Agoston Walter

Professor John Board

ECE 350 Digital Design

April 23, 2020

FSM IMPLEMENTATION OF TRAFFIC LIGHT CONTROLLER

Overall Project Design and Specifications

The design problem chosen was to create a traffic light system for the intersection of a two-lane highway running in the North South direction with a smaller, two-lane road running in the East West direction. This is shown in Figure 1.

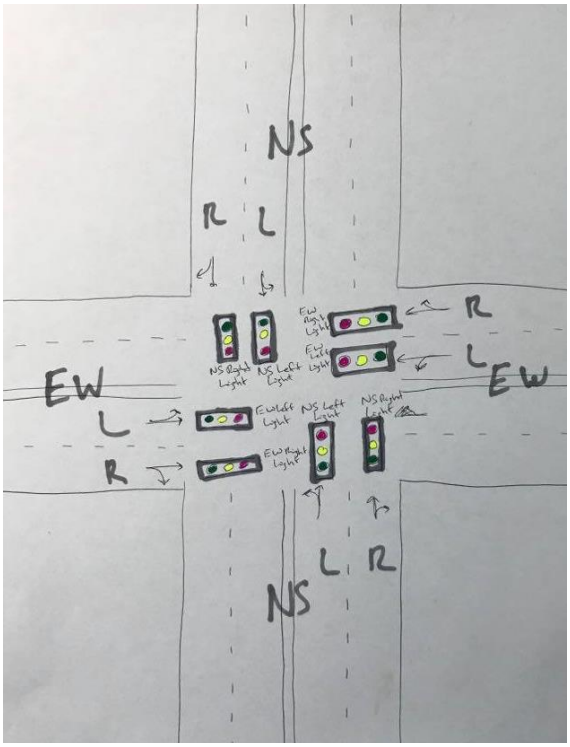

Figure 1: Intersection Diagram

To simplify the design of the traffic light, I chose to make the car presence sensor input HIGH if there was a car in either of the identical lanes opposite each other. For example, the car presence sensor input would be triggered HIGH if there was a car in either the right lane running from North to South, or a car in the right lane running from South to North, or both. This simplified the design problem to a four-bit input problem, where 0 can represent no cars, and 1 represents a car in one of the two lanes for each highway. To further simplify the design problem, I chose to have the traffic light control be the same for identical lanes opposite each other. For example, the traffic signal for the highway right lane running South is the same as the traffic signal for the highway right lane running North. This simplified the design problem produce an eight-bit output; each of the four traffic lights having four possible states (red (00), yellow (01), solid green (proceed with caution) (10), green arrow (protected turn) (11)). The four possible output states of each traffic light can be expressed with two bits of output, and with four traffic lights, the output totals eight bits.

To solve the design problem, I chose to use a Moore Finite State Machine (FSM), which is shown in the Appendix, Figure 2. The Moore FSM has eight states, operates on a four-bit input, and has an eight-bit output. The four-bit input is organized as follows: East West Left Lane, East West Right Lane, North South Left Lane, North South Right Lane. The eight-bit output, with each traffic light being controlled by a pair of bits, is organized as follows: East West Left Lane, East West Right Lane, North South Left Lane, North South Right Lane. To simplify the state equations, I chose to encode the state as the output. For example, state A would be encoded as

0000 1010. This can be seen in the Finite State Machine Truth Table, shown in the Appendix Figure 3, which I used to calculate the state equations, also shown in Appendix Figure 3.

To verify the state equations worked, I implemented the FSM in Logisim using DFFs. I went through the truth table painstakingly, verifying that each transition did indeed occur, and where it did not, I found the mistake in the equation and corrected it. Screenshots of my Logisim circuit file are shown in Appendix Figure 4.

Having confirmed that the state equations for my FSM are true with Logisim, I wanted to create a physical model of my traffic light controller in action. To bring my traffic controller to life, I used an Arduino Uno as my controller, with four push button inputs and 12 LED outputs, with three LEDs (red, yellow, green) used for each traffic controller. The LED outputs were controlled by an LED Driver, the Adafruit TLC59711, which was operated through an SPI interface. The push buttons are implemented with pull down resistors and pressing them registers as an input bit of 1, otherwise the input is 0. The output bits are expressed in the following manner: 00 turns on the red LED, 01 turns on the yellow LED, 10 turns on the green LED, and 11 turns on the green and yellow LED, indicating that lane has a ‘green arrow’ (protected turn). The wiring diagram of my implementation is shown in Appendix Figure 5. A picture of the working system is shown in Appendix Figure 6.

