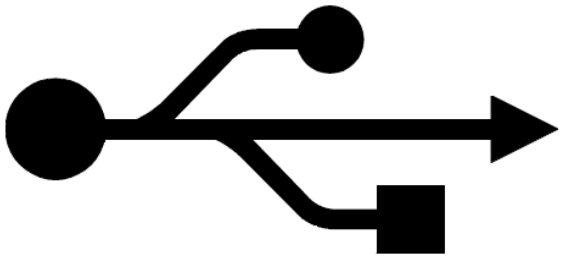

PIC Communication with USB

By Roy Seifert

This page intentionally left blank

Revised 11/28/2009

Table of Contents

This page intentionally left blank

Introduction

The purpose of this article is to explain how to interface a PIC microcontroller to a PC via the USB port. Although the concepts are universal, the examples are specifically for use with mikroElektronika's MikroBASIC Pro for PIC.

My journey into PIC microcontrollers and USB communication began with my interest in Microsoft® Flight Simulator 2004. I am an instrument-rated private pilot, but due to medical reasons, I can no longer fly. FS 2004 gives a very good simulation of instrument flight. There are some things it won't do, nor allow me to do, but overall it satisfies my urge to fly in an instrument environment. Plus, it allows me to fly aircraft that in real life I couldn't afford to really learn how to fly. My wonderfully patient wife thinks it's a dumb game because she can't ever see anything happening, but I know I'm cruising at 24,000 feet at 180 knots in pressurized comfort and in total control!

I already had a flight yoke, throttle quadrant and rudder pedals, but I wanted to increase the realism of flight simulator by using real physical switches and knobs instead of mouse clicks to activate avionics and cockpit functions. I purchased the *Super Rotary Encoder* from Desktop Aviator which allowed me to interface 16 rotary or 32 push-button switches and up to 8 axes. By doing some fancy programming and installing a program called FSUIPC I was able to interface with over 100 push-button switches and 10 rotary switches. Unfortunately, I needed to add one more rotary switch, but I was out of room.

The *Super Rotary Encoder* was built with a PIC18F2450 microcontroller. I have a background in computers and programming, so I figured I could build my own switch interface, and so my journey began.

Hardware

Usually the first place I look for inexpensive goods is Ebay®. I found a large number of PIC programmers, but I felt I needed more. Searching the internet I found mikroElektronika, a company in Serbia, that produces both the hardware and software that I needed.

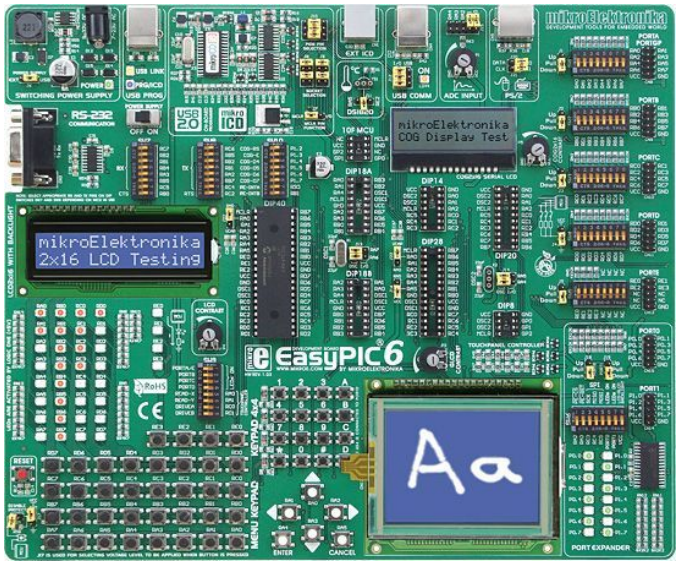

Figure 1: EasyPIC6 Development Board

Development Board

The EasyPIC6 development board they manufacture allows me to program many different PIC microcontrollers, and contains the circuits to develop and test LEDs, LCD graphical and character displays, switches, one analogue to digital converter (ADC) and various interfaces. It also contains a port expander and access to the microcontroller's ports via header pins.

The software that comes with the board contains examples designed for the PIC16F887 microcontroller, and the board comes with one of those microcontrollers. What a great tool for developing different applications! In fact, I purchased an additional Serial 7-Segment Display board for a particular application I have in mind. The EasyPIC6 did not come with the stand-alone 2x16 LCD display, the graphic LCD 128x64 with touch screen, nor the one-wire temperature sensor, but I was able to purchase those from other suppliers.

Software

MikroElektronika allows you to download trial versions of each of their compilers. Assembly language is a bit cumbersome, I know nothing of Pascal, I know some C language, but I am very fluent in BASIC, so I decided to purchase their MikroBASIC Pro compiler for PIC. Prior to purchasing the full license, I used the trial version with the included examples to test my USB connections.

Personal Thanks

Before going any farther, I want to extend my thanks to the mikroElektronika (ME) team for their excellent products and support. Their hardware, software, libraries and examples made my development work almost effortless. Sure I had a pretty steep learning curve, but the tools provided by ME made learning how to program a microcontroller pretty easy. The ME team put a lot of hard work into both the hardware and software, and provide the results at a very reasonable price.

USB Communication

The most difficult part of this project was learning exactly what was required to get the PIC microcontroller to communicate with the USB port. After about two weeks of trial and error and frustration, I finally got it to work. The two most important things that absolutely have to be correct are the microcontroller configuration, and the USB device descriptor. If even the smallest thing is incorrect about either of these, communication will not occur.

