Development Kit For the PIC® MCU

Exercise Book

USB

March 2010

Custom Computer Services, Inc.
Brookfield, Wisconsin, USA
262-522-6500

Recognized
Microchip
Third-Party
Tool Provider

Custom Computer Services, Inc. proudly supports the Microchip brand with highly optimized C compilers and embedded software development tools.

1 UNPACKING AND INSTALLATION

Inventory

☐ Use of this kit requires a PC with Windows 95, 98, ME, NT, 2000 or XP. The PC must have a spare 9 pin serial port, a CD-ROM drive and 75 MB of disk space.

☐ The diagram on the following page shows each component in this kit. Make sure all items are present.

Software

☐ Insert the CD into the computer and wait for the install program to start. If your computer is not set up to auto-run CDs, then select **My Computer** and double-click on the CD drive.

☐ Click on **Install** and use the default settings for all subsequent prompts. Click NEXT, OK, CONTINUE…as required.

☐ Identify a directory to be used for the programs in this booklet. The install program will have created an empty directory **c:\program files\picc\projects** that may be used for this purpose.

☐ Select the compiler icon on the desktop. In the PCW IDE click **Help>About** and verify a version number. This is shown for the IDE and/or PCM. This ensures the software is installed properly. Exit the software.

Hardware

☐ Connect the PC to the ICD using the USB cable.(1) Connect the prototyping board (10) to the ICD using the modular cable. Plug in the AC adaptor to the power socket and plug it into the prototyping board. The first time the ICD-U is connected to the PC Windows would detect new hardware. Install the USB driver from the CD or website using the new hardware wizard. The driver needs to be installed properly before the device can be used.

☐ The LED should be red(2) on the ICD-U to indicate the unit is connected properly.

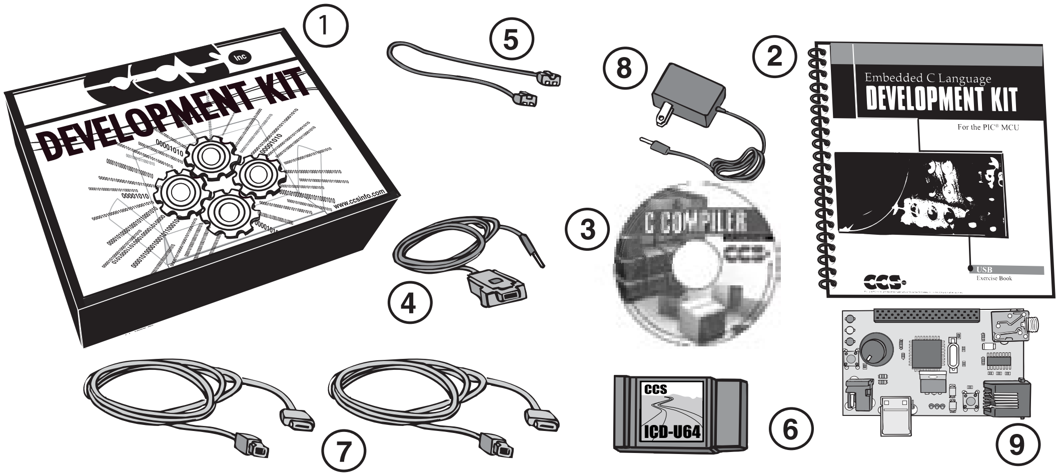
☐ Run the Programmer Control Software by clicking on the CCSLOAD icon on the desktop. Use CCSLOAD Help File for assistance.

☐ The software will auto-detect the programmer and target board and the LED should be illuminated green. If any errors are detected, go to Diagnostic tab. If all tests pass, the hardware is installed properly.

☐ Disconnect the hardware until you are ready for Exercise 3. Always disconnect the power to the prototype board before connection/disconnecting the ICD or changing the jumper wires to the prototype board.

(1) ICS-S40 can also be used in place of ICD-U. Connect it to an available serial port on the PC using the 9 pin serial cable. There is no driver required for S40.

(2) ICD-U40 units will be dimly illuminated green and may blink while connecting.



① Storage box
② Exercise Booklet
③ CD-ROM of Compiler Software
④ Serial PC to Prototype Cable
⑤ Modular Cable (ICD to Prototyping Board)
⑥ ICD unit allows programming and debugging
PICmicro® MCU parts from a PC
⑦ Two USB Cables (PC to Prototyping Board; ICD to PC)
⑧ AC Adaptor (9VDC)
⑨ USB Prototyping Board

CD-Rom of Examples

☐ This CD-ROM contains example source code for PIC18F4550, PC source code and Windows operating system drivers.

☐ Unzip USB.zip and unpack it to C:\USBPIC

2

USING THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Editor

☐ Open the PCW IDE. If any files are open, click **File>Close All**

☐ Click **File>Open>Source File.** Select the file: **c:\program files\picc\examples\ex_stwt.c**

☐ Scroll down to the bottom of this file. Notice the editor shows comments, preprocessor directives and C keywords in different colors.

☐ Move the cursor over the **Set_timer0** and click. Press the F1 key. Notice a help file description for set_timer0 appears. The cursor may be placed on any keyword or built-in function and F1 will find help for the item.

☐ Review the editor special functions by clicking on **Edit**. The IDE allows various standard cut, paste and copy functions.

☐ Review the editor option settings by clicking on **Options>Editor Properties**. The IDE allows selection of the tab size, editor colors, font and many more. Click on **Options>Toolbar** to select which icons appear on the toolbars.

Compiler

☐ Use the drop-down box under Compile to select the compiler. CCS offers different compilers for each family of Microchip parts. All the exercise in this booklet are for the PIC18F4550 chip, an 16-bit opcode part. Make sure **PCH 16 bit** is selected in the drop-down box under the **Compiler** tab.

☐ The main program compiled is always shown in the bottom of the IDE. If this is not the file you want to compile, then click on the tab of the file you want to compile. Right click into editor and select **Make file project**.

☐ Click **Options>Project Options>Include Files…** and review the list of directories the compiler uses to search for included files. The install program should have put two directories in this list: devices and drivers.

☐ Normally the file formats need not be changed and global defines are not used in these exercises. To review these setting, click **Options>Project Options>Output Files** and **Options>Project Options>Global Defines**.

☐ Click **Compile>Compile** or the compile icon to compile. Notice the compilation box shows the files created and the amount of ROM and RAM used by this program. Press any key to remove the compilation box.

Viewer

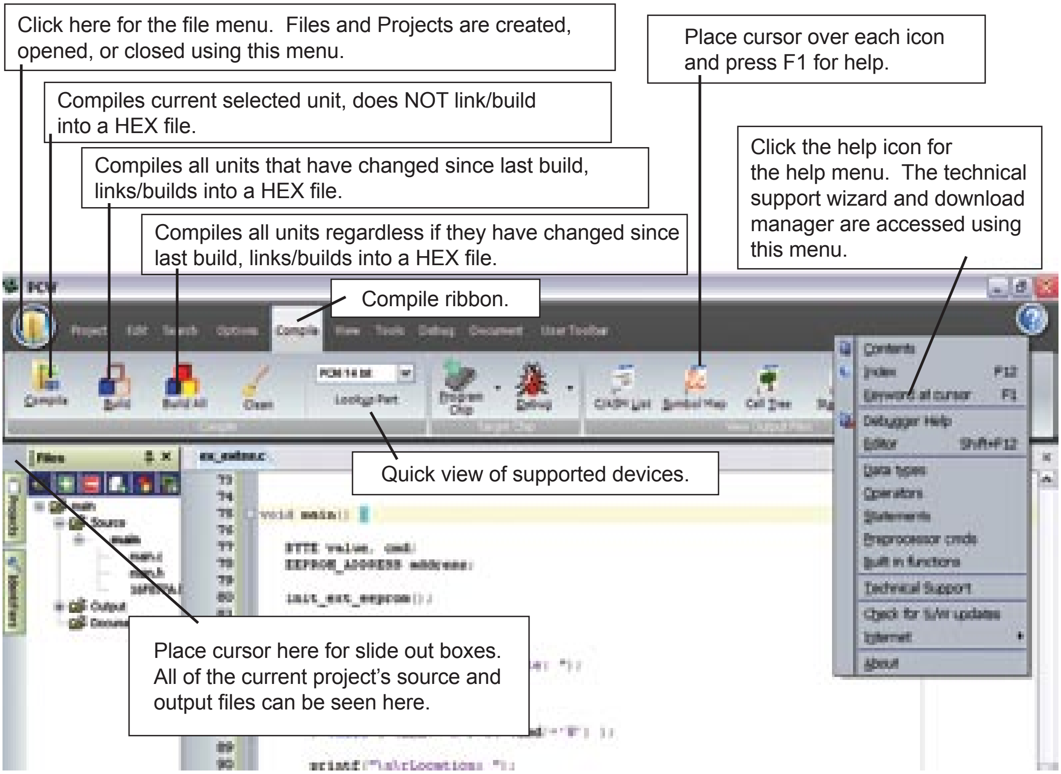
☐ Click **Compile>Symbol Map**. This file shows how the RAM in the micro-controller is used. Identifiers that start with @ are compiler generated variables. Notice some locations are used by more than one item. This is because those variables are not active at the same time.

☐ Click **Compile>C/ASM list**. This file shows the original C code and the assembly code generated for the C. Scroll down to the line:

```
int_count=INTS_PER_SECOND;
```

☐ Notice there are two assembly instructions generated. The first loads 4C into the W register. INTS_PER_SECOND is #defined in the file to 76. 4C hex is 76 decimal. The second instruction moves W into memory. Switch to the Symbol Map to find the memory location is where int_count is located.

☐ Click **View>Data Sheet**, then OK. This brings up the Microchip data sheet for the microprocessor being used in the current project.



Click here for the file menu. Files and Projects are created, opened, or closed using this menu.

Compiles current selected unit, does NOT link/build into a HEX file.

Compiles all units that have changed since last build, links/builds into a HEX file.

Compiles all units regardless if they have changed since last build, links/builds into a HEX file.

Compile ribbon.

Place cursor over each icon and press F1 for help.

Click the help icon for the help menu. The technical support wizard and download manager are accessed using this menu.

Quick view of supported devices.

Place cursor here for slide out boxes. All of the current project's source and output files can be seen here.

3 COMPILING AND RUNNING A PROGRAM

☐ Open the PCW IDE. If any files are open, **click File>Close All**

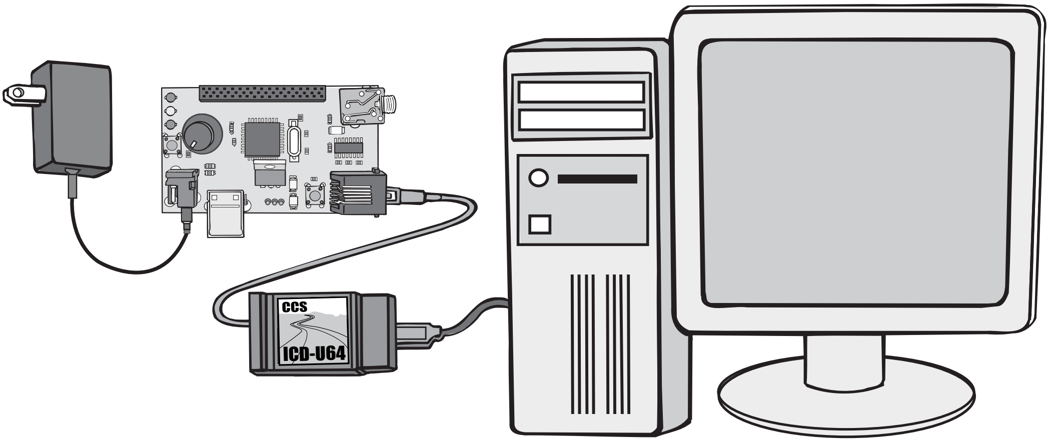
☐ Click **File>New>Source File** and enter the filename **EX3.C**

☐ Type in the following program and **Compile.**

```
#include <18F4550.h>
#fuses  HS,NOLVP,NOWDT,PUT
#use delay (clock=20000000)

#define GREEN_LED PIN_A5

main () {
      while (TRUE) {
            output_low (GREEN_LED);
            delay_ms (1000);
            output_high (GREEN_LED);
            delay_ms (1000);
      }
}
```

NOTES

- The first three lines of this program define the basic hardware environment. The chip being used is the PIC18F4550, running at 20Mhz.
- The #define is used to enhance readability by referring to GREEN_LED in the program instead of PIN_B5.
- The “while (TRUE)” is a simple way to create a loop that never stops.
- Note that the “output_low” turns the LED on because the other end of the LED is +5V. This is done because the chip can tolerate more current when a pin is low than it can source when it is high.
- The “delay_ms(1000)” is a one second delay (1000 milliseconds).



