

# Wireless USB Flash Storage Device

## Senior Project Final Report

Design Team #5

Jeff Czapor

Ben Hartney

Andrew Knight

Faculty Advisor:

Dr. Ugweje

Date Submitted:

May 3, 2006

Table of Contents

List of Figures .....	ii
List of Tables .....	iii
Abstract .....	1
Introduction.....	2
Design Requirements .....	4
Accepted Technical Design .....	7
Testing Procedures.....	52
Financial Budget .....	58
Project Schedule.....	61
Design Team Information .....	63
References.....	65
Appendix A.....	66
Appendix B .....	67

## List of Figures

Figure 1. Adapter Module Block Diagram .....	7
Figure 2. Storage Module Block Diagram .....	8
Figure 3. Pin-Out for the CYWUSB6934 .....	10
Figure 4. Block Diagram of the CYWUSB6934 .....	11
Figure 5. CYWM6934 Block Diagram .....	11
Figure 6. The Functional Block Diagram of the PIC18F4550 .....	15
Figure 7. I/O Pin Assignments of the PIC18F4450 .....	16
Figure 8. The PIC USB External Interface .....	16
Figure 9. Adapter PIC Power Connections .....	17
Figure 10. Storage PIC Power Connections .....	18
Figure 11. Adapter Module schematic .....	19
Figure 12. Pin-out for TC58128AFT .....	22
Figure 13. Memory Cell Layout .....	23
Figure 14. Read Mode(1) Timing Diagram .....	23
Figure 15. Flash Memory Block Diagram .....	24
Figure 16. Identify Initial Bad Blocks Flowchart .....	25
Figure 17. Program Flow Chart .....	26
Figure 18. Read Flow Chart .....	27
Figure 19. Erase Flow Chart .....	28
Figure 20. Pin-Out for MAX1874 .....	33
Figure 21. Pin-Out for MAX8621 .....	33
Figure 22. Pin-Out for MAX4836 .....	34
Figure 23. Storage module schematic .....	35
Figure 24. Storage module power circuitry .....	36
Figure 25. PCB Layout .....	38
Figure 26. Overall Software Program Under Normal Operation .....	40
Figure 27. Adapter Module SW Initialization Flowchart .....	43
Figure 28. Storage Module SW Initialization Flowchart .....	44
Figure 29. Overall Adapter Module Pseudo-Code .....	47
Figure 30. Overall Pseudo-Code .....	49
Figure 31. PIC Output during a Write to the SPI bus .....	54
Figure 32. PIC Output during a read from the SPI bus .....	54
Figure 33. Transmitter output during a read of the received data register .....	55
Figure 34. Measured Flash Timing of a Read Command .....	57
Figure 35. Implentation Project Schedule .....	62

List of Tables

Table 1. SPI Direction and Increment.....	12
Table 2. Power Amplifier Output Power Table .....	13
Table 3. Power Requirements .....	34
Table 4. Project Budget Sheet.....	58
Table 5. Implementation Semester Gantt Chart and Descriptions.....	61

## Abstract

The trend in wireless technology is moving to a universal connection protocol to allow for easy, automatic configuration of a variety of devices. There is a need for a more universally accepted user friendly method of wireless connectivity and data transfer. This project seeks to demonstrate the capabilities of the new wireless USB standard that will be used in future computer peripherals, consumer electronics, and an array of new wireless devices. The project consists of two modules which will transfer data wirelessly in the 2.4GHz frequency range. One module is used to interface with a PC while the other contains the flash memory used for data storage. The overall goal is to build a reliable storage device with low power consumption that is universally accepted on any PC running Windows 2000 or XP with a working range of up to 10 meters while minimizing interference with neighboring devices.

### Key Design Features:

- Wireless connectivity range of up to 10 meters
- Full-speed data transfer rates with the PC using the USB 2.0 standard
- Plug-and-play connection capability for ease of use
- Employs the universally accepted universal serial bus (USB) technology

## Introduction

The overall goal of this design is to create a working flash storage device capable of transferring data wirelessly using Universal Serial Bus.

### Statement of Need:

The trend in wireless technology is moving to a universal connection protocol to allow for easy automatic configuration of a variety of devices. For instance, a common need of many people is a more user-friendly method of data storage for transport between two or more computers. The specific goal of this project is to provide a practical wireless data storage device capable of being formatted using the FAT32 standard to enable file transfer on any computer running Windows 2000 or XP operating system. This is intended to demonstrate the capabilities of the USB standard that will become an essential technology in today's wireless lifestyle.

### Problem Definition:

#### Goals:

- Create two separate modules with the ability to transfer data between them wirelessly in the 2.4GHz frequency range.
- Write the embedded controller chip firmware required for Windows to detect and enumerate the device as a usable mass storage drive.
- Demonstrate the wireless effectiveness of the USB.
- Create a mass storage flash memory device recognized by Windows which will be controlled by the Storage Module firmware code running on the PIC18F4550.

## Wireless USB Project Design Report

### Objectives:

- Create the USB dongle adapter capable of being inserted in to a standard Windows 2000 or XP based PC USB port and responsible for effectively communicating read and write commands to the storage module microcontroller.
- Create a standalone storage device cable of transmitting and receiving data with the PC via the USB dongle adapter module.
- Effectively manage the power needs of the standalone storage device.
- Work reliably within a range of at least 2 meters.

### Constraints:

- Must comply with FCC regulations.
- Work within the power constraints of the portable device.
- Limit compatibility to the use of Windows 2000 or XP operating systems.
- Remain within the department provided budget.

## Design Requirements

### Wireless Communication Specifications:

- The communication protocol between the two separate modules must conform to the attributes and specifications, as stated below, of the Wireless USB set forth by the USB Implementers Forum.
  - The wired data transmission rates between the adapter PIC and the PC will approach 12Mb/s , allowing our wireless product to carry the same full-speed rate as a USB 2.0 under ideal circumstances. However, the actual radio transmission rates will be much lower at around 62kb/s within a distance of at least 10 feet.
  - Slower data rates shall also be supported where physical constraints or multiple wireless components prohibits the data rates of the full-speed USB 2.0 protocol.
  - The radio spectrum used shall be within the unlicensed industrial, scientific, and medical (ISM) frequency range of 2.4GHz to 2.483GHz.
- The Wireless USB module should be as easily configurable and universally accepted as its wired counterpart.
  - We will restrict our design to only being compatible with Windows 2000 and Windows XP due to time constraints.
  - Must have plug-and-play capabilities which will automatically install the needed driver for use on any Windows 2000 or XP machine.
- Power management specifications are defined separately for each of the two modules.



## Wireless USB Project Design Report

- The USB dongle adapter module for the PC will be powered by the computer's power supply by means of the 5VDC USB bus. The storage module will be powered by a 3.7VDC battery and the components operate under lower power constraints using 3.3V.
- Our product must also conform to applicable FCC regulations and should ensure minimal interference with other electronic devices in the close proximity of 10 meters.

### Hardware/Physical Specifications:

- The USB dongle adapter module must conform to the standard physical attributes of a typical USB port on any standard PC.
  - This module must not harm or interfere with the proper operation of the host PC. It must interface correctly with the computer by means of an automatically created Windows driver as defined by proper communication with the device firmware located on the microcontroller.
  - This module will implement the PIC18F4550 USB microcontroller along with the Cypress USB Transceiver.
- The storage module which will house the flash memory storage, a battery, the transceiver, and a controller PIC for transferring data wirelessly must be a completely standalone module, meaning that it must not be connected in any way to outside sources except through a wireless data transmission with the PCI adapter host. This will be powered by 3.7VDC Lithium-Polymer battery.

## Wireless USB Project Design Report

- This module should fit onto a PCB no larger than 10cm by 10cm by 2cm.

Using the discounted PCB design selected, the size of each module actually exceeded this original specification by a few centimeters at most.

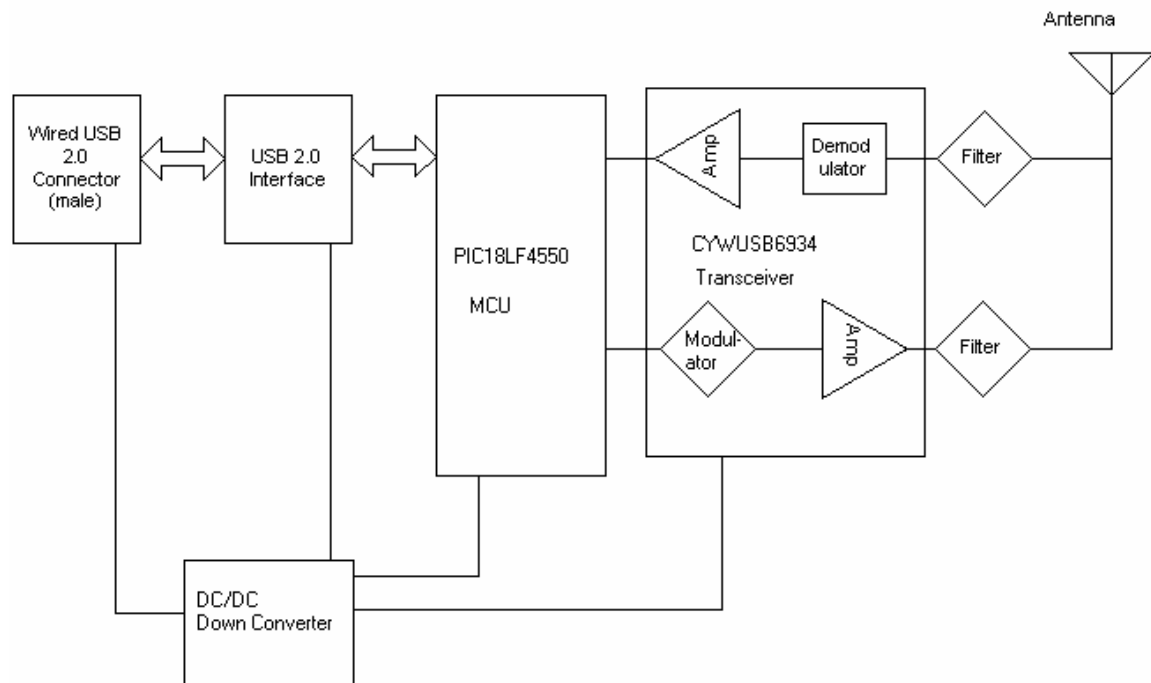
- This module will implement the PIC18F4550 USB microcontroller along with the Cypress USB Transceiver.

## Accepted Technical Design

### Design Basics and Explanation:

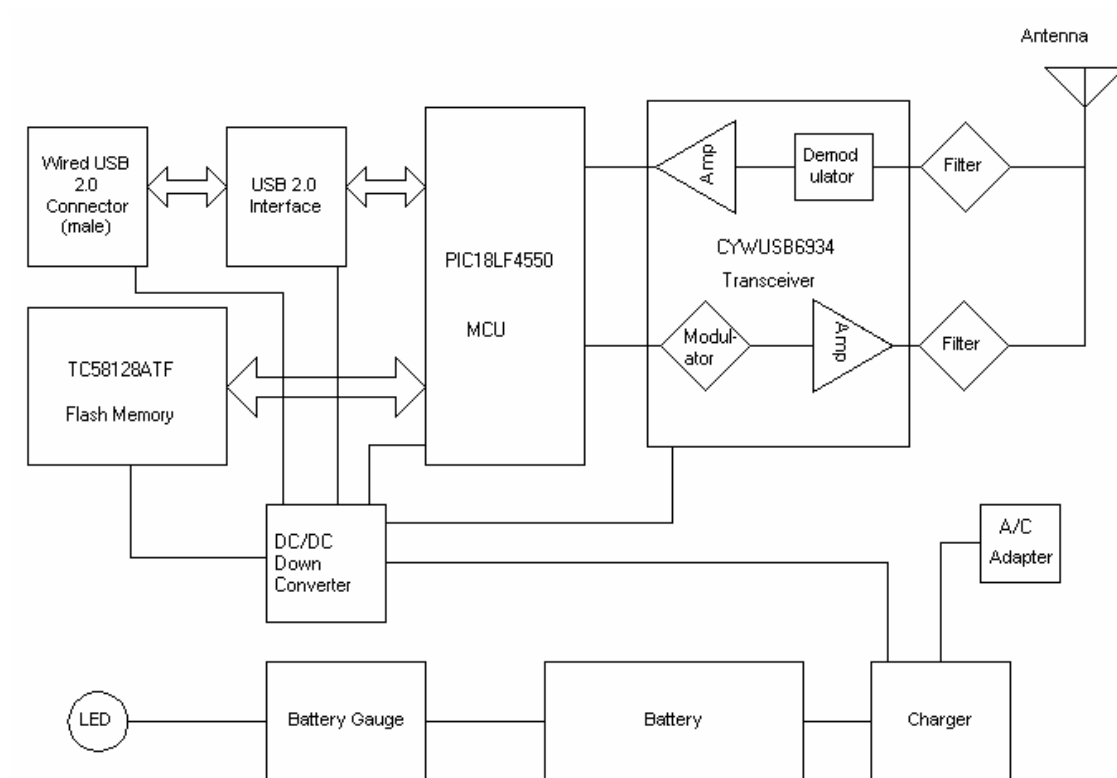
The accepted technical design was, in many ways, similar to the original design. With the exception of a PC interface through a USB dongle instead of a PCI card, and the substitution of a PIC based controller chip for a CMOS integrated circuit, it has many of the same features as the original.

The final accepted design includes two separate modules, the dongle-based adapter module for the interface to the PC and the stand-alone storage module, which includes the flash memory, chip to be used as a mass storage device. These two modules communicate wirelessly within a range of approximately two meters. Please refer to the block diagrams below for the adapter module and for the storage module in Figures 1 and 2 respectively.



**Figure 1. Adapter Module Block Diagram**

The adapter module contains the PIC18F4550 microcontroller and the Cypress wireless USB transceiver. Other components include the USB bus power step-down converter used to power the PIC and the transceiver, and the supporting circuitry needed for proper operation.



**Figure 2. Storage Module Block Diagram**

The storage module contains the low power tested PIC18LF4550 microcontroller along with the Cypress transceiver, which have both been verified to operate under a lower supply voltage of 3.3VDC. However, the storage module also contains the flash memory chip used as the non-volatile high-density mass storage media. It also includes the power

circuitry needed to completely power the components of the stand-alone storage module device.

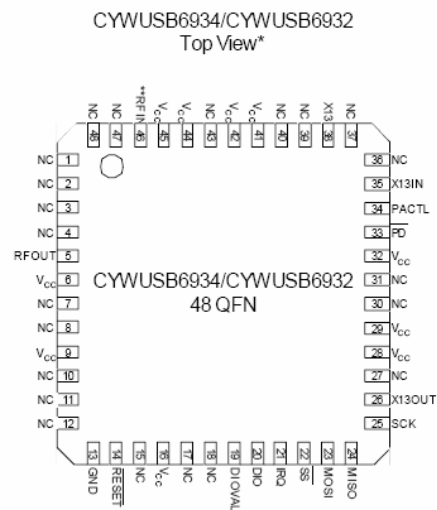
### Transceiver Selection and Communication Standard:

The Cypress CYWUSB6934 wireless transceiver is a Gaussian Frequency Shift Keying (GFSK) transceiver that operates in the Industrial Scientific and Medical (ISM) unregulated portion of the frequency spectrum. This transceiver will provide the wireless link between the adapter module and storage module. The CYWUSB6934 comes in a 48QFN configuration. The pin-out for this device is shown in Figure 3, which was copied from the documentation for the CYWUSB6934 transceiver. The transceiver operates at 3.3V with a nominal current of 3mA. This transceiver uses 7 I/O pins to connect to an application controller. The I/O pins will connect to the PIC18F4550 microcontroller on both modules. The data transfer uses a four wire Serial Peripheral Interface (SPI) with the PIC. The transceiver also has  $RF_{out}$  and  $RF_{in}$  pins that are connected to a PCB antenna through a filter, a block diagram of this device is shown as Figure 4. The filter design was based on the filter used in the CYWM6934 wireless transceiver module. A block diagram of CYWM6934, which is an example circuit designed by cypress, showing the filter is included as Figure 5. A majority of the pins on the transceiver are either not connected, which are actually tied to ground for this device, or are connected to  $V_{cc}$ , which is 3.3V.

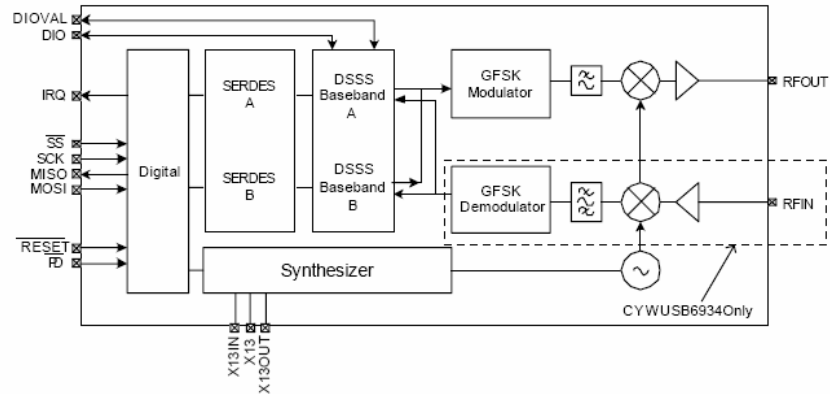
The pins that are connected to the PIC18F4550 are IRQ, nRESET, MOSI, nSS, SCK, MISO, and nPD. The 'n' before the pin name denotes an active low signal. The IRQ pin is the interrupt output from the transceiver to the application controller. RESET is an input that will initiate a reset of the transceiver. MOSI stands for Master Out, Slave

## Wireless USB Project Design Report

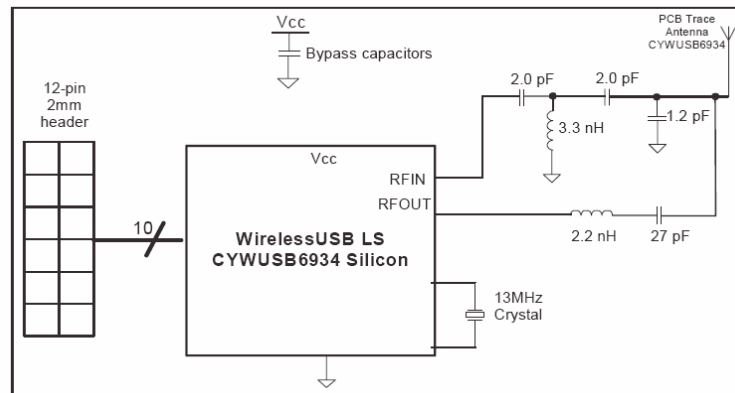
In. This is the data signal from the PIC to the transceiver module. SS is an active low slave select signal. This is an input signal from the PIC and it enables the SPI data transfer. It must be active through the entire data transfer. SCK is a clock input from the PIC that allows the data transfer to be synchronized properly. MISO stands for Master In, Slave Out. This pin is used to transfer data from the transceiver to the PIC. The last pin is the PD pin. This is the Power Down pin. It too is active low and when it is asserted the transceiver module will power itself down. The DIOVAL and DIO pins are for SERDES (Serializer/ Deserializer), which were not used in this project so they were left unconnected. The X13OUT, which is an output 13MHZ clock signal, is also not needed for this project and was disabled by writing to the crystal output control register.



**Figure 3. Pin-Out for the CYWUSB6934**



**Figure 4. Block Diagram of the CYWUSB6934**



**Figure 5. CYWUSB6934 Block Diagram**

The SPI data transfer is initiated by the PIC sending first an address byte and then the data. The data can either be a single byte at a time or in a burst fashion where all the data is transferred in multiple bytes until the transfer is finished. The address byte is the most critical part of the data transfer. There are two bits that determine the direction and the increment of the data transfer. The direction is dependent upon whether the transfer is a read or a write and is determined by bit 7 and the increment, whether it is a single byte being transferred or multiple bytes in succession, is set by bit 6. This is laid out in Table

1. The rest of the address byte is 6 bits that are the address for the actual data transfer.

Once the address byte has been sent the data is then transferred according to the procedure selected by bits 6 and 7. If the increment is set to 1 then the transceiver will automatically increment the address by one after each byte is sent.

**Table 1. SPI Direction and Increment**

Bit	7	6	
Bit name	Direction	Increment	Result
Value	0	0	Single Byte Read
Value	0	1	Burst Read
Value	1	0	Single Byte write
Value	1	1	Burst write

The output power can also be controlled on this transceiver. There are two ways to control the power first is using the PACTL pin (34) on the transceiver the other is by setting the register for output power. The power is changed to one of 8 settings corresponding to two bits for the register. The corresponding output powers are listed in Table 2. The default power amplifier setting is '0' which is  $-29.0\text{dBm}$ , the recommended setting is to use the PA setting of 7 unless lower power is needed, this sets the output power to  $0\text{ dBm}$ . The power control method used was the software control using the PIC firmware. The transceiver was designed in compliance with FCC regulations and also is compliant with European, Canadian and Japanese regulations. With a minimum receiver sensitivity of  $-90\text{dBm}$  the transceiver pair will be able to communicate at a range of about 10 meters.



**Table 2. Power Amplifier Output Power Table**

PA Setting	Typical Output Power (dBm)
7	0
6	-2.4
5	-5.6
4	-9.7
3	-16.4
2	-20.8
1	-24.8
0	-29.0

### Microcontroller Selection:

At the heart of the accepted design is the PIC18F4550 Microcontroller from Microchip Technology Inc. The functional block diagram of the PIC18F4550 is shown in Figure 6 below. Note the large number of I/O pin options configurable through any of the external ports (Ports A through E) and the four internal timer registers that can be used for various interrupt routines. The I/O pin layout for the PIC is shown below in Figure 7. The PIC18F4550 is designed for use with USB 2.0 compatible peripherals and has many features to support the USB 2.0 standard in full-speed mode. The USB provides a serial interface engine for communication with external USB transceivers as shown in Figure 8, also given below. The PIC18F4550 is capable of using its own internal transceiver or communicating through the external serial link, which we will be using to communicate through the Cypress transceivers. In the original design there was an external USB transceiver used, this was eliminated and the internal USB transceiver on the PIC was

## Wireless USB Project Design Report

used instead. This was done to keep more of the data ports on the PIC fully intact and thus making it easier to connect the transceiver and the flash.

FIGURE 1-2: PIC18F4455/4550 (40/44-PIN) BLOCK DIAGRAM

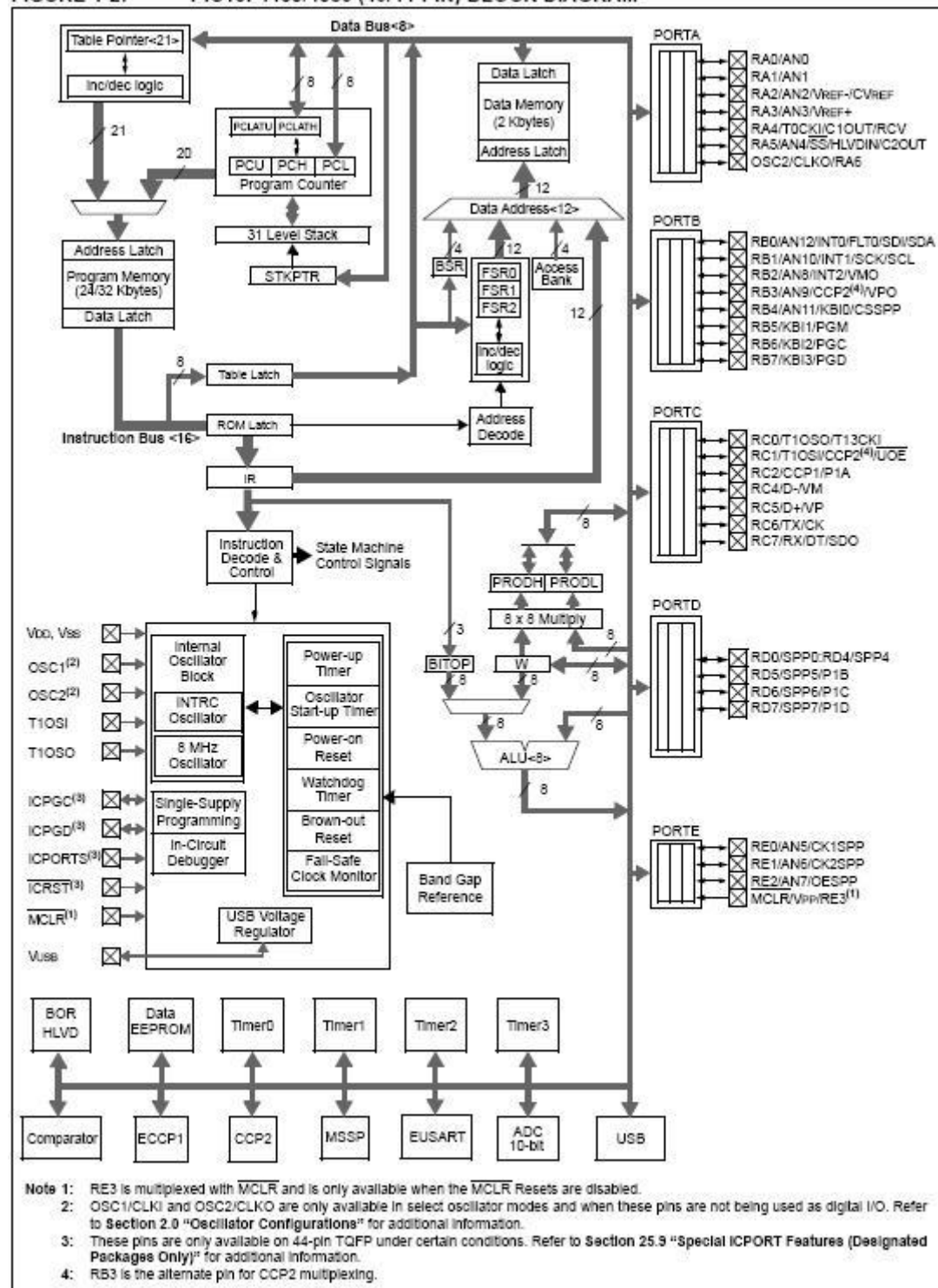


Figure 6. The Functional Block Diagram of the PIC18F4550

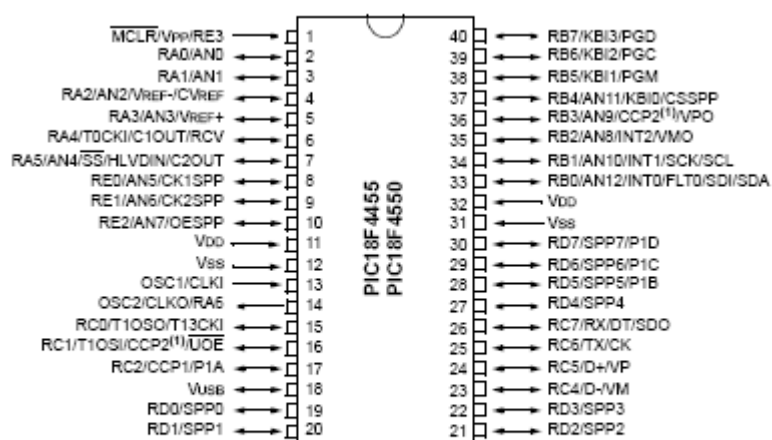


Figure 7. I/O Pin Assignments of the PIC18F4455

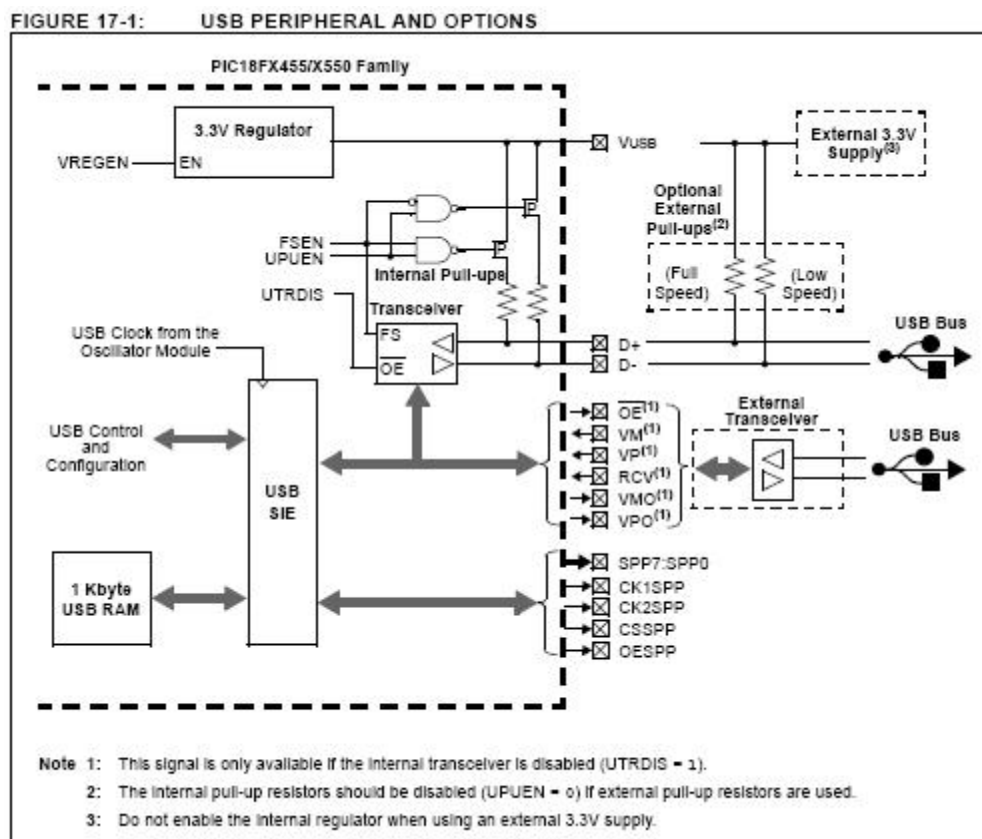
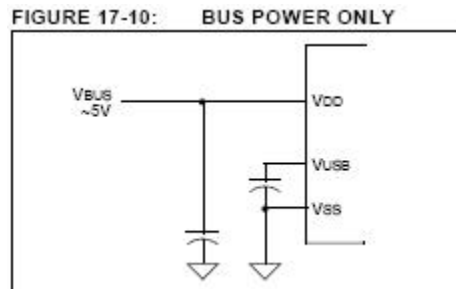


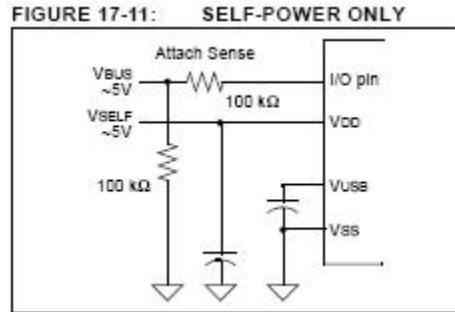
Figure 8. The PIC USB External Interface

The PIC18F4550 was chosen for its versatility, low cost and ease of implementation. It has 2048 bytes of RAM, is capable of transmitting at 12Mbits/s as required for full speed USB operation, has 1024 bytes available for USB buffer, and runs at a max speed of 48MHz. Although the low power version of this same PIC (the PIC18LF4550) will be used for the storage module, no distinction will be made between the two chips because they are essentially the same die and are functionally the same chip. The difference is that the low power chip is specifically tested for lower power inputs.

The power connections are shown below in Figure 9 and 10. Figure 9 shows the PIC power connections for the adapter module where its  $V_{DD}$  input takes the power from the 5V line of the USB bus itself. Figure 10 shows the self-powered connection scheme where the PIC  $V_{DD}$  port would take its 5V input voltage from a DC power source. This was one of the changes we made during the implementation process. Because the PCBs were designed and ordered together for the discounted price, both boards provide a 3.3V supply to PIC. However, no change was needed because both the low power version and the regular version of the PIC were testing to operate correctly at 3.3V.



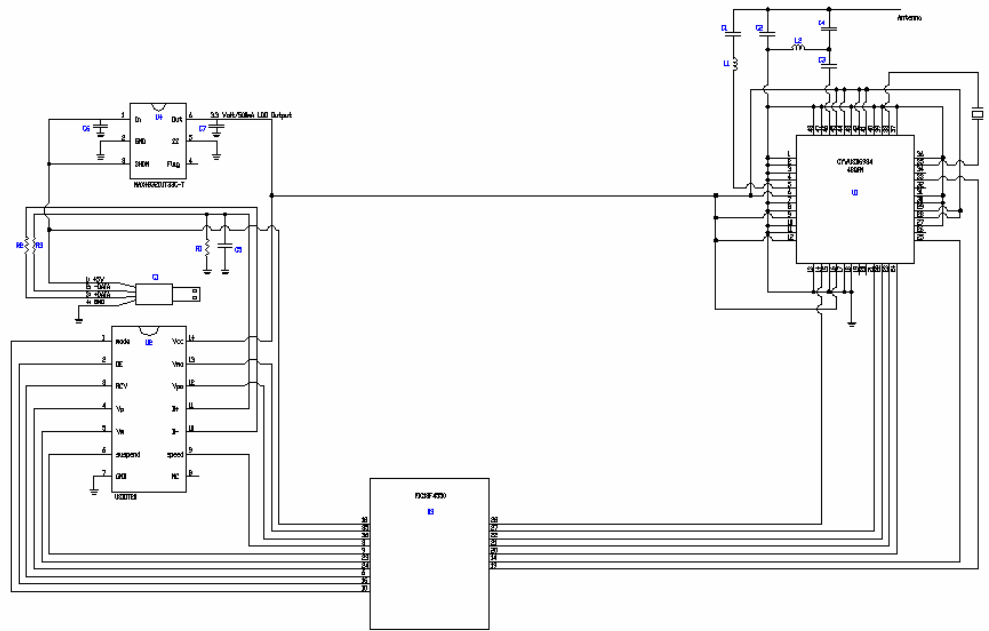
**Figure 9. Adapter PIC Power Connections**



**Figure 10. Storage PIC Power Connections**

The PIC also supports many commands useful to the mass storage device peripheral USB operation. It provides for various data transfer types that are required for this type of device. For USB 2.0, command, interrupt, and bulk transfer will be used to transfer data with the USB Host on the PC. Bulk and interrupt transfers will be of special importance to the device due to the need to transfer a large number of data packets at a single time, and the need for interrupts to handle user interaction and system level computation trade-offs.

The final schematic for the adapter module is included as Figure 11. This schematic shows Adapter module that will be connected to the PC via the wired USB connection. The PIC18F4550 microcontroller is the main component that controls this module. The Cypress CYWUSB6934 wireless transceiver, which provides the link between the two modules, is shown on the right of the schematic. Refer to Appendix A for the list of components in the schematic represented in Figure 11.



**Figure 11. Adapter Module schematic**

Flash Memory:

Flash memory is electrically updateable (stored data can be erased and replaced under system processor control), high density, and non-volatile (retains data stored to it when powered off). It is used to provide the mass storage on the storage module. The Toshiba TC58128AFT Flash Memory chosen is a single 3.3V 128M-bit NAND Flash Memory. It comes in a TSOPI48 package with the pin-out shown in Figure 12. Figure 15 shows a block diagram of the Flash IC. The Flash uses an eight pin I/O data bus, six logic input pins, one output pin that shows if the device is ready or busy, and power supply pins.

The basic concept of how a Flash memory cell works is rather simple. Storing electrons on a floating gate changes the stored cell data from a one to a zero. The memory array of the Toshiba TC58128AFT is organized in 1024 blocks where each block contains 32 pages. Each page is split into a main area with two half pages of 256

bytes each that are used to store the data, and a spare area of 16 bytes that is used to store Error Correction Codes and Bad Block identification. A visual representation of this cell layout is shown in Figure 13.

There are nine possible operations for the flash. They are Serial Data Input, Read mode 1, Read mode 2, Read Mode 3, Reset, Auto Program, Auto Block Erase, Status Read, and ID Read.

Performing any operation involves precisely timed high and low inputs on the 6 logic input pins, in combination with commands sent over the eight I/O pins at there appropriate time in correspondence to the state of the logic pins. Data is then be read or written over the eight I/O pins. Pages of detailed timing diagrams are supplied in the Toshiba Flash data sheets to perform the operations and possible variations of the operations. For lack of space required to cover all these operations in detail, only the read mode 1 will be covered. Figure 14 shows the timing diagram for a read mode(1), which is a read that starts in the first half of a page.

Referencing Figure 14, first the Command Latch Enable (CLE) is driven high to load a command, and the Chip Enable (CE) is driven low to bring the Flash out of Standby. The 00H command is then written over the I/O pins and the Write Enable (WE) is cycled, because the Flash reads the command on its rising edge. Next the Address Latch Enable (ALE) is driven high, and three values specifying a starting address are entered over the I/O pins over three cycles of the Write Enable (WE) pin. Now the ALE pin is driven low again and the Read Enable (RE) pin is cycled. Data is available to read 35nS after a falling edge on the RE pin. The RE pin can be cycled as many times as needed to read the desired amount of data. The other operations follow a similar scheme.



The Microcontroller in addition to performing an operation must verify that the operation was successful. Figures 16 through 19 show the flow charts the software follows when performing an operation.

Figure 16 shows the Identifying Initial Bad Blocks flowchart. NAND Flash typically contains a number of bad blocks from the factory. The Flash memory is installed in to the system and the routine shown in Figure 16 is run by the microprocessor to record the locations of the bad blocks so they can be recorded and managed.

The flow chart labeled as Figure 17 shows the software logic involved with programming the Flash memory. The Page program command is written, the starting address specified, the desired amount of data is written, and the program command is disabled. Then the microcontroller verifies the data by reading the status register and determines if the program was successfully written.

Similarly the flow chart labeled as Figure 18 shows the software logic involved with reading the Flash memory, and the flow chart labeled as Figure 19 shows the software logic involved with erasing the Flash memory.

