University of Hawaii EE 361L
MPLab Quick Tutorial and Project 2.1

Last updated September 1, 2011

This is a quick tutorial of programming the PIC 16F684A processor using the MPLab Integrated Development Environment (IDE) v. 8.5. Programming the processor has the following steps:

- Create a Project in MPLab using the Project Wizard. Alternatively, you can create a new project using the New option or open an existing project.
- Create, add, delete, and edit source code, and in this case the language will be C.
- Compile the code using Build.
- Simulate and debug the program.
- Recompile the code using Rebuild.
- Set up the processor chip on a protoboard.
- Program the processor using Pickit 2.
- Run the processor.

You can get a copy of MPLab for FREE. Go to the following web site and you will find a zipped version

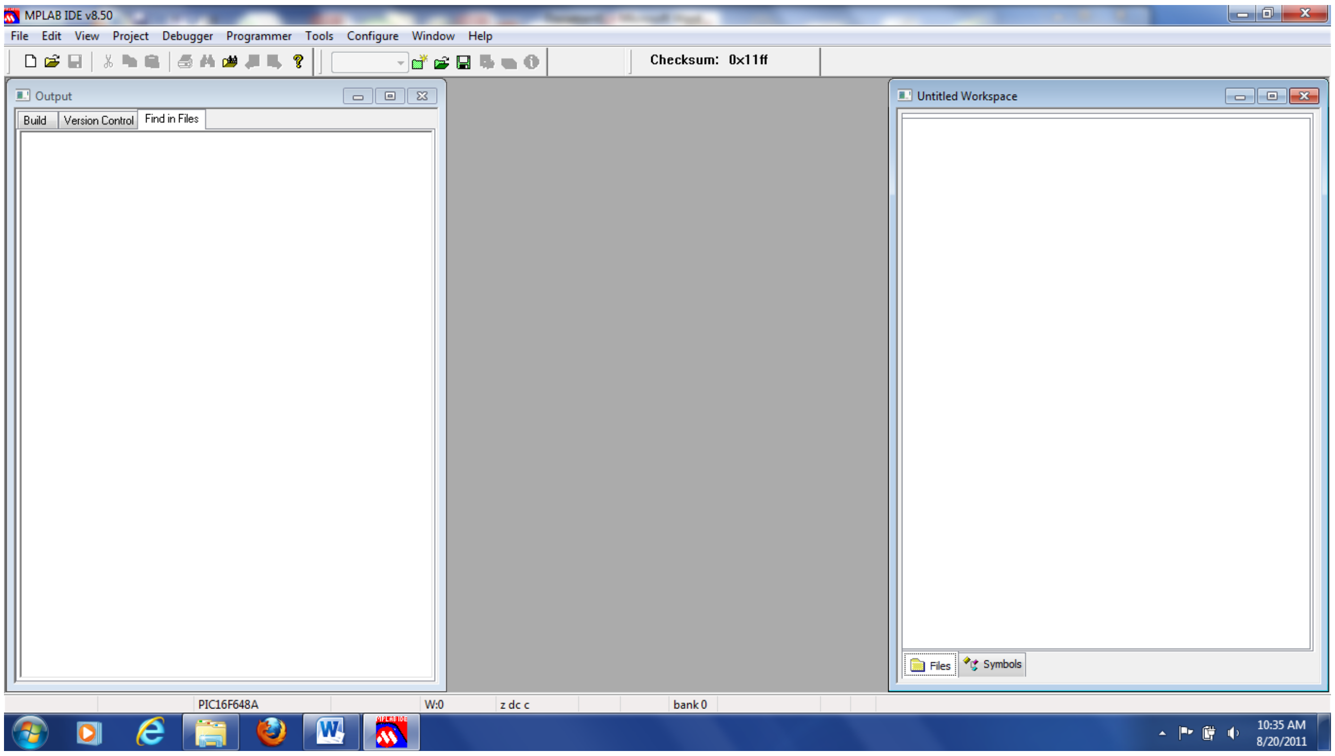
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

The current version at the time of this writing is 8.76. This is a newer version that the one we'll use in the labs which is version 8.5 (this older version came with the PIC programmers, so that's what we installed).

Contents

1 Create a Project

Open MPLab.


