GENERIC USB DRIVER

Manthan Patel
B.E., Gujarat University, India, 2007

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL AND ELECTRONIC ENGINEERING

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010
GENERIC USB DRIVER

A Project

by

Manthan Patel

Approved by:

__________________________________, Committee Chair
Jing Pang, Ph.D.

__________________________________, Second Reader
Fethi Belkhouche, Ph.D.

___________________________________
Date
Student: Manthan Patel

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

_________________________,   Graduate Coordinator      ___________________
Preetham Kumar, Ph.D.                                   Date

Department of Electrical and Electronic Engineering
Abstract

of

GENERIC USB DRIVER

by

Manthan Patel

USB has become one of the most important technologies currently as it offers distinct advantages over previous I/O interfaces such as RS-232 serial port. USB communication is one key technology in our daily interaction with electronic devices. Application of USB communication ranges from Personal Computers (PC) to handheld devices such as mobile phones, memory devices, PDAs, and wireless devices. This project will provide knowledge on working of USB communication protocol.

The initial part of this report focuses on overview of USB communication and also architectural specification of USB. It provides platform for first time readers who are not familiar with technical details of USB communication. It provides readers the basic platform before understanding the design. Later part of the report discusses how a USB compatible device can be implemented.

As a USB compatible hardware, a PIC microcontroller based device has been designed for this project. When connected to the PC, the device communicates with the
PC using USB cable and communicates through USB cable to perform assigned task. PIC microcontroller has built-in USB communication peripherals.

____________________________, Committee Chair
Jing Pang, Ph.D.

____________________________
Date
ACKNOWLEDGEMENTS

First, I would like to take this opportunity to thank everyone whose contribution played significant role in implementing this project. I heartily thank my project advisor Dr. Jing Pang for providing me this wonderful opportunity to work on this project of designing Generic USB Drivers and giving me a great opportunity to work on USB protocol. This project would not have been possible without the continuous support and encouragement from Dr Pang. Dr. Pang has always been there to give guidance and support while working on this project. She has always encouraged me to work hard and focused to overcome challenges that we faced during project. Her vision always inspired me to work harder to accomplish the vision she had. She has always been enthusiastic about sharing her knowledge in the field of embedded system designs which played an important role in completion of this project. I would also like to thank Dr. Pang for always being patient and helpful towards the successful demonstration of PIC microcontroller based USB communication design.

I would also like to thank second reader of my project, Dr. Fethi Belkhouche, for reviewing my report and providing me his valuable guidance to improve my project report. His advice and assistant will enable reader to understand the project in better way.

Finally, the continuous strength and warmth I got from my family members and friends were invaluable and they drove me on through the critical phase of the project.
TABLE OF CONTENTS

Acknowledgements........................................................................................................vi

List of Tables................................................................................................................ix

List of Figures................................................................................................................x

Chapter

1. INTRODUCTION......................................................................................................1

2. UNDERSTANDING USB PROTOCOL.................................................................3
   2.1 Introduction.........................................................................................................3
   2.2 Architectural Overview of USB.........................................................................4
   2.3 USB Data Flow Model.......................................................................................5
   2.4 USB Packet Types.............................................................................................9
   2.5 Signaling of USB..............................................................................................11

3. HARDWARE PROFILE.........................................................................................13
   3.1 PIC18F4550 Microcontroller............................................................................13
   3.2 USB Compatibility of PIC18F4550.................................................................16
   3.3 USB Status and Control...................................................................................16
      3.3.1 UCON – USB Control Register...............................................................17
      3.3.2 UCFG – USB Configuration Register......................................................18
      3.3.3 USTAT – USB Status Register.................................................................19
   3.4 Oscillator and Clocking.....................................................................................20
   3.5 USB Cable.........................................................................................................22
LIST OF TABLES

Table 2.1 Possible Packet ID Values for Different Fields of USB Data......................8
Table 3.1 Brief Description of Pinout of PIC18F4550...........................................15
Table 3.2 Suggested Capacitor Values for Different Oscillators...............................21
LIST OF FIGURES

Figure 2.1 Tiered Star Topology of USB...............................................................5
Figure 2.2 TOKEN Packet..................................................................................9
Figure 2.3 Data Packet.....................................................................................10
Figure 2.4 Handshake Packet..........................................................................10
Figure 2.5 Start of Frame (SOF) Packet............................................................11
Figure 3.1 Pin Diagram of PIC18F4550...............................................................14
Figure 3.2 UCON – USB Control Register.......................................................17
Figure 3.3 UCFG – USB Configuration Register.............................................19
Figure 3.4 USTAT – USB Status Register........................................................20
Figure 3.5 Oscillator Operation in PIC Microcontroller.................................21
Figure 3.6 USB Type-A to Type-B Connector Cable........................................22
Figure 3.7 Wiring of USB B-Type Connector....................................................23
Figure 4.1 Block Diagram of Microcontroller Based Device............................24
Figure 4.2 Visual Studio Form when the Device is Not Connected...............27
Figure 4.3 Flow Chart for Software.................................................................28
Figure 4.4 Snapshot Showing USB Input Device is Ready to Use...............29
Figure 5.1 Device Connected with All Buttons Enabled................................31
Figure 5.2 Device Having Corresponding State When Number “2” is Sent.......32
Chapter 1

INTRODUCTION

Interface of external devices to the computer is playing a key role in the applications of computer in today’s world. Thus, the needs for high speed I/O protocol which can enable the devices to communicate faster with computers become more important. Due to increased importance of high speed I/O interface, developing a standard interface which can satisfy rapidly increasing memory size and high computing power becomes more vital. Input-output interfaces like parallel port and serial port are lacking in terms of performance for high-speed applications. Also, they require a large number of wires to be configured between two devices and thus lead to a lot of complications. In 1994, a group of seven companies came together to develop Universal Serial Bus (USB) standard for peripheral devices which makes the communication between peripheral devices and Personal Computer (PC) easier. As a result, the USB 1.0 specification was introduced in early 1996. Since then, the technology has advanced significantly over period of one decade to meet the requirements of devices used today. It provides high bandwidth, easier interface, and reliable high speed data transfer between devices.