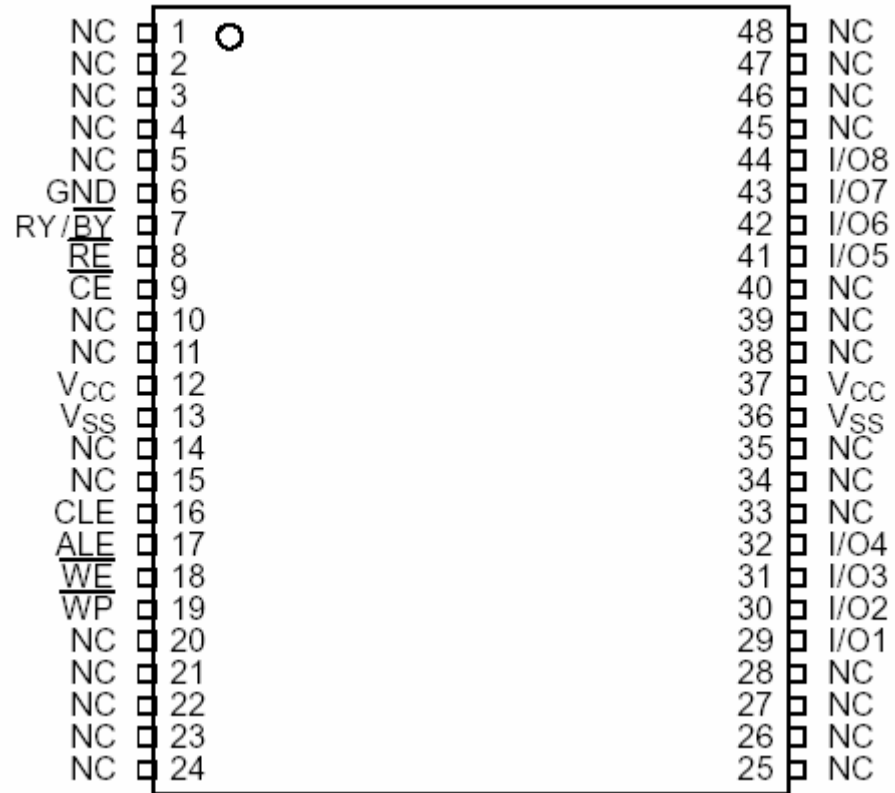


Figure 12. Pin-out for TC58128AFT

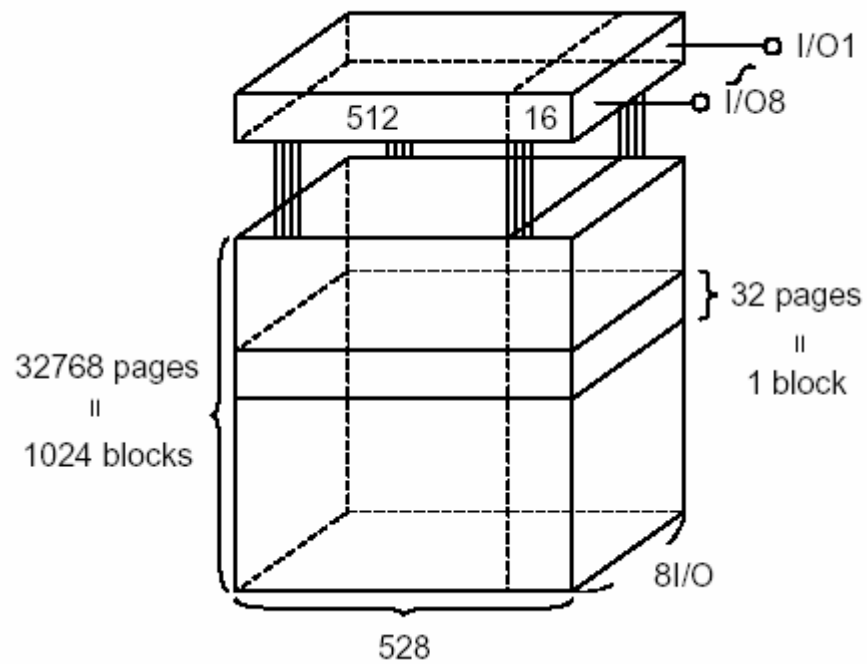


Figure 13. Memory Cell Layout

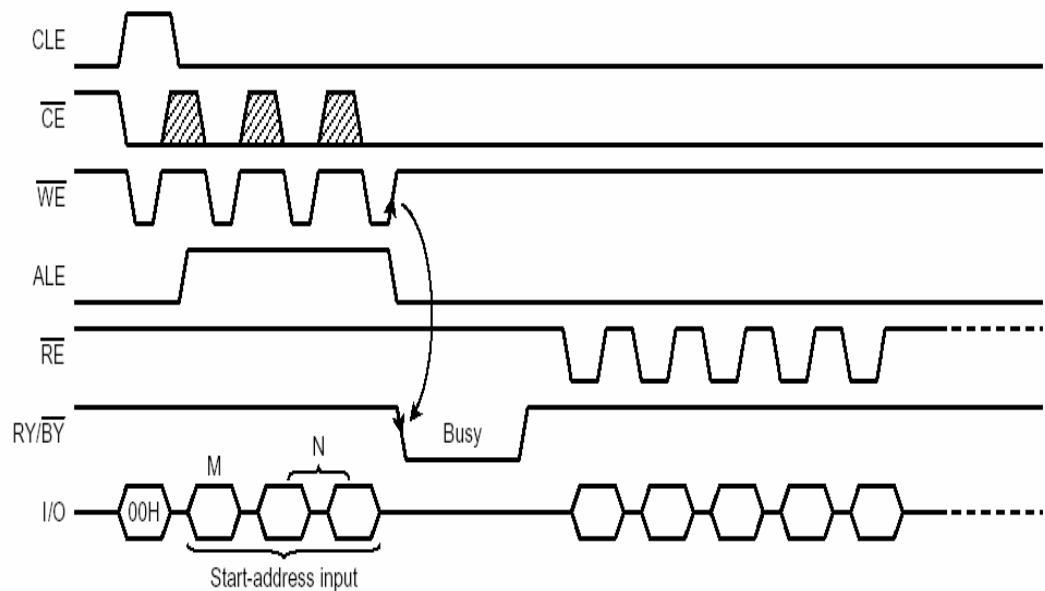
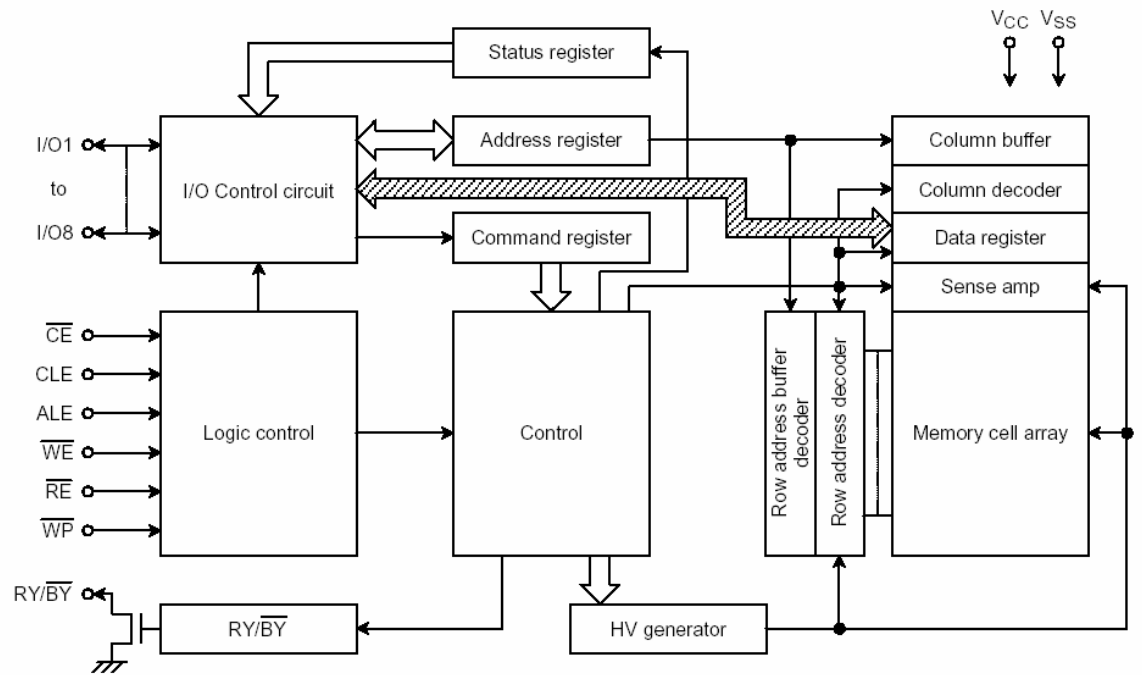
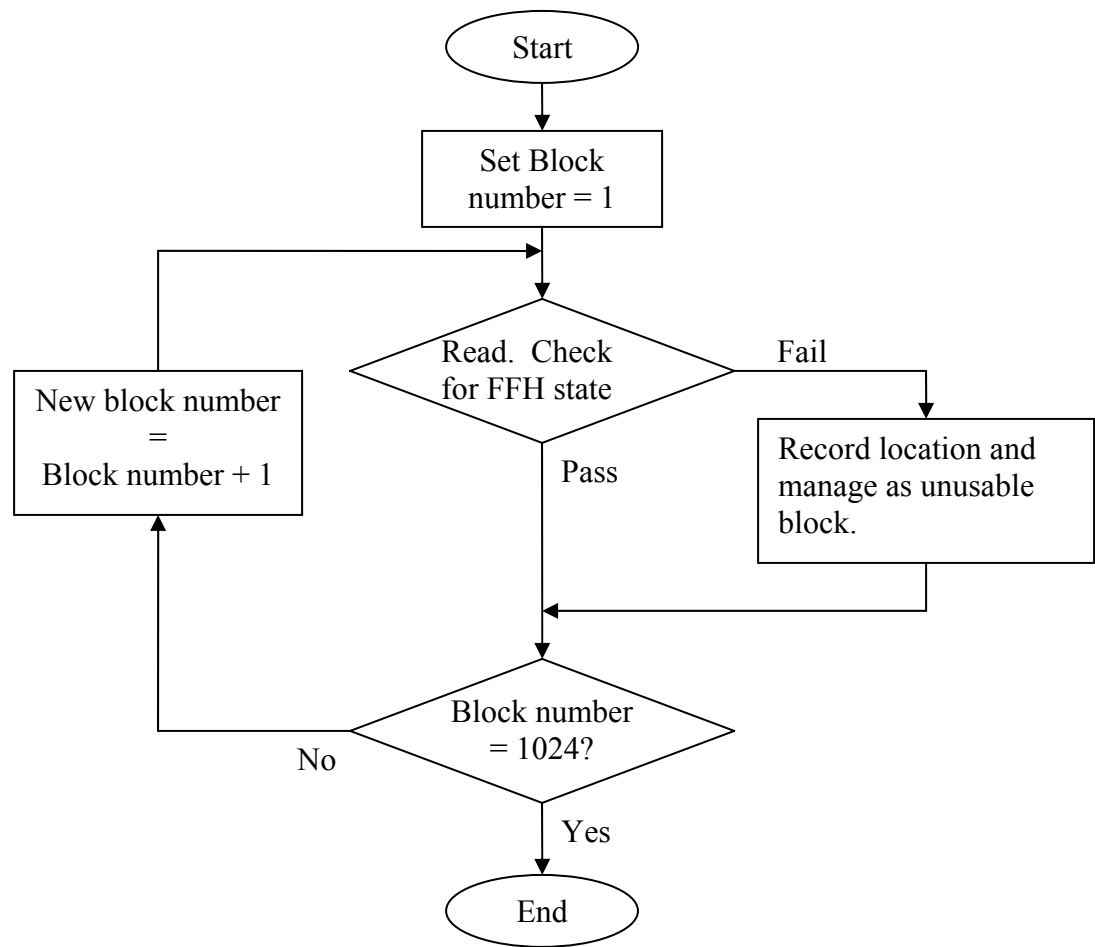


Figure 14. Read Mode(1) Timing Diagram



**Figure 15. Flash Memory Block Diagram**



**Figure 16. Identify Initial Bad Blocks Flowchart**

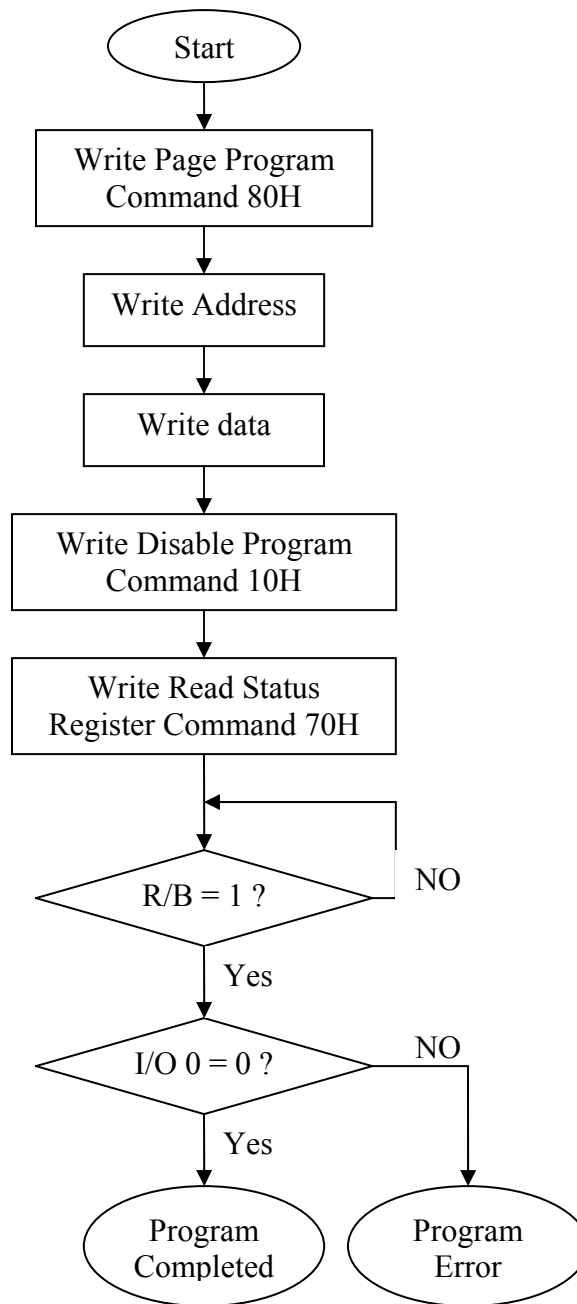
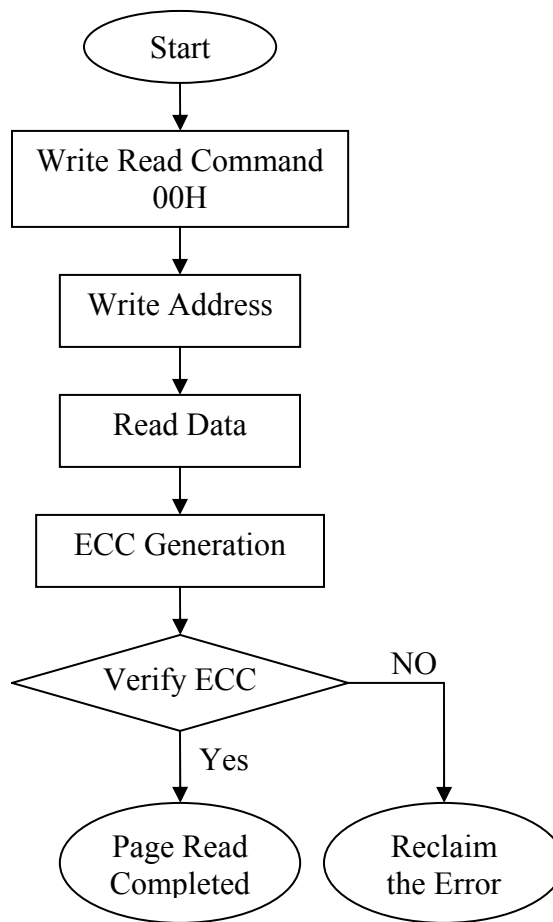
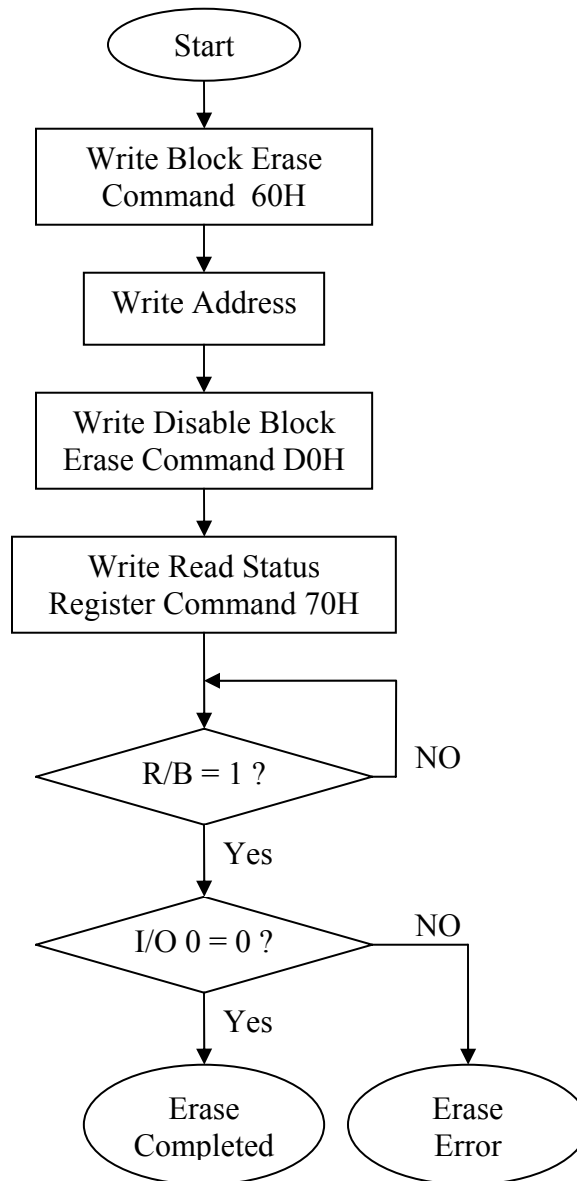


Figure 17. Program Flow Chart



**Figure 18. Read Flow Chart**



**Figure 19. Erase Flow Chart**

The Flash memory section of the project was not implemented in to the final system due to time restraints. Code is written to implement Read, Read Electronic



Signature, Page Program, and Block Erase. The code was successfully compiled. To test the Flash on the Bench it was installed in to the Bread Board and connected to the PIC. The Read Electronic Signature routine was performed. This command reads the part description from the Flash as two 8 bit numbers over two read cycles. Figure 1 is a oscilloscope plot of the command bus input from the PIC microcontroller.

What was not finished was the implementation of Error Detection and Correction. Flash memory has flawed gates from manufacturing and will develop addition flawed gates as the Flash memory is used. It is recommended, but not needed to implement an Error detection scheme for this.

In addition it was assumed that the Flash memory would always be written from the start of a page. This assumption was made due to the way that the FAT file system formats a storage device being similar to the layout of the Flash memory array. The actual way in which Windows would format the device is not entirely clear. If page programs are needed that start in places other than the start of a page, then additional code would need to be written for this, that would include provisions for a starting-byte-in-the-page variable (instead of assuming it to be 0) and the use of the Program B command that is used when programming in the second half of the page.

With the exception of the unknowns when combining with the system, the Flash Memory as a stand alone device was relatively easy to work with. Some mistakes were made in the coding, but walking through the code with the use of breakpoints in the code and the oscilloscope the errors were found and corrected.

## Wireless USB Project Design Report

### Power Management and Calculations:

The power management circuit of the portable flash drive is required to provide 3.3V power to the flash memory and microprocessor when plugged directly in to a USB port, or provide power to the flash memory, microprocessor, and transceiver when used wirelessly. The power management circuit is built around two IC chips, the MAX1874 and the MAX8621, both manufactured by the Maxim division of Dallas Semiconductor.

The function of the MAX1874 is to control the charging current from a USB port or an AC adapter to the Lithium Ion battery based on the charging limitations of the battery, thermal conditions, and available USB bus current conditions. The pin-out of the device is shown in Figure 20.

The temperature of the battery is monitored by a thermistor and input to the THRM input. If battery temperature becomes high the charging current is automatically set lower. Maximum charging current is set by a voltage divider resistor pair off the DCI input to the 658mA restriction of the Panasonic battery. The circuit is initially set to power off a 100mA USB port. During USB enumeration the microcontroller requests the maximum 500mA bus current. If the requested current is met the Microcontroller then drives the USEL input high to enable 500mA charging. If the request is denied the USEL input is left low for 100mA charging.

External circuitry on the MAX1874 provides circuit protection and additional options. A MOSFET on the input side provides over-voltage protection from, for instance using an incorrect AC adapter. The MOSFET's on the output side disconnects the load from the battery and provides power directly from the USB or AC adapter. The

advantage to this is that in the case that the battery becomes completely discharged, the battery fails, or there is no battery, the system can still operate immediately if plugged in to an USB or AC adapter, rather than having to wait for the battery to charge, or not being capable of operating at all. The four diodes prevent reverse feeding to the USB and AC inputs.

The Bi-color LED is used as a charging gauge. It is powered red when plugged in to a USB or AC adapter power source through a MOSFET. When charging is complete the CHG output from the MAX1874 is driven high and the red LED is turned off and the green LED is turned on through a pair of MOSFET's.

The MAX 8621 chip provides 2 switching converters, and 4 low-dropout linear regulators. The pin-out of the device is shown in figure 21. Switching converter voltages are set to 3.3V using an external resistor and inductor combination to set an input voltage at the FB feedback pin. LDO voltages are set to 1 of 18 voltage combinations through the SEL1 and SEL2 input pins by driving high, floating, or grounding. The switching converters are high efficiency but have some noise due to the on-off switching action of internal MOSFET's. They are used to power the flash memory and microcontroller which can tolerate the noise levels. The LDO's have lower efficiency, but also very low noise, due to the internal MOSFETs operating in the triode region as variable resistors. A LDO is used to power the transceiver which is susceptible to noise. The LDO's are also turned on or off by the microprocessor through the EN input pins. The MAX 8621 also provides a low power warning output that goes low when output voltage drops below 87% of regulation. This is input in to the microcontroller so that the system can safely

power down. The unused converters can be used in the future for additional features considered in the alternative designs.

The Power Management circuit of the adapter module consists of a single 500 mA MAX4836 LDO Regulator to convert the incoming 5V's from the USB port to a preset-at-order 3.3V to supply the USB controller, the transceiver, and the microcontroller. The pin-out of the device is shown in figure 22. This circuit requires the adapter module to be connected to a 500mA powered USB port.

Table 4 shows the details of the estimated power required by the modules. It was calculated that the adapter module should operate within the power supplied limits of the USB port. It was calculated that the storage module will operate for nearly 3 hours of data transfers, and over 142 hours in standby, more than long enough to make this a practical device.

Both small decoupling capacitors and larger bypass capacitors are used for the reliable operation of components throughout the entire circuit. In general capacitors smooth out effects of AC transients on the DC supply voltages by supplying excess charge when the voltage drops below the DC level and shunting off excess voltage spikes. Decoupling capacitors are physically located as close to the input pin they are filtering as possible, and connected between the particular voltage supply and ground. If capacitor values are not specified by the manufacturer of the IC, decoupling capacitor values are calculated using Formula 1 as given below with worst case conditions being assumed. The schematics for the storage module and the power circuitry for the storage module are included as Figures 23 and 24 respectively.

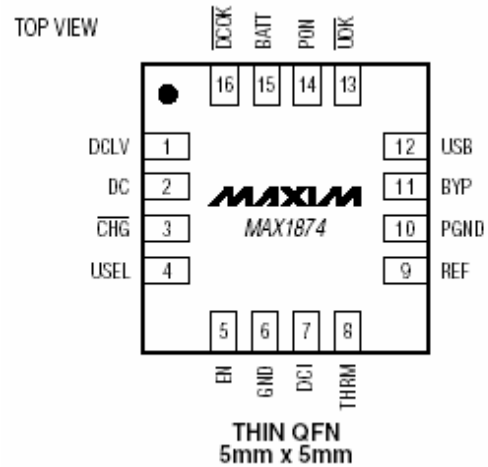


Figure 20. Pin-Out for MAX1874

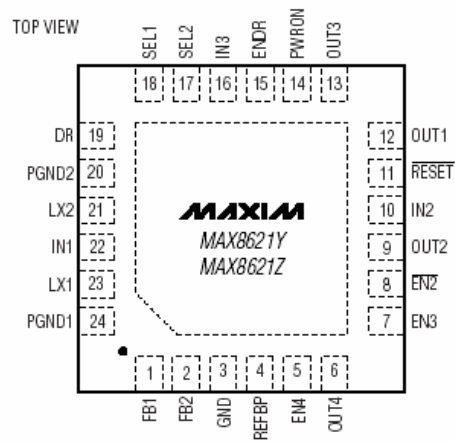


Figure 21. Pin-Out for MAX8621

TOP VIEW

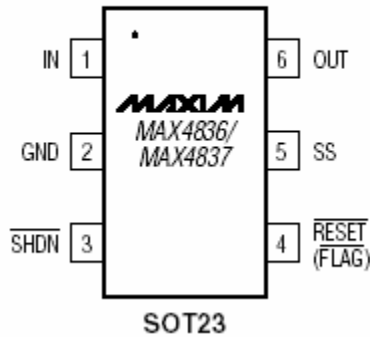


Figure 22. Pin-Out for MAX4836

Table 3. Power Requirements

	Standby	Operating
Flash Memory	$100\mu\text{A}@3.3\text{V}$	$10\text{mA}@3.3\text{V}$
Tranceiver	$3\text{mA}@3.3\text{V}$	$69\text{mA}@3.3\text{V}$
USB Controller	$2\text{mA}@3.3\text{V}$	$30\text{mA}@3.3\text{V}$
Microcontroller	$1\text{mA}@3.3\text{V}$	$200\text{mA}@3.3\text{V}$
Total	$6.1\text{mA}@3.3\text{V}$	$309\text{mA}@3.3\text{V}$

Operating Power Consumption:

$$(309\text{mA})(3.3\text{V}) / (85\% \text{ total efficiency of power management circuit}) = 1.20\text{W}$$

Standby Power Consumption:

$$(6.1\text{mA})(3.3\text{V}) / (85\% \text{ total efficiency of power management circuit}) = 23.7\text{mW}$$

Battery Power:

$$940\text{mAh} @3.6\text{V} = 3.384\text{Wh}$$

$$\text{System life with active data transfers} = 3.384\text{Wh} / 1.20\text{W} = 2\text{hr } 49\text{min}$$

$$\text{Life in standby} = 3.384\text{Wh} / 23.7\text{mW} = 142\text{hr. } 46\text{min.}$$

Formula 1. Decoupling Capacitor Selection Example.

## Wireless USB Project Design Report

$$I_{CC}(\text{peak}) = (60\text{mA}) * (I)$$
$$\text{Max ripple Voltage} = (0.2V_{\text{peak-peak}})(0.1V \text{ dv})$$
$$\text{Switching time} = 20\text{ns dt}$$

$$C = I (dt/dv)$$
$$C = (60\text{mA} * 20\text{ns}) / 0.1V$$
$$C = 12\text{nF}$$
$$C = 12\text{nF} * (4x \text{ margin}) = 0.047\mu\text{F}$$

The schematics for the storage module and the power circuitry for the storage module are included as Figures 24 and 25 respectively. Figure 24 shows a similar schematic to the adapter module only with the addition of the flash memory. Once again all of the components are controlled by the PIC18F4550. Figure 25 shows all of the power management for the storage module. This includes the battery, and charger as well as the AC adapter, which allows this to be charged off of either the wall or the USB port. Please refer to Appendix A for the list of components as designated in Figure 24 and Figure 25.

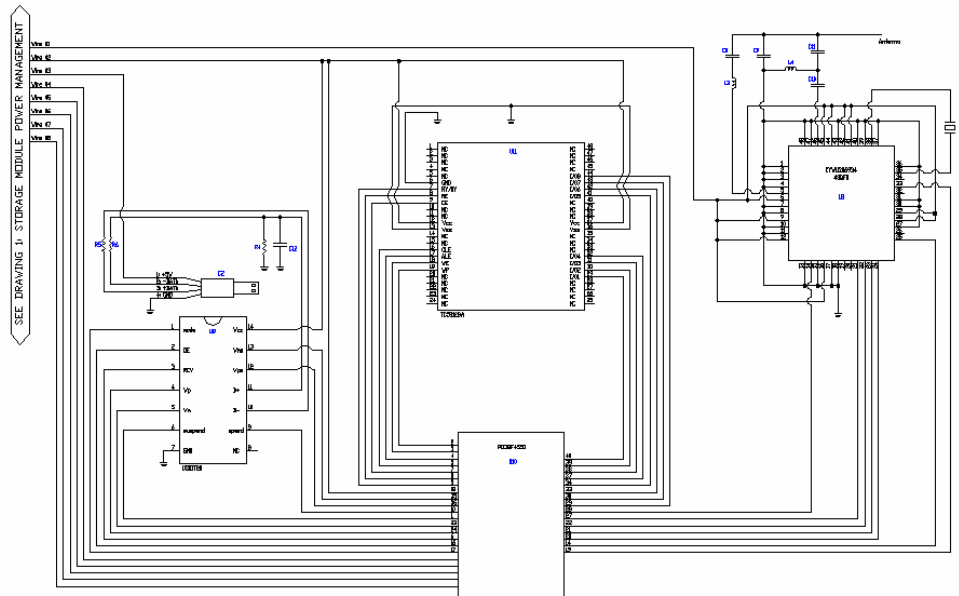
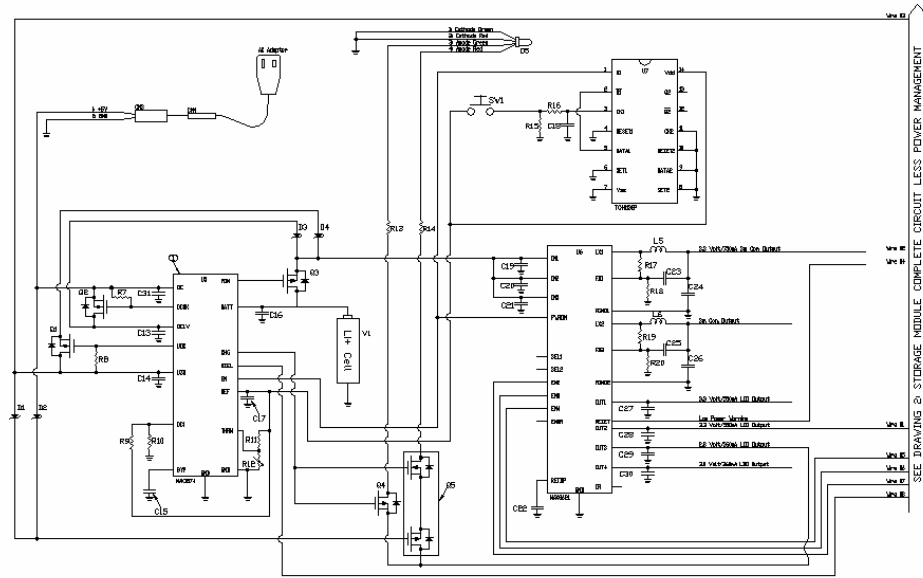


Figure 23. Storage module schematic



**Figure 24. Storage module power circuitry**

The power management section of the design project remained largely unchanged. The changes that were made include the battery type, use of voltage protection MOSFETs, and changes in the resistor values for the de-bounce circuit for the momentary push button.

The battery type was changed from a Lithium Ion battery to a Lithium Polymer battery. The reason for this change was strictly due to the availability of the Lithium Ion battery. Finding a company to sell our design group a Lithium Ion battery that is not designed for an existing electronic device proved to be difficult. A supplier for the Lithium Polymer battery was found though. The Lithium Polymer battery is a similar chemistry to the Lithium Ion battery and did not effect the circuit. The use of the Lithium Ion charger was approved by Maxim Semiconductor to charge the Lithium Polymer battery. It was also of similar capacity.



## Wireless USB Project Design Report

In the original design the use of power MOSFETs on the USB and AC adapters charging inputs were used. There were mistakes made with the PCB layout, and the components affected were surface mount type, so these options had to be abandoned. The result is that the charging inputs are only protected to 6.5V in comparison to 18V using the MOSFETs.

Changes in the resistor values on the debounce RC circuit on the momentary push button were simply due to the time that the push button needed to be depressed was too great. This small miscalculation was due to the logic high on the latching flip-flop being higher than the voltage reached after 1 time constant of the RC circuit. Smaller resistor values were simply tested until the desired delay was achieved.

In order to implement the project hardware easier printed circuit boards were designed and purchased. The layout of these circuit boards was the same as the schematics pictured above. As a way of reducing the cost the connections between the PIC and the transceivers were changed from the original design so they would be the same on both modules, this was done for two reasons. First as was previously mentioned cost, with both modules using the same layout between the transceiver and the PIC one board could be used as a common board with a simpler power circuit used on a proto-board on top of the circuit board for the adapter module since it will receive its power from the USB port. The second important reason for this change was now the software for the interface between the PIC and the wireless transceivers would be exactly the same and the same code can be run on both modules for this aspect of the project. Figure 25 shows the layout of the PCB.

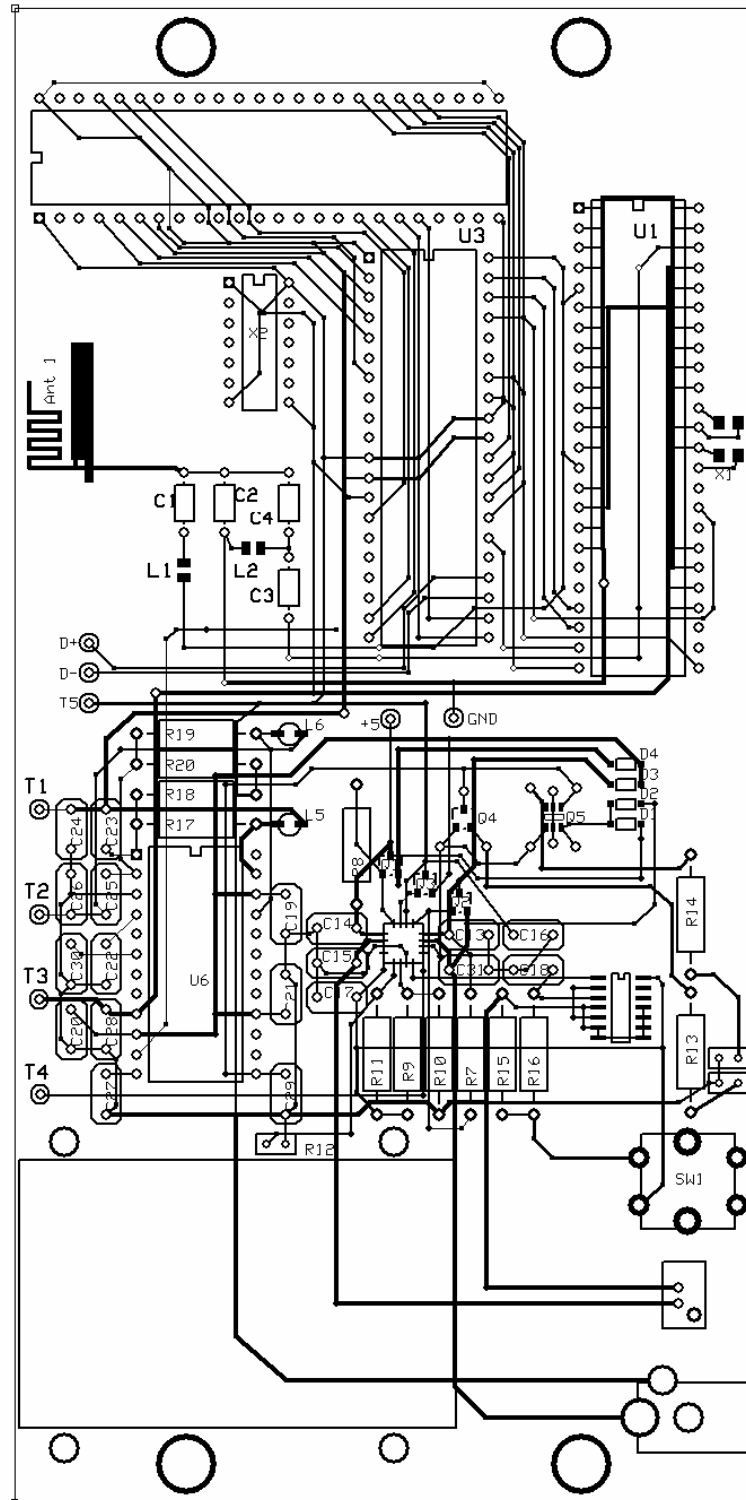


Figure 25. PCB Layout

### Software Design:

A large majority of the implementation of this project came in the form of microcontroller software development. The embedded firmware code for the PIC microcontroller was written in C code. Because of the differences in the inherent function between the adapter controller PIC and the controller chip dedicated for the flash storage module side, there is a need for two separate bodies of code. To get an idea of the function of the software flow under normal operation, refer to the diagram below labeled Figure 26.

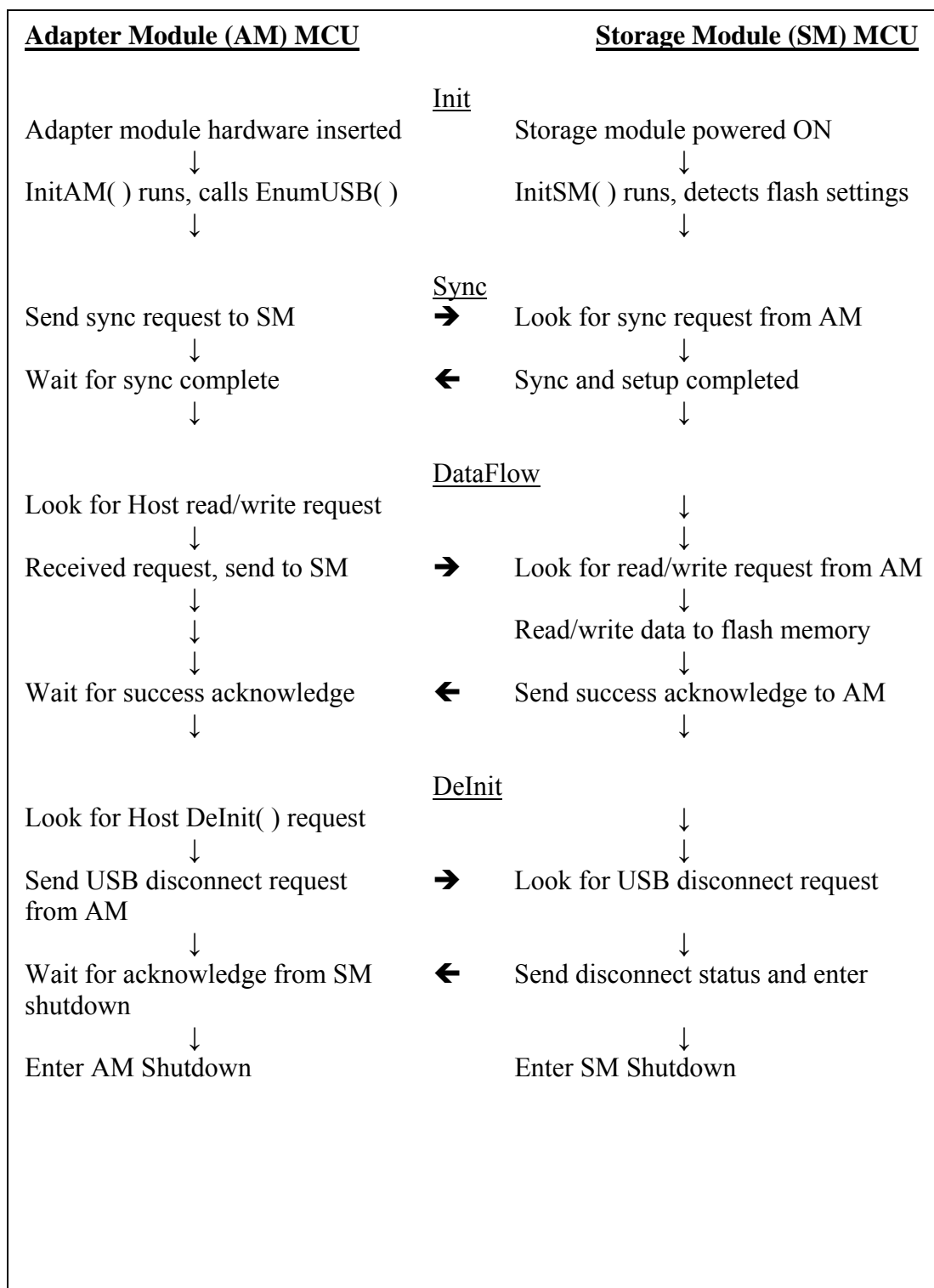


Figure 26. Overall Software Program Under Normal Operation

This shows how the code on each of the two controllers should run basically in parallel until the synchronization between them occurs. First, and most importantly, the device must answer configuration and descriptor requests from the host PC. This will require the PIC on the adapter side to go through a series of host request and command transfers. The USB host must assign a unique endpoint address to the device, it must determine the type of device that it is (in this case it is classified as a mass storage device), it must determine the data transfer types that will be used, and finally the speed at which it operates. These are the most important of the many device descriptors that are sent to the host.

Once it has established its initial communication with the host, it must then communicate with the PIC on the storage module side. Only after it receives information about how much memory is available from the storage PIC, can the adapter finish its PIC-to-PIC setup routine. The synchronization between the two PICs must then be completed. The adapter PIC will then give the storage PIC the USB enumeration acknowledgement and then the two modules are ready to exchange read/write data. Then the program will enter an infinite loop which will look for read/write requests from the host, check for de-initialization or disconnect requests from the host, and finally, check for loss of connection by physical disconnect or by power failure on the storage side. The adapter module is fully responsible for all interactions with the USB Host. The storage module is responsible only for responding to requests from the adapter PIC and does not directly deal with the USB Host enumeration or communication.

The most important part of the code is the setup and USB configuration which is required to correctly communicate with the USB Host. The initial setup routine for the

## Wireless USB Project Design Report

PIC controllers on each of the modules is listed below in flowchart form. Figure 27 shows the adapter module initialization flow diagram. Figure 28 shows the storage module initialization flow diagram.

# Wireless USB Project Design Report

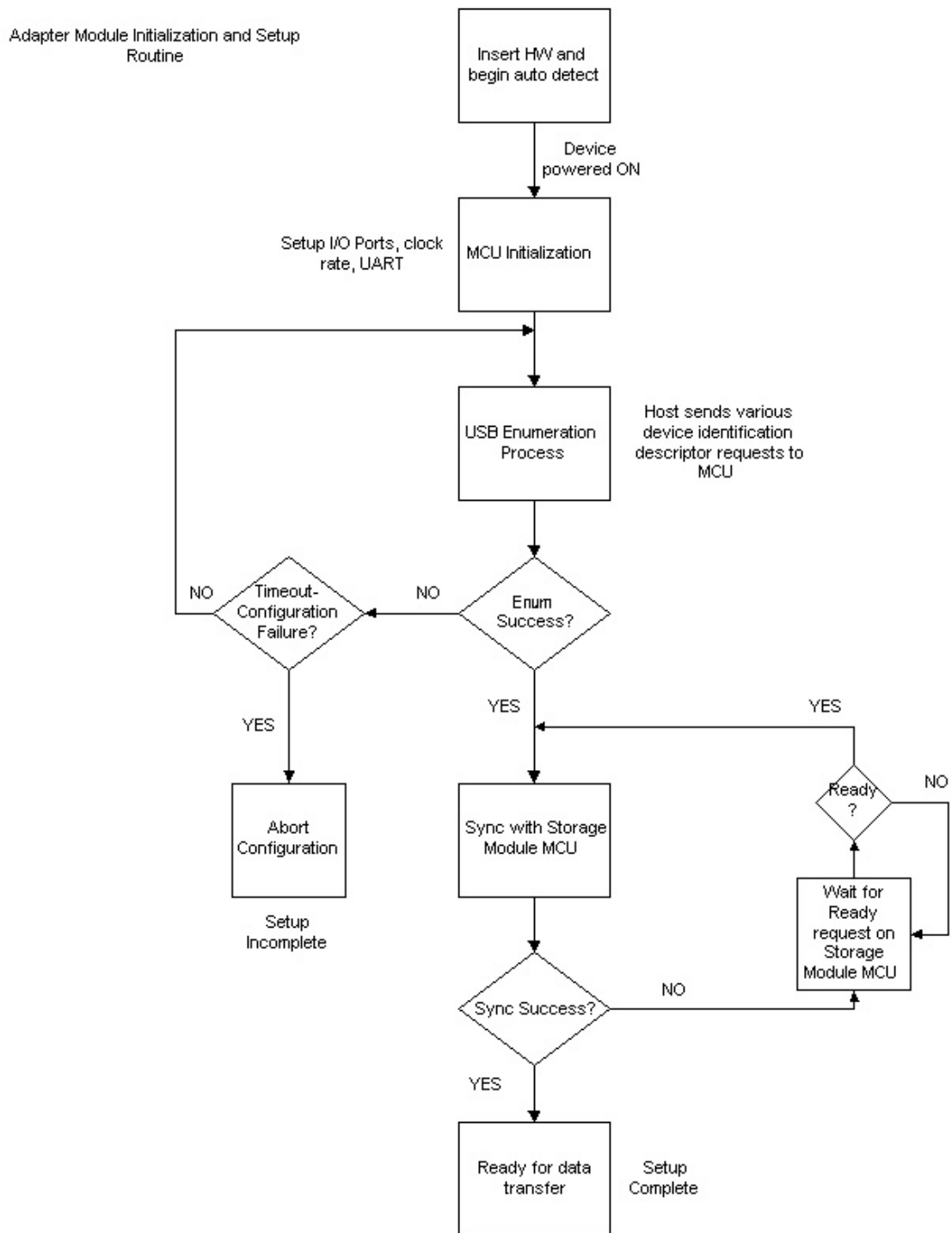


Figure 27. Adapter Module SW Initialization Flowchart

Storage Module Initialization and Setup Routine

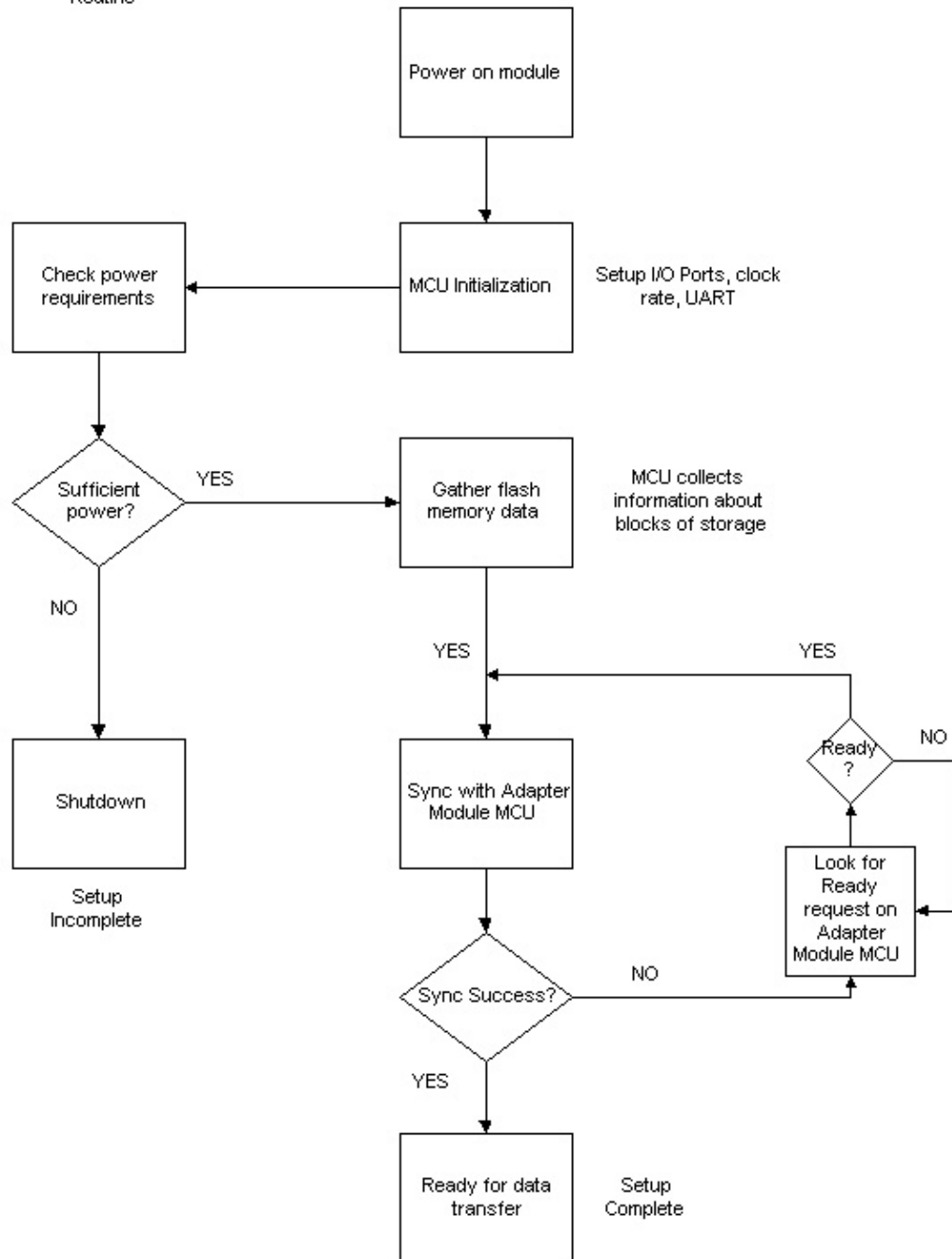


Figure 28. Storage Module SW Initialization Flowchart



## Wireless USB Project Design Report

As you can see from the figures above, each must setup the MCU on power up. The adapter controller then must begin to automatically begin handling device descriptor requests from the USB Host. It is intended that the adapter module will be always connected to the USB port of the PC and that it will always be on so long as the PC is on and operating under normal conditions. The handshake between the two PICs will occur when the storage module comes within range, up to 10m, and is powered on. When powered on, the storage controller will gather data about the number of empty blocks of memory available for use within the flash device and provide this information to the adapter module as requested. Once all of the responses are received by the adapter PIC, it will re-enumerate to reflect the newly gathered information from the actual flash storage and it is now ready to process data read/write operations. Of course, if the storage module has to wait too long and does not have enough power, it will begin a soft shutdown routine. Likewise, if the adapter controller fails during the enumeration requests and a certain amount of retries pass, it will abort configuration and enter a failure state. However, as long as these two attributes for each of the controllers are satisfied, each will continue to send acknowledgement ping requests to each other until the wireless handshake between the two modules is complete. In the main section of the controller code several tasks must be completed.

