Lab 5. Traffic Light Controller

Outline:

Preparation

Book:

Chapter 5.

Data Sheets:

http://users.ece.utexas.edu/~valvano/Datasheets/LED_red.pdf
http://users.ece.utexas.edu/~valvano/Datasheets/LED_yellow.pdf
http://users.ece.utexas.edu/~valvano/Datasheets/LED_green.pdf

Starter Code:

Download, Unzip, Build and Simulate the FSM example from the link in your local directory with the other projects. ⇛ C10_TableTrafficLight.zip
PointerTrafficLight_4C123
edXLab10.dll
Lab5_EE319K (get from Git)

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 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/grader (edXLab10.dll). The call to TExaS_Init will activate 80 MHz PLL.

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.

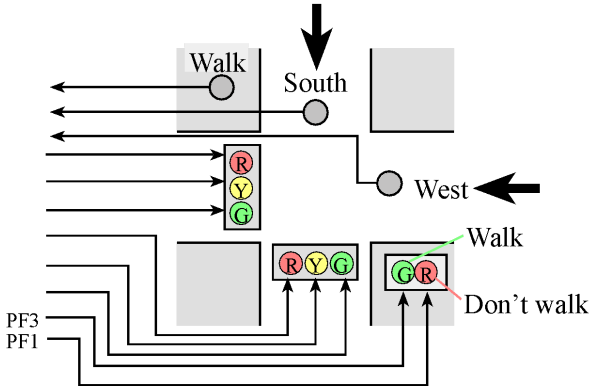


Figure 5.1. Traffic Light Intersection (3 inputs and 8 outputs).

- **INPUTS - 3**
  1. ***South Sensor:*** This sensor or button be activated(set to logic 1), 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 pressing it down.
  2. ***West Sensor:*** Same as south sensor but for westbound traffic.
  3. ***Walk Button:*** Will be pushed by a pedestrian when he or she wishes to cross in any direction. You will simulate this effect by pushing a button.
- **OUTPUTS - 8**

  You will use 8 outputs from your microcomputer that control the traffic lights.

  1. ***(6 LEDS) South & 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.

  2. ***(1 LED) Walk light:*** This will be the green LED (PF3) on the LaunchPad, and will be turned on when pedestrians are allowed to cross. To request a walk, a pedestrian must push and hold the walk button for at least 2 seconds, after which person can release the button, and the walk request should be serviced eventually. If the user does not hold down the button for 2 seconds this document does not specify what will happen, you may either go to the walking states or not. We leave this freedom up to the engineers designing the FSM.

     The walk sequence should be realistic, showing three separate conditions:
     1. ***Walk:*** Your walk light should be on signifying the pedestrians may cross.

2. ***Warning***: Flash your don't walk LED signifying that pedestrians need to hurry up. You may decide how long you want to flash it.
3. ***Don't Walk:*** Your don't walk LED should be on and constant.

3. ***(1 LED) Don't Walk light:*** This will be the red LED (PF1) on the LaunchPad.
   a. When the "don't walk" condition flashes (and the two traffic signals are red), pedestrians should hurry up and finish crossing in any direction.
   b. When the "don't walk" condition is on steady, pedestrians should not enter the intersection.

   All other inputs and outputs must be built on the protoboard. The Lab 5 simulator/grader is implemented in edXLab10.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 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

Decide which port pins you will use for the inputs and outputs. Avoid the pins with hardware already connected. Table 4.4 in the book lists the pins to avoid. Run the starter code in the simulator to see which ports are available for the lights and switches; these choices are listed in Tables 5.1 and 5.2. The "don't walk" and "walk" lights must be PF1 and PF3 respectively, but you have some flexibility when deciding where to attach the remaining signals. In particular, Table 5.1 shows you three possibilities for how you can connect the six LEDs that form the traffic lights. Table 5.2 shows you three possibilities for how you can connect the three positive logic switches that constitute the input sensors. Obviously, you will not connect both inputs and outputs to the same pin. Please note that the possibilities listed in Tables 5.1 and 5.2 are not the only possibilities.

