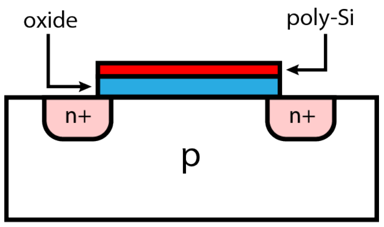
EECS 151/251A Homework 5

Due Monday, Mar 1$^{\text{th}}$, 2021

Please include a short (1-2 sentence) explanation with your responses to each question unless otherwise directed.

Problem 1: Basic IC Processing

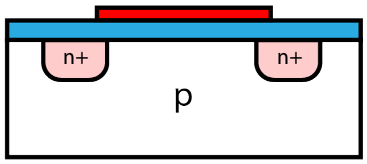
In modern IC fabrication, metal layers are commonly added using a process called *Chemical Vapor Deposition (CVD)* (Wikipedia Article). Starting from the finished transistor below, describe the fabrication steps to manufacture the first metal layer, contacting the terminals of the transistor. The blue layer represents the oxide.



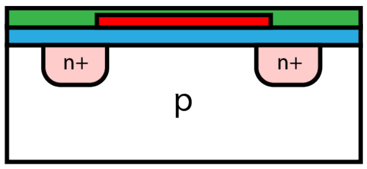
Solution:

Diagrams are not required. These are just to illustrate the steps taken in the manufacturing process.

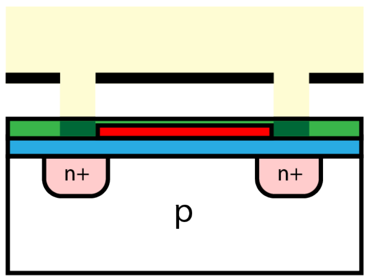
1. Grow oxide



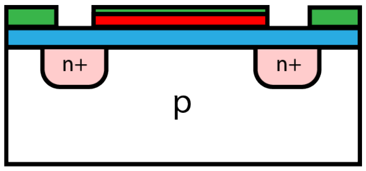
2. Apply photoresist



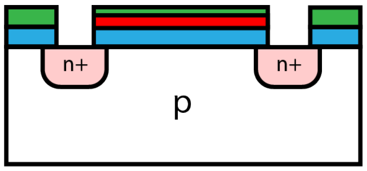
3. Expose photoresist with masks. Mask should be patterned to remove PR over Source/Drain diffusion areas



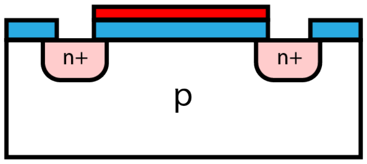
4. Develop photoresist



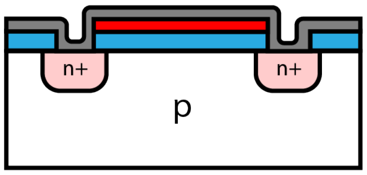
5. Etch oxide with BOE (buffered HF solution)



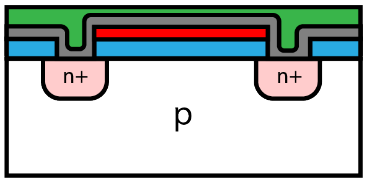
6. Strip photoresist



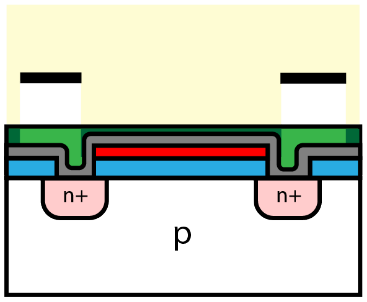
7. Deposit metal with CVD



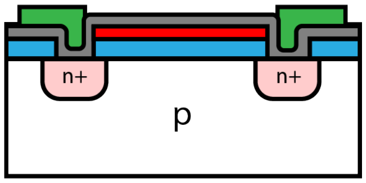
8. Apply photoresist



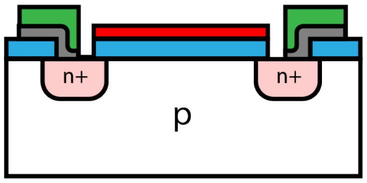
9. Expose photoresist with masks. Mask should be patterned to remove PR over everything except the intended Source/Drain contacts



