Four Way Traffic Lights

ECE 2700 Winter 2021

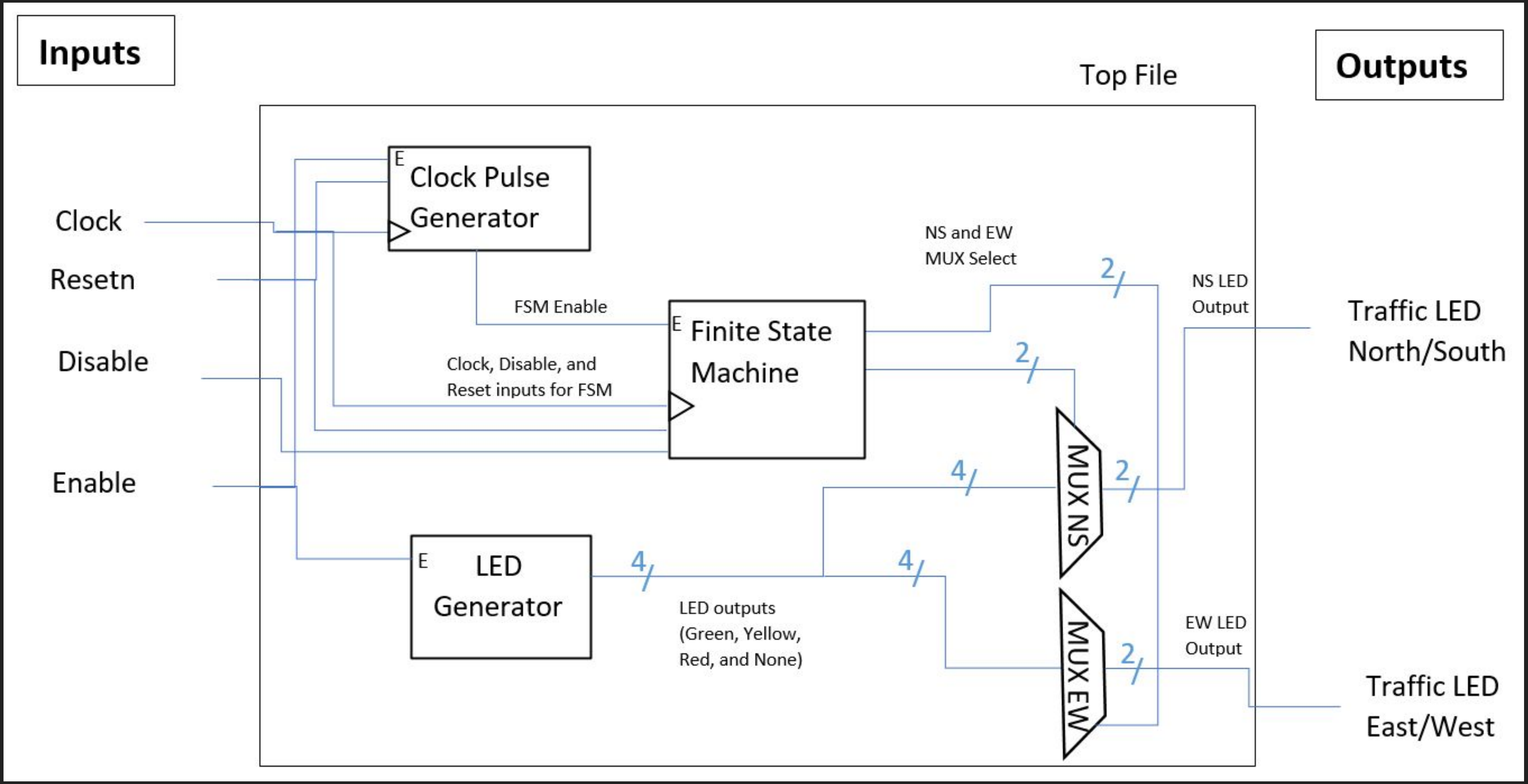
List of Authors (Justin Akhahon, John Antosh, Faristina Jackson, Daorsa Dulaj)

