1	/16
2	/19
3	/17
4	/17
5	/16
6	/15

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Spring 2026

Quiz #1

Name Solutions	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-302 (Jan)	☐ WF 2, 34-302 (Abdullah)	☐ WF 12, 35-308 (Nathan)
☐ WF 11, 34-302 (Jan)	☐ WF 3, 34-302 (Abdullah)	☐ WF 1, 35-308 (Nathan)
☐ WF 12, 34-302 (Qihang)	☐ WF 10, 35-308 (Raul)	☐ WF 2, 8-205 (Aarush)
☐ WF 1, 34-302 (Qihang)	☐ WF 11, 35-308 (Raul)	☐ WF 3, 8-205 (Aarush)
		☐ 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 (16 points)

Module F follows the voltage transfer characteristic given below.

$$\mathbf{V_{out}} = \begin{cases} \mathbf{1V} & \mathbf{max(V_A, V_B, 0.5V_C) > 2.5V} \\ \mathbf{6V} & \mathbf{max(V_A, V_B, 0.5V_C) < 1.5V} \\ ??? & \mathbf{o/w} \end{cases}$$

(A) (3 points) Given the above specification for module F, determine the Boolean expression that module F(A, B, C) implements.

We can see that when any one of A, B, or C are high, the output is low. Conversely, when all three inputs are low, the output is high. This corresponds to a NOR gate.

Boolean Expression for F(A, B, C) = $\overline{A + B + C}$

(B) (2 points) After inputting fixed input voltages into F for a long time, we measure that $V_{OUT} = 2V$. Which of the following is necessarily true? Select all that apply.

1. $V_A \leq 2.5V$
2. $V_A < 1.5V$
3. $V_B \leq 2.5V$
4. $V_B < 1.5V$
5. $V_C \leq 5V$
6. $V_C < 3V$
7. $V_A > 2.5V$
8. $V_A \geq 1.5V$
9. $V_B > 2.5V$
10. $V_B \geq 1.5V$
11. $V_C > 5V$
12. $V_C \geq 3V$
13. None of the above

Since the output is 2V which is an invalid output, we know the inputs must be invalid. This means neither of the two conditions $\max(V_A, V_B, 0.5V_C) > 2.5V$ nor $\max(V_A, V_B, 0.5V_C) < 1.5V$ can be true.

For $\max(V_A, V_B, 0.5V_C) \not> 2.5V$, we must have that $V_A \leq 2.5V$, $V_B \leq 2.5V$, and $V_C \leq 5V$.

The second condition, $max(V_A, V_B, 0.5V_C) < 1.5V$, is also false. But we can't say anything about the individual inputs since only one of them needs to be above 1.5V (for V_A, V_B) or 3V (for V_C) for the entire condition to be false.

(C) (2 points) After inputting fixed input voltages into F for a long time, we measure that $V_{OUT} = 6V$. Which of the following is necessarily true? Select all that apply.

1. $V_A \leq 2.5V$
2. $V_A < 1.5V$
3. $V_B \leq 2.5V$
4. $V_B < 1.5V$
5. $V_C \leq 5V$
6. $V_C < 3V$
7. $V_A > 2.5V$
8. $V_A \geq 1.5V$
9. $V_B > 2.5V$
10. $V_B \geq 1.5V$
11. $V_C > 5V$
12. $V_C \geq 3V$
13. None of the above

A valid output can be produced from valid input or invalid input. All we know from seeing an output of 6V is that it is not the case that $\max(V_A, V_B, 0.5V_C) > 2.5V$ because if so the output would have to be 1V. In other words, a 6V output can result from valid inputs where $\max(V_A, V_B, 0.5V_C) < 1.5V$ or from invalid inputs where $1.5 \leq \max(V_A, V_B, 0.5V_C) \leq 2.5V$. Together these cases correspond to those in which $\max(V_A, V_B, 0.5V_C) \leq 2.5V$. As a result, all of $V_A, V_B, 0.5V_C \leq 2.5V$.

(D) (6 points) For each of the following proposed voltage thresholds for Module F, find the noise immunity or write INVALID if the specification is invalid. **Explain why each specification is valid or not.**

Specification 1: $V_{OL} = 1.1V, V_{IL} = 1.25V, V_{IH} = 5.5V, V_{OH} = 6V$

Noise Immunity or INVALID: _____0.15_____

Explanation:

The output thresholds are valid since $1V \leq V_{OL} = 1.1V$ and $6V \geq V_{OH} = 6V$.

The low input threshold is valid since anytime the inputs are all below 1.25V, they will satisfy the condition for a valid low input (e.g. $max(V_A, V_B, 0.5V_C) < 1.5V$) since $V_A, V_B = 1.25 < 1.5$ and $0.5V_C = 0.5 \cdot 1.25 < 1.5$.

The high input threshold is valid since if any input is above 5.5V, the valid high input condition will be satisfied (e.g. $max(V_A, V_B, 0.5V_C) > 2.5V$) since $V_A, V_B = 5.5 > 2.5V$ and $0.5V_C = 0.5 \cdot 5.5 = 2.75 > 2.5$.

