University of Washington – Computer Science & Engineering
Spring 2023 Instructor: Justin Hsia 2023-05-16

CSE 369 QUIZ 2

Name: _Molly_Model____________________

Student ID Number: _1234567________

Please do not turn the page until 2:20.

Instructions

- This quiz contains 4 pages, including this cover page. You may use the backs of the pages for scratch work.
- Please clearly indicate (box, circle) your final answer.
- The quiz is closed book and closed notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, and watches.
- You have 30 (+5) minutes to complete this quiz.

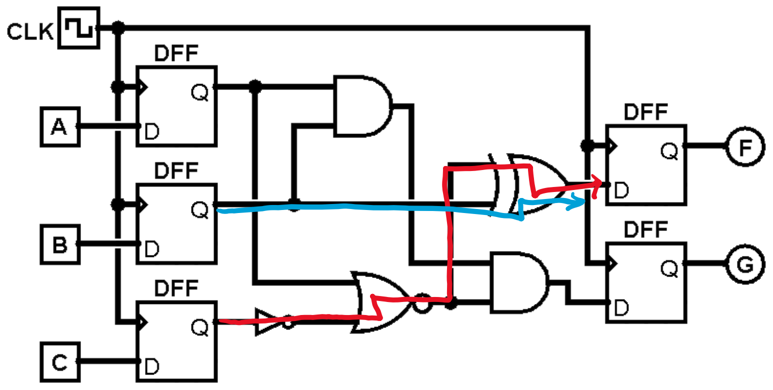
Advice

- Read questions carefully before starting. Read *all* questions first and start where you feel the most confident to maximize the use of your time.
- There may be partial credit for incomplete answers; please show your work.
- Relax. You are here to learn.

Question	Points	Score
(1) SL & Timing	6	6
(2) FSM Implementation	10	10
(3) FSM Design	11	11
Total:	**27**	**27**

Question 1: Sequential Logic & Timing [6 pts]

Consider the following circuit diagram with $t_{\text{setup}} = $ **8 ns** (10^{-9} s), $t_{\text{C2Q}} = $ **10 ns**, $t_{\text{AND}} = $ **20 ns**, $t_{\text{NOR}} = $ **16 ns**, $t_{\text{NOT}} = $ **6 ns**, and $t_{\text{XOR}} = $ **22 ns**.



(A) Calculate the **minimum clock period** that will allow the circuit to function correctly. [3 pts]

$\boxed{62 \text{ ns}}$

The critical path is shown above in red.
We need $t_{\text{C2Q}} + t_{\text{NOT}} + t_{\text{NOR}} + t_{\text{XOR}} \le t_{\text{period}} - t_{\text{setup}}$.
Then $t_{\text{period}} \ge 10 + 6 + 16 + 22 + 8 = 62$ ns.

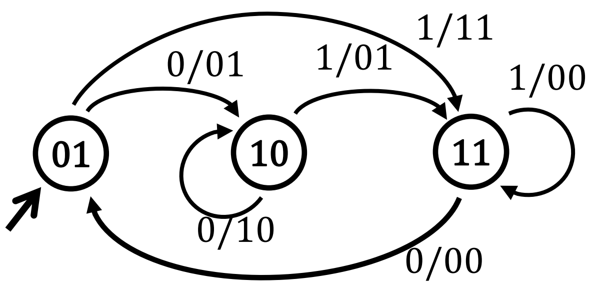
(B) Calculate the **maximum hold time** (t_{hold}) that will allow the circuit to function correctly. [3 pts]

$\boxed{32 \text{ ns}}$

The shortest path to a register input is shown above in blue.
We need $t_{\text{C2Q}} + t_{\text{XOR}} \ge t_{\text{hold}}$.
Then $t_{\text{hold}} \le 10 + 22 = 32$ ns.

Question 2: Finite State Machine Implementation [10 pts]

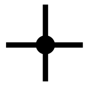
(A) Fill in the provided truth table based on the FSM shown. [2 pts]

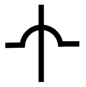


PS_1	PS_0	In	NS_1	NS_0	Out_1	Out_0
0	0	0	X	X	X	X
0	0	1	X	X	X	X
0	1	0	1	0	0	1
0	1	1	1	1	1	1
1	0	0	1	0	1	0
1	0	1	1	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	0	0

(B) Complete the circuit diagram below using *minimal logic* based on the truth table shown below. **Use only 2-input logic gates.** [8 pts]

PS	In_1	In_0	NS	Out
0	0	0	0	1
0	0	1	X	X
0	1	0	0	0
0	1	1	0	0
1	0	0	1	0
1	0	1	X	X
1	1	0	1	0
1	1	1	0	1

Wire connection: 

