Lab 3. Breathing LED with Switch/LED (Spring 2023)

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 all of ebook Chapter 3 or Sections 1.14, 2.4.6, 2.4.7, 3.1, 3.2, 3.3, and 3.6 of textbook
2. Read switch Data Sheet: http://users.ece.utexas.edu/~valvano/Datasheets/B3F-1059.pdf
3. Read LED Data Sheet: http://users.ece.utexas.edu/~valvano/Datasheets/LEDHLMP-4700.pdf
4. Look at the starter project for Lab 3 in the original ECE319K example 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. The last 12% of the lab grade is for making the LED “breathe”. 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 autograder will accept any frequency from 1 to 3 Hz. This will allow the same software to pass both on simulation and on the real board. 2 Hz variable duty-cycle means the LED is on for time *High* and off for time *Low*, where *High*+*Low* equals 500ms. The duty cycle is defined as *High*/(*High*+*Low*). Figure 3.1 illustrates a 2 Hz, 30% duty-cycle wave.

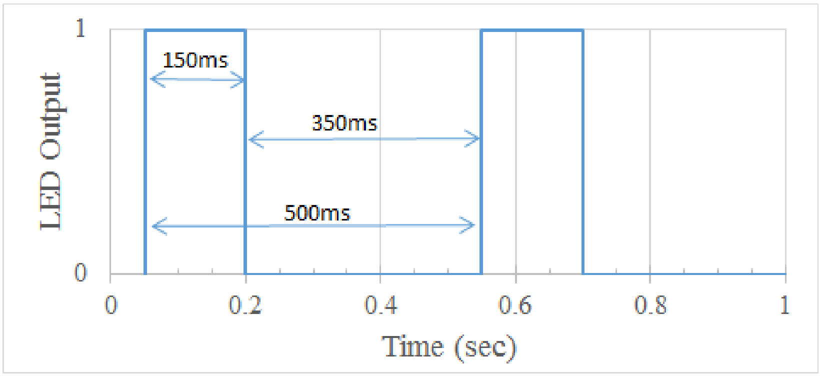


Figure 3.1. Example LED output.

The external hardware for the lab includes, two inputs and one output. Because you are interfacing the switches and LED externally you will need add resistors in the appropriate places. Switches and LEDs will be interfaced in positive logic. Similar to Lab 1, students are randomly assigned to solve one of the eight possible wiring configurations for Lab 2 using Port E.

- The **Change** input switch on PE1 or PE0
- The **Breathe** input switch on PE3 or PE2
- The **LED** output on PE5 or PE4

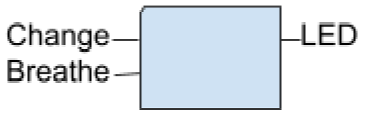
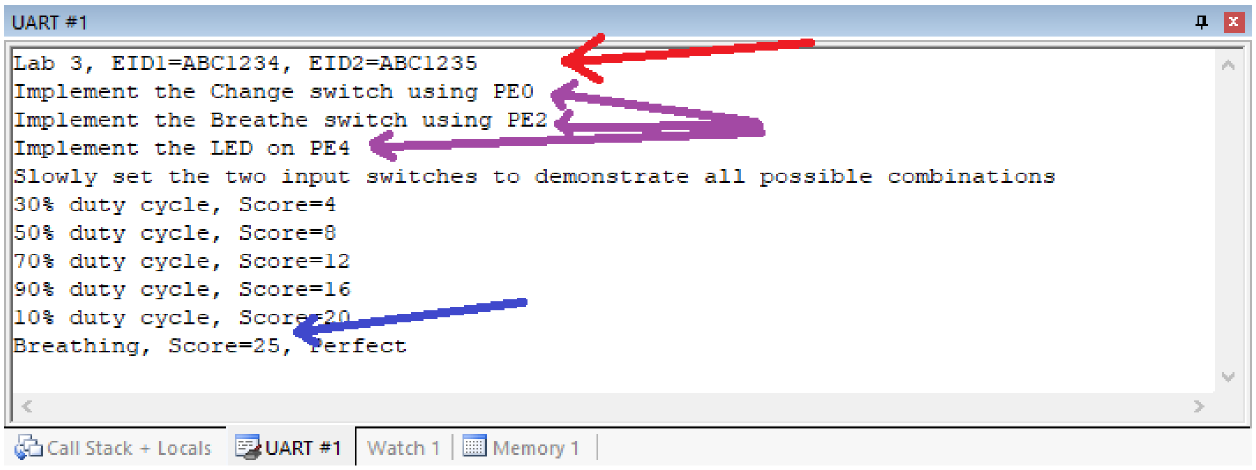


Figure 3.2. Block diagram of the Lab 3 system.

The exact requirements for your lab will be revealed in the UART window when you enter your EID into the project and then build and run the system.

Figure 3.3 shows a typical UART window when running the autograder. The red arrows are your EIDs. The purple arrows are the pin specifications and the blue arrow is the final score.



```
Lab 3, EID1=ABC1234, EID2=ABC1235
Implement the Change switch using PE0
Implement the Breathe switch using PE2
Implement the LED on PE4
Slowly set the two input switches to demonstrate all possible combinations
30% duty cycle, Score=4
50% duty cycle, Score=8
70% duty cycle, Score=12
90% duty cycle, Score=16
10% duty cycle, Score=20
Breathing, Score=25, Perfect
```

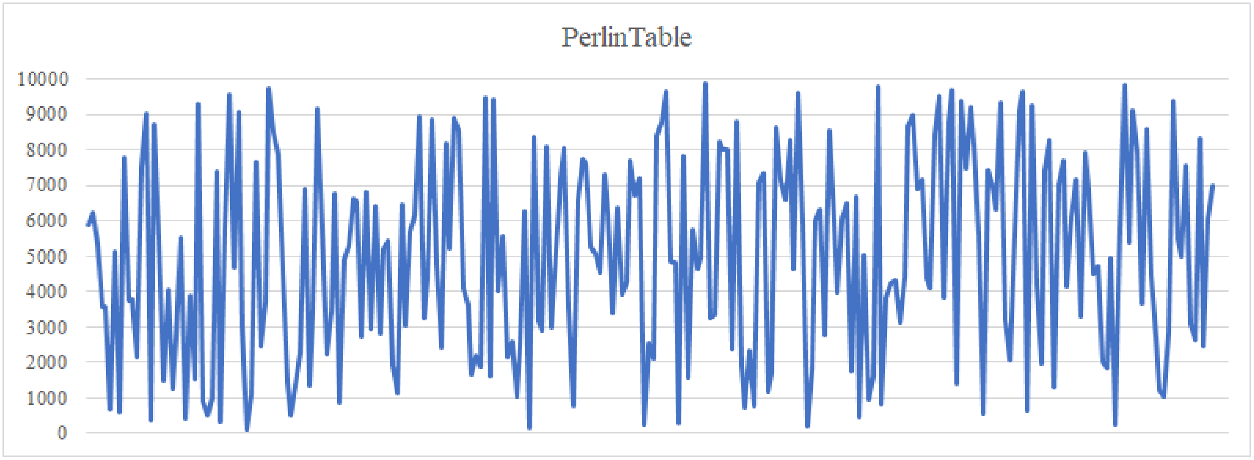
Figure 3.3. The UART window showing the full score. Your window will specify Spring 2023.

