# Lab 2. For-loops, while-loops, if-then branching, subroutines, and time delays



## Preparation

Read Sections 2.7, 3.3.8 and 4.6 of the book (plus the ones listed in Lab 1)
Download, unzip, open, compile, and run the starter project, which has debugging windows configured

| | |
|---|---|
| **Lab2_EE319K_asm** | Starter project with links to grading engine |
| **Lab2_EE319K_C** | Starter project used for HW |

## 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

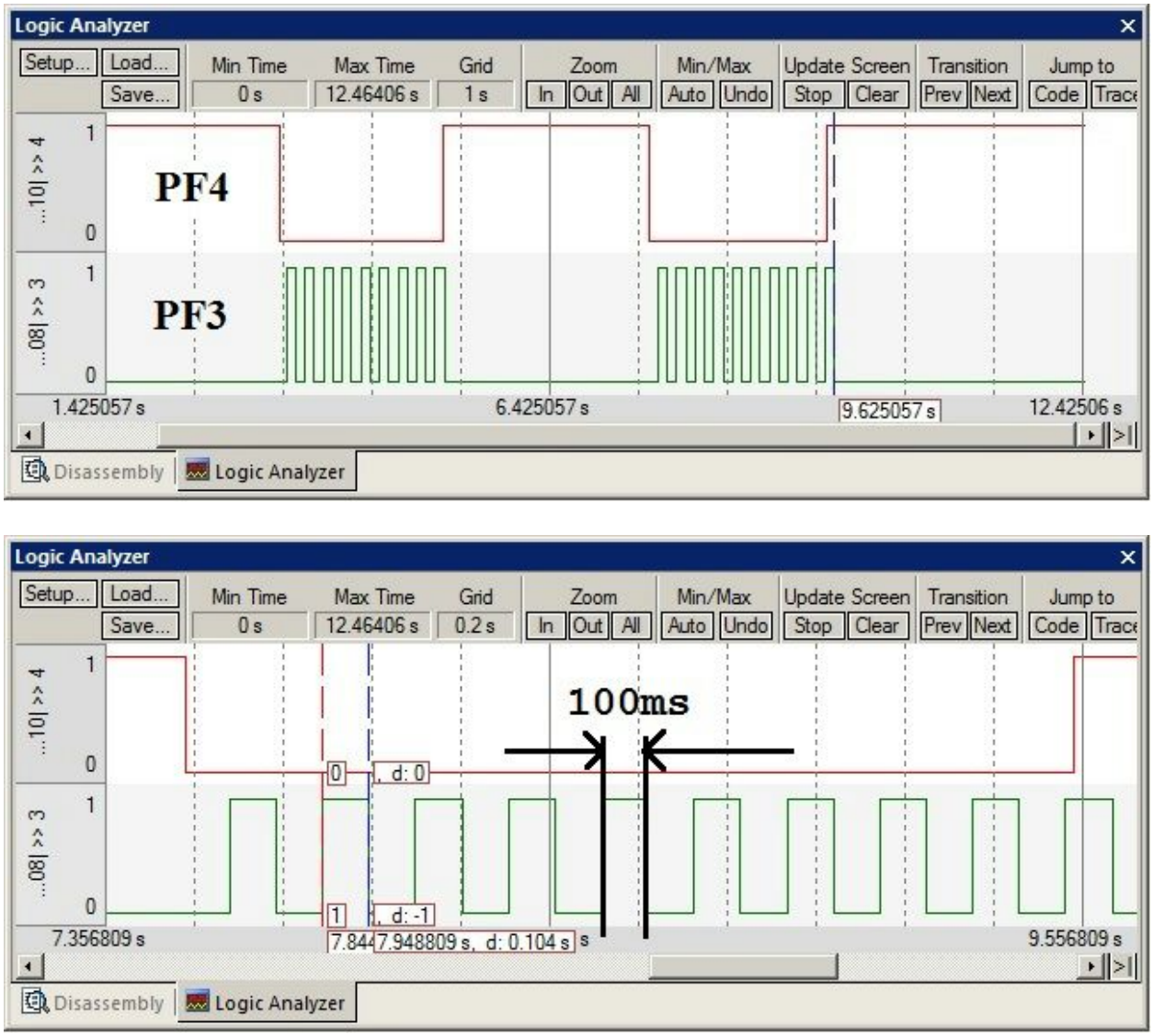



*Figure 2.1. Example screenshots in simulation mode with the switch pressed then released. Delta=104ms.*

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**

| | | | |
|---|---|---|---|
| Load | Word | LDR Rd, [Rn, <op2>] | $2^b$ |
| | To PC | LDR PC, [Rn, <op2>] | $2^b + P$ |
| | Halfword | LDRH Rd, [Rn, <op2>] | $2^b$ |
| | Byte | LDRB Rd, [Rn, <op2>] | $2^b$ |
| | Signed halfword | LDRSH Rd, [Rn, <op2>] | $2^b$ |
| | Signed byte | LDRSB Rd, [Rn, <op2>] | $2^b$ |
| | PC relative | LDR Rd,[PC, #<imm>] | $2^b$ |

*Figure 2.2. From the Technical Reference Manual page 36.*

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.