To code the FSM, I used the next state equations from Figure 3 to calculate each of the next state bits for each iteration of the loop based on the combination of input and current state bits. I then set the current state bits to the calculated next state bits. Using the new current state bits, I set the output LEDs to express the output tied to the new current state. I then delayed for 5 seconds, before iterating through the loop again. To iterate through the states with this simulation, the user must press down on the combination of input bits they wish to input into the system using the push buttons. The Arduino code is included in the Appendix Figure 7.

The Arduino code was written for five second transition times. My FSM only stays in an output state if there are no other cars waiting in front of a red light. If it detects cars waiting at a red light, it will change to a light that will allow them to go, in the precedence order of perpendicular road, then left turning lane. For example, if the highway running in the North South direction has both lights green and the road running in the East West direction has both lights red, and cars arrive at both the highway and road, then car presence sensors will toggle ‘HIGH’ for both lanes of the highway and the road. Then the lights for the highway running in the North South direction will change to yellow, and then to red, and the left lanes of the road running in the East West direction will have a protected turn for five seconds, and then both lanes will be solid green for five seconds, and then they will turn yellow for five seconds, and then red, and the same thing will happen for the highway running in the North South direction.

The highlight of my design is that it improves the existing FSM I see in traffic lights because it accounts for cars performing legal and illegal maneuvers to leave the intersection after having triggered the car presence sensor. In current traffic light implementations this feature is not available. If a car is detected waiting in front of a red light, the other road’s lanes’ lights will turn

from yellow to red, and the waiting car's light will turn green, even if the car has left the intersection.

Before each light transition, my FSM verifies that there is a need for the light to change (there is a car waiting). For example, if the highway running in the North South direction has both lanes green, and a car arrives in the right lane of the road running in the East West direction, which has both lines red, the two lanes of the highway running in the North South direction would turn to yellow. In the case where the car in the right lane of the road takes a legal right on red at the light before the lights for the highway running in the North South direction change from yellow to red, causing there to no longer be any cars waiting in the lanes of the road running in the East West direction, then my FSM would recognize this, and change the lights for the highway running in the North South direction from yellow back to green. This autocorrection feature is present for both lanes, left and right, and both directions, from North South direction and East West direction. I believe it is important to implement because although a car can be detected waiting in front of a red light, there are instances where the car can leave the intersection legally and illegally, and these should be accounted for in the best way so as not to disrupt the flow of traffic.

Challenges you faced and how you overcame them

I first tried created the design problem to have eight independent traffic lights and eight independent inputs, but I soon realized once I began trying to sketch out the FSM State Transition Diagram that an eight-bit input, sixteen-bit output FSM would be too complex to create a diagram and robust equations within the given time. I simplified my design problem so that a four-bit input, eight-bit output FSM solution would suffice, making it possible to solve the system and implement it in Logisim and Arduino within the timeframe.

After I found the next state equations, I tested my FSM using Logisim. I found that my equations had several errors, which lead to erroneous transitions. To correct the equations, I examined the equation for the faulty bit in the erroneous transition and fixed my hand calculations. This was a painstaking way to go about it, but I was able to correct all the errors in a systematic way.

When I first implemented the push button switches, I had one side in contact with power and the other side in contact with the Arduino pin. When I tried to use them, I got false positives, with the Arduino pin registering an input although there was none. I realized my error: I had the Arduino pin floating! To correct it, I connected the Arduino pin to a 1M ohm resistor which led to ground. This way, the pin would be pulled down to ground unless the push button is pressed.

The final challenge I had is that when I implemented the FSM in Arduino, I made mistakes with the bitwise operations. I thought that the bitwise NOT '~' placed before a variable changed it from a 1 to a 0 and vice versa, but it inverts the input, changing a 1 to a -1. I could not figure out why this was happening, until I saw on an online forum that I should actually be using the logical 'NOT' which is '!'. After I substituted '!' for '~', my equations functioned properly.

