Computer-aided design of an ergonomic computer mouse

Mohd Rapid Arifin
Iowa State University

Computer-aided design of an ergonomic computer mouse

by

Mohd Rapid Arifin

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Mechanical Engineering

Program of Study Committee:

Abir Qamhiyah, Co-major Professor
Donald Flugrad, Co-major Professor
Carolina Cruz-Neira

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Mohd Rapid Arifin

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

To God who gave me a chance to live in this world. He blessed me with a one-in-a-lifetime chance to come here and study with the best.

To my mom and dad who have been wonderful to me in my entire life. Without them I would not be here in first place. And to my entire family who give me a lovely support since I was a kid. Without them I would not be as capable as I am right now.

To all my professors and instructors, including those who taught me during my undergraduate and graduate level, especially to the Program of Study committee members, Dr Abir Qamhiyah, Dr Donald Flugrad, Dr Carolina Cruz-Neira and Dr Greg Luecke. They gave me a chance that I will not forget for my whole life.

To all my co-workers, classmates, housemates, staff in Mechanical Engineering Department and Virtual Reality and Application Center (VRAC), and last but not least to all my students in ME 436 Heat Transfer Lab. They taught me what a word 'friendship' and 'honest' means and it means a lot to me. It's been a privilege to work and be in a same room with these people.

May the God bless all the people I have mentioned above.

ABSTRACT

The primary purpose of this thesis is to explain a device which could be used as an alternative for a computer mouse. Instead of using a regular roller found in an ordinary mouse, the device uses a pressure sensitive sensor to control the computer cursor on the monitor.

The device is developed mainly for a personal computer with Universal Serial Bus (USB) capability. The computer should have an operating system of Microsoft Windows 98 or newer. The device does not need any additional driver, and it has a USB hot-plug-and-play feature. It uses a Human Interface Device (HID) driver provided by Windows.

The device mainly has two buttons (right and left) and is approximately 4″ by 3″ by 2″ in size. Users can press their fingers on to the device to control the cursor. The device will be small enough to be fit inside a person's palm. The area has four pressure sensors used to move the cursor to the left, right, upward and downward. The user can control some parameters, such as cursor movement rate, by just controlling the amount of force pressed on that area. The device will be made from a soft material with a hard box inside. All the necessary components will be placed inside the box. Only the sensors are outside the box, so that the user can control the sensors by squeezing the device. This would make it comfortable for users to operate the device.

CHAPTER 1: INTRODUCTION

Often engineers find themselves working with their desktop or laptop in their project presentation. Most of the presentations have something to do with engineering drawing, and they have to go back and forth from their standing spot to the desk just to move the computer cursor around. The engineering application program, such as AutoCAD and IDEAS are so difficult to operate without a computer mouse.

It is important to have a device, which works like a regular mouse, but can be operated even without a flat surface to slide the mouse on. This thesis will explain on how such a device can be designed.

Chapter 1 of this thesis covers the basics of the project, some definitions, and it also explains the basics of how a regular mouse works. It will give readers some idea about the similarities and differences between a mouse and this new device. Chapter 2 covers sensor selection of the new device and how it differs from the original design of a regular mouse. Chapter 3 explains the intelligence part of the project, or microcontroller. Chapter 4 presents how the data is transferred from the device to the host computer and also some signal conditioning. Most of the signal conditioning is being done by the microcontroller. Chapter 5 discusses the responsibility of the host computer, how the data is handled and some discussion about Universal Serial Bus (USB). The final chapter of this thesis, chapter 6, presents the final assembly of the new device and summarizes the final specifications of the device.

Ergonomics

The term ergonomics is based upon two Greek words: ergos means 'work' and nomos means 'the study of' or 'the principles of'. In other words, the ergonomics is the field of study that examines human behavioral, psychological and physiological capabilities and limitations [1]. From these capabilities and limitations, the new product can then be designed, or modified, to maximize productivity, worker comfort and overall efficiency. The primary objective of ergonomics is to improve human health, safety and performance.

Using regular mice in continuous basis can cause several problems to the users. Sometime, the users have to keep their arms straight for several hours. Some of the problems are neurovascular disorders and nerve disorders [5]. The most common neurovascular disorder is thoratic outlet syndrome. This is caused by the compression of nerve and blood vessels between the neck and the shoulder. The symptoms of thoratic outlet syndrome include numbness in the arm and finger. The most famous type of nerve disorder is carpal tunnel syndrome. This occurs when the tunnel containing the tendons, nerves, and blood supply to the hand is collapsed by repeated pressure to the underside of the wrist. It results in pain, numbness, and tingling in the hand.

In some other time, the users have to hold their arms up while using the mice when there is no, or little, space around the mice. This might cause neck, upper-back, and shoulder pain to some of the users [4].

Project Objective

The objective of this project is to develop a new input device that works like a regular mouse but does not need a flat surface to roll the ball on. This new device should be compatible with an already-existing driver. The device is designed to be compatible with a Windows-operated personal computer equipped with at least one Universal Serial Bus (USB) port.

Project Steps

1. Collecting information on regular mice and understanding how they work. This includes understanding the flow of its microcontroller firmware.
2. Gathering information about computer operation system (Windows and etc), data interface (PS/2, RS232, USB and etc), and driver of the mice.
3. Starting the project with eliminating the regular sensor in the mice and replacing them with a sensor that does not need a rolling surface.
4. Choosing a microcontroller that can be used to process the data from the sensors. This step includes writing firmware for the microcontroller.
5. Building necessary circuitry for the microcontroller, complete with the external components, if necessary.
6. Designing an outer shell of the device.
7. Assembling the device.

Literature Review

Input Device

An input device is a tool which is used to interact between the real world and the numeric world of computers. This input device will translate humans' 'language' to the one that can be understood by a computer. The 'language' could be numeric or alphabetic input, motion input, voice input, etc. This step includes using a sensor to transduce a mechanical input and transforms that into an electronic signal.

The flow chart below shows an example of how an input signal (measurand) looks when it travels from sensor to computer. In most cases, the amplifier and filter in the chart can be integrated into the microcontroller chip.

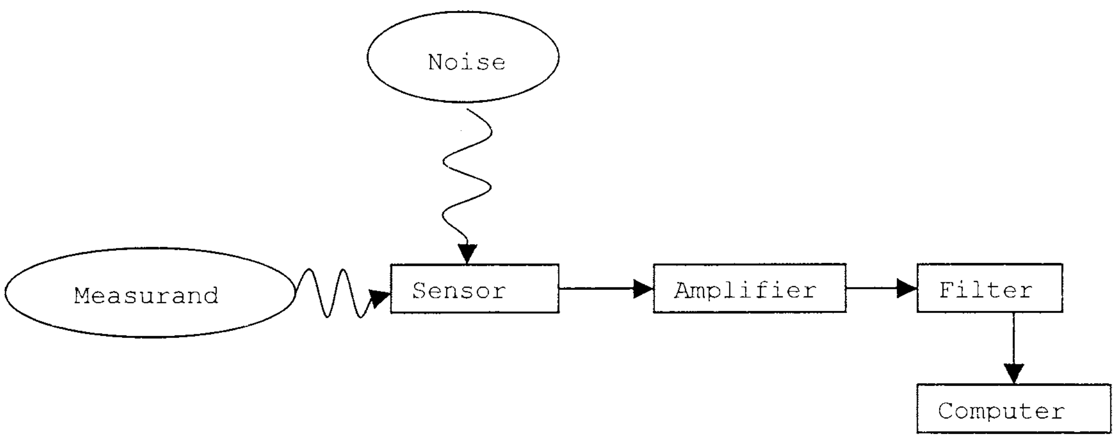


Figure 1-1 Signal Flow [10]

There are two ways in which modern computers receive input. The first way is that the computer gets input from users [8]. Some experts call this kind of input as 'interactive input'. This input could be a command to direct the flow of information, processing the information or telling the computer which data it can operate on. Mice, keyboards, joysticks and trackballs fall under this category.

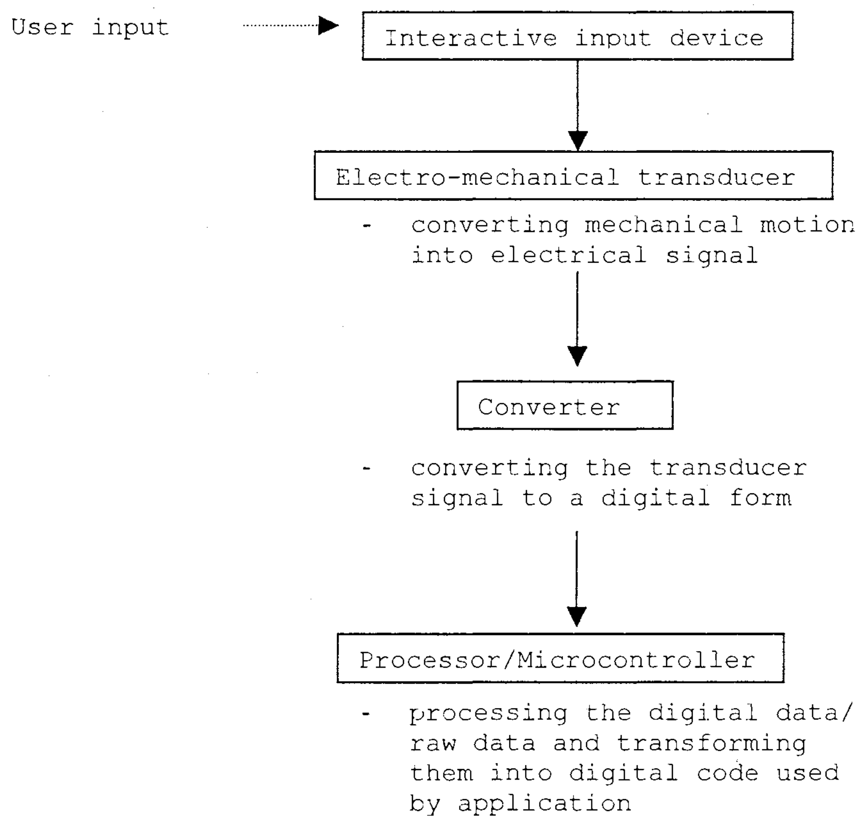


Figure 1-2 Interactive Input Device Flowchart [10]

In the second method, the computer gets input directly from its environment without human intervention [8]. The processor will react to the state of physical world according to commands in its processor or microcontroller. This type of input is the 'passive input' or indirect input. Some building temperature controls, fire-sprinkler detectors and power cutoff system fall under this category.

Computer Mice

'Mice are small, hand-held pointing and selecting devices that are used to control the motion of a cursor on a computer's screen. The motion of the cursor corresponds to the movement of the mouse across a surface. Mice contain a motion-sensing mechanism and one or more switches that can be actuated by an operator's fingers. Switch actuation can cause menus to appear or can select certain commands or options from existing menus [8].

Douglas Engelbert invented the first mouse in 1965 at Stanford Research Institute (US Patent 3,541,541). The first mouse used a pair of wheels to turn potentiometer shafts to encode X and Y positions into analog electrical signals. It was redesigned at the Xerox Palo Alto Research Center where ball bearings were used as wheels and potentiometer shafts were replaced by optical encoders. The optical shaft encoders produce digital quadrature signals. The mouse was redesigned to eliminate wheels but used ball driving mechanical digital shaft encoders (US Patent 3,987,685). The first optical mouse was presented in December 1980 by Steven Kirsch at MIT in Cambridge (US Patent 4,364,035 and 4,390,873) and Richard Lyon at Xerox in Palo Alto (US Patent 4,521,772 and 4,521,773). The optical mouse does not require a working surface but will be able to sense its motion from an arbitrary work surface [6].

How Regular Mice Work

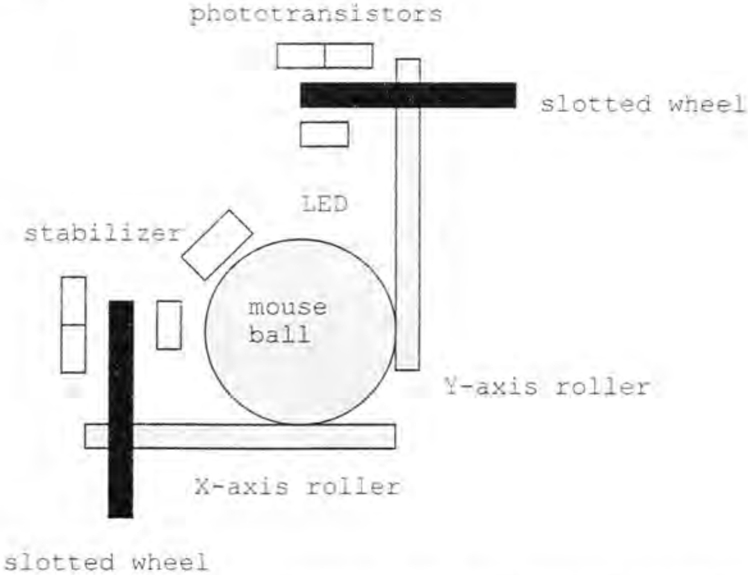


Figure 1-3 Mechanical Hardware of Opto-mechanical Mouse

For a regular mechanical mouse, the X and Y motion of the mouse are measured by counting the pulses generated by the photo couplers. In the case of an opto-mechanical mouse, the rotating wheel blocks the infrared diode, so that the pulses are generated on the phototransistor side. The mouse microcontroller reads the state of those phototransistors and takes into account the current mouse position. If this information changes, the mouse microcontroller will send a packet of data to the computer data interface controller. The mouse driver inside the host computer continuously updates the mouse cursor's position on the screen as the mouse moves, without requiring action from the application program using it. Typically a mouse driver has the information of the current mouse state (position and button states) and sends it to the application or operating system. The mouse driver calls mouse cursor moving routines when the mouse is moved and sends messages to the application when buttons are pressed [8].

The standard PS/2 mouse supports the following input: X (right/left) movement, Y (up/down) movement, left button, middle button, and right button. Most of the USB mice also support the same input. The only difference is in the way they send data from mice microcontroller to the host computer [8].

Another mouse technology uses optical sensors to emit pulses as the mouse moves across a pad with special grid pattern. The infrared light emitted by the infrared diode is reflected off the pad patterned with vertical and horizontal grid lines. It is then received by the phototransistor in the mouse. Then the pulses are processed in the same manner as a mechanical mouse. The optical mice make no noise nor require moving parts [8].

A typical mouse controlling system is shown below.

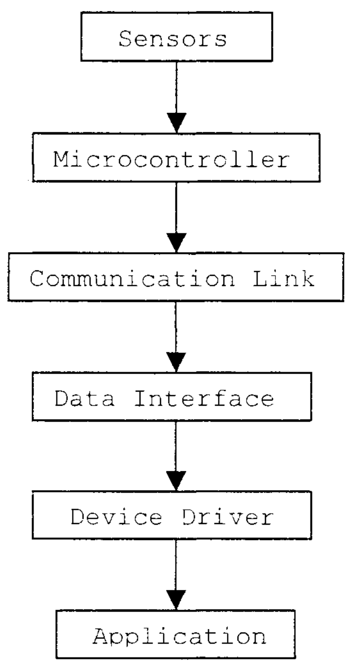


Figure 1-5 Mouse Controlling System

Advantages and Disadvantages of Regular Mice

The most significant advantage of regular mice is that they can be operated in a very small space. This is because the mouse can be lifted and repositioned without causing any signal to split. The mouse is also less expensive than other input devices like graphics tablets.

The mice have disadvantages too. The clearest one is that even though it just needs a small space to operate, this space should be around the keyboard. If the user has a very limited space to work with, this would be a problem. Another problem with mice is that it is unnatural to use mice for drawing. It would be better to use a pen or pencil-like input device for drawing purposes. If we compare mice with digitizers, they have a lower resolution and information transmission rate. So, digitizers are more effective than mice when we talk about resolution and speed.

Summary

This chapter summarized how the project is conducted and also presented how a regular computer mouse operates. Chapter 2 explains the details of a mouse sensor, the elimination of rolling parts of the mice and their replacement with the pressure sensors.

CHAPTER 2:
SIGNAL TRANSDUCER/SENSOR

Motion detection for a mouse consists of four commonly known mechanisms: the mechanical mice, the opto-mechanical mice, the wheel mice, and the optical mice.

The mechanical and opto-mechanical mice use a roller ball. The ball presses against two rollers, which are connected to two disks for the encoding of horizontal and vertical motion. The mechanical mouse has contact points on the disks. As the disks move they touch the contact bars, which in turn generates signals to the microcontroller. The opto-mechanical mouse uses disks that contain evenly spaced slots. Each disk has a pair of Light Emitting Diodes (LED) on one side and a pair of phototransistors on the other side [6].

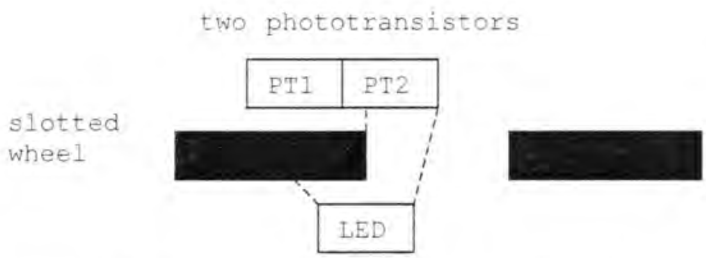


Figure 2-1 Opto-mechanical Details

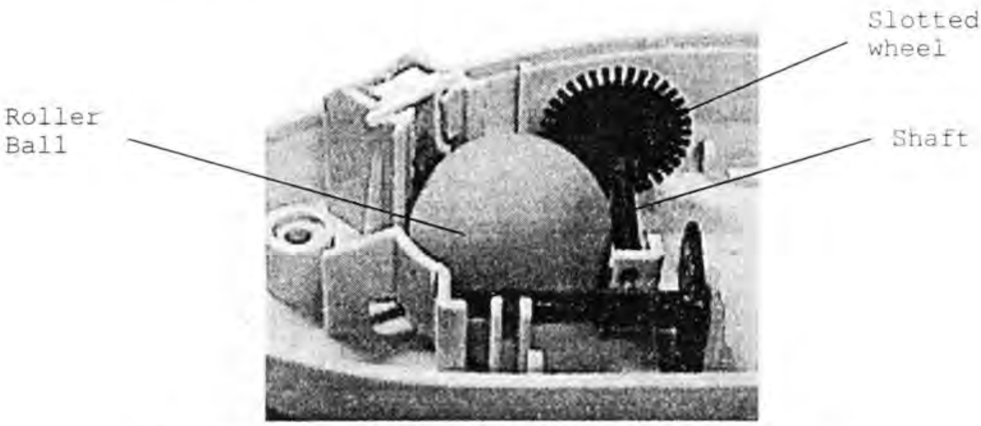


Figure 2-2 Roller Ball, Shaft and Disk

The wheel mouse has the same operation as the mechanical mouse except that the ball is eliminated and the rollers are rotated against the outside surface directly on which the mouse is placed [6].

The optical mouse differs from the rest, as it requires no mechanical parts. It uses a special pad with a reflective surface and grid lines. Light emitted from the Light Emitting Diode (LED) at the bottom of the mouse is reflected by the surface and movement is detected by phototransistor sensors [6].

The most common used structure of mouse movement detectors is the opto-mechanical detector.

Sensor in Opto-mechanical Mice

The mechanical parts of the opto-mechanical mouse consist of one roller ball, two roller cylinders connected to the disks, and the disks with multiple slots (40 slots in most computer mice) on it. These slots are cut so that they are 90 degrees out of phase from each other. The LED and the phototransistors are separated by the disks. As the disks move, light pulses are received by the phototransistor and the frequency of the pulses will depend on the speed of the disks, thus the speed of the mouse. The pulse signals will not be exactly square waves because of unstable hand movement [7].

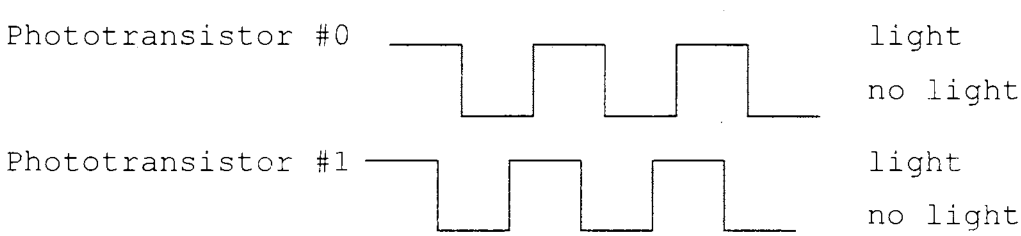


Figure 2-3 Pulses in Phototransistors

So if getting light is called phase '1' and not getting light (when the light is blocked by the wheel) is called phase '0', the signal will look like this:

Signal #0 - 11110000111100001111000011111
Signal #1 - 00111100001111000011110000111

Figure 2-4 Pulse Signal in Phototransistors

With the two tracks being 90 degrees out of phase, there could be a total of four possible track states. It can be observed that the values formed by combining the present and previous states are unique for clockwise and counterclockwise motion. For example, the mice produced by National Semiconductor used binary values of these states to differentiate between clockwise and counterclockwise motion of the wheels. The microcontroller will then encode the states to determine the exact displacement of the mouse [7].

$(Signal0, Signal1)_t$		$(Signal0, Signal1)_{t-1}$		Binary
0	1	0	0	4
1	1	0	1	D
1	0	1	1	B
0	0	1	0	2

Table 2-1 State Table for Counterclockwise Motion of Mouse Wheel [13]

$(Signal0, Signal1)_t$		$(Signal0, Signal1)_{t-1}$		Binary
1	0	0	0	8
0	0	0	1	1
0	1	1	1	7
1	1	1	0	E

Table 2-2 State Table for Clockwise Motion of Mouse Wheel [13]

There are also, at least, two pushbuttons connected to the input port of the microcontroller. When a switch opening or closure is detected, a message is formatted and sent to the host together with the displacement signals.

New Sensor

The new device is designed to be operated by squeezing the mouse. Therefore, all mechanical parts of the mice are eliminated and replaced by pressure sensors. Four pressure sensors are needed for the device.

One rocker switch is needed for right and left click button. One study recommended that left click of the button is for left click and right click for the right button, not the other way around. The rocker switch also should be 13mm minimum in length and 5mm minimum in width. The angle displacement of the switch should not be higher than 30^0 [4].

Figure 2-5 Rocker Switch

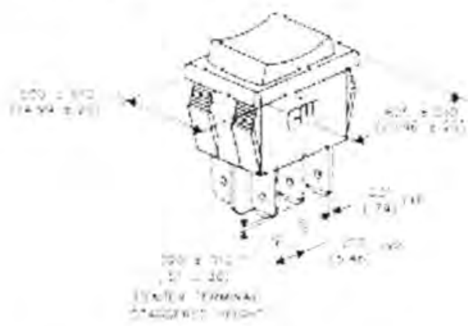


Figure 2-6 Dimension of the Rocker Switch

Force/Pressure Sensor's Specification

The size of the sensor should be small enough since space is the main issue in this new design. Other than the sensor, the microcontroller, connector and other component have to be fit into a shell of the size smaller than human's palm.

Also, since the sensor will be used by a human, its sensitivity must be suitable for the pressure exerted by a human hand.

Piezoresistive/Piezoelectric Sensors

Piezo means pressure while resistive means the opposition to DC current flow. A piezoresistive sensor is a device whose resistance changes as the pressure changes. A piezoelectric material generates an electrical charge when subjected to mechanical strain or, conversely, can change dimensions when subjected to voltage. Most of the piezoresistive sensors are made from silicon [9].

Piezoresistive sensors do not require external power to operate and they have low noise. No external power means fewer components needed and low noise means the signals transduced are more accurate and represent the true value of the input force. There are several options of piezoelectric sensors in the market. FlexiForce™ Sensors from Tekscan Inc. of Boston, Massachusetts are used for this project [9].

FlexiForce™ Sensors

FlexiForce™ Sensors is one example of a piezoresistive sensor. This sensor is thin and flexible enough to measure force between any two surfaces. The sensor is built from two layers of polyester film. Each layer has silver surface (conductive material) followed by pressure-sensitive ink.

Adhesive then is applied to laminate two layers of polyester films together. The sensing area is marked by a silver circle and this silver extends to the connectors at the other end of the sensor. The connector has three pins with the middle pin inactive [12].

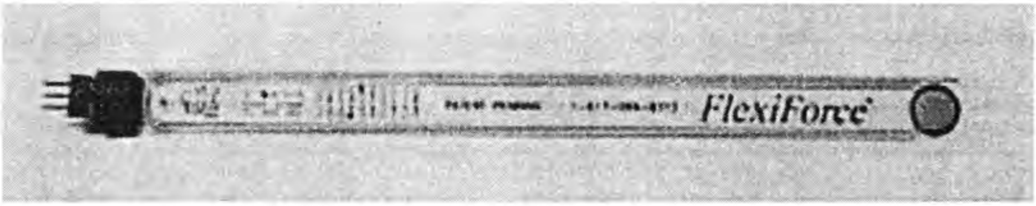


Figure 2-7 FlexiForce™ Sensors

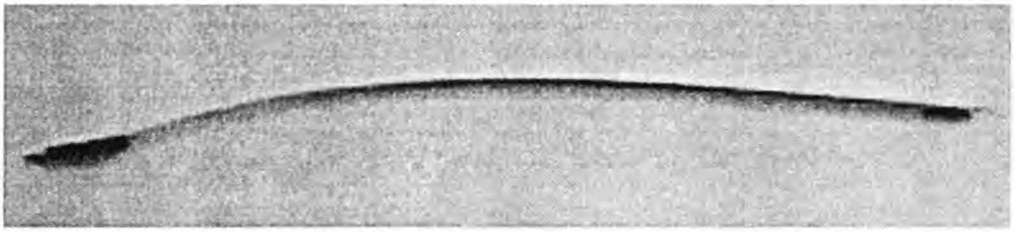


Figure 2-8 Side View of FlexiForce™ Sensors

The sensor yields current as its output, not voltage. Its resistance varies from 20 Mega-Ohm at zero-load to 5 kilo-Ohm at full-load [12]. This sensor comes in force ranges of 1 lb, 25lb, 100 lb, 500 lb and 1000 lb. The 25 lb range is picked for the new mouse. For 25-lb range, the resistance will change approximately 80 kilo-Ohm for every 0.1 lb of force.

Features of a FlexiForce™ sensors

1. **Physical Properties**

   Thickness = 0.005″ (0.127 mm)

   Length = 8.000″ (203 mm) - end of connector to tip of sensor

   Width = 0.55″ (14 mm)

   Active sensing Area = 0.375″ (9.53 mm) diameter

2. Sensitivity

Sensitivity = 80 kΩ/0.1lb or

= 800 kΩ/lb

3. Linearity (Error)

Linearity is defined as how closely the output of a sensor follows a straight line when a linear pressure is applied. This 'linearity error' can be calculated by dividing the maximum deviation of output voltage to input pressure [9].

$$\text{Linearity } \% = [(V_L/V_3)-V_1] \times 100$$

where V_L = maximum deviation or nonlinearity, mV

V_3 = full scale reading, mV

V_1 = no load reading, mV

The Flexiforce™ sensor has less than +/-5% linearity error when the straight line is drawn from 0% to 50% load [12].

4. Repeatability

Repeatability shows how accurate the sensor is in repeating a pressure measurement at any pressure (within the pressure range and temperature range). The sensor is allowed to have full-scale pressure cycles and full-range temperature cycles between the measurements. The repeatability (error) is then the maximum error of consecutive measurements at set reference conditions [9].

Flexiforce™ sensor has repeatability (error) at +/-2.5% of full-scale [12].

5. Drift

Drift is unwanted measuring error, which varies very slowly. In the new design, sensor is used to sense force and force only. The sensor has to be sensitive only to input force and ignore others. It should be insensitive to temperature and others. If the error varies rapidly, it is called noise [9].

Flexiforce™ sensor has 3% drift per each logarithmic time at constant load of 25-lb [12].

6. Rise Time

Rise time is a measurement of time taken to get the output voltage from 10% to 90% of output range. Rise time is mostly used to measure an ability of a system to handle transients. In this case, the equilibrium/final response is the full-scale value of the sensors [9].

Flexiforce™ sensor needs less than 20 microseconds to increase from 10% to 90% of its full-scale voltage [12].

7. Operating Temperature

Flexiforce™ sensor can be operated at temperature between 15^0F to 140^0F (-9^0C to 60^0C) [20]. This device is designed to operate indoors but with this kind of temperature range, the device could be used outdoors [12].

Excitation Circuit

The output of Flexiforce™ sensor is in resistance (Ohm) or current (Ampere). An excitation circuit is needed to convert the resistance/current into voltage. The circuit consists of an op-amp and a resistor. The 5-Volt power from USB line is used to power up the op-amp. The 5-Volt power line could be used as a reference voltage for the op-amp as well, although other values can also be used.

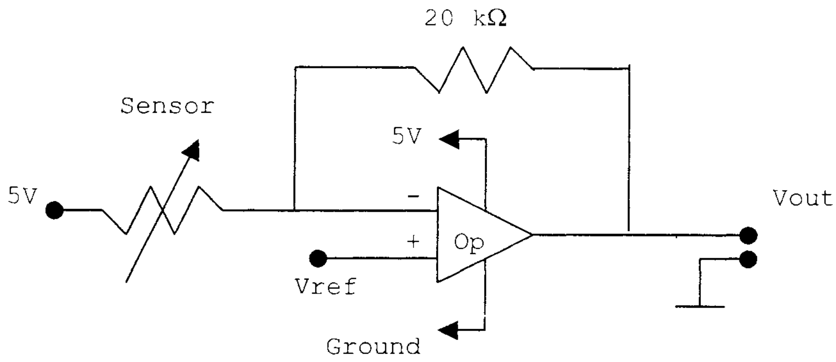


Figure 2-9 Sensor's Excitation Circuit

The Vout of the excitation circuit varies from Vref at zero load and down to 0-volt at full load.

Summary

The op-amp in the excitation circuit does some signal conditioning by converting unstable current input to much stabilized voltage output [11]. The voltage signal is directly proportional to the force applied on the sensor [12].

The microcontroller will process the signal before it can be transferred and accepted by host computer. Chapter 3 explains about the microcontroller.

CHAPTER 3: MICROCONTROLLER

Most microcontroller are grouped into families. The first generation of microcontroller was invented by Intel Computer called 8048 microcontroller. It later be known as 8051 microcontroller. The PIC family was developed by Microchip. The PIC was the first microcontroller to use Reduced Instruction Set Computer (RISC) technology. It has only 35 single instructions compared to 90 for 8051 microcontroller. It was also the first microcontroller to use two different buses for program and data. National Semiconductor, Cypress and several other developers have their own microcontroller family too [14].

All microcontroller need a program or firmware in order for them to run. Most microcontroller do not need any extra programming for interfacing but they still need the firmware to carry out the task of reading inputs, sending out data to the host computer and so forth.

The microcontroller should be capable of handling a low speed USB device transaction. It should be small in size to meet the space requirement. The microcontroller should also have at least 4 analog input pins and 4 analog-to-digital converters to accommodate the 4 outputs from the sensors. It should have 2 other input-output (I/O) inputs for the left click and right click buttons.

Elements of a USB Microcontroller

1. CPU

A controller chip's central processing unit (CPU) controls the chip's actions by executing instructions in the firmware stored in the chip [21].

2. Program Memory

The program memory holds the code that the CPU executes. This memory may be in the CPU chip or a separate chip. It can be ROM (read-only memory), EPROM (erasable programmable ROM), EEPROM (electrically EPROM), RAM (random access memory) or OTP (one-time programmable) memory [21].

3. Data Memory

Data memory provides temporary storage during program execution. Data memory is usually RAM [21].

4. Registers

Registers are another option for temporary storage. Registers are memory locations that are accessed using different instructions than those used for data memory. Most have defined functions, like Analog-to-Digital Result (ADRES), which is used to store the result from Analog-to-Digital Conversion. The register can also be accessed more quickly than other data memory [21].

5. USB Port

A USB microcontroller must of course have a USB port and supporting circuits [21].

6. USB Buffers

A USB microcontroller must have transmit and receive buffers for storing USB data [21].

7. I/O Ports

Other than the USB port, a microcontroller often includes a series of general-purpose input and output (I/O) pins that will connect to other circuits or input sensors [21].

PIC16C745 8-Bit CMOS USB Microcontroller

A Microchip PIC16C745 microcontroller was selected for this project.

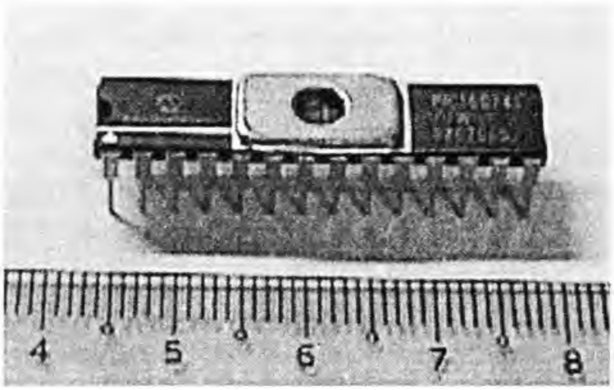


Figure 3-1 PIC16C745

Some of the features of the PIC16C745 include [14]:

- High Performance Reduced-Instruction-Set-Chip/Computer (RISC) CPU
- Only 35 single word instructions
- Interrupt capability (up to 12 internal/external interrupt sources)
- Reset capable
- Programmable code-protection
- Power saving SLEEP mode
- Processor clock of 24 MHz derived from 6 MHz crystal or resonator

- Fully static low-power, high speed Complementary-Metal-Oxide-Semiconductor (CMOS)
- Operating voltage range
  - 4.35 to 5.25 V
- High Sink/Source Current 25/25 mA
- Wide temperature range
  - Industrial (-40^0C - 85^0C)
- Low-power consumption:
  - ~16 mA @ 5V, 24 MHz
  - 100 µA typical standby current
- Universal Serial Bus (USB 1.1)
  - Soft attach/detach
- 22 Input/Output (I/O) pins
  - Individual direction control
  - 1 high voltage open drain (RA4)
  - 8 PORTB pins with:
    - Interrupt-on-change control
    - Weak pull-up control
  - 3 pins dedicated to USB
- 5 8-bit multi-channel Analog-to-Digital converter
- In-Circuit Serial Programming (ICSP)

PIC16C745 Pin Diagram

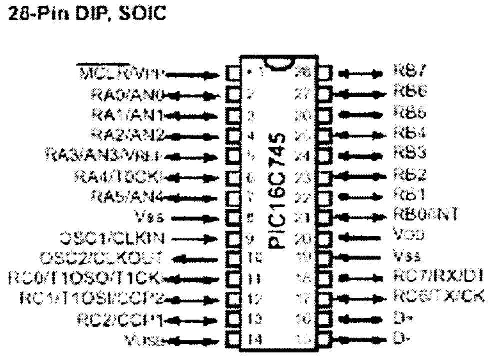


Figure 3-2 PIC16C745 Pin Diagram [Microchip Technology Inc., 2000]

Oscillator Selection

The PIC16C745 has four oscillator alternatives:

1. High Speed (HS) Crystal/Resonator
2. External Clock (EC)
3. High Speed (HS) Crystal/Resonator with internal PLL enabled
4. External Clock (EC) with internal PLL enabled

A high speed crystal is used in this project. In this mode, a 6 MHz crystal is connected between OSC1 and OSC2 pins to provide a 24 MHz processor clock. It is required to use a parallel cut crystal instead a series cut crystal. A series cut crystal may give a frequency out of the crystal manufacturer's specification [14].

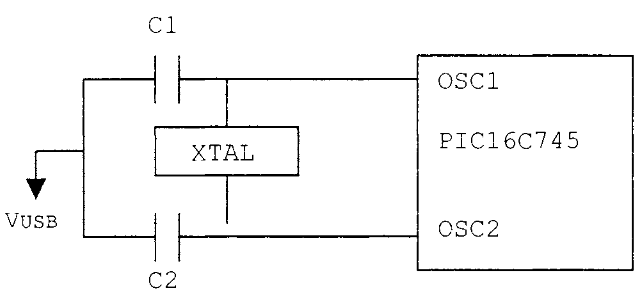


Figure 3-3 Crystal/Resonator Oscillator

Higher capacitance of C1 and C2 increases the stability of the oscillator but also increases the start-up time [14]. If a ceramic resonator is used instead, it could range from 10-68 pF for 6 MHz operation. The capacitor C1 and C2 could be as low as 15 pF to 33 pF for the same 6 MHz operation. In this project, 33pF capacitor is used for both C1 and C2.

Code Protection

The code protection option is used to keep the firmware inside the microcontroller from mishandling. This is to make sure nobody other than the programmer has access to the firmware [14]. But Microchip, the manufacturer of this microcontroller, does not recommend code protecting because devices that are code protected may be erased, but not programmed again. Code protection is set to be off in this project.

In-Circuit Serial Programming (ICSP)

To program PIC16C745, a serial connection is used with two lines for clock and data, power line, ground and the programming voltage. RB6 becomes the programming clock and RB7 becomes the programming data [14].

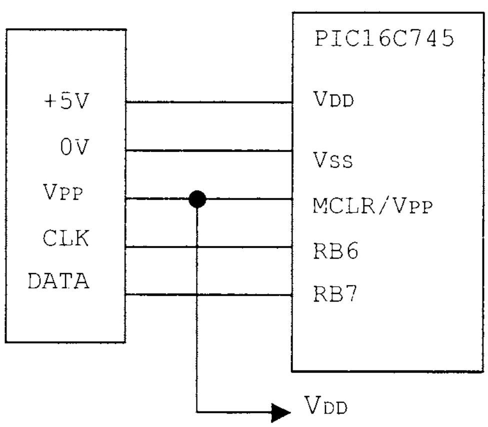


Figure 3-4 In-Circuit Serial Programming (ICSP)

Configuration Word

During the programming, users will have a chance to configure certain features of the microcontroller. These include code protection option, Power-up Timer (PWRT) enable, Watchdog Timer (WDT) enable and oscillator selection [14]. In this project, code protection, PWRT, WDT are all set to be off and H4 is selected for oscillator.

