1	/18
2	/18
3	/15
4	/15
5	/16
6	/18

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Fall 2024

Quiz #1

Name Solutions	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-301 (Varun)	☐ WF 2, 34-303 (Pleng)	☐ WF 12, 34-303 (Ezra)
☐ WF 11, 34-301 (Varun)	☐ WF 3, 34-303 (Pleng)	☐ WF 1, 34-303 (Ezra)
☐ WF 12, 34-302 (Keshav)	☐ WF 10, 34-302 (Hilary)	☐ WF 2, 34-302 (Jessica)
☐ WF 1, 34-302 (Keshav)	☐ WF 11, 34-302 (Hilary)	☐ WF 3, 34-302 (Jessica)
		☐ 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)

Ben Bitdiddle is trying to build a digital buffer by chaining two digital inverters together. He happens to have two inverter chips, each from a different logic family.

One of the inverter chips is the **74LS04** in the TTL family. It only supports a supply voltage of 5V and has the following specifications:

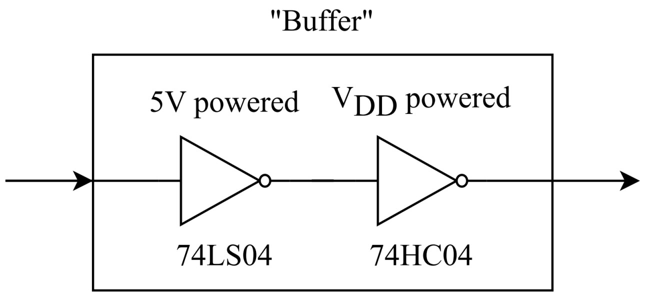
• Guaranteed low output voltage (V_{OL}):	$\leq$0.2V
• Accepted low input voltage (V_{IL}):	$\leq$0.8V
• Accepted high input voltage (V_{IH}):	$\geq$2V
• Guaranteed high output voltage (V_{OH}):	$\geq$3.4V

The other chip is **74HC04** in the CMOS family. The chip has the following specifications where V_{DD} is the supply voltage:

• Guaranteed low output voltage (V_{OL}):	$\leq$3% of V_{DD}
• Accepted low input voltage (V_{IL}):	$\leq$30% of V_{DD}
• Accepted high input voltage (V_{IH}):	$\geq$70% of V_{DD}
• Guaranteed high output voltage (V_{OH}):	$\geq$97% of V_{DD}

Ben decided to use two separate power supplies for his two chips. One power supply is used to provide 5V to the 74LS04. Another power supply can be configured to any value for the 74HC04 chip.

Suppose Ben implements his buffer as in the following diagram:

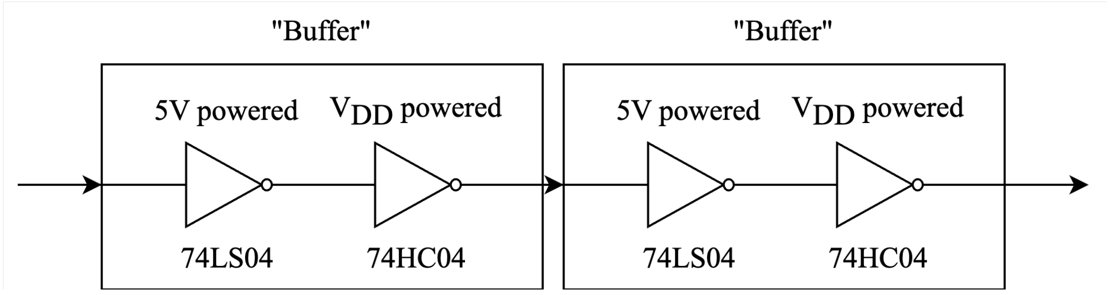


(A) (2 points) If Ben chooses a supply voltage of V_{DD} = 5V for the CMOS chip (74HC04), the specifications for the CMOS chip become: V_{OL} = 0.15 V, V_{IL} = 1.5 V, V_{IH} = 3.5 V, V_{OH} = 4.85 V. With this supply voltage, Ben's circuit does not correctly function as a buffer. Explain why the buffer does not work with this supply voltage. Provide a concrete example in your explanation.

Explanation:

A low input into the TTL inverter may produce a high output of 3.4V which is not a valid high input for the CMOS inverter.

(B) (3 points) Next, Ben decides to try using a supply voltage of V_{DD} = 2V for the CMOS chip (74HC04). The specifications for the CMOS chip become: V_{OL} = 0.06 V, V_{IL} = 0.6 V, V_{IH} = 1.4 V, V_{OH} = 1.94 V. With this supply voltage, Ben's buffer seems to work correctly, but only in isolation. Ben performs an additional testing of his buffer by connecting two of them in series as in the following diagram:



He expects the overall circuit to still function as a buffer but finds that it does not work as expected. Explain why this supply voltage is also not acceptable.

Explanation:

The output from the 74HC04 in the first buffer may produce a high output of 1.94V. However, 1.94V is not considered a valid high input for the next 74LS04 which expects a V_{IH} of 2V.

(C) (4 points) If Ben chooses a supply voltage of V_{DD} = 4.5V for the CMOS chip (74HC04), the specifications for the CMOS chip become: V_{OL} = 0.135 V, V_{IL} = 1.35 V, V_{IH} = 3.15 V, V_{OH} = 4.365 V. It is possible to choose a signaling level specification that makes Ben's circuit a legitimate buffer that can be used in isolation or in combination with other circuits. Choose a signaling level specification for Ben's buffer. You should maximize the noise margins.

Be careful about the ordering of variables here.

V_{OL}: ______**0.135**______ **V**

V_{IL}: ________**0.8**______ **V**

V_{IH}: ________**2.0**______ **V**

V_{OH}: ______**4.365**______ **V**

(D) (3 points) What are the conditions on V_{DD} that must be satisfied to ensure that Ben's circuit functions correctly as a buffer with non-negative noise margins? Make sure to list **all** the required conditions. Do not forget the constraints you discovered in Part (B). You may leave your answers in un-simplified form.

The low output of 74LS04 needs to be lower than 74HC04's accepted low input.

$$\mathbf{0.2 \leq 0.3\, V_{DD}}$$
$$V_{DD} \geq 0.67$$

The high output of 74LS04 needs to be higher than 74HC04's accepted high input.

$$\mathbf{3.4 \geq 0.7\, V_{DD}}$$
$$V_{DD} \leq 4.85$$

The low noise margin must be non-negative, that is, the low output of the 74HC04 needs to be lower than 74LS04's accepted low input.

