CSC 258H1 F 2016 Midterm Test
Duration — 1 hour and 50 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|_|

UTORid: |_|_|_|_|_|_|_|_|_|_|_|_|_|

Last Name: _____ **First Name:** _____

Circle the **lecture section** you are enrolled in.

Instructor	**Lecture Section**
Frank Plavec	L0101 (MWF 11-12)
Myrto Papadopoulou	L5101 (T 6-9)

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)

Good Luck!

This midterm is double-sided, and consists of 9 questions on 23 pages (including this one). When you receive the signal to start, please make sure that you have all pages.

- If you use any space for rough work, indicate clearly what you want marked.
- Do not remove any pages from the exam booklet.
- You may use a pencil; however, work written in pencil will not be considered for remarking.

1: _____/ 24
2: _____/ 20
3: _____/ 6
4: _____/ 17
5: _____/ 18
6: _____/ 20
7: _____/ 20
8: _____/ 12
9: _____/ 24

TOTAL: _____/161

Question 1. [24 MARKS]

Answer the following questions in the space provided. When providing a written answer, please write **as clearly and legibly as possible**. Marks will not be awarded for unreadable answers.

(a) Complete the following sentences about a 6-to-1 multiplexer whose data inputs are each 1-bit wide:

- A 6-to-1 multiplexer requires __3__ 1-bit select (selection) inputs.
- The output of this 6-to-1 multiplexer will be a don't care for how many unique value combinations of all the select inputs? __2__
- The complete truth table for this 6-to-1 multiplexer would contain __512 ($2^{(6+3)}$)__ rows.

(b) Assume that all Boolean functions listed in this question have two 1-bit inputs, A and B, and that A appears in the first column of the truth-table while B in the second. Write the functions below in Sum-Of-Minterms form. Use the m_i notation to specify minterm i.

- F = AB + AB' in Sum-Of-Minterms form is F = __m3 + m2__.
- F = B in Sum-Of-Minterms form is F = __m1 + m3__.

(c) Answer the following questions about different numerical representations.

- The 3-bit binary number 101 corresponds to decimal number __5__ if unsigned or to decimal number __-3__ if in 2's complement notation.
- Adding the 2's complement of a number X to the 2's complement of the 2's complement of that number X results in __0__. You can express the result as a function of X.
- Adding the two **hexadecimal** numbers A and 5 results in **hexadecimal** number __f__. The minimum number of binary digits (bits) needed to represent this result is __4__.

(d) You are given the following set of ModelSim commands:

```
# Both a and b inputs are 1-bit each.
force {a} 0 0ns, 1 10ns -repeat 40ns
force {b} 1 0ns, 0 20ns -repeat 40ns
run 60ns
```

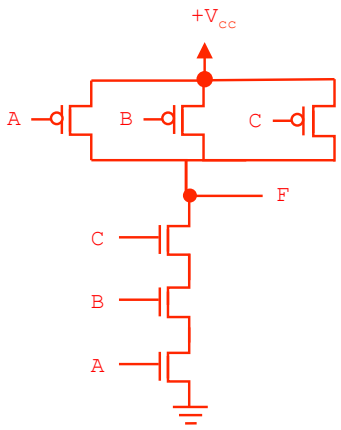
- Complete the next table:

Time in ns	Value of Input a	Value of Input b
5	0	1
15	1	1
30	1	0
45	0	1
55	1	1

- If a and b are inputs to a 2-input XOR gate, write below **all** time-intervals up to time 60ns during which the output F of this circuit will be logic-1. Write each interval in T_{start} - T_{end} format, where both T_{start} and T_{end} are in ns. The output should not change within a given interval. You can assume a propagation delay of zero.

$0 - 10, 20 - 50$

(e) In the empty space below implement a 3-input NAND gate using P-type and N-type transistors. Name the output of that 3-input gate F and its inputs A, B, and C. Any correct implementation will receive partial marks, but only implementations with the smallest number of transistors will receive full marks.



