PICDEM™ FS USB DEMONSTRATION BOARD USER'S GUIDE

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Table of Contents

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the Chapter Name. Items discussed in this chapter include:

- About This Guide
- Warranty Registration
- Recommended Reading
- Troubleshooting
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support

ABOUT THIS GUIDE

Document Layout

This document describes how to use the PICDEM FS USB demonstration board as a development tool for creating full-speed USB applications. The manual layout is as follows:

- **Chapter 1: Introduction to the PICDEM™ FS USB Board** describes the hardware of the demonstration board , and how it can be used in creating new USB solutions.
- **Chapter 2: Getting Started with the PICDEM™ FS USB Board** describes how to connect the demonstration board to a host system, and how to install the Demonstration Tool software.
- **Chapter 3: Using the Demo Tool Software** describes how to use the application in both Demo and Bootload modes.
- **Chapter 4: Using the Microchip USB Firmware Framework** provides an overview of the framework, and how to use it in the design of new USB solutions.
- **Chapter 5: Reconfiguring the PICDEM™ FS USB Hardware** describes how to tailor the demonstration board 's hardware to your application.
- **Chapter 6: Troubleshooting** discusses some common questions about using the demonstration board.
- **Appendix A: PICDEM™ FS USB Board Technical Information** provides the schematics and other technical details about the demonstration board.
- **Appendix B: PICDEM™ FS USB Starter Kit CD** describes the complete contents of the accompanying software CD.

Conventions Used in this Guide

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
'b*nnnn*	A binary number where *n* is a digit	'b00100, 'b10
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier font:		
Plain Courier	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
Italic Courier	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
`0x`*`nnnn`*	A hexadecimal number where *`n`* is a hexadecimal digit	`0xFFFF`, `0x007A`
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in the Warranty Registration Card entitles users to receive new product updates. Interim software releases are available at the Microchip web site.

RECOMMENDED READING

This user's guide describes how to use the PICDEM™ FS USB User's Guide. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

Readme for PICDEM™ FS USB User's Guide

For the latest information on using Chapter Name, read the "`Readme for Chapter Name.txt`" file (an ASCII text file) in the Readmes subdirectory of the MPLAB IDE installation directory. The Readme file contains update information and known issues that may not be included in this user's guide.

Readme Files

For the latest information on using other tools, read the tool-specific Readme files in the Readmes subdirectory of the MPLAB IDE installation directory. The Readme files contain update information and known issues that may not be included in this user's guide.

PIC18F2445/2550/4445/4550 Device Data Sheet (DS39632)

This is the comprehensive reference for Microchip's enhanced microcontroller with Full-Speed USB. For users already familiar with the USB protocol, the data sheet provides the basic information needed for designing the hardware and firmware for a Microchip-based USB solution.

USB Specification, Revision 2.0 (USB Implementers Forum, Inc., www.usb.org)

For developers creating a USB application from the ground up, this is the comprehensive reference on the Universal Serial Bus protocol. All features of USB, from physical and electrical specifications to data and communication protocols to device management are defined here.

TROUBLESHOOTING

See **Chapter 6. "Troubleshooting"** for information on common problems.

THE MICROCHIP WEB SITE

Microchip provides online support via our WWW site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers and other language tools. These include the MPLAB C17, MPLAB C18 and MPLAB C30 C compilers; MPASM™ and MPLAB ASM30 assemblers; MPLINK™ and MPLAB LINK30 object linkers; and MPLIB™ and MPLAB LIB30 object librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB ICE 2000 and MPLAB ICE 4000.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM and MPLAB SIM30 simulators, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus development programmer.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support
- Development Systems Information Line

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://support.microchip.com

In addition, there is a Development Systems Information Line which lists the latest versions of Microchip's development systems software products. This line also provides information on how customers can receive currently available upgrade kits.

The Development Systems Information Line numbers are:

1-800-755-2345 – United States and most of Canada

1-480-792-7302 – Other International Locations

Chapter 1. Introduction to the PICDEM™ FS USB Board

1.1 INTRODUCTION

The PICDEM FS USB Demonstration Kit is designed as an easy-to-use evaluation platform for Microchip's Full Speed USB solution. The devices in Microchip's new family of full-speed USB microcontrollers fully support USB 2.0 and serial communications of up to 12 Mbit/s.The Demonstration Kit provides all of the hardware and software needed to demonstrate and develop a complete USB communication solution. Items discussed in this chapter include:

- PICDEM FS USB Demonstration Kit Contents
- Overview of the PICDEM FS USB Board
- PICDEM FS USB Board Features
- Demonstration Tool Software

1.2 PICDEM FS USB DEMONSTRATION KIT CONTENTS

The Demonstration Kit contains the following items:

1. The PICDEM FS USB demonstration board , pre-programmed with USB bootloader and demonstration firmware.
2. A standard USB cable for use in communicating with the board.
3. The PICDEM FS USB Starter Kit CD-ROM, containing the USB driver, Demo Tool application and development tools.

1.3 OVERVIEW OF THE PICDEM FS USB DEMONSTRATION BOARD

The microcontroller for the PICDEM FS USB board is the PIC18F4550, the principal device of the PIC18F2455/2550/4455/4550 family. All of these devices implement power-saving nanoWatt Technology, enhanced Flash program memory and feature USB modules with the following:

- USB 2.0 compliance
- Full-speed (12Mbit/s) and low-speed (1.5Mbit/s) operation
- Support of control, interrupt, bulk and isochronous transfers
- Support of up to 32 endpoints
- 1 Kbyte of dual-access RAM for USB
- On-chip features for a true single-chip USB implementation, including
  - USB transceiver
  - USB voltage regulator
  - USB pull-up resistors
- Interface for off-chip USB transceiver

The included reference designs and examples use the Microchip USB Firmware Framework. This provides the services for handling lower-level USB protocol management, control transfer, USB interrupt handling, hardware register control and management. Reference designs included with the demonstration kit cover Human Interface Devices (HID), the Communication Device Class (CDC) and the Microchip General Purpose USB device class.

The software consists of the Microchip General Purpose USB Windows driver, `mchpusb.sys`, which allows a PC application to communicate directly to the device's endpoints. A set of user-friendly programming interfaces, the MPUSBAPI Library, is also provided to allow easy application development.

The board comes pre-programmed with a default demo application and is ready for evaluation right out of the box. New programs can be downloaded to the PIC18F4550 microcontroller using a pre-programmed bootloader via the USB interface. A PC-based application, the PICDEM FS USB Demo Tool, facilitates the bootloading process and serves as the host application in the default demonstration.

1.3.1 Benefits of Using the PICDEM FS USB Board

The PICDEM FS USB demonstration board provides two simple advantages:

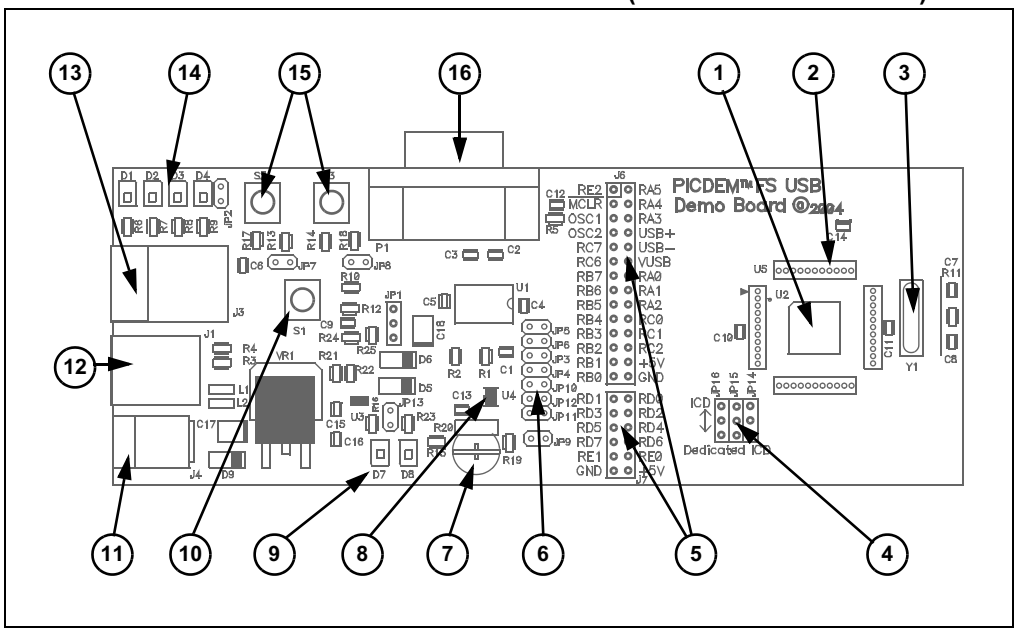
A READY-TO-USE DEMONSTRATION: As delivered, the board is ready for full-speed USB operation; all that is required is the proper driver (included) and the Demo Tool application. Users can also re-program the board with different applications, included on the Starter Kit CD, to evaluate other USB solutions that can be tailored to their needs.

AN EXPANDABLE PLATFORM: Users can also expand the hardware capabilities of the board through its expansion headers, and even interface it with available PICtail™ demonstration and evaluation daughter boards.

1.4 PICDEM FS USB BOARD HARDWARE FEATURES

The overall layout of the board is shown in Figure 1-1, with a list of the main features following. The more complex features are discussed in detail later in this section.

FIGURE 1-1: THE PICDEM FS USB BOARD (TOP ASSEMBLY VIEW)



1. **Microcontroller:** The 44-pin, TQFP PIC18F4550 microcontroller (U2) is the heart of the demonstration board, and provides all USB functionality on one chip. Refer to the device data sheet (DS39632) for a complete discussion of the microcontroller and its feature set.
2. **ICE Interface Riser:** The microcontroller is surrounded by a 44-pin pad (U5), arranged as four groups of 11 on each side. These locations can be used to mount a riser for interfacing with Microchip's MPLAB ICE 2000/4000 emulator system.

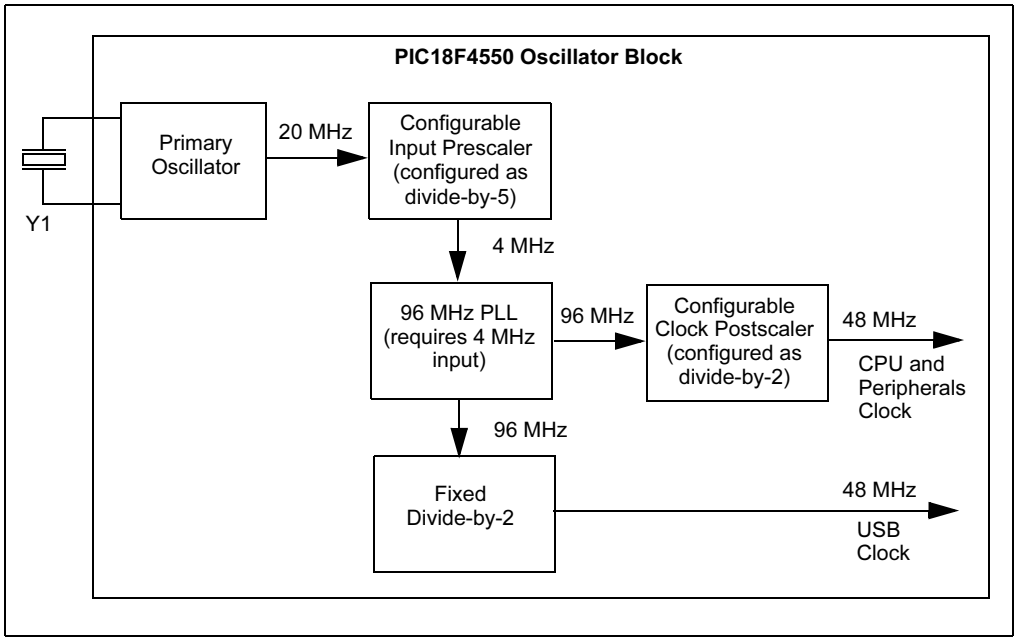
3. **Oscillator:** The demonstration board uses a 20 MHz crystal oscillator (Y1) as the primary clock service. The PIC18F4550 uses this oscillator to generate the necessary clock signals for both the USB serial interface engine (SIE) and the core processor.
4. **ICD Configuration Jumpers:** These three unpopulated jumper positions allow the user to choose either legacy or dedicated ICSP™ and ICD ports for the controller. By default, the board is hard-configured for the legacy port. The configuration and use of these jumpers is detailed in **Section 5.2.4 "Adding In-Circuit Emulation Capability"**.
5. **Expansion and PICtail Headers:** The pads at J6 and J7 are provided for users to install header and directly access the microcontroller's I/O port signals. In addition, the 14 even-numbered pins of J6 (those on the right side as viewed from the top) serve as the interface for Microchip's PICtail daughter boards. This allows the PICDEM FS USB board to be used as a test platform and USB communications interface for the PICtail boards.
6. **Configuration Jumpers:** A total of 13 unpopulated jumper positions are provided across the board; these allow users to modify the board by configuring its hardware to suit their needs. By default, all jumpers are bridged and all features are enabled. The configuration and use of these jumpers is detailed in **Section 5.2 "Configuring the Demonstration Board Options"**.
7. **Potentiometer:** The potentiometer (R20) simulates an analog input for the controller. Its real-time value can also displayed by the Demo Tool host software.
8. **Temperature Sensor:** A Microchip TC77 digital temperature sensor (U4) continuously monitors the board's ambient temperature. Data is transmitted to the controller via a 3-wire SPI interface, and is displayed in real time by the Demo Tool software.
9. **Power LEDs (Green):** These light to show that power is being supplied to the board, and to indicate how the board is being powered. LED D7 indicates that the board is being powered from the bus, while D8 indicates the board is being powered from a separate power supply.
10. **Reset Push Button:** This switch (S1) is tied to the $\overline{\text{MCLR}}$ pin of the PIC18F4550 controller; pressing it causes a hard device reset.
11. **Power Connector:** Power (9 VDC) can be supplied to the board from an external power adapter through a mini barrel jack. Using an external supply is optional, as all examples provided with the demonstration board can use power from the USB cable.
12. **USB Connector:** This is a standard USB series "B" receptacle. The USB port is the primary channel for controlling and communicating with the demonstration board.
13. **ICD Connector:** This 6-wire RJ11 connector provides the standard interface used by Microchip development and demonstration boards for programming and debugging applications, using MPLAB ICD 2 and other development tools.
14. **Status LED Bank:** A bank of four green LEDs is used to show the operational status of the board. Two LEDs (D1 and D2) are used by the application firmware to indicate the status of the USB connection. The other LEDs (D3 and D4) can be defined by the user; they are directly controllable through the Demo Tool software.
15. **User-defined Push Buttons:** These two switches (S2 and S3) are provided to simulate digital control inputs. Pressing either button causes its port to read as '0'.
16. **RS-232 (DB9F) Port:** A standard D-shell connector, along with a standard level shifter (U1), provides an RS-232 serial connection to the demonstration board.

1.4.1 Oscillator and Operating Frequency

The USB module of the PIC18F4550 requires a specific clock frequency input to operate correctly; specifically, 48 MHz is required when operating in full-speed mode, or 6 MHz when operating in low-speed mode. The PICDEM FS USB board uses a 20 MHz crystal oscillator as an external clock, and derives the necessary internal clock frequencies from the 96 MHz PLL module. The clock configuration used on the demonstration board is shown in Figure 1-2. A detailed explanation of how to setup the clock configuration is explained in Chapter 2 of the PIC18F4550 device data sheet (DS39632).

The PICDEM FS USB board is configured to run in Full Speed USB mode, generating an internal system clock of 48 MHz (equivalent to 12 MIPS) from the external 20 MHz crystal.

FIGURE 1-2: PIC18F4550 CLOCK SETUP FOR PICDEM FS USB



1.4.2 Power

The PICDEM FS USB demonstration board operates at 5V. Power can be drawn directly from the USB bus or from an external power supply. There are no jumpers to select which power source to use. Instead, the power circuitry automatically selects the external power source when both power sources are available. Two LEDs, D7 and D8, are used to indicate the active power source. D7 indicates that the board is operating in Bus Power mode; D8 indicates Self Power mode (i.e., an external power supply).

Note:	There are no power conditions that will cause both D7 and D8 to light at the same time.

Like most USB peripherals, the PICDEM FS USB board can be powered from the 5V available from the USB cable. A minimum of 100 mA is always available on the bus for a device; a maximum of 500 mA can be requested and used if available.

A barrel-type power supply connector (2.5 mm diameter) is also provided to run the board from a 9-18 VDC power supply. A transformer is not supplied in the kit because every example included will run from USB bus power.

