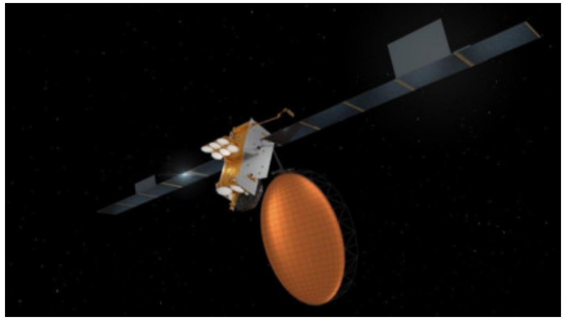
EECS151 : Introduction to Digital Design and ICs

Lecture 7 – Finite State Machines

Bora Nikolić

September 7, 2021, EETimes **5G Takes to the Stars**

Get ready to never have an excuse to be off the grid again. The latest update to the 5G New Radio (5G NR) standard will enable compatible devices to connect with 5G capable satellites anywhere in the world, without requiring specialist phones to get networked.



Artist's rendering of an Inmarsat-6 satellite, which will support 5G. (Source: Inmarsat.)

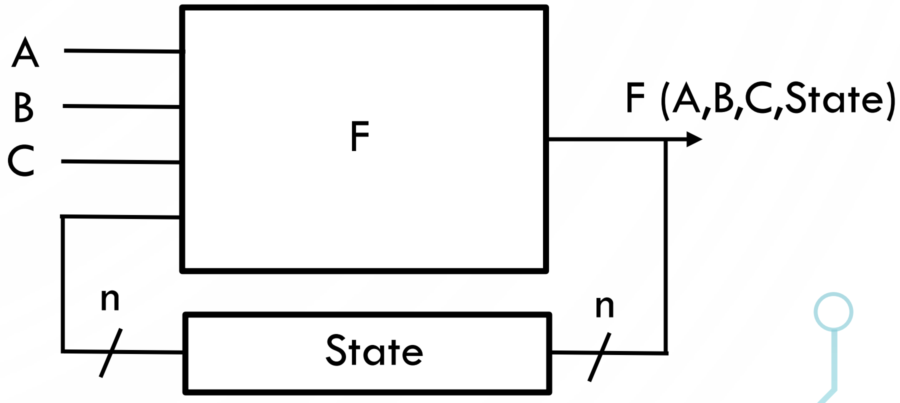
Review

- Combinational logic:
  - The outputs only depend on the current values of the inputs (memoryless)
  - The functional specification of a combinational circuit can be expressed as:
    - A truth table
    - A Boolean equation
- Boolean algebra
  - Deal with variables that are either True or False
  - Map naturally to hardware logic gates
  - Use theorems of Boolean algebra and Karnaugh maps to simplify equations
- Finite state machines: Common example of sequential logic
- Common job interview questions ☺

Finite-State Machines

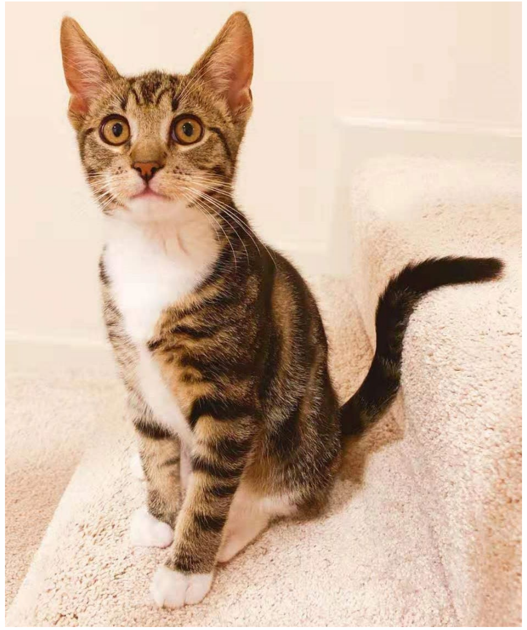
Sequential logic

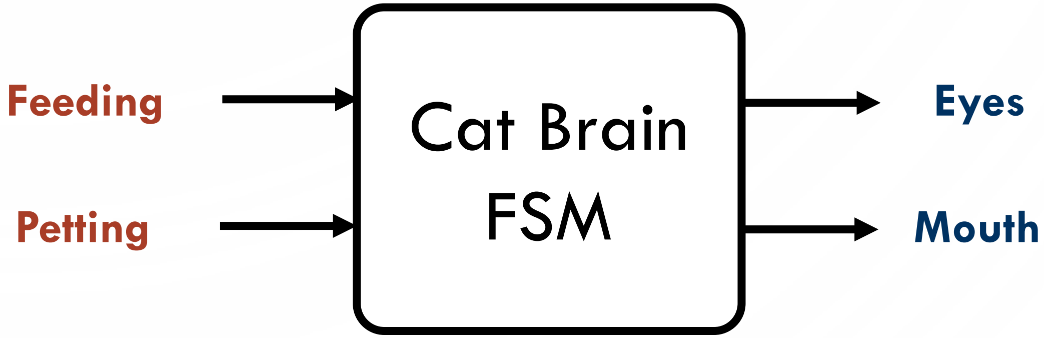
- Combinational logic:
  - Memoryless: the outputs only dependent on the current inputs.
- Sequential logic:
  - Memory: the outputs depend on both current and previous values of the inputs.
    - Distill the prior inputs into a smaller amount of information, i.e., states.
  - State: the information about a circuit
    - Influences the circuit's future behavior
    - Stored in Flip-flops and Latches
  - Finite State Machines:
    - Useful representation for designing sequential circuits
    - As with all sequential circuits: output depends on present and past inputs
    - We will first learn how to design by hand then how to implement in Verilog.