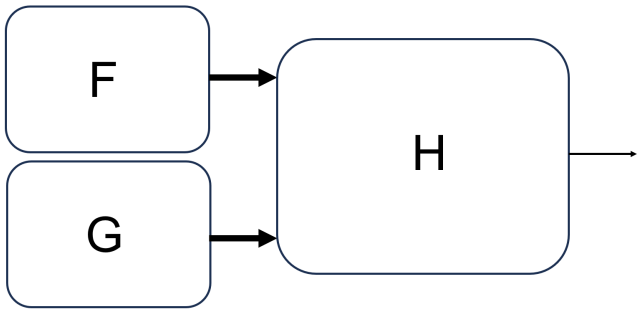
Specification 2: $V_{OL} = 1V, V_{IL} = 3V, V_{IH} = 4V, V_{OH} = 4.5V$

Noise Immunity or INVALID: ___ INVALID ___

Explanation:

This specification allows for a valid input to lead to an invalid output, which is not allowed. An input of 2V would be considered valid (since $2V = V_{IN} < V_{IL} = 3V$). But an 2V input does not satisfy either of the valid output conditions of F, therefore, F could output any possible value – which could very well be an invalid output value. This would mean a valid input led to an invalid output, which is illegal.

(E) (3 points) Consider Module G with voltage thresholds given below, as well as Module H with voltage thresholds given as a ratio of $V_{DD,H}$. Find the valid range of $V_{DD,H}$ such that Module H can be placed after both Module F and G, as shown below.



$V_{OL,G} = 2V$
$V_{IL,G} = 4V$
$V_{IH,G} = 6V$
$V_{OH,G} = 9V$

$V_{OL,H} = 0.2\,V_{DD,H}$
$V_{IL,H} = 0.4\,V_{DD,H}$
$V_{IH,H} = 0.6\,V_{DD,H}$
$V_{OH,H} = 0.8\,V_{DD,H}$

Module H must be able to accept inputs from both modules F and G so it must have input voltage thresholds that accept the outputs of both F and G.

To accept inputs from F, the two following constraints must hold:

$$1V = V_{OL,F} < V_{IL,H} = 0.4V_{DD,H}$$
$$2.5V < V_{DD,H}$$

$$6V = V_{OH,F} > V_{IH,H} = 0.6V_{DD,H}$$
$$10V > V_{DD,H}$$

To accept inputs from G, the two following constraints must additionally be satisfied:

$$2V = V_{OL,G} < V_{IL,H} = 0.4V_{DD,H}$$
$$5V < V_{DD,H}$$

$$9V = V_{OH,G} > V_{IH,H} = 0.6V_{DD,H}$$
$$15V > V_{DD,H}$$

By taking the tightest bounds given by the constraints above, we ensure that H can accept inputs from both F and G. Doing so gives us final constraints of $5V < V_{DD,H} < 10V$.

_____$5V$_____ $\leq \boldsymbol{V_{DD,H}} \leq$ _____$10V$_____

Problem 2. Boolean Algebra (19 points)

Ben Bitdiddle has four friends: Alice, Bob, Carol, and Dave. He greatly enjoys spending time with these friends. However, when Ben is not present, he has noticed they tend to argue with each other and are thus not happy. Fortunately, he has also noticed patterns in who must be present for arguments to break out.

Let $A = 1$ when Alice is present, or $A = 0$ otherwise. Let B, C, and D similarly correspond to Bob, Carol, and Dave, respectively. We let $H(A, B, C, D) = 1$ when the friends are not arguing (and thus happy), and $H = 0$ otherwise.

The following truth table describes the happiness of these four friends depending on who is present and who is not.

A	B	C	D	H
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

(A)

1. (1 point) Ben thinks it's much easier to work with Boolean expressions than truth tables. What is the normal form expression for H?

   Normal form expression for H:
   $\overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,C\,D + A\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,C\,D$

2. (2 points) Ben thought your Boolean expression from (1) was too long. What is the minimal sum-of-products (SOP) expression for H?

$\overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,C\,D + A\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,C\,D$

$= \overline{A}\,\overline{B}\,D\,(\overline{C} + C) + \overline{A}\,B\,C\,(\overline{D} + D) + A\,\overline{B}\,D\,(\overline{C} + C)$

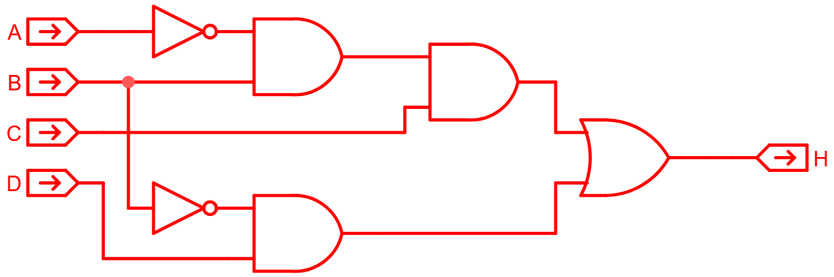
$= \overline{A}\,\overline{B}\,D + \overline{A}\,B\,C + A\,\overline{B}\,D$

$= (\overline{A} + A)\,\overline{B}\,D + \overline{A}\,B\,C$

$= \overline{B}\,D + \overline{A}\,B\,C$

Minimal sum-of-products expression for *H*:_____ $\overline{B}\,D + \overline{A}\,B\,C$ _____

3. (3 points) Ben feels lazy and does not want to implement the Boolean expression for H into a circuit himself. **Draw the minimal SOP circuit using only 2-input AND, 2-input OR, and NOT gates.**



(B) (3 points) Ben tries to accept that arguments arise in friend groups universally. Given this, he has the idea of using the circuit model of his friend group to universally model any Boolean function.

