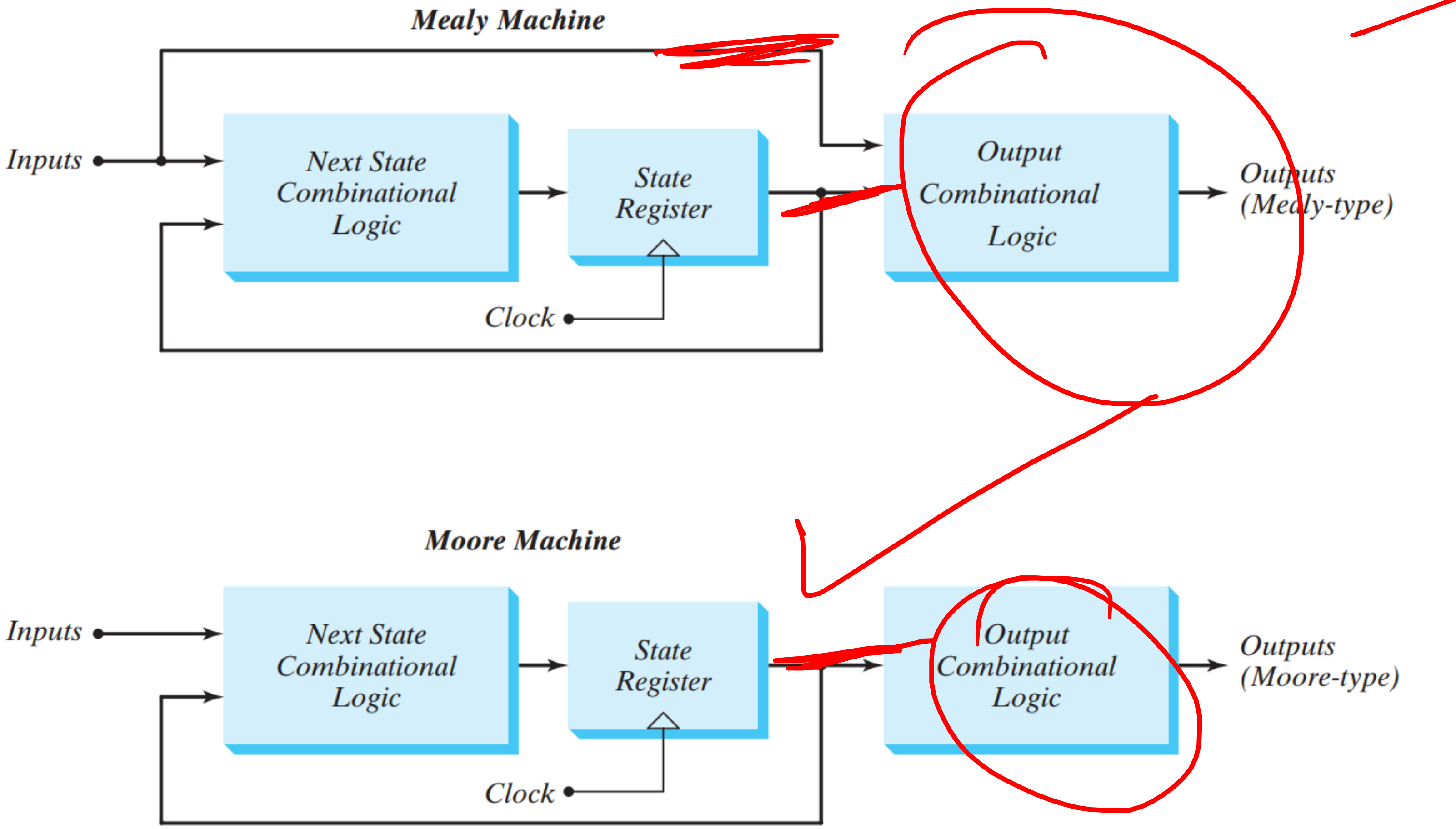
ELD Lab 4
Design of Sequence Detector

Objective

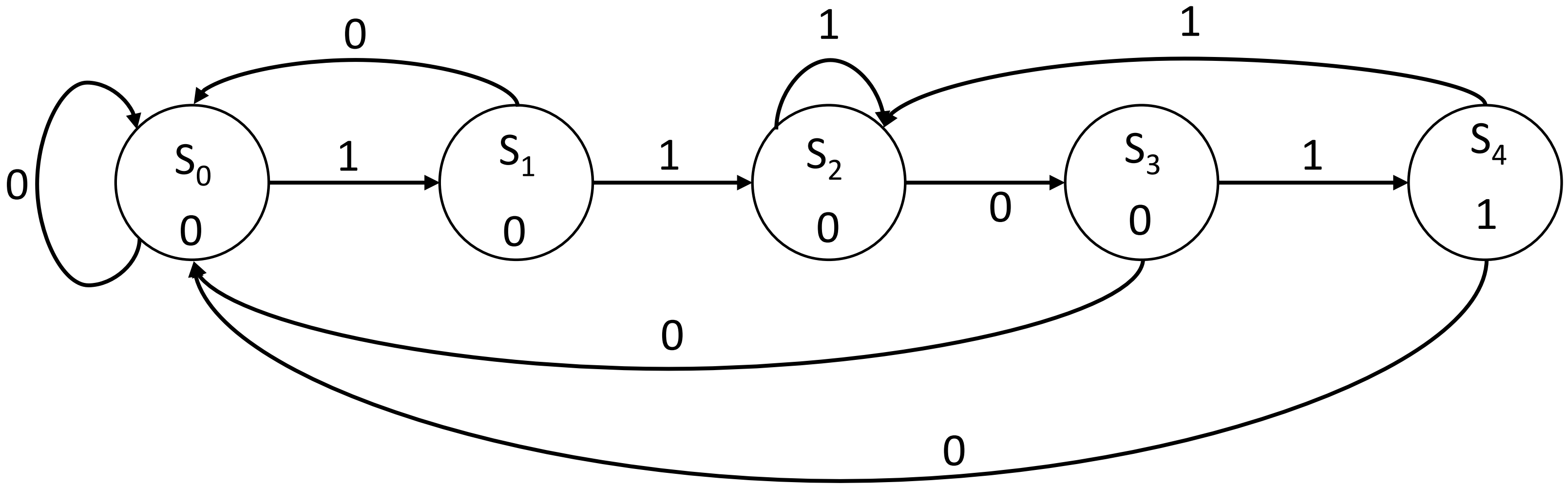
- Design 1011 sequence detector using behavioral modelling
- Verify the circuit using virtual input and output (VIO).
- **Lab Homework:** Change the sequence to 1111