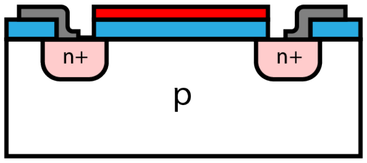
10. Develop photoresist



11. Etch metal with strong acid (usually Sulfuric or Nitric)

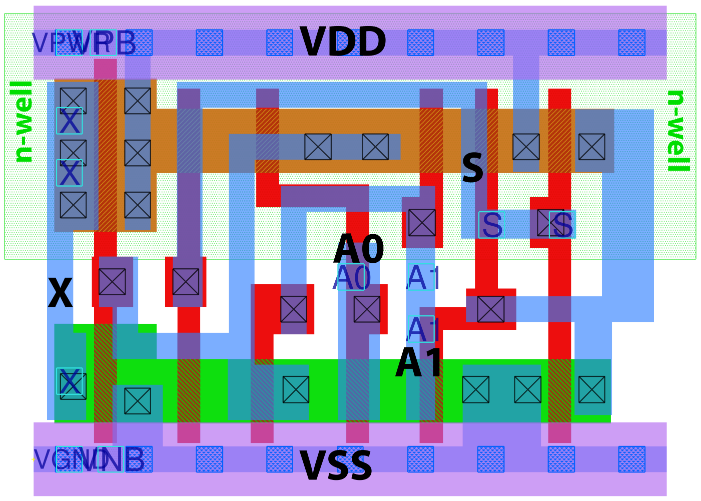


12. Strip photoresist



Problem 2: Standard Cell Layout Reverse Engineering

Take a look at the standard cell layout below from the SkyWater 130 nm open-source PDK. The supply rails, pins, and n-well have been labeled. What function does it implement? Provide a schematic at the transistor level.



Solution:

Following the rules of thumb (each diffusion-poly-diffusion = 1 transistor), we can identify the devices in the layout as such, with corresponding connections.

