1	/18
2	/17
3	/18
4	/19
5	/18
6	/10

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Fall 2025

Quiz #1

Name **SOLUTIONS**	*Athena login name*	*Score*

Recitation section

□ WF 10, 34-301 (Jan)	□ WF 2, 34-302 (Varun)	□ WF 12, 34-303 (Nathan)
□ WF 11, 34-301 (Jan)	□ WF 3, 34-302 (Varun)	□ WF 1, 34-303 (Nathan)
□ WF 12, 34-302 (Abdullah)	□ WF 10, 34-302 (Christina)	□ WF 2, 34-303 (Grace)
□ WF 1, 34-302 (Abdullah)	□ WF 11, 34-302 (Christina)	□ WF 3, 34-303 (Grace)
		□ opt-out

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. Show your work for potential partial credit. You can use the extra white space and the back of each page for scratch work.

Problem 1. Digital Abstraction (18 points)

Suppose we define all signaling thresholds in our digital system to be relative to the supply voltage V_{DD}:

$V_{OL} = 0.1V_{DD}$
$V_{IL} = 0.5V_{DD}$
$V_{IH} = 0.6V_{DD}$
$V_{OH} = 0.8V_{DD}$

Suppose we have some valid digital **inverter** F with supply voltage $V_{DD,F} = 10\text{V}$ and propagation delay of 50ns.

(A) (2 points)

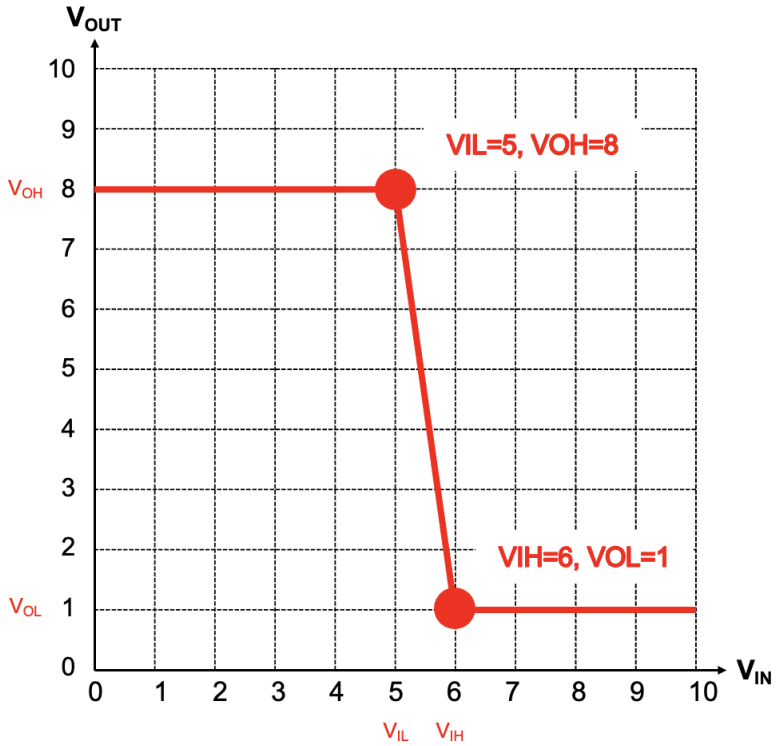
Given the above specification for inverter F, determine the noise immunity of the device.

$V_{OL} = 1\text{V}, V_{IL} = 5\text{V}, V_{IH} = 6\text{V}, V_{OH} = 8\text{V}$.
Low noise margin = $V_{IL} - V_{OL} = 4\text{V}$, High noise margin = $V_{OH} - V_{IH} = 2\text{V}$.

Noise Immunity (V): 2V

(B) (3 points)

Draw one potential voltage transfer curve for F. **Please clearly label all signaling thresholds.**



(C) (3 points)

We apply constant input voltage V_{in} to F for 30ns starting at time $t = 0$ns and measure that $V_{out} = 9$V at $t = 25$ns. What can we conclude about V_{in}? Circle the correct answer and provide an explanation.

V_{in} is (select one): $V_{in} < 1$V $V_{in} > 5$V $V_{in} < 6$V $V_{in} > 8$V None of the above

Explanation: The propagation delay is 50ns; at $t = 25$ns the output may not yet reflect the input, so no conclusion.

(D) (3 points)

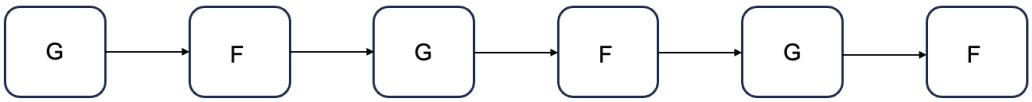
We apply constant input voltage V_{in} to F for 60ns starting at time $t = 0$ns and measure that $V_{out} = 0.5$V at $t = 50$ns. What can we conclude about V_{in}? Circle the correct answer and provide an explanation.

