NANYANG TECHNOLOGICAL UNIVERSITY

SC1005
Digital Logic

Recap and Discussion

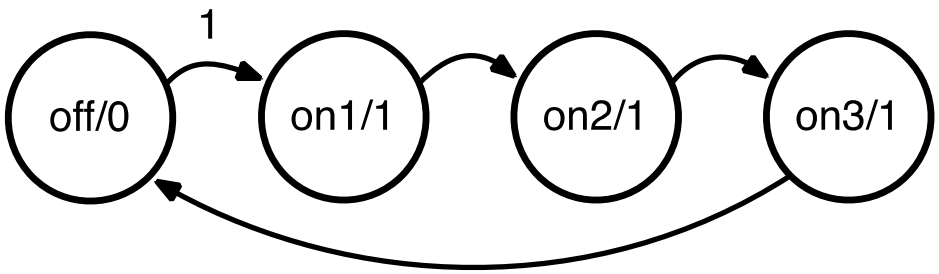
Lecture 21 and 22

Finite State Machine,

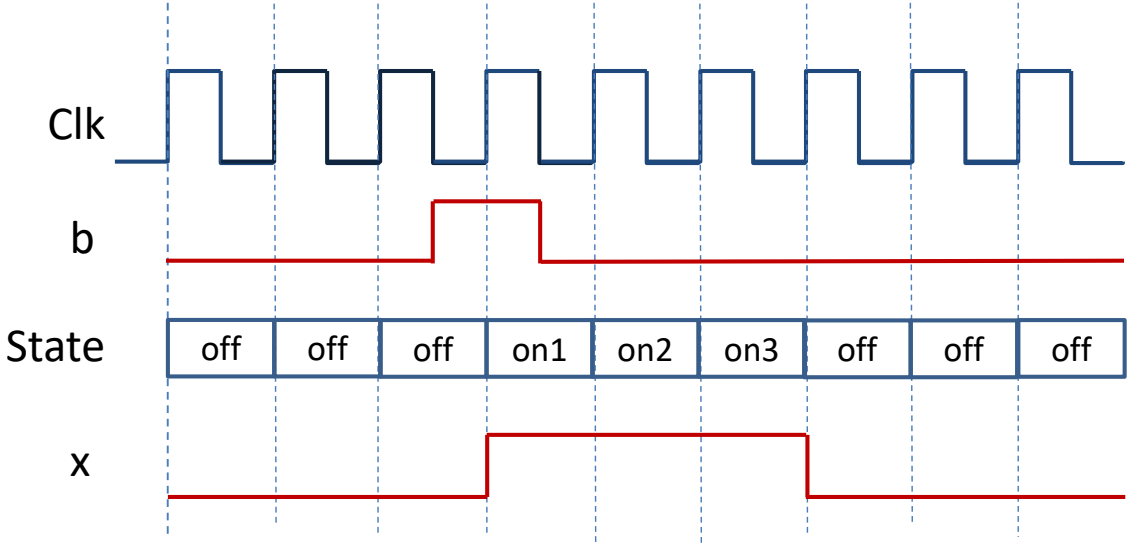
FSM in Verilog

Recap: Finite State Machines

- A *finite state machine (FSM)* describes a system using a finite number of states and the associated transitions
- In synchronous design, an FSM is in one state for the duration of each clock cycle
- At every rising clock edge, the FSM may transition to another state depending on the input values at that rising edge



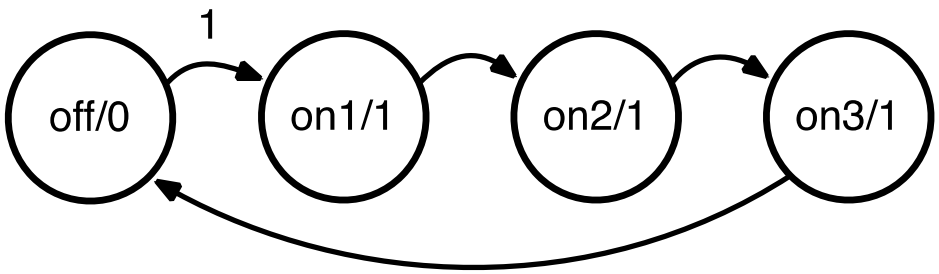
Input: b
Output: x



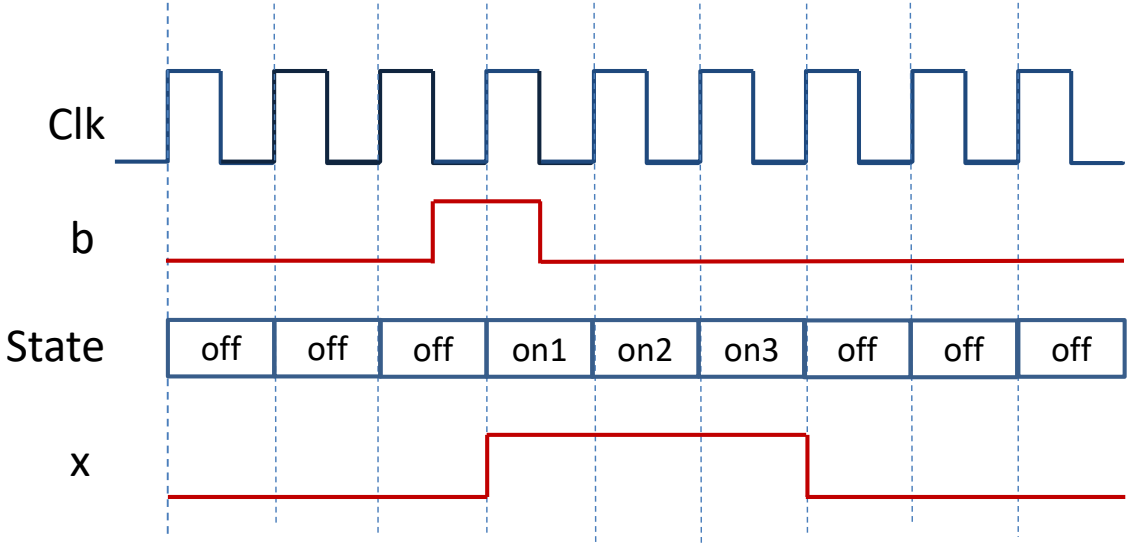
Current State	b	Next State	Output x
off	0	off	0
off	1	on1	0
on1	0	on2	1
on1	1	on2	1
on2	0	on3	1
on2	1	on3	1
on3	0	off	1
on3	1	off	1

Recap: Finite State Machines

- A *finite state machine (FSM)* describes a system using a finite number of states and the associated transitions
- In synchronous design, an FSM is in one state for the duration of each clock cycle
- At every rising clock edge, the FSM may transition to another state depending on the input values at that rising edge



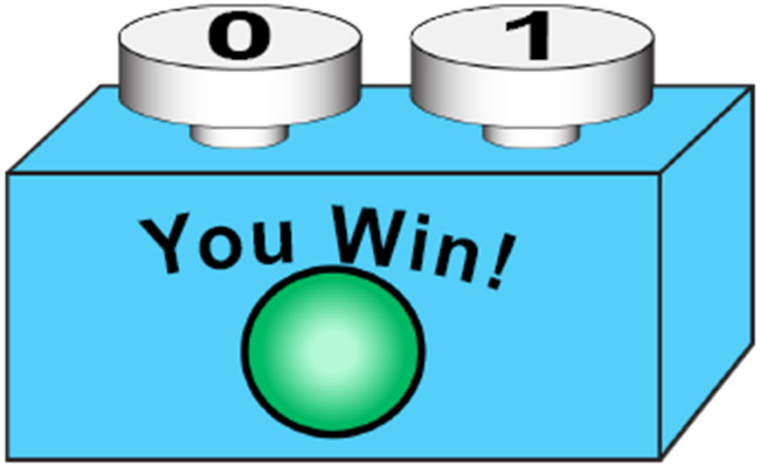
Input: b
Output: x

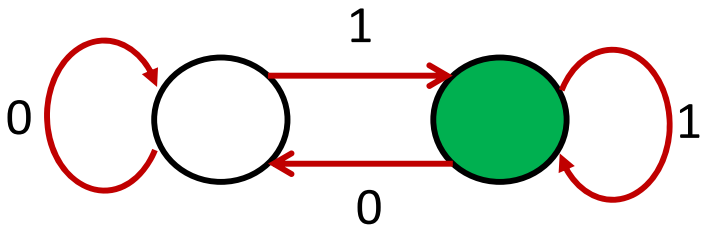


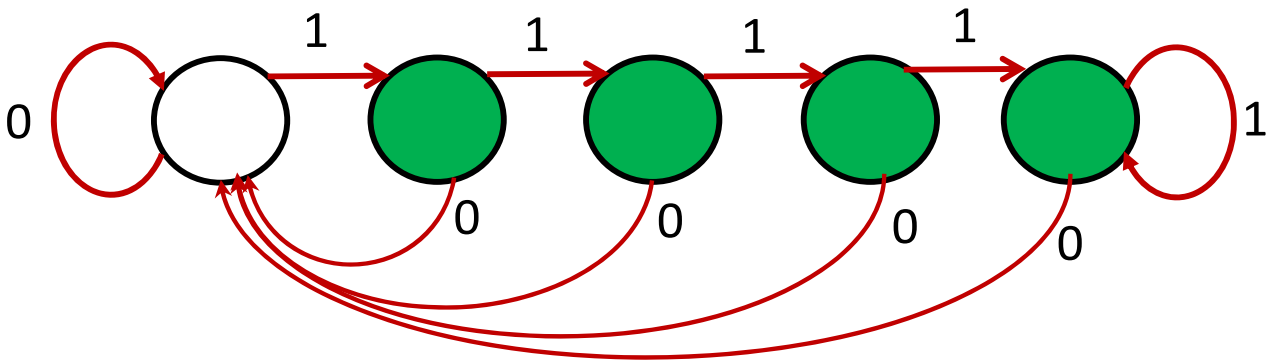
Current State	Next State b = 0	1	Output X
off	**off**	on1	0
on1	on2	on2	1
on2	on3	on3	1
on3	off	off	1

It is often easier to visualize when the state transition table is represented in this form

Exercise 1







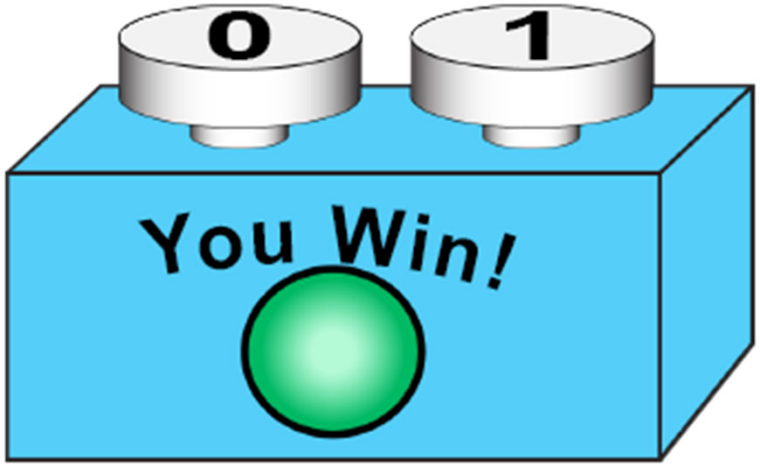
Are these two FSMs different?