PIC16C745 Instructions (OPCODE)

Instruction (OPCODE)

The PIC16C745 uses 14-bit wide words for programming and it has only 35 words of instructions in total [14].

Field	Description
w	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
d	Destination select; d=0:store result in W, d=1:store result in file register f. Default is d=1
k	Literal field, constant data or label
b	Bit address within an 8-bit file register

Table 3-1 OPCODE Field Descriptions

Operands		Descriptions
Byte-Oriented File Register Operations		
ADDWF	f,d	Add W and f
ANDWF	f,d	AND W with f
CLRF	f	Clear f
CLRW	-	Clear W
COMF	f,d	Complement f
DECF	f,d	Decrement f
DECFSZ	f,d	Decrement f, Skip if 0
INCF	f,d	Increment f
INCFSZ	f,d	Increment f, Skip if 0

IORWF	f,d	Inclusive OR W with f
MOVF	f,d	Move f
MOVWF	f	Move W to f
NOP	-	No Operation
RLF	f,d	Rotate Left f through Carry
RRF	f,d	Rotate Right f through Carry
SUBWF	f,d	Subtract W from f
SWAPF	f,d	Swap nibbles in f
XORWF	f,d	Exclusive OR W with f
Bit-Oriented File Register Operations		
BCF	f,b	Bit Clear f
BSF	f,b	Bit Set f
BTFSC	f,b	Bit Test f, Skip if Clear
BTFSS	f,b	Bit Test f, Skip if Set
Literal and Control Operations		
ADDLW	k	Add literal and W
ANDLW	k	AND literal with W
CALL	k	Call Subroutine
CLRWDT	-	Clear Watchdog Timer
GOTO	k	Go to address
IORLW	k	Inclusive OR literal with W
MOVLW	k	Move literal to W
RETFIE	-	Return from interrupt
RETLW	k	Return with literal in W
RETURN	-	Return from Subroutine
SLEEP	-	Go into standby mode
SUBLW	k	Subtract W from literal
XORLW	k	Exclusive OR literal with W

Table 3-2 PIC16C745 Instruction Set (OPCODE)

PIC16C745 Memory

The memory in PIC16C745 uses Harvard architecture where program and data are accessed from separate memories using separate buses [14]. The regular von Neumann architecture allows program and data fetched from the same memory using the same bus. While allowing program and data to use separate buses in PIC16C745, this lets instructions to be sized differently than the 8-bit wide data word. The instructions (OPCODE) for PIC16C745 are 14-bits wide making it possible to have all single word instructions. This 14-bit OPCODE needs only one single cycle (166.6667 ns @ 24 MHz) to run, except for program branches when they need two full cycles.

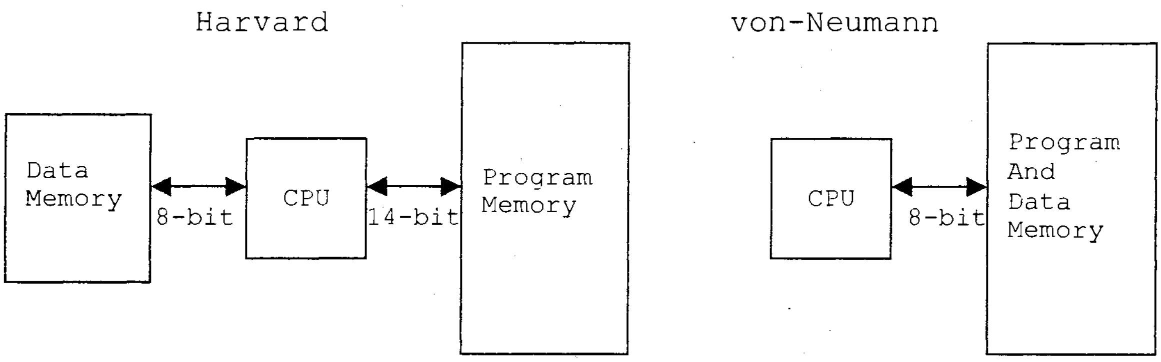


Figure 3-5 Harvard vs. Von-Neumann Architecture

The program memory in PIC16C745 is capable of having up to 8K of 14 bits of program memory [14].

The data memory is divided into four smaller banks and each bank has Special Function Registers (SFR) and General-Purpose Register (GPR) [14]. Register is the same as regular memory except that they have their own name. This would make the process of calling and storing them easier and faster. Each bank has addresses up to 7Fh or 128 bytes. Some highly used SFR, like the STATUS Register is duplicated/mirrored into another bank to reduce access time.

The Arithmetic Logic Unit (ALU) is a general-purpose arithmetic unit that performs arithmetic and Boolean functions between the data in the working register and the one in register file [14]. It is the one who executes addition, subtraction, and etc. The PIC16C745 hold an 8-bit ALU.

PIC16C745 Input-Output (I/O) Ports

The PIC16C745 is equipped with three I/O ports. PORTA has six pins (RA0:RA5) of bi-directional I/O, PORTB has eight (RB0:RB7) of them and PORTC has only five (RC0:RC2, RC6:RC7). Each port has its own data direction bits (TRISA, TRISB, and TRISC) which can be configured to make the pins as output or input [14].

All pins in PORTA have Time-to-Live (TTL) input levels and full CMOS output drivers except RA4, which has an Schmitt Trigger input and an open, drain output. The TRISA register controls the direction of the RA pins, even when they are being used as analog inputs. PORTA is a 6-bit wide port. In this project, Port A is used for the sensors A/D conversion analog input [14].

PORTB is an 8-bit wide bi-directional port and the corresponding data direction register is TRISB. Each of the PORTB pins has a weak internal pull-up. Four of PORTB pins (RB4:RB7) have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur. This interrupt can wake the device from SLEEP. RB0/INT is an external interrupt input pin and is configured using the INTEDG bit in OPTION Register (OPTION_REG). In this project, Port B is used by the left and right button and it utilized the internal pull up feature of the port. By doing this, the pushbutton just needs to be shorted to ground instead of having to connect to an external resistor first [14].

PORTC is a 5-bit bi-directional port with TRISC register. All pins in PORTC have Schmitt Trigger input capability. Port C is left unused in this project [14].

PIC16C745 Analog-to-Digital (A/D) Converter

Some of the advantages of a microcontroller A/D converter include [15]:

- Less cost than fully integrated device
- Minimal necessary hardware or software
- Utilizes available I/O
- Does not require calibration (self-calibrated)

The PIC16C745 is equipped with 5 channels of 8-bit Analog-to-Digital (A/D) converter. This A/D converter converts an analog input signal to a corresponding 8-bit digital value. The module generates the result via a successive approximation method of analog-to-digital conversion. The analog reference voltage is software selectable to either the device's positive supply voltage (V_{DD}) or the voltage level on the V_{REF} pin. In this project, V_{REF} pin is used instead of V_{DD}. The converter can be operated even in SLEEP mode [15].

There are three registers, which are associated with the A/D module. They are A/D Result Register (ADRES), A/D Control Register 0 (ADCON0) and A/D Control Register 1 (ADCON1). The ADRES holds the result from the conversion. The ADCON0 controls the operation of the A/D module, such as A/D conversion clock, A/D conversion status bit and A/D on/off bit. The ADCON1 configures the functions of the port pins. These pins can be configured as analog inputs or as digital input/output [14].

A/D Conversion Steps [18]

1. Configuring the A/D module:
    - Configuring analog pins, voltage reference, digital I/O in ADCON1 register.

- Selecting A/D input channel in ADCON0 register.
- Selecting A/D conversion clock in ADCON0 register.
- Turning on A/D module in ADCON0 register.

2. Configuring A/D interrupt, if desired.
3. Waiting the required acquisition time (T_{AQ}).
4. Starting conversion:
   - Set (1) GO/DONE bit in ADCON0 register
5. Waiting for A/D conversion to complete.
6. Reading conversion results in ADRES register
7. Going to step 1 or 2 for next conversion. A minimum wait of $2T_{AD}$ is required before next acquisition starts. T_{AD} is defined as the A/D conversion time per bit.

Acquisition Time (T_{ACQ})

The acquisition time is important to make sure the A/D converter meets its specified accuracy. After the analog input channel is selected, the acquisition time must follow before the conversion can be started. The charge holding capacitor inside the microcontroller is required to fully charge to the input channel voltage level. The source impedance (Rs) and the internal sampling switch (Rss) impedance directly affect the time required to charge the capacitor [14].

The equation to calculate acquisition time is as below:

T_{ACQ} = Amplifier Settling Time +
Hold Capacitor Charging Time +
Temperature Coefficient

$$T_{ACQ} = T_{AMP} + T_C + T_{COEFF}$$

Where

$$T_{AMP} = 5\mu s$$
$$T_C = -(51.2pF)(1k\Omega + R_{SS} + R_S)\ln(1/511)$$
$$T_{COEFF} = (Temp - 25^0 C)(0.05\mu s/^0 C)$$

Given the maximum source impedance (Rs) at 10kΩ and a worst-case temperature of 100^0C, the acquisition time will not be more than 16 µs [14]. The T_{ACQ} for this device is set at 16 µs.

Conversion Time (T_{AD})

The T_{AD} is defined as the A/D conversion time per bit. The A/D conversion requires 9.5T_{AD} per 8-bit conversion. The minimum T_{AD} time of 1.6µs is required to ensure correct A/D conversion. The source of the A/D conversion clock can be selected during the programming in ADCON0 register [14]. There are four possible options for T_{AD}:

1. 2Tosc
2. 8Tosc
3. 32Tosc
4. Dedicated Internal RC oscillator

A/D Clock Source (TAD)	**Device Frequency**	
	6 MHz	**24 MHz**
2TOSC	333.3 ns	83.3 ns
8TOSC	1 µs	333.3 ns
32TOSC	5 µs	1.333 µs
RC	2-6 µs	2-6 µs

Table 3-3 T_{AD} vs. Device Operating Frequencies

The higher the T_{AD}, the better the result will be [14]. In this project though, the time is important too. So, the A/D converter in this device is set to use 32TOSC conversion clock. With the device operating at 24 MHz, the T_{AD} is 1.333 16 µs for each bit of conversion or 13 µs per conversion.

Sampling/Successive Approximation Method of A/D Conversion

The PIC16C745 uses sampling/successive approximation as its A/D conversion method. The basic principle behind the sampling A/D converter is to use a digital-to-analog converter (DAC) approximation of the input and make a comparison with the input for each bit of resolution. Following the input signal acquisition, the most significant bit (MSB) is tested first. This is achieved by generating ½V_{ref} with the DAC and comparing it to the sampled input signal. The successive approximation register (SAR) drives the DAC to produce estimates of the input signal. The process is started with the MSB and continues to the least significant bit (LSB). For each bit test, the comparator output will determine if the estimate should stay as a 1 or 0 in the result register. If the comparator indicates that the estimated value is under the input level, then the bit stays set. Otherwise, the bit is reset in the result register [16].

PIC16C745 Firmware

Microcontroller will not work if they do not have firmware programmed on them. The firmware is a code where the programmer tells the microcontroller what to do, step by step. The steps include enumeration, input readings, A/D conversion, data transfer and so on. Most of the time, programmers program the firmware in Assembly, or C, and then the development programmer (hardware equipped with development software which usually comes with the microcontroller) of the microcontroller will convert that into hexadecimal language. Most microcontroller, including the PIC16C745, only understands the hexadecimal base language.

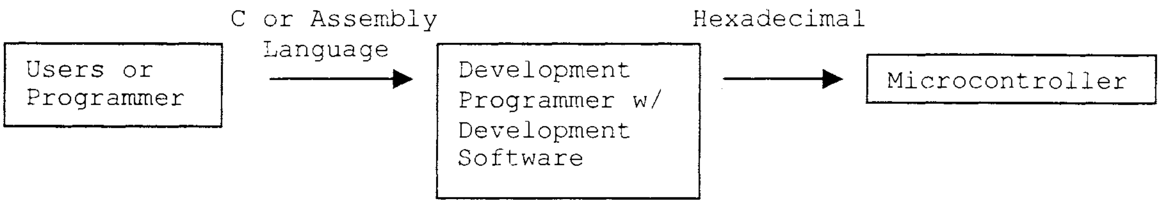


Figure 3-6 Firmware Development

To keep up with a tight dateline, a full time microcontroller expert was hired to write the firmware. Michael J. Cook is a Project Director/Chief Design engineer at ISU Spacecraft Systems & Operations Laboratory (SSOL). His responsibility was to choose a suitable microcontroller for the project and write the firmware for it. Mr. Cook could not finish writing the firmware and this task later was transferred to three undergraduate research assistants. They are Troy Benjegerdes, John Burns and Brenton Rothchild. They were able to program and troubleshoot the firmware.

Summary

In terms of signal flow, the microcontroller accepts any analog input such as resistance, current or voltage. The sensors could be connected directly to the microcontroller without using any excitation circuit. Microchip though recommends the microcontroller input impedance for an analog source to be 10 kΩ, being much lower than the expected ranges of as much as 20 MΩ for the Flexiforce sensors [14]. This recommendation is to provide for suitable acquisition time and accuracy in the A/D converter charge holding capacitor. At larger impedances, it would take much longer to charge the holding capacitor, thus leading to degraded performance. So an op amp is used to convert the sensor's resistance into voltage. This circuit is called the excitation circuit.

The microcontroller does signal conditioning (filtering, amplifying, A/D conversion) according to what the firmware tells it to do. The output of the microcontroller, again, depends on what the firmware tells it to do. In this project the output of the microcontroller is in USB plus and minus differential. These data are ready to be used by the host computer. Chapter 4 explains about the data transferring from microcontroller (or device) into the host computer.

CHAPTER 4:
DATA TRANSFER

Most of the work in data transfer is done automatically by the microcontroller and device driver in the host computer. There is just a little that the programmer/developer has to take care of, like hardware setup for the data transfer. The programmer also has to make sure that the microcontroller follows some requirements, like USB 1.1 Specifications and Low Speed Device requirement.

Unlike an RS232 interface which has 2 separate lines for transmitting and receiving data, USB has 2 lines used in both transmitting and receiving. When transferring data, the USB's two logic states are differential 1 and differential 0. A differential 1 exists at the driver when the D+ output is at least 2.8V and the D- output is no greater than 0.3V. A differential 0 exists at the driver when D- is at least 2.8V and D+ is no greater than 0.3V. At the receiver (microcontroller), a differential 1 exists when D+ is at least 2V and the difference between D+ and D- is greater than 200 mV. A differential 0 exists when D- is at least 2V and the difference between D- and D+ is greater than 200mV [19].

Basic Definitions

It is important to know some of basic definitions related to data transfer.

Enumeration

The enumeration process allows the host to ask the device to introduce itself and negotiates performance parameters, such as power consumption, transfer protocol and polling rate.

The host initiates the process when it detects that a new device has attached itself to the bus [19].

Frames

Data being transferred in the bus is grouped in a format called frame. Each frame is 1 ms in duration and is composed of multiple transfers. Each transfer type can be repeated more than once within a frame [19].

Endpoints

Endpoints can be thought as virtual ports inside the host computer. Endpoints are used to communicate with a device's function. Each endpoint is a source or sink of data. There are a maximum of 6 endpoints for a low speed device. Endpoints have both In and Out associated with them. The In/Out is with respect to the host not the device [21].

Pipes

A pipe is a virtual connection between a software function that exists on the USB host and a given endpoint on a device [19].

Bus

A bus is a means of getting data from one point to another. The bus includes not only the actual capability to transfer data between devices and the host, but also all appropriate signaling information to ensure complete movement of the data from point A to point B. To avoid loss of data, a bus must include a means of controlling the flow of data, in order to ensure that both ends are ready to send and/or receive information. Finally, both ends must understand the speed with which data is to be exchanged [19].

Serial Mode vs. Parallel Mode

In serial mode, the bits of each character are transmitted one at a time, one after another. A single pipe, lead, or channel is used to transmit the data bits serially. Serial transmission is easier to implement than parallel transmission, and allows greater distances between devices. The Universal Serial Bus uses serial transmission [19].

The parallel interface transmits all of a character's bits simultaneously instead of one at a time. Transmission of all the bits at once in parallel requires eight separate data leads. Transmitting all the data bits of a character between devices at the same time allows for a very fast transmission of the data [21].

Most slow speed devices within a computer system like mice and keyboards use serial interface. Most high performance devices that are connected locally within a computer, such as the CPU, RAM and disk drives use a parallel connection [20].

Serial Port vs. Serial Bus

There is a slight difference between serial port and serial bus. A traditional serial port is a point-to-point connection between a computer and a device, whereas on a serial bus many devices can communicate and share the connection to the computer all at the same time. Each device talks to other devices, or the host computer, through well-defined bus protocols. Each device on the USB is individually addressable, and this is all controlled with software [22].

Protocols

A protocol is a set of rules that is instituted between devices to allow for the orderly flow of information. Protocols include rules or capabilities to support aspects such as when to send information, how to send it, how much information can be sent, confirmation that information has been sent, and means of confirming that the correct information has been sent. Protocols include the control mechanisms for two devices to properly communicate [22].

Flow control is an important aspect of a protocol. Flow control is used to regulate the flow of information between the devices. When computers are communicating with other devices, flow control must be used to ensure that data is not lost [22].

Descriptors

The host computer needs a number of descriptors to provide information necessary to identify a device, specify its endpoints, and each endpoint's function. The five general categories of descriptors are Device, Configuration, Interface, Endpoint and String.

Device Descriptors

The device descriptor provides general information such as manufacturer, product number, serial number, USB device class the product falls under, and the number of different configurations supported. There can be only one device descriptor for any given device [19].

Configuration Descriptors

The configuration descriptor provides information on the power requirements of the device and how many different interfaces (USB, RS232 or PS/2) are supported when in this configuration. There may be more than one configuration descriptor for any given device [19].

Interface Descriptors

The interface descriptor provides the number of endpoints used in the interface and the class driver (specific or HID) to use should the device support more than one device class. There can be only one interface descriptor for each configuration [19].

Endpoint Descriptors

The endpoint descriptor provides details about the transfer type supported, direction (in/out), bandwidth requirements and polling interval. There may be more than one endpoint in a device and endpoints may be shared between different interfaces [19].

String Descriptors

The string descriptor is used to provide vendor specific or application specific information. They may be optional depending on vendor and application [19].

Transfer Basics

The USB communications can be divided into two types: transfers used in configuration and the one used in applications [19].

In configuration communications, the host learns about the device and prepares it for exchanging data. Most of these communications take place when the host enumerates the device on power up or attachment.

Application communications occur when applications on the host exchange data with an enumerated device. These are the communications that carry out the device's purpose.

The USB's two signal lines carry data to and from all of the devices on the bus. The wires form a single transmission path that all of the devices must share. Unlike RS232, which has a TX line to carry data in one direction and an RX line for the other direction, USB's pair of wires carries a single differential signal, with the directions taking turn. Because all of the transfers share one data path, each transaction must include the address of the transaction's source or destination. Every device has a unique address assigned by the host. Everything a device sends is in response to receiving a request from the host to send either data or status information in response to received data [19].

Each transfer contains one or more transactions. A transfer with a small amount of data may require just one transaction. If the amount of data is large, a transfer may use multiple transactions, with a portion of the data in each [19].

Each transaction contains a token packet, data packet and handshake packet. In the token phase, the host sends a communications request in a token packet. In the data phase, the host or device may transfer any kind of information in a data packet. In the handshake phase, the host or device sends status, or handshaking, information in a handshake packet [19].

Each packet contains a PID (packet identifier) and may contain additional information and CRC (error-checking) bits.

For example, token packets contain endpoint address in addition to PID, data packets contain data in addition to PID, and handshake packets contain the handshake code in addition to PID [19].

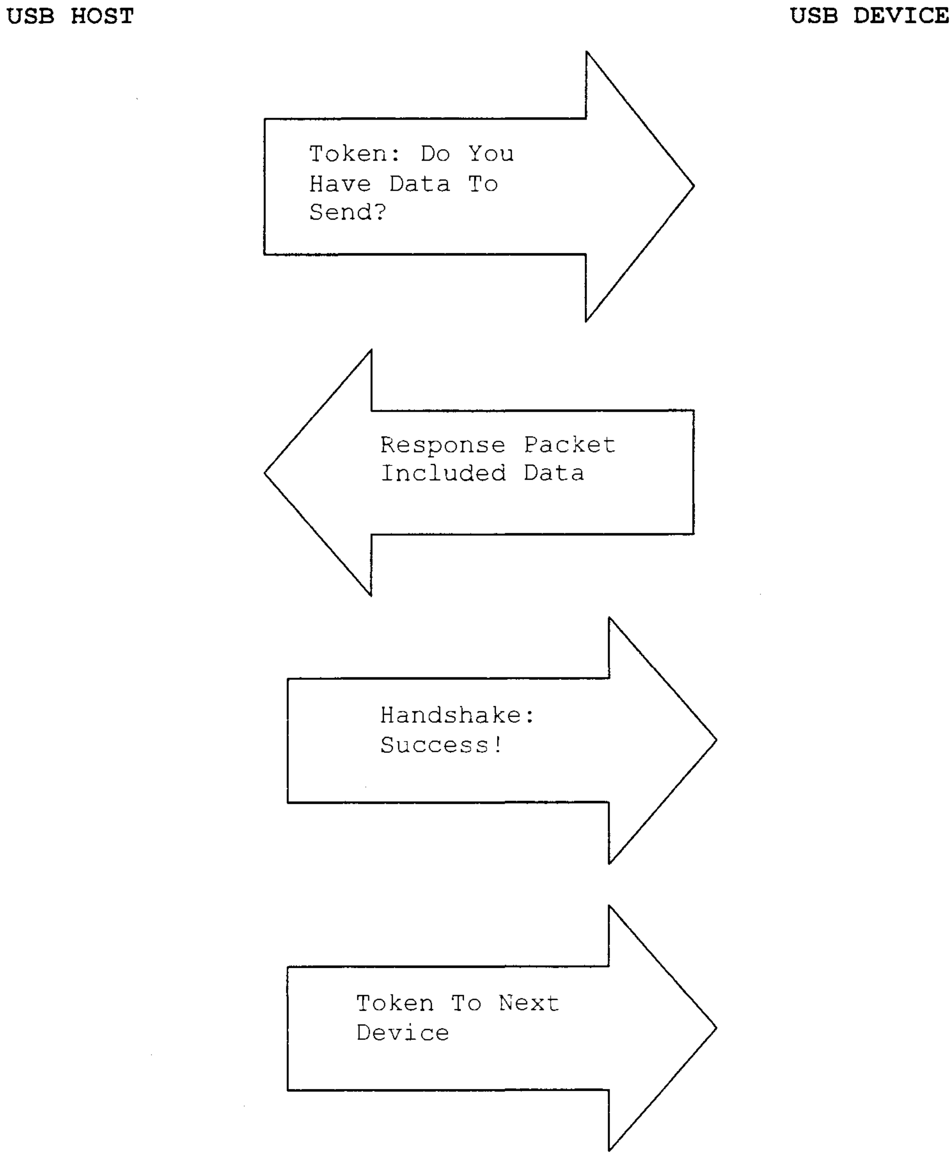


Figure 4-1 Typical USB Bus Transaction

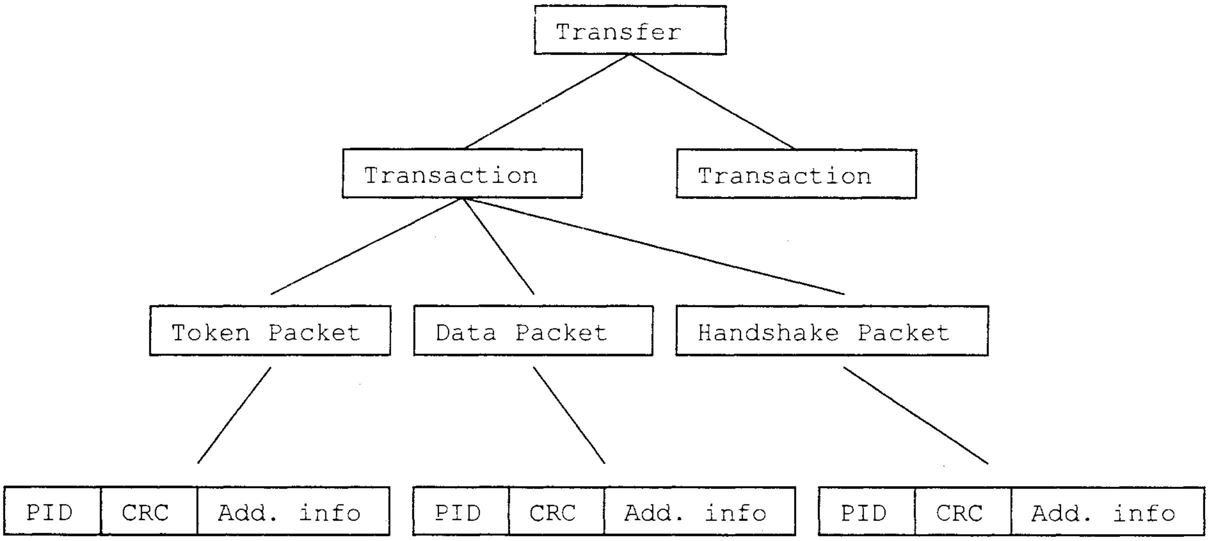


Figure 4-2 Transfer Flowchart

Type of Transfer

Full Speed USB supports all four types of transfer: Isochronous, Bulk, Control and Interrupt. Low Speed USB, like this device, supports only two types of transfer: Control and Interrupt.

Control Transfers

Control transfers send requests and data relating to the device's abilities and configuration. They can also transfer blocks of information for any other purpose. Every device must support control transfers over the default pipe at Endpoint 0 [21].

For the low speed device, the maximum size for the data packet is 8 bytes. The host reads the maximum data size from the device. If a transfer requests more data than will fit in one transaction, the host controller divides the transfer into multiple transactions [21].

The host must make its best effort to ensure that all control transfer get through as quickly as possible.

The host controller reserves 10 percent of the USB bandwidth for control transfers. The specification recommends reserving control transfers for servicing the standard USB requests as much as possible. This helps to ensure that control transfers transmit quickly by keeping the bandwidth reserved for them as open as possible [21].

If a device doesn't return an expected handshake packet during a control transfer, a PC controller will retry twice more. If the host receives no response after a total of three tries, it notifies the software that requested the transfer and stops communicating with the endpoint until the problem is corrected. The two retries include only that sent in response to no handshake at all [21].

Interrupt Transfers

Interrupt transfers are useful when moderate amounts of data have to transfer within a specific amount of time, like in keyboards and mice. Users don't want a noticeable delay between pressing a key or moving a mouse and seeing the result on screen. And a hub needs to report the attachment or removal of devices promptly. Low speed devices, which support only control and interrupt transfers, are likely to use interrupt transfers for generic data. The name interrupt suggests that a device can cause hardware interrupt that results in a fast response from the PC. But the truth is that interrupt transfer, like all other USB transfers, occur only when the host polls a device. The transfers are interrupt-like, however, because they guarantee that the host will request or send data with minimal delay [21].

Low speed devices can use a maximum packet size of 8 bytes. If the amount of data in a transfer won't fit in a single packet, the host controller divides the transfer into multiple transactions [21].

An interrupt transfer guarantees a maximum latency, or time between transaction attempts. In other words, there is no guaranteed transfer rate, just a guaranteed maximum time between transactions. The endpoint descriptor stored in a device specifies latency could be between 10 to 255 ms for low speed devices. The host controller ensures that the transactions have no more than the specified time between them [21].

If a device doesn't return an expected handshake packet, a PC controller will retry twice more. The host will also retry if it receives a negative acknowledge (NAK) from a device [21].

Handshaking

Like other interfaces, the USB has status and control signals that help to manage the flow of data. Most handshaking signals transmit in the handshake, though some use the data packet. The three defined status codes are ACK, NAK, STALL and no response [19].

The ACK (acknowledge) indicates that a host or device has received data without error [19].

The NAK (negative acknowledge) means the device is busy or has no data to return. If the host sends data at a time when the device is too busy to accept it, the device sends a NAK in the 'handshake' packet. If the host requests data from the device when the device has nothing to send, the device sends a NAK in the 'data' packet. In either case, NAK indicates a temporary condition, and the host retries later. Hosts never send NAK [19].

The STALL handshake can mean unsupported control request, control request failed, or endpoint failed. When a device receives a control transfer request that the endpoint doesn't support, the device returns a STALL to the host. The device also sends a STALL if it supports the request but for some reason cannot take the requested action. Another use of STALL is to respond to transfer requests when the endpoint's Halt feature is set, indicating that the endpoint is unable to send or receive data at all. On receiving a functional STALL, the host drops all pending requests to the device and doesn't resume communications until it has sent a successful request to clear the Halt feature on the device. Hosts never send STALL [19].

The No Response status occurs when the host or a device expects to receive a handshake but receives nothing. This usually indicates that the receiver's error-checking calculation detected an error in the data, and informs the sender that it should try again, or take other action if multiple tries have failed [19].

Software Interfacing

For interfacing purpose, PIC16C745 is equipped with a layer of software that handles lowest level interface. It makes the PIC16C745 plug-and-play capable even without firmware written by the user. Most of the processes take place in Interrupt Service Routine (ISR). By having this software, users don't have to do anything while the microcontroller does the enumeration and data communication at the same time. This software gives users simple Put/Get function to interface the microcontroller to the host computer but substantial setup is required to generate appropriate descriptors [14].

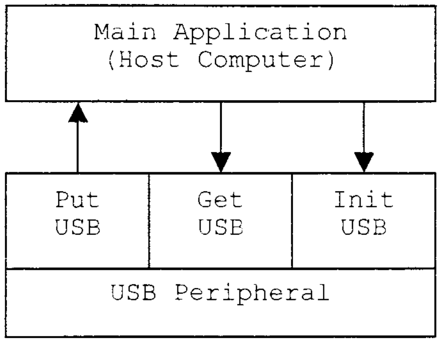


Figure 4-3 PIC16C745 USB Software Interfacing

There are three main functions in this PIC16C745 interfacing software: **InitUSB**, **PutUSB** and **GetUSB**.

The **InitUSB** initializes the USB peripheral, allowing the host to enumerate the device. It enables the USB interrupt so enumeration can begin. The actual enumeration process occurs in the background, driven by the host and the Interrupt Service Routine (ISR). It should be called by the main program immediately upon power-up. It enables the USB peripheral and USB reset interrupt, and transitions the part to the powered state to prepare the device for enumeration. The **PutUSB** sends data to the host computer, and the **GetUSB** receives data from the host [14].

There are other functions related to this PIC16C745 interfacing software: **DelnitUSB**, **ServiceUSBInt**, **StallUSBEP/UnstallUSBEP**, **SoftDetachUSB**, **CheckSleep** and **USBErr**.

The **DelnitUSB** disables the USB peripheral, removing the device from the bus. An application might call this function when it was finished communicating to the host computer. **ServiceUSBInt** handles all interrupts generated by the USB peripheral. The **StallUSBEP/UnstallUSBEP** sets or clears the stall bit in the endpoint control register. The stall bit tells the host computer that user intervention is needed and until such action is made, further attempts to communicate with the endpoint will fail. Once the intervention has been made, UnstallUSBEP clears the bit allowing communication to take place. These calls are useful to signal to the host that user intervention is required, like when a printer is out of paper. The **SoftDetachUSB** electrically disconnects the device from the bus and then reconnects, so that the host could re-enumerate the device. This process is more to check a process to make sure that the host has seen the device disconnect and reattach to the bus. The **CheckSleep** is a test to check if there is no activity on the bus for 3ms. If that is the case, the device can be put to SLEEP to conserve energy, until wakened up by bus activity. This process has to be handled outside the ISR because we need the interrupt to wake us up from SLEEP, and also because the application may not be ready to SLEEP when interrupt occurs. The **USBErr** interrupt notifies the microcontroller that an error has occurred. The device requires no action when an error occurs. Instead, the errors are simply acknowledged and counted. If users wish to pull the device off of the bus when there are so many errors, users have to implement them in the application/firmware. USB or microcontroller does not have a mechanism to do that independently [14].

How Interfacing Software Works/Behind the Scenes [14]

- The InitUSB clears the error counters and enables the 3.3V regulator and the USB Reset interrupt. This will make sure the device responds to commands only after the RESET.

- The computer host sees the device and starts the enumeration process. The RESET will then initializes the Buffer Descriptors Table (BDT), Endpoint Control Registers and enables the remaining USB interrupt sources.
- The interrupt transfers the control to interrupt vector in 04h.
- The host computer sends a setup token requesting device descriptor.
- The host sends an IN transaction to receive the data from the setup transaction.
- This token processing sequence holds true for the entire enumeration sequence.

Demo Program (Courtesy of Microchip)

```
;**************************************************************
; Demo program that initializes the USB peripheral, allows the
; host to Enumerate, then copies buffers from EP1OUT to EP1IN.
;**************************************************************
main
     call       InitUSB   ;set everything so we can enumerate
     ConfiguredUSB        ;wait here until we have enumerated

CheckEP1        ;check Endpoint 1 for an OUT transaction
     bankisel  buffer    ;point to lower banks
     movlw     buffer
     movwf     FSR       ;point FSR to our buffer
     movlw     1         ;check end point 1
     call      GetUSB    ;if data is ready, it will be copied
     btfss     STATUS,C  ;was there any data for us?
     goto      PutBuffer ;nope, check again.
; code host to process out buffer from host

PutBuffer
     bankisel  buffer    ;point to lower banks
                         ;save buffer length
     movlw     buffer
     movwf     FSR       ;point FSR to our buffer
```

```
movlw     0x81      ;put 8 bytes to Endpoint 1
call      PutUSB
btfss     STATUS,C  ;was it successful?
goto      PutBuffer ;No: try again until successful
goto      idleloop  ;Yes: restart loop

end
```

Hardware Interfacing

Conductors

USB cables have four conductors: V_{BUS}, GND, D+ and D-

- V_{BUS} is the +5V supply
- D+ and D- are the differential signal pair
- GND is the ground reference for V_{BUS}, D+ and D-

Low speed cables don't require shielding or twisted pair [20]. This enables low speed cables to be very flexible and without resistance from a stiff cable.

The USB specification requires the following colors and connections for the conductors:

Pin	Conductor	Color
1	VBUS (+5V)	Red
2	D-	White
3	D+	Green
4	GND	Black
Shell	Shield	Drain Wire

Table 4-1 USB Conductor

Connectors

The USB specification describes two connector types

- Type-A plug for the upstream end of the cable
- Type-B plug for the downstream end of the cable

The connectors are keyed so they cannot be plugged in upside-down [20]. The logo is on the topside of the plug.

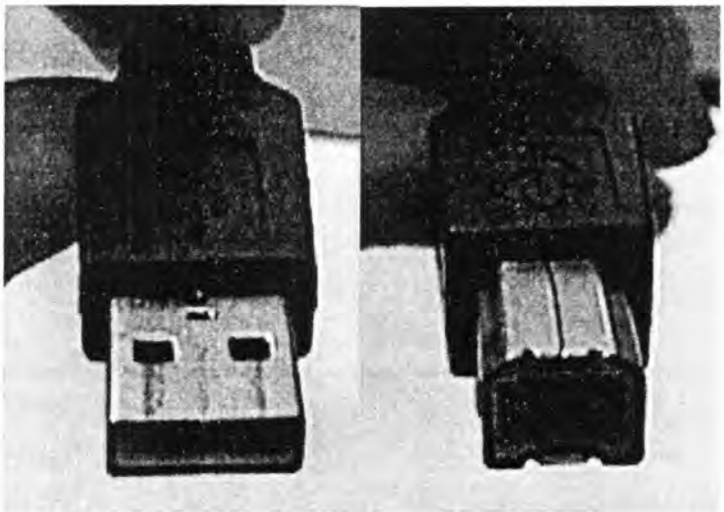


Figure 4-5 USB Connector: Upstream (left) and Downstream (right)

Cable

The USB specification (version 1.0) requires a low speed wire to be less than 3 meters. Version 1.1 of the USB dropped the length specification. The PIC16C745 uses USB 1.1.

The USB specification prohibits extension cables, which would extend the length of a segment by adding a second cable in series [19]. There is one exception where users can use an active extension cable consisting of a hub, a downstream port and a cable. This will work fine because it contains the required hub.

Voltages

The nominal voltage between the VBUS and GND wires in a USB cable is 5V but the actual value can be a little more or quite a bit less [19]. A device that is using bus power must be able to handle the variations and still comply with the specification. If components in the device need a higher voltage, the device can contain a step-up switching regulator. Most USB microcontroller chips require a +5V or +3.3V supply. Components that use 3.3V supplies are handy because the device can use an inexpensive, low dropout linear regulator, or diode, to obtain 3.3V.

Power Needs

The USB specification defines a low power device as one that draws up to 100mA from the bus and a high power device as one that draws up to 500mA from the bus. A self-powered device has its own power supply and can draw as much power as its supply is capable of. On power-up, any device can draw up to 100mA from the bus until the device is configured. A self-powered device may also draw up to 100mA from the bus at any time. This enables the device's USB interface to function even when the device's power supply is off [19].

