CECS 346 Project 1 -- Traffic Light Controller

By Dr. Min He

Preparation

You will need a LaunchPad, 2 sets of red/green/yellow LEDs, 3 switches (you can use one dip-switch with a minimum of 3 switches), and resistors if needed.

Book Reading	Sections 6.4 to 6.9
Reference project	TExasWare/C10_TableTrafficLight
Starter project	Lab 3.

Purpose

This lab has these major objectives: 1) the understanding and implementing of indexed data structures; 2) learning how to create a segmented software system; and 3) the study of real-time synchronization by designing a finite state machine controller. Software skills you will learn include linked data structures, creating fixed-time delays using the SysTick timer, and debugging real-time systems. Please read the entire lab before starting.

System Requirements

Design a traffic light controller for the intersection of two equally busy one-way streets. The goal is to maximize traffic flow, minimize waiting time at a red light, and avoid accidents.

Consider a 4-corner intersection as shown in Figure 10.1. There are two one-way streets are labeled South (cars travel South) and West (cars travel West). There are three inputs to your LaunchPad, two are car sensors, and one is a pedestrian sensor. The *South* car sensor will be true (3.3V) if one or more cars are near the intersection on the South road. Similarly, the *West* car sensor will be true (3.3V) if one or more cars are near the intersection on the West road. The *Walk* sensor will be true (3.3V) if a pedestrian is present and he or she wishes to cross in any direction. This walk sensor is different from a walk button on most real intersections. This means when you are testing the system, you must push and hold the walk sensor until the FSM recognizes the presence of the pedestrian. Similarly, you will have to push and hold the car sensor until the FSM recognizes the presence of the car. In this simple system, if the walk sensor is +3.3V, there is pedestrian to service, and if the walk sensor is 0V, there are no people who wish to walk. In a similar fashion, when a car sensor is 0V, it means no cars are waiting to enter the intersection. You will interface 6 LEDs that represent the two Red-Yellow-Green traffic lights, and you will use the PF3 green LED for the "walk" light and the PF1 red LED for the "don't walk" light. The walk sequence should be showing three separate conditions: 1) "walk", 2) "hurry up" using a flashing red LED, and 3) "don't walk". When the "walk" condition is signified, pedestrians are allowed to cross. When the "don't walk" light flashes (and the two traffic signals are red), pedestrians should hurry up and finish crossing. When the "don't walk" condition is on steady, pedestrians should not enter the intersection.

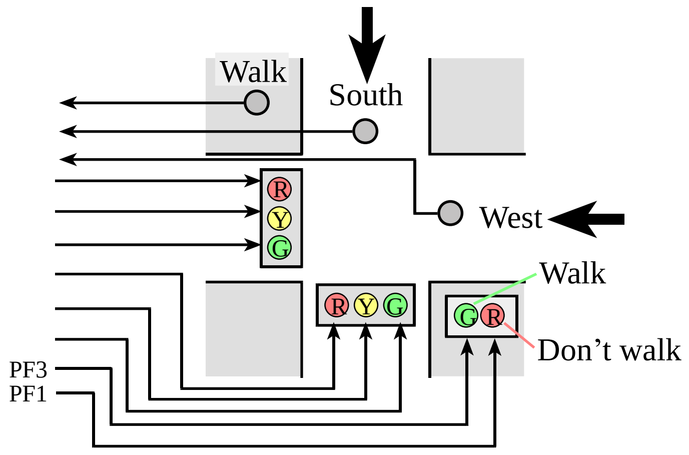

Figure 10.1. Traffic Light Intersection.

When none of the three sensors is true, stay in current state or finish transition to green. If one sensor is true, turn on the green for traffic light of that direction or "walk" for pedestrian and stay on as long as that sensor is true and no other sensor is true. If there are more than one sensor are true: cycle through the requests servicing them in a round robin fashion, i.e., take turns to go through green-yellow-red for traffic light, walk-hurry-don't walk for pedestrian in the following priority order: pedestrian-south-west.

The time duration for each light is: green/walk 6 seconds, yellow/hurry 2 seconds, red/don't walk 8 seconds. Cars should not be allowed to hit the pedestrians. The walk sequence should be realistic, showing three separate conditions: 1) "walk", 2) "hurry up" using a flashing LED that flashes every 0.5 second, and 3) "don't walk". You may assume the three sensors remain active for as long as service is required. The "hurry up" flashing should occur at least twice but at most 4 times.

Implementation requirements:

Implement a Moore machine. There should be a 1-1 mapping between FSM graph and data structure. For a Moore machine, this means each state in the graph has a name, an output, a time to wait, and 8 next state links (one for each input value). The data structure has exactly these components: a name, an output, a time to wait, and 8 next state pointers (one for each input value). There is no more or no less information in the data structure than the information in the state graph.

There can be no conditional branches in program, other than the **while** in **SysTick_Wait** and the **for** in **SysTick_Wait10ms**. This will simplify debugging make the FSM engine trivial.

