**NUST SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE**

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| Faculty Member:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Lab Engineer: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Semester/Section:\_\_\_\_\_\_\_\_\_\_\_\_ |

Department of Electrical Engineering

EE- 222: Microprocessor Systems

**LAB 11: For-loops, while-loops, if-then branching, subroutines, and time delays**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student’s Name | Reg. # | Lab Conduct and Report | Viva | Total |
| 10 | 5 | 15 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

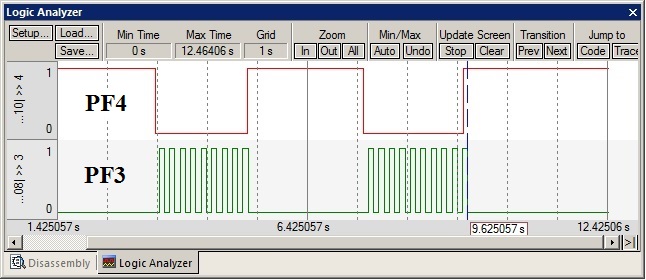
# **Purpose**

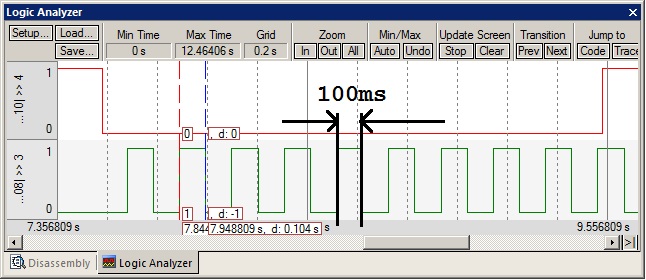
The purpose of this lab is to learn simple programming structures in assembly. You will also learn how to estimate how long it takes to run software, and use this estimation to create a time delay function. You learn how to use the oscilloscope to measure time delay. Assembly software skills you will learn include masking, toggling, if-then, subroutines, and looping.

# **System Requirements**

The system has one negative logic input switch and one positive logic output LED. Figure 2.1 shows the system when simulated as the switch is touched. A negative logic switch means the **PF4** signal will be 1 (high, 3.3V) if the switch is not pressed, and the **PF4** signal will be 0 (low, +0V) if the switch is pressed. A positive logic green LED interface means if the software outputs a 1 to **PF3** (high, +3.3V) the green LED will turn ON, and if the software outputs a 0 to **PF3** (low, 0V) the green LED will be OFF. In Lab 3, you will attach a real switch and and real LED to your protoboard, and interface them to your microcontroller. In Lab 2 you first debug in simulation and then run on the real board, but no external hardware will be required. Overall functionality of this system is described in these rules.

1. The switch at PF4 will be an input (with PUR enabled) and the LED at PF3 will be the output
2. The system starts with the LED OFF (make **PF3** =0).
3. When the switch is pressed, the LED will toggle **PF3** once per 100ms
4. When the switch is not pressed the LED will remain off





*Figure 2.1. Example screenshots in simulation mode with the switch pressed then released. Delta=104ms.*

Time is very important to embedded systems. One of the simplest ways in which we manage time is by determining how long it takes to run our software. One method we use to measure time in our embedded systems is to measure the time each instruction takes to execute. There are two ways to determine how long each instruction takes to execute.

The first method uses the ARM data sheet. For example, the following is a page from the Cortex-M4 Technical Reference Manual. E.g., see pages 34-38 of

[**http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM4\_TRM\_r0p1.pdf**](http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM4_TRM_r0p1.pdf)

On the TM4C123 the default bus clock is 16 MHz ±1%. Starting in Lab 4 we will activate the phase lock loop (PLL) and the bus clock will be exactly 80 MHz. For now, however, we will run at about 16 MHz. The following is a portion of a listing file (dissassembly) with a simple delay loop. The SUBS and BNE instructions are executed 800 times. The SUBS takes 1 cycle and the BNE takes 1 to 4 (a branch takes 0 to 3 cycles to refill the pipeline). The minimum time to execute this code is 800\*(1+1)/16 μs = 100 μs. The maximum time to execute this code is 800\*(1+4)/16 μs = 250 μs. Since it is impossible to get an accurate time value using the cycle counting method, we will need another way to estimate execution speed.