Finite State Machines

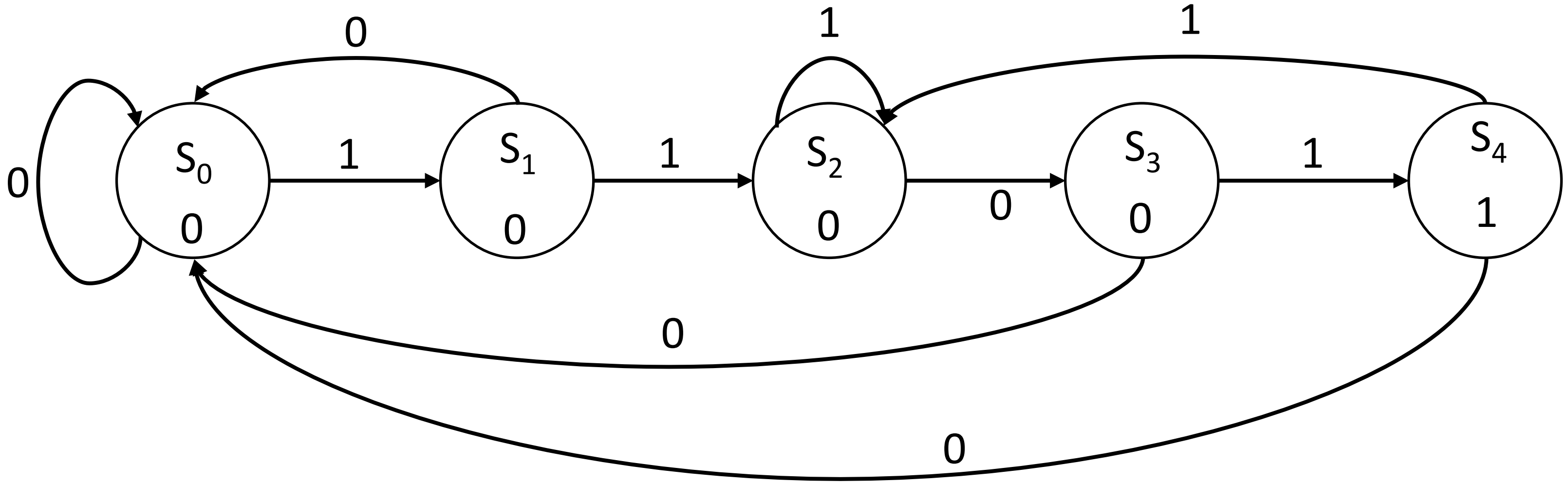
1101



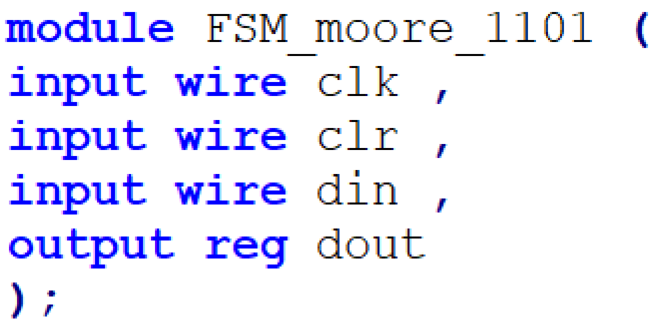
FSM: Sequence Detector 1101 (Moore)



FSM: Sequence Detector 1101 (Moore)

