CMPE 122/222
Digital Simulation and LVS

Prof. Matthew Guthaus

February 10, 2019

Fixes

- 3/10/19 Changed set to setvector.

1 Requirements

This tutorial will use the following CAD tools:

- xcircuit (optional)
- IRSIM
- Netgen

Entering schematics directly into spice format is possible, but more often you do schematic entry and then edit the stimulus in spice directly. It is VERY useful to understand spice syntax as you often have to debug tool output when things go wrong. Spice syntax has evolved over time and there are slight differences in some tools – especially with naming. We will use xcircuit for schematic entry. It is also a nice tool for generating good schematic images. It may seem outdated because it uses an X11 (an old-school graphics) library, but it is quite powerful actually.

Netgen is a layout versus schematic (LVS) checking tool. It confirms whether your layout implements the desired circuit that you design in a schematic editor. While a layout may be DRC clean, it may not have the correct connections or even the correct transistors to implement a design.

IRSIM is a digital simulation tool. It will perform switch based analysis to see if your spice netlist is logically correct. It is not accurate for performance, but it does use simple resistive transistor models.

The next section provides very brief instructions on how to use these three tools. I've consolidated the most important concepts here, but please seek more help in the Piazza forum or office hours if you have particular problems. I am trying to distill the important information, but suggestions of what to add are always appreciated.

2 Introduction to xcircuit

There are a full set of xcircuit tutorials available at:

`http://opencircuitdesign.com/xcircuit/tutorial/`

You can skip most of tutorial 1, but you may need to reference it if something confuses you. You will likely not need the advanced drawing features for this course and can skip those. The most unique part of tutorial 1 is how xcircuit saves files. It uses the postscript langauge to do this. Most people don't know, but postscript is quite an elaborate language and xcircuit leverages this. Tutorial 2 is the important one and covers schematic entry in depth. The goal of this tutorial is to teach you enough to get by for this homework assignment.

2.1 Running xcircuit

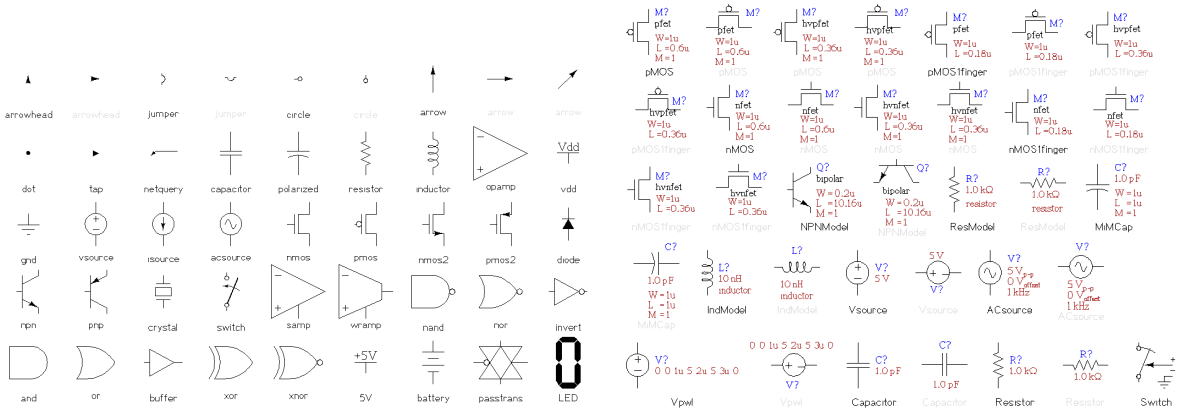
You can run xcircuit from the command line by typing:

```
xcircuit &
```

The ampersand tells it to run the program in the background so you can use your prompt for other things.

2.2 Adding a Library Component

You can add a library component to your schematic using the "l" hotkey to cycle through the libraries. Upper case "L" will show all pages simultaneously. By default, there is a General library, AnalogLib, and an empty UserLib as shown here:



You should use **only** the following instances in the AnalogLib in your design:

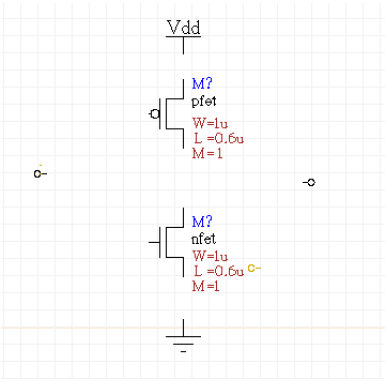
- pfet
- nfet

You should use **only** the following items from the General library in your design:

- vdd

- gnd
- circle

Place one pfet, one nfet, one vdd, one gnd, and two circles as shown in the below image:



You can place more than one instance each time you click the mouse button. You can flip the circle to be placed using the **f** hotkey while placing a second instance. Other useful shortcuts can be applied to components after they have been placed by positioning the mouse over the component and typing the hotkeys (all commands are under the **Edit** menu):

- Copy **c**
- Flip vertical **F**
- Delete **d**
- Undo **u**
- Redo **U**
- Rotate **r**
- Wire dot (filled) **.**

You shoud **NOT** use the voltage sources to drive inputs in your schematic since we are only creating a netlist of your design. You may later create a second sheet with the spice stimulus on it, but more often designers will write a separate spice stimulus file that has the voltage sources and other commands that will be later discussed. These items can cause problems with some LVS tools so you include the plain spice netlist into your spice stimulus file for simulation.

2.3 Wiring

You can start a wire connection by moving your mouse over a wire or pin and selecting the hotkey **w**. You can then left click multiple times to draw a wire. To cancel the wire, press escape. To mark the final point, you must CENTER click. Xcircuit assumes you have a three button mouse and the center (or right) button is frequently used for actions. On OSX, you can achieve this by pressing Option and clicking.

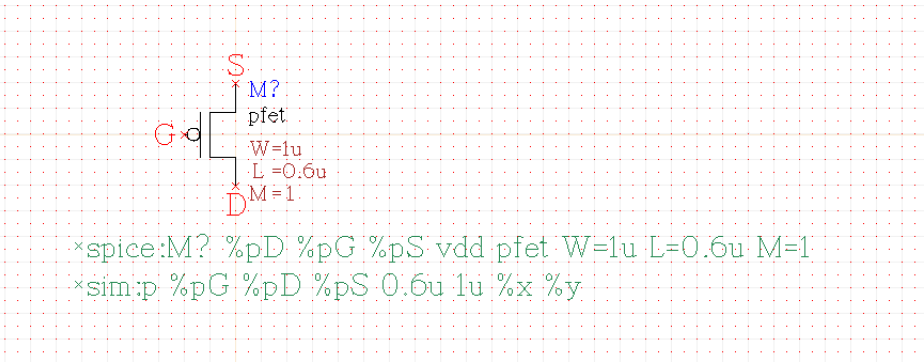
2.4 Pins

You create input and output pins by adding pin labels with the **Netlist→Make Pin** command (shortcut **T**). Note: lower case **t** will just add text, not a pin. The little x specifies what the pin is attached to. You are not required, but I suggest using the "circle" library component to specify an input and output.

2.5 Editing Parameters

Transistors (and other components) can have parameters associated with them. The library will provide default values for these parameters, but they are often not the ones you want and must be modified.

Editing parameters can be done by descending into an instance using the > hotkey. Similarly, < will return to the previous level of hierarchy. Descending into the pfet will take you to a window like this:

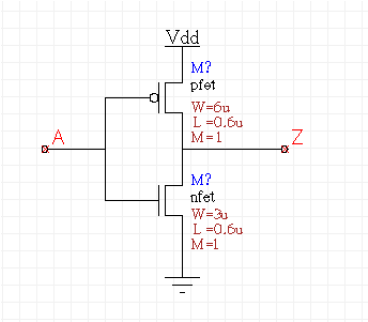


Once you are in a component, you can move the mouse over a text parameters and press **e** for edit to modify the parameter. You can also see the format specifier that is used to generate the output in the spice file.

It is important to note that you can edit a library component or an instance of a library component. Changing a library component will change the default value for all instances. Changing an instance will change a value only for that copy. Xcircuit remembers how you got to a library component to determine whether you are editing the library itself or the instance. If you edit a transistor from the library page, it will apply to all instances that use the default value. If you edit a transistor from your own schematic, it will only modify the values for that instance. You may also wish to learn about Virtual Library components in the tutorial, but it is not required.

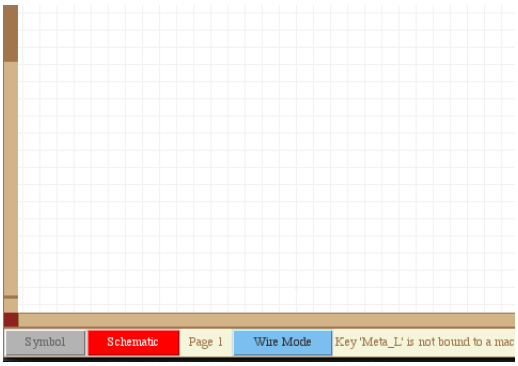
From the library (using hotkey **l** and then >), I modified the body connection of both the PMOS and NMOS to "vdd" and "gnd" instead of "avdd" and "agnd", respectively, since this is necessary for spice simulation later. You can also change your default gate length in the library to 0.6u to save time. Since each transistor will have a per-instance width, you should modify this from your own schematic rather than the library page.