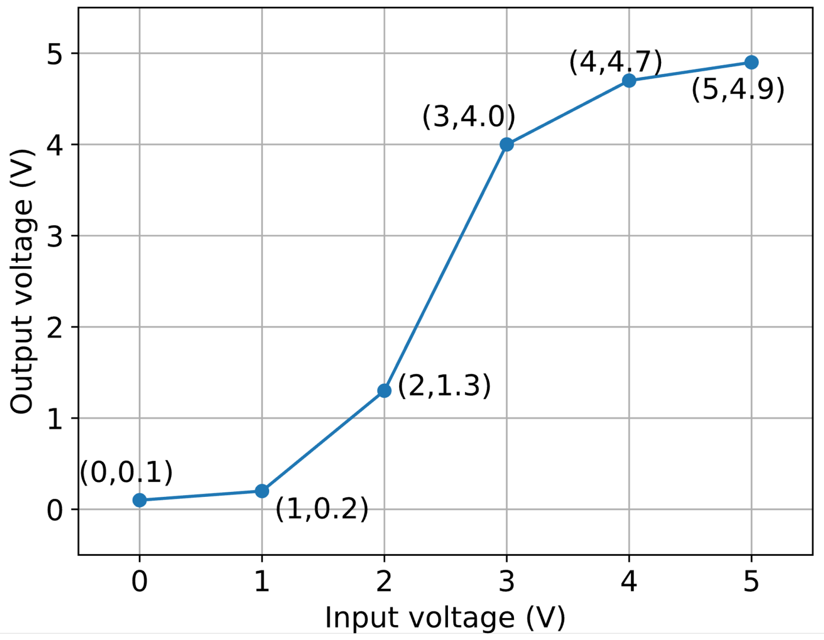
$$\mathbf{0.03\, V_{DD} \leq 0.8}$$
$$V_{DD} \leq 26.67$$

The high noise margin must be non-negative, that is, the high output of the 74HC04 needs to be higher than 74LS04's accepted low input.

$$\mathbf{0.97\, V_{DD} \geq 2}$$
$$V_{DD} \leq 2.06$$

Therefore, V_{DD} must be between 2.06V and 4.85V

Ben finally realized that building a buffer by chaining two inverters from unrelated logic families is a horrible idea. He decided to scrap the idea and build his own buffer from scratch. He managed to obtain a Voltage Transfer Curve (VTC) for his buffer as in the following plot.



(E) (6 points) For each of the signaling level specifications below, find the noise immunity or write ILLEGAL if the specification is illegal. If the specification is illegal, explain why it is illegal.

Be careful about the ordering of variables here.

Case 1: V_{OL} = 1V, V_{IL} = 0.6V, V_{IH} = 3V, V_{OH} = 4V

Noise immunity (V) or ILLEGAL: ___ILLEGAL___

Explanation if ILLEGAL:

$V_{IL} > V_{OL}$ which results in a negative noise margin.

Case 2: V_{OL} = 1V, V_{IL} = 1.4V, V_{IH} = 2.2V, V_{OH} = 3.2V

Noise immunity (V) or ILLEGAL: ___ILLEGAL___

Explanation if ILLEGAL:

High in does not always produce high out. Vin = 2.2V produces Vout ~1.8V which is not < V_{IH} = 2.2V.

Case 3: V_{OL} = 1V, V_{IL} = 1.2V, V_{IH} = 3V, V_{OH} = 4V

Noise immunity (V) or ILLEGAL: _______0.2V____

Explanation if ILLEGAL:

Problem 2. Boolean Algebra (18 points)

(A) (3 points) Consider the truth table below for the Boolean function F. Find the normal form and a minimal sum-of-products expression for $F(A, B, C)$.

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

1. What is the normal form expression for $F(A, B, C)$?

Normal form for F = $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$

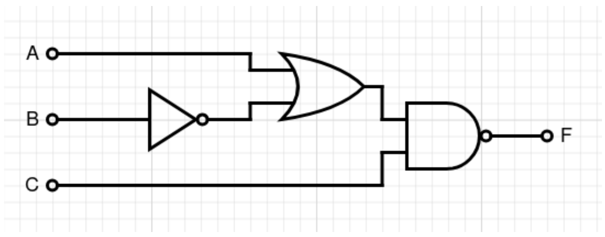
2. What is the minimal sum-of-products expression for $F(A, B, C)$?

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

Minimal sum-of-products form for F = $\overline{C} + \overline{A}\,B$

(B) (3 points) Draw the circuit that implements F **using 3 or fewer gates**. You **may only use inverters and 2-input OR, NOR, AND,** and **NAND** gates in your circuit.

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



(C) (2 points) Determine whether the 3-input Boolean function $G(A,B,C) = AB\overline{C} + C$ is functionally complete (universal). If it is, prove it. Otherwise, explain why it is not.

Cannot be universal because you can’t make an inverter out of it.
$G(A,B,C) = AB\overline{C} + C = AB + C$ which is only AND and OR. Since there is no negation in this equation, there is no way to produce an inverter.

(D) (6 points) Find **a minimal sum-of-products** expression for each of the following Boolean expressions. **Pay close attention to which variables are under the overbars.** To get credit for your answer, you must **use boolean algebra axioms and properties and show your work!**

1. $a(b\overline{c} + a) + \overline{ac}$

   $ab\overline{c} + a + \overline{ac}$ (distribute)
   $ab\overline{c} + a + \overline{a} + \overline{c}$ (de morgans)
   1 (complements)

2. $\overline{a}\mathrm{bc} + bc\overline{d}e + abc\overline{d}$

   $bc(\overline{a} + a\overline{d}) + bc\overline{d}e$ (distribute)
   $bc(\overline{a} + \overline{d}) + bc\overline{d}e$ (distribute and complements)
   $bc(\overline{a} + \overline{d} + \overline{d}e)$ (distribute)
   $bc(\overline{a} + \overline{d})$ (absorption)
   $bc\overline{d} + bc\overline{a}$

3. $\overline{ab(\overline{c} + a) + \overline{bc} + d}$

   $\overline{ab\overline{c} + ab + \overline{bc} + d}$ (distribute)
   $\overline{ab + \overline{bc} + d}$ (absorption)
   $(\overline{ab})\,\overline{\overline{bc} + d}$ (de morgans)
   $(\overline{a} + \overline{b})bc\overline{d}$ (de morgans)
   $(\overline{a}bc\overline{d} + \overline{b}bc\overline{d})$ (distribute)
   $\overline{a}bc\overline{d}$ (complements)

(E) (4 points) Determine whether each of the Boolean expressions below are satisfiable. If it is satisfiable, give the input assignments that make the expression satisfiable. If it is not satisfiable, show why it is not.

