CECS 346 Spring 2018 Project #1

Traffic Light Controller

Jonic Mecija

March 2, 2018

This project implements a finite state machine
to control a traffic intersection. The hardware used replicates
an intersection with two traffic lights and one pedestrian signal.

Introduction

To create the state logic of a traffic controller, a Moore finite state machine is used to control the output of the lights, timing, and the next state. The outputs of the finite state machine mimic the logic two traffic lights at an intersection. This is shown through hardware.

To replicate the intersection lights, eight LEDs were connected to the output pins of the board. Six LEDs are for the two traffic lights, and the other two LEDs are signals for the pedestrian. To replicate the sensor inputs for the cars and pedestrians, we used three buttons. Two buttons are for the car sensors. One in the North/South street, and the other button is for East/West street. The last button is the pedestrian sensor.

The main goal of this project is to make sure cars can pass through without any accidents. This intersection also has two lights signaling when pedestrians can cross the intersection safely. To ensure the safety of everyone in the intersection, we must make sure the logic of the finite state machine is correct.

Operation

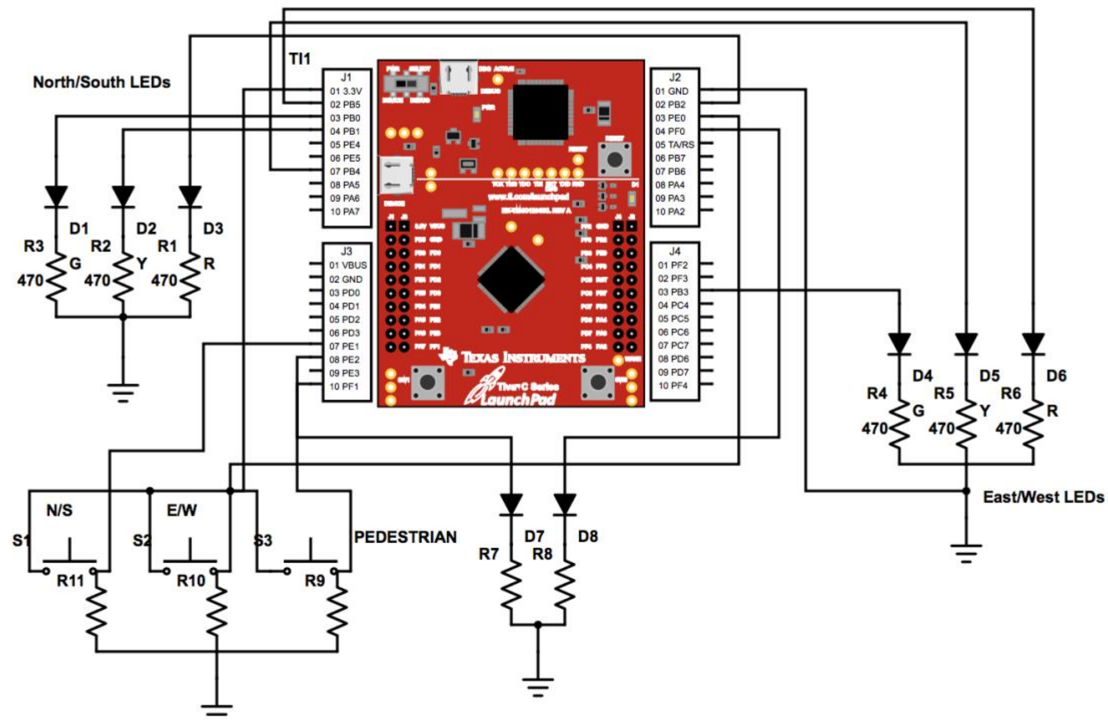
There are only three ways to put in inputs, and they all come from three buttons. The first button is the pedestrian sensor input. When pressed, it signals the controller to let pedestrians cross safely. The same goes for the cars in the North/South street and East/West street. The second button senses the cars from N/S street, and the third button senses cars from the E/W street. The initial state of the traffic light controller is green for the North/South street and red for the East/West street.

