

PICDEM™ USB
User's Guide

© 2001 Microchip Technology Inc.

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products 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 PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be 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 product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, KEELOQ, SEEVAL, MPLAB and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Total Endurance, ICSP, In-Circuit Serial Programming, FilterLab, MXDEV, microID, *Flex*ROM, *fuzzy*LAB, MPASM, MPLINK, MPLIB, PICC, PICDEM, PICDEM.net, ICEPIC, Migratable Memory, FanSense, ECONOMONITOR, Select Mode, dsPIC, rfPIC and microPort are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Term Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2001, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

© 2001 Microchip Technology Inc.

Table of Contents

Preface

Introduction 1
Highlights 1
About This Guide 1
Warranty Registration 3
Recommended Reading 3
The Microchip Internet Web Site 4
Development Systems Customer Notification Service 5
Customer Support 7

Chapter 1. Getting Started with the PICDEM™ USB

1.1 Introduction 9
1.2 Highlights 9
1.3 Unpacking your PICDEM™ USB Kit 9
1.4 Running the Default Demonstration 10
1.5 Branching Out on Your Own 10

Chapter 2. USB Demonstration Code

2.1 Gameport - USB Translator 13
2.2 PS/2 Keyboard/Mouse - USB Translator 17
2.3 Combination Gameport/PS/2/Mouse - USB Translator 23
2.4 Multi-Function LCD Text Display Example 25

Chapter 3. PICDEM™ USB Hardware

3.1 Oscillator Support 31
3.2 Connector Pinout 32
3.3 Buttons and Jumpers 36
3.4 Power 37

© 2001 Microchip Technology Inc.

Chapter 4. Chapter 9 USB Firmware

4.1 Introducing the USB Software Interface 39
4.2 Integrating USB Into Your Application 39
4.3 Interrupt Structure Concerns 40
4.4 File Packaging 41
4.5 Function Call Reference 42
4.6 Behind the Scenes 44
4.7 Examples 45
4.8 Multiple Configuration or Report Descriptors 46
4.9 Optimizing the Firmware 47
4.10 Cursor Demonstration 48

Chapter 5. Troubleshooting

5.1 Introduction 51
5.2 Highlights 51
5.3 FAQ 51

Appendix A. PICDEM™ USB Schematics

Introduction 53
Highlights 53
Schematics 54

Appendix B. PS/2 Lookup Tables

Introduction 61
Scan Codes 61
Command Codes 62

Glossary

Introduction 65
Highlights 65
PICDEM™ USB Terms 65

Index 73

Worldwide Sales and Service 76

© 2001 Microchip Technology Inc.

Preface

Introduction

This chapter contains general information about this manual and contacting customer support.

Highlights

Topics covered in this chapter:

- About this Guide
- Warranty Registration
- Recommended Reading
- The Microchip Internet Web Site
- Development Systems Customer Notification Service
- Customer Support

About This Guide

Document Layout

This document describes how to use PICDEM™ USB to attach a new peripheral to a PC. The manual layout is as follows:

- **Chapter 1: Getting Started with the PICDEM™ USB** – What PICDEM™ USB is and how it works.
- **Chapter 2: PICDEM™ USB Demonstration Code** – Provides USB demonstration code and information on Gameport™, PS/2® Keyboard/ Mouse, Combination Gameport/PS/2, and LCD Demo.
- **Chapter 3: PICDEM™ USB Hardware** – Contains oscillator support, connector pinout, buttons and jumpers, and power information.
- **Chapter 4: Chapter 9 USB Firmware** – Describes the USB software interface.
- **Chapter 5: Troubleshooting** – Provides solutions to common problems users may experience with PICDEM™ USB. It also includes FAQ on Hardware, PC/Windows® and Macintosh® concerning the PICDEM™ USB.
- **Appendix A: Schematics** – Provides the schematics for the PICDEM™ USB.
- **Appendix B: PS/2 Lookup Tables** – Provides scan and command code tables for easy reference.

© 2001 Microchip Technology Inc.

- **Index** – Provides a cross-reference listing of terms, features, and sections of this document.
- **PICDEM™ USB Worldwide Sales and Service** – Lists Microchip sales and service locations and telephone numbers worldwide.

Conventions Used in this Guide

This manual uses the following documentation conventions:

Documentation Conventions

Description	Represents	Examples
Code (Courier font):		
Plain characters	Sample code, file names and paths	`#define START` `c:\autoexec.bat`
Angle brackets: < >	Variables	`<label>, <exp>`
Square brackets [ ]	Optional arguments	`MPASMWIN [main.asm]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments, an OR selection	`errorlevel {0|1}`
Lower case characters in quotes	Type of data	`"filename"`
Ellipses...	Used to imply (but not show) additional text that is not relevant to the example	`list ["list_option.., "list_option"]`
0xnnn	A hexadecimal number where 'n' is a hexadecimal digit	`0xFFFF, 0x007A`
Italic characters	A variable argument; it can be either a type of data (in lower case characters) or a specific example (in uppercase characters)	`char isascii (char, ch);`
Interface (Arial font):		
Underlined, italic text with right arrow	A menu selection from the menu bar	*File > Save*
Bold characters	A window or dialog button to click	**OK**, **Cancel**
Characters in angle brackets < >	A key on the keyboard	<Tab>, <Ctrl-C>
Documents (Arial font):		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*

© 2001 Microchip Technology Inc.

Updates

All documentation becomes dated, and this user's guide is no exception. Since the MPLAB IDE, PICDEM™ USB and other Microchip tools are constantly evolving to meet customer needs, some MPLAB® dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site at http://www.microchip.com to obtain the latest documentation available.

Warranty Registration

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in your Warranty Registration Card entitles you 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 PICDEM™ USB. The data sheets contain current information on programming the specific microcontroller devices.

`README.USB`

For the latest information on using PICDEM™ USB, read the `README.USB` file (ASCII text file) included with the PICDEM™ USB software. The `README.USB` file contains update information that may not be included in this document.

MPLAB® IDE User's Guide (DS51025)

Comprehensive guide that describes installation and features of Microchip's MPLAB Integrated Development Environment (IDE), as well as the editor and simulator functions in the MPLAB environment.

MPASM™ User's Guide with MPLINK™ Linker and MPLIB™ Librarian (DS33014)

Describes how to use Microchip Universal PICmicro® Microcontroller Assembler (MPASM™), Linker (MPLINK™), and Librarian (MPLIB™).

Technical Library CD-ROM (DS00161)

This CD-ROM contains comprehensive data sheets for Microchip PICmicro® MCU devices available at the time of print. To obtain this disk, contact the nearest Microchip Sales and Service location (see back page), or download individual data sheet files from the Microchip web site (http://www.microchip.com).

Embedded Control Handbook (DS00711)

This handbook consists of several documents that contain a wealth of information about microcontroller applications. To obtain these documents, contact the nearest Microchip Sales and Service location (see back page).

The application notes described in these manuals are also obtainable from Microchip Sales and Service locations or from the Microchip web site (http://www.microchip.com).

PICmicro™ Mid-Range MCU Family Reference Manual (DS33023)

This manual explains the general details and operation of the MCU family architecture and peripheral modules. It is designed to complement the device data sheets.

Microsoft® Windows® Manuals

This manual assumes that users are familiar with Microsoft Windows operating system. Many excellent references exist for this software program, and should be consulted for general operation of Windows.

USB Complete

This book is a good introduction to the USB interface and how to use it. It was written by Jan Axelson.

PIC16C745/765 Data Book (DS41124)

This book contains everything you ever wanted to know about the PIC16C745/765 and more.

USB IF

The USB IF can be downloaded from www.usb.org, which has all the specifications for the USB interface. It is a valuable tool. Be sure to download the USBCheck PC tools and register on the USB mailing list.

Microsoft DDK

If you are developing PC drivers for Windows, don't forget to get the driver development kit and the professional version of Visual Studio®.

Apple® USB Software Developer Kit (SDK)

If you are going to develop Mac drivers, go to `www.developer.apple.com/SDK/index.html` and download the SDK and register on the USB mailing list.

The Microchip Internet Web Site

Microchip provides online support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape® Communicator or Microsoft® Internet Explorer®. Files are also available for FTP download from our FTP site.

© 2001 Microchip Technology Inc.

Connecting to the Microchip Internet Website

The Microchip web site is available by using your favorite Internet browser to attach to:

http://www.microchip.com

The file transfer site is available by using an FTP program/client to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles, and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for Products, Development Systems, Technical Information and more
- Listing of Seminars and Events

Development Systems Customer Notification Service

Microchip started the customer notification service to help our customers keep current on Microchip products with the least amount of effort. Once you subscribe to one of our list servers, you will receive email notification whenever we change, update, revise or have errata related to that product family or development tool. See the Microchip web page at http://www.microchip.com for other Microchip list servers.

The Development Systems list names are:

- Compilers
- Emulators
- Programmers
- MPLAB IDE
- Otools (other tools)

Once you have determined the names of the lists that you are interested in, you can subscribe by sending a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe <listname> yourname`

Here is an example:

`subscribe programmers John Doe`

To UNSUBSCRIBE from these lists, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`unsubscribe <listname> yourname`

Here is an example:

`unsubscribe programmers John Doe`

The following sections provide descriptions of the available Development Systems lists.

Compilers

The latest information on Microchip C compilers, Linkers and Assemblers. These include MPLAB® C17, MPLAB® C18, MPLINK™ Object Linker (as well as MPLIB™ Object Librarian), and MPASM™ Assembler.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe compilers yourname`

Emulators

The latest information on Microchip In-Circuit Emulators. These include MPLAB® ICE and PICMASTER® Emulator.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe emulators yourname`

Programmers

The latest information on Microchip PICmicro device programmers. These include PRO MATE® II and PICDEM™ USB.

© 2001 Microchip Technology Inc.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe programmers yourname`

MPLAB IDE

The latest information on Microchip MPLAB IDE, the Windows Integrated Development Environment for development systems tools. This list is focused on MPLAB IDE, MPSIM™ Simulator, MPLAB's Project Manager and general editing and debugging features. For specific information on MPLAB compilers, linkers and assemblers, subscribe to the COMPILERS list. For specific information on MPLAB emulators, subscribe to the EMULATORS list. For specific information on MPLAB device programmers, please subscribe to the PROGRAMMERS list.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe mplab yourname`

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Corporate Applications Engineer (CAE)
- Hotline

Customers should call their distributor, representative, or field application engineer (FAE) for support. Local sales offices are also available to help customers. See the back cover for a listing of sales offices and locations.

Corporate applications engineers (CAEs) may be contacted at (480) 792-7627.

In addition, there is a Systems Information and Upgrade Line. This line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.

The Hotline Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-480-792-7302 for the rest of the world.

NOTES:

© 2001 Microchip Technology Inc.

Chapter 1. Getting Started with the PICDEM™ USB

1.1 Introduction

The Universal Serial Bus (USB) has become the most accepted way for any new peripheral to be attached to a PC. The interface is fully supported by all major computer manufacturers and most operating systems. Since this interface has become more accepted, it has moved out of the consumer market. It is starting to find its way into data acquisition and industrial markets. Within these markets, a large number of PICmicro solutions are looking for a migration path to USB. By providing a USB derivative of the classic PIC16C72/74 devices, Microchip encourages you to move your applications into the new world of USB; and as an addition bonus, the examples provide an opportunity to play games.

1.2 Highlights

The topics covered in this chapter are:

- Unpacking your PICDEM™ USB
- Running the Default Demonstration
- Branching out on your own

1.3 Unpacking your PICDEM™ USB Kit

1.3.1 Supplied Items

The items contained in your PICDEM™ USB box are:

- Microchip USB CD-ROM containing USB support documentation
- PICDEM™ USB Circuit Board with a PIC16C765 installed
- CD-ROM containing MPLAB IDE
- 3 ft. USB A-B cable
- Small box containing a windowed PIC16C745 and PIC16C765

1.3.2 Required Items

The items required are:

- PC for running MPLAB IDE
- Visual Basic and/or Visual C++ to modify the PC examples
- PC with USB running Windows® 98 or newer (for the PC examples)
- Macintosh with USB running MacOS X 10.0 or newer (for the Macintosh examples). The HID examples work with MacOS 8.6 or newer.
- Apple® Project Builder to modify the Macintosh code examples

© 2001 Microchip Technology Inc.

- PICSTART® Plus or PRO MATE® II to program the devices
- UV chip eraser to clean the mistakes
- Copy of Apple's USB DDK so you can use the USB bus monitoring tools on the Macintosh (http://developer.apple.com/hardware/usb/)
- Copy of the USB-IF PC tools (www.usb.org)

1.3.3 Suggested Items

The items suggested are:

- USB protocol analyzer such as CATC
- Membership in the USB-IF, Inc. (www.usb.org)
- MPLAB® ICE 2000

1.4 Running the Default Demonstration

If you have a PC/Macintosh with USB, attach your PICDEM™ USB with the supplied cable. The LED's should quickly blink as the PICDEM™ USB identifies itself. The EP1 ACT light should start to flicker steadily, and the mouse cursor on your computer should start to move in a circle.

Unplug the USB cable and plug a PS/2 mouse into the PS/2 connector. Re-attach the USB cable. You will notice the mouse cursor is no longer moving in a circle, but is responding to the mouse. Unplug the USB cable and plug in a PS/2 keyboard. Re-attach the USB cable. The LED's flicker and the keyboard is functioning as a USB keyboard.

Regardless of what PS/2 device is attached (or not attached), the EP2 ACT light will be flashing. This is due to the PICDEM™ USB constantly streaming gameport data to the host, regardless of whether a gaming device is plugged in or not. You can test the gameport by using the gamepad specified in Section 2.1: Gameport - USB Translator.

This example takes advantage of the existing human interface device (HID) code in the host's operating system. For a more complex example showing host driver code, you will have to consult the LCD Demo example.

1.5 Branching Out on Your Own

1.5.1 Developing New USB Applications

The following steps are recommended to develop most new USB applications.

- Describe the application.
- Create the descriptors.
- Debug the report descriptor with dummy data.
- Develop the rest of the application.

© 2001 Microchip Technology Inc.

By following these steps, the hardest part of the development can be completed right away on known good hardware (the PICDEM™ USB). After the application is communicating to the PC correctly, the application specific hardware and software can be developed.

1.5.1.1 Describing the Application

When you start developing your application, make sure that your data requirements fit the USB specification. A low speed device is limited to 2 channels of communication (end points), with each channel limited to 800 bytes per second. The most common mistake is to assume that the entire 1.5 Mbs is available for your application.

1.5.1.2 Creating the Descriptors

The most difficult part of any USB application is determining what the device descriptors should be. Every USB device communicates its requirements to the host through a process called enumeration. During enumeration, the device descriptors are transferred to the host and the host assigns a unique address to the device. The descriptors are described in detail in Chapter 9 of the USB 2.0 specification. Bundled with the USB tools CD, a descriptor tool is provided to assist you in creating your own descriptors.

1.5.1.3 Debugging the Report Descriptor

The report descriptor allows HID devices to communicate to the host. The report descriptor communicates the exact packet format of your data. This is where the PC determines how large your packets will be. Report descriptors range from the very specific (a multi-function joystick) to the very generic (a specialized communications device for your application). Tools are available to assist you in creating your report descriptors. The descriptor tool, which is bundled in your kit, will have report descriptor capabilities with a future revision. After the report descriptor is written make sure that it is working with the PC by using PC analysis tools. These are available from the USB-IF web site for the PC, and Apple Computer for Macintosh machines. Use simple counters and other dummy data to test report descriptor traffic. After the communications link is working, it will be much easier to develop the rest of the application.

1.5.1.4 Developing the Rest of the Application

The next step is to add your specific hardware and software. Carefully study the schematic in this guide and use the same circuitry for the USB connections. The circuitry will never change for this device. When your hardware is built, you can be confident that your communications code will work because it was developed on the development system with the same communications circuitry.

NOTES:

© 2001 Microchip Technology Inc.

Chapter 2. USB Demonstration Code

A variety of examples have been provided on the CD-ROM to speed you towards a successful start with USB. These examples range from the very basic gameport translator, to a very specialized LCD display. The gameport and PS/2 examples are excellent tools to use, in order to become more familiar with report descriptors and basic USB communications. These basic examples are extremely useful because the OS vendor has already written generic Human Interface Device (HID) drivers. The LCD display example does not have a generic PC driver. In this example, PC software has been provided to demonstrate how you can develop your own PC applications to interface to your device. Example code is provided for PC and Macintosh.