The pseudo-code for the project is an important way to visualize the necessary functions and program flow of the actual code that will be written during the implementation of the project design. The overall pseudo-code for the design is provided in the figures below. The two separate code blocks required for each controller are similar but each is deserving of its own pseudo-code. Figure 29 gives the overall adapter

pseudo-code while Figure 30 provides the pseudo-code for the overall storage controller. Some of the code used will be taken from example code given by Microchip as used by their implementation of a similar Mass Storage Device (MSD) solution. For the Windows drivers to correctly build the drivers for this peripheral device in the MSD device class, every Host Device Descriptor request must be answered correctly by the PIC MCU.

While some aspects of this pseudo-code give a good idea of what is going on at the top level of execution, some of the function names and many of the details have been changed or omitted. For the complete version of the pseudo-code, see Appendix B. The complete version of pseudo-code includes much of the documentation and explanation for the code.

### **Adapter MCU Pseudo-code Functions:**

```
//Adapter Module Initialize Function
InitAM( ){
    Set Timer0 registers with appropriate prescaler;
    Configure ports A thru E and UART;
}

//Function to handle enumeration requests from USB Host
//Called by InitAM( )
EnumUSB( ){
    call USBreset() to set device address to default of 0
    if(host sends Get_Descriptor)
        send Set_Descriptor call to USB Host;
    if(host sends Get_Configuration)
        send Set_Configuration call to USB Host;
    if(host sends Set_Address call)
        set device address to the value sent by host;
    //initialize each USB control function
    USBinterrupt( );
    USBreset( );
    USBdeinit( );
}

//Adapter Module Synchronization Function
//Synchronizes the adapter module MCU with that of the Storage Module
SyncWithSM( ){
    while(no timeout interrupt){
        run sync routine for UART PIC communication;
    }
    set successful connect bit;
```

## Wireless USB Project Design Report

```
}

//DeInit routine for the adapter module
//This is only done if Host requests USB shutdown
DeInitAM(){
    if(host disconnect received){
        send disconnect request to storage PIC;
        while(!timeout interrupt){
            wait for storage PIC ACK;
        }
        send host request ACK;
    }
}

//DataFlow routine for the adapter module
DataFlowAM(){
    if(host is idle)
        Set host to active
    while(!HostIdle){
        set read/write bit DATA_IN or _OUT
        if(DATA_IN){
            send data to storage module PIC
            get sent ACK
        }
        else{ //DATA_OUT
            receive data from storage module PIC
            send receive ACK
        }
    }
}

//Main superloop function for the storage module
main(){
    InitAM( );
    SyncWithSM( );
    check for host diconnect
    while(!adapter PIC sends Host disconnect)
        //check for data send/receive
        DataFlowSM( );
        //check for host disconnect routine
        DeInitSM( );
        //Every 100 cycles check for necessary power
    }
}
```

**Figure 29. Overall Adapter Module Pseudo-Code**

### **Storage Module Pseudo-code Functions:**

```
//Storage Module Initialize Function
InitSM(){
    Power on reset of MCU;
    Configure I/O ports, internal clock, UART;
    Detect external flash memory configuration;
```

## Wireless USB Project Design Report

```
        Detect external transceiver configuration;
    }

//Storage Module Synchronization Function
//Synchronizes the adapter module MCU with that of the Adapter Module
SyncWithAM(){
    while(no timeout interrupt){
        run sync routine for UART PIC communication;
    }
    set successful connect bit;
}

//DataFlow routine for the storage module
DataFlowSM(){
    Set read/write bit DATA_IN or _OUT
    if(DATA_IN){
        while(more data blocks to write)
            if(data block received from adapter PIC){
                send data block out to flash;
                send ACK and ready bit to PIC;
            }
            else
                send negative ACK to PIC;
        }
    }
    else{ //DATA_OUT
        while(more data requested){
            read block of data from flash
            send to adapter PIC
            while(no timeout interrupt)
                wait for receive ACK
        }
    }
}

//DeInit routine for the storage module
//This is done for both USB Host shutdown and for hardware disconnect
DeInitSM(){
    if(host disconnect message received from adapter PIC){
        disconnect external memory I/O port;
        send ACK to adapter PIC;
    }
}

//Main superloop function for the storage module
main(){
    InitSM();
    SyncWithAM();
    check for host diconnect
    while(!adapter PIC sends Host disconnect)
        //check for data send/receive
        DataFlowSM();
        //check for host disconnect routine
        DeInitSM();
        //Every 100 cycles check for necessary power
```

```
}  
}
```

**Figure 30. Overall Pseudo-Code**

The PIC18F4550 has a built in SPI bus which was used in controlling the CYWUSB6934 wireless transceiver. The bus is fully configurable for the PIC in either master or slave mode, different clock periods and also different reads times with respect to the clock. The main functions to initialize the bus and for basic operations are included in the spi.h header file with the Microchip compiler. The actual functions were all in separate files and were copied into one file, TXRX.h. This file also included all of the functions used in communications between the PIC and the wireless transceivers.

There are several functions that were required for the wireless transceivers to communicate over the SPI bus. There are also some functions that were required to turn the transceiver on or off. Since the transceiver would turn on when the power down and the reset pins were pulled high, functions were required to do this. Also there were functions to enable and disable the slave select pin, which is used to signify that a data transfer is in progress. In addition there were functions to initialize the transceiver and also to transfer data to and from the transceivers. The two main functions are used to read the data from the SPI bus and to write to the bus. These functions use the standard SPI functions to read and write but they also enable and disable slave select when necessary, in addition to using the proper read and write structure for the transceivers. A majority of the other functions used in the SPI data transfer used these functions to read and write data. Some of the other functions are for a specific purpose and because of this that will only read or write to one register but they call the same functions but the address is hard

## Wireless USB Project Design Report

coded to the desired address. These functions include a function to transmit data, which will write the data to the transmit data register on the transceiver and then will write to the data valid register and the transceiver will transmit the data. Another is the interrupt check function, when an interrupt is read from the transceiver on the PIC this function is called to see if the interrupt was because of received data from the other wireless transceiver, if it is it will return the data if not it will return '0.' In addition there is an initialization function that initializes the transceivers. This function calls the send data function six times to write to all of the registers to properly configure the transceivers.

The firmware code for each of the controllers were tested using the Microchip C Compiler and the MPLAB development program in conjunction with the MPLAB In-circuit debugger. These were used to program and debug the PIC microcontrollers.

### Final Design Compromises:

The most important original goal that was ultimately were unable to be achieved is the feature of the 480-Mbits high-speed data rate of the USB 2.0 standard. The application knowledge, testing requirements, and components needed to achieve this are not available to an undergraduate non-profit design team and the final product using these data rates would be out of the financial scope of the project. This forced the design to settle for the lower data transfer rates of the full-speed USB 2.0 specifications at approximately 12Mbits per second. What's more, the wireless transmission rates by which the link between the two modules will communicate is further reduced in terms of data rate capabilities. The CYWUSB6932 allows for a transmission rate of only 62.5kB/s. This is of course a big difference from the original goal of over 400MB/s,

however the lower data rates will not compromise the overall effectiveness and operation of the project.

The other major compromise that was made is that the wireless communication will not comply with the ultra-wideband (UWB) standard set forth by the WiMedia Alliance as was originally intended. The Cypress wireless USB transceiver operates in the 2.4GHz frequency range, much lower than what is required by the UWB standard. Also, because a PIC microcontroller and a transceiver running at much lower data rates were chosen, the project could not possibly satisfy the requirements of the WiMedia Alliance standard.

### Testing Procedures

#### Hardware Testing Procedures:

The first test that should be performed is testing the serial interface between the storage module PIC and Toshiba NAND style, flash memory device. It should be verified that the PIC is able to write blocks of data to the device as well as reading from the device. Using a sequence of bit writes and reads, the flash storage device operation can be confirmed.

The testing procedure for the wireless portion of the project will involve using a separate controller other than the PIC to reduce the number of variables. The transceiver will be powered up and then the registers will be set via the controller. Once the transceiver has been properly powered up, both the transmission and reception will be tested. The receiver and transmitter must be tested as a pair for the test to be completed. The output of the receiver unit will be connected to the oscilloscope to monitor ensure that the data is being received.

The first step in testing the transceivers was to test the code that was programmed to the PIC to ensure the proper output. Once this was confirmed the next step was to connect the PIC to one transceiver on a solder less breadboard. When this was connected the code on the PIC was run in debug mode to test the connection between the PIC and the transceivers. This was repeated until the transceiver was powered up properly. Then various pins on the transceiver were probed to make sure they were correct. The first pin checked was the crystal input, since the crystal is powered by the transceiver when it is turned on, if the crystal was turned on then the transceiver was also on. The next step was to transfer data to the transceiver to the transmit data register; this would act as a dummy



data transfer to test if the data was actually being transmitted. This was functioning properly in the middle of the semester, however not long after it was first working it stopped working and through the rest of the semester this worked on and yet never functioned properly again. While the actual data transmission was not working properly the ability to read and write to the transceivers was confirmed using the oscilloscope.

The output from the PIC during the read and write tests were captured on the oscilloscope. Figure 31 shows the data being written to the SPI bus from the PIC. In the figure the line labels  $D_0$  is the output clock signal from the PIC and  $D_2$  is the data being written to the SPI bus. Figure 32 shows a read from a register on the transceiver via the SPI bus. In this figure  $D_0$  and  $D_2$  are the same as in the previous figure and  $D_1$  is the input from the transceiver to the PIC. Figure 33 shows a read from the received data register on from a transceiver during a transmission and it shows that no data is being returned meaning that either no data was received or that no data was ever transmitted.

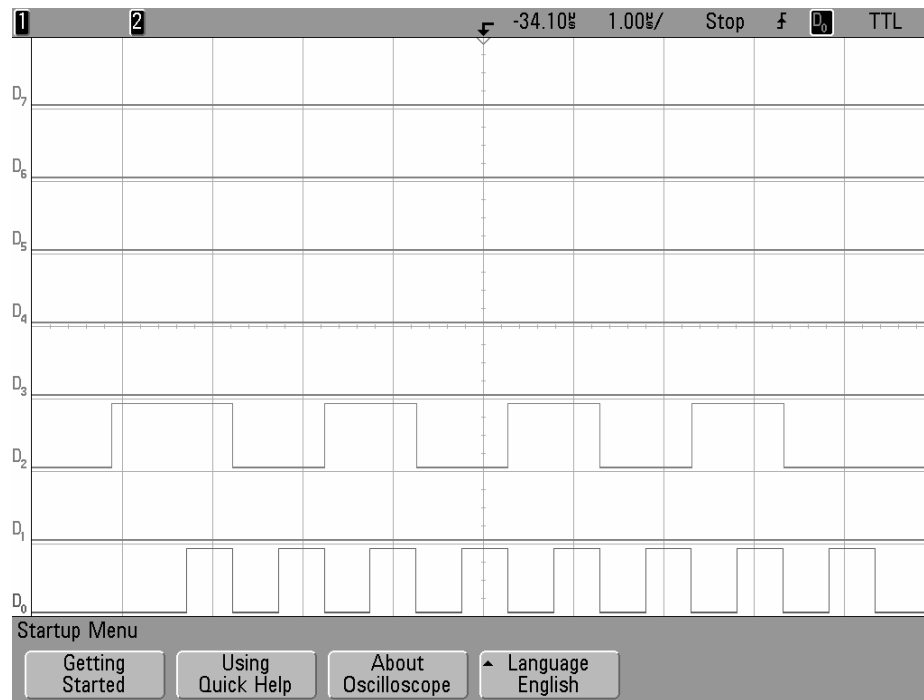


Figure 31. PIC Output during a Write to the SPI bus

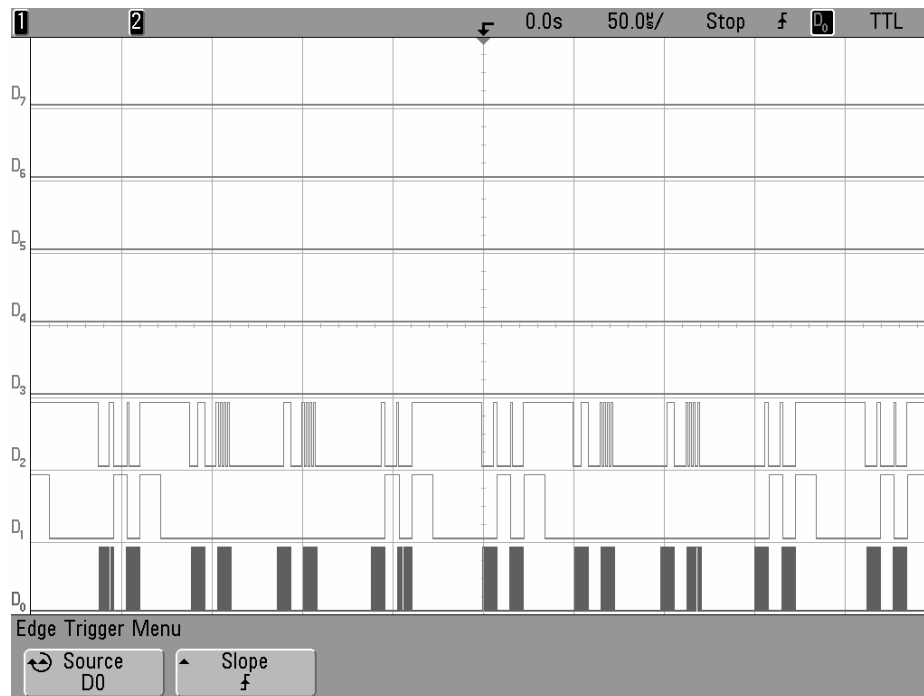
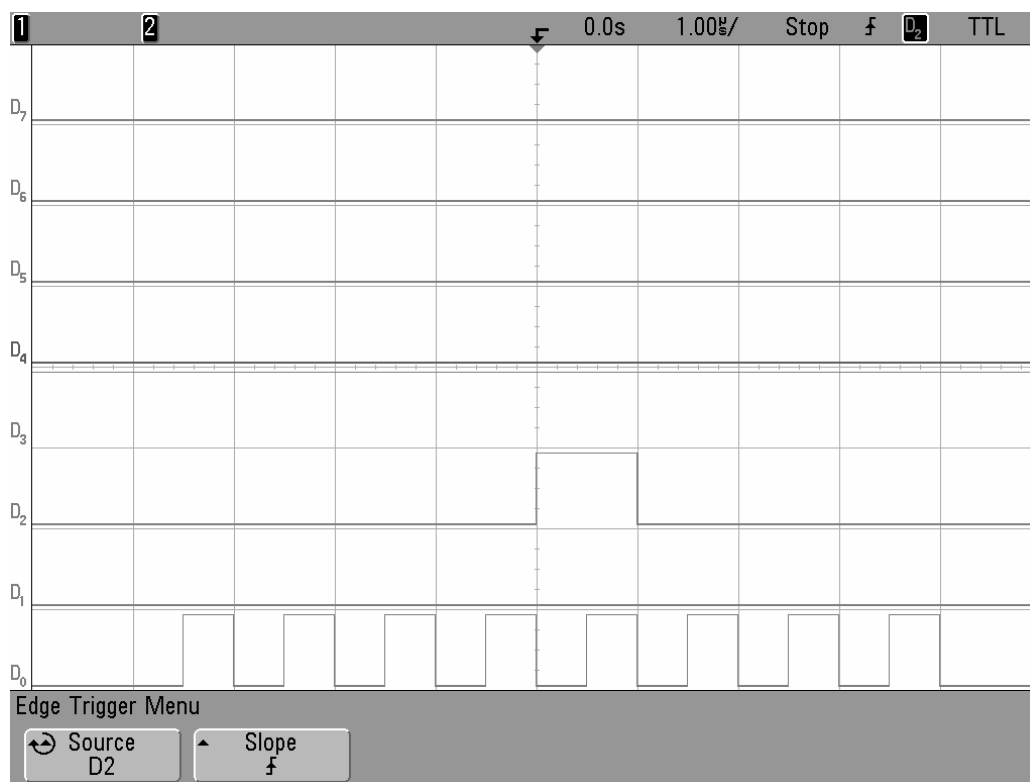


Figure 32. PIC Output during a read from the SPI bus



**Figure 33. Transmitter output during a read of the received data register**

The testing procedure for the wired portion of the project required correctly interfacing the PIC controller to the PC through the USB port. The code must correctly identify the device's type and requirements during the enumeration requests from the USB Host. This code includes the Microchip sublicensed Product ID and Vendor ID along with other device specific identifying descriptors. This firmware code is very complex and difficult to troubleshoot. Once it is confirmed that the USB host is able to correctly identify and configure the device as a full-speed USB 2.0 mass storage device, data can be transferred to the device to test the wired functionality.

There were two different tests that had to be performed. One was when the adapter module is plugged directly into the USB port via the male connector on the module. When this is plugged in, the PC should recognize the device as a mass storage

## Wireless USB Project Design Report

device and then correctly build the drivers. When this is complete the test will consist of transferring files from and to the device and making sure that they can be read and written without error. The other test was to bring the storage module within range of the adapter module and perform wireless handshake requests before wirelessly transferring data to and from the storage module.

As for the power management testing, all Circuits will be constructed powered off of Bench Top DC voltage supplies. Current readings will be made and verified to be within the maximum ratings of the designed power management circuits. The Power Management circuits will then be bread boarded and load tested to verify there output voltages and current supply capabilities.

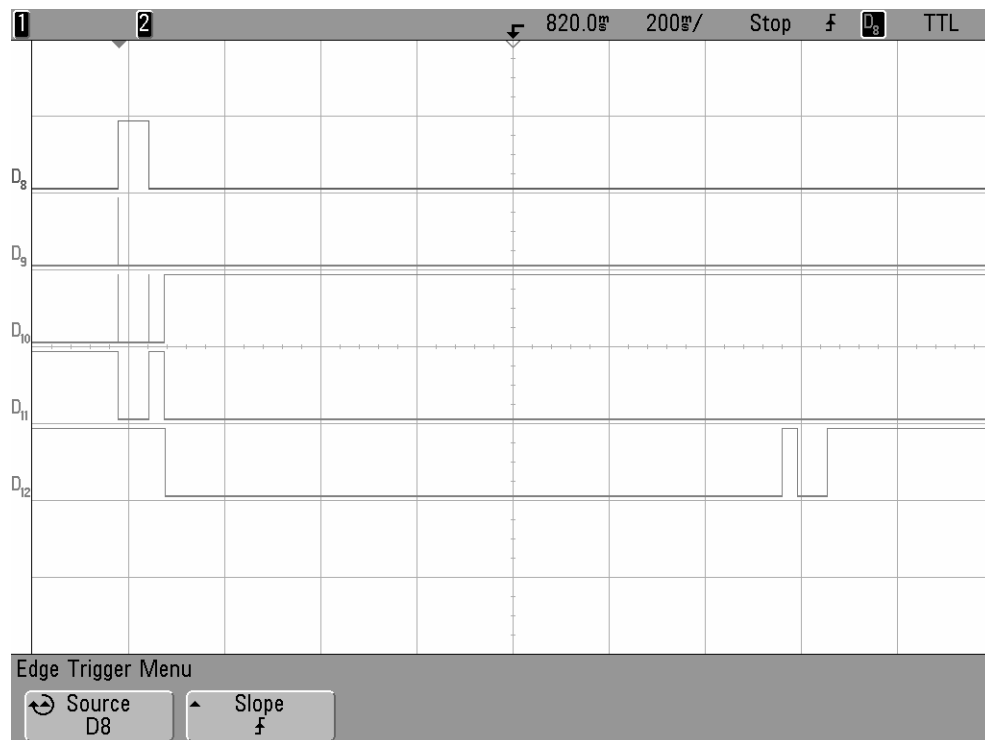
### Software Testing Procedures:

The code for the controllers is written in C using the provided pseudo-code skeleton provided in this report, however, much of the actual code was modified from a similar USB example from Microchip. The main goal was to obtain a working solution but the this was not achieved due to the complexity of the USB enumeration code. Timing schemes are slower than desired to ensure a working solution. Because the project is not completely operational, it is difficult to say what further tests need to be performed in terms of the USB interface. Much of the USB software testing was done by simple trial and error methods to obtain the working portion we now have.

We were planning on testing the possibility of using a real-time operating system approach instead of the provided “superloop” function. This test would weigh the positives and negatives of using this approach to determine whether it would be worth the

extra development time required to implement it. One obvious benefit would be that it would allow the system to perform multiple tasks at the same time making it closer in operation to its PC counterpart. The standard “superloop” technique will work, however, and is the safe route to a working project. We decided against the real-time OS idea first of all because it is not needed for this type of project and offers no benefit and the existing code is far too complicated as it is. Finally, the code portion of this project would not have been possible without the using examples, technical support and researching the specifics of USB extensively.

The results of the extensive USB code were proven to be somewhat correct because our device showed up in the Windows device manager as “DT5 Wireless Storage Device” and showed the same information we obtained using the USB traffic monitor.



**Figure 34. Measured Flash Timing of a Read Command**

## Financial Budget

The project was granted a departmental budget which was not to exceed \$100 per person for use towards parts needed. The project for material parts did not exceed \$300, hence external funding was not required. Any products listed as a cost of \$0.00 denotes that it was acquired or borrowed from the department. All listed costs are estimates. The labor costs are simply included to estimate typical labor costs associated with what a ‘real’ project budget might look like.

**Table 4. Project Budget Sheet**

### **Labor Costs:**

Cost per person per hour  
(USD):

\$10.00

<u>Design Team Member</u>	<u>Weeks</u>	<u>Hours/Week</u>	<u>Cost/Week</u>	<u>Est. Total Cost</u>
Czapor, Jeff	12	8	\$80.00	\$960.00
Hartney, Ben	12	8	\$80.00	\$960.00
Knight, Andrew	12	8	\$80.00	\$960.00
<b>Total:</b>				<b>\$2,880.00</b>

### **Material Costs:**

Cost & quantities of parts.

<u>Material Part Description</u>	<u>Quantity</u>	<u>Cost/Item</u>	<u>Part Total</u>
PIC18F4550 Microcontroller 40-Pin PDIP	2	\$9.41	\$18.82
PIC18LF4550 Low Power 40-Pin PDIP	2	\$11.00	\$22.00
MPLAB Software Development Program	1	0.00	\$0.00
MPLAB In-Circuit Debugger	1	0.00	\$0.00
Maxim/Dallas Semiconductor Battery Charger	1	3.54	\$3.54
Maxim/Dallas Semiconductor DC/DC Converters	2	7.71	\$15.42
ASSMANN Male USB Plug	1	0.92	\$0.92
Female USB Plug	1	7.67	\$7.67
Panasonic Lithium Polymer Battery, 3.6V	1	10.06	\$10.06
Red and Green LED's	3	0	\$0
TST 512-MB NAND Flash, 3.3V	2	16.00	\$32.00

## Wireless USB Project Design Report

Cypress Wired USB Transceiver	2	4.00	\$8.00
Cypress Wireless USB Transceiver	2	4.00	\$8.00
PChannel MOSFET	3	0.33	\$0.99
Schottky Diode	4	0.21	\$0.84
N&P Channel MOSFET Pair	1	0.45	\$0.45
Capacitor, 1uF, 10V, ceramic	1	0.05	\$0.05
Capacitor, 0.1uF, 10V, ceramic	1	0.14	\$0.14
Capacitor, 2.2uF, 6.3V, ceramic	1	0.03	\$0.03
Capacitor, 2.2uF, 10V, ceramic	1	0.04	\$0.04
Capacitor, 4.7uF, 6.3V, ceramic	1	0.15	\$0.15
Capacitor, 10uF, 6.3V, ceramic	1	0.59	\$0.59
Resistor, 150kOhms	1	0.15	\$0.15
Resistor, 115kOhms	1	0.15	\$0.15
Resistor, 10kOhms	1	0.12	\$0.12
Resistor, 1kOhms	1	0.12	\$0.12
Capacitor, 2.0pF	4	2.50	\$10.00
Capacitor, 1.2pF	2	2.50	\$5.00
Capacitor, 27pF	2	2.50	\$5.00
Inductor, 3.3nH	2	0.05	\$0.10
Inductor, 2.2nH	2	0.05	\$0.10
13MHz 10pF Crystal	2	8.75	\$17.50
Resistor, 1.5kOhms	2	0.05	\$0.10
Resistor, 33kOhms	4	0.12	\$0.48
<b>Parts Total</b>			<b>\$168.53</b>

## Second Semester Additions

### Labor Costs:

Cost per person per hour (USD): \$10.00

<u>Design Team Member</u>	<u>Weeks</u>	<u>Hours/Week</u>	<u>Cost/Week</u>	<u>Est. Total Cost</u>
Czapor, Jeff	12	8	\$80.00	\$960.00
Hartney, Ben	12	8	\$80.00	\$960.00
Knight, Andrew	12	8	\$80.00	\$960.00
<b>Total:</b>				<b>\$2,880.00</b>

### Material Costs:

Cost & quantities of parts.

<u>Material Part Description</u>	<u>Quantity</u>	<u>Cost/Item</u>	<u>Part Total</u>
Printed Circuit Boards	2	133.00	266.00
Prototype Adapter 48 pin TSOP to 48 pin DIP	1	30.00	30.00
Prototype Adapter 24 pin QFN to 24 pin	1	22.00	22.00

## Wireless USB Project Design Report

Prototype Adapter 48 pin QFN to 48 pin DIP	1	26.00	26.00
Plastic Enclosure	1	10.58	10.58
Plug connector, 2 pin	2	0.15	0.30
		<b>Parts Total</b>	\$354.88

















































**Material Parts Total:               \$523.41**

**Project Grand Total:               \$6283.41**

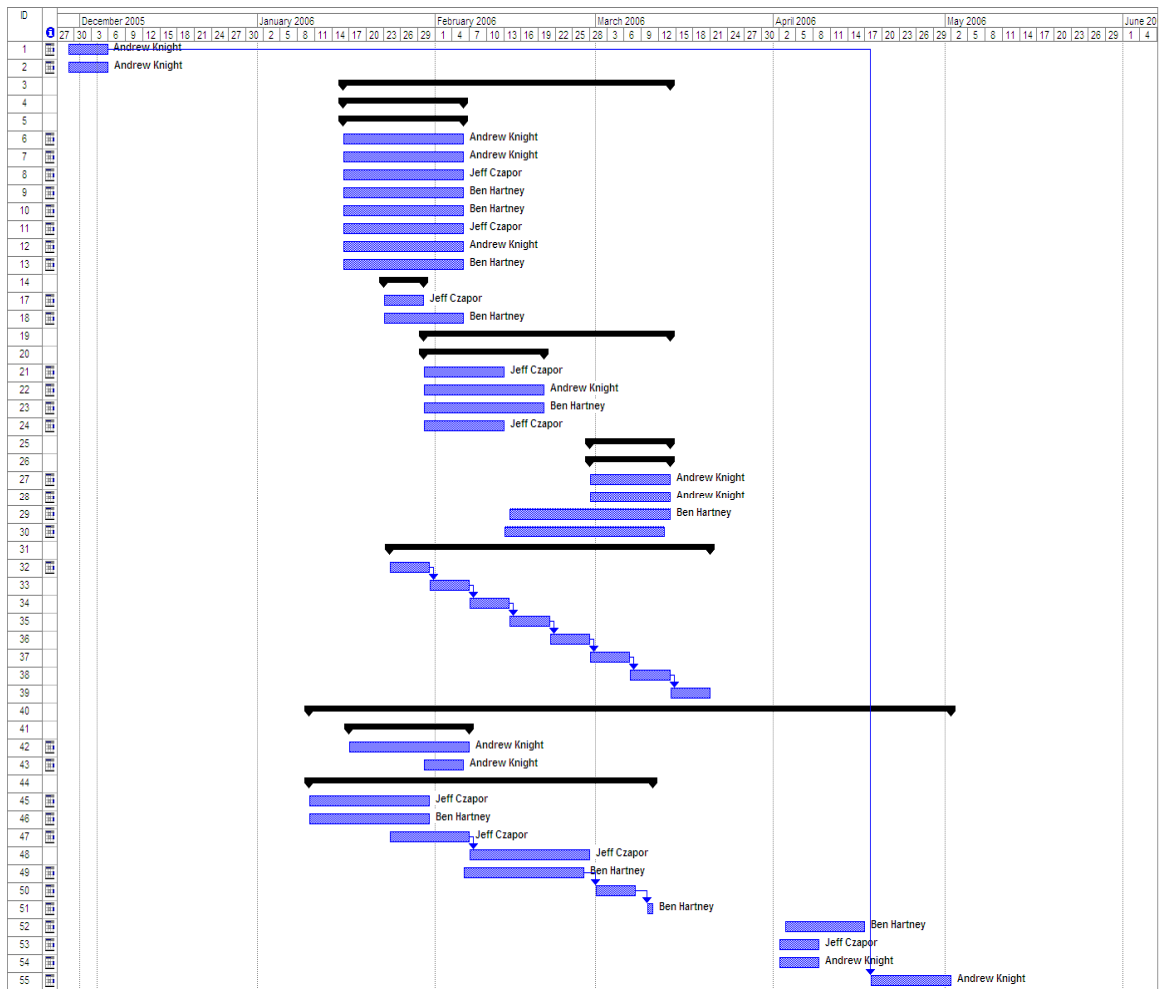


## Project Schedule

**Table 5. Implementation Semester Gantt Chart and Descriptions**

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	 Final Design Report	7 days	Mon 11/28/05	Mon 12/5/05		Andrew Knight
2	 Implementation Gantt Chart	7 days	Mon 11/28/05	Mon 12/5/05		Andrew Knight
3	 Implementation Design and Testing	56.38 days	Mon 1/16/06	Mon 3/13/06		
4	 Implementation Design	21 days	Mon 1/16/06	Mon 2/6/06		
5	 Electrical Design	21 days	Mon 1/16/06	Mon 2/6/06		
6	 Circuit Layout Simulation	21 days	Mon 1/16/06	Mon 2/6/06		Andrew Knight
7	 USB Host Enum	21 days	Mon 1/16/06	Mon 2/6/06		Andrew Knight
8	 USB Flash Recognition	21 days	Mon 1/16/06	Mon 2/6/06		Jeff Czapor
9	 Data Transfer Efficiency	21 days	Mon 1/16/06	Mon 2/6/06		Ben Hartney
10	 Communication Data Rates	21 days	Mon 1/16/06	Mon 2/6/06		Ben Hartney
11	 Power Electronics Tweaking	21 days	Mon 1/16/06	Mon 2/6/06		Jeff Czapor
12	 Error Analysis	21 days	Mon 1/16/06	Mon 2/6/06		Andrew Knight
13	 PC/Module Radiation Conflicts	21 days	Mon 1/16/06	Mon 2/6/06		Ben Hartney
14	 Mechanical Design	7 days	Mon 1/23/06	Mon 1/30/06		
17	 Comm Distance Calculations	7 days	Mon 1/23/06	Mon 1/30/06		Jeff Czapor
18	 EMC and EMI Tests	14 days	Mon 1/23/06	Mon 2/6/06		Ben Hartney
19	 Component Testing	42.38 days	Mon 1/30/06	Mon 3/13/06		
20	 Hardware	21 days	Mon 1/30/06	Mon 2/20/06		
21	 Battery and Power Mgmt	14 days	Mon 1/30/06	Mon 2/13/06		Jeff Czapor
22	 PIC MCU	21 days	Mon 1/30/06	Mon 2/20/06		Andrew Knight
23	 Transceiver	21 days	Mon 1/30/06	Mon 2/20/06		Ben Hartney
24	 Flash Memory	14 days	Mon 1/30/06	Mon 2/13/06		Jeff Czapor
25	 Software	14 days	Mon 2/27/06	Mon 3/13/06		
26	 Development	14 days	Mon 2/27/06	Mon 3/13/06		
27	 MPLAB ICD Testing	14 days	Mon 2/27/06	Mon 3/13/06		Andrew Knight
28	 PIC18 MCU Code Sims	14 days	Mon 2/27/06	Mon 3/13/06		Andrew Knight
29	 Testing Procedures Summary	28 days	Mon 2/13/06	Mon 3/13/06		Ben Hartney
30	 Testing Error Analysis Summary	28 days	Mon 2/13/06	Mon 3/13/06		
31	 Implementation Progress Reports	56 days	Mon 1/23/06	Mon 3/20/06		
32	 Report #1	7 days	Mon 1/23/06	Mon 1/30/06		
33	 Report #2	7 days	Mon 1/30/06	Mon 2/6/06	32	
34	 Report #3	7 days	Mon 2/6/06	Mon 2/13/06	33	
35	 Report #4	7 days	Mon 2/13/06	Mon 2/20/06	34	
36	 Report #5	7 days	Mon 2/20/06	Mon 2/27/06	35	
37	 Report #6	7 days	Mon 2/27/06	Mon 3/6/06	36	
38	 Report #7	7 days	Mon 3/6/06	Mon 3/13/06	37	
39	 Report #8	7 days	Mon 3/13/06	Mon 3/20/06	38	
40	 Final Project Implementation	112 days	Mon 1/9/06	Mon 5/1/06		
41	 Final Software Implementation	21 days	Mon 1/16/06	Mon 2/6/06		
42	 Complete Working MCU Code	21 days	Mon 1/16/06	Mon 2/6/06		Andrew Knight
43	 Full Code Documentation	7 days	Mon 1/30/06	Mon 2/6/06		Andrew Knight
44	 Final Hardware Implementation	60 days	Mon 1/9/06	Fri 3/10/06		
45	 Adapter Bread Board Layout	21 days	Mon 1/9/06	Mon 1/30/06		Jeff Czapor
46	 Storage Bread Board Layout	21 days	Mon 1/9/06	Mon 1/30/06		Ben Hartney
47	 Custom Printed Circuit Boards	14 days	Mon 1/23/06	Mon 2/6/06		Jeff Czapor
48	 Adapter Board Soldering	21 days	Mon 2/6/06	Mon 2/27/06	47	Jeff Czapor
49	 Storage Board Soldering	21 days	Mon 2/6/06	Mon 2/27/06		Ben Hartney
50	 Test and revise Hardware	7 days	Tue 2/28/06	Tue 3/7/06	49	
51	 Complete Working Demonstration	1 day	Thu 3/9/06	Fri 3/10/06	50	Ben Hartney
52	 Testing Procedures Results	14 days	Mon 4/3/06	Mon 4/17/06		Ben Hartney
53	 Final Project Hardware Setup	7 days	Sat 4/1/06	Sat 4/8/06		Jeff Czapor
54	 Final Design Presentation	7 days	Sat 4/1/06	Sat 4/8/06		Andrew Knight
55	 Final Project Report	14 days	Mon 4/17/06	Mon 5/1/06	1	Andrew Knight

# Wireless USB Project Design Report



**Figure 35. Implentation Project Schedule**

Design Team Information

**Team Members:**

Jeff Czapor	- Electrical Engineering Student
Ben Hartney	- Electrical Engineering Student
Andrew Knight	- Computer Engineering Student

**Faculty Advisor:**

Dr. Ugweje	- Department of Electrical and Computer Engineering
------------	---

## Conclusions and Recommendations

This project has seen many challenges throughout the implementation stage. Though the design has been scaled down in performance considerably from its original form, it has still proven to be a difficult application to develop. It should be as easily configurable and universally accepted as any USB flash device in working form. It demonstrated the wireless feature within a generous distance from the PC to which it connects. Data rates for the project were lower than what was initially intended, however it still got the job done. The wireless communication protocol will not satisfy the originally anticipated goal of working within the specifications of the WiMedia Alliance ultra-wideband standards, but it did display a reliable wireless connection in the widely used 2.4GHz frequency range.

As a warning to anyone who is seeking to use the new Wireless USB standard in a similar project design, the technology needs to mature a little more before it will be as universally tested and implemented as its wired counterpart. What seemed to be merely an ambitious project turned out to needing up-to-the-minute technology which was simply out of the scope of a senior design project. By lowering the performance requirements of this project, it became a possibility of a working project. However, it was still too difficult for an undergraduate design team to implement in the time provided. To some degree, it satisfies the goals and design specifications that we set in terms of functionality, reliability, and interoperability. But by focusing on the positive outcomes of the project, the progress that was made during the implementation semester remained as a satisfying and rewarding learning experience.

## References

1. Axelson, Jan. USB Complete, Second Edition. Lakeview Research c. 2001
2. Cypress Wireless USB LS 2.4-GHz DSSS Radio SoC Datasheet. Cypress Semiconductor Corporation c. 2005.
3. Dipert, Brian & Levy, Marcus. Designing with Flash Memory. Anabooks c. 1994.
4. Dual Step-Down DC-DC Power Management IC MAX8621YETG. Maxim Integrated Products.
5. Flash Memory TC58128AFT Data Sheet. Toshiba c. 2001.
6. Li+ Charger IC MAX1874ETE Data Sheet. Maxim Integrated Products.
7. Microchip PIC18F4550 Data Sheet. Microchip Technology Inc c. 2004.
8. Wireless Universal Serial Bus Specification, Revision 1.0. USB Implementers Forum c. 2005.
9. 500mA LDO Data Sheet. Maxim Integrated Products.
10. AN1003 USB Mass Storage Device Using the PIC MCU. Microchip Technology Inc c. 2005.

