

ECE 2300
Digital Logic & Computer Organization
Spring 2025

More Finite State Machines



Cornell University

Announcements

- **TA-led Quartus/Verilog Tutorial next Tuesday**
 - Bring your laptop
- **Prelim 1, Thursday 2/27, 1:25-2:40pm (in class)**
 - Arrive early by 1:15pm
 - TA-led review on Monday, 2/24 @ 7:30 pm (virtual)
 - No lab sessions next week
 - Solutions for HW2, HW3, and practice prelim will be released today

Recap: Procedural Assignments in Verilog

```
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y = A & B;  
    Z = Y;  
end
```

Blocking assignments

```
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y <= A & B;  
    Z <= Y;  
end
```

Nonblocking assignments

Simulator Interpretation

$Y_{\text{next}} \leftarrow A \& B$
 $Z_{\text{next}} \leftarrow (Y_{\text{next}} = A \& B)$ // use "new" Y

$Z_{\text{next}} \leftarrow Y$ // use "old" Y
 $Y_{\text{next}} \leftarrow A \& B$

- RHS evaluated *sequentially*
- Assignment to LHS is *immediate*

- RHS evaluated *in parallel* (order doesn't matter)
- Assignment to LHS is *delayed* until the end of the always block

Recap: Procedural Assignments in Verilog

```
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y = A & B;  
    Z = Y;  
end
```

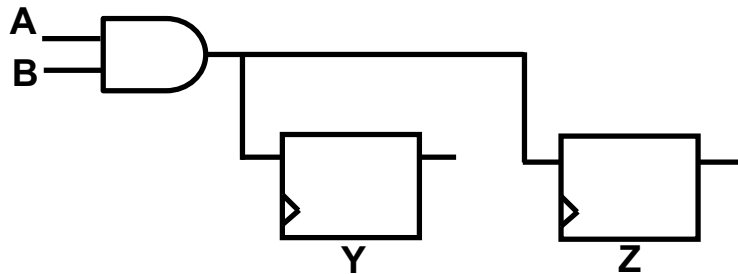
Blocking assignments

```
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y <= A & B;  
    Z <= Y;  
end
```

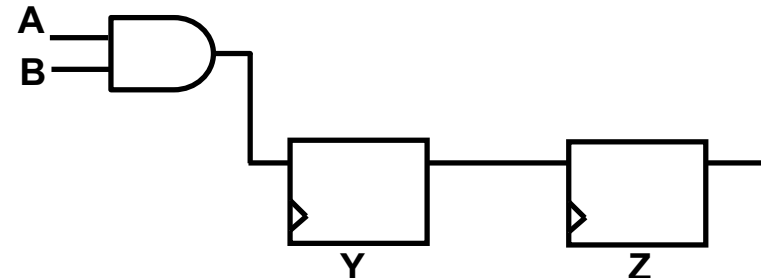
Nonblocking assignments

Actual Circuit

(Synthesizer Interpretation)



When a reg (Y here) is assigned in a *blocking* assignment ($Y=A&B$), employ its D input (i.e., $A&B$) for connection in RHS of a subsequent assignment ($Z=Y$)



When a reg (Y here) is assigned in a *nonblocking* assignment ($Y<=A&B$), employ its Q output for connection in RHS of another assignment ($Z<=Y$)

(Improperly Created) *Inferred* Latches

- To infer combinational logic, you're recommended to ensure that each variable within an always block gets assigned a value under *all possible conditions*
 - Otherwise, the Verilog compiler assumes that the last value should be used, and will create a latch

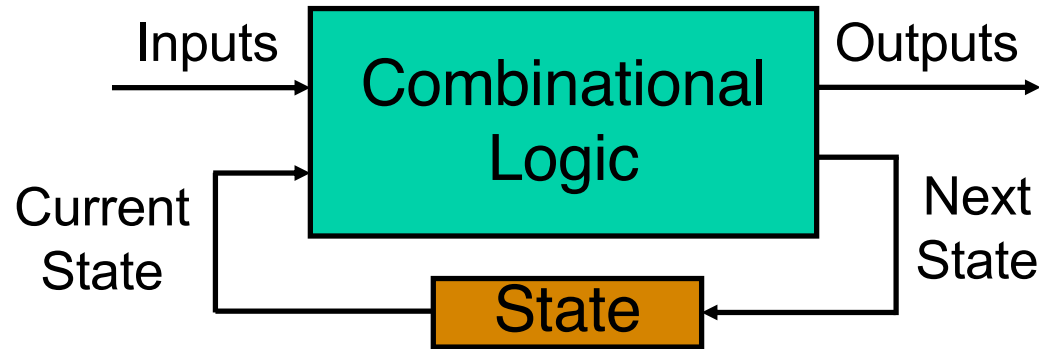
```
reg out;
always @(d, sel)
begin
    if (sel == 1'b1)
        out = d;
end
```

out is keeping state because
it's not always assigned a value
→ **latch inferred**

```
reg out;
always @(d, sel)
begin
    if (sel == 1'b1)
        out = d;
    else
        out = ~d;
end
```

out assigned a value in both conditions
→ **combinational logic inferred**
(not latch)

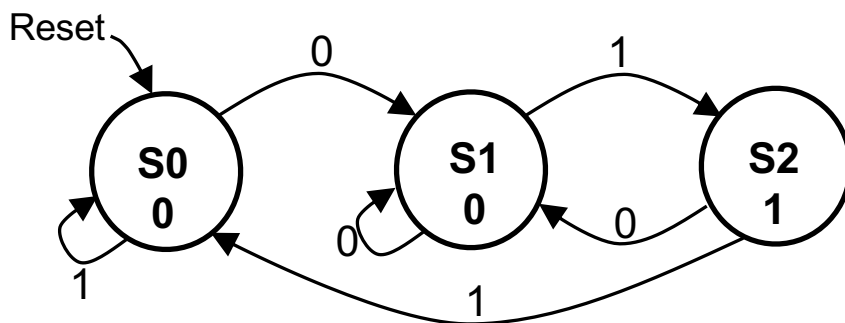
Review: Finite State Machine (FSM)



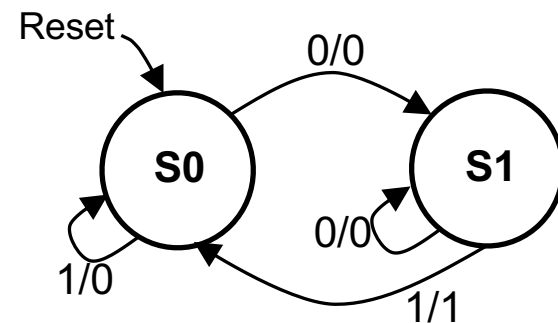
- An FSM is an *abstract representation of a sequential circuit*

1. A finite number of inputs; 2. A finite number of outputs
3. A finite number of states; 4. A specification of all state transitions