A peripheral that requires up to 100mA can be bus powered and will work when attached to any host or hub. A peripheral that requires up to 500mA can use power from the bus with one limitation. Not every battery-powered computer and no bus-powered hub support peripherals that draw more than 100mA from the bus [19].

Transceiver Regulator

USB 1.1 Specification requires the microcontroller to have a pull-up resistor (1.5kΩ ±5%) connected between D- line and USB regulator output voltage (V_{USB}) [14]. This drive current is sufficient for a pull-up only. The V_{USB} is pin 14 in PIC16C745 and D- line is pin 15. This requirement is to signal a low speed device to host computer. And for V_{USB} regulator stability, a ±20% 200nF capacitor has to be connected between V_{USB} and ground.

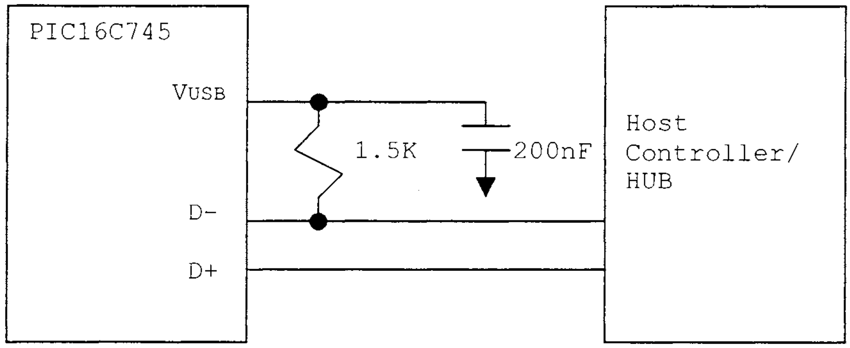


Figure 4-5 Transceiver Regulator

Summary

This chapter summarized some definitions that would help us better understand data transfer. It also presented the way of interfacing the device/microcontroller to the host computer, in software and hardware.

Once the data is transferred to the host computer, it is ready to be used by the computer application, guarded by the device driver. Chapter 5 of this thesis covers the topics of host computer and device driver.

CHAPTER 5:
HOST COMPUTER

The host computer is responsible for enumerating the device every time a device is attached to its port. The device driver located in the host computer is responsible for managing the data between a device and a computer application.

Universal Serial Bus (USB)

The USB system is much more than a serial port. It is a serial bus. This means that a single port on the back of the computer can be the window into a myriad of devices. Devices can be chained together.

USB is implemented as a Tiered Star Topology, with the host at the top, hubs in the middle and spreading out to the individual devices at the end. USB is limited to 127 devices on the bus and the trees cannot be more than 6 levels deep. USB is a host centric architecture. The host is always the master. Devices are not allowed to speak unless spoken to by the host. Transfers take place at one of two speeds. Full Speed is 12 Megabytes/sec and Low Speed is 1.5 Megabytes/sec. Full Speed covers audio/video applications while low speed supports less data intensive applications, like computer mice [22].

Low Speed communication is designed for devices, which in the past used an interrupt to communicate with the host. In the USB scheme, devices do not directly interrupt the processor when they have data. Instead the host periodically polls each device to see if they have any data. This polling rate is negotiated between the host and device giving the system a guaranteed latency [22].

The basic components of a USB are the host computer, the devices and the hubs. The hub is used when there are more devices than available USB ports on the host computer. It doesn't really matter which devices are attached to which ports. There is no performance difference between a device that is 4 hubs away from the computer and one that is attached directly.

History

The Universal Serial Bus (USB) was first invented by a group of computer manufacturers and peripheral vendors named Universal Serial Bus Implementers Forum (USB-IF) in early 1995. The goal of this group was to develop a low to high-speed technology that would provide a shared-access, highly available, robust, self-configuring, extensible, and easy-to-use serial bus to computer owners [21].

In the past, development of a new interface was often the work of a single company. Hewlett Packard developed the HP Interface Bus (HPIB), which came to be known as the GPIB (general-purpose interface bus) and the Centronics Data Computer Corporation popularized a printer interface that is still referred to as the Centronics interface. But an interface controlled by a single company is not ideal. The company may forbid others from using the interface or charge a licensing fee. For these reasons, more recent interfaces are often the product of a collaboration of manufacturers who share a common interest. In some cases, an organization like the IEEE (Institute of Electrical and Electronics Engineers) or TIA (Telecommunications Industry Association) sponsors committees to develop specifications and publishes the results. As a matter of fact, many of the older manufacturer's standards have been taken over by these organizations. The Centronics interface becomes IEEE-1284 standard and GPIB is the basis for IEEE-488. In other cases, the developers of the standard form a new organization to release the standard and handle other development issues. This is the approach used by USB. The copyright on the USB 1.1 specification is assigned by Compaq, Intel, Microsoft, and NEC. All have agreed to make the specification available for use by anyone without charge [21].

The USB 1.0 of the USB specification was released in January 1996 after several years of development and preliminary release. The USB 1.1 was released in September 1998 which fixed a problem identified in release 1.0. The first USB was available on PC with the release of Windows 95. This version is available only for vendors who installed Windows 95 on the PCs they sold. The USB became available to the public in June 1998 with Windows 98. Windows 98 Second Edition (SE) fixed some bugs and further enhanced the USB support. This version of Windows 98 was later called Windows 98 Gold [21].

USB Benefits

One Interface for Many Devices

Instead of having a different connector and protocols for each peripheral, one interface serves many devices [19].

Automatic Configuration

When a user connects a USB peripheral to a powered system, Windows automatically detects the peripheral and loads the appropriate software driver. For a non-HID device, Windows may prompt the user to insert a disk with driver software the first time the device is connected, but other than that, installation is automatic. There is no need to locate and run a setup program or restart the system before using the peripheral [19].

No User Settings

A USB peripheral does not require users to select any settings. It's all done automatically [19].

Easy to Connect

There is no need to open the computer's box to add an expansion card for each peripheral. A typical PC has two USB ports, and if users need more than two ports, a hub can be connected to an existing port. USB can support 127 devices with up to 6 level deeps (Need a hub for every level) [19].

Simple Cables

The USB's cable connectors are keyed so they cannot be plugged in wrong [19].

Hot Pluggable

A peripheral can be connected and disconnected anytime, whether or not the system and peripheral are powered, without damaging the PC or peripheral. The operating system detects when a device is attached and readies it for use [19].

No Power Supply Required

The USB interface includes power-supply and ground lines that provide 5V from the host computer or hub's supply [19].

Host Computer Duty

1. **Detect Devices.** In the enumeration process, the host assigns an address and requests additional information from each device. After power-up, whenever a device is removed or attached, the host learns of the event and enumerates any newly attached device and removes any detached device from the device's available applications [22].
2. **Manage Data Flow.** The host manages the flow of data on the bus. Multiple peripherals may want to transfer data at the same time. The host controller handles this by dividing the data path into 1 ms frames and giving each transmission a portion of each frame [22].
3. **Error Checking.** It adds error-checking bits to the data it sends. When a device receives data, it can perform calculations on the data and compare the results with the received error-checking bits. If the results don't match, the device doesn't acknowledge receiving the data and the host knows that it should retransmit. In a similar way, the host may error-check the data it receives from devices [22].
4. **Provide Power.** The host provides 5V power to its peripherals and works with the devices to conserve power when possible [22].
5. **Exchange Data with Peripherals.** The host's main job is to exchange data. In some cases, a device driver requests the host to poll a peripheral continuously at a requested rate, while in others the host communicates only when an application requests it [22].

Device Duty

1. **Detect Communications.** Each device monitors the device address in each communication on the bus. If the address matches, the device stores the data in it's receive buffer and generates an interrupt to signal that data has arrived. In microcontroller, it's built into the hardware [22].
2. **Respond to Standard Requests.** All USB devices must respond to the eleven standard request codes that query the capabilities and status of the device and select a configuration [22].
3. **Error Check.** Like the host, the device adds error-checking bits to the data it sends. These functions are built into the hardware and don't need to be programmed [22].
4. **Manage Power.** When there is no bus activity, the device must enter its low power Suspend State, while continuing to monitor the bus, exiting the Suspend State when bus activity resumes [22].
5. **Exchange Data with the Host.** The host may poll the device at regular intervals or only when an application requests to communicate with it. The device must respond to each poll by sending an acknowledge code (ACK) that indicates that it received the data, or a negative acknowledge (NAK) to indicate that it's too busy to handle the data. The device's hardware sends the appropriate responses automatically [22].

Enumeration

One of the duties of a hub is to detect the attachment and removal of devices. On system boot-up, the host polls its root hub to learn if any devices are attached. After boot-up, the host continues to poll periodically to learn of any newly attached or removed devices.

On learning of a new device, the host sends a series of requests to the device's hub, causing the hub to establish a communications path between the host and the device. The host then attempts to enumerate the device. Enumeration is the initial exchange of information that enables the host's device driver to communicate with the device. The process consists of assigning an address to the device, reading descriptive data from the device, assigning and loading a device driver, and selecting a configuration from the options presented in the retrieved data. The device is then configured and ready to transfer data using any of the endpoints in its configuration [21].

From the user's perspective, enumeration should be invisible and automatic, except for in some cases a window that announces the detection of a new device and whether or not the attempt to configure it succeeded. For a non-HID device, the user will need to provide a disk containing the INF file and device driver at the first use.

When enumeration is complete, Windows adds the new device to the Device Manager display in the Control Panel. When a user disconnects a peripheral, Windows automatically removes the device from the display.

In device removal, the hub again is the component responsible for telling the host that the device is gone from the bus. The host disables the port that the device was attached to. The host then updates its internal map of the bus to reflect the missing device. At this point, the unique address that the device was using is no longer valid and may be recycled and given to another newly attached device.

Enumeration Steps [20]

1. **The User Plugs a Device into a USB Port.** Or the system powers up with a device already plugged into a port.
2. **The Hub Detects the Device.** The hub has a 15kΩ pull down resistor on each of the port's two signal lines (D+ and D-), while a device has a 1.5 kΩ pull up resistor on either D+ (for a full speed device) or D- (for a low speed device). When a device plugs into a port, the device's pull up brings that line high, enabling the hub to detect that a device is attached.
3. **The Host Learns of the New Device.** Each hub uses its interrupt pipe to report events at the hub. The report indicates only whether the hub or a port has experienced an event.
4. **The Hub Resets the Device.** When a host learns of a new device, the host controller sends the hub a request to reset the port. The hub sends the reset only to the new device. Other hubs and devices on the bus don't see it.
5. **The Hub Establishes a Signal Path Between the Device and the Bus.** At this point, the device can draw no more than 100 mA from the bus.
6. **The Hub Detects the Device's Speed.** The hub detects whether the device is high speed or low speed by determining which line has the higher voltage when idle.
7. **The Host Sends a Get_Descriptor Request to Learn the Maximum Packet Size of the Default Pipe.**

8. **The Host Assigns an Address.** The host controller assigns a unique address to the device. The device reads the request, returns an acknowledge, and stores the new address.
9. **The Host Learns About the Device's Abilities.** The host sends a Get_Descriptor request to the new address to read the device descriptor.
10. **The Host Assigns and Loads a Device Driver.** After the host learns as much as it can about the device from its descriptors, it looks for the best match in a device driver to manage communications with the device.
11. **The Host's Device Driver Selects a Configuration.** After learning about the device from the descriptors, the device driver requests a configuration by sending a Set_Configuration request. Many devices support only one configuration. The device reads the request and sets its configuration to match. The device is now in configured state and the device interface is enabled. The device is now ready for use.

Hubs

Hubs in the USB provide the connection point between devices and the host. All devices plug into hubs, and hub plugs into either the host or other hubs, creating a tiered layer of hubs. It should be noted that the hub is just another USB device but with special responsibilities. Among its responsibilities include device connectivity, power management functions, device attachment/removal detection and bus error detection. The biggest difference between a hub and a regular device is that the hub is controlled by host system software, while a regular device is controlled by client software (microcontroller firmware) [19].

The hub's two main jobs are repeating USB traffic and managing its devices' connections. Managing the connections includes getting newly attached devices up and communicating as well as detecting and blocking communications from misbehaving devices that could interfere with other devices' use of the bus [20].

Each hub has two main components: a hub repeater and a hub controller [20].

The hub repeater is responsible for passing USB traffic between the host's root hub or another upstream hub and whatever downstream devices are attached and enabled. The hub repeater also detects when a device is attached and removed, establishes the connection of a device to the bus, detects bus faults such as over-current conditions, and manages power to the device.

The hub controller manages communications between the host and the hub repeater. As it does for all devices, the host enumerates a newly detected hub to find out its abilities. Hubs are also responsible for disabling any port responsible for loss of bus activity.

States on Ports

A downstream port on a hub is, at any given time, in one of several possible states [20].

- Powered off - no power applied to device
- Disconnected - device is not logically connected to the USB
- Disabled - device may be attached to port but it isn't being recognized

- Enabled - device is attached to the port and can be used
- Suspended - device is attached but is in SLEEP mode

Powered Off State

A port is set to the powered off state at the request of the host. This setting is normally used when the host goes into a power saving mode, such as a laptop going into a suspended state. All of the electrical signals from a port that is in the powered off state are ignored by the hub. The port is treated as dead in this state and all upstream activity from that port to the host is ignored.

Disconnected State

A hub port is in the disconnected state when the port has power but has no device attached. A port transitions from the powered off state to the disconnected state when the host tells the hub to apply power to the port. When a port is in a disconnected state, it doesn't communicate in either the upstream or downstream directions. The port can, however, detect a connect event. A connect event is triggered when a device is plugged into a port on the hub.

Disabled State

A port is put into the disabled state when the hub detects a device being attached. This assumes that the port is currently in the disconnected state, which means that it must be powered on. A device plugged into a port that is in the disabled state cannot talk to the host, but the host can talk to it through the use of a reset signal.

Enabled State

A port transitions to the enabled state when the host tells the hub to put the port into the enabled state. This is done as part of the enumeration process, where the USB device is actually recognized by the host computer. Since the port can transition to the enabled state only from the disabled state, the port can never become enabled if there is no device attached to it.

Suspended State

A device can be temporarily put into a suspended state to keep power from being applied to it. This is different from the powered off state. The suspended state is used when no device is attached to the port. A hub port can be put into the suspended state by either the host requesting that it happen or the device deciding it wants to be suspended.

Device Driver

A device driver is a software component that enables applications to access a hardware device. In the most general sense, a device driver is any code that handles communication details for a hardware device that interfaces to a CPU.

When Windows detects a new USB peripheral, one of the things it has to do is to figure out which device driver applications should use to communicate with the device, and then load the selected driver. This is the job of Windows' Device Manager.

The Device Manager is a Control Panel menu that is responsible for installing, configuring and removing devices. The Device Manager also adds information about each device to the system registry, which is the database that Windows maintains for storing critical information about the hardware and software installed on a system.

The INF file is a text file containing information that helps Windows identify a device. The file tells Windows what driver or drivers to use and what information to store in the registry [20].

When Windows enumerates a new USB device, the Device Manager compares the data in all of the INF files with the information in the descriptors retrieved from the device on enumerating. To prevent having to read through the files themselves each time a new device is detected, Windows maintains a driver information database with information called from the INF files.

Human Interface Devices (HID) Driver

On PCs running Windows 98 or later, applications can communicate with peripheral devices using the drivers built into the operating system called HID Drivers [20]. The HID driver has defined report formats for mice, keyboards, and joysticks. The only requirement of HID is that the device must conform to the requirements of HID class descriptors, and the device must send and receive data using interrupt or control transfers as defined in the HID specification. The device in this project uses a HID driver.

For the host's drivers to communicate with a HID, the device's firmware must meet certain requirements. The device's descriptors must identify the device as having a HID interface, and the firmware must support an interrupt IN endpoint in addition to the default control pipe. The firmware must also contain a report descriptor that defines the format for transmitted and received device data [20].

All device data transferred by a HID must use a defined report format that describes the size and contents of the data in the report [20]. Devices may support one or more reports. A report descriptor in the device's firmware describes the reports, and may include information about how the receiver of the data should use it. Feature reports always use control transfers.

HID Specifications [20]

1. The data exchanged resides in structures called reports. The device's firmware must support the HID report format. The host sends and receives data by sending and requesting reports in control or interrupt transfers.
2. Each transaction can contain a small to moderate amount of data. For a low speed device, the maximum is 8 bytes per transaction.
3. A device may send information to the computer at unpredictable times. The host's driver polls the device periodically to obtain new data.
4. The maximum speed of transfers is limited. A low speed device can have no more than 1 transaction per 10ms, or 800 bytes per second.
5. There is no guaranteed rate of transfer. If the device is configured for 10ms intervals, the time between transactions may be any period equal to or less than this.

Device Manager

The Device Manager is responsible for adding attached devices to the Control Panel display. The Device Manager display shows only the USB devices that are currently detected. Users can unplug a device while viewing the display and watch the device's listing disappear. Plug the device back in, and its listing pops back.

If a newly attached device uses the standard HID drivers, it doesn't need its own INF file to identify it [20]. On the first attachment, the Device Manager will determine that the device is HID class, and when it can't find a Vendor and Product ID match, will decide that the generic HID drivers are the best fit available. The Device Manager will run Add New Hardware Wizard as usual to give users a chance to select a better driver. When users accept default selections, Windows looks for a driver in the INF directory, selects the INF file for the HID class and loads the HID drivers. The Device Manager lists the device as a Standard HID Device with no indication of its specific function or manufacturer.

Summary

This chapter provided some background on USB. The chapter also presented the way the data is handled. It also mentioned that the device in this project uses Human Interface Device (HID) driver to communicate with applications in the host computer. All aspects of the design have been covered. Chapter 6 explains the final assembly of the device.

CHAPTER 6: FINAL ASSEMBLY

Sensor

Four FlexiForce™ sensors are used to transduce input force into resistance/current. These sensors do not need power to operate. The sensors have three pins with the middle pin inactive. Sensor A and sensor B are for horizontal motion while sensor C and sensor D are for vertical motion. These four sensors are placed as below:

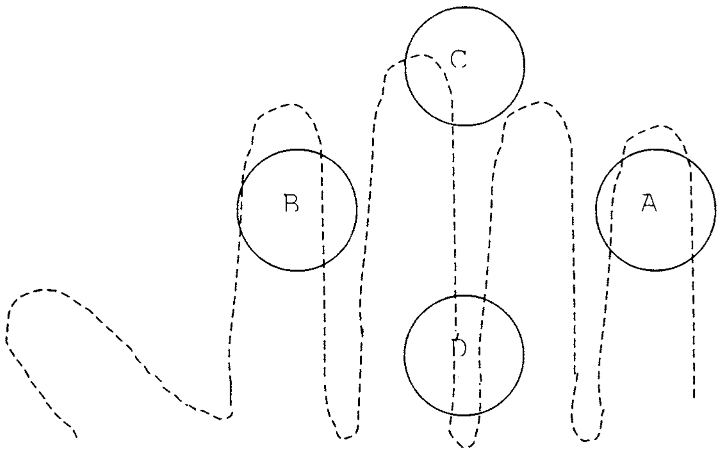


Figure 6-1 Placement of the Sensors

One rocker switch is also used for a left click and right click button. This button has to be shorted to ground.

From the Weimer study [4], the typical time of neural transmission to brain was 2-100 msec, while the neural transmission to muscle was 10-20 msec. So the time taken by sensor to sense the input, to transfer and process the data should be less than the time of neural transmission from and to the brain.

Hand-Eyes Coordination

The objective population of this device was to the users of the regular mice. The users of the regular mice have mastered their hand-eyes coordination so that the motion of the cursor may be performed without any appreciable conscious effort [23]. They learned from the very beginning that if the mice were moved the right, the cursor will move to the right on the screen, and etc. This process of visual motor skill started from the very early age of the person.

The users for this device will go through the same process of learning. The position of the sensors were arranged the way they were to help the users learn the process faster. Although the 'modus operandi' of this device was by squeezing, the arrangement of the sensors might somewhat help. The users used to move the mice to right to move the cursor to the right on the screen. In this device, the users are supposed to press the right sensor to move the cursor to the right.

One study says that humans learn a hand-eyes process faster if the motion of the hand is at the same direction of the movement of the eyes [23]. In this case the motion of the eyes is always at the same direction as the direction of the cursor on the screen. Although the device does not use the same 'modus operandi' as hand motion, the position of the sensor might somewhat help the users to learn the process faster.

The human memory has two parts: short term and long term [24]. The short-term memory is like a RAM in computer. The short-term memory loses its content unless it is refreshed every 200 ms. The long term memory is the main file store of the human system. The cognitive files are one example of the files stored in the long-term memory, which is used to instruct the muscles for movement. The hand-eyes coordination is one of the cognitive files.

Excitation Circuit

The excitation circuit converts input current from the sensors into a voltage. The four outputs (V_{ffa}, V_{ffb}, V_{ffc} and V_{ffd}) from this circuit then are sent into the microcontroller.

Instead of using four single output op-amps (LM 124), one quad op-amp (LM 324) is used in this circuit. The op-amp is powered by 5-volt USB power line. In addition to a 100 kΩ resistor, a 0.1 μF capacitor is added in this circuit to reduce noise from surroundings. The noise could come from power lines and the electromagnetic noise of the circuit.

A 2.5-volt reference voltage circuit is also added here. The circuit consists of an LM336-2.5 diode and a 2.2 kΩ resistor. The 2.5-volt reference is connected to the inverting (positive) pin of the quad op-amp. This means that the output of the excitation circuit (V_{ff}) will be 2.5-volt at no load and down to 0-volt at full load.

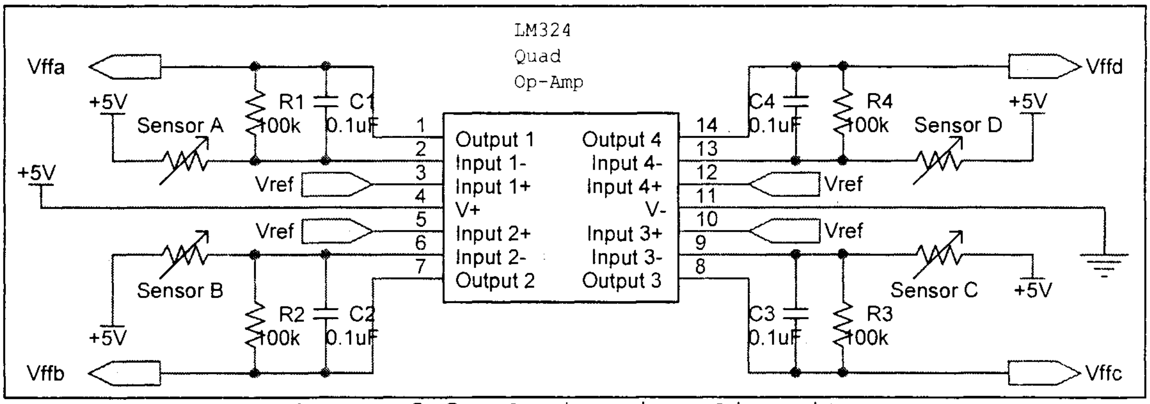


Figure 6-2 Excitation Circuit

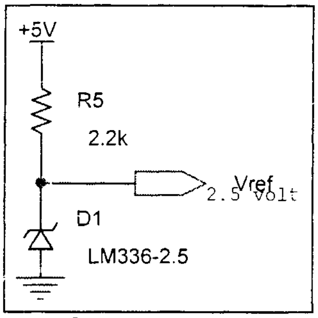


Figure 6-3 Reference Voltage Circuit

Microcontroller circuit

The four inputs (V_{ffa}, V_{ffb}, V_{ffc} and V_{ffd}) are connected to pins AN0, AN1, AN2 and AN4 of the PIC16C745 microcontroller respectively. The other two inputs (left and right button) are connected to RB0 and RB1 of the PIC16C745 microcontroller respectively. This button has to be shorted to ground.

A 6 MHz resonant quartz crystal for the oscillator is connected between pin OSC1 and OSC2 of the microcontroller. Two other 33 pF capacitors are also added into this circuit.

A transceiver regulator circuit has been added into the circuit per USB 1.1 Specification. The circuit consists of one 1.5 kΩ resistor and one 200 nF capacitor. The regulator circuit is used to signal to the host computer that the device is a low speed device. The circuit is also used for V_{USB} stability.

The 2.5-volt reference voltage is also connected between V_{REF} pin and V_{SS} pin of the microcontroller. This voltage will be used as a reference voltage in analog-to-digital conversion. A capacitor is added for stability and to reduce noise.

A 5-volt power is connected between V_{DD} pin and V_{SS} to power up the microcontroller. Per Microchip specification, two other capacitors (0.1 μF and 10 μF) are added to stabilize the incoming power and also to reduce noise.

A 5-volt power is connected to the MCLR pin for the reset function.

The D+ pin and D- pin are directly connected to the back of the host computer, so are the 5-volt power line and the Ground (GND).

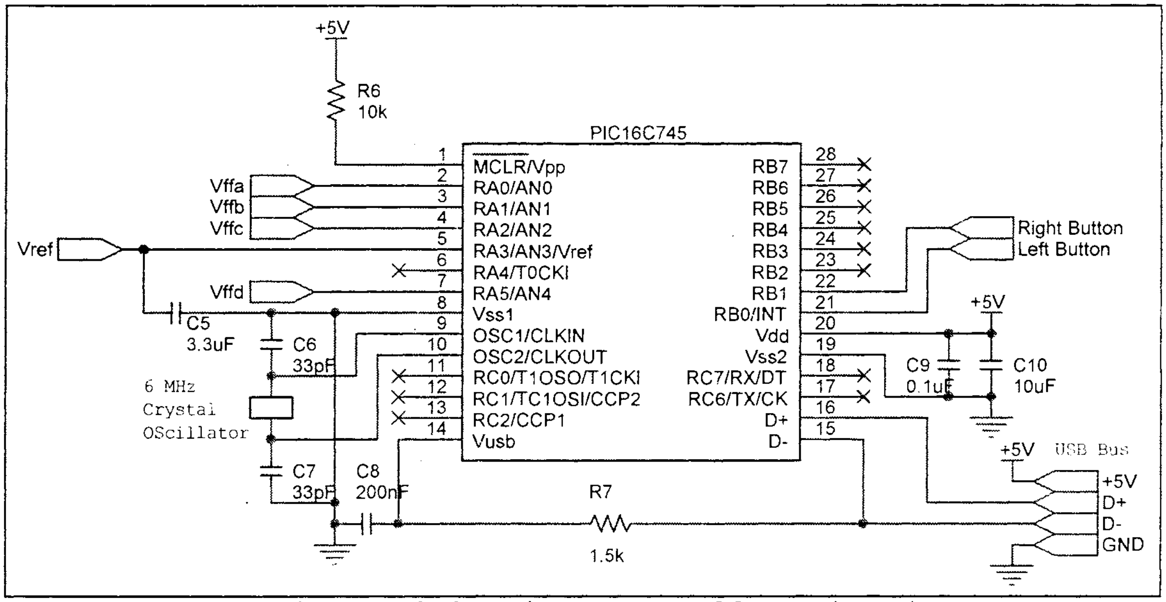


Figure 6-4 Microcontroller Circuit

Internal Box

A 40mm x 40mm x 38mm aluminum box is used to hold the sensors, reference voltage circuit and excitation circuit inside the main shell. The box includes a platform for the rocker switch and also for the actuator to work on.

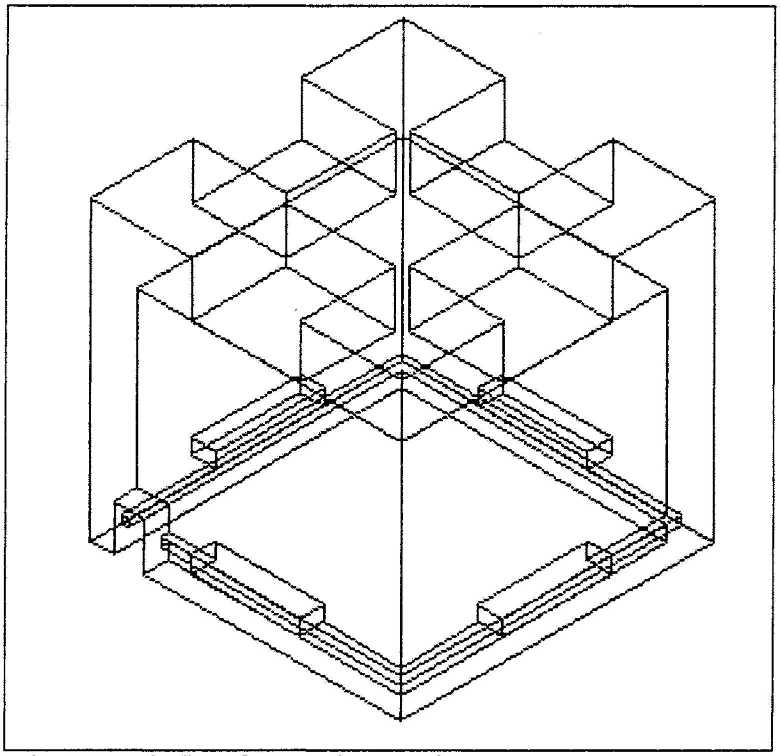


Figure 6-5 3-D View of the Internal Box

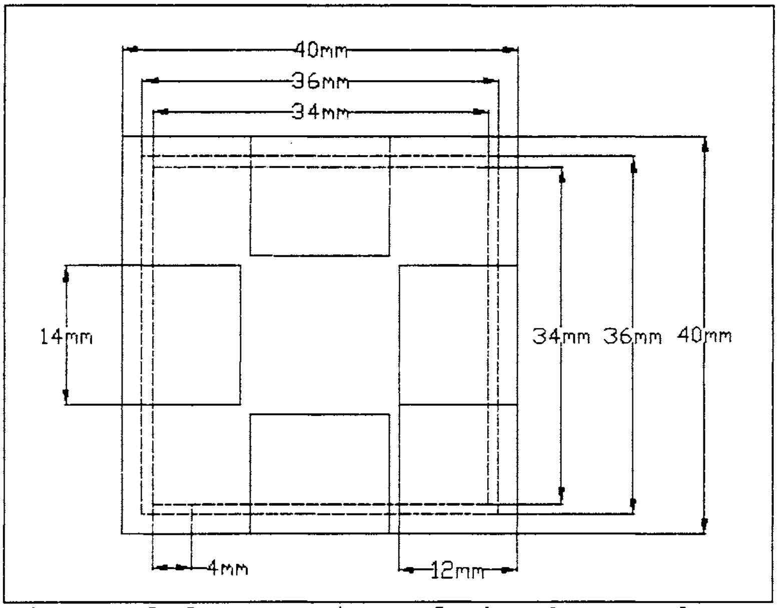


Figure 6-6 Top View of the Internal Box

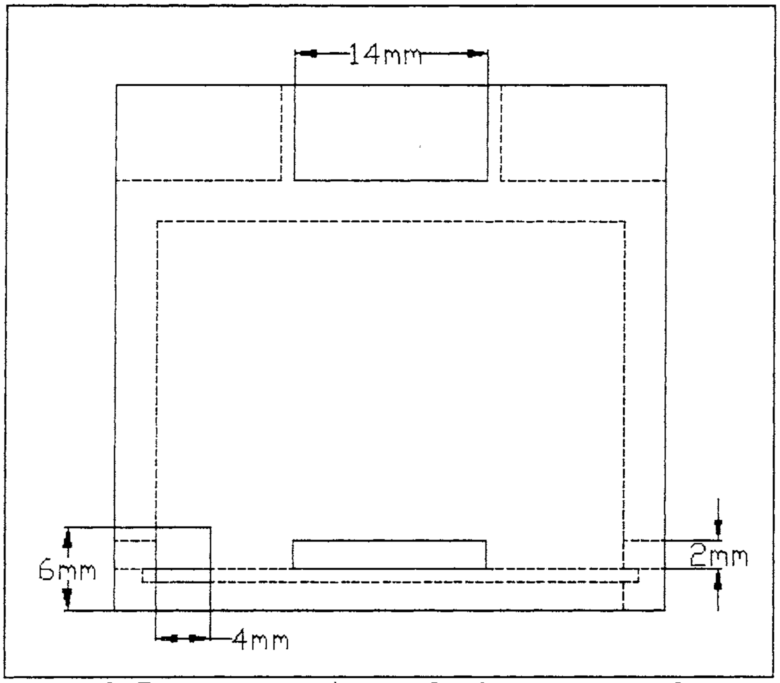


Figure 6-7 Front View of the Internal Box

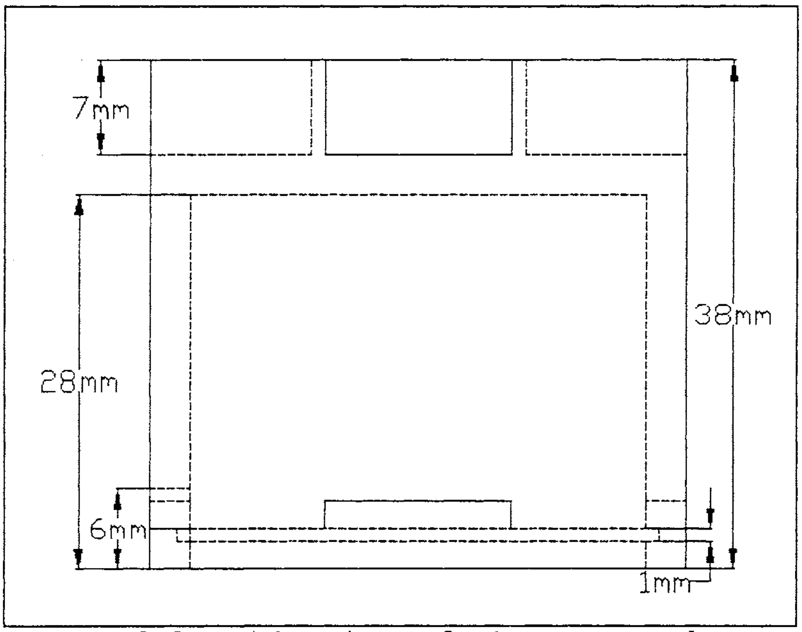


Figure 6-8 Side View of the Internal Box

Cover Plate

The cover plate was used to close the internal box on the bottom.

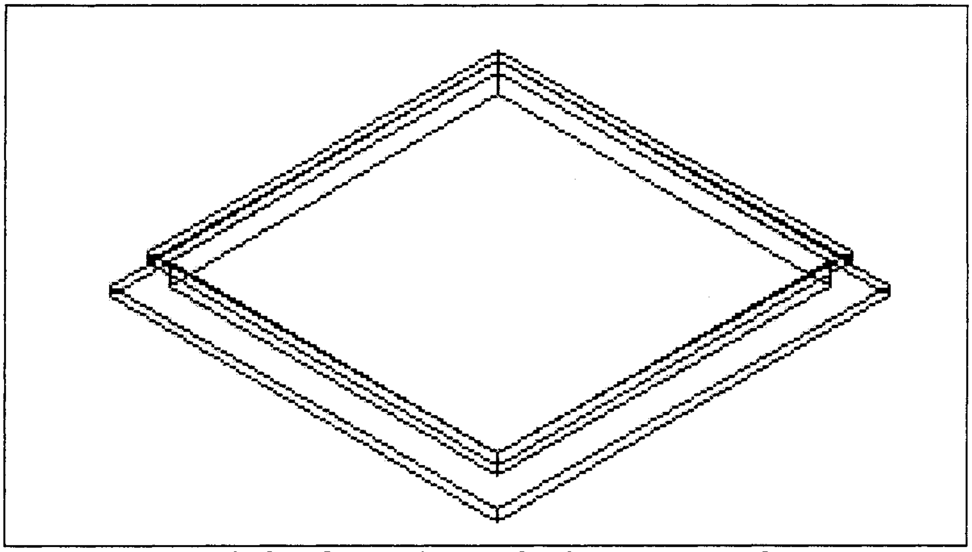


Figure 6-9 3-D View of the Cover Plate

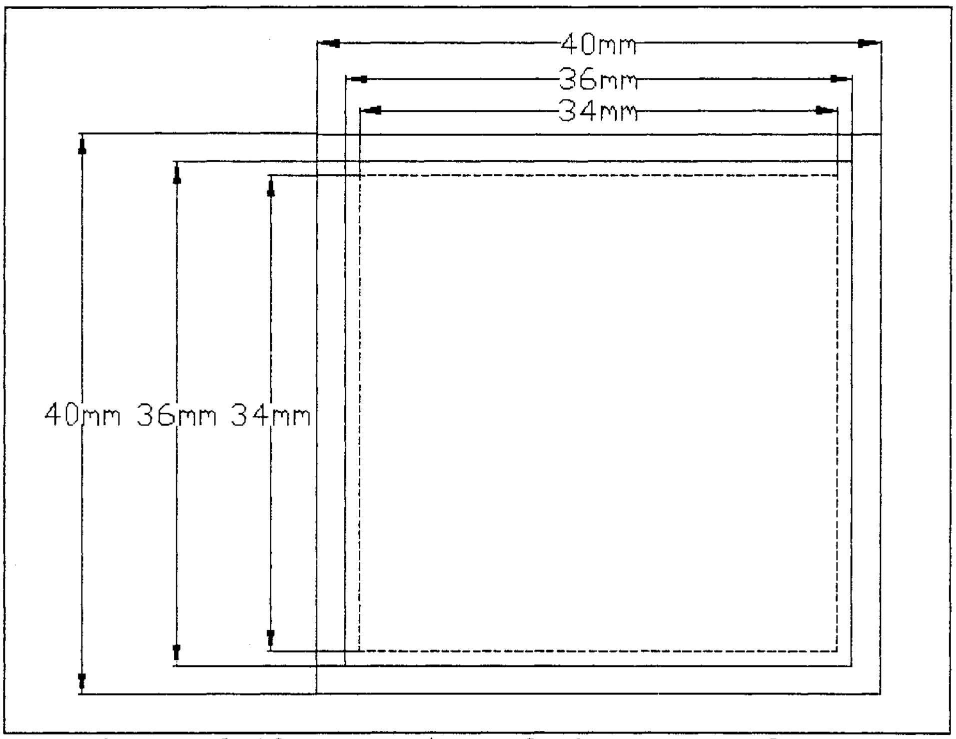


Figure 6-10 Top View of the Cover Plate

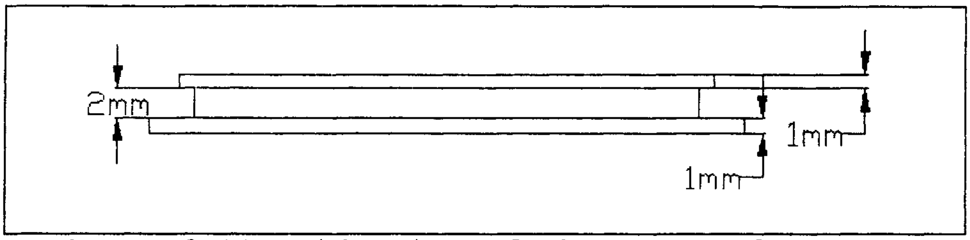


Figure 6-11 Side View of the Cover Plate

Switch Box

Switch box was glued on the side of the internal box to provide a space for the right-left rocker switch.

