Delcom Products Inc. Application Note220

Revision 0 – 11/04/2008

Title:

Migrating to Delcom USB HID Chip Set (Generation 2) from the Delcom USB custom driver Chip Set (Generation 1).

Abstract:

This document outlines the software changes that a user of generation 1 products using the direct communication method must take when migrating to generation 2 (USB HID) products.

Definitions:

Generation 2 – All Delcom products that use the USB HID generic driver.

Generation 1 – All Delcom products that use the USB Delcom custom driver.

Direct Communication – Software connects to the USB device via the USB Driver.

Indirect Communication – Software connects to the USB device via the Delcom DLL.

VID – Vendor ID Number – 0x0FC5

PID – Product ID Number – 0xB080

Related Part Numbers:

All part numbers that start with 90xxxx.

Document Revision:

Revision 0 – 11/04/2008

Document identifier:

Application Note 220

Location:

http://www.delcomproducts.com/downloads/appnote220.pdf

Table of Contents

1 Introduction

In November 2008 we released our second generation of the USB firmware. This second generation firmware is similar to our original (first generation) products in form and function but instead of communicating via the Delcom driver this firmware uses the USB HID driver. Using the USB HID driver has many advantages, see USB HID Driver Advantages below.

Unfortunately there are a few changes that need to be made when converting first generation code to second generation code. If you are communicating via the Delcom DLL driver, then no changes are necessary. Just make sure you are using version 0.7 or greater of the DLL. If you're not currently using the Delcom DLL than you will have to modified your code as described in this document or modify your code to use the Delcom DLL. This application note describes the changes require to communicate directly to the USB device via the USB HID driver, also referred to as 'Direct Communications'. If you plan on convert your code to use the Delcom DLL please see the DLL documentation. Communicating via the Delcom DLL is also referred to as 'Indirect Communications'.

The simplest communication method is to use the Delcom DLL.

2 Overview of required modification for direct communications

There are three areas that will need to be modified, there are:

- Device name – The method that retrieves the device name must be changed.

- Write Command – The write command/data packet changes slightly and the major command number change.

- Read Commands – The reads command packet changes slightly, the minor command is used instead of the major command as the read command. Also read commands that pass more than one byte must also be changed.

3 USB HID Driver Advantages

Overall the USB HID driver is more reliable, more secure, scalable and provides for a true plug and play USB device.

- Driver preinstalled on most operation systems.
- Compatible with 32bit and 64bit machines.
- Multi-core and multi-processor safe.
- Better security. Driver signed and certified by the OS vendor.
- Automatic driver updates by OS vendor.
- Larger read and write packet buffer size.
- More efficient data payload.
- Data verification feature for improved mission critical designs.
- Enhance error detection and handling.
- Scalable to higher performance USB chip sets.

4 Software changes

The following sections describe the require changes to migrate generation1 code to generation 2 (USB HID) code. The following examples are shown in C and are for Microsoft Windows. Other operating systems have similar functions. This application note should be used in conjunction with the example code on the web at:

C++ USB HID Example:
http://www.delcomproducts.com/productdetails.asp?productnum=890607

C#.NET USB HID Example:
http://www.delcomproducts.com/productdetails.asp?productnum=890630

VB.NET USB HID Example: VB.NET user should use the C#.NET example. Following the same instruction as the C#.Net example code. Note VB.NET and C#.NET modules can be mixed in the same project.

All USB HID documentation and examples can be found at:
http://www.delcomproducts.com/productdetails.asp?productnum=900000

Open and closing the device is the same as before, use CreateFile() and CloseFile().

```
hDevice=CreateFile(DeviceName, GENERIC_READ|GENERIC_WRITE, 0, NULL,
        OPEN_EXISTING, 0, NULL);      // Open the device
CloseHandle(hDevice);                 // Close the device
```

4.1 Finding the Device Name

In order to communicate to the USB HID device you must first acquire the device name. To do this we will ask the systems API for all USB devices that match the unique GUID given to all USB HID device. For windows you must include the HID library and header file in your project, they are: hid.lib, hidpi.h, hidsdi, hidusage.h.