Describe how you tested your project

After I finished drawing my diagram for the FSM, shown in Figure 2, I tested it by visually. I made sure that for every possible combination of inputs there was a defined transition either to another state or back to the same state.

As described earlier, I tested the FSM next state equations, shown in Appendix Figure 3, by implementing the FSM in Logisim, shown in Appendix Figure 4. I then tested each combination of inputs and present state to validate that the equations worked. Where I found an error, I reexamined my equation for the faulty next state bit and fixed it.

When I implemented the FSM with the Arduino, I repeated this check, trying every push button input combination for every state, and confirming that the output and next state were correct.

Describe any assembly programs, code, or circuitry you developed

Overall, an FSM was designed for a traffic light controller controlling the intersection of a highway and a road. It was implemented in Logisim to verify that it is functional, and then implemented in hardware using push buttons for inputs and LEDs controlled by an LED driver for outputs, with an Arduino Uno as the controller.

The Logisim circuit file screenshots are in Figure 4 of the Appendix. I implemented the Logisim circuit similarly to how the Moore FSM counter was done in the lab. The DFFs' output was the current state / output and the logic gates' output was the next state.

The Arduino code is described in the last paragraph of the overall project design and specifications section. One thing I may add is that the next state bits were calculated using bitwise operations, essentially simulating the logic gates of the Logisim circuit. The Arduino code can be seen in Figure 7 of the Appendix.

What improvements would you make / new features you would add if you had more time

One improvement I would make is reducing the number of bits it takes to encode the current state / output. Currently, the eight bits it takes is overkill to express the eight different states, it could be done with three bits! The current state / output bit structure was conceived to make it easy to see the current state of each traffic light in the system, but it is not efficient.

Another improvement I would make is changing the 'green arrow' (protected turn) color from green and yellow together to be just a flashing green, to make it more realistic, because one never sees green and yellow together in a real traffic light system. This could potentially confused drivers and lead to accidents.