Stoplight Signal	Possibility 1	Possibility 2	Possibility 3
Red south	PA7	PB5	PE5
Yellow south	PA6	PB4	PE4
Green south	PA5	PB3	PE3
Red west	PA4	PB2	PE2
Yellow west	PA3	PB1	PE1
Green west	PA2	PB0	PE0

Table 5.1. Possible ports to interface the traffic lights (PF1=red don't walk, PF3=green walk).

Stoplight Sensor	Possibility 1	Possibility 2	Possibility 3
Walk sensor	PA4	PB2	PE2
South sensor	PA3	PB1	PE1
West sensor	PA2	PB0	PE0

Table 5.2. Possible ports to interface the sensors.

If you are using PD0, PD1, PB7, PB6, PB1 or PB0, make sure R9, R10, R25, and R29 are removed from your LaunchPad. R25 and R29 are not on the older LM4F120 LaunchPads, just the new TM4C123 LaunchPads. The TM4C123 LaunchPads I bought did not have R25 and R29 soldered on, so I just had to remove R9 and R10. The R9 R10 jumpers are only needed for some MSP430 booster packs running on the TM4C123, so there is little chance you will ever need R9 and R10.

Part b - FSM Design

Design a finite state machine that implements a traffic light system. Include a picture of your finite state machine (state transition graph) in the deliverables showing the various states, inputs, outputs, wait times and transitions. It is advisable that you come to the TAs to verify you 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 c - Debug C Code In Simulation

Write and debug the C code that implements the traffic light control system. In simulation mode, capture logic analyzer screen shots showing the operation of your traffic light when cars are present on both roads, like Figure 5.3. *Your performance grade on this lab will be determined by your TA during checkout. You may run the automatic grader this semester, but this score will not be considered as part of your actual grade.*

Part d - 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_TM4C123_Artist.sch** is a starter file you should use to draw your hardware circuit diagram using the program PCB Artist. 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. Look up the pin assignments in the 7406 data sheet. Be sure to connect +5V (labeled VBUS on the Launchpad) power to pin 14 and ground to pin 7. Write a simple main program to test the LED interface. Set the outputs high and low, and measure the following three voltages: ground to the input to 7406, ground to the output from 7406 which is the LED cathode voltage, and ground to the LED anode voltage.

Part e - Debug on Real Hardware

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

Demonstration

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 the **`SysTick_Wait`** function works, and know how to add more input signals and/or output signals. An interesting question that may be asked during checkout is how you could experimentally prove your system works. In other words, what data should be collected and how would you collect it? If there were an accident, could you theoretically prove to the judge and jury that your software implements the FSM? What type of FSM do you have? What other types are there? How many states does it have? In general, how many next-state arrows are there? Explain how the linked data structure is used to implement the FSM. Explain the mathematical equation used to calculate the address of the next state, depending on the current state and the input. Be prepared to

write software that delays 1 second without using the timer (you can use a calculator and manual). How do you prove the delay will be 1 second? What does it mean for the C compiler to align objects in memory? Why does the compiler perform alignment? List some general qualities that would characterize a good FSM.

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. How long should I wait in each state? *Possible answer*: 1 to 2 seconds of real TA time.
2. What happens if I push 2 or 3 buttons at a time? *Possible answer*: cycle through the requests servicing them in a round robin fashion (service one, then another)
3. What if I push the walk button, but release it before 2 seconds are up? *Possible answer*: service it or ignore it depending on exactly when it occurred.
4. What if I push a car button, 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.
5. 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.
6. What if the walk button is pushed while the don't walk light is flashing? *Possible answer*: ignore it, go to a green light state and if the walk button 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.
7. Does the walk occur on just one street or both? *Possible answer*: stop all cars and let people walk across either or both streets.
8. How do I signify a walk condition? *Answer*: You must use the on board green LED for walk, and on board red LED as the don't walk.

Figure 5.3. 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 properly. 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.

Deliverables

(Items 2, 3, and 4 are one pdf file uploaded to Git, have this file open during demo.)

