ECEN 429: Introduction to Digital Systems Design Laboratory

North Carolina Agricultural and Technical State University

Department of Electrical and Computer Engineering

Michael Newman (Reporter)

Chad Watson (Lab Partner)

November 14, 2019

Lab #9

Introduction

The purpose of this lab was to implement the FSM for a traffic controller. For this implementation, the LEDs served as the traffic lights. And the switches served as different sensors in the road as well as crosswalk buttons. We are doing this lab to get an understanding of real world applications that are using FSMs. Another key takeaway from doing this lab is that it allows us the opportunity to really test ourselves and our understanding of how to design, and implement a reliable and accurate state machine. Using our knowledge gained from previous labs, we were able to design 3 different models of a traffic controller that adhered to different conditions, and implement the design with various components such as the clock divider and counter.

Background, Design Solution, and Results

Part 1:

For part one of this lab, we implemented the traffic controller from part 2a of the prelab. This particular design accounts for a left turn lane sequence to occur when traveling in the East/West direction only. The sequence occurs as follows: First, the North/South G-Y-R sequence occurs. There are sensors in place to detect that a car is in fact in an east or west lane, and once this sensor is detected (equal to 1), the G-Y-R East/West Straight lights go off. Then the G-Y-R East/West left turn lanes go off(after the red light of the East/West straight direction). After this left-turn sequence, the North/South straight G-Y-R lights go through sequence repeatedly until a car is detected in the East/West direction again. Lastly, the Crosswalk buttons are taken into consideration when the light is red for the North/South straight direction, and when the light is red for the East/West straight direction. Figure 1.1 below is the State diagram for this design.

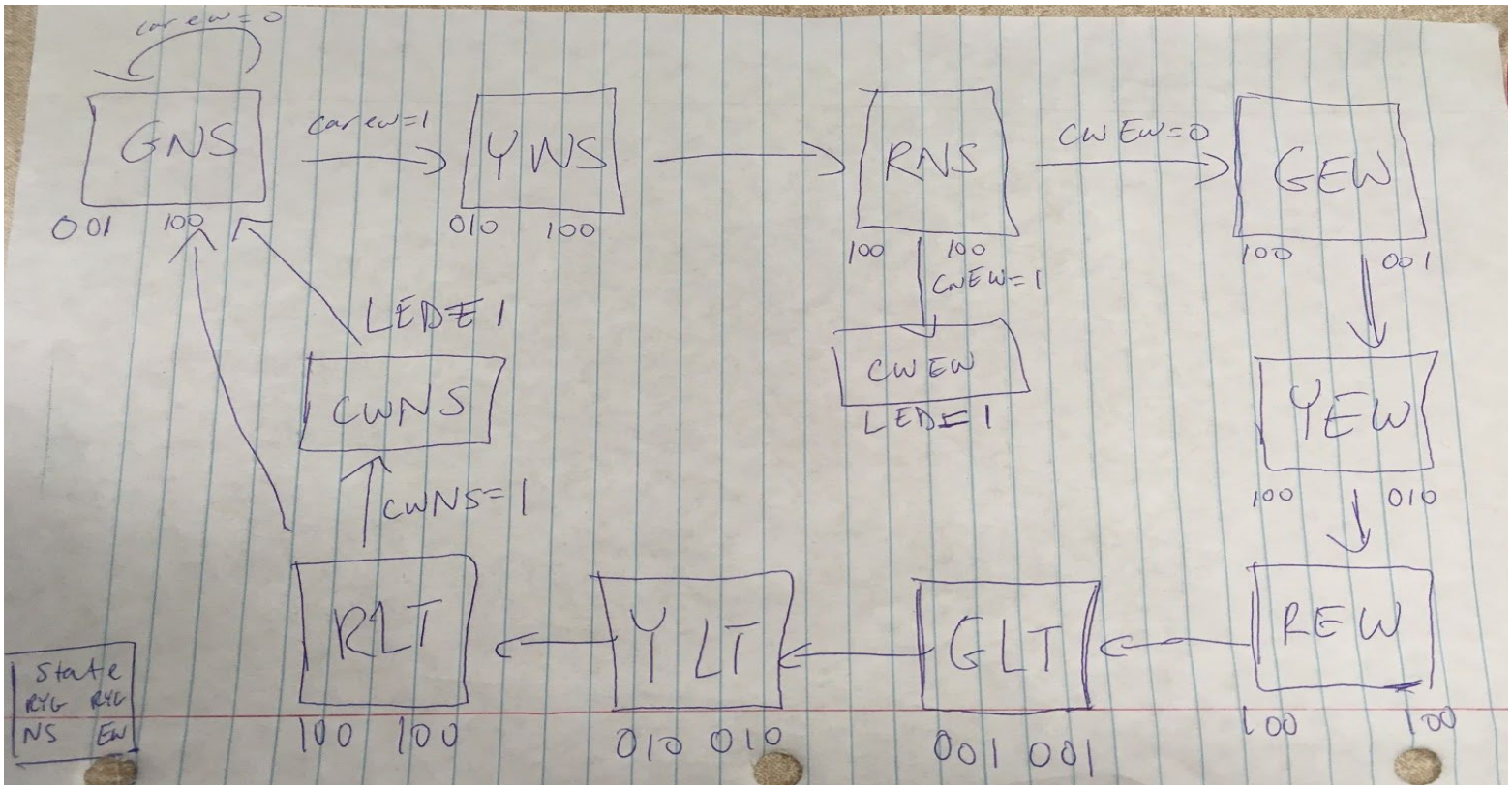

Figure 1.1 shows the state diagram of the traffic controller for Part 1 of the lab

The schematic, pin mapping, and fpga board implementations of our design for part 1 are shown below.

