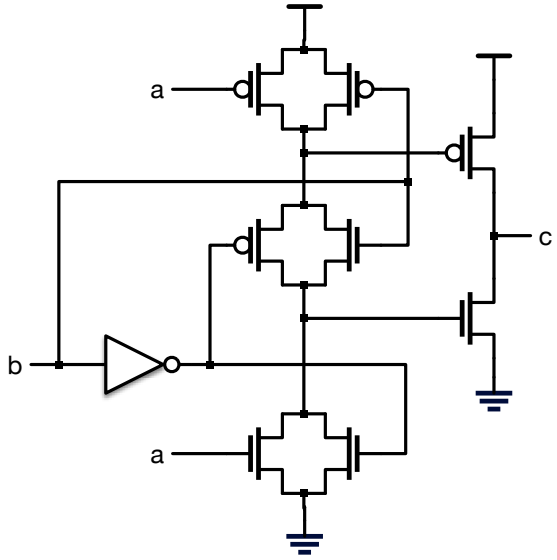
EECS 151/251A Homework 5

Instructor: Prof. John Wawrzynek, TAs: Christopher Yarp, Arya Reais-Parsi

Due Monday, Mar 4th, 2019

Problem 1: Identifying Functions [3 + 3pts]

The circuit shown below implements a common familiar function.



(a) What is the function? Describe the use of a, b, and c. [3 pts]

(b) Why does this implementation provide some advantage over the more common implementation of the same function? [3 pts]

Solution:

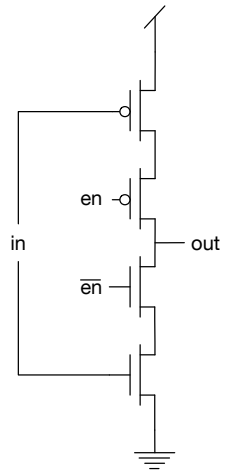
(a) This is a non-inverting tri-state buffer. Looking at the truth table,

a	b	c
0	0	Z
0	1	0
1	0	Z
1	1	1

b acts as an "enable" signal and a as the buffered signal. Whenever b is HIGH, the output is enabled and takes the value of a. The central PMOS/NMOS pair are both ON in order

to connect the output transistors' gates together, so that they take the same value. When b is LOW, the central PMOS/NMOS pair are both off. The pull-up transistors then pull the output PMOS gate up, disabling it, and the pull-down transistors pull the output NMOS gate down, also disabling it. Then c floats with only high-impedance paths to GND and V_{DD}.

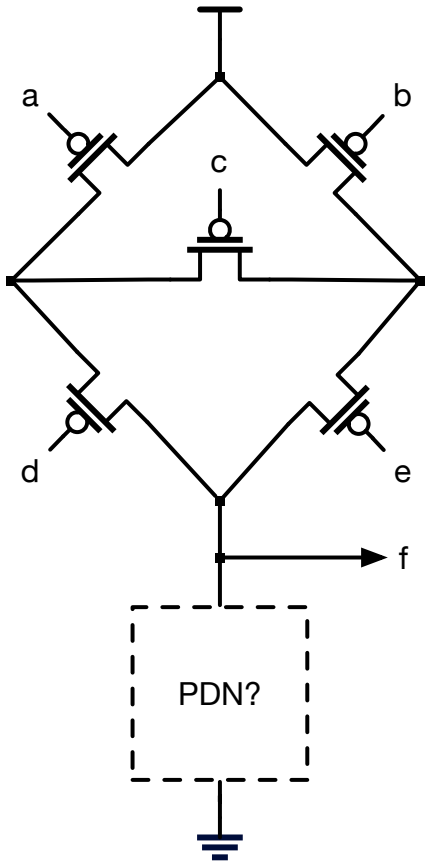
(b) The "traditional" tri-state buffer places four transistors in series.



To drive large outputs, each of these transistors has to be made wider. The design we have considered only requires us to widen two transistors at the output in order to drive the large load, not four, saving area. (For a given area, this design is faster.)

Problem 2: 251A only — *Optional* Challenge Question for 151 [5 + 2 pts]

The circuit below is an incomplete design of a complementary static CMOS logic gate.

