Lab 3. Breathing LED interfacing with Switches

COMP-462 Embedded Systems

Preparation

Purpose

System Requirements

Procedure

- Part a - Develop Incrementally
- Part b - Read Data Sheets
- Part c - Construct Circuit
- Part d - Debug Hardware + Software

Demonstration

Deliverables

Precautions to avoid damaging your system

FAQ

Preparation

1. Read Sections 1.17, 2.4, 2.7, 3.1, 3.2, and 3.3
2. Look at the starter project for Lab 3 in the original EE319K installation

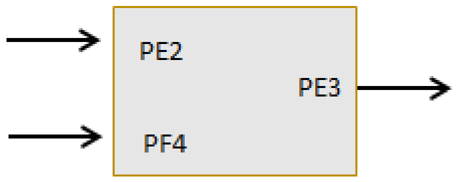
Purpose

The purpose of this lab is to learn how to interface a switch and an LED and to program the LED to operate at a variable duty-cycle determined by the switch. You will also perform explicit measurements on the circuits in order to verify they are operational and to improve your understanding of how they work.

System Requirements

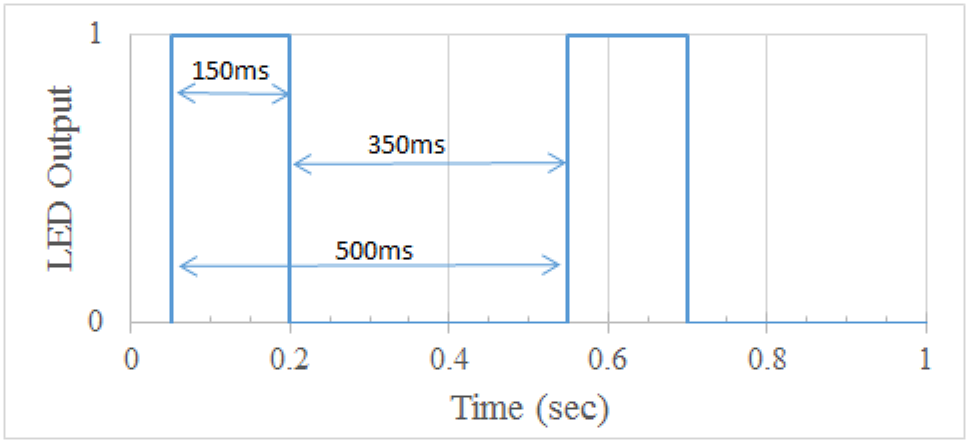
The primary task of the lab is to make an LED toggle at 2 Hz with varying duty-cycle.

The external hardware for the lab includes, Two buttons, one LED, and associated components needed to interface them. The negative logic switch on PF4 exists on the LaunchPad itself.



- PF4 is negative-logic Button input (0 means pressed, 1 means not pressed)
- PE2 is positive-logic Button input (1 means pressed, 0 means not pressed)
- PE3 is positive-logic LED output (1 activates external LED on protoboard)

Overall functionality of this system is to operate as follows:

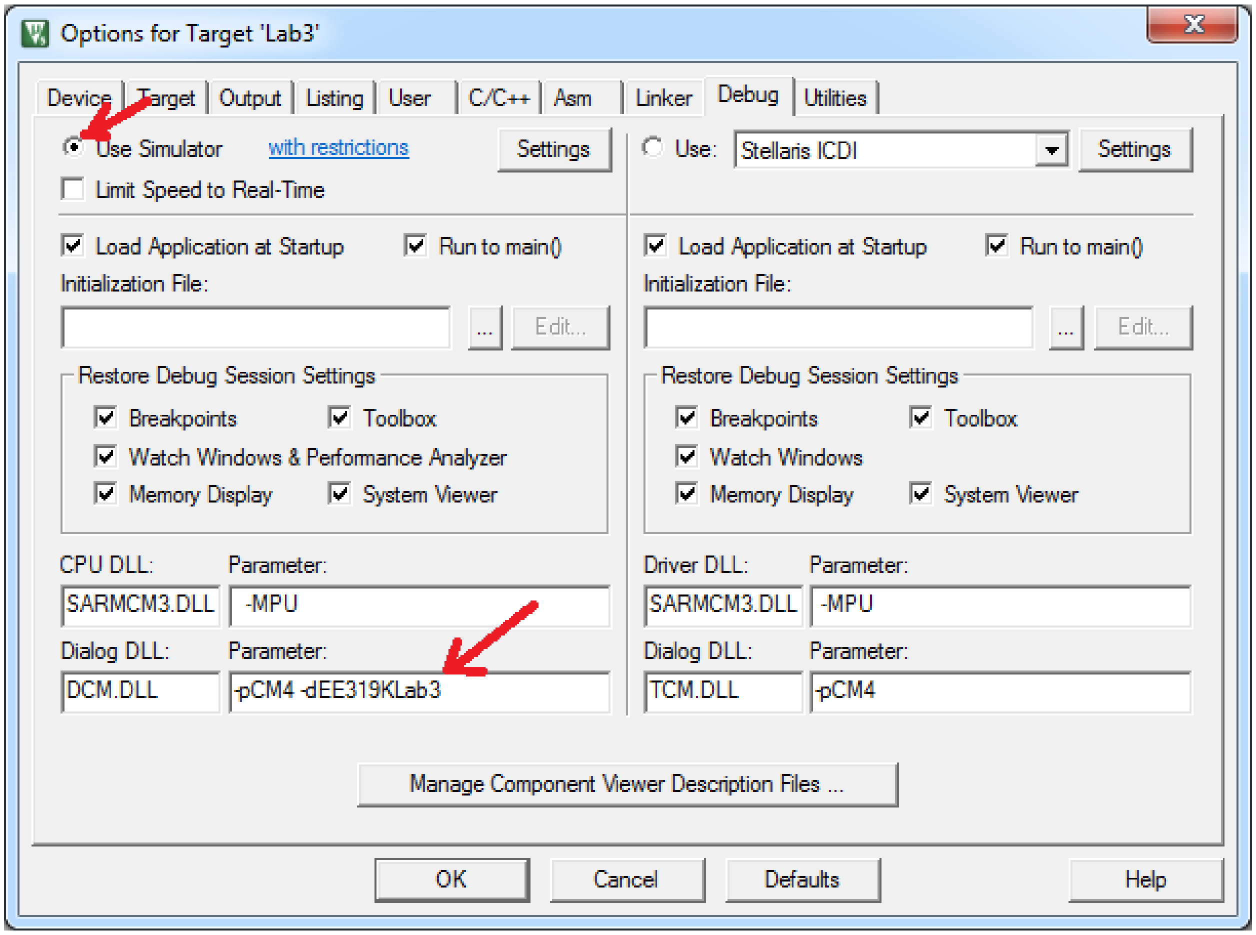
1. Make PE3 an output and make PF4 and PE2 inputs.
2. The system starts with the LED toggling at 2 Hz, which is 2 times per second with a duty-cycle of 30%. Therefore, the LED is ON for 150ms (0.3*½ sec) and OFF for 350ms (0.7*½ sec).
3. 

4. When the button on (PE2) is pressed-and-released increase the duty cycle by 20% (modulo 100%). Therefore, for each press-and-release the duty cycle changes from 30% to 50% to 70% to 90% to 10% to 30% and so on.
5. Implement a "breathing LED" when button on (PF4) is pressed and held:
   a. Be creative and play around with what "breathing" means. An example of "breathing" is most computers power LED in sleep mode (e.g., https://www.youtube.com/watch?v=ZT6siXyIjvQ).
   b. When (PF4) is released while in breathing mode, resume blinking at 2 Hz. The duty cycle can either match the most recent duty-cycle or reset it back to 30%.

