Chapter 2

First C Program

In order to program the microcontroller we are going to:

- Write the code in C.
- Convert the code to a hex file using a compiler.
- Program the hex file into the microcontroller.

The code which we are going to write in the C language can be written on any text editor such as WORD. Any suitable C compiler can be used to convert the code to a hex file and there are numerous programmers on the market that will blow your hex file into the microcontroller.

Throughout this book I am going to use a dedicated piece of software called MPLAB integrated development environment (IDE) written by the PIC microcontroller manufacturer, Microchip. This acts as a text editor, compiler, and driver for the Microchip programmer. MPLAB IDE is free and can be downloaded from the Microchip Web site at Microchip.com

At the time of writing Microchip have upgraded MPLAB v8 and have called it MPLABX. I have discussed both IDEs here and left it up to the reader to decide which one they prefer to use.

MPLAB and MPLABX also include a simulator that help to debug your code. I use Microchips own programmer/in-circuit debugger (ICD) called Microchip MPLAB ICD3 and PICkit3. The ICDs allow you to connect your circuit to the computer so that you can view the registers inside the micro when the program is running. But we can see more of the simulator and debugger later.

MPLAB AND MPLABX INSTALLATION

Install the latest version of MPLAB, as of 7/1/2011 that is MPLAB v8.73a., and the C compiler is MPLAB C v3.40 LITE. NB. MPLAB and the LITE version of the C compiler are free from Microchip.com

Or install MPLABX IDE and the C compiler XC8.

Make a new folder to keep your programs in, say PicProgs on your desktop.

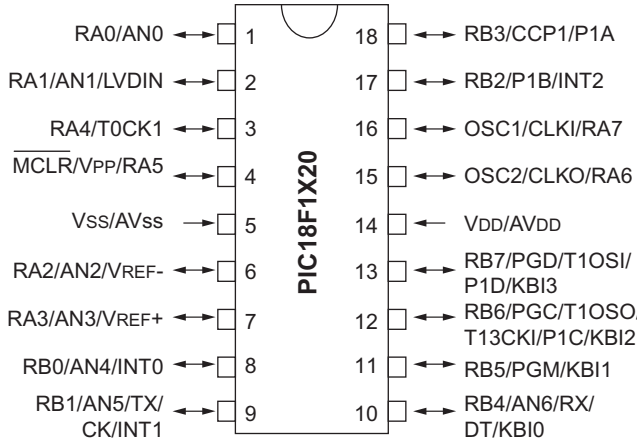

FIGURE 2.1 The 18F1220 pin diagram.

For our first program we are going to flash an LED on and off at 1 s intervals on the output pin, PORTB,4.

The pin connection for the 18F1220 is shown in Figure 2.1.

But before we program our device we need to understand a little of the C language.

A BRIEF INTRODUCTION TO C FOR THE MICROCONTROLLER

Turning an output on/off

If we wish to turn an LED on PORTB bit4 on, the C code is:

PORTBbits.RB4 = 1;
This is called a statement. NB all C statements end in ;
If we wish to turn the LED off the code is:
PORTBbits.RB4 = 0;

Delays

In the C language suite we have installed a file called Delay.h. As its name suggests there are a number of routines in this file which can create a delay in our program. The address for this file if you want to read it is "C:\Program Files\Microchip\mplabc18\v3.40\h" after installing MPLAB from Microchip.com

The subroutines are:

Delay1TCY()
Delay10TCYx()
Delay100TCYx()
Delay1KTCYx()

Delay10KTCYx()
If you wish to call a subroutine in C you just state its name, i.e.,
Delay1KTCYx();

These delays are multiples of timing cycles, i.e., 1, 10, 100, 1000, and 10,000.

A timing cycle is the time taken to execute an instruction and it is the basis of the timing in the microcontroller system. The timing comes from the oscillator which can be an external clock source, an external crystal, or an internal oscillator. For now we are going to use the internal oscillator set at 31.25 kHz.

The timing cycle runs at one-fourth of this frequency, i.e., at 7.8125 kHz. This means the period of the timing cycle is 0.128 ms.

So the Delay100TCYx() subroutine will have a time of 100×0.128 ms = 12.8 ms.

In order to achieve a delay of 1 s we would need 78 of these 12.8 ms.

78 × 12.8 ms = 0.9984 s, not quite 1 s but near enough for this application.

To do this the C code is:

Delay100TCYx(78);

Note the number of times the subroutine is executed is written in the brackets (78) in this case. NB. 255 is the maximum value that can be entered.

Loops

In order to make the program execute some code a number of times or indefinitely, we use a loop.

The WHILE LOOP as it is called looks like this:

```
while ( )
{
}
```

The code to be executed is written between the brackets { } while the condition for executing the code is written between the brackets ( )

Suppose we wish to turn an alarm on if the temperature goes above 60°C, the code would look like:

```
while (Temperature>60)
{
PORTBbits.RB0 = 1;    // turn on PORTB bit0
}
```

If we wish to execute a piece of code indefinitely such as flashing our LED on PORTB bit 4 on and off continuously, the loop is:

```
while (1)
{
PORTBbits.RB4 = 1;                  // turn on PORTB bit4
```

```
Delay100TCYx(78);          // wait 1 s
PORTBbits.RB4 = 0;         // turn off PORTB bit4
Delay100TCYx(78);          // wait 1 s
}
```

The function while (1) means while 1 is true! But 1 is always 1, so the loop is always executing. Note the function while (1) does not end with a ;

The // code means ignore what follows on the line; it is for our reference not for the compiler.

A useful while loop involves just 1 line of code, i.e.,

while (PORTBbits.RB4==1);

This means continue the loop (just 1 line) until PORTBbits.RB4==1 is no longer true, i.e., wait until PORTBbits.RB4==0 before moving on in the program.

Note: == means is equal to,=means equals. One is a question and the other a statement.

Entering Numbers in the Program

Numbers can be entered in the program as hexadecimal, binary or decimal.

A hexadecimal number 7F is written as 0 × 7F.

A binary number 1111 0101 is written as 0b11110101.

A decimal number 45 is just written as 45.

We are now ready to write our C program.

NOTE

In the remainder of this book we will be writing C files and putting them into projects. You can choose to do this with **MPLAB** or **MPLABX**.

WRITING THE CODE USING MPLAB IDE

Run MPLAB and the screen shown in Figure 2.2 will open.

For our first program we are going to flash an LED on and off at 1 s intervals on the output pin, PORTB,4. We will call this, flash.c

Under the file menu select New (or if you have the code Open flash.c) (Figure 2.3).

The screen shown in Figure 2.4 showing the file editor will open.

If the line numbers are not visible on the left hand side turn them on with Edit/Properties as shown in Figure 2.5.

Select File Type/Line Numbers.

Click Apply then.

Click OK as shown in Figure 2.6.

We need to open a project to store our code in. The project will contain our code flash.c and also when we have compiled it a new file, flash.hex which is the machine code file which will be blown into our microcontroller with the

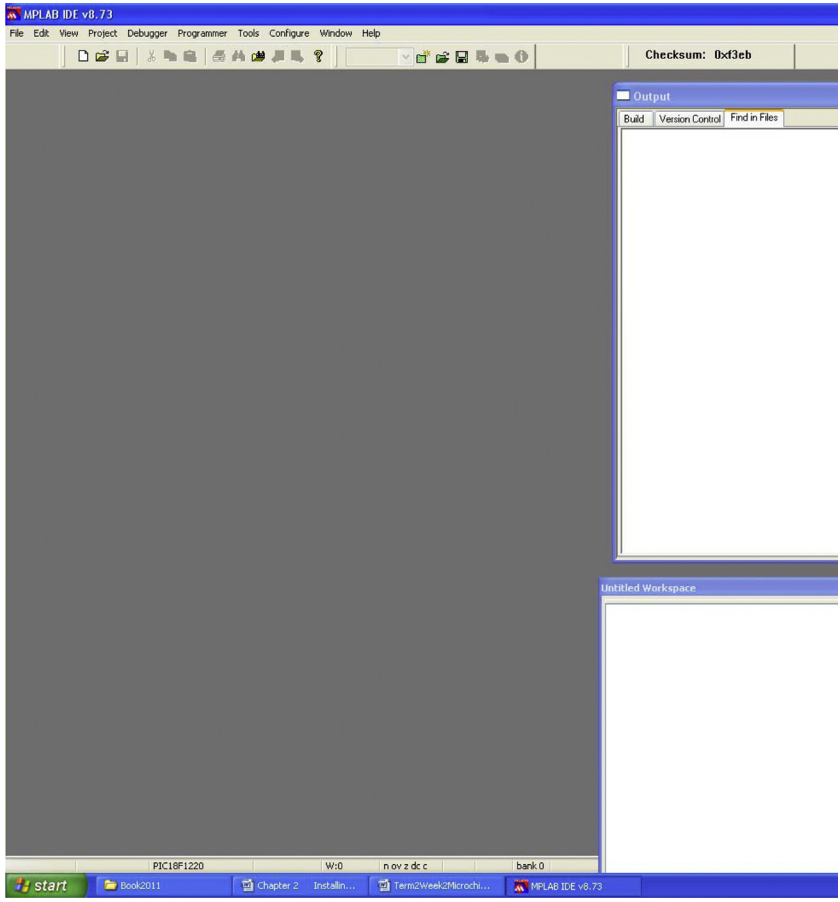

FIGURE 2.2 MPLAB workspace.

aid of a programmer. The project also contains the workspace which is the user screen that would show the registers and the watch window. We will use these later when we look at the simulator.