If you do not see the UART window, execute **View->SerialWindows->UART1** when in the debugger.
Overall functionality of this system is to operate as follows:

- Make the **LED** pin an output, and make the **Change** and **Breathe** pins input.
- 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).
- When the Change button 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.
- Implement a "breathing LED" when the **Breathe** button is pressed and held:
    a. As a minimum requirement, you must define an array of values and use the values to implement breathing. Two possible arrays, called **PerlinTable** and **SinTable**, are included in the starter code. You are however free to define an implement your own table. 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 the **Breathe** button 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%.
    c. While testing "breathing LED", the autograder will verify the LED frequency is about 100 Hz (so it looks continuous to the eye) and the duty-cycle changes.
    d. The autograder will not worry about the output if both switches are pressed.
    e. The autograder will not worry about the output during the time the **Change** button is being pushed

The following is a 256-entry table containing a Perlin shape

```
DCW 5880,6225,5345,3584,3545,674,5115,598,7795,3737,3775,2129,7527,9020,368,8713,5459,1478,4043,1248,2741,5536,406
DCW 3890,1516,9288,904,483,980,7373,330,5766,9555,4694,9058,2971,100,1095,7641,2473,3698,9747,8484,7871,4579,1440
DCW 521,1325,2282,6876,1363,3469,9173,5804,2244,3430,6761,866,4885,5306,6646,6531,2703,6799,2933,6416,2818,5230,5421
DCW 1938,1134,6455,3048,5689,6148,8943,3277,4349,8866,4770,2397,8177,5191,8905,8522,4120,3622,1670,2205,1861,9479
DCW 1631,9441,4005,5574,2167,2588,1057,2512,6263,138,8369,3163,2895,8101,3009,5153,7259,8063,3507,789,6570,7756,7603
DCW 5268,5077,4541,7297,6187,3392,6378,3928,4273,7680,6723,7220,215,2550,2091,8407,8752,9670,4847,4809,291,7833,1555
DCW 5727,4617,4923,9862,3239,3354,8216,8024,7986,2359,8790,1899,713,2320,751,7067,7335,1172,1708,8637,7105,6608,8254
DCW 4655,9594,5919,177,1784,5995,6340,2780,8560,5957,3966,6034,6493,1746,6684,445,5038,942,1593,9785,827,3852,4234
DCW 4311,3124,4426,8675,8981,6914,7182,4388,4081,8445,9517,3813,8828,9709,1402,9364,7488,9211,8139,5613,559,7412
DCW 6952,6302,9326,3201,2052,5651,9096,9632,636,9249,4196,1976,7450,8292,1287,7029,7718,4158,6110,7144,3316,7909
DCW 6838,4502,4732,2014,1823,4962,253,5842,9823,5383,9134,7948,3660,8598,4464,2665,1210,1019,2856,9402,5498,5000
DCW 7565,3086,2627,8330,2435,6072,6991
```

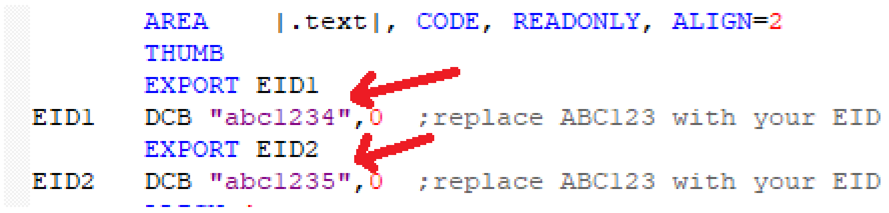


Procedure

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 first debug 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 **-dLaunchPadDLL**. Second, the **LaunchPadDLL.dll** file must be present in your Keil\ARM\BIN folder (already installed during the EE319Kware install step you performed for Lab 1).

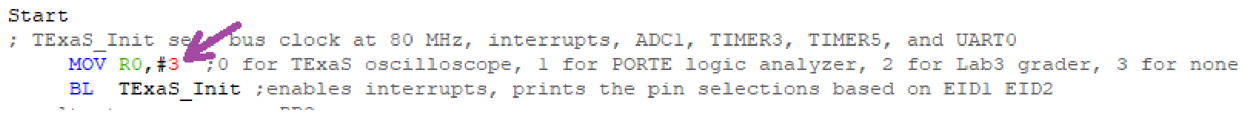
Please enter the EIDs of you and your lab partner into the **Lab3.s** file as shown with the red arrow in Figure 3.4. The two EIDs are used by a random number generator to select the pins you need to implement.



```
        AREA    |.text|, CODE, READONLY, ALIGN=2
        THUMB
        EXPORT EID1
EID1    DCB "abc1234",0  ;replace ABC123 with your EID
        EXPORT EID2
EID2    DCB "abc1235",0  ;replace ABC123 with your EID
```

Figure 3.4. Put the EIDs of you are your lab partner into your Lab3.s file.

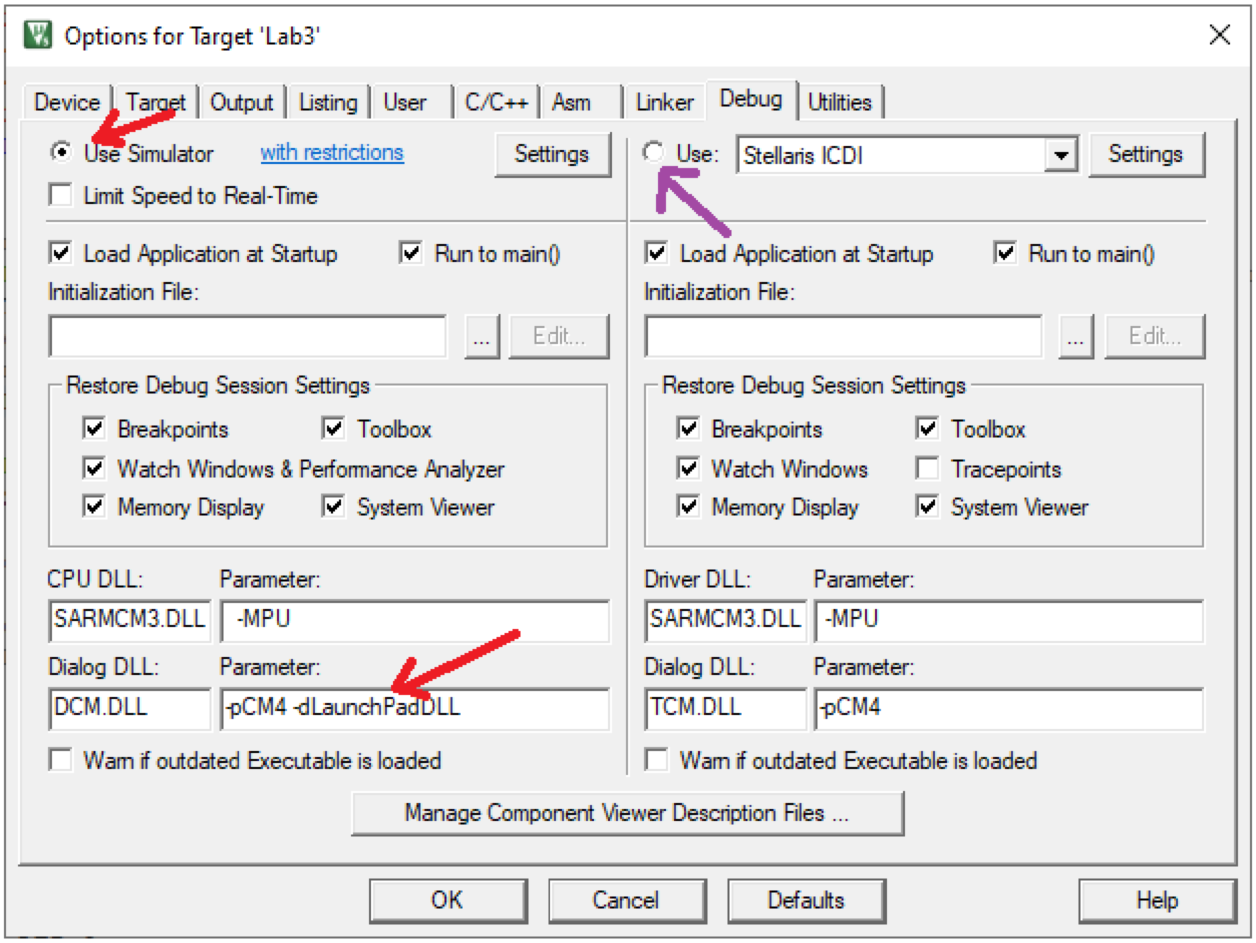
During initial development you should run **TExaS_Init** with the none (R0=3) selected (purple arrow).



```
Start
; TExaS_Init sets bus clock at 80 MHz, interrupts, ADC1, TIMER3, TIMER5, and UART0
     MOV R0,#3  ;0 for TExaS oscilloscope, 1 for PORTE logic analyzer, 2 for Lab3 grader, 3 for none
     BL  TExaS_Init ;enables interrupts, prints the pin selections based on EID1 EID2
```

Figure 3.5. There are four possible configurations for TExaS_Init.

