University of Washington – Computer Science & Engineering

Spring 2018 Instructor: Justin Hsia 2018-05-29

CSE 369 QUIZ 3

Name: _Perry_Perfect_____

UWNetID: _perfect_____

Please do not turn the page until 10:30.

Instructions

- This quiz contains 4 pages, including this cover page.
- Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- 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 35 minutes to complete this quiz.

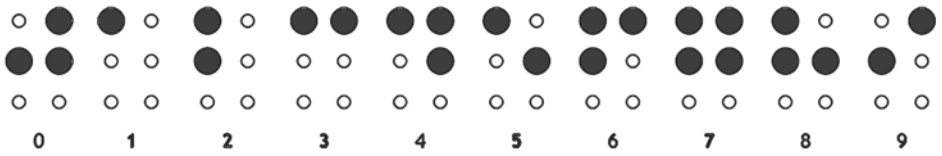
Advice

- 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) Building Blocks	12	12
(2) Shift Registers	9	9
(3) Rotate Right	11	11
Total:	**32**	**32**

Question 1: Building Blocks [12 pts]

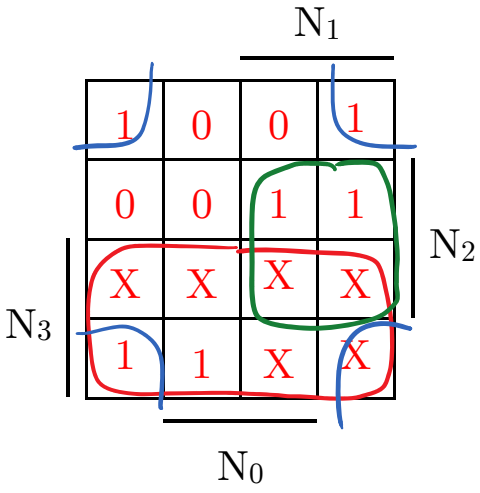
We want to build a **binary-to-braille decoder circuit**. Braille is represented by 6 dots, as shown below. Circles that are black/white represent LEDs that are on (1)/off (0), respectively. The binary-coded digit is given in bits N_3–N_0.



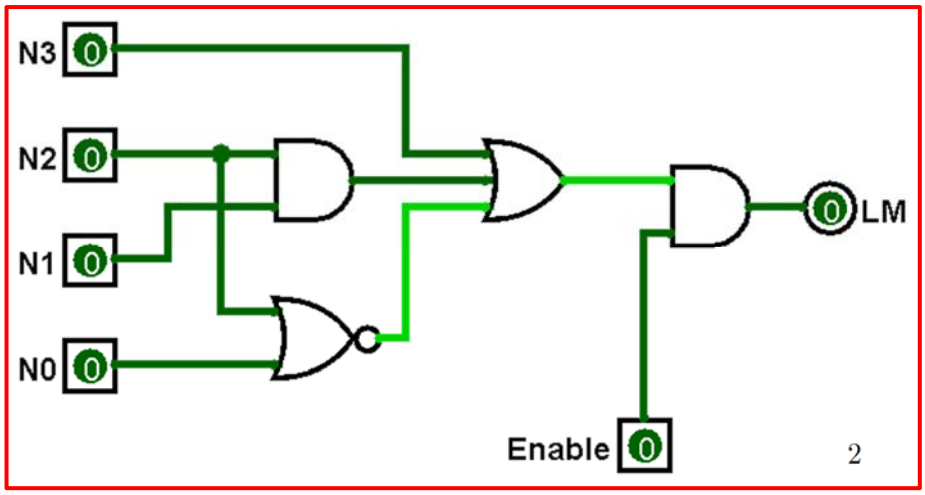
Implement the *simplest* two-level logic for an *enabled* circuit below for the **left-middle dot (LM)**. You may use any 1- to 3-input logic gates discussed in the class.

- The dot should always be off (0) if `Enable` = 0
- Your truth table *will be graded*.

N_3	N_2	N_1	N_0	LM
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

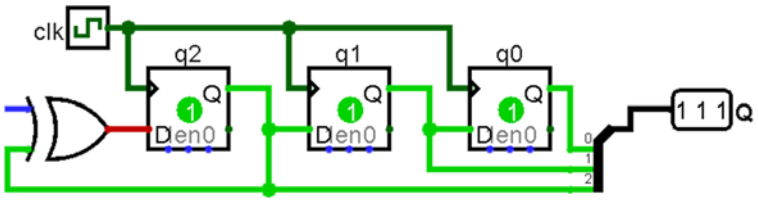


$LM = N_3 + \overline{N_2}\,\overline{N_0} + N_2 N_1$



Question 2: Shift Registers [9 pts]