**0x00000158 F44F7016 MOV R0,#800**

**0x0000015C 3801 wait SUBS R0,R0,#0x01**

**0x0000015E D1FD BNE wait**

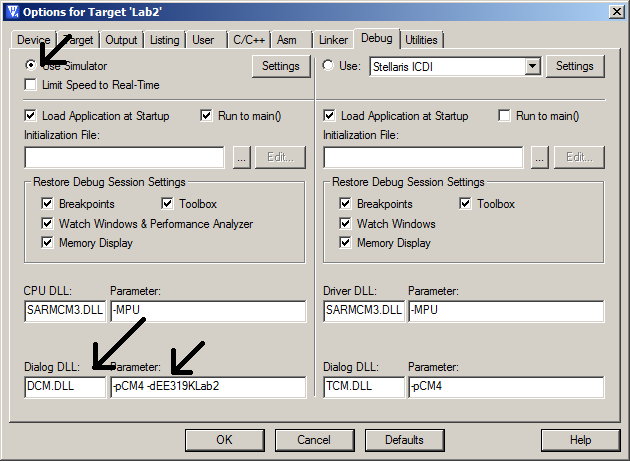
*(note: the* ***BNE*** *instruction executes in 3 cycles on the simulator, but in 2 cycles on the real board)*

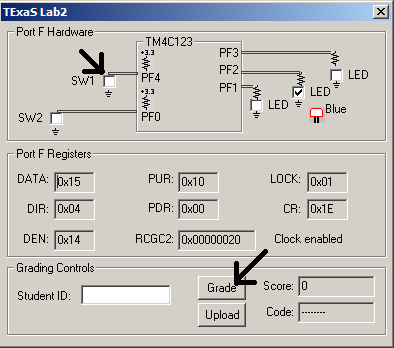
An accurate method to measure time uses a logic analyzer or oscilloscope. In the simulator, we will use a simulated logic analyzer, and on the real board we will use an oscilloscope. To measure execution time, we cause rising and falling edges on a digital output pin that occur at known places within the software execution. We can use the logic analyzer or oscilloscope to measure the elapsed time between the rising and falling edges. In this lab we will measure the time between edges on output PF3.

# **Procedure**

The basic approach to Lab 2 will be to develop and debug your system using the simulator using a negative logic switch (PF4) and a positive logic LED (PF3). In Lab 3, you will build and test an actual switch and LED. This lab will run on the LaunchPad using SW1 on PF4 and the green LED on PF3.

To run the Lab 2 simulator, you must do two things. First, execute Project->Options and select the Debug tab. The debug parameter field must include **-dEE319KLab2**. Second, the **EE319KLab2.dll** file must be added to your Keil\ARM\BIN folder.





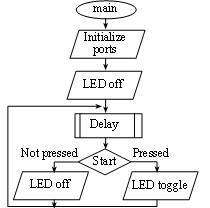
*Figure 2.2. Using TExaS to debug your software in simulation mode (DCM.DLL -pCM4 -dEE319KLab2).*

# **Part a - Design a Delay Subroutine**

Design a subroutine that delays about 100 ms. First, draw a flowchart, and then write pseudocode (both are deliverables). You will call the subroutine with a **BL** instruction and return with a **BX LR** instruction. Any delay from **80 to 120** ms is ok. To implement a delay, you could set a register to a large number, and then count it down to zero. With a bus clock of 16 MHz, there are 16,000 bus clock cycles in 1 ms. You need to know how long it takes to execute the loop once, then determine the number of times you need to execute the loop to create the 1 ms delay. E.g., if the time to execute the loop once is 4 cycles, then executing the loop 4000 times will be about 1 ms. In this simple estimation we are neglecting the instructions outside of the loop, because they are 4000 times less important.

# **Part b - Write a Main Program**

Write a main program in assembly that implements the input/output system. Pseudo code and flowchart are shown in Figures 2.3 and 2.4, illustrating the basic steps for the system. We recommend at this early stage of your design career you access the entire I/O port using GPIO\_PORTF\_DATA\_R. After you fully understand how I/O works, then you can use bit-specific addressing to access I/O ports.



*Figure 2.3: Flowchart of the system*

**main** Turn on the clock for Port F

Set the Port F direction register so

**PF4** is an input and

**PF3** is an output

Enable the **PF4** and **PF3** bits in the Port F DEN register

Set bit 4 in Port F PUR register so it will run on the real board

Set **PF3** so the LED is OFF

**loop** Delay about 100ms

Read the switch and test if the switch is pressed

If **PF4**=0 (the switch is pressed),

toggle **PF3** (flip bit from 0 to 1, or from 1 to 0)

If **PF4**=1 (the switch is not pressed),

clear **PF3** so LED is OFF

Go to **loop**

# *Figure 2.4: Pseudo code of the system*

# 

# **Part c - Test in Simulation**

Test the program in simulation mode. Use the built-in logic analyzer to verify the LED is toggling at the rate at which it was designed. In particular, capture two screenshots like Figure 2.1 showing when the switch is pressed, the LED is ON for 100 ms and OFF for 100 ms.