Chapter 11

Fault Finding, Using the Simulator, and the In-Circuit Debugger

The simulator and in-circuit debugger (ICD) software are loaded automatically with MPLAB.

We will first of all consider the simulator and then the in-circuit debugger.

THE SIMULATOR USING MPLAB

The simulator is a software tool that enables the user to fault find on the program before blowing it into the chip. The simulator can set break points in the code and the program when run will halt at the break point. The Special Function Registers and User Registers can then be viewed by observing them in the Watch Window.

The simulator is excellent for:

- Ensuring that timing cycles are correct. This is done by using the stopwatch facility.
- Checking if the program branches the way it was intended as in an if statement.
- Checking that inputs are functioning correctly using the stimulus file.
- Checking the outputs are switching as required.

Let us look at a number of examples to understand the operation of the simulator.

Stopwatch

We will first of all consider using the stopwatch to check if our delays are correct. Let us consider the program explained in Chapter 9, **TMR0_30min** written below.

```
1. // TMR0_30min.C DW Smith 27-2-12.
2. #include <p18f1220.h>
3. #pragma config WDT=OFF, OSC=LP, PWRT=ON, LVP=OFF, MCLRE=OFF
```

```
4.  #include <delays.h>
5.  int TMR0x;
6.  int TMR0temp;
7.  void Delay30min(void)
8.  {
9.     TMR0H=0;
10.    TMR0L=0;
11.    while(TMR0H<0xE1) TMR0temp=TMR0L;      //to read TMR0H we must read TMR0L
12. }
13. void main (void)
14. {
15.    //SET UP
16.    // OSCCON defaults to 31kHz. So no need to alter it.
17.    ADCON1=0x7F;              //all IO are digital or 0b01111111 in binary
18.    TRISA=0b11111111;         //sets PORTA as all inputs
19.    PORTA=0b00000000;         //turns off PORTA outputs, not required, no outputs
20.    TRISB=0b00000000;         //sets PORTB as all outputs
21.    PORTB=0b00000000;         //turns off PORTB outputs, good start position
22.
23.    T0CON=0b10000111;         //sets TMR0 on, 16bits, internal clock, prescaler assigned,
                                 prescaler=256
24. Delay30min ();
25. PORTAbits.RA0=1;
26. while(1);
27. }
```

The program waits 30 min before turning an LED on, on PORTA, RA0.

Set up a project called **TMR0_30min.mcp**, add the file **TMR0_30min.C** to it and open it. (Refer to Chapter 2 if required.) Build the project.

The screen should resemble that of Figure 11.1.

The program calls a delay of 30 min on line 24 and then turns the LED on PORTAbits.RA0 on.

This program has an error in it. The LED does not turn on. If it was run on hardware it would take 30 min before the LED lit, if it didn't the problem could be in the program or the hardware.

We require to see that the 30 min delay is functioning properly to do this:

We need to

- turn on the simulator
- add break points
- display the stopwatch
- display the watch window
- view the PORTA register in the watch window.
- Turn on the simulator with the toolbar using, Debugger, Select Tool, 5 MPLAB SIM as shown in Figure 11.2.
- Double click on the grey area next to line 27 and again by line 28 in your code.

You should observe two break points (red circle with B inside) have been introduced as shown in Figure 11.3.

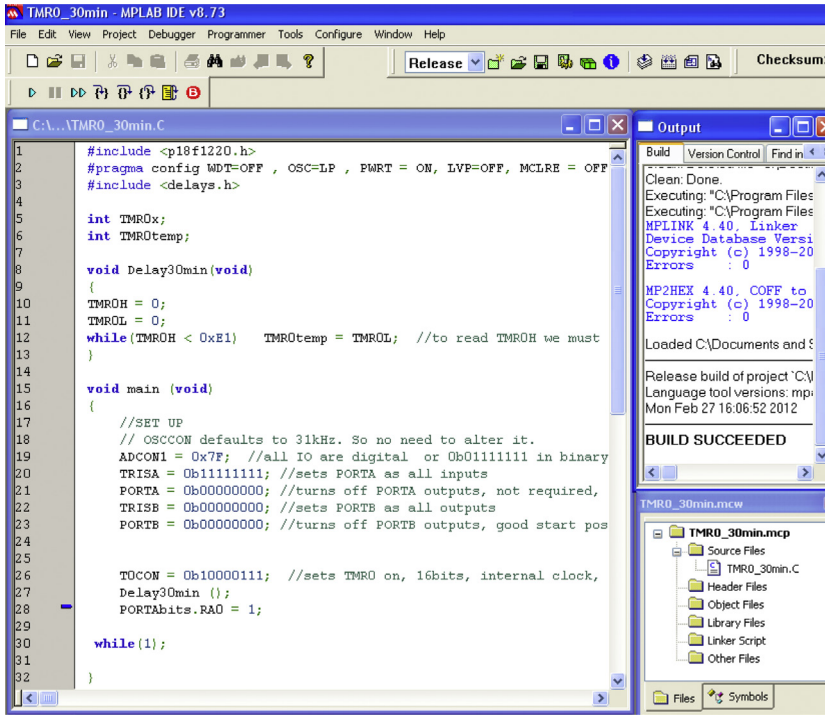


FIGURE 11.1 The TMR0_30min workspace.

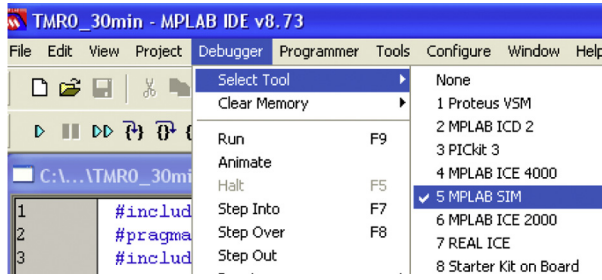


FIGURE 11.2 Turning on the simulator.

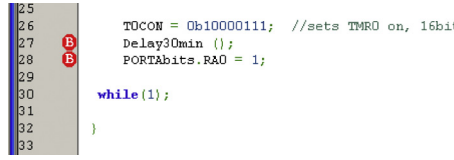


FIGURE 11.3 Adding break points.

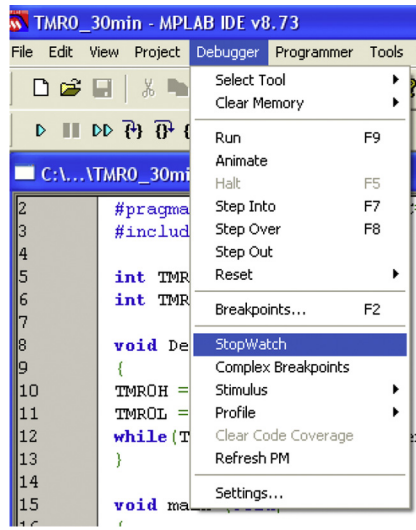


FIGURE 11.4 Displaying the Stopwatch.