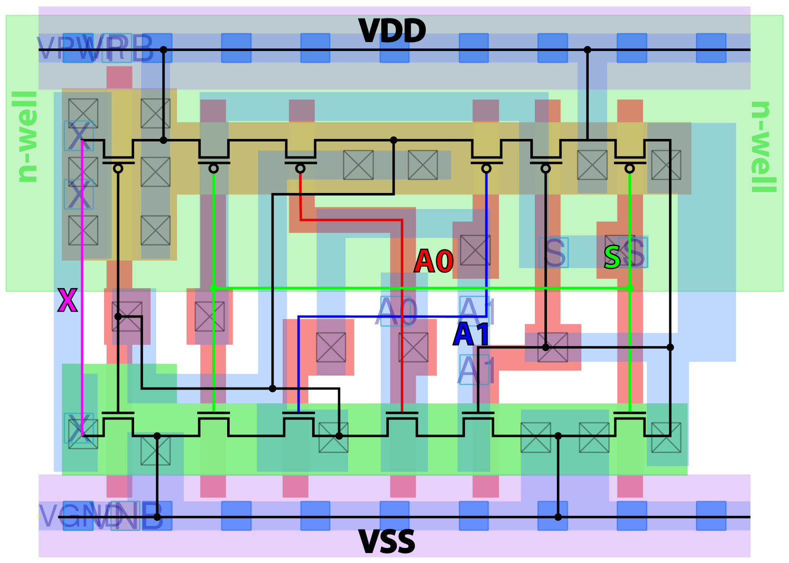


Figure 1: Annotated Standard Cell Layout

Redrawing this as a CMOS schematic, we see this connection

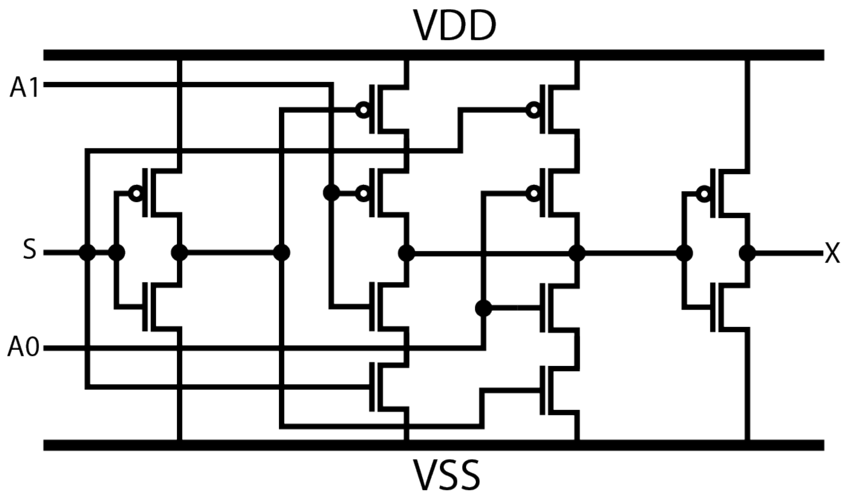


Figure 2: Extracted Standard Cell CMOS Circuit

This is basically two C^{2}MOS inverters. One of them is enabled for A0 when S=0, and the other is enabled for A1 when S=1. Both inverters then drive a final inverter to output X. There is an additional inverter at the front that provides the S' input to the complementary enable gate for each C^{2}MOS inverter.

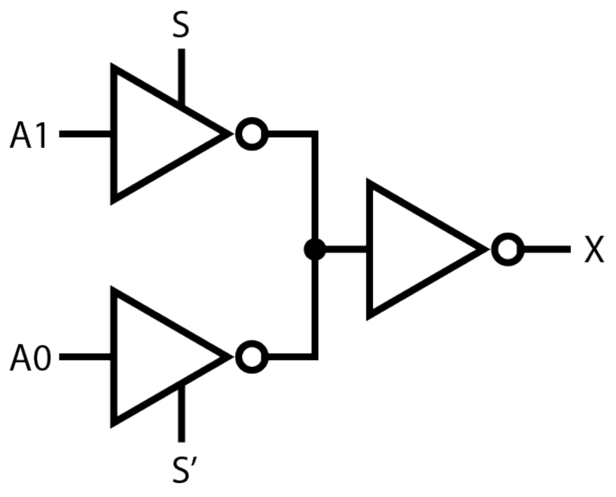


Figure 3: Simplified Standard Cell CMOS Circuit

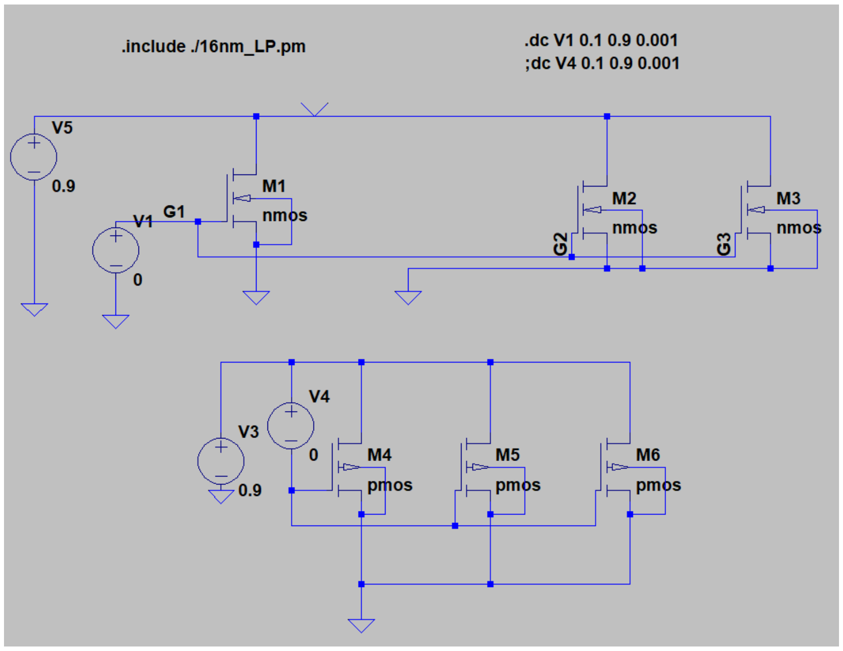
From this we can see that this block implements a **2-to-1 multiplexer**.