MPLAB IDE v8.50
File Edit View Project Debugger Programmer Tools Configure Window Help
Checksum: 0x11ff
Output
Build Version Control Find in Files
Untitled Workspace
Files Symbols
PIC16F648A W:0 z dc c bank 0
10:35 AM 8/20/2011

Go to Project->Project Wizard

Step 1: Go to Next. Then for Device, select PIC16F648A, which is our processor

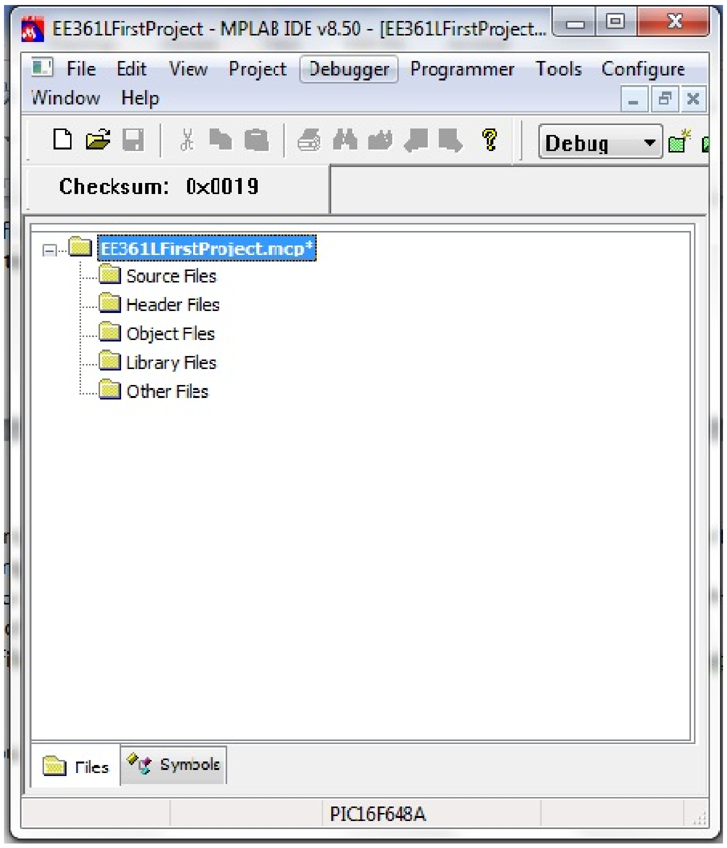
Step 2: For Toolsuite Contents, select HI-Tech ANSI C Compiler

Step 3: For Create New Project File, browse and select a folder to store you files. For this tutorial, go to the Desktop (by going up folders). Then create a new folder, and give it a name such as EE361LMPLabTutorial. Go into the folder, then give the project file a name such as EE361LFirstProject and save. Note that the project file has the suffix mcp.

Step 4: Add any existing files to your project but you don’t have any so skip this by selecting Next. Then you are Finished.

2 Create Source Code

In one of the windows, you should see the project files:


EE361LFirstProject - MPLAB IDE v8.50 - [EE361LFirstProject...
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x0019
EE361LFirstProject.mcp*
Source Files
Header Files
Object Files
Library Files
Other Files
Files
Symbols
PIC16F648A

You can add source files, which can be done in a number of ways. One way is to go to File->New which will open a text editor window. You can write your C language program there. Type the simple program in Figure 2.1. To save the program, go to File->Save As. Then save it in your project folder as “simple.c”.

Add simple.c to your project’s Source Files folder by right clicking the folder’s Icon. simple.c should appear under the Source Files folder icon.

```
/* Simple test program for PIC 16F648A
 * It's a while loop that changes the outputs
 * RB0 and RB1 to zeros and ones
 */
#include <htc.h>  // Header file for PIC processor library files

main(void)
{
int i;
i=0;

TRISB = 0x00; /* RB0 and RB1 are outputs */

/* RB1 is an output and the rest of PORTA are inputs */
while(1) { /* Turn RB0 and RB1 on-and-off */
  if (i < 2) ) {
     RB0 = 0;
     RB1 = 0;
     RB0 = 1;
     RB1 = 1;
  }
  else {
     RB0 = 0;
     RB1 = 0;
  }
  i++;
  if (i >= 4) i = 0;
} /* End while-loop */
} /* End main */
```

Figure 2.1. A simple program for the PIC: simple.c.

3 Build, Simulate, and Rebuild

Before compiling, set the configuration bits by going to the "Configure" menu and selecting "Configuration Bits". Click the "Configuration Bits Set in Code" and select the following.

OSC	XT		BODEN	Enabled
WDT	OFF		LVP	Enabled
PUT	Disabled		CPD	Disabled
MCLRE	Enabled		CP	Off

To compile the program, go to Project->Build.

