ECE/CS 5710/6710 – Digital VLSI Design
Lab Assignment #3

**Due Thursday Oct. 2nd via canvas**

# 1 Introduction

In this assignment you will design a register cell. This cell should be a single-bit edge-triggered D-type flip flop that you could use to make a larger register bank. For example, you might want to put 8 of these together to make an 8-bit register, and then put 16 of those 8-bit registers together to make a 16-word, 8-bit wide register bank. Eventually you will use this cell, or something similar to it, in the data path of your project where you need registers in your design. However, don't stress out about getting it perfect in this assignment. You will probably want to tweak some things, and likely even redesign it completely, later once you have a better view of the whole data path and have some more experience doing layout.

Note also that you should still be working on things individually! We'll form groups for Lab number 5.

The inputs to your register are D, CLK, and CLRB. The outputs are Q and QB. The flop is non-inverting – a 1 on D will result in a 1 on Q and 0 on QB after the clock. The register will be rising-edge triggered. Therefore the data will be captured and presented to the output on the rising edge of the CLK. The CLRB signal is synchronous and active low.

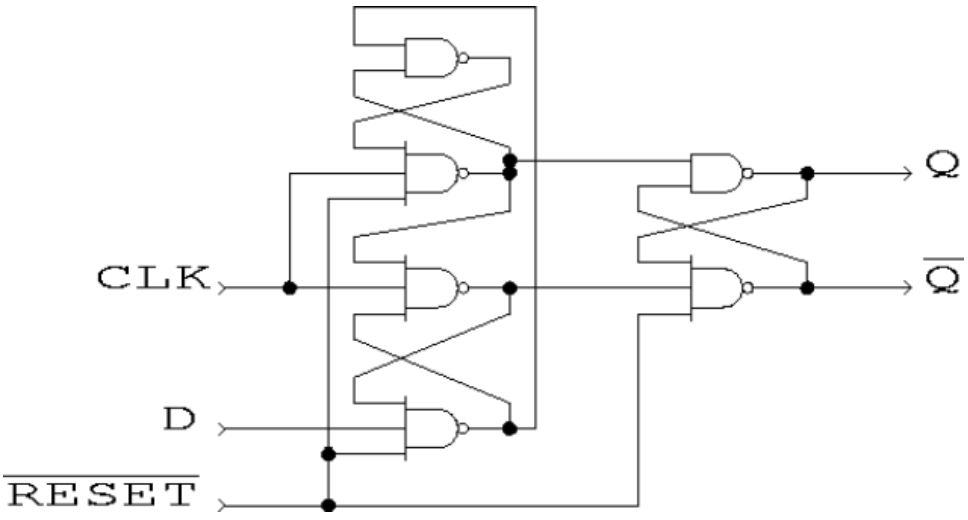


Figure 1: Positive Edge Triggered Flip Flop using Gates

# 2 Background

There are many ways to build a positive edge triggered D flip flop! You can see quite a few ways in our textbook. See Section 1.4.9 and Section 7.3 in *Principles of CMOS VLSI* for some examples. You can also look in other books and articles to find lots of other circuits and variations on the basic circuits. Which one is best? It depends! It depends on the application, the constraints of the implementation, etc. For example, a popular design at the NAND-gate level that you see in many textbooks is shown in Figure 1. This is a fine implementation. It has an active-low reset signal that resets the flip flop to 0, and it is robust in regard to restricting the sampling of the D signal to the rising clock edge. However, note that if you implemented this by directly translating the NAND-gates to CMOS, it would be huge! As a result, you probably *don't* want to use this design as your fundamental register element.

The design of a flip flop that applies transistors in a more "custom" way is to use a pair of latches. These latches are designed using a pair of inverters for the feedback, and a couple of transmission gates (pairs of N- and P-type transistors) to switch between transparent and opaque modes for the latch. This type of simple gated latch is shown in Figure 2 (also see Figure 1.12 in your book).