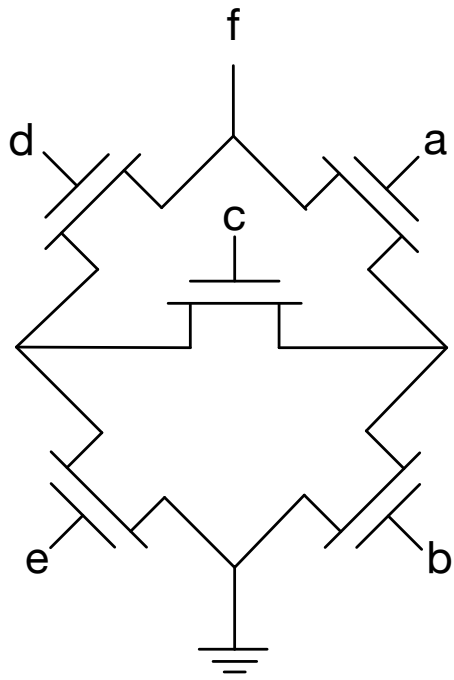


(a) Complete the circuit design by drawing the pull-down network. [5 pts]

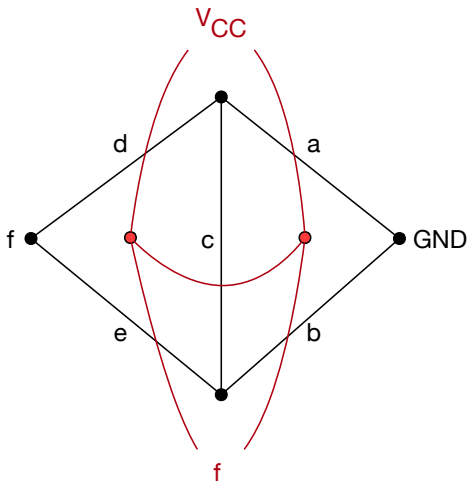
(b) What is the Boolean function this gate implements? [2 pts]

Solution:

(a) The PDN is:



The simplest way to derive the PDN from the PUN is to find the Euler path (which is similar to finding the dual of a graph representing either network, but with the outer space split into two), as below. For more information see Rabaey, Chandrakasan, Nikolic *Digital Integrated Circuits, A Design Perspective*, Second Edition, Design Methodology Insert D.



(b) From the PUN, read off the paths that result in f HIGH. Since the inputs to the PMOS transistors need to be LOW to turn them on, we treat the inputs as inverted:

$$f = \overline{a}\,\overline{d} + \overline{b}\,\overline{e} + \overline{a}\,\overline{c}\,\overline{e} + \overline{b}\,\overline{c}\,\overline{d}$$

From the PDN:

$$\begin{aligned}
\overline{f} &= de + ab + ace + dce \\
f &= \overline{de + ab + ace + dce} \\
&= \overline{de}\,\overline{ab}\,\overline{ace}\,\overline{dce} \\
&= (\overline{d} + \overline{e})(\overline{a} + \overline{b})(\overline{a} + \overline{c} + \overline{e})(\overline{d} + \overline{c} + \overline{e}) \\
&= \ldots \text{(expand and simplify)} \ldots \\
&= \overline{a}\,\overline{d} + \overline{b}\,\overline{e} + \overline{a}\,\overline{c}\,\overline{e} + \overline{b}\,\overline{c}\,\overline{d}
\end{aligned}$$

Problem 3: Transistor Level Implementation of Logic Gate [4 pts]

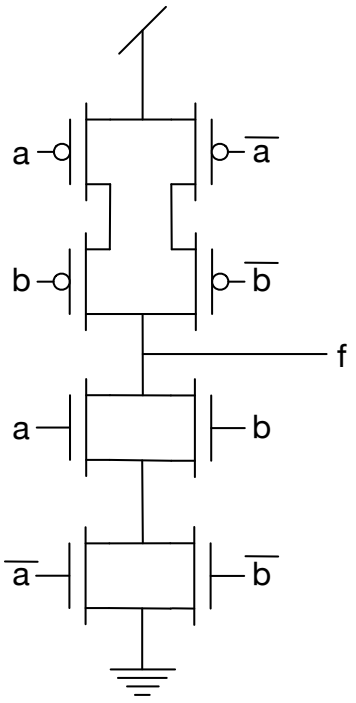
Exclusive-nor (XNOR) is defined as the complement of exclusive-or. Draw a complementary static CMOS gate that implements XNOR(a,b) with the minimum number of transistors. You may assume that both inputs a and b are available in uncomplemented and complemented forms.

