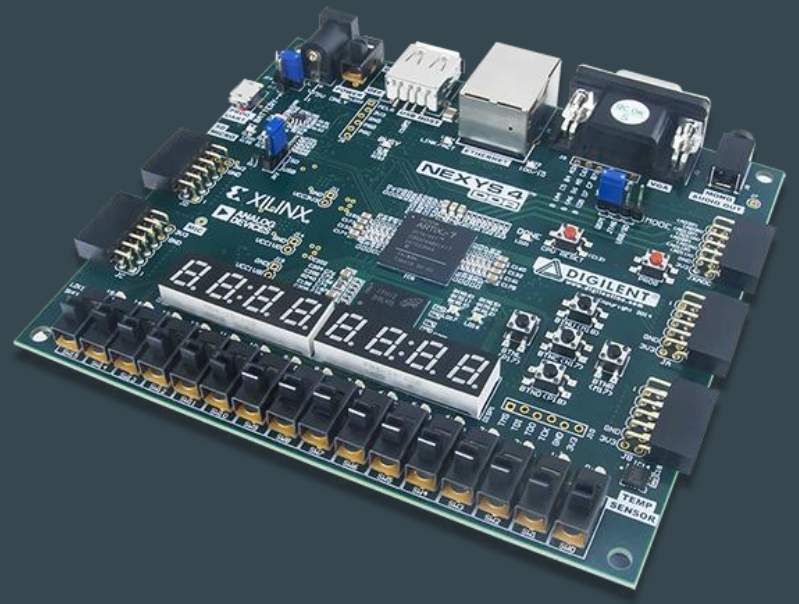
ECE 2700
Fall 2017

# Vhdl 4-Way Traffic Controller

Group members:
Esteban Rodriguez
Jason Gilliam
Allison Nguyen
Michelle Nguyen

# Agenda

- Concept
- Design Implementation
- Demo
- Improvement




# Bill of Materials

- Nexys4 DDR board
- Bread board
- 20 LEDs (Red, yellow, green and blue)
- 20 Resistors


# Concept

Use Nexys4 DDR to control the time of a 4-way traffic signal

Manually control crosswalk indicator.

Modify signal timing for heavy traffic.

