PIC Communication with USB

By Roy Seifert

This page intentionally left blank

Revised July 17, 2010 for use with mikroBASIC PRO for PIC version 3.80

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.

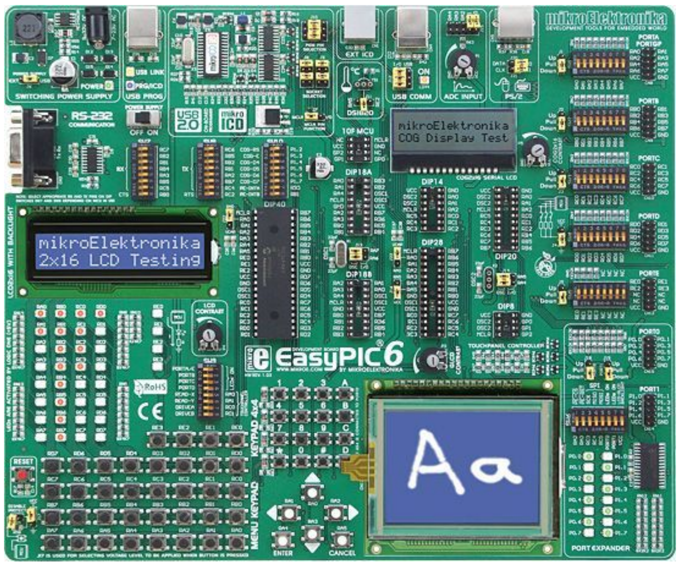


Figure 1: EasyPIC6 Development Board

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

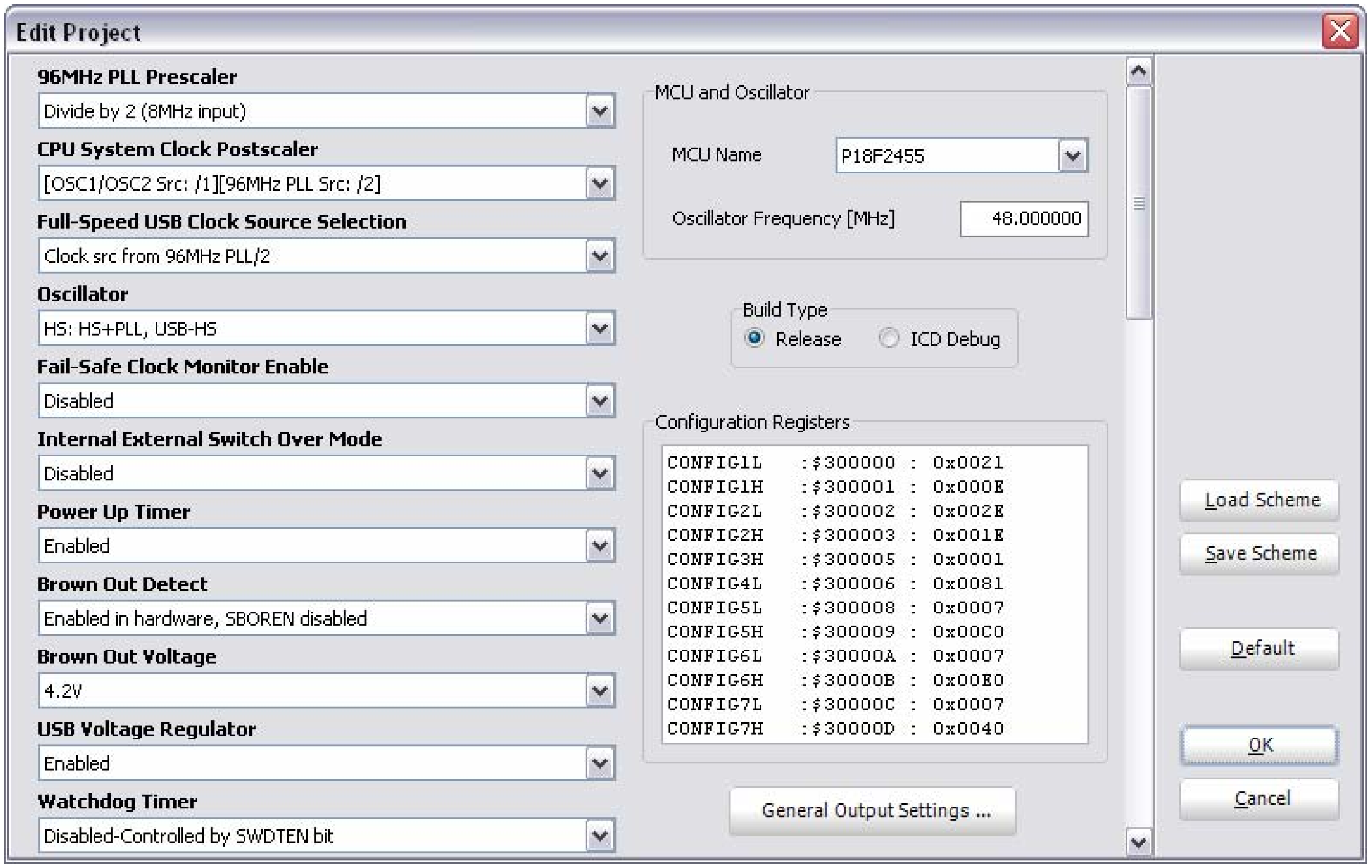
Before going any farther, I want to extend my thanks to the mikroElektronika (ME) team for their excellent products and support. Their hardware, software, libraries and examples made my development work almost effortless. Sure I had a pretty steep learning curve, but the tools provided by ME made learning 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.


