Lab. 08

Logic Design Lab.

Fall 2019

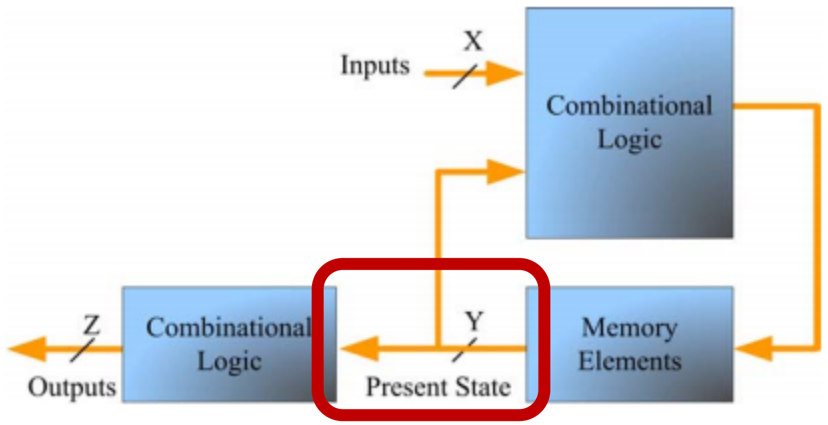
Prof. Sungjoo Yoo
(yeonbin@snu.ac.kr)
TA. Hyunsu Kim
(gustnxodjs@gmail.com)
TA. Hyunyoung Jung
(gusdud1500@gmail.com)

Overview

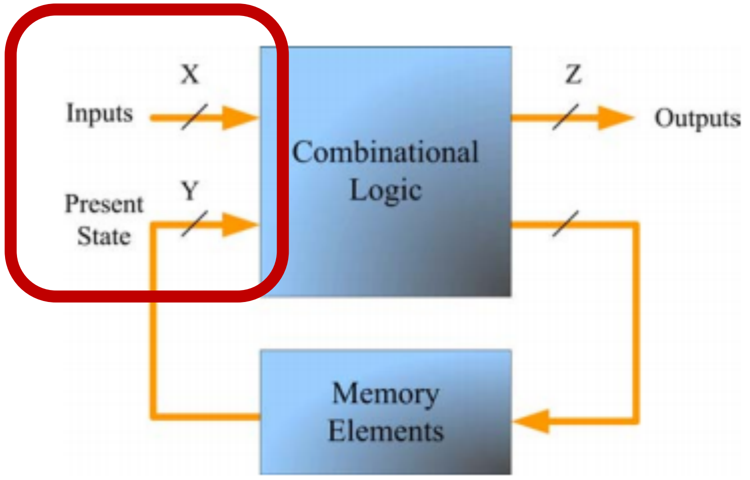
- FSM: Moore & Mealy Machine
- Verilog FSM Implementation Example
- Lab
- Homework

FSM: Moore & Mealy Machine

- Definition of Moore & Mealy Machine



Moore machine



Mealy machine

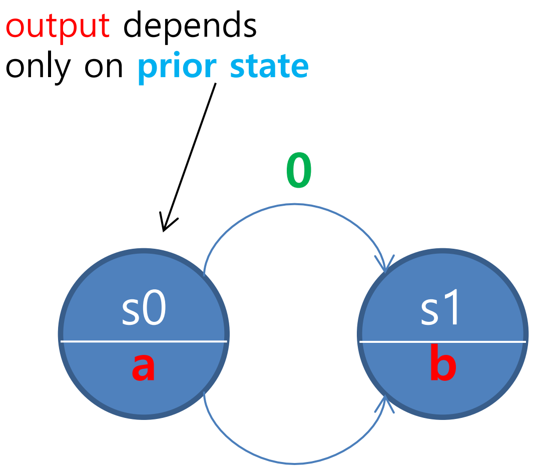
- Outputs are computed via a function of **current state** only
- Output delayed by one clock period

