EE-421: Digital System Design

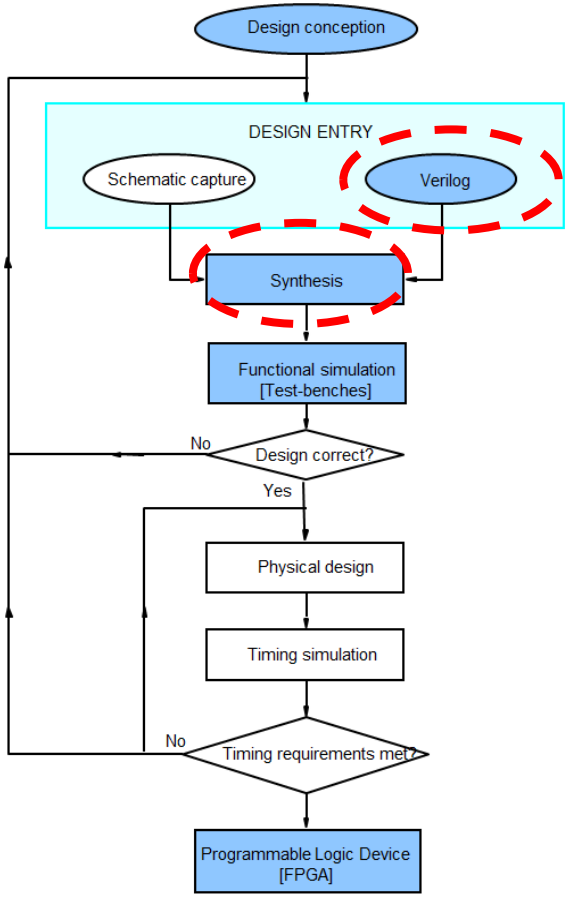
Finite State Machines (FSMs) and It's coding in Verilog HDL

Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

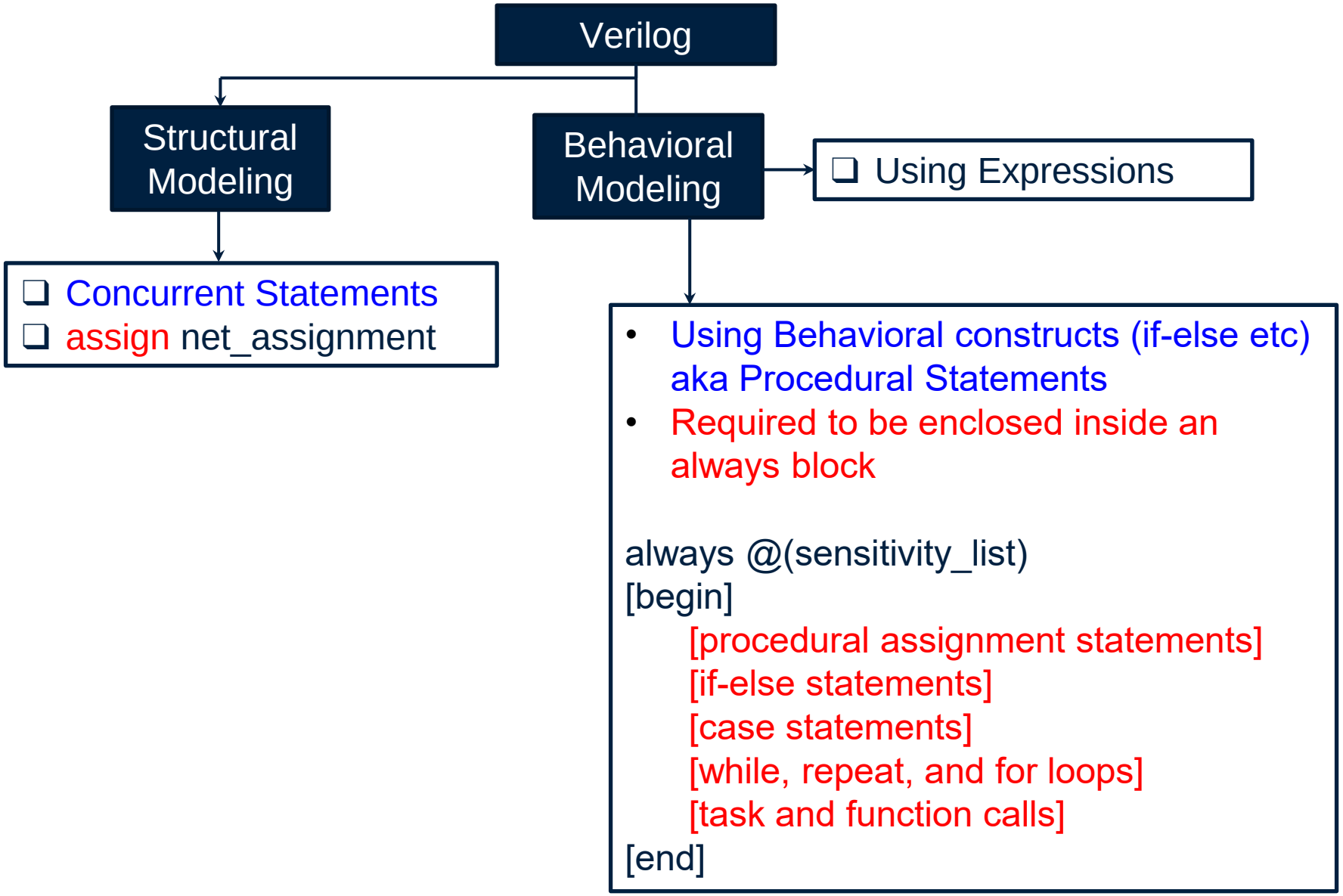
NUST

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE (SEECS)

Where are we heading?



Verilog Recap: What have we seen so far . . .



Takeaway: Expressing Combinational and Sequential Ccts in Verilog

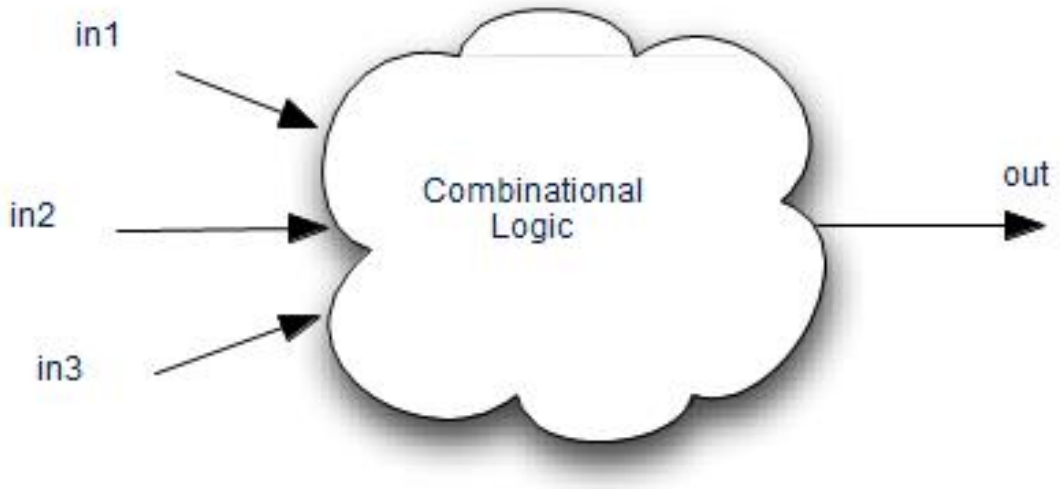
Takeaway

- Expressing Combinational or Sequential circuits using Procedural Statements:
  - Remember THREE patterns (discussed in the next 3-slides)
    - Purely based on observation and experience!
    - Not part of IEEE standard (not at all)!
    - Books might not preach that!