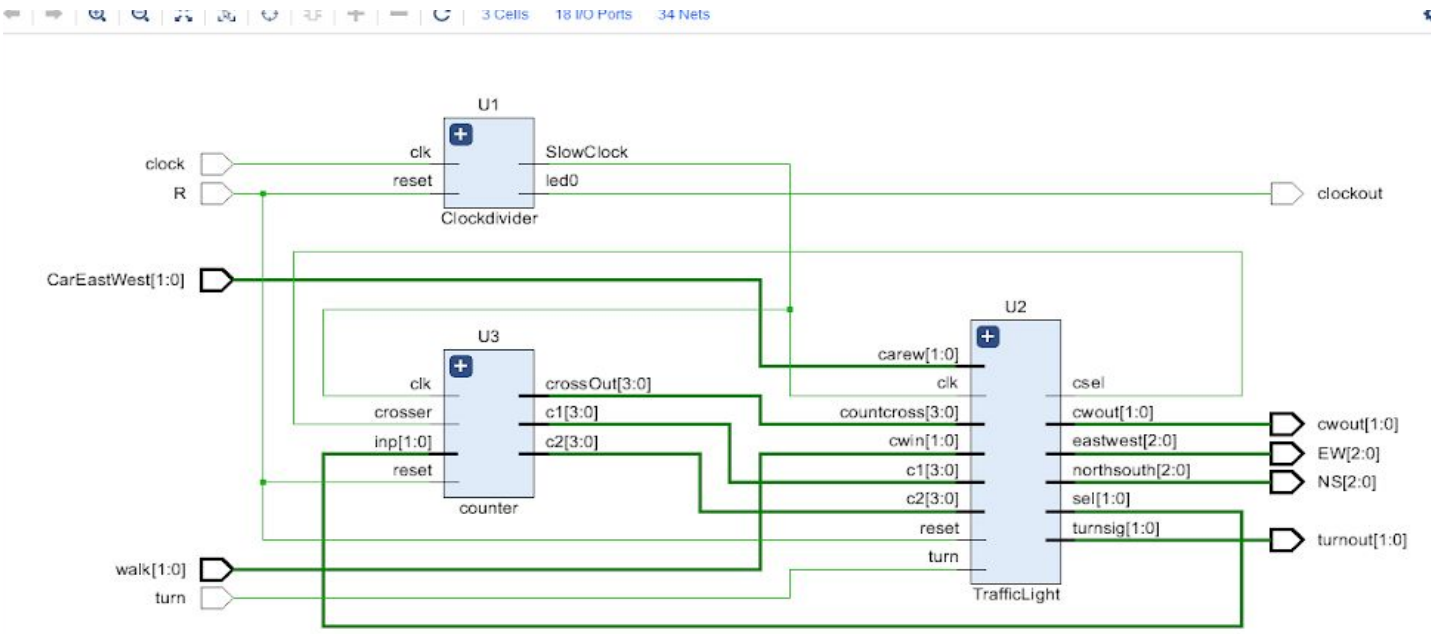

Figure 1.2 shows the schematic for the traffic controller design from part 1a

FPGA Pin Assignment for Traffic Light

input/output	Pin Assignment
clk:input	W5
careast: input	R2
carwest:input	T1
clock:input	W5
CLTNE:input	U1
CLTSW:input	W2
cwEWin:input	R3

cwNS:input	T2
cwNSLED:output	E19
clockout:output	V13
cwEWLED:output	U16
EW(2):output	L1
EW(1):output	P1
EW(0):output	N3
NS(2):output	U3
NS(1):output	W3
NS(0):output	V3
NELT(2):output	V14
NELT(1):output	U14
NELT(0):output	U15
SWLT(2):output	W18
SWLT(1):output	V19
SWLT(0):output	U19
R: input	T3

Table 1.1: Displays the input/output and the corresponding FPGA pin assignment for the Traffic Light

Results on FPGA board

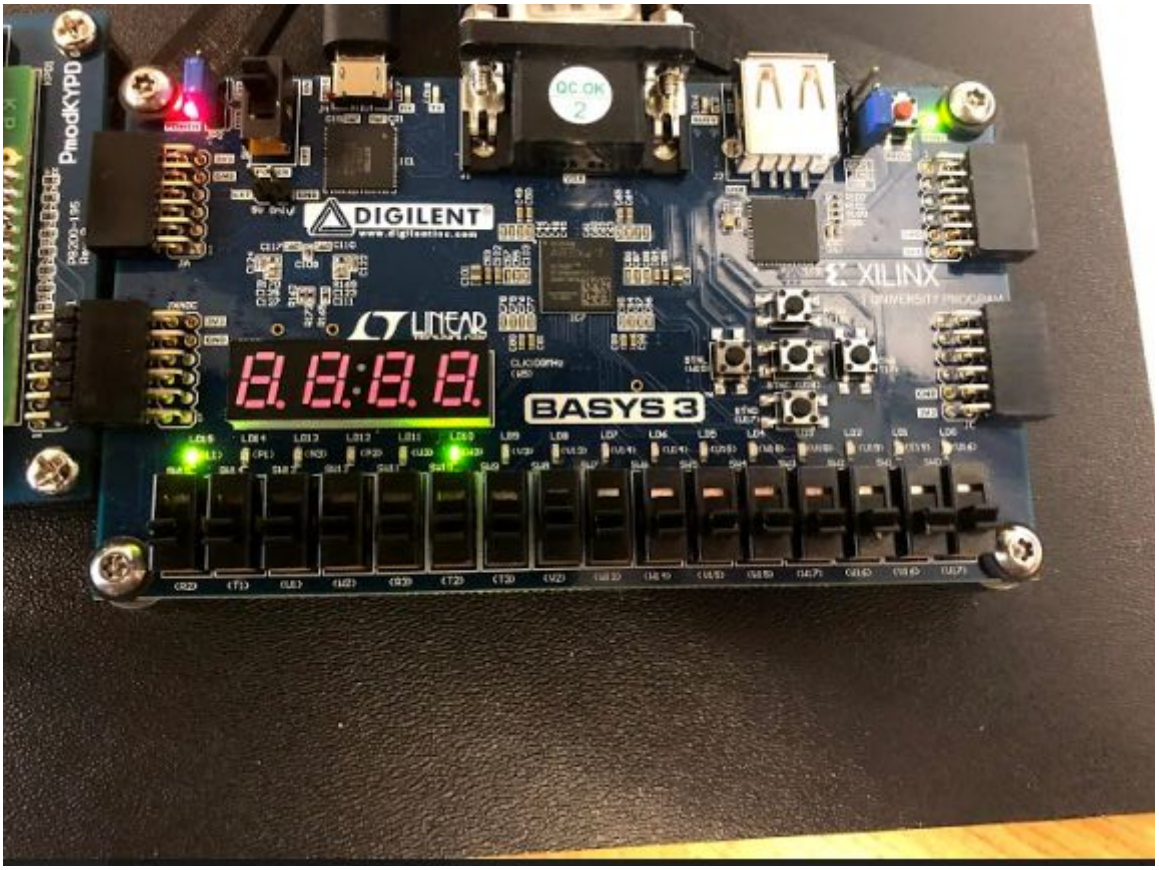

Figure 1.3 This is a picture of the traffic light in the default state GNS. the leds are 100 001 signifying a red in the GEW state and a green in GNS.