USB can connect computer peripherals like keyboard, mouse, printer, digital cameras, camcorders, flash drives, external hard drives and media players. Though USB standard was developed for the personal computers only; it has found its place in other electronic devices such as PDAs, gaming console, and smart phones. Also, some of these
devices can use USB as a power cable to charge their batteries. Technology like plug-and-play which is supported by USB standard makes development of external memories more viable. Thus, the importance of USB standard has grown enormously. This project is about developing a USB compatible device that can communicate with the personal computer via USB port.

Chapter 2 of this report focuses on USB protocol and its architectural overview. It also includes technical details of USB standards such as USB data flow model, its electrical characteristics, and types of USB connectors.

Chapter 3 explains the hardware profile of the device which is designed for USB application. It discusses how a USB compatible device can be designed using PIC microcontroller.

Chapter 4 focuses on block diagram of the hardware developed. This chapter discusses the operation of this project and how different component interact with each other when connected together.

Chapter 5 spotlights the results of this project. It covers all scenarios implemented when the device actually connects the computer and how the computer can communicate with the device using firmware.

Chapter 6 gives conclusion of the project and lists improvements that can be done in the future.
Chapter 2

UNDERSTANDING USB PROTOCOL

2.1 Introduction

Universal Serial Bus (USB) as the name suggests is a serial bus. Main advantage of a USB bus is that it uses only four signals only to communicate. USB cable consists of four shielded wires – two wires for power supply (+5V and GND), while the remaining two wires are data in and data out (DIN and DOUT) which is used for communication. The Universal Serial Bus is host controlled. This means that the host is responsible for undertaking all transactions and allocating bandwidth to different devices which are connected. It is necessary to understand meaning of host and devices in terms of USB communication. The host in most cases is the personal computers. The peripherals that connect to the computer via USB port are called devices. Hence when we connect a device to the PC, PC is responsible to initialize and manage communication with the device. USB uses NRZI (Non Returning to Zero Invert) encoding to send data with a sync field to achieve the synchronization between host and receiver clocks. There can be only one host device per bus. USB specification does not support more than one master on a single bus. Though, new advancement in USB standard 2.0 allows two devices to negotiate the role of master on a single bus, it must be noted that only one of them can perform the task of the host as per the specification.
2.2 Architectural Overview of USB

Older USB version 1.1 supported two different operating speeds, a low speed mode of 1.5MBits/s and a full speed mode of 12MBits/s. The newer version USB 2.0 supports data transfer at a rate of 480MBits/s. It is interesting to note that USB 2.0 has backward compatibility with older USB 1.1 version. Thus, Universal Serial Bus has three operating speeds:

High Speed – 480MBits/s
Full Speed – 12MBits/s
Low Speed – 1.5MBits/s

USB communication consists of a host device, root hub and peripheral devices. The host is a Personal Computer (PC) or a computer that contains a USB host controller and a root hub. These components are responsible to form the communication channel between Personal Computer (PC) and peripherals connected. Host controller formats the data for transmitting on the bus and also translates the data coming from the device so that the operating system understands the data. The host controller performs key functions of managing the communication on the bus. The root hub, which connects to the host controller, has one or more connectors for attaching more devices. 127 different devices can be connected on a USB bus.
As figure 2.1 shows, hub extends the connectivity to more devices to connect further. USB protocol can handle up to five external hubs in series, and up to 127 peripherals and hub including the root hub connected to it. As figure 2.1 shows, the bus connection on USB bus is which is known as tiered star topology. At the center of each star is a hub. Each point indicates the device that connects to a port or a hub. These devices connect in a fashion which is referred to as tiered star topology.

2.3 USB Data Flow Model

In a RS-232 communication or any other similar serial interface, format of the data being sent is not defined. While USB is made up of several layers of protocol in which the data can be sent. It is essential to note that user of any USB device still remains
unaware of these layered communications and act as a transparent layer to the user. As we discussed before, USB is a host controlled bus in which host initiates all transactions and is also responsible for managing the communication on the bus. Based on that, each USB transaction can be divided in three elements: token packet, data packet (optional), and a status packet.

The token packet is the first packet generated by the host which acts like a header that tells the device what to expect to be followed. It contains the information about what type of transfer the host expects to do – a read or a write from the device. It also contains the information about start and endpoint address of the device. The following data transfer is generally a data packet carrying the payload followed by a handshaking packet which reports if the token or data is received successfully or not. It is important to know about the common packet fields of USB data first before going in to further details of these packets.

Data on the USB bus is transmitted as LSB (Least Significant Bit) first. It is very important to have brief idea about the contents of a USB packet. Any USB packet is constructed from different field types. The packets are then put together into frames to create USB messages. Below is a brief description of different field types of packet:
• **Sync**

All packets of data on a USB bus must start with a sync field. The Sync field is used for the clock synchronization between the transmitter and receiver. Sync field is 8 bits long out of which the last two bits indicate the start of PID field.

• **PID**

PID field indicates the type of packet which is being sent. PID is essentially the Packet ID field. Table 2.1 shows the possible values of PID field. Though the table 2.1 lists the PID values being 4 bits long, the field itself is 8 bits long. Remaining four bits contains complemented value of PID to rectify any error in transmission.

