California State University, Fresno

Lyles College of Engineering

Electrical and Computer Engineering Department

TECHNICAL REPORT

Final Project Title: Final Project Report

Course Title: ECE 176

Date Submitted: 5/18/2022

Course ID: 35031

Prepared By:

Edgar Kenner

Aryan Singh

Instructor:

Aaron Stillmaker

INSTRUCTOR SECTION

Comments: _____

_____

_____

_____

Final Grade: _____

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this project is to create an adjustable timing scheme that allows for the optimal flow of traffic through an intersection. This will be an improvement over traffic light control systems that operate strictly on set timers and an improvement over simple detection-based traffic light control schemes.

1.1 Division of Work

Most of the work was done in a group with separate tasks being the timer module and the testbench that Edgar was able to implement and the case statements in the TLC module that Aryan was able to perform. The Report and Presentation were done by Edgar and Aryan with some editing done by both on each other's work.

2. THEORETICAL BACKGROUND

2.1 Four-way Intersection

The typical four-way intersection has a length between traffic lights of about 20 feet, as can be seen below in Figure 2.1 (the actual length in this figure would be a bit greater than 20 feet).

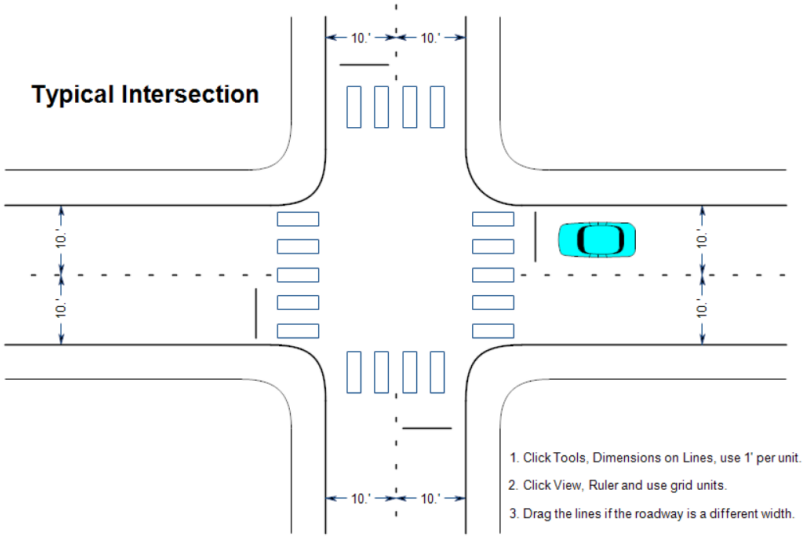

Figure 2.1: Typical Intersection [1]

If a speed limit of 50 miles per hour is assumed, this distance can be used to determine the absolute minimum speed that a signal has to process cars going through the intersection. This can be done using the following equation:

$$min\_speed = (length\_of\_intersection/(5280) * 3600/speed\_limit)^{-1} \quad (2.1)$$

Where:

min_speed = The minimum clock speed in seconds
length_of_intersection = The length of the intersection in feet
speed_limit = The speed limit in miles per hour

The length of the intersection is divided by 5280 to convert it from feet to miles, and it is multiplied by 3600 to go from hours to seconds, and the whole thing is inverted to get the clock speed in hertz. The clock speed should actually be significantly faster than the minimum speed in order to account for vehicles that are speeding and so that cars that are very close to each other do not get counted as a single vehicle.

2.2 Verilog

Verilog is a hardware descriptive language (HDL) used to program the functions of actual physical hardware. It is also a register transfer language (RTL), meaning that it is capable of performing micro-operations which the result of can be stored in and transferred between registers.

Verilog is needed in this project to program the functionality of the traffic lights, controlling when what lights in an intersection are green and for how long depending on the circumstances. Another HDL language, such as VHDL, could have been used, but for this project, Verilog was chosen.

2.3 Synthesis

Synthesis is the act of taking an HDL code and translating it into an appropriate array of gates and flip-flops that can perform the code. This result is known as the gate netlist. A simple example of this is demonstrated below in Figure 2.2 with the synthesization of a LUT.

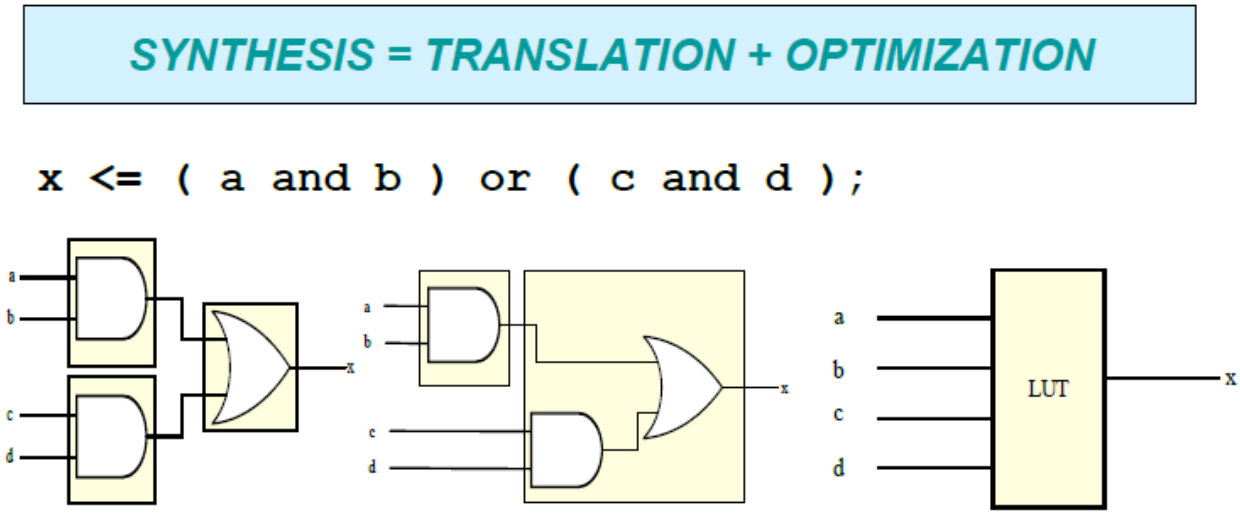

Figure 2.2: Simple Synthesis Example [2]

The synthesis process also includes gathering the area, power, and timing data. This data is crucial to figuring out the overall size, speed, and energy consumption of the resulting device.

