Quarter 1
2023

CCS Inc

www.ccsinfo.com
262-522-6500

<Bits & Bytes> Newsletter

INSIDE THIS ISSUE:

Product Spotlight

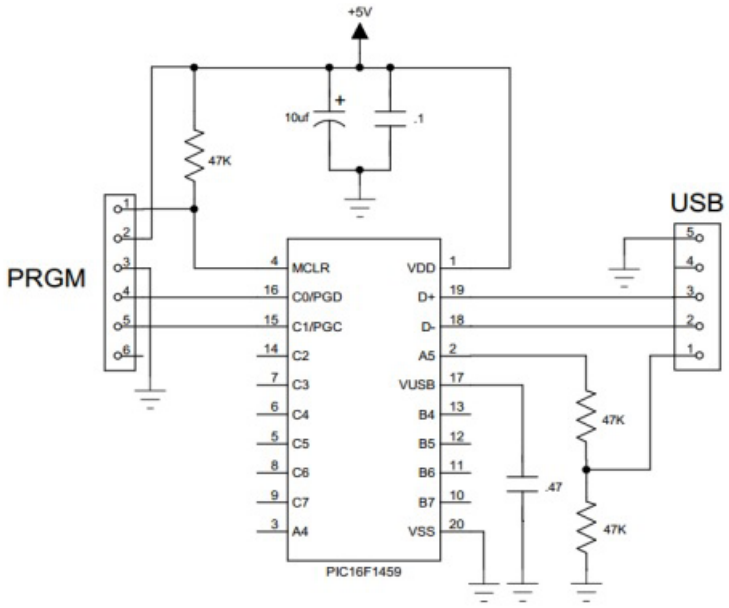
ICD-U80
IN-CIRCUIT PROGRAMMER & DEBUGGER

ICD-U80 is a complete In-Circuit Programming and In-Circuit Debugging solution for Microchip's PIC® MCUs and dsPIC® DSCs. ICD-U80 debug support covers all targets that have debug mode when used in conjunction with the C-Aware IDE Compilers. The unit also provides in-circuit serial programming (ICSP™) support for all Flash chips.

support@ccsinfo.com sales@ccsinfo.com

The CCS C Compiler from the beginning has made it easy to communicate over an RS232 like port. Many of our users have used this extensively, not only for communicating with serial devices but also for diagnostics and debugging. Now, PC's and many other devices have replaced their RS232 ports with USB ports. The USB hardware and software is much more complex than RS232. This has discouraged many from migrating over or they have used a crutch like an external chip to do the USB magic (like FTDI). This article shows how to easily add a USB port to your PICR MCU application.

HARDWARE
Microchip has a number of chips with built in USB. For example, the PIC16F1459 ($1.38/100) or PIC18F14K50 ($1.79/100). Here is an example schematic:
(note: pin numbers apply to PDIP, SOIC, and SSOP packages)



Shown here is a USBmicro style connector. The more common type B connector is the same pinout except there is no pin 5 and 4 is the ground. The D+ and D- pins are for the data and sometimes there will be a 27 ohm series resistor and/or zener protection diodes on those pins. When using a peripheral device, pin 1 will supply 5 volts (up to a half amp). The board can be powered from that 5 volts; however, in this schematic it is to simply detect if the USB cable is plugged in. Although users can do that from the data lines, it is easier to detect the 5 volts like this. Sometimes users will put series coils on the 5 volts and ground to reduce noise.

The Vusb on this chip simply needs a cap for an internal voltage regulator. Some chips have a dedicated Vbus pin for the 5 volts detect. Careful with the pin names, as they are not consistent between chips.

HOST SOFTWARE
The USB bus has a number of protocols that can be used, each with many options and configurations. HID (human Input device) is used for keyboards and mice and is easy to use on any OS because the drivers are built in. There is a data limit however, (like 8 bytes per ms) that makes it impractical for many applications. CDC is a protocol designed to emulate an RS232 port. The Windows drivers will create a virtual COM port for a CDC USB device so the PC application can use COM1... just like an RS232 port.

When a Windows 10 sees a CDC device, it automatically installs a driver for it. For older versions of Windows, users need a short .inf file to describe their device and include the device VID (vendor ID) and PID (product ID). Examples .inf files are in the CCS C Compiler examples directory. Every USB device is supposed to have a unique VID/PID and serial number. If users have two of the same devices plugged in, the VID/PID will match but they are differentiated by serial number. To get a VID users need to register with the USB standards organization.

USB is point to point and one point is a host and the other a device. Hub's can also be involved but that is beyond our concern for this article. The host initiates all activity. For example, a PC is a host and it will poll every device about once a millisecond and transfer any data needing transferring. When the device is first plugged into a port, there is a handshaking that takes place that involves the device identifying itself along with the protocol it will use and various parameters. For example, how much current it expects to draw off the bus. This handshake is called enumeration. In Windows users know it happened by the beep.

