Lecture 12: Finite State Machines

UCSD ECE 111

Prof. Farinaz Koushanfar

Fall 2024

Some slides are courtesy of Prof. Lin or Prof. Karna ECE111 courses

Important announcements (Details in Canvas)

- **Prof office hours this week:** Tues 11/5 from 12-2pm or by email appointment
- **TA office hours:** are now MWF 9-11am for Fall'24
  - Zoom Meeting ID: 948 6397 0932; Passcode 004453
  - https://ucsd.zoom.us/j/94863970932?pwd=iXcQXbLYjOaAQTdOJlrmglCYMSyeir.1
- **TA Discussion session:** Wed 4-4:50PM (in person) Location: CNTRE 214
- **Nov 5:** Homework 6 posted on Canvas
  - Due on Wed Nov 6, **11/13/24**
  - **Late homework policy:** Max of 2 late days, with 20% grade reduction per day

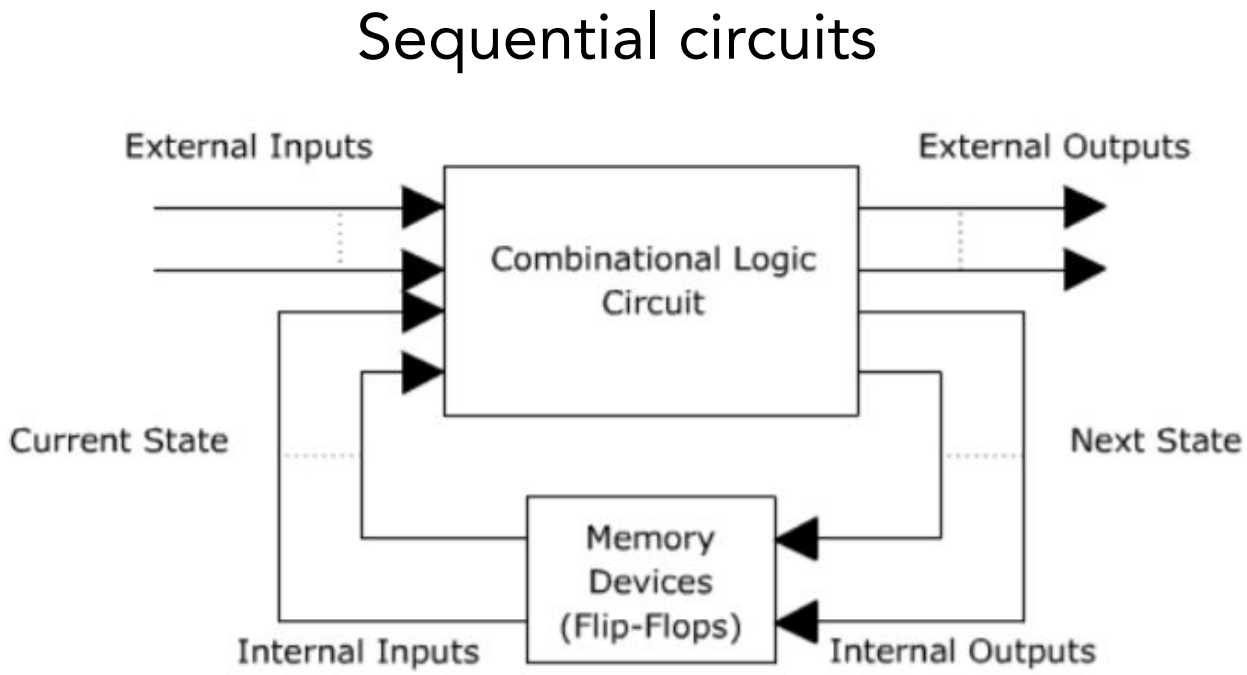
Homework 6 overview

- Design of a synthesizable SystemVerilog code for a Vending Machine (Moore and Mealy), and Synchronous FIFO, and design a system for UART communication protocol by implementing Rx-Tx communication system and a UART controller.
- You will learn how to:
  - Create synthesizable SystemVerilog code
  - Better learn how to use testbenches
  - Design functional SystemVerilog code that can compile post synthesis.
- There will be two parts for this homework:
  - **Homework-6a**: Developing a Synthesizable SystemVerilog model for a of a Vending Machine (Moore and Mealy FSM).
  - **Homework-6b**: Developing a synthesizable SystemVerilog model of a synchronous FIFO.
  - **Homework-6c:** Developing a synthesizable SystemVerilog model of a UART communication system with a Receiver, Transmitter, and a UART controller.

Sequential Circuits

- **Sequential circuits** produce output based on **current** input and **previous** input values
  - as opposed to **combinational** circuits where output depends on the current inputs only

- Sequential circuits require **memory devices** to store "**state**" information and can be further classified into two types:
  - **Asynchronous:** No clock signal; state transitions immediately on input signal change
  - **Synchronous:** Uses a clock signal that governs when state transition occurs



Finite State Machines (FSMs)

- **Finite State Machine (FSM)** is a model of computation used to design **sequential logic circuits**
  1. Finite number of **states**
  2. The machine is in only **one state at a time**; the state it is in at any given time is called the **current or present state**
  3. It can **change** from one **state** to another when initiated by a **triggering event or condition**. This is called a **transition.**
  4. A particular FSM is defined by a list of its states, initial state and the triggering condition for each transition.
  5. Can be represented using a **state table** or **state transition diagram**
  6. It can be implemented using models like **Mealy** and **Moore** machine