Your final design should look like this:

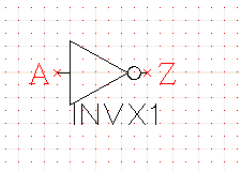


2.6 Creating a Symbol

To create a symbol, select **Netlist→Associate with Symbol** and select the inverter from the general library. If you then select the grey Symbol tab at the bottom as shown here



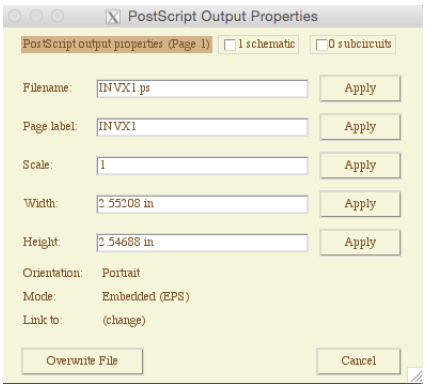
You can change the symbol to have the same input and output name as your transistor-level inverter using the **e** command to edit the pins. If you wish to draw a custom symbol, you can do this with **Netlist→Make Matching Symbol** and draw lines, arcs, etc. An example of your symbol with modified pin names is shown here:



I also added some text **t** to the symbol for clarity. You can use the hotkey / to switch between the symbol and schematic.

2.7 Saving the Design

You must specify the page label and the file name to save the layout. You can do this by selecting **File→Format Page Output** and filling out the following information:



The page label is what the design will be named when it is used hierarchically. Be sure to press the **Apply** button to have the changes take effect.

To actually save, select **File→Write All...** and press the **Write** button. This will create a file **INVX1.ps** which you can directly print since it is a postscript file!

2.8 Exporting Spice and Sim

You can now export two useful views for future steps: the spice netlist and the sim file (which will be later discussed for digital simulation in a later homework). Then select **Netlist→Write SPICE Netlist** (or **Netlist→Write flattened SPICE**)and **Netlist→Write sim**. Your spice netlist should be created in a file called INVX1.spc and look like this:

```
*SPICE circuit <INVX1> from XCircuit v3.8 rev 78

M1 Z A GND gnd nfet w=3u l=0.6u m=1
M2 Z A Vdd vdd pfet W=6u L=0.6u M=1

.end
```

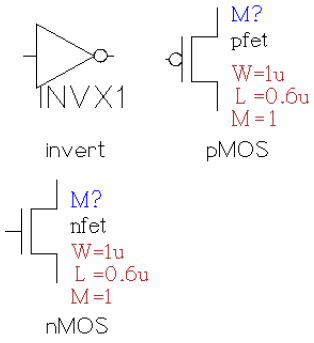
We will discuss spice syntax more in the next section.

Xcircuit names spice files with the extension .spc which is a bit uncommon, but it doesn't really matter.

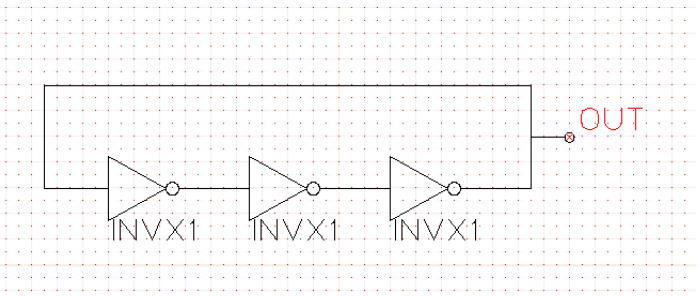
2.9 Creating Hierarchy

If you want to use your new gate in a schematic, you can do this by adding a new page. You can view all pages of your schematic under the **Window→Page Directory** option. By default there are 10 pages, but you can add more as needed using **Window→Goto Page→Add New Page**. Note, I sometimes have issues with it refreshing if I use the Goto Page option, but your mileage may vary.

You can now select some of the User Library components like these:



Note that my customized nfet and pfet are saved here rather than the AnalogLib. You can place and wire these like any other library component as shown in the following hierarchical example:



Now, if I export spice, I will see the subcircuits of the inverter as well:

```
*SPICE circuit <OSC> from XCircuit v3.8 rev 78

.subckt INVX1 A Z
M1 Z A GND gnd nfet w=3u l=0.6u m=1
M2 Z A Vdd vdd pfet W=6u L=0.6u M=1
.ends

X1 int3 OUT INVX1
X2 int2 int3 INVX1
X3 OUT int2 INVX1
```

If you save your schematic in different files, you may need to use the **File→Load Dependencies** command to load all of the files. This should load any undefined symbol by loading a schematic named the same as the symbol name plus ".ps".

