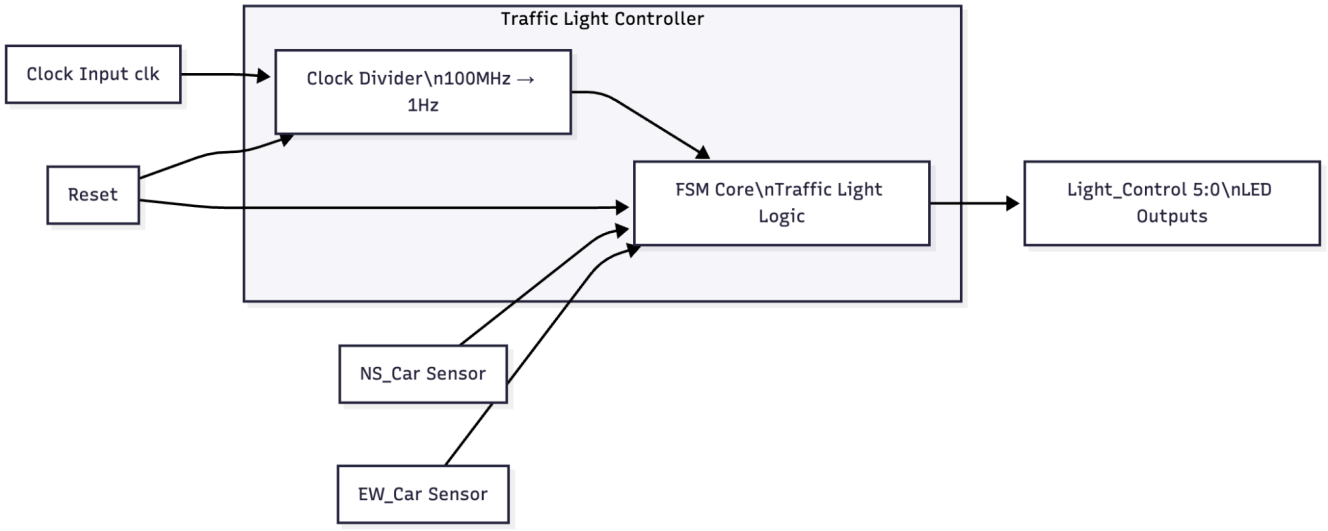
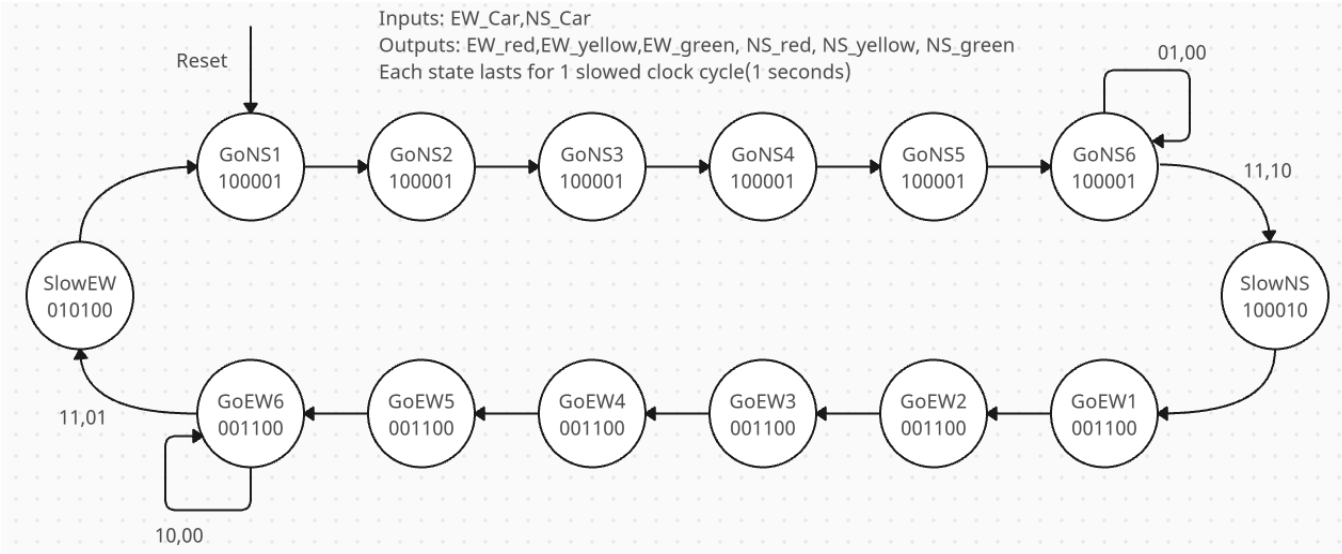
Project Report: Traffic Light Controller
Team 3 Members: Andrew Mousad, Libin Jian, Jerome Poonvichit, Mohamed Khalifa