Always Pattern - 1

- **Combinational Logic:**

always@(in1, in2, in3) = always @ (*)

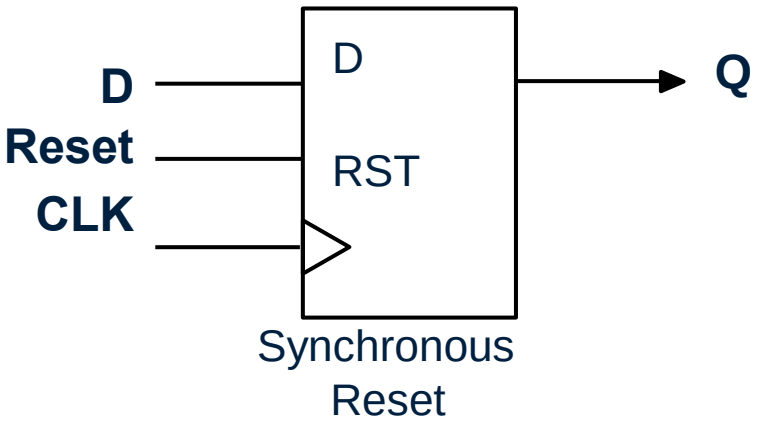


- Simulation – outputs are re-evaluated anytime a signal in the sensitivity list changes.
- Synthesis – The synthesized block's output may only change when a change occurs on one of the sensitivity list signals.

Always Pattern - 2

- **Sequential Logic and Synchronous Reset:**

always@(posedge CLK)

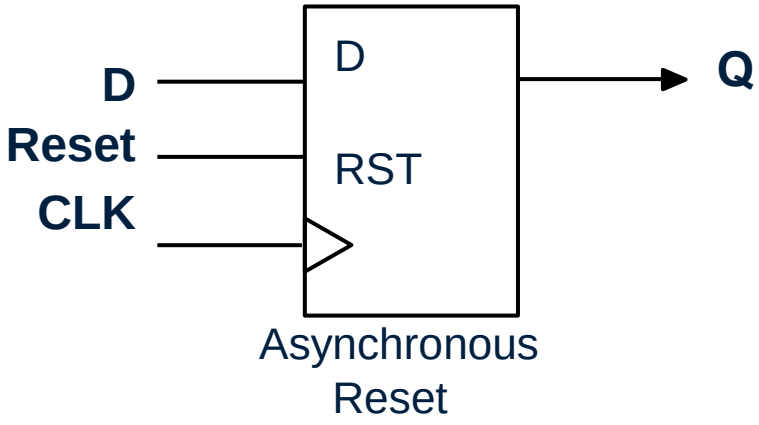


- Only the clock signal goes inside the sensitivity list!!!

Always Pattern - 3

- **Sequential Logic and Asynchronous Reset:**

```
always@(posedge CLK or negedge RESET)
```



- Only the clock and reset signals go inside the sensitivity list!!!

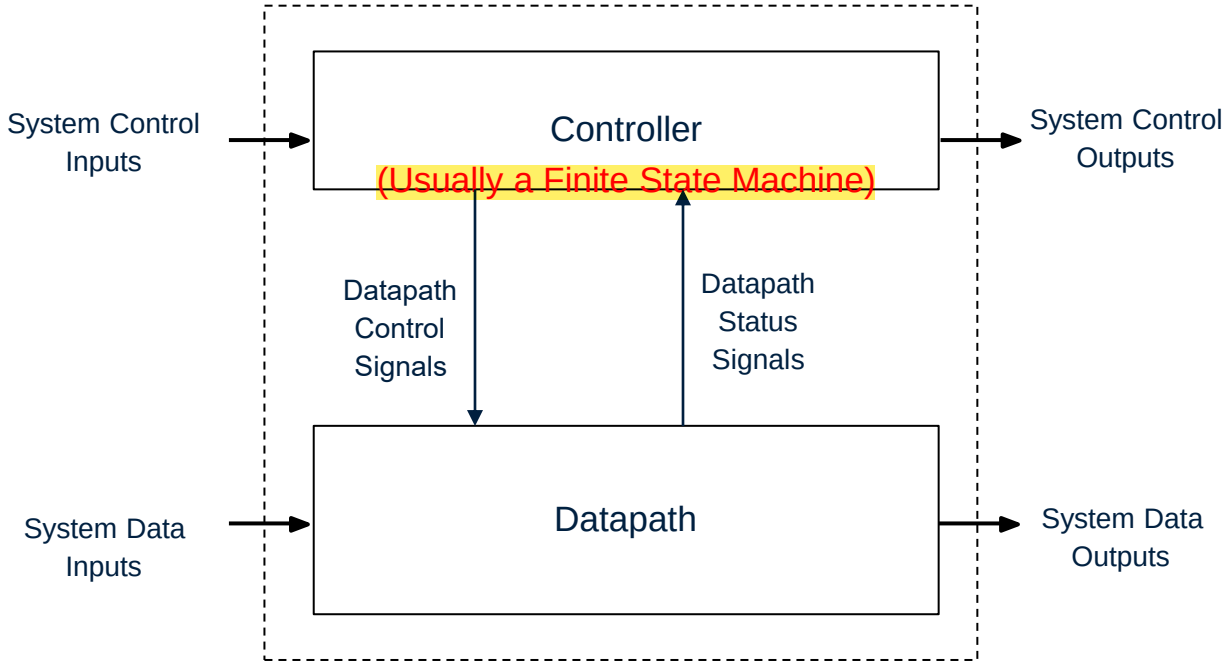
Finite State Machines

Application of FSM

DATAPATH – Collection of functional units that process data

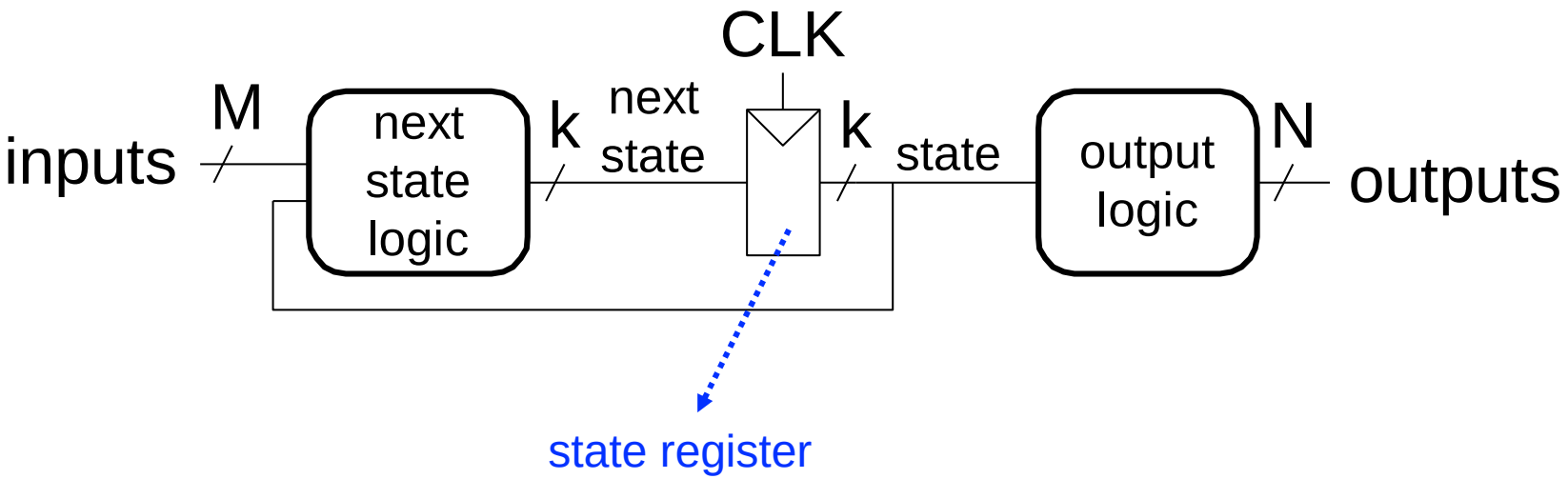
- Functional units are ultimately implemented as combinational and sequential logic

