4-Way Traffic Signal

Jack Noble, Alton Kadow, Sean Koepf, Ravi Prajapati

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

e-mails: jacknoble@oakland.edu, rprajapati2@oakland.edu, skoepf@oakland.edu, altonkadow@oakland.edu

I. INTRODUCTION

To construct a 4-way traffic signal, we will need 12 LED's (three for each intersection). An internal clock will count up to a preset time, which will then change which lights will be switched on or off. Each of the three lights on each side will have it's own time interval (yellow being the shortest duration). In the intersection, the lights opposite each other will be programmed to have the same signals displayed, while the lights perpendicular will wait on red, then vice versa. This gives the intersection the most efficient flow of vehicles.

For our final project we created a four way traffic light, in order to do this we will have to code them to face four directions, one for each cardinal direction. The north and south facing lights will be timed to always be the opposite of the east and west with the exception of a delay when either set of lights turn red. We have several members working in the automotive business and we are passionate about making cars as secure and safe as possible. Nowadays with distracted drivers it is very important for road safety systems to be working at their optimal performance. A traffic light being out of sync or inaccurate could potentially cause devastating accidents. When a light is not correctly timed for the road it could cause traffic jams everywhere. These are things that we would always like to avoid. This is why we settled on a traffic light being the perfect option for our group. We will be using a finite state machine, a clock, and combination logic. These are all things that we learned in this class that apply to our project. Most of our project will be following things we learned already in the class. This traffic light could be used on non-busy roads that don't need a left turn light.

II. METHODOLOGY

The 4-way traffic signal will have a total of 12 lights that will need to be controlled. The naming of these lights in the scope of this project will consist of

the traffic direction each light will be controlling. The lights will be named North, South, and East, and West for each direction that has a light respectively. Each light will have a bit length of 3 bits, each being “XXX”, an example of a red light would be “100”, and an example of a green being “001”. We gleaned that this would be an efficient way to write the code, from an offered YouTube video from 2012, which had as an example a rudimentary version of the circuit we wished to construct.

```
entity fsm is
  generic(COUNT: integer);
  Port ( clock, resetn: in std_logic;
         zC: in std_logic;
         eC: out std_logic;
         sclrC: out std_logic;
         sel: in std_logic_vector(1 downto 0);--input for state machine to oscilate between 2 states of all_red & all_off
         north, south, east, west: out std_logic_vector(2 downto 0));--red"100" yellow"010" green"001"
end fsm;

architecture struct of fsm is

type fsm_state is (all_red_nn, all_red_wn, nsg_ewr, nsy_ewr, nsr_ewg, nsr_ewy, all_yel, all_off);
```

Initialization of FSM & All States

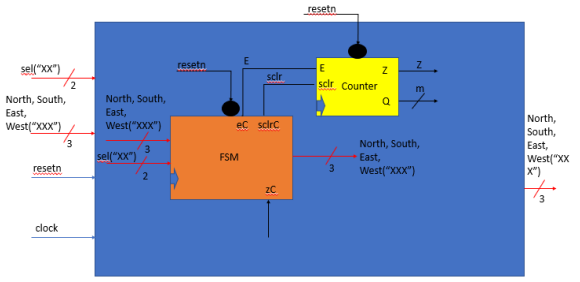
A finite state machine and counter will be used to control which lights will go on and which will go off.

```
        case state is
            when all_red_nn =>
                north <= "100";
                south <= "100";
                east <= "100";
                west <= "100";
                if counter = ((10**8)-1) and sel ="00" then
                    counter <= 0;
                    state <= all_off;
                elsif counter = ((10**8)-1) and sel = "01" then
                    counter <= 0;
                    state <= nsg_ewr;
                elsif counter = ((10**8)-1) and sel = "10" then
                    counter <= 0;
                    state <= nsg_ewr;
                elsif counter = ((10**8)-1) and sel = "11" then
                    counter <= 0;
                    state <= all_yel;
                end if;
            when all_off =>
                north <= "000";
                south <= "000";
                east <= "000";
                west <= "000";
                if counter = ((10**8)-1) and sel ="00" then
                    counter <= 0;
```

Example of FSM case-statement code

III. EXPERIMENTAL SETUP

Before we started coding our final project, it was clear to us that a Finite State Machine (FSM) would be used to run the majority of the logic. A rough sketch of our desired outcome was drawn out on paper, and was used throughout the entire process. We did have to experiment with the counter to get the correct timing we wanted out of it. To make the coding shooter, we tried to use the code for north and west on south and east because the north and south will always have the same values, and the east and west will always have the same values.



Block Diagram of Circuit Structure

This unfortunately caused more problems than it solved, so we controlled each one separately. We used the circuit above to allow for a synchronous clock, counter, and fsm, as well as output the lights all together on the proper timing. The counter used was modified based off of the counter provided to us during lab 6 of the semester.

IV. RESULTS

After some trial and error to see what works best in our case, we achieved the results that we were hoping for. After achieving our goal we even added a couple more functions. We got the counter to cooperate with the internal

clock of the FPGA which allowed us to pause our traffic lights at various instances for any amount of time within one second intervals. This was crucial to our project because without being able to pause the lights, the process would happen so fast it would not be recognizable, rendering it useless. The lights worked flawlessly in the end, and as long as it is viewed in the right orientation, the traffic patterns are clearly visible, as we were hoping.

Conclusions

The takeaway from this project is that traffic control is a very important part of everyday life and it takes a good amount of work to code and make sure things like traffic lights are running smoothly. We managed to create a really well timed light for a mediumly busy intersection. We could look into busier roads that need to have left and right turn lanes. This would just require more steps and more lights than what we had available. Another issue that has yet to be solved is how this light would be timed in concurrence with other traffic lights around it to make sure there isn't too many jams between lights. In conclusion, while we have all of these things we still could consider, our light does work for a four way stop. It changes for rush hour and can change to blinking red at night, or blinking yellow in hazardous situations. We learned in this project that Vivado can be used to make a well functioning traffic light.

References

1. *TY - BOOK*
   AU - Khan, Aamir
   AU - Mallet, Frédéric
   AU - Rashid, Muhammad
   PY - 2015/06/11
   T1 - Modeling SystemVerilog Assertions using SysML and CCSL

2. *LBEbooks. (2012, November 12). Lesson 92 - Example 62: Traffic Light Controller [Video]. YouTube. https://www.youtube.com/watch?v=6_Rotnw1hFM*

3. *"VHDL Code for Traffic Light Controller." Fpga4student.Com, https://www.fpga4student.com/2017/08/vhdl-code-for-traffic-light-controller.html. Accessed 18 Apr. 2021.*

4. *https://forums.xilinx.com/t5/Implementation/traffic-light-controller-nexys4*

ECE 2700 FSM

April 15, 2021