Edit Project

96MHz PLL Prescaler: Divide by 2 (8MHz input)
CPU System Clock Postscaler: [OSC1/OSC2 Src: /1][96MHz PLL Src: /2]
Full-Speed USB Clock Source Selection: Clock src from 96MHz PLL/2
Oscillator: HS: HS+PLL, USB-HS
Fail-Safe Clock Monitor Enable: Disabled
Internal External Switch Over Mode: Disabled
Power Up Timer: Enabled
Brown Out Detect: Enabled in hardware, SBOREN disabled
Brown Out Voltage: 4.2V
USB Voltage Regulator: Enabled
Watchdog Timer: Disabled-Controlled by SWDTEN bit

MCU and Oscillator
MCU Name: P18F2455
Oscillator Frequency [MHz]: 48.000000

Build Type: [✓ Release] ICD Debug

Configuration Registers

```
CONFIG1L   :$300000 : 0x0021
CONFIG1H   :$300001 : 0x000E
CONFIG2L   :$300002 : 0x002E
CONFIG2H   :$300003 : 0x001E
CONFIG3H   :$300005 : 0x0001
CONFIG4L   :$300006 : 0x0081
CONFIG5L   :$300008 : 0x0007
CONFIG5H   :$300009 : 0x00C0
CONFIG6L   :$30000A : 0x0007
CONFIG6H   :$30000B : 0x00E0
CONFIG7L   :$30000C : 0x0007
CONFIG7H   :$30000D : 0x0040
```

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.

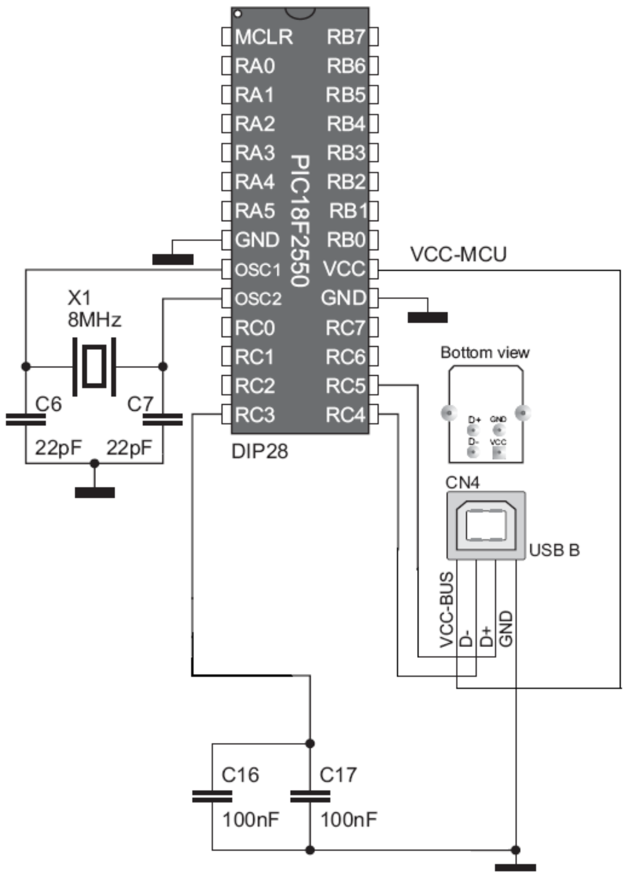


Figure 3: 28-Pin Microcontroller Hardware Configuration

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.