A USB host may send a query to a USB device to determine if it is currently self-powered. Without an ability to sense the output of the 5V regulator (VR1), there would be no way to determine the status of the power supply. The PICDEM FS USB board uses two of the microcontroller's I/O pins (PORTA<2:1>) to sense which supply is available: PORTA<2> monitors the regulator, while PORTA<1> senses the USB cable. When a port is read as '1', its corresponding power source is active. When a port is read as '0', its source is disconnected. The combinations of PORTA states are shown in Table 1-1.

For more details on power requirement and management, refer to Microchip application note AN950, "Power Management for PIC18 USB Microcontrollers with nanoWatt Technology" (DS00950).

TABLE 1-1: PORTA<2:1> STATE COMBINATIONS AND THEIR MEANINGS

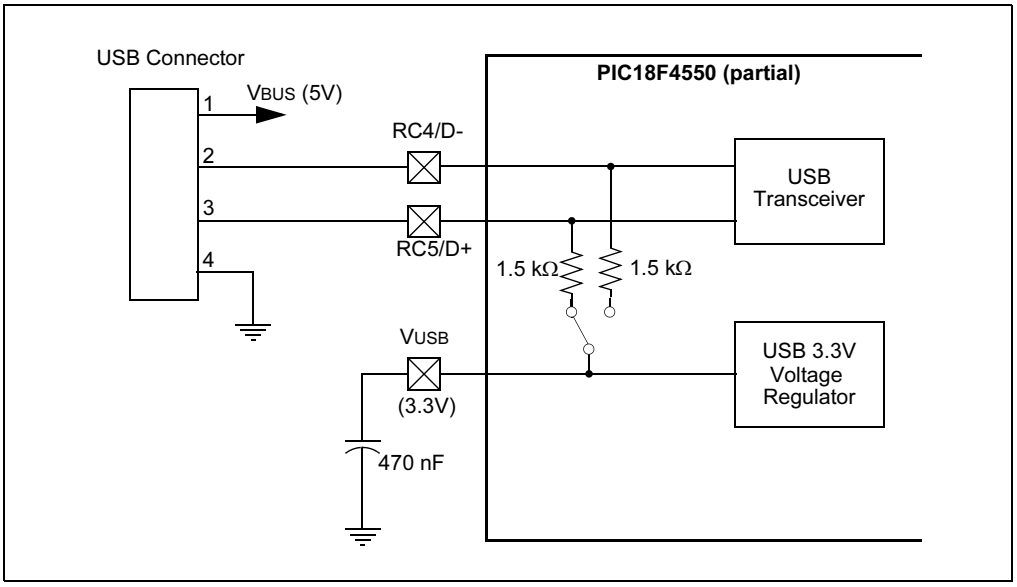
PORTA<1>	PORTA<2>	Status
1	1	USB cable and power supply are both connected; board is self-powered, D8 is lit
1	0	USB cable is attached; board is bus-powered, D7 is lit
0	1	Power supply only is attached; board is self-powered, D8 is lit
0	0	N/A

1.4.3 USB Interface

The PICDEM FS USB board utilizes the on-chip USB voltage regulator, transceivers and pull-up resistors of the PIC18F4550. This helps reduce the number of external components. The USB connection can be electrically detached by disabling the USB module in the firmware. By disabling the USB module in firmware (setting the USBEN bit in the UCON register to '0'), the on-chip USB voltage regulator will also be disabled. This simulates the physical detachment of the USB cable.

Figure 1-3 shows the electrical connection for the USB interface on the board.

FIGURE 1-3: USB INTERFACE, SHOWING ON-CHIP COMPONENTS



1.4.4 RS-232 Interface

The PICDEM FS USB board fully supports RS-232 serial communications, including hardware flow control (RTS/CTS signalling) generated by pins RA2 and RA3 of the microcontroller. One RS-232 female connector and all supporting circuitry are included.

1.4.5 Status LEDs

There are four firmware-controllable LEDs, D1 through D4. An LED is turned on when the corresponding port bit has the value of '1', and off when the port bit is '0'.

D1 and D2 are used to indicate USB device status. Table 1-2 summarizes the LED values in relation to different USB device state. The meaning of each of the USB device state can be found in Chapter 9 of the USB Specification.

Note: In bus-powered applications, flashing D1 and D2 in Suspended mode is not recommended; the current draw is likely to exceed the limits specified in the USB Specification. It is implemented in the original firmware for the demonstration board for demonstration purposes only.

TABLE 1-2: USB DEVICE STATE LED STATUS

USB Device State	LED D1	LED D2
Detached	Off	Off
Attached	On	On
Powered	On	Off
Default	Off	On
Addressed	Blink	Off
Configured	Alternate Flashing	
Suspended	Fast Synchronous Flashing	

1.5 DEMO TOOL APPLICATION SOFTWARE

Included with the demonstration board is the USB Demo Tool software. This simple graphic interface allows users to monitor and control simple board features, and provides the ability to reprogram the PIC18F4550 controller via a bootloader demonstration.

The overall operation of the host software is discussed in **Chapter 3. "Using the Demo Tool Application"**.

Chapter 2. Getting Started with the PICDEM™ FS USB Board

2.1 HIGHLIGHTS

This chapter will cover the following topics:

- Host Computer Requirements
- Installing the Demonstration Board

2.2 HOST COMPUTER REQUIREMENTS

To communicate with and program the PICDEM FS USB demonstration board, the following hardware and software requirements must be met:

- PC-compatible system
- An available USB port
- CD-ROM drive (for use with the accompanying CD)
- Microsoft Windows 98, Second Edition (98SE), Windows 2000 Desktop or Windows XP

2.3 INSTALLING THE DEMONSTRATION BOARD

As a USB device, the demonstration board requires very little effort to install; most of the work is done by the operating system. The three steps required are:

1. Installing the PICDEM FS USB software package
2. Connecting the PICDEM FS USB board
3. Installing the USB driver

2.3.1 Installing the PICDEM FS USB Software Package

The software package that accompanies the PICDEM FS USB board contains all the software required to start using the board immediately. It includes the Demo Tool application, the USB drivers, the reference code projects and all documentation. Installation is completely automated, and does not require any user intervention or configuration once the process is started; however, USB driver installation will require connection of the board to complete. The process is identical for all 32-bit Windows operating systems. Users with Windows NT® -based desktops (2000 and XP) do not need to have administrative rights to their systems for this installation, as long as they have at least Power User rights.

Closing all background applications before proceeding is helpful, but not required. If possible, temporarily disable any anti-virus software that the host system is running before installing the package, and re-enable it after the installation is complete. This is also not absolutely necessary, but may be helpful in some system configurations.

Note: It is possible that some organizations may implement a desktop computer policy sufficiently restrictive to prevent the user from loading any software at all. In theory, this can be done with **any** 32-bit Windows operating system on a network. If this describes your situation, contact your local Information Services provider for assistance in installing this software.

To install the software package:

1. Double-click on "My Computer", then on the icon for the CD-ROM.
2. Double-click on the "MCHPFSUSB_Setup.exe" icon. Installation will proceed automatically, and take 1-2 minutes.

Alternatively, run the installation by selecting *Run* from the Start menu. At the dialog box, enter:

```
x:\MCHPFSUSB_Setup.exe
```

where "x" is the drive letter of your CD-ROM.

The installation process will install the software package. By default, all files are installed by default in the directory C:\MCHPFSUSB. A short cut for the software is also installed under **Programs** from the Start menu (*Programs > Microchip > PICDEM FS USB*).

Note: Using the default installation directory (C:\MCHPFSUSB\) allows the reference projects to retain their original project paths, and requires no additional configuration during setup. If a different installation directory is used, refer to **Section 4.2.3 "Configuring MPLAB for the USB Framework"** on how to update the firmware project paths in the development environment.

2.3.2 Connecting the PICDEM FS USB Board

To connect the demonstration board :

1. Unbox and unwrap the board, and set it on a non-conductive surface near the host system.
2. Connect the USB cable (supplied in the kit) to an open USB port on the host system or a USB hub connected to the host system, and to the USB connector on the board.
3. Check the board. The green Power LED D7 should be lit, and status LEDs D1 and D2 should be flashing rapidly and in unison. If not, check the connections at the USB port or hub and the board. For additional assistance, refer to **Chapter 6. "Troubleshooting"**.

2.3.3 Installing the USB Device Driver

Because of the differences in USB implementation for different versions of Microsoft Windows, the installation of the driver varies with the operating system being used. In general, there are two different procedures: one for Windows 98SE, and one for Windows 2000 Professional Desktop and Windows XP Desktop. Regardless of the operating system, it should not be necessary to close any open applications prior to the driver installation.

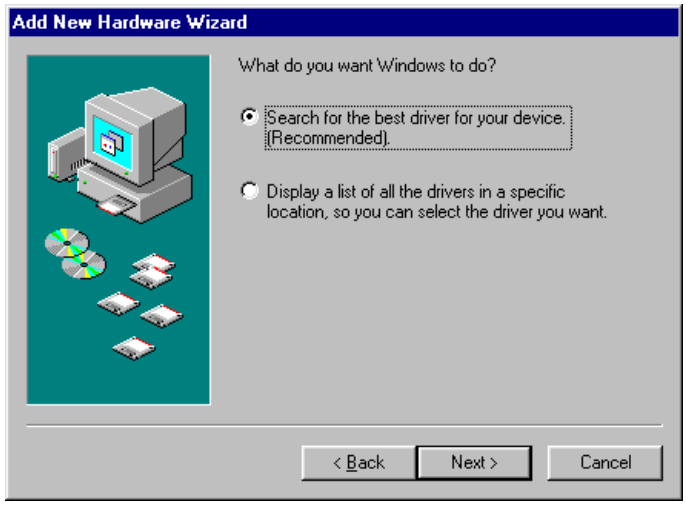
Note 1: The PICDEM FS USB demonstration board is configured in firmware as two devices: a Demo application and a Bootload application. The USB driver will install when each application is launched the first time.

2: It is possible that some organizations may implement a desktop computer policy sufficiently restrictive to prevent the user from adding any Plug and Play hardware, including USB devices. This can be done with **any** 32-bit Windows operating system on a network, including Windows 98SE. If this describes your situation, contact your local Information Services provider for assistance in installing this software.

For Microsoft Windows 98SE:

1. When the USB cable is connected, the “Add New Hardware Wizard” dialog will appear. Click **Next**.
2. At the following dialog (Figure 2-1), you will be asked to choose a driver search method. Select the “Search for the Best Driver” option, and click **Next**.

FIGURE 2-1: HARDWARE WIZARD, DRIVER SEARCH OPTIONS



3. At the next dialog (Figure 2-2), select the “Specify a location” option. Click on **Browse** to open the Windows Open File dialog. Navigate to the directory `C:\MCHPFSUSB\PC\MCHPUSB Driver\Release` and click **Open**.

FIGURE 2-2: HARDWARE WIZARD, SELECT DRIVER SOURCE

4. At the next dialog (Figure 2-3), verify that the path displayed is correct. Click **Next**.mchpusb.inf

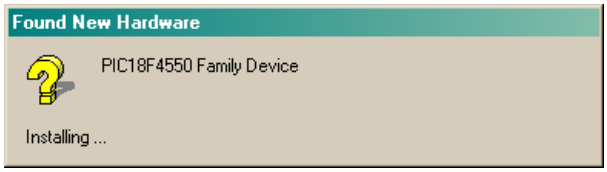
FIGURE 2-3: HARDWARE WIZARD, READY TO INSTALL DRIVER

5. After a brief interval, the final Add Hardware Wizard dialog will confirm that the driver has been installed. Click **Finish**.
6. To complete the installation, allow the system to restart at the next dialog. If other applications are running, you may click **No** at the dialog, then close the other applications before restarting the system.

For Microsoft Windows 2000 Desktop:

1. When the USB cable is connected, a "Found New Hardware" message appears. After a brief interval, this will be replaced by the "Found New Hardware Wizard" dialog. Click **Next**.

FIGURE 2-4: NEW HARDWARE MESSAGE



2. At the “Install Hardware Device Drivers” dialog, select the “Search for a suitable driver” option. Click **Next**.

FIGURE 2-5: HARDWARE WIZARD, DRIVER SEARCH OPTIONS

3. At the “Locate Driver Files” dialog, select the “Specify a location” check box. Click **Next**.
4. At the next dialog, click the **Browse** button to open the Windows Locate File dialog. Navigate to the directory `C:\MCHPFSUSB\PC\MCHPUSB Driver\Release`. Click **Open** to accept the location, then **OK** at the original dialog (see Figure 2-10).

FIGURE 2-6: HARDWARE WIZARD, SPECIFY DRIVER LOCATION

5. At the “Driver Installation” dialog, confirm that the wizard has located the file `mchpusb.inf`. Click **Next**.
6. After a brief interval, the final Hardware Wizard dialog will confirm that the driver has been installed. Click **Finish**.

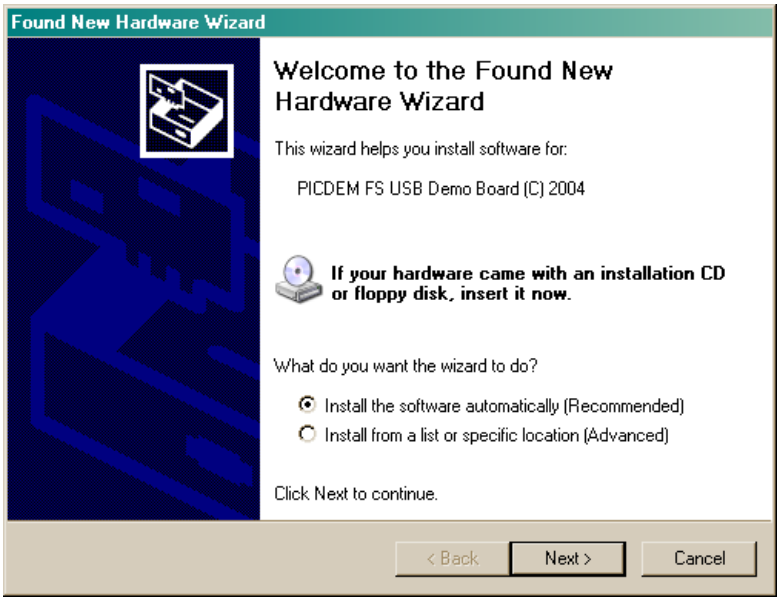
It will not be necessary to restart the system.

For Microsoft Windows XP:

The USB driver installation process is similar to that of Windows 2000, but requires fewer steps.

1. After the USB cable is connected, the initial Hardware Wizard dialog appears (Figure 2-7). Select the "Install from a list of specific locations" option. Click **Next**.

FIGURE 2-7: HARDWARE WIZARD, DRIVER SEARCH OPTIONS



2. At the "Search and Installation Options" dialog, select the "Include this location..." option. Click the **Browse** button to open the Windows Browse for Folder dialog. Navigate to the directory `C:\MCHPFSUSB\PC\MCHPUSB Driver\Release`. Click **OK** to accept the location, then **Next** at the original dialog (see Figure 2-8).

FIGURE 2-8: HARDWARE WIZARD, SPECIFY DRIVER LOCATION

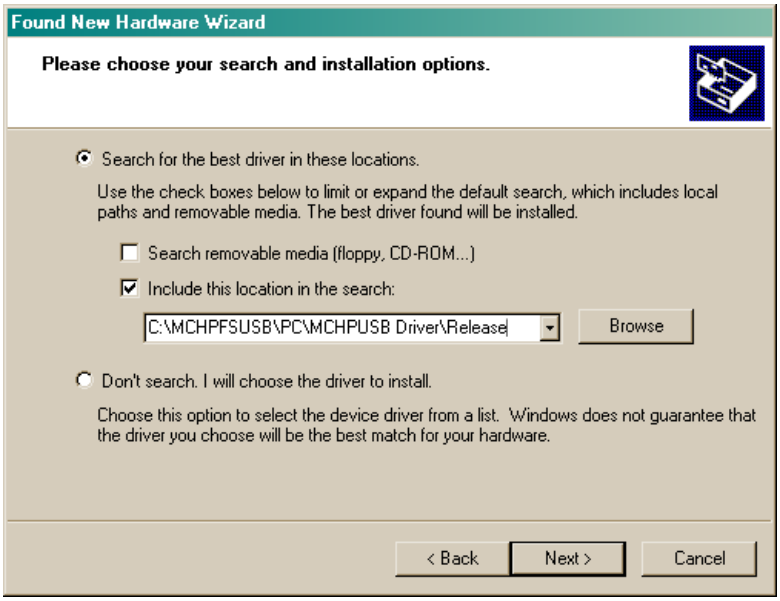
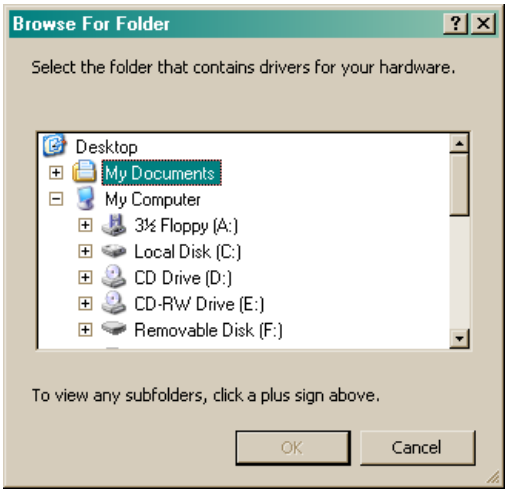


FIGURE 2-9: HARDWARE WIZARD, BROWSE FOR FOLDER DIALOG

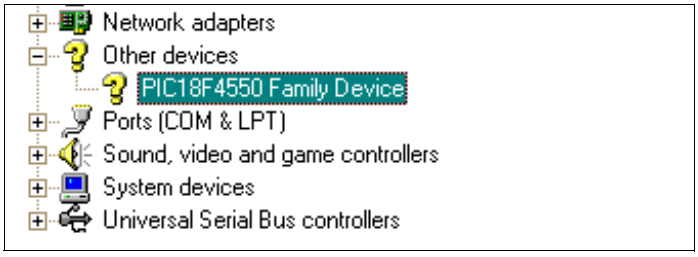


3. The system will copy and install the driver files, and indicate when the process is finished. Click **Next**.
4. The final dialog will indicate that the driver has been successfully installed. Click **Finish**.