☐ Connect the ICD to the Prototyping board using the modular cable, and connect the ICD to the PC. Power up the Prototyping board.

☐ Click **Debug>Enable Debugger** and wait for the program to load.

☐ If you are using the ICD-U40 and the debugger cannot communicate to the ICD unit go to the debug configure tab and make sure ICD-USB from the list box is selected.

☐ Click the green go icon: GO

☐ Expect the debugger window status block to turn yellow indicating the program is running.

☐ The green LED on the Prototyping board should be flashing. One second on and one second off.

☐ The program can be stopped by clicking on the stop icon: S

Before Moving On:

Remove the 9V DC power supply and change the jumper on the prototype board to configure the device for bus power.

4 USB OVERVIEW

Universal Serial Bus (USB) is an interface designed for personal computers. USB employs a master/slave protocol where the PC is the master and controls and schedules all USB communications. The USB devices are slaves that respond to host commands. USB devices cannot talk to other USB devices, only to the master. USB allows up to 127 devices to be connected to one PC.

Modern day PCs include a host controller, which provide the root USB hub from which you can attach your USB devices. To allow more connections, you may attach another external hub to the PC, which add more USB ports. USB hubs (including the host controller in your PC) can provide up to 500mA of power (at 5V) to a USB device.

USB 1.x provides for two bandwidths, low-speed at 1.5Mbit/s and full-speed at 12Mbit/s. USB 2.x provides for fast-speed at 480Mbit/s. These are maximum bus speeds, under ideal conditions the actual performance will be much lower. Since many devices can be attached to the PC at once it is the host controller’s duty to allocate the bandwidth to each device. For example, if there are two full-speed devices connected to the PC the host controller may allocate 10Mbit/s to one device and 2Mbit/s to the other device. The bus always uses the slowest speed possible, for example if you connect a low-speed and a full-speed device to the PC then both devices must use the low-speed. (Some hubs may get around this problem, but it is something to be aware of).

Data on the USB flows in two directions relative to the PC, incoming from a device and outgoing to a device. OUT messages refer to the outgoing message from the PC, and IN messages refer to the incoming message from a device to the PC. Any time there is reference to an OUT or IN message, always use the direction relative to the PC.

Data transmission between the host and the devices uses endpoints. An endpoint can be considered a logical sink or source of data on the USB device. Each endpoint can be configured differently for bandwidth needed, max transmission size, etc. Each device always has one bidirectional endpoint, labeled endpoint 0, which is reserved for control transfers. Devices may have up to 30 endpoints, 15 for receiving and 15 for transmitting.

At the lowest level, an endpoint is a buffer that stores incoming or outgoing messages. Incoming messages (Device to PC) will sit in the buffer until the PC polls the device. Outgoing messages (PC to Device) will sit in the buffer until the device has had a chance to read the data, at which point the endpoint will be ready to accept more data. If the PC attempts to send data to an endpoint that still has data in the buffer, the device will send a NAK and the PC will retry later.

There are four different USB message types:

- *Control Transfers* – Control transfers are used to transfer small blocks of information. Each device must support control transfers. Control messages are guaranteed to have 10% of the USB bandwidth. Control transfers are mostly used to send SETUP tokens, which the PC uses to set and get configuration from the USB device. The following are SETUP request types:

Request	Req. Number	Description
Get_Status	0x00	Host requests current set feature
Clear_Feature	0x01	Host requests to disable feature on device, interface or endpoint
Set_Feature	0x03	Host requests to enable a feature on device, interface or endpoint
Set_Address	0x05	Host specifies an address for USB device, values range from 1 to 127
Get_Descriptor	0x06	Host requests a descriptor
Set_Descriptor	0x07	Host sets a descriptor (Optional)
Get_Configuration	0x08	Host requests current configuration
Set_Configuration	0x09	Host specifies which configuration to use
Get_Interface	0x0A	Host requests current interface
Set_Interface	0x0B	Host specifies which interface device is supposed to use
Synch_Frame	0x0C	Device sets and reports an endpoint's synch frame

- *Bulk Transfers* – Bulk transfers are used to send large blocks of data, or to send non-periodic data. Bulk transfers is the most ideal transfer method for large blocks of data because this transfer is deferred to other message types to prevent the bus bandwidth from getting full. Also, when the bus is idle bulk transfers are allowed to use up to 95% of the bus for messages, which is more than any other message type. Since no bandwidth is reserved for bulk transfers, this is not ideal for sending data at a timely rate.
- *Interrupt Transfers* – Interrupt transfers are used to transfer small blocks of information quickly. Interrupt does not mean it will interrupt the microcontroller or PC, it means that the PC tries to send/receive the data with minimal delay. Interrupt transfers can use up to 90% of the bus bandwidth.
- *Isochronous Transfers* – Isochronous transfers are used to transfer blocks of information at a periodic rate of time. The opposite of bulk transfers, isochronous transfers are sent at a specified period. Since data must be sent at a specified period, there is no error checking because there would be no time to retransmit a failed message. Isochronous messages can take up to 90% of bus bandwidth.

Each USB device has several descriptors which describe the device and fall under the following categories: device descriptors, configuration descriptors, interface descriptors, endpoint descriptors and string descriptors. For more information about these descriptors please see Chapter 10.

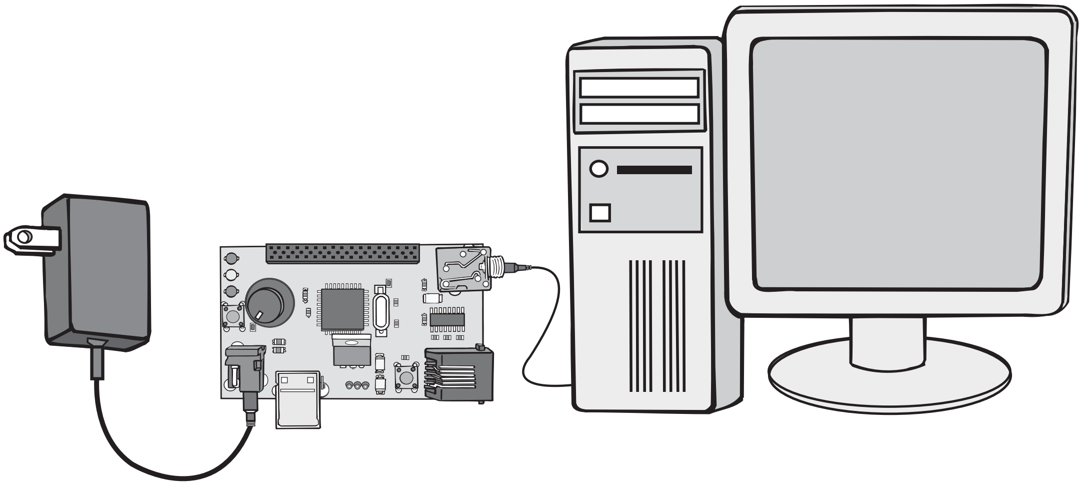
USB is a very in-depth subject, detailing it all is not in the scope of this tutorial. Please see other documentation for more USB details.

5 USB PROTOTYPE BOARD

The CCS USB prototype board provides a PIC18F4550. The PIC18F4550 includes an internal full speed (12Mbit/s) USB peripheral, support for 16 bi-directional endpoints, and 768 bytes of RAM reserved for endpoint buffers.

The USB Development Kit contains a PC cable, which is an RS232 cable with a modified connector on the prototyping board. Connect this cable to the prototyping board and the PC, and then run a serial program. PCW and PCWH includes one which can be opened by selecting **Tools > Serial Port Monitor**. Any instance of printf(), getc(), etc uses this RS232 connection.

The USB prototyping board provides a jumper for configuring between a self powered USB device or a BUS powered device. If a self powered device is jumped, the 9V DC power supply must also be connected to the device. The prototyping board has a push button switch on RA4, a potentiometer on RA0 and three LEDs on RA5, RB4 and RB5.



6 CCS USB API

CCS provides, for all CCS C Compiler customers, an API for developing USB applications on the PIC. There are several files associated with the USB API that are used:

- pic18_usb.h - Provides hardware layer functions for Microchip PIC18 microcontrollers that have a built-in USB peripheral, such as the PIC18F4550. This provides functions for setting up the peripheral, sending and receiving packets,etc.
- usbn960x.c - Provides hardware layer functions for National's USB960x line of USB peripherals. This provides functions for setting up the peripheral, sending and receiving packets, etc. Since this tutorial focuses on the development kit with a PIC18F4550, this driver will not be used in this tutorial.
- pic_usb.h - Provides hardware layer functions for Microchip 14-bit microcontrollers that have a built-in USB peripheral, such as the PIC16C745 or PIC16C765. This provides functions for setting up the peripheral, sending and receiving packets, etc. Since this tutorial focuses on the development kit with a PIC18F4550, this driver will not be used in this tutorial.
- usb.c - Using an interrupt driven by the hardware,this provides token handler support. The bulk of this file automatically handles and responds to the SETUP tokens used by the PC to configure the USB device, also known as "Chapter 9 Requests." ("Chapter 9 Requests" because Chapter 9 of the official USB documentation details SETUP messages.)
- usb.h – Prototypes, global defines and conditional compile statements used in conjunction with usb.c There are some global definitions made here that can be changed to alter the USB API to an application.
- usb_desc_*.h - Provides example device, configuration, interface, endpoint and string descriptors needed to describe the USB devices used in the examples. The SETUP handler in usb.c will use these descriptors to answer Get_Descriptor requests. Since USB descriptors are application dependent, there is a different descriptor for each example program.
- ex_usb_*.c – Example programs that demonstrate common USB applications such as mice, virtual COM ports and generic vendor-specific protocols.

7 ENUMERATION

☐ Enumeration is the process in which operating systems, such as Microsoft Windows, learn about the USB device and load the appropriate driver. Below is a summary of the process an operating system takes to enumerate a device:

1. Device is plugged into USB port. (Or the system powers up with the device already plugged into the USB port).
2. The system detects the device because of the voltage change on the D+ and D- lines. (The D+ and D- are the differential data lines of a USB cable). It also detects the device's speed based on the voltages represented on D+ and D-.
3. System sends a reset signal by holding D+ and D- low for about 10 milli-seconds. The device will now be set to address 0.
4. The system sends a Get_Descriptor request to learn the max packet size of Endpoint 0 of the device at address 0. Because only one device can enumerate at once, only one device will be at address 0. All enumeration communications use Endpoint 0.
5. The system assigns an address between 1 and 127 for the device. No two devices can have the same address.
6. The system uses a number of Get_Descriptor requests to read all of the configuration descriptors present on the USB device.
7. Using the configuration descriptors, the system loads the proper device drivers.
8. The system assigns the USB device a configuration. Since many devices can have more than one configuration, the device needs to be told by the system what configuration to use. (See Chapter 9 of this tutorial for more information). At this point the USB device is operational.

The bulk of the enumeration processes is Step 6 and Step 8, where the system uses control transfers on Endpoint 0 to send "Chapter 9 Requests" to the device. The CCS USB stack automatically processes the "Chapter 9 Requests."