Procedure

Back in Lab 1, you developed and debugged your system using the simulator, however the code you developed could also have been run on the real board. In this lab you will build and test your hardware for the real-board but for software debugging purposes it will be useful to run in simulation mode.

To run the simulator, you must do two things. First, execute Project->Options and select the Debug tab. The debug parameter field must include **-dEE319KLab3**. Second, the **EE319KLab3.dll** file must be present in your Keil\ARM\BIN folder (already installed during the EE319Kware install step you performed for Lab 1).

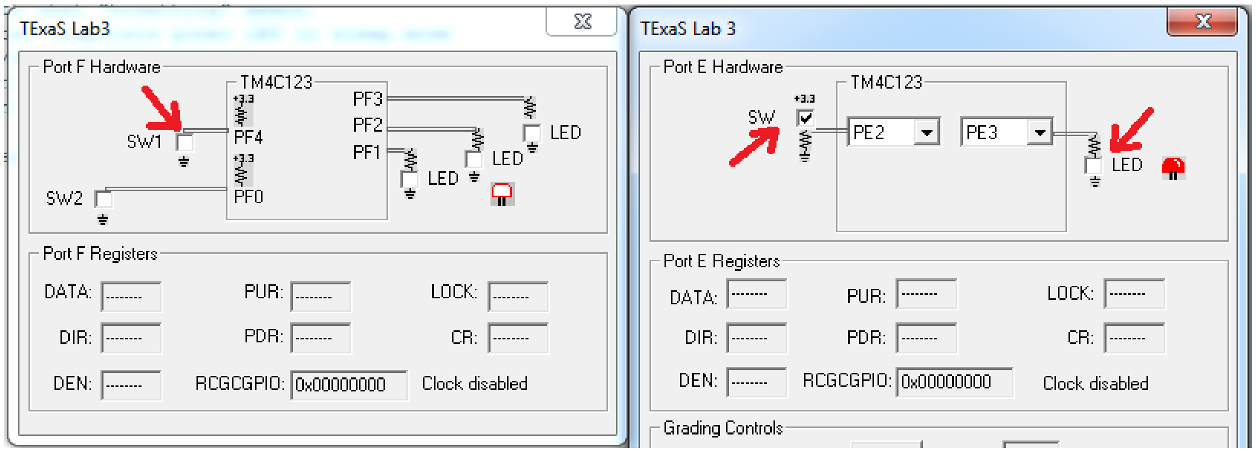


Options for Target 'Lab3'

(●) Use Simulator with restrictions
[ ] Limit Speed to Real-Time
[✓] Load Application at Startup [✓] Run to main()
Initialization File:
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows & Performance Analyzer
[✓] Memory Display [✓] System Viewer
CPU DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: DCM.DLL Parameter: -pCM4 -dEE319KLab3

( ) Use: Stellaris ICDI
[✓] Load Application at Startup [✓] Run to main()
Initialization File:
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows
[✓] Memory Display [✓] System Viewer
Driver DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: TCM.DLL Parameter: -pCM4

Figure 3.1a. Using TExaS to debug your software in simulation mode (DCM.DLL -pCM4 -dEE319KLab3).



Port F Registers

DATA:	--------	PUR:	--------	LOCK:	--------
DIR:	--------	PDR:	--------	CR:	--------
DEN:	--------	RCGCGPIO:	0x00000000	Clock disabled	

Port E Registers

DATA:	--------	PUR:	--------	LOCK:	--------
DIR:	--------	PDR:	--------	CR:	--------
DEN:	--------	RCGCGPIO:	0x00000000	Clock disabled	

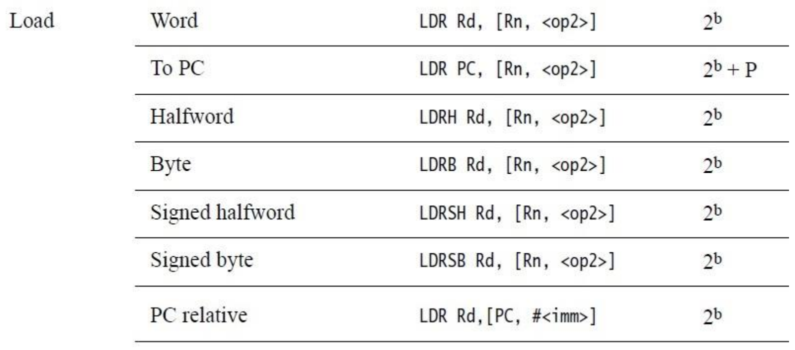
Figure 3.1b. Simulation of Port E. PF4 and PE2 are input. PE3 is output

Part a - Develop Incrementally

The best approach to time delay will be to use a hardware timer, which we will learn to use in Lab 4. In this lab we do not expect you to use the hardware timer. Again, use the logic analyzer on the simulator to verify the software operates properly.

Whenever we call **TExaS_Init**, we activate the phase lock loop (PLL) and run at 80 MHz.

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

On the TM4C123 the default bus clock is 16 MHz ±1%. However using TExaS_Init engages the Phase-Lock-loop (PLL) and runs the TM4C123 at 80 MHz. At 80 MHz one clock-cycle takes 1/80,000,000 seconds or 12.5 nanoseconds. The following is a portion of a listing file (dis-assembly) with a simple delay loop. The SUBS and BNE instructions are executed 800 times. The SUBS takes 1 cycle and the BNE takes (1+P) where P can vary between 1 and 3 cycles. In simulation P is 2 making the wait loop be of 4 cycles. On the real-board P can vary because of optimization using a pipeline. The minimum time to execute this code is 800*(1+(1+1))*12.5 ns= 30 us. The maximum time to execute this code is 800*(1+(1+3))*12.5 ns= 50 us. 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 an indeterminate number of cycles on the real board)*

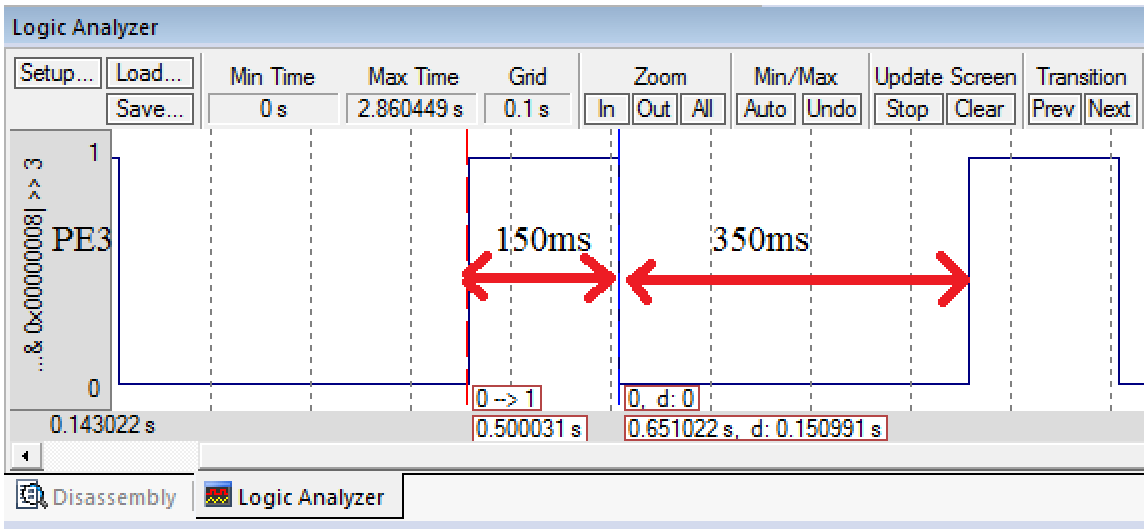
An accurate method to measure time uses a logic analyzer or oscilloscope. In the simulator, we will use a simulated logic analyzer, and on the real board we will use an oscilloscope. To measure execution time, we count rising and falling edges on a digital output pin that occur at known places within 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 PE3.