Microcontroller Configuration

Even before programming the microcontroller, its configuration must be correct. In mikroBASIC Pro this is accomplished by editing the project.

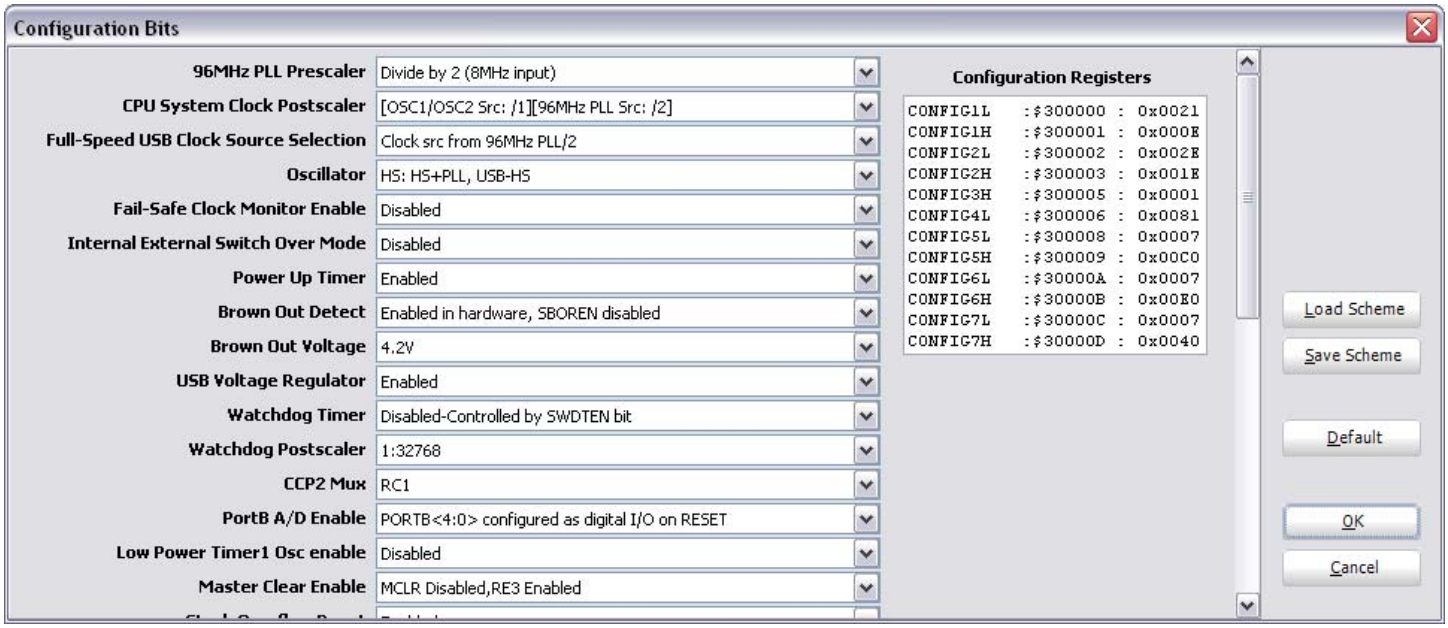

Figure 2: Configuration Bits

On the mikroBASIC Pro menu click on Project, Edit Project, or press Ctrl + Shift + E to open the Configuration Bits dialog box as shown in Figure 2 above. Each configuration choice is selected with a drop-down box. **You need to read the data sheet for the particular device you plan to use in order to determine what to select.** Since the EasyPIC6 development board comes with an 8 MHz external crystal, 8 MHz was selected for the 96 MHz PLL Prescaler. **For USB operations, the crystal must be a multiple of 4 MHz, e.g. 4 MHz, 8 MHz, 12 MHz, etc. Again, refer to the data sheet to determine the correct value for the crystal.**

The configuration can be changed to meet your particular needs; e.g. I changed PORTB to all digital and disabled MCLR. The lines that seem to affect USB communication the most are the first four, and the tenth line, USB Voltage Regulator.

You don't have to remember this setup because this configuration has been saved for you by mikroElektronika in C:\Program Files\Mikroelektronika\mikroBasic PRO for PIC\Schemes. All you have to do is load the appropriate scheme by using the buttons on the right of this dialog box. Once you make any changes, you can also save your scheme so it can be loaded into another microcontroller.

