TouchKit
Software Programming Guide
Version 1.4

Contents

Chapter 1. Introduction

EETI provides a full range of controllers designed to optimize the performance of analog resistive touch panels. The controller communicates with the PC system directly through RS232, PS/2, USB port and even I^2C. In recent years, portable devices become popular, and I^2C transaction is the best way to communicate with these portable devices, like PDA, eBook, Mira, etc.

EETI's superior design combines accuracy, sensitivity and speed to reach the outstanding touch performance and ease of use. The drivers emulate the mouse input and right button function, and support a variety of operation systems, including DOS, Windows 98, Windows NT4, Windows 2000, Windows Me, Windows XP, Windows CE.net, iMac, Linux RedHat and Mandrake Linux.

However some special designs, our honor customers have to develop their own programs communicating with the touch panel controller firmware directly. **In chapter 1** of this application note, firstly the needed protocols are described. Then special notices of programming RS232 is expressed. At the end, the sample code of parsing the protocols and the two points calibration conversion formulas are listed.

And some OEM customers need to use their own logos and brand names instead of that of . Therefore **chapter 2** of this application note is designed to fit our honor customers' needs on Windows OS.

.

1.1 Protocol

All TouchKit controllers including RS232 4-wire use the protocols. And the protocols can be classified into 2 groups: Diagnostics Packet and Report Packet.

1.1.1 Diagnostics Packet

These packets are issued from the host for querying some device information. The controller firmware will report the corresponding data to the host. The packet format is as follows:

0x0A	LengthInByte + 1	Command	Response
1 Byte	1 Byte	1 Byte	LengthInByte Bytes

The maximum packet size is 16 bytes. The first byte is Start of Packet as 0X0A. The second byte is the length of Response. The third byte is the issued command and the last part (length is defined in second byte) is the response from controller firmware.

1. Check active : this packet is to check if the device is working properly.
   Host issues

0x0A	1	‘A’

   Device responds when active

0x0A	1	‘A’

2. Get firmware version
   Host issues

0x0A	1	‘D’

   Controller firmware responds

0x0A	Length	‘D’	Response

   The response is an ASCII string, such as ‘0.99’

3. Get type

This packet is to request the controller type.

Host issues

0x0A	1	'E'	

Controller firmware responds

0x0A	Length	'E'	Response

1.1.2 Report Packet

Touchkit USB HID Touchscreen controllers support Microsoft HID touch digitizer. By default, Touchkit HID compatible controller report with HID format for coordination data according to the HID report descriptor it reported to Host system. In addition, Touchkit serial RS232 controllers support emulation modes. Serial controller's report format depends on the format of command sets it receives from Host. By default, it reports with non-emulated packet format as below. ***To make sure the controllers to report with the below format, the host driver should issue any one of diagnostics packet data to controller***. For example, host driver may send a "Check Active"(0x0A, 1, 'A') packet data to controller to make it report with below report format.

Each report packet may contain 5 or 6 bytes as below:

	MSB							LSB
Byte0	1	Z	M	0	0	AD1	AD0	Status
Byte1	0	A13	A12	A11	A10	A9	A8	A7
Byte2	0	A6	A5	A4	A3	A2	A1	A0
Byte3	0	B13	B12	B11	B10	B9	B8	B7
Byte4	0	B6	B5	B4	B3	B2	B1	B0
Byte5	0	P6	P5	P4	P3	P2	P1	P0

Byte0: Byte0 is the header of the point packet. It contains below point Information

Z : pressure bit. Touchkit controller SAW technology may report with pressure information.

Z=0 means no pressure information

Z=1 means Byte5 is pressure information.

M: Player ID. Touchkit multipler controller report player ID information

M=0 means no player ID information

M=1 means Byte5 is player ID

Status: touch down status.

Status = 1 means touch down

Status = 0 means lift off point

Byte1~Byte4:

AD1,AD0: resolution information of the current point coordination.

AD1:AD0 = 0:0 means the coordination resolution is 11 bits

AD1:AD0 = 0:1 means the coordination resolution is 12 bits

AD1:AD0 = 1:0 means the coordination resolution is 13 bits

AD1:AD0 = 1:1 means the coordination resolution is 14 bits

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

A10/A11/A12/A13 – A0: 11/12/13/14 bits of 1st direction raw data

B10/B11/B12/B13 – B0: 11/12/13/14 bits of 2nd direction raw data

Please be aware that A and B just represent 2 resolution directions of the touch panel.

