<u>**Embedded Systems Module. 6EJ505**</u>

C tutorial 1

Rev. 6.9.16 tjw

Note: This tutorial uses the MPLABX (v. 3.4) Integrated Development Environment (IDE) and the XC8 compiler (v 1.38). It simulates code, and doesn't require any hardware.

This software is on University computers. You can also download MPLABX and the Users' Guide onto your home computer from here:

http://www.microchip.com/mplab/mplab-x-ide

You can download the XC8 C compiler from here:

http://www.microchip.com/mplab/compilers

(scroll down, and click "Downloads" and "Documentation" respectively)

Main Learning Points

C Programming	Using MPLABX, and the 18F2420
Constructing simple programs	Opening a project
Applying features of XC8 to realise "embedded" C programming	Compiling
Simple looping with **while( )**	Simple simulation

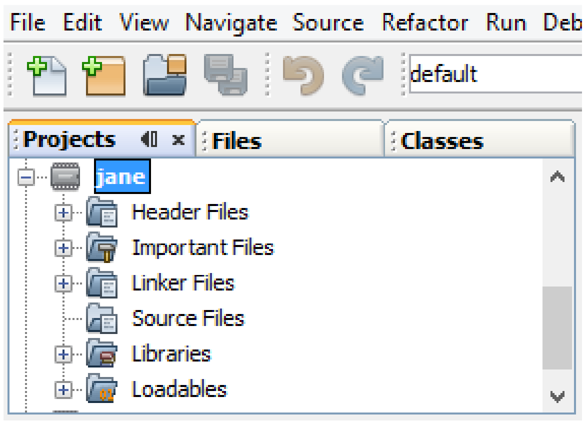
1. Getting started

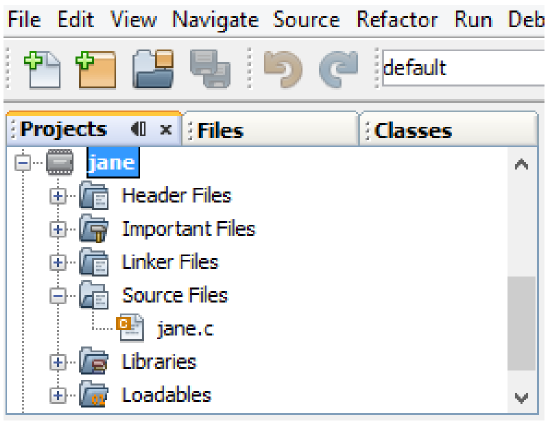
Opening a First Project

Open MPLABX and create a new project with suitable name, by selecting **File > New Project > Microchip Embedded > Standalone Project > Advanced 8-bit MCUs (PIC18) > PIC 18F2420 > Simulator (under Select Tool) > XC8(v. 1.38).** Then give the project a name and a location, as prompted. Your Projects screen, top left of the main screen, should then appear similar to Figure 1a). You can return to and change your project settings by right-clicking on the project name in the Project Window, and then selecting Project Properties (right at the bottom of the list).

Having created the project, open a new file, selecting **File > New File > C > C Source file.** Choose a name for the file (which can be the same as the project name), and note that it is placed in the project folder. It carries a .c extension, showing that it will be a C file. When you have completed you will see the file listed in the Projects window, as in Figure 1b).

Enter into the new file the code of Program Example 1.





a) b)

Figure 1

```
/***********************************************************************
C Tutorial 1, Example 1.(Prog Ex 14.1 in book)
Introductory Example of C Programming, with PIC 18 Series Microcontroller.
8-bit value output by Port B is continuously incremented.
TJW rev. 3.9.16
***********************************************************************/

//Include generic XC header file, which holds processor-specific info
#include <xc.h>

//Configuration settings
#pragma config PBADEN = OFF  //Port B is digital i/o

unsigned char counter;     //specify counter as unsigned character

void main (void){          //main function starts here
  TRISB = 0;               // initialise all bits of PORTB as output
  counter = 0;             //counter value is initialised to 0
  while (1) {
    PORTB = counter;          // Move 'counter' value to Port B
    counter = counter + 1; //Increment counter
   }
}
```

Program Example 1: Incrementing the output value of Port B

2 Building the project

It should now be possible to “build” the project. This compiles your C programme, and produces a version in binary code which can be downloaded to the target hardware. Click on the hammer symbol on the tool bar. If all is correctly entered and linked, the **BUILD SUCCESSFUL** message should ultimately appear in the Output window. A number of subsidiary files are either accessed or created at this time, following the general pattern of Figure 2.