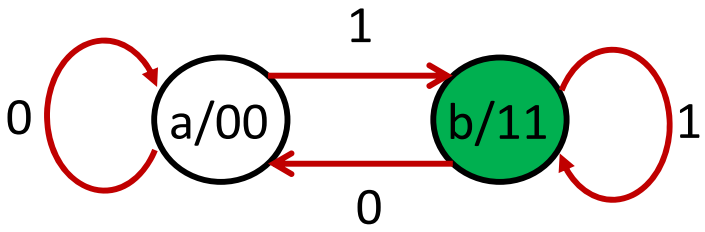
A. Yes

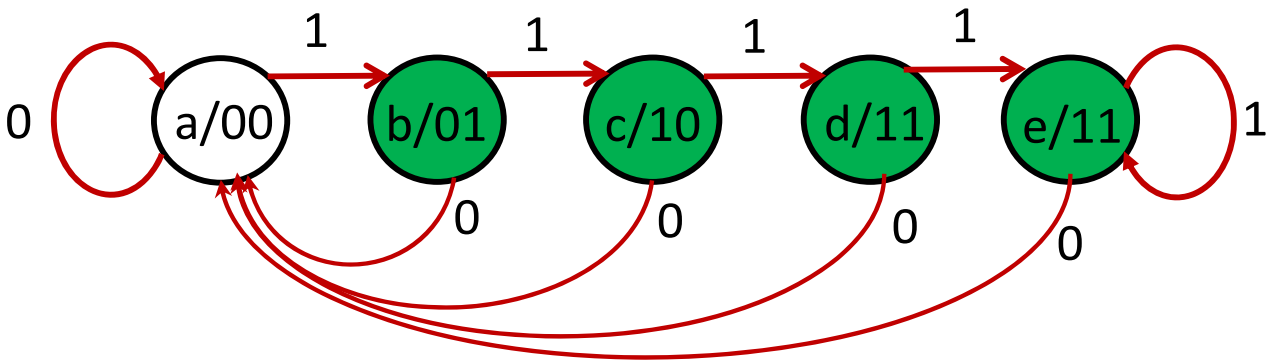
B. No

Ans: No. They are the same.
You can use state reduction techniques to reduce the RHS FSM to the LHS FSM.

Exercise 1a







Are these two FSMs different?

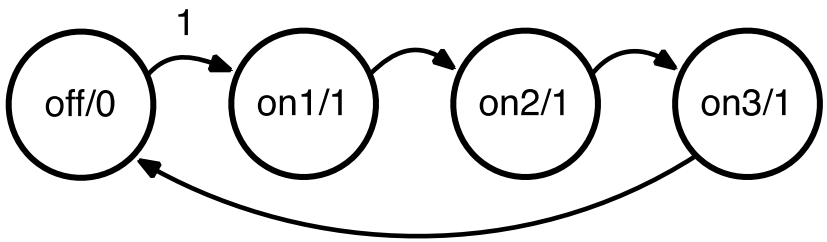
A. Yes

B. No

Ans: Yes, they are different.
You can only use state reduction techniques to combine states d and e, due to the others having different outputs.

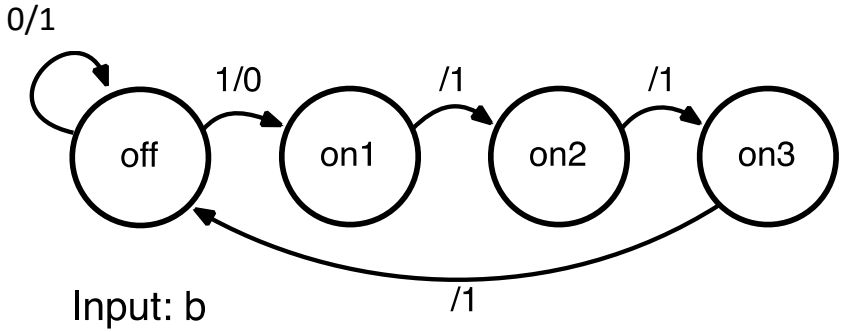
Recap: Moore and Mealy Machines

- Moore Machines: The outputs depend only on the state, i.e. in state *x*, the output is always *f*
- Mealy Machines: The output depends on the state and the current value of the inputs
  - Outputs can change mid-state, if the inputs change



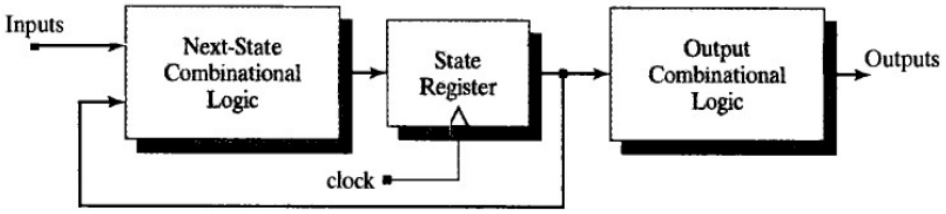
Input: b
Output: x

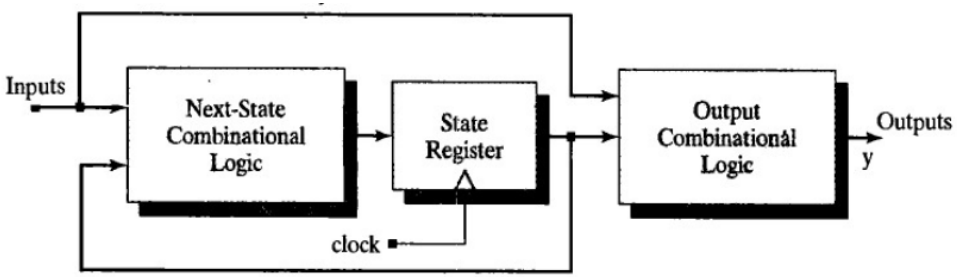
Moore Machine



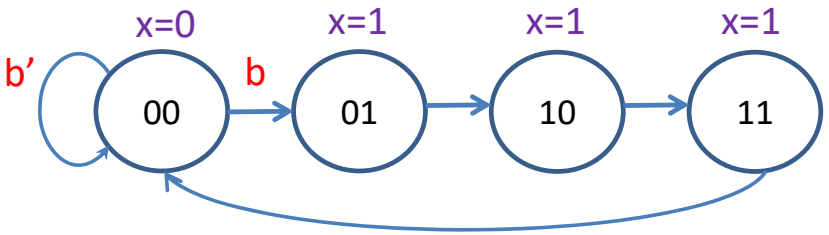
Input: b
Output: x

Mealy Machine





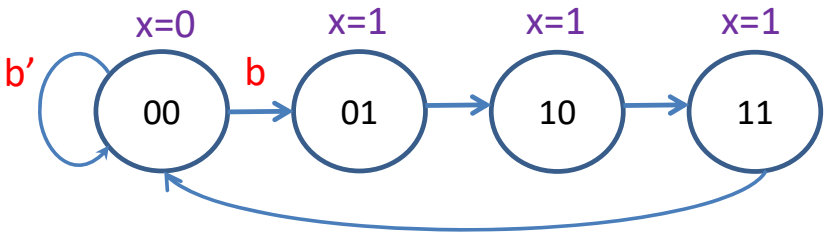
Mapping Example 1



	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

n1 = s1’s0 + s1s0’ = s1 xor s0

Mapping Example 1



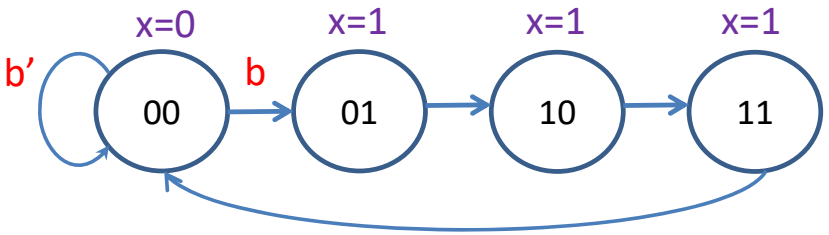
	Inputs			Outputs		
	s1	s0	b	x	n1	n0
off	0	0	0	0	0	0
	0	0	1	0	0	1
on1	0	1	0	1	1	0
	0	1	1	1	1	0
on2	1	0	0	1	1	1
	1	0	1	1	1	1
on3	1	1	0	1	0	0
	1	1	1	1	0	0

n1 = s1' s0 + s1s0' = s1 xor s0

n0 = s1' s0' b + s1s0' = s0' b + s1s0'

x = s1 + s0

Mapping Example 1



	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

n1 = s1' s0 + s1s0' = s1 xor s0

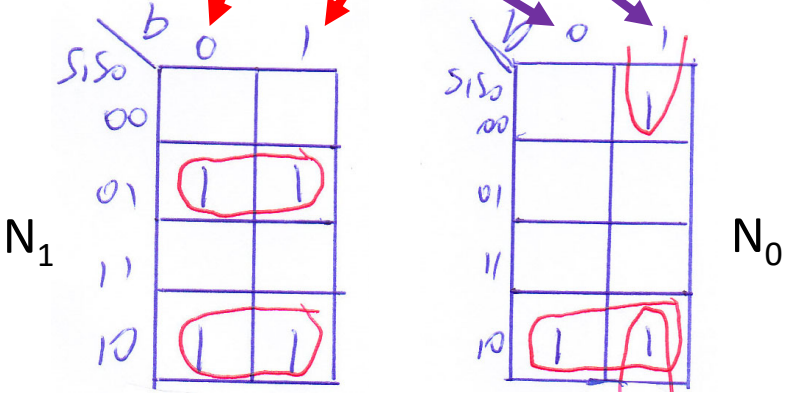
n0 = s0' b + s1s0'

x = s1 + s0

But, it may be easier to redraw the transition table from before (now referred to as an Excitation Table).

