Lab 5. Traffic Light Controller (Spring 2023)

Outline:

Because of the ice storm and dropping Lab 4, we simplified Lab 5. The new Lab 5 starter project is posted on Canvas in the files section https://utexas.instructure.com/files/70514708/download?download_frd=1 .

An alternate place to download is https://www.dropbox.com/s/zv77vo9ad8mebhu/Lab5new.zip?dl=1

Unzip this project and place it adjacent to the other Lab projects.

M1 Macintosh students should use SysTick; you can copy SysTick code from ebook section 5.2 (program 5.2), and ignore the calls to dump,

Preparation

Book: Read all of ebook Chapters 4 and 5 (skip 5.5 on stepper motors) or textbook Chapter 5.
Review Data Sheets from Lab 3:
- http://users.ece.utexas.edu/~valvano/Datasheets/B3F-1059.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/LEDHLMP-4700.pdf

Open, add your two EIDs, build, debug, and run the project (see UART window for more details)
Lab5new new Lab 5 starter project

Youtube Tutorial:
https://www.youtube.com/playlist?list=PLyg2vmIzGxXEle4_R2VA_J5uwTWktdWTu
http://youtu.be/kgABPjf9qLI

Purpose
This lab has these major objectives:
1. The understanding and implementing linked data structures
2. Learning how to create a software system
3. The study of real-time synchronization by designing a finite state machine controller

Software skills you will learn include advanced indexed addressing, linked data structures, creating fixed-time delays using the software delay from Lab 3 ~~SysTick timer~~, and debugging real-time systems.

C Shenanigans
All software in this lab must be developed in C. The Lab 5 starter file has the appropriate connections to the Lab 5 simulator (LaunchPadDLL.dll). The call to **TExaS_Init** will activate 80 MHz PLL. Execute
TExaS_Init(GRADER);
to activate the grader; other options include scope and logic analyzer.

Design Overview
Consider a 2 street intersection as shown below. There are two one-way streets, labeled ***South*** and ***West*** for southbound and westbound cars to travel on.

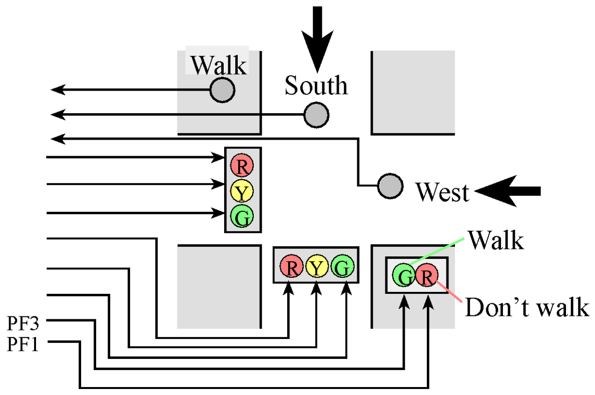


Figure 5.1. Traffic Light Intersection (3 inputs and 9 outputs). Walk light is white LED.

- **Input from switches**
  1. ***South Sensor:*** This sensor or button be activated (positive logic), if one or more cars are near the intersection on the South road. You will simulate this on the hardware by interfacing a button and holding it down. You will push and hold the button for as long as there are cars on the South road.
  2. ***West Sensor:*** This sensor or button be activated (set to logic 1), if one or more cars are near the intersection on the West road. You will simulate this on the hardware by interfacing a button and holding it down. You will push and hold the button for as long as there are cars on the West road.

3. ***Pedestrian sensor:*** Will be activated when a pedestrian is waiting to cross in any direction. You will simulate this effect by pushing a button. You will simulate this on the hardware by interfacing a button and holding it down. You will push and hold the button for as long as there are pedestrians wishing to cross. *This is not a traditional walk button that is pushed and released to request a walk signal.*

- ## Outputs to LED

  You will use 9 outputs from your microcomputer that control the traffic lights.

1. ***(6 LEDS) South and West, R/Y/G Lights***: You will have to interface a total of 6 total LED's for the South and West's, red, yellow and green lights. To reduce accidents, there should be a short time with all lights red, as it transitions from yellow on one road to green on the other road.

2. ***(3-color LED) Walk light:*** This will be the white LED (PF3,2,1) on the LaunchPad, and will be turned white when pedestrians are allowed to cross.

   The walk sequence should be realistic, showing three separate conditions:
   1. ***Walk:*** Your white walk light should be on signifying the pedestrians may cross. The two road signals should be red while pedestrians are walking.
   2. ***Warning***: Flash your red don't walk LED (PF1) at least twice signifying that pedestrians need to hurry up (e.g., white-red-off-red-off-red). You may decide how long you want to flash it. The two road signals should be red while pedestrians are hurrying up to cross the street. There should be a short time with all lights red, as it transitions from warning to allowing cars to pass.
   3. ***Don't Walk:*** Your red don't walk LED should be on and constant.

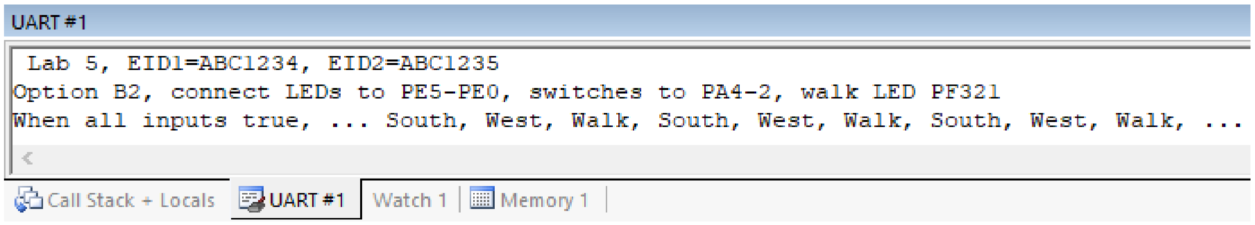
   All other inputs and outputs must be built on the protoboard. The Lab 5 simulator is implemented in **LaunchPadDLL.dll**.

