Lab #11

Introduction and Design of a Finite State Machine (FSM) in Verilog

Objectives

- Understand the fundamental concepts of Finite State Machines (FSMs) and differentiate between Mealy and Moore architectures.
- Apply Verilog behavioral modeling techniques to design, implement, and simulate FSM-based digital systems.
- Construct state diagrams, state tables, and state transition logic to translate system behavior into an FSM representation.
- Implement FSMs using sequential and combinational logic blocks in Verilog and verify correct operation through simulation tools.
- Demonstrate the ability to map real-world digital problems into FSM-based solutions, reinforcing logical thinking and design skills aligned with CLO/PLO requirements.

Introduction:

Finite State Machines (FSMs) are sequential circuits used in many digital systems to control the behavior of systems and dataflow paths. Examples of FSM include control units and sequencers.

A finite state machine is a mathematical model used to describe and analyze the behavior of systems that have a finite number of states. It consists of a set of states, a set of inputs or events that cause state transitions, and a set of rules that determine how the system transitions from one state to another in response to these events.

An FSM consists of the following parameters:

Q: finite set of states

∑: finite set of input symbols

q0: initial state

F: final state

δ: Transition function

The behavior of an FSM can be visualized using a state transition diagram, which is a directed graph where the nodes represent the states, and the edges represent the transitions between them. The FSM can be in only one state at a time, and the current state determines how the system will respond to the next input or event.

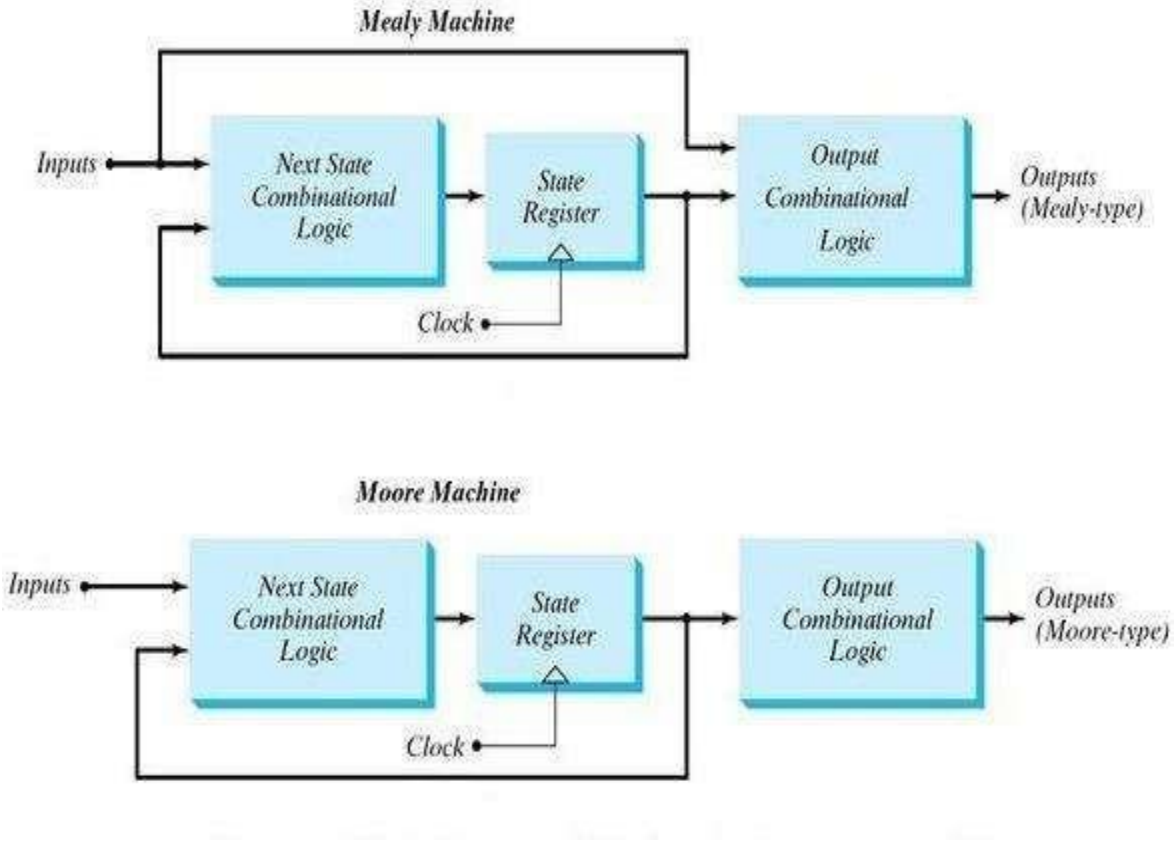
This lab introduces the concept of two types of FSMs, Moore and Mealy, and the modeling styles to develop such machines.

