ECEN 429: Introduction to Digital Systems Design Laboratory

North Carolina Agricultural and Technical State University

Department of Electrical and Computer Engineering

Ian Parker (Reporter)

Tayanna Lee (Lab Partner)

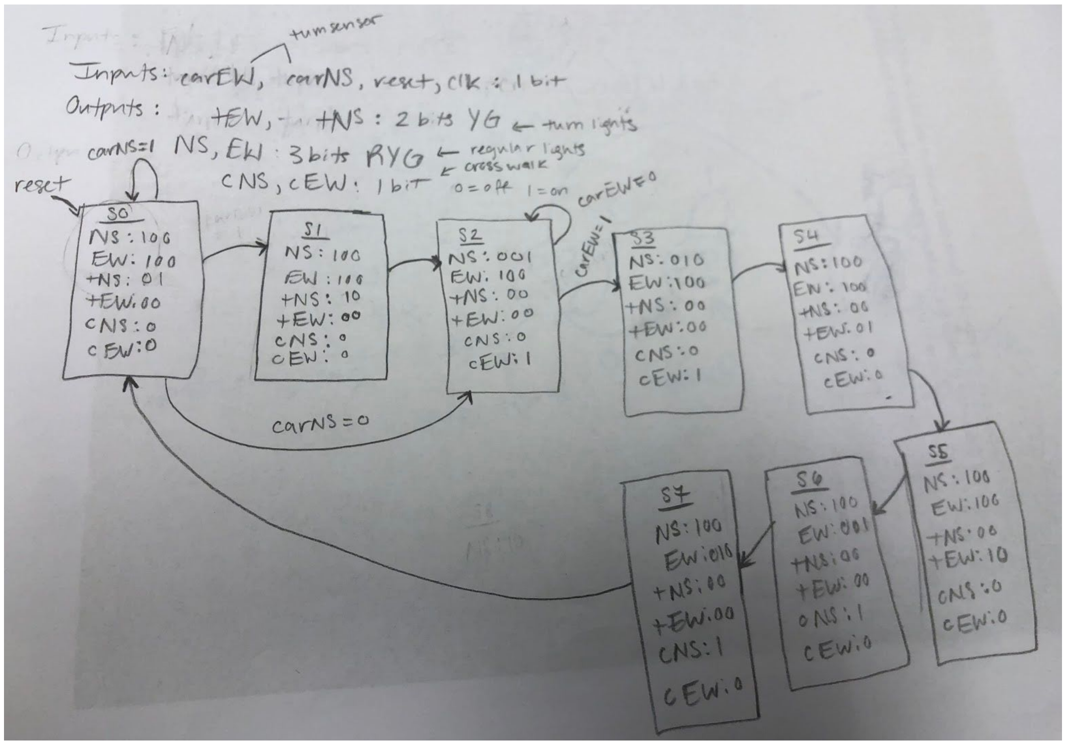
April 18, 2019

Lab #9

Introduction: For lab 9, we continue to implement FSM. This time we used the function of a finite state machine to control traffic lights. The components include: North/South lights, East/West lights, a N/S and E/W turn lane light and sensor and a crosswalk (w/o buttons in Part 1 and with buttons for Part 2).

Background

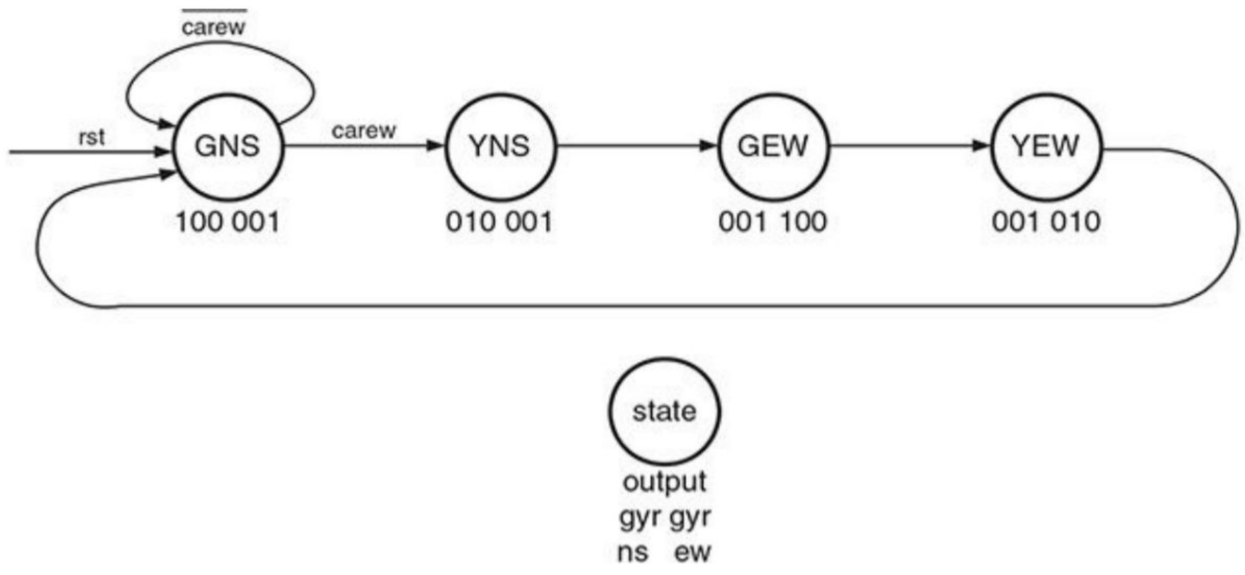
Overall Finite State Machine for traffic light implementation:



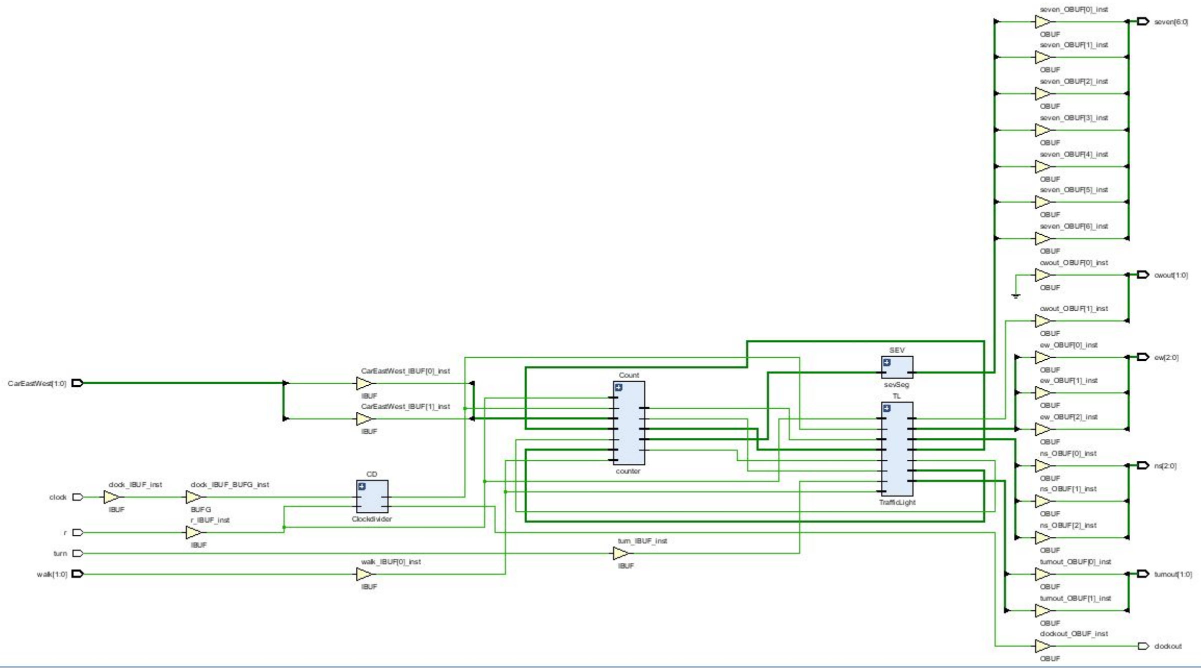
turn sensor
Inputs: carEW, carNS, reset, clk : 1 bit
Outputs : tEW, tNS : 2 bits YG ← turn lights
NS, EW : 3 bits RYG ← regular lights
← crosswalk
CNS, cEW : 1 bit 0 = off 1 = on
reset
carNS=1
carEW=0
carEW=1
carNS=0
S0 NS : 100 EW: 100 tNS: 01 tEW: 00 CNS: 0 cEW: 0
S1 NS: 100 EW: 100 tNS: 10 tEW: 00 CNS: 0 cEW: 0
S2 NS: 001 EW: 100 tNS: 00 tEW: 00 CNS: 0 cEW: 1
S3 NS: 010 EW: 100 tNS: 00 tEW: 00 CNS: 0 cEW: 1
S4 NS: 100 EW: 100 tNS: 00 tEW: 01 CNS: 0 cEW: 0
S5 NS: 100 EW: 100 tNS: 00 tEW: 10 CNS: 0 cEW: 0
S6 NS: 100 EW: 001 tNS: 00 tEW: 00 CNS: 1 cEW: 0
S7 NS: 100 EW: 010 tNS: 00 tEW: 00 CNS: 1 cEW: 0

