CECS 346 Fall 2021 Project #1

Traffic Light Controller

By

Abhishek Jasti

October 15,2021

Brief description: This Project helped me understand and implement index data structures while learning how to create a segmented software system. Also, studying the real-time synchronization by designing a finite state machine controller and to learn how to use a hardware time.

CECS 347 Project Report Template

INTRODUCTION: Given a Launchpad, 3 green, 3 red, 2 yellow LEDs, resistors, and 3 switches to create a traffic light controller to help us understand how to use indexed data structures. Including 2 sets of traffic lights and a set of pedestrian lights to the mix to better understand how to design a finite state machine controller and how to use a hardware timer. This creation of a segmented software system to control the traffic lights to expand our knowledge. Using software skills such as defining data structure for FSM, building Moore FSM engine, creating fixed-time delays using the SysTick timer, and debugging real-time systems. To create a 4-corner intersection we used a traffic light facing west and a traffic light facing south. For south traffic lights we used a south sensor which has 3.3V to determine when the traffic should be green. Like the south traffic light, we used a separate sensor called west sensor for the west traffic light which has 3.3V as well. For the pedestrian lights with a green led and a red led we used a pedestrian sensor which has 3.3V as well. Starting with south traffic light being on, we have the cases to demonstrate the switch between south and west traffic lights or between south traffic light and pedestrian light.

OPERATION: In the beginning the traffic lights should start with a green light on south traffic lights. The sensors used are listed below with a brief detailed description.
SENSORS (PORTE):

1) PEDESTRIAN SENSOR – PE0 being the pedestrian sensor is used to determines whether the pedestrian lights should turn on. Meaning this sensor determines whether the pedestrian or pedestrians can cross the street safely. Mentioned in the project description this means WALK (PF3) or DON'T WALK (PF1).
2) SOUTH TRAFFIC LIGHT SENSOR – PE2 being the south traffic light sensor determines if the cars going south are allowed to pass or not. When this sensor is turned on all the other lights such as the west traffic lights and pedestrian lights should be red. For the south traffic lights the outputs green LED (PB0) is turned on first and stays on green unless any other sensors are turned on. If other sensors were to be turned on the south traffic light has to finish the process by going to the yellow LED (PB1) and then finishing it off with going to the red LED (PB2). Then it proceeds to stay on red until the south traffic light sensor is turned on again.
3) WEST TRAFFIC LIGHT SENSOR – PE1 being the west traffic light sensor. Like the south traffic light sensor, the west traffic light sensor determines if the cars going west are allowed to pass or not. For the west traffic lights the outputs green LED (PB3) is turned on first and should stay on green until another sensor is turned on. If another sensor were to be turned on the west traffic lights should then go to yellow LED (PB4) and then finishing it off with red LED(PB5). Then it proceeds to stay on red until the west traffic light sensor is turned on again.

For example: If the light is on green south traffic lights and the pedestrian sensor is turned on, the south traffic light must go to yellow then red before the pedestrian light is turned green (WALK).