(A) We are designing a 3-bit LFSR to use as a pseudo-random number generator, but we only have a 2-input XOR gate available. Assuming we start in state **111**, which state bit (`q1` or `q0`) should we connect along with `q2` to the XOR gate in order to get the longest possible cycle? What is the length of the cycle we end up in? [5 pt]



a	b	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Bits:	`q2` and `_q0_`
Cycle length:	7

q1: 111 → 011 → 101 → 110

q0: 111 → 011 → 101 → 010 → 001 → 100 → 110

(B) The Verilog code below is supposed to implement an *enabled* version of the LFSR above with bits **`q1` and `q0`** connected to the XOR gate. Find errors in the code and rewrite the offending *lines* in the boxes. [4 pt]

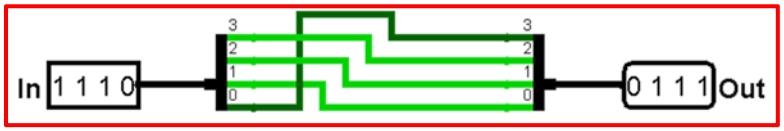
```
module LFSR (Q, enable, reset, clk);
  input            enable, reset, clk;
  output reg [2:0] Q;

  always_ff @(negedge clk)
    if ( reset )
      Q <= 3'b000;
    else if ( enable )
      Q <= { Q[0] ^ Q[1], Q[1], Q[0] };

endmodule
```

1) `always_ff @(posedge clk)`
2) `Q <= 3'b111;`
3) `Q <= { Q[0] ^ Q[1], Q[2], Q[1] };`

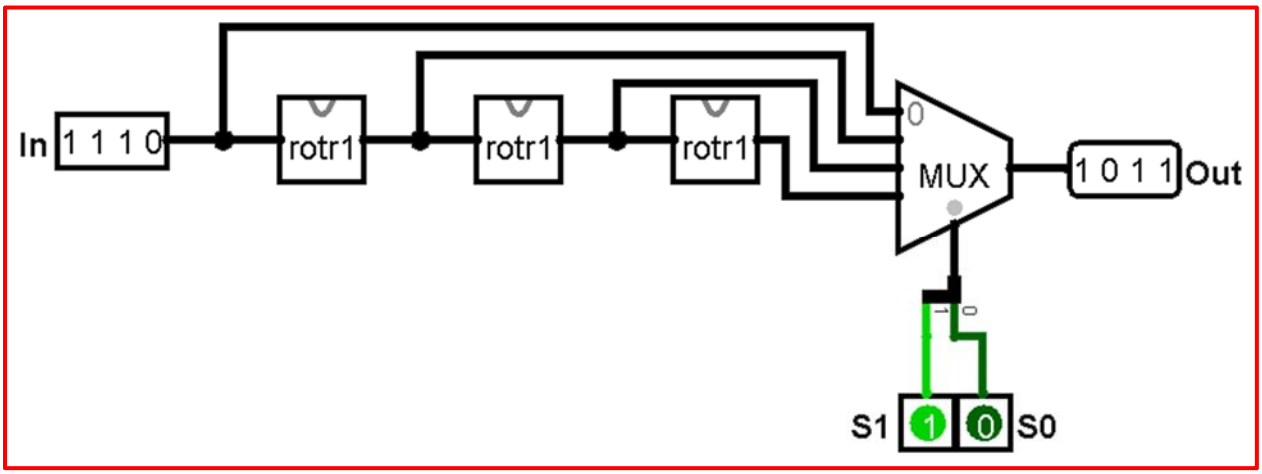
Question 3: Rotate Right [11 pts]

A **rotate right** circuit is similar to a right shift, except that bits that are "shifted off" wrap around and get "shifted in" on the other side. For example, 0b1110 rotated right by 1 is 0b0111.

(A) Below we show two *splitters*, which are used to "break out" busses into wires and vice-versa. Pay special attention to the wire numbering! Wire up the diagram below to perform the **rotate right by 1** operation between `In` and `Out`. [2 pts]



(B) Assume the circuit from Part A can be represented as a block labeled "**rotr1**." Using these blocks and routing elements, implement a **4-bit rotate right circuit** that takes in a 4-bit input and 2 select bits indicating how many times to rotate right (0-3). Make sure it is clear what the ports and directions are. [6 pts]



(C) Chaining blocks together often leads to slower circuits, so Harry the Husky is arguing that we should re-implement our Part B circuit without the **rotr1** blocks. Do you agree with him? Why or why not? [3 pts]

No. The rotr1 block is made entirely of wires, which transmit information effectively instantaneously, so chaining rotr1 blocks does not add to the combinational delay.