David Castro

Last 4 Digits of CWID: 2227

EGEE-406

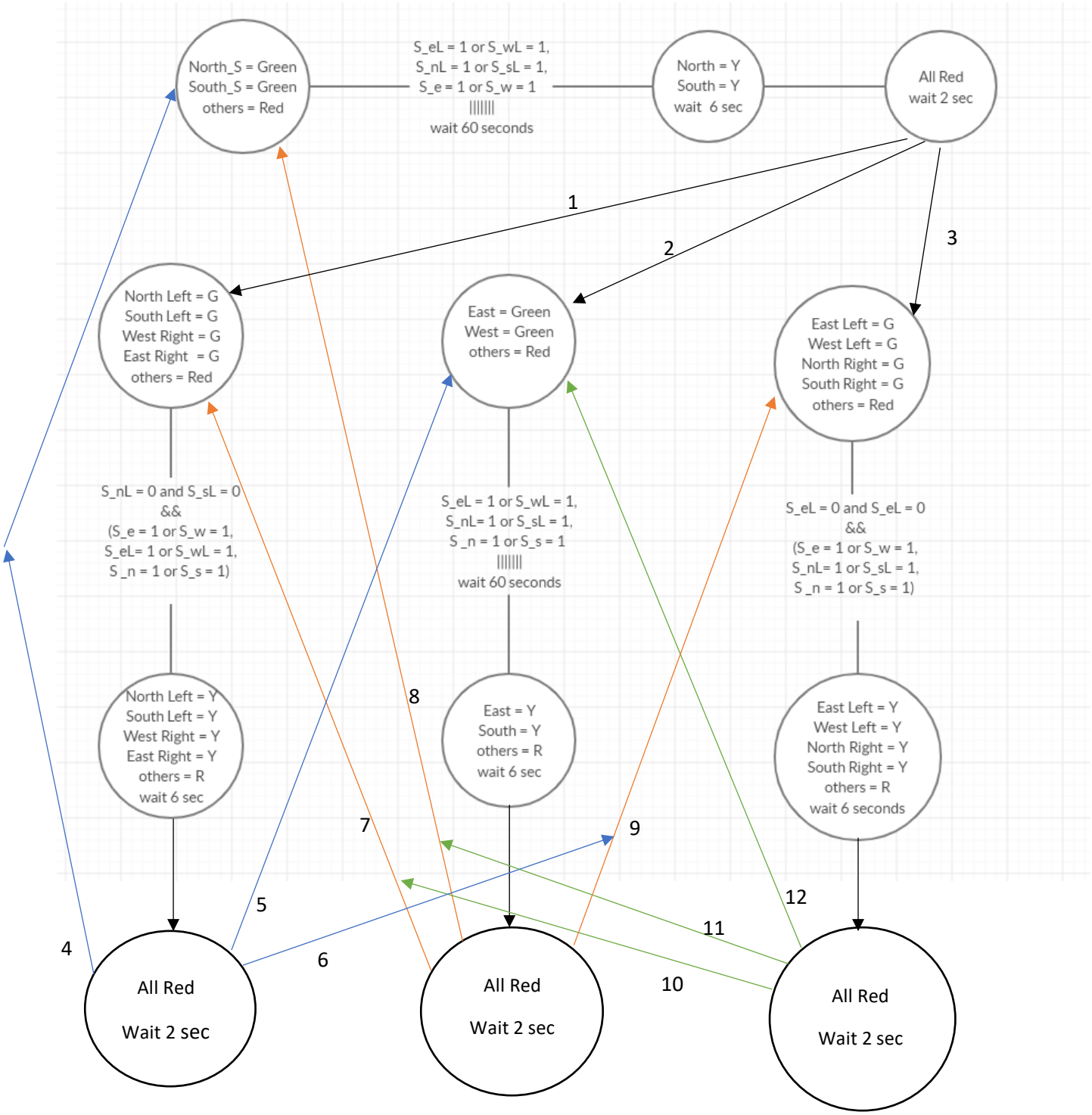
Project 4

Report

1.
   a. The FPGA controls the streetlights in a fairly simple way. Its able to complete everything required in the design and can utilize a clock to act as counter in seconds which allows it to be accurate in its timings. The design also has its conditions loop in a consistent circle when checking for what which state to get next. For example, this means that if there are still cars after the green light for them has turn red, they would have to wait a full cycle for the rest of the lights assuming there are cars in each of the other lanes. This means that it doesn't put give any of the sensors/lanes priority. Public safety wise everything functions according to the design, and there are no problems in controlling the light system. There are no U-turns allowed so that will have to accounted for. Cost effectiveness I'm not sure price wise the amount of silicon that is needed to implement this program in comparison to a microcontroller, but the program appears to be simple in its design so I don't believe much would be needed so probably on a mass scale it could be cheaper. Computing this program functions much quicker in comparison to a MCU because of how an FPGA being able to process things simultaneously, but for something as this I'm not sure if this speed necessary.
   b. The traffic light sequence table shows all the states in the design. You see 12 rows so there will 12 states that need to be implemented. From this you just need to create the logic that shows how each state would go from one to another. The way this is designed though makes it so that there can be no U-turns ever. An external sign on the light system that would show U-turns as illegal. This system doesn't account for pedestrians so there would have to be also signs that prohibit pedestrians from crossing the street.

