CS/EE 6710 Digital VLSI Design
CAD Assignment #3
Due Thursday September 21st, 5:00pm

Overview: 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 file. 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 file. Eventually you will use this cell, or something similar to it, in your custom data path of your project in places where you need registers in your design. However, don't stress about getting it perfect in this assignment. I'm sure that you'll want to tweak some things, and probably 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 CAD assignment number 5.

The inputs to your register bit should be D, CLK, CLRb, and the outputs should be Q and Qb. The D input should be non-inverted (that is, a 1 on D should result in a 1 on Q after the clock, and 0 on Qb). The register should be rising-edge triggered which means the data should be captured and presented to the output on the rising edge of the CLK. The CLRb signal should be active low.

Implementation: There are many, 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 for an example, and look ahead to Section 7.3 for some more examples. You can also look in other books and articles to find lots of other circuits and variations on the basic circuits. I'll link some papers from the Journal of Solid State Circuits that discuss flip flop variations to the web site. 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, it is good at restricting the sampling of the D signal to the rising clock edge, but 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 in your register bit.

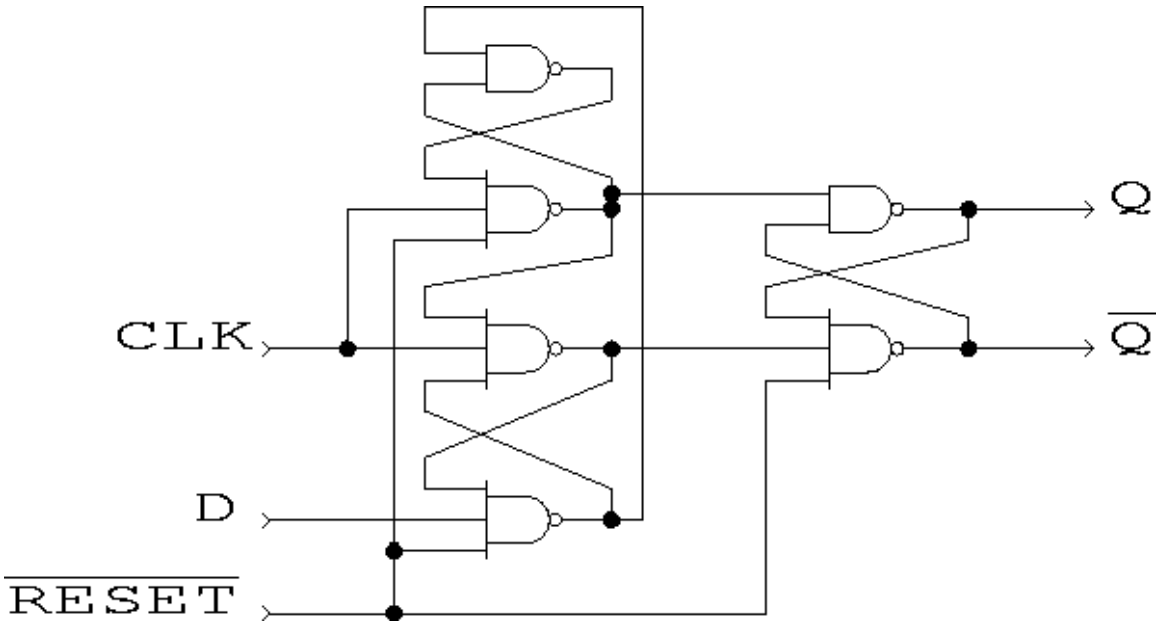

Figure 1: Positive Edge Triggered Flip Flop using Gates

A more CMOS-ish version of a storage element starts from a simple feedback loop that includes 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 (see also Figure 1.12 in your book).

