USB INSTRUMENTATION

by

Eric Brown

This CPSC 490 research project is submitted in partial fulfillment of the requirements for the degree of

B.S. Computer Science

Yale University

2006

Project Advisors

Professor Robert Grober, Applied Physics

Assistant Professor Andreas G. Savvides, Electrical Engineering and Computer Science
The Universal Serial Bus (USB) has slowly replaced the older RS-232 communication protocol as the standard solution for basic PC to peripheral communication. The USB provides larger bandwidths, 500ma of power, and reduces the necessary supporting PC related hardware required for the growing number of peripheral devices. For these reasons, the Applied Physics Department, specifically the Grober Lab, would like the ability to upgrade their legacy RS-232 instruments to USB and gain the necessary information to implement USB into future projects.
Table of Contents

Introduction.................................................................................................................. 1
Background.................................................................................................................. 3
Review of the State of Art ........................................................................................... 8
Materials..................................................................................................................... 10
Methods...................................................................................................................... 12
   Hardware Development.............................................................................................. 12
   Windows Application Development........................................................................... 15
Results....................................................................................................................... 21
Future Work.................................................................................................................. 22
Appendix A: Windows Enumeration Process (UML).................................................. 23
Appendix B: PIC USB Minimal Schematic................................................................. 25
Appendix C: Bandwidth Testing.................................................................................. 28
References................................................................................................................... 30
Introduction

For more than 20 years, the major method of communication between computers and ancillary devices has been conducted using the RS-232 communication protocol. Often referred to as the “serial port”, RS-232 is typically implemented as a 9 pin cable that transmits data serially, one bit at a time, over two lines TX and RX; transmit and receive respectively.

Over the years other communication protocols were developed, each with their own operational characteristics and physical implementations, which brought more harm than good to the end user. Between printers, mice, keyboards, SCSI devices, and general purpose serial devices, no two shared the same physical port configuration and the installation of new peripherals became difficult to manage for not only end users, but vendors as well.

In November of 1995, the USB 1.0 specification was released by the USB Implementers Forum, Inc; (USB-IF). After a lengthy industry adoption phase, the USB communications protocol is now the communications standard of choice for most, if not all computer peripherals. There is no secret why USB is so popular. USB 1.0 specified a bandwidth of 1.5 Mbits/s (Low-Speed), 13 times faster than standard RS-232, and 12 Mbits/s (Full-Speed), 104 times faster. USB devices are “plug and play”; peripherals can be attached and detached without powering off the system. A single USB host can communicate with 127 different peripherals, greatly reducing the necessary peripheral hardware, and USB is capable of supplying a limited amount of power to connected peripherals; thus reducing the
need for external power supplies or batteries. The success of the USB is obvious; it’s easier
to use, faster, and accommodates more peripherals on a smaller hardware footprint than RS-
232.

Since the publication of version 1.0, the USB specification has undergone two major
revisions; version 1.1 and version 2.0. Version 1.1 differed slightly from 1.0 and was written
to provide further clarification of the 1.0 specification. The biggest change in the
specification came in April of 2000 with the release of USB version 2.0 with the addition of a
new High-Speed configuration which is capable of 480Mbits/s bandwidths.

The overwhelming popularity of USB ultimately led to the slow decline of other legacy
communication protocols like RS-232 and as a result, the DB9 connector most commonly
associated with RS-232 is becoming harder to find on computers. For those who spent time
and money developing and implementing peripherals that communicate using the RS-232
protocol, their peripherals are about to become obsolete.

The scope of this project is concerned with gaining an in-depth understanding of the USB
communications protocol, its relationship to the Windows Operating System and a USB
enabled peripheral device. More specifically, this project will aim to implement USB
communications between a PIC Microchip 18F2550/18F4550 microcontroller and a
Windows XP based application written in C# to be used in biomechanical measurements.
Background

Conceptually, USB communication can be likened to the post office. A customer is notified that a package(s) have arrived and that the package(s) must be collected; the customer is rarely concerned with the specific details about its travels from one point to another, nor are they involved with the details. The USB specification strictly defines all the necessary functionality for peripheral communication and this functionality must be implemented by the device vendor in order to obtain the USB certification. The application developer, should only concern themselves with collecting packages from the in-box as they arrive and pushing data to be sent in the “out-box”.

