University of Washington – Computer Science & Engineering

Spring 2021 Instructor: Clarice Larson 2021-06-01

CSE 369 QUIZ 3

Name: _____

UWNetID: _____

Please do not turn the page until 11:40.

Instructions

- This quiz contains 4 pages, including this cover page.
- Show scratch work for partial credit, but put a box around final answers.
- The quiz is open book and open notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- You have 40 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) Counters	12	
(2) Shift Registers	9	
(3) Check Digit	11	
Total:	**32**	

Question 1: Counters [12 pts]

Implement a counter that goes through the following state sequence: **000** → **001** → **011** → **111** → **110** → **100** → 000 → ... using a *minimal number of 2-input logic gates*.

(A) Complete the truth table below. [3 pts]

PS_2	PS_1	PS_0	NS_2	NS_1	NS_0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

(B) Complete the K-maps below and find the *minimum sum-of-products solutions*. [6 pts]

NS_2

	00	01	11	10
0				
1				

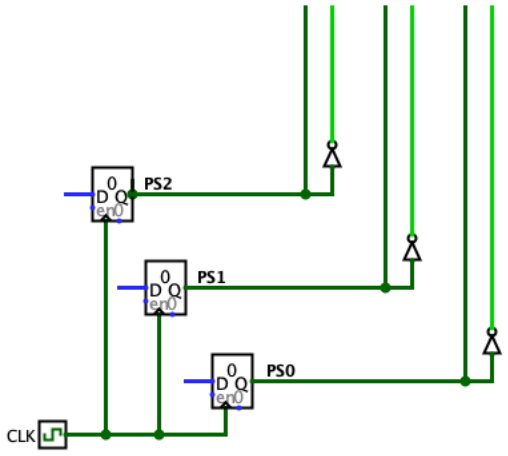
NS_1

	00	01	11	10
0				
1				

NS_0

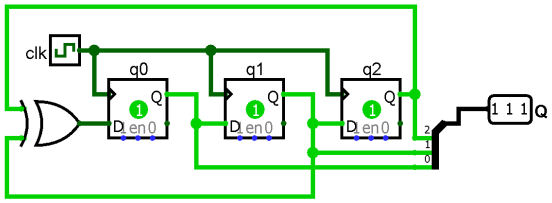
	00	01	11	10
0				
1				

(C) Draw your minimal logic circuit. [3 pts]



Question 2: Shift Registers [9 pts]

We are using the 3-bit LFSR shown below as a pseudo-random number generator. Assume we start in state 111.

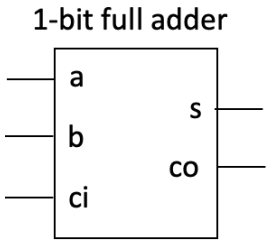


(A) Draw out the full state transition diagram (i.e. include ALL states) for this LFSR below: [4 pts]

(B) What is/are the sink state(s) of this LFSR? [1 pt]

Sinks(s): _____

(C) We only have a 1-bit full adder to implement the logic gate connected to the input of q0 shown above. Show how you would hook up the bits of your LFSR to a full adder to generate the correct input for q0. You may use constants 0 and 1 as needed. [4 pts]



Question 3: Check Digit [11 pts]

Check digit algorithms can be used to verify that a decimal number has been entered correctly. The following check digit algorithm validates 4-digit numbers using the rightmost check digit:

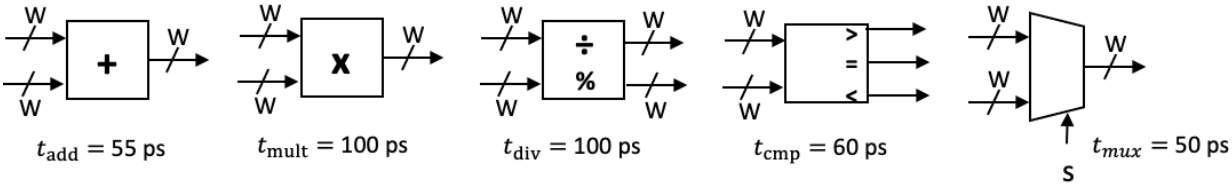
1. If the check digit value equals 0, then set the check digit value to 10.
2. Multiply every *odd* digit by 3 excluding the check digit value.
3. Sum all of the digits excluding the check digit value.
4. Divide the sum by 10 and add the remainder to the check digit value. If the sum = 10, the number is valid.

Examples of valid numbers using this algorithm are 2240 and 8419.

Example 1:

Digits:	$d_3 = 2$	$d_2 = 2$	$d_1 = 4$	check = 0
If check digit = 0, set check digit to 10:	2	2	4	**10**
Multiply odd digits (d_3, d_1) by 3:	**6**	2	**12**	**10**
Sum digits:	20			
(Sum % 10) + check:	10		2240 is *valid* ☑	

Implement the check digit algorithm. You can add **constants** and the following **2-input logic blocks** (with specified combinational delays):



(A) Using the logic blocks above, complete the validation circuit below. Assume the clock inputs are connected properly for you. [8 pts]

(B) Assume $t_{hold} = 15$ ps, $t_{setup} = 30$ ps, and $t_{C2Q} = 75$ ps. How long after a clock trigger does your circuit from Part A take to compute the final value of Valid? Be sure to *include units*. [3 pt]