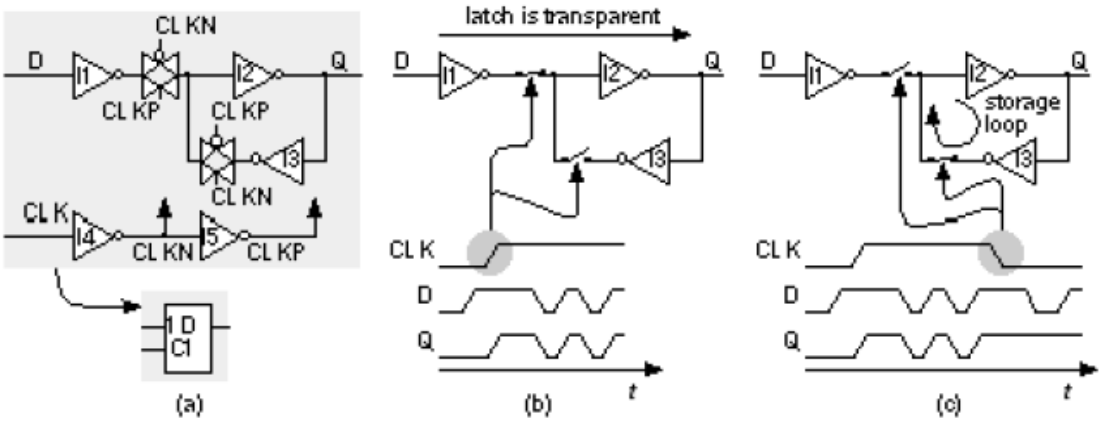


Figure 2: Latch using transmission gates

The gated latches can also be implemented by merging the transmission gates with the inverters that are driving them to make a single CMOS gate that functions as a "enabled inverter" or "tri-state inverter". This circuit combines the functionality of the inverter and pass gate into a single gate with a 2-high PMOS and NMOS stack resulting in a smaller layout. This tri-state inverter will drive its the complement of the input onto the output when the enable lines are active, and will be in a high impedance mode not driving any signal onto the output when the enable signals are inactive. The reason that the input to the pass gate can be disconnected at its input, as shown in Figure 3, is that the P-type transistor

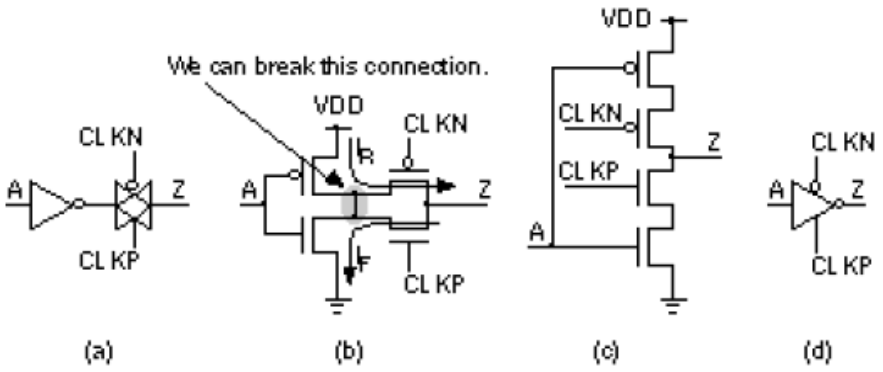


Figure 3: Tristate Driver

of the transmission gate is primarily involved in pulling the output high, and the N-type of the transmission gate is primarily involved in pulling the output low. By breaking the input connection of the pass gate and folding the P-type into the pull-up stack and the N-type into the pull-down stack the two gates exclusively pull the output high and low respectively and still create the logic steering and tristate functionalities of the independent gates.

We can use the gated latch from Figure 2 (modified to use tristate drivers from Figure 3) to build an edge triggered flip flop as shown in Figure 4. In this circuit the D Flip-Flip is formed using transmission gates and tristate drivers (see Figure 1.32 in the text for a different view of this circuit, but without the clear). The reset is accomplished by using NAND gates instead of inverters for part of the feedback path. This is technically a Master-Slave device rather than a "true" edge triggered device, but in practice the result is the same: the D signal is sampled on the rising edge of the clock signal. Implementing your register bit with this circuit would result in *many* fewer transistors than the circuit of Figure 1.