V_{in} is (select one): $V_{in} < 1$V $V_{in} > 5$V $V_{in} < 6$V $V_{in} > 8$V None of the above

Explanation: By $t = 50$ns the output reflects the input. Since F is an inverter and we observed a low output (0.5V), the input must not have been a valid low input, so $V_{in} > V_{IH} = 5$V. An output of 0.5V could have resulted from either a valid high input or an invalid input.

(E) (4 points)

Now consider device G with the same signaling thresholds relative to its supply voltage $V_{DD,G}$. In the circuit below, under what range of supply voltages $V_{DD,G}$ will the system work correctly?

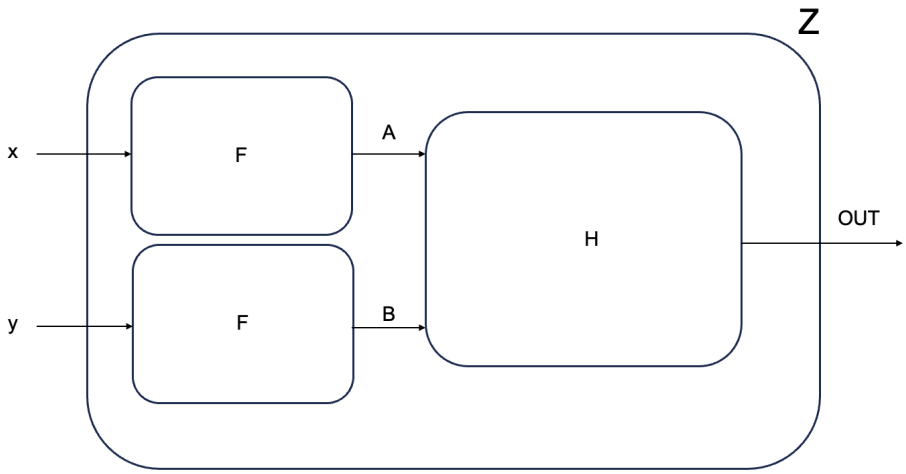


For G→F: $0.1V_{DD,G} \leq 5$V and $0.8V_{DD,G} \geq 6$V.
For F→G: $1\text{V} \leq 0.5V_{DD,G}$ and $8\text{V} \geq 0.6V_{DD,G}$.
Combined: $7.5\text{V} \leq V_{DD,G} \leq 13.33\text{V}$.

Acceptable Range for $V_{DD,G}$: $7.5\text{V} \leq V_{DD,G} \leq 13.33\text{V}$

(F) (3 points)

Now consider the following circuit below.



Circuit H follows the voltage transfer characteristic given below.

$$V_{\text{out}} = \begin{cases} \textbf{8V} & \textbf{(max}(V_A, V_B) \geq \textbf{8V)} \\ \textbf{1V} & \textbf{(max}(V_A, V_B) \leq \textbf{1V)} \end{cases}$$

What Boolean expression does $H(A, B)$ implement? Express your answer in terms of A and B.

Boolean Expression for $H(A, B)$: $A + B$

What gate does Z implement? Express your answer in terms of X and Y.

Gate implemented by Z: NAND(X, Y)

Problem 2. Boolean Algebra (17 points)

(A) (5 points)

Consider the truth table given below:

a	*b*	*c*	*F*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

1. **What is the normal form for $F(a, b, c)$?**

   Normal form for F: $\underline{\overline{a}\overline{b}c + \overline{a}b\overline{c} + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c + ab\overline{c}}$

2. **Find two different minimal sum-of-products expressions for $F(a, b, c)$.**

   Minimal SOP form 1:
   $F = \overline{a}\overline{b}c + \overline{a}b\overline{c} + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c + ab\overline{c}$

   Next we combine the following terms:
   - $\overline{a}b\overline{c}$ with $\overline{a}bc$
   - $a\overline{b}\overline{c}$ with $ab\overline{c}$
   - $\overline{a}\overline{b}c$ with $a\overline{b}c$

   $= \overline{a}b(\overline{c} + c) + a\overline{c}(\overline{b} + b) + \overline{b}c(\overline{a} + a)$
   $= \overline{a}b + a\overline{c} + \overline{b}c$
   First minimal sum-of-products for F: $\underline{\overline{a}b + a\overline{c} + \overline{b}c}$

   Minimal SOP form 2:
   $F = \overline{a}\overline{b}c + \overline{a}b\overline{c} + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c + ab\overline{c}$

   Next we combine the following terms:
   - $\overline{a}\overline{b}c$ with $\overline{a}bc$
   - $\overline{a}b\overline{c}$ with $ab\overline{c}$
   - $a\overline{b}\overline{c}$ with $a\overline{b}c$

   $= \overline{a}c(\overline{b} + b) + b\overline{c}(\overline{a} + a) + a\overline{b}(\overline{c} + c)$
   $= \overline{a}c + b\overline{c} + a\overline{b}$
   Second minimal sum-of-products for F: $\underline{\overline{a}c + b\overline{c} + a\overline{b}}$

(B) (3 points)

Can you construct a circuit for any possible Boolean function using just XOR and AND gates? **If yes, prove it. If no, provide a counterexample**.

