Traffic Light Controller

List of Authors (Nathan Page, Joseph Nichols, Alan Akm, Stavro Alshmani)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: page2@oakland.edu, jnichols2@oakland.edu, aakm@oakland.edu , salshmani@oakland.edu

***Abstract*— The purpose of this project is to make a 4 way traffic light controller based on a Finite State Machine. The circuit is implemented on a Nexys 4 DDR FPGA and a breadboard to demonstrate a real application. Major findings include the process of designing a digital circuit and transforming a block diagram into a real circuit, as well as modifying clock speeds on an FPGA. Conclusions include that FSMs can largely simplify a VHDL circuit as compared to using other components to achieve the same goal. Also, the type of hardware used will depend on the application. Creating a complex circuit to control the exact timing of a quick clock is much more difficult than implementing the same timing on a programmable microcontroller, such as Arduino. Nonetheless, FSM-Based system design is a replicable and efficient method of modeling a digital system which serves as a fundamental tool in digital design.**

I. Introduction

This report will explore the design and implementation of a traffic light controller on an FPGA. Traffic lights control the flow of traffic, and ultimately, get people where they need to go safely and efficiently. Since traffic lights operate in a finite number of states, such as stop and go, the system can be modeled with a Finite State Machine. This is one demonstration of the Finite State Machine, a powerful design tool for modeling digital systems. The advantage of the Finite State Machine is that it suits a traffic light well because traffic lights repeat the same pattern. The traffic light controller is based on a VHDL implementation of a state diagram, which lights external lights. Overall, the traffic light controller is an FSM-based digital system which offers a reliable method of safe traffic flow.

II. Methodology

High-Level Concept: Before determining specific requirements and design strategies, a high level design was created. This included an outline of how the intersection would operate. We decided on a 4-Way, 12 LED Intersection. 3 LEDs for North, (Green, Yellow, Red) another 3 LEDs for East, 3 for South, and 3 for West. From here, we knew that the traffic lights would need a timing element, and that they would be triggered in alternating cycles, based on how much time has passed. In addition to normal operation, a night mode option would be needed, with the LEDs blinking yellow and red at intersection pairs. [1]