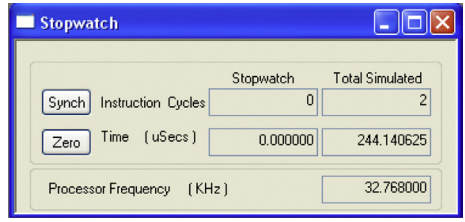


FIGURE 11.5 The Stopwatch.

The break points can be removed by double clicking on them.

- Display the stopwatch (Figure 11.4) using Debugger, Stopwatch. The stopwatch is shown in Figure 11.5.

The processor frequency needs to be set to 32.768 kHz. This is done with Debugger, Settings indicated in Figure 11.6.

- Open the Watch window using View as shown in Figure 11.7. The window is shown in Figure 11.8.
- Display PORTB, a special function register (SFR) by selecting it from the AddFSR list as shown in Figure 11.9.

Notice from Figure 11.9 that the value of PORTB can be shown as hex, decimal, or binary as required.

Any SFR or user register (ADD Symbol) can be added to the watch window.

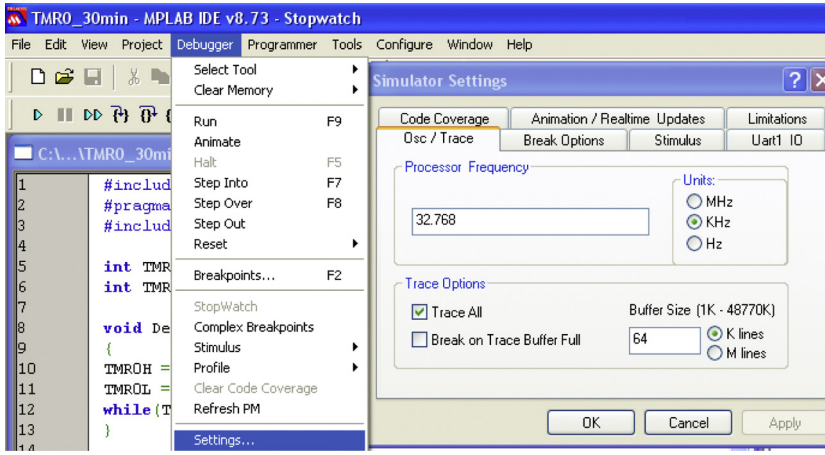


FIGURE 11.6 Setting the clock frequency.

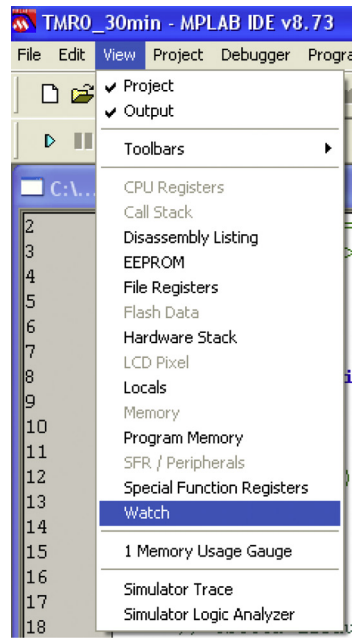


FIGURE 11.7 Opening the Watch window.

Your workspace screen should now look similar to the screen shot of Figure 11.10.

We are now in a position to run the program and determine if the value of the Delay30 min() timer routine is 30 min and why the LED does not come on.

FIGURE 11.8 The Watch window.

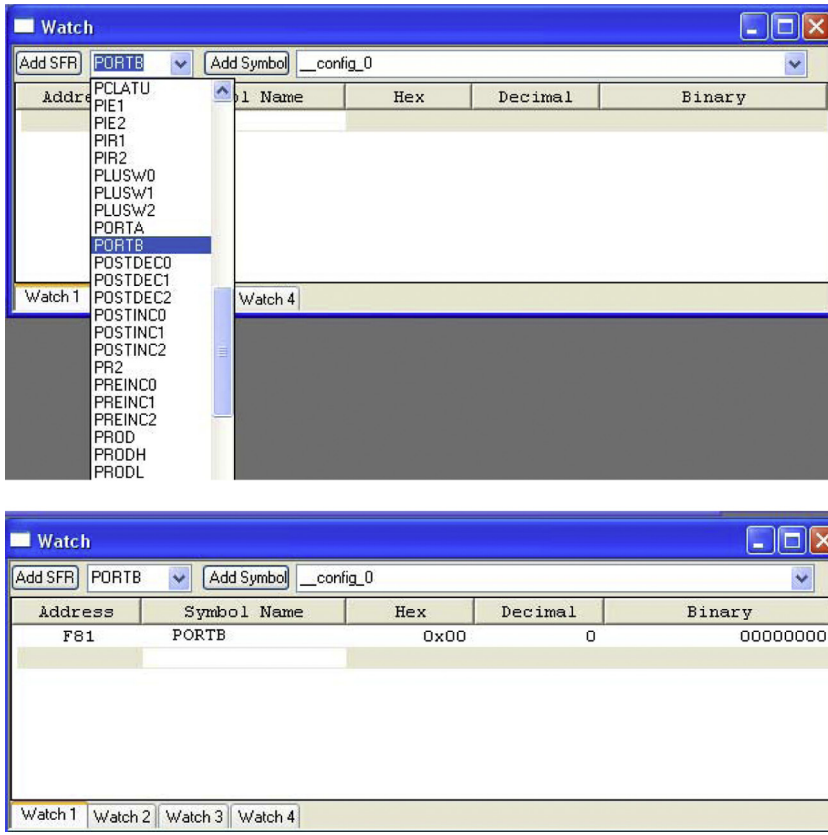


FIGURE 11.9 Adding PORTB to the Watch window.

The LED may not light because the program could be stuck in the 30 min delay routine or it could have been set incorrectly as a 300 min delay, keep waiting! Or it could be that something is wrong with turning the LED on.

Run the debugger by using the menu Debugger, Run, or press F9, or click the run buttons indicated in Figure 11.11.

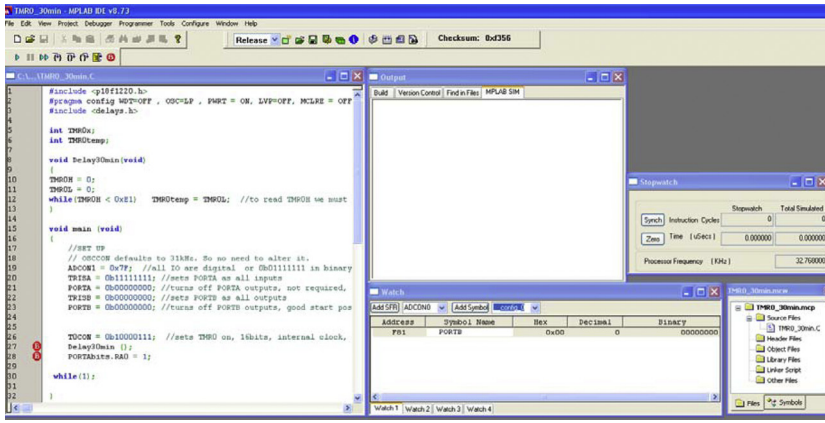


FIGURE 11.10 The TMR0_30 min workspace.

