

Open-Source Analog Design Flow Using Efabless and the SkyWater 130 nm PDK

Documentation Created By: Joshua Thater
Information Current as of 11/27/2023

Table of Contents

| | |
|---|-----------|
| Table of Contents | 1 |
| Preamble/Introduction | 2 |
| Guide to Setting up a Virtual Environment for Analog Design Flow | 3 |
| Guide to Setting up Tool Environment for Analog Design Flow | 10 |
| Required Packages..... | 11 |
| Magic VLSI..... | 12 |
| Xschem..... | 15 |
| Ngspice..... | 17 |
| GAW (optional)..... | 19 |
| Netgen..... | 21 |
| Open PDKs Install..... | 23 |
| Tool Usage + Example “Hello World” Type Project | 25 |
| Schematic Capture/Creation + Simulation..... | 27 |
| Example Inverter Schematic Creation and Simulation..... | 29 |
| Layout Creation + DRC/LVS..... | 44 |
| Example Inverter Layout + LVS..... | 46 |
| Parasitic Netlist Extraction + Post-Layout Simulation..... | 60 |
| Example Inverter R+C Extraction + Post-Layout Simulation..... | 60 |
| Integration of Design with Harness Through Efabless..... | 66 |
| Example Inverter Harness Hookup..... | 66 |
| BONUS: ReRAM Setup | 78 |

Preamble/Introduction

This guide aims to teach the open-source analog design flow integrating with Efabless. This process is currently not well documented and is very complicated for beginners. This guide will hopefully try to remedy that.

Analog design is complicated enough with commercial tools, but it becomes even more complicated with open-source tools. With Cadence, we are able to do all steps of the analog design flow in one tool. However, in the open-source design flow, there is a tool for each step of the process. The core design flow for open-source analog design is as follows:

1. Schematic Capture/Creation - (Tools: Xschem)
2. Simulation of Schematic - (Tools: Ngspice, Xspice, Xyce)
3. Layout Creation - (Tools - Magic, Klayout)
4. LVS check - (Tool: Netgen)
5. Post-Layout Simulations (Tool: Xschem & Ngspice/Xyce)

These are the tools that allow for open-source analog design, but you will still need a library/PDK so that you have devices to create your components and rule sets to follow in layout. This is where Open PDKs come in handy. Open PDKs define a standard set of libraries and files that will allow you to design circuits using a PDK. Open PDKs mainly support two PDKs - the SkyWater 130 nm process and the GF180MCU process. This guide will cover the setup and rules for the SkyWater 130 nm process, but the GF180MCU process should also work - albeit with slightly different rules.

This guide will try to be as explanatory as possible on all aspects of the open-source analog design flow - including a very detailed walkthrough example. As a result, this guide will be very long and perhaps a little long-winded in some places. However, my goal with this guide is to ensure that you have a solid understanding of the tools, the analog design flow, and how everything fits together so that you can focus on your designs - instead of everything leading up to them. This guide is also very beginner-friendly, so a lot of basic commands will be shown. This is so that people with little to no terminal experience can still learn to install and use these tools.

In this guide, I will detail what has worked for our team and try to show alternatives when available. I will also try my best to link out to outside sources, other documents, and other tutorials as well. However, this guide will try and centralize all of this information into one place. I hope that you find this guide helpful, and I also hope that you continue to add and edit this guide for future teams.

One final note: please join this Slack channel and don't be afraid to ask questions, as there are very smart people in there:

<https://open-source-silicon.dev>

Guide to Setting up a Virtual Environment for Analog Design Flow

This will provide a general guide on how to set up a virtual environment that will be able to house all the tools that are needed for the open-source analog design flow.

To start things things off, in order to do the analog design flow, it is *heavily recommended* to use a Linux system or MAC system. It may be possible to use a Windows operating system, but certain software may be out of date or not available at all.

This entire guide will assume you are using a Linux system - specifically an Ubuntu system.

I recommend using a VM running Ubuntu since, in my experience, it is the easiest to work with, and it is what my team has been using during our time working on our senior design project. If you have another preference, go with that, but just remember that this guide is specifically made for people using an Ubuntu system. So, you might have to run different commands, tweak different settings, and set up other dependencies to get things working.