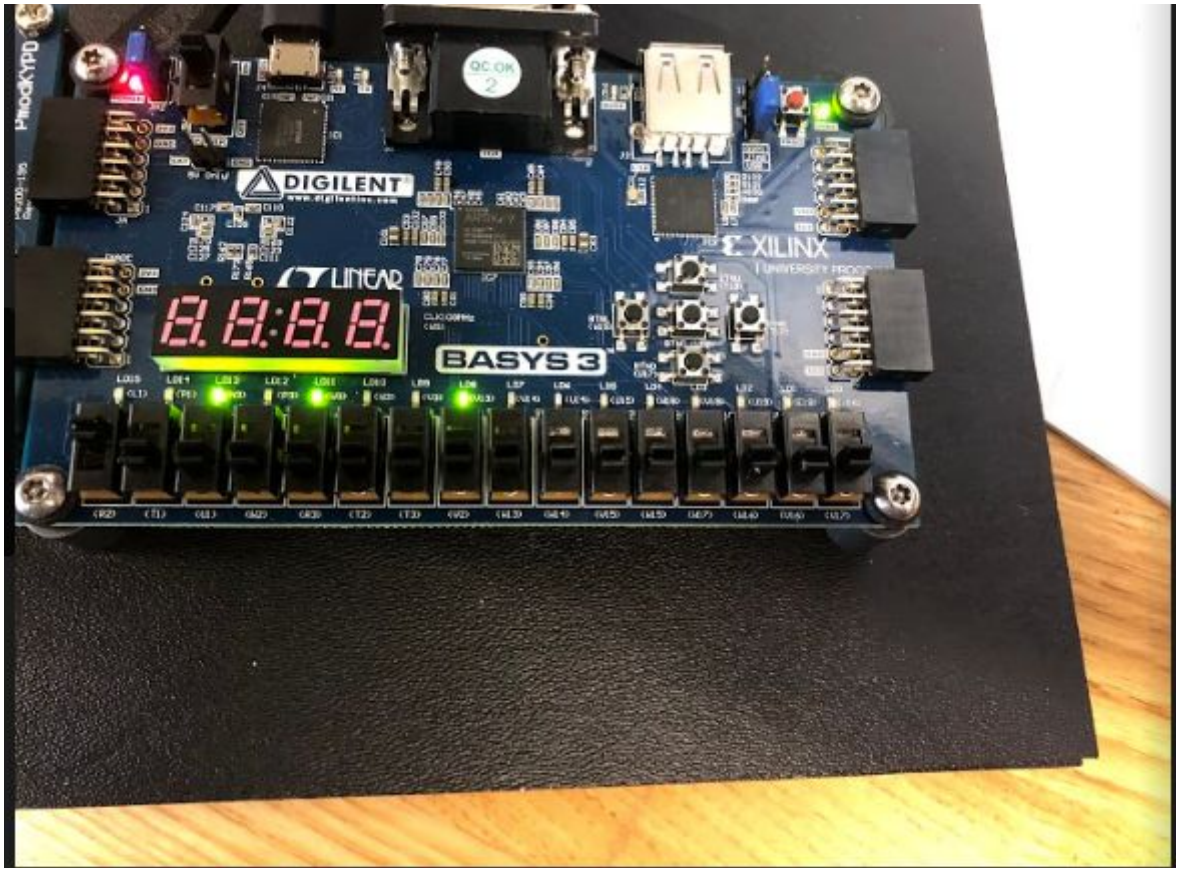

Figure 1.4 This is a picture of the traffic light with the care signal high which moves the state machine to a green in GEW and a red in GNS

Part 2a:

For part 2a of this lab, we implemented the traffic controller from part 2b of the prelab. For this version of the traffic controller, the turn left turn lanes have a sensor to determine if a car is waiting to make a left turn. Figure 2.1 below shows the state machine for this design.

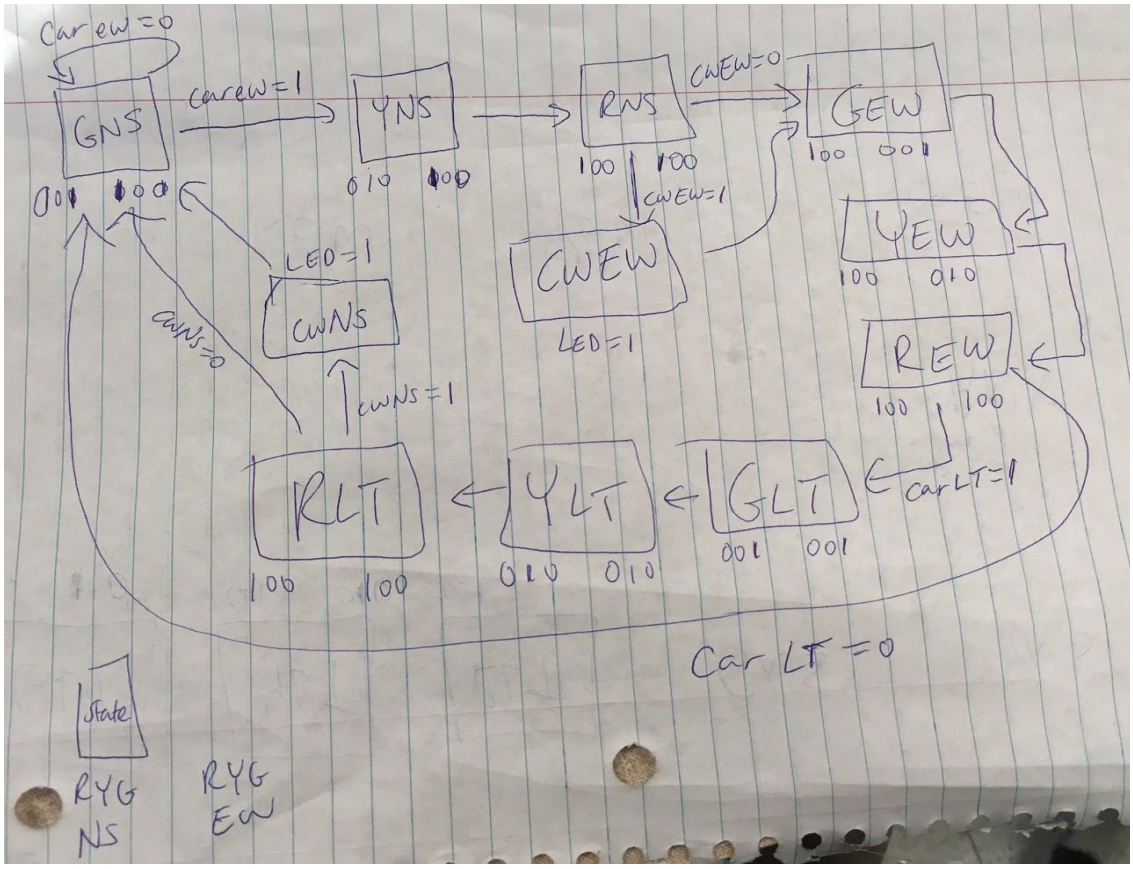

Figure 2.1 State machine for the traffic light system

Part 2b:

For the final part of the lab, we slightly modified part 2a. For this design, when the crosswalk is on for the North/South direction, the seven segment display shows the number 4. Alternatively, when the crosswalk is on for the East/West direction, the seven segment display shows the number 2. Essentially, the design from part 2a remains, with an additional seven segment output added to the FSM, and a seven segment component added to handle the output.

FPGA Pin Assignment for Traffic Light

input/output	Pin Assignment
clk:input	W5
careast: input	R2
carwest:input	T1
clock:input	W5
CLTNE:input	U1
CLTSW:input	W2
cwEWin:input	R3
cwNS:input	T2
cwNSLED:output	E19

clockout:output	V13
cwEWLED:output	U16
EW(2):output	L1
EW(1):output	P1
EW(0):output	N3
NS(2):output	U3
NS(1):output	W3
NS(0):output	V3
NELT(2):output	V14
NELT(1):output	U14
NELT(0):output	U15
SWLT(2):output	W18
SWLT(1):output	V19
SWLT(0):output	U19
R: input	T3
sevenseg(6):output	W7
sevenseg(5):output	V5
sevenseg(4):output	U5
sevenseg(3):output	V8
sevenseg(2):output	U8
sevenseg(1):output	W6
sevenseg(0):output	U7

Table 2.1: Displays the input/output and the corresponding FPGA pin assignment for the Traffic Light