# Concept

Stop indicated by Red LED

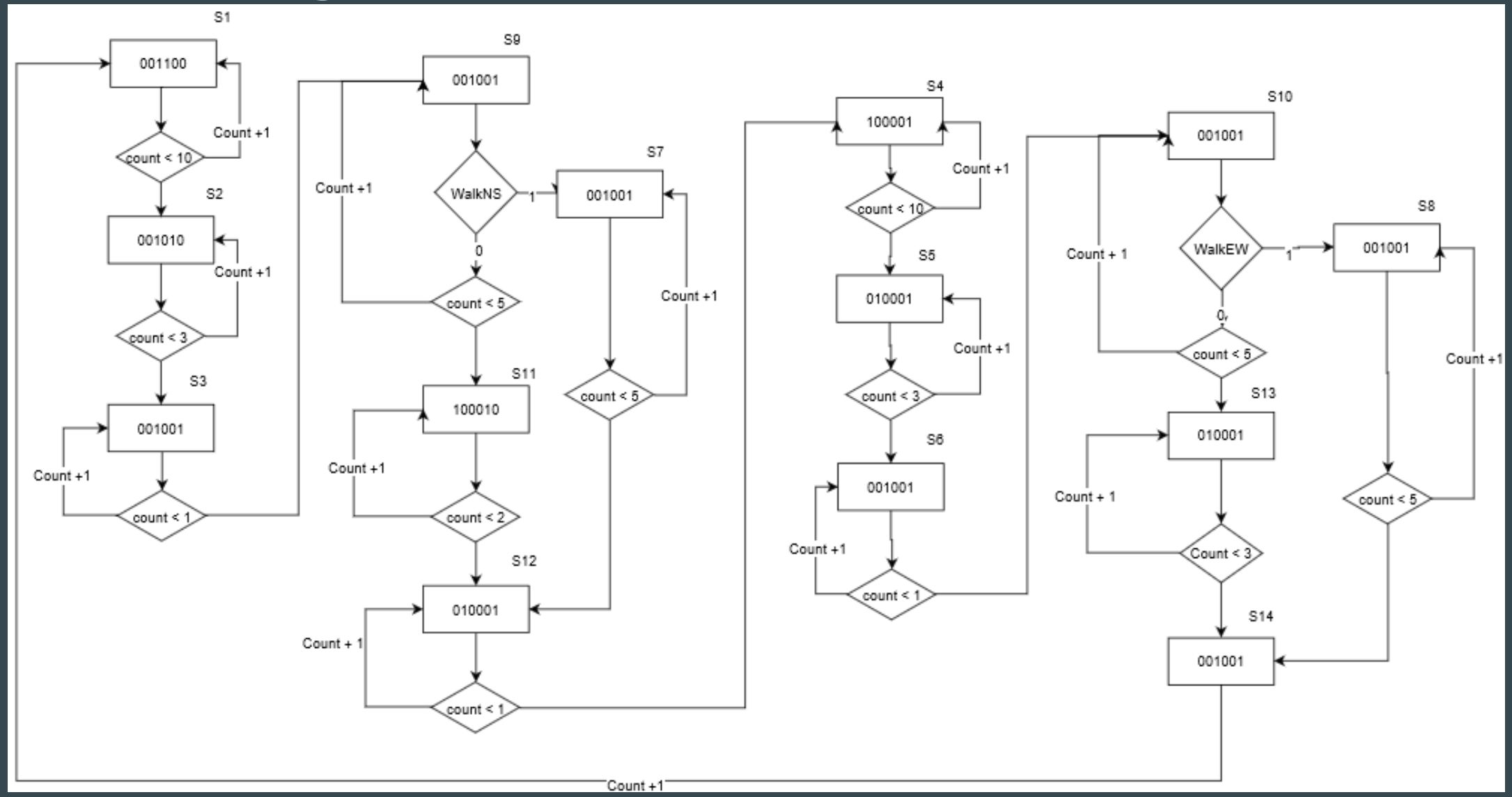
Caution indicated by Yellow LED

Go indicated by Green LED

Turn Left indicated by Yellow LED (Flash when Red)