• **ADDR**

This is the address field used for USB compatible device. Address which is contained in this field indicates which device the data is designated for. This field has been kept 7 bits long to support USB connectivity to as many as 127 different devices at a time. One special address to note here is address 0 which is never assigned to any device connected. Any newly connected device must respond back to packets sent to address 0 until it is assigned a specified address.

• **ENDP**

ENDP is the endpoint field which is made up of 4 bits. Thus it can allow 16 possible endpoints.
• **CRC**

CRC is the Cyclic Redundancy Check bits which can be of two different types.

CRC5 is 5 bits long and is used with the token packet and the start of frame packet. Another type is CRC16 which is 16 bits long and is used with data packet.

<table>
<thead>
<tr>
<th>Group</th>
<th>PID Value</th>
<th>Packet Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>0001</td>
<td>OUT Token</td>
</tr>
<tr>
<td></td>
<td>1001</td>
<td>IN Token</td>
</tr>
<tr>
<td></td>
<td>0101</td>
<td>SOF Token</td>
</tr>
<tr>
<td></td>
<td>1101</td>
<td>SETUP Token</td>
</tr>
<tr>
<td>Data</td>
<td>0011</td>
<td>DATA0</td>
</tr>
<tr>
<td></td>
<td>1011</td>
<td>DATA1</td>
</tr>
<tr>
<td></td>
<td>0111</td>
<td>DATA2</td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td>MDATA</td>
</tr>
<tr>
<td>Handshake</td>
<td>0010</td>
<td>ACK Handshake</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td>NAK Handshake</td>
</tr>
<tr>
<td></td>
<td>1110</td>
<td>STALL Handshake</td>
</tr>
<tr>
<td></td>
<td>0110</td>
<td>NYET (No Response Yet)</td>
</tr>
<tr>
<td>Special</td>
<td>1100</td>
<td>PREamble</td>
</tr>
<tr>
<td></td>
<td>1100</td>
<td>ERR</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>Split</td>
</tr>
<tr>
<td></td>
<td>0100</td>
<td>Ping</td>
</tr>
</tbody>
</table>

Table 2.1 Possible Packet ID Values for Different Fields of USB Data [7]
• **EOP**

This is End of Packet field which is signaled by a SE0 (Single Ended 0) for approximately 2 bit times followed by a J for 1 bit time. It indicates that this is the last field of packet.

### 2.4 USB Packet Types

Now as we briefly discussed contents of a USB packet, we can discuss different types of USB packet in more details. As mentioned previously, there are four different types of USB packet: TOKEN packet, Data Packet, Handshake Packet and Start of Frame packet. Following is the format of packets they must confirm in order to be recognized as corresponding packet.

• **TOKEN Packets**

<table>
<thead>
<tr>
<th>Sync</th>
<th>PID</th>
<th>ADDR</th>
<th>ENDP</th>
<th>CRC5</th>
<th>EOP</th>
</tr>
</thead>
</table>

Figure 2.2 TOKEN Packet [4]

Figure 2.2 shows the format of TOKEN Packet. There are three different types of it:

1. **IN** – tells the USB device that host is initiating a READ transaction

2. **OUT** – tells the USB device that host is initiating a WRITE transaction

3. **SETUP** – used to start control transfer
• **Data Packets**

Data packets carry the payload of transaction and its format is shown in figure 2.3. There are two types of Data Packets: Data0 and Data1. They can be of variable length, depending upon the data.

<table>
<thead>
<tr>
<th>Sync</th>
<th>PID</th>
<th>Data</th>
<th>CRC16</th>
<th>EOP</th>
</tr>
</thead>
</table>

Figure 2.3 Data Packet [4]

• **Handshake Packet**

Figure 2.4 shows format of handshake packets. There are three types of handshake packets.

<table>
<thead>
<tr>
<th>Sync</th>
<th>PID</th>
<th>EOP</th>
</tr>
</thead>
</table>

Figure 2.4 Handshake Packet [4]

1. **ACK** – Acknowledge. Indicates that a data packet has been received successfully.

2. **NAK** – Indicates that the device is not able to send or receive data temporarily. Another application is in interrupt transaction to indicate the host about end of data.

3. **STALL** – Indicates that the device needs intervention from the host.
Start of Frame Packet

Start of Frame (SOF) identifies a new frame of data. SOF packet consists an 11-bit frame number is sent by the host every 1ms with a tolerance of ±500ns. Figure 2.5 shows format of SOF packet.

<table>
<thead>
<tr>
<th>Sync</th>
<th>PID</th>
<th>Frame Number</th>
<th>CRC5</th>
<th>EOP</th>
</tr>
</thead>
</table>

Figure 2.5 Start of Frame (SOF) Packet [4]

2.5 Signaling of USB

USB bus communicates differential pair of data on DIN (Data IN) and DOUT (Data OUT) pins. Data are encoded using NRZI (Non-Return-to-Zero-Invert) encoding. It is bit stuffed to ensure adequate transition in the data stream. In high speed mode, a transceiver activates internal current source which drives current from its positive supply on one of two data lines (D–, D+). This generates high-speed J or K state on the data lines. To signal a J, the current is driven on D+ line. While the current is driven on D– line to signal a K.

A built-in transceiver is used for low-speed and full-speed transmission. It is required to meet all specifications called out in USB 1.1 for low-speed and full-speed operation, with one exception. The exception is that in high-speed capable transceivers, the impedance of each output, including the contribution of RS, must be 45Ω. The line terminations for high-speed operation are created by having this driver drive D+ and D-.
to ground. (This is equivalent to driving SE0 in the full-speed or low-speed mode.) Because of the output impedance requirement described above, this provides a well-controlled high-speed termination on each data line to ground. A differential receiver is used to receive low-speed and full-speed data.