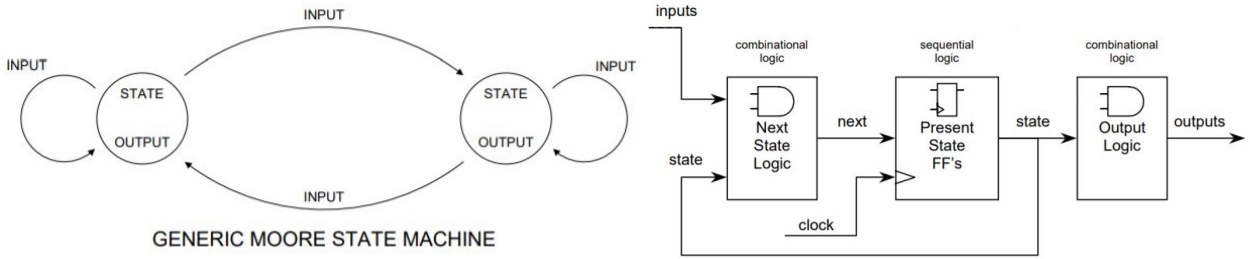
More and Mealy Finite State Machines

Moore machine

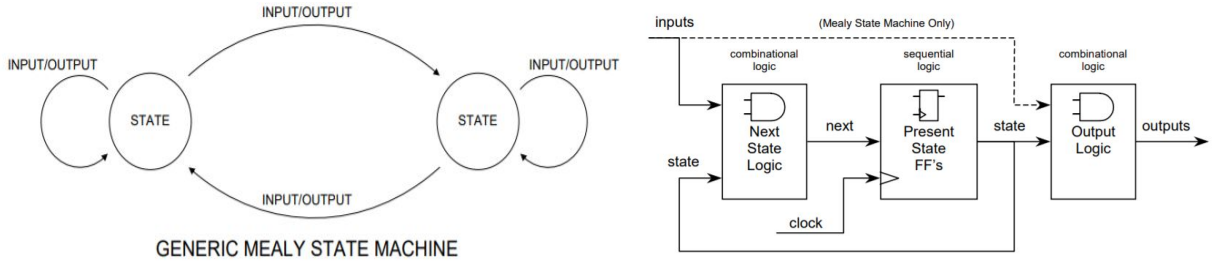
- Output is solely based on **present state** of FSM
- Output is associated **with** a state
- Generally, **more states** than Mealy
- Output **react slower** to input (next clock cycle)
- **Synchronous** output and state generation

Mealy machine

- Output based on **present state and input(s)**
- Output changes during **transition** of states
- Generally, **less states** than Moore
- **Reacts faster** to inputs in the same cycle
- **Asynchronous** output generation



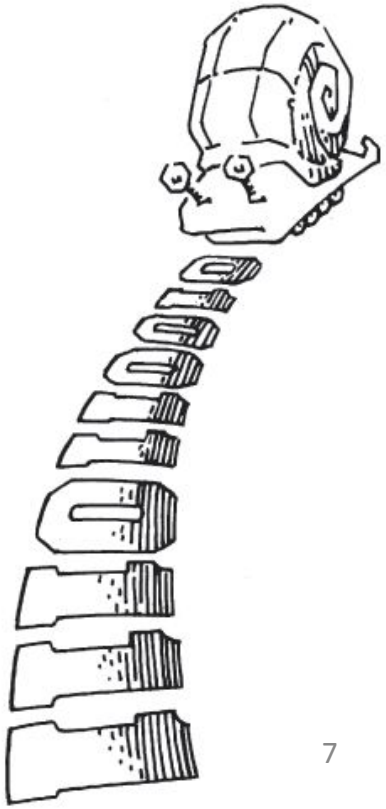
GENERIC MOORE STATE MACHINE



GENERIC MEALY STATE MACHINE

Mealy and Moore FSM

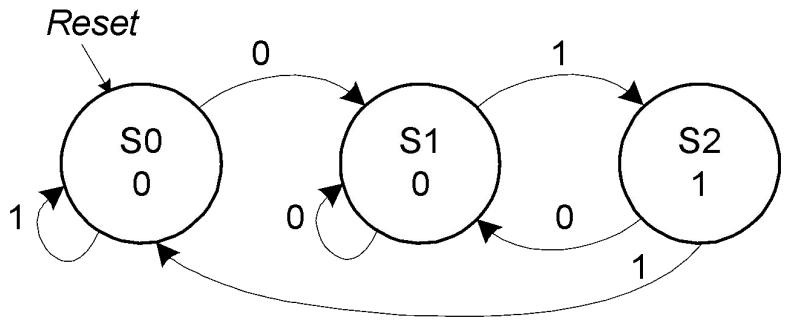
- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.



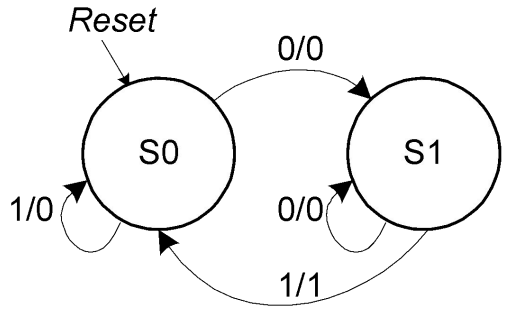
Slide Courtesy of Harris&Harris Book

Mealy and Moore FSMs

Moore FSM



Mealy FSM



Mealy FSM: arcs indicate input/output

Moore FSM state transition table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		

State	Encoding
S0	00
S1	01
S2	10

Slide Courtesy of Harris&Harris Book

Moore FSM State Transition Table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S_1' = S_0 A$$
$$S_0' = A$$