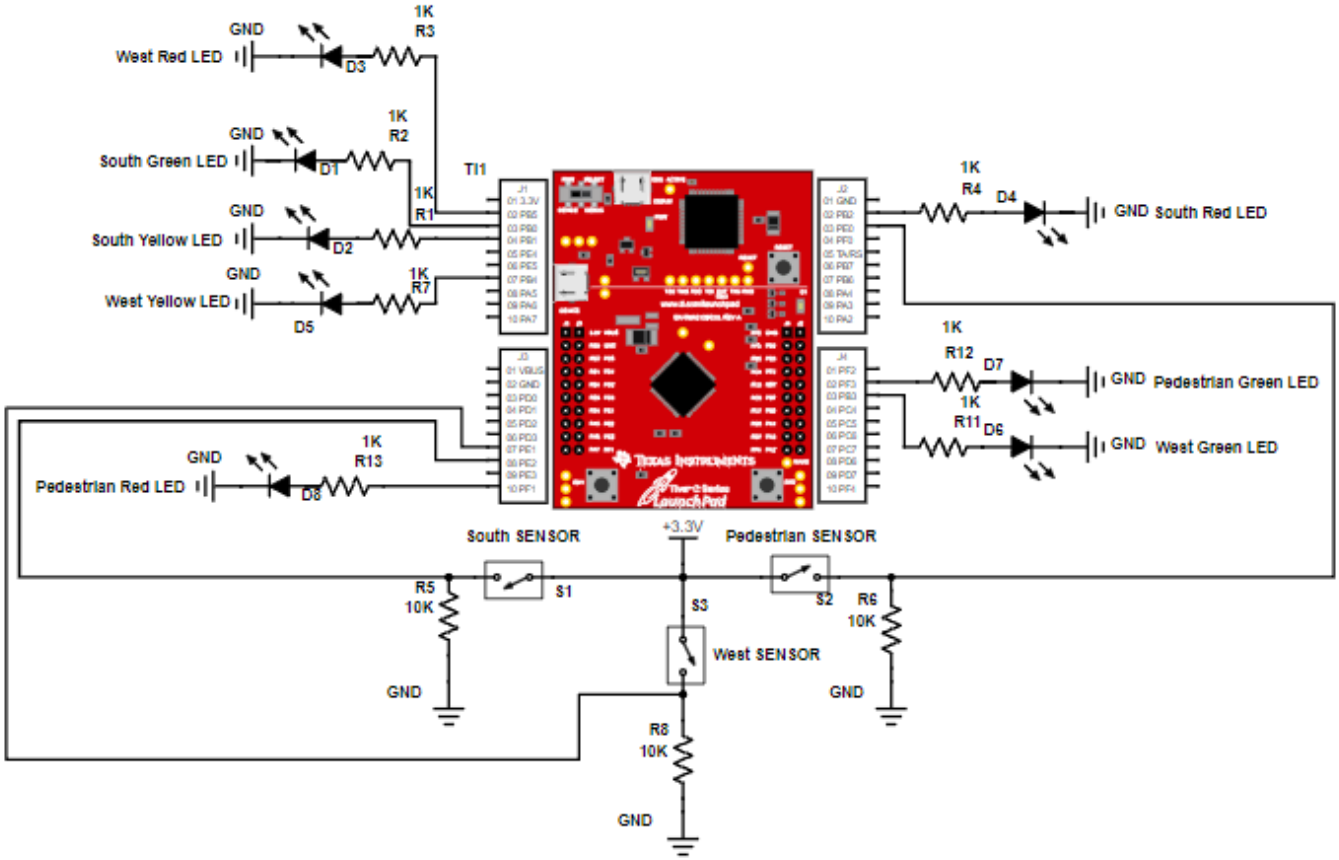
In the Video below are the test cases on what the traffic light controller should do when different sensors are turned on.

Demonstration Video Link:

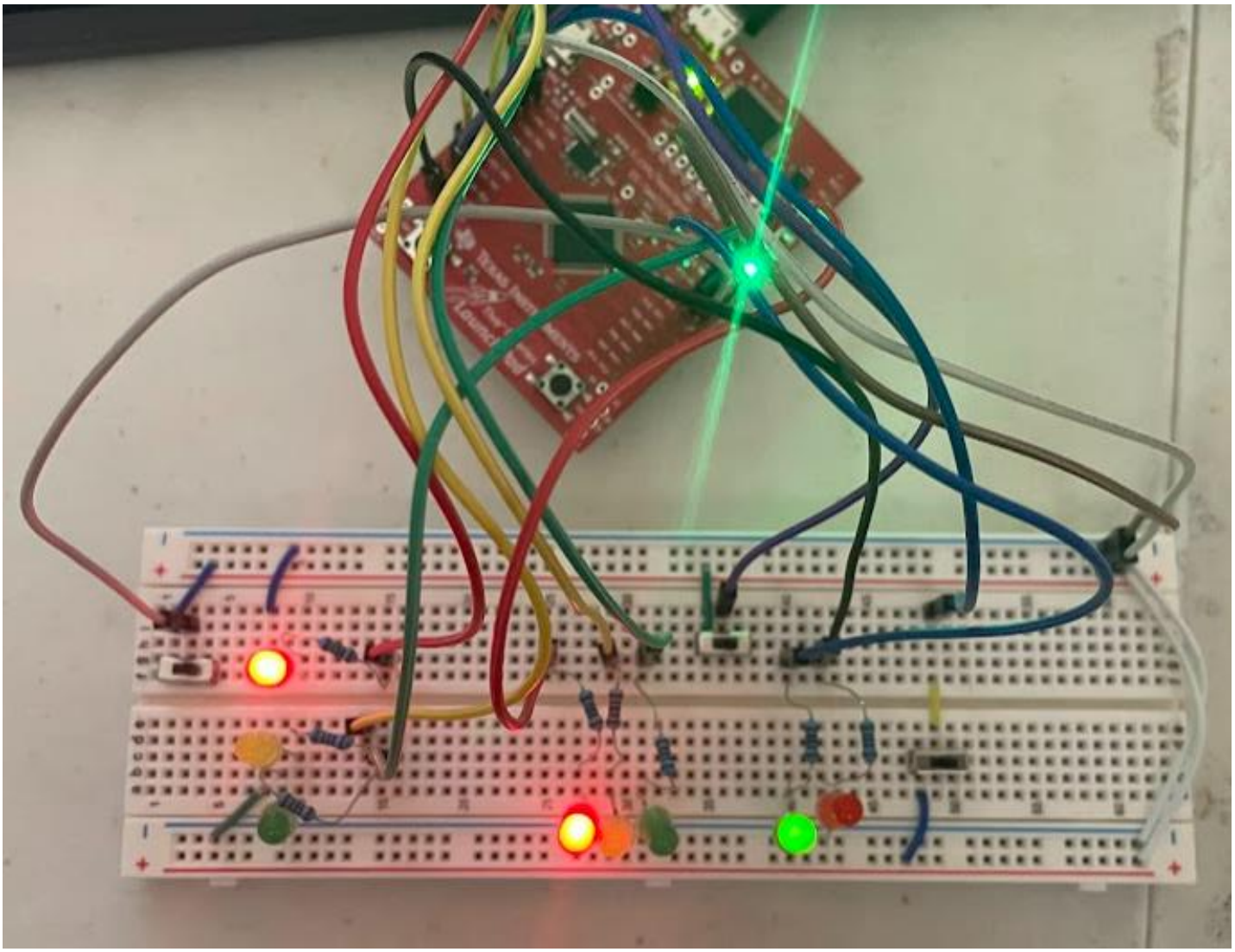
https://drive.google.com/file/d/1Wd4j8vk_45G4ye9ZvZiHQnTGOYj5-WlH/view?usp=sharing