FSM Example

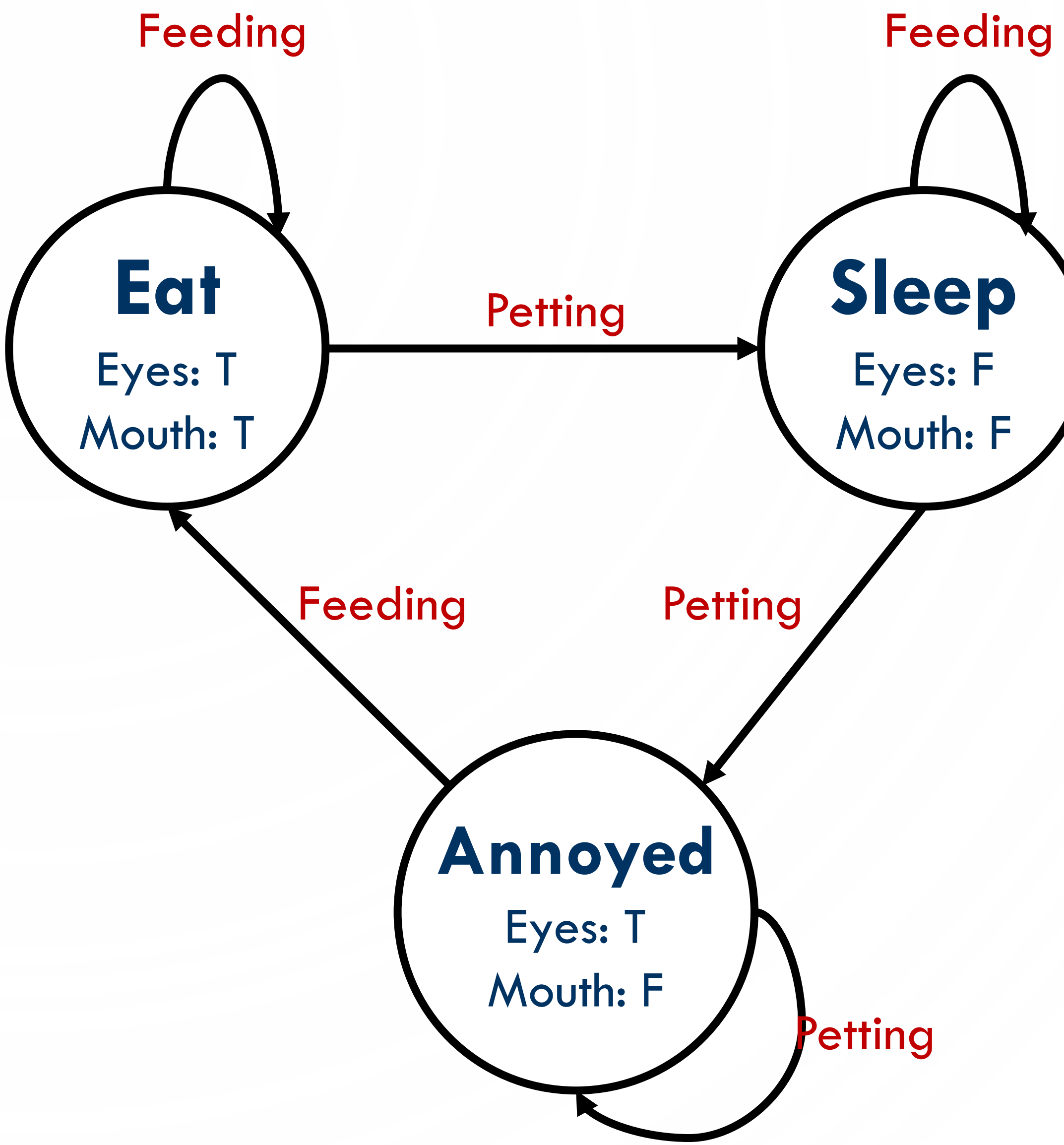
- Cat Brain (Simplified…)
  - Inputs:
    - Feeding
    - Petting
  - Outputs:
    - Eyes: open or close
    - Mouth: open or close
  - States:
    - Eating
    - Sleeping
    - Annoyed…





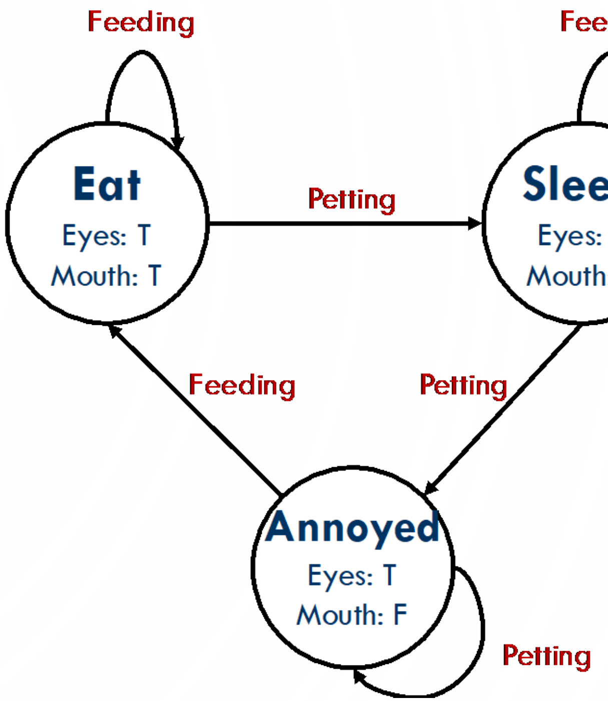
FSM State Transition Diagram

- States:
  - Circles
- Outputs:
  - Labeled in each state
  - Arcs
- Inputs:
  - Arcs



FSM Symbolic State Transition Table

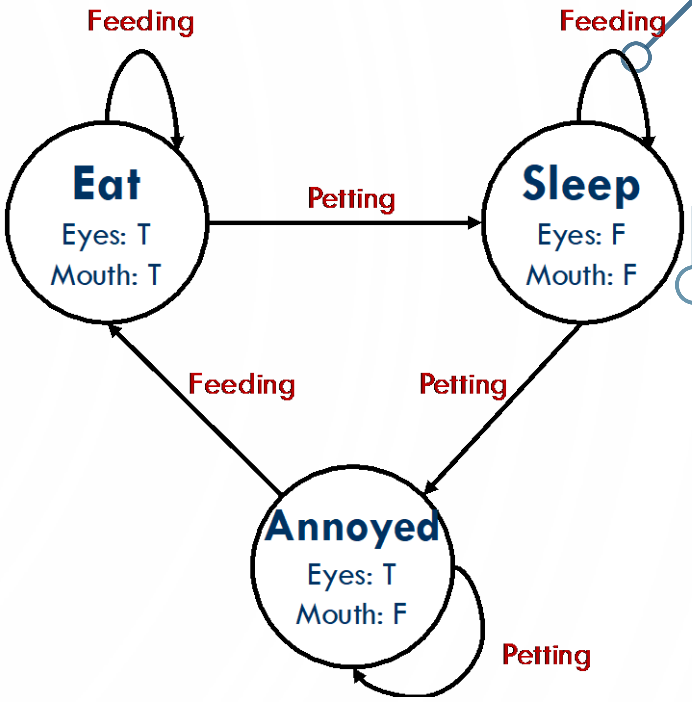
Current State	Inputs	Next State
Eat	Feeding	Eat
Eat	Petting	Sleep
Sleep	Feeding	Sleep
Sleep	Petting	Annoyed
Annoyed	Feeding	Eat
Annoyed	Petting	Annoyed



FSM Encoded State Transition Table

State	Encoding
Eat	00
Sleep	01
Annoyed	10

Current State		Input	Next State	
S1	S0	X	S1’	S0’
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0

