CT Verilog Series

Finite-State Machines

黃稚存
Chih-Tsun Huang
cthuang@cs.nthu.edu.tw

國立清華大學
資訊工程學系

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容 (例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

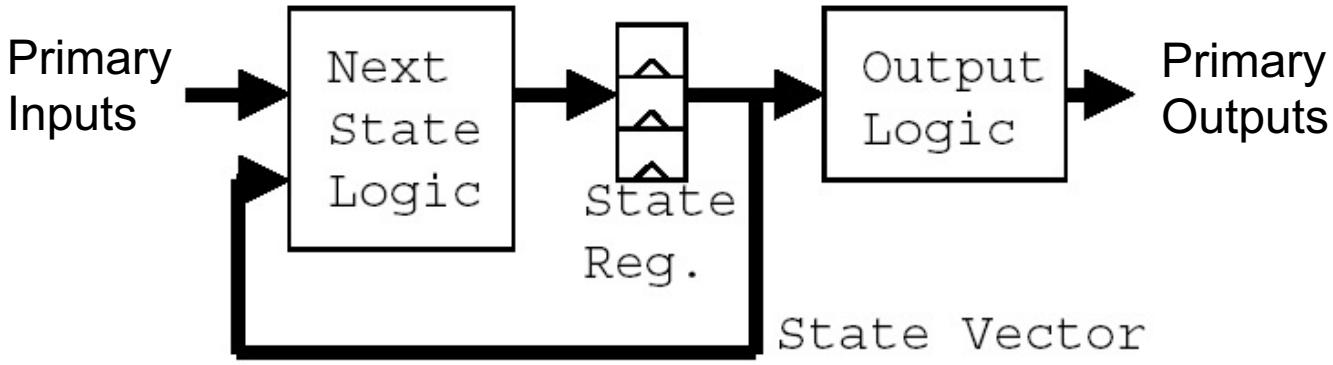
- Mealy machine and Moore machine
- Coding Guidelines
- Example: Level-to-Pulse Converter
- Example: FSM X
- Example: Newspaper Vending Machine
- Summary

digital design
↓
datapath + finite-state machine
(FSM)

