**Moore Finite State Machine**

## **✅ Problem Statement**

We need to design a **Moore Finite State Machine (FSM)** that satisfies the following:

* It has one input: w
* One output: z
* The output z should be **1** if the **last two consecutive inputs** were both 1
* All transitions happen on the **positive edge of the clock**
* If the pattern is not satisfied, z remains 0

## **🧠 Why a Moore FSM?**

In a **Moore FSM**, the output depends **only on the current state**, not directly on the inputs. This means:

* The output becomes valid **after** a full clock cycle (i.e., after reaching the appropriate state)
* The FSM may need **more states** than a Mealy FSM, but it provides more stable outputs

## **🔄 State Transition Process**

To detect **two consecutive 1s**, we need to keep track of past input history. We define three states:

### **📊 States**

| **State** | **Meaning** | **Output z** |
| --- | --- | --- |
| A | Initial state or last w was 0 | 0 |
| B | Last w was 1 (waiting for second 1) | 0 |
| C | Two consecutive 1s seen (pattern hit) | 1 |

## **🔁 State Diagram**

+---------+ w=1 +---------+ w=1 +---------+

| A | ----------> | B | ----------> | C |

| z = 0 | | z = 0 | | z = 1 |

+---------+ +---------+ +---------+

^ | ^ | |

| | w=0 | | w=0 | w=0

| +--------------------+ +---------------------+

|

+---------------------------------------------+

## **⚙️ How the Problem is Solved in Verilog**

### **👇 State Encoding**

parameter [1:0] A = 2'b00, B = 2'b01, C = 2'b10;

Each state is encoded using 2 bits.

### **⏱ Clocked Transition Logic**

always @(posedge clk or posedge reset)

This block is triggered on:

* The **positive edge** of the clock (normal operation)
* The **positive edge** of reset (to initialize the FSM)

### **🔁 Transition Logic**

casex(y)

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

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

C: if (w == 1) Y = C; else Y = A;

This defines how the FSM moves between states based on input w.

### **🧾 Output Logic**

assign z = (y == C);

This line defines the output z. Since it's a Moore machine, output depends **only** on the current state y.

## **📋 Step-by-Step Example**

### **Input Sequence:**

w = 0 1 1 0 1 1 0

### **State & Output:**

| **Cycle** | **Input (w)** | **State (y)** | **Output (z)** |
| --- | --- | --- | --- |
| 1 | 0 | A | 0 |
| 2 | 1 | B | 0 |
| 3 | 1 | C | 1 ✅ |
| 4 | 0 | A | 0 |
| 5 | 1 | B | 0 |
| 6 | 1 | C | 1 ✅ |
| 7 | 0 | A | 0 |

## **✅ Summary**

| **Aspect** | **Description** |
| --- | --- |
| FSM Type | Moore |
| Input | w |
| Output | z = 1 after two 1s in a row |
| States | A, B, C |
| Output logic | Based only on state (Moore property) |
| Implementation | Uses always and casex blocks |
| Reset behavior | Synchronous reset to state A |
| Use case | Pattern detection, controller logic, filters |