The following example simply configures the PIC to be a USB device. When the PC connects to the PIC via USB, the operating system will enumerate the USB device. Remove the 9V DC power supply, if connected, and insert a jumper to configure the device to be self powered.

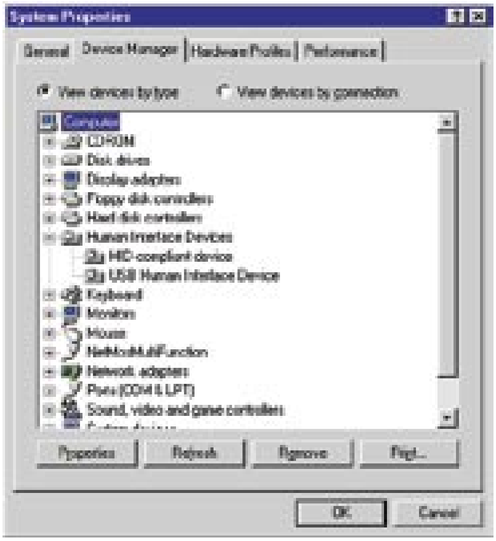
☐ Type in the following program, named ex7.c

```
#include <18F4550.h>
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)

#define LED1 PIN_A5
#define LED2 PIN_B4
#define LED3 PIN_B5
#DEFINE BUTTON PIN_A4
#define LED_ON output_low
#define LED_OFF output_high

#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#define USB_EP1_RX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE 8

#include <pic18_usb.h>
#include <usb_desc_hid.h>
#include <usb.c>

void main(void) {
   LED_ON(LED1);
   LED_OFF(LED2);
   LED_OFF(LED3);

   usb_init();

   while (TRUE) {
      if (usb_enumerated())
         LED_ON(LED3);
      else
         LED_OFF(LED3);
   }
}
```

- ☐ While this program is running, LED1 will light indicating the unit is powered.
  A few seconds later LED3 will light signifying the device has successfully been enumerated by the PC.
- ☐ If a HID device has never been plugged into the PC, there may be a prompt to install HID drivers on the operating system. (See Appendix-A for help with installing HID drivers.)

ENUMERATION, CONTINUED

- ❑ If the HID drivers are installed and the USB prototyping board is working correctly, Windows Device Manager should list *HID-compliant device* under *Human Interface Devices.*



- ❑ When the USB prototyping board is connected to the PC with the USB cable, LED3 turns on. After un-configuring the USB device, LED3 should turn off. A PC USB protocol analyzer can be used to configure and un-configure the device.

- ❑ If LED3 does not turn on or the HID-compliant device under Windows Device Manager does not appear, one of the following may be wrong:
  - The USB cable is not connected from the PC to the USB Prototyping board.
  - The USB Prototyping board is not powered up. Check to ensure the jumper on the prototyping board is set to bus power mode.
  - The microcontroller on the USB Prototyping board was not programmed correctly with an ICD or other programming device. See Chapter 3.
  - USB is not working correctly, or disabled on the PC. Check the Operating System and BIOS settings.

NOTES

- PIC18-usb.h includes all the hardware layer functions needed to send and receive USB packets using the PIC18F4550
- usb.c provides all the functions needed to respond to control messages on endpoint 0. While the example provided looks simple, a lot of USB handling is done during the USB interrupt behind the scenes that does not require user intervention.
- usb_init() initializes the PIC18F4550 device and must be called at power-up before any other USB functions are used.
- usb_init() enables interrupts. During the interrupt, the USB firmware processes IN and OUT messages and handles any control messages. Keep this in mind when writing any time sensitive code as the USB firmware may interrupt it.
- The CCS USB firmware defaults the code to be a HID device. Windows and many popular operating systems should have HID drivers already installed or on the operating system install CD. If needed, the USB and HID drivers may need to be installed the first time the device is plugged in. Follow the documentation provided by the operating system.
- usb_enumerated() returns TRUE once the PC has enumerated the device and specified which configuration to run.
- Many USB transactions require low latency to operate correctly, so traditional debugging methods may not be applicable since the delays introduced through debugging may break the USB code. For these reasons the best way to debug a USB device is to get a protocol analyzer. USB analyzers in hardware may be expensive, but a cheaper alternative might be in software.
- USB_EP1_TX_ENABLE, USB_EP1_RX_ENABLE, USB_EP1_TX_SIZE and USB_EP1_RX_SIZE dynamically configure the USB stack to enable endpoint 1 for IN and OUT transfers, and allocate two 8 byte buffers for them. Although we are not using endpoint 1, we have created a USB HID device that can send and transmit messages on endpoint 1. This will be covered in Chapter 9 of this tutorial. These endpoints have enabled even though they are not being used because the simple descriptor file included (ex_usb_hid.h) requires them to be enabled.

Before Moving On:

For future examples copy the first 10 lines in this ex7.c into an include file named CCSUSB.H.

8

CONNECTION SENSE

☐ Chapter 7 showed a simple demonstration of creating a USB device on a Microchip microcontroller, and connecting it to the PC via USB. For that example, the jumper on the prototyping board was configured to be bus powered; meaning the unit was powered via the 5 volts provided by the USB bus. Move the jumper so the unit is self-powered and connect a 9V DC power supply to the unit. Connect a USB cable to the device, and upon successful enumeration LED3 should turn green, meaning it is enumerated. However, by disconnecting the USB cable, LED3 stays green. Without connection sense the unit does not know if the unit is still connected, and it can only assume that it is.

☐ The following example uses the connection sense code in the USB stack to un-configure the device when it is no longer connected.

☐ Type in the following program, named ex8.c.

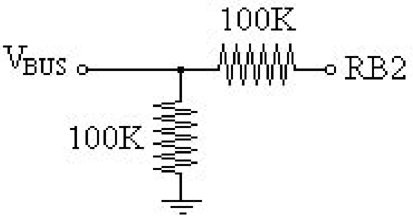
☐ Compile and run the program.

```
#include “CCSUSB.H”

#define USB_CON_SENSE_PIN  PIN_B2

#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#define USB_EP1_RX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE 8

#include <pic18_usb.h>
#include <usb_desc_hid.h>
#include <usb.c>

void main(void) {
    LED_ON(LED1);
    LED_OFF(LED2);
    LED_OFF(LED3);

    usb_init_cs();

    while (TRUE) {
        usb_task();

        if (usb_attached())
            LED_ON(LED2);
        else
            LED_OFF(LED2);

        if (usb_enumerated())
            LED_ON(LED3);
        else
            LED_OFF(LED3);
    }
}
```

☐ The example operates like the example in chapter 7, but code has been written to light LED2 if the microcontroller is connected to USB.

NOTES

- usb_init_cs() is similar to usb_init(), but usb_init_cs() will not enable the USB peripheral and will not enable USB interrupts.
- usb_task() keeps an eye of the connection sense pin. If the USB peripheral is disabled and the unit is connected to USB, usb_task() enables the USB peripheral and enables the USB interrupts. If the USB peripheral is enabled and the unit is dettached from USB, usb_task() disables the USB peripheral and disables the USB interrupts.
- When writing code that uses connection sense, periodically call usb_task(). It is best to put it in the main loop, as this example has done.
- USB_CON_SENSE_PIN configures the pin associated with connection sense. On the USB Development kit sold by CCS RB2 is the connection sense pin, and is connected as follows:



Before Moving On:

The remaining examples will use bus power. Connection sense will not be necessary. Remove the 9V DC power supply and change the jumper on the prototype board to configure the device for bus power.

9

HUMAN INTERFACE DEVICES

☐ USB devices require a device driver installed on the operating system before the PC can talk to the USB device. Human Interface Device, or HID, devices were the first USB devices to have full support in Microsoft Windows and the drivers are built into the system. HID was designed as a general purpose class to support any number of human interface devices, but it can be used for any purpose as long as your design can function in the limits of HID. Since the drivers are built into the operating system, it is great way to develop simple USB devices quickly.

☐ HID devices have the following limitations:

- Data is sent using control or interrupt transfers. HID cannot use bulk or isochronous transfers.
- A full-speed device can only send 64-bytes per transaction. A low-speed device can only send 8-bytes per transaction.
- A report (message) cannot be larger than 255 bytes.
- A full-speed device can have no more than 1 transaction per 1 millisecond (or 64,000 bytes/second). A slow-speed device can have no more than 1 transaction per 10 milliseconds (or 800 bytes/second)
- No guaranteed rate of transfer.

☐ To configure a device to be a HID device, the descriptors must be configured to describe the device as a HID device. Also, two new descriptors must be added: a HID class descriptor and the HID report descriptor. The HID class descriptor documents information about the HID device, such as country code and the size of the report descriptor. The HID report descriptor explains the format of all incoming and outgoing messages so the general purpose HID device driver knows how to handle the data. See Chapter 10 of this tutorial book for more information about descriptors.

☐ The following example configures the microcontroller to be a HID device, which will transmit two bytes to the microcontroller at a constant interval. The PIC can also receive two bytes from the PC. The PC will use the standard Windows HID device driver to communicate with the device, and an example CCS Windows application displays the data.

☐ Type in the following program, named ex9.c.

☐ Compile and run the program.

```
#include “CCSUSB.H”

#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#define USB_EP1_RX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE 8

#include <pic18_usb.h>
#include <usb_desc_hid.h>
#include <usb.c>

void main(void) {
    int8 delay=0;
    int8 out[2];
    int8 in[2];

    LED_OFF(LED1);
    LED_OFF(LED2);
    LED_OFF(LED3);

    setup_adc(ADC_CLOCK_INTERNAL);
    setup_adc_ports(AN0);
    set_adc_channel(0);

    usb_init();

    while (TRUE) {
        if (usb_enumerated()) {
            LED_ON(LED1);

            delay++;
            if (delay>=250) {
                delay=0;
                out[0]=read_adc();
                out[1]=!input(BUTTON);
                usb_put_packet(1,out,2,USB_DTS_TOGGLE);
            }

            if (usb_kbhit(1)) {
                usb_get_packet(1,in,2);
                if (in[0]) {LED_ON(LED2);} else {LED_OFF(LED2);}
                if (in[1]) {LED_ON(LED3);} else {LED_OFF(LED3);}
            }

            delay_ms(1);
        }
        else
            LED_OFF(LED1);
    }
}
```

9 HUMAN INTERFACE DEVICES, CONTINUED

☐ Run HIDDEMO.EXE, which is also included with this tutorial. HIDDEMO.EXE should be located in C:\USBPIC (or the location where the USB.ZIP was extracted to). HIDDEMO is an example USB application that uses the default HID drivers to communicate with the CCS USB Prototyping board running ex9.c. In HIDDEMO, select **File -> Select Device** from the toolbar.. Another window will pop-up. On this new window highlight **CCS HID Demo** and press the **Select** button. On the HIDDEMO window now press the **Establish Connection** button.

☐ The right side of the HIDDEMO application represents data that the USB Prototyping board is sending. By moving the potentiometer on the prototyping board, the A/D voltage displayed in HIDDEMO should change as a result. Also, pressing the button next to the LEDs should cause the Button State icon in the HIDDEMO to change.

☐ The left side of the HIDDEMO application represents messages that can be sent to the USB prototyping board. This pane allows for the LEDs on the prototyping board to be turned on and off by changing the radio buttons marked ON and OFF.

NOTES

- This example sends two bytes and receives two bytes. This is in accordance with the HID report descriptor, which details the format of the messages the HID device driver expects to receive and transmit. The default descriptors provided by CCS are located in usb_desc_hid.c. If different amounts of data are desired to be sent or received, the HID report descriptor must be changed.
- The source code for HIDDEMO is also provided on the examples disk to provide an example of how to develop applications on the PC side.
- Usb_kbhit(endpoint) returns true if there is USB data in the receive buffer for that endpoint. Each endpoint has its own buffer.
- usb_get_packet(endpoint, dest, len) copies the data in the USB receive buffer to dest, and then marks the receive buffer as ready to receive more data. Until you get the packet out of the buffer, any attempt by the PC to write more data to this endpoint will result in a NAK packet and the PC will try again. If using an interrupt control method, the PC will try again in the next polling interval as specified in the endpoint descriptor.