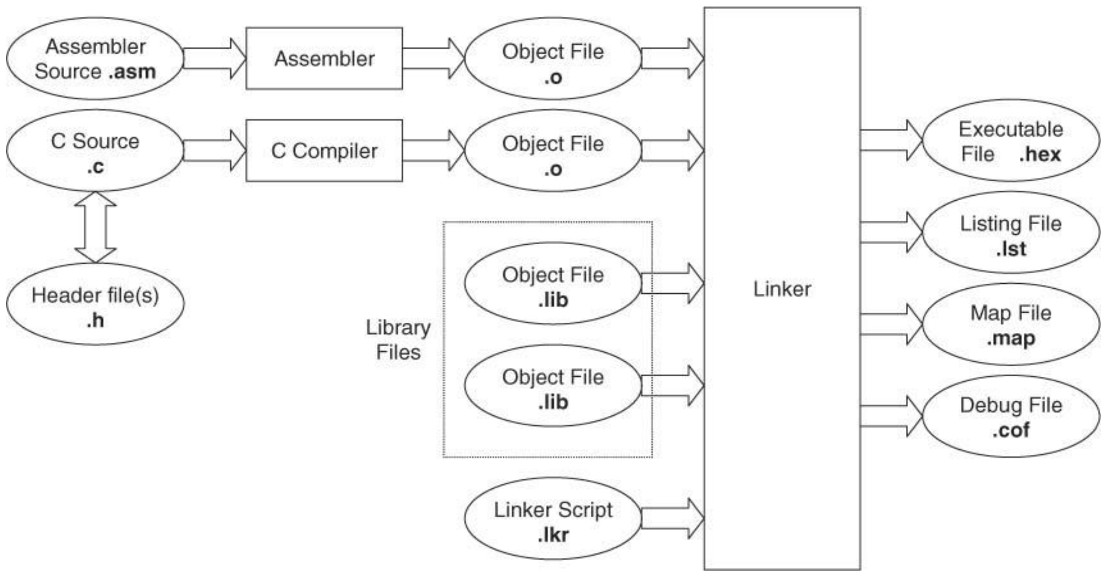


Figure 2. A generic C file structure, extensions may differ in detail from XC8

If the project did not build successfully, it will be necessary to explore the resulting error messages. The compiler first checks that the program has correct syntax, i.e. it is written to obey the formatting rules of C. Even if you did not get any errors, try removing a semicolon or inserting some other small error, and check the response of the compiler on build. If there is a syntax error, the compiler reports this in the Output window, with a line number. It is worth noting that the line number where the compiler perceives the error may not be the exact one where correction is needed. As a simple example, try 'commenting out' the opening brace of **main**. You will see that the syntax error is reported as being in the line which follows – which on its own is a perfectly good piece of C code!

3 The xc.h Header File

This is a generic header file which provides processor-specific information and settings for the program. Its use gives access to the SFRs of the processor in use.

4 Simulating a C program

Explore now the simulation of Program Example 1. You should already have specified the MPLABX simulator as your Debug tool when creating the Project. Now select **Debug** > **Debug Main Project**. This forces another program Build, and launches the simulation, indicated by the moving green bar bottom right of the MPLABX screen. The simulator is now controlled by the button bar shown in Figure 3. This should appear within the main toolbar, or you may need to expand it via the double arrow symbol.

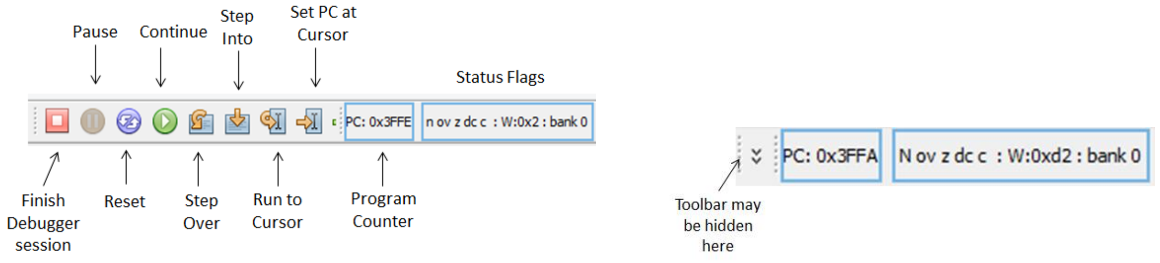


Figure 3

The program is running madly, but you have no way yet of seeing what it's doing! So pause program execution, and then try single stepping through it, using the *Step Into* button. This causes you to step through the little three-line loop at the heart of the program.