Moore Machine: A Moore machine is a type of finite state machine that produces an output based only on its current state. The output is associated with the state itself, rather than with the transition from one state to another. The Moore machine is named after its inventor, Edward F. Moore.

In a Moore machine, *the output is a function of the current state and does not depend on the input received*. The output is generated when the machine transitions from one state to another.

Mealy Machine: A Mealy machine is a type of finite state machine that produces an output based on both its current state and the input it receives. The output is generated when a transition occurs from one state to another and is a function of both the current input and the current state.

In a Mealy machine, the output is associated with the transition from one state to another, rather than with the state itself. This means that the output can change as the machine transitions from one state to another, depending on the input received.



Block diagram of Moore and Mealy Machine

Parameter: In Verilog, a parameter is a value that is declared at the module level and is used to define constants or default values for a module's inputs, outputs, or internal signals. Parameters are like constants in other programming languages and can be used to make a module more flexible and reusable. A parameter is declared using the **parameter** keyword, followed by the parameter name and value.

NOTE: *A Mealy machine typically has one less state than a Moore machine*, since Mealy machines can encode some of the output information in the transitions between states, rather than requiring a separate state for each possible output. This means that fewer states are needed to represent the same behavior.

Example#1: Design a Moore (FSM) circuit to detect a sequence (101).

a. State Diagram: Onboard

b. State Transition Table

Present State	Input x	Next State	Output z
S0	0	S0	0
S0	1	S1	0
S1	0	S2	0
S1	1	S1	0
S2	0	S0	0
S2	1	S3	0
S3	0	S0	1
S3	1	S1	1

CODE:

```
`timescale 1ns / 1ps

module moore_machine (
   input  wire clk,
   input  wire reset,
   input  wire x,
   output reg  z
);

   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;
   parameter S3 = 2'b11;

   reg [1:0] current_state, next_state;

   always @(posedge clk or posedge reset) begin
      if (reset)
         current_state <= S0;
      else
         current_state <= next_state;
   end


   always @(*) begin
      case (current_state)

         S0: begin
            if (x) next_state = S1;
            else   next_state = S0;
         end

         S1: begin
            if (!x) next_state = S2;
            else    next_state = S1;
         end

         S2: begin
            if (x) next_state = S3;
            else   next_state = S0;
         end

         S3: begin
            if (x) next_state = S1;
            else   next_state = S0;
         end

         default: next_state = S0;
```