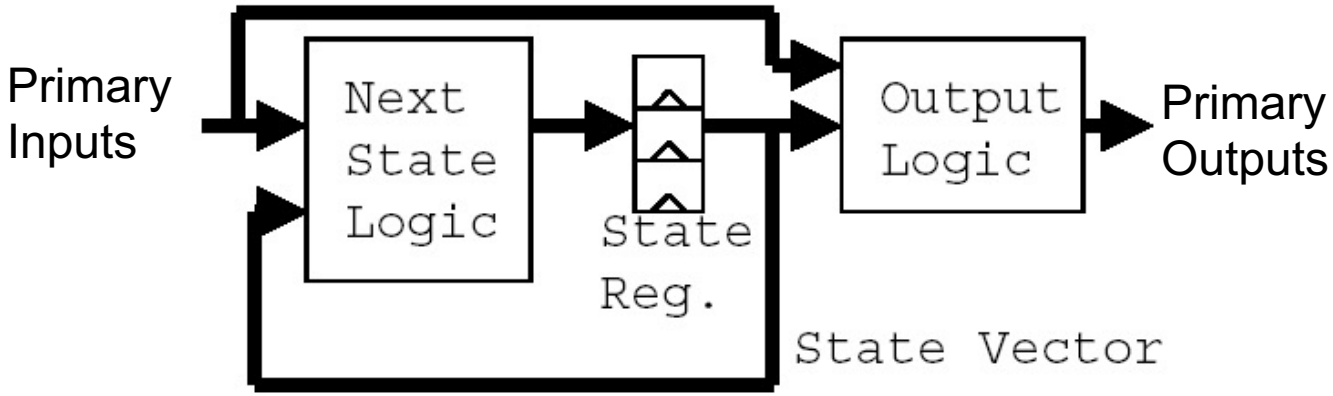
Mealy Machine and Moore Machine

Finite State Machines

- Moore machine and Mealy machine

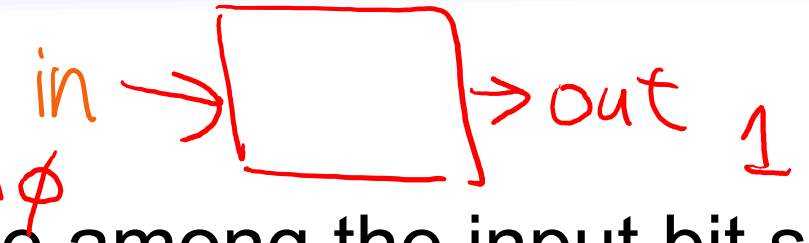


Moore Outputs
Outputs depend
solely on state vector

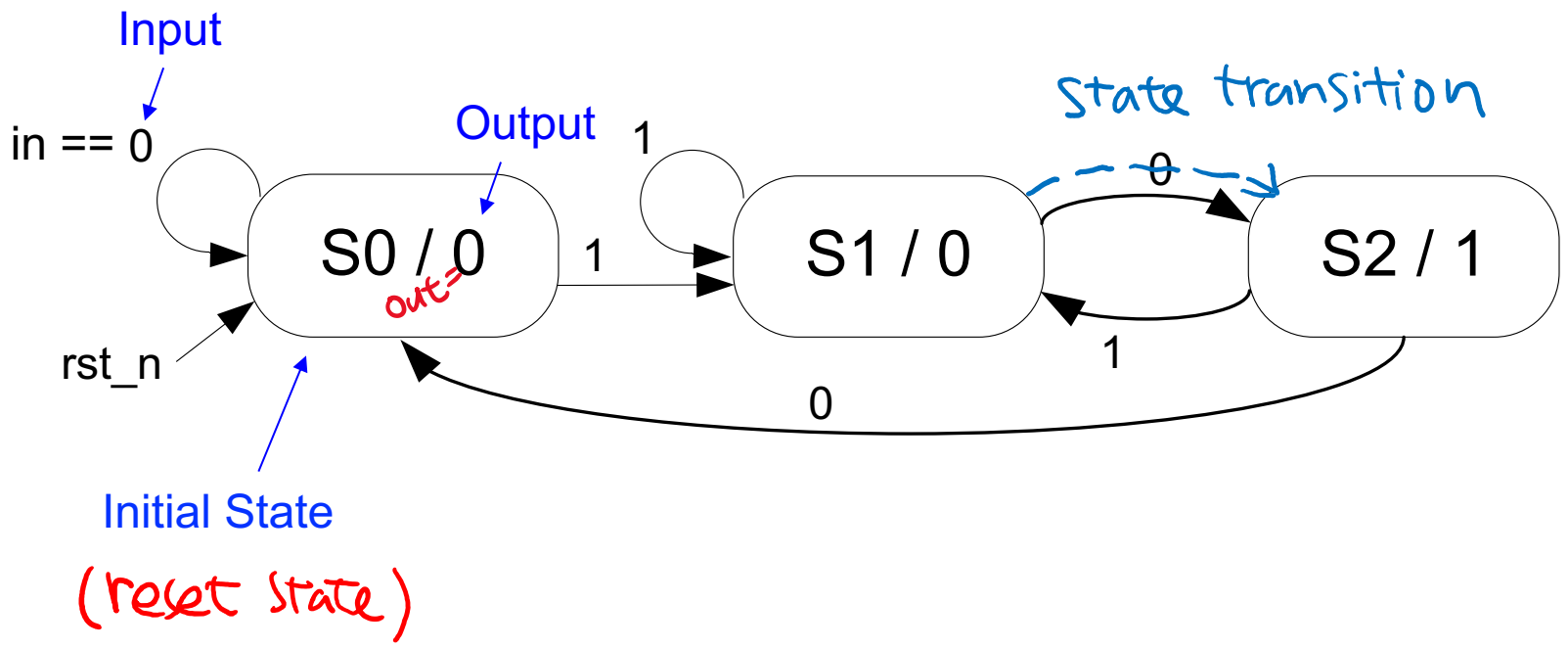


Mealy Outputs
Outputs depend
on inputs and state vector

Moore Machine



- Recognizing the “10” sequence among the input bit stream



Moore Machine Coding

```
parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
reg [2:0] state, next_state;
```

State Update (Sequential)

```
always @(posedge clock, negedge rst_n)
begin
  if (rst_n == 1'b0)
    state <= S0;
  else
    state <= next_state;
end
```

State Transition (Combinational)

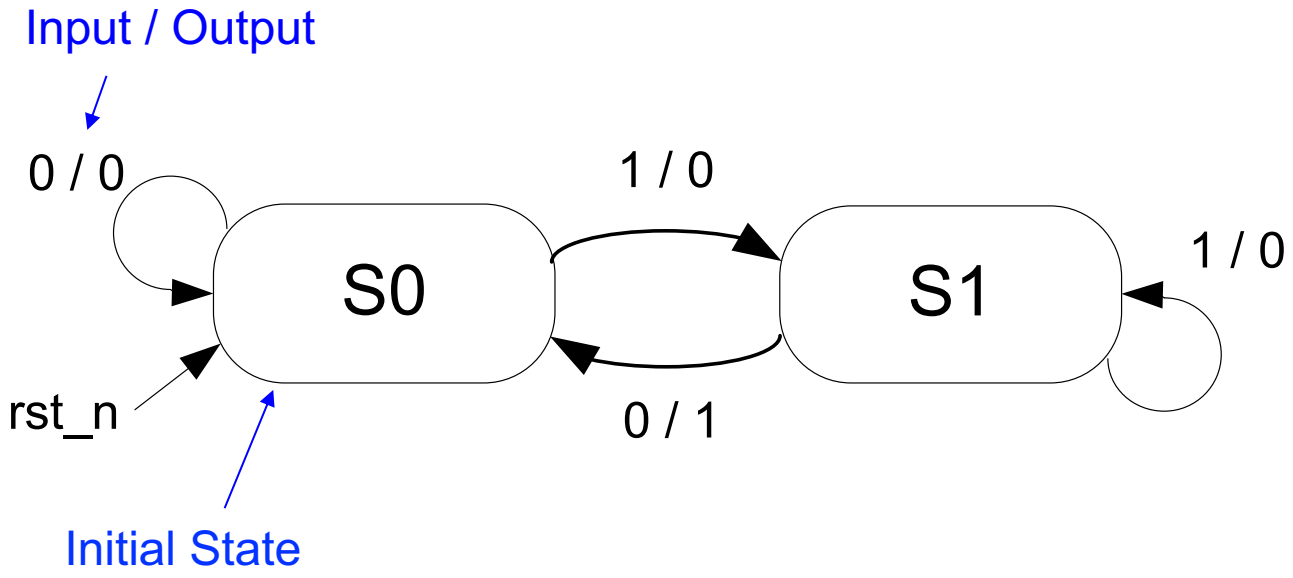
```
always @* begin
  next_state = S0;
  case (state)
    S0: begin
      if (in == 1)
        next_state = S1;
      else
        next_state = S0;
    end
    S1: begin
      if (in == 1)
        next_state = S1;
      else
        next_state = S2;
    end
    S2: begin
      if (in == 1)
        next_state = S1;
      else
        next_state = S0;
    end
  endcase // case end
end //always end
```

```
assign out = state == S2 ? 1 : 0;
```