Part 1

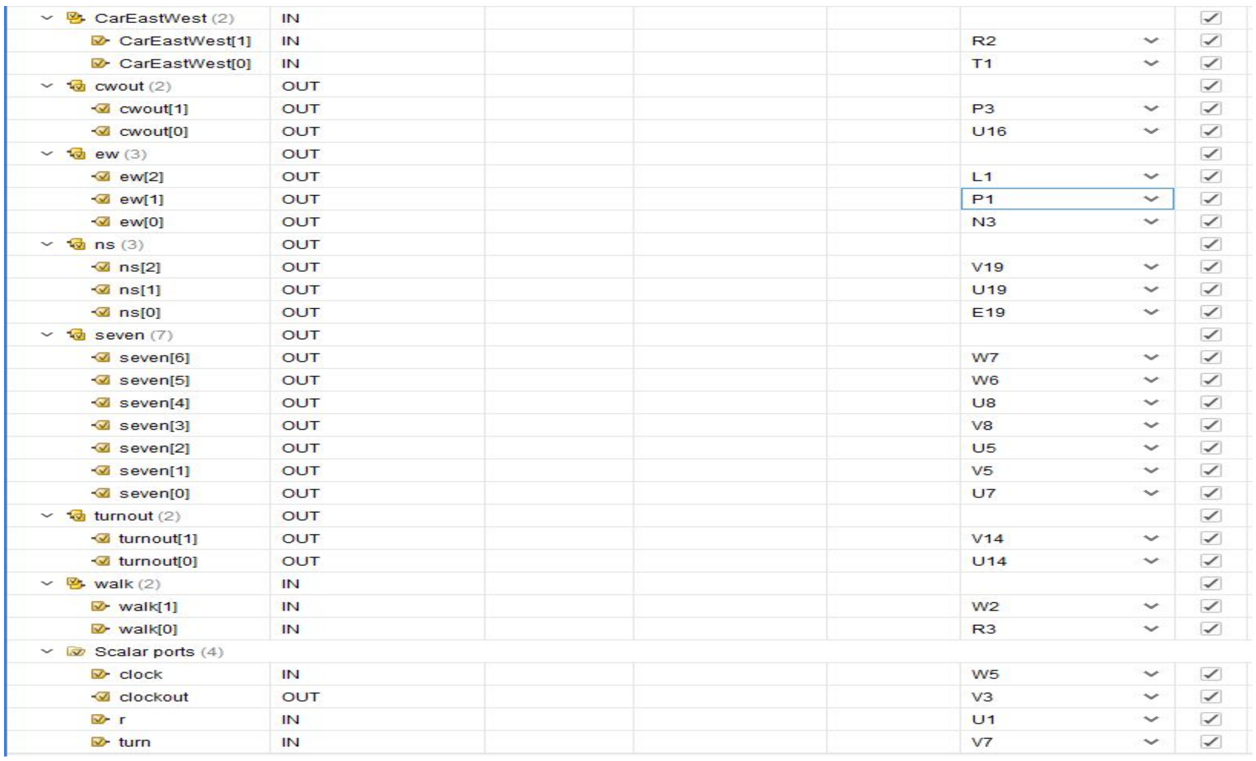
State Diagram



Schematic



Pin Assignments



CarEastWest (2)	IN		✓
CarEastWest[1]	IN	R2	✓
CarEastWest[0]	IN	T1	✓
cwout (2)	OUT		✓
cwout[1]	OUT	P3	✓
cwout[0]	OUT	U16	✓
ew (3)	OUT		✓
ew[2]	OUT	L1	✓
ew[1]	OUT	P1	✓
ew[0]	OUT	N3	✓
ns (3)	OUT		✓
ns[2]	OUT	V19	✓
ns[1]	OUT	U19	✓
ns[0]	OUT	E19	✓
seven (7)	OUT		✓
seven[6]	OUT	W7	✓
seven[5]	OUT	W6	✓
seven[4]	OUT	U8	✓
seven[3]	OUT	V8	✓
seven[2]	OUT	U5	✓
seven[1]	OUT	V5	✓
seven[0]	OUT	U7	✓
turnout (2)	OUT		✓
turnout[1]	OUT	V14	✓
turnout[0]	OUT	U14	✓
walk (2)	IN		✓
walk[1]	IN	W2	✓
walk[0]	IN	R3	✓
Scalar ports (4)			
clock	IN	W5	✓
clockout	OUT	V3	✓
r	IN	U1	✓
turn	IN	V7	✓