Moore output table

Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

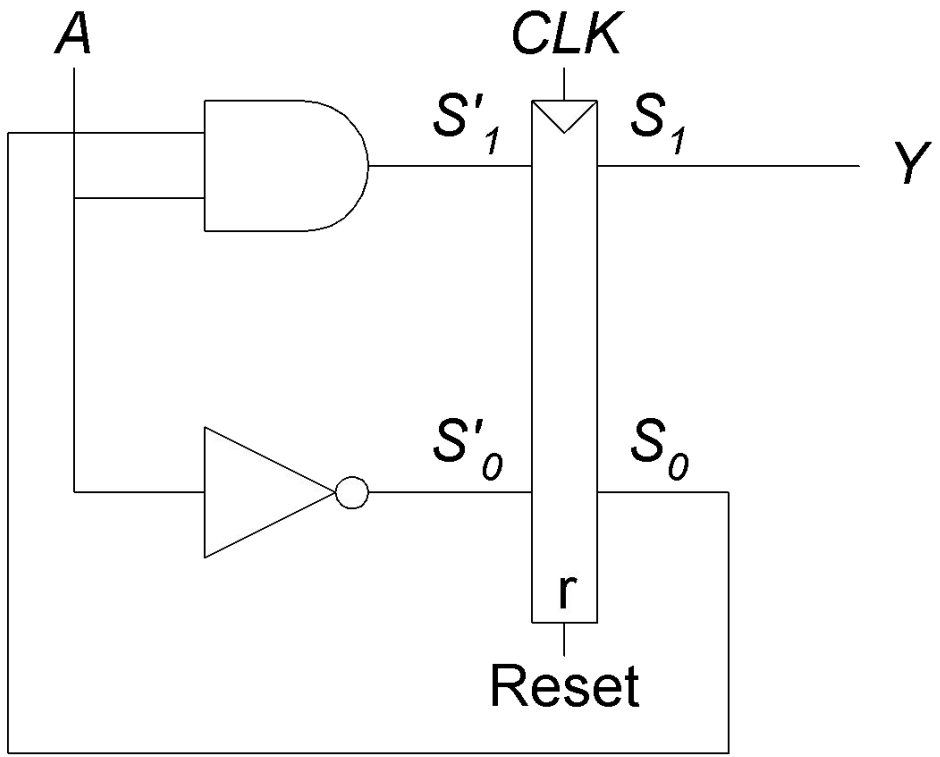
$$Y = S_1$$

Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

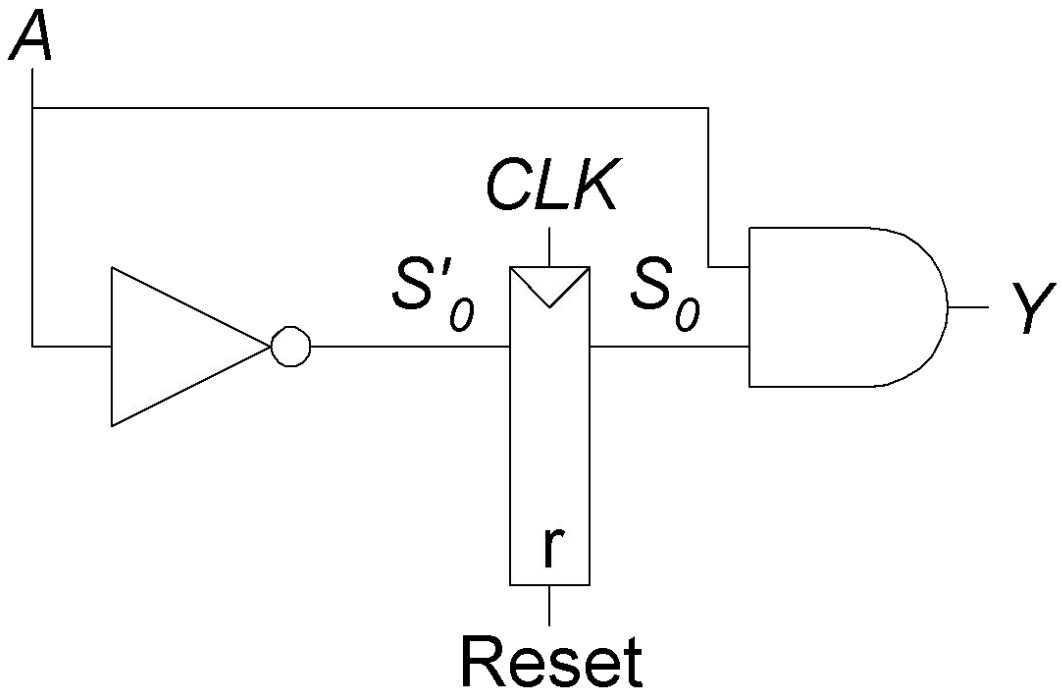
State	Encoding
S0	00
S1	01

Moore FSM Schematic



Slide Courtesy of Harris&Harris Book

Mealy FSM Schematic



Slide Courtesy of Harris&Harris Book

FSM design procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. For Moore machine:
    1. Rewrite state transition table with state encodings
    2. Write output table
6. For a Mealy machine:
    1. Rewrite combined state transition and output table with state encodings

Finite State Machine Design: More Complex Sequence Detector

- Design a Finite State Machine to detect consecutive series of three or more ‘1’s in serial input bit stream i.e., output becomes ‘1’ when three or more consecutive 1’s are detected in the input bit stream

- **4 states** required to such sequence detector state machine :
  - State **IDLE**: reset sate (zero 1s detected)
  - State **S1**: one 1 detected
  - State **S2**: two 1s detected
  - State **S3**: three or more 1s detected

after three consecutive 1’s, out signal is set to 1

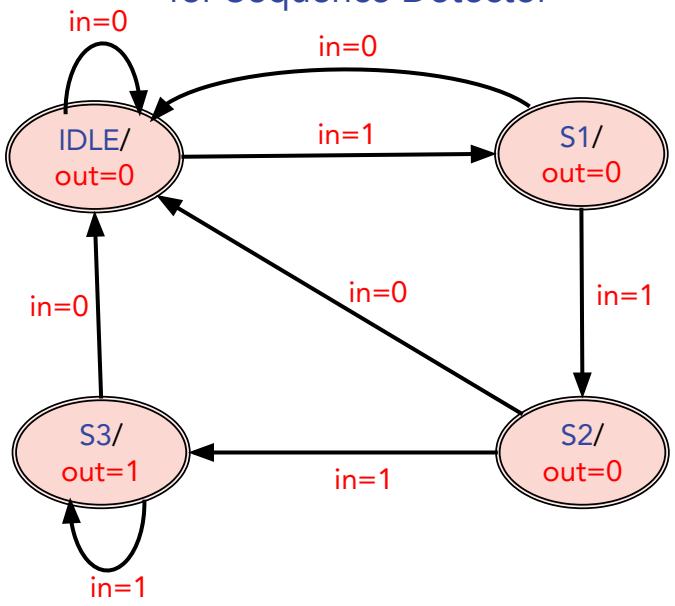
after three consecutive 1’s, out signal is set to 1

Overlap condition should be considered as three consecutive 1’s and out signal is set to 1