	State S_1S_0	Next state N_1N_0 b = 0	Next state N_1N_0 1	Output (x)
off	0 0	0 0	0 1	0
on1	0 1	1 0	1 0	1
on3	1 1	0 0	0 0	1
on2	1 0	1 1	1 1	1

Then the next state & output expressions are:



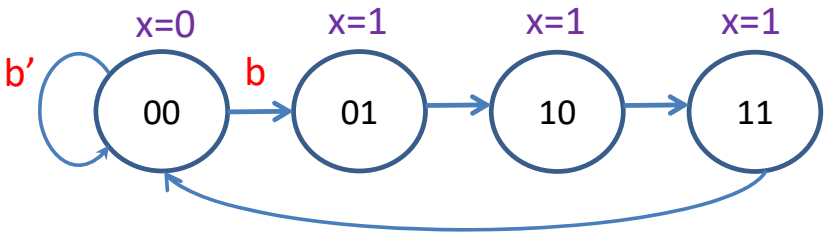
$N_1 = S_1' S_0 + S_1 S_0' = S_1 \text{ xor } S_0$

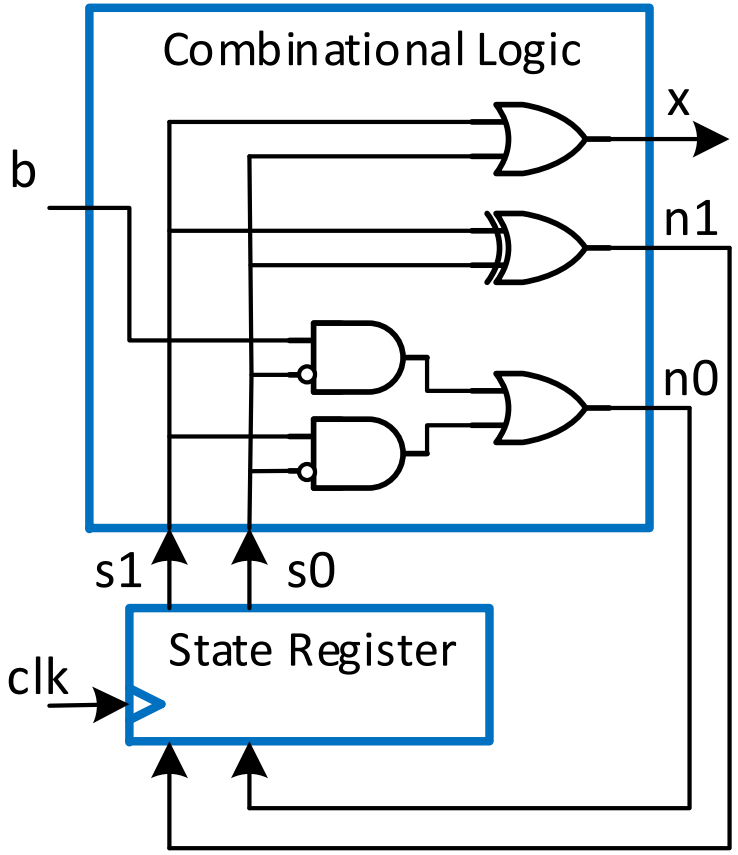
$N_0 = S_0' b + S_1 S_0'$

$x = S_1 + S_0$

Same as before.
Also note grey code order in Ex Table.

Mapping Example 1

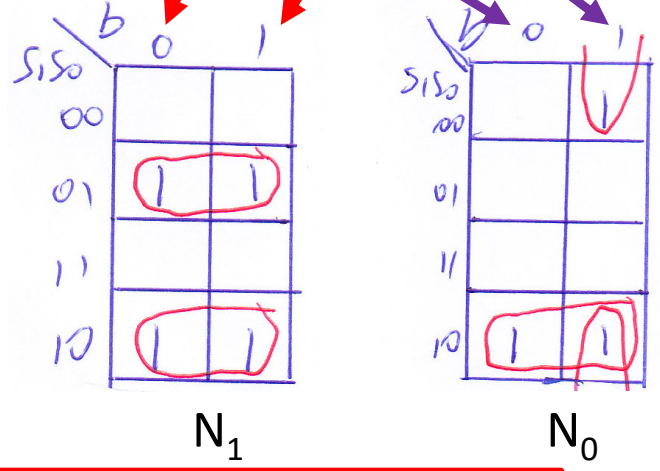




- But, it may be easier to redraw the transition table in a different form.

	State S_1S_0	Next state N_1N_0 b = 0	Next state N_1N_0 1	Output (x)
off	0 0	0 0	0 1	0
on1	0 1	1 0	1 0	1
on3	1 1	0 0	0 0	1
on2	1 0	1 1	1 1	1

Then the next state & output expressions are:



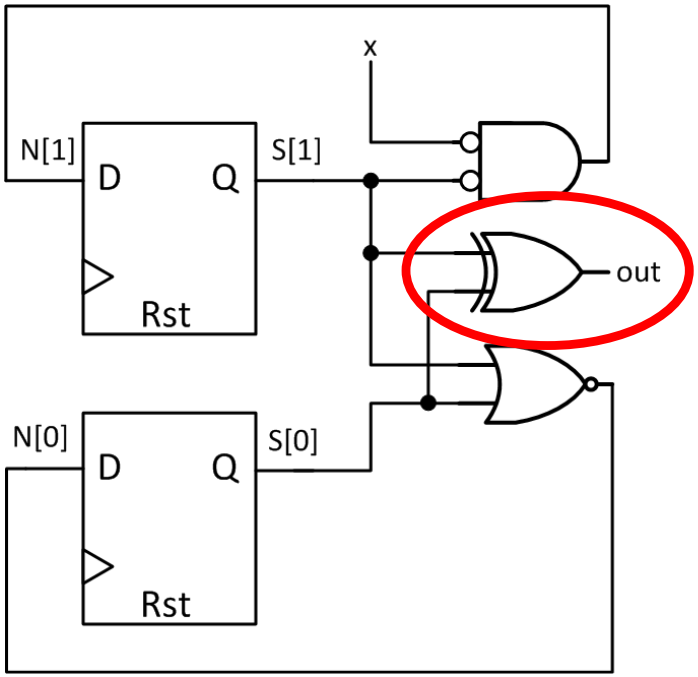
$N_1 = S_1' S_0 + S_1 S_0' = S_1 \text{ xor } S_0$

$N_0 = S_0' b + S_1 S_0'$

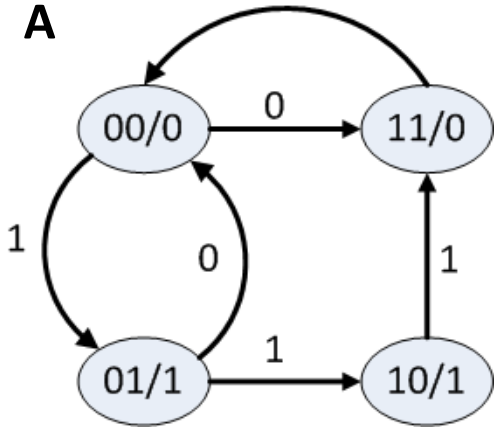
$x = S_1 + S_0$

Exercise 2

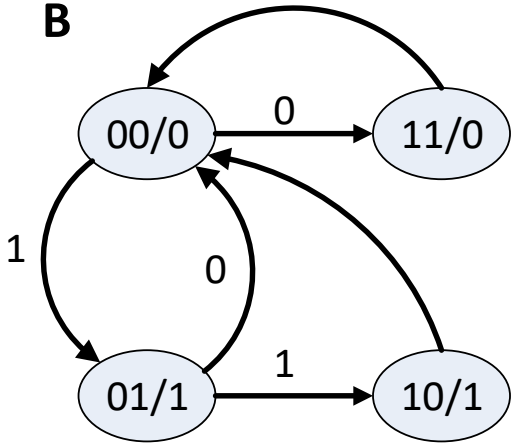
Which of the following FSMs corresponds to the given circuit?



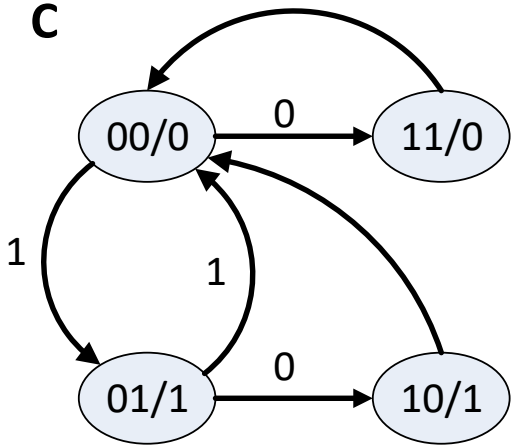
A



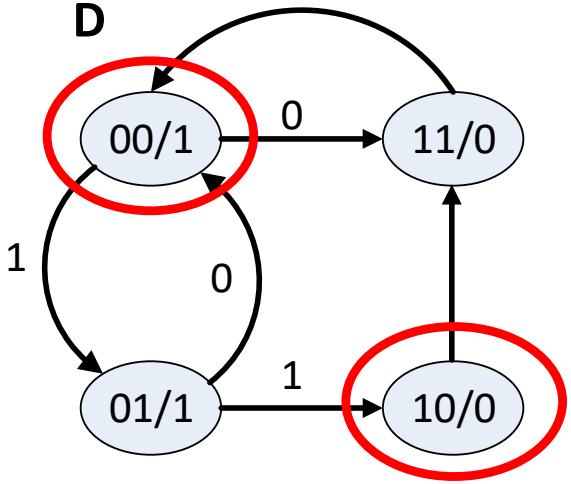
B



C



D



Input: x
Output: out

D is obviously wrong as the output is incorrect.
When in State **01**, a 1 on x will result in:

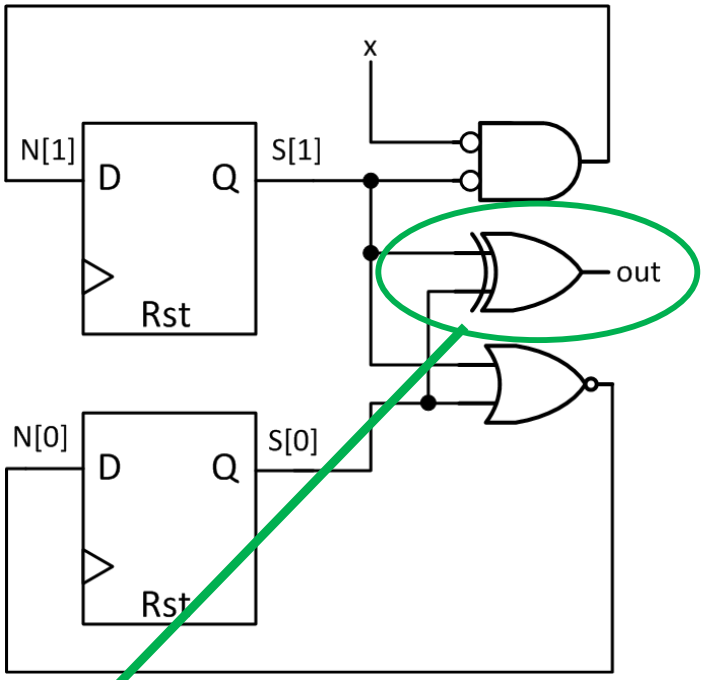
$N[1] = x'.S[1]' = 0$

$N[0] = (S[1]+S[0])' = 0$

That is a next state of 00.
The answer can only be C
Ans: C

Exercise 2 (a better but longer approach)

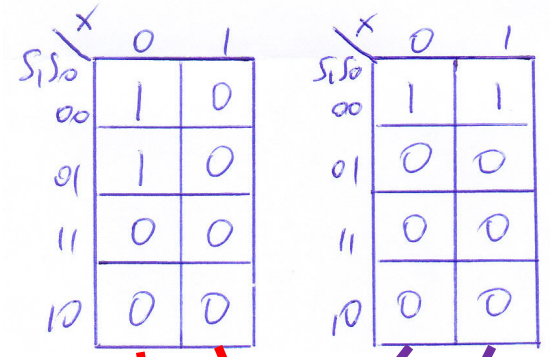
Which FSM corresponds to the given circuit?



