Programming the 8-pin Microchip PIC in Basic Volume 1

Using the PICBASIC PRO Compiler and PIC12F683

Table of Contents

Introduction

I've been programming Microchip PIC microcontrollers for years. When I started programming the most common software option was assembly language programming which can be very confusing to the beginner. Then one day I discovered a product that advertised programming Microchip PIC® microcontrollers in BASIC. It was the PICBASIC compiler from microEngineering Labs. They eventually released a more powerful version called PICBASIC PRO and I have been using that compiler ever since. Along the way they developed a sample version of the PICBASIC PRO compiler that is limited to a few parts and only 31 commands but it was more than enough for someone to try out this great method of programming for free.

This book is a beginner's guide to using the PICBASIC PRO and uses the sample version for all the projects. I will use a low cost programming kit from Microchip as the hardware so the reader can learn right along with me for little investment. This language is so powerful, the 31 commands don't seem like a limitation until you get comfortable with this method of programming and want to do more complex projects. This book is not intended to be a complete source of knowledge on programming with PICBASIC PRO but by the end of the chapters and projects you should be able to branch out and develop your own more complex projects.

It may lead to a profession in programming and that may force you to learn other languages such as the C Language I use in my book "Beginner's Guide to Embedded C Programming". Programming in PICBASIC PRO is very powerful though and can handle just about any task you need. Programming in PICBASIC PRO is also so easy, users from all skill levels can make it work well. After all BASIC stands for Beginner All-purpose Symbolic Instruction Code and the key word is Beginner. BASIC is also great way to introduce microcontroller programming to many skill levels. You cannot read an electronics hobbyist magazine without seeing a project that is microcontroller based. The evolution of the homemade robot has advanced due to the advancements and affordability of microcontrollers. But for some it's still difficult to get started or to find all the right pieces you need to build your own microcontroller development lab. This book is intended to help you get started.

If you have any questions regarding this book or the projects in this book, you can usually get me via email at chuck@elproducts.com. My website at www.elproducts.com shows all the books I’ve written to help readers get started programming. Now lets get started learning how to program Microchip PICs in BASIC.

Chapter 1 – Microcontroller Fundamentals

Many of the programming books I’ve read tend to skip over a couple of important topics; how a microcontroller works and how to use them. Many of the books go into great detail on the software or the project operation but assume you know all the steps to get that program into the microcontroller or exactly what a microcontroller is. I don’t assume the reader knows any of this so in this first chapter I cover the fundamentals of Microcontrollers.

What is a Microcontroller?

Everybody reading this has probably used a Personal Computer (PC) run by a microprocessor which is often times an Intel microprocessor from the Intel Corporation. The PC’s central microprocessor has several support items that allow it to function: 1) The memory; where programs are stored, known as a hard drive or ROM, 2) The RAM, or temporary memory used by the programs running in the microprocessor and 3) The interface to the outside world, also known as the BIOS or input and output (I/O) control. The PC’s system mother board will connect these three components and also have a power supply and a system clock typically running at gigahertz speed and advertised as such to imply how fast the PC is (i.e. 1 Ghz processor).

The PC sends information through the I/O to be displayed on the monitor or printed paper via a connected printer. The I/O also reads the keyboard and mouse position. Basically everything the PC does with a useful purpose to human's runs through the I/O. All these components I’ve described make up the central processing unit or CPU of your home computer and is typically packaged in a large metal box most people just call “the computer”. Now lets assume you could shrink all those components; microprocessor, ROM, RAM and I/O, system clock into a single integrated circuit. It can be done and has been done. It’s called a microcontroller and runs many of the products you already are familiar with such as mp3 players, cell phones or video game players.

A microcontroller is a miniature computer in a single integrated circuit with a small amount of ROM and RAM and lots of I/O. Figure 1-1 shows some of the various sized microcontrollers from Microchip Technology Inc. They offer a full line of microcontrollers that they call the PIC® microcontrollers. They offer many more than shown here but for these are very common packages that range from an 8 pin small package up to a 40 pin large package with lots of I/O.

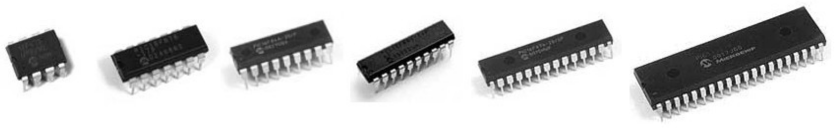


Figure 1-1: Microchip PIC Microcontrollers

How Does a Microcontroller Work?

A microcontroller requires a series of coded electrical charges stored into its ROM to control the micro's I/O. These electrical charges take on two states; high voltage or low voltage. You can think of these charges just like a light switch where it's either on or off. These charges can also be represented mathematically by the binary number system which only includes the digits 1 and 0. The microcontroller is designed to do different operations on the I/O depending on the arrangement of these charges. An eight bit microcontroller uses 8 different charges combined to determine the operation to perform. This is also known as a byte value. Figure 1-2 shows an 8-bit byte value. There are 256 different arrangements of bits in an 8-bit value. Each one can represent a different operation in the microcontroller.

1	0	0	1	0	0	1	0

Figure 1-2: 8-bit byte value

When you have multiple bytes put together it is known as software or program code. When a microcontroller is said to be programmed or have code burned into it, it is getting these coded electrical signals stored into its ROM.

To function or run the code, the microcontroller needs a way to select each command from ROM one at a time, which is referred to as running a program. To do this, the microcontroller requires a clock oscillator, often times this is created with a separate part called a crystal or resonator connected to the microcontroller to create a continuous internal pulse train that drives the microcontroller's central circuitry. This is very similar to the PC Gigahertz speed clock. Gigahertz is one billion pulses per second. The speed in a microcontroller is much slower in the megahertz or million pulses per second. For most applications this is plenty fast enough.