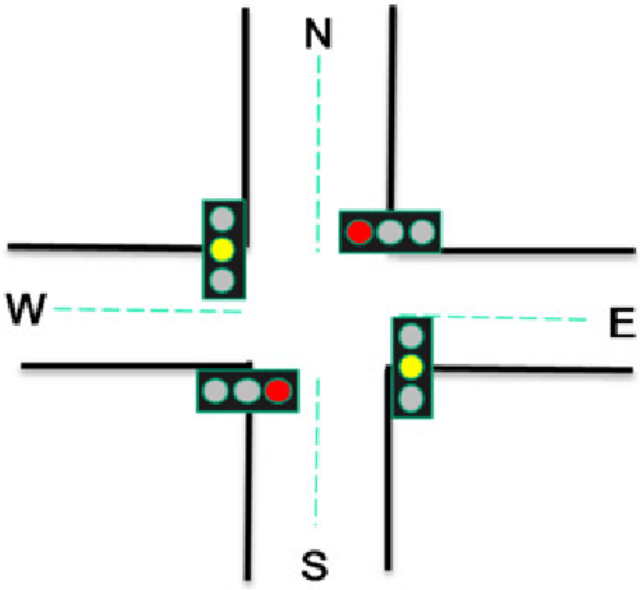


Figure 1: Traffic Light Intersection

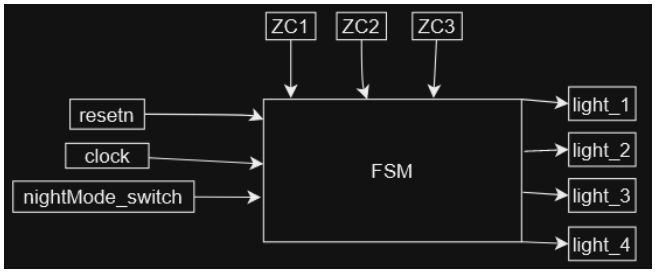


Figure 2: Top Level Block Diagram

Clock Timing: The next large task which needed to be addressed was replicating realistic times. The FPGA has a 10ns clock, meaning in order for 1 second to pass, 100-million clock periods would need to pass. In this stage, various approaches were weighed, such as implementing a simple software workaround with built-in commands, or by incrementing a register. We ended up implementing a real-time counter by using a Modulo-N (generic pulse generator) counter which outputs a logic high after counting to N clock cycles of 10 nanoseconds each. This route was chosen due to simplicity and available documentation on

this approach. When implementing this clock, the code for my_genpulse was taken from [2].

The pulse generated after N clock cycles will be used to signal that the state should be changed.

Finite State Machine: After obtaining a reliable clock and having an outline for the inputs, outputs, and operation of the system, a more specific approach was needed. This is where the FSM came into play. With the LEDs as FSM outputs, and the counter outputs as inputs to the FSM, the Algorithmic State Machine (ASM) chart was derived. The ASM chart depicts how the output changes as different states are entered, as well as how the inputs affect those states.

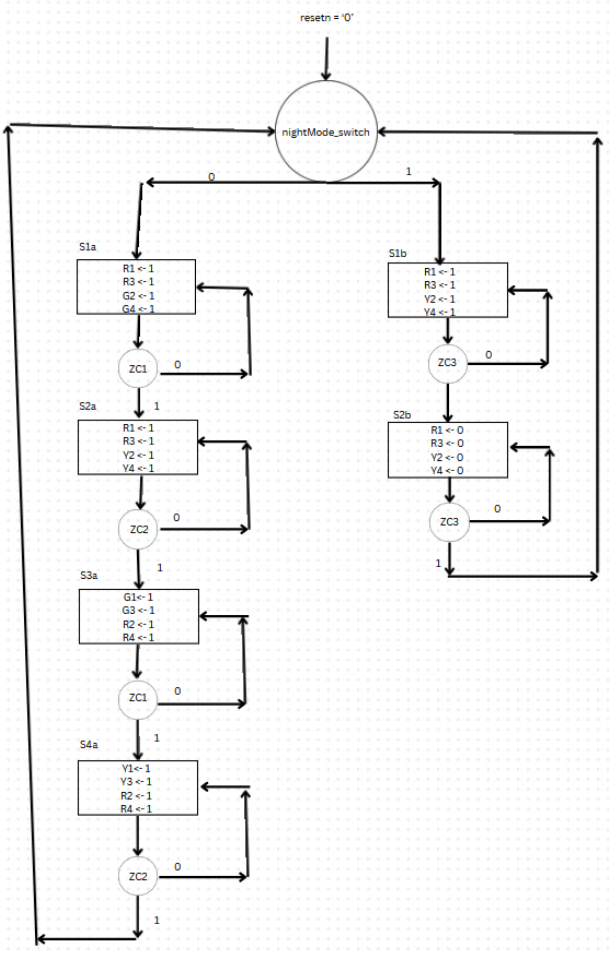


Figure 3: ASM Chart

This FSM works by using three different counters. ZC1 is a 3 second counter, ZC2 is a 1 second counter, and ZC3 is a 0.25 second counter. Starting from the beginning, first the night mode switch is checked for its value, if it is zero then the process continues to daytime mode. When in state S1a, the FSM waits for the 3 second counter to emit a 1 in order to move on to state S2a. A 3 second counter is used there because lights 2 & 4 are green and we must wait 3 seconds for them to turn yellow. In state S2a, lights 2 & 4 now have changed to yellow, and to proceed to state S3a, the 1 second counter must emit a 1. The reason a 1 second counter is used there is because the yellow light is shorter than the green light so its duration has been set to 1 second. Once in state S3a, lights 2 & 4 now switch to red, and lights 1 & 3 will switch to green. If one follows the FSM diagram further, one will observe a very similar process will occur for lights 1 & 3 as compared to the previous process for lights 2 & 4.