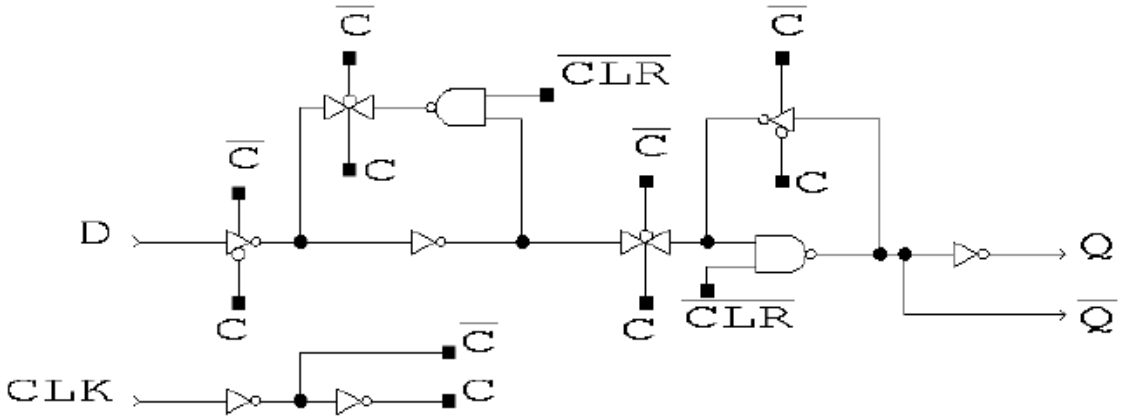


Figure 4: Positive edge triggered flip-flop using CMOS structures

Another variation on this theme of building a gated latch and then using the latch to build a master slave flip flop is shown in Figures 5 and 6. This latch is used in some of the PowerPC chips. In particular it was used in a version of the PowerPC that was built in a process similar to ours. If you use this circuit you will have to figure out where the clear (reset) transistors go.

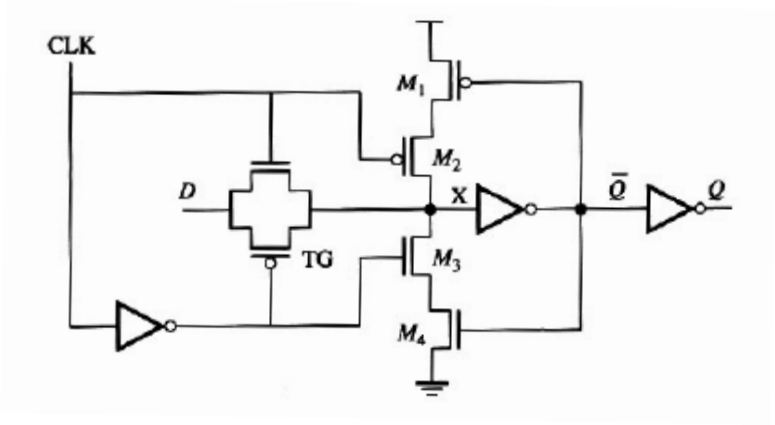


Figure 5: Alternative gated latch

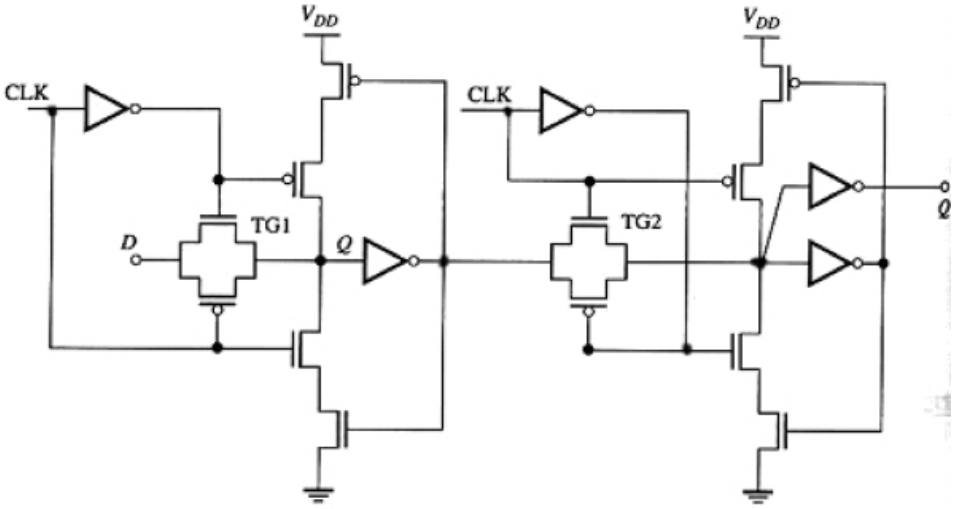


Figure 6: Alternative master slave flip-flop

This assignment is flexible: your register bit can be designed however you like as long as it has the right functionality: a positive edge triggered flip flop with an active-low clear signal. However, you probably want a design that looks more like Figure 4 or 6 than Figure 1! You can make an even more compact circuit using dynamic or pseudo-dynamic logic gates that we haven't talked about, or by using two-phase or other clocking schemes. These are not