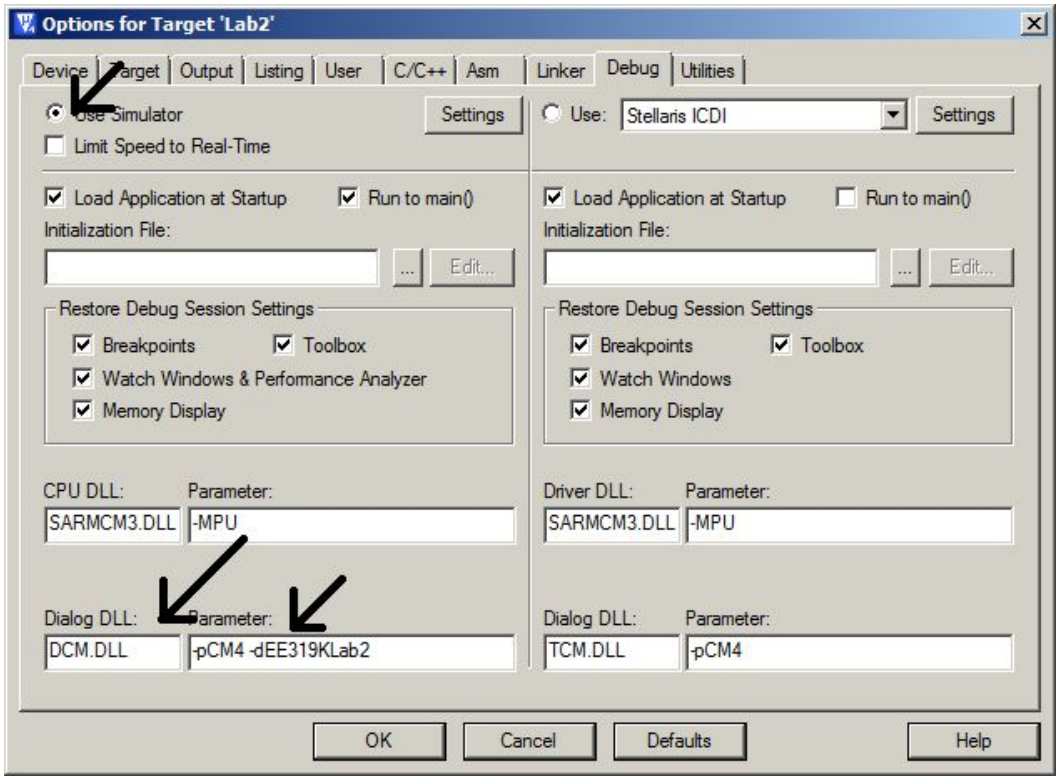
```
0x00000158 F44F7016        MOV R0,#800
0x0000015C 3801     wait   SUBS R0,R0,#0x01
0x0000015E D1FD            BNE  wait
```
*(note: the **BNE** instruction executes in 3 cycles on the simulator, but in 2 cycles on the real board)*

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.