Outputs (Combinational)

Mealy Machine

- Recognizing the “10” sequence among the input bit stream



Mealy Machine Coding

```
parameter S0 = 1'b0;
parameter S1 = 1'b1;
reg state, next_state;
```

State Update (Sequential)

```
always @(posedge clk, negedge rst_n) begin
  if (rst_n == 1'b0)
    state <= S0;
  else
    state <= next_state;
end
```

```
always @* begin
  next_state = S0;
  case(state)
    S0: begin
      if (in == 1)
        next_state = S1;
      else
        next_state = S0;
    end
    S1: begin
      if (in == 1)
        next_state = S1;
      else
        next_state = S0;
    end
  endcase // case end
end // always end
```

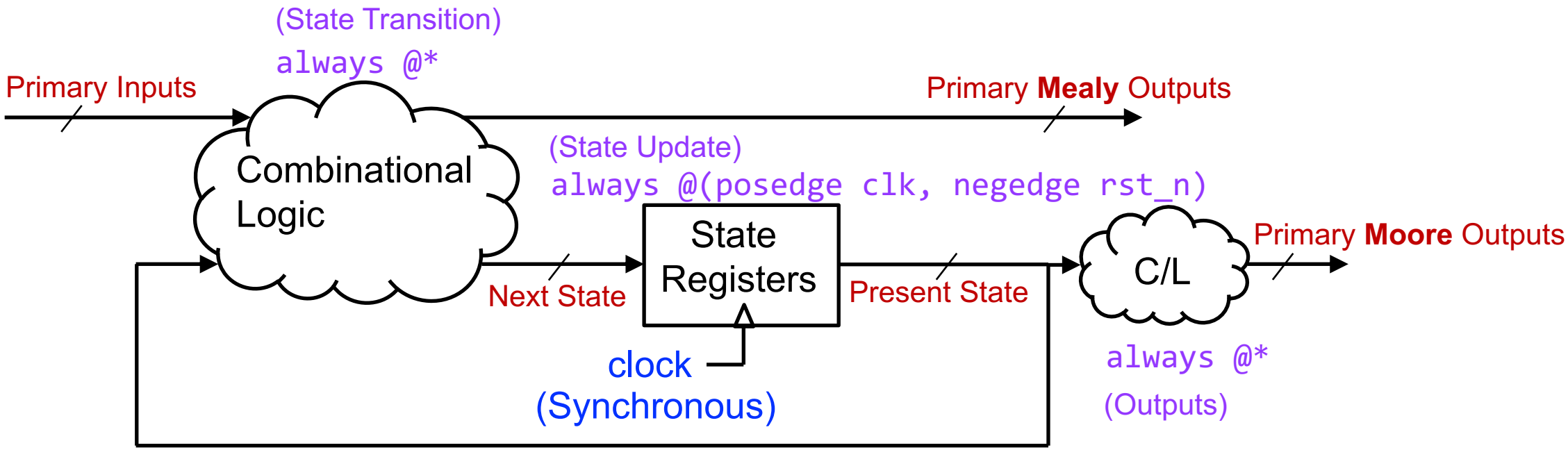
State Transition (Combinational)

```
assign out =
    (state == S1 && in == 0) ? 1 : 0;
```

Outputs (Combinational)

Coding Guidelines

FSM (Sequential Block) Modeling



Guidelines for FSM Modeling

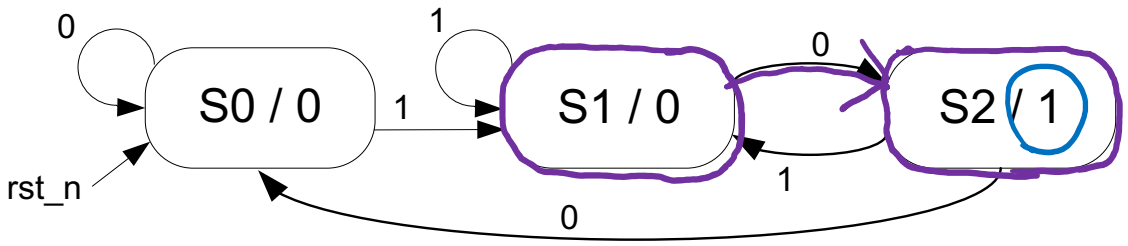
- An `always` block for updating state registers
  - Sequential block
- An `always` block for state transition (next state)
  - Combinational block
- An optional `always` block for output generation
  - Combinational block
- Parameterize all state encoding

Moore vs. Mealy (1/2)

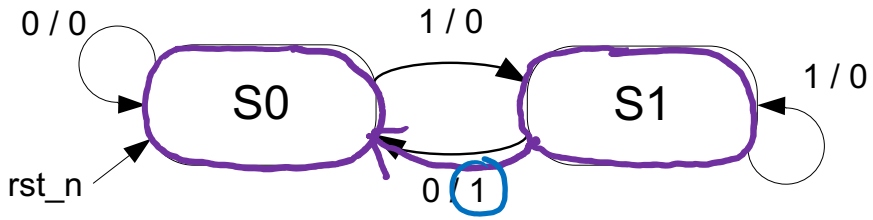
- Difference
  - Moore: outputs depend on state only
  - Mealy: outputs depend on state and inputs
    - Might be less straightforward
    - Generally fewer states
  - Any timing difference?