Show how Ben might achieve this by appropriately giving inputs to H such that it functions as a two-input **NAND gate, NAND(X, Y)**. Use only a single H gate with inputs X, Y, 0, and/or 1.

We will choose $C = D = 1$

$H(X, Y, 1, 1) = \overline{B} \cdot (1) + \overline{A}\,B \cdot (1)$
$= \overline{Y} + \overline{X}\,Y$
$= (\overline{X} + \overline{Y})(\overline{Y} + Y)$
$= (\overline{X} + \overline{Y}) \cdot (1)$
$= \overline{X} + \overline{Y}$
$= \overline{XY}$

With this, we have shown that H can become a NAND gate and is thus universal.

NAND(X, Y) = H(__X__, __Y__, __1__, __1__)

(C) Dave got tired of all the arguing in the friend group and decided to permanently leave the group. Assume the resulting circuit G(A, B, C) = H(A, B, C, 0).

1. (2 points) Is circuit G universal? If so, show how. If not, prove why not.

   We will construct an AND gate and a NOT gate.
   Then, we will compose them into a NAND gate.

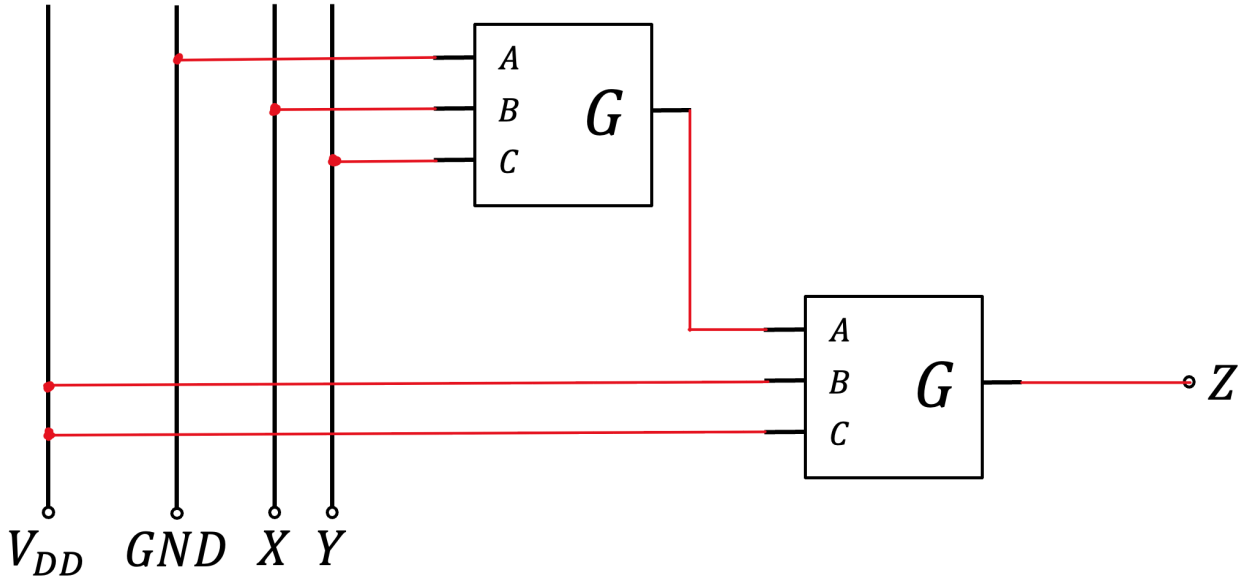
   $G(A,B,C) = H(A,B,C,0)$
   $= \overline{B} \cdot (0) + \overline{A}\,B\,C$
   $= \overline{A}\,B\,C$

   $G(A,1,1) = \overline{A}\,B\,C = \overline{A} \cdot (1) \cdot (1) = \overline{A}$
   This is a NOT gate.

   $G(0,B,C) = \overline{(0)} \cdot BC = 1 \cdot BC = BC$
   This is an AND gate.

   $G(G(0,B,C),1,1)$
   $= \overline{BC}$
   This is a NAND gate.
   Thus, by composing G in this way, we maintain universality.

2. (2 points) If G(A, B, C) is universal then you can create a NAND(X, Y) gate using only G gates. Draw NAND(X, Y) circuit using multiple G gates as a black box. Once again, the inputs are X and Y, and the output is Z. GND acts as a 0 bit, and V_{DD} acts as a 1 bit. If you need other G gates, feel free to draw them.



(D) (6 points) Ben is not satisfied with the fact that his friends don't always get along, so he starts to ponder about satisfiable Boolean expressions. Help him ponder by determining whether the following expressions 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. $\left((\,\overline{x}+y\,)x+z(\,y+x\,)\right)\left(\left(\,\overline{x+y}\,\right)+\left(\,\overline{y+z}\,\right)+\left(\,\overline{z+x}\,\right)\right)$

The first clause expands to:

$$\left((\,\overline{x}+y\,)x+z(\,y+x\,)\right)=\overline{x}x+yx+zy+zx=yx+zy+zx$$

Thus, it is true only if at least two variables are 1.