CONTROLLER – Coordinates the operations in the datapath



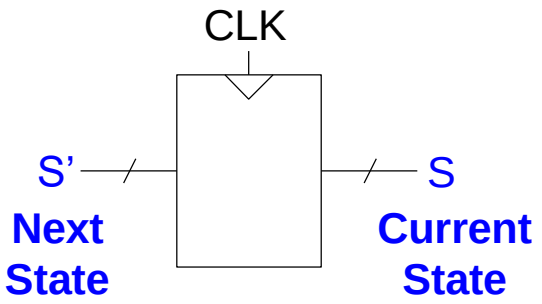
Recall: Finite State Machines (FSMs)

- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic

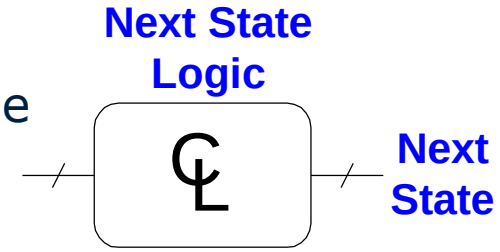


Finite State Machines (FSMs) Comprise

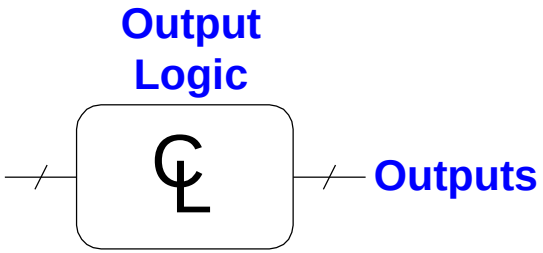
- **Sequential circuits**
  - State register(s)
    - Store the current state and
    - Load the next state at the clock edge



- **Combinational Circuits**
  - Next state logic
    - Determines what the next state will be

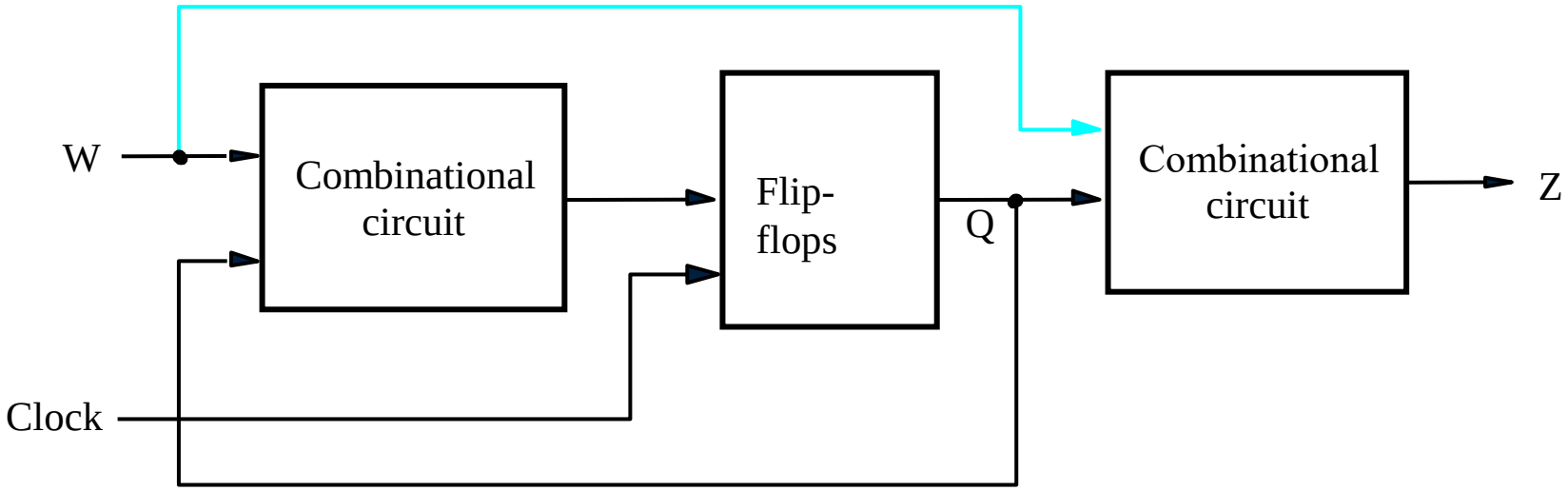


  - Output logic
    - Generates the outputs



Review: Moore vs Mealy

- In a sequential circuit, the values of the outputs depend on the past behavior of the circuit, as well as the present values of its inputs.
  - Moore: If the outputs depend only on the present state.
  - Mealy: If the outputs depend on both the present state and the present values of the inputs.



Example - FSM

- Consider an application where the speed of an automatically-controlled vehicle has to be regulated as follows:
  - The vehicle is designed to run at some predetermined speed. However, due to some operational conditions the speed may exceed the desirable limit, in which case the vehicle has to be slowed down. To determine when such action is needed, the speed is measured at regular intervals.
  - Let a binary signal w indicate whether the speed exceeds the required limit, such that w = 0 means that the speed is within acceptable range and w = 1 indicates excessive speed.
  - The desired control strategy is that if w = 1 during two or more consecutive measurements, a control signal z must be asserted to cause the vehicle to slow down. Thus, z = 0 allows the current speed to be maintained, while z = 1 reduces the speed.
  - Let a signal Clock define the required timing intervals, such that the speed is measured once during each clock cycle.

Example – Circuit Specification

- Therefore, we wish to design a circuit that meets the following specification:

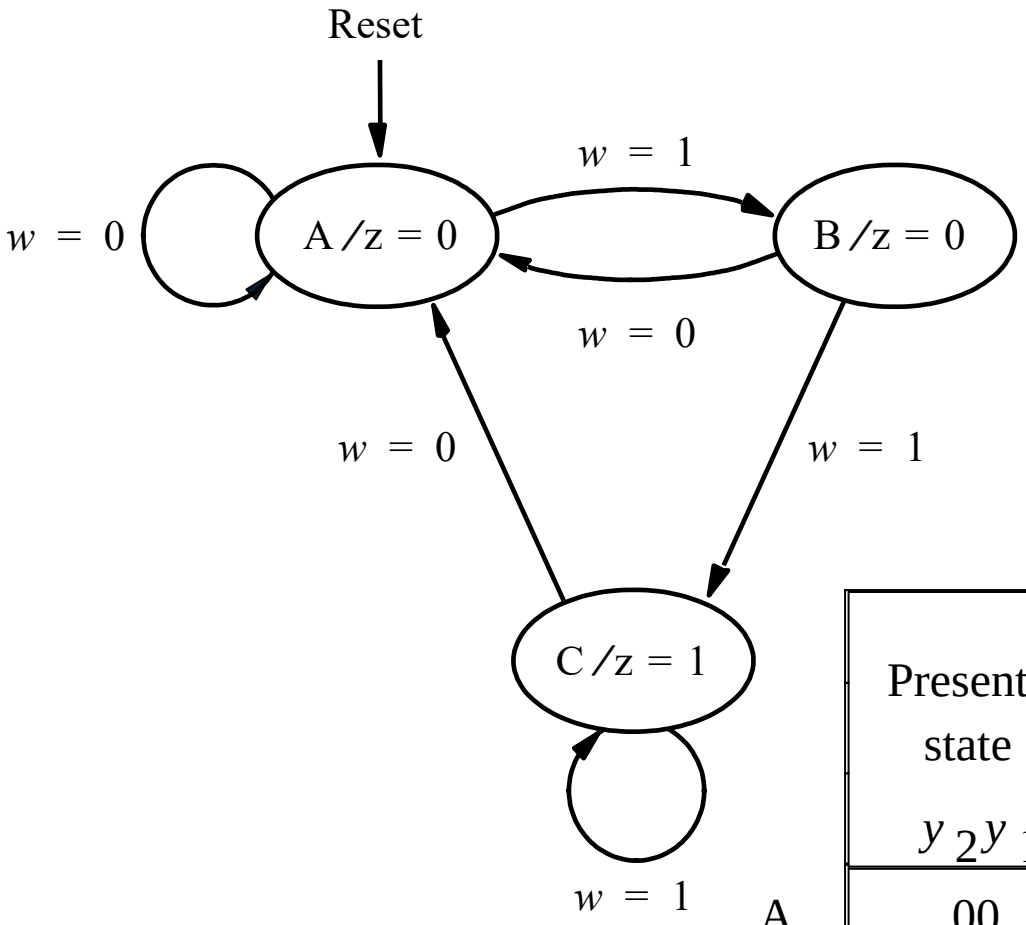
1. The circuit has one input, w, and one output, z.