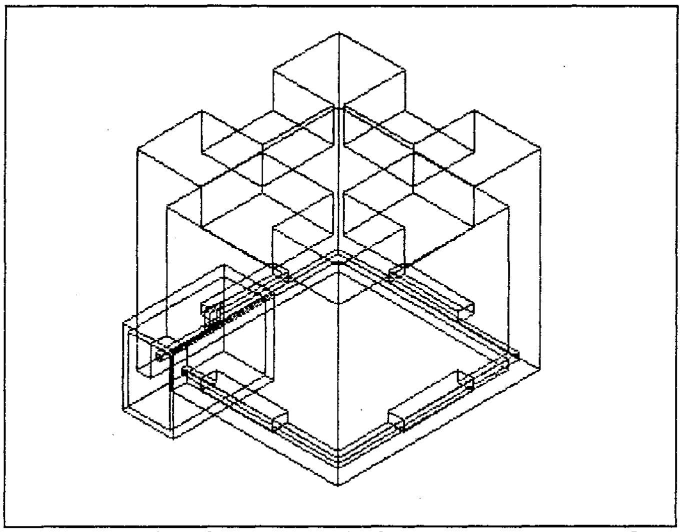


Figure 6-12 Switch Box with the Internal Box

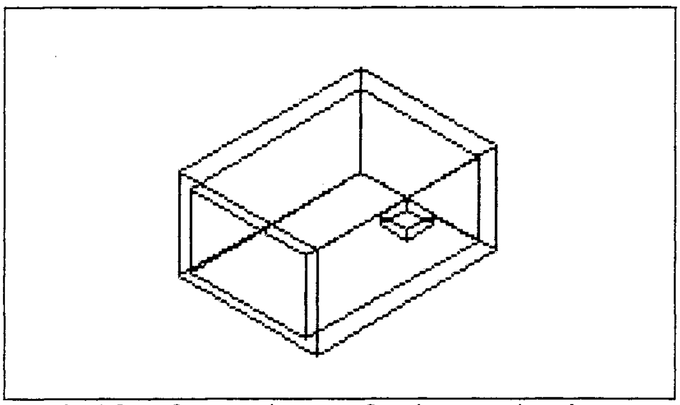


Figure 6-13 3-D View of the Switch Box

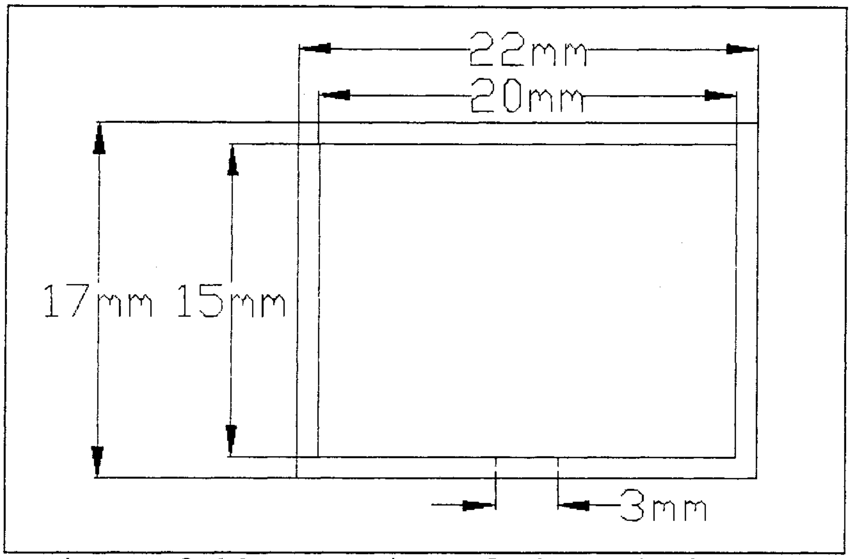


Figure 6-14 Top View of the Switch Box

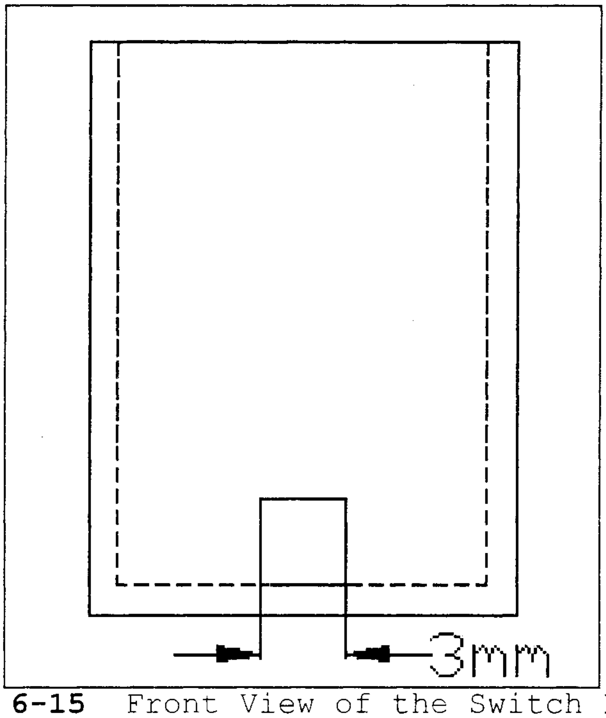


Figure 6-15 Front View of the Switch Box

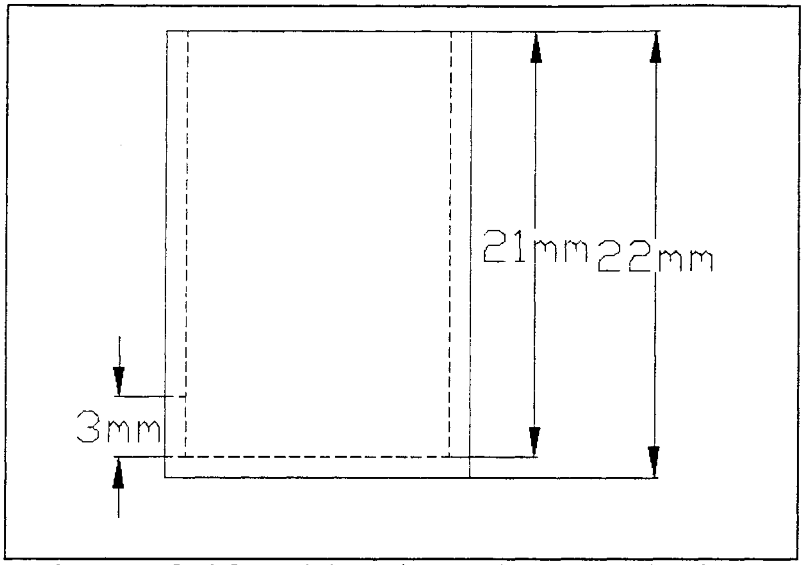


Figure 6-16 Side View of the Switch Box

Actuator

The actuator was placed on the top of the internal box. The actuator was used to transmit the force exerted by the users to the force sensors. This will make the force to be fully distributed among the sensors rather than concentrated at just one sensor. The actuator was also used to make it easier for the users to control the direction of the cursor. Instead of having to press the sensor exactly on top of it, which is so difficult, the users just need to press the actuator at the direction they want the cursor to move, without having to worry about the location of the press.

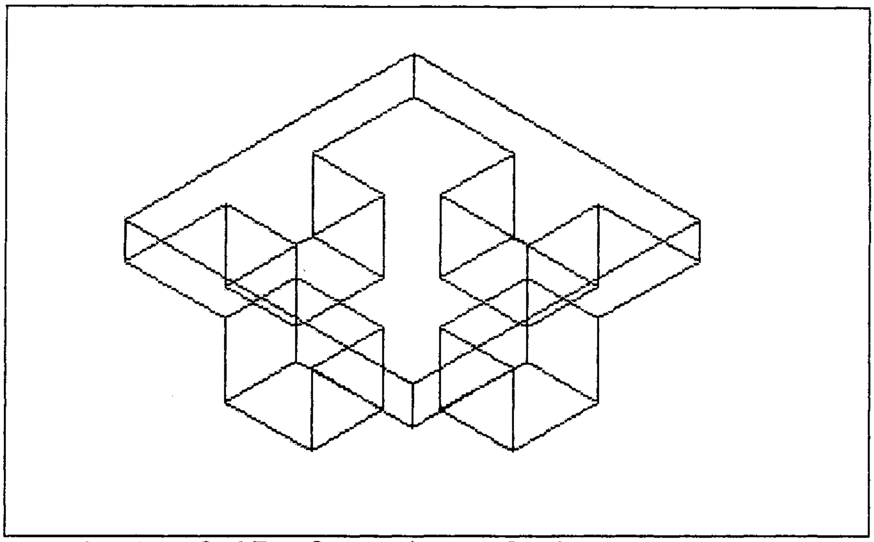


Figure 6-17 3-D View of the Actuator

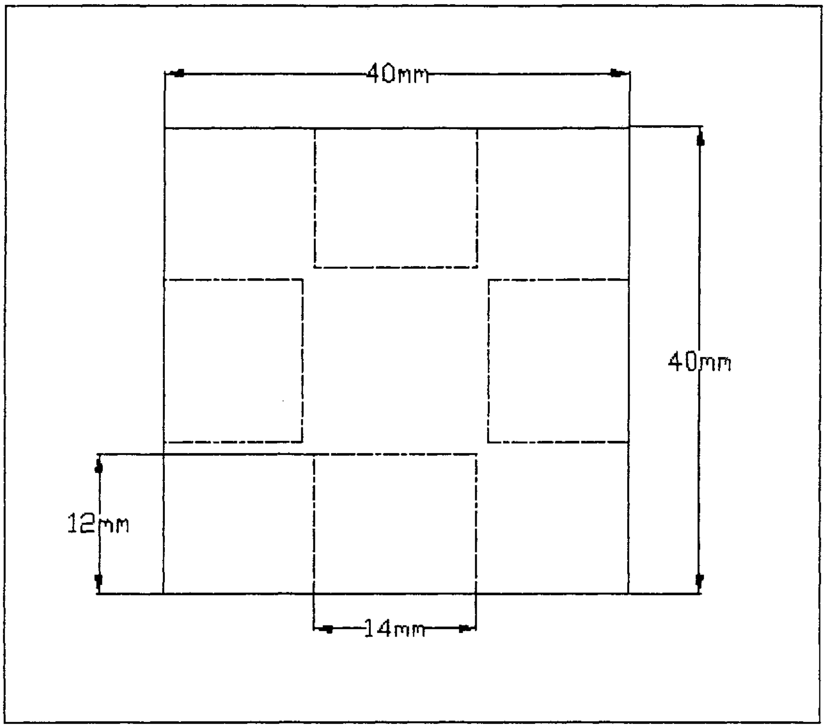


Figure 6-18 Top View of the Actuator

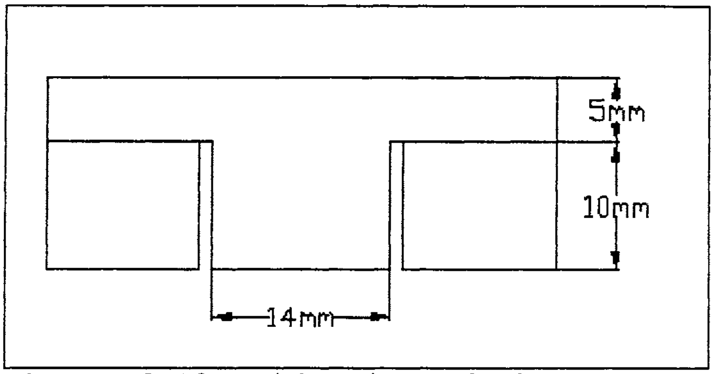


Figure 6-19 Side View of the Actuator

External Box

A 60mm x 50mmm x 30mm plastic external box is used to hold the microcontroller circuit.

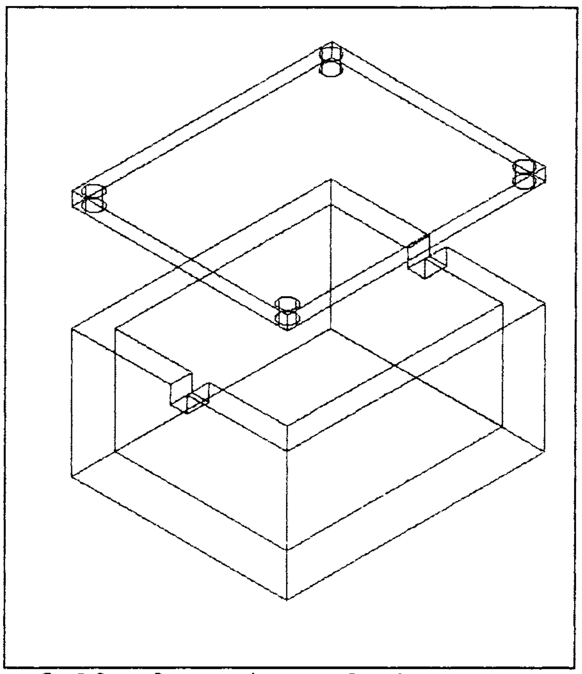


Figure 6-20 3-D View of the External Box

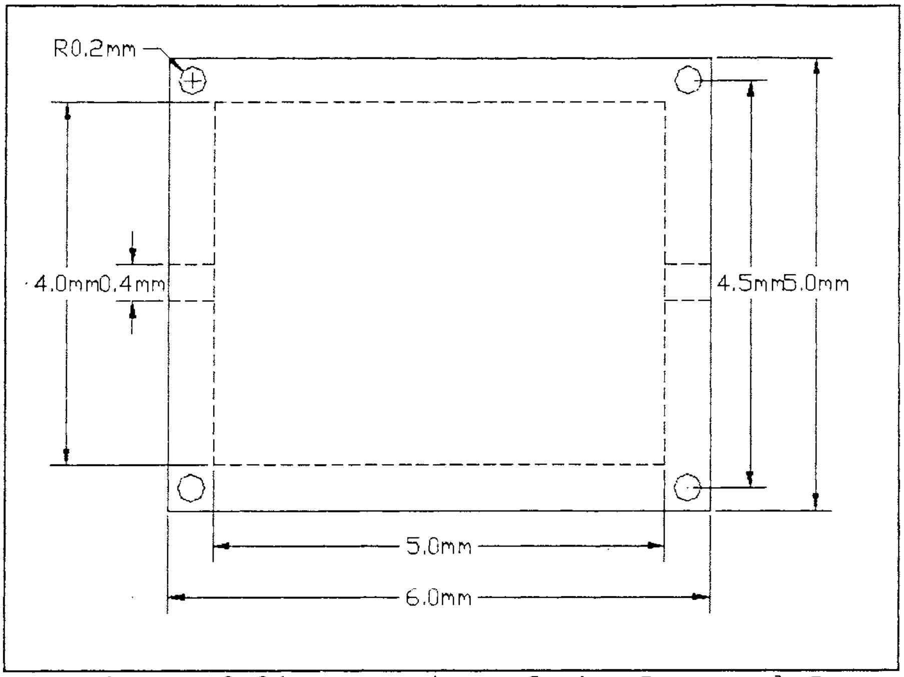


Figure 6-21 Top View of the External Box

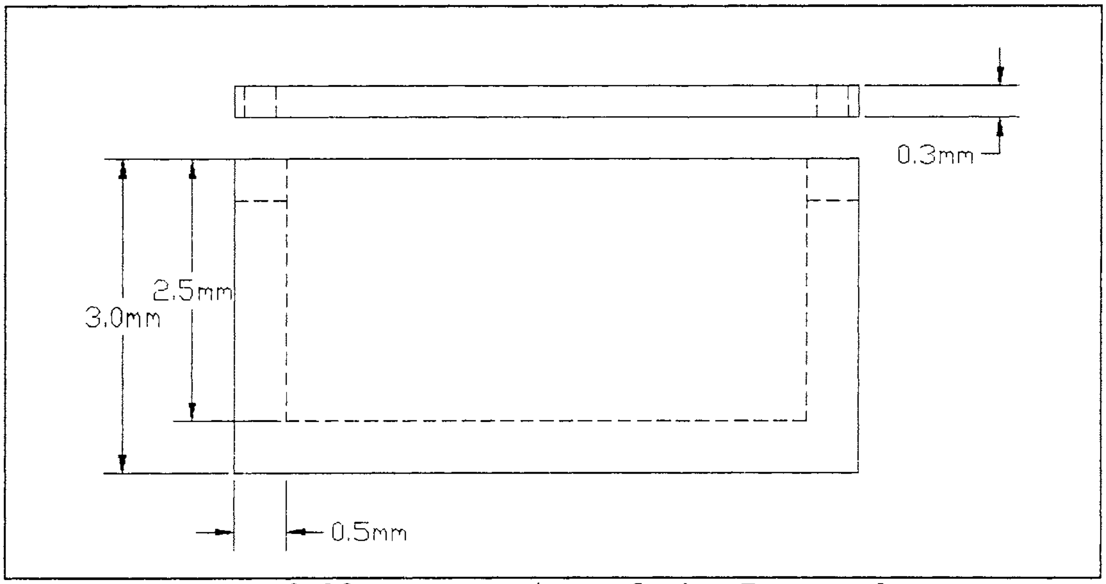


Figure 6-22 Front View of the External Box

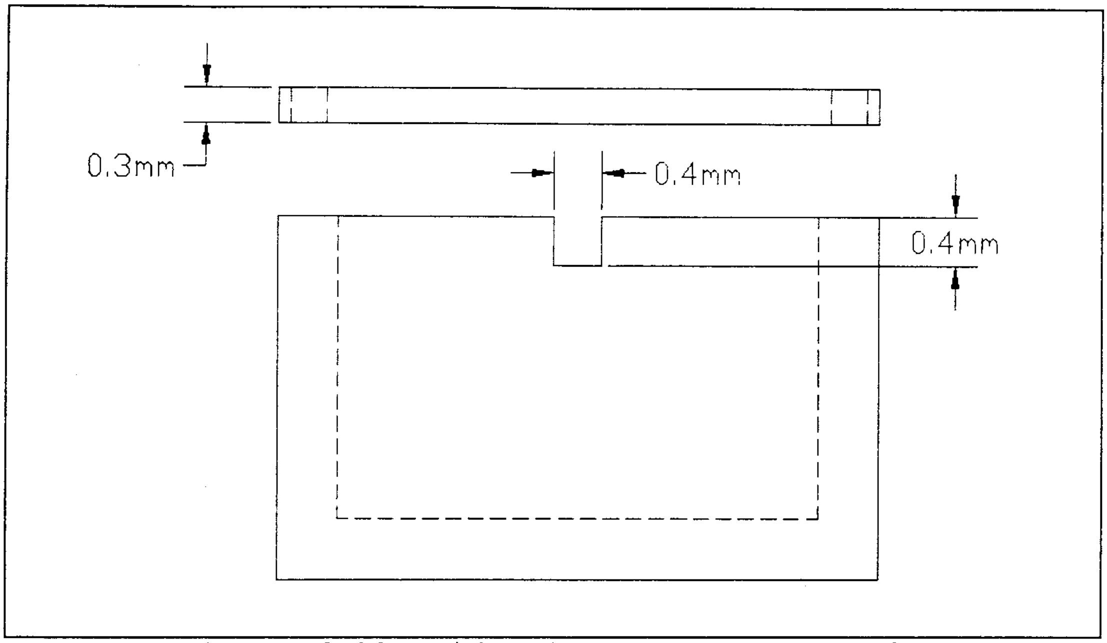


Figure 6-23 Side View of the External Box

Main Shell

Per ISO 9241 requirement for non-keyboard input device/mouse, the overall size of the mouse should be [5]:

1. Sensor - located under fingers
2. Button motion - coincident to finger motion
3. Width - 40mm minimum and 70mm maximum
4. Length - 70mm minimum and 120mm maximum
5. Height - 25mm minimum and 40mm maximum

From a study done by Jon Weimer [4], the average grasp dimensions was 55mm with the maximum at 110mm. This study was based on a 50/50 ratio of men to women. From the same study by Weimer, the average thumb breadth was 21mm with 0.95 percentile at 25mm.

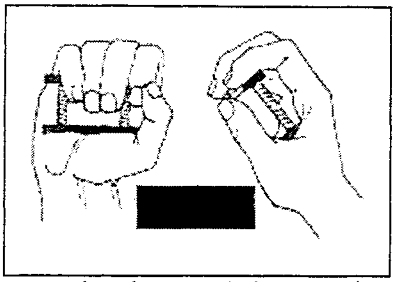


Figure 6-24 Method Used by Weimer to Measure the Dimension of the Grasp

A soft polyester-made shell is used to hold the internal box and serve as the main shell. Users will be squeezing this shell to operate the device.

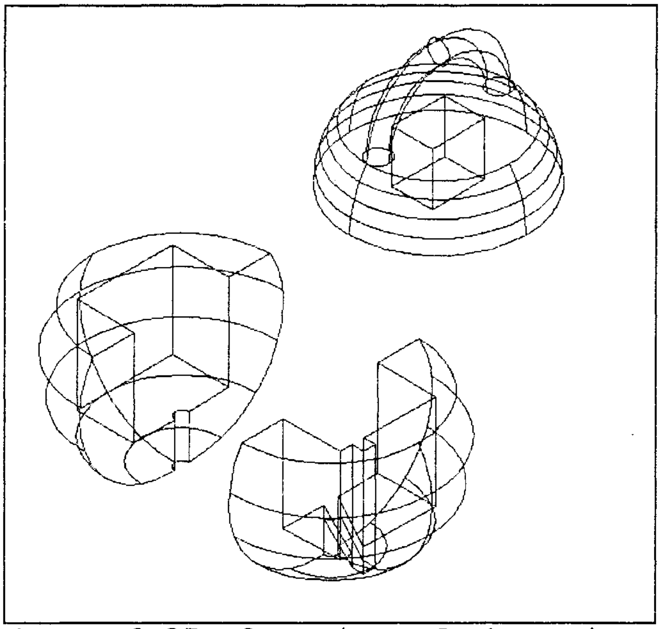


Figure 6-25 3-D View of the Main Shell

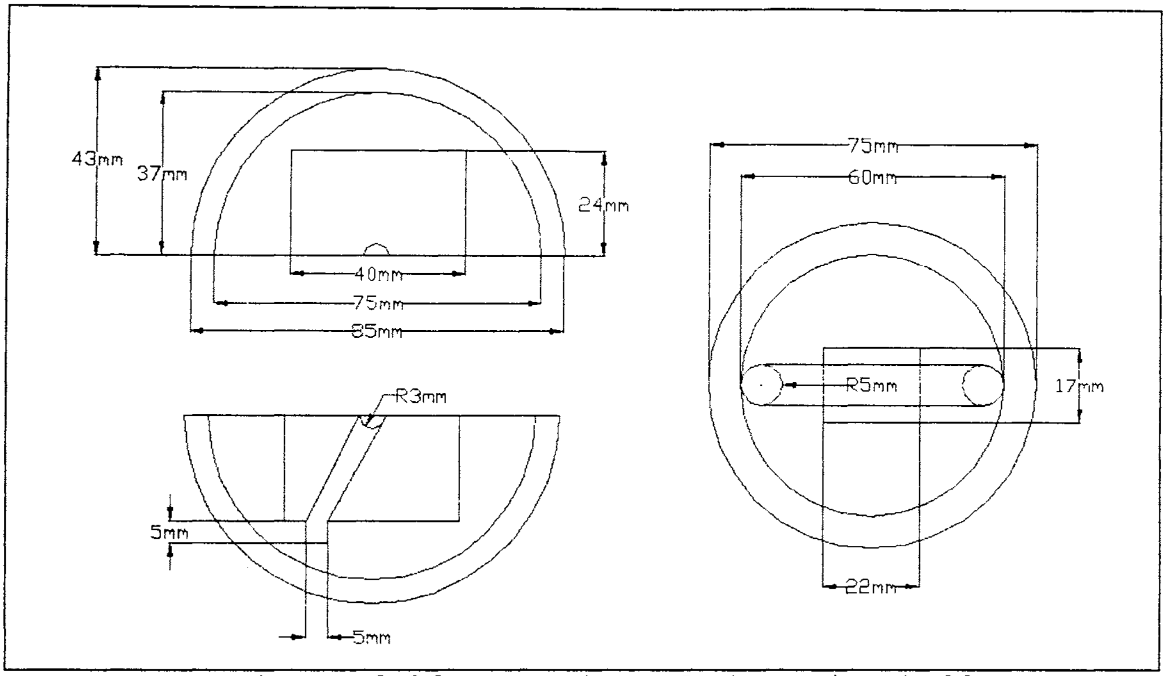


Figure 6-26 Top View of the Main Shell

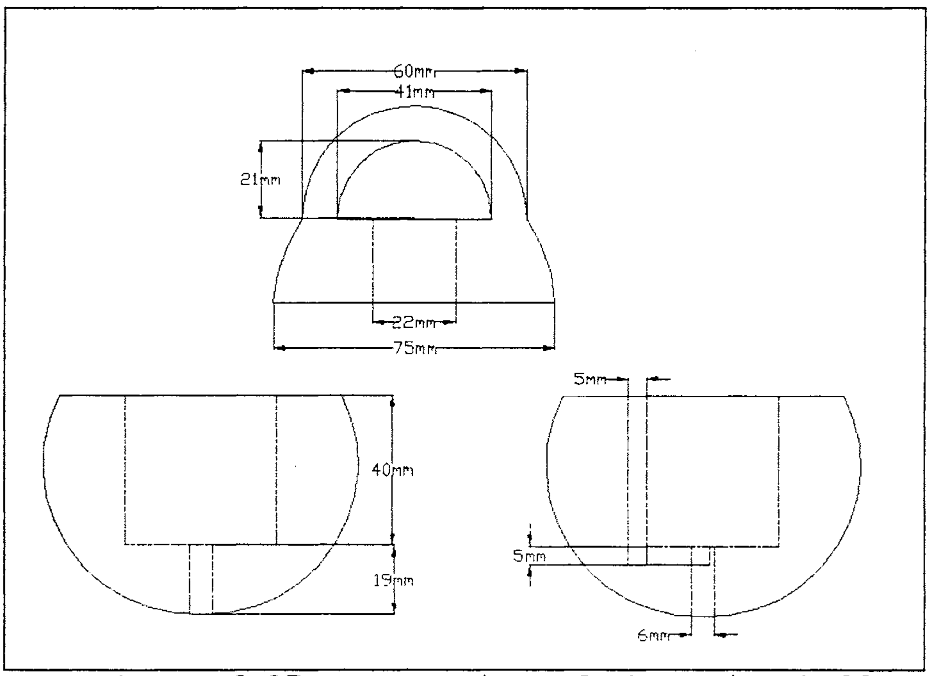


Figure 6-27 Front View of the Main Shell

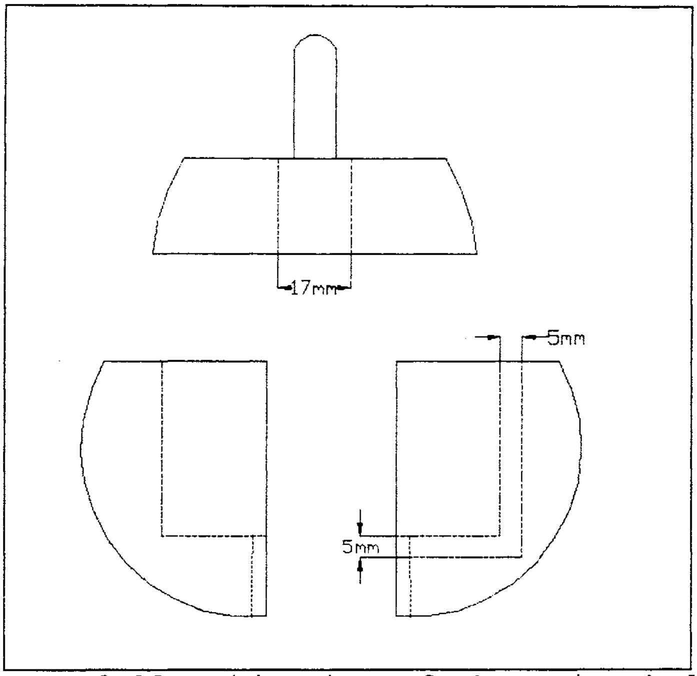


Figure 6-28 Side View of the Main Shell

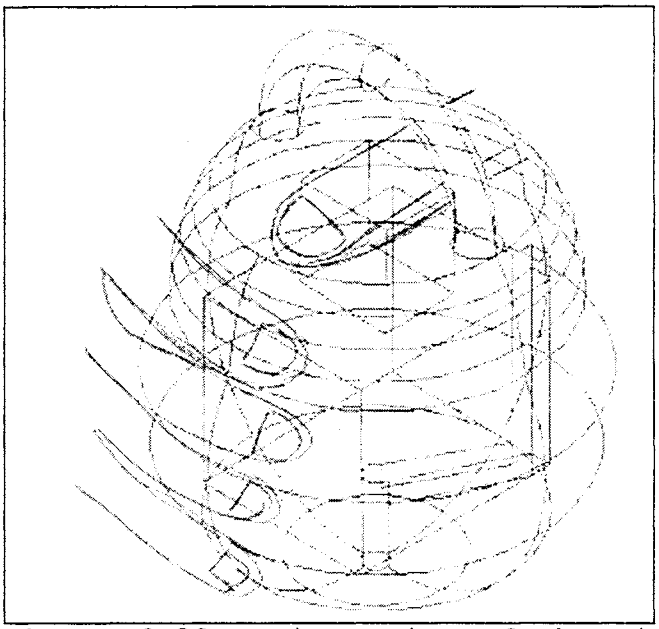


Figure 6-29 Orientation of the Fingers

Assembling

These were the steps in assembling:

1. The excitation circuit and reference voltage circuit were built and then inserted into the internal box with the exception that the rocker switch was inserted into the switch box.
2. The actuator was placed on the top of the internal box.
3. The output cable from the excitation circuit was allowed to come out from the internal box through the hole at the bottom of the box. The cover plate then was used to close the box at the bottom.
4. The whole assembly was put inside the outer shell.
5. The microcontroller circuit was built.
6. The microcontroller circuit was placed inside the external box.
7. The output cable of the microcontroller circuit was allowed to come out through the hole on the top of the box. The Phillip screws were then used to seal the box and the cap.
8. The output cable of the microcontroller was then connected to a standard USB upstream connector.
9. The connector was ready to be plugged into the USB port at the back of the computer.

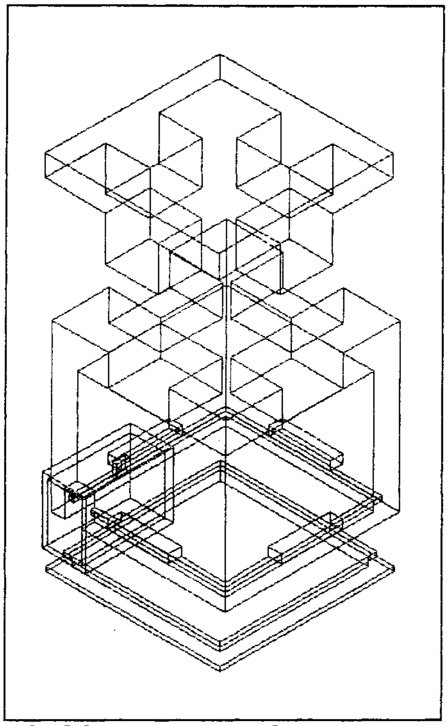


Figure 6-30 Internal Core Assembly

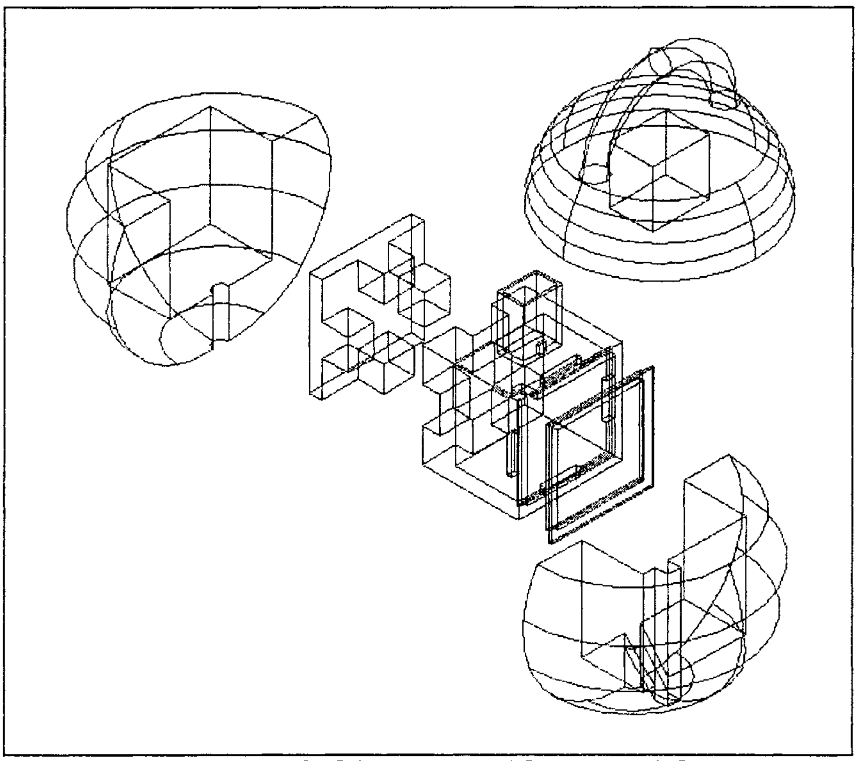


Figure 6-31 Overall Assembly

Device Driver

The device uses a Human Interface Device (HID) from Window as the driver. Therefore, the user does not need any additional driver to operate the device. The enumeration will happen automatically without user intervention.

In addition, the USB 1.1 Specification can be found at http://www.usb.org/developers/data/usbspec.zip .

Microcontroller Firmware

A PICSTART PLUS development programmer kit from Microchip Inc. is used to install the firmware inside the microcontroller. The kit includes MPLAB IDE compiler and all necessary hardware to program the firmware and to connect to computer via a serial RS-232 port. The configuration bits are set as follow:

- Oscillator H4
- Watchdog Timer Off
- Power Up Timer Off
- Code Protect Off

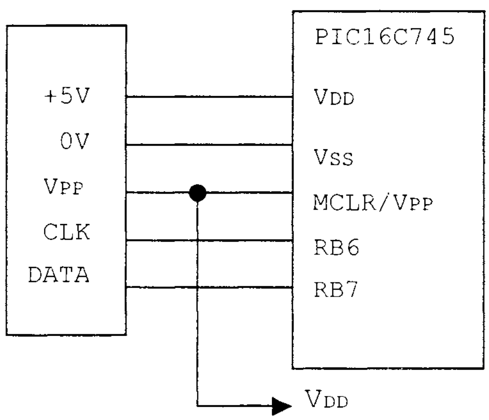


Figure 6-32 Programming Connection

The MPLAB IDE Compiler converts the firmware, input by programmer, from an assembly file to a hexadecimal file. Then the PICSTART PLUS stores these hexadecimal files into the microcontroller.

The five hexadecimal files are:

1. USB_Main.asm
   The main program of the device. The main algorithm of the mouse movement was inserted in this file.
2. USB_Ch9.asm
   Consists of core functions needed to enumerate the device. It also contains the functions that service the USB, send data to the host computer, and receive data from the host computer.
3. HidClass.asm
   Provide the functions for HID Class specific commands.
4. Descript.asm
   Contain a set of descriptors for a standard mouse.
5. USB_Defs.inc
   Contains several microcontroller specific functions including ConfigUSB, PutEP1, PutEP2, GetEP1, and GetEP2. This file works together with USB_Ch9.asm file.

All five files were obtained from Microchip. The only change made was that the additional mouse movement algorithm was added in USB_Main.asm to meet the requirement for the new device. These five files were originally from the previous Microchip USB firmware, which demonstrated a circular cursor movement. The original files can be found at http://www. microchip.com/download/appnote/firmware/usb124as.zip .

Other than the five files above, a linker file for PIC16C745 was also added. The PICSTART Plus will ask during the programming whether the linker file should be included and it can be easily found from PICSTART Plus Library and programmed into the microcontroller. The linker file is used to support the hexadecimal file inside the microcontroller.