Procedure

The basic approach to this lab will be to first develop and debug your system using the simulator and then interface with actual lights and switches on a physical TM4C123. As you have experienced, the simulator requires a different amount of actual time as compared to simulated time. ~~On the other hand, the correct simulation time is maintained in the SysTick timer, which is decremented every cycle.~~ The simulator speed depends on the amount of information it needs to update into the windows and the speed of your personal computer. Because we do not want to wait the minutes required for an actual intersection, the cars in this traffic intersection travel much faster than real cars. In other words, you are encouraged to adjust the timing so that the operation of your machine is convenient for you to debug and for the TA to observe during demonstration.

Part a - Pin/Port Selection

Unless your LaunchPad has broken pins, the Lab 5 autograder will select which port pins you will use for the inputs and outputs. Add your EIDs to the Lab5.c file, and run the project to see the exact requirements for you. Figure 5.2 shows one of 16 possibilities.



```
 Lab 5, EID1=ABC1234, EID2=ABC1235
Option B2, connect LEDs to PE5-PE0, switches to PA4-2, walk LED PF321
When all inputs true, ... South, West, Walk, South, West, Walk, South, West, Walk, ...
```

Figure 5.2. The UART1 window will show the specific pins for your Lab 5. Your window will say Fall 2022.

The "don't walk" and "walk" lights must be PF1, PF2 and PF3. If your LaunchPad has broken pins, contact your TA for other possibilities for how you can connect the six LEDs that form the traffic lights. Figure 5.3 shows the simulator window used to display with the traffic light outputs. Table 5.1 lists the possible connections for the six LEDs.

Stoplight Signal	Option A	Option B
Red west	PB5	PE5
Yellow west	PB4	PE4
Green west	PB3	PE3
Red south	PB2	PE2
Yellow south	PB1	PE1
Green south	PB0	PE0

Table 5.1. Possible ports to interface the traffic lights (PF321=red don't walk, PF321=white walk).

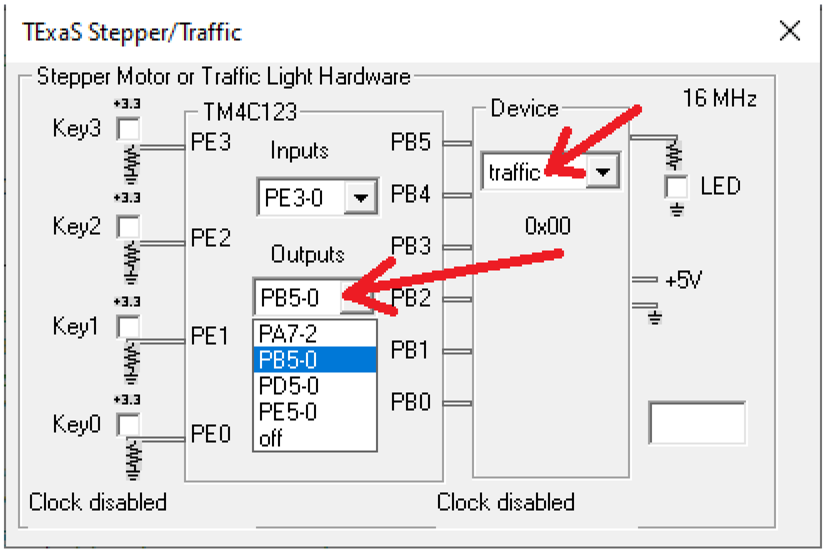


*Figure 5.3. Select the device as traffic and select the port for your 6 outputs. Your **LED** pins will be specified when you enter your two EIDs and run the system*

Table 5.2 shows you possibilities for how you can connect the three positive logic switches that constitute the input sensors. Figure 5.4 shows the simulator window used to input traffic light sensors.

Stoplight Sensor	Option 1	Option 2	Option 3	Option 4	Option 5	Option 6
Walk sensor	PA4	PA5	PC6	PC7	PE2	PE3
South sensor	PA3	PA4	PC5	PC6	PE1	PE2
West sensor	PA2	PA3	PC4	PC5	PE0	PE1

Table 5.2. Possible ports to interface the sensors.

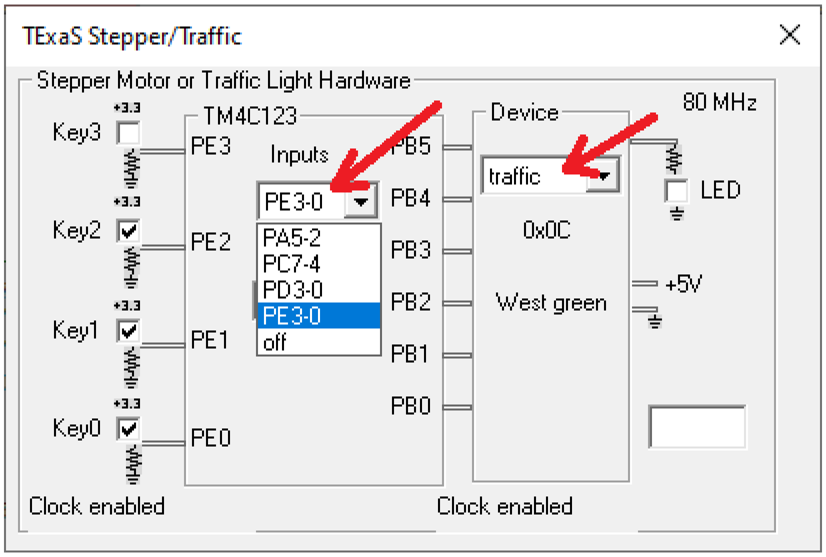


*Figure 5.4. Select the device as traffic and select the port for your 3 inputs. Your **switch** pins will be specified when you enter your two EIDs and run the system*

Part b - Light sequence

If there are no inputs active, the system should remain in a safe pattern. One safe pattern is green on one road, red on the other road, and don't walk. Another safe pattern is red on both roads, and walk.

If one input is active, the system should move in a safe manner, so that one input is served continuously. A safe manner means a traffic light goes green to yellow to red. There is traffic light red on both roads in between yellow on one road and green on other road. Similarly the don't walk light flashes off and on two or more times before a road sees a green.

If two inputs are active, the system should move in a safe manner, so that both inputs are served alternately.

If all three input are active, the system should move in a safe manner, so that all three inputs are served sequentially. However, the desired sequence will be specified by the Lab5 autograder. There are two possible sequences.