recommended because they're sensitive to noise or have more complicated timing. This increases the demands on validation and verification in order to ensure a working design. However, if you find a variation on the Flip-Flop theme that you would like to try, by all means experiment with it! SPICE simulation will tell you a lot about the speed and operation of your circuit.

# 3 Details

## 3.1 Layout Considerations

Since this cell may become part of your datapath, you should pay careful attention to the physical design details. Consider this a bit-sliced component with a target bit-slice height of 20-40 microns (this is the space between the power wire on the top and the ground wire on the bottom), and you should try to make the width (bit pitch) as small as possible (60-100 microns?). You can also use the same vdd/gnd spacing as for the standard cell library so that this bit could fit in your library. Chapter 5 in *Design with Cadence and Synopsys* has details about the standard cell template. In the end you probably may want to choose the template that will be demonstrated in class.

Try to picture how this register bit will work in building either a single register, or a register bank. Think about how this might all fit together with an ALU, shifter, etc. in a datapath. In particular, think about the direction of data flow, the direction of vdd/gnd routing, and the routing of control lines. Look at Figure 7 for an example of this idea. This example is an adder instead of a register, but the idea of planning for how the cells will fit together is the same. Notice how the adder inputs come in from the top, and the sum output goes out the bottom. The carry input comes in the left and leaves out the right. This way you can tile the cells together to make larger adders and the carry signal will be connected because of the abutment of the cells. For your register cell, I would plan on the top wire being vdd and the bottom being gnd. Then I would have the D input come in the top, and the Q and Qb leave from the bottom. Then I would think about routing the clock and clear signal through the cell horizontally so that the clear and clock will make contact with another flip flop cell abutted next to this one. That way you can put a row of these bits together and make a register with all the D inputs coming in the top and all the Q and Qb signals leaving from the bottom. This is shown in stick-diagram form (just the connections, not the transistors) in Figure 8. Also see the class lecture notes on the register example.

Thinking about these issues ahead of time will make your datapath a more manageable design. On the other hand, don't get caught up in trying to make everything perfect with this cell. At the very least, many of your designs won't be used because we'll be forming teams in the future, and you'll likely use only one of the team member's old designs. Even if your cell is the one that's used, you'll have a chance (and will probably want to) modify it later. You will become more skilled at cell design, so these early designs will probably need some adjustment before making into your finished project.