It will not be necessary to restart the system.

Regardless of the versions of the operating system, the PICDEM FS USB demonstration board will appear in the Device Manager view (available from the System applet in the Control Panel) when the installation is complete. The PICDEM FS USB board will appear under the category of “Other Devices” (Figure 2-10). The icon and name will be the same in all versions of Windows.

FIGURE 2-10: PICDEM FS USB BOARD IN DEVICE MANAGER VIEW (PARTIAL WINDOW SHOWN)



2.3.4 Confirming Operation

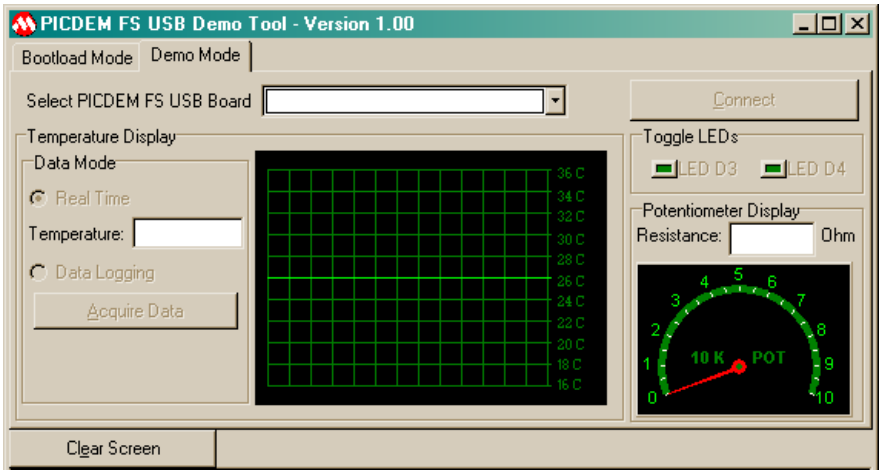
If Power LED D7 is lit (or D8, if an external power supply is being used) and D1 and D2 are flashing, it can generally be assumed that the demonstration board 's hardware is working correctly. However, it may be useful to verify that the board can actually communicate with the host system using the Demo Tool application.

To do this:

1. Launch the Demo Tool application by clicking on the "PICDEM FS USB Demo Tool" icon.
2. Select the **Demo Mode** tab to access the Demo Mode controls (Figure 2-11).
3. In the "Select PICDEM FS USB Board" dropdown list, select "PICDEM FS USB 0 (Demo)".
4. Click the **Connect** button. A temperature number should appear in the "Real Time Temperature" window almost immediately.

If an appropriate temperature appears, the demonstration board is ready for your use.

FIGURE 2-11: THE DEMO MODE WINDOW (PARTIAL VIEW)



Chapter 3. Using the Demo Tool Application

3.1 HIGHLIGHTS

The items discussed in this chapter are:

- Software Overview
- Starting the Program
- Demo mode
- Bootload mode

3.2 SOFTWARE OVERVIEW

The PICDEM FS USB Demo Tool (more simply, PDFSUSB or the "Demo Tool") is a Windows-based software application designed to be used with the PICDEM FS USB board for evaluating Microchip's Full-Speed USB solutions. Using this software, you can evaluate USB features and performance offered by the PIC18F4550 microcontroller.

There are two modes available: Demo mode and Bootload mode. Demo mode demonstrates communication to and from the host PC through USB in a typical embedded system setup. The application shown includes data logging, real-time sensing, and how to use PC to control peripheral components. Bootload Mode allows designers to download and evaluate a different firmware program without using an additional external programmer.

The Demo and Bootloader are two separate applications. Both programs are totally separate and do not share any USB functions or descriptors. The Bootloader code is write-protected and cannot be overwritten by firmware. An external programmer such as MPLAB ICD 2 is required to erase the Bootloader. The two programs also have different USB product IDs (but the same Vendor ID). For reference, the product IDs are:

- Bootload: VID 0x04D8, PID 0x000B
- Demo: VID 0x04D8, PID 0x000C

Note: Because the Demo and Bootloader are two different applications, they are treated as two devices. The Microchip USB driver will attempt to install when each of the applications is launched for the first time. In the procedure described for the previous chapter, the USB driver for the Demo application was installed. The driver for the Bootload application will begin installation on the its first launch.

3.3 STARTING THE PROGRAM

The demo board is shipped from the factory with the default demonstration firmware program and the bootloader. In the case that the demonstration program has been overwritten, you can restore the firmware by using the bootloader or an external programmer such as MPLAB ICD 2.

To run the Demo Tool application, select *Programs > Microchip > PICDEM FS USB > PICDEM FS USB Demo Tool* from the Start menu. Alternatively, double-click on the PDFSUSB icon or shortcut. You will see the Demo Tool window (Figure 3-1). By default, the application launches in Bootload mode. To switch to Demo mode, click on the "Demo Mode" tab (Figure 3-2).

FIGURE 3-1: THE DEMO TOOL WINDOW, BOOTLOAD MODE

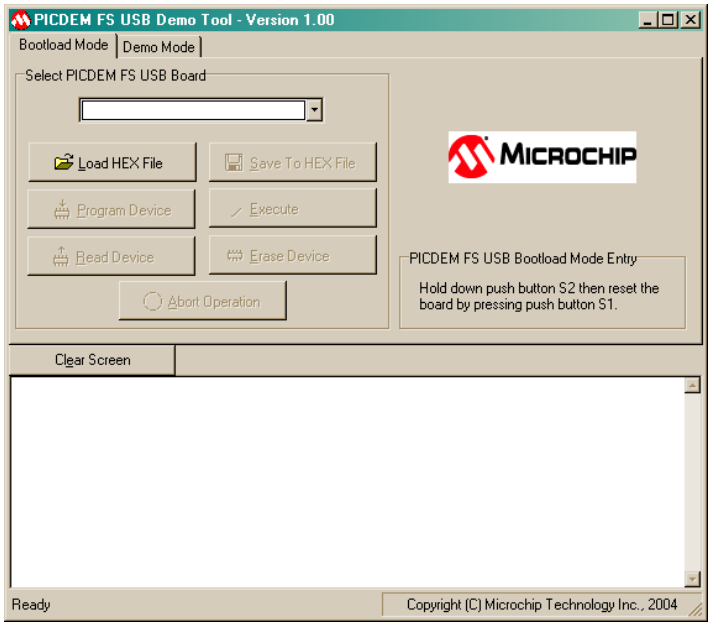
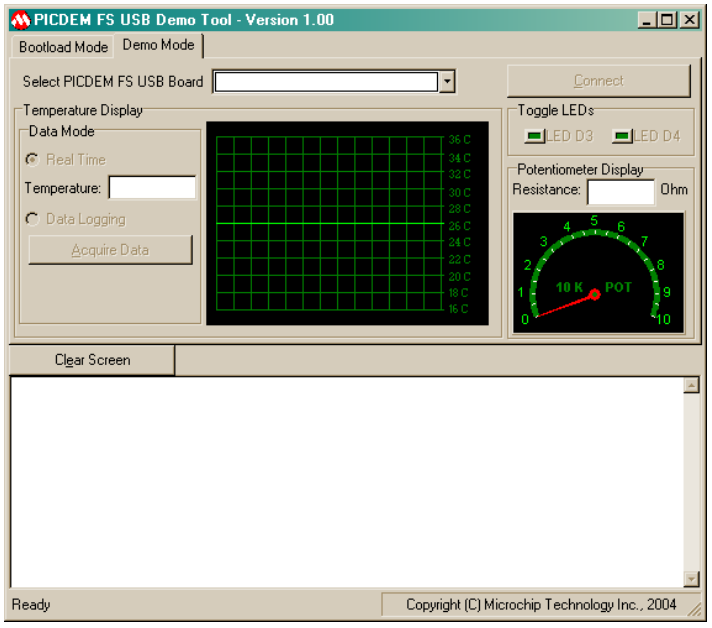


FIGURE 3-2: THE DEMO TOOL WINDOW, DEMO MODE



3.4 DEMO MODE

The Demo mode provides a simple interface between the board and the host system to demonstrate USB connectivity. To start using the application, select “PICDEM FS USB 0 (Demo)” from the “Select PICDEM FS USB Board” dropdown box. Click the “Connect” button in the upper right hand corner. When the Demo Tool application successfully connects to the board, the message “USB Demo Firmware Version x.x” will appear on the left side of the status bar at the bottom of the window. There are three interactive features in this mode.

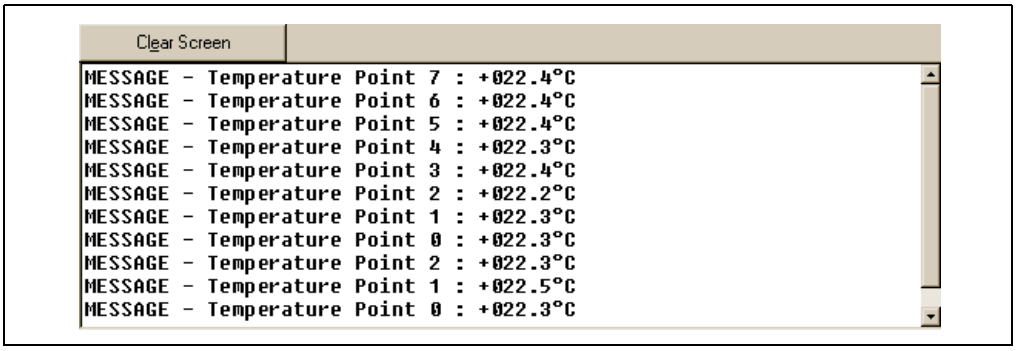
Temperature Display

There are two temperature modes: Real Time and Data Logging. In Real Time mode, temperature data is streamed from the board to the PC continuously. Both the temperature graph and the temperature value are updated periodically. All data is displayed in centigrade unit with the graph display range of 16°C to 36°C. The easiest way to change the temperature is to lightly blow air (warm or cold) on the sensor (U4).

In the Data Logging mode, temperature data is not sent to the host continuously. Instead, temperature data is sampled every second and stored in the data memory buffer until the “Acquire Data” button is clicked. At that point, the firmware sends everything in the buffer to the PC, then empties the buffer. Data can be acquired at any time; if the time elapsed since the last acquisition is less than 30 seconds, then a smaller set of data points will be acquired and displayed in the message window. Clicking on “Acquire Data” too rapidly may return a “WARNING-No Data Acquired” message; this only means that the controller has not had an opportunity to collect a new temperature sample. All data points are displayed at one time; the temperature graph is also updated at that time. Clicking on “Clear Screen” clears the message window.

The firmware can store up to 30 data points; therefore it takes 30 seconds to fill the data memory buffer. When the buffer becomes full, the firmware replaces the oldest data point with a newer data value.

FIGURE 3-3: MESSAGE WINDOW IN DATA LOGGING MODE



Toggle LEDs

The Toggle LEDs box contains two buttons used to control the status of LED D3 and D4 on the demonstration board. The corresponding LED is turned on when a button is pressed and off when the same button is pressed again.

Potentiometer Display

The Potentiometer Display reflects the current value of the potentiometer on the demonstration board. The value ranges from 0 to 10 kΩ. Turning the potentiometer on the board causes the analog dial gauge and digital display to change accordingly in real time.

Ending Demo Mode

To stop the demonstration program, click “Disconnect” or exit the application. Clicking on the “Bootload Mode” tab also terminates Demo mode.

3.5 BOOTLOAD MODE

The Bootload mode provides a simple means of re-programming the on-board microcontroller without using an external programming device. The PICDEM FS USB board is pre-programmed with a bootload program that allows designers to download and test new code through the Demo Tool application.

This section describes the Bootload mode, as well as a brief overview of the architecture of the bootload program. An understanding on how the bootload firmware coexists with the user firmware is important in developing a successful end application.

3.5.1 Bootload Mode Entry

To start using the application, press and hold S2 while resetting the board (pressing and releasing S1). The entry condition on the PICDEM FS USB board and the provided firmware is determined by the status of the switch button S2, which is checked once after each reset. If the button is held down during a Reset, the microcontroller enters the bootload mode; otherwise, it starts executing user code from address 0x0800.

Note:	Even if the Demo Tool application is running in Bootload mode, a simple reset of the board (pressing S1) will not cause the board itself to enter Bootload mode.

After the reset, select "PICDEM FS USB 0 (Boot)" from the "Select PICDEM FS USB Board" dropdown box. When the Demo Tool application successfully connects to the board, the message "USB Bootload Firmware Version x.x" will appear on the left side of the status bar at the bottom of the window.

3.5.2 Memory Organization

Program Memory Usage

Figure 3-4 shows the memory map of the PIC18F4550. The first 2,048 bytes of program memory are reserved as the boot block, which is utilized almost entirely for the bootloader firmware. The bootloader is a self-contained program with its own USB driver firmware, USB descriptor set, bootload command interpreter, and bootload function handlers. The USB driver firmware used with the Bootloader is a modified version of Microchip USB Firmware Framework library, and incorporates a smaller set of features and more static design.

The boot block can be write protected to prevent accidental overwriting of the boot program. The default setting from factory has the boot block write protect option enabled.

Remapped Vectors

Since the hardware reset and interrupt vectors lie within the boot area and cannot be edited if the block is write-protected, they are remapped through software to the nearest parallel location outside the boot block: 0x0800 for Reset, 0x0808 for the High Priority Interrupt vector, and 0x0818 for the Low Priority vector. Remapping is simply a `GOTO` instruction for interrupts. Users should note that an additional latency of two instruction cycles is required to handle interrupts.

Memory Spaces

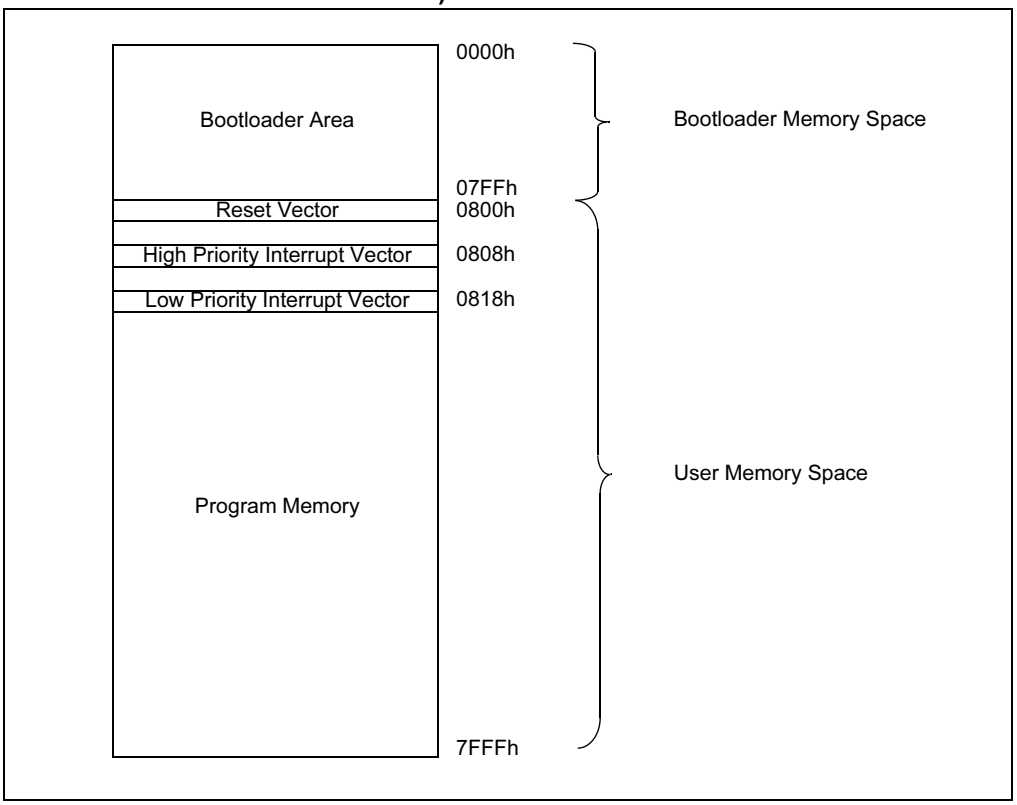
There are four memory spaces that the bootloader can access and program.