Some PIC24 parts have USB hardware that can also be a host. For example, this might be used to read a USB flash drive.

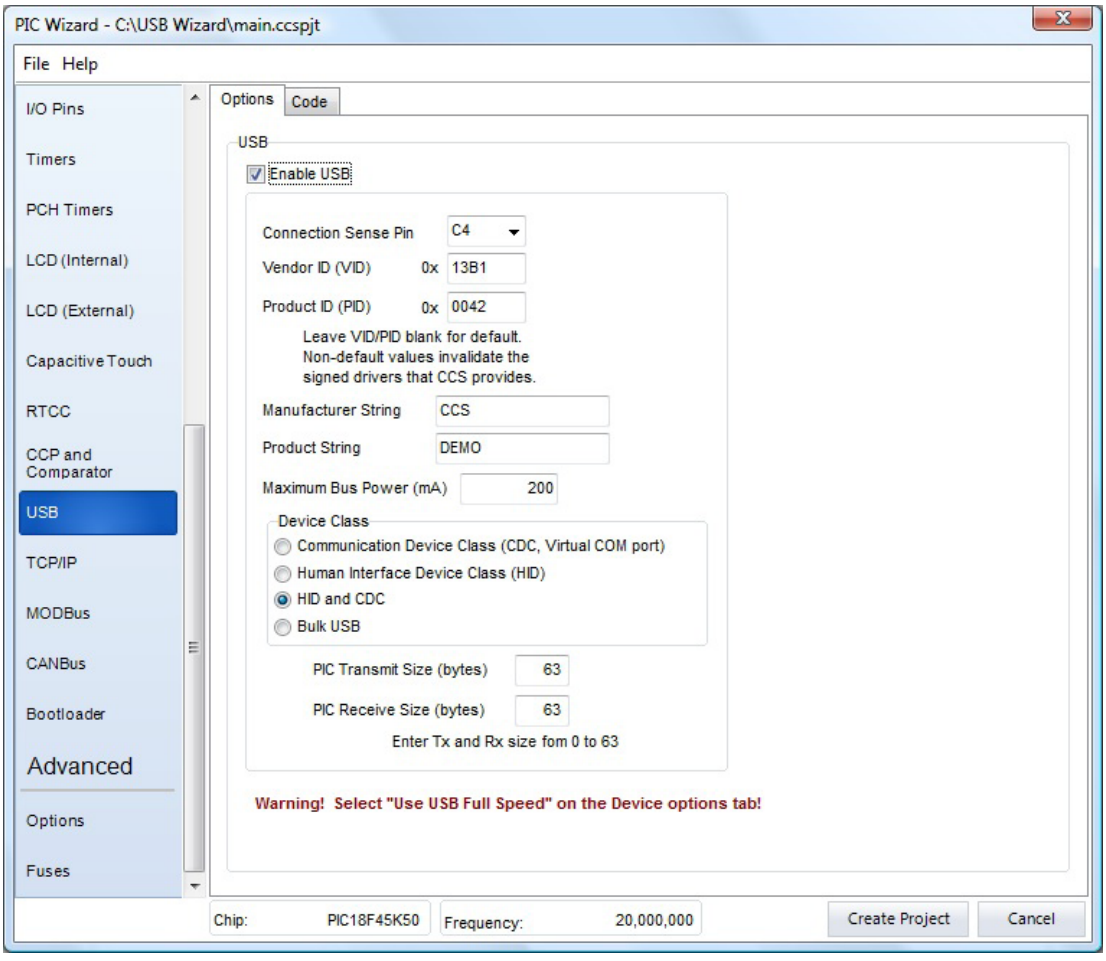
```
#include <16F1459.h>
#use delay(internal=48MHz,USB_FULL,ACT=USB)

#define USB_CON_SENSE_PIN  PIN_A5   // Connected to USB +5V

#define USB_STRINGS_OVERWRITTEN

#define USB_CONFIG_VID 0x2405  // CCS VID
#define USB_CONFIG_PID 0x8001

#define USB_DESC_STRING_TYPE 3
#define USB_STRING(x)  (sizeof(_UNICODE(x))+2),  \
                       USB_DESC_STRING_TYPE,_UNICODE(x)
#define USB_ENGLISH_STRING 4,USB_DESC_STRING_TYPE,0x09,0x04
               //Microsoft Defined for US-English

char const USB_STRING_DESC[]={
     USB_ENGLISH_STRING,          //string 0 - language
     USB_STRING("CCS"),           //string 1 - manufacturer
     USB_STRING("My-PIC-Device") //string 2 - product
};

#include <usb_cdc.h>
```

USB Wizard
The wizard allows the use of one of three communication protocols:

Communication Device Class, or CDC, creates a virtual COM port on a PC. On Windows PCs, this is the legacy COM ports (such as COM1) which can be opened with legacy terminal software. The CCS CDC library allows the use of printf(), putc() and getc() to communicate to the host PC using the virtual COM port.