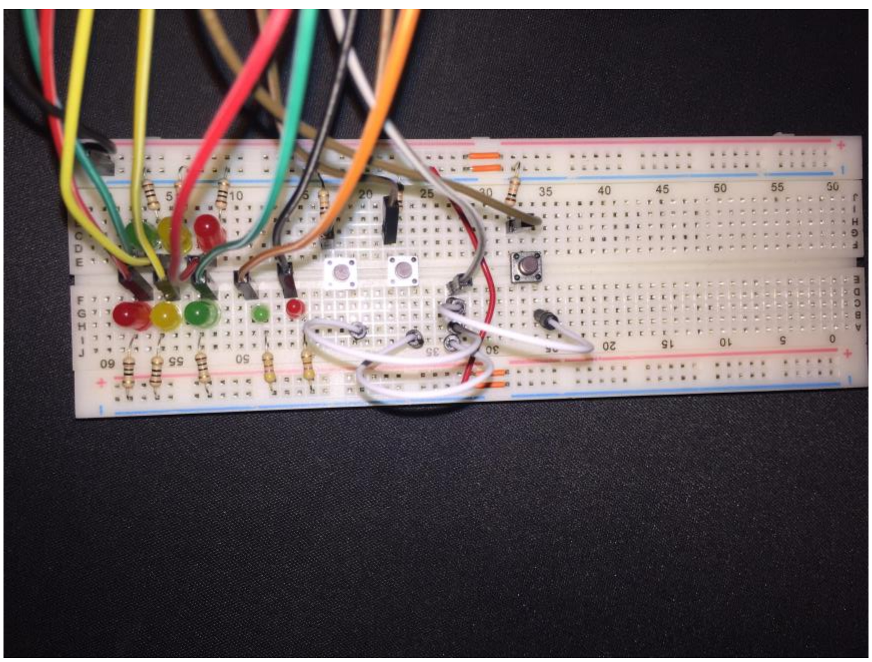
With all of this in mind, the traffic controller has an order of priority addressing multiple inputs at the same time. Priority to cross first goes as follows: pedestrian, N/S cars, and E/W cars. Having a priority list creates a system of order that allows cars and pedestrian to cross safely when sensors are triggered simultaneously.

Theory

The main software concept behind the design of this traffic light controller is the use of a finite state machine and indexed data structures. To create the logic of the two-way traffic light and a pedestrian signal, I used a chart to document the next states according to the inputs and current state. Based on the button inputs, it was possible to go to eight different next states. With this in mind, I created an array with eight elements. The elements were the next states which were determined by input and current state. The finite state machine that I created had four elements: output of port B, output of port F, timing of the LEDs, and the next state. Since next state was a single parameter within the state machine, I was able to place a single array of eight states which held the logic for the next states. This use of indexed data structures is carried over from lecture in class.

Hardware





Video demo: https://www.youtube.com/watch?v=RBQoOyfY1ys

Software

The main modules within my program are related to the finite state machine. The first module defines the elements within the state machine. The elements defined are outputs from port B, outputs from port F, timing, and the next state array. The next module defines the next state logic that comes directly from our state chart. After those two modules, the main is next. The main calls all the port initializations, systick initialization, sets the current state, and sets and reads all the parameters of the finite state machine.