- Program Memory (0x0800 – 0x7FFF)
- User ID Memory (0x200000 – 0x200007)
- EEPROM Memory (0x0F0000 – 0x0F00FF)
- Configuration Memory (0x300000 – 0x30000D)

The Bootload also reads and displays the read-only Device ID words at 0x3FFFFE and 0x3FFFFF.

Note: The data EEPROM is actually located in a different memory space than shown here. The address range given is remapped by the bootloader to the proper memory when data EEPROM access is required.

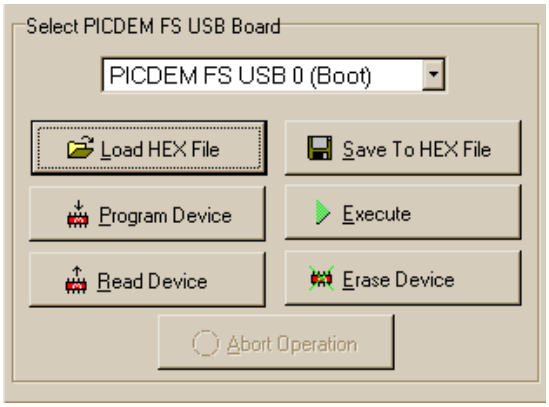
FIGURE 3-4: PIC18F4550 PROGRAM MEMORY MAP (BOOTLOADER IMPLEMENTED)



3.5.3 Using the Bootloader

Once the board has been selected in the dropdown list, the bootloader control buttons become active (Figure 3-5).

FIGURE 3-5: BOOTLOAD MODE CONTROLS



- **Load HEX File:** This loads a hex file into the memory buffer and displays the content in the message window. The data loaded can be used to program a target device. A valid hex file must conform to the Intel HEX 32 format. If an invalid file is selected, a warning message, "WARNING - No HEX file data.", is displayed. If device configuration data is present, the bootloader firmware will check it for configuration conflicts that might disable the board. These are discussed in more detail in **Section 3.5.4 "Consideration When Using the Bootloader"**. Each project's hex file, MCHPUSB.hex, can be found in the '_output' directory, i.e., C:\MCHPFSUSB\fw\Cdc_output and C:\MCHPFSUSB\fw\Hid\Mouse_output.
- **Read Device:** The Read Device function reads the entire memory range into the memory buffer and displays the content in the message window. Program memory, EEPROM, the User ID, and configuration data are all read. A successful read operation is indicated by the message "MESSAGE - Read Completed" in the Message window. If the operation failed, a warning, "WARNING - Failed to read", is shown.
- **Erase Device:** The Erase Device function erases the user program memory space only (0x0800 to 0x7FFF). It does not erase the EEPROM, User ID or configuration data.
- **Execute:** The Execute function sends a Reset command to the bootloader firmware, causing a microcontroller reset. If S2 is not pressed during this time, the user code in program memory will start executing; otherwise, the firmware will re-enter the bootload mode. If the operation failed, perform a hard reset on the demo board by pressing S1.
- **Save To HEX File:** The Save To HEX File function saves the data contained in the memory buffer to a file, whether it was loaded from the Read Device or Load HEX file functions. It only saves the data type present in the memory buffer. If a hex file is loaded and does not have a particular type of memory, i.e., EEPROM, that data section will not be saved in the output file. All memory sections are always loaded after a Read Device function.

- **Program Device:** The Program Device function programs the target device with the data loaded in the memory buffer. Only the type of memory present in the memory buffer will be programmed. The function automatically erases the program memory (0x0800 to 0x7FFF) and User ID memory (0x200000 to 0x200007) before writing new data to these locations. Memory contents are verified after the write operation.
- **Abort Operation:** The Abort Operation function is enabled when programming, reading, or erasing the device. This function causes the current operation to terminate.

3.5.4 Consideration When Using the Bootloader

The PIC18F4550 microcontroller for the PICDEM FS USB board has a specific configuration setting that is necessary for the bootload program to function. The USB Voltage Regulator and device oscillator settings are both critical, and cannot be changed. Other settings, such as code protection, WDT and LVP, are less critical but may cause irreversible side effects. The default configuration values are shown in Table 3-1.

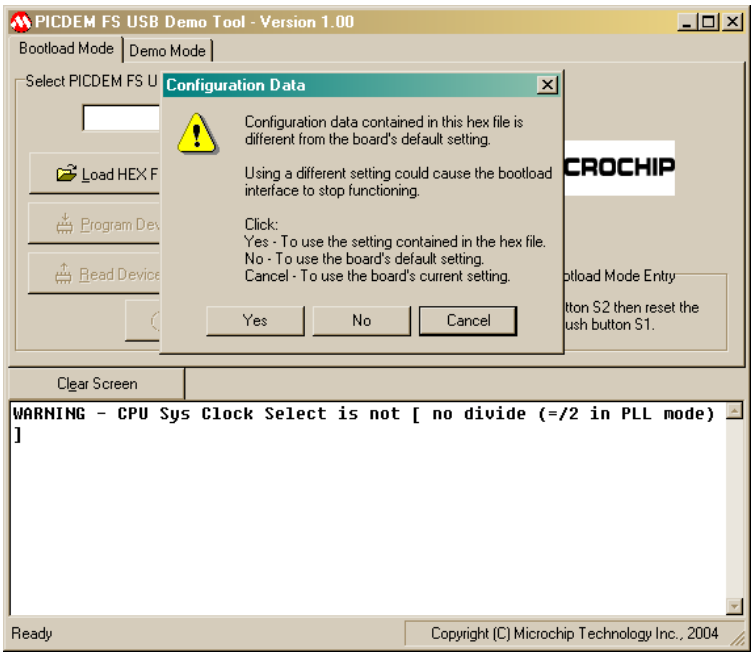
The configuration data contained in a hex file may violate the restrictions on the critical configuration settings described above. Should this happen, the Demo Tool will display a warning dialog box (Figure 3-6). This will also be accompanied by a brief text description of the configuration conflict. Users have the option to accept the new configuration settings, use the board's factory configuration settings, or maintain the current configuration setting.

TABLE 3-1: DEFAULT CONFIGURATION WORD VALUES FOR THE PICDEM FS USB BOARD

Address	Register	Value	Comment
0x300000	CONFIG1L	0x24	Clock configuration
0x300001	CONFIG1H	0x0E	Clock configuration
0x300002	CONFIG2L	0x3F	BOR, PWRT, USB voltage regulator
0x300003	CONFIG2H	0x1E	WDT configuration
0x300005	CONFIG3H	0x81	MCLR, CCP2, A/D configuration
0x300006	CONFIG4L	0x81	Core microcontroller configuration
0x300008	CONFIG5L	0x0F	Code protect (program memory)
0x300009	CONFIG5H	0xC0	Code protect (boot block/EEPROM)
0x30000A	CONFIG6L	0x0F	Write protect (program memory)
0x30000B	CONFIG6H	0xA0	Write protect (boot block/EEPROM)
0x30000C	CONFIG7L	0x0F	Table Read protect (program memory)
0x30000D	CONFIG7H	0x40	Table Read protect (boot block/EEPROM)

Note: Only implemented configuration words are listed.

FIGURE 3-6: CONFIGURATION CONFLICT WARNING DIALOG WITH TYPICAL DIAGNOSTIC MESSAGE



When a file is loaded, either from a device or a file, the Bootloader demo software also parses the file to determine if code is available for the different memory areas. If a device is programmed with that file, only those memory areas with code present are programmed; the other memory areas are left unchanged. For example, a hex file that is missing data for the data EEPROM or configuration words will only program the target device's program memory and User ID spaces; the existing configuration and stored EEPROM information will be preserved.

3.5.5 Writing Application Code with the Bootloader

The bootloader operates as a separate entity, which means that an application can be developed with very little concern about what the bootloader is doing. This is as it should be; the bootloader should be dormant code until an event initiates its operation. Ideally, bootloader code should never be running during an application's intended normal operation.

When developing an application with a resident bootloader, some basic principles must be kept in mind:

3.5.5.1 WRITING IN ASSEMBLY

When writing in assembly, the boot block and new vectors must be considered. For modular code, this is usually just a matter of changing the linker script file for the project. An example is shown in Example 3-1. If an absolute address is assigned to a code section, the address must point somewhere above the boot block.

For those who write absolute assembly, all that is necessary is to remember that the new Reset vector is at 800h, and the interrupt vectors are at 808h and 818h. Except for the bootloader, no code should reside in the boot block area.

EXAMPLE 3-1: ASSEMBLY LINKER SCRIPT FOR USE WITH BOOTLOADER

```
// Sample linker command file for 18F4550 with Bootloader

LIBPATH .

CODEPAGE   NAME=boot         START=0x0         END=0x7FF     PROTECTED
CODEPAGE   NAME=page         START=0x800       END=0x7FFF
CODEPAGE   NAME=idlocs       START=0x200000    END=0x200007  PROTECTED
CODEPAGE   NAME=config       START=0x300000    END=0x30000D  PROTECTED
CODEPAGE   NAME=devid        START=0x3FFFFE    END=0x3FFFFF  PROTECTED
CODEPAGE   NAME=eedata       START=0xF00000    END=0xF000FF  PROTECTED

ACCESSBANK NAME=accessram    START=0x0         END=0x5F
DATABANK   NAME=gpr0         START=0x60        END=0xFF
DATABANK   NAME=gpr1         START=0x100       END=0x1FF
DATABANK   NAME=gpr2         START=0x200       END=0x2FF
DATABANK   NAME=gpr3         START=0x300       END=0x3FF
DATABANK   NAME=usb4         START=0x400       END=0x4FF     PROTECTED
DATABANK   NAME=usb5         START=0x500       END=0x5FF     PROTECTED
DATABANK   NAME=usb6         START=0x600       END=0x6FF     PROTECTED
DATABANK   NAME=usb7         START=0x700       END=0x7FF     PROTECTED
ACCESSBANK NAME=accesssfr    START=0xF60       END=0xFFF     PROTECTED

SECTION    NAME=CONFIG       ROM=config
```

3.5.5.2 WRITING IN C

When using the MPLAB C18 C compiler to develop firmware for an application, the user has the choice to either rebuild the standard start-up object (`c018.o` or `c018i.o`) with the new Reset vector, or to insert the extra code shown in Example 3-2. The latter method is recommended.

Like modular assembly, the linker file must be changed to incorporate the protected boot block and new vectors. An example is shown in Example 3-3.

For users of other compilers, check with the compiler's software user guide to determine how to change the start-up code and vectors.

EXAMPLE 3-2: RESET VECTOR INSERT FOR APPLICATIONS IN C

```
extern void _startup (void);
// See c018i.c in your C18 compiler directory
#pragma code _RESET_INTERRUPT_VECTOR = 0x000800
void _reset (void)
{
       _asm goto _startup _endasm
}
#pragma code
```

EXAMPLE 3-3: C18 LINKER SCRIPT FOR USE WITH BOOTLOADER

```
// Sample linker command file for 18F4550 with Bootloader

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f4550.lib

CODEPAGE   NAME=boot         START=0x0         END=0x7FF     PROTECTED
CODEPAGE   NAME=vectors      START=0x800       END=0x829     PROTECTED
CODEPAGE   NAME=page         START=0x82A       END=0x7FFF
CODEPAGE   NAME=idlocs       START=0x200000    END=0x200007  PROTECTED
CODEPAGE   NAME=config       START=0x300000    END=0x30000D  PROTECTED
CODEPAGE   NAME=devid        START=0x3FFFFE    END=0x3FFFFF  PROTECTED
CODEPAGE   NAME=eedata       START=0xF00000    END=0xF000FF  PROTECTED

ACCESSBANK NAME=accessram    START=0x0         END=0x5F
DATABANK   NAME=gpr0         START=0x60        END=0xFF
DATABANK   NAME=gpr1         START=0x100       END=0x1FF
DATABANK   NAME=gpr2         START=0x200       END=0x2FF
DATABANK   NAME=gpr3         START=0x300       END=0x3FF
DATABANK   NAME=usb4         START=0x400       END=0x4FF     PROTECTED
DATABANK   NAME=usb5         START=0x500       END=0x5FF     PROTECTED
DATABANK   NAME=usb6         START=0x600       END=0x6FF     PROTECTED
DATABANK   NAME=usb7         START=0x700       END=0x7FF     PROTECTED
ACCESSBANK NAME=accesssfr    START=0xF60       END=0xFFF     PROTECTED

SECTION    NAME=CONFIG       ROM=config

STACK SIZE=0x100             RAM=gpr3
```

An example user program is provided as a reference on how to set up a project to be used with the bootloader. The firmware example can be found in `C:\MCHPFSUSB\fw\Demo02`.

Chapter 4. Using the Microchip USB Firmware Framework

4.1 HIGHLIGHTS

The items discussed in this chapter are:

- Overview of the Framework
- USB Firmware in the Framework

4.2 OVERVIEW OF THE FRAMEWORK

The Microchip USB Firmware Framework is a file system construct that can be used to create new USB applications. It can be thought of as a reference design project, containing the necessary firmware code for USB operation and providing a placeholder for the user's code. The whole code project is contained within one single root project directory, with many subdirectories for source code organization.

The USB Framework is based on the latest versions (as this is written) of Microchip's development tools. To provide the best level of USB application support, you should verify that you have at least these revisions of the following tools before starting:

- MPLAB IDE, v 6.62
- Microchip C18 Complier, v 2.30.01

This section describes the importance in setting up the project paths and how the framework is organized.

4.2.1 The Framework Directory Structure

The file structure consists of a collection of subdirectories and specific files under a root project directory. The root directory for a USB project may be located in any location and have any valid directory name; however, the subdirectories structure should always be maintained. Each of the example projects included with the PICDEM FS USB demonstration board uses this structure.

The following subdirectories and files should always be present:

Subdirectories:

- `_output`: Central location for output files
- `autofiles`: Contains USB global configuration file and descriptors
- `system`: Contains USB firmware
- `user`: Placeholder for the user's firmware

Files:

- `CleanUp.bat`: Delete all output files in this folder and any subdirectories.
- `io_cfg.h`: Maps pin names to pin functions; this file should be modify to match each target board. The default example is designed for the PICDEM FS USB demonstration board.
- `main.c`: The file which contains `main()` for the application
- `MCHPUSB.mcp`: MPLAB IDE Project file
- `MCHPUSB.mcw`: MPLAB IDE Workspace file

4.2.2 The Framework Logical Structure

The USB Firmware Framework also provides a set of modular firmware interfaces that handle most of the work for implementing USB communications. Figure 4-1 shows a typical USB program flow. Each firmware reference project is written to have a cooperative multitasking environment; thus, no blocking functions should be implemented.

The `main()` function is an infinite `while` loop that services different tasks. These can be logically thought of as either a USB task or a user task. USB tasks are handled by `USBDriverService()`, which polls and services all USB hardware interrupts. When a control endpoint transaction is received, `USBCtrlEPService()` is called. All transfers over the default control endpoints must follow the control transfer protocol as described in the USB specification.

The control transfer service is provided by functions in `usbctrltrf.c`. The first stage of all control transfers arrive as a request. A USB request can be either standard or class-specific. A standard request is serviced by the `USBCheckStdRequest()`, which handles the required requests as specified in Chapter 9 of the USB specification. A class-specific request must be handled by the firmware file that knows how to service it. Examples of device classes are Human Interface Device (HID) and Communication Device Class (CDC). (The example in Figure 4-1 shows HID; it could refer to any other device class just as easily.) The handlers for a class specific request should be contained in the specific device class firmware file, such as hid.c or cdc.c. The naming scheme for a function that services a class specific request is `USBCheck<class name>Request()`, where `<class name>` can be any class specific names.

The USB enumeration process is handled mainly in `usb9.c`. One of the most important steps in the enumeration process is the handling of a SET_CONFIGURATION request, which is done by `USBStdSetCfgHandler()`. This function can be modified by the user to call the appropriate functions to initialize the application endpoints. Each specific device class firmware should have one endpoint initialization routine. The naming convention for this function is `<class name>InitEP()`, where `<class name>` can be any device class names. An example is `HIDInitEP()`, which contains example code on how to initialize a group of endpoints that are used in the class. Users should take care to check which configuration index is being asked to be set by the USB host. A device can have multiple configurations, and not all interfaces may be the same across different configuration.