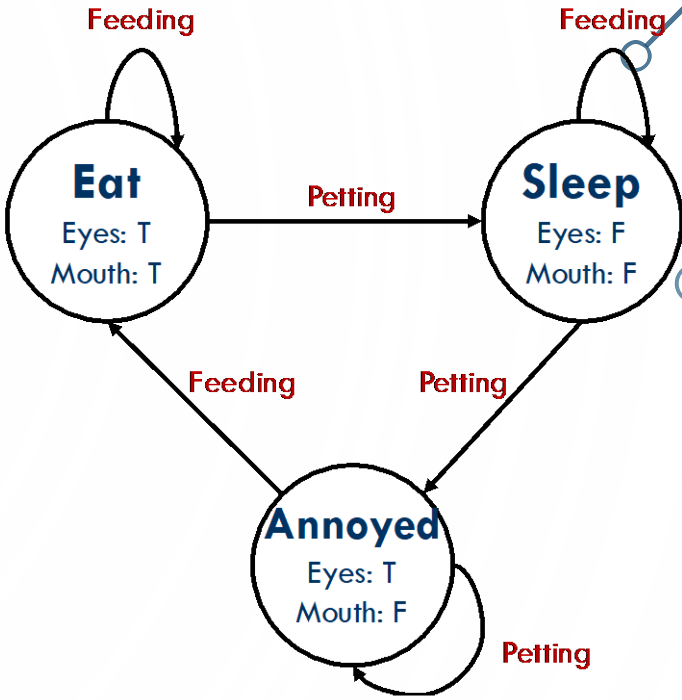


Current State	Inputs	Next State
Eat	Feeding	Eat
Eat	Petting	Sleep
Sleep	Feeding	Sleep
Sleep	Petting	Annoyed
Annoyed	Feeding	Eat
Annoyed	Petting	Annoyed

$$S0' = \overline{S1}\,\overline{S0}X + \overline{S1}S0\overline{X} = \overline{S1}\left(\overline{S0}X + S0\overline{X}\right) = \overline{S1}(S0 \oplus X)$$

$$S1' = \overline{S1}S0X + S1\overline{S0}X = (S1 \oplus S0)X$$

FSM Output Table



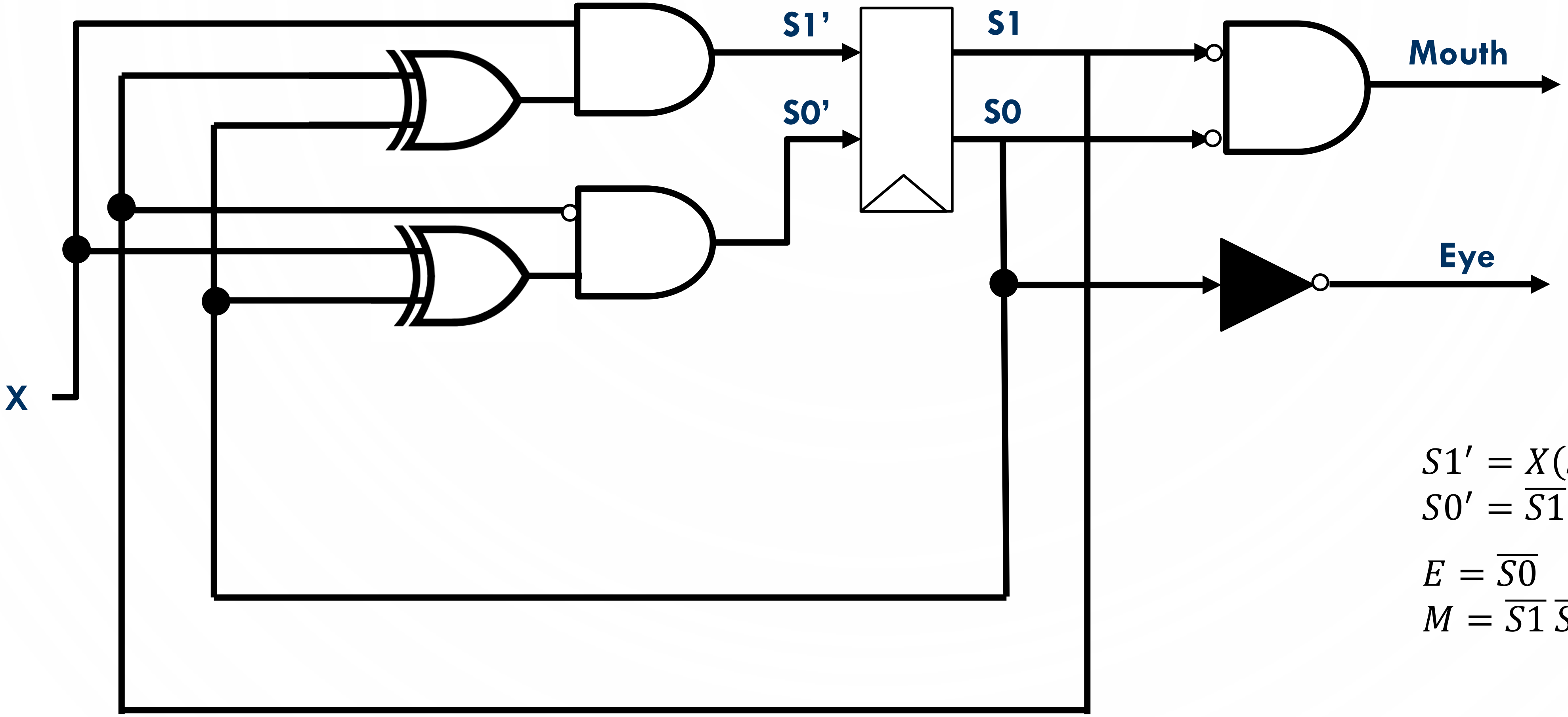
State	Encoding
Eat	00
Sleep	01
Annoyed	10

Current State		Outputs	
S1	**S0**	**E**	**M**
0	0	1	1
0	1	0	0
1	0	1	0

Outputs		Encoding
Eyes	Mouth	
Open	Open	11
Close	Close	00
Open	Close	10

$$E = \overline{S1S0} + S1\overline{S0} = \overline{S0}$$
$$M = \overline{S1}\,\overline{S0}$$

FSM Gate Representation



$$S1' = X(S0 \oplus S1)$$
$$S0' = \overline{S1}(S0 \oplus X)$$

$$E = \overline{S0}$$
$$M = \overline{S1}\,\overline{S0}$$

FSM Design Process

- Specify circuit function
- Draw state transition diagram
- Write down symbolic state transition table
- Write down encoded state transition table
- Derive logic equations
- Derive circuit diagram
  - Register to hold state
  - Combinational logic for next state and outputs

FSM State Encoding

- Binary encoding:
  - i.e., for four states, 00, 01, 10, 11
- One-hot encoding
  - One state bit per state
  - Only one state bit TRUE at once
  - i.e., for four states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic can be simpler