3. SPECIFICATIONS/GOALS

The goal of this project is to allow for the optimal flow of traffic through an intersection, taking into consideration the frequency and prioritizing specific traffic coming through the intersection. This is accomplished by constantly sensing the traffic through the intersection in order to determine which state the traffic light needs to be in (current state), determine what its next state should be, and determine how long a state should be held(the delay in the changing of the traffic lights).

Which state the traffic light should be in will be determined by how much traffic is present at each of the individual states. The next state will be determined by vehicles being sensed and the number of vehicles waiting at each of the non-flowing states, as well as by whether or not it is forward traffic or a protected left turn. Hold time for a state will be determined by the frequency of the traffic currently flowing through the selected state, which would be the green light state, and how many cars are waiting at the unselected, red light, states.

4. DESIGN

4.1 Equipment Used

In order to accomplish this project, a computer with access to LucidChart, Modelsim, and Cadence/Synopsis was used.

4.2 Design Process

To start the project, first, a flow chart was created to plan out how the code would operate so that our goals could be achieved, as shown below in Figure 4.1.

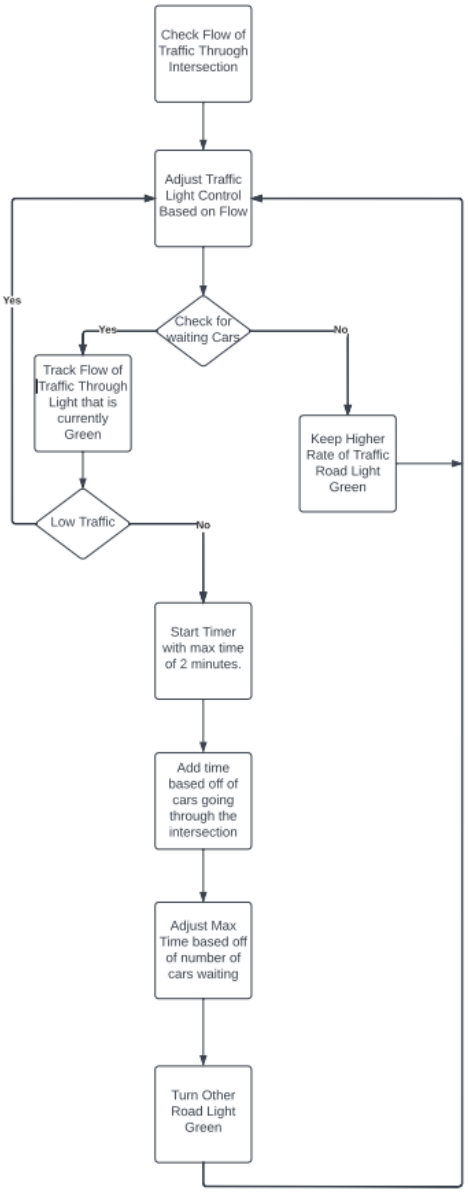

Figure 4.1: FlowChart for Code

The idea is that sensors will be constantly monitoring the flow and rate of traffic, as well as counting the number of cars waiting. The traffic light control is adjusted based on the flow of traffic, and how quickly it adjusts is based on the number of cars waiting and the rate of traffic driving through the currently active state. If there are no cars waiting, then the present light state is held, however if there are cars waiting then the code checks if the rate of traffic is low. If the rate of traffic is low, then the code proceeds to switch the light to the next highest priority state. If the rate of traffic is not low, the cars waiting still cannot be allowed to wait for an unreasonable amount of time. As such, a default max time of two minutes is set for a car to wait (this value was chosen arbitrarily). This max time that cars have to wait is decreased by the number of cars waiting, as switching the lights' state then becomes a higher priority. However,

this max time is not necessarily always gonna be met, as the wait time is based off of the rate of traffic, increasing as the rate of traffic increases. The max time is simply a ceiling for the wait time that the rate of traffic cannot cause the wait time to exceed. Finally, when this wait time has finished, the code adjusts the light state to the next highest priority state.

After the flow chart was created, next, a block diagram was created in order to figure out what signals and modules would be needed to realize the process mapped out in the flow chart, as shown below in Figure 4.2.

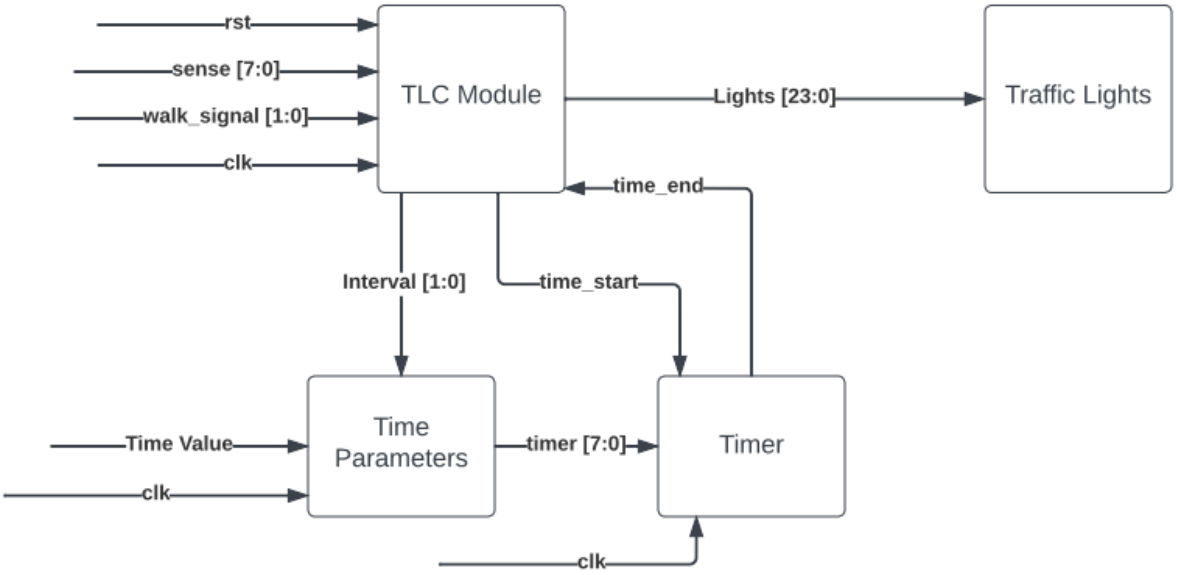

Figure 4.2: Original Block Diagram of Project Code

This block diagram is not representative of what the final code looks like, but it was used to get a good idea of how to start. An eight-bit sense signal (one bit for each of the different function lights in the intersection) would be used to determine what the state would need to change to in line with a clock signal. This clock signal would also be used for timing how long a state needs to be held within a separate timer module. However, in the final traffic light control (TLC) module, the timer module and the TLC module would actually be combined into a single TLC module. The 24-bit lights output is then the signal that actually represents what colors are on, one bit for each of the three colors for the eight different lights. The block diagram shown in Figure 4.3 is a bit more indicative of what the final project actually looks like.