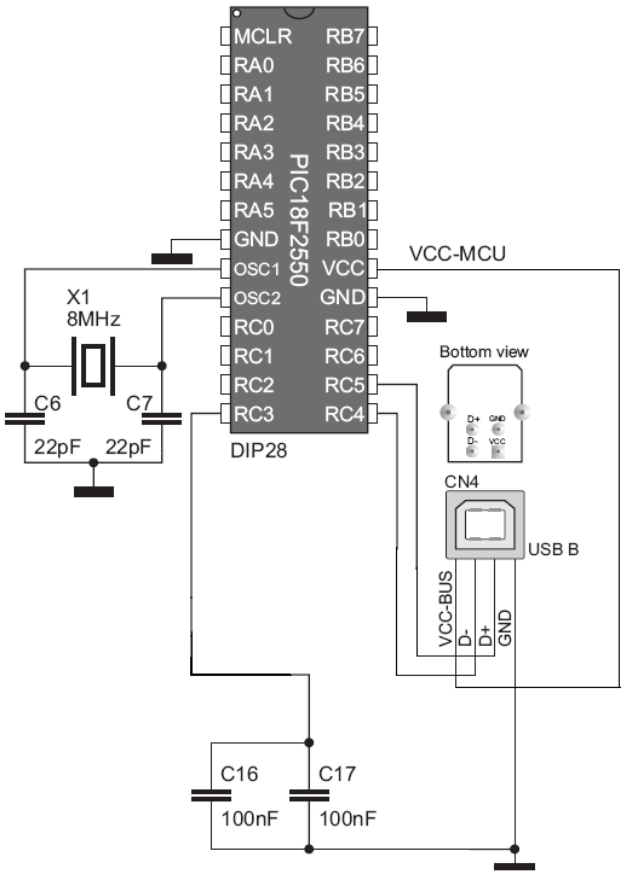

Figure 3: 28-Pin Microcontroller Hardware Configuration

With the above configuration bits selected, the microcontroller hardware is configured as shown in Figure 3. Any changes to lines 1 – 4 or line 10 would require a corresponding change in the hardware configuration. Refer to the appropriate data sheet.

USB Descriptor

This was probably the most difficult for me to get right. The USB descriptor is used during the enumeration process to identify the peripheral device to the host, i.e. identify the microcontroller to the PC. This has to be correct, otherwise your microcontroller will not be recognized and you will get a device error message.

A Few Words about Descriptors

What we are building with the microcontrollers are human interface devices (HID) which require a specific type of descriptor. I did a lot of research on the Internet regarding USB HID descriptors. I've listed those references in the reference section and will list them again here:

- USB Implementers Forum, Inc. www.usb.org – This is where you can find the USB interface specifications and everything else you ever wanted to know about USB. The most useful thing I found here was the HID Descriptor Tool which allows you to build the report descriptor. It also comes with examples of different types of report descriptors. You can use these report descriptors instead of the generic report descriptor that the mikroBASIC Pro descriptor tool built. When using this tool you must first save the descriptor as a .h header file, then convert it to Unicode text.

  .h header example:

```
0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
0x15, 0x00,                    // LOGICAL_MINIMUM (0)
0x09, 0x04,                    // USAGE (Joystick)
0xa1, 0x01,                    // COLLECTION (Application)
```

  Unicode conversion:

```
0x05, 0,                       // USAGE_PAGE (Generic Desktop)
0x01, 0,
0x15, 0,                       // LOGICAL_MINIMUM (0)
0x00, 0,
0x09, 0,                       // USAGE (Joystick)
0x04, 0,
0xa1, 0,                       // COLLECTION (Application)
0x01, 0,
```

  …and so on. Be sure when you get to a `0xC0` end of collection code, you add the `, 0,` to it so it becomes

```
0xC0, 0,
```

- Amr Bekhit, http://www.helmpcb.com/Electronics/USBJoystick/USBJoystick.aspx - This article explains how to convert an old joystick that used a joystick port to one that uses a USB port using a microcontroller. **This article provided my first breakthrough on creating a valid joystick HID descriptor.**
- USB Made Simple, http://www.usbmadesimple.co.uk/index.html - A series of articles on how USB works. The descriptor explanation was outstanding.
- USB in a Nutshell, http://www.beyondlogic.org/usbnutshell/ - Great article on how USB works, especially good explanation of the descriptor.

mikroBASIC Pro Descriptor Tool

mikroBASIC Pro comes with an HID descriptor tool built into the HID Terminal tool. To build a generic descriptor, on the mikroBASIC Pro menu click on Tools, HID Terminal, then click on the Descriptor tab.

Figure 4: mikroBasic PRO Descriptor Tool

The parts I found helpful about this tool were being able to enter the vendor ID (VID) hex value, product ID (PID) hex value, and the Vendor Name and Product Name strings as shown in Figure 4. Everything else I left alone, or changed in the descriptor itself. Technically, the vendor ID is assigned by the USB organization for the measly sum of $2,000! You can pretty much put anything here.

Modifying the Descriptor

I specifically wanted to create a descriptor that emulated a joystick with 5 axes and 32 buttons. 32 is the maximum that Microsoft allows on any one joystick, and I planned to use 5 of the analogue inputs on the PIC18F2455 microcontroller for axes. Following is the descriptor that I created along with an explanation of the various parts. Anything you see in red are changes I made.

```
module DRA_desc
' uses HIDconstant;
'  File Version 1.01
'
'****************************************
 const Serial_No = "DRA00001"
'****************************************
' The number of bytes in each report,
' calculated from Report Size and Report Count in the
report descriptor
'****************************************
 const HID_INPUT_REPORT_BYTES      = 1
 const HID_OUTPUT_REPORT_BYTES     = 1

 const HID_FEATURE_REPORT_BYTES    = 2
'****************************************
' Byte constants
'****************************************
```

I added this serial number constant so I wouldn't have to manually change each character in the string table.