1. Get the next state and output expressions

   $N_1 = S_1' x'$
   $N_0 = (S_1 + S_0)' = S_1' S_0'$
   $out = S_1 \wedge S_0$

2. Generate the next state K-maps



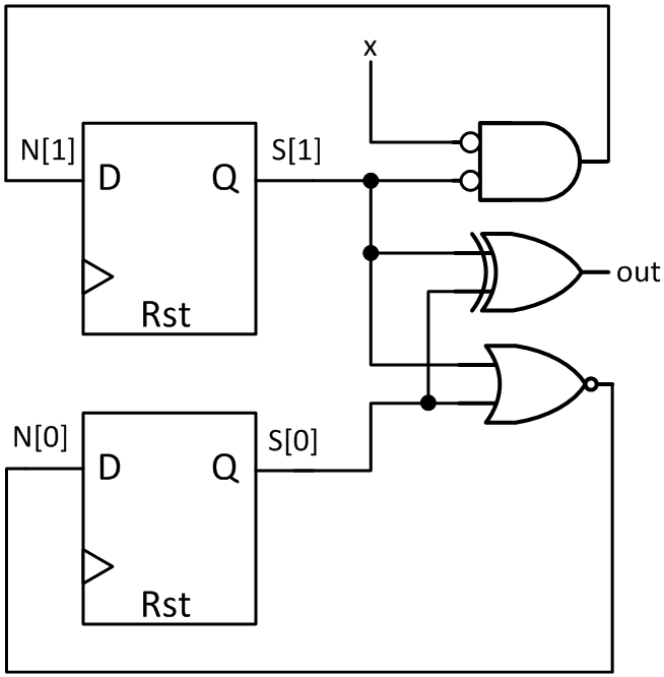
N_1 N_0

3. Then get the state transition table

State S_1S_0	Next state N_1N_0 x = 0	Next state N_1N_0 x = 1	Output (out)
0 0	1 1	0 1	0
0 1	1 0	0 0	1
1 1	0 0	0 0	0
1 0	0 0	0 0	1

Exercise 2 (a better but longer approach)

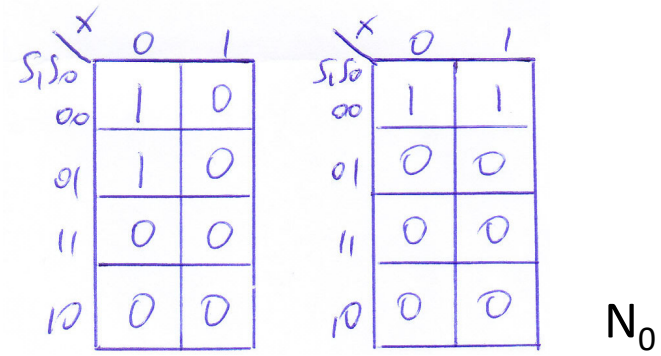
Which FSM corresponds to the given circuit?



1. Get the next state and output expressions

   $N_1 = S_1' x'$
   $N_0 = (S_1 + S_0)' = S_1' S_0'$
   $out = S_1 \wedge S_0$

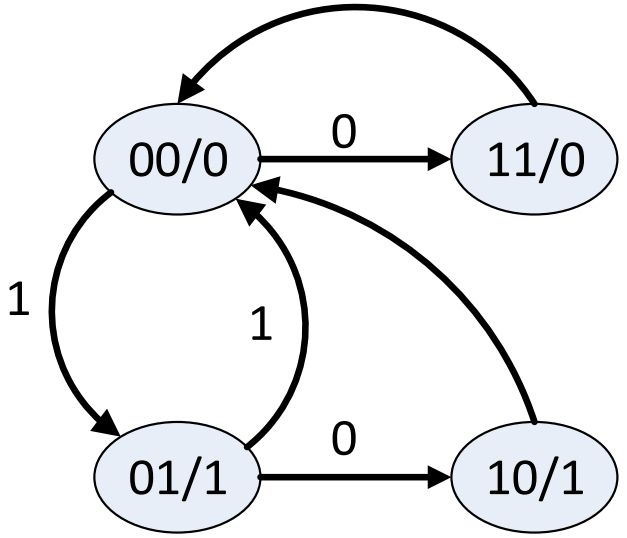
2. Generate the next state K-maps



N_1 N_0

3. Then get the state transition table

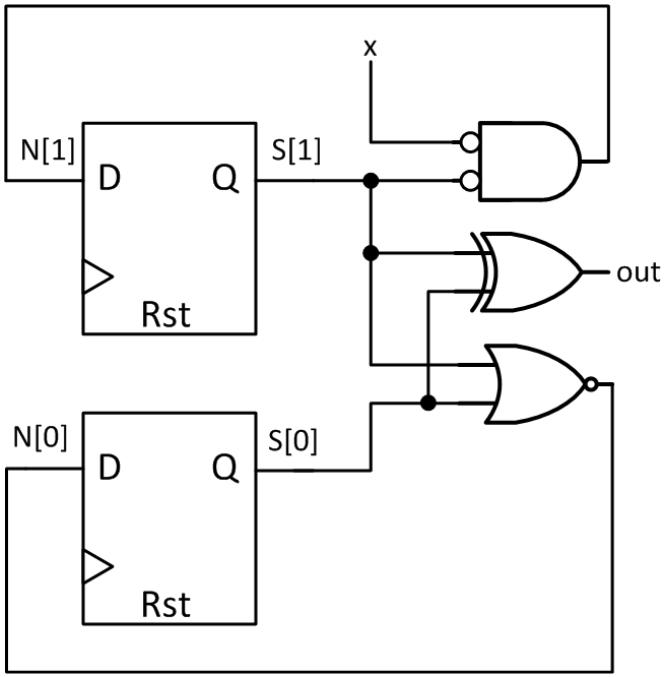
State S_1S_0	Next state N_1N_0 x = 0	1	Output (out)
0 0	1 1	0 1	0
0 1	1 0	0 0	1
1 1	0 0	0 0	0
1 0	0 0	0 0	1



So obviously Ans: C

Exercise 2 (a better but longer approach)

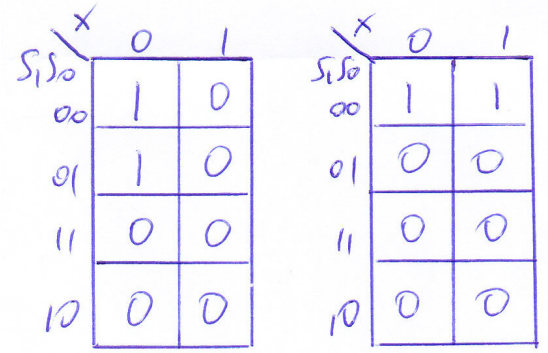
Which FSM corresponds to the given circuit?



1. Get the next state and output expressions

   $N_1 = S_1' x'$
   $N_0 = (S_1 + S_0)' = S_1' S_0'$
   $out = S_1 \wedge S_0$

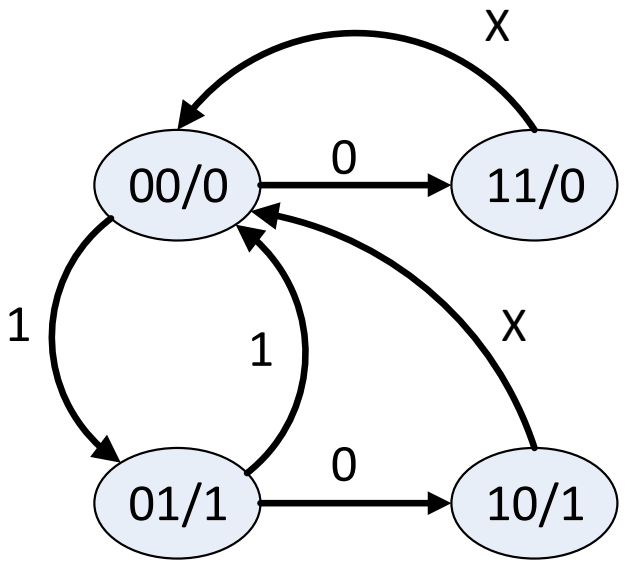
2. Generate the next state K-maps



N_1 N_0

3. Then get the state transition table

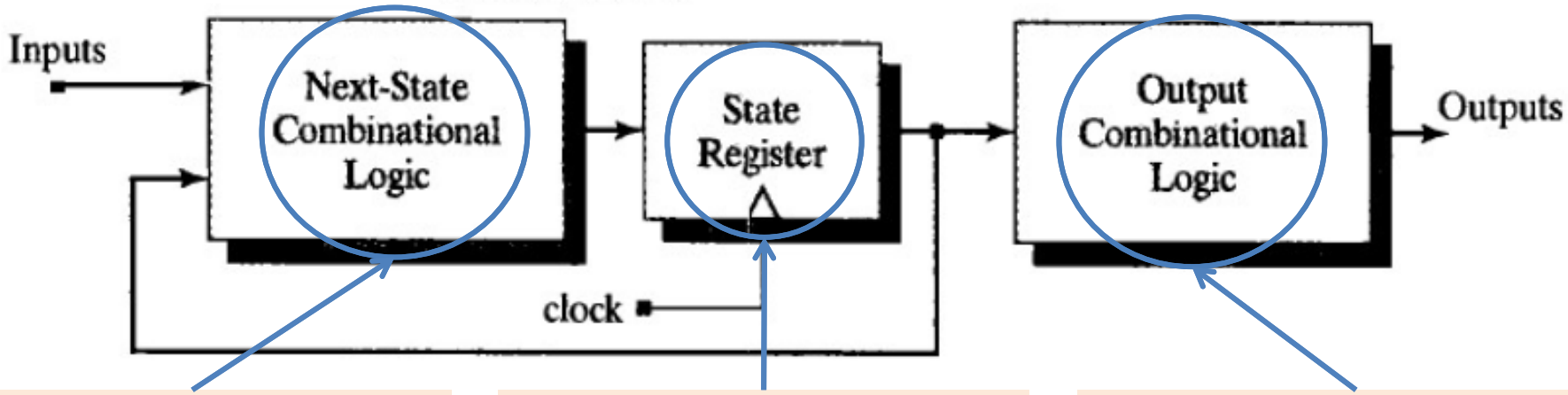
State S_1S_0	Next state N_1N_0 x = 0	1	Output (out)
0 0	1 1	0 1	0
0 1	1 0	0 0	1
1 1	0 0	0 0	0
1 0	0 0	0 0	1



Note the unlabelled transitions are don't cares. Should probably label them as "X".

