CHAPTER 3

MPLAB X IDE and MPLAB XC8 C Programming Language

Chapter Outline

In this chapter, we shall be looking at details of the other popular PIC C programming language/compiler, the MPLAB XC8, and the MPLAB X IDE (Integrated Development Environment), both developed by Microchip Inc.

This chapter is organized as a tutorial so that the user can quickly become familiar and start using the compiler and the IDE. The popular PICDEM PIC18 Explorer Development Board is used in the examples given in this chapter.

Before going into the details of the MPLAB XC8 programming language and the MPLAB X IDE, it is worthwhile to look at the features of the PICDEM Explorer board.

It is recommended that if you are not familiar with programming using the C language, then you should read Chapter 2 before continuing with this chapter.

PIC Microcontroller Projects in C. http://dx.doi.org/10.1016/B978-0-08-099924-1.00003-4

3.1 The PICDEM PIC18 Explorer Development Board

The PICDEM PIC18 Explorer development board (called the Explorer board from now on) is a low-cost development board for PIC18 family of microcontrollers. The board is fitted with a PIC18F8722-type microcontroller.

Figure 3.1 shows the Explorer board where each feature is indicated with a number. The board provides the following features:

1. PIC18F8722 microcontroller,
2. PIM header to connect an alternate PIC18 microcontroller,
3. In-circuit Debugger connector,
4. PICkit 3 programmer/debugger connector,

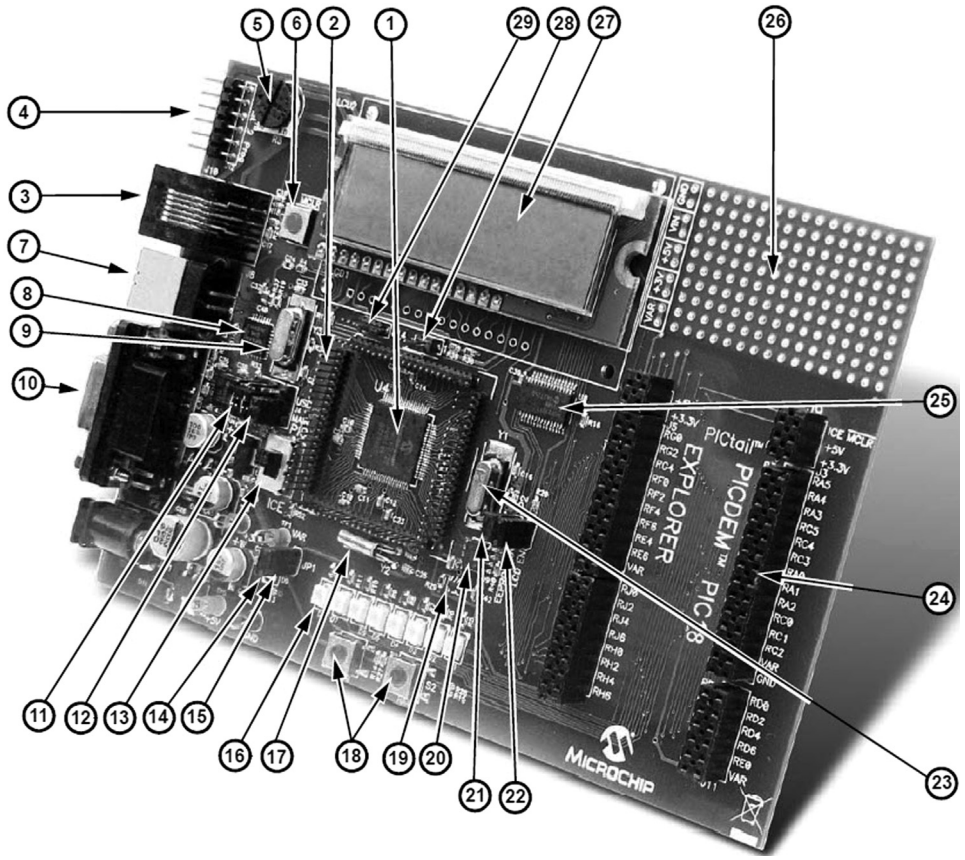

Figure 3.1: PICDEM PIC18 Explorer Board.

5. Potentiometer for analog input,
6. Reset switch,
7. RS232-universal serial bus (USB) connector,
8. PIC18LF2450 microcontroller (for converting an RS232 to a USB),
9. Crystal for the PIC18LF2450 (12 MHz),
10. Nine-pin RS232 connector,
11. Jumper J13 for routing RS232 serial communication to the USB port or the RS232 socket,
12. Jumper J4 for programming the main microcontroller or the PIC18LF2450,
13. Switch S4 for selecting the main microcontroller as either the mounted PIC18F8722 or a PIM module,
14. Power indication light emitting diode (LED),
15. JP1 for disconnecting the eight LEDs,
16. LEDs,
17. A 32,768-kHz crystal (for Timer 1),
18. Push-button switches S1 and S2,
19. MPC9701A analog temperature sensor,
20. 25LC256 electrically erasable programmable read-only memory (EEPROM),
21. Jumper JP2 to enable/disable EEPROM,
22. Jumper JP3 to enable/disable an LCD,
23. Crystal for the main microcontroller (10 MHz),
24. PICtail daughter board connector,
25. I/O expander for the LCD,
26. User prototype area,
27. LCD,
28. Jumper J2 for selecting between 3.3 and 5 V (by default the V_{DD} voltage is +5 V),
29. Jumper J14 for use with a PIM.

To use the Explorer board with the on-board PIC18F8722 microcontroller, the following switches and jumpers should be configured:

- Set Switch S4 to PIC to select the on-board microcontroller,
- Enable LEDs by connecting a jumper at JP1,
- Enable the LCD by connecting a jumper at JP3,
- Connect Jumper J4 to MAIN to program the main microcontroller (PIC18F8722).

The Explorer can be programmed by using several hardware tools, such as the PICkit 2/3, ICD 2/3, and Real In Circuit Emulator (ICE). In this chapter, we shall be seeing how to program and debug a program using the PICkit 3 and the ICD 3 programmer/debugger tools.

3.1.1 Module Connections on the Explorer Board

The various modules on the Explorer board are connected as follows:

- Eight LEDs are connected to PORTD of the microcontroller (the LEDs can be disconnected by removing Jumper JP1).
- The LCD is controlled by port pins RC3, RC4, and RC5.
- Switch S1 is connected to port pin RB0 (active LOW).
- Switch S2 is connected to port pin RA5 (active LOW).
- Switch S3 is connected to the Master Clear (MCLR) reset input.
- Switch S4 is an MCLR select switch.
- The potentiometer is connected to the AN0 input through a resistor, and it can be adjusted from 0 V to V_{DD}.
- The analog temperature sensor MCP9701A is connected to port pin RA1.
- The RS232-USB port is connected to pins RC6 and RC7.