in	0	1	1	1	0	1	0	1	1	1	1
out	0	0	0	1	0	0	0	0	0	1	1

Finite State Machine: Sequence Detector

Moore FSM Diagram for Sequence Detector

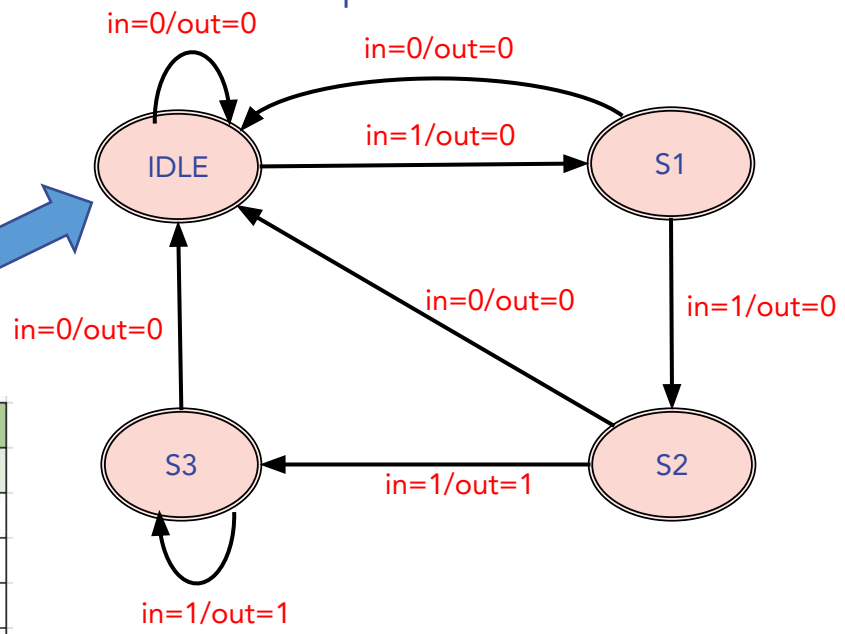


Note : In Moore FSM, output is specified as part of present state

State Transition Table for Sequence Detector (MEALY)

inputs			outputs	
rstn	present_state	in	next_state	out
0	-	-	IDLE	0
1	IDLE	0	IDLE	0
1	IDLE	1	S1	0
1	S1	0	IDLE	0
1	S1	1	S2	0
1	S2	0	IDLE	0
1	S2	1	S3	1
1	S3	0	IDLE	0
1	S3	1	S3	1

Mealy FSM Diagram For Sequence Detector



Note : In Mealy FSM, output is specified in state transitions

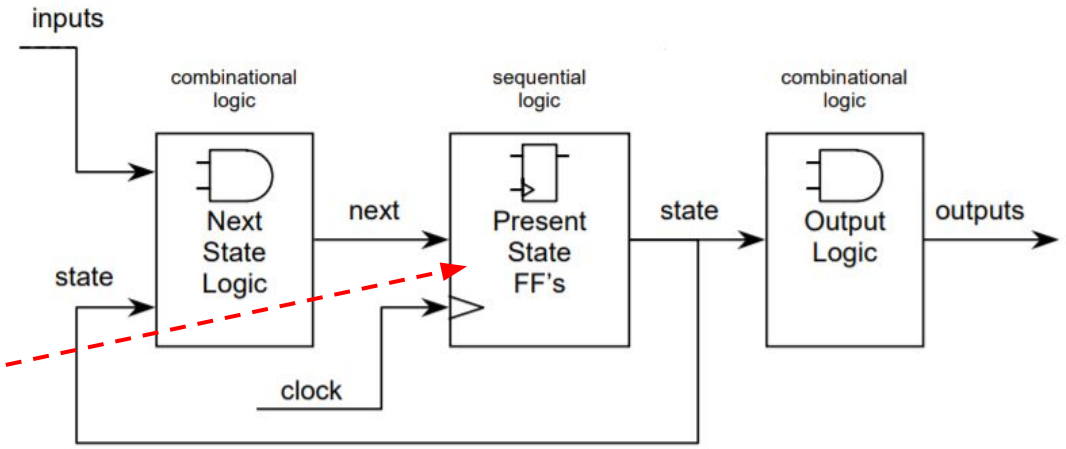
More FSM: Two always Block Approch

Sequence Detector Moore FSM: 2 always block approach

```
module sequence_detector_moore(
 input logic clk, rstn,
 input logic in,
 output logic out);

 // Parameters to define FSM state encodings
 localparam [1:0] IDLE=2'b00,
                  S1=2'b01,
                  S2=2'b10,
                  S3=2'b11;

 // Current state and next state variables
 logic[1:0] present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if(!rstn)
    present_state <= IDLE;
  else
   present_state <= next_state;
 end

 (continued on next slide ...)
```

Synthesis will generate d-flipflops for present_state

(note the use of non-blocking assignments)



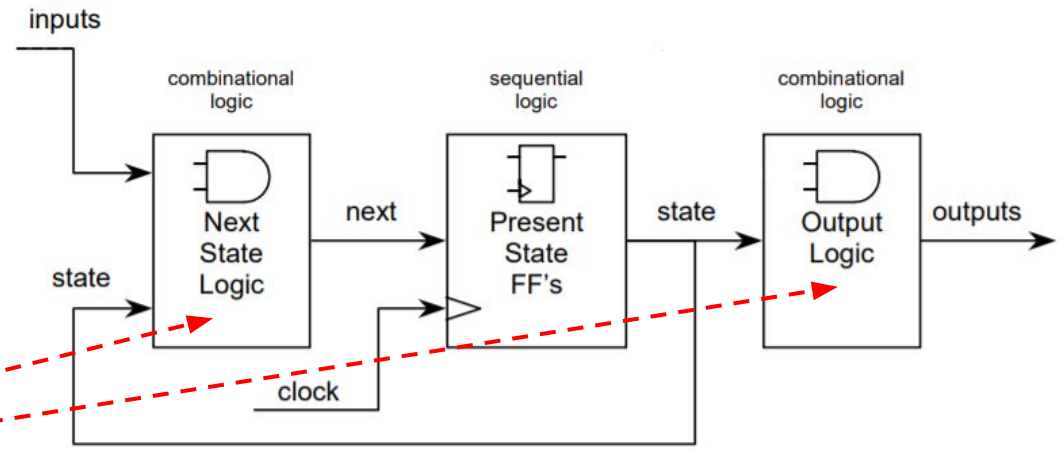
Sequence Detector Moore FSM: 2 always block approach