2.1 Gameport - USB Translator

2.1.1 Introduction

The gameport to USB translator is a simple example that reads a PC gameport and reports the information over USB. Since peripherals for gameports come in all shapes and sizes, it is important to identify exactly what peripheral is used, in this example -- the Dexxa® 8-button gamepad. Other gamepads can be used, however, some change to the firmware functions may be required. The example code will enumerate as a gamepad with 2 axis and 6 buttons. PORTA on the PICmicro MCU is used to read the analog voltages from resistors in the direction pad (D-pad) and two of the buttons, while PORTD reads the switches for the four remaining buttons. Refer to Figure 2.1.

© 2001 Microchip Technology Inc.

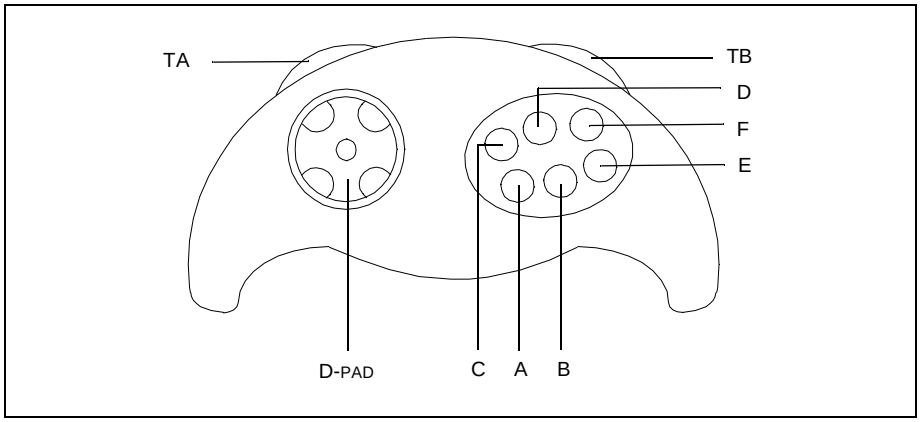


Figure 2.1: Dexxa® Gamepad

Note: Although the Dexxa® gamepad has 8 buttons, the code enumerates as a 6-button gamepad. This is due to the gamepad having two "special" buttons, that rapidly fire two of the other button outputs.

2.1.2 About the PC Gameport

The PC gameport was designed to support 2 joysticks. Each joystick was intended to have 2 axis and 2 buttons. The gameport was later extended to support a MIDI serial interface. The gameport hardware supplied with the PICDEM™ USB does not support the MIDI interface, so those pins have been disconnected. Two joysticks are supported, but to simplify the example, only one joystick is used.

2.1.3 Hardware Implementation

The hardware is supplied on the PICDEM™ USB circuit board. The board is wired with PORTA<3:0> and PORTD<3:0>, connected to the DB-15 connector of the gameport. PORTD<3:0> are the digital inputs for buttons A-D. PORTA<0> and PORTA<1> are the X-Y inputs for the D-pad on the gamepad. PORTA<2> and PORTA<3> are the inputs for buttons E and F (see Appendix A for the gamepad to PICDEM™ USB schematic). All of the analog pins have a series resistor, a resistor, and capacitor tied to ground. The reason for this comes from the way analog pins were originally read by the PC. The PC would clear a capacitor tied to ground at its end and then time how long it took the capacitor to charge up. The capacitor would charge at a rate proportional to the resistance of a pot, varied by one axis of the joystick, for instance. This is legacy technology and is not needed when using a PICmicro MCU with an analog-to-digital converter. As a result, the before mentioned circuitry was put into place in order to obtain an analog output from the D-pad of the gamepad and buttons E and F. Another point of confusion develops from viewing the analog output of the gamepad with this circuitry in place. The

© 2001 Microchip Technology Inc.

voltage level is not symmetrical around 2.5 volts. You would expect the center position of the D-pad's x-axis, for instance, to output 2.5 volts, the left position to output 0 volts and the right position to output 5 volts. Ideally it would, but the circuitry in the gamepad is very simple; besides, this symmetry is not needed. All that is needed is a lower voltage input into the PICmicro MCU when the pad is pressed one direction from center, and a higher voltage input when the pad is pressed in the other direction from center. The circuitry found on the PICDEM™ USB board amplifies the differences in voltages between these three positions and therefore, gives three very distinct analog-to-digital readings.

Note 1: Buttons E and F are connected to analog pins because the gameport has four analog pins and four digital pins. Buttons A-D use up all the digital pins so the analog pins are the only inputs left over. Many joysticks have a multiplexer on button numbers greater than four, in order to achieve the same result using just the digital pins.

2: Be very careful when using the LCD interface and the gameport together because they share PORTD.

2.1.4 Gamepad Firmware

The firmware has four functions. Each function either returns one piece of information concerning the gameport status, or initiates the gameport registers. Refer to Table 2.1.

Table 2.1: Gameport Firmware

Function	Description
ReadXAxis	Returns a digital value for the x-axis in W
ReadYAxis	Returns a digital value for the y-axis in W
ReadButtons	Returns the state of the six buttons in W (bits 0-5 correspond to buttons A-F)
InitGameport	Initialize the system to use the Gameport

2.1.5 Gamepad Report Descriptor

The report descriptor used in this example is for a gamepad with 6 buttons. The minimum and maximum report values for the axis are set to 0 - 255. This is to be used with Windows. Some versions don't seem to handle a range of -127 to 127. The -127 value is translated to 255 and causes extreme movement to the right. The range selected works correctly on Windows and on Macintosh. Interestingly, the Macintosh performs a simple filter on the center of the range. It will filter changes of 1 count to prevent cursor jitter when the stick is centered. This caused problems with a minimum and maximum range of 0 - 2. When the range was extended to 0 - 4, it worked much better. In this example, the analog input from the D-pad will be converted to digital, filtered, and then sent to the PC via USB. Since this digital value can range from

0 - 255, the descriptor calls for a range of 0 - 255. The descriptor, shown below in HEX form, is in the gamepad descriptor file (usb_ch9.asm).

```
0x05, 0x01        USAGE_PAGE (Generic Desktop)
0x09, 0x05        USAGE (Game Pad)
0xA1, 0x01        COLLECTION (Application)
0x09, 0x01           USAGE (Pointer)
0xA1, 0x00           COLLECTION (Physical)
0x09, 0x30              USAGE (X)
0x09, 0x31              USAGE (Y)
0x15, 0x00              LOGICAL_MINIMUM (0)
0x26, 0xFF, 0x00        LOGICAL_MAXIMUM (255)
0x75, 0x08              REPORT_SIZE (8)
0x95, 0x02              REPORT_COUNT (2)
0x81, 0x02              INPUT (Data,Var,Abs)
0xC0                 END_COLLECTION
0x05, 0x09           USAGE_PAGE (Button)
0x19, 0x01           USAGE_MINIMUM (Button 1)
0x29, 0x06           USAGE_MAXIMUM (Button 6)
0x15, 0x00           LOGICAL_MINIMUM (0)
0x25, 0x01           LOGICAL_MAXIMUM (1)
0x75, 0x01           REPORT_SIZE (1)
0x95, 0x06           REPORT_COUNT (6)
0x81, 0x02           INPUT (Data,Var,Abs)
0x95, 0x02           REPORT_COUNT (2)
0x81, 0x03           INPUT (Constant,Var,Abs)
0xC0              END_COLLECTION
```

This report descriptor describes the packet format for the USB data. The data is filled from Least Significant Byte, Least Significant bit through to the Most Significant Byte, Most Significant bit. The first field found will be the first bit/byte. In the report descriptor above, the first data is 8 bits (the REPORT_SIZE is 8) and it is the X axis (the first USAGE of the physical collection is X). So the first byte on the bus will be the X axis value. The second byte will be the Y axis. The third byte will be button A in bit 0, followed by button B in bit 1, and so on. Because every USB transaction must be in whole number bytes, the data is padded by one constant report, 2-bits long.

2.1.6 Gameport Translation

Translating the bits from the physical hardware to the USB buffer is very simple. Because we set the logical minimum and maximum to be 0 to 255, it exactly matches the scaling of the analog-to-digital converter. So first, we convert the X and Y axis and store the values in the first two buffer locations. Secondly, we read the six buttons and store the values in the third buffer location in bits 0-5. Lastly, we inform the serial interface engine that data is available and wait for the host PC to come pick it up.

© 2001 Microchip Technology Inc.

2.1.7 Vendor/Product Identification

Besides the report information, the descriptors also contain manufacturing and product identification codes. Microchip has a registered Vendor ID with the USB IF forum, which identifies Microchip's Vendor ID as 0x04D8. You are allowed to use this ID for your own testing, but you may not ship any products with this code without written permission from Microchip. Microchip has defined product ID's for each demonstration code included in this kit. As additional demonstration devices are released, Microchip will ensure that no duplicate ID's are used.

2.2 PS/2 Keyboard/Mouse - USB Translator

2.2.1 Introduction

This is the demonstration firmware that is programmed into the PIC16C765 and installed on the PICDEM™ USB demonstration board. The PS/2 connector was added so PS/2 mice and keyboards could be translated to USB. Again, this is a straightforward application intended to provide practice with device descriptors. The PS/2 interface is a synchronous serial interface with different data protocols for keyboards and mice. Device descriptors in the PICmicro microcontroller allows the unit to report itself as either a keyboard or a mouse. When a mouse is attached to a PS/2 port, it identifies itself as a mouse. When the PICmicro MCU receives this identification, it will perform a soft detach from the USB bus and re-attach as a mouse. When the PICmicro MCU identifies the PS/2 device as a keyboard, it will perform a soft detach and re-attach as a keyboard. Using soft detach may be a useful feature in your application, so you can practice using it with this example.

Note: It may be necessary to attach 110 kOhm pull-down resistors on RC0 and RC1, in order for auto-detect to work without pressing MCLR between plugging and unplugging PS/2 devices.

2.2.2 About the PS/2 Port

IBM® originally developed the PS/2 port for use on its PS/2 family of computers. This port is a synchronous serial port clocked by the PS/2 device (keyboard/mouse). Generally, PC's have two PS/2 ports labeled as keyboard or mouse. The PICDEM™ USB only has one PS/2 port. Since the hardware is the same, either the keyboard or mouse can be used, simply by interpreting the data correctly.

2.2.3 Hardware Implementation

The PS/2 port is a 6-pin DIN which only uses 4 pins. The pins are used for power, ground, clock, and data. Power and ground pins are directly tied to V_{DD} and V_{SS}. If power management is desired, the power pins must be driven via switches from other I/O pins.The clock pin is connected to RC0, while the data pin is connected to RC1. The PS/2 device clocks the host even when it is receiving data. The data pin is used to send and receive data from the keyboard.

2.2.4 Data Format

The data is sent via PS/2 one byte at a time, regardless of direction, host-to-device, or vice versa. The data is as follows:

- First comes a START bit (always low), followed by a
- data byte (Least Significant bit to Most Significant bit), then by a
- parity bit (high for an even number of high bits in the data byte and low for an odd number), then by a
- STOP bit (always high)

In the case of host-to-device communication, the STOP bit is immediately followed by an ACK bit (low), which is sent by the device to the host. The bits are read on the falling edge of the clock for device-to-host communication and on the rising edge for host-to-device communication. In the IDLE state, the clock and data lines are held high by the device. See Figure 2.2 and Figure 2.3 for device-to-host and host-to-device communication, respectively.

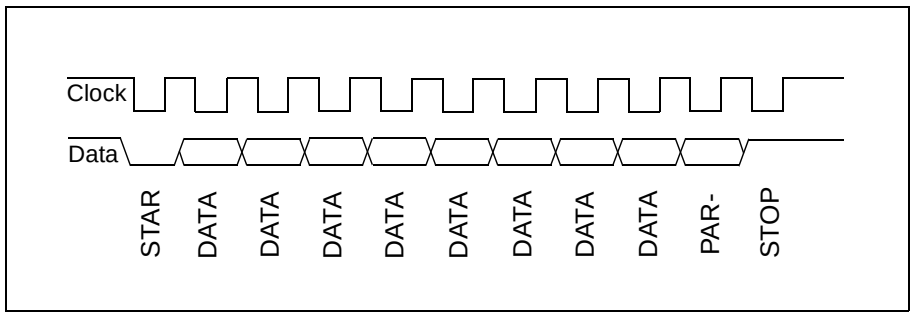


Figure 2.2: Device-to-Host Communication (Data bit Read on Falling Edge of Clock)

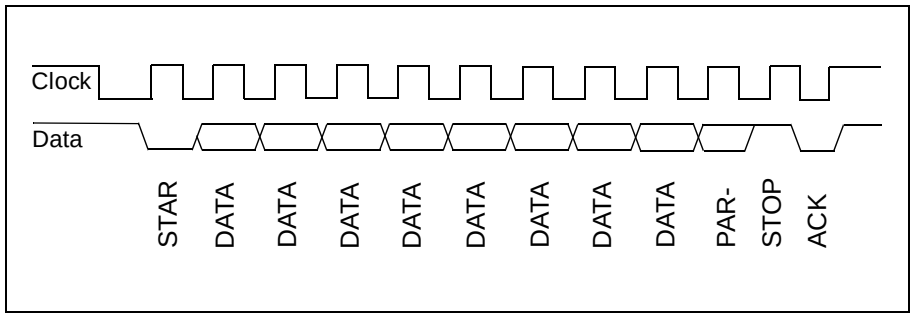


Figure 2.3: Host-to-Device Communication (Data bit Read on Rising Edge of Clock)

© 2001 Microchip Technology Inc.

2.2.5 Keyboard

The PS/2 keyboard data report format is summarized for every key in Appendix B. Make codes are the byte or bytes that the PS/2 keyboard sends to the host when a certain key is pressed. Break codes are the bytes that the PS/2 keyboard sends when the user releases a key. If the user does not release a specific key for several hundreds of a millisecond, the make code will be sent repeatedly until the user releases the key. At this point, the break code is sent. The Translation to USB, Section 2.2.11, details how the firmware converts PS/2 keycodes to USB keycodes.

Note: The PS/2 keycodes shown in Appendix B do not apply to all PS/2 keyboards. Several code sets have been used through the years. However, this code set is the most common.

2.2.6 Mouse

Table 2.2 details a typical PS/2 mouse data format.

Table 2.2: PS/2 Mouse Data Report Format

Byte	Bit	Description
3	7 6-1 0	MSB of Y Data Y Data LSB of Y Data
2	7 6-1 0	MSB of X Data X Data LSB of X Data
1	7 6 5 4 3 2 1 0	Y Data Overflow, 1 = overflow X Data Overflow, 1 = overflow Y Data Sign, 1 = negative X Data Sign, 1 = negative Reserved Reserved Right Button Status, 1 = pressed Left Button Status, 1 = pressed

2.2.7 Hardware Implementation

A PS/2 port is a 6-pin DIN, but only four pins are used (see Appendix A for pinout.) The pins are power, ground, clock and data. The clock pin is connected to RC0, while the data pin is connected to RC1. A PS/2 device clocks the host even when it is receiving data. By attaching the data pin to RC1, a START bit will interrupt the PICmicro MCU through a Capture/Compare/PWM (CCP) event. Refer to the PIC16C7XX data sheet for details on CCP. Power and ground are directly tied to V_{DD} and V_{SS}. If power management were desired, the power pins would be driven via switches from other I/O pins.

© 2001 Microchip Technology Inc.