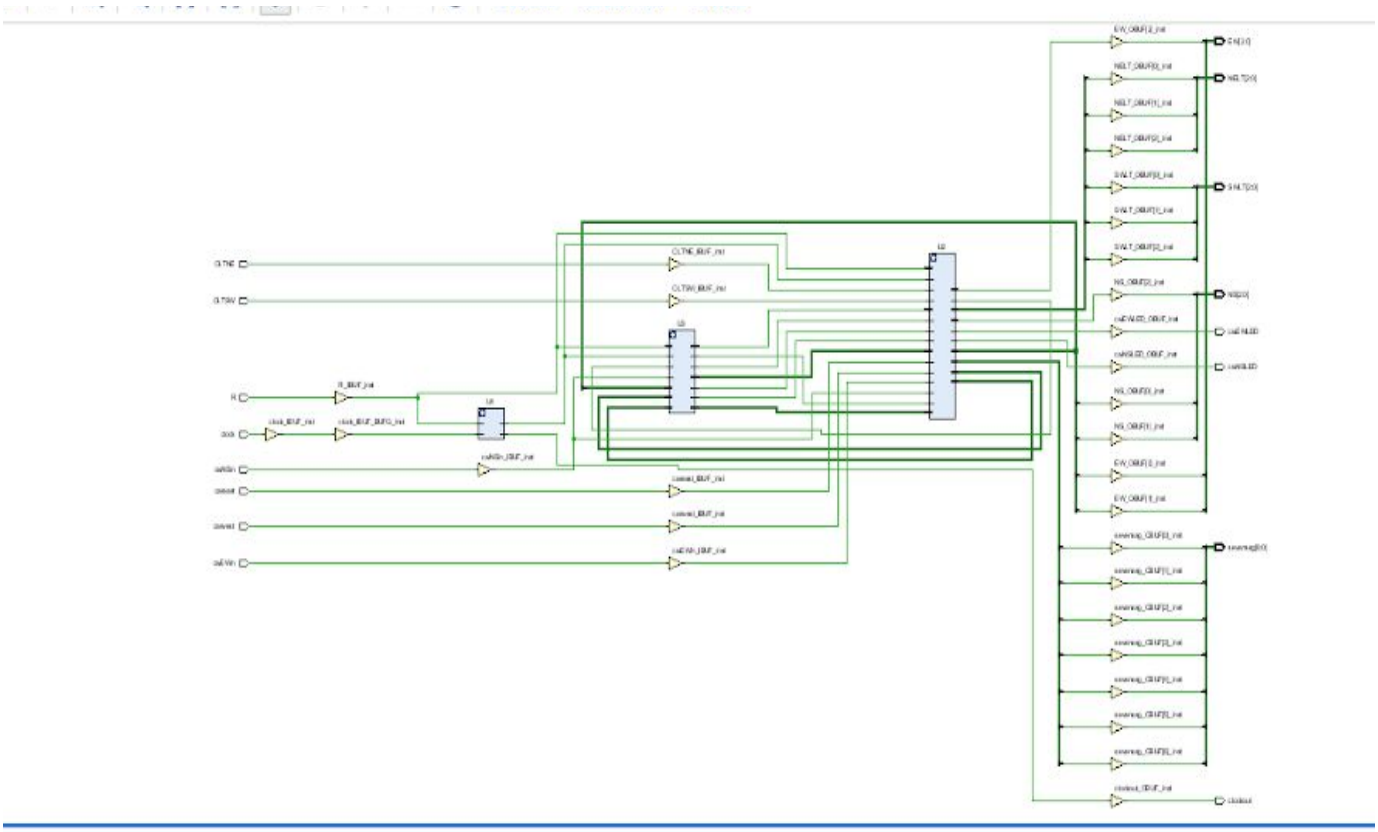

Figure 2.1 This is a schematic of the top level of the traffic light

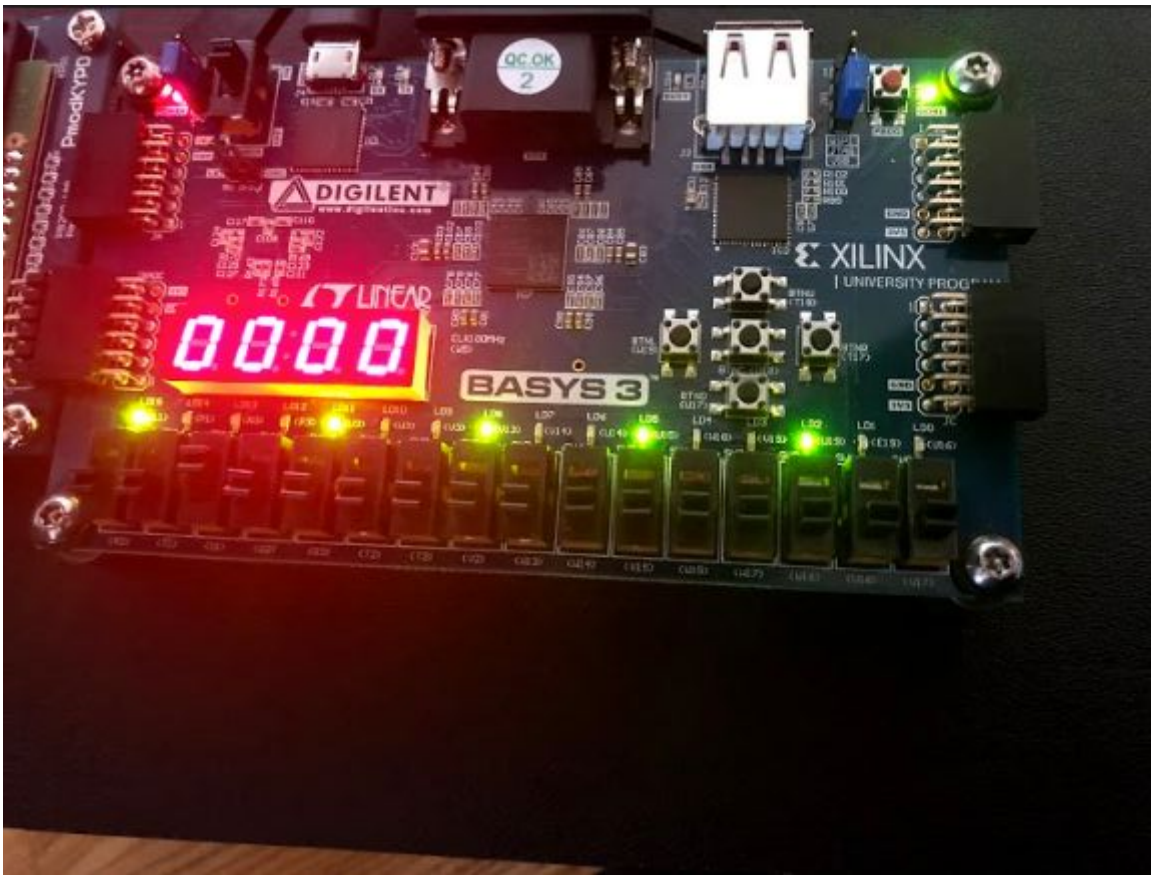

Figure 2.3 This is a picture of a yellow in the turn lanes. The CLTNE is high signifying a car present in the North turn lane therefore EW and NS are both 100 and SWLT and NELT are both 010 signifying a yellow in the turn lanes.

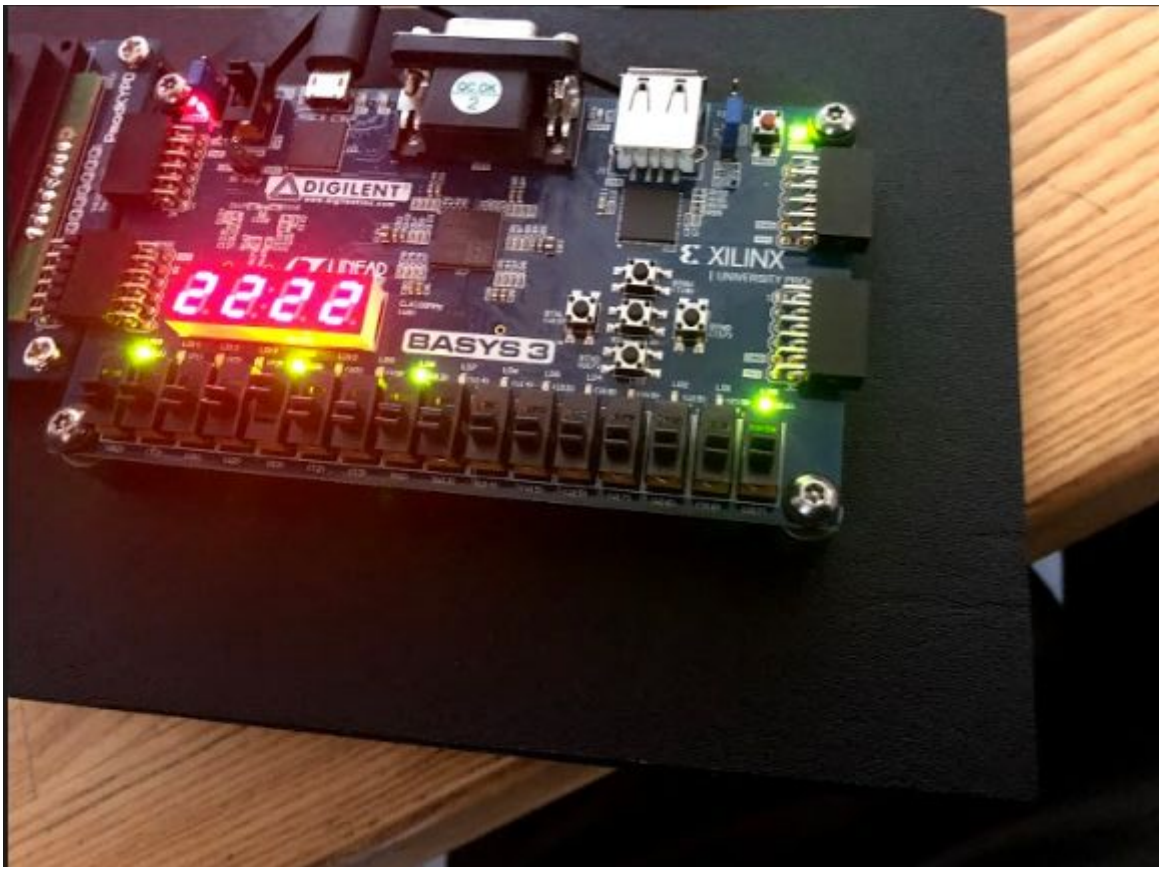

Figure 2.4 This is a picture of the traffic light system with the crosswalk signal in the EW direction high. The crosswalk LED is on (U16) and EW and NS are both 100 which is a red in both cases

Conclusion:

In conclusion, this lab was a fun and engaging experience, which allowed us to fully implement a working traffic controller system that might be seen on a real four way intersection. As in the past few labs, we were able to strengthen our expertise with using components in a top level design to generate a fully working system. While this was a fun assignment to complete, we were met with numerous challenges. One challenge faced was debugging the FSM sequence. Our state machine seemed to constantly get "stuck" in the Red East/West state, and not turning off the crosswalk LED or advancing to the Green East/West state. We overcame this challenge after spending ample time troubleshooting, and setting the LED off as soon as the GEW state was entered. All in all, we are more confident with state machine designs and implementations, and looking forward to the next lab where we will be implementing the vending machine state machine.