As discussed before, high speed mode of USB supports signaling at 480 Mb/s. For such high signaling rate, the cable is terminated at each end with a resistance to ground. The value of the resistor is usually kept to half the specified differential impedance of the cable, or a value of $45 \Omega$. This presents a differential termination of $90 \Omega$. 
Chapter 3

HARDWARE PROFILE

3.1 PIC18F4550 Microcontroller

The microcontroller used for this project is PIC18F4550 which is a PIC family microcontroller by Microchip. The microcontroller (PIC18F4550) incorporates a fully functional Universal Serial Bus (USB) communication module which is made fully compliant with USB specification Revision 2.0. PIC18F4550 supports USB communication in full speed and low speed mode. It supports USB communication for all data transfer types. It has an on-chip transceiver and a voltage regulator, at the same time it extends support to external transceiver and voltage regulators. Since one of the key distinguished features added in this project is its plug-and-play support, it uses on-chip transceiver and in built 3.3V regulator. Figure 3.1 shows the pin layout of the microcontroller used. Below are the key features of the microcontroller:

- USB V2.0 compliant device
- Supports both low-speed and high-speed operation of USB
- Supports Control, Interrupt, Isochronous and Bulk transfer types of USB
- On-chip USB transceiver and on-chip voltage regulator.
- Flexible oscillator structure which extends support to various operating frequencies
• In-Circuit Serial Programming (ICSP) which makes it easier to program

Figure 3.1 Pin Diagram of PIC18F4550 [1]

Package shown in the figure 3.1 is a 40-pin DIP package. PIC18F4550 has individual data port. Table 3.1 gives a brief description of functionality of the pins of the microcontroller. For simplicity, pins with similar functions are merged together. These pins are also used to extend support to alternate features which can be found in PIC18F4550 datasheet.
<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>Positive supply for logical and I/O pins</td>
</tr>
<tr>
<td>VSS</td>
<td>Ground reference for logic and I/O pins</td>
</tr>
<tr>
<td>VUSB</td>
<td>Internal USB 3.3V voltage regulator. Used to provide voltage source for USB communication.</td>
</tr>
<tr>
<td>OSC1</td>
<td>Crystal oscillator input or external clock source input to the internal clock operating circuit.</td>
</tr>
<tr>
<td>OSC2</td>
<td>Crystal oscillator output or clock output. Connects to crystal resonator.</td>
</tr>
<tr>
<td>RESET</td>
<td>A low level of this signal for more than one cycle puts whole chip in the reset condition even if the clock is not running.</td>
</tr>
<tr>
<td>RA0 – RA6 (PORT A)</td>
<td>7 – bit bidirectional port. Can be configured to provide digital input/output or analog input. Pins also have alternate functionality as shown in figure 3.1.</td>
</tr>
<tr>
<td>RB0 – RB7 (PORT B)</td>
<td>8 – bit bidirectional port. Can be configured to provide digital input/output or analog input. Pins also have alternate functionality as shown figure 3.1.</td>
</tr>
<tr>
<td>RC0 – RC7 (PORT C)</td>
<td>8 – bit bidirectional port. Can be configured to provide digital input/output or analog input. Pins also have alternate functionality as shown in figure 3.1. Port C has D+/D- pins used for USB communication in this project.</td>
</tr>
<tr>
<td>RD0 – RD7 (PORT D)</td>
<td>8 – bit bidirectional port. Can be configured to provide digital input/output or analog input. Pins also have alternate functionality as shown in figure 3.1.</td>
</tr>
<tr>
<td>MCLR</td>
<td>Master clear input for device. Active low hardware reset for the device.</td>
</tr>
</tbody>
</table>

Table 3.1 Brief Description of Pinout of PIC18F4550
3.2 USB Compatibility of PIC18F4550

A USB device is built to communicate with the personal computer using USB port. The device uses PIC18F4550 as its core element which is programmed to send and receive data on the USB bus with the help of the supporting hardware. PIC18F4550 embeds a Serial Interface Engine (SIE) which enables PIC microcontroller to communicate faster with the host. PIC family microcontroller embeds a USB supporting hardware to itself compatible with USB devices.

When connected to the PC, microcontroller performs enumeration task. Microchip provides product ID and vendor ID for its USB compatible microcontroller families. It can be used while programming the microcontroller so that it can perform initialization with the PC. This makes PIC18F4550 compatible for USB communication and thus making it an ideal choice for such a project.

3.3 USB Status and Control

There are control registers that configure and manage the operation of USB. It becomes essentially important to understand the register set and how it manages communication on USB. Following are the important registers of microcontroller which decides the way microcontroller communicates with the host via USB. These registers play a key role in deciding the data rate, setting interrupts on USB transactions and managing communication on USB channel. They are:

UCON – USB Control Register
UCFG – USB Configuration Register

USTAT – USB Transfer Status Register

3.3.1. UCON – USB Control Register

UCON register shown in figure 3.4 contains bits needed to control module behavior during USB data transfer. UCON register contains bits which are responsible to control following:

- Main USB peripheral Enable
- Ping-Pong Buffer Pointer Reset
- Control of the Suspend Mode
- Packet Transfer Disable

<table>
<thead>
<tr>
<th>U-0</th>
<th>R/W-0</th>
<th>R-x</th>
<th>R/C-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>U-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>PPBRST</td>
<td>SE0</td>
<td>PKTDIS</td>
<td>USBEN</td>
<td>RESUME</td>
<td>SUSPND</td>
<td>–</td>
</tr>
</tbody>
</table>