To construct a circuit for any possible Boolean function using just XOR and AND gates, we need to show that we can implement AND, NOT and OR operations using these two gates.

We can perform AND operation directly using the AND gate.

Implementing NOT operation using XOR gate:
$A \oplus B = A\overline{B} + \overline{A}B$
If we set $B = 1$, then $A \oplus 1 = \overline{A}$
Hence, we can perform NOT operation using XOR gate by setting $B = 1$.

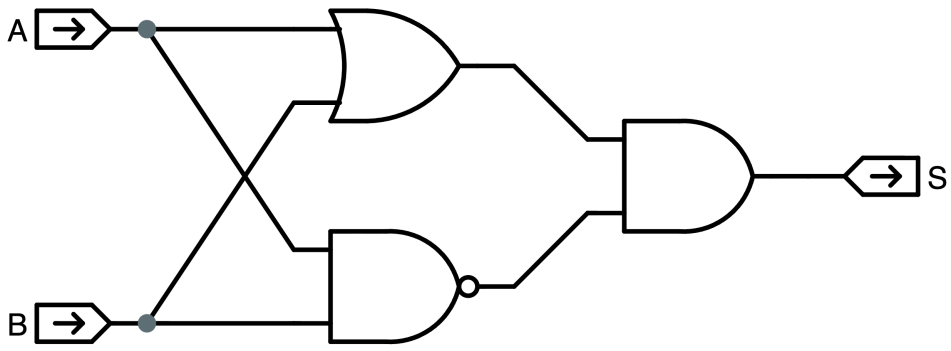
Implementing OR operation using XOR gate and AND gate:
Using De Morgan's laws, A + B = $\overline{\overline{A} \cdot \overline{B}}$
This helps us perform OR operation using just NOT and AND operations.
From previous results, we know that we can perform NOT operation using XOR gate by setting $B = 1$ and we can perform AND operation using the AND gate.
Hence, we can perform OR operation using XOR gate and AND gate.

Hence, we can construct a circuit for any possible Boolean function using just XOR and AND gates.

(C) (3 points)

Implement the function $G = A \oplus B = XOR(A, B)$ using one 2-input OR, one 2-input AND, and one 2-input NAND gate. **Using Boolean algebraic proprties, demonstrate that your circuit is equivalent in functionality to an XOR gate**.

$G = A \oplus B = A\overline{B} + \overline{A}B = A\overline{A} + A\overline{B} + \overline{A}B + B\overline{B} = (A + B)(\overline{A} + \overline{B}) = (A + B)(\overline{AB})$
The circuit figure is below:



(D) (6 points)

Determine whether each of the Boolean expressions below are satisfiable. If it is satisfiable, give an input assignment that makes the expression satisfiable. You must provide a valid value for each variable. If it is not satisfiable, show why it is not.

1. **Expression** $(\overline{a} + b)(\overline{b} + c)(\overline{c} + a)(\overline{c} + d + e)(\overline{d} + f)(\overline{f} + \overline{a} + \overline{b})$

   Satisfiable.
   Input assignments that satisfy the expression are below:
   Input assignment 1: $a = 0, b = 0, c = 0, d = 0, e = 0, f = 0$
   Input assignment 2: $a = 1, b = 1, c = 1, d = 0, e = 1, f = 0$

2. **Expression** $(\overline{a} + b)(\overline{b} + c)(\overline{c} + d)(\overline{d} + e)(a + \overline{e})(a \oplus c + f)(\overline{f})$

   Not satisfiable.
   For the expression to be satisfiable, $\overline{f}$ term must be 1 and hence f must be 0. Given that $f = 0$, the term $(a \oplus c + f)$ reduces to $(a \oplus c)$, which must be 1. Thus a and c must differ.
   Case 1: $a = 1$. Then $(\overline{a} + b)$ reduces to b, so $b = 1$. Next $(\overline{b} + c)$ reduces to c, so $c = 1$, contradicting $a \oplus c = 1$.
   Case 2: $a = 0$. Then $(a + \overline{e})$ reduces to $\overline{e}$ so $e = 0$; $(\overline{d} + e)$ reduces to $\overline{d}$ so $d = 0$; $(\overline{c} + d)$ reduces to $\overline{c}$ so $c = 0$, again contradicting $a \oplus c = 1$.
   Hence this expression is not satisfiable.