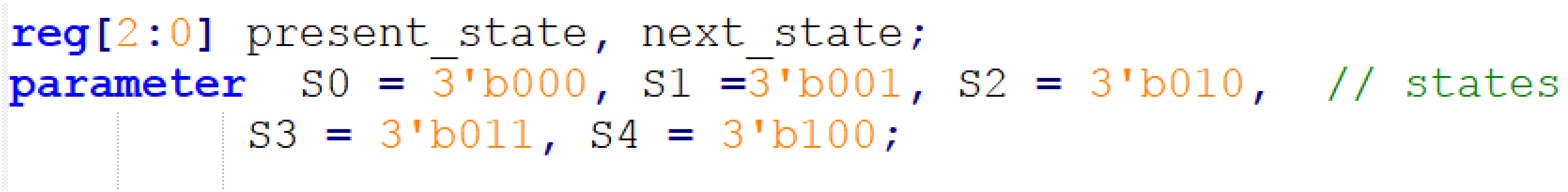


1



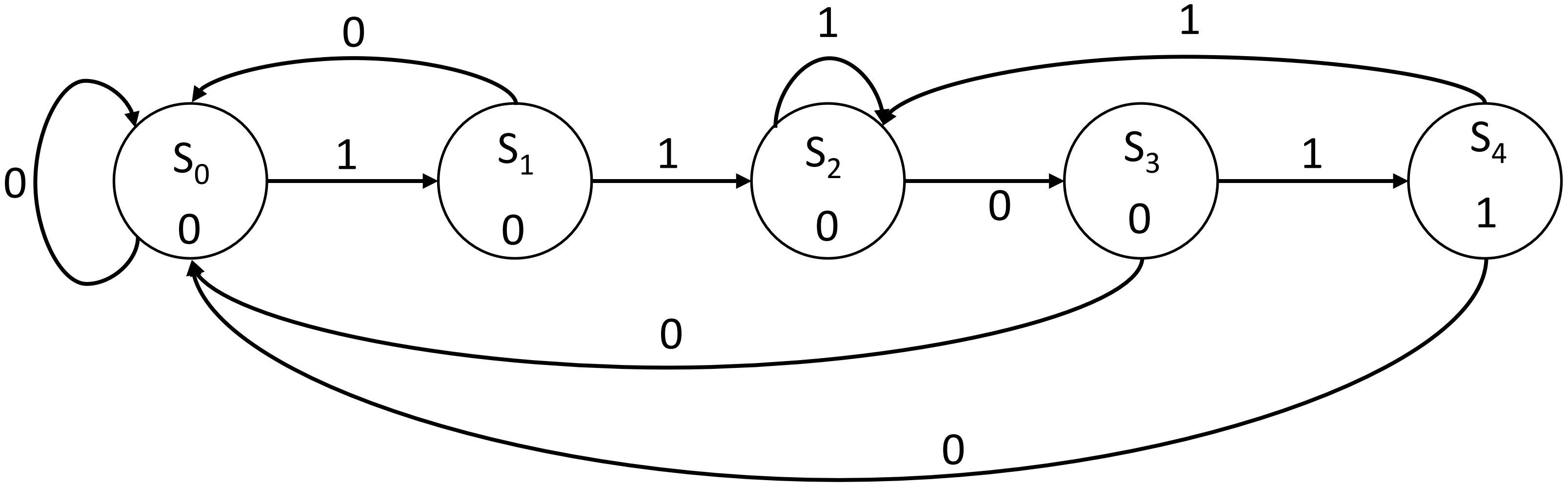
```
module FSM_moore_1101 (
input wire clk ,
input wire clr ,
input wire din ,
output reg dout
);
```

