CSE 308: Digital System Design | Homework5 | Due: 1 July 2022

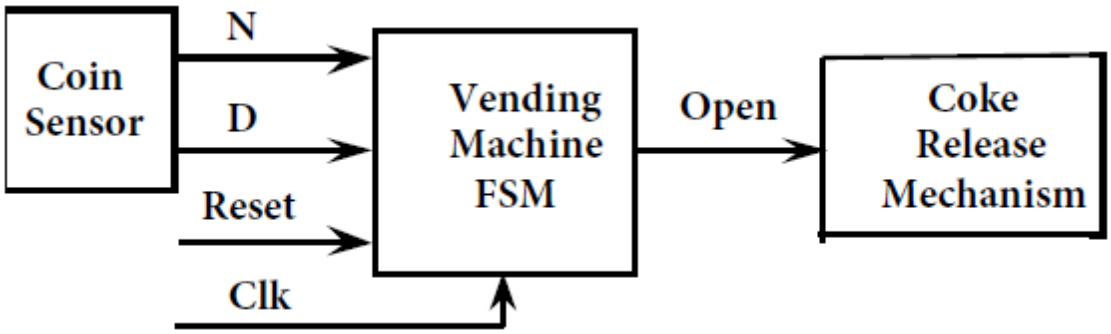
HOMEWORK POLICY

- The purposes of homework include:
  - developing responsible study skills and work habits,
  - practicing a skill or process that students can do independently but not fluently,
  - previewing or preparing for new content, and reflecting or elaborating on information learned in class to deepen students' knowledge.
- Late submission of homework will result in zero score.
- Independently complete homework to the best of your ability.
- If you have any questions related to homework:
  - study with a friend,
  - form a study group in the class,
  - drop your query on Classroom's Stream.
- Collaboration is fine, but it should be you alone who writes up the answers.
- Academic dishonesty will not be tolerated and will result in at least an F for the homework. Your Prospectus clearly states "suspension from the university is the expected penalty" for "plagiarism, cheating, and academic integrity issues" and this includes submitting the work of another person as your own or permitting another to submit yours as his/her own.
- Homework will be checked via the quiz policy. There will usually be one problem directly from the homework. If you have done all the homework, the quiz should be automatic! If you cheat, you may get a zero or negative score. If you do not do the homework, you are very unlikely to do well on the quiz/exam, and you should then expect to fail.

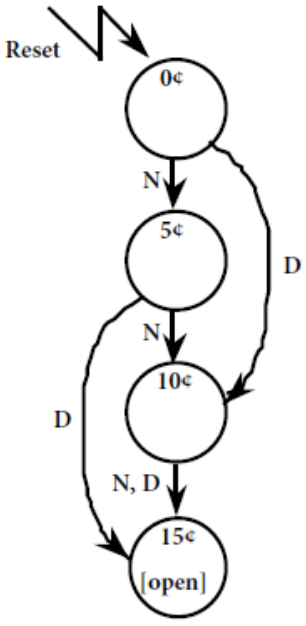
CSE 308: Digital System Design | Homework5 | Due: 1 July 2022

Problem 1: You are working as a Hardware Design Engineer at ASML in the Netherlands (just suppose!). Your boss asked you to design a vending machine with the following specifications.

- delivers a coke after 15 cents deposited
- has a single coin slot for dimes and nickels
- does not return change



