Four-Way Traffic Light Controller

Angelo Esho, Jason Mikho, Ethan Talampas
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: angeloesho@oakland.edu, jasonmikho@oakland.edu, etalampas@oakland.edu

***Abstract*— In this project, we will use the Nexys A7-50T board to design and simulate a four-way traffic controller. There will be a path going North and South (P1) and another path going East and West (P2). We will use VHDL to code the timing of when the lights should change color. We will also use a state table, along with an FSM to display all the possible scenarios that could be outputted. This project will help us better understand how the traffic lights work.**

I. Introduction

We have chosen a four-way traffic light controller for this project. On the roads, traffic lights are used to regulate the flow of cars, limiting traffic congestion. A standard traffic light has three colors that tells drivers whether to stop for red, slow down for yellow, or go for green. When the light changes color, each change lasts for a specific amount of time. This time will be modeled by the counters in this project. Our goal for this project is to use VHDL code that will be uploaded to a Nexys A7-50T board to control LEDs that will be on a breadboard.

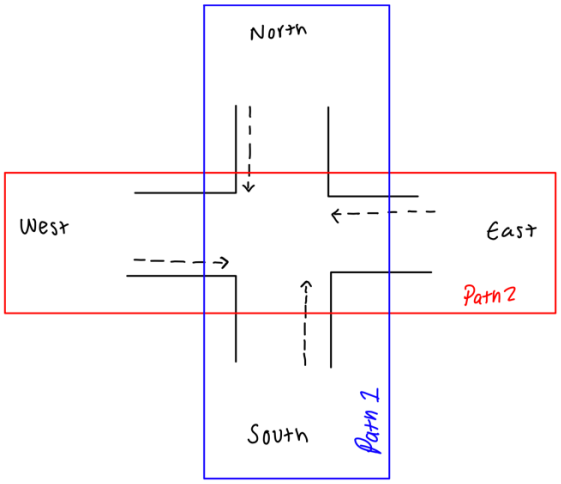

Figure 1: Four-Way Traffic Light Controller Process.

II. Methodology

We will be using VHDL coding language to program our Nexys A7-50T board. We will use a breadboard, LED lights, resistors, counter registers, and power ports to simulate how a traffic light will perform. In Table 1, we grouped two paths going together (North and South, and East and West), this allows us to determine which LEDs should be turned on at specific times.

Outputs	Path	Signal	Description
North South	1	P1G	Path 1 green
		P1Y	Path 1 yellow
		P1R	Path 1 red
West East	2	P2G	Path 2 green
		P2Y	Path 2 yellow
		P2R	Path 2 red

Table 1: Output Table with Signal Description.

In Table 2, we used three counters to demonstrate the time it would take for each light to change colors. We decided to use 3 different counters being 15, 5, and 3 seconds.

Name	Inputs	Outputs
Counter 1	Ea	C_1
	sclra	
Counter 2	Eb	C_2
	sclrb	
Counter 3	Ec	C_3
	Sclrc	
FSM	X_1, X_2, X_3	P1, P2 LEDs

Table 2: Component Inputs and Outputs.

To visualize the inputs, present states, next states, and outputs for this controller, we had to construct a state table. In the table, all possible states are represented, along with the next state which depends on the input. The table also demonstrates the output, which is what color the traffic light would be in both paths. These findings are represented in Table 3 and 4.

Input $C_a, C_b, \& C_c$			Present state	Next state	output (P1 & P2)	
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	101	100	110
x	x	1	101	000	100	100

Table 3: Excitation Table.

Input $C_a, C_b, \& C_c$			Present state	Next state	output (P1 & P2)	
1	x	x	S0	S1	010	100
x	1	x	S1	S2	110	100
x	x	1	S2	S3	100	100
1	x	x	S3	S4	100	010
x	1	x	S4	S5	100	110
x	x	1	S5	S0	100	100

Table 4: State Table.

In the excitation and state tables, state 0 = 000, state 1 = 001, state 2 = 010, state 3 = 011, state 4 = 100, and S5 = 101 the LEDs are presented with green = 010, yellow = 110, and red = 100.

Path 1	Green	Yellow
Path 2	Red	
Time seconds	15	5

Table 5: State One Timing of Traffic Light.

Path 2	Green	Yellow
Path 1	Red	
Time seconds	15	5

Table 6: State Two Timing of Traffic Light.

Path 1	Red
Path 2	Red
Time seconds	3

Table 7: State Three Timing of Traffic Light.

In Table 5, we have Path 1 (North and South) that will be green for 15 seconds then it will change to yellow for 5 seconds, while Path 2 (East and West) will be red for 20 seconds. Following this, the state changes and Path 1 will be red for 20 seconds while path 2 will be green for 15 seconds and yellow for 5 seconds, this is shown in Table 6. In between these state changes, both paths will remain red for three seconds, this is shown in Table 7.

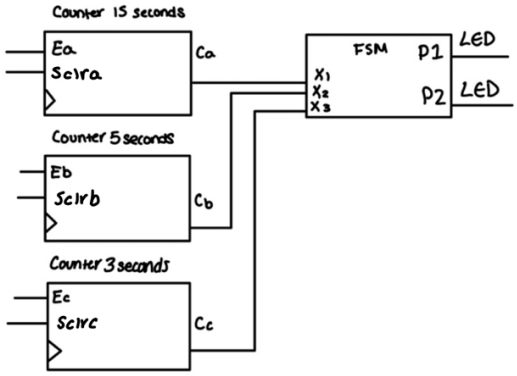

Figure 2: Circuit.

