Four-Way Traffic Lights

ECE 2700 Winter 2017

List of Authors (Jason Gilliam, Esteban Rodriguez, Allison Nguyen, Michelle Nguyen)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-Mails: jagillia@oakland.edu, rodriguez@oakland.edu, atnguyen234@oakland.edu, atnguyen23@okland.edu

***Abstract*—a four-way traffic light controller design project was presented to help us gain in-depth experience in implementing, interfacing and solving problems within a digital system. In this project, we implemented a fully functional traffic signal controller for a four-way intersection traffic. The lights are assumed to operate at a four-way intersection with one road north and southbound and the other road east and westbound. A VHDL code for a traffic light controller on FPGA is presented, and uploading the VHDL design code on Digilent Nexys 4 DDR board to verify our design.**

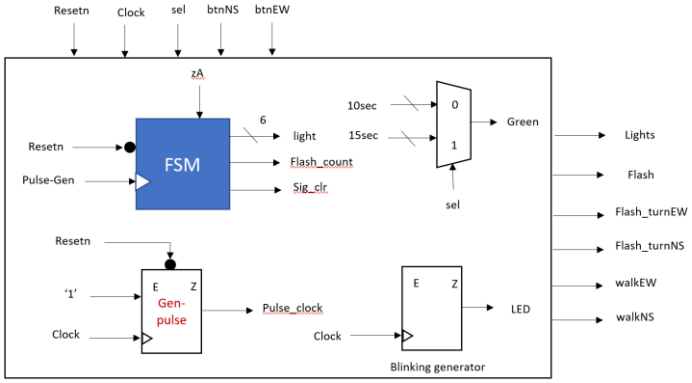
I. INTRODUCTION

Traffic lights, also known as traffic signals, are signaling devices positioned at pedestrian crossings, road intersections, and other locations to control Intersections for the flow of traffic. Currently, traffic lights have been installed in most small and major cities around the world to control the flow of traffic. It allocates the right of way to road users using lights in standard colors (Red - Yellow – Green). Traffic lights are used at busy intersections to aid the flow of traffic more smoothly and safely. The increasing amount of traffic in cities have a large impact on the overcrowding and the time it takes to reach destinations. The method of adding more roads is not enough, since they will always reach an endpoint, like intersections or traffic jams. Traffic jams cannot be prevented. However, the way intersections are controlled has room for improvement. Intersections are controlled mainly by traffic lights and will reduce the amount of traffic jams, especially during rush hour. For this project, we created a prototype traffic light model using LEDs. Since LEDs consume less energy and generally last longer compared to any other light source. This project actively involved the use of a digital logic gate, integrated circuit, timer for our entire design and circuit simulation to help us analyze the entire system efficiently.

II. METHODOLOGY

We created a top file and three sub modules. The top file controls the traffic lights and their states. The first sub module is the pulse generator that takes the edges of the clocks up to one second and returns a one, which triggers the counter to add one second. The second sub module is the LED blinker, which takes the clock on the board and uses it to generate pulses. If the pulse is less than half of 48 million, then the LED continues blinking, when it passes half of 48 million, then the light turns off. The last sub module is the multiplexer to toggle the timing of the green light between 10 seconds and 15 seconds.

The Top File level is shown in Figure 1.

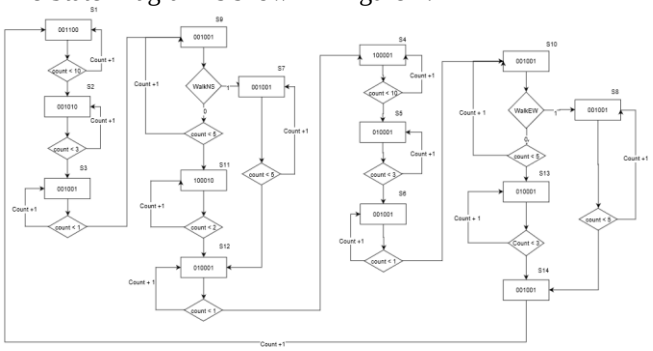


A. *Finite State Machine*

The Finite State Machine (FSM) responds to input signals and generates the outputs to control signals needed for the system functions. In our case, the FSM is constructed in If and Case statements, also each changing state is decided by a predefined counter for each scenario. For example, the red light stays on for 10 seconds, green light can be toggled to stay on 10 seconds or 15 seconds using a switch that depends on the flow of the traffic. The crosswalk light stays on for 5 seconds and the left turn light stays for 10 seconds when the straight light is green and 5 seconds for left turn only. Two switches are also dedicated to represent a push button for crosswalk. This allows the

crosswalk signal lights to be turned on only when the switch = 1.

The State Diagram is shown in Figure 2:



A piece of Finite State Machine code is also shown here:

```
process (pulse_clk, resetn)

begin
    if resetn = '0' then y <= S1;
       seconds_count <= (others => '0');
    elsif (pulse_clk'event and pulse_clk = '1') then
       case y is
                when S1 =>
                if seconds_count < Green_Light_Time then
                    y <= S1;
                    seconds_count <= seconds_count + 1;
                else
                    y <= S2;
                    seconds_count <= (others => '0');
                end if;

                when S2 =>
                if seconds_count < three_sec then
                    y <= S2;
                    seconds_count <= seconds_count +1;
                else
                    y <= S3;
                    seconds_count <= (others => '0');
                end if;
```

III. EXPERIMENTAL SETUP

B. *Hardware:*

VHDL code is written in the program and Vivado is uploaded to Digilent Nexys 4 DDR board. Four breadboards, each represents the traffic lights at each corner and were powered directly from the Nexys board. 20 LEDs are used at four corners to represent the traffic lights (red, green, orange and blue) and 20 resistors for each LED. We also uses 4 different switches on the Nexys board to control: 1) toggle the green light between 10sec and 15sec, 2) EW crosswalk, 3) NS crosswalk and 4) the turning on/off of the left turn blinking yellow.