- Outputs are computed via a function of **current state** and **input values**

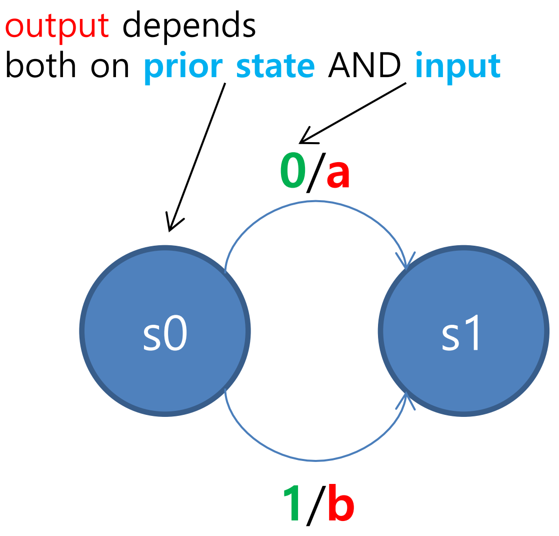
FSM: Moore & Mealy Machine

- State Transition Diagram

0, 1: input
a, b: output
s0, s1: state



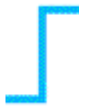
Moore machine

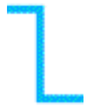


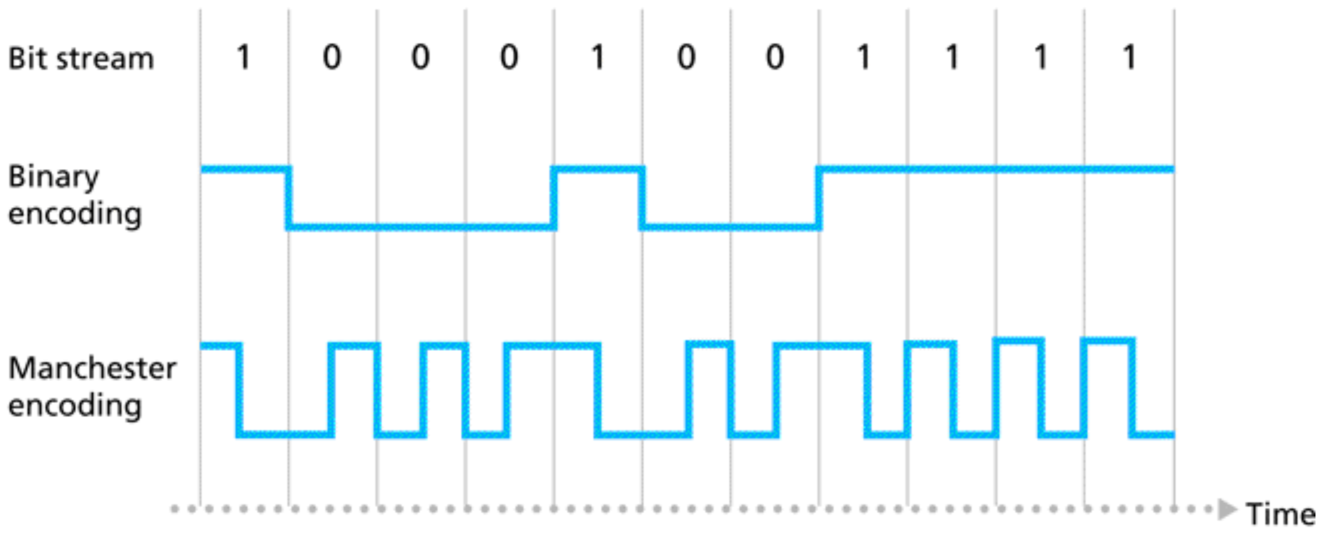
Mealy Machine

Verilog FSM Implementation Example

- Manchester Encoding

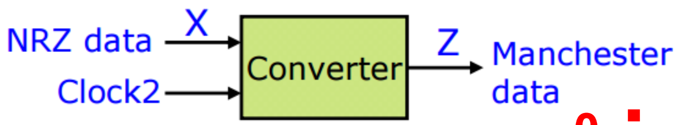
0:


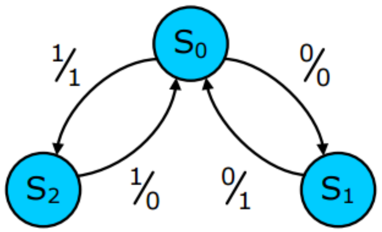
1:


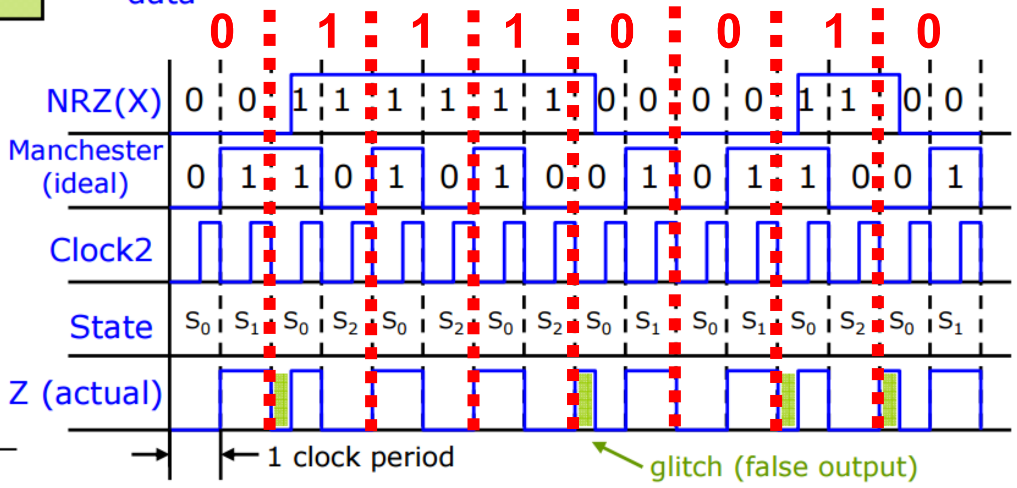


Serial Data Code Conversion NRZ-Code to Manchester-Code

- Mealy machine implementation
  - Use Clock2, twice the frequency of the basic clock
    - If the NRZ bit is 0 (1), it will be 0 (1) for two Clock2 periods





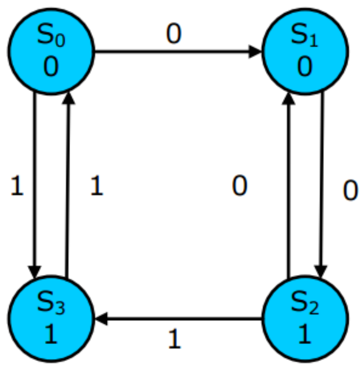


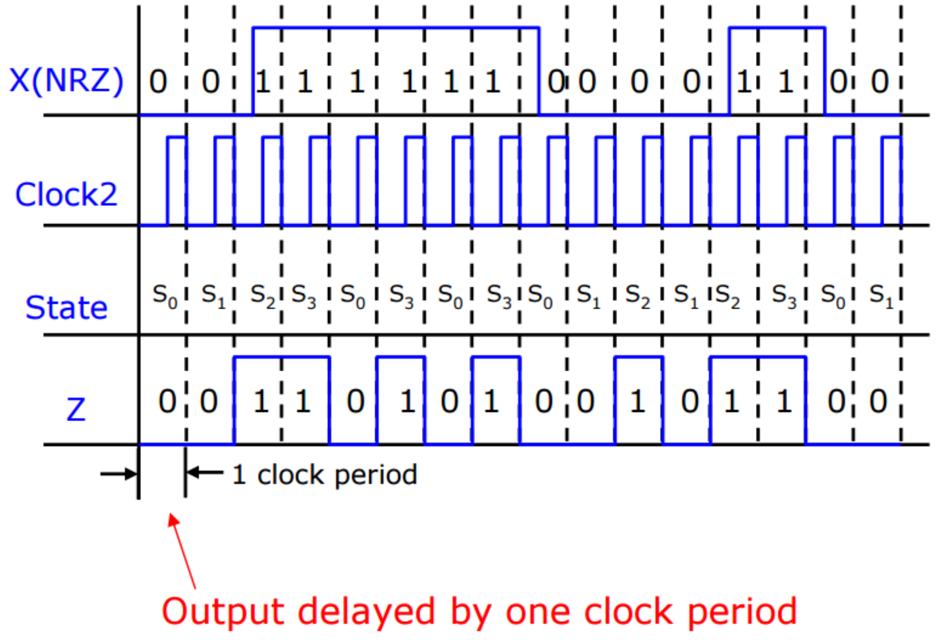
Present State	Next State X=0	Next State X=1	Output Z X=0	Output Z X=1
S_0	S_1	S_2	0	1
S_1	S_0	-	1	-
S_2	-	S_0	-	0

