

CECS 346 Fall 2021 Project 1

Traffic Light Controller

By

Dylan Dang

October 15, 2021

The purpose of this project was to implement a traffic light controller system with the implementation of a Moore Finite State Machine, SysTick Timer, and GPIO. With the press of a button, sensors will tell both cars and pedestrians when to cross the intersection with minimized waiting time and no accidents.

1

**Introduction**

The objective is to maximize traffic flow, minimize waiting time, and have no accidents within the intersection of two equally busy one-way streets. The two one-way streets are labeled South (cars going south) and West (cars going west) with Pedestrians being able to go in any direction.

The South sensor will be active if there are 1 or more cars on the South road intersection. The West sensor will be active if there are 1 or more cars on the West road intersection. The Walk button will be active if there is at least 1 pedestrian who wishes to cross the intersection. If more than 1 pedestrians press the Walk button before the traffic change, it is treated as one press.

There will be 6 LEDs that will be interfaced through the GPIO ports to represent the 2 Red-Yellow-Green traffic lights, and the PF3 green LED will represent “walk” and PF1 red LED will represent “hurry up” and “don’t walk”. When “walk” light is solid, pedestrians are allowed to cross. When “don’t walk” light is flashing, pedestrians should hurry up and finish crossing. When “don’t walk” light is solid, pedestrians should not enter the intersection.

The only valid transition for traffic lights is Green-Yellow-Red, and if the transition has started, it must finish before another participant can go. The only valid transition for pedestrians is Walk-Hurry-Don’t walk. When multiple participants want to go, each participant must have a chance to go. If a participant just had a green light and another participant needs to cross the intersection, then the first participant must pass the chance to the other participant since the first participant can’t go again unless no one else needs to go.

The time durations for traffic lights are Green/Walk = 2 seconds, Yellow/Hurry Up = 1 second, and Red/Don’t Walk = 3 seconds. “Hurry Up” uses a flashing LED that flashes on for 0.25 second and off for 0.25 second that repeats for a total of 1 second. The traffic light controller system will begin with GoS (Green on South).

**Operation**

*SENSORS* is used to control PE0, PE1, and PE2 which controls the traffic light sensors for Pedestrian, West, and South sensors respectively.

2

Pressing/Holding PE0 will transition the traffic light LEDs until the state GoP (Green on Pedestrian) is reached, where the Pedestrian light is green and both West and South lights are red.

Pressing/Holding PE1 will transition the traffic light LEDs until the state GoW (Green on West) is reached, where the West light is green and both Pedestrian and South lights are red.

Pressing/Holding PE2 will transition the traffic light LEDs until the state GoS (Green on South) is reached, where the South light is green and both West and Pedestrian lights are red.

Pressing/Holding more than 1 input will transition the traffic light LEDs back and forth as you would expect in a traffic intersection. There will always be a green light in the intersection.