2.10 Spice File Format

Every line in the spice file format is a "card" which is basically a command or a circuit element. Spice was originally intended to be an interactive, text-based program but as circuits got bigger, people prefer to use files of spice commands to be able to repeat simulations easily. The syntax is very simple and allows simple output from tools and parsing of circuits for customization by your own scripts in other languages (e.g., Python, C, Perl, etc.).

The first line of any spice file is always ignored! It is there to have a comment about the file. It is a common mistake to put something important here, so always add a comment at the top of any spice file! In addition to this first line, any asterisk (*) indicates the start of a comment and the remainder of the line is ignored.

2.10.1 Passive Elements

This section describes passive elements: resistors, inductors, and capacitors. Assorted magnetic elements are supported by spice but they are not presented.

Resistors, inductors, and capacitors come in two types:

- a simple, linear element with a value and some dependence on temperature, initialization, and scaling
- an element that refers to a more complicated model statement

Using the set of passive elements and model statements available, you can construct a wide range of board and integrated circuit designs. To use a particular element, an element statement is needed. It specifies the type of element used and has fields for the element name, the connecting nodes, a component value and optional parameters. The general form is:

```
name node1 node2 ... nodeN [model reference] value [optional parameters]
```

Element parameters within the element statement describes the device type, device terminal connections, associated model reference, element value, DC initialization voltage or current, element temperature, and parasitic.

We can specify these three passive devices merely by using the first letter of the device name such as:

- Rxxx for resistor
- Lxxx for inductor
- Cxxx for capacitor

To specify the device in the circuit file, we include the name of the device, how it is connected into the circuit, and its value. Ngspice uses the basic electrical units for voltage (volts) and current (amperes) and uses the basic electrical units for device values: ohms, farads, and henries. You will find that common prefixes also work such as m for milli-, u for micro-, p for pico-, n for nano-, f for femto-, etc.

Here are some example devices:

```
R12 A B 5k *a 5-kiloohm resistor connected between node A and node B
C7 OUT GND 3u *a 3-microfarad capacitor connected between node OUT and node GND
L5 6 8 1m *a 1-milihenry inductor connected between node 6 and node 8
```

Nodes are essentially ideal wires that connect a circuit by sharing a name. Often these are given numbers because of very old spice standards, but they can also be given names such as “in”, “a”, etc. Some spice simulators have a limit on how long the names can be and whether they are case sensitive. **Node 0 is always considered ground.**

2.10.2 Active Devices (MOSFETS only)

A MOS transistor is described by use of an element statement and a .MODEL statement. These are usually provided to you by a foundry using characterized data from their fabrication technology.

The element statement defines the connectivity of the transistor and references the .MODEL statement. The .MODEL statement specifies either an n- or p-channel device, the level of the model, and a number of user-selectable model parameters.

The following example specifies a PMOS MOSFET with a model reference name, PCH. The parameters are selected from the model parameters. The most common are the width (W) and length (L) of the transistor.

```
M3 3 2 1 0 PCH <parameters such as L=50n W=180n>
```

Note that the connections in the MOSFET are given in the order: drain, gate, source, and body. Since MOSFETs are generally symmetric, you can usually swap the drain and source.

3 Introduction to IRSIM

Irsim is an event-driven logic-level simulator for MOS (both N and P) circuits. This is useful for logic verification of digital circuits BEFORE you create the layout and without doing detailed, often slow spice simulation. Two simulation models are available:

- Switch model - each transistor is modeled as a voltage controlled switch. It is useful for initializing or determining the functionality of the circuit.
- Linear model - each transistor is modeled as a resistor in series with a voltage controlled switch; each node has a capacitance.

For our purpose we will concentrate on the switch model. If you want accuracy, you should use Spice instead.

3.1 Starting IRSIM

To start IRSIM, type the following command from the shell:

```
irsim scmos30.prm file.sim
```

scmos30.prm is the electrical parameters file that configure the devices to be simulated. It defines the capacitance of the various layers, transistor resistances, threshold voltages, etc.

File.sim is the input file for the simulation generated by xcircuit or by Magic using ext2sim if you made a spice file without xcircuit. You should not create one by hand!