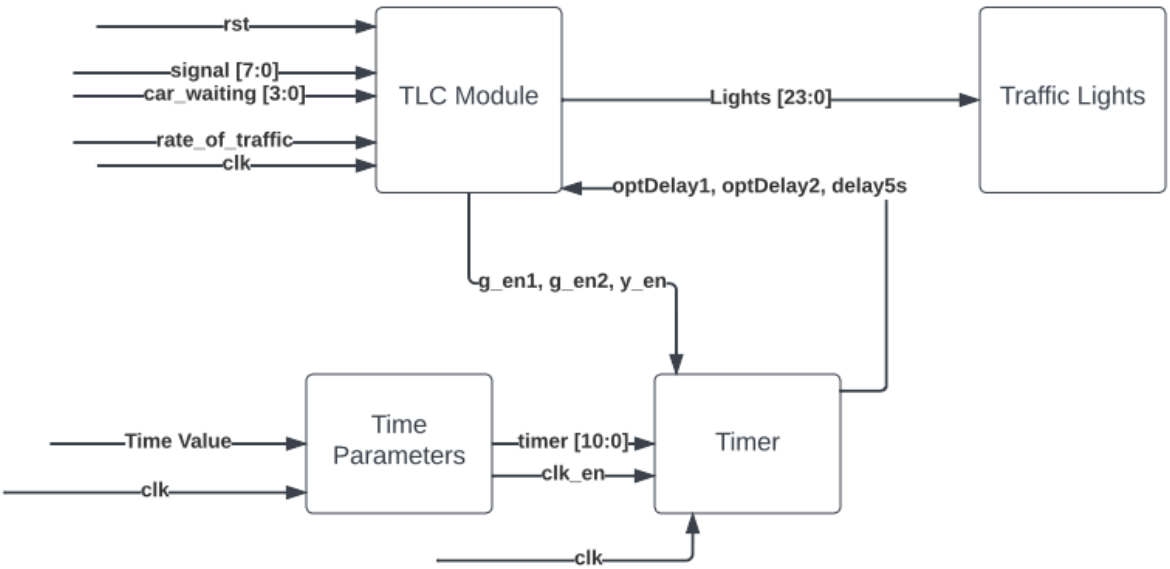

Figure 4.3: Final Block Diagram for Project Code

Finally, a state diagram was created to figure out how the modules would work and give an idea of how the body of the modules would be started, as shown below in Figure 4.3.