We suggest developing the code in stages:
Stage 1: Write a simple main loop that toggles an LED (PE3) at 2 Hz with a 50% duty-cycle calling a delay subroutine. Stage 1 software could be implemented this way:

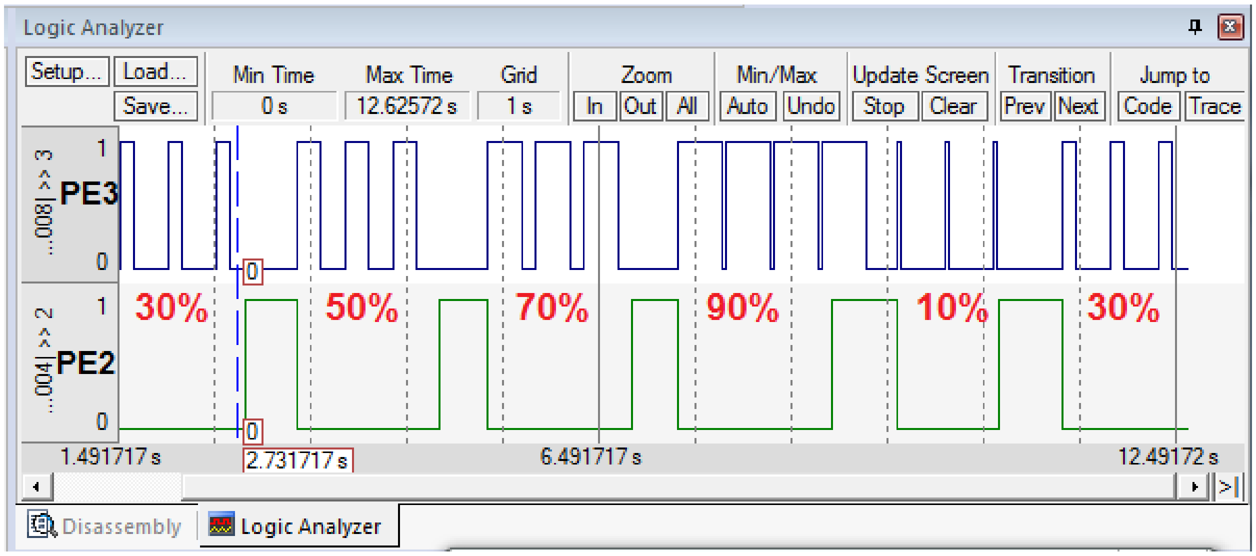
```
        BL TExaS_Init ; voltmeter, bus clock to 80 MHz
        Turn on Port E clock
        Make PE3 output
        Enable interrupts so the TExaS oscilloscope runs
   Loop
        Delay for ¼ th of a second
        Toggle PE3
        B   Loop
```

Stage 2: Rewrite code in Stage1 so you can program a target duty-cycle (say 30%). Verify in simulator using the Logic Analyzer that you indeed have a 2 Hz signal with the target duty-cycle of 30 % (on for 150 ms, and off for 350 ms). Stage 2 software could be implemented this way:

```
          BL TExaS_Init ; activate voltmeter, bus clock to 80 MHz
          Turn on Port E clock
          Make PE3 output
          Enable interrupts so the TExaS oscilloscope runs
     Loop
          Set PE3 high
          Delay for 150ms
          Clear PE3 low
          Delay for 350ms
          B   Loop
```



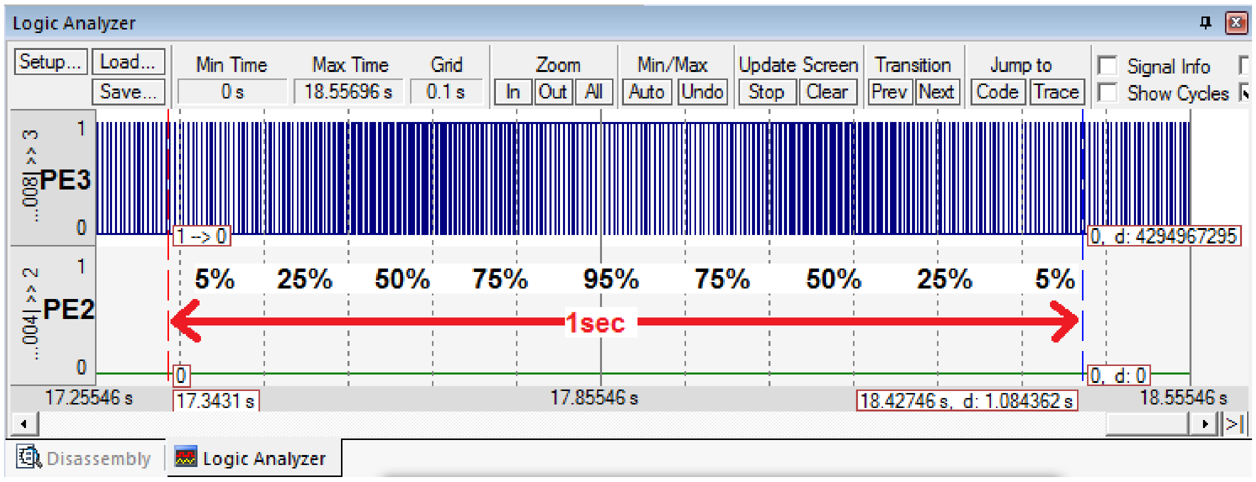
Stage 3: Add software to read the switch on PE2, and use the switch to modify the duty-cycle. Note that the duty-cycle change occurs on a press followed by a release. What the LED does while touching the switch does not matter. So, while the switch is being touched, you can stop the oscillations, or continue the oscillations, your choice. The change takes effect on the release. Test in the simulator and verify function using the simulated Logic Analyzer (*this video shows a similar lab solution, but with different pins, different frequency and different duty cycles* https://youtu.be/5fD71LdAXZs). In the following Logic Analyzer recording, notice the LED stops toggling when the switch is pressed, then the toggle resumes when the switch is released. Feel free to implement other similar strategies as long the the duty cycle is changed exactly once each time the switch is pressed and released.



Stage 4: Build the circuit (see figure below) and check twice to make sure you have it correct. Connect the real-board and flash it with the code you wrote in Stage 3. You will see a flashing LED and the effect of changes in the duty-cycle are visible to the naked eye. Note at this point that the Logic Analyzer in Keil no longer is available as a tool as it only works in Simulation.