Solution:

The truth table for exclusive-NOR is:

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

From which we derive that the boolean function is $f = ab + \overline{a}\,\overline{b}$. Assuming that inverted inputs are available (if they aren't we can use a simple inverter to acquire them), an implementation is:



Problem 4: Transistor Level Implementation of Logic Function [6 pts]

Assuming the inputs are available in uncomplemented and complemented forms, draw a complementary static CMOS gate that implements the majority function of three inputs a, b, and c, using the minimum number of transistors.

Solution:

The majority function outputs one whenever the *majority* of inputs are 1. For three inputs a, b and c its truth table is:

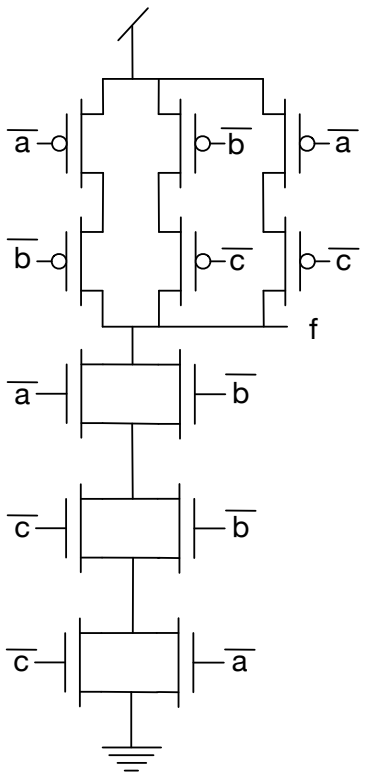
a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

It's straightforward to write the boolean expression you'd expect: $f = ab + bc + ac$. From this

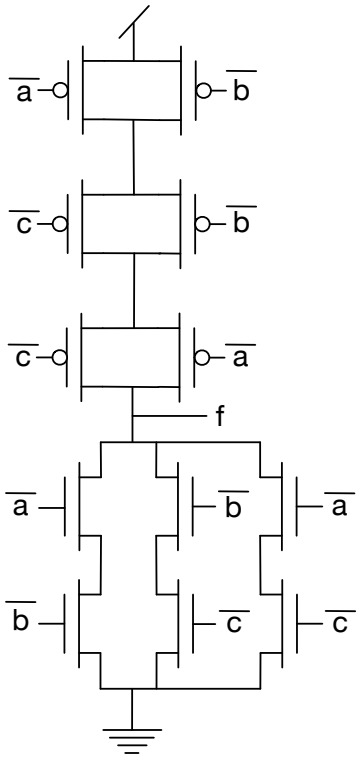
we can derive an expression for the or the PUN. For the PDN in this case, it's:

$$\begin{aligned}\overline{f} &= \overline{ab + bc + ac} \\ &= \overline{ab}\,\overline{bc}\,\overline{ac} \\ &= (\overline{a} + \overline{b})(\overline{b} + \overline{c})(\overline{a} + \overline{c})\end{aligned}$$

Which yields:



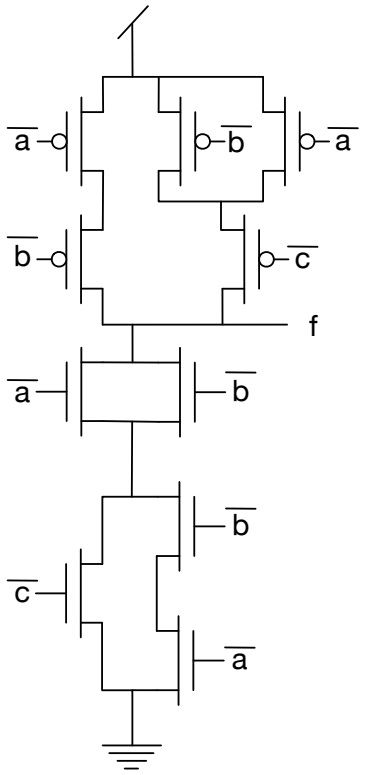
Note that at this point there is symmetry in how you could implement the PDN. An alternative solution is:



To reduce the number of gates used, factor out of one the terms in the first expression $f = ab + bc + ac = ab + c(b + a)$. This makes it easy to deduce the PUN. Deriving the PDN from scratch is a little more work:

$$\begin{aligned}
\overline{f} &= (\overline{a} + \overline{b})(\overline{b} + \overline{c})(\overline{a} + \overline{c}) \\
&= (\overline{a} + \overline{b})(\overline{b}\,\overline{a} + \overline{b}\,\overline{c} + \overline{c}\,\overline{a} + \overline{c}) \\
&= (\overline{a} + \overline{b})(\overline{b}\,\overline{a} + \overline{c}(\overline{b} + \overline{a} + 1) \\
&= (\overline{a} + \overline{b})(\overline{b}\,\overline{a} + \overline{c}(1)) \\
&= (\overline{a} + \overline{b})(\overline{b}\,\overline{a} + \overline{c})
\end{aligned}$$

Either way, this removes a transistor from each of the PDN and PUN:



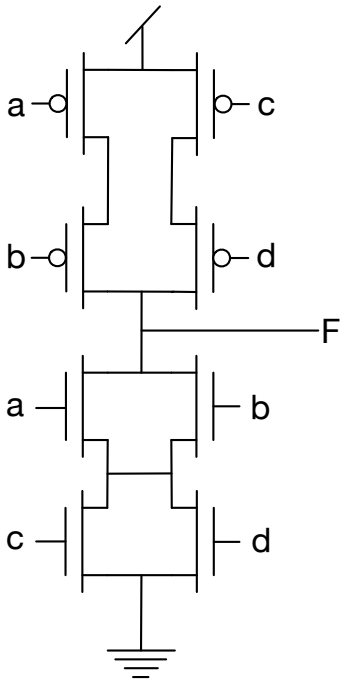
Problem 5: Transistor Level Implementation of Logic Function [4 pts]

Assuming that only uncomplemented inputs are available, draw a complementary static CMOS gate that implements $F = (a + b)' + (c + d)'$, using the minimum number of transistors.

Solution:

$$F = \overline{a + b} + \overline{c + d}$$
$$\overline{F} = (a + b)(c + d)$$

gives us the PDN. We can derive the PUN by identifying the Euler paths or, in this simple case, by converting parallel gates to series ones. The static CMOS gate is thus:



Problem 6: Transistor Level Implementation of Logic Function [5 pts]

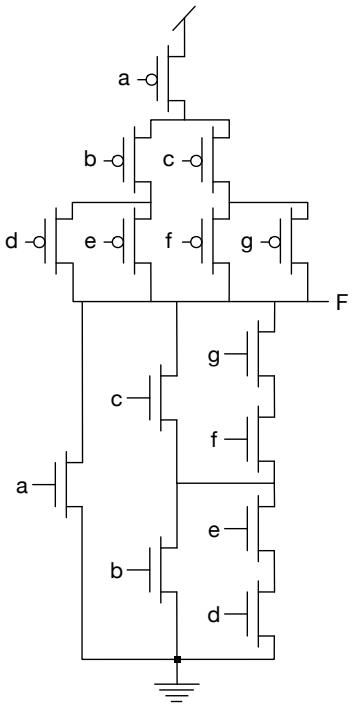
Assuming that only uncomplemented inputs are available, draw a complementary static CMOS gate that implements $F = a'b'd' + a'b'e' + a'c'f' + a'c'g'$, using the minimum number of transistors.

Solution:

Develop the pull-*up* network by factoring the boolean expression for F:

$$\begin{aligned} F &= \overline{a}\,\overline{b}\,\overline{d} + \overline{a}\,\overline{b}\,\overline{e} + \overline{a}\,\overline{c}\,\overline{f} + \overline{a}\,\overline{c} + \overline{g} \\ &= \overline{a}\,(\overline{b}\,\overline{d} + \overline{b}\,\overline{e} + \overline{c}\,\overline{f} + \overline{c}\,\overline{g}\,) \\ &= \overline{a}\,(\overline{b}\,(\overline{d} + \overline{e}\,) + \overline{c}\,(\overline{f} + \overline{g}\,)) \end{aligned}$$

With complementary PDN, the circuit is:

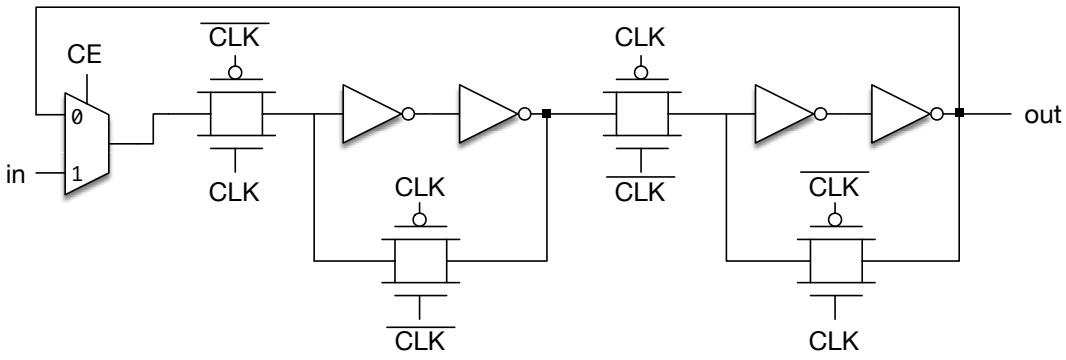


Problem 7: Pass-Transistor Implementation of Flip-Flop with CE [4 pts]

Draw the CMOS positive-edge triggered Flip-flop implementation shown in class (lecture 9) as we did using inverters and transmission gates. Show how you would modify it to include the clock enable (CE) input. You may use additional transistors and gates as needed.

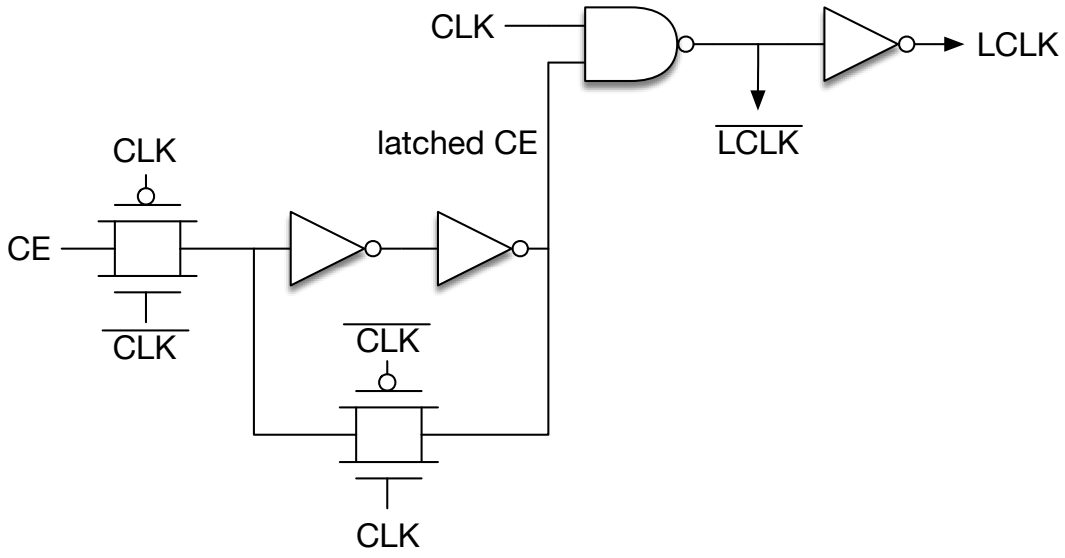
Solution:

Use the clock-enable signal to gate the input to the first latch. If CE is LOW, the input from the flip-flop output is fed back into the input. This method prevents an unintended state change if CE goes HIGH midway through CLK being HIGH.



An alternative solution is to gate CLK with CE, generating LCLK and $\overline{\text{LCLK}}$ to use in place of

CLK throughout the rest of the flip-flop. However, in order to prevent spurious state changes, CE must first be latched:

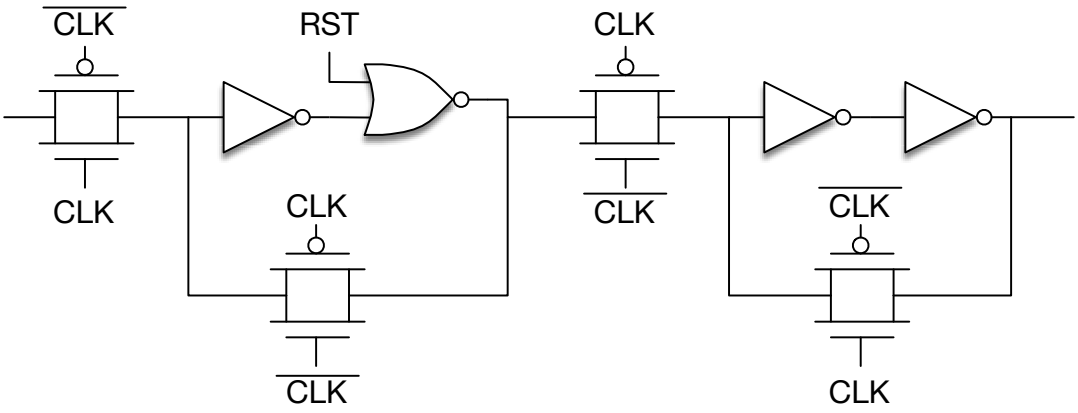


Problem 8: Pass-Transistor Implementation of Flip-Flop with rst [4 pts]