Walk indicated by Blue LED

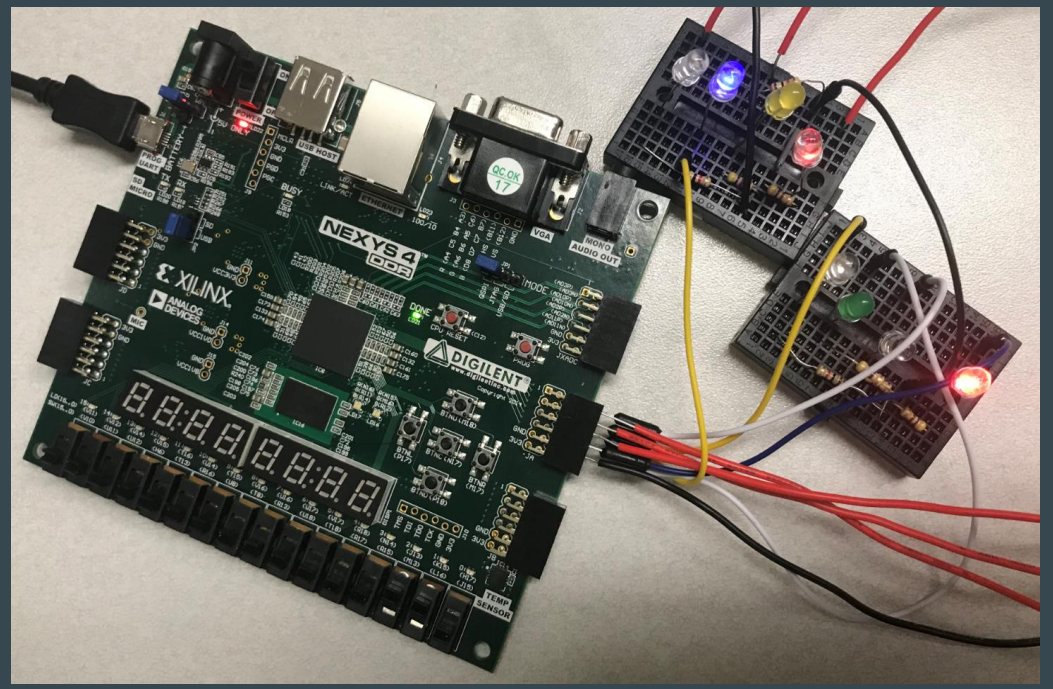
# State Diagram



# Implementation

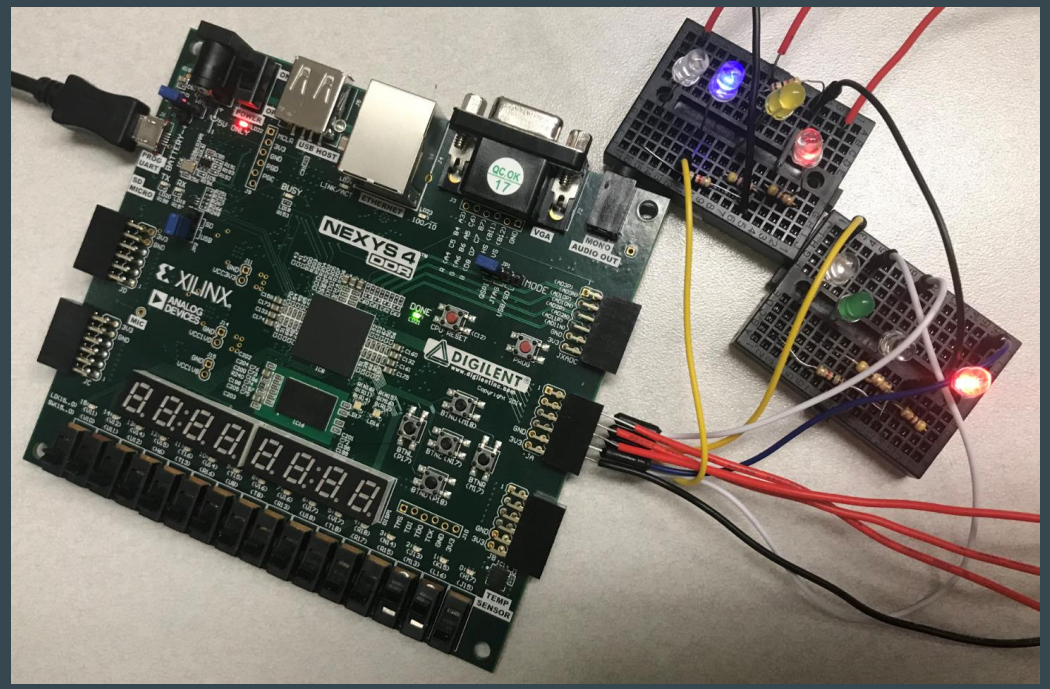
- Bread board
  - 5 LEDs each connected to a pull-down resistor
  - Each LED connect to PMOD port (JA & JB)

- Control
  - Rush hour mode enabled by SW0
  - EW crosswalk enabled by SW1
  - NS crosswalk enabled by SW2
  - Secondary road turn enabled by SW3