These two values were created by the ME descriptor tool

```
 const NUM_ENDPOINTS               = 2
 const ConfigDescr_wTotalLength    =
USB_CONFIG_DESCRIPTOR_LEN + USB_INTERF_DESCRIPTOR_LEN +
USB_HID_DESCRIPTOR_LEN + (NUM_ENDPOINTS *
USB_ENDP_DESCRIPTOR_LEN)
 const HID_ReportDesc_len          = 68

 const Low_HID_ReportDesc_len      = HID_ReportDesc_len
 const High_HID_ReportDesc_len     = HID_ReportDesc_len >>
8

 const Low_HID_PACKET_SIZE         = HID_PACKET_SIZE
 const High_HID_PACKET_SIZE        = HID_PACKET_SIZE >> 8

'*************************************
' Descriptor Tables
'*************************************
 const DescTables as byte[(18+9+9+9+7+7+68)*2] = (

' Device Descriptor
    USB_DEVICE_DESCRIPTOR_LEN, 0,           ' bLength
- Length of Device descriptor (always 0x12)
    USB_DEVICE_DESCRIPTOR_TYPE, 0,          '
bDescriptorType       - 1 = DEVICE descriptor
    0x00, 0,                                ' bcdUSB
- USB revision 2.00 (low byte)
    0x02, 0,                                '
(high byte)
    0x00, 0,                                ' bDeviceClass
- Zero means each interface operates independently (class
code in the interface descriptor)
    0x00, 0,                                '
bDeviceSubClass
    0x00, 0,                                '
bDeviceProtocol
    EP0_PACKET_SIZE, 0,                     '
bMaxPacketSize0       - maximum size of a data packet for a
control transfer over EP0
    0x00, 0,           ' idVendor  - Vendor  ID (low byte)
    0x8B, 0,           '                        (high byte)
    0xCB, 0,           ' idProduct - Product ID (low byte)
    0x27, 0,           '                        (high byte)
    0x21, 0,           ' bcdDevice            - ( low byte)
    0x01, 0,           '                        (high byte)
    0x01, 0,           ' iManufacturer - String1
    0x02, 0,           ' iProduct      - String2
    0x03, 0,           ' iSerialNumber - String3
    0x01, 0,           ' bNumConfigurations    - 1

' Configuration Descriptor
    USB_CONFIG_DESCRIPTOR_LEN, 0,           ' bLength
- Length of Configuration descriptor (always 0x09)
    USB_CONFIG_DESCRIPTOR_TYPE, 0,          '
bDescriptorType       - 2 = CONFIGURATION descriptor
    ConfigDescr_wTotalLength, 0,            ' wTotalLength
- Total length of this config. descriptor plus the
interface and endpoint descriptors that are part of the
configuration.
    0x00, 0,                                '
( high byte)
    0x01, 0,                                '
bNumInterfaces        - Number of interfaces
```

This defines the number of endpoints in the descriptor. This descriptor has 2 endpoints, one input and one output. Don't change this.

This is the number of bytes in the report descriptor. My report descriptor has 68 bytes. This number must be correct or the descriptor will not compile.

The actual descriptor begins here and is made up of multiple descriptors.
I broke out the length of each type of descriptor, which becomes the total length of the descriptor. The device descriptor is first and describes the device and is always 18 bytes long.

Vendor ID entered in ME descriptor tool

Product ID entered in ME descriptor tool

Pointer to manufacturer name
Pointer to product name
Pointer to serial number. Change this to 0x03 if you want to enter a serial number.

Beginning of configuration descriptor, always 9 bytes in length. Nothing to change in this descriptor

```
    0x01, 0,                                    '
bConfigurationValue   - Configuration Value
    0x00, 0,                                    '
iConfiguration        - String Index for this configuration
( None )
    0xA0, 0,                                    ' bmAttributes
- attributes - "Bus powered" and "Remote wakeup"
    50, 0,                                      ' MaxPower
- bus-powered draws 50*2 mA from the bus.

' Interface Descriptor
    USB_INTERF_DESCRIPTOR_LEN, 0,               ' bLength
- Length of Interface descriptor (always 0x09)
    USB_INTERFACE_DESCRIPTOR_TYPE, 0,           '
bDescriptorType       - 4 = INTERFACE descriptor
    0x00, 0,                                    '
bInterfaceNumber      - Number of interface, 0 based array
    0x00, 0,                                    '
bAlternateSetting     - Alternate setting
    NUM_ENDPOINTS, 0,                           ' bNumEndPoints
- Number of endpoints used in this interface
    0x03, 0,                                    '
bInterfaceClass       - assigned by the USB
    0x00, 0,                                    '
bInterfaceSubClass    - Not A boot device
    0x00, 0,                                    '
bInterfaceProtocol    - none
    0x00, 0,                                    ' iInterface
- Index to string descriptor that describes this interface
( None )

' HID Descriptor
    USB_HID_DESCRIPTOR_LEN, 0,                  ' bLength
- Length of HID descriptor (always 0x09)
    USB_HID_DESCRIPTOR_TYPE, 0,                 '
bDescriptorType       - 0x21 = HID descriptor
    0x01, 0,                                    ' HID class
release number (1.01)
    0x01, 0,
    0x00, 0,                                    ' Localized
country code (none)
    0x01, 0,                                    ' # of HID
class descriptor to follow (1)
    0x22, 0,                                    ' Report
descriptor type (HID)
    Low_HID_ReportDesc_len, 0,
    High_HID_ReportDesc_len, 0,

' EP1_RX Descriptor
    USB_ENDP_DESCRIPTOR_LEN, 0,                 ' bLength
- length of descriptor (always 0x07)
    USB_ENDPOINT_DESCRIPTOR_TYPE, 0,            '
bDescriptorType       - 5 = ENDPOINT descriptor
    0x81, 0,                                    '
bEndpointAddress      - In, EP1
    USB_ENDPOINT_TYPE_INTERRUPT, 0,             ' bmAttributes
- Endpoint Type - Interrupt
    Low_HID_PACKET_SIZE, 0,                     '
wMaxPacketSize        - max packet size - low order byte
    High_HID_PACKET_SIZE, 0,                    '
- max packet size - high order byte
```

Attributes and power are set by the ME descriptor tool.