Can be described by a state diagram

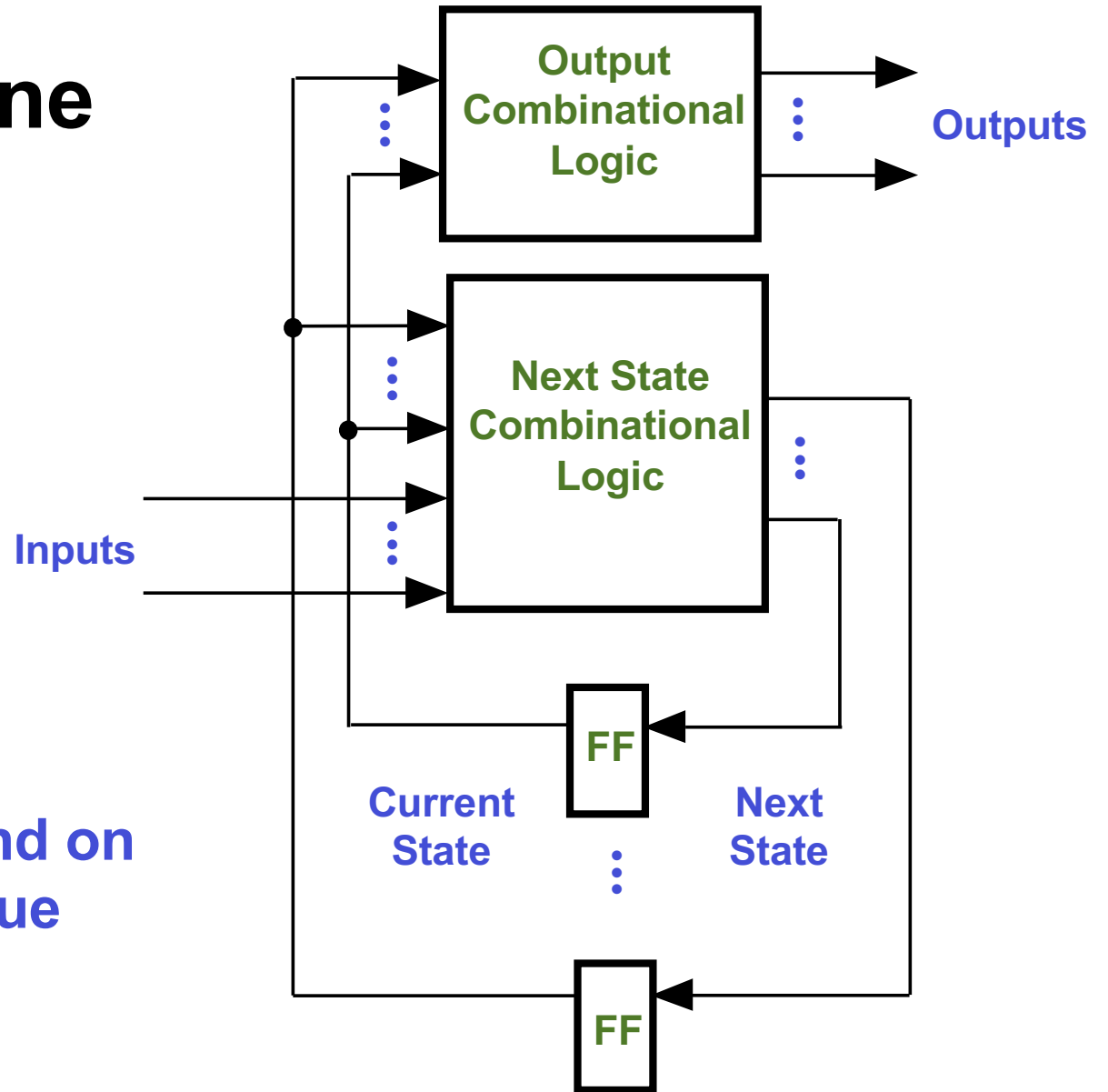


Moore FSM Example



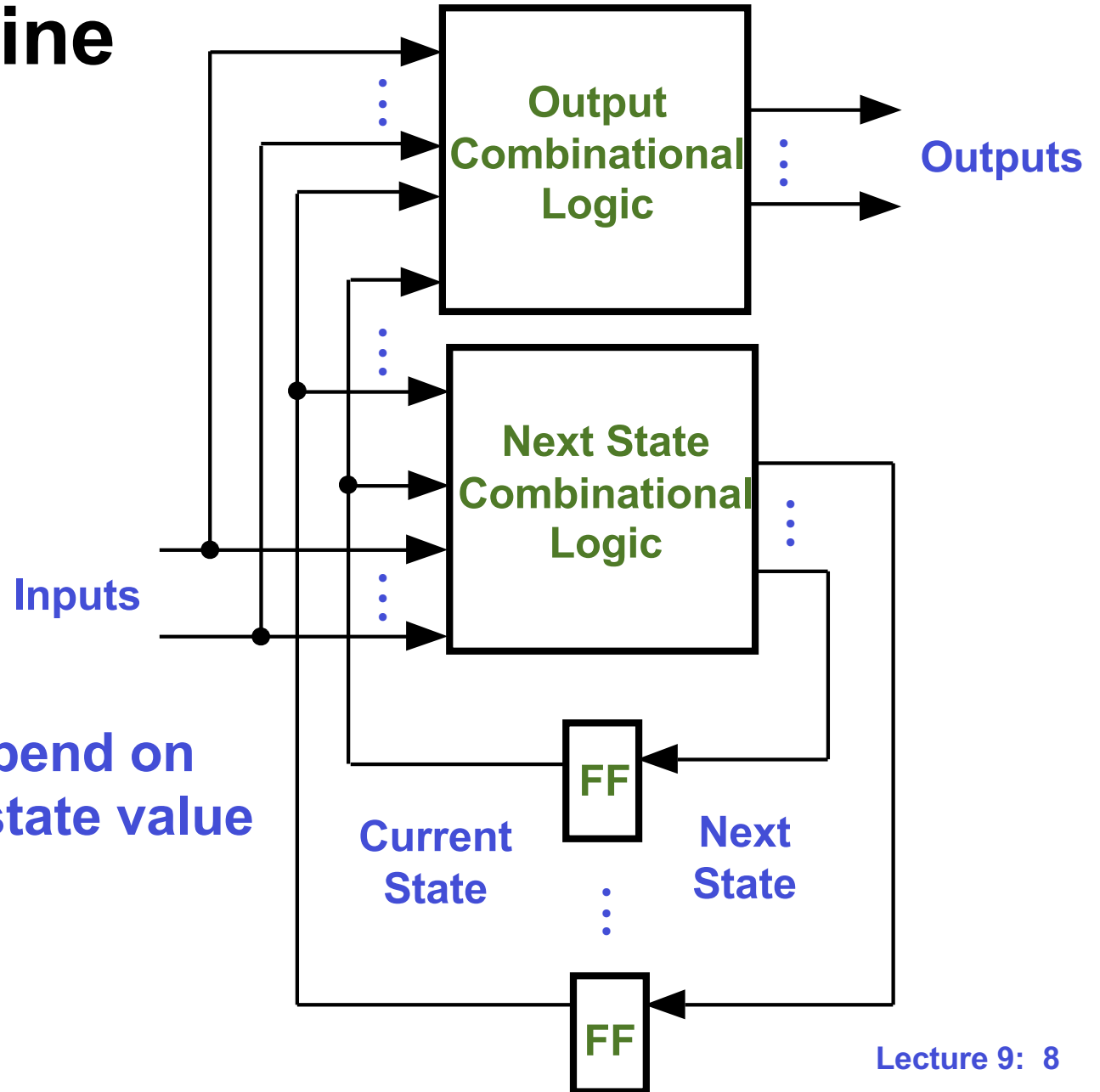
Mealy FSM Example

Review: Moore Machine



Outputs only depend on
current state value

Review: Mealy Machine



Outputs only depend on
input and current state value

FSM Design Procedure

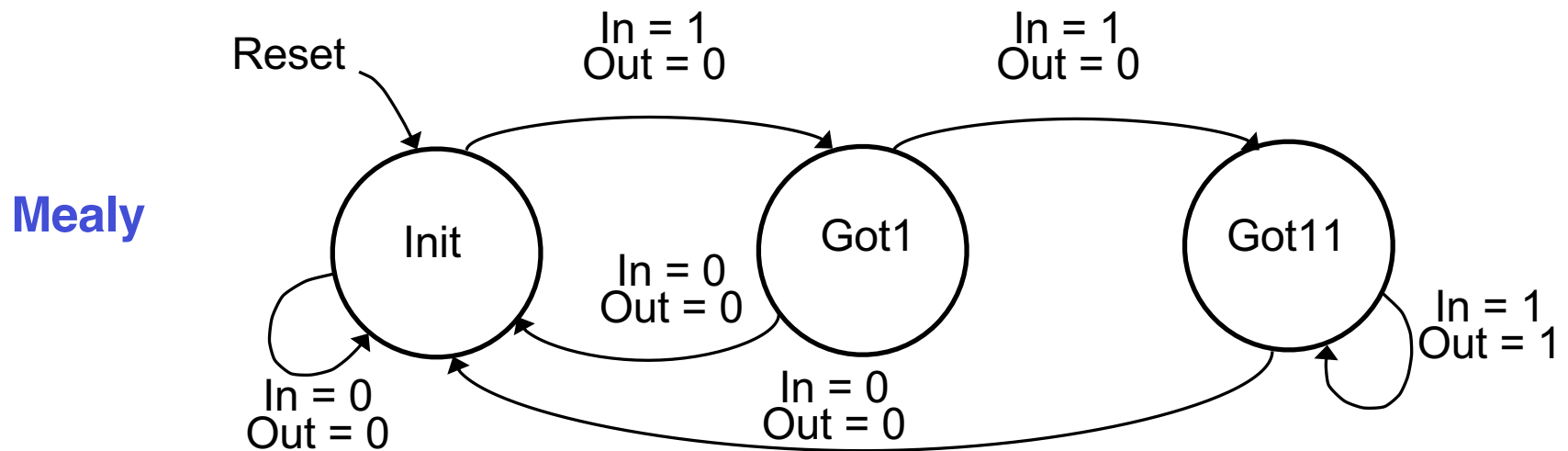
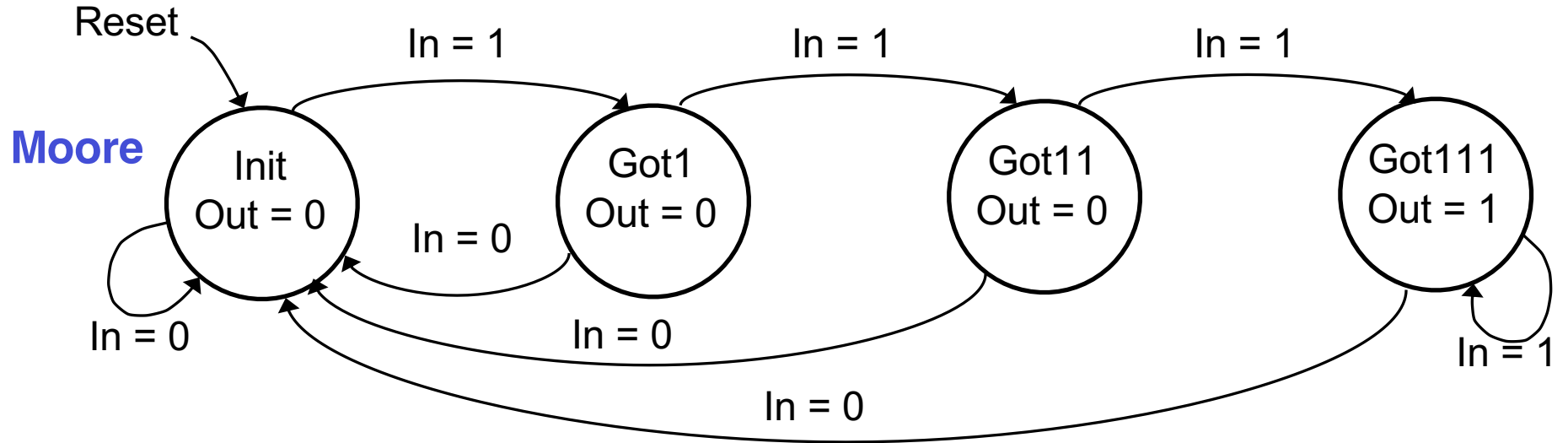
- (1) Understand the problem statement and determine inputs and outputs
- (2) Identify states and create a *state diagram*
- (3) Determine the number of required FFs
- (4) Implement combinational logic for outputs and next state
- (5) Simulate the circuit to test its operation