Device Testing

Per ISO 9241, there are several testing that could be done for non-keyboard input device [5]. They are:

1. One-direction tapping test - pointing and moving cursor along one axis. For example inserting a cursor at point along a character string.
2. Multi-directional tapping test - pointing in different directions. For example repositioning cursor at different areas.
3. Dragging test - clicking and dragging to specific locations. For example inserting, clicking and dragging cursor along a string of text to highlight it.
4. Tracing test - clicking and dragging objects to specific locations or duplications shapes. For example duplicating lines or shapes area filling of objects.
5. Freehand input test - hand drawn images. For example graphics creation.
6. Grasp and park test - moving the cursor to a specific location on the screen and using a key on the keyboard to click the cursor into place. For example numeric data entry in a spreadsheet.

Manufacturing

The manufacturing process can be divided into two parts. Part 1 consists of all the circuits and connection. Part 2 consists of manufacturing the core and the shell.

Part 1

For research purpose, all the connection in the excitation circuit and reference voltage circuit were done using regular electronic wires and connector ports. In the real manufacturing, these components are replaced by a Prototype Circuit Board (PCB) which is smaller in size.

Figure 6-33 Prototype Circuit Board (PCB)

For research purpose too, the regular ceramic resistors and capacitors were used. In the real manufacturing, these components are replaced by much smaller chip resistors and capacitors. These components are soldered directly onto the PCB.

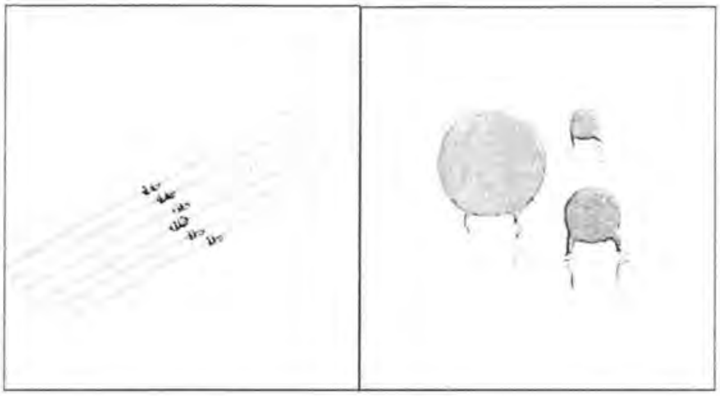


Figure 6-34 Ceramic Resistors and Capacitors

Figure 6-35 Chip Resistors and Capacitors

The microcontroller used in this research was an Electrically Erasable Programmable Read Only Memory (EEPROM) microcontroller. The firmware inside the microcontroller can be reprogrammed again and again. The microcontroller was connected to the other components through a PCB supplied by Microchip. In the real manufacturing, the much cheaper One Time Programmable (OTP) microcontroller is used. The same PCB is used to connect the microcontroller to other components of the circuit.

Part 2

The internal box, cover plate, actuator and switch box were made from aluminum. Each component was manufactured separately through milling process. The switch box was then glued on the top of the internal box.

The outer shell was made from polyester. The inside cavity was done using a knife and scissor. The glue was used to attach the three parts of the outer shell together.

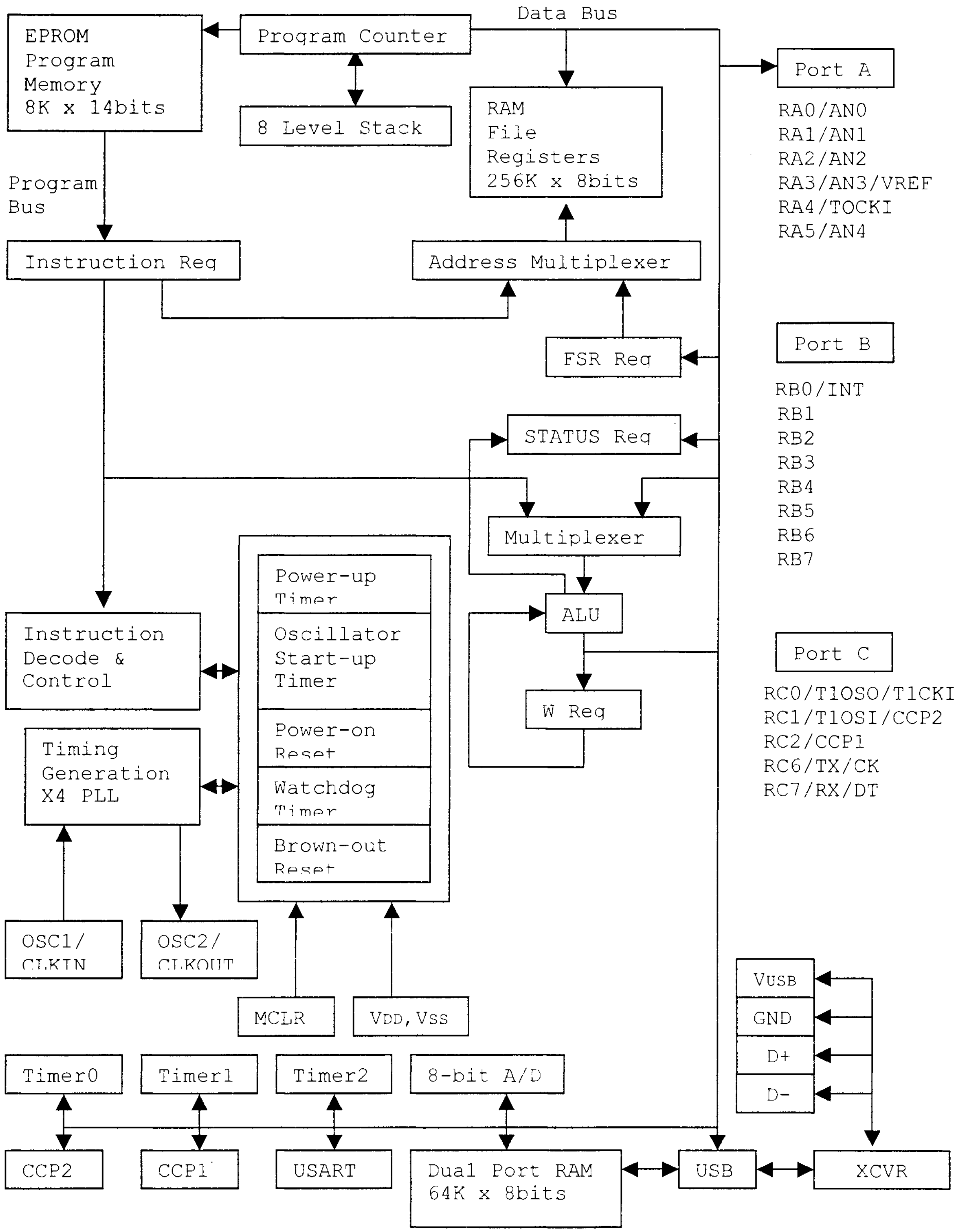
The external box was made from plastic through casting process. Four 8mm Stainless Steel Phillips screws were used to fasten the cover plate of the box to the body of the box.

Part List

Name	Quantity
PIC16C745 Microcontroller	1
Flexiforce Sensor	4
LM324 Quad Op-Amp	1
6 MHz Crystal	1
LM336-2.5 Diode	1
100 kΩ Resistor	4
2.2 kΩ Resistor	1
10 kΩ Resistor	1
1.5 kΩ Resistor	1
0.1 μF Capacitor	5
33 pF Capacitor	2
3.3 μF Capacitor	1
200 nF Capacitor	1
Plastic External Box	1
Aluminum Internal Box	1
Aluminum Actuator	1
Aluminum Cover Plate	1
Aluminum Switch Box	1
Polyester Outer Shell	1
Electrical Wire	2 meter
Screws for External Box	4
USB Connector	1

Table 6-1 Part List

APPENDIX A: PIC16C745 Block Diagram



APPENDIX B: PIC16C745 Pin Description

Name	Function	Description
$\overline{\text{MCLR}}$/V_{PP}	MCLR	Master Clear
	Vpp	Programming Voltage
OSC1/CLKIN	OSC1	Crystal/Resonator
	CLKIN	External Clock Input
OSC2/CLKOUT	OSC2	Crystal/Resonator
	CLKOUT	External Clock (FINT/4) Output
RA0/AN0	RA0	Bi-directional I/O
	AN0	A/D Input
RA1/AN1	RA1	Bi-directional I/O
	AN1	A/D Input
RA2/AN2	RA2	Bi-directional I/O
	AN2	A/D Input
RA3/AN3/VREF	RA3	Bi-directional I/O
	AN3	A/D Input
	VREF	A/D Positive Reference
RA4/TOCKI	RA4	Bi-directional I/O
	TOCKI	Timer 0 Clock Input
RA5/AN4	RA5	Bi-directional I/O
	AN5	A/D Input
RB0/INT	RB0	Bi-directional I/O
	INT	Interrupt
RB1	RB1	Bi-directional I/O
RB2	RB2	Bi-directional I/O
RB3	RB3	Bi-directional I/O
RB4	RB4	Bi-directional I/O
RB5	RB5	Bi-directional I/O
RB6/ICSPC	RB6	Bi-directional I/O
	ICSPC	In-Circuit Serial Programming Clock Input
RB7/ICSPD	RB7	Bi-directional I/O
	ICSPD	In-Circuit Serial Programming Data I/O
RC0/T1OSO/T1CKI	RC0	Bi-directional I/O
	T1OSO	Timer 1 Oscillator Output
	T1CKI	Timer 1 Clock Input
RC1/T1OSI/CCP2	RC1	Bi-directional I/O
	T1OSI	Timer 1 Oscillator Input
	CCP2	Capture In/Compare out/PWM Out 2
RC2/CCP1	RC2	Bi-directional I/O
	CCP1	Capture In/Compare out/PWM Out 1
VUSB	VUSB	Regulator Output Voltage

D-	D-	USB Differential Bus
D+	D+	USB Differential Bus
RC6/TX/CK	RC6	Bi-directional I/O
	TX	USART Async Transmit
	CK	USART Master Out/Slave In Clock
RC7/RX/DT	RC7	Bi-directional I/O
	RX	USART Async Receive
	DT	USART Data I/O
V_{DD}	V_{DD}	Power
V_{SS}	V_{SS}	Ground

APPENDIX C: PIC16C745 Data Memory Map

Bank 0	Addr.	Bank 1	Addr.	Bank 2	Addr.	Bank 3	Addr.
Indirect Addr.	00h	Indirect Addr.	80h	Indirect Addr.	100h	Indirect Addr.	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
PIR2	0Dh	PIE2	8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh		8Fh		10Fh		18Fh
T1CON	10h		90h		110h	UIR	190h
TMR2	11h		91h		111h	UIE	191h
T2CON	12h	PR2	92h		112h	UEIR	192h
	13h		93h		113h	UEIE	193h
	14h		94h		114h	USTAT	194h
CCPR1L	15h		95h		115h	UCTRL	195h
CCPR1H	16h		96h		116h	UADDR	196h
CCP1CON	17h		97h		117h	USWSTAT	197h
RCSTA	18h	TXSTA	98h		118h	UEP0	198h
TXREG	19h	SPBRG	99h		119h	UEP1	199h
RCREG	1Ah		9Ah		11Ah	UEP2	19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	USB Dual Port Memory	1A0h 1Dfh
			EFh		16Fh		1E0h 1EFh
	7Fh	Accesses 70h-7Fh	F0h FFh	Accesses 70h-7Fh	170h 17Fh	Accesses 70h-7Fh	1F0h 1FFh

APPENDIX D: MICROCONTROLLER FIRMWARE
Usb_main.asm

```
;                          Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated (the "Company")
; for its PICmicro(r) Microcontroller is intended and supplied to you, the Company's
; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
; products.
;
; The software is owned by the Company and/or its supplier, and is protected under
; applicable copyright laws. All rights are reserved. Any use in violation of the
; foregoing restrictions may subject the user to criminal sanctions under applicable
; laws, as well as to civil liability for the breach of the terms and conditions of
; this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;###############################################################################
; filename: USB_MAIN.ASM
;           USB Mouse Project.  Developed by John P. Burns and Brenton D. Rothchild
;           on contractual basis for Iowa State University ("ISU").  Any
;           applicable intellectual property, copyrights, or other protections
;           of this software are owned by ISU.
;
; [Microchip's original description of supplied software:                 ]
; [This file implements a basic interrupt service routine and shows how the  ]
; [USB interrupt would be serviced, and also how InitUSB and PutUSB         ]
; [should be called.  It may be used as a reference, or as a starting point  ]
; [from which to build an application.                                      ]
;
; This file uses Microchip's USB sample firmware (as described above), and implements
; ADC routines with averaging math to compute output values for a USB mouse.
;
;###############################################################################
;
; [ Microchip's original software comments:
; [  Author:              Dan Butler and Reston Condit
; [  Company:             Microchip Technology Inc
; [
; [  Revision:            1.24
; [  Date:                5 March 2002
; [  Assembled using:     MPASM 2.61
; [  Configuration Bits: H4 Oscillator, WDT Off, Power up timer off
; [  Revision History:
; [  23 August 2000       DZB Changed descriptor pointers to 16 bits.
; [  24 August 2000       DZB Moved EP1 & 2 configuration from USBReset
; [                           to Set_Configuration to implement requirement in
; [                           USB V1.1 spec paragraph 5.3.1.2
; [  28 August 2000       DZB Force data toggle on OUT packets in PutUSB
; [  20 March 2001        DZB Reduced use of common RAM
; [  20 March 2001        DZB Put and Get use their own temp variable (GPtemp) to
; [                           avoid collisions with the ISR's use of temp.
; [  29 March 2001        DZB Fixed saving of bank bits in GetUSB
; [  02 May 2001          DZB Implemented SHOW_ENUM_STATUS to show enumeration
; [                           status on the PORTB LEDs: 0- Powered, 1- Default,
; [                           2- addressed, 3- configured, 4- sleep,
; [                           5- EP0 Activity, 6- EP1 Activity, 7- EP2 Activity
; [  03 August 2001       RAC Made distinct GetEP and PutEP macros for endpoints 1
; [                           and 2.  These functions are GetEP1, GetEP2, PutEP1, and
; [                           PutEP2.  Instances of the these macros are created in
; [                           usb_ch9.asm.
```

```
; [  08 August 2001      RAC Corrected various banking and paging issues.
; [  15 August 2001      RAC Added Report_desc_index function in descript.asm.
; [                          This function allows more than one report descriptor
; [                          to be used.
; [  08 September 2001   RAC Correctly set DATA0/1 bit (BDndST:<6>) in
; [                          Set_Configuration (usb_ch9.asm). It wasn't being set
; [                          before.
; [  15 January 2002     RAC BD0OST was being written to after control was given
; [                          to the SIE in HID_SET_REPORT.  This was fixed.
; [  01 February 2002    RAC Made sure this version was consistent with the C
; [                          version of the firmware.  Misc changes.
; [  14 February 2002    RAC Corrected USBSleep and USBActivity to suspend and
; [                          unsuspend the SIE respectively
; [  25 February 2002    RAC Remote Wakeup initialization was moved from a
; [                          PORTB interrupt to the RA4 pin.  The move was made
; [                          because this firmware uses PORTB for USB status
; [                          outputs.  RA4 is a button on the PICDEM USB board.
; [                          For users who don't have the PICDEM USB PCB, RA4 is
; [                          active low.
; [  05 March 2002       RAC Clear <UCTRL: SUSPND> bit in USBActivity rather than
; [                          setting it.
;
; Authors: John P. Burns, Brenton D. Rothchild, Troy Benjegerdes
;
; Revision:             1.12
; USB Firmware Rev.:    1.24
; Date:                 28 July 2002
; Assembled using:      MPASM 2.61
; Revision History:
; 25 July 2002          BDR,JPB Started with USB Firmware, added ADC code.
; 26 July 2002          BDR,JPB Removed Remote Wakeup capability from USB Firmware.
; 27 July 2002          BDR,TB  Added averaging routines, test mode capability.
; 28 July 2002          BDR     Code freeze for revision 1.12.
;
;###############################################################################
;
;   include files:
;       P16C765.inc     Rev 1.00
;       usb_defs.inc    Rev 1.10
;
;###############################################################################
#include        <p16c745.inc>
#include        "usb_defs.inc"

        __CONFIG        _H4_OSC & _WDT_OFF & _PWRTE_OFF & _CP_OFF

unbanked        udata_shr
W_save          res     1       ; register for saving W during ISR

bank0           udata
Status_save     res     1       ; registers for saving context
PCLATH_save     res     1       ;  during ISR
FSR_save        res     1
CUR_STAT        res     1       ; Direction cursor moves on the screen
BUFFER          res     8       ; Location for data to be sent to host
COUNTER         res     1       ; General counter variable
INNER           res     1       ; Loop control variable
OUTER           res     1       ; Loop control variable
input           res     4       ; 4-byte input variable for 4 channels
a_h             res     4       ; 4-byte average (high bytes)
a_l             res     4       ; 4-byte average (low bytes)
offset          res     1       ; Channel offset variable
templ           res     1       ; Low byte temp variable
temph           res     1       ; High byte temp variable
FIXED0          RES     1       ; Fixed value for Test Mode
FIXED1          RES     1       ; Fixed value for Test Mode
TESTMODE        RES     1       ; Test Mode variable
temp            RES     1       ; General temporary variable
```

```
        extern          InitUSB
        extern          PutEP1
        extern          GetEP1
        extern          ServiceUSBInt
        extern          CheckSleep
        extern          RemoteWakeup            ; Remote Wakeup works with the use of the RA4
                                                ; pin (active low)

STARTUP code
        pagesel         Main
        goto            Main
        nop

InterruptServiceVector
        movwf           W_save                  ; save W
        movf            STATUS,W
        clrf            STATUS                  ; force to page 0
        movwf           Status_save             ; save STATUS
        movf            PCLATH,W
        movwf           PCLATH_save             ; save PCLATH
        movf            FSR,W
        movwf           FSR_save                ; save FSR

; ***************************************************************
; Interrupt Service Routine
; First we step through several stages, attempting to identify the source
; of the interrupt.
; ********************************************************************

Process_ISR
; Step 1, what triggered the interrupt?

        btfsc           INTCON,T0IF             ; Timer 0
        nop
        btfsc           INTCON,RBIF             ; Port B
        nop

TEST_INTCON
        btfsc           INTCON,INTF             ; External Interrupt
        nop
        banksel         PIR1
        pagesel         ServiceUSBInt
        btfsc           PIR1,USBIF              ; USB interrupt flag
        call            ServiceUSBInt           ; Service USB interrupt

TEST_PIR1
        btfsc           PIR1,ADIF               ; AD Done?
        nop
        btfsc           PIR1,RCIF
        nop
        btfsc           PIR1,TXIF
        nop
        btfsc           PIR1,CCP1IF
        nop
        btfsc           PIR1,TMR2IF
        nop
        btfsc           PIR1,TMR1IF
        nop
        btfsc           PIR2,CCP2IF
        nop

; ********************************************************************
; End ISR, restore context and return to the Main program
; ********************************************************************
EndISR
        clrf            STATUS                  ; select bank 0
        movf            FSR_save,W              ; restore the FSR
        movwf           FSR
        movf            PCLATH_save,W           ; restore PCLATH
```

```
        movwf           PCLATH
        movf            Status_save,W           ; restore Status
        movwf           STATUS
        swapf           W_save,F                ; restore W without corrupting STATUS
        swapf           W_save,W
        retfie

        code
; ****************************************************************
; test program that sets up the buffers and calls the ISR for processing.
;
; ****************************************************************
Main
        movlw           .30                     ; delay 16 uS to wait for USB to reset
        movwf           W_save                  ; SIE before initializing registers
        decfsz          W_save,F                ; inner is merely a convienient register
        goto            $-1                     ; to use for the delay counter.

        banksel         TRISB
        movlw           0x07                    ; 7 - 0 = Output
                                                ; 6 - 0 = Output
                                                ; 5 - 0 = Output
                                                ; 4 - 0 = Output
                                                ; 3 - 0 = Output
                                                ; 2 - 1 = Input (Test Mode input)
                                                ; 1 - 1 = Input (Right button)
                                                ; 0 - 1 = Input (Left button)
        movwf           TRISB

        banksel         TRISA
        movlw           0xff                    ; 7 - Unimplemented
                                                ; 6 - Unimplemented
                                                ; 5 - Unimplemented
                                                ; 4 - 1 = Input, Channel 3 (Y-)
                                                ; 3 - 1 = Input, Vref
                                                ; 2 - 1 = Input, Channel 2 (Y+)
                                                ; 1 - 1 = Input, Channel 1 (X-)
                                                ; 0 - 1 = Input, Channel 0 (X+)

        movwf           TRISA

        banksel         OPTION_REG
        bcf             OPTION_REG,NOT_RBPU     ; Turn on weak pull-ups on PORTB

        banksel         PORTB
        clrf            PORTB

        banksel         PORTA
        clrf            PORTA

; Initialize variables
        clrf            offset
        clrf            input
        clrf            input+1
        clrf            input+2
        clrf            input+3
        clrf            a_h
        clrf            a_h+1
        clrf            a_h+2
        clrf            a_h+3
        clrf            a_l
        clrf            a_l+1
        clrf            a_l+2
        clrf            a_l+3
        clrf            templ
        clrf            temph

        clrf            FIXED0
        clrf            FIXED1
```

```
        clrf            TESTMODE

        pagesel         InitUSB                  ; These six lines of code show the appropriate
        call            InitUSB                  ; way to initialize the USB. First, initialize
                                                 ; the USB (wait for host enumeration) then wait
        pagesel         SKIPUSB
        banksel         TESTMODE
        btfsc           TESTMODE,0               ; If the lowest bit of TESTMODE is set,
        goto            SKIPUSB                  ; goto SKIPUSB, to skip waiting for
                                                 ; ConfiguredUSB
        nop
        ConfiguredUSB                            ; until the enumeration process to complete.
        nop
SKIPUSB
        bcf             STATUS,RP0               ; Make sure you include all pagesels and return
        bcf             STATUS,RP1               ; to the desired bank (in this case Bank 0.)

;configure A/D

        banksel         ADRES
        clrf            ADRES

        banksel         ADCON0
        movlw           0x80                     ; 7-6:ADCS<1:0> A/D Conversion Clock Select bits
                                                 ;     10 = Fint/32 to maintain accuracy at 24MHz
                                                 ; 5-3:CHS<2:0> Analog Channel Select bits
                                                 ;     000 = channel 0, (RA0/AN0)
                                                 ;   2:GO/DONE: A/D Conversion Status bit
                                                 ;     0 = A/D conversion not in progress
                                                 ;   1:Unimplemented
                                                 ;   0:ADON: A/D On bit
                                                 ;     0 = A/D converter module is shut off
        movwf           ADCON0

        banksel         ADCON1
        movlw           0x01                     ; 7-3:Unimplemented
                                                 ; 2-0:PFCG<2:0> A/D Port Configuration Control
                                                 ; bits
                                                 ;     001 - AN5,AN4,AN2,AN1,AN0 are analog
                                                 ;              inputs, and AN3 is Vref.
                                                 ;
        movwf           ADCON1

;A/d configure end

        pagesel         start
        Goto            start

; Delay16 - Delay for 16us (roughly)
;   This loop is used for the worst case Tacq of the ADC. [See section
;   12.1, DS41124C-page 95, PIC16C745/765 Datasheet.]
;
;   INNER is calculated from a clock cycle of 666.6667ns (1/[6MHz/4])
;   and that 'decfsz' and 'goto' (while INNER > 0) take 3 instructions.
;   Thus 8*3 (=24) instructions are to be used.  An extra instruction
;   cycle will be used on the last loop iteration (INNER = 0) because
;   'decfsz' and 'return' will both take 2 instructions.
;
Delay16
        banksel         INNER
        movlw           0x08
        movwf           INNER                    ; 8 -> INNER

Delay16_InnerLoop
        decfsz          INNER,F
        goto            Delay16_InnerLoop        ; Loop until INNER = 0
        return
```

```
start
;*******************************************************************
; CursorDemo
; Generate USB mouse data and send it to the PC
;*******************************************************************
CursorDemo
       banksel        BUFFER
       clrf           BUFFER                 ; 0 -> BUFFER
       clrf           BUFFER+1               ; 0 -> BUFFER+1
       clrf           BUFFER+2               ; 0 -> BUFFER+2
       clrf           BUFFER+3               ; 0 -> BUFFER+3

;Begin X+ Acquisition ---------------------------
;a/d conversion - channel 0
       banksel        ADCON0
       movlw          0x80                   ; 7-6:ADCS<1:0> A/D Conversion Clock Select bits
                                             ;     10 = Fint/32 to maintain accuracy at 24MHz
                                             ; 5-3:CHS<2:0>  - Analog Channel Select bits
                                             ;     000 = channel 0, (RA0/AN0)
                                             ;   2:GO/DONE: A/D Conversion Status bit
                                             ;     0 = A/D conversion not in progress
                                             ;   1:Unimplemented
                                             ;   0:ADON: A/D On bit
                                             ;     0 = A/D converter module is shut off
       movwf          ADCON0
       bsf            ADCON0,ADON            ; enable ADC
       call           Delay16                ; Delay 16us for Tacq
       bsf            ADCON0,GO_DONE         ; start conversion on channel 0

AdwaitCH0
       btfsc          ADCON0,GO_DONE         ; If GO_DONE is not 0,
       goto           AdwaitCH0              ; goto AdwaitCH0
       movlw          0x00                   ; offset -> W
       addlw          input                  ; input+W(offset) -> W
       bcf            STATUS,IRP             ; Make sure we're in Bank 0/1
       movwf          FSR                    ; Set FSR at input+offset
       movf           ADRES,W                ; ADC result -> W
       btfsc          TESTMODE,0             ; If we're in Test Mode,
       movf           FIXED0,W               ; move fixed value into W
       movwf          INDF                   ; W -> index+offset
       bcf            ADCON0,ADON            ; disable ADC
;End X+ Acquisition -----------------------------

;Begin X- Acquisition ---------------------------
       banksel        ADCON0
       movlw          0x88                   ; 7-6:ADCS<1:0> A/D Conversion Clock Select bits
                                             ;     10 = Fint/32 to maintain accuracy at 24MHz
                                             ; 5-3:CHS<2:0>  - Analog Channel Select bits
                                             ;     001 = channel 1, (RA1/AN1)
                                             ;   2:GO/DONE: A/D Conversion Status bit
                                             ;     0 = A/D conversion not in progress
                                             ;   1:Unimplemented
                                             ;   0:ADON: A/D On bit
                                             ;     0 = A/D converter module is shut off
       movwf          ADCON0
       bsf            ADCON0,ADON            ; enable ADC
       call           Delay16                ; Delay 16us for Tacq
       bsf            ADCON0,GO_DONE         ; start conversion on channel 0

AdwaitCH1
       btfsc          ADCON0,GO_DONE         ; If GO_DONE is not 0,
       goto           AdwaitCH1              ; goto AdwaitCH1
       movlw          0x01                   ; offset -> W
       addlw          input                  ; input+W(offset) -> W
       bcf            STATUS,IRP             ; Make sure we're in Bank 0/1
       movwf          FSR                    ; Set FSR at input+offset
       movf           ADRES,W                ; ADC result -> W
       btfsc          TESTMODE,0             ; If we're in Test Mode,
       movf           FIXED1,W               ; move fixed value into W
```

```
        movwf           INDF                    ; W -> index+offset
        bcf             ADCON0,ADON             ; disable ADC

        movlw           0x00                    ; offset -> W
        call            AVG                     ; Perform averaging on input+offset
;End X- Acquisition -----------------------------

;Begin Y+ Acquisition ---------------------------
        banksel         ADCON0
        movlw           0x90                    ; 7-6:ADCS<1:0> A/D Conversion Clock Select bits
                                                ;     10 = Fint/32 to maintain accuracy at 24MHz
                                                ; 5-3:CHS<2:0>  - Analog Channel Select bits
                                                ;     010 = channel 2, (RA2/AN2)
                                                ;   2:GO/DONE: A/D Conversion Status bit
                                                ;     0 = A/D conversion not in progress
                                                ;   1:Unimplemented
                                                ;   0:ADON: A/D On bit
                                                ;     0 = A/D converter module is shut off
        movwf           ADCON0
        bsf             ADCON0,ADON             ; enable ADC
        call            Delay16                 ; Delay 16us for Tacq
        bsf             ADCON0,GO_DONE          ; start conversion on channel 0

AdwaitCH2
        btfsc           ADCON0,GO_DONE          ; If GO_DONE is not 0,
        goto            AdwaitCH2               ; goto AdwaitCH2
        movlw           0x02                    ; offset -> W
        addlw           input                   ; input+W(offset) -> W
        bcf             STATUS,IRP              ; Make sure we're in Bank 0/1
        movwf           FSR                     ; Set FSR at input+offset
        movf            ADRES,W                 ; ADC result -> W
        btfsc           TESTMODE,0              ; If we're in Test Mode,
        movf            FIXED2,W                ; move fixed value into W
        movwf           INDF                    ; W -> index+offset
        bcf             ADCON0,ADON             ; disable ADC

        movlw           0x01                    ; offset -> W
        call            AVG                     ; Perform averaging on input+offset
;End Y+ Acquisition -----------------------------

;Begin Y- Acquisition ---------------------------
        banksel         ADCON0
        movlw           0xA0                    ; 7-6:ADCS<1:0> A/D Conversion Clock Select bits
                                                ;     10 = Fint/32 to maintain accuracy at 24MHz
                                                ; 5-3:CHS<2:0>  - Analog Channel Select bits
                                                ;     100 = channel 4, (RA5/AN4)
                                                ;   2:GO/DONE: A/D Conversion Status bit
                                                ;     0 = A/D conversion not in progress
                                                ;   1:Unimplemented
                                                ;   0:ADON: A/D On bit
                                                ;     0 = A/D converter module is shut off
        movwf           ADCON0
        bsf             ADCON0,ADON             ; enable ADC
        call            Delay16                 ; Delay 16us for Tacq
        bsf             ADCON0,GO_DONE          ; start conversion on channel 0

AdwaitCH3
        btfsc           ADCON0,GO_DONE          ; If GO_DONE is not 0,
        goto            AdwaitCH3               ; goto AdwaitCH3
        movlw           0x03                    ; offset -> W
        addlw           input                   ; input+W(offset) -> W
        bcf             STATUS,IRP              ; Make sure we're in Bank 0/1
        movwf           FSR                     ; Set FSR at input+offset
        movf            ADRES,W                 ; ADC result -> W
        btfsc           TESTMODE,0              ; If we're in Test Mode,
        movf            FIXED3,W                ; move fixed value into W
        movwf           INDF                    ; W -> index+offset
        bcf             ADCON0,ADON             ; disable ADC
```

```
        movlw           0x02                    ; offset -> W
        call            AVG                     ; Perform averaging on input+offset
;End Y- Acquisition --------------------------------

        movlw           0x03                    ; offset -> W
        call            AVG                     ; Perform averaging on input+offset

;Compute X magnitude values
        movlw           0x01                    ; offset -> W
        addlw           a_h                     ; a_h+W -> W
        movwf           FSR                     ; Set FSR at a_h+offset
        movf            INDF,W                  ; a_h+offset -> W
        movwf           templ                   ; W -> templ
        bcf             STATUS,C                ; Clear carry bit
        rrf             templ                   ; templ/2 -> templ

        movlw           0x00                    ; offset -> W
        addlw           a_h                     ; a_h+W -> W
        movwf           FSR                     ; Set FSR at a_h+offset
        movf            INDF,W                  ; a_h+offset -> W
        movwf           temph                   ; W -> temph
        bcf             STATUS,C                ; Clear carry bit
        rrf             temph                   ; temph/2 -> temph
        movf            templ,W                 ; templ -> W
        subwf           temph,W                 ; temph-W -> W
        movwf           BUFFER+1                ; W -> BUFFER+1

        movlw           0x03                    ; offset -> W
        addlw           a_h                     ; a_h+W -> W
        movwf           FSR                     ; Set FSR at a_h+offset
        movf            INDF,W                  ; a_h+offset -> W
        movwf           templ                   ; W -> templ
        bcf             STATUS,C                ; Clear carry bit
        rrf             templ                   ; templ/2 -> templ

        movlw           0x02                    ; offset -> W
        addlw           a_h                     ; a_h+W -> W
        movwf           FSR                     ; Set FSR at a_h+offset
        movf            INDF,W                  ; a_h+offset -> W
        movwf           temph                   ; W -> temph
        bcf             STATUS,C                ; Clear carry bit
        rrf             temph                   ; temph/2 -> temph
        movf            templ,W                 ; templ -> W
        subwf           temph,W                 ; temph-W -> W
        movwf           BUFFER+2                ; W -> BUFFER+2

        clrf            BUFFER+3                ; Clear BUFFER+3 (USB ReportID)

;mouse buttons
        banksel         PORTB
        movf            PORTB,W                 ; PORTB states -> W
        andlw           0x03                    ; Mask off lower 2 bits
        xorlw           0x03                    ; Invert their states
        banksel         BUFFER
        movwf           BUFFER                  ; W -> BUFFER (USB button byte)
;end mouse buttons

;Send USB Packet
CursorDemo1
        clrf            INNER                   ; INNER and OUTER are delay registers
        movlw           2
        movwf           OUTER                   ; 2 -> OUTER
        banksel         BUFFER
        bcf             STATUS,IRP              ; Make sure we're in Bank 0/1
        movlw           BUFFER                  ; Send four bytes to the PC starting
        movwf           FSR                     ;  with BUFFER
        movlw           4                       ; Number of bytes to send
        pagesel         PutEP1
        call            PutEP1                  ; Send the bytes
```

```
        pagesel         CursorDemo
        goto            CursorDemo              ; Start all over again

; AVG - Averages the current input against a running 16-bit average (a_h:a_l).
;   Assumes W is the offset (0-3) for the input and average registers.
;
;   Uses temph and templ.
;
;   Modifies a_h:a_l as a return value. a_h:a_l is 15/16 of the previous a_h:a_l
;    value plus 1/16 of the new input value from the ADC channel.
AVG
        movwf           offset                  ; Save offset from W

        movlw           0x04
        movwf           OUTER                   ; 4 -> OUTER, used for a loop

        ; Move a_h+offset -> temph; a_h+offset = 0
        movf            offset,W
        addlw           a_h
        bcf             STATUS,IRP
        movwf           FSR
        movf            INDF,W
        movwf           temph
        clrf            INDF

        ; Move a_l+offset -> templ; a_l+offset = 0
        movf            offset,W
        addlw           a_l
        bcf             STATUS,IRP
        movwf           FSR
        movf            INDF,W
        movwf           templ
        clrf            INDF

AVG_BEGIN

        ; Call RRT with 1 loop to divide by 2
        movlw           0x01
        call            RRT

        call            ADDATOT                 ; Add temph:templ to a_h:a_l

        decfsz          OUTER
        goto            AVG_BEGIN               ; Loop until OUTER = 0

        ; temph = 0
        clrf            temph

        ; Move input+offset -> templ
        movf            offset,W
        addlw           input
        bcf             STATUS,IRP
        movwf           FSR
        movf            INDF,W
        movwf           templ

        ; Call RLT with 4 loops to multiply by 16
        movlw           0x04
        call            RLT

        call            ADDATOT                 ; Add temph:templ to a_h:a_l

        return

; RLT - Rotate Left Temp.  Rotates the 16-bit temp value stored in temph:templ.
;   Assumes W is the number of rotates that should be performed.
;
;   Uses temph and templ.
;
```

```
;   Modifies temph:templ to be rotated left by W times.
RLT
        movwf           INNER                   ; W -> INNER
RLT_loop
        bcf             STATUS,C                ; Clear carry bit
        rlf             temph                   ; Rotate temph left
        bcf             STATUS,C                ; Clear carry bit
        rlf             templ                   ; Rotate templ left
        btfsc           STATUS,C                ; If there was a carry up from templ,
        bsf             temph,0                 ;  set the lowest bit of temph
        decfsz          INNER
        goto            RLT_loop                ; Loop until INNER = 0
        return

; RRT - Rotate Right Temp. Rotates the 16-bit temp value stored in temph:templ.
;   Assumes W is the number of rotates that should be performed.
;
;   Uses temph and templ.
;
;   Modifies temph:templ to be rotated right by W times.

RRT
        movwf           INNER                   ; W -> INNER
RRT_loop
        bcf             STATUS,C                ; Clear carry bit
        rrf             templ                   ; Rotate templ right
        bcf             STATUS,C                ; Clear carry bit
        rrf             temph                   ; Rotate temph right
        btfsc           STATUS,C                ; If there was a carry down from temph,
        bsf             templ,7                 ; set the highest bit of templ
        decfsz          INNER
        goto            RRT_loop                ; Loop until INNER = 0
        return

; ADDATOT - Adds a_h:a_l to temph:templ
;  Uses a_h and a_l.
;
;  Modifies a_h:a_l to be a_h:a_l+temph:templ
ADDATOT
        movf            offset,W                ; offset -> W
        addlw           a_l                     ; a_l+W(offset) -> W
        movwf           FSR                     ; Set FSR at a_l+offset
        movf            INDF,W                  ; a_l+offset -> W
        addwf           templ,W                 ; templ+W -> W
        movwf           INDF                    ; W -> a_l+offset
        clrf            temp                    ; 0 -> temp
        btfsc           STATUS,C                ; If there was a carry from the add,
        bsf             temp,0                  ;  set the lowest bit in temp

        movf            offset,W                ; offset -> W
        addlw           a_h                     ; a_h+W(offset) -> W
        movwf           FSR                     ; Set FSR at a_h+offset
        btfsc           temp,0                  ; If the lowest bit in temp is set,
        incf            INDF                    ;  increment a_h+W to account for the carry up
        movf            INDF,W                  ; a_h+offset -> W
        addwf           temph,W                 ; temph+W -> W
        movwf           INDF                    ; W -> a_h+offset

        return

end                                     ; END OF CODE
```