** the notations on the following page are:
-- n = north, s = south, w = west, e = east
-- L = left, R = Right,
-- R = red, G = Green, y = yellow
-- ex. R_nL = Red light that is for north side intersection going left
-- S = sensor , ex = S_s = South sensor

c. State Transition Diagram



1. S_nL = 1 or S_sL = 1
2. S_w = 1 or S_e = 1
3. S_eL = 1 or S_nR = 1
4. S_n = 1 or S_s =1
5. S_w = 1 or S_e = 1
6. S_eL = 1 or S_nR = 1

7. S_nL = 1 or S_sL = 1
8. S_n = 1 or S_s =1
9. S_eL = 1 or S_nR = 1
10. S_nL = 1 or S_sL = 1
11. S_n = 1 or S_s =1
12. S_w = 1 or S_e = 1

2. VHDL Code:

**Something to Note of is that right now this code is made to function off a 1 Hz clk from the Test Bench. The slow clock is coded in right now but is not being used. The reason for this is if I was simulating a 100 Mhz clk in the test bench then it would take an hours to simulate it when in comparison to a 1 Hz clk. I've highlighted where the change that would be necessary in order if you wanted it to run off the slow clock. You would simply replace "slowClk" with "clk" although I'm not sure it would work as I have no way of testing it since the simulation with it doesn't really work. This would need to be tested on the board since it has a 100 Mhz clock already. The same operation occurs though where everything is in seconds either way.

```
--Right now triggered off the TB clk instead of the slow clk
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- n = north,  s = south,  w = west,  e = east
-- L = left,   R = Right,
-- R = red,    G = Green,   y = yellow
-- ex. R_NL = Red light that is for north side intersection going left
-- S = sensor , ex = S_s = South sensor
entity TLC is
Port (clk, S_n, S_s, S_w, S_e, S_nL, S_sL, S_wL, S_eL: in std_logic; -- sensors
R_N, R_NR, R_NL, R_E, R_ER, R_EL, R_S, R_SR, R_SL, R_W, R_WR, R_WL: inout std_logic;
Y_N, Y_NR, Y_NL, Y_E, Y_ER, Y_EL, Y_S, Y_SR, Y_SL, Y_W, Y_WR, Y_WL: inout std_logic;
G_N, G_NR, G_NL, G_E, G_ER, g_EL, G_S, G_SR, G_SL, G_W, G_WR, G_WL: inout std_logic;
timer: inout integer := 0);
end TLC;

architecture Behavioral of TLC is

signal state, nextstate : integer range 0 to 11;
```

```
type light is (R, Y, G);
signal lightN, lightNR, lightNL: light;
signal lightS, lightSR, lightSL: light;
signal lightE, lightER, lightEL: light;
signal lightW, lightWR, lightWL: light;
signal clkCnt: integer := 0;
signal slowClk: std_logic;
signal rst_timer: std_logic := '0';



begin

--clk scaler with 100 MHz clk
  Prescaler: Process(clk)
   begin
    if rising_edge(clk) then
        if clkCnt = 100000000 - 1 then -- if counts up to 100 million
            clkCnt <= 0;    -- when one second passes reset counter
        else
            clkCnt <= clkCnt + 1; --incrementing counter to simulate seconds
        end if;
    end if;
  end process Prescaler;

process(state, S_n, S_s, S_w, S_e, S_nL, S_sL, S_wL, S_eL)
  begin
        --initial setup of all the lights
        R_N <= '0';
        R_NR <= '0';
        R_NL <= '0';
        R_S <= '0';
```

```
R_SR <= '0';
R_SL <= '0';
R_E <= '0';
R_ER <= '0';
R_EL <= '0';
R_W <= '0';
R_WR <= '0';
R_WL <= '0';

Y_N <= '0';
Y_NR <= '0';
Y_NL <= '0';
Y_S <= '0';
Y_SR <= '0';
Y_SL <= '0';
Y_E <= '0';
Y_ER <= '0';
Y_EL <= '0';
Y_W <= '0';
Y_WR <= '0';
Y_WL <= '0';

G_N <= '0';
G_NR <= '0';
G_NL <= '0';
G_S <= '0';
G_SR <= '0';
G_SL <= '0';
G_E <= '0';
G_ER <= '0';
G_EL <= '0';
G_W <= '0';
```

```
    G_WR <= '0';
    G_WL <= '0';

case state is
    when 0 =>   --when north and south straights are green
        R_NR <= '1';
        R_NL <= '1';
        R_SR <= '1';
        R_SL <= '1';
        R_E <= '1';
        R_ER <= '1';
        R_EL <= '1';
        R_W <= '1';
        R_WR <= '1';
        R_WL <= '1';

        G_N <= '1';
        G_S <= '1';

        --checking the sensors
        if S_e = '1' or S_w = '1' or S_nL = '1' or S_sL = '1' or S_wL = '1' or S_eL = '1' then
            nextstate <= state + 1;
        end if;

    when 1 =>       -- when transition from Green to Yellow for north and south