To make the project, select Project/Project Wizard as shown in Figure 2.7.

Click Next on the dialogue box of Figure 2.8.

Step 1: select the microcontroller you require, the PIC18F1220, from the Project Wizard as shown in Figure 2.9.

Step 2: select the language toolsuite to use, i.e., the C18 compiler.

If not already selected choose Microchip C18 Toolsuite as your Active Toolsuite as shown in Figure 2.10.

Then click, Next.

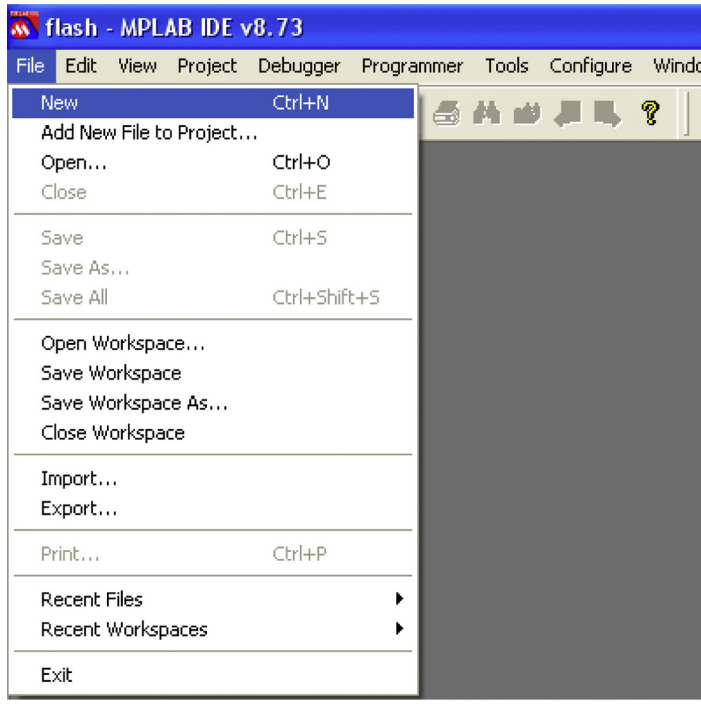

FIGURE 2.3 Creating the file flash.c.

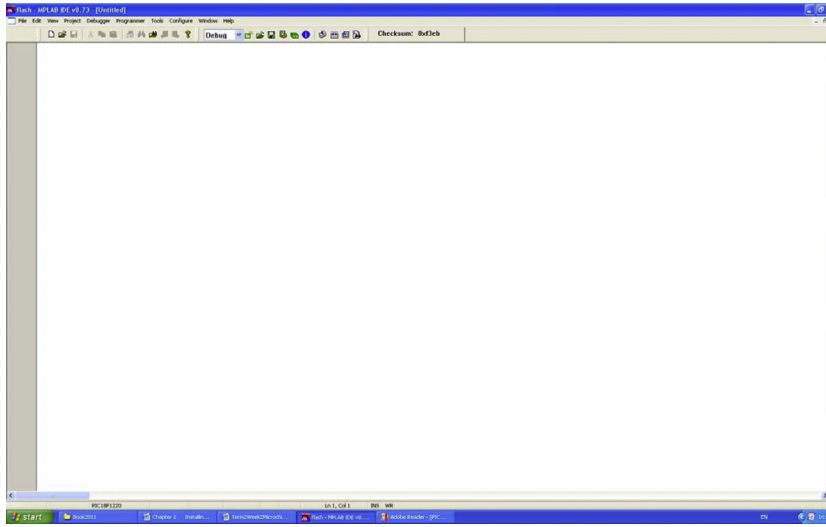

FIGURE 2.4 File editor screen.

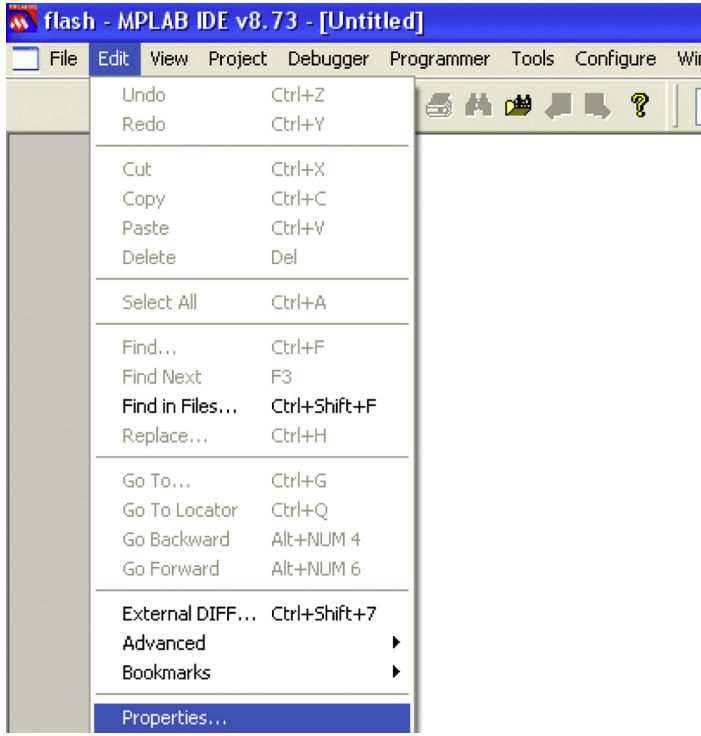

FIGURE 2.5 Editor properties.

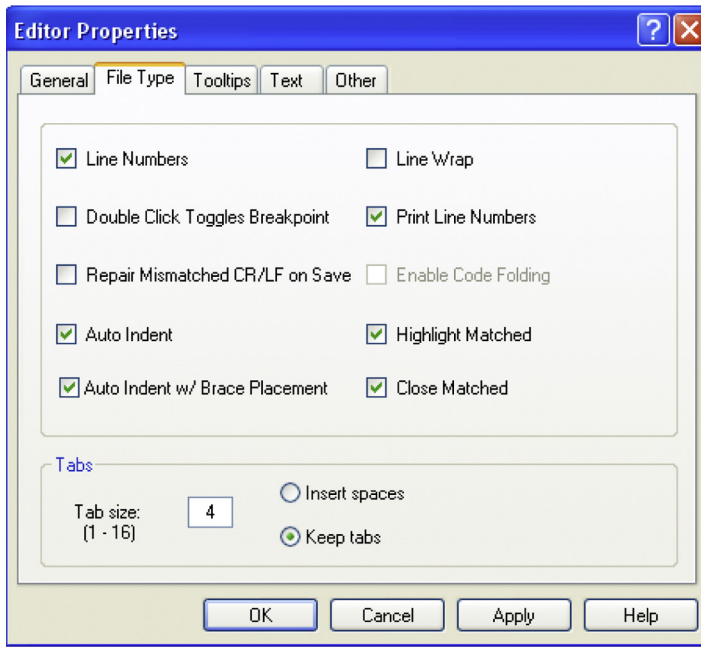

FIGURE 2.6 Turning on line numbers.

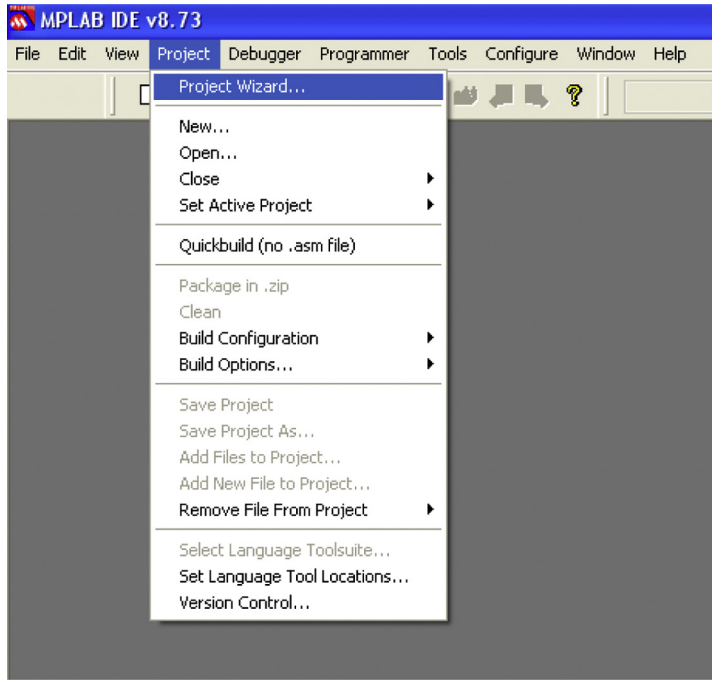

FIGURE 2.7 Selecting the Project Wizard.

