


Contact Information

Anupam Chattopadhyay

Email: anupam@ntu.edu.sg

Office: N4-02c-105

Go to **wooclap.com** and use the code **AUEBTY** 

Any questions? Send them in now by



- 1 Go to **wooclap.com**
- 2 Enter the event code in the top banner

Event code
AUEBTY

BIG PICTURE

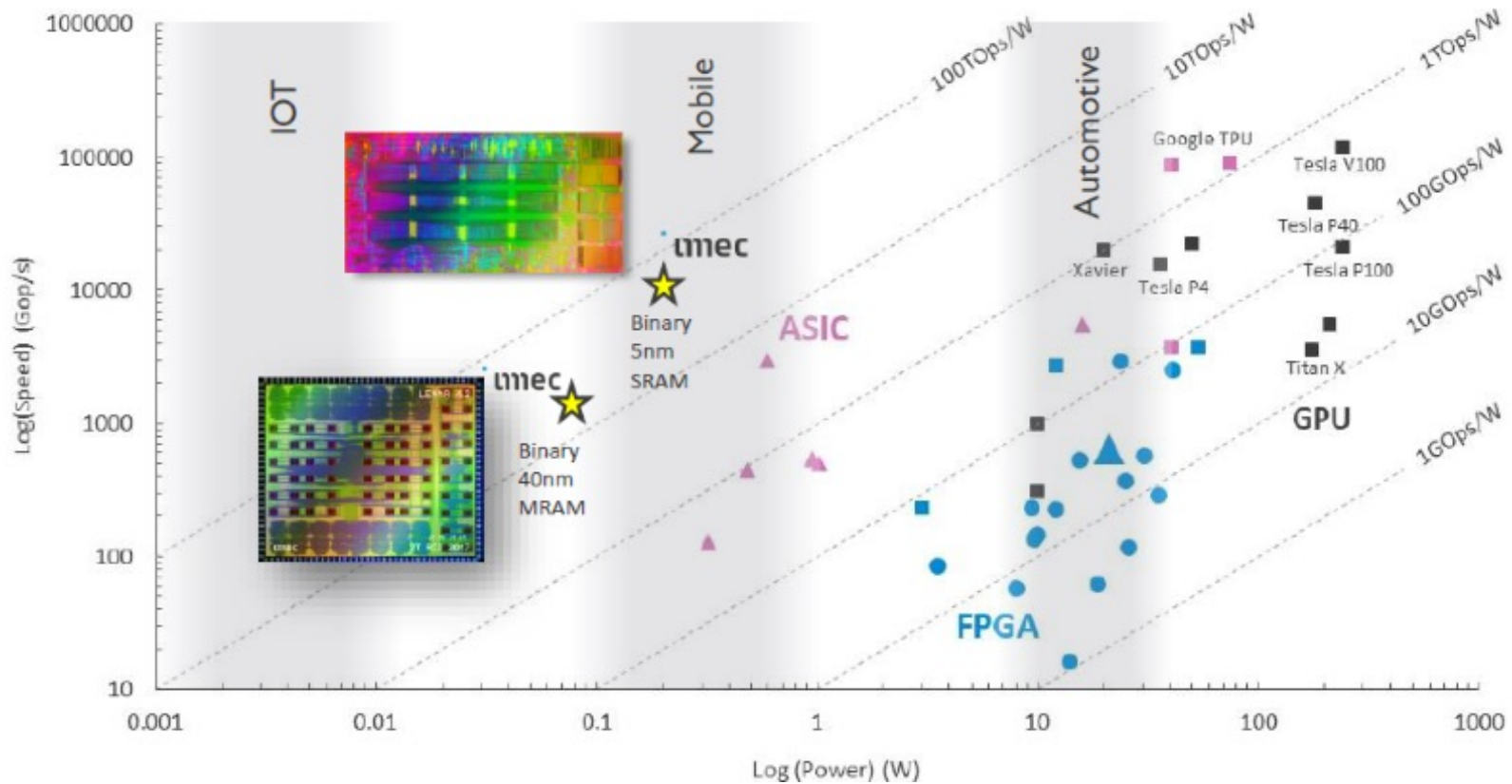


image courtesy: IMEC

BIG PICTURE

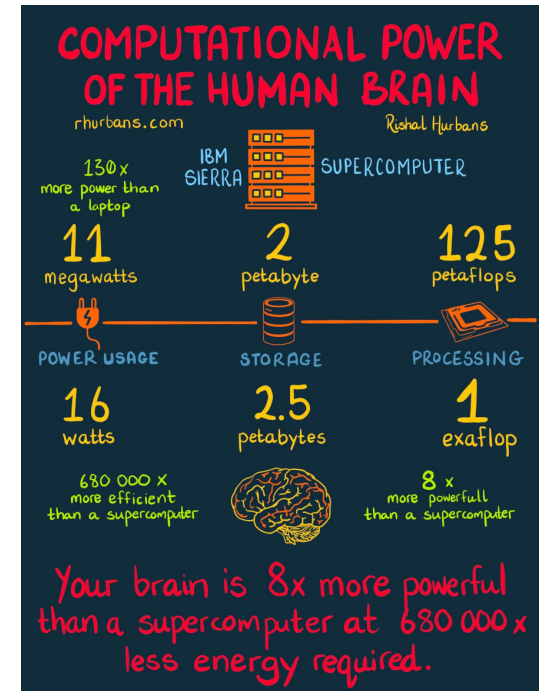
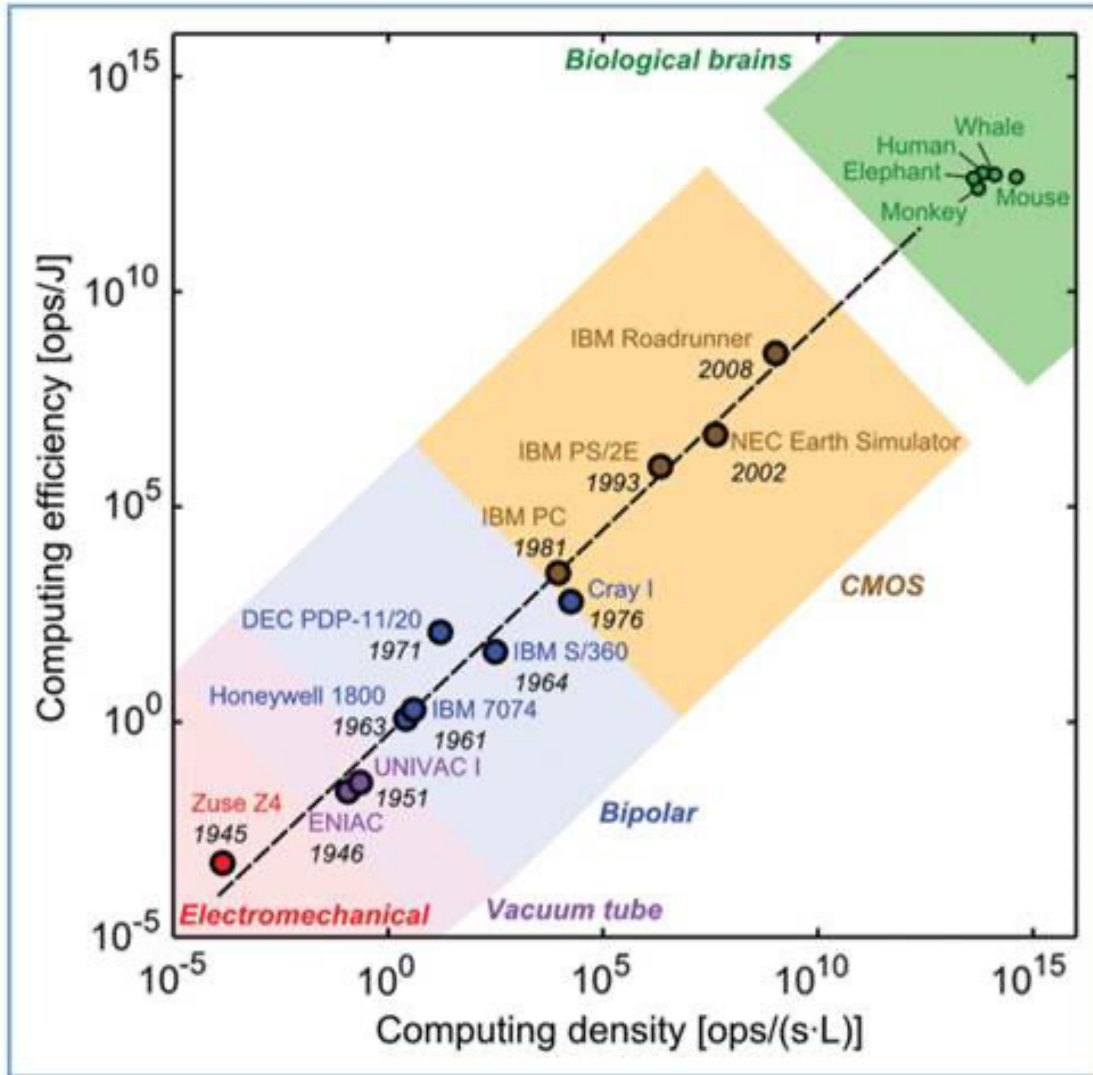
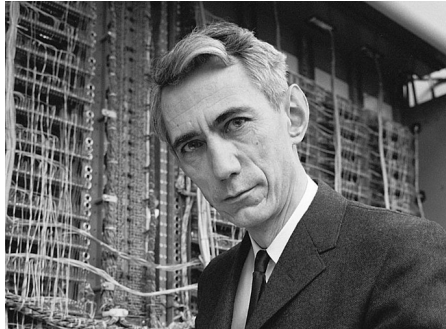
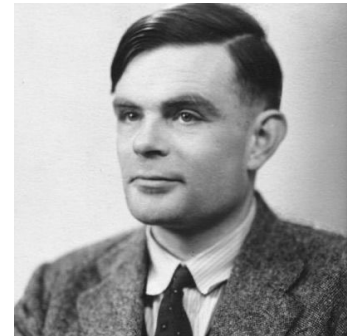
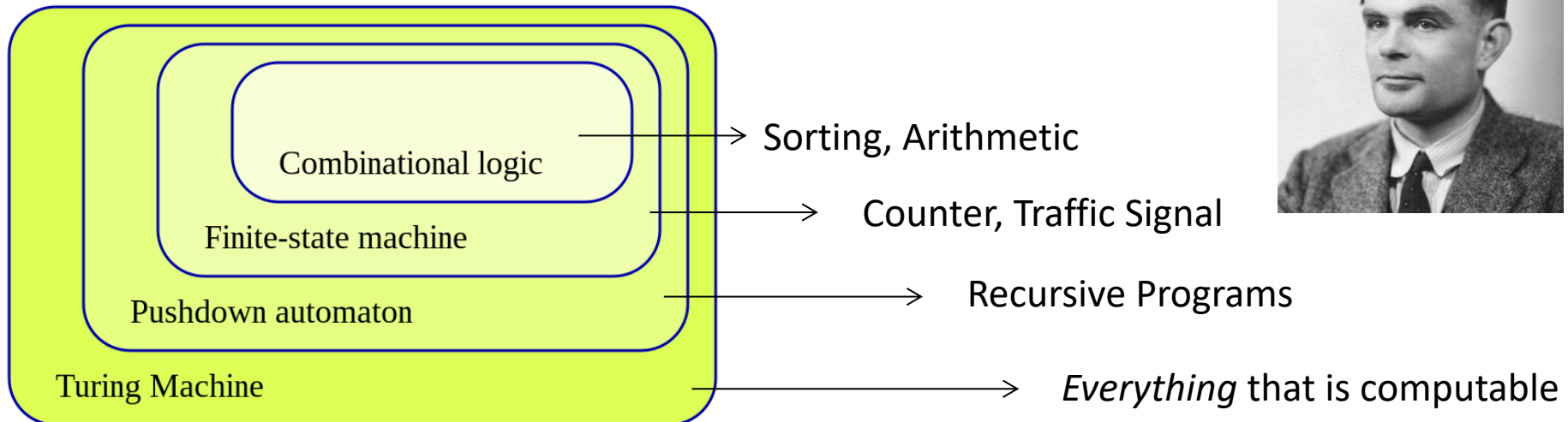