Introduction:

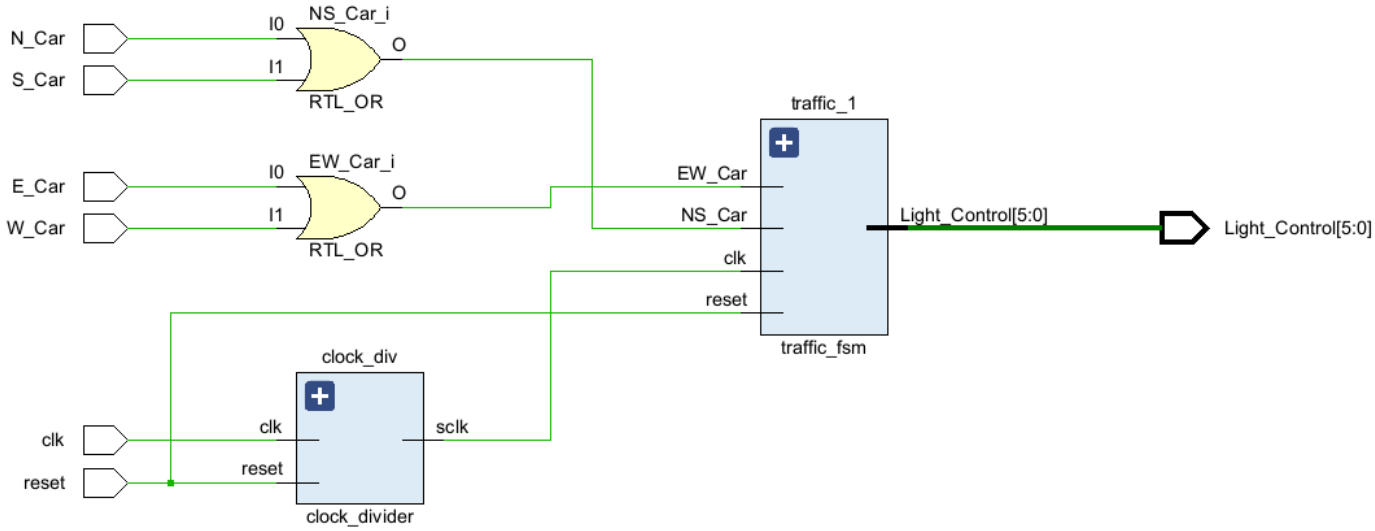
For this project, a Finite State Machine (FSM), specifically a Moore Machine is being utilized and implemented into a traffic light controller system. The outputs will turn the LEDs on the FPGA board on and off based on the use of timers, FSM and a clock divider. For the timer, a clock divider is required to reduce the speed of the board's clock in order to be able to time each state. As for the LEDs, the LEDs on top of the switches on the FPGA board are used for this project.