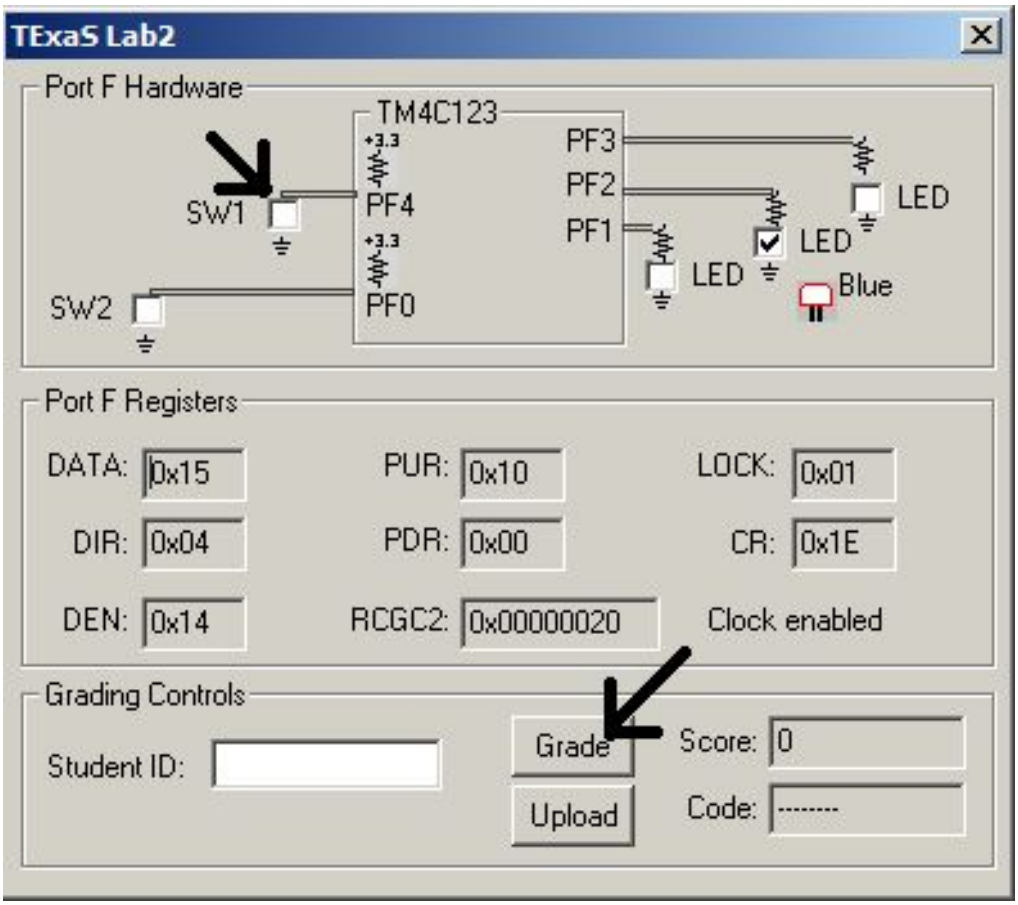




*Figure 2.3. Using TExaS to debug your software in simulation mode (DCM.DLL -pCM4 -dEE319KLab2).*

## 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 2.4 and 2.5, 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.

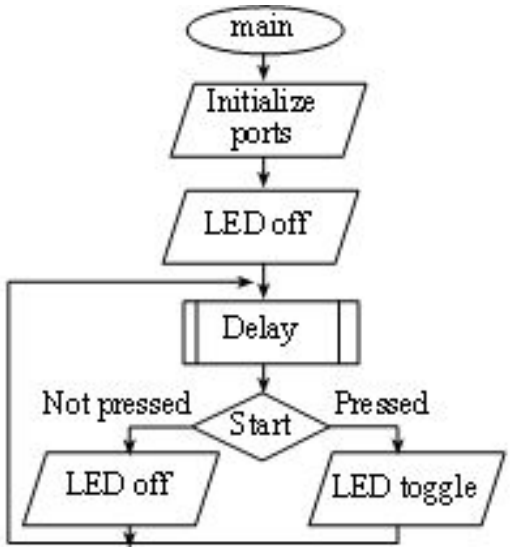



*Figure 2.4: Flowchart of the system*

**`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 2.5: 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.

## Part d - Simulation Timing

Simulators typically run slower than real hardware. Use the built in logic analyzer to measure how much Cortex-M time is simulated in 10 human seconds. Hit the reset twice to clear the time axis on the plot. Run the simulator for 10 human seconds (real time measured with your watch), and then stop simulation. Observe the logic analyzer time to determine the amount of time Cortex-M simulated. For example, my computer simulated 15 sec of Cortex-M time in 10 sec of human time, meaning the simulator was running 50% faster (1.5x) compared to a real Cortex-M. There are many factors that affect this ratio, so expect to see this ratio vary. The point of the exercise is to get a sense of how a simulator manages time. Discuss this time with other students in the class.

## Part e - Test in Hardware

Load your software onto the board and test it again. If the PF4 switch is not pressed, then the green LED is off, which you can see by noticing its bright color. If the PF4 switch is pressed, then the green LED toggles 5 times a second, which you can see by noticing the green LED oscillate.

## Demonstration

(both partners must be present, and demonstration grades for partners may be different)

You will show the TA your program operation on both the simulator and the real board. Be prepared to explain how the delay function works. How would it be different if it were 1 ms instead of 100 ms? The TA will pick an instruction from your code and ask you which addressing mode it uses. Execute the program step-by-step and run to cursor. What is a Reset Vector and why do you need it? What does **`AREA`** do? What does **`EXPORT`** do?

