

CSE-308: Digital System Design

Lecture 7

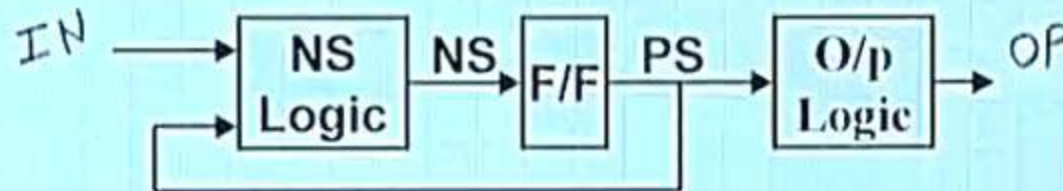
VERILOG – Part V

Modeling Finite State Machines

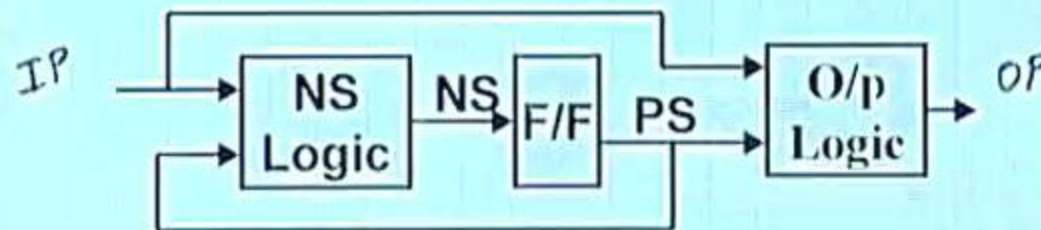
©CET
I.I.T. KGP

- Two types of FSMs

- Moore Machine



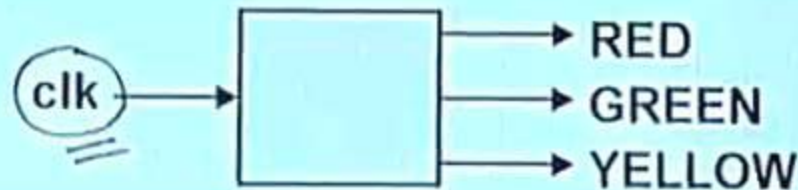
- Mealy Machine



Moore Machine : Example 1

© CET
I.I.T. KGP

- Traffic Light Controller
 - Simplifying assumptions made
 - Three lights only (RED, GREEN, YELLOW)
 - The lights glow cyclically at a fixed rate
 - Say, 10 seconds each
 - The circuit will be driven by a clock of appropriate frequency




```

module traffic_light (clk, light);
    input clk;
    output [0:2] light;    reg [0:2] light;
    parameter S0=0, S1=1, S2=2;
    parameter RED=3'b100, GREEN=3'b010,
               YELLOW=3'b001;

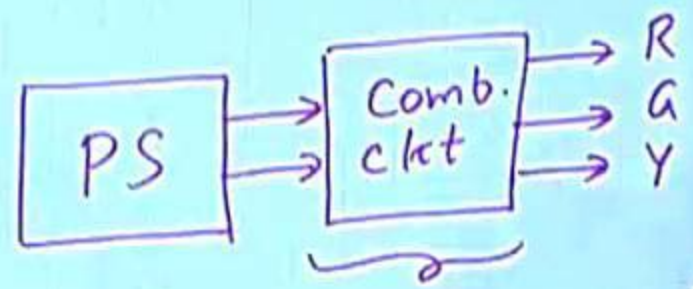
    reg [0:1] state;
    always @(posedge clk)
        case (state)
            S0: begin                                // S0 means RED
                    light <= YELLOW;
                    state <= S1;
                end
        end
    
```

S0 → RED
S1 → YELLOW
S2 → GREEN

==

```
S1: begin                                // S1 means YELLOW
    light <= GREEN;
    state <= S2;
end
S2: begin                                // S2 means GREEN
    light <= RED;
    state <= S0;
end
default: begin
    light <= RED;
    state <= S0;
end
endcase
endmodule
```