The rest of this section will cover how to download and install a Ubuntu VM with the specific hardware requirements needed for this analog design flow. If you do not plan to use an Ubuntu VM for your environment, then you can safely skip this section. However, be sure that your environment will be able to run all the tools that are needed.

I recommend using Virtual Box for your VM. Below is the link to download Virtual Box:
<https://www.virtualbox.org/wiki/Downloads>

Once you have Virtual Box downloaded, it is time to set up the Ubuntu VM. First, head to the webpage linked below. Ubuntu Desktop Virtual Machine Set Up:
<https://ubuntu.com/download/desktop>

Once there, download an Ubuntu image. It shouldn't matter which one you choose; just make sure it is past Ubuntu 20 (which all of them on the download page should be anyway). For this example, I will be using Ubuntu 22.04.3, which can be seen below.

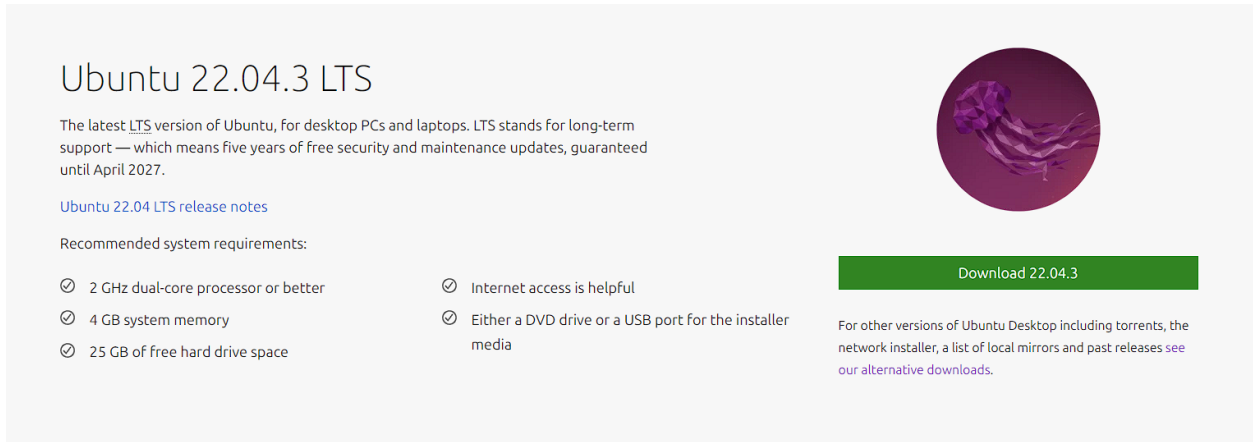


Figure 1: Ubuntu Image

This ISO is a few GB, so it may take a bit to install. Once it is installed, you can head over to Virtual Box. Once in Virtual Box, you should see a screen similar to the one seen below (you will not see the VMs that I currently have set up). To begin setting up the virtual machine, click on “New” as seen in the figure below.