NOTES

- Usb_put_packet(endpoint, source, len, toggle) copes data from source into the USB transmit buffer. The data will sit in the USB transmit buffer until the host PC retrieves it. Usb_put_packet() returns TRUE if the data was copied successfully to the buffer, and will return FALSE if the data was not copied successfully because there is still data in the buffer waiting to be transmitted.
- Usb_put_packet() transmits just one packet to the host PC. A packet is not always a full message. If the message is larger than the maximum packet size defined in the descriptor, the message must be sent over multiple packets. To act as an end of message marker, a zero length packet must be sent if the last packet is equal to the size of the maximum packet length.

☐ If a message is smaller than the packet size of this endpoint, USB_PUT_PACKET() needs to be called only once. If a message is the same size or larger of the maximum packet size of the endpoint, then USB_PUT_PACKET() must be used to send a 0 length end of message marker.

For example, if maximum packet size is 8:

Sending a 4 byte message

Packet 1	B1	B2	B3	B4					Len = 4

Sending an 8 byte message

Packet 1	B1	B2	B3	B4	B5	B6	B7	B8	Len = 8
Packet 2									Len = 0

Sending a 10 byte message

Packet 1	B1	B2	B3	B4	B5	B6	B7	B8	Len = 8
Packet 2	B9	B10							Len = 2

10 HID MOUSE

☐ Earlier in this tutorial Descriptors were briefly mentioned as the mechanism that the PC Operating System uses to learn about the USB device. There are six different kinds of Descriptors:

Device	Only one Device descriptor is present for each USB device. Holds configuration information such as what version of USB (1.x, 2.x, etc) this device runs under, pointers to String descriptors, Vendor ID, Product ID and the number of Configuration Descriptors
Configuration	Contains information about power usage (if powered over USB bus), and contains one or more Interface Descriptor. There may be more than one Configuration descriptor for more than one configuration mode.
Interface	Contains information about how many endpoints this interface has, what class this interface is, and if this interface supports Boot protocol. (Boot protocol lets the BIOS use a Mouse or Keyboard without the operating system loading drivers)
Class	Contains information about what class this interface is, number of report descriptors, and length of report descriptor. Most common Class is the HID class.
Endpoint	Contains information about the endpoint, such as its address, transfer type, max packet size and polling interval.
String	A Unicode string that acts as a description for another descriptor. For example, the Device descriptor has a pointer to a device name descriptor which Microsoft Windows displays in the Add Device Wizard.
Report	Contains information about how data sent and received by this device should be handled by the class driver on the host PC's operating system. In the case of HID, tells the HID driver on the PC what kind of data the device is sending.

☐ A USB device can contain several configurations, with each configuration needing a different Configuration Descriptor. Each Configuration Descriptor can then have more than one interface, with each interface requiring its own Interface Descriptor. This descriptor hierarchy branches down to the Endpoint descriptor.

☐ Here is an example descriptor hierarchy:

- Device
- Config 1 (Max Power)
  - Interface 1 (HID Mouse)
    - Class Descriptor
    - Endpoint Descriptor (Address 0x81, IN to PC)
    - HID Report Descriptor
  - Interface 2 (HID Keyboard)
    - Class Descriptor
    - Endpoint Descriptor (Address 0x82, IN to PC)
    - HID Report Descriptor
  - Interface 3 (Keyboard/Mouse LED Control)
    - Endpoint Descriptor (Address 0x01, OUT to Device)
    - Endpoint Descriptor (Address 0x83, IN to PC)
- Config 2 (Low Power)
  - Interface 1 (Mouse)
    - Class Descriptor
    - Endpoint Descriptor (Address 0x81, IN to PC)
    - HID Report Descriptor

☐ This USB device has two Config Descriptors for two possible configurations: low power mode and full power mode. Since a USB Hub has to allocate power to many devices it is common for USB devices to include more than one power mode configuration in the event the HUB runs out of power. The USB Hub knows the power needs for each configuration because the power need is detailed in the Config Descriptor. It is the device's job, and therefore the firmware's job, to know what configuration the USB Hub puts the device into and act accordingly.

☐ The maximum power configuration has three integrated interfaces: A HID keyboard, a HID mouse and LED control. Each interface has its own Interface Descriptor, and each Interface Descriptor than has its own Endpoint Descriptor. Since the Mouse and Keyboard are HID devices, they require Class and HID Report descriptors.

☐ Each endpoint can be configured for different transfer methods; for example the Mouse endpoint may be isochronous, the keyboard endpoint could be bulk, one of the LED control endpoints could be interrupt, and the other LED control endpoint could be bulk. The physical endpoint address 0x01 points to the OUT endpoint 1, while 0x81 is for IN endpoint 1.

☐ The low power configuration has only a HID Mouse interface. Since the other interfaces are not used, the firmware can turn them off in an attempt to save power.

10 HID MOUSE, CONTINUED

☐ Although this hierarchy is complex, most applications will only use one configuration and one interface. All examples in this book use one configuration and one interface.

☐ For an example of a two interface device, see ex_usb_kbmouse.c in the examples directory where the CCS Compiler, is installed.

☐ The goal in this chapter is to create a USB Mouse using the HID drivers. In the previous chapter a generic device was created that sent two bytes of data over USB, which could then be read on the host PC by using the standard HID drivers. Since the USB HID Descriptor details the format of the data to the HID driver, changing the previous example to a Mouse mostly requires a change to the HID Descriptor.

☐ Here is the old HID Report Descriptor side-by-side with the new HID Report Descriptor which will tell the HID driver to format received data as mouse data:

Example 9

```
0x06, 0x00, 0xFF, // Usage Page = Vendor
0x09, 0x01,       // Usage = IO device
0xa1, 0x01,       // Collection = Application
0x19, 0x01,       // Usage minimum
0x29, 0x08,       // Usage maximum
0x15, 0x80,       // Logical minimum (-128)
0x25, 0x7F,       // Logical maximum (127)
0x75, 0x08,       // Report size = 8 (bits)
0x95, 0x02,       // Report count = 16 (bits)
0x81, 0x08,       // Input (Data, Var, Abs)
0x19, 0x01,       // Usage minimum
0x29, 0x08,       // Usage maximum
0x91, 0x02,       // Output (Data, Var, Abs)
0xc0              // End Collection
```

USB HID Mouse

```
0x05, 0x01,  // usage page=generic desktop
0x09, 0x02,  // usage=mouse
0xA1, 0x01,  // collection (application)
0x09, 0x01,  // usage=pointer
0xA1, 0x00,  // collection (physical)
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 (data, var, abs)
0x95, 0x01,  // report count (1)
0x75, 0x05,  // report size (5)
0x81, 0x01,  // input (constant)
0x05, 0x01,  // usage page (generic desktop)
0x09, 0x30,  // usage (X)
0x09, 0x31,  // usage (Y)
0x09, 0x38   // usage (wheel
0x15, 0x81,  // logical minimum (-127)
0x25, 0x7F,  // logical maximum (127)
0x75, 0x08,  // report size (8)
0x95, 0x03,  // report count (3)
0x81, 0x06,  // input (data, var, abs)
0xC0,        // end collection (physical)
0xC0         // end collection (application)
```

By using a Vendor Specific usage page, the HID driver ignores received data and expects the Vendor's application to know how to format the data. The input and output data size is then set to two bytes (report size=8bits, report count=2 reports).	This Report descriptor is much more complicated. First it tells the HID Driver that the first byte of data is a bit map of buttons being pressed (bits 0-2 for the buttons, bits 3-7 are not used). Then it says the next three bytes are used for X, Y and wheel position. Since the whole report is of Usage (Mouse) the HID driver passes the data to the operating system to use as the mouse.

- ☐ HID Report Descriptors are fairly complex, and can be used to specify devices such as keyboards, mice, point-of-sale cash registers and vendor specific applications. This topic is out of the scope of this tutorial, and there are several more in depth resources that are available.

10 HID MOUSE, CONTINUED

☐ Type in the following program for ex10.c

```
#include “CCSUSB.H”

#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#include <pic18_usb.h>
#include <usb_desc_mouse.h>
#include <usb.c>

void main(void) {
    #define MOUSE_SEQUENCE_STEPS  16
    const char mouse_seq[MOUSE_SEQUENCE_STEPS]=
        {0, 1, 3, 4, 4, 4, 3, 1, 0, -1, -3, -4, -4, -4, -3, -1};

    int8 out_data[4]={0,0,0,0};
    int8 x_seq=0;  int8 y_seq=MOUSE_SEQUENCE_STEPS/4;
    int8 count=0;

    LED_ON(LED1);
    LED_OFF(LED2);
    LED_OFF(LED3);

    usb_init();

    while (TRUE) {
        if (usb_enumerated()) {
            out_data[1]=mouse_seq[x_seq];
            out_data[2]=mouse_seq[y_seq];

            if (usb_put_packet(1,out_data,4,USB_DTS_TOGGLE))
                count++;

            if (count > 10) {
                if (++x_seq>=MOUSE_SEQUENCE_STEPS) {x_seq=0;}
                if (++y_seq>=MOUSE_SEQUENCE_STEPS) {y_seq=0;}
                count=0;
            }

            delay_ms(10);
        }
    }
}
```

☐ Compile and run the program.

☐ This example will cause the mouse cursor to move in a circle. When the USB Prototyping board is connected to the PC for the first time, the operating system should automatically install the HID driver. See Appendix-A if installation problems occur.

NOTES

- usb_desc_mouse.h is a descriptor file supplied by CCS that configures the USB device as a USB HID Mouse. It is these descriptors that tells the operating system how to deal with data that is transmitted by the USB device.
- By changing the HID Descriptor, the whole application could easily be changed to a USB HID Joystick. This would be done by changing the Usage Page to Gamepad, and defining which bytes of the data represent button press and which bytes of data represent X and Y.
- In all HID applications, send/receive as much data as specified in the HID report descriptor. In this example the input report size is four bytes, so the mouse application must always send four bytes. If the mouse application were to send any other amount of data on endpoint 1 the PC would throw the data away. This is true of all HID applications.
- If it is desired to specify several different protocols, for example one protocol where the transmitted size is four bytes and another protocol where the transmitted size is six bytes, you can use the HID report ID. A HID report ID is the first byte in the message that specifies which protocol is being used. For an example, see ex_usb_kbmouse2.c which is in the examples directory of the CCS C Compiler. This example creates a Mouse/Keyboard combo device using two HID report IDs.

11

BULK DEVICES

☐ Bulk devices, such as printers and scanners, send large amounts of data. The benefit of bulk transfers over other transfer methods is that bulk transfers can allocate the highest amount of bandwidth.

☐ Unlike HID, there are no generic bulk device drivers that are installed with operating systems. A generic bulk drivers written by a third party can be found or written on one’s own. CCS has written and provided a bulk driver to use in the following example.

☐ This example sends a 512 byte message to the PC using bulk transfers. The 512 byte message represents a sine wave, which we will view on the PC using CCS’S sample bulk driver and sample PC application.