image courtesy: https://www.i-sis.org.uk/The_Computer_Aspires_to_the_Human_Brain.php, <https://rhurbans.com/>

Boolean Algebra using Circuits



Automata theory



SC1005

Digital Logic

Recap and Discussion

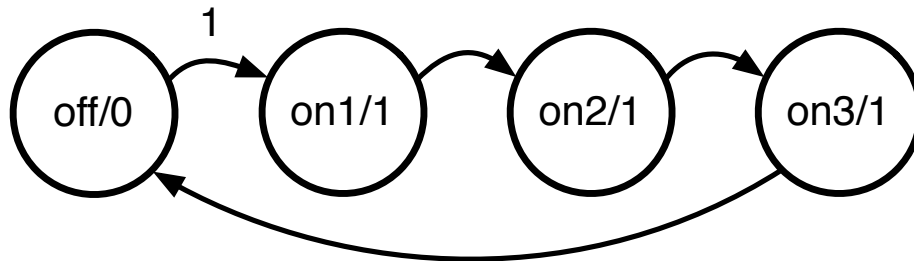
Lecture 21, 22

Finite State Machines

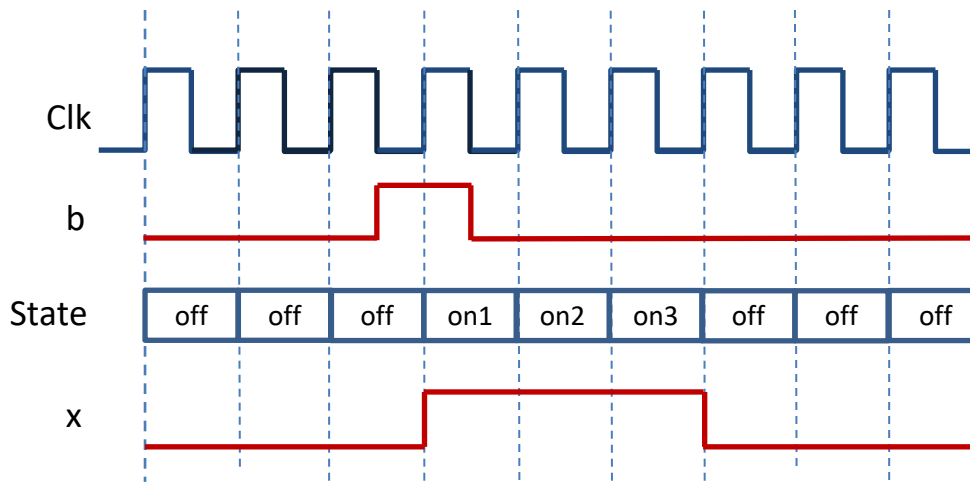
FSM in Verilog

Recap: Finite State Machines

- Each node is a possible state of the system
- Each edge is a transition from one state to the next
- Here, outputs are labelled within the state (Moore FSM)