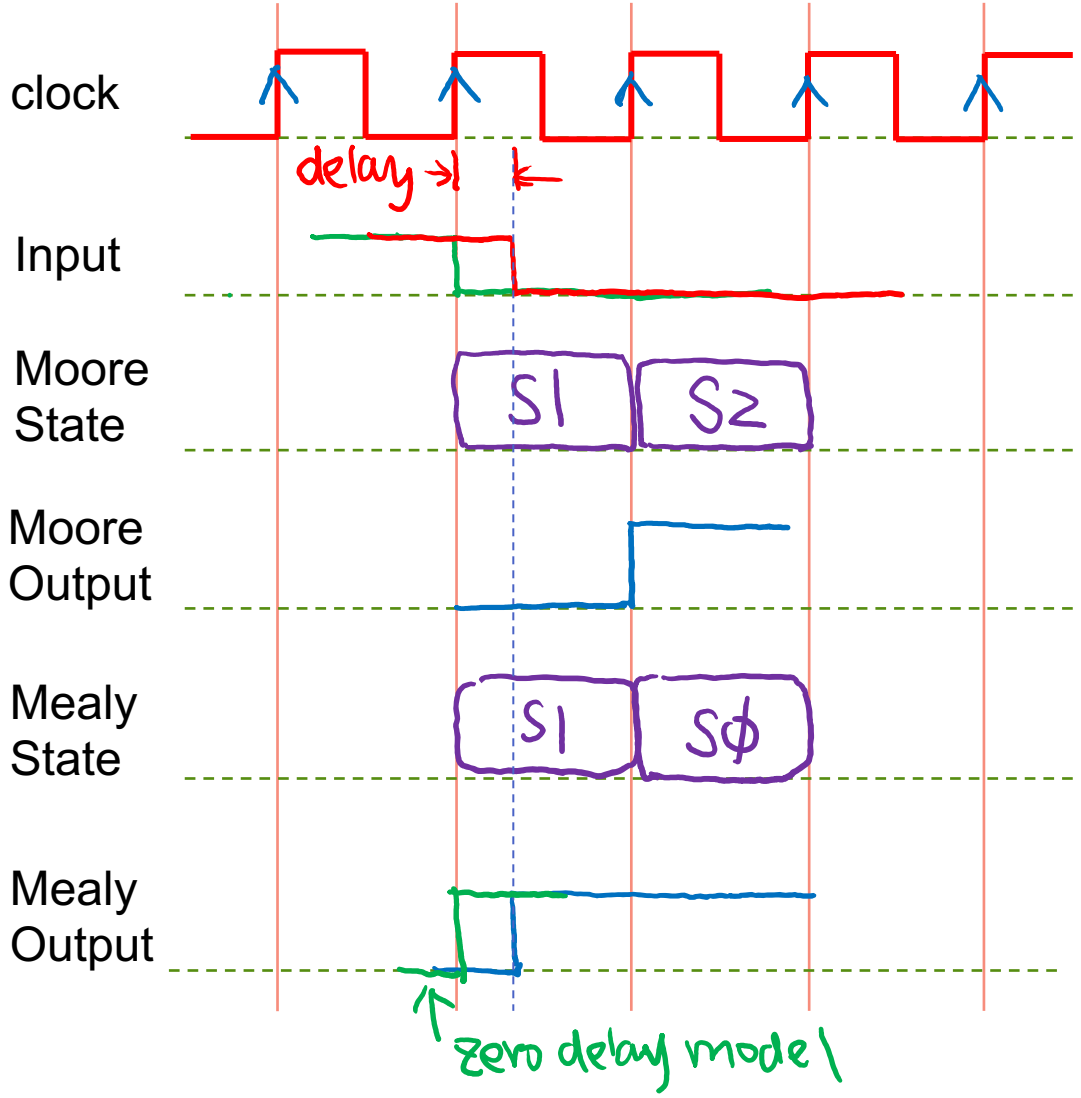
Moore vs. Mealy (2/2)