Appendices

VHDL for parts of lab

Part 1

TOP LEVEL

```
entity Top is
 Port (--walk: in std_logic_vector (1 downto 0);
       carwest, careast: in std_logic;
       clock, R, turn: in std_logic;
       clockout: out std_logic;
        cwNSLED,cwEWLED: out std_logic;
      -- turnout: out std_logic_vector(1 downto 0); --turn signal and crosswalk
       NS, EW, NELT, SWLT: out std_logic_vector(2 downto 0)); --lights for North South and East
West
end Top;

architecture Behavioral of Top is

signal clkTmp :std_logic;
signal C1Tmp, C2Tmp: std_logic_vector(3 downto 0);
signal selTmp: std_logic_vector(1 downto 0);

--trafficlight component dec
  component TrafficLight
   Port (clk, reset,turn: in std_logic;
   c1, c2: in std_logic_vector(3 downto 0);
    care,carw :in std_logic;
     cwNSLED,cwEWLED: out std_logic;
    --cwin: in std_logic_vector(1 downto 0);
   --cwout: out std_logic_vector(1 downto 0);
   --csel:out std_logic;
   sel: out std_logic_vector(1 downto 0);
   northsouth,eastwest,SWLT, NELT: out std_logic_vector(2 downto 0));
   end component;

   --Counter component dec
   component counter port(clk, reset: in std_logic;
          inp: in std_logic_vector(1 downto 0);

          c1,c2: out std_logic_vector(3 downto 0) );
      end component;
   --component clockdivider declaration
```

```
   component Clockdivider port(clk : in std_logic;
                reset : in std_logic;
           SlowClock, led0 : out std_logic);
           end component;
begin
   --instantiation
   U1: Clockdivider port map(clk=> clock, reset=> R, SlowClock=> clkTmp, led0=>clockout);
   U2: TrafficLight port map(clk=> clkTmp, reset=>R, turn=> turn, c1=> C1Tmp,
c2=>C2Tmp,care=>careast, carw=>carwest,
                           sel=> selTmp,  northsouth=>NS,
eastwest=>EW,SWLT=>SWLT,NELT=>NELT,
                            cwNSLED=>cwNSLED, cwEWLED=> cwEWLED);
   U3: counter port map(clk=>clkTmp, reset=>R, inp=>selTmp, c1=>C1Tmp, c2=>C2Tmp);

end Behavioral;
```

TRAFFIC LIGHT FSM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_arith.all;

entity TrafficLight is
  Port (clk, reset,turn: in std_logic;
       c1, c2, countcross: in std_logic_vector(3 downto 0);
        care,carw :in std_logic; --car in the east west lane
       -- cwin: in std_logic_vector(1 downto 0);-- crosswalk input in NS and EW
       cwNSLED,cwEWLED: out std_logic; --cross walk output in NS and EW
       csel:out std_logic;
       sel: out std_logic_vector(1 downto 0);--sekect bit and turn signal
       northsouth,eastwest,NELT,SWLT: out std_logic_vector(2 downto 0));
end TrafficLight;

architecture Behavioral of TrafficLight is
type state_type is(GNS, YNS,RNS,GEW,YEW, REW, GLT, YLT, RLT,cwEW,cwNS);
signal NS, CS:state_type;
signal walk: std_logic_vector(1 downto 0);
begin
   process(clk, reset)
       begin
       if(reset='1')then
       CS<= GNS;
       elsif (clk'event and clk='1')then
       CS<=NS;
       end if;
       end process;
     process(CS, care, carw)
           begin

          --   crosswalk<="00";
             csel<='0';
             case CS is
             when GNS =>-- green north south
             northsouth <="001"; --green
             eastwest <= "100"; --red
             cwEWLED <= '0';
             cwNSLED <= '0';
             sel <="10"; --stay green for 2 seconds
```

```
            if (c2="0010" and (care='1' or carw='1'))then--or cwin="00" or cwin="01" or cwin ="10"
or cwin="11"))then --no crosswalk input or EW cross input or ignore
             NS <=YNS;
             else
             NS <=CS;
             end if;
   when YNS => --yellow north south
           northsouth <="010"; --yellow
           eastwest <= "100"; --red
           sel <="01";--stay yellow for 1 second
           walk<="10";
           if (c1="0001") then
           NS<=RNS;
           end if;
   when RNS => --red north south
           northsouth <= "100"; --red
           eastwest <= "100"; --red
           sel <= "01"; -- stay red for 1 second
           if (c1="0001")then
              NS<=cwEW;
           end if;
   when cwEW =>
            northsouth <= "100"; --red
             eastwest <= "100"; --red
            cwEWLED <='1';
             sel <="01";
             if (c1="0001")then
             cwEWLED <='0';
             NS <=GEW;
             end if;

    when GEW => --east west green
    -- outputa of this state
            northsouth <= "100"; --red
            eastwest <= "001"; --green
            sel <= "10"; -- stay red for 2 seconds

            if(c2="0010")then
            NS <=YEW;
            end if;
   when YEW => --east west yellow
           northsouth <= "100"; --red
           eastwest <= "010";--yellow
```

```
        sel<= "01";-- count to 1
        if (c1="0001") then
        NS <= REW;
        end if;

  when REW=> --east west red
        --outputs of this state
        northsouth <= "100";-- red
        eastwest <="100";--red
        sel <= "01";--stay red for 1 second
        if (c1="0001")then
        NS <= GLT;
        end if;
  when GLT => --turng green
   --outputs of this state
        northsouth <= "100"; --red
        eastwest <= "100"; --red
        NELT <="001";
        SWLT <= "001";
        sel <= "10"; --stay green for 2 seconds

        if (c2 = "0010") then
           NS <= YLT;
         end if;
  when YLT => -- turn yellow
       --  turnsig <= "01";
        northsouth <= "100"; --red
        eastwest <= "100";-- red
        NELT <="010";
        SWLT <= "010";

        sel <= "01"; --stay yellow for 1 second
        --cwout <= "00"; --no walking
        if (c1 = "0001") then
           NS <=RLT;
        end if;
when RLT=>
     northsouth <= "100"; --red
         eastwest <= "100";-- red
         NELT <="100";
         SWLT <= "100";
         sel<="01";
         if (c1 ="0001") then
```

