COMP551 – Interfacing

Fall 2017

Lab 3

Light an LED Using the MPLAB Simulator

Students:		
Student ID's:		
Section:	**01** **02** **03** **04**	

NOTE: Labs are due at the start of the next lab period. Only submit one lab per group of two students.

Lab 3 – Light an LED Using the MPLAB Simulator

3.1 PROGRAM: LIGHT an LED USING the SIMULATOR

The last lab demonstrated the basics of creating, building and testing a project using MPLAB C18 with the MPLAB IDE. It did not go into the details of what the target processor would do with that code. In this next program, code will be generated to simulate turning on a Light Emitting Diode (LED) connected to a pin of the PIC18F458.

3.1.1 Create a New Project

Create a new project named "**Light LED**" in a new folder named "**Lab 3**". Make sure the language tools are set up and the Build Options are properly configured as shown in Section 3.7 "Verify Installation and Build Options" of the Project Basics handout.

3.1.2 Write the Source Code

Create a new file and type in the code shown in Example 3-1. Save it with the name main.c in the "Lab 3" folder:

EXAMPLE 3-1: LIGHT LED CODE

```
#include <p18cxxx.h>            //Could also use <P18F458.h>
#pragma config WDT = OFF

void main (void)

{
        TRISB = 0;              //Set all Port B pins as outputs

        PORTB = 0;              //Reset the LED's

        PORTB = 0x5A;           //Light the LED's

        while (1);              //Loop forever
}
```

- The first line in this code includes the generic processor header file for all PIC18XXXX devices, named p18cxxx.h. This file chooses the appropriate header file as selected in MPLAB IDE; in this case, the file named P18F458.h (which could have been included explicitly instead). This file contains the definitions for the Special Function Registers in these devices.

- "#pragma config WDT = OFF" controls the Watchdog Timer of the target microcontroller, disabling it so it won't interfere with these programs.

> **Note:** In MPLAB C18, the main function is declared as returning void, since embedded applications do not return to another operating system or function.

- This example will use four pins on the 8-bit I/O port with the register name PORTB. "TRISB = 0" sets the PIC18F458 register named TRISB to a value of zero. The TRIS registers control the direction of the I/O pins on the ports. Port pins can be either inputs or outputs. Setting them all to zero will make all eight pins function as outputs.

> **Note:** A simple way to remember how to configure the TRIS registers is that a bit set to '0' will be an output. Zero ('0') is like the letter "O" (O = 0). Bits set to a '1' will be inputs. The number one ('1') is like the letter "I" (I = 1).

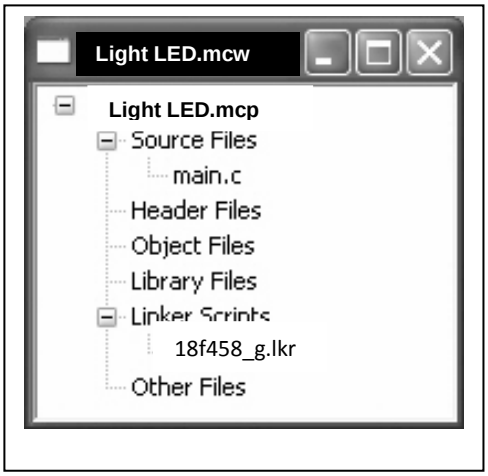
- "PORTB = 0" sets all eight pins of the PORTB register to '0' or to a low voltage.

- "PORTB = 0x5A" sets four pins on PORTB to '1' or to a high voltage (0x5A = 0b01011010).

When this program is executed on a PIC18F458, an LED properly connected to one or all of the pins that went high will turn on (more on that in a future lab).

Add main.c as a source file to the project. Select the 18F458_g.lkr file as the linker script for the project. The project window should look like Figure 3-1:

FIGURE 3-1: LIGHT LED PROJECT



3.1.3 Build the Program

Build the project with **Project>Build All**. If there are errors, check language tool locations and build options, or see section 2.1.5 of Lab 2, Resolving Problems.