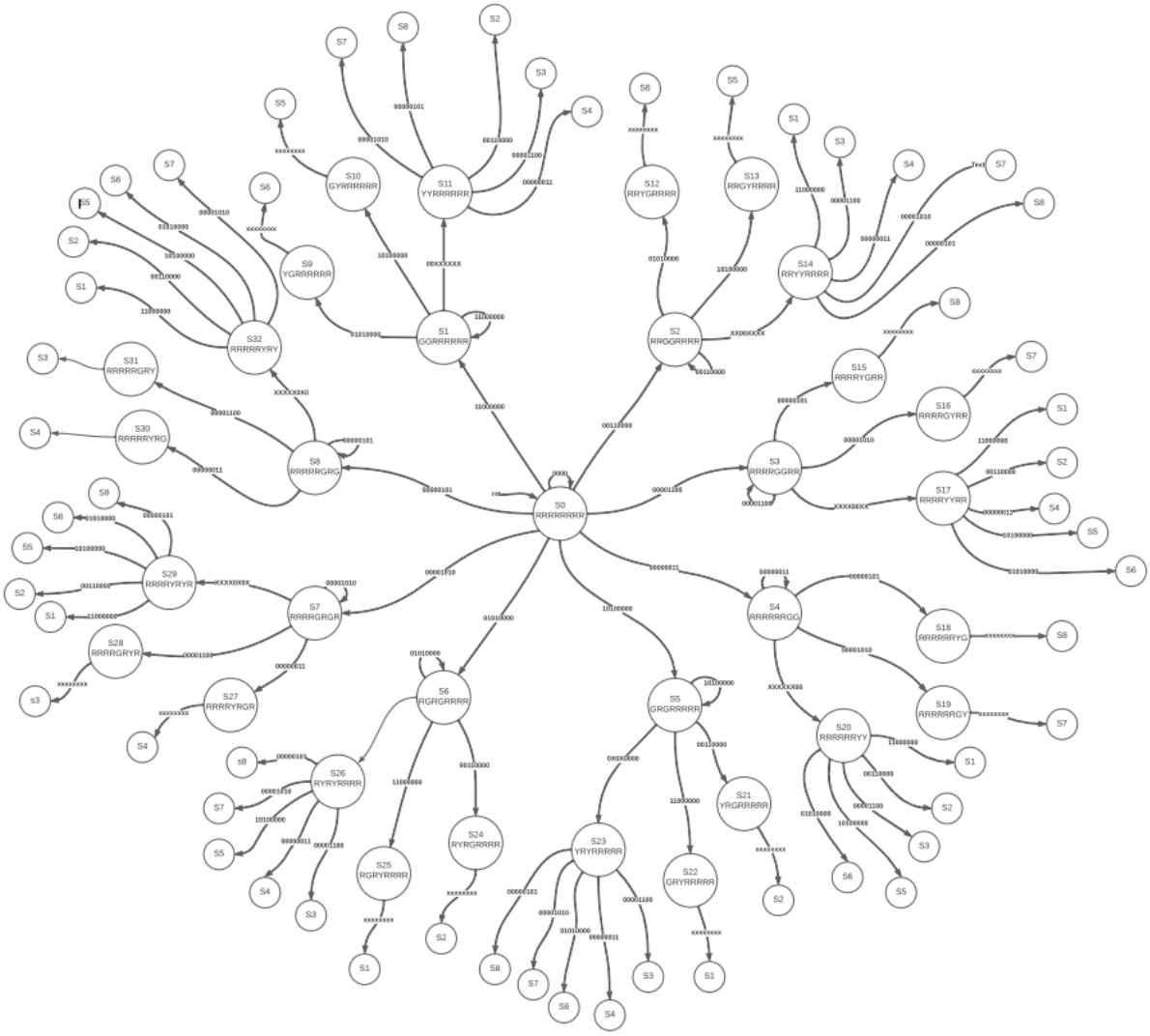

Figure 4.4: Traffic Light State Diagram

In the state diagram states 1-8 represent the eight different possible green light combinations (with state 0 being a default state where all of the lights are red, this program is only ever in this state at the very start of operation before a signal is detected, or if there is a reset). These eight possible states consist of the two parallel and opposite forward lights, a forward light and its adjacent protected right turn light, and the two opposite protected right turn lights running at the same time. Each of these states must go to one of three possible yellow lights states as its next state (one for both green lights turning yellow, and two for both cases where just one of the green lights turns yellow). These three yellow states then go to one of the seven remaining green light states. Note that two of the three yellow light states for each green light state only has one other green light state that it can go to. This is because only one of their green lights turned yellow, which means there is only one possible green light state that could be its next state. For the yellow light states where both green lights turned yellow, it can possibly go to any of the original green light states, except for the state it came from and the two states that the other two yellow light states have to go to.

5. IMPLEMENTATION

To start, the implementation of the state diagram was coded in Verilog using Modelsim within the TLC module. This was accomplished by using case statements where the case of the present state was used to set the value of the light, and another case statement, where the case was the signal being sensed by the sensors, which would set the state to its appropriate next_state at the positive edge of the clock. Note that every state consisting only of green and red lights needs to first go to a state with the appropriate yellow lights before going on to the next green and red light state, as demonstrated in Figure 5.1.

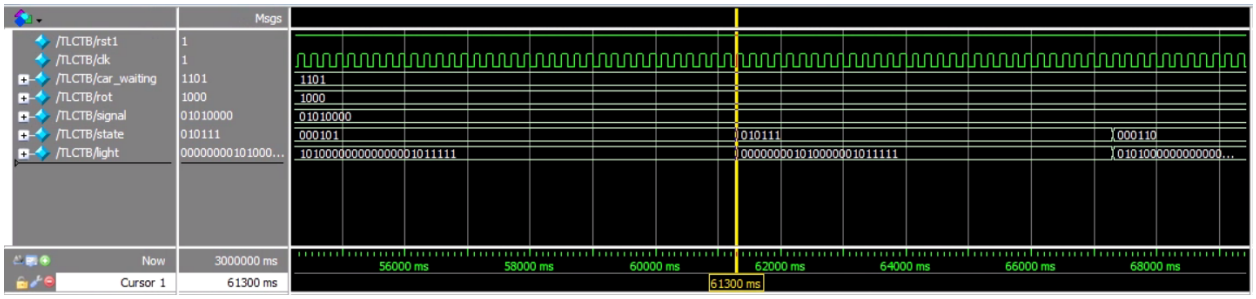

Figure 5.1: Current State Transitioning to a Yellow State before the Next Green State

Next, a timer needed to be coded such that the states would be held for an appropriate amount of time. When holding a green state, the amount of time held would be influenced by the rate of traffic through the current state and the number of cars waiting in other states. If there are no cars waiting then the present state is held, but if there is a car waiting the time that the present state continues to be held starts counting down, with this countdown being either increased or decreased based on how many vehicles are actively driving through the active state and the maximum countdown time is decreased the more cars there are waiting. When holding a yellow state the timer countdown is much simpler, the state is just held for five seconds.

During the coding of this timer, it was realized that enable signals would be needed to use the appropriate timers at the right time. These enable signals then had their values set by going back to the case statements and enabling the adjustable timer whenever the traffic light was in a green state, and enabling the five-second timer whenever the traffic light was in a yellow state.

For the timer to actually work three one-bit registers were made, such that the state could only be changed if its corresponding register held a one. These registers were named “delay5s”, “optDelay1”, and “optDelay2”, and they were the condition for the yellow light states, forward traffic states, and protected left-turn states respectively. Signals called clk_en and count were then used to count a second, with count having one added to it at every positive clock edge and clk_en was set to one whenever count would be equal to the frequency of the clock, where the count was then reset to zero, effectively allowing clk_en to be set equal to one every second

If the clk_en is equal to one, then as long as any of the other enable signals (g_en1, g_en2, and y_en) are set to one then a signal count_delay, which was originally initialized to zero, is increased by one. Then, depending on which enable signal is on, count_delay will count until it is either equal to timer1, timer2, or 4 so that optDelay1, optDelay2, or delay5s can be set equal to one and the others to zero, and count_delay is then reset back to zero. If none of the

enable signals are active, then all of the delay conditions are set to zero. States being held for their respective appropriate times thanks to this timer system can be seen in Figure 5.2.

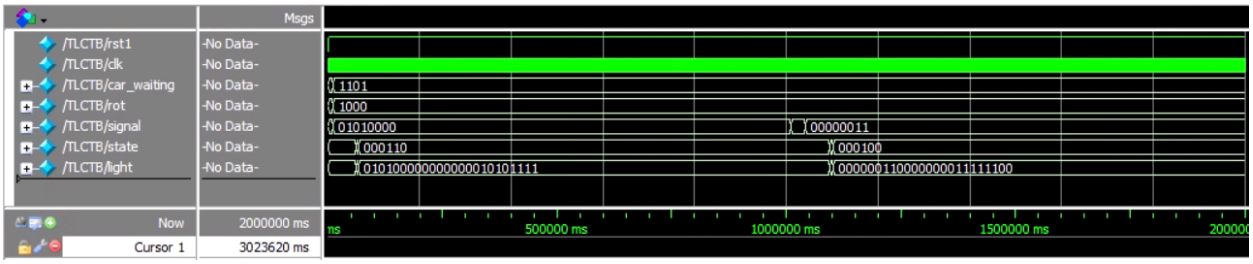

Figure 5.2: Waveform Showing Testbench Results Over 2000 Seconds

6. CONCLUSION

This project has resulted in a traffic light control scheme that can result in an optimal flow of traffic, greatly reducing the average time spent stopped for a vehicle at an intersection. However, one condition that wasn't considered and was not noticed as a problem, was what to do when multiple signals are being sensed before the light has had a chance to change. Some sort of lock mechanism should have been implemented to make sure that each signal received has a chance to change the state of the lights. As is, if a new signal is detected before the previous signal has had a chance to change the state due to the previous state still being held, then the new signal overwrites the previously detected signal and the hold timer is reset from there. Through the completion of this project, Verilog coding techniques that were taught throughout the semester were relearned and reinforced, being used in a project of original design. These techniques include using if-else statements and case statements, setting parameters, using for loops in a testbench, and using a condition expression. Some other learnings that we can take away from this class and this project specifically, is how the implementation of traffic lights and sensors takes place in the real world, showing us how the software and hardware part of engineering are interlinked with each other and how crucial both these systems are for optimal implementation of day to day scenarios.