Appendix

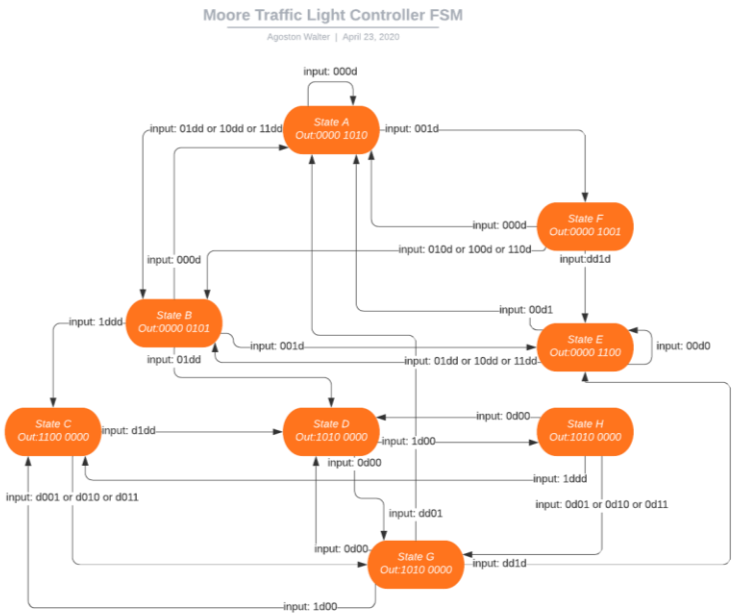

Figure 2: Moore Traffic Light Controller FSM

	Next State ($Y_8Y_7Y_6Y_5Y_4Y_3Y_2Y_1$) for $i_4i_3i_2i_1$ =															
Present State / Output	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
A 0000 1010	0000 1010	0000 1010	0000 1001	0000 1001	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101
B 0000 0101	0000 1010	0000 1010	0000 1100	0000 1100	1010 0000	1010 0000	1010 0000	1010 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000
C 1100 0000	1100 0000	0101 0000	0101 0000	0101 0000	1010 0000	1010 0000	1010 0000	1010 0000	1100 0000	0101 0000	0101 0000	0101 0000	1010 0000	1010 0000	1010 0000	1010 0000
D 1010 0000	1010 0000	0101 0000	0101 0000	0101 0000	1010 0000	0101 0000	0101 0000	0101 0000	1001 0000	0101 0000	0101 0000	0101 0000	1001 0000	0101 0000	0101 0000	0101 0000
E 0000 1100	0000 1100	0000 1010	0000 1100	0000 1010	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101	0000 0101
F 0000 1001	0000 1010	0000 1010	0000 1100	0000 1100	0000 0101	0000 0101	0000 1100	0000 1100	0000 0101	0000 0101	0000 1100	0000 1100	0000 0101	0000 0101	0000 1100	0000 1100
G 0101 0000	1010 0000	0000 1010	0000 1100	0000 1100	1010 0000	0000 1010	0000 1100	0000 1100	1100 0000	0000 1010	0000 1100	0000 1100	1100 0000	0000 1010	0000 1100	0000 1100
H 1001 0000	1010 0000	0101 0000	0101 0000	0101 0000	1010 0000	0101 0000	0101 0000	0101 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000	1100 0000
Y_8	(B & (i3 | i4)) | (C & (i3 | (!i2 & !i1))) | (D & (!i1 & !i2)) | (G & (!i1 & !i2)) | (H & (i4 | (!i1 & !i2 & !i4)))															
Y_7	(B & i4) | (C & !i3) | (D & (i1 | i2)) | (G & (!i1 & !i2 & i4)) | (H & (i4 | i1 | i2))															
Y_6	(B & i3 & !i4) | (C & i3) | (D & (!i1 & !i2 & !i4)) | (G & (!i1 & !i2 & !i4)) | (H & (!i4 & !i1 & !i2))															
Y_5	(C & !i3 & (i1 | (!i1 & i2))) | (D & (i1 | i2 | i4)) | (H & (!i4 & (i1 | (!i1 & i2))))															
Y_4	(A & !i3 & !i4) | (B & !i3 & !i4) | (E & !i3 & !i4) | (F & ((i4 & i2) | (!i4 & !i3) | (!i4 & i3 & i2))) | (G & (i2 | (i1 & !i2)))															
Y_3	(A & (i3 | i4)) | (B & (i2 & !i3 & !i4)) | (E & (!i1 | i3 | i4)) | (F & (i4 | i3 | i2)) | (G & i2)															
Y_2	(A & !i3 & !i4 & !i2) | (B & !i3 & !i4 & !i2) | (E & !i3 & !i4 & i1) | (F & !i4 & !i2 & !i3) | (G & !i2 & i1)															
Y_1	(A & (i3 | i4 | i2)) | (E & (i3 | i4)) | (F & ((i4 & !i2 & !i3) | (i3 & !i2)))															