Moore



Mealy



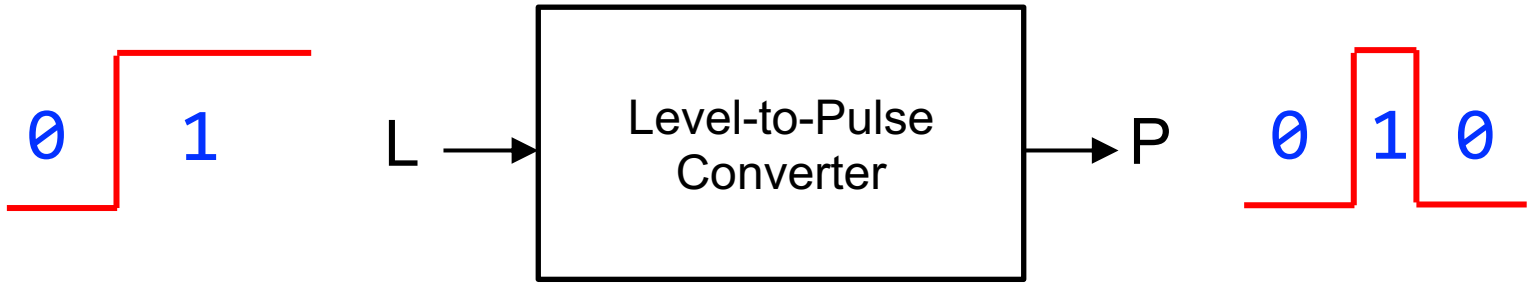


Timing Difference

Example: Level-to-Pulse Converter

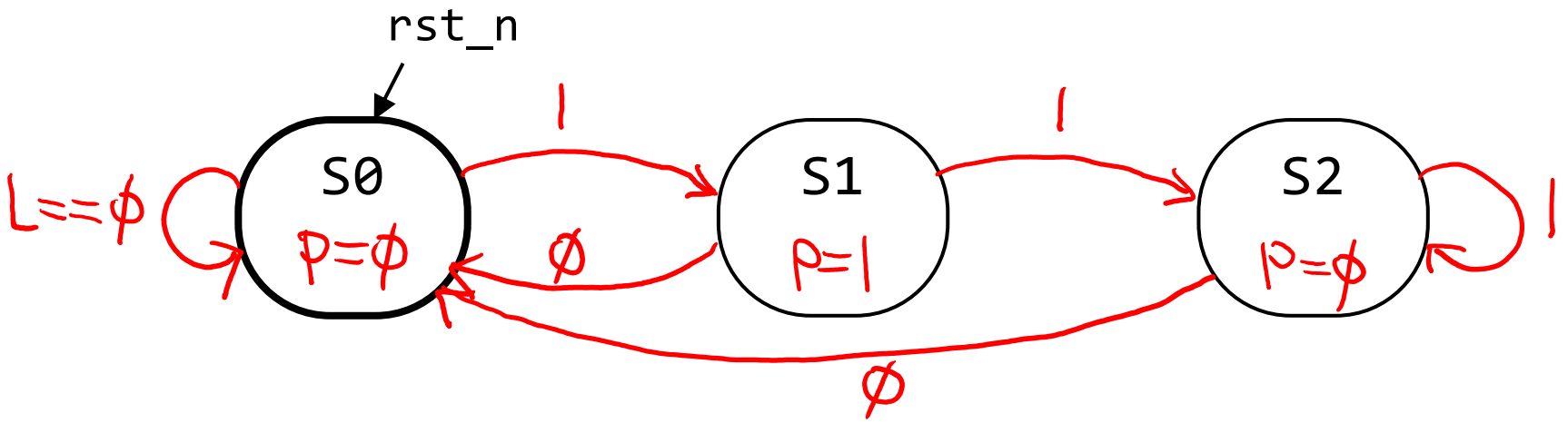
Level-to-Pulse Converter

- A level-to-pulse converter
  - Produces a single-cycle pulse each time its input goes high
  - Synchronous rising-edge detector
- Applications
  - Pushbuttons and switches pressed by humans
  - Single-cycle enable signals