You should rename the Lab3 starter folder to include your EID. Figure 3.6 shows the debug options. The red arrows show settings for simulation. The purple arrow shows how to configure the system for the real board.


Options for Target 'Lab3'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

[✓ Use Simulator] with restrictions Settings | Use: Stellaris ICDI Settings

☐ Limit Speed to Real-Time

☑ Load Application at Startup ☑ Run to main() | ☑ Load Application at Startup ☑ Run to main()

Initialization File: | Initialization File:

Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows & Performance Analyzer ☑ Memory Display ☑ System Viewer | Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows ☐ Tracepoints ☑ Memory Display ☑ System Viewer

CPU DLL: SARMCM3.DLL Parameter: -MPU | Driver DLL: SARMCM3.DLL Parameter: -MPU

Dialog DLL: DCM.DLL Parameter: -pCM4 -dLaunchPadDLL | Dialog DLL: TCM.DLL Parameter: -pCM4

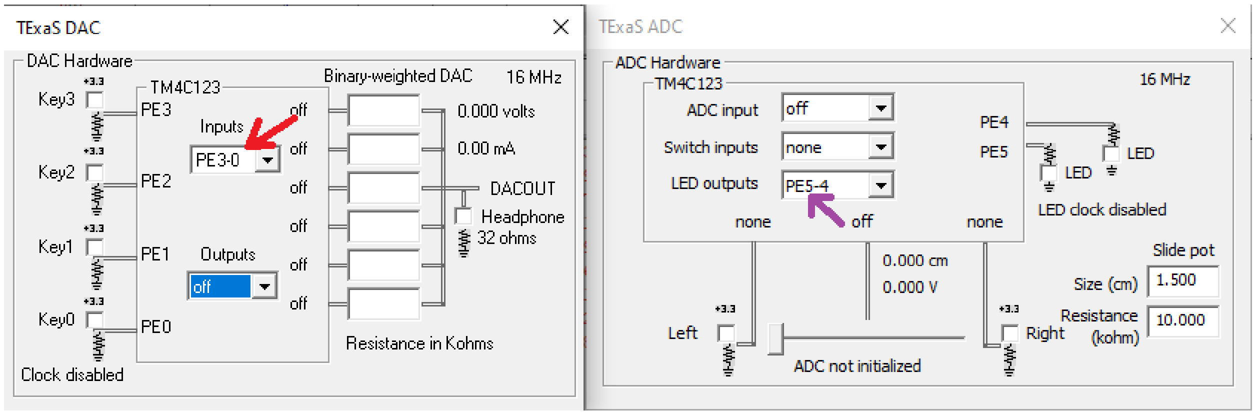
☐ Warn if outdated Executable is loaded | ☐ Warn if outdated Executable is loaded

Manage Component Viewer Description Files ...

OK | Cancel | Defaults | Help

Figure 3.6. Debug the software using the simulator (DCM.DLL -pCM4 -dLaunchPadDLL), same as Lab 1.

To interact with the I/O during simulation, make sure that **View->Periodic Window Update** is checked or the simulator will not update! See Figure 3.7. Execute the **Peripherals->TExaS DAC** command to get access to input switches. Set the input pins to Port E. Execute the **Peripherals->TExaS ADC** command to get access to output LED. Set the output pins also to Port E. When running the simulator, we check and uncheck bits in the I/O Port box to change input pins. We observe the output pin in the window.



TExaS DAC: Inputs: PE3-0; Outputs: off; 0.000 volts; 0.00 mA

TExaS ADC: ADC input: off; Switch inputs: none; LED outputs: PE5-4; 0.000 cm; 0.000 V; 16 MHz; Size (cm): 1.500; Resistance (kohm): 10.000

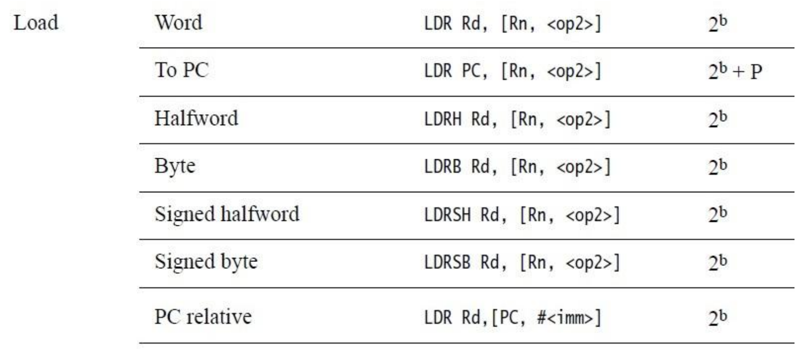
*Figure 3.7. Simulation of Port E. Your **Change Breathe** and **LED** pins will be specified when you enter your two EIDs and run the system.*

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 assembly code that implements a software delay loop. Assume R0 has the value **n** at the start of the code. The six instructions instructions are executed n times. The ALIGN 8 and the NOP instructions forces the BNE instruction to take 3 cycles regardless of whether it is running on the board or in simulation. Each time through the loop will be 8 cycles, so the delay will be 100ns***n**. Since it is very difficult to get an accurate time delays using this cycle counting method, we will use the hardware timer in Labs 5 and 6.

```
     ALIGN 8
;function to wait 100ns*R0
Delay
     NOP ;dummy operation
     NOP
     NOP
     NOP
     SUBS R0,R0,#1
     BNE  Delay
     BX   LR
```

(note: the ***BNE*** *instruction executes in 3 cycles on the simulator, but a different 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 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 your **LED** pin.

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

```
          MOV R0,#0 ; TExaS in voltmeter/scope mode
          BL TExaS_Init ; voltmeter, bus clock to 80 MHz
          Turn on Port E clock
          Make your LED pin an output
          Enable interrupts so the TExaS oscilloscope runs
     Loop
          Delay for 250ms
          Toggle your LED pin
          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:

```
          MOV R0,#0 ; TExaS in voltmeter/scope mode
          BL TExaS_Init ; activate voltmeter, bus clock to 80 MHz
          Turn on Port E clock
          Make your LED pin an output
          Enable interrupts so the TExaS oscilloscope runs
     Loop
          Set your LED pin high
          Delay for 150ms
          Clear your LED pin low
          Delay for 350ms
          B   Loop
```

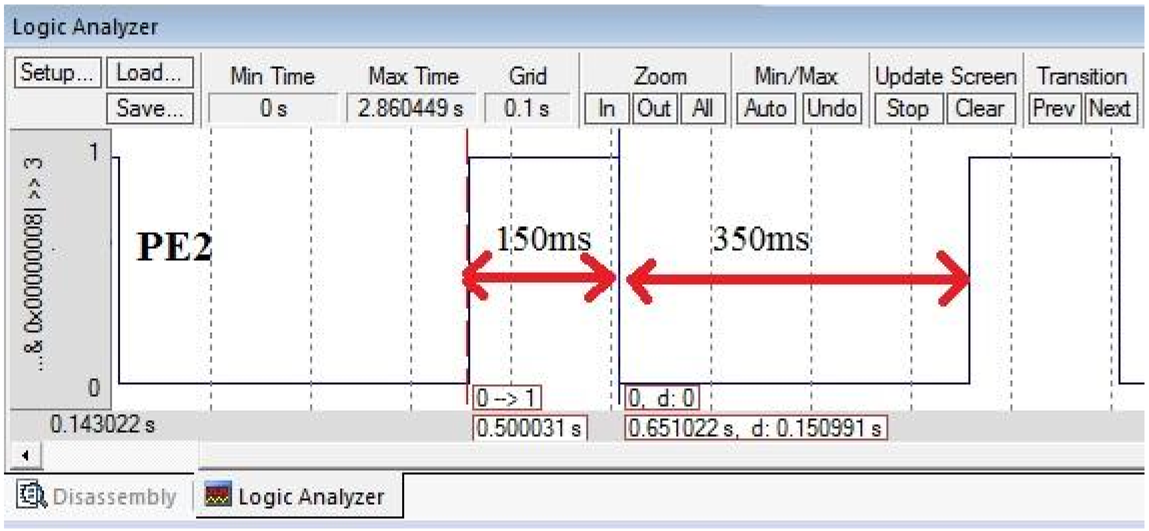


Figure 3.8. Pulse width modulation. Pin here is PE2, but your ***LED*** *pin will be specified when you enter your two EIDs and run the system.*

Stage 3: Add software to read the switch on your **Change** pin, 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 **Change** switch does not matter. So, while the **change** switch is being touched, you can stop the oscillations, or continue the oscillations, your choice. The duty-cycle 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 duty cycle is changed exactly once each time the **Change** switch is pressed and released.

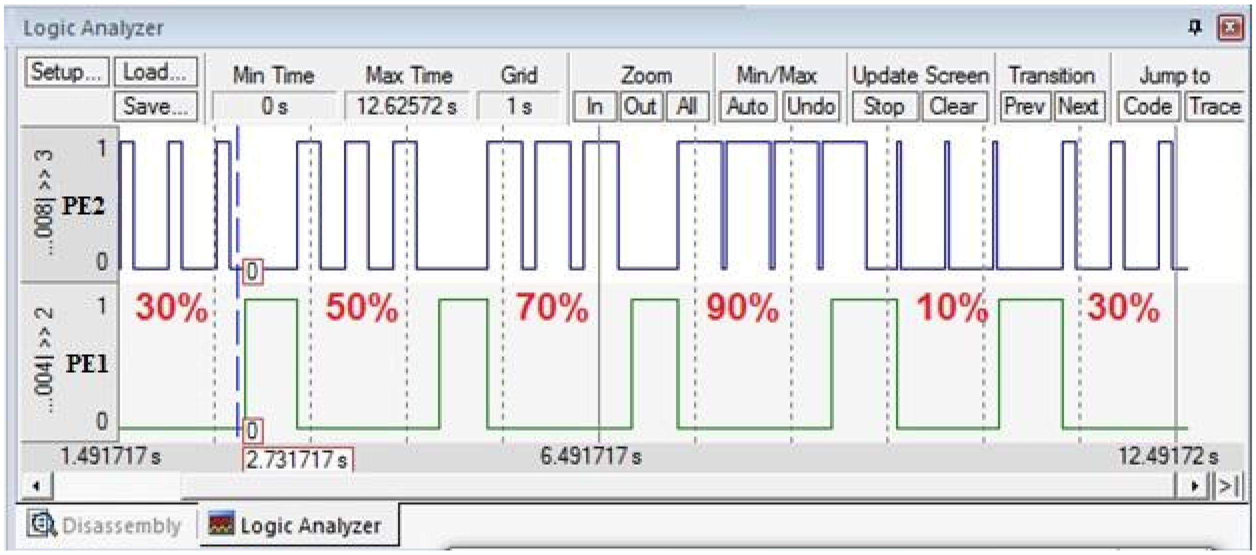


*Figure 3.9. Pulse width modulation. Pins here are PE1 and PE2, but your **Change** and **LED** pins will be specified when you enter your two EIDs and run the system.*

Stage 4: Build the circuit (see Figure 3.14) and check twice to make sure you have it correct. If in doubt take it to a TA during their office hours. 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. To verify the timing of the LED's toggling you can use an external Oscilloscope or TexasDisplay (this video shows a similar lab solution, but with different pins, different frequency and different duty cycles https://youtu.be/3Gi5e4iljuE).

Stage 5: At this point you implemented 88% of the requirements of this lab. Now you will add the breathing feature which is enabled when your **Breathe** switch is pressed and disabled when released. We want you to be creative in devising a solution to implement this feature. However, it is a requirement to put data into an array, and use the array to implement breathing. However, here are some ideas:

- A breathing LED increases in brightness gradually and once it reaches its full brightness it decreases its brightness gradually until 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-cycles. 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.

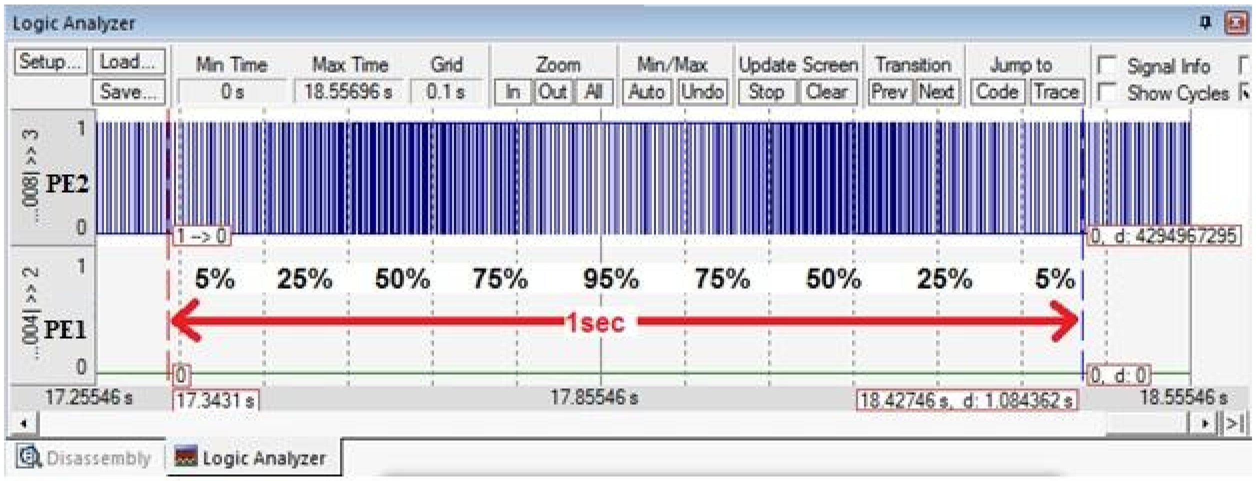


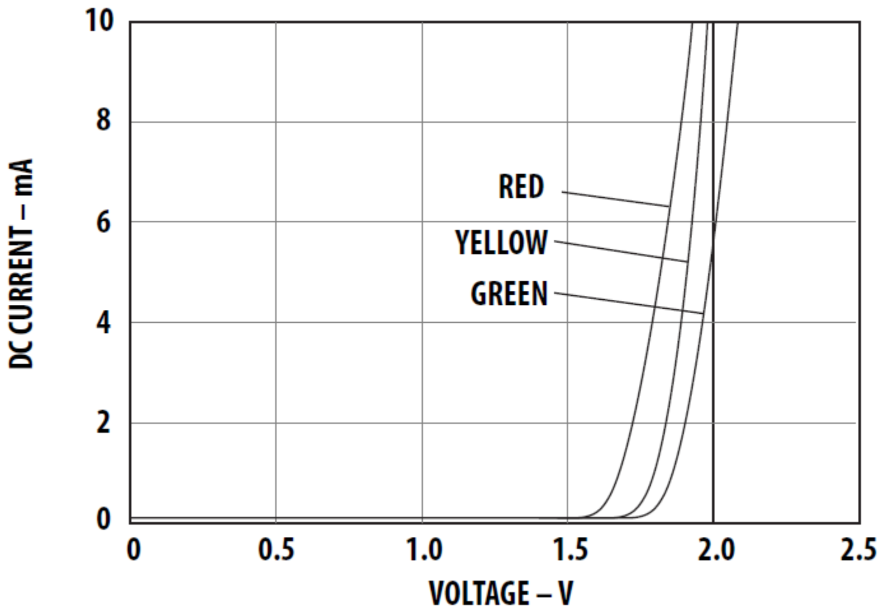
Figure 3.10. Breathing. Pins here are PE1 and PE2, but your ***Change*** *and* ***LED*** *pins will be specified when you enter your two EIDs and run the system.*

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. Download the datasheet for the LEDs

http://users.ece.utexas.edu/~valvano/Datasheets/LEDHLMP-4700.pdf

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. Because the LEDs require less than 8 mA, we will not need a driver circuit like a PN2222, 7406 or ULN2003. Normally, we would pick a desired operating point (V_d, I_d), and find a resistor value to establish that point. However, because the EE319K kit has six 470 ohm resistors, we will use these resistors for the six LEDs. Use the circuit and the curve to estimate the V_d, I_d resulting operating point for the Red LED with the 470 ohm resistor. For now, assume output high voltage (V_{OH}) will be 3.2V. To simplify the math you can approximate the Red LED curve as $I_d = 20*V_d-32$, where V_d is in volts and I_d is in mA for the values of V_d from 1.65 to 1.8V.



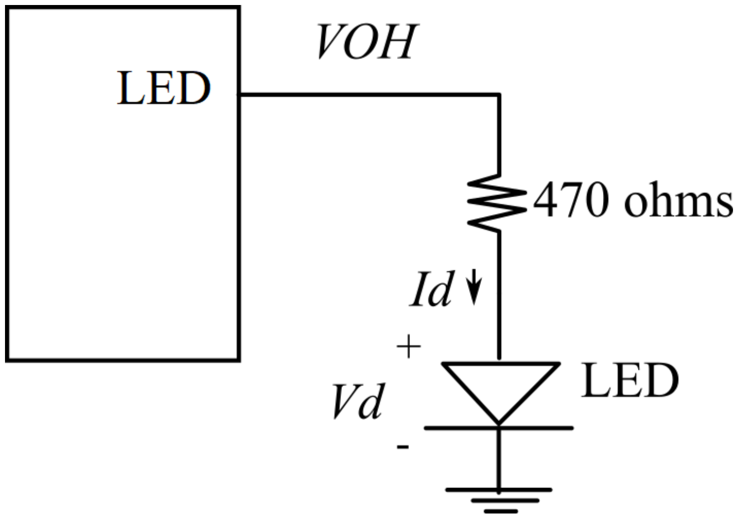


Figure 3.11. We wish to have the LED current around 3mA. The LED circuit will use a 470 ohm resistor.

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 3.12. 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 3.12, showing how the four pins are connected to the switch.

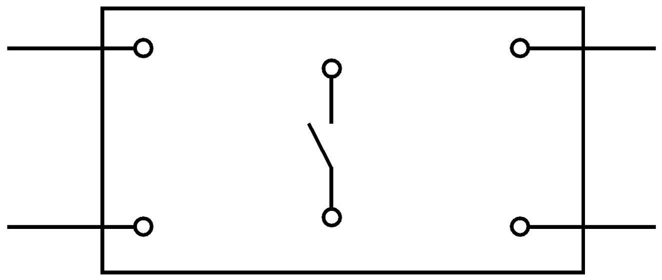


Figure 3.12. Connection diagram for the normally open switch.

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 3.13. The long rows of 50 holes along the outer sides of the protoboard are electrically connected. Some protoboards like the one in Figure 3.13 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. WE WILL NOT CONNECT TO VBUS (+5V) line in this lab.

