University of Washington – Computer Science & Engineering

Winter 2020 Instructor: Justin Hsia 2020-03-10

CSE 369 QUIZ 3

Name: _Perry_Perfect___________________

Student ID Number: _1234567________

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 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 40 minutes to complete this quiz.

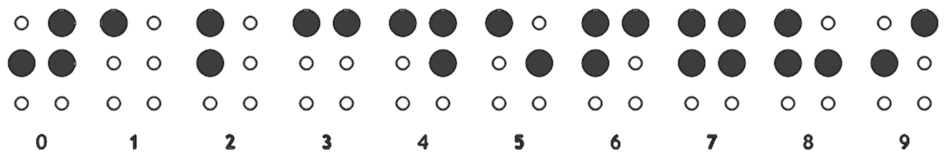
Advice

- 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) Building Blocks	12	12
(2) Shift Registers	10	10
(3) Sequential Computation	10	10
Total:	**32**	**32**

Question 1: Building Blocks [12 pts]

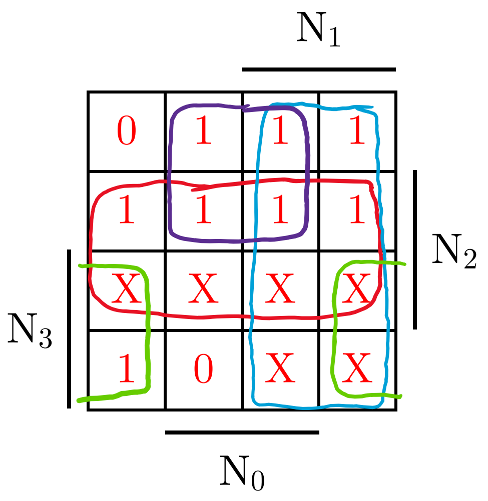
We want to build a **binary-to-braille decoder circuit**. Braille is represented by 6 dots, as shown below. Circles that are black/white represent LEDs that are on (1)/off (0), respectively. The binary-coded digit is given in bits N_3–N_0.



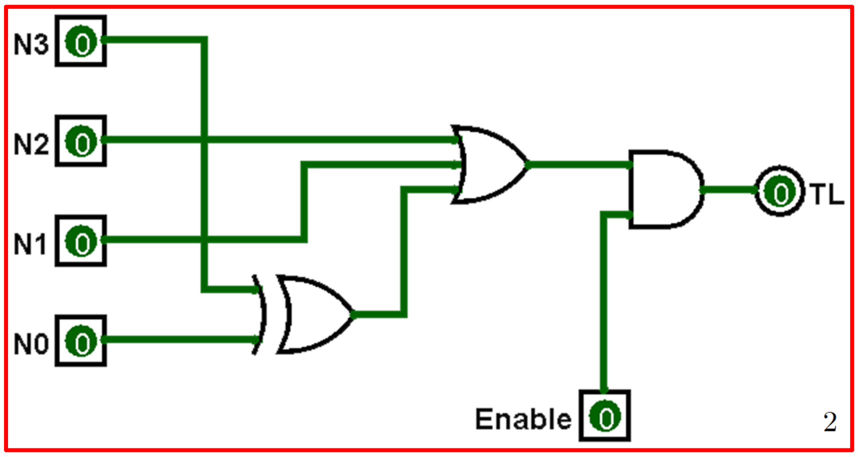
Implement the *simplest* two-level logic for an *enabled* circuit below for the **top-left dot (TL)**. You may use any 1- to 3-input logic gates discussed in the class.

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

N_3	N_2	N_1	N_0	**TL**
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**1**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**0**
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**

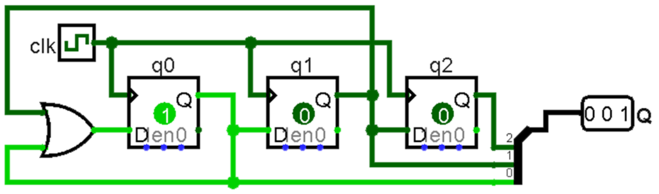


$$TL = N_2 + N_1 + \overline{N_3}N_0 + N_3\overline{N_0}$$
$$= N_2 + N_1 + (N_3 \oplus N_0)$$



Question 2: Shift Registers [10 pts]

We are using a 3-bit LFSR as a pseudo-random number generator by connecting **q1** and **q0** to a 2-input OR gate. As shown below (look at Q), we are shifting bits to the LEFT.