To simulate the program, go to Debugger->Select Tool->MPLAB SIM

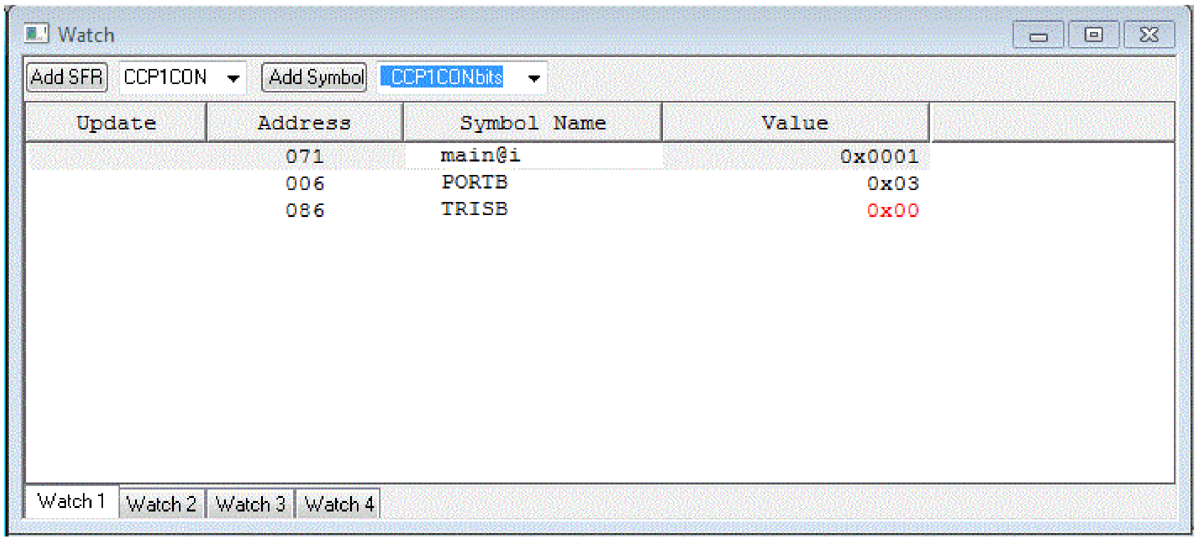
Then reset the simulation with Debugger -> Reset -> Processor Reset

There are a couple of options to run the simulation. Here's the simplest. Go to Debugger->Step Over (F8). (You can also use Step Into (F7)). This will "single step" through the program. The current instruction is shown either as a green-arrow or some other highlight. To continue simulating one instruction at a time, keep pressing the F8.

On the other hand, if you select Run, then the simulator will just run the program. To stop it, you can Halt. The simulation will often run too quickly for you to observe what's going on. So single stepping is more useful in many cases. But try it to see what happens, you won't break the machine.

If you make changes to the source code, you can Rebuild rather than Build.

There are different ways to watch the variable values of the simulation change over time. They can be found under the menu "View". You can watch the PICs hardware (such as PORTB, RB0, and RB1) using the Simulator Logic Analyzer and Special Function Registers. The Logic Analyzer will display the values as a graph over time. Another way is "Watch". This opens a window where you can select the PIC hardware you would like to view (e.g., PORTB, TRISB) as well as variables from your program (e.g., the variable "i" in Figure 2.1 is the variable "main@i" in the Watch window). You can select two types of values: SFR which are the registers in the PIC, and Add Symbol which are variables in your program such as "i". See Figure 3.1 for an example snapshot of Watch.



Update	Address	Symbol Name	Value
	071	main@i	0x0001
	006	PORTB	0x03
	086	TRISB	0x00

Figure 3.1. An example snapshot of the Watch window.

Another window to trace your simulation is "Simulation Trace" which you can find in the View menu. This will show the machine instructions executed and when they were executed (in CPU clock cycles). There are a number of variables that Simulation Trace will follow. You can enable/disable the variables to view by right clicking the Simulation Trace window and then selecting.

Reset your program and single step. Observe how the values change.

Now let's change the simple.c program so that it accepts an input from port RB2, as shown in Figure 3.2 which also shows the modifications. Note that TRISB has to be changed so that the bit 2 is set to 1, indicating that RB2 is an input port. Recall that 0x04 in binary is 0000 0100.

Also, the if-statement is changed so that it is conditioned on RB2. Thus, the program flow will depend on the value of RB2.

