Traffic Light Controller

ECE 276 Fall 2016

List of Authors (Jason Calabro, Hamid Nasrollahzadeh, Yupei Liang, Ryan Cohen)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: jmcalabro@oakland.edu, nasrollahzadeh@oakland.edu, yupeiliang@oakland.edu, rjcohen@oakland.edu

Abstract

The purpose of this project is to create a traffic light controller using an FPGA board. The requirements of this project included using VHDL code in the Xilinx software program to write the functionality, and have control over the timing of the lights. The team decided to connect the FPGA board to a breadboard, in order for a more organized demonstration of the traffic light. The team created a six-state finite state machine, and controlled the timing by staying in each state for different periods of time. The most difficult part of this project was figuring out how to control the timing, or the clock inside of the VHDL code. The team found the simplest way to create this project was by using a finite state machine, pulse generator, and counter. Our team recommends coming up with a plan before implementation, or coding. By spending a week or two deciding how we wanted to implement the design, instead of making design decisions as we implemented, saved us a large amount of time and made the project flow much smoother. In conclusion, the traffic light controller is a great final project because it allowed our group to utilize many of the ECE 278 concepts and our team took a lot of pride in being able to create something that is such an important part of society.

I. INTRODUCTION

This report is intended to detail the design and implementation of our traffic light controller. The decision of a project idea, the planning of project, and putting the plan into action will all be covered in this report.

Traffic light controllers play a major role in American society, and unless traffic circles (roundabouts) replace all of the intersections in America, traffic lights will continue to play a critical role in American society for many years to come. In a nutshell, that statement is why our team decided on creating a traffic light. Any time a person or group of people creates something from scratch, there is a sense of pride in the creation, but creating something that actually has use to people we believed is just as important. Our team also thought it was important to fully understand what we would be creating for our project. Many applications of FPGA boards are unknown to the average undergraduate engineering student, but we felt a traffic light was easy enough for all group members to understand and rally behind the simplicity of the idea. This is the main reason behind the selection of the traffic light controller.

The core of this project is centered around the idea of building a finite state machine, and the amount time spent in each state depending on the counter that was built. Everything used in this project was covered in our course, and the blueprint for almost all of the modules, which include finite state machine, pulse generator, counter, multiplexer, and a constraint file were provided by Dr. Llamocca's webpage or the courses Moodle page. The challenge involved the team trying to figure out how to make the appropriate modifications to these blueprints, and then combine these modified blueprints into a top-level file in order to produce a working traffic light. The team also learned how to create a test bench specified for this project.

The application for this project is simply to control the flow of traffic via a traffic light signal. The team created a traffic light for a four -way intersection. The four-way traffic light does not include a state for a turn lane, or blinking red for yielded turns, but does include a state for blinking red and blinking yellow lights in order to symbolize night time where there is less traffic flow.

II. METHODOLOGY

A. Finite State Machine

A finite state machine is a machine (not a physical machine, but in terms of coding) that changes its "state" based on the input it receives, or the amount of time spent in the state. In the case of our traffic light controller, our finite state machine will change states based upon how much time it has spent in its current state. For example, for any single red light, when the state machine detects it has spent enough time in a certain state (in this case, the state of being red), one of the lights will change to green upon entering the next state. Shown below is a partial code for our finite state machine. State machines are made up of if- when statements, and our machine specifically uses if-when statement while comparing between two values to determine the current state. The variable "seconds", "green", and "three_sec" are all four bit vectors. In the code below, if the

binary value of "seconds" is greater than its compared value, the machine will move into the next state, otherwise, the machine stays in the current state (light stays the same color). Since the counter is always counting, "seconds" will always end up greater than it's compared value and the state will eventually always change.

```
when S1 =>
        if seconds < Green then
                y <= S1;
                        seconds <= seconds + 1;
        else
                y <= S2;
                seconds <= (others => '0');
        end if;

when S2 =>
        if seconds < three_sec then
                y <= S2;
                        seconds <= seconds +1;
        else
                y <= S3;
                seconds <= (others => '0');
```

Our team determined how long each state will be by using a modified pulse generator and a counter. As with any traffic light, more time is spent in yellow and green states than yellow states. For our project, our lights spend seven to fifteen seconds in the green states, three seconds in yellow state, and two seconds in the red/red state (before one turns green). Below is a simplified state table to help further understand our traffic light controller and finite state machines.

