University of Washington – Computer Science & Engineering

Winter 2020 Instructor: Justin Hsia 2020-02-25

CSE 369 QUIZ 2

Name: _Perry_Perfect_____

Student ID Number: _1234567_____

Please do not turn the page until 11:30.

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 25 minutes to complete this quiz.

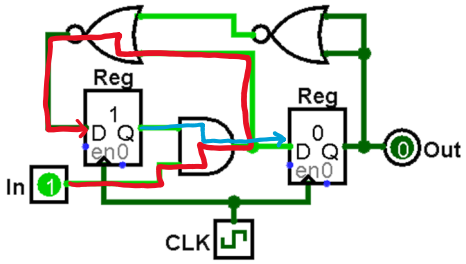
Advice

- 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	10	10
Total:	**26**	**26**

Question 1: Sequential Logic & Timing [6 pts]

Consider the following circuit diagram with t_{setup} = **12 ns** (10^{-9} s), t_{hold} = **18 ns**, t_{C2Q} = **8 ns**, t_{AND} = **25 ns**, and t_{NOR} = **20 ns**. Assume that `In` changes **9 ns** after every clock trigger.



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

$t_{\text{period}} \geq$ **66 ns**

The critical path is shown above in red.
We need $t_{\text{In}} + t_{\text{AND}} + t_{\text{NOR}} \leq t_{\text{period}} - t_{\text{setup}}$.
Then $t_{\text{period}} \geq 9 + 25 + 20 + 12 = 66$ ns.

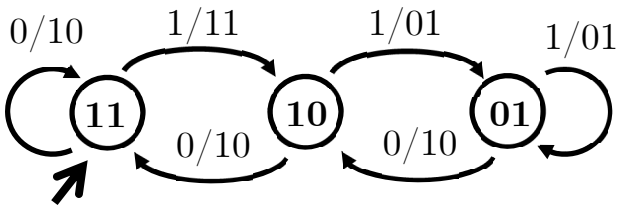
(B) If we swap out our AND gate for one with a different combinational delay, what is the **minimum t_{AND}** that will allow the circuit to function correctly. Make sure to *include units*. [3 pts]

$t_{\text{AND}} \geq$ **10 ns**

The shortest path to a register input is shown above in blue.
We need $t_{\text{C2Q}} + t_{\text{AND}} \geq t_{\text{hold}}$.
Then $t_{\text{AND}} \geq 18 - 8 = 10$ ns.

Question 2: Finite State Machine Implementation [10 pts]

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



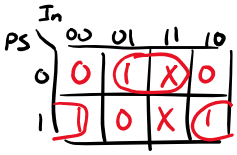
PS_1	PS_0	In	NS_1	NS_0	Out_1	Out_0
0	0	0	X	X	X	X
0	0	1	X	**X**	**X**	X
0	1	0	1	0	1	0
0	1	1	**0**	1	0	**1**
1	0	0	1	1	1	0
1	0	1	0	**1**	**0**	1
1	1	0	1	1	1	0
1	1	1	**1**	0	1	**1**

(B) Complete the circuit diagram below using *minimal logic* based on the truth table shown below. You are welcome to use 2- and 3-input logic gates. [8 pts]

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

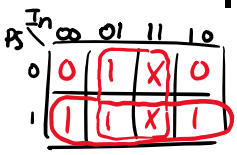
Wire connection:

Wire crossing:

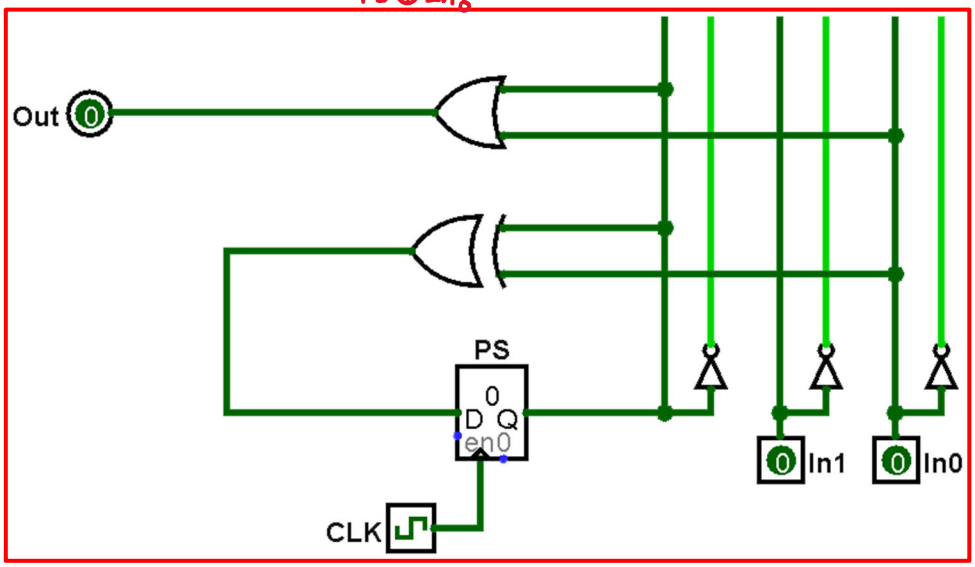


$NS = PS \cdot \overline{In_0} + \overline{PS} \cdot In_0$

$PS \oplus In_0$

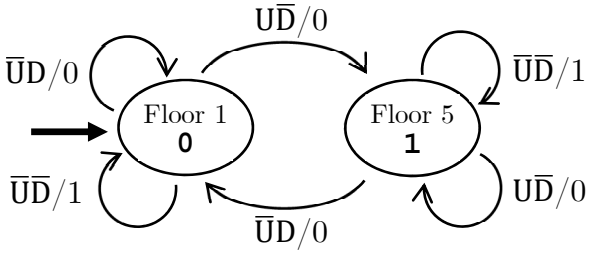


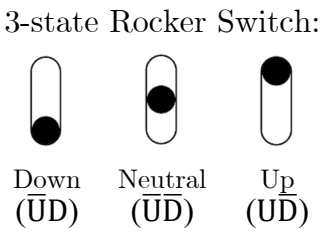
$Out = PS + In_0$



Question 3: Finite State Machine Design [10 pts]

Justin has built a *secret elevator* that goes directly to his office! It is controlled by 3-state rocker switch (see diagram below) that passes the input signals up (`U`) and down (`D`). The output is whether the elevator doors are open (1) or closed (0).





(A) How many total rows are in the truth table for the 3-way switch FSM? How many of the rows are filled with Don't Cares? [2 pt]

1 state + 2 input bits → $2^3 = 8$ rows in TT.
Each arrow covers 1 transition, so 6 are present.

Rows: **8**	Don't Care Rows: **2**

(B) Complete the testbench `initial` block to *thoroughly* test the state diagram. Even though they may be unnecessary, please fill in all blanks. You are welcome to fill out the Verilog comments to help you keep track of state, but these will not be graded. [5 pts]

```
initial begin
                          U <= 0;      D <= 1;      // state: 0
      @(posedge clk);     U <= 1___;   D <= 0___;   // state: 0__
      @(posedge clk);     U <= 1;      D <= 0;      // state: 1__
      @(posedge clk);     U <= 0___;   D <= 0___;   // state: 1__
      @(posedge clk);     U <= 0;      D <= 1;      // state: 1__
      @(posedge clk);     U <= 0___;   D <= 0___;   // state: 0__
      @(posedge clk);
      $stop();
end
```

(C) The rocker switch imposes a physical constraint: when going between the up and down states, you *must* pass through the neutral state. Does your testbench above conform to this constraint or not? Explain *briefly*. [3 pt]

No, it does not. Between the first two clock cycles, we jump from $\overline{U}D$ to $U\overline{D}$.