Results

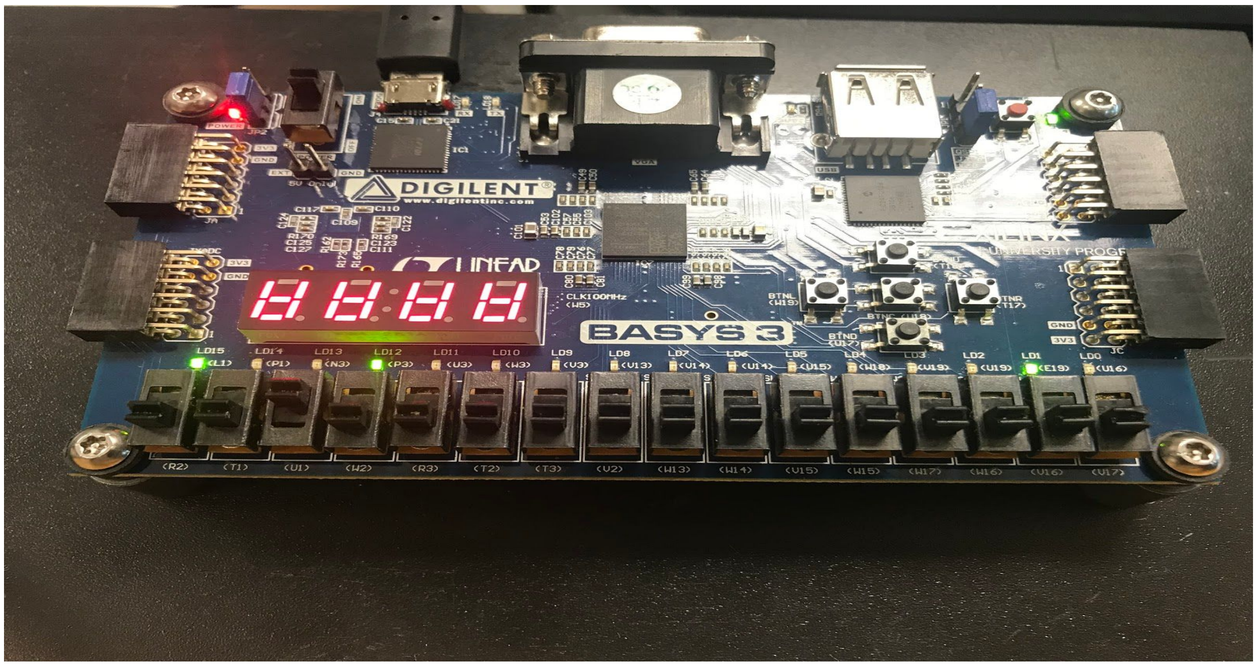


Figure 1.1) Reset State to North South on Green

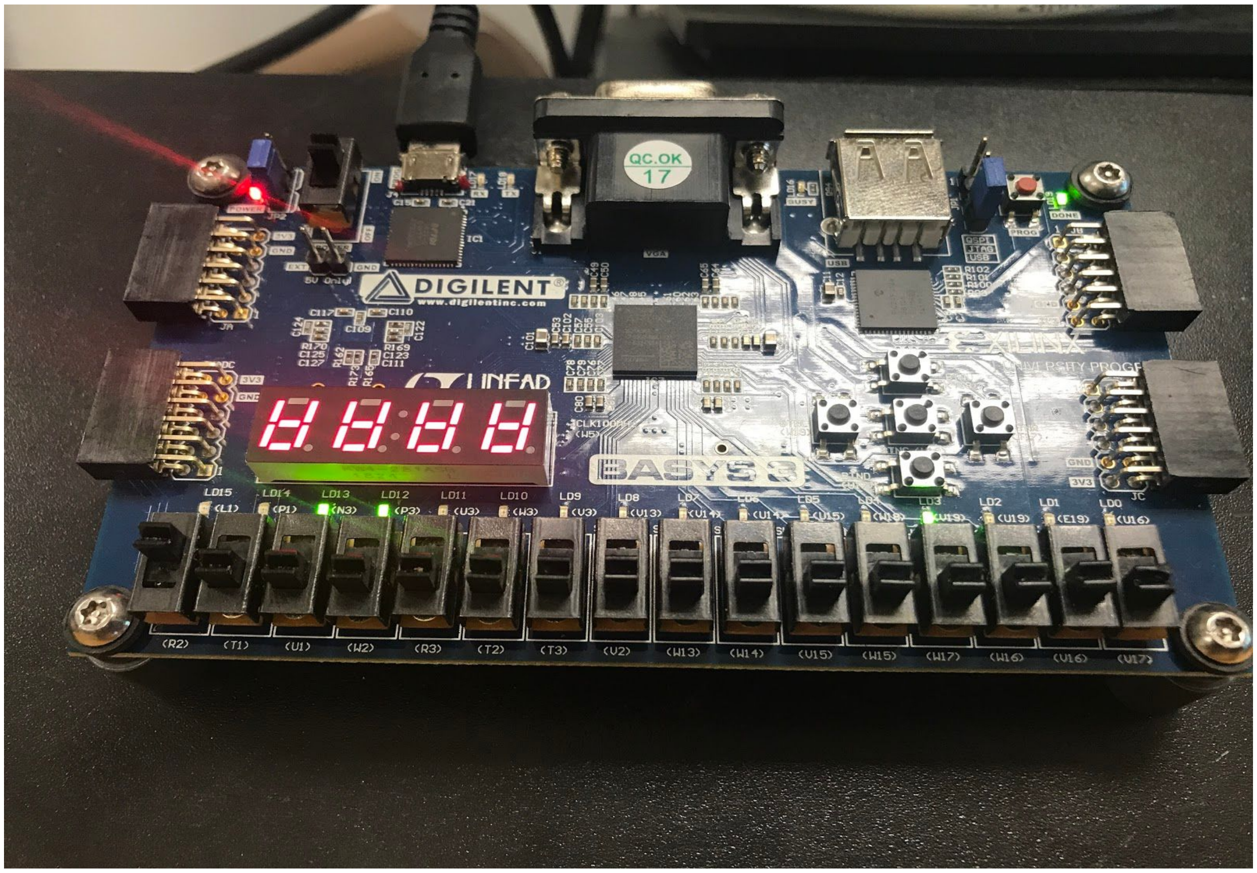
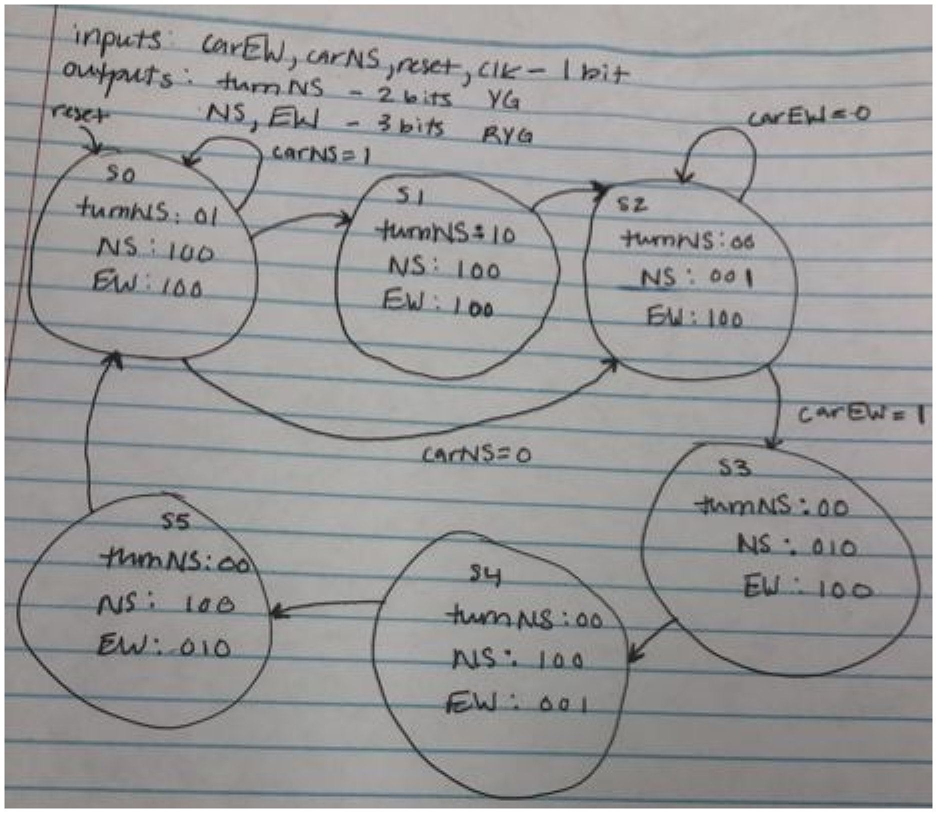