☐ Type in the following program, named ex11.c,

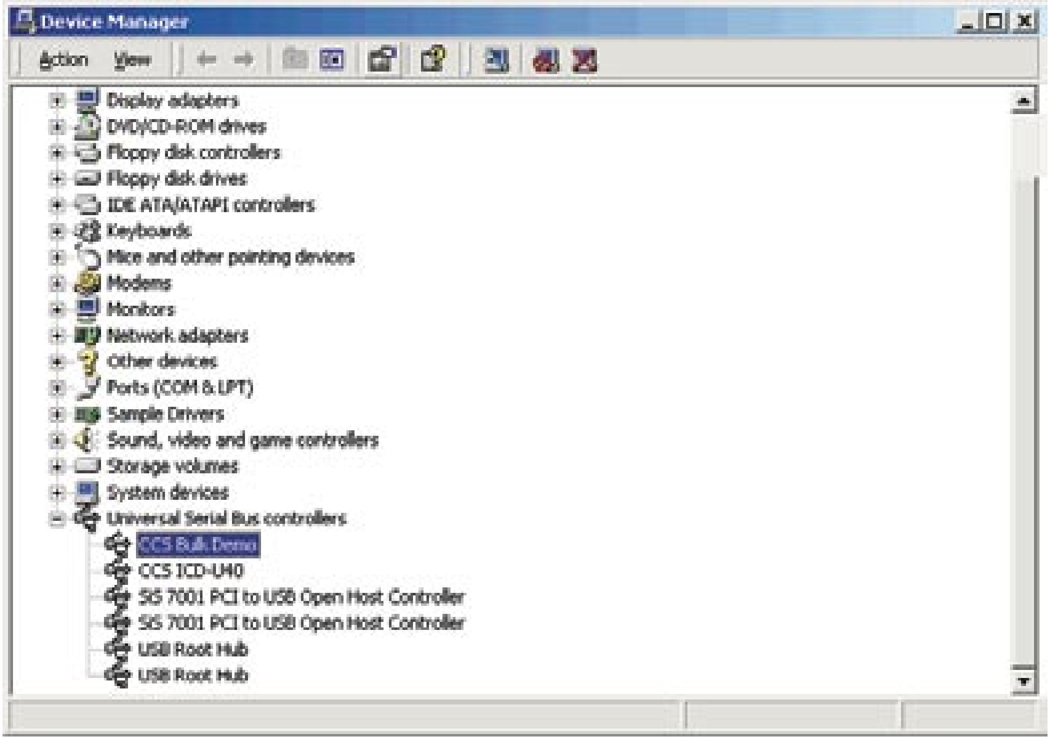
☐ Compile and run the program

```
#include “CCSUSB.H”

#define USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_TX_SIZE    256
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_SIZE    8

#include <pic18_usb.h>
#include <usb_desc_scope.h>
#include <usb.c>

#define OSCDEMO_MESSAGE_SIZE  512

#include <math.h>

int in_data[2], msg[OSCDEMO_MESSAGE_SIZE];
#define threshold    in_data[0]
#define sample_rate  in_data[1]

int read_simulated_adc(void) {
    #define step   (2*3.14/OSCDEMO_MESSAGE_SIZE)
    static float f=0.0;
    int adc, i;

    adc=((sin(f)+1.0)/2.0)*0xFF;
    for (i=0;i<=sample_rate;i++)
        f=f+step;
    if (f>=6.28)
        f=f-6.28;
    return(adc);
}
(continued...)
```

```
(continued...)

void main(void) {
    int16 i=0;
    int8 last_adc, adc, trigger=FALSE;

    LED_ON(LED1);
    LED_OFF(LED2);
    LED_OFF(LED3);

    usb_init();

    while (TRUE) {
        if (usb_enumerated()) {
            LED_ON(LED3);

            adc=read_simulated_adc();
            if ((adc>=threshold)&&(last_adc<threshold)) {trigger=TRUE;}
            if (trigger) {msg[i++]=adc;}
            if (i>=OSCDEMO_MESSAGE_SIZE) {
                usb_puts(1, msg, OSCDEMO_MESSAGE_SIZE, 15);
                trigger=FALSE;
                i=0;
            }
            last_adc=adc;

            if (usb_kbhit(1))
            usb_get_packet(1,in_data,2);
        }
    }
}
```

11 BULK DEVICES, CONTINUED

☐ When this program is run for the first time and connected to the PC, a Windows dialog box should open and ask to install drivers. For this example CCS has provide some sample bulk drivers. See **Appendix-B** for help on installing drivers in Windows 2000/XP. See **Appendix-C** for help on installing drivers in Windows 98/ME. When these drivers are installed and the device enumerates correctly, the device in Windows Device Manager should appear.



- Display adapters
- DVD/CD-ROM drives
- Floppy disk controllers
- Floppy disk drives
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Modems
- Monitors
- Network adapters
- Other devices
- Ports (COM & LPT)
- Sample Drivers
- Sound, video and game controllers
- Storage volumes
- System devices
- Universal Serial Bus controllers
  - CCS Bulk Demo
  - CCS ICD-U40
  - SiS 7001 PCI to USB Open Host Controller
  - SiS 7001 PCI to USB Open Host Controller
  - USB Root Hub
  - USB Root Hub

☐ After the driver is loaded, start OSCOPE.EXE provided with this tutorial. OSCOPE.EXE should be located in C:\USBPIC (or the location where the USB.ZIP on the floppy was extracted to). OSCOPE.EXE uses the CCS sample bulk driver to talk to this example application, and displays the received 512 bytes like an oscilloscope.

☐ In OSCOPE.EXE, use **File->Start** to open communications with the microcontroller. In the application there are two panes. The main pane, on the right, displays the data the PIC is sending. The left pane is a trigger slider. Use the trigger slider to adjust the threshold at which the microcontroller starts sending data. At the bottom of the application, choose between three sample rates to change the frequency of the simulated sampling.

NOTES

- The max packet size of this endpoint is 256 bytes. In order to send a 512 byte message, send two 256 byte packets. A final 0 length packet is sent to indicate and end of message marker. See **Section 9** of this tutorial for examples of multi-packet messages.
- usb_puts() will send multi-packet messages for you. usb_puts() knows how many packets to send for the message because the packet size was defined with USB_EP1_TX_SIZE. If other endpoints were used it would use that endpoint's respective endpoint size definition.
- Windows knows to load the USBDemo.sys drivers for this demo because the .INF file for this driver specifies the Vendor ID and Product ID of the USB device the driver is valid for. The same Vendor ID and Product ID in the .INF file is also in the device descriptor.
- A 256 byte packet is allowed in this example because of using bulk transfers. The previous examples used HID, which required interrupt transfers, and interrupt transfers can only have a max packet size of 64.

12

VIRTUAL COM PORT

☐ To make USB as robust as possible, the USB organization has specified many standard classes and protocols from which to base a device. One standard class, HID, which deals with simple human interface devices was reviewed in Chapters 9 & 10. Another standard class is the Communication Device Class, or CDC, which deals with communication devices such as POTS, Telephony and Ethernet adapters. One sub-class of CDC is an Abstract Control Model Serial Emulation which will create a virtual COM port on a PC, creating a simple USB to UART converter.

☐ Using this CDC class it is easy to adapt previous RS232 applications to USB because upon enumeration the PC assigns a COM port to the USB device from which can be written and read to like previous RS232 legacy devices. Another feature of the CDC class is that CDC drivers are included in many operating systems, and a driver for a specific application does not need to be written.

☐ CCS provides a library, that sits upon the already provided USB stack, to create a Virtual COM port out of a USB device. The following example demonstrates this library.

☐ Type in the following program, named ex12.c.

☐ Compile and run the program.

```
#include “CCSUSB.H”

#include <usb_cdc.h>

void main(void) {
    char c;
    int8 delay=0;

    LED_ON(LED1);
    LED_OFF(LED2);
    LED_OFF(LED3);

    setup_adc(ADC_CLOCK_INTERNAL);
    setup_adc_ports(AN0);
    set_adc_channel(0);

    usb_init();

    while (TRUE) {
        if (usb_enumerated()) {
            LED_ON(LED3);

            if (usb_cdc_kbhit()) {
                c=usb_cdc_getc();
                if (c==’!’)
                    output_toggle(LED2);
            }

            if (++delay>200) {
                delay=0;
                printf(usb_cdc_putc,”\r\nADC = %U”,read_adc());
            }

            delay_ms(5);
        }
    }
}
```

- ☐ When the PC detects the device it will ask for a driver. CCS provides an .INF to use for Windows NT / 2000 / XP, during the install process have the Add Device Wizard search the C:\USBPIC\ directory (or where you un-zipped the USB.ZIP from Chapter 1 of this tutorial). Since the actual drivers are part of the operating system only the .INF file is needed, but it may need to copy driver files from the operating system install CD.
  Note: CCS does do not have an .INF file for Windows 98 / ME
- ☐ After successful enumeration (LED3 will be lit), open a serial terminal program (such as Hyperterminal) and open the COM port of a USB device. You can find the COM port of a USB device by inspecting the Device Manager.
- ☐ Every second the ADC value read on channel 0 will be sent to the COM port.
- ☐ From a serial terminal program, sending a ! character will toggle LED2.

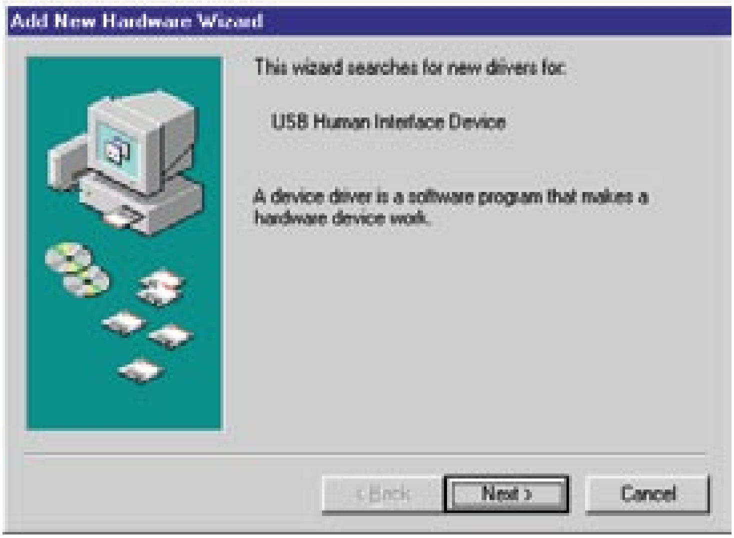
NOTES

- usb_cdc.h will include all the USB stack and descriptors required to configure the USB device for CDC. It also provides all the handler code for normal USB requests and CDC requests, which are processed in the USB interrupt.
- When incoming serial characters are received by the USB stack, they are stored into a RAM buffer. usb_cdc_kbhit() then returns TRUE if there is data in the receive buffer, and usb_cdc_getc() returns the next character in the receive buffer.
- usb_cdc_putc() will put data into a RAM buffer, and when the endpoint buffer is ready to transmit the library will send the data.
- usb_cdc_kbhit(), usb_cdc_getc() and usb_cdc_putc() provide similar functionality to kbhit(), getc() and putc(). Re-write any previous serial I/O to use these functions to migrate from RS232 to USB.
- usb_cdc.h provides other features, such as ports of functions from input.c, a method for getting DTE/DTR from terminal and getting the baud rate as specified in your terminal program. Read the documentation in usb_cdc.h for more information.

Appendix A: Installing HID drivers in Windows 98

Note: These drivers should already be included in versions of Windows after Windows 98. For non-Windows 98 machines the install process should be similar but not exact to these directions. Refer to your operating system documentation. USB does not work in Windows 95.

☐ **Step 1:**
After connecting the USB device to the system, Windows should detect the HID device and the install wizard should start:



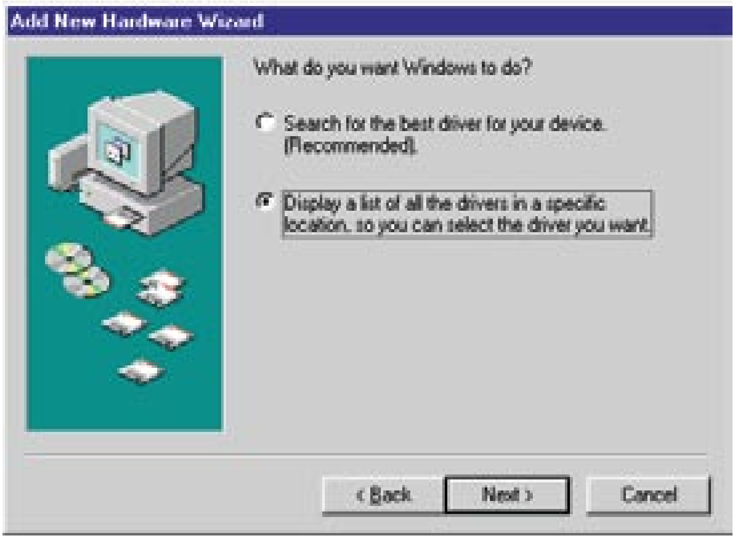
This wizard searches for new drivers for:

USB Human Interface Device

A device driver is a software program that makes a hardware device work.