If you are unsure about the wiring, please show it to your TA before plugging in the USB cable. 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.

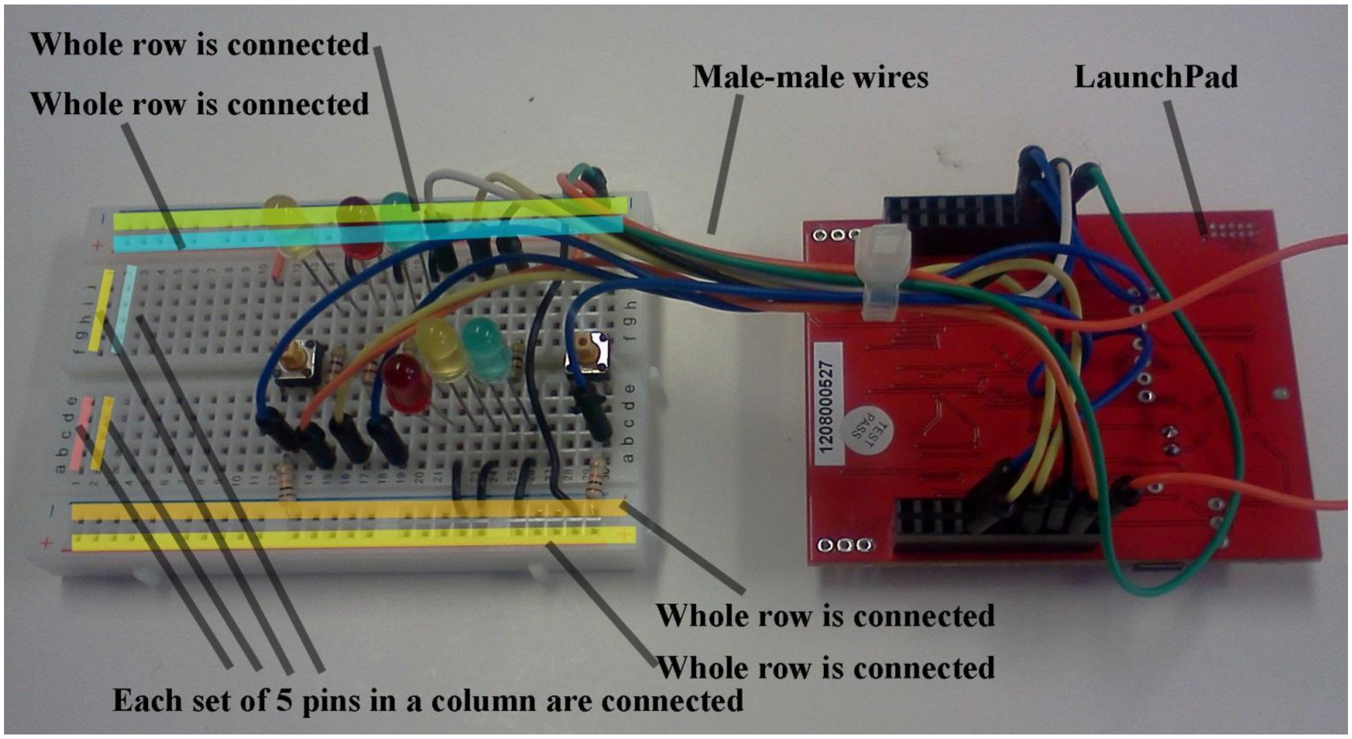


Figure 3.13. All the pins on each of the four long rows are connected. The 5 pins in each short column are connected. Use male-male wires to connect signals on the LaunchPad to devices on the protoboard. Make sure the ground wire is connected between the LaunchPad and your circuit. The +3.3V power can 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. We will not connect VBUS(+5V) in EE319K.

Part c - Construct Circuit

After the software has been debugged on the simulator, you will build the hardware on the real board. Please get your design checked by the TA 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 3.14. Figure 3.14 is what we mean in EE319K when we say "draw a circuit diagram".

ECE319K_Baseline_Schematic.sch is a starter file you should use to draw your hardware circuit diagram using the program Eagle. If you wish to use Eagle, complete **Pre-assignment 1** at https://docs.google.com/document/d/1cOHCwdiEaHmoeEZ_W6gog5rjur9hdnO_B3D8G8cSX3E/edit

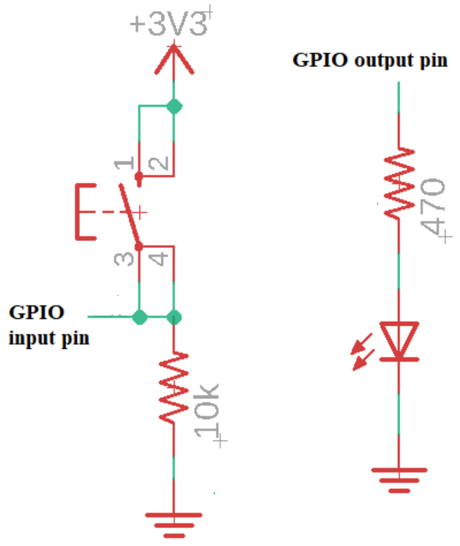


Figure 3.14. Eagle drawing showing Port E, 3.3V power, a 10kΩ resistor, a switch, an LED, a 470Ω resistor, and ground. Your circuit will have two switches and one LED. Your ***Change Breathe*** *and* ***LED*** *pins will be specified when you enter your two EIDs and run the system.*

Before connecting the two switches to **Port E** of the microcontroller, please take the measurements in Table 3.1 using your digital multimeter. The input voltage (V_{PE1}) is the signal that will eventually be connected to your **Change** pin. With a positive logic switch interface, the resistor current will be V_{PE1}/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_{PE1} voltage is low when the switch is not pressed and high when the switch is pressed.

Next, you can connect the input voltage to your **Change** pin 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 your **Change** pin. 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.

Next, you repeat the switch interface process with a second switch connected to your **Breathe** pin.

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 3.15. 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.

The resistor R1 in Table 3.2 is the 10k resistor in the switch circuit.

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_{PE1}		volts	Powered, but with switch not pressed (measured with voltmeter)
Resistor current		mA	Powered, but switch not pressed $I=V_{PE1}/R1$ (calculated and measured with an ammeter)
Input Voltage, V_{PE1}		volts	Powered and with switch pressed (measured with voltmeter)
Resistor current		mA	Powered and switch pressed $I=V_{PE1}/R1$ (calculated and measured with an ammeter)

*Table 3.1. Switch measurements. The table shows PE1, but your Lab 3 will connect the **Change** switch to PE1 or PE0 depending on your EID*

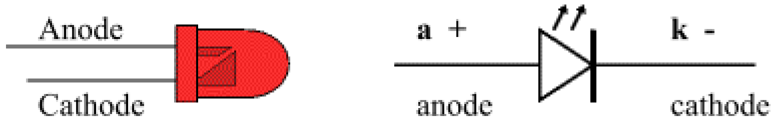


Figure 3.15. Left: a side view of an LED with leads labeled; Right: the corresponding circuit diagram

Take the measurements as described in Table 3.2. The resistor R19 is the 470 ohm resistor in the LED circuit. The R19 measurement occurs before R19 is inserted into the circuit. Single step your software to make your **LED** pin to output. Initially your **LED** pin will be low. So take four measurements with your **LED** pin low, rows 2,3,4,5 in Table 3.2. Then, single step some more until your **LED** pin 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 470Ω resistor, R19		ohms	with power off and disconnected from circuit (measured with ohmmeter)
3	TM4C123 Output, V_{PE4} input to 470Ω		volts	with **LED**= 0 (measured with voltmeter relative to ground). We call this V_{OL} of the TM4C123.