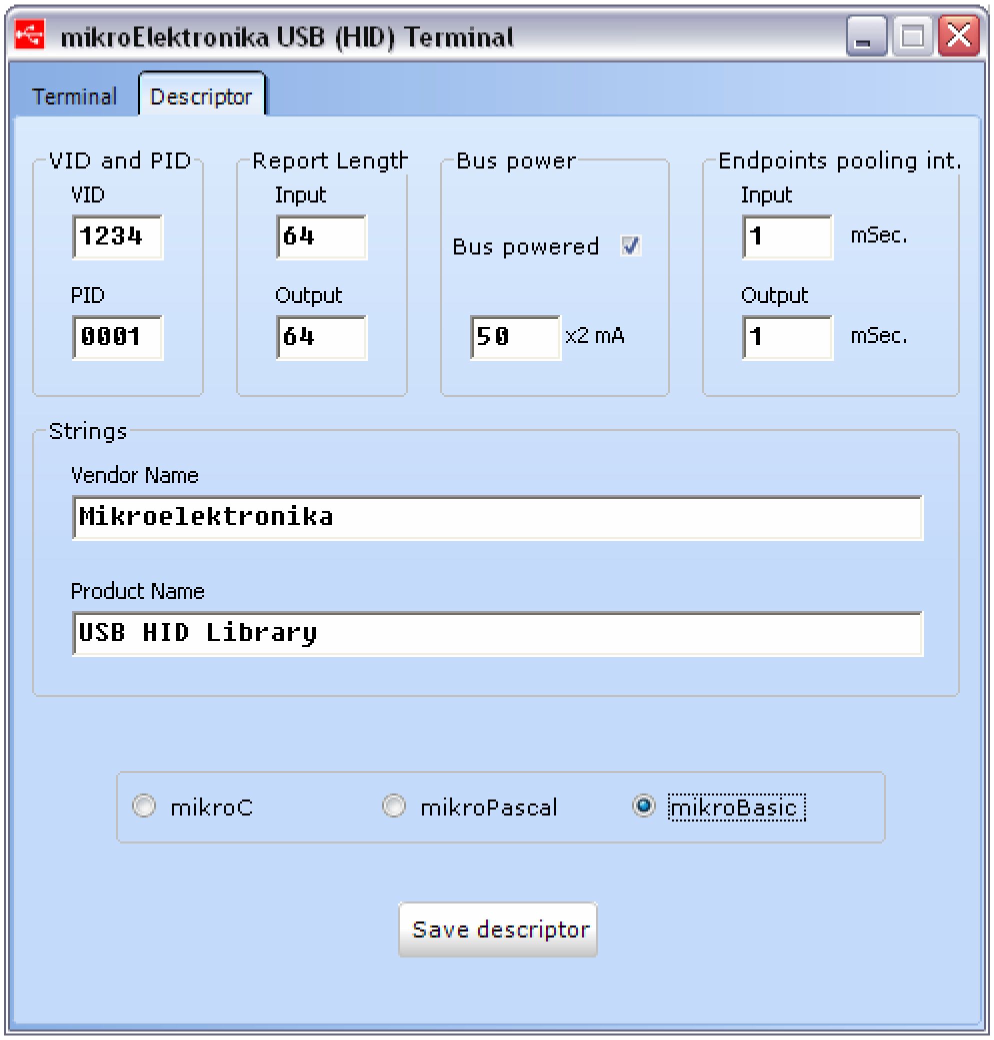
A Few Words about Descriptors

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.
- 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.


mikroElektronika USB (HID) Terminal

Terminal | Descriptor

VID and PID
VID: 1234
PID: 0001

Report Length
Input: 64
Output: 64

Bus power
Bus powered ☑
50 x2 mA

Endpoints pooling int.
Input: 1 mSec.
Output: 1 mSec.

Strings
Vendor Name: Mikroelektronika
Product Name: USB HID Library

mikroC mikroPascal [✓ mikroBasic]

Save descriptor

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 modify a CH Products F16 joystick that interfaced to a joystick port so it would interface with a USB port. Following is the descriptor that I created along with an explanation of the various parts. Anything you see in red is a change I made.