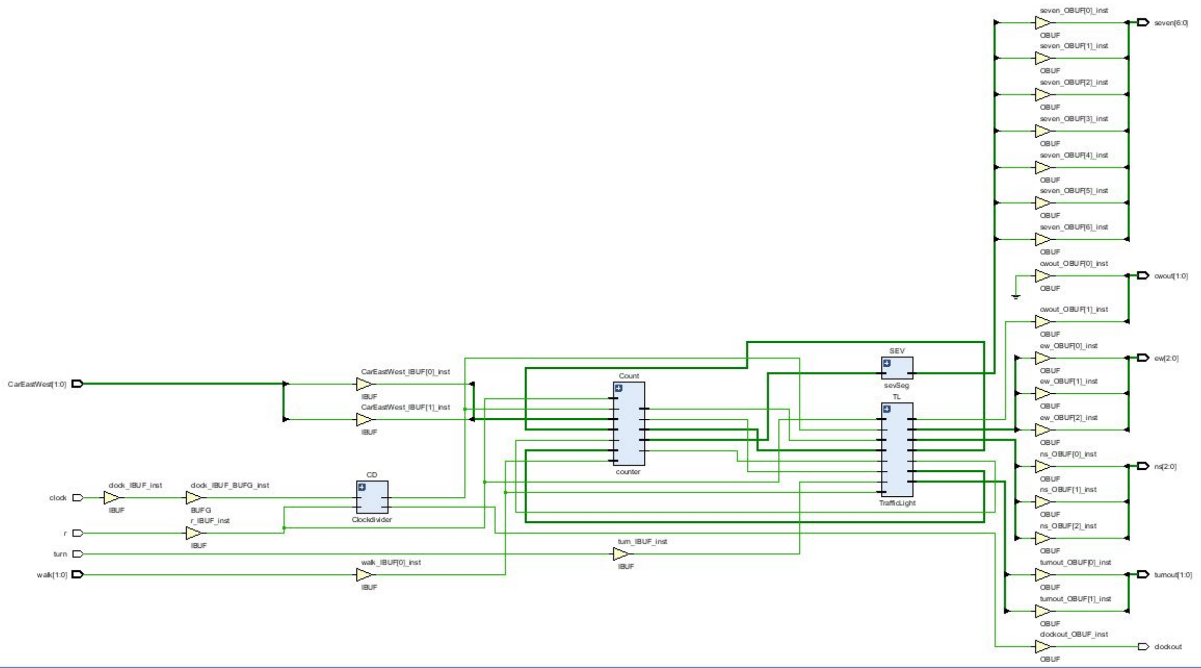
Figure 1.2) A car is in the east west lane triggering a signal to turn North South light from Green->Yellow->Red

Part2

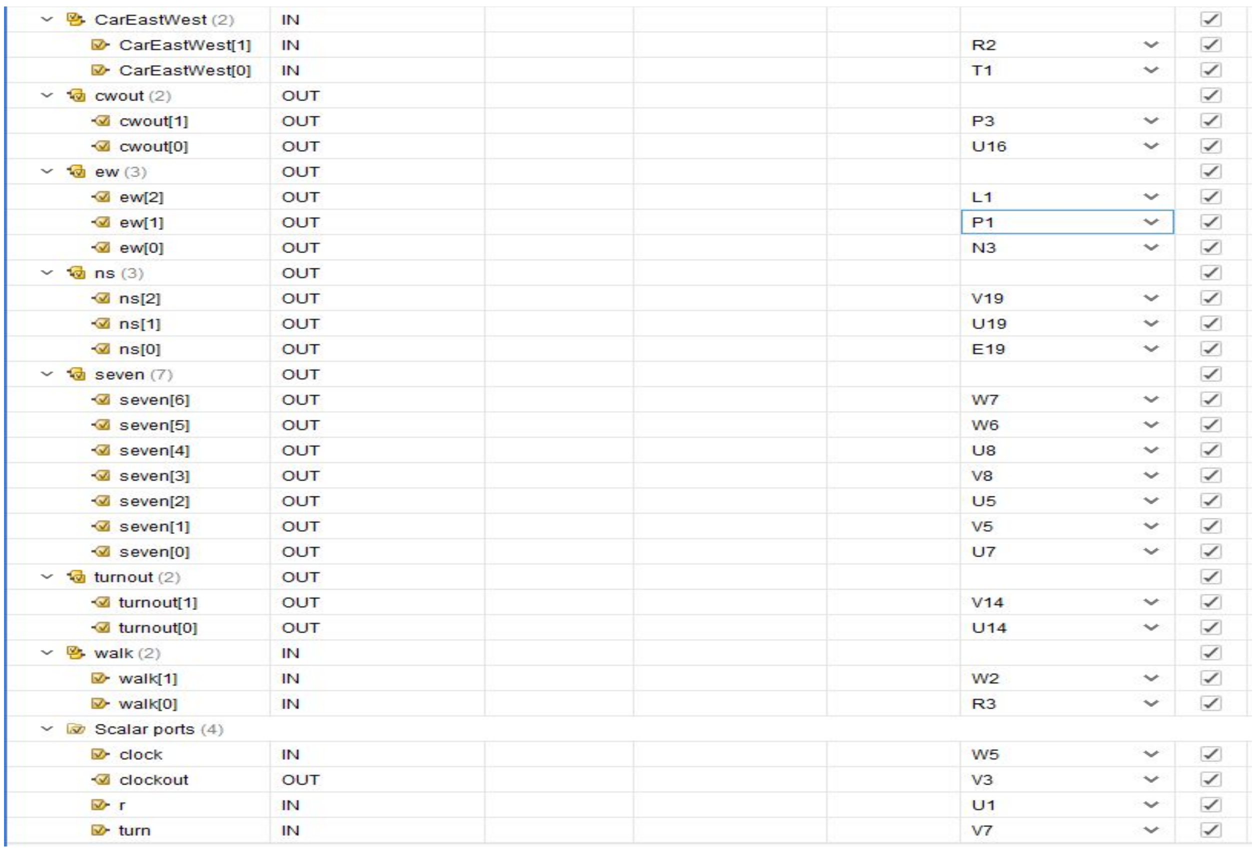
State Diagram



Schematic



Pin Assignments



CarEastWest (2)	IN					✓
CarEastWest[1]	IN				R2	✓
CarEastWest[0]	IN				T1	✓
cwout (2)	OUT					✓
cwout[1]	OUT				P3	✓
cwout[0]	OUT				U16	✓
ew (3)	OUT					✓
ew[2]	OUT				L1	✓
ew[1]	OUT				P1	✓
ew[0]	OUT				N3	✓
ns (3)	OUT					✓
ns[2]	OUT				V19	✓
ns[1]	OUT				U19	✓
ns[0]	OUT				E19	✓
seven (7)	OUT					✓
seven[6]	OUT				W7	✓
seven[5]	OUT				W6	✓
seven[4]	OUT				U8	✓
seven[3]	OUT				V8	✓
seven[2]	OUT				U5	✓
seven[1]	OUT				V5	✓
seven[0]	OUT				U7	✓
turnout (2)	OUT					✓
turnout[1]	OUT				V14	✓
turnout[0]	OUT				U14	✓
walk (2)	IN					✓
walk[1]	IN				W2	✓
walk[0]	IN				R3	✓
Scalar ports (4)						
clock	IN				W5	✓
clockout	OUT				V3	✓
r	IN				U1	✓
turn	IN				V7	✓