For night mode, only one 0.25 second counter is used. Lights 1 & 3 will flash red while lights 2 & 4 will flash yellow. These pairs of lights are perpendicular to each other. Using a 0.25 second counter, the lights will flash twice per second.

Behavioral Simulation:

Figure 4: Behavioral Simulation

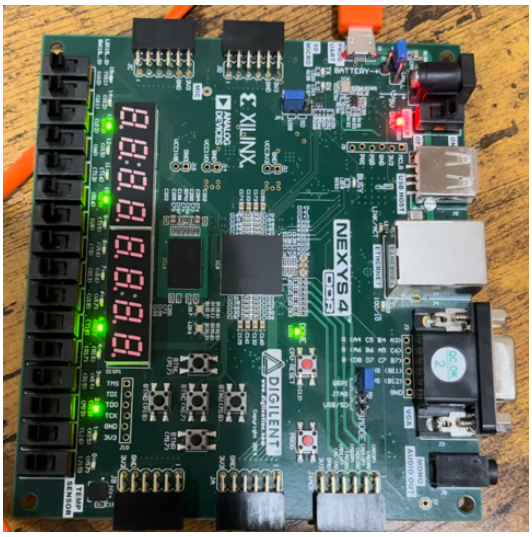
The test bench runs a loop that switches the circuit between night mode and day mode every 2 us. The clock values were made smaller when running the behavioral simulation so that everything would fit in a small time frame.

III. Experimental Setup

One thing that helped us know if we were on the right track was comparing our simulation to a real-life traffic light intersection.

Before connecting the fpga to the breadboard, we used the on board leds to simulate the traffic light.

Figure 5: Leds on FPGA



The Nexys 4 DDR board was used with a breadboard that was populated with twelve LED's, three 100 ohm resistors in parallel, and jumper wires. All code was done on Vivado in VHDL. One switch on the Nexys board was used to switch between modes.

Throughout various points in the design stage, the goal was to accomplish one task at a time, instead of many tasks all at once. For instance, before wiring external LEDs, and introducing room for error in properly interfacing the digital circuit with the external circuit, the onboard LEDs were used. In this way, we verified that the clock timing was correct, and that the LEDs would light as expected. From here, it was now a matter of driving LEDs through the FPGA's PMOD ports, rather than the onboard LEDs. With this approach, any errors would be guaranteed to be due to further alterations, which allows for quick troubleshooting.

In our second design setup, we temporarily used white LED's to figure out how to wire everything up. It was decided there would be a unique ground for each LED that would be connected to the board outputs, and the power traces would be wired to resistors. (Active low setup with VCC always present). At first we did not know what resistors were needed, so we took three 1kΩ resistors in parallel, as shown below in Figure 1, to see if it was good enough. Although it worked, we wanted something a little more precise.

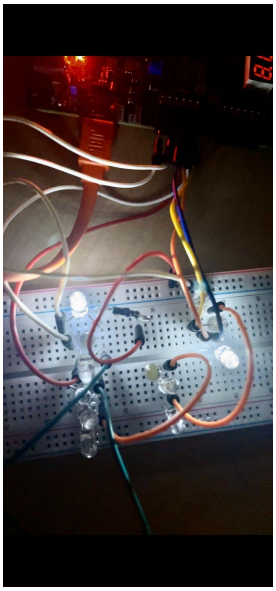


Figure 6: Initial Design Set Up

The reason for using three 100 ohm resistors in parallel is to amount to an equivalent resistance of 33 ohms. We did not have a 30-40 ohm resistor, so we emulated it by taking what we had and making it 33 ohms. The reason for 33 ohms for the entire circuit is because it was decided we wanted 10 mA per LED to generate an adequate brightness. The forward voltage drop of the LED is 2V at a current of 10mA, and the FPGA has an output of 3.3V. With 4 LEDs on at any given instant, the current draw totals to 40mA. Therefore, we can find the resistance by rearranging Ohm's Law as follows:

$R = \Delta V/i = (3.3 - 2)/0.04 = 32.5\Omega$.