2.2.8 PS/2 Firmware

The PS/2 firmware is entirely interrupt driven. As mentioned before, an interrupt is generated when the START bit is received, at which time the firmware will begin its receive routine. In addition to this interrupt, every 168 ms, a timer overflow interrupts the normal program flow and implements one state of the mouse/keyboard/cursor demonstration state machine. This state machine handles sending bytes to and translating bytes received from the PS/2 device, automatically. These two interrupts essentially handle everything, except for transferring the bytes via USB to the PC. In addition, it does all of this work in the background while a developer's code runs in the foreground. The only operation that the developer's program must implement is sending keyboard or mouse data to the PC via USB. The developer needs only to be concerned with the TYPE and eight BUFFER registers. BUFFER registers 0 to 7 are the registers where translated PS/2 device data gets placed. TYPE contains the following status bits:

Table 2.3: PS/2 State Machine Status Report

TYPE bit	Name	Description
0	CONNECTED	1 = device connected
1	MOUSE	1 = device connected is a mouse
2	KEYBOARD	1 = device connected is a keyboard
3	DATA READY	1 = data is ready; must be cleared by user

2.2.9 Report Descriptor

The report descriptors used in the example code for both the keyboard and mouse were copied directly out of the *HID Usage Tables*. The *HID Usage Tables* document is published by the USB Implementers Forum (www.usb.org). Many other useful HID report descriptor examples can be found in this document. The keyboard and mouse report descriptors are not sent out at the same time that the PICmicro MCU is enumerated by the host. Rather, the PICmicro MCU will only send the report descriptor that corresponds to the device it has detected as being attached at that time.

2.2.9.1 Keyboard Descriptor

```
0x05, 0x01     usage page (generic desktop)
0x09, 0x06     usage (keyboard)
0xA1, 0x01     collection (application)
0x05, 0x07        usage page (key codes)
0x19, 0xE0        usage minimum (224)
0x29, 0xE7        usage maximum (231)
0x15, 0x00        logical minimum (0)
0x25, 0x01        logical maximum (1)
0x75, 0x01        report size (1)
0x95, 0x08        report count (8)
0x81, 0x02        input (data, variable, absolute)
```


© 2001 Microchip Technology Inc.

```
0x95, 0x01        report count (1)
0x75, 0x08        report size (8)
0x81, 0x01        input (constant)
0x95, 0x05        report count (5)
0x75, 0x01        report size (1)
0x05, 0x08        usage page (page# for Led)
0x19, 0x01        usage minimum (1)
0x29, 0x05        usage maximum (5)
0x91, 0x02        output (data, variable, absolute)
0x95, 0x01        report count (1)
0x75, 0x03        report size (3)
0x91, 0x01        output (constant)
0x95, 0x06        report count (6)
0x75, 0x08        report size (8)
0x15, 0x00        logical minimum (0)
0x25, 0x65        logical maximum (101)
0x05, 0x07        usage page (key codes)
0x19, 0x00        usage minimum (0)
0x29, 0x65        usage maximum (101)
0x81, 0x00        input (data, array)
0xC0           end collection
```

2.2.9.2 Mouse Descriptor

```
0x05, 0x01     usage page (generic desktop)
0x09, 0x02     usage (mouse)
0xA1, 0x01     collection (application)
0x09, 0x01        usage (pointer)
0xA1, 0x00        collection (linked)
0x05, 0x09           usage page (buttons)
0x19, 0x01           usage minimum (1)
0x29, 0x03           usage maximum (3)
0x15, 0x00           logical minimum (0)
0x25, 0x01           logical maximum (1)
0x95, 0x03           report count (3)
0x75, 0x01           report size (1)
0x81, 0x02           input (3 button bits)
0x95, 0x01           report count (1)
0x75, 0x05           report size (5)
0x81, 0x01           input (constant 5 bit padding)
0x05, 0x01           usage page (generic desktop)
0x09, 0x30           usage (X)
0x09, 0x31           usage (Y)
0x15, 0x81           logical minimum (-127)
0x25, 0x7F           logical maximum (127)
0x75, 0x08           report size (8)
0x95, 0x03           report count (2)
0x81, 0x06           input (2 position bytes X & Y)
0xC0              end collection
0xC0           end collection
```

2.2.10 Soft Detach

The soft detach command is one of the most useful features on the USB PICmicro MCU. The reason the soft detach command is so useful is that in applications such as the PS/2 mouse/keyboard example, no special driver has to be created for the host. Most often for this application, the PICmicro MCU would have two configurations - one for a mouse and the other for a keyboard. A special driver would have to be created for the host that would instruct the PICmicro MCU to change configurations. Furthermore, in order to do this, the host's driver would have to be able to detect what peripheral is attached. An easy way to avoid difficulty with creating a unique driver for the host, is to give the PICmicro MCU control over whether it will send the host mouse or keyboard data. Based on the type of data it will send to the PC, the PICmicro MCU can soft detach and re-enumerate as the peripheral of choice.

In this example, several events precede the implementation of a soft detach. As mentioned before, an interrupt is generated when the data line goes low. This initiates the receive routine, because it is assumed that the data line dropping low is the result of a START bit being sent by the device. However, the data line will also drop if the user disconnects the device. To distinguish between these two cases, the firmware starts a timer when the data line goes low. If the timer times out before the clock line goes high, the device has been disconnected. Otherwise, the data line going low is the result of a START bit. Once it has been determined that a device has been disconnected, the firmware waits for another device to be attached. When the clock and data lines are both high, a device has been attached. At this point, the PICmicro MCU will ask the device to identify itself (see Appendix B for keyboard and mouse commands). Based on the device's response, the firmware will know whether the newly attached device is a keyboard or mouse and perform a soft detach.

During the soft detach process, several things occur. First, the PICmicro MCU turns off the pull-up resistor to V_{USB}. The firmware does this by clearing the DEV_ATT bit. Turning the pull-up resistor off, removes the PICmicro MCU from the bus. After approximately 50 ms, which is enough time for the host to see the device disconnect, the firmware sets DEV_ATT and reconnects the PICmicro MCU to the bus. The host then re-enumerates the PICmicro MCU.

2.2.11 Translation to USB

In order to act like a USB mouse or keyboard, the PICmicro MCU has to translate the PS/2 data format for each device to its USB data format. For the mouse, this is very simple. For the keyboard, however, the process is more complex.

© 2001 Microchip Technology Inc.

2.2.11.1 Keyboard

Incoming PS/2 data can be one to eight bytes long. No pattern or mathematical expression can be used to convert incoming PS/2 data to a USB keycode. USB keycodes are one byte in length. There is one USB keycode for every key on the keyboard. A lengthy lookup table is used to implement the translation from PS/2 to USB keycodes. The lookup table is found in file `table_kb.asm` of the example firmware. The USB keycodes are shown in Section 9 of the *HID Usage Tables* document.

2.2.11.2 Mouse

The mouse translation from a PS/2 format to a USB format is very simple. The PS/2 button states are simply copied into the correct bits in the USB buffer. The data overflow and data sign bits in the first PS/2 byte are not necessary and are therefore, not transferred to the corresponding USB byte. The X-axis bytes are identical in both the PS/2 and USB data formats. The PS/2 Y-axis byte is complimented in order to mirror the byte and is then passed into the last byte of the USB mouse data. This makes three bytes of USB mouse data. You may notice, however, that four bytes are sent to the PC in the example code. This is because a mouse is unique, in that it is a boot device and as such, has a stringent format defined for it. This format requires that four bytes always be sent, despite the possibility that the last byte may not be needed for a particular mouse. This fourth byte is reserved for the scrolling wheel found on some mice.

2.3 Combination Gameport/PS/2/Mouse - USB Translator

2.3.1 Introduction

The low speed USB interface has two endpoints. Each endpoint can be configured for transmit or receive, but not both. Each endpoint has its own report descriptor. This example shows how you can create a combination device that provides a completely different interface over two different endpoints. A gamepad is reported on EP2 while EP1 reports a mouse.

2.3.2 Using Multiple Endpoints

There is a hierarchy of descriptors for every USB device. For a HID device that uses only one interface, the hierarchy is shown in Figure 2.4. During enumeration, the host will ask the device for its descriptors with the following sequence of requests.

1. Get_Device_Descriptor
2. Get_Configuration_Descriptor
3. Get_Report_Descriptor
4. Get_String_Descriptor

The device will send out its configuration, interface, HID, and endpoint descriptors (in that order), when it receives the Get_Configuration_Descriptor request. Therefore, you'll find in the firmware that these descriptors fall between the labels Config1 and EndConfig1. In the case of two interfaces, shown in Figure 2.5, the device sends the host the configuration, interface1, HID1, endpoint1, interface2, HID2, and endpoint2 descriptors (in that order), in response to the Get_Configuration_Descriptor request. In this example, interface1 is a mouse that sends data over EP1, and interface2 is a gamepad communicating with the host over EP2. Each interface requires its own report descriptor.

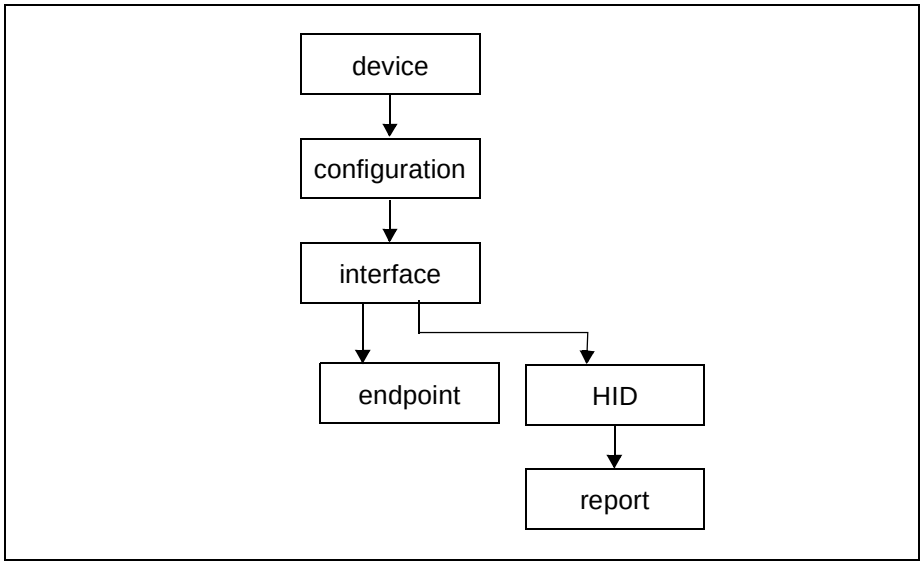


Figure 2.4: One Interface

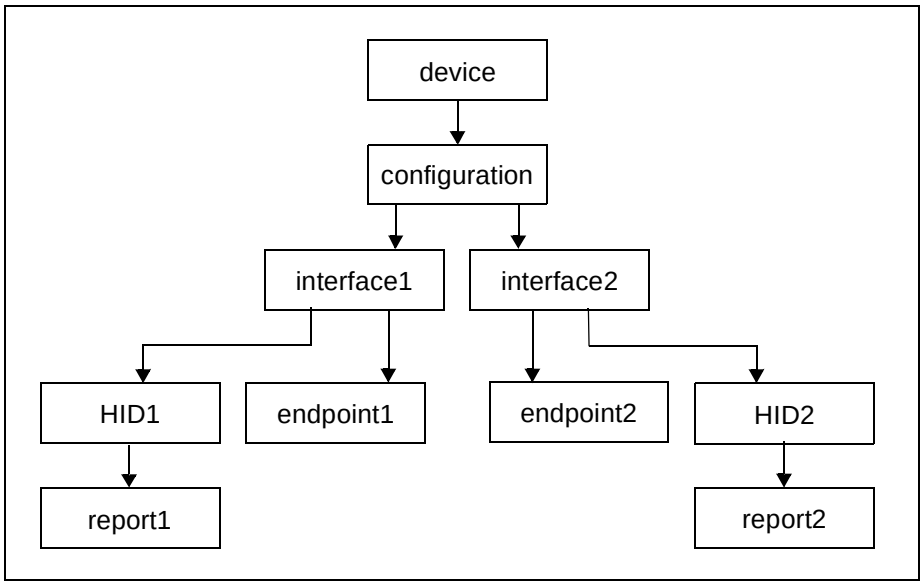


Figure 2.5: Two Interfaces

© 2001 Microchip Technology Inc.

There is no way to tell what endpoint a report descriptor is associated with by looking at the report descriptor. In other words, unlike other descriptor types, report descriptors do not have a field that identifies them as a report descriptor. Nor do report descriptors have an index number that distinguishes one from another. Report descriptors are associated with the appropriate endpoint in the firmware with the use of a Report Descriptor Index.

Looking at Figure 2.5 again, you can see that HID1 is associated with endpoint1 and that HID1 specifies that it has a report descriptor. The host assigns this report descriptor an index value of 0 (this is a zero based array), simply because the HID1 descriptor is the first descriptor sent to the host that specifies a report descriptor. Similarly, the host assigns the report descriptor specified by HID2 an index value of 1. The host keeps track of what descriptors branch off a particular interface, therefore, the host is able to deduce that report descriptors with indexes 0 and 1 are describing the communication that will take place over endpoint1 and endpoint2, respectively. The host will call these report descriptors with the Get_Report_Descriptor request. As part of this request, the host specifies the index number of the descriptor it desires. It's up to the developer to make sure that Report Descriptor 1 is sent to the host when index 0 is specified in the request, and Report Descriptor 2 is sent to the host when index 1 is specified. To make this process as painless as possible, the Report Descriptor Index in file `descript.asm` was created. In this index, all that is needed, is for you to list the labels of your report descriptors in the order that the corresponding HID descriptors are sent to the host.

2.4 Multi-Function LCD Text Display Example

2.4.1 Introduction

USB endpoints can be used to send data from the host as well as receive data from a device. In this example, a simple vendor defined protocol is used to draw text on the LCD. Additional commands are provided to move the cursor around, clear the screen, download custom font characters, send data through the serial port, and light LED's on PORTB. This example does not use the alphanumeric HID class defined by the USB-IF. This example used the second endpoint to provide additional functionality. The second endpoint will receive serial data from the UART and read the analog-to-digital converter.

2.4.2 Hardware Implementation

The hardware is simply the PICDEM™ USB with an LCD character module attached to the LCD connector. The LCD control signals are PORTE, while the data is PORTD. Provision is in place for backlight power on the PICDEM™ USB, but backlight control is not provided. If you use an LCD with an LED backlight, your board could draw slightly more than 100 mA. Fortunately, current PC drivers will not shut-down the device for using too much current, unless the current draw from all the USB ports or a hub exceeds 500 mA. Future PC drivers may require additional power management capabilities.

© 2001 Microchip Technology Inc.

The LCD module used is a standard 2x16 character module, which uses a Hitachi 44780 controller.

2.4.3 LCD Driver Firmware

The `lcd.asm` file contains the LCD firmware. For additional information on the LCD features and command set, consult the documentation for the Hitachi 44780 controller. The firmware provides the following functions to the rest of the USB application.

Table 2.4: LCD Driver Firmware

Command	Function
LCDINIT	Initializes the LCD in 8-bit mode, clears the screen and hides the cursor.
LCDCLEAR	Clears the display and homes the cursor.
LCDHOME	Homes the cursor without clearing the display.
LCDEMODE	Sets the entry mode of the display. Required entry mode must be set in W. b0 : 0 = no display shift 1 = display shift b1 : 0 = auto-decrement 1 = auto-increment b2-7 : not used
LCDDMODE	Sets the display control. Required display mode must be set in W. b0 : 0 = cursor blink off 1 = cursor blink on b1 : 0 = cursor off 1 = cursor on b2 : 0 = display off 1 = display on (display data is intact) b3-7 : not used
LCDSCGA	Sets the LCD Character RAM Address to the value in W. This prepares the display to accept font data at the indicated address.
LCDSDDA	Sets the LCD Data RAM Address to the value in W. This prepares the display to accept characters at the indicated cursor address. It can also be used to move the cursor around the screen.
LCDADDR	Returns the current address in W. Useful for saving the current cursor position before updating the font data.
LCDPUTCHAR	Writes the character in W to the current data memory address. Causes a character to be placed on the screen.
LCDPUTCMD	Writes the byte in W to the command memory. This is used to implement most of the previous commands. This won't be used frequently but if you want to use a specific feature of the display, this is where to look.

Any LCD operation can be performed with these functions. The LCD firmware supplied is a linkable file (`lcd.asm`).

© 2001 Microchip Technology Inc.