APPENDIX E: MICROCONTROLLER FIRMWARE
Usb_ch9.asm

```
;                               Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated (the "Company")
; for its PICmicro(r) Microcontroller is intended and supplied to you, the Company's
; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
; products.
;
; The software is owned by the Company and/or its supplier, and is protected under
; applicable copyright laws. All rights are reserved. Any use in violation of the
; foregoing restrictions may subject the user to criminal sanctions under applicable
; laws, as well as to civil liability for the breach of the terms and conditions of
; this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
; ###############################################################################
;    filename:           USB_CH9.ASM
;
; Implements the chapter 9 enumeration commands for Microchip's
; PIC16C7x5 parts.
;
; ###############################################################################
;
;    Author(s):          Dan Butler and Reston Condit
;    Company:            Microchip Technology Inc
;
;    Revision:           1.24
;    Date:               5 March 2002
;    Assembled using     MPASM 2.61
;################################################################################
;
;     include files:
;        P16C765.inc     Rev 1.00
;        usb_defs.inc    Rev 1.00
;
;################################################################################

#include <p16C765.inc>
#include "usb_defs.inc"

    errorlevel -302 ; supress "register not in bank0, check page bits" message

;#define SHOW_ENUM_STATUS

unbanked     udata_shr   ; these will get assigned to unbanked RAM (0x70-0x7F)
temp         res     1   ; short term temp register used in Get Interface
GPtemp       res     1   ; temporary storage location used in Get and PutEPn

    global  BufferDescriptor
    global  BufferData
    global  temp
    global  temp2
    global  EP0_maxLength
    global  EP0_start
    global  EP0_end

bank2   udata
BufferDescriptor    res     3
BufferData          res     8
```

```
USBMaskedInterrupts res     1
USB_Curr_Config     res     1
USB_status_device   res     1   ; status of device
USB_dev_req         res     1
USB_address_pending res     1
USBMaskedErrors     res     1
PIDs                res     1
EP0_start           res     2   ; pointer to first byte of data to send
EP0_end             res     1   ; pointer to last byte of data to send
EP0_maxLength       res     1
temp2               res     1
bufindex            res     1
USB_Interface       res     3   ; allow 3 interfaces to have alternate endpoints
inner               res     1
outer               res     1
dest_ptr            res     1   ; used in buffer copies for Get and
source_ptr          res     1   ; Put USB calls
hid_dest_ptr        res     1   ; used in buffer copies for HIDSetReport
hid_source_ptr      res     1   ;
counter             res     1
bytecounter         res     1   ; saved copy that will be returned in W
RP_save             res     1   ; save bank bits while copying buffers
IS_IDLE             res     1
USB_USTAT           res     1   ; copy of the USTAT register before clearing TOK_DNE

    global  USB_Curr_Config
    global  USB_status_device
    global  USB_dev_req
    global  USB_Interface

#ifdef      COUNTERRORS
USB_PID_ERR         res     2   ; 16 bit counters for each error condition
USB_CRC5_ERR        res     2
USB_CRC16_ERR       res     2
USB_DFN8_ERR        res     2
USB_BTO_ERR         res     2
USB_WRT_ERR         res     2
USB_OWN_ERR         res     2
USB_BTS_ERR         res     2
#endif

    extern  Config_desc_index
    extern  Descriptions
    extern  string_index
    extern  String0
    extern  String0_end
    extern  ClassSpecificRequest
    extern  Check_Class_Specific_IN
    extern  Get_Report_Descriptor
    extern  Get_HID_Descriptor
    extern  DeviceDescriptor
    extern  StringDescriptions

; ******************************************************************
; This section contains the functions to interface with the main
; application.
; ******************************************************************

interface       code

; ******************************************************************
; GETEP1 and GETEP2
;
; Note:  These functions are, in reality, macros defined in usb_defs.inc.
;        To save ROM, delete the instances below that you will not need.
;
; Enter with buffer pointer in IRP+FSR.
; Checks the semaphore for the OUT endpoint, and copies the buffer
; if available.  Restores the bank bits as we found them.
```

```
;
; Returns the bytecount in the W register and return status in the carry
; bit as follows:
; 0 - no buffer available,
; 1 - Buffer copied and buffer made available for next transfer.
;
; The number of bytes moved is returned in W reg.
; ******************************************************************

       GETEP1          ; create instance of GETEP1
       GETEP2          ; create instance of GETEP2

; ******************************************************************
; PUTEP1 and PUTEP2
;
; Note:  These functions are, in reality, macros defined in usb_defs.inc.
;        To save ROM, delete the instances below that you will not need.
;
; Enter with bytecount in W and buffer pointer in IRP+FSR.
; the bytecount is encoded in the lower nybble of W.
;
; Tests the owns bit for the IN side of the specified Endpoint.
; If we own the buffer, the buffer pointed to by the FSR is copied
; to the EPn In buffer, then the owns bit is set so the data will be
; TX'd next time polled.
;
; Returns the status in the carry bit as follows:
; 1 - buffer available and copied.
; 0 - buffer not available (try again later)
; ******************************************************************

       PUTEP1       ; create instance of PUTEP1
       PUTEP2       ; create instance of PUTEP2

; ******************************************************************
; Stall Endpoint.
; Sets the stall bit in the Endpoint Control Register.  For the control
; Endpoint, this implements a Protocol stall and is used when the request
; is invalid for the current device state.  For non-control Endpoints,
; this is a Functional Stall, meaning that the device needs outside
; intervention and trying again later won't help until it's been serviced.
; enter with endpoint # to stall in Wreg.
; ******************************************************************
StallUSBEP
    bsf     STATUS,IRP      ; select banks 2/3
    andlw   0x03            ; try to keep things under control
    addlw   low UEP0        ; add address of endpoint control reg
    movwf   FSR
    bsf     INDF,EP_STALL   ; set stall bit
    return

; ******************************************************************
; Unstall Endpoint.
; Sets the stall bit in the Endpoint Control Register.  For the control
; Endpoint, this implements a Protocol stall and is used when the request
; is invalid for the current device state.  For non-control Endpoints,
; this is a Functional Stall, meaning that the device needs outside
; intervention and trying again later won't help until it's been serviced.
; enter with endpoint # to stall in Wreg.
; ******************************************************************
UnstallUSBEP
    bsf     STATUS,IRP      ; select banks 2/3
    andlw   0x03            ; try to keep things under control
    addlw   low UEP0        ; add address of endpoint control reg
    movwf   FSR
    bcf     INDF,EP_STALL   ; clear stall bit
    return

; ******************************************************************
```

```
; CheckSleep
; Checks the USB Sleep bit.  If the bit is set, the
; Endpoint, this implements a Protocol stall and is used when the request
; is invalid for the current device state.  For non-control Endpoints,
; this is a Functional Stall, meaning that the device needs outside
; intervention and trying again later won't help until it's been serviced.
; enter with endpoint # to stall in Wreg.
; ********************************************************************
CheckSleep
    global  CheckSleep

    banksel IS_IDLE
    btfss   IS_IDLE,0       ; test the bus idle bit
    return

#ifdef SHOW_ENUM_STATUS
    banksel PORTB
    bsf     PORTB,4         ; turn on LED 4 to indicate we've gone to sleep
    banksel UIR
#endif
    bsf     STATUS,RP0      ; point to bank 3
    bcf     UIR,ACTIVITY
    bsf     UIE,ACTIVITY    ; enable the USB activity interrupt
    bsf     UCTRL,SUSPND    ; put USB regulator and transciever in low power state
    sleep                   ; and go to sleep
    nop
    bcf     UCTRL,SUSPND
    bcf     UIR,UIDLE
    bsf     UIE,UIDLE
    bcf     UIR,ACTIVITY
    bcf     UIE,ACTIVITY
#ifdef SHOW_ENUM_STATUS
    banksel PORTB
    bcf     PORTB,4         ; turn off LED 4 to indicate we're back.
#endif

; ********************************************************************
; Remote Wakeup
; Checks USB_status_device to see if the host enabled Remote Wakeup
; If so, perform Remote wakeup and disable remote wakeup feature
; It is called by PortBChange.
; ********************************************************************
RemoteWakeup
    global  RemoteWakeup

    banksel USB_status_device  ; BANK 2
    btfss   USB_status_device, 1
    return

    bsf     STATUS, RP0     ; BANK 3
       bcf            UCTRL, SUSPND
    bsf     UIE,UIDLE
    bcf     UIR,UIDLE
    bcf     UIE,ACTIVITY
    bcf     UIR,ACTIVITY
    bsf     UCTRL, 2        ; RESUME SIGNALING
       bcf            STATUS, RP0           ; BANK 2

       clrf   inner
       movlw  0x80
       movwf  outer
       pagesel RemoteLoop
RemoteLoop
       decfsz inner, f
       goto   RemoteLoop
       decfsz outer, f
       goto   RemoteLoop

    bsf     STATUS, RP0     ; BANK 3
```

```
    bcf     UCTRL, 2        ; Clear Resume bit
        return


; *****************************************************************
; USB Soft Detach
; Clears the DEV_ATT bit, electrically disconnecting the device to the bus.
; This removes the device from the bus, then reconnects so it can be
; re-enumerated by the host.  This is envisioned as a last ditch effort
; by the software.
; *****************************************************************
SoftDetachUSB
    global  SoftDetachUSB

    banksel UCTRL
    bcf     UCTRL,DEV_ATT   ; clear attach bit

    bcf     STATUS, RP0     ; bank 2

    clrf    outer
    clrf    inner
        pagesel SoftDetachLoop
SoftDetachLoop
    incfsz  inner,f
    goto    SoftDetachLoop
    incfsz  outer,f
    goto    SoftDetachLoop

    pagesel InitUSB
    call    InitUSB         ; reinitialize the USB peripheral
    return


; *****************************************************************
; Init USB
; Initializes the USB peripheral, sets up the interrupts
; *****************************************************************
InitUSB
    global  InitUSB

    banksel USWSTAT
    clrf    USWSTAT         ; default to powered state
    movlw   0x01            ; mask all USB interrupts except reset
    movwf   UIE
    clrf    UIR             ; clear all USB Interrupt flags
    movlw   0x08            ; Device attached
    movwf   UCTRL

    bcf     STATUS, RP0     ; bank 2
    clrf    USB_Curr_Config
    movlw   1
    movwf   USB_status_device
    clrf    USB_Interface
    clrf    USB_Interface+1
    clrf    USB_Interface+2
    movlw   0xFF
    movwf   USB_dev_req     ; no device requests in process
#ifdef COUNTERRORS
    clrf    USB_PID_ERR
    clrf    USB_PID_ERR+1
    clrf    USB_CRC5_ERR
    clrf    USB_CRC5_ERR+1
    clrf    USB_CRC16_ERR
    clrf    USB_CRC16_ERR+1
    clrf    USB_DFN8_ERR
    clrf    USB_DFN8_ERR+1
    clrf    USB_BTO_ERR
    clrf    USB_BTO_ERR+1
    clrf    USB_WRT_ERR
```

```
    clrf     USB_WRT_ERR+1
    clrf     USB_OWN_ERR
    clrf     USB_OWN_ERR+1
    clrf     USB_BTS_ERR
    clrf     USB_BTS_ERR+1
#endif

    banksel PIR1            ; bank 0
    bcf      PIR1,USBIF     ; clear the USB flag
    bsf      STATUS,RP0     ; bank 1
    bsf      PIE1,USBIE     ; enable usb interrupt
    bsf      INTCON, 6      ; enable global and peripheral interrupts
    bsf      INTCON, 7
#ifdef SHOW_ENUM_STATUS
    bcf      STATUS,RP0     ; select bank 0
    bsf      PORTB,0        ; set bit zero to indicate Powered status
#endif
    return


; ******************************************************************
; DeInit USB
; Shuts down the USB peripheral, clears the interrupt enable.
; ******************************************************************
DeInitUSB
    global  DeInitUSB

    banksel UCTRL
    bcf      UCTRL,DEV_ATT   ; D+/D- go high Z
    bsf      UCTRL,SUSPND    ; Place USB module in low power mode.

    clrf     USWSTAT         ; set device state to powered.

    bcf      STATUS,RP1      ; select bank 1
    bcf      PIE1,USBIE      ; clear USB interrupt enable
#ifdef SHOW_ENUM_STATUS
    bcf      STATUS,RP0
    movlw    0x01
    movwf    PORTB           ; clear all lights except powered
    bsf      STATUS,RP0
#endif

    return


core    code
; The functions below are the core functions
; ******************************************************************
; USB interrupt triggered, Why?
; Poll the USB interrupt flags to find the cause.
; ******************************************************************
ServiceUSBInt
    global  ServiceUSBInt

    banksel UIR
    movf     UIR,w           ; get the USB interrupt register
    andwf    UIE,w           ; mask off the disabled interrupts
    bcf      STATUS, RP0     ; BANK 2
        pagesel ExitServiceUSBInt
    btfsc    STATUS,Z        ; is there any unmasked interrupts?
    goto     ExitServiceUSBInt ; no, bail out.

    movwf    USBMaskedInterrupts
    pagesel TokenDone
    btfsc    USBMaskedInterrupts,TOK_DNE    ; was it a token done?
    call     TokenDone
    pagesel USBReset
    btfsc    USBMaskedInterrupts,USB_RST
    call     USBReset
```

```
    pagesel USBStall
    btfsc   USBMaskedInterrupts,STALL
    call    USBStall
    pagesel USBError
    btfsc   USBMaskedInterrupts,UERR
    call    USBError
    pagesel USBSleep
    btfsc   USBMaskedInterrupts,UIDLE
    call    USBSleep
    pagesel USBActivity
    btfsc   USBMaskedInterrupts,ACTIVITY
    call    USBActivity
    pagesel ServiceUSBInt
    goto    ServiceUSBInt
ExitServiceUSBInt
    banksel PIR1
    bcf     PIR1,USBIF
    return

; ******************************************************************
; USB Reset interrupt triggered (SE0)
; initialize the Buffer Descriptor Table,
; Transition to the DEFAULT state,
; Set address to 0
; enable the USB
; ******************************************************************
USBReset    ; START IN BANK2

    clrf    USB_Curr_Config
    clrf    IS_IDLE
    bsf     STATUS, RP0     ; bank 3

    bcf     UIR,TOK_DNE     ; hit this 4 times to clear out the
    bcf     UIR,TOK_DNE     ; USTAT FIFO
    bcf     UIR,TOK_DNE
    bcf     UIR,TOK_DNE

    movlw   0x8
    movwf   BD0OBC
    movlw   USB_Buffer      ; Endpoint 0 OUT gets a buffer
    movwf   BD0OAL          ; set up buffer address
    movlw   0x88            ; set owns bit (SIE can write)
    movwf   BD0OST

    movlw   USB_Buffer+8    ; Endpoint 0 IN gets a buffer
    movwf   BD0IAL          ; set up buffer address
    movlw   0x08            ; Clear owns bit (PIC can write)
    movwf   BD0IST

    clrf    UADDR           ; set USB Address to 0
    clrf    UIR             ; clear all the USB interrupt flags
    banksel PIR1            ; switch to bank 0
    bcf     PIR1,USBIF

; Set up the Endpoint Control Registers.  The following patterns are defined
; ENDPT_DISABLED - endpoint not used
; ENDPT_IN_ONLY  - endpoint supports IN transactions only
; ENDPT_OUT_ONLY - endpoint supports OUT transactions only
; ENDPT_CONTROL     - Supports IN, OUT and CONTROL transactions - Only use with EP0
; ENDPT_NON_CONTROL - Supports both IN and OUT transactions

    banksel UEP0
    movlw   ENDPT_CONTROL
    movwf   UEP0            ; endpoint 0 is a control pipe and requires an ACK

    movlw   0x3B            ; enable all interrupts except activity
    movwf   UIE

    movlw   0xFF            ; enable all error interrupts
```

```
    movwf   UEIE

    movlw   DEFAULT_STATE
    movwf   USWSTAT

    bcf     STATUS,RP0       ; select bank 2
    movlw   0x01
    movwf   USB_status_device ; Self powered, remote wakeup disabled
    bcf     STATUS,RP1       ; bank 0
#ifdef SHOW_ENUM_STATUS
    bsf     PORTB,1          ; set bit one to indicate Reset status
#endif
    bsf     STATUS,RP1
    return                   ; to keep straight with host controller tests

; ******************************************************************
; Enable Wakeup on interupt and Activity interrupt then put the
; device to sleep to save power.  Activity on the D+/D- lines will
; set the ACTIVITY interrupt, waking up the part.
; ******************************************************************
USBSleep    ; starts from bank2
    bsf     STATUS, RP0      ; up to bank 3
    bcf     UIE,UIDLE
    bcf     UIR,UIDLE
    bcf     UIR,ACTIVITY
    bsf     UIE,ACTIVITY
        bsf           UCTRL, SUSPND
    banksel PIR1             ; switch to bank 0
    bcf     PIR1,USBIF

    bsf        STATUS, RP1      ; switch to bank 2
    bsf     IS_IDLE, 0

    return

; ******************************************************************
; This is activated by the STALL bit in the UIR register.  It really
; just tells us that the SIE sent a STALL handshake.  So far, Don't
; see that any action is required.  Clear the bit and move on.
; ******************************************************************
USBStall    ; starts in bank 2
    bsf     STATUS, RP0      ; bank 3
    bcf     UIR, STALL       ; clear STALL

    banksel PIR1             ; switch to bank 0
    bcf     PIR1,USBIF
    bsf     STATUS,RP1       ; bank 2
    return


; ******************************************************************
; The SIE detected an error.  This code increments the appropriate
; error counter and clears the flag.
; ******************************************************************
USBError    ; starts in bank 2
    bsf     STATUS, RP0      ; bank 3
    bcf     UIR,UERR
    banksel PIR1             ; switch to bank 0
    bcf     PIR1,USBIF       ; clear the USB interrupt flag.
    bsf     STATUS,RP1       ; switch to bank 2

#ifdef COUNTERRORS
    banksel UEIR
    movf    UEIR,w           ; get the error register
    andwf   UEIE,w           ; mask with the enables
    clrf    UEIR
    bcf     STATUS, RP0      ; Bank 2
    movwf   USBMaskedErrors ; save the masked errors
```

```
    btfss   USBMaskedErrors,PID_ERR
    goto    CRC5Error
    INCREMENT16 USB_PID_ERR
CRC5Error
    btfss   USBMaskedErrors,CRC5
    goto    CRC16Error
    INCREMENT16 USB_CRC5_ERR
CRC16Error
    btfss   USBMaskedErrors,CRC16
    goto    DFN8Error
    INCREMENT16 USB_CRC16_ERR
DFN8Error
    btfss   USBMaskedErrors,DFN8
    goto    BTOError
    INCREMENT16 USB_DFN8_ERR
BTOError
    btfss   USBMaskedErrors,BTO_ERR
    goto    WRTError
    INCREMENT16 USB_BTO_ERR
WRTError
    btfss   USBMaskedErrors,WRT_ERR
    goto    OWNError
    INCREMENT16 USB_WRT_ERR
OWNError
    btfss   USBMaskedErrors,OWN_ERR
    goto    BTSError
    INCREMENT16 USB_OWN_ERR
BTSError
    btfss   USBMaskedErrors,BTS_ERR
    goto    EndError
    INCREMENT16 USB_BTS_ERR
EndError
#endif
    banksel USBMaskedInterrupts
    return

; ******************************************************************
; Service the Activity Interrupt.  This is only enabled when the
; device is put to sleep as a result of inactivity on the bus.  This
; code wakes up the part, disables the activity interrupt and reenables
; the idle interrupt.
; ******************************************************************
USBActivity     ; starts in bank 2
    bsf     STATUS, RP0     ; Bank 3
    bcf     UIE,ACTIVITY    ; clear the Activity and Idle bits
    bcf     UIR,ACTIVITY
    bcf     UIR,UIDLE
    bsf     UIE,UIDLE
       bcf            UCTRL, SUSPND

    banksel PIR1            ; switch to bank 0
    bcf     PIR1,USBIF      ; clear the USB interrupt flag.
    bsf     STATUS,RP1      ; switch to bank 2

    clrf    IS_IDLE

    return

; ******************************************************************
; Process token done interrupt...  Most of the work gets done through
; this interrupt.  Token Done is signaled in response to an In, Out,
; or Setup transaction.
; ******************************************************************
TokenDone       ; starts in bank 2
    COPYBUFFERDESCRIPTOR    ; copy BD from dual port to unbanked RAM
    banksel USTAT
    movf    USTAT,w         ; copy USTAT register before...
    bcf     UIR,TOK_DNE     ; clearing the token done interrupt.
```

```
    banksel PIR1            ; switch to bank 0
    bcf     PIR1,USBIF      ; clear the USB interrupt flag.
    bsf     STATUS,RP1      ; switch to bank 2

    movwf   USB_USTAT       ; Save USTAT in bank 2

#ifdef SHOW_ENUM_STATUS
; This toggles the activity bits on portB  (EP0 -> Bit 5; EP1 -> bit 6; EP2 -> bit 7)
    bcf     STATUS,RP1      ; bank 0
    andlw   0x18            ; save endpoint bits
       pagesel tryEP1activity
    btfss   STATUS,Z        ; is it EP0?
    goto    tryEP1activity
    movlw   0x20
       pagesel maskport
    goto    maskport
tryEP1activity
    xorlw   0x08            ; is it bit one?
    btfss   STATUS,Z
    movlw   0x80            ; No, It's not EP0, nor 1 so it must be EP2.  toggle bit 7
    btfsc   STATUS,Z
    movlw   0x40            ; Yes, toggle bit 6 to Show EP1 activity
maskport
    xorwf   PORTB,f
    bsf     STATUS,RP1      ; bank 2
#endif

; check UOWN bit here if desired
    movf    BufferDescriptor,w    ; get the first byte of the BD
    andlw   0x3c            ; save the PIDs
    movwf   PIDs

    xorlw   TOKEN_IN
       pagesel TokenInPID
    btfsc   STATUS,Z
    goto    TokenInPID

    movf    PIDs,w
    xorlw   TOKEN_OUT
       pagesel TokenOutPID
    btfsc   STATUS,Z
    goto    TokenOutPID

    movf    PIDs,w
    xorlw   TOKEN_SETUP
       pagesel TokenSetupPID
    btfsc   STATUS,Z
    goto    TokenSetupPID

    return                  ; should never get here...

; ******************************************************************
; Process out tokens
; For EP0, just turn the buffer around.  There should be no EP0
; tokens to deal with.
; EP1 and EP2 have live data destined for the application
; ******************************************************************
TokenOutPID    ; STARTS IN BANK2
    movf    USB_USTAT,w     ; get the status register
       pagesel tryEP1
    btfss   STATUS,Z        ; was it EP0?
    goto    tryEP1          ; no, try EP1

    movf    USB_dev_req,w
    xorlw   HID_SET_REPORT
       pagesel ResetEP0OutBuffer
    btfss   STATUS,Z
    goto    ResetEP0OutBuffer
```

```
HIDSetReport

; ****************************************************************
; You must write your own SET_REPORT routine.  The following
; commented out code is provided if you desire to make a SET_REPORT
; look like a EP1 OUT Interrupt transfer.  Uncomment it and use it
; if you desire this functionality.
; ****************************************************************
;    movlw   0xFF
;    movwf   USB_dev_req     ; clear the request type
;    banksel BD1IST
;    movf    BD0OST,w
;    movwf   BD1OST          ; Copy status register to EP1 Out
;    movf    BD0OAL,w        ; get EP0 Out buffer address
;    bcf     STATUS,RP0      ; bank 2
;    movwf   hid_source_ptr
;    bsf     STATUS,RP0      ; bank 3
;    movf    BD1OAL,w        ; get EP1 Out Buffer Address
;    bcf     STATUS,RP0      ; bank 2
;    movwf   hid_dest_ptr
;    bsf     STATUS,RP0      ; bank 3
;    movf    BD0OBC,w        ; Get byte count
;    movwf   BD1OBC          ; copy to EP1 Byte count
;    bcf     STATUS,RP0      ; bank 2
;    movwf   counter
;    bankisel BD1IST         ; indirectly to bank 3
;HIDSRCopyLoop
;    movf    hid_source_ptr,w
;    movwf   FSR
;    movf    INDF,w
;    movwf   temp
;    movf    hid_dest_ptr,w
;    movwf   FSR
;    movf    temp,w
;    movwf   INDF
;    incf    hid_source_ptr,f
;    incf    hid_dest_ptr,f
;    decfsz  counter,f
;    goto    HIDSRCopyLoop
;
;    bsf     STATUS,RP0      ; bank 3
;    movlw   0x08
;    movwf   BD0OST          ; REset EP0 Status back to SIE

ResetEP0OutBuffer
    bsf     STATUS,RP0      ; no, just reset buffer and move on.

    movlw   0x08            ; it's EP0.. buffer already copied,
    movwf   BD0OBC          ; just reset the buffer
    movlw   0x88
    movwf   BD0OST          ; set OWN and DTS Bit
        pagesel Send_0Len_pkt
    bcf     STATUS,RP0      ; bank 2
    goto    Send_0Len_pkt
    return

tryEP1  ; bank 3
    xorlw   0x08            ; was it EP1?
        pagesel tryEP2
    btfss   STATUS,Z
    goto    tryEP2

; **** Add Callout here to service EP1 in transactions.  ****

    return

tryEP2  ; bank 3
    movf    USB_USTAT,w
    xorlw   0x10            ; was it EP2?
```

```
    btfsc   STATUS,Z
    return                  ; unrecognized EP (Should never take this exit)

; **** Add Callout here to service EP2 in transactions.  ****
    return

; ******************************************************************
; Process in tokens
; ******************************************************************
TokenInPID      ; starts in bank2
; Assumes EP0 vars are setup in a previous call to setup.
EP0_in
    movf    USB_USTAT,w     ; get the status register
    andlw   0x18            ; save only EP bits (we already know it's an IN)
       pagesel tryEP1in
    btfss   STATUS,Z        ; was it EP0?
    goto    tryEP1in        ; no, try EP1

    movf    USB_dev_req,w
    xorlw   GET_DESCRIPTOR
       pagesel check_GSD
    btfss   STATUS,Z
    goto    check_GSD
    pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0
    goto    exitEP0in

; Check for Get String Descriptor
check_GSD
    movf    USB_dev_req,w
    xorlw   GET_STRING_DESCRIPTOR
       pagesel check_SA
    btfss   STATUS,Z
    goto    check_SA
       pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0
       pagesel exitEP0in
    goto    exitEP0in

; Check for Set Address
check_SA
    movf    USB_dev_req,w
    xorlw   SET_ADDRESS
       pagesel check_SF
    btfss   STATUS,Z
    goto    check_SF
       pagesel finish_set_address
    call    finish_set_address
       pagesel exitEP0in
    goto    exitEP0in

check_SF
    movf    USB_dev_req,w
    xorlw   SET_FEATURE
       pagesel check_CF
    btfss   STATUS,Z
    goto    check_CF
       pagesel exitEP0in
    goto    exitEP0in

check_CF
    movf    USB_dev_req,w
    xorlw   CLEAR_FEATURE
       pagesel Class_Specific
    btfss   STATUS,Z
    goto    Class_Specific
       movf    BufferData+4, w       ; clear endpoint 1 stall bit
       xorlw   1
       pagesel clear_EP2
```

```
        btfss   STATUS,Z
        goto    clear_EP2
        bsf             STATUS, RP0           ; bank 3
        bsf             UEP1, EP_STALL
    bcf         STATUS, RP0           ; bank 2
        pagesel exitEP0in
    goto    exitEP0in
clear_EP2
        movf    BufferData+wIndex, w  ; clear endpoint 2 stall bit
        xorlw   2
        pagesel exitEP0in
        btfss   STATUS,Z
        goto    exitEP0in
        bsf             STATUS, RP0           ; bank 3
        bsf             UEP2, EP_STALL
    bcf         STATUS, RP0           ; bank 2
        pagesel exitEP0in
    goto    exitEP0in

Class_Specific
    pagesel Check_Class_Specific_IN
    goto    Check_Class_Specific_IN

exitEP0in
    return


; ******************************************************************
; though not required, it might be nice to have a callback function here
; that would take some action like setting up the next buffer when the
; previous one is complete.  Not necessary because the same functionality
; can be provided through the PutUSB call.
; ******************************************************************
tryEP1in        ; starts in bank 2
    xorlw   0x08            ; was it EP1?
        pagesel tryEP1in
    btfss   STATUS,Z
    goto    tryEP2in
; **** Add Callout here to service EP1 in transactions.  ****
    return

tryEP2in        ; starts in bank 2
; **** Add Callout here to service EP2 in transactions.  ****
    return
; ******************************************************************
; Return a zero length packet on EP0 In
; ******************************************************************
Send_0Len_pkt
    global  Send_0Len_pkt

    banksel BD0IBC
    clrf    BD0IBC          ; set byte count to 0
    movlw   0xc8
    movwf   BD0IST          ; set owns bit
    bcf     STATUS,RP0      ; back to bank 2
    clrf    USB_dev_req
    return

; *******************************************************************
; process setup tokens
; ******************************************************************
TokenSetupPID    ; starts in bank 2
    bsf     STATUS,IRP      ; indirectly to pages 2/3
    movf    BufferDescriptor+ADDRESS,w ; get the status register
    movwf   FSR             ; save in the FSR.
    movf    INDF,w
    movwf   BufferData      ; in shared RAM
    incf    FSR,f
    movf    INDF,w
```

```
movwf   BufferData+1
incf    FSR,f
movf    INDF,w
movwf   BufferData+2
incf    FSR,f
movf    INDF,w
movwf   BufferData+3
incf    FSR,f
movf    INDF,w
movwf   BufferData+4
incf    FSR,f
movf    INDF,w
movwf   BufferData+5
incf    FSR,f
movf    INDF,w
movwf   BufferData+6
incf    FSR,f
movf    INDF,w
movwf   BufferData+7
bsf     STATUS, RP0             ; bank 3
movlw   0x08
movwf   BD0OBC          ; reset the byte count too.
movwf   BD0IST          ; return the in buffer to us (dequeue any pending requests)
   bcf          STATUS, RP0           ; bank 2
   movf   BufferData+bmRequestType, w
   xorlw  HID_SET_REPORT ; set EP0 OUT UOWNs back to SIE
movlw   0x88                    ; set DATA0/DATA1 packet according to request type
   btfsc  STATUS, Z
   movlw  0xC8
   bsf          STATUS, RP0           ; bank 3
movwf   BD0OST

bcf     UCTRL,PKT_DIS    ; Assuming there is nothing to dequeue, clear the packet disable
                          ; bit

bcf     STATUS,RP0      ; bank 2
clrf    USB_dev_req     ; clear the device request..

movf    BufferData+bmRequestType,w
   pagesel HostToDevice
btfsc   STATUS,Z
goto    HostToDevice

movf    BufferData+bmRequestType,w
xorlw   0x01            ; test for host to Interface tokens
   pagesel HostToInterface
btfsc   STATUS,Z
goto    HostToInterface

movf    BufferData+bmRequestType,w
xorlw   0x02            ; test for host to Endpoint tokens
   pagesel HostToEndpoint
btfsc   STATUS,Z
goto    HostToEndpoint

movf    BufferData+bmRequestType,w
xorlw   0x80            ; test for device to Host tokens
   pagesel DeviceToHost
btfsc   STATUS,Z
goto    DeviceToHost

movf    BufferData+bmRequestType,w
xorlw   0x81            ; test for device to Interface tokens
   pagesel InterfaceToHost
btfsc   STATUS,Z
goto    InterfaceToHost

movf    BufferData+bmRequestType,w
xorlw   0x82            ; test for device to Endpoint tokens
```

```
        pagesel EndpointToHost
    btfsc   STATUS,Z
    goto    EndpointToHost

    movf    BufferData+bmRequestType,w
    andlw   0x60            ; mask off type bits
    xorlw   0x20            ; test for class specific
    pagesel ClassSpecificRequest
    btfsc   STATUS,Z        ; was it a standard request?
    goto    ClassSpecificRequest    ; nope, see if it was a class specific request

CheckForVendorRequest
    movf    BufferData+bmRequestType,w
    andlw   0x60            ; mask off type bits
    xorlw   0x40            ; test for vendor specific
    pagesel wrongstate
    btfss   STATUS,Z        ; was it a standard request?
    goto    wrongstate
    pagesel CheckVendor
    goto    CheckVendor     ; nope, see if it was a vendor specific
    return

; now test bRequest to see what the request was.

CheckForStandardRequest
; bmRequestType told us it was a Host to Device transfer.  Now look at
; the specifics to see what's up
HostToDevice    ; starts in bank 2
    movf    BufferData+bRequest,w   ; what was our request
    xorlw   CLEAR_FEATURE
        pagesel Clear_Device_Feature
    btfsc   STATUS,Z
    goto    Clear_Device_Feature

    movf    BufferData+bRequest,w   ; was our request Set Address
    xorlw   SET_ADDRESS
        pagesel Set_Address
    btfsc   STATUS,Z
    goto    Set_Address

    movf    BufferData+bRequest,w   ; was our request Set Configuration
    xorlw   SET_CONFIGURATION
        pagesel Set_Configuration
    btfsc   STATUS,Z
    goto    Set_Configuration

    movf    BufferData+bRequest,w   ; was our request Set Feature
    xorlw   SET_FEATURE
        pagesel Set_Device_Feature
    btfsc   STATUS,Z
    goto    Set_Device_Feature

        pagesel wrongstate
    goto    wrongstate

HostToInterface ; starts in bank 2
    movf    BufferData+bRequest,w   ; what was our request
    xorlw   CLEAR_FEATURE
        pagesel Clear_Interface_Feature
    btfsc   STATUS,Z
    goto    Clear_Interface_Feature

    movf    BufferData+bRequest,w   ; was our request Set Interface
    xorlw   SET_INTERFACE
        pagesel Set_Interface
    btfsc   STATUS,Z
    goto    Set_Interface

    movf    BufferData+bRequest,w   ; was our request Set Feature
```

```
    xorlw   SET_FEATURE
       pagesel Set_Interface_Feature
    btfsc   STATUS,Z
    goto    Set_Interface_Feature

       pagesel wrongstate
       goto    wrongstate

HostToEndpoint  ; starts in bank2
    movf    BufferData+bRequest,w   ; what was our request
    xorlw   CLEAR_FEATURE
       pagesel Clear_Endpoint_Feature
    btfsc   STATUS,Z
    goto    Clear_Endpoint_Feature

    movf    BufferData+bRequest,w   ; was our request Set Feature
    xorlw   SET_FEATURE
       pagesel Set_Endpoint_Feature
    btfsc   STATUS,Z
    goto    Set_Endpoint_Feature

DeviceToHost    ; starts in bank2
    movf    BufferData+bRequest,w   ; what was our request
    xorlw   GET_CONFIGURATION
       pagesel Get_Configuration
    btfsc   STATUS,Z
    goto    Get_Configuration

    movf    BufferData+bRequest,w   ; was our request Get Decriptor?
    xorlw   GET_DESCRIPTOR
       pagesel Get_Descriptor
    btfsc   STATUS,Z
    goto    Get_Descriptor

    movf    BufferData+bRequest,w   ; was our request Get Status?
    xorlw   GET_STATUS
       pagesel Get_Device_Status
    btfsc   STATUS,Z
    goto    Get_Device_Status

InterfaceToHost ; starts in bank2
    movf    BufferData+bRequest,w   ; was our request Get Interface?
    xorlw   GET_INTERFACE
       pagesel Get_Interface
    btfsc   STATUS,Z
    goto    Get_Interface

    movf    BufferData+bRequest,w   ; was our request Get Status?
    xorlw   GET_STATUS
       pagesel Get_Interface_Status
    btfsc   STATUS,Z
    goto    Get_Interface_Status

    movf    BufferData+bRequest,w   ; was our request Get Decriptor?
    xorlw   GET_DESCRIPTOR
       pagesel Get_Descriptor
    btfsc   STATUS,Z
    goto    Get_Descriptor

EndpointToHost  ; starts in bank2
    movf    BufferData+bRequest,w   ; was our request Get Status?
    xorlw   GET_STATUS
       pagesel Get_Endpoint_Status
    btfsc   STATUS,Z
    goto    Get_Endpoint_Status

    pagesel wrongstate              ; unrecognised token, stall EP0
    goto    wrongstate
```

```
    return

; ****************************************************************
; Get Descriptor
; Handles the three different Get Descriptor commands
; ****************************************************************
Get_Descriptor  ; starts in bank2
    movf    BufferData+(wValue+1),w ; request, which seems to be undefined,
    xorlw   0x22                    ; but it won't enumerate without it
       pagesel Get_Report_Descriptor
    btfsc   STATUS,Z
    goto    Get_Report_Descriptor

    movf    BufferData+(wValue+1),w
    xorlw   0x21
       pagesel Get_HID_Descriptor
    btfsc   STATUS,Z
    goto    Get_HID_Descriptor

GetCh9Descriptor
    movlw   high StartGDIndex       ; set up PCLATH with the current address
    movwf   PCLATH                  ; set up pclath for the computed goto
       bcf          STATUS, C
       movf    BufferData+(wValue+1),w       ; move descriptor type into w
       andlw   0x03                                 ; keep things under control
    addlw   low  StartGDIndex
    btfsc   STATUS,C                ; was there an overflow?
    incf    PCLATH,f                ; yes, bump PCLATH
    movwf   PCL                     ; adjust PC
StartGDIndex
    goto    wrongstate              ; 0
    goto    Get_Device_Descriptor   ; 1
    goto    Get_Config_Descriptor    ; 2
    goto    Get_String_Descriptor   ; 3


; ****************************************************************
; Looks up the offset of the device descriptor via the low order byte
; of wValue.  The pointers are set up and the data is copied to the
; buffer, then the flags are set.
;
; EP0_start points to the first word to transfer
; EP0_end points to the last, limited to the least of the message length
;         or the number of bytes requested in the message (wLength).
; EP0_maxLength is the number of bytes to transfer at a time, 8 bytes
; ****************************************************************
Get_Device_Descriptor          ; starts in bank 2
    movlw   GET_DESCRIPTOR
    movwf   USB_dev_req         ; currently processing a get descriptor request

    movlw   8
    movwf   EP0_maxLength

    movlw   low DeviceDescriptor
    movwf   EP0_start
    movlw   high DeviceDescriptor
    movwf   EP0_start+1
       pagesel Descriptions
    call    Descriptions        ; get length of device descriptor
    movwf   EP0_end             ; save length

    movf    BufferData+(wLength+1),f ; move it to itself, check for non zero.
    pagesel DeviceEndPtr
    btfss   STATUS,Z            ; if zero, we need to compare EP0_end to requested length.
    goto    DeviceEndPtr        ; if not, no need to compare.  EP0_end is shorter than
request length

    subwf   BufferData+wLength,w    ; compare against requested length
    movf    BufferData+wLength,w
```

```
    btfss   STATUS,C
    movwf   EP0_end

DeviceEndPtr
    incf    EP0_end,f
    pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0

    return


; *******************************************************************
; Looks up the offset of the config descriptor via the low order byte
; of wValue.  The pointers are set up and the data is copied to the
; buffer, then the flags are set.
;
; EP0_start points to the first word to transfer
; EP0_end points to the last, limited to the least of the message length
;         or the number of bytes requested in the message (wLength).
; EP0_maxLength is the number of bytes to transfer at a time, 8 bytes
; *****************************************************************
Get_Config_Descriptor    ; starts in bank2
    movlw   GET_DESCRIPTOR
    movwf   USB_dev_req         ; currently processing a get descriptor request

    bcf     STATUS,C
    rlf     BufferData+wValue,w
       pagesel Config_desc_index
    call    Config_desc_index   ; translate index to offset into descriptor table
    movwf   EP0_start
    bcf     STATUS,C
    rlf     BufferData+wValue,w
    addlw   1                   ; point to high order byte
    call    Config_desc_index   ; translate index to offset into descriptor table
    movwf   EP0_start+1

    movlw   2                   ; bump pointer by 2 to get the complete descriptor
    addwf   EP0_start,f         ; length, not just config descriptor
    btfsc   STATUS,C
    incf    EP0_start+1,f
       pagesel Descriptions
    call    Descriptions        ; get length of the config descriptor
    movwf   EP0_end             ; Get message length

    movlw   2                   ; move EP0_start pointer back to beginning
    subwf   EP0_start,f
    btfss   STATUS,C
    decf    EP0_start+1,f

    movf    BufferData+(wLength+1),f ; test for 0
    pagesel CmpLowerByte
    btfsc   STATUS,Z
    goto    CmpLowerByte
       pagesel ConfigEndPtr
    goto    ConfigEndPtr        ; if not, no need to compare.  EP0_end is shorter than
request length

CmpLowerByte
    movf    EP0_end,w
    subwf   BufferData+wLength,w    ; compare against requested length
       pagesel ConfigEndPtr
    btfsc   STATUS,C
    goto    ConfigEndPtr
LimitSize
    movf    BufferData+wLength,w    ; if requested length is shorter..
    movwf   EP0_end                 ; save it.
ConfigEndPtr

    movlw   8
```

```
    movwf   EP0_maxLength
    incf    EP0_end,f

    pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0
    return

; ****************************************************************
; Set up to return String descriptors
; Looks up the offset of the string descriptor via the low order byte
; of wValue.  The pointers are set up and the data is copied to the
; buffer, then the flags are set.
; ****************************************************************
Get_String_Descriptor    ; starts in bank2
    movlw   GET_STRING_DESCRIPTOR
    movwf   USB_dev_req         ; currently processing a get descriptor request

    movf    BufferData+wIndex,w
       pagesel not_string0
    btfss   STATUS,Z
    goto    not_string0
    movf    BufferData+(wIndex+1),w
    btfss   STATUS,Z
    goto    not_string0
    movlw   low String0
    movwf   EP0_start
    movlw   high String0
    movwf   EP0_start+1
       pagesel found_string
    goto    found_string

not_string0
    movlw   high (String0+2)
    movwf   EP0_start+1
    movlw   low (String0+2)
    movwf   EP0_start
    clrf    inner

check_langid
    pagesel StringDescriptions
    call    StringDescriptions
    incf    EP0_start,f
    subwf   BufferData+wIndex, w
    pagesel wrong_langid
    btfss   STATUS, Z
    goto    wrong_langid
    pagesel StringDescriptions
    call    StringDescriptions
    subwf   BufferData+(wIndex+1), w
    pagesel right_langid
    btfsc   STATUS, Z
    goto    right_langid

wrong_langid
    incf    EP0_start,f
    incf    inner,f
    movlw   low String0_end     ; compare EP0_start to the addr of
    subwf   EP0_start,w         ; the last langid
       pagesel check_langid
    btfss   STATUS,C            ; if EP0_start is equal or lager,
    goto    check_langid        ; we've checked all langid and didn't find it
    clrf    USB_dev_req         ; clear USB_dev_req, since GET_descriptor is over
    pagesel wrongstate
    goto    wrongstate

right_langid
    movlw   6                   ; number of strings we have per language + 1
    subwf   BufferData+wValue,w
       pagesel right_string
```

```
    btfss   STATUS,C
    goto    right_string
    clrf    USB_dev_req
    pagesel wrongstate
    goto    wrongstate

right_string
    rlf     BufferData+wValue,w
    movwf   EP0_start+1
    movf    inner,w
    pagesel string_index
    call    string_index
    movwf   EP0_start
    incf    EP0_start+1,f
    movf    inner,w
    call    string_index
    movwf   EP0_start+1

found_string
    pagesel StringDescriptions
    call    StringDescriptions  ; get length of the string descriptor
    movwf   EP0_end             ; save length

    subwf   BufferData+wLength,w    ; compare against requested length
    movf    BufferData+wLength,w    ; if requested length is shorter..
    btfss   STATUS,C
    movwf   EP0_end             ; save it.

    movlw   8                   ; each transfer may be 8 bytes long
    movwf   EP0_maxLength

    incf    EP0_end,f
    pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0
    return

; ******************************************************************
; Stalls the EP0 endpoint to signal that the command was not recognised.
; This gets reset as the result of a Setup Transaction.
; ******************************************************************
wrongstate
    global  wrongstate

    banksel UEP0
    bsf     UEP0,EP_STALL
    bcf     STATUS,RP0      ; back to page 2

    return

; ******************************************************************
; Loads the device status byte into the EP0 In Buffer.
; ******************************************************************
Get_Device_Status           ; starts in bank2
    bsf     STATUS,RP0
    movf    BD0IAL,w        ; get buffer pointer
    movwf   FSR
    bcf     STATUS,RP0      ; bank 2
    bsf     STATUS,IRP      ; select indirectly banks 2-3
    movf    USB_status_device,w ; get device status byte
    movwf   INDF
    incf    FSR,f
    clrf    INDF

    bsf     STATUS,RP0      ; bank 3
    movlw   0x02
    movwf   BD0IBC          ; set byte count to 2
    movlw   0xC8
    movwf   BD0IST          ; Data 1 packet, set owns bit
    return
```

```
; ****************************************************************
; A do nothing response.  Always returns a two byte record, with all
; bits zero.
; ****************************************************************
Get_Interface_Status        ; starts in bank 2
    bsf     STATUS, RP0     ; bank 3
        movf    USWSTAT,w
        xorlw   ADDRESS_STATE
        pagesel Get_Interface_Status2
        btfss   STATUS, Z
        goto    Get_Interface_Status2

        bcf             STATUS, RP0             ; bank 2
        movf    BufferData+wIndex, w
        pagesel Get_Interface_Status2
        btfss   STATUS, Z
        goto    Get_Interface_Status2

Get_Interface_Status2
        bsf             STATUS, RP0             ; bank3
        movf    USWSTAT,w
        xorlw   CONFIG_STATE
        pagesel wrongstate
        btfss   STATUS, Z
        goto    wrongstate

        bcf             STATUS, RP0
        movf    BufferData+wIndex,w     ; if Interface < NUM_INTERFACES
        sublw   (NUM_INTERFACES-1)
        pagesel wrongstate
        btfss   STATUS, C
        goto    wrongstate

Get_Interface_Status_end
        movf    BufferData+wIndex,w ; get interface ID
    addlw   low USB_Interface
    movwf   FSR
    bsf     STATUS,IRP
    movf    INDF,w
    movwf   temp            ; store in temp register

    bsf     STATUS,RP0      ; bank3
    movf    BD0IAL,w        ; get address of buffer
    movwf   FSR
    movf    temp,w          ; load temp
    movwf   INDF            ; write byte to buffer

    movlw   0x02
    movwf   BD0IBC          ; set byte count to 2
    movlw   0xc8            ; DATA1 packet, DTS enabled
    movwf   BD0IST          ; give buffer back to SIE
    return

; ****************************************************************
; Returns the Endpoint stall bit via a 2 byte in buffer
; ****************************************************************
Get_Endpoint_Status         ; starts in bank 2
    movlw   0x0f
    andwf   BufferData+wIndex,w    ; get endpoint, strip off direction bit
    xorlw   0x01            ; is it EP1?
        pagesel get_EP1_status
    btfsc   STATUS,Z
    goto    get_EP1_status

    movlw   0x0f
    andwf   BufferData+wIndex,w    ; get endpoint, strip off direction bit
    xorlw   0x02            ; is it EP2?
        pagesel wrongstate
```

```
    btfss   STATUS,Z
    goto    wrongstate

get_EP2_status
    bcf     STATUS,C
    bsf     STATUS,RP0
    btfsc   UEP2,EP_STALL
    bsf     STATUS,C
       pagesel build_status_buffer
    goto    build_status_buffer

get_EP1_status
    bcf     STATUS,C
    bsf     STATUS,RP0
    btfsc   UEP1,EP_STALL
    bsf     STATUS,C

build_status_buffer
    movf    BD0IAL,w        ; get address of buffer
    movwf   FSR
    clrf    INDF            ; clear byte 0 in buffer
    rlf     INDF,f          ; rotate in carry bit (EP_stall bit)
    incf    FSR,f           ; bump pointer
    clrf    INDF            ; clear byte

    movlw   0x02
    movwf   BD0IBC          ; set byte count to 2
    movlw   0xC8
    movwf   BD0IST          ; Data 1 packet, set owns bit
    return

; ******************************************************************
; The low order byte of wValue now has the new device address as assigned
; from the host.  Save it in the UADDR, transition to the ADDRESSED state
; and clear the current configuration.
; This assumes the SIE has already sent the status stage of the transaction
; as implied by Figure 3-35 of the DOS (Rev A-7)
; ***************************************************************
Set_Address     ; starts in bank 2
    movf    BufferData+wValue,w    ; new address in low order byte of wValue
    movwf   USB_address_pending
       pagesel wrongstate
       btfsc   USB_address_pending, 7
       goto    wrongstate
       pagesel Send_0Len_pkt
       call    Send_0Len_pkt            ; send zero length packet
    movlw   SET_ADDRESS
    movwf   USB_dev_req            ; currently processing a get descriptor request
    return

finish_set_address          ; starts in bank 2
    clrf    USB_dev_req     ; no request pending
    clrf    USB_Curr_Config ; make sure current configuration is 0
    movf    USB_address_pending,w
    bsf     STATUS, RP0
    movwf   UADDR           ; set the device address
       pagesel endfinishsetaddr
    btfsc   STATUS,Z        ; was address 0?
    goto    endfinishsetaddr    ; yes: don't change state

    movlw   ADDRESS_STATE   ; non-zero: transition to addressed state
    movwf   USWSTAT         ; transition to addressed state
#ifdef SHOW_ENUM_STATUS
    banksel PORTB
    bsf     PORTB,2         ; set bit 2 to indicate Addressed state
    banksel USWSTAT         ; not necessary, Send_0LenPkt resets bank bits
#endif

endfinishsetaddr
```

```
    return

; ******************************************************************
; only feature valid for device feature is Device Remote wakeup
; ******************************************************************
Clear_Device_Feature        ; starts in bank2
    movf    BufferData+wValue,w
    xorlw   0x01            ; was it a Device Remote wakeup? If not, return STALL,
       pagesel wrongstate
    btfss   STATUS,Z        ; since we only implement this feature on this device.
    goto    wrongstate

right_state_clear_feature
    bcf     USB_status_device,1 ; set device remote wakeup
       pagesel Send_0Len_pkt
    call    Send_0Len_pkt
    return

; ******************************************************************
; Only endpoint feature is Endpoint halt.
; ******************************************************************
Clear_Endpoint_Feature  ; starts in bank 2
       movf    BufferData+wValue, w
       pagesel wrongstate
       btfss   STATUS, Z      ; only valid feature is 0 (Remote Wakeup)
       goto    wrongstate
       movf    BufferData+(wValue+1), w
       btfss   STATUS, Z
       goto    wrongstate

       bsf             STATUS, RP0           ; bank3
       movlw   0x03                  ; if ((USWSTAT & 0x03) == ADDRESS_STATE)
       andwf   USWSTAT, w
       xorlw   ADDRESS_STATE
       pagesel clear_endpoint_feature2
       btfss   STATUS, Z
       goto    clear_endpoint_feature2
       bcf             STATUS, RP0           ; bank2
       movlw   0x0F                  ; if ((Bufferdata+wIndex & 0x07) = 0)
       andwf   BufferData+wIndex, w
       btfss   STATUS, Z
       goto    clear_endpoint_feature2
       bsf             STATUS, RP0           ; bank 3
       bcf             UEP0, 0
       pagesel Send_0Len_pkt
       call    Send_0Len_pkt
       return

clear_endpoint_feature2
       bsf             STATUS, RP0
       movlw   0x03                  ; if ((USWSTAT & 0X03) == CONFIG_STATE)
       andwf   USWSTAT, w
       xorlw   CONFIG_STATE
       pagesel wrongstate
       btfss   STATUS, Z
       goto    wrongstate
       bcf             STATUS, RP0           ; bank2
       movlw   0x0F
       andwf   BufferData+wIndex, w  ; if (BufferData+wIndex < 3)
       sublw   2
       pagesel wrongstate
       btfss   STATUS, C
       goto    wrongstate
       bsf             STATUS, IRP
       movlw   0x0F
       andwf   BufferData+wIndex,w
       bsf             STATUS, RP0   ; bank3
       addlw   UEP0&0xFF
       movwf   FSR
```

```
        bcf             INDF, 0
        pagesel Send_0Len_pkt
        call    Send_0Len_pkt
        return

Clear_Interface_Feature     ; starts in bank2
        pagesel wrongstate
        goto    wrongstate

; ******************************************************************
; only feature valid for device feature is Device Remote wakeup
; ******************************************************************
Set_Device_Feature  ; starts in bank 2
    movf    BufferData+wValue,w ; get high order byte of wValue
    xorlw   0x01            ; was it a Device Remote wakeup?
        pagesel wrongstate
    btfss   STATUS,Z
    goto    wrongstate      ; request error
    bsf     USB_status_device,1 ; set device remote wakeup
        pagesel Send_0Len_pkt
    call    Send_0Len_pkt
    return

; ******************************************************************
; Only endpoint feature is Endpoint halt.
; ******************************************************************
Set_Endpoint_Feature            ; starts in bank 2
        movf    BufferData+wValue, w
        pagesel wrongstate
        btfss   STATUS, Z       ; only valid feature is 0 (Remote Wakeup)
        goto    wrongstate
        movf    BufferData+(wValue+1), w
        btfss   STATUS, Z
        goto    wrongstate

        bsf             STATUS, RP0         ; bank3
        movlw   0x03                    ; if ((USWSTAT & 0x03) == ADDRESS_STATE)
        andwf   USWSTAT, w
        xorlw   ADDRESS_STATE
        pagesel set_endpoint_feature2
        btfss   STATUS, Z
        goto    set_endpoint_feature2
        bcf             STATUS, RP0         ; bank2
        movlw   0x0F                    ; if ((Bufferdata+wIndex & 0x07) = 0)
        andwf   BufferData+wIndex, w
        btfss   STATUS, Z
        goto    set_endpoint_feature2
        bsf             STATUS, RP0         ; bank 3
        bsf             UEP0, 0
        pagesel Send_0Len_pkt
        call    Send_0Len_pkt
        return

set_endpoint_feature2
        bsf             STATUS, RP0
        movlw   0x03                    ; if ((USWSTAT & 0X03) == CONFIG_STATE)
        andwf   USWSTAT, w
        xorlw   CONFIG_STATE
        pagesel wrongstate
        btfss   STATUS, Z
        goto    wrongstate
        bcf             STATUS, RP0         ; bank2
        movlw   0x0F
        andwf   BufferData+wIndex, w    ; if (BufferData+wIndex < 3)
        sublw   2
        pagesel wrongstate
        btfss   STATUS, C
        goto    wrongstate
        bsf             STATUS, IRP
```

```
        movlw   0x0F
        andwf   BufferData+wIndex,w
        bsf             STATUS, RP0     ; bank3
        addlw   UEP0&0xFF
        movwf   FSR
        bsf             INDF, 0
        pagesel Send_0Len_pkt
        call    Send_0Len_pkt
        return

Set_Interface_Feature       ; starts in bank 2
        pagesel wrongstate
    goto    wrongstate      ; invalid request

; ****************************************************************
; Get configuration returns a single byte Data1 packet indicating the
; configuration in use.
; Default State    - undefined
; Addressed State  - returns 0
; Configured state - returns current configured state.
; **************************************************************
Get_Configuration           ; starts in bank 2
    bsf     STATUS, RP0
    movf    low BD0IAL,w    ; get address of buffer
    movwf   FSR
    bcf     STATUS, RP0
    bsf     STATUS,IRP      ; indirectly to banks 2-3
    movf    USB_Curr_Config,w
    movwf   INDF            ; write byte to buffer
    bsf     STATUS, RP0
    movlw   0x01
    movwf   BD0IBC          ; set byte count to 1
    movlw   0xc8            ; DATA1 packet. DTS enabled
    movwf   BD0IST          ; give buffer back to SIE
    return

; **************************************************************
; Set configuration uses the configuration selected by the low order
; byte of wValue.  Sets up a zero length data1 packet as a reply.
; **************************************************************
Set_Configuration           ; starts in bank 2
; All we do is set a meaningless number.  This'll
; need more code here to actually give meaning to each configuration
; we choose.
    movf    BufferData+wValue,w ; is it a valid configuration?
    sublw   NUM_CONFIGURATIONS
        pagesel wrongstate
    btfss   STATUS,C        ; if config <= num configs, request appears valid
    goto    wrongstate

    movf    BufferData+wValue,w
    movwf   USB_Curr_Config ; store new state in configuration

        pagesel AckSetConfigCmd
    btfsc   STATUS,Z        ; was the configuration zero?
    goto    AckSetConfigCmd ; yes: stay in the addressed state

    bsf     STATUS, RP0     ; bank 3
    movlw   CONFIG_STATE    ; No: transition to configured
    movwf   USWSTAT         ; save new state.
#ifdef SHOW_ENUM_STATUS
    banksel PORTB
    bsf     PORTB,3         ; set bit 3 to show configured
#endif

AckSetConfigCmd
        pagesel Send_0Len_pkt
    call    Send_0Len_pkt
```

```
; These configure the EP1 and EP2  endpoints.  Change these as necessary
; for your application.
    banksel BD1OAL
    movlw   USB_Buffer+0x10 ; Endpoint 1 OUT gets a buffer
    movwf   BD1OAL          ; set up buffer address
    movlw   8
    movwf   BD1OBC          ; set byte count
    movlw   0x88            ; set own bit of EP1 (SIE can write)
    movwf   BD1OST

        movlw   8
        movwf   BD1IBC                  ; set byte count
    movlw   USB_Buffer+0x18 ; Endpoint 1 IN gets a buffer
    movwf   BD1IAL          ; set up buffer address
    movlw   0x48            ; set own bit of EP1 (PIC can write)
    movwf   BD1IST

    movlw   USB_Buffer+0x20 ; Endpoint 2 OUT gets a buffer
    movwf   BD2OAL          ; set up buffer address
    movlw   8
    movwf   BD2OBC          ; set byte count
    movlw   0x88            ; set own bit of EP2 (SIE can write)
    movwf   BD2OST

        movlw   8
        movwf   BD2IBC                  ; set byte count
    movlw   USB_Buffer+0x20 ; EP1 In and EP2 In share a buffer
    movwf   BD2IAL          ; set up buffer address
    movlw   0x48            ; set own bit of EP2 (PIC can write)
    movwf   BD2IST
; Set up the Endpoint Control Registers.  The following patterns are defined
; ENDPT_DISABLED - endpoint not used
; ENDPT_IN_ONLY  - endpoint supports IN transactions only
; ENDPT_OUT_ONLY - endpoint supports OUT transactions only
; ENDPT_CONTROL     - Supports IN, OUT and CONTROL transactions - Only use with EP0
; ENDPT_NON_CONTROL - Supports both IN and OUT transactions
    movlw   ENDPT_NON_CONTROL
    movwf   UEP1            ; enable EP's 1 and 2 for In and Outs...
    movlw   ENDPT_NON_CONTROL
    movwf   UEP2

;    pagesel SetConfiguration    ; call SetConfiguration etc. after configuration changed
;    movf    USB_Curr_Config,w   ; if you have multiple configurations
;    call    SetConfiguration
;    pagesel Set_Configuration
    return


; ******************************************************************
; Get interface returns a single byte Data1 packet indicating the
; interface in use.
; Default State    - undefined
; Addressed State  - Not valid - returns stall
; Configured state - returns current configured state.
; ******************************************************************
Get_Interface   ; STARTS IN BANK 2
    bsf     STATUS, RP0
    movf    USWSTAT,w       ; Only valid in the configured state
        xorlw   CONFIG_STATE
        pagesel wrongstate
        btfss   STATUS, Z
        goto    wrongstate

        bcf             STATUS, RP0
        movf    BufferData+wIndex,w     ; if Interface < NUM_INTERFACES
        sublw   (NUM_INTERFACES-1)
        pagesel wrongstate
        btfss   STATUS, C
        goto    wrongstate
```

```
    movf    BufferData+wIndex,w ; get interface ID
    addlw   low USB_Interface
    movwf   FSR
    bsf     STATUS,IRP
    movf    INDF,w
    movwf   temp            ; store in temp register

    bsf     STATUS,RP0      ; bank 3
    movf    BD0IAL,w        ; get address of buffer
    movwf   FSR
    movf    temp,w          ; load temp
    movwf   INDF            ; write byte to buffer

    movlw   0x01
    movwf   BD0IBC          ; set byte count to 1
    movlw   0xc8            ; DATA1 packet, DTS enabled
    movwf   BD0IST          ; give buffer back to SIE
    return

; ******************************************************************
; Set configuration uses the configuration selected by the low order
; byte of wValue.  Sets up a zero length data1 packet as a reply.
; ******************************************************************
Set_Interface   ; start bank 2
    bsf     STATUS, RP0             ; bank3
    movf    USWSTAT,w       ; test to make sure we're configured
    bcf     STATUS,RP0      ; bank2
    andlw   0x03
    xorlw   CONFIG_STATE
       pagesel wrongstate
    btfss   STATUS,Z
    goto    wrongstate

    movf    BufferData+wIndex,w ; get interface
    addlw   USB_Interface   ; add offset to array
    movwf   FSR
    bsf     STATUS,IRP      ; indirectly to banks 2-3
    movf    BufferData+wValue,w ; get alternate interface
    movwf   INDF            ; store in array
; All we do is set a meaningless number.  This'll
; need more code here to actually give meaning to each configuration
; we choose.
       pagesel Send_0Len_pkt
    call    Send_0Len_pkt
    return

; *********************************************************************
; copies the next chunk of buffer descriptor over to the EP0 In buffer.
; Inputs:
;    EP0_start - points to first byte of configuration table to transfer
;    EP0_end - total number of bytes to transfer
;    EP0_maxLength - maximum number of bytes that can be sent during
;    a single transfer
;
; toggles the data0/1 bit before setting the UOWN bit over to SIE.
; ******************************************************************
copy_descriptor_to_EP0
    global  copy_descriptor_to_EP0
    banksel BD0IAL
    bankisel BD0IAL
    movf    BD0IAL,w        ; get buffer address
    movwf   FSR
    banksel bufindex
    clrf    bufindex        ; bufindex = 0
gdd_loop
    movf    bufindex,w      ; while (bufindex < EP0_maxLength)
    subwf   EP0_maxLength,w ;     && (EP0_start < EP0_end)
       pagesel end_gdd_loop
```

```
    btfsc   STATUS,Z
    goto    end_gdd_loop

       pagesel gdd_copy_loop
    decfsz  EP0_end, f
    goto    gdd_copy_loop
       pagesel end_gdd_loop_short_packet
    goto    end_gdd_loop_short_packet

gdd_copy_loop
    pagesel Descriptions
    call    Descriptions
    movwf   INDF

    incf    bufindex,f
    incf    FSR,f
       pagesel gdd_loop
    incfsz  EP0_start,f
    goto    gdd_loop
    incf    EP0_start+1,f
    goto    gdd_loop

end_gdd_loop_short_packet
    clrf    USB_dev_req     ; we're sending a short packet, clear the device request
end_gdd_loop
    movf    bufindex,w      ; write number of bytes to byte count
    bsf     STATUS,RP0      ; Bank 3
    movwf   BD0IBC
    movlw   (0x01<<DATA01)  ; toggle data0/1 bit
    xorwf   BD0IST,w
    andlw   (0x01<<DATA01)  ; clear PID bits
    iorlw   0x88            ; set OWN and DTS bits
    movwf   BD0IST          ; write the whole mess back
    pagesel copy_descriptor_to_EP0
    return
; ******************************************************************
; SetConfiguration
;
; This function is called when the host issues a Set Configuration
; command.  The housekeeping within USB is handled within the CH9 commands
; This function should be filled in to give meaning to the command within
; the application.
;
; SetConfiguration is called from within the ISR so this function should
; be kept as short as possible.
; ******************************************************************
SetConfiguration
    global  SetConfiguration
    return

; ******************************************************************
; Vendor Specific calls
; control is transferred here when bmRequestType bits 5 & 6 = 10 indicating
; the request is a vendor specific request.  This function then would
; interpret the bRequest field to determine what action is required.
; The end of each vendor specific command should be terminated with a
; return.
; ******************************************************************
CheckVendor
    global  CheckVendor
    return  ; *** remove this line and uncomment out the remainder

    end
```