Draw the CMOS positive-edge triggered Flip-flop implementation shown in class (lecture 9) as we did using inverters and transmission gates. Show how you would modify it to include the reset (rst) input. You may use additional transistors and gates as needed.

Solution:

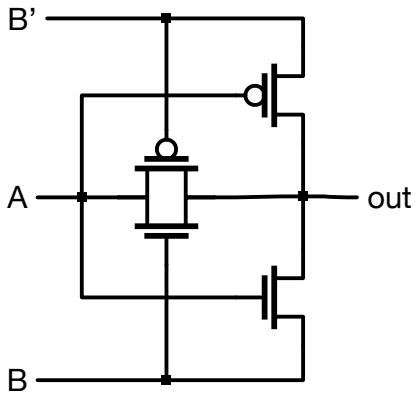
The reset signal, RST, inserts into the first of the two latch stages to force the value between the latches to 0, irrespective of the input:



An alternative solution is to use the input multiplexor shown in the solution problem 7, but with RST used as the select input and the RST = 1 input tied to 0.

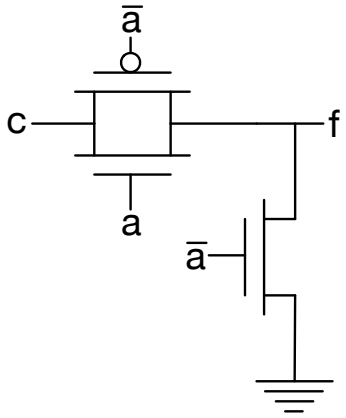
Problem 9: 251A only — *Optional* Challenge Question for 151 [6 pts]

Shown below is a pass-transistor-logic version of a XNOR gate (note this is not a complementary static CMOS gate). Make sure you understand the operation of this gate, then use the concepts in the XNOR gate to design a pass-transistor version of an AND gate implementing AND(A, B).



Solution:

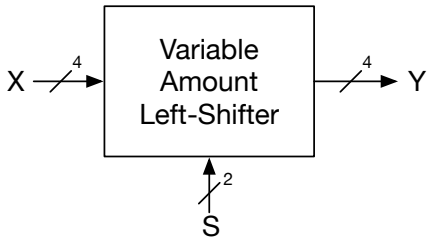
The $f = \text{AND}(a, b)$ function can be considered the gating of one signal by the other. Use a transmission gate that's ON only when $a = 1$, and a second NMOS transistor to pull the output LOW the transmission gate is OFF:



An alternative way to construct this is as a 2-1 multiplexor with one input constant and the other c, where a becomes the selection signal (or vice versa).

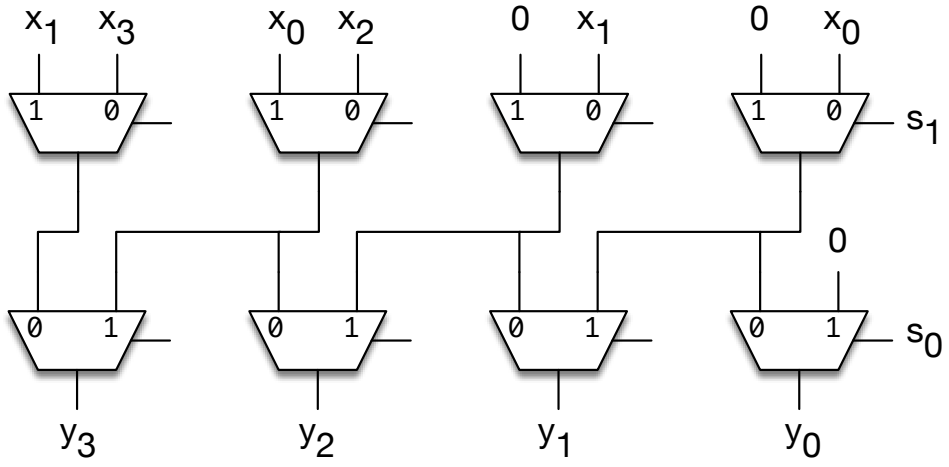
Problem 10: Transistor Level Implementation of Shifter [10 pts]

Using only transistors, show how you would implement a combinational logic circuit for a variable left-shifter as shown below. X is a 4-bit input, Y is a 4-bit output, and S is a 2-bit control signal that indicates a shift amount of 0, 1, 2, 3 bit positions. Design your circuit to minimize the delay X to Y.