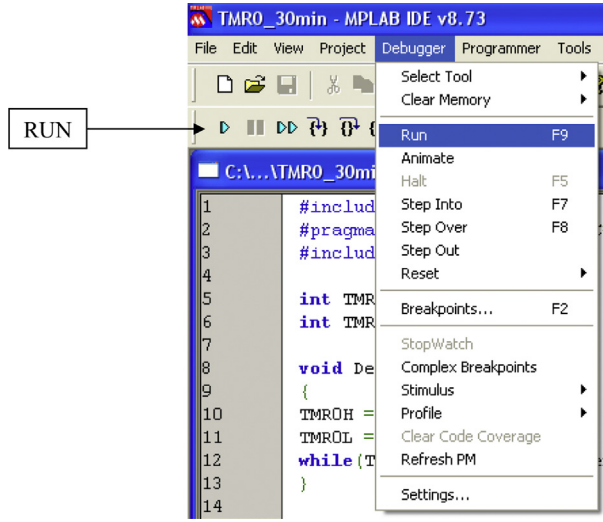


FIGURE 11.11 Running the debugger.

The program will stop at the first break point on line 27 shown with a green arrow. This, the start of the 30 min delay, is shown in Figure 11.12.

Now click the zero button in the stopwatch window, the result should resemble that shown in Figure 11.13.

Now click run again the program will run to the next break point which is on line 28. The program has just completed the Delay30 min() routine and the time taken is displayed in the stopwatch window (Figure 11.14).

Notice the time taken is 1800.002319 s which is 2.319 ms over 30 min (0.0013% accuracy). So we know our 30 min delay is correct. Not only that

```
23      PORTB = 0b00000000; //turns off PORTB outputs, good start pos
24
25
26      T0CON = 0b10000111;  //sets TMR0 on, 16bits, internal clock,
27      Delay30min ();
28      PORTAbits.RA0 = 1;
29
30    while(1);
31
32  }
```

FIGURE 11.12 Program halted at the break point.

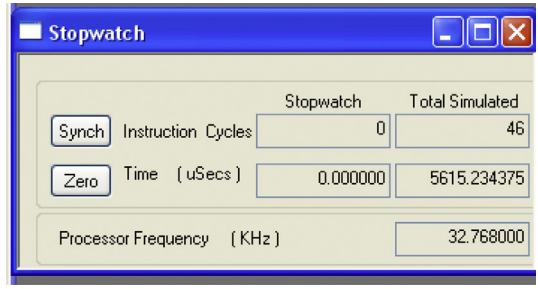


FIGURE 11.13 Clearing the Stopwatch.

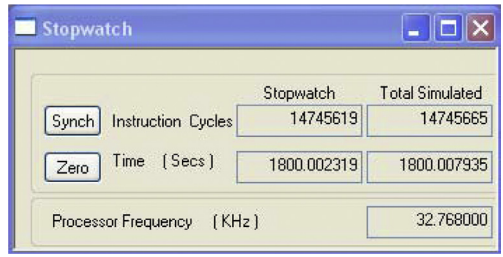


FIGURE 11.14 The Stopwatch timing Delay30min().

the simulation did not work in real time but also it only took 3 s to simulate the 30 min delay. So there was no waiting to see if the delay was correct.

In our program the error is not in the 30 min delay.

The program has halted on line 28. Now single step onto the next line of code, line30, as indicated in Figure 11.15 using the icon, the menu Debugger—Step Into, or the key F7.

The green arrow will have moved onto line 30 as shown in Figure 11.16.

Inspecting the watch window of Figure 11.17 has shown that PORTAbits RA0 has not been set.

There is nothing wrong with line 28 PORTAbits.RA0 = 1 (Figure 11.16). So there must be another reason why RA0 is not set. The only way RA0 is not set is if it has not been configured as an output.

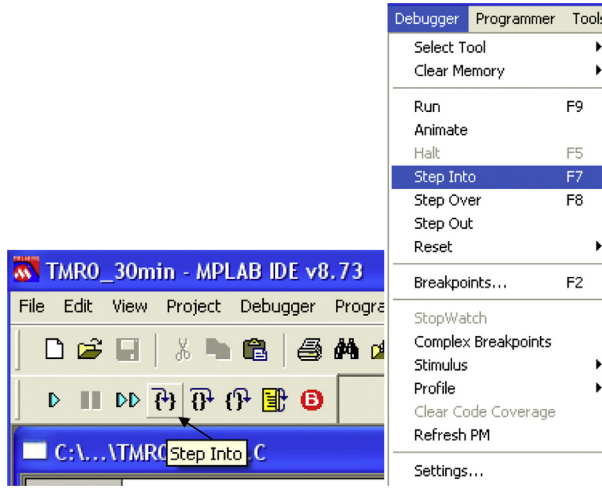


FIGURE 11.15 Step Into.

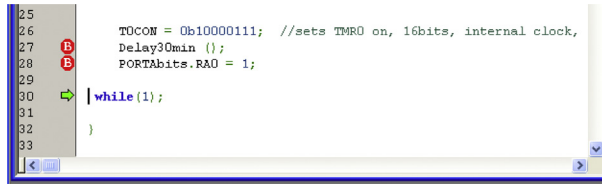


FIGURE 11.16 Single stepping through the code.

FIGURE 11.17 PORTA in the Watch window.

In the code the TRISA instruction

TRISA=0b11111111; //sets PORTA as all inputs

shows clearly that RA0 has been set as an input, not an output as required. Changing this line to:

TRISA=0b11111110; //sets PORTAbits.RA0 as output others as inputs

rectifies the problem and the code now works correctly.

Using Inputs

The simulator does not use hardware but we can simulate the operation of an input, switch, by setting it to a 0 or 1 or by toggling it (changing its state). This is done using the Stimulus window.

Let us consider the program seen in Chapter 4, inputs.C written below.

```
1.      // inputs.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.
6.      void main (void)
7.      {
8.              //SET UP
9.              // OSCCON defaults to 31 kHz. 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.             if (PORTAbits.RA0==0)     PORTB=0b00000001;
19.
20.             if (PORTAbits.RA1==0)     PORTB=0b00000011;
21.
22.             if (PORTAbits.RA2==0)     PORTB=0b00000111;
23.
24.             if (PORTAbits.RA3==0)     PORTB=0b00001111;
25.         }
26.     }
```

Open the project inputs.mcp if you have it or load the file inputs.c and make the project.