Stage 5: At this point you implemented 90% of the requirement of this lab. Now you will add the breathing feature which is enabled when PF4 (SW1) is pressed and disabled when released. We want you to be creative in devising a solution to implement this feature. However, here are some ideas:

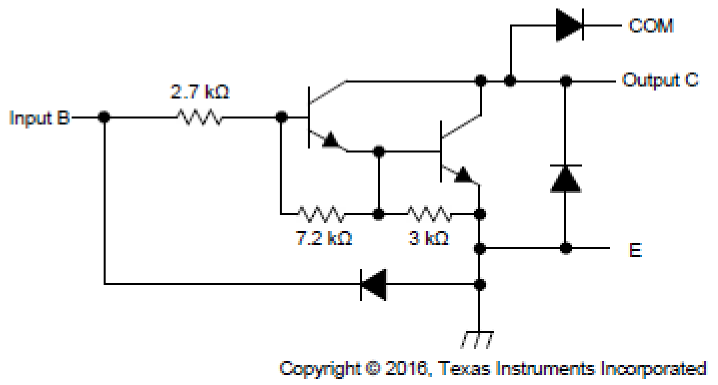
- A breathing LED increases in brightness gradually and once it reaches its full brightness it decreases it brightness gradually till it reaches zero brightness. At which point it again repeats the increase.
- 2 Hz is too slow and will be visible to the naked eye as distinct on and off. We need the toggle the LED at a higher frequency (say 100 Hz) to be able to see the desired effect of duty-cycle impacting brightness.
- Varying brightness is achieved by varying duty-cycle. You may need more than 5 levels of duty-cycle for better breathing feel.
- Consider changing the duty-cycle at a programmable rate. That is, if your current duty-cycle is x%, stay at this duty-cycle for N iterations before changing to (x ± d)%.
- Remember, you can play with both the frequency and the duty-cycle.

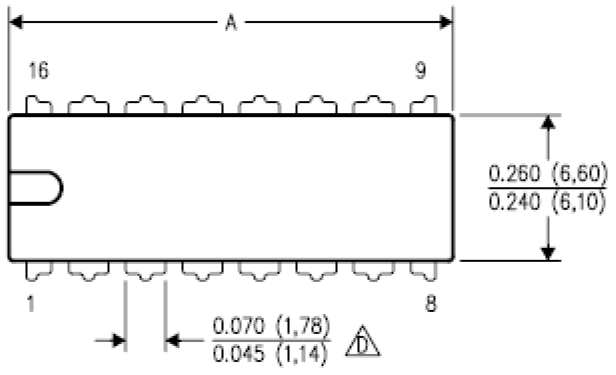


Part b - Read Data Sheets

Engineers must be able to read datasheets during the design, implementation and debugging phases of their projects. During the design phase, datasheets allow us to evaluate alternatives, selecting devices that balance cost, package size, power, and performance. For example, we could have used other IC chips like the 7406 to interface the LED to the TM4C123. In particular, we chose the ULN2003 because it has a large output current (I_{OL} = 500 mA), 7 drivers, and is very inexpensive (59¢). During the implementation phase, the datasheet helps us identify which pins are which. During the debugging phase, the datasheet specifies input/output parameters that we can test. Download the ULN2003B and LED datasheets, and LED_red.pdf, and find the two pictures as shown in Figure 3.5. Next, hold your actual ULN2003B chip and identify the location of pin 1. Find in the datasheet the specification that says the output low voltage (V_{OL}) will be 0.4V when the output low current (I_{OL}) is 16 mA (this is close to the operating point we will be using for the LED interface).

You will connect the E pin to ground. You will not connect anything to the COM pin in Lab 3. In Lab 5, you will connect COM to +5V when using this chip for the stepper motor.





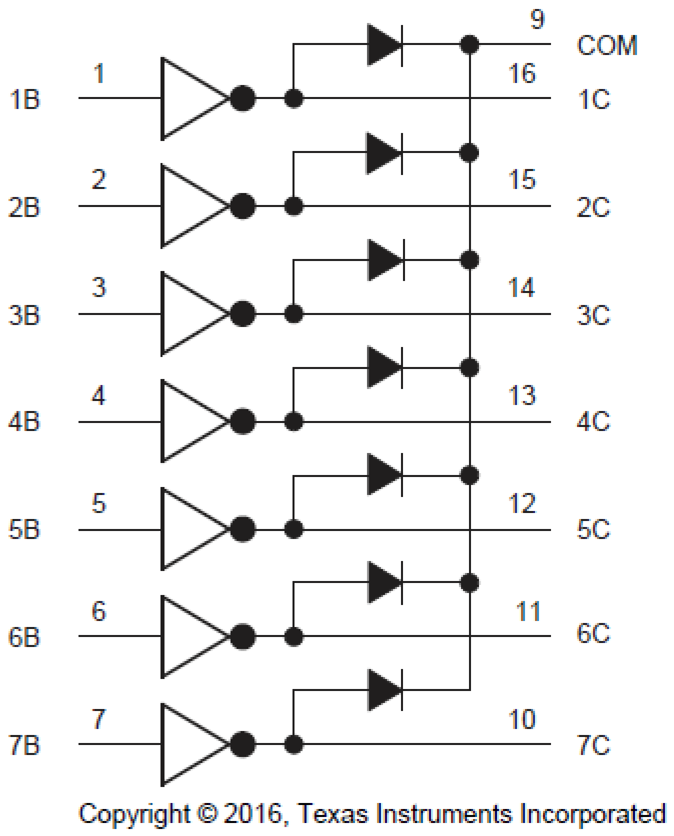


Figure 3.5. Connection diagram and physical package diagram for the ULN2003B. Connect E (pin 8) to ground.

Using the data sheet, hold an LED and identify which pin is the anode and which is the cathode. Current flows from anode to cathode when the LED is on.

Sometimes we are asked to interface a device without a data sheet. Notice the switch has 4 pins in a rectangular shape, as shown in Figure 3.6. Each button is a single-pole single-throw normally-open switch. All four pins are connected to the switch. Using your ohmmeter determine which pairs of pins are internally connected (having a very small resistance), and across which pair of pins is the switch itself. In particular, draw the internal connections of the switch, started in Figure 3.6, showing how the four pins are connected to the switch.

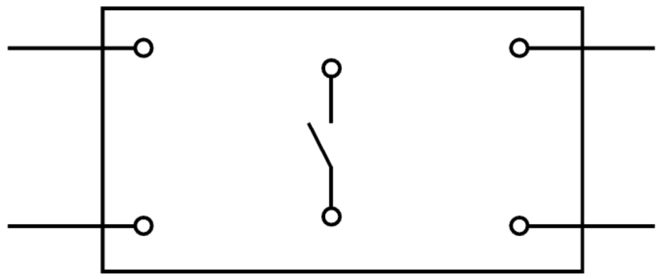


Figure 3.6. Connection diagram for the normally open switch.

To build circuits, we'll use a solderless breadboard, also referred to as a protoboard. The holes in the protoboard are internally connected in a systematic manner, as shown in Figure 3.7. The long rows of of 50 holes along the outer sides of the protoboard are electrically connected. Some protoboards like the one in Figure 3.7 have four long rows (two on each side), while others have just two long rows (one on each side). We refer to the long rows as power buses.

If your protoboard has only two long rows (one on each side), we will connect one row to +3.3V and another row to ground. If your protoboard has two long rows on each side, then two rows will be ground, one row will be +3.3V and the last row will be +5V (from VBUS). Use a black marker and label the voltage on each row. In the middle of the protoboard, you'll find two groups of holes placed in a 0.1 inch grid. Each adjacent row of five pins is electrically connected. We usually insert components into these holes. IC chips are placed on the protoboard, such that the two rows of pins straddle the center valley.

To make connections to the TM4C123 we can run male-male solid wire from the bottom of the microcontroller board to the protoboard. For example, assume we wish to connect TM4C123 PE3 output to the ULN2003B input pin 1. First, cut a 24 gauge solid wire long enough to reach from PE3 and pin 1 of the ULN2003B. Next, strip about 0.25 inch off each end. Place one end of the wire in the hole for the PE3 and the other end in one of the four remaining holes next to the ULN2003B pin 1.