Administrivia

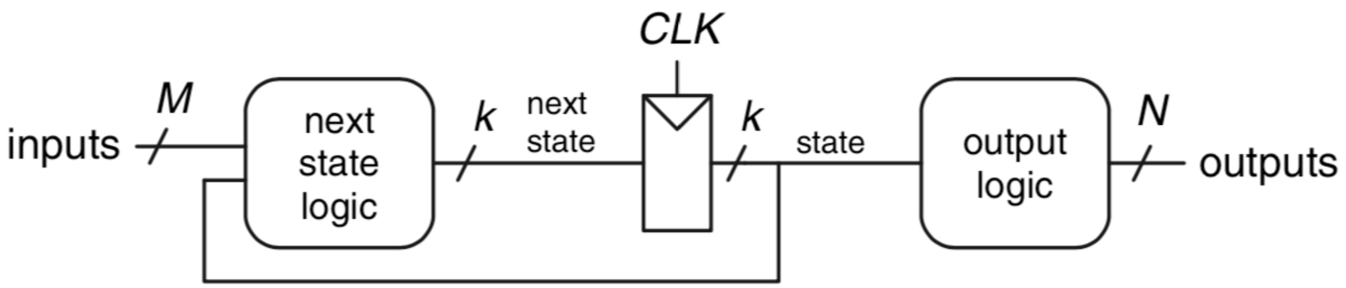
- Homework 3 is due next Monday
  - Homework 4 will be posted this week, due before midterm 1
- Lab 4 this week
- Lab 5 next week
- Midterm 1 on October 7, 7-8:30pm

Moore and Mealy FSMs

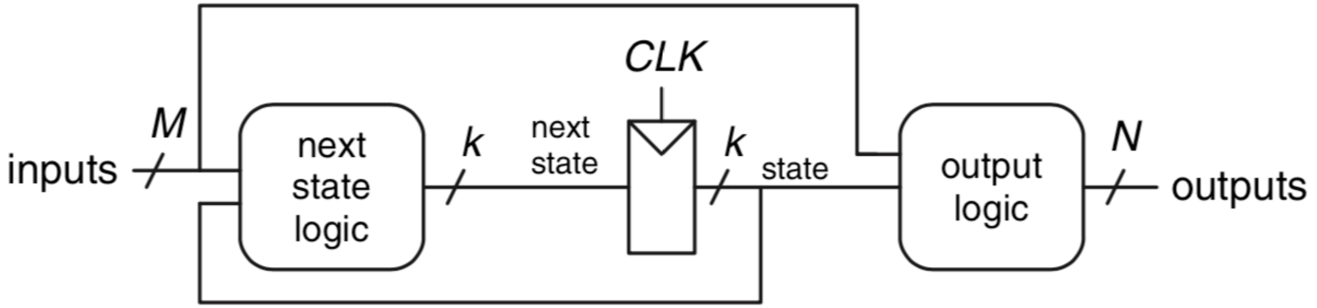
Moore's vs Mealy's FSMs

- Next state is always determined by current state and inputs
- Differ in output logic:
  - Moore FSM: outputs depend only on current state
  - Mealy FSM: outputs depend on current state and inputs

Moore FSM

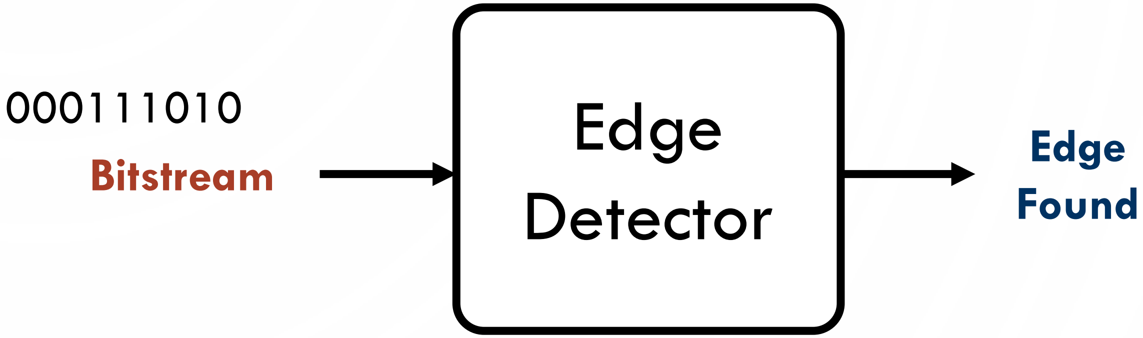


Mealy FSM

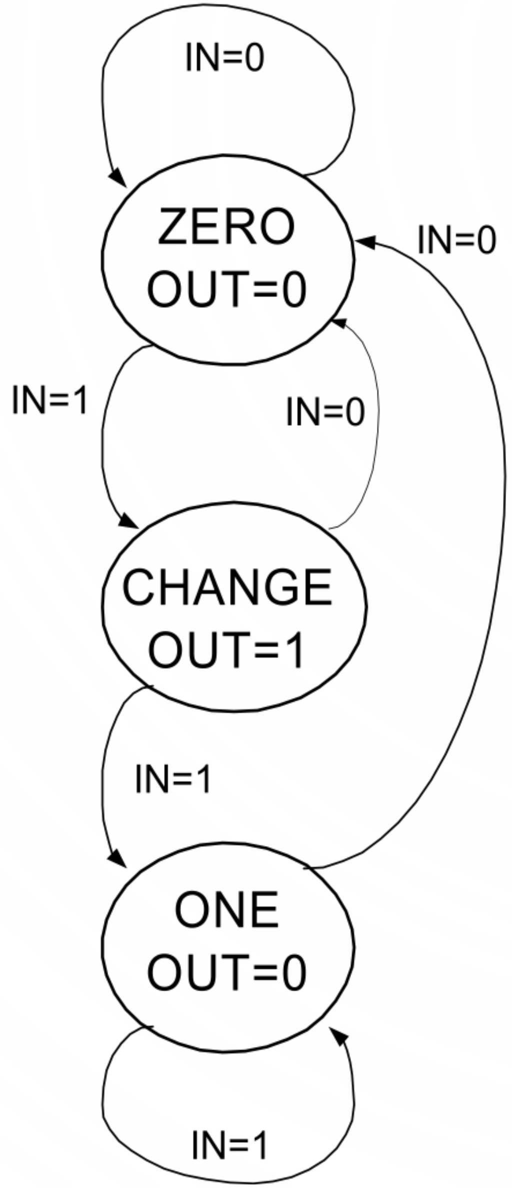


Example: Edge Detector

- Input:
  - A bit stream that is received one bit at a time.
- Output:
  - 0/1
- Circuit:
  - Asserts its output to be true when the input bit stream changes from 0 to 1.