This
lecture

Example FSM: Pattern Detector

- Monitors the input, and outputs a 1 whenever a specified input pattern is detected
- Example: Output a 1 whenever 111 is detected on the input over 3 consecutive clock cycles
 - Overlapping patterns also detected (1111...)
- Input *In* (one bit)
- Output *Out* (one bit)
- *Reset* causes FSM to start in initial state
- *Clock* input not shown (always present)

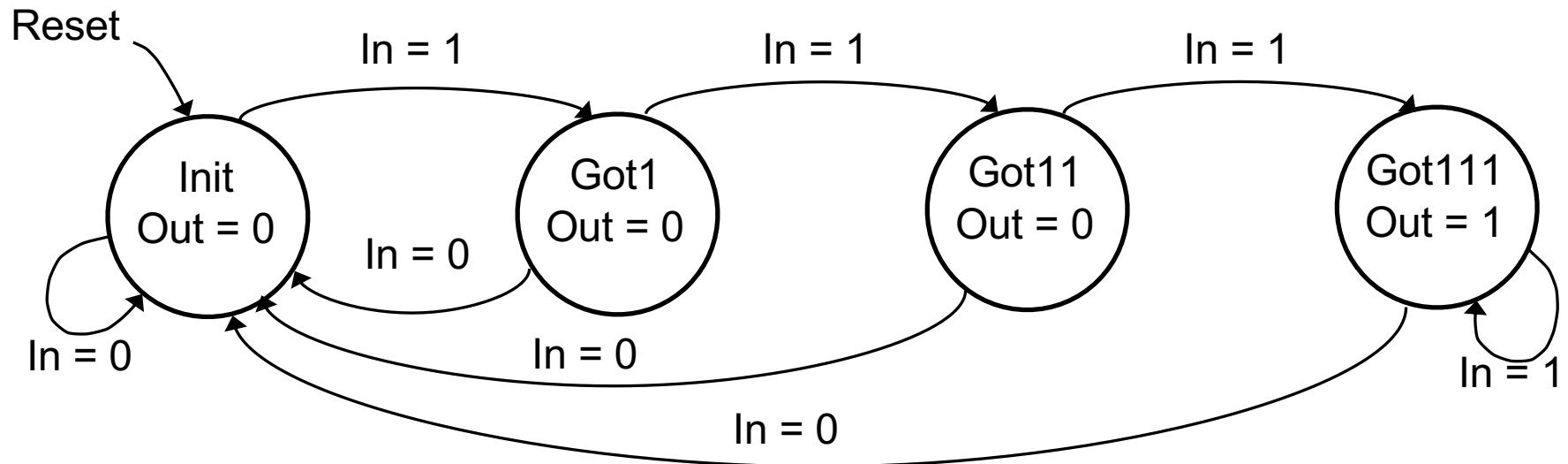
State Diagrams



Transition/Output Table

- Shows the next state (S^*) and output values for each combination of current state (S) and inputs
- Used to derive the minimized *state transition* (S^*) and *output* Boolean equations

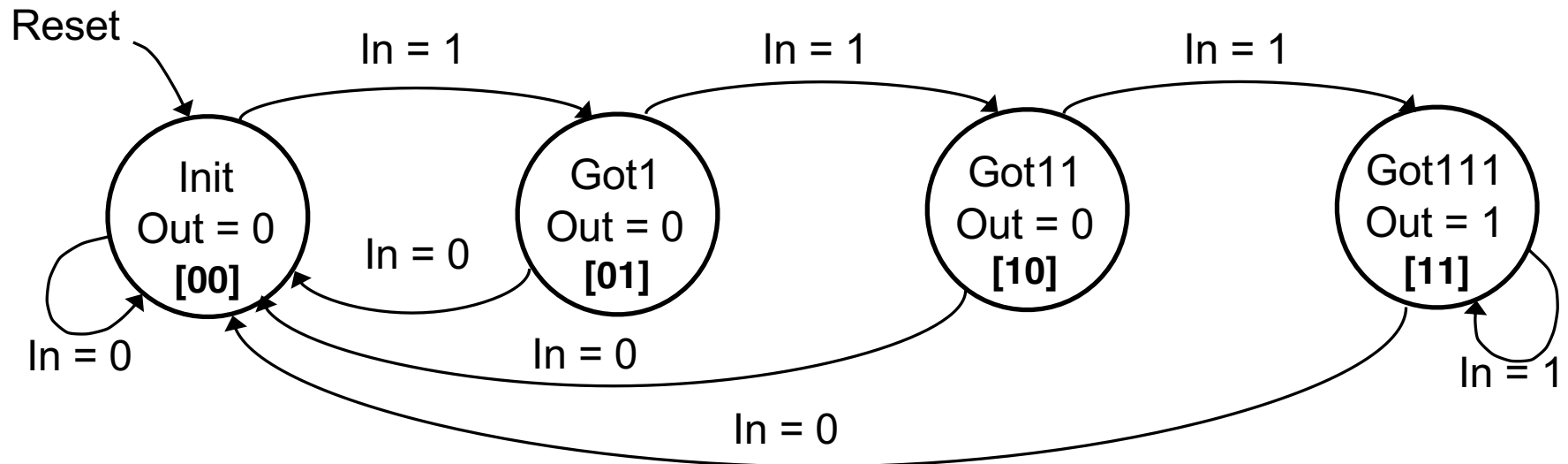
Moore Transition/Output Table 1



Current State (S)	Next State (S*)		Out
	In = 0	In = 1	
Init	Init	Got1	0
Got1	Init	Got11	0
Got11	Init	Got111	0
Got111	Init	Got111	1

- **Version 1: uses descriptive state names**

Moore Transition/Output Table 2



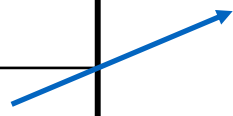
$S_1 S_0$	$S_1^* S_0^*$		Out
	In = 0	In = 1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

- **Version 2: uses state binary encodings**

Minimized Equations for S^* and Out

$S_1 S_0$	$S_1^* S_0^*$		Out
	In = 0	In = 1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

Out = $S_1 \cdot S_0$



Minimized Equations for S^* and Out

		$S_1 S_0$			
		00	01	11	10
In	0	0	0	0	0
	1	0	1	1	1

$$S_1^* = S_0 \cdot In + S_1 \cdot In$$

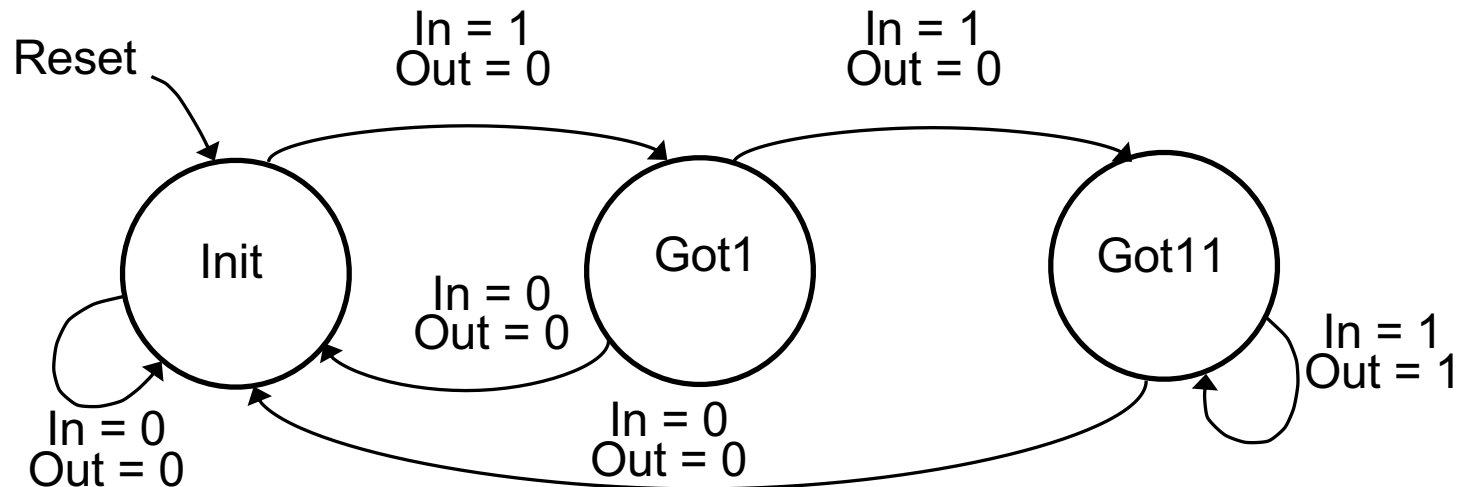
		$S_1 S_0$			
		00	01	11	10
In	0	0	0	0	0
	1	1	0	1	1