Figure 3: Finite State Machine Truth Table with Next State Bit Equations

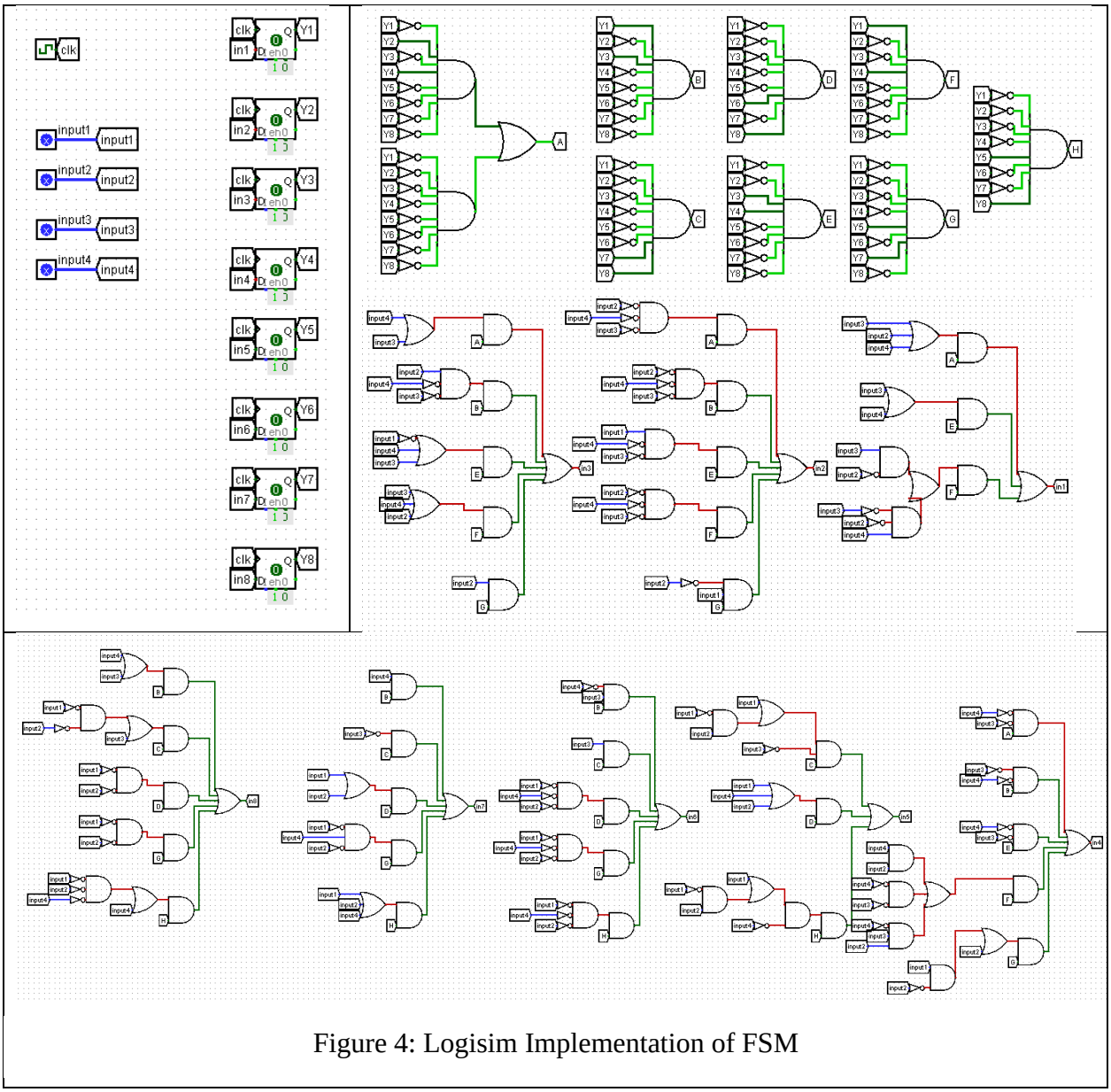

Figure 4: Logisim Implementation of FSM

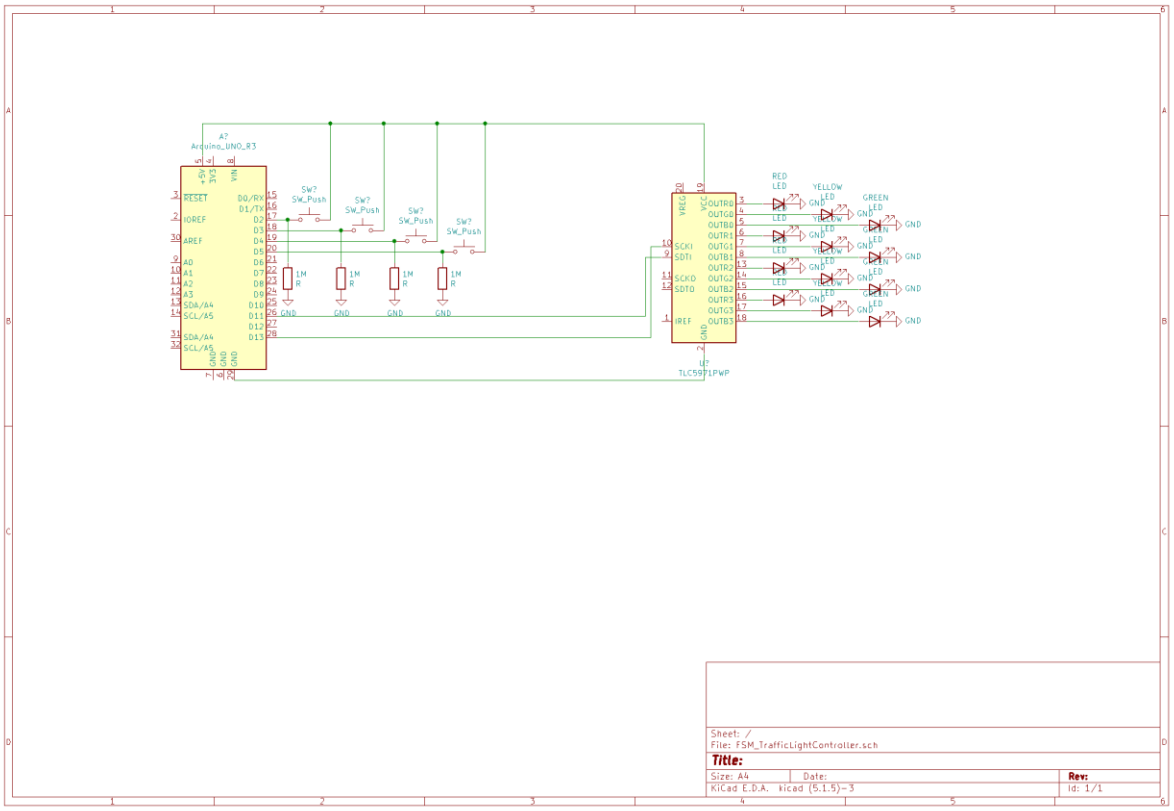

Figure 5: KiCad Schematic of Arduino FSM Controller Circuit

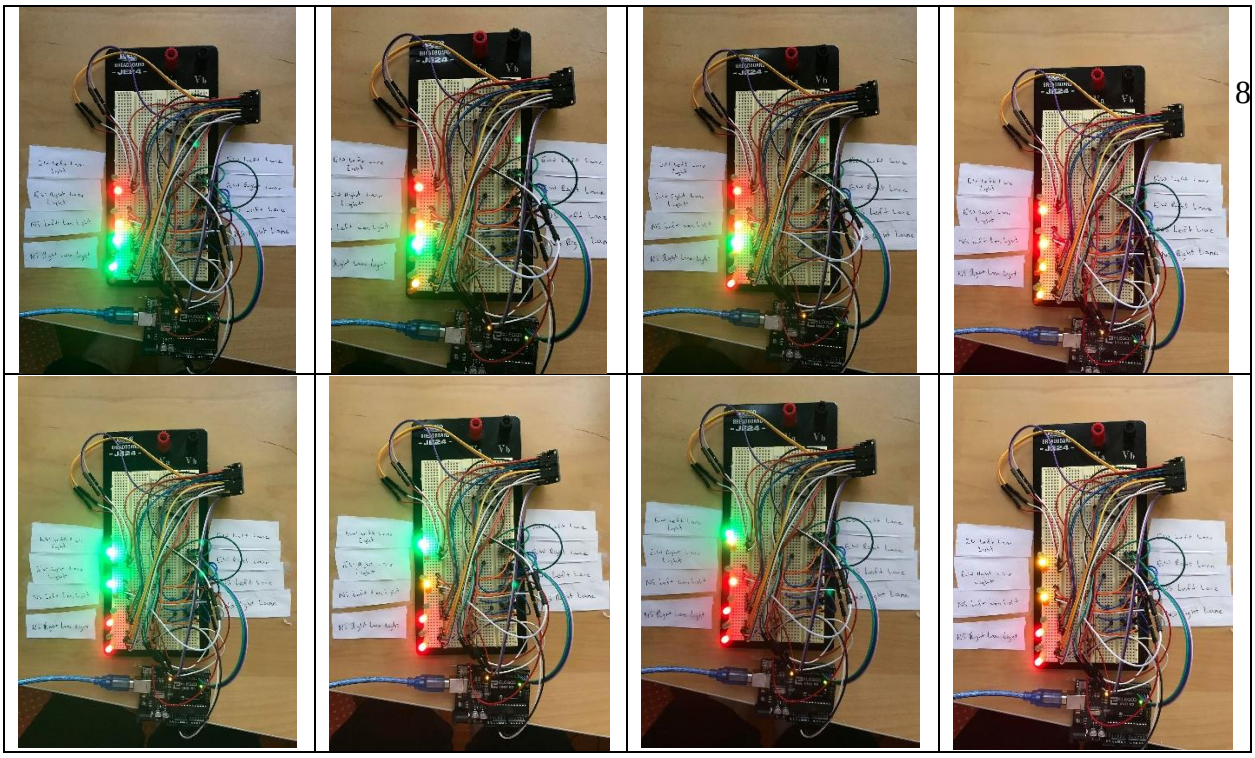

Figure 6: Pictures of Eight Different Outputs States for Physical Arduino Setup

LED labels on left, Push button input labels on right

```
#include "Adafruit_TLC59711.h"
#include <SPI.h>

#define NUM_TLC59711 2

#define data 11
#define clock 13


unsigned int cs8 = B0;
unsigned int cs7 = B0;
unsigned int cs6 = B0;
unsigned int cs5 = B0;
unsigned int cs4 = B1;
unsigned int cs3 = B0;
unsigned int cs2 = B1;
unsigned int cs1 = B0;

Adafruit_TLC59711 tlc = Adafruit_TLC59711(NUM_TLC59711, clock, data);

void setup() {
Serial.begin(9600);
pinMode(10,OUTPUT);
pinMode(2,INPUT);
pinMode(3,INPUT);
pinMode(4,INPUT);
pinMode(5,INPUT);
tlc.begin();
tlc.write();
}

void loop() {
unsigned int i1 = digitalRead(2);Serial.print("i1 = ");Serial.println(i1);
unsigned int i2 = digitalRead(3);Serial.print("i2 = ");Serial.println(i2);
```

```
unsigned int i3 = digitalRead(4);Serial.print("i3 = ");Serial.println(i3);
unsigned int i4 = digitalRead(5);Serial.print("i4 = ");Serial.println(i4);