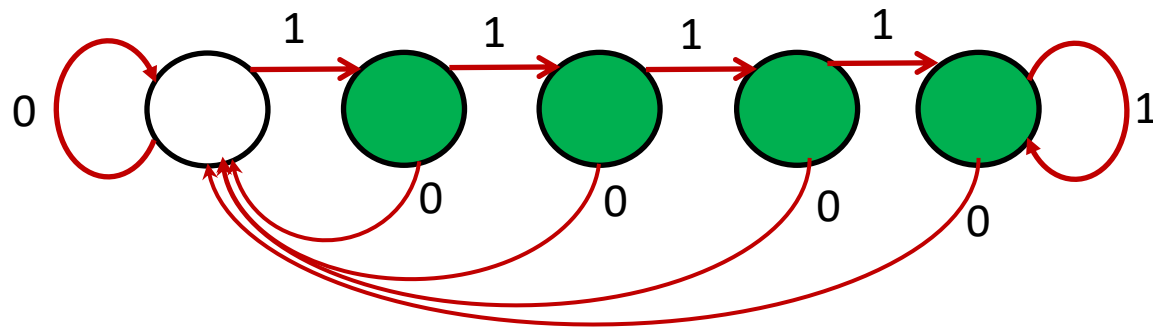
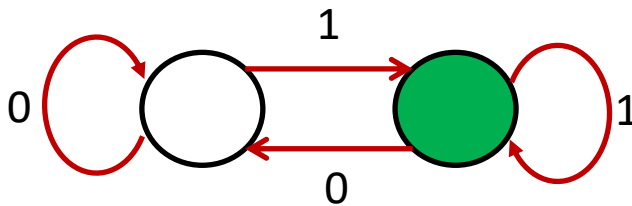
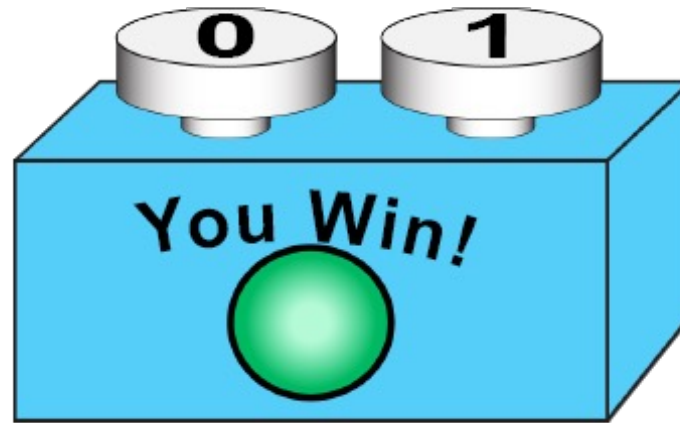


Input: b
Output: x



Current State	b	Next State	Output x
off	0	off	0
off	1	on1	0
on1	0	on2	1
on1	1	on2	1
on2	0	on3	1
on2	1	on3	1
on3	0	off	1
on3	1	off	1

Exercise 1



Are these two FSMs different?

Yes

No



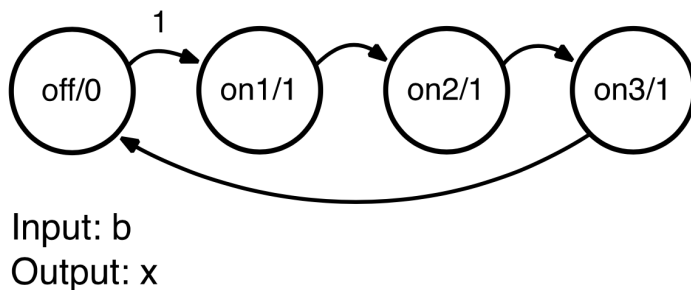
1 Go to wooclap.com

2 Enter the event code in the top banner

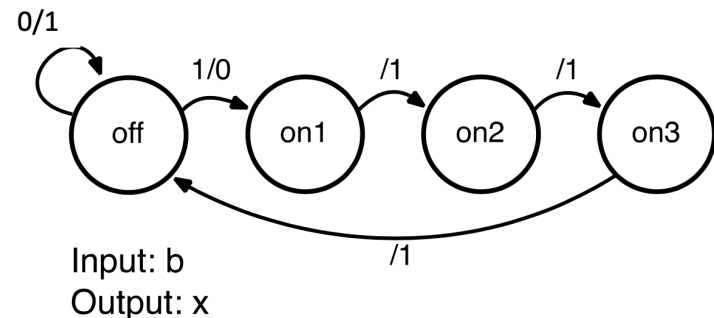
Event code
AUEBTY

Recap: Moore and Mealy Machines

- If the outputs depend only on the state, it is called **Moore Machine**
- Another type of state machine is called a **Mealy Machine**
- In Mealy Machines, the output depends on the state *and the current value of the inputs*



