University of Washington – Computer Science & Engineering

Winter 2019 Instructor: Justin Hsia 2019-02-26

CSE 369 QUIZ 2

Name: _Perry_Perfect_____

UWNetID: _perfect_____

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.

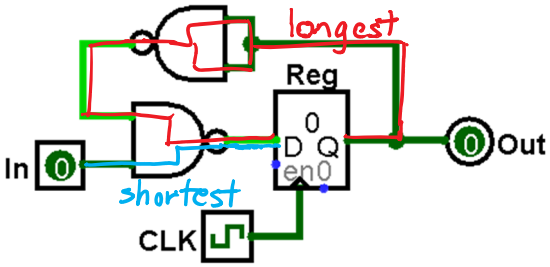
Advice

- 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	9	9
(3) FSM Design	11	11
Total:	**26**	**26**

Question 1: Sequential Logic & Timing [6 pts]

Consider the following circuit diagram with t_{period} = **150 ns** (10^{-9} s), t_{NAND} = **35 ns**, and t_{C2Q} = **50 ns**. Assume that `In` changes **10 ns** after every clock trigger.



(A) Calculate the **maximum setup time** that will allow the circuit to function correctly. Make sure to *include units*. [3 pts]

$t_{setup} \leq$ **30 ns**

The *critical path* is shown above in red.
So we need $t_{C2Q} + t_{NAND} + t_{NAND} \leq t_{period} - t_{setup}$.
Then $t_{setup} \leq 150 - 50 - 35 - 35 = 30$ ns.

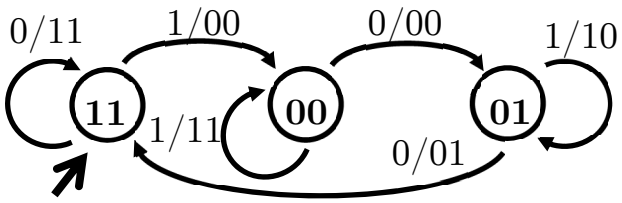
(B) Calculate the **maximum hold time** that will allow the circuit to function correctly. Make sure to *include units*. [3 pts]

$t_{hold} \leq$ **45 ns**

The shortest path to the register input is shown above in light blue.
So we need $t_{In} + t_{NAND} \geq t_{hold}$.
Then $t_{hold} \leq 10 + 35 = 45$ ns.

Question 2: Finite State Machine Implementation [9 pts]

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



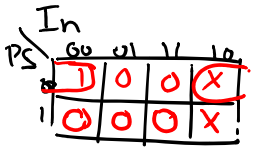
PS_1	PS_0	In	NS_1	NS_0	Out_1	Out_0
0	0	0	0	1	**0**	0
0	0	1	0	0	1	**1**
0	1	0	1	1	**0**	1
0	1	1	0	1	**1**	0
1	0	0	X	**X**	X	X
1	0	1	X	**X**	X	X
1	1	0	**1**	1	1	1
1	1	1	0	**0**	0	0

(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. [7 pts]

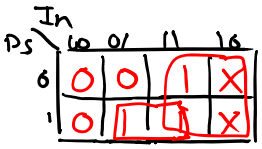
PS	In_1	In_0	NS	Out
0	0	0	1	0
0	0	1	0	0
0	1	0	X	X
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	X	X
1	1	1	0	1

Wire connection:

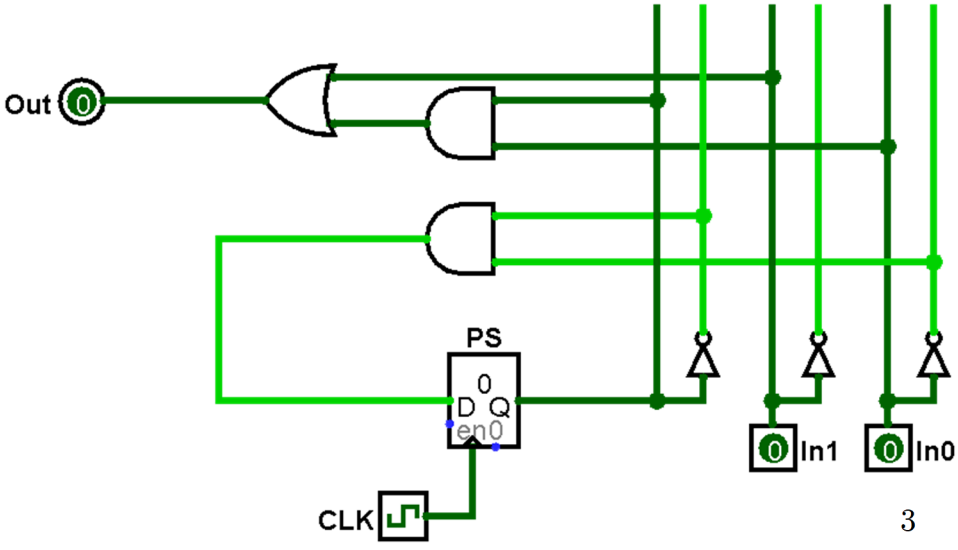
Wire crossing:



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

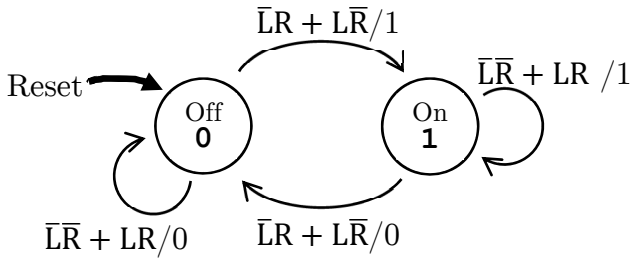


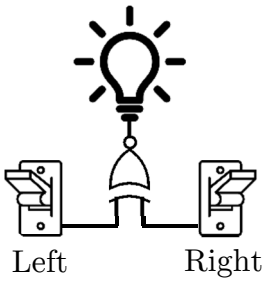
$Out = In_1 + PS \cdot In_0$



Question 3: Finite State Machine Design [11 pts]

The following FSM represents a *3-way switch*, where two switches (left and right) control the same light. If both switches are in the same position (*i.e.* both up or both down), then the output (light bulb) is on (1), otherwise it is off (0). Here the inputs L and R are 1 if someone flips (off→on or on→off) the left or right switch, respectively.





(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 2 transitions, so 8 are present.

Rows: **8**	Don't Care Rows: **0**

(B) What is the max number of transitions per state in this FSM? [2 pt]

2 input bits → $2^2 = 4$ transitions per state.

Max Transitions Per State: **4**

(C) 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. [7 pts]

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