Rounding this to the nearest value and using 100Ω resistors due to availability, we obtain the following equivalent resistance:

$$Req = [(1/100) + (1/100) + (1/100)]^{-1} = 33\,\Omega$$

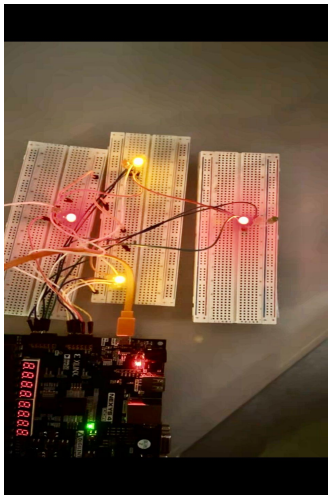


Figure 7: Night Mode Final Design Set Up

We finalized our circuit by adding the red, yellow, and green LED's. Using a switch on the Nexys board, we can switch between night and day mode.

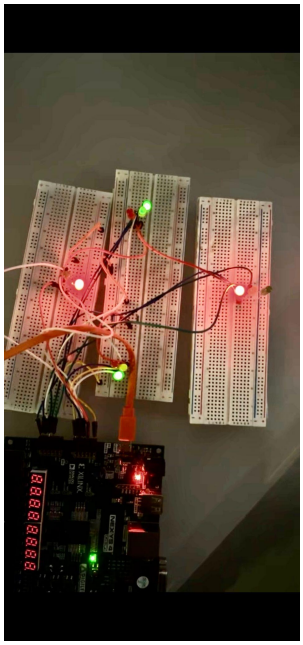


Figure 8: Day Mode Final Design Set Up

IV. Results

The result of the traffic light turned out as expected. Day mode and the transition to night mode was a success. When the switch is off, day mode is enabled and acts as a normal traffic light intersection during the day. When the switch is flipped on, nighttime mode is enabled and North and South LEDs continuously blink red, and East and West LEDs continuously blink yellow.

Conclusions

One major takeaway from this project is the usefulness of FSMs and ASMs in describing and implementing digital systems. In the context of VHDL coding, any digital circuit which has a finite number of states can be composed into a Finite State Machine, which interacts with and controls the datapath circuit operations as needed. This is necessary in larger digital systems where much is occurring at once. It is also very true for all Engineering, that breaking a large system into manageable constituent parts allows each one to be handled more clearly and properly. Lastly, VHDL's powerful conversion of Processes allows an ASM to be transformed into a functioning digital circuit. With all of this put together, the ability to effectively model and implement a digital system is greatly enhanced. Another takeaway is the importance of the ability to work and communicate as a team. Working as a team allowed the workload to be split among different members, which increased the speed at which the project could be accomplished. With some group members focusing more on specific tasks, while having a good understanding of the rest of the system, we were able to effectively specialize in areas of design, and ensure that our designs would be cohesive when they came together. Working as a team and communicating progress, plans, and ideas allowed for the final version of the project to be the best it could be. Lastly, an improvement that could be made would be to implement the project with a programmable microcontroller, rather than an FPGA. This would provide much greater simplicity and flexibility in the circuit. Of course, this simple traffic light does not have much digital circuit "heavy-lifting". It is merely a counter, and so in our specific context, the design cycle could have been massively accelerated by using an altogether different board. Still, it provided a valuable experience and reference if and when the time comes where digital complexity is increased, and a microcontroller can not properly control the system.

References

[1] https://www.researchgate.net/figure/Typical-traffic-light-system-at-a-four-way-intersection_fig4_325530865

[2] https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html