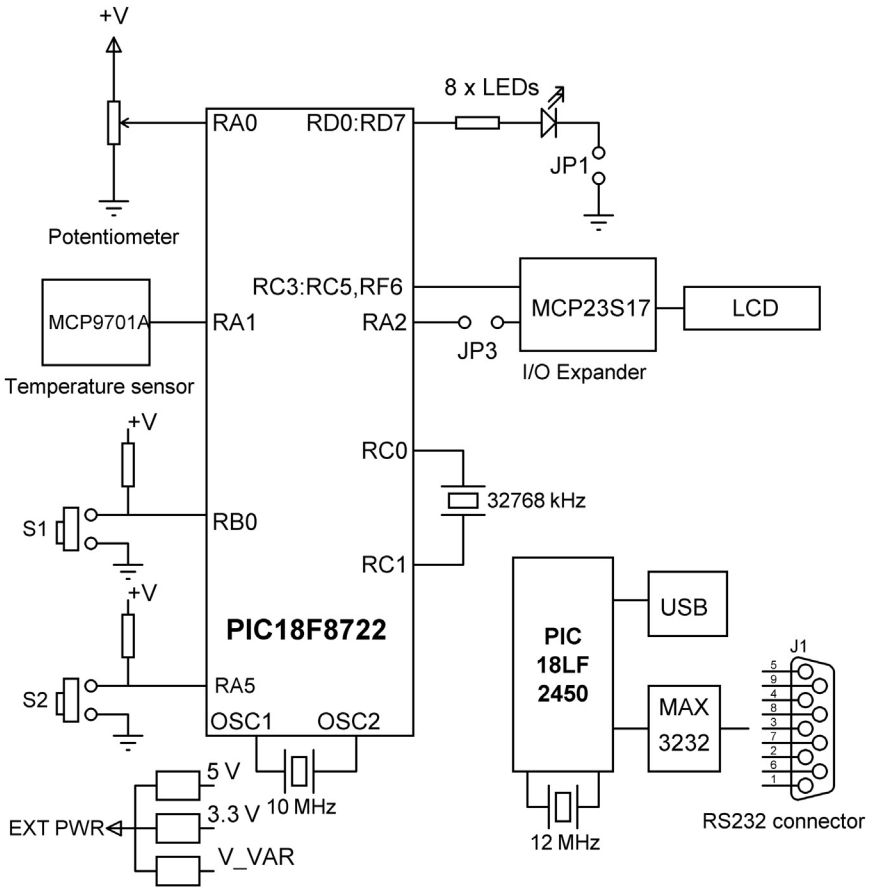

Figure 3.2: The Simplified Block Diagram of the Explorer Board.

A simplified block diagram showing the module connections on the Explorer board is shown in Figure 3.2. Note that the various chips on the board are fed with the adjustable voltage source V_VAR, which is set to +5 V by default. The board is powered from an external 9 V DC mains adapter capable of providing up to 1.3 A.

3.1.2 Using the PICkit 3 to Program/Debug

A PICkit 3 programmer/debugger can be attached to the 6-pin connector mounted at the top-left corner of the Explorer board for programming and debugging user programs.

3.2 MPLAB X IDE

The MPLAB X IDE is the integrated development environment (just like the mikroC Pro for PIC IDE) that enables the user to create a source file, edit, compile, simulate, debug, and send the generated code to the target microcontroller with the help of a compatible programmer/debugger device. The Microchip Inc. web site (http://www.microchip.com/pagehandler/en-us/family/mplabx/) contains detailed information on the features and the use of the MPLAB X IDE. In this chapter, we will be looking at the program development steps using this IDE.

MPLAB X IDE is available at the Microchip Inc. web site (http://www.microchip.com/pagehandler/en-us/family/mplabx/#downloads), and it must be installed on your PC before it can be used. At the time of writing this book, the latest version of the MPLAB X IDE was v1.85.

3.3 MPLAB XC8 Compiler

The MPLAB XC8 compiler is a powerful C compiler developed for the PIC10/12/16/18 family of microcontrollers (there are also versions for the 24- and 32-bit PIC microcontrollers). The MPLAB XC8 compiler has three versions: Pro, Standard, and Free. In this book, we will be using the Free version. The main difference between the different versions is the level of optimization applied during the compilation.

The XC8 compiler must be installed after installing the MPLAB X IDE. The compiler can be installed during the last stage of installation of the MPLAB X IDE. Alternatively, it can be installed from the Microchip Inc. web site (http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/). At the time of writing this book, the latest version of the compiler was v1.20.

The XC8 language has many similarities to the mikroC Pro for PIC language. In this chapter, we will be looking at the steps of developing a simple XC8-based project. The similarities and differences between the two languages will also be explained.

Example 3.1—A Simple Project

A simple project is given in this section to show the steps in creating a source file using the MPLAB X IDE, compiling the file, and downloading the generated hex file to the PIC18F8722 microcontroller on the Explorer board using the PICkit 3.

In this project, we will be using a push-button switch S1 and the LED connected to port pin RD0. The program will turn the LED on whenever the button is pressed.

Solution 3.1

The steps are given below.

Step 1. Double click the icon to start the MPLAB X IDE. You should see the opening window as in Figure 3.3.

Step 2. Move the right-hand cursor down and click on icon *Create New Project.* Select the default (Category: *Microchip Embedded*, Projects: *Standalone Project*) as in Figure 3.4 since we are creating a new standalone project.

Step 3. Click Next. Select the target microcontroller. Family: *Advanced 8-bit MCUs (PIC18)*, and Device: *PIC18F8722* as in Figure 3.5.

Step 4. Click Next and select Hardware Tools: *PICkit 3* as in Figure 3.6.

Step 5. Click Next. Select compiler XC8 as in Figure 3.7.

Step 6. Click Next. Give your project a name. In this example, the project is given the name *BUTTON-LED* and is stored in the folder *C:\Users\Dogan\MPLABXProjects.* Click *Set as main project* option as shown in Figure 3.8.

Step 7. Click Finish to create the required project files.

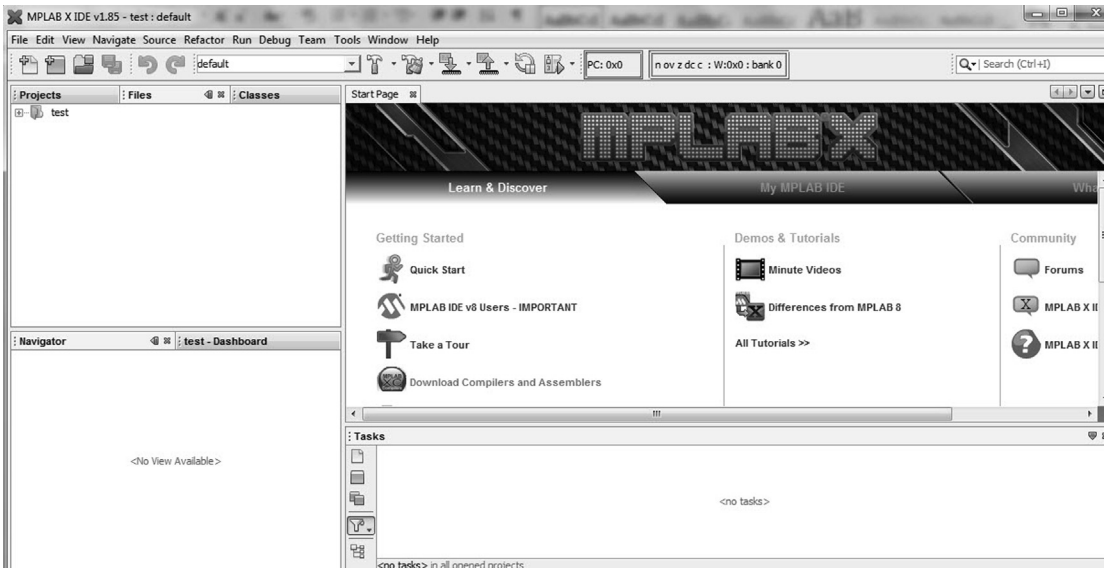

Figure 3.3: Opening Window of MPLAB X IDE.

