Traffic Project

EGC 332 Microprocessor Laboratory

George Dagis (CE), [PARTNER NAME OMITTED] (CE)

December 22nd, 2017

Instructor: Michael Otis

Abstract

Modern traffic engineering technology has come a long way since before the modern traffic light. The current traffic engineering systems are often taken for granted; however the project laid out within this report shows that simulations modeling this type of technology is both highly complex and useful. Within this report is a traffic light intersection simulation. This report also contains details of how the hardware and software interact, the theory and research behind the project, and finally the conclusions made.

Table of Contents

Background Information

The information in the following sections may be used to inform readers of what is meant by traffic engineering. The following content also explains the problems that were solved through this project, as well as the significance of the project itself.

1.0 Introduction

Dictionary.com defines traffic engineering as “a branch of civil engineering concerned with the design and construction of streets and roads that will best facilitate traffic movement" [1]. While this definition may be useful to some, in the case of this project, traffic engineering is concerned with the interaction between electronic devices and the traffic that surrounds them, in order to create a safe and effective road system. The latter definition is more fit to the project described within this report.

The project, based on a traffic light, simulates a 4-way intersection. There are two one-way streets labeled south (cars traveling north to south) and west (cars traveling east to west). The Texas Instruments Tiva Arm Cortex m4 launchpad (TM4C123GH6PM) was used and takes three inputs which correspond to three separate push-button switches. The inputs act as sensors; two sensors correspond to the cars arrivals while the last corresponds to a pedestrian.

1.1 Problem Statement

In order to properly simulate a working traffic system, it is important to analyze the system as a whole and define the problems needed to be solved. This is especially in the case of traffic engineering, where small mistakes may result in injury or even death.

As seen in Figure 1 below, a system seemingly as simple as a four way intersection contains many possible issues and variables which must be accounted for.

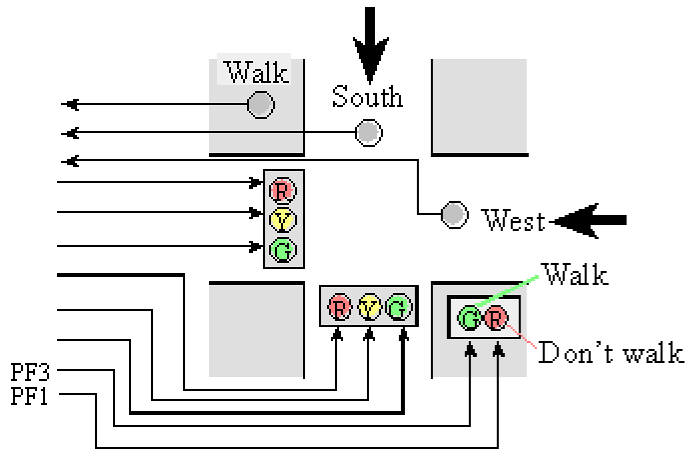

Figure 1: Diagram of 4-way intersection

The most likely problem to be encountered was improper interaction between the stoplights and pedestrian lights, which may cause a traffic accident. For the case of a traffic light simulation, it is crucial that green, red, and even yellow lights were synchronized in such a way that disallowed traffic to flow in two directions simultaneously. Initially timers and delays were thought to be the most logical way to implement this system, through the use of finite state machines (FSMs) were used in the end as they were much more efficient. As stated before, it's important for systems like these to be tested repeatedly to carefully avoid any future issues. This is also important to note because in the case of traffic systems much more complex than this simulation, finite state machines are most likely to be utilized because of this efficiency.

Before going into more detail about the project itself however, it is both important and interesting to understand the history of traffic engineering. You may skip to the bottom of page 8 - Design, to see the technical aspects of the project.

Theory

The research found in the following section contained in this document is not intended to cover every major technological breakthrough in traffic engineering technology, as that would be far beyond the scope of a paper this size. Instead the following research is focused on “transportation engineering,” a subset of traffic engineering that is less focused on the mathematical concepts such as speed limits, road shape and more focused on the application of technology in order to transport people and things safely, efficiently and quickly. The following is an attempt to inform readers on only the major traffic engineering related breakthroughs related to the project, containing the necessary information on how and why we have current traffic lights today.

2.0 Traffic Engineering History

Long before automobiles, traffic engineering was a crucial consideration for transport with horse-drawn carriages. It was known that in the mid 1800’s in London, streets would often crowd due to lack of traffic engineering systems [2]. Because of this, British railway manager John Knight, suggested adapting a railroad method for controlling traffic. This type of traffic signal was first implemented in 1868 near the Houses of Parliament in London. Unfortunately, as mentioned before, traffic engineering deals with dangerous situations. A month after the first iteration of this traffic signal was invented, a gas leak within the system had caused an explosion and badly injured a police officer

Because of the slow technological development during this time, traffic lights didn't regain interest until 1910 when Ernest Sirrine patented an automatic traffic signal (seen in Figure 2, below) in Chicago.