unsigned int A = (!cs1 & cs2 & !cs3 & cs4 & !cs5 & !cs6 & !cs7 & !cs8) | (!cs1 & !cs2 & !cs3 & !cs4 & !cs5 & !cs6 & !cs7 & !cs8); Serial.print("A IS
THIS LINE:");Serial.print(A,BIN);
unsigned int B = cs1 & !cs2 & cs3 & !cs4 & !cs5 & !cs6 & !cs7 & !cs8; Serial.print("B IS THIS LINE:");Serial.println(B,BIN);
unsigned int C = !cs1 & !cs2 & !cs3 & !cs4 & !cs5 & !cs6 & cs7 & cs8; Serial.print(C,BIN);
unsigned int D = !cs1 & !cs2 & !cs3 & !cs4 & !cs5 & cs6 & !cs7 & cs8; Serial.print(D,BIN);
unsigned int E = !cs1 & !cs2 & cs3 & cs4 & !cs5 & !cs6 & !cs7 & !cs8; Serial.print(E,BIN);
unsigned int F = cs1 & !cs2 & !cs3 & cs4 & !cs5 & !cs6 & !cs7 & !cs8; Serial.print(F,BIN);
unsigned int G = !cs1 & !cs2 & !cs3 & !cs4 & cs5 & !cs6 & cs7 & !cs8; Serial.print(G,BIN);
unsigned int H = !cs1 & !cs2 & !cs3 & !cs4 & cs5 & !cs6 & !cs7 & cs8; Serial.println(H,BIN);

unsigned int ns8 = (B & (i3 | i4)) | (C & (i3 | (!i2 & !i1))) | (D & (!i1 & !i2)) | (G & (!i1 & !i2)) | (H & (i4 | (!i1 & !i2 & !i4)));//confirmed
unsigned int ns7 = (B & i4) | (C & !i3) | (D & (i1 | i2)) | (G & (!i1 & !i2 & i4)) | (H & (i4 | i1 | i2)); //confirmed
unsigned int ns6 = (B & i3 & !i4) | (C & i3) | (D & (!i1 & !i2 & !i4)) | (G & (!i1 & !i2 & !i4)) | (H & (!i4 & !i1 & !i2)); //confirmed
