ECE/CS 5710/6710 – Digital VLSI Design
Lab Assignment #5

Due Thursday Oct. 30th via canvas

1 Introduction

In this assignment you will, **as a group**, design and characterize a small (5-cell) standard cell library that has enough combinational cells to build combinational logic using synthesis from Verilog.

2 Background

For this assignment you will be making a small library of five primitive cells that are sufficient to use for synthesis. These include an inverter, a 2-input NAND gate, and a 2-input NOR gate. You will also design a *TIEHI* and *TIELO* module. These modules provide safe high and low voltage sources for gate inputs in the design.

Your group will design five cells as described below following all the cell library template requirements in Chapter 6 of *Design with Cadence and Synopsys* and in the rules and template as provided on the class web page. The distance between the center of the power and ground wires is fixed at 27 microns, for example, as defined in the template. The width of a cell is not fixed, but must be in increments of 2.4 microns. There are lots of details in Chapter 6.

2.1 Shared Group Access

This is a group assignment. Groups can range from two to five members. Send email to teach-5710 **no later than Oct. 9th** with the names of the people in your group and everyone's CADE user login name. Have one team member email the team list to teach-5710@list.eng.utah.edu and cc all your team mates. *If you are having a hard time finding other team members, please contact the instructor immediately.*

Once you have formed your groups and informed the instructor and TA's, you will be assigned a *group number* that you will use to define your cell library. Once your groups are formed, you will also be assigned a UNIX group so that all members of the group can have shared access to one single library and project. If you make the directories owned by the group (`chgrp -R NEWGRP PROJDIR`) and set your UNIX permissions to `rwx` for your group (`chmod -R 770 PROJDIR`), then everyone in the group will have access to the library and project. *Make sure you follow this procedure!*

2.2 Group Library

For this lab you should make a new Cadence library called **Lib6710_xx**, where xx is the number of your group. *This library will **only** hold the cells you define for your cell library*. All designs that use your cell library will be in a different cadence library. This will make things easier later to keep your library separate from your project design.

You will choose one group member to place the library in their user directory. You will need to point to that directory in your Library Path when you do designs. This is done through the library manager in Cadence.

There are logistical issues with maintaining group-shared files. Everyone in the group needs to remember to make sure that the files that they create are in the right group, and have the right group permissions. Otherwise other group members will not be able to read or modify the library files.

There is a UNIX command called `sg` that will allow you to run any command in an environment that has a different default group. The CAD scripts (cad-ncsu, etc.) all have a tweak in them so that if you define a group in your CAD_GROUP environment variable, it will run the tools using that group (and with the correct group `rwx` permissions) using `sg`. Once you get your group number and UNIX group, if you set your CAD_GROUP environment variable to that group, then every time you use one of the scripts for the class, it will automatically use that group and `rwx` permissions. Make sure you do this by putting `setenv CAD_GROUP NEWGRP` in your .tcshrc or .cshrc file. You can select the group when working on the VLSI project by executing `newgrp NEWGRP` in a shell.

Best practice is to always give access to all your files after working on the project. Thus, you may want to put `chmod` and `chgrp` commands in your .logout file so that they get run every time you logout. This will ensure that you won't forget to change the group and permissions before you skip town for the weekend and leave your other group members in a bind. This is done by putting `chmod -R 770 PROJDIR` and
`chgrp -R CAD_GROUP PROJDIR` commands in a .logout file in your home directory.

3 Details

3.1 Tiehi and Tielo cells

Figures 1 and 2 show the schematics for the tiehi and tielo cells that you will define for your library. These will typically be used to drive cell library inputs either high or low. The tiehi cell creates a diode connected n-type device that provides a low voltage output. The tielo cell is similar, creating a diode-connected p-type to produce a high voltage output.

Gate inputs can not be tied directly to the power or ground rails. Doing so can destroy the delicate gate oxides if there are power supply voltage spikes. The TIEHI cell uses a