Appendix A

## Schematic Component List:

Qty.	Refdes	Part Num.	Description
2	U1,U8	428-1624-ND	Cypress Wireless USB Transceiver
2	U2,U9	USB1T20MTC	Fairchild wired USB transceiver
1	U3	PIC18F4550-I/P	PIC18F4550 Microcontroller 40-pin DIP
1	U4	MAX4836EUT33C_T	LDO regulator, 500mA, Maxim
1	U5	MAX1874ETE	Maxim/Dallas Semiconductor Battery Charger
1	U6	MAX8621YETG	Maxim/Dallas Semiconductor DC/DC Converters
1	U7	74074	Flip-Flop, Set/Reset D-type
1	U10	PIC18LF4550-I/P	PIC18F4550 Low Power Microcontroller 40-pin DIP
1	U11	TC58128AFT-ND	Toshiba 128M-bit NAND Flash
4	Q1,Q2,Q3,Q4	FDN302P	Fairchild PChannel MOSFET
1	Q5	FDC6420C	Fairchild N&P Channel MOSFET Pair
4	D1,D2,D3,D4	MBR0520L	Fairchild Schottky Diode
1	D5	350-1357-1-ND	Dialight Red/Green LED
2	C2, C7		Capacitor, 1.2pF
4	C3, C4, C8, C9		Capacitor, 2.0pF
2	C1, C6		Capacitor, 27pF
2	C14, C22		Capacitor, 0.01uF, min 10V, ceramic
2	C6, C17		Capacitor, 0.1uF, min 10V, ceramic
1	C13		Capacitor, 1uF, min 10V, ceramic
6	C15, C16, C24, C26, C29,C30		Capacitor, 2.2uF, min 10V, ceramic
1	C7		Capacitor, 3.3uF, min 10V, ceramic
5	C14, C20, C21, C27,C28		Capacitor, 4.7uF, min 10V, ceramic
1	C31		Capacitor, 4.7uF, min 25V, ceramic
2	C11, C19		Capacitor, 10uF, min 10V, ceramic
4	R2, R3, R5, R6		Resistor, 36 Ohm, 1/8 Watt
2	R13, R14		Resistor, 39 Ohm, 1/8 Watt
1	R7		Resistor, 1K Ohm, 1/8 Watt
2	R11, R15		Resistor, 10K Ohm, 1/8 Watt
2	R1, R4		Resistor, 19K Ohm, 1/8 Watt
1	R16		Resistor, 100K Ohm, 1/8 Watt
1	R9		Resistor, 130K Ohm, 1/8 Watt
1	R10		Resistor, 270K Ohm, 1/8 Watt
1	R12	ERT-D2FHL103S	Panasonic NTC Termistor, 10k @ 25deg.
1	L1		Inductor, 2.2nH
1	L2		Inductor, 3.3nH
2	L5, L6		Inductor, 4.7uH
1	CN3	SC1153-ND	Switch Craft Power Jack
1	CN4	SC1052-ND	Switch Craft Power Plug
1	CN1, CN2	AE9930-ND	AssMann USB Cable

## **Appendix B**

### Updated Pseudo-Code:

```
//*****
// DT5 Wireless USB Project
// Contributions: Andrew Knight, Jeff Czapor
// CodeDocAll Document
// Notes and Explanations:
//     Adapter Module PIC18F4550 (AM-PIC)
//     Adapter Module CYWUSB Transceiver (AT)
//     Storage Module PIC18LF4550 (SM-PIC)
//     Storage Module CYWUSB Transceiver (ST)
//     Tx = Transmit, Rx = Receive
//     Actual values for definitions and variables will be provided with the complete code
//
//*****
```

```
//*****
//Explanation of Host Enumeration Sequence
//This is an overview of the program flow for the enumeration process only
//The code for this is located on two separate PICs, one on the Adapter Module, one on the Storage
//*****
```

The main function superloop will be located on both the AM-PIC and on the SM-PIC.  
It is responsible for initializing and running the entire USB Mass Storage Device (MSD) application.

```
//*****
//Pseudo-code for the AM main function superloop sequence
//Refer to AM subfunction definitions for further explanation
//*****
Include appropriate header files
Define vars used globally for entire code
```

```
void main() {
    Set appropriate variables and timers        //TMR0 and TMR1 will be used
    Call InitAM() function to init PIC and enum the USB Host device
    While (CONFIGURATION_COMPLETED == 0) {} //do nothing and wait until configured
    if (configured successfully) {
        while () { //start of endless superloop
            Perform actions as requested by Host
            //These will be signaled by sending the appropriate reqs using interrupts
        }
    }
}
```

```
//*****
//Pseudo-code for the SM main function superloop sequence
//Refer to SM subfunction definitions for further explanation
//*****
Include appropriate header files
Define vars used globally for entire code
```

```
void main() {
    Set appropriate variables and timers        //TMR0 and TMR1 will be used
    Call InitSM() function to init PIC and enum the USB Host device
    While (CONFIGURATION_COMPLETED == 0) {} //do nothing and wait until configured
    if (configured successfully) {
        while () { //start of endless superloop
            Perform actions as requested by Host
            //These will be signaled by sending the appropriate reqs using interrupts
        }
    }
}
```

# Wireless USB Project Design Report

```
}  
}  
  
//*****  
//*****  
//Explanation of Host Enumeration Sequence  
//This is an overview of the program flow for the enumeration process only  
//The code for this is located on two separate PICs, one on the Adapter Module, one on the Storage  
//*****  
//*****  
  
When the Adapter Module is plugged into the USB port it receives power from the bus and is powered ON  
The Host will immediately detect the device, the device must then be ready to reply to device  
    descriptor reqs from the Host  
The function responsible for this process is the InitUSB() function  
Host sends Get_Status req to get info about the particular device, this also contains info about the  
    speed the device will use  
The host then sends a Set_Status req to reset the device  
After the reset, the device should be ready to reply to the Host control reqs,  
    the default Endpoint is 0 and the default address is 00h  
Host sends Get_Descriptor req to obtain a specific descriptor for the device including a max packet size  
Next, the Host sends a Set_Address to give the device a new address, the device enters the Address state  
Host again sends Get_Descriptor req, this time obtaining complete info about the device,  
    sub-descriptors are sent as well  
Based on the device's response to each of the USB Host's control reqs, the Host will build and load  
    a driver for the device  
The new device driver selects a config and the Host sends a Set_Configuration req to the device,  
    device enters Configured state  
USB Enumeration is complete and the device is ready to use  
  
//*****  
//*****  
//This is the actual pseudo-code for the HOST ENUMERATION PROCESS  
//This code is physically located in and run from two different PICs  
//*****  
//*****  
  
//*****  
//AMEnumUSB Function of the AM-PIC  
//AM-PIC function routine pseudo-code for HOST ENUMERATION only  
//AMDefaultEnumUSB() is called first once the AM device is plugged into the USB port on the PC  
//AMEnumUSB() is called if and when the info about the SM and flash are gathered by the AM-PIC  
//This function is called from the InitAM() function  
//*****  
void AMDefaultEnumUSB() {  
    Define local vars  
    Enumerate Report, Config, HID, Interface, and String Descriptors  
    //This will not actually be the configuration state after the complete process is finished  
}  
  
void AMEnumUSB() {  
    Define local vars  
    Re-Enumerate Report, Config, HID, Interface, and String Descriptors  
}  
  
//*****  
//SMEnumUSB Function of the SM-PIC  
//SM-PIC function routine pseudo-code for HOST ENUMERATION only  
//This function is called from the InitSM() function  
//*****  
void SMEnumUSB() {  
    Define local vars  
    //This function will basically serve in a supporting role to provide info about the flash and the  
    // "slave" SM-PIC to the AM-PIC which is directly connected to the USB Host on the PC.  
    Call the InitFlashInfo() function  
}
```



# Wireless USB Project Design Report

```
//*****
//*****
//Explanation of Initialization Sequence
//This is an overview of the program flow for the init sequence only
//The code for this is located on two separate PICs, one on the Adapter Module, one on the Storage
//The AM-PIC will be the Master MCU and the SM-PIC will be the slave MCU
//*****
//*****

//InitAM Function of the AM-PIC
//This is the AM-PIC function routine pseudo-code for Initialization only
//This function is called from the main routine on the AM-PIC
//*****
void InitAM() {
    Init external function defs and global vars
    Define local function vars
    //Due to the strict nature of the Host requests, the AM-PIC must perform a default enumeration
    //of the device
    Call AMDefaultEnumUSB() to reply to USB Host descriptor, address and configuration requests
    Call FindSMReq() //Detect Storage Module
    if (the SM is detected and connected properly) {
        Set the global var PIC_HANDSHAKE_SUCCESS = 1
        //handshake between two PICs successful and ready for further comm
        Call AMEnumUSB()
        //Enum must take place a second time after the SM-PIC is detected
        //and info about it is gathered
        if () {
            Set global EnumSuccess var = 1 //enum OK and ready for use
        } else
            Set global EnumSuccess var = 0 //enum of device failed
    } else
        //SM device not found or communication not successful, AMDefaultEnumUSB still holds
        Set the global var PIC_HANDSHAKE_SUCCESS = 0
    //return to the main function
}

//*****
//InitSM Function of the SM-PIC
//This is the SM-PIC function routine pseudo-code for Initialization only
//This function is called from the main routine on the SM-PIC
//*****
void InitSM() {
    Init external function defs and global vars
    Define local function vars
    Call InitFlashInfo() function to gather info about the external flash memory
    //Once the SM-PIC has established a connection/config of the Toshiba flash memory,
    //a search for the AM-PIC can be done
    //setup ST for receiving req data from AM-PIC
    Prepare for receiving the AM-PIC handshake requests
    Start TMR1 to look for timeout on the receive req
    if (TMR1 reaches a max value) {
        Set AMCONNECT_TIMEOUT var = 1
        //return to main
    } else {
        //Wait for connection reqs from the AM-PIC
        if (req arrives from Am-PIC via the ST) {
            //this is responsible for replying to AM-PIC req data
            Call ReceiveAMReq() function
        }
    }
    //return to main function
}

//*****
//Other functions needed including interrupts
//These will be called from the code segment below or from main
//*****

//This function is used during PIC initialization to gather the info about the flash memory
```

## Wireless USB Project Design Report

```
//as needed by the host enumeration
void InitFlashInfo() {
    Read initial flash properties and set corresponding Host descriptors
    Enable global interrupts and vars
}

//This function is used by the AM-PIC ONLY to find the SM-PIC used in the initial handshake
void FindSMReq() {
    //Simply do a read req of the SM-PIC with descriptors about the flash capacity, speed of data
    //transfer, and timing and interrupt schedulers.
    Enable appropriate interrupt, timers, and global vars
}

//This function is used by the SM-PIC ONLY to reply to the AM-PIC's req during the handshake process
void ReceiveAMReq() {
    //Simply read back the read reqs from the SM-PIC of the previous FindSMReq() function.
    Enable appropriate interrupts and set global vars indicating handshake is complete
}

//Read of the flash device
//Reads are performed by the host in clusters (denoted as blocks by Toshiba).
//To perform a read the Flash is input the Read command, followed by a starting address over I/O.
//Data is then available to read from the I/O.
void FlashRead() {
    Set Logic inputs
    //Use Read Mode 1 sequence
    CLE: high
    CE: low
    WE: high
    ALE: low
    RE: high
    Set Operation Mode
    I/O: Send 00H command //Read Mode 1
    WE: high to low to high // command sent on rising edge of WE
    Set pointer to the Starting address input //address is entered over 3 clock cycles
    Reset logic inputs
    ALE: high
    Send address of first byte in page
    I/O: Send 00H //Always start read from 1st byte of page.
    WE: high to low to high //Address sent on rising edge of WE.
    I/O: Send ??H
    //First 4 bytes of page address (page address is equal to sector number*32
    WE: high to low to high //Address sent on rising edge of WE.
    I/O: Send ??H //Second 4 bytes of page address.
    WE: high to low to high //Address sent on rising edge of WE.
    Read one cluster of data in to PIC and buffer in memory
    Reset logic inputs:
    ALE: low
    Set counter to 16384 // number of bytes in 1 cluster
    RE: high to low to high //data is available to read after falling edge.
    Decrement counter by 1
    If
        Counter > 0 restart loop
    Else
        Cluster is finished reading
    Value stored in File Allocation Table for this Cluster.
    Case (value is 0xFFFF8-0xFFFFF)
        Read is finished. End routine
    Case (value is 0x0002-0xFFEF)
        Set cluster to read to this value. Restart loop.
    Default
        FAT table is corrupted. Unable to read file.
}

//Write of data to the flash device.
//Writes are performed by inputting 1 page of data at a time in to the Flash memories buffer.
//After inputting 526 bytes of data the Auto Page Program routine is called.
//This is repeated 32 times to write 1 sector of data.
void FlashWrite() {
```

## Wireless USB Project Design Report

```
Set Logic inputs //Use Auto-Program sequence
    CLE: high
    CE: low
    WE: high
    ALE: low
    RE: high
Set Operation Mode
    I/O: Send 80H command //Serial Data Input command
    WE: high to low to high //Command sent on rising edge of WE.
Set pointer to Starting address input //address is entered over 3 clock cycles
Reset logic inputs
    ALE: high
Send address of first byte to write to
    I/O: Send 00H //Always start write from 1st byte of page.
    WE: high to low to high //Address sent on rising edge of WE.
    I/O: Send ??H //First 4 bytes of page address, equal to sector number*32
    WE: high to low to high //Address sent on rising edge of WE.
    I/O: Send ??H //Final 4 bytes of page address.
    WE: high to low to high //Address sent on rising edge of WE.
Send Data to write to Flash
Reset logic inputs
    ALE: low
Set counter to 512 // number of bytes on 1 page
Send 1 byte of data over I/O
    WE: high to low to high //Data sent on rising edge.
Decrement counter by 1
If
    Counter > 0. Restart loop
Else
    Cluster is finished reading. End loop
Send Auto-Program Command
Reset logic inputs
    CLE: high
Send Command
    I/O: Send 80H //Auto-Program command
    WE: high to low to high //Auto Program is executed on rising edge
Delay 1000us //max time of page program
Verify successful page program
//After programming, the data is read back in to the register to be auto verified by the device.
I/O: Send 70H command //Status Read command
Case 1
    I/O 0 = 0 program failed. Repeat ProgramFlash subroutine
    //Repeat the routine 20 times. If after 20 attempts the program was still unsuccessful,
    //mark the page as bad, and do not use in future. Write data to another unused page.
Case 2
    I/O 0 = 1 program was successful.
End Auto-Program
Reset logic inputs
    WP: low
Loop this routine 32 times to write the 32 pages of data that make up 1 cluster.

}

//Erasing of the Flash memory is performed per block (also referred to as a sector) of
//memory (1block = 32 pages = 16384 bytes). When a file is deleted the actual data is not deleted from the memory.
//Only the files Directory Table and FAT entries are deleted. Erasing of Flash memory is time consuming and so
//can be done at any time when the device would otherwise be idle. Erasing must be performed before programming
//(Subjecting a programmed NAND cell to the program voltage will significantly reduce the life of the cell).
void FlashErase() {
    Set Logic inputs //Use Auto Block Erase sequence
        CLE: high
        CE: low
        WE: high
        ALE: low
        RE: high
    Set Operation Mode
        I/O: Send 60H command //Auto Block Erase Setup command
        WE: high to low to high //Command sent on rising edge of WE.
    Set pointer to Starting address input //address is entered over 3 clock cycles
    Reset logic inputs
```

## Wireless USB Project Design Report

```
        ALE: high
    Send address of block to erase
        I/O: Send ??H //First 4 bytes of block address
        //(This is the page address of the 1st page in the block, Value = block address*32)
        WE: high to low to high //Address sent on rising edge of WE.
        I/O: Send ??H //Final 4 bytes of block address.
        WE: high to low to high //Address sent on rising edge of WE.
    Send Erase Start Command
    Reset logic inputs
        ALE: low
    Send Command
        CLE: high
        I/O: Send D0H //Erase start command
        WE: high to low to high //command is sent on rising edge
        CLE: low
    Delay 10ms //max time of block erase
    Verify successful block erase
    //After a block erase, the data is read back in to the register to be auto verified by the device.
        I/O: Send 70H command //Status Read command
    Case 1
        I/O 0 = 0 Erase failed. Repeat ProgramFlash subroutine
        // Mark the Block as bad, and do not use in future.
    Case 2
        I/O 0 = 1 Erase was successful. End subroutine
    End Auto Block Erase
    Reset logic inputs
        WP: low
    //End of erase portion of code
}

//*****
//Interrupt Definitions and Explanations
//The main interrupt service routine branches off to various interrupts
//Sub-function definitions are listed below as well
//Interrupts are called from both the SM-PIC and the AM-PIC, needs to be located on both as well
//Much of the actual interrupt code will borrowed from Microchip examples
//*****

//Interrupt Service Routine
//This must be located on both PICs and may be called by both
void interrupt ISR () {
    if (USB interrupts are enabled and interrupt flag is set)
        Service the USB interrupt
    } else
        PIC core interrupt occurred //return to caller function
}

//USB Interrupt Handler Function Definition
//Based on the interrupt that has fired, the appropriate interrupt function will be called
void USBInterrupt() {
    if (there is USB activity and it is enabled)
        USBActivity()
    else if (there is a USB reset req and it is enabled)
        USBReset()
    else if (there is a USB token done and token done is enabled)
        USBTokenDone()
    else if (there is a USB error and USB errors are enabled)
        USBError()
    else if (there is a USB idle req to save power and it is enabled)
        USBSleep()
}

//This is called when the activity int is called re-waking the device after a period of inactivity
void USBActivity() {
    Disable activity int and activity int enable
    Enable idle int ad set IF flag to 0 to wake the device
}

//This is called to reset the USB
```

## Wireless USB Project Design Report

```
void USBReset() {
    Init the buffer descriptors
    Set all vars to their default values
    Set device endpoint address to 0 for enumeration
    Re-enable the USB
}

//This is called to process the token done interrupt to signal a response to a USB transaction
void USBTokenDone() {
    Depending on the token request from the Host, this will perform the appropriate action
    //This code will basically be taken from the Microchip example code almost as is
}

//If the SIE on the PIC detects an error sets appropriate flags
void USBError() {
    Simply update the appropriate error counter variable and exit
    //Uses a series of if statements to determine which counter to update
}

//This is used to put the device to sleep to save power during inactivity
void USBSleep() {
    Enable the idle var and disable the activity var
    Enable the activity interrupt var
}

//*****
//*****
//Explanation of HOST READ REQ of the MEMORY
//This is an overview of the program flow for the read operation
//The code for this is located on two separate PICs, one on the Adapter Module, one on the Storage
//*****
//*****
Host sends req to AM-PIC
AM-PIC sends req to AT
AT moves req data to AT SPI
AT Tx req
ST receives req from AT
ST throws Rx interrupt to SM-PIC
SM-PIC reads req data from ST SPI
SM-PIC sends read req to flash memory
Flash returns data to SM-PIC
while (there is more data to be read as per the read req) {
    SM-PIC receives data in a series of 8-bit packets
    SM-PIC sends data from internal RAM to ST once it is full
    if (SM_PIC RAM is full) {
        ST receives data and moves it to ST SPI to be Tx
        ST Tx data to AT
        AT receives data and signals and interrupt to the AM-PIC
        //the AM-PIC has the same capacity as the SM-PIC
        AM-PIC reads data in from AT into internal buffer RAM
        AM-PIC sends data thru the SIE to the Host
    }
}
Host sends completed ACK

//*****
//*****
//This is the actual pseudo-code for the READ OPERATION
//*****
//*****

//*****
//AMReadReq Function of the AM-PIC
//AM-PIC function routine pseudo-code for Read Operation only
//This section must be preceeded by the Init() code section in the complete SW design main section
//All data transmission is by the serial connection of the transceivers
//*****
void AMReadReq() {
    Define global vars and external dependencies
```

## Wireless USB Project Design Report

```
Host sends read req command to AM-PIC
//will use USB I/O Port for comm with external CYWUSB transceiver
Toggle the dir bit of the UCON reg on the AM-PIC SIE equal to 0 //set as output
//Write to the SIE which is dedicated to the DIO I/O port on the AT
//For each of the following, must wait appropriate delay before the next operation
Write to REG_CONTROL register addr: 0x03 on the AT for Tx enable //set to enable Tx
Write to REG_TX_DATA register addr: 0x0F and then 8-bit data of the address for the read req
Tx the req data
Write to REG_CONTROL on the AT for Rx enable
//get ready to receive the first returned data from memory
Toggle direction bit of the UCON reg on the AM-PIC equal to 1 //set as input
Start TMR0 //this is used as a timeout timer on the Rx signal

//Wait for returned first packet of returned data from the ST
while(no interrupt received from AT) {
    //wait loop for interrupt
    if (TMR0 reaches time limit)
        break out of while loop and set NO_RESPONSE var = 1
}
if (NO_RESPONSE var = 1)
    then return read error signal to Host
else {
    //Once the data is returned from the memory via the ST, the AT will signal a Rx
    //interrupt to the AM-PIC
    while (there is more data to be read from memory) {
        //AM-PIC reads data in 8-bit packets until the internal RAM is full
        if (RAM is full)
            Send data from internal RAM of AM-PIC to Host through USB SIE
        else {
            read data from REG_RX_DATA_A reg addr: 0x09 of the AT which
            holds the 8-bit received value
            data is put into the internal RAM buffer of AM-PIC
        }
    }
    Set READ_COMPLETE var = 1
}
Send read success signal to Host
//End of read section of the AM-PIC
}

//*****
//SMReadReq Function of the SM-PIC
//SM-PIC main routine pseudo-code for Read Operation only
//This section must be preceded by the Init() code section in the complete SW design main section
//All data transmission is by the serial connection of the transceivers
//*****
void SMReadReq() {
    Define global vars and external dependencies
    Toggle dir bit of USTAT reg on SM-PIC to 0 //set as output
    //Write to SIE control reg UCON which is dedicated to the DIO I/O port on the ST
    //For each of the following, must wait appropriate delay before the next operation
    //get ready to receive the first returned data from memory
    Write to REG_CONTROL register addr: 0x03 on the ST for Rx enable
    //Tx the req data
    Write to REG_RX_DATA_A register addr: 0x0F and then 8-bit data of the addr for the read req
    //set as input ready to receive first data from ST
    Toggle dir bit of USTAT reg on SM-PIC to 1
    Reset TMR0

    //Must then wait for a req from the AM-PIC via the ST
    //If after a long time of no req has passed, this should relinquish control back
    //to the SM-PIC main function.
    //After checking necessary power, deinit req, write req, and other status bits,
    //it will again return to check for a read req
    while (no Rx interrupt from ST) {
        //wait loop for interrupt
        Poll the REG_RX_INT_STAT register on the ST for int based on a certain timing scheme
        if (TMR0 reaches time limit)
            break out of while loop and set NO_READ_REQ var = 1
    }
}
```

## Wireless USB Project Design Report

```
}
if (NO_READ_REQ var = 1)
    then break fuction execution and return to SM-PIC main routine
else {
    //if code has entered the else portion, an interrupt has occurred
    Rx interrupt was thrown by ST
    SM-PIC reads data from REG_RX_DATA_A register on the ST
    //this should contain the location address and size of the requested data to be read
    SM-PIC sends req to the external flash memory chip
    Flash returns first packet of data to SM-PIC
    while (there is more data to be read as per the read req) {
        SM-PIC receives data in a series of 8-bit packets
        SM-PIC sends data from internal RAM once it is full to the ST
        if (SM-PIC RAM is full) {
            ST receives data and moves it to ST SPI to be Tx
            ST Tx data to AT
        }
    }
    Set the REG_CONTROL to Rx enabled
    //Wait for the data received ACK from the AT
}
//End of Read fuction of the SM-PIC
}

//*****
//*****
//Explanation of HOST WRITE REQ to the MEMORY
//This is an overview of the program flow for the write operation
//The code for this is located on two separate PICs, one on the Adapter Module, one on the Storage
//*****
//*****
Host sends write req to AM-PIC
AM-PIC reads data from the Host to be written in 8-bit packets until USB RAM is full
while (there is more data to write) {
    AM-PIC sends data to write to AT
    AT moves data to be written to AT SPI
    AT Tx data
    ST receives data from AT
    ST throws Rx Interrupt to SM-PIC
    SM-PIC reads data from ST SPI
    SM-PIC sends write request to flash memory until the buffer RAM is empty
    while (there is more data to write in the SM-PIC RAM)
        write the data to flash
    SM-PIC sends packet write completed ACK to ST
    ST Tx ACK data
    AT receives data from ST
    AT sends to AM-PIC
}
AM-PIC sends write complete ACK to Host

//*****
//*****
//This is the actual pseudo-code for the WRITE OPERATION
//*****
//*****

//*****
//AMReadReq Function of the AM-PIC
//AM-PIC function routine pseudo-code for Write Operation only
//This section must be preceeded by the Init() code section in the complete SW design main section
//*****
void AMWriteOp() {
    Define global vars
    Host sends read req command to AM-PIC
    //The host will communicate with the flash via the AM-PIC SIE
    Toggle dir bit of USTAT reg on SM-PIC to 0    //set as output
```

## Wireless USB Project Design Report

```

//For each of the following, must wait appropriate delay before the next operation
//get ready to receive the first returned data from memory
Write to REG_CONTROL register addr: 0x03 on the ST for Rx enable
Write to REG_RX_DATA_A register addr: 0x0F and then 8-bit data for the addr of the read req
Tx the req data
//set as input ready to receive first data from ST
Toggle dir bit of USTAT reg on SM-PIC to 1
Reset the timer to wait for the reply from the SM-PIC //uses TMR0

//Wait for returned ACK data packet returned from the ST
while(no interrupt received from AT) {
    //wait loop for interrupt
    if (TMR0 reaches time limit)
        break out of while loop and set NO_WRITE_RESPONSE var = 1
}
if (NO_WRITE_RESPONSE var = 1)
    then return write error signal to Host
else {
    //Once all the data is returned from the memory via the ST,
    //the AT will signal a Rx interrupt to the AM-PIC
    while (there is more data to be written to memory) {
        //AM-PIC reads data in 8-bit packets until the internal RAM is full
        if (RAM is full)
            Send data from internal RAM of AM-PIC to Host through USB SIE
        else {
            //data is received in a serial fashion
            data is written to REG_RX_DATA_A register addr: 0x09 of the AT
            //this holds the 8-bit received value
            data is put into the internal RAM buffer of AM-PIC
        }
    }
    Set WRITE_COMPLETE var = 1
}
}

//*****
//SMReadReq Function of the SM-PIC
//SM-PIC main routine pseudo-code for Write Operation only
//This section must be preceded by the Init() code section in the complete SW design main section
//*****
void SMWriteOp() {
    Define global vars
    Toggle dir bit of USTAT reg on SM-PIC to 0 //set as output
    //Write to SIE control reg UCON which is dedicated to the DIO I/O port on the ST
    //For each of the following, must wait appropriate delay before the next operation
    Write to REG_CONTROL register addr: 0x03 on the ST for Rx enable
    //get ready to send the first returned data from memory
    Write to REG_RX_DATA_A register addr: 0x0F and then 8-bit data of the addr for the read req
    Tx the req data
    Toggle dir bit of USTAT reg on SM-PIC to 1
    //set as input ready to receive first data from ST
    Reset TMR0

    //Must then wait for the ACK from the AM-PIC via the ST
    //If after a long time of no req has passed, this should relinquish control back to the
    //SM-PIC main function.
    //After checking necessary power, deinit req, write req, and other status bits,
    //it will again return to check for a write req
    while (no Rx interrupt from ST) {
        //wait loop for interrupt
        Poll the REG_RX_INT_STAT register on the ST for int based on a certain timing scheme
        if (TMR0 reaches time limit)
            break out of while loop and set NO_WRITE_REQ var = 1
    }
    if (NO_WRITE_REQ var = 1)
        then break fuction execution and return to SM-PIC main routine
    else {
        //if code has entered the else portion, an interrupt has occured from ST
        Rx interrupt was thrown by ST
    }
}

```



## Wireless USB Project Design Report

```
SM-PIC reads data from REG_RX_DATA_A register on the ST
//this should contain the location address and size of the requested data to be written
SM-PIC sends data to the external flash memory chip
while (there is more data to be written as per the write req) {
    SM-PIC receives data in a series of 8-bit packets
    SM-PIC writes data from internal RAM once it is full to the flash
    if (SM-PIC RAM is NOT full) {
        ST receives data and moves it to ST SPI
        //this is to be loaded into internal RAM on SM-PIC
        Data is loaded in a serial fashion until full
    }
}
Flash returns ACK data to SM-PIC
Set the REG_CONTROL to Rx enabled
//Wait for the data received ACK from the AT
if (receives ACK from AT)
    Set WRITE_SUCCESS bit = 1
else
    Set WRITE_SUCCESS bit = 0
}
//End of Read fuction of the SM-PIC
}

//*****
//*****
//End of Pseudo-Code Explanations and Definitions
//*****
//*****
```

## Appendix C

### Final Source Code:

```

/*****
/
/      Filename:          AdapterTest.c
/      Directory:         AdapterConnect
/      Authors:           Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:           2.20.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:             Contains main() function
/
*****/

/** I N C L U D E S *****/
#include <pl18f4550.h>
#include "adapter_global_defs.h"

/** PIC Configuration Settings *****/
#pragma config PLLDIV = 5           /* PLL Prescaler */
#pragma config CPUDIV = OSC4_PLL6   /* CPU Clock Prescaler */
#pragma config USBDIV = 2           /* USB Clock Selection */
#pragma config FOSC = ECPLL_EC      /* Oscillator Selection */
#pragma config FCMEM = OFF          /* Fail Safe Clock Monitor */
#pragma config IESO = OFF           /* Oscillator Switchover */
#pragma config PWRT = OFF           /* Power Up Timer */
#pragma config BOR = OFF            /* Brown Out Reset */
#pragma config BORV = 1             /* Brown Out Voltage */
#pragma config VREGEN = ON          /* USB Voltage Regulator */
#pragma config WDT = OFF            /* Watchdog Timer */
#pragma config WDTPS = 1            /* Watchdog Timer Postscaler */
#pragma config MCLRE = OFF          /* MCLR */
#pragma config LPT1OSC = OFF        /* Low Power Timer 1 */
#pragma config PBDEN = OFF          /* Port B A/D */
#pragma config CCP2MX = OFF         /* CCP2 MUX */
#pragma config STVREN = OFF         /* Stack Full/Underflow Reset */
#pragma config LVP = OFF            /* Single Supply ICSP */
#pragma config ICPRT = OFF          /* In-Circuit Debug/Programming */
#pragma config XINST = OFF          /* Extended Instruction Set */
#pragma config DEBUG = OFF          /* Debugger */
#pragma config CP0 = OFF            /* Code Protection Block 0 */
#pragma config CP1 = OFF            /* Code Protection Block 1 */
#pragma config CP2 = OFF            /* Code Protection Block 2 */
#pragma config CPB = OFF            /* Boot Block Protection */
#pragma config CPD = OFF            /* EEPROM Protection */

/** V A R I A B L E S *****/

/** P R I V A T E   P R O T O T Y P E S *****/
void initAMPIC(void);
void runSystem(void);

/** D E C L A R A T I O N S *****/
/** TESTING VARS AND FUNCTIONS *****/
int count = 1;
    //testing only

/*****
* Function:          void main(void)
* Overview:          Main program entry point.
*****/
void main(void)
{
    msDelay(10);    //Startup Delay

```

## Wireless USB Project Design Report

```
//Adapter Module Init Routine
initAMPIC();

//Infinite Loop
while(1)
{
    //runSystem();
    msDelay(30000);
    //if(count == 1){
        transmitData(0B10101010);
        count = 0;
    //}
}

/*****
 * Function:      void initAMPIC(void)
 * Overview:      Initializes all aspects of the Adapter Module PIC
 *****/
void initAMPIC(void)
{
    //Init AM here
    ADCON0 = 0x07;                //set all ports as digital I/O
    TRISD = 0;                    //set PORTD as output only
    TRISB = 0;                    //set PORTB as output only

    //Initialize the Pic in SPI mode then power up the transceiver and Configure it.
    initTrans();
}

/*****
 * Function:      void runSystem(void)
 * Overview:      All checks of the PIC during normal operation
 *****/
void runSystem(void)
{
    /*
    if(count == 1){
        transmitData(0B10101010);
        count = 0;
    }
    else{
        transmitData(0B01010101);
        count = 1;
    }
    */
}

//End of AdapterTest.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                TxRx.h
/      Directory:              Adapter Connect
/      Authors:                Ben Hartney
/                               Design Team 5 Wireless USB
/      Version:                4.21.2006
/      Processor:              PIC18F4550
/      Compiler:               Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

#include <p18f4550.h>
#include <spi.h>

//Delay function
extern void usDelay(unsigned int);

//TxRx functions
extern void disableSS(void);
extern void enableSS(void);
extern void initTX(void);
extern void B4(unsigned int);
extern void B5(unsigned int);
extern void B7(unsigned int);
extern unsigned char irqCheck(void);
extern void powerUpTX(void);
extern void powerDownTX(void);
extern unsigned int readData(unsigned int);
extern void reset(void);
extern void sendData(unsigned int, unsigned int);
extern void transmitData(unsigned int);
extern void initTrans(void);
extern unsigned char signalStrength(void);

//EOF TxRx.h
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          adapter_global_defs.h
/      Directory:         Adapter Connect
/      Authors:           Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:           2.20.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

#ifndef ADAPTER_GLOBAL_DEFS_H
#define ADAPTER_GLOBAL_DEFS_H

#include "TxRx.h"          //TxRx functions
#include "pic2pic.h"

/** GLOBAL VARIABLES *****/
/*****

//Universal Variables *****/
#define TRUE    1
#define FALSE   0
#define NULL    0

/** GLOBAL FUNCTIONS *****/
/*****
//Global Delay function defs
extern void msDelay(unsigned int);
extern void usDelay(unsigned int);
extern void findSMPIC(void);

//End of adapter_global_defs.h

```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          pic2pic.h
/      Directory:         Adapter Connect
/      Authors:           Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:           2.20.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/
#ifndef PIC2PIC_H
#define PIC2PIC_H

extern void findSMPIC(void);

//End of pic2pic.h

```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          TxRx.c
/      Directory:        Adapter Connect
/      Authors:          Ben Hartney
/                        Design Team 5 Wireless USB
/      Version:          4.21.2006
/      Processor:        PIC18F4550
/      Compiler:         Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****/

#include "TxRx.h"

//Start of functions used to communicate to the CYWUSB6934
//Wireless Transceivers

/*****/

//Disable SS

void disableSS(void){

    B5(1);          //Sets SS Low

} // End of DisableSS

/*****/

//Enable SS

void enableSS(void){

    B5(0);          //Sets SS Low

} //End of EnableSS

/*****/

//initTX

void initTX(void){

//The first bit should be set to a '1' to signify a write to that location

    sendData(0b10011100,0b00000001);    //Wake Enable Register Set

    sendData(0b10000011,0b10000000);    //Control Register Set for receive mode

    sendData(0b10110010,0b01000001);    //Clock Manual Register Set

    sendData(0b10110011,0b01000001);    //Clock Enable Resigter Set

    sendData(0b10100100,0b01000000);    //Output Clock Disable Register Set

    sendData(0b10100011,0b00000011);    //Internal Power Amplifier Gain Set

} //End of initTX

/*****/

//IO handling

void B4(unsigned int valueB4){
    if(valueB4==1){
        PORTB = PORTB + 0b00010000;
    }
}
```

## Wireless USB Project Design Report

```
    }
    else
        PORTB = PORTB - 0b00010000;

} //End of B4

void B5(unsigned int valueB5){
    if(valueB5==1){
        PORTB = PORTB + 0b00100000;
    }
    else
        PORTB = PORTB - 0b00100000;

} //End of B5

void B7(unsigned int valueB7){
    if(valueB7==1){
        PORTB = PORTB + 0b10000000;
    }
    else
        PORTB = PORTB - 0b10000000;

} //End of B7

//End of IO handling

/*****

//IRQ Check

char irqSTATUS;

unsigned char irqCheck(void){

    irqSTATUS = readData(0b00001000);

    if(irqSTATUS!=0){
        irqSTATUS=readData(0b00001001);
    }

    return(irqSTATUS);
} //End of IRQ Check

*****/

//Power Up

void powerUpTX(void){

    B7(1);                //Set Reset High

    B4(1);                //Set PD High

    usDelay(2100);

    B5(1);                //Set SS High

} //End of Power UP

*****/

void powerDownTX(void){

    B4(0);

    usDelay(2000);

} //End of Power Down
```



## Wireless USB Project Design Report

```

/*****

//Read Data

char addressCHAR = 0;
char statusCHAR = 0;
char dataCHAR = 0;

int dataINT = 0;

unsigned int readData(unsigned int addressINT){

    addressCHAR = (char) addressINT;    //cast int addressCHAR to a char

    WriteSPI(addressCHAR);              //Calls the WriteSPI function in
spi.h

/*****
*    statusCHAR = DataRdySPI();
*
*    while(statusCHAR=='0'){
*        statusCHAR = DataRdySPI();
*    };
*****/

    dataCHAR = ReadSPI();                //Calls the ReadSPI function in spi.h

    dataINT = (int) dataCHAR;            //cast char dataCHAR to an int

    return(dataINT);

} //End of Read Data

/*****

//Reset

void reset(void){

    B7(0);

    usDelay(100);

    B7(1);

    usDelay(2100);
} //End of Reset

/*****

//Send Data

char address1;
char data1;

void sendData(unsigned int address,unsigned int data){

    enableSS();

    address1 = (char) address;           //cast int address to a char
    data1 = (char) data;                 //cast int data to a char

    WriteSPI(address1);

    WriteSPI(data1);

    disableSS();
} //End of Send Data

*****/

```

## Wireless USB Project Design Report

```

/*****

//Transmit Data

char dataCH;
int addressTX1;
char addressCH1;
int addressTX2;
char addressCH2;

void transmitData(unsigned int dataTX){

    addressTX1 = 0b10001111;
    addressTX2 = 0b10010000;

    addressCH1 = (char) addressTX1;
    addressCH2 = (char) addressTX2;

    //Writting data to the transmit register

    sendData(0x83,0x40);                                //Control Register Set for transmit
mode

    enableSS();

    dataCH = (char) dataTX;                                //cast int data to a char

    WriteSPI(addressCH1);                                //Writes the data to the
                                                         //data
register on the
    WriteSPI(dataCH);                                //CYWUSB6934 Wireless
transceiver

    disableSS();

    //Writting data to the data valid register

    enableSS();

    WriteSPI(addressCH2);                                //Writes the data to the
                                                         //Data valid
register on the
    WriteSPI(0x11111111);                                //CYWUSB6934 Wireless transceiver

    disableSS();

    sendData(0x83,0x80);                                //Control Register Set for receive
mode

} //Transmit Data

*****/

//Signal Strength

char strengthCHAR;
int strengthINT;

unsigned char signalStrength(void){

    strengthCHAR = readData(0b00100010);

    strengthINT = (int) strengthCHAR;                                //cast char dataCHAR to an
int

    if(strengthINT>1){
        return(1);
    }
    else{

```

## Wireless USB Project Design Report

```
        return(0);
    }

}

/*****

//Close SPI

#undef CloseSPI
void CloseSPI( void )
{
    SSPCON1 &= 0xDF;           // disable synchronous serial port
} //End of close SPI

*****/

//SPI Data Ready

#undef DataRdySPI
unsigned char DataRdySPI( void )
{
    if ( SSPSTATbits.BF )
        return ( +1 );        // data in SSPBUF register
    else
        return ( 0 );         // no data in SSPBUF register
} //SPI Data Ready

*****/

//GetsSPI

void getsSPI( unsigned char *rdptr, unsigned char length )
{
    while ( length )           // stay in loop until length = 0
    {
        *rdptr++ =getcSPI();   // read a single byte
        length--;              // reduce string length count by 1
    }
} //End of getsSPI

*****/

//open SPI

void OpenSPI( unsigned char sync_mode, unsigned char bus_mode, unsigned char smp_phase)
{
    SSPSTAT &= 0x3F;           // power on state
    SSPCON1 = 0x00;            // power on state
    SSPCON1 |= sync_mode;      // select serial mode
    SSPSTAT |= smp_phase;      // select data input sample phase

    switch( bus_mode )
    {
        case 0:                // SPI bus mode 0,0
            SSPSTATbits.CKE = 1; // data transmitted on rising edge
            break;
        case 2:                // SPI bus mode 1,0
            SSPSTATbits.CKE = 1; // data transmitted on falling edge
            SSPCON1bits.CKP = 1; // clock idle state high
            break;
        case 3:                // SPI bus mode 1,1
            SSPCON1bits.CKP = 1; // clock idle state high
            break;
        default:                // default SPI bus mode 0,1
            break;
    }

    switch( sync_mode )
    {
        case 4:                // slave mode w /SS enable
    
```

## Wireless USB Project Design Report

```
#if defined(__18F6310) || defined(__18F6390) || \
    defined(__18F6410) || defined(__18F6490) || \
    defined(__18F8310) || defined(__18F8390) || \
    defined(__18F8410) || defined(__18F8490)
    TRISFbits.TRISF7 = 1;          // define /SS pin as input
#else
    TRISAbits.TRISA5 = 1;          // define /SS pin as input
#endif
case 5:
    // slave mode w/o /SS enable
#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
    TRISBbits.TRISB1 = 1;          // define clock pin as input
#else
    TRISCbits.TRISC3 = 1;          // define clock pin as input
#endif
SSPSTATbits.SMP = 0;              // must be cleared in slave SPI mode
break;
default:
    // master mode, define clock pin as output
#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
    TRISBbits.TRISB1 = 0;          // define clock pin as output
#else
    TRISCbits.TRISC3 = 0;          // define clock pin as output
#endif
break;
}

#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
    TRISC &= 0x7F;                  // define SDO as output (master or slave)
    TRISB |= 0x01;                  // define SDI as input (master or slave)
#else
    TRISC &= 0xDF;                  // define SDO as output (master or slave)
    TRISC |= 0x10;                  // define SDI as input (master or slave)
#endif
SSPCON1 |= SSPENB;                 // enable synchronous serial port
} //End of Open SPI

/*****

//putsSPI

void putsSPI( unsigned char *wrptr )
{
    while ( *wrptr )                // test for string null character
    {
        SSPBUF = *wrptr++;          // initiate SPI bus cycle
        while( !SSPSTATbits.BF );   // wait until 'BF' bit is set
    }
} //End of putsSPI

/*****

//Read SPI

unsigned char ReadSPI( void )
{
    SSPBUF = 0x00;                  // initiate bus cycle
    while ( !SSPSTATbits.BF );      // wait until cycle complete
    return ( SSPBUF );               // return with byte read
} //End of ReadSPI

/*****

//writSPI

unsigned char WriteSPI( unsigned char data_out )
{
    SSPBUF = data_out;              // write byte to SSPBUF register
    if ( SSPCON1 & 0x80 )           // test if write collision occurred
```

## Wireless USB Project Design Report

```
    return ( -1 );           // if WCOL bit is set return negative #
else
{
    while( !SSPSTATbits.BF ); // wait until bus cycle complete
}
return ( 0 );               // if WCOL bit is not set return non-negative#
} //End of WritSPI

/*****

//init

void initTrans(void)
{
    OpenSPI(SPI_FOSC_16, MODE_00, SMPMID);           //Configures the PIC for SPI

    //Calls the OpenSPI function defined in spi.h

    powerUpTX();

    initTX();

} //End of init

//End of TxRx.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          pic2pic.c
/      Directory:         Adapter Connect
/      Authors:           Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:           2.20.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#include "pic2pic.h"

void findSMPIC(void)
{
}

//End of pic2pic.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                delay.c
/      Directory:              Adapter Connect
/      Authors:                Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                3.30.2006
/      Processor:              PIC18F4550
/      Compiler:               Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#include <pl18f4550.h>
#define IMAX 210          //was 206, 250

void msDelay(unsigned int count)
{
    unsigned int outer;
    unsigned char inner;
    for(outer=0; outer < count*4; outer++){           //was count*2
        for(inner=0; inner<IMAX; inner++)
            //asm("nop"); //do not use with MCC18
            Nop();
    }
}

void usDelay(unsigned int count){
    unsigned int outer;
    for(outer=0; outer < count/16; outer++){          //was count/32
    }
}

//End of delay.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          Test.c
/      Directory:        Storage Connect
/      Authors:          Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:          3.30.2006
/      Processor:        PIC18F4550
/      Compiler:         Microchip C18 ver. 3.02
/      Notes:            Contains main() function
/                        This is the main file for the Storage Module
PIC
/
/*****
*****/

/** I N C L U D E S *****/
#include <stdio.h>
#include <pl18f4550.h>
#include "storage_global_defs.h"                //globally used vars and functions

/** PIC Configuration Settings *****/
#pragma config PLLDIV = 5                      /* PLL Prescaler */
#pragma config CPUDIV = OSC4_PLL6             /* CPU Clock Prescaler */
#pragma config USBDIV = 2                     /* USB Clock Selection */
#pragma config FOSC = ECPLL_EC                /* Oscillator Selection */
#pragma config FCMEM = OFF                    /* Fail Safe Clock Monitor */
#pragma config IESO = OFF                     /* Oscillator Switchover */
#pragma config PWRT = OFF                     /* Power Up Timer */
#pragma config BOR = OFF                      /* Brown Out Reset */
#pragma config BORV = 1                       /* Brown Out Voltage */
#pragma config VREGEN = ON                    /* USB Voltage Regulator */
#pragma config WDT = OFF                      /* Watchdog Timer */
#pragma config WDTPS = 1                      /* Watchdog Timer Postscaler */
#pragma config MCLRE = OFF                    /* MCLR */
#pragma config LPT1OSC = OFF                  /* Low Power Timer 1 */
#pragma config PBADEN = OFF                   /* Port B A/D */
#pragma config CCP2MX = OFF                   /* CCP2 MUX */
#pragma config STVREN = OFF                   /* Stack Full/Underflow Reset */
#pragma config LVP = OFF                      /* Single Supply ICSP */
#pragma config ICPT = OFF                     /* In-Circuit Debug/Programming */
#pragma config XINST = OFF                    /* Extended Instruction Set */
#pragma config DEBUG = OFF                    /* Debugger */
#pragma config CP0 = OFF                      /* Code Protection Block 0 */
#pragma config CP1 = OFF                      /* Code Protection Block 1 */
#pragma config CP2 = OFF                      /* Code Protection Block 2 */
#pragma config CPB = OFF                      /* Boot Block Protection */
#pragma config CPD = OFF                      /* EEPROM Protection */

/** V A R I A B L E S *****/
//Test vars only!
#define LED_ON      0x01;
#define LED_OFF     0x00;

/** P R I V A T E   P R O T O T Y P E S *****/
void initSMPIC(void);                          //Storage Module PIC
init
void checkLoop(void); //Testing

/** D E C L A R A T I O N S *****/

/** TESTING VARS AND FUNCTIONS ONLY!!! *****/
int LED_SLOW_BLINK = 1000;
int LED_FAST_BLINK = 100;
int picsConnected = 0;
int count = 1;
//testing var used by activity_led function below
char RecievedDataCHAR = 0;                    //testing only
int RecievedDataINT = 0;                      //testing only