Serial Data Code Conversion NRZ-Code to Manchester-Code

- Moore machine implementation

glitch있어도 상관없다. ->
state 절대적/ input -output
mealy 는 glitch 영향



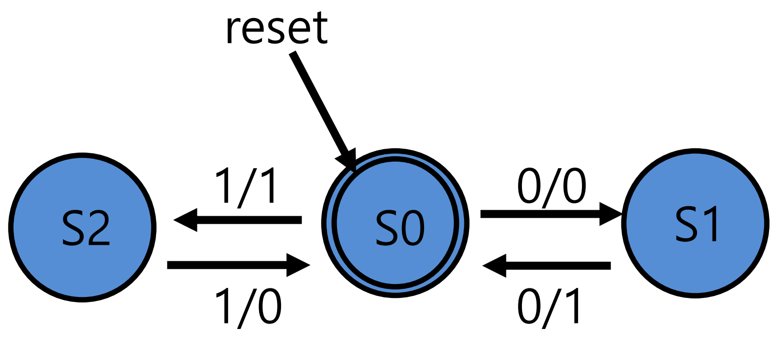


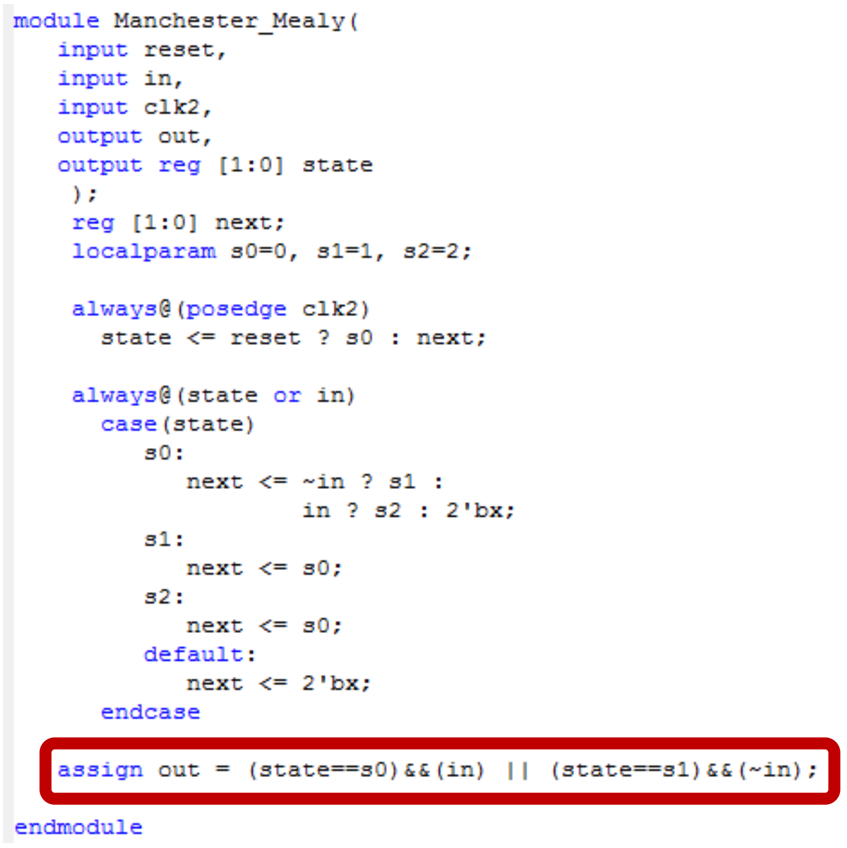
X(NRZ)
0 0 1 1 1 1 1 1 0 0 0 0 1 1 0 0
Clock2
State
S_0 S_1 S_2 S_3 S_0 S_3 S_0 S_3 S_0 S_1 S_2 S_1 S_2 S_3 S_0 S_1
Z
0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0
1 clock period
Output delayed by one clock period