**Link for Video Demonstration of the Traffic Light Controller:** <https://youtu.be/tPfT7VtUL1U>

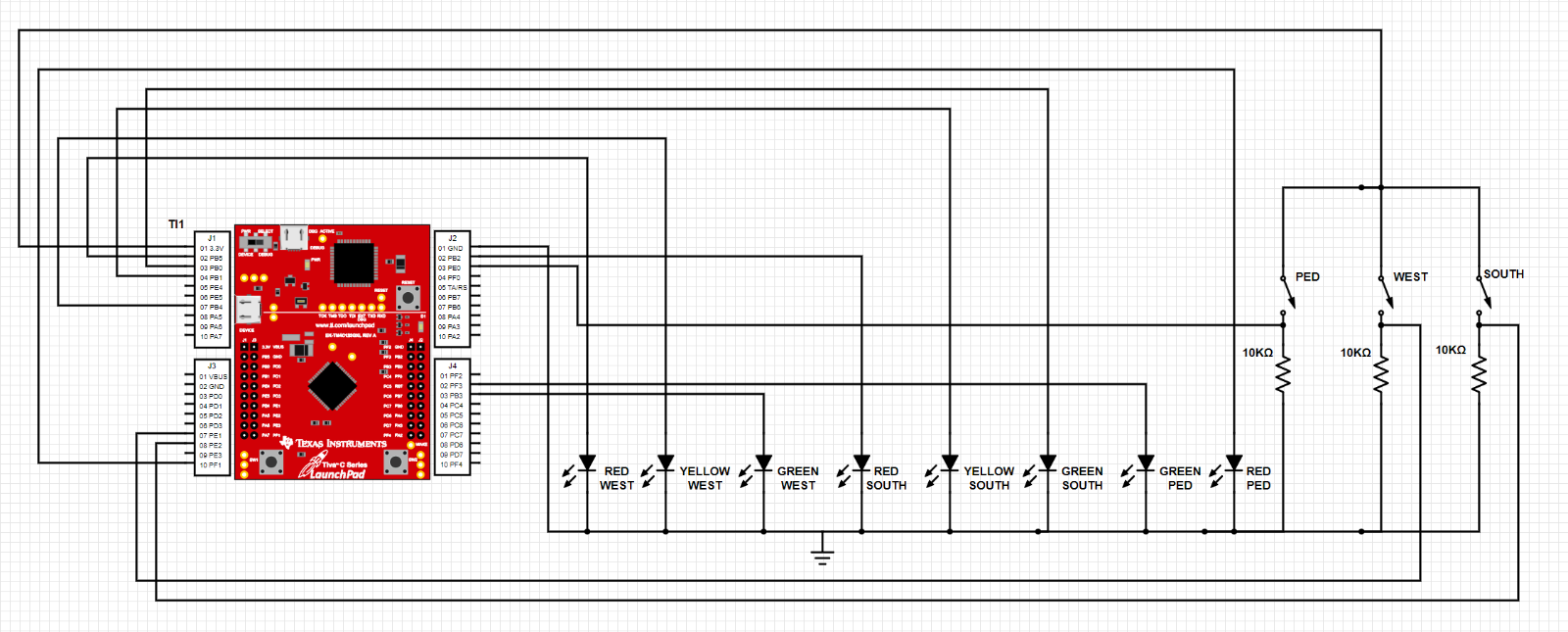
**Theory**

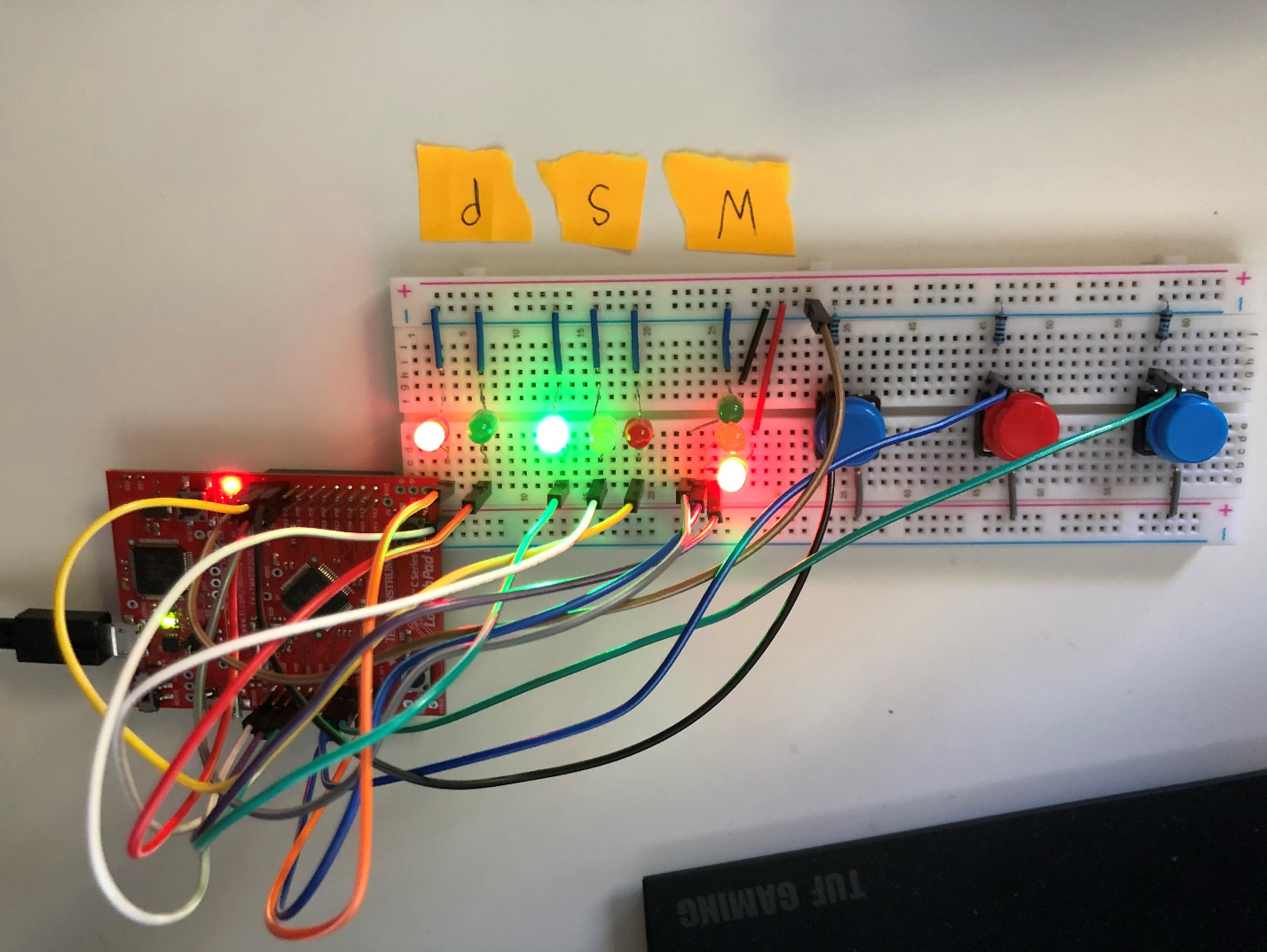
Implementation of the traffic light LEDs and sensors was not possible without a complete understanding of the GPIO. For example, *SENSORS*, the traffic light controller was interfaced digitally so it was crucial to enable digital signals on ports used by *SENSORS*. Learning to enable the pull-down resistors via GPIO for the ports used by *SENSORS* would have spared us three resistors.

Since using a software loop delay was efficient, but not accurate, learning to implement a hardware timer was necessary for the traffic light controller. The SysTick Timer was used due to its ability to generate accurate timing for an embedded system. Since there is a delay of 0.25s, using a SysTick Timer was a must.

Understanding the Moore Finite State Machine was the most important perquisite for this project. Although the project could be done with conditionals, using the FSM makes the process of implementing the traffic light controller more efficient and simpler. Using an FSM makes debugging each state effortless as removing/adding a state does not require multiple conditionals as it only takes one line to do with the FSM. With a FSM, each combination of the *SENSOR* outputs can be accounted for immediately.

3

**Hardware Design**



4

**Software Design**

|  |  |
| --- | --- |
| LEDs | PORT |
| Red West | PB5 |
| Yellow West | PB4 |
| Green West | PB3 |
| Red South | PB2 |
| Yellow South | PB1 |
| Green South | PB0 |
| Walk | PF3 |
| Don’t Walk | PF1 |