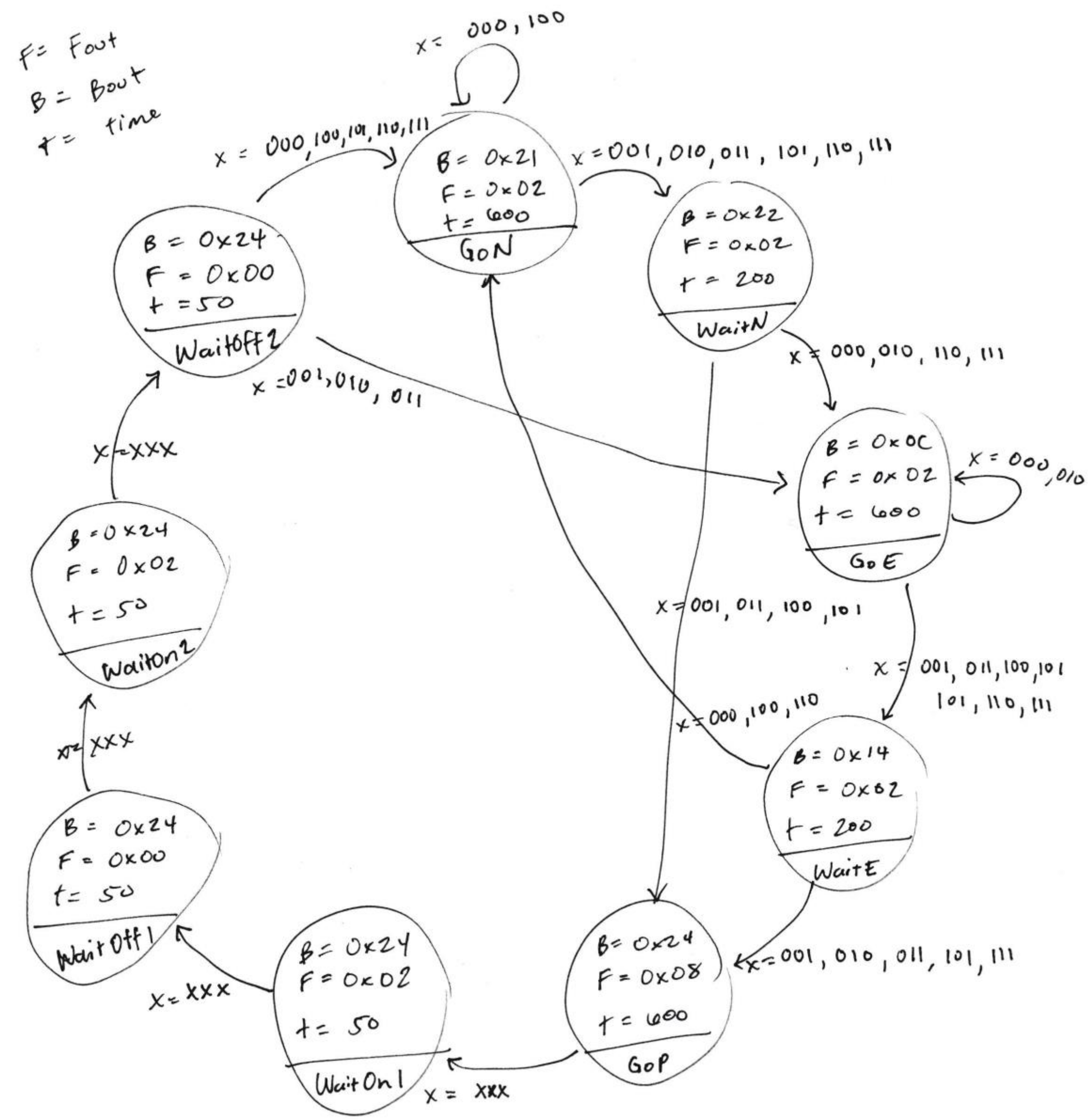
State	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
GoN	goN	waitN	waitN	waitN	goN	waitN	waitN	waitN
WaitN	goE	goP	goE	goP	goP	goP	goE	goE
GoE	goE	waitE	goE	waitE	waitE	waitE	waitE	waitE
WaitE	goN	goP	goP	goP	goN	goP	goN	goP
GoP	waitOn1	waitOn1	waitOn1	waitOn1	waitOn1	waitOn1	waitOn1	waitOn1
WaitPOn1	waitOff1	waitOff1	waitOff1	waitOff1	waitOff1	waitOff1	waitOff1	waitOff1
WaitPOff1	waitOn2	waitOn2	waitOn2	waitOn2	waitOn2	waitOn2	waitOn2	waitOn2
WaitPOn2	waitOff2	waitOff2	waitOff2	waitOff2	waitOff2	waitOff2	waitOff2	waitOff2
WaitPOff2	goN	goE	goE	goE	goN	goN	goN	goN