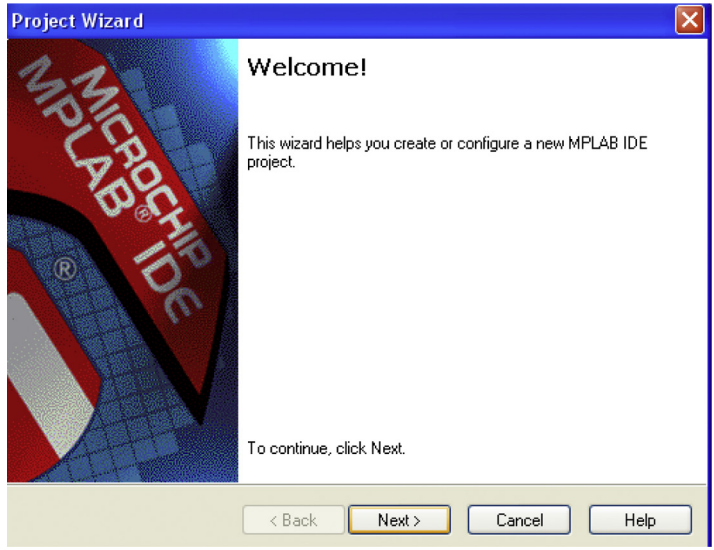

FIGURE 2.8 Project Wizard dialogue box.

FIGURE 2.9 Selecting the device.

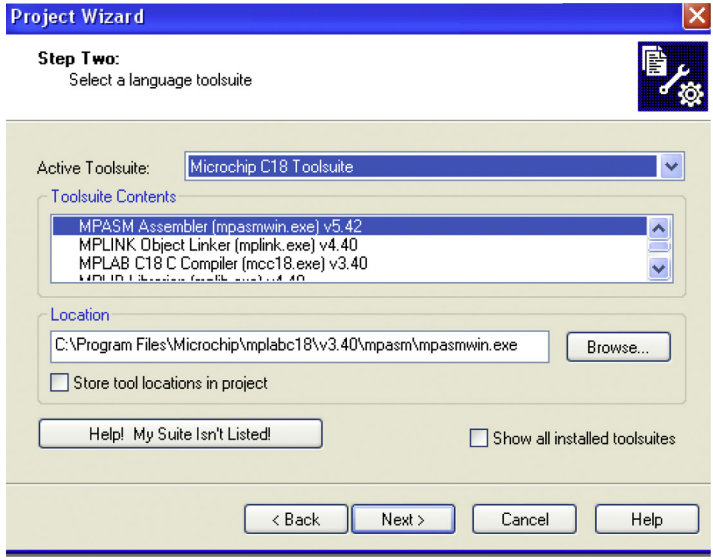

FIGURE 2.10 Selecting the Microchip C18 Toolsuite.

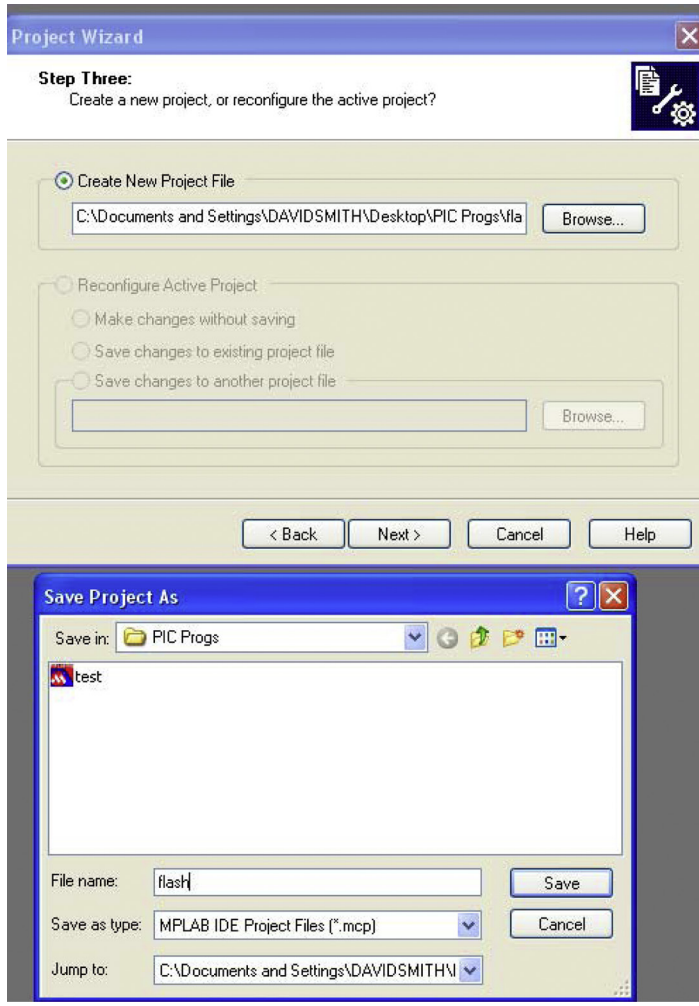

FIGURE 2.11 Creating a project file.

Step 3: create the project, browse for the directory you have created, i.e., PICProgs, and create a new project file, flash, or open it if it is already created, as shown in Figure 2.11.

Click Save, then.

Click Next.

Step 4: add existing files to your project (Figure 2.12).

At the moment we do not have any existing files. If you have a copy of flash.c you can add it now. If not just click Next.

That's the project made.

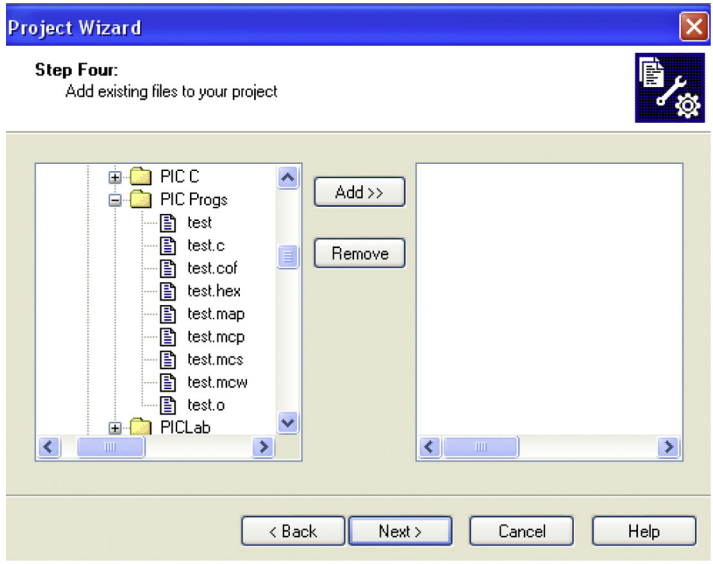

FIGURE 2.12 Adding a file to your project.

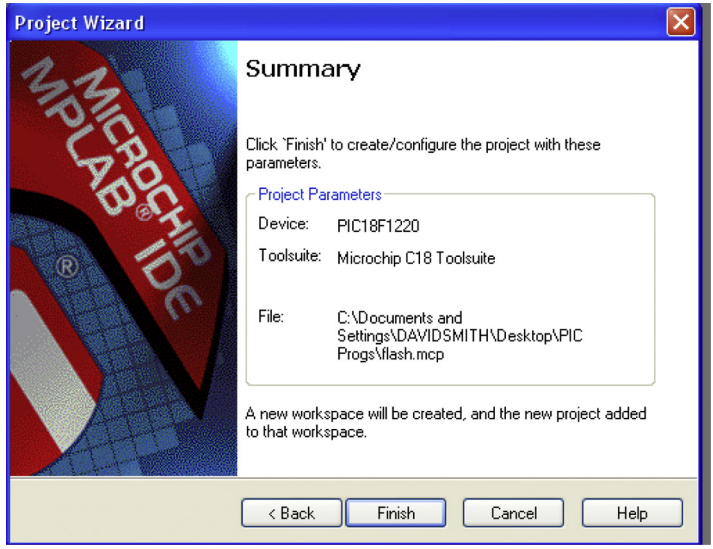

FIGURE 2.13 Finish making the project.

Click Finish (Figure 2.13) to finish making the project and return to the workspace.

We now have a project called flash saved in the PICProgs folder. Now we can write our C program called flash.c.

We have already discussed the steps involved in the program.

Flashing an LED On and Off

The LED is connected to PORTB pin 4 and is to be flashed on and off every second. The circuit for this is shown in Figure 1.2.

The code is:

```
#include <p18f1220.h>
#pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE = OFF
#include <delays.h>

void main (void)
{
        //SET UP
        // OSCCON defaults to 31 kHz. So no need to alter it.
        ADCON1 = 0×7 F;          //all IO are digital or 0b01111111 in binary
        TRISA = 0b11111111;      //sets PORTA as all inputs
        PORTA = 0b00000000;      //turns off PORTA outputs, not required, no outputs
        TRISB = 0b00000000;      //sets PORTB as all outputs
        PORTB = 0b00000000;      //turns off PORTB outputs, good start position
        while (1)
        {
        PORTBbits.RB4 = 1;       // turn on PORTB bit4
        Delay100TCYx(78);        // wait 1 s
        PORTBbits.RB4 = 0;       // turn off PORTB bit4
        Delay100TCYx(78);        // wait 1 s
        }
}
```

If you do not have this file saved click File New as shown in Figure 2.14 and copy/paste it into the untitled space and then save it in your folder as flash.c.

If you already have the file saved then select Open—PicProg/flash.c (Figure 2.14).