Moore Machine

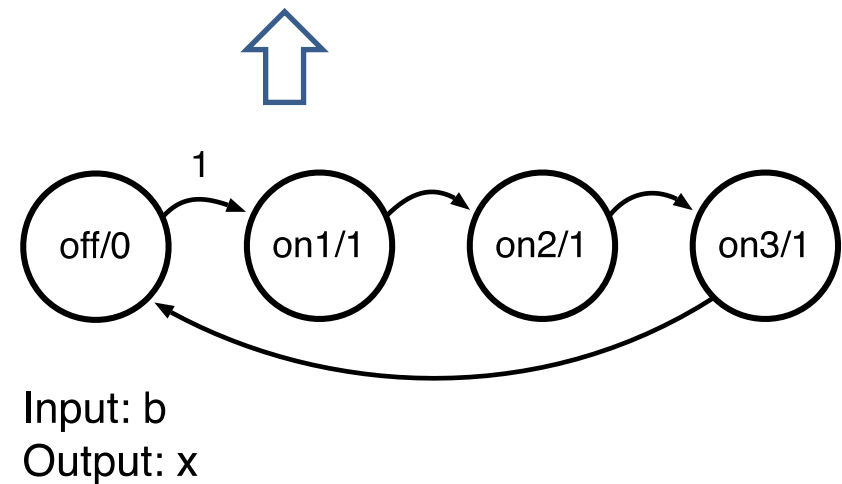
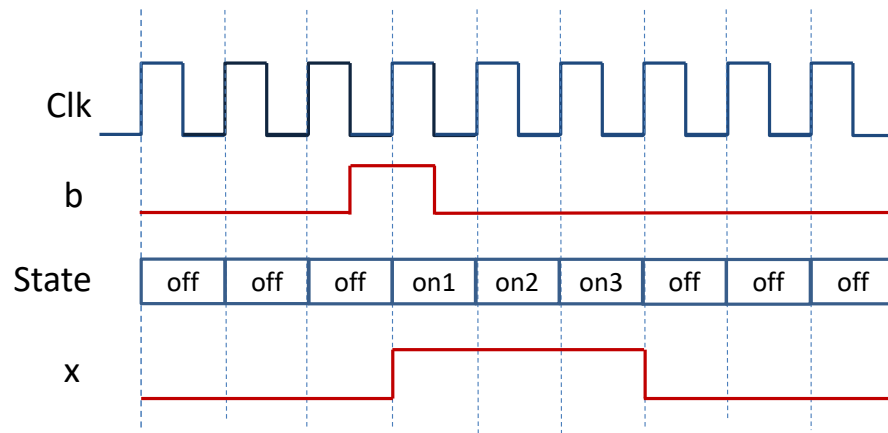
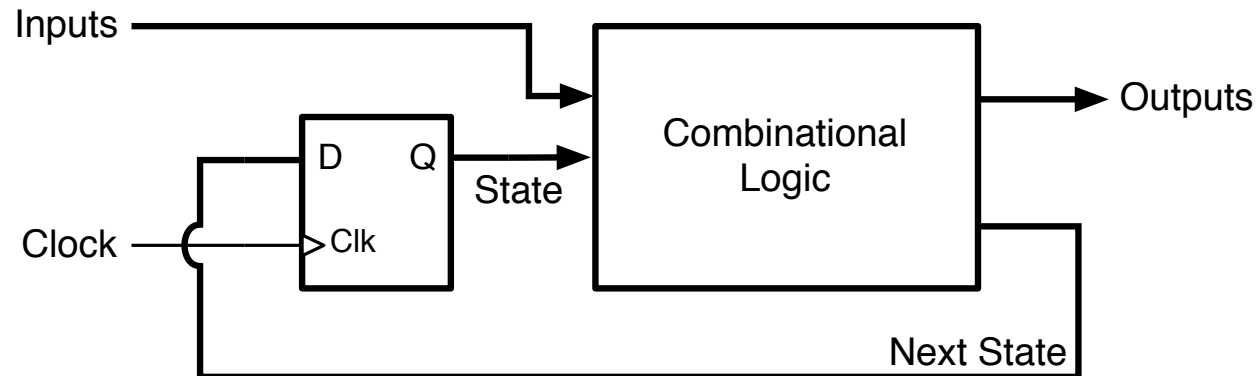


Mealy Machine

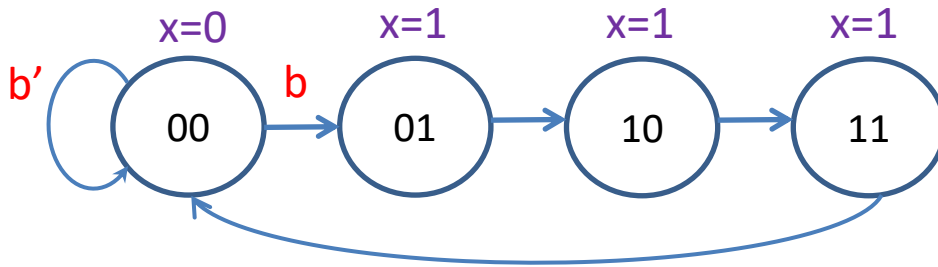
Mapping to Implementation

- Hence, the basic structure looks like this:

- A state register that holds the current state
- Some logic to determine what the next state should be
- Some logic to drive the output correctly



Recap: Mapping Example



Inputs				Outputs		
	s1	s0	b	x	n1	n0
off	0	0	0	0	0	0
	0	0	1	0	0	1
on1	0	1	0	1	1	0
	0	1	1	1	1	0
on2	1	0	0	1	1	1
	1	0	1	1	1	1
on3	1	1	0	1	0	0
	1	1	1	1	0	0

$$x = s1 + s0$$

$$n1 = s1' s0 + s1 s0' = s1 \text{ xor } s0$$

$$n0 = s1' s0' b + s1 s0'$$

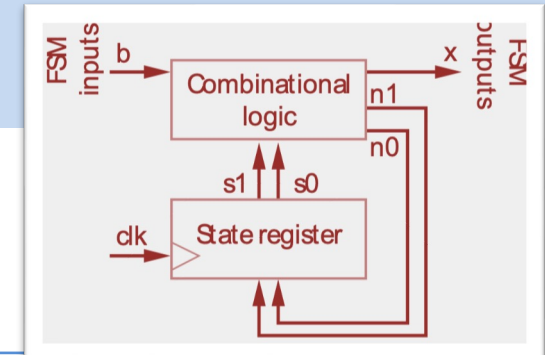
```

module pulse3 (input b,
               input clk, rst,
               output x);

wire n1, n0;
reg s1, s0;

assign n1 = s1 ^ s0;
assign n0 = (~s1&~s0&b) | (s1&~s0);
assign x = s1 | s0;

always @ (posedge clk)
begin
    if(rst) begin
        s1 <= 1'b0;
        s0 <= 1'b0;
    end else begin
        s1 <= n1;
        s0 <= n0;
    end
end
endmodule
  
```



Exercise 2

Which of the following Verilog code segment is correct for the circuit shown?

A

```
assign N[1] = ~(x & S[1]);
assign N[0] = ~S[1] | ~S[0];
assign out  = S[1] ^ S[0];
```

```
always @ (posedge clk)
begin
    if(Rst)
        N <= 2'd0;
    else
        N <= S;
end
```

B

```
assign N[1] = ~x & ~S[1];
assign N[0] = ~(S[1] | S[0]);
assign out  = S[1] ^ S[0];
```

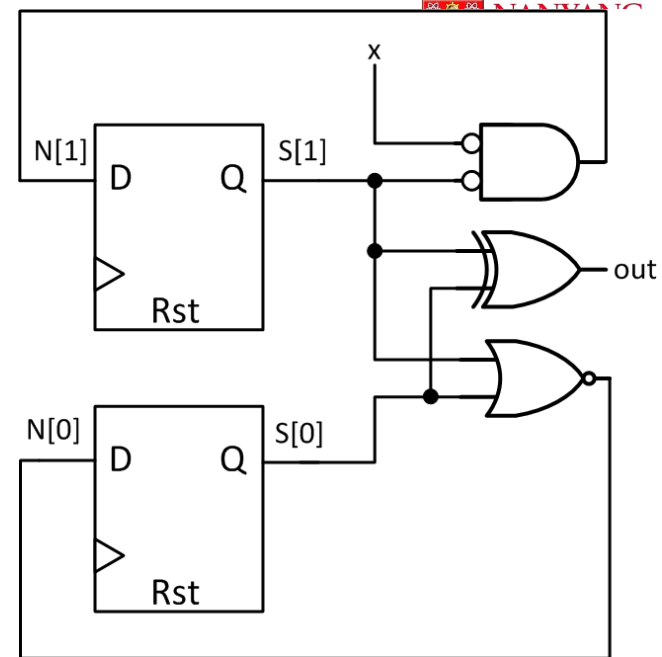
```
always @ (posedge clk)
begin
    if(Rst)
        N <= 2'b00;
    else
        N <= S;
end
```

```
assign N[1] = ~(x & S[1]);
assign N[0] = ~S[1] | ~S[0];
assign out  = S[1] ^ S[0];
```

```
always @ (posedge clk)
begin
    if(Rst)
        S <= 2'b00;
    else
        S <= N;
end
```

```
assign N[1] = ~x & ~S[1];
assign N[0] = ~(S[1] | S[0]);
assign out  = S[1] ^ S[0];
```

```
always @ (posedge clk)
begin
    if(Rst)
        S <= 2'd0;
    else
        S <= N;
end
```