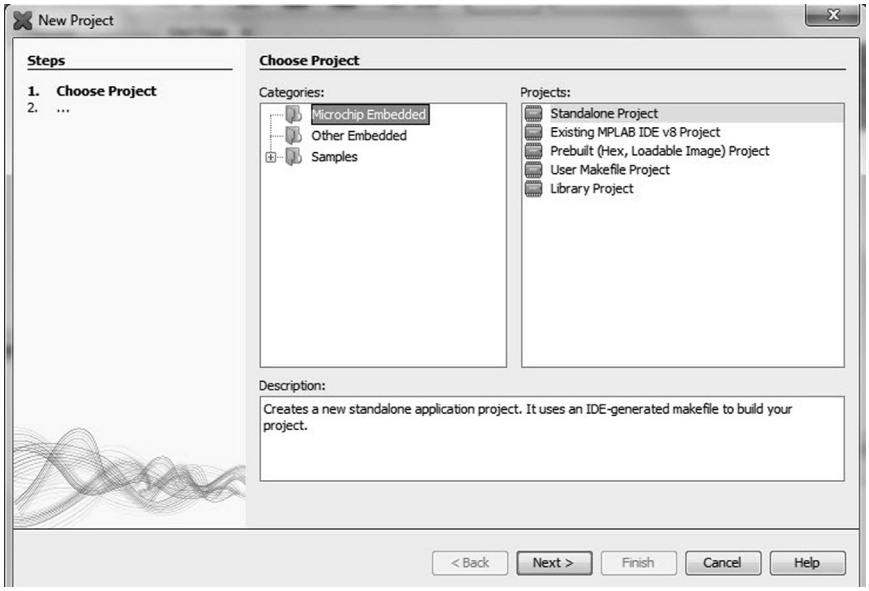

Figure 3.4: Create a Standalone Project.

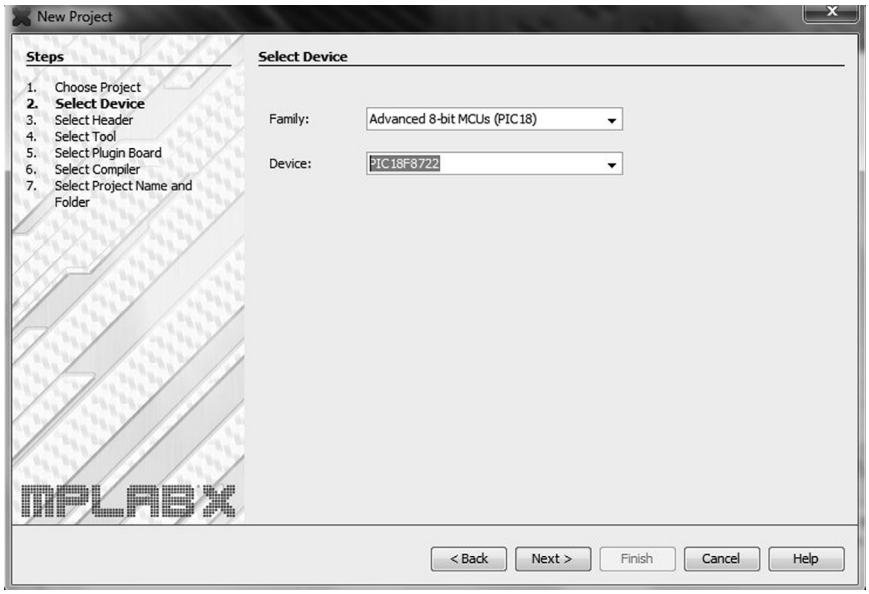

Figure 3.5: Select the Target Microcontroller.

Step 8. Right click *Source Files* on the left-hand window and select *New* → *C Main File*. Name the new source file as *NEWMAIN* (extension .C) as in Figure 3.9.
Step 9. Click Finish, and you should get an empty template C file as shown in Figure 3.10.

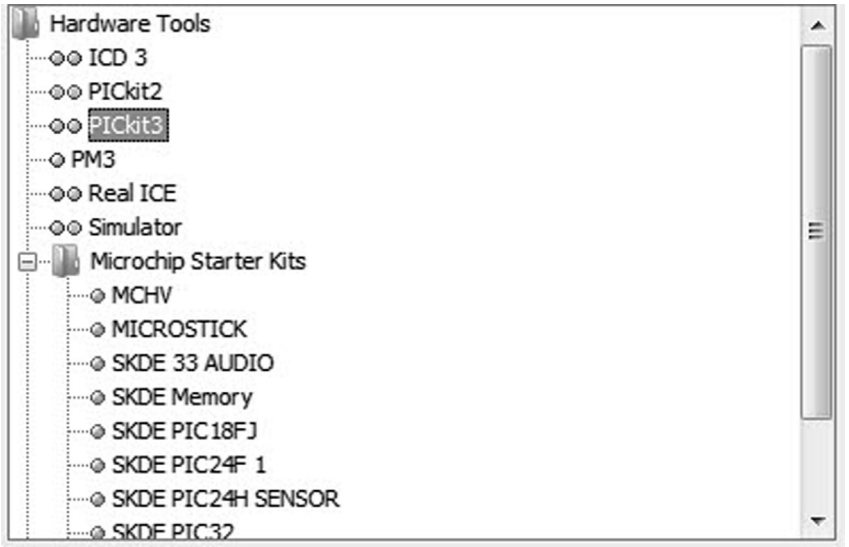

Figure 3.6: Select PICkit 3.

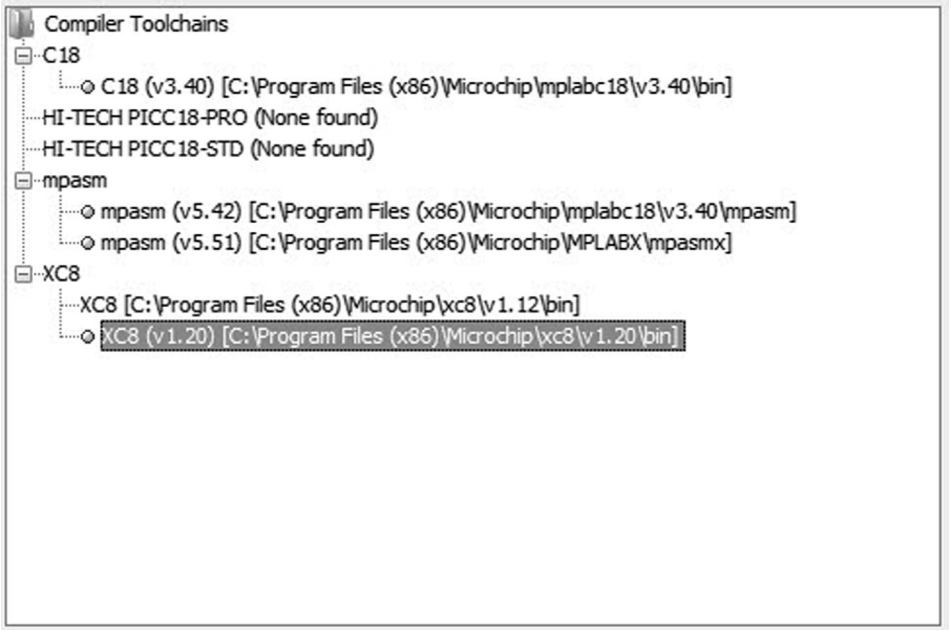

Figure 3.7: Select XC8 Compiler.