APPENDIX F: MICROCONTROLLER FIRMWARE
Hidclass.asm

```
;                              Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated (the "Company")
; for its PICmicro(r) Microcontroller is intended and supplied to you, the Company's
; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
; products.
;
; The software is owned by the Company and/or its supplier, and is protected under
; applicable copyright laws. All rights are reserved. Any use in violation of the
; foregoing restrictions may subject the user to criminal sanctions under applicable
; laws, as well as to civil liability for the breach of the terms and conditions of
; this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
; ###############################################################################
;   filename:           HIDCLASS.ASM
;
;   Implements USB Human Interface Device (HID) class specific commands.
;
; ###############################################################################
;
;   Author(s):          Dan Butler and Reston Condit
;   Company:            Microchip Technology Inc
;
;   Revision:           1.24
;   Date:               5 March 2002
;   Assembled using:    MPASM 2.61
;################################################################################
;
;   include files:
;       P16C765.inc     Rev 1.00
;       usb_defs.inc    Rev 1.10
;
;################################################################################
#include <p16C765.inc>
#include "usb_defs.inc"

    extern  ReportDescriptor
    extern  ReportDescriptorLen
    extern  HID_Descriptor
    extern  Descriptions
    extern  BufferData
    extern  BufferDescriptor
    extern  wrongstate
    extern  USB_dev_req
    extern  EP0_maxLength
    extern  EP0_start
    extern  EP0_end
    extern  copy_descriptor_to_EP0
    extern  Send_0Len_pkt
    extern  Report_desc_index

    global  ClassSpecificRequest

USBBANK     code
; ****************************************************************
; Get Class Specific Descriptor
; ****************************************************************
```

```
ClassSpecificRequest
    pagesel Dev2HostHIDRequest
    movf    BufferData+bmRequestType,w
    xorlw   0x21
    btfsc   STATUS,Z
    goto    Host2DevHIDRequest

       pagesel Host2DevReportRequest
    movf    BufferData+bmRequestType,w
    xorlw   0x22
    btfsc   STATUS,Z
    goto    Host2DevReportRequest

       pagesel Host2DevPhysicalRequest
    movf    BufferData+bmRequestType,w
    xorlw   0x23
    btfsc   STATUS,Z
    goto    Host2DevPhysicalRequest

       pagesel Dev2HostHIDRequest
    movf    BufferData+bmRequestType,w
    xorlw   0xA1
    btfsc   STATUS,Z
    goto    Dev2HostHIDRequest

       pagesel Dev2HostReportRequest
    movf    BufferData+bmRequestType,w
    xorlw   0xA2
    btfsc   STATUS,Z
    goto    Dev2HostReportRequest

       pagesel Dev2HostPhysicalRequest
    movf    BufferData+bmRequestType,w
    xorlw   0xA3
    btfsc   STATUS,Z
    goto    Dev2HostPhysicalRequest

    pagesel wrongstate
    goto    wrongstate

    ; Need to add code if you need to handle optional functions
    ; such as get/set_idle. Otherwise, send STALL buy calling
    ; to signal the host that the feature is not implemented.

Host2DevHIDRequest
    movf    BufferData+bRequest,w
    xorlw   0x01
       pagesel GetHIDReport
    btfsc   STATUS,Z
    goto    GetHIDReport

    movf    BufferData+bRequest,w
    xorlw   0x02
       pagesel GetIdle
    btfsc   STATUS,Z
    goto    GetIdle

    movf    BufferData+bRequest,w
    xorlw   0x03
       pagesel GetPhysical
    btfsc   STATUS,Z
    goto    GetPhysical

    movf    BufferData+bRequest,w
    xorlw   0x06
       pagesel Get_Report_Descriptor
    btfsc   STATUS,Z
    goto    Get_Report_Descriptor
```

```
    movf    BufferData+bRequest,w
    xorlw   0x09
       pagesel SetHIDReport
    btfsc   STATUS,Z
    goto    SetHIDReport

    movf    BufferData+bRequest,w
    xorlw   0x0A
       pagesel SetIdle
    btfsc   STATUS,Z
    goto    SetIdle

    movf    BufferData+bRequest,w
    xorlw   0x0B
       pagesel SetProtocol
    btfsc   STATUS,Z
    goto    SetProtocol

    pagesel wrongstate
    goto    wrongstate

; ****************************************************************
; Get Report Descriptor
; Returns the Mouse Report descriptor
; Checks for the report type (input, output or Feature).
; ****************************************************************
Get_Report_Descriptor
    global  Get_Report_Descriptor
    banksel EP0_start
    movlw   GET_DESCRIPTOR
    movwf   USB_dev_req     ; currently processing a get descriptor request

    movlw   8
    movwf   EP0_maxLength

    movf    BufferData+(wValue+1),w ; check report ID
    xorlw   0x01                    ; was it an Input Report?
       pagesel TryOutputReport
    btfsc   STATUS,Z
    goto    TryOutputReport

    bcf     STATUS,C
    rlf     BufferData+wIndex,w
    pagesel Report_desc_index
    call    Report_desc_index   ; translate index to offset into descriptor table
    movwf   EP0_start
    bcf     STATUS,C
    rlf     BufferData+wIndex,w
    addlw   1                   ; point to high order byte
    call    Report_desc_index   ; translate index to offset into descriptor table
    movwf   EP0_start+1
       pagesel Descriptions
    call    Descriptions
    movwf   EP0_end
    incf    EP0_start,f
    pagesel CheckReportLength
    goto    CheckReportLength

TryOutputReport
    movf    BufferData+(wValue+1),w ; check report ID
    xorlw   0x02                    ; was it an Output Report?
       pagesel TryFeatureReport
    btfsc   STATUS,Z
    goto    TryFeatureReport

    bcf     STATUS,C
    rlf     BufferData+wIndex,w
       pagesel Report_desc_index
```

```
    call     Report_desc_index    ; translate index to offset into descriptor table
    movwf    EP0_start
    bcf      STATUS,C
    rlf      BufferData+wIndex,w
    addlw    1                    ; point to high order byte
    call     Report_desc_index    ; translate index to offset into descriptor table
    movwf    EP0_start+1
       pagesel Descriptions
    call     Descriptions
    movwf    EP0_end
    incf     EP0_start,f
    pagesel CheckReportLength
    goto     CheckReportLength

TryFeatureReport
    movf     BufferData+(wValue+1),w ; check report ID
    xorlw    0x03            ; was it an Output Report?
    pagesel wrongstate
    btfsc    STATUS,Z
    goto     wrongstate

; Fill EP0IN buffer here...
    return

CheckReportLength
    movf     BufferData+(wLength+1),w ; Is the host requesting more than 255 bytes?
       pagesel nolimit_rpt
    btfss    STATUS,Z        ; If so, the host is requesting more than we have
    goto     nolimit_rpt

check_low_bytes
    movf     BufferData+wLength,w
    subwf    EP0_end,w       ; if not, compare the amount the host is request
    movf     BufferData+wLength,w ; with the length of the descriptor
    btfsc    STATUS,C        ; if the host is request less than the descriptor
    movwf    EP0_end         ; length, send only as much as what the host wants

nolimit_rpt
    incf     EP0_end,f
    pagesel copy_descriptor_to_EP0
    call     copy_descriptor_to_EP0
    return

Get_HID_Descriptor
    global   Get_HID_Descriptor
    movlw    GET_DESCRIPTOR
    movwf    USB_dev_req     ; currently processing a get descriptor request

    movlw    8
    movwf    EP0_maxLength

    movlw    low HID_Descriptor
    movwf    EP0_start
    movlw    high HID_Descriptor
    movwf    EP0_start + 1
    pagesel Descriptions
    call     Descriptions    ; get the HID descriptor length
    movwf    EP0_end

    movf     BufferData+(wLength+1),f
       pagesel nolimit_hid
    btfss    STATUS,Z
    goto     nolimit_hid

    subwf    BufferData+wLength,w
    movf     BufferData+wLength,w
    btfss    STATUS,C
    movwf    EP0_end
```

```
nolimit_hid
    incf    EP0_end,f
    pagesel copy_descriptor_to_EP0
    call    copy_descriptor_to_EP0
    return

Get_Physical_Descriptor
    return

Check_Class_Specific_IN
    global  Check_Class_Specific_IN
    pagesel copy_descriptor_to_EP0
    movf    USB_dev_req,w
    xorlw   GET_DESCRIPTOR
    btfsc   STATUS,Z
    call    copy_descriptor_to_EP0
    return

; ******************************************************************
; These requests are parsed out, but nothing is actually done with them
; currently they simply stall EP0 to show that the request is not
; supported.  If you need to support them, fill in the code.
; ******************************************************************
Host2DevReportRequest
Host2DevPhysicalRequest
Dev2HostHIDRequest
Dev2HostReportRequest
Dev2HostPhysicalRequest
GetHIDReport
GetIdle
GetPhysical
SetProtocol
SetIdle
    pagesel wrongstate
    goto    wrongstate

SetHIDReport
    movlw   HID_SET_REPORT
    movwf   USB_dev_req     ; store status
    banksel BD0OST
    return

    end
```