If the above dialog box appears, Press **Next.**

☐ **Step 2:**
The following dialog box should be displayed:



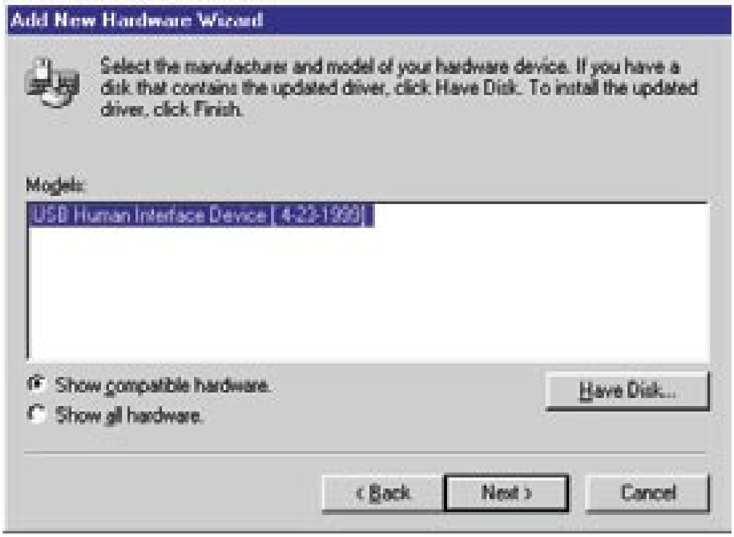
What do you want Windows to do?

○ Search for the best driver for your device. (Recommended).

● Display a list of all the drivers in a specific location, so you can select the driver you want.

Select **Display a list of all the drivers in a specific location**, and press **Next**.

☐ **Step 3:**
The following dialog box appears:



Select the manufacturer and model of your hardware device. If you have a disk that contains the updated driver, click Have Disk. To install the updated driver, click Finish.

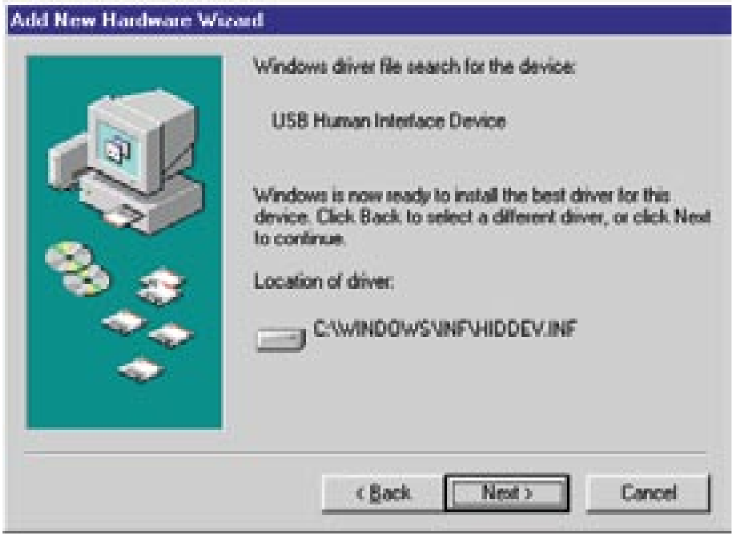
Models:
USB Human Interface Device [4-23-1999]

● Show compatible hardware.
○ Show all hardware.

Make sure **USB Human Interface Device** is highlighted, and press **Next**.

Appendix A: Installing HID drivers in Windows 98 Continued

- ☐ **Step 4:**
  The following dialog box should be displayed:



Add New Hardware Wizard

Windows driver file search for the device:

USB Human Interface Device

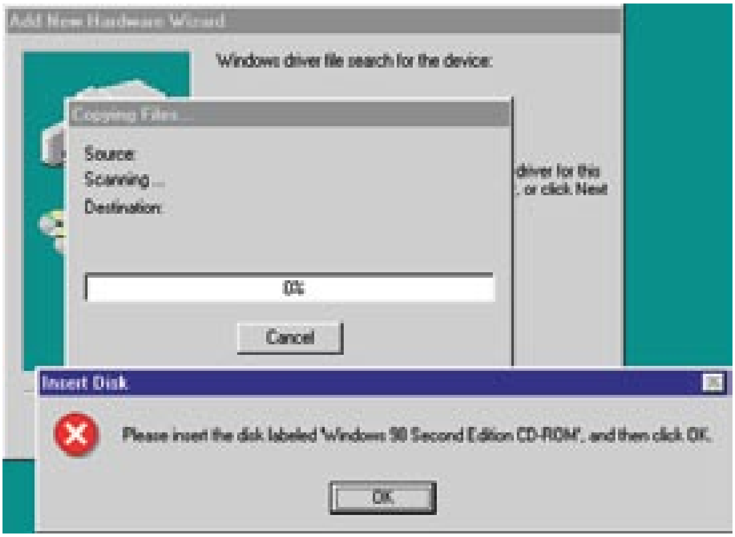
Windows is now ready to install the best driver for this device. Click Back to select a different driver, or click Next to continue.

Location of driver:

C:\WINDOWS\INF\HIDDEV.INF

Press **Next**

- ☐ **Step 5:**
  The following dialog box may appear:



Add New Hardware Wizard

Windows driver file search for the device:

Copying Files...

Source:
Scanning ...
Destination:

0%

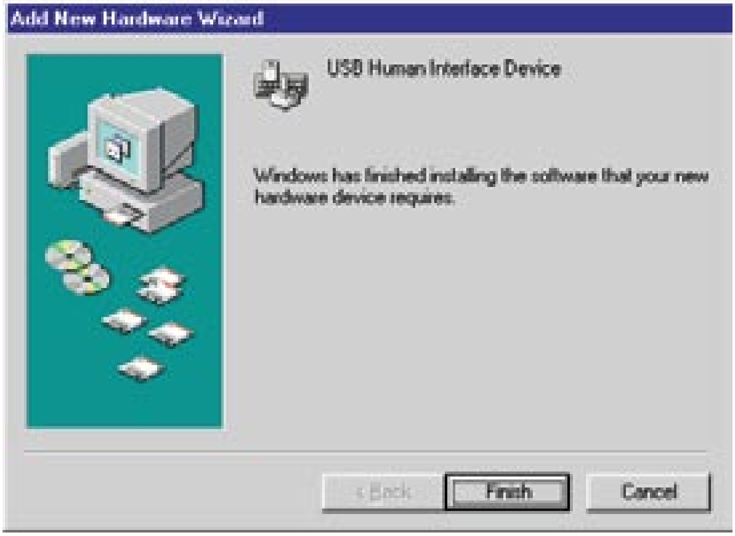
Insert Disk

Please insert the disk labeled 'Windows 98 Second Edition CD-ROM', and then click OK.

If the above dialog box was not present, then go to Step 6.

If the above dialog box occurred , put in the Windows 98 CD into the CD-ROM and press OK. If the PC cannot auto-detect the CD, another dialog box may open stating it cannot find the file. Press the **Browse** button on this dialog box and find the file specified on the CD, and press **OK**.

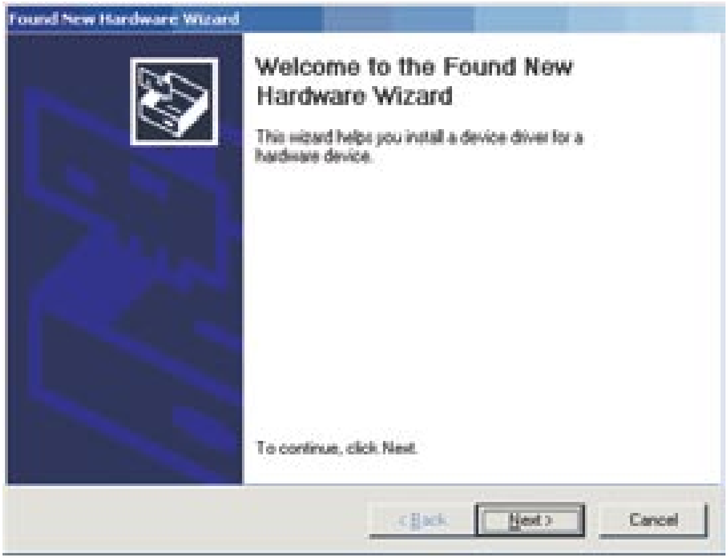
- ☐ **Step 6:**
  If the installation was successful, the following dialog box will appear.



Windows has finished installing the software that your new hardware device requires.

Appendix B: Installing CCS Bulk drivers for Windows 2000/XP

- ☐ **Step 1:**
  After plugging in the USB Prototyping board, the following dialog box will appear:



Welcome to the Found New Hardware Wizard

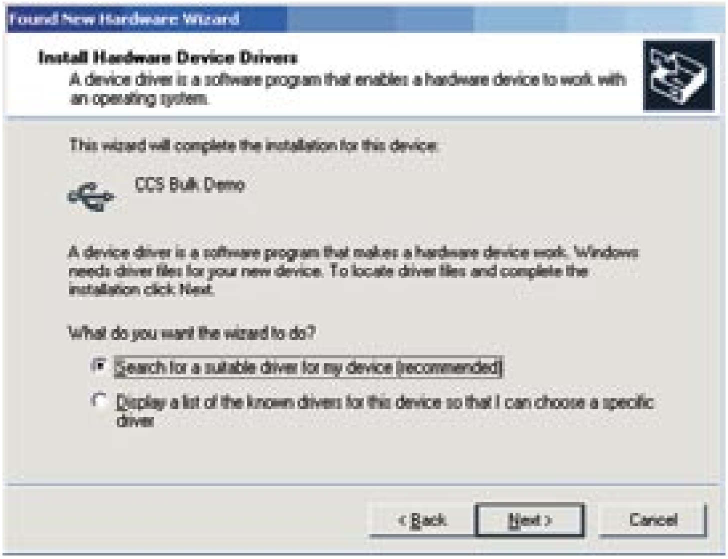
This wizard helps you install a device driver for a hardware device.

To continue, click Next.

If this dialog box does not pop up, go to Chapter 9 of this tutorial to program the USB Prototyping board with a bulk USB device example.

Otherwise press **Next**.

☐ **Step 2:**



Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

CCS Bulk Demo

A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

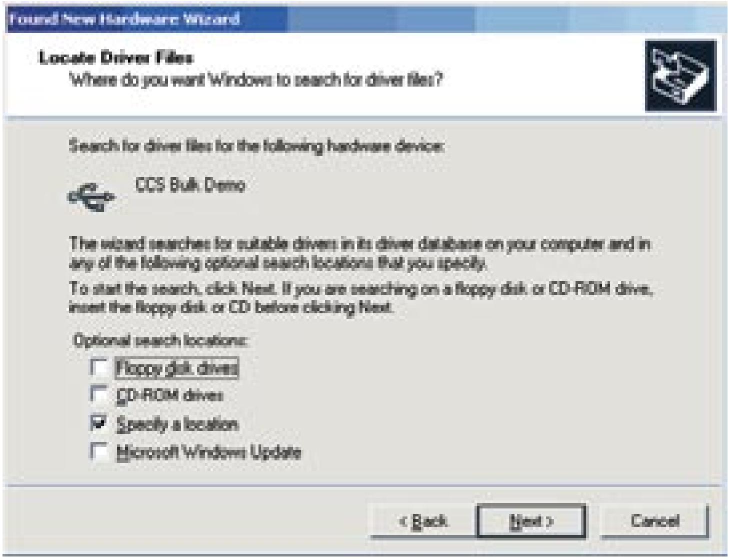
What do you want the wizard to do?