2. All changes in the circuit occur on the positive edge of the clock signal.

3. The output z is equal to 1 if during two immediately preceding clock cycles the input w was equal to 1. Otherwise, the value of z is equal to 0.

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Recall: State-Diagram and State-Tables



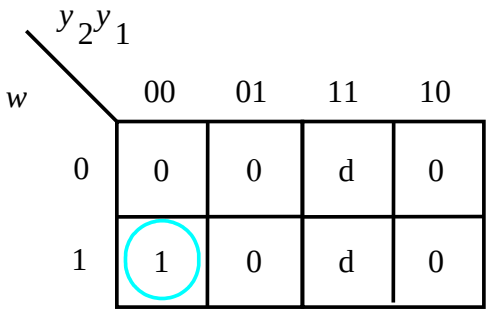
State Table

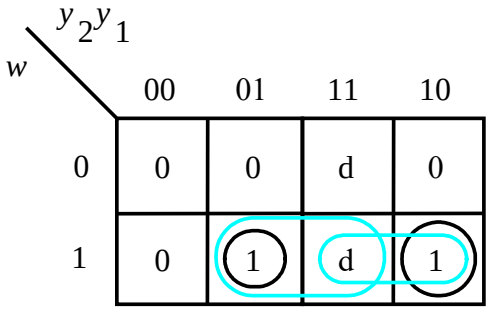
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

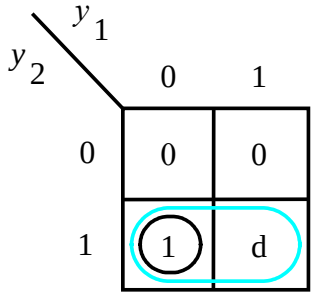
State Assigned Table

	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Derivation for Next-state and Output Logic Expressions







Ignoring don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$$

$$z = \bar{y}_1y_2$$

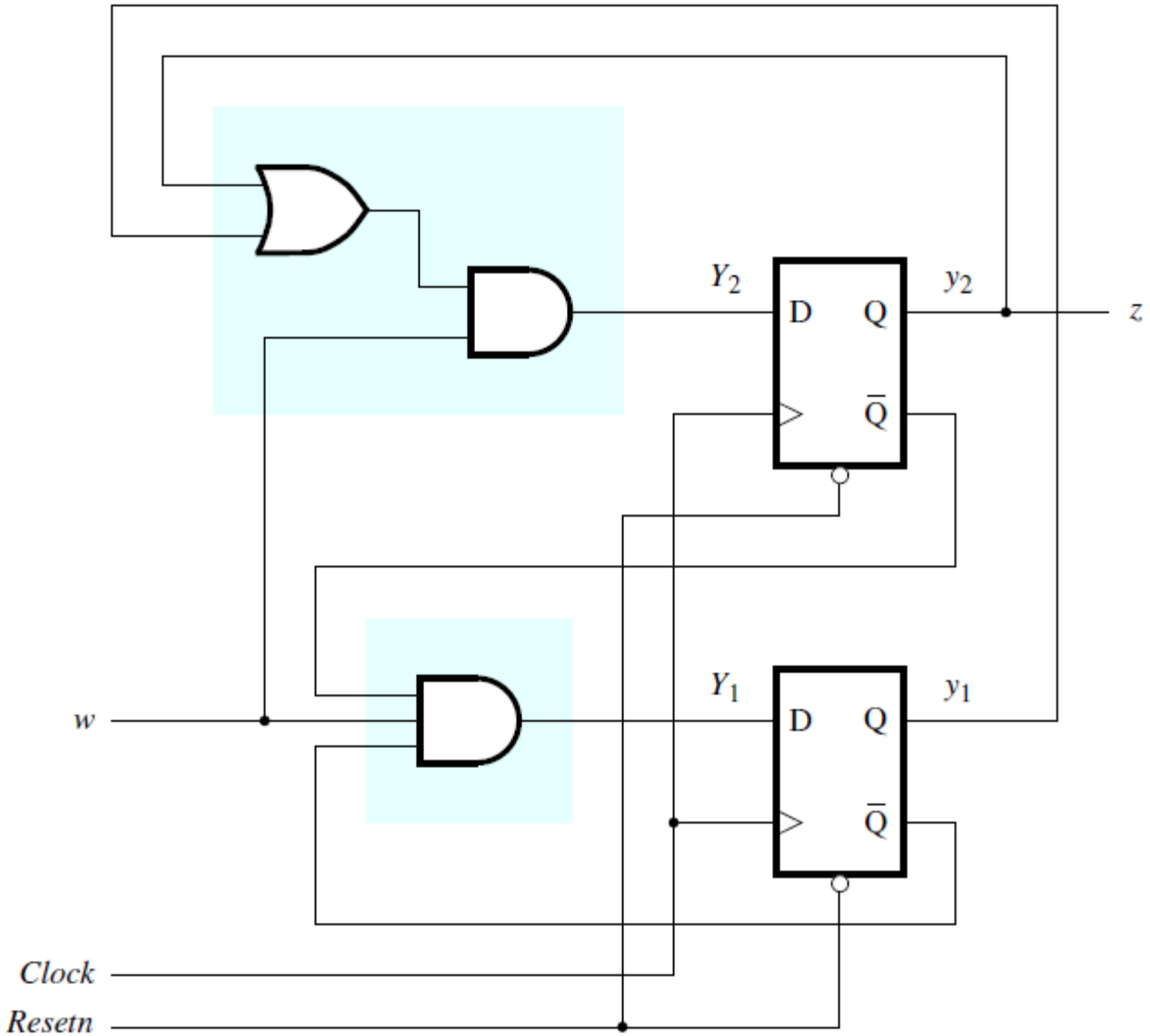
Using don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1 + wy_2$$
$$= w(y_1 + y_2)$$

