
}
```

```
// IICSendACK function
```

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 14 / 23 |
|------|---------|

```
void IICSendACK( void )
{
SCLK_LOW;
sbIICSDA = LOW; PusedoDelay( DELAY_TIME ); SCLK_HIGH;
PusedoDelay( DELAY_TIME ); SCLK_LOW;
PusedoDelay( DELAY_TIME );
}
```

```
// IICStopBit function
void IICStopBit( void )
{
SCLK_LOW;
PusedoDelay( DELAY_TIME ); sbIICSDA = LOW; PusedoDelay( DELAY_TIME );
SCLK_HIGH;
PusedoDelay( DELAY_TIME ); sbIICSDA = HIGH; PusedoDelay( DELAY_TIME );
}
```

```
// PusedoDelay function
void PusedoDelay( int iCnt )
{
int i;
for ( i = 0; i < iCnt; i++ ){ }
}
```

Example 2

```
#define TP_SLAVE_ADDR 0x0A

#define delay_short 200
#define delay_long 200 //i2c rate is 50KHz

#define TP_SET_I2C_CLK_OUTPUT GPIO_InitIO(OUTPUT,TP_SCK)
#define TP_SET_I2C_DATA_OUTPUT GPIO_InitIO(OUTPUT,TP_SDA)
#define TP_SET_I2C_CLK_INPUT GPIO_InitIO(INPUT,TP_SCK)
#define TP_SET_I2C_DATA_INPUT GPIO_InitIO(INPUT,TP_SDA)

#define TP_SET_I2C_CLK_HIGH GPIO_WriteIO(1,TP_SCK)
#define TP_SET_I2C_CLK_LOW GPIO_WriteIO(0,TP_SCK)
#define TP_SET_I2C_DATA_HIGH GPIO_WriteIO(1,TP_SDA)
#define TP_SET_I2C_DATA_LOW GPIO_WriteIO(0,TP_SDA)
#define TP_GET_I2C_DATA_BIT GPIO_ReadIO(TP_SDA)
#define TP_GET_I2C_CLK_BIT GPIO_ReadIO(TP_SCK)

#define TP_I2C_ADDR_R ((TP_SLAVE_ADDR<<1) | 0x1 )
#define TP_I2C_ADDR_W ((TP_SLAVE_ADDR<<1) | 0x0 )

void serial_init(void)
{
GPIO_ModeSetup(TOUCH_PANEL_RESET_PIN,0x0);
GPIO_InitIO(OUTPUT,TOUCH_PANEL_RESET_PIN);
GPIO_WriteIO(0,TOUCH_PANEL_RESET_PIN);

GPIO_ModeSetup(TP_SCK, 0x0); GPIO_ModeSetup(TP_SDA, 0x0);

TP_SET_I2C_CLK_OUTPUT; TP_SET_I2C_CLK_HIGH; TP_SET_I2C_DATA_OUTPUT;
TP_SET_I2C_DATA_HIGH;
}

void IICStartBit( void )
{
kal_int32 j; TP_SET_I2C_DATA_OUTPUT; TP_SET_I2C_CLK_OUTPUT;

TP_SET_I2C_DATA_HIGH; TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_short;j++);
TP_SET_I2C_DATA_LOW; for(j=0;j<delay_short;j++);
```

| | |
|-------------|--------|
| Product No. | REV. C |
|-------------|--------|

| | |
|------|---------|
| Page | 16 / 23 |
|------|---------|

```

}

void IICSendData( kal_uint8 byChar )
{
int i,j; TP_SET_I2C_DATA_OUTPUT; TP_SET_I2C_CLK_OUTPUT;

TP_SET_I2C_CLK_LOW; TP_SET_I2C_DATA_LOW;

for (i=7;i>=0;i--)
{
/* data bit 7~0 */ TP_SET_I2C_CLK_LOW; for(j=0;j<delay_short;j++); if (byChar &
(1<<i)){
TP_SET_I2C_DATA_HIGH;
}else{ TP_SET_I2C_DATA_LOW;
} for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_short;j++);
}