(f) You are given the following Verilog snippet:

```
wire b, c1, f;
wire [2:0] a;

assign c1 = a[0] & a[1] & a[2];
assign f = c1 & b;
```

You can simplify this Verilog code by substituting the last two assign statements with the one below. Use one character/symbol for each underscore below.

assign f = & {b , a};

(g) You created Verilog code for a multiplexer design. You compile the code in Quartus Prime, and observe the following warning message:

Warning (10240): Verilog HDL Always Construct warning at mux.v(27): inferring latch(es) for variable "f", which holds its previous value in one or more paths through the always construct

Clearly explain (i) what the likely cause of this message is and (ii) what needs to be done to address it. Your explanation should **not** be longer than three full sentences. **Vague answers will not be awarded any marks**.

Likely cause of this message is missing *else* in an *if* statement, or a missing case in a *case* statement inside of a combinational (*always @(*)*) always block.
To fix the issue, we would have to add the missing case or provide a default value.

Question 2. [20 MARKS]

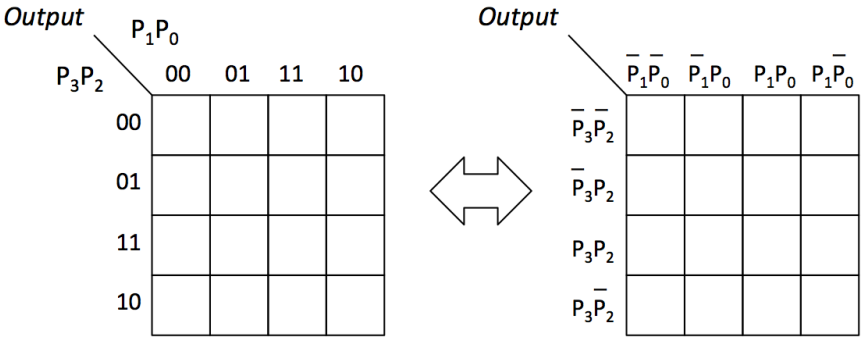
You were hired by CoolBoardGames Inc. as their chief scientist in charge of digital logic design for their upcoming board game VoterMatic. VoterMatic is a four-player game, where each player secretly casts a vote to accept or reject a proposal. If a player wants to accept the proposal they vote 1, and if they want to reject it they vote 0. Players do not know how other players voted, but they get hints about the overall voting process through the following three indicators:

- U: Indicates whether the vote outcome was unanimous. A vote is considered unanimous if all players voted exactly the same (**regardless of whether the proposal was accepted or rejected**).
- A: Indicates whether the proposal was accepted. For the proposal to be accepted, the majority of players have to vote 1.
- R: Indicates whether the proposal was rejected. If the majority of players voted 0, or the number of players that voted 0 is the same as the number of players that voted 1, the proposal is rejected.

To give the new game that extra cool factor, you wish to include a little logic device that will display these hints to players using LEDs. Each player (P_0, P_1, P_2, P_3) will control a switch that they will set to 0 if they wish to reject the proposal, or to 1 if they wish to accept it. During the vote, LEDs are covered so players cannot see them. After all players have cast their vote, LEDs are revealed showing the three voting indicators (U, A, R) described above. That is, U will be lit up (i.e., produce output 1) if the vote was unanimous, A will be lit up if the proposal was accepted, and R will be lit up if the proposal was rejected.

On the following pages, use Karnaugh maps to derive Boolean expressions for the logic that will control the LEDs. Note that your K-maps, your groupings, as well as the Boolean expressions you derive will ALL be marked.

Notation: As a reminder, the following two notations of K-Maps, both used in lectures, are equivalent.



Part (a) [10 MARKS]

Use Karnaugh maps below to produce the minimal sum of products **OR** the minimal product of sums implementation of Boolean functions U, A and R. After you fill in the corresponding K-maps, show your groupings clearly. Write the Boolean expression corresponding to your groupings below each Karnaugh map.

U	P_1P_0 00	01	11	10
P_3P_2 00	1	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0

$U = P_3P_2P_1P_0 + \bar{P}_3\bar{P}_2\bar{P}_1\bar{P}_0$

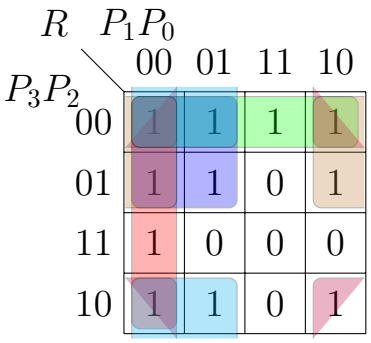
A	P_1P_0 00	01	11	10
P_3P_2 00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