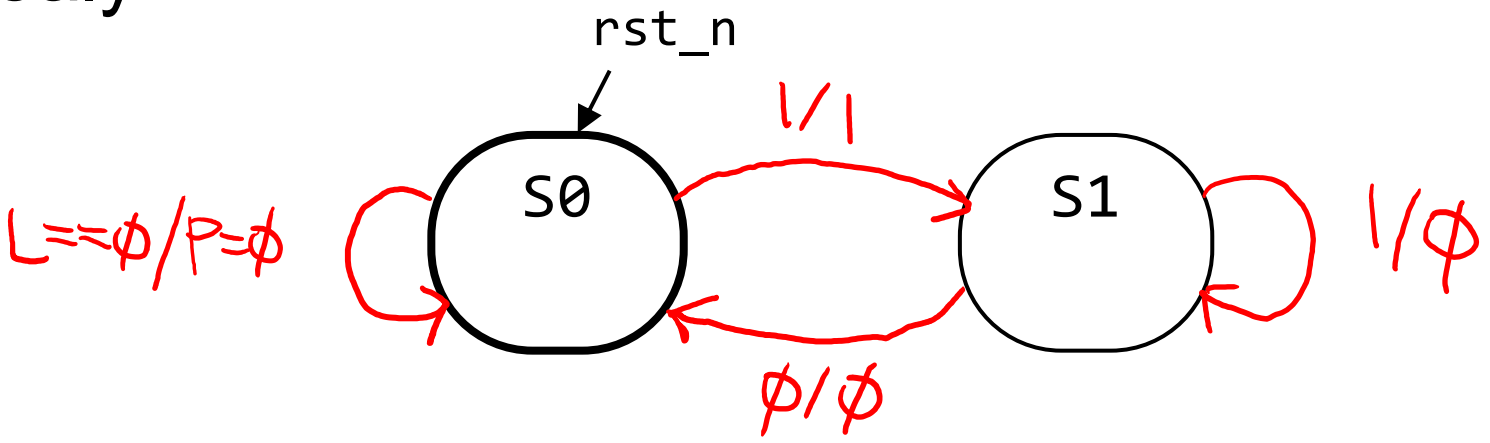


Finite-State Machine

- Moore

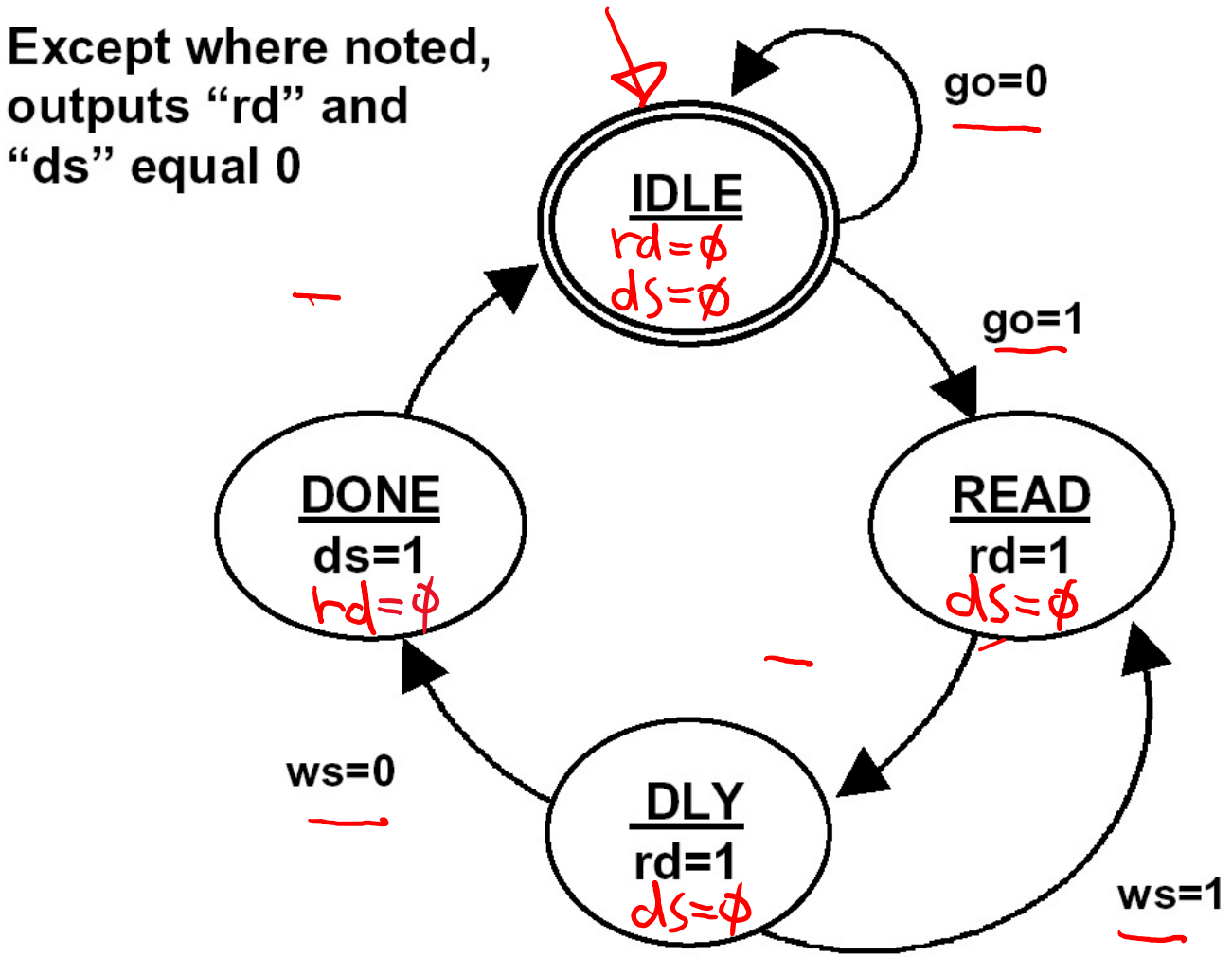


- Mealy



Example: FSM X

FSM X


Except where noted, outputs “rd” and “ds” equal 0
IDLE
rd=ϕ
ds=ϕ
go=0
go=1
READ
rd=1
ds=ϕ
DLY
rd=1
ds=ϕ
ws=1
ws=0
DONE
ds=1
rd=ϕ

Coding Example 1

```
module fsm1 (ds, rd, go, ws, clk, rst_n);
  output ds, rd;
  input go, ws;
  input clk, rst_n;
  parameter [1:0]
    IDLE = 2'b00,
    READ = 2'b01,
    DLY = 2'b10,
    DONE = 2'b11;
  reg [1:0] state, next;
  always @(posedge clk or negedge rst_n)
    if (!rst_n) state <= IDLE;
    else state <= next;
```

```
  always @* begin
    next = IDLE;
    case (state)
      IDLE: if (go) next = READ;
            else next = IDLE;
      READ: next = DLY;
      DLY:  if (ws) next = READ;
            else next = DONE;
      DONE: next = IDLE;
    endcase
  end
  assign rd = (state==READ || state==DLY);
  assign ds = (state==DONE);
endmodule
```

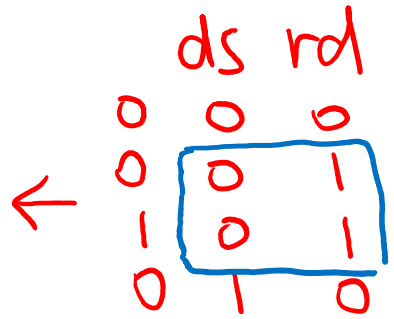
Coding Example 2

```
module fsm2 (ds, rd, go, ws, clk, rst_n);
  output ds, rd;
  input go, ws;
  input clk, rst_n;
  reg ds, rd;
  parameter [1:0]
    IDLE = 2'b00,
    READ = 2'b01,
    DLY = 2'b10,
    DONE = 2'b11;
  reg [1:0] state, next;
  always @(posedge clk or negedge rst_n)
    if (!rst_n) state <= IDLE;
    else state <= next;
```

```
  always @* begin
    next = IDLE;
    ds = 1'b0;
    rd = 1'b0;
    case (state)
      IDLE: if (go) next = READ;
            else next = IDLE;
      READ: begin
        rd = 1'b1;
        next = DLY;
      end
      DLY: begin
        rd = 1'b1;
        if (ws) next = READ;
        else next = DONE;
      end
      DONE: begin
        ds = 1'b1;
        next = IDLE;
      end
    endcase
  end
endmodule
```