TP_SET_I2C_CLK_LOW;
}

kal_uint8 IICReceiveData( void )
{
kal_int32 i,j; kal_int16 tmp=0; TP_SET_I2C_DATA_INPUT;

for (i=7;i>=0;i--)
{
// data bit 7~0
TP_SET_I2C_CLK_LOW;
for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_HIGH;
for(j=0;j<delay_short;j++);
if (TP_GET_I2C_DATA_BIT)
tmp |= (1<<i);
} TP_SET_I2C_CLK_LOW; return tmp;
}

void IICStopBit( void )
{
kal_uint32 j; TP_SET_I2C_CLK_OUTPUT; TP_SET_I2C_DATA_OUTPUT;

```

```
TP_SET_I2C_CLK_LOW; for(j=0;j<delay_short;j++); TP_SET_I2C_DATA_LOW;
for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_short;j++);
TP_SET_I2C_DATA_HIGH; for(j=0;j<delay_short;j++);
}
```

```
kal_bool IICWaitACK( void )
```

```
{
kal_int16 j,tmp;
int timeout = 100; TP_SET_I2C_DATA_INPUT;
```

```
for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_LOW; for(j=0;j<delay_short;j++);
TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_short;j++);
```

```
while(timeout--)
{
tmp = TP_GET_I2C_DATA_BIT;
if( !tmp ){ TP_SET_I2C_CLK_LOW; break;
}
for(j=0;j<delay_short;j++);
}
```

```
if(timeout==0)
return KAL_FALSE;
```

```
return KAL_TRUE;
}
```

```
void IICSendACK( void )
```

```
{
kal_int16 j; TP_SET_I2C_DATA_OUTPUT;
```

```
TP_SET_I2C_CLK_LOW; TP_SET_I2C_DATA_LOW;
```

```
for(j=0;j<delay_short;j++);
TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_long;j++); TP_SET_I2C_CLK_LOW;
}
```

```
void IICSendNoACK( void )
```

```
{
```

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 18 / 23 |
|------|---------|

```
kal_int16 j; TP_SET_I2C_DATA_OUTPUT;
```

```
TP_SET_I2C_CLK_LOW; TP_SET_I2C_DATA_LOW; for(j=0;j<delay_short;j++);
TP_SET_I2C_DATA_HIGH; TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_long;j++);
TP_SET_I2C_CLK_LOW;
}
```

```
kal_bool IICWaitNoACK( void )
```

```
{
kal_int16 j,tmp;
int timeout = 100;
```

```
TP_SET_I2C_DATA_HIGH;
for(j=0;j<delay_short;j++);
```

```
TP_SET_I2C_DATA_INPUT; for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_LOW;
for(j=0;j<delay_short;j++); TP_SET_I2C_CLK_HIGH; for(j=0;j<delay_short;j++);
```

```
while(timeout--)
{
tmp = TP_GET_I2C_DATA_BIT;
if(tmp){ TP_SET_I2C_CLK_LOW; break;
}
for(j=0;j<delay_short;j++);
}
```

```
if(timeout==0)
return KAL_FALSE;
```

```
return KAL_TRUE;
}
```

```
kal_int8 serial_read_data(kal_uint8 *get_byte,kal_uint8 n)
```

```
{
kal_int32 i;
int ret;
```

```
IICStartBit();
IICSendData( TP_I2C_ADDR_R );
ret = IICWaitACK();
if(ret != KAL_TRUE){ IICStopBit();
kal_print("send slave addr:no ack:%d\n");
return -1;
```

| | |
|-------------|--------|
| Product No. | REV. C |
|-------------|--------|

| | |
|------|---------|
| Page | 19 / 23 |
|------|---------|

```

}

for( i=0; i < n; i++ ){
get_byte[i] = IICReceiveData();

if( i == n-1 ) IICSendNoACK();
else IICSendACK();
}

IICStopBit();

return 0;
}

void serial_write_data(kal_uint8 reg,kal_uint8 data)
{
int ret; IICStartBit();
//----- I2C addr Write-----
IICSendData(TP_I2C_ADDR_W); /* 0x0a -> slave address & write */
ret = IICWaitACK();
if(ret != KAL_TRUE){
kal_print("i2c read: Addr&W ACK Error\n");
goto out;
}
IICSendData( reg ); //write reg address ret = IICWaitACK();
if(ret != KAL_TRUE){
kal_print("i2c read: Reg ACK Error\n");
goto out;
}
IICSendData(data); //write reg value ret = IICWaitNoACK();
if(ret != KAL_TRUE){
kal_print("i2c read: Addr&R ACK Error\n");
goto out;
}

out:

IICStopBit();
return;
}

```

Software – Parsing Packet

After retrieved data from I2C, put it into a queue. The queue can be read via pPort->Read

method. Create a thread called PortThreadRoutine, this thread gets data from queue and translate data into single point or multi-touch gesture. Please add the behavior of what to do after recognized a single point or multi gesture.

```
#define MAX_BUFFER    1024
#define MOUSE_PACKET_LEN    6
#define MAX_CMD_LEN    16
#define POLLING_BUFFER_SIZE    3
#define _SYNCBIT    0x80
#define _SOP    0x0A

unsigned  stdcall PortThreadRoutine( LPVOID pContext )
{
    CPort *pPort = ( CPort *) pContext; CHAR pBuffer[ MAX_BUFFER ];
    CHAR pMsgBuffer[ MAX_BUFFER ]; DWORD dwRead = 0;
    DWORD dwCnts = 0;
    BOOL bPointPacket = FALSE ; BOOL bCmdPacket    = FALSE; DWORD dwCmdPacketLen;
    UCHAR ucChar; INT i;

    while( TRUE )
    {
        if( WAIT_OBJECT_0 == ::WaitForSingleObject( pPort->m_hStopEvent, 0 ) )
        {
            //condition for exit the thread return 100;
        }
        // read packet from I2C data queue
        if ( pPort->Read( pBuffer, POLLING_BUFFER_SIZE, &dwRead, pPort->m_hReadEvent ) )
        {
            // parse the packet
            for( i = 0; i < (INT)dwRead; i++ )
            {
                ucChar = pBuffer[ i ];
                if( ( pBuffer[ i ] & 0xF0 ) == _SYNCBIT ) && !bCmdPacket )
                {
                    dwCnts = 0;
                    pMsgBuffer[ dwCnts ] = pBuffer[ i ];
                    bPointPacket = TRUE;
                    dwCnts++;
                    continue;
                }
            }
        }
    }
}
```

| | |
|-------------|--------|
| Product No. | REV. C |
|-------------|--------|

| | |
|------|---------|
| Page | 21 / 23 |
|------|---------|

```

}
else if( _SOP == ucChar && !bPointPacket && !bCmdPacket )
{
bCmdPacket = TRUE; dwCmdPacketLen = ( DWORD )-1; bPointPacket = FALSE; continue;
}
else if( bCmdPacket )
{
if( ( DWORD )-1 == dwCmdPacketLen )
{
dwCmdPacketLen = ( DWORD )pBuffer[ i ];
dwCnts = 0;
if( dwCmdPacketLen > MAX_CMD_LEN )
dwCmdPacketLen = MAX_CMD_LEN;
continue;
}
pMsgBuffer[ dwCnts ] = pBuffer[ i ];
dwCnts++;
if( dwCmdPacketLen == dwCnts )
{

```

```

dwCmdPacketLen = 0;
pMsgBuffer[ dwCnts ] = 0;
dwCnts++;
// Here, a completely Cmd packet received !!!
// Do what you want to do!
// For instance, check pMsgBuffer and send event according
// to Multi-touch type dwCnts = 0;
bCmdPacket = FALSE;
continue;
}
continue;
}
if( bPointPacket )
{
pMsgBuffer[ dwCnts ] = pBuffer[ i ];
dwCnts++;
if( MOUSE_PACKET_LEN == dwCnts )
{
// Here, a completely point packet received !!!
// Do what you want to do!
// For instance, send mouse event dwCnts = 0;
bPointPacket = FALSE;

```

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 22 / 23 |
|------|---------|

}
continue;
}
}
}
}
}

PRELIMINARY

| | | |
|-------------|--|--------|
| Product No. | | REV. C |
|-------------|--|--------|

| | |
|------|---------|
| Page | 23 / 23 |
|------|---------|