- … South, Walk, West, South, Walk, West, South, Walk, West, …
- … South, West, Walk, South, West, Walk, South, West, Walk, …

Leaving the walk state the PortF output should see this pattern: white,red,off,red,off,red. The number of times it goes off, red, must be at least twice. The autograder will ask you to set all three inputs, wait until your system enters the South state, and then it will check the sequence. So where you start in the sequence should not matter. It will check for an "all stop" condition

- between yellow on one road and green on another road.
- between yellow on one road and walk.
- between walking and green on any road.

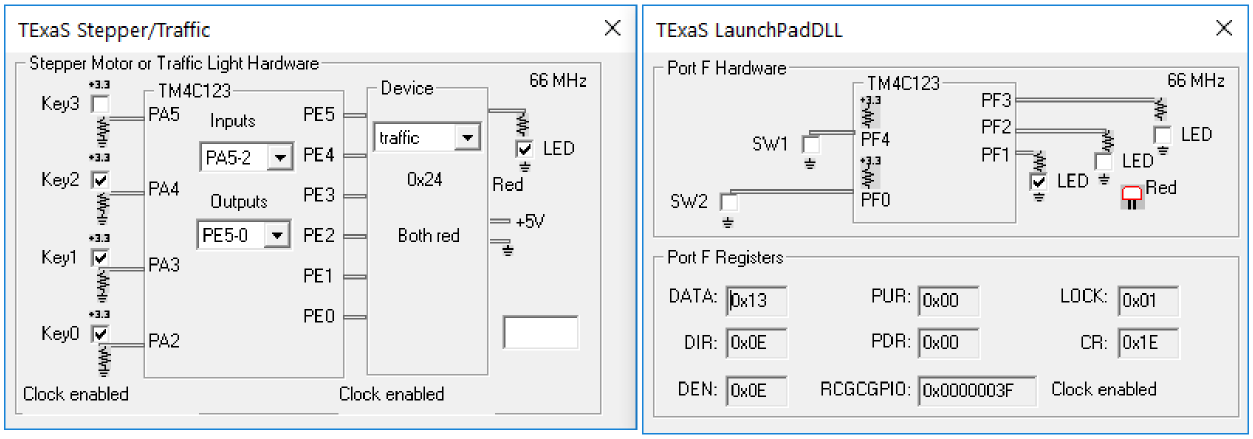
Part c - FSM Design

Design a finite state machine that implements a traffic light system. Include a picture of your state transition graph in the deliverables showing the various states, inputs, outputs, wait times and transitions. It is advisable that you show your FSM to a TA to verify your FSM design before continuing to code.

It may be helpful to look at the Civil_Engineering and Also the Tips and Tricks Sections below for guidance.

Part d - Debug C Code In Simulation

Write and debug the C code that implements the traffic light control system. In simulation mode, capture logic analyzer screenshots showing the operation of your traffic light when all three inputs are active, like Figure 5.5. Your performance grade on this lab will be determined by the autograder. *However, it is possible the autograder has bugs, so if you are sure your system works, contact a TA.*

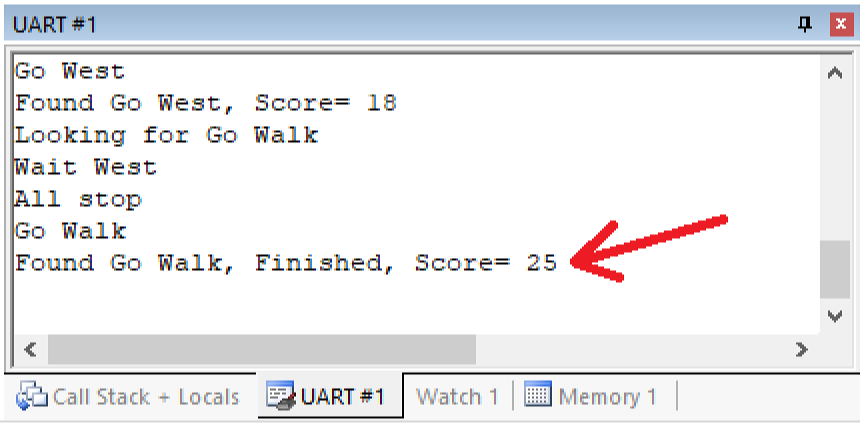


Inputs: PA5-2
Outputs: PE5-0
traffic
0x24
Both red

DATA: 0x13 PUR: 0x00 LOCK: 0x01
DIR: 0x0E PDR: 0x00 CR: 0x1E
DEN: 0x0E RCGCGPIO: 0x0000003F

Figure 5.5. Example output with all stop.

In GRADER mode, activate all three inputs and watch the UART window as the grader tests your FSM.



```
Go West
Found Go West, Score= 18
Looking for Go Walk
Wait West
All stop
Go Walk
Found Go Walk, Finished, Score= 25
```

Figure 5.6. UART windows with full score of 25 in simulation.