Detailed information on IRSIM is available at:

`http://opencircuitdesign.com/irsim/reference.html#Documentation`

3.2 .sim File Syntax

The SIM file syntax is MOS-specific. In general, it is created from your netlist or layout, but it is helpful to understand the syntax to debug designs. SIM files have no hierarchy; they are simply a list of devices, capacitors and connections. Here are the file entities:

Unit scale and SIM file format type:

```
|units: scale tech: tech format: fmt
```

Scale is an integer scale factor that is used with the certain option of tools. Transistor lengths and widths in this file are multiplied by this factor and the result is in centimicrons. Tech is ignored. Fmt selects a format type from the set MIT, UCB, LBL. If no format type is specified, no property information is assumed on the transistor lines. If the entire line is absent then MIT format is assumed.

Transistors types:

```
n gate source drain {substrate} length width x y
p gate source drain {substrate} length width x y
```

The arguments gate, source and drain are required. The argument substrate is required only if the LBL format type was selected in the previous line. The other arguments are all integers denoting distance in centimicrons and are scaled by the unit scale factor.

User-defined device types:

```
DEFINE usertype param1 {param2 ... paramn}
```

Capacitors:

```
C net GND capacitance
```

Net aliases:

```
= net1 net2
```

For further information on the SIM file format, see the man page entry for SIM (section 5, file formats).

3.3 IRSIM commands

After processing the files named on the command line, IRSIM will accept further commands from its command prompt:

```
irsim>
```

Note that if you generate a sim file from xcircuit, it may not have lambda set properly and will give a warning like this:

```
(inv.sim,1): WARNING: sim file lambda (0.01) != config lambda (0.3)
(inv.sim,1): WARNING: Using the config lambda (0.3)
```

Since this is for switch-level verification, you can ignore this.

Here are some of the most frequently used commands in IRSIM:

- @ filename - sources commands from file filename.
- ana wnode - display nodes in analyzer window. (Same as analyzer).
- vector label node1 node2... - group signals in a bit vector displayed/set as a group.
- set label 01101 - set a group of signals created as a vector above to a binary value
- h wnode1 wnode2 ... - sets nodes to logic level high (1).
- l wnode1 wnode2 ... - sets nodes to logic level low (0).
- u wnode1 wnode2 ... - sets nodes to logic level undefined (x).
- x wnode1 wnode2 ... - remove nodes from being inputs (externally driven).
- inputs - display list of nodes which are inputs (externally driven).
- d [wnode] ... - print display list or specified node(s).
- s [n] - simulate for n ns. Default is stepsize.
- c [n] - simulate for n clock cycles. Default is 1.
- p - step clock cycle one simulation step
- stepsize [n] - set simulation step size to n ns. Default is 10ns.
- clear - clear analyzer display (a new ana command will display the signals).
- exit - exit the IRSIM simulation

An example usage:

```
irsim> vector C c7 c6 c5 c4 c3 c2 c1 c0
irsim> setvector C 10110110
irsim> exit
```

For a full list of available commands see the IRSIM web page previously given.

You can create a file with IRSIM commands and run this from the command line with:

```
irsim scmos30.prm mydesign.sim -example.cmd
```

or you can source this from the IRSIM prompt:

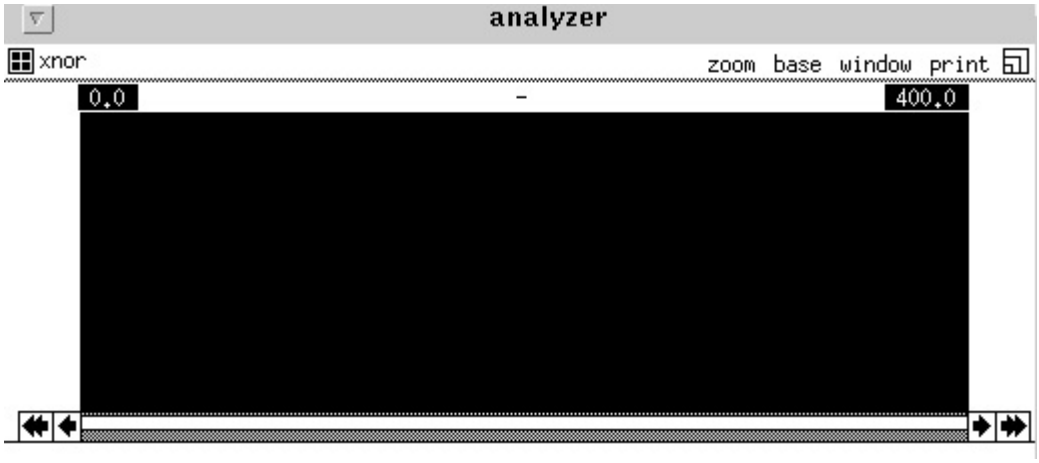
```
source example.cmd
```

or

```
@ example.cmd
```

3.4 The Analyzer Window

After invoking the **ana** command, the Analyzer window will be displayed on the screen like this:



You will use this window to monitor the simulation waveforms. First, you have to set the values for the input nodes. Any node can be set using the h, l, u or set commands. Related signals (e.g., 16 bits of a register) can be grouped in a vector using the vector command and display it as a vector. Or you can combine all the commands to use in a file and use the **@ filename** command to source the commands from the specified file. This is useful to reduce the amount of typing you have to do during debugging!

