Chapter 3

Using Eight Outputs

FLASHING 8 LEDs.

For this program we are going to flash all eight outputs on PORTB.
Consider the circuit as shown in Figure 3.1.
In order to achieve our objective we will be using the following steps:

- Open MPLAB.
- Create the C file (see Figure 2.3). Call it **flash8.c**.

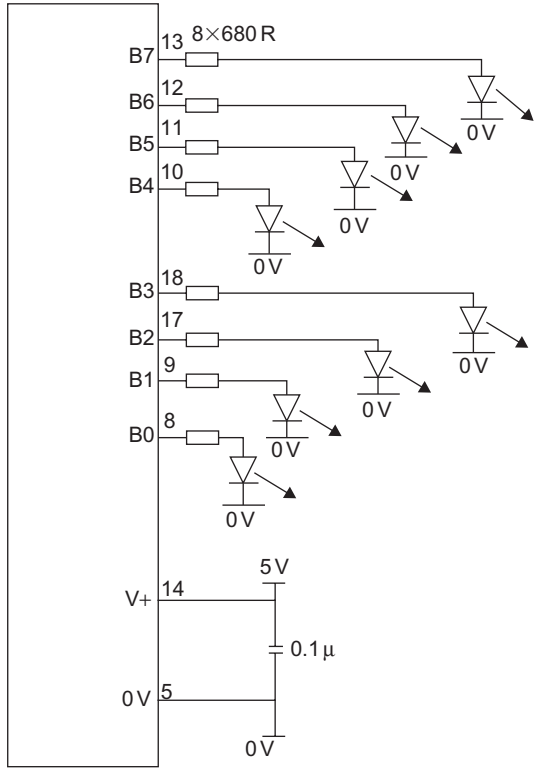

FIGURE 3.1 Eight outputs on PORTB.

- Create a project called flash8, using the Project Wizard (see Figures 2.7–2.13).
- Add **flash8.c** to your project (see Figure 2.16).
- Compile your project (see Figure 2.18).
- Program your target device (see Figure 2.23).

To create the C file flash8.c copy and paste lines 1–13 of flash.c onto the file editor screen (see Figure 2.15) and add the following lines of code shown in Figure 3.2.

NB. The line numbers are not required in the program; they are only included as a guide.

EXPLANATION OF THE CODE

- Line 17 has written to all 8 bits of PORTB with the instruction PORTB = 0b11111111.
- Line 18 waits for 1 s.
- Line 19 turns all PORTB outputs off and so on.

In order to compile (build) the code a new project called flash8 must be made and the file flash8.c added to it.

```
1.    #include <p18f1220.h>
2.    #pragma config WDT=OFF , OSC=INTIO2 , PWRT = ON, LVP=OFF, MCLRE = OFF
3.    #include <delays.h>
4.
5.    void main (void)
6.    {
7.              //SET UP
8.              // OSCCON defaults to 31kHz. So no need to alter it.
9.              ADCON1 = 0x7F;  //all IO are digital  or 0b01111111 in binary
10.             TRISA = 0b11111111;     //sets PORTA as all inputs
11.             PORTA = 0b00000000;     //turns off PORTA outputs, not required, no outputs
12.             TRISB = 0b00000000;     //sets PORTB as all outputs
13.              PORTB = 0b00000000;    //turns off PORTB outputs, good start position
14.
15.   while (1)
16.     {
17.             PORTB = 0b11111111;             // all outputs on
18.             Delay100TCYx(78);               // wait 1s
19.             PORTB = 0b00000000;             // all outputs off
20.             Delay100TCYx(78);               // wait 1s
21.
22.             PORTB = 0b10101010;             // alternate outputs on
23.             Delay100TCYx(78);               // wait 1s
24.             PORTB = 0b01010101;             // alternate outputs on
25.             Delay100TCYx(78);               // wait 1s
26.     }
27.   }
```

FIGURE 3.2 Flash8.c code.

If you are confident with what you are doing then you can use the short cut instead of using the Project Wizard, as follows:

- Select Project—New as shown in Figure 3.3.
- In the New Project box that opens up enter the project name, flash8, and the directory location, PIC Progs.
- Now add your file to the project with Project—Add files to project as shown in Figure 3.4.

In the Add files project box enter the file name flash8 and the directory PIC Progs in the Look in box and click Open.