$$S_0^* = S_0' \cdot In + S_1 \cdot In$$

$S_1 S_0$	$S_1^* S_0^*$		Out
	In = 0	In = 1	
00	00	01	0
01	00	10	0
10	00	11	0
11	00	11	1

$$Out = S_1 \cdot S_0$$

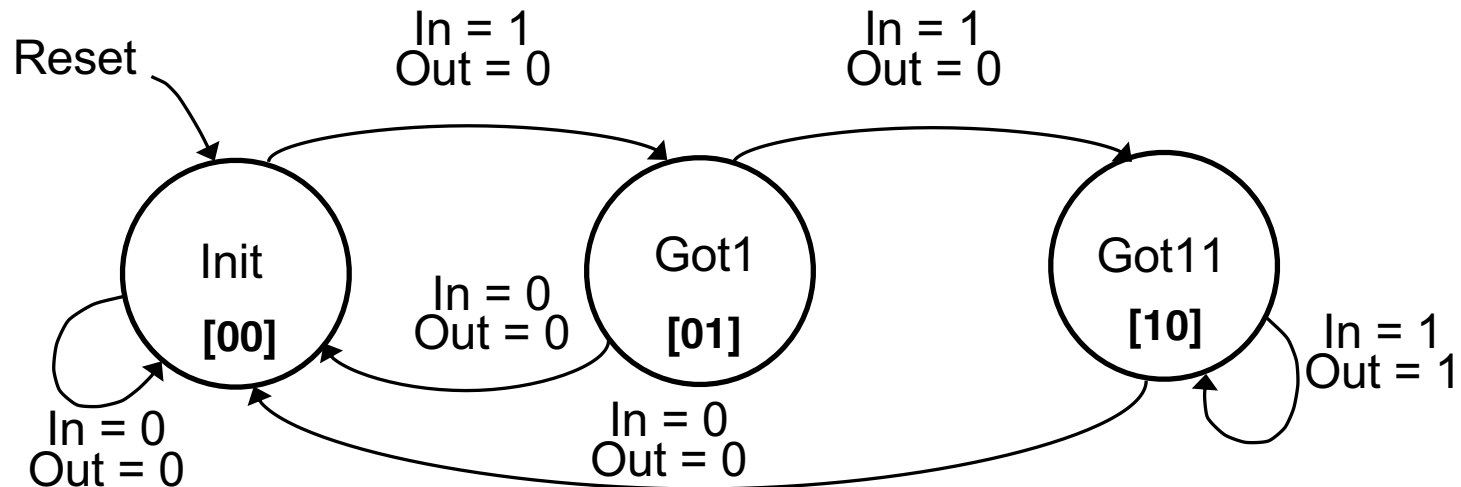
Mealy Transition/Output Table 1



Current State (S)	Next State (S*), Out	
	In = 0	In = 1
Init	Init, 0	Got1, 0
Got1	Init, 0	Got11, 0
Got11	Init, 0	Got11, 1

- **Version 1: uses descriptive state names**

Mealy Transition/Output Table 2



$S_1 S_0$	$S_1^* S_0^*, \text{Out}$	
	In = 0	In = 1
0 0	0 0, 0	0 1, 0
0 1	0 0, 0	1 0, 0
1 0	0 0, 0	1 0, 1

- **Version 2: uses state binary encodings**

Minimized Equations for S^* and Out

$S1\ S0$	$S1^*\ S0^*,\ Out$	
	$In = 0$	$In = 1$
0 0	0 0, 0	0 1, 0
0 1	0 0, 0	1 0, 0
1 0	0 0, 0	1 0, 1

Minimized Equations for S^* and Out

		S_1S_0			
		00	01	11	10
In	0	0	0	d	0
	1	0	1	d	1

$$S_1^* = S_0 \cdot In + S_1 \cdot In$$

		S_1S_0			
		00	01	11	10
In	0	0	0	d	0
	1	1	0	d	0

$$S_0^* = S_1' \cdot S_0' \cdot In$$

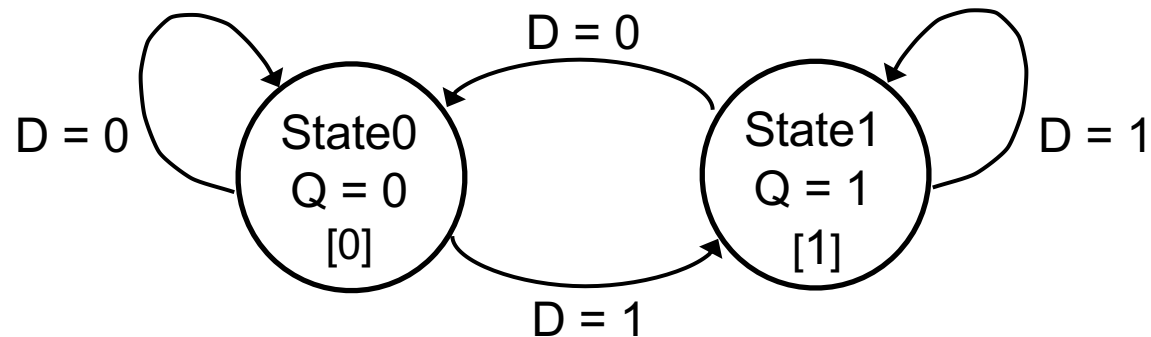
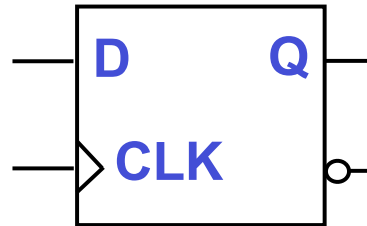
		S_1S_0			
		00	01	11	10
In	0	0	0	d	0
	1	0	0	d	1

$$Out = S_1 \cdot In$$

$S_1 S_0$	$S_1^* S_0^*, Out$	
	In = 0	In = 1
0 0	0 0, 0	0 1, 0
0 1	0 0, 0	1 0, 0
1 0	0 0, 0	1 0, 1

Moore State Diagram for DFF

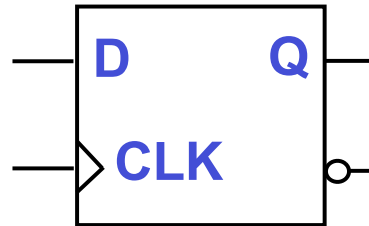
- Input: D
- Output: Q



S	S*		Q
	D = 0	D = 1	
0	0	1	0
1	0	1	1

Mealy State Diagram for DFF?

- Input: D
- Output: Q

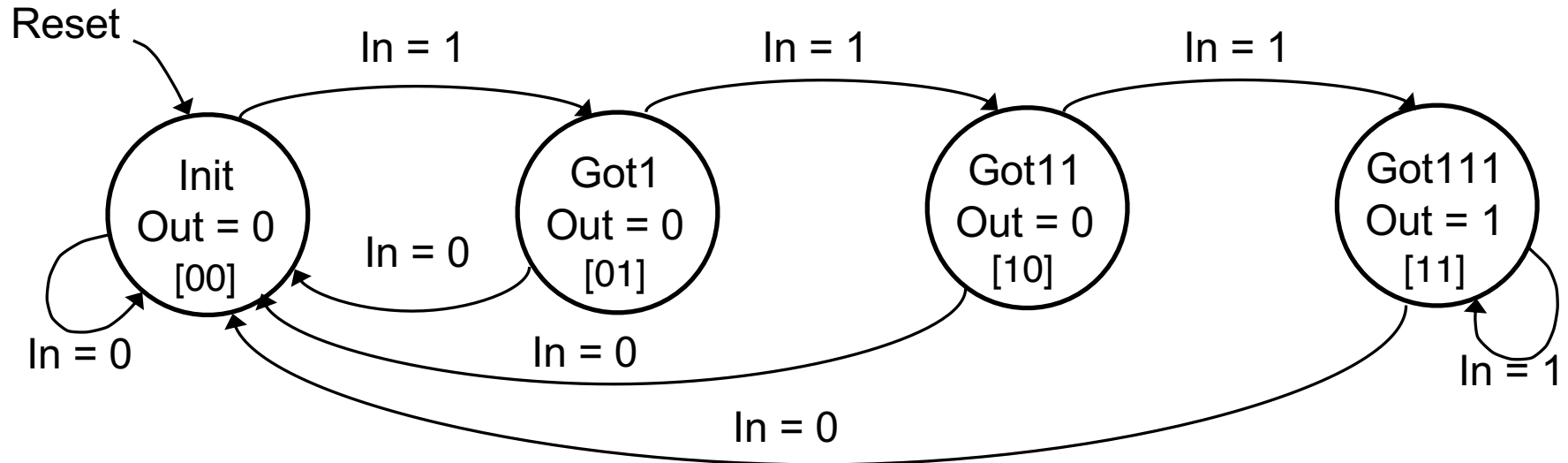


FSMs in Verilog

```
<module statement>  
<input and output declarations>  
  
<reg declarations>  
  
<parameter or typedef statement>  
  
<always block for next state>  
  
<always block for output>  
  
<always block for state FFs>  
  
endmodule
```