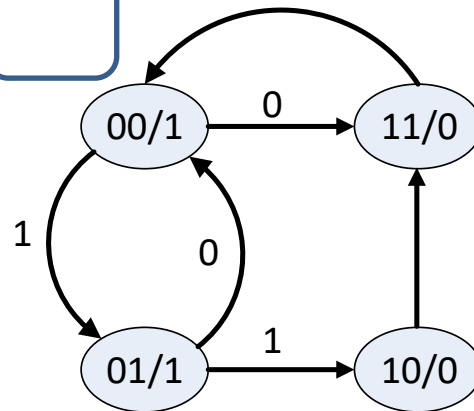
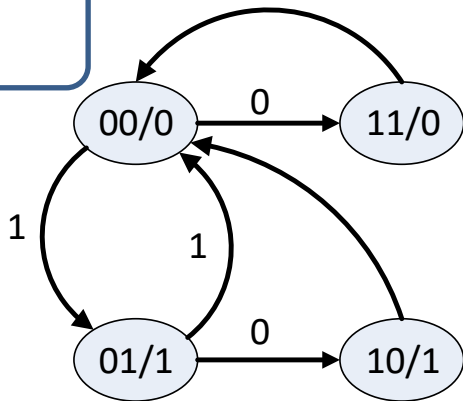
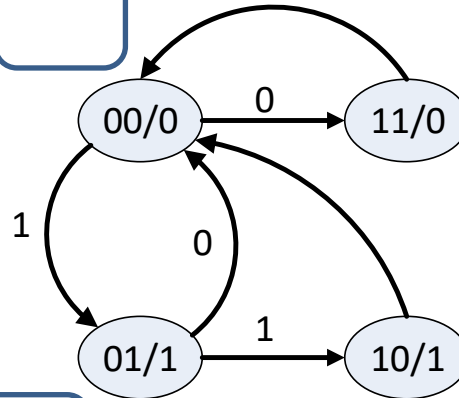
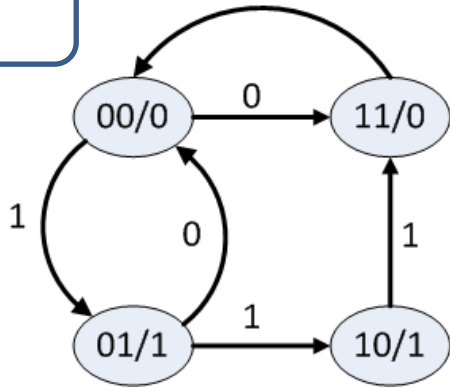
1 Go to wooclap.com

2 Enter the event code in the top banner

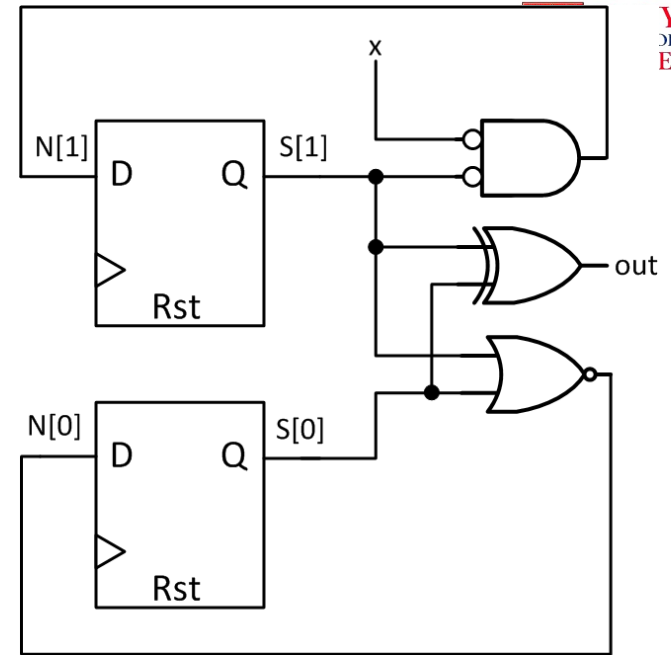
Event code
AUEBTY

Exercise 3

Which of the following FSM corresponds to the given circuit?