2.4.4 LCD Report Descriptor

The report descriptor for this example is very simple; a vendor defined usage page and a vendor defined usage with an 8-byte payload. Two reports are defined, one for input and one for output. The report formats are identical.

```
dt 006h, 000h, 0ffh ; USAGE_PAGE (Vendor Defined Page 1)
dt 009h, 001h       ; USAGE (Vendor Usage 1)
dt 0a1h, 001h       ; COLLECTION (Application)
dt 019h, 001h       ;      USAGE_MINIMUM (Vendor Usage 1)
dt 029h, 008h       ;      USAGE_MAXIMUM (Vendor Usage 8)
dt 015h, 000h       ;      LOGICAL_MINIMUM (0)
dt 026h, 0ffh, 000h ;      LOGICAL_MAXIMUM (255)
dt 075h, 008h       ;      REPORT_SIZE (8)
dt 095h, 008h       ;      REPORT_COUNT (8)
dt 081h, 002h       ;      INPUT (Data,Var,Abs)
dt 019h, 001h       ;      USAGE_MINIMUM (Vendor Usage 1)
dt 029h, 008h       ;      USAGE_MAXIMUM (Vendor Usage 8)
dt 091h, 002h       ;      OUTPUT (Data,Var,Abs)
dt 0c0h             ; END_COLLECTION
```

2.4.5 USB Command Set

The USB command set implementation is simple. The first byte in the packet is the command byte, while the remaining seven bytes contain data. The command byte has a command number in the low nibble, while the upper nibble contains a command modifier. Although six commands are implemented, sixteen commands are available.

Table 2.5: USB Command Set

Command Number	Command Name	Function
0	Clear Display	This command clears the display and homes the cursor. No additional data is required so the data bytes are ignored.
1	Move Cursor	This command moves the cursor to the indicated row and column. The row is specified in the first data byte while the column is specified in the second data byte. The row byte is limited to 2 rows. This code works on a 2x20 LCD. It should also work on a 2x16 LCD. Beyond that, expect to modify the code.
2	Write Text	This commands writes the N characters to the current cursor position. N is the value in the upper nibble of the command and cannot exceed 7. The data is in the 7 data bytes following the command.

Table 2.5: USB Command Set (Continued)

Command Number	Command Name	Function
3	CGRAM Data	This command modifies one of the eight programmable characters in the character generator RAM. The upper nibble defines which character is to be modified. The remaining 7 bytes contain the data. The data is formatted as a 5x7 bit array. The 7 bytes are the 7 rows. Only the 5 LSB's are used in each byte. The top row is the first byte.
4	Serial Data	This command works the same way as the Write Text command. The number of bytes to send is the upper nibble. The data is sent through the serial port at 28800 8n1.
5	LED Data	If the LED's are not being used by the Chapter 9 firmware, you can light the LED's with the contents of data byte 0. This command simply copies the data byte to PORTB.
6-15	Unused	Unused

© 2001 Microchip Technology Inc.

2.4.5.1 Input Commands

The input pipe will send these commands in random order depending on when data is present.

Table 2.6: Input Commands

Command Number	Command Name	Function
0	Unused	Unused
1	Unused	Unused
2	Unused	Unused
3	Unused	Unused
4	Serial Data	Pass up to 7 serial characters back to the host. Total number of characters is in the high nibble. Characters queue until the host asks for them. Baud rate is 28800 8n1.
5	Unused	Unused
6	Unused	Unused
7	ADC Data	Pass 5 bytes of ADC data. Each byte comes from a different ADC channel. Channels AN0-AN4 are read.
8-15	Unused	Unused

2.4.6 PC Code Support

The PC code is written with Visual Basic using an ActiveX® control for the USB interface. The ActiveX control is based on the code in *USB Complete* by Jan Axelson. The PC code interfaces to this example through the windows HID API. Unfortunately, the HID API's don't provide direct access to the endpoints.

2.4.7 Macintosh Code Support

Macintosh computers have supported USB since the introduction of the iMac® and MacOS version 8.6. Operating systems prior to MacOS X are referred to as the "Classic" environment.

2.4.7.1 MacOS 8.6 through 9.1

At this time, no examples are available for the older "Classic" versions of the Macintosh operating system. Consult Apple's USB software development kit for help in creating applications for these operating systems.

2.4.7.2 MacOS X

The MacOS X code is based on the simple example code supplied with the USB DDK from Apple Computer. This example interfaces directly with the USB layer in the operating system, which bypasses the HID system. The advantage to this method is the direct access to the endpoints. Fortunately, this level of access is available from a user program and a kernel level device driver is not required.

© 2001 Microchip Technology Inc.

Chapter 3. PICDEM™ USB Hardware

3.1 Oscillator Support

The PIC16C745/765 can support a few different oscillator options. Regardless of what oscillator is used, the internal clock must be 24 MHz. This is required by the USB Serial Interface Engine (SIE). To assist in running at 24 MHz with a low cost oscillator, the HS and EC clock modes are provided with an internal 4x PLL clock multiplier. This allows 6 MHz oscillators to be used to save cost and EMI. The PICDEM™ USB hardware has the provisions for a canned oscillator, a crystal, or resonators. Refer to Figure 3.1.

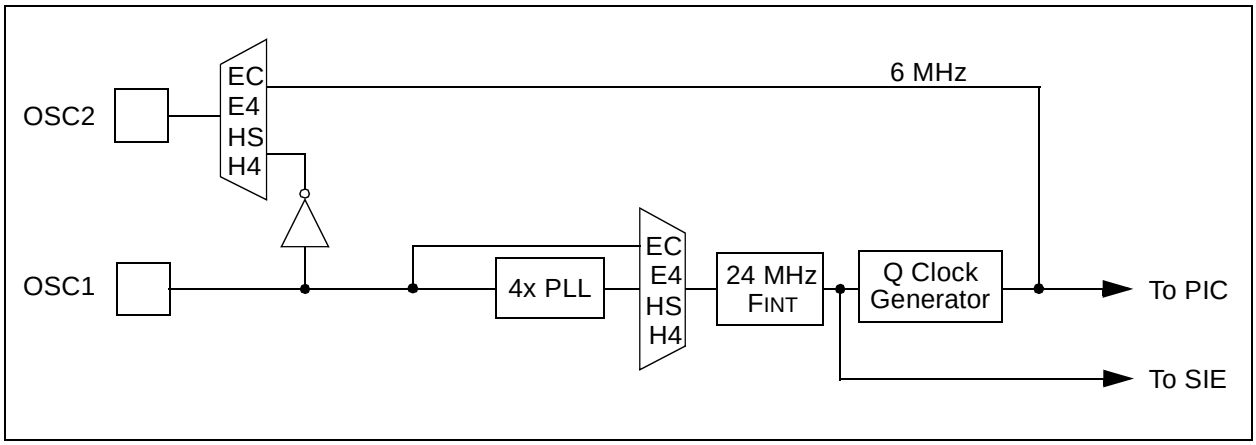


Figure 3.1: Oscillator/PLL Clock Control

3.1.1 Canned Oscillator

Any full size canned oscillator of 6 MHz or 24 MHz can be used. Remove the resonator from the resonator pads to use the oscillator.

3.1.2 Resonator

A 6 MHz or 24 MHz resonator can be used. Pads are available to use resonators with internal or external capacitors. The resonator supplied with the kit is a 6 MHz resonator with internal capacitors. If you decide to use a resonator with external capacitors, you should select capacitors that will maximize the voltage swing across the resonator.

© 2001 Microchip Technology Inc.

3.1.3 Crystal

A fundamental cut crystal of 6 MHz or 24 MHz can be used with additional capacitors installed in the pads provided. The selection of external capacitors must be chosen to maximize the voltage across the crystal.

3.2 Connector Pinout

This board was designed to provide experience with LS HID USB code. To accommodate this goal, two popular interfaces have been selected to allow for HID experimentation, the PS/2 and Gameport. A serial port was added, but it cannot be used at baud rates below 4800 due to the high clock speed. This prevents serial mice from being used. Additional support for LCD and keypad was added to allow for the support of other popular devices.

3.2.1 Gameport (J4) Pinouts

The PC gameport is typically read through a one-shot multi-vibrator. This configuration uses capacitor charge timing to determine the joystick position. The implementation for the PICDEM™ USB uses the analog converters to accomplish the same task. A 50K resistor is used to pull down each analog channel, allowing the ADC to read voltages between 1.25V and 5V. This is not full range, but it will allow joystick functions to work. Removing the pull-down resistors will allow the full 5V range of the ADC to be used. Each button input is pulled to 5V through a 1K resistor. Consult the schematic for additional details.

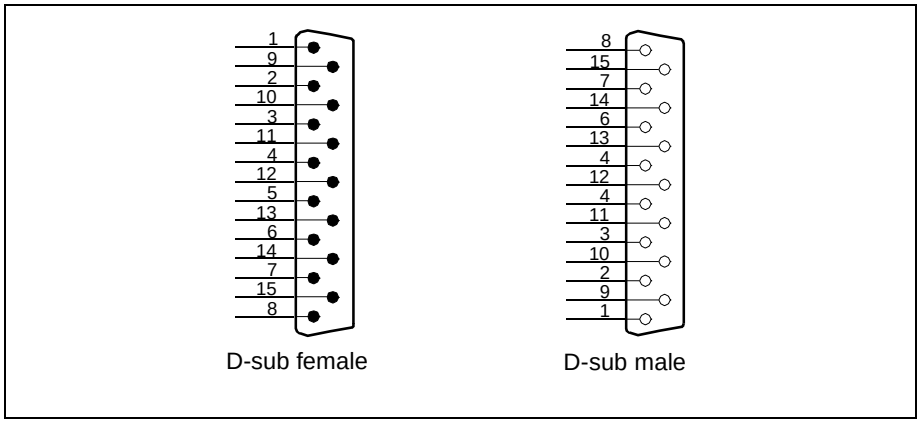


Figure 3.2: Gameport Pinouts (J4)

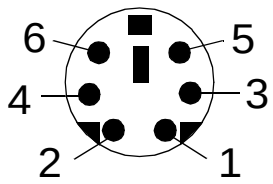
© 2001 Microchip Technology Inc.

Table 3.1: D-sub 15 Female

Pin #	PIC Pin	Pin Function
1	VDD	VDD
2	RD0	Stick 1, Button 1
3	RA0	Stick 1, X axis
4	Vss	Ground
5	Vss	Ground
6	RA1	Stick 1, Y axis
7	RD1	Stick 1, Button 2
8	VDD	VDD
9	RD2	Stick 2, Button 2
10	RA2	Stick 2, Y axis
11	Vss	Ground
12	Vss	Ground
13	RA3	Stick 2, X axis
14	RD3	Stick 2, Button 1
15	VDD	VDD

3.2.2 PS/2 (J12) Pinouts

PS/2 interfaces are all the same, including the mouse and keyboard ports. The interface is a device initiated synchronous serial port. This interface is wired to RC0 and RC1, to allow CCP interrupts to be used to read the serial data.



Note: This is the view of the Din6 female connector on the PICDEM™ USB.

Figure 3.3: PS/2 (J12) Pinouts

Table 3.2: PS/2 (J12)

Pin #	PIC Pin	Pin Function
1	RC0	Data
2		No Connect
3	VSS	Ground
4	VDD	+5V
5	RC1	Clock
6		No Connect

3.2.3 RS-232 (J7) Pinouts

The RS-232 connector is a DCE device, just as in a PC. To connect to a PC, a "null modem" device must be used (swap TX and RX). This allows standard PC peripherals to be attached to the PICDEM™ USB without using any special cables. Presumably the USB would be used for any communications with the PC.

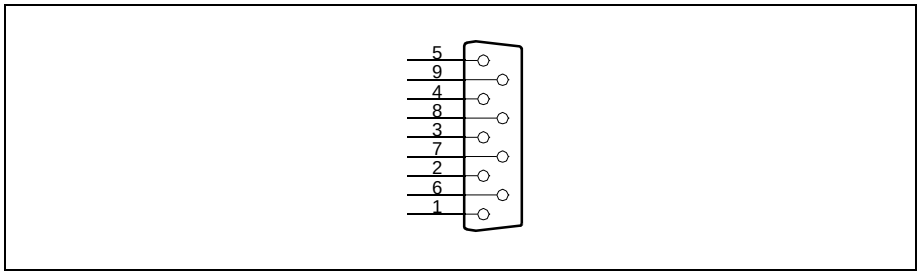


Figure 3.4: RS-232 (J7) Pinouts

Table 3.3: RS-232 (J7)

Pin #	PIC Pin	Pin Function
1		CD
2	RC7	RX
3	RC6	TX
4		DTR
5	Vss	GND
6		DSR
7		RTS
8		CTS
9		RI

© 2001 Microchip Technology Inc.

3.2.4 LCD (J1) Pinouts

The LCD connector provides support for a standard 14-pin LCD module. These modules are available through many electronics supply houses. Any module using a Hitachi 44780A controller should work.

This connector also provides access to the parallel slave port.

Table 3.4: J1 - LCD

Pin #	PIC pin	Pin Function
1	VSS	Ground
2	VDD	+5V
3		LCD Contrast
4	PE0 – RD	Register Select (H for data, L for instruction)
5	PE1 – WR	Read/Write (H for read, L for write)
6	PE2 – CS	Enable
7	PD0 – PSP0	Data 0
8	PD1 – PSP1	Data 1
9	PD2 – PSP2	Data 2
10	PD3 – PSP3	Data 3
11	PD4 – PSP4	Data 4
12	PD5 – PSP5	Data 5
13	PD6 – PSP6	Data 6
14	PD7 – PSP7	Data 7

3.2.5 Keypad (J10) Pinouts

The keypad connector is for attaching a HEX keypad. Any row/column strobbed keypad, with at most 4 rows and columns should work. The pinout was chosen to be compatible with most devices but you may have to make a special cable.

Table 3.5: J10 - Keypad

Pin #	PIC pin	Pin Function
1	PB0	Row Driver
2	PB1	Row Driver
3	PB2	Row Driver
4	PB3	Row Driver
5	PB4	Column Driver
6	PB5	Column Driver
7	PB6	Column Driver
8	PB7	Column Driver
9	No Connect	

© 2001 Microchip Technology Inc.

3.2.6 USB (J8) Pinouts

A standard USB Type B connector was used to allow a USB cable to connect the PICDEM™ USB to the PC. This is against the USB specification. Technically, only full speed devices are allowed to use the type B connector.

Table 3.6: J8 - USB

Pin #	PIC pin	Pin Function
1		Power
2	D+	USB Signal D+
3	D-	USB Signal D-
4		Ground

3.3 Buttons and Jumpers

3.3.1 S1 - $\overline{\text{MCLR}}$

Switch 1 is the $\overline{\text{MCLR}}$ line to the PIC16C745/765. This line resets the entire system.

3.3.2 S2 - RA4

Switch 2 is connected to RA4. This is to allow a general purpose switch input for code experiments. Debounce filtering is provided.

3.3.3 J3 - Bus/Self Power Selection

Jumper 3 switches the V_{DD} power line from Bus power (V_{BUS} from the USB cable) to Self power (5V from the onboard regulator).

3.3.4 J9 - LED Enable

Some applications will want to use PORTB for some purpose other than driving the LED's. By removing the jumper in J9, the LED's will be disabled. The default Chapter 9 firmware uses the LED's to indicate activity at the different stages of enumeration. To use this firmware mode, you should enable the LED's.

© 2001 Microchip Technology Inc.

3.4 Power

3.4.1 Self Power Options

A power supply is provided to run the PICDEM™ USB from a 9-20VAC/DC power supply. A transformer is not supplied in the kit because every example included will run from the bus power.

3.4.2 Bus Power

Most USB peripherals are powered from the 5V available from the USB cable. A minimum of 100 mA is always available. In some situations, a maximum of 500 mA can be requested and used.

NOTES:

© 2001 Microchip Technology Inc.

Chapter 4. Chapter 9 USB Firmware

4.1 Introducing the USB Software Interface

Microchip provides a layer of software for the PIC16C745/65, which handles the lowest level interface. This software provides a simple Put/Get interface for communication so your application won't have to. Most of the USB processing takes place in the background through the Interrupt Service Routine. From the application viewpoint, the enumeration process and data communication takes place without further interaction. However, substantial setup is required in the form of generating appropriate descriptors.

