CECS 346 Fall 2021 Project # 1

Traffic Light Controller

By

Anand Jasti

October 15, 2021

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

CECS 346 Project 1 Report

Introduction

Consider a 4-corner intersection, there are two one-way streets where cars travel toward south on one street and cars travel toward west on the other street. There are three inputs to the LaunchPad, two are car sensors, and one is a pedestrian button. The south car sensor will be true (3.3V) if one or more cars are near the intersection on south. The west car sensor will be true (3.3V) if one or more cars are near the intersection on west. The pedestrian sensor will be true (3.3V) if at least one pedestrian is present, and they wish to cross in any direction. All three sensors will be false (0V) when there are no cars or pedestrians near the intersection. We will use 6 LEDs to represent the two Red-Yellow-Green traffic lights, and we will use a green LED on PF3 to represent the "walk" light and a red Led on PF1 to represent "don't walk" light for the pedestrian. When the "walk" light is on, pedestrians are allowed to cross. When the "don't walk" light flashes, pedestrians should hurry up and finish crossing. When the "don't walk" light is on steady, pedestrians should not enter the intersection. The time durations for traffic lights are 2 seconds for green/walk, 1 second for yellow/hurry, and 3 seconds for red/don't walk. The "hurry up" sequence will flash the "don't walk" LED for 1 second at 0.25 second speed. We will implement this using a Moore machine and use SysTick timer to generate delay.

Operation

We will use 6 LEDs to represent the two Red-Yellow-Green traffic lights, and we will use a green LED on PF3 to represent the "walk" light and a red Led on PF1 to represent "don't walk" light for the pedestrian. Red west (LED) will be connected to PB5, Yellow west (LED) will be connected to PB4, Green west (LED) will be connected to PB3, Red south (LED) will be connected to PB2, Yellow south (LED) will be connected to PB1, Green south (LED) will be connected to PB0. The South sensor (slide switch) will be connected to PE2, West sensor (slide switch) will be connected to PE1, Pedestrian walk sensor (slide switch) will be connected to PE0. We will be using slide switches for all three sensors. We would like to call the two streets plus the pedestrian as three participants. When no participants need green light, we stay in the current state or finish the transition to green. If one participant needs green, we turn on the green for that participant and stay at that state if no other participant need green. If there are more the one participant needing green, we cycle through the request servicing each participants needs. We need to provide fair chances for each participant involved in the cycle. The only valid transition for a traffic light is green-yellow-red. No other transition is allowed. The only valid transition for pedestrian lights is walk-hurry-don't walk, no other sequence is allowed. If one participant starts a green-yellow-red transition, it needs to finish the whole transition before giving green to one of the other two participants. When two participants compete for green light, we make sure to give each one a chance to get green. The system will start with green on south.

Link to Demonstration video:
https://drive.google.com/file/d/1VV6YcyJuq4cMqffnZUwydkhXFu3yCwZn/view?usp=sharing