3.1.4 Testing the Light LED Program

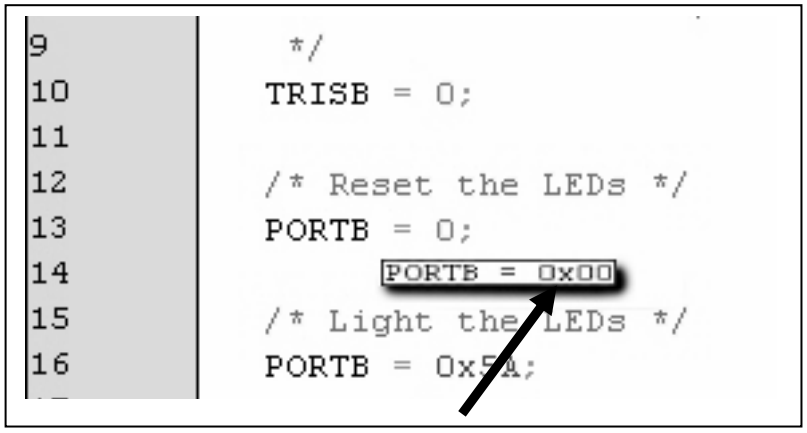
Like in the first programs, the simulator in MPLAB IDE will be used to test this code. Make sure that the simulator is enabled. The project may need to be built again if the simulator was not already selected.

To test the code, the state of the pins on PORTB must be monitored. In MPLAB IDE, there are three methods you can use to do this. The first two methods will be discussed next. The third method will be discussed near the end of the lab, section 3.3.1 Using the Logic Analyzer.

Method 1: Using the Mouse over Variable

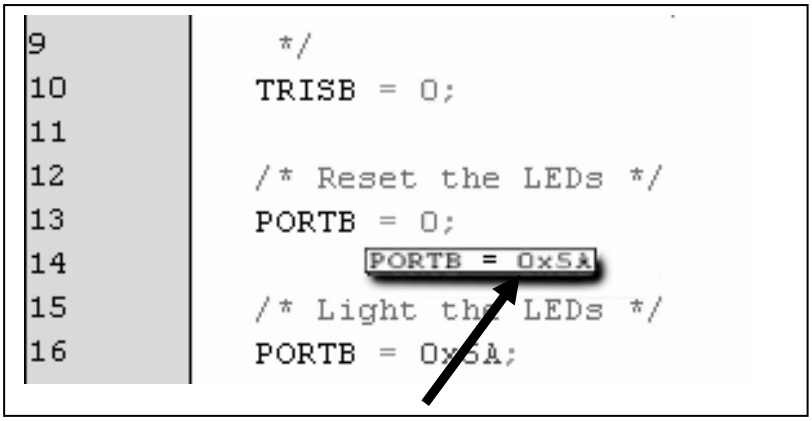
After the project is successfully built, use the mouse to place the text editor cursor over a variable name in the editor window to show the current value of that variable. Before this program is run, a mouse over PORTB should show its value of zero or 0x00 (see Figure 3-2):

FIGURE 3-2: MOUSE OVER PORTB BEFORE PROGRAM EXECUTION



Click the Run icon (or select Debug>Run), then click the Halt icon and do the mouse over again. The value should now be 0x5A (Figure 3-3):