- Now build your project (see Figure 2.17)
- Program the target device (see Figure 2.23)
- Try out your Microcontroller in your circuit

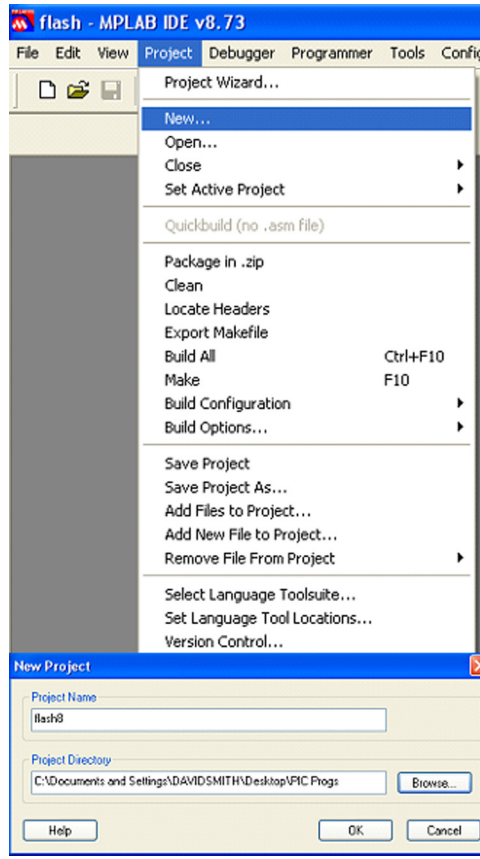

FIGURE 3.3 Making a new project.

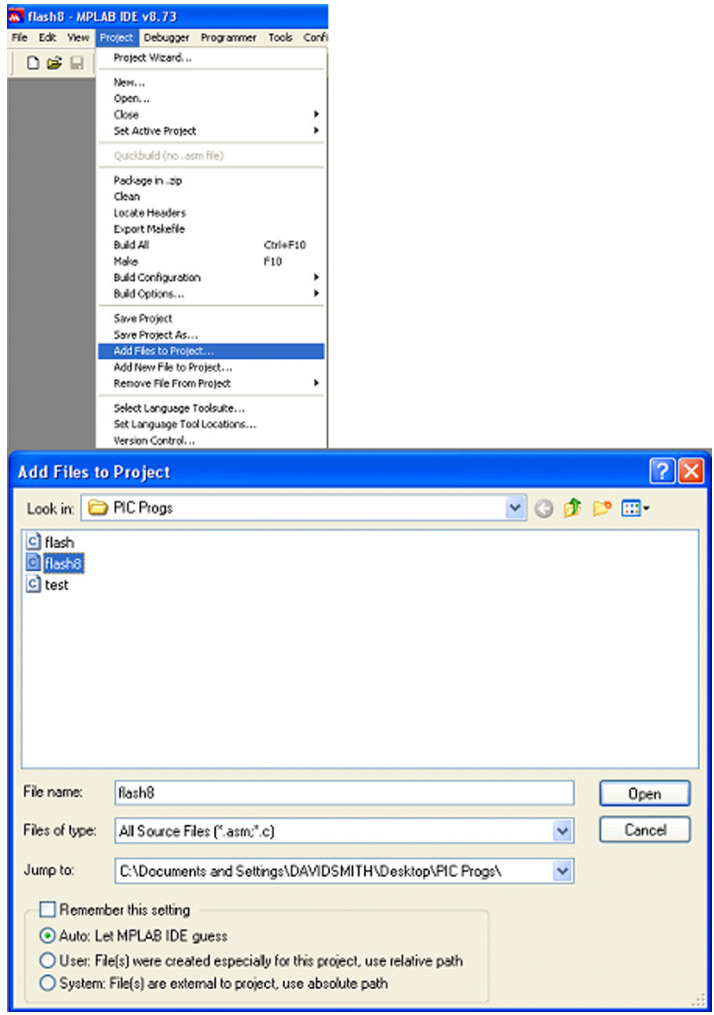

FIGURE 3.4 Adding a file to a project.

In order to help your understanding of what we have done try adding a few more lines of code to produce a different pattern of flashing lights. You can also speed up and slow down the flash rate by altering the number in the delay. A 1 s delay is given by:

```
Delay100TCYx(78);
```

Where the delay of 100 timing cycles is called 78 times.
So,

```
Delay100TCYx(39);    // is a ½ s delay
Delay100TCYx(156);   // is a 2 s delay
```

You hopefully will now have achieved what I believe is the most difficult part of embedded control, i.e., getting your first program to work!

LOOPING IN A PROGRAM

Following on from this success and I do not apologize for it, but I love seeing the lights flash. Let us write a program that will flash a sequence of lights a number of times before moving on to another sequence.

That is flash the lights on and off at 1 s intervals five times and then alternate four lights on and four off at say ½ s intervals for 10 times before repeating.

Copy and Paste what you need from your previous code and add the extra.

I will call the program **disco.c** which is illustrated below.