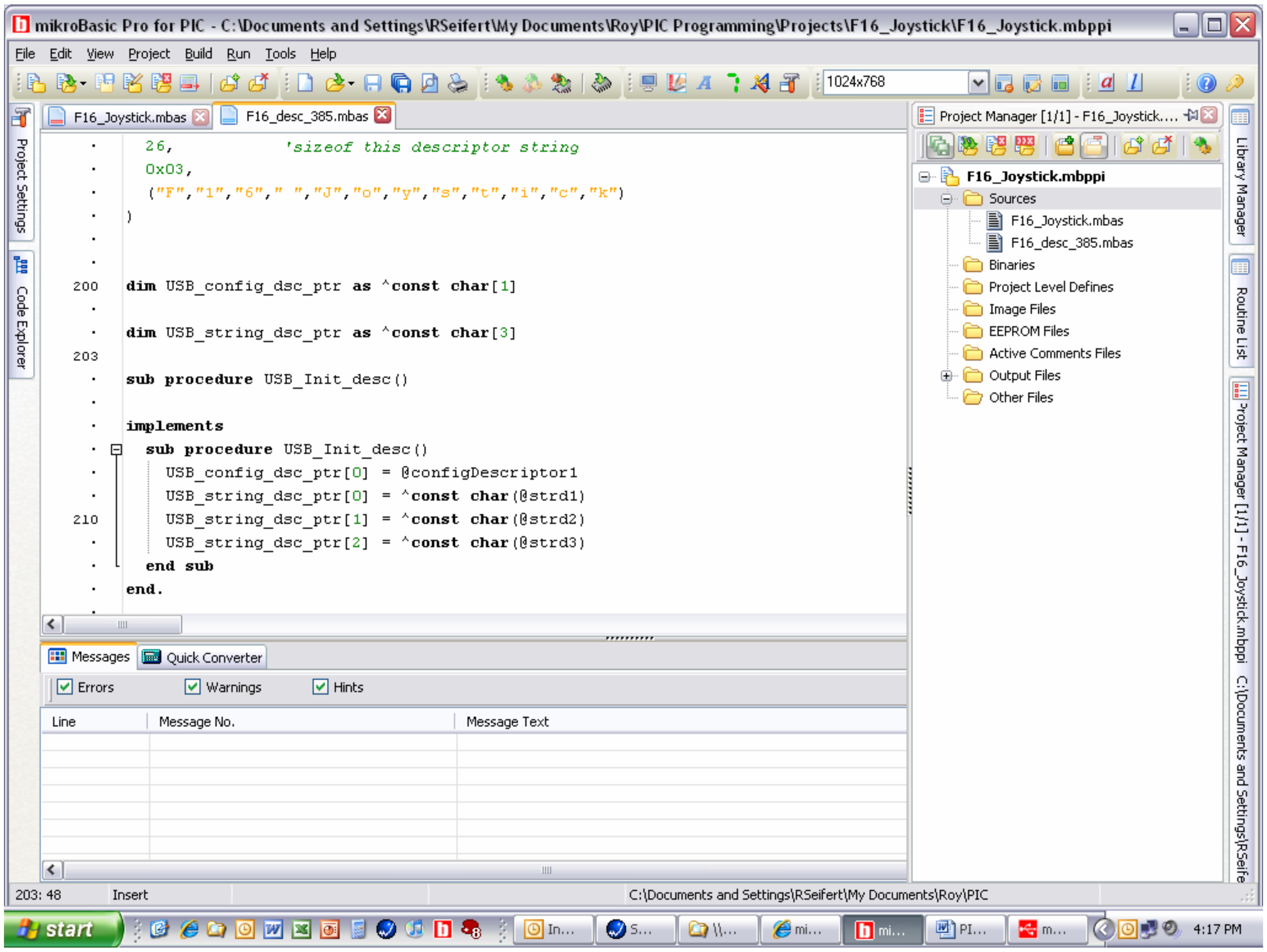
```
module F16dsc

const USB_VENDOR_ID as word = 0x1234
const USB_PRODUCT_ID as word = 0x0001
const USB_SELF_POWER as char = 0x80            ' Self powered 0xC0,  0x80 bus powered
const USB_MAX_POWER as char = 50               ' Bus power required in units of 2 mA
const HID_INPUT_REPORT_BYTES as char = 64
const HID_OUTPUT_REPORT_BYTES as char = 64
const EP_IN_INTERVAL as char = 1
const EP_OUT_INTERVAL as char = 1

const USB_INTERRUPT as char = 0
const USB_TRANSFER_TYPE as char = 0x03         '0x03 Interrupt
const USB_HID_EP as char = 1
const USB_HID_RPT_SIZE as char = 78            'Changed report descriptor size for my report descriptor

structure device_descriptor
    dim bLength as char                ' bLength         - Descriptor size in bytes (12h)
    dim bDescriptorType as char        ' bDescriptorType - The constant DEVICE (01h)
    dim bcdUSB as word                 ' bcdUSB          - USB specification release number (BCD)
    dim bDeviceClass as char           ' bDeviceClass    - Class Code
    dim bDeviceSubClass as char        ' bDeviceSubClass - Subclass code
    dim bDeviceProtocol as char        ' bDeviceProtocol - Protocol code
    dim bMaxPacketSize0 as char        ' bMaxPacketSize0 - Maximum packet size for endpoint 0
    dim idVendor as word               ' idVendor        - Vendor ID
    dim idProduct as word              ' idProduct       - Product ID
    dim bcdDevice as word              ' bcdDevice       - Device release number (BCD)
    dim iManufacturer as char          ' iManufacturer   - Index of string descriptor for the manufacturer
    dim iProduct as char               ' iProduct        - Index of string descriptor for the product.
    dim iSerialNumber as char          ' iSerialNumber   - Index of string descriptor for the serial number.
    dim bNumConfigurations as char     ' bNumConfigurations - Number of possible configurations
end structure

const device_dsc as device_descriptor = (
  0x12,                    ' bLength
  0x01,                    ' bDescriptorType
  0x0200,                  ' bcdUSB
  0x00,                    ' bDeviceClass
  0x00,                    ' bDeviceSubClass
  0x00,                    ' bDeviceProtocol
  8,                       ' bMaxPacketSize0
  USB_VENDOR_ID,           ' idVendor
  USB_PRODUCT_ID,          ' idProduct
  0x0020,                  ' bcdDevice            'Change this 4-digit hex value for the version number
  0x01,                    ' iManufacturer
  0x02,                    ' iProduct
  0x03,                    ' iSerialNumber        'Change to 0x03 to add a serial number
  0x01                     ' bNumConfigurations
)

' Configuration 1 Descriptor
const configDescriptor1 as byte[41] = (
    ' Configuration Descriptor
    0x09,                    ' bLength             - Descriptor size in bytes
    0x02,                    ' bDescriptorType     - The constant CONFIGURATION (02h)
    0x29,0x00,               ' wTotalLength        - The number of bytes in the configuration descriptor and all
of its subordinate descriptors
    1,                       ' bNumInterfaces      - Number of interfaces in the configuration
    1,                       ' bConfigurationValue - Identifier for Set Configuration and Get Configuration
requests
    0,                       ' iConfiguration      - Index of string descriptor for the configuration
    USB_SELF_POWER,          ' bmAttributes        - Self/bus power and remote wakeup settings
    USB_MAX_POWER,           ' bMaxPower           - Bus power required in units of 2 mA

    ' Interface Descriptor
    0x09,                    ' bLength - Descriptor size in bytes (09h)
    0x04,                    ' bDescriptorType - The constant Interface (04h)