I like to place my voltmeter on the +3.3V power when I first power up a new circuit. If the +3.3V line doesn't immediately jump to +3.3V, I quickly disconnect the USB cable. Alternatively, you can monitor the +3.3V line with the green led in the upper portion of the LaunchPad near the USB connectors. If this led does not turn on when you think you are applying power, you might have a short circuit.

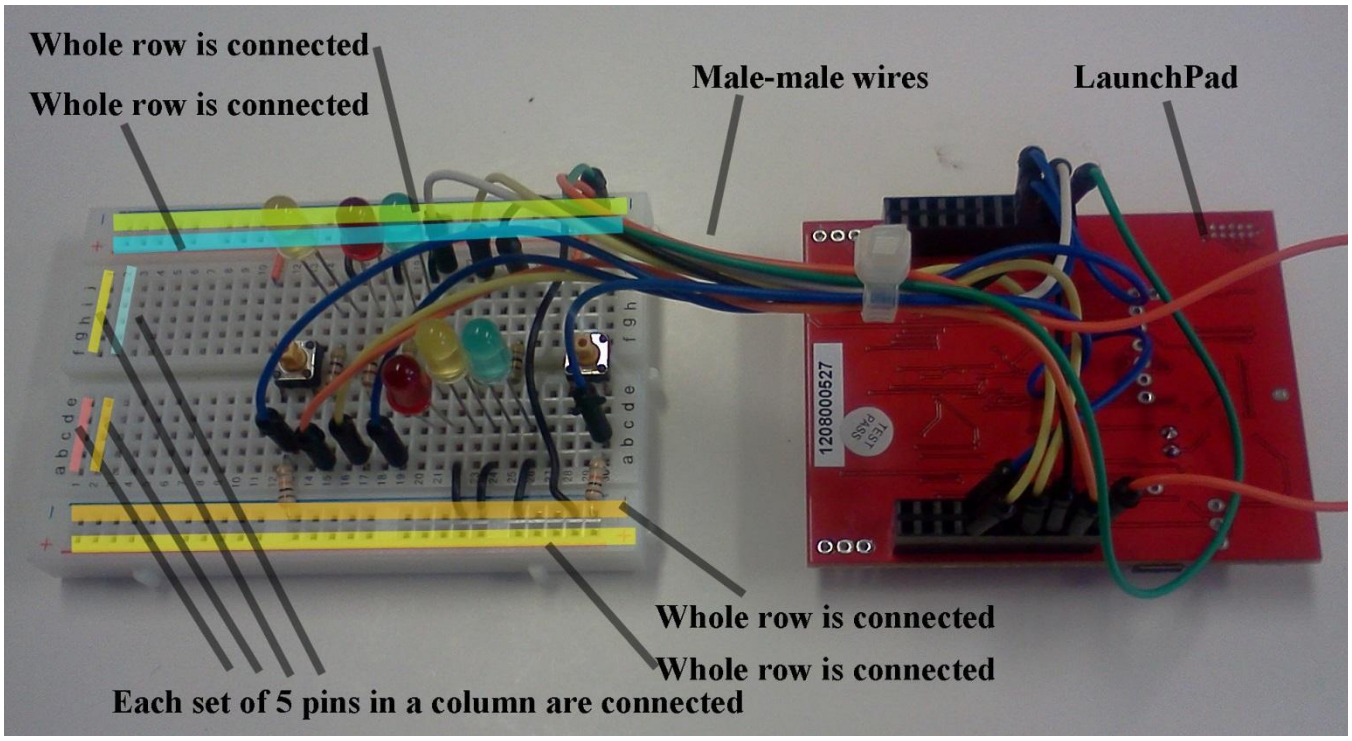


Figure 3.7. All the pins on each of the four long rows are connected. The 5 pins in each short column are connected. Use male-male wires to connect signals on the LaunchPad to devices on the protoboard. Make sure ground wire connected between the LaunchPad and your circuit. The +3.3V and VBUS (+5V) power can be wired from the LaunchPad to your circuit. I like to connect the two dark blue rows to ground, one red row to +3.3V and the other red row to VBUS(+5V).

Part c - Construct Circuit

After the software has been debugged on the simulator, you will build the hardware on the real board. Please double check your design before you apply power (plug in the USB cable). *Do not place or remove wires on the protoboard while the power is on.* One possible circuit diagram for this lab is given in Figure 3.8.

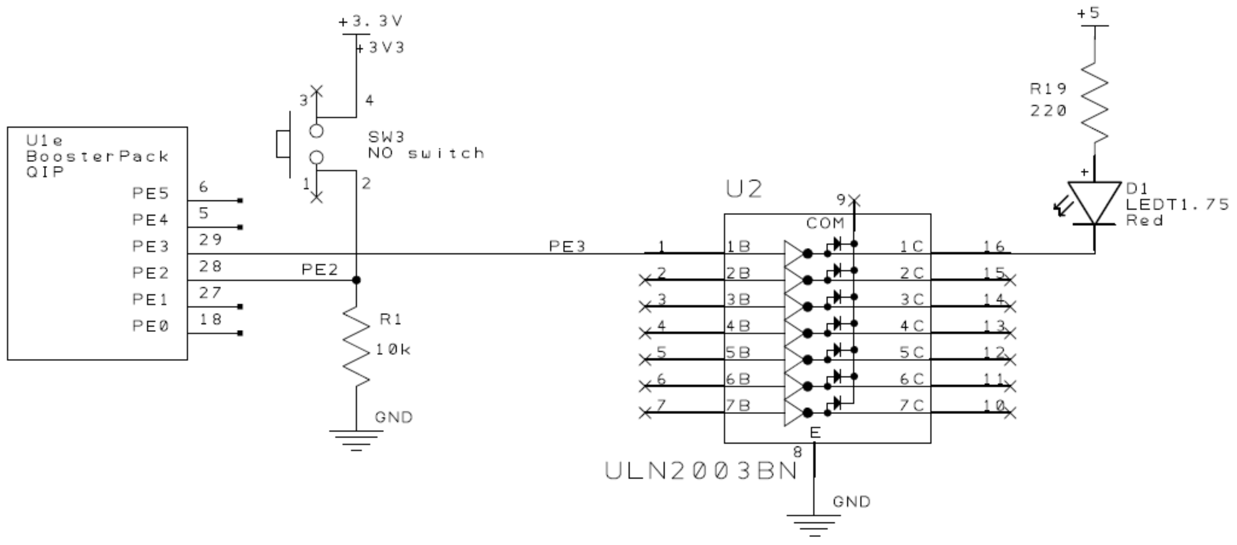


Figure 3.8. PCB Artist drawing showing Port E, 3.3V power, a 10kΩ resistor, a switch, one ULN2003B gate, an LED, a 220Ω resistor, +5V power and ground.

Before connecting the switch to **PE2** of the microcontroller, please take the measurements in Table 3.1 using your digital multimeter. The input voltage (V_{PE2}) is the signal that will eventually be connected to **PE2**. With a positive logic switch interface, the resistor current will be V_{PE2}/10kΩ. The voltages should be near +3.3 V or near 0 V and the currents will be less than 1 mA. The goal is to verify the V_{PE2} voltage is low when the switch is not pressed and high when the switch is pressed.

Next, you can connect the input voltage to **PE2** and use the debugger to observe the input pin to verify the proper operation of the switch interface. You will have to single step through the code that initializes Port E, and PE2. You then execute the **Peripherals->TExaS Port E** command. As you single step you should see the actual input as controlled by the switch you have interfaced, see Figure 3.8.