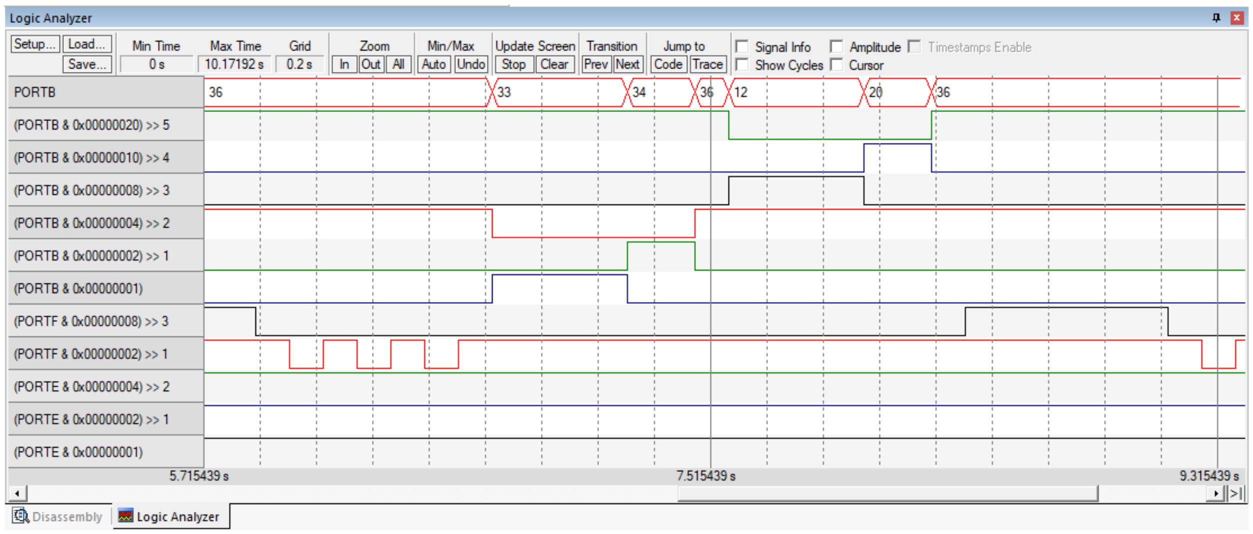


Figure 5.7. Simulation mode logic analyzer trace showing all states in one cycle.

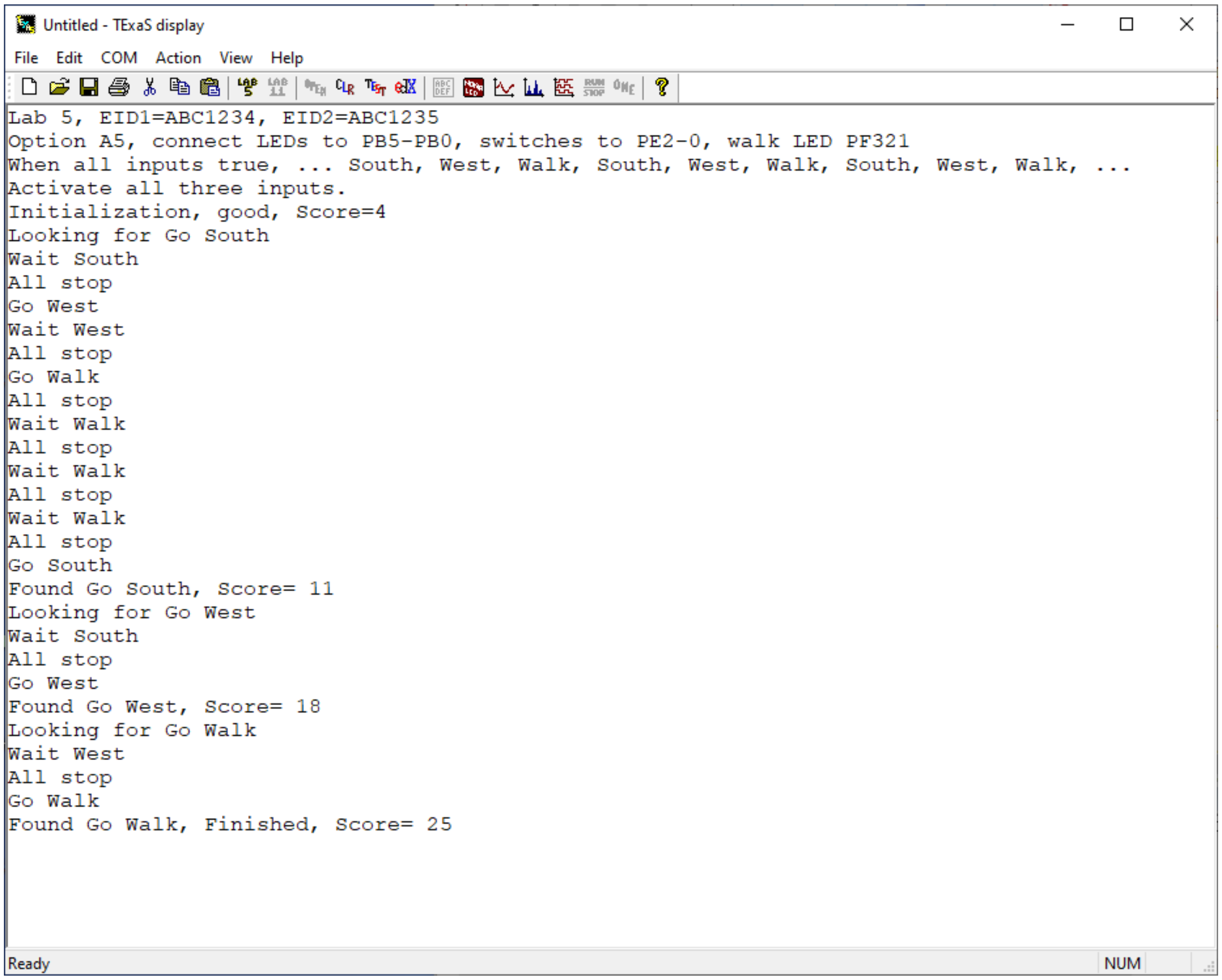
Part e - Construct and Test Circuit

After you have debugged your system in simulation mode, you will implement it on the real board. Use the same ports you used during simulation. **EE319K_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/1nlcl44ikmhpNv0spW9V664TYjdz_SAEiIu6bD7UkXUA/edit#
The first step is to interface three push button switches for the sensors. You should implement positive logic switches. *Do not place or remove wires on the protoboard while the power is on.* Build the switch circuits and test the voltages using a voltmeter. You can also use the debugger to observe the input pin to verify the proper operation of the interface.

The next step is to build six LED output circuits. Build the system physically in a shape that matches a traffic intersection, so the TA can better visualize the operation of the system. You will NOT connect any signals to +5V (labeled VBUS on the LaunchPad) this semester. The six LED circuits should be similar to the one LED connected in Lab 3. Each LED will draw about 3mA when active.