The user's application code is also called from the main program loop; it resides under the default function name `ProcessIO()`. When the application needs to send or receive a USB transaction, it can call a pre-written function that services the transmit or receive functionality which are declared in the class specific firmware code, such as `HIDRxReport()` and `HIDTxReport()`.

USB device configuration is also handled in a modular manner, by modifying variables in a small number of files; the information is then made globally available at compile time. The files are tightly interdependent, and pass information between themselves during compile time to create the complete USB configuration. The relationships are shown in Figure 4-2. Customizing the configuration files is discussed in **Section 4.3 "USB Firmware in the Framework"**.

FIGURE 4-1: RELATIONSHIPS BETWEEN MICROCHIP USB FRAMEWORK FILES AND A TYPICAL HID APPLICATION

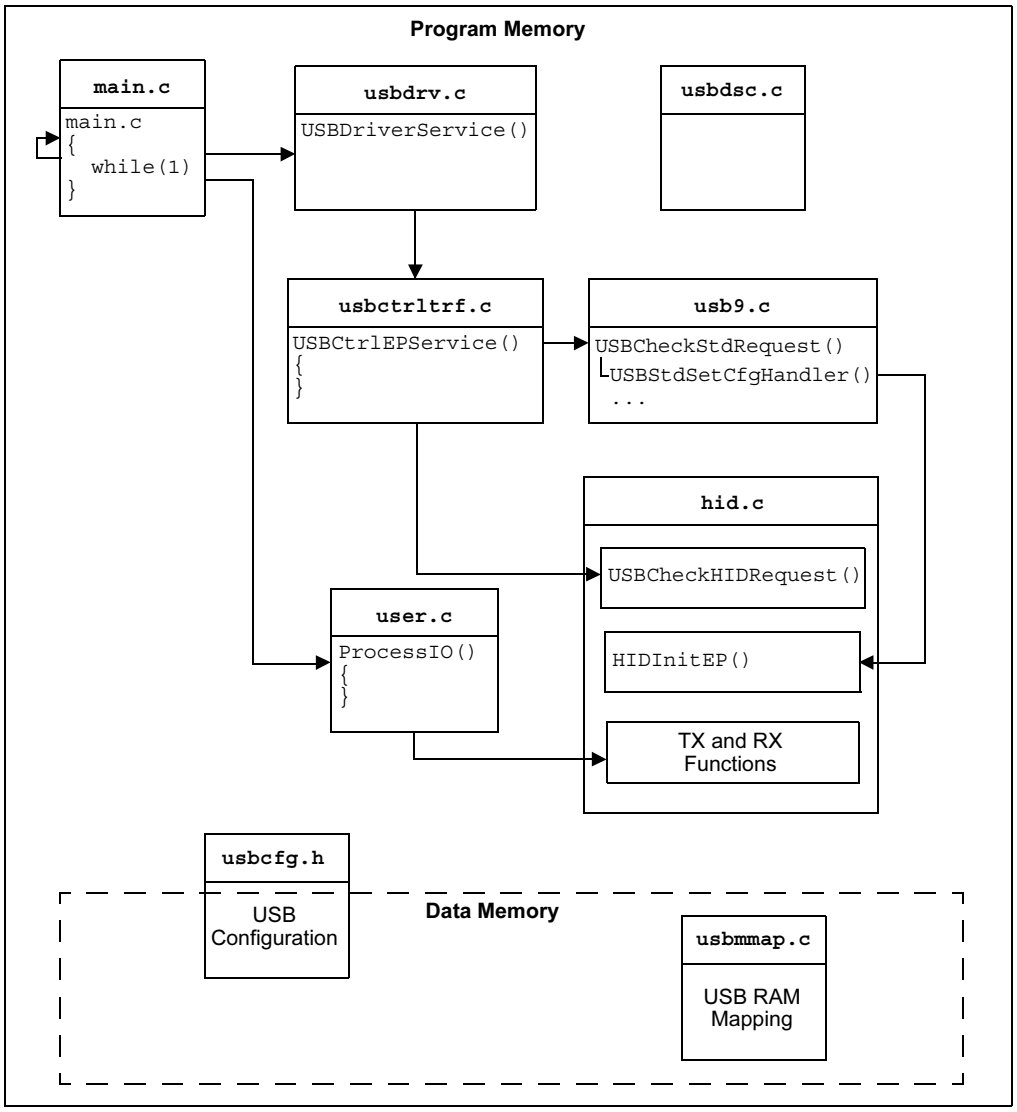
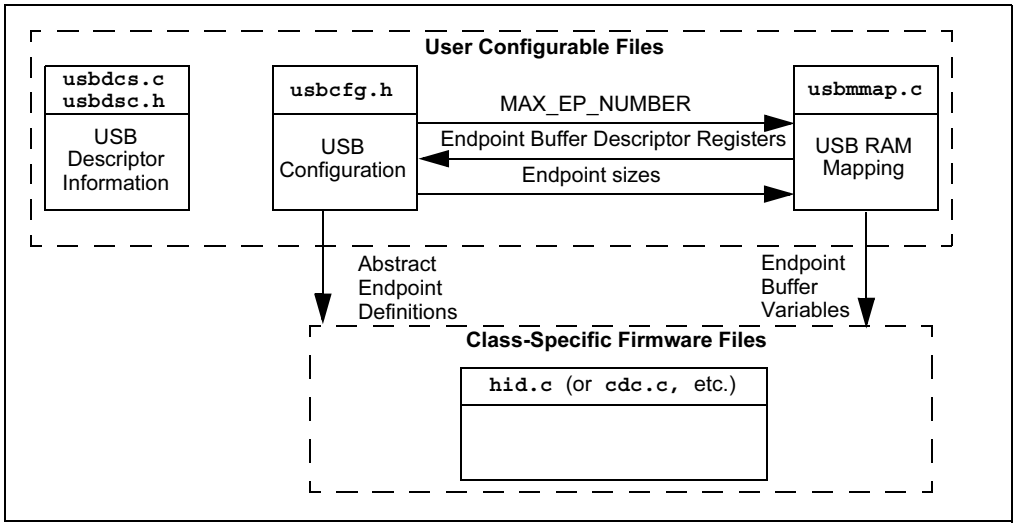


FIGURE 4-2: COMPILE-TIME RELATIONSHIPS BETWEEN USB CONFIGURATION FILES



4.2.3 Configuring MPLAB for the USB Framework

When setting up a project from the reference design, the first thing to check is the project paths. These should be updated to reflect the actual project's root directory as needed. The steps in updating the project paths will be demonstrated through an example.

In this example, we will use the HID Mouse Reference project, located in the directory `C:\MCHPFSUSB\fw\HID\Mouse`. You may need to adjust your target directories according to where the project is actually located during installation.

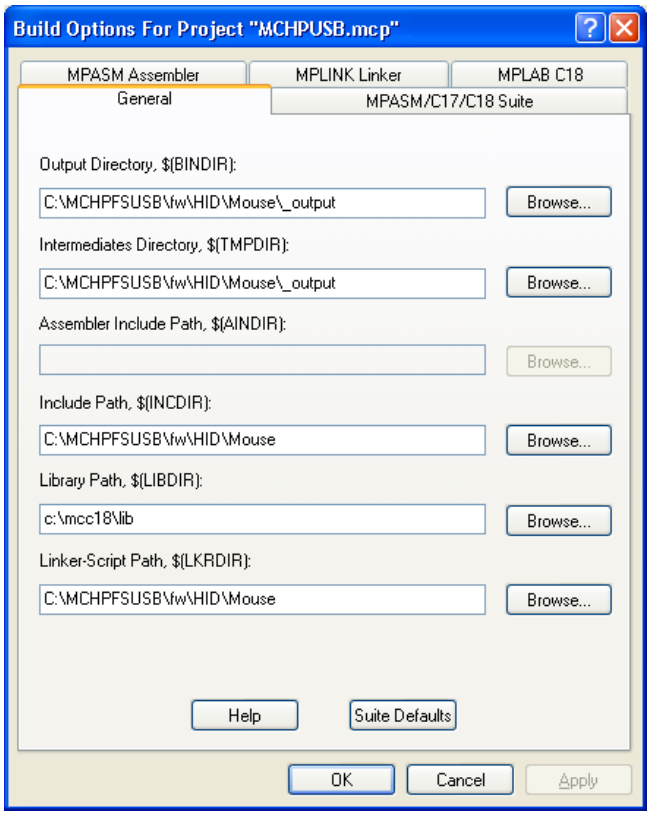
To open a project workspace:

1. Launch MPLAB.
2. From the MPLAB menu bar, select *File > Open Workspace...*
3. Navigate and select the project file `C:\MCHPFSUSB\fw\HID\Mouse\MCHPUSB.mcw`.
4. Return to the menu bar and select *Project > Build Options... > Project*. The Build Options dialog appears (Figure 4-3).
5. Click on the **General** tab and verify the following:
   - The "Output Directory" and "Intermediates Directory" paths should point to the `_output` folder under your root project directory.
   - The "Include Path" directory should point to the project's root directory.
   - The "Library Path" should point to the "lib" directory under the C18 program folder.
   - The "Linker-Script Path" should point to the directory of the project's linker script file.

Note 1: In some cases, the C18 compiler may fail to compile correctly, giving an error that certain header files from the mcc18 library could not be found. In this case, add the path `C:\mcc18\h` after the project root path in the "Include Path" field (separated with a semicolon).

2: If the C18 compiler is installed in a path other than `C:\mcc18\`, the actual path should be used instead.

FIGURE 4-3: CONFIGURING MPLAB IDE FOR THE MICROCHIP USB FRAMEWORK



4.3 USB FIRMWARE IN THE FRAMEWORK

The subdirectories `autofiles` and `system` contain the USB firmware source code in the framework. Some of these files should never be modified, while others must be edited to configure the application. Each folder and its files will be discussed in details of what they do and how to modify them to match your application.

4.3.1 Autofiles Directory

This directory contains three key configuration files for USB operation:

- `usbcfg.h`: Global USB Configuration File
- `usbdsc.c`: USB Descriptor File
- `usbdsc.h`: USB Descriptor File Header

4.3.1.1 USBCFG.H

This header file is the most important file of the entire project. It defines the endpoints mapping to their class functions. It also defines the Microchip USB ID (MUID) number, which is used to determine which USB class currently owns the control transfer in progress. Finally, it defines the maximum buffer size for endpoint 0.

To centralize the configuration of important USB features, all of the USB-related compile time options are found here. The file is constructed as a series of `#define` compiler directives. The options to configure are:

- EP0_BUFF_SIZE [*buffer_size*]
- MAX_NUM_INT [*max*]
- MODE_PP [*_PPBMn*]
- UCFG_VAL [*option1* | *option2...*]
- USE_SELF_POWER_SENSE_IO
- USE_USB_BUS_SENSE_IO
- USB_USE_GEN or USB_USE_CDC or USB_USE_HID
- MAX_EP_NUMBER [*max_ep*]

EP0_BUFF_SIZE defines the buffer size for endpoint 0. It can have a valid value of 8, 16, 32 or 64 bytes. This definition is used globally in the project for many things. It is used during project build to allocate appropriate buffer size for endpoint 0. It is used in the USB descriptor to notify the USB host of the size of the endpoint 0 buffer. It is also used during control transfer operation.

When defining this variable, note that a low-speed USB device can only use an 8-byte buffer, while a Full-Speed USB device can use an 8, 16, 32 or 64-byte buffer.

MAX_NUM_INT defines the size of the array which keeps track of the active alternate setting for each interface, which can be switched during operation. Valid values are integers [0, 1, 2,...]. If a device has multiple configurations, the number of interfaces from the configuration with the highest number of interfaces should be used.

For example, a device with two configurations has three interfaces in the first configuration and two interfaces in the second. In this case, MAX_NUM_INT should be three.

MODE_PP defines the ping-pong buffer mode to be used during runtime. The allowed values are _PPBM0, _PPBM1 and _PPBM2; each of these are in turn are defined in the header `usbdrv.h`. The function of each mode is explained in the PIC18F2455/2550/4455/4550 device data sheet.

Note:	The current version of the firmware only supports _PPBM0 mode.

UCFG_VAL defines the initial value for the UCFG special function register. The allowed values are:

- _LS or _FS: Select Low- or Full-speed mode
- _TRINT or _TREXT: Select Internal or External Transceiver mode
- _PUEN: Use internal USB pull-up resistor
- _OEMON: Use SIE output indicator
- _UTEYE: Enable eye-pattern test output

The options can be ORed together (for example, `#define UCFG_VAL _PUEN|_TRINT|_FS`). This value is used by the firmware framework to initialize the UCFG register. A full explanation of the options can be found in the description of the UCFG register in PIC18F2455/2550/4455/4550 device data sheet.

Note 1: If the _LS option is selected, the device clock configuration must be changed to support Low Speed operation, Refer to the device data sheet for additional information.

2: The _UTEYE option is used only for device testing, and never in live applications. When _UTEYE is enabled, the device transmits continuously. Do not plug the device to a USB host when the option is enabled.

USE_SELF_POWER_SENSE_IO indicates that the microcontroller is sensing the presence of on-board power through an I/O pin. If the target board design does not use an I/O pin to detect the present of self-power, this definition must be commented out.

USE_USB_BUS_SENSE_IO indicates that the firmware will use the pin defined in `io_cfg.h` to determine when to enabled the USB module. If the target board design does not use an I/O pin to detect the present of the USB bus, this definition must be commented out.

When USE_USB_BUS_SENSE_IO is undefined, the USB module will always be enabled. Using this feature helps to improve the power efficiency of the system because the USB module is only enabled when the bus is present.

USB_USE_GEN, **USB_USE_CDC** and **USB_USE_HID** are used to indicate which USB classes should be included in the code project. When each of these are defined, it tells the USB global header file `usb.h` which class-specific header files to include. The `usb.h` header is used globally as the necessary include file when using the USB library. If the HID class is used, then `hid.c` and `hid.h` should also be added to the MPLAB project. If the CDC class is used, then `cdc.c` and `cdc.h` should also be added to the MPLAB project.

This definition is also used when the SET_CONFIGURATION request is received. User can modify the 'Modifiable Section' in the `USBStdSetCfgHandler` function in `usb9.c`. The calls located in the modifiable section initialize endpoints used by each specific USB class, which are mapped by `usbcfg.h`.

MAX_EP_NUMBER must equal the highest endpoint number used in the project. For example, if the highest endpoint number used is endpoint 5, then MAX_EP_NUMBER should equal five. This definition is used mainly in the `usbmmap.c` to allocate the buffer descriptor registers.

4.3.1.2 USBDSC.C AND USBDSC.H

These files contain the USB descriptor information for the device. The main file, `usbdsc.c`, defines the descriptor itself; `usbdsc.h` defines the descriptor structure, which is used to calculate the descriptor size with the `sizeof()` statement. 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 `usbdsc.h`.

A typical configuration descriptor consists of these components:

- At least one configuration descriptor (USB_CFG_DSC)
- One or more interface descriptors (USB_INTF_DSC)
- One or more endpoint descriptors (USB_EP_DSC)

In addition, there is usually a descriptor string that provides a plain text description of the device.

4.3.1.2.1 Customizing USBDSC.C

The version of `usbdsc.c` included with the Framework Firmware indicates which parameters must be defined, and can serve as a template for developing new device descriptors. Most items should be self-explanatory, but some options are not explained in the file's comments and are expanded upon below.

- **USB_CFG_DSC**

  The configuration attribute (the item immediately following the string index) must always have the _DEFAULT definition at the minimum. Two additional options, _SELF and _RWU, can be ORed with _DEFAULT. _SELF tells the USB host that this device is self-powered, while _RWU tells the USB host that the device supports Remote Wakeup. Definitions for these options are provided in `usbdefs_std_dsc.h`.