1. Lab 5 grading sheet. You can print it yourself or pick up a copy in lab. You fill out the information at the top.
2. Logic analyzer screenshot while in simulation mode, when cars are present on both roads.
3. Drawing of the finite state machine, by hand or done on computer.
4. All source files that you have changed or added should be committed to Git.
5. 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 require very specific boundaries on how the FSM controller must be written. This flexibility causes students to be 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 9 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 then 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 code.
3. **No Conditional Branches:**
   There can be no conditional branches (do-while, while-loop, if-then, or for-loops) in your system, other than in `SysTick_Wait` and 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. Do not allow pedestrians to walk in one direction while any cars are allowed to go. Engineers do not want people to get hurt. I.e., there should not be only a green or only a yellow LED on one road at the same time there is only a green or only a yellow LED on the other road.
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 changing your design.

9. **2 Second Walk Button Timing**:
   If the pedestrian pushes the walk button for 2 or more seconds, eventually a walk light must occur. If the pedestrian pushed the walk button for less than 2 seconds, it is up to you to decide what happens. However, if a light has just turned green and there continues to be traffic in that direction, then that light should stay green for at least 2 more seconds after the pedestrian button has been released and then allow the pedestrian to cross.
10. **No Starvation:**
    When all of the inputs are held down, your FSM should cycle through allowing southbound cars to go, westbound cars to go, and pedestrians to cross the street.

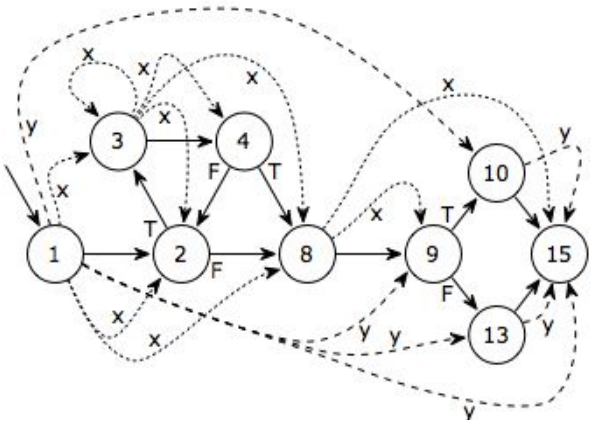
Tips and Tricks Section:

- 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.
- 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.

Handling the 2 Second Walk

- One way to handle the 2-second walk button requirement is to have a simplified state graph centered around a center home or check all state. The idea is that you have one state that you can always come back to that controls what your state graph is doing. Initially when thinking about this problem you can identify that a servicing traffic on the east or west road or servicing the pedestrians can be thought of a set sequence of events that have to start and end somewhere. So just make them start and end at the check all state. This approach may make sense to some of you or there is another approach mentioned next.
- Another way to handle the 2-second walk button requirement is to add a duplicate set of states. The first set of states means the walk button is not pressed. The second set of states means the walk operation is requested. Go from the first set to the second set whenever the walk is pushed. Go from the second back to the first whenever the walk condition is output. The two sets of states allow you to remember that a walk has been requested; in this way the request is serviced when appropriate.

  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.