The second clause expands to:

$$\left(\,\overline{x+y}\,\right)+\left(\,\overline{y+z}\,\right)+\left(\,\overline{z+x}\,\right)=\overline{x}\,\overline{y}+\overline{y}\,\overline{z}+\overline{z}\,\overline{x}$$

Thus, it is true only if at least two variables are 0.

With three variables, it is impossible for two variables to be 1 and another two to be 0. Therefore, this expression is NOT satisfiable.

2. $\overline{\left(\left(\,\overline{ba}\cdot\overline{b\overline{a}}\,\right)+\left(\,\overline{cb+c\overline{b}}\,\right)+\left(\,\overline{ac+a\overline{c}}\,\right)\right)}$

To solve this, we apply DeMorgan's Laws repeatedly.
Applying it to the outermost negation, we get:

$$\begin{aligned}&\overline{\left(\,\overline{ba}\cdot\overline{b\overline{a}}\,\right)}\cdot\overline{\left(\,\overline{cb+c\overline{b}}\,\right)}\cdot\overline{\left(\,\overline{ac+a\overline{c}}\,\right)}\\&=\overline{\left(\,\overline{ba}\cdot\overline{b\overline{a}}\,\right)}\left(\,cb+c\overline{b}\,\right)\left(\,ac+a\overline{c}\,\right)\\&=\overline{\left(\,\overline{ba+b\overline{a}}\,\right)}\left(\,cb+c\overline{b}\,\right)\left(\,ac+a\overline{c}\,\right)\\&=\left(\,ba+b\overline{a}\,\right)\left(\,cb+c\overline{b}\,\right)\left(\,ac+a\overline{c}\,\right)\end{aligned}$$

Now, we factor out common factors via the Distributive Axiom:

$$=abc\left(\,a+\overline{a}\,\right)\left(\,b+\overline{b}\,\right)\left(\,c+\overline{c}\,\right)$$

Using the Complements Axiom, we arrive at:

$$=abc(\,1\,)(\,1\,)(\,1\,)=abc$$

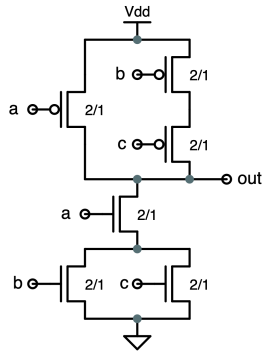
Therefore, choosing $a=b=c=1$, we satisfy this expression.

Problem 3. CMOS Logic (17 points)

(A) (6 points) For each of the following three functions (specified as a Boolean expression or as a truth table), determine if it can be implemented as a single CMOS gate. If it can, draw the CMOS gate using a minimal number of FETs. If not, describe why it cannot be implemented as a single CMOS gate.

1. $F = \overline{ab + a\bar{b}c}$

This simplifies to F = $\overline{a(b + c)}$, which corresponds to the following CMOS gate.



2.

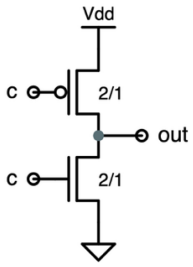
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

This cannot be implemented using CMOS logic because F is not inverting. One example of non-inverting logic is that F(0, 0, 1)=0 and F(1, 0, 1)=1.

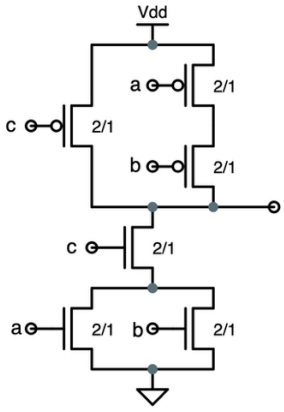
(B) (4 points) Consider the partially specified truth table below. One row of the truth table is unspecified, so there are two possible functions consistent with the table. For each function, determine if it can be implemented as a single CMOS gate. If it can, draw the CMOS gate using a minimal number of FETs. If not, describe why it cannot be implemented as a single CMOS gate.

A	B	C	F
0	0	0	1
0	0	1	**?**
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

When F(0, 0, 1) = 0, the expression simplifies to $F = \overline{C}$. This corresponds to the following CMOS gate.



When F(0, 0, 1) = 1, the expression simplifies to $\overline{F} = C(A + B)$. This corresponds to the following CMOS gate.

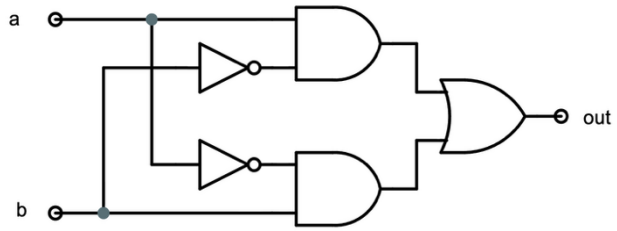


(C) **For the following parts, let function $G = a \wedge b$.**

1. (2 points) If *G* can be implemented using a single CMOS gate, please draw the corresponding single CMOS gate. If it cannot be implemented using a single CMOS gate, then write NONE and explain why. For full credit, use a minimum number of FETs.

   NONE. G(0, 0) = 0, which is not inverting logic, so G cannot be implemented using a single CMOS gate.