Coding Example 3

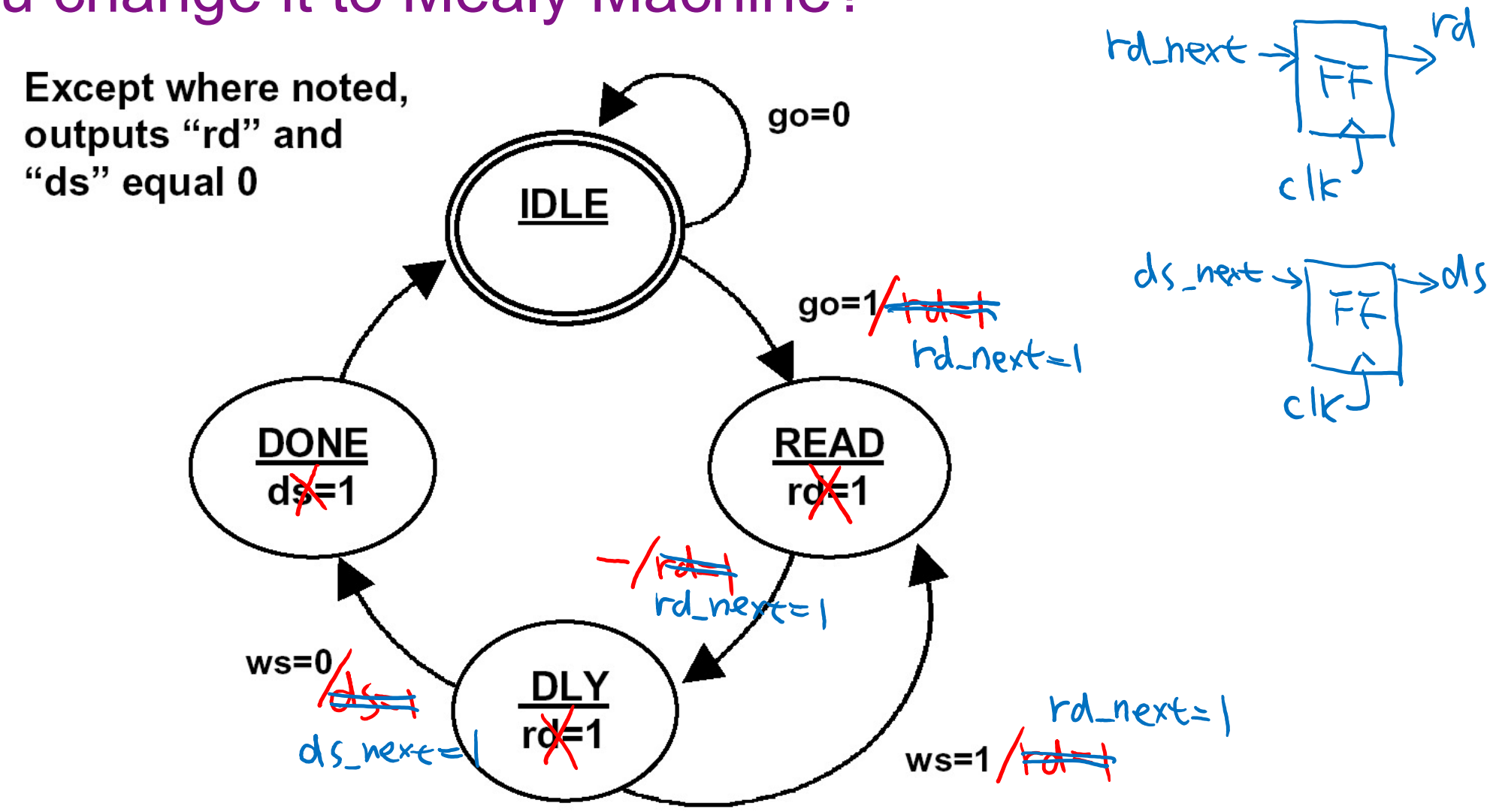
```
module fsm3 (ds, rd, go, ws, clk, rst_n);
 output ds, rd;
 input go, ws;
 input clk, rst_n;

 parameter [2:0]
     IDLE = 3'b0_00,
     READ = 3'b0_01,
     DLY =  3'b1_01,
     DONE = 3'b0_10;
 reg [2:0] state, next;
 always @(posedge clk or negedge rst_n)
   if (!rst_n) state <= IDLE;
   else state <= next;
```




```
 always @* begin
   next = IDLE;
   case (state)
      IDLE: if (go) next = READ;
            else next = IDLE;
      READ: next = DLY;
      DLY: if (ws) next = READ;
           else next = DONE;
      DONE: next = IDLE;
    endcase
 end

 assign {ds,rd} = state[1:0];

endmodule
```

Example: FSM X

Can you change it to Mealy Machine?