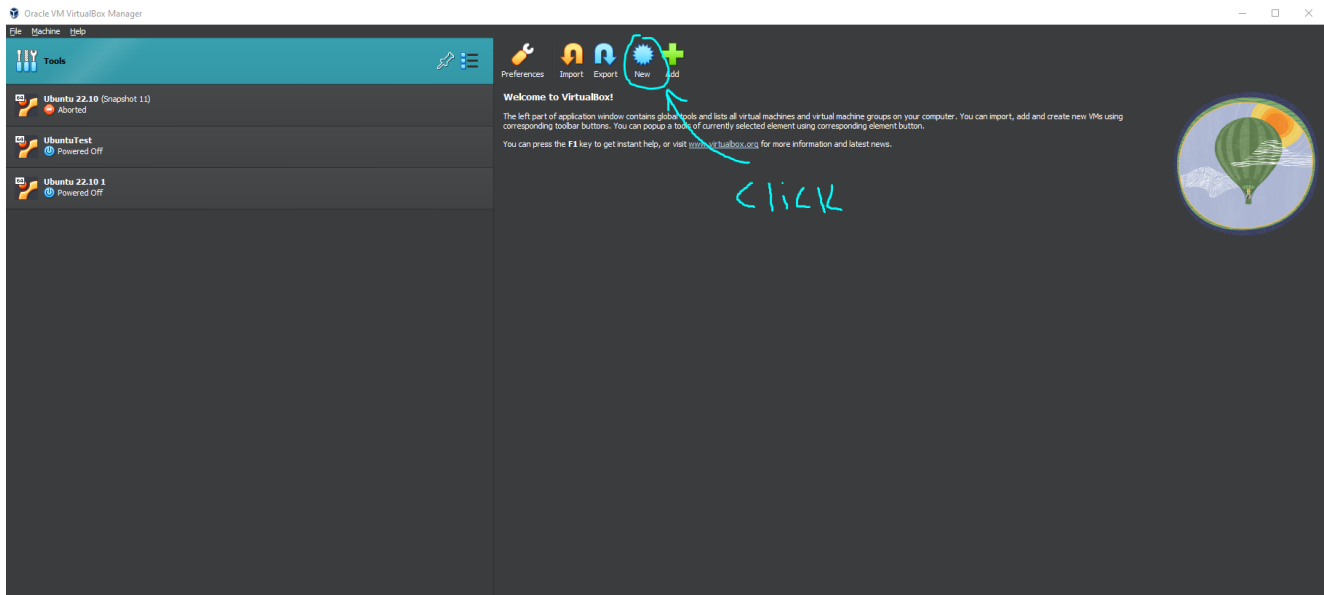


Figure 2: Virtual Box Setup

Once that is done, simply give a name to your virtual machine and select the ISO image you just downloaded (the ISO should be in your downloads directory). If you did all that correctly, it should look something like this.