Problem 3: MOS Characteristics

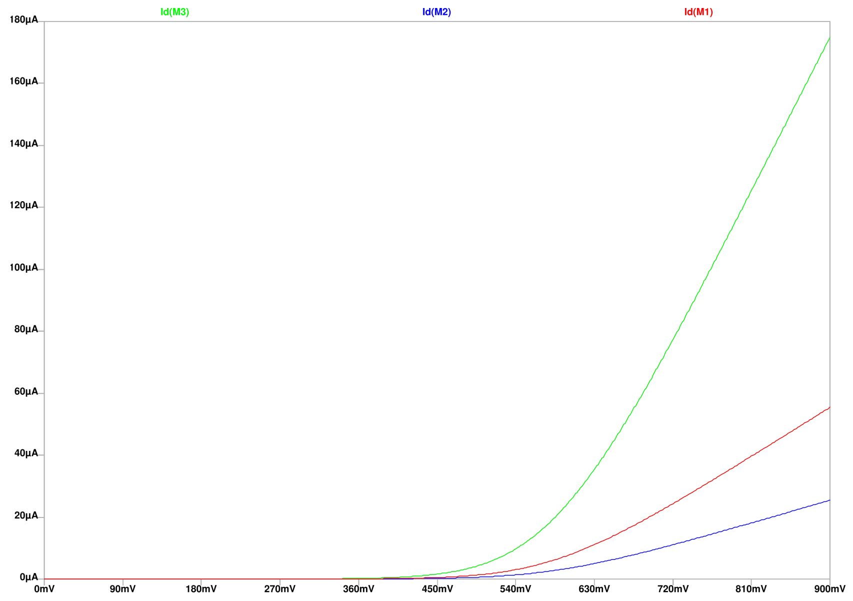
Using the same 16nm predictive LTSpice model for transistors as Homework 1, set up a circuit to measure I_{DS} vs. V_{GS} with $V_{DS} = 0.9\,V$ for both NMOS and PMOS. Start with both devices at 0.1 µm wide. Repeat for $W = 0.05\,\mu m$ and $W = 0.3\,\mu m$. Plot the three current vs. voltage curves on one waveform for each MOS and submit it along with a screenshot of your schematic.

Solution:

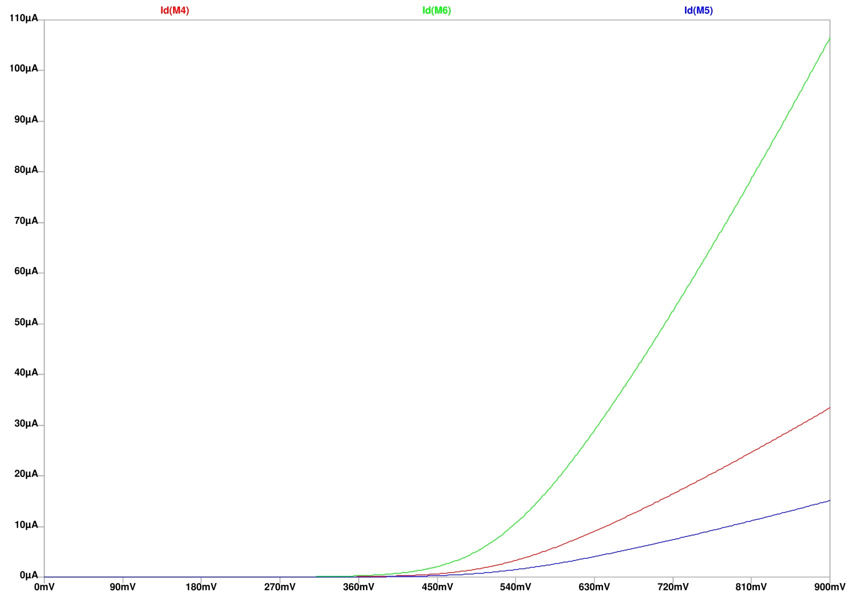
Schematic Setup



NMOS Currents



PMOS Currents

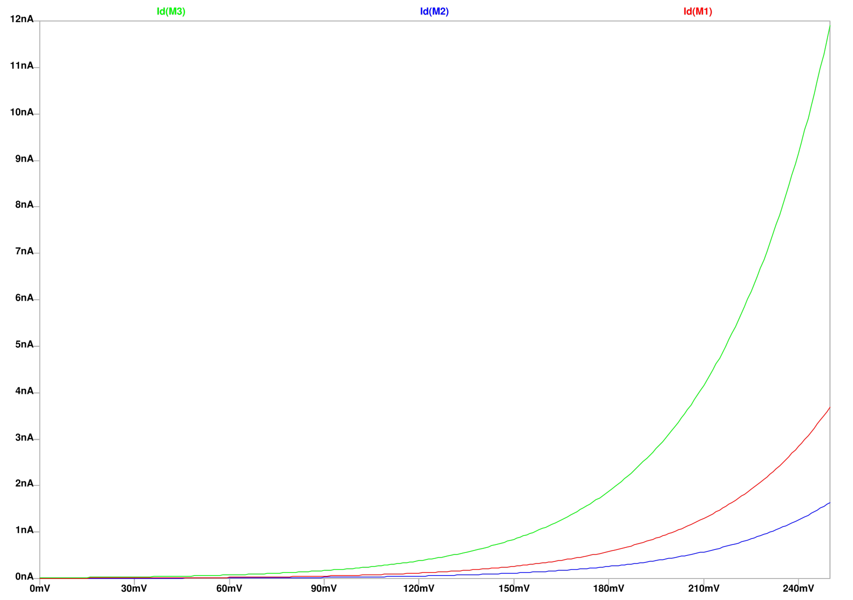


251A only — *Optional* Challenge Question for 151

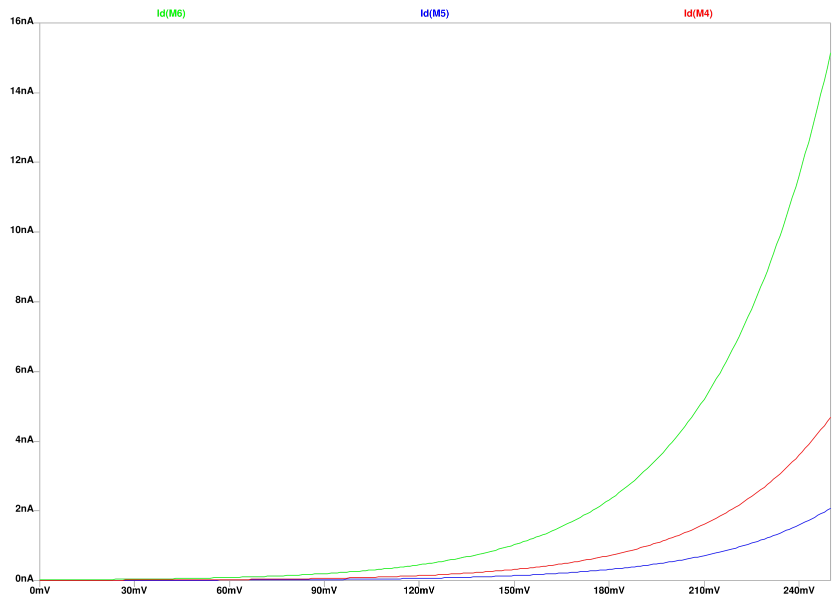
In MOSFETs, there is a region of operation known as *subthreshold*. From your results in Problem 3, choose new voltage sweep endpoints to demonstrate this region of operation and plot the drain currents for the same devices as Problem 3. Does the current in the MOSFET still follow the square law relationship with the gate voltage in this region? Turn in screenshots of your schematic setup and waveforms.

Solution:

The subthreshold region for these devices is up to approximately 0.25 V, so plot the currents for $V_{GS} < 0.25$ V. NMOS Currents



PMOS Currents



Note that the current appears to increase exponentially rather than as the square of V_{GS}. This is expected behavior in the subthreshold region as when the device is biased in this region, it appears like a bipolar transistor, which exhibits exponential transconductance.

Problem 4: CMOS

Build a complex CMOS gate that performs the following function:

$$y = (a'b' + c')(d + e)'$$

Turn in a transistor-level schematic of your CMOS gate. You may **not** use complemented inputs.

Solution:

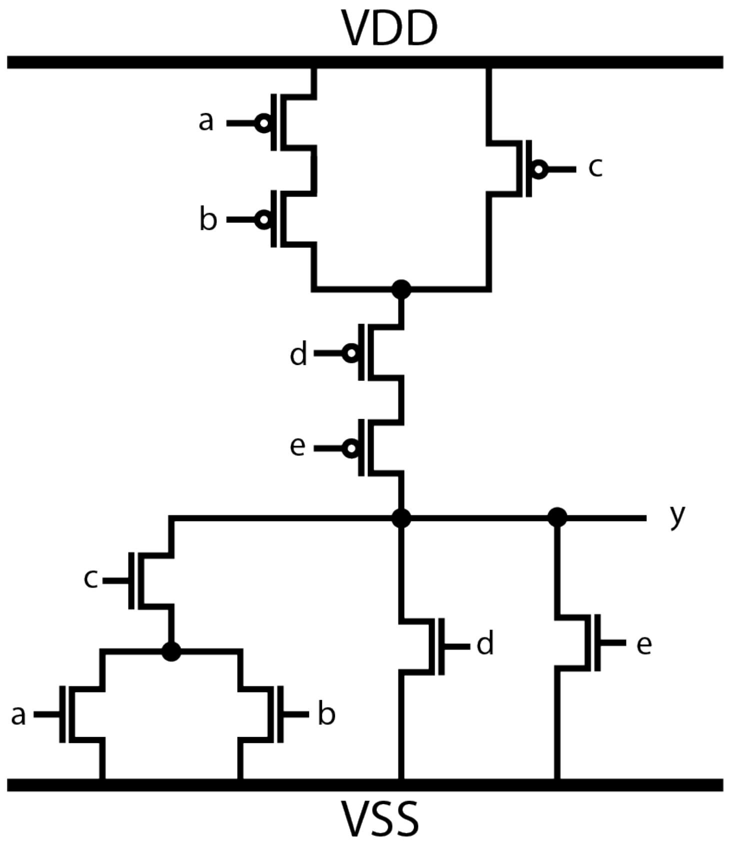
This problem is easiest to approach by applying De Morgan's law on the $(d + e)'$ term and applying this to the pull-up network, then designing the pull-down network as the dual.

$$\begin{aligned} y &= (a'b' + c')(d + e)' \\ &= (a'b' + c')(d'e') \end{aligned}$$

Alternatively, you could also find the complement and design the pull-down network first.

$$\begin{aligned} y' &= ((a'b' + c')(d + e)')' \\ &= (a'b' + c')' + (d + e) \\ &= ((a'b')'c) + d + e \\ &= (a + b)c + d + e \end{aligned}$$

With this expression, we can now design the pull-down network, then take the dual of it to get the pull-up network, resulting in this final CMOS circuit



Problem 5: Transmission Gate Logic

A *rotator* is a circuit that takes a set of input bits and rotates them in either direction by some number of bits, configurable with another input. For example, a 3-bit rotator could rotate the bits either right or left by 0, 1, or 2 bits (3 bit rotation would be the same as a 0 bit rotation). Design a 4-bit rotator using transmission gate logic, using only 1 transmission gate per input/output path. The rotator will always rotate to the **right** (for example, 0110 will be come 0011 with a 1-bit rotation). *(Hint: Note that the number of possible rotations is equal to the number of input bits. This lends itself well to an NxN grid of circuit elements to implement the function.)*

Solution:

A 4x4 grid of transmission gates will cover all possible combinations of input bit to output bit for the rotations. These gates can then be connected in such a way that each rotation will rotate the bits some number of places, which can then be selected by decoding the control bits. In the figure below, I have color-coded the enable signals responsible for each shift to make it easier to identify the rotation.

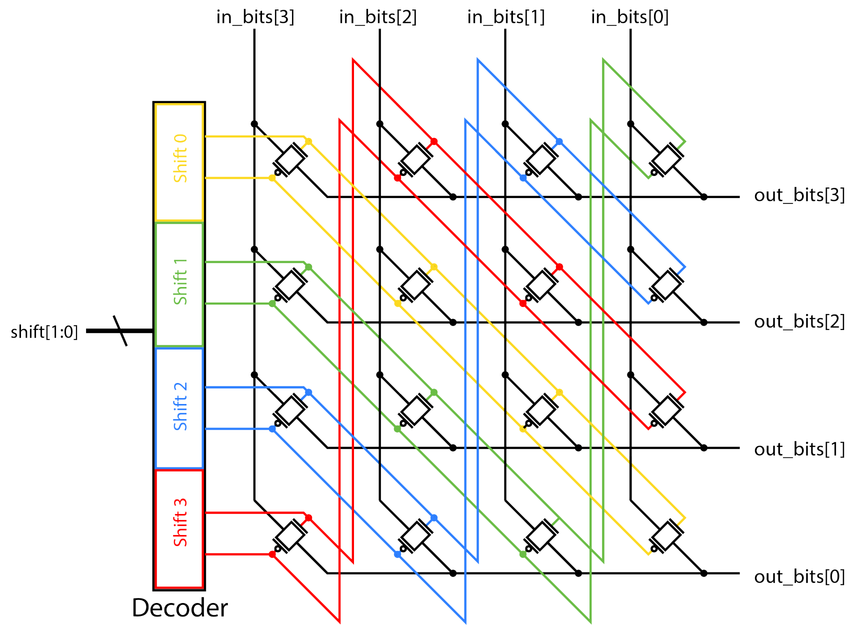
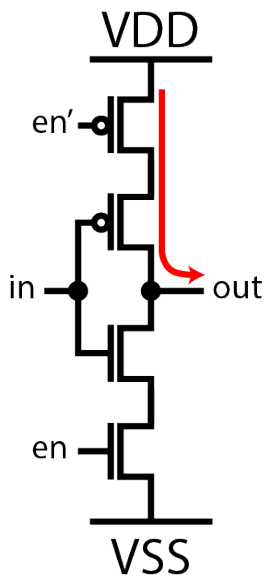