2

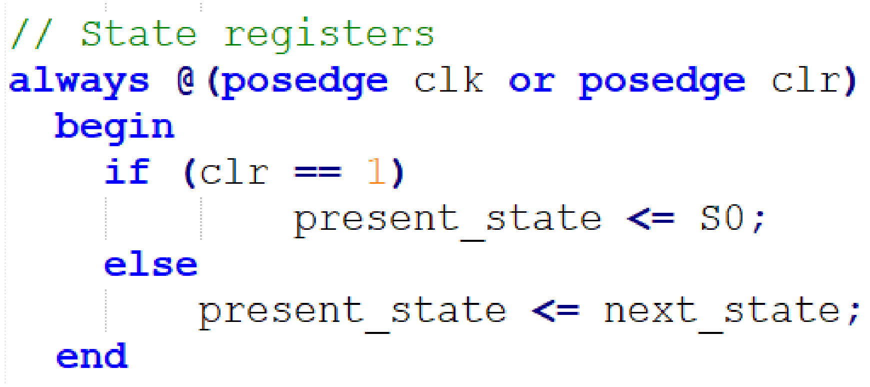


```
reg[2:0] present_state, next_state;
parameter  S0 = 3'b000, S1 =3'b001, S2 = 3'b010,  // states
          S3 = 3'b011, S4 = 3'b100;
```

FSM: Sequence Detector 1101 (Moore)

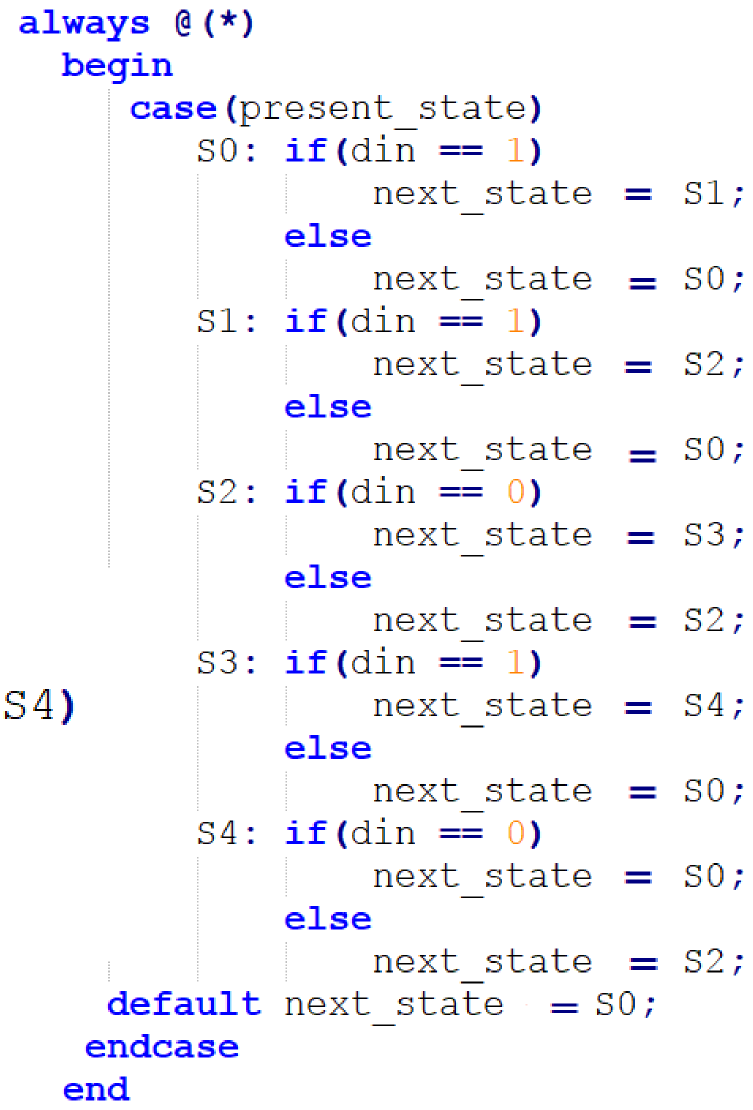


3



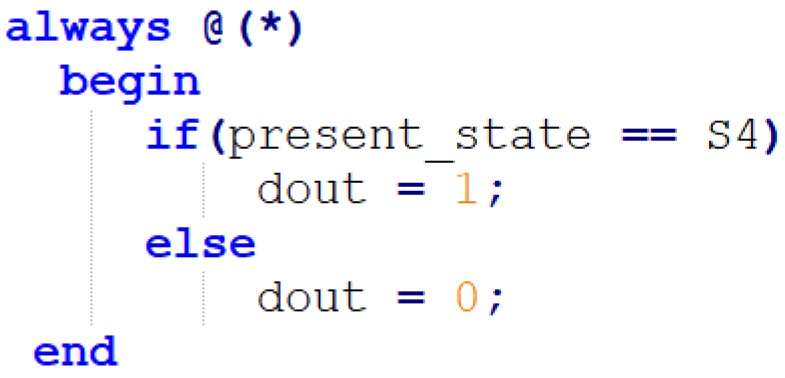
```
// State registers
always @(posedge clk or posedge clr)
  begin
    if (clr == 1)
          present_state <= S0;
    else
        present_state <= next_state;
  end
```