When the micro is first powered up, the oscillator clock starts pulsing the same way our heart pulses our blood through our body. On each pulse of the clock, the micro retrieves a new command from ROM to execute on the I/O. By arranging these binary codes properly you can make the I/O pins switch on and off to control other electrical circuitry connected to the I/O pins. That circuitry could be a simple relay that turns a light on during the night and off during the day or it could be more complex and control the motors of a robot while reading an obstacle sensor. All you need to do is write the series of binary codes properly, which is the software. To make it easier to develop this binary code, compilers were developed. The PICBASIC PRO language compiler is a BASIC language compiler.

What is a Basic Language Compiler?
You could create software but individually setting or clearing each bit in memory but most microcontrollers (PIC's included) offer a software creation tool called assembly language. Assembly language uses short little acronyms to represent various simple operations in the microcontroller and the assembler built into this software will convert those acronyms into the 1's and 0's. Asssembly language can be considered a very cryptic language to most people and because of this compilers were created. A compiler is a PC software application that converts easy to read and understand words (BASIC commands) into an assembly language file and then lets the assembler covert the result into the binary code (1's and 0's) the micro needs. Binary code is the lowest level of software and compiler is considered a high level language. Once that binary code file is created, then the microcontroller can be programmed. The binary file will have the .hex suffix (i.e. program.hex).
The PICBASIC PRO compiler from microEngineering Labs is a BASIC language compiler for the Microchip 8-bit microcontrollers.

The PICBASIC PRO compiler is very powerful and the full version sells for around $250. You can use the full version for developing complex projects and products. In this book though, we will use the sample version of the PICBASIC PRO compiler which is limited to selected PIC microcontrollers and 31 command lines. This may not sound like a lot of capability but you will soon be surprised by how much we can accomplish in 31 commands. The best part is the sample version is free to download so you can try out their great compiler by recreating the projects in this book.

How does the Microcontroller actually get programmed?
A microcontroller programming tool is a custom designed module that receives the binary code file created by the compiler and generates the electrical signals the microcontroller needs to see.

The binary file is downloaded to the program memory or ROM of the device through specific I/O pins on the microcontroller. There are many different types of programming tools and are typically just called a Microchip PIC Programmer.

Figure 1-3 shows the PICkit™2 Starter Kit which is a complete starter kit for programming Microchip Technology microcontrollers. The development board included with the PICkit 2 Starter Kit makes it easy to get started programming Microchip PIC microcontrollers. The development board has a 20 pin socket and comes with a PIC16F690 microcontroller. The development board can be powered from the PICkit™2 programmer or separately. The PICkit™2 gets its power from the PC USB port so the development power is limited to about 50 milliamps. If you needed more power then you can power the board separately. For all the experiments in this book I will let the PICkit 2 or a PICkit 2 clone power the board. The development board has four LEDs, a momentary switch and a potentiometer wired directly to the 20 pin socket. This can offer a quick way for a beginner to get started.

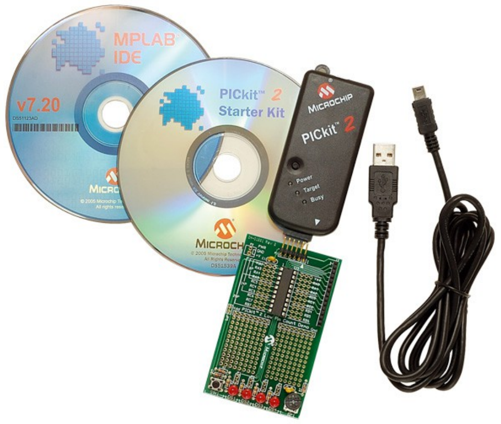


Figure 1-3: PICkit™2 Starter Kit

There are also many clone versions of the PICkit 2 programmer that you can also purchase from various sources. The clone version shown in Figure 1-4 is a shrunk version of the PICkit 2. It has the USB connector built into the end and a ribbon cable with the programming connection wires. This eliminates the need for a separate USB cable and is very compact. I will use this clone in most of the experiments in this book and any future project books that follow.

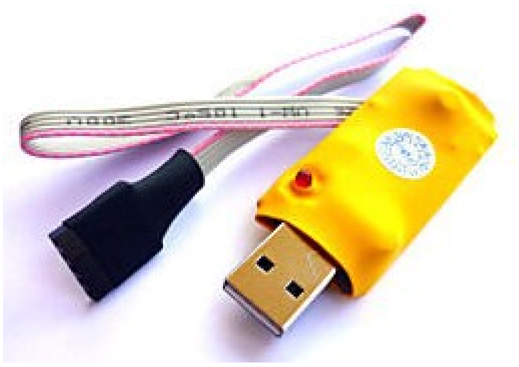


Figure 1-4: PICkit 2 Clone Programmer

This programmer can plug into the same development board as the PICkit 2 Starter Kit or any In-Circuit Serial Programming board which I will cover in a little bit.

In-Circuit Serial Programming

The PICkit™2 has only five or six connections to the development board for powering the board and for programming the PIC microcontroller. Two of the connections are power and ground. Some programmers like the clone PICkit 2 only have 5 connections. For those that have six, the extra pin is for calibrating the internal oscillator on some PIC's. This is rarely needed so many clones leave it off. This leaves three connections that all programmers have and are specifically for programming the code into the microcontroller. All of these programmers use these three pins in a serial method of programming called In-Circuit Serial Programming (ICSP). Having only a few connections to your circuit board can be very handy so I wanted to explain how to use the ICSP feature.