The state graph defines exactly what the system does in a clear and unambiguous fashion. In other words, do not embed functionality (e.g., flash 3 times) into the software that is not explicitly defined in the state graph.

Each state has the same format. This means every state has exact one name, one 8-bit output (could be stored as one or two fields in the struct), one time to wait, and 8 next indices.

Please use good names and labels (easy to understand and easy to change). Examples of bad state names are **S0** and **S1**.

Procedure

If you are using PD0, PD1, PB7, PB6, PB1 or PB0, make sure R9, R10, R25, and R29 are removed from your LaunchPad as shown in Figure 10.3. R25 and R29 are not on the older LM4F120 LaunchPads, just the new TM4C123 LaunchPads. The TM4C123 LaunchPads I bought over summer 2013 did not have R25 and R29 soldered on, so I just had to remove R9 and R10.

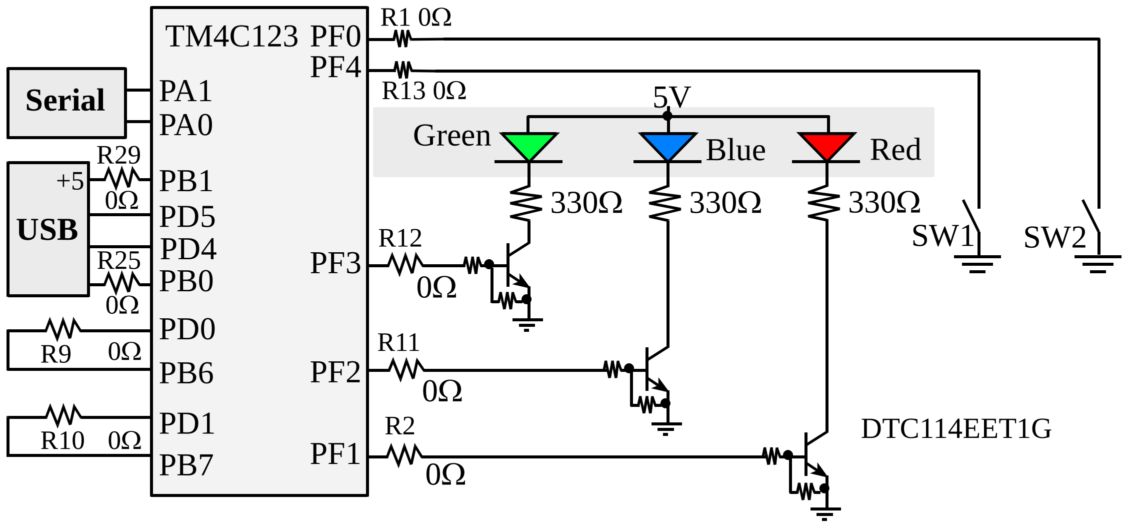

Figure 10.3. Switch and LED interfaces on the Stellaris® LaunchPad Evaluation Board. The zero ohm resistors can be removed so the corresponding pin can be used for its regular purpose.

Most microcontrollers have a rich set of timer functions. For this lab, you will use SysTick timer to wait a prescribed amount of time.

Part a) Decide which port pins you will use for the inputs and outputs. Avoid the pins with hardware already connected. The “don’t walk” and “walk” lights must be PF1 and PF3 respectively, but where to attach the others have some flexibility. Recommended connections are shown in the following two tables.

Red east/west	PB5
Yellow east/west	PB4
Green east/west	PB3
Red north/south	PB2
Yellow north/south	PB1
Green north/south	PB0

Walk sensor	PE2
North/south sensor	PE1
East/west sensor	PE0

Make the following changes to reference project for the connections shown above:

```
#define P_LIGHT (*((volatile unsigned long *)0x40025028))
#define SENSOR (*((volatile unsigned long *)0x4002401C))
```

Part b) Design a finite state machine that implements a good traffic light system. Draw a graphical picture of your finite state machine showing the various states, inputs, outputs, wait times and transitions.

Part c) Write and debug the C code that implements the traffic light control system. During the debugging phase with the simulator, use the logic analyzer to visualize the input/output behavior of your system.

Part d) 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. 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 and test the switch circuits.

The next step is to build the six LED output circuits. Build the system physically in a shape that matches a traffic intersection. You will use the PF3-2-1 LED interface for the walk light (green for walk and red for don't walk). Write a simple main program to test the LED interface, similar to the way you tested Lab 3.

Part e) Debug your combined hardware/software system on the actual LaunchPad.

During the real board grading you will have to push the sensors so that all cases are tested (just west, just south, just walk, two of the three, and all three).

Deliverable

1. Demonstrate your system on LaunchPad
2. Submit a video or a link to your video that records all the required behavior of your embedded system.
3. Submit a project report: follow the project report template for report format. The following information needs to be included in your report:
   a. State table for your Moore FSM
   b. State diagram for your Moore FSM
   c. Software source code
   d. Hardware schematic and picture of your embedded system.