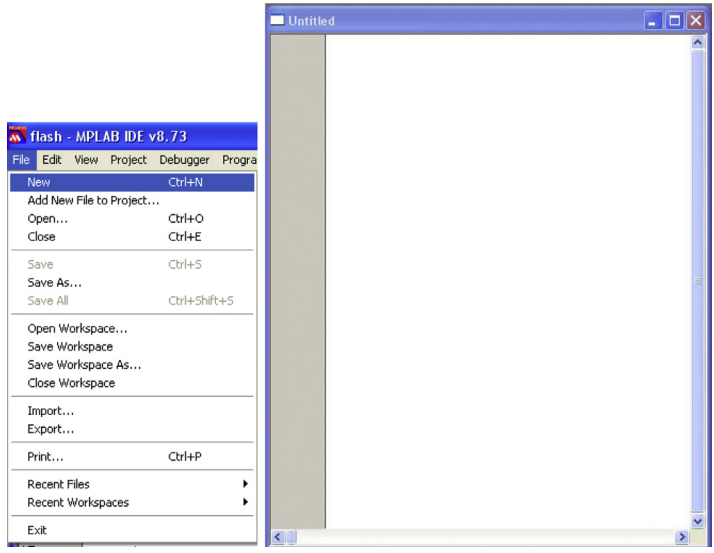

FIGURE 2.14 Creating a new file.

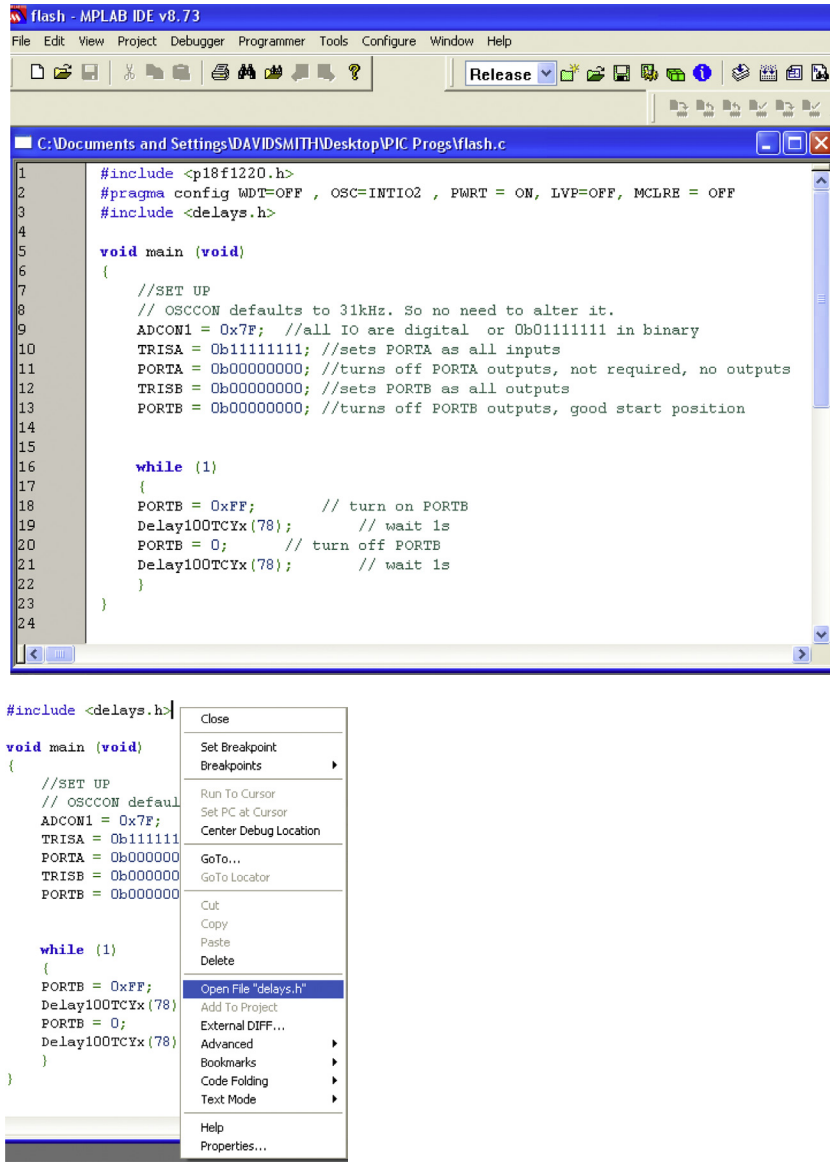

FIGURE 2.15 Flash.c code entered in MPLAB.

Your screen should look like Figure 2.15 with the code entered.

Explanation of the Code

- Line 1. include, adds the file p18f1220.h to your code. This file has the names of all the registers and their locations, i.e., ADCON1, TRISA,

PORTA, TRISB, and PORTB. Make sure you use the correct register and pin names in your code. You can view p18f1220.h to see what the registers and pins are called.

- Line 2. Pragma describes a task which is performed before the code is run. Here in the configuration register we are turning the watch dog timer (WDT) off. The WDT is a timer which will cause the program to reset if the WDT is not reset. We will not be using this, so do not worry about it, but turn it off. The oscillator is configured to run internally with OSC = INT02. The power up timer, PWRT, is turned on by PWRT = ON. This waits 20 ms after the power is applied before running the program, to give the voltages time to settle. The low voltage programming, LVP, is turned off, LVP = OFF because it conflicts with RB5 and the Master Clear is disabled, MCLRE = OFF because a low on RA5 would clear the program. We will be adding other pragmas in future programs.

  If any of these pragma configs need changing then the codes can be found in the MPLAB HELP menu under Topics/ PIC18 Config Settings.
- Line 3. We are adding the delays.h file which of course has the delay code as described earlier.

  The header files can be viewed by right clicking on the name in the program. To view the delays.h file right click on the #include line and Open File "delays.h" as shown in Figure 2.15.
- Line 4. It is blank, blank lines make the program easier to read.
- Line 5. Void main (void) is a function or method in C. Void means the function does not return anything. A function Sum (answer) would return the answer to an addition. The main function is the point at which the C code starts running. The main function executes the code between the brackets { } on lines 6 and 23. Of course more about functions later.
- Line 7. It is a comment explaining that the following code paragraph provides the initial configuration for our micro.
- Line 8. Another comment explaining the register OSCCON, oscillator control register, defaults to 31 kHz. This is what we require for this program.
- Line 9. The 18f1220 has seven of its pins that can be used as analogue inputs when performing an A–D conversion. If we are not using the A–D then we need to configure the inputs as digital. The default is they are analogue. A 1 in ADCON1, Analogue to digital conversion register 1, sets the input as digital, a 0 sets the input as analogue.
- Line 10. Sets up the PORTA pins as inputs, a 1 in a TRISA bit sets the corresponding pin as an input, a 0 sets the pin as an output.
- Line 11. Sets the PORTA output pins to 0. Not required since the pins are all inputs.
- Line 12. Configures PORTB pins as all outputs.
- Line 13. Sets all PORTB outputs to 0. A good starting point, all outputs off.
- Line 16. Calls the 'while function' which runs continually. The code executed by the while function is between the brackets { } line 17 and line 22.
- Line 18. Turns PORTB bit 4 on.

- Line 19 and 21. Call the Delay100TCYx function 78 times, (78). Delay100TCYx is 100 clock cycles as mentioned previously, giving a total time of 0.998 s.
- Line 20. Turns PORTB bit 4 off.

Lines 1–13 can be used as a header for all of your programs, modify it to suit your application.

Add the File to Your Project

If the file hasn't been added to your project then do it now with File—Add New File to Project (Figure 2.16).

Compiling the C code

Now that the flash.c file has been added to the project we are ready to convert it to a format that we can program into the microcontroller.

All we need to do is click on the Build All icon as shown in Figure 2.17.

The Output box shown in Figure 2.18 will be shown and should have BUILD SUCCEEDED at the bottom. If not the errors will be indicated in the output box.

Fixing Errors

In my code, line 18, I have introduced a syntax error. I have changed PORTBbits.RB4 to PORTBbits.4

The error is shown in the output box (Figure 2.19) which ends with the statement in red, BUILD FAILED.

Clicking on the error in the output box will highlight the error in the code as indicated.

Fix all of the errors and save your changed code and Build All again. You should then see BUILD SUCCEEDED in the output box. Once the build has been successful you have automatically created the HEX file to program in the microcontroller—in this case flash.hex.

TYPES OF ERRORS

When your program compiles the errors indicated are syntax errors. The compiler will not tell you if your program is going to work correctly.

The types of mistakes you could make are:

- Not spelling the Register name correctly.
- Not spelling the Bit name correctly.
- Adding a ; when it is not required.
- Missing a ; when it is required.
- Misspelling a C command.