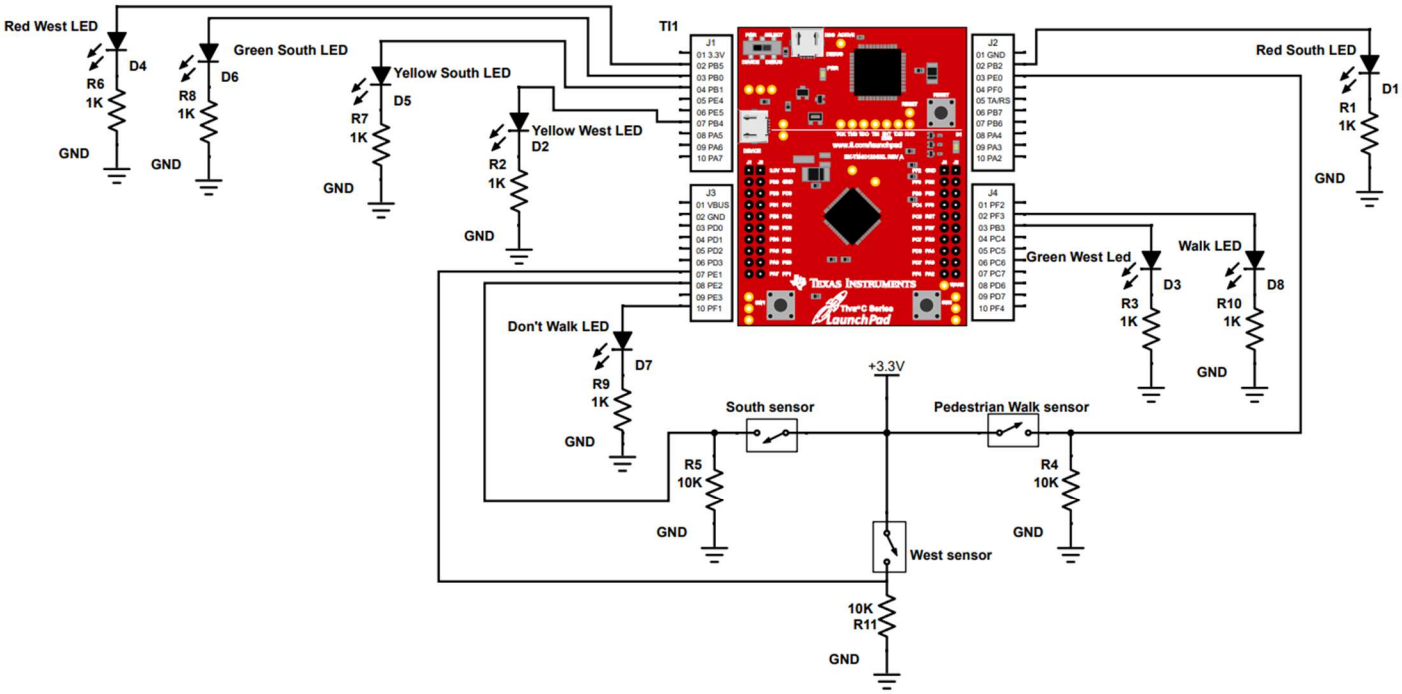
---

Theory

This project uses ARM Cortex TM4C123GH6PM Microcontroller. For this project we will use three of the six General-Purpose I/O ports (PB,PF,PE). We are using some of port B pins and port F pins for outputs, some pins from port E will be used for input (as specified in Operation and Hardware design). This project also uses a Moore Finite State Machine to implement system requirements for a simple traffic light(as specified in Software design). We use a Systick Timer with an internal oscillator (PIOSC) at 16MHz provided by the microcontroller to create a delay. This delay is used to implement the different time durations required for the traffic lights (as specified in the Introduction).