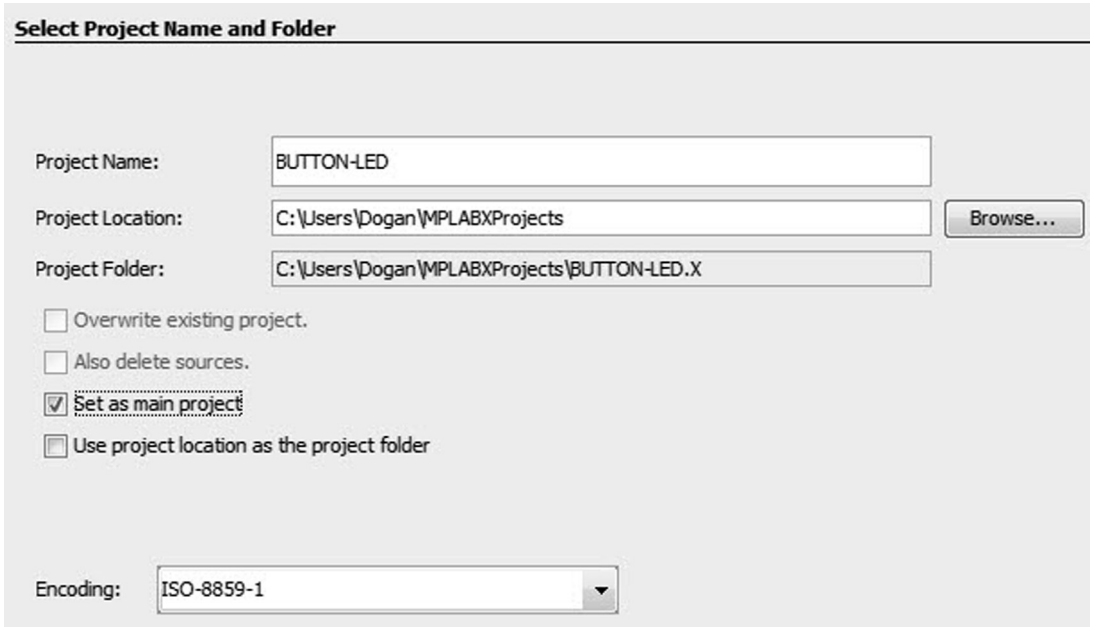

Figure 3.8: Give the Project a Name.

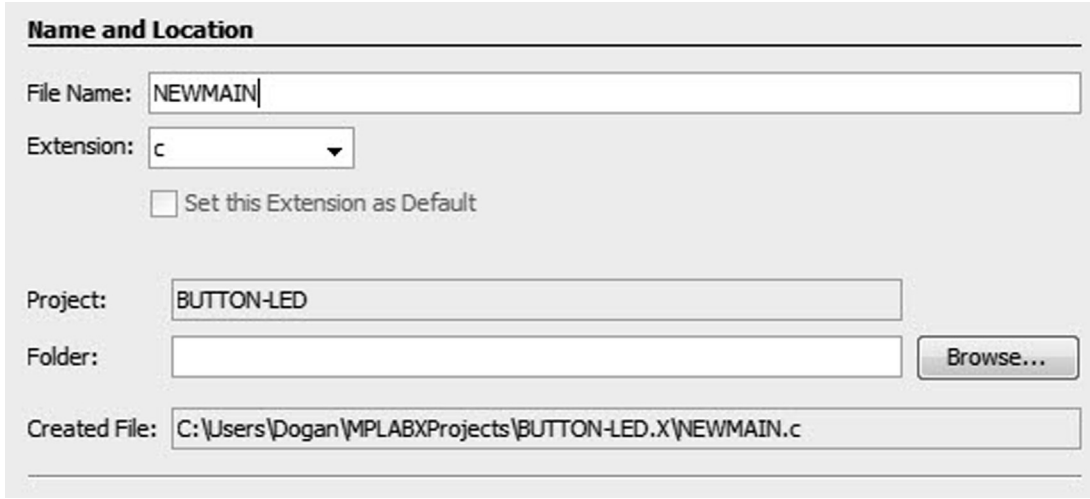

Figure 3.9: Create the New Source File.

```
1  /*
2   * File:   NEWMAIN.c
3   * Author: Dogan
4   *
5   * Created on 03 August 2013, 13:54
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 /*
12  *
13  */
14 int main(int argc, char** argv) {
15
16      return (EXIT_SUCCESS);
17 }
18
```

Figure 3.10: Template C File.

Step 10. Modify the file by inserting the following lines for our program. The program turns on the LED connected to port pin RD0 whenever push-button switch S1 (connected to port pin RB0) is pressed. See Figure 3.11 for part of the program listing in MPLAB X IDE:

```
*******************************************************************************
* File:   NEWMAIN.c
* Author:  Dogan
* Date:  August, 2013
*
* This program uses the PICDEM PIC18 EXPLORER DEVELOPMENT BOARD.
* The program turns on an LED when a push-button switch is pressed.
*
* The LED is connected to port pin RD0, and the switch is connected to
* port pin RB0 of the microcontroller.
*
*******************************************************************************/
#include <xc.h>

// Configuration: Ext reset, Watchdog OFF, HS oscillator
#pragma config MCLRE = ON, WDT = OFF, OSC = HS
//
// Define switch and LED connections
#define S1 PORTBbits.RB0
#define LED PORTDbits.RD0
//
// Define clock frequency
#define _XTAL_FREQ 10000000

//
```

```
// Start of main program
//
int main()
{
 TRISBbits.TRISB0 = 1;          // Configure RB0 as input
 TRISDbits.TRISD0 = 0;          // Configure PORTD as outputs
 MEMCONbits.EBDIS = 1;         // Enable PORTD I/O functions

 for(;;)                  // Do FOREVER
 {
  if(S1 == 0)LED = 1; else LED = 0;
 }
}
```

The description of the program is as follows:

- The *#include <xc.h>* statement at the beginning of the program identifies the microcontroller in use and calls the appropriate header files to include the processor specific definitions at the beginning of the program (notice that the mikroC Pro for the PIC compiler does not require a header file).
- The configuration statement *#pragma config MCLRE = ON, WDT = OFF, OSC = HS* defines the processor configuration. Here, master clear (reset) is enabled, watchdog timer is turned off, and the external high-speed crystal is selected as the clock source. The file *pic18_chipinfo.html* in the docs directory of the XC8 compiler installation (usually the folder: *C:\Program Files (x86)\Microchip\xc8\v1.20\docs\pic18_chipinfo.html*) contains a list of all the processors and a list of all possible configuration options for each processor.
- The statement *#define S1 PORTBbits.RB0* defines symbol S1 as port pin RB0. Similarly, the statement *#define LED PORTDbits.RD0* defines symbol LED as port pin RD0 of the microcontroller.
- The microcontroller clock frequency is then defined as 10 MHz.
- At the beginning of the main program, port pin RB0 is configured as an input port. Similarly, RD0 is configured as an output port.