For DotNet programs just include the Delcom HID code in your project.

The function we will be using are:

```
GetHidGuid()
SetupDiGetClassDevs()
SetupDiEnumDeviceInterfaces ()
SetupDiGetDeviceInterfaceDetail()
CreateFile()
HidD_GetAttributes()
CloseFile() or CloseHandle()
```

The Unique USB HID GUID can be acquired with the `HidD_GetHidGuid()` functions or hardcoded to `{4D1E55B2-F16F-11CF-88CB-001111000030}.`

Using the unique GUID we use the above functions to return all of the matching USB HID devices presently found on the system. One by one we need to test these devices until we find the correct device of interest. The correct device will have VID number of 0x0FC5 and PID number of 0xB080. Also at this time we can optionally check for the device family type number and/or the unique serial number in the device (see Appendix B). Once we have confirmed that we have the correct device we note the device handle and optional save the device name.

For those of you that based your code on the C++ example, the changes are minimal. There is just one added step of asking the system API for the VID and PID information.

The following C function scans for the USB HID device and optionally tests for the family type number and serial number. If the device is found it copies the device handle to the global variable hDevice, saves the device name, and leaves the file open.

```
#define USB_VID 0x0FC5 // USB Vendor ID (Always 0x0FC5 for Delcom products)
#define USB_PID 0xB080 // USB Product ID (Always 0xB080 for Delcom HID device)
#define USB_TID 0x0001 // USB Type ID (0=all, 1=USBHIDIO, 2=USBHIDVI,…)
#define USB_SID 0x0000 // USB Serial ID (zero=scan for all)
HANDLE  hDevice;              // Handle to the device
char   DeviceName[512];  // Devicename string

// ------------------------------------------------------ //
// ScanForHidDevice(VID,PID,TID,SID) - Scan thru all the HID device lookking
// for a match on the VID, PID and optional TID (Type ID) and SID(SerialNum)
// Sets the hDevice ghandle varible if found and opens the device
// Return zero if found, else non-zero error code.
// 0 = Success
// 1 = No matching HID devices
// ------------------------------------------------------ //
unsigned int ScanForHIDDevice(unsigned int VID, unsigned int PID, unsigned int TID,
unsigned int SID )
{
        //Use a series of API calls to find a HID with a matching Vendor,Product,
Type and Serial ID.
        DelcomDeviceInfoStruct        DelcomInfo;
        PSP_DEVICE_INTERFACE_DETAIL_DATA      detailData;
        GUID                                          HidGuid;
        HANDLE                                        hDevInfo;
        ULONG                                         Required;
        HIDD_ATTRIBUTES                               Attributes;
        SP_DEVICE_INTERFACE_DATA                      devInfoData;
        bool                                          LastDevice = FALSE;
        int                                           MemberIndex = 0;
        bool                                          MyDeviceDetected = FALSE;
        LONG                                          Result;
        ULONG                                         Length;

        // Variable init
        Length = 0;
        detailData = NULL;
        hDevice=NULL;
        MemberIndex = 0;
        LastDevice = FALSE;

        HidD_GetHidGuid(&HidGuid);            // Get the GUID for all system HIDs.

        hDevInfo=SetupDiGetClassDevs(&HidGuid, NULL, NULL, DIGCF_PRESENT |
DIGCF_INTERFACEDEVICE);
        devInfoData.cbSize = sizeof(devInfoData);
        do      {
        MyDeviceDetected=FALSE;
        Result=SetupDiEnumDeviceInterfaces(hDevInfo, 0, &HidGuid, MemberIndex,
&devInfoData);
        if (Result != 0)
        {       //A device has been detected, so get more info
        Result = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData, NULL, 0, 
&Length, NULL);  // zero length call to get size
        detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);
        detailData -> cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

        Result = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData, detailData, 
Length, &Required, NULL);

        hDevice=CreateFile(detailData->DevicePath, GENERIC_READ|GENERIC_WRITE, 0, 

        (LPSECURITY_ATTRIBUTES)NULL, OPEN_EXISTING, 0, NULL);

        Attributes.Size = sizeof(Attributes);
        Result = HidD_GetAttributes(hDevice, &Attributes);
        MyDeviceDetected = FALSE;
        if( (Attributes.VendorID == VID) && (Attributes.ProductID == PID))
        {
        MyDeviceDetected = TRUE;
strncpy(DeviceName, detailData->DevicePath, 512);  // save the devicename
/* optional check for Family id and serial number, See Appendix B
        if(TID || SID) {       // Now check for TID and SID if non-zero
        if(GetDeviceInfo(&DelcomInfo)) MyDeviceDetected = FALSE;
        else {
                if(TID && (DelcomInfo.Family != TID)) MyDeviceDetected = FALSE;
                if(SID && (DelcomInfo.Serial != SID)) MyDeviceDetected = FALSE;
                }
          }     // end of TID or SID
*/
   }
Else {//The PID and/or VID doesn't match. Close the device try the next one
        CloseHandle(hDevice);
        hDevice = 0;
        }
free(detailData);
}  //if (Result != 0)
else    {       // End of List - No HID devices detected!
        LastDevice=TRUE;      // returned 0, so there are no more devices to check.
}
//If we haven't found the device yet,  then try the next one.
MemberIndex++;
} // loop till either end of deivce list or we find our device
while ((LastDevice == FALSE) && (MyDeviceDetected == FALSE));

SetupDiDestroyDeviceInfoList(hDevInfo);             //Free the memory
if (MyDeviceDetected == FALSE) {
        // Device not found
        hDevice = 0;
        return(1);
        }
else    {
        // Device Found
        HidD_SetNumInputBuffers(hDevice,1);
        return(0); // Success
 }
}
```