To cause input changes to occur in the simulation, we use the "Stimulus" option that's part of the Debugger. Selecting Stimulus will open a window, where you can control the input values to the PIC. Let's have the input RB2 be initially zero. Select the tab Pin/Register Actions. There are two columns: the left corresponds to time and these values are in decimal (dec), and the right corresponds to the different pins and their values, which are in binary (bin). In the first row, under Time, enter 0, which indicates our values at the initial time 0. We can fill the row with different pins but in our case we're only interested in RB2. Click the where it states "Click here to Add Signals", and add RB2. Then enter the value 0 for the pin. Also, make sure that "Repeat" is disabled.

Select the Asynch tab. This allows you to change signals during simulation by the user input. For now, we're interested in the Fire, PIN/FSR, and Action columns. In the first row, select RB2 under PIN/FSR. Then for action select Set High. Now, this row is set up so that when we click "Fire", the RB2 pin will go high. Click the Apply button so that the Stimulate information is attached to your simulation.

```
/* Simple test program for PIC 16F648A
 * It's a while loop that changes the outputs
 * RB0 and RB1 to zeros and ones
 */
#include <htc.h>  // Header file for PIC processor library files

main(void)
{
int i;
i=0;

TRISB = 0x04; /* RB0 and RB1 are outputs */

/* RB1 is an output and the rest of PORTA are inputs */
while(1) { /* Turn RB0 and RB1 on-and-off */
  if (~~i < 2~~  RB2==1) ) {
     RB0 = 0;
     RB1 = 0;
     RB0 = 1;
     RB1 = 1;
  }
  else {
     RB0 = 0;
     RB1 = 0;
  }
  i++;
  if (i >= 4) i = 0;
} /* End while-loop */
} /* End main */
```

Figure 3.2. Modified simple2.c that accepts input from RB2. The changes are highlighted in yellow.

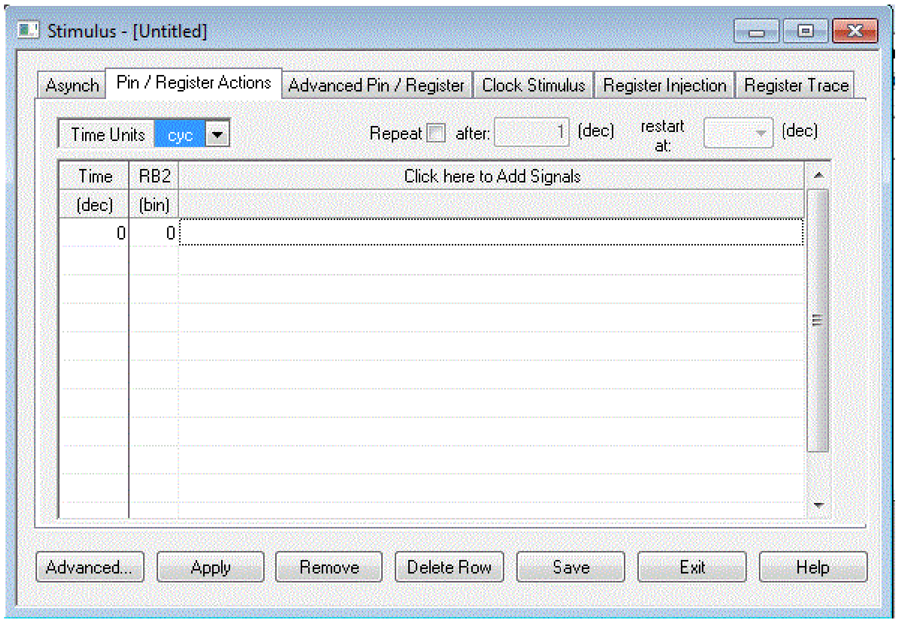
Run the modified simple1.c program by single stepping. Run it through the while-loop a few times. Notice that RB2 is zero so that it executes the else-statement rather than the if-statement. You can also check the Watch window.

Then click the Fire button in the Stimulate window. This will cause RB2 to change value the next time you single step. Now the if-statement is executed rather than the else-statment. You can also view the Logic Analyzer to see the signals varying over time.

Figures 3.3 show snapshots of the two tabs of the Stimulus window.

You can also generate periodic signals from the Stimulus window by clicking the Repeat option. Then add signals so they change over time, each row being a different time. These input signals will repeat when the simulation is run.

For more information about MPLAB SIM, select Help and under Topics select the MPLAB SIM under Debugger. This is a tutorial. You can go over the various chapters of the tutorial including the chapter on Using Stimulus.