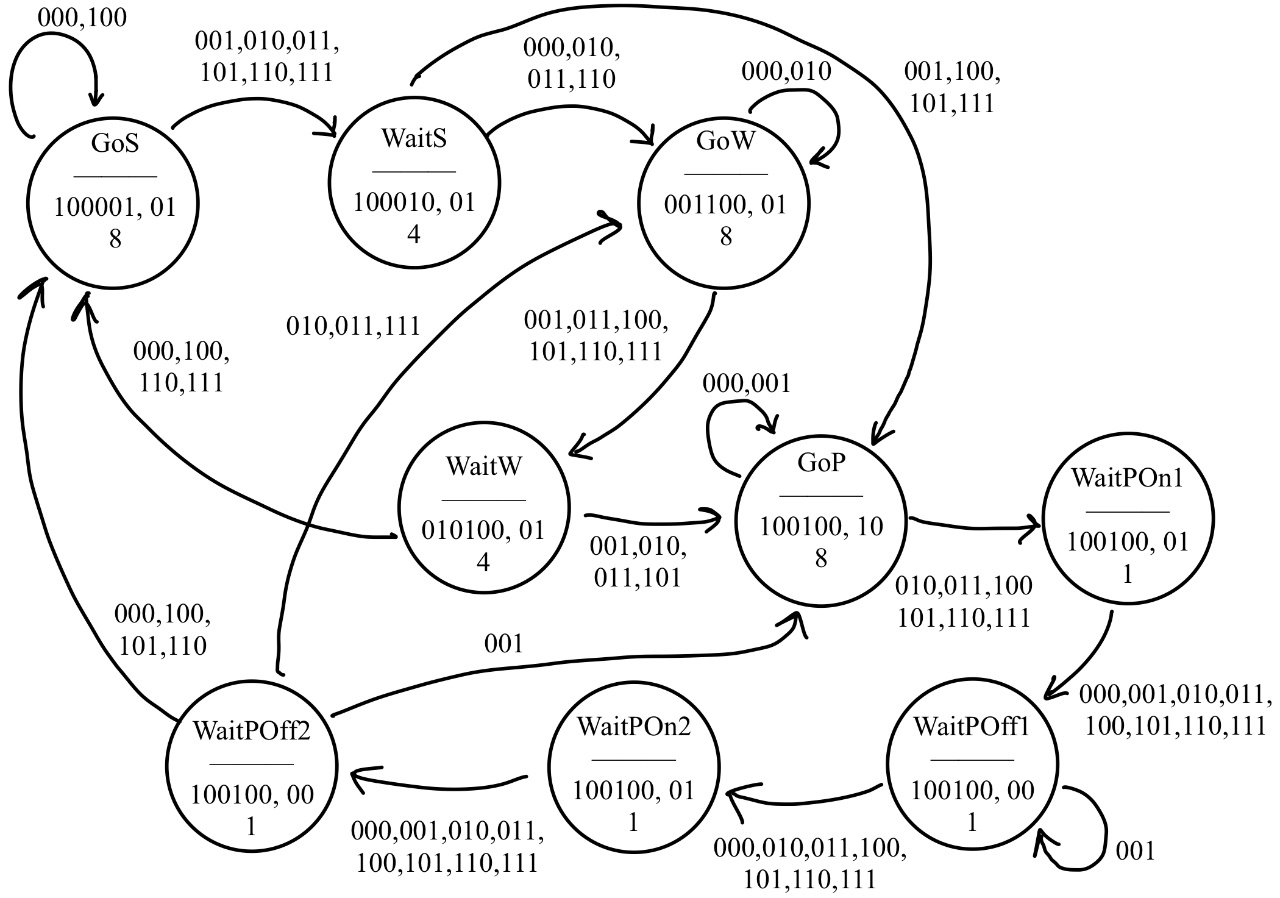
|  |  |
| --- | --- |
| Sensors/button | port |
| South Sensor | PE2 |
| West Sensor | PE1 |
| Pedestrian Walk Button | PE0 |

