CSE 369 QUIZ 3

Name: _Molly_Model_____

Student ID
Number: __1234567_____

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	13
(2) Routing Elements	10	10
(3) Error Detection	10	10
Total:	33	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.

1 2 3 4 5 6 7 8 9 0

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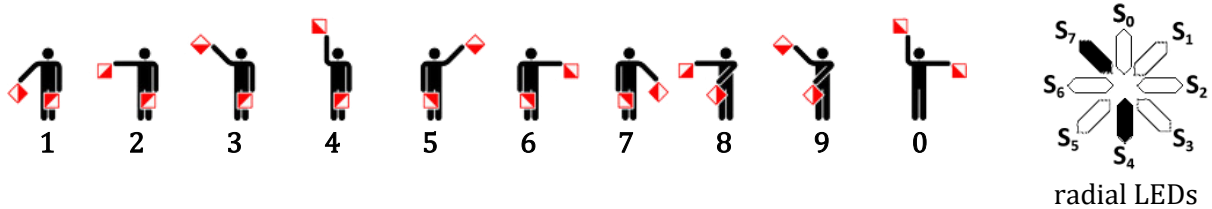
(A) Complete the truth table. [4 pt]

(B) In the space below, solve for the minimal logical expression for S_4. [5 pt]

$$S_4 = B_2 + B_1 + \overline{B_3}\,B_0$$

B_3	B_2	B_1	B_0	S_4	S_7
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	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: (examples, not exhaustive)
- No need for hardware for extra output signal.
- Unambiguous output because LEDs are off for unexpected inputs.

Option 2 benefit: (examples, not exhaustive)
- Likely (not guaranteed) simpler circuitry for S_i signals because Don't Cares can be specifically chosen.
- Valid signal better/more easily communicates in/validity to user instead of having to interpret S_i.
- Allows the use of the "all off" state for other purposes (instead of stand-in for "invalid").

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]

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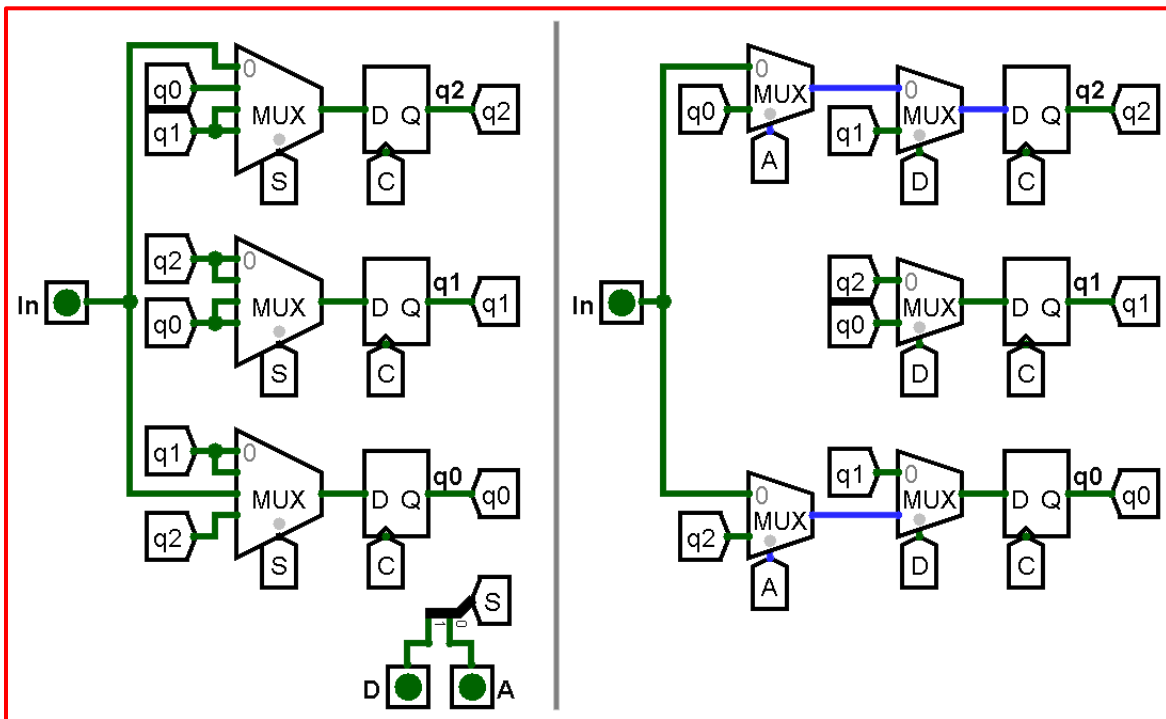
(B) In the Verilog test bench below, fill in the blanks to indicate how the state of our bidirectional shifter/rotator updates. [5 pts]

```
initial begin                                // state: q₂q₁q₀
   {In, D, A} <= 3'd0;                        // state:  011
   @(posedge clk);  A  <= 1;    // state: 001 (>> in 0)
   @(posedge clk);  D  <= 1;    // state: 100 (rot >>)
   @(posedge clk);  In <= 1;    // state: 001 (rot <<)
   @(posedge clk);  A  <= 0;    // state: 010 (rot <<)
   @(posedge clk);  $stop();    // state: 101 (<< in 1)
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]
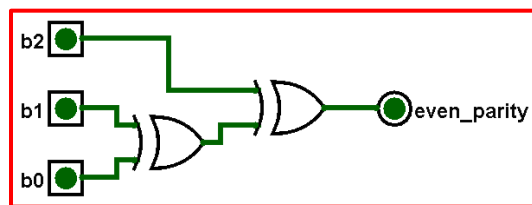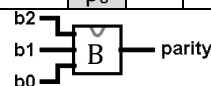

Any valid variation
on connections
accepted, such as
$(b2 \oplus b1) \oplus b0$.

(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:

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]

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:		7	6	5	4	3	2	1	0
	p_1	✓		✓		✓		✓	
	p_2		✓	✓			✓	✓	
	p_4				✓	✓	✓	✓	
	p_8	✓	✓	✓	✓	✓	✓	✓	✓

The shown solution relies on the invariant that parity group 1 will have even parity and therefore only computes the parity of the remaining bits (positions 6, 4, 2) for p8. Any valid computation of p8 (*e.g.*, XOR all 7 bits, using a different parity group invariant) accepted.