$A = P_3P_2P_0 + P_3P_2P_1 + P_2P_1P_0 + P_3P_1P_0$

R	P_1P_0 00	01	11	10
P_3P_2 00	1	1	1	1
01	1	1	0	1
11	1	0	0	0
10	1	1	0	1

$R = (\bar{P}_3 + \bar{P}_2 + \bar{P}_0)(\bar{P}_3 + \bar{P}_2 + \bar{P}_1)(\bar{P}_2 + \bar{P}_1 + \bar{P}_0)(\bar{P}_3 + \bar{P}_1 + \bar{P}_0)$

Alternative sum of products:



$R = \bar{P}_3\bar{P}_2 + \bar{P}_1\bar{P}_0 + \bar{P}_3\bar{P}_1 + \bar{P}_2\bar{P}_0 + \bar{P}_3\bar{P}_0 + \bar{P}_2\bar{P}_1$

Part (b) [10 MARKS]

The company is also investigating a different version of the game. In this modified version of the game there are still four players (P_0, P_1, P_2, P_3), but players P_0 and P_1 need to collaborate and agree on the vote that they will cast. That is, players P_0 and P_1 can choose to vote either 0 or 1, but once they decide, they must both vote the same. Derive Boolean expressions for this modified version of the game.

Use Karnaugh maps below to produce the minimal sum of products **OR** the minimal product of sums implementation of Boolean functions U, A and R for the modified version of the game. After you fill in the corresponding K-maps, show your groupings clearly. Write the Boolean expression corresponding to your groupings below each Karnaugh map.

U P_3P_2 \ P_1P_0	00	01	11	10
00	1	X	0	X
01	0	X	0	X
11	0	X	1	X
10	0	X	0	X

A P_3P_2 \ P_1P_0	00	01	11	10
00	0	X	0	X
01	0	X	1	X
11	0	X	1	X
10	0	X	1	X

$U = \bar{P}_3\bar{P}_2\bar{P}_1 + P_3P_2P_0$

$A = P_2P_0 + P_3P_0$

In any of these equations P_0 and P_1 are interchangeable depending on the chosen groupings of course.

R P_3P_2 \ P_1P_0	00	01	11	10
00	1	X	1	X
01	1	X	0	X
11	1	X	0	X
10	1	X	0	X

$R = (\bar{P}_2 + \bar{P}_0)(\bar{P}_3 + \bar{P}_0)$

Alternative sum of products:

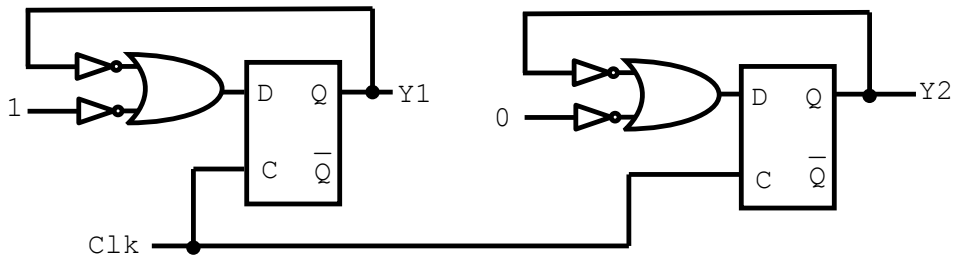
R P_3P_2 \ P_1P_0	00	01	11	10
00	1	X	1	X
01	1	X	0	X
11	1	X	0	X
10	1	X	0	X

$R = \bar{P}_1 + \bar{P}_3\bar{P}_2$

Again, in any of these equations P_0 and P_1 are interchangeable, depending on the selected K-map groupings.

Question 3. [6 MARKS]

The following circuit contains two gated D **latches**. Assuming the outputs Y1 and Y2 were both initially logic-0, answer the following questions in 1 to 2 sentences the most:



- What will the behaviour of output Y1 be when input Clk is logic-1?

  Y1 will oscillate between 1 and 0 while Clk is high

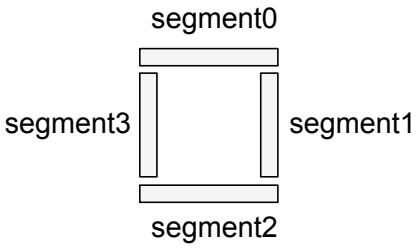
  as the logic circuit in front of D is a NOT gate.