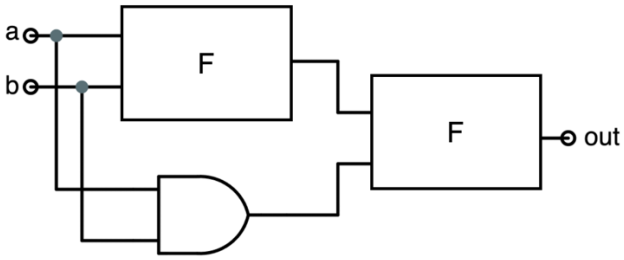
2. (1 point) Consider the implementation shown below, which uses two AND gates and an OR gate. Because a single CMOS gate cannot implement AND or OR, each AND gate is implemented with a CMOS NAND gate followed by a CMOS inverter, and the OR gate is implemented with a CMOS NOR gate followed by a CMOS inverter. How many transistors does this implementation have?



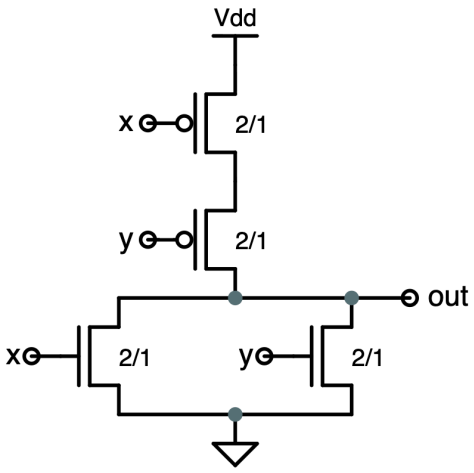
Each AND and OR gate is composed of 6 transistors, and each inverter is composed of 2 transistors, giving us a total of **22 transistors**.

Total number of transistors: _____22_____

3. (3 points) Now consider this alternate implementation shown below, which uses an AND gate and two instances of gate F. Find the Boolean expression for F. If F can be built using a single CMOS gate, draw its CMOS implementation. Otherwise, explain why F cannot be implemented as a single CMOS gate.



F is a NOR gate. Let H represent the AND gate. We know that F(F(a, b), H(a, b)) should be equivalent to G. One intuitive way to approach this problem is to realize that when a = 1 and b = 1, G(a, b) should produce 0, so we do not want the output of H(1, 1) to result in out=1. The CMOS gate that represents NOR is shown below.



4. (1 point) How many transistors does the implementation of **G** from part 3 have?

Each NOR gate is composed of 4 transistors, and the AND gate is composed of 6 transistors, giving us a total of **14 transistors**.

Total number of transistors: _____14_____

4. Combinational Minispec (17 points)

In this problem, we will build efficient circuits to perform a given operation *f* on a sequence of *m*-bit values.

(A) (2 points) Consider the following Minispec function `f`. To better understand the behavior of this function, fill out the values of `result` for the **specified values of `a` and `b`** in the table below (for *m=2*).

```
function Bit#(m) f#(Integer m) (Bit#(m) a, Bit#(m) b);
      Bit#(m) result = 0;
      for (Integer i = 0; i < m; i = i + 1) begin
         case (i % 4)
           0: result[i] = a[i] & b[i];
           1: result[i] = a[i] ^ b[i];
           2: result[i] = a[i] | b[i];
           3: result[i] = 1'b1;
         endcase
      end
      return result;
endfunction
```

a[1]	a[0]	b[1]	b[0]	result[1]	result[0]
0	1	0	0	**0**	**0**
0	0	0	1	**0**	**0**
0	0	1	0	**1**	**0**
1	0	1	0	**0**	**0**
0	1	1	0	**1**	**0**
1	0	0	0	**1**	**0**
1	1	1	1	**0**	**1**

(B) (2 points) Assuming a direct translation without optimization, determine the delay and area of this implementation of `f#(m)` in terms of `m`. Justify your answers.

Delay of `f#(m)`: Θ(____1____)

Area of `f#(m)`: Θ(____m___)

Justification:
The circuits for each iteration of the for loop can be constructed fully in parallel as there are no dependencies between the iterations. Therefore, the delay of the synthesized circuit *will not grow* with m, i.e. Θ(1). However, there are still m subcircuits, one for each loop iteration, so the area will be Θ(m).

You may find the following Minispec review helpful for the remainder of this problem. You may use these functions in your solutions if you find them helpful.

- *Vector is like an array, so given* `Vector#(n, Bit#(m)) inputs` *one can access* `inputs[i]` *for* `0 ≤ i ≤ n-1`. *(*`inputs[i]` *returns an m-bit value).*
- *One can access a subset of the inputs using the following functions:*

  `function Vector#(k, t) take(Vector#(n, t) v);`
  - `take(v)` *returns the first k elements of the vector*

  `function Vector#(k, t) takeTail(Vector#(n, t) v);`
  - `takeTail(v)` *returns the last k elements of the vector.*
- *One can prepend an element x of type t to a vector v of type t using the function:*

  `function Vector#(n+1, t) cons(t x, Vector#(n, t) v);`
  - `cons(x, v)`*returns an n+1 element vector of type t.*

(C) (4 points) Now, write the Minispec function `g` that takes a vector of *n* inputs, each of *m* bits, and applies a chain of `f` functions to the vector elements. For example, for a three-element vector `{A, B, C}`, this circuit `g({A, B, C})` would have output `f(f(A, B), C)`. In other words, it would first perform `f(A, B)` and it would then apply `f` to that result and `C`. It would repeat this for every additional element of the input vector. You may use the already implemented function `f`. You may assume that $n >= 2$.