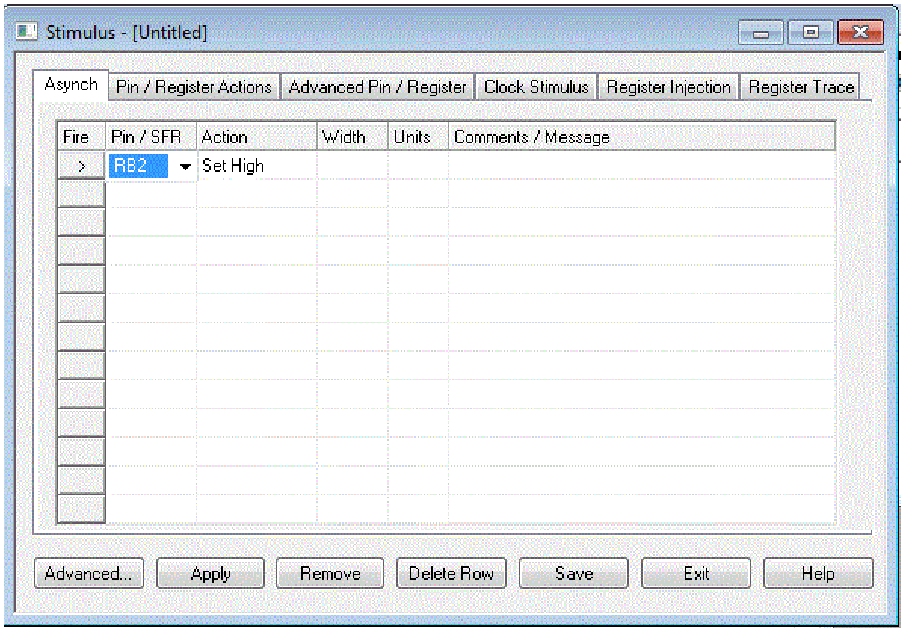


Figure 3.3. The Stimulus window: top is the Pin/Register and the bottom is Asynch.

Enclosed with this manual is a program testlab1.c that will have the PIC 16F648A control an LED at output port RB0. The LED will blink every half-second. You can simulate this program. However, single stepping isn't as helpful because this program will run millions of instructions before any changes in the output RB0 occurs – very long delay loops. In this case, you should "Run" the program. Let it run for a while and then Halt. You can verify that the output is blinking by checking the Logic Analyzer. In the Analyzer, you must select the Channels to view – there's a Channels button where you can choose RB0 to view. Actually, the blinking transitions take so long that you can see a square wave in the window of the Analyzer. Though you can see it transition from low to high or from high to low.

4 Hardware Setup

This section explains how to connect up PIC 16F648A to the Pickit 2 via the protoboard.

Connect the Pickit 2 to a protoboard using wires, which are slightly thicker than the ordinary wires we use for protoboards (ordinary wires are used to connect telephones but we need thicker wires for the connectors in Pickit 2). These wires should be in a small plastic bag (snack bag size) that comes with your Pickit 2. Each Pickit 2 should be in a plastic bag with a number, that corresponds to a lab bench. The Pickit 2 is shown in Figure 4.1. It also comes with a USB 2.0 connector.

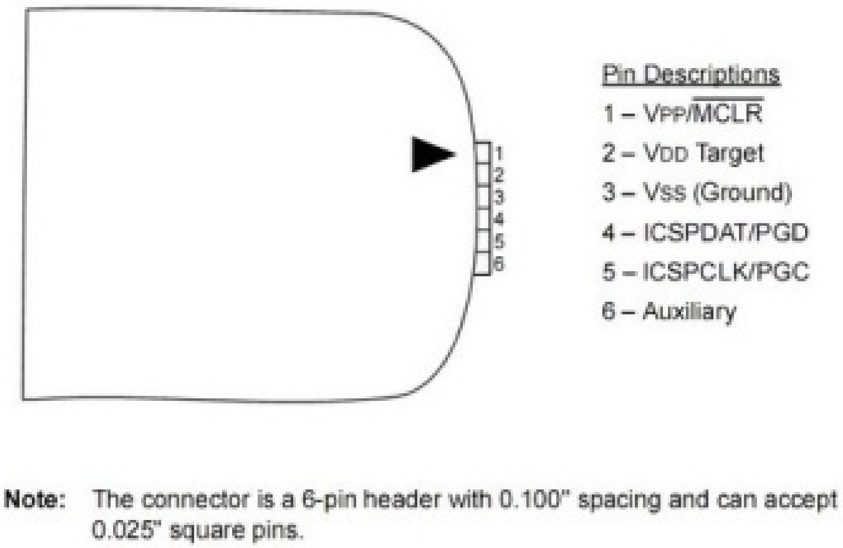


Figure 4.1. Pickit2 and its pin out.