```



## Wireless USB Project Design Report

```
void activity_led(unsigned int value)
{
    //Simple running LED blink test
    if(count == 1) { PORTD = 0B00000001; count = 0; }
    else { PORTD = 0B00000000; count = 1; }
    msDelay(value);
}

/*****
 * Function:      void main(void)
 * Overview:      Main program entry point.
 *****/
void main(void)
{
    msDelay(100);
    //Initial delay check
    initSMPIC();
    //Storage PIC setup
    while(1){
        //Simple Test Code
        //Storage Module Test Code Here!
        //RecievedDataCHAR = irqCheck(); //irqCheck() is
        problematic!
        //RecievedDataCHAR = 0B00000010;
        //RecievedDataINT = (int) RecievedDataCHAR; //cast char ReceivedDataCHAR
        to an int
        //PORTD = RecievedDataINT;

        if(picsConnected == 0)
            PORTD = 0B00000001;
        else
            activity_led(LED_FAST_BLINK);

        checkLoop();
        //servicePICRequests();

        /*//Simple LED Test for main
        msDelay(100);
        if(count == 1){
            PORTD = 0B10101010;
            count = 0;
        }
        else{
            PORTD = 0B01010101;
            count = 1;
        }
        */
    }
}

/*****
 * Function:      void initSMPIC(void)
 * Overview:      Initializes all aspects of the Adapter Module PIC
 *****/
void initSMPIC(void)
{
    //Universal Setup
    ADCON0 = 0x07; //set all ports as digital I/O
    TRISD = 0; //set PORTD as output only
    TRISB = 0; //set PORTB as output only

    //Setup the PIC for SPI mode then power up the transceiver and Configure it.
    initTransceiver();
}
```

## Wireless USB Project Design Report

```
//For Testing
void checkLoop(void)
{
    char data = 0;
    if(picsConnected == 0){
        data = irqCheck();
        if(data != 0)
            picsConnected == 1;
    }
    //else{
    //}
}

//End of Test.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                flash.h
/      Directory:              Storage Connect
/      Authors:                Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                3.30.2006
/      Processor:              PIC18F4550
/      Compiler:               Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****
#define FLASH_H
#define FLASH_H

#include <p18f4550.h>

//Flash Definitions *****/
#define SMALL_DATA
#define EEPROM_SIZE          512
// #define FLASH_ERASE
#define RELOAD_FLASH

//Macros to access bytes within words and words within longs
//Taken from pic18.h
#define LOW_BYTE(x)          ((unsigned char)(x)&0xFF)
#define HIGH_BYTE(x)         ((unsigned char)(x>>8)&0xFF)
#define LOW_WORD(x)          ((unsigned int)(x)&0xFFFF)
#define HIGH_WORD(x)         ((unsigned int)(x>>16)&0xFFFF)

#define __mkstr1(x)          #x
#define __mkstr(x)           __mkstr1(x)
#define __CONFIG(n, x)      asm("\tpsect config,class=CONFIG");\
                             asm("global config_word" __mkstr(n)); \
                             asm("config_word" __mkstr(n)":"); \
                             asm("\torg (" __mkstr(n) "-1)*2"); \
                             asm("\tdw " __mkstr(x))

#define __IDLOC(w)           asm("\tpsect idloc,class=IDLOC");\
                             asm("\tglobal\tidloc_word"); \
                             asm("idloc_word:"); \
                             asm("\tirpc\t__arg," __mkstr(w)); \
                             asm("\tdb 0f&__arg&h"); \
                             asm("\tendm")

#if defined(EEPROM_SIZE)
#define __EEPROM_DATA(a, b, c, d, e, f, g, h) \
                             asm("\tpsect eeprom_data,class=EEDATA"); \
                             asm("\tdb\t" __mkstr(a) ", " __mkstr(b) ", " __mkstr(c) ", "
__mkstr(d) ", " \
                             __mkstr(e) ", " __mkstr(f) ", " __mkstr(g) ", "
__mkstr(h))

// MACROS for EEPROM Access
//Macro versions of EEPROM read and write
//Taken from pic18.h
#if EEPROM_SIZE > 256
/*
#define EEPROM_READ(addr) \
    (EEADRH=((addr)>>8)&0xFF),EEADR=((addr)&0xFF), \
    wait_on_wr(), \
    CARRY=GIE,GIE=0, \
    EECON1&=0x3F,RD=1, \
    (EEDATA)); \
    if(CARRY)GIE=1
#else
#define EEPROM_READ(addr) \
    (EEADR=(addr),\

```

## Wireless USB Project Design Report

```
        wait_on_wr(), \
        CARRY=GIE,GIE=0,\
        EECON1&=0x3F,RD=1, \
        (EEDATA)); \
        if(CARRY)GIE=1
#endif

#if    EEPROM_SIZE > 256
#define EEPROM_WRITE(addr, value) \
    wait_on_wr(); \
    EEADRH=((addr)>>8)&0xFF;EEADR=((addr)&0xFF); \
    EEDATA=(value); \
    EECON1&=0x3F; \
    CARRY=0;if(GIE)CARRY=1;GIE=0; \
    WREN=1;EECON2=0x55;EECON2=0xAA;WR=1; \
    EEIF=0;WREN=0; \
    if(CARRY)GIE=1
#else
#define EEPROM_WRITE(addr, value) \
    wait_on_wr(); \
    EEADR=((addr)&0xFF); \
    EEDATA=(value); \
    EECON1&=0x3F; \
    CARRY=0;if(GIE)CARRY=1;GIE=0; \
    WREN=1;EECON2=0x55;EECON2=0xAA;WR=1; \
    EEIF=0;WREN=0; \
    if(CARRY)GIE=1
#endif

//Macro Flash operations
//erasing a flash program memory row
//Taken from pic18.h
#if defined(SMALL_DATA)
#define FLASH_ERASE(addr) \
    TBLPTRU=0;\
    TBLPTR=(far unsigned char *)addr; \
    EECON1|=0x94;EECON1&=0xBF; \
    CARRY=0;if(GIE)CARRY=1;GIE=0;\
    EECON2=0x55;EECON2=0xAA;WR=1; \
    asm("\tNOP"); \
    if(CARRY)GIE=1
#else
#define FLASH_ERASE(addr) \
    TBLPTR=(far unsigned char *)addr; \
    EECON1|=0x94;EECON1&=0xBF; \
    CARRY=0;if(GIE)CARRY=1;GIE=0;\
    EECON2=0x55;EECON2=0xAA;WR=1; \
    asm("\tNOP"); \
    if(CARRY)GIE=1
#endif
*/

/* read/write EEPROM data memory */
/* Taken from pic18.h */
extern unsigned char eeprom_read(unsigned int address);
extern void eeprom_write(unsigned int address,unsigned char data);

/* read/write/erase sections of program memory */
/* Taken from pic18.h */
extern unsigned char flash_read(unsigned long addr);
extern void flash_write(far unsigned char *,unsigned char,far unsigned char *);
extern void flash_erase(unsigned long addr);

/* read/write device configuration registers */
/* Taken from pic18.h */
extern unsigned int config_read(unsigned char reg_no);
extern void config_write(unsigned char reg_no, unsigned int dataword);

/* read factory-programmed device ID code */
/* Taken from pic18.h */
extern unsigned int device_id_read(void);
```

## Wireless USB Project Design Report

```
/* read/write to user ID regs */
/* Taken from pic18.h */
extern unsigned char idloc_read(unsigned char reg_no);
extern void idloc_write(unsigned char reg_no,unsigned char data);

/* general purpose EE/flash init sequence used by above functions */
/* Taken from pic18.h */
extern void initiate_write(void);
extern void wait_on_wr(void);

#endif

/* Taken from pic18.h */
/* Accurate read/write macros for 16-Bit timers */
/** please note, the timer needs to be enabled **
 *** to handle 16-Bit read/write operations for ***
 *** these routines to be of benefit ***/
#define T1RD16ON  T1CON|=0x80
#define T3RD16ON  T3CON|=0x80
#define WRITETIMER0(x)  TMR0H=(x>>8);TMR0L=(x&0xFF)
#define WRITETIMER1(x)  TMR1H=(x>>8);TMR1L=(x&0xFF)
#define WRITETIMER3(x)  TMR3H=(x>>8);TMR3L=(x&0xFF)
#define READTIMER0  (TMR0)
#define READTIMER1  (TMR1)
#define READTIMER3  (TMR3)

//End of flash.h
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          flash.c
/      Directory:         Storage Connect
/      Authors:           Andrew Knight
/                        Design Team 5 Wireless USB
/      Version:           3.30.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#include <p18f4550.h>
#include "storage_global_defs.h"
#include "flash.h"

//Variables
//volatile bit RESET;

void flash_ISR(void)
{
    //Flash ISR code here
    /*
        if (RELOAD_FLASH)
        {
            RESET=1;
        }
    */
}

#ifdef FLASH_ERASE
void flash_erase(unsigned long addr)
{
    FLASH_ERASE(addr);
}
#endif
/*
void flash_write(far unsigned char * source_addr,unsigned char length,far unsigned char *
dest_addr)
{
    //variable declaration
    unsigned char BUFFER[64];
    unsigned char index;
    unsigned char offset;

    offset=(unsigned char)dest_addr;
    while(offset>64)
        offset-=64;
    dest_addr-=offset;

    while(length)
    {
        // fill the 64 byte data buffer either from ...
        for(index=0;index<64;index++)
        {
            if((index>=offset)&&(length)) // specifed data area or ...
            {
                BUFFER[index]=*(source_addr++);
                length--;
            }
            else // otherwise from destination sector.
                BUFFER[index]=*(dest_addr+index);
        }

        FLASH_ERASE(dest_addr);

        // now begin to copy the buffer to the destination area
        #if defined(SMALL_DATA)

```

## Wireless USB Project Design Report

```
TBLPTRU=0;        // if 16 bit code pointers selected, TBLPTRU must be
cleared manually
#endif
TBLPTR=--dest_addr;    // load the destination address

offset=0;
for(index=0;index<64;index++)
{
    TABLAT=BUFFER[index]; // copy the data buffer to the
    asm("\tTBLWT*");       // internal write buffer

    if(++offset==8)        // after every 8th byte, the
    {                      // the write buffer is written to flash.
        EEPGD=1;WREN=1;    // this is the required
sequence
        CARRY=0;if(GIE)CARRY=1;GIE=0;
        EECON2=0x55;
        EECON2=0xAA;
        WR=1;
        asm("\tNOP");
        if(CARRY)GIE=1;
        WREN=0;
        offset=0;
    }
    dest_addr+=65;
}

}

unsigned char flash_read(unsigned long addr)
{
    TBLPTRL=((addr)&0xFF);
    TBLPTRH=((addr)>>8)&0xFF);
    TBLPTRU=((addr)>>8)>>8);
    asm("\tTBLRD*");
    return TABLAT;
}
*/
//End of flash.c
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbdsc.c
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.5.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

/*****
* -usbdsc.c-
* This file contains the USB descriptor information. It is used
* in conjunction with the usbdsc.h file. When a descriptor is added
* or removed from the main configuration descriptor, i.e. CFG01,
* the user must also change the descriptor structure defined in
* the usbdsc.h file. The structure is used to calculate the
* descriptor size, i.e. sizeof(CFG01).
*
* A typical configuration descriptor consists of:
* At least one configuration descriptor (USB_CFG_DSC)
* One or more interface descriptors (USB_INTF_DSC)
* One or more endpoint descriptors (USB_EP_DSC)
*
* Naming Convention:
* To resolve ambiguity, the naming convention are as followed:
* - USB_CFG_DSC type should be named cdxx, where xx is the
*   configuration number. This number should match the actual
*   index value of this configuration.
* - USB_INTF_DSC type should be named i<yy>a<zz>, where yy is the
*   interface number and zz is the alternate interface number.
* - USB_EP_DSC type should be named ep<##><d>i<yy>a<zz>, where
*   ## is the endpoint number and d is the direction of transfer.
*   The interface name should also be listed as a suffix to identify
*   which interface does the endpoint belong to.
*
* Example:
* If a device has one configuration, two interfaces; interface 0
* has two endpoints (in and out), and interface 1 has one endpoint(in).
* Then the CFG01 structure in the usbdsc.h should be:
*
* #define CFG01 rom struct                                \
* {   USB_CFG_DSC          cd01;                          \
*     USB_INTF_DSC         i00a00;                        \
*     USB_EP_DSC           ep01o_i00a00;                  \
*     USB_EP_DSC           ep01i_i00a00;                  \
*     USB_INTF_DSC         i01a00;                        \
*     USB_EP_DSC           ep02i_i01a00;                  \
* } cfg01
*
* Note the hierarchy of the descriptors above, it follows the USB
* specification requirement. All endpoints belonging to an interface
* should be listed immediately after that interface.
*
* -----
* Filling in the descriptor values in the usbdsc.c file:
* -----
*
* Most items should be self-explanatory, however, a few will be
* explained for clarification.
*
* [Configuration Descriptor(USB_CFG_DSC)]
* The configuration attribute must always have the _DEFAULT
* definition at the minimum. Additional options can be ORed
* to the _DEFAULT attribute. Available options are _SELF and _RWU.
* These definitions are defined in the usbdefs_std_dsc.h file. The
* _SELF tells the USB host that this device is self-powered. The
* _RWU tells the USB host that this device supports Remote Wakeup.

```



## Wireless USB Project Design Report

```
*
* [Endpoint Descriptor(USB_EP_DSC)]
* Assume the following example:
* sizeof(USB_EP_DSC),DSC_EP,_EP01_OUT,_BULK,64,0x00
*
* The first two parameters are self-explanatory. They specify the
* length of this endpoint descriptor (7) and the descriptor type.
* The next parameter identifies the endpoint, the definitions are
* defined in usbdefs_std_dsc.h and has the following naming
* convention:
* _EP<##>_<dir>
* where ## is the endpoint number and dir is the direction of
* transfer. The dir has the value of either 'OUT' or 'IN'.
* The next parameter identifies the type of the endpoint. Available
* options are _BULK, _INT, _ISO, and _CTRL. The _CTRL is not
* typically used because the default control transfer endpoint is
* not defined in the USB descriptors. When _ISO option is used,
* addition options can be Ored to _ISO. Example:
* _ISO|_AD|_FE
* This describes the endpoint as an isochronous pipe with adaptive
* and feedback attributes. See usbdefs_std_dsc.h and the USB
* specification for details. The next parameter defines the size of
* the endpoint. The last parameter in the polling interval.
*
* -----
* Adding a USB String
* -----
* A string descriptor array should have the following format:
*
* rom struct{byte bLength;byte bDscType;word string[size];}sdxxx={
* sizeof(sdxxx),DSC_STR,<text>;
*
* The above structure provides a means for the C compiler to
* calculate the length of string descriptor sdxxx, where xxx is the
* index number. The first two bytes of the descriptor are descriptor
* length and type. The rest <text> are string texts which must be
* in the unicode format. The unicode format is achieved by declaring
* each character as a word type. The whole text string is declared
* as a word array with the number of characters equals to <size>.
* <size> has to be manually counted and entered into the array
* declaration. Let's study this through an example:
* if the string is "USB" , then the string descriptor should be:
* (Using index 02)
* rom struct{byte bLength;byte bDscType;word string[3];}sd002={
* sizeof(sd002),DSC_STR,'U','S','B'};
*
* A USB project may have multiple strings and the firmware supports
* the management of multiple strings through a look-up table.
* The look-up table is defined as:
* rom const unsigned char *rom USB_SD_Ptr[]={&sd000,&sd001,&sd002};
*
* The above declaration has 3 strings, sd000, sd001, and sd002.
* Strings can be removed or added. sd000 is a specialized string
* descriptor. It defines the language code, usually this is
* US English (0x0409). The index of the string must match the index
* position of the USB_SD_Ptr array, &sd000 must be in position
* USB_SD_Ptr[0], &sd001 must be in position USB_SD_Ptr[1] and so on.
* The look-up table USB_SD_Ptr is used by the get string handler
* function in usb9.c.
*
* -----
*
* The look-up table scheme also applies to the configuration
* descriptor. A USB device may have multiple configuration
* descriptors, i.e. CFG01, CFG02, etc. To add a configuration
* descriptor, user must implement a structure similar to CFG01.
* The next step is to add the configuration descriptor name, i.e.
* cfg01, cfg02,..., to the look-up table USB_CD_Ptr. USB_CD_Ptr[0]
* is a dummy place holder since configuration 0 is the un-configured
* state according to the definition in the USB specification.
*
```

## Wireless USB Project Design Report

```

/*****
/*****
* Descriptor specific type definitions are defined in:
* system\usb\usbdefs\usbdefs_std_dsc.h
*
* Configuration information is defined in:
* autofiles\usbcfg.h
*****/

/** I N C L U D E S *****/
#include "system\typesdefs.h"
#include "system\usb\usb.h"

/** C O N S T A N T S *****/
#pragma romdata

/* Device Descriptor */
rom USB_DEV_DSC device_dsc=
{
    sizeof(USB_DEV_DSC),    // Size of this descriptor in bytes
    DSC_DEV,                // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    EP0_BUFF_SIZE,         // Max packet size for EP0, see usbcfg.h
    0x04D8,                 // VENDOR ID sublicensed to DT5 from Microchip
    0xFF44,                 // PRODUCT ID sublicensed to DT5 from Microchip
    0x0001,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string index
    0x01,                   // Number of possible configurations
};

/* Configuration 1 Descriptor */
CFG01={
    /* Configuration Descriptor */
    sizeof(USB_CFG_DSC),    // Size of this descriptor in bytes
    DSC_CFG,                // CONFIGURATION descriptor type
    sizeof(cfg01),          // Total length of data for this cfg
    1,                      // Number of interfaces in this cfg
    1,                      // Index value of this configuration
    0,                      // Configuration string index
    _DEFAULT|_SELF,        // Attributes, see usbdefs_std_dsc.h
    50,                     // Max power consumption (2X mA)

    /* Interface Descriptor */
    sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes
    DSC_INTF,               // INTERFACE descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
    MSD_INTF,               // Class code
    MSD_INTF_SUBCLASS,      // Subclass code
    MSD_PROTOCOL,           // Protocol code
    0,                      // Interface string index

    /* Endpoint Descriptor */
    sizeof(USB_EP_DSC),DSC_EP,_EP01_IN,_BULK,MSD_IN_EP_SIZE,0x00,
    sizeof(USB_EP_DSC),DSC_EP,_EP01_OUT,_BULK,MSD_OUT_EP_SIZE,0x00
};

rom struct{byte bLength;byte bDscType;word string[1];}sd000={
    sizeof(sd000),DSC_STR,0x0409};

rom struct{byte bLength;byte bDscType;word string[25];}sd001={
    sizeof(sd001),DSC_STR,
    'M','i','c','r','o','c','h','i','p',' ',
    'T','e','c','h','n','o','l','o','g','y',' ',',','I','n','c','.'};

```

## Wireless USB Project Design Report

```
rom struct{byte bLength;byte bDscType;word string[31];}sd002={
sizeof(sd002),DSC_STR,
'D','T','5',' ','W','i','r','e','l','e','s','s',' ','M','a','s','s',' ','
'S','t','o','r','a','g','e',' ','D','r','i','v','e'};

rom const unsigned char *rom USB_CD_Ptr[]={&cfg01,&cfg01};
rom const unsigned char *rom USB_SD_Ptr[]={&sd000,&sd001,&sd002};

rom pFunc ClassReqHandler[1]=
{
    &USBCheckMSDRequest
};

#pragma code

/** EOF usbdsc.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usbdsc.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:                   Descriptor specific type definitions are defined in:
/                               system\usb\usbdefs\usbdefs_std_dsc.h
/
/*****
*****/

#ifndef USBDSC_H
#define USBDSC_H

/** I N C L U D E S *****/
#include "system\typedefs.h"
#include "autofiles\usbcfg.h"

#if defined(USB_USE_MSD)
#include "system\usb\class\msd\msd.h"
#endif

#include "system\usb\usb.h"

/** D E F I N I T I O N S *****/
#define CFG01 rom struct
{
    USB_CFG_DSC          cd01;
    USB_INTF_DSC         i00a00;
    USB_EP_DSC           ep01i_i00a00;
    USB_EP_DSC           ep01o_i00a00;
} cfg01

/** E X T E R N S *****/
extern CFG01;
extern rom USB_DEV_DSC device_dsc;
extern rom const unsigned char *rom USB_CD_Ptr[];
extern rom const unsigned char *rom USB_SD_Ptr[];

//extern rom struct{byte report[HID_RPT01_SIZE];} hid_rpt01;
extern rom pFunc ClassReqHandler[1];

#endif //USB_DSC_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbcfg.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.5.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#ifndef USBCFG_H
#define USBCFG_H

/** D E F I N I T I O N S *****/
#define EP0_BUFF_SIZE      16    // 8, 16, 32, or 64
#define MAX_NUM_INT        1    // For tracking Alternate Setting

/* Parameter definitions are defined in usbdrv.h */
#define MODE_PP              _PPBMO                                // ping pong buffer
mode 0
#define UCFG_VAL              _PUEN|_TRINT|_FS|MODE_PP            // internal pull-up resistor,
internal transceiver, FS USB mode, ppong b0

#define USE_SELF_POWER_SENSE_IO
#define USE_USB_BUS_SENSE_IO

/** D E V I C E   C L A S S   U S A G E *****/
#define USB_USE_MSD

/*
 * MUID = Microchip USB Class ID
 * Used to identify which of the USB classes owns the current
 * session of control transfer over EP0
 */
#define MUID_NULL              0
#define MUID_USB9              1
#define MUID_HID               2
#define MUID_CDC               3
#define MUID_MSD                4

/** E N D P O I N T S   A L L O C A T I O N *****/
* See usbmmmap.c for an explanation of how the endpoint allocation works
*/

/* MSD*/
#define MSD_INTF_ID            0x00
#define MSD_UEP                UEP1
#define MSD_BD_OUT              ep1Bo
#define MSD_OUT_EP_SIZE        64                                /*Max endpoint size for F/S since
4550 cannot do H/S*/
#define MSD_BD_IN                ep1Bi
#define MSD_IN_EP_SIZE          64
#define MSD_NUM_OF_DSC          1

/* MSD macros */
//Because we don't have a descriptor for MSD class separately and msd_i00a00 is not
defined.
//#define mUSBGetMSDDscAdr(ptr)
//{
//    if(usb_active_cfg == 1)
//        ptr = (rom byte*)&cfg01.msd_i00a00;
//}

//#define mUSBGetHIDRptDscAdr(ptr)    \
//{                                  \
//    if(usb_active_cfg == 1)          \

```

## Wireless USB Project Design Report

```
//      ptr = (rom byte*)&hid_rpt01;      \
//}
//
//#define mUSBGetHIDRptDscSize(count)      \
//{                                         \
//      if(usb_active_cfg == 1)             \
//          count = sizeof(hid_rpt01);      \
//}

#define MAX_EP_NUMBER 1                    // UEPl

#endif //USBCFG_H
```

## Wireless USB Project Design Report

```

/*****
* PIC USB
* interrupt.h
*****/

#ifndef INTERRUPT_H
#define INTERRUPT_H

/** I N C L U D E S *****/
#include "system\typedefs.h"

/** D E F I N I T I O N S *****/
#define mEnableInterrupt()          INTCONbits.GIE = 1;

/** S T R U C T U R E S *****/

/** E X T E R N S *****/

/** P R O T O T Y P E S *****/
void low_isr(void);
void high_isr(void);

#endif //INTERRUPT_H

```

## Wireless USB Project Design Report

```

/*****
 * PIC USB
 * interrupt.c
 *****/

/** I N C L U D E S *****/
#include <pl8cxxx.h>
#include "system/typedefs.h"
#include "system/interrupt/interrupt.h"

/** V A R I A B L E S *****/

/** I N T E R R U P T   V E C T O R S *****/

#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    _asm goto high_isr _endasm
}
#pragma code

#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
{
    _asm goto low_isr _endasm
}
#pragma code

/** D E C L A R A T I O N S *****/
/*****
 * Function:      void high_isr(void)
 * PreCondition:  None
 * Input:
 * Output:
 * Side Effects:
 * Overview:
 *****/
#pragma interrupt high_isr
void high_isr(void)
{
}

/*****
 * Function:      void low_isr(void)
 * PreCondition:  None
 * Input:
 * Output:
 * Side Effects:
 * Overview:
 *****/
#pragma interruptlow low_isr
void low_isr(void)
{
}
#pragma code

/** EOF interrupt.c *****/
```



## Wireless USB Project Design Report

```
#ifndef HID_H
#define HID_H

/** I N C L U D E S *****/
#include "system\typedefs.h"

/** D E F I N I T I O N S *****/

/* Class-Specific Requests */
#define GET_REPORT      0x01
#define GET_IDLE        0x02
#define GET_PROTOCOL    0x03
#define SET_REPORT      0x09
#define SET_IDLE        0x0A
#define SET_PROTOCOL    0x0B

/* Class Descriptor Types */
#define DSC_HID          0x21
#define DSC_RPT          0x22
#define DSC_PHY          0x23

/* Protocol Selection */
#define BOOT_PROTOCOL    0x00
#define RPT_PROTOCOL    0x01

/* HID Interface Class Code */
#define HID_INTF         0x03

/* HID Interface Class SubClass Codes */
#define BOOT_INTF_SUBCLASS 0x01

/* HID Interface Class Protocol Codes */
#define HID_PROTOCOL_NONE 0x00
#define HID_PROTOCOL_KEYBOARD 0x01
#define HID_PROTOCOL_MOUSE 0x02

/*****
 * Macro:          (bit) mHIDRxIsBusy(void)
 * Overview:       This macro is used to check if HID OUT endpoint is
 *                 busy (owned by SIE) or not.
 *                 Typical Usage: if(mHIDRxIsBusy())
 *
 * Note:           None
 *****/
#define mHIDRxIsBusy()      HID_BD_OUT.Stat.UOWN

/*****
 * Macro:          (bit) mHIDTxIsBusy(void)
 * Overview:       This macro is used to check if HID IN endpoint is
 *                 busy (owned by SIE) or not.
 *                 Typical Usage: if(mHIDTxIsBusy())
 *
 * Note:           None
 *****/
#define mHIDTxIsBusy()      HID_BD_IN.Stat.UOWN

/*****
 * Macro:          byte mHIDGetRptRxLength(void)
 * Output:         mHIDGetRptRxLength returns hid_rpt_rx_len
 * Overview:       mHIDGetRptRxLength is used to retrieve the number of bytes
 *                 copied to user's buffer by the most recent call to
 *                 HIDRxReport function.
 *
 * Note:           None
 *****/
#define mHIDGetRptRxLength()  hid_rpt_rx_len

/** S T R U C T U R E S *****/
typedef struct _USB_HID_DSC_HEADER
```

## Wireless USB Project Design Report

```
{
    byte bDscType;
    word wDscLength;
} USB_HID_DSC_HEADER;

typedef struct _USB_HID_DSC
{
    byte bLength;          byte bDscType;          word bcdHID;
    byte bCountryCode;     byte bNumDsc;
    USB_HID_DSC_HEADER hid_dsc_header[HID_NUM_OF_DSC];
    /*
     * HID_NUM_OF_DSC is defined in autofiles\usbcfg.h
     */
} USB_HID_DSC;

/** E X T E R N S *****/
extern byte hid_rpt_rx_len;

/** P U B L I C   P R O T O T Y P E S *****/
void HIDInitEP(void);
void USBCheckHIDRequest(void);
void HIDTxReport(char *buffer, byte len);
byte HIDRxReport(char *buffer, byte len);

#endif //HID_H
```

## Wireless USB Project Design Report

```
/** I N C L U D E S *****/
#include <p18cxxx.h>

#include "system\usb\usb.h"

#ifdef USB_USE_HID

/** V A R I A B L E S *****/
#pragma udata
byte idle_rate;
byte active_protocol;           // [0] Boot Protocol [1] Report Protocol
byte hid_rpt_rx_len;

/** P R I V A T E   P R O T O T Y P E S *****/
void HIDGetReportHandler(void);
void HIDSetReportHandler(void);

/** D E C L A R A T I O N S *****/
#pragma code

/** C L A S S   S P E C I F I C   R E Q *****/
/*****
 * Function:      void USBCheckHIDRequest(void)
 * Overview:      This routine checks the setup data packet to see if it
 *                knows how to handle it
 *****/
void USBCheckHIDRequest(void)
{
    if(SetupPkt.Recipient != RCPT_INTF) return;
    if(SetupPkt.bIntfID != HID_INTF_ID) return;

    /*
     * There are two standard requests that hid.c may support.
     * 1. GET_DSC(DSC_HID,DSC_RPT,DSC_PHY);
     * 2. SET_DSC(DSC_HID,DSC_RPT,DSC_PHY);
     */
    if(SetupPkt.bRequest == GET_DSC)
    {
        switch(SetupPkt.bDscType)
        {
            case DSC_HID:
                ctrl_trf_session_owner = MUID_HID;
                mUSBGetHIDDscAdr(pSrc.bRom);           // See usbcfg.h
                wCount._word = sizeof(USB_HID_DSC);
                break;
            case DSC_RPT:
                ctrl_trf_session_owner = MUID_HID;
                mUSBGetHIDRptDscAdr(pSrc.bRom);         // See usbcfg.h
                mUSBGetHIDRptDscSize(wCount._word);    // See usbcfg.h
                break;
            case DSC_PHY:
                // ctrl_trf_session_owner = MUID_HID;
                break;
        } //end switch(SetupPkt.bDscType)
        usb_stat.ctrl_trf_mem = _ROM;
    } //end if(SetupPkt.bRequest == GET_DSC)

    if(SetupPkt.RequestType != CLASS) return;
    switch(SetupPkt.bRequest)
    {
        case GET_REPORT:
            HIDGetReportHandler();
            break;
        case SET_REPORT:
            HIDSetReportHandler();
            break;
        case GET_IDLE:
            ctrl_trf_session_owner = MUID_HID;
            pSrc.bRam = (byte*)&idle_rate;           // Set source
            usb_stat.ctrl_trf_mem = _RAM;              // Set memory type
    }
}
```

## Wireless USB Project Design Report

```
        LSB(wCount) = 1;                                // Set data count
        break;
    case SET_IDLE:
        ctrl_trf_session_owner = MUID_HID;
        idle_rate = MSB(SetupPkt.W_Value);
        break;
    case GET_PROTOCOL:
        ctrl_trf_session_owner = MUID_HID;
        pSrc.bRam = (byte*)&active_protocol; // Set source
        usb_stat.ctrl_trf_mem = _RAM;        // Set memory type
        LSB(wCount) = 1;                    // Set data count
        break;
    case SET_PROTOCOL:
        ctrl_trf_session_owner = MUID_HID;
        active_protocol = LSB(SetupPkt.W_Value);
        break;
} //end switch(SetupPkt.bRequest)

} //end USBCheckHIDRequest

void HIDGetReportHandler(void)
{
    // ctrl_trf_session_owner = MUID_HID;
} //end HIDGetReportHandler

void HIDSetReportHandler(void)
{
    // ctrl_trf_session_owner = MUID_HID;
    // pDst.bRam = (byte*)&hid_report_out;
} //end HIDSetReportHandler

/** U S E R   A P I *****/
/*****
 * Function:      void HIDInitEP(void)
 * Overview:      HIDInitEP initializes HID endpoints, buffer descriptors,
 *                internal state-machine, and variables.
 *                It should be called after the USB host has sent out a
 *                SET_CONFIGURATION request.
 *                See USBStdSetCfgHandler() in usb9.c for examples.
 *****/
void HIDInitEP(void)
{
    hid_rpt_rx_len = 0;

    HID_UEP = EP_OUT_IN|HSHK_EN;                // Enable 2 data pipes

    HID_BD_OUT.Cnt = sizeof(hid_report_out);    // Set buffer size
    HID_BD_OUT.ADR = (byte*)&hid_report_out;    // Set buffer address
    HID_BD_OUT.Stat._byte = _USIE|_DAT0|_DTSEN; // Set status

    /*
     * Do not have to init Cnt of IN pipes here.
     * Reason:  Number of bytes to send to the host
     *          varies from one transaction to
     *          another. Cnt should equal the exact
     *          number of bytes to transmit for
     *          a given IN transaction.
     *          This number of bytes will only
     *          be known right before the data is
     *          sent.
     */
    HID_BD_IN.ADR = (byte*)&hid_report_in;      // Set buffer address
    HID_BD_IN.Stat._byte = _UCPU|_DAT1;         // Set status
} //end HIDInitEP

/*****
 * Function:      void HIDTxReport(char *buffer, byte len)
 * PreCondition:  mHIDTxIsBusy() must return false.
 *                Value of 'len' must be equal to or smaller than

```

## Wireless USB Project Design Report

```
*
*          HID_INT_IN_EP_SIZE
*          For an interrupt endpoint, the largest buffer size is
*          64 bytes.
* Input:    buffer : Pointer to the starting location of data bytes
*          len     : Number of bytes to be transferred
* Overview: Use this macro to transfer data located in data memory.
*
*          Remember: mHIDTxIsBusy() must return false before user
*          can call this function.
*          Unexpected behavior will occur if this function is called
*          when mHIDTxIsBusy() == 0
*
*          Typical Usage:
*          if(!mHIDTxIsBusy())
*              HIDTxReport(buffer, 3);
*          *****/
void HIDTxReport(char *buffer, byte len)
{
    byte i;

    /*
    * Value of len should be equal to or smaller than HID_INT_IN_EP_SIZE.
    * This check forces the value of len to meet the precondition.
    */
    if(len > HID_INT_IN_EP_SIZE)
        len = HID_INT_IN_EP_SIZE;

    /*
    * Copy data from user's buffer to dual-ram buffer
    */
    for (i = 0; i < len; i++)
        hid_report_in[i] = buffer[i];

    HID_BD_IN.Cnt = len;
    mUSBBufferReady(HID_BD_IN);
}

//end HIDTxReport

/*****
* Function:    byte HIDRxReport(char *buffer, byte len)
*
* PreCondition: Value of input argument 'len' should be smaller than the
*               maximum endpoint size responsible for receiving report
*               data from USB host for HID class.
*               Input argument 'buffer' should point to a buffer area that
*               is bigger or equal to the size specified by 'len'.
*
* Input:       buffer : Pointer to where received bytes are to be stored
*               len     : The number of bytes expected.
*
* Output:      The number of bytes copied to buffer.
*
* Side Effects: Publicly accessible variable hid_rpt_rx_len is updated
*               with the number of bytes copied to buffer.
*               Once HIDRxReport is called, subsequent retrieval of
*               hid_rpt_rx_len can be done by calling macro
*               mHIDGetRptRxLength().
*
* Overview:    HIDRxReport copies a string of bytes received through
*               USB HID OUT endpoint to a user's specified location.
*               It is a non-blocking function. It does not wait
*               for data if there is no data available. Instead it returns
*               '0' to notify the caller that there is no data available.
*
* Note:        If the actual number of bytes received is larger than the
*               number of bytes expected (len), only the expected number
*               of bytes specified will be copied to buffer.
*               If the actual number of bytes received is smaller than the
*               number of bytes expected (len), only the actual number
*               of bytes received will be copied to buffer.
* *****/
```

## Wireless USB Project Design Report

```
byte HIDRxReport(char *buffer, byte len)
{
    hid_rpt_rx_len = 0;

    if(!mHIDRxIsBusy())
    {
        /*
         * Adjust the expected number of bytes to equal
         * the actual number of bytes received.
         */
        if(len > HID_BD_OUT.Cnt)
            len = HID_BD_OUT.Cnt;

        /*
         * Copy data from dual-ram buffer to user's buffer
         */
        for(hid_rpt_rx_len = 0; hid_rpt_rx_len < len; hid_rpt_rx_len++)
            buffer[hid_rpt_rx_len] = hid_report_out[hid_rpt_rx_len];

        /*
         * Prepare dual-ram buffer for next OUT transaction
         */
        HID_BD_OUT.Cnt = sizeof(hid_report_out);
        mUSBBufferReady(HID_BD_OUT);
    } //end if

    return hid_rpt_rx_len;
} //end HIDRxReport

#endif //def USB_USE_HID

/** EOF hid.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          msd.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.7.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

#ifdef MSD_H
#define MSD_H

/** I N C L U D E S *****/
#include "system\typedefs.h"
#include "io_cfg.h"          // I/O pin mapping

/** D E F I N I T I O N S *****/

/* MSD Interface Class Code */
#define MSD_INTF              0x08

/* MSD Interface Class SubClass Codes */
#define MSD_INTF_SUBCLASS    0x06

/* MSD Interface Class Protocol Codes */
#define MSD_PROTOCOL         0x50

/* Class Commands */
#define MSD_RESET 0xff
#define GET_MAX_LUN 0xfe

#define BLOCKLEN_512          0x0200

#define STMSDTRIS TRISD0
#define STRUNTRIS TRISD1
#define STMSDLED LATDbits.LATD0
#define STRUNLED LATDbits.LATD1
#define ToggleRUNLED() STRUNLED = !STRUNLED;

/* SCSI Transparent Command Set Sub-class code */
#define INQUIRY                0x12
#define READ_FORMAT_CAPACITY   0x23
#define READ_CAPACITY          0x25
#define READ_10                0x28
#define WRITE_10               0x2a
#define REQUEST_SENSE          0x03
#define MODE_SENSE             0x1a
#define PREVENT_ALLOW_MEDIUM_REMOVAL 0x1e
#define TEST_UNIT_READY        0x00
#define VERIFY                  0x2f
#define STOP_START              0x1b

/* Various States of Mass Storage Firmware */
#define MSD_WAIT 0              // Waiting for a valid Command Block Wrapper (CBW)
#define MSD_DATA_IN 2           // IN Data State ( Device-> Host)
#define MSD_DATA_OUT 3         // OUT Data State (Host -> Device)

#define MSD_CSW_SIZE 0x0d       // 10 bytes CSW data
#define MSD_CBW_SIZE 0x1f       // 31 bytes CBW data

#define INVALID_CBW 1
#define VALID_CBW !INVALID_CBW
#define MAX_LUN 0

/* Sense Key Error Codes */
#define S_NO_SENSE 0x0

```

## Wireless USB Project Design Report

```
#define S_RECOVERED_ERROR 0x1
#define S_NOT_READY 0x2
#define S_MEDIUM_ERROR 0x3
#define S_HARDWARE_ERROR 0x4
#define S_ILLEGAL_REQUEST 0x5
#define S_UNIT_ATTENTION 0x6
#define S_DATA_PROTECT 0x7
#define S_BLANK_CHECK 0x8
#define S_VENDOR_SPECIFIC 0x9
#define S_COPY_ABORTED 0xa
#define S_ABORTED_COMMAND 0xb
#define S_OBSOLETE 0xc
#define S_VOLUME_OVERFLOW 0xd
#define S_MISCOMPARE 0xe
#define S_CURRENT 0x70
#define S_DEFERRED 0x71

/* ASC ASCQ Codes for Sense Data (only those that we plan to use) */
// with sense key Illegal request for a command not supported
#define ASC_INVALID_COMMAND_OPCODE 0x20
#define ASCQ_INVALID_COMMAND_OPCODE 0x00

// from SPC-3 Table 185
// with sense key Illegal Request for test unit ready
#define ASC_LOGICAL_UNIT_NOT_SUPPORTED 0x25
#define ASCQ_LOGICAL_UNIT_NOT_SUPPORTED 0x00

// with sense key Not ready
#define ASC_LOGICAL_UNIT_DOES_NOT_RESPOND 0x05
#define ASCQ_LOGICAL_UNIT_DOES_NOT_RESPOND 0x00

#define ASC_MEDIUM_NOT_PRESENT 0x3a
#define ASCQ_MEDIUM_NOT_PRESENT 0x00

#define ASC_LOGICAL_UNIT_NOT_READY_CAUSE_NOT_REPORTABLE 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_CAUSE_NOT_REPORTABLE 0x00

#define ASC_LOGICAL_UNIT_IN_PROCESS 0x04
#define ASCQ_LOGICAL_UNIT_IN_PROCESS 0x01

#define ASC_LOGICAL_UNIT_NOT_READY_INIT_REQD 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_INIT_REQD 0x02

#define ASC_LOGICAL_UNIT_NOT_READY_INTERVENTION_REQD 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_INTERVENTION_REQD 0x03

#define ASC_LOGICAL_UNIT_NOT_READY_FORMATTING 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_FORMATTING 0x04

#define ASC_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE 0x21
#define ASCQ_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE 0x00

#define ASC_WRITE_PROTECTED 0x27
#define ASCQ_WRITE_PROTECTED 0x00

/*****
 * Macro:          (bit) mMSDRxIsBusy(void)
 * Overview:       This macro is used to check if MSD OUT endpoint is
 *                 busy (owned by SIE) or not.
 *                 Typical Usage: if(mMSDRxIsBusy())
 *****/
#define mMSDRxIsBusy() MSD_BD_OUT.Stat.UOWN

/*****
 * Macro:          (bit) mMSDTxIsBusy(void)
 * Overview:       This macro is used to check if MSD IN endpoint is
 *                 busy (owned by SIE) or not.
 *                 Typical Usage: if(mMSDTxIsBusy())
 *****/
#define mMSDTxIsBusy() MSD_BD_IN.Stat.UOWN
```



## Wireless USB Project Design Report

```

/*****
 * Macro:          (bit) mMin(void)
 * Overview:       This macro is used to find the lower of two arguments
 *                Typical Usage: mMin(A, B)
 *****/
#define mMin(A,B) (A<B)?A:B

/** S T R U C T U R E S *****/
typedef struct _USB_MSD_CBW //31 bytes total Command Block Wrapper
{
    dword dCBWSignature; // 55 53 42 43h
    dword dCBWTag; // sent by host, device echos this
    value in CSW (associated a CSW with a CBW)
    dword dCBWDataTransferLength; // number of bytes of data host expects to transfer
    byte bCBWFlags; // CBW flags, bit 7 = 0-data out from host
    to device, //
    //
    = 1-device to host, rest bits 0
    byte bCBWLUN; // Most Significant 4bits are always
    zero, 0 in our case as only one logical unit
    byte bCBWCBLLength; // Here most significant 3bits are zero
    byte CBWCBL[16]; // Command block to be executed by the
    device
} USB_MSD_CBW;

typedef struct { // Command Block for Read 10 (0x28)&
Write 10 (0x2a)commands
    byte Opcode;
    byte Flags; // b7-b5 RDProtect, b4 DPO,
    b3 FUA, b2 Reserved, b1 FUA_NV, b0 Obsolete
    DWORD LBA; //
    byte GroupNumber; // b4-b0 is Group Number rest are
    reserved
    WORD TransferLength;
    byte Control;
} ReadWriteCB;

typedef struct { // Inquiry command format
    byte Opcode;
    byte EVPD; // only b0 is enable vital
    product data
    byte PageCode;
    word AllocationLength;
    byte Control;
} InquiryCB;

typedef struct { // Read Capacity 10
    byte Opcode;
    byte Reserved1;
    dword LBA; // Logical Block Address
    word Reserved2;
    byte PMI; // Partial medium Indicator
    b0 only
    byte Control;
} ReadCapacityCB;

typedef struct { // Request Sense 0x03
    byte Opcode;
    byte Desc;
    word Reserved;
    byte AllocationLength;
    byte Control;
} RequestSenseCB;

typedef struct { // Mode Sense 0x1a
    byte Opcode;
    byte DBD; // actually only b3 is used
    as disable block descriptor
    byte PageCode; // b7,b6 PC=Page Control, b5-b0
    PageCode
}
```

