Experiment 12:

Traffic Light Controller

Objectives

In this experiment, you will:

- ✓ Learn how to write a Verilog model for a given finite state machine.
- ✓ Design and implement a traffic light controller.
- ✓ Perform simple experimental design.

Material Required

- ✓ An FPGA prototyping board
- ✓ Design and simulation software tools
- ✓ `OneHz` module from previous experiment

Design Overview

In this lab, you are going to develop a Finite State Machine (FSM) for a traffic light controller that will control the operation of traffic lights of three roads at a crossing as shown in Figure 1. It also shows the three traffic light signals, S1, S2 and S3.

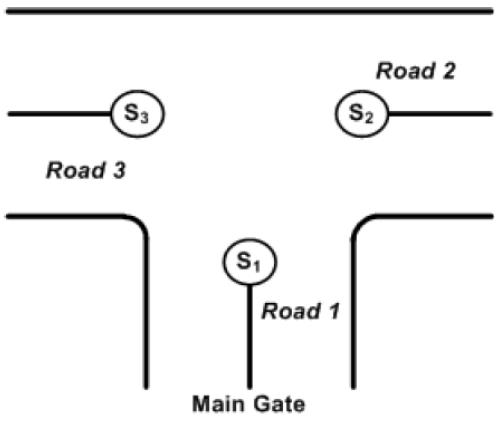

Figure 1: The three traffic light signals at a crossing.

Design Specifications

There are three traffic light signals, S1, S2, and S3, each alternating between two states, RED and GREEN. These signals control the traffic flow on the three roads: road1, road2 and road3 in four possible states as follows:

- In STATE 0: There is no traffic: so priority is given to road1. (S1 = GREEN, S2 = RED, S3 = RED)
- In STATE 1: traffic is coming through road1. (S1 = GREEN, S2 = RED, S3 = RED)
- In STATE 2, traffic is coming through road2. (S1 = RED, S2 = GREEN, S3 = RED)
- In STATE 3, traffic is coming through road3. (S1 = RED, S2 = RED, S3 = GREEN)

An illustration of the last three states is shown in Figure 2.