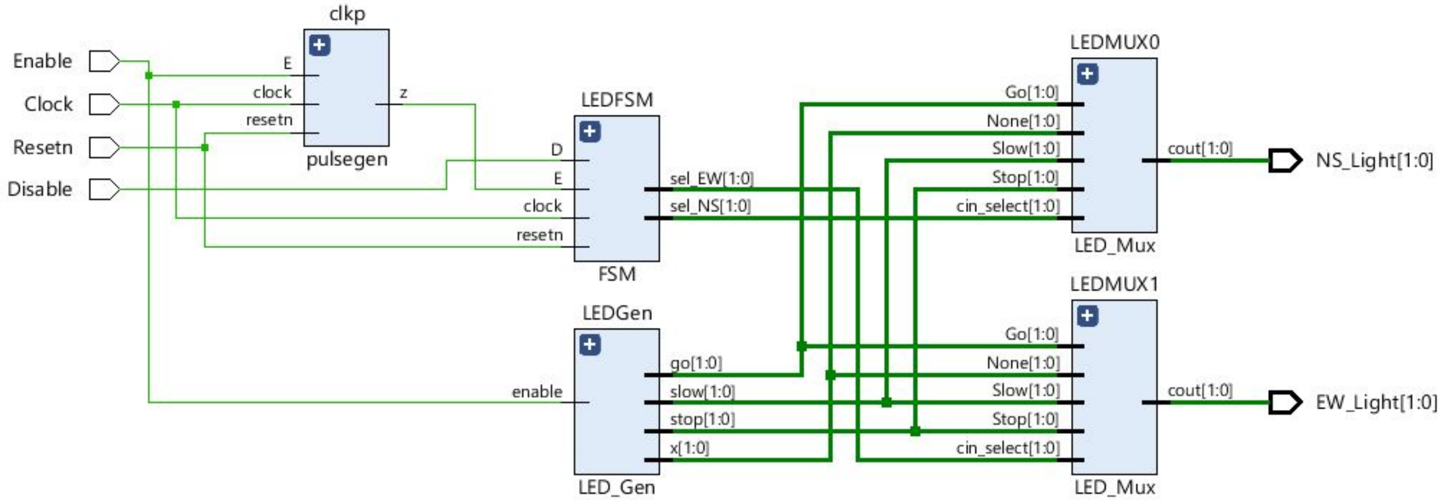
The circuit: An overview

- Our circuit is a 4-way traffic light controller
- It controls a sequence of different colored LEDs for two perpendicular roads.
- Our circuit has four inputs:
  - Clock - the internal circuit clock
  - Enable - when shut off, turns off the LEDs
  - Disable - when shut off, sets the LEDs to a blinking red state
  - Resetn - when shut off, sets the LEDs to a solid red state
- Our circuit has two outputs:
  - North/South road
  - East/West road

(Two LEDs emulate the sequence for lights in all four directions)