Figures 4.2, 4.3, and 4.4 have the information you need to connect the PIC to the Pickit 2 programmer. This is called in-circuit, serial programming (ICSP) because the programmer (the Pickit 2) is can be attached to the PIC while the PIC is running. You can reprogram the PIC without removing it from the protoboard.

Figure 4.2 has a typical set up to connect the PIC to Pickit 2, but the pinouts shown are not of the PIC 16F648A. The pinouts for the PIC 16F648A are shown in Figure 4.3. Note that ICSPDAT and ICSPCLK in Figure 4.2 correspond to PGD and PGC, respectively in Figure 4.3.

Figure 4.4 shows the PIC's configuration with the crystal oscillator. We'll be using 4MHz crystals so the mode should be "XT". The capacitors in the figure should be 15-30 pF. We will not use the series resistor.

Connect the Pickit 2 to your computer using the USB 2.0 port.

Connect an LED to port RB0. Make sure the LED is connected in the correct way. You can test which side goes to 5v by first connecting the LED to a 5 volt source and ground. The long lead of the LED should connect to RB0 and the short lead should connect to ground.

For this project, you should be able to use the 5v and ground from the Pickit 2 to supply power and ground for your PIC and LED. Other projects may need a separate power supply but this project has few current draining components.

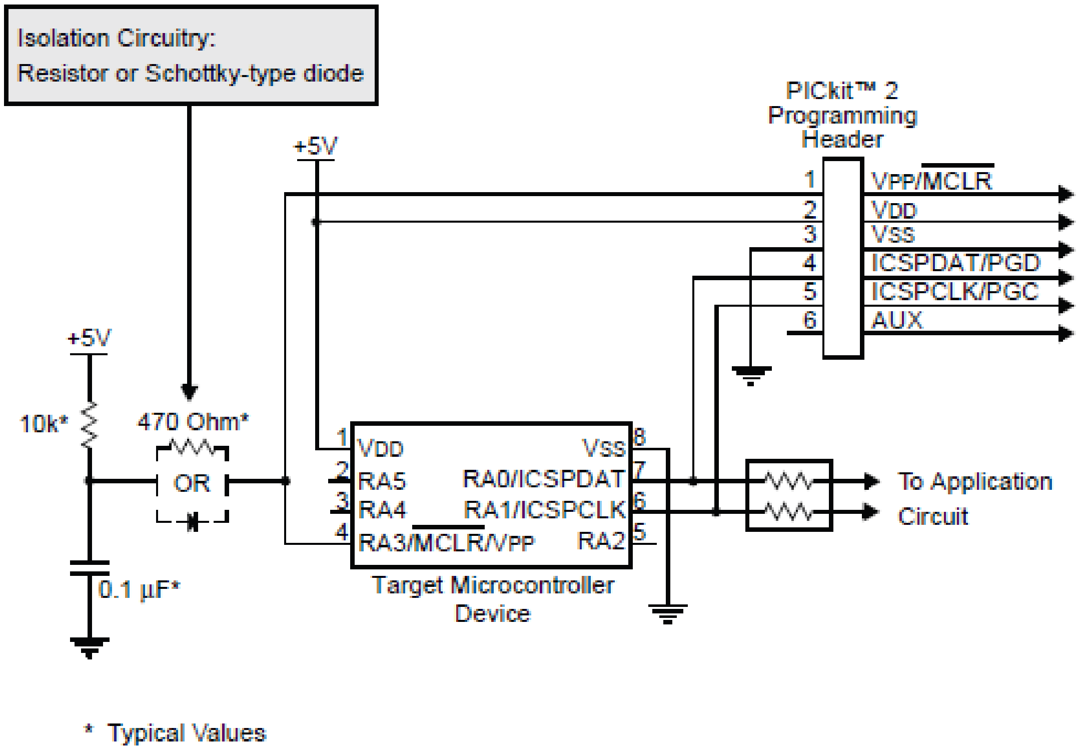


Figure 4.2. Typical set up for the Pickit 2 and the PIC processor, but not the correct pin numbers for 16F648A.