## Wireless USB Project Design Report

```
// Page Control bits
00=> CurrentValue, 01=>Changeable Values,10=>Default Value, 11=>Saved Values
    byte SubPageCode;
    byte AllocationLength;
    byte Control;
} ModeSenseCB;

typedef struct {                                // PREVENT_ALLOW_MEDIUM_REMOVAL 0x1e
    byte Opcode;
    byte Reserved[3];
    byte Prevent;                                // only b1-b0 is prevent, rest
reserved
    byte Control;
} PreventAllowMediumRemovalCB;

typedef struct {                                // TEST_UNIT_READY 0x00
    byte Opcode;
    dword Reserved;
    byte Control;
} TestUnitReadyCB;

typedef struct {                                // VERIFY 10 Command 0x2f
    byte Opcode;
    byte VRProtect;                             // b7-b5 VRProtect, b4 DPO,
b3-b2,Reserved, b1 BYCHK, b0 Obsolete
    dword LBA;
    byte GroupNumber;                           // b4-b0 Group Number, rest reserved
    word VerificationLength;
    byte Control;
} VerifyCB;

typedef struct {                                // STOP_START 0x1b
    byte Opcode;
    byte Immed;
    word Reserved;
    byte Start;                                 // b7-b4 PowerCondition, b3-
b2reserved, b1 LOEJ, b0 Start
    byte Control;
} StopStartCB;

typedef struct _USB_MSD_CSW                    // Command Status Wrapper
{
    dword dCSWSignature;                        // 55 53 42 53h Signature of a CSW packet
    dword dCSWTag;                             // echo the dCBWTag of the CBW packet
    dword dCSWDataResidue;                     // difference in data expected
(dCBWDataTransferLength) and actual amount processed/sent
    byte bCSWStatus;                           // 00h Command Passed, 01h Command
Failed, 02h Phase Error, rest obsolete/reserved
} USB_MSD_CSW;

typedef struct
{
    byte Peripheral;                            // Peripheral_Qualifier:3;
Peripheral_DevType:5;
    byte Removable;                            // removable medium bit7 = 0
means non removable, rest reserved
    byte Version;                             // version
    byte Response_Data_Format;                 // b7,b6 Obsolete, b5 Access control
co-ordinator, b4 hierarchical addressing support
// b3:0
response data format 2 indicates response is in format defined by spec
    byte AdditionalLength;                     // length in bytes of remaining in
standard inquiry data
    byte Scstp;                                // b7 SCCS, b6 ACC, b5-b4
TGPS, b3 3PC, b2-b1 Reserved, b0 Protected
    byte bqueetc;                             // b7 bque, b6- EncServ, b5-
VS, b4-MultiP, b3-MChngr, b2-b1 Obsolete, b0-Addr16
    byte CmdQue;                              // b7-b6 Obsolete, b5-WBUS, b4-Sync, b3-Linked,
b2 Obsolete,b1 Cmdque, b0-VS
    char vendorID[8];
}
```

## Wireless USB Project Design Report

```
        char productID[16];
        char productRev[4];
    } InquiryResponse;

typedef struct {
    byte ModeDataLen;
    byte MediumType;
    unsigned Resv:4;
    unsigned DPOFUA:1; // 0 indicates DPO and FUA
bits not supported
    unsigned notused:2;
    unsigned WP:1; // 0 indicates not write
protected
    byte BlockDscLen; // Block Descriptor Length
} tModeParamHdr;

/* Short LBA mode block descriptor (see Page 1009, SBC-2) */
typedef struct {
    byte NumBlocks[4];
    byte Resv; // reserved
    byte BlockLen[3];
} tBlockDescriptor;

/* Page_0 mode page format */
typedef struct {
    unsigned PageCode:6; // SPC-3 7.4.5
    unsigned SPF:1; // SubPageFormat=0
means Page_0 format
    unsigned PS:1; // Parameters Saveable

    byte PageLength; // if 2..n bytes of mode
parameters PageLength = n-1
    byte ModeParam[]; // mode parameters
} tModePage;

typedef struct {
    tModeParamHdr Header;
    tBlockDescriptor BlockDsc;
    tModePage modePage;
} ModeSenseResponse;

/* Fixed format if Desc bit of request sense cbw is 0 */
typedef union {
    struct
    {
        byte _byte[18];
    };
    struct {
        unsigned ResponseCode:7; // b6-b0 is Response Code
Fixed or descriptor format
        unsigned VALID:1; // Set to 1 to
indicate information field is a valid value

        byte Obsolete;

        unsigned SenseKey:4; // Refer SPC-3 Section 4.5.6
        unsigned Resv:1;
        unsigned ILI:1; // Incorrect
Length Indicator
        unsigned EOM:1; // End of
Medium
        unsigned FILEMARK:1; // for READ and SPACE
commands

        DWORD Information; // device type or
command specific (SPC-33.1.18)
        byte AddSenseLen; // number of
additional sense bytes that follow <=244
        DWORD CmdSpecificInfo; // depends on command on
which exception occurred
    };
};
```

## Wireless USB Project Design Report

```

        byte ASC;                                // additional
sense code
        byte ASCQ;                                // additional
sense code qualifier Section 4.5.2.1 SPC-3
        byte FRUC;                                // Field
Replaceable Unit Code 4.5.2.5 SPC-3

        byte SenseKeySpecific[3];                // msb is SKSV sense-key
specific valied field set=> valid SKS
                                                    // 18-n
additional sense bytes can be defined later
                                                    // 18
Bytes Request Sense Fixed Format
    };
} RequestSenseResponse;

/** E X T E R N S *****/
extern CSD gblCSDReg;                                // declared in sdcard.c

/** P U B L I C   P R O T O T Y P E S *****/
void USBCheckMSDRequest(void);
void ProcessIO(void);
void SDCardInit(void);
void MSDInitEP(void);
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                msd.c
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.7.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

/** I N C L U D E S *****/
#include <pl18f4550.h>
#include "system\typedefs.h"
#include "system\usb\usb.h"
#include <string.h>

#ifdef USB_USE_MSD

/** V A R I A B L E S *****/
#pragma udata
byte MSD_State;                // Takes values MSD_WAIT, MSD_DATA_IN or
                                MSD_DATA_OUT
USB_MSD_CBW gblCBW;
byte gblCBWLength;
SDCSTATE gblFlag;             //SDC
RequestSenseResponse gblSenseData;
byte *ptrNextData;

CSD gblCSDReg;                //Moved from sdcard.h

/*
 * Number of Blocks and Block Length are global because
 * for every READ_10 and WRITE_10 command need to verify if the last LBA
 * is less than gblNumBLKS
 */
DWORD gblNumBLKS=0x00,gblBLKLen=0x00;

/* Standard Response to INQUIRY command stored in ROM */
const rom InquiryResponse inq_resp = {
    0x00,          // peripheral device is connected, direct access block device
    0x80,          // removable
    0x04,          // version = 00=> does not conform to any standard, 4=> SPC-2
    0x02,          // response is in format specified by SPC-2
    0x20,          // n-4 = 36-4=32= 0x20
    0x00,          // sccs etc.
    0x00,          // bque=1 and cmdque=0,indicates simple queueing 00 is obsolete,
                    // but as in case of other device, we are just using 00
    0x00,          // 00 obsolete, 0x80 for basic task queueing
    "Microchip",// this is the T10 assigned Vendor ID
    "Mass Storage",
    "0001"
};

/** P R I V A T E   P R O T O T Y P E S *****/
void MSDCommandHandler(void);
void MSDInquiryHandler(void);
void MSDReadCapacityHandler(void);
void MSDReadHandler(void);
void MSDWriteHandler(void);
void MSDModeSenseHandler(void);
void MSDMediumRemovalHandler(void);
void MSDRequestSenseHandler(void);
void MSDTestUnitReadyHandler(void);
void MSDVerifyHandler(void);
void MSDStopStartHandler(void);
byte IsMeaningfulCBW(void);
byte IsValidCBW(void);

```

## Wireless USB Project Design Report

```
void PrepareCSWData(void);
void SendData(byte*, byte);
void SendCSW(void);
void ResetSenseData(void);
void MSDDataIn(void);
void MSDDataOut(void);

//extern SDC_Error MediaInitialize(SDCSTATE*);//SDC
//extern void SocketInitialize(void);           //SDC
//extern SDC_Error SectorRead(dword, byte*); //SDC
//extern SDC_Error SectorWrite(dword, byte*);   //SDC
//extern SDC_Error CSDRead(void);               //SDC
//extern int DetectSDCard (void);               //SDC
//extern byte IsWriteProtected(void);           //SDC

/** D E C L A R A T I O N S *****/
#pragma code

/** C L A S S   S P E C I F I C   R E Q *****/
/*****
 * Function:      void USBCheckMSDRequest(void)
 * Overview:      This routine handles the standard RESET and GET_MAX_LUN
 *                  command requests received on the control endpoint EP0
 *****/
void USBCheckMSDRequest(void)
{
    switch(SetupPkt.bRequest)
    {
        case MSD_RESET:
            ctrl_trf_session_owner = MUID_MSD;
            mDisableEP1to15();           // See usbdrv.h
            if (UEPlbits.EPSTALL==1) {
                UEPlbits.EPSTALL = 0;
                MSDInitEP();
            }
            UIRbits.STALLIF = 0;
            break;
        case GET_MAX_LUN:
            ctrl_trf_session_owner = MUID_MSD;
            CtrlTrfData._byte[0] = MAX_LUN;
            wCount._word = 1;
            pSrc.bRam = (unsigned char*)&CtrlTrfData;
            usb_stat.ctrl_trf_mem = _RAM;
            break;
    } //end switch(SetupPkt.bRequest)
}

/*****
 * Function:      void ProcessIO(void)
 *
 * PreCondition:  MSDInitEP() and SDCardInit() have beed called.
 *                  MSDInitEP() is called from
USBStdSetCfgHandler(void)(usb9.c)
 *                  NOT THIS - SDCardInit() is called from InitializeSystem()
in main.c
 *
 * Overview:      This routine is called in continuous loop from main.c
 *                  All the Bulk Transport Commands on EndPoint 1 are
 *                  handled here. MSD_State holds the current state of the
 *                  Mass Storage Module.
 *                  In MSD_WAIT State - Wait for a Command Block Wrapper (CBW)
 *                  on EP1. If a valid and meaningful CBW is received,
 *                  depending on the command received MSD_State is changed to
 *                  MSD_DATA_IN if data is to be sent to host (for all
commands
 *                  other than WRITE_10)
 *                  MSD_DATA_OUT if host is expected to send data (only in case
 *                  of WRITE_10). At the end of Data Transfer Command Status
 *                  Wrapper (CSW) is sent by calling SendCSW()
 *****/
void ProcessIO(void)
```

## Wireless USB Project Design Report

```
{
    byte i;
    dword size;
    if (MSD_State==MSD_DATA_IN) {
        /* Send Data to Host */
        if(gblCBW.dCBWDataTransferLength==0)
        {
            /* Finished sending the data send the Status */
            /* SendCSW() send the csw and sets the state to wait */
            SendCSW();
            if ((msd_csw.bCSWStatus==0x00)&&(gblCBW.CBWCb[0]==INQUIRY)) {
                // Turn on the MSD LED when we have successfully
                // responded to the INQUIRY Command
                STMSDLED=1;
            }
        }
        else
        {
            /* Still have data to Send */
            MSDDataIn();
        }
        return;
    }
    if (MSD_State==MSD_DATA_OUT) {
        /* Receive data from Host*/
        if(gblCBW.dCBWDataTransferLength==0) {
            /* Finished receiving the data prepare and send the status */
            if ((msd_csw.bCSWStatus==0x00)&&(msd_csw.dCSWDataResidue!=0))
                msd_csw.bCSWStatus=0x02;
            SendCSW(); // sends the csw and sets the state to wait
        }
        /*
        * Note that MSD_DATA_OUT State is reached only for the WRITE_10 COMMAND
        * Also note that this code is reached in MSD_DATA_OUT State only after
        * we have read the required amount of data from the host
        * This proccssing is done in WriteCommandHandler because we have
        * limited buffer space. We read from host in 64Bytes chunks
        * (size of MSD_BD_OUT), fill the msd_buffer(512B) and write the
        * data into the SDCard
        */
        return;
    }
    if((MSD_BD_OUT.Stat.UOWN==_UCPU) && (MSD_State==MSD_WAIT)) {
        /* If the CPU owns the BD OUT (we)and the MSD_State is WAIT*/
        /* Copy the received cbw into the gblCBW */
        gblCBW.dCBWSignature=msd_cbw.dCBWSignature;
        gblCBW.dCBWTag=msd_cbw.dCBWTag;
        gblCBW.dCBWDataTransferLength=msd_cbw.dCBWDataTransferLength;
        gblCBW.bCBWFlags=msd_cbw.bCBWFlags;
        gblCBW.bCBWLUN=msd_cbw.bCBWLUN;
        gblCBW.bCBWCbLength=msd_cbw.bCBWCbLength; // 3 MSB are zero
        for (i=0;i<msd_cbw.bCBWCbLength;i++)
            gblCBW.CBWCb[i]=msd_cbw.CBWCb[i];
        gblCBWLength=MSD_BD_OUT.Cnt; // Length of CBW
        if (IsValidCBW()) {
            if (IsMeaningfulCBW()) {
                PrepareCSWData();
                /* If direction is device to host*/
                if (gblCBW.bCBWFlags==0x80)
                    MSD_State=MSD_DATA_IN;
                else if (gblCBW.bCBWFlags==0x00) {
                    /* If direction is host to device*/
                    /* prepare to read data in msd_buffer */
                    MSD_BD_OUT.Cnt=MSD_OUT_EP_SIZE;
                    MSD_BD_OUT.ADR=(byte*)&msd_buffer[0];
                    MSD_State=MSD_DATA_OUT;
                }
            }
            /* Decode and process the valid and meaningful CBW received
            */
            MSDCommandHandler();
        }
    }
}
```

## Wireless USB Project Design Report

```
    }
    /* NOTE:
     * In case when the received CBW is not valid or meaningful,
     * one can take action such as Stall the EP1 and go through reset
     * recovery or turn on error LED etc.
     */
}
/*
 * NOTE: Call after every read or write on nonEP0 EP
 * Basically, toggles DTS and gives ownership to SIE
 */
mUSBBufferReady(MSD_BD_OUT);
/* clears the TRNIF */
USBDriverService();
}

}

/*****
 * Function:      void MSDInitEP(void)
 * Overview:      This routine is called from USBStdSetCfgHandler(void)
 *                Initializes the Bulk-In and Bulk-Out endpoints MSD_BD_IN
 *                and MSD_BD_OUT Size = 64B (See usbmmmap.c and
 *                usbdefs_std_dsc.h for endpoint definitions)
 *****/
void MSDInitEP(void)
{
    //mInitAllLEDs();
    MSD_UEP = EP_OUT_IN|HSHK_EN;                // Enable 2 data pipes
    MSD_BD_OUT.Cnt=sizeof(msd_cbw);
    MSD_BD_OUT.ADR=(byte*)&msd_cbw;
    MSD_BD_OUT.Stat._byte = _USIE|_DAT0|_DTSEN;    //usbmmmap.h owner SIE,
                                                // DAT0 expected next,
                                                //data toggle sunc enable

    /*
     * Do not have to init Cnt (size) of IN pipes here.
     * Reason:  Number of bytes to send to the host
     *           varies from one transaction to
     *           another. Cnt should equal the exact
     *           number of bytes to transmit for
     *           a given IN transaction.
     *           This number of bytes will only
     *           be known right before the data is
     *           sent.
     */
    MSD_BD_IN.ADR = (byte*)&msd_buffer[0];        // Set buffer address
    MSD_BD_IN.Stat._byte = _UCPU|_DAT1;            // Set status CPU owns Data1
                                                // expected next

} //end MSDInitEP

/*****
 * Function:      void SDCardInit(void)
 * Side Effects:  gblFlag is updated according to result of Initialization
 *                MSD_State is set to MSD_WAIT
 *
 * Overview:      This routine is called from InitializeSystem() in main.c
 *                It initializes the SD card if there is some error in
 *                initialization all the LEDs are turned ON.
 *                Also, set the MSD_State = MSD_WAIT
 *****/
void SDCardInit(void)
{
    SDC_Error status;
    /*AK SocketInitialize();
    //mInitAllLEDs();
    status=MediaInitialize(&gblFlag);
    if (status) {
        //If there was some error, turn on all leds
        //mLED_1_On(); mLED_2_On(); mLED_3_On(); mLED_4_On();
        gblFlag.isSDMMC=0;
    } else gblFlag.isSDMMC=1;
```



## Wireless USB Project Design Report

```
*/      MSD_State=MSD_WAIT;
}

/*****
* Function:      void MSDCommandHandler(void)
* Overview:      This routine is called from ProcessIO()
*                when the MSD_State = MSD_WAIT. This function decodes the CBW
*                Command and takes appropriate action.If the CBW command is
*                not supported the Sense Data is set, CSW status
*                is set to Command Failed (bCSWStatus=01h)
*                *****/
void MSDCommandHandler(void)      // In reality it is to read from EP1
{
    switch(gblCBW.CBWCb[0]) {
    case INQUIRY:
        MSDInquiryHandler();
        break;
    case READ_CAPACITY:
        MSDReadCapacityHandler();
        break;
    case READ_10:
        MSDReadHandler();
        break;
    case WRITE_10:
        MSDWriteHandler();
        break;
    case REQUEST_SENSE:
        MSDRequestSenseHandler();
        break;
    case MODE_SENSE:
        MSDModeSenseHandler();
        break;
    case PREVENT_ALLOW_MEDIUM_REMOVAL:
        MSDMediumRemovalHandler();
        break;
    case TEST_UNIT_READY:
        MSDTestUnitReadyHandler();
        break;
    case VERIFY:
        MSDVerifyHandler();
        break;
    case STOP_START:
        MSDStopStartHandler();
        break;
    default:
        ResetSenseData();
        gblSenseData.SenseKey=S_ILLEGAL_REQUEST;
        gblSenseData.ASC=ASC_INVALID_COMMAND_OPCODE;
        gblSenseData.ASCQ=ASCQ_INVALID_COMMAND_OPCODE;
        msd_csw.bCSWStatus=0x01;
        msd_csw.dCSWDataResidue=0x00;
        break;
    } // end switch

    ptrNextData=(byte*)&msd_buffer[0];
}

/*****
* Function:      void SendCSW(void)
* Overview:      This function sends the CSW and sets the State to MSD_WAIT
*                It also changes MSD_BD_OUT to point to msd_csw (structure
*                for reading CSW) Note that this was changed in
*                MSD_DATA_OUT state to point to msd_buffer in order to
*                read data from host
*                *****/
void SendCSW(void)
{
    while(mMSDTxIsBusy());
    MSD_BD_IN.ADR=(byte*)&msd_csw;
    MSD_BD_IN.Cnt=MSD_CSW_SIZE;
    mUSBBufferReady(MSD_BD_IN);
}
```

## Wireless USB Project Design Report

```
        USBDriverService();
        MSD_BD_OUT.Cnt=sizeof(msd_cbw);
        MSD_BD_OUT.ADR=(byte*)&msd_cbw;
        // in MSD_DATA_OUT state the address
        // was changed to point to msd_buffer
        MSD_State=MSD_WAIT;
    }

/*****
 * Function:      void SendData(byte* dataAddr, byte dataSize)
 * Overview:      This function sends "dataSize" bytes of data
 *                (< MSD_EP_IN_SIZE) starting at address "dataAddr".
 *****/
void SendData(byte* dataAddr, byte dataSize)
{
    while(mMSDTxIsBusy());
    MSD_BD_IN.ADR=dataAddr;
    MSD_BD_IN.Cnt=dataSize;
    mUSBBufferReady(MSD_BD_IN);
    USBDriverService();
}

/*****
 * Function:      void MSDDataIn(void)
 * Overview:      This function sends 512B of data in msd_buffer to the
 *                host in 64B chunks using MSD_BD_IN. Various conditions
 *                when data to be sent is less than MSD_IN_EP_SIZE and
 *                error condition bCSWStatus = 0x01 are checked. As per
 *                specifications, in case of error 0 filled data of the size
 *                expected by the host dCBWDataTransferLength is sent.
 *****/
void MSDDataIn(void)
{
    byte i;
    dword size;
    //Case (status==0) and (data to be sent > MSD_IN_EP_SIZE)
    if ((msd_csw.bCSWStatus==0x00)&&(msd_csw.dCSWDataResidue>=MSD_IN_EP_SIZE)) {
        //Write next chunk of data to EP Buffer and send
        SendData(ptrNextData,MSD_IN_EP_SIZE);
        gblCBW.dCBWDataTransferLength-= MSD_IN_EP_SIZE;
        msd_csw.dCSWDataResidue-=MSD_IN_EP_SIZE;
        ptrNextData+=MSD_IN_EP_SIZE;
    } else {
        if (msd_csw.bCSWStatus!=0x00) { // error path status!=0
            size=mMin(MSD_IN_EP_SIZE,gblCBW.dCBWDataTransferLength);
            for (i=0;i<size;i++) msd_buffer[i]=0; // prepare 0 data
            if (gblCBW.dCBWDataTransferLength > MSD_IN_EP_SIZE) {
                //Case (status!=0) and (data to be sent > MSD_IN_EP_SIZE)
                //Write next chunk of data to EP Buffer and send
                SendData((byte*)&msd_buffer[0],MSD_IN_EP_SIZE);
                gblCBW.dCBWDataTransferLength -= MSD_IN_EP_SIZE;
                msd_csw.dCSWDataResidue-=MSD_IN_EP_SIZE;
            } else {
                //Case (status!=0) and (data to be sent < MSD_IN_EP_SIZE)
                //write next chunk of data to EP Buffer and send
                SendData((byte*)&msd_buffer[0],gblCBW.dCBWDataTransferLength);
                gblCBW.dCBWDataTransferLength = 0;
                //we have sent 0s for what was expected by host
                msd_csw.dCSWDataResidue -= gblCBW.dCBWDataTransferLength;
            }
        } else {
            //Case (status==0) and (data to be sent < MSD_IN_EP_SIZE)
            //write next chunk of data to EP Buffer and send
            SendData(ptrNextData,msd_csw.dCSWDataResidue);
            //we have sent all the data that was expected by host
            gblCBW.dCBWDataTransferLength -= msd_csw.dCSWDataResidue ;
            msd_csw.dCSWDataResidue = gblCBW.dCBWDataTransferLength;
            //In case the host expected more than what we had to send
            //Setting DataTransferLength=0 so that CSW is sent after this
        }
    }
}
```

## Wireless USB Project Design Report

```
gblCBW.dCBWDataTransferLength = 0;

    }
}

/*****
 * Function:      void IsValidCBW()
 * Overview:      This checks if the received CBW is valid
 *                  According to the Mass Storage Class Specifications,
 *                  a CSW is considered to be valid if
 *                  1. It was received in MS_WAIT State
 *                  2. CBW length is 1Fh bytes (MSD_CBW_SIZE)
 *                  3. dCBWSignature is equal to 0x43425355h
 *****/
byte IsValidCBW()
{
    if ((gblCBWLength!=MSD_CBW_SIZE)|| (gblCBW.dCBWSignature!=0x43425355))return FALSE;
    else return TRUE;
}

/*****
 * Function:      void IsMeaningfulCBW()
 * Overview:      This checks if the received CBW is meaningful
 *                  According to the Mass Storage Class Specifications,
 *                  a CSW is considered to be meaningful if
 *                  1. No reserved bits are set
 *                  2. bCBWLUN contains a valid LUN supported by device
 *                  3. bCBWCBLLength and CBWCB are in accordance with
 *                  bInterfaceSubClass
 *****/
byte IsMeaningfulCBW()
{
    /* 3msb bits of CBWCBLLength are reserved and must be 0,
     * 4msb bits of CBWLUN are reserved and must be 0
     * valid CBWCBLLength is between 1 and 16B
     * In bCBWFlags only msb indicates data direction rest must be 0
     */
    if((gblCBW.bCBWLUN<=0x0f)&&(gblCBW.bCBWCBLLength<=0x10)&&(gblCBW.bCBWCBLLength>=0x01)
    &&(gblCBW.bCBWFlags==0x00|gblCBW.bCBWFlags==0x80))
        return TRUE;
    else return FALSE;
}

/*****
 * Function:      void PrepareCSWData()
 * Overview:      This prepares the Status data of CSW by copying the
 *                  dCSWTag from CBWTag and sets the signature
 *                  of valid CSW=53425355h
 *****/
void PrepareCSWData()
{
    /* Residue and Status fields are set after decoding and executing the command */
    msd_csw.dCSWTag=gblCBW.dCBWTag;
    msd_csw.dCSWSignature=0x53425355;
}

/*****
 * Function:      void MSDInquiryHandler(void)
 * Overview:      This function prepares the response of the Inquiry command
 *                  A fixed Inquiry response is copied from ROM to the
 *                  msd_buffer and CSWStatus, CSWDataResidue values are set
 *****/
void MSDInquiryHandler(void)
{
    byte i;
    byte *buffer;
    memcpypgm2ram((byte *)&msd_buffer[0],(byte *)&inq_resp,sizeof(InquiryResponse));
    msd_csw.dCSWDataResidue=sizeof(InquiryResponse);
    msd_csw.bCSWStatus=0x00;                // success
}
```

## Wireless USB Project Design Report

```
        return;
    }

/*****
 * Function:      void ResetSenseData(void)
 * Overview:      This routine resets the Sense Data, initializing the
 *                structure RequestSenseResponse gblSenseData.
 *****/
void ResetSenseData(void)
{
    gblSenseData.ResponseCode=S_CURRENT;
    gblSenseData.VALID=0;                // no data in the information field
    gblSenseData.Obsolete=0x0;
    gblSenseData.SenseKey=S_NO_SENSE;
    gblSenseData.Resv;
    gblSenseData.ILI=0;
    gblSenseData.EOM=0;
    gblSenseData.FILEMARK=0;
    gblSenseData.Information._dword=0x00;
    gblSenseData.AddSenseLen=0x0a;      // n-7 (n=17 (0..17))
    gblSenseData.CmdSpecificInfo._dword=0x0;
    gblSenseData.ASC=0x0;
    gblSenseData.ASCQ=0x0;
    gblSenseData.FRUC=0x0;
    gblSenseData.SenseKeySpecific[0]=0x0;
    gblSenseData.SenseKeySpecific[1]=0x0;
    gblSenseData.SenseKeySpecific[2]=0x0;
}

/*****
 * Function:      void MSDReadCapacityHandler()
 * Overview:      This function processes the data from CSD register
 *                (read during initialization of sdcard) to find the number
 *                of blocks (gblNumBLKS) and block length (gblBLKLen)
 *                This data is then copied to msd_buffer and a response
 *                for Read Capacity Command is prepared
 *****/
void MSDReadCapacityHandler()
{
    dword one=0x1, C_size, C_mult, Mult, C_size_U, C_size_H, C_size_L, C_mult_H,
    C_mult_L;
    dword C_Read_Bl_Len;

    // Get the block length
    C_Read_Bl_Len=gblCSDReg._byte[5]&0x0f;
    gblBLKLen._dword=one<<C_Read_Bl_Len;

    // Get the number of blocks using C_size and C_mult
    C_size_U=gblCSDReg._byte[6]&0x03;        // 2 LSB bits
    C_size_H=gblCSDReg._byte[7];
    C_size_L=(gblCSDReg._byte[8]&0xC0)>>6;    // 2 MSB, right shift by 6places
                                           // to get in LSB
    C_size=(C_size_U<<10)|(C_size_H<<2)|(C_size_L);
    C_mult_H=gblCSDReg._byte[9]&0x03;
    C_mult_L=(gblCSDReg._byte[10]&0x80)>>7;
    C_mult=(C_mult_H<<1)|C_mult_L;
    Mult = one<<(C_mult+2);
    gblNumBLKS._dword=Mult*(C_size+1)-1; // last LBA is noLBAS-1

    // prepare the data response
    msd_buffer[0]=gblNumBLKS.v[3];
    msd_buffer[1]=gblNumBLKS.v[2];
    msd_buffer[2]=gblNumBLKS.v[1];
    msd_buffer[3]=gblNumBLKS.v[0];
    msd_buffer[4]=gblBLKLen.v[3];
    msd_buffer[5]=gblBLKLen.v[2];
    msd_buffer[6]=gblBLKLen.v[1];
    msd_buffer[7]=gblBLKLen.v[0];

    msd_csw.dCSWDataResidue=0x08;        // size of response
    msd_csw.bCSWStatus=0x00;            // success
}
```

## Wireless USB Project Design Report

```
}

/*****
* Function:      void MSDReadHandler(void)
* Overview:      Decodes the CBWCB of READ(10) command to calculate
*                the starting LBA and the Transfer length
*                (number of blocks to be read). Reads a block of 512B data
*                from SD Card in msd_buffer (by calling SectorRead).
*                If successfully read (sdValid), the data is sent to the
*                host in 64B chunks (MSD_IN_EP_SIZE) (see MSDDataIn()).
*                This is repeated for TransferLength number of blocks.
*                In case of error bCSWStatus is set to 0x01 and sense data
*                with sense key NOT READY and appropriate ASC,
*                ASCQ codes is prepared.
*****/
void MSDReadHandler()
{
    word i;
    SDC_Error status;                //SDC
    WORD TransferLength;
    DWORD LBA;
    byte Flags;
    dword sectorNumber;

    LBA.v[3]=gblCBW.CBWCB[2];
    LBA.v[2]=gblCBW.CBWCB[3];
    LBA.v[1]=gblCBW.CBWCB[4];
    LBA.v[0]=gblCBW.CBWCB[5];

    TransferLength.v[1]=gblCBW.CBWCB[7];
    TransferLength.v[0]=gblCBW.CBWCB[8];

    Flags=gblCBW.CBWCB[1];

    msd_csw.bCSWStatus=0x0;
    msd_csw.dCSWDataResidue=0x0;

    if (LBA._dword + TransferLength._word > gblNumBLKS._dword) {
        msd_csw.bCSWStatus=0x01;
        // prepare sense data See page 51 SBC-2
        gblSenseData.SenseKey=S_ILLEGAL_REQUEST;
        gblSenseData.ASC=ASC_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE;
        gblSenseData.ASCQ=ASCQ_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE;
    } else {
        while (TransferLength._word > 0) {

            TransferLength._word--;                // we
            have read 1 LBA

            //AK status = SectorRead(LBA._dword, (byte*)&msd_buffer[0]);
            LBA._dword++;                // read
            the next LBA

            if (status==sdValid) {
                msd_csw.bCSWStatus=0x00;                // success
                msd_csw.dCSWDataResidue=BLOCKLEN_512; //in order to send the
                //512 bytes of data read
                ptrNextData=(byte *)&msd_buffer[0];
                while (msd_csw.dCSWDataResidue>0)
                    MSDDataIn();                // send
                the data

                msd_csw.dCSWDataResidue=0x0;                // for next time
            } else {
                msd_csw.bCSWStatus=0x01;                // Error 0x01
                Refer page#18

                // of BOT specifications
                //Don't read any more data
                msd_csw.dCSWDataResidue=0x0;

                break;                // break the loop
            }
        }
    }
}
```

## Wireless USB Project Design Report

```
    }
}

/*****
 * Function:      void MSDDataOut(void)
 *
 * Side Effects:   MSD_BD_OUT.ADR is incremented by MSD_OUT_EP_SIZE
 *                 (to read next 64B into msd_buffer)
 *
 * Overview:       This function reads 64B (MSD_OUT_EP_SIZE)
 *                 from EPl OUT MSD_BD_OUT
 *****/
void MSDDataOut(void)
{
    mUSBBufferReady(MSD_BD_OUT);
    USBDriverService();
    while(mMSDRxIsBusy());

    gblCBW.dCBWDataTransferLength-=MSD_BD_OUT.Cnt;           // 64B read
    msd_csw.dCSWDataResidue-=MSD_BD_OUT.Cnt;
    MSD_BD_OUT.Cnt=MSD_OUT_EP_SIZE;
    MSD_BD_OUT.ADR+=MSD_OUT_EP_SIZE;
}

/*****
 * Function:      void MSDWriteHandler()
 * Overview:       Decodes the CBWCB of WRITE(10) command to calculate
 *                 the starting LBA and theTransfer length (number of
 *                 blocks to be written). Reads TransferLength blocks
 *                 of data, 1 block=512B at a time in msd_buffer.
 *                 The data from the host, 64B in MSD_BD_OUT, is received
 *                 in the msd_buffer (see MSDDataOut()).
 *                 The MSD_BD_OUT.ADR pointer is manipulated to fill the 512B
 *                 msd_buffer and when full the data is written to the SD Card
 *                 by calling the function SectorWrite(...) (see sdc card.c)
 *                 In case of error bCSWStatus is set to 0x01 and sense
 *                 data with sense key NOT READY and appropriate ASC,
 *                 ASCQ codes is prepared.
 *****/
void MSDWriteHandler()
{
    word i;
    byte* adr;
    SDC_Error status=sdcValid;           //SDC
    WORD TransferLength;
    DWORD LBA;
    byte Flags;
    dword sectorNumber;

    /* Read the LBA, TransferLength fields from Command Block
    NOTE: CB is Big-Endian */

    LBA.v[3]=gblCBW.CBWCB[2];
    LBA.v[2]=gblCBW.CBWCB[3];
    LBA.v[1]=gblCBW.CBWCB[4];
    LBA.v[0]=gblCBW.CBWCB[5];
    TransferLength.v[1]=gblCBW.CBWCB[7];
    TransferLength.v[0]=gblCBW.CBWCB[8];

    msd_csw.bCSWStatus=0x0;
    while (TransferLength._word > 0) {
        msd_csw.dCSWDataResidue=BLOCKLEN_512;
        //Read 512B into msd_buffer
        while (msd_csw.dCSWDataResidue>0)
            MSDDataOut();
/*AK
        if(IsWriteProtected()) {
            gblSenseData.SenseKey=S_NOT_READY;
            gblSenseData.ASC=ASC_WRITE_PROTECTED;
            gblSenseData.ASCQ=ASCQ_WRITE_PROTECTED;
            msd_csw.bCSWStatus=0x01;
        } else {
```

## Wireless USB Project Design Report

```
        //AK status = SectorWrite((LBA._dword), (byte*)&msd_buffer[0]);
    }
*/
    if (status) {
        msd_csw.bCSWStatus=0x01;
        //add some sense keys here?
    }
    LBA._dword++; // One LBA is written. Write the next

    TransferLength._word--;

    //Point MSD_BD_OUT to msd_cbw after done reading the WRITE data from host

    if (TransferLength._word>0){
        MSD_BD_OUT.Cnt=MSD_OUT_EP_SIZE;
        MSD_BD_OUT.ADR=(byte*)&msd_buffer[0];
    } else {
        MSD_BD_OUT.Cnt=sizeof(msd_cbw);
        MSD_BD_OUT.ADR=(byte*)&msd_cbw;
    }
    // end of while
    return;
}

/*****
 * Function:      void MSDRequestSenseHandler(void)
 * Overview:      This function prepares the Sense Data in response
 *                to the Request Sense Command The contents of structure
 *                RequestSenseResponse are copied to msd_buffer and a
 *                success bCSWStatus=0x00 is set.
 *****/
void MSDRequestSenseHandler(void)
{
    byte i;
    for(i=0;i<sizeof(RequestSenseResponse);i++)
        msd_buffer[i]=gblSenseData._byte[i];

    msd_csw.dCSWDataResidue=sizeof(RequestSenseResponse);
    msd_csw.bCSWStatus=0x0; // success
    return;
}

/*****
 * Function:      void MSDModeSenseHandler()
 * Overview:      This function prepares response to the Mode Sense command
 *                a basic response is implemented in this version of the code
 *                00h implies no other mode pages and 0x03 is the size of the
 *                data (in bytes) that follows.
 *****/
void MSDModeSenseHandler()
{
    msd_buffer[0]=0x03;
    msd_buffer[1]=0x00;
    msd_buffer[2]=0x00;
    msd_buffer[3]=0x00;

    msd_csw.bCSWStatus=0x0;
    msd_csw.dCSWDataResidue=0x04;
    return;
}

/*****
 * Function:      void MSDMediumRemovalHandler()
 * Overview:      This function prepares response to Prevent Allow
 *                Medium Removal Command No data response is expect only
 *                a CSW with command execution status is expected
 *                Since we cannot control the removal of media,
 *                we respond by a Success CSW
 *****/
void MSDMediumRemovalHandler()
```

## Wireless USB Project Design Report

```
{
/*AK    if(DetectSDCard()) {
        msd_csw.bCSWStatus=0x00;
        msd_csw.dCSWDataResidue=0x00;
    } else {
        gblSenseData.SenseKey=S_NOT_READY;
        gblSenseData.ASC=ASC_MEDIUM_NOT_PRESENT;
        gblSenseData.ASCQ=ASCQ_MEDIUM_NOT_PRESENT;
        msd_csw.bCSWStatus=0x01;
    }
*/
    return;
}

/*****
 * Function:        void MSDTestUnitReadyHandler()
 * Overview:        This function prepares response to Test Unit Ready Command
 *                  No data response is expected, only a CSW is to be sent
 *                  Based on the current state of the SDCard an error or
 *                  success status value is set
 *****/
void MSDTestUnitReadyHandler()
{
    msd_csw.bCSWStatus=0x0;
    ResetSenseData();
/*AK
    if(!gblFlag.isSDMMC) {
        gblSenseData.SenseKey=S_NOT_READY;
        gblSenseData.ASC=ASC_LOGICAL_UNIT_NOT_SUPPORTED;
        gblSenseData.ASCQ=ASCQ_LOGICAL_UNIT_NOT_SUPPORTED;
        msd_csw.bCSWStatus=0x01;
        mLED_2_On();
    }

    if(!DetectSDCard()) {
        gblSenseData.SenseKey=S_UNIT_ATTENTION;
        gblSenseData.ASC=ASC_MEDIUM_NOT_PRESENT;
        gblSenseData.ASCQ=ASCQ_MEDIUM_NOT_PRESENT;
        msd_csw.bCSWStatus=0x01;
        gblFlag.isSDMMC=0;
    } else {
        //AKToggleRUNLED();
    }
    //}
*/
    msd_csw.dCSWDataResidue=0x00;
    SendCSW();
    return;
}

/*****
 * Function:        void MSDVerifyHandler()
 * Overview:        This function prepares response to Verify Command
 *                  No data response is expected, we reply by a success CSW
 *                  The command is not being processed in this version of code
 *****/
void MSDVerifyHandler()
{
    msd_csw.bCSWStatus=0x0;
    msd_csw.dCSWDataResidue=0x00;
    return;
}

/*****
 * Function:        void MSDStopStartHandler()
 * Overview:        This function prepares response to Start Stop Unit Command
 *                  No data response is expected, we reply by a success CSW
 *                  The command is not being processed in this version of code
 *****/
void MSDStopStartHandler()
```



## Wireless USB Project Design Report

```
{
    msd_csw.bCSWStatus=0x0;
    msd_csw.dCSWDataResidue=0x00;
    return;
}
#endif //def USB_USE_MSD
/** EOF msd.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usb9.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#define USB9_H
#define USB9_H

/* I N C L U D E S *****/
#include "system\typedefs.h"

/* D E F I N I T I O N S *****/
/*****
 * Standard Request Codes
 * USB 2.0 Spec Ref Table 9-4
 *****/
#define GET_STATUS 0
#define CLR_FEATURE 1
#define SET_FEATURE 3
#define SET_ADR 5
#define GET_DSC 6
#define SET_DSC 7
#define GET_CFG 8
#define SET_CFG 9
#define GET_INTF 10
#define SET_INTF 11
#define SYNCH_FRAME 12

/* Standard Feature Selectors */
#define DEVICE_REMOTE_WAKEUP 0x01
#define ENDPOINT_HALT 0x00

/*****
 * Macro:                void mUSBCheckAdrPendingState(void)
 * Overview:             Specialized checking routine, it checks if the device
 *                       is in the ADDRESS PENDING STATE and services it if it is.
 *****/
#define mUSBCheckAdrPendingState() if(usb_device_state==ADR_PENDING_STATE) \
{ \
    UADDR = SetupPkt.bDevADR._byte; \
    if(UADDR > 0) \
        usb_device_state=ADDRESS_STATE; \
    else \
        usb_device_state=DEFAULT_STATE; \
} //end if

/* E X T E R N S *****/
// None?

/* P U B L I C P R O T O T Y P E S *****/
void USBCheckStdRequest(void);

#endif //USB9_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usb9.c
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

/** I N C L U D E S *****/
#include <p18cxxx.h>
#include <p18f4550.h>
#include "system\typedefs.h"
#include "system\usb\usb.h"
#include "io_cfg.h"                // Required for self_power status

/** V A R I A B L E S *****/
#pragma udata

/** P R I V A T E   P R O T O T Y P E S *****/
void USBStdGetDscHandler(void);
void USBStdSetCfgHandler(void);
void USBStdGetStatusHandler(void);
void USBStdFeatureReqHandler(void);

/** D E C L A R A T I O N S *****/
#pragma code
/*****
* Function:        void USBCheckStdRequest(void)
* Overview:        This routine checks the setup data packet to see if it
*                  knows how to handle it
*****/
void USBCheckStdRequest(void)
{
    if(SetupPkt.RequestType != STANDARD) return;

    switch(SetupPkt.bRequest)
    {
        case SET_ADR:
            ctrl_trf_session_owner = MUID_USB9;
            usb_device_state = ADR_PENDING_STATE;        // Update state only
            /* See USBCtrlTrfInHandler() in usbctrltrf.c for the next step */
            break;
        case GET_DSC:
            USBStdGetDscHandler();
            break;
        case SET_CFG:
            USBStdSetCfgHandler();
            break;
        case GET_CFG:
            ctrl_trf_session_owner = MUID_USB9;
            pSrc.bRam = (byte*)&usb_active_cfg;        // Set Source
            usb_stat.ctrl_trf_mem = _RAM;                // Set memory type
            LSB(wCount) = 1;                             // Set data count
            break;
        case GET_STATUS:
            USBStdGetStatusHandler();
            break;
        case CLR_FEATURE:
        case SET_FEATURE:
            USBStdFeatureReqHandler();
            break;
        case GET_INTF:
            ctrl_trf_session_owner = MUID_USB9;
            pSrc.bRam = (byte*)&usb_alt_intf+SetupPkt.bIntfID; // Set source
            usb_stat.ctrl_trf_mem = _RAM;                // Set memory type
    }
}
```