$$z = y_2$$

Final Implementation



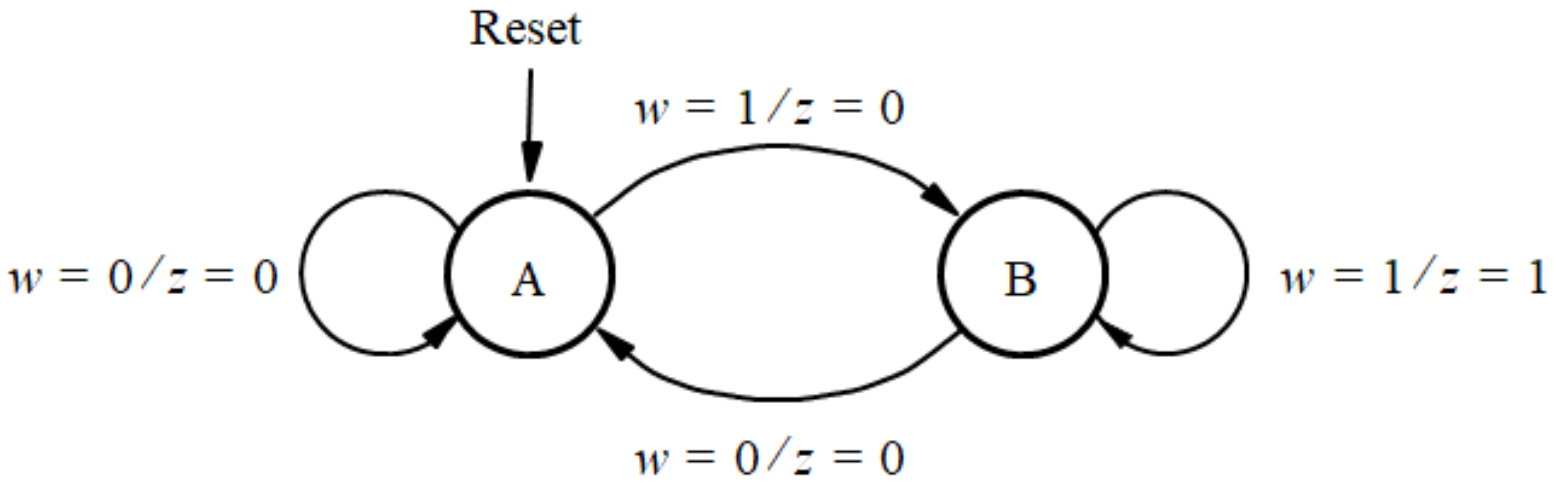
Mealy State Model

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

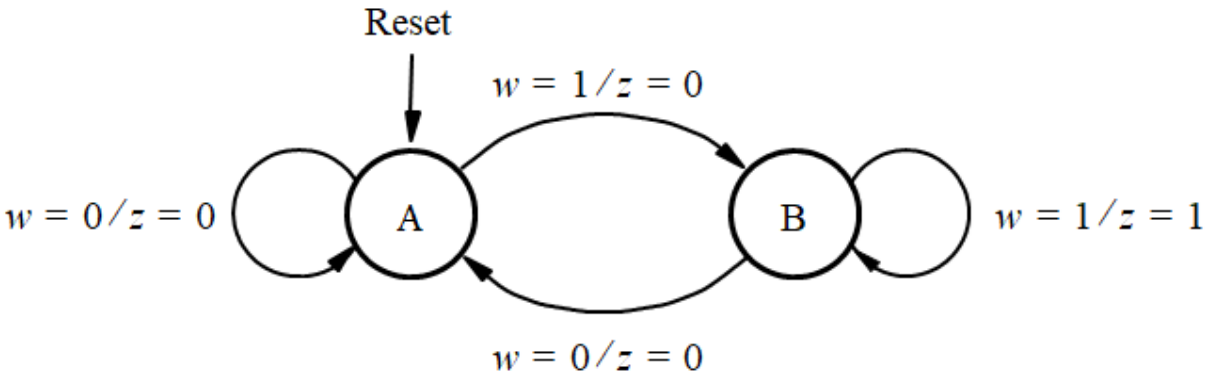
- What has changed?
  - The output z should be equal to 1 in the same clock cycle when the second occurrence of w = 1 is detected.
- How many states do we need now?

Mealy State Model

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0



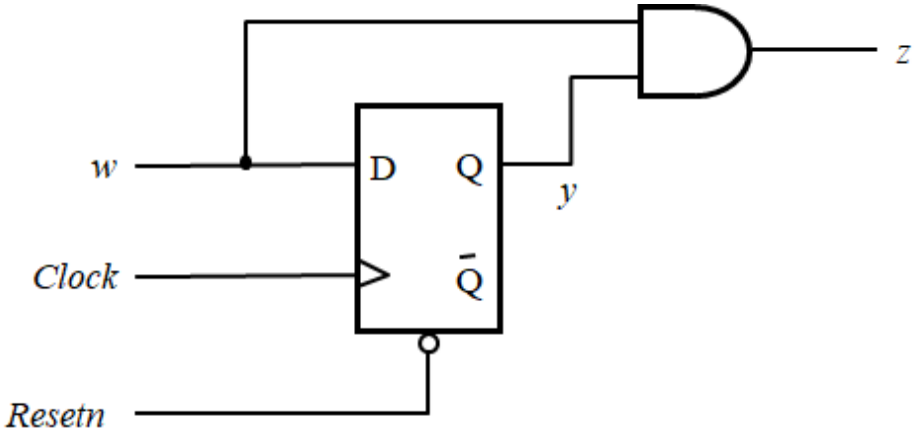
Mealy State Model: State Tables



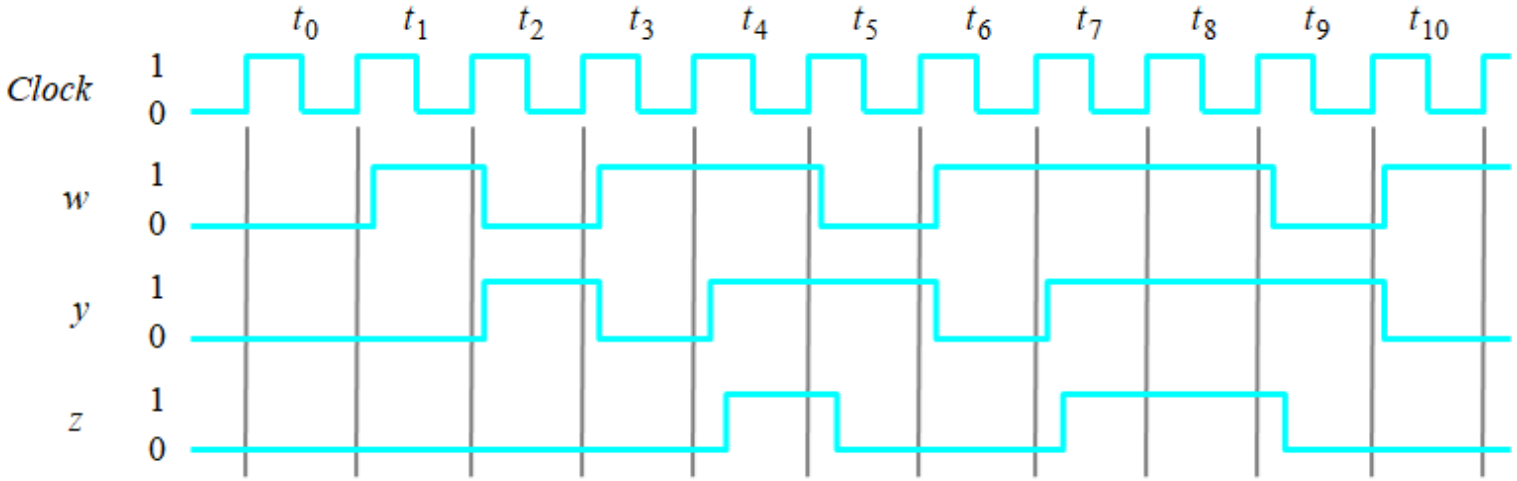
Present state	Next state		Outpu z	
	w = 0	w = 1	w = 0	w = 1
A	A	B	0	0
B	A	B	0	1

	Present state	Next state		Output	
		w = 0	w = 1	w = 0	w = 1
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Mealy State Model: Final Implementation