The main advantage to ICSP is the ability to program the PIC in-circuit. The biggest hang-up you may have with ICSP is the serial communication signal can get affected by the circuitry connected to the PIC I/O. For example, the three connections used to program a PIC in-circuit are: Vpp (MCLR pin), Data (PGD pin) and Clock (PGC pin). If the Clock or Data signals are not able to send the correct signal, the PIC will not program properly and you will get a verify error. If you build your own development board or connect circuitry to the programming pins of the PIC then you may affect these signals. There are recommended connection methods from Microchip so let me explain those.

ICSP Hardware Interface

The schematic in Figure 1-5 shows the ICSP connections and all the possible connection issues to watch out for in your design. Because of the way the ICSP feature works, you don't want to add any capacitance to the programming connections since this can delay the signals. Even the capacitance on the Vdd line should be monitored per the PIC programming specification. The PIC programmer actually cycles the Vdd line off and on while sending the Vpp signal to the MCLR pin. This is done to put the PIC in programming mode. If there is too much capacitance, it may slow the signal down and not meet the programming specs. You can get the programming specs for any PIC microcontroller at the Microchip.com website.

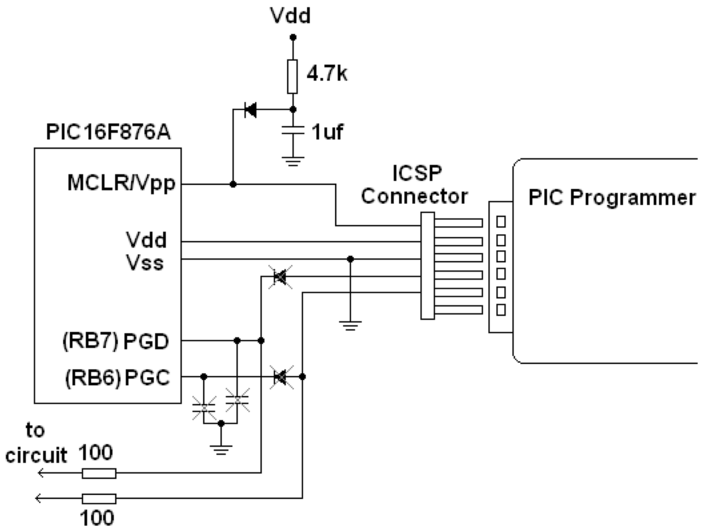


Figure 1-5: PICkit 2 Hardware Interface

You also don't want to load down the clock or data signal. The components that are crossed out show what could affect the signal. The diodes on the Data and Clock lines are a mistake because two-way communication occurs when programming and verifying the part. These are pretty easy to see why they should not be included in your design.

What isn't quite so clear is the diode between the MCLR reset circuit and the MCLR/Vpp pin. This is recommended because the PIC programmer sends a high voltage, low current signal to the Vpp line of around 12v -13.5v for a short period of time. You don't want that signal feeding into your Vdd regulator so the diode helps protect for that. This is actually just an extra safety precaution though because the current entering the MCLR pin is extremely small and the MCLR pull-up resistor will knock it down to prevent any damage. In most cases you can get by without the diode.

Another recommendation which is often missed is the series resistors on the PGD and PGC lines between the PIC and the rest of the circuit. These isolate your circuit from the PGD and PGC signals so your circuit doesn't load down the PIC programmer. This is usually where a problem may occur with ICSP. One hundred ohm resistors should not affect your circuit function but it should be plenty of resistance to isolate the programmer.

Another option to get around all this is to add a switch to your circuit. A four pole switch that allows you to disconnect the PGC, PGD, Vdd and MCLR pins from the circuit during programming prevents the rest of the connected circuitry it from loading down clock and data and disconnects the MCLR resistor from the programming connection. Figure 1-6 shows the schematic for that type of arrangement. This is a premium type of setup and most development board won't offer this connection arrangement.

What is interesting is the development board included with the PICkit 2 Starter Kit only includes some of these protections. You have to make sure you don't add circuitry to the data and clock lines that might affect the programming operation. Therefore it is best to add the 100 ohm resistors to your circuit if you use those I/O pins in your design.

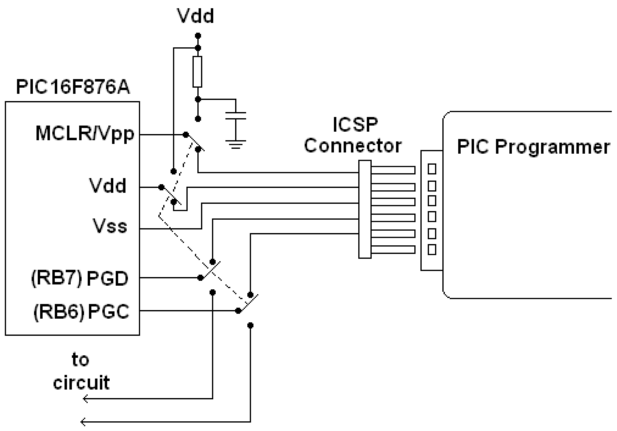


Figure 1-6: Switched ICSP Connection

ICSP Pin-Out

To be helpful I will give you some of the ICSP pins for the various PIC microcontroller DIP packages. Table 1 below summarizes it for you. This should save you the trouble of looking through all those data sheets.