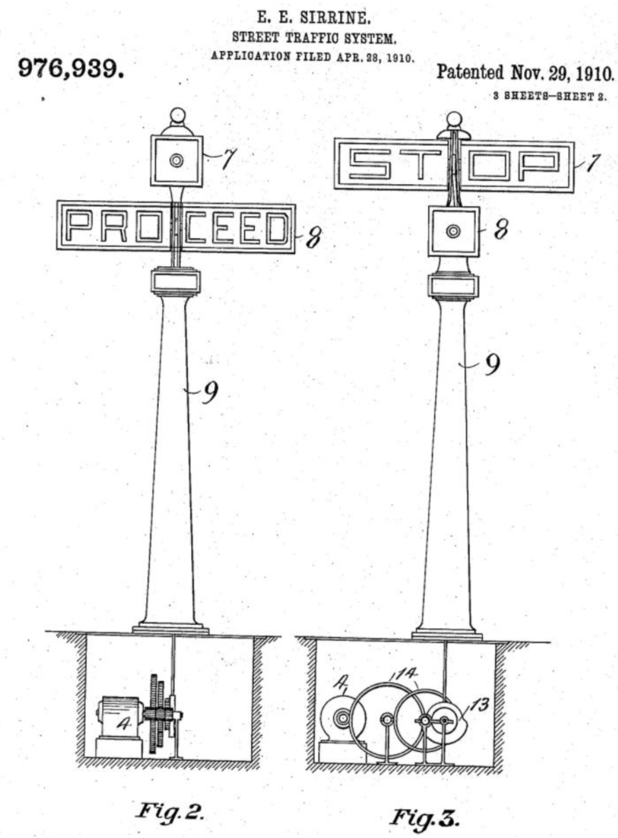

Figure 2: First automatically controlled traffic signal

This automatic traffic light used two non-illuminated sides reading "PROCEED" and "STOP." It was only two years later when Lester Farnsworth Wire invented the first illuminated traffic light, utilizing the familiar red and green lights, still used today. This device closely resembled a birdhouse with four sides. This contraption was centered in an intersection and needed to manually be switched by traffic / police officers. A depiction of this structure can be seen in Figure 3, on the next page.

Figure 3: First electric traffic device

It wasn’t until 1917 when William Ghiglieri introduced the modern automatic traffic light, equipped with both green and red lights. William Potts expanded upon this idea three years later by including a yellow "caution" light.

2.1 Present Day Traffic Engineering

The three colors utilized in 1917 are the same three used today, and while the idea of the modern traffic light hasn’t changed much within the last century, rapidly evolving technology has brought some improvements to the traffic light. Some examples of these improvements are the ability for automobiles to monitor real-time traffic situations automatically, the ability to perceive direction (GPS), and even the ability to read volume and density of traffic. There are currently efforts to enable traffic light systems to communicate with automobiles in order to send basic messages such as the state of upcoming traffic lights as well as recommendations for speed or different routes to take based on traffic.

2.2 Future of Traffic Engineering

Because the requirements regarding speed are only so high for traffic signals, the focus of traffic engineering has recently shifted to self-driving cars. In fact, some researchers have even stated that traffic lights will become obsolete and eventually non-existent because of self-aware automobile technology. Autonomous cars, which are quickly becoming reality, are capable of adjusting speed and direction by taking in information about the world around them. Using cameras as well as pulse-sensing technology, autonomous vehicles are even able to recognize pedestrians and bicyclists.

A second innovation which is now starting to be implemented is called Surtrac. Surtrac aims to combine technology from artificial intelligence as well as traffic engineering in order to optimize traffic [3]. This technology aims to accomplish this by picking up information regarding live current/traffic flow in a large area. It does this by constantly feeding in data and adjusting the timing of traffic systems according to things like time of day.

Design

When implementing this four way traffic intersection, both hardware and software components were designed and implemented. The following sections are used to outline both hardware and software design separately.

