**Finite State Machines with Mealy and Moore**

## **Introduction to FSMs**

**Finite State Machines (FSMs)** are a type of sequential circuit. Unlike combinational circuits, FSM outputs depend not only on current inputs but also on previous inputs stored in memory (states).

FSMs operate under a clock and consist of:

* Flip-flops to store states
* Combinational logic to determine next states and outputs

### **Types of FSMs:**

1. **Moore FSM**
   * Output depends only on the current state
   * More stable but may require more states
2. **Mealy FSM**
   * Output depends on the current state **and** input
   * Faster response with fewer states

## **Mealy FSM Design**

### **Problem Statement:**

Design an FSM that sets output z = 1 only if the last **two consecutive inputs** w were 1. All transitions occur on the **positive edge** of the clock.

### **State Diagram (from Lab PDF):**

| **Present State** | **Input w** | **Next State** | **Output z** |
| --- | --- | --- | --- |
| A | 0 | A | 0 |
| A | 1 | B | 0 |
| B | 0 | A | 0 |
| B | 1 | B | 1 |

## **Verilog Implementation (Mealy FSM)**

module Mealy11(clk, reset, w, z, y, Y);

input clk, reset, w;

output z;

output reg y, Y;

parameter A = 0, B = 1;

always @(posedge clk, negedge reset)

begin

if (reset == 0)

y = A;

else begin

y = Y;

casex(y)

A: if (w == 1) Y = B;

else Y = A;

B: if (w == 1) Y = B;

else Y = A;

endcase

end

end

assign z = ((y == B) & (w == 1));

endmodule

### **Explanation:**

* **States**: A (initial), B (after one w=1)
* **Transitions**: Move to B on w=1, return to A on w=0
* **Output**: z=1 only when in state B **and** input w=1

This matches exactly the Mealy FSM in the PDF (pages 13–15).

## **Moore FSM Design (As per Lab PDF)**

### **Problem Statement:**

Design an FSM where the output z = 1 only if the **last two inputs** were both 1. Unlike the Mealy machine, here the output depends **only on the current state**.

### **State Diagram (from Lab PDF):**

| Present State | Input w | Next State | Output z |
| --- | --- | --- | --- |
| A | 0 | A | 0 |
| A | 1 | B | 0 |
| B | 0 | A | 0 |
| B | 1 | C | 0 |
| C | 0 | A | 1 |
| C | 1 | C | 1 |

### **Verilog Code:**

module moore11(clk, reset, w, z);

input clk, reset, w;

output reg z;

reg [1:0] state, next\_state;

parameter A = 2'b00, B = 2'b01, C = 2'b10;

// State transition

always @(posedge clk or posedge reset) begin

if (reset)

state <= A;

else

state <= next\_state;

end

// Next state logic

always @(\*) begin

case (state)

A: next\_state = (w) ? B : A;

B: next\_state = (w) ? C : A;

C: next\_state = (w) ? C : A;

default: next\_state = A;

endcase

end

// Output logic

always @(\*) begin

case (state)

A, B: z = 0;

C: z = 1;

endcase

end

endmodule

### **Explanation:**

* **States**: A (initial), B (after one w=1), C (after two w=1s)
* **Output**: z = 1 only in state C, **independent of w**
* **More stable output** than Mealy but slower response

## **How It Works**

### **Mealy FSM Example Simulation:**

| Clock Cycle | w | State (y) | Output z |
| --- | --- | --- | --- |
| 1 | 1 | A → B | 0 |
| 2 | 1 | B | 1 |
| 3 | 0 | A | 0 |

### **Moore FSM Example Simulation:**

| Clock Cycle | w | State | Output z |
| --- | --- | --- | --- |
| 1 | 1 | B | 0 |
| 2 | 1 | C | 1 |
| 3 | 0 | A | 0 |

## **Comparison of Mealy and Moore FSMs**

| Feature | Moore FSM | Mealy FSM (Your Code) |
| --- | --- | --- |
| Output depends | State only | State + Input |
| States used | More | Fewer |
| Response | Delayed by clock | Faster (Immediate) |
| Stability | More stable | Can glitch if not timed |

## **Extendability**

You can extend both FSM templates to detect longer input patterns (e.g., "1001") by:

* Adding more states
* Modifying casex or case logic
* Adjusting output logic appropriately