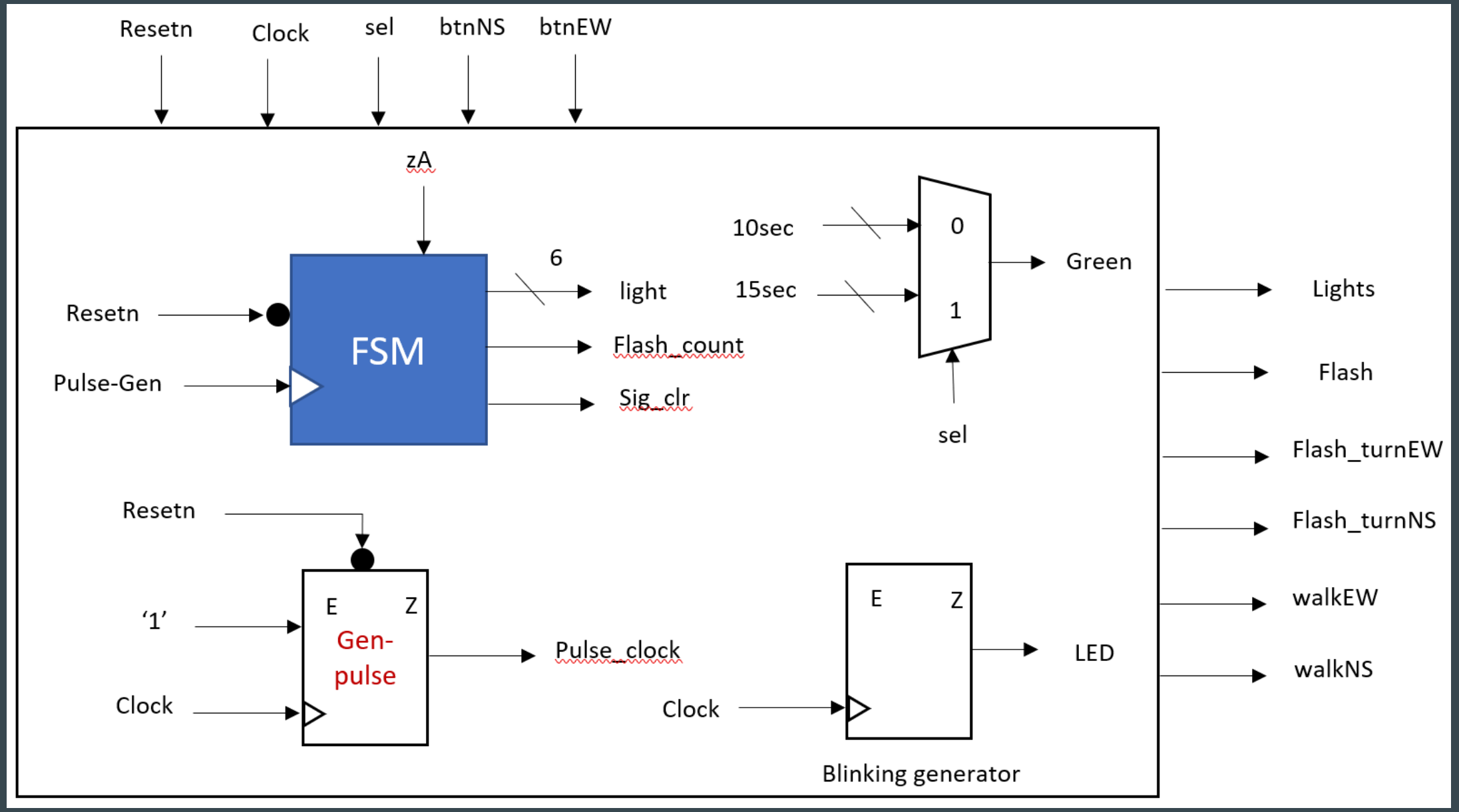


# Implementation

- Top file and 2 sub functions
  - Gen pulse function
  - MUX (10 or 15 sec green light)
  - Blinking function for Cross Walk LED
  - Finite State Machine for controlling lights
    - Timing of  the light