1. $(a+b)a\overline{b}(d+cd)$

   To be satisfiable, the expression must = 1 for some assignment of the variables.

   Middle term $a\overline{b}$ implies a=1, b=0

   a=1 makes left term = 1, right term can be simplified as d=1

   a=1, b=0, d=1, c = 1 or 0

2. $(a\overline{e}+c)\left(\overline{ab}c+ad\right)(\overline{ca})$

   To satisfy the last term $\overline{ca}=1,$ so either c or a or both must be 0. To satisfy the first term, either a or c must be 1. So either (c = 1 and a = 0) or (c = 0 and a = 1).

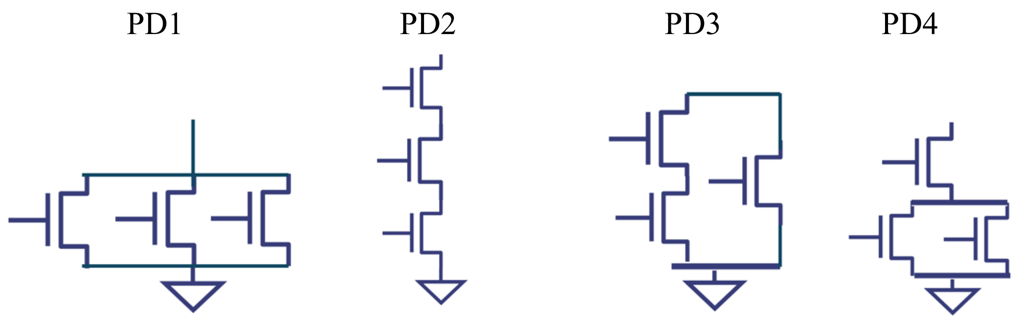
   Try to fix c=1 which implies a=0 and e can be 0 or 1 (now left and right term = 1)

   if c=1 and a=0, then from $\left(\overline{ab}c+ad\right),$ b and d can be 0 or 1.

   a=0, b=1 or 0, c=1, d=1 or 0, e = 1 or 0

Problem 3. CMOS Logic (15 points)

(A) (5 points) You are given several **pulldown** circuits for 3-input CMOS gates. You can choose between the following 4 options



You can connect one of the inputs (A, B, C, D) or a constant (GND, or VDD) to each nFET in the pulldown circuits shown above. Assume that there is a complementary pullup circuit connected to each pulldown. For each of the following Boolean functions, determine whether it can be implemented as one of the given CMOS gates. This could require simplifying the logic equations. If none of the pulldowns can be used to represent the function, write "NONE." If multiple CMOS gates are usable please feel free to choose any of them.

1. F(A, B, C) = $\overline{AB + C}$ **CMOS or NONE= __ PD3___**

   To find the pulldown we need to take not(F) = $AB + C$ which translates directly to (A in series with B, in parallel with C) which is PD3.

2. F(A, B, C, D) = $\bar{A}\,\bar{B}\,\bar{C}\,\bar{D} + \bar{A}\,\bar{B}\,\bar{C}\,D$ **CMOS or NONE= __ PD1___**

   $\bar{A}\,\bar{B}\,\bar{C}\,\bar{D} + \bar{A}\,\bar{B}\,\bar{C}\,D = \bar{A}\,\bar{B}\,\bar{C} = \overline{A + B + C}$
   So not(F) = A + B + C which corresponds to PD1.

3. F(A,B,C) = $\overline{CA} + (\overline{BA} + BC)B$ **CMOS or NONE= __NONE__**

   Consider what happens when we set (A=1, B=0, C=1). In this case the output is 0.
   Then, increasing B to 1 gives us an output of 1.
   This means a rising input has caused a rising output, which is illegal in CMOS.

4. F(A,B,C,D) = $\overline{(AB)\overline{(C + D)} + \overline{D + AC}}$ **CMOS or NONE= __NONE__**

   This is tedious to simplify, but the double-nots suggests this is not implementable with a CMOS. Let's try some examples. Trying A=1, B = 1, C = 0, D = 0 gives us an output of 0. But then flipping C to 1 here gives us an output of 1. Once again we get a rising input leading to rising output which is illegal.

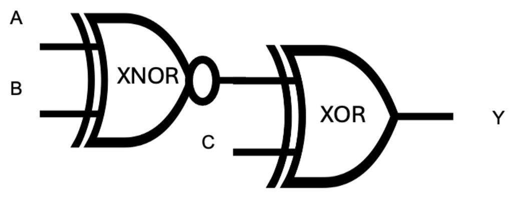
5. F(A,B,C,D) = $(\bar{B} + \bar{C}D)\bar{C} + \overline{DA}$ **CMOS or NONE= __ PD2___**

   $= \bar{B}\bar{C} + D\bar{C} + \overline{DA} = \bar{B}\bar{C} + D\bar{C} + \bar{D} + \bar{A}$

We can then use the absorption law to simplify this to $\bar{B}\bar{C} + \bar{C} + \bar{D} + \bar{A} = \bar{C} + \bar{D} + \bar{A}$

This is implementable as a series of three-parallel signals in our pullup, which becomes three-series signals in our pulldown or PD2.

(B) (4 points) For the combinational logic below, determine if it can be implemented with a single CMOS gate. If so, draw the gate using the **minimum number of transistors**. If not explain why this cannot be implemented as a single CMOS gate**.**



CMOS gate or explanation:

The truth table shows the XNOR of A and B and then the result of XOR of that with C.

A	**B**	**C**	**XNOR**	**XOR**
0	0	0	1	1
0	0	1	1	0
0	1	**0**	0	**0**
0	1	**1**	0	**1**
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

Looking at the 3rd and 4th rows of the truth table, we see that C rising from 0 to 1 while A and B don't change results in the XOR rising from 0 to 1. The implies that a rising input leads to a rising output which cannot be implemented with a single CMOS gate.

