4-Way Traffic Light

To control incoming traffic

Cali Malone, Bianca Muller, Marielly Perez Lugo, Azra Jakupovic
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: calimalone@oakland.edu, bmuller@oakland.edu, mperelugo@oakland.edu, azrajakupovic@oakalnd.edu

***Abstract*—We will be using an Artix-7 Nexys-4 DDR FPGA board to simulate a four-way traffic light at an intersection. The code will be in VHDL language. The inputs will be counter values, while the outputs will be LED lights. This simulation will demonstrate how traffic light controls are conducted in order to maintain safety in daily traffic.**

I. Introduction

For this project we have chosen a 4 way traffic light controller. A traffic light controller plays an important role in controlling the vehicular flow on a busy road, such as the four-way intersection that this traffic light is designed for.

A traffic light controller assigns the right of way to pedestrians and road users by the use of lights in standard colors: green for go, yellow for slow down and prepare to stop, and red for a complete stop. Each color of light lasts for a different amount of time. In this project, there will be no right of way for pedestrians as this is intended to be a light for a busy, high speed intersection without crosswalks. The goal for this four-way traffic light controller is to follow the methodology of coding using VHDL uploaded to an Artix-7 DDR FPGA board, to create a successful model of a four-way traffic controller.

II. Methodology

VHDL language will be used in order to successfully program the Artix-7 Nexys-4 DDR FPGA board. The power ports, and LED lights will be used to simulate the four way traffic light. Specific registers such as the counter register will be utilized to calculate the time in which the color of the LED will change. The state machine that will be made to demonstrate the functionality of a four way traffic light intersection will have its series of sequence color changes, dependent on clock ticks. Grouping of the states can be shown in Figure 1 and Table 2. An example of one of the state processes would be G1G is a green light for a car going north on the main road, or as in group one, whereas G2G is a green light for a car going south from the side road and in group 2. There will also be a clock variable to add timing to the program in order to change the light sequence, otherwise known as the states, that will occur. This will depend on how much time has passed and whether or not the "count" for the traffic lights has reached a "HIGH" level. There will also be a resend signal to restart the count back to 0. This is demonstrated in Table 1.

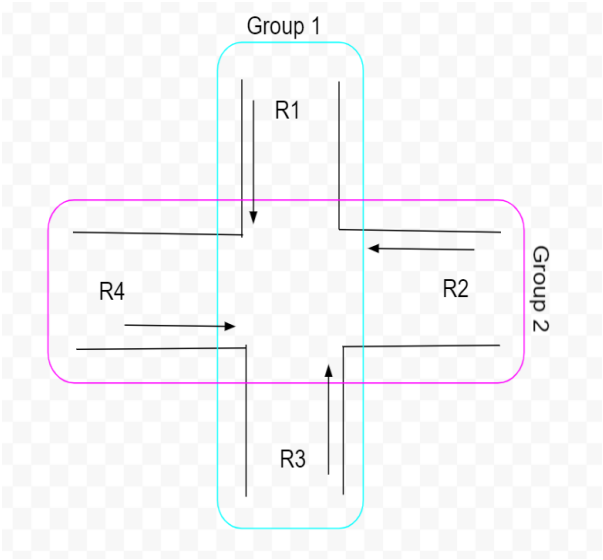

Figure 1: Four Way-Traffic Light Controller Scenario.

Outputs	Group	Signal	Description
Road 1 Road 3	1	G1G	Group 1 Green Light
		G1Y	Group 1 Yellow Light
		G1R	Group 1 Red Light
Road 2 Road 4	2	G2G	Group 2 Green Light
		G2Y	Group 2 Yellow Light
		G2R	Group 2 Red Light

Table 1: Output table with group number and signal description.

Name	Inputs	Outputs
Counter 1	Ea	Za
	Sclra	
Counter 2	Eb	Zb
	Sclrb	
Counter 3	Ec	Zc
	Sclrc	
FSM	Q1, Q2, Q3	G1, G2, RGBs

Table 2: Inputs and Outputs of each component.