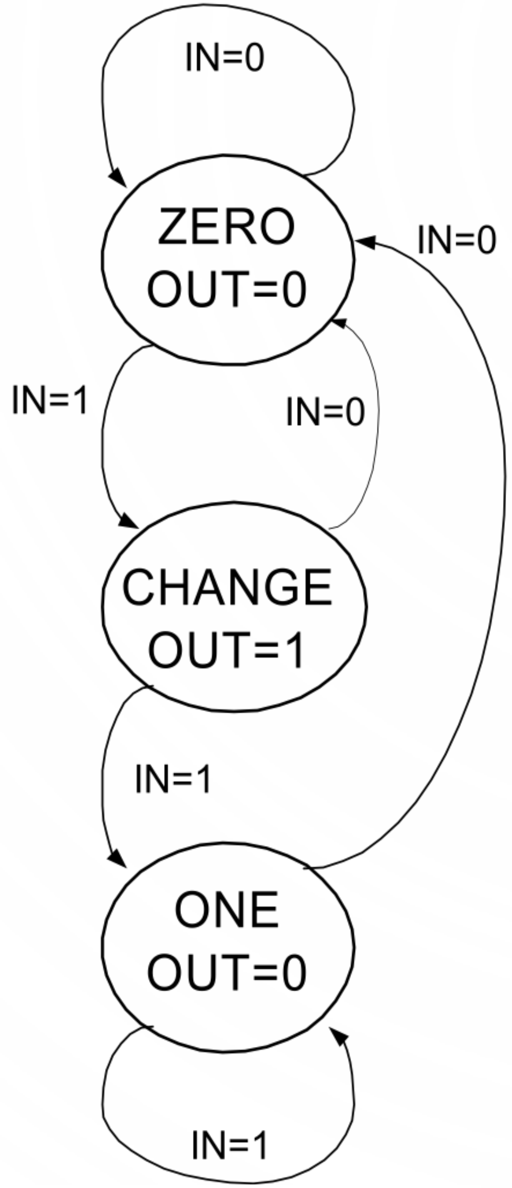


State Transition Diagram Solution A



Input	Current State	Next State	Output
0	Zero (00)	Zero	0

State Transition Diagram Solution A



Input	Current State	Next State	Output
0	Zero (00)	Zero	0
1	Zero (00)	Change	0
0	Change (01)	Zero	1
1	Change (01)	One	1
0	One (11)	Zero	0
1	One (11)	One	0

State Transition Diagram Solution A

CS = {CS1, CS0}

IN \ CS	00	01	11	10
0	0	0	0	-
1	0	1	1	-

NS_1 = IN *AND* CS0

CS

IN \ CS	00	01	11	10
0	0	0	0	-
1	1	1	1	-

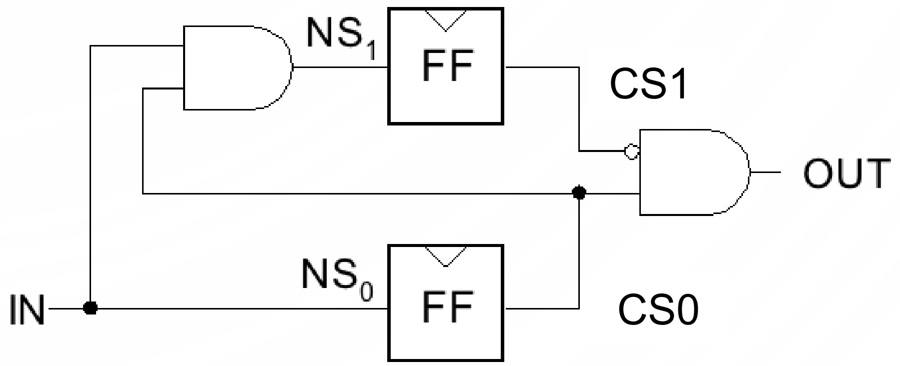
NS_0 = IN

CS

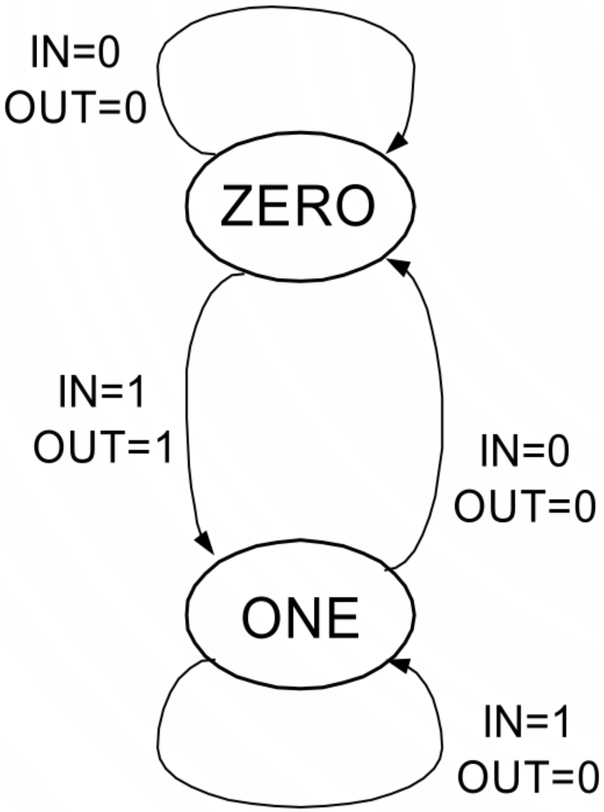
IN \ CS	00	01	11	10
0	0	1	0	-
1	0	1	0	-

OUT= *NOT* (CS1) *AND* CS0

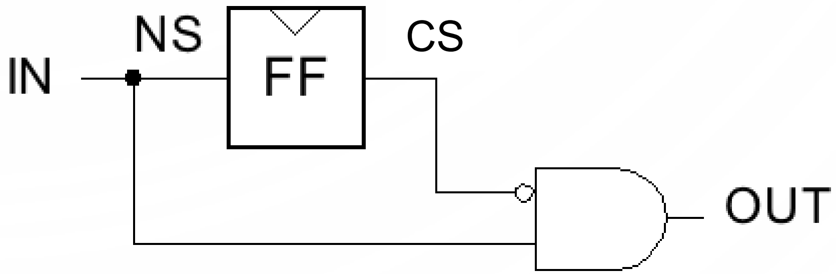
Input	Current State	Next State	Output
0	Zero (00)	Zero	0
1	Zero (00)	Change	0
0	Change (01)	Zero	1
1	Change (01)	One	1
0	One (11)	Zero	0
1	One (11)	One	0