Input: N, E, P

Legend bit2 bit1 bit0

X = North, East, Pedestrian

STATE ~~DIAGRAM~~ DIAGRAM



Conclusion

Overall this project was a great learning experience in terms of implementing a finite state machine with an indexed data structure. Most of my learning was within the design of the software. The use of a chart to help diagram the logic of the state machine was really helpful. The main problem was figuring out the next states of the chart. There was a problem in lecture where the next states were filled out incorrectly. This led me to think that the wrong states were correct, but eventually I corrected it and tested it out. After testing out my logic on the software, I was certain that there were erroneous states from lecture.

Surprisingly once I had the state table chart filled out and I programmed the FSM, the simulation worked for the most part with some minor issues displaying the correct LEDs. The in class lectures and lecture slides provided great examples to follow that added to my learning.

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

// ***** 2. Global Declarations Section *****

// FUNCTION PROTOTYPES: Each subroutine defined
void Init_PortB(void);                    // Port B initialzation
void Init_PortE(void);                    // Port E initialzation
void Init_PortF(void);               // Port F initialization
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
//void Delay(unsigned int x);   // interrupt from Lab 2


// ***** 3. Subroutines Section *****

#define LIGHT   (*((volatile unsigned long *)0x400053FC))
#define SENSOR  (*((volatile unsigned long *)0x4002401C))
#define P_LIGHT (*((volatile unsigned long *)0x40025028))

//
struct TrafficLight
{
	unsigned long B_out;    // PB5-0 outputs ( traffic lights )
	unsigned long F_out;    // PF3 and PF1 outputs ( pedestrian lights )
	unsigned long time;                // Time delay for lights
	unsigned char next[8];  // Next state array
};

typedef const struct TrafficLight STyp;

// define FSM states
#define goN       0
#define waitN      1
#define goE       2
#define waitE      3
#define goP       4
#define waitPOn_1  5
#define waitPOff_1 6
#define waitPOn_2  7
#define waitPOff_2 8
```

```
// Finite State Machine definition

STyp FSM[9] = {

// B_out, F_out, time, { next state array [8] }

        // goN
        { 0x21,  0x02,  600, { goN, waitN, waitN, waitN, goN, waitN, waitN, waitN } },

        // waitN
        { 0x22,  0x02,  200, { goE, goP, goE, goP, goP, goP, goE, goE } },

        // goE
        { 0x0C,  0x02,  600, { goE, waitE, goE, waitE, waitE, waitE, waitE, waitE } },

        // waitE
        { 0x14,  0x02,  200, { goN, goP, goP, goP, goN, goP, goN, goP } },

        // goP
        { 0x24,  0x08,  600, { waitPOn_1, waitPOn_1, waitPOn_1, waitPOn_1, waitPOn_1,
waitPOn_1, waitPOn_1 , waitPOn_1 } },

        // waitPOn1
        { 0x24,  0x02,  50,  { waitPOff_1, waitPOff_1, waitPOff_1, waitPOff_1, waitPOff_1,
waitPOff_1, waitPOff_1, waitPOff_1 } },

        // waitPOff1
        { 0x24,  0x00,  50,  { waitPOn_2, waitPOn_2, waitPOn_2, waitPOn_2, waitPOn_2,
waitPOn_2, waitPOn_2, waitPOn_2 } },

        // waitPOn2
        { 0x24,  0x02,  50,  { waitPOff_2, waitPOff_2, waitPOff_2, waitPOff_2, waitPOff_2,
waitPOff_2, waitPOff_2, waitPOff_2 } },

