Traffic Light Controller

Authors: Kyle Alspach, Ryan Kelly
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: kalspach@oakland.edu, ryankelly@oakland.edu

***Abstract*— The purpose of this project is to implement the student's current understanding of VHDL and apply it to create a simple traffic light controller. The controller will have two modes, day and night, to simulate an actual four way intersection seen in real life. Through the implementation of processes such as counters, and finite state machines, the desired simulation of a four way traffic light will be achieved.**

I. INTRODUCTION

Traffic lights serve an important purpose in everyday life. With the most popular form of transportation being automobiles, traffic lights are vital in order to keep traffic flowing. The goal of this system is to ensure that traffic moves quickly and safely as to prevent backups. Additionally, traffic lights need to be able to adapt to changes in the behavior of traffic. For instance, there is generally less traffic at night compared to the daytime. As a result of this, the controller needs to adapt to keep traffic moving at a busy intersection. Many of the topics learned in class were implemented for this project.

II. METHODOLOGY

The goal of this project was achieved through the underlying design. In order to keep time of the traffic lights, each light needs to stay lit for an exact amount of seconds to allow motorists enough time to clear an intersection. The counters are used to determine how many seconds have elapsed for each green, yellow and red light. These counters are a modified version of Prof. Llamocca's [1]. In order to control the switching of each color of the traffic light, a finite state machine was built for each mode. This controlled the state of each traffic light to prevent lights from changing at will. This will read from each counter depending on the current state and move to the next one once each counter has reached its limit. For the implementation of both day and night modes, five counters were needed and two finite state machines, one finite state machine is used to coordinate the day time mode and, the other is used to coordinate the night time mode.

Once the main system was achieved we implemented a multiplexor controlled by a switch that turns the default daytime mode off, and turns on the night time mode. The night mode will act like any other night time traffic lights and flash yellow for the busier street, and flash red for the slower streets. The block diagram of the overall data path is shown in Figure 1.

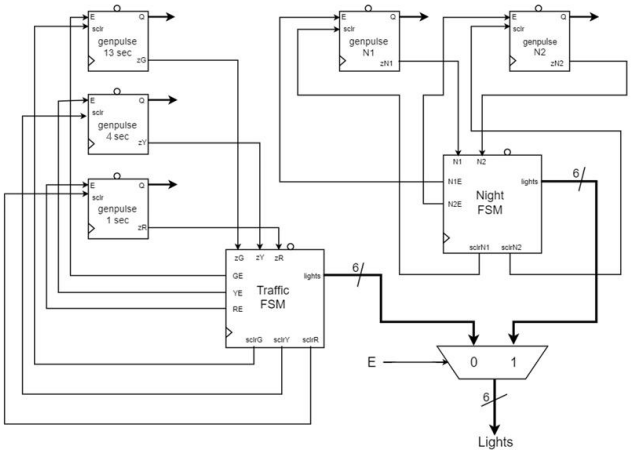

Figure 1 - Block Diagram

The most important part of the design is the finite state machines. They control how the lights are displayed and which counters are enabled. The day mode finite state machine includes six different states. Each state includes a six bit output for the LED display, an enable for the current counter and an asynchronous clear to clear the previous counter.

The first state demonstrates a green light being on for one intersection, and a red light being on for the other intersection. The state machine then looks at the output from the green light counter which is set for thirteen seconds in order to determine whether or not to change state. Since technically these counters are pulse generators, once the counter reaches 13 seconds, a signal is sent back to the FSM enabling it to move onto state two, which switches off the green light and turns on the yellow light while the red light from state one remains on. The FSM then goes through the same process as before for the yellow light which runs for four seconds. Finally after state two, the current state is moved into state three where both red lights are on. This state lasts for one second and then the process cycles over again only this time with the opposite lights to simulate changing of the direction of allowed traffic. The machine repeats the same process for the different direction then

ultimately goes back to state one. Figure 2 is a diagram of this algorithmic state machine.