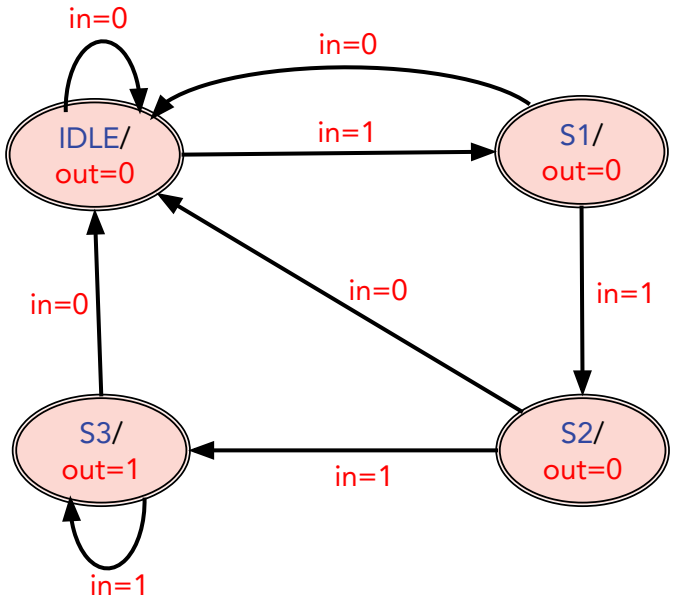
```
// Combination Logic for Next State and Output
always@(present_state,in) begin
 case(present_state)
  IDLE: begin
    out = 0;
    if(in==1) next_state = S1;
    else next_state = IDLE;
  end
  S1: begin
    out = 0;
    if(in==1) next_state = S2;
    else next_state = IDLE;
  end
  S2: begin
    out = 0;
    if(in==1) next_state = S3;
    else next_state = IDLE;
  end
  S3: begin
    out = 1;
    if(in==1) next_state = S3;
    else next_state = IDLE;
  end
(continued on next slide ...)
```

both present_state and input "in" variable should be listed in sensitivity list

Next state logic and output logic code in same always block

(note the use of blocking assignments)

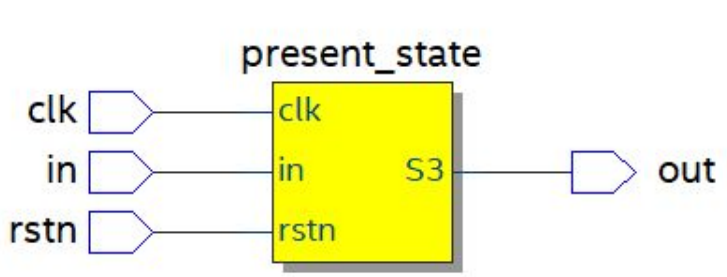




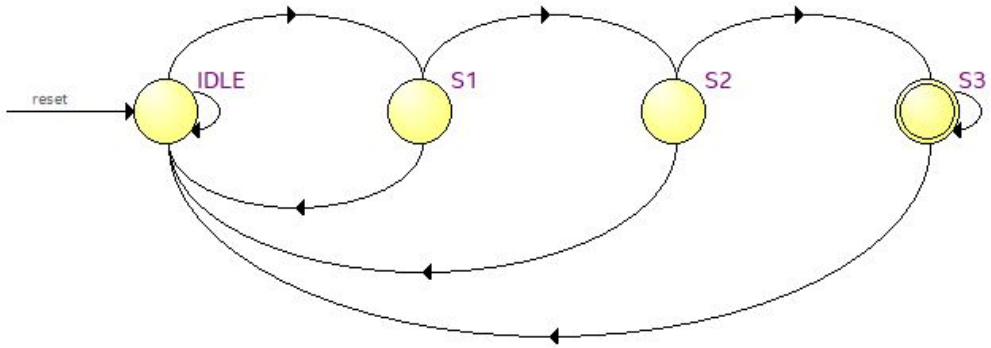
Sequence Detector Moore FSM: 2 always block approach

```
default: begin
    out = 0;
    next_state = IDLE;
  end
 endcase
end
endmodule: sequence_detector_moore
```

default is specified in case a bad state is reached



Sequence Detector RTL Netlist View generated from Synthesizer



Sequence Detector State Machine Diagram generated by Synthesizer

	Source State	Destination State	Condition
1	IDLE	IDLE	(!in) + (in).(!rstn)
2	IDLE	S1	(in).(rstn)
3	S1	IDLE	(!in) + (in).(!rstn)
4	S1	S2	(in).(rstn)
5	S2	IDLE	(!in) + (in).(!rstn)
6	S2	S3	(in).(rstn)
7	S3	IDLE	(!in) + (in).(!rstn)
8	S3	S3	(in).(rstn)

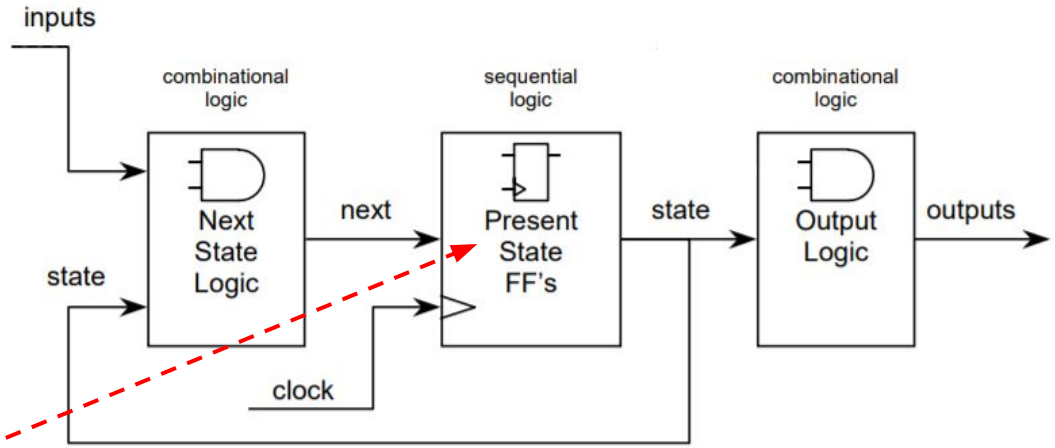
State Machine Transition Table

More FSM: Three always Block Approch

Sequence Detector Moore FSM: 3 always block approach

```
module sequence_detector_moore(
 input logic clk, rstn,
 input logic in,
 output logic out);

 // Parameters to define FSM state encodings
 localparam [1:0] IDLE=2'b00,
                  S1=2'b01,
                  S2=2'b10,
                  S3=2'b11;

 // Current state and next state variables
 logic[1:0] present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if(!rstn)
    present_state <= IDLE;
  else
   present_state <= next_state;
 end

 (continued on next slide ...)
```

