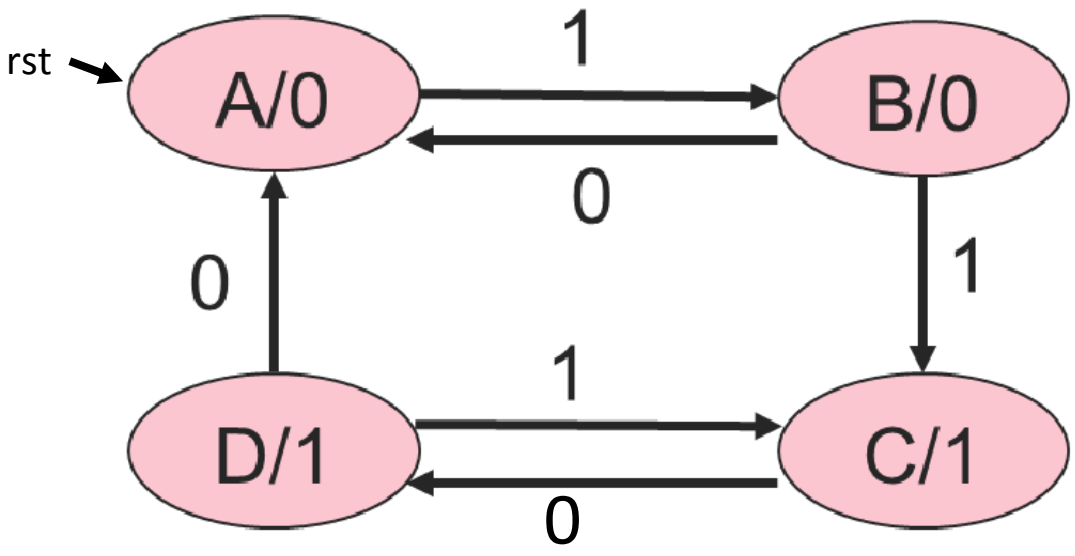
CE/CZ1005 Digital Logic
Tutorial 10

Q1. Design a finite state machine that has a single input and single output. It outputs a 1 from the second consecutive high input, and only then outputs a zero after the second consecutive low input. Hence, two consecutive 1 inputs, get a high output, that stays until two consecutive low inputs are received.

(a) Implement the finite state machine in Verilog using a combinational always block for the state transition logic.

(b) Redo the implementation using only assign statements for the state transition logic.

(c) Show how the state machine would respond to the following sequence of inputs:
0,1,0,1,1,0,1,1,0,0,0,1,0,1,1



State Transition Diagram

State (S_1S_0)	Next state (n_1n_0) a = 0	1	Output (x)
A (00)	A (00)	B (01)	0
B (01)	A (00)	C (10)	0
C (10)	D (11)	C (10)	1
D (11)	A (00)	C (10)	1

State Transition Table

We can modify the state transition table so that the states (and possibly the inputs) are in grey code format. That makes it easier in part (b).

State (S_1S_0)	Next state (n_1n_0) a = 0	1	Output (x)
A (00)	A (00)	B (01)	0
B (01)	A (00)	C (10)	0
D (11)	A (00)	C (10)	1
C (10)	D (11)	C (10)	1

Modified State Transition Table

(States C and D are swapped to make K-map easy)

Q1 (a) This now gives us enough information to implement the FSM using a combinational always block for the state transition logic. Assume sta is reset state.

```
module fsma (input a, clk, rst,
             output reg x);

parameter sta=2'b00, stb=2'b01, stc=2'b10, std=2'b11;

reg [1:0] n, s;

always @ *
begin
    n = s; x = 1'b0;
    case(s)
        sta: if (a) n = stb;
        stb: if (a) n = stc; else n = sta;
        stc: begin
               if (!a) n = std;
               x = 1'b1;
             end
        std: begin
               if (!a) n = sta;
               else n = stc;
               x = 1'b1;
             end
    endcase
end

always @ (posedge clk)
begin
    if(rst)
        s <= 2'b0;
    else
        s <= n;
end
endmodule
```

Use default assignments at the top of the combinational always block so that we only have to define the other conditions.

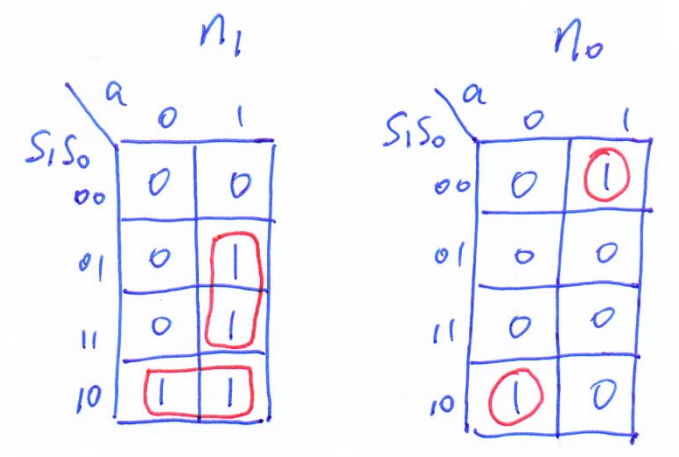
Q1 (b) To implement the FSM using assign statements, we need to determine the logic for the assign statements.

Again assume sta (A) is the reset state.

State (S_1S_0)	Next state (n_1n_0) a = 0	1	Output (x)
A (00)	A (00)	B (01)	0
B (01)	A (00)	C (10)	0
D (11)	A (00)	C (10)	1
C (10)	D (11)	C (10)	1

Modified State Transition Table

(States C and D are swapped to make K-map easy)



Note: The K-maps come directly from the modified state transition table.

n[1]= s[0]a + s[1]s[0]'

n[0]= s[1]'s[0]'a + s[1]s[0]'a'

x = s[1]

```
module fsma (input a, clk, rst,
             output x);
wire [1:0] n;
reg  [1:0] s;
assign n[1] = (s[0]&a) | (s[1]&~s[0]);
assign n[0] = (~s[1]&~s[0]&a) | (s[1]&~s[0]&~a);
assign x = s[1];
always @ (posedge clk)
begin
    if(rst) begin
        s <= 2'b0;
    end else begin
        s <= n;
    end
end
endmodule
```

Q1 (c) Determine the state machine output for the input sequence:

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

We assume we start in the idle/initial state (state A). We then consider what state transitions we will get based on the given input sequence. Once we have the state sequence, we can determine the outputs.