R = Readable bit   C = Clearable bit   W = Writable bit
U = Unimplemented bit ‘1’ = Bit is set   ‘0’ = Bit is cleared

Figure 3.2 UCON : USB Control Register [1]

Brief functional description of each bit of UCON register is as below:
Bit 7 and Bit 1 are unimplemented and they are read as a logic 0. Bit 6 is Ping-Pong Buffers Reset bit. When set to 1, it reset all Ping-Pong Buffer pointers to the even Buffer Descriptor (BD) banks. When set to 0, Ping-Pong Buffer Pointers not being reset. Bit 5 is Live Single-Ended Zero Flag bit. When set to 1, it makes Single-ended zero active on the USB bus. When set to reset, it means that no single-ended zero is detected. Bit 4 is Packet Transfer Disable bit. It enables/disables SIE token and packet processing disabled. It sets automatically set when a SETUP token is received. Bit 3 is USB Module Enable bit. Bit 2 activates resume signaling when set to ‘1’. Bit 1 is suspend bit which determines the USB module and supporting circuitry in Power Conserve mode or in normal operation.

3.3.2. UCFG – USB Configuration Register

UCFG register contains bits that are responsible to define the system level behavior of the USB module. These include following:

- Bus Speed (Whether to operate bus on full-speed or low-speed)
- On-chip pull-up resistor enable
- On-chip transceiver enable
- Ping-Pong buffer usage

Bit 6 of UCFG register is unimplemented and read as ‘0’. Bit 7 enables the eye pattern test supported by the device when set to 1. It disables the eye pattern test
supported when reset. Bit 6 indicates intervals during which the D+/D- lines are driving. Bit 4 enables or disables on-chip pull up depending on its value. Bit 3 enables digital

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTEYE</td>
<td>UOEMON</td>
<td>-</td>
<td>UPUEN</td>
<td>UTRDIS</td>
<td>FSEN</td>
<td>PPB1</td>
<td>PPB0</td>
</tr>
</tbody>
</table>

R = Readable bit  C = Clearable bit  W = Writable bit  U = Unimplemented bit  ‘1’ = Bit is set  ‘0’ = Bit is cleared

Figure 3.3 UCFG : USB Configuration Register [1]

transceiver interface and disables on-chip transceiver when set to ‘1’. Bit 1 and Bit 0 of UCFG register are used to configure even/odd Ping-Pong buffers. For any USB transaction, setting the values in UCFG register is a very important part as it configures the USB devices connected to the host.

3.3.3. USTAT – USB Status Register

USTAT is the register which indicates the current status of USB communication on the channel. It reflects the current state of transaction on USB bus. It contains bit which gives information on number of BDT updated and also the direction of last transaction.
### Figure 3.4 USTAT: USB Status Register [1]

<table>
<thead>
<tr>
<th>U-0</th>
<th>R-x</th>
<th>R-x</th>
<th>R-x</th>
<th>R-x</th>
<th>R-x</th>
<th>R-x</th>
<th>U-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>ENDP3</td>
<td>ENDP2</td>
<td>ENDP1</td>
<td>ENDP0</td>
<td>DIR</td>
<td>PPB1</td>
<td>–</td>
</tr>
</tbody>
</table>

- **R** = Readable bit  
- **C** = Clearable bit  
- **W** = Writable bit  
- **U** = Unimplemented bit  
- ‘1’ = Bit is set  
- ‘0’ = Bit is cleared

Bit 7 and Bit 0 of USTAT register are unimplemented and they read as a value ‘0’. Bit 6 to Bit 3 contains a four bit encoded number of last endpoint activity bit. Bit 2 has a value of ‘1’ when last transaction is an input transaction and it is reset to ‘0’ when the last transaction is output data or a setup packet transfer.

#### 3.4 Oscillator and Clocking

PIC18F4550 has an internal clock source which feeds the clock internally in case we are not using an external oscillator. PIC18F4550 has an integral USB module. Hence for any USB application it requires a highly precise and stable clock source. Thus it is necessary to provide a separate clock source which is compatible with both low-speed and high-speed operation of USB. PIC18F4550 includes a clock branch to support full-speed operation at 48MHz. A primary clock source drives the microcontroller. The microcontroller has built-in system of prescalers and postscalers to contain a wide range of operating frequencies.
Figure 3.2 shows the operation of an external oscillator connected to the microcontroller. Here, the values of capacitor C1 and C2 must be decided very carefully. If the value of capacitor is high, it increases the stability of the oscillator but it can lead to a slow start-up. Table 3.1 lists suggested values for capacitor. Clock plays a very vital role in USB communication.

<table>
<thead>
<tr>
<th>Crystal Frequency</th>
<th>Values of Capacitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4MHz</td>
<td>C1 = 27pF</td>
</tr>
<tr>
<td></td>
<td>C2 = 27pF</td>
</tr>
<tr>
<td>8MHz</td>
<td>C1 = 22pF</td>
</tr>
<tr>
<td></td>
<td>C2 = 22pF</td>
</tr>
<tr>
<td>18MHz</td>
<td>C1 = 22pF</td>
</tr>
<tr>
<td></td>
<td>C2 = 22pF</td>
</tr>
<tr>
<td>20MHz</td>
<td>C1 = 15pF</td>
</tr>
<tr>
<td></td>
<td>C2 = 15pF</td>
</tr>
</tbody>
</table>

Table 3.2 Suggested Capacitor Values for Different Oscillators
The device used for this project uses a 4MHz oscillator. Values of capacitor used are 27pF for both C1 and C2. This configuration gives a stable output for USB operation. To utilize internal prescalers and postscalers, values of bits CPUDIV1 and CPUDIV0 is set to ‘0’. Thus it supports full-speed operation of USB at 48MHz.