Input: x; Output: out

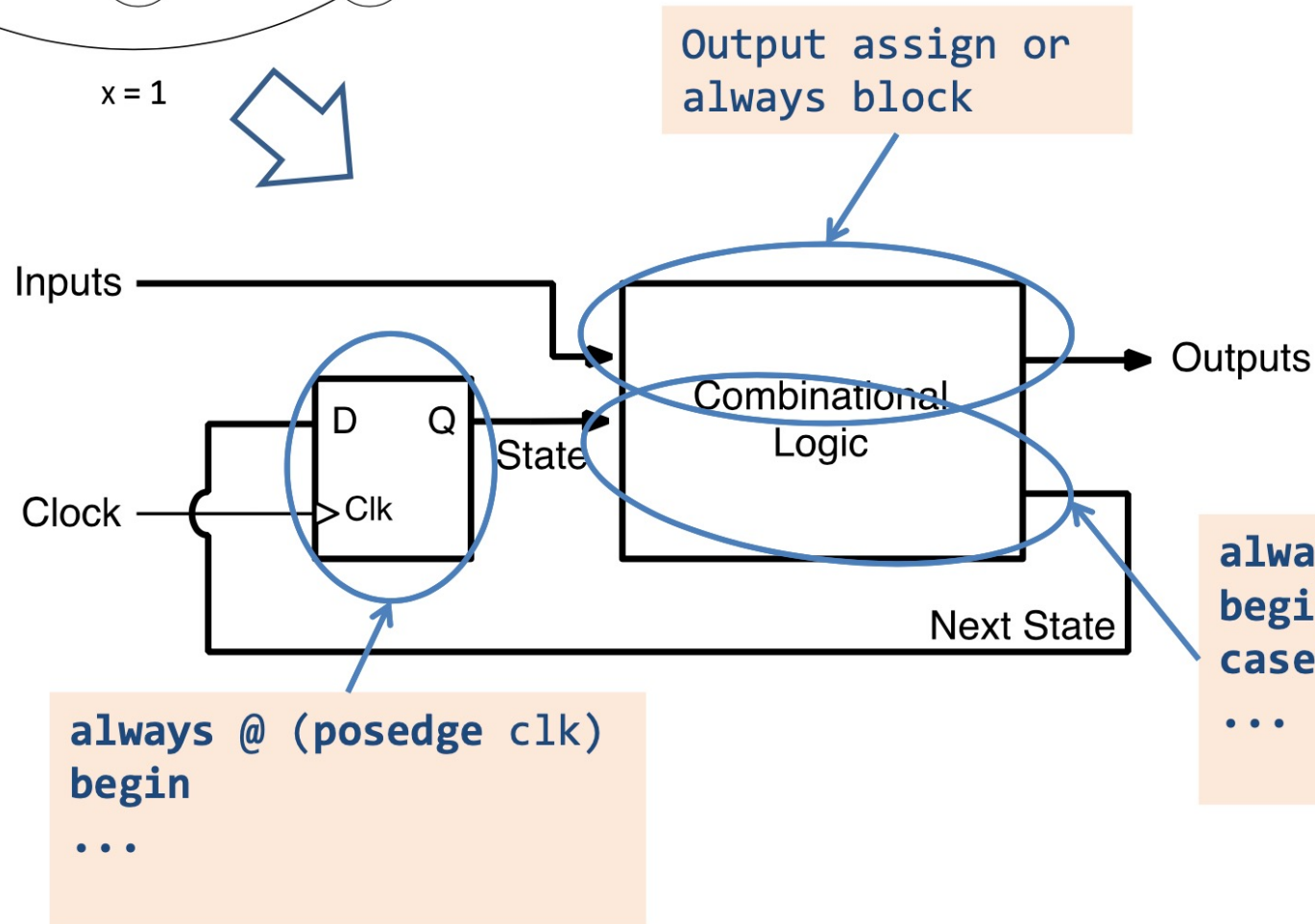
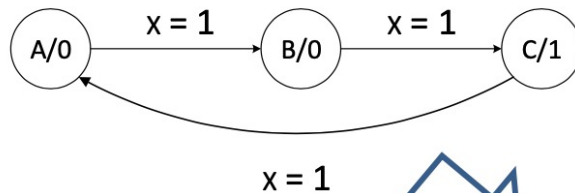


1 Go to wooclap.com

2 Enter the event code in the top banner

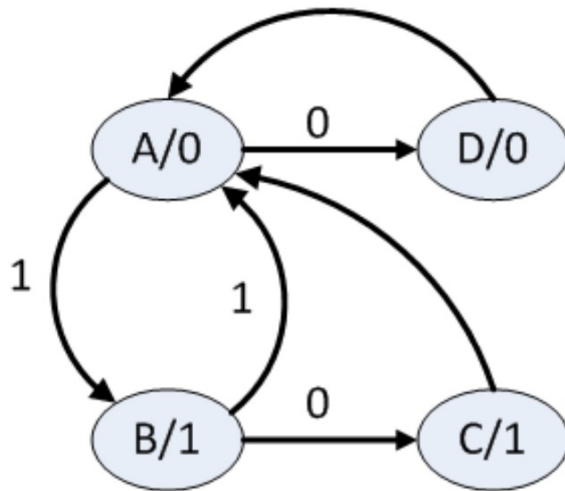
Event code
AUEBTY

FSM Implementation with Behavioral Models



Exercise 4

Write the Verilog module for the given FSM.



Input: x

Output: out

```
module fsm(input x, clk, rst,
           output reg out);

parameter A=2'b00, B=2'b01, C=2'b10, D=2'b11;
reg [1:0] nst, st;

Always @ (posedge clk)
begin
    if (rst)
        st <= A;
    else
        st <= nst;
end

always @ *
begin
    nst = st;
    out = 1'b0;
    case (st)
        A: if(x) nst = B;
           else nst = D;
        B: begin
            if(x) nst = A;
            else nst = C;
            out = 1'b1;
        end
        C: begin
            nst = A;
            out = 1'b1;
        end
        D: nst = A;
    endcase
end

endmodule
```

END OF L21, L22 SUMMARY