FSM Structure in Verilog

- Typically Moore FSMs are easier to implement and to analyse
  - Mealy FSMs are typically more efficient (in terms of states)
- We can use this basic structure to implement any FSM



Some logic to determine what the next state is

```
always @ *
begin
    case(st)
    ...
or continuous assign
```

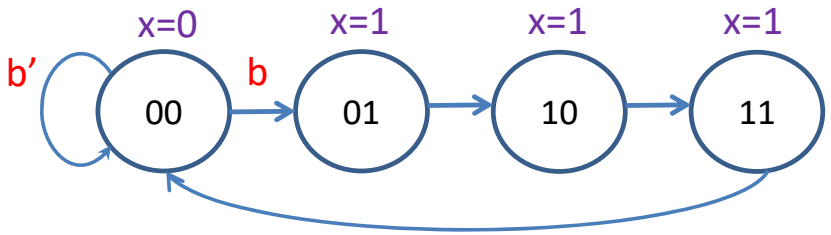
A state register that holds the current state

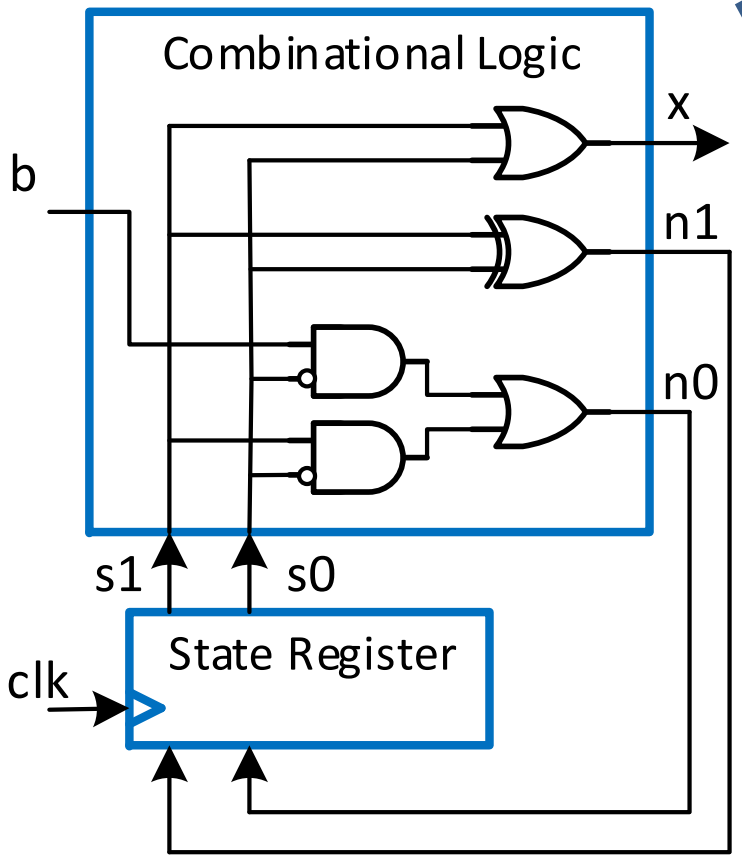
```
always @(posedge clk)
begin
...
```

Some logic to drive the output

```
Continuous assign or
always @ * block
```

The previous Example 1

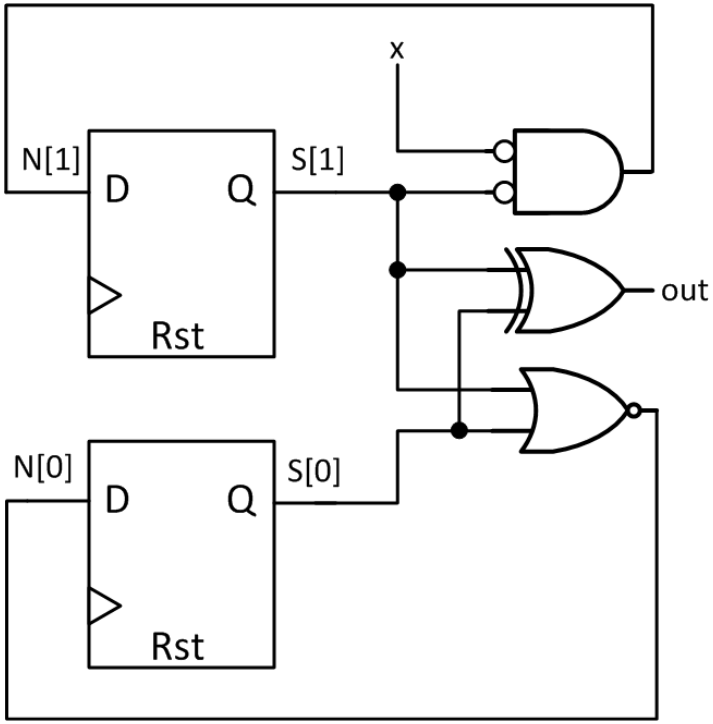





```
module FSM_E1 (input clk, rst, b,
                output x);

  wire n1, n0;   //not really needed
  reg  s1, s0;

  assign n1 = s1^s0;
  assign n0 = (~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 (revisited)

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
```

C

```
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
```

D

```
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
```

S is the state. So N (next state) is assigned to S on the clock edge. (Not N <= S;) Anyway, cannot assign to N using both **assign** and **always**.

N[1] is determined by:

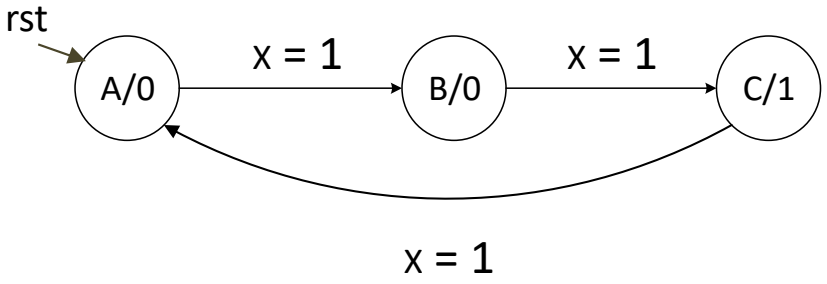
N[1] = ~x & ~S[1];

So the only answer is D

Ans: D

Lecture 22 (Task 3)

Determine the Verilog code for the following FSM



First, is it a Moore or Mealy FSM?

We can implement the FSM with 3 separate logic blocks

1. The synchronous state logic
2. The combinational next state logic
3. The combinational output logic

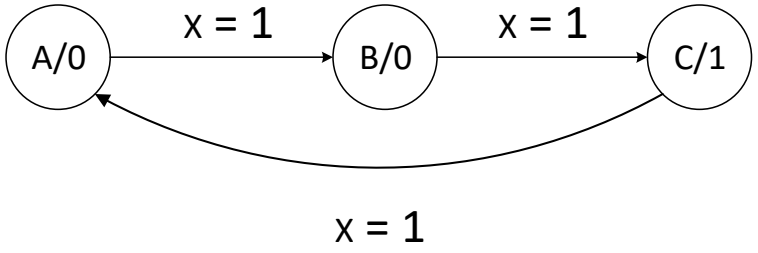
```
module fsm (input x, clk, rst, output reg u);

   parameter A=2'b00, B=2'b01, C=2'b10;

   reg [1:0] nst, st;

   always @ (posedge clk)  // State logic
   begin
      if(rst) st <= A;
      else st <= nst;
   end

   always @ *                    //Next state logic
   begin
      nst = st;
      case(st)
         A: if(x) nst = B;
         B: if(x) nst = C;
         C: if(x) nst = A;
      endcase
   end

   always @ *                    // Output logic
   begin
      u   = 1'b0;
      case(st)
         C: u = 1'b1;
      endcase
   end

endmodule
```

No need for else statements as we have used a default assignment for ***nst***.

The default assignments mean that we do not need the default statement in the case blocks

Lecture 22 (Task 3)

Determine the Verilog code for the following FSM



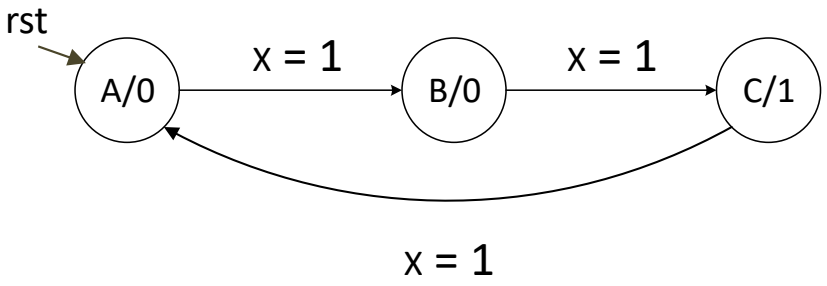
Usually, it is simpler to use just 2 separate logic blocks (even for a Moore FSM)

1. The synchronous state logic
2. The combinational next state and output logic

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

   parameter A=2'b00, B=2'b01, C=2'b10;

   reg [1:0] nst, st;

   always @ (posedge clk)    // State logic
   begin
      if(rst) st <= A;
      else st <= nst;
   end

   always @ *                // NS & out logic
   begin
      nst = st;
      u   = 1'b0;
      case(st)
         A: if(x) nst = B;
         B: if(x) nst = C;
         C: begin
               if(x) nst = A;
               u = 1'b1;
            end
      endcase
   end

endmodule
```

Lecture 22 (Task 3)

Determine the Verilog code for the following FSM



We could also start with the first implementation but use a continuous assignment for the output (u).

u = st[1]

We implement the synchronous state logic and the combinational next state logic using always blocks. Then the output logic using a continuous assignment.

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

   parameter A=2'b00, B=2'b01, C=2'b10;

   reg [1:0] nst, st;

   always @ (posedge clk)  // State logic
   begin
      if(rst) st <= A;
      else st <= nst;
   end

   always @ *              //Next state logic
   begin
      nst = st;
      case(st)
         A: if(x) nst = B;
         B: if(x) nst = C;
         C: if(x) nst = A;
      endcase
   end

   assign u = st[1];

endmodule
```

This is probably the simplest and easiest way!