3.5 USB Cable

The USB cable used for this project has a USB Type-A connector on one side while the other end has a Type-B connector. The is shown in figure 14.

![USB Type-A to Type-B Connector Cable](image)

Figure 3.6 USB Type-A to Type-B Connector Cable [5]

The end with Type-A connector is traditional USB connector that goes in personal computers, it can be seen at right in figure 4.2. Figure 4.3 shows internal wiring of the other end with Type-B connector.
Figure 3.7 Wiring of USB B-Type Connector [4]
Chapter 4

BLOCK DIAGRAM AND OPERATION

4.1 Block Diagram

Figure 4.1 Block Diagram of Microcontroller Based Device
4.2 Microcontroller Based Device

The design used for this project is implemented on a breadboard. As mentioned earlier, the microcontroller used for the design is PIC18F4550. For this project, the microcontroller was configured to use a 4MHz external oscillator. A crystal oscillator was used for the design. A USB B-type connector is required at the device end to form communication channel. Two 27pF capacitors were used to stabilize the signal generated by the oscillator. Two 100nF capacitors were mounted on VCC and VSS pins of microcontroller. This avails the design with stable signals on both VCC and VSS (Ground) lines. 470pF capacitor was required so that the microcontroller can operate the internal USB circuitry. This capacitor on VUSB pin of microcontroller was used to regulate the USB voltage required by an on-board USB interface of microcontroller. This enabled device to draw its power from USB cable and it did not require external power source. A seven segment display was mounted on the board which displayed the number sent from the computer.

The PIC18F4550 microcontroller is self-powered. It drove required power from the host (PC) via USB cable which was connected to the VUSB pin of microcontroller. Thus, it did not require any external power supply which made the device easier to use and extended the portability. 470nF capacitor was required for the microcontroller to operate the internal USB circuitry as discussed before. Any capacitor with a rating of 5V or above should work with microcontroller’s internal hardware. It is important to note here that it is not essential to use only a 20MHz oscillator for USB communication. The
microcontroller supported flexible oscillator configuration if the value of internal register are set appropriately.

Once the hardware outline was fixed, mounting of components started for the device. Enough care was taken while mounting the components on the breadboard so that none of any open or short connection went unnoticed. Connection of USB B-Type connector also required special attention as any mistake in connection can affect the overall functionality of the project. It could sometime be severe enough to have permanent damaging effect on components like microcontroller. Wiring of USB B-type connector provided a good reference while connecting USB connector to the device. PICKit 2.0 programmer was used to program the microcontroller. The microcontroller supported In-Circuit Serial Programming (ICSP) which means it can be programmed serially on board. After mounting of all the components was done, the device was connected to the PC using USB cable.

4.3 Software Development

The software for this project consisted of two main parts. One was microcontroller programming while the other was script on PC side which run in Microsoft Visual Studio. Visual Studio script was responsible for sending user commands to the microcontroller device. PC was sending data on the USB bus along with the command using control transfer. Figure 4.2 shows the form which is developed in Microsoft Visual Studio 2010. When the microcontroller based device was connected with the PC, the PC realized that a device has been connected via USB port. This
changed the state of Visual Studio form shown in figure 4.2. The form recognized the connected device and changes the status of USB device field. As the device extended support to plug-and-play configuration, the microcontroller was responsible to install its drivers for the operating system so that the PC can recognize the hardware. Figure 4.3 shows the flow of software.

Figure 4.2 Visual Studio Form when the Device is Not Connected
Idle

When the device is connected to the PC

Enumeration and Initialization

When the device enumeration is done and device is recognized by the PC with its Product ID (PID) and Vendor ID (VID) values stored

Device is connected and ready to use

When user sends command using Visual Basic script and USB communication is working properly

Data communication

Data communication ends

When the device is disconnected from PC

Figure 4.3 Flow Chart for Software
Appendix A includes the main file for the microcontroller code. The firmware written for this device is supported by operating system Microsoft Windows 7 which is the latest operating system by Microsoft. In microcontroller, the values of PLL division bits (PLLDIV2:PLLDIV0) was set to 000. This was giving a division factor of 1 to 4MHz frequency signal of externally connected oscillator. The value of MCU division was set to 00 to provide microcontroller with internal clock frequency of 48MHz. Once the device had been connected and it was recognized by the PC as a compatible device, the PC started enumeration with the device along with all initialization needed to establish a USB communication channel. The PC recognized the device from its specific vendor ID and product ID field. These IDs are provided by the manufacturer of the device which is Microchip for PIC microcontroller.

Figure 4.4 Snapshot Showing USB Input Device is Ready to Use

After completion of initialization, PC displayed a window which showed that USB input device was installed and it was ready to use. This window is shown in figure 4.5. At this time, the device became a part of the computer hardware and a specific
address was assigned to the device for the PC to communicate with. This could be seen if we navigate to Start -> Control Panel -> Hardware and Sound -> Device and Printers.

Port B of microcontroller was configured as digital output port. This was done by setting value of register TRISB to all zeros. Setting value of TRIS register for any port configures it as a digital output port while setting a value of one configures it as a digital input pin. Port B was used to provide digital output for the seven segment display. Whenever the value of USB received buffer is updated, the microcontroller decoded the command and performed the task.