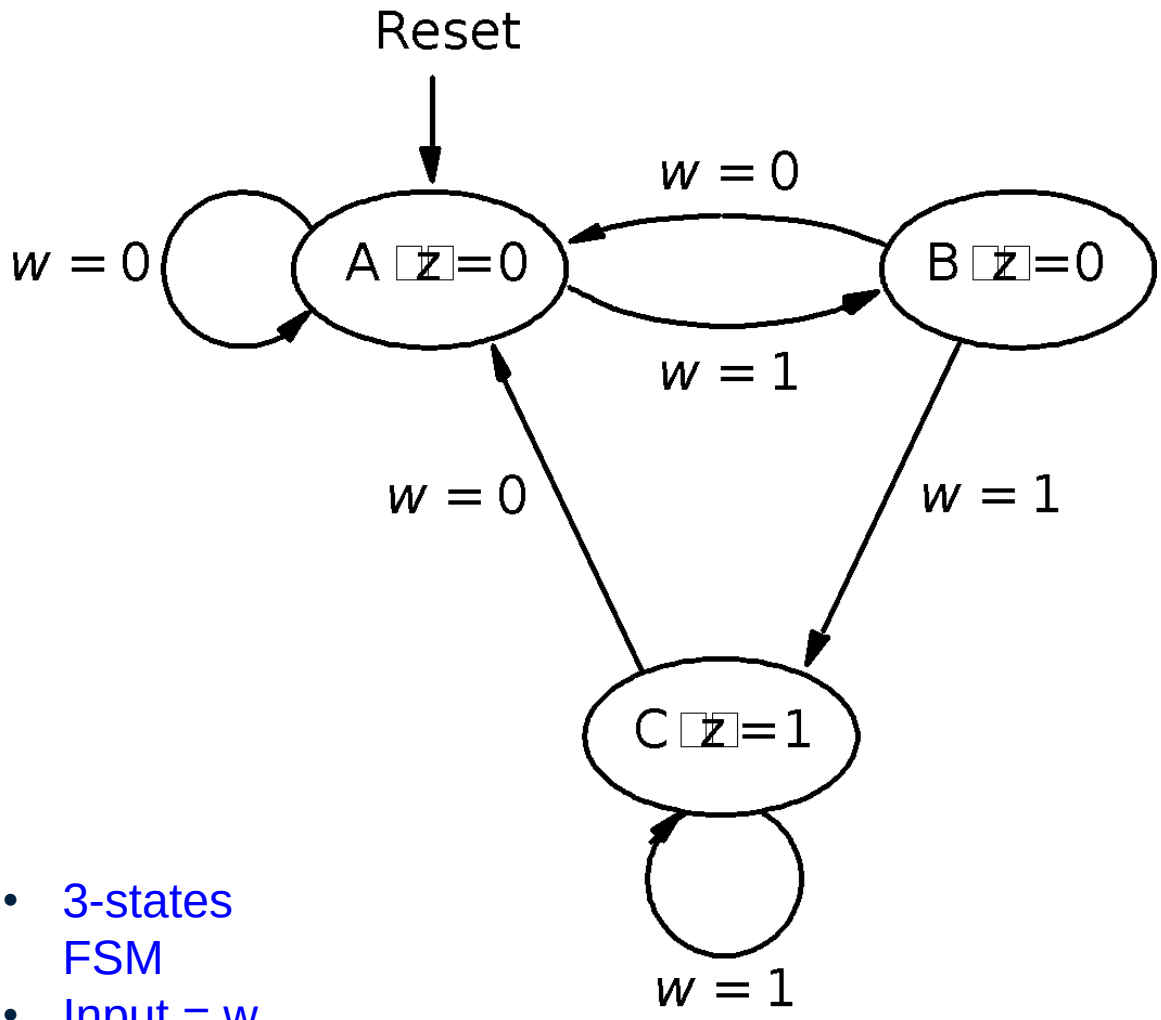
(a) Circuit



(b) Timing diagram

Expressing Finite State Machines in Verilog HDL

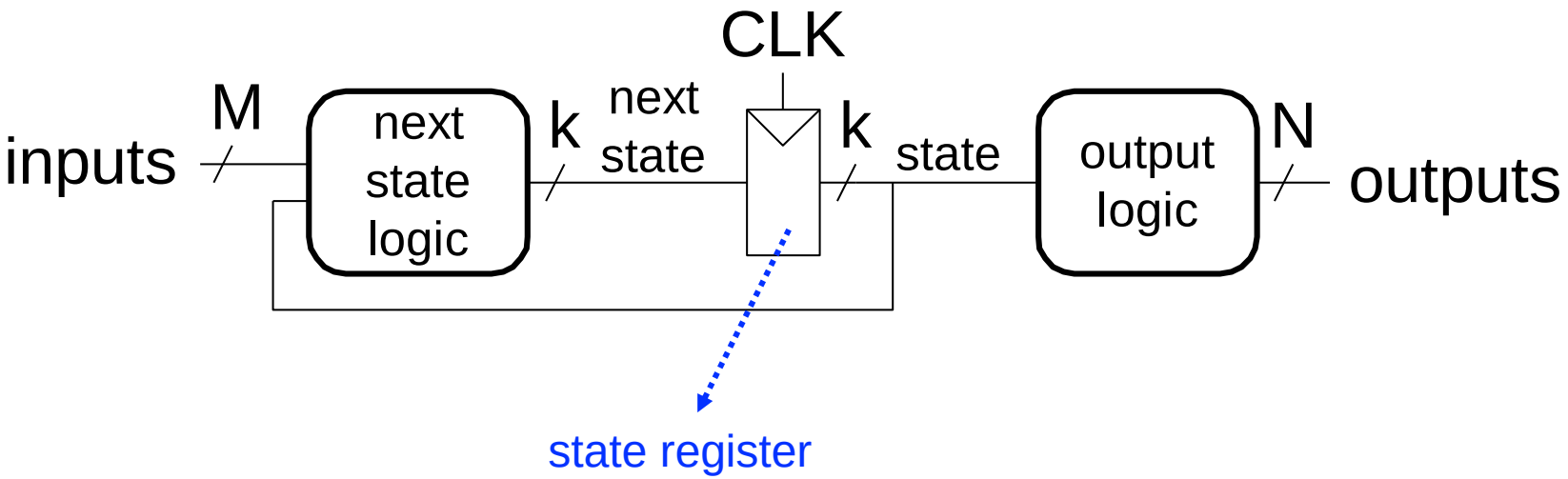
Expressing FSM in Verilog: Moore-type FSM Example



- 3-states FSM
- Input = w
- Output = z

Recall: Finite State Machines (FSMs)

- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic



Recipe for Describing FSM in Verilog

1. Define the state codes as parameters to list all states from your state diagram.
2. Declare a reg [log2(num states) wide] to store the current state (flip-flops) and next state.
3. Next-state logic: Write one always block to specify the values that next state should have for each value of present state:
   - This is a combinational process (Type-1)
   - Inputs are the current_state (outputs from the first process), and FSM Inputs
   - Use CASE to specify next state
   - Outputs are the FSM outputs
4. Present-state Advancement: Write second always block which specifies that present state is assigned the value of next state on the positive clock edge:
   - This is a sequential process (Type 2 or 3)
   - Inputs to this always block are clock and async reset (if any)
   - Output is the current state signal
5. Output Logic: Write a third always block (optional) that drives the output based on present state (and input in case of Mealy).
   - This is a combinational process (Type-1).
   - Alternatively, can be modeled through continuous statements.

```
module moore (Clock, w, Resetn, z);
  input  Clock, w, Resetn;
  output z;
1 parameter A = 2'b00, B = 2'b01, C = 2'b10;

2 reg [1:0] p_state, n_state;

3   always @(w, p_state)
    begin
      case (p_state)
        A:  if (w = = 0)  n_state = A;
            else  n_state = B;
        B:  if (w = = 0)  n_state = A;
            else  n_state = C;
        C:  if (w = = 0)  n_state = A;
            else  n_state = C;
       default: n_state = 2'bxx;
      endcase
    end

4   always @(posedge Clock, negedge Resetn)
    begin
      if (Resetn = = 0)
          p_state <= A;
      else
          p_state <= n_state;
    end

5   assign z = (p_state = = C);

endmodule
```