ICSP Connection	40 pin pin #	28 pin pin#	20 pin pin#	18 pin pin#	14 pin pin#	8 pin pin#
Vpp\MCLR	1	1	4	4	4	4
Vdd	11 & 32	20	1	14	1	1
Vss	12 & 31	8 & 19	20	5	14	8
Data	40	28	19	13	13	7
Clock	39	27	18	12	12	6

Table 1: ICSP Connection Pin Numbers

Figure 1-7 shows a simple 40 pin development board I made to test out ICSP with the PICkit™2 programmer. It's not pretty but it worked. I even added a couple LEDs so I could test it with a few flash LED programs. The 16F887 part used here has an internal oscillator so I didn't need to add an external resonator. I prefer to use a breadboard setup for most of my projects so I can easily move wires around so I often just use a breadboard module like the one shown in Figure 1-8.

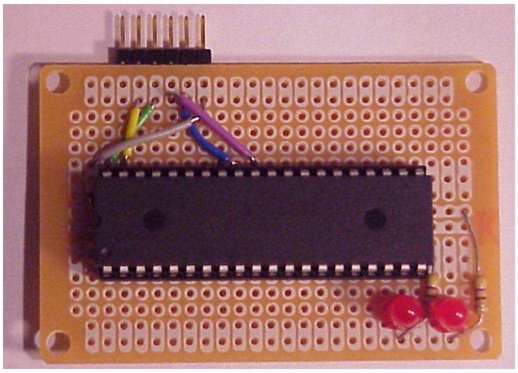


Figure 1-7: Simple Development Board uses ICSP

These small boards with ICSP connectors make it real easy to plug a PIC microcontroller into a breadboard and still program with the PICkit™2 or a PICkit 2 clone programmer. I'll use one of these for the projects in this book. There isn't any 100 ohm resistors included on the board so you may have to add those when you use a PGD or PGC pin. The projects in this book will use some resistance when necessary to eliminate the ICSP interference concern.

<ICSP Breadboard Module>
Figure 1-8: ICSP Breadboard Module

Installing the Software

Now you are ready to get your own programming setup installed on your computer so you can complete the projects in this book. The PICBASIC PRO compiler sample version comes with its own windows editor for actually writing the programs and sending the binary file to the programmer. The editor is called the MicroCode Studio. The PICkit™ 2 programmer has a command line option which allows us to use the PICkit 2 within the MicroCode Studio editor to one click compile and program the PIC microcontroller using the ICSP. We have to set all this up on your PC before we can create the first project. The steps we will take are:

1) Download all the files needed from my website at www.elproducts.com/picbasicbook_vol1.htm.
2) Install the PICBASIC PRO compiler sample version.
3) Install MicroCode Studio Software (this is part of the PICBASIC PRO installation).
4) Setup the PICkit 2 interface and command line software within Microcode Studio.

Installing MicroCode Studio and PICBASIC PRO

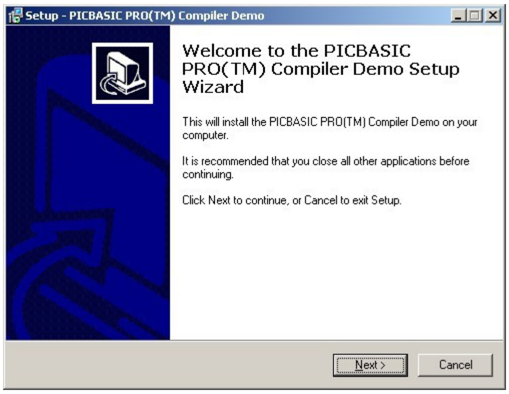
Download all the files for this installation on to your computer. You will need a PC running windows XP or Vista for best performance. You can get the PICBASIC PRO sample version at:

www.melabs.com/pbpdemo.htm

The full set of project files can be downloaded at my website:

www.elproducts.com/picbasicbook_vol1.htm.

You need to run the PICBASIC PRO installation file so run the PBPDEMO4.exe file to install the sample version of the PICBASIC PRO compiler. The compiler will step you through several screens starting with the one shown in Figure 1-9.


Setup - PICBASIC PRO(TM) Compiler Demo

Welcome to the PICBASIC PRO(TM) Compiler Demo Setup Wizard

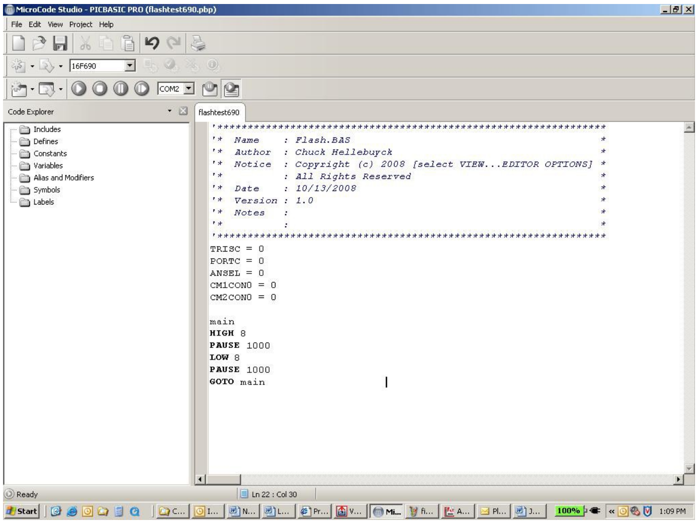
This will install the PICBASIC PRO(TM) Compiler Demo on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 1-9: PICBASIC PRO Installation

When the PICBASIC PRO installation is complete the last screen will offer to “Install MicroCode Studio IDE”. Make sure that option is checked before clicking on the “Finish” button. The MicroCode Studio will install automatically after you press Finish. After installation is complete, start Microcode studio and it should look similar to the picture in Figure 1-10.



```
'****************************************************************
'*  Name    : Flash.BAS                                         *
'*  Author  : Chuck Hellebuyck                                  *
'*  Notice  : Copyright (c) 2008 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 10/13/2008                                        *
'*  Version : 1.0                                               *
'*  Notes   :                                                   *
'*          :                                                   *
'****************************************************************
TRISC = 0
PORTC = 0
ANSEL = 0
CM1CON0 = 0
CM2CON0 = 0

main
HIGH 8
PAUSE 1000
LOW 8
PAUSE 1000
GOTO main
```

Figure 1-10: MicroCode Studio IDE

PICkit2 Setup

First make sure you have the pk2cmd.exe file installed on your computer. It was part of the package of files you download from my website www.elproducts.com/picbasicbook_vol1.htm. You can also download it at www.microchip.com/pickit2.