3.1 Hardware Design

Components of the circuit:

1x - Texas Instruments TM4C123 Microcontroller

1x - Breadboard to connect components

2x - LED's of one color (preferably green)

2x - LED's of a second color (preferably red)

2x- LED's of a third color (preferably yellow, white LEDs are used in this project)

3x - Push-button switches

9x - 300Ω resistors (resistors in depictions are much larger)

20x (approx.) - Wires, various lengths

The first components added to the circuit were push buttons, separated evenly throughout the board. The push buttons only fit longways across the board, so that pins 1 and 3 share a column. Pins 2 and 4 also share a column on the other side of the separation on the breadboard. This is shown in Figure 4, below.

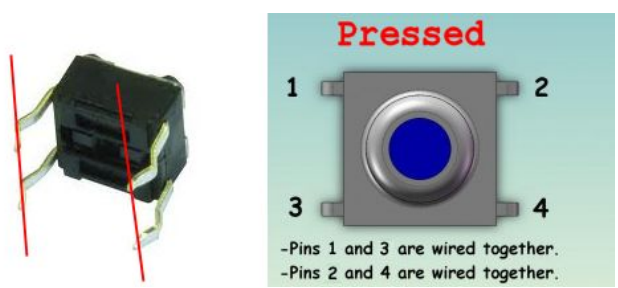

Figure 4: Push-button switch diagram

After the push buttons were secured into the breadboard, the LEDs were placed. The LEDs were placed in a similar fashion, with the longer, positive legs on the rows above the shorter, negative legs. It’s important to stay consistent with placement methods, as LEDs are just diodes. More detail is shown in Figure 5, below.

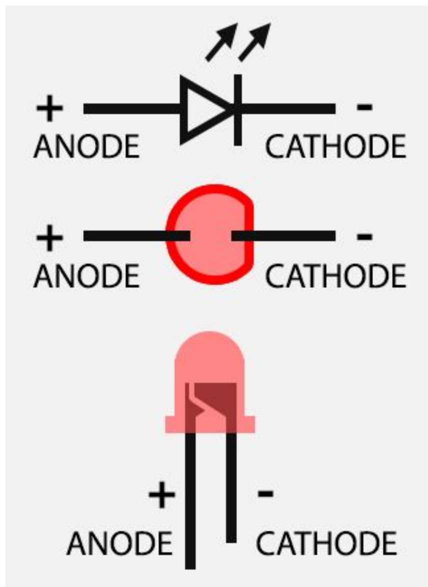

Figure 5: Diagram of LED

These LEDs were grouped in groups of one green, one white (lack of yellow LED), and one red. Another group of the same 3 colors were placed next to each other halfway down the board. Lastly, the 900Ω resistors were added by connecting the negative leg of each of the 6 LEDs to ground, as well as one end of each of the push-button switches. A diagram of these connections can be seen in Figure 6, on the next page.

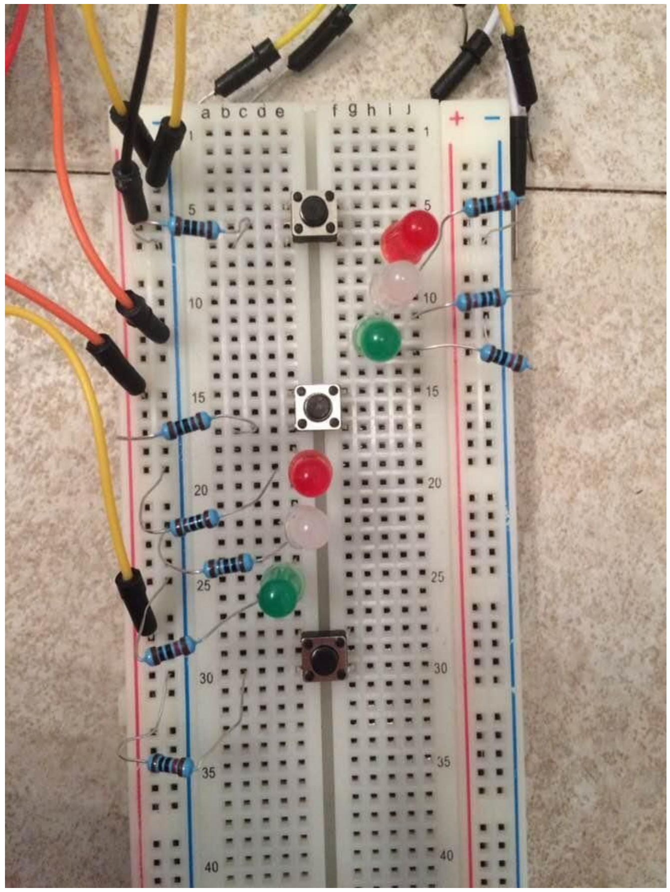

Figure 6: Hardware design of breadboard (before wires)

After the small, physical components were placed into the breadboard, the wires were placed to ground each resistor.