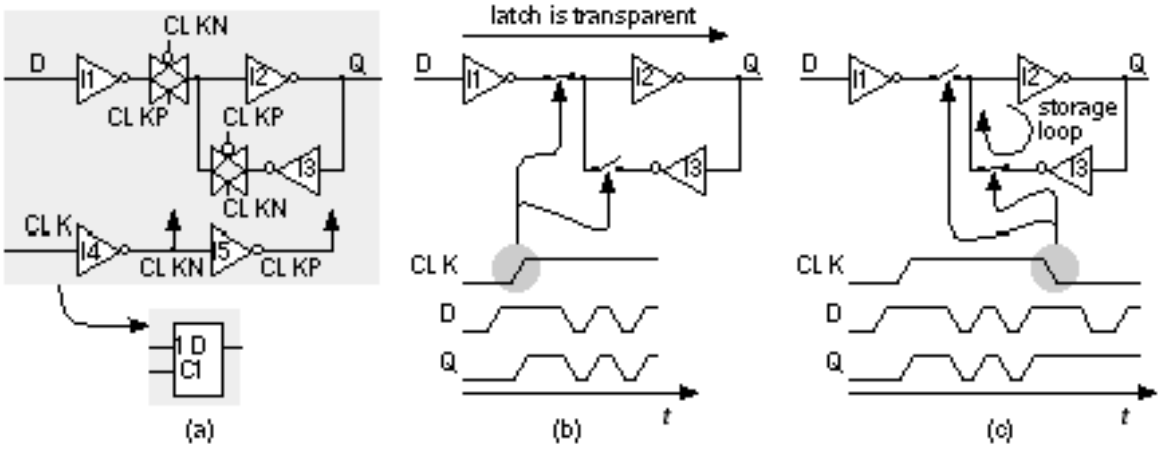

Figure 2: Clocked Latch Using Transmission Gates

These gated latches can also be implemented by merging the transmission gates with the inverters that are driving them to make a “clocked inverter” or “enabled inverter” or “tri-state inverter.” This circuit combines the functions in a smaller layout than using both an inverter and a transmission gate. The clocked inverter, or enabled inverter, will drive its output with the inverted input if the select lines are active, and will be disconnected from the output (the output will be in high impedance) when the select signal is inactive. The reason we can break the connection at the

output node of the inverter, as shown in Figure 3, is that the P-type transistor of the transmission gate will only be involved in pulling the output high, and the N-type of the transmission gate will only be involved in pulling the output low. Thus we can break the connection and fold the P-type into the pullup stack and fold the N-type into the pulldown stack.

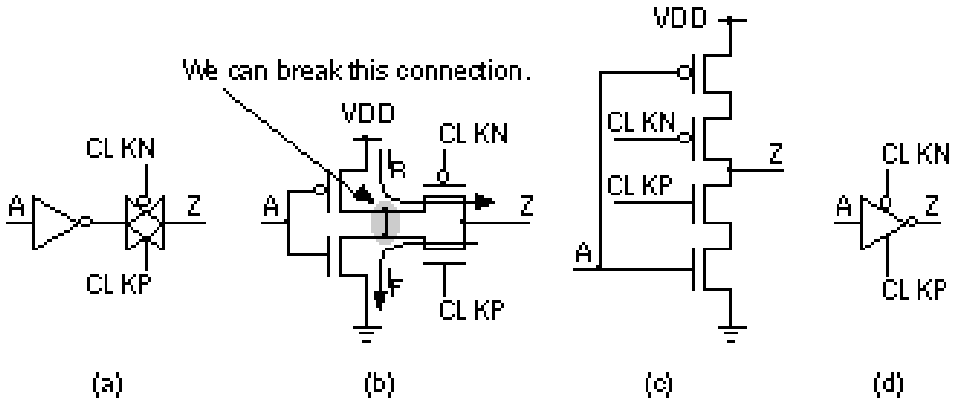

Figure 3: Clocked Inverter

We can use the gated latch from Figure 2 (modified to use clocked inverters 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 clocked inverters (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 Figure1 circuit.

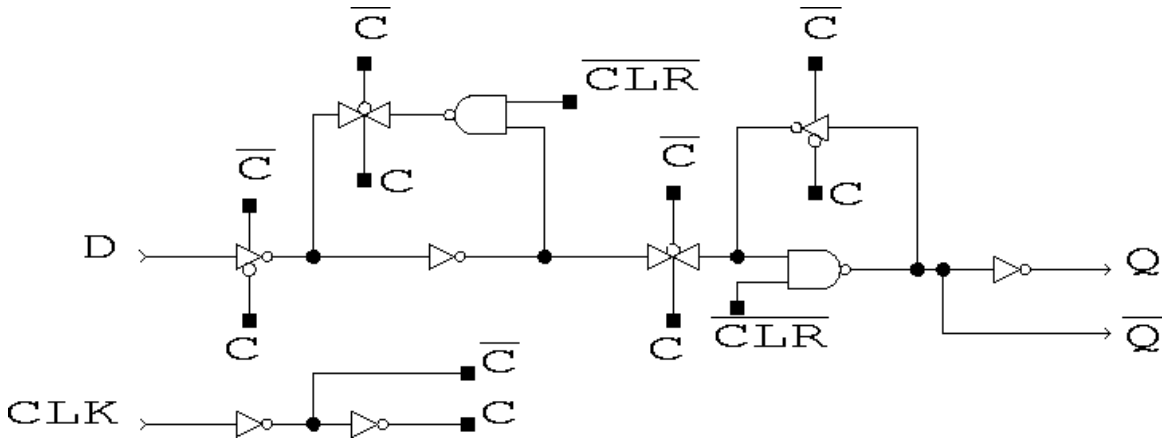

Figure 4: Positive Edge Triggered Flip-Flop using CMOS Structures

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 has the advantage that there is never a drive fight for the internal node (the one that is driven by both transmission gates in Figure 2 or by the transmission gate and the enabled inverter in Figure 4). 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 clearing transistors go.

