Module #13 : Verilog HDL Finite State Machines (FSMs)

13.1: Finite State Machines (FSMs):

- In Digital VLSI design, Finite State Machines play a very important role in implementing the correct behaviour of the system during different operating modes.
- The FSM enables the system to go through different operating modes as per the user requirements or during the self booting process

Examples :

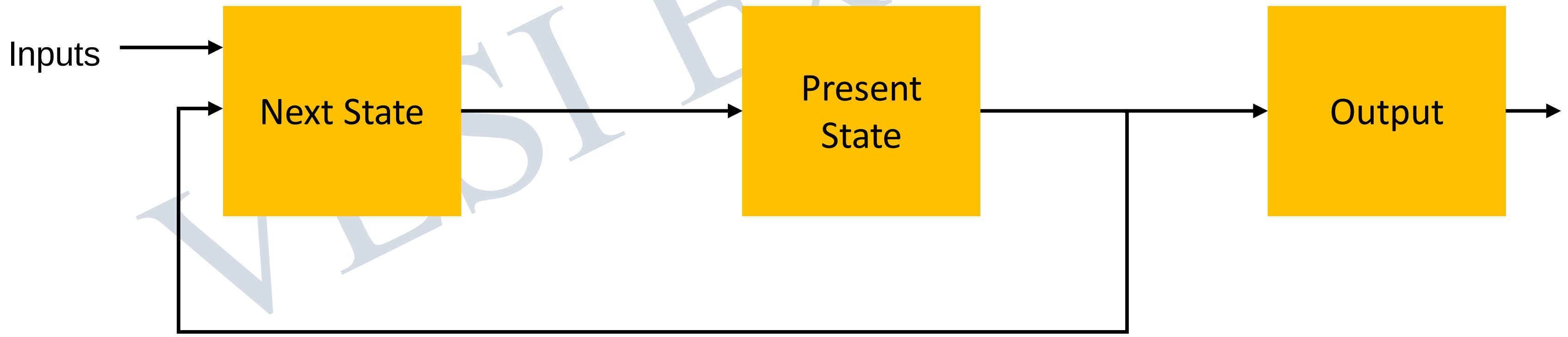
1) Implementing different Low Power Modes in a SoC using State Machines
2) Transmitter and Receiver Logic Implementation (UART, SPI etc)

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.2: Moore and Mealy FSMs:

- In Moore FSM, output depends only on the present state
- Output is less prone to glitch because of registered present state
- More number of states are required
- They react slower to inputs (One clock cycle later)

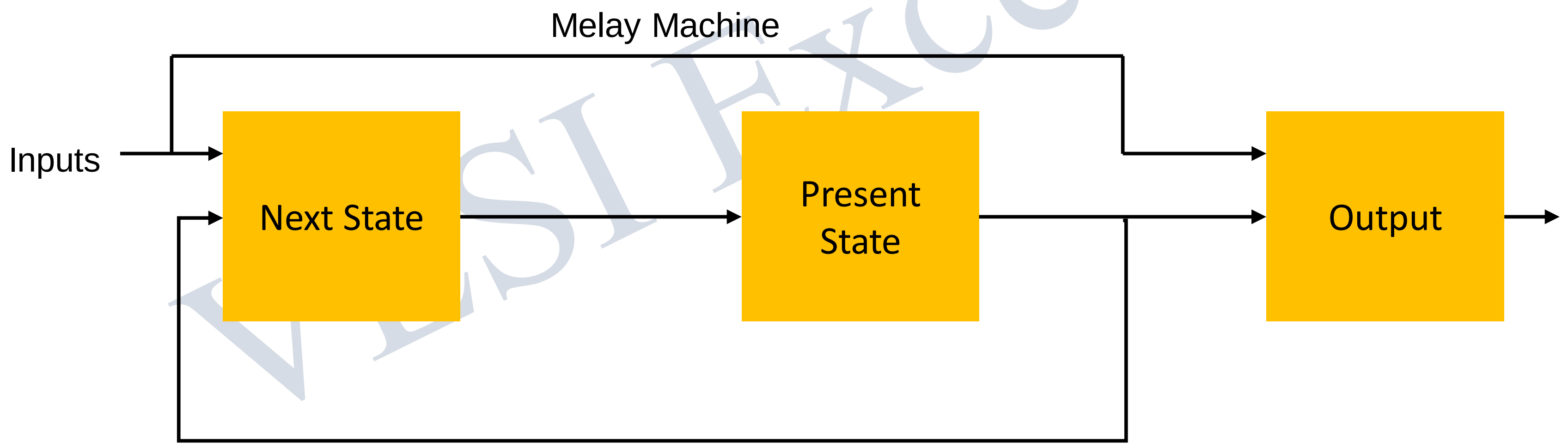
Moore Machine



Module #13 : Verilog HDL Finite State Machines (FSMs)

13.3: Moore and Mealy FSMs:

- In Mealy FSM, output depends on the present state as well as present input
- Output is more prone to glitch because of asynchronous dependency on present input
- Less number of states are required
- They react faster to inputs



Module #13 : Verilog HDL Finite State Machines (FSMs)

13.4: FSM Design Techniques :

1) Using a Single Process (Procedural Block) to Code Present State, Next State and Output Logic
2) Using Two Process, One to code Present State and Next State logic and another to code Output Logic,
3) Using Three Process each to code Present State, Next State and Output Logic

Note: When modeling finite state machines, it is recommended to separate the sequential current-state logic from the combinational next-state and output logic.