Figure 2 shows the circuit of our design. Input X of the FSM has a value of time for which the desired light will stay lit up. These outputs will also be connected to one counter out of the three. This will correspond to the specific LED on the breadboard. Path 1 is the first group that will be green for 15 seconds for State 0. State 1 changes the light from green to yellow and will stay yellow for 5 seconds. State 2 has both paths at red for 3 seconds. The pattern then flips for state 3, 4, and 5, but now for Path 2. The process from the state changes will work automatically.

III. Experimental Setup

Finite State Machine

Through Vivado, we were able to implement a finite state machine that switched through all the possible states at the correct time. In the code, we had transitions and outputs for the finite state machine. In Figure 3, the transition component of the finite state machine is shown on Vivado.

```
architecture behavioral of FSM is
    type state is (S0, S1, S2, S3, S4, S5);
    signal y: state;
begin

    Transitions: process (resetn, clock, X1, X2, X3)
    begin
        if resetn = '0' then
            y <= S0;
        elsif (clock'event and clock = '1') then
            case y is
                when S0 =>
                    if X1 = '1' then y <= S1; else y <= S0; end if;
                when S1 =>
                    if X2 = '1' then y <= S2; else y <= S1; end if;
                when S2 =>
                    if X3 = '1' then y <= S3; else y <= S2; end if;
                when S3 =>
                    if X1 = '1' then y <= S4; else y <= S3; end if;
                when S4 =>
                    if X2 = '1' then y <= S5; else y <= S4; end if;
                when S5 =>
                    if X3 = '1' then y<= S0; else y <= S5; end if;


                end case;
        end if;

    end process;
```

Figure 3: Transitions of FSM.

The finite state machine starts with state 0 and depends on X1. If X1 is 1, then the state moves to state 1. The states depend on the X values until the fifth state where if X3 is 1, then it goes back to state 0. The outputs portion of the finite state machine is shown in Figure 4.

```
    case y is

        when S0 => --- P1 Green P2 Red
        debug <= "000";
        ea <= '1';
        P1_G <= '1';
        P1_R <= '0';
        P1_Y <= '0';
        P2_G <= '0';
        P2_R <= '1';
        P2_Y <= '0';
        if X1 ='1' then sclra <= '1'; else sclra <= '0'; end if;


        when S1 => --- P1 yellow P2 Red
        debug <= "001";
         eb <= '1';
        P1_G <= '0';
        P1_R <= '0';
        P1_Y <= '1';
        P2_G <= '0';
        P2_R <= '1';
        P2_Y <= '0';
        if X2 ='1' then sclrb <= '1'; else sclrb <= '0'; end if;

        when S2 => --- P1 red P2 Red
        debug <= "010";
         ec <= '1';
        P1_G <= '0';
        P1_R <= '1';
        P1_Y <= '0';
        P2_G <= '0';
        P2_R <= '1';
        P2_Y <= '0';
        if X3 ='1' then sclrc <= '1'; else sclrc <= '0'; end if;

        when S3 => --- P1 Red P2 Green
        debug <= "011";
        ea <= '1';
        P1_G <= '0';
        P1_R <= '1';
        P1_Y <= '0';
        P2_G <= '1';
        P2_R <= '0';
        P2_Y <= '0';
        if X1 ='1' then sclra <= '1'; else sclra <= '0';  end if;

        when S4 => --- P1 Red P2 Yellow
        debug <= "100";
         eb <= '1';
        P1_G <= '0';
        P1_R <= '1';
        P1_Y <= '0';
        P2_G <= '0';
        P2_R <= '0';
        P2_Y <= '1';
        if X2 ='1' then sclrb <= '1'; else sclrb <= '0';  end if;
```

Figure 4: Outputs of FSM.

The outputs of the finite state machine determine which color and path of LEDs should be on. They also determine the enables and synchronous clears for each counter, so that the correct LED turns on for the correct path at the correct time.

Results

The results gave us exactly what we were looking for. Everything worked correctly. The LEDs, the wiring, and the code all ran correctly which allowed us to create a properly functioning traffic light controller. A link to the results shows a simulation of how it should run.

https://drive.google.com/file/d/1Je0m3rJZmP8-ihVjJtNSMbNc3if9Gmjd/view?usp=share_link

Conclusions

In conclusion this project allowed us to learn much more about VHDL code and how it works in the real world. There were a lot of obstacles within the VHDL code. The main obstacle that happened near the end was fixing the counter and the FSM to work accordingly. The counter issue was changing the 15 second count from nanoseconds to seconds. When programming the board all the LEDs were on, this is because the code is running so fast it, we couldn't see any change. The FSM was not looping back to the first state and continuing the cycle. We fixed this by adding another state that allowed the FSM to loop back to the first state. Eventually our code worked out and everything came together. Not only did we learn more about VHDL code, but we learned more about the Artix-7 Nexys A7-50T board and how to implement it to structure and control a circuit on the breadboard. In this project we deepen our knowledge on the board and the code to program it.

References

[1] Llamoca. "Generic Pulse Generator." VHDL Coding For FPGAS, http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html