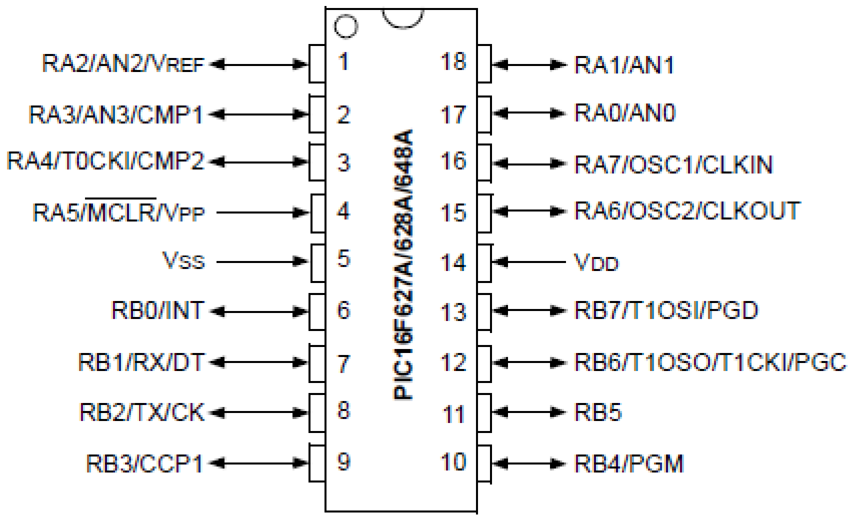
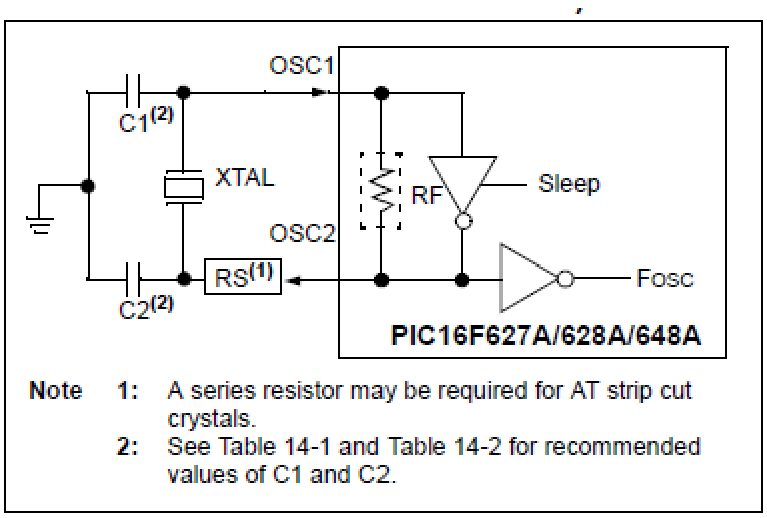


Figure 4.3. Pin out for the PIC 16F648A.



Note 1: A series resistor may be required for AT strip cut crystals.
2: See Table 14-1 and Table 14-2 for recommended values of C1 and C2.

TABLE 14-1: CAPACITOR SELECTION FOR CERAMIC RESONATORS

Mode	Freq	OSC1(C1)	OSC2(C2)
XT	455 kHz 2.0 MHz 4.0 MHz	22-100 pF 15-68 pF 15-68 pF	22-100 pF 15-68 pF 15-68 pF
HS	8.0 MHz 16.0 MHz	10-68 pF 10-22 pF	10-68 pF 10-22 pF

TABLE 14-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Mode	Freq	OSC1(C1)	OSC2(C2)
LP	32 kHz 200 kHz	15-30 pF 0-15 pF	15-30 pF 0-15 pF
XT	100 kHz 2 MHz 4 MHz	68-150 pF 15-30 pF 15-30 pF	150-200 pF 15-30 pF 15-30 pF
HS	8 MHz 10 MHz 20 MHz	15-30 pF 15-30 pF 15-30 pF	15-30 pF 15-30 pF 15-30 pF

Figure 4.4. PIC's configuration with the crystal oscillator.

For this tutorial, omit the resistors and capacitors. Connect the crystal to OSC1 and OSC2 (and forget the capacitors – of course don't connect the crystal to ground).

For the Pickit 2 header, connect the outputs

- VPP/MCLR: Pin used to reset the PIC. Reset is 0v and to let the PIC run is 5v. By connecting this to the PIC's MCLR pin, the MPLAB software can control this pin, which means you can reset the PIC by the MPLAB software. Note that MCLR has an overbar in the documentation. This means that the pin is enabled with a low voltage. So a low voltage will reset the PIC, and a high voltage will allow the PIC to run.
- VDD: Voltage source. You can use this as the voltage source for the PIC. This can be controlled by the MPLAB software.
- VSS: Ground
- PGD: Used to program the PIC. It is the data connection.
- PGC: Used to program the PIC. It is the clock signal.

to the protoboard. Then from the protoboard, connect these outputs to the different pins of the PIC. Use different colored wires to help distinguish them.

5 Programming the PIC

Programming can be done using the Programmer menu in MPLAB. Be sure that you select Pickit 2 as the programmer. Then Connect. This should check if the Pickit 2 and the PIC are connected properly.