    0,                       ' bInterfaceNumber - Number identifying this interface
    0,                       ' bAlternateSetting - A number that identifies a descriptor with alternate settings
for this bInterfaceNumber.
    2,                       ' bNumEndpoint - Number of endpoints supported not counting endpoint zero
    0x03,                    ' bInterfaceClass - Class code
    0,                       ' bInterfaceSubclass - Subclass code
    0,                       ' bInterfaceProtocol - Protocol code
    0,                       ' iInterface - Interface string index

    ' HID Class-Specific Descriptor
    0x09,                    ' bLength         - Descriptor size in bytes.
    0x21,                    ' bDescriptorType - This descriptor's type: 21h to indicate the HID class.
    0x01,0x01,               ' bcdHID          - HID specification release number (BCD).
    0x00,                    ' bCountryCode    - Numeric expression identifying the country for localized
hardware (BCD) or 00h.
    1,                       ' bNumDescriptors - Number of subordinate report and physical descriptors.
    0x22,                    ' bDescriptorType - The type of a class-specific descriptor that follows
    USB_HID_RPT_SIZE,0x00,   ' wDescriptorLength - Total length of the descriptor identified above.

    ' Endpoint Descriptor
    0x07,                    ' bLength - Descriptor size in bytes (07h)
    0x05,                    ' bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP or 0x80,      ' bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,       ' bmAttributes - Transfer type and supplementary information
    0x40,0x00,               ' wMaxPacketSize - Maximum packet size supported
    EP_IN_INTERVAL,          ' bInterval - Service interval or NAK rate

    ' Endpoint Descriptor
    0x07,                    ' bLength - Descriptor size in bytes (07h)
    0x05,                    ' bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP,              ' bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,       ' bmAttributes - Transfer type and supplementary information
    0x40,0x00,               ' wMaxPacketSize - Maximum packet size supported
    EP_OUT_INTERVAL          ' bInterval - Service interval or NAK rate
)

structure hid_report_descriptor
  dim report as char[USB_HID_RPT_SIZE]
end structure

const hid_rpt_desc as hid_report_descriptor = (
'   USB Data Map
'  |----|----|----|----|----|----|----|----|    'This USB data map helps me to determine where to place the
' 0|               Throttle                |    'data into the USB write buffer.  This data map is determined
'  |----|----|----|----|----|----|----|----|    'by the declarations in the report descriptor.
' 1|                X Axis                 |
'  |----|----|----|----|----|----|----|----|
' 2|                Y Axis                 |
'  |----|----|----|----|----|----|----|----|
' 3|SW4 |SW3 |SW2 |SW1 |    Hat Switch     |
'  |----|----|----|----|----|----|----|----|
' 4|N/A |N/A |SW10|SW9 |SW8 |SW7 |SW6 |SW5 |
'  |----|----|----|----|----|----|----|----|

   (0x05, 0x01,                                  ' USAGE_PAGE (Generic Desktop)
    0x15, 0x00,                                  ' LOGICAL_MINIMUM (0)
    0x09, 0x04,                                  ' USAGE (Joystick)
    0xA1, 0x01,                                  ' COLLECTION (Application)
    0x05, 0x02,                                  '   USAGE_PAGE (Simulation Controls)
    0x09, 0xBB,                                  '   USAGE (Throttle)
    0x15, 0x00,                                  '   LOGICAL_MINIMUM (0)
    0x26, 0xFF, 0x00,                            '   LOGICAL_MAXIMUM (255)
    0x75, 0x08,                                  '   REPORT_SIZE (8)