1st always block for present state sequential logic



Sequence Detector Moore FSM: 3 always block approach

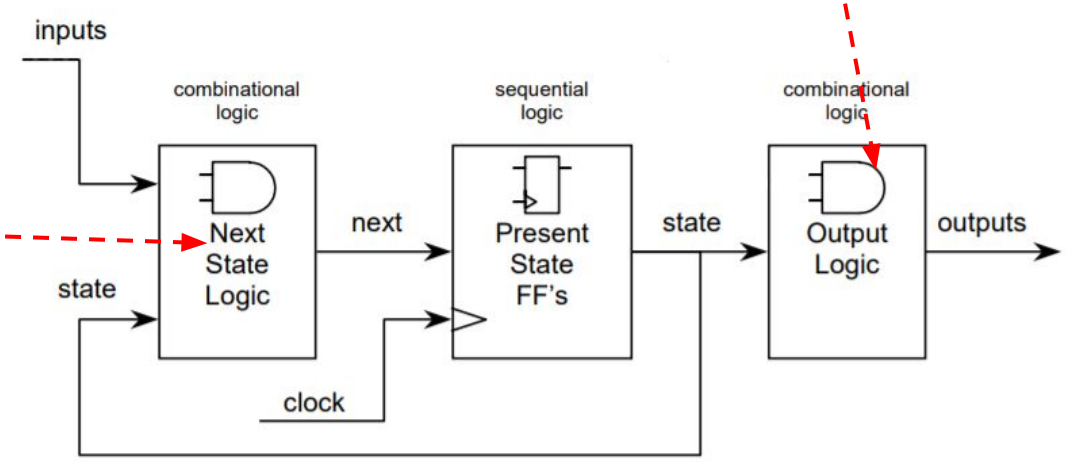
```
// Combination Logic for Next State and Output
always@(present_state,in) begin
 case(present_state)
  IDLE: begin
    if(in==1) next_state = S1;
    else next_state = IDLE;
  end
  S1: begin
    if(in==1) next_state = S2;
    else next_state = IDLE;
  end
  S2: begin
    if(in==1) next_state = S3;
    else next_state = IDLE;
  end
  S3: begin
    if(in==1) next_state = S3;
    else next_state = IDLE;
  end
  default: next_state = IDLE;
endcase
 end
 (continued….)
```

both present_state and input "in" variable should be listed in sensitivity list

2nd always block for separate combinational block for next state logic

```
always@(present_state) begin
 case(present_state)
  S3: out = 1;
  default: out = 0;
 endcase
end
endmodule: sequence_detector_moore
```

3rd always block separates combinational block for output logic



True or False?

Two or three always block approaches for Moore FSMs are equivalent

- ❑ True
- ❑ False

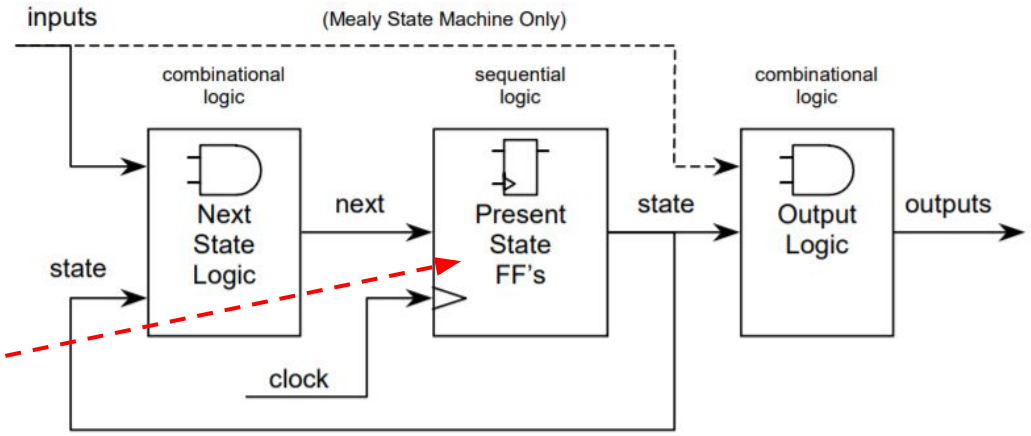
Mealy FSM: Two always Block Approch

Sequence Detector Mealy FSM: 2 always block approach

```
module sequence_detector_mealy(
 input logic clk, rstn,
 input logic in,
 output logic out);

 // Parameters to define FSM state encodings
 localparam [1:0] IDLE=2'b00,
                  S1=2'b01,
                  S2=2'b10,
                  S3=2'b11;

 // Current state and next state variables
 logic[1:0] present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if(!rstn)
    present_state <= IDLE;
  else
    present_state <= next_state;
 end

 (continued on next slide ...)
```

Synthesis will generate d-flipflops for present_state

(note the use of non-blocking assignments)



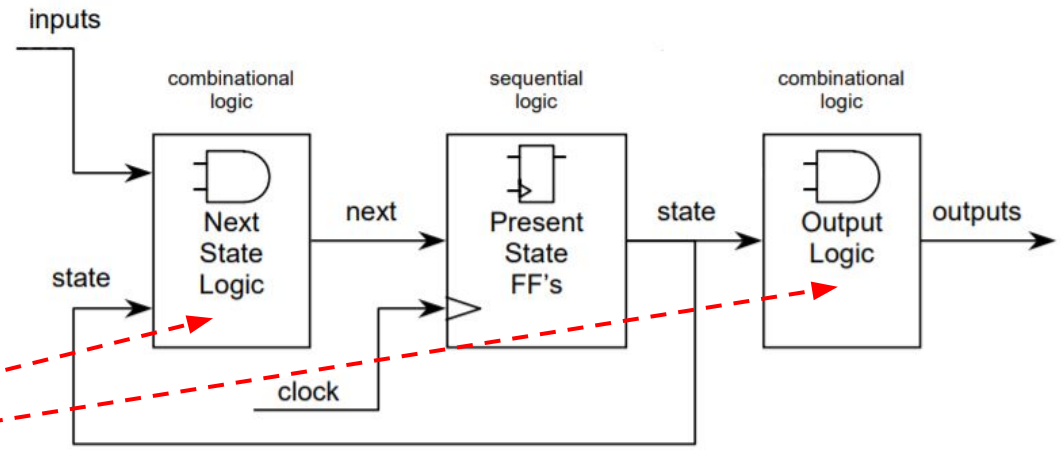
Sequence Detector Mealy FSM: 2 always block approach