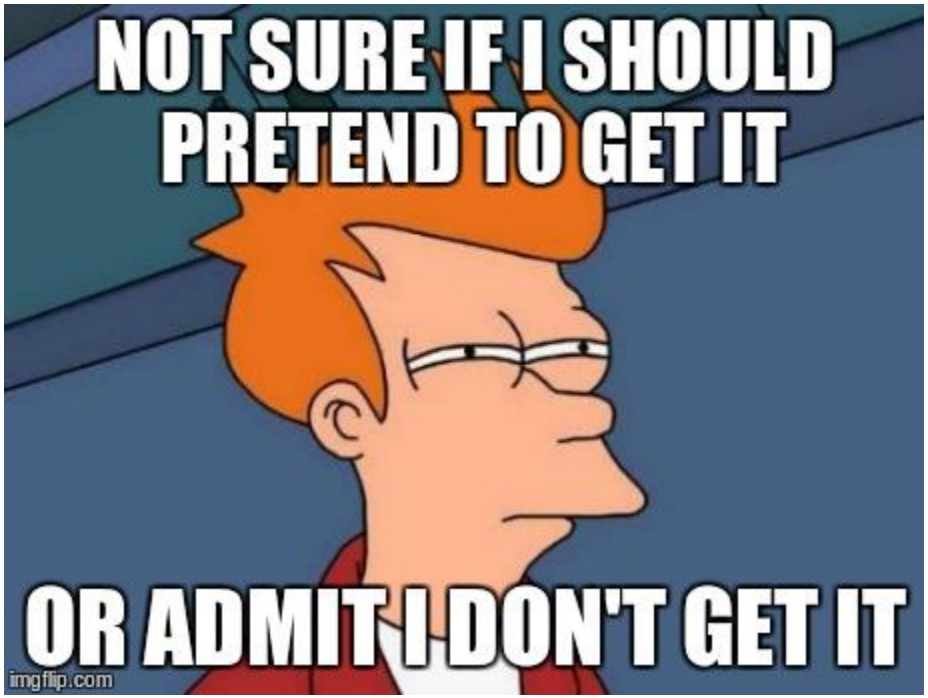
Alternate Coding Style 2 – Combining Next Sate Logic and Output Logic

```
module simple (Clock, Resetn, w, z);
        input Clock, Resetn, w;
        output reg z;
        reg [2:1] y, Y;
        parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

        // Define the next state combinational circuit
        always @(w, y)
        begin
                case (y)
                        A: if (w)       Y = B;
                           else         Y = A;
                        B: if (w)       Y = C;
                           else         Y = A;
                        C: if (w)       Y = C;
                           else         Y = A;
                        default:        Y = 2'bxx;
                endcase
                z = (y = = C);          //Define output
        end

        // Define the sequential block
        always @(negedge Resetn, posedge Clock)
                if (Resetn = = 0)  y <= A;
                else            y <= Y;

endmodule
```

Alternate Coding Style 3 – One Always block

```
module simple (Clock, Resetn, w, z);
        input Clock, Resetn, w;
        output z;
        reg [2:1] y;
        parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

        // Define the sequential block
        always @(negedge Resetn, posedge Clock)
                if (Resetn = = 0)       y <= A;
                else
                        case (y)
                                A: if (w)   y <= B;
                                           else        y <= A;
                                B: if (w)   y <= C;
                                           else        y <= A;
                                C: if (w)   y <= C;
                                           else        y <= A;
                                default:    y <= 2'bxx;
                        endcase

        // Define output
        assign z = (y = = C);

endmodule
```

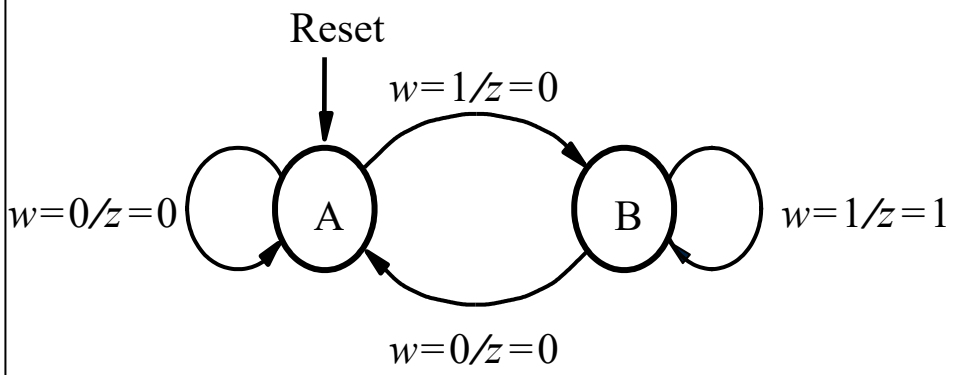
Which Coding Style do I opt for my Moore FSM?



- While all the three coding styles yield the same functionally equivalent FSM:
  - It is suggested to stick with the first style, i.e. three always blocks; one for each distinct part of FSM.
    - More closer to the circuit description
    - Easier to read and debug

Expressing Mealy FSM in Verilog

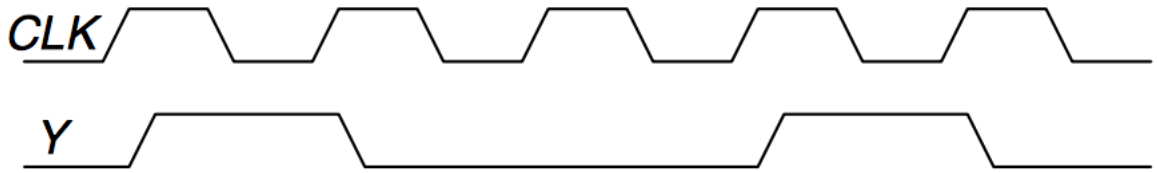
```
module mealy (Clock, Resetn, w, z);
        input Clock, Resetn, w;
        output reg z;
        reg y, Y;
        parameter A = 1'b0, B = 1'b1;

        // Define the next state and output combinational
circuits
        always @(w, y)
        case (y)
        A:  if (w)
        begin
            z = 0;
           Y = B;
        end
        else
        begin
            z = 0;
            Y = A;
        end
        B:  if (w)
        begin
             z = 1;
             Y = B;
        end
        else
        begin
        z = 0;
        Y = A;
        end
       endcase

        // Define the sequential block
        always @(negedge Resetn, posedge Clock)
                if (Resetn = = 0) y <= A;
                else              y <= Y;

endmodule
```



- Compared to Moore, the main difference is the way in which the code for the output is written.
- The output, z, is driven from within the case.
- The sensitivity list for the always block includes w (input)

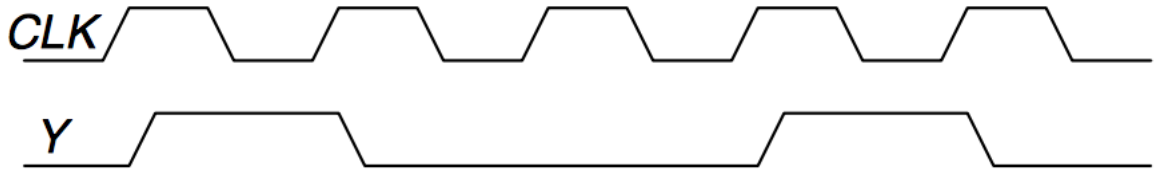
Another FSM Example

FSM Example: Divide the Clock Frequency by 3

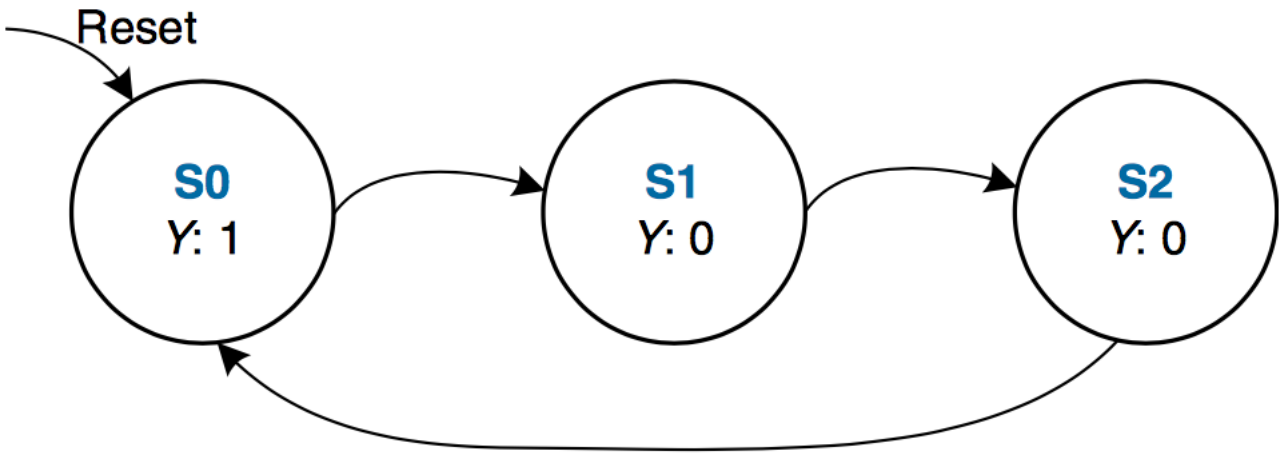


The output *Y* is HIGH for **one clock cycle out of every *3***. In other words, the output **divides the frequency of the clock by *3***.