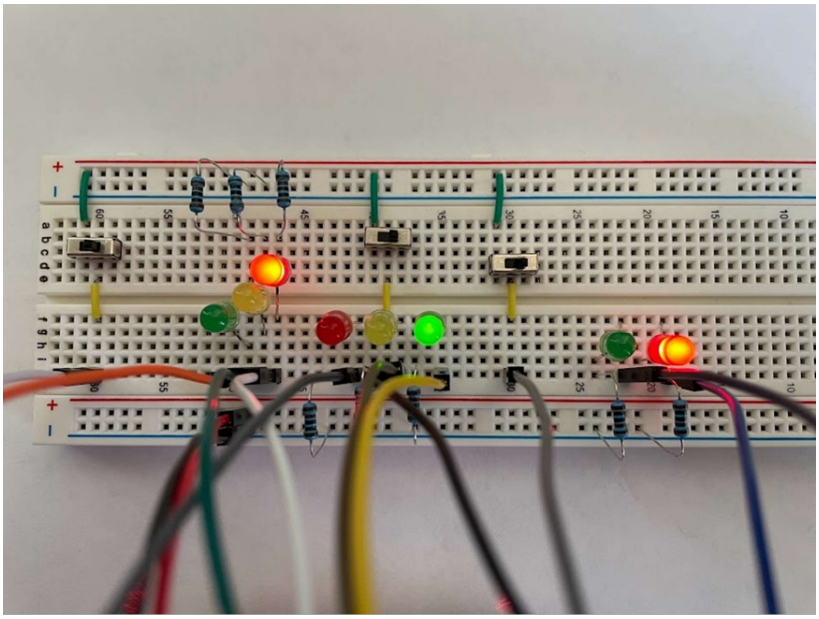
Hardware design

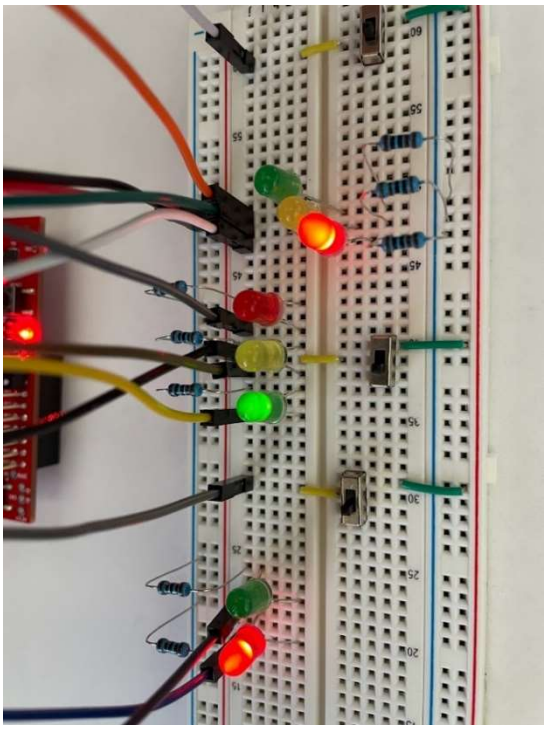
Schematic:



Pin	Devices
PF1	“Don’t walk” Red LED
PF3	“Walk” Green LED
PB5	West Red LED
PB4	West Yellow LED
PB3	West Green LED
PB2	South Red LED
PB1	South Yellow LED
PB0	South Green LED
PE2	South Sensor
PE1	West Sensor
PE0	Pedestrian Walk Sensor

Picture of Hardware System:





Software design

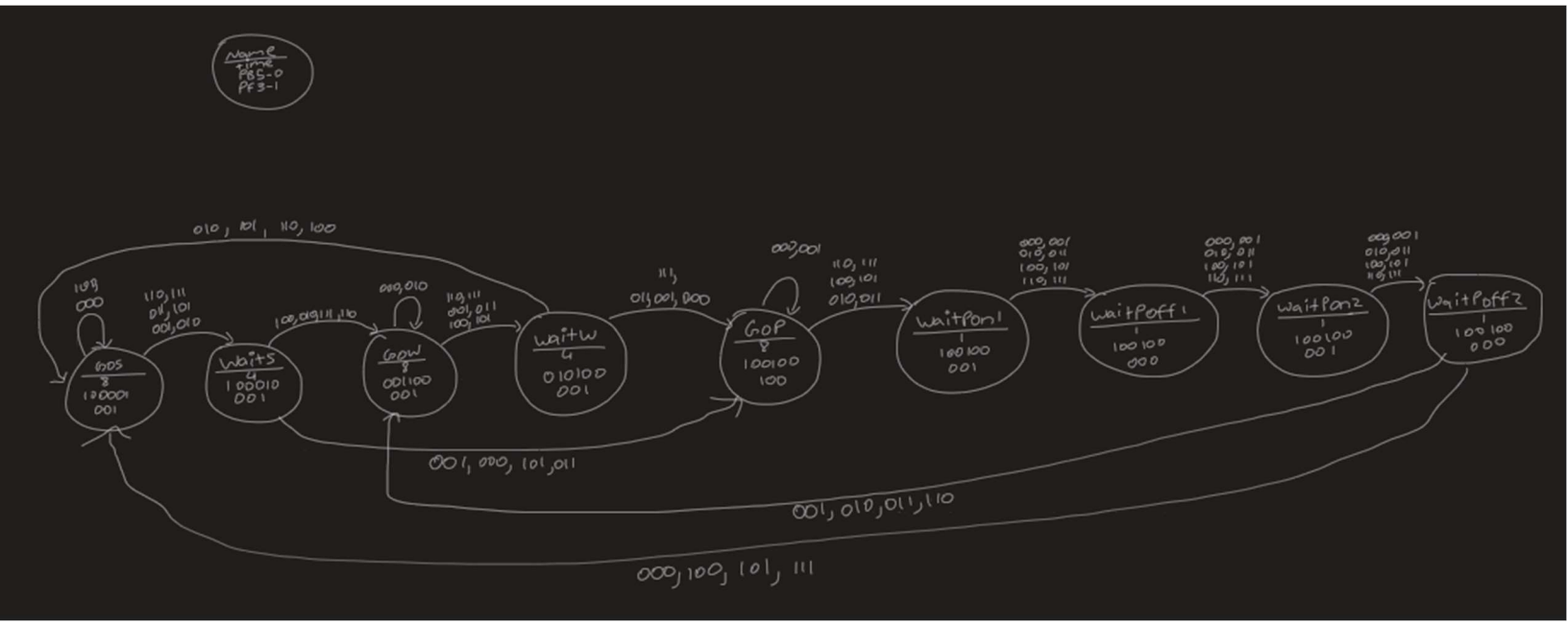
Using a Moore Finite State Machine, we first start at state GoS which turns on South Green Led, West Red Led, and “Don’t Walk” Red Led. When we detect nothing on the three sensors we stay at this state. We also stay in this state when we detect a car at the south sensor and detect nothing on the other two sensors. When we detect any other combinations on the three sensors, we move to the next state WaitS. WaitS turns on South Yellow Led, West Red Led, and “Don’t Walk” Red Led. When in this state if we detect a