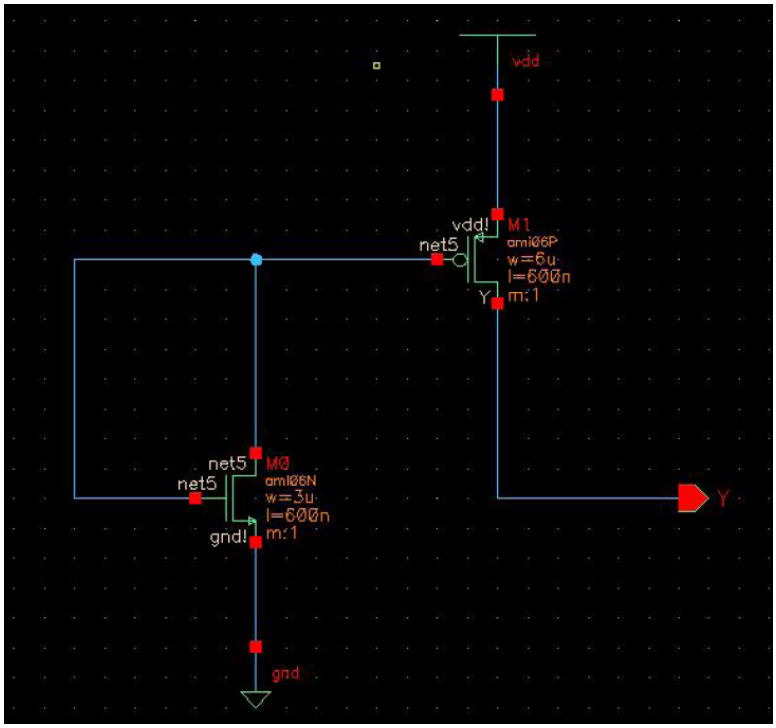

Figure 1: Schematic of the tiehi cell

diode-connected nmos device that provides a low voltage to the gate of the pmos transistor. That pmos then provides a high voltage at the drain output. The TIELO is similar, using a diode-connected pmos and regular nmos to provide a low voltage at the nmos drain. A "diode-connected device" is a transistor that has its drain and gate tied together so that it functions as a diode from drain to source. Note that there are no transistor gates connected directly to a power supply in either of these schematics.

There is an interesting issue when trying to simulate these cells with nc-verilog: the switch level simulator doesn't know how to simulate diode-connected transistors. This is because from nc-verilog's point of view, it doesn't know how to initialize the diode output. So, in order to correctly simulate these cells you need to provide additional information. Remember the parameter we used in the register schematic of giving a node a `netType` attribute of `trireg` so that it could store its old value when disconnected to the driver? We use the same parameterization in this case, but rather than specify the node as tristate, we initialize the node to VDD or ground. A Cadence node initialized to VDD is called tri1 and a node that is initialized to GND is called tri0. This is accomplished the same technique that you used in Lab 3 by adding the netType attribute to the wire and assigning the value as tri1 or tri0. By assigning the correct values to the internal nodes of the tiehi and tielo cells they will simulate correctly in nc-verilog.

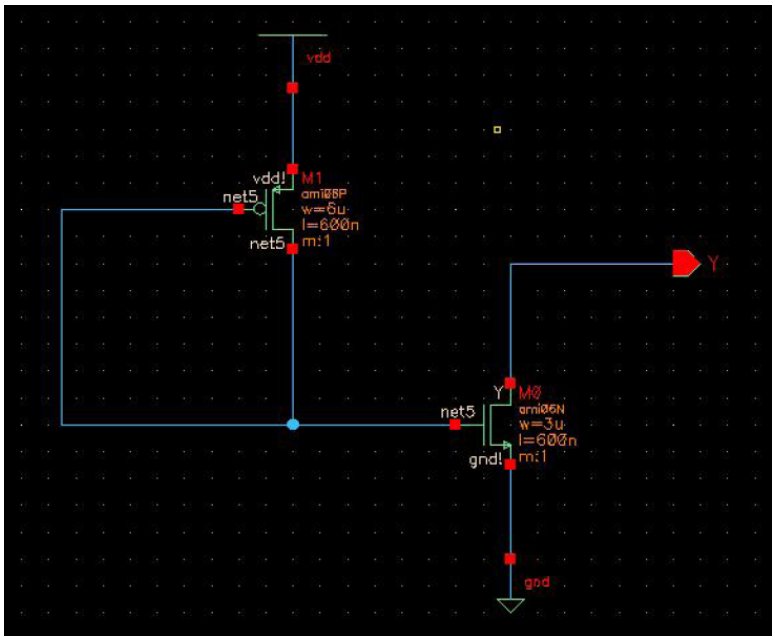

Figure 2: schematic of the tielo cell

Note that these attributes have no effect on SPICE simulations using Spectre. Your analog simulations will correctly model the diode-connected transistors by using DC analysis to converge on the voltage values for these nodes.

3.2 Library Characterization

You will need to characterize the performance of your cells in order to use the cells in high level synthesis. All cell libraries contain their characterization information the *liberty* (.lib) format. This results in a two-dimensional matrix containing delays based on input slope and effective output load. The actual slope and loads are estimated when running synthesis in order to help implement timing driven synthesis and place-and-route.