FSM Example: Divide the Clock Frequency by 3



The output *Y* is HIGH for **one clock cycle out of every *3***. In other words, the output **divides the frequency of the clock by *3***.

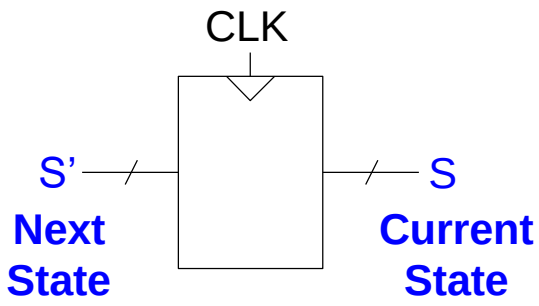


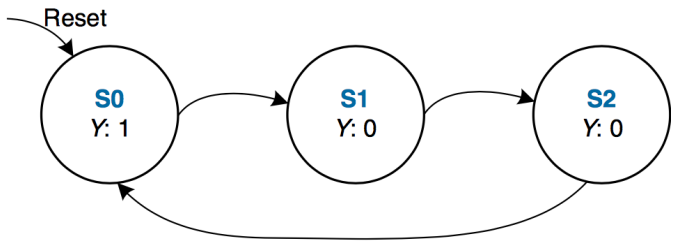
Implementing FSM Example 1: Definitions

```
module divideby3FSM (input clk,
                     input reset,
                     output q);

   reg  [1:0] state, nextstate;

   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;
```

- We define `state` and `nextstate` as 2-bit **reg**
- The parameter descriptions are optional, it makes reading easier

Implementing FSM Example 1: State Register

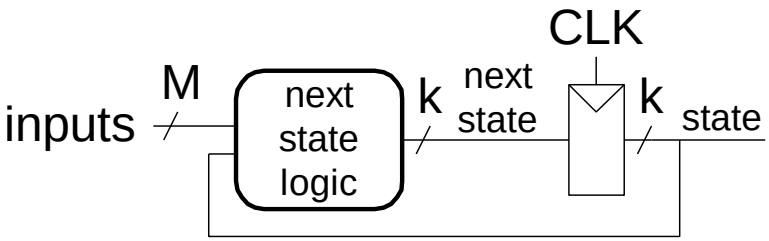


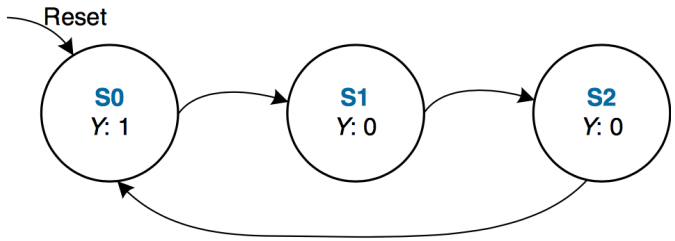



```
// state register
   always @ (posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;
```

- This part defines the state register (memorizing process)
- Sensitive to only clk, reset
- In this example, reset is active when it is '1' (active-high)

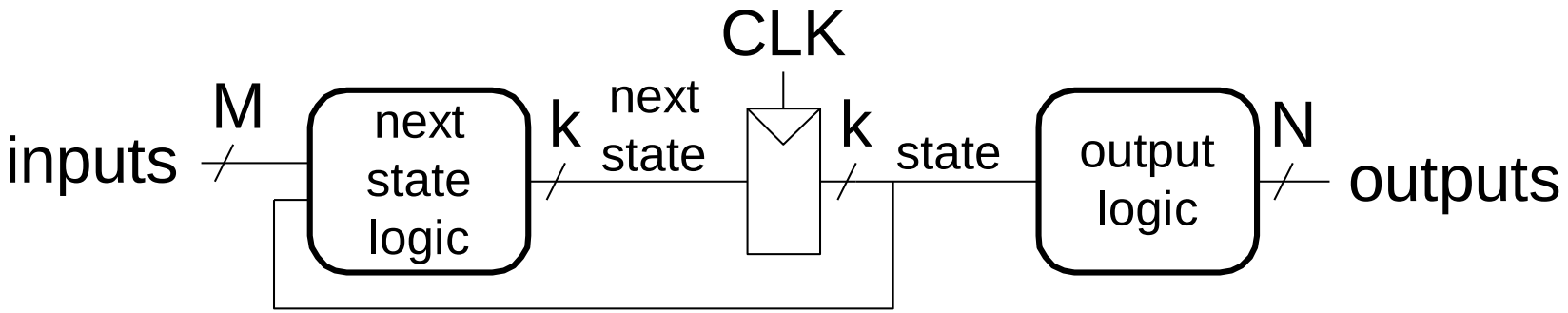
Implementing FSM Example 1: Next State Logic

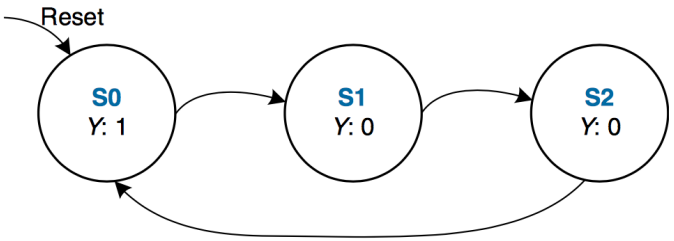





```
// next state logic
  always @ (*)
     case (state)
        S0:      nextstate = S1;
        S1:      nextstate = S2;
        S2:      nextstate = S0;
        default: nextstate = S0;
     endcase
```

Implementing FSM Example 1: Output Logic






```
// output logic
   assign q = (state == S0);
```

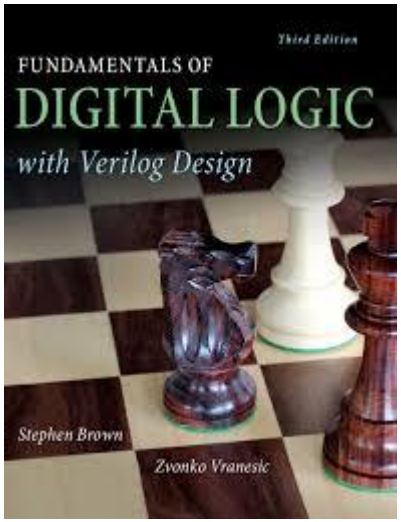
- In this example, output depends only on state
  - **Moore type FSM**

Implementation of FSM Example 1

```
module divideby3FSM (input clk, input reset, output q);
   reg  [1:0] state, nextstate;

   parameter S0 = 2'b00; parameter S1 = 2'b01; parameter S2 = 2'b10;

   always @ (posedge clk, posedge reset) // state register
      if (reset) state <= S0;
      else       state <= nextstate;

   always @ (*)                          // next state logic
      case (state)
         S0:      nextstate = S1;
         S1:      nextstate = S2;
         S2:      nextstate = S0;
         default: nextstate = S0;
      endcase
   assign q = (state == S0);             // output logic
endmodule
```

Recommended Reading

- Digital System Design with Verilog HDL, 3/e, b **S**tephen Brown and **Z**vonko Vranesic. [**S&Z**]
  - S&Z,
    - Chapter-6



THANK YOU