4	LED a+, V_{a+} Bottom side of R19 (anode side of LED)		volts	with **LED** = 0 (measured with voltmeter relative to ground). This measurement is also weird, because it too is floating.
5	LED voltage		volts	calculated as V_{a+} - V_{k-} (V_k is ground). (same as row 4)
6	LED current (off)		mA	calculated as (V_{OL} - V_{a+})/R19
7	TM4C123 Output, V_{PE4} input to 470Ω		volts	with **LED** = 1 (measured with voltmeter relative to ground). We call this V_{OH} of the TM4C123. We previously assumed this was 3.2V.
8	LED a+, V_{a+} Bottom side of R19 (anode side of LED)		volts	with **LED** = 1 (measured with voltmeter relative to ground)
9	LED voltage		volts	calculated as V_{a+} - V_{k-}(V_k is ground). Same as row 8
10	LED current (on)		mA	calculated as (V_{OH} - V_{a+})/R19
11	LED current (on)		mA	measured with an ammeter (should be similar to row 10)

Table 3.2. LED measurements (assuming the R19 is the 470 Ω resistor in Figure 3.14). The table shows PE4, but your Lab 3 will connect the LED to PE4 or PE5 depending on your EID.

Compare the three LED on currents: mathematical analysis in Part b) using the simplified equation as I_d= 20*V_d-32, calculated current in row 10, and measured current in row 11.

Part d - Debug Hardware + Software

Debug your combined hardware and software system on the real-board. There is an auto-grader that can be activated by setting R0=2 when calling TExaS_Init. The maximum score is 25.

When running on the real board, you need to run TExaSdisplay to see the UART output.

Demonstration

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

There are grading sheets for every lab so you know exactly how you will be evaluated. You will show the TA your program operation on the actual TM4C123 board. The TA 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. We may test to see if you can measure voltage, current and/or resistance with your meter (so bring your meter to the demonstration).

Please make note of which TA checked you out. The name of the TA will greatly help you when resolving any grading issues later.

Do all these well in advance of your checkout

1. **Signup for a time with a TA. If you have a partner, then both must be present**
2. **Upload your software to canvas, make sure your names are on all your software**
3. **Upload your one pdf with deliverables to Canvas**

Do all these during the TA checkout meeting

1. **Have your one pdf with deliverables open on your computer**
2. **Have Keil Lab 3 open so TA can ask about your code**
3. **Start promptly, because we are on a schedule. If you have a partner, then both must be present**
4. **Demonstrate lab to TA**
5. **Answer questions from TA to determine your understanding**
6. **TA tells you your score (later the TA will upload scores to Canvas)**

Deliverables

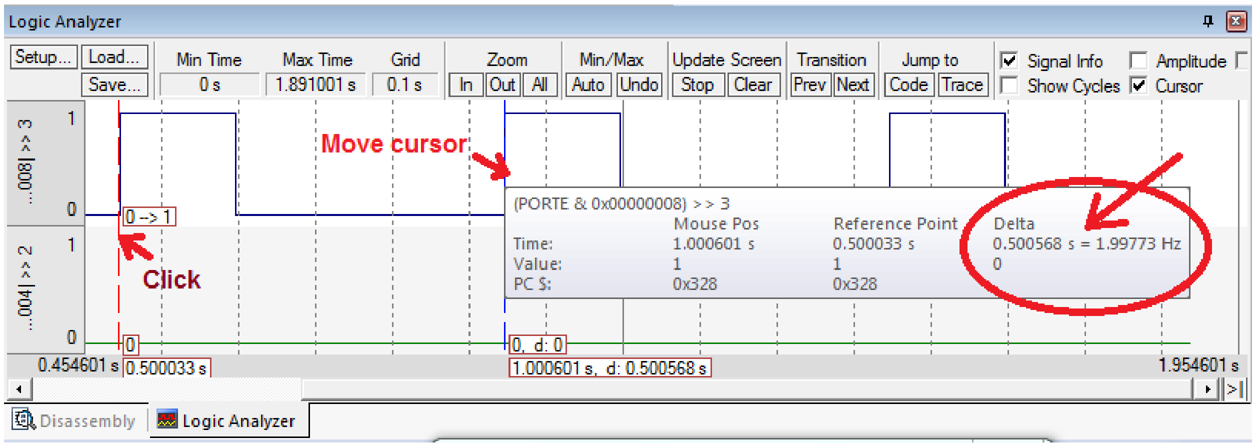
Upload your main.s file to Canvas. Combine the following components into one pdf file and upload this file also to Canvas. UPLOAD ONLY ONE COPY PER TEAM (names on both). Have the pdf file and Keil open on the computer during demonstration

1. Your names, professors, and EIDs
2. Circuit diagram (hand-drawn or optionally using Eagle), like Figure 3.14
3. Estimated LED voltage and current using the data sheet
4. Screenshot of Keil logic analyzer showing your debugging, like Figure 3.9
5. Screenshot of Keil logic analyzer showing your breathing, like Figure 3.10
6. Switch measurements (Table 3.1)
7. LED measurements (Table 3.2). Compare the three LED on currents: mathematical analysis in Part b) using the simplified equation as $I_d = 20*V_d$-32, calculated current in Table 3.2 row 10, and measured current in Table 3.2 row 11.
8. UART window of autograder of standard part (TAs will need to see breathing on the board). You can give either simulation or real board autograder.

Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

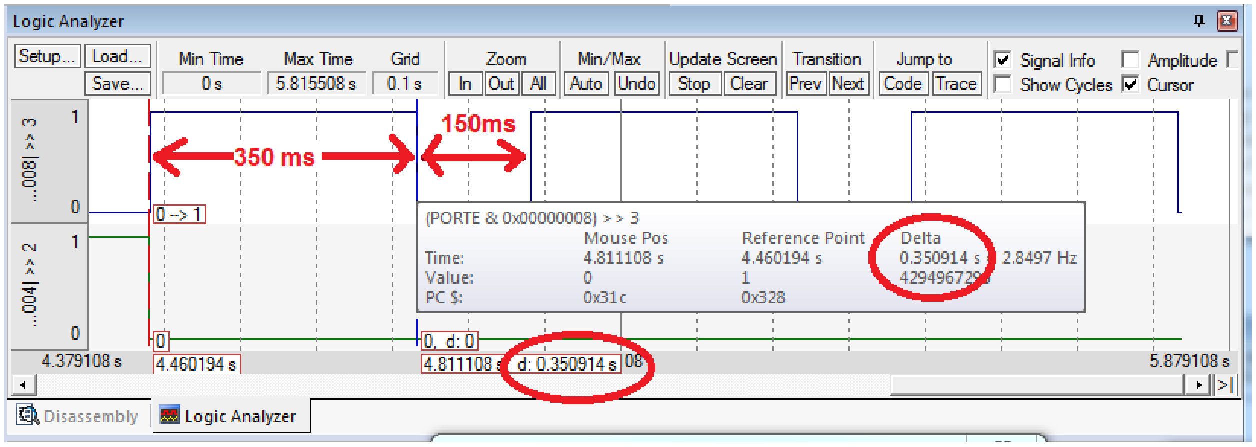
Precautions to avoid damaging your system

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. If you use Port C, be friendly. PC3,PC2,PC1,PC0 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.