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

$$\begin{tikzcd}
 & & 010 \arrow[dr, "*"] & & 001 \arrow[d, "*"] & \\
100 \arrow[r, "*"] & 000 \arrow[loop below, "*"] & & 101 \arrow[r, "*"] & 011 \arrow[r, "*"] & 111 \arrow[loop below, "*"] \\
 & & 110 \arrow[ur, "*"] & & &
\end{tikzcd}$$

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

Sink(s): **000, 111**

(C) Complete the Verilog implementation below. [3 pt]

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

  always_ff @(posedge clk)
    if ( reset )        // choose a state that yields the
      Q <= 3'b_010_;    //  longest non-repeating chain
    else if ( enable )
      Q <= { __Q[1]__, __Q[0]__, __Q[1] | Q[0]__ };

endmodule
```

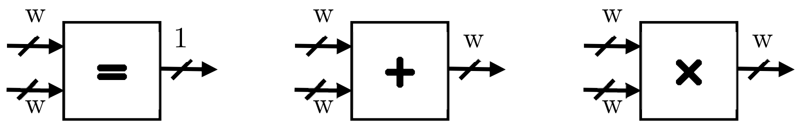
(D) Is using OR in an LFSR a good choice? *Briefly* explain. [2 pt]

No. You are guaranteed to eventually end up in a sink state, which means you will always get the same "random" number afterwards.

Question 3: Sequential Computation [10 pts]

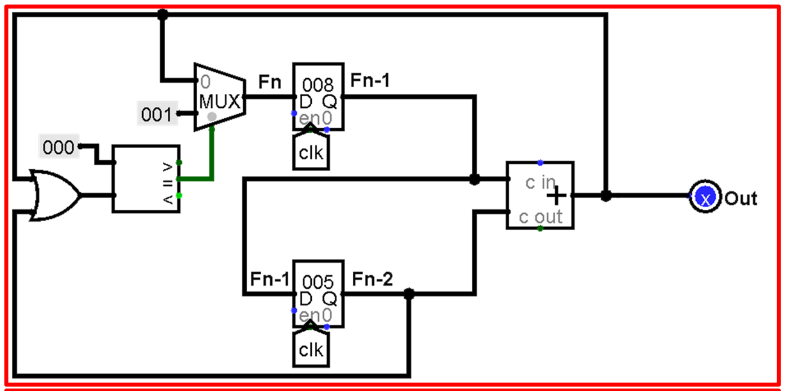
Implement a circuit that computes the **Fibonacci sequence** $F_n = F_{n-1} + F_{n-2}$. Note that it will take about `n` clock cycles to compute F_n and we will let it run infinitely (*no stop condition*).

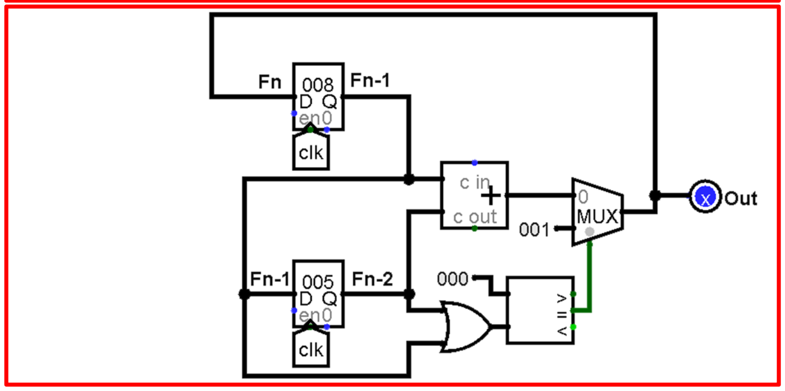
- Both registers (after a `Reset`, which is not shown) start with value 0, but you will need to make sure that your circuit doesn't get stuck there! It is suggested that you tackle this part *last*. Hint: you need to detect this very special situation (both registers with value 0).
- You can freely use gates and routing elements discussed in class plus the constants **`0`** and **`1`** and the following logic blocks (where `w` is the bus width of our Fibonacci circuit):



Two possible solutions:
(the top has 1 cycle of output 0 while the bottom one doesn't)

(any solution that produced 1, 1, 2, 3, ... after 0-2 cycles of 0 were accepted)





Many other working alternatives exist:

- MUX could be placed on the input to the lower register or on the F_{n-2} input to the adder.
- MUX behavior could be achieved with a variety of logical combinations, though using an adder instead usually led to a bus width mismatch (both inputs to the adder should be w-bits wide, whereas the output of a comparator or 2-input logic gate is 1-bit wide).
- We were lenient with bit-widths of constants and with the combinational logic. 2w-to-1 gates were accepted (particularly 2w-to-1 OR or NOR).