(C) (6 points) For the truth table below, determine if it can be implemented with a single CMOS gate. If so, draw the gate using the **minimum number of transistors**. If not explain why this cannot be implemented as a single CMOS gate**.**

A	B	C	Y
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	0
1	1	1	0

CMOS gate or explanation:

First we can analyze the signal for the inverse of Y to determine the pulldown circuit

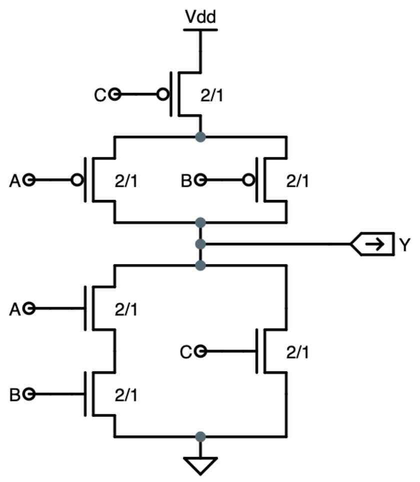
$$\bar{Y} = \bar{A}\,\bar{B}\,C + AB\bar{C} + A\,\bar{B}\,C + \bar{A}\,B\,C + ABC$$

We can then group this into the terms

$$\bar{Y} = C(AB + \bar{A}B + A\bar{B} + \bar{A}\bar{B}) + CAB$$

$$\bar{Y} = C + C\,AB = C + AB$$

Which means this can be turned into a pull-down circuit which is C in parallel with (A in series with B) as shown below.



Problem 4: Combinational Circuits in Minispec (15 points)

The company Ben Bitdiddle works at, *John Street*, performs a lot of comparisons between very large dollar amounts stored as integers. Until now, their code made use of chained comparators to perform the computations. To speed things up, Ben decides to convert the chained comparator implementation to a recursively called function to reduce propagation delay.

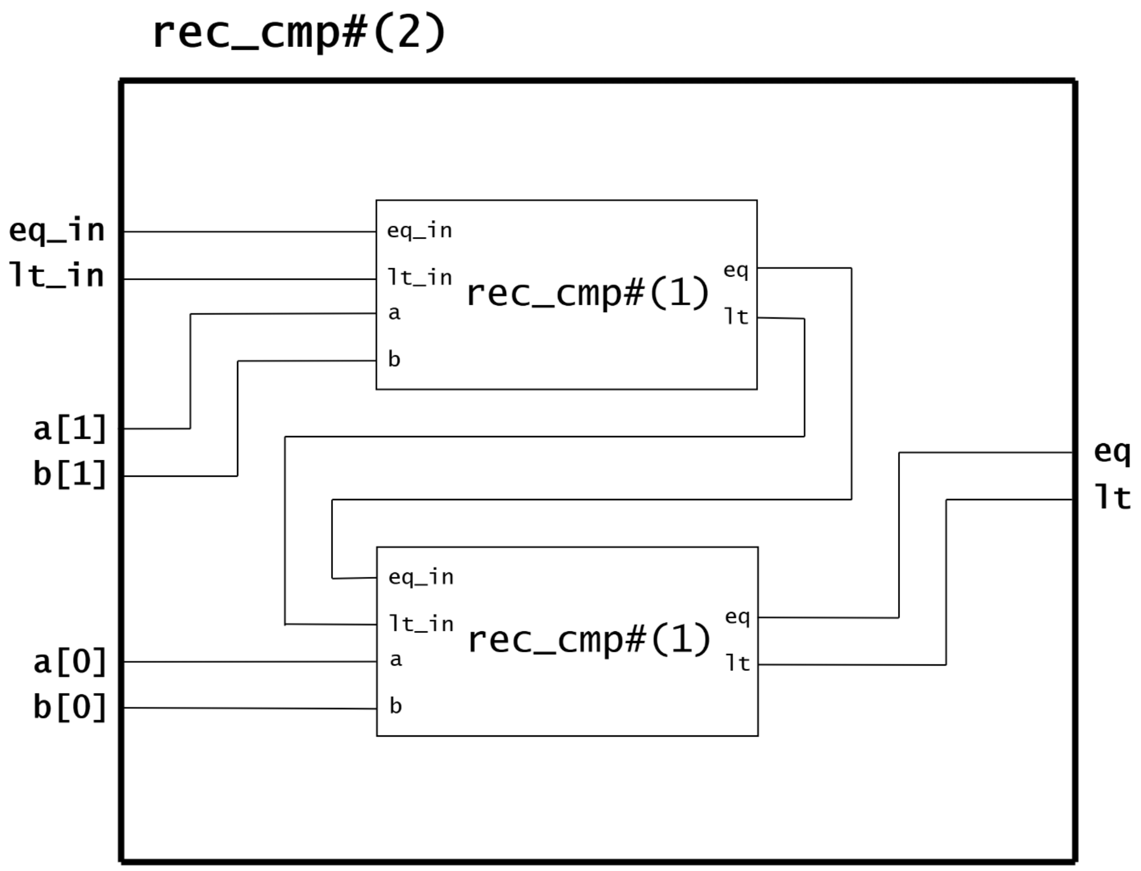
Ben first thinks of splitting the long chain of bitwise compare operations into upper and lower halves recursively. The `rec_cmp` function compares two n-bit values, `a` and `b`, and returns a 2-bit output equal to :

- `‘2b10` if `a` and `b` are equal (`eq=1`) ,
- `‘2b01` if $a < b$ (`lt=1`),
- `‘2b00` if $a > b$.

Ben's recursive implementation is shown below.

```
function Bit#(2) rec_cmp#(1) (Bit#(1) a, Bit#(1) b, Bit#(1) eq, Bit#(1) lt);

    Bit#(1) eq_i = eq & ~(a^b);
    Bit#(1) lt_i = lt | eq & ~a & b;
    return {eq_i, lt_i};

endfunction

function Bit#(2) rec_cmp#(Integer n)
                         (Bit#(n) a, Bit#(n) b, Bit#(1) eq, Bit#(1) lt);

  Bit#(2) upper = rec_cmp#(n-n/2)(a[n-1:n/2], b[n-1:n/2], eq, lt);
  Bit#(2) lower = rec_cmp#(n/2)(a[n/2-1:0], b[n/2-1:0], upper[1], upper[0]);
  return lower;

endfunction
```

(A) (4 points) Complete the circuit diagram for the above parametric function when `n = 2`. Blocks representing function calls at `n = 1` have been provided. You can also use the standard gates, muxes, and constant inputs of 0 and 1 as needed.