Results

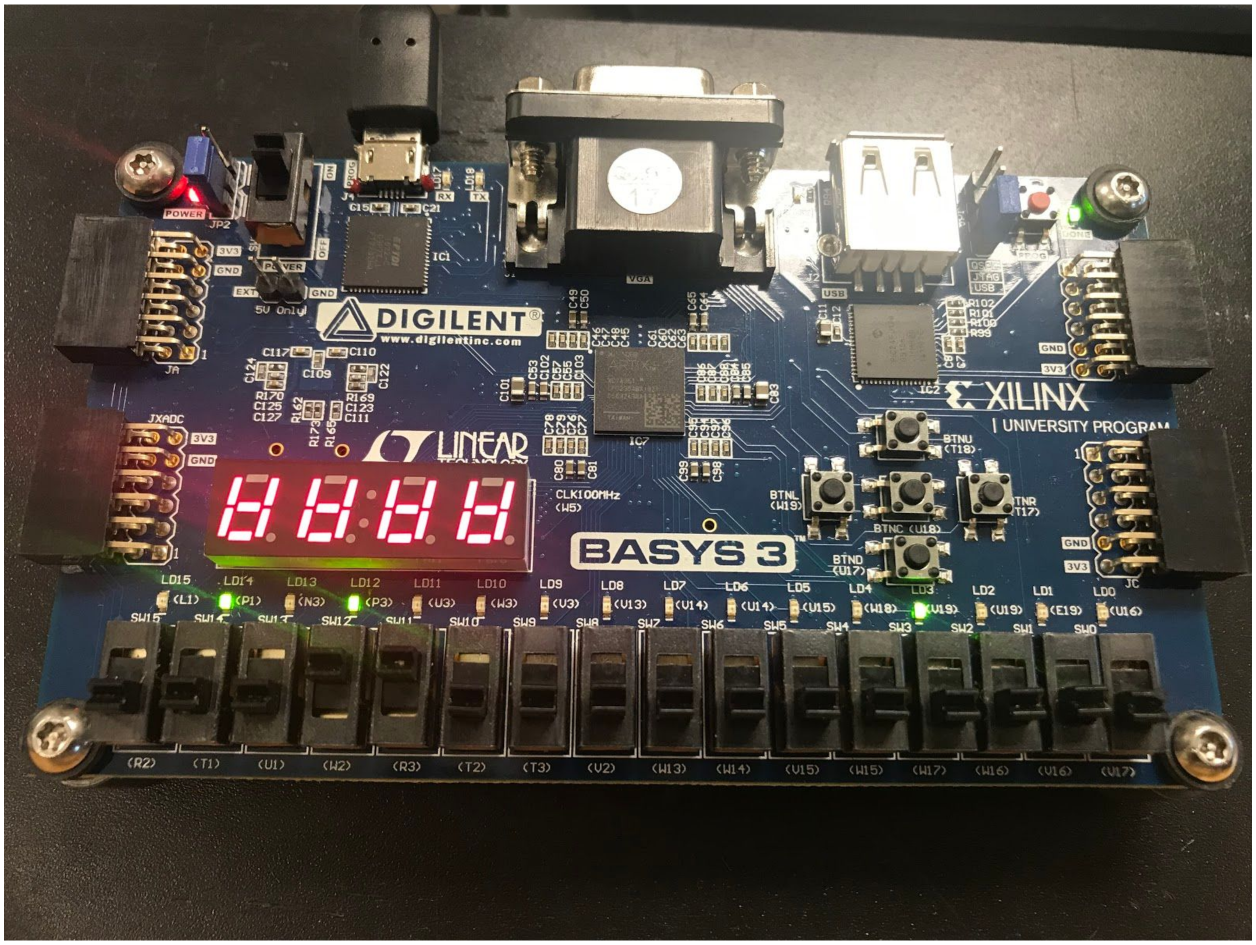


Figure 2.1) Crosswalk button has been pressed and count yet to start because there is still traffic

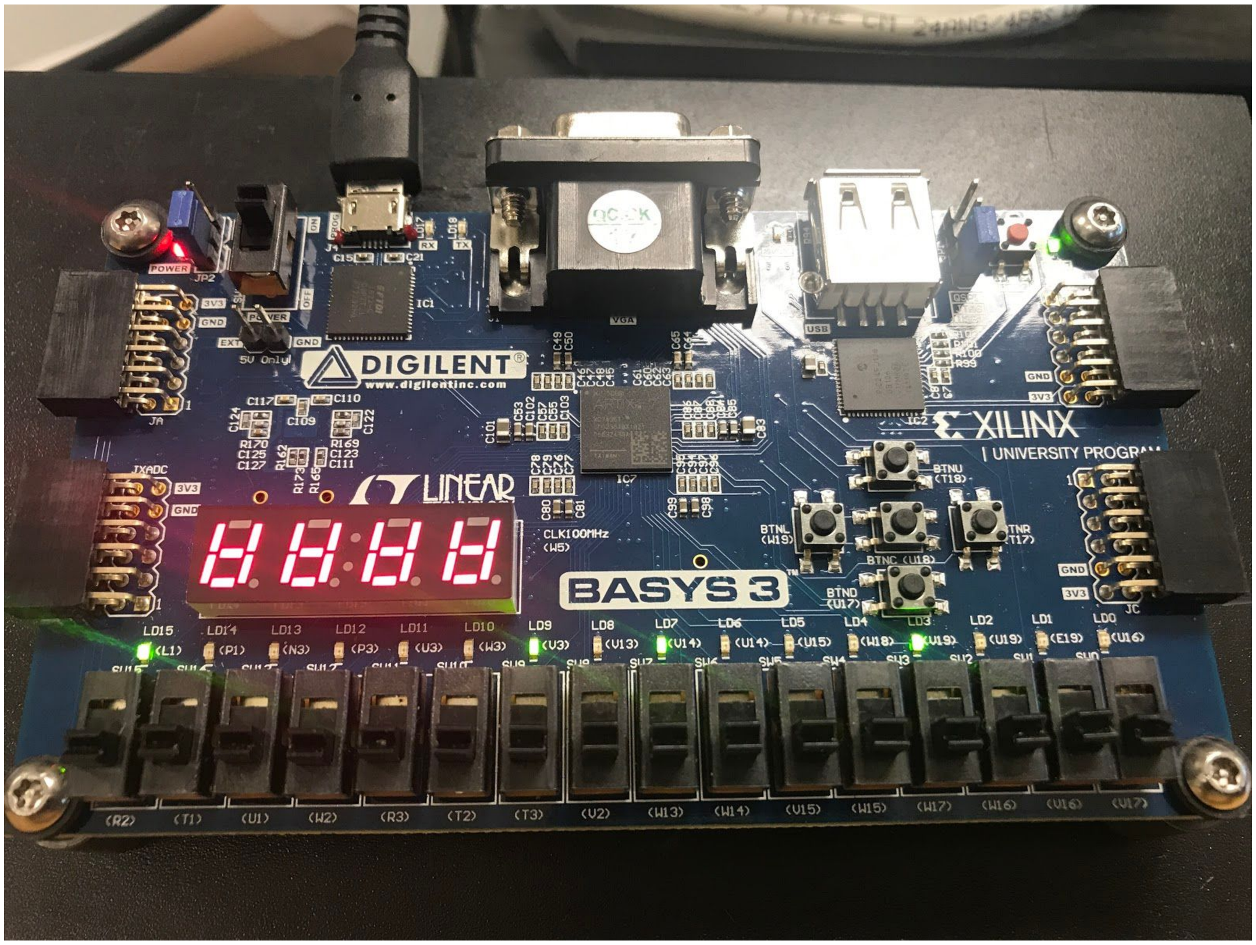


Figure 2.2) No input signals from east west or crosswalk so light as remains at North South Green

Conclusion: With this lab, we continued our deep learning about finite state machines and how to perform advance implementations. As mentioned before, we used a FSM to control traffic lights that include turning sensors and crosswalk buttons. We learn how to implement the FSM, Seven-segment display, and clock divider.

Appendices :

Part 1 VHDL

TopLevel

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

```vhdl
entity TopLevel is
  Port (CarEastWest,walk: in std_logic_vector(1 downto 0);
        clock,r,turn:in std_logic;
        clockout: out std_logic;
        turnout, cwout: out std_logic_vector(1 downto 0);--turn signal and cross walk
        ns,ew:out std_logic_vector(2 downto 0);--lights for NortSouth and EastWest
        seven : out std_logic_vector(6 downto 0));--seven seg display
end TopLevel;

architecture Behavioral of TopLevel is
signal tmpclk,tmpcross: std_logic;
signal tmpC1,tmpC5,crossOut,countcrosstmp: std_logic_vector(3 downto 0);
signal tmpSel,tmpwalk: std_logic_vector(1 downto 0);

    --Traffic Light Component dec
    component TrafficLight port (clk,reset,turn:in std_logic;
                              c1,c5,countcross:in std_logic_vector(3 downto 0);--counters to one second and 5 seconds
                              cew: in std_logic_vector(1 downto 0);--car in the east west lane signal
                              cwin : in std_logic_vector(1 downto 0);--cross walk input (button pressed) in NS and EW
                              cwout: out std_logic_vector(1 downto 0);-- cross walk output (g or not) in NS and EW
                              csel : out std_logic;
                              sel,turnsig,crosswalk: out std_logic_vector(1 downto 0);--select bit and turn signal
                              northsouth,eastwest:out std_logic_vector(2 downto 0));

    end component;
    --Counter Component dec
    component counter port(clk,reset:in std_logic;
                        input:in std_logic_vector(1 downto 0);
                        crosser : in std_logic;
                        c1,c5,crossOut:out std_logic_vector(3 downto 0));
    end component;
    --ClockDivider Component dec
    component clockdivider port(clk : in  STD_LOGIC;
                             reset : in  STD_LOGIC;
                             SlowClock,ledO : out  STD_LOGIC);
    end component;
    --Seven Segment Display Component dec
    component sevSeg port(input: in std_logic_vector (3 downto 0);
                             output:out std_logic_vector(6 downto 0));
    end component;

begin
    --Instantiation
    CD: clockdivider port map(clk=>clock,reset=>r,SlowClock=>tmpclk,ledO=>clockout);
    TL: TrafficLight port
map(clk=>tmpclk,reset=>r,turn=>turn,c1=>tmpc1,c5=>tmpc5,countcross=>countcrosstmp,cew=>CarEastWest,cwi
n=>walk,cwout=>cwout,csel=>tmpcross,sel=>tmpSel,turnsig=>turnout,northsouth=>ns,eastwest=>ew);
```

```
    Count: counter port map (clk=>tmpclk,reset=>r,input=>tmpSel,crosser=>tmpcross,
c1=>tmpc1,c5=>tmpc5,crossout=>countcrosstmp);
    SEV: sevSeg port map (input=>countcrosstmp, output=>seven);


end Behavioral;
```