We do not require the stopwatch for this section but we do need to open the watch window with View, Watch.

Using Figure 11.17 add PORTA and PORTB in the Watch window which should appear as in Figure 11.18.

In order to add the Stimulus window click Debugger, Stimulus, New Workbook.

Figure 11.19 shows the Stimulus window with inputs RA0, RA1, RA2, and RA3 set by clicking the boxes the Pin/SFR column and selecting the relevant pins from the drop down list. The Action column is then clicked and Toggle selected in the relevant boxes.

Your screen should resemble Figure 11.20.

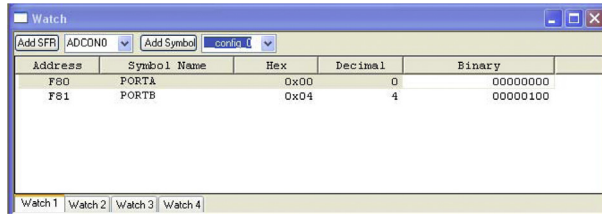


FIGURE 11.18 The Watch window sowing PORTA and PORTB being monitored.

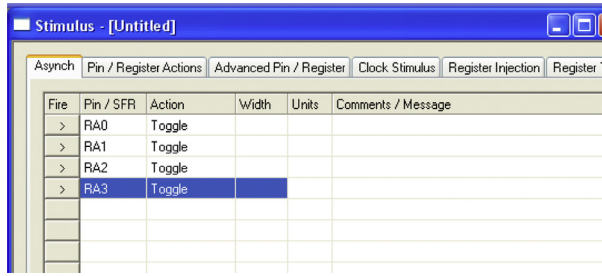
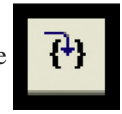


FIGURE 11.19 The Stimulus window.

There are three ways to use the simulator here:

- Put in break points and run the simulator using the  icon or F9.
- Or run an animation using the  icon.
- Or single step using the  icon or F7.

Using the animation and single stepping will update the registers on every line but running will only update when the program is stopped with the icon  or by using F5.

Try each method and see how the outputs change as you toggle the inputs.

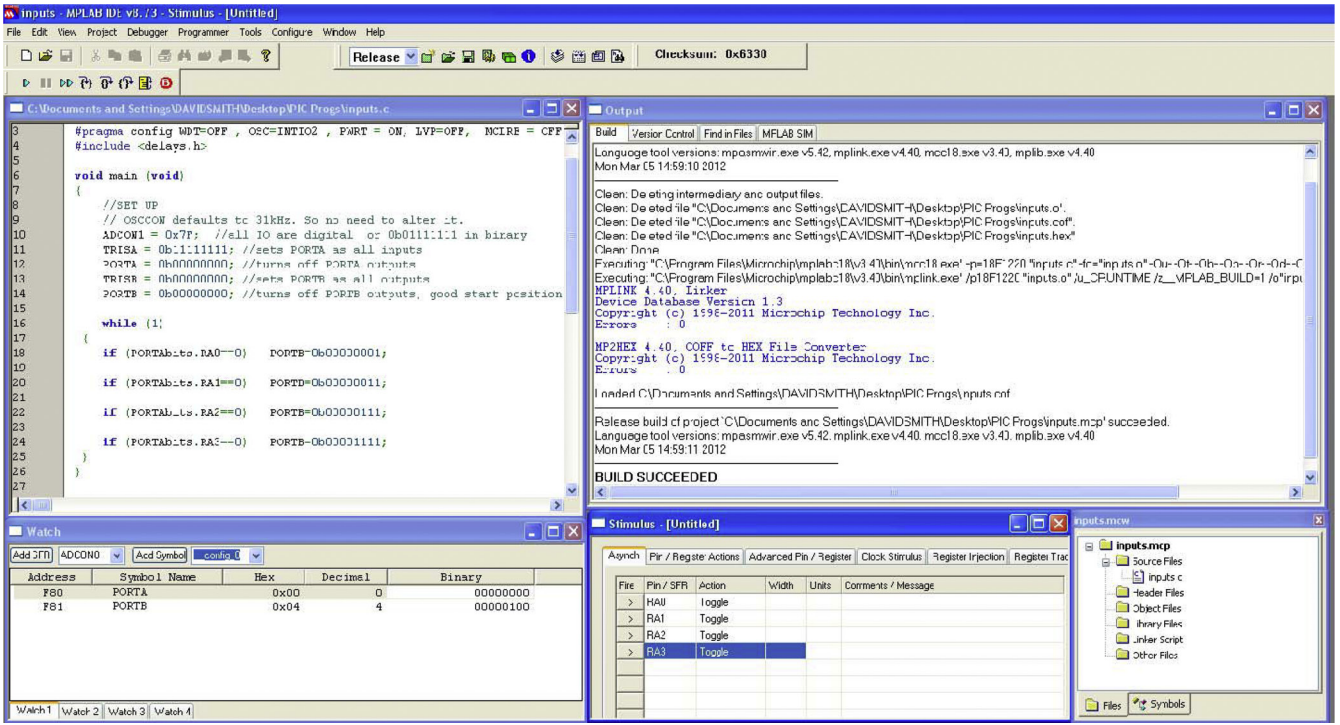


FIGURE 11.20 Screen display for **inputs.c**

Making Analogue Measurements

The simulator doesn't use any hardware so you cannot make an analogue measurement, but what you can do is, comment out taking the measurement and then put a value in the RESULT register and see if your code handles the result correctly.

Let us consider the program **Temperature.C** from Chapter 6.

```
1.  //Temperature.C DW Smith, 21st November 2011
2.
3.  #include <p18f1220.h>
4.  #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
5.  #include <delays.h>
6.
7.  int READING;
8.  float TEMPERATURE;          //Temperature is a decimal number not an integer.
9.
10. void main (void)
11. {
12. //SET UP
13. // OSCCON defaults to 31 kHz. So no need to alter it.
14. ADCON1=0x7E;               //AN0 is analogue or 0b01111110 in binary
15. TRISA=0b11111111;          //sets PORTA as all inputs, bit0 is AN0
16. PORTA=0b00000000;          //turns off PORTA outputs, not required, no outputs
17. TRISB=0b00000000;          //sets PORTB as all outputs
18. PORTB=0b00000000;          //turns off PORTB outputs, good start position
19. ADCON0bits.ADON=1;         //turn on A/D
20. ADCON2=0b10000000;         //right justified, acquisition times are ok at 0 with 31 kHz
21.
22. while (1)
23.    {
24. ADCON0bits.GO_DONE=1;              // do A/D measurement
25. while (ADCON0bits.GO_DONE== 1);     //wait until bit=0 for measurement
                                        completed
26. READING=ADRESL+(ADRESH * 256);
27. TEMPERATURE=READING/2.046−273.0;
28.
29. if (TEMPERATURE <= 0)       PORTB=0b00000011;          //buzzer and heater on
30. if (TEMPERATURE>0 && TEMPERATURE <=18)    PORTB=0b00000010;
                                                           //heater on
31. if (TEMPERATURE>18 && TEMPERATURE <= 25)   PORTB=0b00000000;
                                                           //no output required
32. if (TEMPERATURE>25 && TEMPERATURE <= 30)   PORTB=0b00001000;
                                                           //fan on
33. if (TEMPERATURE>30)             PORTB=0b00001100;          //fan and alarm on
34.    }
35. }
```

We will comment out the lines 24 and 25 so that we don't make an analogue measurement and rewrite line 26 with a made up value for the result as shown below:

```
24.     //ADCON0bits.GO_DONE=1;              // do A/D measurement
25.     //while (ADCON0bits.GO_DONE== 1);    //wait until bit=0 for measurement
                                             completed
26.     READING=599;      gives a temperature of 20
```

Put the modified Temperature.C into a project and build the project. Open a Watch window and view the PORTB, READING, and TEMPERATURE registers.