Unlike RS-232, there is no access to the USB hardware directly. All communications are brokered by a device driver, which in turn interacts directly with the hardware by way of the operating system. This hardware interface layer is often referred to as the kernel layer and programming at this level is difficult and potentially hazardous to the stability of the entire operating system. For this reason, developing a custom device driver is outside the scope of this project and an off the shelf option will be used.
Wired USB communications are transmitted across a 4 connector cable consisting of a dedicated 5+ volt line (pin 1 / Red), data transmission lines (pins 2 & 3 / White & Green), and a ground (pin 4 / Black). There are a number of different female sockets into which the USB cables are inserted and they are identified by A, B, Mini A, and Mini B. The “Mini” plugs are 5 connector and belong to the new USB ON-THE-GO specification. ON-THE-GO devices, as well as the wireless USB standard, is outside the scope of this project and will not be covered in this document.

Besides the ease of use, by way of a common peripheral connector, USB devices offer higher transfer speeds than previous communication protocols. USB 1.0 specifies a maximum theoretical bandwidth of 1.5Mbps, in a Low-Speed configuration and 12Mbps in a Full-Speed configuration. The current USB version 2.0 specifies a new High-Speed configuration that has a maximum bandwidth of 480Mbps; these are upper bounds on bandwidth. True throughput is heavily affected by the underlying firmware, application code, and bus traffic.

The current USB specification also defines inter-compatibility among the three configurations. In a mixed-mode environment, different USB devices will coexist peacefully on the same bus, but bandwidth is limited by the host controller if the controller is not capable of the same transfer rates as an attached, higher performance device. To

---

1 A discussion of IEEE-1394 (Firewire) has purposely been omitted in this document.
accomplish this, host controllers and hubs are required to contain a small amount of memory to buffer communication between differently configured devices.

To determine the specific capabilities of each device, the peripheral and the host go through a series of data exchanges upon attachment that inform the operating system of the device and the maximum speed the two can communicate; low, full, or high. This is the first step in the enumeration process; a common process on Windows systems through which all peripherals are registered.

Each device on the USB is identified by a Vendor ID and a Product ID; VID & PID. VID numbers are managed by USB-IF and each number is licensed to a specific vendor; no two VID numbers are used by more than one vendor. On the other hand, the PID is used by the vendor to distinguish vendor products and is assigned without regard of other vendor PIDs. These numbers together constitute a unique key that identifies devices on the bus which the operating system uses during communication.

Every USB peripheral is required to implement one or more device descriptors that include information about the device class (type) and its capabilities; Endpoints and communication configurations. The host will request descriptor(s) during the enumeration process, which it will use to register the device with the operating system using the VID and PID, load the appropriate drivers, and set the rules of communication. A detailed UML diagram of the enumeration process can be found in Appendix A.
While the underlying mechanics of USB transmissions are invisible to the application developer, debugging a USB peripheral is often difficult and requires a general understanding of how the host and peripheral communicate. Conceptually, there are two physical components that make up the USB hardware: Endpoints and Pipes.

Endpoints are sinks and sources of data. USB 2.0 devices can be configured to have a maximum of 16 independently configured Endpoints, 0 to 15, and a minimum of 1 Endpoint; configured for host control transfers. An Endpoint is small amount of physical storage at either end of the transmission pipe for holding data during the transmission process. Each Endpoint has a separate IN and OUT buffer and can be viewed as the hardware interface from the perspective of the application developer.

The firmware is responsible for packaging and transmitting the data between Endpoints over “pipes”. A pipe is a logical connection between the Endpoints of the host and peripheral device. The USB 2.0 specification defines four different types of pipes for the transfer of data; Control, Interrupt, Bulk, and Isochronous.

Control transfers are the path through which the host and peripheral communicate. The enumeration process and subsequent management related communication are conducted using the Control pipe and because of this, Control transfers on Endpoint 0 are mandatory for all USB devices. Any additional Endpoint and transfer implementations are optional and application specific.
Interrupt transfers, unlike the other pipes, generate an internal flag signaling that a particular peripheral requires attention. All other transfers immediately push data onto the bus if there’s room. Endpoint status flags are read periodically by the host, so Interrupt transfers have a built in latency and are best suited for devices that require unidirectional communication that is not time critical; e.g. a USB mouse or keyboard.

Isochronous transfers are specifically designed for time sensitive, high bandwidth transfers; e.g. audio or video data. Isochronous transfers are guaranteed access to a majority of the USB bandwidth, but not guaranteed delivery. Isochronous transfers are best suited for applications which are tolerant of data errors and missing packets. Isochronous transfers are only support in Full and High speed USB devices.