Visual Studio script was written for PC to be able to communicate with the device. Once the USB input device was installed, next was to run the script in Visual Studio. Running the script opened another window where it displayed that the device was properly connected and ready to communicate. Visual Studio communicated to the device via USB port. Visual Studio script sent signal via USB port to the device to change the status of LED when we press the button for “Change LED State”. The script will then wait for the device to acknowledge whether the command had been successfully executed or not. Once it confirms the action taken by the device, it shows it on the PC as LED state. It shows The script was sending continuous command over the USB bus to the device to monitor the state of the switch. The script detected any change of state in the switch and reflected it on the PC. When any numberical button on the PC was clicked, the script sent corresponding command to the device.
Chapter 5

RESULTS

5.1 Testing Hardware-Software Interface

When the device was first connected to the PC, the device did perform all initial tasks along with PC to establish a communication channel. Figure 5.1 shows a screenshot of the same event. At the same time Visual Studio script recognized the connection of the device.

Figure 5.1 Device is Connected with All Buttons Enabled
Figure 5.1 shows the Visual Studio Form once the device was connected to the PC. It changed the status of USB device to “Device is Connected and Now Ready to Use”. This window allowed user to click any numberical number on the screen and showed corresponding value on the connected device.

Figure 5.2 Device Having Corresponding State When Number “2” is Sent
When PC sent signal to the board to change the status of LED, the board also responded back and acknowledged whether it received and executed command or not. This was seen on PC by looking at LED state which toggles between “ON” and “OFF” each time “Change LED State” button was pressed corresponding the state of LED. This confirmed communication between microcontroller and PC. Also, the script continuously sent command to the device using USB cable which monitored the state of switch on the device. The device continuously sent the state of switch connected on pin 3 of microcontroller via USB. Visual Studio script confirmed the state of switch on window shown as “Switch State” which correspond to the state of switch on the other side. Whenever the state of switch was change on device’s side, the script immediately showed the result on PC side. Data communication between PC and the device was implemented at 12MBPS. This proved continous data transfer from device’s side to the PC in form of current state of pushbutton. This confirmed the uninterrupted connection between USB device to the PC and also continous data communication on Universal Serial Bus.
Chapter 6

CONCLUSION

The focus of this project was to establish USB communication channel between PC and a device that uses PIC microcontroller. Main objective is achieved, where I successfully demonstrated the results of the project. I implemented all scenarios using PIC18F4550 microcontroller based device.

As the project touches both aspects of digital design – hardware and software, it is surely a very highly regarded learning experience working on this project. I initially had some trouble with hardware interfacing with PC, although I managed to correct the mistakes I made in the design and thus improved my knowledge and skills on hardware–software interface. This project gives a great experience with the USB standard and its layered communication. This project can work like a stepping stone towards even deeper knowledge in USB communication. This project can work as a reference for a larger USB communication project which deals with large amount of data.

6.1 Future Improvements

Due to vast scope of this project, I keep this project limited to only one device per host. This can be extended to multiple devices which are connected to the same host. A design which can provide interface between more than one devices connected to a same host can be a very useful improvement in the design.
Another area of improvement is to communicate large amount of data. Though additional care of synchronizing the data on both PC and device side becomes extremely important. Synchronization in such communication can be very challenging and a great learning experience. Thus, using this project as a reference and keep adding a bit of challenging improvement can lead to deep knowledge of USB communication.
#ifndef MAIN_C
#define MAIN_C

// Global includes
#include <htc.h>

// Local includes
#include "usb.h"
#include "HardwareProfile.h"
#include "usb_function_hid.h"
#include "genericHID.h"

// PIC 18F4550 fuse configuration:
// Config word 1 (Oscillator configuration)
// 20Mhz crystal input scaled to 48Mhz and configured for USB operation
__CONFIG(1, USBPLL & IESODIS & FCMDIS & HSPLL & PLLDIV1);

// Config word 2
__CONFIG(2, VREGEN & PWRTDIS & BOREN & BORV20 & WDTDIS & WDTPS32K);

// Config word 3
__CONFIG(3, PBDIGITAL & LPT1DIS & MCLREN);

// Config word 4
__CONFIG(4, XINSTDIS & STVREN & LVPDIS & ICPORTDIS & DEBUGDIS);

// Config word 5, 6 and 7 (protection configuration)
__CONFIG(5, UNPROTECT);
__CONFIG(6, UNPROTECT);
__CONFIG(7, UNPROTECT);
// local prototypes
static void InitialiseSystem(void);
void ProcessIO(void);

// Main function
void main(void)
{
    InitialiseSystem();

    while(1)
    {
        // Check bus status and service USB interrupts.
        USBDeviceTasks();

        // Application-specific tasks.
        ProcessIO();
    }
}

// Initialise system
static void InitialiseSystem(void)
{
    ADCON1 = 0x0F; // Default all pins to digital

    // Configure ports as inputs (1) or outputs(0)
    TRISA = 0b00000010;
    TRISB = 0b00000010;
    TRISC = 0b00000000;
TRISD = 0b00000000;
TRISE = 0b00000000;

// Clear all ports
PORTA = 0b00000000;
PORTB = 0b00000000;
PORTC = 0b00000000;
PORTD = 0b00000000;
PORTE = 0b00000000;

#if defined(USE_SELF_POWER_SENSE_IO)
tris_self_power = INPUT_PIN;
#endif

// Initialize the variable holding the handle for the last
// transmission
USBOutHandle = 0;
USBinHandle = 0;

blinkStatusValid = TRUE;

USBDeviceInit();

LED0 = 1;
}

// Process input and output
void ProcessIO(void)
{

// If we are not in the configured state just return
if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;