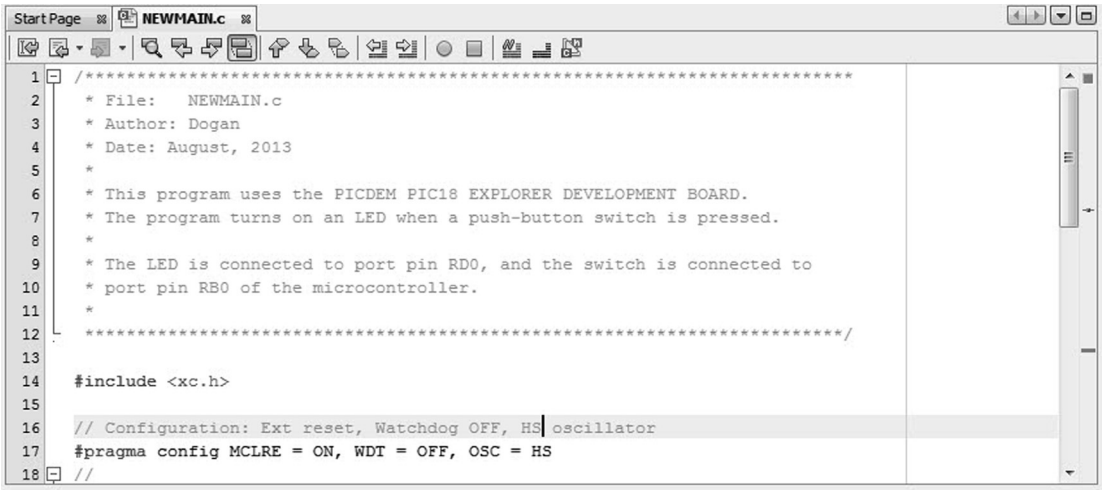

Figure 3.11: The Program Listing.

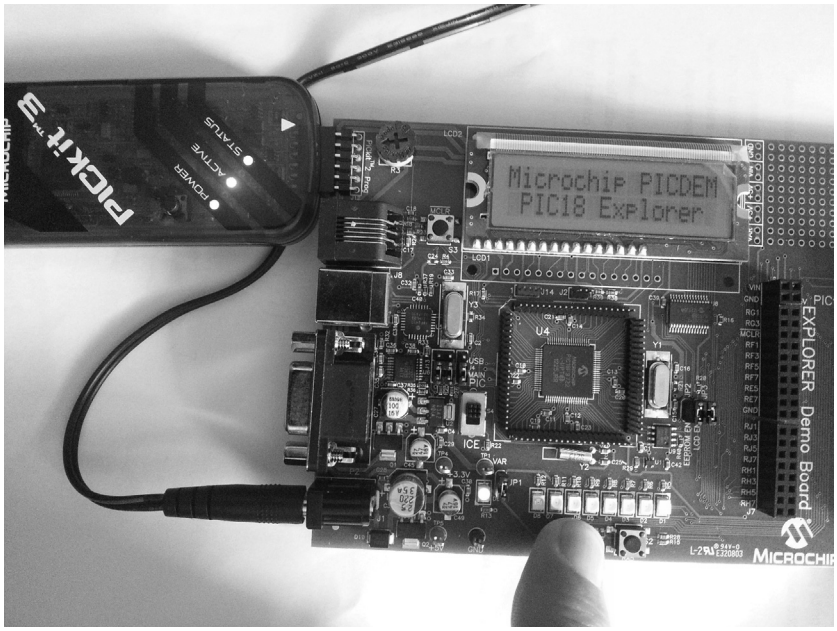

Figure 3.12: Turning the LED on.

- PORTD I/O functions are enabled by setting MEMCON bit EBDIS (see the PIC18F8722 data sheet).
- The program then enters an infinite loop where switch S1 is checked. Whenever the switch is pressed (i.e. when S1 becomes 0), the LED is turned on.

Step 11. Compile the program by clicking the *Build Main Project* button (shown as a hammer). The program should compile successfully and *Loading completed* message should be displayed.
Step 12. Connect the PICkit 3 programmer/debugger to the Explorer board. Click *Make and Program Device Main Project* button to load the program to the target microcontroller on the Explorer board. You should get the messages *Programming* and then *Programming/Verify complete* when the target microcontroller is programmed.
Step 13. The LED connected to RD0 should now turn on when push-button S1 is pressed (Figure 3.12).

Example 3.2 Flashing the LEDs

In this simple example, we will write a program to flash all the LEDs on the Explorer board with a 1-s interval.

Solution 3.2

The required program is called FLASH.C, and its listing is shown in Figure 3.13. Notice in this program that a 1-s delay is created using the built-in function Delay10KTCYx(*n*). This function creates a 10,000**n* instruction cycle delay. With a 10-MHz clock the instruction cycle is 10/4 = 2.5 MHz, which has the period of 0.4 μs. Thus, each unit of Delay10KTCYx

```
/*****************************************************************************
 * File:   FLASH.c
 * Author: Dogan
 * Date:   August, 2013
 *
 * This program uses the PICDEM PIC18 EXPLORER DEVELOPMENT BOARD.
 * The program flashes all the LEDs on the board with 1 s interval
 *
 * The LEDs are connected to PORTD of the microcontroller
 *
 *****************************************************************************/

#include <xc.h>

// Configuration: Ext reset, Watchdog OFF, HS oscillator
#pragma config MCLRE = ON, WDT = OFF, OSC = HS
//
// Define LED connections
#define LEDS PORTD
//
// Define clock frequency
#define _XTAL_FREQ 10000000

//
// Start of main program
//
int main()
{
   TRISD = 0;                    // Configure PORTD as outputs
   MEMCONbits.EBDIS = 1;         // Enable PORTD I/O functions

   for(;;)                       // Do FOREVER
   {
      LEDS = 0;                  // LEDs OFF
      Delay10KTCYx(250);         // 1 s delay
      LEDS = 0xFF;               // LEDs ON
      Delay10KTCYx(250);         // 1 s delay
   }
}
```

Figure 3.13: Flashing all the LEDs.

corresponds to 0.4 μs × 10,000 = 4 ms. To generate a 1-s delay, the argument should be 1000/4 = 250.

Example 3.3—Running in the Debug Mode

In this section, we will see how to debug the program developed in Example 3.2. The steps for debugging the program are given below:

- Compile the program for debugging by clicking *Build for Debugging Main Project* (Figure 3.14).

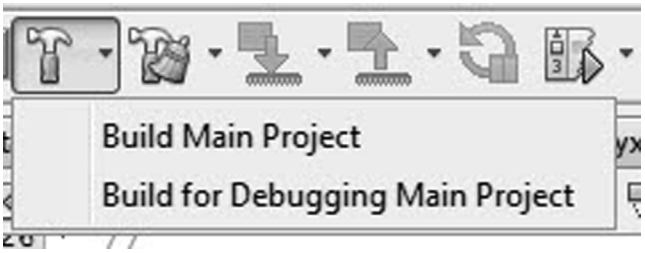

Figure 3.14: Compile for Debugging.

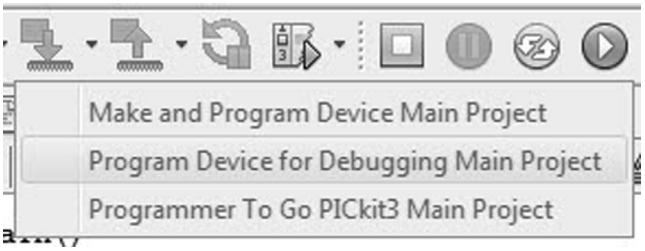

Figure 3.15: Load the Target Microcontroller.

