

Getting started with PIC Microcontrollers (incl. Example Programs)

Contributed by Evan
Thursday, 24 August 2006
Last Updated Thursday, 30 August 2007

PIC Microcontroller Guide

This is designed to be a collection of knowledge about PIC microcontrollers, both general, and pertaining to my chosen methods. It may not be much of a tutorial for a while, at least until I get some little mini tutorial articles written up.

Compiler

The first thing you have to deal with is of course writing some code. The 'standard' compiler for PICs is MPASM, an assembly compiler which is available from Microchip themselves; they make an entire IDE known as MPLAB freely available.

However, while writing in assembly language has its advantages, for quick code development I find it far easier to use a C compiler. The compiler I use is called BoostC, from <http://www.sourceboost.com>. They also provide an IDE, called SourceBoost. Sourceboost is a great program; not only does it encompass the compiler and provide a good code editor, it also includes debugging which is a HUGE plus, and something that is usually reserved to assembly language programming. There's even a handful of virtual devices you can use when debugging, such as LED bars. SourceBoost is being constantly improved, and the guys at the support forum on their website are quite helpful as well.

However, I don't think it's a good idea to simply dive right into C on a PIC without ever touching a word of assembly. I have seen many examples of people struggling with PIC programming because they were starting off with only C or BASIC. The fact is, a microcontroller is not the same as a computer. In a computer, you are abstracted well away from the actual hardware in most cases; however, in a PIC you are always dealing somewhat directly with the hardware. And since you are so close to the hardware, you really need to understand it before you can program effectively. And, unfortunately to some of you, the best way to become familiar with the hardware is assembly language, because assembly language is pretty much as close as you can get to the hardware, and programming in assembly does require a decent understanding of the way things are working. My programming experience began with a course in assembly programming on a Motorola HC11. While the assembly was of course not quite the same as it would be on a PIC, the general concepts of microcontroller architecture, and using it with assembly, were there and so when moving to the PIC, it was only a matter of skimming some reference info in the datasheet to know how to interface with things.

For a newcomer, I would highly suggest finding a good book on PIC assembly language programming, because they naturally also cover the important hardware aspects. A book I recommend is "Programming and Customizing the PIC Microcontroller" by Myke Predko. If you don't want to buy a book, at least look around on the web a bit, it will save you lots of headaches down the road.

The important point is that C compilers for PIC programming should be used as a TOOL to help you write code more easily, and to make it more straightforward to understand; it should NOT be used simply as a method to abstract things and "escape" from dealing with things on a hardware level

Programmer

The programmer I use runs on USB, and has a 40-pin ZIF socket, as well as an in-circuit serial programming (ICSP) connector. It originally required a 16VDC power adapter, which is tricky to find, so I made a small capacitor charge-pump voltage booster that allows the board to run directly from the 5v USB port, but still generate the necessary 13v programming voltage.

The programmer is known only as "kit 150", you can find info about it (and similar ones) at kitsrus.com. The software they provide is adequate; it does the job it is supposed to do, has some useful features, and is stable (unless you unplug the programmer without first exiting the program).

This programmer does not currently support many of the newer 18F series PICs, such as the USB-capable ones like the 18F4550. For those, I use a standard, simple serial JDM-style programmer purchased from sparkfun.com.

Example Programs

Note: these example programs are for the PIC 16F88

helloworld.c

This program is your standard "hello world" type. It simply toggles all the output-capable port pins once per second, so if

you hook up some LEDs you can see them flash. It's just meant as a simple beginning program, to demonstrate the general format of a program, the configuration you need to do, and is an easy way to make sure everything's working.

I have not yet personally tested this program on hardware; although it should work, I make no promise yet

adctest.c

This program is basically a step up from a "hello world" application. What it does, is read values from the analog-to-digital converter (ADC) on RA0 (port A pin 0) and output them via the serial port. It also echos any characters it receives via serial, back to the computer via serial. While this program was really the first step of a project I was working on, it contains a lot of the important and useful elements of a program in BoostC:

- *Defining bit variables to bits in register locations
- *Using subroutines
- *Setting configuration fuses via #pragma DATA
- *Using an interrupt
- *Configuring the ports via TRIS and PORT registers
- *Configuring ADC module
- *Configuring USART module and associated interrupt

RGBLEDCON.c

This isn't really designed as an example program, but it's not too complicated so it can still be useful. Basically it controls 2 RGB LED's (common anode) in various modes, including either solid color output selectable with a button, or output corresponding to serial input, so the color can be controlled externally. It utilizes software-generated PWM to control the LED brightnesses.

Other Files

4BitLCDInterface.h

This is the file I use whenever I am using an HD44780-compatible character LCD. It is for 2-line displays, and interfaces them in 4-bit mode, so 6 I/O are needed. {mos_sb_discuss:8}