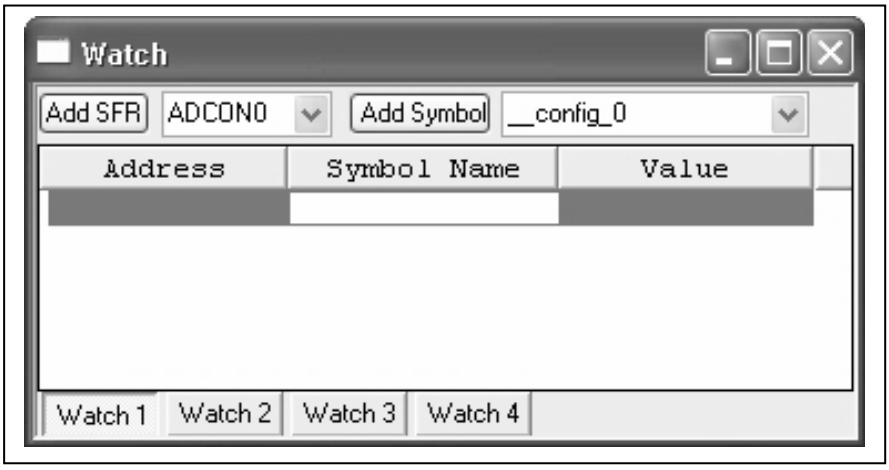
FIGURE 3-3: MOUSE OVER PORTB AFTER PROGRAM EXECUTION



Method 2: Using the Watch Window

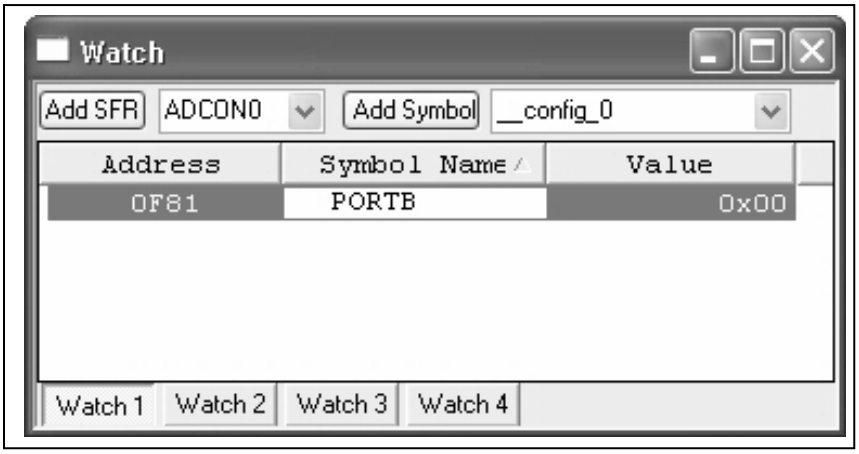
The second method you can use to check the value of a variable is to put it into a Watch window. Select ***View>Watch*** to bring up a new Watch window (Figure 3-4).

FIGURE 3-4: NEW WATCH WINDOW



Drag the Watch window away from the source file window so that it is not on top of any part of it. Highlight the word PORTB in main.c. When the word is highlighted, drag it to the empty area of the Watch window. The Watch window now looks like Figure 3-5.

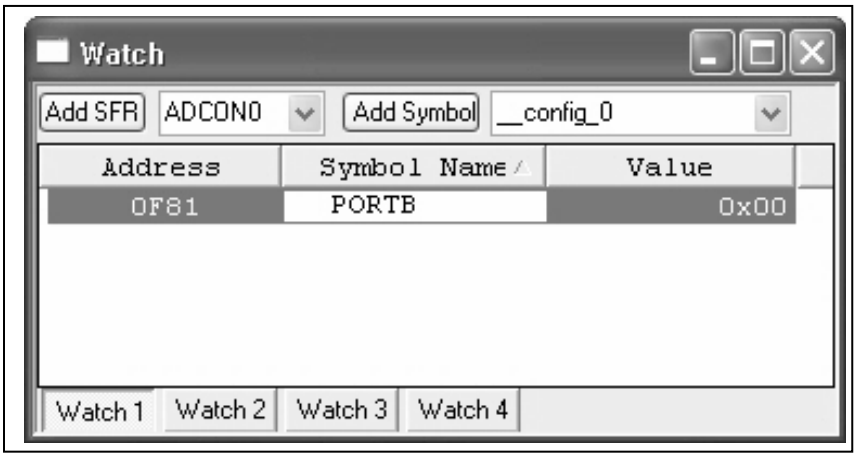
FIGURE 3-5: WATCH WINDOW FOR PORTB



Note: If the value of PORTB shows 0x5A, the program was executed previously. Double click on the value in the Watch window and type zero to clear it.

Select the **Run** icon, then after a few seconds, select the **Halt** icon. The Watch window should show a value of 0x5A in PORTB (see Figure 3-6).

FIGURE 3-6: WATCH WINDOW AFTER PROGRAM EXECUTION



Double click on the 0x5A value of PORTB in the Watch window to highlight it, type any other 8-bit value. Select **Reset**, **Run**, wait a few seconds and then press **Halt** to see the value return to 0x5A.