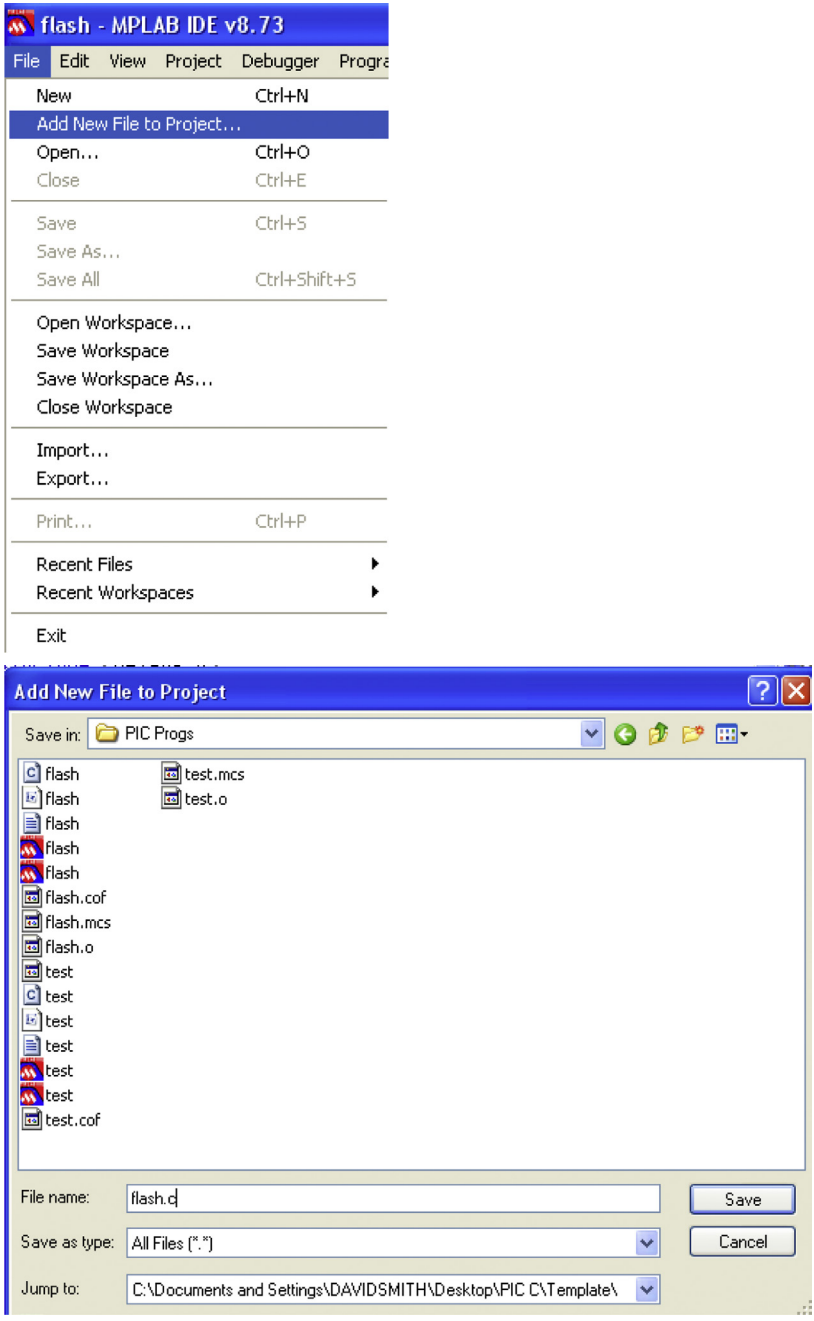

FIGURE 2.16 Adding a New File to Project.

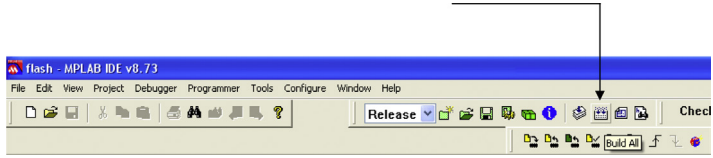

FIGURE 2.17 Build All.

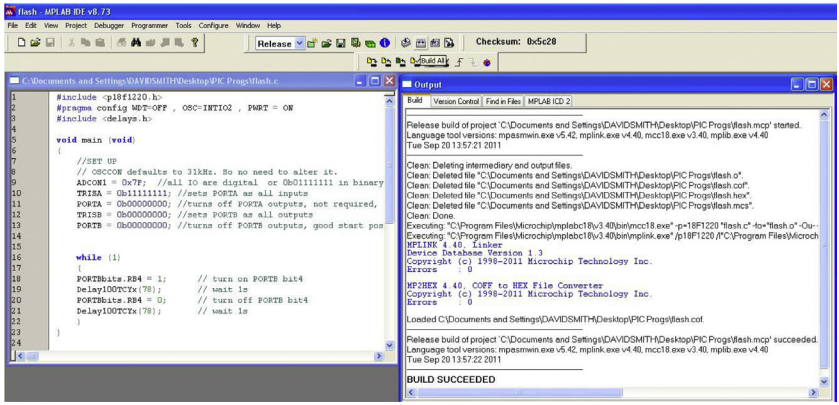

FIGURE 2.18 Compiling the C code.

PROGRAMMING THE MICROCONTROLLER

Once the code has been compiled (built) correctly you can then program the hex code into the microcontroller.

There are many programmers on the market, they will require the hex file and to follow the manufacturer's instructions. In this book I am going to outline the use of Microchip's own programmer the ICD3 shown in Figure 2.20A and the PICkit3 shown in Figure 2.20B. They can program and debug the microcontroller.

Connect the ICD3 or the PICkit3 and the chip holder to the PC according to the manufacturer's instructions.

Click "Debugger—Select Tool—None" as shown in Figure 2.21.

Click "Programmer—Select Programmer—MPLAB ICD3" as shown in Figure 2.22 (or PICkit3).

And finally to program the microcontroller, click the Program target device icon  as shown in Figure 2.23.

The output box should then indicate "Programming succeeded" as indicated in Figure 2.24.

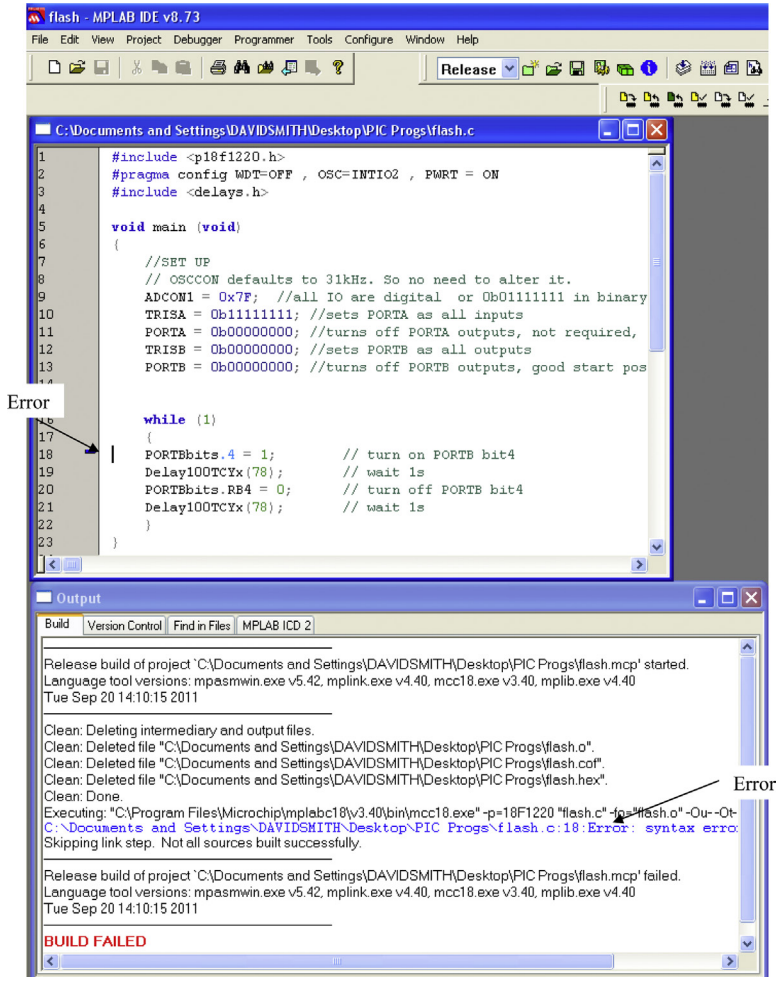

FIGURE 2.19 Indicating errors.

If the programming is not successful check:

- You are using the correct microcontroller.
- The correct microcontroller has been selected on MPLAB.
- The microcontroller is the correct way round in the socket.
- The microcontroller is a good one.

Connect the microcontroller to your circuit and observe the LED flashing.

Now that we have written our first program let us consider switching all eight outputs on PORTB.

FIGURE 2.20 (A) The Microchip ICD3 and (B) The PICkit3.

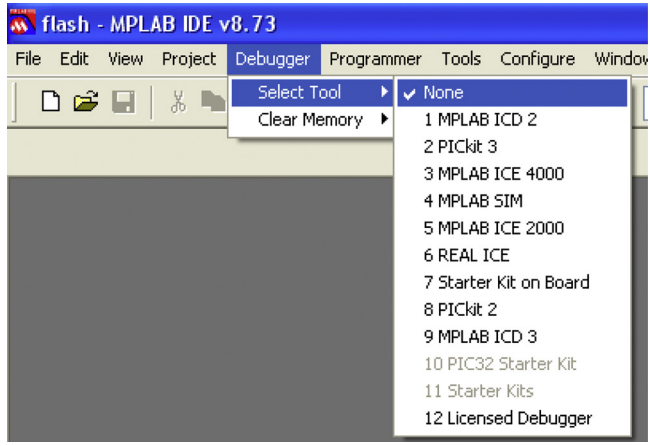

FIGURE 2.21 Switching off the Debugger.