Next, the microcontroller board must be configured properly. The software, which ran signals to and from the push button-switch and LED configuration was connected using ports B, E and F. This can be seen in Figure 7, on the next page.

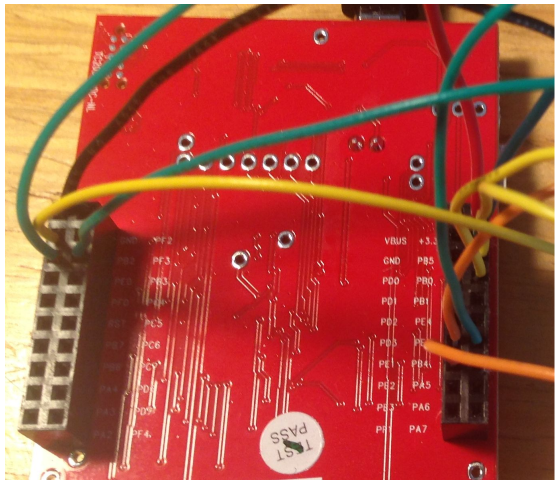

Figure 7: Close-up view of Microcontroller board

After the input output (IO) ports were connected to the physical hardware properly, the simulation was ready to be compiled and run. Alternate views of the hardware can be seen in Appendix A - Detailed View of Hardware (Pg 17).

3.2 Software Design

Most of the code is rather simplistic, although the use of a finite state machine, mentioned in the introduction, was particularly difficult to implement..

Outside of the main method, ports B E and F are initialized to output to the car (south and west) LEDs, output to the pedestrian lights, and to be inputs from the push-button switch sensors respectively. SysTick methods were written to establish a delay timer. Inside of the main function, a while loop was utilized to update the state of the car and pedestrian lights. Inside this loop is also a line responsible for transitioning from one state to the next state.

Aside from the while loop, a finite state machine was implemented within the main method. The finite state machine was designed in such a way that at all times, only one of the three sets (west, south, pedestrian) of lights was in a green state. Along with this, a light may only turn green once the old green light has turned to red (or transitioned from yellow then to red, in the case of the cars). The finite state machine also works in a way that there is never a green and yellow light on at the same time, similar to in real life scenarios. Lastly, the state machine establishes that if all inputs are recognized (south and west cars arrive alongside a pedestrian), each light takes turns transitioning to green. A compact illustration of this finite state machine can be seen in Figure 8, below.

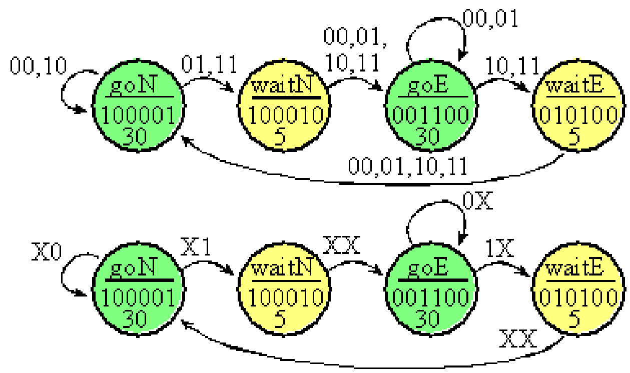

Figure 8: Compact View of FSM

To cement the workings of the finite state machine, Figure 9, on the next page, shows the states and transitions. Below that, in Figure 10, the code to implement the FSM is shown for comparison.

#	Name	Traffic LEDs	Pedestrian LED	Time	If (in=nothing)	If (in=west car)	If (in=south car)	If (in=west & south car)	if (in=pedestrian)	if (in=west & pedestrian)	if (in=south & pedestrian)	if (in=west & south & pedestrian)
					0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0	WestGreen	0x0C	0x02	20	0	0	1	1	1	1	1	1
1	WestYellow	0x14	0x02	30	1	0	2	2	4	4	2	2
2	SouthGreen	0x21	0x02	20	2	3	2	3	3	3	3	3
3	SouthYellow	0x22	0x02	30	3	0	2	0	4	0	4	4
4	WalkGreen	0x24	0x08	20	4	5	5	5	4	5	5	5
5	WalkFlash[Off1]	0x24	0x00	5	4	6	6	6	4	6	6	6
6	WalkFlash[On1]	0x24	0x02	5	4	7	7	7	4	7	7	7
7	WalkFlash[Off2]	0x24	0x00	5	4	8	8	8	4	8	8	8
8	WalkFlash[On2]	0x24	0x02	5	4	9	9	9	4	9	9	9
9	WalkFlash[Off3]	0x24	0x00	5	4	10	10	10	4	10	10	10
10	WalkFlash[On3]	0x24	0x02	5	5	0	2	0	4	0	2	0
West Sensor = PE0					Green	Yellow	Red					
South Sensor = PE1				West	PB3	PB4	PB5					
Walk Sensor = PE2				South	PB0	PB1	PB2					
				Walk	PF3	NA	PF1					