Beginning of the interface descriptor, always 9 bytes in length. Nothing to change here.

Beginning of the HID descriptor, always 9 bytes in length. Nothing to change here.

This is the first endpoint descriptor. It is a receive endpoint descriptor, meaning it describes data being received by the PC. Nothing to change here.

```
    1, 0,                                   ' bInterval
- polling interval (1 ms)

' EP1_TX Descriptor
    USB_ENDP_DESCRIPTOR_LEN, 0,             ' bLength
- length of descriptor (always 0x07)
    USB_ENDPOINT_DESCRIPTOR_TYPE, 0,        '
bDescriptorType       - 5 = ENDPOINT descriptor
    0x01, 0,                                '
bEndpointAddress      - Out, EP1
    USB_ENDPOINT_TYPE_INTERRUPT, 0,         ' bmAttributes
- Endpoint Type - Interrupt
    Low_HID_PACKET_SIZE, 0,                 '
wMaxPacketSize        - max packet size - low order byte
    High_HID_PACKET_SIZE, 0,                '
- max packet size - high order byte
    1, 0,                                   ' bInterval
- polling interval (1 ms)

' HID_Report Descriptor
' USB Data Map
'  |----|----|----|----|----|----|----|----|
' 0|SW8 |SW7 |SW6 |SW5 |SW4 |SW3 |SW2 |SW1 |
'  |----|----|----|----|----|----|----|----|
' 1|SW16|SW15|SW14|SW13|SW12|SW11|SW10|SW9 |
'  |----|----|----|----|----|----|----|----|
' 2|SW24|SW23|SW22|SW21|SW20|SW19|SW18|SW17|
'  |----|----|----|----|----|----|----|----|
' 3|SW32|SW31|SW30|SW29|SW28|SW27|SW26|SW25|
'  |----|----|----|----|----|----|----|----|
' 4|               Slider 1                |
'  |----|----|----|----|----|----|----|----|
' 5|                   Z                   |
'  |----|----|----|----|----|----|----|----|
' 6|                   Rx                  |
'  |----|----|----|----|----|----|----|----|
' 7|                   Ry                  |
'  |----|----|----|----|----|----|----|----|
' 8|                   Rz                  |
'  |----|----|----|----|----|----|----|----|
0x05, 0,
0x01, 0,                    ' USAGE_PAGE (Generic Desktop)
0x15, 0,
0x00, 0,                    ' LOGICAL_MINIMUM (0)
0x09, 0,
0x04, 0,                    ' USAGE (Joystick)
0xa1, 0,
0x01, 0,                    ' COLLECTION (Application)
0x05, 0,
0x09, 0,                    '   USAGE_PAGE (Button)
0x19, 0,
0x01, 0,                    '   USAGE_MINIMUM (Button 1)
0x29, 0,
0x20, 0,                    '   USAGE_MAXIMUM (Button 32)
0x15, 0,
0x00, 0,                    '   LOGICAL_MINIMUM (0)
0x25, 0,
0x01, 0,                    '   LOGICAL_MAXIMUM (1)
0x75, 0,
0x01, 0,                    '   REPORT_SIZE (1)
0x95, 0,
0x20, 0,                    '   REPORT_COUNT (32)
```

Polling interval set by ME descriptor tool.

This is the second endpoint descriptor. It is a transmit endpoint descriptor, meaning it describes data being sent by the PC. Nothing to change here.

This is the report descriptor and describes what type of device(s) are connected to the USB port, and what the data looks like. In my report descriptor, the first four bytes contain switch data, and the next 5 bytes contain axis data. This descriptor was created with the USB Descriptor Tool.

IMPORTANT NOTE: The report descriptor must fill exact byte boundaries. If your report descriptor does not fill exact multiples of 8 bits, you can enter a filler as follows:

```
'0x95, 0,
'0x04, 0,      '  REPORT_COUNT (4)
               ' Adds 4 bits to data (you add what
               ' you need)
'0x81, 0,
'0x02, 0,      '  INPUT (Data,Var,Abs)
```