THE DISCO CODE

```
1.  // disco.c by DW Smith, 21 September 2011
2.  #include <p18f1220.h>
3.  #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
4.  #include <delays.h>
5.  int count;
6.
7.  void main (void)
8.  {
9.          // SET UP
10.         // OSCCON defaults to 31 kHz. So no need to alter it.
11.   ADCON1 = 0x7F;//all IO are digital or 0b01111111 in binary
12.   TRISA = 0b11111111;   //sets PORTA as all inputs
13.   PORTA = 0b00000000;   //turns off PORTA outputs
14.   TRISB = 0b00000000;   //sets PORTB as all outputs
15.   PORTB = 0b00000000;   //turns off PORTB outputs, good start position
16.
17. while (1)
18.   {
19. for (count = 0; count<5; count++)
20.         {
21.         PORTB = 0b11111111;          // all outputs on
22.         Delay100TCYx(39);            // wait 1/2 s
23.         PORTB = 0b00000000;          // all outputs off
24.         Delay100TCYx(39);            // wait 1/2 s
25.         }
26.
27.         for (count = 0; count<10; count++)
28.         {
29.         PORTB = 0b10101010;          // all outputs on
30.         Delay100TCYx(39);            // wait 1/2 s
31.         PORTB = 0b01010101;          // all outputs off
32.         Delay100TCYx(39);            // wait 1/2 s
33.         }
34.   }
35. }
```

Explanation of the Disco Code

One of the first things that concerned me about C code is the number of curly brackets, { } that seem to be sprinkled around haphazardly. Do not worry they come in pairs, and in order to read the code easily make sure you put the pairs in the same column.

The brackets identify the code that comes with a particular function (or method).

In the main function—void main (void) the code for this is written between the brackets shown on lines 8 and 35.

In the while (1) function the code for this is written between lines 18 and 34. This continually runs the program after the initial setup has been configured.

The new lines of code not used in the flash.c program are:

Line 1. The title of the program is a good idea for identifying the code especially if you print it.

Line 18. The program counts the number of times it has gone round a loop. This is done with a variable I have called count. In order to use a variable we must declare it in the program and say what kind of variable it is. My variable is an integer and has an initial value of 0. Integers can have values up to 32,767. A long integer can have a value up to 2,147,483,647, i.e., long population = 1,000,000 would specify a long integer as population with an initial value of 1,000,000. NB. The variable is declared in the function in which it is being used.

Line 19. **The for loop**. The setup for the for loop is shown below;

```
for (start condition; end condition; increment)
{
}
```

The section between the ( ) brackets is made up of three parts, the first part is the start condition, i.e., count = 0, the second part is the end condition, i.e., when count <5 is no longer true, which means when count is 5 or more. The third part is what are the step increments, it could be 10 or 100; in this case it is 1, shown as count++ which means count +1.

The for loop starts the count at zero, checks if the count is less than 5 which it is. The code then executes returning to the for statement and adding 1 to count, checking, executing, and incrementing every time it goes around the loop.

It will finish and move on to the next section when count is no longer less than 5. In our case when count = 5.

Line 25. At the end of the loop, count = count +1 means that we add 1 to the count, counting the number of times we have gone round the loop.

Line 27. We are zeroing the count in order that we can count the next loop 10 times.

This may seem confusing, it did confuse me, but after writing one or two you will soon understand the idea!

I hope by now you are getting used to programming your micro in C and can see the possibilities that this powerful language has to offer.

To make sure you have understood the Disco program try adding a number of other sequences and alter the times to speed up and slow down the lights. If you want you can also include PORTA in your code for more outputs. But be aware PORTA bit RA5 is an input ONLY pin. So you have a maximum 15 outputs with the 18F1220. The PIC18F2220 has 25 I/O and the PIC18F4220 has 36 I/O.

Try producing a set of traffic lights (two directions, at a cross road) if you need extra practice.

We can copy and paste the required bits of code from the disco program to use in the programs we are going to write, but rather than this let us copy the code we require and call this a header program which we can open und modify each time.

THE HEADER PROGRAM

Figure 3.5 shows the header program.

In the header:

- Enter the program name and date in Line 1.
- Your program will be written between the curly brackets {Lines 17 and 21}.

```
1.      // Name.c   by DW Smith,  dd/mm/yyyy
2.      #include <p18f1220.h>
3.      #pragma config WDT=OFF , OSC=INTIO2 , PWRT = ON, LVP=OFF, MCLRE = OFF
4.      #include <delays.h>
5.
6.      void main (void)
7.      {
8.              //SET UP
9.              // OSCCON defaults to 31kHz. So no need to alter it.
10.             ADCON1 = 0x7F;  //all IO are digital  or 0b01111111 in binary
11.             TRISA = 0b11111111;     //sets PORTA as all inputs
12.             PORTA = 0b00000000;     //turns off PORTA outputs
13.             TRISB = 0b00000000;     //sets PORTB as all outputs
14.              PORTB = 0b00000000;    //turns off PORTB outputs, good start position
15.
16.     while (1)
17.        {
18.
19.
20.
21.        }
22.     }
```

FIGURE 3.5 The header program.

Save the program as Header.c

NB. This header can be used for other PIC Microcontrollers by changing line 2 to the new number, i.e., 18f4220. This provides an easy way to use any micro you require.

So far we have only used outputs; let's move on to using inputs.