**Suggested
coding style
for FSM**

Moore FSM in Verilog

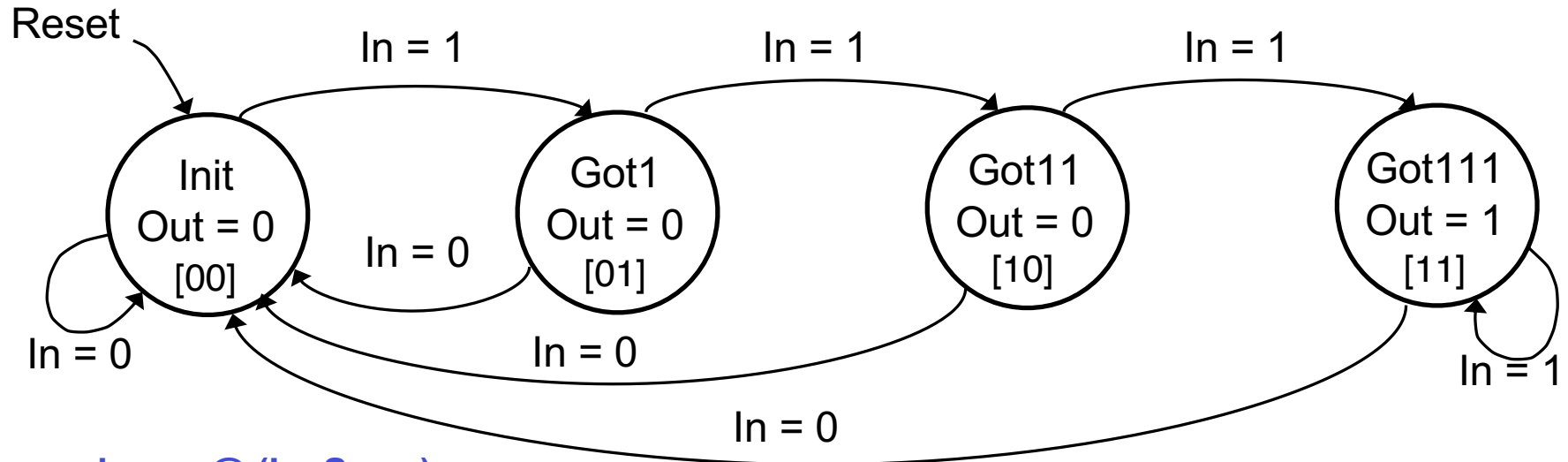


```
module PatDetectMoore (Clk, In, Reset, Out);  
input Clk, In, Reset;  
output Out;
```

```
reg Out;  
reg [1:0] Scurr, Snext;
```

```
parameter [1:0] Init = 2'b00,  
              Got1 = 2'b01,  
              Got11 = 2'b10,  
              Got111 = 2'b11;
```


Moore FSM in Verilog



```
always @ (In, Scurr)
```

```
begin
```

```
  case (Scurr)
```

```
    Init: if (In == 1) Snext = Got1; else Snext = Init;
```

```
    Got1: if (In == 1) Snext = Got11; else Snext = Init;
```

```
    Got11: if (In == 1) Snext = Got111; else Snext = Init;
```

```
    Got111: if (In == 1) Snext = Got111; else Snext = Init;
```

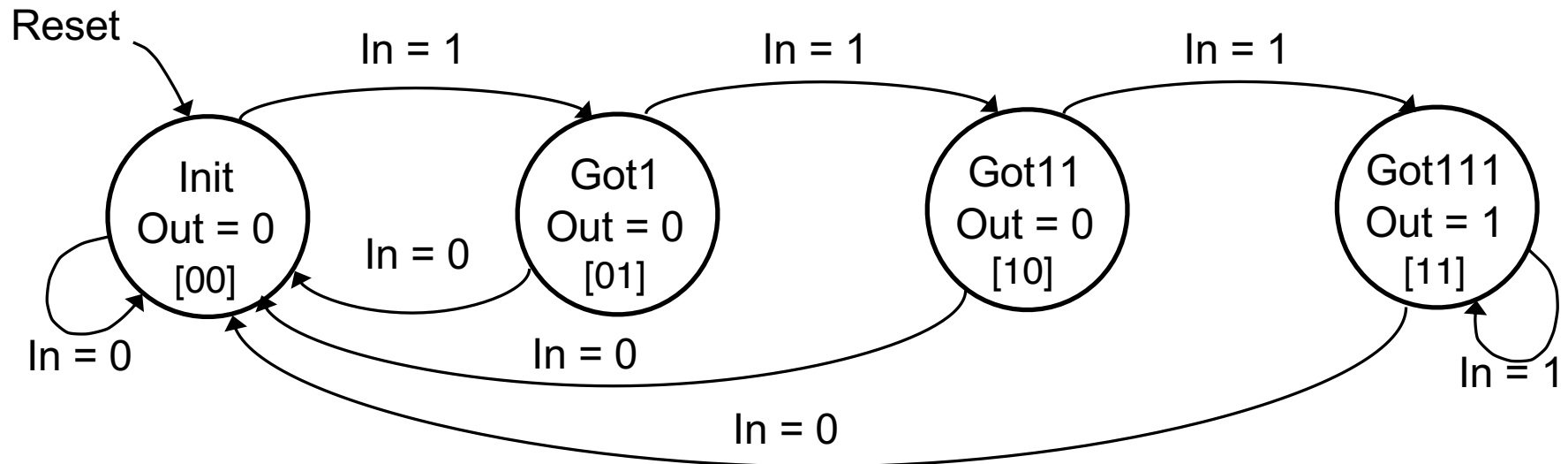
```
    default: Snext = Init;
```

```
  endcase
```

```
end
```

**next state
comb logic**

Moore FSM in Verilog



always @ (Scurr)

```
if (Scurr == Got111) Out = 1;  
else Out = 0;
```

output comb logic

always @ (posedge Clk)

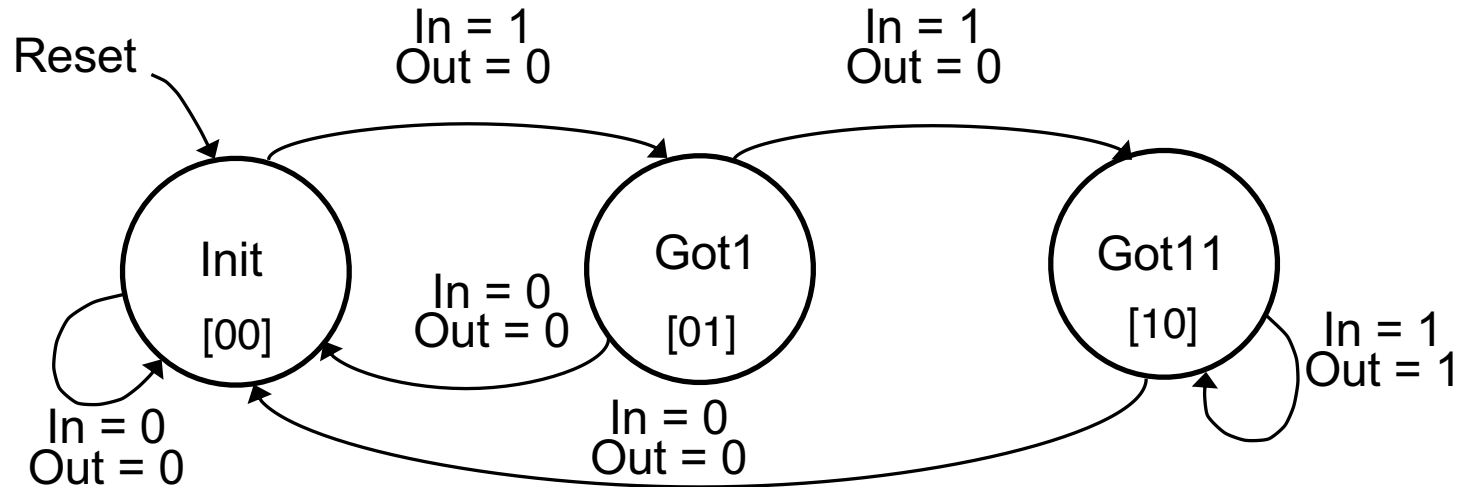
```
if (Reset == 1) Scurr <= Init;  
else Scurr <= Snext;
```