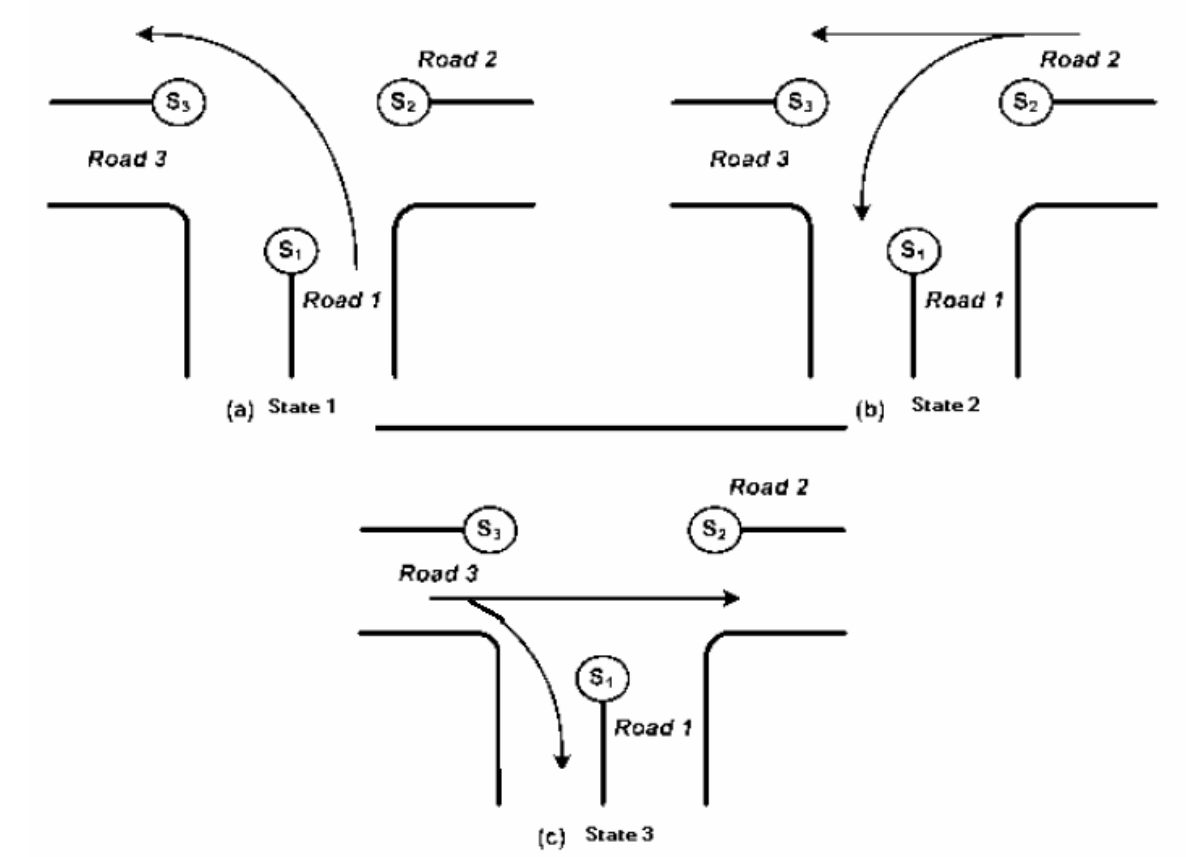

Figure 2: Illustration of the last three states.

The operation of the three traffic light signals, S1, S2, and S3, is controlled through an arrangement of traffic sensors and traffic light controller circuit as shown in Figure 3. There are three traffic sensors X1, X2, and X3, which sense the presence of traffic on the three roads, as illustrated in Table 1. The controller operation is determined by the output of these three sensors as enumerated in Table 2.

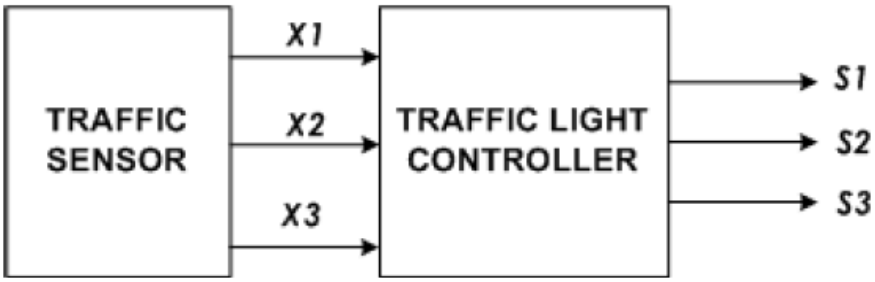

Figure 3: Traffic sensor and traffic light controller circuit.

Table 1: Traffic sensor signals.

X3	X2	X1	Indication
0	0	0	No traffic in all roads
0	0	1	Traffic in Road 1 only
0	1	0	Traffic in Road 2 only
0	1	1	Traffic in Road 1 and Road 2 only
1	0	0	Traffic in Road 3 only
1	0	1	Traffic in Road 1 and Road 3 only
1	1	0	Traffic in Road 2 and Road 3 only
1	1	1	Traffic in all Roads

Table 2: Traffic sensor and controller operation.

X3	X2	X1	Indication
0	0	0	Go to STATE 0
0	0	1	Go to STATE 1
0	1	0	Go to STATE 2
0	1	1	Alternate between STATE 1 and STATE 2, going to STATE 1 from other states
1	0	0	Go to STATE 3
1	0	1	Alternate between STATE 1 and STATE 3, going to STATE 1 from other states
1	1	0	Alternate between STATE 2 and STATE 3, going to STATE 2 from other states
1	1	1	Normal operation: STATE 1, STATE 2, STATE 3, STATE 1…

Assume that the controller has three inputs: X1, X2, and X3 coming from the traffic sensors, and three outputs: S1, S2, and S3, which control the operation of the three traffic light signals (logic 1 represents a GREEN signal and logic 0 represents a RED signal).