Clock Divider

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

```
entity Clockdivider is
Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           SlowClock,ledO : out  STD_LOGIC);
end Clockdivider;

architecture behavioral of Clockdivider is
signal slowSig: std_logic;
begin
     process
      variable cnt :    std_logic_vector(26 downto 0):= "000000000000000000000000000";
          begin                            -- calculations
            wait until ((clk'EVENT) AND (clk = '1'));
            if (reset = '1') then
                  cnt := "000000000000000000000000000";
             else
               cnt := cnt + 1; -- count to 26
             end if;

        SlowClock <= cnt(26);
        slowSig<=cnt(26);

      if(slowSig='1')then
        ledO<='1';
      else
        ledO<='0';
      end if;

   end process;
end Behavioral;
```

Traffic Light

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

```
entity TrafficLight is
  Port (clk,reset,turn:in std_logic;
        c1,c5,countcross:in std_logic_vector(3 downto 0);--counters to one second and 5 seconds and for the cross
walk
        cew: in std_logic_vector(1 downto 0);--car in the east west lane signal
        cwin : in std_logic_vector(1 downto 0);--cross walk input (button pressed) in NS and EW
        cwout: out std_logic_vector(1 downto 0);-- cross walk output (g or not) in NS and EW
        csel : out std_logic;
        sel,turnsig,crosswalk : out std_logic_vector(1 downto 0);--select bit and turn signal
        northsouth,eastwest:out std_logic_vector(2 downto 0));

end TrafficLight;

architecture Beh of TrafficLight is
    type state_type is (nsg,nsy,nsr,ewr,ewg,ewy,tg,ty);
    signal cs,ns: state_type;
    signal walk : std_logic_vector(1 downto 0);

    begin
    process(clk,reset)
    begin
        if (reset='1')then
            cs<=nsg;
        elsif(clk'event and clk='1')then
           cs<=ns;
        end if;
    end process;

    process(cs,cew)
    begin
        turnsig <="00";
        crosswalk<= "00";
        csel <= '0';
        case cs is
            when nsg =>--north south green
                        --outputs of this state
                        northsouth <= "001";--green
                        eastwest <= "100";--red

                        sel<="10"; --count to 5
                        if(cwin = "01" or cwin = "11") then
                         cSel<= '1';
                        end if;
                        if(c5="0101" and (cew="01" or cew="10" or cew="11" or cwin = "00" or cwin ="01" or cwin="10" or
cwin="11")) then-- no crosswalk input or east west cross input or ignore
                            ns<=nsy;
                        else
                            ns<=cs;
                        end if;

            when nsy =>--north south yellow
                        --outputs of this state
                        northsouth <= "010";--yellow
                        eastwest <= "100";--red
                        sel<="01";  --count to 1
                        --cwout<= "01";--walk in NS
                        walk<= "10";
                        if(c1="0001") then
                            ns <= nsr;
                        end if;
            when nsr =>--north south red
                        --outputs of this state
                        northsouth <= "100";--red
                        eastwest <= "100";--red
                        sel<="01";  --count to 1
                        --cwout<= "00";-- no walkling
                        if (c1 = "0001" and (cew="01" or cew ="10" or cew="11" or cwin = "00" or cwin ="01" or cwin="10"
or cwin="11" ))then
                            ns <= ewg;
                        elsif(c1="0001" and turn = '1' and cew="00") then
                            ns<= tg;
                        end if;

            when ewg =>--east west green
                         --outputs of this state
                         northsouth <= "100";--red
                         eastwest <= "001";--green
                         sel<="10";  --count to 5
                         walk<="01";
                         --cwout<= "01"; --walk in EW
                         if(walk<="01" and (cwin = "01" or cwin = "11")) then -- no cross input or north south cross input or
ignore
                 csel<='1';
                 crosswalk<="01";
                            if(countcross="0110" )then
                 crosswalk<= "01";
                            end if;
                         end if;

                         if(c5="0101" and (cew="00" or cwin="01" or cwin="11")) then
                            ns<=ewy;
                           else
                            ns<=cs;
                           end if;
            when ewy =>--easy west yellow
                        --outputs of this state
```

```
                    northsouth <= "100";--red
                    eastwest <= "010";--yellow
                    sel<="01";  --count to 1
                    cwout<= "10"; --walk in ew
                    if(c1="0001") then
                        ns <= ewr;
                    end if;
            when ewr =>--east west red
                    --outputs of this state
                    northsouth <= "100";--red
                    eastwest <= "100";--red
                    sel<="01";  --count to 1
                    cwout<= "00";----no walking
                    if(c1="0001" and turn ='1') then
                        ns <= tg;
                    elsif(c1="0001" and (turn ='0' or cwin="10" or cwin="11"))then
                        ns<=nsg;
                    end if;
            when tg =>--turn green
                    --outputs of this state
                    turnsig<= "10"; --turnsig signal light is green
                    northsouth <= "100";--red
                    eastwest <= "100";--red
                    sel<="10";  --count to 5
                    cwout<= "00";--no walking
                    if(c5="0101")then
                        ns <= ty;
                    else
                        ns<= cs;
                    end if;
            when ty =>--turn yellow
                    --outputs of this state
                    turnsig <="01"; --turnsig signal light is yellow
                    northsouth <= "100";--red
                    eastwest <= "100";--red
                    sel<="01";  --count to 1
                    cwout<= "00"; --no walking
                    if(c1="0001") then
                        ns <= nsg;
                    else
                        ns<=cs;
                    end if;


    end case;
    end process;
end Beh;
```

Counter

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

```
entity counter is
  Port (clk,reset:in std_logic;
        input:in std_logic_vector(1 downto 0);
        crosser : in std_logic;--person needs to cross street
        c1,c5, crossOut :out std_logic_vector(3 downto 0));--count to 1 and count to 5
end counter;

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

    begin

    process(reset,clk)
    begin
    tmp1<="0000";
    tmp2<="0000";
    crossertemp<="0000";
        if(reset='1')then
            --reset counts
            tmp1<="0000";
            tmp2<="0000";
        elsif(clk'event and (clk='1'))then--on the rising edge
            if(input = "01")then
                if(tmp1 < 2) then --count to 1 (0,1)
                    tmp1 <= tmp1 + "0001";
                end if;
            elsif(input="10")then
                if(tmp2 < 5)then    --count to 5 (0,1,2,3,4)
                    tmp2 <= tmp2 + "0001";
                end if;
            end if;
            if(crosser = '1')then
                if(crossertemp<6)then
                    crossertemp<= crossertemp + "0001";
                end if;
            end if;
        end if;
        --set the values
        c1 <= tmp1;
        c5 <=tmp2;
        crossOut <= crossertemp;
    end process;
end Behavioral;
```