- **USB_EP_DSC**

  An endpoint descriptor has a form similar to:

  ```
  sizeof(USB_EP_DSC),DSC_EP,_EP01_OUT,_BULK,64,0x00
  ```

  The first two parameters specify the length of this endpoint descriptor (7) and the descriptor type. The next parameter identifies the endpoint. It takes the format:

  ```
  _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 definitions are provided in:

  `usbdefs_std_dsc.h.`

  The next parameter identifies the type of the endpoint. Available options are `_BULK`, `_INT`, `_ISO` and `_CTRL` (for Bulk, Interrupt, Isochronous and Control endpoints, respectively). The `_CTRL` is not typically used because the default control transfer endpoint is not defined in the USB descriptors. When `_ISO` is used, additional options can be ORed with it. For example:

  ```
  _ISO |_AD |_FE
  ```

  describes the endpoint as an isochronous pipe with adaptive and feedback attributes. See `usbdefs_std_dsc.h` and the USB specification for details.

  The final parameters define the maximum size of the endpoint and the polling interval.

- **The USB Descriptor String**

  Rather than appearing as a simple string of text, the descriptor is formatted in a particular data structure. The string descriptor array takes the format:

  `rom struct{byte bLength;byte bDscType;word string[size];}sdxxx={ sizeof(sdxxx),DSC_STR,<text>};`

  This structure provides a means for the C compiler to calculate the length of string descriptor `sdxxx`, where `xxx` is the string index number. The first two bytes of the descriptor are the descriptor length and type.

  The remaining `<text>` are string texts which must be in Unicode format. This 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>`; which must be calculated manually by counting characters and then entered into the array declaration. For 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. The firmware supports the management of multiple strings through a look-up table, which is defined as:

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

  The above example has 3 strings (`sd000`, `sd001` and `sd002`). Strings can be removed or added as needed. The strings are managed by the look-up table `USB_SD_Ptr`; the index of the string must match the index position of the array, (`&sd000` must be in position USB_SD_Ptr[0], and so on). The look-up table `USB_SD_Ptr` is used by the `USBStdGetDscHandler` function in `usb9.c`.

  `sd000` is a specialized string descriptor. It defines the language code, which is usually US English (0x0409).

4.3.1.2.2 Customizing USBDCS.H

In the header file `usbdsc.h`, variables for each of the descriptor components are named with the following conventions:

USB_CFG_DSC types are named `cdxx`, where `xx` is the configuration number. This number should match the actual index value of the configuration.

USB_INTF_DSC types are named `i<yy>a<zz>`, where `yy` is the interface number and `zz` is the alternate interface number.

USB_EP_DSC types are 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 the endpoint belongs to.

Example 4-1 shows the structure in `usbdsc.h` of a configuration descriptor. This device has one configuration with two interfaces: interface 0 with two endpoints (in and out), and interface 1 with one endpoint (in). The hierarchy of the descriptors shown here follows the USB specification requirement. All endpoints belonging to an interface should be listed immediately after that interface.

EXAMPLE 4-1: DEFINING A CONFIGURATION IN USBDSC.H

```
#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
```

4.3.1.2.3 Adding Configurations

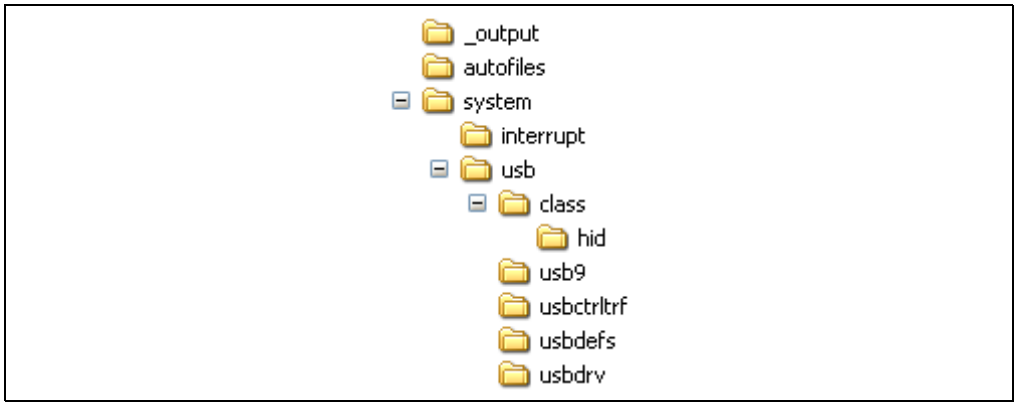
A USB device may have more than one configuration descriptors (e.g., CFG02, CFG03, etc.). To add another configuration descriptor, implement a new set of structures, similar to CFG01, to both `usbdsc.c` and `usbdsc.h`. These will be named CFG02 (or CFG03, and so on). Once this is done, add the new configuration descriptor name (`cfg02`, `cfg03`) to the look-up table `USB_CD_Ptr`, in the same method used for managing descriptor strings. `USB_CD_Ptr[0]` is a dummy place holder for configuration 0, the unconfigured state defined by the USB specification.

The configuration handler `USBStdSetCfgHandler` must also be modified to support the additional configurations.

4.3.2 System Directory

The project system directory contains many of the USB core function files. Its structure is shown in Figure 4-5.

FIGURE 4-4: DIRECTORY TREE STRUCTURE FOR THE USB FRAMEWORK



In addition to the USB memory manager `usbmmap.c`, the key files discussed in this section are located in the following folders in the USB folder:

- `usb9`: `usb9.c` and `usb9.h`
- `usbctrltrf`: `usbctrltrf.c` and `usbctrltrf.h`
- `usbdrv`: `usbdrv.c` and `usbdrv.h`
- `usbdefs`: the standard definition files `usbdefs_ep0_buff.h` and `usbdefs_std_dsc.h`.

4.3.2.1 USBMMAP.C

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 buffers.

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 `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 a 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. (This assumes Ping-Pong Buffer mode 0, which assigns 4 bytes to each BDT, and one BDT to the in and out endpoints for each endpoint.) It does not make much sense to skip endpoints, but the final decision lies with the user.

The instantiated endpoint name is then used in `usbcfg.h` for mapping the endpoint to its function. For example, assume a USB device class “foo”, which uses one in and one out endpoint, each one being 64 bytes. In Example 4-2, we have chosen this class to use endpoint 1. (The names are arbitrary and can be anything other than FOO_??????). For abstraction, any code written for devices of this class should use the abstract definitions such as FOO_BD_OUT, FOO_BD_IN and so on, and not `ep1Bo` or `ep1Bi`. Note that the endpoint size defined in `usbcfg.h` is again used in `usbmmap.c`. This shows that the relationship between the two files are tightly related.

EXAMPLE 4-2: DEFINING THE ENDPOINTS FOR CLASS FOO

```
#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 endpoint buffer for each USB function must be located in the dual-port RAM area (0x400 to 0x7FF) 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 to 0x05F).

For the variable to be globally accessible by other files, its prototype should also be listed in the header file `usbmmap.h` with an `extern` keyword, such as:

```
extern volatile far unsigned char[FOO_EP_SIZE] data;
```

4.3.2.2 USBDRV.C

This file provides low-level USB services, and handles all USB hardware interrupts. This section describes functions which are relevant to user in details.

4.3.2.2.1 USBCheckBusStatus Function

This function should be called once per main loop. When an I/O sense pin is used to detect the attachment and detachment of a USB cable, this routine enables or disables the USB module accordingly. This helps increase the power efficiency of the system. If an I/O sense pin is not used, the `#define USE_USB_BUS_SENSE_IO` statement in `usbcfg.h` should be commented out. This causes `USBCheckBusStatus()` to enable the USB module after device reset regardless of the attachment of a USB cable.

4.3.2.2.2 USBDriverService Function

This functions polls all of the USB hardware interrupt flags in the UIR and UIE registers to check if a USB event has occurred, and services the event. It also clears the USB transaction completion flag, TRNIF, which is set after each USB transaction completion.

Issues can occur when an application tries to send or receive transactions more than four times without clearing the TRNIF flag. This is because the 4-level hardware FIFO stack for storing transaction completions can only be emptied one level at a time by clearing TRNIF. When more than four transactions are received without TRNIF being cleared, no more transactions can be processed. An example of how this may happen is shown in Example 4-4. This stops the USB module from responding to any further host requests until more stack becomes available. More information regarding USTAT and transaction FIFO can be found in the USB chapter of the of the PIC18F2455/2550/4455/4550 device data sheet.

The user's application code should not clear the TRNIF bit directly, but call `USBDriverService()` instead, as in Example 4-3 and Example 4-5. Clearing the TRNIF bit directly if an endpoint 0 transaction occurs first will cause that transaction to be lost and not be serviced by the library firmware.

EXAMPLE 4-3: TYPICAL METHOD OF PERFORMING ONE TRANSACTION PER LOOP (PSEUDO CODE)

```
main()
{
    while(1)
    {
        USBDriverService();

        if(!mHIDTxIsBusy())
            HIDTxReport();
    }//end while
}//end main
```

EXAMPLE 4-4: INCORRECT METHOD OF PERFORMING MULTIPLE TRANSACTIONS PER LOOP (PSEUDO CODE)

```
main()
{
    while(1)
    {
        USBDriverService();

        while(mHIDTxIsBusy());
        HIDTxReport();              // 1st Tx
        while(mHIDTxIsBusy());
        HIDTxReport();              // 2nd Tx
        while(mHIDTxIsBusy());
        HIDTxReport();              // 3rd Tx
        while(mHIDTxIsBusy());
        HIDTxReport();              // 4th Tx
        while(mHIDTxIsBusy());      // This will never turn false
                                    // because the USTAT FIFO is full
        HIDTxReport();              // 5th Tx - Unreachable code

    }//end while
}//end main
```

EXAMPLE 4-5: CORRECT METHOD OF PERFORMING MULTIPLE TRANSACTIONS PER LOOP (PSEUDO CODE)

```
main()
{
   while(1)
   {
      USBDriverService();

      while(mHIDTxIsBusy());
      HIDTxReport();                // 1st Tx
      USBDriverService();
      while(mHIDTxIsBusy());
      HIDTxReport();                // 2nd Tx
      USBDriverService();
      while(mHIDTxIsBusy());
      HIDTxReport();                // 3rd Tx
      USBDriverService();
      while(mHIDTxIsBusy());
      HIDTxReport();                // 4th Tx
      USBDriverService();
      while(mHIDTxIsBusy());
      HIDTxReport();                // 5th Tx
      USBDriverService();

      //other application code
   }//end while
}//end main
```

4.3.2.2.3 USBSuspend Function

This routine first puts the USB module into suspended mode and enables USB bus activity interrupt. There is also a modifiable section where user can insert code to carry out a power-saving scheme. Options for implementing power saving are discussed in Microchip application note AN950, *"Power Management for PIC18 USB Microcontrollers with nanoWatt Technology"* (DS00950).

Before putting a device to sleep, a wakeup source must be enabled first. For the USB bus activity interrupt to be a wakeup source, set the ACTVIE bit in the UIE register. Note that the main USB interrupt in the PIE2 register must also be enabled, as shown in Example 4-6.

EXAMPLE 4-6: ENABLING USB SUSPEND

```
PIR2bits.USBIF = 0;     // Make sure flag is cleared
PIE2bits.USBIE = 1;     // Set USB wakeup source
Sleep();                // Goto sleep
PIR2bits.USBIF = 0;     // Clear flag
PIE2bits.USBIE = 0;     // Disable USB interrupt
```

4.3.2.2.4 USBWakeFromSuspend

The routine is called when a bus activity interrupt is set. It disables the bus activity interrupt and re-enable the USB module.

4.3.2.2.5 USBRemoteWakeup Function

If the application supports the Remote Wakeup function, calling this routine sends a USB resume signal to the host. It should be called when an external stimulus other than USB causes the device to become active. Example 4-7 shows a sample code that would reside in the `USBSuspend()` function which would enable both the USB and external stimulus wakeup sources, and call the `USBRemoteWakup()` function if the wakeup source is not USB.

EXAMPLE 4-7: ENABLING USB REMOTE WAKEUP

```
PIR2bits.USBIF = 0;             // Make sure flag is cleared
INTCONbits.RBIF = 0;            // Make sure flag is cleared
PIE2bits.USBIE = 1;             // Set USB wakeup source
INTCONbits.RBIE = 1;            // Set port change wakeup source

Sleep();                        // Goto sleep

if(INTCONbits.RBIF == 1)        // Check if the wakeup source
                                // is the port change interrupt
{
   USBRemoteWakeup();           // If yes, attempt Remote Wakeup
}                               // end if

PIR2bits.USBIF = 0;             // Clear flag
PIE2bits.USBIE = 0;             // Disable USB interrupt
INTCONbits.RBIF = 0;            // Clear flag
INTCONbits.RBIE = 0;            // Disable port change interrupt
```

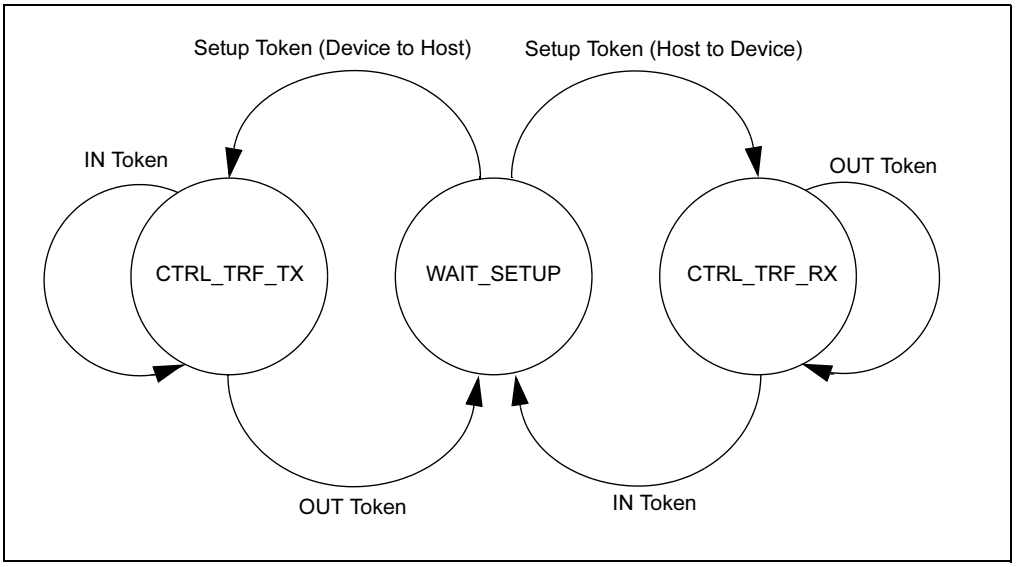
4.3.2.3 USBCTRLTRF.C

This file manages the control transfer, and provides services to all USB classes that need to handle USB requests (USB Chapter 9, HID, CDC, etc.). Since the control transfer service is shared among many USB classes and most transfers expand over multiple transactions, it is important to keep track of which class owns the current control transfer session. This is done by the use of Microchip USB ID (MUID). When MUID is equal to MUID_NULL, it means no classes are capable of handling the request. If the handler in `usb9.c` knows how to service a request, the current control transfer session variable is updated to equal MUID_USB9. Similarly if a HID request is received, then the MUID would be MUID_HID.

The definitions for MUID are defined in `usbcfg.h` and are only project-specific. This means two different projects could have different sets of MUIDs. It is up to the user to manage the usage and assignment of MUID in each project in `usbcfg.h`.

Each control transfer has three stages: Setup, Data and Status. The state machine for handling different control transfer stages in Microchip USB firmware is illustrated in Figure 4-5. There are three different states: WAIT_SETUP, CTRL_TRF_TX and CTRL_TRF_RX.

FIGURE 4-5: USB CONTROL TRANSFER STATE MACHINE



`USBCtrlEPService()` is called from `usbdrv.c`. It only services transactions that come through endpoint 0. It checks for different transaction type to call appropriate handlers (Endpoint 0 SETUP, Endpoint 0 OUT or Endpoint 0 IN).

If the transfer stage is SETUP, then `USBCtrlTrfSetupHandler()` is called to service the setup packet. Each USB control transfer setup packet is always 8 bytes long. There are three steps in servicing a setup packet:

1. The routine initializes the control transfer state machine.
2. It then calls on each of the class specific request handlers in an attempt to find out if any handlers know how to service the request.
3. Once all request handlers have had a chance to analyze the setup packet, the function `USBCtrlEPServiceComplete()` checks the transfer direction of the data stage to determine how to prepare endpoint 0.

`USBCtrlEPServiceComplete()` wraps the remaining tasks in servicing a setup packet. Its main task is to set the endpoint controls appropriately for a given situation. There are three possible outcomes:

- Endpoint 0 is stalled if the request cannot be serviced.
- Endpoint 0 is set up for transferring data to the host during the data stage.
- Endpoint 0 is set up for receiving data from the host during the data stage.

The data stage can be expanded over multiple USB transactions. It is important to keep track of the data source, data destination, data count and data type. These are tracked and updated using four dedicated variables:

- `pSrc`
- `pDst`
- `wCount`
- the memory type flag bit `usb_stat.ctrl_trf_mem`, which can have the value of _ROM or _RAM.

When writing a request handler to send control transfer data from a peripheral device to a host, make sure to do the following:

1. Set data source:
   - For a RAM location: `pSrc.bRam = <RAM buffer location>;`
   - For a ROM location: `pSrc.bRom = <program memory location>;`
2. Set memory source type:
   `usb_stat.ctrl_trf_mem = <_ROM or _RAM>;`
3. Set the size of data to transfer:
   `wCount._word = <size of data to transfer>;`
4. The data destination address is handled automatically by `usbctrltrf.c`; the destination is the `CtrlTrfData` buffer space defined in `usbmmap.c`

When writing a request handler to receive control transfer data from a host over multiple transactions, make sure to do the following:

1. Set data destination:
   `pDst.bRam = <RAM buffer location>` (Note that the memory destination type has to always be `_RAM`)
2. Initialize the receive counter, `wCount._word`, to zero
3. The data source address is handled automatically by `usbctrltrf.c`; the destination is the `CtrlTrfData` buffer space defined in `usbmmap.c`

Once the status stage is complete, the control transfer session owner (determined by the MUID value) can process the received data. The data destination is defined in the setup stage, and the length of received data is stored in `wCount._word`.