# Top File Design



# Blink LED Signal

```
constant max_count : natural := 48000000;
signal Rst : std_logic;
```

```
-- 0 to max_count counter
compteur : process(refclk, Rst)
    variable count : natural range 0 to max_count;
        begin
            if Rst = '1' then
                count := 0;
                led <= '1';
            elsif rising_edge(refclk) then
                if count < max_count/2 then
                    count := count + 1;
                    led <= '1';
                elsif count < max_count then
                    led <= '0';
                    count := count + 1;
                else
                    led <= '1';
                    count := 0;
                end if;
            end if;
end process compteur;

end struct;
```

# Finite State Machine Code

```
99  process (pulse_clk, resetn)
100
101 begin
102     if resetn = '0' then y <= S1;
103        seconds_count <= (others => '0');
104     elsif (pulse_clk'event and pulse_clk = '1') then
105        case y is
106                when S1 =>
107                if seconds_count < Green_Light_Time then
108                    y <= S1;
109                    seconds_count <= seconds_count + 1;
110                else
111                    y <= S2;
112                    seconds_count <= (others => '0');
113                end if;
114
115                when S2 =>
116                if seconds_count < three_sec then
117                    y <= S2;
118                    seconds_count <= seconds_count +1;
119                else
120                    y <= S3;
121                    seconds_count <= (others => '0');
122                end if;
```

```
123
124                when S3 =>
125                if seconds_count < one_sec then
126                    y <= S3;
127                    seconds_count <= seconds_count +1;
128                else
129                    y <= S9;
130                    seconds_count <= (others => '0');
131                end if;
132
133                when S4 =>
134                if seconds_count < Green_Light_Time then
135                    y <= S4;
136                    seconds_count <= seconds_count +1;
137                else
138                    y <= S5;
139                    seconds_count <= (others => '0');
140                end if;
141
142                when S5 =>
143                if seconds_count < three_sec then
144                    y <= S5;
145                    seconds_count <= seconds_count +1;
146                else
147                    y <= S6;
148                    seconds_count <= (others => '0');
149                end if;
```