Part f - Debug on Real Hardware

Debug your combined hardware/software system on the actual TM4C123 board.



```
Lab 5, EID1=ABC1234, EID2=ABC1235
Option A5, connect LEDs to PB5-PB0, switches to PE2-0, walk LED PF321
When all inputs true, ... South, West, Walk, South, West, Walk, South, West, Walk, ...
Activate all three inputs.
Initialization, good, Score=4
Looking for Go South
Wait South
All stop
Go West
Wait West
All stop
Go Walk
All stop
Wait Walk
All stop
Wait Walk
All stop
Wait Walk
All stop
Go South
Found Go South, Score= 11
Looking for Go West
Wait South
All stop
Go West
Found Go West, Score= 18
Looking for Go Walk
Wait West
All stop
Go Walk
Found Go Walk, Finished, Score= 25
```

Figure 5.8. UART windows with full score of 25 on real board. Your output will include Spring 2023.

~~Part g - Debugging dump (skip this Spring 2023)~~

~~You will add a debugging dump that proves your FSM works. There are two 32-element buffers called **DumpBuf1** and **DumpBuf2**. The buffers are defined in **Debug.h** and imported into **Debug.s**. Each time after you output, but before you wait, you will call **Debug_Dump** and pass in input and output values. For Option A1, your inputs are on PA5-PA3, traffic lights are on PB5-0, and walk lights are on PF3-PF1, you call~~
~~**Debug_Dump(GPIO_PORTA_DATA_R>>3, (GPIO_PORTB_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option A2, your inputs are on PA4-PA2, traffic lights are on PB5-0, and walk lights are on PF3-PF1, you call~~
~~**Debug_Dump(GPIO_PORTA_DATA_R>>2, (GPIO_PORTB_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option A5, your inputs are on PE2-PE0, traffic lights are on PB5-0, and walk lights are on PF3-PF1, you call~~
~~**Debug_Dump(GPIO_PORTE_DATA_R, (GPIO_PORTB_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option A6, your inputs are on PE3-PE1, traffic lights are on PB5-0, and walk lights are on PF3-PF1, you call~~
~~**Debug_Dump(GPIO_PORTE_DATA_R>>1, (GPIO_PORTB_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

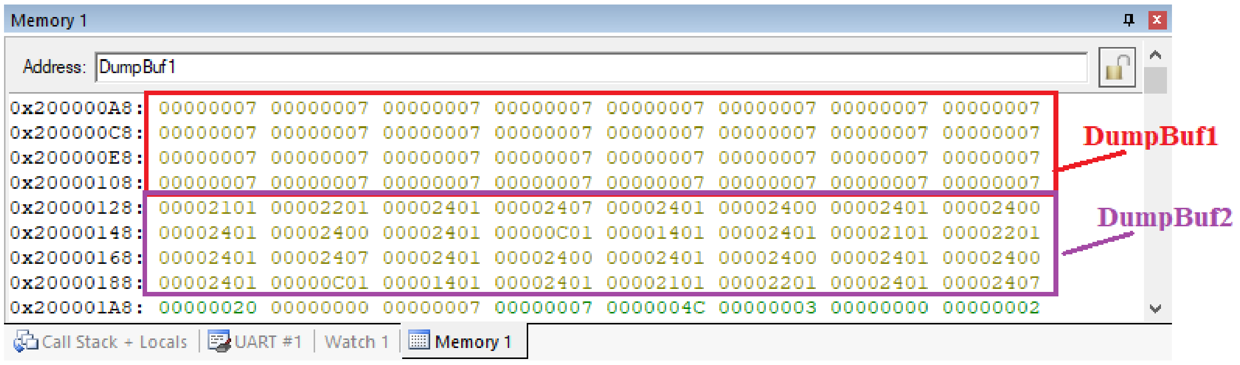
~~For Option B1, your inputs are on PA5-3, traffic lights are on on PE5-0, and walk lights are on PF3-PF1, you call~~
~~**Debug_Dump(GPIO_PORTA_DATA_R>>3, (GPIO_PORTE_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option B2, your inputs are on PA4-2, traffic lights are on on PE5-0, and walk lights are on PF3-PF1, you call~~ ~~**Debug_Dump(GPIO_PORTA_DATA_R>>2, (GPIO_PORTE_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option B3, your inputs are on PC6-4, traffic lights are on on PE5-0, and walk lights are on PF3-PF1, you call~~ ~~**Debug_Dump(GPIO_PORTC_DATA_R>>4, (GPIO_PORTE_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~For Option B4, your inputs are on PC7-5, traffic lights are on on PE5-0, and walk lights are on PF3-PF1, you call~~ ~~**Debug_Dump(GPIO_PORTC_DATA_R>>5, (GPIO_PORTE_DATA_R<<8) | ((GPIO_PORTF_DATA_R&0x0E)>>1));**~~

~~To collect the debugging dump, reset the software, activate all three input switches, and then run. Figure 5.9 shows a typical dump showing the first 32 states. The memory 1 window is set for unsigned hex long. The red rectangle shows the inputs are all 7, the purple rectangle shows the proper output sequence 0x2101, 0x2201, 0x2401, 0x2407,....~~



```
0x200000A8: 00000007 00000007 00000007 00000007 00000007 00000007 00000007 00000007
0x200000C8: 00000007 00000007 00000007 00000007 00000007 00000007 00000007 00000007
0x200000E8: 00000007 00000007 00000007 00000007 00000007 00000007 00000007 00000007
0x20000108: 00000007 00000007 00000007 00000007 00000007 00000007 00000007 00000007
0x20000128: 00002101 00002201 00002401 00002407 00002401 00002400 00002401 00002400
0x20000148: 00002401 00002400 00002401 00000C01 00001401 00002401 00002101 00002201
0x20000168: 00002401 00002407 00002401 00002400 00002401 00002400 00002401 00002400
0x20000188: 00002401 00000C01 00001401 00002401 00002101 00002201 00002401 00002407
0x200001A8: 00000020 00000000 00000007 00000007 0000004C 00000003 00000000 00000002
```

~~*Figure 5.9. Typical debugging dump.*~~

~~*Note: Debug.h and Debug.s are located in the inc folder so you can use this debugging dump in other labs.*~~

Demonstration

There are grading sheets for every lab so you know exactly how you will be evaluated. During checkout, you will be asked to show both the simulated and actual TM4C123 systems to the TA. The TAs will expect you to know how everything about your Lab 5 works.

Civil Engineering Questions

There are many civil engineering questions that students ask. How you choose to answer these questions will determine how good a civil engineer you are, but will not affect your grade on this lab. For each question, there are many possible answers, and you are free to choose how you want to answer it. It is reasonable however for the TA to ask how you would have implemented other answers to these civil engineering questions using the same FSM structure.