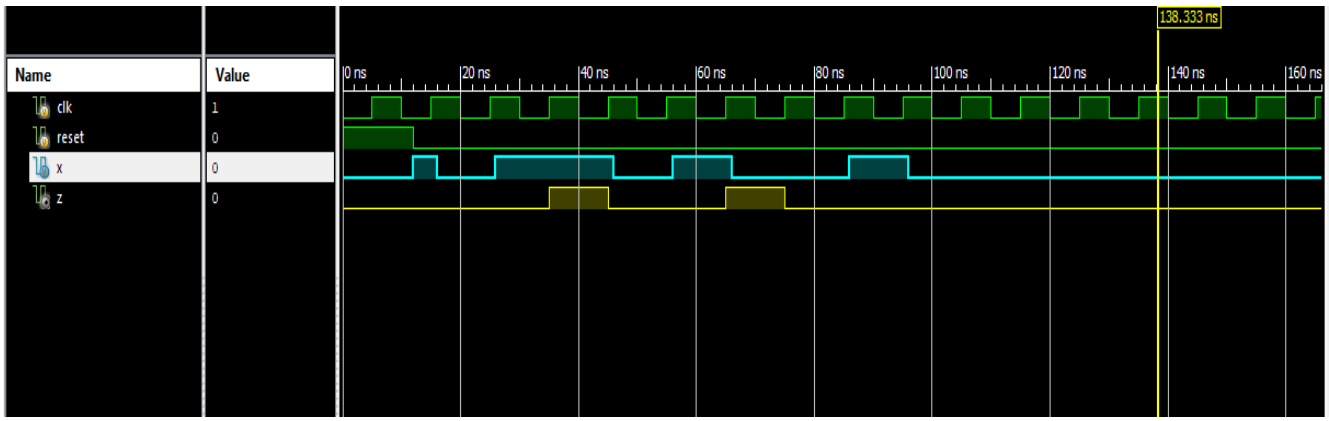
```
        endcase
    end

    always @(*) begin
        if (current_state == S3)
            z = 1;
        else
            z = 0;
    end

endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module moore_machine_tb;

    reg clk;
    reg reset;
    reg x;
    wire z;

   moore_machine DUT (
        .clk(clk),
        .reset(reset),
        .x(x),
        .z(z)
    );

    always #5 clk = ~clk;

    reg [15:0] input_stream;

    integer i;

    initial begin
        clk = 0;
        reset = 1;
        x = 0;

        #15;
        reset = 0;

        input_stream = 16'b1011010010000000;

        for (i = 15; i >= 0; i = i -1)
                        begin
            x = input_stream[i];
            @(posedge clk);
            #1;
```

```
        end
        $finish;
    end
endmodule
```

OUTPUT



Example#2: Design a Mealy (FSM) circuit to detect a sequence (101).

a. State Diagram: Onboard.

b. State Transition Table

Present State	Input x	Next State	Output z
S0	0	S0	0
S0	1	S1	0
S1	0	S2	0
S1	1	S1	0
S2	0	S0	0
S2	1	S1	1

CODE:

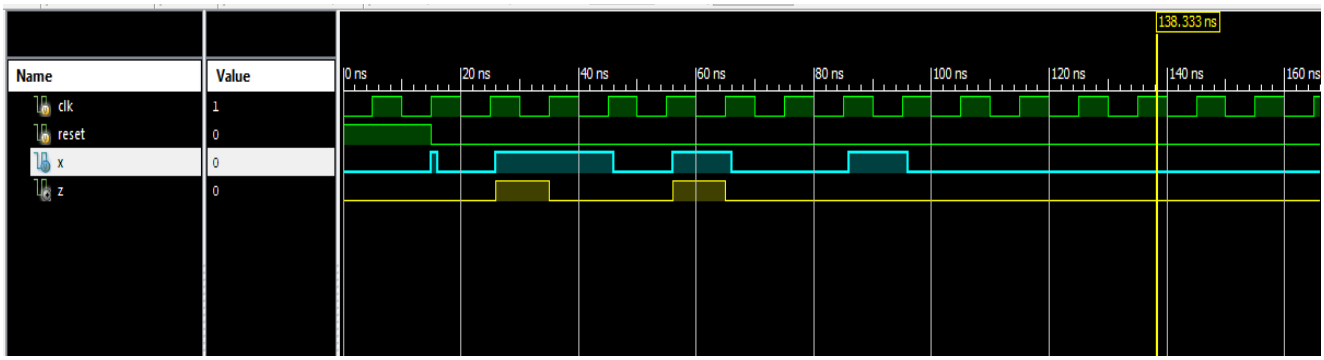
```
`timescale 1ns / 1ps

module mealy_machine(
    input  wire clk,
    input  wire reset,
    input  wire x,
    output reg  z
);

    parameter S0 = 2'b00;
    parameter S1 = 2'b01;
    parameter S2 = 2'b10;

    reg [1:0] current_state, next_state;

    always @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    always @(*) begin
        // Default output
        z = 0;
```

```
        case (current_state)
            S0: begin
                if (x)
                    next_state = S1;
                else
                    next_state = S0;
            end

            S1: begin
                if (x)
                    next_state = S1;
                else
                    next_state = S2;
            end

            S2: begin
                if (x) begin
                    next_state = S1;
                    z = 1;   // sequence detected
                end
                else
                    next_state = S0;
            end

            default: next_state = S0;
        endcase
    end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module mealy_machine_tb;

    reg clk;
    reg reset;
    reg x;
    wire z;


    mealy_machine DUT (
        .clk(clk),
        .reset(reset),
        .x(x),
        .z(z));

    always #5 clk = ~clk;

    reg [15:0] input_stream;
    integer i;

    initial begin

        clk = 0;
        reset = 1;
        x = 0;

        #15;
        reset = 0;

        input_stream = 16'b1011010010000000;

        for (i = 15; i >= 0; i = i - 1) begin
            x = input_stream[i];
            @(posedge clk);
            #1;

        end

       $finish;
    end

endmodule
```