Then “Program” which should download the program into the PIC. You can Verify if the programming was successful.

At this point, the PIC may not run because it may be in Reset mode. You can Release from Reset. This changes the MCLR value to high, disabling it and allowing the PIC to run. To reset the PIC, you can select Hold in Reset.

The LED should be blinking.

Read the program testlab1.c if you haven’t already.

6 Manuals and Tutorials in MPLAB

MPLAB has manuals and tutorials that you may want to explore

- Hi-Tech C Manual: This is the manual for the C compiler and it can be found in the Debugger menu. Chapter 7 is useful because it gives library of available function calls and methods to set fuses and registers.
- MPLAB SIM Manual: This can be found in the Help menu. There is a tutorial to explain how to run a simulation. Most of you can follow much of this tutorial. It’s somewhat straight forward to follow if you’re familiar with microcontrollers.
- Other Manuals can be found in the Help menu too.

7 Lab 2.1 Project

Write a program for the PIC 16F648A so that RB0 and RB1 are two outputs that drive two LEDs, and RB2 is an input, which is connected to either 0v or 5v, for logic 0 or 1, respectively. When input RB2 is 0 then the LEDs blink on-and-off simultaneously every half-second. When input RB2 is 1 then the LEDs blink on-and-off every half-second but not together, i.e., when one LED is on the other is off.

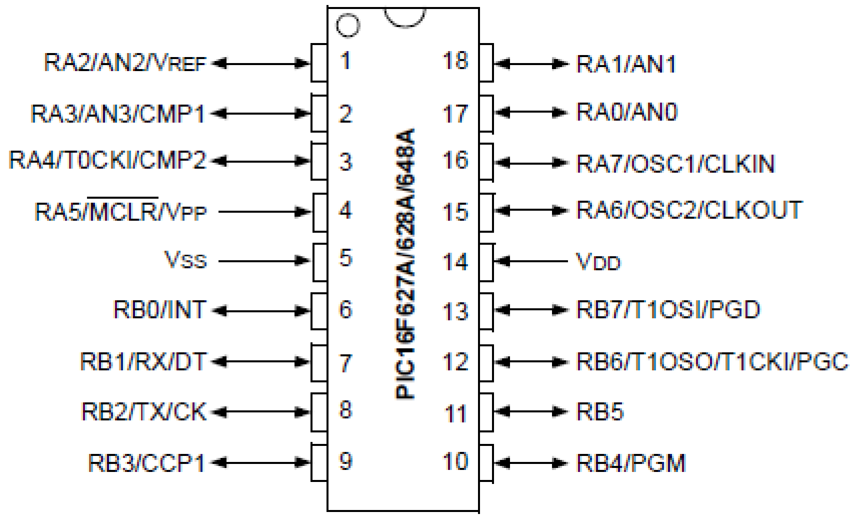
You can modify the program testlab1.c.

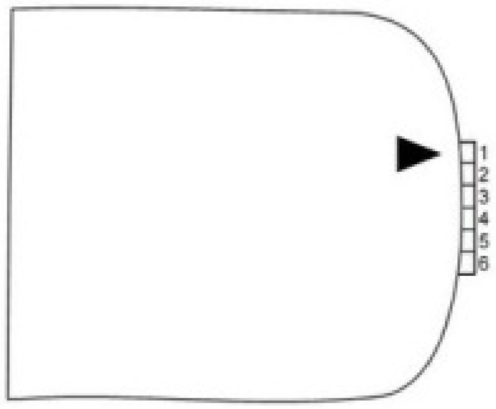
Show your TA that it works. Include your code in your lab report.

Comments: The MPLAB C compiler does not appear to give good error messages. Error messages give the type of errors, which can be looked up in the Hi-Tech C compiler manual. It also appears to give the line number where the error occurred but doesn’t have a good way to jump to the error. One approach to finding the location of an error is commenting chunks of code out and compiling. Perhaps even to the point where your program is simply “main( ) { }”. When it finally compiles then you uncomment code until it doesn’t compile again. That will lead you to the location of the error. If you find a better way to find and correct compile errors, share them with your fellow students.

8. Circuit Diagram

Extra page to draw your circuit diagram.





Pin Descriptions
1 – VPP/$\overline{\text{MCLR}}$
2 – VDD Target
3 – VSS (Ground)
4 – ICSPDAT/PGD
5 – ICSPCLK/PGC
6 – Auxiliary

Note: The connector is a 6-pin header with 0.100" spacing and can accept 0.025" square pins.