Bulk transfers on the other hand, are not guaranteed bandwidth, but are guaranteed delivery. Most USB printers use bulk transfer to transmit data between the host and the printer. The Bulk transfer is best suited for bursts of data which may not be time critical, but are error intolerant. Bulk transfers fill in un-allocated bandwidth on the bus after all other types of transmission have been allocated.

The USB is host centric; every transmission across any Endpoint using any transfer type is initiated and managed by the host. The USB specification defines exactly how the bandwidth of the USB is allocated between transfer types, but is not covered in the document. For more information, please refer to the USB specification located here. Very simply, all transmissions, regardless of type and direction, consist of a Token Packet, Data Packet, and
a Status Packet. The Token Packet contains information about the data or payload to follow, the Data Packet is the actual data to be transmitted, and the Status Packet acknowledges transmission and provides a means of simple error correction. However, from the application developers’ perspective, these details are invisible.

As far as accessing the USB, the application developer need only concern themselves with understanding a device configuration and selecting the proper driver for accessing that device. For each driver, there is a supporting DLL that will allow access programmatically to the driver, allowing the developer to take advantage of the USB.
Review of the State of Art

There are many hardware related solutions on the market aimed at converting manufactured or custom peripherals from RS-232 to USB. Dallas Semiconductor manufactures a USB PCB add-on that connects to existing projects through a microcontroller’s SPI unit. Micro Digital manufactures a product that implements the USB stack in specialized hardware to emulate serial communication, mouse communication and or mass storage communication. At a higher level, companies like Ram Electronics sell USB to DB9 interconnects that simply allow you to plug your serial cable into a USB cable. On the software side, chip manufactures like Microchip, Amtel, and FTDI supply Windows based device drivers that emulate serial communication through USB; thus requiring only a cable change from a 9 pin serial cable to a standard 4 pin USB cable.

While these solutions are easy to implement, they do not take advantage of the transfer speeds that USB can deliver. Every proposed conversion from RS-232 to USB is ultimately restricted by the maximum bandwidth of the RS-232. For this reason, this project specified true USB communication and not serial emulation through USB.
Materials

The USB capable peripheral specified in this project is built around the Microchip PIC 18F2550/4550 series microcontroller. The 18Fx550 series chips are 8 bit microcontrollers with 32K flash program memory, 24-35 I/O pins, 10 ADC channels, and over 2048 bytes of data memory. In addition, the 18Fx550 has a built in Full-Speed USB 2.0 transceiver, 1KB dual port RAM, 16 Endpoints (both in/out), internal pull-up resistors on D+/D-, and operates at 48 MHz or 12MIPS. At under $10 a piece and the size of a dime, the 18Fx550 series microcontroller is the perfect microcontroller for instrumentation where space and power are limited.

As a development platform, the Microchip PICDEM 2 Plus demonstration board was used. (see appendix B) This board offered the widest feature set for general PIC microcontroller development; many more than the USB specific development board contained. For instance, the PICDEM 2 Plus

- Accepts 18-, 28-, 40-pin PIC 18 and PIC16 series devices.
- 2x16 LCD
- 4 LED’s
- 2 Pushbuttons
- Temperature Sensor, Potentiometer, Buzzer
- Prototyping Area
All firmware development was written in C using MPLAB IDE version 7.31 and MPLAB C18 version 3.02. C18 is a full featured ANSI C compiler for the PIC18 family of 8 bit microcontrollers. For hardware debugging, the Microchip ICD2, an In-Circuit Debugger, was used to interface with the demonstration board. The Microchip Bootloader demonstration project was also used and can be downloaded here.
Methods

Hardware Development

The PICDEM 2 Plus demonstration board includes a 4 MHz crystal that drives the internal clock. However, the USB specification requires a 48 MHz high precision clock for Full-Speed USB operation. The 18Fx550 has a fully configurable oscillator, accepting a wide range of crystals. Because Microchips USB development platform uses a 20 MHz crystal, one was added into secondary oscillator socket (Y1) on the PICDEM 2 Plus and 22pf capacitors were added into C5 and C4. The removal of J7 is required to disconnect the primary oscillator socket Y2. Please note the chip speed does not have to run at the 48 MHz. The 18Fx550 has a separate clock branch for the USB clock. The postscaler configuration was adjusted to set the core chip clock speed to 24 MHz.