Human Interface Devices, or HID, is a simple protocol for things like Mice and Keyboards. This wizard allows the user to create a HID project using a vendor specific HID descriptor report. CCS also provides a PC program written in VB.NET to communicate with the PIC in this manner. CCS also provides HID keyboard and HID mouse examples for the PIC.

Bulk transfers is a method of reading/writing directly to the buffer endpoint. Using this method does require drivers for your operating system, but since XP Microsoft has been shipping WinUSB which is compatible with this mode. CCS does provide a VB.NET demo application to show how to use this driver to communicate with the PIC.



CCS IDE Project Wizard includes USB. The IDE requires a minimum operating system of Windows 95(or later) or Linux.

These options lets the user configure the Pulse Width Modulation (PWM) pins. They are only available on devices equipped with PWM. The options for these functions vary depending on the chip and are listed in the device header file.

Relevant Functions:

- **setup_power_pwm(config)** - Sets up the PWM clock, period, dead time etc.
- **setup_power_pwm_pins(module x)** - Configure the pins of the PWM to be in Complementary, ON or OFF mod.
- **set_power_pwmx_duty(duty)** - Stores the value of the duty cycle in the PDCXL/H register. This duty cycle value is the time for which the PWM is in active state.
- **set_power_pwm_override(pwm,override,value)** - This function determines whether the OVDCONS or the PDC registers determine the PWM output .

Relevant Preprocessor:
None

Relevant Interrupts:

- #INT_PWMTB - PWM Timebase Interrupt (Only available on PIC18XX31)

Relevant Include Files:
None, all functions are built-in

Relevant getenv() Parameters:
None

Example Code:

```
....
long duty_cycle, period;
...
                                // Configures PWM  pins to be ON,OFF or in
                                // Complimentary mode.
setup_power_pwm_pins(PWM_COMPLEMENTARY ,PWM_OFF, PWM_OFF, PWM_OFF_;
                                // Sets up PWM clock , postscale and period.  Here
                                // Here period is used to set the PWM Frequency as follows
                                // Frequency=Fosc/(4* (period+1)*postcae)
                                // *postscale)

setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN,1,0,period,0,1,0);

set_power_pwm0_duty(duty_cycle));     // Sets the duty cycle of the PWM 0,1 in
                                      // Complementary mode
```

Program and Debug with Mulitiple ICD Units

CCS has implemented the ability to use multiple ICD units on the same PC. This allows you program or debug different target devices in separate projects without needing to close out of any files.

How to Do It with PCW:
Need two ICD units and target devices.

To run two PCWs → use the command-line option force_new

c:ProgramFilesPICCpcw → will open the first PCW

c:ProgramFilesPICCpcw+force_new → will open the second PCW

The USB port used by each target device will be visible in the Debug Configure tab within the Advanced Debugger under “port”. You can select which ICD the Debugger should talk to using this setting.

How to Use with ICD Control Panels:
Need two ICD units and target devices

Open two ICD control panels by running the executable from your desktop.

At the start-up, the USB port will automatically be assigned for each ICD unit. You can then go the “Configure Port” button the control panel to switch ports.

Why Does the .LST File Look Out of Order?

The list file is produced to show the assembly code created for the C source code. Each C source line has the corresponding assembly lines under it to show the compiler's work. The following three special cases make the .LST file look strange to the first time viewer. Understanding how the compiler is working in these special cases will make the .LST file appear quite normal and very useful.

1. Stray code near the top of the program is sometimes under what looks like a non-executable source line.

Some of the code generated by the compiler does not correspond to any particular source line. The compiler will put this code either near the top of the program or sometimes under a #USE that caused subroutines to be generated.

2. The addresses are out of order.

The compiler will create the .LST file in the order of the C source code. The linker has re-arranged the code to properly fit the functions into the best code pages and the best half of a code page. The resulting code is not in source order. Whenever the compiler has a discontinuity in the .LST file, it will put a * line in the file. This is most often seen between functions and in places where INLINE functions are called. In the case of an INLINE function, the addresses will continue in order up where the source for the INLINE function is located.

3. The compiler has gone insane and generated the same instruction over and over.

For example:

```
.....A=0;
03F:     CLRF  15
*
46:CLRF  15
*
051:     CLRF  15
*
113:     CLRF  15
```

This effect is seen when the function is an INLINE function and is called from more than one place. In the above case, the A=0 line is in an INLINE function called in four places. Each place it is called from gets a new copy of the code. Each instance of the code is shown along with the original source line, and the result may look unusual until the addresses and the * are noticed.