4 Way Traffic Light Controller

Traffic Light Controller with an FPGA

Sam Narusch, Richard Pinto, Christina Salama, Adrian Sinishtaj
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-mails: adriansinishtaj@oakland.edu, cmsalama@oakland.edu, richardpinto@oakland.edu, snarusch@oakland.edu

***Abstract*— The importance of general driving safety for everyday commute and travel leads all the way up to traffic lights, and depending on them to operate properly. Comprehending how a four-way traffic light controller works and replicating such in this project was the ultimate objective, as we enforced the knowledge learned from this course to achieve this goal. It was also found that certain components were necessary to make this controller fully operable, such as the implementation of a gen-pulse in addition to a basic counter. Through this, the implementation of the code onto the Nexys A7 FPGA board was much more efficient, and the desired results were achieved. To better streamline the process of completing this project, it is strongly recommended to remember when it is necessary to implement a gen-pulse, and to avoid at all costs running a simulation if a gen-pulse is used, to avoid any unwanted complications with the operating computer.**

I. INTRODUCTION

The goal of this project is to create a four-way traffic light controller, using a PCB circuit connected to the FPGA board. This project will implement the knowledge from various topics learned in this course and a variety of components including a finite state machine (FSM), a counter, and a gen-pulse.

We found this project to be fascinating and a perfect opportunity to further explore what was first introduced in this class. Additionally, we recognized that understanding the operations of a traffic light system, along with the different stages each changing light follows, is the optimal way to see how each factor affects the big picture. It is critical to understand how a product works before tackling methods of improving the design and execution of it. Given the importance of a four-way traffic light controller, especially due to the fact that they are integral to human society, we wanted to use what we learned in class and apply our newfound knowledge to dissecting and reworking the controller for our own discernment.

When working on this project, the topics found to be the most important from this course were the sections focusing on the Finite State Machine (FSM), building a counter— and more specifically, exploring how to create and implement a gen-pulse— along with the importance of when these tools are necessary when creating a project. In other aspects, we took it upon ourselves to learn and further explore the Nexys A7 FPGA board past what was taught in class, such as utilizing the headers on the board. Additionally, we decided to construct the circuit on a PCB, with various LEDs and resistors securely soldered to the PCB, in order to replicate the way a four-way traffic light controller operated, giving it a much more polished and clean final look.

II. METHODOLOGY

The plan to design the project began with designing the state machine. The main purpose of the state machine was to create a different state for each combination of lights being turned on, then creating individual states inside the machine to emulate the states of the lights. After the state machine was complete, some kind of input was needed to control the transition of the states.

To control the transition of the states, a counter was used. This counter was created to count to twenty eight, and output different signals depending on the value of the current count. Six different signals were created for this purpose, each becoming high at a different count. These signals would just change the

state in a process that cycles through the state of the traffic lights.

Finally, a module was needed to correlate the counter with real time instead of clock cycles. The pulse generator module is used for exactly this purpose [1]. The pulse generator was calibrated to output a high signal every second, and this signal was used as an input to the counter. This enabled the counter to increment the count every second, instead of every clock tick.

III. EXPERIMENTAL SETUP

The materials needed for this project will include the Nexys A7 board, copper wires, and various LEDs (red, green, and yellow). The design will consist of a counter that utilizes a pulse generator that is input into a Mealy FSM. The FSM ultimately controls the lights on the PCB board. The pulse generator was set to create a high pulse every one second, which was then used by the counter to increment the count. Six output signals were created to be set to high based on what number the count is at. The six signals, "a, b, c, d, e, f", represent the variables at each stage of the FSM that will determine the next state. Each signal was set to high for a specific count and this signal was fed to the FSM to determine the state of the system.

The FSM that was used consisted of five different states. Starting at state 0, "S0", both the North/South and East/West lights are automatically set to red and the activation of the reset button always reverts the system back to S0. While in S0, the system will wait for 2 seconds, then based on the value of "a" and "d" that is read, it will move to state 1 or state 3. If "a" is a "1", the system will move to state 1. If "a" is "0" and "d" is a "1", the system moves to state 3; otherwise, the system will stay in state 0. In state 1, the system will wait for signal "b" to go high at 10 seconds and then transition to state 2. While in state 2, the system waits until the 14 second mark for signal "c" to go high and then will return to state 0. Once the system is in state 0 for the second time, it will wait until the 16 second mark for signal "d" to go high and then will move to state 3. At state 3, the signal "e" will go high at the 24 second mark and move the system to the final state: state 4. In the final state, the system waits until the 28 second mark for "f" to go high and return to state 0 to complete a full cycle. As seen above, a full cycle for the system takes 28 seconds. In state 0, both lights are red. Then, in state 1, the North/South light is green and the East/West light remains red. In state 2, the North/South light changes to yellow and then goes back to state 0 where both lights are red again. Next in state 3, the East/West light turns green while the North/South light remains red. Lastly, in state 4, the East/West light transitions to yellow before returning back to state 0 to start another cycle. In the test bench file, the system was simulated to run for 28 seconds or one whole cycle.