While the PICDEM 2 Plus demonstration board accepts the 18F series chips that support Full-Speed USB, it does not include the necessary hardware to implement it. In the prototyping area, a female USB B style receptacle was attached. Power for the board was supplied by the 9 volt batter and not the USB 5+ line.

The PICDEM 2 Plus draws more than 100ma and some laptops will not supply this amount of current on the USB; so pin 1 was not attached to the board. Pins 2 and 3 were connected to D-/D+ respectively and pin 4 was connected to ground. A diagram of the minimal
schematic can be found in Appendix B along with a photograph of the PICDEM 2 Plus used in this experiment.

NOTE: RC5 (D+) on the PICDEM 2 Plus is also connected to 5+ volts through R8, U5, and U4. These components must be removed from the board for successful operation. If a voltage exists on pins RC4 or RC5 before the enumeration process, Windows will report that a malfunctioning device has been attached to the bus.

Bootsloader, a Microchip supplied example of USB communications, was initially used to test the hardware and form as skeleton code for the rest of the project. Because this code was written for the USB development board and not the PICDEM 2 Plus, changes were made to the firmware to match the existing hardware. The LEDs used by the firmware to communicate state information were assigned to port D. On the PICDEM 2 Plus, the pins are assigned to port B. All references to port D were changed to port B.

The USB development board also incorporates circuitry that senses the proper power source from which to drawn the board’s operational 5+ volts. The different options include 9 volt battery, 120 volt A/C adapter, and USB supplied 5+ volts. Because the PICDEM 2 Plus lacks the necessary circuitry, lines 57 to 72 were removed from the file io_cfg.h.

Please note, the PICDEM 2 Plus draws 100ma and overloads some older laptop USB ports. When the USB 5+ power supply is overloaded, Windows will disconnect the port complete and no harm will be done to the computer. On the PICDEM 2 Plus development board, a 9
volt battery was used and will last approximately 3 or 4 hours. Once the firmware was changed to match our hardware, the 18F4550 was programmed with the Bootloader firmware and attached to a computer running Windows XP.

Upon initial connection to the host, the firmware sits inside a “program” loop. This is an empty loop where general operational code would be placed. To enter “boot mode”, RESET / MCLR was pressed and held while RB4 was pressed and held. RESET / MCLR was then released followed by RB4. The delayed release of RB4 enabled the firmware to skip the program loop and enter into the main USB service loop.

Once in the main USB service loop, Windows will detect the newly “attached” device and look for the required device driver. If Windows cannot find a device driver, the user is prompted to supply one. The mchusb.sys is the Microchip supplied device driver that communicates with the USB hardware on the PC. The accompanying DLL, mcusbapi.dll, is used at the application level to communicate with the device driver (mchusb.sys) when access to the USB is required.

While the Bootloader firmware was used as a starting point in the project, it was enhanced to meet project specific goals. Most of the Bootloader code was removed and replaced by new project specific functionality that mainly focused on the management and transmission of analog data into Endpoint1.
Windows Application Development

The host application was written in C#, using Visual Studio 2005. Other options were considered and included, Visual Basic 6, VB.NET, C++, MFC, and Delphi. .Net offered the most flexibility, ease of use, and the lowest learning curve for those who will be developing applications in the future, so it was chosen as the platform of choice for this project.

Perhaps the most time intensive portion of this project was spent choosing the best USB library; i.e. DLL. There are several choices and each varied in stability, functionality, and extensibility. From a PIC specific perspective, PICdotNET is a free library written in VB.NET/C# that offered what seemed to be simple routines for accessing USB enabled PIC microcontrollers’. However, the supplied DLL did not work properly and did not include the source code or much documentation.

Another option was LIBUSB. LIBUSB is an open-source USB library developed for Linux and is currently being ported over to the Windows platform as Libusb-win32. Libusb-win32 remained under development throughout this project and documentation was scarce. A C# wrapper around the Libusb-win32 was also in development, but the library had bugs and again suffered from no documentation.

There are also driver/library combinations that can be purchased from companies like Tetradyne, Thesyccon, and EnTech. These companies produce general USB drivers, the
associated firmware, and necessary library files in various languages. The price for this type of solution is around $1000 and can include extra costs per unit distributed.