- What will the behaviour of output Y2 be when input Clk is logic-1?

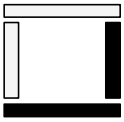
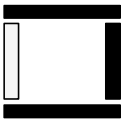
  Y2 will always be 1.

Question 4. [17 MARKS]

You are building a new symbolic language for fun and want to implement a decoder circuit for the special 4-segment display shown below.



Given two inputs C1 and C0, the 4-segment display should show the symbols shown below. A black segment means the segment is illuminated, while a white segment means it is not (i.e., it is turned off). The manual says that the segments in this display are **active low**, like in the 7-segment hex display you used in the labs.

C1	C0	Segments
0	0	
0	1	
1	0	
1	1	

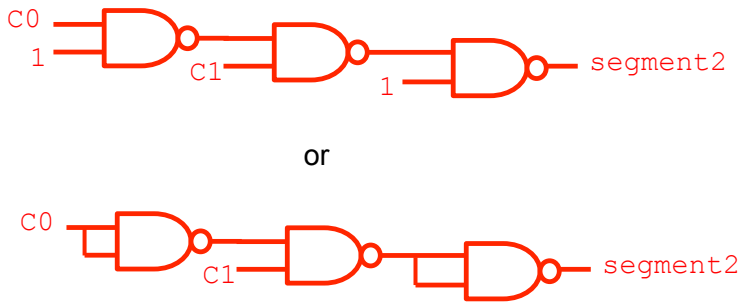
Part (a) [8 MARKS]

Write the Boolean expressions for this decoder circuit below. Make sure to simplify them, if possible!

- segment0 = C1'C0
- segment1 = 0
- segment2 = C1C0'
- segment3 = C0

Part (b) [6 MARKS]

Draw the gate-level circuit for `segment2` using **only 2-input NAND gates**. Using any other gate types will earn you **no** marks for this part.



or

Part (c) [3 MARKS]

While testing your circuit you realize that the specifications of the 4-segment display were wrong and each segment is active-high instead of active-low. Unfortunately you have already implemented your circuit on the breadboard. What would be the fastest way to fix your design **without removing any of the existing gates in the circuit**? In other words, your new solution has to reuse the existing circuit in full.

(i) Specify what kind of gates, and how many you would need to use to fix this issue?

4 NOT gates

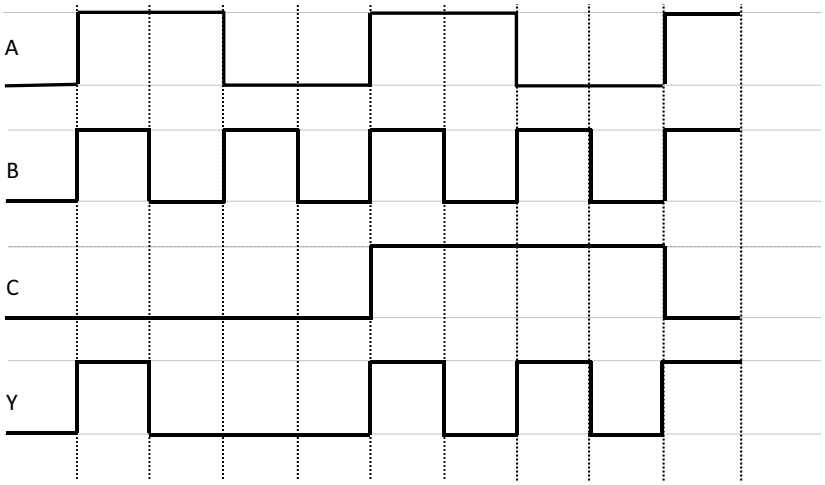
(ii) Briefly explain, in one or two sentences, how you will use the gates you chose to fix your circuit.

I will connect each segment output of the current circuit to a NOT gate.
I could use 3 NOT gates instead of 4 if I connect segment1 directly to Vcc.

Question 5. [18 MARKS]

You are given the following two waveforms each describing the behaviour of a digital logic circuit with inputs **A**, **B**, and **C**, and output **Y**. For each waveform answer whether the behaviour described by the waveform can be implemented using pure combinational logic. Depending on your answer, you will be asked to either explain why or derive the Boolean expression for the output `Y` and draw the circuit.