Part of Stimulus for FSM X

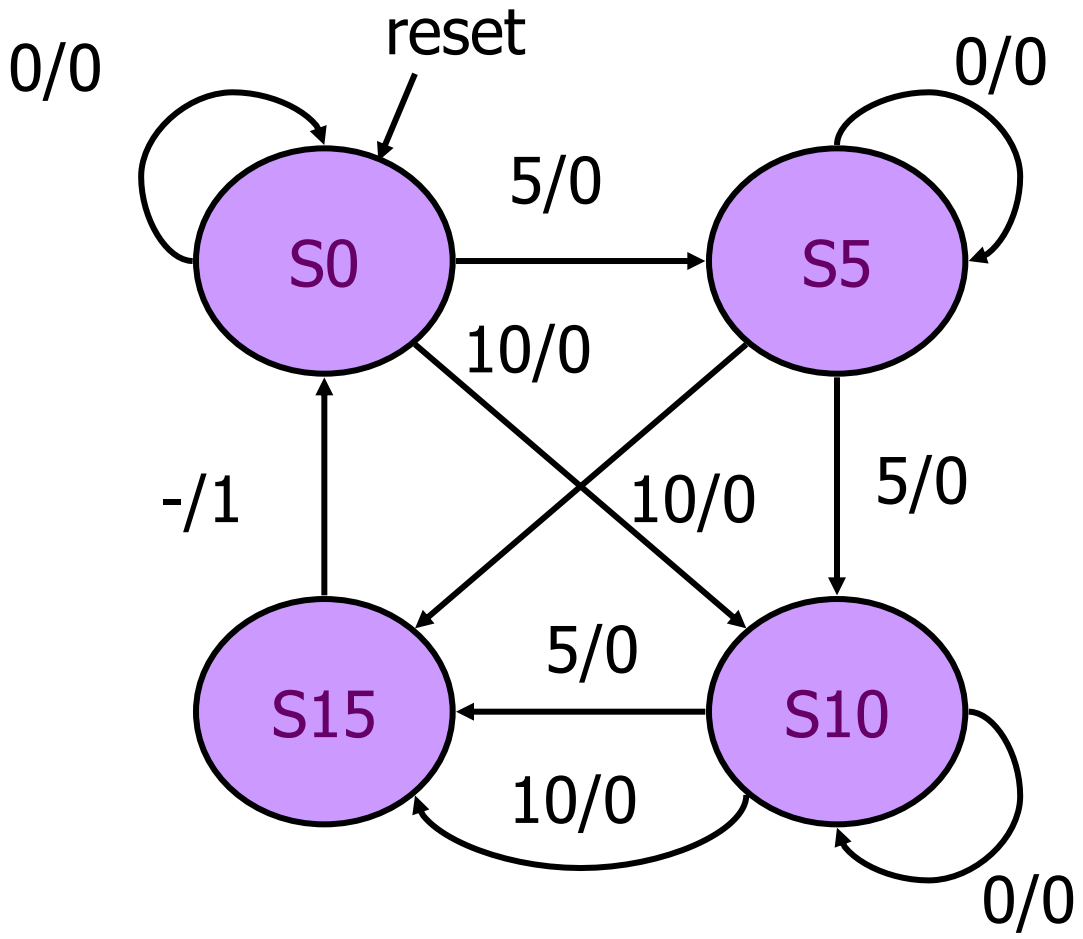
```
initial begin
  clk = 0;
  rst_n = 1;
  go = 0;
  ws = 0;

  #(cyc/2);
  #(cyc*2) rst_n = 0;
  #(cyc*2) rst_n = 1;
  #(cyc);
  #(cyc);
  #(cyc) go = 1; ws = 1;
  #(cyc*3);
  #(cyc); ws = 0;
  #(cyc); go = 0;
  #(cyc*5);
  $finish;
end
```

Example:
Newspaper Vending Machine

Newspaper Vending Machine Example

- Rule:
  - Each newspaper cost 15 Dollars
  - Only 10 and 5 Dollars coin
  - No change
  - Possible combination: 3x 5 coins, one 10 coin and one 5 coin. Other combinations are not allowable.
- Requirements:
  - Send out control signal for each coin inserted
  - One output signal which enable newspaper delivery
  - One reset signal to reset the vending machine

State Diagram



FSM Verilog Code (1/2)

```
module vend (
  coin, clock, reset, newspaper
);
  input [1:0] coin;
  input clock;
  input reset;
  output newspaper;
  reg newspaper;
  reg [1:0] state, next_state;

  // state encoding
  parameter s0 = 2'b00;
  parameter s5 = 2'b01;
  parameter s10 = 2'b10;
  parameter s15 = 2'b11;
```

```
// combinational
 always @* begin
   case (state)
     s0: begin
       if (coin == 2'b10) begin
         newspaper = 1'b0;
         next_state = s10;
       end else if (coin == 2'b01) begin
         newspaper = 1'b0;
         next_state = s5;
       end else begin
         newspaper = 1'b0;
         next_state = s0;
       end
     end
     s5: begin
       if (coin == 2'b10) begin
         newspaper = 1'b0;
         next_state = s15;
```

FSM Verilog Code (2/2)

```
      end else if (coin == 2'b01) begin
        newspaper = 1'b0;
        next_state = s10;
      end else begin
        newspaper = 1'b0;
        next_state = s5;
      end
    end
    s10: begin
      if (coin == 2'b10) begin
        newspaper = 1'b0;
        next_state = s15;
      end else if (coin == 2'b01) begin
        newspaper = 1'b0;
        next_state = s15;
      end else begin
        newspaper = 1'b0;
        next_state = s10;
      end
    end
```

```
      s15: begin
        newspaper = 1'b1;
        next_state = s0;
      end
      default: begin
        newspaper = 1'b0;
        next_state = s0;
      end
    endcase
  end

  // state update, synchronous reset
  always @(posedge clock) begin
    if (reset == 1'b1)
      state = s0;
    else
      state = next_state;
  end
endmodule
```

Summary

- Design your FSM before Verilog coding
  - Define states, state transitions, and outputs clearly
- Follow the coding guidelines
  - Explicit combinational blocks (state transition / output logic) + sequential block (state registers)
- One design can have multiple FSMs
  - Keep each FSM simple enough
- One state can execute for multiple clock cycles
- FSM must be able to return its initial state without using the reset