7. REFERENCES

[1] Rff.com. 2022. A Typical Traffic Intersection. [online] Available at: <https://www.rff.com/TypicalIntersection.php> [Accessed 18 May 2022].

[2] Vlsi-project.blogspot.com. 2022. Synthesis Issues. [online] Available at: <http://vlsi-project.blogspot.com/2011/01/synthesis-issues.html> [Accessed 18 May 2022].

8. APPENDICES

Appendix A: Main TLC Module

```
`timescale 100ms/1ms
module TLC(
input rst1, /*rst2,*/ clk,
input [3:0] car_waiting, rot,
input [7:0] signal, //might need to replace 8-bit signal with 4 2-bit sensors or just add them
//input [1:0] wr, //s0, s1, s2, s3,
//output reg [3:0] rot,
output reg [5:0] state,
output reg [23:0] light
);
reg low_traffic = 0;
//reg [3:0] rot = 4'b0000;
reg [5:0] p_state = 6'b000000, n_state = 6'b000000;
reg [7:0] p_signal;
parameter start_time = 10;
parameter start_timepl = 30;
parameter max_time = 120;
reg g_en1 = 0, g_en2 = 0, y_en = 0, optDelay1 = 0, optDelay2 = 0, delay5s = 0;
reg [3:0] count = 4'b0000;
reg [10:0] count_delay = 11'b00000000000, timer1 = 11'b00000000000, timer2 =
11'b00000000000;
wire clk_en;
//Trying to increase rot whenever a car goes through the intersection and decrease it whenever
one doesn't
/*always@(negedge clk) begin
p_state <= n_state;
p_signal <= signal;
end
always@(posedge clk)begin
if(p_signal == signal)
        rot <= rot +1;
else
        rot <= rot -1;
end*/

//Represents the 33 possible states of the Traffic Lights
always@(posedge clk or negedge rst1) begin
if(~rst1) begin
        state <= 6'b00000;
```

```
        light <= 24'b000000000000000011111111;
end else
        state <= n_state;
end

always@(*) begin
case(state)
        6'b00000 : begin
        light <= 24'b000000000000000011111111;
        case(signal)
                8'b00000000 : begin
                n_state <= 6'b000000;
                end
                8'b11000000 : begin
                n_state <= 6'b000001;
                end
                8'b00110000 : begin
                n_state <= 6'b000010;
                end
                8'b00001100 : begin
                n_state <= 6'b000011;
                end
                8'b00000011 : begin
                n_state <= 6'b000100;
                end
                8'b10100000 : begin
                n_state <= 6'b000101;
                end
                8'b01010000 : begin
                n_state <= 6'b000110;
                end
                8'b00001010 : begin
                n_state <= 6'b000111;
                end
                8'b00000101 : begin
                n_state <= 6'b001000;
                end
                default : n_state <= 6'b000000;
        endcase
        end
```

```
6'b00001 : begin
light <= 24'b110000000000000000111111;
y_en = 0;
case(signal)
        8'b11000000 : begin
        /*g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b000001;
        else */n_state <= 6'b000001;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b001011;
        else n_state <= 6'b000001;
        end
        8'b00001100 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b001011;
        else n_state <= 6'b000001;
        end
        8'b00000011 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b001011;
        else n_state <= 6'b000001;
        end
        8'b10100000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001010;
        else n_state <= 6'b000001;
        end
```

```
        8'b01010000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001001;
        else n_state <= 6'b000001;
        end
        8'b00001010 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001011;
        else n_state <= 6'b000001;
        end
        8'b00000101 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001011;
        else n_state <= 6'b000001;
        end
        default : n_state <= 6'b000001;
endcase
end

6'b00010 : begin
light <= 24'b001100000000000011001111;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b001110;
        else n_state <= 6'b000010;
        end
        8'b00110000 : begin
        /*g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
```

```
n_state <= 6'b000010;
else */n_state <= 6'b000010;
end
8'b00001100 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b001110;
else n_state <= 6'b000010;
end
8'b00000011 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b001110;
else n_state <= 6'b000010;
end
8'b10100000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b001101;
else n_state <= 6'b000010;
end
8'b01010000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b001100;
else n_state <= 6'b000010;
end
8'b00001010 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b001110;
else n_state <= 6'b000010;
end
8'b00000101 : begin
g_en1 = 1;
```

```
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001110;
        else n_state <= 6'b000010;
        end
        default : n_state <= 6'b000010;
endcase
end

6'b00011 : begin
light <= 24'b000011000000000011110011;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010001;
        else n_state <= 6'b000011;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010001;
        else n_state <= 6'b000011;
        end
        8'b00001100 : begin
        /*g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b000011;
        else */n_state <= 6'b000011;
        end
        8'b00000011 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010001;
        else n_state <= 6'b000011;
```

```
        end
        8'b10100000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010001;
        else n_state <= 6'b000011;
        end
        8'b01010000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010001;
        else n_state <= 6'b000011;
        end
        8'b00001010 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010000;
        else n_state <= 6'b000011;
        end
        8'b00000101 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b001111;
        else n_state <= 6'b000011;
        end
        default : n_state <= 6'b000011;
endcase
end

6'b00100 : begin
light <= 24'b000000110000000011111100;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
```

```
if(optDelay2)
n_state <= 6'b010100;
else n_state <= 6'b000100;
end
8'b00110000 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b010100;
else n_state <= 6'b000100;
end
8'b00001100 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b010100;
else n_state <= 6'b000100;
end
8'b00000011 : begin
/*g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b000100;
else */n_state <= 6'b000100;
end
8'b10100000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b010100;
else n_state <= 6'b000100;
end
8'b01010000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b010100;
else n_state <= 6'b000100;
end
8'b00001010 : begin
```

```
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010011;
        else n_state <= 6'b000100;
        end
        8'b00000101 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010010;
        else n_state <= 6'b000100;
        end
        default : n_state <= 6'b000100;
endcase
end

6'b00101 : begin
light <= 24'b101000000000000001011111;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2) n_state <= 6'b010110;