        R_NR <= '1';
        R_NL <= '1';
        R_SR <= '1';
        R_SL <= '1';
        R_E <= '1';
        R_ER <= '1';
        R_EL <= '1';
```

```
    R_W <= '1';
    R_WR <= '1';
    R_WL <= '1';

    Y_N <= '1';
    Y_S <= '1';

    nextstate <= 2; --automatically go to next state after timer

when 2 =>  --red lights (dead-time)
    R_N <= '1';
    R_NR <= '1';
    R_NL <= '1';
    R_S <= '1';
    R_SR <= '1';
    R_SL <= '1';
    R_E <= '1';
    R_ER <= '1';
    R_EL <= '1';
    R_W <= '1';
    R_WR <= '1';
    R_WL <= '1';

    --checking sensors
    if S_e = '1' or S_w = '1' then
        nextstate <= 6; -- go to west and east green straights

    elsif S_eL = '1' or S_wL = '1' then
        nextstate <= 3; -- go to east and west left

    elsif S_nL = '1' or S_sL = '1' then
        nextstate <= 9; -- go to north and south left
```

```
    end if;

when 3 => -- the East and West green left lights
    R_N <= '1';
    R_NL <= '1';
    R_S <= '1';
    R_SL <= '1';
    R_E <= '1';
    R_ER <= '1';
    R_W <= '1';
    R_WR <= '1';

    G_NR <= '1';
    G_SR <= '1';
    G_EL <= '1';
    G_WL <= '1';

    --checking the sensors
    if S_e = '1' or S_w = '1' or S_nL = '1' or S_sL = '1' or S_s = '1' or S_s = '1'  then
        nextstate <= 4;
    end if;

when 4  => -- the East and West yellow left lights
    R_N <= '1';
    R_NL <= '1';
    R_S <= '1';
    R_SL <= '1';
    R_E <= '1';
    R_ER <= '1';
    R_W <= '1';
    R_WR <= '1';
```

```
    Y_NR <= '1';
    Y_SR <= '1';
    Y_EL <= '1';
    Y_WL <= '1';

    nextstate <= 5; --atuotmatically go to next lights after timer

when 5 => -- all read lights
    R_N <= '1';
    R_NR <= '1';
    R_NL <= '1';
    R_S <= '1';
    R_SR <= '1';
    R_SL <= '1';
    R_E <= '1';
    R_ER <= '1';
    R_EL <= '1';
    R_W <= '1';
    R_WR <= '1';
    R_WL <= '1';

    if S_e = '1' or S_w = '1' then
        nextstate <= 6; -- go to west and east green straights

    elsif S_n = '1' or S_s = '1' then
        nextstate <= 0; -- go to north and sourth green straights

    elsif S_nL = '1' or S_sL = '1' then
        nextstate <= 9; -- go to north and south left
    end if;
```

```
when 6 =>       -- green and west green straights
   R_N <= '1';
   R_NR <= '1';
   R_NL <= '1';
   R_S <= '1';
   R_SR <= '1';
   R_SL <= '1';
   R_ER <= '1';
   R_EL <= '1';
   R_WR <= '1';
   R_WL <= '1';

   G_W <= '1';
   G_E <= '1';

  --checking sensors
  if S_s = '1' or S_n = '1' or S_nL = '1' or S_sL = '1' or S_wL = '1' or S_eL = '1'  then
       nextstate <= 7;
   end if;

when 7 =>         -- east and west yellow straights
   R_N <= '1';
   R_NR <= '1';
   R_NL <= '1';
   R_S <= '1';
   R_SR <= '1';
   R_SL <= '1';
   R_ER <= '1';
   R_EL <= '1';
   R_WR <= '1';
   R_WL <= '1';
```

```
    Y_W <= '1';
    Y_E <= '1';

    nextstate <= 8;

when 8 => -- all red lights
    R_N <= '1';
    R_NR <= '1';
    R_NL <= '1';
    R_S <= '1';
    R_SR <= '1';
    R_SL <= '1';
    R_E <= '1';
    R_ER <= '1';
    R_EL <= '1';
    R_W <= '1';
    R_WR <= '1';
    R_WL <= '1';

    if S_nL = '1' or S_sL = '1' then
        nextstate <= 9; -- go to north and south left

    elsif S_n = '1' or S_s = '1' then
        nextstate <= 0; -- go to north and sourth green straights

    elsif S_wL = '1' or S_eL = '1' then
        nextstate <= 3;
    end if;

when 9 =>   -- green for north and south left
    R_N <= '1';
    R_NR <= '1';
```

```
    R_S <= '1';
    R_SR <= '1';
    R_E <= '1';
    R_EL <= '1';
    R_W <= '1';
    R_WL <= '1';

    G_NL <= '1';
    G_SL <= '1';
    G_ER <= '1';
    G_WR <= '1';

    --checking sensors for any cars
    if S_s = '1' or S_n = '1' or S_w = '1' or S_e = '1' or S_wL = '1' or S_eL = '1'  then
        nextstate <= 10;
    end if;

when 10 =>  --yellow for north and south left
    R_N <= '1';
    R_NR <= '1';
    R_S <= '1';
    R_SR <= '1';
    R_E <= '1';
    R_EL <= '1';
    R_W <= '1';
    R_WL <= '1';

    Y_NL <= '1';
    Y_SL <= '1';