4



```
always @(*)
  begin
   case(present_state)
      S0: if(din == 1)
              next_state  = S1;
          else
              next_state  = S0;
      S1: if(din == 1)
              next_state  = S2;
          else
              next_state  = S0;
      S2: if(din == 0)
              next_state  = S3;
          else
              next_state  = S2;
      S3: if(din == 1)
              next_state  = S4;
          else
              next_state  = S0;
      S4: if(din == 0)
              next_state  = S0;
          else
              next_state  = S2;
    default next_state  = S0;
   endcase
  end
```

5

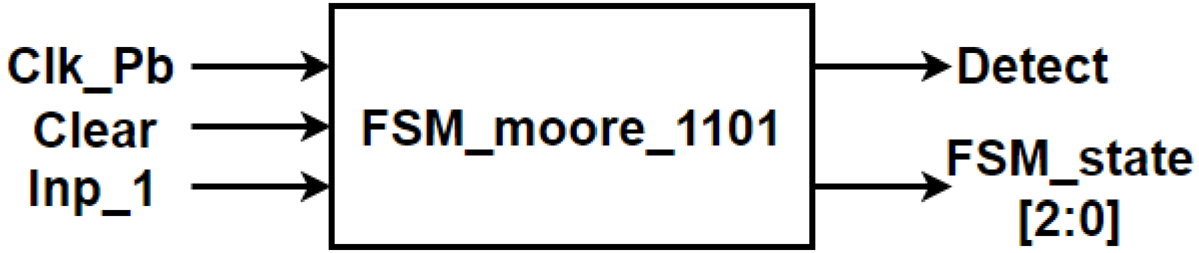


```
always @(*)
  begin
    if(present_state == S4)
        dout = 1;
    else
        dout = 0;
  end
```

Lab

Proposed Approach

FSM



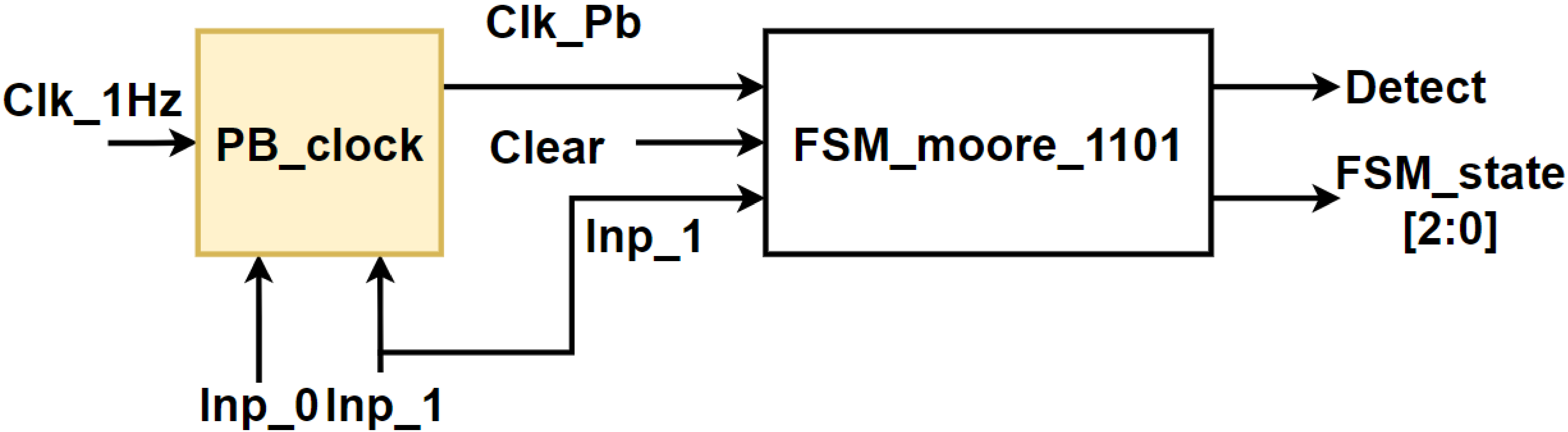
```
module FSM_moore_1101(
    input Clk_pb,
    input Clear,
    input Inp_1,
    output reg Detect,
    output [2:0] FSM_state
    );

    parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 = 3'b100;
    reg [2:0] present_state = S0;
    reg [2:0] next_state;
```

```
always@(posedge Clk_pb or posedge Clear)
begin
    if (Clear)
        present_state <= S0;
    else
        present_state <= next_state;
end
```