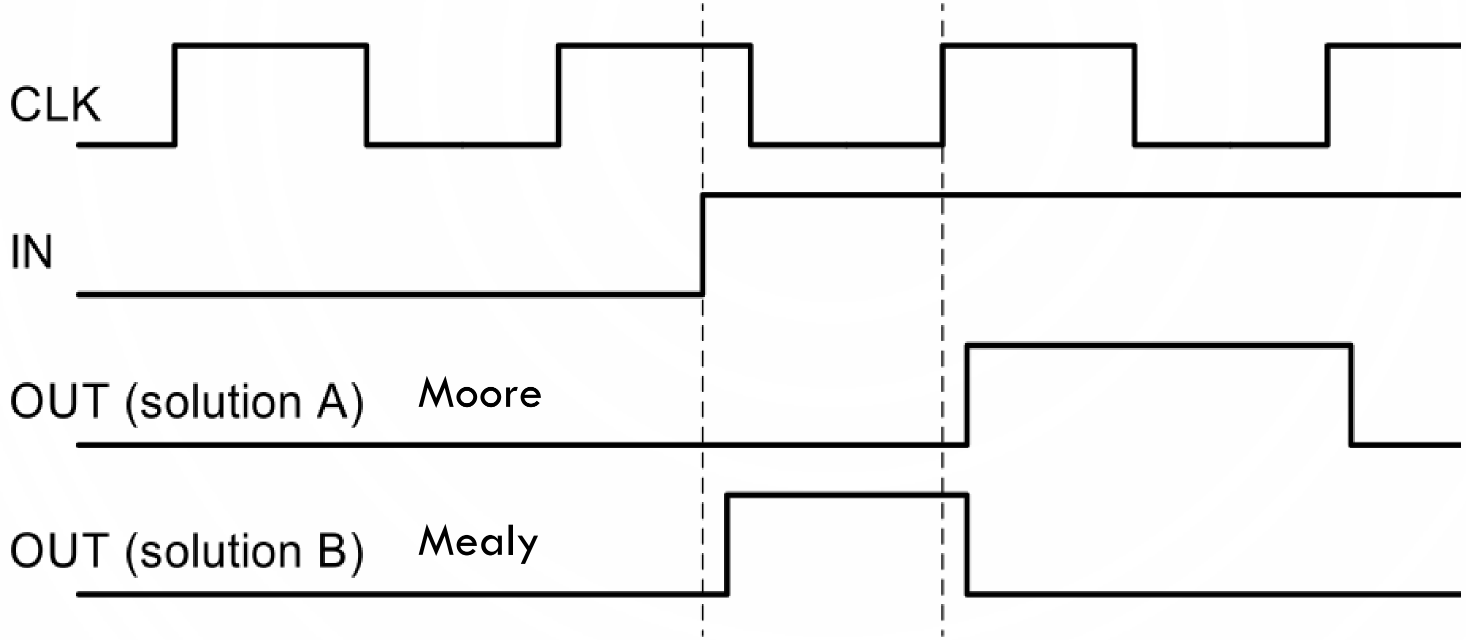
State Transition Diagram Solution B



Input	Current State	Next State	Output
0	Zero (0)	Zero	0
1	Zero (0)	One	1
0	One (1)	Zero	0
1	One (1)	One	0



Edge Detection Timing Diagrams



- Solution A (Moore) : both edges of output follow the clock
- Solution B (Mealy) : output rises with input rising edge and is asynchronous wrt the clock, output fails synchronous with next clock edge

FSM Comparison

Solution A

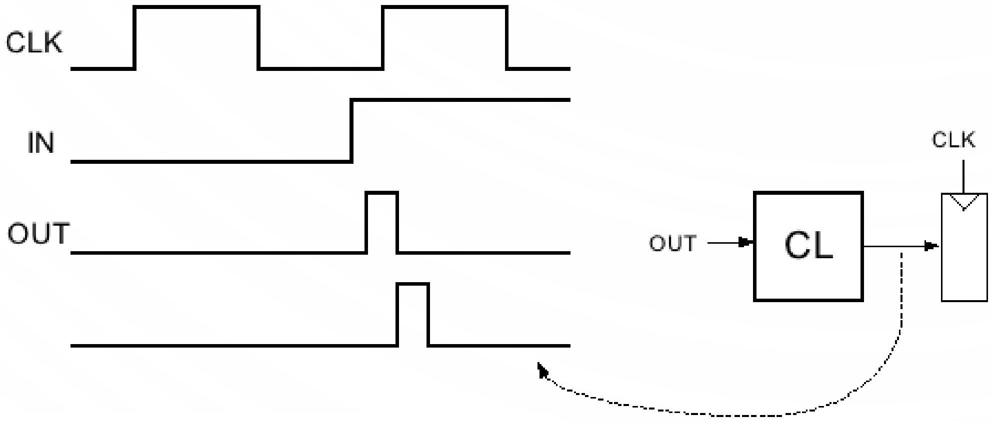
Moore Machine

- output function only of current state
- maybe <u>more</u> states (why?)
- synchronous outputs
  - Input glitches not sent to output
  - one cycle "delay"
  - full cycle of stable output

Solution B

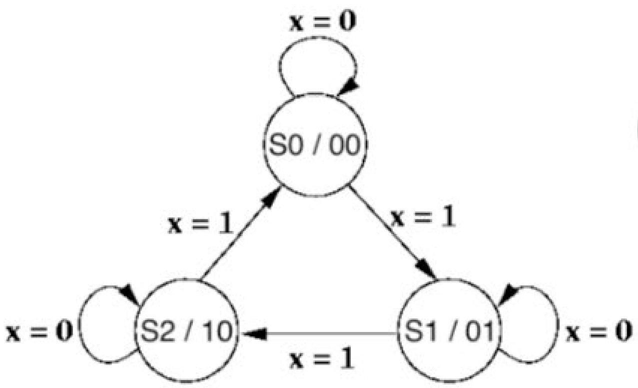
Mealy Machine

- output function of both current = & input
- maybe fewer states
- asynchronous outputs
- – if input glitches, so does output
- – output immediately available
- – output may not be stable long enough to be useful (below):

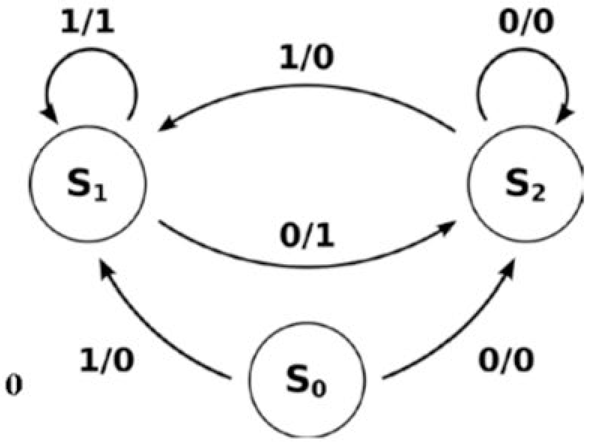


If output of Mealy FSM goes through combinational logic before being registered, the CL might delay the signal and it could be missed by the clock edge (or violate setup time requirement)

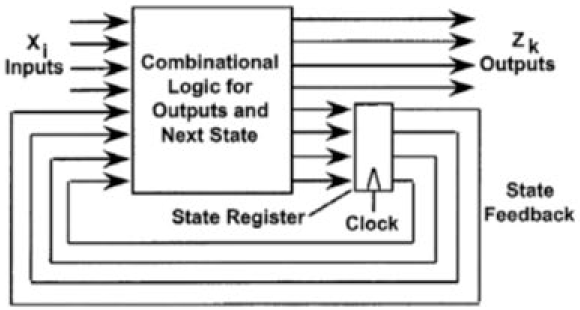
Quiz: Which of the diagrams are **Moore** machines?