Present State	Next State X=0	Next State X=1	Present Output Z
S_0	S_1	S_3	0
S_1	S_2	-	0
S_2	S_1	S_3	1
S_3	-	S_0	1

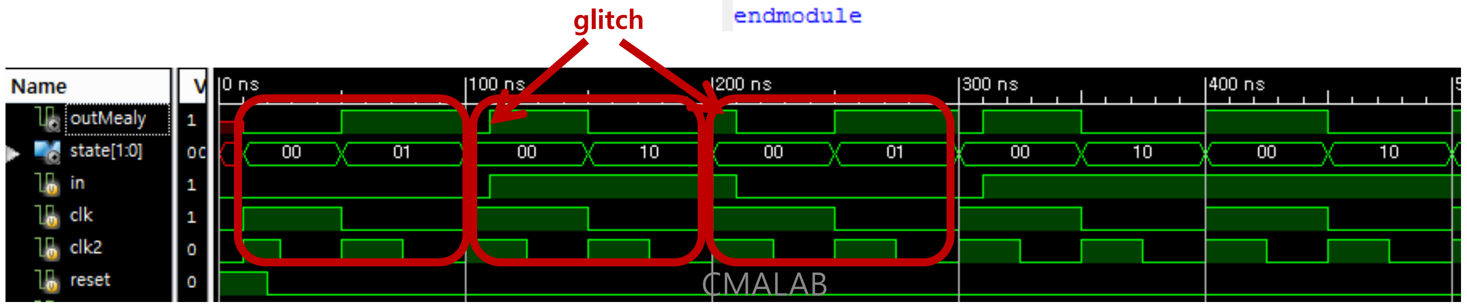
Verilog FSM Implementation Example

- Mealy machine





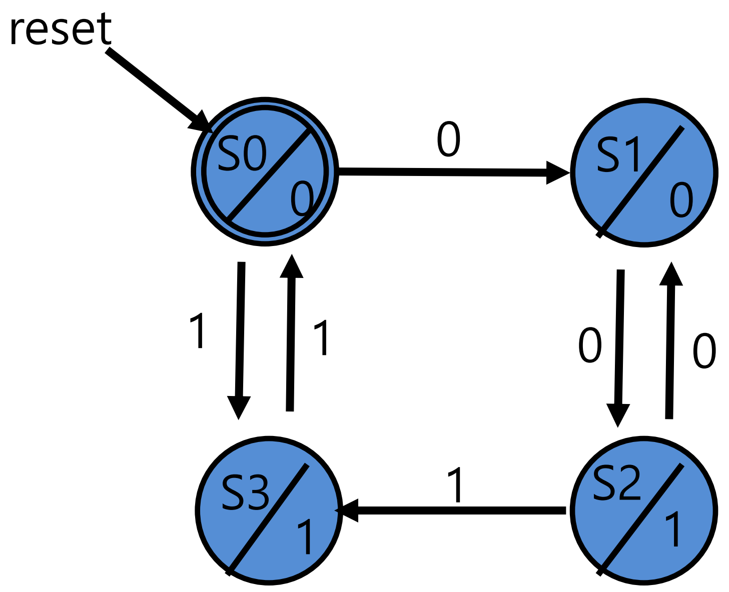
```
module Manchester_Mealy(
   input reset,
   input in,
   input clk2,
   output out,
   output reg [1:0] state
    );
    reg [1:0] next;
    localparam s0=0, s1=1, s2=2;

    always@(posedge clk2)
      state <= reset ? s0 : next;

    always@(state or in)
      case(state)
         s0:
            next <= ~in ? s1 :
                    in ? s2 : 2'bx;
         s1:
            next <= s0;
         s2:
            next <= s0;
         default:
            next <= 2'bx;
      endcase

   assign out = (state==s0)&&(in) || (state==s1)&&(~in);

endmodule
```