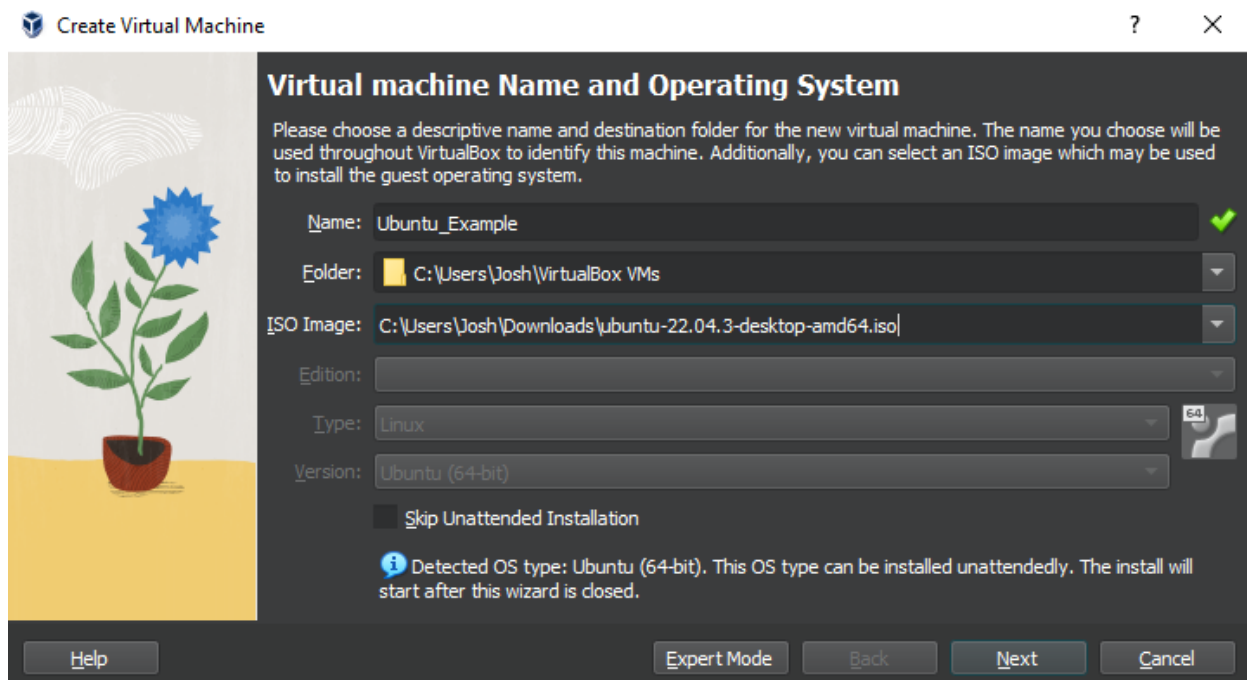


Figure 3: VM Name

Once this is done, click next. On the following screen, you can change the username and password of your machine. It is recommended that you change these settings so that you can get sudo access on your machine. I would not worry about changing the hostname or domain name. Finally, on this screen, make sure to select the box for “Guest Additions”; this adds a bunch of quality-of-life features, such as dynamic screen resolution. Once this is all done, it should look something like this.

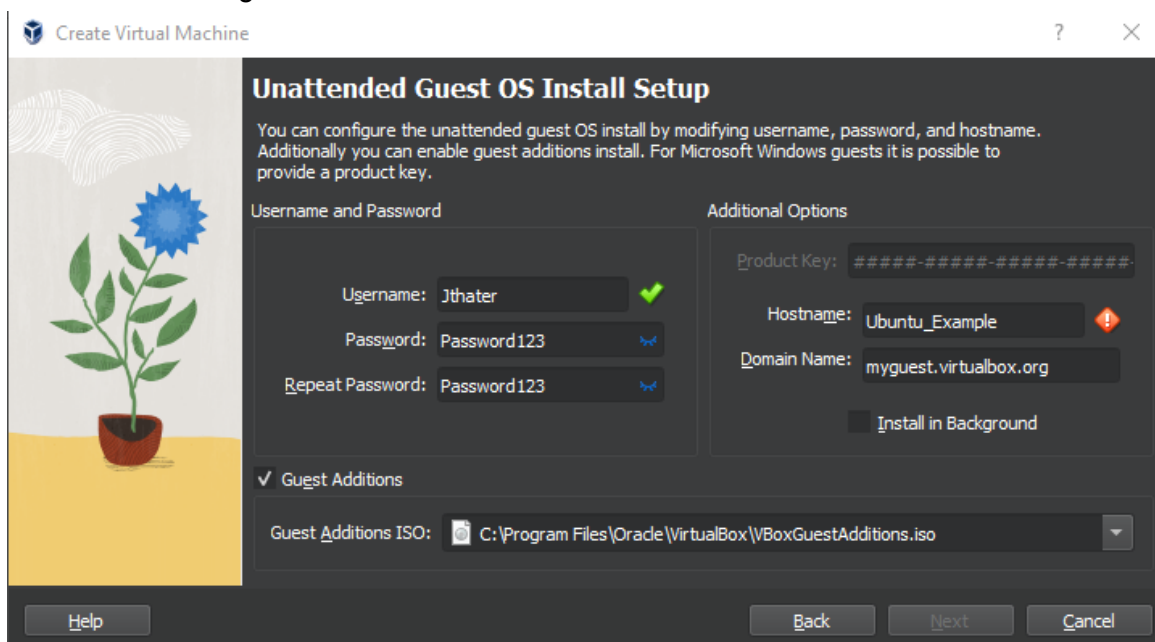


Figure 4: VM User Setup

Once you click next, you will be greeted with resource allocation. This is gonna be up to your system, but it is recommended to allocate around 8GB of RAM and 4 CPUs. However, it is more important to stay within the green so that your host machine will still be stable. This is what my settings look like.