**Moore FSM: State Table**

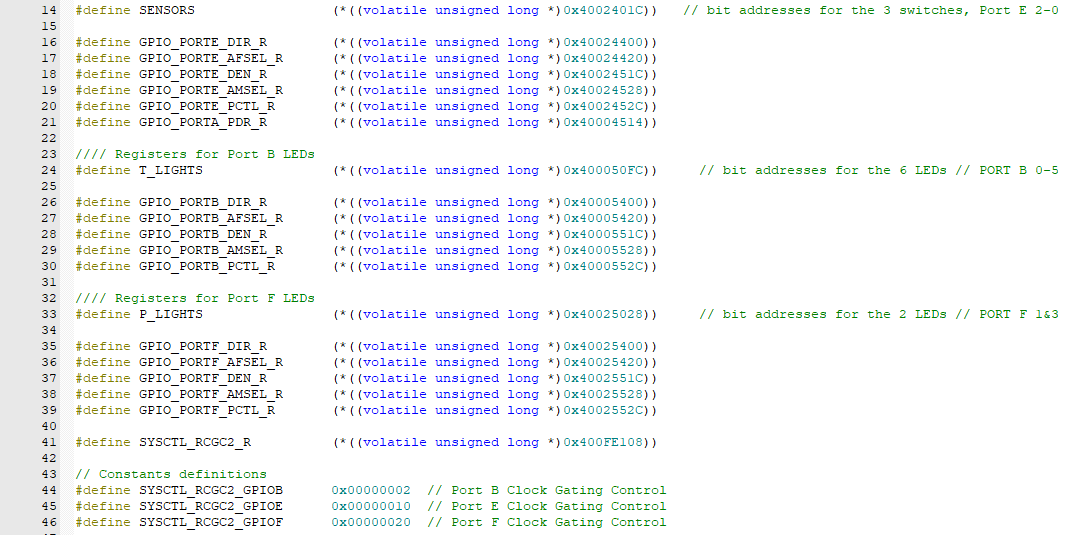
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Current State** | **Time (0.25s)** | **Outputs** | | **Inputs (South, West, Pedestrian) PE2, PE1, PE0** | | | | | | | |
| **PB5-0** | **PF3,1** | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |
| **GoS** | **8** | **100001** | **01** | **GoS** | **WaitS** | **WaitS** | **WaitS** | **GoS** | **WaitS** | **WaitS** | **WaitS** |
| **WaitS** | **4** | **100010** | **01** | **GoW** | **GoP** | **GoW** | **GoW** | **GoP** | **GoP** | **GoW** | **GoP** |
| **GoW** | **8** | **001100** | **01** | **GoW** | **WaitW** | **GoW** | **WaitW** | **WaitW** | **WaitW** | **WaitW** | **WaitW** |
| **WaitW** | **4** | **010100** | **01** | **GoS** | **GoP** | **GoP** | **GoP** | **GoS** | **GoP** | **GoS** | **GoS** |
| **GoP** | **8** | **100100** | **10** | **GoP** | **GoP** | **WaitPOn1** | **WaitPOn1** | **WaitPOn1** | **WaitPOn1** | **WaitPOn1** | **WaitPOn1** |
| **WaitPOn1** | **1** | **100100** | **01** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** | **WaitPOff1** |
| **WaitPOff1** | **1** | **100100** | **00** | **WaitPOn2** | **WaitPOff1** | **WaitPOn2** | **WaitPOn2** | **WaitPOn2** | **WaitPOn2** | **WaitPOn2** | **WaitPOn2** |
| **WaitPOn2** | **1** | **100100** | **01** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** | **WaitPOff2** |
| **WaitPOff2** | **1** | **100100** | **00** | **GoS** | **GoP** | **GoW** | **GoW** | **GoS** | **GoS** | **GoS** | **GoW** |