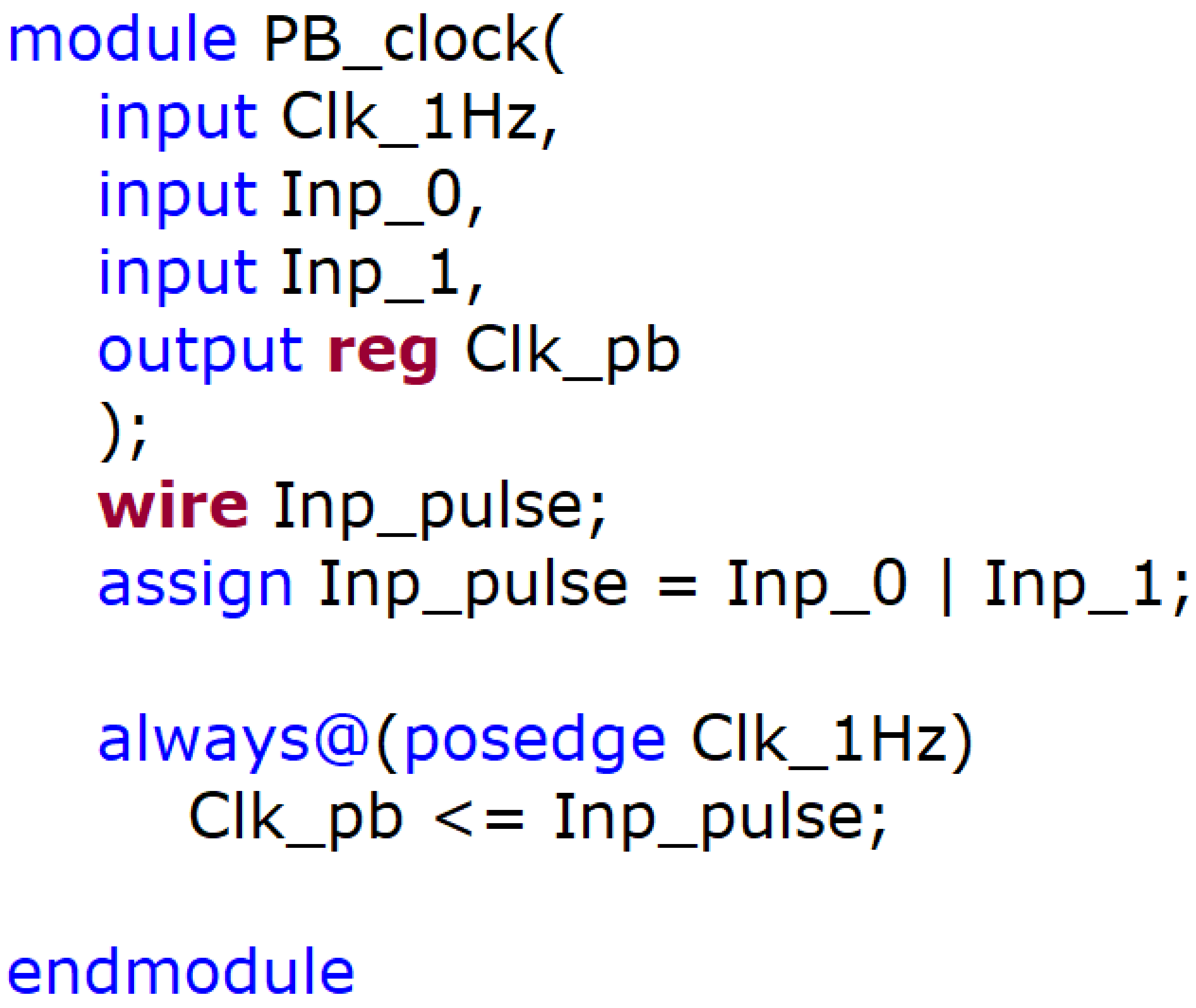
```
always@(*)
begin
    if(present_state == S4)
        Detect = 1;
    else
        Detect = 0;
end
assign FSM_state = present_state;
```

```
always@(*)
begin
   case(present_state)
      S0: if(Inp_1 == 1)
            next_state = S1;
          else
            next_state = S0;
      S1: if(Inp_1 == 1)
            next_state = S2;
          else
            next_state = S0;
      S2: if(Inp_1 == 0)
            next_state = S3;
          else
            next_state = S2;
      S3: if(Inp_1 == 1)
            next_state = S4;
          else
            next_state = S0;

       S4: if(Inp_1 == 0)
             next_state = S0;
           else
             next_state = S2;
       default next_state = S0;
     endcase
 end
```