Maximum of 32 buttons declared

Each button takes up one bit in the report

32 bits total for the buttons; an even 4 bytes

```
0x55, 0,
0x00, 0,                    '   UNIT_EXPONENT (0)
0x65, 0,
0x00, 0,                    '   UNIT (None)
0x81, 0,
0x02, 0,                    '   INPUT (Data,Var,Abs)
0x05, 0,
0x01, 0,                     ' USAGE_PAGE (Generic Desktop)
0x09, 0,
0x36, 0,                     ' USAGE (Slider)
0x15, 0,
0x00, 0,                     ' LOGICAL_MINIMUM (0)
0x26, 0,
0xff, 0,
0x00, 0,                     ' LOGICAL_MAXIMUM (255)
0x75, 0,
0x08, 0,                     ' REPORT_SIZE (8)
0x95, 0,
0x01, 0,                     ' REPORT_COUNT (1)
0x81, 0,
0x02, 0,                     ' INPUT (Data,Var,Abs)
0x05, 0,
0x01, 0,                    '   USAGE_PAGE (Generic Desktop)
0x09, 0,
0x01, 0,                    '   USAGE (Pointer)
0x15, 0,
0x00, 0,                     ' LOGICAL_MINIMUM (0)
0x26, 0,
0xff, 0,
0x00, 0,                     ' LOGICAL_MINIMUM (255)
0xa1, 0,
0x00, 0,                    '   COLLECTION (Physical)
0x09, 0,
0x32, 0,                    '     USAGE (Z)
0x09, 0,
0x33, 0,                    '     USAGE (Rx)
0x09, 0,
0x34, 0,                    '     USAGE (Ry)
0x09, 0,
0x35, 0,                    '     USAGE (Rz)
0x95, 0,
0x04, 0,                    '     REPORT_COUNT (4)
0x81, 0,
0x02, 0,                    '     INPUT (Data,Var,Abs)
0xc0, 0,                    '   END_COLLECTION
0xc0, 0                     ' END_COLLECTION
)
'***************************************
 const LangIDDescr as byte[8] = (
    0x04, 0,
    USB_STRING_DESCRIPTOR_TYPE, 0,
    0x09, 0,                                ' LangID
(0x0409) - Low
    0x04, 0                                 '
- High
)
'***************************************
 const ManufacturerDescr as byte[28] = (
    14, 0,
    USB_STRING_DESCRIPTOR_TYPE, 0,
    "8", 0, 0, 0,
    "-", 0, 0, 0,
```

First axis declared as a slider.

Minimum value of the slider

Maximum value of the slider

One byte dedicated to report slider value

Pointer declaration. A pointer can have a number of different types of axes.

Second axis declared as Z

Third axis declared as X rotational axis

Fourth axis declared as Y rotational axis

Fifth axis declared as Z rotational axis, all part of the pointer control

4 bytes to report the values of the four pointer axes

Manufacturer name entered with ME descriptor tool

```
    "B", 0, 0, 0,
    "1", 0, 0, 0,
    "t", 0, 0, 0,
    "s", 0, 0, 0
)
'****************************************
 const ProductDescr as byte[56] = (
    28, 0,
    USB_STRING_DESCRIPTOR_TYPE, 0,
    "D", 0, 0, 0,
    "R", 0, 0, 0,
    "A", 0, 0, 0,
    " ", 0, 0, 0,
    "I", 0, 0, 0,
    "n", 0, 0, 0,
    "t", 0, 0, 0,
    "e", 0, 0, 0,
    "r", 0, 0, 0,
    "f", 0, 0, 0,
    "a", 0, 0, 0,
    "c", 0, 0, 0,
    "e", 0, 0, 0
)
'****************************************
 const StrUnknownDescr as byte[36] = (
    18, 0,
    USB_STRING_DESCRIPTOR_TYPE, 0,
    Serial_No[0], 0, 0, 0,
    Serial_No[1], 0, 0, 0,
    Serial_No[2], 0, 0, 0,
    Serial_No[3], 0, 0, 0,
    Serial_No[4], 0, 0, 0,
    Serial_No[5], 0, 0, 0,
    Serial_No[6], 0, 0, 0,
    Serial_No[7], 0, 0, 0
)
'****************************************

sub procedure InitUSBdsc
'****************************************
' Initialization Function
'****************************************
implements

sub procedure InitUSBdsc
dim dummy as byte volatile register

dummy = NUM_ENDPOINTS
dummy = ConfigDescr_wTotalLength
dummy = HID_ReportDesc_len
dummy = Low_HID_ReportDesc_len
dummy = High_HID_ReportDesc_len
dummy = Low_HID_PACKET_SIZE
dummy = High_HID_PACKET_SIZE
dummy = HID_INPUT_REPORT_BYTES
dummy = HID_OUTPUT_REPORT_BYTES
dummy = HID_FEATURE_REPORT_BYTES
'****************************************
' Byte constants
'****************************************
dummy = NUM_ENDPOINTS
dummy =  ConfigDescr_wTotalLength
```

Product name entered with ME descriptor tool

Serial Number. Here I take apart each byte of the serial number constant I declared at the beginning. You can just enter a character the same as the manufacturer or product strings above.

```
dummy = HID_ReportDesc_len
dummy =  Low_HID_ReportDesc_len
dummy = High_HID_ReportDesc_len
dummy =  Low_HID_PACKET_SIZE
dummy =  High_HID_PACKET_SIZE
dummy = @DescTables
dummy = @LangIDDescr
dummy = @ManufacturerDescr
dummy = @LangIDDescr
dummy = @ProductDescr
dummy = @StrUnknownDescr
end sub
end.
```

A number of key things to remember if you want your descriptor to work:

1. Be sure `const HID_ReportDesc_len = 68` equals the correct length of the report descriptor in bytes.
2. Start with a good working report descriptor before making any changes. Then make one change at a time and see if it still works.
3. Start with a USB sample program that works before making any changes.
4. If you make any changes to the report descriptor, be sure to always use even bytes.

How to Check if Your Descriptor is Working or Can the PC See My Device?

So how do you know if your descriptor is working? Simple, just plug a USB cable into the other USB port on the EasyPIC6 development board. You need two USB cables plugged into the board, one to communicate with the board and provide power, the other to communicate with the microcontroller you just programmed.