        else n_state <= 6'b000101;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010101;
        else n_state <= 6'b000101;
        end
        8'b00001100 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010111;
        else n_state <= 6'b000101;
```

```
        end
        8'b00000011 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b010111;
        else n_state <= 6'b000101;
        end
        8'b10100000 : begin
        /*g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b000101;
        else */n_state <= 6'b000101;
        end
        8'b01010000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010111;
        else n_state <= 6'b000101;
        end
        8'b00001010 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010111;
        else n_state <= 6'b000101;
        end
        8'b00000101 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b010111;
        else n_state <= 6'b000101;
        end
        default : begin
        n_state <= 6'b000101;
        end
endcase
```

```
end

6'b00110 : begin
light <= 24'b010100000000000010101111;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011001;
        else n_state <= 6'b000110;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011000;
        else n_state <= 6'b000110;
        end
        8'b00001100 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011010;
        else n_state <= 6'b000110;
        end
        8'b00000011 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011010;
        else n_state <= 6'b000110;
        end
        8'b10100000 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b011010;
        else n_state <= 6'b000110;
```

```
        end
        8'b01010000 : begin
        /*g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b000110;
        else */n_state <= 6'b000110;
        end
        8'b00001010 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b011010;
        else n_state <= 6'b000110;
        end
        8'b00000101 : begin
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b011010;
        else n_state <= 6'b000110;
        end
        default : n_state <= 6'b000110;
endcase
end

6'b00111 : begin
light <= 24'b000010100000000011110101;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011101;
        else n_state <= 6'b000111;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
```

```
if(optDelay2)
n_state <= 6'b011101;
else n_state <= 6'b000111;
end
8'b00001100 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b011100;
else n_state <= 6'b000111;
end
8'b00000011 : begin
g_en1 = 0;
g_en2 = 1;
if(optDelay2)
n_state <= 6'b011011;
else n_state <= 6'b000111;
end
8'b10100000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b011101;
else n_state <= 6'b000111;
end
8'b01010000 : begin
g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b011101;
else n_state <= 6'b000111;
end
8'b00001010 : begin
/*g_en1 = 1;
g_en2 = 0;
if(optDelay1)
n_state <= 6'b000111;
else */n_state <= 6'b000111;
end
8'b00000101 : begin
```

```
        g_en1 = 1;
        g_en2 = 0;
        if(optDelay1)
        n_state <= 6'b011101;
        else n_state <= 6'b000111;
        end
        default : n_state <= 6'b000111;
endcase
end

6'b01000 : begin
light <= 24'b000001010000000011111010;
y_en = 0;
case(signal)
        8'b11000000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b100000;
        else n_state <= 6'b001000;
        end
        8'b00110000 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b100000;
        else n_state <= 6'b001000;
        end
        8'b00001100 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011111;
        else n_state <= 6'b001000;
        end
        8'b00000011 : begin
        g_en1 = 0;
        g_en2 = 1;
        if(optDelay2)
        n_state <= 6'b011110;
```

```
            else n_state <= 6'b001000;
            end
            8'b10100000 : begin
            g_en1 = 1;
            g_en2 = 0;
            if(optDelay1)
            n_state <= 6'b100000;
            else n_state <= 6'b001000;
            end
            8'b01010000 : begin
            g_en1 = 1;
            g_en2 = 0;
            if(optDelay1)
            n_state <= 6'b100000;
            else n_state <= 6'b001000;
            end
            8'b00001010 : begin
            g_en1 = 1;
            g_en2 = 0;
            if(optDelay1)
            n_state <= 6'b100000;
            else n_state <= 6'b001000;
            end
            8'b00000101 : begin
            /*g_en1 = 1;
            g_en2 = 0;
            if(optDelay1)
            n_state <= 6'b001000;
            else */n_state <= 6'b001000;
            end
            default : n_state <= 6'b001000;
   endcase
   end

   6'b001001 : begin
light <= 24'b010000001000000000111111;
   g_en1 = 0;
   g_en2 = 0;
   y_en = 1;
if(delay5s) n_state <= 6'b000110;
```

```
    else n_state <= 6'b001001;
end

    6'b001010 : begin
    light <= 24'b100000000100000000111111;
    g_en1 = 0;
    g_en2 = 0;
    y_en = 1;
    if(delay5s) n_state <= 6'b000101;
    else n_state <= 6'b001010;
    end

    6'b001011 : begin
    light <= 24'b000000001100000000111111;
    g_en1 = 0;
    g_en2 = 0;
    y_en = 1;
    if(delay5s) begin
            case(signal)
    8'b00001010 : begin
            n_state <= 6'b000111;
            end
            8'b00000101 : begin
            n_state <= 6'b001000;
            end
            8'b00110000 : begin
            n_state <= 6'b000010;
            end
            8'b00001100 : begin
            n_state <= 6'b000011;
            end
            8'b00000011 : begin
            n_state <= 6'b000100;
            end
            default : n_state <= 6'b001011;
    endcase
    end else n_state <= 6'b001011;
    end

    6'b001100 : begin
```

```
light <= 24'b000100000010000011001111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000110;
else n_state <= 6'b001100;
end

6'b001101 : begin
light <= 24'b001000000001000011001111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000101;
else n_state <= 6'b001101;
end

6'b001110 : begin
light <= 24'b000000000011000011001111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b11000000 : begin
        n_state <= 6'b000001;
        end
        8'b00001100 : begin
        n_state <= 6'b000011;
        end
        8'b00000011 : begin
        n_state <= 6'b000100;
        end
        8'b00001010 : begin
        n_state <= 6'b000111;
        end
        8'b00000101 : begin
        n_state <= 6'b001000;
        end
        default : n_state <= 6'b001110;
```

```
endcase
end else n_state <= 6'b001110;
end

6'b001111 : begin
light <= 24'b000001000000100011110011;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b001000;
else n_state <= 6'b001111;
end

6'b010000 : begin
light <= 24'b000010000000010011110011;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000111;
else n_state <= 6'b010000;
end

6'b010001 : begin
light <= 24'b000000000000110011110011;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b11000000 : begin
        n_state <= 6'b000001;
        end
        8'b00110000 : begin