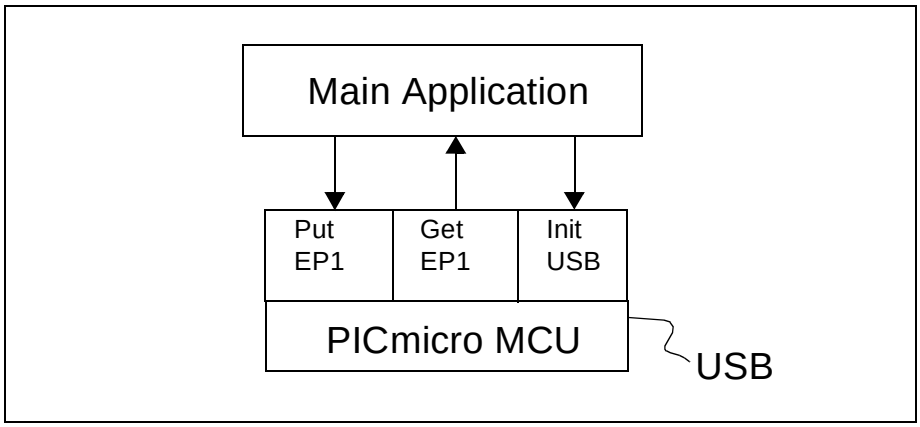


Figure 4.1: USB Software Interface

4.2 Integrating USB Into Your Application

The latest version of the USB interface software is available on Microchip's web site (http://www.microchip.com/).

The interface to the application is packaged in five functions: **InitUSB**, **PutEP1**, **PutEP2**, **GetEP1**, and **GetEP2**. **InitUSB** initializes the USB peripheral, allowing the host to enumerate the device. Then, for normal data communications, the **PutEPn** functions send data to the host and **GetEPn** functions receive data from the host.

Note:	Wherever the command names **PutEPn** and **GetEPn** are used, 'n' represents the endpoint number.

Since the USB depends heavily on the descriptors, a bit of setup work must be completed. The descriptors are the software parameters which communicate to the host what the device is and how to communicate with it. See USB V1.1 Spec., Section 9.5 for more details.

InitUSB enables the USB interrupt so enumeration can begin. The actual enumeration process occurs in the background, driven by the host and Interrupt Service Routine. Macro **ConfiguredUSB** waits until the device is in the CONFIGURED state. The time required to enumerate is completely dependent on the host and bus loading.

4.3 Interrupt Structure Concerns

4.3.1 Processor Resources

Most of the USB processing occurs via the interrupt and thus, is invisible to the application. However, it still consumes processor resources. These include ROM, RAM, Common RAM, Stack Levels, and Processor Cycles. This section attempts to quantify the impact on each of these resources and shows ways to avoid conflicts.

These considerations should be taken into account if you write your own Interrupt Service Routine: Save W, STATUS, FSR, and PCLATH, which are the file registers that may be corrupted by servicing the USB interrupt.

The file `usb_main.asm` provides a skeleton ISR, which does this for you, and includes tests for each of the possible ISR bits. This provides a good starting point if you haven't already written your own.

4.3.2 Stack Levels

The hardware stack on the PICmicro MCU is only eight levels deep. Therefore, the worst case call between the application and ISR can only be eight levels. The enumeration process requires four levels, so it's best if the main application inhibits processing until enumeration is complete. **ConfiguredUSB** is a macro that waits until the enumeration process is complete for exactly this purpose. This macro does this by testing the lower two bits of USWSTAT (0x197).

4.3.3 ROM

The code required to support the USB interrupt, including Chapter 9 interface calls, but excluding the descriptor tables is about 1 kW. The descriptor and string descriptor tables can each require an additional 256W. The location of these descriptors is not restricted, and the linker script may be edited to control the placement of each descriptor's part. See the Strings and Descriptors sections in the linker script.

4.3.4 RAM

With the exception of Common RAM discussed below, servicing the USB interrupt requires ~40 bytes of RAM in Bank 2. This leaves all the General Purpose RAM in Bank 0 and Bank 1, while leaving half of Bank 2 available for your application.

© 2001 Microchip Technology Inc.

4.3.5 Common RAM Usage

The PIC16C745/765 has 16 bytes of common RAM. They are the last 16 addresses in each bank and all refer to the same 16 bytes of memory, without regard to which register bank is currently addressed by the RP0, RP1 and IRP bits.

They are particularly useful when responding to interrupts. When an interrupt occurs, the ISR doesn't immediately know which bank is addressed. With devices that don't support common RAM, the W register must be provided for in each bank. The PIC16C745/765 can save the appropriate registers in Common RAM and not have to waste a byte in each bank for the W register.

4.3.6 Buffer Allocation

The PIC16C745/765 has 64 bytes of Dual Port RAM. Twenty-four bytes are used for the Buffer Descriptor Table (BDT), leaving 40 bytes for buffers.

Endpoint 0 (EP0) IN and OUT need dedicated buffers, since a setup transaction can never be NAKed. That leaves three buffers for four possible endpoints. However, the USB Spec. requires that low speed devices are only allowed 2 endpoints (USB V1.1, paragraph 5.3.1.2), where an endpoint is a simplex connection that is defined by the combination of endpoint number and direction.

The default configuration allocates individual buffers to EP0 OUT, EP0 IN, EP1 OUT, and EP1 IN. The last buffer is shared between EP2 IN and EP2 OUT. Again, the USB Spec. states that low speed devices can only use two endpoints beyond EP0. This configuration supports most of the possible combinations of endpoints (EP1 OUT and EP1 IN, EP1 OUT and EP2 IN, EP1 OUT and EP2 OUT, EP1 IN and EP2 OUT, EP1 IN and EP2 IN). The only combination that is not supported by this configuration is EP2 IN and EP2 OUT. If your application needs both EP2 IN and EP2 OUT, the function **USBReset** will need to be edited to give each of these endpoints dedicated buffers at the expense of EP1.

4.4 File Packaging

The software interface is packaged into four files, designed to simplify the integration with your application.

File `usb_ch9.asm` contains the interface and core functions needed to enumerate the bus. File `descript.asm` contains the device, configuration, interface, endpoint, and string descriptors. Both of these files must be linked in with your application.

File `hidclass.asm` provides some HID Class specific functions. Currently, only **Get_Report_Descriptor** is supported. Other class specific functions can be implemented in a similar fashion. When a get token interrupt determines that it's a class specific command on the basis that ReportType bit 6 is set, control is passed to function **ClassSpecific**. If you're working with a different class, this is your interface between the core functions and the class specific functions.

File `usb_main.asm` is useful as a starting point on a new application, and as an example of how an existing application needs to service the USB interrupt and communicate with the core functions.

4.5 Function Call Reference

Interface between the Application and Protocol layer takes place in three main functions: **InitUSB**, **PutEPn**, and **GetEPn** (with 'n' representing the endpoint number).

4.5.1 Application Layer Function

4.5.1.1 InitUSB

InitUSB should be called by the main program immediately upon power-up. It enables the USB peripheral and the USB Reset interrupt, and transitions the part to the powered state to prepare the device for enumeration. At this point, the USB Reset is the only USB interrupt allowed, preventing the part from responding to anything on the bus until it's been reset. The USB Reset interrupt initializes the Buffer Descriptor Table (BDT) and transitions the part to the default state, where it responds to commands on address zero. When it receives a Set Address command, the device transitions to the addressed state and now responds to commands on the new address.

4.5.1.2 PutEPn

PutEPn (Buffer pointer, Buffer size) sends data to the host. The pointer to the block of data to transmit is in the FSR/IRP, and the block is passed in the W register. The endpoint is set with a Set Instruction preceding the function definition. If the IN buffer is available for that endpoint, the block of data is copied to the buffer. The Data 0/1 bit is then flipped and the OWNS bit is set. A "Buffer Not Available" will occur when it has been previously loaded and the host has not requested that the USB peripheral transmit it. In this case, a failure code would be returned so the application can try again later.

© 2001 Microchip Technology Inc.

4.5.1.3 GetEPn

GetEPn (Buffer Pointer) returns data sent from the host. Provided with a buffer pointer in FSR/IRP, if there is a buffer available (data has been received from the host), the data it is copied to the destination pointed to by FSR/IRP. The endpoint is set with a set instruction preceding the function definition. If no data is available, it returns a failure code. Thus, the functions of polling for buffer ready and copying the data are combined into one function.

4.5.2 Protocol Layer Functions

4.5.2.1 ServiceUSBInt

ServiceUSBInt handles all interrupts generated by the USB peripheral. First, it copies the active buffer to common RAM, which provides a quick turn-around on the buffer in dual port RAM. It also avoids having to switch banks during processing of the buffer.

4.5.2.2 StallUSBEP/UnstallUSBEP

StallUSBEP/UnstallUSBEP sets or clears the STALL bit in the endpoint control register. The STALL bit indicates to the host that user intervention is required and until such intervention is made, further attempts to communicate with the endpoint will not be successful. Once the user intervention has been made, **UnstallUSBEP** will clear the bit, allowing communications to take place. These calls are useful in signaling the host that user intervention is required. An example of this might be a printer out of paper.

4.5.2.3 SoftDetachUSB

SoftDetachUSB clears the DEV_ATT bit, electrically disconnecting the device from the bus, then reconnecting so it can be re-enumerated by the host. This process takes approximately 50 ms, to ensure that the host has seen the device disconnect and re-attach to the bus.

4.5.2.4 CheckSleep

CheckSleep tests the UCTRL.UIDLE bit if set, indicating that there has been no activity on the bus for 3 ms. If set, the device can be put to SLEEP, which puts the part into a Low Power Standby mode until awakened by bus activity. This has to be handled outside the ISR because we need the interrupt to wake us from SLEEP, and because the application may not be ready to SLEEP when the interrupt occurs. Instead, the application should periodically call this function to poll the bit when the device is in a good place to SLEEP.

© 2001 Microchip Technology Inc.

Prior to putting the device to SLEEP, **CheckSleep** enables the activity interrupt so the device will be awakened by the first transition on the bus. The PICmicro MCU will immediately jump to the ISR, which recognizes the activity interrupt and then disables the interrupt and resumes processing with the instruction following the **CheckSleep** call.

4.5.2.5 ConfiguredUSB

ConfiguredUSB (Macro) continuously polls the enumeration status bits and waits until the host has configured the device. This should be used after the call to **InitUSB** and prior to the first time your application attempts to communicate on the bus.

4.5.2.6 SetConfiguration

SetConfiguration is a callback function that allows your application to associate some meaning to a Set Configuration command from the host. The Chapter 9 software stores the value in USB_Curr_Config so it can be reported back on a Get Configuration call. This function is also called passing the new configuration into W. This function is called from within the ISR so it should be kept as short as possible.

4.6 Behind the Scenes

4.6.1 InitUSB

InitUSB clears the error counters and enables the 3.3V regulator and the USB Reset interrupt. This implements the requirement to prevent the PICmicro MCU from responding to commands until the device has been RESET.

The host sees the device and resets the device to begin the enumeration process. The RESET then initializes the Buffer Descriptor Table (BDT) and End-Point Control Registers, in addition to enabling the remaining USB interrupt sources.

The interrupt transfers control to the interrupt vector (address 0x0004). Any Interrupt Service Routine must preserve the processor state by saving the FSR's that might change during interrupt processing. We recommend saving W, STATUS, PCLATH, and FSR. W can be stored in Common RAM to avoid banking issues. Then, the Interrupt Service Routine starts polling the interrupt flags to see what triggered the interrupt. The USB interrupts are serviced by calling **ServiceUSBInt**, which further tests the USB interrupt sources to determine how to process the interrupt.

© 2001 Microchip Technology Inc.

Subsequently, the host sends a setup token requesting the device descriptor. The USB Peripheral receives the Setup transaction, places the data portion in the EP0 out buffer, loads the USTAT register to indicate which endpoint received the data, and triggers the Token Done (TOK_DNE) interrupt. The Chapter 9 commands interpret the Setup token and sets up the data to respond to the request in the EP0 IN buffer. It then sets the UOWN bit to tell the SIE there is data available.

Next, the host sends an IN transaction to receive the data from the setup transaction. The SIE sends the data from the EP0 IN buffer and next sets the Token Done interrupt to notify us that the data has been sent. If there is additional data, the next buffer full is set up in the EP0 IN buffer.

Token processing takes place for the entire enumeration sequence. The device starts in the powered state, transitions to RESET via the Reset interrupt, moves to ADDRESSED via the Set Address command, and shifts to CONFIGURED via a Set Configuration command.

The USB peripheral detects several different errors and handles most internally. The USB_ERR interrupt notifies the PIC that an error has occurred. No action is required by the PICmicro MCU when an error occurs. Instead, the errors are simply acknowledged and counted. There is no mechanism to pull the device off the bus if there are too many errors. If this behavior is desired, it must be implemented in the application.

The Activity interrupt is left disabled until the USB peripheral detects no bus activity for 3 ms. Then, it suspends the USB peripheral and enables the Activity interrupt. The Activity interrupt subsequently re-activates the USB peripheral when bus activity resumes, so processing may continue.

4.6.2 CheckSleep

CheckSleep is a separate call that takes the bus idle one step further and puts the PICmicro MCU to SLEEP if the USB peripheral has detected no activity on the bus. This powers down most of the device to minimal current draw. This call should be made at a point in the main loop where all other processing is complete.

4.7 Examples

This example shows how the USB functions are used. This example first initializes the USB peripheral, which allows the host to enumerate the device. The enumeration process occurs in the background, via an Interrupt Service Routine. This function waits until enumeration is complete. Then, it polls EP1 OUT to see if there is any data available. When a buffer is available, it is copied to the IN buffer. Presumably, your application would be more interesting than Example 4.1.

Example 4.1: USB Peripheral Demo Program

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

CheckEP1                   ; Check Endpoint 1 for an OUT transaction
   bankisel    BUFFER      ; point to lower banks
   pagesel     GetEP1
   movlw       BUFFER
   movwf       FSR         ; point FSR to our buffer
   call        GetEP1      ; If data is ready, it will be copied.
   pagesel     CheckEP1
   btfss       STATUS,C    ; was there any data for us?
   goto        CheckEP1    ; Nope, check again.