THEORY: With the use of General-Purpose I/O PORTS B, F, and E as ports for inputs such as the sensors and outputs such as the LEDs. PORTB only used for outputs such as the South and West traffic light LEDs. PORTF also only used for outputs for the pedestrian lights. PORTE being used for inputs such as the three sensors. Using a MOORE FINITE STATE Machine to develop this project to its requirements. To create a delay to implement the time differentiations I used a SysTick Timer which is an internal oscillator. With an ARM Cortex Microcontroller providing internal resistors, I decided to use my own external resistors to avoid the problem of the resistance being too big or not working.

HARDWARE DESIGN:



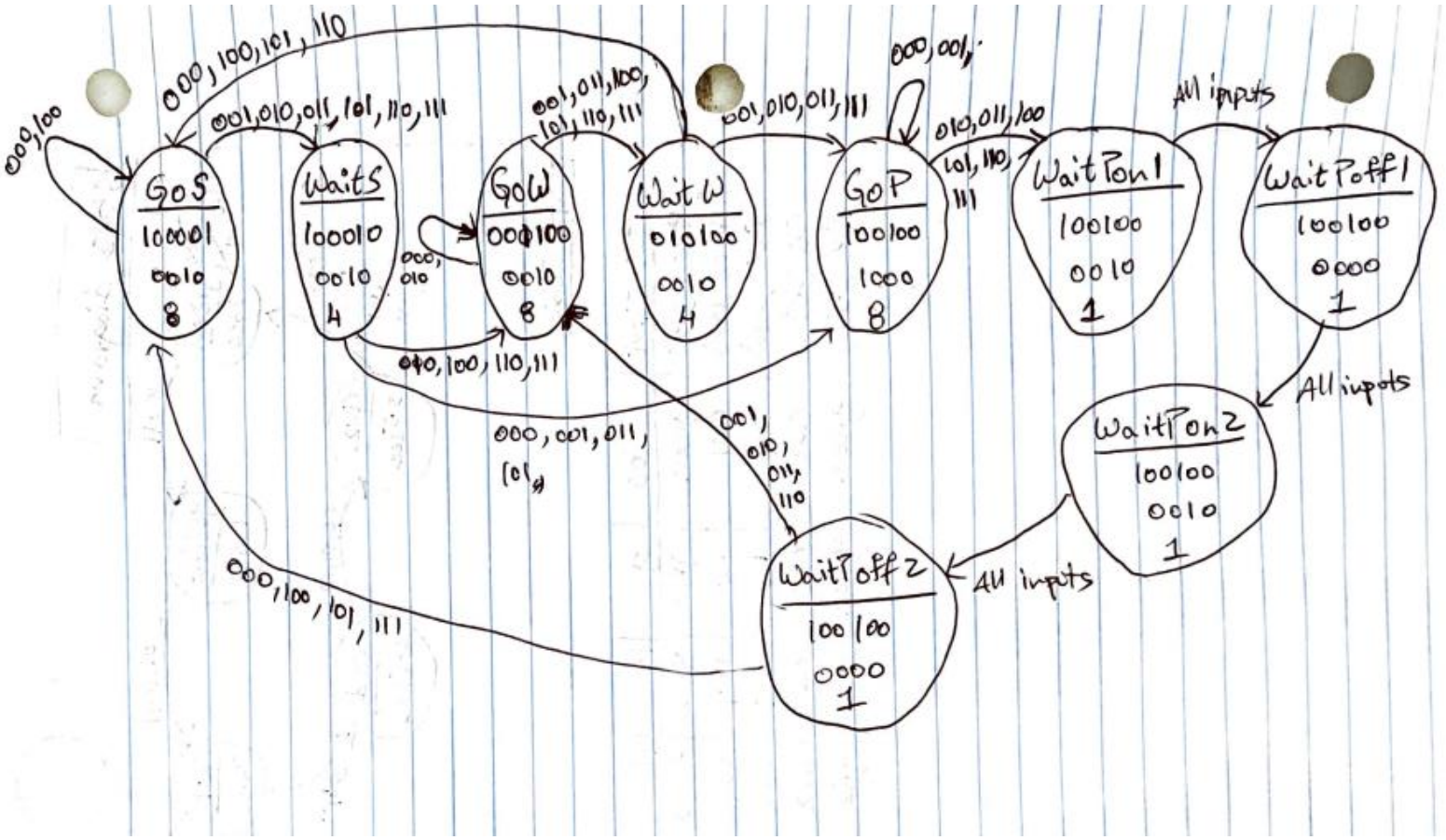
WEST RED LED	PB5
WEST YELLOW LED	PB4
WEST GREEN LED	PB3
SOUTH RED LED	PB2
SOUTH YELLOW LED	PB1
SOUTH GREEN LED	PB0
SOUTH TRAFFIC LIGHT SENSOR	PE2
WEST TRAFFIC LIGHT SENSOR	PE1
PEDESTRIAN LIGHT SENSOR	PE0