5

**Moore FSM: State Transition Diagram {S, W, P}**

****

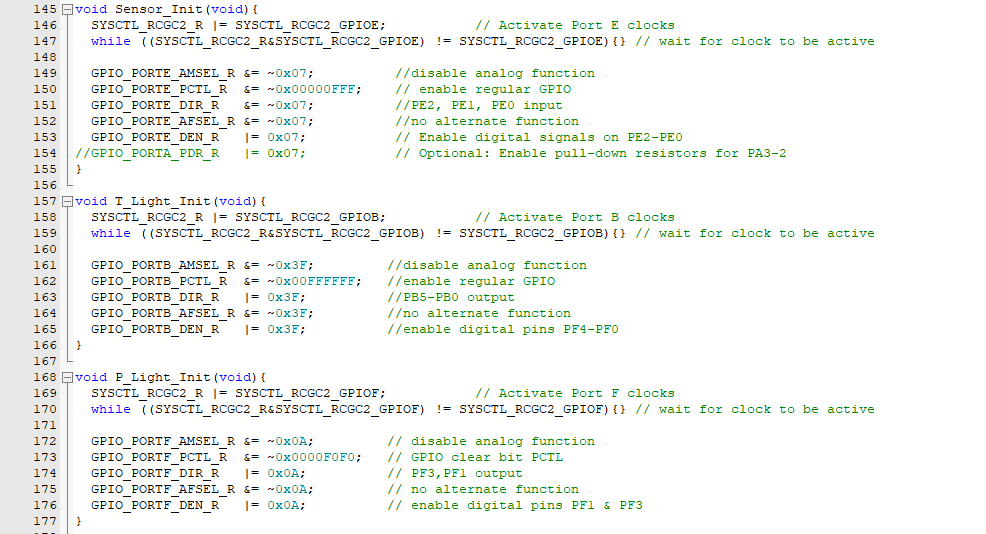
**GPIO Initialization**

****

In this block, instantiating the following GPIO ports allowed the use of Ports E, B, and F. Port E was initialized for buttons controlling the traffic & pedestrian sensor/buttons. Port B was initialized for the 6 LEDs of the 2 sets of traffic lights. Port F was initialized for the “Walk-Hurry-Don’t walk” transitions of the pedestrian LEDs.

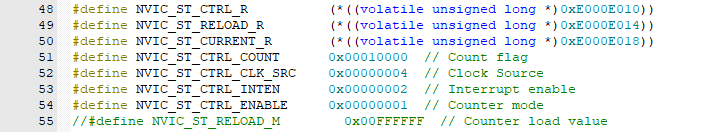
6

**Sensor/Traffic Light/Pedestrian Light Functions**

****

Because the GPIO ports were initialized, creating the functions using each of the ports was possible. The bit value used for each of the functions was obtained due to the constraints of which pins were available. Since PE2-0 were the only pins useable out of the total 6 of Port E, taking the hex value of the pins from the most significant bit we get 00000111 which equals 0x07.

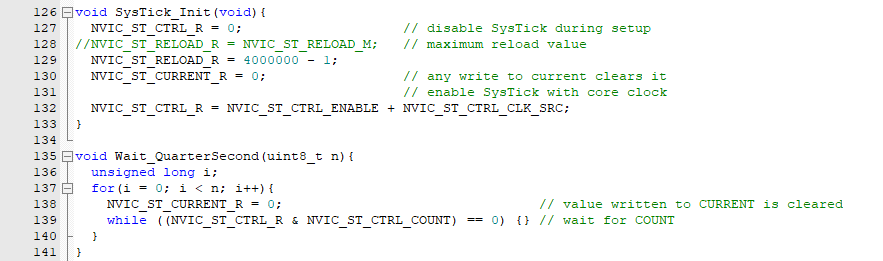
**SysTick Timer Initialization**

****

To initialize the SysTick Timer, setting ENABLE to ‘1’ will start the counter mode. Next, set the RELOAD to a desired value and then clear the counter. Clearing the counter can be done by writing any value to NVIC\_ST\_CURRENT\_R. Setting the Clock Source is required for the TM4C123 board. Setting ITEN to a ‘0’ disables interrupt and ‘1’ enables interrupt.

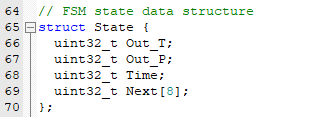
7

**SysTick Timer Function**

****

In the SysTick\_Init(void) function, RELOAD is set to ‘4000000 – 1’ to get the desired time of 0.25 seconds. In the Wait\_QuarterSecond(uint8\_t n) function, the while loop waits for the counter to decrement from ‘4000000 – 1’ to 0 which equals 0.25 seconds. After it the counter equals 0, writing any value to NVIC\_ST\_CURRENT\_R will clear the counter, starting back at ‘4000000 – 1’.