        n_state <= 6'b000010;
        end
        8'b00000011 : begin
        n_state <= 6'b000100;
        end
        8'b10100000 : begin
        n_state <= 6'b000101;
```

```
        end
        8'b01010000 : begin
        n_state <= 6'b000110;
        end
        default : n_state <= 6'b010001;
endcase
end else n_state <= 6'b010001;
end

6'b010010 : begin
light <= 24'b000000010000001011111100;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b001000;
else n_state <= 6'b010010;
end

6'b010011 : begin
light <= 24'b000000100000000111111100;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000111;
else n_state <= 6'b010011;
end

6'b010100 : begin
light <= 24'b000000000000001111111100;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b11000000 : begin
        n_state <= 6'b000001;
        end
        8'b00110000 : begin
        n_state <= 6'b000010;
        end
```

```
        8'b00001100 : begin
        n_state <= 6'b000011;
        end
        8'b10100000 : begin
        n_state <= 6'b000101;
        end
        8'b01010000 : begin
        n_state <= 6'b000110;
        end
        default : n_state <= 6'b010100;
endcase
end else n_state <= 6'b010100;
end

6'b010101 : begin
light <= 24'b001000001000000001011111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000010;
else n_state <= 6'b010101;
end

6'b010110 : begin
light <= 24'b100000000010000001011111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000001;
else n_state <= 6'b010110;
end

6'b010111 : begin
light <= 24'b000000001010000001011111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b00000101 : begin
```

```
            n_state <= 6'b001000;
            end
            8'b00001010 : begin
            n_state <= 6'b000111;
            end
            8'b01010000 : begin
            n_state <= 6'b000110;
            end
            8'b00000011 : begin
            n_state <= 6'b000100;
            end
            8'b00001100 : begin
            n_state <= 6'b000011;
            end
            default : n_state <= 6'b010111;
endcase
end else n_state <= 6'b010111;
end

6'b011000 : begin
light <= 24'b000100000100000010101111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000010;
else n_state <= 6'b011000;
end

6'b011001 : begin
light <= 24'b010000000001000010101111;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000001;
else n_state <= 6'b011001;
end

6'b011010 : begin
light <= 24'b000000000101000010101111;
g_en1 = 0;
```

```
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b00000101 : begin
        n_state <= 6'b001000;
        end
        8'b00001010 : begin
        n_state <= 6'b000111;
        end
        8'b10100000 : begin
        n_state <= 6'b000101;
        end
        8'b00000011 : begin
        n_state <= 6'b000100;
        end
        8'b00001100 : begin
        n_state <= 6'b000011;
        end
        default : n_state <= 6'b011010;
endcase
end else n_state <= 6'b011010;
end

6'b011011 : begin
light <= 24'b000000100000100011110101;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000100;
else n_state <= 6'b011011;
end

6'b011100 : begin
light <= 24'b000010000000001011110101;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000011;
else n_state <= 6'b011100;
```

```
end

6'b011101 : begin
light <= 24'b000000000000101011110101;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b00000101 : begin
        n_state <= 6'b001000;
        end
        8'b01010000 : begin
        n_state <= 6'b000110;
        end
        8'b10100000 : begin
        n_state <= 6'b000101;
        end
        8'b00110000 : begin
        n_state <= 6'b000010;
        end
        8'b11000000 : begin
        n_state <= 6'b000001;
        end
        default : n_state <= 6'b011101;
endcase
end else n_state <= 6'b011101;
end

6'b011110 : begin
light <= 24'b000000010000010011111010;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000100;
else n_state <= 6'b011110;
end

6'b011111 : begin
light <= 24'b000001000000000111111010;
```

```
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) n_state <= 6'b000011;
else n_state <= 6'b011111;
end

6'b100000 : begin
light <= 24'b000000000000010111111010;
g_en1 = 0;
g_en2 = 0;
y_en = 1;
if(delay5s) begin
        case(signal)
        8'b00001010 : begin
        n_state <= 6'b000111;
        end
        8'b01010000 : begin
        n_state <= 6'b000110;
        end
        8'b10100000 : begin
        n_state <= 6'b000101;
        end
        8'b00110000 : begin
        n_state <= 6'b000010;
        end
        8'b11000000 : begin
        n_state <= 6'b000001;
        end
        default : n_state <= 6'b011101;
endcase
end else n_state <= 6'b100000;
end
default : begin
n_state <= 6'b000101;
light <= 24'b101000000000000001011111;
g_en1 = 1;
g_en2 = 0;
y_en = 0;
end
```

```
endcase
end

always@(*) begin
if(rot<3)
low_traffic <= 1;
else
low_traffic <= 0;
end
always@(*) begin
/*if(!rst2) begin
        timer1 <= 0;
        timer2 <= 0;
        //cd <= 0;
end */

if(car_waiting>0) begin //If there are cars waiting check for traffic of currently green light
        if(!low_traffic) begin //If there isn't low traffic start max_timer
                timer1 <= start_time+rot*5;
                if(timer1>(max_time-10*car_waiting))
                        timer1 <= (max_time-10*car_waiting);

        end else if(low_traffic)
                timer1 <= 10;
end else
                timer1 <= 11'b11111111111; //If no car is waiting timer1 is continuously set to 11
at the start of the clk the clk period is 10 so timer1 never reaches 0
if(car_waiting>0) begin //If there are cars waiting check for traffic of currently green light
        if(!low_traffic) begin //If there isn't low traffic start max_timer
                timer2 <= start_timepl+rot*5;
                if(timer2>(max_time-10*car_waiting))
                        timer2 <= (max_time-10*car_waiting);

        end else if(low_traffic)
                timer2 <= 10;
end else
                timer2 <= 11'b11111111111;
/*if(rst2) begin
        if(cd <= 0) // latch input when previous count is completed. double check <=
      cd <= timer1;
```

```
        else if(cd > 0)
        cd <= cd-2;
                //if(cd = 0) //need some sort of case statement that allows me to return to the main
module once counting has completed

end*/
end

always @(posedge clk)
begin
if(clk_en == 1) begin
 if(g_en1 || g_en2 || y_en)
  count_delay <= count_delay + 1;
  if((count_delay == timer1) && g_en1)
  begin
   optDelay1 = 1;
   optDelay2 = 0;
   delay5s = 0;
   count_delay <= 0;
  end
  else if((count_delay == timer2) && g_en2)
  begin
   optDelay1 = 0;
   optDelay2 = 1;
   delay5s = 0;
   count_delay <= 0;
  end
  else if((count_delay == 4) && y_en)
  begin
   optDelay1 = 0;
   optDelay2 = 0;
   delay5s = 1;
   count_delay <= 0;
  end
  else
  begin
   optDelay1 = 0;
   optDelay2 = 0;
   delay5s = 0;
  end
```

```
 end
end

always @(posedge clk)
begin
 count <= count + 1;
 if(count == 5) // 5 Hz clk, 1 sec enable
  count <= 0;
end
 assign clk_en = count == 5 ? 1: 0; //enable every time count reaches 5

endmodule
```