Figure 9: Table to show FSM transitions

```
SType FSM[11]={
  // States of Finite State Machine
  {0x0C,0x02,20,{0,0,1,1,1,1,1,1}},
  {0x14,0x02,30,{1,0,2,2,4,4,2,2}},
  {0x21,0x02,20,{2,3,2,3,3,3,3,3}},
  {0x22,0x02,30,{3,0,2,0,4,0,4,4}},
  {0x24,0x08,20,{4,5,5,5,4,5,5,5}},
  {0x24,0x00,5,{4,6,6,6,4,6,6,6}},
  {0x24,0x02,5,{4,7,7,7,4,7,7,7}},
  {0x24,0x00,5,{4,8,8,8,4,8,8,8}},
  {0x24,0x02,5,{4,9,9,9,4,9,9,9}},
  {0x24,0x00,5,{4,10,10,10,4,10,10,10}},
  {0x24,0x02,5,{5,0,2,0,4,0,2,0}}
};
```

Figure 10: Corresponding code to implement FSM in Figure 9

Results & Concluding Thoughts

Overall, this project is an active demonstration of how a 4-way intersection may operate, proving the complexity of traffic / transportation engineering. Equipped with consideration for both two traffic lanes, as well as one dedicated to pedestrians, this demo is capable of simulating a common everyday traffic scenario. This project has also described how far technology has come in the field of traffic engineering by covering the past inventions, current innovations, and future possibilities of this technology. It is evident that both hardware and software are used in harmony in order to build and operate complex machines, such as the 10 state finite state machine found in the “software design” section above. With the use of many figures, this project is replicable for anyone to build, and perhaps expand upon.

Bibliography

1. Dictionary.com (2017, Dec 18), Definition of Traffic Engineering. Dictionary.com [Online] Available: http://www.dictionary.com/browse/traffic-engineering
2. Rachel Ross (2017, Dec 18), Who Invented the Traffic Light? LiveScience [Online] Available: https://www.livescience.com/57231-who-invented-the-traffic-light.html
3. Surtrac (2017, Dec 18), Surtrac Technology. Surtrac [Online] Available: https://www.surtrac.net/technology/

Appendices

Appendix A - Detailed View of Hardware

Below are numerous pictures of the hardware setup for replication purposes.

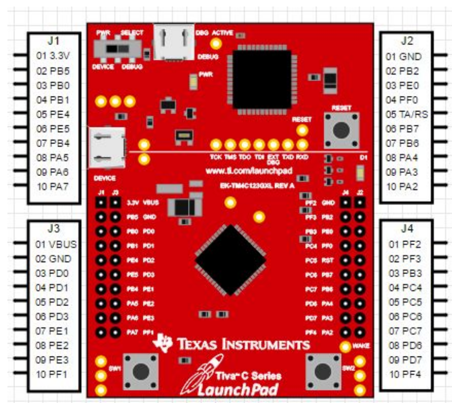

Figure A1: Texas Instruments Launchpad Microcontroller Pinout

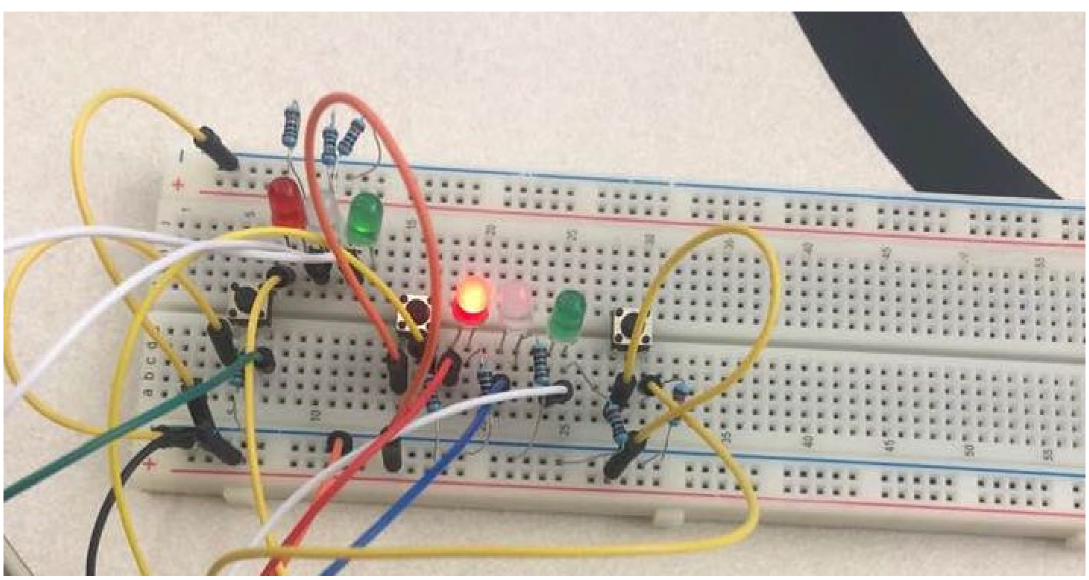
Figure A2: Demonstrated Circuit (wide perspective)