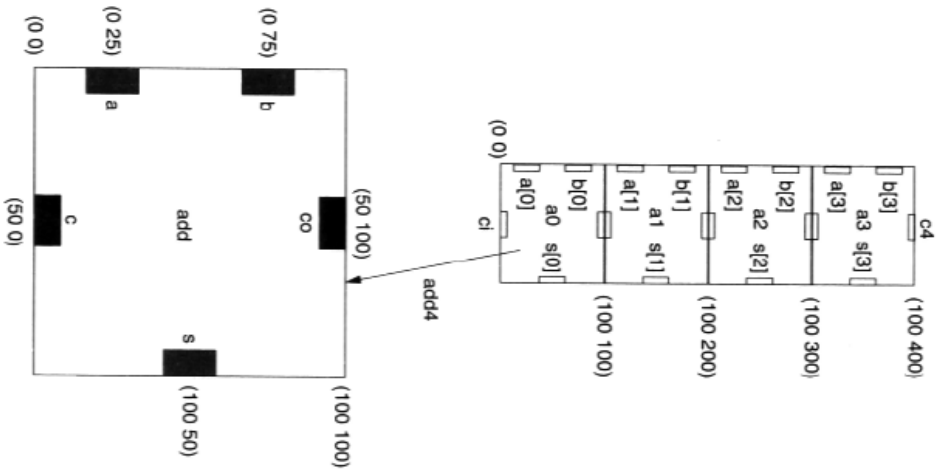


Figure 7: Four-bit adder slice

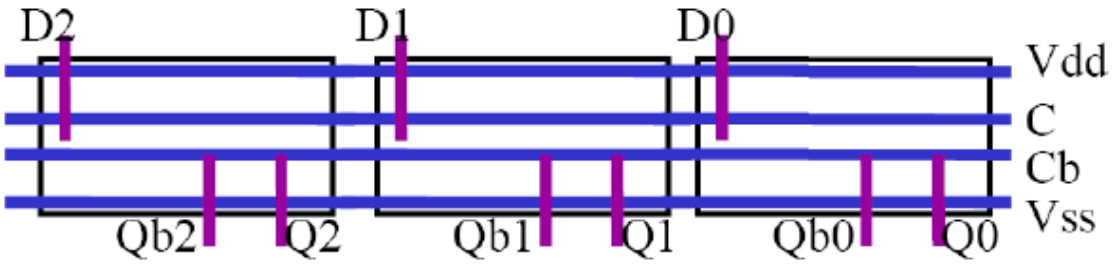


Figure 8: Three-bit register slice

## 3.2 Simulation Issues

These tips will help you successfully simulate your design.

### 3.2.1 Verilog Simulation of Tristate Nodes

Notice that the cell in Figure 2 has a node that is driven by one of two transmission gates (or enabled inverters) depending on the logic level of the Clock. This is an interesting node because depending on how the Clock and Clock-bar signals are generated there will be a short amount of time where both transmission gates are on, or a short amount of time where both are off. If both are on, then there is the potential for a drive fight where both transmission gates are trying to drive that node to a different value. As long as the overlap is short, this will cause no real problems and the node will resolve quickly to one level or another when there is only one transmission gate driving the node. If both transmission gates turn off for a short time, nothing will be driving that node. That's okay in this instance because the time constant is very small and the capacitance of that node will likely hold it at its old value until one of the gates turns on. This is all accurately modeled in the

analog SPICE simulator. However, this is *not* accurately modeled in the Verilog switch-level simulation unless you give the simulation some help!

The help Verilog needs is to know that the internal node has a capacitance that holds its value if both transmission gates are off. To do this you need to pass additional information to the netlisting process. Netlisting is the process that generates a simulation netlist from your schematic. This happens automatically when you fire up the Verilog interactive simulator. You must tell the netlister to extract the internal latch node as a trireg node instead of a wire. In order to do this, you need to put an attribute on the wire so that Cadence knows it is supposed to be a trireg. This procedure is described in Chapter 4, Section 4.4.4 of *Design with Cadence and Synopsys*.

Select the wire and select `Edit | Properties | Object` in the schematic editor. In the dialog box that pops up select "ADD" to add a new property. The name of the new property is netType (make sure it's spelled that way), the type of the property is string, and the value is trireg. Also, you may want to turn on the display of the value so that by looking at the schematic you can tell that this wire node has been designated a trireg. If you do this with any wire that has the potential to be undriven for any amount of time you will be able to simulate your latch with Cadence Verilog.

### 3.2.2 Transistor model directionality

You have probably noticed that the physical layout for the source and drain of our transistors is identical. Thus there is no physical distinction between what we call the source and drain of a MOS transistor. However, this is not the case with Verilog simulation. Verilog models are "directional" by default because this improves simulation performance. If you use NMOS and PMOS transistors in a design (such as in a transmission gate) you need to be very careful to get the source and drains connected correctly. Note how the transistors are oriented when you put them in an inverter (without rotating the transistor symbols). There is a little arrow on the side of the transistor that is closest to the power supply. That is, the "leg" with the arrow is on the top for the PMOS (close to vdd), and on the bottom for the NMOS (close to gnd). These are the source nodes of the transistors, which is why they are always connected to the power supplies. The drain is the other side. The output of the inverter is connected to the drains. When you make a transmission gate you need to keep this in mind. The input to the transmission gate should be the source (arrow-side of the transistor), and the output should be the drain (non-arrow-side) for both NMOS and PMOS devices. The Verilog built-in transistor models always pass data from source to drain. That is, the drain is considered the output. If you orient the transistors in some other way the transmission gate will not simulate correctly in NC-Verilog (again, it will work fine with SPICE simulation using Spectre). If you have a situation where you really need bi-directional data transfer in a transistor, you can use the bi_nmos and bi_pmos cells in the Analog_Parts libraries, but typically a transmission gate doesn't need this bidirectional data flow.