Detection of Push Button Press

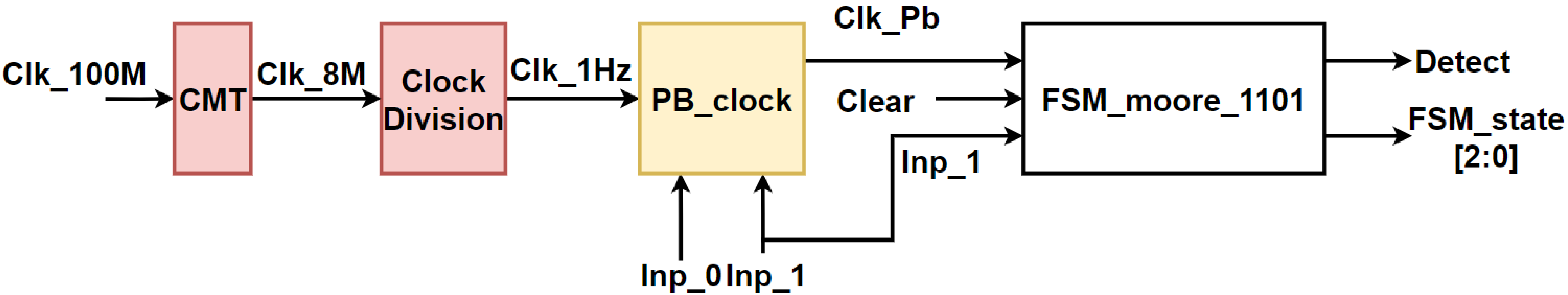


Detection of Push Button Press

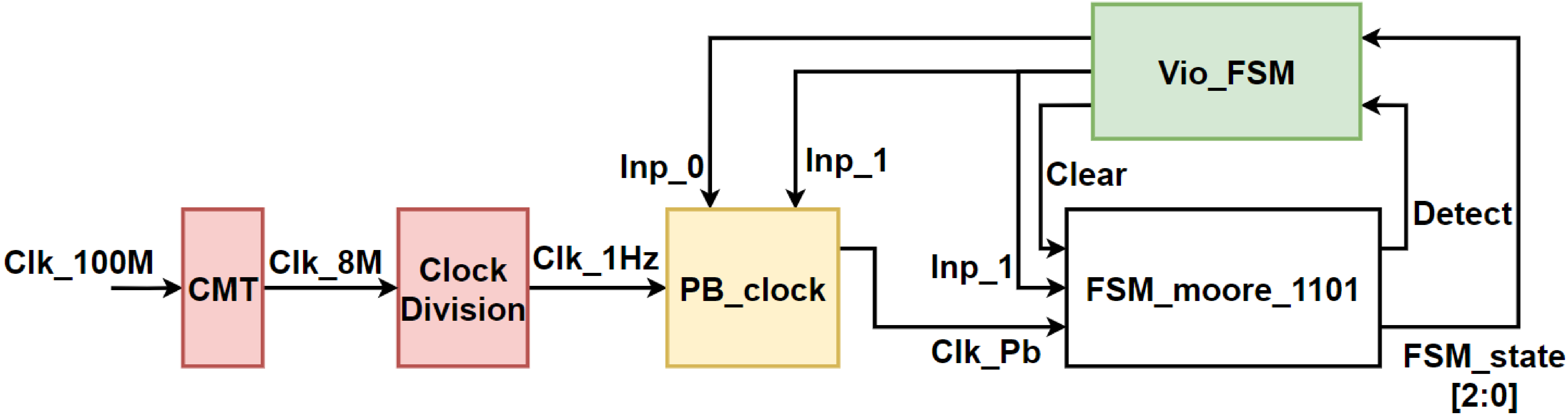


```
module PB_clock(
    input Clk_1Hz,
    input Inp_0,
    input Inp_1,
    output reg Clk_pb
    );
    wire Inp_pulse;
    assign Inp_pulse = Inp_0 | Inp_1;

    always@(posedge Clk_1Hz)
        Clk_pb <= Inp_pulse;

endmodule
```