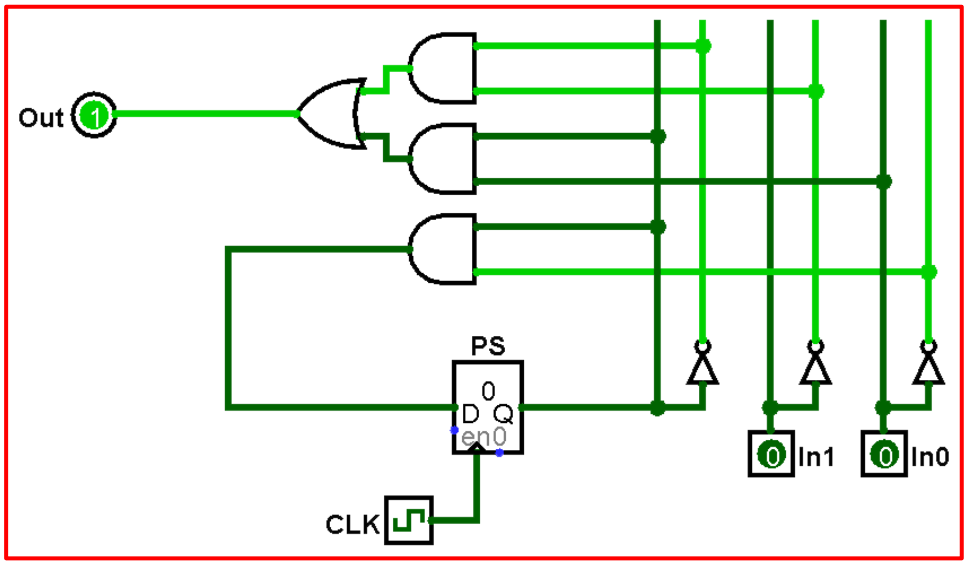
Wire crossing: 

PS \ In	00	01	11	10
0	0	X	0	0
1	1	X	0	1

NS = $PS \cdot \overline{In_0}$

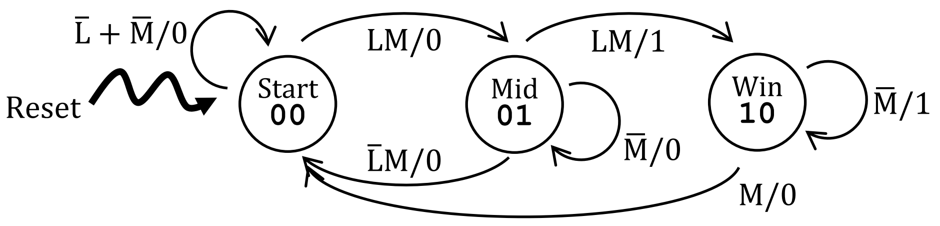
PS \ In	00	01	11	10
0	1	X	0	0
1	0	X	1	0

Out = $\overline{PS} \cdot \overline{In_1} + PS \cdot In_0$



Question 3: Finite State Machine Design [11 pts]

The following FSM represents a *Red Light, Green Light game*, where a player is only allowed to move forward (`M = 1`) when the light is green (`L = 1`). Here, the player wins (output `W = 1`) after successfully moving twice; moving when the light is red (`L = 0`) results in returning to the start.



(A) How many total rows are in the truth table for this FSM? How many of the rows are filled with Don't Cares?

2 state + 2 input bits → $2^4 = 16$ rows in TT.
One missing state which has 4 transitions.

Rows: **16**	Don't Care Rows: **4**

(B) Complete the testbench `initial` block to *thoroughly* test JUST the `Start` and `Mid` states. You need to fill in all bolded blanks. You are welcome to fill out the Verilog comments to help you keep track of state, but these will not be graded. [7 pts]

```
initial begin
                      L <= 0;     M <= 0;     // state: 0
    @(posedge clk);   L <= 1___;  M <= 1___;  // state: 0__
    @(posedge clk);   L <= 1;     M <= 0;     // state: 1__
    @(posedge clk);   L <= 0;     M <= 1;     // state: 1__
    @(posedge clk);   L <= 0___;  M <= 1___;  // state: 0__
    @(posedge clk);   L <= 1;     M <= 0;     // state: 0__
    @(posedge clk);   L <= 1;     M <= 1;     // state: 0__
    @(posedge clk);   L <= 0___;  M <= 0___;  // state: 1__
    @(posedge clk);   L <= 1;     M <= 1;     // state: 1
    @(posedge clk);
     ... // test the Win state
    @(posedge clk);
    $stop();
end
```

(C) If we change the game so that it takes THREE successful moves to win, what would your updated answers be for Part A? [2 pt]

Still 2 state + 2 input bits → $2^4 = 16$ rows in TT.
All states present now.

Rows: **16**	Don't Care Rows: **0**