The steps to set up the PICkit2 in MicroCode Studio are easy once you know what to do. Click on the “View > Compile and Program Options” selection as shown in Figure 1-11. The window in Figure 1-12 should appear.

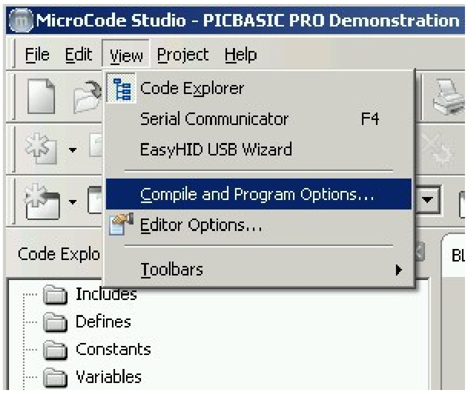
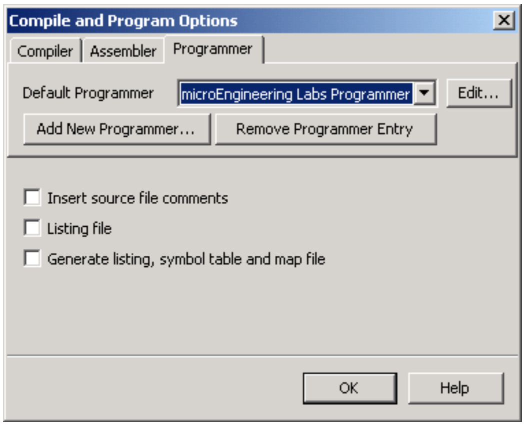


Figure 1-11: Programmer Setup Menu Option

The default programmer is microEngineering Labs own programmer which is another PIC programmer you can consider. I prefer to use the PICkit 2 programmer so we need to set that up.



Default Programmer: microEngineering Labs Programmer

- [ ] Insert source file comments
- [ ] Listing file
- [ ] Generate listing, symbol table and map file

Figure 1-12: Programmer Setup Window

Click on the “Add New Programmer” button to create the PICkit2 setup in MicroCode Studio. Figure 1-13 shows the “Add New Programmer” window that appears. The “Create a custom programmer entry” should already be selected so click on the “Next >” button.

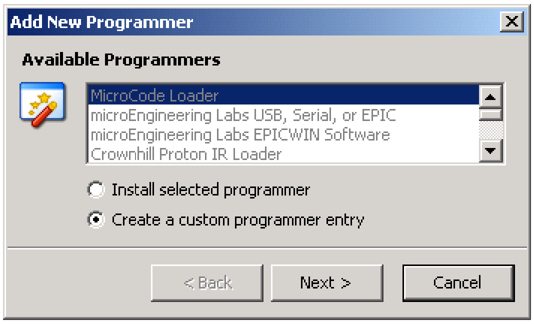
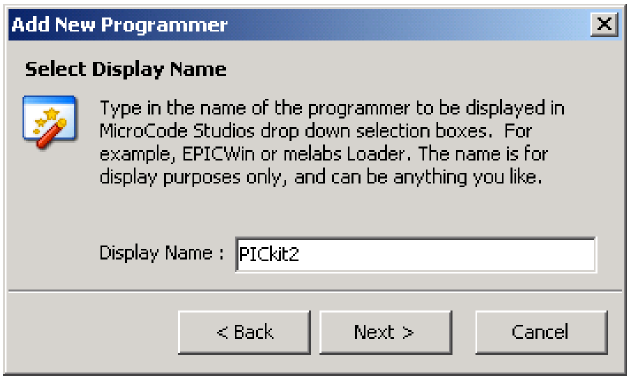

Add New Programmer
Available Programmers
MicroCode Loader
microEngineering Labs USB, Serial, or EPIC
microEngineering Labs EPICWIN Software
Crownhill Proton IR Loader
Install selected programmer
Create a custom programmer entry
< Back
Next >
Cancel

Figure 1-13: Add New Programmer Screen

A second "Add New Programmer" window will appear with a blank line. Enter the name you want to appear in the programmer selection window when you select your programmer at compile time. I chose to call it simply "PICkit2" as you can see in 1-14.

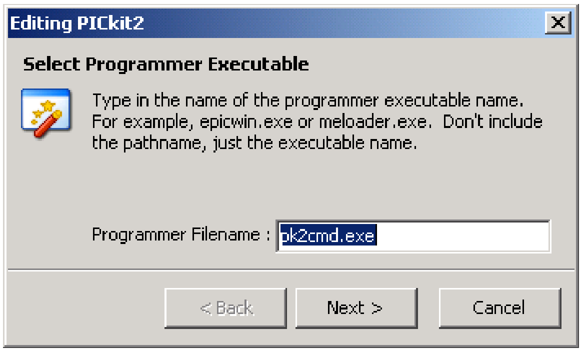


Type in the name of the programmer to be displayed in MicroCode Studios drop down selection boxes. For example, EPICWin or melabs Loader. The name is for display purposes only, and can be anything you like.