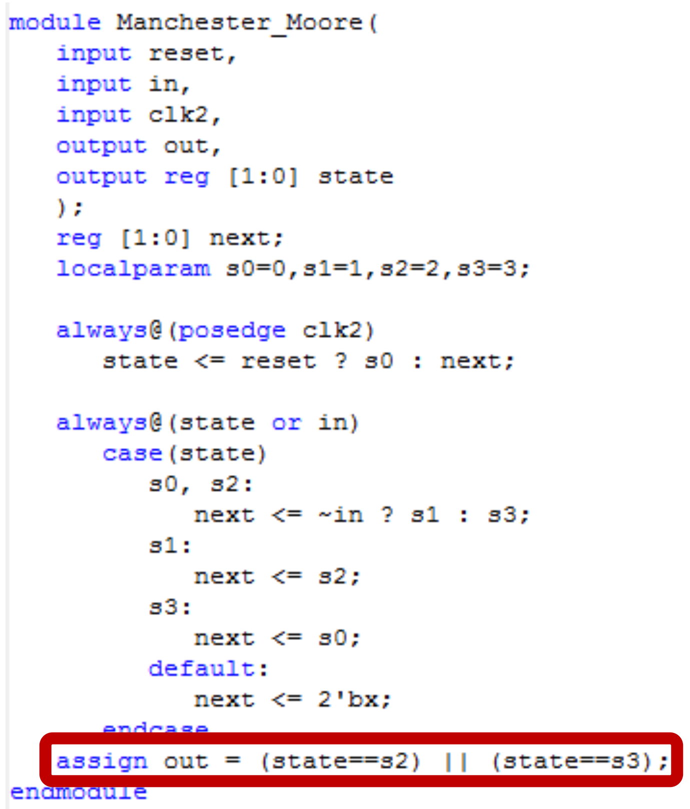


- **Output change is NOT synchronous with state change; output can change during one state**