```
(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
```

Figure 3.16. Simulation of Lab 3, showing PE2 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 3.17. Simulation of Lab 3, showing PE2 output toggling at 2 Hz with 70% duty-cycle.

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. 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 current mode) 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.

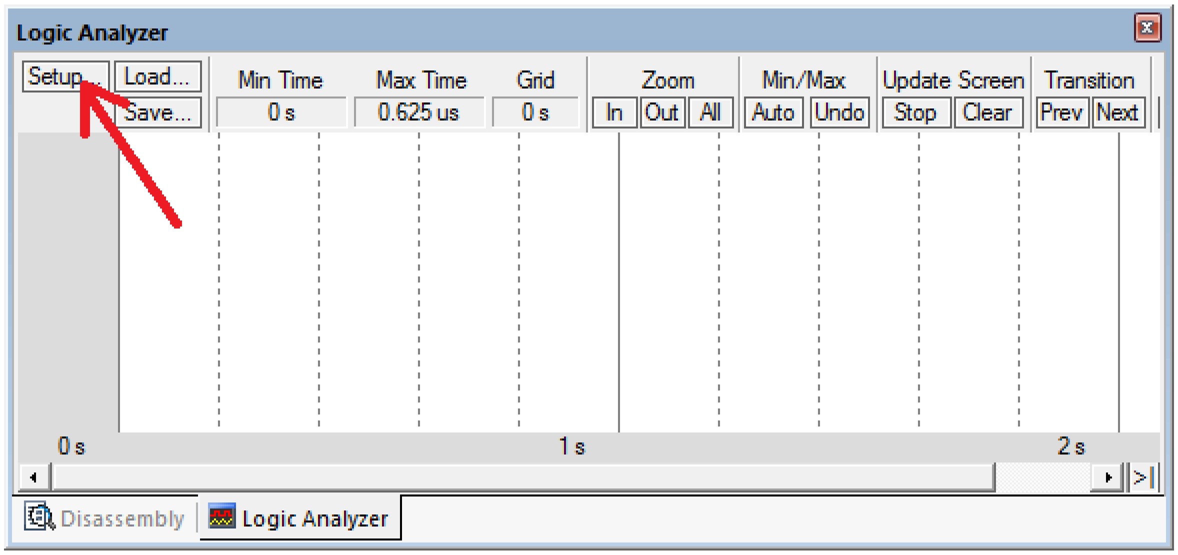
   With Keil you can debug your circuit in real time. If you click under Projects --> Options for Target --> Debug and use the **Stellaris ICDI** instead of the simulator you can press debug and actually step through your code and watch how it affects your circuit. Think about when in your program should PE2 be set to zero and jump to it. You can now measure these elements knowing that PE2 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?

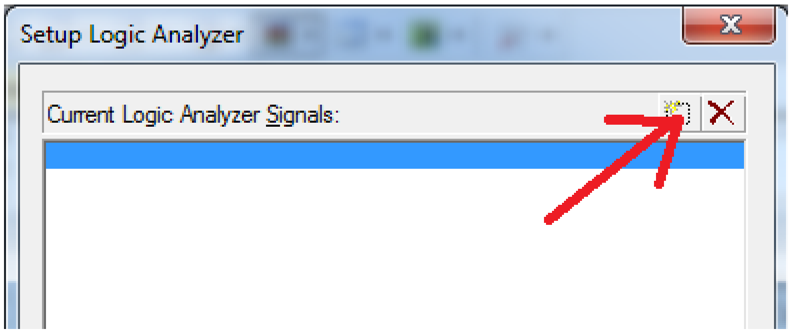
   Before moving on to circuits, make sure your program works first on the simulator. Once that is checked, make sure everything is connected securely. Circuit problems come from crappy wires most of the time.. are you using your own wires or wires from the check out desks? A multimeter will be your best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

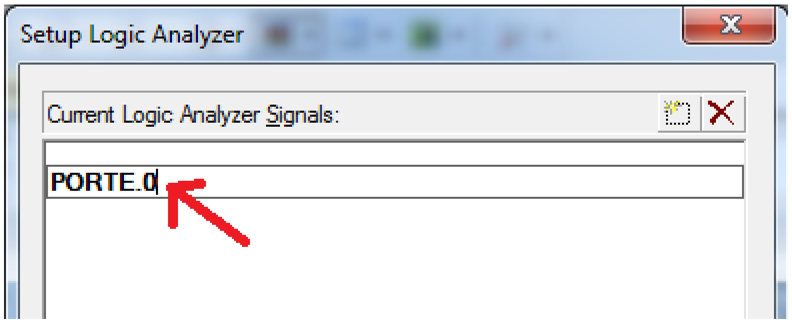