Selected Past Exam Questions

CE/CZ1005 2018-2019 Sem 1 (Nov/Dec 2018)

Q4(b) A simple vending machine dispenses healthy muesli bars. The vending machine accepts only 50 cent and one dollar coins and a muesli bar costs one dollar. If an excess amount is entered, for example 50 cents followed by one dollar, the transaction is rejected and all coins are returned.

(i) Sketch the Moore-style state transition diagram if the FSM has 2 inputs (*fifty and dollar*), and 3 outputs (*insert_coin*, *dispense* and *reject_money*). Use states *Ready*, *Dispense*, *Reject* and any other states necessary. A reset forces the FSM to the *Ready* state with output {*insert_coin*, *dispense*, *reject_money*} = {1, 0, 0}.

I assume that the FSM will return to the *Ready* state one clock cycle after *Dispense* or *Reject*. There are other assumptions that could be made.

(5 marks)

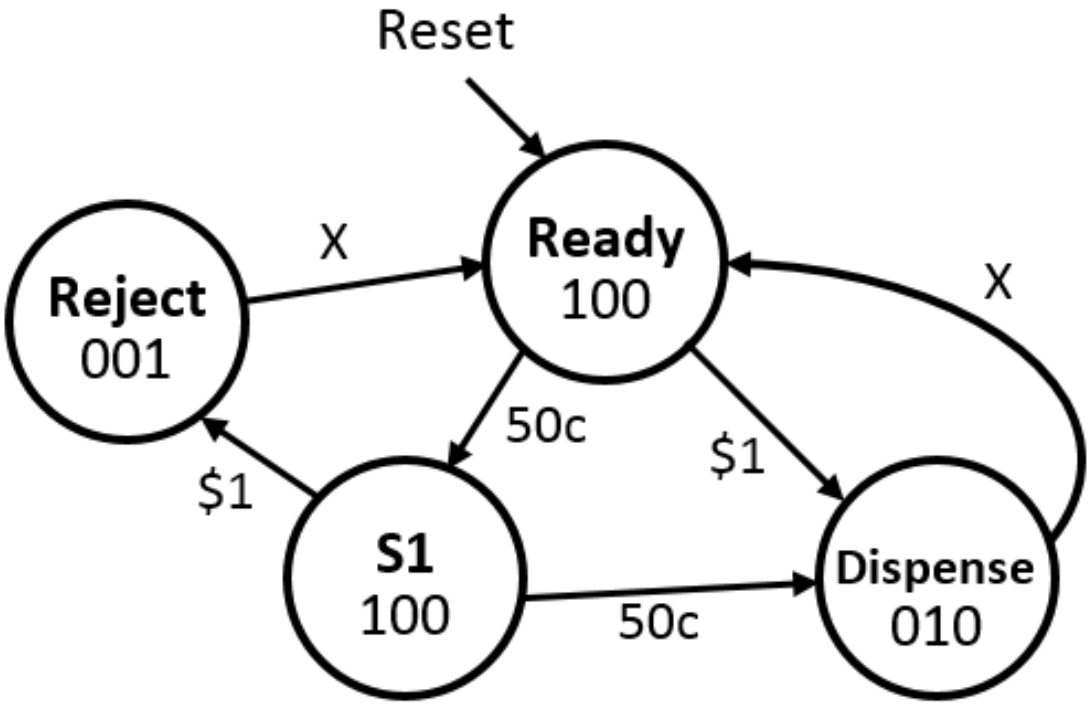
(ii) Write the Verilog module for the FSM using an active low asynchronous reset. Use a separate always block for the next state and output logic.

(12 marks)

Selected Past Exam Questions

Q4(b) A simple vending machine dispenses healthy muesli bars. The vending machine accepts only 50 cent and one dollar coins and a muesli bar costs one dollar. If an excess amount is entered, for example 50 cents followed by one dollar, the transaction is rejected and all coins are returned.

(i) Sketch the Moore-style state transition diagram if the FSM has 2 inputs (*fifty and dollar*), and 3 outputs (*insert_coin*, *dispense* and *reject_money*). Use states *Ready*, *Dispense*, *Reject* and any other states necessary. A reset forces the FSM to the *Ready* state with output {*insert_coin*, *dispense*, *reject_money*} = {1, 0, 0}.



Selected Past Exam Questions

CE/CZ1005 2018-2019 Sem 1 (Nov/Dec 2018)

Q4(b) (ii) Write the Verilog module for the FSM using an active low asynchronous reset. Use a separate always block for the next state and output logic.

```
module vend_fsm (input clk, rst, fifty, dollar, output reg insert, dispense, reject);
        //------------state Constants-------------------------
        parameter READY  = 2'b00, DISPENSE = 2'b01, REJECT = 2'b10, S1 = 2'b11;
        //------------Internal Variables----------------------
        reg [1:0] state, next_state;
        //--------next state and output logic---------------
        always @ * begin
                insert = 1;
                dispense = 0;
                reject = 0;
                next_state = state;
                case (state)
                        READY : begin
                                if (fifty) next_state = S1;
                                if (dollar) next_state = DISPENSE;
                                // outputs set by default above
                        end
```

Default Assignments

NOTE: The coin mechanism of a typical vending machine only allows coins to be put in one at a time. HENCE it is NOT possible to have the fifty and dollar inputs asserted at the same time. Thus you do NOT need to use **if** … **else** … statements

Selected Past Exam Questions

CE/CZ1005 2018-2019 Sem 1 (Nov/Dec 2018) Q4(b) (ii) continued

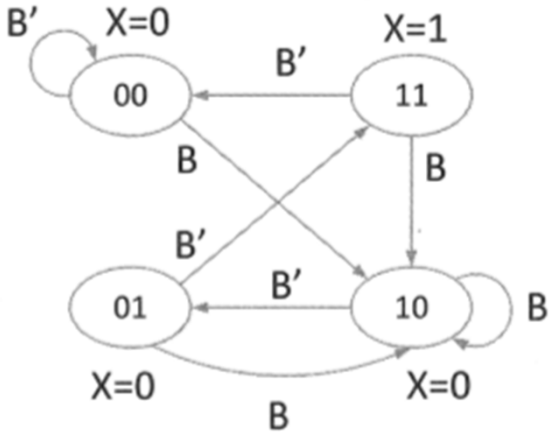
```
                DISPENSE : begin
                        next_state = READY;
                        insert = 0;
                        dispense = 1;
                end
                REJECT : begin
                        next_state = READY;
                        insert = 0;
                        reject = 1;
                end
                S1 : begin
                        if (fifty) next_state = DISPENSE;
                        if (dollar) next_state = REJECT;
                        // outputs set by default above
                end
        endcase         //default: no need as covered by default assignments above
    end                 // end of always block: next state and output logic

    //----------Seq Logic-----------------------------
    always @ (posedge clk, negedge rst)
    begin
        if (~rst) state <= READY;
        else state <= next_state;
    end
endmodule           // End of Module vending_fsm
```

Selected Past Exam Questions

CE/CZ1005 2018-2019 Sem 2 (April/May 2019)

Q4(b) (i) The figure shows a state transition diagram with 4 states.

Draw the state transition table



Current State	Next State $B=$ 0	1	Output X
A (00)	**A**	C	0
B (01)	D	C	0
C (10)	B	**C**	0
D (11)	A	C	1

Could also draw the excitation table as

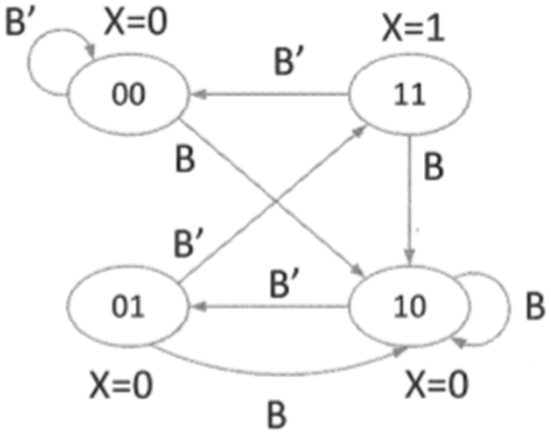
Current State	Next State $B=$ 0	1	Output X
A (00)	**00**	10	0
B (01)	11	10	0
C (10)	01	**10**	0
D (11)	00	10	1

Selected Past Exam Questions

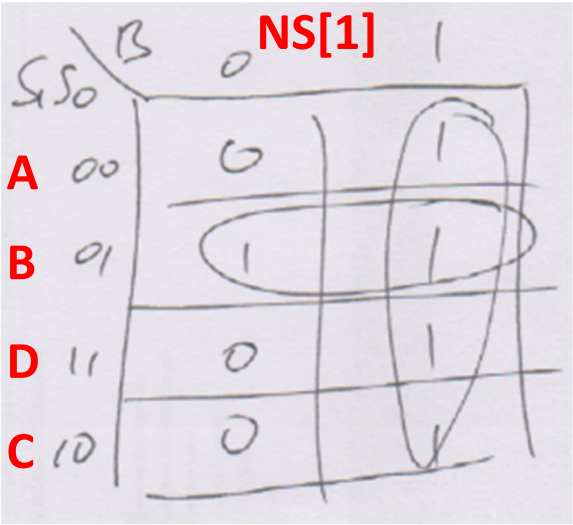
CE/CZ1005 2018-2019 Sem 2 (April/May 2019)

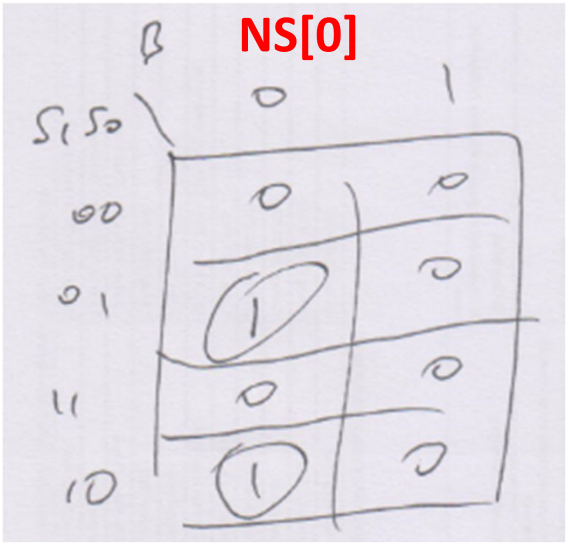
(ii) Find the minimum SOP expressions for the next states and X. (current state is S[1] S[0])

