Introduction

A state machine represents a system as a set of states, the transitions between them, along with the associated inputs and outputs. So, a state machine is a particular conceptualization of a particular sequential circuit. State machines can be used for many other things beyond logic design and computer architecture.

Finite State Machines (FSM)

- Any Circuit with Memory is a Finite State Machine, Even computers can be viewed as huge FSMs
- Design of FSMs Involves
  - Defining states
  - Defining transitions between states
  - Optimization / minimization
- Above Approach is Practical for Small FSMs Only
• A finite state machine is a sequential logic circuit which moves between a finite set of states, dependent upon the values of the inputs and the previous state. The state transitions are synchronized on a clock.

• There are many ways to describe a finite state machine in VHDL. The most convenient is with a process statement. The state of the machine can be stored in a variable or signal, and the possible states conveniently represented with an enumeration type.

❖ Moore FSM

Output is a Function of Present State Only.
Outputs are independent of the inputs, i.e. outputs are effectively produced from within the state of the state machine.
**Mealy FSM**

**Output is a Function of a Present State and Inputs**

Outputs can be determined by the present state alone, or by the present state and the present inputs, i.e. outputs are produced as the machine makes a transition from one state to another.
Moore vs. Mealy FSM

- Moore and Mealy FSMs Can Be Functionally Equivalent, So Equivalent Mealy FSM can be derived from Moore FSM and vice versa.

- Mealy FSM Has Richer Description and Usually Requires Smaller Number of States.

- Mealy FSM Computes Outputs as soon as Inputs Change, So Mealy FSM responds one clock cycle sooner than equivalent Moore FSM.

- Moore FSM Has No Combinational Path Between Inputs and Outputs, Moore FSM is more likely to have a shorter critical path.

**State memory** :

Considered the engine that moves the state machine and used to implement reset as well:

```vhdl
architecture fsm of fsm is
  type State_Type is (s0,s1,s2,s3,s4);
  signal Current_State,Next_State: State_Type ;
begin

  STATE_MEMORY : process (CLK)
  begin
  --much as a Dff that transfers value to output every clock
  if (CLK'event and CLK='1') then
    Current_State<= Next_State;
  end if;
  end process;

end fsm;
```
How to add sync and async reset?

**Example:**
Design a machine by hand that takes in a serial bit stream and looks for the pattern “1011”. When the pattern is found, a signal called “Found” is asserted - use sync. Reset

Same steps every time:

1. Decide whether you need Mealy or Moore? Or both works?
2. Draw the flow chart “carefully” for your state machine

**Mealy:**

![Mealy State Machine Diagram](image)

**Moore:**

![Moore State Machine Diagram](image)
3- Add your state memory and notice rest if exists

architecture fsm of fsm is

    type State_Type is (s0,s1,s2,s3,s4);
    signal Current_State,Next_State: State_Type ;
begin

    STATE_MEMORY : process (CLK)
    begin
        --much as a Dff that transfers value to output every clock

        if (CLK'event and CLK='1') then
            if(rst='1') then
                Current_State<= s0;
            else
                Current_State<= Next_State;
            end if;
        end if;
    end process;
end fsm;

4- Add a process for Next state logic (assuming mealy)

NEXT_STATE_LOGIC : process (inp, Current_State)
begin
    case (Current_State) is
        when S0 =>
            if (inp='0') then Next_State<= S0;
            elsif(inp='1') then Next_State<= S1;
            end if;
        when S1 =>
            if (inp='0') then Next_State<= S2;
            elsif(inp='1') then Next_State<= S1;
            end if;
        when S2 =>
            if (inp='0') then Next_State<= S0;
            elsif(inp='1') then Next_State<= S3;
            end if;
        when S3 =>
            if (inp='0') then Next_State<= S2;
            elsif(inp='1') then Next_State<= S1;
            end if;
    end case;
end process;
5- Add a process for output logic (assuming mealy)

```
OUTPUT_LOGIC : process (inp, Current_State)
begin
    case (Current_State) is
        when S0 =>
            if (inp='0') then Found <= '0';
            elsif(inp='1') then Found <= '0';
            end if;

        when S1 => if (inp='0') then Found <= '0';
            elsif(inp='1') then Found <= '0';
            end if;

        when S2 =>
            if (inp='0') then Found <= '0';
            elsif(inp='1') then Found <= '0';
            end if;

        when S3 =>
            if (inp='0') then Found <= '0';
            elsif(inp='1') then Found <= '1';
            end if;
    end case;
end process;
```

<<EX>>
1- change step 4 and 5 for Moore state machine

2- Let’s design a 2-bit Up/Down Gray Code Counter using User-Enumerated State Encoding

   -In=0, Count Up
   -In=1, Count Down
   -this will be a Moore or mealy ? which is easier?
   -no Reset

3- Implement a sequence detector for “1001” and test your work