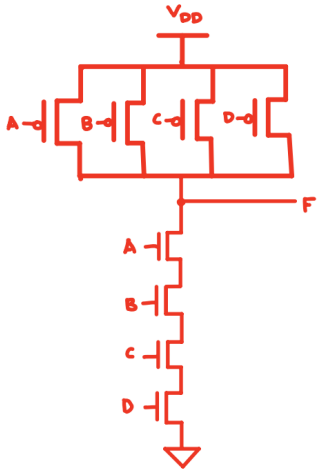
Problem 3. CMOS Logic (18 points)

(A) (12 points)

For each of the following four functions, determine if it can be implemented as a single CMOS gate. **If it can, draw the complete CMOS implementation using a minimal number of transistors. If not, describe why it cannot be implemented as a single CMOS gate.**

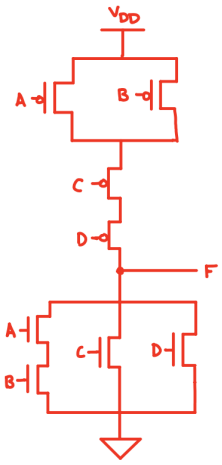
1. $F(A,B,C,D) = \overline{ABD} + \overline{C}$

The simplified boolean expression is $F(A,B,C,D) = \overline{A} + \overline{B} + \overline{C} + \overline{D}$. The CMOS gate is shown below:



2. $F(A,B,C,D) = \overline{AB + C} \cdot \overline{D}$

The simplified boolean expression is $F(A,B,C,D) = (\overline{A} + \overline{B}) \cdot \overline{C} \cdot \overline{D}$. The CMOS gate is shown below:

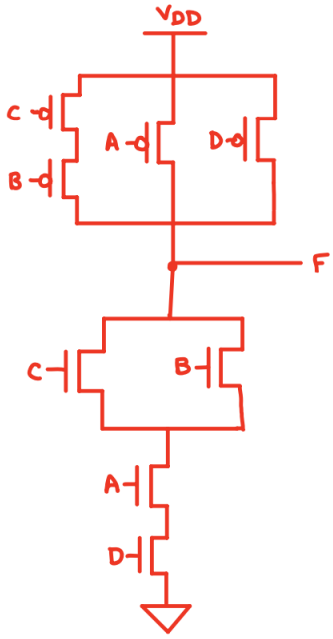


3. $F(A,B,C,D) = \overline{\overline{A} + D \cdot \overline{B + C}}$

The simplified boolean expression is $F(A,B,C,D) = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$. F is not inverting, so it can't be implemented as a single CMOS gate. To formally prove that F is not inverting, one can observe that, for example, F(0, 0, 0, 0) = 0, when every inverting function must output 1 when given all 0s.

4. $F(A,B,C,D) = \overline{D \cdot (AB + AC)}$

The simplified boolean expression is $F(A,B,C,D) = \overline{A} + \overline{D} + \overline{B} \cdot \overline{C}$. The CMOS gate is shown below:



(B) (6 points)

Given the following truth tables, find whether it is possible for F to be implemented using a single CMOS gate. **If it is possible, draw the CMOS gate using the least number of transistors possible. If it is not possible, explain why**.

1.

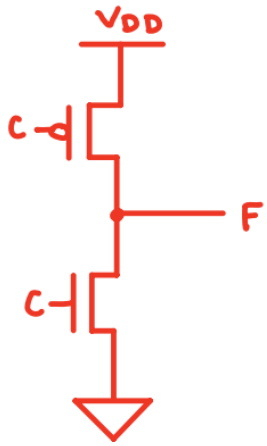
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

F is not inverting, so it can't be expressed as a single CMOS gate. One proof of this is that F(0,0,1)=0 and F(0,1,1)=1. Since flipping B from 0 to 1 causes the output to also flip from 0 to 1, F is not inverting.

2.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

F simplifies to $F(A,B,C) = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C} = \overline{A} \cdot \overline{C} \cdot (B + \overline{B}) + A \cdot \overline{C} \cdot (B + \overline{B}) = \overline{C}$. The CMOS gate is shown below:



Problem 4. Combinational Minispec (19 points)

(A) (4 points)

The following parametric function `g#(n)` performs a specific operation using a, b and c. We want the function `g_copy#(n)` to implement `g#(n)` in a single line of code. Fill in the blanks in `g_copy#(n)` to make the two functions equivalent. Write a single-line expression that uses the ternary operator `(?  :)`.

```
function Bit#(n) g#(Integer n)(Bit#(n) a, Bit#(1) b, Bit#(1) c);
  Bit#(n) ret;
  for(Integer i = 0; i < n; i = i + 1) begin
    ret[i] = a[i] | (b ^ c);
  end
  return ret;
endfunction

function Bit#(n) g_copy#(Integer n)(Bit#(n) a, Bit#(1) b, Bit#(1) c);

  return ((b^c) == 1) ? signExtend(1'b1) : a;

endfunction
```

In this function each bit of `a` is or'ed with `(b^c)`, resulting in one of two cases:

1. `(b^c) == 0` – Each bit of `a` is or'ed with `0`, resulting in that same bit.
2. `(b^c) == 1` – Each bit of `a` is or'ed with `1`, resulting in `1`.

Note the parenthesis around `b^c` are required, and so is specifing the number of bits in the `signExtend`.

(B) (4 points)

The following parametric function `f#(n)` performs a specific operation using a. We want the function `f_copy#(n)` to implement `f#(n)` in a single line of code. Fill in the blank with a single expression to make `f_copy#(n)` equivalent to `f#(n)`.