Of course every copy of Windows includes the necessary device drives for most USB devices. These drivers are not always found in the default Windows installation and downloading the Windows DDK is often required. While this was seen as a stable solution, support was scarce. Ultimately, the Microchip supplied driver/API combination was chosen. This combination offered the most stable learning environment, quickest implementation time, the best documentation and largest knowledge base.

MCHUSB.DLL is a library file supplied by Microchip that worked in conjunction with their USB device driver; MCHUSB.SYS. This library was written in Borland C++ and compiled for non-Win32 systems. The first step in the application development phase was to recompile the DLL into a Win32 compatible library file. To recompile the library, Borland’s C++ Builder was installed and the following string was used at the command line prompt to create a Win32 compatible DLL; ”C:\Borland\BCC55\Bin\ enter bcc32.exe -WD -ps -a1 ”;

–WD creates a GUI DLL were all functions are exportable
–ps specifies the standard calling convention as used in Windows
-a1 aligns the code by bytes
To use the newly complied mpusbapi.dll library in .Net, a reference to the library was added into the project. In the .Net environment, Win32 calls are not managed in the same way as C# and VB.Net code are. In order to call Win32 functions from .Net, the class definition that uses the mpusbapi.dll library must include the “unsafe” keyword, indicating to the CLR, the Common Language Runtime, that what follows is a section of unmanaged code. Each function in the mpusbapi.dll must also be independently imported and declared static extern in the class. A list of all the available functions can be found in mpusbapi.h. An example important statement can be seen below.

```csharp
[DllImport("mpusbapi.dll")]
private static extern DWORD _MPUSBGetDLLVersion();
```

Figure 2 - DLL Function Import Example
For initial testing purposes, a very simple application was created that toggled the LEDs on the PICDEM 2 Plus. This simple exercise was beneficial in learning the structure of the USB data-packet and how to interact with the peripheral through the library. This example also served as a good introduction to multi-IDE debugging techniques; debugging firmware and application software at the same time. The mpusbapi.h library supplies the following small set of commands:

MPUSBGetDllVersion – Returns the current user define DLL version

MPUSBGetDeviceCount – Returns the number of devices with a matching VID & PID

MPUSBOpen – Returns the handle to the endpoint pipe with matching VID & PID

MPUSBRead – Returns content from the specified endpoint

MPUSBWrite – Writes data to a specified endpoint

MPUSBClose – Closes an opened handle to an endpoint
When communicating with the microcontroller, the data packet structure is a maximum 64 bytes long, with 5 bytes of overhead reserved for the firmware instruction set. See MCHPFSUSB, the example project from Microchip for more details.

Every CMD is defined (enumerated) in the BOOT_DATA_PACKET structure. Any new functionality is assigned a CMD value and the resulting functions are added to the large switch statement in the BootService() routine. Please see the MCHPFSUSB example project from Microchip for more details. To communicate with the PIC microcontroller from C#, the communication pipes must first be opened for both inbound and outbound traffic.

```c
_MPUSBOpen(dwInstance, vid_pid_boot, out_pipe, 0, 0);
_MPUSBOpen(dwInstance, vid_pid_boot, in_pipe, 1, 0);
```

To send and receive data using the USB, the following commands are used from within C#.

```c
_MPIUSBWrite(usbPipeOut,(void*)SendData,SendLen,&SentDataLen,SendDelay)
_MPIUSBRead(usbPipeIn,(void*)RecData,ExpectedRecLen,ReceiveLength,ReceiveDelay)
```
The write call identifies which Endpoint is used for communication and an array, SendData, which contain firmware commands. The first byte of SendData is CMD and is used by the firmware to identify which function to call within the firmware. The next byte, SendLen, is used to store the number of bytes being sent. The third value is a pointer to number that represents the number of bytes of data to be sent. The firmware will check this number against what it received to determine if the message was corrupted along the way. The last value is delay length. If the instruction is not executed within this amount of time, the function will return a 0, or transmission failed. Reading data from the device is just as easy; pick an Endpoint, set up the byte array with the appropriate CMD, storage space and expected length. The firmware will return the requested data in the array supplied. If the data requested is longer than the 64 bytes, multiple transmission will be done automatically by the firmware until all data has been sent.