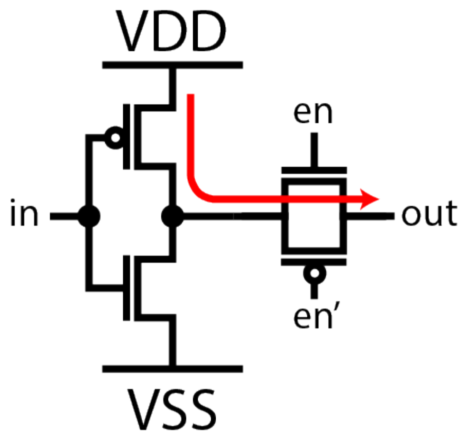
Figure 4: Transmission Gate Rotator

Problem 6: Tri-State Buffer

In the lecture, you learned about tri-state buffers and how they are made with either a transmission gate or C^{2}MOS (stacking CMOS). Both of these approaches use two devices in series (transmission gate uses the driving static gate's devices as well as the transmission gate device itself).



(a) C^{2}MOS Buffer



(b) Transmission Gate Buffer

Current paths in both tri-state buffer topologies. Note the two series devices.

There is another way to implement a tri-state buffer, with only one device along the path from either supply rail to the output. Design such a buffer and provide a circuit schematic. Explain its operation. In what situation would such a circuit be preferred over the classic way of making tri-state buffers?

Solution:

The operation of tri-stating the output is simply to provide no closed path from output to either supply. We can accomplish this by decoupling the pull-up and pull-down networks of an inverter from each other and driving each separately. We now need to introduce some additional logic to selectively enable the output, which results in the following schematics.

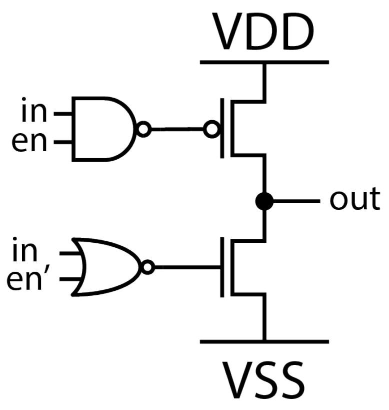


Figure 6: Non-inverting Tri-State Buffer

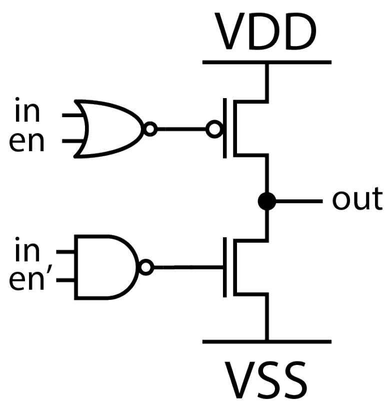


Figure 7: Inverting Tri-State Buffer

There is one more potential solution using transmission gates and some clever CMOS circuitry.