3.2 PROGRAM: FLASH the LED USING the SIMULATOR

3.2.1 Modify the Source Code

This program will build on the last program to flash the LED's on PORTB. The program will be modified to run in a loop to set the pins high and low, alternately. Modify the code from the previous program to look like Example 3-2:

EXAMPLE 3-2: FLASH LED CODE

```
#include <p18cxxx.h>
#pragma config WDT = OFF

void main (void)
{
        TRISB = 0;              //Set all Port B pins as outputs

        while(1)                //Loop forever

        {

                PORTB = 0;              //Reset the LED's

                PORTB = 0x5A;           //Light the LED's
        }
}
```

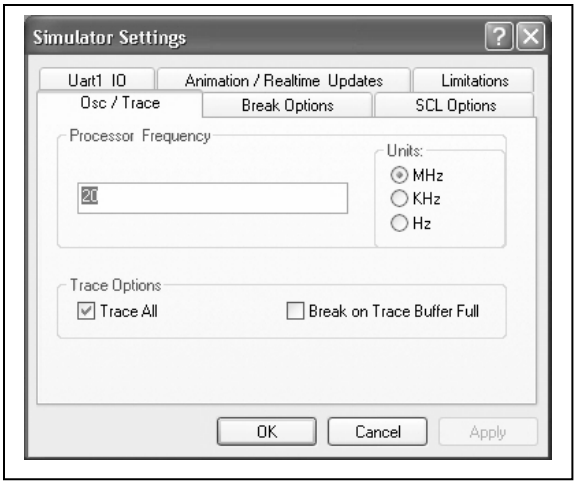
Now the code within the infinite while() loop sets and resets the pins of PORTB continuously. Will this produce the effect of flashing LED's? Yes and no.

PIC18F458 instructions execute very fast, typically in less than a microsecond, depending upon the clock speed. The LED's are probably turning off and on, but very, very fast – maybe too fast for the human eye to perceive them as flashing. The simulator has control over the processor's clock frequency.

Select ***Debugger>Settings*** to display the Simulator Settings dialog (Figure 3-7):

The problem with changing the processor's clock frequency, is that the program will execute slower. In most applications the microcontroller is dealing with data in near real-time, so slowing down the program is not very practical in the 'real world, only in simulations.

FIGURE 3-7: SIMULATOR SETTINGS

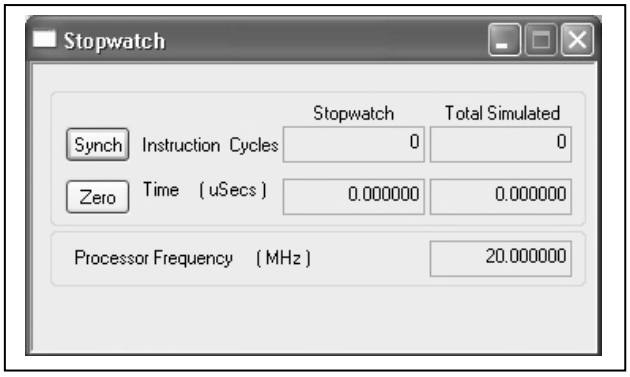


3.2.2 Select the Stopwatch

On the **Osc/Trace** tab, the default setting for the Processor Frequency is 20 MHz. If it does not show 20 MHz, change it to match the settings in Figure 3-7. Then click **OK**.

The time between the pins going high and low can be measured with the MPLAB Stopwatch. Select ***Debugger>Stopwatch*** to view the MPLAB Stopwatch (Figure 3-8).

FIGURE 3-8: STOPWATCH

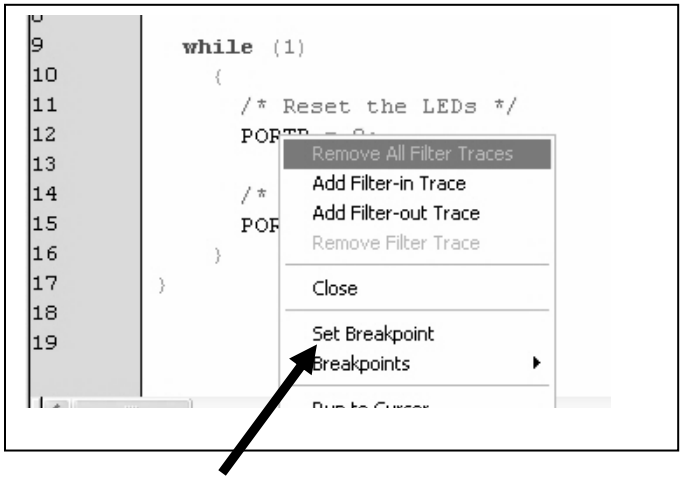


The Stopwatch also shows the current processor frequency setting of 20 MHz. To measure the time between the LED's flashing off and on, breakpoints will be set at the appropriate places in the code. Use the right mouse button to set breakpoints.