The next step is to build the LED output circuit. An LED is a **diode** that emits light when an electric current passes through it, as shown in Figure 3.9. LEDs have polarity, meaning current must pass from anode to cathode to activate. The anode is labeled **a** or **+** , and cathode is labeled **k** or **-**. The cathode is the short lead and there may be a slight flat spot on the body of round LEDs. Thus, the anode is the longer lead. LEDs are not usually damaged by heat when soldering. Furthermore, LEDs will not be damaged if you plug it in backwards. However, LEDs won't work plugged in backwards. Look up the pin assignments in the ULN2003 data sheet. Be sure to connect E (pin 8) to ground. For this lab, the COM (pin 9) on the ULN2003B will not be connected.

Parameter	Value	Units	Conditions
Resistance of the 10kΩ resistor, R1		ohms	with power off and disconnected from circuit (measured with ohmmeter)

Supply Voltage, $V_{+3.3}$		volts	Powered (measured with voltmeter)
Input Voltage, V_{PE2}		volts	Powered, but with switch not pressed (measured with voltmeter)
Resistor current		mA	Powered, but switch not pressed $I=V_{PE2}/R1$ (calculated and measured with an ammeter)
Input Voltage, V_{PE2}		volts	Powered and with switch pressed (measured with voltmeter)
Resistor current		mA	Powered and switch pressed $I=V_{PE2}/R1$ (calculated and measured with an ammeter)

Table 3.1. Switch measurements.

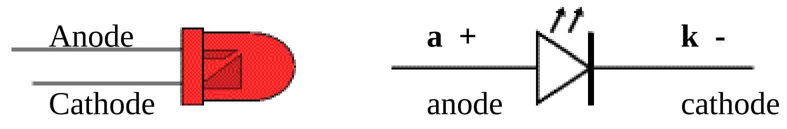


Figure 3.9. Left: a side view of an LED with leads labeled; Right: the corresponding circuit diagram

Take the measurements as described in Table 3.2. The R19 measurement occurs before R19 is inserted into the circuit. Single step your software to make **PE3** to output. Initially **PE3** will be low. So take four measurements with **PE3** low, rows 2,3,4,5 in Table 3.2. Then, single step some more until **PE3** is high and measure the three voltages (rows 8,9,10 in Table 3.2). When active, the voltage across the LED should be about 2 V, and the LED current should be about 10 mA. The remaining rows are calculated values, based on these 8 measurements. The LED current (row 12) can be determined by calculation or by direct measurement using the ammeter function. You should perform both ways to get LED current.

Warning: NEVER INSERT/REMOVE WIRES/CHIPS WHEN THE POWER IS ON.

Row	Parameter	Value	Units	Conditions
1	Resistance of the 220Ω resistor, R19		ohms	with power off and disconnected from circuit (measured with ohmmeter)

2	+5 V power supply V_{+5}		volts	(measured with voltmeter relative to ground, *notice that the +5V power is not exactly +5 volts*)
3	TM4C123 Output, V_{PE3} input to ULN2003B		volts	with **PE3** = 0 (measured with voltmeter relative to ground). We call this V_{OL} of the TM4C123.
4	ULN2003B Output, pin 16, V_{k-} LED k-		volts	with **PE3** = 0 (measured with voltmeter relative to ground). This measurement will be weird, because it is floating.
5	LED a+, V_{a+} Bottom side of R19 (anode side of LED)		volts	with **PE3** = 0 (measured with voltmeter relative to ground). This measurement is also weird, because it too is floating.
6	LED voltage		volts	calculated as V_{a+} - V_{k-}
7	LED current (off)		mA	calculated as (V_{+5} - V_{a+})/R19 and measured with an ammeter
8	TM4C123 Output, V_{PE3} input to ULN2003B		volts	with **PE3** = 1 (measured with voltmeter relative to ground). We call this V_{OH} of the TM4C123.
9	ULN2003B Output pin 16, V_{k-} LED k-		volts	with **PE3** = 1 (measured with voltmeter relative to ground). We call this V_{OL} or $V_{CE(sat)}$ of the ULN2003B.
10	LED a+, V_{a+} Bottom side of R19 (anode side of LED)		volts	with **PE3** = 1 (measured with voltmeter relative to ground)
11	LED voltage		volts	calculated as V_{a+} - V_{k-}
12	LED current (on)		mA	calculated as (V_{+5} - V_{a+})/R19 and measured with an ammeter

Table 3.2. LED measurements (assuming the 220 Ω resistor is labeled R19 in FIgure 3.8).

Part d - Debug Hardware + Software

Debug your combined hardware and software system on the real-board.

Demonstration

I may look at your data and expect you to understand how the data was collected and how the switch and LEDs work. Also be prepared to explain how your software works and to discuss other ways the problem could have been solved.

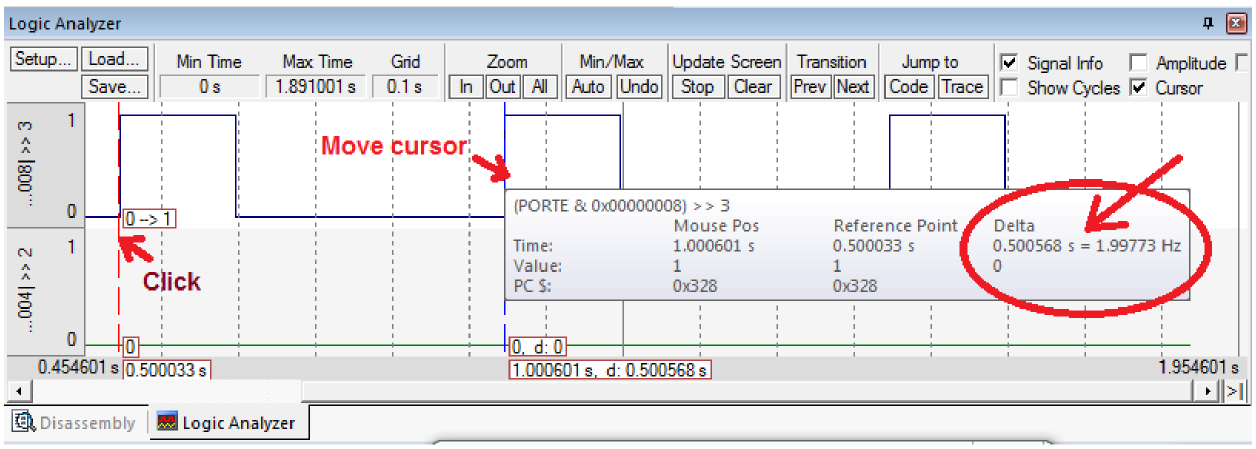
Please answer these questions in your report. Why the ULN2003B was used to interface the LED? I.e., why did we not connect the LED directly to the TM4C123. What would the flashing LED “look” like if the delay were 1ms? How would you modify the software to change the rate at which LED flickers? What operating point (voltage, current) exists when the LED is on? Sketch the approximate current versus voltage curve of the LED. Explain how you use the resistor value to select the operating point. What is the difference between a positive logic and negative logic interface for the switch or the LED? We may test to see if you can measure voltage, current and/or resistance with your meter (so bring your meter to the demonstration).