    Y_ER <= '1';
    Y_WR <= '1';
```

```
        --automatically go to next state after timer
        nextstate <= 11;

      when 11 =>  -- all red lights
        R_N <= '1';
        R_NR <= '1';
        R_NL <= '1';
        R_S <= '1';
        R_SR <= '1';
        R_SL <= '1';
        R_E <= '1';
        R_ER <= '1';
        R_EL <= '1';
        R_W <= '1';
        R_WR <= '1';
        R_WL <= '1';

        if S_n = '1' or S_s = '1' then
          nextstate <= 0; -- go to north and sourth green straights

        elsif S_wL = '1' or S_eL = '1' then
          nextstate <= 3;     -- go to east and west green left

        elsif S_w = '1' or S_e = '1' then
          nextstate <= 6; -- go to west and east green
        end if;
    end case;
  end process;

process(timer) -- this transitions to the next state and is triggered of a timer in seconds
  begin
```

```
--reseting the timer reset
if rst_timer = '1' then rst_timer <= '0'; end if;

if state = 0 then
  if timer = 60 then    -- when 60 seconds
    state <= nextstate;
    rst_timer <= '1';
  end if;
end if;

if state = 1 then
    if timer = 6 then   -- when 6 seconds
      state <= nextstate;
      rst_timer <= '1';
    end if;
end if;

if state = 2 then
    if timer = 2 then   -- when 2 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 3 then
    if S_wL = '0' and S_eL = '0' then -- when no more cars
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 4 then
```

```
    if timer = 6 then -- when 6 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 5 then
    if timer = 2 then --when 2 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 6 then
    if timer = 60 then  -- when 60 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 7 then
    if timer = 6 then       -- when 6 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
end if;

if state = 8 then
    if timer = 2 then       -- when 2 seconds
        state <= nextstate;
        rst_timer <= '1';
    end if;
```

```
    end if;

    if state = 9 then
        if S_nL = '0' and S_sL = '0' then   -- when no more cars
            state <= nextstate;
            rst_timer <= '1';
        end if;
    end if;

    if state = 10 then
        if timer = 6 then   -- when 6 seconds
            state <= nextstate;
            rst_timer <= '1';
        end if;
    end if;

    if state = 11 then
        if timer = 2 then -- when 2 seconds
            state <= nextstate;
            rst_timer <= '1';
        end if;
    end if;
end process;

-- right now functions off a 1 Hz clk in the testbench and not the slowclk
second_timer: Process(clk,rst_timer)      -- this acts as a second counter
  begin
        if(rising_edge(clk)) then       --increments until reset timer is triggered
            timer <= timer + 1;
        end if;
        if(rst_timer = '1') then
            timer <= 0;
```

```
    end if;
  end process;

--this would be the slowclk that converts 100Mhz to 1Hz
slowClk <= '1' when clkCnt = 100000000 - 1 else '0';


--This is an easier way of displaying the lights in the simulation/testbench
lightN <= R when R_N = '1' else Y when Y_N = '1' else G when G_N = '1';
lightS <= R when R_S = '1' else Y when Y_S = '1' else G when G_S = '1';
lightE <= R when R_E = '1' else Y when Y_E = '1' else G when G_E = '1';
lightW <= R when R_W = '1' else Y when Y_W = '1' else G when G_W = '1';
lightNR <= R when R_NR = '1' else Y when Y_NR = '1' else G when G_NR = '1';
lightNL <= R when R_NL = '1' else Y when Y_NL = '1' else G when G_NL = '1';
lightSR <= R when R_SR = '1' else Y when Y_SR = '1' else G when G_SR = '1';
lightSL <= R when R_SL = '1' else Y when Y_SL = '1' else G when G_SL = '1';
lightER <= R when R_ER = '1' else Y when Y_ER = '1' else G when G_ER = '1';
lightEL <= R when R_EL = '1' else Y when Y_EL = '1' else G when G_EL = '1';
lightWR <= R when R_WR = '1' else Y when Y_WR = '1' else G when G_WR = '1';
lightWL <= R when R_WL = '1' else Y when Y_WL = '1' else G when G_WL = '1';

end Behavioral;
```