## Deliverables

(Items 2-6 are one pdf file uploaded to Canvas, have this file open during demo.)

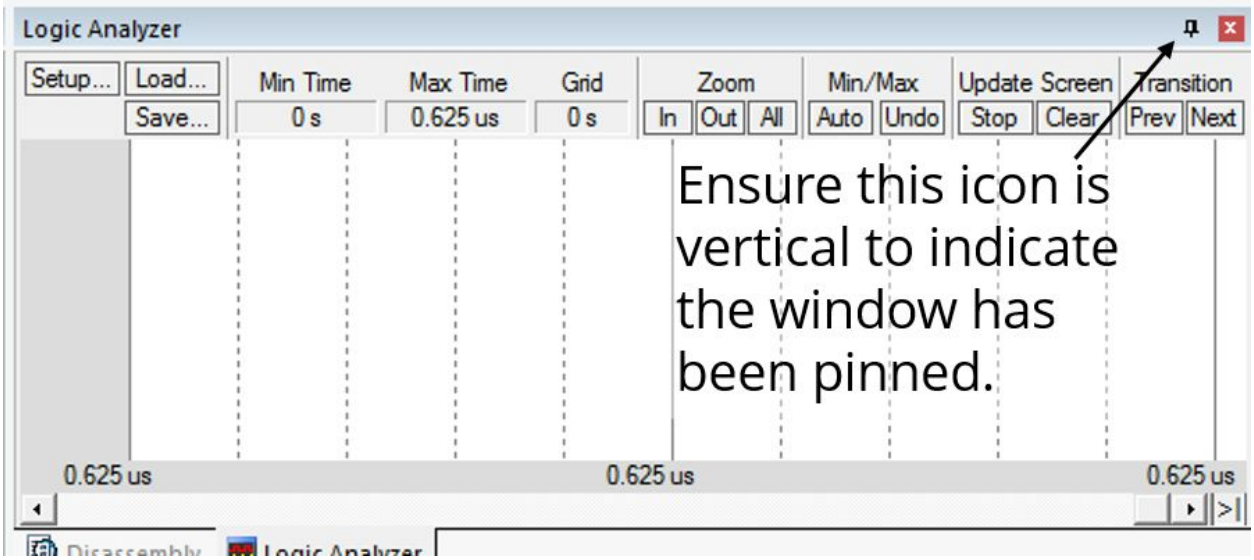
1. Lab 2 grading sheet. You can print it yourself or pick up a copy in lab. You fill out the information at the top.
2. Two screenshots of the logic analyzer, like the two pictures in Figure 2.1 above, showing the system running on the simulator. One showing the touch and release, and the other showing a close up of the LED toggling
3. Flowchart of the delay function
4. Pseudocode of the delay function
5. Assembly source code of your final program
6. Measurement of how much microcontroller time is simulated in 10 seconds of actual time.
7. Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

## FAQ

The list of FAQ below are populated from Piazza over the semesters (thanks to the contributions of all past TAs and students). More questions may be posted so please check back regularly.

1. My LED “dims” more than actually blinking on and off. Am I doing something wrong?
   The LED is toggling at 10 times a second when the switch is pressed--the human eye should be able make out this 5 Hz blinking. You are probably not waiting the proper amount of time. To verify that your LED is toggling at the correct frequency, verify it with the logic analyzer and/or the oscilloscope.

2. How do I add a pin to the logic analyzer?
   Click "Setup", then click “New(insert)” on the top right corner of the setup window, and add whatever port you need to track in the following format "PORTF.2" or "PORTE.3" for examples. Ensure you set up the port as a **bit**, not in **analog mode**.
3. I added a pin in the “Setup Logic Analyzer” window but nothing shows up when I close the window. How should I fix it?
   This may be an issue of the logic analyzer not being pinned to Keil. See details in figure below:




4. The logic analyzer doesn’t work when I am debugging the board!
   Logic analyzer only works on the simulator. If you want to examine the waveform of a signal coming out from the board, use the real oscilloscope when debugging on the board.
5. My program works fine on the simulator but it is not functioning after I download it to the board. What am I doing wrong?
   Make sure to press “Reset” for the program to actually start running on the board.
6. Is the simulation suppose to run faster or slower than real time?
   How quickly the simulator runs depends on how fast your computer runs. For some it will take 10 real-time seconds to simulate 5 seconds of Cortex-M time, for others it will take 5 real-time seconds to simulate 10 seconds of Cortex-M time. However, when you see 5 seconds of "Cortex-M time" in the logic analyzer, it should be roughly equivalent to what would happen if you ran the program on the actual board for 5 real-time seconds (i.e., not in the simulator).