Seven Segment Display

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

```
entity sevSeg is
 Port (input: in std_logic_vector (3 downto 0);
       output:out std_logic_vector(6 downto 0));-- lsb
end ;

architecture Behavioral of sevSeg is

begin
process(input)
begin
case input is
when "0000" =>
    output <= "1000000"; --0
when "0001" =>
    output <= "1111001"; --1
when "0010" =>
    output <= "0100100";--2
when "0011" =>
    output <= "0110000"; --3
when "0100" =>
    output <= "0011001"; --4
when "0101" =>
    output <= "0010010";--5
when "0110" =>
    output <= "0000010";--6
when "0111" =>
    output <= "1111000";--7
when "1000" =>
    output <= "0000000";--8
when "1001" =>
    output <= "0010000";--9
when "1010" =>
     output <= "0100000";--10 A
when "1011" =>
     output <= "0000011";--11 B
when "1100" =>
     output <= "1000110";--12 C
when "1101" =>
      output <= "0100001";--13 D
when "1110" =>
      output <= "0000110";--14 E
when "1111" =>
      output <= "0001110";--15 F
end case;
end process;

end Behavioral;
```

Constraints

```
set_property IOSTANDARD LVCMOS33 [get_ports {CarEastWest[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CarEastWest[0]}]
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 {ns[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ns[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ns[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {turnout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {turnout[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clockout]
set_property IOSTANDARD LVCMOS33 [get_ports r]
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 V19 [get_ports {ns[2]}]
set_property PACKAGE_PIN U19 [get_ports {ns[1]}]
set_property PACKAGE_PIN E19 [get_ports {ns[0]}]
set_property PACKAGE_PIN V3 [get_ports clockout]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property PACKAGE_PIN V14 [get_ports {turnout[1]}]
set_property PACKAGE_PIN U14 [get_ports {turnout[0]}]
set_property PACKAGE_PIN R2 [get_ports {CarEastWest[1]}]
set_property PACKAGE_PIN U1 [get_ports r]


set_property IOSTANDARD LVCMOS33 [get_ports {cwout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cwout[0]}]
set_property PACKAGE_PIN P3 [get_ports {cwout[1]}]
set_property PACKAGE_PIN U16 [get_ports {cwout[0]}]


set_property PACKAGE_PIN T1 [get_ports {CarEastWest[0]}]


set_property IOSTANDARD LVCMOS33 [get_ports {seven[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[0]}]
set_property PACKAGE_PIN W7 [get_ports {seven[6]}]
set_property PACKAGE_PIN W6 [get_ports {seven[5]}]
set_property PACKAGE_PIN U8 [get_ports {seven[4]}]
set_property PACKAGE_PIN V8 [get_ports {seven[3]}]
set_property PACKAGE_PIN U5 [get_ports {seven[2]}]
set_property PACKAGE_PIN V5 [get_ports {seven[1]}]
set_property PACKAGE_PIN U7 [get_ports {seven[0]}]
set_property PACKAGE_PIN W2 [get_ports {walk[1]}]
set_property PACKAGE_PIN R3 [get_ports {walk[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {walk[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {walk[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports turn]


set_property PACKAGE_PIN V7 [get_ports turn]
```

Part 2 VHDL

TopLevel

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

```
entity TopLevel is
  Port (CarEastWest,walk: in std_logic_vector(1 downto 0);
        clock,r,turn:in std_logic;
        clockout: out std_logic;
        turnout, cwout: out std_logic_vector(1 downto 0);--turn signal and cross walk
        ns,ew:out std_logic_vector(2 downto 0);--lights for NortSouth and EastWest
        seven : out std_logic_vector(6 downto 0));--seven seg display
end TopLevel;

architecture Behavioral of TopLevel is
signal tmpclk,tmpcross: std_logic;
signal tmpC1,tmpC5,crossOut,countcrosstmp: std_logic_vector(3 downto 0);
signal tmpSel,tmpwalk: std_logic_vector(1 downto 0);

   --Traffic Light Component dec
   component TrafficLight port (clk,reset,turn:in std_logic;
                              c1,c5,countcross:in std_logic_vector(3 downto 0);--counters to one second and 5 seconds
                              cew: in std_logic_vector(1 downto 0);--car in the east west lane signal
                              cwin : in std_logic_vector(1 downto 0);--cross walk input (button pressed) in NS and EW
                              cwout: out std_logic_vector(1 downto 0);-- cross walk output (g or not) in NS and EW
                              csel : out std_logic;
                              sel,turnsig,crosswalk: out std_logic_vector(1 downto 0);--select bit and turn signal
                              northsouth,eastwest:out std_logic_vector(2 downto 0));

   end component;
   --Counter Component dec
   component counter port(clk,reset:in std_logic;
                       input:in std_logic_vector(1 downto 0);
                       crosser : in std_logic;
                       c1,c5,crossOut:out std_logic_vector(3 downto 0));
   end component;
   --ClockDivider Component dec
   component clockdivider port(clk : in  STD_LOGIC;
                            reset : in  STD_LOGIC;
                            SlowClock,ledO : out  STD_LOGIC);
   end component;
   --Seven Segment Display Component dec
   component sevSeg port(input: in std_logic_vector (3 downto 0);
                            output:out std_logic_vector(6 downto 0));
   end component;

begin
   --Instantiation
   CD: clockdivider port map(clk=>clock,reset=>r,SlowClock=>tmpclk,ledO=>clockout);
   TL: TrafficLight port
map(clk=>tmpclk,reset=>r,turn=>turn,c1=>tmpc1,c5=>tmpc5,countcross=>countcrosstmp,cew=>CarEastWest,cwi
n=>walk,cwout=>cwout,csel=>tmpcross,sel=>tmpSel,turnsig=>turnout,northsouth=>ns,eastwest=>ew);
```

```
    Count: counter port map (clk=>tmpclk,reset=>r,input=>tmpSel,crosser=>tmpcross,
c1=>tmpc1,c5=>tmpc5,crossout=>countcrosstmp);
    SEV: sevSeg port map (input=>countcrosstmp, output=>seven);


end Behavioral;
```

Clock Divider

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

```
entity Clockdivider is
Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          SlowClock,ledO : out  STD_LOGIC);
end Clockdivider;

architecture behavioral of Clockdivider is
signal slowSig: std_logic;
begin
     process
      variable cnt :    std_logic_vector(26 downto 0):= "000000000000000000000000000";
          begin                            -- calculations
            wait until ((clk'EVENT) AND (clk = '1'));
            if (reset = '1') then
                  cnt := "000000000000000000000000000";
             else
               cnt := cnt + 1; -- count to 26
             end if;

        SlowClock <= cnt(26);
        slowSig<=cnt(26);

      if(slowSig='1')then
        ledO<='1';
      else
        ledO<='0';
      end if;

   end process;
end Behavioral;
```