You can now "watch" program variables by selecting **Debug** > **New Watch**, which opens the window shown in Figure 4. In this, you can observe *Special Function Registers* (SFRs, we meet these in lectures) which are internal to the microcontroller, or *Global Symbols*, which are software variables introduced in the program.

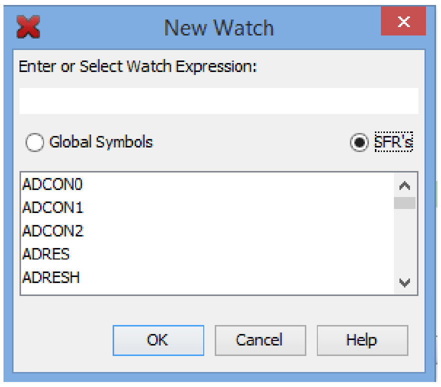


Figure 4

Display the values of **counter**, **PORTB** and **PCL** (Program Counter, lower byte), which will then appear in a *Variables* window across the bottom of the screen. It is then possible to see in this window the values of **counter** and **PORTB** being incremented. From here, program execution stays indefinitely in the **while** loop. Reset the program from the toolbar, to see it run from the beginning.

5 A second C example – the Fibonacci program

Consider now a program of slightly increased complexity. Program Example 2 provides a C version of a Fibonacci series generator; in a Fibonacci series every number is the sum of the two previous ones, e.g. 0,1,1,2,3,5,8,13,21,34…. The program calculates the series, first by going up to a certain level and then working backwards again. This action is repeated indefinitely.

```
/************************************************************************
C Tutorial 1, Example 2: Fibonacci
In a Fibonacci series each number is the sum of the two previous numbers.
This program calculates Fibonacci numbers within an 8-bit range,
Program intended for simulation only, hence no input/output.
 rev. 14.9.14
;***********************************************************************/

#include <xc.h>

//these memory locations hold the Fibonacci series
unsigned char fib0; //lowest number
                    //(oldest when going up, newest when reversing down)
unsigned char fib1; //middle number
unsigned char fib2; //highest number
unsigned char fibtemp;  //temporary location for newest number
unsigned char counter;  //indicates which value series has reached

void main (void) {
  fib0 = 0;
  fib1 = 1;
  fib2 = 1;
  counter = 3; //have preloaded the first three numbers, so start at 3
  while (1){
    while (counter<12){
        fibtemp = fib1 + fib2;
        counter = counter + 1;
//now shuffle numbers held, discarding the oldest
        fib0 = fib1; //first move middle number, to overwrite oldest
        fib1 = fib2;
        fib2 = fibtemp;
        }
//when reversing down, we will subtract fib0 from fib1 to form new fib0
    while (fib0>0){
        fibtemp = fib1 - fib0; //latest number now placed in fibtemp
        counter = counter - 1;
//now shuffle numbers held, discarding the oldest
        fib2 = fib1; //first move middle number, to overwrite oldest
        fib1 = fib0;
        fib0 = fibtemp;
        }
    }
}
```

Program Example 2

6 Some points on declaring variables

Following the opening comments, the early program lines declare five unsigned character variables, while the first three lines within **main( )** initialises three of these to certain values. There are certain ways in which this process can be shortened. Firstly, a declaration of one data type need not be confined to a single variable. Therefore, the five variables *could* all be declared together, as

```
     unsigned char fib0, fib1, fib2, fibtemp, counter;
```

It would be up to the programmer to decide whether this format was preferable. One disadvantage is that it is less easy to add a comment per variable, should this be desired.

It is also possible to initialise a variable at the time of declaration. The following would be a possible format:

```
     unsigned char fib0 = 0;  //lowest number
     unsigned char fib1 = 1;  //middle number
     unsigned  char  fib2 = 1; //highest number
```

It would be up to the programmer to decide whether this format gave any advantage.

7 Simulating the Fibonacci program

Copy the Fibonacci program into an appropriate new project and simulate with MPLABX. Open a Watch window, with **PCL** and all variables displayed. Single-step through the program and, once in **main**, observe how the looping is controlled. Notice how execution moves forward when **counter** reaches 12.

Coding Exercises

1. Appling the conditional **while( )** loop seen in the Fibonacci program, rewrite the first program, so it counts up to 10, and then starts again from 0, doing this repeatedly. Test through simulation.

2. Rewrite the first program, so it counts up to 10, then counts down 0, doing this repeatedly. Test through simulation.