We will use a tool called *Encounter Library Characterizer* (ELC) in order to characterize our library. ELC will run *many* SPICE simulations in order to characterize the performance of the library and create the .lib file.

Chapter 8 in *Design with Cadence and Synopsys* goes in to more detail about cell characterization, the format of the liberty (.lib) file, and how to generate those files with ELC. Following is a very short cheat-sheet version of the process. You will need to read *Design with Cadence and Synopsys* to successfully characterize your library.

You need the following views for each cell you will characterize:

1. **cmos_sch** – transistor level schematics

2. **behavioral** – Verilog behavioral views
3. **symbol** – the schematic symbols
4. **layout** – mask layout
5. **extracted** – parasitic extracted values.
   This is generated from the extract process and used for LVS.
6. **analog_extracted** – generated from successful LVS.

This is the process you will follow to generate these views, and to run library characterization:

1. Make a schematic that holds one copy of each of your five cells (**INVX1**, **NAND2X1**, **NOR2X1**, **TIEHI**, and **TIELO**).
2. Use the Analog Environment to generate a SPICE netlist for Spectre. Call it foo.scs.
3. Modify the foo.scs into dut.scs using a text editor (I suggest emacs), or the sp2elc script in the /uusoc/facility/cad_common/local/bin/F13 directory.
4. make an ELC directory in which to run cad-elc. Put the dut.scs file in your ELC directory and connect to that directory before running `cad-elc`.
5. In your ELC directory, run `cad-elc` three times, once with each step file. The script will make links to the correct step1, step2, and step3 files from the /uusoc/facility/cad_common/local/class/6710/F13/cadence/ELC directory. Make sure that your scs file is named dut.scs, or you will need to change the step scripts to match the file name that you used.
6. Finally, use the cad-alf2lib script to convert the alf file from ELC to the .lib file that you need. Make sure to properly edit your footprints.def file before running cad-alf2lib.

4 Assignment

You will create a shared library named **lib6710_xx**, where xx is the number of your group. This holds your core library cells. You will build the following five cells that you will add to your Lib6710_xx library:

- INV1 – a standard 1X sized inverter.

- NAND2X1 – a two-input NAND gate with standard size transistors (remember to take series transistor stacks into account).
- NOR2X1 – a two-input standard NOR gate.
- TIEHI – A cell that provides a connection to VDD. There is only one output, and no inputs. The single output is a connection to VDD. This is a resistive connection through a transistor so that the gate that it connects is protected from direct connection to the power supply, as shown in Figure 1.
- TIELO – A cell that provides a connection to GND. This is the dual to the TIEHI cell. The schematic for this cell is shown in Figure 2.

Your cells will need to have the following views: cmos_sch, symbol, layout, behavioral (Verilog), extracted, and analog_extracted.

The cells will need be designed, have layout made, simulated, and then run successfully through DRC and LVS. The cells will then need to be characterized as described in Chapter 8 in *Design with Cadence and Synopsys*. This should result in a liberty format file for your five-cell library named **Lib6710_xx.lib**. This is a very involved and somewhat picky procedure! The Encounter Library Characterizer described in Chapter 8 is the best way for you to characterize normal static CMOS cells. You can also simulate by hand or with scripts using Spectre directly if you have trouble using the library characterizer.

Once you have a Lib6710_xx.lib, you should compile it into a **Lib6710_xx.db** using Synopsys design compiler. This is described at the end of Chapter 8 in Section 8.4.

With a .db file of your five-cell library, you should demonstrate that you can synthesize a simple combinational circuit using Synopsys. You can use the simple beh2str script as described in the first section of Chapter 9, or you can use the more general syn-dc script or design_vision versions if you like (also described in Chapter 9). The idea for this assignment is just to make sure that your cell library is specified correctly so that synthesis will work! *Note that because your library currently doesn't have a flip flop this must be a fully combinational circuit!*

5 Deliverables

Create a tar file containing all deliverables and turn it in via canvas. Please only turn in a single tar file for each group, and only include the files indicated below.

Create a directory (such as LibData_xx using your group number for xx), and place the following files in that directory.

1. A README file that describes the files in this directory.
2. The Verilog testbenches for each of the five cells.
3. Image files (such as png) of the waveforms of the Verilog simulations of each of your cells. Note that these simulations should be the switch-level simulations of the cmos_sch view, NOT simulations of the behavioral views. Make sure that your Verilog netlist settings are getting the correct views.
4. A copy of your Lib6710_xx.lib file that you generated using ELC or Spectre simulation.
5. The Verilog code that describes the combinational circuit that you synthesized as a demonstration that the library works with synthesis.
6. The structural Verilog result from synthesis that demonstrates that the synthesis procedure used the cells in your library.