PutBuffer
   bankisel    BUFFER      ; point to lower banks
   pagesel     PutEP1
   movlw       BUFFER
   movwf       FSR         ; point FSR to our buffer
   movlw       0x08        ; send 8 bytes to endpoint 1
   call        PutEP1
   pagesel     PutBuffer
   btfss       STATUS,C    ; was it successful?
   goto        PutBuffer   ; No: try again until successful
   pagesel     CheckEP1
   goto        CheckEP1    ; Yes: restart loop

    end
```

4.8 Multiple Configuration or Report Descriptors

The Ch9 firmware makes allowances for the fact that more than one configuration or report descriptor may be desired. Allowances for multiple interface, HID, and endpoint descriptors were not made, because they are not needed. These descriptors are all read in with the configuration descriptor, regardless of how many there are.

© 2001 Microchip Technology Inc.

The host requests the descriptors by specifying the type of descriptor it wants and an index value. If more than one configuration descriptor exists, it will request the first one by specifying an index of zero and the second, by specifying an index of one. To make this process as easy as possible for developers, the functions **Config_descr_index** and **Report_descr_index** have been created in `descript.asm`. These functions will need to be modified if your code has more than one configuration, or more than one report descriptor. All you need to do is specify the starting label for additional descriptors in the lookup table for these functions.

Note: String descriptors also use an index function, string_index (in `descript.asm`). This function will need to be modified in the same manner if your code has a number of strings other than six.

4.9 Optimizing the Firmware

This firmware has been created to provide developers with ready-made USB functions, so they don't have to create these functions for themselves. Most developers will not utilize all of the functions in the Ch9 firmware. In order to optimize the program memory, unused functions can be taken out of the firmware. The following guidelines are a good place to start the optimization.

4.9.1 USB Status on PORTB

The firmware outputs the status of USB communication on PORTB. This feature is intended for use with the PICDEM™ USB circuit board, which drives an LED with each PORTB pin. The LEDs indicate the following USB status information: RB0 – powered, RB1 – default, RB2 – addressed, RB3 – configured, RB4 – sleeping, RB5 – EP0 active, RB6 – EP1 active, RB7 – EP2 active. Obviously, these USB status indicators will probably not be used in a finished product by a developer, although they are very helpful during development. All code associated with the USB status LEDs can be eliminated from the program memory by ensuring that SHOW_ENUM_STATUS is not defined at the top of `usb_ch9.asm`.

4.9.2 Error Counter

Similar to the USB status LEDs, code exists in the firmware that counts various errors for debugging purposes. Program memory space can be saved by making sure that COUNTERRORS is not defined.

4.9.3 GetEP1, GetEP2, PutEP1, and PutEP2

These functions are all macros defined in `usb_defs.inc`. Instances of each of these macros occur in `usb_ch9.asm`. If a developer does not utilize one or more of these functions, space can be saved by removing the instance(s) not needed from `usb_ch9.asm`.

4.9.4 HID Class

The HID class is one of several classes suitable for low speed USB. In addition to these other classes, a vendor-defined class can be specified. Should a developer use a class other than the HID class, any HID class specific code in the firmware would be wasting space. The HID class specific code is found in the file `hidclass.asm`. In a case where the HID class is not being utilized by a developer, this file and any variables or labels associated with it, should be removed from the project.

4.10 Cursor Demonstration

Microchip provides a working USB demonstration. This demonstration has the effect of moving the cursor in a small circle on the user's screen. The following steps will get the demonstration working with an actual part or the MPLAB ICE.

4.10.1 Getting the USB Demonstration to Work on a PIC16C745/65

1. Unzip `usbxxxasm.zip` to a project folder.

Note: In the file name `usbxxxasm.zip`, `xxx` is the version number and may change without notice.

2. Build the project in MPLAB IDE.
3. Program a PIC16C745 using a PICSTART Plus or a PRO MATE II programmer. Make sure the configuration bits are set as follows:
   - Oscillator: H4
   - Watchdog Timer: Off
   - Power-up Timer: Off
   - Code Protect: Off

Note: To program a PIC16C765, you will need to use the linker file and include files associated with the PIC16C765. In addition, you will need to identify the part in the Development mode dialog. See the MPLAB IDE User's Guide for details.

4. Appendix A details the circuit in which to implement the PICmicro MCU. Attach the USB cable to your computer.
5. Provide power to the PICmicro MCU. If you are running Windows 98 (with the USB upgrade), Windows NT, or Windows 2000, your operating system will detect a new device and install the necessary drivers automatically. After this occurs, you should see the cursor rotating in a small circle on your screen. To stop the cursor from rotating, detach the USB cable.

© 2001 Microchip Technology Inc.

4.10.2 Getting the USB Demonstration to Work Using the MPLAB ICE

1. Unzip `usbxxxasm.zip` to a project folder.

Note: In the file name `usbxxxasm.zip, xxx` is the version number and may change without notice.

2. Build the project in MPLAB IDE.
3. Make sure the emulator is set up in the Development Mode dialog, as follows:
   - Tools: MPLAB ICE Emulator
   - Clock: Desired Frequency: 24 MHz
   - Configuration: Watchdog Timer: None
   - Power: Processor Power: From Target Board

Note: The firmware is set up to run on a PIC16C745 as the default. To emulate a PIC16C765, you will need to use the linker file and include files associated with the PIC16C765. In addition, you will need to identify the part in the Development Mode dialog.

4. Appendix A illustrates the circuit in which to implement the MPLAB ICE. Connect the USB cable to your computer. The external oscillator portion of the circuit is optional when using MPLAB ICE.
5. Run the project on the emulator. If you are running Windows 98 (with the USB upgrade), Windows® NT, or Windows® 2000, your operating system will detect a new device and install the necessary drivers automatically. After this occurs, you should see the cursor rotating in a small circle on your screen. To stop the cursor from rotating, press F5.

4.10.3 Emulation Tips

1. Turn on the emulator. Open the project to initialize the emulator. If the project is in a mode other than Emulation, follow step 2.
2. If your project is already open before you turn the emulator on, you can initialize it by first, turning the emulator on. Then, go to the option menu and click Development Mode. Make sure that the MPLAB ICE Emulator radio button is selected in the Tools folder of the Development Mode dialog. Click OK. The emulator will initialize.
3. Be certain the proper processor module and device adapters are installed.
4. Should you have other problems, please refer to the MPLAB ICE Users Guide as your first resource.

Note: Be sure to pull D- up to V_{USB} (via R1) *not* V_{DD}. For more on why this is done, see the section Universal Serial Bus: Transceiver, Regulator in the PIC16C745/765 data sheet.

© 2001 Microchip Technology Inc.

Chapter 5. Troubleshooting

5.1 Introduction

This chapter describes some common problems associated with using PICDEM™ USB and steps to follow to resolve those problems. It also includes frequently asked questions (FAQ's) associated with using PICDEM™ USB.

5.2 Highlights

Highlights covered in this chapter:

- Common Problems
- FAQ's

5.3 FAQ

5.3.1 Hardware Questions

Q: Should I connect VBUS to VUSB?

A: NO, these are completely different voltages. VUSB is a 3.3V power supply, intended to source power to pull-up resistors on D+ or D-. VBUS is a 5V power supply, intended to power the entire circuit board.

Q: Ok, but can I leave the 220 nF capacitor off of VUSB?

A: NO, this capacitor is required to stabilize the VUSB voltage. Make sure you copy the circuit as drawn in the data book.

5.3.2 PC/Windows Questions

Q: When I made a descriptor change, why is Windows still showing my old description?

A: Windows stores most of the descriptor information in the registry. It then makes the assumption that devices with the same vendor and product ID have the same descriptor and will load the old descriptor from the registry. This may be fine for consumers, but engineers require different behavior. Before you enumerate with a new registry, simply use `regedit` to delete all old entries about your device from the registry. If this is slowing you down, use a Macintosh to develop the device descriptors and then switch to Windows to develop the actual application code. The Macintosh does not have a permanent device registry and rereads all the information every time.

© 2001 Microchip Technology Inc.

5.3.3 Macintosh Questions

Q: Where do I go to get example code for Macintosh systems?

A: Going to www.apple.com/usb is a good start.

Q: Why doesn't the keyboard example work on my Macintosh?

A: You are using a version earlier than MacOS 10.0. The SetIdle and GetIdle are required for versions earlier than MacOS 10.0. These functions are not implemented yet.

© 2001 Microchip Technology Inc.

Appendix A. PICDEM™ USB Schematics

A.1 Introduction

This appendix contains schematics associated with PICDEM™ USB.

A.2 Highlights

Topics covered in this appendix:

- PC Peripheral Connectors
- LCD, LED and ICD Connections
- PIC16C745 and Expansion Connector
- PIC16C765 and Oscillator
- Power Supply and USB Connections
- Parts Placement Diagram

© 2001 Microchip Technology Inc.

A.3 Schematics

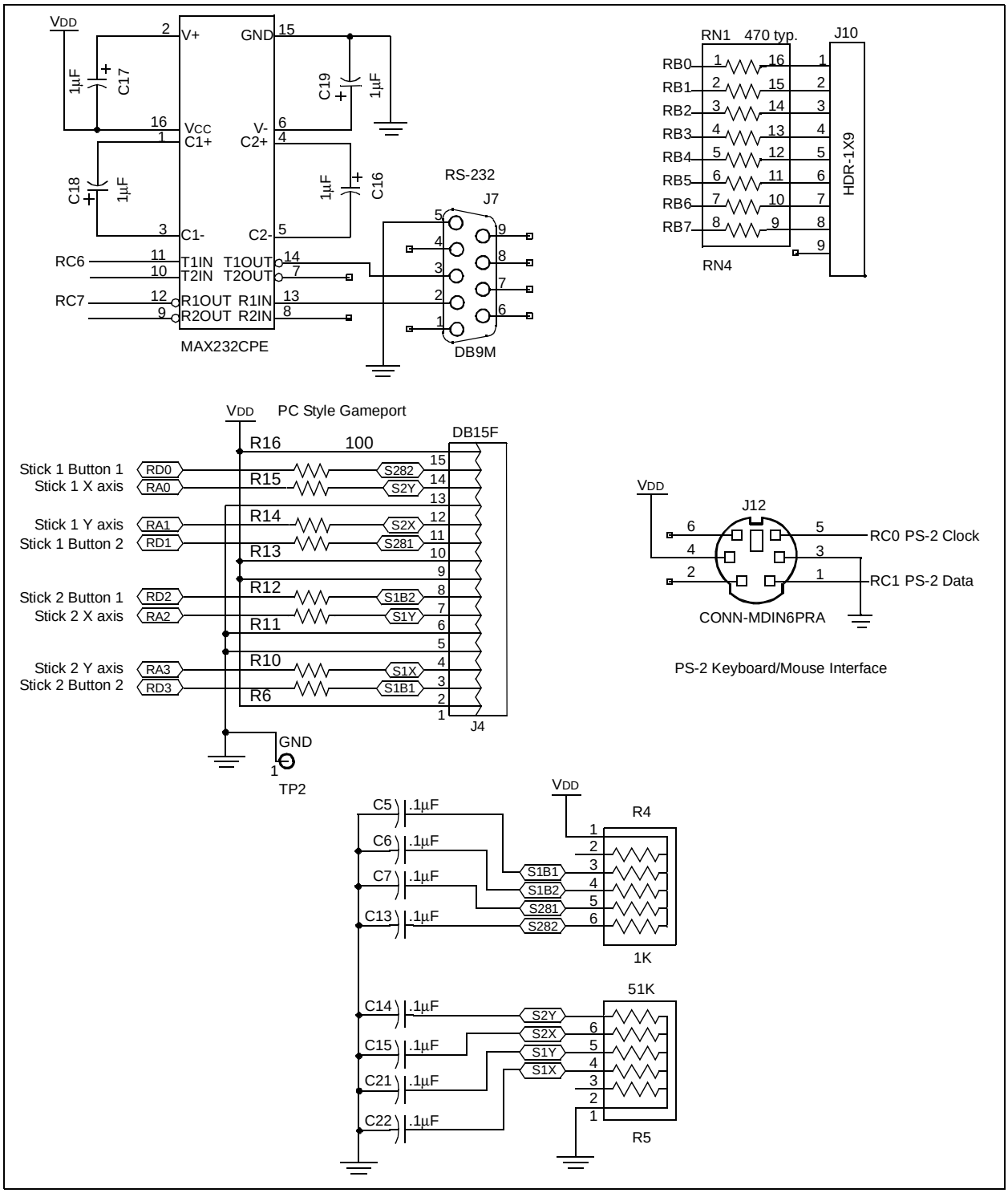


Figure A.1: PC Peripheral Connectors

© 2001 Microchip Technology Inc.

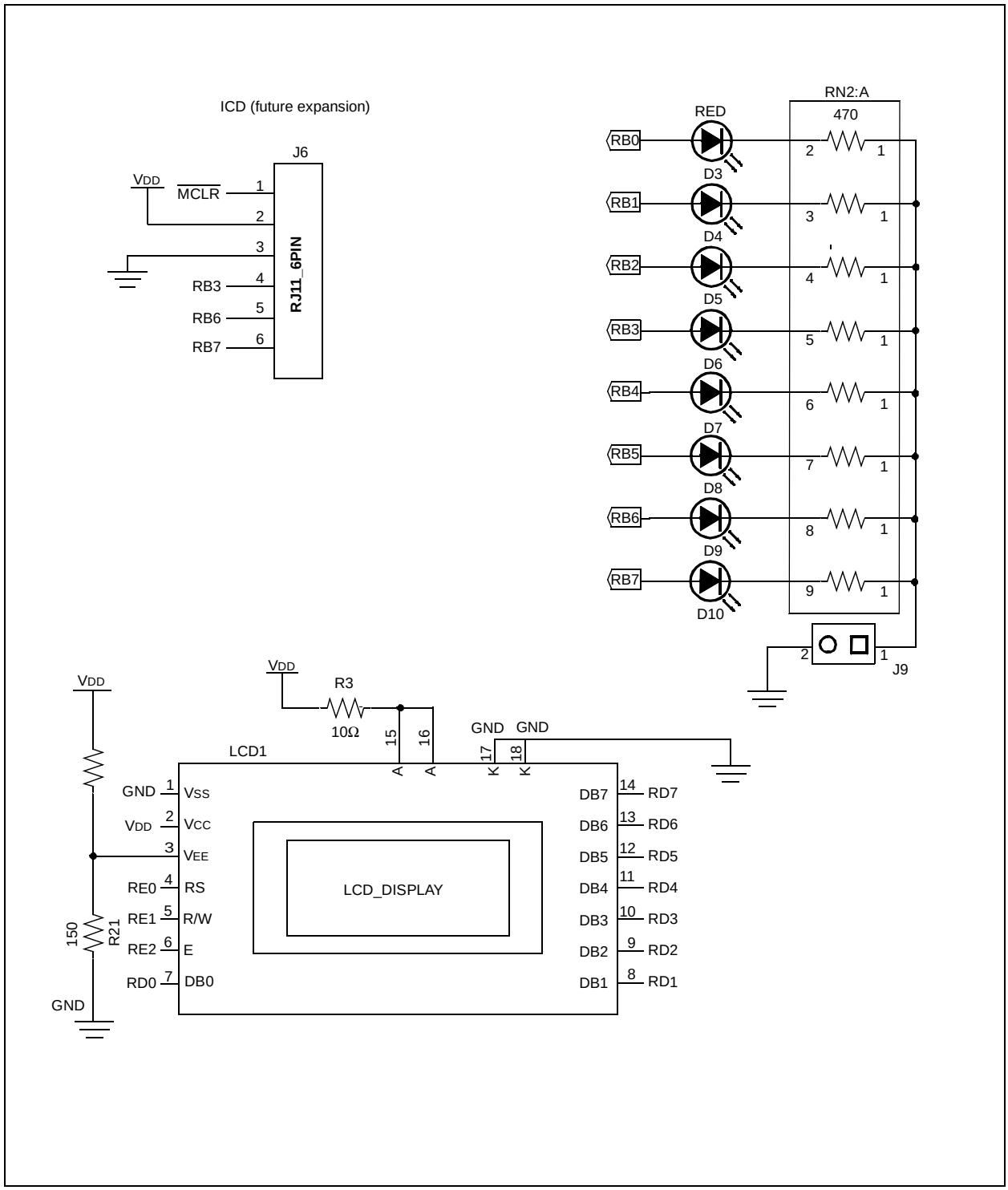


Figure A.2: LCD, LED and ICD Connections

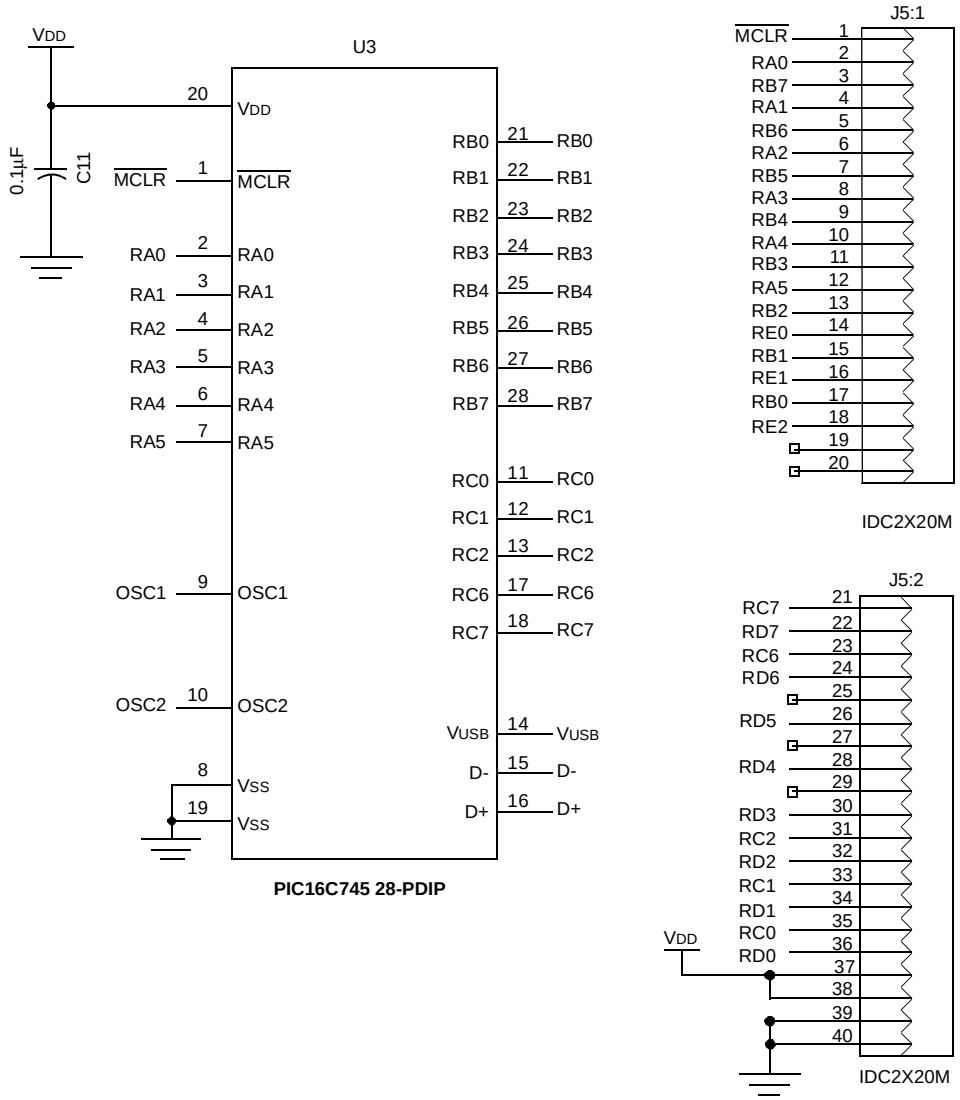


Figure A.3: PIC16C745 and Expansion Connector

© 2001 Microchip Technology Inc.

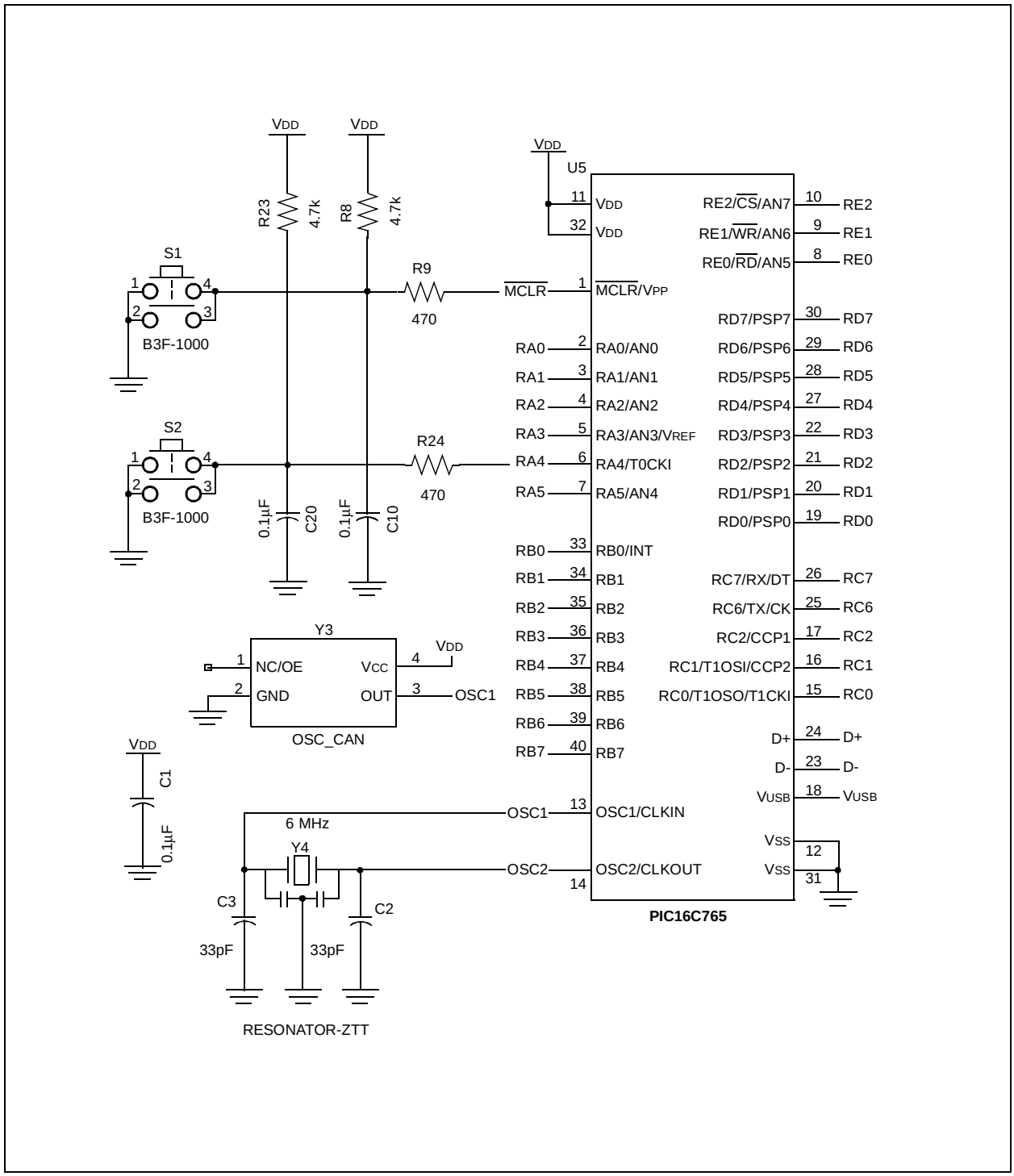


Figure A.4: PIC16C765 and Oscillator

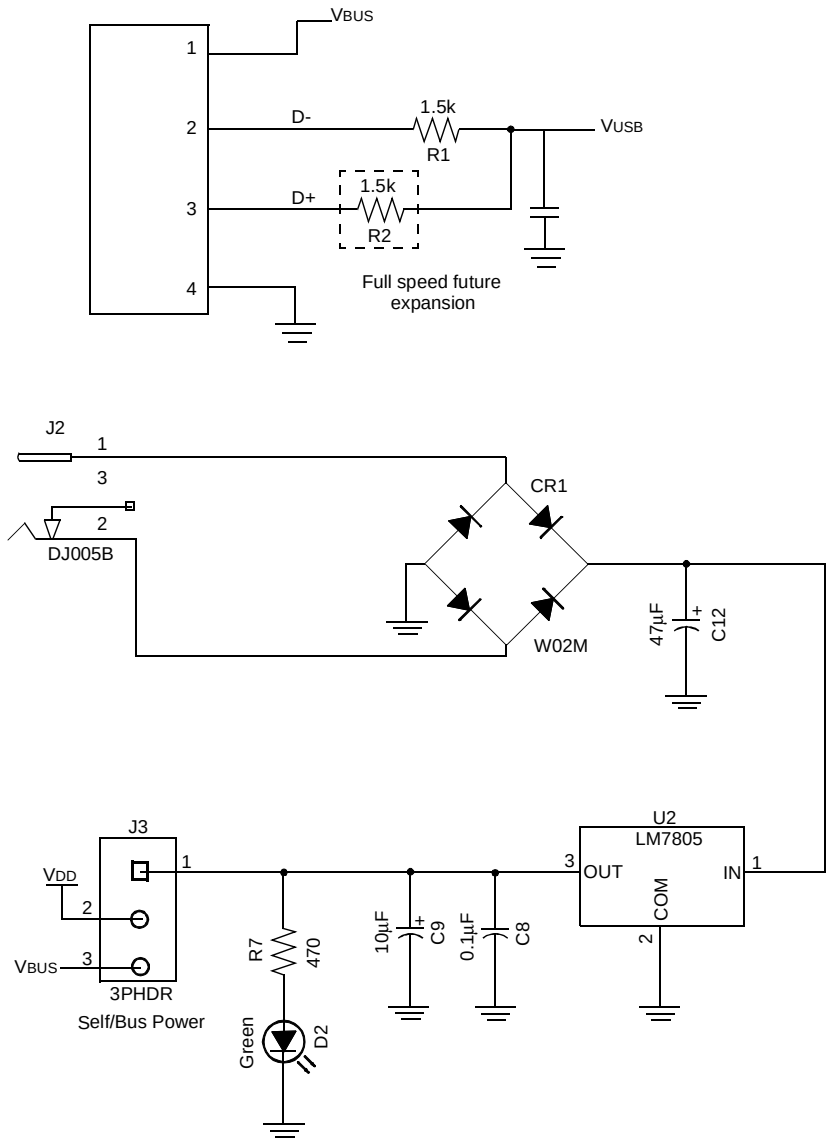


Figure A.5: Power Supply and USB Connections

© 2001 Microchip Technology Inc.

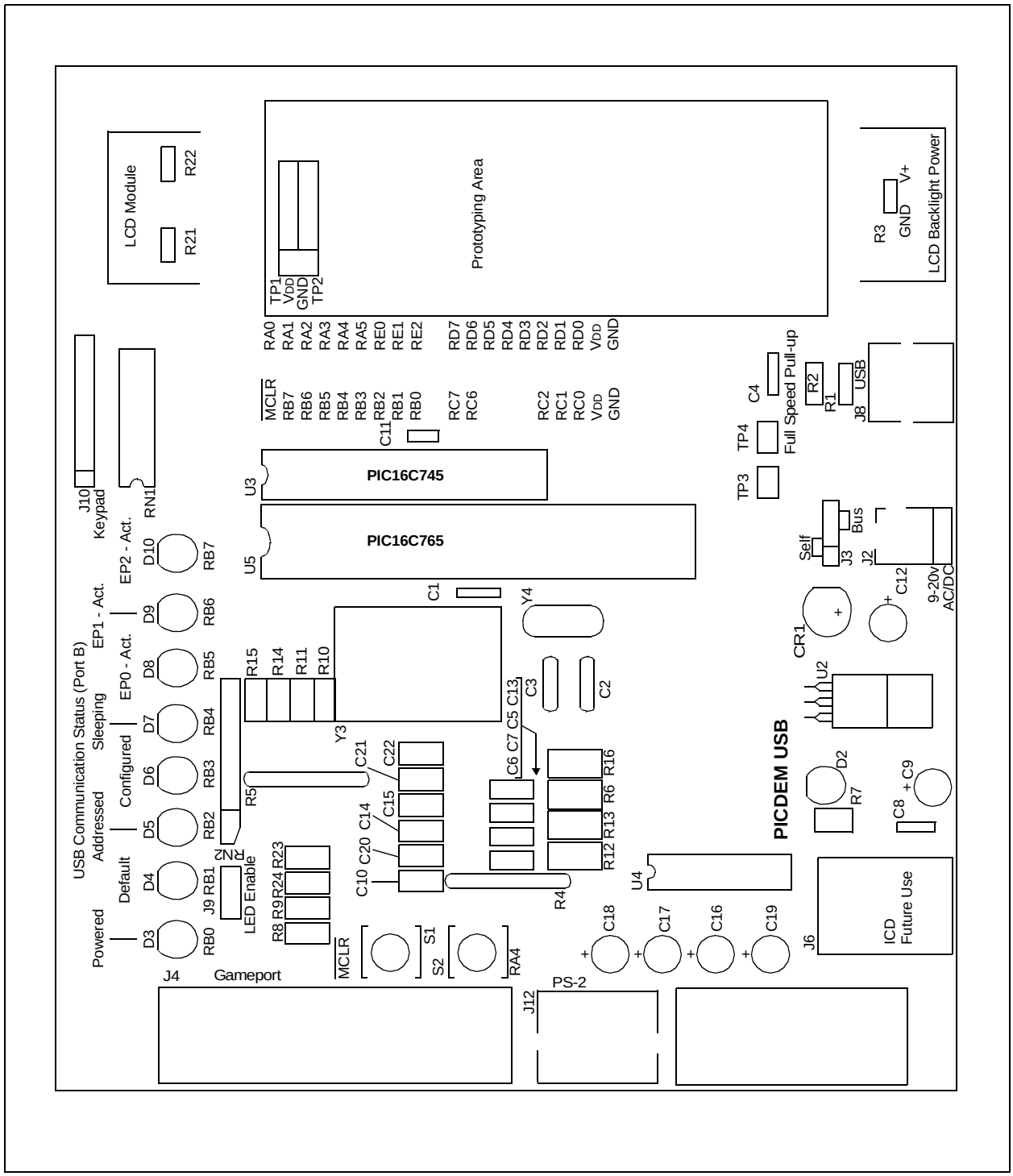


Figure A.6: Parts Placement Diagram

NOTES:

© 2001 Microchip Technology Inc.

Appendix B. PS/2 Lookup Tables

B.1 Introduction

This appendix contains tables identifying command and scan codes associated with PICDEM™ USB.

B.2 Scan Codes

Table B-1: PS/2 Keycodes

Key	Make	Break	Key	Make	Break	Key	Make	Break
A	1C	F0,1C	9	46	F0,46	[	54	F0,54
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	F0,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,6	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	F0,12	D ARROW	E0,72	E0,F0,72
K	42	F0,42	L CTRL	14	F0,14	R ARROW	E0,74	E0,F0,74