4.3.2.4 USB9.C

This file handles standard USB requests as defined in Chapter 9 of the USB specification, and handles the enumeration process. The function of interest in this file is `USBStdSetCfgHandler`(), which users must modify to match the application's configuration setup and behavior.

4.3.3 Compile Time Validation

A set of rules can be set to check the correctness of the USB variables. To enable compile time validation, add the following line in one of your project files:

```
#include "system\usb\usb_compile_time_validation.h"
```

This file validates the value of EP0_BUFF_SIZE, which must be a legal buffer size (8, 16, 32 or 64 bytes). Additional validation rules can be added by users, as required.

Chapter 5. Reconfiguring the PICDEM™ FS USB Hardware

5.1 HIGHLIGHTS

This chapter covers the following:

- Configuring the Demonstration Board Options
- Restoring the PICDEM™ FS USB Firmware

5.2 CONFIGURING THE DEMONSTRATION BOARD OPTIONS

The PICDEM FS USB board can be configured to enable or disable its various hardware features. A total of 16 jumper locations are controlled in various places around the board. As shipped from the factory, all of the locations are bridged by circuit traces, and all of the features are enabled. To change this, the user will need to cut the traces, and install pins and a block jumper. Afterwards, the features can be enabled or disabled easily by installing or removing the jumper.

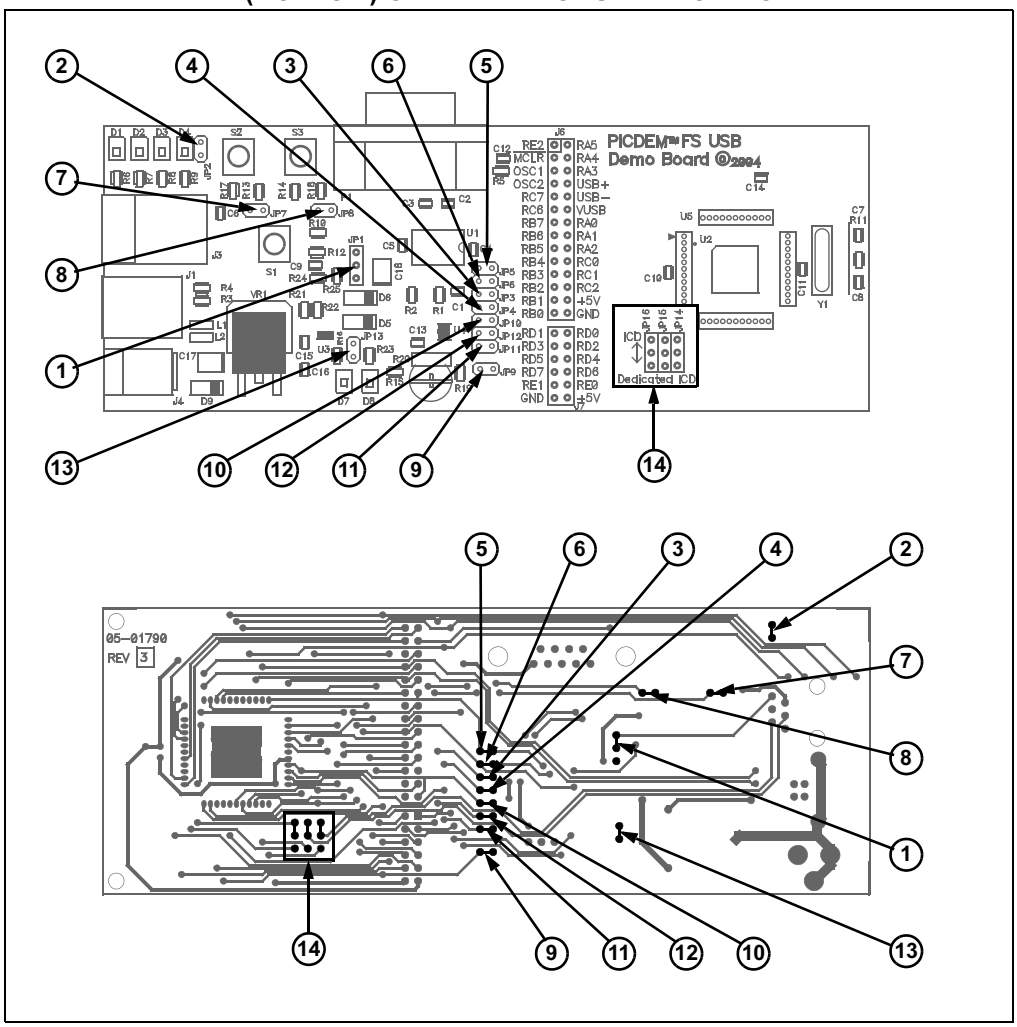
In some instances, a single function (such as the digital temperature sensor) is connected to the rest of the board through more than one jumper. This allows selective tailoring of the controller's I/O ports to any application that the user may develop. Specific cases are discussed in the following sections.

The functions of the jumpers are listed in Table 5-1; their locations are shown in Figure 5-1.

TABLE 5-1: JUMPER DESCRIPTION

Number	Board ID(s)	Type	Function
1	JP1	2-way	Selects user-controlled Reset (S1) or microcontroller disable for external emulation
2	JP2	Bridge	Status LED bank (D1 through D4)
3	JP3	Bridge	USART Receive (microcontroller's perspective)
4	JP4	Bridge	USART Flow Control (RTS)
5	JP5	Bridge	USART Transmit (microcontroller's perspective)
6	JP6	Bridge	USART Flow Control (CTS)
7	JP7	Bridge	S2 (User-defined switch 1)
8	JP8	Bridge	S3 (User-defined switch 2)
9	JP9	Bridge	R20 (potentiometer)
10	JP10	Bridge	U4 (temperature sensor)
11	JP11	Bridge	
12	JP12	Bridge	
13	JP13	Bridge	D7 (USB bus power LED)
14	JP14, JP15, JP16	2-way	ICD Port Configuration

FIGURE 5-1: JUMPER LOCATIONS (TOP) AND TRACE LOCATIONS (BOTTOM) ON THE DEMONSTRATION BOARD



5.2.1 Serial Port Configuration

The RS-232 serial port on the demonstration board incorporates hardware flow control with CTS and RTS for applications that require these control signals.This makes the port useful for developing a wide range of serial-to-USB translators. For those applications that do not require hardware flow control, or use software flow control instead, the hardware flow control feature can be selectively disabled by cutting the traces at JP4 and JP6.

Note that the CTS signal and the self-power sense signal share a single controller port (RA2). These two functions cannot be used concurrently. The CTS is an output signal from the microcontroller, while the self-power sense signal is an input signal. The port must be configured either as an input or an output to use each mode. JP6 is provided to explicitly disable the CTS signal.

5.2.2 Disabling the Temperature Sensor

For the TC77 temperature sensor, each of the three lines that it uses to communicate with the controller are bridged with a separate jumper. Removing all three jumpers (JP10, JP11 and JP12) disables the sensor's function, and makes all three controller ports available to the user.

5.2.3 ICSP/ICD Configuration

Using the MPLAB IDE and MPLAB ICD 2, users can reprogram the board's microcontroller using In-Circuit Serial Programming™ (ICSP), and debug firmware code using In-Circuit Debugger. The RJ-11 receptacle (J3), also known as the ICSP/ICD connector, is the standard MPLAB ICD 2 interface found on most Microchip development boards.

Most PIC18 microcontroller only have one legacy ICSP/ICD port, which shares I/O pins RB6 and RB7. When used, the legacy port prevents applications from using these pins as normal I/O ports. The 44-pin TQFP version of the PIC18F4550 is the only variant of the full-speed USB family of devices to have a second ICSP/ICD port on pins not used for I/O. This dedicated port allows RB6 and RB7 to be utilized by the user's application.

The PICDEM FS USB board can be configured to work with either the legacy or dedicated port by using jumpers JP14 through JP16. As shipped, the demonstration board is hardware-configured for the legacy port. To change this, the user must cut the traces indicated in Figure 5-1 (item 14), and install pins and box jumpers. When installed, the ICD connector is configured by the jumpers as shown in Table 5-2.

In addition to setting the jumpers, the ICPRT configuration bit in CONFIG4L must also be properly set. To enable the dedicated ICSP/ICD port, ICPRT must be set (= `1`).

When the board is configured to use the dedicated ICSP/ICD port, the reset switch S1 will no longer function. The ICRST pin acts only as an active-high reset port, which works when the MPLAB ICD 2 sends a High-Voltage programming signal, (V_{IHH}). The $\overline{MCLR}$ pin acts as both active-low and active-high reset ports. Thus, the reset switch S1 only works when connected to the $\overline{MCLR}$ pin.

Additional information regarding the dedicated ICSP/ICD port can be found in the "Special Features" section of the device data sheet (DS39632).

TABLE 5-2: JUMPER CONFIGURATION FOR THE ICSP/ICD CONNECTOR

ICD Connector Configuration	JP14	JP15	J16
Legacy ICD (CHP_$\overline{MCLR}$/RB7/RB6)	Pins 1-2	Pins 1-2	Pins 1-2
Dedicated Port (ICRST/ICPGD/ICPGC)	Pins 2-3	Pins 2-3	Pins 2-3

Note 1: For JP14 through JP16, pin 1 is the location closest to the microcontroller.

2: Before using the ICD connector for programming or debugging, verify that all three jumpers are set correctly. Failure to do so may result in programming or debugger failure.

5.2.4 Adding In-Circuit Emulation Capability

In addition to the on-board PIC18F4550 microcontroller, the PICDEM FS USB demonstration board can also be configured to work with hardware emulators such as the MPLAB ICE 2000 and MPLAB ICE 4000. To do this, the user must install a 44-pin riser in the area designated U5 on the board. This is comprised of four 11-pin pads arranged in a square around the microcontroller. A device adapter (Microchip part number DVA18PQ440) will also be required for either of the MPLAB ICE devices.

Before using an external emulator, users must also configure the board to disable the existing controller. This is done by installing and configuring jumper JP1. In its default configuration with the upper pins connected, JP1 assigns device reset control to S1. With the lower pins connected, JP1 grounds the MCLR pin of the PIC18F4550, holding it in permanent reset. In this state, any external controller or emulator connected to the riser will control the demonstration board.

5.3 RESTORING THE PICDEM FS USB FIRMWARE

As shipped from the factory, the microcontroller on the PICDEM FS USB board is pre-programmed with the firmware required to interact with the Demo Tool application. This provides the code that makes the interactive Demo mode possible, and enables communication to establish the Bootload mode.

As users develop their own USB applications, it is likely that the controller will be re-programmed with new firmware. If the original firmware is replaced, the status LEDs and other interactive features of the board may no longer work as previously described. It is even possible that USB connectivity may become disabled. For those users developing USB bootloader applications, it is possible that a change of device configuration with firmware loaded through a bootloader may render the demonstration board inoperable. In either case, it will be necessary to reprogram the board using an in-circuit programmer, such as MPLAB ICD 2.

Should it ever become necessary to return the board to its original state, it will be necessary to restore the original firmware. The necessary files have been installed on the host PC with the rest of PICDEM FS USB software, in the directory `MCHPFSUSB\fw\_factory_hex\`. To reprogram the microcontroller directly via the ICD connector with the original firmware, use the hex file `picdemfsusb.hex`.

Users should follow the procedure appropriate for their device programmer and development environment.

Chapter 6. Troubleshooting

6.1 HIGHLIGHTS

This chapter discusses the following:

- Common issues with the PICDEM FS USB demonstration board , and how to solve them

6.2 COMMON PROBLEMS

1. The Power LED is not lit

Normally, the PICDEM FS USB board behaves by default as a bus-powered device. If it is connected to a functioning USB port, the board will power up and D7 will light. If this does not happen:

- Verify that the USB port on the host system is actually working by plugging a USB device that is known to be good into the port
- Verify that the cable is working by substituting a known functional cable

If a power supply is connected to the board, it will switch to Self Power mode automatically; D7 will not be lit, and D8 will light instead. If this does not happen:

- Verify that the power supply is plugged in, and the wall outlet has power. If battery connection is used, verify that the correct polarity is used.
- Check that voltage is available (9 VDC) at the power supply's plug. If an OEM power supply is not being used, check for appropriate voltage (9 VDC).
- Check that the regulated voltage (5 VDC) is available.

2. The appropriate Power LED is lit, but the system does not recognize the board

Check the USB cable for proper connections to the board and the computer. If necessary, verify the cable by swapping in another cable that is known to be good.

If the Demo Tool software is running, press and release S1to reset the board.

In Windows Device Manager (accessed through the System applet in the Control Panel), use the "Scan for Hardware Changes" function to attempt to force the system to detect the board.

It is possible that the only firmware programmed into the microcontroller is the Bootload firmware. In this case, enter Bootload mode by pressing and holding S2 while pressing and releasing S1. If the Bootload mode does not work, it is possible that the firmware has become corrupted. In this case, it will be necessary to reprogram the microcontroller with the original firmware through an external in-circuit programmer.

3. The board is functioning, but has stopped communicating with the Demo Tool.

In Demo mode, press and release S1, then re-select the board from the drop down menu and click on "Connect".

In Bootload mode, press and hold S2 while resetting the board (pressing and releasing S1), then re-select the board from the drop down menu.

4. After switching from Demo mode to Bootload mode in the Demo Tool, the board is not available for selection in the dropdown list.

The demonstration board 's default firmware behaves as two separate USB devices. Invoking the Bootloader firmware requires a special hardware signal, generated by holding S2 while resetting the board (pressing and releasing S1).

Similarly, the board will not be available automatically when switching from Bootload to Demo mode. In this case, it should only be necessary to reset the board (press and release S1).

5. On running Bootload mode for the first time, the host system tries to repeat the driver installation process.

Because the demo and bootloader firmware are essentially two separate pieces of code with individual Product IDs, the host system will see each as a separate device. Each device will require the USB driver to be installed separately for it. Allow the Windows Install Wizard to use the same `.inf` file used for the original driver installation.