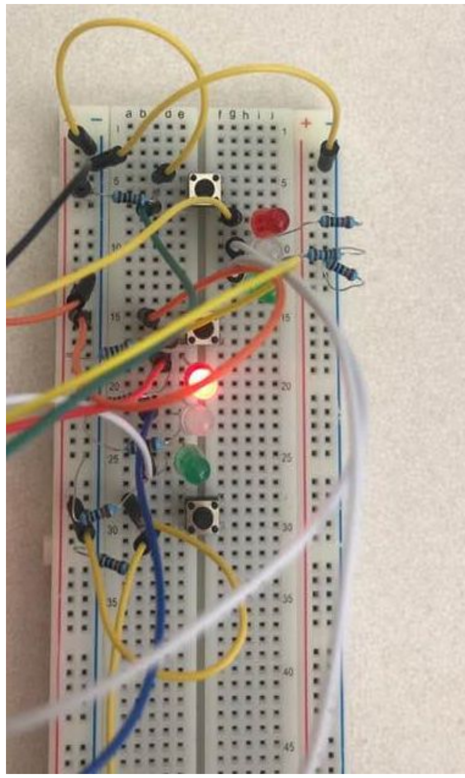

Figure A2: Demonstrated Circuit (narrow perspective)

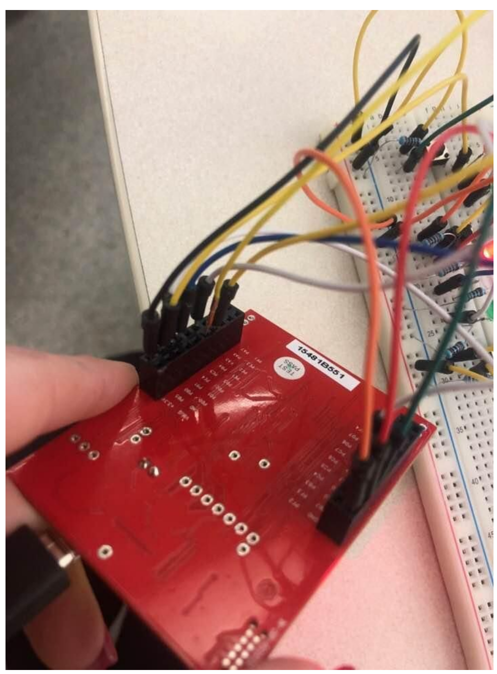

Figure A3: Board Connections (view 1)