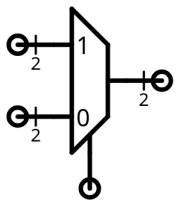
```
function Bit#(n) f#(Integer n)(Bit#(n) a);
  Bit#(n) ret;
  for(Integer i = 0; i < n-1; i = i + 1) begin
    ret[i] = a[i] | a[i+1];
  end
  ret[n-1] = a[n-1] | a[0];
  return ret;
endfunction

function Bit#(n) f_copy#(Integer n)(Bit#(n) a);

  return a | {a[0], a[n-1:1]};

endfunction
```

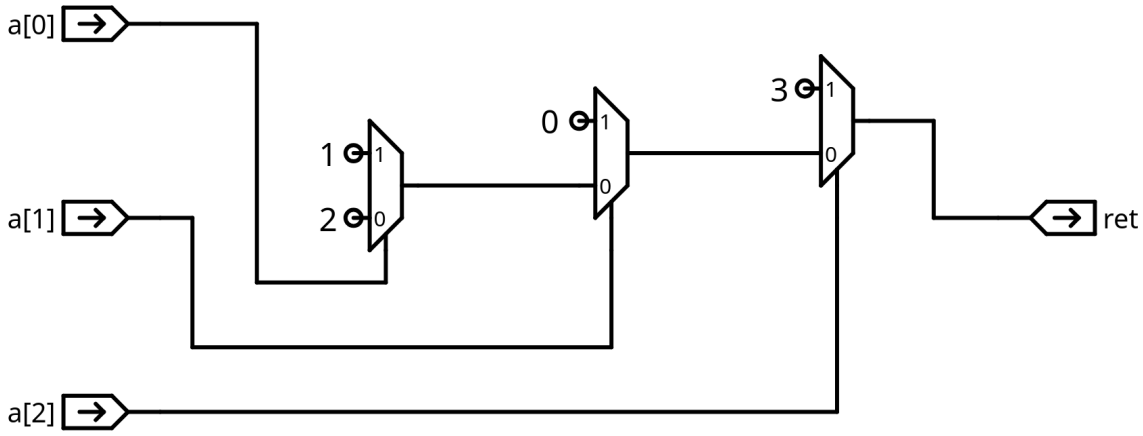
In this function `a` is or'ed with a copy of `a` shifted by one to the right. The MSB of `a` is or'ed with the LSB, which is achieved by shifting in the LSB in the right shift.

(C) (5 points)

Finish the following circuit diagram to implement function `h`, given below. You may use **at most four** 2-bit 2-to-1 multiplexers (shown below), and any number of constants (0, 1, 2, 3, ...). **When drawing multiplexers clearly specify each of the two inputs, the selector, and the output.**

```
function Bit#(2) h(Bit#(3) a);
  Bit#(2) ret = 2;
  for(Integer i = 0; i < 3; i = i + 1) begin
    if(a[i] == 1) begin
      ret = i ^ 1;
    end
  end
  return ret;
endfunction
```





Each if statement is synthesized into a multiplexer. The input values for these multiplexers are all either constants derived from `i^1` or are passed from the output of the previous multiplexer (when the if condition is not met). No XOR gates are needed to evaluate `i^1` since `i` is an Integer and the expression is evaluated at compile time.

(D) (6 points)

Alice wants to send Bob a secret message. For that purpose she implemented the following `encode#(n)` function in minispec which obfuscates an n-bit message. She showed Bob her implementation and now he has to come up with a `decode#(n)` function which will retrieve the original message (i.e., `decode#(n)(encode#(n)(x))==x` for all n-bit `x`). **Help Bob implement the `decode#(n)` function and answer the question about delay below. Make sure to provide a justification for the delay.**

```
function Bit#(n) encode#(Integer n)(Bit#(n) message);
  Bit#(n/2) upper;
  Bit#(n - n/2) lower;
  for(Integer i = 0; i < n; i = i + 1) begin
    if(i % 2 == 0) begin
      lower[i/2] = message[i];
    end else begin
      upper[i/2] = message[i];
    end
  end
  return {upper, lower};
endfunction

function Bit#(n) decode#(Integer n)(Bit#(n) encr_message);
  // Implement Bob's function here

endfunction
```

```
function Bit#(n) decode#(Integer n)(Bit#(n) encr_message);

  Bit#(n) message;
  Bit#(n/2) upper = encr_message[n-1:n-n/2];
  Bit#(n - n/2) lower = encr_message[n-n/2-1:0];
  for(Integer i = 0; i < n; i = i + 1) begin
    if(i % 2 == 0) begin
      message[i] = lower[i/2];
    end else begin
      message[i] = upper[i/2];
    end
  end
  return message;

endfunction
```