Block Diagram

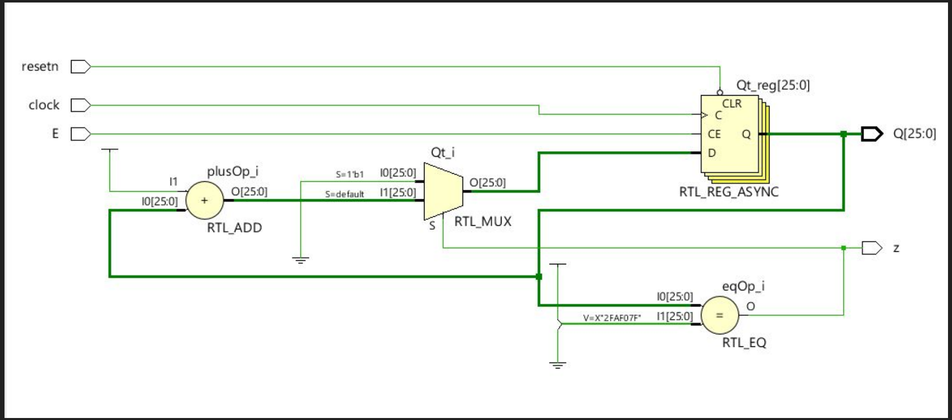


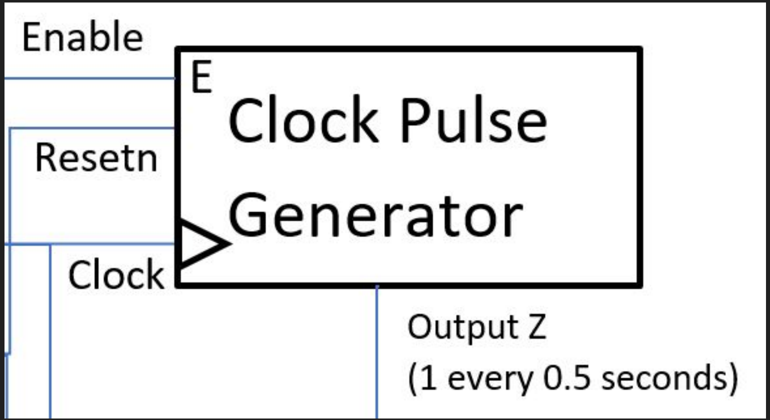


clkp
pulsegen
E
clock
resetn
z
Enable
Clock
Resetn
Disable
LEDFSM
FSM
D
E
clock
resetn
sel_EW[1:0]
sel_NS[1:0]
LEDGen
LED_Gen
enable
go[1:0]
slow[1:0]
stop[1:0]
x[1:0]
LEDMUX0
LED_Mux
Go[1:0]
None[1:0]
Slow[1:0]
Stop[1:0]
cin_select[1:0]
cout[1:0]
NS_Light[1:0]
LEDMUX1
LED_Mux
Go[1:0]
None[1:0]
Slow[1:0]
Stop[1:0]
cin_select[1:0]
cout[1:0]
EW_Light[1:0]

Clock Pulse Generator

For this project, we used a clock pulse generator.
Following the state diagram, the pulse generator was placed at the beginning of the program. The purpose of a pulse generator is to control the pulse ticks and the moments when the clock hits high. It serves as an input for our Finite State Machine. For this project we are using a 4-bit integer. The output from the pulse generator, then is used
by the FSM as an input to initialize the FSM.





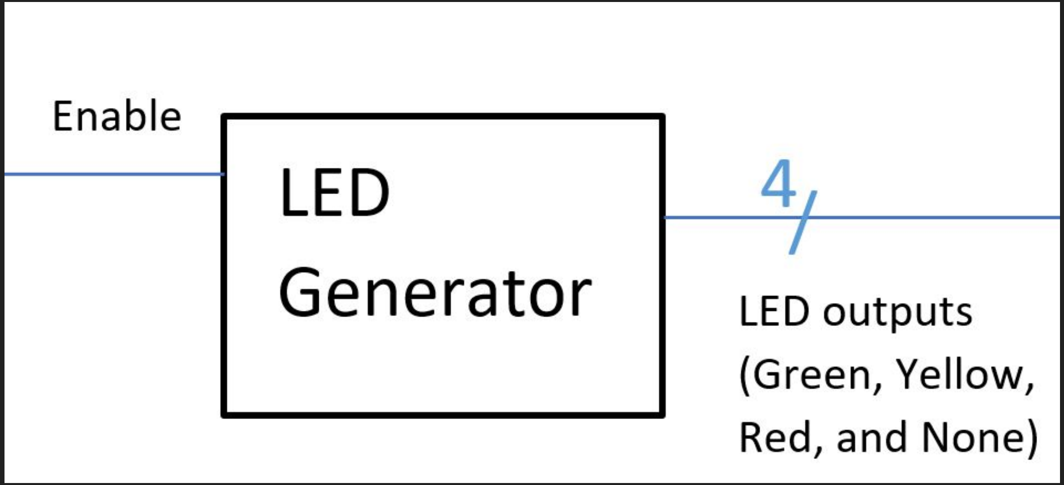
LED GENERATOR

Used to output our lights
Generated three different lights (Red, Yellow, and Green)
Programmed code to corresponded with different LED’s

(LED generator code)

```
begin

LED_gen: process (enable)
begin
if enable = '0' then go <="00"; slow <= "00"; stop <= "00"; x <= "00";
else go <="10"; slow <="11"; stop <="01"; x <="00";
end if;
end process;
```



MUX

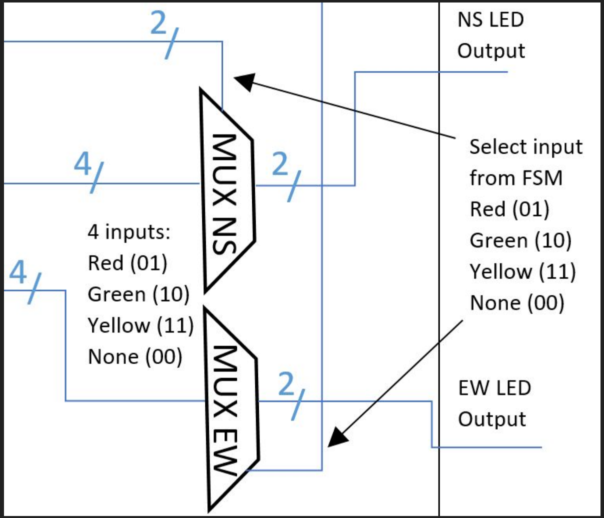
- Is used to regulate the state the LED light
- It selects data from the LED generator and it will be sent to the circuit
- It has four options to send to the LED output:
- RED=STOP (01)
- GREEN=GO (10)
- YELLOW=SLOW (11)
- NONE=NULL (00)
- The multiplexer outputs are sent to the external circuit LEDs

```
architecture struct of LED_Mux is

begin

with cin_select select

    cout <= Go when "10",
            Slow when "11",
            Stop when "01",
            None when others;
```