Once the correct device has been opened you can start write and read to the device using the handle to the device (hDevice).

Getting the device name is bit labor intensive as an alterative you could just use the Delcom DLL to get the device name and then continue with the write and read functions below. Or you could can the device name from one of our example programs and cut and paste it in to your code for quick test. Note that device names are dynamic and change when ever the device re-plugged in to the machine.

4.2 Write Command/Data Packet

Sending a write command in generation 2 products is very similar to generation1. There are three minor changes here.

First the MajorCmd number changes, in generation 1 products, the write major command was always 10. Now the major command is either 101 or 102. Major command 101 sends only the first 8 bytes of the write packet. Major command 102 sends all 16 bytes of the write packet. Use command 102 if you using the DataExt area of the write packet. If in doubt, use command 102, you can always send 16 bytes weather the command requires it on not. Note the length parameter in the API function should also match the 101 or 102 command. Again, when in doubt set the length parameter to 16 and use command 102.

Secondly the packet data structure changes slightly.

And lastly the API function to call is different.

The write command/data packet and API for generation 1 (Delcom driver) was:

```
// GEN1 USB Write Command/Data Packet
Typedef struct {
               unsigned char Recipient;
               unsigned char DeviceModel;
               unsigned char MajorCmd;
               unsigned char MinorCmd;
               unsigned char DataLSB;
               unsigned char DataMSB;
               unsigned short Length;
               unsigned char DataExt[8];
        } PacketStruct,*pPacketStruct;

// GEN1 API function
BOOLEAN DeviceIoControl(hDevice, IOCTL_USBIO_SEND_PACKET,
              pTxPacket, 8+pPacket->Length, pRxPacket, 8, &nBytes, NULL );
```

The new write command/data packet and API for generation 2 (USB HID) is:

```
// GEN2 USB Write Command/Data Packet
typedef struct _HIDPacketStruct {
        unsigned char MajorCmd;
        unsigned char MinorCmd;
        unsigned char DataLSB;
        unsigned char DataMSB;
        unsigned char DataHID[4];
        unsigned char DataExt[8];
        } HIDPacketStruct, *pHIDPacketStruct;

// GEN2 API function
BOOLEAN HidD_SetFeature(hDevice, pPacket, Length);
```

As you can see the packet are very similar. All you need to do is re-declare your packet structure, use the new 101 or 102 major command numbers and call the new API function.