System Diagram / State Diagram:





RTL Schematic:



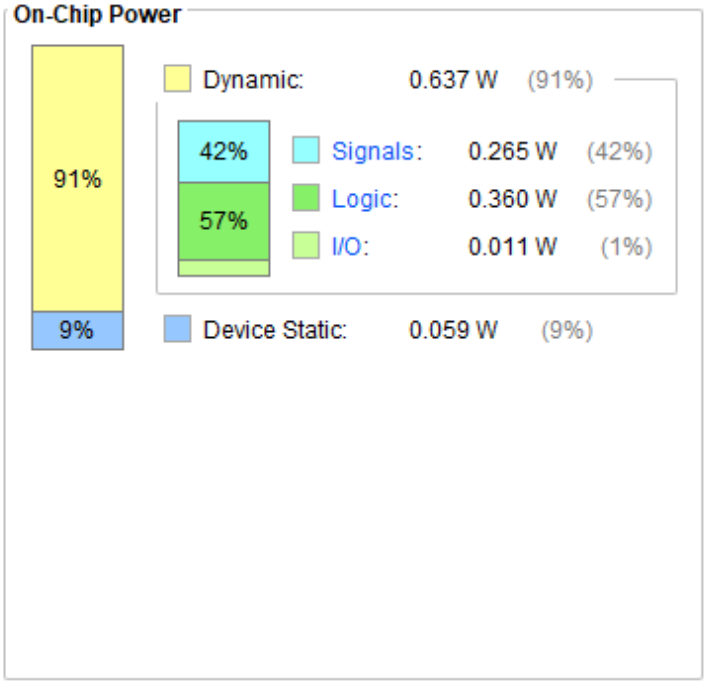
Post-Implementation Area / Power / Static Timing Report:

Name	Slice LUTs (32600)	Slice Registers (65200)	Slice (8150)	LUT as Logic (32600)	Bonded IOB (210)	BUFGCTRL (32)
traffic_light_controller	47	43	23	47	10	1

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.696 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**28.3°C**
Thermal Margin:	71.7°C (14.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Total On-Chip Power: 0.696 W

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.875 ns	Worst Hold Slack (WHS):	0.202 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	33	Total Number of Endpoints:	33	Total Number of Endpoints:	34

All user specified timing constraints are met.

Demonstration:
[demo.mp4]

Discussion:

For this project, we built a working traffic light controller using a Moore Machine. The clock divider enables real-time LED timing to slow down the FPGA clock. We also implemented state changes based on N, S, E, and W emulating sensors of the car's direction as button inputs. Moreover, we verified the correct transitions and light outputs on the FPGA board. Lastly, we demonstrated key EGEC 446 concepts such as FSM design, timing, and sequential logic.

Challenges:

The challenges we faced were mainly deciding how many states we want to include for a smooth traffic flow and to be able to distinguish between each result. This included making sure the LED light patterns clearly matched each traffic phase. In addition, we also needed to adjust the clock speed so the demonstration will look clean and easy to follow. We also needed to ensure the sensors, Northsouth (NS) and Eastwest (EW) buttons, triggered the state changes at the right moment.

If we were to redo this project, we would implement the RGB LEDs on the FPGA board to show the color of the traffic lights. This would make the traffic light easier to understand, rather than having to look at the positions of the lit LEDs.

References:

https://www.chipverify.com/verilog/verilog-fsm#verilog-fsm-structure

Appendix

Verilog Code:

traffic_light_controller.v:

```
`timescale 1ns / 1ps

module traffic_light_controller(
    input N_Car,
    input S_Car,
    input E_Car,
    input W_Car,
    input clk,
    input reset,
    output [5:0] Light_Control
);
wire NS_Car, EW_Car;
wire slow_clk;
assign NS_Car = N_Car | S_Car;
assign EW_Car = E_Car | W_Car;

// clock divider divides the system clock into a 1-second clock
clock_divider clock_div1(
  .sclk(slow_clk),
  .clk(clk),
  .reset(reset)
);
traffic_fsm traffic_1(
  .NS_Car(NS_Car),
  .EW_Car(EW_Car),
  .clk(slow_clk),
  .reset(reset),
  .Light_Control(Light_Control)
);
endmodule
```