unsigned int ns5 = (C & !i3 & (i1 | (!i1 & i2))) | (D & (i1 | i2 | i4)) | (H & (!i4 & (i1 | (!i1 & i2)))); //confirmed
unsigned int ns4 = (A & !i3 & !i4) | (B & !i3 & !i4) | (E & !i3 & !i4) | (F & ((i4 & i2) | (!i4 & !i3) | (!i4 & i3 & i2))) | (G & (i2 | (i1 & !i2)));
//confirmed
unsigned int ns3 = (A & (i3 | i4)) | (B & (i2 & !i3 & !i4)) | (E & (!i1 | i3 | i4)) | (F & (i4 | i3 | i2)) | (G & i2); //confirmed
unsigned int ns2 = (A & !i3 & !i4 & !i2) | (B & !i3 & !i4 & !i2) | (E & !i3 & !i4 & i1) | (F & !i4 & !i2 & !i3) | (G & !i2 & i1);//confirmed
unsigned int ns1 = (A & (i3 | i4 | i2)) | (E & (i3 | i4)) | (F & ((i4 & !i2 & !i3) | (i3 & !i2))); //confirmed

cs8 = ns8;cs7 = ns7;cs6 = ns6;cs5 = ns5;cs4 = ns4;cs3 = ns3;cs2 = ns2;cs1 = ns1;
Serial.print("CURRENT STATE:
");Serial.print(cs8,BIN);Serial.print(cs7,BIN);Serial.print(cs6,BIN);Serial.print(cs5,BIN);Serial.print(cs4,BIN);Serial.print(cs3,BIN);Serial.print(cs2,BI
N);Serial.print(cs1,BIN);Serial.println();

if (cs2 ==0 && cs1 == 0){
tlc.setLED(4, 65535, 0, 0); //set to red
tlc.write();
delay(100);
}
if (cs2 ==0 && cs1 == 1){
tlc.setLED(4, 0, 65535, 0); //set to yellow
tlc.write();
delay(100);
}
if (cs2 ==1 && cs1 == 0){
tlc.setLED(4, 0, 0, 65535); //set to green
tlc.write();
delay(100);
}
if (cs2 ==1 && cs1 == 1){
tlc.setLED(4, 0, 65535, 65535); //set to priority green