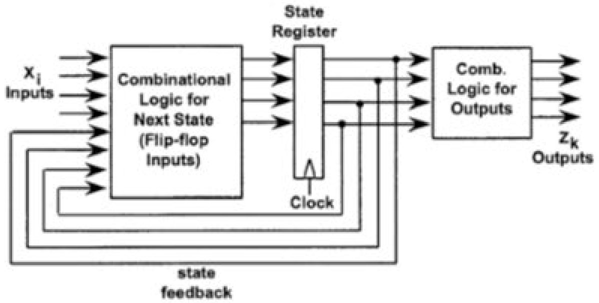
A.



B.



C.



D.

A. AC
B. BD
C. AD
D. BC

www.yellkey.com/oil

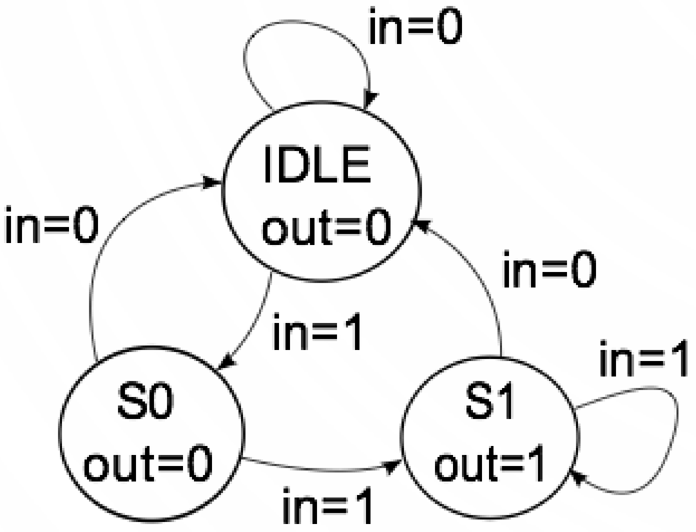
FSMs in Verilog

Implement FSM with Verilog

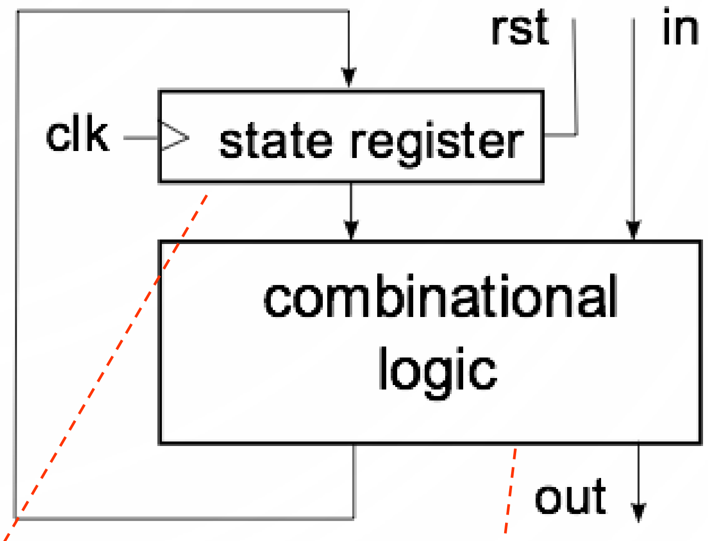
- Specify circuit function
- Draw state transition diagram
- Write down symbolic state transition table
- Assign encodings (bit patterns) to symbolic states
- Code as Verilog behavioral description
  - Use parameters to represent encoded states
  - Use separate always blocks for register assignment and combinational logic block
  - Use case statement for combinational logic.
    - Within each case section (state), assign outputs and next state based on inputs
    - Moore: outputs only dependent on states not on inputs

Finite State Machine in Verilog

State Transition Diagram



Circuit Diagram



Holds a symbol to keep track of which bubble the FSM is in.

CL functions to determine output value and next state based on input and current state.

out = f(in, current state)

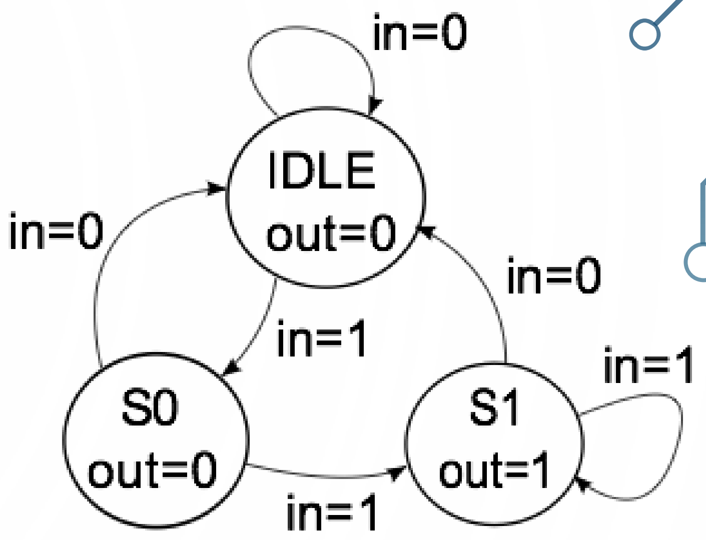
next state = f(in, current state)

Finite State Machine in Verilog

```
module FSM1(clk, rst, in, out);
input clk, rst;
input in;
output out;
```

Must use reset to force to initial state.

reset not always shown in STD




```
// Defined state encoding:
parameter IDLE = 2'b00;
parameter S0 = 2'b01;
parameter S1 = 2'b10;
reg out;
reg [1:0] current_state, next_state;
```

Constants local to this module.

Combinational logic signals for transition.