OUTPUT



Home Tasks:

In this lab activity, you are required to design and implement **Finite State Machines (FSMs)** in Verilog using both **Moore** and **Mealy** models. The objective is to detect a specific binary sequence **X**, where **X corresponds to the last non-zero digit of your registration number represented in binary (e.g., $7_{10} = 0111_2$).** The complete **binary representation of your registration number** will serve as the input signal to the FSMs.

NOTE: Input signals are your registration number. for example, Reg # **2024123** (00100000001001000001001000011)

You must perform the following tasks:

1. Identify the sequence X based on the last non-zero digit of your registration number.
2. Construct a state diagram and develop a state transition table for both Moore and Mealy FSMs.
3. Write Verilog code for Moore and Mealy FSMs that detect the selected sequence X.
4. Develop appropriate test benches to verify the functionality of each FSM model.
5. Simulate the designs and visualize their behavior using waveform outputs.
6. Analyze and compare the operational differences between Moore and Mealy implementations.

CE221L-Digital Logic Design Lab

RUBRIC SHEET

Criteria	Excellent (Full Marks)	Good (Partial Marks)	Needs Improvement (Low Marks)	Max Marks
Code Writing (Verilog) **(Apply)** (4)	Writes fully optimized and error-free Verilog code for combinational and sequential circuits independently.	Writes mostly correct code with minor errors, requiring limited guidance.	The code contains major errors or is incomplete, requiring significant help to correct.	**/4**
Test Bench **(Construct)** (3)	Develops accurate and complete test benches to simulate and verify circuit functionality without errors.	Creates test benches with minor issues that require small corrections.	Fails to create a proper test bench or is unable to verify functionality.	**/3**
FPGA Implementation **(Construct & Apply)** (3)	Successfully implements the design on the FPGA trainer board, demonstrating correct and reliable hardware performance.	Implement the design with minor issues that require troubleshooting.	Unable to correctly implement the design on the FPGA, or the hardware fails.	**/3**
Design Objectives **(Apply & Construct**) (3)	Clearly states and applies theoretical concepts to fully achieve all given design objectives.	States and applies concepts partially, achieving some design objectives.	Fails to state or apply theoretical concepts, achieving none or minimal objectives.	**/3**
Viva **(Collaborate)** (2)	Communicates ideas clearly and confidently, demonstrating full understanding of concepts and the design process.	Communicates with some hesitation or minor conceptual gaps.	Struggles to explain concepts or answer questions correctly.	**/2**
Total				**/15**

Name: _____ **Instructor Signature:** _____

Reg #: _____ **Date:** _____