1. Do I have to implement a short all red condition when switching between South/West/Walk? *Answer*: yes this is a requirement. We wish to reduce accidents at this intersection.
2. How long should I wait in each state? *Answer*: 0.5 to 2 seconds of real TA time.
3. What happens if I push 2 or 3 buttons at a time? *Answer*: cycle through the requests servicing them in a round robin fashion (service one, then another). The required sequence for all three buttons is revealed in your UART window when you run the code.
4. What if I push the pedestrian sensor, but release it before the walk light activates? *Possible answer*: service it. *Possible answer*: ignore it. The grader does not check for this case.
5. What if I push a car sensor, but release it before it is serviced? *Possible answer*: ignore the request as if it never happened (e.g., car came to a red light, came to a full stop, and then made a legal turn). *Possible answer*: service the request or ignore it depending on when it occurred. The grader does not check for this case.

6. What if I push just one of the three sensors and not the other two? *Answer*: Service this one request indefinitely.
7. Assume there are no cars and the light is green on the North, what if a car now comes on the East? Do I have to recognize a new input right away or wait until the end of the wait time? *Possible answer:* no, just wait until the end of the current wait, then service it. *Possible answer*: yes; break states with long waits into multiple states with same output but shorter waits. The grader does not check for this case.
8. What if the pedestrian sensor is pushed while the don't walk light is flashing? *Possible answer*: ignore it, go to a green light state and if the pedestrian sensor is still pushed, then go to walk state again. *Possible answer:* if no cars are waiting, go back to the walk state. *Possible answer*: remember that the button was pushed, and go to a walk state after the next green light state.
9. Does the walk occur on just one street or both? *Answer*: stop all cars and let people walk across either or both streets.
10. How do I signify a walk condition? *Answer*: You must use the on board white LED for walk, and on board red LED as the don't walk.
11. Does the walk light need to flash? *Answer*: Yes, a typical light pattern for walk is white, red, off, red, off, red, off, red. This pattern requires 8 states in the FSM.
12. How many times does the don't walk (red) LED need to flash on and off? *Answer*: At least twice. This sequence will pass the grader: white, red, off, red, off, red. This pattern requires 6 states in the FSM.

Figure 5.10. Massachusetts walk signified by red and yellow (phased out in 2011).

In real products that we market to consumers, we put the executable instructions and the finite state machine linked data structure into the nonvolatile memory such as Flash. A good implementation will allow minor changes to the finite machine (adding states, modifying times, removing states, moving transition arrows, changing the initial state) simply by changing the linked data structure, without changing the executable instructions. Making changes to executable code requires you to debug/verify the system again. If there is a 1-1 mapping from FSM to linked data structure, then if we just change the state graph and follow the 1-1 mapping, we can be confident our new system operate the new FSM exactly as drawn in the STG. Obviously, if we add another input sensor or output light, it may be necessary to update the executable part of the software, re-assemble or re-compile and retest the system.

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 so it can be shared**
2. **Have Keil Lab 5 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

Combine your delay.s ~~SysTick.c, your debug.s~~, and your main C file into one text file called Lab5.c, and upload this Lab5.c 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 electronic (you do not need to show Port F)
3. Logic analyzer screenshot while in simulation mode, when all inputs are active, Figure 5.7.
4. Drawing of your state transition graph, by hand or done on computer.
5. ~~Debugging dump, like Figure 5.9 (skip this Spring 2023)~~

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

Grading Requirements

This lab was written in a manner intended to give you a great deal of flexibility in how you draw the FSM graph, while at the same time requiring very specific boundaries on how the FSM controller must be written. This flexibility causes students to question "when am I done?" or "is this enough for an A?" To clarify the distinction between computer engineering, and civil engineering, I re-state the computer engineering requirements. In particular do these 10 requirements well and you can get a good grade on this lab.

1. **Input Dependence:**
This means each state has 8 arrows such that the next state depends on the current state and the input. This means you can not solve the problem by simply cycling through all the states regardless of the input. You should not implement a Mealy machine.
2. **1-1 Mapping**:
There must be a 1-1 mapping between state graph and data structure. For a Moore machine, this means each state in the graph has an output, a time to wait, and 8 next state arrows (one for each input). The data structure has exactly these components: an output, a time to wait, and 8 next state pointers (one for each input). There is no more or no less information in the data structure than the information in the state graph. In other words what you have down on your state graph is exactly mapped to your data structures in the software. <Spring 2022, EE319H, the one combined with EE312H, must use pointers and not indices for the next states> The other sections can use pointers or indices.
3. **No Conditional Branches:**
There can be no conditional branches (do-while, while-loop, switch, if-then, or for-loops) in your system, other than in **`Delay`** and in **`Wait10ms`**. ~~**`SysTick_Wait`** and/or in **`SysTick_Wait10ms`**. See the Example 5.2.1 in the book.~~ You will have an unconditional while-loop in the main program that runs the FSM controller.
4. **Clear State Graph:**
The state graph defines exactly what the system does in a clear and unambiguous fashion.
5. **Consistent State Format:**
Each state has the same format as every other state. This means every state has exactly 8-bits of output, one time to wait, and 8 next pointers.
6. **Naming Convention:**
Please use good names (easy to understand and easy to change). Examples of bad state names are **S0** and **S1**.
7. **No Accidents:**
Do not allow cars to crash into each other or into pedestrians. On the South road, exactly one LED (red yellow or green) should be on at all times. On the West road, exactly one LED (red yellow or green) should

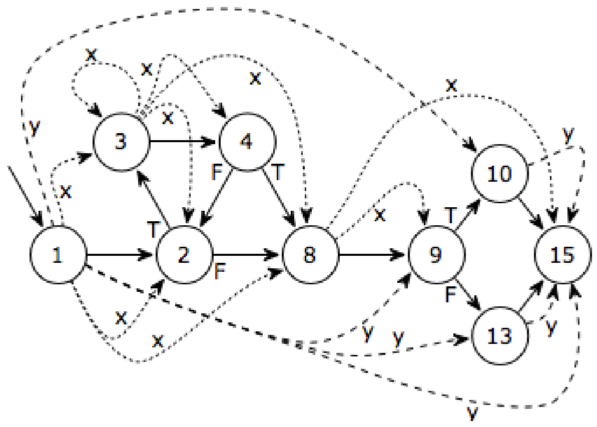
be on at all times. When the South road is green or yellow, the West and Walk lights should be red. When the West road is green or yellow, the South and Walk lights should be red. When the walk signal is white or or flashing red, the South and West lights should be red. Engineers do not want people to get hurt. There should be a short time of all red between yellow on one road and green on another. There should be a short time of all red before and after the walk sequence.