A. State Table/Excitation Table

A state table was necessary to easily visualize the inputs (za, zb, and zc), present states, the next states, and outputs for this project. The table represents all possible states, as well as the correct next state when per input. It also demonstrates the color a traffic signal will be for each group according to the present state. The outputs are represented using the signals shown in Table 2.

Input (za | zb | zc)			Present State	Next State	Output	
1	X	X	S1	S2	G1G	G2R
X	1	X	S2	S3	G1Y	G2R
X	X	1	S3	S4	G1R	G2R
1	X	X	S4	S5	G1R	G2G
X	1	X	S5	S3	G1R	G2Y
0	X	X	S1	S1	G1G	G2R
X	0	X	S2	S2	G1Y	G2R
X	X	0	S3	S3	G1R	G2R
0	X	X	S4	S4	G1R	G2G
X	0	X	S5	S5	G1R	G2Y

Table 3: State Table.

The excitation table mirrors the state table with state 1 = 000, state 2 = 001, state 3 = 010, state 4 = 011, and state 5 = 100. The output is split for group 1 and group 2 and the LED colors are listed as RGB values where green = 010, yellow = 110, and red = 100.

Input (za | zb | zc)			Present State	Next State	Output (G1 | G2)	
1	X	X	000	001	010	100
X	1	X	001	010	110	100
X	X	1	010	011	100	100
1	X	X	011	100	100	010
X	1	X	100	010	100	110
0	X	X	000	000	010	100
X	0	X	001	001	110	100
X	X	0	010	010	100	100
0	X	X	011	011	100	010
X	0	X	100	100	100	110

Table 4: Excitation Table

B. Finite State Machine

A Finite State Machine (FSM) helps visualize the order of states and the color changes in the traffic lights, all while keeping track of the current state the program is in. Changes in state and color are signalled by changes in the inputs (Ea, Eb, and Ec) that the FSM receives. From there, the current state will determine the light's color, as well as the time in which the two groups on the traffic light will stay in each color. A Mealy-type Algorithmic State Machine is used to exhibit this in Figure 3. This FSM was designed with a loop in mind, with state 3 acting as an intermission state where both lights are red.