Your screen should be like Figure 11.21.

For clarity the code section and the Watch window have been highlighted in Figures 11.22 and 11.23.

Notice in the watch window of Figure 11.23 that PORTB is an 8 bit register, READING is an integer stored in a 16 bit register, and TEMPERATURE is a floating point number stored in a 32 bit register.

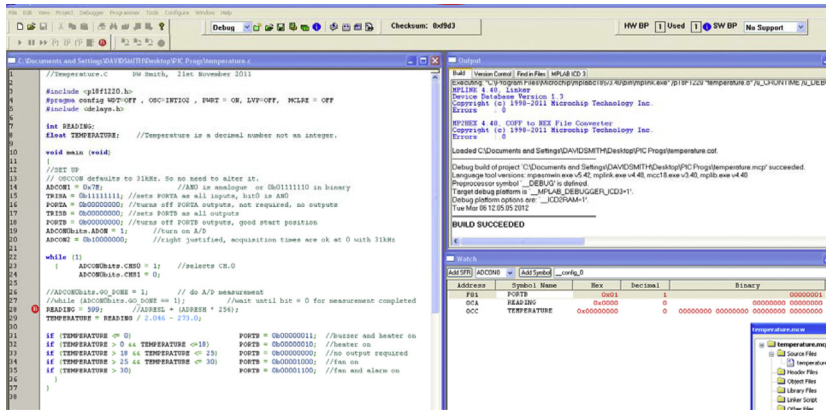


FIGURE 11.21 The modified Temperature.mcp project screen.

```
26      //ADCONUbits.GO_DONE = 1;        // do A/D measurement
27      //while (ADCONUbits.GO_DONE == 1);          //wait until bit = 0 for
28      READING = 599;        //ADRESL + (ADRESH * 256);
29      TEMPERATURE = READING / 2.046 - 273.0;
30
31      if (TEMPERATURE <= 0)                       PORTB = 0b00000011;
32      if (TEMPERATURE > 0 && TEMPERATURE <=18)    PORTB = 0b00000010;
33      if (TEMPERATURE > 18 && TEMPERATURE <= 25)  PORTB = 0b00000000;
34      if (TEMPERATURE > 25 && TEMPERATURE <= 30)  PORTB = 0b00001000;
35      if (TEMPERATURE > 30)                       PORTB = 0b00001100;
36        }
37      }
38
```

FIGURE 11.22 The Temperature.C code.

Put a break point in the code at line 31 as shown in Figure 11.22 and run the program to it. If you put the cursor over 'READING' in line 28 as shown in Figure 11.24 you will see:

The value of READING is displayed as 0×0257. Putting the cursor over TEMPERATURE on line 29 displays 19.76639, displayed to seven digits for a floating point number, as shown in Figure 11.25.

These results can also be seen in the Watch window as shown in Figure 11.26.

Notice on Figure 11.26 that the TEMPERATURE result is not 20! Actually it is, but it is written in the format of a floating point number which also displays amongst other things the exponent part of the number.

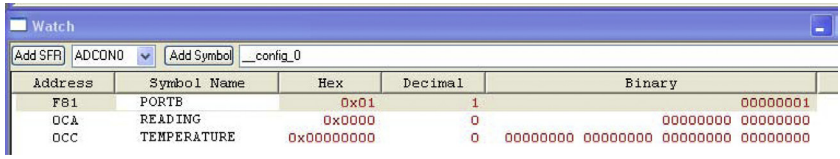


FIGURE 11.23 The Temperature Watch window.

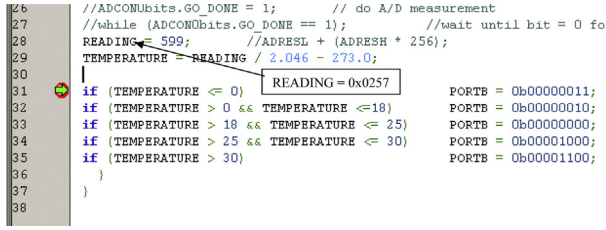


FIGURE 11.24 The simulator showing the READING result.

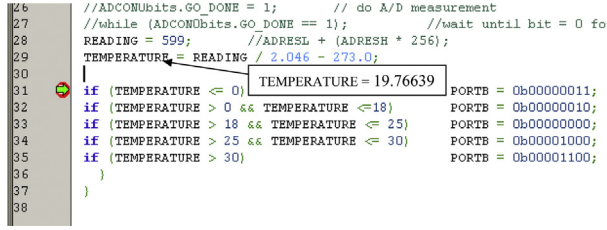


FIGURE 11.25 The simulator showing the TEMPERATURE result.

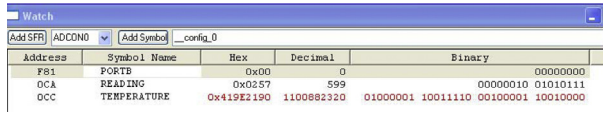


FIGURE 11.26 Watch window results.

You can now single step through your program and see the **if** statement:

```
if (TEMPERATURE>18 && TEMPERATURE <= 25) PORTB=0b00000000;
                                          //no output required
```

is being executed.

The In-Circuit Debugger

One of the simulator limitations is that it is a software simulation and doesn't connect to the real world—that is where the ICD comes in. The ICD3 is shown earlier in Fig. 2.20. You can now change the analogue input and see the results changing in the watch window while using break points, animating or single stepping. The hardware used here is the author's development kit shown in Figure 11.27 available from d.w.smith@mmu.ac.uk

FIGURE 11.27 A microcontroller development kit.

