University of Washington – Computer Science & Engineering

Winter 2017 Instructor: Justin Hsia 2017-02-21

CSE 369 QUIZ 2

Name: _Perry_Perfect__________________

UWNetID: _1234567______________________

Please do not turn the page until 10:30.

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.

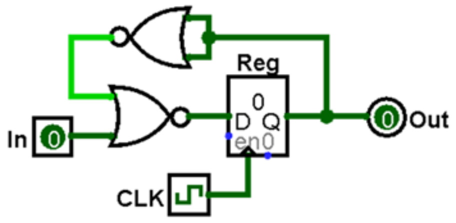
Advice

- 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	7	7
(2) FSM Implementation	10	10
(3) FSM Design	10	10
Total:	**27**	**27**

Question 1: Sequential Logic & Timing [7 pts]

Consider the following circuit diagram with t_{setup} = **60 ps** (10^{-12} s), t_{hold} = **40 ps**, t_{C2Q} = **140 ps**, and t_{NOR} = **200 ps**. Consider each part below *independently* and fill in your answers in the boxes below, making sure to ***include units***.



(A) If the input `In` changes exactly on clock triggers, what is the minimum clock period that we can use and still ensure proper behavior? [3 pts]

600 ps

The critical path is from the output of the register, through the two NOR gates, and back to the input of the register.

$t_{C2Q} + t_{NOR} + t_{NOR} \leq t_{period} - t_{setup}$
$140 + 200 + 200 \leq t_{period} - 60$
$t_{period} \geq 600$ ps

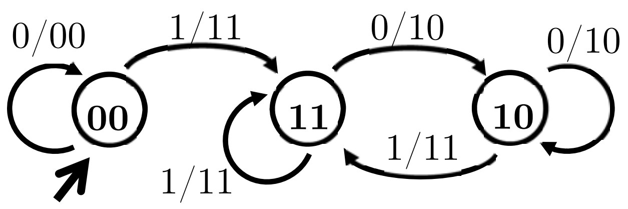
(B) If we fix the clock period at **750 ps**, what range of times (measured from each clock trigger) will changing the input `In` cause a *setup time violation*? Answer using inclusive interval notation: $[t_{start}, t_{end}]$. [4 pts]

[_**490**_ , _**550**_] ps

A change between $[t_{period} - t_{setup}, t_{period}] = [690,750]$ ps at the register input will cause a setup time violation. After `In` changes, it takes $t_{NOR} = 200$ ps delay before it reaches the input of the register, so we shift this interval back 200 ps.

Question 2: Finite State Machine Implementation [10 pts]

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



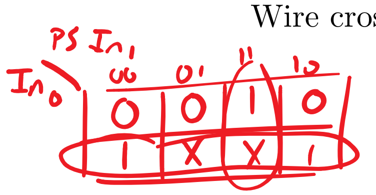
PS$_1$	**PS$_0$**	**In**	**NS$_1$**	**NS$_0$**	**Out$_1$**	**Out$_0$**
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	X	X	X	X
0	1	1	X	X	X	X
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	0	1	0
1	1	1	1	1	1	1

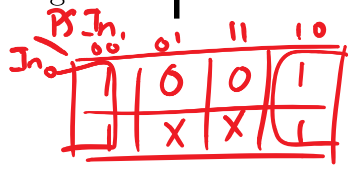
(B) Complete the circuit diagram below using *minimal logic* based on the truth table shown below. You are welcome to use 2- and 3-input logic gates. [8 pts]

PS	**In$_1$**	**In$_0$**	**NS**	**Out**
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	X	X
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	X	X

Wire connection:

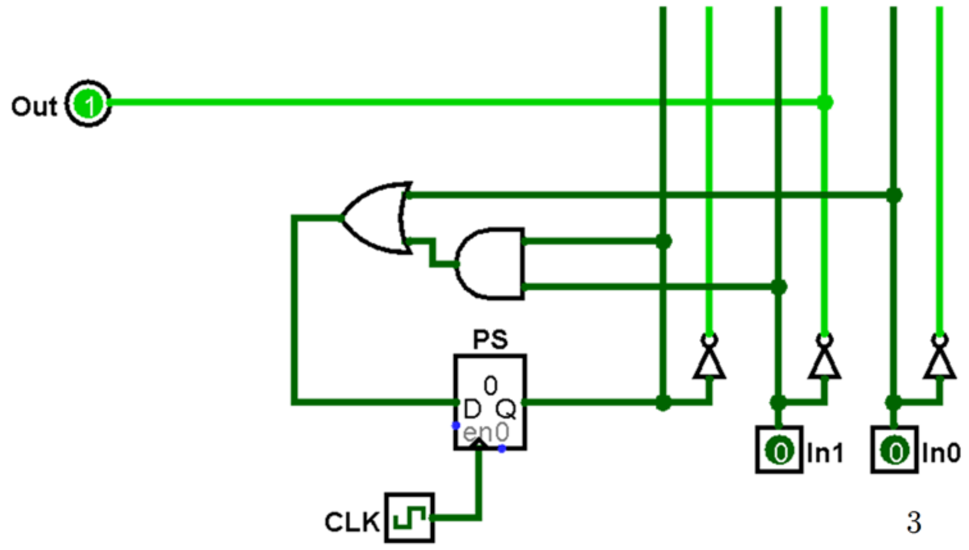
Wire crossing:





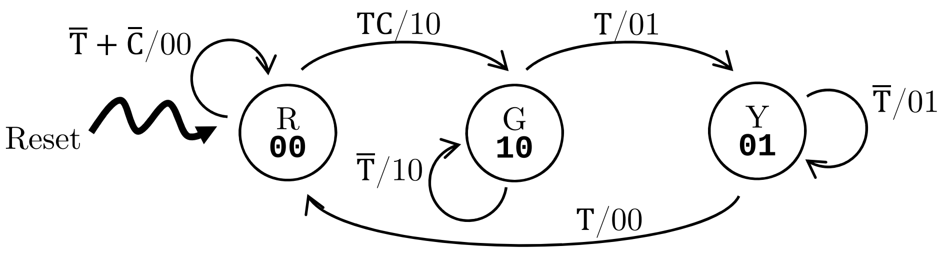
$NS = In_0 + PS \cdot In_1$

$Out = \overline{In_1}$



Question 3: Finite State Machine Design [10 pts]

The following FSM represents a stop light that is controlled by a timer (input `T` pulses high at regular intervals) and a sensor that signals high when a car is stopped at the intersection (input `C`). The light outputs the colors 00 – red, 01 – yellow, 10 – green:



(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 pt]

Only don't cares are state 11.

Rows: $2^4 = 16$	Don't Care Rows: 4

(B) The testbench `initial` block below doesn't cover every transition! In the table on the right, write out the *four* missing state and input combinations. Don't include Don't Care situations. You are welcome to fill out the Verilog comments to help you keep track of state, but these will not be graded. [8 pts]

```
initial begin
                    T <= 0;  C <= 0;  // state:  00
   @(posedge clk);  T <= 1;  C <= 0;  // state: _00_
   @(posedge clk);  T <= 1;  C <= 1;  // state: _00_
   @(posedge clk);  T <= 0;  C <= 0;  // state: _10_
   @(posedge clk);  T <= 0;  C <= 1;  // state: _10_
   @(posedge clk);  T <= 1;  C <= 1;  // state: _10_
   @(posedge clk);  T <= 0;  C <= 1;  // state: _01_
   @(posedge clk);  T <= 1;  C <= 0;  // state: _01_
   @(posedge clk);                    // state: _00_
   $stop();
end
```

PS_1	PS_0	T	C
0	0	0	1
1	0	1	0
0	1	0	0
0	1	1	1