Display Name : PICkit2

Figure 1-14: Enter PICkit2 in Window

After you enter the programmer name, click the "Next >" button and the "Editing PICkit2" window will ask for the PICkit2 command line executable. If you chose a different name the window title will show the name you selected. The command line executable for the PICkit2 is the pk2cmd.exe file. Enter this into the Programmer Filename: window as seen in Figure 1-15.


Editing PICkit2

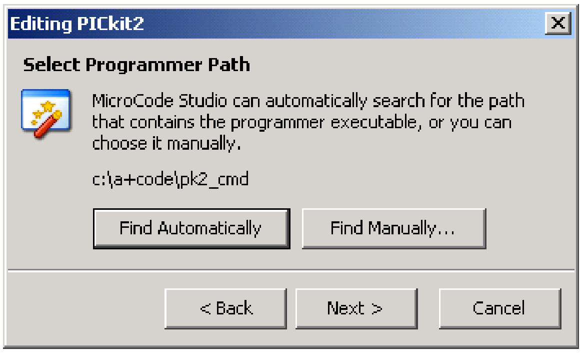
Select Programmer Executable

Type in the name of the programmer executable name. For example, epicwin.exe or meloader.exe. Don't include the pathname, just the executable name.

Programmer Filename : pk2cmd.exe

Figure 1-15: Enter PICkit2 Command Line

You don't have to know where the file is for the next step but when you click on the "Next >" button the screen in Figure 1-16 will pop up asking for the file location. You have a choice in how you want to find the "pk2cmd.exe" executable file. You can have MicroCode Studio search for the command line automatically or you can manually select it yourself if you remember where you put it on your computer. The manual option is a lot faster.


Editing PICkit2

Select Programmer Path

MicroCode Studio can automatically search for the path that contains the programmer executable, or you can choose it manually.

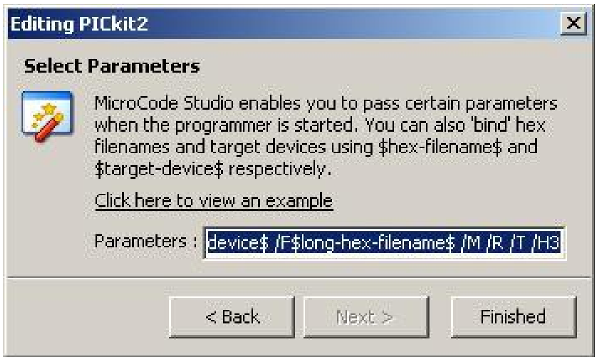
c:\a+code\pk2_cmd

Figure 1-16: Find the PICkit2 Command line

The next step I'm going to describe is the most critical and the most difficult because these are the command line options that will transfer the PIC MCU you are using and the binary .hex file the compiler creates to the PICkit2 prior to starting the PICkit2 command line executable. The command line options include both PICkit2 format and MicroCode Studio format options combined. Enter the command line exactly as you see it here with spacing and capitalization. If you are an advanced PC user then you may modify the command line from what I show below depending on how you want to use the PICkit2. The "Readme for PK2CMD.txt" file that downloaded with the "pk2cmd.exe" file explains all the PICkit2 command line options. I highly suggest you read that over before changing from what I show below.

/PPIC$target-device$ /F$long-hex-filename$ /M /R /T /H3

Figure 1-17 shows the command line entered in the MicroCode Studio screen. Let me explain my choices for those that want to understand how it works.


Editing PICkit2

Select Parameters

MicroCode Studio enables you to pass certain parameters when the programmer is started. You can also 'bind' hex filenames and target devices using $hex-filename$ and $target-device$ respectively.

Click here to view an example

Parameters : device$ /F$long-hex-filename$ /M /R /T /H3

Figure 1-17: Command Line Option Window

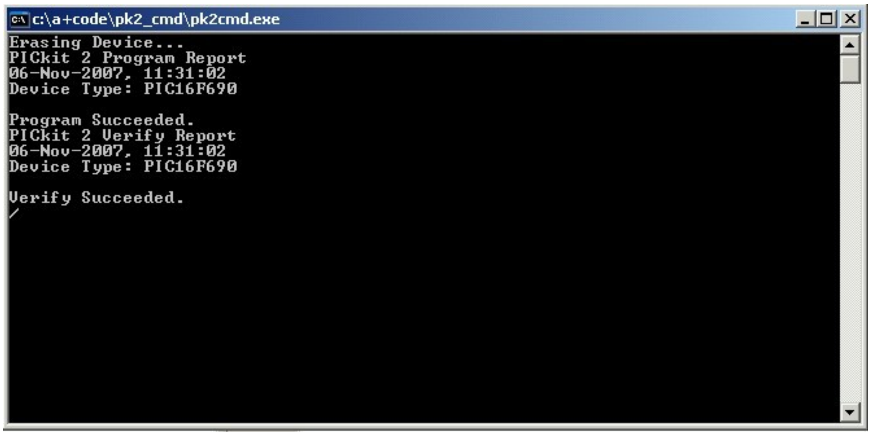
The **/P** option selects the PIC MCU but I wanted MicroCode Studio to pass that on from its own selection window. MicroCode Studio offers that detail as a parameter "$target-device$" but it doesn't include the PIC in front of the number (i.e. 16F690 instead of PIC16F690) so you have to type that in. This gives us the **"/PPIC$target-device$"** section of the command line option.