3. **Force Commands: -** (is in ms because seconds have the 10x bug for force commands)
   add_force clk {0 0} {1 500us} -repeat_every 1ms
   add_force S_s {0 0ms}
   add_force S_w {0 0ms} {1 80ms}
   add_force S_e {0 0ms}
   add_force S_nL {0 0ms}
   add_force S_sL {0 0ms} {1 105ms}
   add_force S_wL {0 0ms} {1 10ms} {0 90ms}
   add_force S_eL {0 0ms}
   add_force S_n {1 0ms}

4. VHDL Test Bench Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TLC_TB is
--  Port ( );
end TLC_TB;

architecture Behavioral of TLC_TB is

component TLC --I/O from main code
    Port (clk, S_n, S_s, S_w, S_e, S_nL, S_sL, S_wL, S_eL: in std_logic; -- sensors
R_N, R_NR, R_NL, R_E, R_ER, R_EL, R_S, R_SR, R_SL, R_W, R_WR, R_WL: inout std_logic;
Y_N, Y_NR, Y_NL, Y_E, Y_ER, Y_EL, Y_S, Y_SR, Y_SL, Y_W, Y_WR, Y_WL: inout std_logic;
G_N, G_NR, G_NL, G_E, G_ER, g_EL, G_S, G_SR, G_SL, G_W, G_WR, G_WL: inout std_logic;
timer: inout integer := 0);
end component;

--inputs
signal wire_clk: std_logic;
signal S_n: std_logic;
signal S_s: std_logic;
```

```
signal S_e: std_logic;
signal S_w: std_logic;
signal S_nL: std_logic;
signal S_sL: std_logic;
signal S_wL: std_logic;
signal S_eL: std_logic;

--signal/outputs
signal R_N, R_NR, R_NL, R_E, R_ER, R_EL: std_logic;
signal R_S, R_SR, R_SL, R_W, R_WR, R_WL: std_logic;
signal Y_N, Y_NR, Y_NL, Y_E, Y_ER, Y_EL, Y_S: std_logic;
signal Y_SR, Y_SL, Y_W, Y_WR, Y_WL: std_logic;
signal G_N, G_NR, G_NL, G_E, G_ER, G_EL, G_S: std_logic;
signal G_SR, G_SL, G_W, G_WR, G_WL: std_logic;

--main output
type light is (R, Y, G);
signal lightN, lightNR, lightNL: light;
signal lightS, lightSR, lightSL: light;
signal lightE, lightER, lightEL: light;
signal lightW, lightWR, lightWL: light;
signal timer: integer;



begin

uut: TLC Port Map( --connecting the test bench to the main code
    clk => wire_clk,
    S_n => S_n,
    S_s => S_s,
    S_e => S_e,
```

S_w => S_w,

S_nL => S_nL,

S_sL => S_sL,

S_wL => S_wL,

S_eL => S_eL,

R_N => R_N,

R_NR => R_NR,

R_NL => R_NL,

R_E => R_E,

R_ER => R_ER,

R_EL => R_EL,

R_S => R_S,

R_SR => R_SR,

R_SL => R_SL,

R_W => R_W,

R_WR => R_WR,

R_WL => R_WL,

Y_N => Y_N,

Y_NR => Y_NR,

Y_NL => Y_NL,

Y_E => Y_E,

Y_ER => Y_ER,

Y_EL => Y_EL,

Y_S => Y_S,

Y_SR => Y_SR,

Y_SL => Y_SL,

Y_W => Y_W,

Y_WR => Y_WR,

Y_WL => Y_WL,

G_N => G_N,

G_NR => G_NR,

G_NL => G_NL,

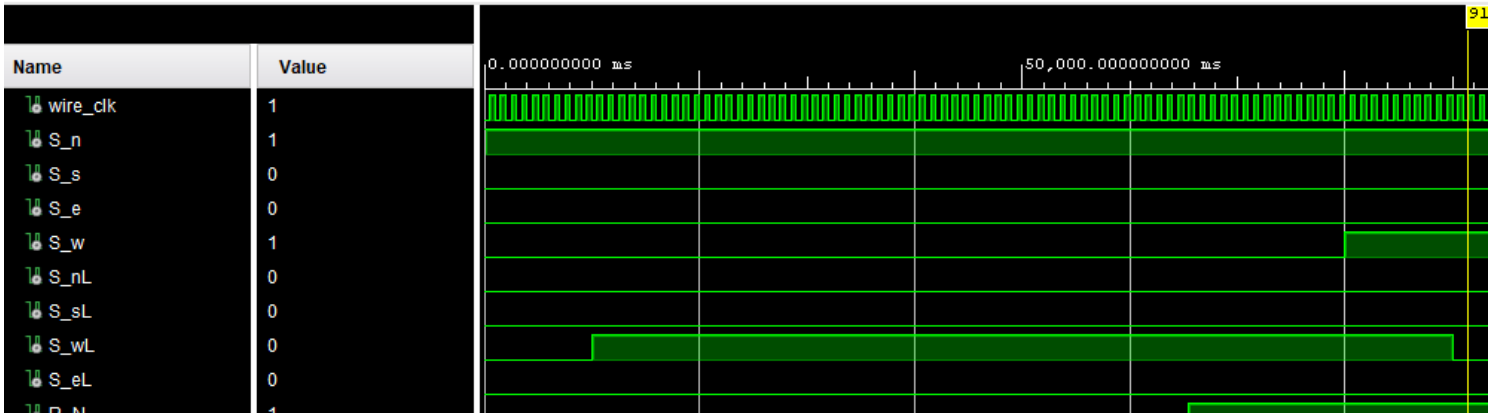
G_E => G_E,

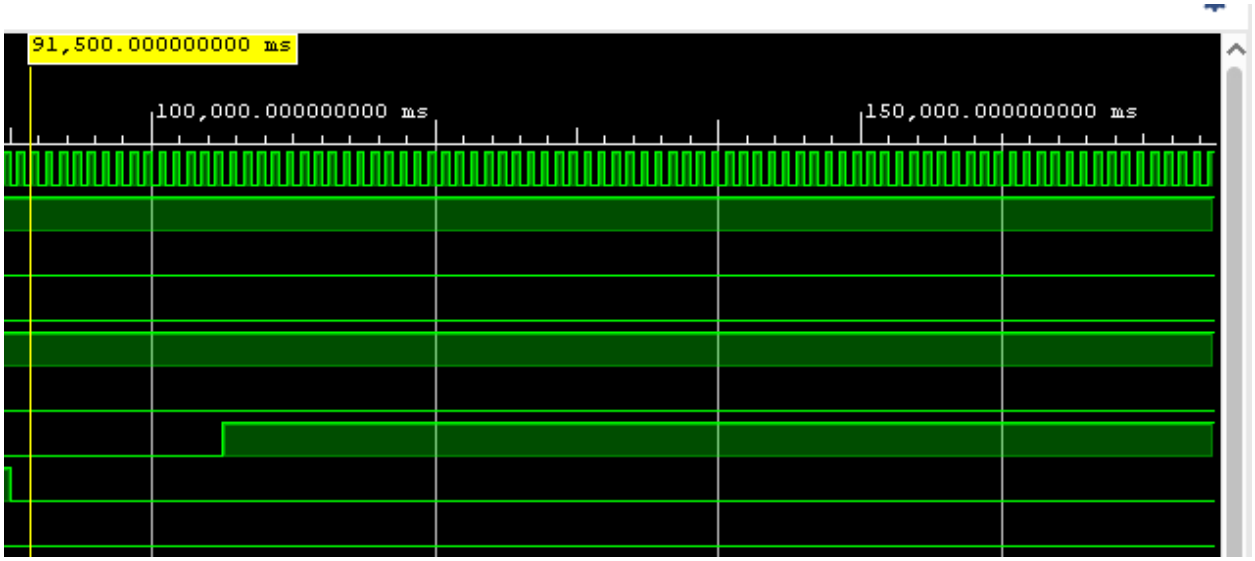
```
G_ER => G_ER,
G_EL => G_EL,
G_S => G_S,
G_SR => G_SR,
G_SL => G_SL,
G_W => G_W,
G_WR => G_WR,
G_WL => G_WL,
timer => timer
);

