University of Washington – Computer Science & Engineering

Spring 2024 Instructor: Justin Hsia 2024-05-28

CSE 369 QUIZ 3

Name: _____

Student ID Number: _____

Please do not turn the page until 1:40.

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 60 (+10) minutes to complete this quiz.

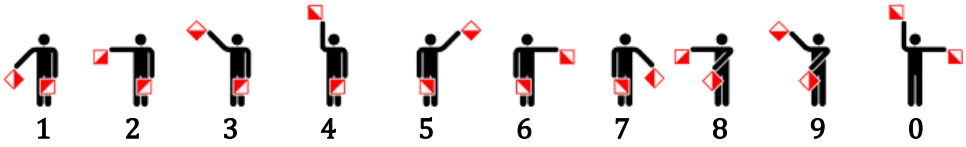
Advice

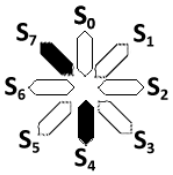
- Read questions carefully before starting. Read *all* questions first and start where you feel the most confident to maximize the use of your time.
- There may be partial credit for incomplete answers; please show your work.
- Relax. You are here to learn.

Question	Points	Score
(1) Decoders	13	
(2) Routing Elements	10	
(3) Error Detection	10	
Total:	**33**	

Question 1: Decoders [13 pts]

We are building a **binary-to-flag semaphore decoder circuit**. A flag semaphore is a textual encoding using the positions of two flags. The *digit* semaphores are shown below on the left. We will use the radial arrangement of 8 LEDs, which are lit/black when the corresponding signal S_i is high/1, to display the flag positions (example below shows "3"). B represents a digit encoded in binary.





radial LEDs

(A) Complete the truth table. [4 pt]

(B) In the space below, solve for the minimal logical expression for $\mathbf{S_4}$. [5 pt]

B_3	B_2	B_1	B_0	S_4	S_7
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	**0**	
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**	

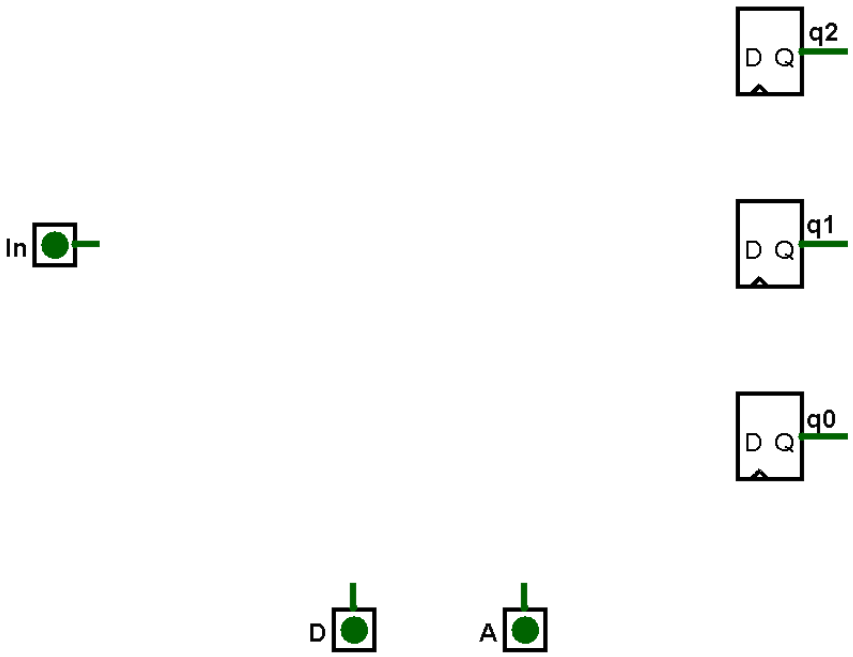
(C) The Don't Cares will be resolved to 0's and 1's in implementation, which can lead to confusing outputs. We can handle these inputs "gracefully" by either (1) changing these all to 0's or (2) adding an extra "Valid" output. Name a benefit of each approach over the other. [4 pt]

Option 1 benefit:
Option 2 benefit:

Question 2: Routing Elements [10 pts]

We are creating a sequential circuit with 1-bit inputs `A` (action), `D` (direction), `In` (input) and n-bit output `Q`. The circuit will either shift (`A=0`, filling vacant bit with `In`) or rotate (`A=1`, filling vacant bit with the bit that "falls off") each cycle. `D=0` indicates to the *right* (toward less significant bit) and `D=1` indicates to *left* (toward more significant bit).

(A) Draw the circuit diagram below using *logic gates* and *routing elements* discussed in class. Assume the clock inputs are connected properly for you. You may use multiple copies of a signal name (*e.g.*, `q2`, `q1`, `q0`), which are assumed connected to the same net/wire. [5 pt]



(B) In the Verilog test bench below, fill in the blanks to indicate how the state of our bidirectional shifter/rotator updates. [5 pts]

```
initial begin                                  // state: q2q1q0
   {In, D, A} <= 3'd0;                         // state:  011
   @(posedge clk);  A  <= 1;                   // state: _____
   @(posedge clk);  D  <= 1;                   // state: _____
   @(posedge clk);  In <= 1;                   // state: _____
   @(posedge clk);  A  <= 0;                   // state: _____
   @(posedge clk);  $stop();                   // state: _____
end
```

Question 3: Error Detection [10 pts]

In computing, an *even parity bit* is a bit added to the end of a string of bits to ensure that the parity of the overall group (*i.e.*, the string of bits plus the parity bit) is even (*i.e.*, there are an even number of `1`'s). Parity bits can be used in various forms of *error checking*.

Examples: For `0b111`, the even parity bit would be **`1`** so that `0b1111` has four `1`'s.
For `0b1010`, the even parity bit would be **`0`** so that `0b10100` has two `1`'s.

(A) (Circle one) Which type of gate below will be the most helpful in computing parity? [1 pt]

AND NAND NOR OR XNOR XOR

(B) Implement a 3-bit even parity bit **generator** (*i.e.*, compute the even parity bit for a 3-bit input string to complete a 4-bit group). Use only the *2-input variant of your Part A choice*. [3 pt]

(C) **Hamming double error detection** generates a specific code word from data bits that allows us to detect errors in the transmission or storage of the code word. For 4 data bits ($d_3d_2d_1d_0$), we generate the code word by adding 4 even parity bits (p_i) for these groups:

Bit position:		7	6	5	4	3	2	1	0
Code word:		p_1	p_2	d_3	p_4	d_2	d_1	d_0	p_8
Parity bit groups:	p_1	✓		✓		✓		✓	
	p_2		✓	✓			✓	✓	
	p_4				✓	✓	✓	✓	
	p_8	✓	✓	✓	✓	✓	✓	✓	✓

Complete the circuit below that creates the 8-bit code word. You can use 2-input gates, constants, and any number of the logic block from Part B.
Write `d3`, `d2`, `d1`, `d0` wherever needed. [6 pts]