8. **State Number Requirement:**
   There should be approximately 10 to 30 states with a Moore finite state machine. Usually students with less than 10 states did not flash the don't walk light, or they flashed the lights using a counter. Counters and variables violate the "no conditional branch" requirement. If your machine has more than 30 states you have made it more complicated than we had in mind and you should consider simplifying your design.
9. **Pedestrian sensor**:
   If the pedestrian sensor is pushed, eventually a walk light must occur regardless if cars are present.
10. **No Starvation:**
   When all of the inputs are held active, your FSM should cycle through allowing southbound cars to go, westbound cars to go, and pedestrians to cross the street (in any order).

Tips and Tricks Section:

- When a car sensor is released or deactivated (set to logic 0), it means no cars are waiting to enter the intersection. I.e., When you are not pressing the button no cars are on the road.
- When a pedestrian sensor is released or deactivated (set to logic 0), it means no people are waiting to enter the intersection. I.e., When you are not pressing the button no people at the intersection.
- You should exercise common sense when assigning the length of time that the traffic light will spend in each state; so that the system changes at a speed convenient for the TA (stuff changes fast enough so the TA doesn't get bored, but not so fast that the TA can't see what is happening).
- There is no single, "best" way to implement your system. However, your scheme must use a linked data structure stored in ROM. There should be a 1-1 mapping from the FSM states and the linked elements. An example solution will have about 10 to 30 states in the finite state machine, and provides for input dependence.
- Try not to focus on the civil engineering issues. I.e., the machine does not have to maximize traffic flow or minimize waiting. On the other hand if there are multiple requests, the system should cycle through, servicing them all. Build a quality computer engineering solution that is easy to understand and easy to change.

Aside: If your FSM starts to look like the image below, you may want to go to a TA for advice on cleaning it up. We mention this to reinforce point 4 of the grading requirements which is to have a ***Clear State Graph.***



Drawing your FSM

- Because we have three inputs, there will be 8 next state arrows. One way to draw the FSM graph to make it easier to read is to use X to signify don't care. For example, compare the Figure 5.2.4 in the book to the FSM graph in Figure 5.2 below. Drawing two arrows labeled **01** and **11** is the same as drawing one arrow with the label **X1**. When we implement the data structure, we will expand the shorthand and explicitly list all possible next states.

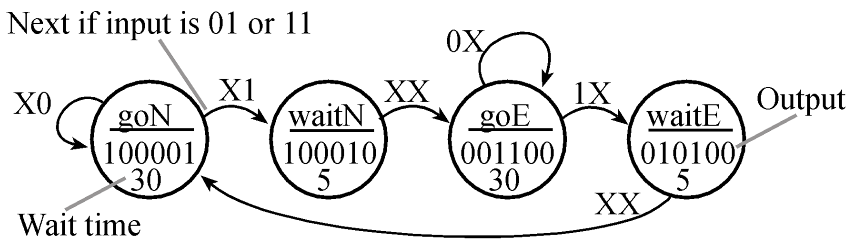
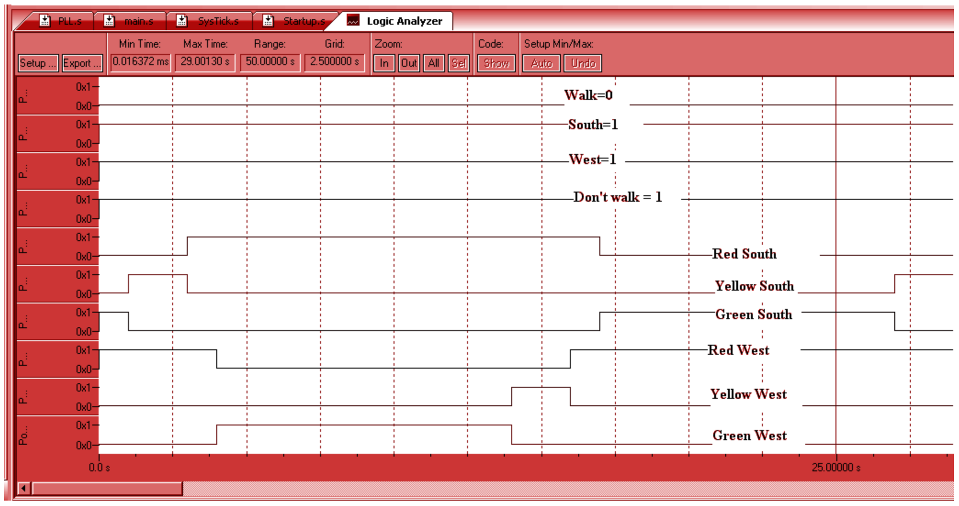


Figure 5.11. FSM from the book Figure 5.2.4 redrawn with a shorthand format.

Example Logic Analyzer Window.

Your version will not be red, and also your I/O window may look slightly different from these examples (e.g., you are free to assign signals to different port bits.)