FSM with Clock Division

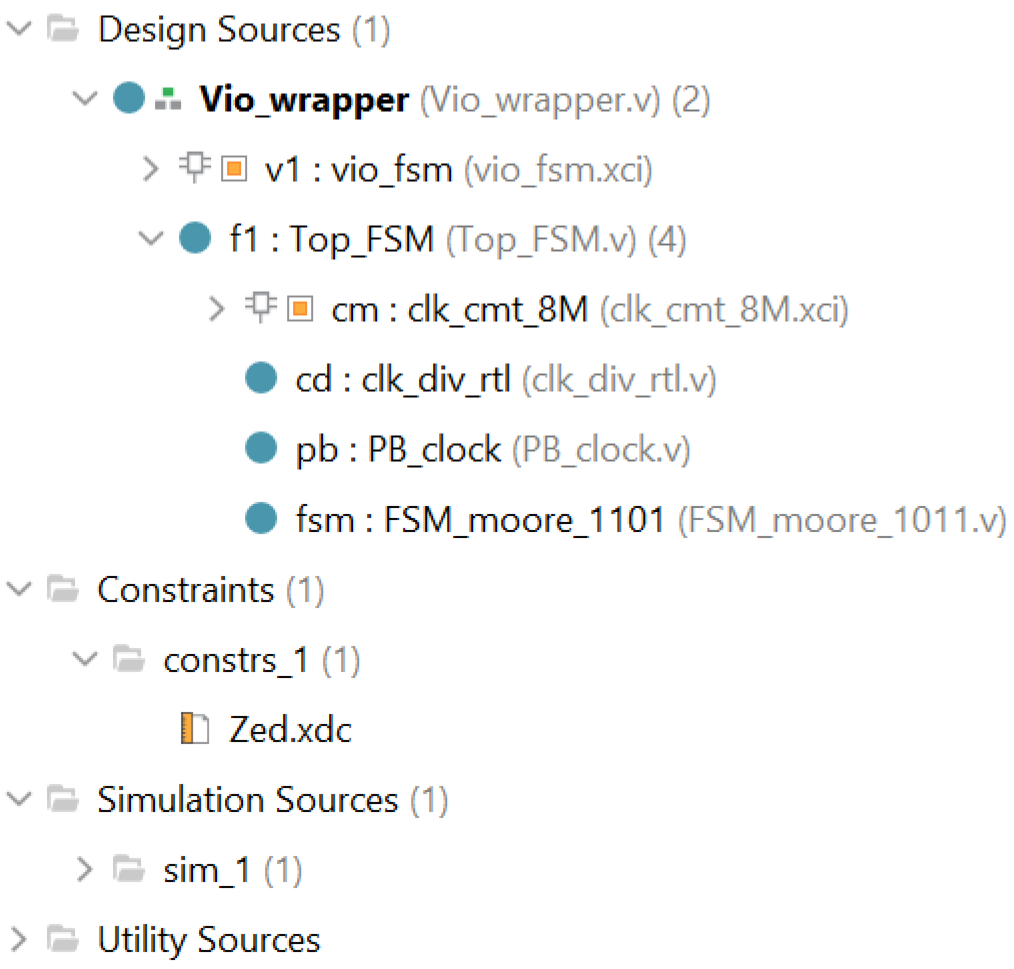


VIO Wrapper


Vio_FSM
Inp_0
Inp_1
Clear
Detect
Clk_100M
CMT
Clk_8M
Clock Division
Clk_1Hz
PB_clock
Inp_1
FSM_moore_1101
Clk_Pb
FSM_state [2:0]

Demo

- Add XDC file and generate bitstream
- Verify the functionality using VIO
- Use toggle button option in VIO



- Design Sources (1)
  - Vio_wrapper (Vio_wrapper.v) (2)
    - v1 : vio_fsm (vio_fsm.xci)
    - f1 : Top_FSM (Top_FSM.v) (4)
      - cm : clk_cmt_8M (clk_cmt_8M.xci)
      - cd : clk_div_rtl (clk_div_rtl.v)
      - pb : PB_clock (PB_clock.v)
      - fsm : FSM_moore_1101 (FSM_moore_1011.v)
- Constraints (1)
  - constrs_1 (1)
    - Zed.xdc
- Simulation Sources (1)
  - sim_1 (1)
- Utility Sources

Possible Extensions

- Moore/Mealy FSMs for any sequence
- Asynchronous/Synchronous active high/low Clear
- Change the FSM input clock to Clk_xHz where x can be any positive integer
- Design a counter (2->4->8->2->4->8…..) using FSM
- FSM with two outputs: Detect and Error. Detect is set to 1 when sequence is correct and Error is set to 1 when sequence is wrong.