Traffic_fsm.v:

```
`timescale 1ns / 1ps

module traffic_fsm(
    input NS_Car,
    input EW_Car,
    input clk,
    input reset,
    output reg [5:0] Light_Control
    );
wire NS_Car, EW_Car, clk, reset;
reg [3:0] current_state, next_state;
// each state will last for 1 clock cycle and with clock divider, a clock cycle is 1 second
```

```
parameter GoNS1 = 4'b0001, GoNS2 = 4'b0010, GoNS3 = 4'b0011, GoNS4 = 4'b0100, GoNS5 =
4'b0101, GoNS6 = 4'b0110, // NS green, EW red
        SlowNS = 4'b0111, // NS yellow, EW red
        GoEW1 = 4'b1000, GoEW2 = 4'b1001, GoEW3 = 4'b1010, GoEW4 = 4'b1011, GoEW5 = 4'b1100,
GoEW6 = 4'b1101, // EW green, NS red
        SlowEW = 4'b1110; // EW yellow, NS red

// light bits follow: EWred, EWyellow, EWgreen, NSred, NSyellow, NSgreen
parameter [5:0] GoNS_light = 6'b100001, SlowNS_light = 6'b100010,
              GoEW_light = 6'b001100, SlowEW_light = 6'b010100;

// update current state on posedge clock with asynchronous reset
always @(posedge(clk), posedge(reset))
begin
  if(reset)
    current_state <= GoNS1;
  else
    current_state <= next_state;
end

// next state logic
always@(*)
begin
  case(current_state)
    GoNS1, GoNS2, GoNS3, GoNS4, GoNS5, GoEW1, GoEW2, GoEW3, GoEW4, GoEW5: // lights will
finish their cycle regardless of input
      next_state = current_state + 1;
    GoNS6:  // NS lights will change if there are cars detected on EW
      if(NS_Car & ~EW_Car)
        next_state = GoNS6;
      else if(NS_Car & EW_Car)
        next_state = SlowNS;
      else if(EW_Car & ~NS_Car)
        next_state = SlowNS;
      else if(~NS_Car & ~EW_Car)
        next_state = GoNS6;
    GoEW6:  // EW lights will change if there are cars detected on NS
      if(~NS_Car & EW_Car)
        next_state = GoEW6;
      else if(EW_Car & NS_Car)
        next_state = SlowEW;
      else if(~EW_Car & NS_Car)
        next_state = SlowEW;
      else if(~EW_Car & ~NS_Car)
        next_state = GoEW6;
    SlowNS: // next state will be GoEW1
      next_state = GoEW1;
    SlowEW: // next state will be GoNS1
      next_state = GoNS1;
```

```
    default:
      next_state = GoNS1;
  endcase
end
// LED output is clocked
always @(posedge(clk))
begin
    case(current_state)
      GoNS1, GoNS2, GoNS3, GoNS4, GoNS5, GoNS6: begin
        Light_Control <= GoNS_light;
      end
      GoEW1, GoEW2, GoEW3, GoEW4, GoEW5, GoEW6: begin
        Light_Control <= GoEW_light;
      end
      SlowNS: begin
        Light_Control <= SlowNS_light;
      end
      SlowEW: begin
        Light_Control <= SlowEW_light;
      end
      default: begin
        Light_Control <= Light_Control;
      end
    endcase
end
endmodule
```

clock_divider.v:

```
module clock_divider(

input clk,
input reset,
output reg sclk
);
reg [31:0] count;
    always@(posedge clk or posedge reset)
        begin
    if(reset == 1'b1) begin
    count <= 32'd0;
    sclk <= 1'b0;
        end else begin
    if(count == 32'd50000000) begin //50000000 for 1 sec 250000000 for 5 sec
    count <= 32'd0;
    sclk <= ~sclk;
        end else begin
    count <= count + 1;
        end
    end
    end
```

```
endmodule
```