Appendix B: TLC Testbench

```
`timescale 100ms/1ms
module TLCTB();
reg rst1, /*rst2,*/ clk;
reg [3:0] car_waiting, rot;
reg [7:0] signal;
wire [5:0] state;
wire [23:0] light;

TLC h1(.rst1, /*.rst2,*/ .clk, .car_waiting, .rot, .signal, .state, .light);

integer i = 0;

initial
begin
clk = 0;
forever #1 clk = ~clk;
end

initial
begin
rst1 = 0;
//rst2 = 0;
#2
rst1 = 1;
//rst2 = 1;
rot = 4'b0100;
```

```
car_waiting = 4'b0010;
signal = 8'b10100000;
#2
car_waiting = 4'b0011;
signal = 8'b10100000;
#2
signal = 8'b11000000;
#2
signal = 8'b00110000;
#100
signal = 8'b01010000;
rot = 4'b1000;
car_waiting = 4'b1101;
#10000
signal = 8'b00001010;
#300
signal = 8'b00000011;
for(i = 0; i<100; i =i+1) begin
        #10000
        car_waiting = $urandom%15;
        rot = $urandom%15;
        signal = $urandom%255;
end
end
endmodule
```

Appendix C: Timer Module

```
// (Was added to main TLC module, but was originally by itself and was testbenched by itself)

`timescale 100ms/1ms
module timer_pr(
//input [7:0] sense,
input [3:0] car_waiting, rot,
input clk, //reset, //rot is rate of traffic
output reg [10:0] timer1, timer2//, cd //cd is countdown
);
parameter start_time = 10;
parameter start_timepl = 30;
parameter max_time = 120;
reg low_traffic, g_en1 = 0, g_en2 = 0, y_en = 0, optDelay1 = 0, optDelay2 = 0, delay5s = 0;
```

```
reg [3:0] count;
reg [10:0] count_delay;
wire clk_en;
//reg [7:0] p_sense;
//Trying to increase rot whenever a car goes through the intersection and decrease it whenever
one doesn't
/*always@(negedge clk) begin
p_sense <= sense;
end
always@(posedge clk)begin
if(p_sense == sense)
        rot = rot +1;
else
        rot = rot -1;
end*/
always@(*) begin
if(rot<3)
low_traffic <= 1;
else
low_traffic <= 0;
end
always@(*) begin
/*if(!reset) begin
        timer1 <= 0;
        cd <= 0;
end
*/
if(car_waiting>0) begin //If there are cars waiting check for traffic of currently green light
        if(!low_traffic) begin //If there isn't low traffic start max_timer
                timer1 <= start_time+rot*5;
                if(timer1>(max_time-10*car_waiting))
                        timer1 <= (max_time-10*car_waiting);

        end else if(low_traffic)//If traffic is low return to main module so that light can be
changed
                timer1 <= 10;
end else
                timer1 <= 11'b11111111111;
if(car_waiting>0) begin //If there are cars waiting check for traffic of currently green light
        if(!low_traffic) begin //If there isn't low traffic start max_timer
```

```
                timer2 <= start_timepl+rot*5;
                if(timer2>(max_time-10*car_waiting))
                        timer2 <= (max_time-10*car_waiting);

        end else if(low_traffic)//If traffic is low return to main module so that light can be
changed
                timer2 <= 10;
end else
                timer2 <= 11'b11111111111; //If no car is waiting timer1 is continuously set to 11
at the start of the clk the clk period is 10 so timer1 never reaches 0
/*if(reset) begin
        if(cd <= 0) // latch input when previous count is completed. double check <=
      cd <= timer1;
        else if(cd > 0)
        cd <= cd-2;
                //if(cd = 0) //need some sort of case statement that allows me to return to the main
module once counting has completed

end*/
end

always @(posedge clk)
begin
if(clk_en == 1) begin
 if(g_en1 || g_en2 || y_en)
  count_delay <= count_delay + 1;
  if((count_delay == timer1-1) && g_en1)
  begin
   optDelay1 = 1;
   optDelay2 = 0;
   delay5s = 0;
   count_delay <= 0;
  end
  else if((count_delay == timer2-1) && g_en2)
  begin
   optDelay1 = 0;
   optDelay2 = 1;
   delay5s = 0;
   count_delay <= 0;
  end
```

```
  else if((count_delay == 5) && y_en)
  begin
   optDelay1 = 0;
   optDelay2 = 0;
   delay5s = 1;
   count_delay <= 0;
  end
  else
  begin
   optDelay1 = 0;
   optDelay2 = 0;
   delay5s = 0;
  end
 end
end

always @(posedge clk)
begin
 count <=count + 1;
 if(count == 5) // 5 Hz clk, 1 sec enable
  count <= 0;
end
 assign clk_en = count == 5 ? 1: 0; //enable every time count reaches 10

endmodule
```

Appendix D: Timer Testbench

```
//Cars go through Intersection in .2727 seconds on average
`timescale 100ms/1ms
module timer_prTB();

reg [3:0] car_waiting, rot;
reg clk;//, reset; //rot is rate of traffic
wire [10:0] timer1, timer2;//, cd;

timer_pr h1(.car_waiting, .rot, .clk, /*.reset,*/ .timer1, .timer2);//, .cd);

initial
begin
clk = 0;
```

```
forever #1 clk = ~clk;
end

initial
begin
car_waiting = 4'b0100;
rot = 4'b0001;
#5
rot = 4'b1000;
//#2 reset = 1'b0;
end
endmodule
```