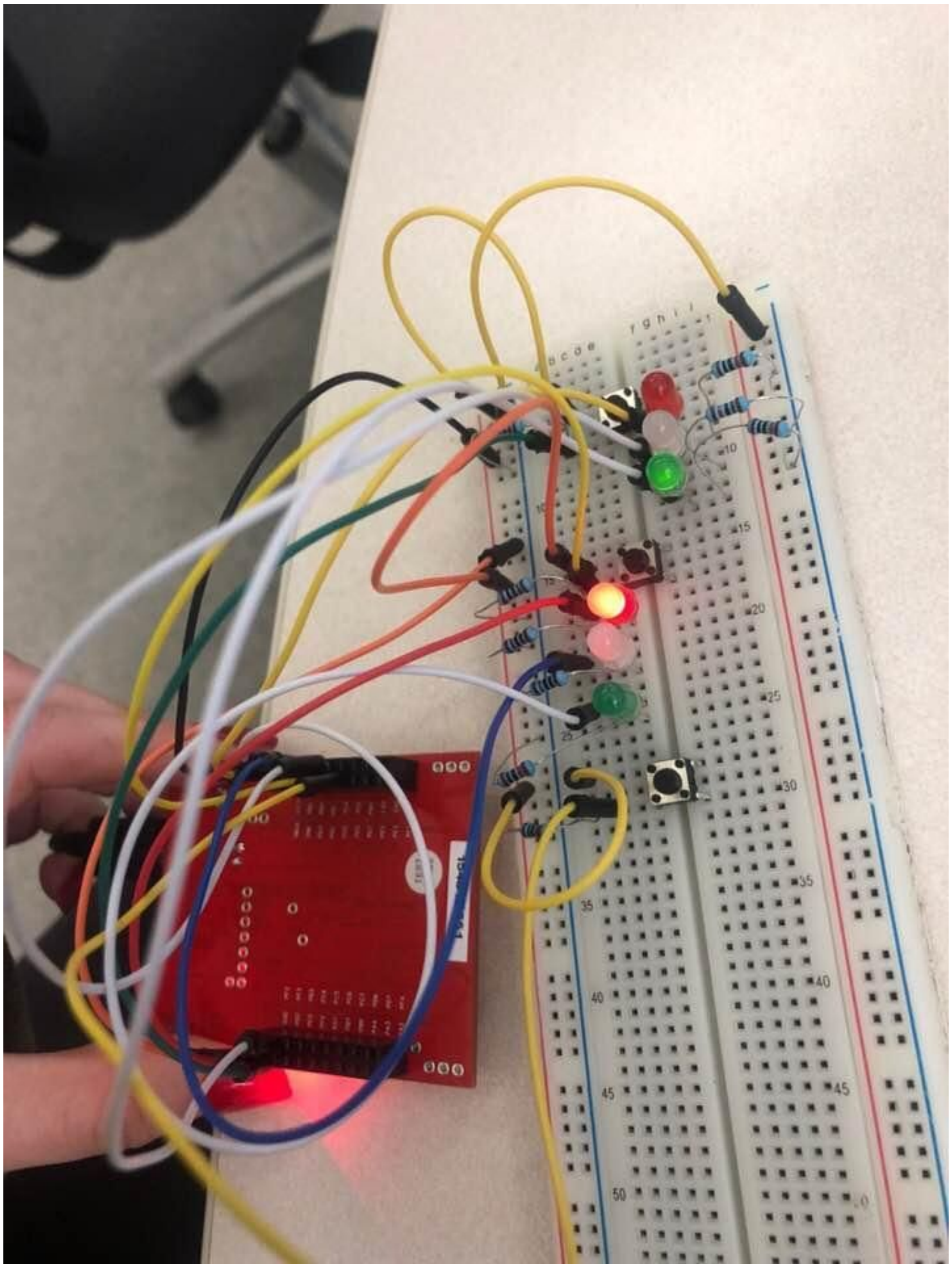

Figure A4: Demonstrated Circuit with Board

Appendix B - Code

Similarly to Appendix A - Detailed View of Hardware, the entirety of the code used in the main TrafficLight.c file can be found below.

```
// George Dagis, [PARTNER NAME OMITTED]
// 12-02-17

// Program simulated 4-way intersection
// Utilizes 3 inputs, taken from push-button switches
//
// Southwards green,yellow,red light connected to PB0,PB1,PB2
// Westwards green,yellow,red light connected to PB3,PB4,PB5
// Walk,stop light connected to PF3,PF1
// Detectors for westwards,southwards,pedestrian connected to PE0,PE1,PE2

// ***** 1. Pre-processor Directives Section *****
#include "TExaS.h"
#include "tm4c123gh6pm.h"

// ***** 2. Global Declarations Section *****
#define NVIC_ST_CTRL_R          (*((volatile unsigned long *) 0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile unsigned long *) 0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile unsigned long *) 0xE000E018))
#define TRAFFIC_LIGHTS          (*((volatile unsigned long *) 0x400050FC))
#define PEDESTRIAN_LIGHTS       (*((volatile unsigned long *) 0x40025028))
#define SENSORS                 (*((volatile unsigned long *) 0x4002401C))

// FUNCTION PROTOTYPES: Each subroutine defined
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

// ***** 3. Subroutines Section *****
void ports_Init(void){

	// Port B, E, & Finitialization
	unsigned long volatile delay;
	SYSCTL_RCGC2_R |= 0x32; // Activate clock for Port B, E, & F
	 delay = SYSCTL_RCGC2_R; // Delay for clock to start

	// Port B initialization
	GPIO_PORTB_LOCK_R = 0x4C4F434B;      // Unlock port
	GPIO_PORTB_CR_R = 0x3F;              // Allow changes to PB5-0
```

```
	GPIO_PORTB_PCTL_R = 0x00000000;        // Clear PCTL
	GPIO_PORTB_AMSEL_R &= ~0x3F;          // Disable analog on PB5-0
	GPIO_PORTB_AFSEL_R &= ~0x3F;          // Disable alt funct on PB5-0
	GPIO_PORTB_DEN_R |= 0x3F;         // Enable digital I/O on PB5-0
	GPIO_PORTB_DIR_R |= 0x3F;         // PB5-0 outputs

	// Port E initialization
	GPIO_PORTE_LOCK_R = 0x4C4F434B;     // Unlock port
	GPIO_PORTE_CR_R = 0x07;                      // Allow changes to PE2-0
	GPIO_PORTE_PCTL_R = 0x00000000;        // Clear PCTL
	GPIO_PORTE_AMSEL_R &= ~0x07;           // Disable analog on PE2-0
	GPIO_PORTE_AFSEL_R &= ~0x07;           // Disable alt funct on PE2-0
	GPIO_PORTE_PUR_R &= ~0x07;              // Disable pull-up on PE2-0