"One"-liner (using cons):

```
function Bit#(m) g#(Integer m, Integer n)(Vector#(n, Bit#(m)) inputs);
   if (n == 2) return f#(m)(inputs[0], inputs[1]);
   else return g#(m, n-1)(cons(f#(m)(inputs[0], inputs[1]),
                          takeTail(inputs)));

endfunction
```

Explicit implementation

```
function Bit#(m) g#(Integer m, Integer n)(Vector#(n, Bit#(m)) inputs);
      Bit#(m) res = f#(m)(inputs[0], inputs[1]);
      for (Integer i = 2; i < n; i = i + 1) begin
         res = f#(m)(res, inputs[i]);
      end
      return res;
endfunction
```

(D) (3 points) Assuming a direct translation without optimization, determine the delay and area of this implementation of `g#(m, n)` in terms of `m` and `n`. Justify your answers.

Delay of `g#(m, n)`: Θ(____n ____)

Area of `g#(m, n)`: Θ(__mn ____)

Justification:

Since f#(m) has Θ(1) delay (as established in part B), and each instantiation of f depends on the output of the previous iteration, the n instantiations must be chained together sequentially. Therefore, g#(m, n) has Θ(n) delay. Because we require n copies of f to construct g, the area of g is Θ(mn).

(E) (4 points) It is in fact possible to design a faster (i.e. lower asymptotic delay) implementation for `g`. Fill out the Minispec function for this faster implementation: `g_fast`. You may assume that $n >= 2$ and that `n` is a power of 2.

"One"-liner:

```
function Bit#(m) g_fast#(Integer m, Integer n)(Vector#(n, Bit#(m)) inputs);
   if (n == 2) return f#(m)(inputs[0], inputs[1]);
   else return f#(m)(
         g_fast#(m, n/2)(take(inputs)),
         g_fast#(m, n-n/2)(takeTail(inputs))
   );
endfunction
```

Explicit implementation:

```
function Bit#(m) g_fast#(Integer m, Integer n)(Vector#(n, Bit#(m)) inputs);

      if (n == 2) begin
         return f#(m)(inputs[0], inputs[1]);
      end else begin

         Vector#(n/2, Bit#(m)) inp_first_half;
         Vector#(n/2, Bit#(m)) inp_second_half;

         // manually extract using vector indexing

         for (Integer i = 0; i < n; i = i + 1) begin
           if (i < n/2) begin
               inp_first_half[i] = inputs[i];
           end else begin
               inp_second_half[i-n/2] = inputs[i];
           end
         end

         // Run recursively on inp_first_half, inp_second_half
         Bit#(m) first_res  = g_fast#(m, n/2)(inp_first_half);
         Bit#(m) second_res = g_fast#(m, n/2)(inp_second_half);

         return f#(m)(first_res, second_res);
      end

endfunction
```

(F) (2 points) Assuming a direct translation without optimization, determine the delay of `g_fast#(m, n)` in terms of `m` and `n`. Justify your answer.

Delay of `g_fast#(m, n)`: Θ(____log n _____)

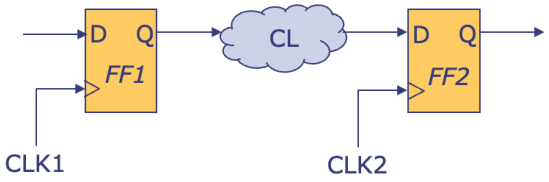
Justification:

This recursive Minispec function will generate a tree of f circuits, which will have depth of log n (base 2), meaning that the delay will scale as Θ(log n).

Problem 5. Sequential Logic Timing (16 points)

To analyze the timing of sequential circuits, we normally assume that all flip-flops in the circuit use exactly the same clock signal. But in practice, this is not quite true: the clock signal arrives to different flip-flops at slightly different times, e.g., due to having different wire lengths. This phenomenon is known as *clock skew*. In this problem, we're going to analyze the effect of skew on sequential logic timing.

Consider the register-to-register path below, annotated with timing parameters:



	CL	FF1 and FF2
Propagation delay	$t_{PD,CL}$	$t_{PD,FF}$
Contamination delay	$t_{CD,CL}$	$t_{CD,FF}$
Setup time		t_{SETUP}
Hold time		t_{HOLD}

(A) (2 points) Assume CLK1=CLK2, i.e., both flip-flops receive exactly the same clock (no skew). What are the setup- and hold-time timing constraints of this circuit?

Setup-time constraint: $t_{PD,FF} + t_{PD,CL} + t_{SETUP} \leq t_{CLK}$

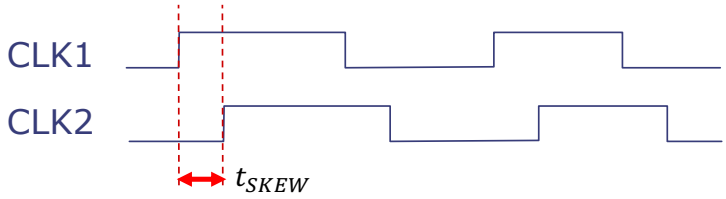
Hold-time constraint: $t_{CD,FF} + t_{CD,CL} \geq t_{HOLD}$

(B) (2 points) What are the propagation and contamination delays of this circuit?