Consider the same circuit for a general parameter `n`. How does the propagation delay of the circuit grow with `n`? Provide an explanation for why this is the propagation delay for this implementation.

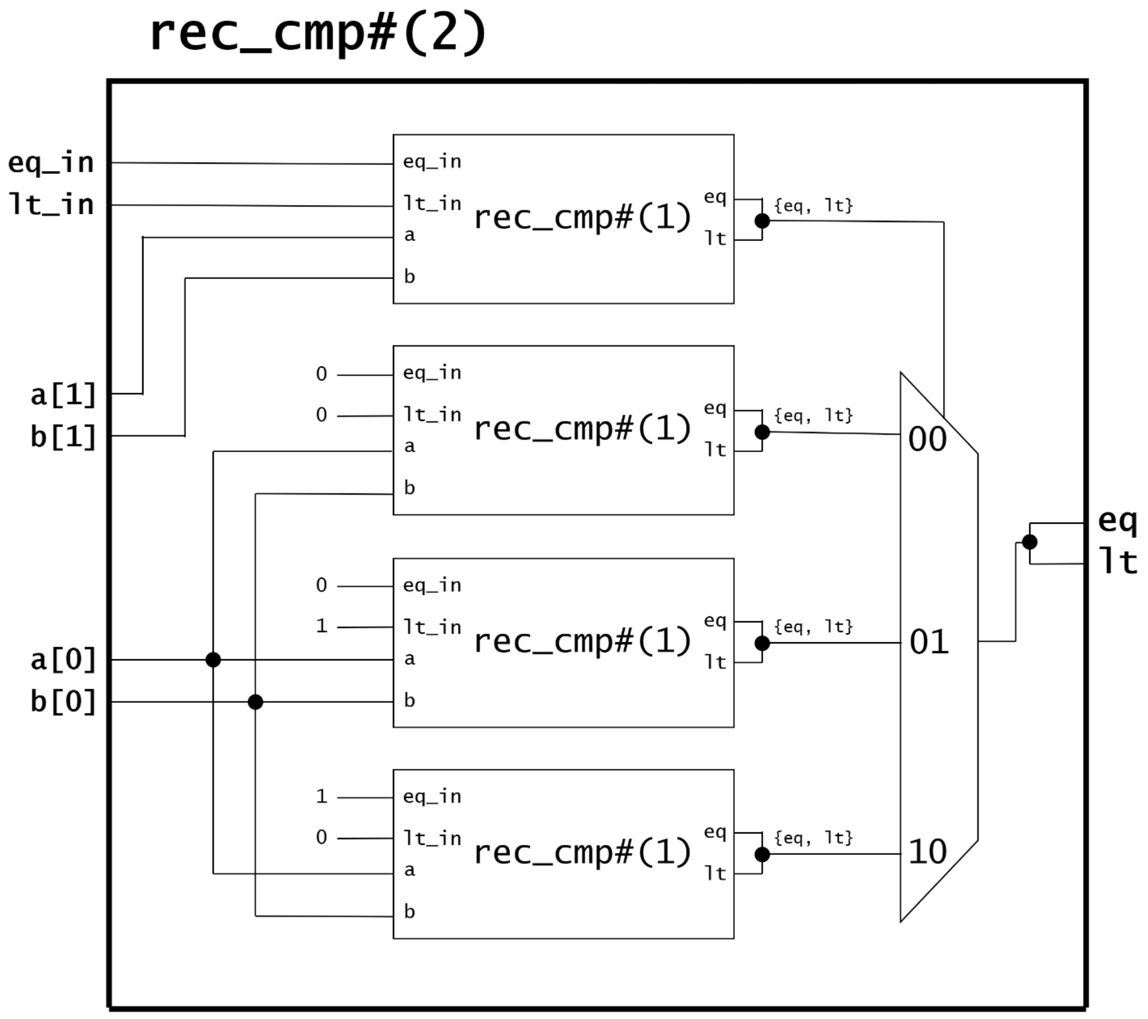
The propagation delay for this circuit in terms of `n` is `Θ(`____n______`)`

Justification: The critical path from input `a[n-1]` to the output `out[0]` passes through all `n` blocks, each of which has constant delay.

(B) (5 points) After testing his original code on various large numbers, Ben finds that his recursive implementation didn't quite get the speedup he expected. He decides to fix his code to **remove the chaining between the `upper` and `lower`** recursive calls. Fix Ben's `rec_cmp` function to remove this chaining **between the `upper` and `lower`** (Hint: you may need up to 4 recursive calls to `rec_cmp`). You may assume that the base case `rec_cmp#(1)` is already defined in the same way as in Ben's original code. **Note that `n` is not necessarily a power of 2.**

```
function Bit#(2) rec_cmp#(Integer n) (Bit#(n) a, Bit#(n) b,
                                      Bit#(1) eq, Bit#(1) lt);

    Bit#(2) upper = rec_cmp#(n-n/2)(a[n-1:n/2], b[n-1:n/2], eq, lt);
    Bit#(2) lower_00 = rec_cmp#(n/2)(a[n/2-1:0], b[n/2-1:0], 0, 0);
    Bit#(2) lower_01 = rec_cmp#(n/2)(a[n/2-1:0], b[n/2-1:0],
                                               0, 1'b1);
    Bit#(2) lower_10 = rec_cmp#(n/2)(a[n/2-1:0], b[n/2-1:0],
                                               1'b1, 0);
    return case (upper)
        2'b00: lower_00;
        2'b01: lower_01;
        2'b10: lower_10;
        default: 0;
    endcase;

endfunction
```

(C) (6 points) Complete the circuit diagram for the above parametric function when `n = 2`. Blocks representing function calls at `n = 1` have been provided. You can choose to use any number of these blocks, the standard gates, muxes, and constant inputs of 0 and 1 as needed.



Consider the same circuit for a general parameter `n`. How does the propagation delay of the circuit grow with `n`? Provide an explanation for why this is the propagation delay for this implementation.

The propagation delay for this circuit in terms of `n` is `Θ(__log(n)_)`

Justification: Any path from input to output goes through exactly one `rec_cmp#(1)` block and `log(n)` layers of similarly-sized muxes.