C. *Software:*

As to what we are being taught in ECE 2700 class, our group mainly used language called VHDL to code for the entire project. We used Dr. Llamocca's coding examples to develop the feasible coding that can be used and applied to our project.

Pulse Generator

The pulse generator is a function that takes the internal clock and creates a signal that would output every one second. This was done by creating a variable that had a value of 100 million. We then used a if statement to implement a count increase every uptick of the clock. Since the internal clock was a 100 Mhz, the count would have to wait 100 million ticks before allowing the output to be one.

```
    process (resetn, clock)
    begin
        if resetn = '0' then
            Qt <= (others => '0');
        elsif (clock'event and clock = '1') then
            if E = '1' then
                if Qt = conv_std_logic_vector (COUNT-1,nbits) then
                    Qt <= (others => '0');
                else
                    Qt <= Qt + conv_std_logic_vector (1,nbits);
                end if;
            end if;
        end if;
    end process;

    z <= '1' when Qt = conv_std_logic_vector (COUNT-1,nbits) else '0';
   Q <= Qt;
```

Blinking Generator

The blinking generator function was the same as the pulse generator, which would send an output high every quarter of a second. Then it would send an output low to the other quarter of a sec. We were then able to link the signal to a led to blink.

```
architecture struct of Blink_LED is

constant max_count : natural := 48000000;
signal Rst : std_logic;

begin

Rst <= '0';

-- 0 to max_count counter
compteur : process(refclk, Rst)
    variable count : natural range 0 to max_count;
        begin
            if Rst = '1' then
                count := 0;
                led <= '1';
            elsif rising_edge(refclk) then
                if count < max_count/2 then
                    count := count + 1;
                    led <= '1';
                elsif count < max_count then
                    led <= '0';
                    count := count + 1;
                else
                    led <= '1';
                    count := 0;
                end if;
            end if;
end process compteur;

end struct;
```

Multiplexer

The multiplexer was a function that acted as a switch that would turn the green light on for a longer or shorter period of time. We did this to make a demo mode.

```
13  end MUX_2to1;
14
15  architecture Behavioral of MUX_2to1 is
16
17  begin
18
19      with sel select
20
21          MUX_out <= ten_sec when "0",
22          fifteen_sec when others;
23
24  end Behavioral;
```

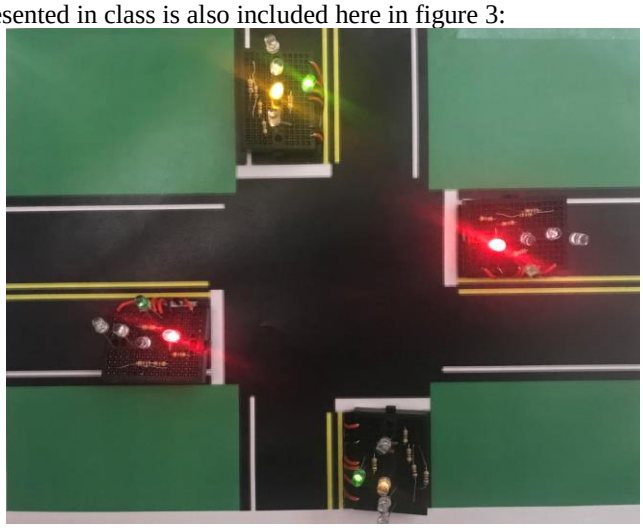
D. Test Bench:

Before we can implement our code to test the hardware components, we also created a test bench to make sure that our code worked properly. A test bench is the best solution to verify if our code is working or not.

IV. Results

After trials and errors generating the bitstream, we were able to debug and successfully program the board. As a whole, the project functioned exactly how we expected. Our sequence of coding is demonstrated faultlessly based on the LEDs' behavior on the breadboard. Due to the limitation of the breadboard space, the solid left turning light is represented by the solid yellow light, that is also used for the left-blinking yellow turning light. We used codes from Dr. Llamocca for reference to create our own code as well. In class, we have analyzed many state diagrams which allowed us to create our state diagram more smoothly. Also, it benefits us a lot by learning about finite state machine close to the end of the semester, it helps us to have better analyzing skills on the finite state machine, from then we can apply this knowledge into our project and create a functional finite state machine on our own without the help of the professor or TAs.

An image of the working traffic controller that we presented in class is also included here in figure 3:



V. Conclusions

For this project, it is intended for the design and implementation of four-way traffic light control system.

The design is accomplished by taking the tasks of searching, brainstorming, creating, and reviewing our design to producing a functioning product. The design was preceded by getting all necessary components to initialize and maintain the proper functions of the desired design code.

The control circuit was then designed and faults were corrected before the final project was presented. It was quite challenging to accomplish the perfection required to achieve an accurate output or result due to some software bug issues. Proper functioning of the circuit and an error free connection was safeguarded.

The circuit is finally put to test and automatic control of the traffic light is achieved by the VHDL code we created.

Some of the improvements that we suggest for the system are detection of traffic volume by adding more logic into the coding and sensors; Emergency vehicle detection such as ambulance, police etc. by using wireless sensor network at the signal intersection; system can give more time to an intersection approach that is experiencing heavy traffic, or shorten or even skip a phase that has little or no traffic waiting for a green light; or traffic signals are activated to detect the approach of a train near a rail crossing.

Reference

A Arefín, Utsho & Hafizullah, M.R.M & Sabuj Ahamed, Md. (2013). Implementation of Automatic Traffic Light Controller. . 10.13140/2.1.4076.0640.