For an example to set port0 to a value of 0xFF, we would use the 101-1 command. In this example we are going to change the data structure slightly because it will make the read commands easier later on. This packet structure is the recommend one to use.

```
// GEN2 Example to set port0 & port1 to 0xFF.
typedef union HIDPacketStruct {
        unsigned char Data[256];
        struct {
                unsigned char MajorCmd;
                unsigned char MinorCmd;
                unsigned char DataLSB;
                unsigned char DataMSB;
                unsigned char DataHID[4];
                unsigned char DataExt[8];
                } Tx;
        struct {
                unsigned char Cmd;
                } Rx;
        } HIDPacketStruct, *pHIDPacketStruct;

HIDPacketstruct MyPacket;             // Declare the packet
MyPacket.Tx.MajorCmd = 101;    // Fill the packet
MyPacket.Tx.MinorCmd = 1;
MyPacket.Tx.DataLSB = 0xFF;
HidD_SetFeature(hDevice,MyPacket, 8) // lastly send the packet
```

4.3 Read Command/Data Packet

There are four changes in sending a read command in generation 2 products.

First the MajorCmd number changes, in generation 1 products the read major command was always 11. In generation 2 products there is a limitation of only being able to send one data byte with the read command. Therefore we use the minor command as the new read command number. There is one exception and that is for the read ports command (11-0). This command is now 100 because the USB HID protocol does not allow zero to be used as command number.

Secondly the packet data structure changes.

Thirdly the API function to call is different.

Lastly, because of the one data byte restriction, any generation 1 read commands that past more than one byte of data must now be prefixed with a setup command. The following table shows what prefix or setup write command must be sent before the read command is sent.

Generation 1 Read Commands	Generation 2 Read & Setup Commands
11-1 Read port0 with Strobe	Setup Command: 101-3 Read Command: 1
11-2 Read port1 with Strobe	Setup Command: 101-3 Read Command: 2
11-18 Write 2 bytes, Read 8 Bytes	Setup Command: 101-8 Read Command: 1
11-60 Read from I2C port	Setup Command: 101-63 Read Command: 60
11-61 Selective read from I2C Port	Setup Command: 101-63 Read Command: 61
11-91 SPI Write 1Byte, read 1-64bits	Setup Command: 101-92 Read Command: 91

A generation 2 example on reading the port values of the device.

```
// GEN2 USB Write/Read Command/Data Packet
typedef union HIDPacketStruct {
        unsigned char Data[256];
        struct {
                unsigned char MajorCmd;
                unsigned char MinorCmd;
                unsigned char DataLSB;
                unsigned char DataMSB;
                unsigned char DataHID[4];
                unsigned char DataExt[8];
                } Tx;
        struct {
                unsigned char Cmd;
                } Rx;
        } HIDPacketStruct, *pHIDPacketStruct;

HIDPacketstruct MyPacket;            // Declare the packet
MyPacket.Rx.Cmd = 100;               // Fill the packet – read ports cmd
HidD_GetFeature(hDevice,MyPacket, 8) // lastly send the packet

unsigned char Port0 = MyPacket Data[0];
unsigned char Port2 = MyPacket Data[1];
```