FSM Modeling in Verilog

A synchronous sequential circuit is composed of data storage elements, often implemented by D flip flops, and combinational logic to implement the output and the next state functions as shown in Figure 4. The inputs to the combinational logic are the current state variables and the primary inputs while the outputs of the combinational logic are the next state variables and primary outputs.

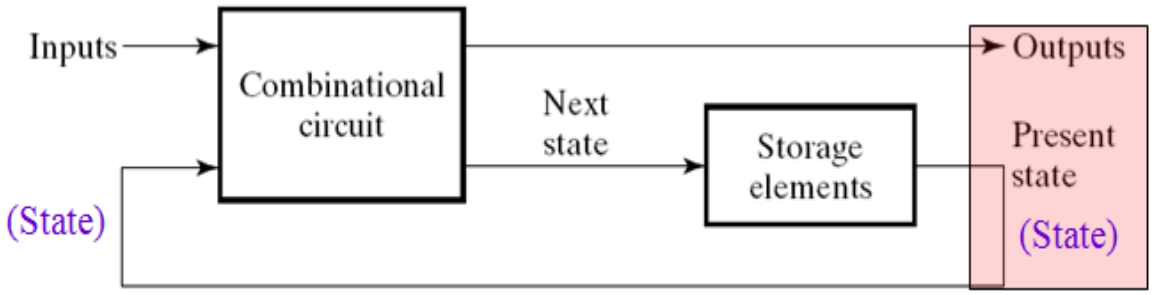

Figure 4: Sequential circuit model.

To write a Verilog model for a sequential circuit, we can have two always blocks, one modeling the storage elements and another modeling the combinational logic. As an illustrative example, let us consider the Moore sequence detector for detecting the sequence 1 followed by 1 followed by 0, shown in Figure 5.

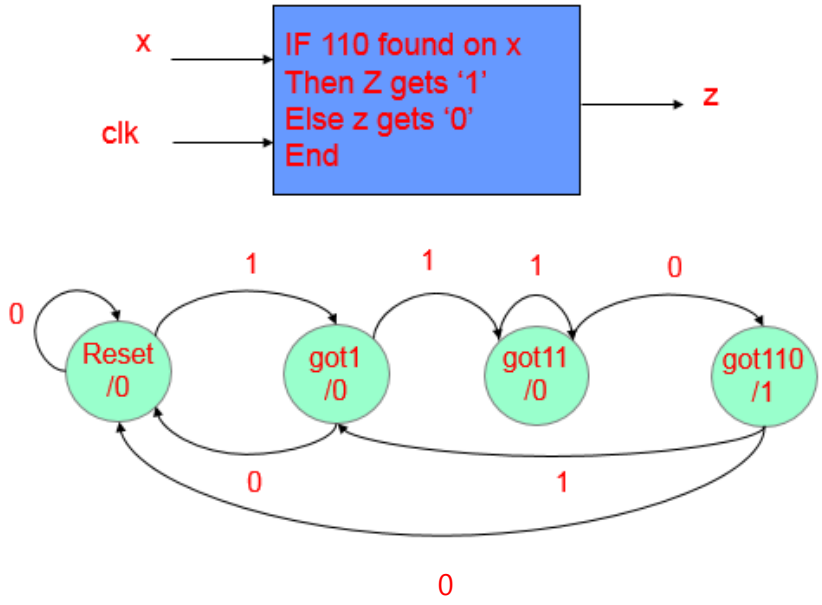

Figure 5: Moore sequence detector (detecting sequence 110).