clk: process    -- 1 Hz clock
 begin
   wire_clk <= '0';
   wait for 0.5 sec;
   wire_clk <= '1';
   wait for 0.5 sec;
 end process;

stim_proc: process    --simulating traffic
  begin
   S_s <= '0';
   S_w <= '0';
   S_e <= '0';
   S_nL <= '0';
   S_sL <= '0';
   S_wL <= '0';
   S_eL <= '0';
   S_n <= '1';
   wait for 10 sec;
   S_wL <= '1';
   wait for 70 sec;
   S_w <= '1';
```

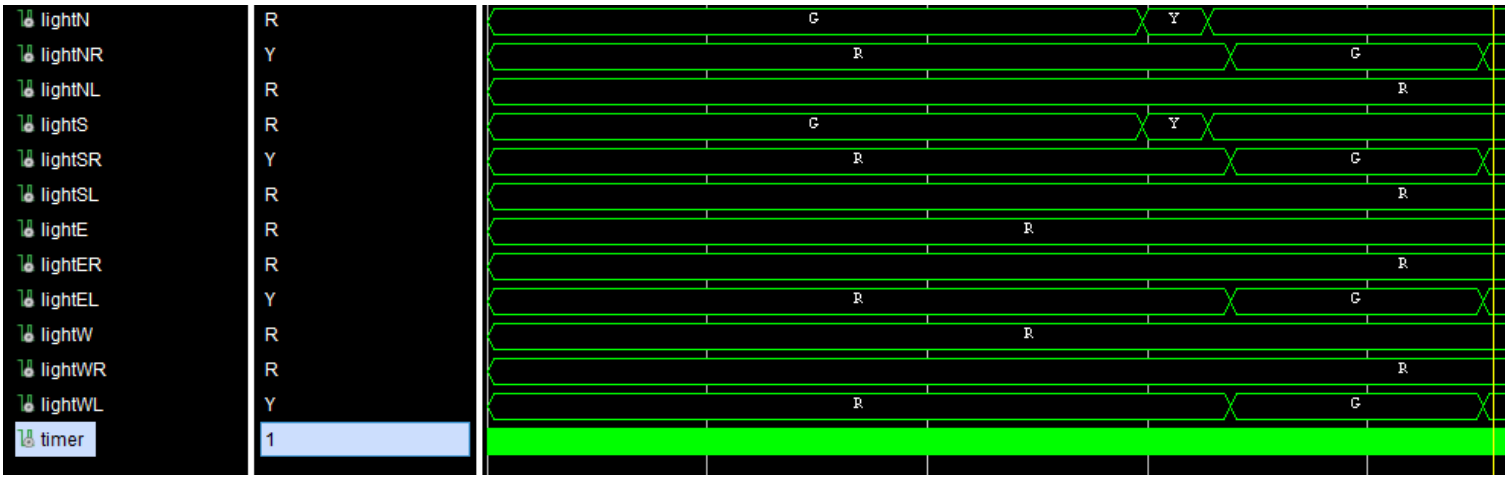
```
    wait for 10 sec;
    S_wL <= '0';
    wait for 15 sec;
    S_sL <= '1';
    wait for 70 sec;
    assert false report "End of Simulation" severity failure;


  end process;

