Basic USB - Using Microchip Stack and C#.Net

Contributed by Mat

Monday, 27 February 2006
Last Updated Tuesday, 14 March 2006

This article is an introduction to designing a USB PIC based device. It is based on the original C# API I developed, and heavily influenced by J1M's article. The article will demonstrate how to use a PIC18F4550 to produce a USB 2.0 Device which will perform basic input/output operations, whilst communicating with a C#.Net application via the microchip drivers.

Please note all code is released AS-IS and should used at the users own risk, no guarantee is made by PICcoder.co.uk or the author with regards to the accuracy of the diagrams or code contained in this article.

{mos_sb_discuss:22}

Example Application

In this article I will be discussing a basic C# application which communicates with a PIC via usb, allowing control of two LED's and can send two bytes to the PIC, which returns the summation of the two. The example application may be seen on the right.

The project consists of four main components-
- The Hardware
- PIC Software
- PC Software
- Windows Driver

Without one of these components working correctly, the project will fail entirely and be very difficult to debug, this makes initial development of such a system particularly taxing, and I would strongly recommend taking this work as a basis for developing any further systems.

All the code discussed code and dircuit diagram can be found in the zip file below.

PIC USB demo (378.78 KB)

Overview

It is important to consider how the overall system is going to work, and how the different components interact with each other to complete the overall task which in this case is a rather trivial one. At the basic level we are going to use the on board USB interface on the PIC to communicate with PC, all communications on the PIC are handled by the microchip usb stack. The USB stack is responsible for enumerating the device on the PC, this is essentially a fancy way of letting the PC know that we are connected and to which port it is connect to. The stack is also responsible for sending the appropiate descriptors to the PC, essentially telling the PC what sort of device it is. It is possible to change both of these however I shall leave these for another article! Once the pic is connected to the PC, we shall be using the standard microchip drivers to communicate with the device. This forms a data channel from Windows to a segment of code in the PIC. The final link in the chain is the code at both ends of the channel, which can be customised to produce the functionallity required.

So now you know roughly what we are going to try and do, lets get cracking!! {mospagebreak title=Hardware&heading=Hardware} Hardware

The basic hardware design can be seen below. If you can get hold of a PICDEM FS USB demo board, then thats perfect if not a bare bones version of it can be produced using the circuit diagram below.

USB Hardware

Please note the Schematic is for a TQFP pin out and will need adjusting if another package is used. Also please connect a 4.7K resistor between RA0 and VDD, to avoid bootup issues.

As you can see there is not much to the hardware side of things, most of the complicated electronics is already in the main body of the PIC, including the pull up resistors which were necessary on the previously microchip USB devices to signal the USB standard being used. The only slight complication with the hardware is that I was using a USB On The Go (OTG) connector, and if you dont want to use one of these you'll find a normal USB connector only has 4 pins avaiable, in which case connect the ground to pin 4 of the usb, instead of pin 5 as shown above. If you have problems locating the USB connectors, try Molex for free samples.

Right Lets get onto the PIC software. {mospagebreak title=PIC Software&heading=PIC Software} PIC Software

The example PIC code discussed in this article can be found below, or in the Complete Code file at the start of the article.

PIC USB demo Code (79.26 KB)

Extract the zip file into C:, this should create a folder called "c:\picusbdemo\" containing all the project files, if you choose to extract it to a different directory be careful setting up the include directories in the project configuration settings. Compiling the USB firmware

To compile the PIC source code you will require MPLAB and Microchip's C18 Compiler. Free versions of both are avaiable from the links provided (student version for C18). The code has been sucessfully compiled on MPLAB v7.31 and C18 V3.00, however it should compile on more recent version if these are available.

To open the project open MCHPUSB.mcw in MPLAB. Before compiling the code, please ensure that the correct device is selected. This is done by clicking 'Configure','Select Device...' then selecting PIC18F4550 as the device.

It is probably a good idea to check the include directories even if you have extracted the files to the correct location. To do this click 'Project','Build options','Project'. Ensure that the 'Output Directory','Intermediates Directory','Include Path' and 'Linker-Script Path' all point to the location of the extracted files. The 'Library Path' should point to the location of the C18 library files, these by default are located in "C:\mcc18\lib", and will need adjusting to your setup.

You should now be ready to compile the source code, press ctrl+F10 to start the build. All being well the code should have compiled sucessfully, however if it is unsucessful please go back and check all the project and include paths are setup correctly. (Any other strange errors, please let me know!)

You should now be able to test the code, to do this program the hardware using your favorate PIC programmer, and connect it to your test PC (warning check the hardware first, there is a risk of causing damage to the PC), all being well the PC should request for the drivers for a "PIC18F4550 Family Device", you can now either skip to the Driver section or keep reading if you want to understand how to add in extra functionality to the device. Modifying the USB Firmware

The two main sections of the USB firmware to be concerned about when modifying the code are the initialisation routines and the main switch statement, which responds to usb commands sent from the C# application.

If you require and further initialisation routines these should be inserted into the UserInit function in user.c, currently only mInitAllLEDs is called which defines the output pins for the led's.