● Search for a suitable driver for my device (recommended)

○ Display a list of the known drivers for this device so that I can choose a specific driver

Select *“Search for suitable driver for my device”* and press **Next**.

☐ **Step 3:**



Locate Driver Files

Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

CCS Bulk Demo

The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

☐ Floppy disk drives

☐ CD-ROM drives

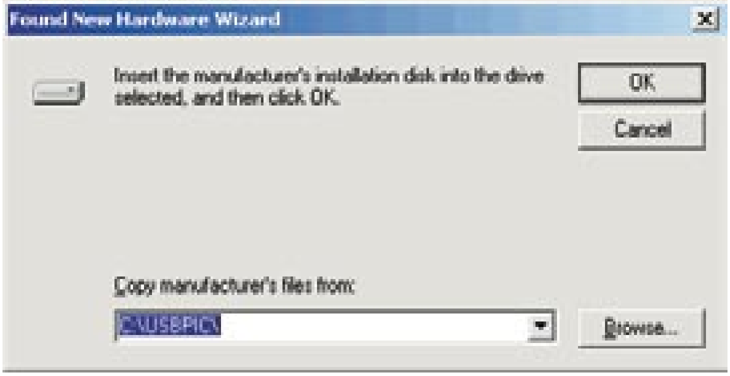
☑ Specify a location

☐ Microsoft Windows Update

Select “Specify a location” and press **Next.**

Appendix B: Installing CCS Bulk drivers for Windows 2000/XP, Continued

☐ **Step 4:**



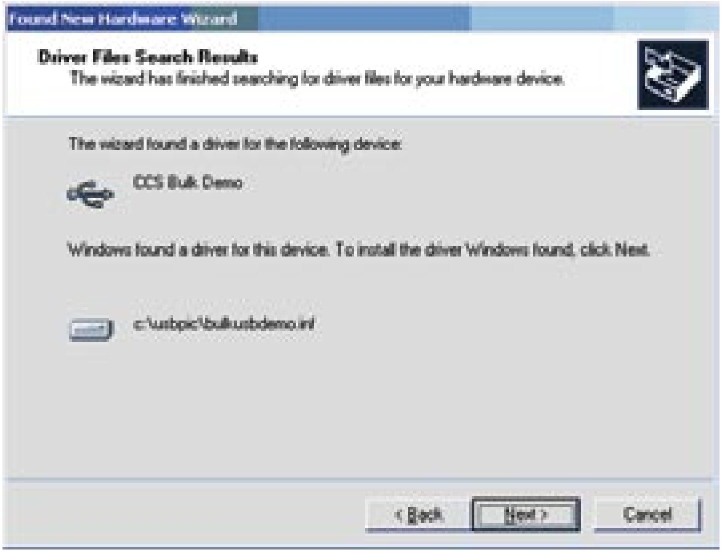
Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from: C:\USBPIC\

Type in *C:\USBPIC* in the text box (where C:\USBPIC is the location where the unzipped example files from the Development Tools CD-ROM are saved. If saved to a different location then put that location here).

After the proper directory is entered into the text field, press the **Next** button.

☐ **Step 5:**



Driver Files Search Results

The wizard has finished searching for driver files for your hardware device.

The wizard found a driver for the following device:

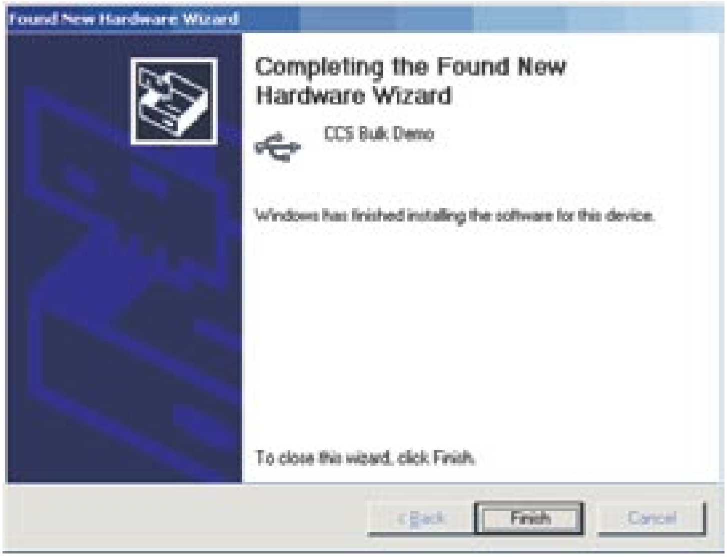
CCS Bulk Demo

Windows found a driver for this device. To install the driver Windows found, click Next.

c:\usbpic\bulkusbdemo.inf

Press **Next**.

☐ **Step 6:**



Completing the Found New Hardware Wizard

CCS Bulk Demo

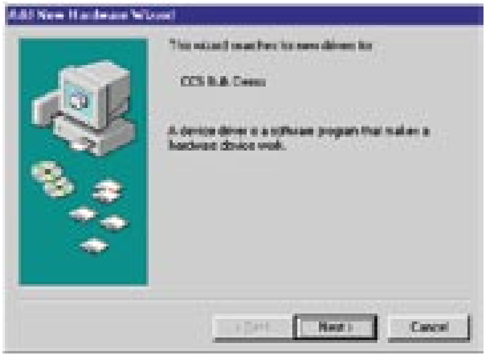
Windows has finished installing the software for this device.

To close this wizard, click Finish.

Press **Finish**.

Appendix C: Installing CCS Bulk drivers for Windows 98/ME

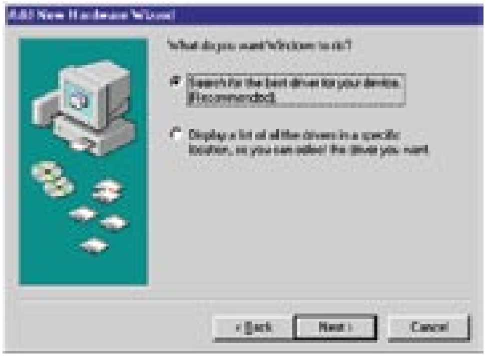
- ☐ **Step 1:**
  Plug-in the USB Prototyping board, the following dialog box will occur:



If this dialog box is not shown, go to Chapter 9 in this tutorial to program the USB Prototyping board with a bulk USB device example.

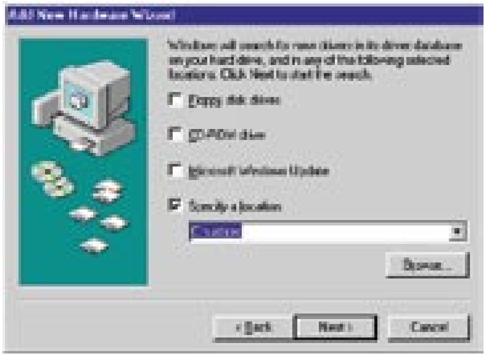
Otherwise, press **Next**.

- ☐ **Step 2:**



Select *"Search for the best driver for your device."* and press **Next**.

- ☐ **Step 3:**

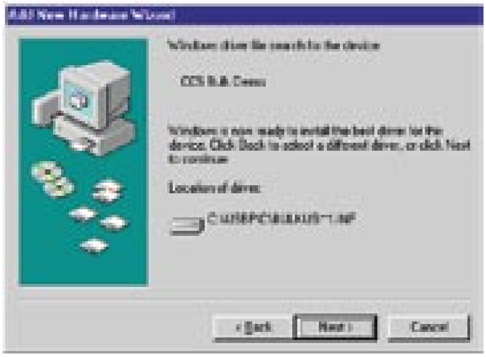


Select "*Specify a location.*"

Type in *C:\USBPIC* in the text box (where C:\USBPIC is the location where the unzipped example files. If saved to a different location then put that location here).

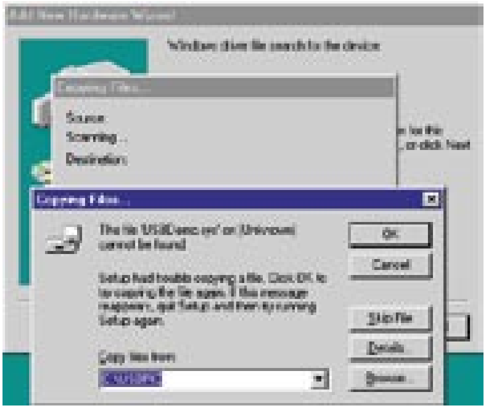
After the proper directory is entered into the text field, press the **Next** button.

☐ **Step 4:**



Press **Next**

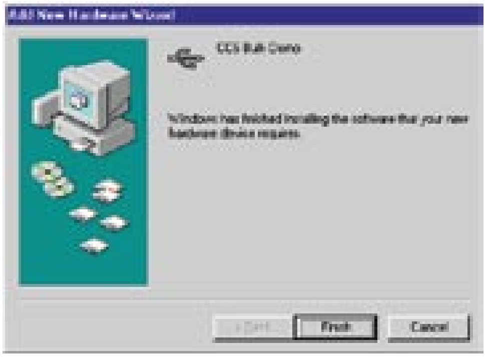
☐ **Step 5:**



Type in *C:\USBPIC* in the text box (where C:\USBPIC is the location where the unzipped example files are saved. If saved to a different location then put that location here).

After the proper directory is entered into the text field, press the **OK** button.

☐ **Step 6:**



Press **Finish**.

Further References

Official Website: http://www.usb.org/ (Official documentation, discussion forum, certification tools and software)
Microchip's USBApp Note Central: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1490&filterID=404 (Microchip has written several application notes about using USB with their processors. Also includes example firmware.)
Jan Axelson's USB Central: http://www.lvr.com/usb.htm (Frequently asked developer questions, software/firmware/driver development examples, links)
Trace Systems HIDMaker: http://www.tracesystemsinc.com/ (Useful software application that automatically generates HID applications, including the PC software and the PIC firmware. Supports CCS compiler)
Thesycon USBIO: http://www.thesycon.de/eng/usbio.shtml (Sells very useful USB debugging/testing software and a development kit with a generic USB driver. Demo downloads are available.)
Ellisys: http://www.ellisys.com/ (Sell low cost hardware USB packet analyzers.)

On The Web

Comprehensive list of PIC® MCU Development tools and information	www.mcuspace.com
Comprehensive list of PICmicro® Development tools and information	www.pic-c.com/links
Microchip Home Page	www.microchip.com
CCS Compiler/Tools Home Page	www.ccsinfo.com
CCS Compiler/Tools Software Update Page	www.ccsinfo.com click: Support → Downloads
C Compiler User Message Exchange	www.ccsinfo.com/forum
Device Datasheets List	www.ccsinfo.com click: Support → Device Datasheets
C Compiler Technical Support	support@ccsinfo.com

Other Development Tools

EMULATORS

The ICD used in this booklet uses two I/O pins on the chip to communicate with a small debug program in the chip. This is a basic debug tool that takes up some of the chip's resources (I/O pins and memory). An emulator replaces the chip with a special connector that connects to a unit that emulates the chip. The debugging works in a simulator manner except that the chip has all of its normal resources, the debugger runs faster and there are more debug features. For example an emulator typically will allow any number of breakpoints. Some of the emulators can break on an external event like some signal on the target board changing. Some emulators can break on an external event like some that were executed before a breakpoint was reached. Emulators cost between $500 and $3000 depending on the chips they cover and the features.