**update state FFs
(current state)**

endmodule

Moore FSM Verilog Simulation

Mealy FSM in Verilog

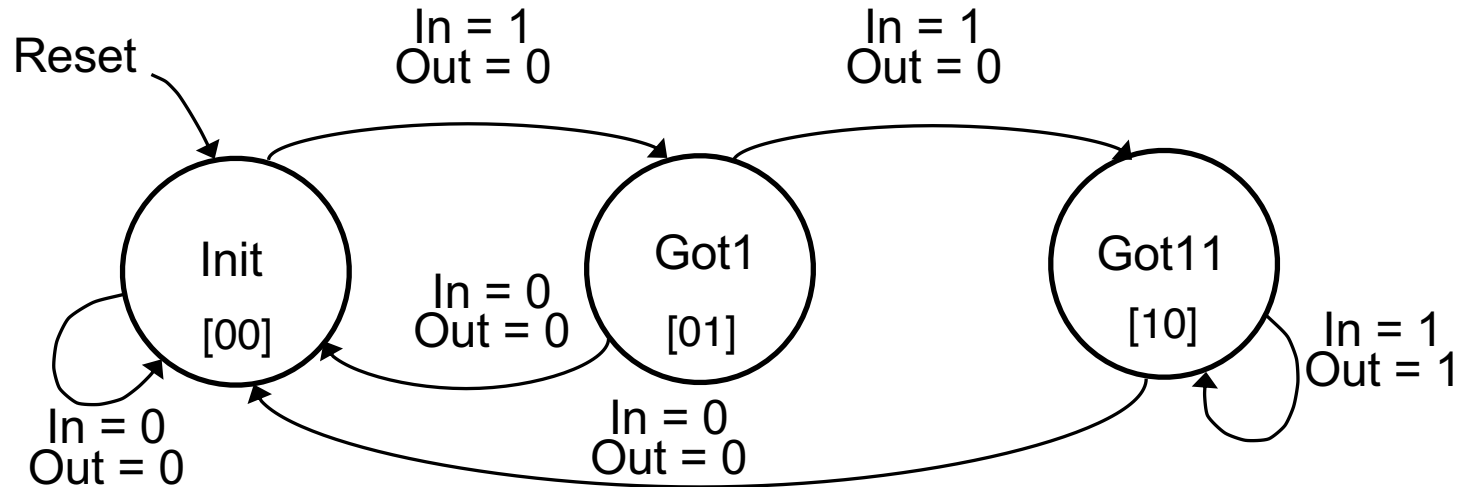


```
module PatDetectMealy (Clk, In, Reset, Out);  
input Clk, In, Reset;  
output Out;
```

```
reg Out;  
reg [1:0] Scurr, Snext;
```

```
parameter [1:0] Init = 2'b00,  
              Got1 = 2'b01,  
              Got11 = 2'b10;
```

Mealy FSM in Verilog



```
always @ (In, Scurr)
```

```
begin
```

```
  case (Scurr)
```

```
    Init: if (In == 1) Snext = Got1; else Snext = Init;
```

```
    Got1: if (In == 1) Snext = Got11; else Snext = Init;
```

```
    Got11: if (In == 1) Snext = Got11; else Snext = Init;
```

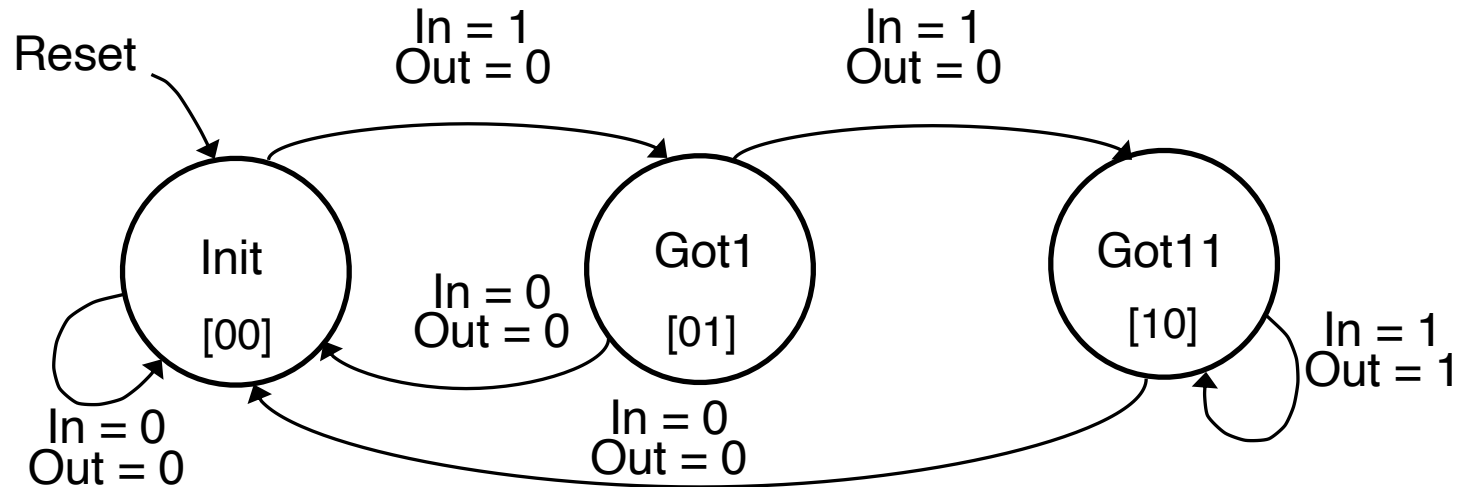
```
    default: Snext = Init;
```

```
  endcase
```

```
end
```

next state
comb logic

Mealy FSM in Verilog



```
always @ (Scurr, In)
  if ((Scurr == Got11) && (In == 1)) Out = 1;
  else Out = 0;
```

output comb logic

```
always @ (posedge Clk)
  if (Reset == 1) Scurr <= Init;
  else Scurr <= Snext;
```

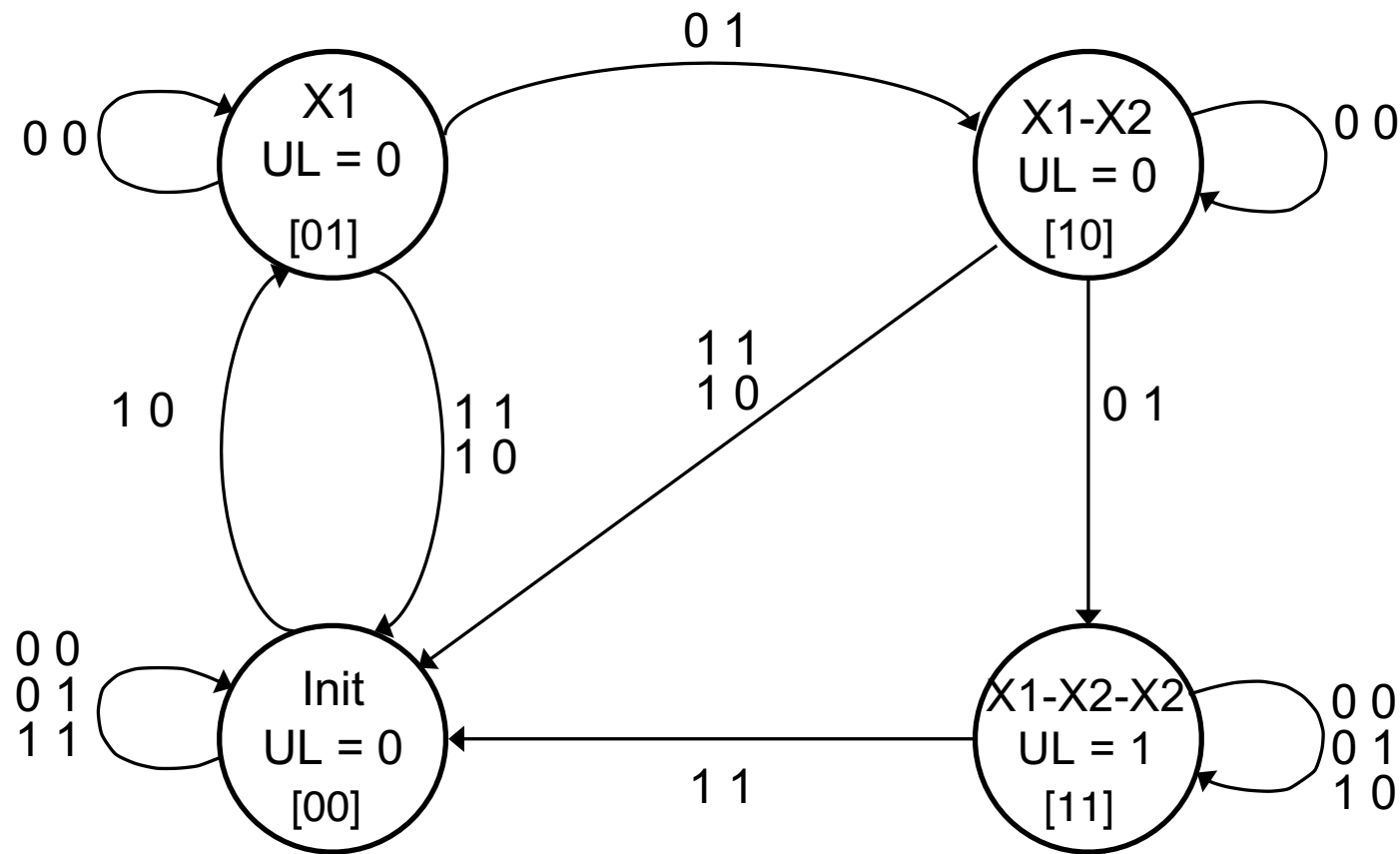
update state FFs
(current state)

```
endmodule
```