Min Time: 0.016372 ms
Max Time: 29.00130 s
Range: 50.00000 s
Grid: 2.500000 s

Simulation showing cars on both South and West

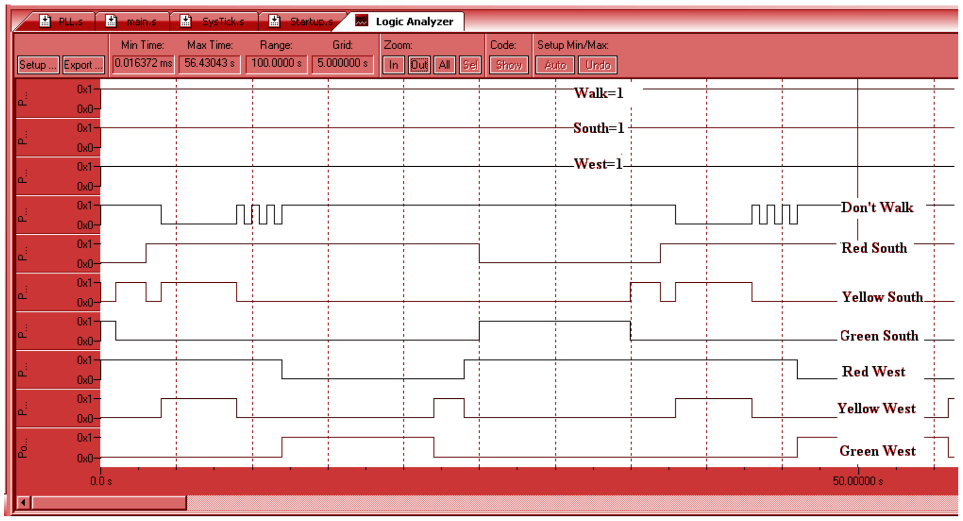


Figure 5.12. Simulation showing walk pushed, and cars on both South and West

FAQ: Frequently Asked Questions

1. **Is the walk signal for a certain direction?**
   No, the walk signal will be pushed when a pedestrian is present in the intersection.

2. **We are getting an error when we compile, saying that the file TM4C123GH6PM.h cannot be opened because it cannot be found. Where can we find this file?**
   If you still have your EE319kware, you could copy paste it into your folder that includes your lab 5. It should be in the inc folder.
   C:\Keil\EE319Kware\inc\TM4C123GH6PM.h
   You could also try to update your whole git folder? They might have dropped in the file for it already.

3. **Should we use the 7406/ULN2003B IC for our leds in this lab? Or is that not necessary?**
   <Fall 2022, no driver is needed. The LEDs require only 3mA> <Spring 2022, You should use the ULN2003B. There are 7 pairs of inputs and outputs (check the datasheet for which is which), which is exactly how many you need. > Two of your outputs (the walk and the don't walk lights) are on PortF, which is the onboard 3-color LED.

4. **Is #define GPIO_PORTF_OUT (*((volatile uint32_t *)0x40025038)) the correct code to set bits 1, 2, and 3 of Port F so that I can change the lights on them? I got this number by adding 0x20 0x10 and 0x08 to 0x4002.5000 which is the start memory map in peripherals in one of the data sheets.**
   Yes, that address will allow you to read/write to PF1, PF2, and PF3 without affecting the other pins on Port F

5. **What is "error: L6002U: Could not open file .\texas.o"?**
   There is something wrong with your EE319K installation.

6. **When I try to build my files I get this error.**
   linking...
   .\Lab5.axf: error: L6002U: Could not open file --ro-base: No such file or directory

".\Lab5.axf" - 1 Errors, 0 Warning(s).
Target not created

Someone asked how to fix this in the lab 3 FAQ where some TAs offered an interim solution in reference to assembly files. Is there a formal solution yet?

Some automatic file backup systems like GoogleDrive OneDrive Box or DropBox open files to back them up. Try quitting keil and restart. Try debugging in a directory not automatically backed up. (Of course remember to back up manually so your work is not lost if the computer crashes)

7. **What is a possible source of error to the warning "_____ macro redefined"? This occurs whenever I try to do a**

```
#define GPIO_PORTE_DIR_R    (*((volatile unsigned long *)0x40024400))
```

This macro is defined in the "tm4c123gh6pm.h" file. There is no need to redefine this macro since you should be including "tm4c123gh6pm.h" in your project.

8. **How do we do a NOP properly in C? We keep getting an error saying that Port A doesn't have a clock set.**

Make sure none of your code uses the old clock registers like SYSCTL_RCGC2_R. Rather use SYSCTL_RCGCGPIO_R to set the clock.

9. **I don’t know why there should be 8 next pointers? I don't see how there could be 8 other states.**

Since you have three different inputs buttons, you have 8 different combinations of inputs that you need to account for. This is a maximum however, there are many states where you will progress to another state regardless of what the other switches are (like when you are blinking the walk LED to signify it is almost done. Even if another button is pressed, you should finish blinking before handling that request). Your next state pointer array will have many duplicates typically, but you will need to have 8 to account for each input combination in every state.

10. **How would you show the microboard's LED lights on Eagle? Do you have to show it at all?**

The circuit is already embedded in the board, so I would only worry about showing pinouts and external hardware interfaces for the leds, and switches. You do not need to show Port F

11. **I know we can't use conditional branches, but switches were not listed as something we weren't allowed to use in the lab manual. Will points be deducted if we implement a switch?**

The reason we prohibit branches is because the logic of which next state to go to for each state should be specified in your FSM data, not in your engine. The only thing your engine should do is find the next state in the data, using the current state as an index or pointer (whichever you used in your design). This way, you should have no need to use switches or conditional branches.

Consider a software system with only 20 conditional branch statements. This system might have up to 1,000,000 different execution paths, all of which must be tested if the system is to be deployed in a safety critical application.

12. **Do we need pull down resistors for the LEDs on the board? PF3 and PF1?**

You only need to use the PDR and PUR registers when configuring inputs. For outputs, you don't use pull resistors. You do set PUR=1 for switches on PF4 and PF0

13. **When we run the program in our simulation, the appropriate lights turn on, but then do not turn off when they are supposed to. Such as the walk warning light does not blink. Is anyone else having this problem?**
    This sounds like a problem with your state machine. Double check your outputs in your state machine making sure you set pins equal to zero.

14. **I am not able to have lasting color changes on their logic analyzers?**
    Select a color to change the pin to, click ok to close the palette, then click "close" on the setup analyzer window. Repeat for all pins. It's tedious that you have to do it for each pin individually, but it worked for me.

15. **Why are we disabling and then enabling interrupts?**
    Typically global interrupts rather than individual functions are disabled during "critical sections". Consider what would happen if you were initializing a timer or some other function when another interrupt fired? Or if you were reading or writing to a piece of data when an interrupt fired that modified the data at the same address? When you start using interrupts more in your labs in the upcoming weeks it'll become more of something to think about.