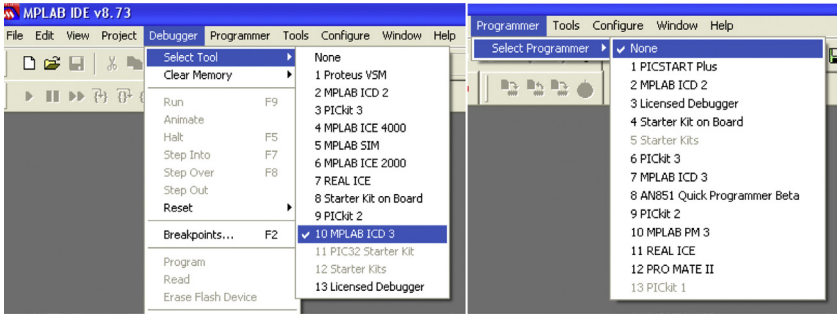


FIGURE 11.28 Selecting the ICD.

The debugger is selected with Debugger—Select Tool—MPLAB ICD 3. The programmer also requires setting to none, shown in Figure 11.28.

To use the ICD we not only program our code into the chip but we also need to program a block of code which allows the chip to communicate with the PC via the ICD. The debugging code is automatically entered when we program the device in the debugger mode.

NB. PORTAbits RA5, PORTBbits RB6 and RB7 are required to perform the debugging communication and cannot be used by the program.

Consider the **Temperature.C** program written below:

```
1.    //Temperature.C DW Smith, 21st November 2011
2.
3.    #include <p18f1220.h>
4.    #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
5.    #include <delays.h>
6.
7.    int READING;
8.    float TEMPERATURE;          //Temperature is a decimal number not an integer.
9.
10.   void main (void)
11.   {
12.   //SET UP
13.   // OSCCON defaults to 31 kHz. So no need to alter it.
14.   ADCON1=0x7E;              //AN0 is analogue or 0b01111110 in binary
15.   TRISA=0b11111111;         //sets PORTA as all inputs, bit0 is AN0
16.   PORTA=0b00000000;         //turns off PORTA outputs, not required, no outputs
17.   TRISB=0b00000000;         //sets PORTB as all outputs
18.   PORTB=0b00000000;         //turns off PORTB outputs, good start position
19.   ADCON0bits.ADON=1;        //turn on A/D
20.   ADCON2=0b10000000;        //right justified, acquisition times are ok at 0 with 31 kHz
21.
22.   while (1)
23.   {
24.   ADCON0bits.GO_DONE=1;           // do A/D measurement
25.   while (ADCON0bits.GO_DONE== 1);   //wait until bit=0 for measurement
                                        completed
26.   READING=ADRESL+(ADRESH * 256);
27.   TEMPERATURE=READING/ 2.046−273.0;
28.
29.   if (TEMPERATURE <= 0) PORTB=0b00000011;          //buzzer and heater on
30.   if (TEMPERATURE>0 && TEMPERATURE <=18) PORTB=0b00000010;
                                                       //heater on
31.   if (TEMPERATURE>18 && TEMPERATURE <= 25) PORTB=0b00000000;
                                                       //no output required
32.   if (TEMPERATURE>25 && TEMPERATURE <= 30) PORTB=0b00001000;
                                                       //fan on
33.   if (TEMPERATURE>30) PORTB=0b00001100;            //fan and alarm on
34.   }
35.   }
```

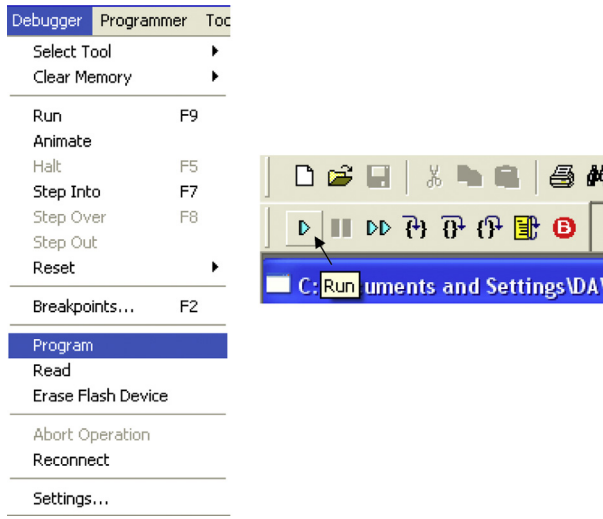


FIGURE 11.29 Programming and running the debugger.

Place a break point on line 29 so that the program halts after the READING has been taken and the TEMPERATURE calculated. Program and run the program as shown in Figure 11.29

By studying the Watch Window you can check that the temperature reading is correct and then single step through the code to see that the program selects the correct **if** statement to execute.

Using the Simulator with MPLABX

Before putting the microcontroller into our circuit we can run it through the simulator to see if it will perform as expected or to identify any faults.

To activate the Simulator: Run—Set Project Configuration—Customize Figure 11.30.

In the Project Properties box that pops up, Figure 11.31, select Simulator from the Hardware Tool Box, XC8 from the Compiler Toolchain and PIC18F1220 device—if not already selected. Click—Apply.

In the categories box, Figure 11.32 select Simulator.

The required oscillator frequency can then be entered and set by clicking—Apply—OK.

The simulator is used by setting break points in the code, so that the program will halt at the breakpoint. The registers can then be viewed in the Watch window.

To open the watch window from the Debug menu, click New Watch Figure 11.33.

The New Watch dialogue box will appear, Figure 11.34, and the registers can then be selected

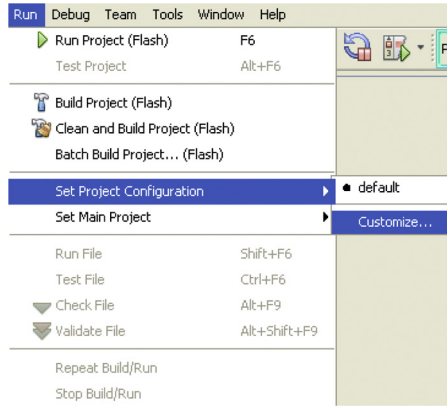


FIGURE 11.30 Activating the simulator.

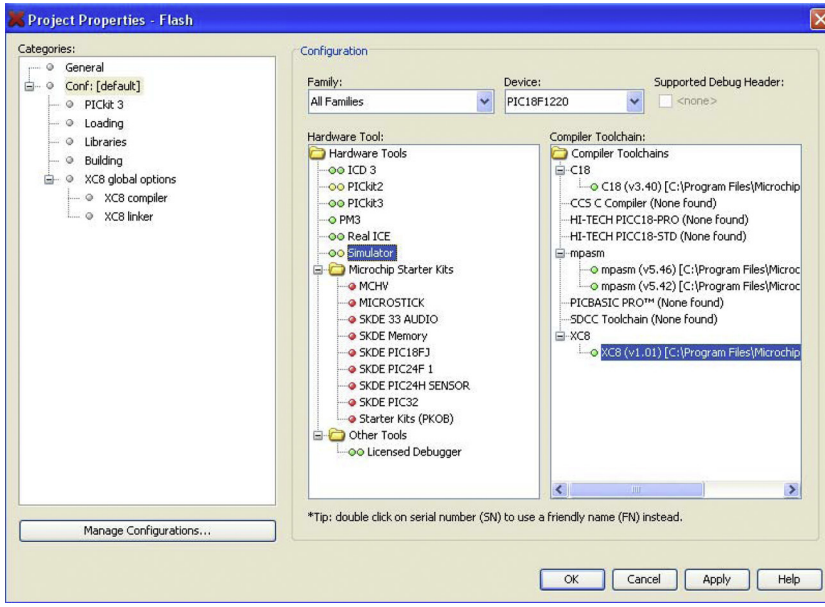
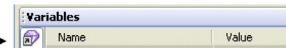


FIGURE 11.31 Project Properties Box.