When you have set the inputs, you have to advance the simulation clock using the **s** command. For example, if you want to simulate an inv device (like the previous inverter with input in and output out) for input value in=1 and then in=0, you should type:

```
irsim> ana in out
irsim> h in
irsim> s
irsim> l in
irsim> s
irsim> exit (when done)
```

This will generate an output value on the Analyzer window, corresponding to the input values. To simulate the device for all possible input combinations, repeat the described steps, while setting different values for the input nodes.

For more information, see the IRSIM tutorial:

`http://opencircuitdesign.com/irsim/tutorial/tutorial.html`

3.5 Common Pitfalls

The analyzer does not recognize input nodes. The reason for this is most likely missing labels or failed signal declaration. Go back to your circuit drawing (layout) and check the node labels and signal declarations.

The output value produced by the simulation is undefined. This might be a result of missing connection or bad circuit design. Go back to your circuit drawing (layout) and check the circuit structure.

The **set** command only works with labels and not inputs directly. For single bit inputs, use h, l, or x.

3.6 Printing the Results

From the File menu of the Analyzer window, choose the Print option. This will produce a .ps (postscript format) file, which you can print using the lpr command. You can also convert this to a PDF or another image format using standard image utilities.

4 Layout Versus Schematic (LVS) using Netgen

Netgen is a program for comparing circuit netlists. Normally it is used to verify circuit layout by comparing a netlist extracted from a circuit layout to a specification netlist that is known to be functionally correct (e.g., from schematic entry and simulation). This kind of verification is commonly known as LVS (layout versus schematic). You will use Netgen to compare the netlist generated from xcircuit (based on the schematic) with the one generated from Magic by ext2spice (based on the layout). The full documentation on Netgen is available at:

`http://opencircuitdesign.com/netgen/`

4.1 Running Netgen

Netgen takes two files in spice format as its input and compares whether these are the same. You must first output your spice file from xcircuit using information in the previous tutorial. This will have a suffix of **.spc**. Magic will generate a spice file as well with the suffix **.spice**. These all use the standard spice syntax from the last tutorial.

To extract a magic layout:

```
:set VDD vdd
:set GND gnd
:extract
:ext2spice scale off
:ext2spice
```

The first two commands are usually executed in user's system .magirc file, so you may only need them if you use a different name for your supplies. The default is "vdd!" and "gnd!". This will create your design with the .spice suffix.

You can start the Netgen shell by running:

```
netgen
```

or

```
netgen -noconsole
```

if you don't like the GUI shell (ick!). Once you are in the TCL shell, it leaves you at the TCL prompt "%". To quit, type quit or Ctrl-d.

You can also run a TCL script that has all of your commands. For example, if you create a script called "hw3.tcl", you can run it with:

```
netgen -noconsole source hw3.tcl
```

You could also run netgen and type the source command at the prompt. If you want the script to exit netgen at the end, you should have the last command be quit.

4.1.1 Running LVS

The short way of running LVS is with the following command in netgen:

```
lvs file.spc file.spice  [setupfile] [logfile]
```

where file.spc is the xcircuit spice file and file.spice is the Magic extracted spice layout. The optional setup file defaults to "setup.tcl" if ommitted and is ignored if there is no such file[1]. The logfile is an optional parameter which will default to "comp.out" if ommitted. However, this assumes all default configuration options and won't work for our homework. I recommend to create a TCL file for each design to keep the options together in one script

By default, the above LVS command will read the file and run LVS. However, sometimes you will need to read the files, set some options, and then run LVS. You can put these in the setup.tcl file, but the options sometimes need the names of the circuit, devices, etc. It is my preference to have a self-contained file so that there is no chance for human error in re-running LVS. (Also, you will need to submit your LVS file with your assignment!)

If you want to reference a subcircuit in the files rather than the top-level of the spice file, you can do this with {design.sp NAND3X1}, for example. This will use subcircuit NAND3X1 in the file design.sp.