Table 1. Simplified State Table for Traffic Light Controller

State	Road 1	Road 2
S1	Green	Red
S2	Yellow	Red
S3	Red	Red
S4	Red	Green
S5	Red	Yellow
S6	Red	Red
S7	Yellow (flash)	Red (flash)

B. Pulse Generator

In this project, we incorporated a pulse generator into our top- level design. The job of our pulse generator is simply to divide the clock from 100 MHz, which is standard for this type of FPGA board, into one second intervals. These one second intervals are what is used to increment the finite state machine. Below are the vectors used in the finite state machine which the pulse generator helps count to.

```
constant ten_sec: std_logic_vector (3 downto 0):= "1010";
constant three_sec: std_logic_vector (3 downto 0):= "0011";
constant two_sec: std_logic_vector (3 downto 0):= "0010";
constant one_sec: std_logic_vector (3 downto 0):= "0001";
constant seven_sec: std_logic_vector (3 downto 0):= "0111";
constant fifteen_sec: std_logic_vector (3 downto 0):= "1111";
constant six_sec: std_logic_vector (3 downto 0):= "0110";
```

C. Counter

In order to simulate night time and demonstrate a flashing yellow and red traffic light state, our team created a counter. To reach the flashing state, the counter needs to be incremented a certain number of times. The counter would increment each time the state machine would pass through S6. When the state machine detects the counter has reached the predetermined value , meaning the state machine has cycled through S1-S6, the state machine automatically moves into the flashing state of S7.

D. Multiplexer

The team implemented a 2-1 multiplexer to further control the timing of the traffic light. Depending on how busy traffic is, the input of the multiplexer can be changed which will change how long the lights stay in certain states. For example, if the input (sel) on the multiplexer is "1", the green signal of Road 1 will stay in its state for 15 seconds. If the select input to the multiplexer is "0", it will stay green for 10 seconds.

E. Top- Level Design

A top- level design is put together to combine all of the components into one file in the Xilinx software. The top- level design (along with a constraint file) is where our code gets connected to our hardware, including our FPGA board, Arduino board, and bread board with LED lights. In this part of the project, our team had to give the traffic lights state-dependent binary values and connect them to various pins on the FPGA board. Below is an example of how the finite state machine is directly related to the lights being turned on our off.

```
when S1 => lights <= "011110";
when S2 => lights <= "101110";
when S3 => lights <= "110110";
when S4 => lights <= "110011";
when S5 => lights <= "110101";
when S6 => if (seconds = one_sec) then flash_count <= '1'; end if; lights <= "110110";
when S7 => lights <= "101110"; sig_sclr <= '1';
when S8 => lights <= "111111";
when S9 => lights <= "101110";
when S10 => lights <= "111111";
when S11 => lights <= "101110";
when S12 => lights <= "111111";
```

As shown in the partial code above, the 'lights' variable is assigned a six- digit value depending on which state the traffic lights are in. The six digits represent red, green, and yellow values for two traffic lights (we only used six digits even though there are four traffic lights because any two traffic lights will always be identical to the other two). Inverse logic was used here, so a '0' assigned meant the LED was on. It is possible to follow the logic of the states

by tracking the 0's in the code above. In S1, the leftmost and rightmost digits are 0's, meaning those lights are on (two lights are on according to the code, but four lights are actually on). Notice in S2, the leftmost digit turns off while the rightmost digit stays 0. By following the state table, this can only mean the leftmost digit is a green LED, the second to leftmost is a yellow LED, and the rightmost digit is a red LED.