one on the Pedestrian walk sensor, or West Sensor and Pedestrian walk sensor, or South Sensor and Pedestrian walk sensor, or nothing on the sensors, we go to the next state GoP. If we detect a one on the West Sensor, or South Sensor, or West Sensor and South Sensor, or all three sensors, we go to the next state GoW. GoW turns on South Red Led, West Green Led, and "Don't Walk" Red Led. When in this state if we detect a one on the West Sensor, or nothing on the three sensors, we stay in the same state. If we detect any other combinations on the three sensors, we move to the next state WaitW. WaitW turns on South Red Led, West Yellow Led, and "Don't Walk" Red Led. When in this state if we detect a one on the Pedestrian walk sensor, or West Sensor and Pedestrian walk sensor, or on all three sensors, or nothing on all three sensors, we go to the next state GoP. If we detect a one on the West Sensor, or South Sensor, or South Sensor and Pedestrian walk sensor, or South Sensor and West Sensor, we move to the next state GoS. GoP turns on South Red Led, West Red Led, and "Walk" Green Led. When in this state if we detect a one on Pedestrian walk sensor, or nothing on the three sensors, we stay in the same state. If we detect any other combinations on the three sensors, we move to the next satate WaitPOn1. WaitPOn1 turns on South Red Led, West Red Led, and "Don't Walk" Red Led. When in this state if we detect any combinations on the three sensors, we move to the next state WaitPOff1. WaitPOff1 turns on South Red Led, and West Red Led. When in this state if we detect any combinations on the three sensors, we go to the next state WaitPOn2. WaitPOn2 turns on South Red Led, West Red Led, and "Don't Walk" Red Led. When in this state if we detect any combinations on the three sensors, we move to the next state WaitPOff2. WaitPOff2 turns on South Red Led, and West Red Led. When in this state if we detect a one on the South sensor, or South sensor and Pedestrian walk sensor, or on all three sensors, or nothing on all three sensors, we move to the next state GoS. If we detect a one on the Pedestrian walk sensor, or West sensor, or West sensor and Pedestrian walk sensor, or South sensor and West sensor, we go to the next state GoW.

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	GoP	GoP	GoW	GoP	GoW	GoP	GoW	GoW
GoW	8	001100	01	GoW	WaitW	GoW	WaitW	WaitW	WaitW	WaitW	WaitW
WaitW	4	010100	01	GoP	GoP	GoS	GoP	GoS	GoS	GoS	GoP
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	WaitPOn2	WaitPOn2	WaitPOn2	WaitPOn2	WaitPOn2	WaitPOn2	WaitPOn2
WaitPOn2	1	100100	01	WaitPOff2	WaitPOff2	WaitPOff2	WaitPOff2	WaitPOff2	WaitPOff2	WaitPOff2	WaitPOff2
WaitPOff2	1	100100	00	GoS	GoW	GoW	GoW	GoS	GoS	GoW	GoS

Moore FSM State Diagram:



Software Source Code:

```
// Subroutine to initialize port E pins for input
// PE2, PE1, PE0 are inputs for Swtich 1, Switch 2, Switch 3
// Inputs: None
// Outputs: None
void PortE_Init(void){
  SYSCTL_RCGC2_R |= 0x00000010;       // activate E clock
  while((SYSCTL_RCGC2_R&0x00000010)!=0x00000010){} // wait for the clock to be ready

  GPIO_PORTE_AMSEL_R &= ~0x07;        // disable analog function
  GPIO_PORTE_PCTL_R &= ~0x00000FFF;   // GPIO clear bit PCTL
  GPIO_PORTE_DIR_R &= ~0x07;          // PE2, PE1, PE0 inputs
  GPIO_PORTE_AFSEL_R &= ~0x07;        // no alternate function
  GPIO_PORTE_PDR_R |= 0x07;           // enable pull-down resistor on PE2, PE1, PE0
  GPIO_PORTE_DEN_R |= 0x07;           // enable digital pins PE2-PE0
}

// Subroutine to initialize Port B pins for output
// PB0, PB1, PB2, PB3, PB4, PB5 are outputs for 6 LEDs
// Inputs: None
// Outputs: None
void PortB_Init(void){
  SYSCTL_RCGC2_R |= 0x00000002;       // activate B clock
  while((SYSCTL_RCGC2_R&0x00000002)!=0x00000002){} // wait for the clock to be ready

  GPIO_PORTB_AMSEL_R &= ~0x3F;        // disable analog function
  GPIO_PORTB_PCTL_R &= ~0x00FFFFFF;   // GPIO clear bit PCTL
  GPIO_PORTB_DIR_R |= 0x3F;           // PB0, PB1, PB2, PB3,PB4, PB5 outputs
  GPIO_PORTB_AFSEL_R &= ~0x3F;        // no alternate function
  GPIO_PORTB_DEN_R |= 0x3F;           // enable digital pins PB5-PB0
}

// Subroutine to initialize Port F pins for output
// PF1 and PF3 are outputs for 2 Pedestrian LEDs
// Inputs: None
// Outputs: None
void PortF_Init(void){
  SYSCTL_RCGC2_R |= 0x00000020;       // activate F clock
  while((SYSCTL_RCGC2_R&0x00000020)!=0x00000020){} // wait for the clock to be ready

  GPIO_PORTF_AMSEL_R &= ~0x0A;        // disable analog function
  GPIO_PORTF_PCTL_R &= ~0x0000F0F0;   // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R |= 0x0A;           // PE1, PE3 outputs
  GPIO_PORTF_AFSEL_R &= ~0x0A;        // no alternate function
  GPIO_PORTF_DEN_R |= 0x0A;           // enable digital pins PB5-PB0
}
```

In this bit of code above I am initializing Ports E,B, and F. Port E is being initialized to disable analog function, clear GPIO bit PCTL, set no alternate function, enable pull-down resistor, enable digital pins, and set pins PE2,PE1,PE0 as inputs. Port B is being

initialized to disable analog function, clear GPIO bit PCTL, set no alternate function, enable digital pins, and set pins PB5-0 as outputs. Port F is being initialized to disable analog function, clear GPIO bit PCTL, set no alternate function, enable digital pins, and set pins PE1, PE3 as outputs.

```
63  // define constants and alies
64  #define P_LIGHTS  (*((volatile unsigned long *)0x40025028))     // PORTF, bits 1,3
65  #define T_LIGHTS  (*((volatile unsigned long *)0x400050FC))     // PORTB, bits 0,1,2,3,4,5
66  #define SENSORS   (*((volatile unsigned long *)0x4002401C))     // PORTE, bits 0,1,2
67
```

In this bit of code I am using bit specific addressing to define P_LIGHTS to access Port F bits 1 and 3, T_LIGHTS to access Port B bits 0,1,2,3,4, and 5, SENSORS to access Port E bits 0,1, and 2.

```
55  // SysTick register definitions
56  #define NVIC_ST_CTRL_R          (*((volatile unsigned long *)0xE000E010))
57  #define NVIC_ST_RELOAD_R        (*((volatile unsigned long *)0xE000E014))
58  #define NVIC_ST_CURRENT_R       (*((volatile unsigned long *)0xE000E018))
59
```

In this code I'm using bit specific addressing to define NVIC_ST_CTRL_R as the SysTick Control Register. I also defined NVIC_ST_RELOAD_R and NVIC_ST_CURRENT_R as the 24-bit Reload register and 24-bit Current register respectively.

```
83  // Subroutine to initialize SysTick Timer
84  // Inputs: None
85  // Outputs: None
86  // Initialize the SysTick timer with maximum reload value
87  // Enable SysTick timer with system clock
88  void SysTick_Init(void){
89    NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
90    NVIC_ST_RELOAD_R = 4000000 - 1;       // number of counts to wait 0.25 seconds
91    NVIC_ST_CURRENT_R = 0;                // any write to CURRENT clears
92                                          // enable SysTick with core clock
93    NVIC_ST_CTRL_R |= NVIC_ST_CTRL_CLK_SRC;
94  }
95
96  // Subroutine to wait 0.25 sec
97  // Inputs: None
98  // Inputs: None
99  // Use busy waiting approach to generate n*0.25s delay
00  // Enable SysTick timer with system clock
01  void Wait_250Milliseconds(uint8_t n){
02    unsigned long i;
03    NVIC_ST_CTRL_R |= NVIC_ST_CTRL_ENABLE;
04    for(i=0; i<n; i++){
05      NVIC_ST_CURRENT_R = 0;              //any value written to CURRENT clears
06      while ((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0){ }  // wait for COUNT
07    }
08    NVIC_ST_CTRL_R &= ~NVIC_ST_CTRL_ENABLE;
09  }
10
```