L	4B	F0,4B	L WIN	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R WIN	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	5	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	6	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	4	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	3	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	1	F0,01	]	5B	F0,5B
3	26	F0,26	F10	9	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	'	52	F0,52
5	2E	F0,2E	F12	7	F0,07	,	41	F0,41
6	36	F0,36	PAUSE	E1,14,77 E1,F0,14 F0,77	NONE	.	49	F0,49
7	3D	F0,3D				/	4A	F0,4A
8	3E	F0,3E				PRNT	E0,12,	E0,F0,7C
SCROLL	7E	F0,7E				SCRN	E0,7C	E0,F0,12

© 2001 Microchip Technology Inc.

B.3 Command Codes

Table B-2: Host to PS/2 Keyboard Commands

HEX Code	Description
ED	Turns on/off LEDs -- Keyboard replies with ACK (FA) and waits for another byte to be sent. Next byte sent determines the state of the LEDs. (Bits 0-2 correspond to LEDs 1,2,3. Bits 3-7 should always be 0.)
EE	Echo -- Keyboard should respond with Echo (EE).
F0	Set Scan Code Set -- Responds with ACK (FA) and waits for another byte to be sent. Next byte sent will be either 01, 02, or 03 (corresponding to scan code sets 1, 2, and 3.) If 00 is sent (instead of 01, 02, or 03), keyboard will respond with ACK (FA), followed by the current scan code set (again, 01, 02, or 03).
F2	Get ID -- Responds with ACK (FA), followed by an ID (A3, AB). This also enables scanning.
F3	Set repeat rate -- Keyboard replies with ACK (FA) and waits for another byte to be sent. Next byte sent will determine the type of automatic repeat rate for the keyboard. After this byte is sent, keyboard responds with another ACK (FA).
F4	Enable keyboard -- Clear the buffer and start scanning for data; replies with ACK (FA).
F5	Disable keyboard -- Disables scanning and replies with ACK (FA). Does not affect indicator LEDs.
F6	Restore default values. Does not affect indicator LEDs.
F7	Set all keys typematic. Responds with ACK (FA).
F8	Set all keys make/break. Responds with ACK (FA).
F9	Set all keys make. Responds with ACK (FA).
FA	Set all keys typematic/make/break. Responds with ACK (FA).
FB	Set key type typematic.
FC	Set key type make/break.
FD	Set key type make.
FE	Resend -- Keyboard responds by retransmitting the last command it sent.
FF	Reset -- Resets the keyboard.

© 2001 Microchip Technology Inc.

Table B-3: PS/2 Keyboard to Host Commands

HEX Code	Description
00	Key detection error/keyboard buffer overflow (if sets 2 or 3 scan codes are enabled).
83	ABKeyboard ID.
AA	Self-test passed.
EE	Echo -- Sent to Host after receiving "Echo" command from host.
FA	Acknowledge (ACK).
FC	Self-test failed.
FE	Resend -- Host responds by retransmitting the last command sent.
FF	Key detection error/keyboard buffer overflow (if set 1 scan codes are enabled).

Table B-4: PS/2 Mouse Commands

HEX Code	Description
FF	Reset -- This command causes the mouse to enter the RESET mode and do an internal Self-Reset.
FE	Resend -- Any time the mouse receives an invalid command, it returns a resend command to the host system. The host system, in turn, sends this command when it detects any error in any transmission from the mouse. When the mouse receives the resend command, it retransmits the last packet of data sent.
F6	Set Default -- This command re-initializes all conditions to the power-on default state.
F5	Disable -- This command is used in the Stream mode to stop transmissions initiated by the mouse. The mouse responds to all other commands while disabled. If the mouse is in the Stream mode, it must be disabled before sending it any command that requires a response.
F4	Enable -- This command is used in the Stream mode to begin transmission.
"F3,XX"	Set sampling rate -- In the stream mode, this command sets the sampling rate to the value indicated by byte XX (HEX)/sex.
F2	Read device type -- This command always receives a response of 0x00 from the mouse.
F0	Set Remote mode -- Sets the mouse to Remote mode. Data values are reported on in response to a read data command.
EE	Set Wrap mode -- Sets the mouse to Wrap mode. The mode remains until 0xFF or 0xEC is received.
EC	Reset Wrap mode -- The mouse returns to the previous mode of operation after receiving this command.

© 2001 Microchip Technology Inc.

Table B-4: PS/2 Mouse Commands (Continued)

HEX Code	Description
EB	Read data -- This command requests that all data defined in the data packet format be transmitted. This command is executed in either Remote or Stream mode. This data is transmitted even if there has been no movement since the last report or the switch status is unchanged.
EA	Set Stream mode -- This command sets the mouse to Stream mode.
E9	Status request -- When this command is issued by the system, the mouse responds with a 3-byte status report, same as Data Report.
"E8,XX"	Set resolution -- The mouse provides 4 resolutions, selected by the second byte of the command.
E7	Set scaling 2:1 -- Scaling is used to provide a course/fine tracking response. At the end of a sample interval in the stream mode, the current X and Y data values are converted to new values. The sign bits are not involved in the conversion. 2:1 scaling is performed only in Stream mode. In response to a read data command, the current value before conversion is sent.
E6	Reset scaling -- This command restores scaling to 1:1.

© 2001 Microchip Technology Inc.

Glossary

Introduction

This glossary provides a common frame of reference by defining the terms that follow.

Highlights

This glossary contains:

- PICSTART Plus Terms

PICDEM™ USB Terms

Application

A set of software and hardware developed by the user, usually designed to be a product controlled by a PICmicro microcontroller.

Assembler Source Code

A text file that is processed by an assembler to produce a one-to-one correspondence between assembler instructions and PICmicro machine code.

Assemble

To translate a user's "ASM" source text code into machine code.

Breakpoint – Software

An address where execution of the firmware will HALT.

Breakpoint – Hardware

An event whose execution will cause a HALT.

Build

A function that recompiles all the source files for an application.

C Source Code

A program written in the high level language called "C," and which will be converted into PICmicro machine code.

Compile

To translate a user's "C" source text code into machine code.

Configuration Bits

Unique bits programmed to set modes of operation. A configuration bit may or may not be preprogrammed. For MPLAB ICE, these bits are set in the *Options>Processor Setup* dialog.

© 2001 Microchip Technology Inc.

Data RAM

General purpose file registers from RAM on the PICmicro device being emulated. The File Register window displays data RAM.

Download

Download is the process of sending data from the host PC to the emulator or to the target board.

EEPROM

Electrically Erasable Programmable Read Only Memory. A special type of PROM that can be erased electrically. Data is written or erased one byte at a time. EEPROM retains its contents even when power is turned off.

Emulation

The process of executing software loaded into memory on the emulator probe, as if the firmware resided on the microcontroller device under development.

Emulation Memory

Program memory contained within the emulator.

Emulator

Hardware that performs emulation.

Emulator System

The Microchip Emulator System includes the MPLAB ICE Pod, processor module, device adapter, cables and the MPLAB software.

Enumeration

Enumeration is the process of a device communicating its requirements to the host and the host assigning a unique address to the device.

Event

A description of a bus cycle which may include address, data, pass count, external input, cycle type (fetch, R/W) and time stamp.

Export

Send data out of MPLAB IDE in a standardized format.

Extended Microcontroller Mode (PIC17CXX Devices Only)

In Extended Microcontroller mode, on-chip program memory, as well as external memory, are available. Execution automatically switches to external if the program memory address is greater than the internal memory space of the PIC17CXX device. Inaccessible memory in Extended Microcontroller mode includes configuration bits, test memory, and boot memory.

External Input Line (MPLAB ICE 2000 Only)

An External Input Signal Logic Probe Line (TRIGIN) for setting an event based upon external signals.

© 2001 Microchip Technology Inc.

External RAM (PIC17CXX Devices Only)

Off-chip Read/Write memory.

File Registers

On-chip general purpose and special function registers.

FLASH

A type of EEPROM where data is written or erased in blocks instead of bytes.

FNOP

Forced No Operation. A forced NOP cycle is the second cycle of a two-cycle instruction. Since the PICmicro architecture is pipelined, it pre-fetches the next instruction in the physical address space while it is executing the current instruction. However, if the current instruction changes the program counter, this pre-fetched instruction is explicitly ignored, causing a forced NOP cycle.

HALT

The command that stops the emulator. Executing HALT is the same as stopping at a break point. The program counter stops, and the user can inspect and change register values, and single step through code.

HEX Code

A file of executable instructions assembled or compiled from source code into standard HEX format code. HEX code can be directly converted to object code.

High Level Language

A language for writing programs that is of a higher level of abstraction from the processor than assembler code. High level languages (such as C) employ a compiler to translate statements into machine instructions that the target processor can execute.

ICE

In-Circuit Emulator. MPLAB ICE is Microchip's In-Circuit Emulator that works with MPLAB IDE.

IDE

Integrated Development Environment. An application that has multiple functions for firmware development. The MPLAB IDE integrates a compiler, an assembler, a project manager, an editor, a debugger, a simulator, and an assortment of other tools within one Windows application. A user developing an application should be able to write code, compile, debug and test an application without leaving the MPLAB desktop.

Import

Bring data into the MPLAB Integrated Development Environment (IDE) from an outside source

Mac

Short for Macintosh.

MacOS

The Macintosh Operating System. Versions 8.6 and higher have USB support.

Make Project

A command that rebuilds an application, recompiling only those source files that have changed since the last complete compilation.

Microcontroller Mode

One of the possible program memory configurations of the PIC17CXX family of microcontrollers. In Microcontroller mode, only internal execution is allowed. Thus, only the on-chip program memory is available in Microcontroller mode. Accessible memory includes: program memory, configuration bits, test memory, and boot memory (FE00h to FFFFh).

Microprocessor Mode