Part (a) [9 MARKS]

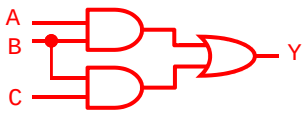


Can the behaviour described by the above waveform be implemented using pure combinational logic?
Circle the correct answer: **Yes** No

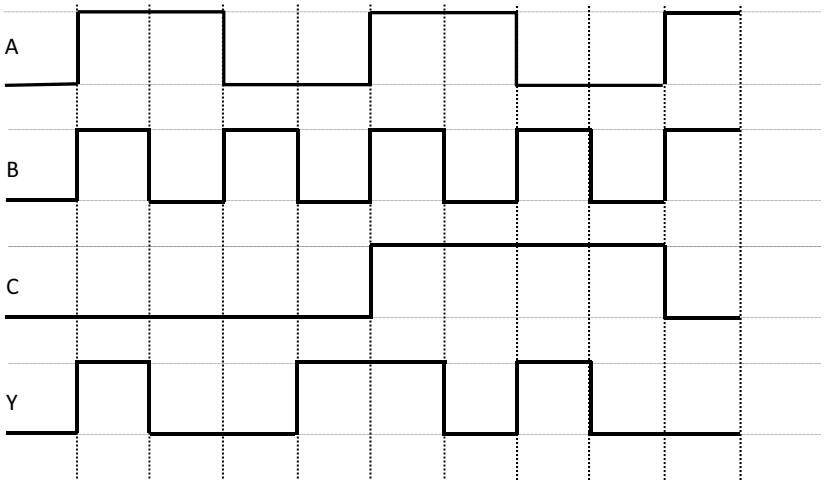
- If you answered no, clearly explain why not in one or two full sentences. **Vague answers will not be awarded any marks**.

- If you answered yes:
  - Derive the Boolean expression that implements the behaviour described by the waveform.

    $Y = AB + BC$.
  - Draw the circuit (gate-level) below. There are no constraints in the logic gates you can use.



Part (b) [9 MARKS]



Can the behaviour described by the above waveform be implemented using pure combinational logic?
Circle the correct answer: Yes **No**

- If you answered no, clearly explain why not in one or two full sentences. **Vague answers will not be awarded any marks**.

  The reason is that for the same combinations of input values (e.g., 000 or 110), there are different output values, which is not possible to achieve using just logic gates.
  Clearly some form of memory (latch, or FF) is required to implement this circuit.

- If you answered yes:
  - Derive the Boolean expression that implements the behaviour described by the waveform.

    Y = _____

  - Draw the circuit (gate-level) below. There are no constraints in the logic gates you can use.

Question 6. [20 MARKS]

You are to implement a comparator circuit that can compare two 4-bit **signed** numbers represented using **2's complement** notation. You can assume that you are already given a module that implements a 4-bit ripple-carry adder whose module and port declarations are specified below. Note that this module is available to you and you **do not need to implement it.**

```
module adder (A, B, cin, S, cout);
    input [3:0] A;
    input [3:0] B;
    input cin;
    output [3:0] S;
    output cout;
```

Assuming the above adder unit is available to you, design a circuit that can implement comparison of two 4-bit **signed** numbers. Your circuit **must** use the adder module specified above, and can use any additional gates as necessary. You can use NOT gates, AND gates, and OR gates with arbitrary number of inputs. Only circuits with minimal number of additional gates will be awarded full marks. The circuit takes two 4-bit **signed** numbers X and Y as inputs, and produces three outputs: eq, gr, ls, whose meaning is as follows:

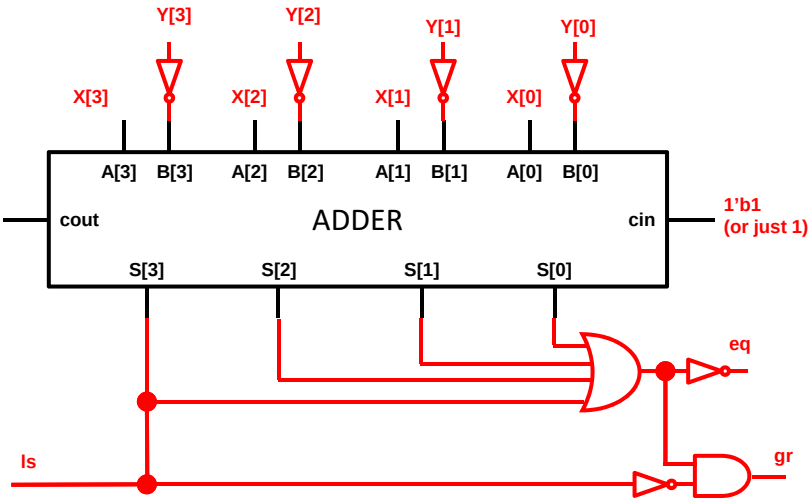
- eq: Output is set to 1 only if signed numbers X and Y compare equal.
- gr: Output is set to 1 only if signed number X is arithmetically greater than signed number Y. As an example, -1 is greater than -3.
- ls: Output is set to 1 only if signed number X is arithmetically less than signed number Y.

Although the inputs are 4-bit **signed** numbers represented using **2's complement** notation, you are guaranteed that inputs X and Y will never be outside the [-4, +3] range (inclusive). That is, the lowest negative number your circuit has to handle is -4, and the highest positive number your circuit has to handle is 3.

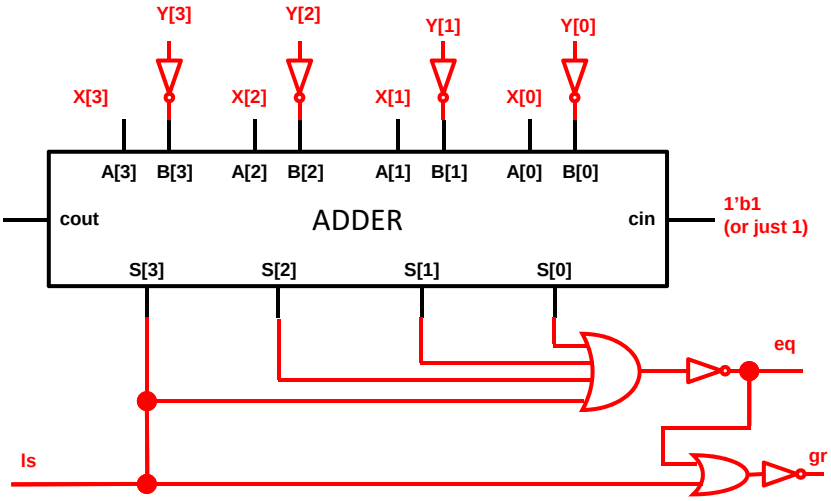
Complete Parts (a) and (b) of this question in the subsequent pages. Part (a) asks you to draw the schematic of this circuit, while Part (b) asks you to write the Verilog code describing your design.

Part (a) [12 MARKS]

Draw a schematic of the circuit that can implement comparison of two 4-bit **signed** numbers, as described above. Your schematic **must** use the adder module specified above. Only circuits with minimal number of additional gates will be awarded full marks. You can use NOT gates, AND gates, and OR gates with arbitrary number of inputs. You are **not** required to draw the inside of the adder circuit.



OR



[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Part (b) [8 MARKS]

Write Verilog code implementing this circuit. Your code **must** use the adder module specified above. Make sure your Verilog code is syntactically correct as much as possible. Marks will not be deducted for minor mistakes, such as a **single** missing semicolon, but marks will be deducted if your code consistently makes the same mistake, or for any mistakes impacting functionality. Marks **will** be deducted if there is any ambiguity in what you wrote, so make sure to write legibly.

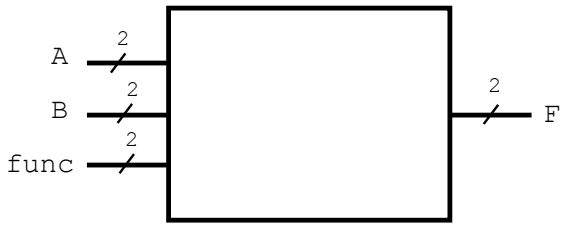
```
module comparator (X, Y, eq, gr, ls);
input [3:0] X;
input [3:0] Y;
output eq;
output gr;
output ls;

wire [3:0] sub;

adder a1 (.A(X),
          .B(~Y),
          .cin(1'b1),
          .S(sub)
         );
assign eq = ~|sub; // Also accepted: assign eq = (sub == 4'h0);, or anything similar
assign gr = ~sub[3] & ~eq; // Also accepted: assign gr = ~(sub[3] | eq);
assign ls = sub[3]; // For ls, one could also include ~eq but it is not needed.
endmodule
```

Question 7. [20 MARKS]

Part (a) [14 MARKS]

Complete the following Verilog module to implement the design shown in the schematic below. This design is a variant of an ALU module that will perform different operations on its two **unsigned** operands `A` and `B` based on the value of the input `func`. The operations are described below.