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.5: Defining State Definition :

1) Using Parameters:

```
parameter state0 = 0, state1 = 1, state2 = 2, state3 = 3;
```

2) Using Macros:

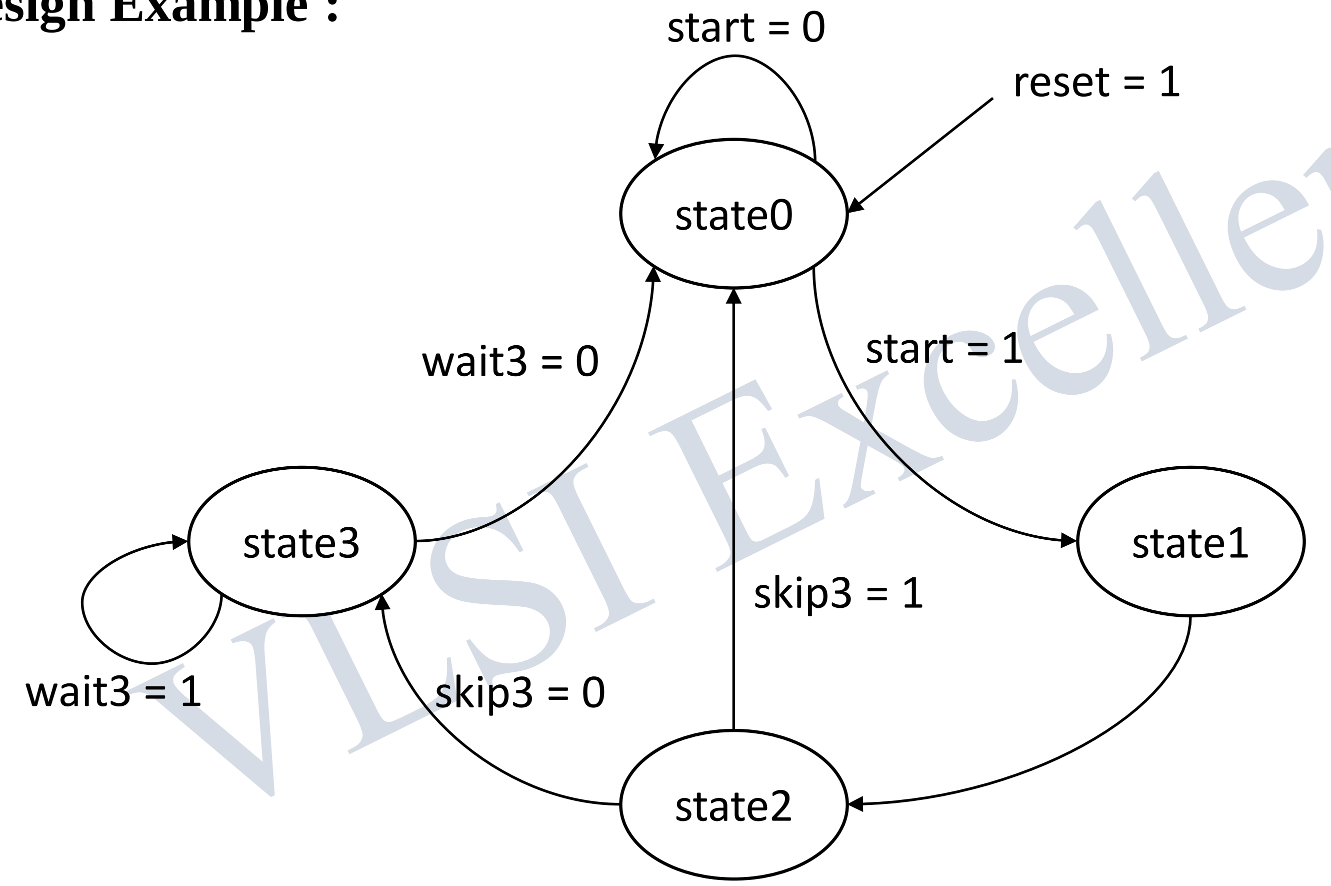
```
`define state0 2'd0
`define state1 2'd1
`define state2 2’d2
`define state3 2'd3;
```

When using macro definitions one must put a back quote in front. For example:

```
case (state)
    `state0: Z = 3’b000;
    `state1: Z = 3’b101;
    `state2: Z = 3’b111;
    `state3: Z = 3’b001;
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.5: FSM Design Example :



state0: Z = 3’b000;
state1: Z = 3’b101;
state2: Z = 3’b111;
state3: Z = 3’b001;

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.5: FSM Design Example :

Example 13.1:

```
module my_fsm (clk, rst, start, skip3, wait3, Z);
input clk, rst, start, skip3, wait3;
output [2:0] Z; // Z is declared reg so that it can
reg [2:0] Z;   // be assigned in an always block.
parameter state0=0, state1=1, state2=2, state3=3;
reg [1:0] state, nxt_st;
always @ (state or start or skip3 or wait3)
begin : next_state_logic //Name of always procedure.
case (state)
state0: begin
if (start) nxt_st = state1;
else nxt_st = state0;
end
state1: begin
nxt_st = state2;
end
state2: begin
if (skip3) nxt_st = state0;
else nxt_st = state3;
end
state3: begin
if (wait3) nxt_st = state3;
else nxt_st = state0;
end
default: nxt_st = state0;
endcase // default is optional since all 4 cases are
end      // covered specifically. Good practice
         // says uses it
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.5: FSM Design Example :

```
always @(posedge clk or posedge rst)
begin : register_generation
if (rst) state = state0;
else state = nxt_st;
end
always @(state) begin : output_logic
case (state)
state0: Z = 3'b000;
state1: Z = 3'b101;
state2: Z = 3'b111;
state3: Z = 3'b001;
default: Z = 3'b000;  // default avoids latches
endcase
end
endmodule
```