Traffic Light

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

```
entity TrafficLight is
  Port (clk,reset,turn:in std_logic;
        c1,c5,countcross:in std_logic_vector(3 downto 0);--counters to one second and 5 seconds and for the cross
walk
        cew: in std_logic_vector(1 downto 0);--car in the east west lane signal
        cwin : in std_logic_vector(1 downto 0);--cross walk input (button pressed) in NS and EW
        cwout: out std_logic_vector(1 downto 0);-- cross walk output (g or not) in NS and EW
        csel : out std_logic;
        sel,turnsig,crosswalk : out std_logic_vector(1 downto 0);--select bit and turn signal
        northsouth,eastwest:out std_logic_vector(2 downto 0));

end TrafficLight;

architecture Beh of TrafficLight is
    type state_type is (nsg,nsy,nsr,ewr,ewg,ewy,tg,ty);
    signal cs,ns: state_type;
    signal walk : std_logic_vector(1 downto 0);

    begin
    process(clk,reset)
    begin
        if (reset='1')then
            cs<=nsg;
        elsif(clk'event and clk='1')then
           cs<=ns;
        end if;
    end process;

    process(cs,cew)
    begin
        turnsig <="00";
        crosswalk<= "00";
        csel <= '0';
        case cs is
            when nsg =>--north south green
                        --outputs of this state
                        northsouth <= "001";--green
                        eastwest <= "100";--red

                        sel<="10"; --count to 5
                        if(cwin = "01" or cwin = "11") then
                         cSel<= '1';
                        end if;
                        if(c5="0101" and (cew="01" or cew="10" or cew="11" or cwin = "00" or cwin ="01" or cwin="10" or
cwin="11")) then-- no crosswalk input or east west cross input or ignore
                             ns<=nsy;
                        else
```

```
                    ns<=cs;
                end if;

        when nsy =>--north south yellow
                --outputs of this state
                northsouth <= "010";--yellow
                eastwest <= "100";--red
                sel<="01";  --count to 1
                --cwout<= "01";--walk in NS
                walk<= "10";
                if(c1="0001") then
                    ns <= nsr;
                end if;
        when nsr =>--north south red
                --outputs of this state
                northsouth <= "100";--red
                eastwest <= "100";--red
                sel<="01";  --count to 1
                --cwout<= "00";-- no walkling
                if (c1 = "0001" and (cew="01" or cew ="10" or cew="11" or cwin = "00" or cwin ="01" or cwin="10"
or cwin="11" ))then
                    ns <= ewg;
                elsif(c1="0001" and turn = '1' and cew="00") then
                    ns<= tg;
                end if;

        when ewg =>--east west green
                 --outputs of this state
                 northsouth <= "100";--red
                 eastwest <= "001";--green
                 sel<="10";  --count to 5
                 walk<="01";
                 --cwout<= "01"; --walk in EW
                 if(walk<="01" and (cwin = "01" or cwin = "11")) then -- no cross input or north south cross input or
ignore
                       csel<='1';
                       crosswalk<="01";
                    if(countcross="0110" )then
                       crosswalk<= "01";
                    end if;
                 end if;

                 if(c5="0101" and (cew="00" or cwin="01" or cwin="11")) then
                    ns<=ewy;
                  else
                    ns<=cs;
                  end if;
        when ewy =>--easy west yellow
```

```
                --outputs of this state
                northsouth <= "100";--red
                eastwest <= "010";--yellow
                sel<="01";  --count to 1
                cwout<= "10"; --walk in ew
                if(c1="0001") then
                    ns <= ewr;
                end if;
        when ewr =>--east west red
                --outputs of this state
                northsouth <= "100";--red
                eastwest <= "100";--red
                sel<="01";  --count to 1
                cwout<= "00";----no walking
                if(c1="0001" and turn ='1') then
                    ns <= tg;
                elsif(c1="0001" and (turn ='0' or cwin="10" or cwin="11"))then
                    ns<=nsg;
                end if;
        when tg =>--turn green
                --outputs of this state
                turnsig<= "10"; --turnsig signal light is green
                northsouth <= "100";--red
                eastwest <= "100";--red
                sel<="10";  --count to 5
                cwout<= "00";--no walking
                if(c5="0101")then
                    ns <= ty;
                else
                    ns<= cs;
                end if;
        when ty =>--turn yellow
                --outputs of this state
                turnsig <="01"; --turnsig signal light is yellow
                northsouth <= "100";--red
                eastwest <= "100";--red
                sel<="01";  --count to 1
                cwout<= "00"; --no walking
                if(c1="0001") then
                    ns <= nsg;
                else
                    ns<=cs;
                end if;


   end case;
   end process;
end Beh;
```

Counter

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

```vhdl
entity counter is
 Port (clk,reset:in std_logic;
     input:in std_logic_vector(1 downto 0);
     crosser : in std_logic;--person needs to cross street
     c1,c5, crossOut :out std_logic_vector(3 downto 0));--count to 1 and count to 5
end counter;

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

  begin

  process(reset,clk)
  begin
  tmp1<="0000";
  tmp2<="0000";
  crossertemp<="0000";
     if(reset='1')then
       --reset counts
       tmp1<="0000";
       tmp2<="0000";
     elsif(clk'event and (clk='1'))then--on the rising edge
       if(input = "01")then
         if(tmp1 < 2) then --count to 1 (0,1)
           tmp1 <= tmp1 + "0001";
         end if;
       elsif(input="10")then
         if(tmp2 < 5)then    --count to 5 (0,1,2,3,4)
           tmp2 <= tmp2 + "0001";
         end if;
       end if;
       if(crosser = '1')then
         if(crossertemp<6)then
           crossertemp<= crossertemp + "0001";
         end if;
       end if;
     end if;
     --set the values
     c1 <= tmp1;
     c5 <=tmp2;
     crossOut <= crossertemp;
  end process;
end Behavioral;
```

Seven Segment Display

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

```
entity sevSeg is
 Port (input: in std_logic_vector (3 downto 0);
       output:out std_logic_vector(6 downto 0));-- lsb
end ;

architecture Behavioral of sevSeg is

begin
process(input)
begin
case input is
when "0000" =>
    output <= "1000000"; --0
when "0001" =>
    output <= "1111001"; --1
when "0010" =>
    output <= "0100100";--2
when "0011" =>
    output <= "0110000"; --3
when "0100" =>
    output <= "0011001"; --4
when "0101" =>
    output <= "0010010";--5
when "0110" =>
    output <= "0000010";--6
when "0111" =>
    output <= "1111000";--7
when "1000" =>
    output <= "0000000";--8
when "1001" =>
    output <= "0010000";--9
when "1010" =>
     output <= "0100000";--10 A
when "1011" =>
     output <= "0000011";--11 B
when "1100" =>
     output <= "1000110";--12 C
when "1101" =>
      output <= "0100001";--13 D
when "1110" =>
      output <= "0000110";--14 E
when "1111" =>
      output <= "0001110";--15 F
end case;
end process;

end Behavioral;
```

Constraints

```
set_property IOSTANDARD LVCMOS33 [get_ports {CarEastWest[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CarEastWest[0]}]
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 {ns[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ns[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ns[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {turnout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {turnout[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clockout]
set_property IOSTANDARD LVCMOS33 [get_ports r]
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 V19 [get_ports {ns[2]}]
set_property PACKAGE_PIN U19 [get_ports {ns[1]}]
set_property PACKAGE_PIN E19 [get_ports {ns[0]}]
set_property PACKAGE_PIN V3 [get_ports clockout]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property PACKAGE_PIN V14 [get_ports {turnout[1]}]
set_property PACKAGE_PIN U14 [get_ports {turnout[0]}]
set_property PACKAGE_PIN R2 [get_ports {CarEastWest[1]}]
set_property PACKAGE_PIN U1 [get_ports r]


set_property IOSTANDARD LVCMOS33 [get_ports {cwout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cwout[0]}]
set_property PACKAGE_PIN P3 [get_ports {cwout[1]}]
set_property PACKAGE_PIN U16 [get_ports {cwout[0]}]


set_property PACKAGE_PIN T1 [get_ports {CarEastWest[0]}]


set_property IOSTANDARD LVCMOS33 [get_ports {seven[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven[0]}]
set_property PACKAGE_PIN W7 [get_ports {seven[6]}]
set_property PACKAGE_PIN W6 [get_ports {seven[5]}]
set_property PACKAGE_PIN U8 [get_ports {seven[4]}]
set_property PACKAGE_PIN V8 [get_ports {seven[3]}]
set_property PACKAGE_PIN U5 [get_ports {seven[2]}]
set_property PACKAGE_PIN V5 [get_ports {seven[1]}]
set_property PACKAGE_PIN U7 [get_ports {seven[0]}]
set_property PACKAGE_PIN W2 [get_ports {walk[1]}]
set_property PACKAGE_PIN R3 [get_ports {walk[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {walk[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {walk[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports turn]


set_property PACKAGE_PIN V7 [get_ports turn]
```