You can read files, perform setup commands, and run LVS like this, for example:

```
lvs file.spc {file.spice INVX1} INVX1.tcl INVX1.out
```

This will use the setup in INVX1.tcl and save the LVS results in INVX1.out.

4.1.2 Devices and permutations

You can ignore classes of devices from an LVS using the ignore command like this:

```
ignore class c
```

Capacitors, and sometimes resistors, are created in the extracted spice netlist to represent the parasitics that give you more accurate simulations. However, if you don't do this it will complain about missing capacitors in your schematic! If you are creating an analog design that has resistors or capacitors, you will not want to do this.

You can also tell Netgen whether your devices are symmetric or not.

```
permute transistors
```

Without the argument, resistors and transistors will be permutable when you run this command. Since transitor layout is symmetric, this is a good option.

[1]This is actually not quite true. If you set some options and don't have a setup.tcl file, it will try to reinitialize some of the options. In general, I create an empty setup.tcl until Tim fixes this bug in Netgen!

4.1.3 LVS Procedure

By default, the lvs command will go through your design starting at the bottom-most level of hierarchy. If there is a circuit name that matches in your layout and schematic, it will compare them. If there is not a matching name, it will flatten the circuit into the next level of hierarchy and try to perform LVS at a higher level. It is important that names match or, if not, you must tell LVS what subcircuits correspond. Capitalization matters!

In the output, Netgen produces certain information about the circuits (e.g., number of devices, number of nets) and tries to match all the nodes. You will get output like the following from an LVS run:

```
Contents of circuit 1:  Circuit: 'INVX1.spc'
Circuit INVX1.spc contains 2 device instances.
  Class: pfet                  instances:   1
  Class: nfet                  instances:   1
Circuit contains 4 nets.
Contents of circuit 2:  Circuit: 'inverter.spice'
Circuit inverter.spice contains 2 device instances.
  Class: pfet                  instances:   1
  Class: nfet                  instances:   1
Circuit contains 4 nets.

Circuit 1 contains 2 elements, Circuit 2 contains 2 elements.
Circuit 1 contains 4 nodes,    Circuit 2 contains 4 nodes.
```

When you run a successful LVS, you also get an output (in comp.out, by default) like the following:

```
Subcircuit summary:
Circuit 1: INVX1.spc                       |Circuit 2: inverter.spice
-------------------------------------------|-------------------------------------------
nfet (1)                                   |nfet (1)
pfet (1)                                   |pfet (1)
Number of devices: 2                       |Number of devices: 2
Number of nets: 4                          |Number of nets: 4
-----------------------------------------------------------------------------------------
```

This shows the number of pfet and nfet (or devices) in each circuit along with the number of nets.

The topology, which is the devices and how they are connected, matches if if you get a message such as:

```
Netlists match uniquely.
Result: Circuits match uniquely.
Logging to file "comp.out" disabled
LVS Done.
```

This means that all the transistors and interconnections are correct (but not necessarily the sizes). If not, you will get an error like:

```
Netlists do not match.
```

4.1.4 Graph errors

If the comparison fails, you will also get an error message such as Graphs do not match, Node mismatch, Property mismatch, etc. Graph and node errors mean that your transistors are not the same or are not connected together the same in both netlists. For example, the following shows a mismatch:

```
Subcircuit summary:
Circuit 1: INVX1.spc                       |Circuit 2: inverter.spice
-------------------------------------------|-------------------------------------------
nfet (1)                                   |nfet (1)
pfet (1)                                   |pfet (1)
Number of devices: 2                       |Number of devices: 2
Number of nets: 5 **Mismatch**             |Number of nets: 4 **Mismatch**
-----------------------------------------------------------------------------------------
NET mismatches: Class fragments follow (with fanout counts):
Circuit 1: INVX1.spc                       |Circuit 2: inverter.spice

-----------------------------------------------------------------------------------------
Net: Y                                     |Net: out
  nfet/(drain|source) = 1                  |  pfet/(drain|source) = 1
  pfet/(drain|source) = 1                  |  nfet/(drain|source) = 1
                                           |
Net: GND                                   |Net: in
  nfet/(drain|source) = 1                  |  pfet/gate = 1
  nfet/bulk = 1                            |  nfet/gate = 1
                                           |
Net: Vdd                                   |Net: vdd
  pfet/(drain|source) = 1                  |  pfet/(drain|source) = 1
  pfet/bulk = 1                            |  pfet/bulk = 1
                                           |
(no matching net)                          |Net: gnd
                                           |  nfet/(drain|source) = 1
                                           |  nfet/bulk = 1
-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
Net: A                                     |(no matching net)
  pfet/gate = 1                            |
                                           |
Net: A1                                    |(no matching net)
  nfet/gate = 1                            |
-----------------------------------------------------------------------------------------
```

From this dialog, you can see that there are MORE nets in INVX1.spc (my schematic) than my layout. This most likely means something is disconnected. Similarly, if you have fewer nets, you likely connected something accidentally. If you look further, there is a listing of the iIllegal

fragments around areas where the netlists cannot be resolved. The worst matches will be listed at the top, which is usually the place to start looking.