# 4 Assignment

Design and simulate your single bit register cell. The cell should implement a positive edge triggered flip flop with an active-low clear signal. In particular, do the following:

1. **Schematic View:** Create a schematic for your register bit using nmos and pmos transistors from the UofU_Analog_Parts library. Simulate the schematic using Verilog to make sure it is functioning properly. Remember that you get transistors with delay if you use the UofU_Analog_Parts versions.

2. **Schematic Justification:** Write a few lines about what D-type flip-flop circuit you've chosen to build and why you chose that circuit. Describe briefly how your circuit latches data, and how it clears its value.

3. **Layout View:** Create layout for your register cell. Keep in mind that you want to be able to tile these together to make a multi-bit wide register, and that you may also want to tile the multi-bit registers to make a register bank. Make sure that when you tile the layout in Virtuoso that you don't violate design rules in the cell or between cells when they're placed next to each other. Run DRC on your cell, and then LVS to comparing it to the schematic view.

4. **Analog Simulation and Characterization:** Run a SPICE simulation of your register cell extracted layout. Add a capacitance to ground of 100fF to the output Q node and another to the output Qb node to model the load that the register will be driving. Use the cap cell from the NCSU_Analog_Parts library for the capacitor. Use this simulation to find the rise and fall times and propagation delays for your register cell.

   The rise time is defined as how long it takes for the output of the flip flop to rise from 20% to 80% of its steady state value. The fall time is from 80% to 20% of its steady state value (vdd or gnd are the steady state values). Propagation delay (for this assignment) is defined as how long it takes from a 50% input level to a 50% output level. The propagation delay you are measuring is from the rising clock as the input to a changing Q at the output. There may be a different propagation delay for Q rising and for Q falling. Check them both.

   You also will measure the setup and hold times. The setup time is defined as the point in time at which if the separation between the clock and input decrease, the clock-to-Q delay increases. It can be measured as follows. Assume the clock signal is the fixed reference point. The data arrives infinitely early, and you measure the clock-to-Q delay. The data is incrementally moved closer to the clock signal and the clock-to-Q delay is measured. The separation between the input and clock edges when the clock-to-Q output delay increases is the setup time of the latch. The hold time is the dual of the setup time. It is defined as the point in time at which if the separation between the clock and input decreases, the clock-to-Q delay increases.

The measurement of the hold time is similar. The input arrives substantially before the setup time and it again switches after the clock. The time at which data switches following the clock edge is moved closer to the clock edge until there is a increase in delay for the clock-to-Q signal. As you can see, the setup and hold measurements will require doing several SPICE simulations by changing the relative times of the clock and data transitions. You don't need to go overboard on picosecond resolution on this, but do try to come up with a reasonable characterization of your flop. While doing this part of the flip-flop characterization, think of the following questions. Can you have a negative setup or hold time? (For example, the setup time occurs with the data switching *after* the clock edge.) If this is possible, why is that the case? What is the cause of the increase in delay due to setup and hold time violations? What is the worst-case scenario for setup and hold time violations in a flip flop?

5. **Layout view of an 8-bit register:** Create a layout for an 8-bit register by tiling 8 of your register bits together. You don't need to simulate this. This is just to demonstrate that your flip flop can be tiled into a register correctly. Make a schematic and DRC and LVS the layout to show that the CLOCK and CLRb signals are, in fact, connected in the layout.

# 5 Deliverables

Create a tar file containing all deliverables and submit it via canvas. Include a README file describing your schematic justification, comments regarding your characterization flow and results, any problems you had with the lab, and the file structure of your results.

For your single flip-flop, include:

1. An `xv` image of your schematic.
2. Verilog test benches.
3. Writeup of the justification of your schematic.
4. Image of your layout.
5. LVS log.
6. SPICE simulation results, including a table of propagation delays, rise and fall times, and setup and hold times for your flop.

For your 8-bit flip-flop, include:

1. An `xv` image of your layout.

2. An `xv` image of your schematics.

3. The LVS log.