Traffic Light Controller

Aaron Gott, Andres Manzanares-Davila, Kristi Stefa, Damion Duemling
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: aarongott@oakland.edu, andresmanzanare@oakland.edu, kstefa@oakland.edu, dduemling@oakland.edu

Abstract*— this project was centered about the development of a 4-way intersection traffic light controller. We learned plenty of new information but most importantly how traffic lights change between different states and modes. Also, adding simple modes such as Rush Hour, After Hours, and Crosswalk made the project a lot more challenging than we originally thought. We put our prior knowledge in practice and used VHDL code as well as a Nexys Board to build the traffic light controller.

I. Introduction

When deciding what to do for our final project we had one thing in mind: choose something that we used on a daily basis. Traffic lights ensure the safety of millions of people and allows them to cross intersections without trouble. Before we had traffic lights, crossing intersections was a hassle but since the early 1900s traffic lights have helped us relieving traffic while making it safer to cross intersections. For this project a 4-way traffic light controller was created. The controller used an FPGA that was programmed using VHDL. the components of the controller consist of a finite state machine, three counters, and two D Flip-flops. The motivation of this project is to take these things that we learned in class and apply them to a real-world application to better understand their uses.

Along with a standard function traffic light, there are different modes that can be enabled just like in the real world. The Other features of this traffic light controller will include, an 4-way flashing mode that will flash the traffic lights red or yellow as they normally do late at night. The controller will also feature two crosswalk buttons, one to cross the north-south street and the other will be to cross the east-west street. When the corresponding switch is flipped to trigger the cross walk the controller will then stop the traffic in the corresponding direction so that a pedestrian is able to cross the street.

II. Methodology

A. Counters

There are three counters used to control the traffic lights. One for the green light, one for the yellow light, and one for the crosswalk. Notice how we didn't need a red light counter. Our goal was to have the green light counter for ~15 seconds, the yellow counter for ~5 seconds, and therefore the combination of these would make our "imaginary" red light counter ~20 seconds.

```
architecture Behavioral of CWCounter is
signal Qt: integer range 0 to 1000000000;
begin
    process (resetn,clock,sclr,E)
    begin
        if resetn = '0' then qt <= 0;
        elsif (clock'event and clock ='1') then
           if sclr = '0' then qt <= 0;
           else
                if E = '1' then
                    Qt <= Qt + 1;
--
               end if;
           end if;
        end if;
        end process;
        z <= '1' when Qt = 200000000 else '0';
        Q <= Qt;
end Behavioral;
```

B. D Flip-Flops

We used two D Flip-Flops in our project. One was for the Main Road crosswalk and the other for the back road crosswalk. The main function of these was to register if/when either of the crosswalk buttons was pressed and if so, when it was pressed.

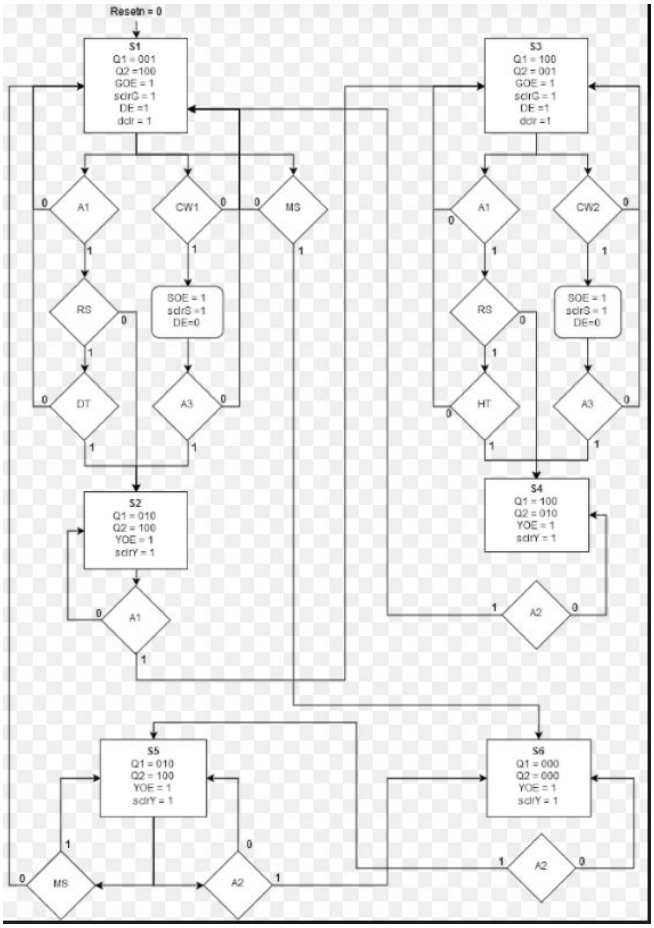
```
architecture Behavioral of DFlipFlop is

begin
    process (clock, E, prn, clrn)
    begin
        if clrn = '0' then q <= '0';
        elsif prn = '0' then q <= '1';
        elsif (clock'event and clock='1') then
            if E = '1' then q <= d; end if;
        end if;
     end process;

end Behavioral;
```

C. FSM

The Finite State Machine was the most important part of the code. It allowed us to change between six different states depending on which mode we were in.
State 1 was the Main Road = Green, Back Road = Red.
State 2 was the Main Road = Yellow, Back Road = Red.
State 3 was the Main Road = Red, Back Road = Green.
State 4 was the Main Road = Red, Back Road = Yellow.