One of the possible program memory configurations of the PIC17CXX family of microcontrollers. In Microprocessor mode, the on-chip program memory is not used. The entire 64K program memory is mapped externally. Inaccessible memory in Microprocessor mode includes configuration bits, test memory, and boot memory.

MPLAB ICE

Microchip's in-circuit emulator that works with MPLAB IDE.

MPSIM

Microchip's simulator that works with MPLAB IDE.

MPLAB Software

The name of the main executable program that supports the IDE with an Editor, Project Manager, and Emulator/Simulator Debugger. The MPLAB Software resides on the PC host. The executable file name is `MPLAB.EXE`. `MPLAB.EXE` calls many other files.

MRU

Most Recently Used. Refers to files and windows available to be selected from MPLAB main pull-down menus.

Non Real-Time

Refers to the processor executing single step instructions, or MPLAB IDE being run in Simulator mode.

NOP

No operation.

© 2001 Microchip Technology Inc.

Object Code

The machine code that is produced from the source code after it is processed by an assembler or compiler. This code will be the memory resident code that will run on the PICmicro MCU in the user's application. Relocatable code is code produced by MPASM Assembler or MPLAB C17, that can be run through MPLINK Object Linker.

Off-Chip Memory

Off-Chip Memory refers to the memory selection option for the PIC17CXX device, where memory may reside on the target board, or where all program memory may be supplied by the emulator. *Options > Processor Setup > Hardware* provides the Off-Chip Memory selection dialog box.

PC

Any IBM® or compatible Personal Computer. MPLAB IDE needs a 486X or better machine.

Host

The computer that is serving as master to a USB device. This could be any machine with a USB interface, including PC's or Mac's.

PICmicro MCU

PICmicro MCU refers to the PIC12CXX, PIC14000, PIC16C5X, PIC16CXX, PIC17CXX and PIC18CXX Microchip microcontroller families.

PICMASTER Emulator

The hardware unit that provides tools for emulating and debugging firmware applications. This unit contains emulation memory, break point logic, counters, timers, and a trace analyzer among some of its tools. MPLAB ICE is the newest emulator from Microchip.

Pod

The external emulator box that contains emulation memory, trace memory, event and cycle timers, and trace/break point logic. Occasionally used as an abbreviated name for the MPLAB ICE Emulator.

Power-on Reset Emulation

A software randomization process that writes random values in data RAM areas to simulate uninitialized values in RAM upon initial power application.

Program Counter

A register that specifies the current execution address for emulation and simulation.

Program Memory

Memory containing the downloaded target application firmware.

Project

A set of source files and instructions to build the object code for an application.

PRO MATE II

A device programmer from Microchip. Can be used with MPLAB IDE or stand-alone.

Prototype System

A term referring to a user's target application, or target board.

PWM Signals

Pulse Width Modulation Signals. Certain PICmicro devices have a PWM peripheral.

Radix

The number base, hex, or decimal, used in selecting an address and for entering data in the *Window > Modify* command.

Real-Time

When released from the HALT state, the processor runs in Real-Time mode and behaves exactly as the normal chip would behave. In Real-Time mode, the real-time trace buffer is enabled and constantly captures all selected cycles, and all break logic is enabled. In the emulator, the processor executes in real-time until a valid break point causes a HALT, or until the user halts the emulator.

In the simulator, real-time simply means execution of the microcontroller instructions as fast as they can be simulated by the host CPU.

Run

The command that releases the emulator or simulator from HALT, allowing it to run the application code and change or respond to I/O in real-time.

SFR

Special Function Registers of a PICmicro MCU.

Simulator

A software program that models the operation of the PICmicro microprocessor.

Simulator Stimulus

Data generated to exercise the response of MPSIM Simulator to external signals. Often the data is put into the form of a list of actions in a text file.

Single Step

This command steps though code, one instruction at a time with MPSIM Simulator or an emulator. After each instruction, MPLAB IDE updates register windows, watch variables, and status displays, so you can analyze and debug instruction execution.

You can also single step C compiler source code, but instead of executing single instructions, MPLAB IDE will execute all assembly level instructions generated by the line of the high level C statement.

© 2001 Microchip Technology Inc.

Source

Source code, usually a text file of assembly instructions or C code.

Special Function Registers

Registers that control I/O processor functions, I/O status, timers, or other modes or peripherals.

Stack

“Push-Down” list of calling routines. Each time a PICmicro microcontroller executes a call or responds to an interrupt, the software pushes the return address to the stack. A return command pops the address from the stack.

Step-Into

This command is the same as Single Step. Step-Into (as opposed to Step-Over) follows a CALL instruction into a subroutine.

Step-Over

Step-Over allows you to debug code without stepping into subroutines. When stepping over a CALL instruction, the next break point will be set at the instruction after the CALL. If for some reason, the subroutine gets into an endless loop or does not return properly, the next break point will never be reached.

The Step-Over command is similar to Single Step, except for its handling of CALL instructions.

Symbol

A label usually produced by an assembler or compiler that refers to machine locations by function names, variable locations, constant declarations, source line number, or other reference to user source code.

System Button

The System Button is located in the upper left corner of Windows and some dialogs. This button usually has “Minimize,” “Maximize,” and “Close.” In some MPLAB windows, additional modes or functions can be found under the System Button.

Target

Refers to user hardware.

Target Application

Firmware residing on the target board.

Target Board

The circuitry and programmable device that makes up the target application.

Target Processor

The microcontroller device on the target application board that is being emulated.

Template

Lines of text that you build for inserting into your files at a later time. The MPLAB Editor stores templates in template files.

Tool Bar

A row or column of icons that you can click on to execute MPLAB functions.

Trace

An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer, which is uploaded to MPLAB's trace window.

Trace Memory

Trace Memory contained within the Emulator. Trace Memory is sometimes called the Trace Buffer.

Trigger Output

Trigger Output refers to an emulator output signal that can be generated at any address or address range, and is independent of the trace and break point settings. Any number of trigger output points can be set.

Upload

The Upload function transfers data from the emulator to the host PC, or from the target board to the emulator.

USB

Universal Serial Bus. A popular bus for connecting peripherals to personal computers.

Warning

An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

Watchdog Timer

A timer on a PICmicro microcontroller that resets the processor after a selectable length of time.

Watch Variable

A variable that you may monitor during a debugging session. Watch windows contain a list of watch variables that are updated at each break point.

© 2001 Microchip Technology Inc.

Index

A

Applications 10
  Describing 11
  Integrating USB 39

B

Breakpoints 65
Buttons 36

C

Codes 61
  Command 62
    Host to PS/2 62
    PS/2 Mouse 63
    PS/2 to Host 63
  Scan 61
    PS/2 Keycodes 61
Components 9
Configuration Bits 65
Cursor Demonstration 48
  MPLAB ICE 49
  PIC16C745/765 48
Customer Support 7

D

Default Demonstration 10
Demonstration Code 13
Descriptors
  Creating 11
  Multiple 46
  Report - Debugging 11
Development Systems
  Compilers 6
  Emulators 6
  MPLAB IDE 7
  Programmers 6
Development Systems Customer
    Notification Service 5
Document Conventions 2
  Updates 3

E

EEPROM 66
Emulation 50
Emulator System 66
Endpoints, Multiple 23
Error Counter 47
Extended Microcontroller Mode 66
External Input Line 66
External RAM 67

F

Function Call Reference
  GetEPn 43
  PutEPn 42
Functions
  CheckSleep 43, 45
  ConfiguredUSB 44
  InitUSB 42, 44
  ServiceUSBInt 43
  SetConfiguration 44
  SoftDetachUSB 43
  StallUSBEP/UnstallUSBEP 43

G

Gameport 13
  Connector Pinout 32
  Hardware Implementation 14
  Identification Codes 17
  Reading Firmware 15
  Translation 16
  USB Translator 13
Gameport/PS/2 Combination 23
GetEPn 47
Glossary 65

H

HID Class 48

© 2001 Microchip Technology Inc.

I

ICE 67
IDE 67
Internet Address 4
Interrupt Structure 40
    Buffer Allocation 41
    File Packaging 41
    Function Call Reference 42
    Processor Resources 40
    RAM 40
    ROM 40
    Stack Levels 40

J

Jumpers 36

K

Keypad
    Connector Pinout 35

L

LCD
    Connector Pinout 35
    Hardware Implementation 25
    Macintosh Code Support 30
    PC Code Support 29
    PC/Macintosh Identification 29
    Report Descriptor 27
    USB Command Set 27
LCD Driver Firmware 26
LCD Text Display 25

M

Microchip Internet Web Site 4
Microcontroller Mode 68
Microprocessor Mode 68
MPLAB 68
MPLAB ICE 68
MPSIM 68

O

Off-Chip Memory 69
Oscillator Support 31
    Canned 31
    Crystal 32
    Resonator 31

P

Pass Counter 69
PICMASTER 69
Power 37
    Bus Power 37
    Self Power 37
Project 69
PS/2 17
    Connector Pinout 33
    Firmware 20
    Hardware Implementation 17
PutEPn 47

R

Recommended Reading 3
RS-232
    Connector Pinout 34

S

Schematics 54

T

Text Terminal
    USB Command Set
        Input Commands 29
Trace 72
Troubleshooting 51

U

USB
    Connector Pinout 36
    Software Interface 39
    Status on PORTB 47

W

Warranty Registration 3
WWW Address 4

© 2001 Microchip Technology Inc.

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: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Austin - Analog
13740 North Highway 183
Building J, Suite 4
Austin, TX 78750
Tel: 512-257-3370 Fax: 512-257-8526

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Boston - Analog
Unit A-8-1 Millbrook Tarry Condominium
97 Lowell Road
Concord, MA 01742
Tel: 978-371-6400 Fax: 978-371-0050

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Two Prestige Place, Suite 130
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Rm. 531, North Building
Fujian Foreign Trade Center Hotel
73 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7557563 Fax: 86-591-7557572

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Germany - Analog
Lochhamer Strasse 13
D-82152 Martinsried, Germany
Tel: 49-89-895650-0 Fax: 49-89-895650-22

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

08/01/01

© 2001 Microchip Technology Inc.