    0x95, 0x01,                                  '   REPORT_COUNT (1)
    0x81, 0x02,                                  '   INPUT (Data Var Abs)
    0x05, 0x01,                                  '   USAGE_PAGE (Generic Desktop)
    0x09, 0x01,                                  '   USAGE (Pointer)
    0xA1, 0x00,                                  '   COLLECTION (Physical)
    0x09, 0x30,                                  '     USAGE (X)
    0x09, 0x31,                                  '     USAGE (Y)
    0x95, 0x02,                                  '     REPORT_COUNT (2)
    0x81, 0x02,                                  '     INPUT (Data Var Abs)
    0xC0,                                      '   END_COLLECTION
    0x09, 0x39,                                   '   USAGE (Hat switch)
    0x15, 0x00,                                   '   LOGICAL_MINIMUM (0)
    0x25, 0x03,                                   '   LOGICAL_MAXIMUM (3)
    0x35, 0x00,                                   '   PHYSICAL_MINIMUM (0)
    0x46, 0x0E, 0x01,                             '   PHYSICAL_MAXIMUM (270)
    0x65, 0x14,                                   '   UNIT (Eng Rot:Angular Pos)
    0x75, 0x04,                                   '   REPORT_SIZE (4)
    0x95, 0x01,                                   '   REPORT_COUNT (1)
    0x81, 0x02,                                   '   INPUT (Data Var Abs)
    0x05, 0x09,                                   '   USAGE_PAGE (Button)
    0x19, 0x01,                                   '   USAGE_MINIMUM (Button 1)
    0x29, 0x0A,                                   '   USAGE_MAXIMUM (Button 10)
    0x15, 0x00,                                   '   LOGICAL_MINIMUM (0)
    0x25, 0x01,                                   '   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                                   '   REPORT_SIZE (1)
    0x95, 0x0C,                                   '   REPORT_COUNT (12) 2 bits added to switch report count
                                                  '                     so bytes are even.  Bytes must be even.
    0x55, 0x00,                                   '   UNIT_EXPONENT (0)
    0x65, 0x00,                                   '   UNIT (None)
    0x81, 0x02,                                   '   INPUT (Data Var Abs)
    0xC0)                                         ' END_COLLECTION
  )

'Language code string descriptor
structure str1
  dim bLength as char
  dim bDscType as char
  dim wString as word[1]
end structure

const strd1 as str1 = (
  4,
  0x03,
  (0x0409)
)

'Manufacturer string descriptor
structure str2
  dim bLength as char
  dim bDscType as char
  dim wString as word[16]
end structure

const strd2 as str2 = (
  14,           'sizeof this descriptor string
  0x03,
  ("8"," ","B","i","t","s")
)

'Product string descriptor
structure str3
  dim bLength as char
  dim bDscType as char
  dim wString as word[23]
end structure

const strd3 as str3 = (
  26,           'sizeof this descriptor string
  0x03,
  ("F","1","6"," ","J","o","y","s","t","i","c","k")
)

'Serial number string descriptor                  'Add this string for serial number
structure str4
  dim bLength as char
  dim bDscType as char
  dim wString as word[7]
end structure

const strd4 as str4 = (
  16,           'sizeof this descriptor string
  0x03,
  ("F","1","6","0","0","0","1")
)


dim USB_config_dsc_ptr as ^const char[1]

dim USB_string_dsc_ptr as ^const char[3]

sub procedure USB_Init_desc()

implements
  sub procedure USB_Init_desc()
    USB_config_dsc_ptr[0] = @configDescriptor1
    USB_string_dsc_ptr[0] = ^const char(@strd1)
    USB_string_dsc_ptr[1] = ^const char(@strd2)
    USB_string_dsc_ptr[2] = ^const char(@strd3)
    USB_string_dsc_ptr[3] = ^const char(@strd4)   ‘Send the serial number to the PC
  end sub
end.
```