NOTE

In the remainder of this book we will be writing C files and putting them into projects. You can choose to do this with MPLAB or MPLABX.

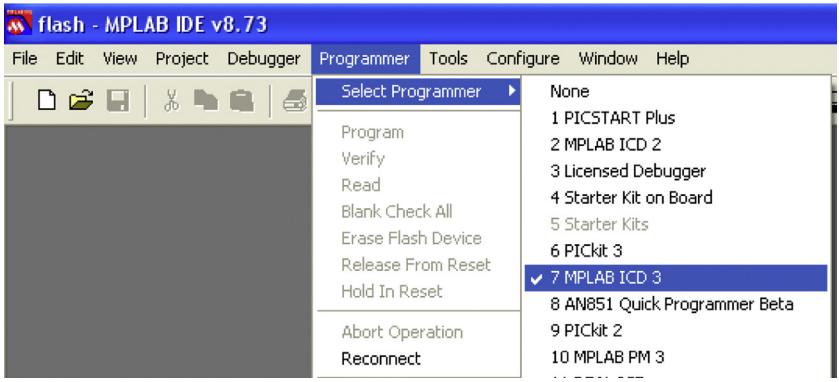

FIGURE 2.22 Selecting the programmer.

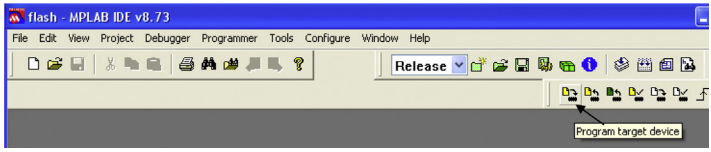

FIGURE 2.23 Programming the target device.

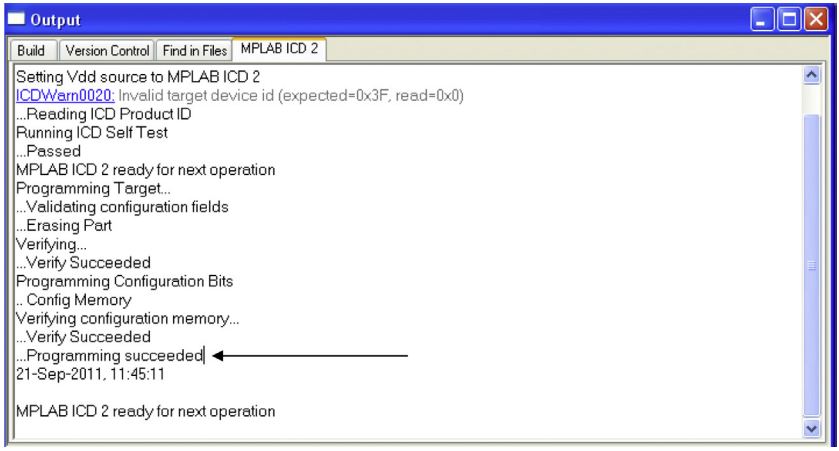

FIGURE 2.24 Output box showing a successful programming of the Microcontroller.

WRITING THE CODE USING MPLABX

MPLAB IDE reached version 8 and has recently had a makeover rather than become version 9; Microchip have called it MPLABX. Version 1.30 is used here.

Install MPLABX IDE and the C compiler XC8 available from Microchip.com.

While following this section please also refer to MPLABX IDE User Guide available from Microchip.

In order to program our code into the microcontroller we will need to build a project.

Every embedded project contains the following items.

1. Group of files to build a final embedded image.
2. Device the resultant image will be built for, e.g., 18F1220.
3. Compiler to use to build the image; we will be using the XC8.
4. Hardware Tool to be used for debugging/programming the target device.

CREATING A NEW PROJECT

Create a new project in one of two ways:

Select "Create New Project" from the "Learn & Discover" tab of the MPLAB X Start Page (Figure 2.25)

or

Select *File>New Project* from the menu bar (Figure 2.26)

1. A project wizard will open to help walk you through the creation of your project. Select "Microchip Embedded" from **Categories** and "Standalone Project" from **Projects**. Click **Next** (Figure 2.27).

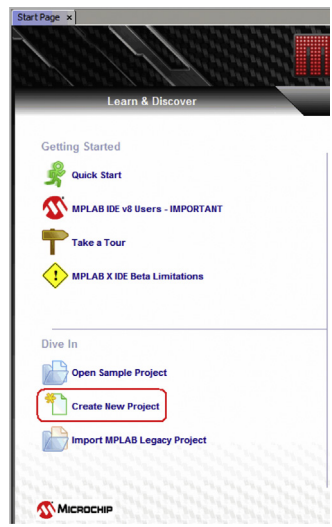

FIGURE 2.25 Creating a new project.

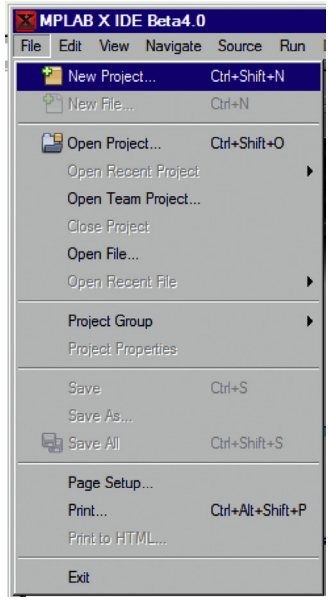

FIGURE 2.26 Selecting a new project.

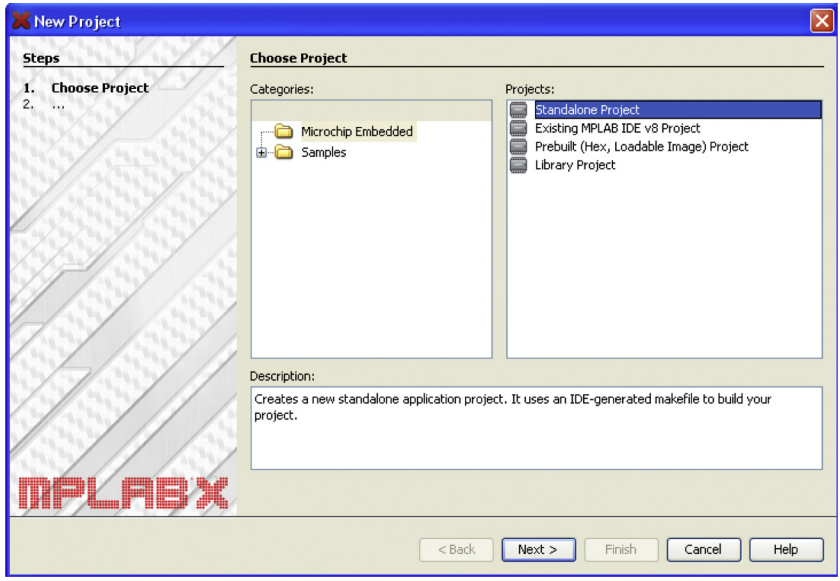

FIGURE 2.27 Choosing the project.

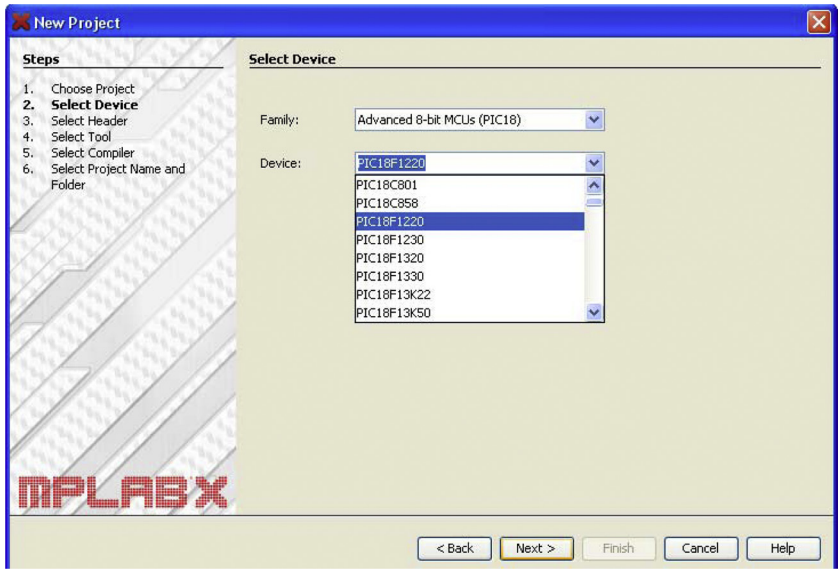

FIGURE 2.28 Selecting the device.

2. Select the device you want to use in your project, PIC18F1220. Click **Next** (Figure 2.28).
3. Select the Hardware Tool you want to use to either debug your application or program your target device, e.g., PICkit3. Click **Next** (Figure 2.29).
4. Select the compiler toolchain (toolsuite) you want to use to build the output image of your project, e.g., XC8, Click **Next** (Figure 2.30).
5. Provide a project name and directory where you want the project to reside. Click **Finish**. I have used the Flash program to produce the project (Figure 2.31).

Your project should now be visible in the "Project" window (Figure 2.32).

You have now created a new embedded project. The next step is to add your code (C file) to the project.