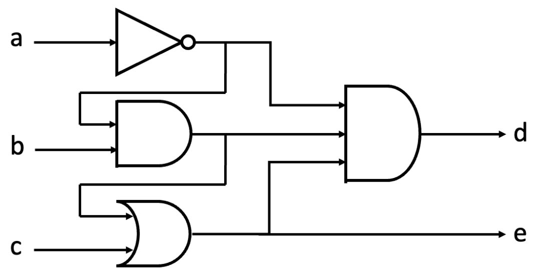
Problem 5. Combinational and Sequential Logic Timing (16 points)

(A) (4 points) *When-vidia* is a spinoff of a large gaming and AI hardware company that focuses on clocks and other timing devices. As a bright MIT intern there, you have been tasked with figuring out the exact propagation and contamination delays of circuits that they have built.

They use a logic family with the following characteristics for their circuits (in *nanoseconds*):

Gate	t_{PD}	t_{CD}
AND2	1.7	0.4
OR2	1.8	0.4
NOT	0.7	0.3
AND3	2.4	0.6

The first circuit they ever designed was purely combinational and looked as follows:



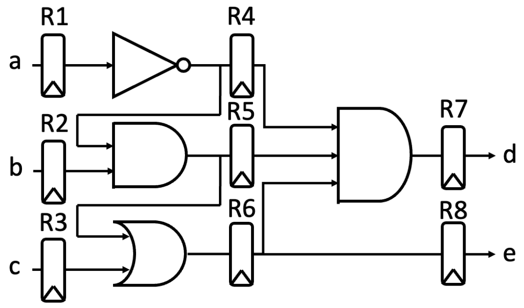
What are the overall t_{PD} and t_{CD} values of this circuit?

The longest path goes through all four gates. The shortest path goes through only the OR2 gate.

t_{PD} (ns): _____**6.6**_____

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

They are now moving on to connecting their combinational logic through some sequential logic elements.



They need to select which type of register to use for their design. They are considering three choices, `Reg A`, `Reg B`, and `Reg C` with timing characteristics shown in the table below. For convenience, we also include a copy of the combinational timing specifications table from the previous page.

Gate	t_{PD}	t_{CD}	t_{Setup}	t_{Hold}
Reg A	0.5	0.2	0.3	0.4
Reg B	0.4	0.1	0.4	0.2
Reg C	0.6	0.4	0.3	0.3

Gate	t_{PD}	t_{CD}
AND2	1.7	0.4
OR2	1.8	0.4
NOT	0.7	0.3
AND3	2.4	0.6

(B) (4 points) If their goal is to produce a **valid** sequential circuit with minimum $\mathbf{t_{CLK}}$ which type of register should they use? Circle which register type to select and explain why you should choose that register type.

Circle one: **Reg A** **Reg B** [✓ **Reg C**]

Explanation:

Back to back registers require $t_{CD,Reg}$ >= $t_{Hold,Reg}$. Reg C is the only register that satisifies this constraint.

(C) (4 points) Using the register type you selected in part B, please find the minimum $\mathbf{t_{CLK}}$ value for which this circuit operates correctly. For full credit, you must show that all timing constraints are fully satisfied.

From part B, since Reg C satisfies the hold time constraint for back to back registers it also satisfies it for all other paths that have some additional combinational logic between the registers.

The longest Register to Register delay is from R1 to R6 because it includes the delays of the inverter, the AND2, and the OR2 gates whose sum is larger than the delay of the AND3 gate. So tCLK >= tPD,Reg + tPD,comb + tSetup,Reg = 0.6 + 0.7 + 1.7 + 1.8 + 0.3 = 5.1

Minimum $\mathbf{t_{CLK}}$ (ns): ______**5.1**__________

(D) (4 points) What are the $\mathbf{t_{PD}}$ and $\mathbf{t_{CD}}$ values of the sequential circuit?

The propagation and contamination delay of a sequential circuit is measured from the rising edge of the clock until the output is ready. Since there is no combinational logic between R7 and output d or R8 and output e that means that the delay is equal to the propagation and contamination delay of the register.

t_{PD} (ns): _____0.6_____

t_{CD} (ns): _____0.4_____

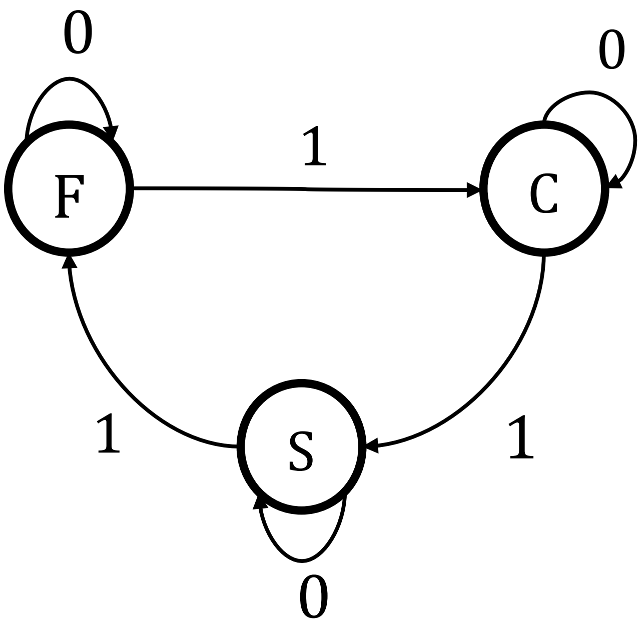
Problem 6. Finite State Machines (18 points)

Carrie Ripple is an engineer and just had a baby. She's trying to learn how to get the baby to stop crying, so she has made an FSM to understand the baby's behavior. The baby can be in one of three states:

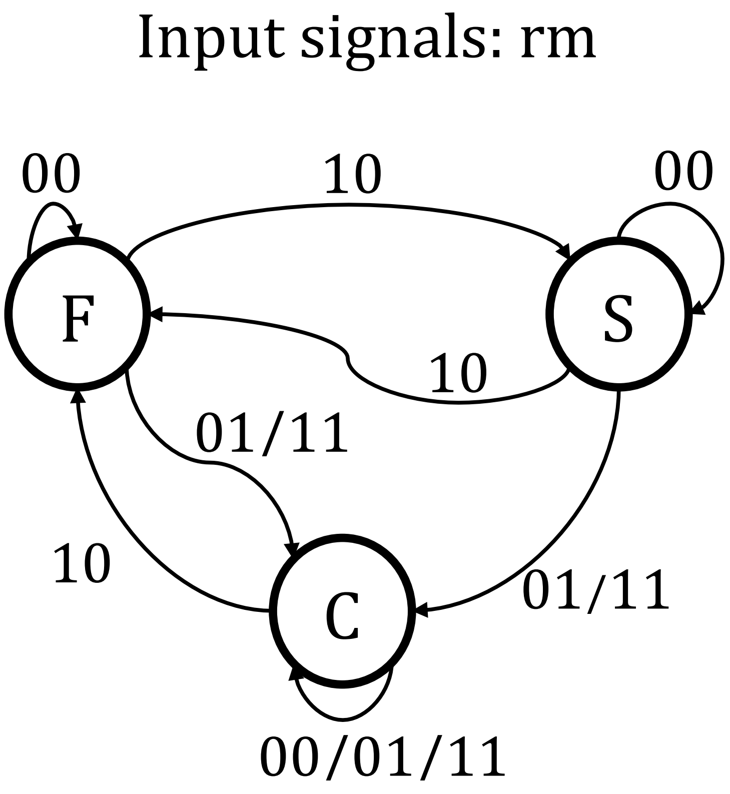
- Fed (F)
- Cleaned (C)
- Slept (S)

(A) (2 points) The baby does not speak yet, and for now assume it only has one input signal: crying (r). If the baby is fed and cries, she tries cleaning it. If the baby still cries, she tries getting it to sleep. If it still cries, she tries feeding it again. If in any state the baby stops crying, she does nothing and walks away quietly.

Draw the 3-state FSM for the baby.



(B) (3 points) Carrie realized that crying is not a sufficient signal to debug the baby. She now also uses the baby smelling sub-optimally (m) to decide what to try next. Below we give you the new FSM for the baby. The input signals are specified using 2-bits where the most significant bit is r (crying) and the least significant bit is m (smelling). If multiple input signals result in the same state transition, they are indicated with a separating **/** between them.



If the baby is in the Slept state (S) and it cries but doesn't smell, what state does this FSM go to?

Next State: _______F_________

If the baby is in the Cleaned state (C) and it smells, what state does the FSM go to?

Next State: _______C_________

How many flip flops does the baby debugging FSM require to encode all possible states?

Number of Flip Flops: ________2________

(C) (3 points) Before Carrie starts using the FSM, she wants to know how the FSM will behave given a series of inputs.

What is the resulting state of the baby at the end of each sequence of inputs provided? **Assume the baby is in the Cleaned state (C), at the beginning of each sequence.**

i. 10 10 01 00 ________**C**________

ii. 10 11 01 00 10 ________**F**________

iii. 11 10 00 11 01 00 01 00 10 10 ________**S**________

(D) (3 points) Fill out the following truth table for Carrie's FSM based on her state transition diagram. The output should be 1 whenever the baby stops crying and stops smelling and it is Fed or Cleaned, and 0 otherwise. **Note that although in many FSMs the output only depends on the current state, that is not the case for this FSM.**

State	**Input (rm)**	**Next State**	**Output**
F 00	00	**F 00**	**1**
F 00	01	**C 10**	**0**
F 00	10	**S 01**	**0**
F 00	11	**C 10**	**0**
S 01	00	**S 01**	**0**
S 01	01	**C 10**	**0**
S 01	10	**F 00**	**0**
S 01	11	**C 10**	**0**
C 10	00	**C 10**	**1**
C 10	01	**C 10**	**0**
C 10	10	**F 00**	**0**
C 10	11	**C 10**	**0**

From this truth table we can derive the following Boolean expressions for the next state and the output.

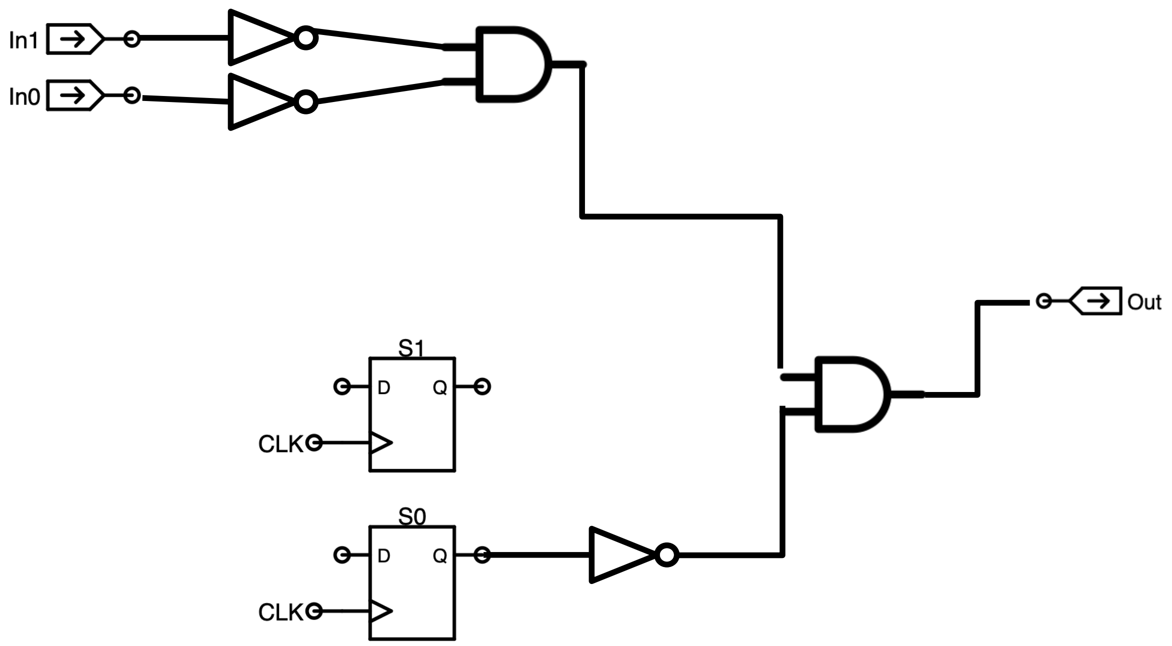
Next S1 $= \overline{S_1}\,\overline{S_0}\,\overline{I_1}\,I_0 + \overline{S_1}\,\overline{S_0}\,I_1\,I_0 + \overline{S_1}\,S_0\,\overline{I_1}\,I_0 + \overline{S_1}\,S_0\,I_1\,I_0 + S_1\,\overline{S_0}\,\overline{I_1}\,\overline{I_0} +$
$S_1\,\overline{S_0}\,\overline{I_1}\,I_0 + S_1\,\overline{S_0}\,I_1\,I_0$
$= \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}\,S_0\,I_0 + S_1\,\overline{S_0}\,\overline{I_1} + S_1\,\overline{S_0}\,I_1\,I_0$
Next S1 $= \boldsymbol{\overline{S_1}\,I_0 + S_1\,\overline{S_0}\,\overline{I_1} + S_1\,\overline{S_0}\,I_1\,I_0}$