I’m using the PICBASIC PRO compiler and the **/F** option selects the .hex file the compiler created. MicroCode Studio offers the path to the compiled and assembled .hex file through a parameter called “$long-hex-filename$”. This part of the command line thus becomes **“/F$long-hex-filename$”**.
The rest of the command line options are where you may modify the setup that I have. I first added the **/M** option which directs the PICkit2 to erase and then program all the memory locations including program memory, the configuration, the EEPROM and the ID memory.

The **/R** directs the PICkit 2 to release the MLCR line or reset line after programming. This prevents the programmer from holding the development board in reset mode.

The **/T** option tells the PICkit 2 to power the development board from the USB port. Remember you only have 50 milliamps to use with this.

/H3 is the final option I chose. When the programmer command line starts operating, a pop up window will appear and show the programming operation running. I like to keep that window open for a few seconds after the programming is complete. The /H3 keeps it open for three seconds. Figure 1-18 shows a typical pop up window.

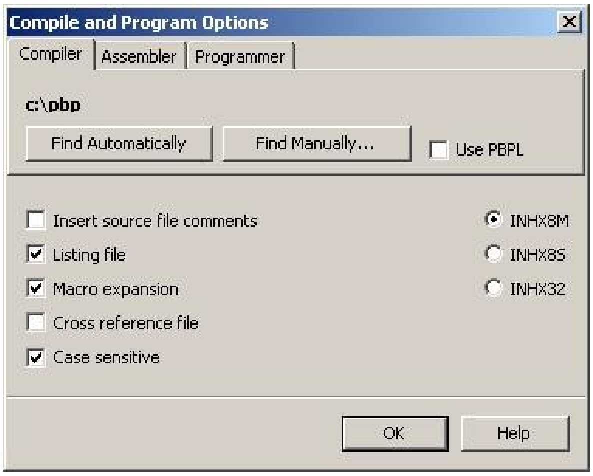


```
c:\a+code\pk2_cmd\pk2cmd.exe
Erasing Device...
PICkit 2 Program Report
06-Nov-2007, 11:31:02
Device Type: PIC16F690

Program Succeeded.
PICkit 2 Verify Report
06-Nov-2007, 11:31:02
Device Type: PIC16F690

Verify Succeeded.
/
```

Figure 1-18: pk2cmd.exe Programming Screen

Special Note:
If you try to program your first part and the programming screen pops up quickly and then disappears, then you probably have a mistake in your command line or the pk2cmd.exe cannot find the PICkit 2 programmer. This could be a problem with the USB port so make sure you have everything connected properly.

After you setup the PICkit 2 programmer you also need to verify PICBASIC PRO can be found by the MicroCode Studio. Click on the “View > Compile and Program Options” as shown in Figure 1-11 again but this time click on the compiler tab. The path to where PICBASIC PRO is stored on your computer should show up as seen in Figure 1-19. Verify this is correct for your setup otherwise MicroCode Studio will give you an error. This should be automatically set up correctly by the installation but it’s best to check.


Compile and Program Options

Compiler | Assembler | Programmer

c:\pbp

Find Automatically | Find Manually... | [ ] Use PBPL

- [ ] Insert source file comments
- [✓] Listing file
- [✓] Macro expansion
- [ ] Cross reference file
- [✓] Case sensitive

[● INHX8M] INHX8S INHX32

OK | Help

Figure 1-19: PICBASIC PRO Setup

That's it. Follow these steps and you should have your PICkit2 running within MicroCode Studio with the PICBASIC PRO compiler and have a one click, compile and program solution.

Configuration Settings

There is one final step you need to understand before we can create our first program. Outside the structure of your program, the PIC has certain bits that are set at program time to control the special internal settings of the microcontroller. These settings control the internal watchdog timer, power-up timer, and oscillator selection and a few more.

All of the options for the part can be seen in the PIC data sheet under the CONFIG register section. The PICBASIC PRO compiler puts those configuration settings in a separate file that it calls at compile time. The settings file will be in an .inc file that has the name of the MCU you are using. The projects in this book will use the PIC12F683 eight pin microcontroller. In this case, the configuration file is named 12F683.inc. You will find it in the

PBPDEMO directory, where you installed the PICBASIC PRO compiler. The file will contain two separate configuration lines.

device pic12F683, intrc_osc_noclkout, wdt_on, mclr_on, protect_off

and

__config _INTRC_OSC_NOCLKOUT & _WDT_ON & _MCLRE_ON & _CP_OFF

This line in the file is where the PICBASIC PRO compiler gets the information on how to set the configuration bits inside the .hex file. In this example, the internal RC oscillator is used as the system clock. This is the setting we will use in the book projects. You can open the 12F683.inc file with notepad and modify it if necessary.

If you needed to use the MCLR pin as a digital I/O pin then you would change the **mclr_on** to **mclr_off** and **MCLRE_ON** to **MCLRE_OFF**. You can then save the file for any future builds.

For the projects in this book you shouldn't need to change the default settings but if you want to change them in the future you now know where to look.

Chapter 2 – Flash an LED

Now we can write our first program to test this out. This first program will simply flash an LED connected to the GP0 pin of the 12F683 eight pin microcontroller. This is a simple project but proves out the whole process of writing software, programming the microcontroller and watching the application run.

Figure 2-1 shows the completed project built into a breadboard. The PICkit 2 clone programmer powers the board and the RED LED will flash at a rate of ½ second on and ½ second off.