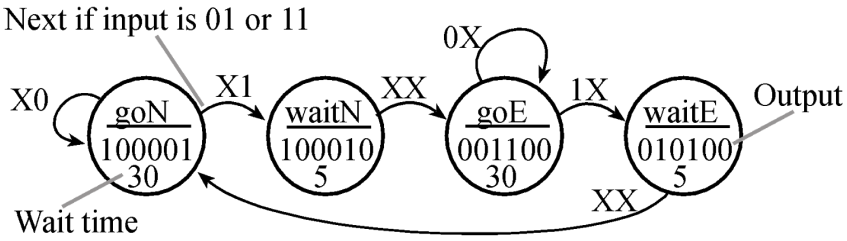
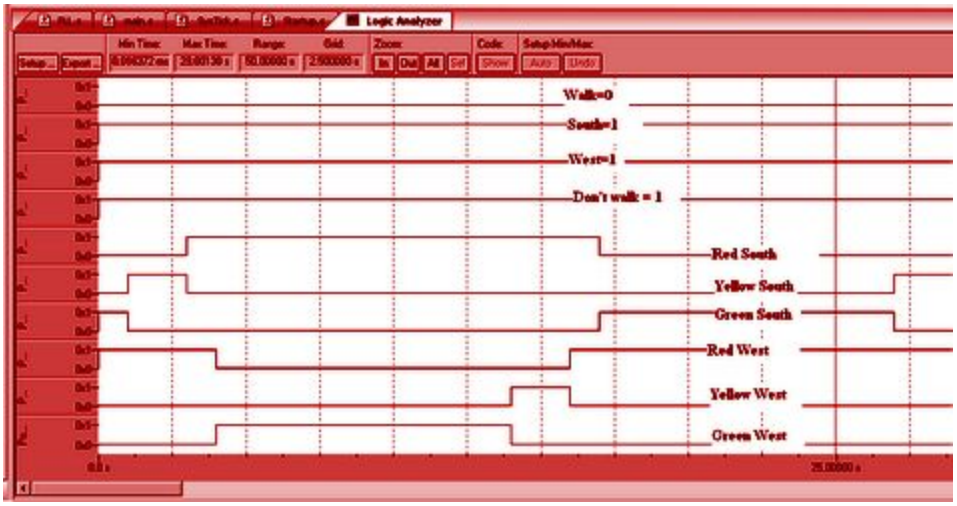


Figure 5.2. FSM from the book Figure 5.2.4 redrawn with a shorthand format.

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.)



Simulation showing cars on both South and West

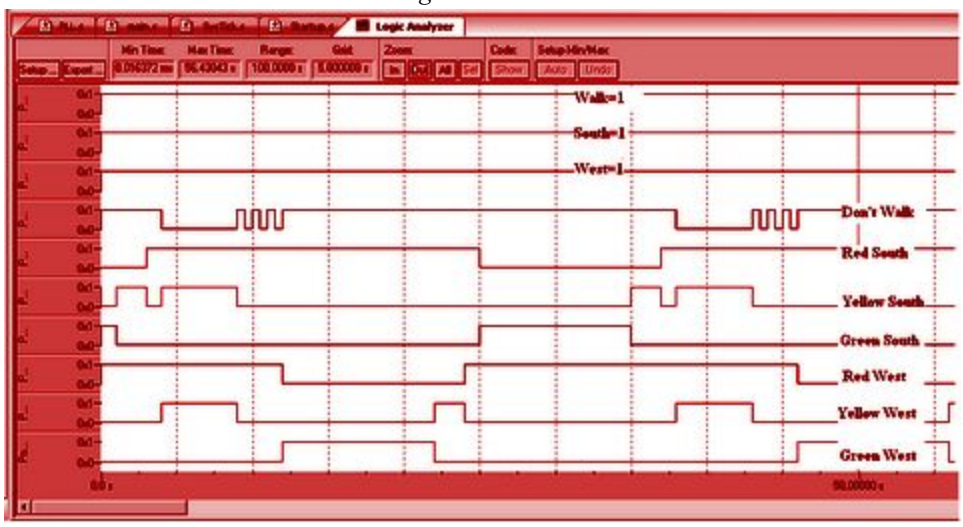


Figure 5.5. Simulation showing walk pushed, and cars on both South and West

FAQ: Frequently Asked Questions

1. **Is the walk signal for a certain direction?**
   No, the walk signal will be pushed by a pedestrian when he or she wishes to cross in **any** direction.

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 lists includes your lab 5. It should be in the inc folder.
   C:\Keil\EE319KwareSpring2016\inc\TM4C123GH6PM.h
   You could also try to update your whole svn folder? They might have dropped in the file for it already.

3. **Should we use the 7406 IC used in Lab 3 for our leds in this lab too? Or is that not necessary?**
   You should use the 7406. There are 6 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 LED.