```
            NS <=cwNS;
            end if;
    when cwNS=>
         northsouth <= "100"; --red
          eastwest <= "100"; --red
             cwNSLED <='1';
             sel <="01";
             if (c1="0001")then
               cwNSLED <='0';
                  NS <=GNS;
                end if;
      end case;
   end process;
end Behavioral;
```

CLOCK DIVIDER

```
entity Clockdivider is
    port(clk : in std_logic;
        reset : in std_logic;
          SlowClock, led0 : out std_logic);
end Clockdivider;

architecture behavioral of Clockdivider is
--signal for clock
signal slowClock_Sig:std_logic;
begin
    process

          variable cnt :   std_logic_vector(26 downto 0):= "00000000000000000000000000";
   --Counter. When this variable reches MSB approximately one second (1 Hz) will have passed
        begin

                wait until ((clk'EVENT) AND (clk = '1')); --wait until clock is high

                if (reset = '1') then
                --reset the counter
                cnt := "00000000000000000000000000";
            else
           --start counting by incrementing by 1
           cnt := std_logic_vector (unsigned (cnt) + 1);
              end if;


          SlowClock <= cnt(26);  --SlowClock = 1 Hz


          slowClock_sig <= cnt(26);--same as slow clock

          --Display the clock info to the LED
          if (SlowClock_sig = '1') then
                    led0 <= '1';
          else
                    led0 <= '0';
          end if;
 end process;
end behavioral;
```

COUNTER

```
entity counter is
  Port (clk, reset: in std_logic;
        inp: in std_logic_vector(1 downto 0);
        c1,c2: out std_logic_vector(3 downto 0) );-- count to 1 and 2 seconds
end counter;

architecture Behavioral of counter is
signal tmp1, tmp2: std_logic_vector (3 downto 0); --counter signals

begin

 process (reset, clk)
 begin
 tmp1<= "0000";
 tmp2 <="0000";
 if (reset ='1')then
 tmp1<= "0000";
 tmp2 <="0000";
    elsif(clk'event and (clk='1')) then
        if (inp ="01") then
        if (tmp1 <2) then --stay in state for 0,1 clock ticks
        tmp1 <= tmp1 +"0001";
        end if;
        elsif (inp="10") then
        if (tmp2 < 3) then --stay in state for 0,1,2 clock ticks
        tmp2 <= tmp2 + "0001";
        end if;
       end if;

      end if;
      c1<= tmp1;
      c2<= tmp2;

     end process;

end Behavioral;
```

Constraints

```
set_property IOSTANDARD LVCMOS33 [get_ports {EW[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {EW[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {EW[0]}]
set_property PACKAGE_PIN L1 [get_ports {EW[2]}]
set_property PACKAGE_PIN P1 [get_ports {EW[1]}]
set_property PACKAGE_PIN N3 [get_ports {EW[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[0]}]
set_property PACKAGE_PIN U3 [get_ports {NS[2]}]
set_property PACKAGE_PIN W3 [get_ports {NS[1]}]
set_property PACKAGE_PIN V3 [get_ports {NS[0]}]


set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clockout]
set_property PACKAGE_PIN V13 [get_ports clockout]
set_property IOSTANDARD LVCMOS33 [get_ports R]
set_property IOSTANDARD LVCMOS33 [get_ports turn]
set_property PACKAGE_PIN T2 [get_ports R]
set_property PACKAGE_PIN T3 [get_ports turn]

set_property IOSTANDARD LVCMOS33 [get_ports {NELT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NELT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NELT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports careast]
set_property IOSTANDARD LVCMOS33 [get_ports carwest]
set_property IOSTANDARD LVCMOS33 [get_ports cwEWLED]
set_property IOSTANDARD LVCMOS33 [get_ports cwNSLED]
set_property PACKAGE_PIN U15 [get_ports {NELT[0]}]
set_property PACKAGE_PIN U14 [get_ports {NELT[1]}]
set_property PACKAGE_PIN V14 [get_ports {NELT[2]}]
set_property PACKAGE_PIN U19 [get_ports {SWLT[0]}]
set_property PACKAGE_PIN V19 [get_ports {SWLT[1]}]
set_property PACKAGE_PIN W18 [get_ports {SWLT[2]}]
set_property PACKAGE_PIN R2 [get_ports careast]
```

```
set_property PACKAGE_PIN T1 [get_ports carwest]
set_property PACKAGE_PIN U16 [get_ports cwEWLED]
set_property PACKAGE_PIN E19 [get_ports cwNSLED]
```

Part 2

TOP LEVEL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Top is
 Port (carwest, careast: in std_logic;
       clock, R: in std_logic;
       clockout: out std_logic;
        cwNSLED,cwEWLED: out std_logic;
        cwNSin, cwEWin: in std_logic;
        CLTNE, CLTSW: in std_logic;
       NS, EW, NELT, SWLT: out std_logic_vector(2 downto 0);
       sevenseg: out std_logic_vector(6 downto 0)); --lights for North South and East West
end Top;

architecture Behavioral of Top is

signal clkTmp :std_logic;
signal C1Tmp, C2Tmp: std_logic_vector(3 downto 0);
signal displayTmp:std_logic_vector(3 downto 0);
signal selTmp: std_logic_vector(1 downto 0);
```

```
--trafficlight component dec
  component TrafficLight
   Port (clk, reset: in std_logic;
         c1, c2: in std_logic_vector(3 downto 0);
          care,carw :in std_logic; --car in the east west lane
         cwNSin,cwEWin: in std_logic;-- crosswalk input in NS and EW
         CLTNE, CLTSW: in std_logic;-- left turn input for left from N->E and left from S->W
         cwNSLED,cwEWLED: out std_logic; --cross walk output in NS and EW
         sevensegOut: out std_logic_vector(3 downto 0);
         csel:out std_logic;
         sel: out std_logic_vector(1 downto 0);--select bit
         northsouth,eastwest,NELT,SWLT: out std_logic_vector(2 downto 0));
   end component;

   --Counter component dec
   component counter port(clk, reset: in std_logic;
          inp: in std_logic_vector(1 downto 0);

          c1,c2: out std_logic_vector(3 downto 0) );
      end component;
   --component clockdivider declaration
   component Clockdivider port(clk : in std_logic;
               reset : in std_logic;
          SlowClock, led0 : out std_logic);
          end component;
  component display port(inp: in std_logic_vector (3 downto 0);
                    output: out std_logic_vector(6 downto 0) );
          end component;
begin
   --instantiation
   U1: Clockdivider port map(clk=> clock, reset=> R, SlowClock=> clkTmp, led0=>clockout);
   U2: TrafficLight port map(clk=> clkTmp, reset=>R, c1=> C1Tmp, c2=>C2Tmp,care=>careast,
carw=>carwest,
                          sel=> selTmp,  northsouth=>NS,
eastwest=>EW,SWLT=>SWLT,NELT=>NELT,
                           cwNSLED=>cwNSLED, cwEWLED=> cwEWLED, cwNSin=> cwNSin,
cwEWin=>cwEWin,
                           CLTNE=> CLTNE, CLTSW=>CLTSW, sevensegOut=> displayTmp);
   U3: counter port map(clk=>clkTmp, reset=>R, inp=>selTmp, c1=>C1Tmp, c2=>C2Tmp);
   U4: display port map(inp=> displayTmp, output=>sevenseg);
end Behavioral;
```

COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity counter is
  Port (clk, reset: in std_logic;
        inp: in std_logic_vector(1 downto 0);
        c1,c2: out std_logic_vector(3 downto 0) );-- count to 1 and 2 seconds
end counter;

architecture Behavioral of counter is
signal tmp1, tmp2: std_logic_vector (3 downto 0); --counter signals

begin

 process (reset, clk)
 begin
 tmp1<= "0000";
 tmp2 <="0000";
 if (reset ='1')then
 tmp1<= "0000";
 tmp2 <="0000";
    elsif(clk'event and (clk='1')) then
        if (inp ="01") then
        if (tmp1 <2) then --stay in state for 0,1 clock ticks
        tmp1 <= tmp1 +"0001";
        end if;
        elsif (inp="10") then
```

```
        if (tmp2 < 3) then --stay in state for 0,1,2 clock ticks
        tmp2 <= tmp2 + "0001";
        end if;
       end if;

      end if;
      c1<= tmp1;
      c2<= tmp2;

     end process;

end Behavioral;
```

DISPLAY

CLOCK DIVIDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.All;

entity Clockdivider is
    port(clk : in std_logic;
         reset : in std_logic;
          SlowClock, led0 : out std_logic);
end Clockdivider;

architecture behavioral of Clockdivider is
--signal for clock
signal slowClock_Sig:std_logic;
begin
    process
          variable cnt :   std_logic_vector(26 downto 0):= "00000000000000000000000000";
   --Counter. When this variable reches MSB approximately one second (1 Hz) will have passed
        begin

                wait until ((clk'EVENT) AND (clk = '1')); --wait until clock is high
```

```
                if (reset = '1') then
                --reset the counter
                cnt := "00000000000000000000000000";
              else
            --start counting by incrementing by 1
            cnt := std_logic_vector (unsigned (cnt) + 1);
              end if;


        SlowClock <= cnt(26);  --SlowClock = 1 Hz


        slowClock_sig <= cnt(26);--same as slow clock

        --Display the clock info to the LED
        if (SlowClock_sig = '1') then
                led0 <= '1';
        else
                led0 <= '0';
        end if;
 end process;
end behavioral;
```

CONSTRAINTS

```
set_property IOSTANDARD LVCMOS33 [get_ports {EW[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {EW[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {EW[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NELT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NELT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NELT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {NS[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWLT[0]}]
set_property PACKAGE_PIN L1 [get_ports {EW[2]}]
set_property PACKAGE_PIN P1 [get_ports {EW[1]}]
set_property PACKAGE_PIN N3 [get_ports {EW[0]}]
set_property PACKAGE_PIN V3 [get_ports {NS[0]}]
set_property PACKAGE_PIN W3 [get_ports {NS[1]}]
set_property PACKAGE_PIN U3 [get_ports {NS[2]}]

set_property PACKAGE_PIN V14 [get_ports {NELT[2]}]
```

```
set_property PACKAGE_PIN U14 [get_ports {NELT[1]}]
set_property PACKAGE_PIN U15 [get_ports {NELT[0]}]
set_property PACKAGE_PIN U19 [get_ports {SWLT[0]}]
set_property PACKAGE_PIN V19 [get_ports {SWLT[1]}]
set_property PACKAGE_PIN W18 [get_ports {SWLT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports clockout]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports carwest]
set_property IOSTANDARD LVCMOS33 [get_ports careast]
set_property IOSTANDARD LVCMOS33 [get_ports CLTNE]
set_property IOSTANDARD LVCMOS33 [get_ports CLTSW]
set_property IOSTANDARD LVCMOS33 [get_ports cwEWLED]
set_property IOSTANDARD LVCMOS33 [get_ports cwEWin]
set_property IOSTANDARD LVCMOS33 [get_ports cwNSin]
set_property IOSTANDARD LVCMOS33 [get_ports cwNSLED]
set_property IOSTANDARD LVCMOS33 [get_ports R]

set_property PACKAGE_PIN R2 [get_ports careast]
set_property PACKAGE_PIN T1 [get_ports carwest]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property PACKAGE_PIN V13 [get_ports clockout]
set_property PACKAGE_PIN U1 [get_ports CLTNE]
set_property PACKAGE_PIN W2 [get_ports CLTSW]
set_property PACKAGE_PIN R3 [get_ports cwEWin]
set_property PACKAGE_PIN U16 [get_ports cwEWLED]
set_property PACKAGE_PIN E19 [get_ports cwNSLED]
set_property PACKAGE_PIN T2 [get_ports cwNSin]
set_property PACKAGE_PIN T3 [get_ports R]

set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sevenseg[0]}]
set_property PACKAGE_PIN W7 [get_ports {sevenseg[6]}]
set_property PACKAGE_PIN V5 [get_ports {sevenseg[5]}]
set_property PACKAGE_PIN U5 [get_ports {sevenseg[4]}]
set_property PACKAGE_PIN V8 [get_ports {sevenseg[3]}]
set_property PACKAGE_PIN U8 [get_ports {sevenseg[2]}]
set_property PACKAGE_PIN W6 [get_ports {sevenseg[1]}]
set_property PACKAGE_PIN U7 [get_ports {sevenseg[0]}]
```