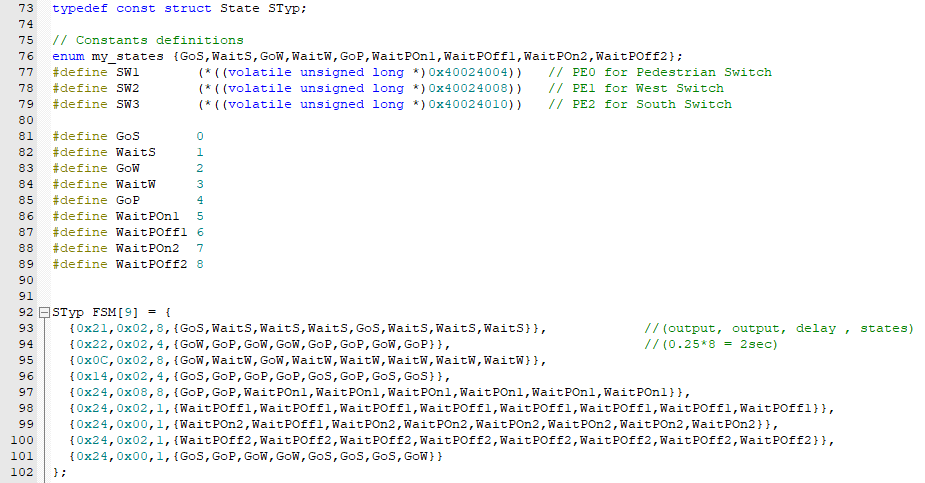
**FSM: Definition of structure**

****

Since there are two Ports that control LEDs, Port B and Port F, it is necessary to have two outputs in the FSM. Out\_T represents the LED output for the traffic lights and Out\_P represents the LED output for the pedestrian. Next[8] is used because there are 8 different inputs that can be taken from the sensors/buttons.

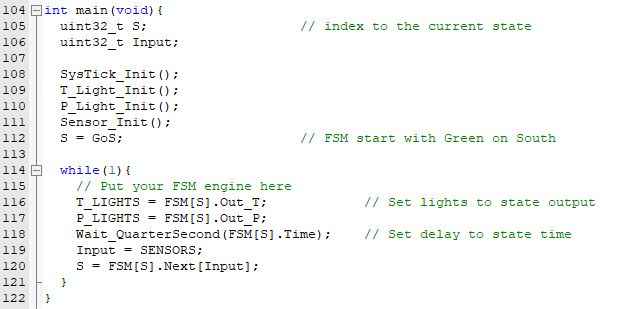
8

**Definition of Finite State Machine**

****

SW1, SW2, and SW3 were defined as inputs with the relevant bit addresses for each of their pins. Since there 9 states in the FSM, there are 9 individual states defined. Next is converting the transition state diagram from before into a Moore FSM. For each state, there are two outputs and a delay. Since the first state, GoS, needs a delay of 2 seconds, 8 is used because 8\*0.25 = 2 seconds.