4. **The lab manual mentions "SysTick_Wait" and "SysTick_Wait10ms" subroutines. Are we supposed to program these, or are they included somewhere?**
   These files are found in the SysTick_4c123 folder of the 319kware. You can copy and paste the SysTick.c and .h files into your working directory, then add them to your project by right clicking the source folder in the Keil project explorer on the left side and selecting "Add existing files...". Make sure to include the #include "SysTick.h" in your main.c to get the function prototypes for SysTick_Init and SysTick_Wait. SysTick.c will be in C, so no worries of assembly there. You can call assembly functions from C, but it would be easier to just stick with the C file in this case.

5. **Is #define GPIO_PORTF_OUT (*((volatile uint32_t *)0x40025028)) the correct code to set bits 1 and 3 of Port F so that I can change the lights on them? I got this number by adding x20 and x08 to x4002.5000 which is the start memory map in peripherals in one of the data sheets.**
   It looks like that address will allow you to read/write to PF1 and PF3 without affecting the other pins on Port F

6. **What is "error: L6002U: Could not open file .\texas.o"?**
   Please SVN update your Lab5 folder again and see if the texas.o file shows up in your folder.

7. **Are there limitations to SysTick_Wait that we should know about? It seems that when we call this subroutine to wait for 2 seconds, it gets stuck in an infinite loop**
   Make sure to call SysTick_Init

8. **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?
   Try uninstalling and reinstalling Keil, reinstall EE319Kware, and make sure your launchpad drivers are up to date. Follow the instructions on the EE319K website for doing all of those, I had the same error it worked for me

9. **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 this file in your "TableTrafficLight.c" file.

10. **How do we do a NOP properly in C? We keep getting an error saying that Port A doesn't have a clock set.**
    Instead of using SYSCTL_RCGCGPIO_R to set clock, use the following instead:
    ```
    SYSCTL_RCGC2_R |= 0x32;     // 1) enable clock to F E B
    delay = SYSCTL_RCGC2_R;     // 2) no need to unlock
    ```
    The second line is just wasting a few clock cycles before modifying the port registers to allow the clock to settle.

11. **I copy and pasted the Systick.h Systick.c and systick.crf into my Lab 5 SVN folder but whenever I try to call Systick.init(); the compiler says it's an undefined symbol.**
    You will have to do a couple more steps for that to work:
    1) adding **#include "SysTick.h"** in your TrafficLight.c
    2) right click on the "source" folder on the left, click on "Add Existing File to Group 'Source'", and add "SysTick.c". After this step you should see SysTick.c also listed under the source folder.
    3) since the directory for "tm4c123gh6pm.h" for Lab 5 is different from the 319kware, change**#include "inc/tm4c123gh6pm.h"** to **#include "tm4c123gh6pm.h"** in SysTick.c

12. **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.

13. **We are using the 7406 driver to implement our RYG LEDs as outputs, as was suggested. Do we need to use capacitors with the LEDs like we did for Lab 3, or is just using the 220 ohm resistors and two corresponding pins acceptable? I assumed we wouldn't need them at all because only one capacitor is given in the PCB artist file, but I just wanted to clarify.**
    You just need to keep a single capacitor across the 5V line. You will essentially take your circuit from Lab3, and copy it 5 more times on different input/output pins.

14. **How would you show the microboard's LED lights on PCBartist? 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 7406 driver, capacitor, leds, switches, etc.

15. **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.

16. **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 the onboard switches. For the LEDs, you don't need to worry about them.

17. **What's a good way of reading if the pedestrian button has been pressed for 2 seconds or more? I'm confused on this. Can we just continue to check the input value over and over again until the cycles * 12.5ns equals 2 seconds and then check the final input value? If at any moment within those two seconds it hasn't been pressed we can assume they let it go?**
    Try to think of time in terms of states instead of clock cycles for this lab. The state machine reads inputs at the end of state before it transitions. So one idea is to make each state last 1 or 2 seconds so each button being pressed gets read.

18. **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. Also, if you are using Port D for your lights, the signals will not be processed correctly since Port D was designed for special cases/uses.

19. **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.

20. **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.