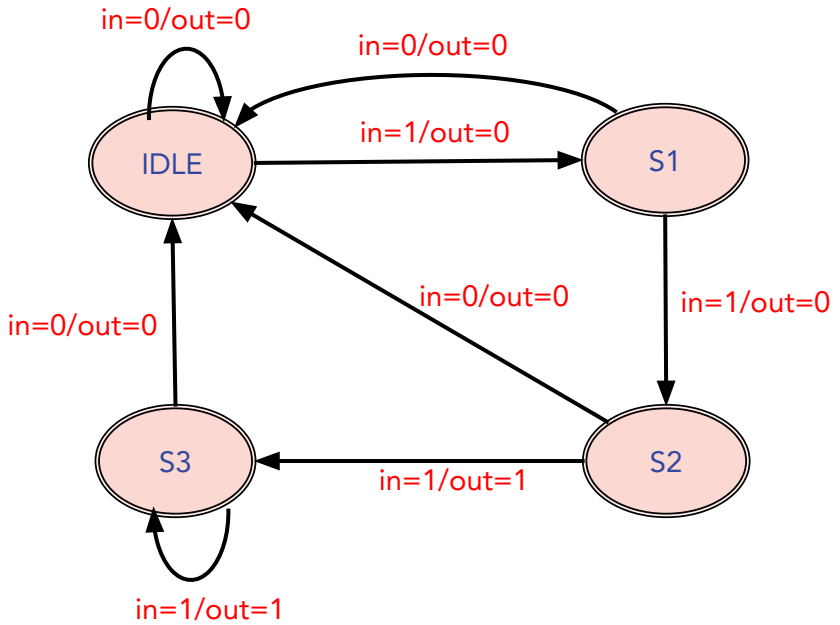
```
// Combination Logic for Next State and Output
always@(present_state, in) begin
 case(present_state)
  IDLE: begin
   if(in==1) begin
    next_state = S1;
    out = 0;
   end
   else begin
    next_state = IDLE;
    out = 0;
   end
  end
  S1: begin
   if(in==1) begin
    next_state = S2;
    out = 0;
   end
   else begin
    next_state = IDLE;
    out = 0;
   end
  end
(continued on next slide ...)
```

both present_state and input "in" variable should be listed in sensitivity list

For Mealy FSM, output is set based on influence of both input "in" signal and present_state. Note : out = 0; statement is specified within if(in == 1) condition in mealy. And in <oore, out = 0; is specified outside if(in == 1) condition

(note the use of blocking assignments)



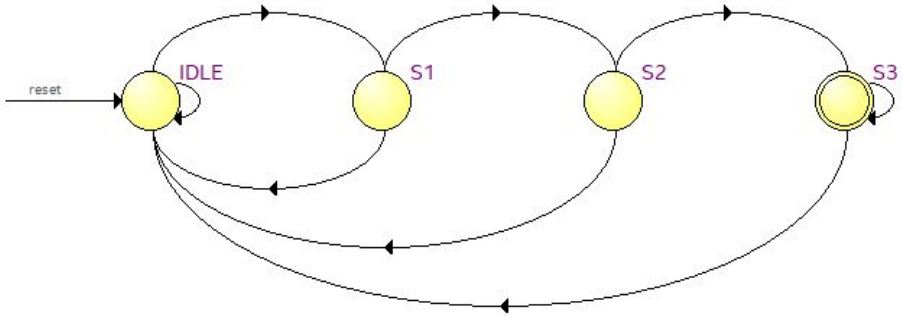


Sequence Detector Mealy FSM: 2 always block approach

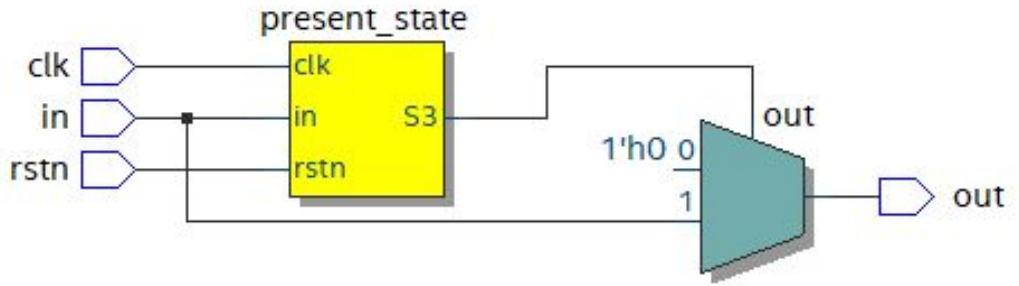
```
  S2: begin
    if(in==1) begin
      next_state = S3;
      out = 1;
    end
    else begin
      next_state = IDLE;
      out = 0;
    end
  end
  S3: begin
    if(in==1) begin
      next_state = S3;
      out = 1;
    end
    else begin
      next_state = IDLE;
      out = 0;
    end
  default: begin out = 0; next_state = IDLE; end
  endcase
 end
endmodule: sequence_detector_mealy
```

For each state, under else condition if "out=0" is not present then Synthesis compiler will create latch for the output "out" signal