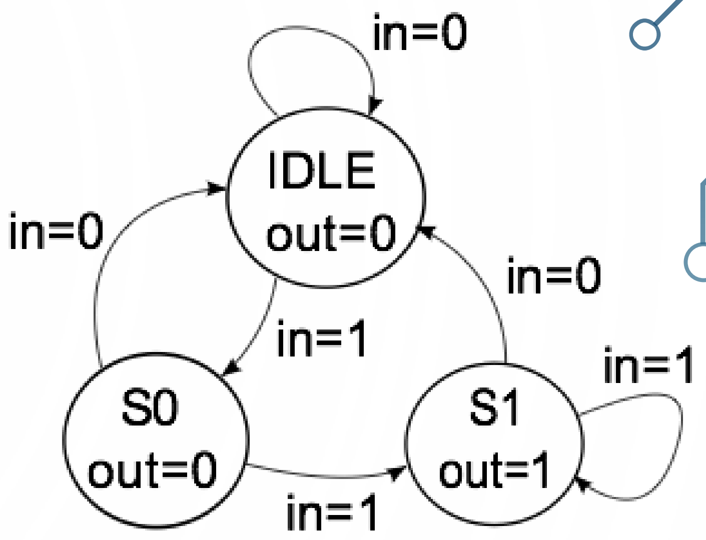
THE register to hold the "state" of the FSM.

```
// always block for state register
always @(posedge clk)
   if (rst) current_state <= IDLE;
   else current_state <= next_state;
```

A separate always block should be used for combination logic part of FSM. Next state and output generation. (Always blocks in a design work in parallel.)

Finite State Machine in Verilog (cont.)

```
// always block for combinational logic portion
always @(current_state or in)
case (current_state)
// For each state def output and next
  IDLE    : begin
             out = 1'b0;
             if (in == 1'b1) next_state = S0;
             else next_state = IDLE;
           end
  S0      : begin
             out = 1'b0;
             if (in == 1'b1) next_state = S1;
             else next_state = IDLE;
           end
  S1      : begin
             out = 1'b1;
             if (in == 1'b1) next_state = S1;
             else next_state = IDLE;
         end
  default: begin
           next_state = IDLE;
           out = 1'b0;
         end
endcase
endmodule
```



Each state becomes a case clause.

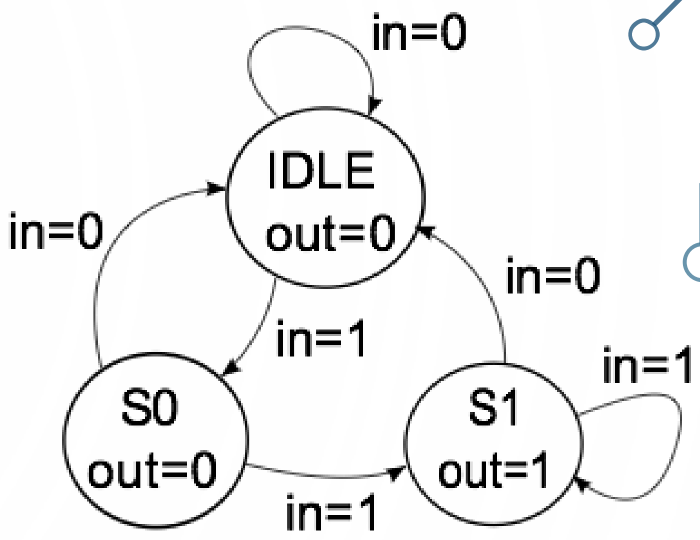
For each state define:

Output value(s)

State transition

Use "default" to cover unassigned state. Usually unconditionally transition to reset state.

Finite State Machine in Verilog (cont.)




```
always @*
  begin
   next_state = IDLE;
   out = 1'b0;
   case (state)
    IDLE    : if (in == 1'b1) next_state = S0;
    S0      : if (in == 1'b1) next_state = S1;
    S1      : begin
               out = 1'b1;
               if (in == 1'b1) next_state = S1;
              end
    default: ;
   endcase
  end
endmodule
```

* for sensitivity list

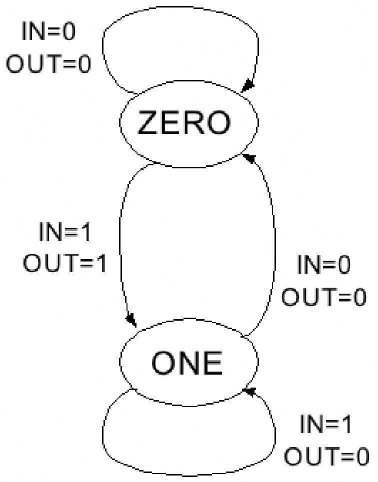
Nominal values: used unless specified below.

Within case only need to specify exceptions to the nominal values.

Note: The use of "blocking assignments" allow signal values to be "rewritten" (evaluated immediately), simplifying the specification.

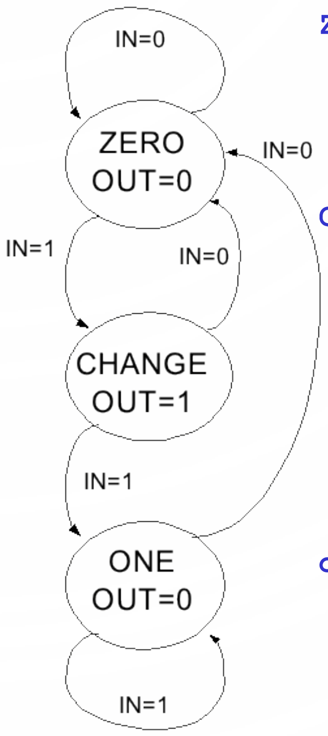
Edge Detector Example

Mealy Machine




```
always @(posedge clk)
     if (rst) ps <= ZERO;
     else ps <= ns;
always @(ps in)
    case (ps)
      ZERO: if (in) begin
               out = 1'b1;
               ns = ONE;
            end
        else begin
          out = 1'b0;
          ns = ZERO;
        end
      ONE: if (in) begin
        out = 1'b0;
        ns = ONE;
       end
       else begin
         out = 1'b0;
         ns = ZERO;
       end
      default: begin
           out = 1'bx;
           ns = default;
         end
```

Moore Machine




```
always @(posedge clk)
     if (rst) ps <= ZERO;
     else ps <= ns;
always @(ps in)
    case (ps)
      ZERO: begin
         out = 1'b0;
         if (in) ns = CHANGE;
                else ns = ZERO;
       end
      CHANGE: begin
          out = 1'b1;
          if (in) ns = ONE;
          else ns = ZERO;
         end
        ONE: begin
          out = 1'b0;
          if (in) ns = ONE;
          else ns = ZERO;
      default: begin
          out = 1'bx;
          ns = default;
         end
```

Summary

- Finite state machines: Common example of sequential logic
  - Moore's machine: Output depends only on the current state
  - Mealy's machine: Output depends on the current state and the input
- Large state machines can be factored
- Common Verilog patterns for FSMs
- Common job interview questions ☺