tlc.write();
delay(100);
}

if (cs4 ==0 && cs3 == 0){
tlc.setLED(5, 65535, 0, 0); //set to red
tlc.write();
delay(100);
}
if (cs4 ==0 && cs3 == 1){
tlc.setLED(5, 0, 65535, 0); //set to yellow
tlc.write();
delay(100);
}
if (cs4 ==1 && cs3 == 0){
```

```
tlc.setLED(5, 0, 0, 65535); //set to green
tlc.write();
delay(100);
}
if (cs4 ==1 && cs3 == 1){
tlc.setLED(5, 0, 65535, 65535); //set to priority green
tlc.write();
delay(100);
}

if (cs6 ==0 && cs5 == 0){
tlc.setLED(6, 65535, 0, 0); //set to red
tlc.write();
delay(100);
}
if (cs6 ==0 && cs5 == 1){
tlc.setLED(6, 0, 65535, 0); //set to yellow
tlc.write();
delay(100);
}
if (cs6 ==1 && cs5 == 0){
tlc.setLED(6, 0, 0, 65535); //set to green
tlc.write();
delay(100);
}
if (cs6 ==1 && cs5 == 1){
tlc.setLED(6, 0, 65535, 65535); //set to priority green
tlc.write();
delay(100);
}

if (cs8 ==0 && cs7 == 0){
tlc.setLED(7, 65535, 0, 0); //set to red
tlc.write();
delay(100);
}
if (cs8 ==0 && cs7 == 1){
tlc.setLED(7, 0, 65535, 0); //set to yellow
tlc.write();
delay(100);
}
if (cs8 ==1 && cs7 == 0){
tlc.setLED(7, 0, 0, 65535); //set to green
tlc.write();
delay(100);
}
if (cs8 ==1 && cs7 == 1){
tlc.setLED(7, 0, 65535, 65535); //set to priority green
tlc.write();
delay(100);
}
delay(5000);
}
```

Figure 7: Arduino Code