The first register can be selected. The Watch window, listed as Variables, will then open and further registers can be selected by right clicking on the <Enter new watch> row. Figure 11.35

To change the variable's parameters ie. Name, Value etc.

Right click on the parameter row →



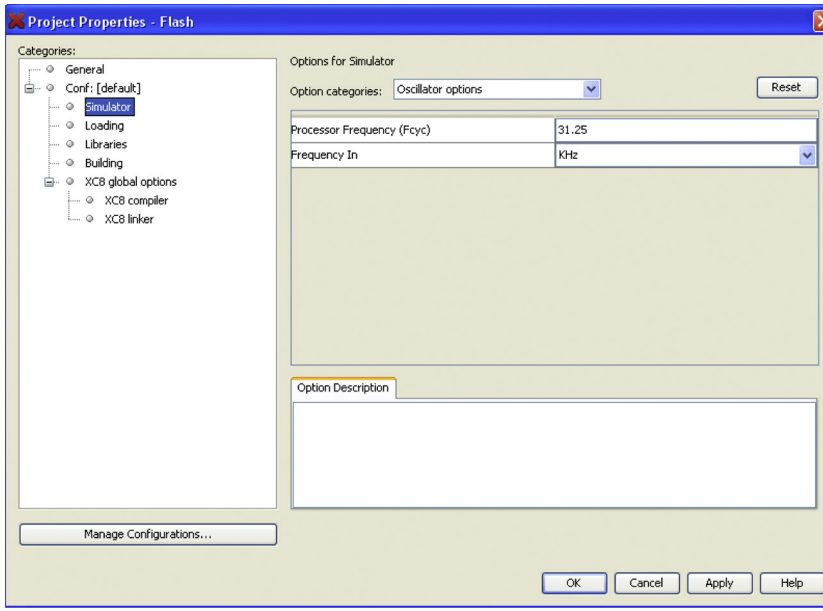


FIGURE 11.32 Selecting the Simulator.

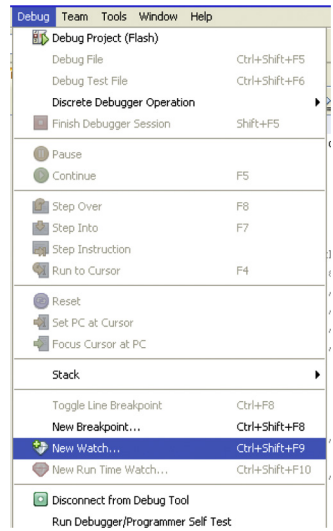


FIGURE 11.33 Opening the Watch Window.

FIGURE 11.34 Selecting Registers for the Watch Window.

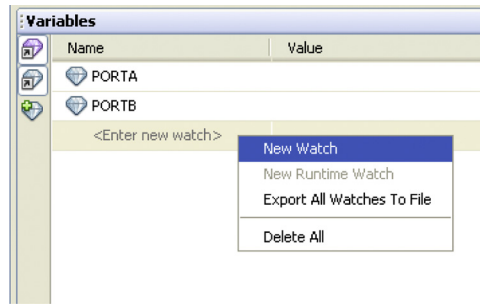


FIGURE 11.35 Opening a New Watch Window.

The Change Visible Columns dialogue box will open up, Figure 11.36, where binary and decimal values can be chosen to view in the watch window.

SETTING BREAKPOINTS

To set breakpoints: Click the line number in the code, a grey rectangle will take the place of the line number indicating a breakpoint, shown on line 13 in Figure 11.37. Clicking the rectangle will turn the breakpoint off. (The line numbers can be turned on and off by right clicking the line numbers column.)

The grey rectangle signifies a disabled breakpoint. To enable the breakpoint, right click on it and click Disabled Breakpoint—Enabled, shown in Figure 11.38. The enabled breakpoint turns orange.

The Simulator is switched on using Debug—Debug Main Project, Figure 11.39, this will enable the debug toolbar, Figure 11.40, which controls the debugging actions.

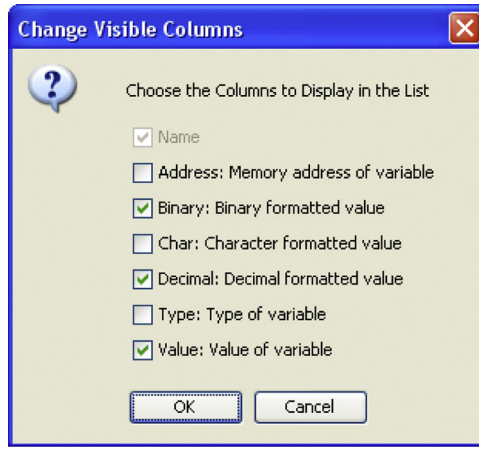


FIGURE 11.36 Selecting Parameters in the Watch Window.

```
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
12          TRISB = 0b00000000;     //sets PORTB as all outputs
13          PORTB = 0b00000000;     //turns off PORTB outputs, good start posit
14
15
16          while (1)
17          {
18          PORTB = 0xFF;           // turn on PORTB
19          Delay100TCYx(78);               // wait 1s
20          PORTB = 0;              // turn off PORTB
21          Delay100TCYx(78);               // wait 1s
22          }
23  }
24
```

FIGURE 11.37 Setting Breakpoints.

STOPWATCH

The stopwatch is enabled from Window—Debugging—Stopwatch, Figure 11.41

The stopwatch is used to time program execution between 2 breakpoints. At the moment MPLABX IDE v1.30 stopwatch will not permit oscillator frequencies to be altered. It does display the number of timing cycles. I find it useful to check my delays are correct. Eg. In flash.c to check Delay100TCYx(78) on line 19 is a 1 second delay, put the curser on line19 and run to curser. Then step over to line 20. Figure 11.42

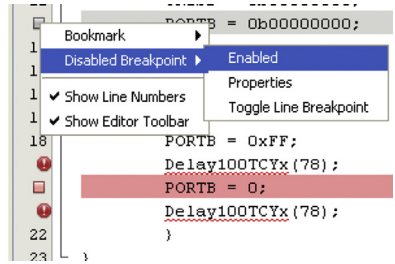


FIGURE 11.38 Enabling Breakpoints.