Note: If the Stopwatch cannot be selected from the pull down menu, the simulator may not be set up (***Debugger>Select Tool>MPLAB SIM***).

Click on the first instance of the word PORTB, line 12 (assuming you have the same number of lines in your code) and press the right mouse button. The debug menu will appear as shown in Figure 3-9.

FIGURE 3-9: RIGHT MOUSE MENU

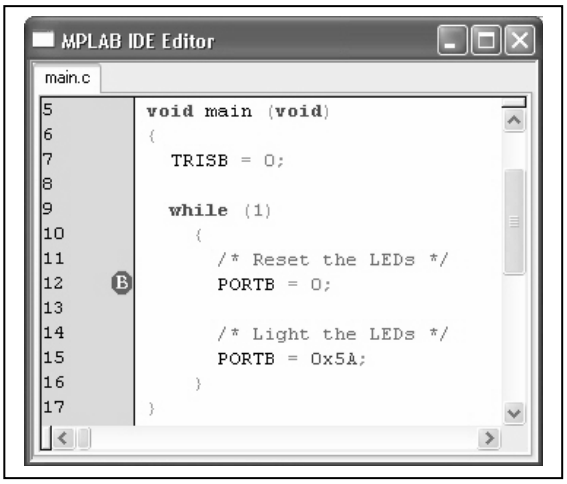


3.2.3 Set Breakpoints

Select “Set Breakpoint” from the menu, and the screen should now show a breakpoint on this line, signified by a red icon with a “**B**” in the left gutter (see Figure 3-10).

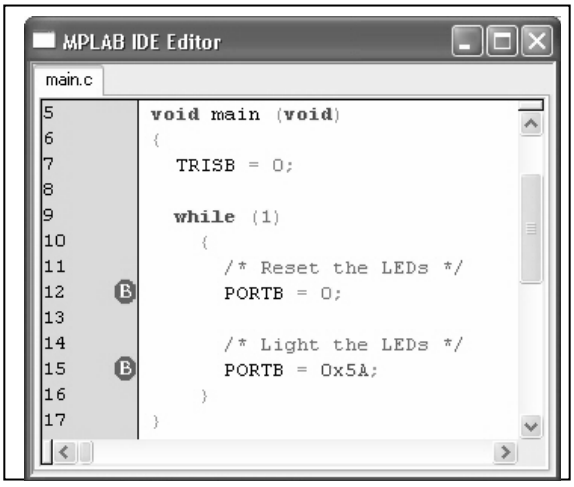
Note: If a breakpoint cannot be set, it may be because the project has not been built. Select ***Project>Build All*** and try to set the breakpoint again.

FIGURE 3-10: BREAKPOINT



Put a second breakpoint on line 15, the line to send a value of 0x5A to PORTB. The Editor window should have two breakpoints and look similar to Figure 3-11:

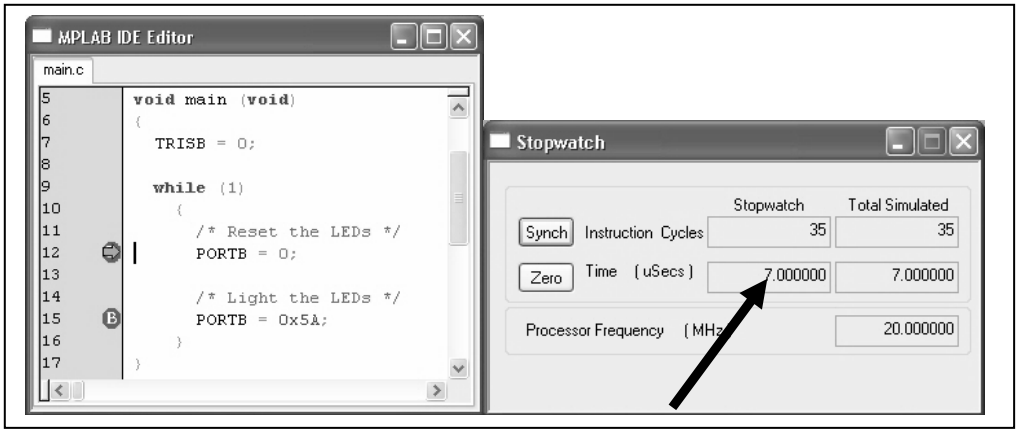
FIGURE 3-11: SECOND BREAKPOINT



3.2.4 Run the Program

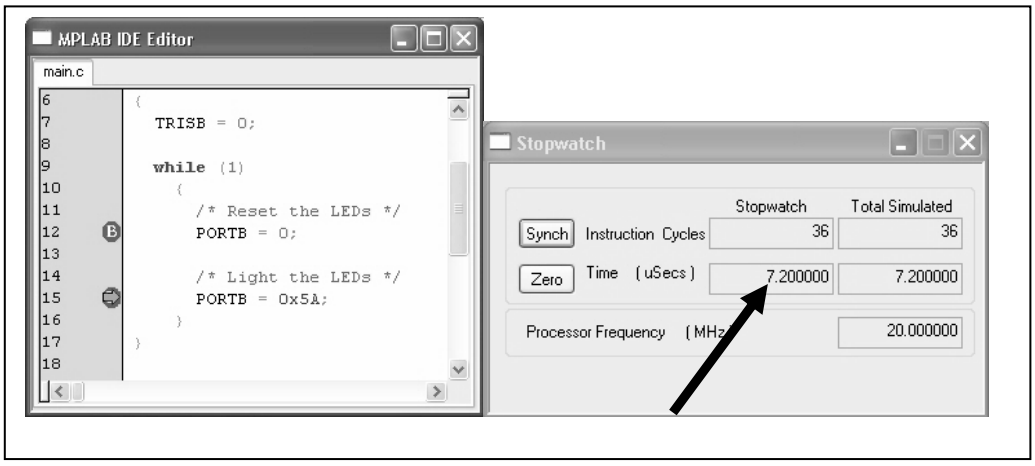
Select the **Run** icon and the program should execute and then stop at a breakpoint indicated by a green arrow on the first breakpoint. Note that the Stopwatch has measured how much time it has taken to get to this point (see Figure 3-12).

FIGURE 3-12: RUN TO FIRST BREAKPOINT



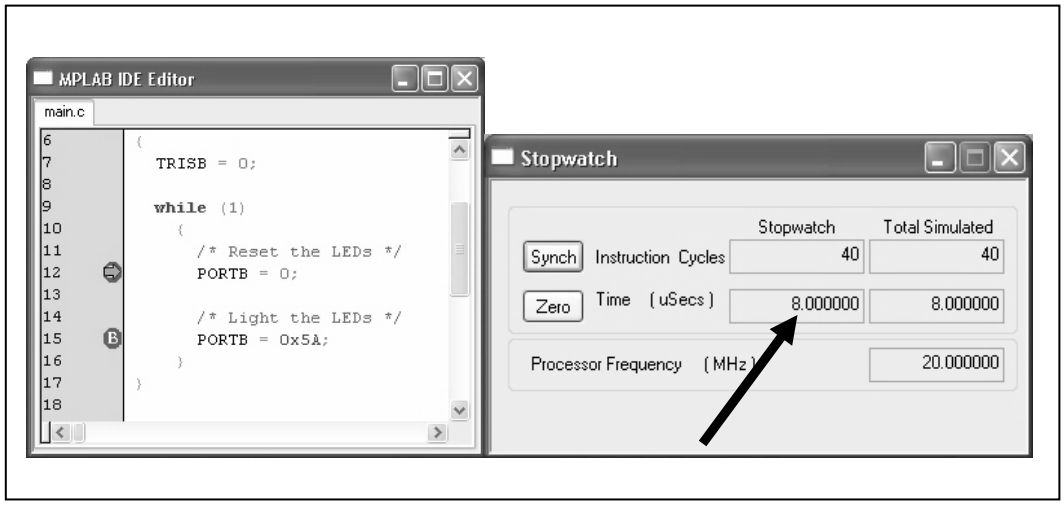
The Stopwatch reading is 7.000000 microseconds, indicating it took seven microseconds to start from reset to run to this point in the program. Select Run again to run to the second breakpoint (Figure 3-13):

FIGURE 3-13: RUN TO SECOND BREAKPOINT



The Stopwatch now reads 7.200000 microseconds, indicating it took 0.2 microseconds to get here from the last breakpoint. Select Run again to go around the loop back to the first breakpoint (Figure 3-14):

FIGURE 3-14: LOOP BACK TO FIRST BREAKPOINT



3.2.5 Analyze the Program

The question posed earlier can now be answered. The Stopwatch is reading 8.000000 microseconds, so it took 8.0 – 7.2 = 0.8 microseconds to get around the loop. If the LED's are flashing on and off, faster than once per microsecond, they are flashing too fast for the human eye to see. To make the LED's flash at a perceptible rate, either the processor frequency must be decreased or some time delays must be added.