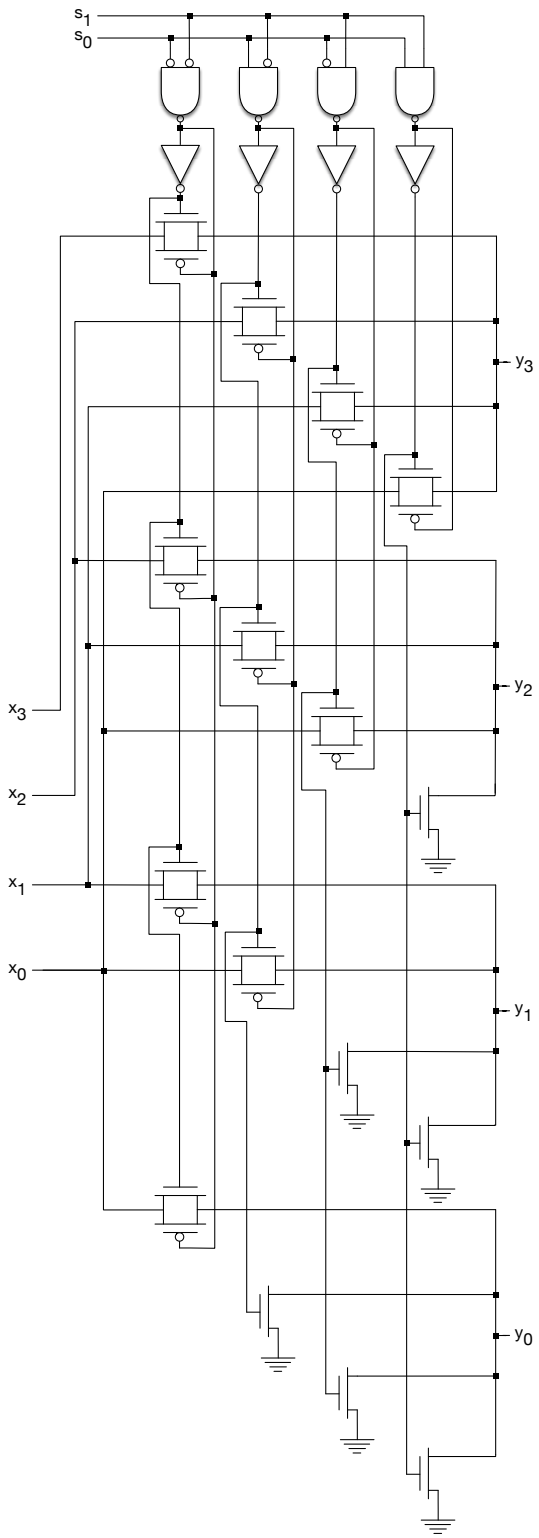
Solution:

The suboptimal solution is to use two layers of 2-to-1 muxes, each for one of the selection bits s_1 and s_0:

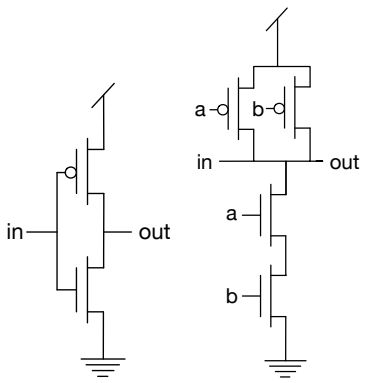


However, this places two transmission gates between the inputs and the outputs, adding unnecessary delay.

The better (faster) solution is to build a 4-to-1 multiplexor for each signal directly, using pass gates to minimise delay as shown in lectures. In this solution there is only 1 transmission gate between each input and the output:



The CMOS inverter and NAND gates shown are the familiar ones:



(Note that this solution also leaves room to optimise away gates where we have fixed 0-inputs.)