--Main outputs for ease of reading in simulation
lightN <= R when R_N = '1' else Y when Y_N = '1' else G when G_N = '1';
lightS <= R when R_S = '1' else Y when Y_S = '1' else G when G_S = '1';
lightE <= R when R_E = '1' else Y when Y_E = '1' else G when G_E = '1';
lightW <= R when R_W = '1' else Y when Y_W = '1' else G when G_W = '1';
lightNR <= R when R_NR = '1' else Y when Y_NR = '1' else G when G_NR = '1';
lightNL <= R when R_NL = '1' else Y when Y_NL = '1' else G when G_NL = '1';
lightSR <= R when R_SR = '1' else Y when Y_SR = '1' else G when G_SR = '1';
lightSL <= R when R_SL = '1' else Y when Y_SL = '1' else G when G_SL = '1';
lightER <= R when R_ER = '1' else Y when Y_ER = '1' else G when G_ER = '1';
lightEL <= R when R_EL = '1' else Y when Y_EL = '1' else G when G_EL = '1';
lightWR <= R when R_WR = '1' else Y when Y_WR = '1' else G when G_WR = '1';
```

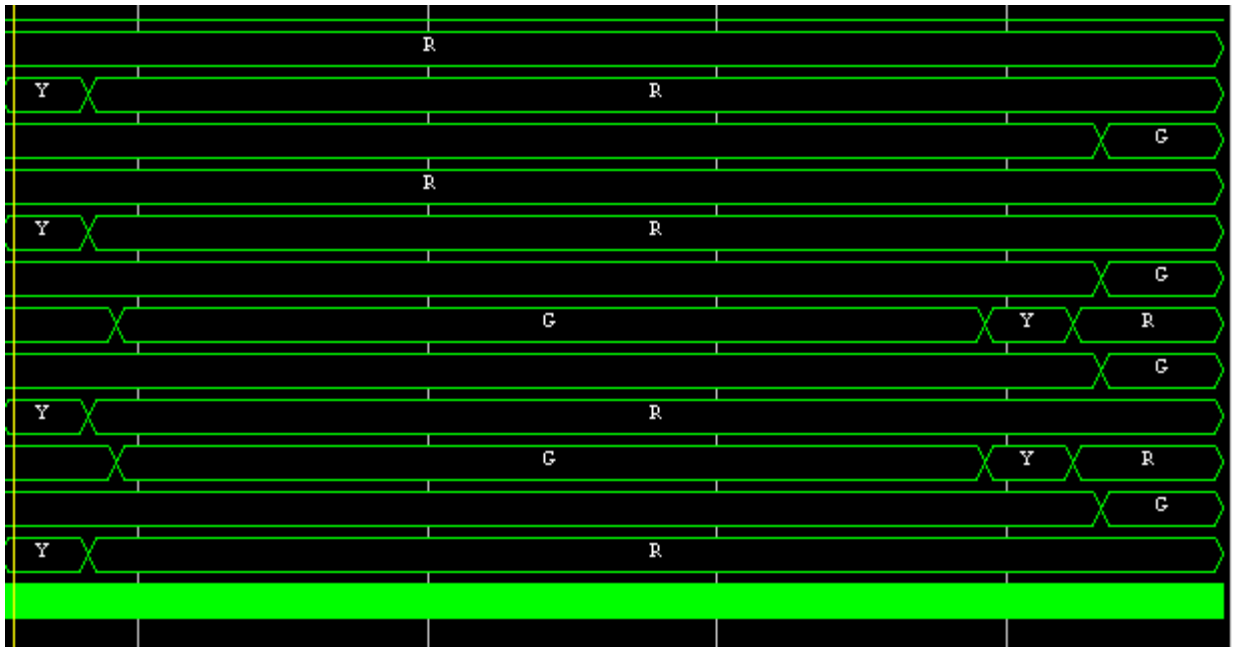
5. Simulation of Test Bench

*shows each of the lights operating

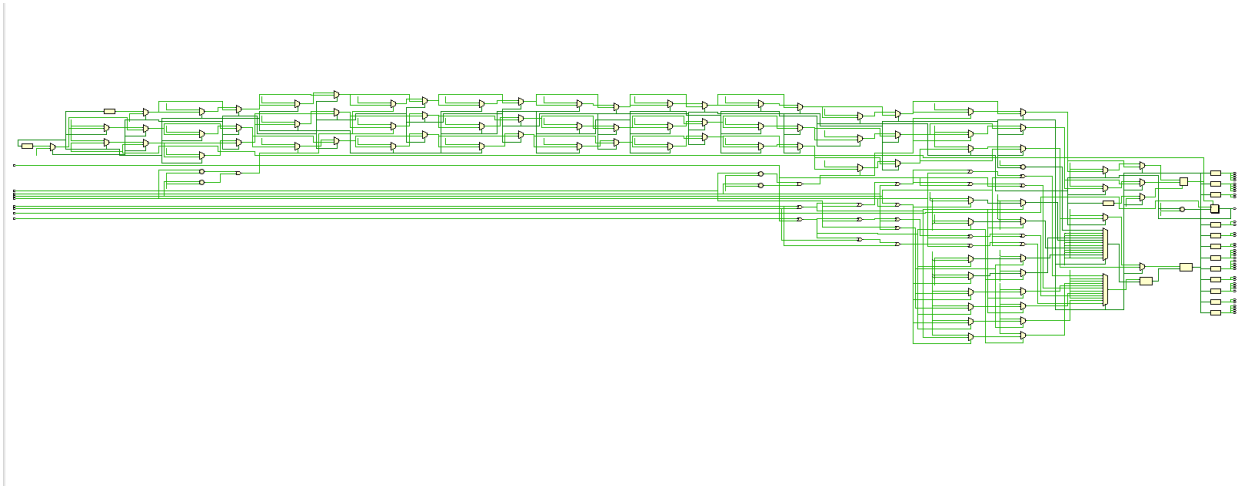




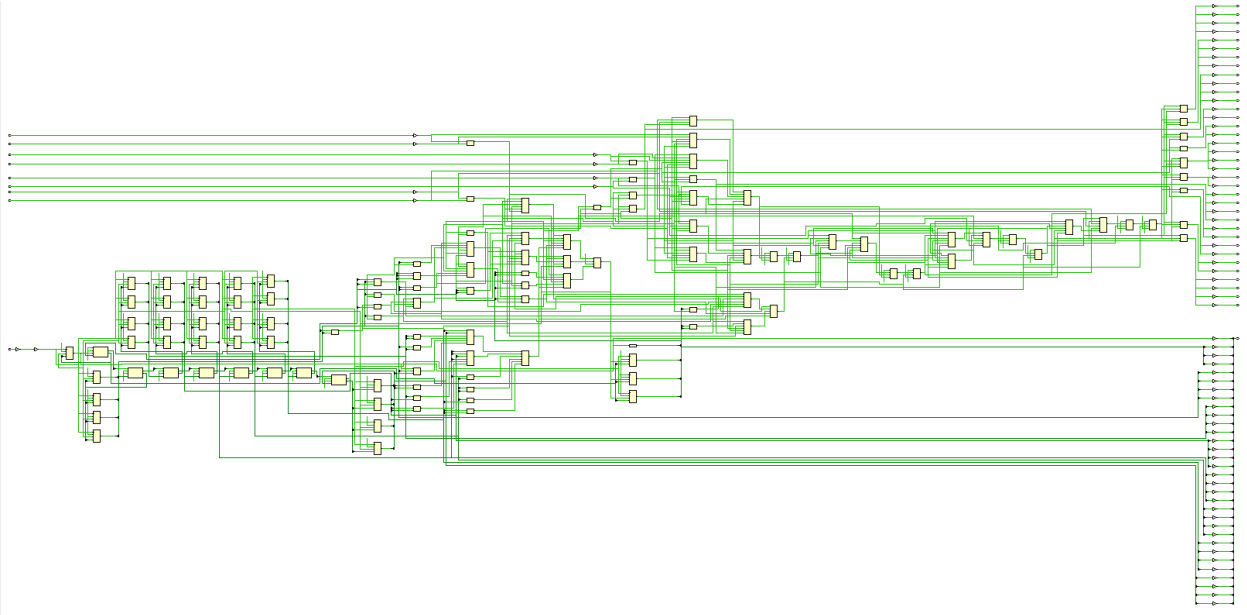

lightN
lightNR
lightNL
lightS
lightSR
lightSL
lightE
lightER
lightEL
lightW
lightWR
lightWL
timer
R
Y
R
R
R
Y
R
R
R
Y
R
R
Y
1
G
Y
R
G
R
R
G
Y
R
G
R
R
R
G
R
R
R
G
R
R
R
G


R
Y
R
G
R
Y
R
G
G
Y
R
G
Y
R
G
Y
R

6. RTL Schematic:



7. Synthesized Schematic:



8. Conclusions:

    a. An FPGA has a lot of flexibility when using it to design something. An MCU on other hand is limited by its components. Depending on the application but if you were to upgrade or complete a design quickly then an FPGA allows for a lot of power/processing speed as it can compute things parallelly. This allows for some applications which may be throttled by the hardware they are being implemented on be upgraded easily by an FPGA. Also, the flexibility of the FPGA allows for you to implement or create a lot of things on your own while an MCU would require another purchase which requires recoding the program.
    b. Most FPGA's don't have non-volatile memory so if for some reason power was to shut off then the FPGA would stop functioning. This could be solved though by having external wiring that programs the device upon reboot, but this requires more work and to learn the internal hardware of the device/PCB it is on. An FPGA also requires a lot more power in comparison to other devices so if you had a limit then that would have to be taken into consideration as well.