In this screen shot I am initializing the SysTick timer reload value to 4million minus 1 counts to wait ¼ seconds. And clear the 24-bit Current value of the SysTick counter. After setting the reload value and clearing the current register I enabled SysTick

CLK_SRC. In the second function I implement a busy way delay to generate a n*0.25s delay, where “n” is the desired amount of times we want to wait 0.25s.

```
82  // FSM state data structure
83  struct State{
84    uint32_t Out1;
85    uint32_t Out2;
86    uint32_t Time;
87    uint32_t Next[8];
88  };
89
```

The Moore Finite State Machine data structure is as follows. The first variable Out1 is the output for the six Traffic Lights. The second variable Out2 is the output for the two Pedestrian Lights. The third variable Time is the output for the SysTick Timmer especially the “n” mentioned before for the desired amount of times we want to wait 0.25s. The last variable Next is an array with length 8, this array holds the next states which are defined by the index of the array. The 8 indices of the array relates to the 8 different combinations we can have with three different sensors.

```
103 STyp FSM[9]={
104   {0x21,0x02,8,{GoS,WaitS,WaitS,WaitS,GoS,WaitS,WaitS,WaitS}},
105   {0x22,0x02,4,{GoP,GoP,GoW,GoP,GoW,GoP,GoW,GoW}},
106   {0x0C,0x02,8,{GoW,WaitW,GoW,WaitW,WaitW,WaitW,WaitW,WaitW}},
107   {0x14,0x02,4,{GoP,GoP,GoS,GoP,GoS,GoS,GoS,GoP}},
108   {0x24,0x08,8,{GoP,GoP,WaitPOn1,WaitPOn1,WaitPOn1,WaitPOn1,WaitPOn1,WaitPOn1}},
109   {0x24,0x02,1,{WaitPOff1,WaitPOff1,WaitPOff1,WaitPOff1,WaitPOff1,WaitPOff1,WaitPOff1,WaitPOff1}},
110   {0x24,0x00,1,{WaitPOn2,WaitPOn2,WaitPOn2,WaitPOn2,WaitPOn2,WaitPOn2,WaitPOn2,WaitPOn2}},
111   {0x24,0x02,1,{WaitPOff2,WaitPOff2,WaitPOff2,WaitPOff2,WaitPOff2,WaitPOff2,WaitPOff2,WaitPOff2}},
112   {0x24,0x00,1,{GoS,GoW,GoW,GoW,GoS,GoS,GoW,GoS}}
113 };
114
```

This the how I choose to define my Moore Finite State Machine. This definition follows the state table listed above.

```
  S = GoS;

  while(1) {
    T_LIGHTS = FSM[S].Out1;    // set traffic lights
    P_LIGHTS = FSM[S].Out2;    //set pedestrian lights
    Wait_250Milliseconds(FSM[S].Time);
    Input = SENSORS;           // read switch
    S = FSM[S].Next[Input];
  }
```

This is the Finite State Machine Engine. It first starts the state with Gos, and when in the while loop it first sets the Traffic Lights to the first variable of the state. Then it sets the Pedestrian Lights to the second variable of the state. After setting the Pedestrian Lights we then call the Wait_250Millisecons with the third variable of the state. Then we read the Sensors by setting Input to SENSORS. And finally we change the state to the next state based on the Input variable.

Conclusion

This simple traffic light project was very fun and challenging to implement. It took way longer to implement this project than I expected, mostly because of some software implementation challenges. After I finished the coding, I ran the simulation on Keil to check my code. While simulating I noticed that the pedestrian lights (PortF) were not right, So I went back to check my code and the state table. I was having problems with how to understand the State table, I read the outputs on my state table wrong. I read 01(PF3-1) as 0x01 and set that to PortF, So the data on PortF was 00001(PF5-0) which is not what was intended by the state table. After figuring this out I went back and changed all the state outputs for pedestrian lights in my code to fix this mistake. This was my only challenge I ran into while implementing this project. I had no problems implementing the hardware because I tested each component beforehand. During this project I learned more about how to implement/use a Moore Finite State Machine into software. I also learned how to debug my software code more efficiently using Keil debugging tool.