## Wireless USB Project Design Report

```
        LSB(wCount) = 1;                                // Set data count
        break;
    case SET_INTF:
        ctrl_trf_session_owner = MUID_USB9;
        usb_alt_intf[SetupPkt.bIntfID] = SetupPkt.bAltID;
        break;
    case SET_DSC:
    case SYNCH_FRAME:
    default:
        break;
    } //end switch
} //end USBCheckStdRequest

/*****
 * Function:      void USBStdGetDscHandler(void)
 * Overview:      This routine handles the standard GET_DESCRIPTOR request.
 *                It utilizes tables to dynamically look up descriptor size.
 *                This routine should never have to be modified if the tables
 *                in usbdsc.c are declared correctly.
 *****/
void USBStdGetDscHandler(void)
{
    if(SetupPkt.bmRequestType == 0x80)
    {
        switch(SetupPkt.bDscType)
        {
            case DSC_DEV:
                ctrl_trf_session_owner = MUID_USB9;
                pSrc.bRom = (rom_byte*)&device_dsc;
                wCount._word = sizeof(device_dsc);          // Set data count
                break;
            case DSC_CFG:
                ctrl_trf_session_owner = MUID_USB9;
                pSrc.bRom = *(USB_CD_Ptr+SetupPkt.bDscIndex);
                wCount._word = *(pSrc.wRom+1);              // Set data count
                break;
            case DSC_STR:
                ctrl_trf_session_owner = MUID_USB9;
                pSrc.bRom = *(USB_SD_Ptr+SetupPkt.bDscIndex);
                wCount._word = *pSrc.bRom;                  // Set data count
                break;
        } //end switch

        usb_stat.ctrl_trf_mem = _ROM;                      // Set memory type
    } //end if
} //end USBStdGetDscHandler

/*****
 * Function:      void USBStdSetCfgHandler(void)
 * Overview:      This routine first disables all endpoints by clearing
 *                UEP registers. It then configures (initializes) endpoints
 *                specified in the modifiable section.
 *****/
void USBStdSetCfgHandler(void)
{
    ctrl_trf_session_owner = MUID_USB9;
    mDisableEP1to15();                                     // See usbdrv.h
    ClearArray((byte*)&usb_alt_intf, MAX_NUM_INT);
    usb_active_cfg = SetupPkt.bCfgValue;
    if(SetupPkt.bCfgValue == 0)
        usb_device_state = ADDRESS_STATE;
    else
    {
        usb_device_state = CONFIGURED_STATE;

        /* Modifiable Section */
        #if defined(USB_USE_HID)                          // See autofiles\usbcfg.h
            HIDInitEP();
        #endif
        /* End modifiable section */
    }
}
```

## Wireless USB Project Design Report

```
    } //end if(SetupPkt.bcfgValue == 0)
} //end USBStdSetCfgHandler

/*****
 * Function:      void USBStdGetStatusHandler(void)
 * Overview:      This routine handles the standard GET_STATUS request
 *****/
void USBStdGetStatusHandler(void)
{
    CtrlTrfData._byte0 = 0;                // Initialize content
    CtrlTrfData._byte1 = 0;

    switch(SetupPkt.Recipient)
    {
        case RCPT_DEV:
            ctrl_trf_session_owner = MUID_USB9;
            /*
             * _byte0: bit0: Self-Powered Status [0] Bus-Powered [1] Self-Powered
             *          bit1: RemoteWakeup       [0] Disabled   [1] Enabled
             */
            if(self_power == 1)             // self_power defined in io_cfg.h
                CtrlTrfData._byte0 |= 0b000000001; // Set bit0

            if(usb_stat.RemoteWakeup == 1)  // usb_stat defined in usbmmap.c
                CtrlTrfData._byte0 |= 0b00000010; // Set bit1
            break;
        case RCPT_INTF:
            ctrl_trf_session_owner = MUID_USB9; // No data to update
            break;
        case RCPT_EP:
            ctrl_trf_session_owner = MUID_USB9;
            /*
             * _byte0: bit0: Halt Status [0] Not Halted [1] Halted
             */
            pDst.bRam = (byte*)&ep0Bo+(SetupPkt.EPNum*8)+(SetupPkt.EPDir*4);
            if(*pDst.bRam & _BSTALL) // Use _BSTALL as a bit mask
                CtrlTrfData._byte0 = 0x01; // Set bit0
            break;
    } //end switch

    if(ctrl_trf_session_owner == MUID_USB9)
    {
        pSrc.bRam = (byte*)&CtrlTrfData; // Set Source
        usb_stat.ctrl_trf_mem = _RAM;     // Set memory type
        LSB(wCount) = 2;                 // Set data count
    } //end if(ctrl_trf_session_owner == MUID_USB9)
} //end USBStdGetStatusHandler

/*****
 * Function:      void USBStdFeatureReqHandler(void)
 * Overview:      This routine handles the standard SET & CLEAR FEATURES
 *                requests
 *****/
void USBStdFeatureReqHandler(void)
{
    if((SetupPkt.bFeature == DEVICE_REMOTE_WAKEUP)&&
        (SetupPkt.Recipient == RCPT_DEV))
    {
        ctrl_trf_session_owner = MUID_USB9;
        if(SetupPkt.bRequest == SET_FEATURE)
            usb_stat.RemoteWakeup = 1;
        else
            usb_stat.RemoteWakeup = 0;
    } //end if

    if((SetupPkt.bFeature == ENDPOINT_HALT)&&
        (SetupPkt.Recipient == RCPT_EP)&&
        (SetupPkt.EPNum != 0))
    {
        ctrl_trf_session_owner = MUID_USB9;
    }
}
```

## Wireless USB Project Design Report

```
/* Must do address calculation here */
pDst.bRam = (byte*)&ep0Bo+(SetupPkt.EPNum*8)+(SetupPkt.EPDir*4);

if(SetupPkt.bRequest == SET_FEATURE)
    *pDst.bRam = _USIE|_BSTALL;
else
{
    if(SetupPkt.EPDir == 1) // IN
        *pDst.bRam = _UCPU;
    else
        *pDst.bRam = _USIE|_DAT0|_DTSEN;
} //end outer if else
} //end if
} //end USBStdFeatureReqHandler

/** EOF usb9.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbctrltrf.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:          4.5.2006
/      Processor:        PIC18F4550
/      Compiler:         Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****/

#ifndef USBCTRLTRF_H
#define USBCTRLTRF_H

/** I N C L U D E S *****/
#include "system\typedefs.h"

/** D E F I N I T I O N S *****/

/* Control Transfer States */
#define WAIT_SETUP      0
#define CTRL_TRF_TX     1
#define CTRL_TRF_RX     2

/* USB PID: Token Types - See chapter 8 in the USB specification */
#define SETUP_TOKEN     0b00001101
#define OUT_TOKEN       0b00000001
#define IN_TOKEN        0b00001001

/* bmRequestType Definitions */
#define HOST_TO_DEV     0
#define DEV_TO_HOST     1

#define STANDARD        0x00
#define CLASS           0x01
#define VENDOR          0x02

#define RCPT_DEV        0
#define RCPT_INTF       1
#define RCPT_EP         2
#define RCPT_OTH        3

/** E X T E R N S *****/
extern byte ctrl_trf_session_owner;

extern POINTER pSrc;
extern POINTER pDst;
extern WORD wCount;

/** P U B L I C   P R O T O T Y P E S *****/
void USBCtrlEPService(void);
void USBCtrlTrfTxService(void);
void USBCtrlTrfRxService(void);
void USBCtrlEPServiceComplete(void);
void USBPrepareForNextSetupTrf(void);

#endif //USBCTRLTRF_H

```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usbctrltrf.c
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

/** I N C L U D E S *****/
#include <p18cxxx.h>
#include <p18f4550.h>
#include "system\typedefs.h"
#include "system\usb\usb.h"

/** V A R I A B L E S *****/
#pragma udata
byte ctrl_trf_state;           // Control Transfer State
byte ctrl_trf_session_owner;  // Current transfer session owner
POINTER pSrc;                 // Data source pointer
POINTER pDst;                 // Data destination pointer
WORD wCount;                  // Data counter

/** P R I V A T E   P R O T O T Y P E S *****/
void USBCtrlTrfSetupHandler(void);
void USBCtrlTrfOutHandler(void);
void USBCtrlTrfInHandler(void);

/** D E C L A R A T I O N S *****/
#pragma code
/*****
* Function:        void USBCtrlEPService(void)
* PreCondition:    USTAT is loaded with a valid endpoint address.
* Overview:        USBCtrlEPService checks for three transaction types that
*                  it knows how to service and services them:
*                  1. EP0 SETUP
*                  2. EP0 OUT
*                  3. EP0 IN
*                  It ignores all other types (i.e. EP1, EP2, etc.)
*****/
void USBCtrlEPService(void)
{
    if(USTAT == EP00_OUT)
    {
        if(ep0Bo.Stat.PID == SETUP_TOKEN)           // EP0 SETUP
            USBCtrlTrfSetupHandler();
        else                                         // EP0 OUT
            USBCtrlTrfOutHandler();
    }
    else if(USTAT == EP00_IN)                       // EP0 IN
        USBCtrlTrfInHandler();
}

//end USBCtrlEPService

/*****
* Function:        void USBCtrlTrfSetupHandler(void)
* PreCondition:    SetupPkt buffer is loaded with valid USB Setup Data
* Overview:        This routine is a task dispatcher and has 3 stages.
*                  1. It initializes the control transfer state machine.
*                  2. It calls on each of the module that may know how to
*                     service the Setup Request from the host.
*                     Module Example: USB9, HID, CDC, MSD, ...
*                     As new classes are added, ClassReqHandler table in
*                     usbdsc.c should be updated to call all available
*                     class handlers.
*                  3. Once each of the modules has had a chance to check if

```



## Wireless USB Project Design Report

```
*           it is responsible for servicing the request, stage 3
*           then checks direction of the transfer to determine how
*           to prepare EP0 for the control transfer.
*           Refer to USBCtrlEPServiceComplete() for more details.
*
* Note:      Microchip USB Firmware has three different states for
*           the control transfer state machine:
*           1. WAIT_SETUP
*           2. CTRL_TRF_TX
*           3. CTRL_TRF_RX
*           Refer to firmware manual to find out how one state
*           is transitioned to another.
*
*           A Control Transfer is composed of many USB transactions.
*           When transferring data over multiple transactions,
*           it is important to keep track of data source, data
*           destination, and data count. These three parameters are
*           stored in pSrc, pDst, and wCount. A flag is used to
*           note if the data source is from ROM or RAM.
*
*****/
void USBCtrlTrfSetupHandler(void)
{
    byte i;

    /* Stage 1 */
    ctrl_trf_state = WAIT_SETUP;
    ctrl_trf_session_owner = MUID_NULL;    // Set owner to NULL
    wCount._word = 0;

    /* Stage 2 */
    USBCheckStdRequest();                  // See system\usb9\usb9.c

    /* Modifiable Section */
    for(i=0; i < (sizeof(ClassReqHandler)/sizeof(pFunc)); i++)
    {
        if(ctrl_trf_session_owner != MUID_NULL) break;
        ClassReqHandler[i]();              // See autofiles\usbdsc.c
    } //end while
    /* End Modifiable Section */

    /* Stage 3 */
    USBCtrlEPServiceComplete();
} //end USBCtrlTrfSetupHandler

*****/
* Function:      void USBCtrlTrfOutHandler(void)
* Overview:      This routine handles an OUT transaction according to
*                which control transfer state is currently active.
*
* Note:          Note that if the the control transfer was from
*                host to device, the session owner should be notified
*                at the end of each OUT transaction to service the
*                received data.
*
*****/
void USBCtrlTrfOutHandler(void)
{
    if(ctrl_trf_state == CTRL_TRF_RX)
    {
        USBCtrlTrfRxService();

        /*
         * Don't have to worry about overwriting _KEEP bit
         * because if _KEEP was set, TRNIF would not have been
         * generated in the first place.
         */
        if(ep0Bo.Stat.DTS == 0)
            ep0Bo.Stat._byte = _USIE|_DAT1|_DTSEN;
        else
    }
```

## Wireless USB Project Design Report

```
        ep0Bo.Stat._byte = _USIE|_DAT0|_DTSEN;
    }
    else // CTRL_TRF_TX
        USBPrepareForNextSetupTrf();
} //end USBCtrlTrfOutHandler

/*****
 * Function:      void USBCtrlTrfInHandler(void)
 * Overview:      This routine handles an IN transaction according to
 *                which control transfer state is currently active.
 *
 * Note:          A Set Address Request must not change the actual address
 *                of the device until the completion of the control
 *                transfer. The end of the control transfer for Set Address
 *                Request is an IN transaction. Therefore it is necessary
 *                to service this unique situation when the condition is
 *                right. Macro mUSBCheckAdrPendingState is defined in
 *                usb9.h and its function is to specifically service this
 *                event.
 *****/
void USBCtrlTrfInHandler(void)
{
    mUSBCheckAdrPendingState(); // Must check if in ADR_PENDING_STATE

    if(ctrl_trf_state == CTRL_TRF_TX)
    {
        USBCtrlTrfTxService();

        if(ep0Bi.Stat.DTS == 0)
            ep0Bi.Stat._byte = _USIE|_DAT1|_DTSEN;
        else
            ep0Bi.Stat._byte = _USIE|_DAT0|_DTSEN;
    }
    else // CTRL_TRF_RX
        USBPrepareForNextSetupTrf();
} //end USBCtrlTrfInHandler

/*****
 * Function:      void USBCtrlTrfTxService(void)
 * PreCondition:  pSrc, wCount, and usb_stat.ctrl_trf_mem are setup properly.
 * Overview:      This routine should be called from only two places.
 *                One from USBCtrlEPServiceComplete() and one from
 *                USBCtrlTrfInHandler(). It takes care of managing a
 *                transfer over multiple USB transactions.
 *
 * Note:          This routine works with isochronous endpoint larger than
 *                256 bytes and is shown here as an example of how to deal
 *                with BC9 and BC8. In reality, a control endpoint can never
 *                be larger than 64 bytes.
 *****/
void USBCtrlTrfTxService(void)
{
    WORD byte_to_send;

    /*
     * First, have to figure out how many byte of data to send.
     */
    if(wCount._word < EP0_BUFF_SIZE)
        byte_to_send._word = wCount._word;
    else
        byte_to_send._word = EP0_BUFF_SIZE;

    /*
     * Next, load the number of bytes to send to BC9..0 in buffer descriptor
     */
    ep0Bi.Stat.BC9 = 0;
    ep0Bi.Stat.BC8 = 0;
    ep0Bi.Stat._byte |= MSB(byte_to_send);
    ep0Bi.Cnt = LSB(byte_to_send);
}
```

## Wireless USB Project Design Report

```
/*
 * Subtract the number of bytes just about to be sent from the total.
 */
wCount._word = wCount._word - byte_to_send._word;

pDst.bRam = (byte*)&CtrlTrfData;          // Set destination pointer

if(usb_stat.ctrl_trf_mem == _ROM)          // Determine type of memory source
{
    while(byte_to_send._word)
    {
        *pDst.bRam = *pSrc.bRom;
        pDst.bRam++;
        pSrc.bRom++;
        byte_to_send._word--;
    } //end while(byte_to_send._word)
}
else // RAM
{
    while(byte_to_send._word)
    {
        *pDst.bRam = *pSrc.bRam;
        pDst.bRam++;
        pSrc.bRam++;
        byte_to_send._word--;
    } //end while(byte_to_send._word)
} //end if(usb_stat.ctrl_trf_mem == _ROM)
} //end USBCtrlTrfTxService

/*****
 * Function:      void USBCtrlTrfRxService(void)
 *
 * PreCondition:  pDst and wCount are setup properly.
 *                pSrc is always &CtrlTrfData
 *                usb_stat.ctrl_trf_mem is always _RAM.
 *                wCount should be set to 0 at the start of each control
 *                transfer.
 * Overview:      *** This routine is only partially complete. Check for
 *                new version of the firmware.
 *****/
void USBCtrlTrfRxService(void)
{
    WORD byte_to_read;

    MSB(byte_to_read) = 0x03 & ep0Bo.Stat._byte;    // Filter out last 2 bits
    LSB(byte_to_read) = ep0Bo.Cnt;

    /*
     * Accumulate total number of bytes read
     */
    wCount._word = wCount._word + byte_to_read._word;

    pSrc.bRam = (byte*)&CtrlTrfData;

    while(byte_to_read._word)
    {
        *pDst.bRam = *pSrc.bRam;
        pDst.bRam++;
        pSrc.bRam++;
        byte_to_read._word--;
    } //end while(byte_to_read._word)
} //end USBCtrlTrfRxService

/*****
 * Function:      void USBCtrlEPServiceComplete(void)
 * Overview:      This routine wrap up the remaining tasks in servicing
 *                a Setup Request. Its main task is to set the endpoint
 *                controls appropriately for a given situation. See code
 *****/
```

## Wireless USB Project Design Report

```
*
*          below.
*          There are three main scenarios:
*          a) There was no handler for the Request, in this case
*             a STALL should be sent out.
*          b) The host has requested a read control transfer,
*             endpoints are required to be setup in a specific way.
*          c) The host has requested a write control transfer, or
*             a control data stage is not required, endpoints are
*             required to be setup in a specific way.
*
*          Packet processing is resumed by clearing PKTDIS bit.
*****/
void USBCtrlEPServiceComplete(void)
{
    /*
    * PKTDIS bit is set when a Setup Transaction is received.
    * Clear to resume packet processing.
    */
    UCONbits.PKTDIS = 0;
    if(ctrl_trf_session_owner == MUID_NULL)
    {
        /*
        * If no one knows how to service this request then stall.
        * Must also prepare EP0 to receive the next SETUP transaction.
        */
        ep0Bo.Cnt = EP0_BUFF_SIZE;
        ep0Bo.ADR = (byte*)&SetupPkt;

        ep0Bo.Stat._byte = _USIE|_BSTALL;
        ep0Bi.Stat._byte = _USIE|_BSTALL;
    }
    else // A module has claimed ownership of the control transfer session.
    {
        if(SetupPkt.DataDir == DEV_TO_HOST)
        {
            if(SetupPkt.wLength < wCount._word)
                wCount._word = SetupPkt.wLength;
            USBCtrlTrfTxService();
            ctrl_trf_state = CTRL_TRF_TX;
            /*
            * Control Read:
            * <SETUP[0]><IN[1]><IN[0]>...<OUT[1]> | <SETUP[0]>
            * 1. Prepare OUT EP to respond to early termination
            *
            * NOTE:
            * If something went wrong during the control transfer,
            * the last status stage may not be sent by the host.
            * When this happens, two different things could happen
            * depending on the host.
            * a) The host could send out a RESET.
            * b) The host could send out a new SETUP transaction
            *    without sending a RESET first.
            * To properly handle case (b), the OUT EP must be setup
            * to receive either a zero length OUT transaction, or a
            * new SETUP transaction.
            *
            * Since the SETUP transaction requires the DTS bit to be
            * DAT0 while the zero length OUT status requires the DTS
            * bit to be DAT1, the DTS bit check by the hardware should
            * be disabled. This way the SIE could accept either of
            * the two transactions.
            *
            * Furthermore, the Cnt byte should be set to prepare for
            * the SETUP data (8-byte or more), and the buffer address
            * should be pointed to SetupPkt.
            */
            ep0Bo.Cnt = EP0_BUFF_SIZE;
            ep0Bo.ADR = (byte*)&SetupPkt;
            ep0Bo.Stat._byte = _USIE; // Note: DTSEN is 0!
        }
        /*

```

## Wireless USB Project Design Report

```
        * 2. Prepare IN EP to transfer data, Cnt should have
        *   been initialized by responsible request owner.
        */
        ep0Bi.ADR = (byte*)&CtrlTrfData;
        ep0Bi.Stat._byte = _USIE|_DAT1|_DTSEN;
    }
    else    // (SetupPkt.DataDir == HOST_TO_DEV)
    {
        ctrl_trf_state = CTRL_TRF_RX;
        /*
        * Control Write:
        * <SETUP[0]><OUT[1]><OUT[0]>...<IN[1]> | <SETUP[0]>
        *
        * 1. Prepare IN EP to respond to early termination
        *
        *   This is the same as a Zero Length Packet Response
        *   for control transfer without a data stage
        */
        ep0Bi.Cnt = 0;
        ep0Bi.Stat._byte = _USIE|_DAT1|_DTSEN;

        /*
        * 2. Prepare OUT EP to receive data.
        */
        ep0Bo.Cnt = EP0_BUFF_SIZE;
        ep0Bo.ADR = (byte*)&CtrlTrfData;
        ep0Bo.Stat._byte = _USIE|_DAT1|_DTSEN;
    } //end if(SetupPkt.DataDir == DEV_TO_HOST)
} //end if(ctrl_trf_session_owner == MUID_NULL)

} //end USBCtrlEPServiceComplete

/*****
 * Function:      void USBPrepareForNextSetupTrf(void)
 * Overview:      The routine forces EP0 OUT to be ready for a new Setup
 *                 transaction, and forces EP0 IN to be owned by CPU.
 *****/
void USBPrepareForNextSetupTrf(void)
{
    ctrl_trf_state = WAIT_SETUP;           // See usbctrltrf.h
    ep0Bo.Cnt = EP0_BUFF_SIZE;             // Defined in usbcfg.h
    ep0Bo.ADR = (byte*)&SetupPkt;
    ep0Bo.Stat._byte = _USIE|_DAT0|_DTSEN; // EP0 buff dsc init, see usbmmap.h
    ep0Bi.Stat._byte = _UCPU;              // EP0 IN buffer initialization
} //end USBPrepareForNextSetupTrf

/** EOF usbctrltrf.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usbdefs_ep0_buff.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

/*****
* USB Definitions: Endpoint 0 Buffer
*****
#ifndef USBDEFS_EP0_BUFF_H
#define USBDEFS_EP0_BUFF_H

/** I N C L U D E S *****/
#include "system\typedefs.h"
#include "autofiles\usbcfg.h"          // usbcfg.h contains required definitions

/*****
* CTRL_TRF_SETUP:
*
* Every setup packet has 8 bytes.
* However, the buffer size has to equal the EP0_BUFF_SIZE value specified
* in autofiles\usbcfg.h
* The value of EP0_BUFF_SIZE can be 8, 16, 32, or 64.
*
* First 8 bytes are defined to be directly addressable to improve speed
* and reduce code size.
* Bytes beyond the 8th byte have to be accessed using indirect addressing.
*****/
typedef union _CTRL_TRF_SETUP
{
    /** Array for indirect addressing *****/
    struct
    {
        byte _byte[EP0_BUFF_SIZE];
    };

    /** Standard Device Requests *****/
    struct
    {
        byte bmRequestType;
        byte bRequest;
        word wValue;
        word wIndex;
        word wLength;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        WORD W_Value;
        WORD W_Index;
        WORD W_Length;
    };
    struct
    {
        unsigned Recipient:5;           //Device,Interface,Endpoint,Other
        unsigned RequestType:2;         //Standard,Class,Vendor,Reserved
        unsigned DataDir:1;             //Host-to-device,Device-to-host
        unsigned :8;
        byte bFeature;                  //DEVICE_REMOTE_WAKEUP,ENDPOINT_HALT
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
};

```

## Wireless USB Project Design Report

```
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        byte bDscIndex;           //For Configuration and String DSC Only
        byte bDscType;           //Device,Configuration,String
        word wLangID;            //Language ID
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        BYTE bDevADR;            //Device Address 0-127
        byte bDevADRH;           //Must equal zero
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        byte bCfgValue;          //Configuration Value 0-255
        byte bCfgRSD;            //Must equal zero (Reserved)
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        byte bAltID;             //Alternate Setting Value 0-255
        byte bAltID_H;           //Must equal zero
        byte bIntfID;            //Interface Number Value 0-255
        byte bIntfID_H;          //Must equal zero
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
        byte bEPID;              //Endpoint ID (Number & Direction)
        byte bEPID_H;            //Must equal zero
        unsigned :8;
        unsigned :8;
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned EPNum:4;        //Endpoint Number 0-15
        unsigned :3;
        unsigned EPDir:1;        //Endpoint Direction: 0-OUT, 1-IN
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    /** End: Standard Device Requests *****/
```

## Wireless USB Project Design Report

```
} CTRL_TRF_SETUP;

/*****
 * CTRL_TRF_DATA:
 *
 * Buffer size has to equal the EP0_BUFF_SIZE value specified
 * in autotfiles\usbcfg.h
 * The value of EP0_BUFF_SIZE can be 8, 16, 32, or 64.
 *
 * First 8 bytes are defined to be directly addressable to improve speed
 * and reduce code size.
 * Bytes beyond the 8th byte have to be accessed using indirect addressing.
 *****/
typedef union _CTRL_TRF_DATA
{
    /*** Array for indirect addressing *****/
    struct
    {
        byte _byte[EP0_BUFF_SIZE];
    };

    /** First 8-byte direct addressing *****/
    struct
    {
        byte _byte0;
        byte _byte1;
        byte _byte2;
        byte _byte3;
        byte _byte4;
        byte _byte5;
        byte _byte6;
        byte _byte7;
    };
    struct
    {
        word _word0;
        word _word1;
        word _word2;
        word _word3;
    };
} CTRL_TRF_DATA;

#endif //USBDEFS_EP0_BUFF_H
```



## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbdefs_std_dsc.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/
/      Version:           4.7.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

/*****
* USB Definitions: Standard Descriptors
*****
#ifndef USBDEFS_STD_DSC_H
#define USBDEFS_STD_DSC_H

/** I N C L U D E S *****/
#include "system\typedefs.h"

/** D E F I N I T I O N S *****/

/* Descriptor Types */
#define DSC_DEV      0x01
#define DSC_CFG      0x02
#define DSC_STR      0x03
#define DSC_INTF     0x04
#define DSC_EP       0x05

/*****
* USB Endpoint Definitions
* USB Standard EP Address Format: DIR:X:X:EP3:EP2:EP1:EP0
* This is used in the descriptors. See autofiles\usbdsc.c
*
* NOTE: Do not use these values for checking against USTAT.
* To check against USTAT, use values defined in "system\usb\usbdrv\usbdrv.h"
*****/
#define _EP01_OUT    0x01
#define _EP01_IN     0x81
#define _EP02_OUT    0x02
#define _EP02_IN     0x82
#define _EP03_OUT    0x03
#define _EP03_IN     0x83
#define _EP04_OUT    0x04
#define _EP04_IN     0x84
#define _EP05_OUT    0x05
#define _EP05_IN     0x85
#define _EP06_OUT    0x06
#define _EP06_IN     0x86
#define _EP07_OUT    0x07
#define _EP07_IN     0x87
#define _EP08_OUT    0x08
#define _EP08_IN     0x88
#define _EP09_OUT    0x09
#define _EP09_IN     0x89
#define _EP10_OUT    0x0A
#define _EP10_IN     0x8A
#define _EP11_OUT    0x0B
#define _EP11_IN     0x8B
#define _EP12_OUT    0x0C
#define _EP12_IN     0x8C
#define _EP13_OUT    0x0D
#define _EP13_IN     0x8D
#define _EP14_OUT    0x0E
#define _EP14_IN     0x8E
#define _EP15_OUT    0x0F
#define _EP15_IN     0x8F

```

## Wireless USB Project Design Report

```
/* Configuration Attributes */
#define _DEFAULT      0x01<<7      //Default Value (Bit 7 is set)
#define _SELF         0x01<<6      //Self-powered (Supports if set)
#define _RWU          0x01<<5      //Remote Wakeup (Supports if set)

/* Endpoint Transfer Type */
#define _CTRL         0x00          //Control Transfer
#define _ISO          0x01          //Isochronous Transfer
#define _BULK         0x02          //Bulk Transfer
#define _INT          0x03          //Interrupt Transfer

/* Isochronous Endpoint Synchronization Type */
#define _NS           0x00<<2      //No Synchronization
#define _AS           0x01<<2      //Asynchronous
#define _AD           0x02<<2      //Adaptive
#define _SY           0x03<<2      //Synchronous

/* Isochronous Endpoint Usage Type */
#define _DE           0x00<<4      //Data endpoint
#define _FE           0x01<<4      //Feedback endpoint
#define _IE           0x02<<4      //Implicit feedback Data endpoint

/** S T R U C T U R E *****/
/*****
 * USB Device Descriptor Structure
 *****/
typedef struct _USB_DEV_DSC
{
    byte bLength;      byte bDscType;      word bcdUSB;
    byte bDevCls;      byte bDevSubCls;     byte bDevProtocol;
    byte bMaxPktSize0; word idVendor;      word idProduct;
    word bcdDevice;    byte iMFR;          byte iProduct;
    byte iSerialNum;   byte bNumCfg;
} USB_DEV_DSC;

/*****
 * USB Configuration Descriptor Structure
 *****/
typedef struct _USB_CFG_DSC
{
    byte bLength;      byte bDscType;      word wTotalLength;
    byte bNumIntf;     byte bCfgValue;     byte iCfg;
    byte bmAttributes; byte bMaxPower;
} USB_CFG_DSC;

/*****
 * USB Interface Descriptor Structure
 *****/
typedef struct _USB_INTF_DSC
{
    byte bLength;      byte bDscType;      byte bIntfNum;
    byte bAltSetting;  byte bNumEPs;       byte bIntfCls;
    byte bIntfSubCls;  byte bIntfProtocol; byte iIntf;
} USB_INTF_DSC;

/*****
 * USB Endpoint Descriptor Structure
 *****/
typedef struct _USB_EP_DSC
{
    byte bLength;      byte bDscType;      byte bEPAddr;
    byte bmAttributes; word wMaxPktSize;   byte bInterval;
} USB_EP_DSC;

#endif //USBDEFS_STD_DSC_H
```

# Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbdrv.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.7.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

#ifdef USBDRV_H
#define USBDRV_H

/** I N C L U D E S *****/
#include "system\typedefs.h"
#include "system\usb\usb.h"

/** D E F I N I T I O N S *****/

/* UCFG Initialization Parameters */
#define _PPBM0      0x00      // Pingpong Buffer Mode 0
#define _PPBM1      0x01      // Pingpong Buffer Mode 1
#define _PPBM2      0x02      // Pingpong Buffer Mode 2
#define _LS          0x00      // Use Low-Speed USB Mode
#define _FS          0x04      // Use Full-Speed USB Mode
#define _TRINT       0x00      // Use internal transceiver
#define _TREXT       0x08      // Use external transceiver
#define _PUEN        0x10      // Use internal pull-up resistor
#define _OEMON        0x40      // Use SIE output indicator
#define _UTEYE        0x80      // Use Eye-Pattern test

/* UEPn Initialization Parameters */
#define EP_CTRL      0x06      // Cfg Control pipe for this ep
#define EP_OUT       0x0C      // Cfg OUT only pipe for this ep
#define EP_IN        0x0A      // Cfg IN only pipe for this ep
#define EP_OUT_IN    0x0E      // Cfg both OUT & IN pipes for this ep
#define HSHK_EN      0x10      // Enable handshake packet
                                // Handshake should be disable for isoch

/*****
* USB - PICmicro Endpoint Definitions
* PICmicro EP Address Format: X:EP3:EP2:EP1:EP0:DIR:PPBI:X
* This is used when checking the value read from USTAT
*
* NOTE: These definitions are not used in the descriptors.
* EP addresses used in the descriptors have different format and
* are defined in: "system\usb\usbdefs\usbdefs_std_dsc.h"
*****/
#define OUT          0
#define IN            1

#define PIC_EP_NUM_MASK 0b01111000
#define PIC_EP_DIR_MASK 0b00000100

#define EP00_OUT      (0x00<<3)|(OUT<<2)
#define EP00_IN       (0x00<<3)|(IN<<2)
#define EP01_OUT      (0x01<<3)|(OUT<<2)
#define EP01_IN       (0x01<<3)|(IN<<2)
#define EP02_OUT      (0x02<<3)|(OUT<<2)
#define EP02_IN       (0x02<<3)|(IN<<2)
#define EP03_OUT      (0x03<<3)|(OUT<<2)
#define EP03_IN       (0x03<<3)|(IN<<2)
#define EP04_OUT      (0x04<<3)|(OUT<<2)
#define EP04_IN       (0x04<<3)|(IN<<2)
#define EP05_OUT      (0x05<<3)|(OUT<<2)
#define EP05_IN       (0x05<<3)|(IN<<2)
#define EP06_OUT      (0x06<<3)|(OUT<<2)

```

## Wireless USB Project Design Report

```
#define EP06_IN      (0x06<<3)|(IN<<2)
#define EP07_OUT    (0x07<<3)|(OUT<<2)
#define EP07_IN     (0x07<<3)|(IN<<2)
#define EP08_OUT    (0x08<<3)|(OUT<<2)
#define EP08_IN     (0x08<<3)|(IN<<2)
#define EP09_OUT    (0x09<<3)|(OUT<<2)
#define EP09_IN     (0x09<<3)|(IN<<2)
#define EP10_OUT    (0x0A<<3)|(OUT<<2)
#define EP10_IN     (0x0A<<3)|(IN<<2)
#define EP11_OUT    (0x0B<<3)|(OUT<<2)
#define EP11_IN     (0x0B<<3)|(IN<<2)
#define EP12_OUT    (0x0C<<3)|(OUT<<2)
#define EP12_IN     (0x0C<<3)|(IN<<2)
#define EP13_OUT    (0x0D<<3)|(OUT<<2)
#define EP13_IN     (0x0D<<3)|(IN<<2)
#define EP14_OUT    (0x0E<<3)|(OUT<<2)
#define EP14_IN     (0x0E<<3)|(IN<<2)
#define EP15_OUT    (0x0F<<3)|(OUT<<2)
#define EP15_IN     (0x0F<<3)|(IN<<2)

/*****
 * Macro:          void mInitializeUSBDriver(void)
 * Overview:       Configures the USB module, definition of UCFG_VAL can be
 *                 found in autofiles\usbcfg.h
 *
 *                 This register determines: USB Speed, On-chip pull-up
 *                 resistor selection, On-chip tranceiver selection, bus
 *                 eye pattern generation mode, Ping-pong buffering mode
 *                 selection.
 *****/
#define mInitializeUSBDriver() {UCFG = UCFG_VAL;          \
                               usb_device_state = DETACHED_STATE; \
                               usb_stat._byte = 0x00;          \
                               usb_active_cfg = 0x00;          \
                               }

/*****
 * Macro:          void mDisableEP1to15(void)
 * Overview:       This macro disables all endpoints except EP0.
 *                 This macro should be called when the host sends a RESET
 *                 signal or a SET_CONFIGURATION request.
 *****/
#define mDisableEP1to15()      ClearArray((byte*)&UEP1,15);
/*
#define mDisableEP1to15()      UEP1=0x00;UEP2=0x00;UEP3=0x00;\
                               UEP4=0x00;UEP5=0x00;UEP6=0x00;UEP7=0x00;\
                               UEP8=0x00;UEP9=0x00;UEP10=0x00;UEP11=0x00;\
                               UEP12=0x00;UEP13=0x00;UEP14=0x00;UEP15=0x00;

*/

/*****
 * Macro:          void mUSBBufferReady(buffer_dsc)
 * PreCondition:   IN Endpoint: Buffer is loaded and ready to be sent.
 *                 OUT Endpoint: Buffer is free to be written to by SIE.
 *
 * Input:          byte buffer_dsc: Root name of the buffer descriptor group.
 *                 i.e. ep0Bo, ep1Bi, ... Declared in usbmmap.c
 *                 Names can be remapped for readability, see examples in
 *                 usbcfg.h (#define HID_BD_OUT      ep1Bo)
 *
 * Output:         None
 *
 * Side Effects:   None
 *
 * Overview:       This macro should be called each time after:
 *                 1. A non-EP0 IN endpoint buffer is populated with data.
 *                 2. A non-EP0 OUT endpoint buffer is read.
 *                 This macro turns the buffer ownership to SIE for servicing.
 *                 It also toggles the DTS bit for synchronization.
 *****/
```

## Wireless USB Project Design Report

```
* Note:          None
*****
#define mUSBBufferReady(buffer_dsc)
{
    buffer_dsc.Stat._byte &= _DTSMASK;          /* Save only DTS bit */          \
    buffer_dsc.Stat.DTS = !buffer_dsc.Stat.DTS; /* Toggle DTS bit */          \
    buffer_dsc.Stat._byte |= _USIE|_DTSEN;      /* Turn ownership to SIE */    \
}

/** T Y P E S *****/

/** E X T E R N S *****/

/** P U B L I C   P R O T O T Y P E S *****/
void USBCheckBusStatus(void);
void USBDriverService(void);
void USBRemoteWakeup(void);
void USBSoftDetach(void);
void ClearArray(byte* startAdr,byte count);

#endif //USBDRV_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usbdrv.c
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

/** I N C L U D E S *****/
//#include <p18cxxx.h>
#include <p18f4550.h>
#include "system\typedefs.h"
#include "system\usb\usb.h"
#include "io_cfg.h"           // Required for USBCheckBusStatus()

/** V A R I A B L E S *****/
#pragma udata

/** P R I V A T E   P R O T O T Y P E S *****/
void USBModuleEnable(void);
void USBModuleDisable(void);
void USBSuspend(void);
void USBWakeFromSuspend(void);
void USBProtocolResetHandler(void);
void USB_SOF_Handler(void);
void USBStallHandler(void);
void USBErrorHandler(void);

/** D E C L A R A T I O N S *****/
#pragma code
/*****
* Function:        void USBCheckBusStatus(void)
* Overview:        This routine enables/disables the USB module by monitoring
*                  the USB power signal.
*****/
void USBCheckBusStatus(void)
{
    /*****
    * Bus Attachment & Detachment Detection
    * usb_bus_sense is an i/o pin defined in io_cfg.h
    *****/
    #define USB_BUS_ATTACHED    1
    #define USB_BUS_DETACHED    0

    if(usb_bus_sense == USB_BUS_ATTACHED)           // Is USB bus attached?
    {
        if(UCONbits USBEN == 0)                     // Is the module off?
            USBModuleEnable();                       // Is off, enable it
        }
    else
    {
        if(UCONbits USBEN == 1)                     // Is the module on?
            USBModuleDisable();                      // Is on, disable it
    } //end if(usb_bus_sense...)