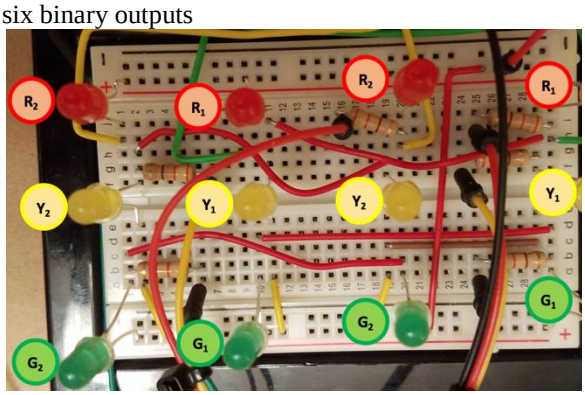
III. EXPERIMENTAL SETUP

A. *Hardware*

NEXYS 4 DDR Artix- 7 FPGA board: VHDL code from Xilinx software was uploaded to this board. Required.

Arduino Uno: microcontroller used only to provide power (5V) to the traffic lights (LEDs).

Bread board: used as the base for our traffic light. Housed all 12 RGY LEDs and current limiting resistors. Wires ran from the FPGA and Arduino pins to the bread board in order to supply power and link the correct pins to their appropriate lights, based on the constraint file. Shown below are the 12 LED lights on the bread board ready for demonstration. By following the red wires running horizontal in the circuit, it is revealed that power was distributed to two signals on the bread board instead of one, this is how we were able to control 12 LEDs with only six binary outputs



B. *Software*

As required for ECE278, our team used the Xilinx software program. In this program, we coded using a language called VHDL. This is the language the class was taught throughout the duration of the course, and is the only feasible language to use at this point to complete this project. The blueprint coding for our components, found on Dr. Llamocca's website, were written in VHDL. The final project will be executed using the Xilinx software.

C. *Test Bench*

Before we assembled the hardware, we created a test bench to ensure our code worked properly. A test bench is an easy way to verify coding without having to waste the time of uploading the code to your hardware first, in case your code does not work. The test bench for our traffic light controller was fairly simple since we only have two different test scenarios (from the multiplexer) and a reset button. Shown below is code from our test bench. This code reflects a test of the reset button and both multiplexer selections. Both reset states and multiplexer selections are separated by clock intervals. Before simulating the g0 section of the VHDL code must be commented out and the "pulse_clock <= clock" line must be uncommented.

```
wait for clock_period*10;

-- insert stimulus here

        resetn <= '0';

        wait for clock_period*2;

        resetn <= '1';

        sel <= "0";

        --wait for clock_period*50;

        --sel <= "1";
```

IV. RESULTS

All in all, our project was successful and functioned exactly how it was designed to. Our results on the bread board are explained perfectly by our code. The team felt our code was as simple as it could possibly be, and our traffic light functions as the code intends. Here is a link to a video of our project working: https://drive.google.com/file/d/0B6CzOACe_c16RUFWd1RvQWxMMHc/view?ts=583673c9

You must sign in to Google to view this video.

In class, we have analyzed many state diagrams in lecture, quizzes, and homework. The state diagram for this project is much simpler than the ones analyzed in class. This is appropriate given it is much harder to create and implement a diagram versus analyzing a ready-made one. The majority of this project was reliant on the finite state machine, which was learned at the tail end of this course. The benefit of learning this material at the end of this course was that coincides with the timing of the building of our project, and the idea of state machines was fresh in our

mind. By learning this material when we did, we saved a lot of time and possibly confusion relating to the project.

CONCLUSIONS

The main take away from this project for our team is that state machines are great organizers of seemingly complex issues. The hardest part about this project was the understanding of the clock in the Xilinx software. This could possibly be from the fact we did not start implementing clocks until later in our labs, and there was less lecture time devoted to understanding and calculating the clock speed, compared to ideas like finite state machines or counters. Our team learned how to connect our code to the output pins on the Artix- 7 FPGA board, and then relay those pins to the bread board through wires. Connecting to the FPGA pins instead of switches was a new concept for our group, but most of our team has previous knowledge from other classes to know how to interface with various FPGA or microprocessor development boards.

In the future, a traffic light controller could be made with more functionality to accurately represent many of the traffic lights we see on a daily basis. These functions would include: representation of a turn lane, blinking red light for yielded left turns, and representation for a pedestrian cross walk. However, the current project is sufficient to represent many simple traffic signals in society today, and the representation of such signals was the main goal for this project.

REFERENCES

[1] VHDL Traffic Light Controller. (2015, November). Retrieved from: http://stackoverflow.com/questions/27783576/vhdl-traffic-light-controller