If all the application needed to do is flash these LED's, the processor frequency could be reduced, as mentioned earlier. Doing that would make all code run very slowly, and any code added to do anything more than flash the LEDs would also run slowly. A better solution is to add a delay.

3.2.6 Add a Delay

A delay can be a simple routine that decrements a variable many times. For this program, a delay can be written as Example 3-3:

EXAMPLE 3-3: DELAY ROUTINE

```
void delay (void)
{
        int i;
        for (i = 0; i < 100; i++);
}
```

Add this to the code to main.c and insert a call to this function after the LED's are turned off and again after they are turned on.

3.2.7 Build the Program

Once again, select ***Project>Build All*** to rebuild everything after these changes are made to the source code, and add breakpoints where PORTB is written. Use the Stopwatch to measure the code. The previously set breakpoints may also show up at different places in the code. Use the right mouse menu to Remove Breakpoint, leaving just the two desired breakpoints.

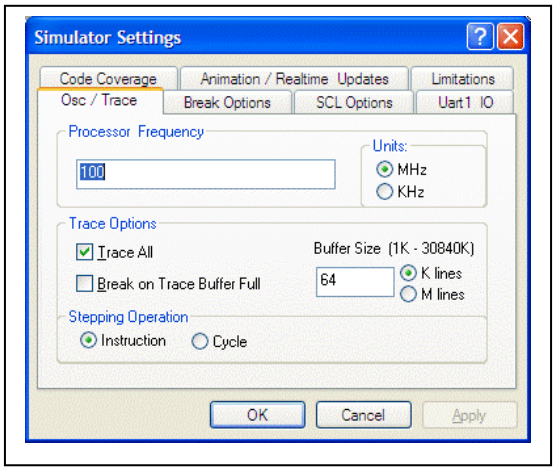
Measure the time between breakpoints again. After stopping at the first breakpoint, press the **Zero** button on the Stopwatch to start measuring from this breakpoint. With the variable ***i*** counting down from 10000, the measured time is about 36 milliseconds. Remember that ***i*** is defined as an ***int***, which has a range of -32,768 to 32,767. The largest value (32,767) will make the delay about 3 times longer. If ***i*** is declared as ***unsigned int***, its range can be extended to 65,535. When set to this value, the measured delay is about 301 milliseconds, meaning that with both delays, it takes about 602 milliseconds to go around the loop. This is just over a half second, so the lights will be flashing about twice a second. Modify your program to verify that using an unsigned integer value, set at its maximum value of 65,535, will produce a delay of about 301 milliseconds.

3.3.1 Using the Logic Analyzer

The MPLAB SIM Logic Analyzer gives you a graphical and extremely effective view of the recorded values for any number of the device output pins, but it requires a little care in the initial setup.

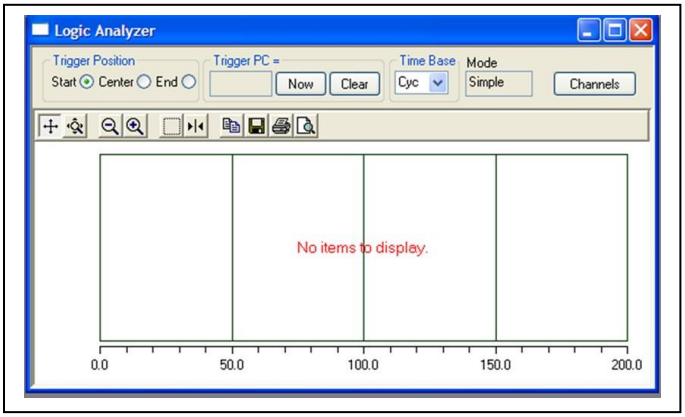
Make sure the ***Tracing*** function of the simulator is turned on: Select ***Debugger>Settings*** from the menu, then choose the **Osc/Trace** tab. In the Tracing options section, check the Trace All box (Figure 3-15).