First, LVS does NOT need pin names to match! It does a symbolic matching. However, it is convenient for debugging if they do match.

Second, the order of the fragments does NOT matter. It does not think that Y and out correspond, even if they do. Similarly, it does not think that GND and in correspond.

In each case, it shows the number of gates or drain—source (either one since we allow the devices to be permuted) connected to each net. In our example, you may be able infer some things. For example, our output in the schematic is Y and it is out in the layout (based on the number of connections). The vdd and gnd connections look the same. The outputs also look the same each being connected to a transistor and a bulk.

The inputs nets, however, don't match. In our schematic, there are two nets, A and A1, each connected to one gate. The extracted circuit, however, has a single net named in that is connected to two gates. The problem is that our input schematic doesn't have the transistor gates connected!

Further debugging can be done with the nodes and elements commends. Connectivity of elements (instances) and nodes (wires or signals) can be traced with commands:

```
nodes <element_name> <cellname>
elements <node_name> <cellname>
```

where cellname is the filename (one of the two files loaded for comparison) or the name and subcircuit such as inverter.spice INVX1 if you aren't comparing the top level.. The elements command prints all of the elements (transistors, capacitors, resistors, etc.) connected to a specific named node. The nodes command prints the node names for each pin of the specified element (i.e., a device instance).

4.1.5 Property errors

If you get an error such as

```
Netlists match uniquely.
There were property errors.
```

then the topology matched but some of the transistors had properties that did not match. **LVS did not pass!** The devices and their connections were correct, but some of the devices were not exactly the same due to some parameter differences. A property is typically a transistor width (W), length (L), or the number of fingers (M). Netgen will match these by default within a tolerance of 1%. If there is an error on these parameters, you need to fix your layout or schematic. This ensures that your simulated sizes are the same as your layout sizes.

4.1.6 Missing properties

By default, magic will also output four properties for extracted circuits that are not in your xcircuit spice file: as, ad, ps, pd. Netgen will complain about this with messages like this:

```
No property as found for device nfet
No property ad found for device nfet
```

These are the area and perimeter of source and drain and are used for capacitance estimation in spice using layout information. We don't have this information in our schematic, because there is no layout yet!

You can remove extra properties with some extra setup.tcl commands:

```
property {inverter.spice nfet} remove as ad ps pd
property {inverter.spice pfet} remove as ad ps pd
```

which tell it to remove these properties from comparison. Note that the device is in a given netlist/file, so you need to specify that.

4.1.7 Different model names

Sometimes you may have different model names from your layout and schematic. If you followed the instructions I gave for xcircuit, you should have made these the same, but there are other cases sometimes. (Please ensure these are the same for this assignment!) For example, xcircuit could output pfet and nfet, while a different extraction tool may output p and n for your transistors. You can add options to equate these in netgen:

```
equate class {file.spice nfet} {file.sp n}
equate class {file.spice pfet} {file.sp p}
```

where the file.sp is the extracted layout from another tool.

4.2 LVS Debugging Tips

LVS is rarely wrong. (It is possible there is a bug in the LVS tool, but likely there isn't.) If your circuit doesn't match, it is for a reason. All designs should match in this class for full credit. It is your responsibility to figure out why it doesn't match and fix the problem in your final submission.

In order to debug your design, you should consider these hints that can find common LVS problems:

1. Isolate the problem. You must have all of your sub-circuits match LVS to get the top-level to match.
2. Make sure that your supply rails are labeled so that they are automatically connected in extraction. (It is case sensitive...)
3. Make sure the number of devices (transistors of each type) match. Different fingers of transistors can cause problems in some tools since this is a special "M" parameter on spice transistors.
4. Make sure the number of nets match. If you have more nets, something is likely disconnected. If you have fewer nets, something is likely shorted.
5. Make sure that the body of the transistors are connected to the appropriate supply rail. Body connections are typically special and dealt with in unique ways by different tools.
6. Check if a bus has the bits flipped (MSB connected to LSB, etc.)

There are numerous tutorials at: `http://opencircuitdesign.com/netgen/tutorial/tutorial.html` which delve further into debugging and configuration options.