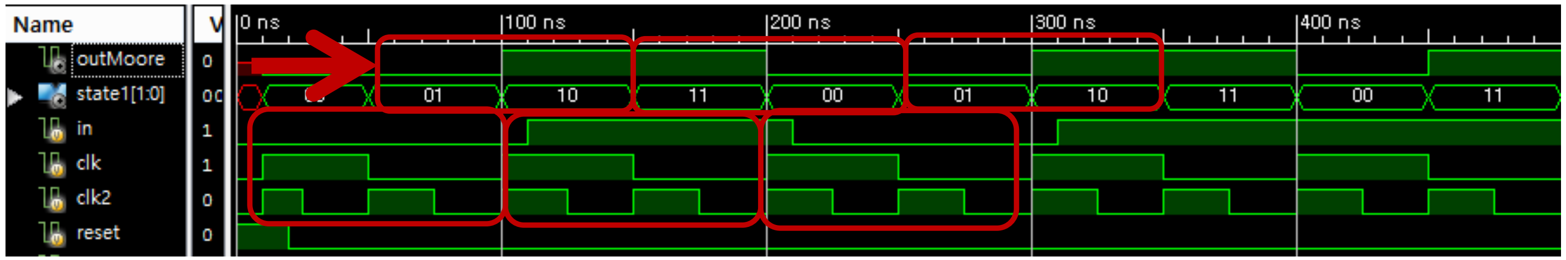
Verilog FSM Implementation Example

- Moore machine





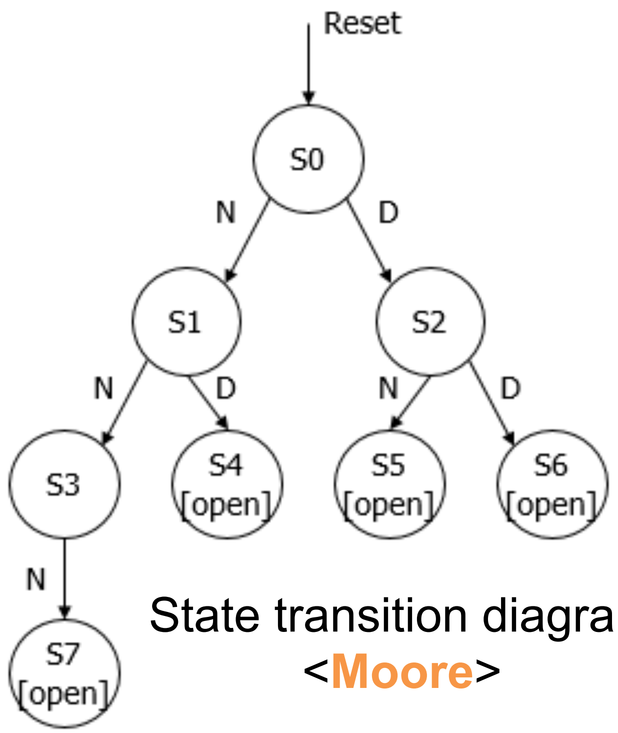
```
module Manchester_Moore(
    input reset,
    input in,
    input clk2,
    output out,
    output reg [1:0] state
    );
    reg [1:0] next;
    localparam s0=0,s1=1,s2=2,s3=3;

    always@(posedge clk2)
        state <= reset ? s0 : next;

    always@(state or in)
        case(state)
            s0, s2:
                next <= ~in ? s1 : s3;
            s1:
                next <= s2;
            s3:
                next <= s0;
            default:
                next <= 2'bx;
        endcase
    assign out = (state==s2) || (state==s3);
endmodule
```



- **Output change is synchronous with state change; output doesn't change during one state**
- **Same as synchronous Mealy**

Today

- Implement the vending machine described in Lecture07_FSM slide p19-p31 in Verilog and load it on the Logic Design Board
  - Implement it as a **Mealy** machine and **Moore** machine
  - Discuss the difference of two types of implementation
    - Difference between Mealy and Moore style

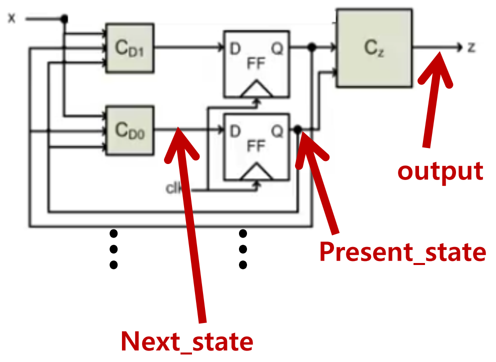


State transition diagram
<**Moore**>

Homework

1. Implement and simulate ‘010’ string recognizer in **Schematic** using Xilinx ISE
   - **1 bit input, 1 bit output**
   - For all other string that isn’t ‘010’, the output must be ‘0’
   - Use D F/Fs to represent states
   - Test with input ‘**0100101001101011**’ to verify your circuit

   - (TIP) Follow the steps below
     1. Draw a state transition diagram and a state transition table
     2. Specify D F/F inputs and one 1-bit output
     3. Draw a circuit block diagram with combinational logic blocks and D F/F like below:



Homework

2-1. Implement your string recognizer in Verilog

2-2. **Try to** test your string recognizer on the Logic Design Board

- If possible, test with input '**0100101001101011**' to verify your implementation
- If not, discuss the problems and possible solutions
  - Ex) What is the problem, how to fix the problem, etc.

- You may get full score if you can discuss (or fix) the problem **properly**, although your design does not work on the FPGA Board!

3. Write a report

- # of pages doesn't matter
- All the files related experiments should be submitted
- The file size should be less than 50MB
- **Due : 11 Nov. (Before class begins at 7:00pm)**