$\mathbf{t_{PD}}$ = $t_{PD,FF}$

$\mathbf{t_{CD}}$ = $t_{CD,FF}$

Now, assume that the time difference between CLK1 and CLK2 is t_{SKEW}, as shown below (in general, t_{SKEW} can be positive or negative, meaning CLK2 can arrive later or earlier than CLK1, respectively).



(C) (3 points) Derive the setup- and hold-time constraints for this circuit, accounting for t_{SKEW}.

Setup-time constraint: $t_{PD,FF} + t_{PD,CL} + t_{SETUP} \leq t_{CLK} + t_{SKEW}$

Hold-time constraint: $t_{CD,FF} + t_{CD,CL} \geq t_{HOLD} + t_{SKEW}$

Often, the clock skew of register-to-register paths is unknown (e.g., due to manufacturing variations). A common way to account for skew is to assume that it can be in a band of values, $t_{SKEW} \in [-\Delta, \Delta]$. In other words, CLK2 can arrive from Δ earlier to Δ later than CLK1.

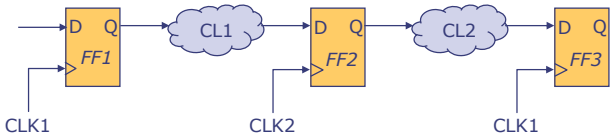
(D) (3 points) Derive the setup- and hold-time constraints for this circuit, accounting for an unknown $t_{SKEW} \in [-\Delta, \Delta]$.
Hint: You should use the worst-case value for t_{SKEW} in each case.

Increasing skew relaxes setup-time constraint, but worsens hold-time constraint; worst-case t_{SKEW} is $-\Delta$ for setup, Δ for hold.

Setup-time constraint: ___$t_{PD,FF} + t_{PD,CL} + t_{SETUP} \leq t_{CLK} - \Delta$___

Hold-time constraint: ___$t_{CD,FF} + t_{CD,CL} \geq t_{HOLD} + \Delta$___

Designers often have some control over skew, and this control can be useful. Consider the circuit below, where **R1 and R3 have the clock signal arrive at exactly the same time (i.e., no skew), but R2's clock signal may be skewed by t_{SKEW}.**



	CL1	CL2	FF1 ,FF2, FF3
Propagation delay	1.5ns	0.5ns	0.2ns
Contamination delay	0.7ns	0.2ns	0.1ns
Setup time			0.3ns
Hold time			0.4ns

(E) (2 points) Analyze the timing of the circuit above when $t_{SKEW} = 0$. Does it meet hold-time constraints? If so, what is its minimum clock cycle time? Justify your answers.

FF2 -> CL2 -> FF3 fails hold-time constraint (0.1ns + 0.2ns < 0.4ns), so this circuit doesn't work without skew.

(F) (2 points) Find the range of values of t_{SKEW} that produce a working circuit.
Hint: Since t_{SKEW} only affects CLK2 (of FF2), the FF1 to FF2 path behaves like what you have already seen in this problem, but the FF2 to FF3 behaves in the opposite way since the skew is on the leading register. Make sure to consider how this affects your constraints.

$t_{SKEW} = 0.4ns$ results in a working circuit with minimum $t_{CLK} = 0.2 + 1.5 + 0.3 - 0.4 = 1.6ns$.

$t_{SKEW} = 0.1ns$ is the smallest skew that makes the circuit work, meeting the FF2->CL2->FF3 hold-time constraint. As skew increases, the hold-time constraint gets looser for FF2->CL2->FF3, but stricter for FF1->CL1->FF2. Past $t_{SKEW} = 0.4ns$, the circuit violates FF1->CL1->FF2 hold time. In this range, FF1->CL1->FF2 sets clock cycle time, since CL1 is 1ns slower than CL2.

_____**0.1**_____ $\leq t_{SKEW} \leq$ _____**0.4**_____

(G) (2 points) What is t_{SKEW} that minimizes clock cycle time? What is that minimum clock cycle time, t_{CLK}. *Hint: Minimum t_{SKEW} does not necessarily result in minimum t_{CLK}.*

$t_{SKEW} = 0.4ns$ results in a working circuit with minimum $t_{CLK} = 0.2 + 1.5 + 0.3 - 0.4 = 1.6ns$.

t_{SKEW} for minimum t_{CLK} = _____**0.4 ns**_____

Minimum t_{CLK} = _____**1.6 ns**_____

Problem 6. Finite State Machines (15 points)

Alice owns an HVAC (Heating, Ventilation, and AC) system in her office. The HVAC has a rudimentary thermostat that allows Alice to set her office to one of the following temperature levels:

- Freezing (F)
- Cold (C)
- Room Temperature (R)
- Warm (W)
- Hot (H)