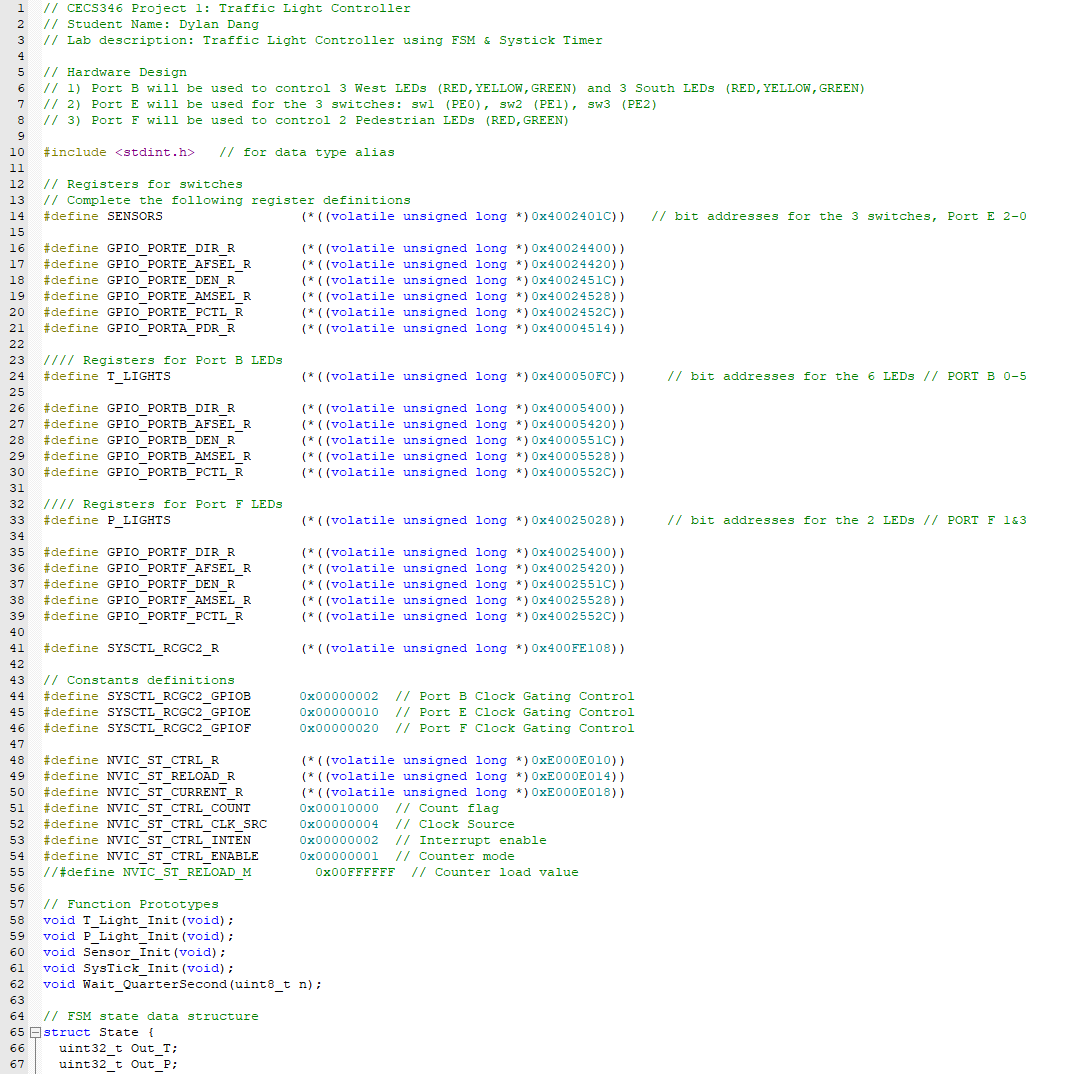
**FSM Engine**

****

In main, the functions that have been defined are called. The start state for the FSM is GoS. In the FSM engine, the two LED outputs are set to the state outputs. This is what allowed the definition of the FSM from above to have two outputs. After every transition there is a delay and if there are any inputs, the FSM will read it and will output the state accordingly.

9

**Source Code:**

**Text

Description automatically generated** 10

**Text

Description automatically generated**

**Conclusion**

Throughout this project, there were many failures. There were software failures, hardware failures, and mental failures. For the software failures, it wasn’t that bad. Because of the inherent design of the FSM, debugging was quick. It was easy to comment out some of the states to debug the others. Implementing an FSM instead of a bunch of conditionals was a success in and of itself. Most of the hardware failures were on me. The wiring was straightforward, but the breadboard that I was equipped with did not want to cooperate. The holes in breadboard were very tight, thus no resistors, LEDs, and buttons would easily plug in. Switching out the entire breadboard was my hardware success, as everything was able to plug in with no issues. For mental failures, it was entirely on me as I had to re-evaluate my knowledge on FSMs. Implementing the FSM into code was simple enough, but converting the instructions given to me into a state table or graph was somewhat difficult for. What made it even more difficult was making sure each participant had a fair chance at a green light. Reading over lecture slides and referencing example labs gave me confidence during my FSM journey and thus was my mental success. Overall, this lab taught me the importance and value of the Moore Finite State Machine. I learned that many things in life are run on an FSM. I feel confident that I will be able to create a Moore FSM for anything I want now and in the future.

11