        // waitPOff2
        { 0x24,  0x00,  50,  { goN, goE, goE, goE, goN, goN, goN, goN } },
};

// main function
int main(void)
{
        // variable declaration
        unsigned char current_state;
        unsigned int input;
```

```
	// port initialization
	Init_PortB();
	Init_PortE();
	Init_PortF();
	// timer initialization
  SysTick_Init();

	// set the initial state
	current_state = goN;


// super loop
  while(1)
	{
		LIGHT = FSM[current_state].B_out;			// set LEDs ( cars )
		P_LIGHT = FSM[current_state].F_out;			// set LEDs ( pedestrians )
		SysTick_Wait10ms(FSM[current_state].time);  // set timing
		input = SENSOR;
									// read sensors
		current_state = FSM[current_state].next[input];  // set current state
	}

}

// port B used for output LEDs ( 6 LEDs )
void Init_PortB(void)
{
	unsigned int delay;
  SYSCTL_RCGC2_R    |= 0x00000002;   // 1) B clock
  delay = SYSCTL_RCGC2_R;           // delay
//GPIO_PORTB_CR_R    = 0x07;        // allow changes to PB2-0
  GPIO_PORTB_AMSEL_R = 0x00;        // 3) disable analog function
  GPIO_PORTB_PCTL_R  = 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTB_DIR_R  |= 0x3F;        // 5) PB5-PB0 are outputs ( 1's mean output )
  GPIO_PORTB_AFSEL_R = 0x00;        // 6) no alterna=te function
//GPIO_PORTB_PUR_R   = 0x00;        // enable pullup resistors
  GPIO_PORTB_DEN_R  |= 0x3F;        // 7) enable digital I/O on PB5-PB0
}

// port E used for input sensors ( 3 buttons )
void Init_PortE(void)
{
	unsigned int delay;
```

```
  SYSCTL_RCGC2_R    |= 0x00000010;   // 1) E clock
  delay = SYSCTL_RCGC2_R;           // delay
//GPIO_PORTE_CR_R    = 0x03;        // allow changes to PE-0
  GPIO_PORTE_AMSEL_R = 0x00;        // 3) disable analog function
  GPIO_PORTE_PCTL_R  = 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTE_DIR_R   = 0x00;        // 5) PE2-0 are inputs ( 0's mean input )
  GPIO_PORTE_AFSEL_R = 0x00;        // 6) no alternate function
//GPIO_PORTE_PUR_R   = 0x00;        // enable pullup resistors
  GPIO_PORTE_DEN_R  |= 0x07;        // 7) enable digital I/O on PFE-PF0
}

// port F used for pedestrian LEDs ( 2 LEDs )
void Init_PortF(void)
{
  unsigned int delay;
  SYSCTL_RCGC2_R    |= 0x00000020;   // 1) F clock
  delay = SYSCTL_RCGC2_R;           // delay
	GPIO_PORTF_LOCK_R  = 0x4C4F434B;   // 2) unlock GPIO Port F
//GPIO_PORTE_CR_R    = 0x03;        // allow changes to PE-0
  GPIO_PORTF_AMSEL_R = 0x00;        // 3) disable analog function
  GPIO_PORTF_PCTL_R  = 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTF_DIR_R  |= 0x0A;        // 5) PF1 and PF3 are outputs ( 1's are outputs )
  GPIO_PORTF_AFSEL_R = 0x00;        // 6) no alternate function
//GPIO_PORTE_PUR_R   = 0x00;        // enable pullup resistors
  GPIO_PORTF_DEN_R  |= 0x0A;        // 7) enable digital I/O on PF3 and PF1
}

/*
// timer used in lab 2
void Delay(unsigned int x)
{
	unsigned long volatile time;
  time = 727240*x/91;

	// passes in paramater x
	// bigger X = more delay, smaller X = less delay
	// x = 200 gives a 0.1 sec delay

  while(time)
	{
		time--;
  }
}
*/
```