APPENDIX G: MICROCONTROLLER FIRMWARE

Descript.asm

```
;                              Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated (the "Company")
; for its PICmicro(r) Microcontroller is intended and supplied to you, the Company's
; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
; products.
;
; The software is owned by the Company and/or its supplier, and is protected under
; applicable copyright laws. All rights are reserved. Any use in violation of the
; foregoing restrictions may subject the user to criminal sanctions under applicable
; laws, as well as to civil liability for the breach of the terms and conditions of
; this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
; ###############################################################################
;   filename:           DESCRIPT.ASM
;
; This file contains a set of descriptors for a standard mouse.
;
; ###############################################################################
;
;   Author:             Dan Butler and Reston Condit
;   Company:            Microchip Technology Inc
;
;   Revision:           1.24
;   Date:               5 March 2002
;   Assembled using     MPASM 2.61
;################################################################################
;
;   include files:
;       P16C765.inc     Rev 1.00
;       usb_defs.inc    Rev 1.10
;
;################################################################################
#include <p16C765.inc>
#include "usb_defs.inc"

USBBANK code
    global  Config_desc_index
    global  Report_desc_index
    global  Descriptions
    global  string_index
    global  DeviceDescriptor
    global  ReportDescriptor
    global  ReportDescriptorLen
    global  String0
    global  String0_end
    global  StringDescriptions
    global  HID_Descriptor

    extern  EP0_start
    extern  temp    ; temp var used in get config index
    extern  temp2   ; another temp, in bank2

; ******************************************************************
; Given a configuration descriptor index, returns the beginning address
; of the descriptor within the descriptions table
; ******************************************************************
```

```
Config_desc_index
    movwf   temp
    movlw   HIGH CDI_start
    movwf   PCLATH
    movlw   low  CDI_start
    addwf   temp,w
    btfsc   STATUS,C
    incf    PCLATH,f
    movwf   PCL
CDI_start           ; this table calculates the offsets for each configuration
    retlw   low  Config1        ; descriptor from the beginning
    retlw   high Config1        ; of the table, effectively
    ; more configurations can be added here
    ; retlw   low Config2
    ; retlw   high Config2
    ; etc....

; ******************************************************************
; Given a report descriptor index, returns the beginning address
; of the descriptor within the descriptions table
; ******************************************************************
Report_desc_index
    movwf   temp
    movlw   HIGH RDI_start
    movwf   PCLATH
    movlw   low  RDI_start
    addwf   temp,w
    btfsc   STATUS,C
    incf    PCLATH,f
    movwf   PCL
RDI_start       ; this table calculates the offsets for each report
    retlw   low  ReportDescriptorLen  ; descriptor from the beginning
    retlw   high ReportDescriptorLen  ; of the table, effectively
    ; more reports can be added here
    ; retlw   low  ReportDescriptorLen2
    ; retlw   high ReportDescriptorLen2
    ; etc....

; ******************************************************************
; This table is polled by the host immediately after USB Reset has been released.
; This table defines the maximum packet size EP0 can take.
; See section 9.6.1 of the Rev 1.0 USB specification.
; These fields are application DEPENDENT. Modify these to meet
; your specifications.
; the offset is passed in P0 and P1 (P0 is low order byte).
; ******************************************************************
Descriptions
    banksel EP0_start
    movf    EP0_start+1,w
    movwf   PCLATH
    movf    EP0_start,w
    movwf   PCL

DeviceDescriptor
StartDevDescr
    retlw   0x12    ; bLength    Length of this descriptor
    retlw   0x01    ; bDescType This is a DEVICE descriptor
    retlw   0x00    ; bcdUSB     USB revision 1.10 (low byte)
    retlw   0x01    ; high byte
    retlw   0x00    ; bDeviceClass    zero means each interface operates independently
    retlw   0x00    ; bDeviceSubClass
    retlw   0x00    ; bDeviceProtocol
    retlw   0x08    ; bMaxPacketSize0 - inited in UsbInit()
    retlw   0xD8    ; idVendor - 0x04D8 is Microchip Vendor ID
    retlw   0x04    ; high order byte
    retlw   0x00    ; idProduct
    retlw   0x00
    retlw   0x41    ; bcdDevice
    retlw   0x04
```

```
    retlw   0x01    ; iManufacturer
    retlw   0x02    ; iProduct
    retlw   0x00    ; iSerialNumber - 3
    retlw   NUM_CONFIGURATIONS    ; bNumConfigurations

; ******************************************************************
; This table is retrieved by the host after the address has been set.
; This table defines the configurations available for the device.
; See section 9.6.2 of the Rev 1.0 USB specification (page 184).
; These fields are application DEPENDENT.
; Modify these to meet your specifications.
; ******************************************************************
Config1
    retlw   0x09    ; bLength    Length of this descriptor
    retlw   0x02    ; bDescType    2=CONFIGURATION
    retlw   EndConfig1 - Config1
    retlw   0x00
    retlw   0x01    ; bNumInterfaces    Number of interfaces
    retlw   0x01    ; bConfigValue    Configuration Value
    retlw   0x04    ; iConfig    String Index for this config = #01
    retlw   0xA0    ; bmAttributes    attributes - bus powered
    retlw   0x32    ; MaxPower    self-powered draws 0 mA from the bus.
Interface1
    retlw   0x09    ; length of descriptor
    retlw   INTERFACE
    retlw   0x00    ; number of interface, 0 based array
    retlw   0x00    ; alternate setting
    retlw   0x01    ; number of endpoints used in this interface
    retlw   0x03    ; interface class - assigned by the USB
    retlw   0x01    ; boot device
    retlw   0x02    ; interface protocol - mouse
    retlw   0x05    ; index to string descriptor that describes this interface
HID_Descriptor
    retlw   0x09    ; descriptor size (9 bytes)
    retlw   0x21    ; descriptor type (HID)
    retlw   0x00
    retlw   0x01    ; HID class release number (1.00)
    retlw   0x00    ; Localized country code (none)
    retlw   0x01    ; # of HID class descriptor to follow (1)
    retlw   0x22    ; Report descriptor type (HID)
    retlw   (end_ReportDescriptor - ReportDescriptor)
    retlw   0x00
Endpoint1
    retlw   0x07    ; length of descriptor
    retlw   ENDPOINT
    retlw   0x81    ; EP1, In
    retlw   0x03    ; Interrupt
    retlw   0x04    ; max packet size (4 bytes) low order byte
    retlw   0x00    ; max packet size (4 bytes) high order byte
    retlw   0x0A    ; polling interval (10ms)
EndConfig1

ReportDescriptorLen
    retlw   low     (end_ReportDescriptor-ReportDescriptor)

ReportDescriptor
    retlw   0x05
    retlw   0x01    ; usage page (generic desktop)
    retlw   0x09
    retlw   0x02    ; usage (mouse)
    retlw   0xA1
    retlw   0x01    ; collection (application)
    retlw   0x09
    retlw   0x01    ;   usage (pointer)
    retlw   0xA1
    retlw   0x00    ;   collection (linked)
    retlw   0x05
    retlw   0x09    ;     usage page (buttons)
    retlw   0x19
```

```
    retlw   0x01    ;     usage minimum (1)
    retlw   0x29
    retlw   0x03    ;     usage maximum (3)
    retlw   0x15
    retlw   0x00    ;     logical minimum (0)
    retlw   0x25
    retlw   0x01    ;     logical maximum (1)
    retlw   0x95
    retlw   0x03    ;     report count (3)
    retlw   0x75
    retlw   0x01    ;     report size (1)
    retlw   0x81
    retlw   0x02    ;     input (3 button bits)
    retlw   0x95
    retlw   0x01    ;     report count (1)
    retlw   0x75
    retlw   0x05    ;     report size (5)
    retlw   0x81
    retlw   0x01    ;     input (constant 5 bit padding)
    retlw   0x05
    retlw   0x01    ;     usage page (generic desktop)
    retlw   0x09
    retlw   0x30    ;     usage (X)
    retlw   0x09
    retlw   0x31    ;     usage (Y)
    retlw   0x15
    retlw   0x81    ;     logical minimum (-127)
    retlw   0x25
    retlw   0x7F    ;     logical maximum (127)
    retlw   0x75
    retlw   0x08    ;     report size (8)
    retlw   0x95
    retlw   0x03    ;     report count (2)
    retlw   0x81
    retlw   0x06    ;     input (2 position bytes X & Y)
    retlw   0xC0    ;   end collection
    retlw   0xC0    ; end collection
end_ReportDescriptor

StringDescriptions
    banksel EP0_start
    movf    EP0_start+1,w
    movwf   PCLATH
    movf    EP0_start,w
    movwf   PCL

; ******************************************************************
; Given a configuration descriptor index, returns the beginning address
; of the descriptor within the descriptions table
; ******************************************************************
string_index        ; langid in W reg, string offset in EP0_start
    movwf   temp
    bcf     STATUS,C
    rlf     temp, f
    pagesel langid_index
    call    langid_index
    movwf   temp2
    incf    temp, f
    pagesel langid_index
    call    langid_index
    movwf   temp

    movf    temp, w
    movwf   PCLATH
    movf    temp2,w
    addwf   EP0_start+1,w
    btfsc   STATUS,C
    incf    PCLATH, f
    movwf   PCL
```

```
langid_index
    movlw   high langids
    movwf   PCLATH
    movlw   low langids
    addwf   temp, w
    btfsc   STATUS,C
    incf    PCLATH,f
    movwf   PCL

langids
    retlw   low lang_1
    retlw   high lang_1
    retlw   low lang_2      ; string indexes of different languages
    retlw   high lang_2

lang_1                      ; english
    retlw   low  String0    ; LangIDs
    retlw   high String0
    retlw   low  String1_l1
    retlw   high String1_l1
    retlw   low  String2_l1
    retlw   high String2_l1
    retlw   low  String3_l1
    retlw   high String3_l1
    retlw   low  String4_l1
    retlw   high String4_l1
    retlw   low  String5_l1
    retlw   high String5_l1
    retlw   low  String6_l1
    retlw   high String6_l1

lang_2
    retlw   low  String0    ; also point to LangID
    retlw   high String0
    retlw   low  String1_l2
    retlw   high String1_l2
    retlw   low  String2_l2
    retlw   high String2_l2
    retlw   low  String3_l2
    retlw   high String3_l2
    retlw   low  String4_l2
    retlw   high String4_l2
    retlw   low  String5_l2
    retlw   high String5_l2
String0
    retlw   low (String1_l1 - String0)    ; length of string
    retlw   0x03    ; descriptor type 3?
    retlw   0x09    ; language ID (as defined by MS 0x0409)
    retlw   0x04
    retlw   0x04    ; some other language ID for testing
    retlw   0x08
String0_end
String1_l1
    retlw   String2_l1-String1_l1    ; length of string
    retlw   0x03    ; string descriptor type 3
    retlw   'M'
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   'r'
    retlw   0x00
    retlw   'o'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   'h'
```

```
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'p'
    retlw   0x00
String2_l1
    retlw   String3_l1-String2_l1
    retlw   0x03
    retlw   'P'
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '6'
    retlw   0x00
    retlw   'C'
    retlw   0x00
    retlw   '7'
    retlw   0x00
    retlw   '4'
    retlw   0x00
    retlw   '5'
    retlw   0x00
    retlw   '/'
    retlw   0x00
    retlw   '7'
    retlw   0x00
    retlw   '6'
    retlw   0x00
    retlw   '5'
    retlw   0x00
    retlw   ' '
    retlw   0x00
    retlw   'U'
    retlw   0x00
    retlw   'S'
    retlw   0x00
    retlw   'B'
    retlw   0x00
    retlw   ' '
    retlw   0x00
    retlw   'M'
    retlw   0x00
    retlw   'o'
    retlw   0x00
    retlw   'u'
    retlw   0x00
    retlw   's'
    retlw   0x00
    retlw   'e'
    retlw   0x00
String3_l1
    retlw   String4_l1-String3_l1
    retlw   0x03
    retlw   'V'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '.'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '1'
    retlw   0x00
String4_l1
    retlw   String5_l1-String4_l1
```

```
    retlw   0x03
    retlw   'C'
    retlw   0x00
    retlw   'f'
    retlw   0x00
    retlw    'g'
    retlw   0x00
    retlw    '1'
    retlw   0x00
String5_l1
    retlw   String6_l1-String5_l1
    retlw   0x03
    retlw   'E'
    retlw   0x00
    retlw   'P'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '0'
    retlw   0x00
    retlw   'I'
    retlw   0x00
    retlw   'n'
    retlw   0x00
String6_l1
String1_l2          ; lang 2, chinese. String can be totally different than english
    retlw   String2_l2-String1_l2    ; length of string
    retlw   0x03    ; string descriptor type 3
    retlw   'M'
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   'r'
    retlw   0x00
    retlw   'o'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   'h'
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'p'
    retlw   0x00
String2_l2
    retlw   String3_l2-String2_l2
    retlw   0x03
    retlw   'P'
    retlw   0x00
    retlw   'i'
    retlw   0x00
    retlw   'c'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '6'
    retlw   0x00
    retlw   'C'
    retlw   0x00
    retlw   '7'
    retlw   0x00
    retlw   '4'
    retlw   0x00
    retlw   '5'
    retlw   0x00
    retlw   '/'
    retlw   0x00
```

```
    retlw   '7'
    retlw   0x00
    retlw   '6'
    retlw   0x00
    retlw   '5'
    retlw   0x00
    retlw   ' '
    retlw   0x00
    retlw   'U'
    retlw   0x00
    retlw   'S'
    retlw   0x00
    retlw   'B'
    retlw   0x00
    retlw   ' '
    retlw   0x00
    retlw   'M'
    retlw   0x00
    retlw   'o'
    retlw   0x00
    retlw   'u'
    retlw   0x00
    retlw   's'
    retlw   0x00
    retlw   'e'
    retlw   0x00
String3_12
    retlw   String4_12-String3_12
    retlw   0x03
    retlw   'V'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '.'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '1'
    retlw   0x00
String4_12
    retlw   String5_12-String4_12
    retlw   0x03
    retlw   'C'
    retlw   0x00
    retlw   'f'
    retlw   0x00
    retlw   'g'
    retlw   0x00
    retlw   '1'
    retlw   0x00
String5_12
    retlw   String6_12-String5_12
    retlw   0x03
    retlw   'E'
    retlw   0x00
    retlw   'P'
    retlw   0x00
    retlw   '1'
    retlw   0x00
    retlw   '0'
    retlw   0x00
    retlw   'I'
    retlw   0x00
    retlw   'n'
    retlw   0x00
String6_12

    end
```

APPENDIX H: MICROCONTROLLER FIRMWARE

Usb_defs.inc

```
;                           Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated (the "Company")
; for its PICmicro® Microcontroller is intended and supplied to you, the Company's
; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
; products.
;
; The software is owned by the Company and/or its supplier, and is protected under
; applicable copyright laws. All rights are reserved. Any use in violation of the
; foregoing restrictions may subject the user to criminal sanctions under applicable
; laws, as well as to civil liability for the breach of the terms and conditions of
; this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
; ###############################################################################
;       filename:               USB_DEFS.INC
;                               Definitions used throughout the Chapter 9 code
;
; ###############################################################################
;
;   Author:                         Dan Butler and Reston Condit
;   Company:                Microchip Technology Inc
;
;       Revision:                   1.22
;       Date:                       07 September 2001
;       Assembled using:        MPASM 2.61
;
;################################################################################
;
;       include files:
;               none
;
;################################################################################
; Edit these as appropriate for your descriptors
#define NUM_CONFIGURATIONS  1
#define NUM_INTERFACES      1

; Define the states that the USB interface can be in
#define POWERED_STATE  0x00
#define DEFAULT_STATE  0x01
#define ADDRESS_STATE  0x02
#define CONFIG_STATE   0x03

; Define the states for Control EndPoints
#define EP_IDLE_STATE  0x00
#define EP_SETUP_STATE 0x01
#define EP_DISABLED_STATE     0xff

#define ENDPT_DISABLED 0x00
#define ENDPT_IN_ONLY  0x02
#define ENDPT_OUT_ONLY 0x04
#define ENDPT_CONTROL  0x06    ; enable for in, out and setup
#define ENDPT_NON_CONTROL 0x0E        ; enable for in, and out


#define INT_STAT_MASK_RESET          0x01
#define INT_STAT_MASK_ERROR          0x02
#define INT_STAT_MASK_TOKEN_DONE     0x04
```

```
#define INT_STAT_MASK_SLEEP          0x08
#define INT_STAT_MASK_STALL          0x10

#define TOKEN_OUT      (0x01<<2)
#define TOKEN_ACK       (0x02<<2)
#define TOKEN_IN       (0x09<<2)
#define TOKEN_SETUP    (0x0D<<2)

#define USB_Buffer    0xB8    ; on page 3 so actual address 0x1B8

; offsets from the beginning of the Buffer Descriptor
#define BYTECOUNT     0x01
#define ADDRESS       0x02

; Descriptor types
#define  DEVICE          1
#define  CONFIGURATION  2
#define  STRING          3
#define  INTERFACE       4
#define  ENDPOINT        5

; offsets from the beginning of the setup data record
#define  bmRequestType  0x00
#define  bRequest      0x01
#define  wValue                0x02
#define  wValueHigh    0x03
#define  wIndex                0x04
#define  wIndexHigh    0x05
#define wLength        0x06
#define  wLengthHigh   0x07

#define  CLEAR_FEATURE        0x01
#define  GET_CONFIGURATION    0x08
#define  GET_DESCRIPTOR               0x06
#define  GET_STRING_DESCRIPTOR        0x66
#define  GET_INTERFACE        0x0A
#define  GET_STATUS           0x00
#define SET_ADDRESS           0x05
#define  SET_CONFIGURATION    0x09
#define  SET_FEATURE          0x03
#define  SET_INTERFACE        0x0B
#define  HID_SET_REPORT               0x21
#define  VEND_SET_MEMORY      0x80

#define SVCUSBINT             0x01 << 2
#define SVCTOKENDONE          0x02 << 2
#define SVCRESET              0x03 << 2
#define SVCSLEEP              0x04 << 2
#define SVCSTALL              0x05 << 2
#define SVCERROR              0x06 << 2
#define SVCACTIVITY           0x07 << 2
#define TOKENOUT              0x08 << 2
#define TOKENIN                       0x09 << 2
#define TOKENSETUP            0x0A << 2
#define CLEARFEATURE          0x0B << 2
#define GETCONFIG             0x0C << 2
#define GETDESCRIPTOR         0x0D << 2
#define GETINTERFACE          0x0E << 2
#define GETSTATUS             0x0F << 2
#define SETADDRESS            0x10 << 2
#define SETCONFIG             0x11 << 2
#define SETFEATURE            0x12 << 2
#define SETINTERFACE          0x13 << 2
#define FINISHSETADDRESS      0x14 << 2
#define COPYDESC2EP0          0x15 << 2
#define COPYSTRINGDESC2EP0    0x16 << 2
#define ZEROLENPACKET         0x17 << 2

COPYBUFFERDESCRIPTOR  macro
```

```
        bankisel BD0OST
        banksel BD0OST
        movf    USTAT,w         ; get the status register
        addlw   0xA0            ; add the offset to the beginning of the BD's
        movwf   FSR             ; save in the FSR.
        bcf     STATUS, RP0     ; back to bank 2
        movf    INDF,w
        movwf   BufferDescriptor       ; in shared RAM
        incf    FSR,f
        movf    INDF,w
        movwf   BufferDescriptor+1
        incf    FSR,f
        movf    INDF,w
        movwf   BufferDescriptor+2
        endm

; Increments a 16bit counter, stored Lowbyte:Highbyte
INCREMENT16     macro   index
        local   endinc16
        incfsz  index,f
        goto    endinc16
        incf    index+1,f
endinc16
        endm

REQUESTERROR    macro
        bsf     STATUS,RP0      ; page 3
        movf    USTAT,w         ; get the status register
        addlw   0xA0            ; add the offset to the beginning of the BD's
        movwf   FSR
        bsf     INDF,EP_STALL   ; set endpoint stall bit
        bcf     STATUS,RP0      ; back to page 2
        endm

; ********************************************************************
; wait here until the enumeration process is complete.
; This is implemented as a macro to avoid chewing up another stack level
; ********************************************************************
ConfiguredUSB macro
        local   enumloop
        banksel USWSTAT
        pagesel enumloop
enumloop
        clrwdt                  ; clear the watch dog timer.
        movlw   0x03
        andwf   USWSTAT,w       ; save lower 2 bits of USWSTAT
        xorlw   CONFIG_STATE    ; compare with configured state
        btfss   STATUS,Z        ; are we configured?
        goto    enumloop        ; nope, keep waiting ...
        endm

GETEP1  macro
; *********************************************************************
; GetEP1
; Enter with buffer pointer in IRP+FSR.
; Checks the semaphore for the OUT endpoint, and copies the buffer
; if available.  Restores the bank bits as we found them.
;
; Returns the bytecount in the W register and return status in the carry
; bit as follows:
; 0 - no buffer available,
; 1 - Buffer copied and buffer made available for next transfer.
;
; The number of bytes moved is returned in W reg.
; *********************************************************************
GetEP1
    global  GetEP1
    local   getEPloop
    local   exitgetloop
```

```
    local   nobuffer

    movf    STATUS,w            ; save bank bits before we trash them
    banksel RP_save             ; switch to bank 2
    movwf   RP_save

    movf    FSR,w
    movwf   dest_ptr            ; save the buffer destination pointer

    banksel BD1OST              ; bank 3
       pagesel nobuffer
    btfsc   BD1OST,UOWN         ; Has the buffer been filled?
    goto    nobuffer            ; nope, OWN = 1 ==> SIE owns the buffer
                                ; Yep: OWN = 0 ==> PIC owns the buffer

    movf    BD1OBC,w            ; get byte count
    banksel counter             ; bank 2
    movwf   counter
    movwf   bytecounter         ; # of bytes that will be moved
       pagesel exitgetloop
    btfsc   STATUS,Z            ; is it a zero length buffer?
    goto    exitgetloop         ; yes, bail out now and avoid the rush
    banksel BD1OAL              ; bank 3
    movf    BD1OAL,w            ; get address pointer
    banksel source_ptr          ; bank 2
    movwf   source_ptr

; This loop operates with the direct bank bits set to bank 2, while IRP
; gets switched as needed to for the buffer copy
getEPloop
    bsf     STATUS,IRP          ; select high banks on INDF
    movf    source_ptr,w        ; get source pointer
    movwf   FSR
    movf    INDF,w
    movwf   GPtemp              ; in common RAM to avoid paging issues

    movf    dest_ptr,w
    movwf   FSR
    btfss   RP_save,IRP         ; should it be zero?
    bcf     STATUS,IRP          ; yes: make it so.
    movf    GPtemp,w            ; no, get the byte we read.
    movwf   INDF                ; store it
    incf    dest_ptr,f
    incf    source_ptr,f
       pagesel getEPloop
    decfsz  counter,f
    goto    getEPloop

exitgetloop
    bsf     STATUS,RP0          ; bank 3
    movf    BD1OST,w
    andlw   0x40                ; save only the data 0/1 bit
    xorlw   0x40                ; toggle the data o/1 bit
    iorlw   0x88                ; set owns bit and DTS bit
    movwf   BD1OST

    movlw   0x08                ; reset byte counter
    movwf   BD1OBC
    bcf     STATUS,RP0          ; bank 2
    movf    bytecounter,w       ; return # of bytes moved in W reg
    movwf   GPtemp              ; move byte counter to temporary ram
    movf    RP_save,w           ; restore bank bits
    movwf   STATUS
    movf    GPtemp,w            ; load W with the byte count
    bsf     STATUS,C            ; signal success
    return

nobuffer
    banksel RP_save             ; restore the bank bits
```

```
    movf    RP_save,w
    movwf   STATUS
    bcf     STATUS,C
    return
    endm

GETEP2  macro
; *******************************************************************
; GetEP2
; Enter with buffer pointer in IRP+FSR.
; Checks the semaphore for the OUT endpoint, and copies the buffer
; if available.  Restores the bank bits as we found them.
;
; Returns the bytecount in the W register and return status in the carry
; bit as follows:
; 0 - no buffer available,
; 1 - Buffer copied and buffer made available for next transfer.
;
; The number of bytes moved is returned in W reg.
; *******************************************************************
GetEP2
    global  GetEP2
    local   getEPloop2
    local   exitgetloop2
    local   nobuffer2

    movf    STATUS,w        ; save bank bits before we trash them
    banksel RP_save         ; switch to bank 2
    movwf   RP_save

    movf    FSR,w
    movwf   dest_ptr        ; save the buffer destination pointer

    banksel BD2OST          ; bank 3
       pagesel nobuffer2
    btfsc   BD2OST,UOWN     ; Has the buffer been filled?
    goto    nobuffer2       ; nope, OWN = 1 ==> SIE owns the buffer
                            ; Yep: OWN = 0 ==> PIC owns the buffer

    movf    BD2OBC,w        ; get byte count
    banksel counter         ; bank 2
    movwf   counter
    movwf   bytecounter     ; # of bytes that will be moved
       pagesel exitgetloop2
    btfsc   STATUS,Z        ; is it a zero length buffer?
    goto    exitgetloop2    ; yes, bail out now and avoid the rush
    banksel BD2OAL          ; bank 3
    movf    BD2OAL,w        ; get address pointer
    banksel source_ptr      ; bank 2
    movwf   source_ptr

; This loop operates with the direct bank bits set to bank 2, while IRP
; gets switched as needed to for the buffer copy
getEPloop2
    bsf     STATUS,IRP      ; select high banks on INDF
    movf    source_ptr,w    ; get source pointer
    movwf   FSR
    movf    INDF,w
    movwf   GPtemp          ; in common RAM to avoid paging issues

    movf    dest_ptr,w
    movwf   FSR
    btfss   RP_save,IRP     ; should it be zero?
    bcf     STATUS,IRP      ; yes: make it so.
    movf    GPtemp,w        ; no, get the byte we read.
    movwf   INDF            ; store it
    incf    dest_ptr,f
    incf    source_ptr,f
       pagesel getEPloop2
```

```
    decfsz  counter,f
    goto    getEPloop2

exitgetloop2
    bsf     STATUS,RP0          ; bank 3
    movf    BD2OST,w
    andlw   0x40                ; save only the data 0/1 bit
    xorlw   0x40                ; toggle the data o/1 bit
    iorlw   0x88                ; set owns bit and DTS bit
    movwf   BD2OST

    movlw   0x08                ; reset byte counter
    movwf   BD2OBC
    bcf     STATUS,RP0          ; bank 2
    movf    bytecounter,w       ; return # of bytes moved in W reg
    movwf   GPtemp              ; move byte counter to temporary ram
    movf    RP_save,w           ; restore bank bits
    movwf   STATUS
    movf    GPtemp,w            ; load W with the byte count
    bsf     STATUS,C            ; signal success
    return

nobuffer2
    banksel RP_save             ; restore the bank bits
    movf    RP_save,w
    movwf   STATUS
    bcf     STATUS,C
    return
    endm

PUTEP1  macro
; ******************************************************************
; PutEP1
; Enter with bytecount in W and buffer pointer in IRP+FSR.
; the bytecount is encoded in the lower nybble of W.
;
; Tests the owns bit for the IN side of the specified Endpoint.
; If we own the buffer, the buffer pointed to by the FSR is copied
; to the EPn In buffer, then the owns bit is set so the data will be
; TX'd next time polled.
;
; Returns the status in the carry bit as follows:
; 1 - buffer available and copied.
; 0 - buffer not available (try again later)
; ******************************************************************
PutEP1
    global  PutEP1
    local   putEPloop
    local   exitputloop
    local   nobufferputep

    movwf   GPtemp              ; save Bytecount temporarily in common RAM

    movf    STATUS,w            ; save bank bits before we trash them
    banksel RP_save             ; switch to bank 2
    movwf   RP_save

    movf    GPtemp,w
    andlw   0x0F                ; extract byte count.
    movwf   counter

    movf    FSR,w
    movwf   source_ptr

    movf    counter,w           ; prepare to copy the byte count
    banksel BD1IST              ; bank 3
        pagesel nobufferputep1
    btfsc   BD1IST,UOWN         ; is the buffer already full?
    goto    nobufferputep1      ; yes - don't write over it
```

```
    movwf   BD1IBC          ; set byte count in BD
       pagesel exitputloop
    btfsc   STATUS,Z        ; is it a zero length buffer?
    goto    exitputloop     ; yes, bail out now and avoid the rush
    movf    BD1IAL,w        ; get address pointer
    bcf     STATUS,RP0      ; back to bank 2
    movwf   dest_ptr

; This loop operates with the direct bits set to bank 2, while IRP
; gets switched as needed to for the buffer copy
putEPloop
    bsf     STATUS,IRP      ; assume IRP is set
    btfss   RP_save,IRP     ; should it be zero?
    bcf     STATUS,IRP      ; yes: make it so.
    movf    source_ptr,w
    movwf   FSR
    movf    INDF,w
    movwf   GPtemp

    bsf     STATUS,IRP      ; select high banks on INDF
    movf    dest_ptr,w
    movwf   FSR
    movf    GPtemp,w        ; no, get the byte we read.
    movwf   INDF            ; store it

    incf    dest_ptr,f
    incf    source_ptr,f
       pagesel putEPloop
    decfsz  counter,f
    goto    putEPloop

exitputloop
    bsf     STATUS,RP0      ; back to bank 3
    movf    BD1IST,w
    andlw   0x40            ; save only the data 0/1 bit
    xorlw   0x40            ; toggle the data o/1 bit
    iorlw   0x88            ; set owns bit and DTS bit
    movwf   BD1IST
    banksel RP_save
    movf    RP_save,w       ; restore bank bits the way we found them
    movwf   STATUS
    bsf     STATUS,C        ; set carry to show success
    return

nobufferputep1
    bcf     STATUS,C
    return
    endm

PUTEP2  macro
; ******************************************************************
; PutEP2
; Enter with bytecount in W and buffer pointer in IRP+FSR.
; the bytecount is encoded in the lower nybble of W.
;
; Tests the owns bit for the IN side of the specified Endpoint.
; If we own the buffer, the buffer pointed to by the FSR is copied
; to the EPn In buffer, then the owns bit is set so the data will be
; TX'd next time polled.
;
; Returns the status in the carry bit as follows:
; 1 - buffer available and copied.
; 0 - buffer not available (try again later)
; ******************************************************************
PutEP2
    global  PutEP2
    local   putEPloop2
    local   exitputloop2
```

```
    local   nobufferputep2

    movwf   GPtemp          ; save Bytecount temporarily in common RAM

    movf    STATUS,w        ; save bank bits before we trash them
    banksel RP_save         ; switch to bank 2
    movwf   RP_save

    movf    GPtemp,w
    andlw   0x0F            ; extract byte count.
    movwf   counter

    movf    FSR,w
    movwf   source_ptr

    movf    counter,w       ; prepare to copy the byte count
    banksel BD2IST          ; bank 3
       pagesel nobufferputep2
    btfsc   BD2IST,UOWN     ; is the buffer already full?
    goto    nobufferputep2  ; yes - don't write over it

    movwf   BD2IBC          ; set byte count in BD
       pagesel exitputloop2
    btfsc   STATUS,Z        ; is it a zero length buffer?
    goto    exitputloop2    ; yes, bail out now and avoid the rush
    movf    BD2IAL,w        ; get address pointer
    bcf     STATUS,RP0      ; back to bank 2
    movwf   dest_ptr

; This loop operates with the direct bits set to bank 2, while IRP
; gets switched as needed to for the buffer copy
putEPloop2
    bsf     STATUS,IRP      ; assume IRP is set
    btfss   RP_save,IRP     ; should it be zero?
    bcf     STATUS,IRP      ; yes: make it so.
    movf    source_ptr,w
    movwf   FSR
    movf    INDF,w
    movwf   GPtemp

    bsf     STATUS,IRP      ; select high banks on INDF
    movf    dest_ptr,w
    movwf   FSR
    movf    GPtemp,w        ; no, get the byte we read.
    movwf   INDF            ; store it

    incf    dest_ptr,f
    incf    source_ptr,f
       pagesel putEPloop2
    decfsz  counter,f
    goto    putEPloop2

exitputloop2
    bsf     STATUS,RP0      ; back to bank 3
    movf    BD2IST,w
    andlw   0x40            ; save only the data 0/1 bit
    xorlw   0x40            ; toggle the data o/1 bit
    iorlw   0x88            ; set owns bit and DTS bit
    movwf   BD2IST
    banksel RP_save
    movf    RP_save,w       ; restore bank bits the way we found them
    movwf   STATUS
    bsf     STATUS,C        ; set carry to show success
    return

nobufferputep2
    bcf     STATUS,C
    return
    end
```

APPENDIX I: MICROCONTROLLER FIRMWARE
PIC16C745.lkr

```
// File: 16c745.lkr
// Sample linker command file for 16C765, 16C745

LIBPATH  .

CODEPAGE   NAME=vectors   START=0x0      END=0x3F     PROTECTED

CODEPAGE   NAME=page0     START=0x40     END=0x7FF
CODEPAGE   NAME=page1     START=0x800    END=0xFFF
CODEPAGE   NAME=page2     START=0x1000   END=0x17FF
CODEPAGE   NAME=page3     START=0x1800   END=0x1FFF
CODEPAGE   NAME=.idlocs   START=0x2000   END=0x2003
CODEPAGE   NAME=.config   START=0x2007   END=0x2007

SHAREBANK  NAME=gprnobnk  START=0x70     END=0x7F
SHAREBANK  NAME=gprnobnk  START=0xF0     END=0xFF
SHAREBANK  NAME=gprnobnk  START=0x170    END=0x17F
SHAREBANK  NAME=gprnobnk  START=0x1F0    END=0x1FF

DATABANK   NAME=gpr0      START=0x20     END=0x6F
DATABANK   NAME=gpr1      START=0xA0     END=0xEF
DATABANK   NAME=gpr2      START=0x120    END=0x16F
DATABANK   NAME=gpr3      START=0x190    END=0x1EF

DATABANK   NAME=sfr0      START=0x0      END=0x1F    PROTECTED
DATABANK   NAME=sfr1      START=0x80     END=0x9F    PROTECTED
DATABANK   NAME=sfr2      START=0x100    END=0x11F   PROTECTED
DATABANK   NAME=sfr3      START=0x180    END=0x18F   PROTECTED

SECTION    NAME=STARTUP   ROM=vectors    // Reset and interrupt vectors
SECTION    NAME=PROG1     ROM=page0      // ROM code space - page0
SECTION    NAME=PROG2     ROM=page1      // ROM code space - page1
SECTION    NAME=PROG3     ROM=page2      // ROM code space - page2
SECTION    NAME=PROG4     ROM=page3      // ROM code space - page3
SECTION   NAME=IDLOCS   ROM=.idlocs   // ID locations
SECTION    NAME=CONFIG    ROM=.config // Configuration bits location
SECTION    NAME=bank0     RAM=gpr0
SECTION    NAME=bank1     RAM=gpr1
SECTION    NAME=bank2     RAM=gpr2
SECTION    NAME=unbanked RAM=gprnobnk    // unbanked RAM - last 16bytes of each bank
```

REFERENCE LIST

[1] Kohn, James. The Ergonomic Casebook: Real World Solutions. Boca Raton, FL: CRC Press, 1997.

[2] Peterson, Baird and Richard Patten. The Ergonomic PC: Creating A Healthy Computing Environment. New York City: McGraw Hill, 1995.

[3] Grandjean, Etienne. Fitting The Task To The Man: An Ergonomic Approach. London: Taylor & Francis, 1969.

[4] Weimer, Jon. Handbook Of Ergonomic And Human Factors Tables. Englewood Cliffs, NJ; PTR Prentice Hall, 1993.

[5] Smith, Wanda. ISO And ANSI Ergonomic Standards For Computer Products: A Guide To Implementation And Compliance. Upper Saddle River, NJ: PTR Prentice Hall, 1996.

[6] Grogono, Peter. Mouse, A Language For Microcomputers. New York City: Petrocelli Books, 1983.

[7] Brain, Marshall. How Computer Mice Work. Available: http://www.howstuffworks.com/mouse.htm.

[8] Goy, Carl. Input Devices: Mice. Ed. Sol Sherr. San Diego: Academic Press Inc., 1988.

[9] Tandeske, Duane. Pressure Sensors: Selection And Application. New York City: Marcel Dekker Inc., 1991.

[10] Beckwith, Thomas G., Roy D. Marangoni, and John H. Lienhard V. Mechanical Measurements. 5^{th} ed. New York City: Addison-Wesley Publishing Company Inc., 1995.

[11] Bell, David A. Operational Amplifiers: Applications, Troubleshooting, and Design. Englewood Cliffs, NJ: Prentice Hall Inc., 1990.

[12] Tekscan Inc. Flexiforce Sensors. Available: http://www.tekscan.com/flexiforce.html

[13] Iovine, John. PIC Microcontroller Project Book. New York City: McGraw Hill Inc., 2000.

[14] Katzen, Sid. The Quintessential PIC Microcontroller. London: Springer-Verlag, 2001.

[15] Daugherty, Kevin M. Analog-To-Digital Conversion: A Practical Approach. New York City: McGraw-Hill Inc., 1995.

[16] Demler, Michael J. High-Speed Analog-To-Digital Conversion. San Diego: Academic Press Inc., 1991.

[17] Pallas-Areny, Ramon, and John G. Webster. Analog Signal Processing. New York City: John Wiley & Sons Inc., 1999.

[18] The Engineering Staff of Analog Devices Inc. Analog-To-Digital Conversion Handbook. Ed. Daniel Sheingold. 3rd ed. Norwood, CA: Analog Devices Inc., 1986.

[19] Axelson, Jan. USB Complete: Everything You Need To Develop Custom USB Peripherals. Madison, WI: Lakeview Research, 1999.

[20] Hyde, John. USB Design By Example: A Practical Guide To Building I/O Devices. New York City: John Wiley & Sons, 1999.

[21] McDowell, Steven, and Martin D. Seyer. USB Explained. Upper Saddle River, NJ: Prentice Hall, 1999.

[22] Anderson, Don, and Dave Dzatko. Universal Serial Bus System Architecture. 2nd ed. Upper Saddle River, NJ: Mindshare Inc., 2001.

[23] Bard, Chantal, Michelle Fleury and Laurette Hay. Development Of Eye-hand Coordination Across The Life Span. Columbia, SC: University of South Carolina Press, 1990.

[24] Sutcliffe, Alistair. Human-Computer Interface Design. London: MacMillan Education Ltd., 1988.

VITA

NAME OF AUTHOR: Mohd Rapid Arifin

DATE AND PLACE OF BIRTH: October 11, 1978,
Pontian, Johor, MALAYSIA

DEGREES AWARDED:

A.A.D in Engineering, Unitek College of Malaysia, 1997
B.S in Mechanical Engineering, Iowa State University, 2000
M.S in Mechanical Engineering, Iowa State University, 2002

HONORS AND AWARDS:

High School Academic Outstanding Student Award (Chemistry), 1995
College/University Scholarship Award from Malaysian Government, 1996
Unitek College of Malaysia Dean's List, 1996-1997,
Iowa State University Dean's List, 1999 & 2000

PROFESSIONAL EXPERIENCE:

Research Assistant, Department of Mechanical Engineering, Iowa State University, 2000-2002
Teaching Assistant, Department of Mechanical Engineering, Iowa State University, 2002