Byte5: Pressure or player ID

The point packet has 6th byte only when Z=1 or M=1. Otherwise, the point packet has 5 bytes only. If Z=1, this byte is pressure value. If M=1, this byte is player ID.

1.2 Interface

RS232 Interface

If RS232 controller is used, please specify the following information in the driver programs:

- Baud rate：9600 bps.
- Data bits：8
- Stop bit：1
- Parity check：NONE.

1.3 Packet Parser Sample Code

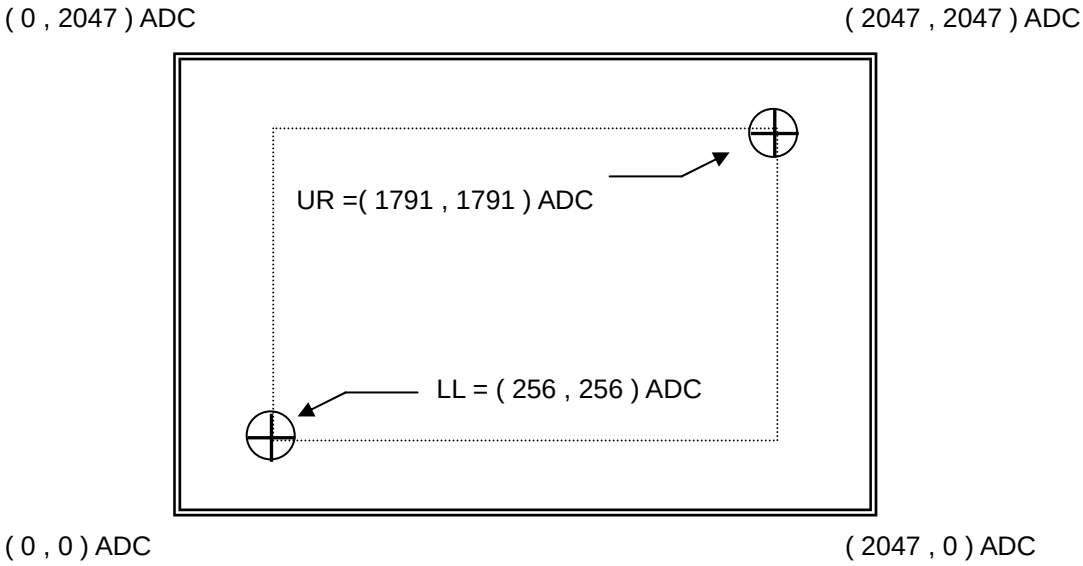
```
#define MAX_BUFFER                  1024
#define MOUSE_PACKET_LEN        5
#define MAX_CMD_LEN                 16
#define POLLING_BUFFER_SIZE             3

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 ) )
                {
                        return 100;
```

```
}
// read packet from COM port or USB port
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;
                }
                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,
                        // pPort->DisPatchMessage( pMsgBuffer, dwCnts );
                        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,
                        //pPort->DisPatchMessage( pMsgBuffer, dwCnts );
                        dwCnts = 0;
                        bPointPacket = FALSE;
                    }
                    continue;
                }
            }
        }
    }
}
```

1.4 2 Points Calibration for Position Decoding

System software developer can develop their own simple calibration tool based on below sample. However, Touchkit *Saturn Resistive* and *ESC7000* Capacitive controller also support advanced 4, 9, and 25 points calibration. Please reference to the document "*EETI Calibration Design Guide*".



1. LL and UR are the calibration points of touch panel, the points are setup at
   LL = (1/8 screen X, 1/8 screen Y) = (256 , 256) ADC ;
   UR = (7/8 screen X, 7/8 screen Y) = (1791 , 1791) ADC
2. When we do the calibration, press on these two points, then we get the row data LL 'and UR':
   LL' = (LL_X, LL_Y) ; UR' = (UR_X, UR_Y)
3. After the calibration, when you touch the panel and get another row data X and Y. The new position after calibration are X' and Y' , and the conversion formulas are as follows:

$$X' = \frac{X - LL_X}{UR_X - LL_X} * 1536 + 256$$

$$Y' = \frac{Y - LL_Y}{UR_Y - LL_Y} * 1536 + 256$$