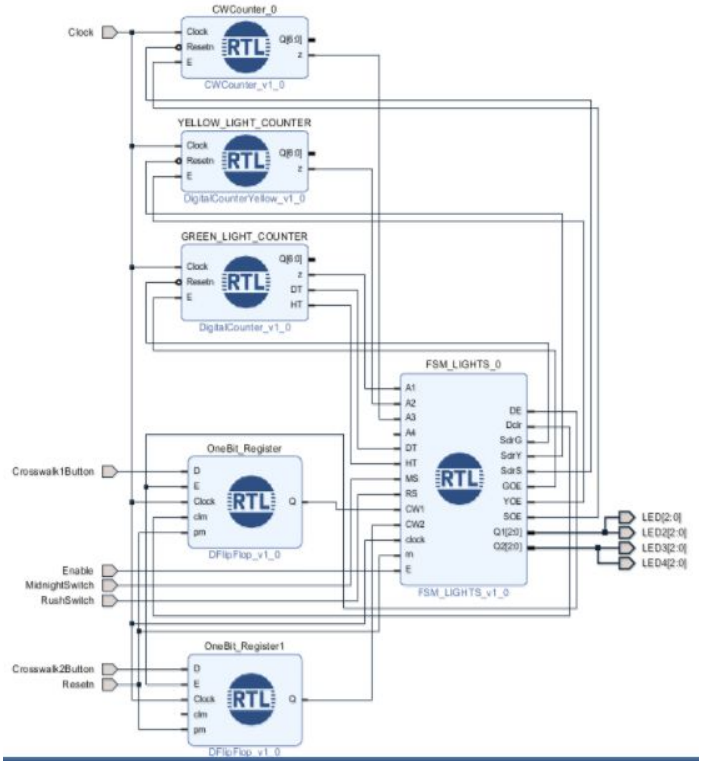
The first four states were the most important since we had all possible outcomes of both traffic lights. However, for our After Hours mode we had to introduce two more states:
State 5 was the Main Road = Yellow, back road = Red.
State 6 was the Main Road = OFF, back road = OFF.

These two states allowed us to have the Main road blinking yellow (proceed with caution) and the back road blinking red (treat it as a stop sign). We used the yellow light counter to switch between these two states.



D. Top Level Design

The top file allowed us to connect all of our components together. Our top file had seven inputs: clock, resetn, enable, RushSwitch, MidnightSwitch, Crosswalk1 Button, and Crosswalk2 Button. The top file also had four 3 bit outputs: LED, LED2, LED3, LED4. The first two represent the traffic lights in the Main Road, while the last two represent the traffic lights in the back road, This can be seen in the following Block Diagram/Top File.



III. Experimental Setup

A. Software

The software that was used to create the traffic light controller was Vivado design suite by Xilinx. This software uses the VHDL programming language to create each component of the Program. The goal was to create the distinct states at appropriate times and get the right outputs for the top file. The components were coded individually and then mapped together to create a central and governing top file. The top file was then connected to the physical hardware with the board and LEDs for visualization.

B. Testbench

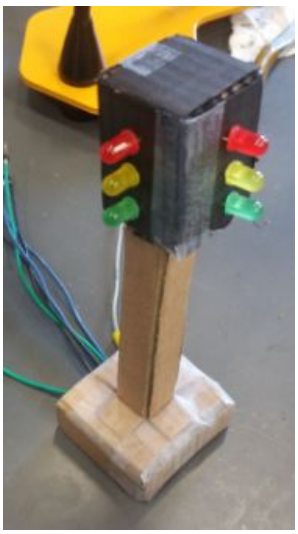
The testbench that was created to test the program that was written was meant to cycle the inputs into each possible state. This was to make sure that the state machine

and the outputs were functioning correctly before it was uploaded to the FPGA. By verifying that code is working with a testbench we are saving time by not having to re-upload the code to the FPGA every time a change is made.

```
clock_process: process
begin
    clock <= '0'; wait for (T-T*DC);
    clock <= '1'; wait for T*DC;
end process;

stim_process:process
begin
    E <= '1';resetn <='1';wait for 50*T;
    CW1 <= '1'; wait for 3*T;CW1 <= '0';wait for 300*T;
    CW2 <= '1'; wait for 3*T; CW2 <= '0';wait for 300*T;
    MS <= '1'; wait for 300*T;MS<= '0';wait for 150*T;
    resetn<='0'; wait for 20*T; RS<= '1';wait for 300*T;

end process;
end Behavioral;
```

C. Hardware

The hardware that was used to model the traffic light consisted of a NEXYS 4 DDR FPGA, four red LEDs, four yellow LEDs, four green LEDs, one 100Ω resistor, jumper wires and a breadboard. The LEDs were arranged on the breadboard to represent the 4-way traffic light and then wired together so that the north and south facing lights would activate the same color LEDs at the same time, this was also done with the east and west facing lights. After the lights were wired they were connected to the NEXYS FPGA through the PMOD headers. These PMOD headers were programmed to turn on and off the LEDs as the program cycles through its different states.



IV. Results

During the preliminary testing, some issues were found where the system would be stuck in a certain state if the rush switch was enabled. After the code was properly debugged, the lights worked as expected according to our plan and the different modes could be displayed. Because the LEDs came out very dim, a new set was implemented using a breadboard for the demonstration. After initial testing, the timing was deemed lengthy and so the program was adjusted to be shorter. The program worked and displayed in logical steps so issues were very apparent and easy to diagnose.

Conclusions

The main takeaway that could be taken from this project is the importance of planning out a circuit and being able to implement it in a real situation. Trying to implement a design and plan is inherently difficult so the labs done in class were a helpful guide for the project. A lot of technology that is in use today makes use of programmable boards and circuits that are designed and implemented. Also of importance is how well the program visualizes and how the code can control lights and other physical implements. More can be attached to the base project such as sensors to check cars, alarms if passed on red, and others that make the system more complex and useful.

References

[1] D. Llamocca-Obregon, VHDL Coding for FPGAs http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html