After general communication with the PICDEM 2 Plus was established, alteration to the firmware was essential in order to meet the requirements of the product specification. To model the conditions of a real-time data acquisition device, a photo resistor was added to the PICDEM 2 Plus as a means of generating a constant stream of analog data. The constant analog stream of data will be used to measure the operational bandwidth of the peripheral. This experiment has not been conducted at this point and will be saved for future work.
Results

The minimal circuit diagram, as shown in Appendix B, has been tested and can be implemented in Microchip based projects that support USB. The difficulties experienced with the hardware implementation were strictly related to the use of the PICDEM 2 Plus. As a stand alone circuit, the hardware is very simple to implement and can be done so in a very small package using the 4550-I/PT.

No documented evidence of using the PICDEM 2 Plus demonstration board as a USB development platform exist and therefore its use required much more investigation and troubleshooting, ultimately resulting in project delays. Even though the project did not entirely meet the specified goals, much was learned about hardware and software interfacing as well as the use of the oscilloscope as a hardware troubleshooting tool.

Even though the peripheral was not integrated into a laboratory instrument, some limited bandwidth testing was performed. Using a USB monitor, small bursts of data were measured just over 100Kbs using Bulk transfers on Endpoint1. In order to fully understand the transfer capabilities, more work is required on the firmware and application software. To see experimental bulk read and bulk erase transfer speeds please refer to Appendix C.
Future Work

Work is ongoing and will focus on finishing the application software and peripheral firmware. Once completed, constant analog data will be used to measure the maximum sustainable transfer rate between peripheral and host. If the transfer rates provide enough bandwidth, a new prototype instrument will be developed to be used to measure the acceleration of pendulums.

One possible project improvement of interest includes the design and implementation of a configurable set of graphical widgets that can be used to display captured data in real-time, manipulate, in a spreadsheet like fashion, data in real-time, and save it to different formats, e.g. XML, for offline processing.
Appendix A: Windows Enumeration Process (UML)
USB Enumeration Process on Windows Operating Systems

- **USB Hub**
  - Device Attached
  - Voltage Change D-/D+?
  - Report Event (interrupt endpoint)
  - Get_Port_Status()
  - Hi/Low Speed Device?
  - Report Device Speed
  - Set_Port_Feature() / Reset Device
  - High Speed Support? JK Chirp
  - Response to K/J Chirp
  - Get_Port_Status()
  - Completed Reset State Bit
  - Get_Descriptor()
  - Return Device Descriptor
  - Set_Address()
  - Assign Address
  - Get_Device_Descriptor()
  - Return Device Descriptor
  - Get_Interface_Descriptors()
  - Return Interface Descriptors
  - Get_Endpoint_Descriptors
  - Return Endpoint Descriptors
  - Get_Other_Descriptors
  - Return Other Descriptors

- **Host**
  - Power State
  - Default State
  - Address State
  - Configured State
  - Bus Monitoring

- **Windows OS**
  - Search INF files for Vendor / Product ID #'s
  - Assign / Load Driver

- **Device Driver**
  - Process Descriptors

**Configured State**
Adapted from USB Complete; Jan Axelson
Appendix B: PIC USB Minimal Schematic
Minimal Schematic for USB (18F2550/4550)

1 (MCLR) 28
2 (RA0) 27
3 26
4 (RB4) 25
5 (RB3) 24
6 (RB2) 23
7 (RB1) 22
8 (RB0) 21
9 (OSC1) (VDD) 20
10 (OSC2) (VSS) 19
11 18
12 17
13 (D+) 16
14 (VUSB) (D-) 15

S1 RESET

10k

22pF

20MHz

1M

22pF

.47uF

5+

0.1uF

43uF
PICDEM 2 Plus

- Remove oscillator and J7
- 20MHz Crystal
- Remove U5, U4, & R8
- 18F4550
- RB4 Button
- Photo-Resistor
- USB Connection
Appendix C: Bandwidth Testing
Bulk Erase

- Bulk memory erase on Endpoint 1
- 33805 bytes sent from peripheral to host

Bulk Read

- Bulk Memory read on Endpoint 1
- 35,973 bytes sent from peripheral to host

NOTE: These results are used to gain an understanding of the basic bandwidth characteristics of the peripheral using Microchip supplied code. Once the software is completed, more in-depth testing will be conducted to determine the actual bandwidth characteristics.
References

Axelson, Jan. *USB Complete: Everything you need to Develop USB peripherals*. Madison: Lakeview Research LLC, 2005


Microchip Technology Inc. [Internet]. Chandler (AZ): c2006. Available from www.microchip.com