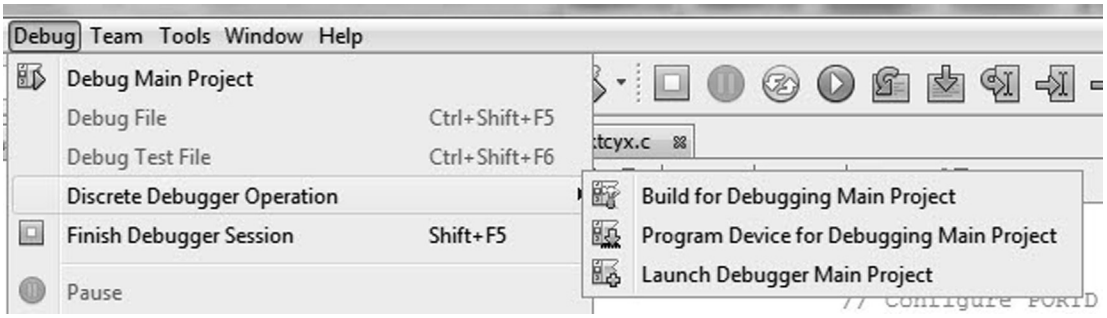

Figure 3.16: Launch the Debugger.

- Load the target microcontroller by clicking *Program Device for Debugging Main Project* (Figure 3.15).
- Start the debugger by clicking *Debug* → *Discrete Debugger Operation* → *Launch Debugger Main Project* (Figure 3.16). You should see the message *Target reset* displayed.
- Single step through the program by pressing F7. Step over the delay functions by pressing F8. As you single step through the program, you should see the LEDs turning on and off.

You can set breakpoints in the program by clicking the mouse on the numbers at the left-hand column of the program. Alternatively, breakpoints can be set by clicking Debug → New Breakpoint.

The program memory, Special Function Register (SFR), configuration bits and EE data can be observed by clicking *Window* → *PIC Memory Views* and then selecting the required display. Figure 3.17 shows a list of the SFR.

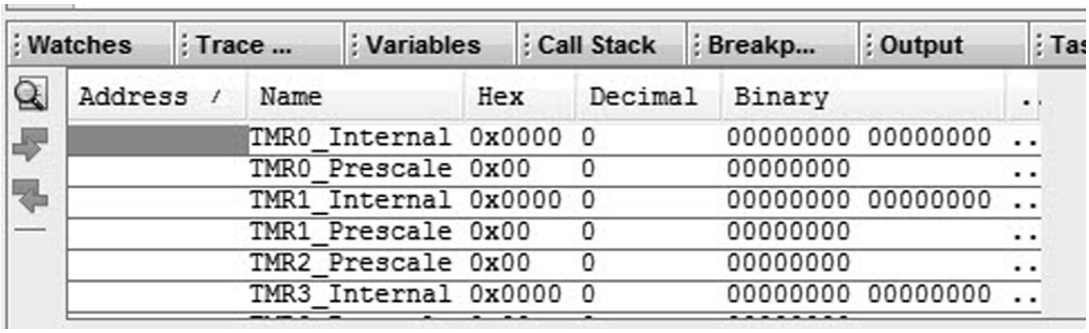

Figure 3.17: Displaying the SFR.

The program variables, breakpoints, call stack, etc. can be watched by clicking *Window* → *Debugging* and selecting the required feature.

3.3.1 Programming Other Boards Using the MPLAB X

In some applications, a program may be developed using the MPLAB XC8 compiler, but the development board we are using may not be a Microchip board. Under these circumstances, we can compile the program using the MPLAB XC8 and generate the hex code. An external programming device or a development board with an on-board programmer can then be used to load the program (hex code) to the target microcontroller.

An example is given here where the program developed in Example 3.2 is loaded to the microcontroller on the popular EasyPIC V7 development board (www.mikroe.com). This board includes an ICD 3 compatible socket for programming/debugging using Microchip programming/debugging hardware tools. In this example, the ICD 3 debugger/programmer device is used to program the EasyPIC V7. We shall be using the EasyPIC V7 development board together with mikroC Pro for PIC and MPLAB XC8 compilers in the project sections of this book.

Example 3.4

In this example, we will modify the program in Example 3.2 to flash the PORTD LEDs on the EasyPIC V7 development board. This board is equipped with a PIC18F45K22 microcontroller running with an 8-MHz crystal.

Solution 3.4

Create a project as described in Example 3.1. Select the microcontroller device as PIC18F45K22 and the hardware tool as ICD 3. The required program listing is shown in Figure 3.18. Note that there are some differences in the code since a different microcontroller is used here.

```
/*****************************************************************************
 * File:   FLASH.c
 * Author: Dogan
 * Date:   August, 2013
 *
 * This program uses the EasyPIC V7 development board.
 * The program flashes all the LEDs on the board with 1 s interval
 *
 * The LEDs are connected to PORTD of the microcontroller
 *
 *****************************************************************************/

#include <xc.h>

// Configuration: Ext reset, Watchdog OFF, HS oscillator
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
//
// Define LED connections
#define LEDS PORTD
//
// Define clock frequency
#define _XTAL_FREQ 8000000

//
// Start of main program
//
int main()
{
   TRISD = 0;                    // Configure PORTD as outputs

   for(;;)                       // Do FOREVER
   {
      LEDS = 0;                  // LEDs OFF
      Delay10KTCYx(250);         // 1 s delay
      LEDS = 0xFF;               // LEDs ON
      Delay10KTCYx(250);         // 1 s delay
   }
}
```

Figure 3.18: The Program Listing.

Compile the program as described previously. Now, we will transfer the program to the microcontroller on the EasyPIC V7 board. The steps are given below:

- Connect the USB port of ICD 3 to the PC.
- Connect the ICD 3 plug into the ICD socket on the EasyPIC V7 board.
- Turn on power to the development board.
- Load the program to the target microcontroller by clicking *Make and Program Device Main Project* in MPLAB X IDE.
- Enable the PORTD LED on the EasyPIC V7 board by setting switch SW3 PORTD to ON. You should see the LEDs flashing.

If you already have the older MPLAB IDE installed on your computer, then you may need to select the correct ICD 3 driver before loading the target microcontroller. The steps for this are as follows:

- Select All Programs → Microchip → MPLAB X IDE → MPLAB driver switcher. You should see a window as in Figure 3.19.
- Select ICD3 and MPLAB X, and click *Apply Changes* as in Figure 3.20.

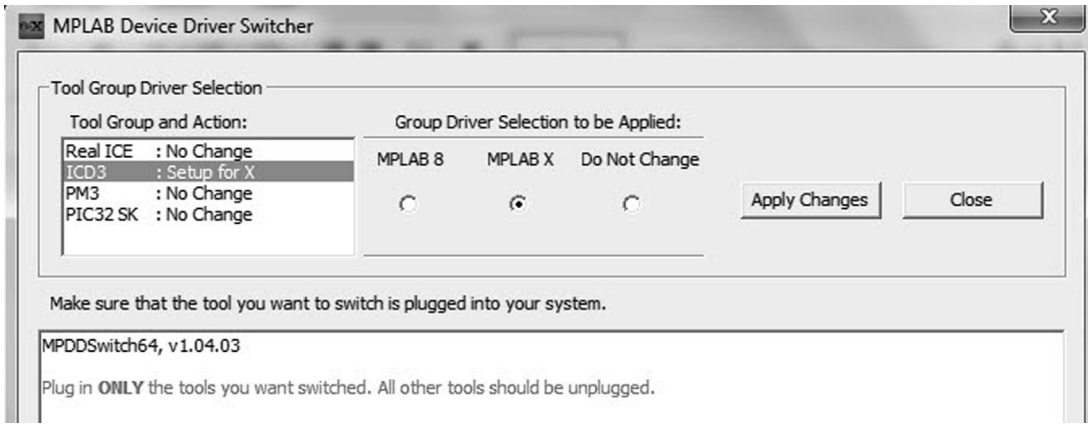

Figure 3.19: Driver Switcher Window.