The Verilog model for this Moore sequence detector is given below:

```
module moore_110_detector (output reg z, input x, clk, rst);

parameter reset = 2'b00, got1=2'b01, got11=2'b10, got110=2'b11;

reg [1:0] state, next_state;

always @(posedge clk, posedge rst)
  if (rst) state <= reset; else state <= next_state;

always @(state, x) begin
z = 0;
case (state)
      reset: if (x)  next_state=got1; else next_state=reset;
      got1:  if (x)  next_state=got11; else next_state=reset;
      got11: if (x)  next_state=got11; else next_state=got110;
      got110: begin
                z=1;
                if (x) next_state=got1;
                else next_state=reset;
              end
endcase
end
endmodule
```

Note that parameter is used to define the used state codes. We also need to define two variables of type reg one to represent the current state (state) and the other to represent the next state (next_state). The first always block models the storage elements using D-FFs assuming asynchronous reset. The second always block models the combinational block behavior using a case statement. You can use this model to model any FSM.

Tasks

1. Complete the state table given below for the traffic light control, assuming Moore model. Assume that STATE0 is the reset state. Verify the correctness of your state table with your lab instructor.

Current State	Input			Next State	Output
	X3	X2	X1		S3 S2 S1
STATE0	0	0	0	STATE0	0 0 1
STATE0	0	0	1	STATE1	0 0 1
STATE0	0	1	0	STATE2	0 0 1
STATE0	0	1	1	STATE1	0 0 1

STATE0	1	0	0	STATE3	0 0 1
STATE0	1	0	1	STATE1	0 0 1
STATE0	1	1	0	STATE2	0 0 1
STATE0	1	1	1	STATE1	0 0 1
STATE1	0	0	0		
STATE1	0	0	1		
STATE1	0	1	0		
STATE1	0	1	1		
STATE1	1	0	0		
STATE1	1	0	1		
STATE1	1	1	0		
STATE1	1	1	1		
STATE2	0	0	0		
STATE2	0	0	1		
STATE2	0	1	0		
STATE2	0	1	1		
STATE2	1	0	0		
STATE2	1	0	1		
STATE2	1	1	0		
STATE2	1	1	1		
STATE3	0	0	0		
STATE3	0	0	1		
STATE3	0	1	0		
STATE3	0	1	1		
STATE3	1	0	0		
STATE3	1	0	1		
STATE3	1	1	0		
STATE3	1	1	1		

2. Complete the following Verilog model for the FSM of the traffic light controller:

```
module TLC (input CLK,  Reset, EN, X3, X2, X1, output reg S3, S2,
S1, output st1, st0);

parameter STATE0 = 2'b00, STATE1 =2'b01, STATE2 =2'b10, STATE3
=2'b11;

reg [1:0]  state, next_state;

assign st1 = state[1];
assign st0 = state[0];


always @(posedge CLK, posedge Reset)
  if (Reset) state <= STATE0;
  else  if (EN) state <= next_state;
```

```
always @(state, X1, X2, X3) begin
S1 = 0; S2 = 0; S3 = 0;
case (state)
    STATE0:
          begin
             S1=1;
             if ({X3,X2,X1}==3'b000) next_state=STATE0;
             else if ({X3,X2,X1}==3'b100) next_state=STATE3;
             else if ({X2,X1}==2'b10) next_state=STATE2;
             else  next_state=STATE1;
           end
     STATE1:




     STATE2:




     STATE3:




endcase
end
endmodule
```

3. Simulate your TLC module and verify its correct operation. Assuming that the circuit is initially in **state0**, determine the input sequence to obtain the following output sequence: 001, 100, 100, 001, 010, 001.

4. Implement the following Verilog module on FPGA, which changes between traffic light signals at a frequency of 1 HZ. Use the OneHZ module from previous labs. Test the implemented module by connecting CLK to the system clock (V10), Reset to one of the push buttons, X1, X2 and X3 to three switches, S1, S2, S3, st1, and st0 to LEDs. Verify correct operation of your implemented module.

```
module TLC_Test (input CLK, Reset, X3, X2, X1, output  S3, S2, S1,
st1, st0);

OneHZ M1 (CLK, Reset, CLK1HZ);
TLC M2 (CLK, Reset, CLK1HZ, X3, X2, X1,  S3, S2, S1, st1, st0);

endmodule
```

Grading Sheet

Task	*Points*
Complete the state table	10
Complete the state machine Verilog model	20
Simulate your traffic light controller	15
Implement your traffic light controller	30
Lab notebook and discussion	25