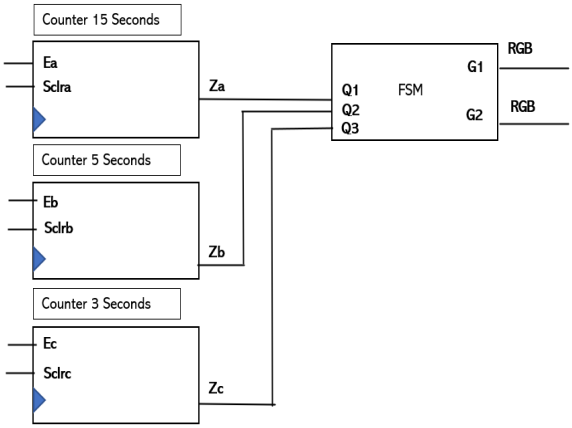

Figure 2: Circuit

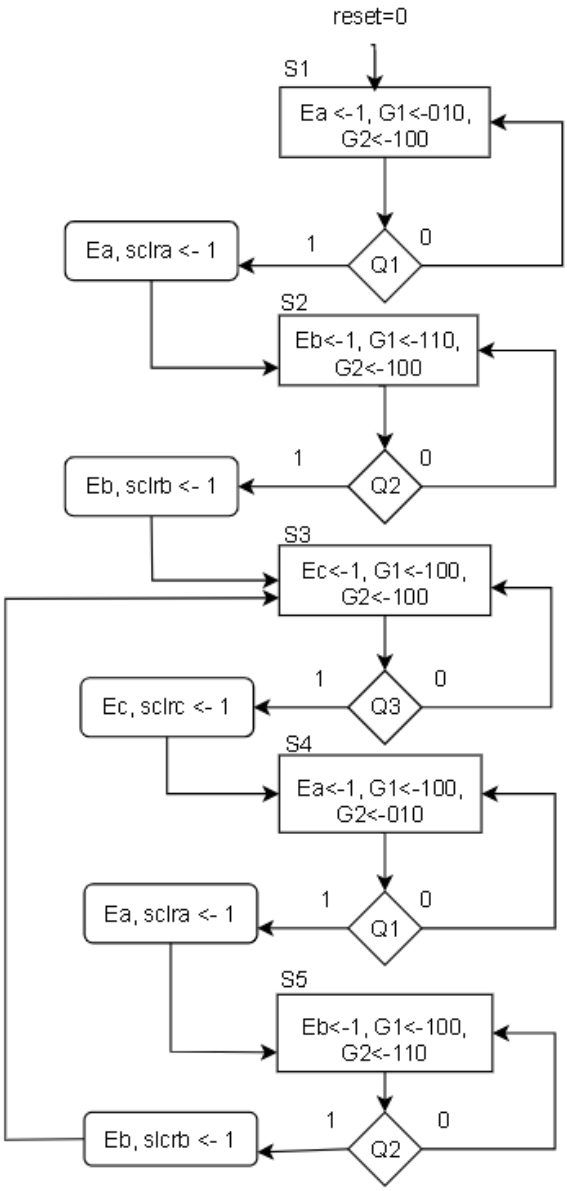

Figure 3: Mealy- type Finite State Diagram.

C. Clock/Counter

The purpose of the clock and counter in the four-way traffic controller is to determine how long each LED should be a certain color in each state and to determine when to change states. One counter is used to count the clock ticks when an LED is supposed to turn from green to yellow in the same state. Another counter is used to determine when to change the FSM from one state into the next. States 1, 2, 4, and 5 will be a total of 20 seconds. When the traffic lights for group 1 are green, they will stay green for 15 seconds and proceed to turn yellow for 5 seconds while group 2 lights are red for the entire 20 second period shown in Table 5. This is the same when group 2 lights are green. State 3 occurs after state 2 and state 5 for a total of 3 seconds as a reset shown in Table 6.

Group 1	Green	Yellow
Group 2	Red	
Time (seconds)	15	5

Group 2	Green	Yellow
Group 1	Red	
Time (seconds)	15	5

Table 5: Timing of the traffic lights for states 1, 2, 4, 5.

Group 1	Red
Group 2	Red
Time (seconds)	3

Table 6: Timing of the traffic lights for state 3.

III. EXPERIMENTAL SETUP

In regard to the experiment, we have compiled a stateschange, top, top_tb, my_genpulse_sclr, and xdc file.

For the state changes we have grouped the four different light changes into 5 different stages, S1, S2, S3, S4, and S5. The inputs will determine in which state we will be along with the previous state. For state 1, if Q1 is 1 then we will go into state 2. If Q1 is 0, then we will remain in the same state. This changes when we get to S5. When we get to S5, if Q2 is 1, we will go back to S3. If Q2 is 0, then we will stay at S5. This would cause a reset for the traffic lights.

```
Transitions: process (resetn,clock,Q1,Q2,Q3)
begin
    if resetn = '0' then
        y <= S1;
    elsif (clock'event and clock = '1') then
        case y is
            when S1 =>
                if Q1 = '1' then y <= S2; else y <= S1; end if;
            when S2 =>
                if Q2 = '1' then y <= S3; else y <= S2; end if;
            when S3 =>
                if Q3 = '1' then y <= S4; else y <= S3; end if;
            when S4 =>
                if Q1 = '1' then y <= S5; else y <= S4; end if;
            when S5 =>
                if Q2 = '1' then y <= S3; else y <= S5; end if;
            end case;
    end if;

end process;


case y is
    when S1 => --G1G G2R
    Ea <= '1';
    G1_G <= '1';
    G1_R <= '0';
    G1_B <= '0';
    G2_R <= '1';
    G2_B <= '0';
    G2_G <= '0';
    if Q1 = '1' then  sclra <= '1'; else sclra <= '0'; end if;

    when S2 => --GIY G2R
    Eb <= '1';
    G1_G <= '1';
    G1_R <= '1';
    G1_B <= '0';
    G2_R <= '1';
    G2_B <= '0';
    G2_G <= '0';
    if Q2 = '1' then  sclrb <= '1'; else sclrb <= '0'; end if;
```