    /*
    * After enabling the USB module, it takes some time for the voltage
    * on the D+ or D- line to rise high enough to get out of the SE0 condition.
    * The USB Reset interrupt should not be unmasked until the SE0 condition is
    * cleared. This helps preventing the firmware from misinterpreting this
    * unique event as a USB bus reset from the USB host.
    */
    if(usb_device_state == ATTACHED_STATE)
    {

```

## Wireless USB Project Design Report

```
        if(!UCONbits.SE0)
        {
            UIR = 0;                // Clear all USB interrupts
            UIE = 0;                // Mask all USB interrupts
            UIEbits.URSTIE = 1;    // Unmask RESET interrupt
            UIEbits.IDLEIE = 1;    // Unmask IDLE interrupt
            usb_device_state = POWERED_STATE;
        } //end if                // else wait until SE0 is cleared
    } //end if(usb_device_state == ATTACHED_STATE)

} //end USBCheckBusStatus

/*****
 * Function:      void USBModuleEnable(void)
 * Overview:      This routine enables the USB module.
 *                An end designer should never have to call this routine
 *                manually. This routine should only be called from
 *                USBCheckBusStatus().
 *
 * Note:          See USBCheckBusStatus() for more information.
 *****/
void USBModuleEnable(void)
{
    UCON = 0;
    UIE = 0;                // Mask all USB interrupts
    UCONbits.USBEN = 1;    // Enable module & attach to bus
    usb_device_state = ATTACHED_STATE;    // Defined in usbmmmap.c & .h
} //end USBModuleEnable

/*****
 * Function:      void USBModuleDisable(void)
 * Overview:      This routine disables the USB module.
 *                An end designer should never have to call this routine
 *                manually. This routine should only be called from
 *                USBCheckBusStatus().
 *
 * Note:          See USBCheckBusStatus() for more information.
 *****/
void USBModuleDisable(void)
{
    UCON = 0;                // Disable module & detach from bus
    UIE = 0;                // Mask all USB interrupts
    usb_device_state = DETACHED_STATE;    // Defined in usbmmmap.c & .h
} //end USBModuleDisable

/*****
 * Function:      void USBSoftDetach(void)
 * Side Effects:  The device will have to be re-enumerated to function again.
 *
 * Overview:      USBSoftDetach electrically disconnects the device from
 *                the bus. This is done by stop supplying Vusb voltage to
 *                pull-up resistor. The pull-down resistors on the host
 *                side will pull both differential signal lines low and
 *                the host registers the event as a disconnect.
 *
 *                Since the USB cable is not physically disconnected, the
 *                power supply through the cable can still be sensed by
 *                the device. The next time USBCheckBusStatus() function
 *                is called, it will reconnect the device back to the bus.
 *****/
void USBSoftDetach(void)
{
    USBModuleDisable();
} //end USBSoftDetach

/*****
 * Function:      void USBDriverService(void)
 * Overview:      This routine is the heart of this firmware. It manages
 *                all USB interrupts.
 *
 * Note:          Device state transitions through the following stages:
```

## Wireless USB Project Design Report

```
*          DETACHED -> ATTACHED -> POWERED -> DEFAULT ->
*          ADDRESS_PENDING -> ADDRESSED -> CONFIGURED -> READY
*****/
void USBDriverService(void)
{
    /*
    * Pointless to continue servicing if USB cable is not even attached.
    */
    if(usb_device_state == DETACHED_STATE) return;

    /*
    * Task A: Service USB Activity Interrupt
    */
    if(UIRbits.ACTVIF && UIEbits.ACTVIE)    USBWakeFromSuspend();

    /*
    * Pointless to continue servicing if the device is in suspend mode.
    */
    if(UCONbits.SUSPND==1) return;

    /*
    * Task B: Service USB Bus Reset Interrupt.
    * When bus reset is received during suspend, ACTVIF will be set first,
    * once the UCONbits.SUSPND is clear, then the URSTIF bit will be asserted.
    * This is why URSTIF is checked after ACTVIF.
    *
    * The USB reset flag is masked when the USB state is in
    * DETACHED_STATE or ATTACHED_STATE, and therefore cannot
    * cause a USB reset event during these two states.
    */
    if(UIRbits.URSTIF && UIEbits.URSTIE)    USBProtocolResetHandler();

    /*
    * Task C: Service other USB interrupts
    */
    if(UIRbits.IDLEIF && UIEbits.IDLEIE)    USBSuspend();
    if(UIRbits.SOFIF && UIEbits.SOFIE)      USB_SOF_Handler();
    if(UIRbits.STALLIF && UIEbits.STALLIE)   USBStallHandler();
    if(UIRbits.UERRIF && UIEbits.UERRIE)    USBErrorHandler();

    /*
    * Pointless to continue servicing if the host has not sent a bus reset.
    * Once bus reset is received, the device transitions into the DEFAULT
    * state and is ready for communication.
    */
    if(usb_device_state < DEFAULT_STATE) return;

    /*
    * Task D: Servicing USB Transaction Complete Interrupt
    */
    if(UIRbits.TRNIF && UIEbits.TRNIE)
    {
        /*
        * USBCtrlEPService only services transactions over EP0.
        * It ignores all other EP transactions.
        */
        USBCtrlEPService();

        /*
        * Other EP can be serviced later by responsible device class firmware.
        * Each device driver knows when an OUT or IN transaction is ready by
        * checking the buffer ownership bit.
        * An OUT EP should always be owned by SIE until the data is ready.
        * An IN EP should always be owned by CPU until the data is ready.
        *
        * Because of this logic, it is not necessary to save the USTAT value
        * of non-EP0 transactions.
        */
        UIRbits.TRNIF = 0;
    } //end if(UIRbits.TRNIF && UIEbits.TRNIE)
}
```



## Wireless USB Project Design Report

```
//end USBDriverService

/*****
 * Function:      void USBSuspend(void)
 *****/
void USBSuspend(void)
{
    /*
     * NOTE: Do not clear UIRbits.ACTVIF here!
     * Reason:
     * ACTVIF is only generated once an IDLEIF has been generated.
     * This is a 1:1 ratio interrupt generation.
     * For every IDLEIF, there will be only one ACTVIF regardless of
     * the number of subsequent bus transitions.
     *
     * If the ACTIF is cleared here, a problem could occur when:
     * [      IDLE      ][bus activity ->
     * <--- 3 ms ---->   ^
     *                   ^ ACTVIF=1
     *                   IDLEIF=1
     * #           #           #           #   (#=Program polling flags)
     *
     * This polling loop will see both
     * IDLEIF=1 and ACTVIF=1.
     * However, the program services IDLEIF first
     * because ACTIVIE=0.
     * If this routine clears the only ACTVIF,
     * then it can never get out of the suspend
     * mode.
     */
    UIEbits.ACTVIE = 1;                // Enable bus activity interrupt
    UIRbits.IDLEIF = 0;
    UCONbits.SUSPND = 1;                // Put USB module in power conserve
                                        // mode, SIE clock inactive

    /*
     * At this point the PIC can go into sleep,idle, or
     * switch to a slower clock, etc.
     */

    /* Modifiable Section */
    PIR2bits.USBIF = 0;
    INTCONbits.RBIF = 0;
    PIE2bits.USBIE = 1;                // Set USB wakeup source
    INTCONbits.RBIE = 1;                // Set sw2,3 wakeup source
    Sleep();                           // Goto sleep

    if(INTCONbits.RBIF == 1)            // Check if external stimulus
    {
        USBRemoteWakeup();              // If yes, attempt RWU
    }
    PIE2bits.USBIE = 0;
    INTCONbits.RBIE = 0;
    /* End Modifiable Section */
}

//end USBSuspend

/*****
 * Function:      void USBWakeFromSuspend(void)
 *****/
void USBWakeFromSuspend(void)
{
    /*
     * If using clock switching, this is the place to restore the
     * original clock frequency.
     */
    UCONbits.SUSPND = 0;
    UIEbits.ACTVIE = 0;
    UIRbits.ACTVIF = 0;
}

//end USBWakeFromSuspend

/*****/
```

## Wireless USB Project Design Report

```
* Function:      void USBRemoteWakeup(void)
* Overview:      This function should be called by user when the device
*                is waken up by an external stimulus other than ACTIVIF.
*                Please read the note below to understand the limitations.
*
* Note:          The modifiable section in this routine should be changed
*                to meet the application needs. Current implementation
*                temporary blocks other functions from executing for a
*                period of 1-13 ms depending on the core frequency.
*
*                According to USB 2.0 specification section 7.1.7.7,
*                "The remote wakeup device must hold the resume signaling
*                for at least 1 ms but for no more than 15 ms."
*                The idea here is to use a delay counter loop, using a
*                common value that would work over a wide range of core
*                frequencies.
*                That value selected is 1800. See table below:
*                =====
*                Core Freq(MHz)      MIP      RESUME Signal Period (ms)
*                =====
*                48      12      1.05
*                4       1     12.6
*                =====
*                * These timing could be incorrect when using code
*                optimization or extended instruction mode,
*                or when having other interrupts enabled.
*                Make sure to verify using the MPLAB SIM's Stopwatch
*****/
void USBRemoteWakeup(void)
{
    static word delay_count;

    if(usb_stat.RemoteWakeup == 1)          // Check if RemoteWakeup function
    {                                       // has been enabled by the host.
        USBWakeFromSuspend();             // Unsuspend USB module
        UCONbits.RESUME = 1;              // Start RESUME signaling

        /* Modifiable Section */
        delay_count = 1800U;              // Set RESUME line for 1-13 ms
        do
        {
            delay_count--;
        }while(delay_count);
        /* End Modifiable Section */

        UCONbits.RESUME = 0;
    }//endif
}//end USBRemoteWakeup

/*****
* Function:      void USB_SOF_Handler(void)
* Overview:      The USB host sends out a SOF packet to full-speed devices
*                every 1 ms. This interrupt may be useful for isochronous
*                pipes. End designers should implement callback routine
*                as necessary.
*
* Note:          None
*****/
void USB_SOF_Handler(void)
{
    /* Callback routine here */
    UIRbits.SOFIF = 0;
}//end USB_SOF_Handler

/*****
* Function:      void USBStallHandler(void)
*
* PreCondition:  A STALL packet is sent to the host by the SIE.
* Overview:      The STALLIF is set anytime the SIE sends out a STALL
*                packet regardless of which endpoint causes it.
*                A Setup transaction overrides the STALL function. A stalled
```

## Wireless USB Project Design Report

```
*          endpoint stops stalling once it receives a setup packet.
*          In this case, the SIE will accept the Setup packet and
*          set the TRNIF flag to notify the firmware. STALL function
*          for that particular endpoint pipe will be automatically
*          disabled (direction specific).
*
*          There are a few reasons for an endpoint to be stalled.
*          1. When a non-supported USB request is received.
*             Example: GET_DESCRIPTOR(DEVICE_QUALIFIER)
*          2. When an endpoint is currently halted.
*          3. When the device class specifies that an endpoint must
*             stall in response to a specific event.
*             Example: Mass Storage Device Class
*                   If the CBW is not valid, the device shall
*                   STALL the Bulk-In pipe.
*                   See USB Mass Storage Class Bulk-only Transport
*                   Specification for more details.
*
* Note:          UEPn.EPSTALL can be scanned to see which endpoint causes
*                the stall event.
*                If
*****/
void USBStallHandler(void)
{
    /*
     * Does not really have to do anything here,
     * even for the control endpoint.
     * All BDs of Endpoint 0 are owned by SIE right now,
     * but once a Setup Transaction is received, the ownership
     * for EP0_OUT will be returned to CPU.
     * When the Setup Transaction is serviced, the ownership
     * for EP0_IN will then be forced back to CPU by firmware.
     */
    if(UEP0bits.EPSTALL == 1)
    {
        USBPrepareForNextSetupTrf();           // Firmware work-around
        UEP0bits.EPSTALL = 0;
    }
    UIRbits.STALLIF = 0;
} //end USBStallHandler

/*****
 * Function:      void USBErrorHandler(void)
 * Overview:      The purpose of this interrupt is mainly for debugging
 *                during development. Check UEIR to see which error causes
 *                the interrupt.
 *
 * Note:          None
*****/
void USBErrorHandler(void)
{
    UIRbits.UERRIF = 0;
} //end USBErrorHandler

/*****
 * Function:      void USBProtocolResetHandler(void)
 *
 * PreCondition:  A USB bus reset is received from the host.
 * Side Effects:  Currently, this routine flushes any pending USB
 *                transactions. It empties out the USTAT FIFO. This action
 *                might not be desirable in some applications.
 *
 * Overview:      Once a USB bus reset is received from the host, this
 *                routine should be called. It resets the device address to
 *                zero, disables all non-EP0 endpoints, initializes EP0 to
 *                be ready for default communication, clears all USB
 *                interrupt flags, unmask applicable USB interrupts, and
 *                reinitializes internal state-machine variables.
 *
 * Note:          None
*****/
```

## Wireless USB Project Design Report

```
void USBProtocolResetHandler(void)
{
    UEIR = 0;                // Clear all USB error flags
    UIR = 0;                 // Clears all USB interrupts
    UEIE = 0b10011111;      // Unmask all USB error interrupts
    UIE = 0b01111011;       // Enable all interrupts except ACTVIE

    UADDR = 0x00;            // Reset to default address
    mDisableEP1to15();       // Reset all non-EP0 UEPn registers
    UEP0 = EP_CTRL|HSHK_EN;  // Init EP0 as a Ctrl EP, see usbdrv.h

    while(UIRbits.TRNIF == 1) // Flush any pending transactions
        UIRbits.TRNIF = 0;

    UCONbits.PKTDIS = 0;     // Make sure packet processing is enabled
    USBPrepareForNextSetupTrf(); // Declared in usbctrltrf.c

    usb_stat.RemoteWakeUp = 0; // Default status flag to disable
    usb_active_cfg = 0;        // Clear active configuration
    usb_device_state = DEFAULT_STATE;
} //end USBProtocolResetHandler

/* Auxiliary Function */
void ClearArray(byte* startAdr, byte count)
{
    *startAdr;
    while(count)
    {
        _asm
            clrf POSTINC0,0
        _endasm
        count--;
    } //end while
} //end ClearArray

/** EOF usbdrv.c *****/
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbmmap.c
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.7.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

/*****
* -usbmmap.c-
* USB Memory Map
* This file is the USB memory manager; it serves as a compile-time memory
* allocator for the USB endpoints. It uses the compile time options passed
* from usbcfg.h to instantiate endpoints and endpoint buffer.
*
* Each endpoint requires to have a set of Buffer Descriptor registers(BDT).
* A BDT is 4-byte long and has a specific RAM location for each endpoint.
* The BDT for endpoint 0 out is located at address 0x400 to 0x403.
* The BDT for endpoint 0 in is located at address 0x404 to 0x407.
* The BDT for endpoint 1 out is located at address 0x408 to 0x40B.
* and so on... The above allocation assumes the Ping-Pong Buffer Mode 0 is
* used. These locations are already hard-wired in the silicon. The point
* of doing instantiation, i.e. volatile far BDT ep0Bo;, is to provide the
* C compiler a way to address each variable directly. This is very important
* because when a register can be accessed directly, it saves execution time
* and reduces program size.
*
* Endpoints are defined using the endpoint number and the direction
* of transfer. For simplicity, usbmmap.c only uses the endpoint number
* in the BDT register allocation scheme. This means if the usbcfg.h states
* that the MAX_EP_NUMBER is number 1, then four BDTs will be
* instantiated: one each for endpoint0 in and endpoint0 out, which must
* always be instantiated for control transfer by default, and one each sets
* for endpoint1 in and endpoint1 out. The naming convention for instantiating
* BDT is
*
* ep<#>B<d>
*
* where # is the endpoint number, and d is the direction of
* transfer, which could be either <i> or <o>.
*
* The USB memory manager uses MAX_EP_NUMBER, as defined in usbcfg.h, to define
* the endpoints to be instantiated. This represents the highest endpoint
* number to be allocated, not how many endpoints are used. Since the BDTs for
* endpoints have hardware-assigned addresses in Bank 4, setting this value too
* high may lead to inefficient use of data RAM. For example, if an application
* uses only endpoints EP0 and EP4, then the MAX_EP_NUMBER is 4, and not 2.
* The in-between endpoint BDTs in this example (EP1, EP2, and EP3) go unused,
* and the 24 bytes of memory associated with them are wasted. It does not make
* much sense to skip endpoints, but the final decision lies with the user.
*
* The next step is to assign the instantiated BDTs to different
* USB functions. The firmware framework fundamentally assumes that every USB
* function should know which endpoint it is using, i.e., the default control
* transfer should know that it is using endpoint 0 in and endpoint 0 out.
* A HID class can choose which endpoint it wants to use, but once chosen, it
* should always know the number of the endpoint.
*
* The assignment of endpoints to USB functions is managed centrally
* in usbcfg.h. This helps prevent the mistake of having more
* than one USB function using the same endpoint. The "Endpoint Allocation"
* section in usbcfg.h provides examples for how to map USB endpoints to USB
* functions.
* Quite a few things can be mapped in that section. There is no
* one correct way to do the mapping and the user has the choice to

```

## Wireless USB Project Design Report

```
* choose a method that is most suitable to the application.
*
* Typically, however, a user will want to map the following for a given
* USB interface function:
* 1. The USB interface ID
* 2. The endpoint control registers (UEPn)
* 3. The BDT registers (ep<#>B<d>)
* 4. The endpoint size
*
* Example: Assume a USB device class "foo", which uses one out endpoint
*         of size 64-byte and one in endpoint of size 64-byte, then:
*
* #define FOO_INTF_ID          0x00
* #define FOO_UEP              UEP1
* #define FOO_BD_OUT           ep1Bo
* #define FOO_BD_IN            ep1Bi
* #define FOO_EP_SIZE          64
*
* The mapping above has chosen class "foo" to use endpoint 1.
* The names are arbitrary and can be anything other than FOO_?????.
* For abstraction, the code for class "foo" should use the abstract
* definitions of FOO_BD_OUT, FOO_BD_IN, and not ep1Bo or ep1Bi.
*
* Note that the endpoint size defined in the usbcfg.h file is again
* used in the usbmmmap.c file. This shows that the relationship between
* the two files are tightly related.
*
* The endpoint buffer for each USB function must be located in the
* dual-port RAM area and has to come after all the BDTs have been
* instantiated. An example declaration is:
* volatile far unsigned char[FOO_EP_SIZE] data;
*
* The 'volatile' keyword tells the compiler not to perform any code
* optimization on this variable because its content could be modified
* by the hardware. The 'far' keyword tells the compiler that this variable
* is not located in the Access RAM area (0x000 - 0x05F).
*
* For the variable to be globally accessible by other files, it should be
* declared in the header file usbmmmap.h as an extern definition, such as
* extern volatile far unsigned char[FOO_EP_SIZE] data;
*
* Conclusion:
* In a short summary, the dependencies between usbcfg and usbmmmap can
* be shown as:
*
* usbcfg[MAX_EP_NUMBER] -> usbmmmap
* usbmmmap[ep<#>B<d>] -> usbcfg
* usbcfg[EP size] -> usbmmmap
* usbcfg[abstract ep definitions] -> usb9/hid/cdc/etc class code
* usbmmmap[endpoint buffer variable] -> usb9/hid/cdc/etc class code
*
* Data mapping provides a means for direct addressing of BDT and endpoint
* buffer. This means less usage of pointers, which equates to a faster and
* smaller program code.
*****/

/** I N C L U D E S *****/
#include "system\types.h"
#include "system\usb\usb.h"

/** U S B   G L O B A L   V A R I A B L E S *****/
#pragma udata
byte usb_device_state;          // Device States: DETACHED, ATTACHED, ...
USB_DEVICE_STATUS usb_stat;     // Global USB flags
byte usb_active_cfg;           // Value of current configuration
byte usb_alt_intf[MAX_NUM_INT]; // Array to keep track of the current alternate
                                // setting for each interface ID

/** U S B   F I X E D   L O C A T I O N   V A R I A B L E S *****/
#pragma udata usbram4=0x400      //See Linker Script,usb4:0x400-0x4FF(256-byte)
```

## Wireless USB Project Design Report

```

/*****
 * Section A: Buffer Descriptor Table
 * - 0x400 - 0x4FF(max)
 * - MAX_EP_NUMBER is defined in autofiles\usbcfg.h
 * - BDT data type is defined in system\usb\usbmmmap.h
 *****/

#if(0 <= MAX_EP_NUMBER)
volatile far BDT ep0Bo;           //Endpoint #0 BD Out
volatile far BDT ep0Bi;           //Endpoint #0 BD In
#endif

#if(1 <= MAX_EP_NUMBER)
volatile far BDT ep1Bo;           //Endpoint #1 BD Out
volatile far BDT ep1Bi;           //Endpoint #1 BD In
#endif

#if(2 <= MAX_EP_NUMBER)
volatile far BDT ep2Bo;           //Endpoint #2 BD Out
volatile far BDT ep2Bi;           //Endpoint #2 BD In
#endif

#if(3 <= MAX_EP_NUMBER)
volatile far BDT ep3Bo;           //Endpoint #3 BD Out
volatile far BDT ep3Bi;           //Endpoint #3 BD In
#endif

#if(4 <= MAX_EP_NUMBER)
volatile far BDT ep4Bo;           //Endpoint #4 BD Out
volatile far BDT ep4Bi;           //Endpoint #4 BD In
#endif

#if(5 <= MAX_EP_NUMBER)
volatile far BDT ep5Bo;           //Endpoint #5 BD Out
volatile far BDT ep5Bi;           //Endpoint #5 BD In
#endif

#if(6 <= MAX_EP_NUMBER)
volatile far BDT ep6Bo;           //Endpoint #6 BD Out
volatile far BDT ep6Bi;           //Endpoint #6 BD In
#endif

#if(7 <= MAX_EP_NUMBER)
volatile far BDT ep7Bo;           //Endpoint #7 BD Out
volatile far BDT ep7Bi;           //Endpoint #7 BD In
#endif

#if(8 <= MAX_EP_NUMBER)
volatile far BDT ep8Bo;           //Endpoint #8 BD Out
volatile far BDT ep8Bi;           //Endpoint #8 BD In
#endif

#if(9 <= MAX_EP_NUMBER)
volatile far BDT ep9Bo;           //Endpoint #9 BD Out
volatile far BDT ep9Bi;           //Endpoint #9 BD In
#endif

#if(10 <= MAX_EP_NUMBER)
volatile far BDT ep10Bo;          //Endpoint #10 BD Out
volatile far BDT ep10Bi;          //Endpoint #10 BD In
#endif

#if(11 <= MAX_EP_NUMBER)
volatile far BDT ep11Bo;          //Endpoint #11 BD Out
volatile far BDT ep11Bi;          //Endpoint #11 BD In
#endif

#if(12 <= MAX_EP_NUMBER)
volatile far BDT ep12Bo;          //Endpoint #12 BD Out
volatile far BDT ep12Bi;          //Endpoint #12 BD In
#endif

```

## Wireless USB Project Design Report

```
#if(13 <= MAX_EP_NUMBER)
volatile far BDT ep13Bo;          //Endpoint #13 BD Out
volatile far BDT ep13Bi;          //Endpoint #13 BD In
#endif

#if(14 <= MAX_EP_NUMBER)
volatile far BDT ep14Bo;          //Endpoint #14 BD Out
volatile far BDT ep14Bi;          //Endpoint #14 BD In
#endif

#if(15 <= MAX_EP_NUMBER)
volatile far BDT ep15Bo;          //Endpoint #15 BD Out
volatile far BDT ep15Bi;          //Endpoint #15 BD In
#endif

/*****
 * Section B: EP0 Buffer Space
 *****/
* - Two buffer areas are defined:
*
*   A. CTRL_TRF_SETUP
*   - Size = EP0_BUFF_SIZE as defined in autofiles\usbcfg.h
*   - Detailed data structure allows direct addressing of bits and bytes.
*
*   B. CTRL_TRF_DATA
*   - Size = EP0_BUFF_SIZE as defined in autofiles\usbcfg.h
*   - Data structure allows direct addressing of the first 8 bytes.
*
* - Both data types are defined in system\usb\usbdefs\usb_defs.h
*****/
volatile far CTRL_TRF_SETUP SetupPkt;
volatile far CTRL_TRF_DATA CtrlTrfData;

/*****
 * Section C: MSD Buffer
 *****/
*/
#if defined(USB_USE_MSD)
//volatile far USB_MSD_CBW_CSW msd_cbw_csw;
volatile far USB_MSD_CBW msd_cbw;
volatile far USB_MSD_CSW msd_csw;
//#pragma udata
#pragma udata myMSD=0x600
volatile far char msd_buffer[512];
#endif
//#pragma udata

/*****
 * Section C: HID Buffer
 *****/
*/
#if defined(USB_USE_HID)
volatile far unsigned char hid_report_out[HID_INT_OUT_EP_SIZE];
volatile far unsigned char hid_report_in[HID_INT_IN_EP_SIZE];
#endif
#pragma udata
*/

/** EOF usbmmmap.c *****/
```



## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:          usbmmmap.h
/      Authors:           Andrew Knight
/                          Design Team 5 Wireless USB
/      Version:           4.7.2006
/      Processor:         PIC18F4550
/      Compiler:          Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****/

#ifndef USBMMAP_H
#define USBMMAP_H

/** I N C L U D E S *****/
#include "system\typedefs.h"

/** D E F I N I T I O N S *****/

/* Buffer Descriptor Status Register Initialization Parameters */
#define _BSTALL      0x04           //Buffer Stall enable
#define _DTSEN       0x08           //Data Toggle Synch enable
#define _INCDIS      0x10           //Address increment disable
#define _KEN         0x20           //SIE keeps buff descriptors enable
#define _DAT0        0x00           //DATA0 packet expected next
#define _DAT1        0x40           //DATA1 packet expected next
#define _DTSMASK     0x40           //DTS Mask
#define _USIE        0x80           //SIE owns buffer
#define _UCPU        0x00           //CPU owns buffer

/* USB Device States - To be used with [byte usb_device_state] */
#define DETACHED_STATE      0
#define ATTACHED_STATE      1
#define POWERED_STATE       2
#define DEFAULT_STATE       3
#define ADR_PENDING_STATE   4
#define ADDRESS_STATE       5
#define CONFIGURED_STATE    6

/* Memory Types for Control Transfer - used in USB_DEVICE_STATUS */
#define _RAM 0
#define _ROM 1

/** T Y P E S *****/
typedef union _USB_DEVICE_STATUS
{
    byte _byte;
    struct
    {
        unsigned RemoteWakeup:1; // [0]Disabled [1]Enabled: See usbdrv.c,usb9.c
        unsigned ctrl_trf_mem:1; // [0]RAM      [1]ROM
    };
} USB_DEVICE_STATUS;

typedef union _BD_STAT
{
    byte _byte;
    struct{
        unsigned BC8:1;
        unsigned BC9:1;
        unsigned BSTALL:1;           //Buffer Stall Enable
        unsigned DTSEN:1;           //Data Toggle Synch Enable
        unsigned INCDIS:1;          //Address Increment Disable
        unsigned KEN:1;             //BD Keep Enable
        unsigned DTS:1;             //Data Toggle Synch Value
        unsigned UOWN:1;            //USB Ownership
    };
    struct{

```

## Wireless USB Project Design Report

```
        unsigned BC8:1;
        unsigned BC9:1;
        unsigned PID0:1;
        unsigned PID1:1;
        unsigned PID2:1;
        unsigned PID3:1;
        unsigned :1;
        unsigned UOWN:1;
    };
    struct{
        unsigned :2;
        unsigned PID:4;           //Packet Identifier
        unsigned :2;
    };
} BD_STAT;                       //Buffer Descriptor Status Register

typedef union _BDT
{
    struct
    {
        {
            BD_STAT Stat;
            byte Cnt;
            byte ADRL;           //Buffer Address Low
            byte ADRH;          //Buffer Address High
        };
        struct
        {
            unsigned :8;
            unsigned :8;
            byte* ADR;           //Buffer Address
        };
    };
} BDT;                           //Buffer Descriptor Table

/** E X T E R N S *****/
extern byte usb_device_state;
extern USB_DEVICE_STATUS usb_stat;
extern byte usb_active_cfg;
extern byte usb_alt_intf[MAX_NUM_INT];

extern volatile far BDT ep0Bo;   //Endpoint #0 BD Out
extern volatile far BDT ep0Bi;   //Endpoint #0 BD In
extern volatile far BDT ep1Bo;   //Endpoint #1 BD Out
extern volatile far BDT ep1Bi;   //Endpoint #1 BD In
extern volatile far BDT ep2Bo;   //Endpoint #2 BD Out
extern volatile far BDT ep2Bi;   //Endpoint #2 BD In
extern volatile far BDT ep3Bo;   //Endpoint #3 BD Out
extern volatile far BDT ep3Bi;   //Endpoint #3 BD In
extern volatile far BDT ep4Bo;   //Endpoint #4 BD Out
extern volatile far BDT ep4Bi;   //Endpoint #4 BD In
extern volatile far BDT ep5Bo;   //Endpoint #5 BD Out
extern volatile far BDT ep5Bi;   //Endpoint #5 BD In
extern volatile far BDT ep6Bo;   //Endpoint #6 BD Out
extern volatile far BDT ep6Bi;   //Endpoint #6 BD In
extern volatile far BDT ep7Bo;   //Endpoint #7 BD Out
extern volatile far BDT ep7Bi;   //Endpoint #7 BD In
extern volatile far BDT ep8Bo;   //Endpoint #8 BD Out
extern volatile far BDT ep8Bi;   //Endpoint #8 BD In
extern volatile far BDT ep9Bo;   //Endpoint #9 BD Out
extern volatile far BDT ep9Bi;   //Endpoint #9 BD In
extern volatile far BDT ep10Bo;  //Endpoint #10 BD Out
extern volatile far BDT ep10Bi;  //Endpoint #10 BD In
extern volatile far BDT ep11Bo;  //Endpoint #11 BD Out
extern volatile far BDT ep11Bi;  //Endpoint #11 BD In
extern volatile far BDT ep12Bo;  //Endpoint #12 BD Out
extern volatile far BDT ep12Bi;  //Endpoint #12 BD In
extern volatile far BDT ep13Bo;  //Endpoint #13 BD Out
extern volatile far BDT ep13Bi;  //Endpoint #13 BD In
extern volatile far BDT ep14Bo;  //Endpoint #14 BD Out
extern volatile far BDT ep14Bi;  //Endpoint #14 BD In
extern volatile far BDT ep15Bo;  //Endpoint #15 BD Out
extern volatile far BDT ep15Bi;  //Endpoint #15 BD In
```

## Wireless USB Project Design Report

```
extern volatile far CTRL_TRF_SETUP SetupPkt;
extern volatile far CTRL_TRF_DATA CtrlTrfData;

#if defined(USB_USE_MSD)
//extern volatile far USB_MSD_CBW_csw msd_cbw_csw;
extern volatile far USB_MSD_CBW msd_cbw;
extern volatile far USB_MSD_CSW msd_csw;
extern volatile far char msd_buffer[512];
#endif

#endif //USBMMAP_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                usb.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#ifndef USB_H
#define USB_H

/*
 * usb.h provides a centralize way to include all files
 * required by Microchip USB Firmware.
 *
 * The order of inclusion is important!!!
 * Dependency conflicts are resolved by the correct ordering.
 */

// #include "system\types.h" //not needed?
#include "autofiles\usbcfg.h" //either use this or io_cfg.h
#include "io_cfg.h" //includes usbcfg.h
#include "system\usb\usbdefs\usbdefs_std_dsc.h" //msd specific of usb_defs.h
#include "autofiles\usbdsc.h"

#include "system\usb\usbdefs\usbdefs_ep0_buff.h" //msd
#include "system\usb\usbmmmap.h"

#include "system\usb\usbdrv\usbdrv.h"
#include "system\usb\usbctrltrf\usbctrltrf.h"
#include "system\usb\usb9\usb9.h"

#if defined(USB_USE_MSD) // See autofiles\usbcfg.h
#include "system\usb\class\msd\msd.h"
#endif

#endif //USB_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                typedefs.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
*****/

#ifndef TYPEDEFS_H
#define TYPEDEFS_H

typedef unsigned char   byte;           // 8-bit
typedef unsigned int    word;           // 16-bit
typedef unsigned long   dword;         // 32-bit

typedef union _BYTE
{
    byte _byte;
    struct
    {
        unsigned b0:1;
        unsigned b1:1;
        unsigned b2:1;
        unsigned b3:1;
        unsigned b4:1;
        unsigned b5:1;
        unsigned b6:1;
        unsigned b7:1;
    };
} BYTE;

typedef union _WORD
{
    word _word;
    struct
    {
        {
            byte byte0;
            byte byte1;
        };
        struct
        {
            BYTE Byte0;
            BYTE Byte1;
        };
        struct
        {
            BYTE LowB;
            BYTE HighB;
        };
        struct
        {
            byte v[2];
        };
    };
} WORD;
#define LSB(a)      ((a).v[0])
#define MSB(a)      ((a).v[1])

typedef union _DWORD
{
    dword _dword;
    struct
    {
        byte byte0;
        byte byte1;
        byte byte2;
    };
};
```

## Wireless USB Project Design Report

```
        byte byte3;
    };
    struct
    {
        word word0;
        word word1;
    };
    struct
    {
        BYTE Byte0;
        BYTE Byte1;
        BYTE Byte2;
        BYTE Byte3;
    };
    struct
    {
        WORD Word0;
        WORD Word1;
    };
    struct
    {
        byte v[4];
    };
} DWORD;
#define LOWER_LSB(a)    ((a).v[0])
#define LOWER_MSB(a)    ((a).v[1])
#define UPPER_LSB(a)    ((a).v[2])
#define UPPER_MSB(a)    ((a).v[3])

typedef void(*pFunc)(void);

typedef union _POINTER
{
    struct
    {
        byte bLow;
        byte bHigh;
        //byte bUpper;
    };
    word _word;                // bLow & bHigh

    //pFunc _pFunc;            // Usage: ptr.pFunc(); Init: ptr.pFunc =
    <Function>;

    byte* bRam;                // Ram byte pointer: 2 bytes pointer pointing
                                // to 1 byte of data
    word* wRam;                // Ram word pointer: 2 bytes pointer pointing
                                // to 2 bytes of data

    rom byte* bRom;            // Size depends on compiler setting
    rom word* wRom;
    //rom near byte* nbRom;    // Near = 2 bytes pointer
    //rom near word* nwRom;
    //rom far byte* fbRom;    // Far = 3 bytes pointer
    //rom far word* fwRom;
} _POINTER;

typedef enum _BOOL { FALSE = 0, TRUE } BOOL;

#define OK        TRUE
#define FAIL      FALSE

//Might need to add card section of typedef.h code
///////////////////////////////////////////////////////////////////

typedef enum
{
    sdcValid=0,                // Everything is golden
    sdcCardInitCommFailure,    // Communication has never been established with card
    sdcCardNotInitFailure,     // Card did not go into an initialization phase
    sdcCardInitTimeout,        // Card initialization has timedout
}
```

## Wireless USB Project Design Report

```
sdcCardTypeInvalid,          // Card type was not able to be defined
sdcCardBadCmd,               // Card did not reconized the command
sdcCardTimeout,             // Card timedout during a read, write or erase sequence
sdcCardCRCError,            // A CRC error has occurred during a read, data should be
invalidated
sdcCardDataRejected,        // Card and data sent's CRC did not match
sdcEraseTimeout             // Erase took longer than it should have
}SDC_Error;

typedef union _SDCstate
{
    struct
    {
        byte isSDMMC : 1;    // set if it is a SDMMC
        byte isWP     : 1;    // set if it is write protected
    };
    byte _byte;
} SDCSTATE;

typedef union
{
    struct
    {
        DWORD _u320;
        DWORD _u321;
        DWORD _u322;
        DWORD _u323;
    };
    struct
    {
        byte _byte[16];
    };
    struct
    {
        unsigned CSD_STRUCTURE      :2;
        unsigned SPEC_VERS          :4;
        unsigned RESERVED_3         :2;

        unsigned TAAC               :8;

        unsigned NSAC               :8;

        unsigned TRAN_SPEED         :8;

        unsigned CCC_H              :8;

        unsigned READ_BL_LEN        :4;
        unsigned CCC_L              :4;

        unsigned READ_BL_PARTIAL     :1;
        unsigned WRITE_BLK_MISALIGN  :1;
        unsigned READ_BLK_MISALIGN  :1;
        unsigned DSR_IMP             :1;
        unsigned RESERVED_2         :2;
        unsigned C_SIZE_U            :2;

        unsigned C_SIZE_H            :8;

        unsigned C_SIZE_L            :2;
        unsigned VDD_R_CURR_MIN      :3;
        unsigned VDD_R_CURR_MAX      :3;

        unsigned VDD_W_CUR_MIN      :3;
        unsigned VDD_W_CURR_MAX      :3;
        unsigned C_SIZE_MULT_H       :2;

        unsigned C_SIZE_MULT_L       :1;
        unsigned SECTOR_SIZE         :5;
        unsigned ERASE_GRP_SIZE_H    :2;
```

## Wireless USB Project Design Report

```
    unsigned ERASE_GRP_SIZE_L    :3;
    unsigned WP_GRP_SIZE         :5;

    unsigned WP_GRP_ENABLE       :1; //bit 016 - 031
    unsigned DEFAULT_ECC         :2;
    unsigned R2W_FACTOR          :3;
    unsigned WRITE_BL_LEN_H      :2;

    unsigned WRITE_BL_LEN_L      :2;
    unsigned WRITE_BL_PARTIAL    :1;
    unsigned RESERVED_1          :5;

    unsigned FILE_FORMAT_GRP     :1; //bit 008 - 015
    unsigned COPY                :1;
    unsigned PERM_WRITE_PROTECT  :1;
    unsigned TMP_WRITE_PROTECT   :1;
    unsigned FILE_FORMAT         :2;
    unsigned ECC                 :2;

    unsigned CRC                 :7; //bit 000 - 007
    unsigned NOT_USED            :1;
};
} CSD;

#endif //TYPEDEFS_H
```



## Wireless USB Project Design Report

```
/*  
  
hid.h  
=====
```

```
mHIDRxIsBusy  
mHIDTxIsBusy  
mHIDGetRptRxLength  
  
void HIDInitEP(void);  
void USBCheckHIDRequest(void);  
void HIDTxReport(char *buffer, byte len);  
byte HIDRxReport(char *buffer, byte len);  
  
usb9.h  
=====
```

```
mUSBCheckAdrPendingState  
  
void USBCheckStdRequest(void);  
  
usbdrv.h  
=====
```

```
mDisableEP1to15  
mUSBBufferReady  
  
void USBCheckBusStatus(void);  
void USBDriverService(void);  
void USBRemoteWakeup(void);  
void USBSoftDetach(void);  
  
void USBModuleEnable(void)  
void USBModuleDisable(void)  
void USBSuspend(void)  
void USBWakeFromSuspend(void)  
void USB_SOF_Handler(void)  
void USBStallHandler(void)  
void USBErrorHandler(void)  
void USBProtocolResetHandler(void)  
  
usbctrltrf.h  
=====
```

```
void USBCtrlEPService(void);  
void USBCtrlTrfTxService(void);  
void USBCtrlTrfRxService(void);  
void USBCtrlEPServiceComplete(void);  
void USBPrepareForNextSetupTrf(void);  
  
void USBCtrlTrfSetupHandler(void);  
void USBCtrlTrfOutHandler(void);  
void USBCtrlTrfInHandler(void)
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                io_cfg.h
/      Authors:                 Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                 4.5.2006
/      Processor:               PIC18F4550
/      Compiler:                Microchip C18 ver. 3.02
/      Notes:
/
/*****
/*****

#ifndef IO_CFG_H
#define IO_CFG_H

/** I N C L U D E S *****/
#include "autofiles\usbcfg.h"
//#ifndef USBCFG_H
//#define USBCFG_H

/** S E T U P *****/
//Def code for Tx or other setup goes here.

/** T R I S *****/
#define INPUT_PIN                1
#define OUTPUT_PIN              0

/** U S B *****/
#define tris_usb_bus_sense      TRISAbits.TRISA1    // Input

#if defined(USE_USB_BUS_SENSE_IO)
#define usb_bus_sense           PORTAbits.RA1
#else
#define usb_bus_sense           1
#endif

#define tris_self_power         TRISAbits.TRISA2    // Input

#if defined(USE_SELF_POWER_SENSE_IO)
#define self_power              PORTAbits.RA2
#else
#define self_power              1
#endif

/** S P I : Chip Select Lines *****/
#define tris_cs_temp_sensor     TRISBbits.TRISB2    // Output
#define cs_temp_sensor          LATBbits.LATB2

#define tris_cs_sdmmc           TRISBbits.TRISB3    // Output
#define cs_sdmmc                LATBbits.LATB3

/** S D M M C *****/
#define TRIS_CARD_DETECT        TRISBbits.TRISB4    // Input
#define CARD_DETECT             PORTBbits.RB4

#define TRIS_WRITE_DETECT       TRISAbits.TRISA4    // Input
#define WRITE_DETECT            PORTAbits.RA4

/** D E V I C E C L A S S U S A G E *****/
#define USB_USE_HID

/** D E F I N I T I O N S *****/
#define EP0_BUFF_SIZE           8    // 8, 16, 32, or 64
#define MAX_NUM_INT             1    // For tracking Alternate Setting

/* Parameter definitions are defined in usbdrv.h */
//#define MODE_PP                 _PPBM0
//#define UCFG_VAL                _PUEN|_TRINT|_FS|MODE_PP

```

## Wireless USB Project Design Report

```
//#define USE_SELF_POWER_SENSE_IO
//#define USE_USB_BUS_SENSE_IO

/*
 * MUID = Microchip USB Class ID
 * Used to identify which of the USB classes owns the current
 * session of control transfer over EP0
 */
#define MUID_NULL          0
#define MUID_USB9         1
#define MUID_HID           2
#define MUID_CDC           3

/** E N D P O I N T S   A L L O C A T I O N *****
 * See usbmmmap.c for an explanation of how the endpoint allocation works
 */

/* HID */
#define HID_INTF_ID        0x00
#define HID_UEP            UEP1
#define HID_BD_OUT         ep1Bo
#define HID_INT_OUT_EP_SIZE 3
#define HID_BD_IN          ep1Bi
#define HID_INT_IN_EP_SIZE 3
#define HID_NUM_OF_DSC     1
#define HID_RPT01_SIZE     50

/* HID macros */
#define mUSBGetHIDDscAdr(ptr) \
{ \
    if(usb_active_cfg == 1) \
        ptr = (rom byte*)&cfg01.hid_i00a00; \
}

#define mUSBGetHIDRptDscAdr(ptr) \
{ \
    if(usb_active_cfg == 1) \
        ptr = (rom byte*)&hid_rpt01; \
}

#define mUSBGetHIDRptDscSize(count) \
{ \
    if(usb_active_cfg == 1) \
        count = sizeof(hid_rpt01); \
}

#define MAX_EP_NUMBER      1          // UEP1

#endif //USBCFG_H
////////// end if  USBCFG_H */

/** Hardware Pin configuration here *****/
//code here
/** end of Hardware Pin configuration *****/

#endif //IO_CFG_H
```

## Wireless USB Project Design Report

```

/*****
/*****
/
/      Filename:                main.c
/      Directory:              EnumTest4
/      Authors:                Andrew Knight
/                               Design Team 5 Wireless USB
/      Version:                4.26.2006
/      Processor:              PIC18F4550
/      Compiler:               Microchip C18 ver. 3.02
/      Notes:                  Contains main() function
/
/*****
/*****/

/** I N C L U D E S *****/
#include <stdio.h>
#include <pl18f4550.h>
#include "adapter_global_defs.h"    //Non-USB global defs for AM
#include "system\sdcard\sdcard.h"  //Not used
#include "system\types.h"           //Needed?
#include "system\usb\usb.h"         //Required!!!
#include "io_cfg.h"
#include "system\usb\usb_compile_time_validation.h" //Optional! - I did not
include!

/** PIC Configuration Settings *****/
#pragma config PLLDIV = 5           /* PLL Prescaler */
#pragma config CPUDIV = OSC4_PLL6   /* CPU Clock Prescaler */
#pragma config USBDIV = 2           /* USB Clock Selection */
#pragma config FOSC = ECPLL_EC      /* Oscillator Selection */
#pragma config FCMEM = OFF          /* Fail Safe Clock Monitor */
#pragma config IESO = OFF           /* Oscillator Switchover */
#pragma config PWRT = OFF           /* Power Up Timer */
#pragma config BOR = OFF            /* Brown Out Reset */
#pragma config BORV = 1             /* Brown Out Voltage */
#pragma config VREGEN = ON          /* USB Voltage Regulator */
#pragma config WDT = OFF            /* Watchdog Timer */
#pragma config WDTPS = 1            /* Watchdog Timer Postscaler */
#pragma config MCLRE = OFF          /* MCLR */
#pragma config LPT1OSC = OFF        /* Low Power Timer 1 */
#pragma config PBADEN = OFF         /* Port B A/D */
#pragma config CCP2MX = OFF         /* CCP2 MUX */
#pragma config STVREN = OFF         /* Stack Full/Underflow Reset */
#pragma config LVP = OFF            /* Single Supply ICSP */
#pragma config ICPT = OFF           /* In-Circuit Debug/Programming */
#pragma config XINST = OFF          /* Extended Instruction Set */
#pragma config DEBUG = OFF          /* Debugger */
#pragma config CP0 = OFF            /* Code Protection Block 0 */
#pragma config CP1 = OFF            /* Code Protection Block 1 */
#pragma config CP2 = OFF            /* Code Protection Block 2 */
#pragma config CPB = OFF            /* Boot Block Protection */
#pragma config CPD = OFF            /* EEPROM Protection */

/** V A R I A B L E S *****/
#pragma udata

/** TESTING VARS AND FUNCTIONS *****/
int count = 1;
    //testing only

/** P R I V A T E   P R O T O T Y P E S *****/
static void initAMPIC(void);
static void RunSystem(void);
void USBTasks(void);
void InitializeUSBDriver(void);

/** D E C L A R A T I O N S *****/
#pragma code
/*****
 * Function:        void main(void)
*****/
```

## Wireless USB Project Design Report

```
* Overview:          Main program entry point.
*****/
void main(void)
{
    msDelay(10);                //Startup Delay

    initAMPIC();                //See function below
    while(1)
    {
        USBTasks();            // USB Tasks
        ProcessIO();           // In msd.c and msd.h
        //RunSystem();         // Perform other ops needed
    } //end while
} //end main

/*****
* Function:          void InitializeUSBDriver(void)
* Overview:          Configures the USB module, definition of UCFG_VAL can be
*                   found in autotfiles\usbcfg.h
*
*                   This register determines: USB Speed, On-chip pull-up
*                   resistor selection, On-chip tranceiver selection, bus
*                   eye pattern generation mode, Ping-pong buffering mode
*                   selection.
*
* Note:              None
*****/
void InitializeUSBDriver(void)
{
    UCFG = UCFG_VAL;
    usb_device_state = DETACHED_STATE;
    usb_stat._byte = 0x00;
    usb_active_cfg = 0x00;
}

/*****
* Function:          void initAMPIC(void)
* Overview:          Initializes all aspects of the Adapter Module PIC.
*                   It is a centralize initialization routine.
*                   All required USB initialization routines are called from
*                   here. User application initialization routine should also be
*                   called from here.
*****/
static void initAMPIC(void)
{
    //Init AM here
    ADCON1 |= 0x0F;                //default all pins to digital
    ADCON0 = 0x07;                //default all ports as digital I/O
    TRISD = 0;                    //default PORTD as output only
    TRISB = 0;                    //default PORTB as output only

    //All USB Inits here
    InitializeUSBDriver();         // See usbdrv.h

    //Init Pic for SPI mode then power up the transceiver and Configure it.
    initTrans();
}

/*****
* Function:          static void RunSystem(void)
* Overview:
*****/
static void RunSystem(void)
{
    // Currently does nothing
    //msDelay(10);
    //transmitData(0B10101010);

    /*
```

## Wireless USB Project Design Report

```
        if(count == 1){
            transmitData(0B10101010);
            count = 0;
        }
        else{
            transmitData(0B01010101);
            count = 1;
        }
    }
    */

    //msDelay(30000);
    //if(count == 1){
    //    transmitData(0B10101010);
    //    count = 0;
    //}
}

/*****
 * Function:      void USBTasks(void)
 * PreCondition:  InitializeSystem has been called.
 * Overview:      Service loop for USB tasks.
 *****/
void USBTasks(void)
{
    /*Servicing Hardware*/
    USBCheckBusStatus();           // usbdrv.c           // Must use polling method
    if(UCFGbits.UTEYE!=1)         // Interrupt or polling
method
    USBDriverService();           // usbdrv.c
} // end USBTasks

/** EOF main.c *****/
```