Block diagram of the vending machine

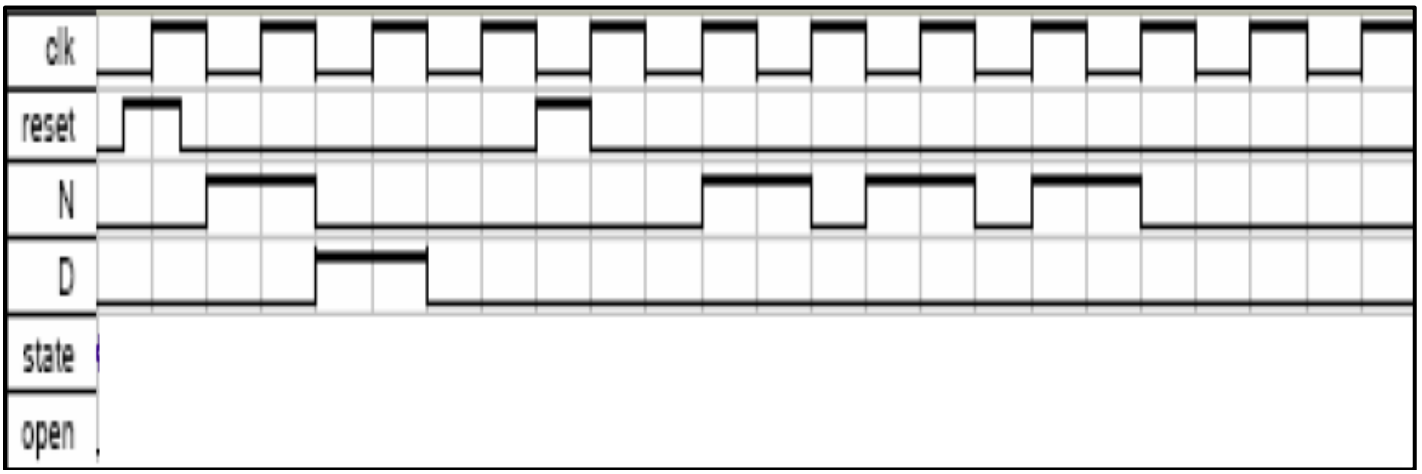


Present State	Inputs D	Inputs N	Next State	Output Open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

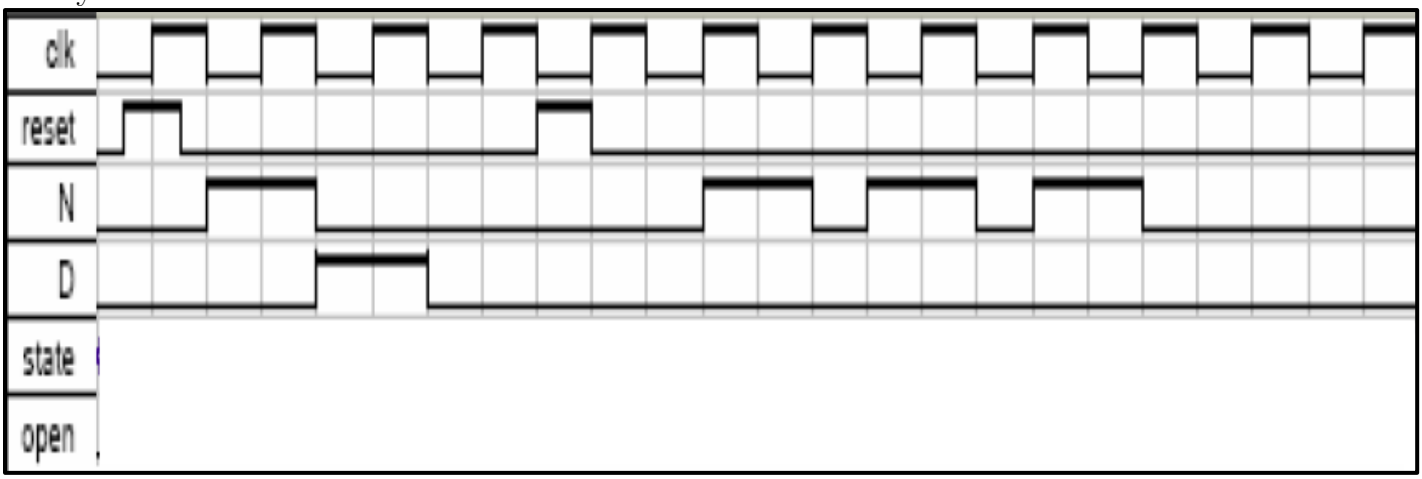
State diagram and state table of the vending machine

(a) Draw a Moore FSM for the above vending machine.
(b) Draw a Mealy FSM for the above vending machine.
(c) Implement the FSMs in (a) and (b) in Verilog.
(d) Show simulation results of your FSMs from (c) for the following values of clk, reset, N, and D.

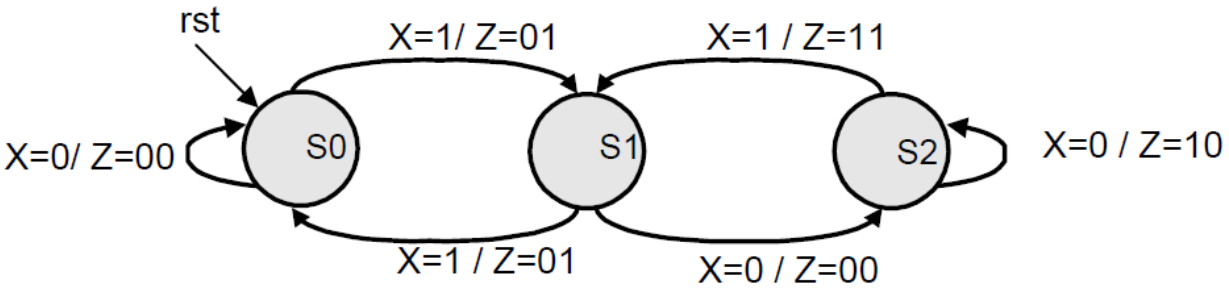
Moore FSM



Mealy FSM



Problem 2:



(a) Complete the Verilog code given below for the FSM shown above. The FSM has a synchronous **reset**, which forces the FSM into state S0, when **reset** is zero. State transitions occur on the positive edge of **clock**. If an invalid state is entered, the **state** should go to 2b'xx and Z should go to 2'bxx until the FSM is reset.

```
module FSM(output reg [1:0] Z, input X, clock, reset);
 reg [1:0]      state, nstate;
 parameter      S0 = 2'00, S1 = 2'b01, S2 = 2'b10;

// implement the state register
always @ (                                          ) begin



  end
// implement the combinational next state and output logic
always @ (                                                  ) begin

end
endmodule
```

(b) Repeat part (a), but implement the FSM using continuous assignments for the output and flip-flop input equations, and the interface to the D flip-flop shown below for the state register.

module dff(q, d, clk, rst, en);

```
module FSM(output [1:0] Z, input X, clock, reset);
wire [1:0]      state, nstate;

// implement the state register



// implement the flip-flop input equations



// implement the output equations



endmodule
```

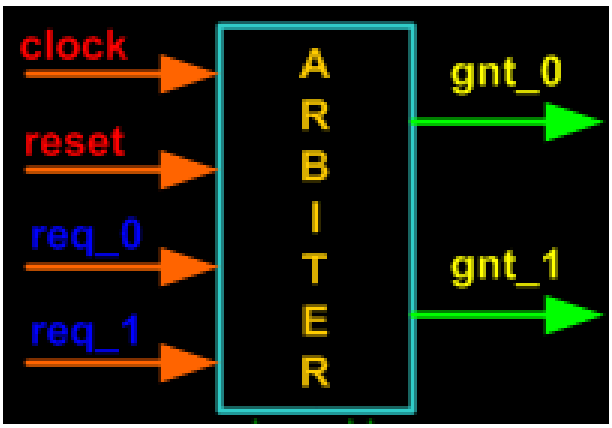
(c) Show your state table and how you derived your input and output.

	nstate		Z	
state	X=0	X=1	X=0	X=1
00				
01				
10				
11				

(d) Complete the testbench for the FSM developed in part (b) that exhaustively tests all of the transitions shown in the state diagram. Your testbench should include a monitor statement that prints the current simulation time and all inputs and outputs when any input (except the clock) or output changes. . Your clock should have a period of 10 time units. Comment your test bench so that it is easy to see which state transitions are tested when applying a particular input value. For example, use "// S0 to S1" when testing the transition from S0 to S1.

```
module t_FSM();

// Declare your nets and variables



// Set up your clock



// Set up your monitor statement



// Instantiate your FSM


// Apply the inputs to test the transitions









endmodule
```

Problem 3: Implement the following simple Arbiter in Verilog. The Arbiter has got two request inputs and two grant outputs, as shown in the signal diagram below.



- When req_0 is asserted, gnt_0 is asserted
- When req_1 is asserted, gnt_1 is asserted
- When both req_0 and req_1 are asserted then gnt_0 is asserted i.e. higher priority is given to req_0 over req_1.

The Arbiter has got following states.

- **IDLE:** In this state the Arbiter waits for the assertion of req_0 or req_1 and drives both gnt_0 and gnt_1 to inactive state (low). This is the default state of the Arbiter; it is entered after the reset and also during fault recovery condition.
- **GNT0:** The Arbiter enters this state when req_0 is asserted, and remains here as long as req_0 is asserted. When req_0 is de-asserted, the Arbiter returns to the IDLE state.
- **GNT1:** The Arbiter enters this state when req_1 is asserted, and remains there as long as req_1 is asserted. When req_1 is de-asserted, the Arbiter returns to the IDLE state.

(a) Draw an FSM for the above Arbiter.
(b) Implement the FSM in (a) in Verilog using two Always blocks.
(c) Implement the FSM in (a) in Verilog using single Always block.