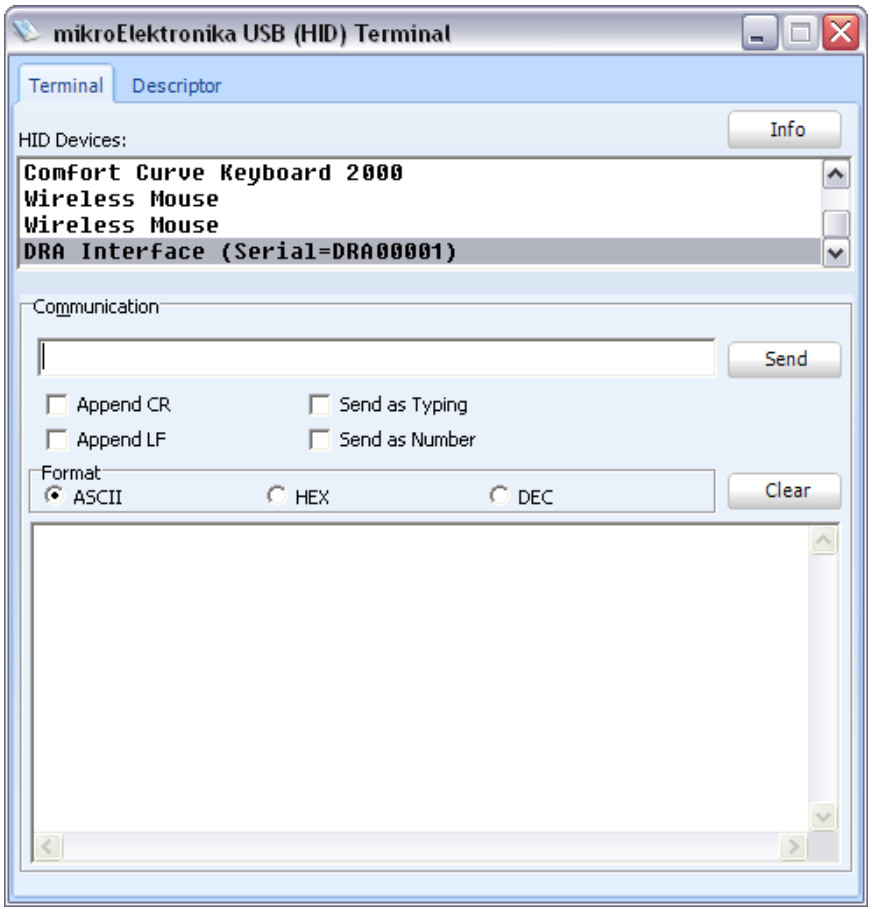

Figure 5: ME HID Terminal Recognizing Device

If the PC can successfully communicate with the device, when the device is plugged in you should hear the "bong-bing" chime. If this is the first time your device is connected to the PC you should also get messages indicating that new hardware was found, connected, and ready. Your device should also appear in the HID Terminal tool. Just scroll down until you see your device, then click on it.

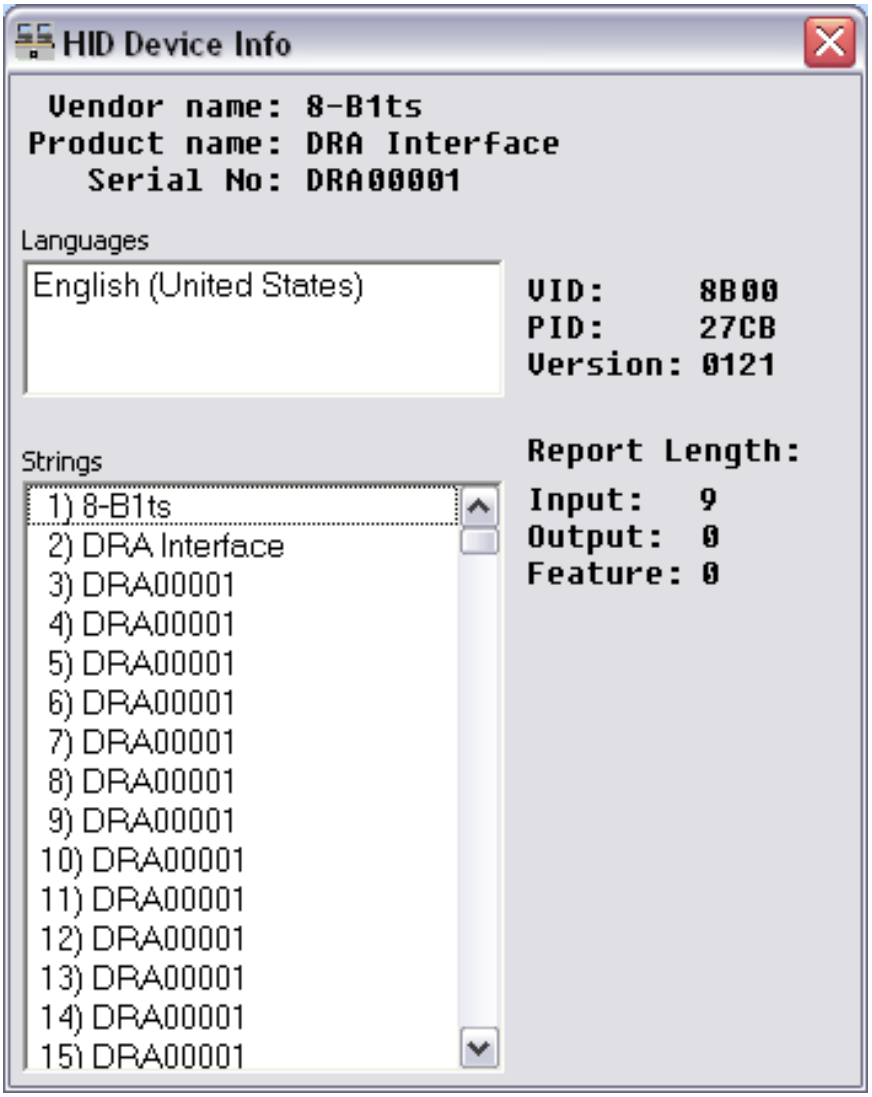

Figure 6: Device Information Screen

After you click on your device, if you click on the Info button you should see the screen shown in Figure 6. This contains all the information that came from your descriptor