Appendix A. PICDEM™ FS USB Board Technical Information

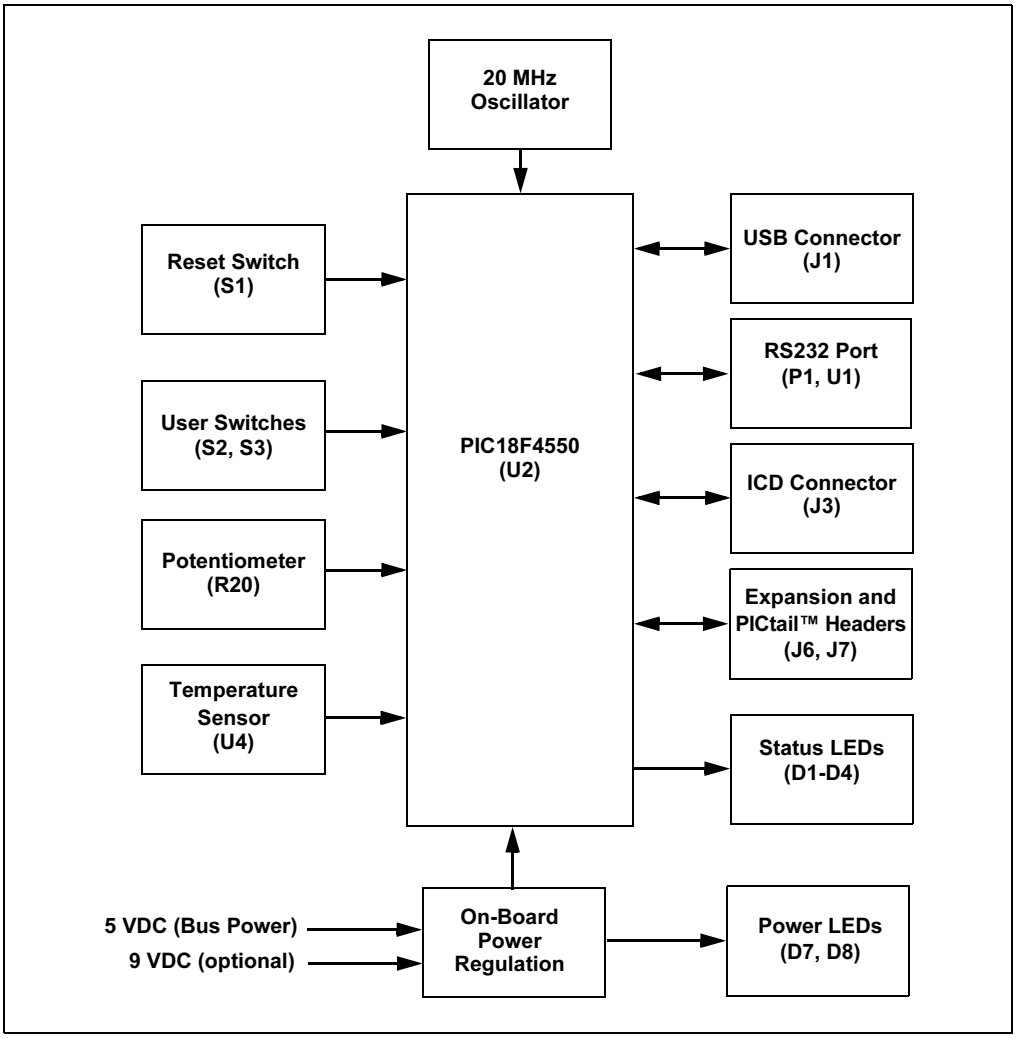
A.1 HIGHLIGHTS

This chapter will cover the following topics:

- PICDEM FS USB Block Diagram
- PICDEM FS USB Board Schematics

A.2 BLOCK DIAGRAM

FIGURE A-1: PICDEM FS USB DEMONSTRATION BOARD FUNCTIONAL BLOCK DIAGRAM



A.3 PICDEM FS USB BOARD SCHEMATICS

FIGURE A-2: BOARD SCHEMATIC, PART 1 (MICROCONTROLLER, VOLTAGE REGULATION AND ASSOCIATED PARTS)

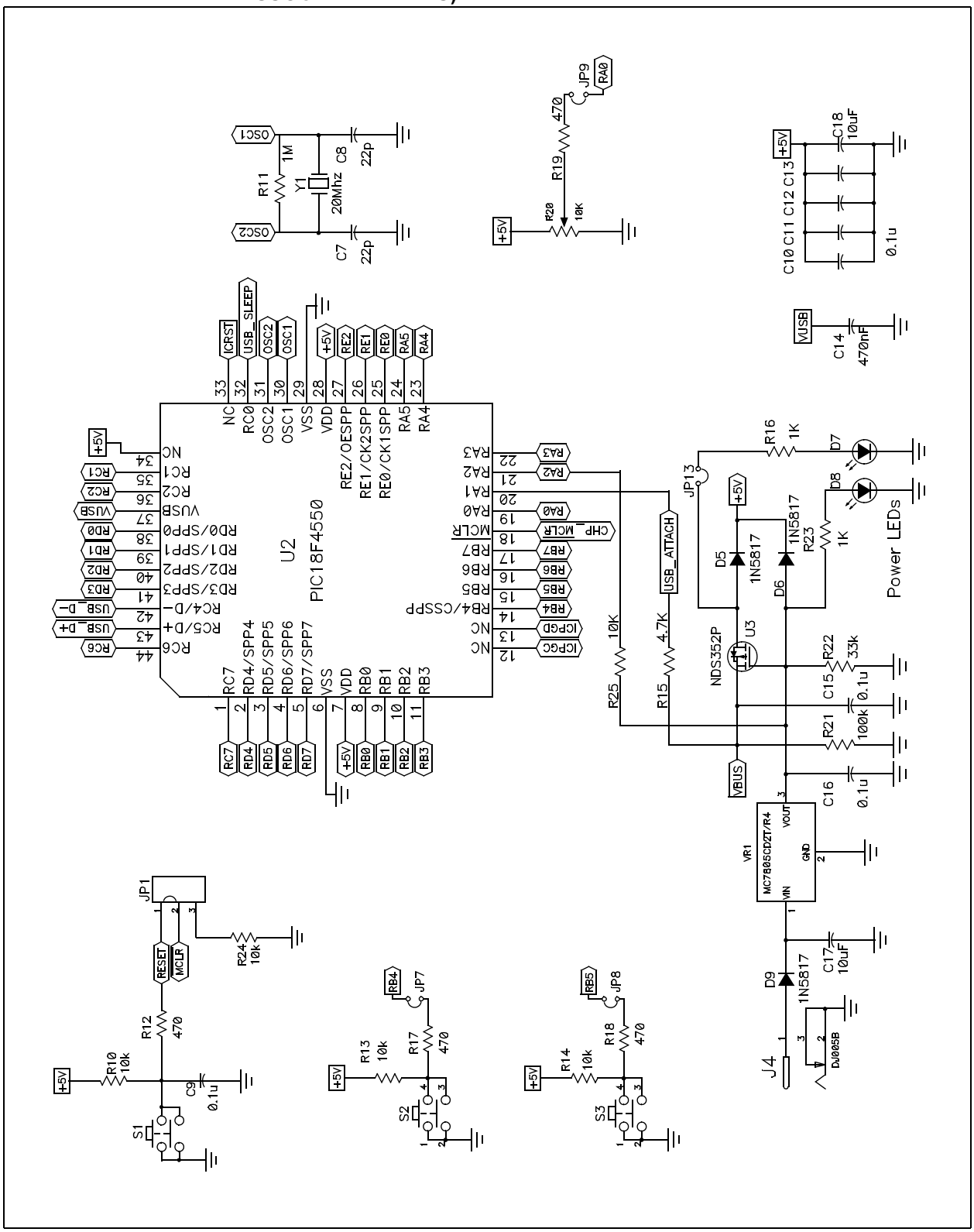


FIGURE A-3: BOARD SCHEMATIC, PART 2 (USART, USB INTERFACE, THERMAL SENSOR AND STATUS LEDS

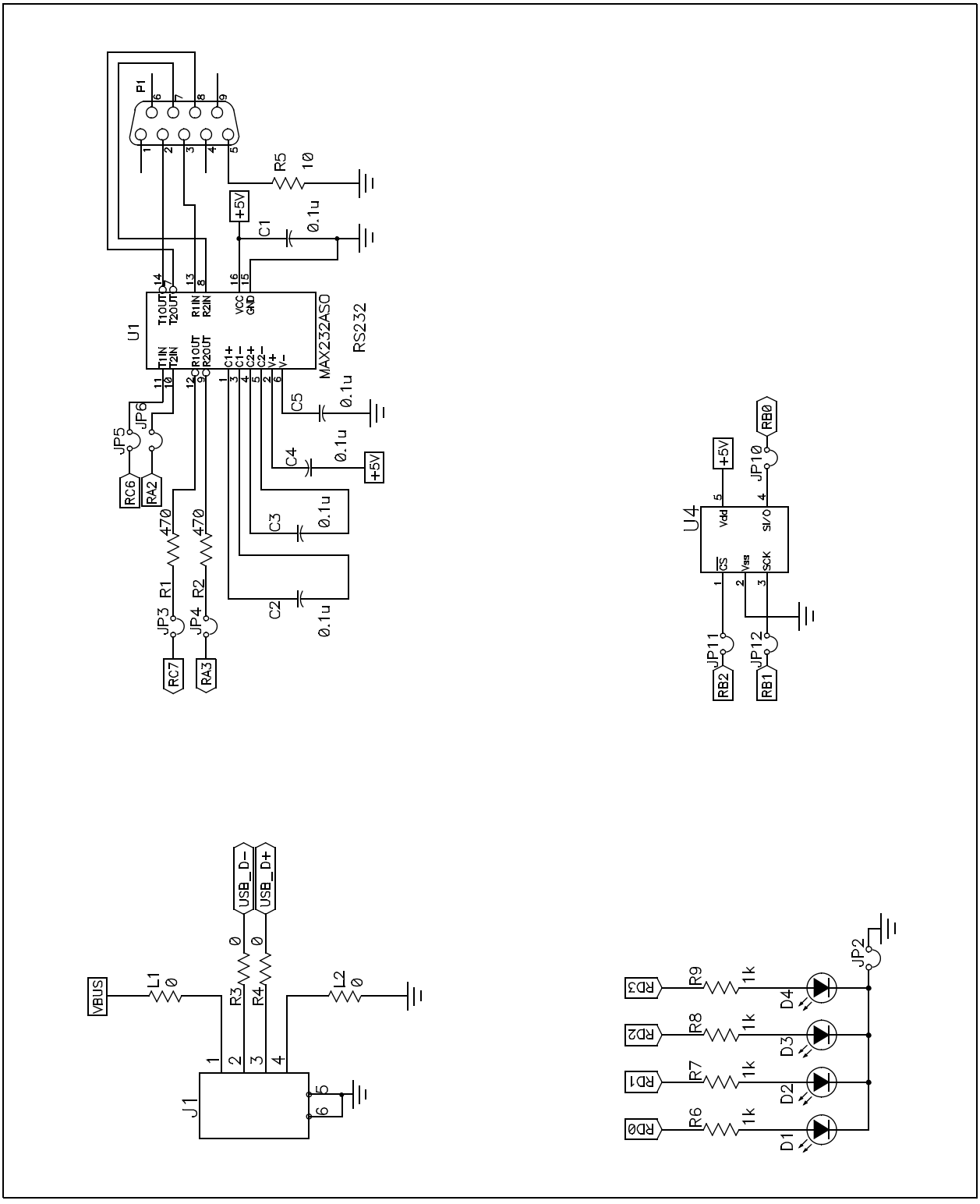


FIGURE A-4: BOARD SCHEMATIC, PART 3 (CONFIGURATION JUMPERS, PICkit HEADERS AND OPTIONAL MPLAB ICE HEADER)

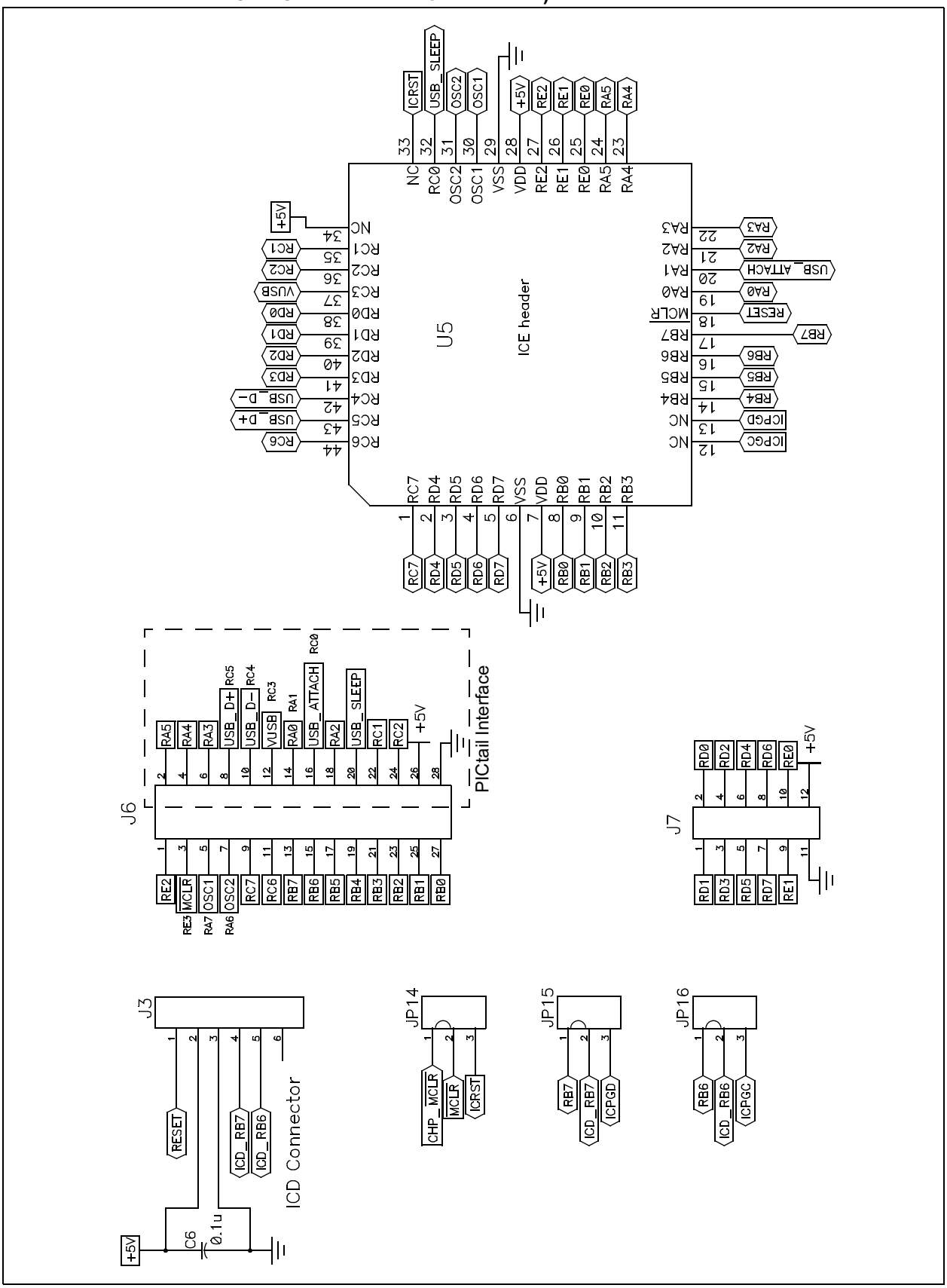


TABLE A-1: SIGNALS USED IN THE PICDEM FS USB SCHEMATIC

Signal Name	Function
CHP_$\overline{MCLR}$	Hardware Master Clear signal (direct connection to $\overline{MCLR}$ pin of microcontroller)
ICPGC, ICPGD, ICRST	Dedicated ICD port clock, data and chip reset signals
ICD_RB6, ICD_RB7	Legacy ICD port signals (PGC and PGD)
$\overline{MCLR}$	Hardware Master Clear signal
OSC1, OSC2	External oscillator input for the microcontroller
RESET	Generic chip reset signal (routable to $\overline{MCLR}$ or ICRST)
RXn	Bit n of microcontroller Port X
USB_D+, USB_D-	Differential USB signals
USB_ATTACH	Bus attachment signal generated by power supply
USB_SLEEP	Optional USB suspend signal from an external transceiver
VUSB	Regulated 3.3V V_{USB} supplied by microcontroller's on-chip regulator
VBUS	Bus power voltage supplied from USB cable

TABLE A-2: RS-232 INTERFACE PINOUT

RS-232 Pin	PIC18F4550 Pin	Pin Function
1	—	CD
2	RC6/TX	RX (Received Data, PC Perspective)
3	RC7/RX	TX (Transmit Data, PC Perspective)
4	—	DTR
5	Vss	GND
6	—	DSR
7	RA3	RTS (Request To Send)
8	RA2	CTS (Clear To Send)
9	—	RI

TABLE A-3: PERIPHERAL INPUT AND OUTPUT PIN ASSIGNMENTS

PIC18F4550 Pin	Function
RD0	LED D1 Output
RD1	LED D2 Output
RD2	LED D3 Output
RD3	LED D4 Output
RA0	R20 (potentiometer) Input
SDI/RB0	TC77 (SPI Data Input)
SCK/RB1	TC77 (SPI Clock Output)
RB2	TC77 (Chip Select Output)

NOTES:

Appendix B. PICDEM™ FS USB Starter Kit CD

B.1 HIGHLIGHTS

This chapter summarizes the contents of the PICDEM FS USB Starter CD.

B.2 WHAT'S ON THE CD

The PICDEM FS USB Starter CD includes the following software tools and support documentation.

PICDEM FS USB Software Package

The entire utility is contained in the self-extracting installer file `MCHPFSUSB_Setup.exe`. Executing this file will install the Demo Tool application and USB drivers described in **Chapter 3. "Using the Demo Tool Application"**, plus the documentation described below. It will also install demonstration and tutorial applications for several USB classes.

License Agreement

This is the full text of Microchip USB Software licensing agreement that appears at the beginning of the software installation process. It is provided in Adobe® Acrobat® format.

Documentation

This folder contains all relevant documentation to the PIC18F2455/2550/4455/4550 family of microcontrollers and the PICDEM FS USB demonstration board. Documents are distributed among these folders:

- **Applications Notes:** the most current Microchip application notes for the PIC18F2455/2550/4455/4550 family.
- **Data Sheets:** the latest revision of the PIC18F2455/2550/4455/4550 device data sheet
- **Errata:** the latest silicon errata and data sheet clarifications for the PIC18F2455/2550/4455/4550 family
- **Users Guide:** This document, provided in Adobe Acrobat format.

Tools

This folder contains links to the latest versions of Microchip's MPLAB IDE and C18 compiler. Both are available for download at the Microchip web site, **www.microchip.com**.

NOTES:

Index

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

Boston
Westford, MA
Tel: 978-692-3848
Fax: 978-692-3821

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Qingdao
Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

Japan - Kanagawa
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Taiwan - Hsinchu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Ballerup
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Massy
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Ismaning
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

England - Berkshire
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/20/04