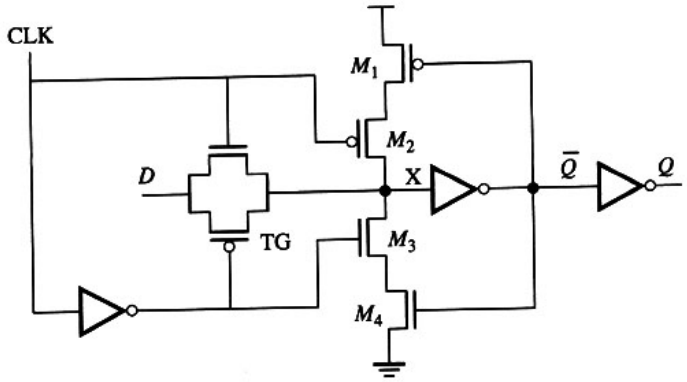

Figure 5: An Alternative Gated Latch

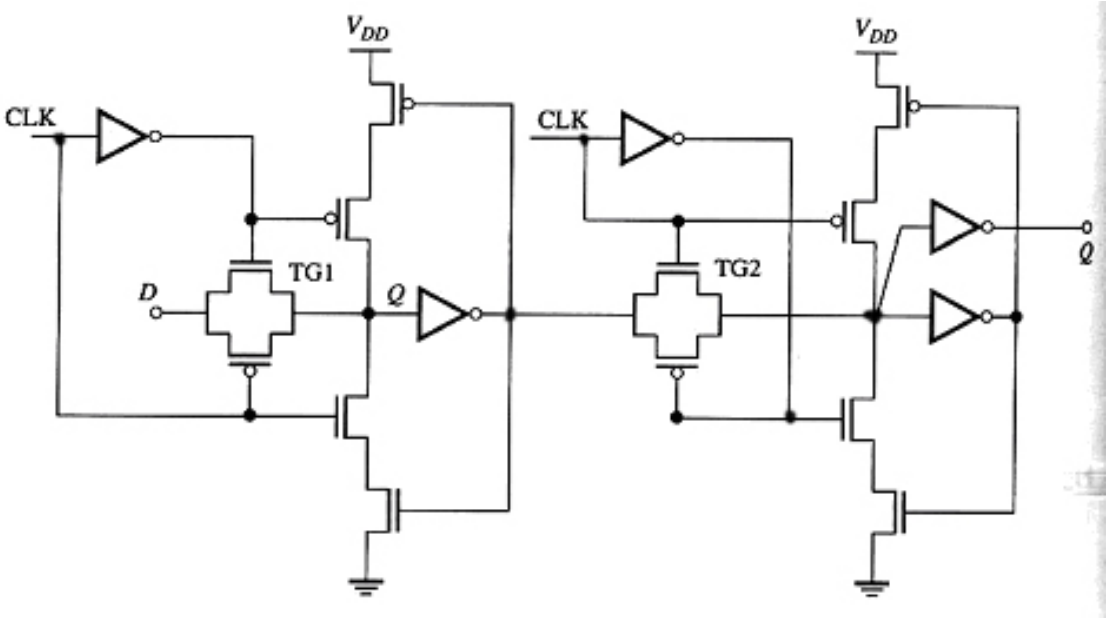

Figure 6: An Alternative Master Slave Flip Flop

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), but we recommend that it look a lot more like Figure 4 or 6 than Figure 1! You can make an even more compact circuit using dynamic or pseudo-dynamic circuits that we haven't talked about, or by using two-phase or other clocking schemes. These are also not recommended because they're

tricky and increase the demands on verification in order to ensure a working circuit. However, if you find a variation on the Flip-Flop theme that you'd like to try, by all means experiment with it! Analog simulation will tell you a lot about how the circuit is going to work.

Layout Considerations: Since this cell may become part of your datapath, you should pay careful attention to the layout aspects. 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?). See the handout on routing on the web page for more information about bit-slice pitch and over-cell routing. 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 4 in the Lab Manual has details about the standard cell template.