Mealy FSM Verilog Simulation

Pushbutton Lock: Moore State Diagram

- Output: $UL=1$ with sequence $X1, X2, X2$
- Input: 00 (neither), 10 ($X1$), 01 ($X2$), 11 (reset)



Moore Transition/Output Table 1

Current State (S)	Next State (S*)				UL
	Input 0 0 (neither)	Input 0 1 (X2)	Input 1 0 (X1)	Input 1 1 (reset)	
Init	Init	Init	X1	Init	0
X1	X1	X1-X2	Init	Init	0
X1-X2	X1,X2	X1-X2-X2	Init	Init	0
X1-X2-X2	X1-X2-X2	X1-X2-X2	X1-X2-X2	Init	1

- **Version 1: uses descriptive state names**

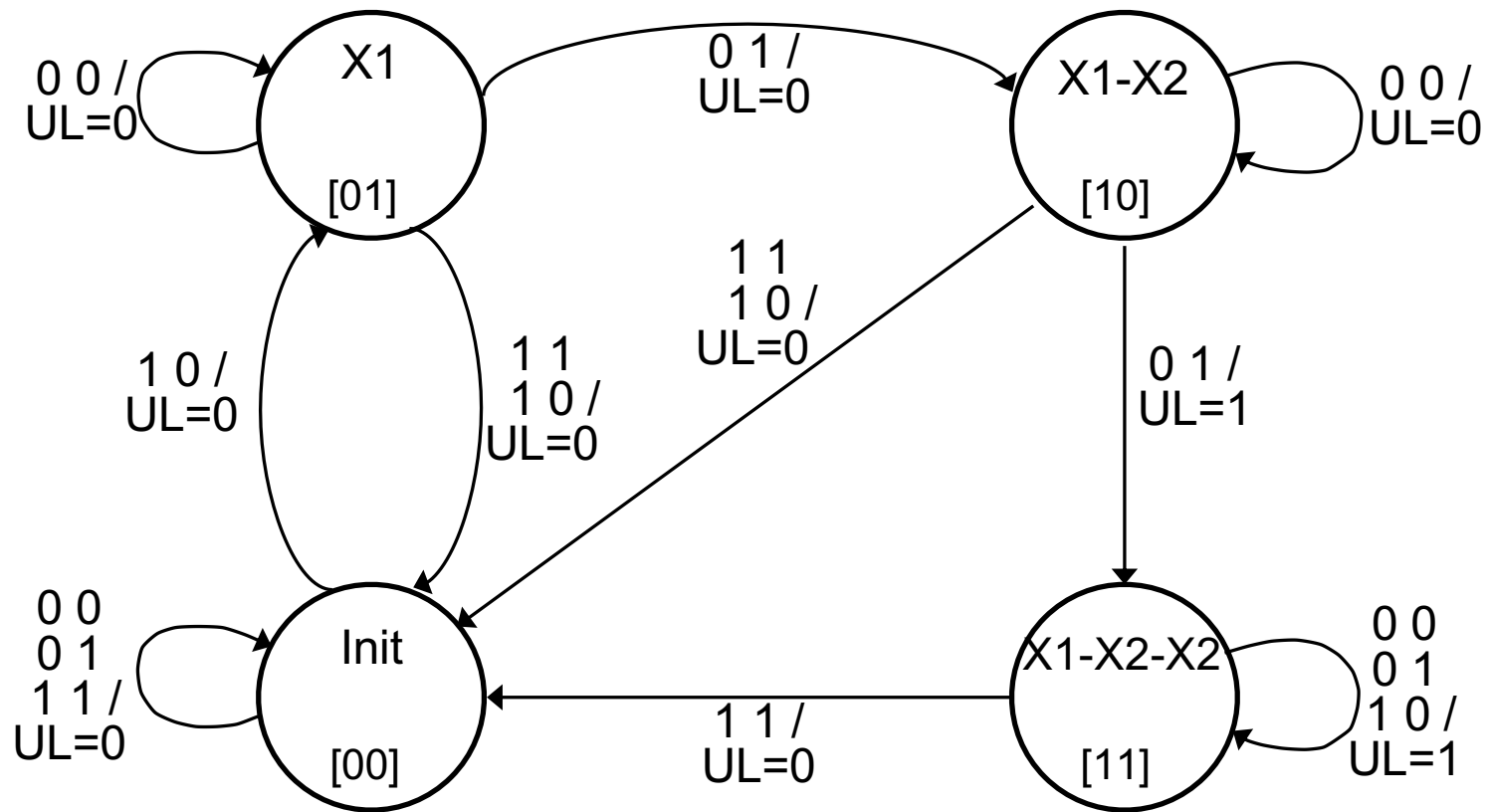
Moore Transition/Output Table 2

$S_1 S_0$	$S_1^* S_0^*$				UL
	Input 0 0	Input 0 1	Input 1 0	Input 1 1	
0 0	0 0	0 0	0 1	0 0	0
0 1	0 1	1 0	0 0	0 0	0
1 0	1 0	1 1	0 0	0 0	0
1 1	1 1	1 1	1 1	0 0	1

- Version 2: uses state binary encodings

Pushbutton Lock: Mealy State Diagram

- Output: $UL=1$ with sequence $X1, X2, X2$
- Input: 00 (neither), 10 ($X1$), 01 ($X2$), 11 (reset)