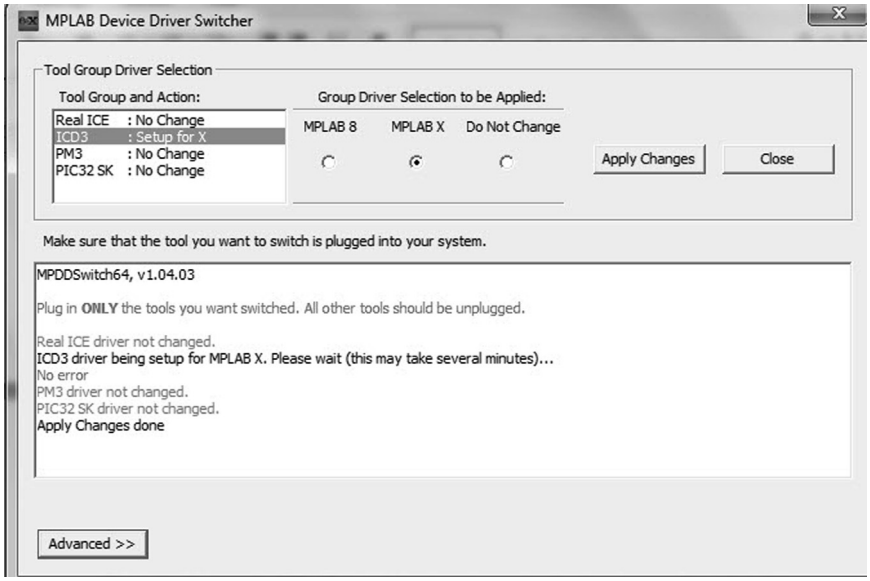

Figure 3.20: Select the ICD 3 Driver.

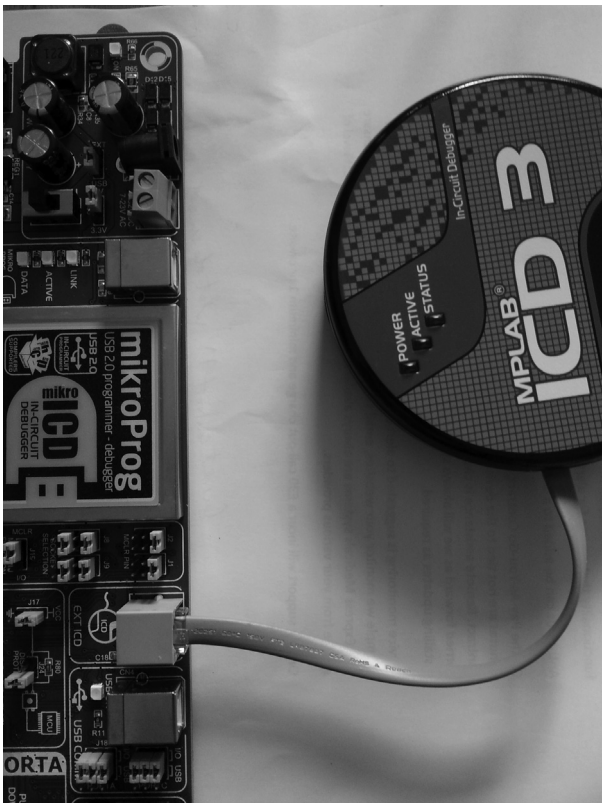

Figure 3.21: Connecting ICD 3 to the EasyPIC V7 Board.

Figure 3.21 shows the ICD 3 debugger/programmer connected to the ICD socket of the EasyPIC V7 board.

It is also possible to load the generated hex code to a PIC microcontroller using any type of PIC programming device as long as the device supports the microcontroller in use.

3.3.2 Features of the XC8 Language

In this section, we shall be looking at some features of the XC8 C language. Readers who are not familiar with the C language should read Chapter 4 before continuing with this chapter. Differences between the mikroC Pro for PIC and XC8 will be given where appropriate.

Detailed information about the XC8 language can be obtained from the *MPLAB XC8 Compiler User's Guide*, available from the Microchip Inc. web site.

Program Template

When a new XC8 program is created, the compiler generates the template shown in Figure 3.22. In our programs, we shall be modifying this template and use the one given in Figure 3.23 instead.

```
/*
* File: newmain.c
* Author: Dogan Ibrahim
*
* Created on November 6, 2013, 4:21 PM
*/
#include <stdio.h>
#include <stdlib.h>
/*
*
*/
int main(int argc, char** argv) {
return (EXIT_SUCCESS);
}
```

Figure 3.22: XC8 Program Template Created by the Compiler.

```
/*****************************************************************************
 * File:   Filename.c
 * Author: Author name
 * Date:   Date program created
 *
 * Write a brief description of the program here, including the type of
 * microcontroller used, I/O connections etc.
 *
 *
 ****************************************************************************/

#include <xc.h>

// Define Configuration fuse settings
#pragma config = .....
//
// Define clock frequency
#define _XTAL_FREQ .....

//
// Start of main program
//
int main()
{
   Program body.....
}
```

Figure 3.23: Modified Program Template.

An XC8 program has the following structure:

```
Program description
 #include <xc.h>
Configuration bits
Oscillator frequency
Global variables
Functions
Main program
```

A single header <xc.h> must be declared at the beginning of a program to declare all compiler and device-specific types and SFRs.

Variables Types

XC8 supports the variable types shown in Table 3.1. Notice that variable type *char* on its own is same as *unsigned char*.

In addition, XC8 compiler supports 24- or 32-bit floating point variables, declared using keywords *double* and *float*.

Constants

Constant objects are read only, and they are stored in the program memory of the microcontroller. Objects that do not change their values in a program should be stored as constants to save the limited random access memory space.

Table 3.1: Variable Types Supported by XC8

Type	Size (Bits)
Bit	1
Unsigned char	8
Signed char	8
Unsigned short	16
Signed short	16
Unsigned int	16
Signed int	16
Unsigned short long	24
Signed short long	24
Unsigned long	32
Signed long	32
Unsigned long long	32
Signed long long	32

Examples of constant declarations are as follows:

```
const int Max = 10;                    // constant integer
const int Tbl[] = {0, 2, 4,6, 8};      // constant table
const Tbl[] @ 0x100 = {2, 4, 6, 8};    // constant table
```

In the last example, the table constants are stored starting from program memory location 0x100.