Try to picture how this register bit will work in building either a single register, or a register file. 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 inputs come in the left and leave 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's leaving from the bottom. This is shown in stick-diagram form (just the connections, not the transistors) in Figure 8.

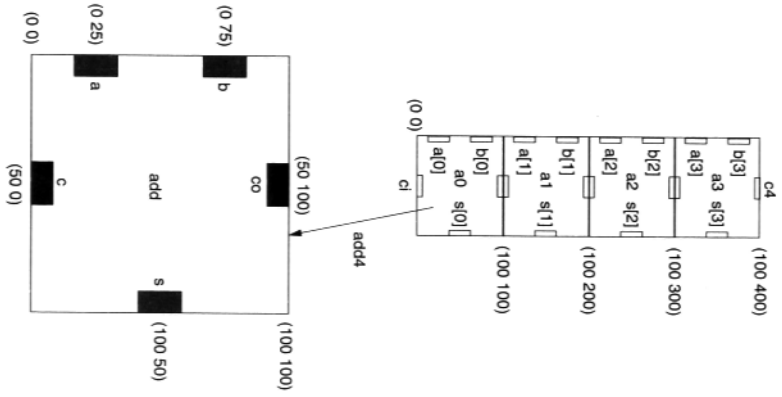

Figure 7: An Adder Slice

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 (I would be very surprised if any cells designed early in the semester make it through to the end!).

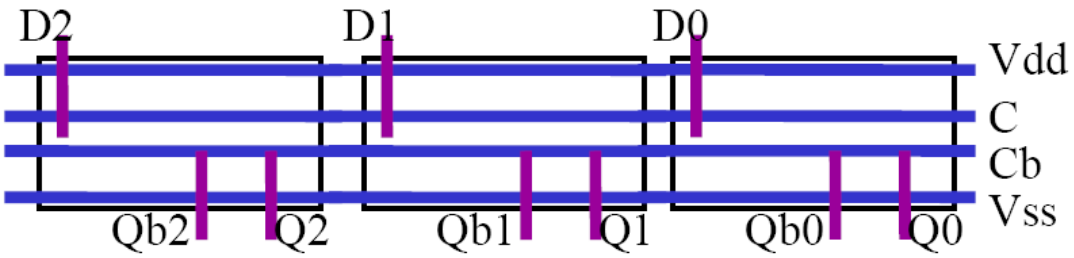

Figure 8: Three registers tiled together

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

Schematic view: Create a schematic for your register bit using nmos and pmos transistors from the **NCSU_Analog_Parts** or 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.

A note about the simulation – 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 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, but that's OK too because the capacitance of that node will hold it at its old value until one of the gates turns on. This is all accurately modeled in the analog 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 have the netlisting process (the

process that generates a simulation netlist from your schematic – this happens automatically when you fire up the Verilog interactive simulator) extract that node as a “trireg” instead of a “wire.” In order to do this, you need to put an attribute on the wire so that Cadence knows it’s supposed to be a trireg. This procedure is described in Chapter 4, Section 4.4.4 of the Lab Manual.

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, I like to turn on the display of the value so that by looking at the schematic you can tell that that wire 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 Verilog.

Another note about simulation – If you use nmos and pmos transistors to make a transmission gate you need to be very careful to get them pointing the correct way. 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 connections to 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 Verilog-XL (again, it will work fine in SpectreS). 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.

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 values, and how it clears its value.

Layout view: Create layout for your register bit. 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 file. You want to 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 LVS comparing it to the schematic view.

Analog Simulation: Do an analog simulation of your register bit 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** Lib for the capacitor). Use this simulation to find the rise and fall times and propagation delays for your register bit. The rise time is defined as how long it takes for the output of the flip flop to rise from 10% to 90% of its steady state value. The fall time is from 90% to 10% of its steady state value (i.e. vdd or gnd are steady state values). Propagation delay (for this assignment) is defined as how long it takes from a 30% input level to a 70% 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 should also be able to determine the setup and hold times. That is, how close to the clock can the data change and still result in valid data being latched? How close do you have to come for the data to be latched incorrectly? For how long after the clock do you need to keep the data around to get the right value latched in the flip-flop? The setup and hold measurements will require doing a few simulations and changing the relative times of the clock and data transitions until things mess up. You don't need to go overboard on picosecond resolution on this, but do try to come up with something reasonable.

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.

Turn in hard copy of the FF schematic, Verilog testbenches, schematic justification, layout, LVS log, analog simulation results, and a table of propagation delays, rise and fall times, and setup and hold times for your register bit. Also include layout, schematic, and LVS for the 8-bit register.