   This happens when you run the simulator with the logic analyzer, run the debugger on the board, and then run the simulator again. Running on the real board clears the logic analyzer configuration. You will have to execute **Setup** again. First, open the logic analyzer window and click Setup.



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

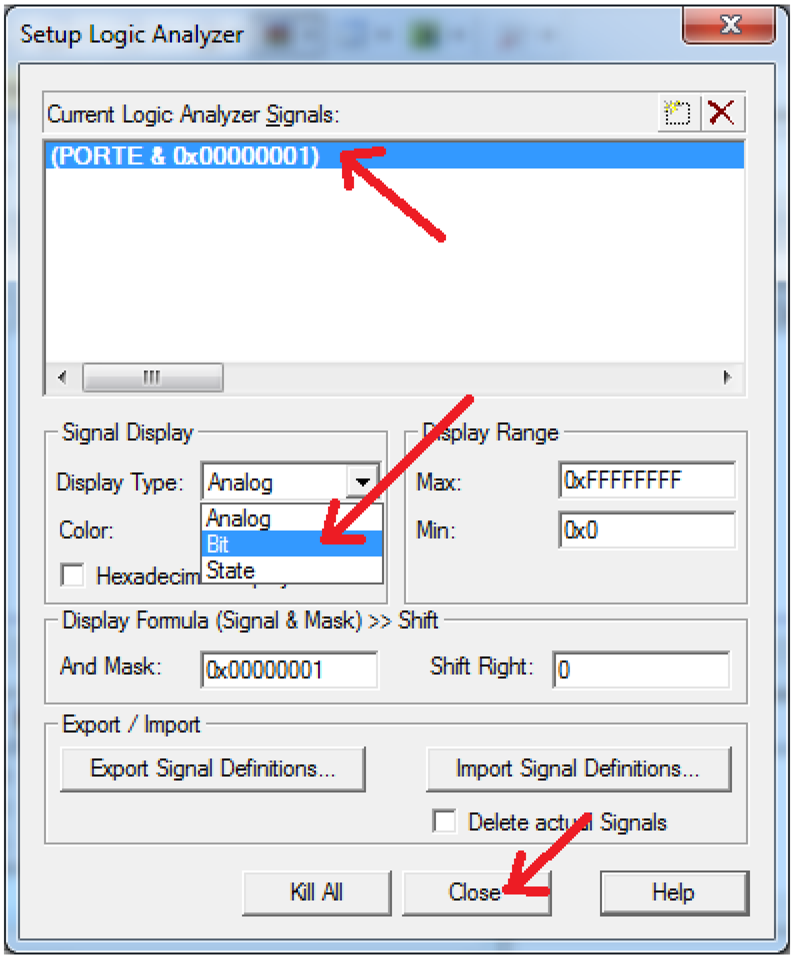


Add the port pin you want and then enter



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:

Hexadecim

Analog

Bit

State

Display Range

Max: 0xFFFFFFFF

Min: 0x0

Display Formula (Signal & Mask) >> Shift

And Mask: 0x00000001

Shift Right: 0

Export / Import

Export Signal Definitions...

Import Signal Definitions...

Delete actual Signals

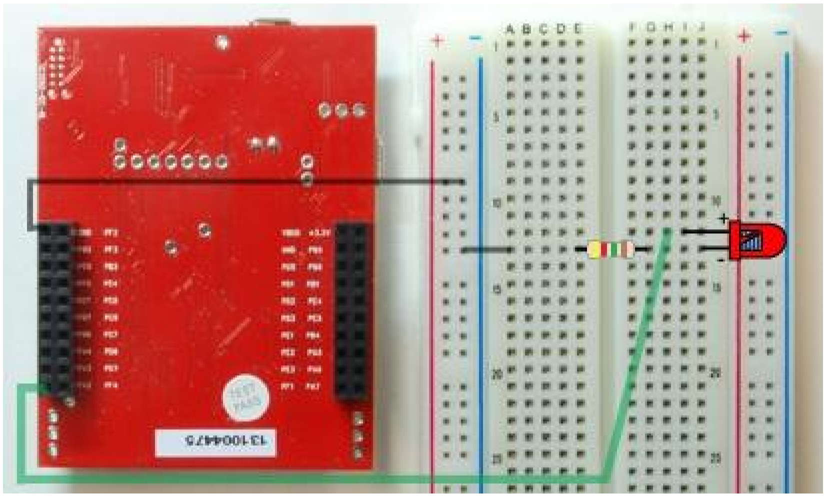
Kill All

Close

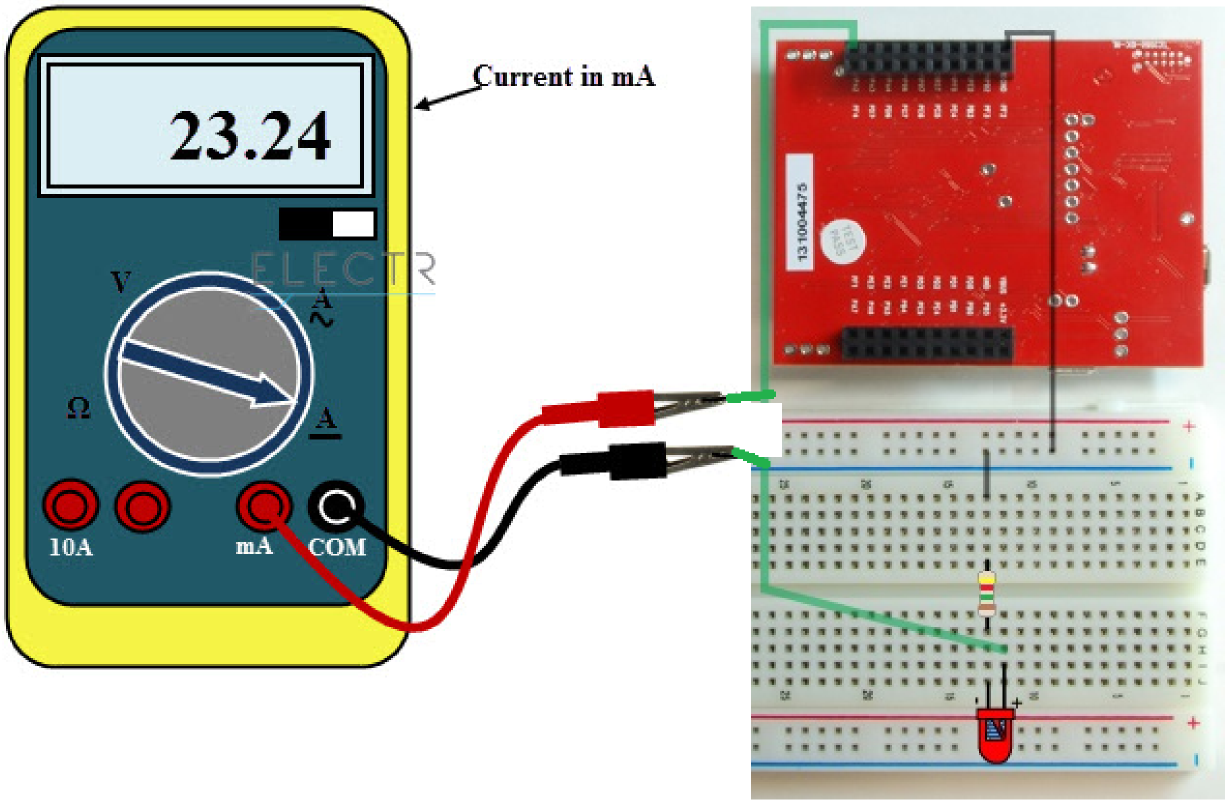
Help

Tutorial on how to measure current through an LED

1) Start with a circuit that is operational. For example, output to the GPIO so the LED is on



2) Turn off the power, break the circuit, and connect the +probe and -probe of the DVM current meter in place of the break. The meter must be in current mode and not voltage mode. Most DVMs require you to connect the probes to the meter in a special place to measure current



https://www.electronicshub.org/current-measurement-using-multimeter/

3) Turn on the power (and make the GPIO pin high) and the system should run the same (LED on)