A generation 2 example where more than one byte is needed. This example reads port0 with strobe on port1 pin 0.

```
// GEN2 USB Write/Read Command/Data Packet
typedef union HIDPacketStruct {
        unsigned char Data[256];
        struct {
                unsigned char MajorCmd;
                unsigned char MinorCmd;
                unsigned char DataLSB;
                unsigned char DataMSB;
                unsigned char DataHID[4];
                unsigned char DataExt[8];
                } Tx;
        struct {
                unsigned char Cmd;
                } Rx;
        } HIDPacketStruct, *pHIDPacketStruct;

HIDPacketstruct MyPacket;            // Declare the packet
MyPacket.Tx.MajorCmd = 101;          // Fill the tx setup packet
MyPacket.Tx.MinorCmd = 3;            // Setup the strobe
MyPacket.Tx.DataLSB = 0x01;          // Pin0
HidD_SetFeature(hDevice,MyPacket, 8) // send the setup or prefix packet cmd

// Now send the read command
MyPacket.Rx.Cmd = 100;               // Fill the rx packet – read ports cmd
HidD_GetFeature(hDevice,MyPacket, 8) // lastly send the packet

unsigned char Port0 = MyPacket Data[0];      // Get the data
unsigned char Port2 = MyPacket Data[1];
```

5 References

Delcom USB HID Documentation and Examples

http://www.delcomproducts.com/productdetails.asp?productnum=900000

Delcom DLL

http://www.delcomproducts.com/productdetails.asp?productnum=890510

Delcom USB HID Chips

http://www.delcomproducts.com/products_USBIO.asp

Delcom Distribution Disk

http://www.delcomproducts.com/delcomcd/Start.htm

Appendix A. Revision History

Rev	Date	Author	Description
0	11/04/2008	Doug Lovett	Initial Release

Appendix B. Optional TID and SID Test

If you plan on also testing for the family type and/or serial number add the following code to your product. And uncomment the ‘optional check for Family id and serial number’ code.

```
// Family Type Values
enum FamilyType{
        ALL,                      // all Delcom USB device
        USBIO,                    // all Delcom USB IO Chips & foot switch
        USBVI,                    // all Delcom USB Visual Indicators
        USBND                     // all Delcom USB Numeric Displays
};

// DataStruct used by the GetDeviceInfo functions
typedef struct DelcomDeviceInfoStruct_ {
                unsigned short int Family;
                unsigned short int Security;
                unsigned char Version;
                unsigned char Day;
                unsigned char Month;
                unsigned char Year;
                unsigned int Serial;
                unsigned int Spare;
                } DelcomDeviceInfoStruct, *pDelcomDeviceInfoStruct;

// ----------------------------------------------------------------- //
// Reads device info
// Returns zero on success, else non-zero
// Return data in a 16byte data buffer. Buffer must be predeclared
// ----------------------------------------------------------------- //
int GetDeviceInfo(pDelcomDeviceInfoStruct pInfo)
{
        myPacket.Rx.Cmd = 104;

        if(!HidD_GetFeature(hDevice,&myPacket, 16)) {

                return(1);        // command failed
                }

        // now get the data if the variable has been passed
        if(!pInfo) return(1);
        memcpy(pInfo,&myPacket,16);
        return 0;
}
```

Appendix C. Notices

DELCOM PRODUCTS INC. takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on DELCOM PRODUCTS INC procedures with respect to rights in DELCOM PRODUCTS INC. specifications can be found at the DELCOM PRODUCTS INC. website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers' or users of this specification, can be obtained from the DELCOM PRODUCTS INC. Executive Director.

DELCOM PRODUCTS INC. invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the DELCOM PRODUCTS INC. Executive Director.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to DELCOM PRODUCTS INC., except as needed for the purpose of developing DELCOM PRODUCTS INC. specifications, in which case the procedures for copyrights defined in the DELCOM PRODUCTS INC. Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by DELCOM PRODUCTS INC. or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and DELCOM PRODUCTS INC. DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

LIFE SUPPORT POLICY - Delcom Products are not authorized for use in life support devices and/or systems without the express written approval of Delcom.