Alternative solution:

```
function Bit#(n) decode#(Integer n)(Bit#(n) encr_message);

  Bit#(n) message;
  for(Integer i = 0; i < n; i = i + 1) begin
    if(i % 2 == 0) begin
      message[i] = encr_message[i/2];
    end else begin
      message[i] = encr_message[n - n/2 + i/2];
    end
  end
  return message;
```

```
endfunction
```

Another alternative solution:

```
function Bit#(n) decode#(Integer n)(Bit#(n) encr_message);

  Bit#(n) message;
  for(Integer i = 0; i < n/2; i = i + 1) begin
    message[2*i] = encrypted_message[i];
    message[2*i+1] = encrypted_message[n - n/2 + i];
  end
  if(n % 2 == 1) begin
    message[n-1] = encrypted_message[n/2];
  end
  return message;

  endfunction
```

The delay of Alice's implementation of `encode#(n)` is $\Theta(1)$

Justification: Even though the function uses a for loop iterating from 0 to $n-1$, the logic in each iteration is independent, so the resulting hardware will be implemented in parallel. Each iteration only contains a single assignment, so the resulting delay is $\Theta(1)$.

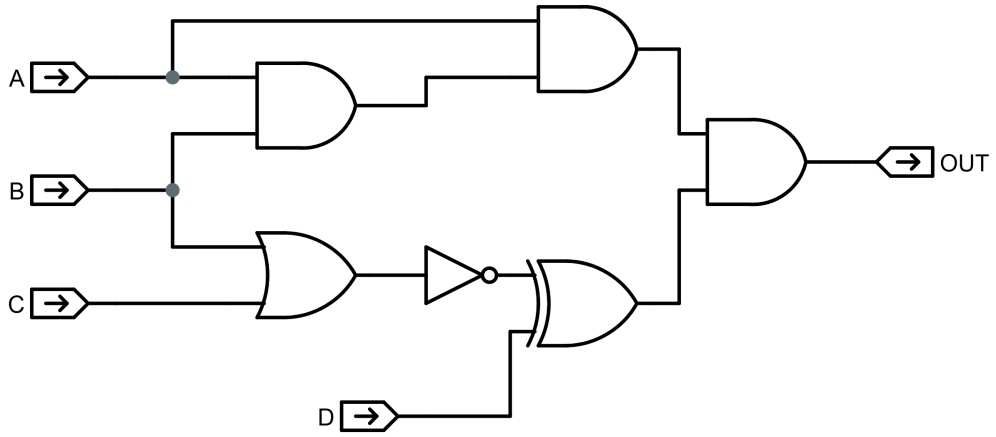
Problem 5. Combinational and Sequential Logic Timing (18 points)

The TAs made a new circuit to be part of their Important-Computation-Machine™ but were busy writing this exam so they're asking the 6.191 students for help analyzing and improving their circuit. The gates they used have the following characteristics:

Gate	t_{PD}	t_{CD}	t_{SETUP}	t_{HOLD}
AND2	1.4ns	0.5ns	—	—
OR2	1.6ns	0.2ns	—	—
XOR2	1.2ns	0.4ns	—	—
INV1	0.7ns	0.2ns	—	—
BUF1	1.0ns	1.0ns	—	—
IREG	0ns	0ns	0ns	0ns
REG	2.5ns	0.6ns	1.0ns	0.4ns

(A) (3 points)

What are the propagation and contamination delays for this combinational circuit they create as their first design?



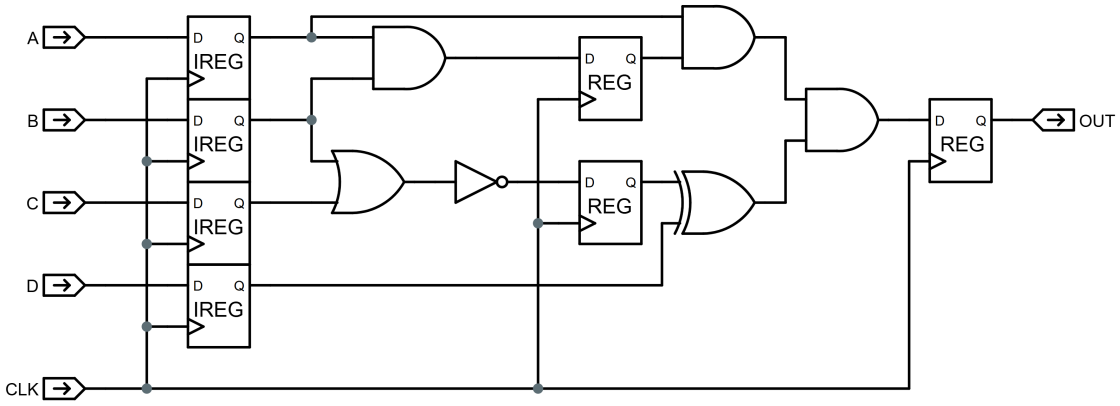
t_{PD} **(ns):** 4.9ns

The longest path is the C (or B) $\rightarrow$ OR $\rightarrow$ INV $\rightarrow$ XOR $\rightarrow$ AND $\rightarrow$ OUT path which has $t_{PD} = 1.6 + 0.7 + 1.2 + 1.4 = 4.9ns$.

t_{CD} **(ns):** 0.9ns

The shortest path is the $D \rightarrow XOR \rightarrow AND \rightarrow OUT$ path which has $t_{CD} = 0.4 + 0.5 = 0.9$.