To implement the design to the external interface, The six top level output variables, (gNS, yNS, rNS, gEW, yEW, rEW) were mapped to the JA and JB Pmod header pins on the FPGA board. The JA headers were used for signals rEW, rNS, and yEW while the JB headers were used for gNS, yNS, and gEW. From these pins, wires were connected to the corresponding LEDs on the PCB board to control the lights.

IV. RESULTS

What was found after working on this project and uploading the code to the Nexys A7 board was that there can be a lot of frustration involved in a project like this. While developing the code, there were a few instances when a single line was missing that would make the simulation completely non-functioning. Additionally, there were times when a single line would only slightly affect the simulation, but it was still not functional enough to be a proper stoplight. For example, since the counter depends on the output of the pulse generator, we were unsure if it also needed a dependence on the clock. This created an issue where the counter would increment on the rising and falling edges of the clock, instead of only on the rising edge. This created an issue that would make the state machine do an entire cycle of the count, while staying in state zero. Once this was discovered, it was an easy fix.

After the code was completed, a PCB was soldered together to display the outputs. Initially, the PCB did not function properly. Unsure of whether this was an issue in VHDL or on the PCB, a DC voltage source was used to probe the LED's. Since some of them functioned and some did not, examination of the PCB began. It was discovered that there was a bad joint with a resistor connected to the ground pin.

CONCLUSIONS

While working on this project, one issue arose when implementing the code onto the board. The issue, upon later discovery, was that our first design did not include a gen-pulse signal. Without the gen pulse signal, the counter could not count to a high enough value for the lights on the traffic light to visually

show the different states. This issue was then quickly resolved once the gen-pulse was introduced to the project and made so that it had an output after every second, and using that output for the counter. The main take away from this project was being able to implement all the ideas learned in class to create one project. The expectation we had as a group was high, as we strived to design and implement a fully functioning traffic light, with our next goal being how to figure out how to implement one in the most concise and simple way, as we wanted to imitate a real world situation. For the audio-visual portion of our project, we created a sample of our traffic light to better illustrate the results for our project. As shown in the figure below (Figure 1), 12 LEDs were placed in series along with resistors.

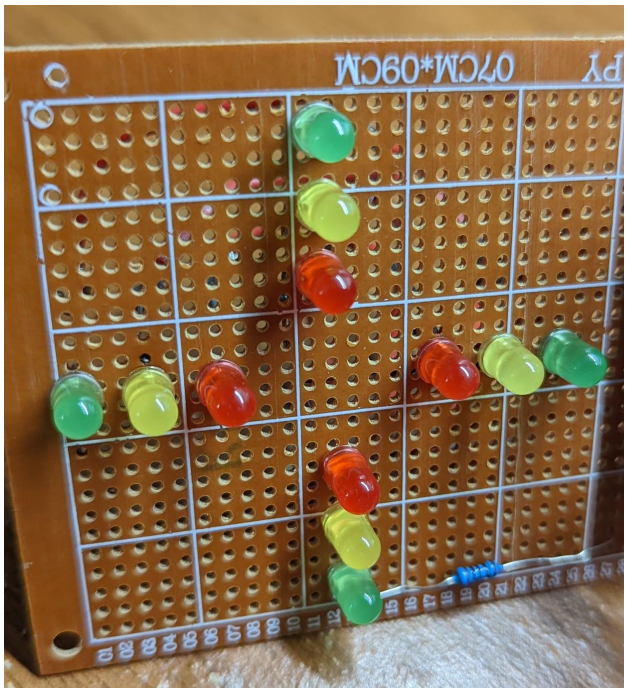

Figure 1: LEDs placed in series on the PCB

REFERENCES

[1] Llamocca, Daniel. “Digital Logic Design.” ECE 2700, Oakland University, Rochester, MI, December 2021.