The other main section to modify is the service request switch statement, this is located in the ServiceRequests function in user.c, however before modifying this a new enum value should be defined for CMD, located in user.h. To do this simple add a new command name and associate it to an available id (0-256), in the CMD enum.

After adding the new command to user.h, you need to modify user.c to respond to the new command. In order for the pic to respond to the new command it must be added to the switch statement found in ServiceRequests. When a new command is added to the switch statement it is important to respond to it in the correct manner. There are two possible ways to communicate with the application, either responding with data back or simply just an achknowledge response. The later is easiest to implement so we shall deal with this first.

Essentially the PIC accepts a usb data packet stored in the variable datapacket, which is a byte array (64 bytes long), byte 0 represents the command sent by the application (e.g 0x00 - READ_VERSION) and byte 1 the transmission length (if a none fixed length is used), the remaining bytes can be arbitarily selected depending on the format of the request, and used accordingly. This gives a data format as follows. <Command><length><data><data><data><data>...... When you have finished dealing with the request, counter should be set to 0x01 to indicate that only the command will be sent back to the application as an acknowledgement. However this on its own is not a very useful system, as the pic has no method for transmitting data back to the application and therefore a format for sending data back must be defined. If the command you are programming requires data to be sent back to the pc, then the following format must be constructed within datapacket before exiting the switch statement. Byte 0 should be left alone and represent the command, as per the request, byte 1 should represent the length of the packet to be transmitted back, and the following

bytes may be used for data. This should give the same format as is used for sending bytes to the PIC by the application. You should then set counter to the packet total packet length so the pic knows how much of the buffer to send.
Now you should be able to add your own commands and extra functionality to the PIC, give it a go! If you have any issues leave a message in the forums.   It is possible to change parameters such as the amount of current which may be drawn from the USB port and modifying the descriptors of the device so that the device name and description found in Windows represent your device and not the generic Microchip setup, however I shall leave this for another more advanced article! {mospagebreak title=Windows Drivers&heading=Windows Drivers} Windows Drivers

Now you've built the hardware, compiled the source, you are ready to go. All being well when you connect the PIC to the PC, windows should request the drivers, if not go back and check the hardware and that the PIC programmed correctly.

The drivers we shall be using are the standard Microchip ones, unmodified. These may be changed to alter the icon, change the device class etc... however this isnt the place to explain all that, all being well a new article might come out containing this information.

The drivers can be found below, or in the Complete code at the start of the article.

Microchip USB Driver (90.94 KB)

Hopefully you shouldn't have any issues installing these (as long as windows ask's for them, you should be in the clear), so I'll assume you are all technically competent enough to do this (as you are designing USB stuff after all!!).

Once the drivers are installed correctly, two of the LED's connected to the pic should be flashing alternatively to indicate the connection has been established. {mospagebreak title=Application Code&heading=Application Code} Application Code

The application code can be found below or in the Complete Code at the top of the article.

USB Demo PC Code (52.44 KB)

The application was developed using Visual Studio 2003 C#.Net, however I believe it should also compile correctly under C# Express 2005 , let me know if you have any success with this. In order to run the code (or compile it for that matter) you will also require the .NET Framework to be installed. The code should be pretty self explanitary however I will go over the general concepts.Example Application

The idea behind the program is to create a simple way of controlling two LED's attached to the PIC, and to create a data channel, which is tested by sending 2 bytes to the pic which it sums and returns to the application.

The application consists of three main parts, the mpusbapi.dll, usb_interface class and the main form. The DLL (located in the debug directory) acts an interface between the application and windows driver, therefore do not modify this, and just ensure that it is located in the directory of the program. The form is used to interface to the user and trigger the usb_interface to send commands to the PIC when required.

The main work of the program is located in the usb_interface class, this constructs the required packet to send to the PIC and deals with the response if necessary. The class is setup to only function with a single PIC based usb device connected, this is to reduce the complexity as most systems will not require multiple devices (I will possibly do an advanced article on this later).

You probably won't want to change any of the private functions as these are the fundemental interface to the DLL, and should only be changed if you really know what your doing! As for the public functions these are fair game, as can be modified to provide any functionality required.

Everytime a new command is added to the pic, a coresponding function should be added to the usb_interface class, if you base the new commands on the UpdateLED and Add functions then these shouldn't be far wrong. The key points to remember are that the RecvLength should match the packet length sent by the PIC otherwise any response sent, will be disregarded and also to ensure that the second field in SendReceivePacket represents the length of the send buffer to transmit.

The packets should be constructed in send_buf first, following the same format as before, and then the

SendReceievePacket function called, as in the examples. This will return any required data to receive_buf and can be used as necessary in your application.
<command><length - if required><data><data><data>....

(If the application returns an incorrect value for the Add function, this is most likely due to the total lack of overflow code present in the pic, this is to maintain simplicity!) Conclusion

So there you have it, you should now be a PIC usb expert! You should all being well be able to compile the code, and start adding your own functions to the PIC, if you have any question or comments about the article please leave them below or in the forums. Hope thats been helpful!

Mat Clayton

{mos_sb_discuss:22}