Current State	Next State $B=0$	Next State $B=1$	Output X
A (00)	**00**	10	0
B (01)	11	10	0
C (10)	01	**10**	0
D (11)	00	10	1



Input: B
Output: X





NS[1] = B + S[1]'.S[0]

NS[0] = S[1]'.S[0].B' + S[1].S[0]'.B'

X = S[1].S[0]

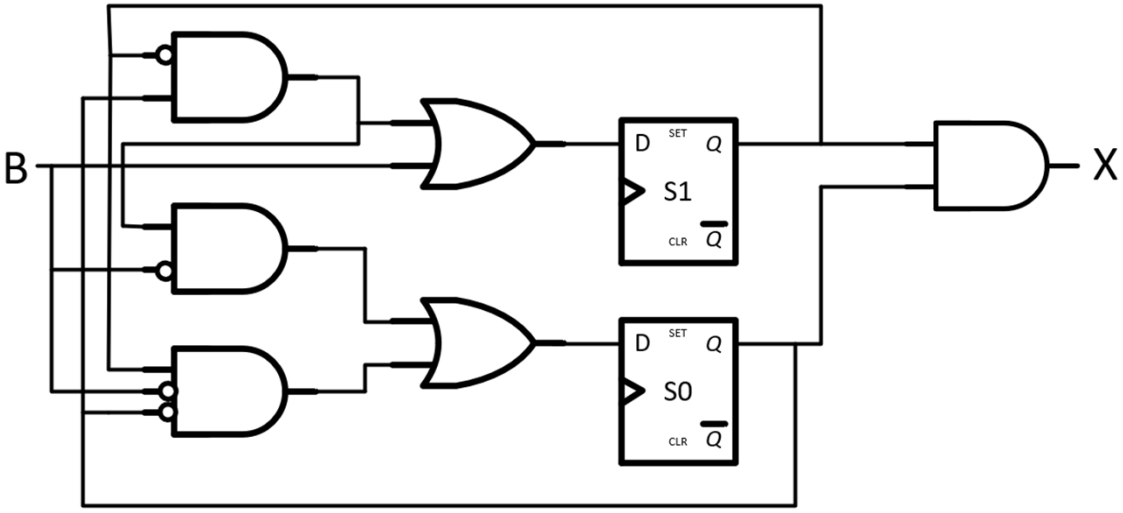
Selected Past Exam Questions

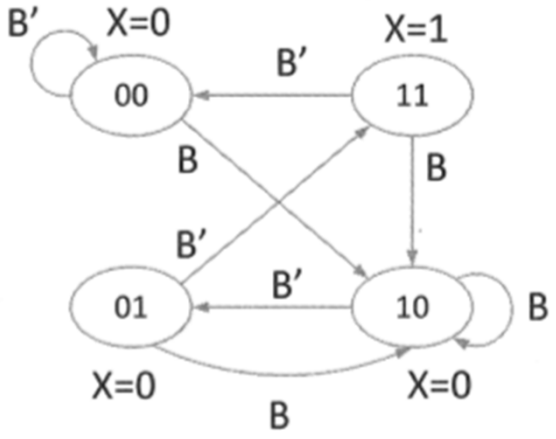
CE/CZ1005 2018-2019 Sem 2 (April/May 2019)

(iii) Based on the minimum SOP expressions, draw the circuit for the FSM.

Use D-FFs and gates.

NS[1] = B + S[1]'.S[0] NS[0] = S[1]'.S[0].B' + S[1].S[0]'.B' X = S[1].S[0]





Input: B
Output: X

Selected Past Exam Questions

NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2017-2018 Sem 2

I assume they mean **"increments to" the next value**, rather than **counts the next value.**

Q4(b) A 3-bit odd/even up-counter has a 1-bit input *In* and a 3-bit output *Count*. When *In* is 0 it counts the next even value, and when *In* is 1 it counts the next odd value. The counter goes back to zero (initial state) only when it reaches the maximum odd count. It stays at the maximum even count state when *In* is 0.

(i) Draw the state transition diagram for the 3-bit odd/even up-counter.

(7 marks)

(ii) Determine the output *Count* for the following input sequence: 1,1,0,1,0,1,0.

I assume the counter starts from zero.

(3 marks)

(iii) Write the behavioral Verilog module for the odd/even up-counter discussed above using a CASE statement. Assume two additional inputs *CLK* and *RESET* for the sequential circuit implementation.

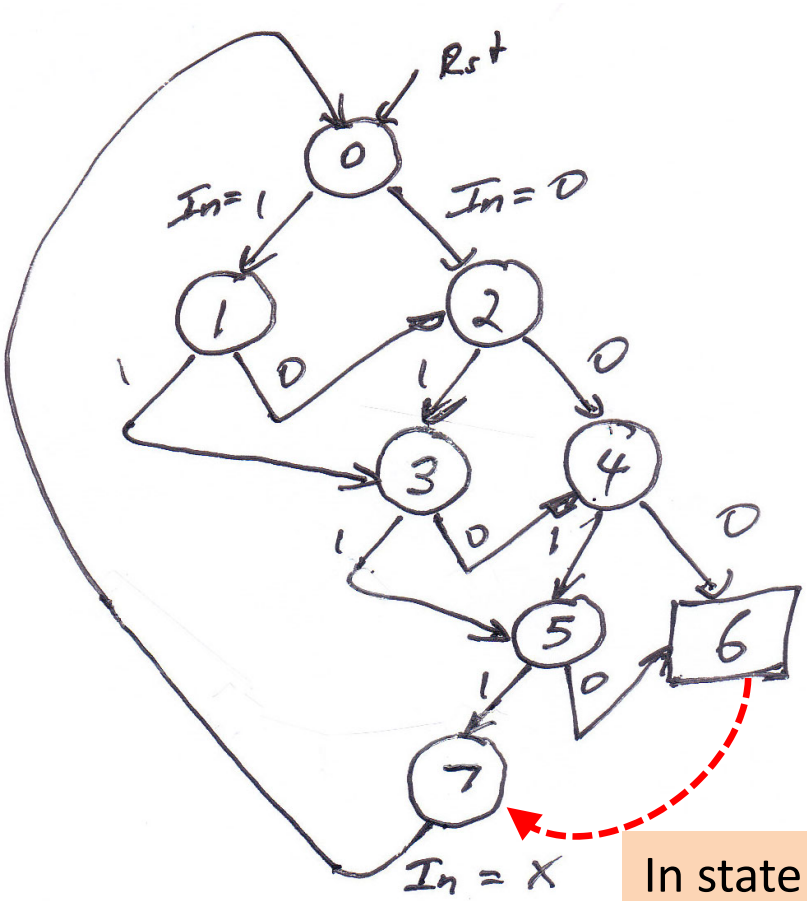
(10 marks)

Selected Past Exam Questions

CE/CZ1005 2017-2018 Semester 2 (Apr/May 2018)

Q4(b) (i) If *In*=0, it should count 0, 2, 4, 6 and stop.

If *In*=1, it should count 0, 1, 3, 5, 7, 0, 1, …



(ii) Determine *Count* for the input seq.:

1, 1, 0, 1, 0, 1, 0

assuming we start at the reset state

Ans: Just follow the states in the state diagram

starting from S0, In=1 takes us to S1

then we go S3, S4, S5, S6, then S7 and lastly back to S0

so *Count* is

0, 1, 3, 4, 5, 6, 7, 0

In state 6, IF *In*=1 we SHOULD GO TO 7

Selected Past Exam Questions

CE/CZ1005 2017-2018 Sem 2 (Apr/May 2018)

Q4(b) (iii) Write the behavioral Verilog module for the odd/even up-counter discussed above using a CASE statement. Assume two additional inputs *CLK* and *RESET* for the sequential circuit implementation.

As it says to use a case statement, I would just treat it as a FSM problem.

```
module up_counter (input CLK,RESET,In,
                   output [2:0] Count);
   parameter S0=0,S1=1,S2=2,S3=3;
   parameter S4=4,S5=5,S6=6,S7=7;
   reg [2:0] nst, st;
   always @ (posedge CLK)
   begin
      if(RESET) st <= S0;
      else st <= nst;
   end
   always @ *          // NS & out logic
   begin
      nst = st;
      case(st)
         S0: if(In) nst = S1; else  nst = S2;
         S1: if(In) nst = S3; else  nst = S2;
         S2: if(In) nst = S3; else  nst = S4;
         S3: if(In) nst = S5; else  nst = S4;
         S4: if(In) nst = S5; else  nst = S6;
         S5: if(In) nst = S7; else  nst = S6;
         S6: if(In) nst = S7;  //S6 is default
         S7: nst = S0;   //always
      endcase
   end
   assign Count = st;
endmodule
```

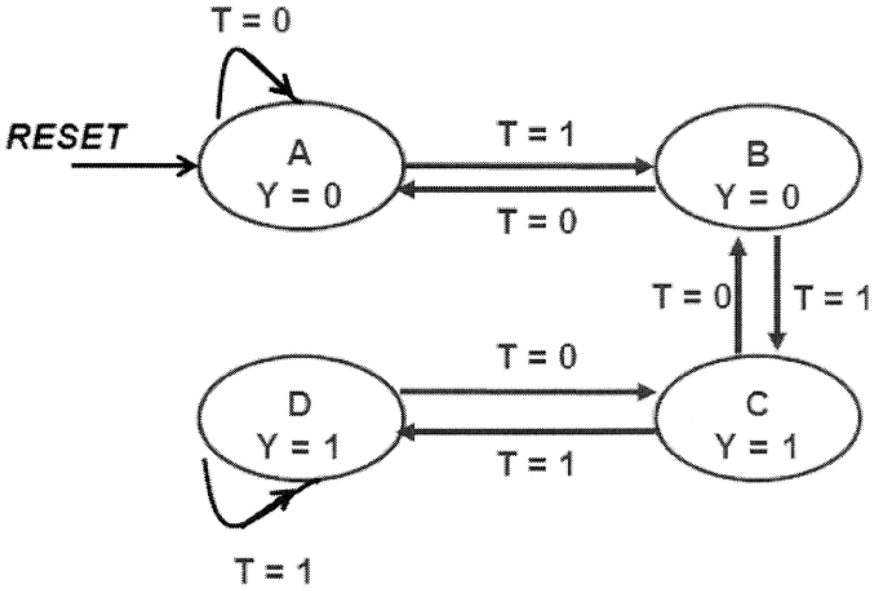
NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2017-2018 Semester 1 (Nov/Dec 2017)

Q4(c) The state transition diagram of a Finite State Machine (FSM) is shown in Figure Q4b. There are three inputs RESET, CLK and T, and one output Y.