Figure 7: Microsoft® Game Controller Window

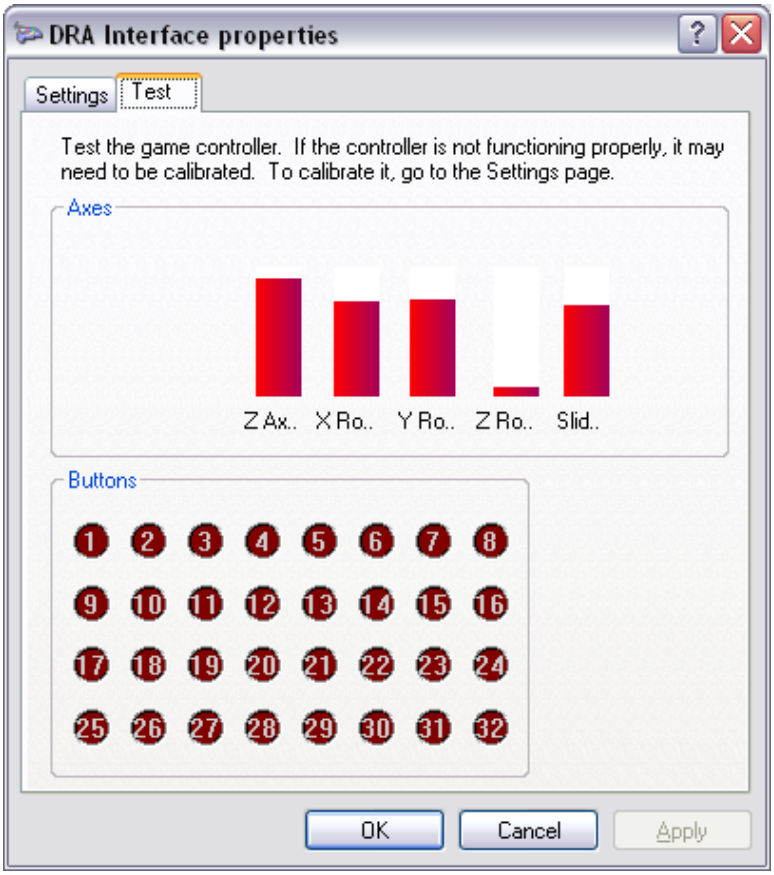

Figure 8: Device Properties

Since I declared my device as a joystick, it appears in the Game Controller window found in Control Panel as seen in Figure 7. When I click on Properties I can see all 32 switches and the five axes as shown in Figure 8.

I'm not going to go into the details of my firmware program, but basically I read the five analogue inputs for the five axes and drop the results into bytes 4 – 8 of the USB write buffer, then put switch data into the first four bytes of the write buffer, then send all 9 bytes to the USB port with the statement:

```
HID_Write(@userWR_buffer, 9)
```

So what happens if the PC does not recognize your device? A hardware problem caused the message "USB Device not Recognized" to pop up. I created a stand-alone board that had problems with the type B USB connector that I have since corrected.

When the report descriptor was not correct I heard "bong" four times in rapid succession and got the message, "Unknown Device", and "There was a problem communicating with the device". This was caused by not declaring the correct multiples of 8 bits in the report descriptor. Again, I started with a good working descriptor and a good working program, then made small changes to see what happened.

Summary

As I stated before, it took me weeks of research and frustration to get to the point where the PC recognized my device and the descriptor. I cannot stress enough:

- Make sure your microcontroller configuration is correct
- Start with a good working descriptor and a good working program
- Make small changes and test to see if your microcontroller is still communicating with the PC

Now that I have a good working descriptor, I can concentrate on my program.

References

Company/Product	Web Site	Description
mikroElektronika	www.mikroe.com	Source for : • Easy Pic6 development board • MikroBASIC PRO for PIC compiler • Microcontroller manuals
Desktop Aviator	www.desktopaviator.com	Aviation simulation controls and interface boards

FSUIPC by Peter Dowson	http://www.schiratti.com/dowson.html	Add-in .dll file for Flight Simulator that interfaces with switches. It does a lot of other things, but I only used the switch interface capabilities.
Microchip Technology Inc	www.microchip.com	Data sheets for PIC microcontrollers
USB Implementers Forum, Inc.	www.usb.org	USB standards and the HID Descriptor Tool
Amr Bekhit	http://www.helmpcb.com/Electronics/USBJoystick/USB Joystick.aspx	Great article on how to convert an old joystick port joystick to USB using a microcontroller. **This was my first breakthrough for creating a valid descriptor. Thanks Amr!**
USB Made Simple	http://www.usbmadesimple.co.uk/index.html	A series of articles on how USB works. Great write up on the descriptor.
USB In a Nutshell	http://www.beyondlogic.org/usbnutshell/	Another great article on how USB works. Good information on the descriptor.