Deliverables

1. Circuit diagram (hand-drawn or optionally using a CAD software)
2. Screenshots like Figure 3.10a,b showing your debugging in the simulator
3. Switch measurements (Table 3.1)
4. LED measurements (Table 3.2)
5. Assembly source code of your final program

Precautions to avoid damaging your system

1. Do not attach or remove wires on the protoboard when power is on. Always shut power off when making hardware changes to the system.
2. Touch a grounded object before handling CMOS electronics. Try not to touch any exposed wires.
3. Do not plug or unplug the modules into the LaunchPad while the system is powered.
4. Do not use the TM4C123 with any external power sources, other than the USB cable. In particular, avoid connecting signals to the TM4C123 that are not within the 0 to +5V range. Voltages less than 0V or greater than +5V will damage the microcontroller.
5. Do not use PC3,PC2,PC1,PC0. These are needed for the debugger.
6. You can’t use PA1 PA0 PD4 and PD5. These are connected to the serial port and USB.
7. If you use both PD0 and PB6, remove R9 from your board. If you use both PD1 and PB7, remove R10 from your board.

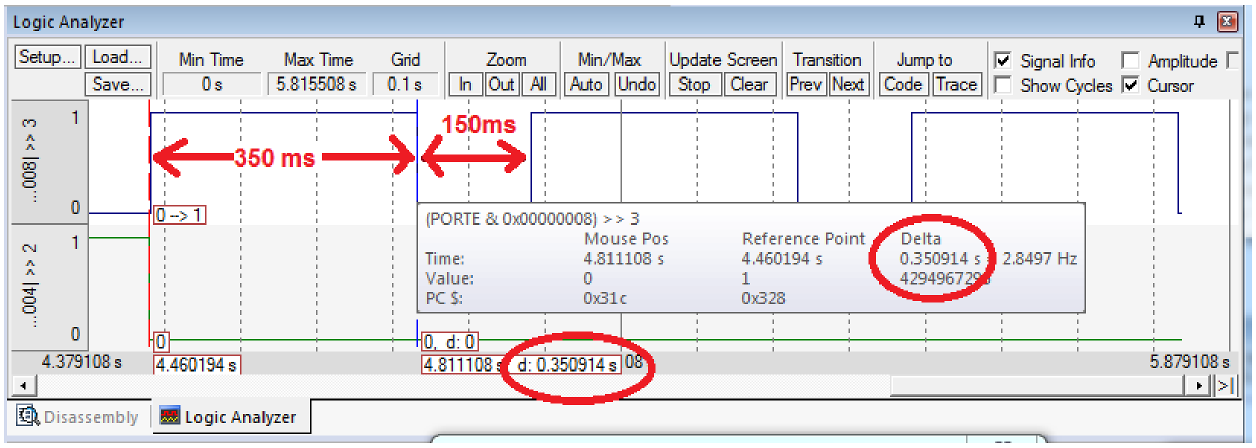


```
Min Time   Max Time     Grid
0 s        1.891001 s   0.1 s

(PORTE & 0x00000008) >> 3
          Mouse Pos     Reference Point   Delta
Time:     1.000601 s    0.500033 s        0.500568 s = 1.99773 Hz
Value:    1             1                 0
PC $:     0x328         0x328

0 -> 1
0
0,  d: 0
0.454601 s   0.500033 s   1.000601 s,  d: 0.500568 s   1.954601 s
```

Figure 3.10a. Simulation of Lab 3, showing PE3 output toggling at 2 Hz with 30% duty-cycle.



(PORTE & 0x00000008) >> 3

	Mouse Pos	Reference Point	Delta	
Time:	4.811108 s	4.460194 s	0.350914 s	2.8497 Hz
Value:	0	1	4294967295	
PC $:	0x31c	0x328		

Figure 3.10b. Simulation of Lab 3, showing PE3 output toggling at 2 Hz with 70% duty-cycle.

FAQ

Before asking the instructor, please check the following if you have any questions about this lab:

1. For my ½ s (500 ms) delay, I'm having trouble getting the delay up to the right number of ms. I've tried combining multiple delay subroutines to increase the delay, but i'm still quite a ways from 500ms. Can anyone share a smart way of increasing the delay?

   Notice that the clock for this lab is running at 80 MHz instead. Your calculation in writing your delay loop count must account for this speed which implies each cycle is 12.5ns.

2. How are we supposed to determine which pair of pins has the switch across them using the ohmmeter? Each pair has the exact same resistance between them when I measure it.

   Were you measuring the resistance with the switch pressed? If so, all of the pins will be connected together and have the same, albeit small, resistance. Otherwise, if you were measuring with the switch not pressed, you should be measuring different resistances between different pairs. If you are absolutely sure this is not the case, it is possible that you could have a broken switch.

   The size of the resistance might also provide a hint as to what is going on (think about when the resistance should be very large and when it should be very small, assuming the switch is working correctly).

3. How do we measure current through a resistor?

   You put the multimeter in series with the resistor. Or you can kind of "cheat" and measure the voltage, then calculate the current. You should understand how to do both ways though.

4. What is the bottom side of a resistor?

   The lab manual is sort of confusing when it says 'bottom side of R19', but what it wants is the voltage of the LED anode side. It refers to the part of the circuit 'below' the 220 resistor and 'above' the LED.

5. How do we measure stepwise data for rows 8 -10 on the table? I'm reading fluctuating data on the multimeter.

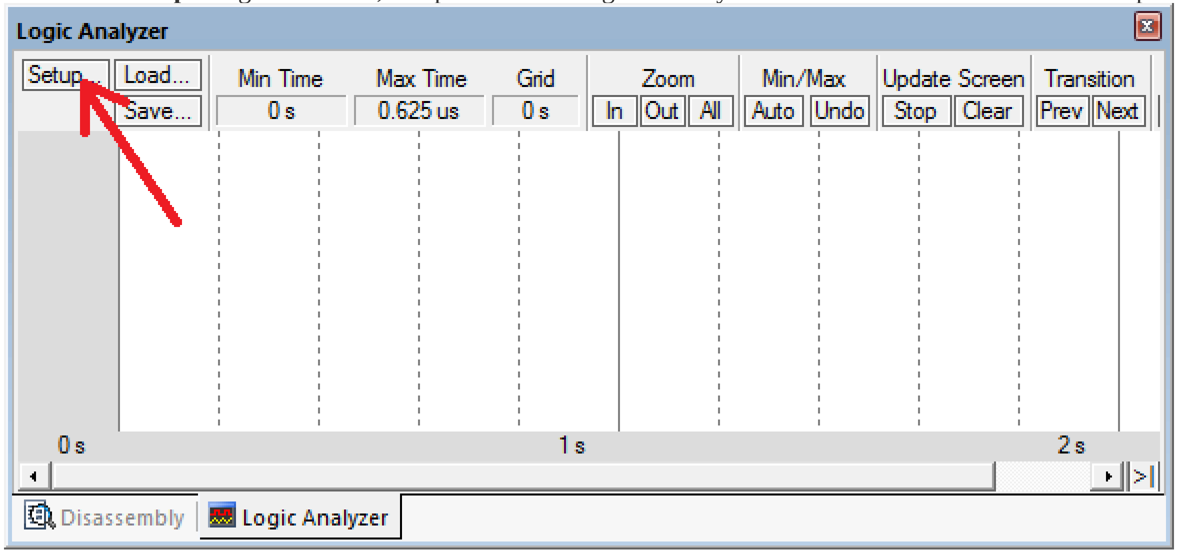
   With Keil you can debug your circuit in real time. If you click under Projects --> Options for Target --> Debug and use the **Stellaris ICDI** instead of the simulator you can the press debug and actually step through your code and watch how it affects your circuit. Think about when in your program should PE3 be set to zero and jump to it. You can now measure these elements knowing that PE3 is supplying no voltage. The values can be expected to vary slightly.

6. After thoroughly checking my hardware wiring, I tried implementing it with the software and couldn't even get the LED to turn on. Are there any possible explanations for why this could be happening?