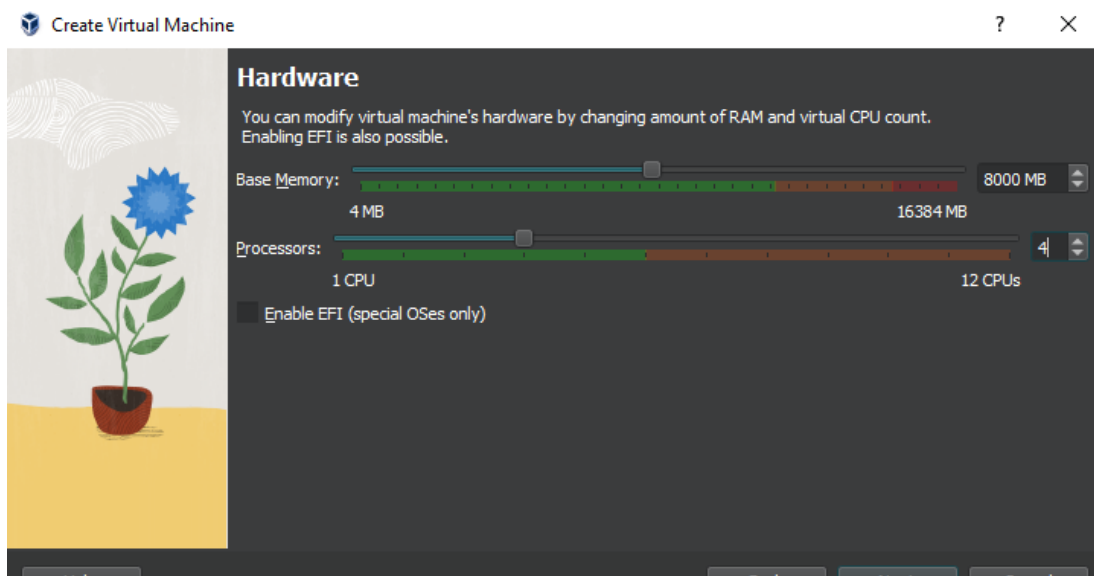


Figure 5: VM Hardware Allocation

Once you click next, you will be greeted with setting up the virtual hard disk. For the environment that will be set up, I would recommend a **minimum of 75 GB**, but depending on what you end up doing, you may want up to 120 GB allocated. For this example, I am allocating 100 GB of space. You may also opt to pre-allocate the full size, but I leave this setting turned off.

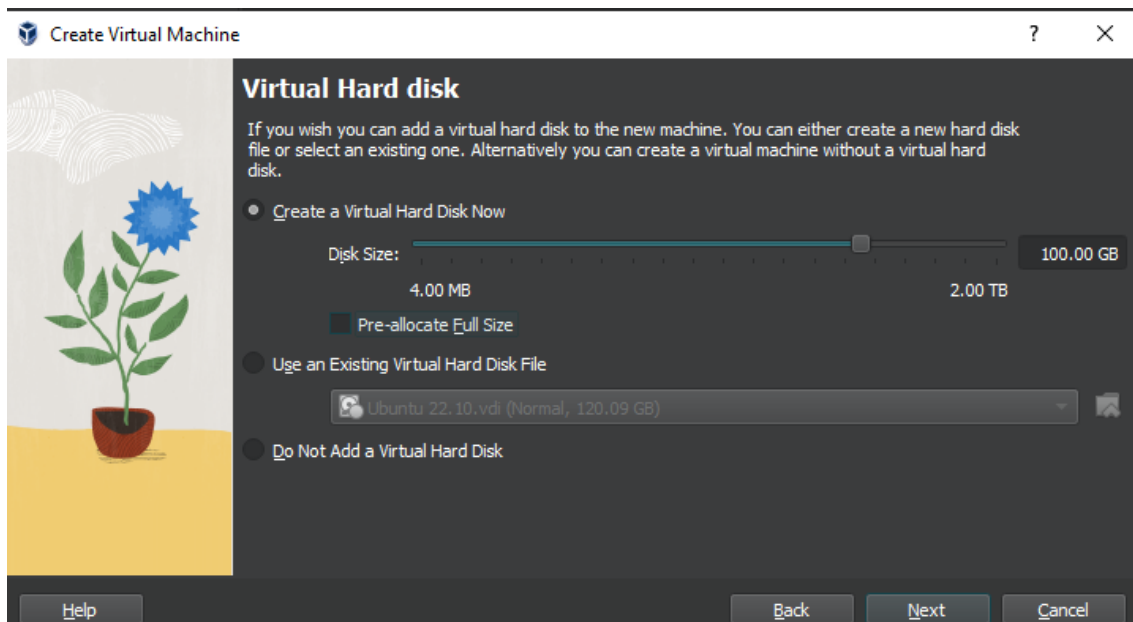


Figure 6: VM Hard Disk Allocation

Once all of that is done, you will see a summary page. Ensure that everything is set up correctly, and click “Finish.”