FIGURE 3-15: SIMULATOR SETTINGS



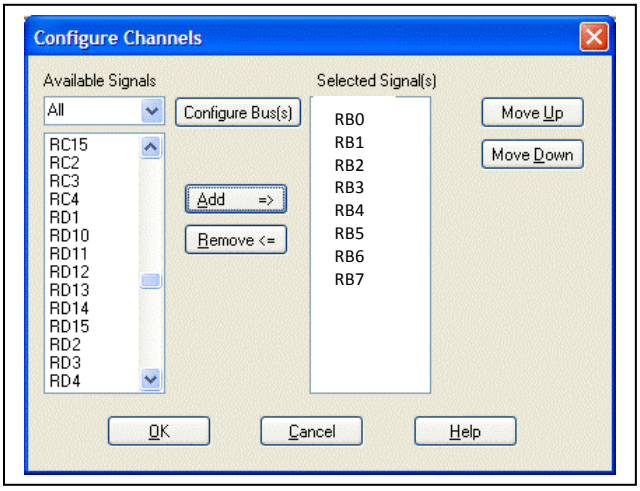
Now you can open the Analyzer window by selecting ***View>Simulator Logic Analyzer*** from the top menu (Figure 3-16).

FIGURE 3-16: SIMULATOR LOGIC ANALYZER



Click the **Channels** button to bring up the channel selection dialog box. Select all the output pins (RB0-RB7) of PORTB and click Add (Figure 3-17).

FIGURE 3-17: CHANNEL SELECTION



Click **OK** to close the selection dialog box.

You can run the simulation by pressing the **Run** button on the Debugger toolbar, selecting the ***Debugger>Run*** menu item, or pressing the ***F9*** shortcut key. Keep in mind that you still have breakpoints set in your program, so the program will only execute up to the first breakpoint.

Remove the breakpoints and run your program again. After a short while, press the **Halt** button on the Debugger toolbar, or select the ***Debugger>Halt*** menu item, or press the **F5** shortcut key.

The logic analyzer should display its results.

You may get better results by pressing the **Animate** button on the Debugger toolbar, or select the ***Debugger>Animate*** menu item. The logic analyzer will now update continuously (as long as there are no breakpoints). You can stop the display whenever you have collected enough information.

Use the Logic Analyzer to see if your delay loop is working correctly. Compare the results to the results obtained by the Stopwatch.

Exercises:

1. Write a program to send the values FFH to 00H, to Port B. Use the simulator to see how Port B displays the values FF-00H in binary.

2. Write a program to read from memory, then send hex values for the ASCII characters 0, 1, 2, 3, 4, 5, A, B, C, D to Port B. The characters are stored in RAM using the following command;

   char *'variable name'*[]= "012345ABCD";

   Run the program in the simulator to see how Port B displays the hex values of each ASCII character.

3. Write a program to send values of +5 to -5 to Port C. The values are to be stored in memory using the following command;

   char '*variable name*'[] = {+5, +4, +3, +2, +1, 0, -1, -2, -3, -4, -5};

   Run the program in the simulator to see how Port C displays the hex values of number.

   Note: For information on ***"Representing Negative Numbers on a PIC",*** refer to page 18 of this lab.

4. Write a program to toggle all bits of Port B, 20,000 times. Start with 0x5A as the output. Run the program in the simulator to verify the code.

5. Modify the program in exercise 4, to include a 1 second delay after each time the bits are toggled. Make the delay a function.

Submit a hardcopy of the source code for each program, as well as a Logic Analyzer screenshot demonstrating the proper output of each program.

Note: Use a marker to label the binary and hex values on the output waveforms.

Representing Negative Numbers on a PIC

- D7 (MSB) is 1
- Magnitude is represented in 2's complement form

To convert to negative number representation (2's complement), follow these steps;

1. Write the number in 8-bit binary (no sign).
2. Invert each bit.
3. Add 1 to it.

Example

How would the PIC represent -5?

Solution

1. 0000 0101 // 5 in 8-bit binary
2. 1111 1010 // Invert each bit
3. 1111 1011 // Add 1, which becomes FBH