Adding Source Code to the Project

Now that you have created the project you can add existing or create new source files within the project. We will add the program flash.C discussed in the MPLAB section (Figure 2.15).

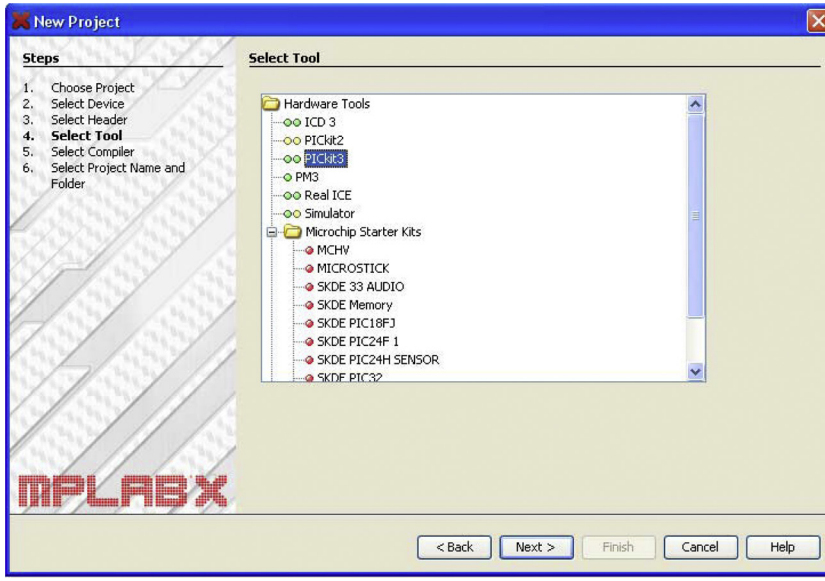

FIGURE 2.29 Selecting the programming tool.

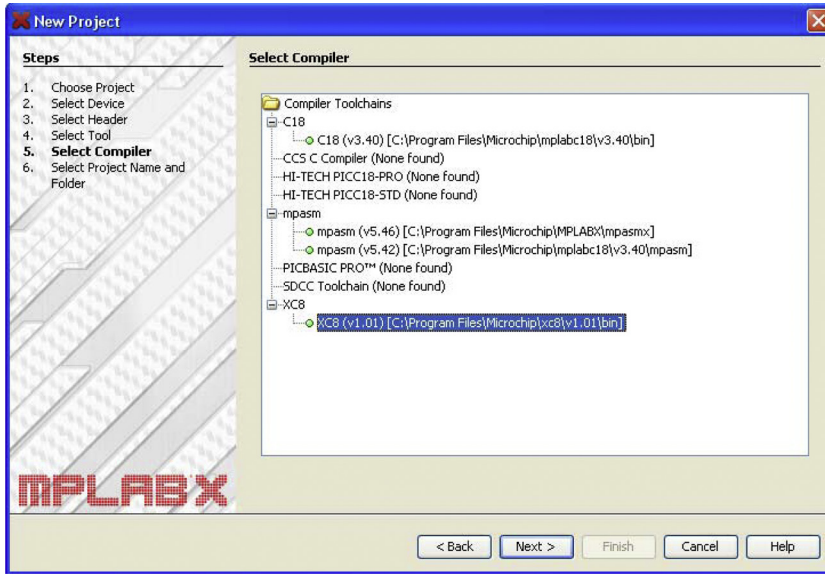

FIGURE 2.30 Selecting the XC8 compiler.

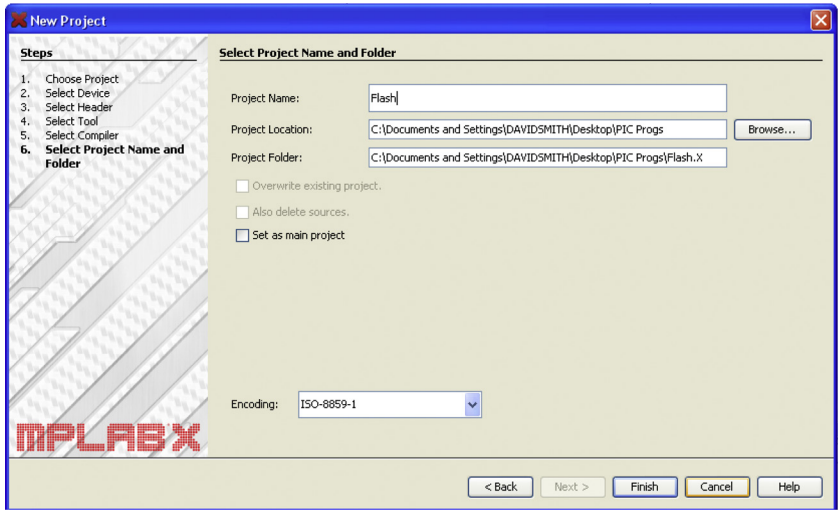

FIGURE 2.31 Selecting the project name and directory.

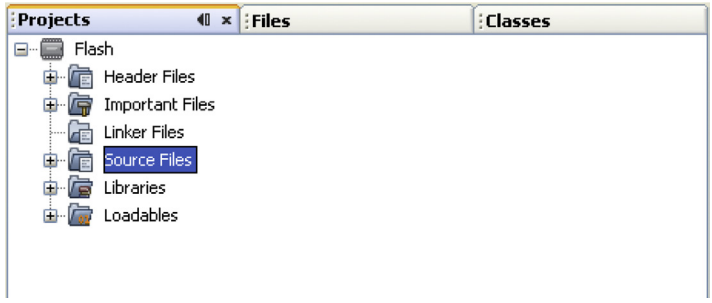

FIGURE 2.32 The project window.

Select *File>New File* to launch a File wizard (Figure 2.33).

1. The project to which the new file will be added is listed in the Project field. Select the file type (C and C Source File) and click **Next** (Figure 2.34).
2. Name the file and the location at which to store the file. Click **Finish** (Figure 2.35).

A new file will be created, opened in the editor, and placed into the project identified. Drag and drop the file to place it in the Source Files virtual folder (if needed) (Figure 2.36).

You can also add existing source files or create new files from the right click (context) menu (Figure 2.37).

Double click on the source file, in this example flash.C, and the file will open in the window as shown in Figure 2.38.

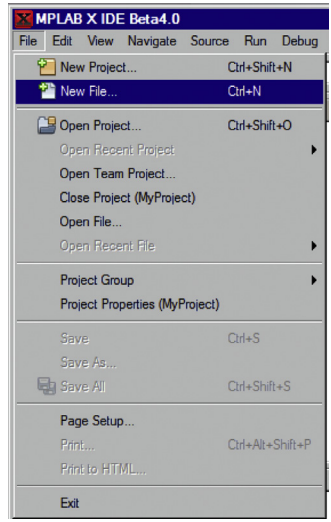

FIGURE 2.33 Creating a new file.

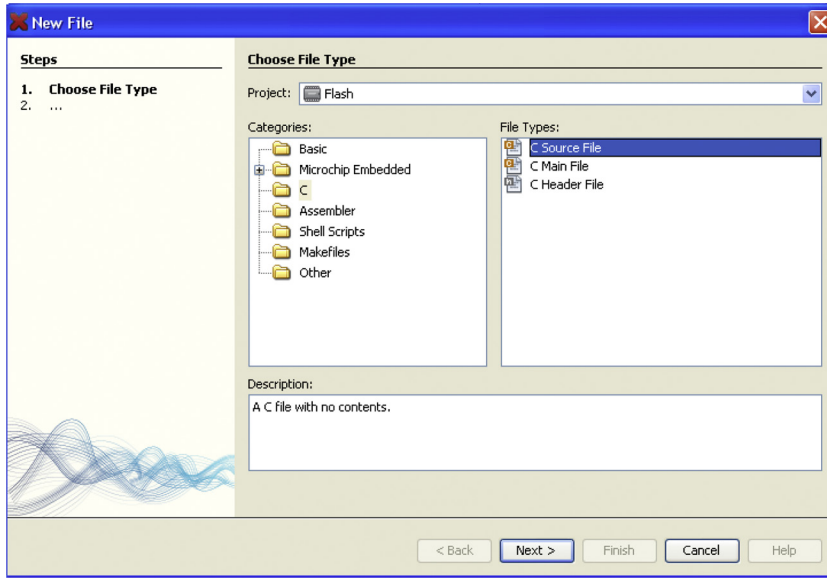

FIGURE 2.34 Selecting the file type.

The line numbers can be turned on and off by right clicking the line numbers column.

The project is now ready to build, i.e., to convert the C code to hex code for the chip.