- If `func` is 0: The most-significant bit of the output should be 1 if only one of the most significant bits of A and B is 1. The least significant bit of the output should be 0 if all bits of input B are 0.
- If `func` is 1: The output should contain all the bits of A but in the reverse order; the most significant bit of A should be the least significant of the output, etc.
- If `func` is 2: The output should be the one's complement of input `A`.
- If `func` is 3: The ALU should perform addition between `A` and `B`.




```
module my_ALU(A, B, func, F );

    input [1:0] A;

    input [1:0] B;

    input [1:0] func;

    output reg [1:0] F;

    always@(*)
        begin
            case(func)

                2'b00: F = {A[1]^B[1], |B};

                2'b01: F = {A[0], A[1]};

                2'b10: F = ~A;

                2'b11: F = A + B;

            endcase
        end
endmodule
```

Part (b) [6 MARKS]

It is possible for the output `F` of the aforementioned design to be incorrect for a specific `func` operation. Assuming that `A` and `B` can each take values in the [0, 2] range, answer the following questions:

- For which `func` operation could the output `F` be incorrect? **Addition, $func = 3$**
- Which input combination(s) of `A` and `B` could result in an incorrect output for the previously identified `func` operation?

  When both A = 2 and B is 2.

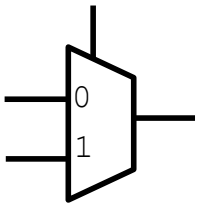
- Assume you can add a 1-bit output `error_led` to the `my_ALU` module that will be turned-on (logic-1) only when the error mentioned above for the specific `func` operation occurs, and will be turned-off in all other cases. Write a Boolean expression for this `error_led` function. You do **not** need to minimize (simplify) this Boolean expression.

  ***error_led* = A[1]B[1]A[0]'B[0]'func[1]func[0]**

  Doing A[1]B[1]func[1]func[0] is fine too given the aforementioned constraints.

Question 8. [12 MARKS]

You are asked to implement a **half-adder** module with inputs X and Y and outputs S and C using **only** 2-to-1 multiplexers (schematic shown below) and **nothing else**. You are welcome to connect any input(s) of the multiplexer(s) to logic-0 or logic-1 as needed, in addition to X and Y. All correct answers will receive partial marks but only implementations with the smallest number of multiplexers will receive full marks.

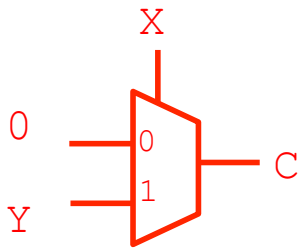


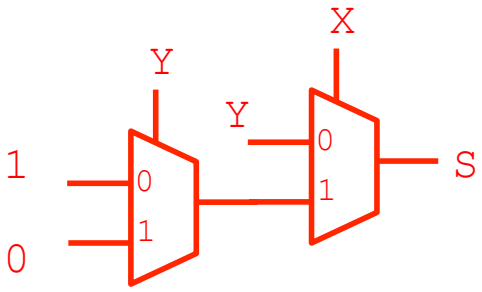
First write the Boolean expressions for S and C.

$S = X'Y + XY'$ $C = XY$

Then draw the circuit below using only 2-to-1 multiplexers, as mentioned earlier:

Here's one possible solution:

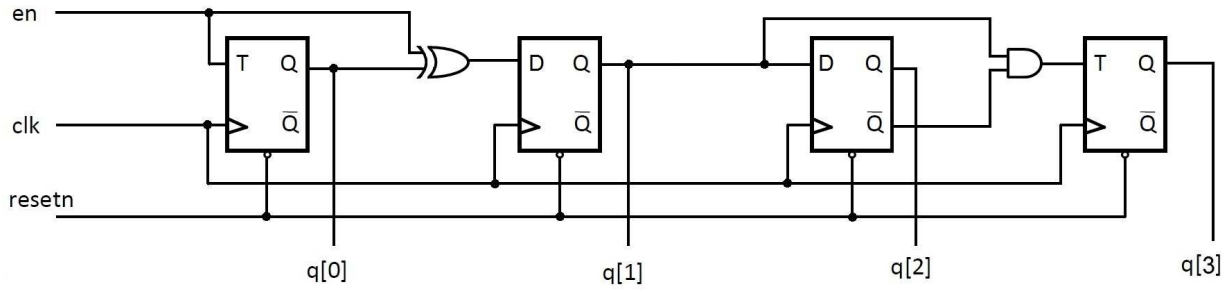




Note that X and Y are interchangeable, as long as for a given schematic you substitute ALL instances of X with Y and vice versa.

Question 9. [24 MARKS]

You are given the following counter design. **Pay particular attention to the fact that not all flip-flops are of the same type**; some of them are T flip-flops, while others are D flip-flops. The *resetn* signal is an **asynchronous** active-low reset signal.



Part (a) [14 MARKS]

The following table shows the initial state of counter outputs $q[0]$ through $q[3]$ at time 0. Before time 0, *resetn* is active (i.e., 0), and *en* is inactive (i.e., 0). At time 0, *resetn* is deactivated and *en* is activated, as shown in the table. Fill all the empty rows of the table with the state of the outputs $q[0]$ through $q[3]$ in subsequent clock cycles to determine the sequence that this counter is counting. By doing so, you should be able to conclude how all future counts would proceed if the *en* signal was held active indefinitely.

Inputs		Outputs			
resetn	*en*	$q[3]$	$q[2]$	$q[1]$	$q[0]$
1	1	0	0	0	0
1	1	0	0	1	1
1	1	1	1	0	0
1	1	1	0	1	1
1	1	0	1	0	0
1	1	0	0	1	1
1	1				
1	1				

Part (b) [4 MARKS]

Assuming $q[0]$ is the least significant bit, and $q[3]$ is the most significant bit, what is the sequence of unsigned numbers (**in decimal**) that this counter is counting?

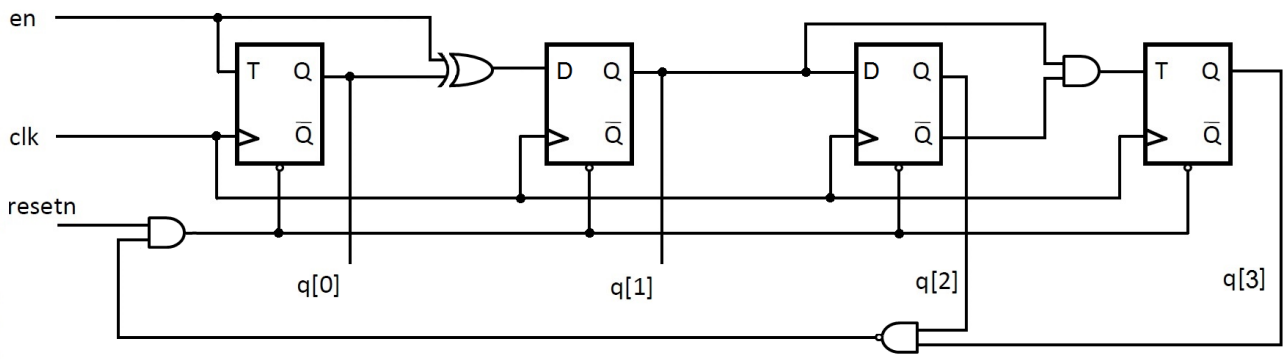
The sequence is 3, 12, 11, 4, 3,...

Also acceptable answer is 0, 3, 12, 11, 4, 3,...

However, 0, 3, 12, 11, 4, **0**, 3,... is not an acceptable answer

Part (c) [6 MARKS]

The circuit is then modified to include additional logic that drives the **asynchronous** *resetn* of the flip-flops. The rest of the circuit driving T and D flip-flops is **not** modified, as shown in the figure below.



Answer the following question: What is the new sequence of unsigned numbers (in decimal) that this counter is counting?

The sequence is 0, 3, 0, 3,...

Also acceptable answer is 0, 3, 12, 0, 3, 12,... since 12 appears on the output for a very brief period of time before asynchronous reset resets all flip-flops.