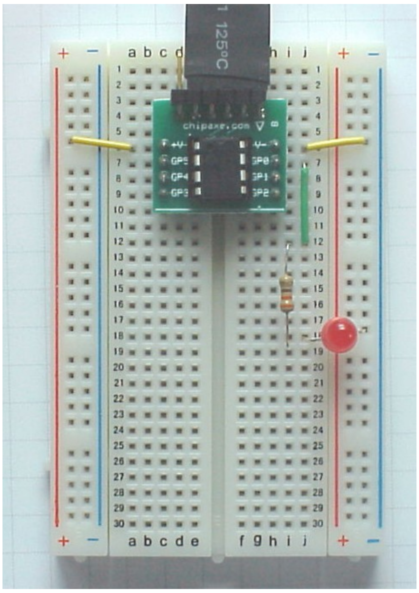


Figure 2-1: Final Flash LED Project

How to Program:

1) Enter the program into the editor window
2) Make sure Target Processor window in MCStudio shows "12F683"
3) Connect the PICkit 2 clone programmer to the development board with the PIC12F683 in its socket.

4) Click on the little arrow next to the compile/program button (it's next to the target processor window from step 1) and make sure "PICkit2" is selected.
5) Click on the compile/program button.

This should compile your program and bring up the PICkit 2 command line pop-up window. You should see it program and then complete the process. The pop-up window should close after the three second delay. The RED LED should be blinking on the development board. If you don't get this working, go back through the steps and see if you missed something. Getting a simple LED to flash is a great

Hardware

The hardware is built on a breadboard that has letters lined up with the column of connections and numbers for the rows. The connections can be reproduced based on the table of connections below. Figure 2-2 also shows the schematic for this project.

<u>**Connection Table**</u>

```
Micro        - Pin 1 at C6
Vdd Jumper   - a6 to +rail
Vss Jumper   - j6 to -rail
Green Jumper- j7 to j12
330 ohm      - i12 to i18
Red LED      - Anode j18, Cathode -rail
```

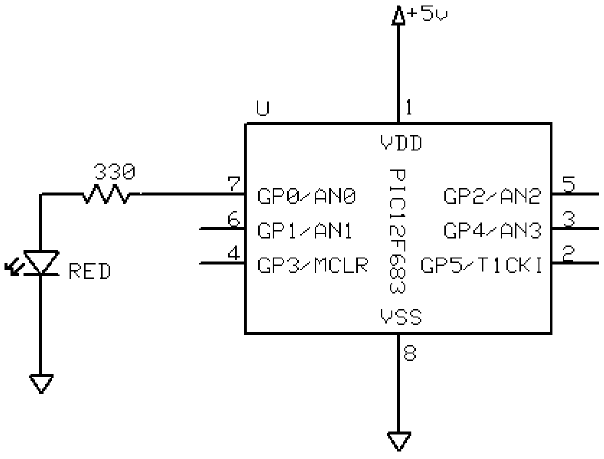


Figure 2-2: Flash LED Schematic

Software

The software is quite simple because it uses some of the commands that make PICBASIC PRO easier to use that many other compilers. The first steps though require the I/O to be setup as digital. When the PIC12F683 is first powered up, the I/O defaults to analog mode. The I/O pins share a connection to both analog and digital features. The ANSEL = 0 sets all the I/O pins to digital mode.

```
ANSEL = 0  ' Set I/O to digital
```

The 12F683 also has an internal comparator which can be shut down with the CMCON0 = 7 line.

```
CMCON0 = 7 ' Comparator off
```

The main program loop begins with the label main followed by a semi colon. We will use this a marker in a future GOTO command.

```
main:
```

The I/O pins on the 12F683 are referred to as General Purpose Input Output pins or GPIO. The internal register that controls these pins individually is also called the GPIO register. These can be controlled by writing to the GPIO register directly but we would also need to setup the TRISIO register inside the 12F683. The TRISIO determines if the pins are a digital input pin or a digital output pin. Both of these are automatically controlled with the **HIGH** or **LOW** command.

The GPIO.0 is the nickname for the GP0 pin. The software uses the HIGH command to place a high signal on that pin. This will light the LED. PICBASIC PRO doesn't care if you use capitals or small case letters. The MCStudio should recognize the command an automatically capitalize the command.

```
HIGH GPIO.0  'LED on
```

The next command is the **PAUSE** command. This command just creates a ½ second delay as the value 500 represents 500 milliseconds or ½ second.

```
PAUSE 500    'Delay 1/2 second
```

The program then turns the LED off by setting the same pin low.

```
LOW GPIO.0   'LED off
```

The same pause command line delays another ½ second.

```
PAUSE 500    'Delay 1/2 second
```

The program then uses the GOTO command to jump back to the main label and repeat the operation over again.

```
GOTO main    'Loop back and do it again
```

The complete program is shown below for you to type into the MCStudio window or you can load it from the files download.

Software Listing

```
'****************************************************
'*  Name    : Blink.BAS
'*  Author  : Chuck Hellebuyck
'*  Notice  : Copyright (c) 2009 Electronic Products
'*          : All Rights Reserved
'*  Date    : 8/20/2009
'*  Version : 1.0
'*  Notes   :
'*          : CHIPAXE-8 Pin 1 at C6
'*          : Vdd Jumper - a6 to +rail
'*          : Vss Jumper - j6 to -rail
'*          : Green Jumper - j7 to j12
'*          : 330 ohm - i12 to i18
'*          : Red LED - Anode j18, Cathode -rail
'****************************************************

 ANSEL = 0  ' Set I/O to digital
 CMCON0 = 7 ' Comparator off

 main:
 HIGH GPIO.0  'LED on
 PAUSE 500    'Delay 1/2 second
 LOW GPIO.0   'LED off
 PAUSE 500    'Delay 1/2 second
 GOTO main    'Loop back and do it again
```

Next Steps

Simple next steps are to change the pause value to a lower number to flash the LED faster. You could also connect the LED to a different pin and then change the number in the high and low command lines to make that new connection pin flash the LED.