// Check if data was received from the host.
if(!HIDRxHandleBusy(USBOutHandle))
{
    switch(ReceivedDataBuffer[0])
    {
        case 0x80: // Toggle the LED; Also Display 't' for Toggle
                    if (LED0 == 1) LED0 = 0; else LED0 = 1;
                    /*
                     * DATABIT0 = 0;
                     * DATABIT1 = 0;
                     * DATABIT2 = 0;
                     * DATABIT3 = 1;
                     * DATABIT4 = 1;
                     * DATABIT5 = 1;
                     * DATABIT6 = 1;
                     * DATABIT7 = 1;*/
            PORTB = 0x00;
            PORTB = 0xF8;
            break;

        case 0x91: // Display Number '1'
            /*
             * DATABIT0 = 0;
             * DATABIT1 = 1;
             * DATABIT2 = 1;
             * DATABIT3 = 0;
             * DATABIT4 = 0;
            */

    } // end switch(ReceivedDataBuffer[0])
} // end if(!HIDRxHandleBusy(USBOutHandle))
DATABIT5 = 0;
DATABIT6 = 0;
DATABIT7 = 0;/*
PORTB = 0x00;
PORTB = 0x06;
break;

case 0x92: // Display Number '2'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
        DATABIT0 = 1;
        DATABIT1 = 1;
        DATABIT2 = 0;
        DATABIT3 = 1;
        DATABIT4 = 1;
        DATABIT5 = 0;
        DATABIT6 = 1;
        DATABIT7 = 0;*/
    PORTB = 0x00;
    PORTB = 0x5B;
break;

case 0x93: // Display Number '3'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
        DATABIT0 = 1;
        DATABIT1 = 1;
        DATABIT2 = 1;
        DATABIT3 = 1;
        DATABIT4 = 0;
        DATABIT5 = 0;
        DATABIT6 = 1;
        DATABIT7 = 0;*/
    PORTB = 0x00;
    PORTB = 0x5B;
DATABIT6 = 1;
DATABIT7 = 0;*
PORTB = 0x00;
PORTB = 0x4F;
break;

case 0x94: // Display Number '4'
  // if (LED0 == 1) LED0 = 0; else LED0 = 1;
  /*  DATABIT0 = 0;
       DATABIT1 = 1;
       DATABIT2 = 1;
       DATABIT3 = 0;
       DATABIT4 = 0;
       DATABIT5 = 1;
       DATABIT6 = 1;
       DATABIT7 = 0;*/
PORTB = 0x00;
PORTB = 0x66;
break;

case 0x95: // Display Number '5'
  // if (LED0 == 1) LED0 = 0; else LED0 = 1;
  /*  DATABIT0 = 1;
       DATABIT1 = 0;
       DATABIT2 = 1;
       DATABIT3 = 1;
       DATABIT4 = 0;
       DATABIT5 = 1;
       DATABIT6 = 1;*/
case 0x96: // Display Number '6'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
    DATABIT0 = 1;
    DATABIT1 = 0;
    DATABIT2 = 1;
    DATABIT3 = 1;
    DATABIT4 = 1;
    DATABIT5 = 1;
    DATABIT6 = 1;
    DATABIT7 = 0;*/
    PORTB = 0x00;
    PORTB = 0x7D;
    break;

case 0x97: // Display Number '7'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
    DATABIT0 = 1;
    DATABIT1 = 1;
    DATABIT2 = 1;
    DATABIT3 = 0;
    DATABIT4 = 0;
    DATABIT5 = 0;
    DATABIT6 = 0;
    DATABIT7 = 0;*/
    PORTB = 0x00;
    PORTB = 0x7D;
    break;
PORTB = 0x00;
PORTB = 0x07;
break;

case 0x98: // Display Number '8'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
    DATABIT0 = 1;
    DATABIT1 = 1;
    DATABIT2 = 1;
    DATABIT3 = 1;
    DATABIT4 = 1;
    DATABIT5 = 1;
    DATABIT6 = 1;
    DATABIT7 = 0;*/
PORTB = 0x00;
PORTB = 0x7F;
break;

case 0x99: // Display Number '9'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
    DATABIT0 = 1;
    DATABIT1 = 1;
    DATABIT2 = 1;
    DATABIT3 = 1;
    DATABIT4 = 0;
    DATABIT5 = 1;
    DATABIT6 = 1;
    DATABIT7 = 0;*/
PORTB = 0x00;
PORTB = 0x6F;
break;

case 0x90: // Display Number '0'
    // if (LED0 == 1) LED0 = 0; else LED0 = 1;
    /*
     * DATABIT0 = 1;
     * DATABIT1 = 1;
     * DATABIT2 = 1;
     * DATABIT3 = 1;
     * DATABIT4 = 1;
     * DATABIT5 = 1;
     * DATABIT6 = 0;
     * DATABIT7 = 0;*/
    PORTB = 0x00;
PORTB = 0x7E;
break;

case 0x81: // Read the push-switch status
    ToSendDataBuffer[0] = 0x81;
    ToSendDataBuffer[1] = SWITCH0; // return the current switch state

    // Transmit the response to the host
    if(!HIDTxHandleBusy(USBInHandle))
    {
        USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
    }
    break;
case 0x82: // Read the LED status
   ToSendDataBuffer[0] = 0x82;
   ToSendDataBuffer[1] = LED0; // return the current LED state
    // Transmit the response to the host
    if(!HIDTxHandleBusy(USBInHandle))
    {
        USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
        break;
    }
    // Re-arm the OUT endpoint for the next packet
    USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
}
REFERENCES

   [http://www.usb.org/developers/docs/hs_usb_pdg_r1_0.pdf]
   [http://www.beyondlogic.org/usbnutshell/usb1.shtml]