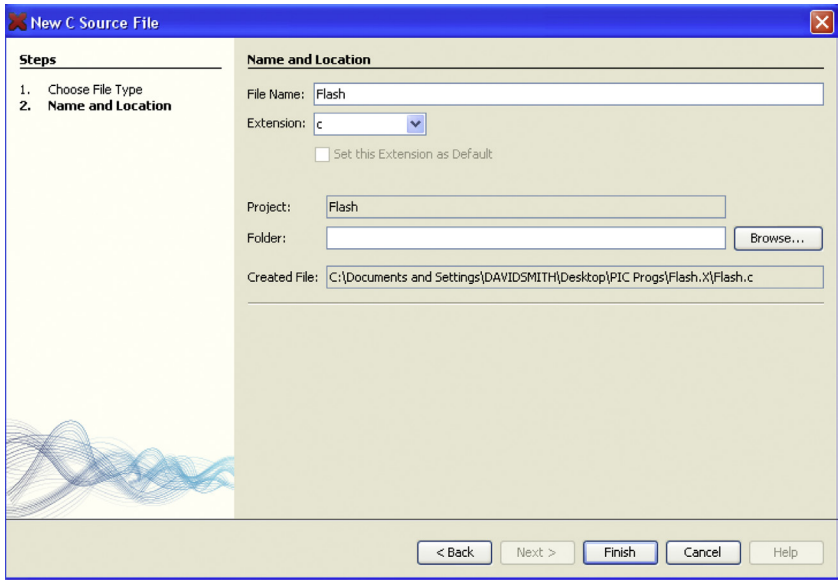

FIGURE 2.35 Locating the file in a folder.

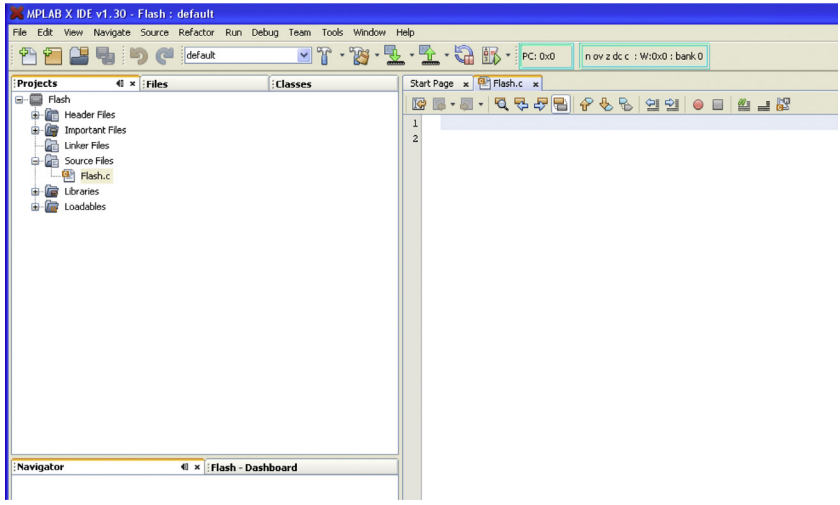

FIGURE 2.36 Adding the file to a folder.

Click the build program icon, the hammer, shown. 

The output should show "Build Successful" (Figure 2.39).

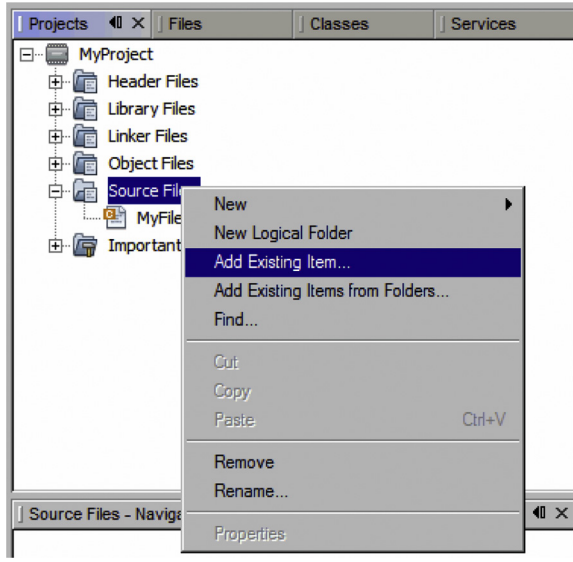

FIGURE 2.37 Adding a file to a project.

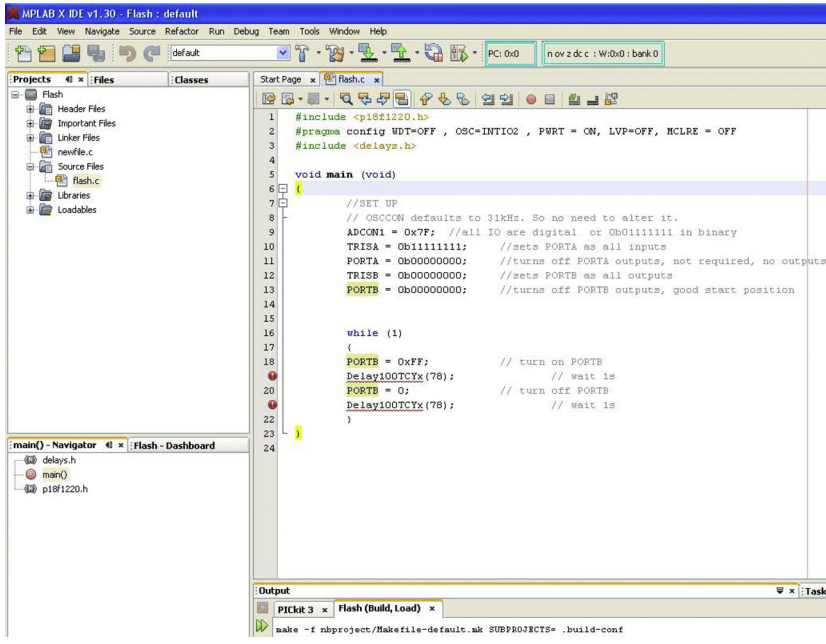

FIGURE 2.38 Opening a source file in the project window.

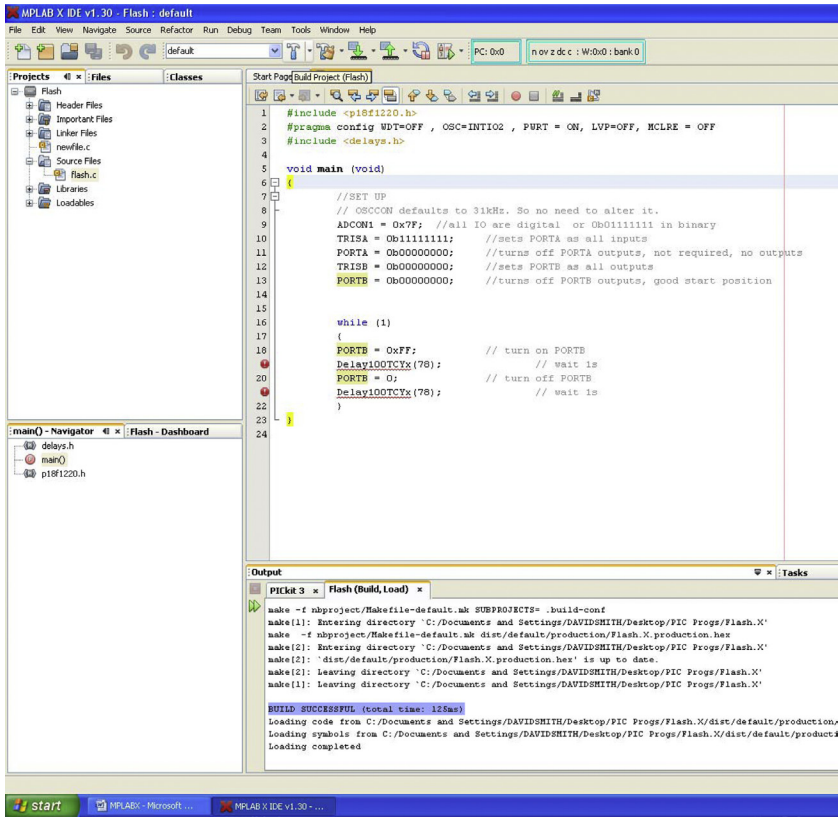

FIGURE 2.39 Building the project.

FIGURE 2.40 Programming the code into the Microcontroller.

The program should have built successfully. If not double click the error statement and correct the errors.

The project hex file is now ready to be programmed into the chip.

Connect the PICkit3 (or any other programmer) to the chip following the manufacturer's instructions. Click the program icon as shown in Figure 2.40.

The output box (Figure 2.41) should indicate that the Programming has been verified and is complete.

Connect the microcontroller to your circuit and observe the LED flashing.

Now that we have written our first program let us consider switching all 8 outputs on PORTB.

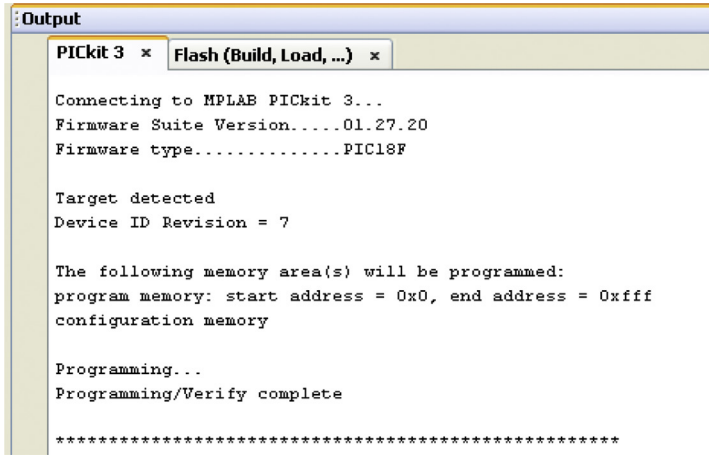

FIGURE 2.41 Indicating programming has been completed successfully.