SOFTWARE DESIGN:

With the first state GoS (Green Light on South) will stay on GoS (Green Light on South) unless the west traffic light sensor (PE1) or the pedestrian sensor (PE0) is turned on, in that case we will go to the next state which is WaitS (Yellow then Red on South). In the WaitS (Yellow then Red on South) state, we will go to the GoP (Green on Pedestrian Light) state when the pedestrian sensor (PE0) is turned on or we will go to the GoW (Green Light on West) state when the west traffic light sensor (PE1) is turned on. Moving on to GoW (Green Light on West) state, we stay at GoW (Green Light on West) unless the south traffic light sensor (PE2) or the pedestrian sensor (PE0) is turned on, in that case we will go to WaitW (Yellow then Red on West). In the next state GoP (Green on Pedestrian Light), we stay at GoP (Green on Pedestrian Light) unless the south (PE2) or the west traffic light sensor (PE1) is turned on, in that case we will go to WaitPon1 (Red on Pedestrian Light). In WaitPon1 (Red on Pedestrian Light) state, any input will go to the next state WaitPoff1 (Red Light off on Pedestrian). In WaitPoff1 (Red Light flash on Pedestrian) state, any input will go to the next state WaitPon2 (Red flash on Pedestrian Light). In WaitPon2 (Red flash on Pedestrian Light) state, any input (PE1 || PE2) will go to the next case WaitPoff2 (Red Light flash on Pedestrian). In the WaitPoff2 (Red Light flash on Pedestrian) state, depending on the sensor (PE1 || PE2) turned on between west and south traffic lights it will choose its respective state (GoW || GoS). For Example, in WaitPoff2 state, the input or the sensor turned on is the south traffic light sensor then it will go to GoS.

CURRENT STATE	TIME (0.25s)	OUTPUTS (PB5-PB0)	OUTPUTS (PF3,PF0)	INPUT 000	INPUT 001	INPUT 010	INPUT 011	INPUT 100	INPUT 101	INPUT 110	INPUT 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	GoS	GoP	GoP	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	GoW	GoW	GoS



Defining P_Lights with PORTF with bits 1 and 3.
Defining T_Lights with PORTB with bits 0, 1, 2, 3, 4, and 5
Defining SENSORS with PORTE with bits 0, 1, and 2

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

In the while loop:
Setting T_Lights to the state output
Setting P_Lights to the state output
Calling the wait_quartersecond to set delay to state time
Getting sensors inputs

```
while(1) {
  T_LIGHTS = FSM[S].PBOut; //Set light to state output
  P_LIGHTS = FSM[S].PFOut; //Set light to state output
  Wait_quarterSecond(FSM[S].Time); //Set delay to state time
  Input = SENSORS;         //Read Inputs
  S = FSM[S].Next[Input];
```