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

1. Be sure `const USB_HID_RPT_SIZE as char = 78` 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.

Project Manager

Now that the USB descriptor is completed you must tell the compiler to use the file. This is done with the Project Manager.



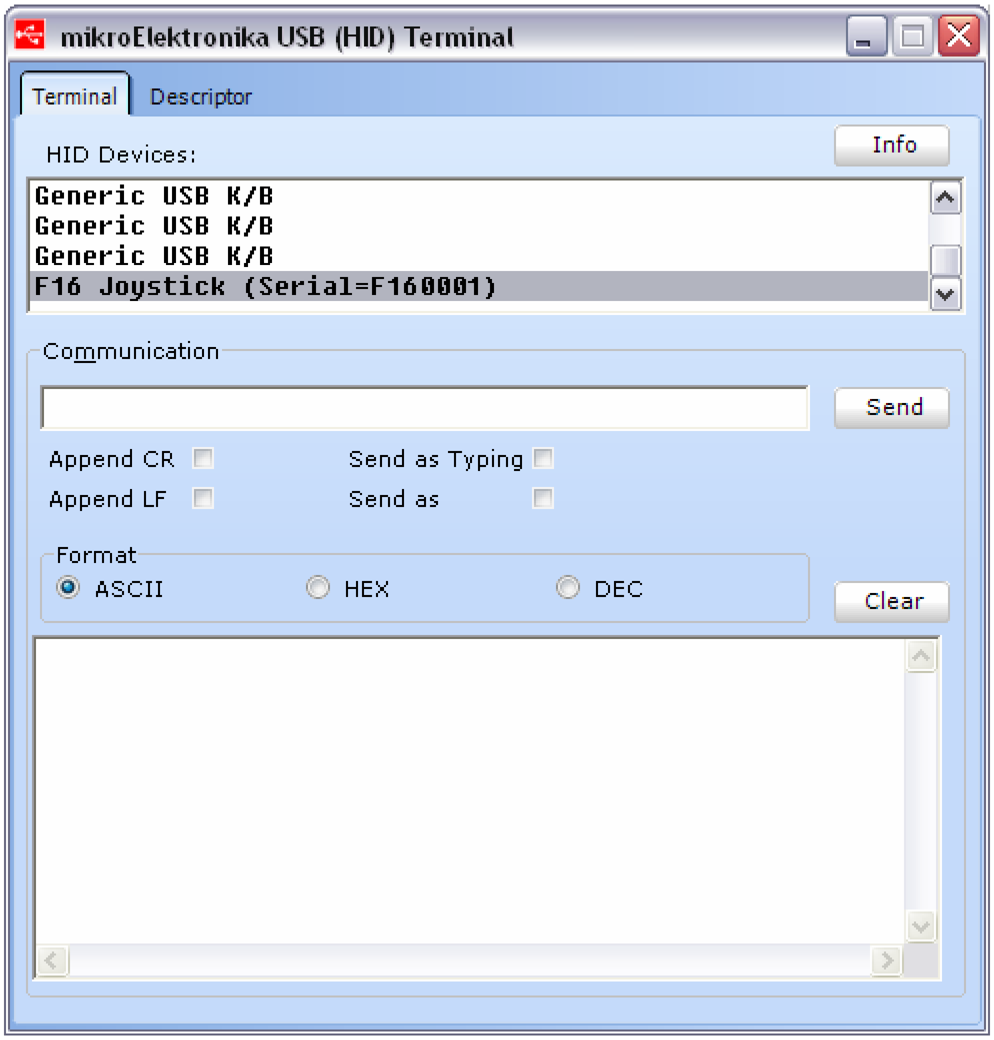
```
  26,           'sizeof this descriptor string
  0x03,
  ("F","1","6"," ","J","o","y","s","t","i","c","k")
)


dim USB_config_dsc_ptr as ^const char[1]

dim USB_string_dsc_ptr as ^const char[3]

sub procedure USB_Init_desc()

implements
  sub procedure USB_Init_desc()
    USB_config_dsc_ptr[0] = @configDescriptor1
    USB_string_dsc_ptr[0] = ^const char(@strd1)
    USB_string_dsc_ptr[1] = ^const char(@strd2)
    USB_string_dsc_ptr[2] = ^const char(@strd3)
  end sub
end.
```

Figure 5: Project Manager

On the right side of the compiler screen click on the Project Manager tab. This opens the Project Manager as shown in Figure 5. Be sure the descriptor file appears below the Sources folder. If it does not appear, right-click on the Sources folder, click Add File to Project…, then select your descriptor file. Now, when your main program compiles, the descriptor file will automatically be included. Be sure to save your project by clicking on Project, Save Project in the menu.

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.



HID Devices:

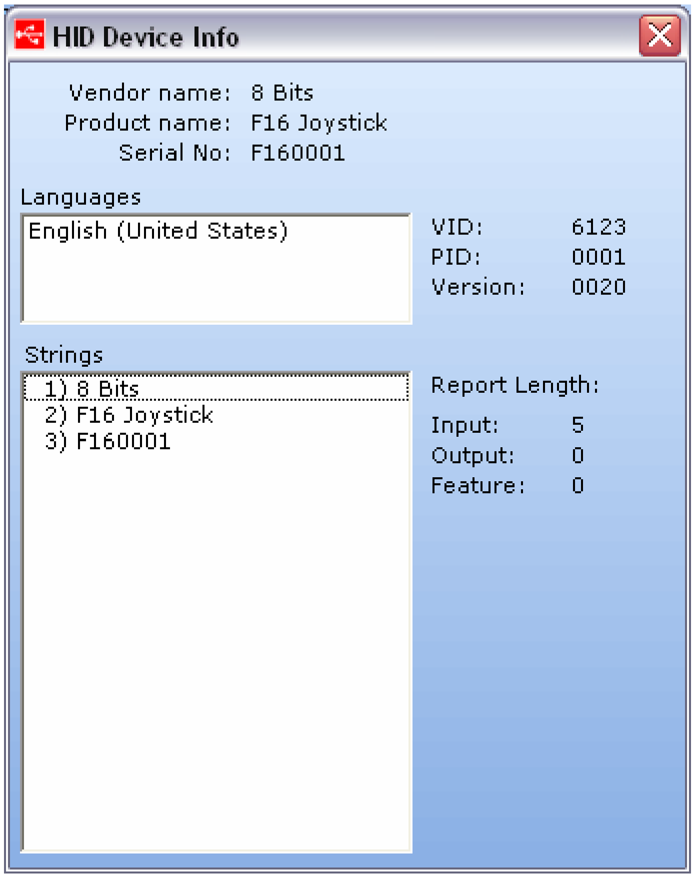
Generic USB K/B
Generic USB K/B
Generic USB K/B
F16 Joystick (Serial=F160001)

Append CR [ ]
Send as Typing [ ]
Append LF [ ]
Send as [ ]

Format: [✓ ASCII] HEX DEC

Figure 6: 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, note the serial number. Just scroll down until you see your device, then click on it.