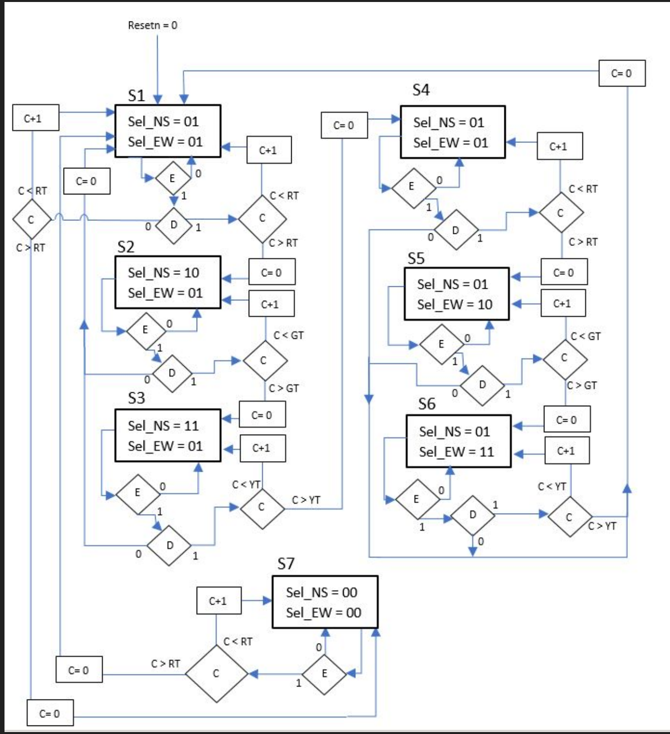


FINITE STATE MACHINE

- Used to represent system with a finite number of states
- Finite state machine toggles the multiplexer select inputs to switch to appropriate LED
- Outputs for FSM were designed to match the inputs for the multiplexer
- Receives enable input from Clock Pulse Generator (Set to 1 every .5 seconds)
- A counter is embedded inside the FSM to manage the state changes and the various time lengths of those states
- Disable input was created to instantaneously set the LEDs in a blinking red state. (An oscillation between S1 and S7)
- Resetn input sets the FSM state to S1

FINITE STATE MACHINE

- State diagram along with part of the FSM code




```
if resetn = '0' then
    y <= S1; counter <= (others=> '0');
elsif (clock'event and clock = '1') then
    case y is
        when S1 =>
            if E= '0' then y <= S1;
            elsif D = '0' then
                if counter > RT then counter <= (others=> '0'); y <= S7;
                else counter <= counter + 1; y <= S1; end if;
            elsif counter < RT then counter <= counter + 1; y <= S1;
            else y <= S2; counter <= (others=> '0'); end if;
        when S2 =>
            if E= '0' then y <= S2;
            elsif D = '0' then y <= S1;
            elsif counter < GT then counter <= counter + 1; y <= S2;
            else y <= S3; counter <= (others=> '0'); end if;
        when S3 =>
            if E= '0' then y <= S3;
            elsif D = '0' then y <= S1;
            elsif counter < YT then counter <= counter + 1; y <= S3;
            else y <= S4; counter <= (others=> '0'); end if;
        when S4 =>
            if E= '0' then y <= S4;
            elsif D = '0' then y <= S1;
            elsif counter < RT then counter <= counter + 1; y <= S4;
            else y <= S5; counter <= (others=> '0'); end if;
        when S5 =>
            if E= '0' then y <= S5;
            elsif D = '0' then y <= S1;
            elsif counter < GT then counter <= counter + 1; y <= S5;
            else y <= S6; counter <= (others=> '0'); end if;
        when S6 =>
            if E= '0' then y <= S6;
            elsif D = '0' then y <= S1;
            elsif counter < YT then counter <= counter + 1; y <= S6;
            else y <= S1; counter <= (others=> '0'); end if;
        when S7 =>
            if E= '0' then y <= S7;
            elsif counter < RT then counter <= counter + 1; y <= S7;
            else y <= S1; counter <= (others=> '0'); end if;
    end case;
```

Project Simulation

- The video to the right is demonstrating all the code working together.

- It will display 7 different states