

TouchKit
Software Programming Guide
Version 1.2

www.densitron.com

10400-4 Pioneer Blvd.
Santa Fe Springs, Ca 90670

TEL: (562) 941-5000 ext. 252
FAX: (562) 941-5757

CONTENTS

1. Introduction.....	Pg.3
2. Programming Guide of Using Touch Kit Controller	
Board	Pg.4
2.1 Packets Format.....	Pg.4
2.1.1 Diagnostics Packet.....	Pg.4
2.1.2 Report Packet	Pg.5
2.2 Communication Interface	Pg.5
2.2.1 RS232 Interface	Pg.5
2.2.2 PS/2 Interface	Pg.5
2.2.3 USB Interface	Pg.6
2.2.4 IIC Interface	Pg.7
2.3 Packet Parser Sample Code	Pg.8
2.4 2 Points Calibration for Position Decoding	Pg.11

1. Introduction

Densitron 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 I²C. In recent years, portable devices become popular, and I²C transaction is the best way to communicate with these portable devices, like PDA, eBook, Mira, etc.

Densitron'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. The data command packet structure will be described in **chapter 1**. Then special notices of programming RS232, PS/2, USB port and I²C are expressed. At the end, the sample code of parsing the packet from controller and the two points calibration/ alignment conversion formulas are listed.

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

2. Programming Guide of Using Touch Kit Controller Board

2.1 Packets Format

All Touch Kit controllers including RS232, PS2 and USB for 4-wire, 5-wire and 8-wire use the same packet format. And the packets can be classified into 2 groups: Diagnostics Packet and Report Packet.

2.1.1 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	Length In Byte	Command	Response
1 Byte	1 Byte	1 Byte	(Length In Byte-1) 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	1	“D”	Response
------	---	-----	----------

The response is an ASCII string, such as “0.99”

3. Get type

This packet is to request the controller type. The possible responses are 4 wire or wire.

Host Issues

0x0A	1	“E”
------	---	-----

Controller firmware responds

0x0A	Length	“E”	Response
------	--------	-----	----------

2.1.2 Report Packet

Each report packet contains 5 bytes. The packet format is as follows:

	MSB						LSB	
Byte 1	1	0	0	0	0	0	0	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

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.

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

2.2 Communication Interface

2.2.1 RS232 Interface

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

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

2.2.2 PS/2 Mouse Interface

If PS/2 mouse interface controller is used, please follow PS/2 mouse specification for standard PS/2 mouse command sets for BIOS plug and play. Then, follow Touch kit controller packet format to communicate with controller for touch screen features.

2.2.3 USB Interface

If USB controller is used, please notice that the board firmware is designed with standard "Vendor Request Command."

- USB firmware VID and PID may be one of
 1. VID = 0123, PID = 0001
 2. VID = 0EEF, PID = 0001
 3. VID = 0EEF, PID = 0002
- Maximum FIFO size is 8 bytes.
- Two end points are used
 1. Control pipe: for standard USB PnP and writing packets to controller device controller.
 2. Interrupt pipe: for reading packet from controller device.
- It needs to wait for 3 ms at least to issue another write command after one write command is issued.
- The polling interval of reading with interrupt pipe is 5 ms.
- Control Write Urb format is as follows:

```
/*=====*/
/* Vendor specific request Urb format for Touch Panel controller kit          */
/* with Win2000 DDK                                                         */
/*=====*/
```

```
UsbBuildVendorRequest(    pWriteUrb,          //IN PURB Urb,
                          URB_FUNCTION_VENDOR_DEVICE , //IN USHORT Function,
                          sizeof( struct _URB_CONTROL_VENDOR_OR_CLASS_REQUEST ), //IN USHORT Length,
                          0, //IN ULONG TransferFlags,
                          0, //IN UCHAR ReservedBits,
                          0, //IN UCHAR Request,
                          0, //IN USHORT Value,
                          0, //IN USHORT Index,
                          pTxBuffer, //IN PVOID TransferBuffer OPTIONAL,
                          NULL, //IN PMDL TransferBufferMDL OPTIONAL,
                          ulBytesToSend, //IN ULONG TransferBufferLength,
                          NULL //IN PURB Link OPTIONAL,
                          );
```


2.3 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;
    }
}

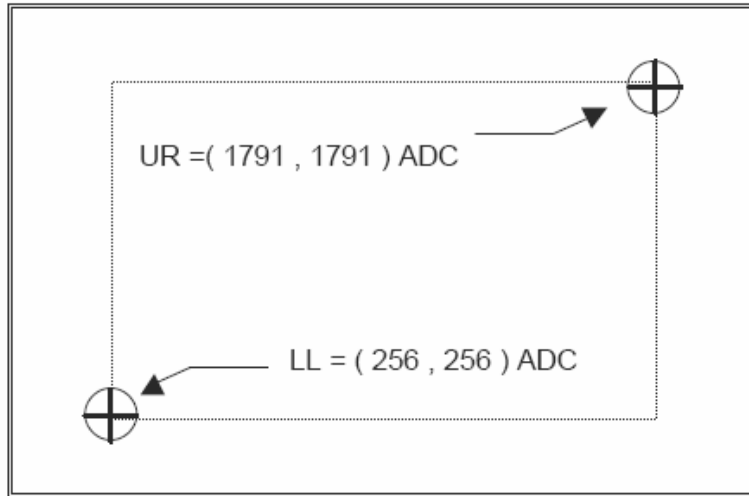
```

```
        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;
}
}
}
}
}
```

2.1 2 Points Calibration for Position Decoding

(0 , 2047) ADC

(2047 , 2047) ADC



(0 , 0) ADC

(2047 , 0) ADC

1. LL and UR are the calibration target points of touch panel, the points are setup at:
 $LL = (1/8 \text{ screen } X, 1/8 \text{ screen } Y) = (256, 256) \text{ ADC};$
 $UR = (7/8 \text{ screen } X, 7/8 \text{ screen } Y) = (1791, 1791) \text{ ADC}$
2. During calibration, press on these two target points, the raw data are obtained as:
 LL' and UR' ;
 $LL' = (LL_X, LL_Y);$
 $UR' = (UR_X, UR_Y)$
3. After the calibration, whenever the panel was touched, firmware report the raw data X and Y . Then, the calibrated position X' and Y' are calculated with the formulation as follows:

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

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