HID Device Info

Vendor name: 8 Bits
Product name: F16 Joystick
Serial No: F160001

Languages
English (United States)

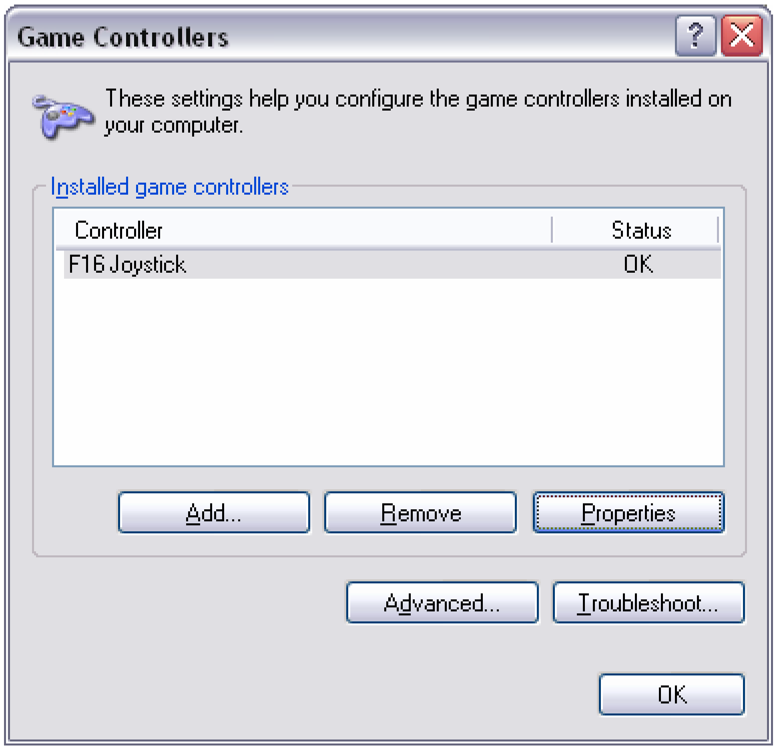
VID: 6123
PID: 0001
Version: 0020

Strings
1) 8 Bits
2) F16 Joystick
3) F160001

Report Length:
Input: 5
Output: 0
Feature: 0

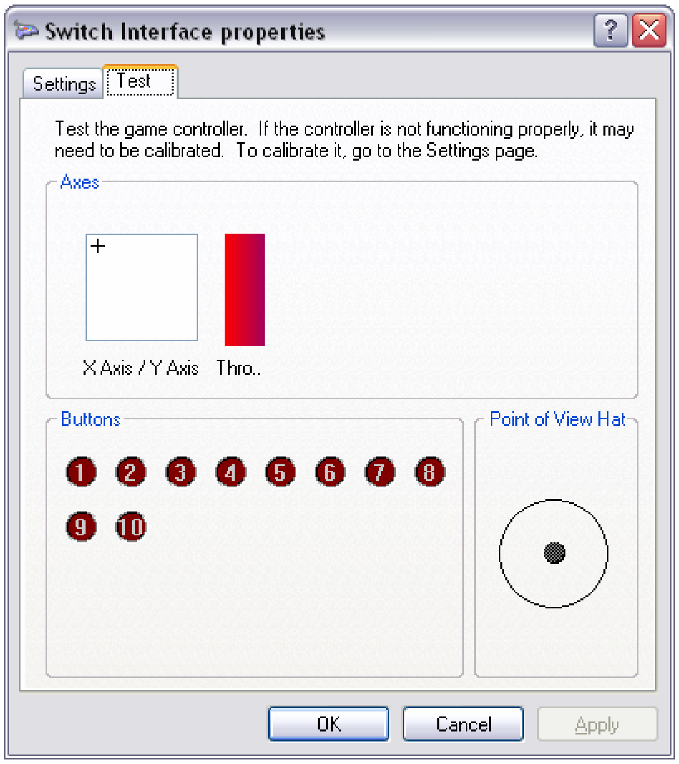
Figure 7: Device Information Screen

After you click on your device, if you click on the Info button you should see the screen shown in Figure 7. This contains all the information that came from your descriptor. Note the version and serial number match what I had entered in the descriptor.


Game Controllers
These settings help you configure the game controllers installed on your computer.
Installed game controllers
Add...
Remove
Properties
Advanced...
Troubleshoot...
OK

Controller	Status
F16 Joystick	OK

Figure 8: Microsoft® Game Controller Window



Test the game controller. If the controller is not functioning properly, it may need to be calibrated. To calibrate it, go to the Settings page.

Figure 9: 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 8. When I click on Properties I can see all the switches and axes as shown in Figure 9.

I'm not going to go into the details of my firmware program, but basically I read the three analogue inputs for the three axes and drop the results into bytes 0 - 2 of the USB write buffer. The 4 hat switches and first four push-button switches I put into byte 3, and the last 6 switches I put into byte 4. I then send all 5 bytes to the USB port with the statement: `HID_Write(@userWR_buffer, 5)`

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/USBJoystick.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.