	GPIO_PORTE_DEN_R |= 0x07;                 // Enable digital I/O on PE2-0
	GPIO_PORTE_DIR_R &= ~0x07;               // PE2-0 inputs

	// Port F initialization
	GPIO_PORTF_LOCK_R = 0x4C4F434B;     // Unlock port
	GPIO_PORTF_CR_R = 0x0A;                     // Allow changes to PF1 & PF3
	GPIO_PORTF_PCTL_R = 0x00000000;       // Clear PCTL
	GPIO_PORTF_AMSEL_R &= ~0x0A;         // Disable analog on PF1 & PF3
	GPIO_PORTF_AFSEL_R &= ~0x0A;         // Disable alternate function on PF1 & PF3
	GPIO_PORTF_DEN_R |= 0x0A;                // Enable digital I/O on PF1 & PF3
	GPIO_PORTF_DIR_R |= 0x0A;                 // PF1 & PF3 outputs
}

void SysTick_Init(void) {
	// Systick initialization
	NVIC_ST_CTRL_R = 0;                         // Disable SysTick during setup
	NVIC_ST_CTRL_R = 0x00000005;      // Enable SysTick with core clock
}

void SysTick_Wait10ms() {
	// Delay for 10ms
	NVIC_ST_RELOAD_R = 8000000 - 1;                      // Wait (80Mhz PLL)
	NVIC_ST_CURRENT_R = 0;                                    // Value written to CURRENT is cleared
	while((NVIC_ST_CTRL_R&0x00010000)==0)   {    // Wait for count flag
	}
}


void SysTick_Wait(unsigned long delay) {
	// Delay
	unsigned long i;
	for(i=0; i < delay; i++)
		SysTick_Wait10ms();
}
```

```c
typedef struct Stype {

	// Structure of a single state in the Finite State Machine
	unsigned long TrafficOut;		// Output for car lights (Port B)
	unsigned long WalkOut;		// Output for pedestrian lights (Port F)
	unsigned long Time;			// Delay time
	unsigned long Next[8];		// Next state
} SType;

int main(void){

	unsigned long S = 0; // Current state
	SType FSM[11]={
		// States of Finite State Machine
		{0x0C,0x02,20,{0,0,1,1,1,1,1,1}},
		{0x14,0x02,30,{1,0,2,2,4,4,2,2}},
		{0x21,0x02,20,{2,3,2,3,3,3,3,3}},
		{0x22,0x02,30,{3,0,2,0,4,0,4,4}},
		{0x24,0x08,20,{4,5,5,5,4,5,5,5}},
		{0x24,0x00,5,{4,6,6,6,4,6,6,6}},
		{0x24,0x02,5,{4,7,7,7,4,7,7,7}},
		{0x24,0x00,5,{4,8,8,8,4,8,8,8}},
		{0x24,0x02,5,{4,9,9,9,4,9,9,9}},
		{0x24,0x00,5,{4,10,10,10,4,10,10,10}},
		{0x24,0x02,5,{5,0,2,0,4,0,2,0}}
	};

	// Initialization
	ports_Init();		// Initialize ports B, E, & F
	SysTick_Init(); // Initialize systick
	EnableInterrupts();

	// Loop through FSM
	while(1) {
		TRAFFIC_LIGHTS = FSM[S].TrafficOut;  // Set car lights
		PEDESTRIAN_LIGHTS = FSM[S].WalkOut;  // Set pedestrian lights
		SysTick_Wait(FSM[S].Time); // Delay
		 S = FSM[S].Next[SENSORS];  // Next state
	 }
} //end main
```