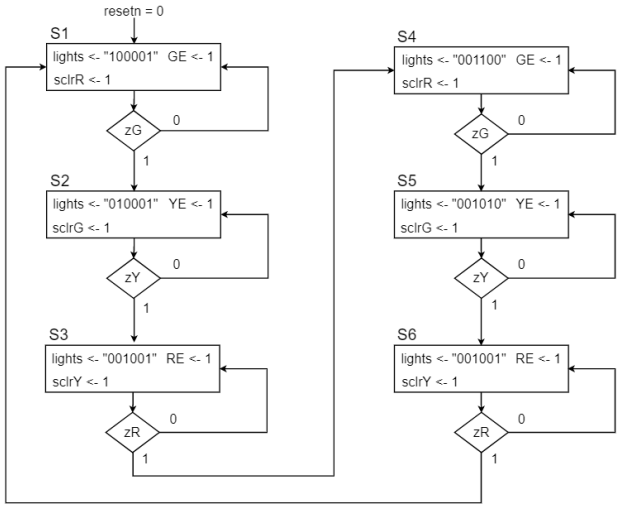

Figure 2 – Day Mode ASM State Diagram

In a typical night time operation of a traffic light, a main road has a blinking yellow and a side road will have a blinking red. A blinking yellow allows for all traffic to proceed through the intersection while a blinking red requires you to stop. This FSM is used to create a blinking action of two lights with the use of only two states. In state one, none of the lights are on, and in state two one yellow light, and one red light is turned on. Two counters were used in the same fashion of the day time FSM but the output of the lights were configured in a way to simulate the blinking one would find at an actual intersection at night. Figure 3 shown below represents this algorithmic state machine.

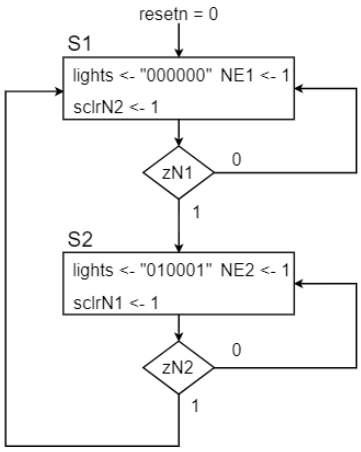

Figure 3 – Night Mode ASM State Diagram

III. Experimental Setup

Now that the method of how the traffic light controller has been laid out, the next step was to build it using VHDL in Vivado. The final code used a top level design to port map all of the counters and FSM's to connect each component, with the counters using a generic map to manipulate the counters to change how man seconds each ran. Once all of the components had been laid out and connected, the circuit was simulated to verify the desired result.

The simulation however had some issues when ran. Typically, simulations are run for very short periods of times which requires little memory. When an attempt to run a simulation with something with a lot of time, it causes too much memory to run or a long time. In order to truly test this, the circuit had to be programmed to the Nexys A7 and tested on the breadboard to see if it function properly or not.

Since most of the testing was required to take place on the external peripheral, a breadboard was hooked up with some wires and LED's. This required the use of 12 LED's, 12 220 ohm resistors. Only six of lights were hooked up through the Jmod headers of the Nexys A7 board but the other six were connected in parallel to bridge of their connection. Six of the LED's were used to simulate a North/South road and the other six were used to simulate an East/West road. Switch 0 was connected to the multiplexor to switch between day and night mode and the CPU_Reset was connected to resetn of the circuit.

IV. Results

Overall, the results ended up being what was originally desired when this project started after several circuit designs and different testing methods. State one runs for 13 seconds, state two runs for four seconds and state three runs for one second with states four, five and six running for the same time respectively. With each state change, each light changes with each corresponding state as well. Night Mode was implemented similarly with it effectively achieving a blinking action for the yellow and red lights.

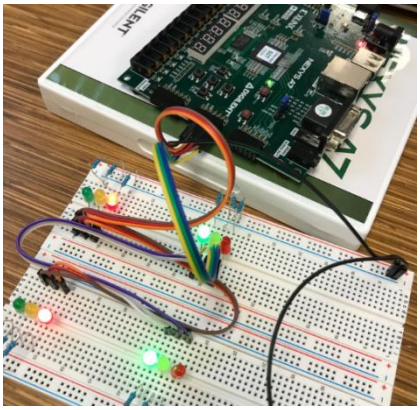

Figure 4 – Results

CONCLUSIONS

This project allowed the team to experience what it is like to be tasked with a real world problem and apply digital design knowledge and solve it. While it seems as though if traffic lights have all been figured out by now, there are still issues and improvements out there which the team experienced when going through this project.

Being able to go through the process of designing a digital system and going through different iterations of each design to achieve a final one really opens the eyes of what it is like to do design like this for a living. Experiencing firsthand how initial designs can be changed so much by the end gave the team a lot of knowledge and real world experience.

This project also speaks to how much more a project can be improved by adding more and more. The main improvement to any traffic light is sensors and while this project didn't use any, it would have made it multitudes better. Sensors can help the flow of traffic and prevent unnecessary accidents. In the end however, this project achieved two simple ideas and was implemented correctly.

REFERENCES

[1] Llamocca, Daniel. VHDL Coding for FPGAs, http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html