Using the timing constraints from above, they designed a new sequential circuit based on the original combinational circuit. **Note that they used ideal registers, labeled IREG, for the input signals while the other registers are regular registers, labeled REG**.



(B) (2 points)

Demonstrate that the sequential circuit satisfies all hold time constraints.

There are 8 register to register paths in our circuit that we need to show have valid hold time constraints:
$IREG \rightarrow AND \rightarrow AND \rightarrow REG : 0 + 0.5 + 0.5 \geq 0.4$
$IREG \rightarrow AND \rightarrow REG : 0 + 0.5 \geq 0.4$
$IREG \rightarrow AND \rightarrow REG$: same as above
$IREG \rightarrow OR \rightarrow INV \rightarrow REG : 0 + 0.2 + 0.2 \geq 0.4$
$IREG \rightarrow OR \rightarrow INV \rightarrow REG$: same as above
$IREG \rightarrow XOR \rightarrow AND \rightarrow REG : 0 + 0.4 + 0.5 \geq 0.4$
$REG \rightarrow AND \rightarrow AND \rightarrow REG : 0.6 + 0.5 + 0.5 \geq 0.4$
$REG \rightarrow XOR \rightarrow AND \rightarrow REG : 0.6 + 0.4 + 0.5 \geq 0.4$

(C) (3 points)

What is the shortest clock period that can be used for this circuit?

The longest register-to-register path is the $REG \rightarrow AND \rightarrow AND \rightarrow REG$ path which has setup constraint $t_{PD,REG} + 2 \cdot t_{PD,AND} + t_{SETUP,REG} \leq t_{CLK} \rightarrow 2.5 + 2 \cdot 1.4 + 1.0 = 6.3 \leq t_{CLK}$.

Minimum t_{CLK} (ns): 6.3ns

(D) (2 points)

What are the t_{PD} and t_{CD} values of the sequential circuit?

OUT is connected directly to the rightmost REG so $t_{PD,OUT}$ and $t_{CD,OUT}$ are just those of REG.

t_{PD} (ns): 2.5ns

t_{CD} (ns): 0.6ns

(E) (2 points)

The TAs would like to switch out their AND gates for a different model, what's the smallest t_{CD} the new AND gates could have such that the circuit is still valid?

t_{CD} **(ns):** 0.4ns

We're limited by the $IREG \rightarrow AND \rightarrow REG$ path which must have $t_{CD,IREG} + t_{CD,AND} \geq t_{HOLD,REG}$ i.e. $t_{CD,AND} \geq 0.4$.

(F) (3 points)

These TAs are so fickle! After switching their AND gates they now also want to switch the REG registers. Which of the following registers allows them to have the fastest clock frequency while still having a valid circuit? Provide an explanation for your answer. **Note that you should use the new AND gates and that only the REG registers are being changed**. The IREG ideal registers remain the same.

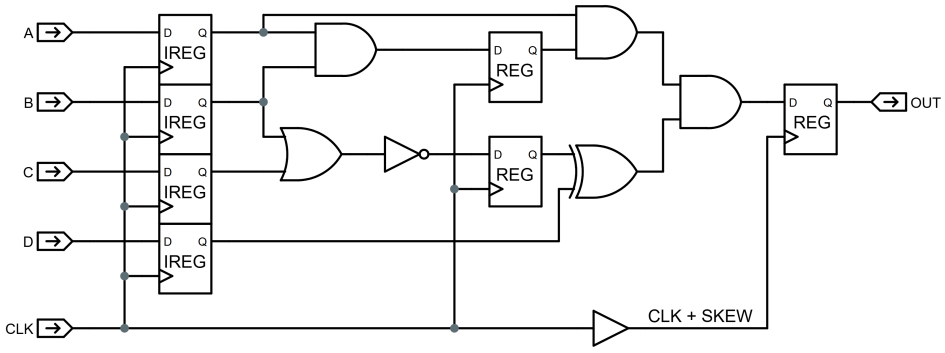
Register	t_{PD}	t_{CD}	t_{SETUP}	t_{HOLD}
REG (current)	2.5ns	0.6ns	1.0ns	0.4ns
REGA	1.4ns	0.7ns	0.5ns	0.5ns
REGB	1.6ns	0.1ns	0.8ns	0.3ns