DEVICE PROGRAMMERS

The ICD can be used to program FLASH chips as was done in these exercises. A stand alone device programmer may be used to program all the chips. These programmers will use the .HEX file output from the compiler to do the programming. Many standard EEPROM programmers do know how to program the Microchip parts. There are a large number of Microchip only device programmers in the $100-$200 price range. Note that some chips can be programmed once (OTP) and some parts need to be erased under a UV light before they can be re-programmed (Windowed). CCS offers the Mach X which is a stand-alone programmer and can be used as an in-circuit debugger.

PROTOTYPING BOARDS

There are a large number of Prototyping boards available from a number of sources. Some have an ICD interface and others simply have a socket for a chip that is externally programmed. Some boards have some advanced functionality on the board to help design complex software. For example, CCS has a Prototyping board with a full 56K modem on board and a TCP/IP stack chip ready to run internet applications such as an e-mail sending program or a mini web server. Another Prototyping board from CCS has a USB interface chip, making it easy to start developing USB application programs.

SIMULATORS

A simulator is a program that runs on the PC and pretends to be a microcontroller chip. A simulator offers all the normal debug capability such as single stepping and looking at variables, however there is no interaction with real hardware. This works well if you want to test a math function but not so good if you want to test an interface to another chip. With the availability of low cost tools, such as the ICD in this kit, there is less interest in simulators. Microchip offers a free simulator that can be downloaded from their web site. Some other vendors offer simulators as a part of their development packages.

CCS Programmer Control Software

The CCSLOAD software will work for all the CCS device programmers and replaces the older ICD.EXE and MACHX.EXE software. The CCSLOAD software is stand-alone and does not require any other software on the PC. CCSLOAD supports ICD-Sxx, ICD-Uxx, Mach X, Load-n-Go, and PRIME8.

Powerful Command Line Options in Windows and Linux

- Specify operational settings at the execution level
- Set-up software to perform, tasks like save, set target Vdd
- Preset with operational or control settings for user

Easy to use Production Interface

- Simply point, click and program
- Additions to HEX file organization include associating comments or a graphic image to a file to better ensure proper file selection for programming
- Hands-Free mode auto programs each time a new target is connected to the programmer
- PC audio cues indicate success and fail

Extensive Diagnostics

- Each target pin connection can be individually tested
- Programming and debugging is tested with known good programs
- Various PC driver tests to identify specific driver installation problems

Enhanced Security Options

- Erase chips that failed programming
- Verify protected code cannot be read after programming
- File wide CRC checking

Automatic Serial Numbering Options

- Program memory or Data EEPROM
- Incremented, from a file list or by user prompt
- Binary, ASCII string or UNICODE string

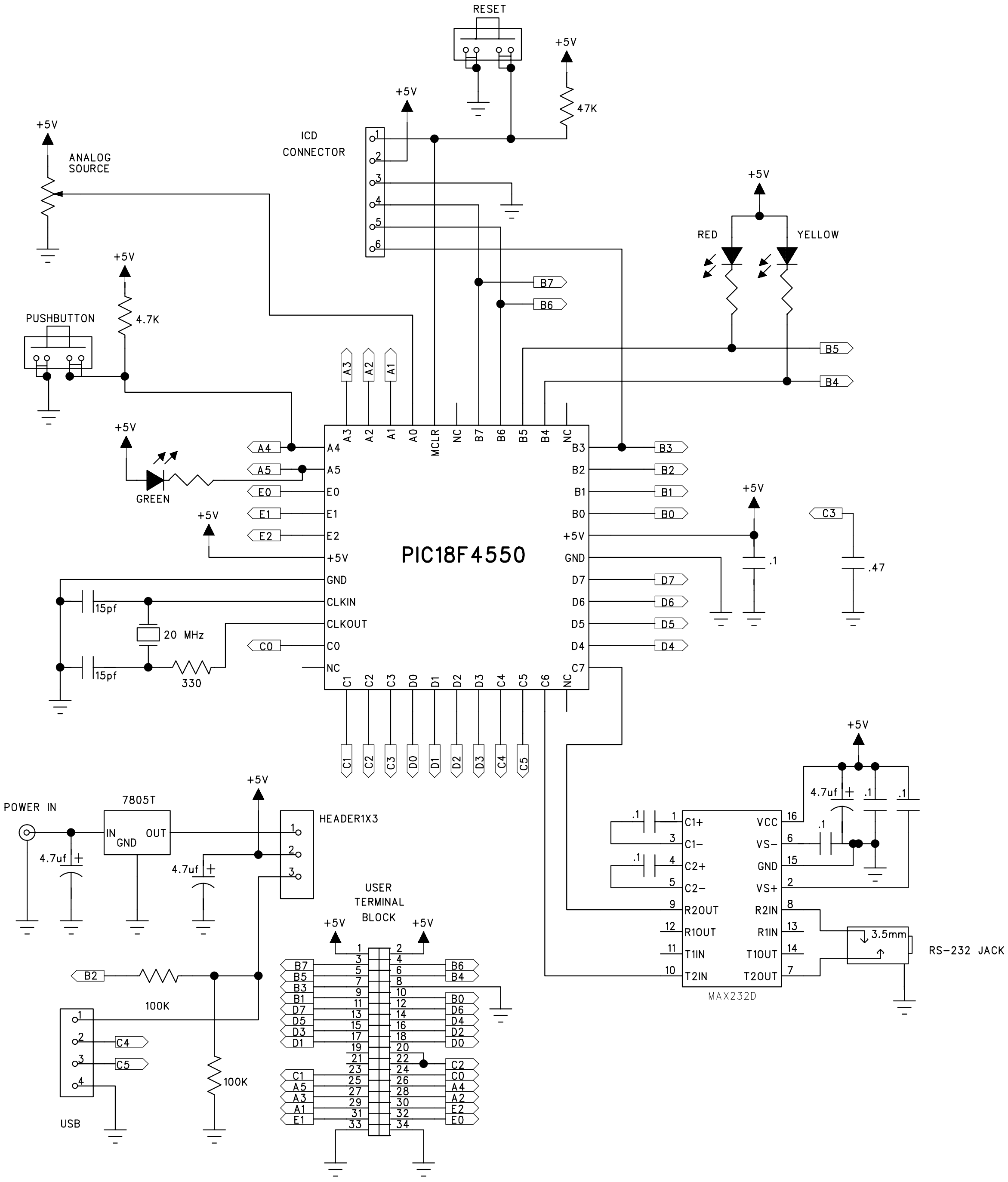
CCS IDE owners can use the CCSLOAD program with:

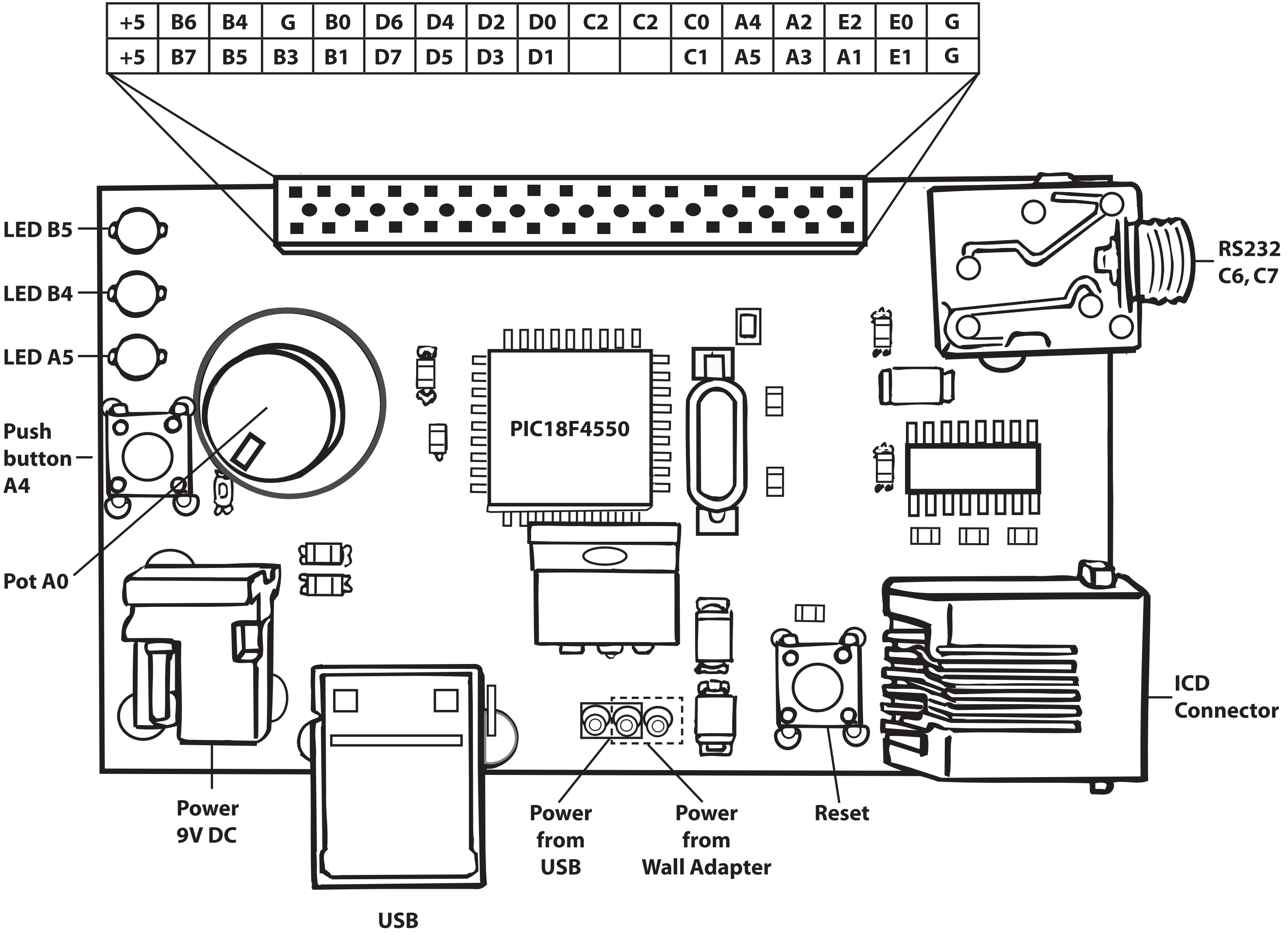
- MPLAB®ICD 2/ICD 3
- MPLAB®REAL ICE™
- **All CCS programmers and debuggers**

How to Get Started:

Step 1: *Connect Programmer to PC and target board. Software will auto-detect the programmer and device.*
Step 2: *Select Hex File for target board.*
Step 3: *Select Test Target. Status bar will show current progress of the operation.*
Step 4: *Click “Write to Chip” to program the device.*

Use the Diagnostics tab for troubleshooting or the ccsload.chm help file for additional assistance.


RESET
+5V
47K
ICD CONNECTOR
1
2
3
4
5
6
ANALOG SOURCE
PUSHBUTTON
4.7K
RED
YELLOW
B7
B6
B5
B4
A3
A2
A1
A0
MCLR
NC
A4
A5
E0
E1
E2
+5V
GND
CLKIN
CLKOUT
C0
NC
GREEN
15pf
20 MHz
330
PIC18F4550
B3
B2
B1
B0
D7
D6
D5
D4
C7
C1
C2
C3
D0
D1
D2
D3
C4
C5
C6
.1
C3
.47
POWER IN
7805T
IN
GND
OUT
4.7uf
HEADER1X3
B2
100K
USB
USER TERMINAL BLOCK
MAX232D
C1+
C1-
C2+
C2-
R2OUT
R1OUT
T1IN
T2IN
VCC
VS-
GND
VS+
R2IN
R1IN
T1OUT
T2OUT
3.5mm
RS-232 JACK



+5	B6	B4	G	B0	D6	D4	D2	D0	C2	C2	C0	A4	A2	E2	E0	G
+5	B7	B5	B3	B1	D7	D5	D3	D1			C1	A5	A3	A1	E1	G