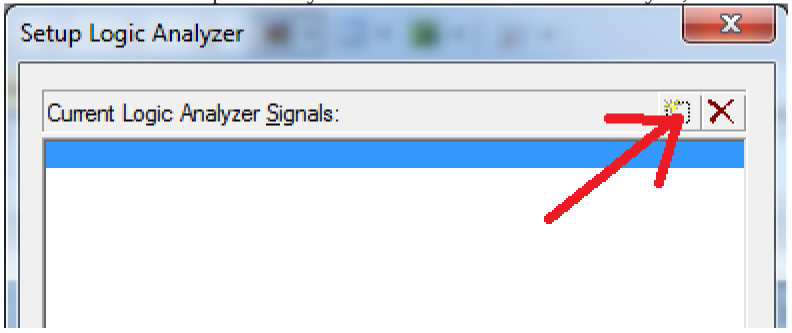
   Before moving on to circuits, make sure your program work first on the simulator. Once that is checked, make sure everything is connected securely. Circuit problems come from crappy wires most of the time.. are you using your own wires or wires from the check out desks? A multimeter will be your best friend when it comes to debugging a circuit.

7. My logic analyzer no longer is showing the ports on the side. When I run it the analyzer is completely blank. Any idea on how to get the values of the ports back on the logic analyzer?

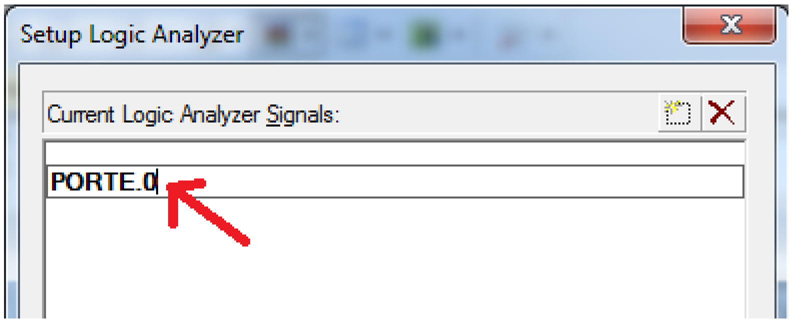
   This happens when you run the simulator with logic analyzer, run the debugger on the board, and then run the simulator again. Running on the real board clears the logic analyzer configuration. You will have to execute **Setup** again. First, open the logic analyzer window and click Setup.



   For each pin you wish to analyze, click new insert button



   Add the port pin you want and then enter

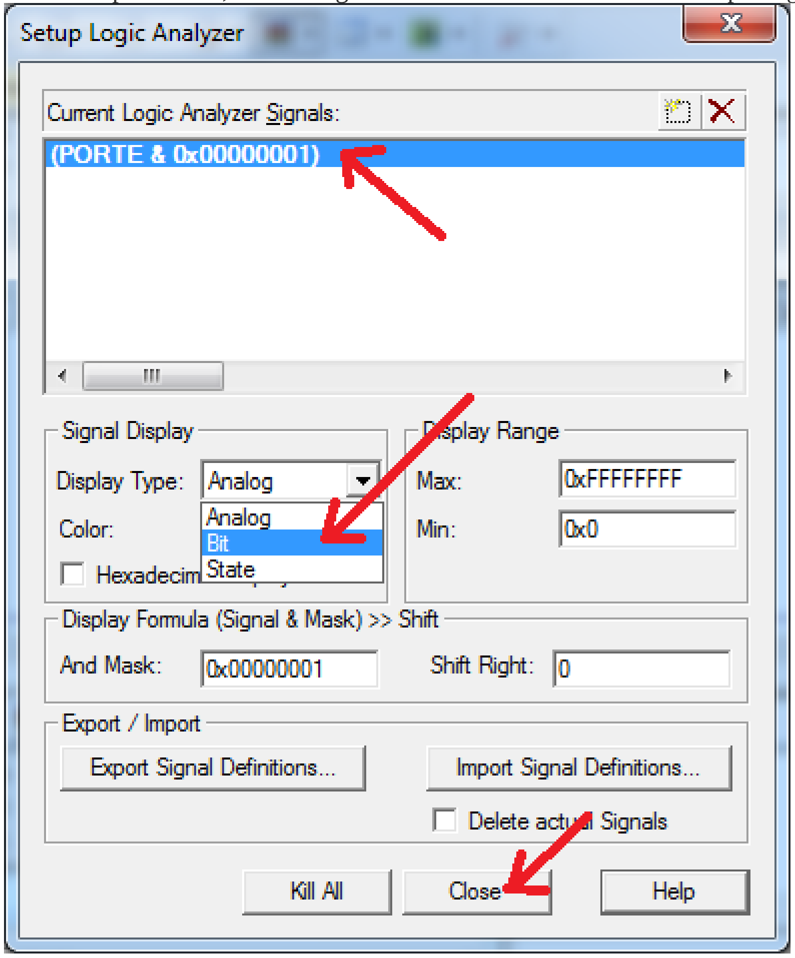


Setup Logic Analyzer

Current Logic Analyzer Signals:

PORTE.0

Click the port name, and change format to bit. You can add multiple signals, and when done click close.



Setup Logic Analyzer

Current Logic Analyzer Signals:

(PORTE & 0x00000001)

Signal Display

Display Type: Analog

Color:

Hexadecimal

Display Range

Max: 0xFFFFFFFF

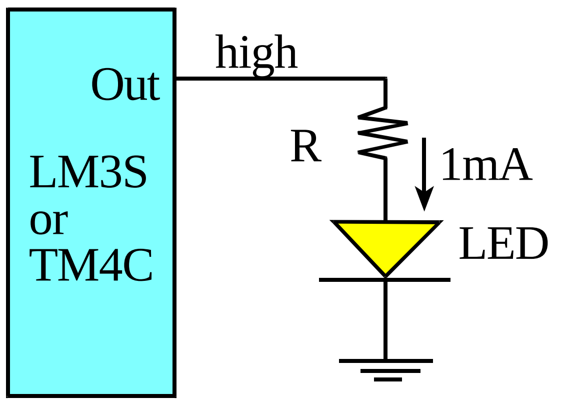
Min: 0x0

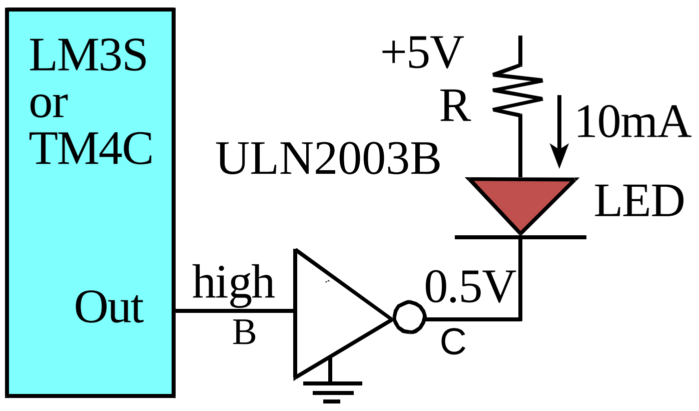
Display Formula (Signal & Mask) >> Shift

And Mask: 0x00000001

Shift Right: 0

Update: For the LED circuit, in this week try the circuit on the left first (Choose R = 220 ohm). Next week, we will implement circuit on the right.


high
Out
LM3S
or
TM4C
R
1mA
LED


LM3S
or
TM4C
+5V
R
10mA
ULN2003B
LED
high
B
Out
0.5V
C