There will be a bit change of plus one for every state change. By the time we get to S5, we are at a value of 4. These values will then be converted into time values in the testbench.

```
-- Counter: 15s
ga: my_genpulse_sclr generic map (COUNT => 1500000000)
    port map (clock => clock, resetn => resetn, sclr => sclra, E => Ea, z => za);

-- Counter: 5
gb: my_genpulse_sclr generic map (COUNT => 500000000)
    port map (clock => clock, resetn => resetn, sclr => sclrb, E => Eb, z => zb);

-- Counter: 3
gc: my_genpulse_sclr generic map (COUNT => 300000000)
    port map (clock => clock, resetn => resetn, sclr => sclrc, E => Ec, z => zc);

gfsm: statechanges port map (
          clock => clock,
          resetn => resetn,
          Q1 => za, Q2 => zb, Q3 => zc,
          Ea => Ea, Eb => Eb, Ec => Ec, sclra => sclra, sclrb => sclrb, sclrc => sclrc
          G1_R => G1_R, G1_B => G1_B, G1_G => G1_G,
          G2_R => G2_R, G2_B => G2_B, G2_G => G2_G );
```

Everyoutput Q will have a corresponding value of time in which the light will stay on, and is linked to one of the three counters as shown in Figure 2. It will also correspond to a specific color of RGB LED, one for each group. The first group's RGB values in S1, will be green and will last for 15 seconds. In S2, the color will be yellow and will last for 5 seconds. In S3, the value is red. During these states group 2s RGB has been red the entire time. S3 is the "break" state of 3 seconds. This pattern flips from S3-S5. The inputs will be placed into the testbench in order to control these states.There is no need to specify any inputs in the testbench, because the inputs and process from the state changes file should work automatically.

IV. RESULTS

When running the program for the traffic light controller, everything seemed to work as it should.The RGB LEDs turned on correctly, changed colors correctly, and the timing turned out as it should. A link to the results of the simulation is shown below.

https://drive.google.com/file/d/1w2frjjPWzK5oxb2tL_8uT6yruc0_M6GY/view?usp=sharing

CONCLUSION

After implementing and completing this project, there was one key takeaway learned. This is how the counter (or multiple counters) can be manipulated to control the status of a finite state machine.There were many challenges that came along the implementation as well. The most plentiful challenge was the timing of each RGB LED for each state. Until the issue was figured out, the timing was originally showing in nanoseconds. We converted the program from nanoseconds to seconds and for the most part it world, besides for State 3. State 3 kept occurring in nanoseconds until muleptile recreations of our code to fix it. Eventually the program worked as it should. Another key takeaway for the project was the exposure and insight to the Artix-7 Nexys DDR FPGA board. Although we were all exposed to labs and got exposure to an FPGA board there, we never used it in the way we did in this project. Therefore, we were able to deepen our learning on the board even more.

REFERENCES

[1] Llamocca. "RECRlab." *VHDL Coding for Fpgas*, http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.

[2] "Finite State Machines: Sequential Circuits: Electronics Textbook." *All About Circuits*, https://www.allaboutcircuits.com/textbook/digital/chpt-11/finite-state-machines/.

[3] "Digital Circuits - Finite State Machines." *Digital Circuits - Finite State Machines*, https://www.tutorialspoint.com/digital_circuits/digital_circuits_finite_state_machines.htm.

[4] *Vivado Tutorial - Xilinx*. https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2013x/Nexys4/Verilog/docs-pdf/Vivado_tutorial.pdf.