- **Comment on the solution**
 - Five flip-flops are synthesized
 - Two for 'state' //
 - Three for 'light' (outputs are also latched into flip-flops)
 - If we want non-latched outputs, we have to modify the Verilog code.
 - Assignment to 'light' made in a separate 'always' block.
 - Use blocking assignment.



module traffic_light_nonlatched_op (clk, light);

input clk;

output [0:2] light; reg [0:2] light;

parameter S0=0, S1=1, S2=2;

parameter RED=3'b100, GREEN=3'b010,
YELLOW=3'b001;

reg [0:1] state;

always @(posedge clk)

case (state)

S0: state <= S1;

S1: state <= S2;

S2: state <= S0;

default: state <= S0;

endcase

⇒ 2 f/f's
synthesized
for storing
the state.

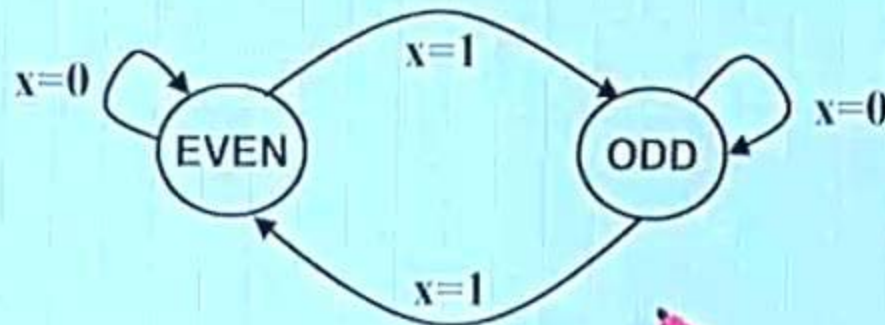
```
always @ (state)
  case (state)
    S0:    light = RED;
    S1:    light = YELLOW;
    S2:    light = GREEN;
    default: light = RED;
  endcase
endmodule
```

*Output generation
logic synthesized
as comb. ckt.*

Moore Machine: Example 2

© CET
L.T.KGP

- Serial parity detector




```
module parity_gen (x, clk, z);  
  input x, clk;  
  output z;    reg z;  
  reg even_odd; // The machine state  
  parameter EVEN=0, ODD=1;  
  
  always @(posedge clk)  
    case (even_odd)  
      EVEN: begin  
        z <= x ? 1 : 0;  
        even_odd <= x ? ODD : EVEN;  
      end
```

```
    ODD: begin
        z <= x ? 0 : 1;
        even_odd <= x ? EVEN : ODD;
    end
endcase
endmodule
```

- If no output latches need to be synthesized, we can follow the principle shown in the last example.

always @ (posedge clock)

[next state

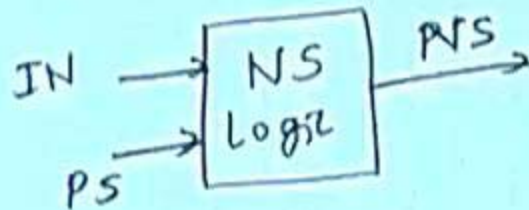
always @ (even-odd)