Persistent Qualifier

The persistent qualifier can be used to indicate that variables should not be cleared by the runtime startup code. An example is given below:

```
static persistent int x;
```

Accessing Individual I/O Pins

An individual I/O pin can be accessed by specifying the port name, followed by the word bits, then a dot symbol, and the name of the port pin. An example is given below:

```
PORTBbits.RB0 = 1;  // Set RB0 of PORTB to 1
```

Accessing Individual Bits of a Variable

Individual bits of a variable can be set or reset using the following macro definitions:

```
#define setbit(var, bit) ((var)|=(1<<(bit)))
#define clrbit(var, bit)((var) = ~(1<<(bit)))
```

The following example sets bit 2 of variable Count:

```
setbit(Count, 2);
```

Specifying Configuration Bits

The #pragma config directive should be used to program the configuration bits for a microcontroller. An example is given below:

```
#pragma config = MCLR = ON, WDT = OFF
```

Assembly Language Instructions in C Programs

Assembly language instructions can be inserted into C programs using the asm statement. An example is given below:

```
asm("MOVLW 12");
```

Interrupt Service Routines

An interrupt service routine is recognized by the keyword *interrupt*, followed by the name of the routine. An example is given below:

```
void interrupt Myint(void)
{
      Body of the interrupt service routine
}
```

The interrupt priority can be specified after the keyword interrupt. For example,

```
 void interrupt low_priority Myint(void)
{
      Body of the interrupt service routine
}
```

If variables are to be accessed between the main program and the interrupt service routine, then it is recommended to declare such variables as *volatile*.

The statements *ei()* and *di()* enable and disable global interrupts, respectively.

Program Startup

The function main() is the first function executed after Reset. However, after Reset additional code provided by the compiler, known as the startup code, is executed first. The startup code transfers control to function main(). During the startup code, the global variables with assigned values are loaded with these values, and global variables with no assigned values are cleared to zeros. A jump to address 0 (Reset) is included at the end of function main(). Thus, if a return statement is included after the last instruction in main() or if the code execution reaches the final terminating bracket at the end of main(), then the program performs a software reset. It is recommended that a loop should be added to the end of a program so that the program never performs a soft reset at the end.

MPLAB XC8 Software Library Functions

The MPLAB XC8 compiler includes a large number of software libraries that can be very useful during program development. In this section, we shall be looking at some of the commonly used library functions.

__delay_ms, __delay_us _delay, _delay3

Functions __delay_ms and __delay_us can be used to create small millisecond and microsecond delays in our programs. Before using these functions, the clock frequency should be declared using the definition _XTAL_FREQ. Assuming the clock frequency is 8 MHz, the following code generates a 20-ms delay:

```
#define _XTAL_FREQ 8000000
__delay_ms(20);
```

Function _delay is used to create delays based on the instruction cycle specified in the argument. In the following example, the delay is 20 instruction cycles:

```
_delay(20);
```

Function _delay3 is used to create delays based on 3 times the instruction cycle. In the following example, the delay is 60 instruction cycles:

```
_delay3(20);
```

__EEPROM_DATA

This function stores data in the EEPROM memory. The data must be specified in blocks of 8 bytes. An example is given below:

```
__EEPROM_DATA(0x01,0x03,0x20,0x3A,0x00,0x78,0xAA,0x02);
```

ab, labs

Returns the absolute value of an integer (abs) or a long (labs). Header file <stdlib.h> must be declared at the beginning of the program. An example is given below:

```
#define <xc.h>
#define <stdlib.h>

signed int x, y;
x = –3;             // x = –3
y = abs(x);         // y = 3
```

cos, sin, tan

These functions return the results of trigonometric functions. The argument must be in radians. The header file <math.h> must be included at the beginning of the program. An example is given below to calculate the sin of 30° and store the result in variable s:

```
#include <xc.h>
#include <math.h>

#define conv 3.14159/180.0
float s;
s = sin(30 * conv);
```

cosh, sinh, tanh

These functions implement the hyperbolic functions cosh, sinh, and tanh. The header file <math.h> must be included at the beginning of the program. An example is given below to calculate the sinh of 3.2:

```
#include <xc.h>
#include <math.h>

float s;
s = sinh(3.2);
```

acos, asin, atan, atan2

These functions return the inverses of trigonometric functions in radians. The header file <math.h> must be included at the beginning of the program.

itoa

This function converts a number into a string with the specified number base. The header file <math.h> must be included at the beginning of the program. In the following example, number 25 is converted into a string in variable *bufr* with a hexadecimal base:

```
#include <xc.h>
#include <math.h>

char bufr[5];
itoa(bufr, 25, 16);
```

log, log10

Function log returns the natural logarithm of a floating point number. The function log10 returns the logarithm to base 10. The header file <math.h> must be included at the beginning of the program.

memcmp

This function fills *n* bytes of memory with the specified byte. The header file <string.h> must be included at the beginning of the program. In the following example, *bufr* is filled with 10 character 'x' s:

```
#include <xc.h>
#include <string.h>

char bufr[10];
memset(bufr, 'x', 10);
```

rand

This is a random number generator function. It returns an integer between 0 and 32,767 that changes on each call to the function. The header file <stdlib.h> must be included at the beginning of the program. The starting point is set using function *srand*. An example is given below:

```
#include <xc.h>
#include <stdlib.h>

srand(5);
j = rand();
```

round

This function rounds the argument to the nearest integer value in floating point format. The header file <math.h> must be included at the beginning of the program. An example is given below:

```
#include <xc.h>
#include <math.h>

double rnd;
rnd = round(23.456);
```

SLEEP

Used to put the microcontroller into the sleep mode.

sqrt

This function calculates the square root of a floating point number. The header file <math.h> must be included at the beginning of the program.

String Functions

Some of the string functions provided are as follows:

```
Strcat, strncat:     string concatenate
Strchr, strrchr:     string search
Strcmp, strncmp:     string compare
Strcpy, strncpy:     string copy
Strlen:              string length
Strstr, Strpbrk:     occurrence of a character in a string
```

tolower, toupper, toascii

Convert a lower case character to the upper case character, upper case character to the lower case character, and to ASCII.

trunc

This function rounds the argument to the nearest integer. The header file <math.h> must be included at the beginning of the program.

MPLAB XC8 Peripheral Libraries

In addition to the useful functions XC8 compiler offers a number of peripheral libraries that can be useful while developing complex projects using peripheral devices. Some of these libraries are for an LCD, SD card, USB port, CAN bus, I^2C bus, SPI bus, and so on.

3.4 Summary

This chapter has described the basic features of the MPLAB XC8 C compiler. Step-by-step examples are given to show how to use the MPLAB X IDE to create a project and how to load the executable code to the target microcontroller.

Examples are given to show how to load the target microcontrollers on the two popular development boards: the PICDEM 18 Explorer board and the EasyPIC V7 development board.

Finally, a list of some commonly used MPLAB XC8 functions are given. Interested readers can obtain further details from the *MPLAB XC8 Compiler User's Guide*.

3.5 Exercises

1. Write an XC8 C program to set bits 0 and 7 of PORT C to logic 1.
2. Write an XC8 C program to count down continuously and send the count to PORTB.
3. Write a C program to multiply each element of a 10-element array with number 2.
4. Explain how the individual bits of a port can be accessed. Write the code to clear bit 3 of PORTB.
5. Write an XC8 C program to calculate the average value of the numbers stored in an array. Assume that the array is called **M** and that it has 20 elements.
6. Write a function to convert inches to centimeters. The function should receive inches as a floating point number and then calculate the equivalent centimeters.
7. An LED is connected to port pin RB7 of a PIC18F8722 microcontroller. Write a program to flash the LED such that the ON time is 5 s, and the OFF time is 3 s.
8. Write a program to calculate the trigonometric sine of angles from 0 to 90° in steps of 10°. Store the results in a floating point array.
9. Explain how a compiled XC8 C program can be downloaded to the target microcontroller on the PICDEM 18 Explorer board.
10. Explain how a program can be debugged using the PICkit 3 programmer/debugger and the PICDEM 18 Explorer board.
11. Explain how a compiled XC8 program can be downloaded to the target microcontroller on the EasyPIC V7 development board.