(i) Determine if it is a Mealy or Moore FSM. Justify your answer.

(2 marks)



ANS: As the output Y is only determined by the current state, it is a **Moore** FSM.

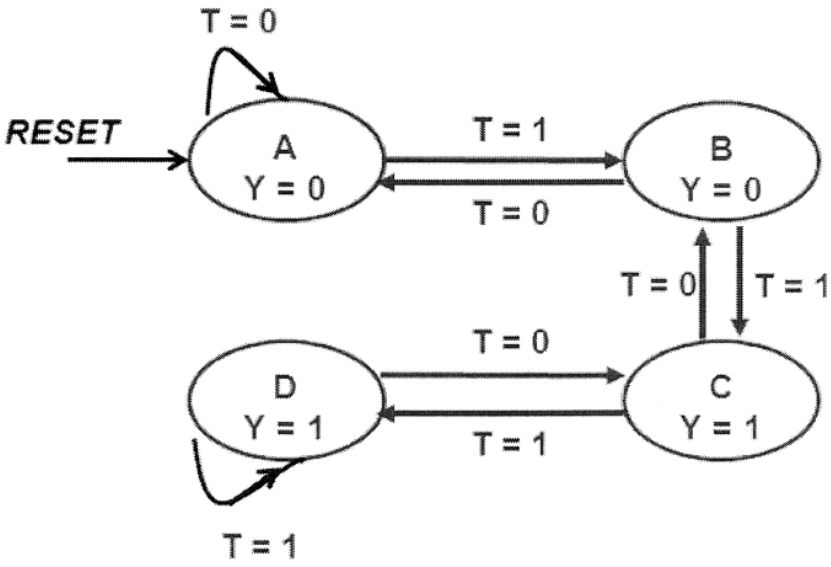
Selected Past Exam Questions

CE/CZ1005 2017-2018 Semester 1 (Nov/Dec 2017)

Q4(c) (ii) Write the Verilog module for the FSM shown in Figure Q4b, using an asynchronous RESET.

(10 marks)

```
module fsmQ4(input CLK,RESET,T, output reg Y);
   parameter A=2'b00,B=2'b01,C=2'b10,D=2'b11;
   reg [1:0] nst, st;
   always @ (posedge CLK or posedge RESET)
   begin
      if(RESET) st <= A;
      else st <= nst;
   end
   always @ *              // NS & out logic
   begin
      nst = st;  Y   = 1'b0;
      case(st)
         A: if(T) nst = B;
         B: if(T) nst = C; else  nst = A;
         C: begin if(T) nst = D; else  nst = B;
               Y = 1'b1;
            end
         D: begin if(~T) nst = C;
               Y = 1'b1;
            end
      endcase
   end
endmodule
```



NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2016-2017 Semester 2 (Apr/May 2017)

Q4(a) An autonomous vehicle developed in NTU moves students between five designated locations across the campus. The movement of the vehicle is controlled by a 1-bit input *MOVE* which is sent wirelessly from the control room. The 3-bit output *LOC* of the FSM shows the current location of the autonomous vehicle. Table Q4 shows the rules to perform a test run of this vehicle which will start from location SCSE.

Table Q4

Current Location	Next Location	
	MOVE = 0	***MOVE = 1***
A SCSE	B Student Gym	D Innovation Center
B Student Gym	A SCSE	C Auditorium
C Auditorium	C Auditorium	E Library
D Innovation Center	A SCSE	E Library
E Library	D Innovation Center	C Auditorium

I am very lazy and do not like typing lots of text, so I will rename the states as A, B, C, D, & E.

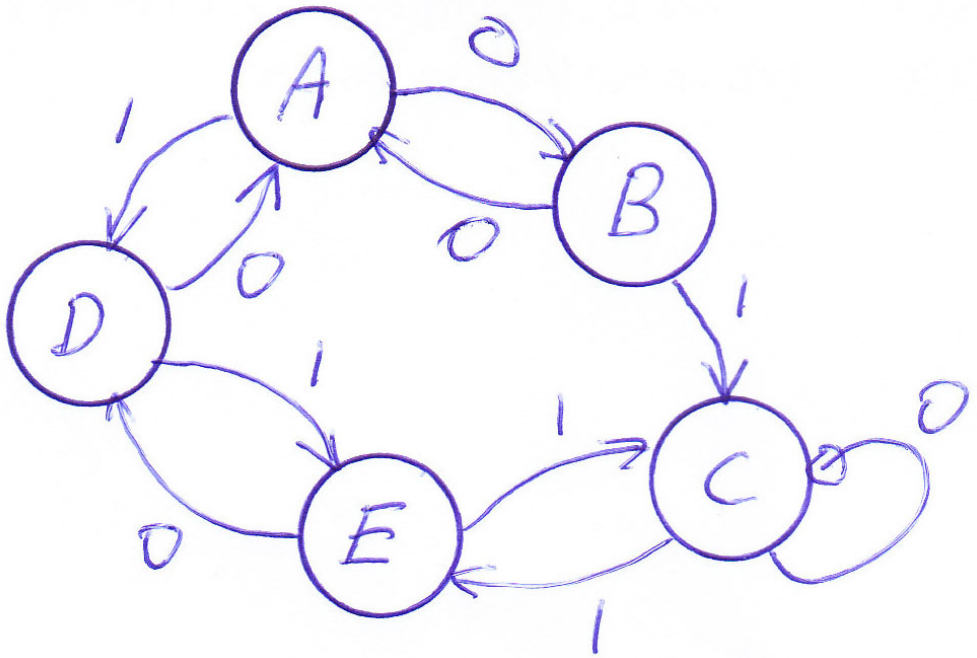
NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2016-2017 Semester 2 (Apr/May 2017)

Q4(a) (i) Draw a Finite State Machine (FSM) for the autonomous vehicle.

Current State	Next State M = 0	1	Output
A	B	D	A
B	A	C	B
C	C	E	C
D	A	E	D
E	D	C	E

(6 marks)



(ii) What should be the input sequence such that the vehicle visits each location once and returns to SCSE? (State A)

(2 marks)

Ans: It must move A, B, C, E, D, A. There is no other solution.
Thus, the input sequence is: **0 1 1 0 0**.

Selected Past Exam Questions

CE/CZ1005 2016-2017 Semester 2 (Apr/May 2017)

Q4(a) (iii) Write the Verilog module for the *FSM* designed in Q4(a)(i) with inputs *MOVE, RESET, CLK* and output *LOC*. Assume the vehicle returns to SCSE when *RESET* is high.

(10 marks)

```
module vfsm (input CLK, RST, M, output [2:0] LOC);
   parameter A=0, B=1, C=2, D=3, E=4;
   reg [2:0] nst, st;
   always @ (posedge CLK)    // State logic
   begin
      if(RST) st <= A;
      else st <= nst;
   end
   always @ *                // NS logic
   begin
      nst = st;
      case(st)
         A: if(M) nst = D; else nst = B;
         B: if(M) nst = C; else nst = A;
         C: if(M) nst = E; else nst = C;
         D: if(M) nst = E; else nst = A;
         E: if(M) nst = C; else nst = D;
      endcase
   end
   assign LOC = st;          // Output Logic
endmodule
```

Current State	Next State M = 0	1	Output
A	B	D	A
B	A	C	B
C	C	E	C
D	A	E	D
E	D	C	E

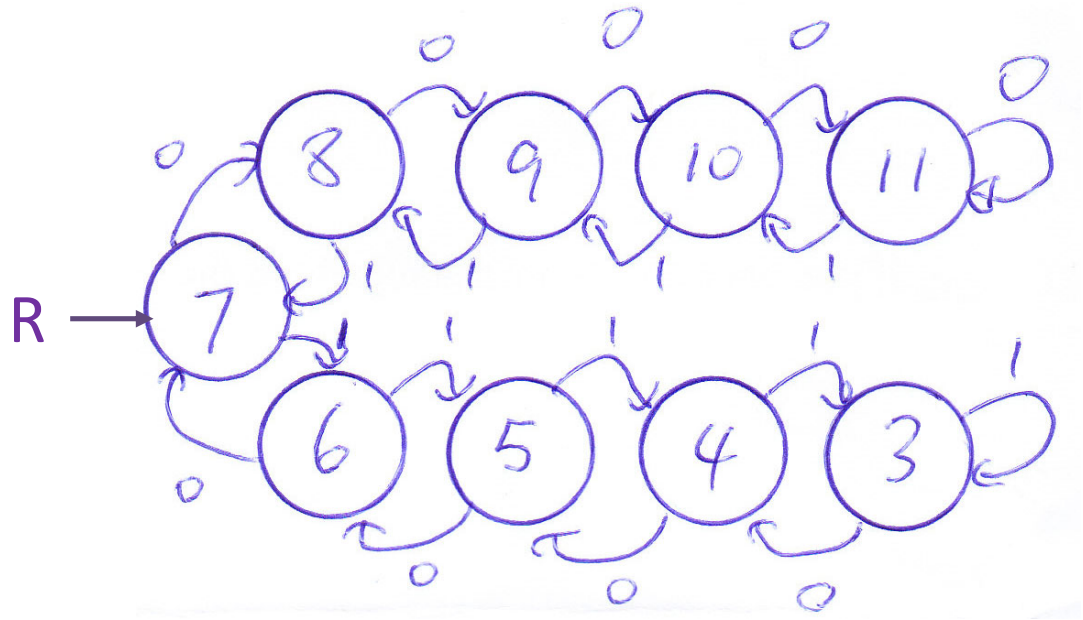
NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2016-2017 Semester 1 (Nov/Dec 2016)

Q4(b) You are required to design a special up-down counter. The counter never counts below 3 and above 11. The counter has a reset input (R) that sets it to 7 (that is, when R is 1), and a direction input (D) that determines whether it counts up or down (up if D is 0 and down if D is 1). If the counter is reset to 7 and D is 1, its successive states will then be 6, 5, 4, 3 and it will stay at 3 until R is 1 or D is 0. Similarly, if after reset D is 0, its successive states will then be 8, 9, 10, 11 and it will stay at 11 until R is 1 or D is 1.

(i) Draw the state transition diagram for the special up-down counter.

(7 marks)



NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ1005 2016-2017 Semester 1 (Nov/Dec 2016)

Q4(b) (ii) Write a Verilog module to implement the special up-down counter.

(8 marks)

```
module spec_cnt (input clk, R, D,
                 output reg [3:0] cout);

    always @(posedge clk)
    begin
        if (R) cout <= 7;
        else begin
            if (D && (cout != 3))
                cout <= cout -1;
            else if (!D && (cout != 11))
                cout <= cout +1;
        end
    end
endmodule
```

While it would be possible to write this as a normal state machine using a case statement to move between states, it is probably much easier to just use a counter.