Defining count flag, clock source, interrupt enable, counter mode, and counter load value for SysTick Timer.

```
// define constants and alias for SysTick Timer
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value
```

Initializing the SysTick Timer with maxim reload value
Using a busy wait approach to generate n*0.25 delay

```
// Subroutine to initialize SysTick timer
// Initialize the SysTick timer with maximum reload value
// Enable Systick timer with system clock
// Inputs: None
// Outputs: None
void SysTick_init(void){
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = 4000000 - 1;       // 0.25sec reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
  // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE + NVIC_ST_CTRL_CLK_SRC;
}

// Subroutine to wait 0.25 sec
// Use busy waiting approach to generate n*0.25s delay
// n is specified by the parameter of this function
// Inputs: None
// Outputs: None
void Wait_quarterSecond(uint8_t n){
  unsigned long i;
  for(i=0; i<n; i++){
    NVIC_ST_CURRENT_R = 0;          // any value writtern to current clears
    while ((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0){} // Wait for count
  }
}
```

Structuring the Finite states

```
//FSM state data structure
struct State{
  uint32_t Time;
  uint32_t PBOut;
  uint32_t PFOut;
  uint32_t Next[8];
};
```

```
STyp FSM[9]={
  {8,0x21,0x02,{GoS,WaitS,WaitS,WaitS,GoS,WaitS,WaitS,WaitS}},
  {4,0x22,0x02,{GoP,GoP,GoW,GoP,GoW,GoP,GoW,GoW}},
  {8,0x0C,0x02,{GoW,WaitW,GoW,WaitW,WaitW,WaitW,WaitW,WaitW}},
  {4,0x14,0x02,{GoS,GoP,GoP,GoP,GoS,GoS,GoS,GoP}},
  {8,0x24,0x08,{GoP,GoP,WaitPon1,WaitPon1,WaitPon1,WaitPon1,WaitPon1,WaitPon1}},
  {1,0x24,0x02,{WaitPoff1,WaitPoff1,WaitPoff1,WaitPoff1,WaitPoff1,WaitPoff1,WaitPoff1,WaitPoff1}},
  {1,0x24,0x00,{WaitPon2,WaitPon2,WaitPon2,WaitPon2,WaitPon2,WaitPon2,WaitPon2,WaitPon2}},
  {1,0x24,0x02,{WaitPoff2,WaitPoff2,WaitPoff2,WaitPoff2,WaitPoff2,WaitPoff2,WaitPoff2,WaitPoff2}},
  {1,0x24,0x00,{GoS,GoW,GoW,GoW,GoS,GoS,GoW,GoS}}
};
```

GPIO Ports
Initializing PortE pins for inputs
Initializing PortB pins PB5, PB4, PB3, PB2, PB1, and PB0 for outputs
Initializing PortF pins PF1 and PF3 for outputs

```
// Subroutine to initialize port E pins for inputs
// 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;        // PE2-0 output
  GPIO_PORTE_AFSEL_R &= ~0x07;      // no alternate function
  GPIO_PORTE_DEN_R |= 0x07;         // enable digital pins PE2-0
  GPIO_PORTE_PDR_R |= 0x07;         // enable pull down resistor on PE2-0
}

// Subroutine to initialize port B pins PB5-0 for outputs
// 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;         // PB5-0 output
  GPIO_PORTB_AFSEL_R &= ~0x3F;      // no alternate function
  GPIO_PORTB_DEN_R |= 0x3F;         // enable digital pins PB5-0
}

// Subroutine to initialize port F pins PF1,PF3 for outputs
// 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;         // PF1,PF3 output
  GPIO_PORTF_AFSEL_R &= ~0x0A;      // no alternate function
  GPIO_PORTF_DEN_R |= 0x0A;         // enable digital pins PF1,PF3
}
```

CONCLUSION:

Implementing this project onto the board seemed much easier than developing the code to run the traffic light controller project. With the code I had issues dealing with the SysTick Timer because of me forgetting to use quarter of a second instead half second. Also, had issues with port initialization for the PortE being inputs only while PortB and PortF being outputs only. With the onboard aspect I didn't have many issues but the change of switches from being push buttons to slide switches made this project a little easier to see the lights switch. I had trouble setting the west traffic lights perpendicular to the south traffic lights but then realized none of my LEDs were connected to power. All in all, this project was not too hard to understand and to implement given the specific requirements.