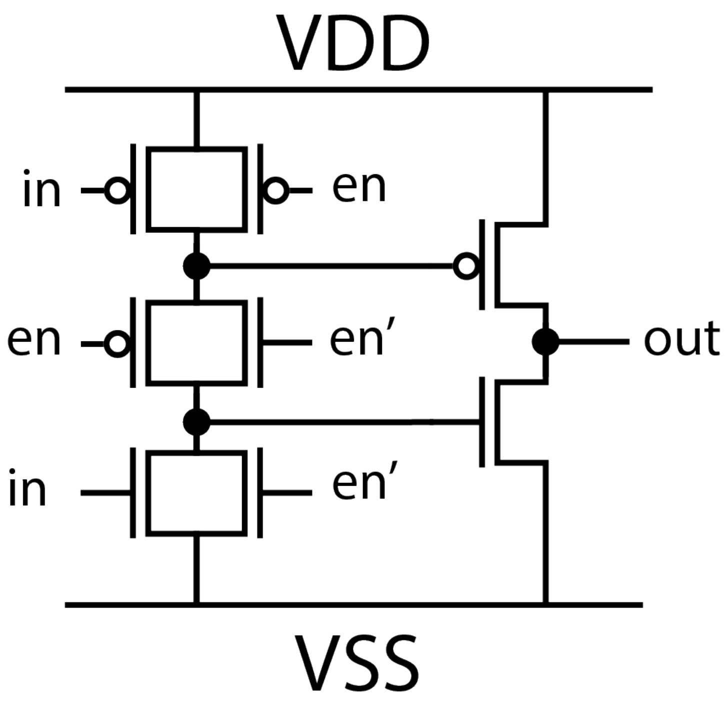


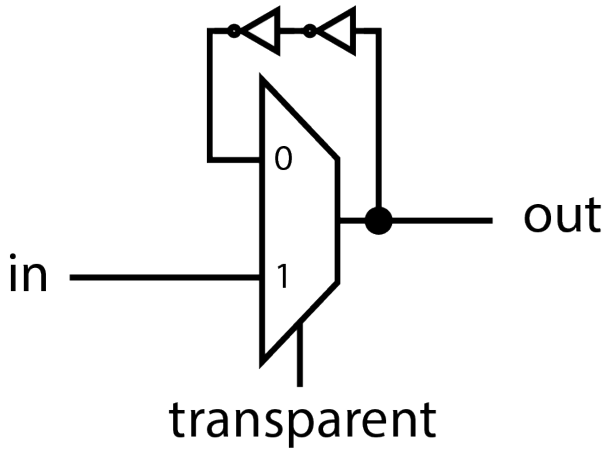
Figure 8: Transmission Gate-based Tri-state Buffer

These topologies are useful as the driving devices can be minimum-sized and will not suffer from the increase in delay from having two devices in series from the supply to the output.

Problem 7: Latch

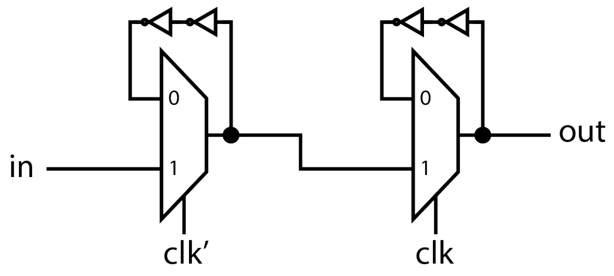
A latch is a circuit that holds its output steady when an input control signal is either high or low. When the control signal allows the latch to update to its data input, we call the latch *transparent* to the input. A positive latch will be transparent when the control signal is high, and a negative latch will be transparent when the control signal is low. For a positive latch, the latch will remain transparent as long as the control signal is high, and vice-versa for the negative latch. Once the control signal changes, the latch will then ignore its input and retain whatever its last output was until the control signal requests the latch become transparent again. One way to implement such a latch is with multiplexers. Design a positive latch based around a 2-MUX and provide the circuit schematic (you may abstract the mux itself as just a block, but the rest of the circuit should be transistor-level). How might such a circuit be used to design a positive edge-triggered flip-flop?

Solution:



Note that there are two inverters in series in the feedback path. If the multiplexer does not have regeneration built-in (that is if it is implemented using only transmission gates), you must have some regenerative elements in the feedback path to preserve the value in the latch. Otherwise, the value in the latch will decay over time due to leakage in the devices, so the latch would not be able to hold a value for very long.

To make a flip-flop, we could then put two of these latches in a cascade, with the first latch transparent on `clk’` and the second transparent on `clk`. This way, the first latch will continuously update its value until the rising edge, where it will latch and hold its output. The second latch will block the output from updating until after the clock edge, thereby implementing the edge-triggered behavior.


0
1
0
1
in
out
clk'
clk