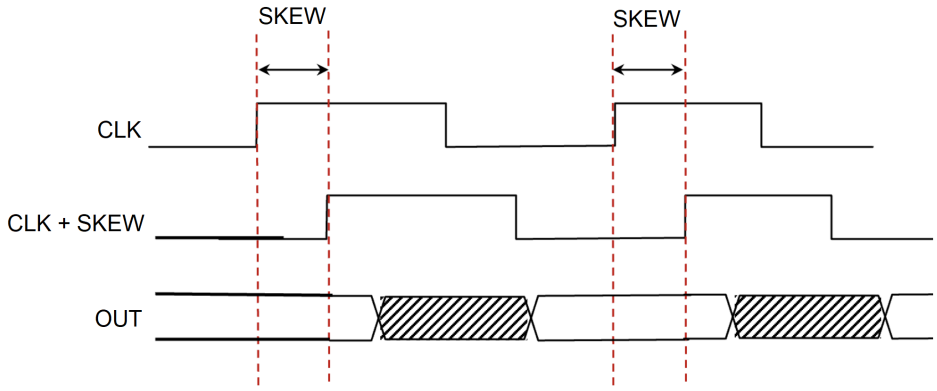
Select one: **REGA** **REGB** **Neither**

Explanation: REGA breaks the hold constraint of the $IREG \rightarrow AND \rightarrow REG$ path when AND has the new $t_{CD} = 0.4ns$. REGB is valid while cutting both the t_{PD} and t_{SETUP} of the register.

(G) (3 points)

After experimenting with their sequential circuit, they realized they need to strengthen the clock signal by using a buffer to drive the clock of the rightmost register. Unfortunately, this adds skew to their clock signal. **Using the original AND and REG timing specifications, what is the minimum t_{CLK} the circuit can now use? If the circuit is not valid, then write "Invalid" and provide an explanation for why the circuit is not valid**.





Minimum t_{CLK} (ns) or Invalid + Explanation: Invalid

The buffer adds 1ns of skew which delays the rightmost register's transition by 1ns. This breaks the hold constraint via the $IREG \rightarrow AND \rightarrow AND \rightarrow REG$ path since $t_{AND2,CD} + t_{AND2,CD} = 0.5 + 0.5 < t_{skew} + t_{REG,HOLD} = 1.0 + 0.4$.

Problem 6. Bit Invader Finite State Machine (10 points)

You're designing the logic for a simple enemy AI, the "Bit Invader", for a new retro arcade game. The invader's behavior is determined by a finite state machine that responds to the player's position.

FSM Specifications:

- Inputs (2-bit SR): S (Player is in Sight (S=1) or not in Sight (S=0)), R (Player is in Range (R=1) or not in Range (R=0)).
- Output (1-bit U): U=1 when the invader is attacking, U=0 otherwise.
- Initial State: The FSM's default state is PATROL.

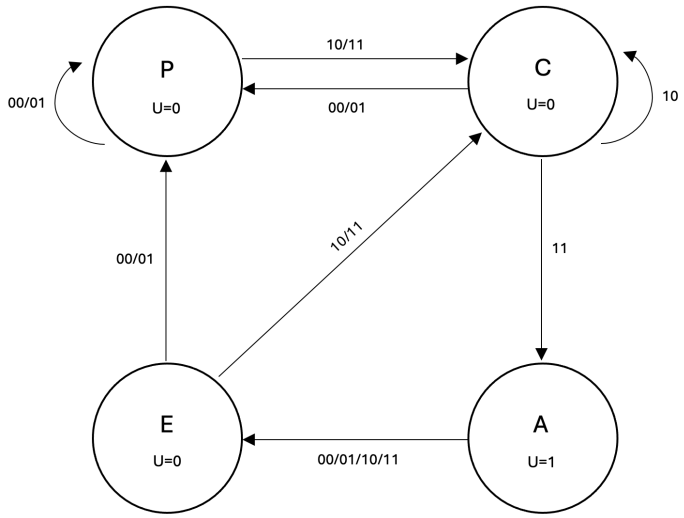
Behavioral Description:

- The invader starts in and remains in the PATROL state as long as the player is unseen. It begins a chase as soon as the player is spotted.
- If visual contact is lost while in a chase, it reverts to PATROL.
- During a chase, the invader will attack if the player moves into close range.
- An attack is a single-cycle action that is always immediately followed by a single-cycle evasive maneuver.
- After the evasive maneuver, the invader resumes its chase if the player is still visible; otherwise, it returns to PATROL.

(A) (5 points)

Draw the state transition diagram for the Bit Invader FSM.

- You must determine the minimum number of states required. Give each state a label (e.g., PATROL (P), etc.).
- Label each state with its name and its corresponding output value for U.
- Label each transition arc with the input condition(s) (SR) that cause it.



This FSM can be implemented in 4 states: PATROL, CHASE, ATTACK, EVADE in this example.

(B) (4 points)

For the following input sequence, determine the state and output for each cycle.

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Current State	P	C	C	A	E	C	A	E
Input (SR)	10	10	11	00	10	11	10	00
Next State	C	C	A	E	C	A	E	P
Output (W)	0	0	0	1	0	0	1	0

(C) (1 points)

How many flip-flops are required to implement this FSM?

Number of flip-flops: 2 flip-flops

This FSM can be implemented in 4 states, and therefore needs 2 bits to represent all states.

END OF QUIZ 1!