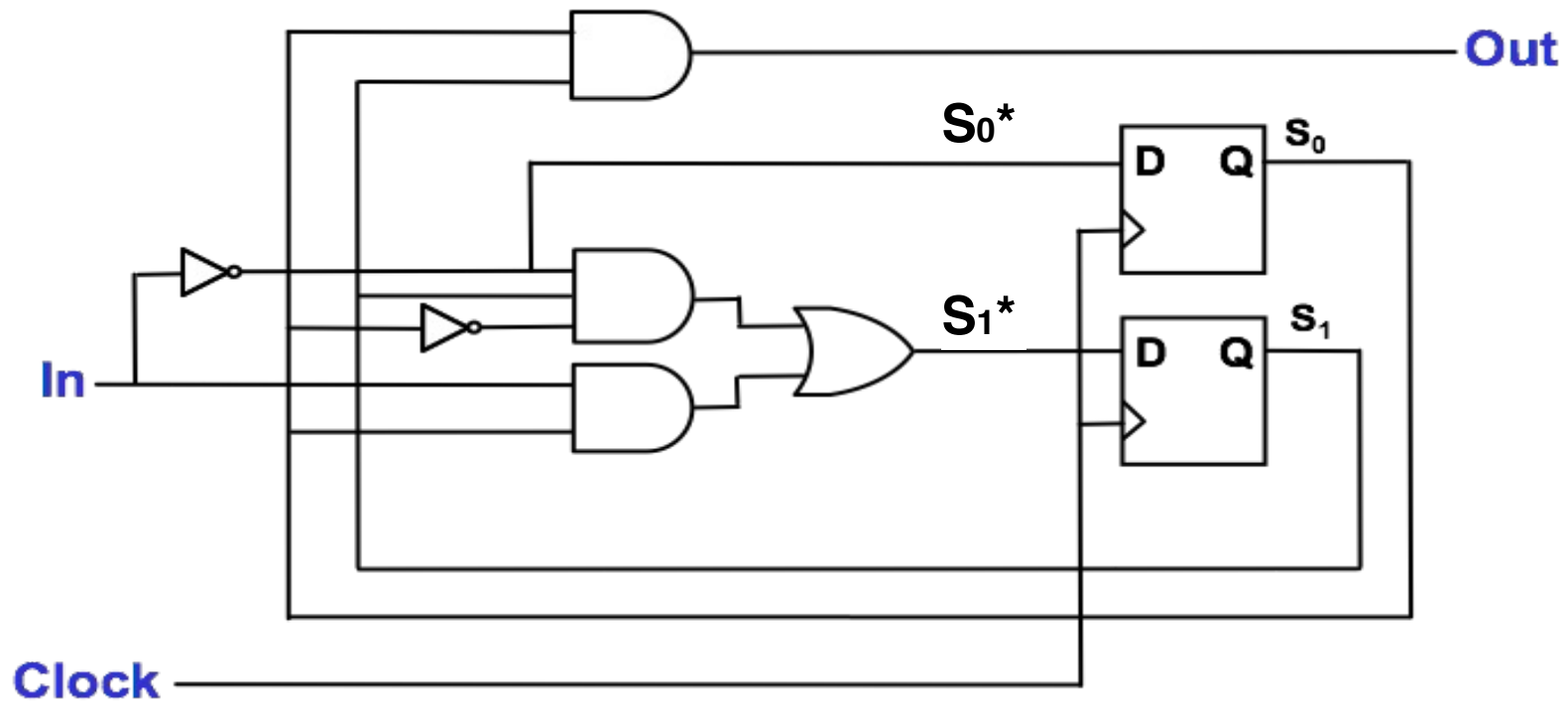
Mealy Transition/Output Table

$S_1 S_0$	$S_1^* S_0^* , UL$			
	Input 0 0	Input 0 1	Input 1 0	Input 1 1
0 0	0 0, 0	0 0, 0	0 1, 0	0 0, 0
0 1	0 1, 0	1 0, 0	0 0, 0	0 0, 0
1 0	1 0, 0	1 1, 1	0 0, 0	0 0, 0
1 1	1 1, 1	1 1, 1	1 1, 1	0 0, 0

- uses state binary encodings

Analyzing the Sequential Logic

What does this circuit do?

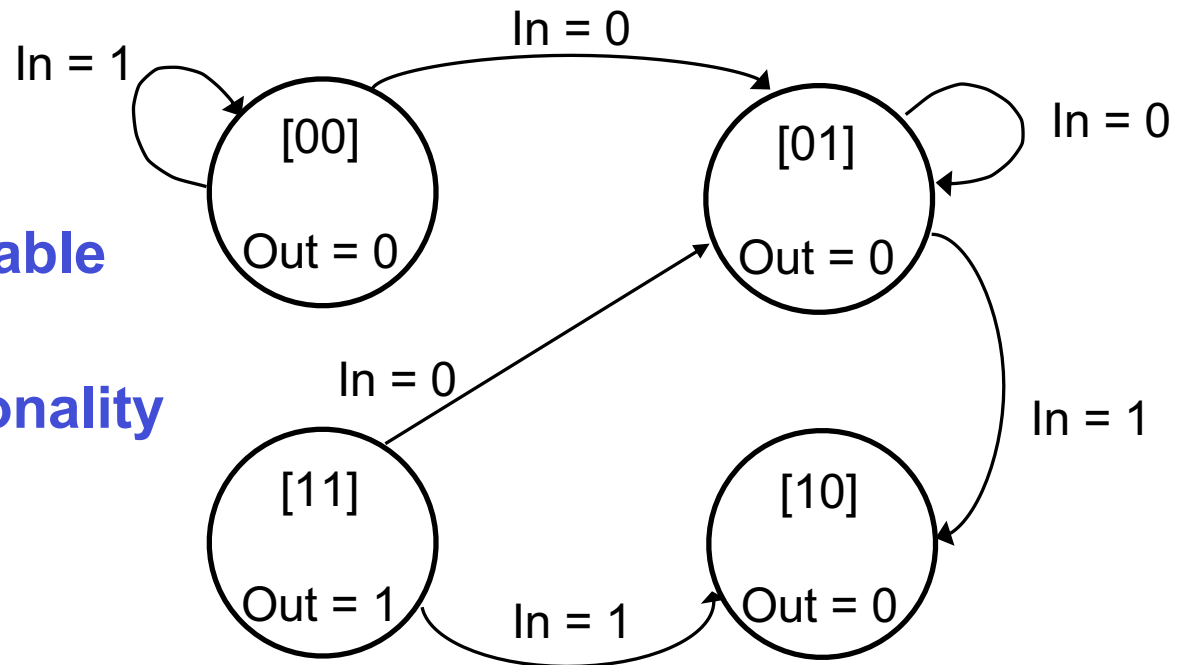


Write down
transition and
output equations

$S_0^* =$
 $S_1^* =$
 $Out =$

Reconstruct State Diagram

- Complete the transition/output table and state diagram
- Identify the functionality of the FSM



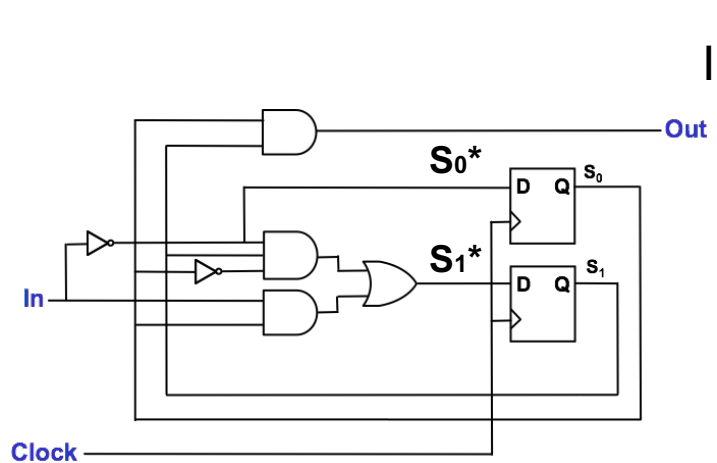
$$S_0^* = In'$$

$$S_1^* = In' \cdot S_1 \cdot S_0' + In \cdot S_0$$

$$Out = S_1 \cdot S_0$$

$S_1 S_0$	$S_1^* S_0^*$		Out
	In = 0	In = 1	
0 0	0 1	0 0	0
0 1	0 1	1 0	0
1 0			0
1 1	0 1	1 0	1

Another Pattern Detector

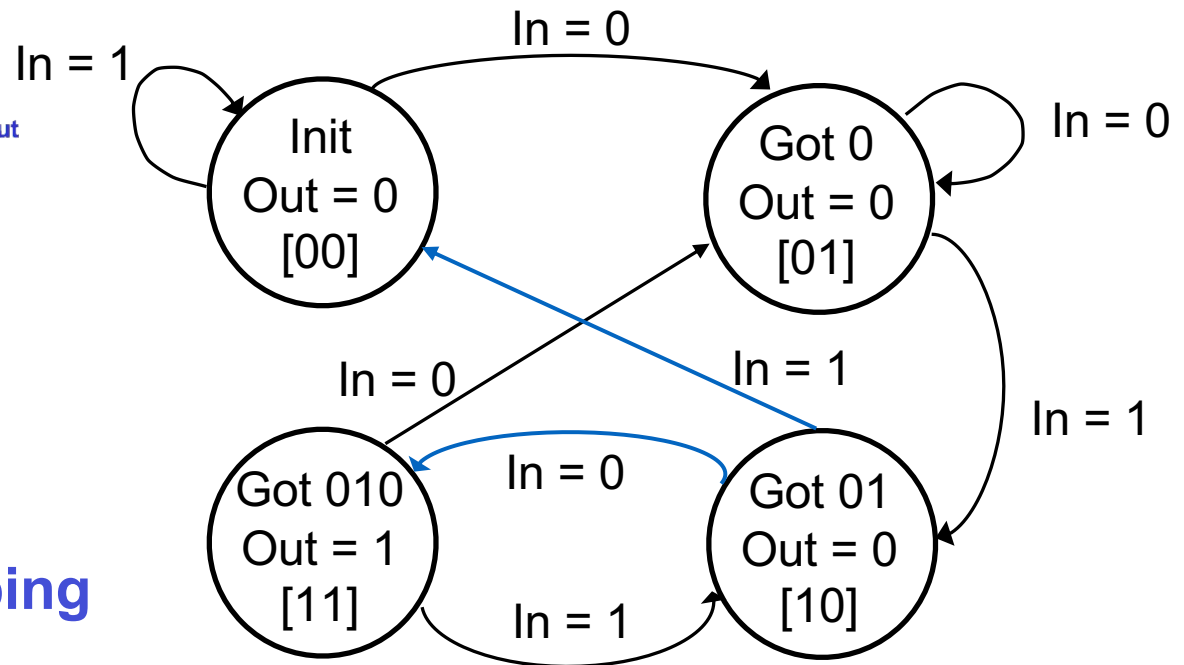


Detects 010 (overlapping patterns included)

$$S_0^* = In'$$

$$S_1^* = In' \cdot S_1 \cdot S_0' + In \cdot S_0$$

$$Out = S_1 \cdot S_0$$



$S_1 S_0$	$S_1^* S_0^*$		Out
	In = 0	In = 1	
00	01	00	0
01	01	10	0
10	11	00	0
11	01	10	1

Tuesday (2/25): Verilog/Quartus Tutorial

Thursday (2/27): In-Class Prelim 1

Next Lecture

More FSMs

Timing

Clocking

(H&H 2.9, 3.4.4)