default is specified in case a bad state is reached



Sequence Detector State Machine diagram generated by Synthesizer

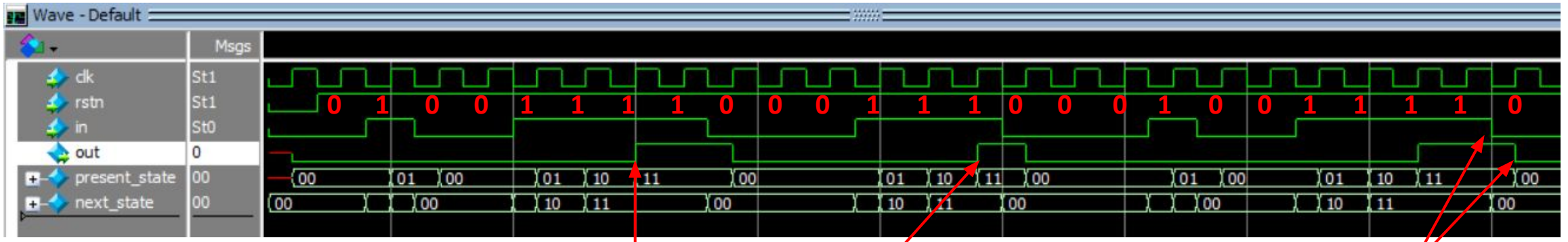


Sequence Detector RTL Netlist view generated by Synthesizer

Simulation Snapshot: Moore vs. Mealy

Sequence Detector Snapshot: Moore Vs. Mealy

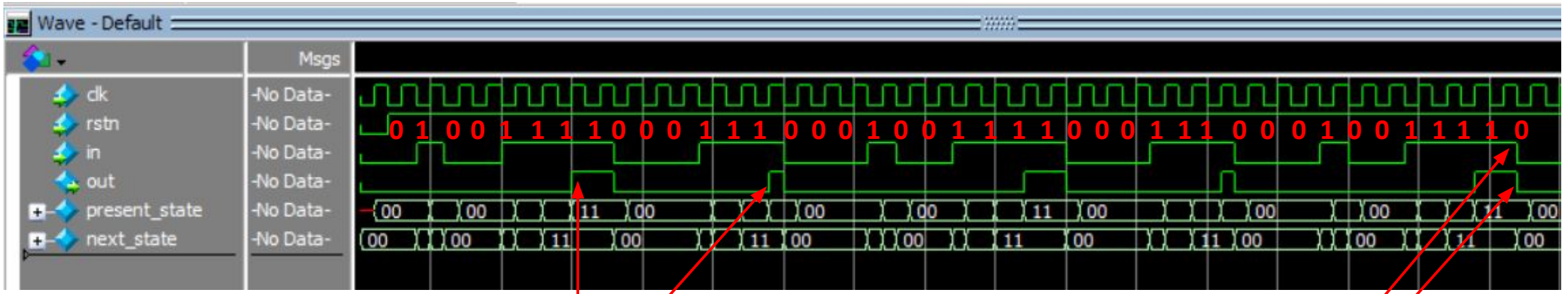
Moore



After three '1' detected out goes to '1'

Output did not react to change in input immediately in Moore

Mealy



After three `1' detected out goes to '1'

Output reacted to change in input immediately in Mealy

FSM Using One always Block

True or False?

Two or three always block approaches for Moore FSMs are equivalent

- ❑ True
- ❑ False

Sequence Detector: One always Block

```
module sequence_detector_one_always_block(
 input logic clk, rstn,
 input logic in,
 output logic out);

 // Parameters to define FSM state encodings
 localparam [1:0] IDLE=2'b00,
     S1=2'b01, S2=2'b10, S3=2'b11;

 // Current state and next state variables
 logic[1:0] state;

 // Use of same clocked always block
always_ff@(posedge clk) begin
  if(!rstn)
    state <= IDLE;
    out <= 0;
  else
    case(state)
      IDLE: begin
        out <= 0;
        if(in==1) state <= S1;
        else state <= IDLE;
       end
```

Does not need two separate state variable

Inputs are no longer asynchronously sampled. Change in input is captured with respect to clock edge event

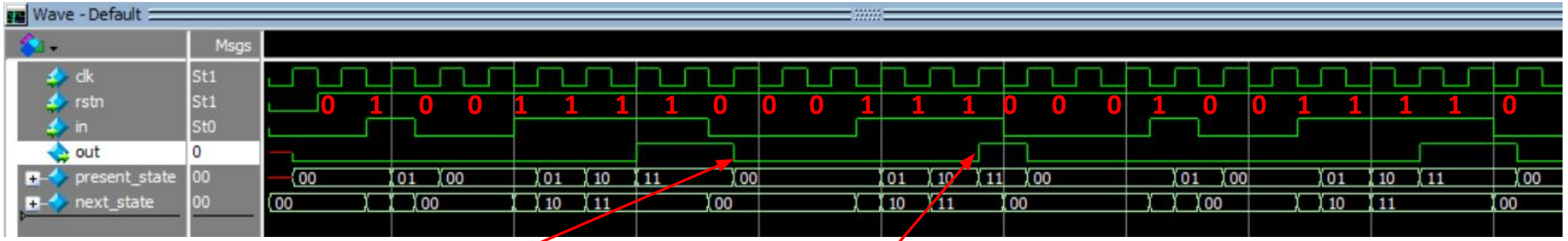
Use of non-blocking assignment statement in case encoding branches

```
 // Combination Logic for Next State and Output
  S1: begin
     out <= 0;
     if(in==1) state <= S2;
     else state <= IDLE;
   end
   S2: begin
     out <= 0;
     if(in==1) state <= S3;
     else state <= IDLE;
   end
   S3: begin
     out <= 1;
     if(in==1) state <= S3;
     else state <= IDLE;
   end
   default: begin
     out <= 0;
     state <= IDLE;
   end
  endcase
 end
endmodule: sequence_detector_one_always_block
```

Output will stay asserted longer even when state has transitioned to another state where output should have gone back to reset or some other value

Sequence Detector Simulation Snapshot: One vs. Two always Block

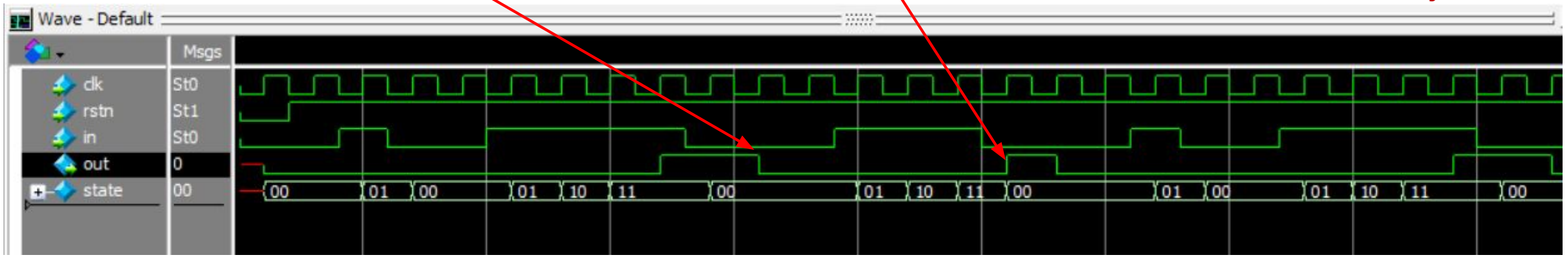
2 always blocks (Moore FSM)



In case of 1 always block implementation out = 1 stays for one additional clock cycle then 2 or 3 always block approach

out = 1 took additional one cycle in case of 1 always block approach compared to 2 or 3 always block implementation

1 always block



One Always Block Approach For FSM Modeling

- **One always block** state machine is slightly **more simulation-efficient** than the two always block state machine :
  - Since the inputs are only examined on clock changes (less work for simulator)

- **Disadvantages** of one always block approach:
  1. State machine can be **more difficult to modify and debug** since all sequential and combinational logic is mixed in one always block
  2. Placing output assignments inside of the always block will infer **output flip-flops**
     - This might lead to **late availability of output**. This may not be desirable in some applications.
  3. **Mealy FSM cannot be modeled** using one always block!