Next S0 = $\boldsymbol{\overline{S_1}\,\overline{S_0}\,I_1\,\overline{I_0} + \overline{S_1}\,S_0\,\overline{I_1}\,\overline{I_0}}$

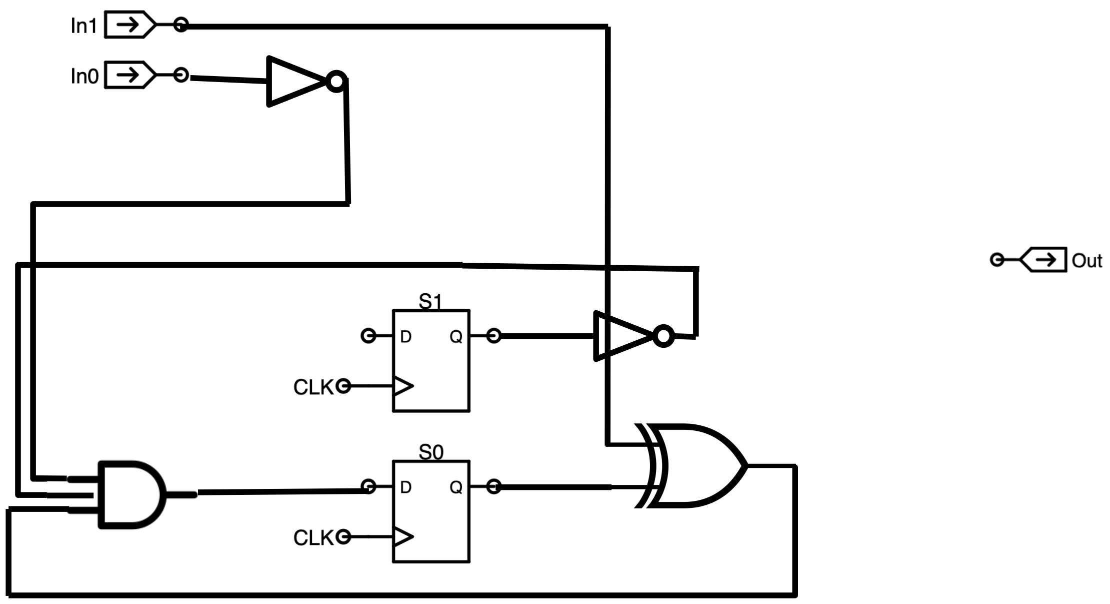
Out $= \overline{S_1}\,\overline{S_0}\,\overline{I_1}\,\overline{I_0} + S_1\,\overline{S_0}\,\overline{I_1}\,\overline{I_0} = \boldsymbol{\overline{S_0}\,\overline{I_1}\,\overline{I_0}}$

(E) (7 points) Given that the state of this FSM is encoded using 2 bits S_1S_0, and that the encoding for states F, S, and C are 00, 01, and 10 respectively, implement the FSM.

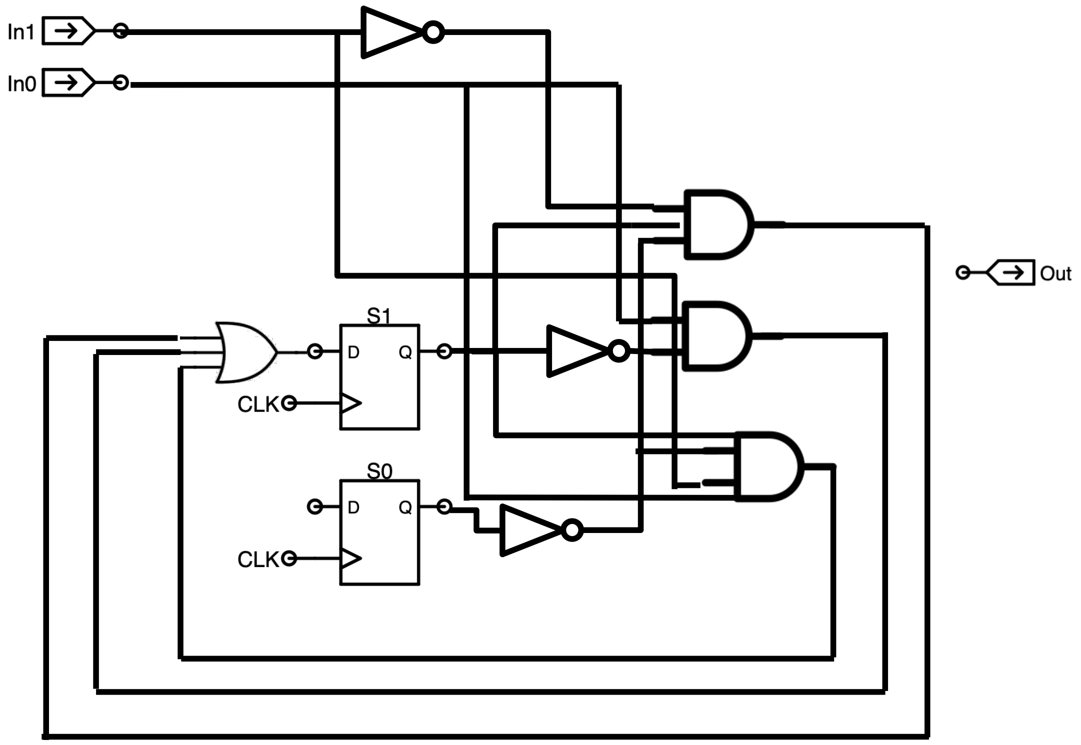
Circuit for Out



Circuit for S0



Circuit for S1



END OF QUIZ 1!