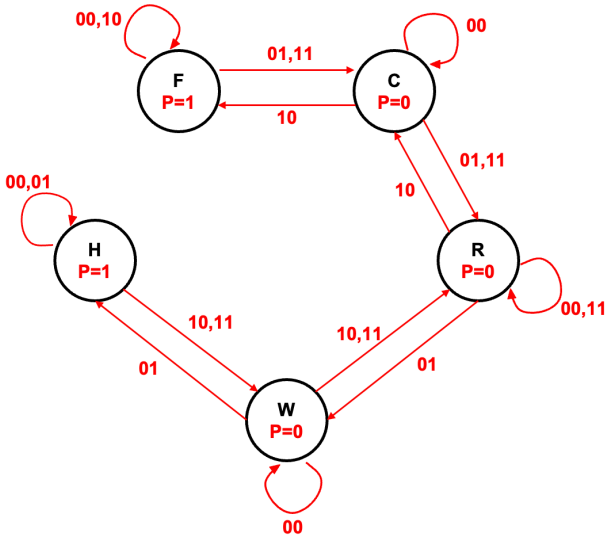
These levels range from coldest (F) to hottest (H) in the order listed above.
The office's initial state is Room Temperature (R).
The thermostat receives a 2-bit input on every clock cycle with the following meanings:

- 00: Maintain the current temperature
- 01: Increase the temperature
- 10: Decrease the temperature
- 11: Return to Room Temperature

Due to limitations of the HVAC system, the **temperature may change by at most one level per cycle**. If the HVAC system is set to "Return to Room Temperature", the office temperature moves one level per cycle toward the Room Temperature state (R), unless it is already at that state. If the office is freezing and receives a decrease command or hot and receives an increase command, then the office temperature will remain the same.

When the office is freezing or hot, the HVAC consumes considerably more power than at other temperature levels, so the thermostat provides a 1-bit output (P) each clock cycle that indicates when the power usage is high. P = 1 when the system consumes more power.

(A) (4 points) Complete the following state transition diagram for HVAC system. Make sure to specify the value of the output, P, for each state. Make sure to label each transition with 00, 01, 10, or 11. Your FSM must handle all possible input combinations.



(B) (1 point) How many flip flops are needed to implement the FSM in part (A)?

Number of flip flops: _____3_____

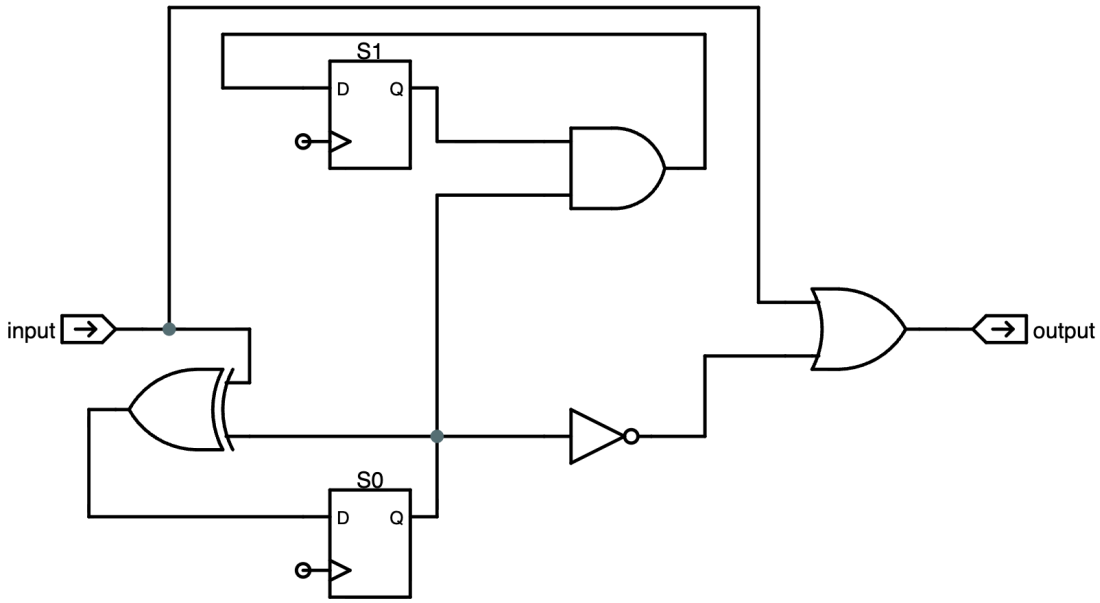
(C) (5 points) For the following input sequence, determine the current state, next state, and output for each cycle. Assume we start at the Room Temperature state in cycle 1.

Recall that the range of temperatures and valid transitions are as follows:
Temperatures: Freezing (F), Cold (C), Room Temperature (R), Warm (W), and Hot (H)
Transitions: Maintain current temperature (00), Increase temperature (01), Decrease temperature (10), Return to Room temperature (11)

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Current State	R	C	F	F	C	R	R	W	W	H
Input	10	10	10	11	11	11	01	00	01	11
Next State	C	F	F	C	R	R	W	W	H	W
Output	0	0	1	1	0	0	0	0	0	1

(D) (5 points) Consider a **new FSM** with the following truth table. Complete the circuit diagram below to implement this FSM. You may only use NOT gates and 2-input AND, OR, and XOR gates in your implementation. For full credit, your implementation must consist of 4 gates or fewer.

S_1^t	S_0^t	Input	S_1^{t+1}	S_0^{t+1}	Output
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	0	1



END OF QUIZ 1!