# Finite State Machine Output

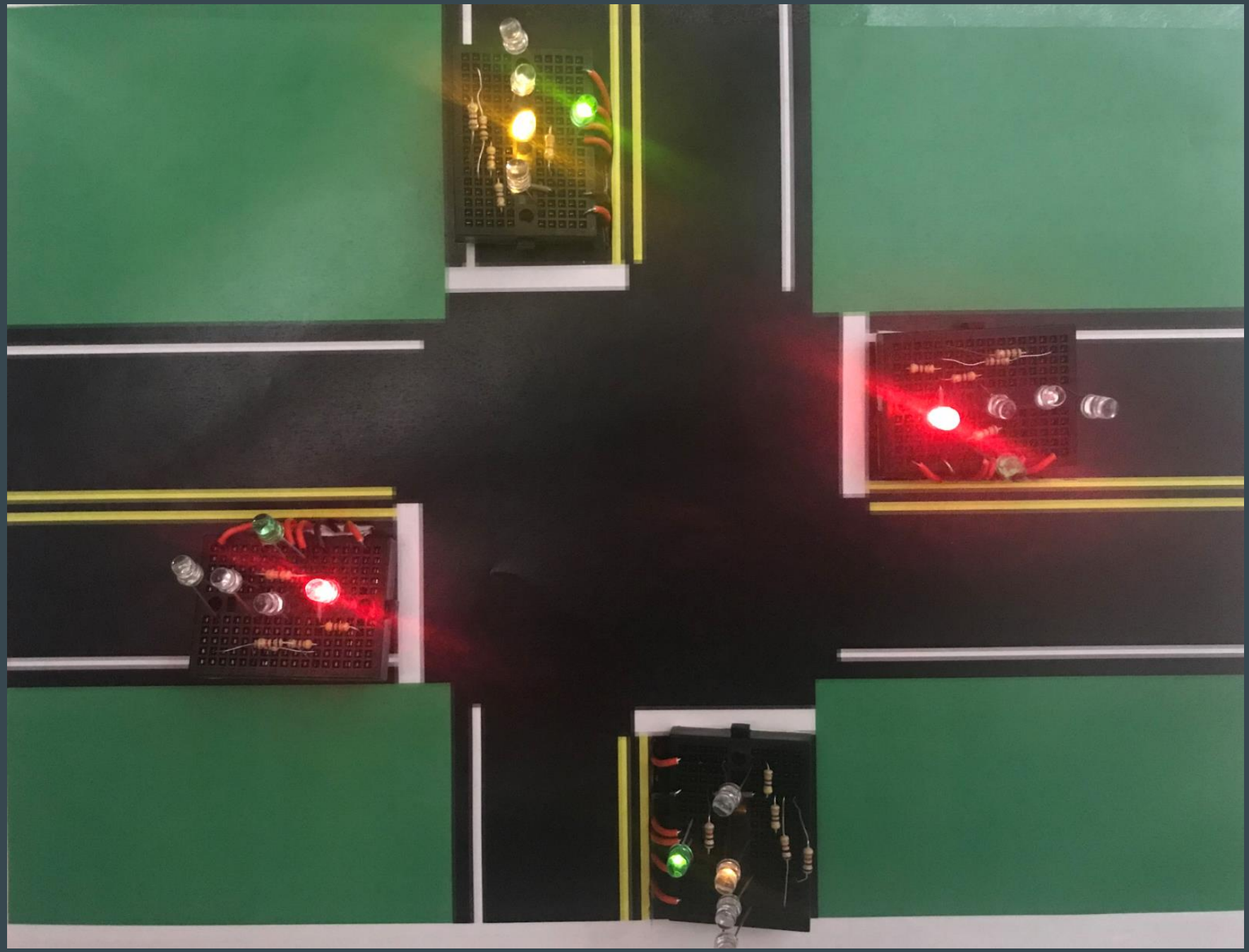
```
Outputs: process (y, seconds_count)
   begin
       walkNS <= "0";
       walkEW <= "0";
       flash_turnNS <= '0';
       flash_turnEW <= '0';


       case y is

           when S1 => if btnnoflash = '1' then flash_turnNS <= '0'; lights <= "001100"; else flash_turnNS <= blink_led_turn; lights <= "001100"; end if;
           when S2 => lights <= "001010";
           when S3 => lights <= "001001";
           when S4 => if btnnoflash = '1' then flash_turnEW <= '0'; lights <= "100001"; else flash_turnEW <= blink_led_turn; lights <= "100001"; end if;
           when S5 => lights <= "010001";
           when S6 => lights <= "001001";
           when S7 => walkNS <= "1"; lights <= "001001";
           when S8 => walkEW <= "1"; lights <= "001001";
           when S9 => flash_turnNS <= '1'; lights <= "001001";
           when S10 => flash_turnEW <= '1'; lights <= "001001";
           when S11 => lights <= "001001";
           when S12 => lights <= "001001";
           when S13 => lights <= "001001";
           when S14 => lights <= "001001";
       end case;

   end process;
```

# Demo



# Improvement

Below is the list of some features we would like to add into the system to make it better:

1)Detection of traffic volume by adding more logics into the coding and sensors --> More efficient.
2)Emergence vehicle detection such as ambulance, police etc…by using wireless sensor network at the signal intersection --> Efficient operation during emergency mode.
3)It can give more time to an intersection approach that is experiencing heavy traffic, or shorten or even skip a phase that has little or no traffic waiting for a green light --> Less waiting time.
4)Traffic signals are activated to detect the approach of a train near a rail crossing --> Reduce unnecessary stopping and safety.