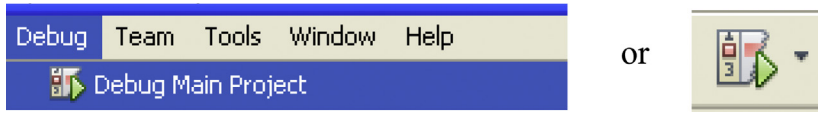


FIGURE 11.39 Switching on the Simulator.

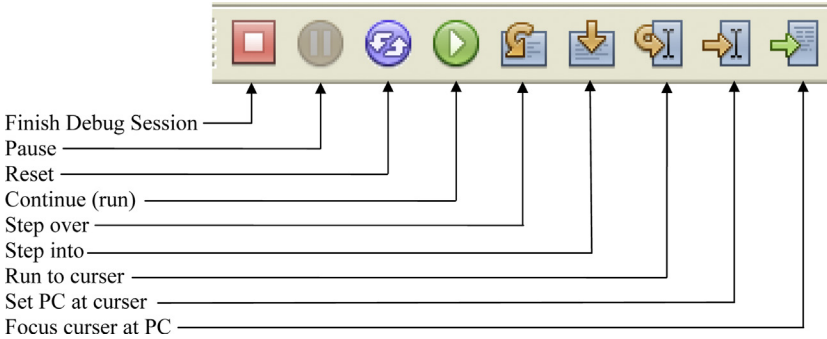


FIGURE 11.40 The Debug Toolbar.

The stopwatch displays 8039 cycles. At a frequency of 31.25 kHz. a timing cycle is 0.128 ms. So $8039 \times 0.128\,\text{ms} = 1.029\,\text{s}$. There are a couple of inconsistencies with this stopwatch reading:

- The time is incorrect because the stopwatch was set at a frequency of 1 MHz and could not be altered.
- The cycle count should have been 7800, (100×78) giving a time of 0.9984 s

But this does give a reasonable indication of the value, if not the exact one. Connect the microcontroller to your circuit and observe the LED flashing. The following photographs show

- a header board and ZIF socket with Microcontroller connected to the ICD3, Figure 11.43

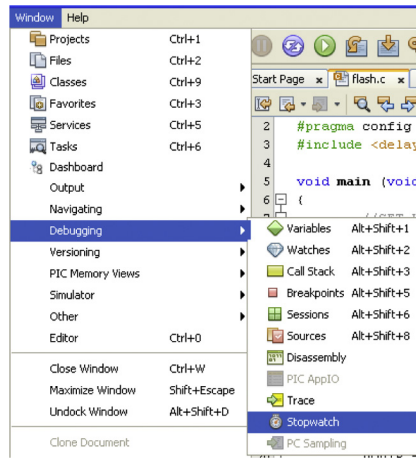


FIGURE 11.41 Enabling the Stopwatch.

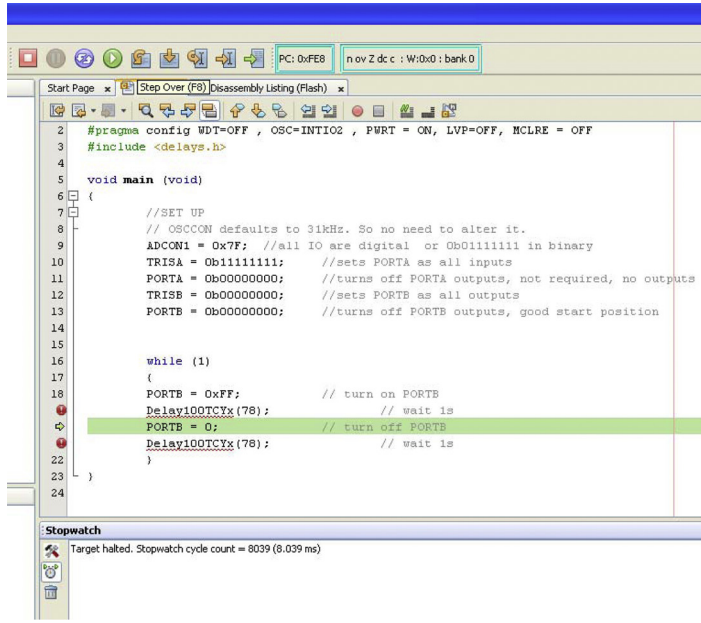


FIGURE 11.42 Using the stopwatch.

- a header board and leads to connect the PICkit3 to the development board, Figure 11.44.
- The PICkit3 connected to the development kit via the header, Figure 11.45

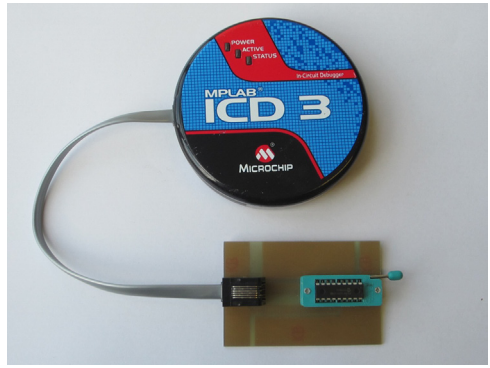


FIGURE 11.43 Header board and ZIF socket with Microcontroller connected to the ICD3.

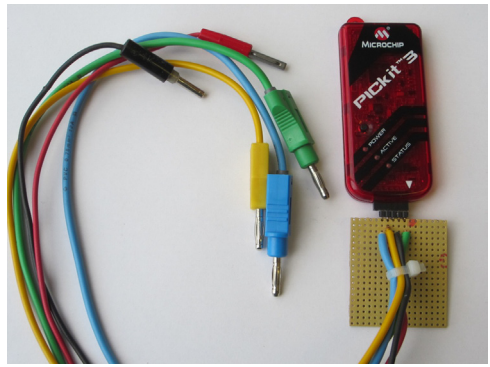


FIGURE 11.44 Header board and leads to connect the PICkit3 to the development board.

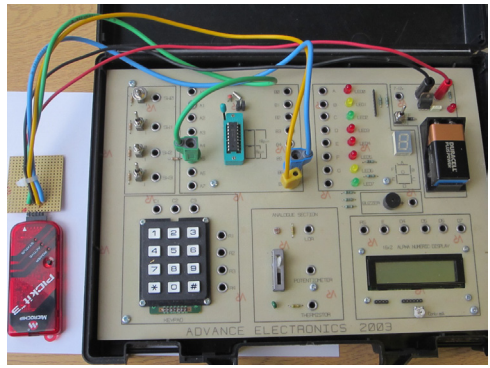


FIGURE 11.45 The PICkit3 connected to the development kit via the header.

FURTHER INFORMATION

For further information on the use of the PICkit3 and the ICD3 please refer to Microchip's

- MPLAB User Guide
- ICD3 User Guide
- PICkit3 User Guide