[output

Serial Parity Detector

© CET
L.T. KGP

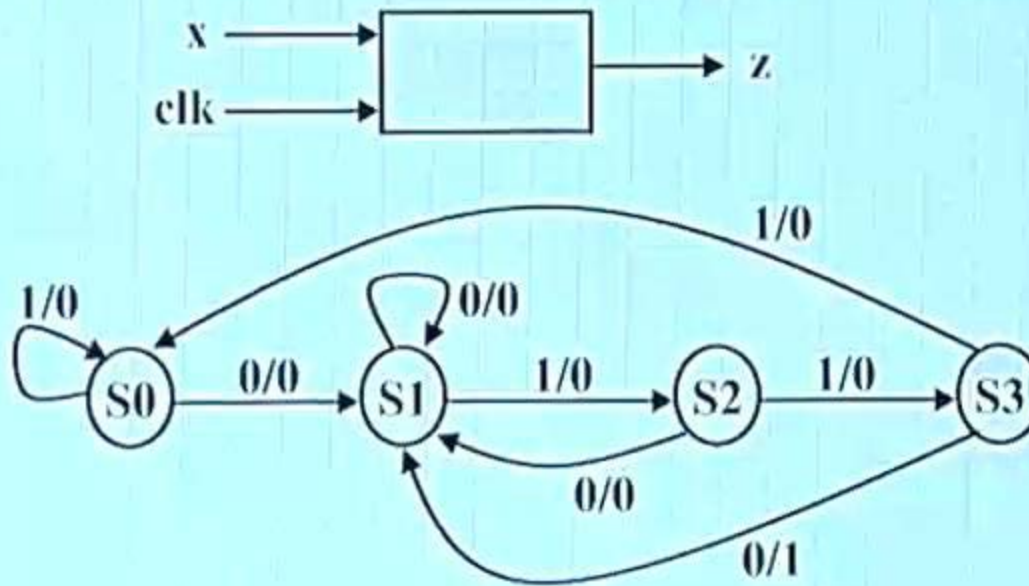
..10010110 : x
01110010 : odd-even



Mealy Machine: Example

©CET
L.T.KGP

- Sequence detector for the pattern '0110'.

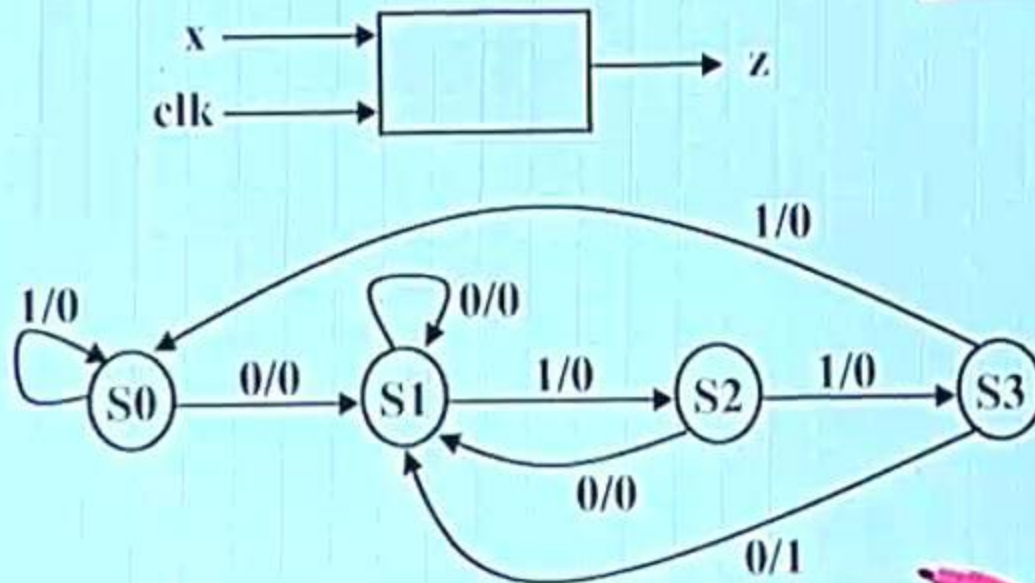


X: 0011001101100
Z: 0000100010010

Mealy Machine: Example

© CET
I.I.T. KGP

- Sequence detector for the pattern '0110'.



// Sequence detector for the pattern '0110'

```
module seq_detector (x, clk, z)
  input x, clk;
  output z;      reg z;
  parameter S0=0, S1=1, S2=2, S3=3;
  reg [0:1] PS, NS;

  always @(posedge clk)
    PS <= NS;
```


always @ (PS or x)

case (PS)

S0: begin

z = x ? 0 : 0;

NS = x ? S0 : S1;

end;

S1: begin

z = x ? 0 : 0;

NS = x ? S2 : S1;

end;

S2: begin

z = x ? 0 : 0;

NS = x ? S3 : S1;

end;

S3: begin

z = x ? 0 : 1;

NS = x ? S0 : S1;

end;

endcase

endmodule

© CSE
I.I.T. KGP