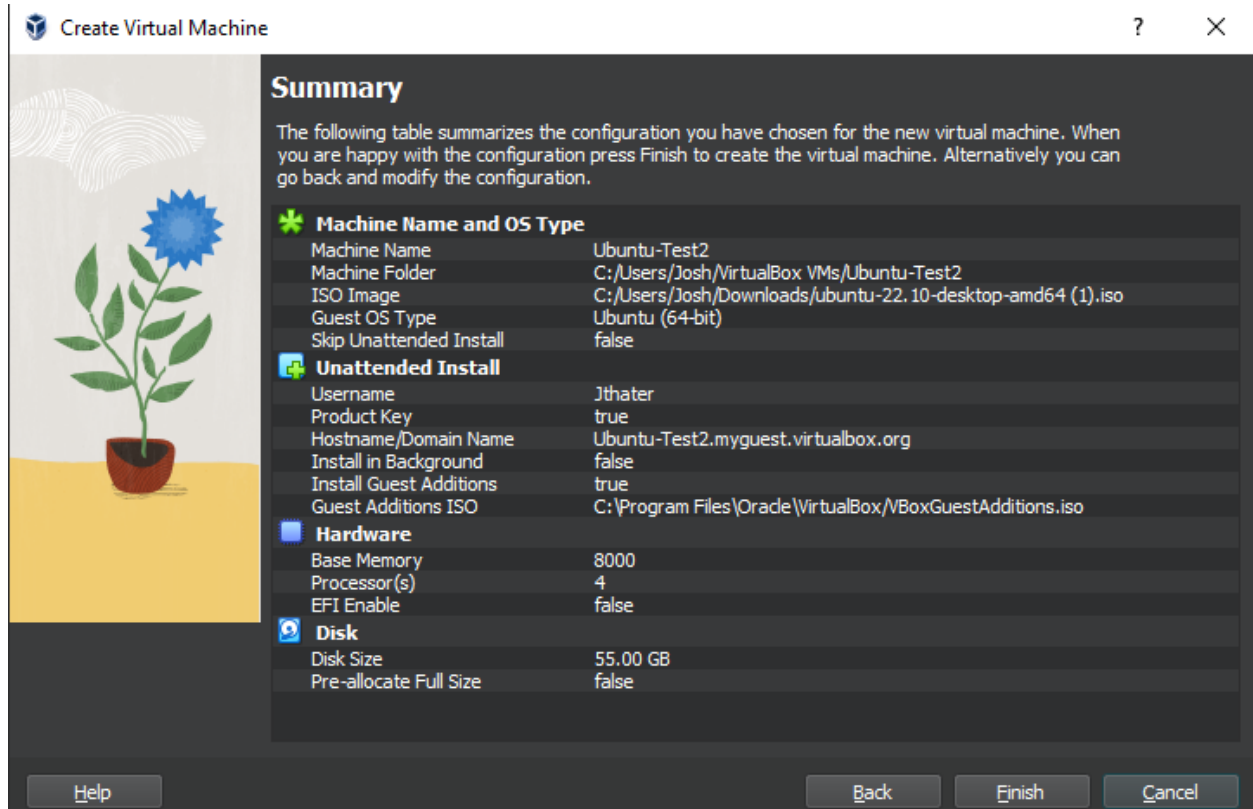


Figure 7: VM Setup Summary

Once you hit “Finish”, Virtual Box will start running the machine and will begin downloading the ISO. It will take a few minutes to get everything fully set up. Once it is done downloading, it should kick you to a login screen. It should look something like this:

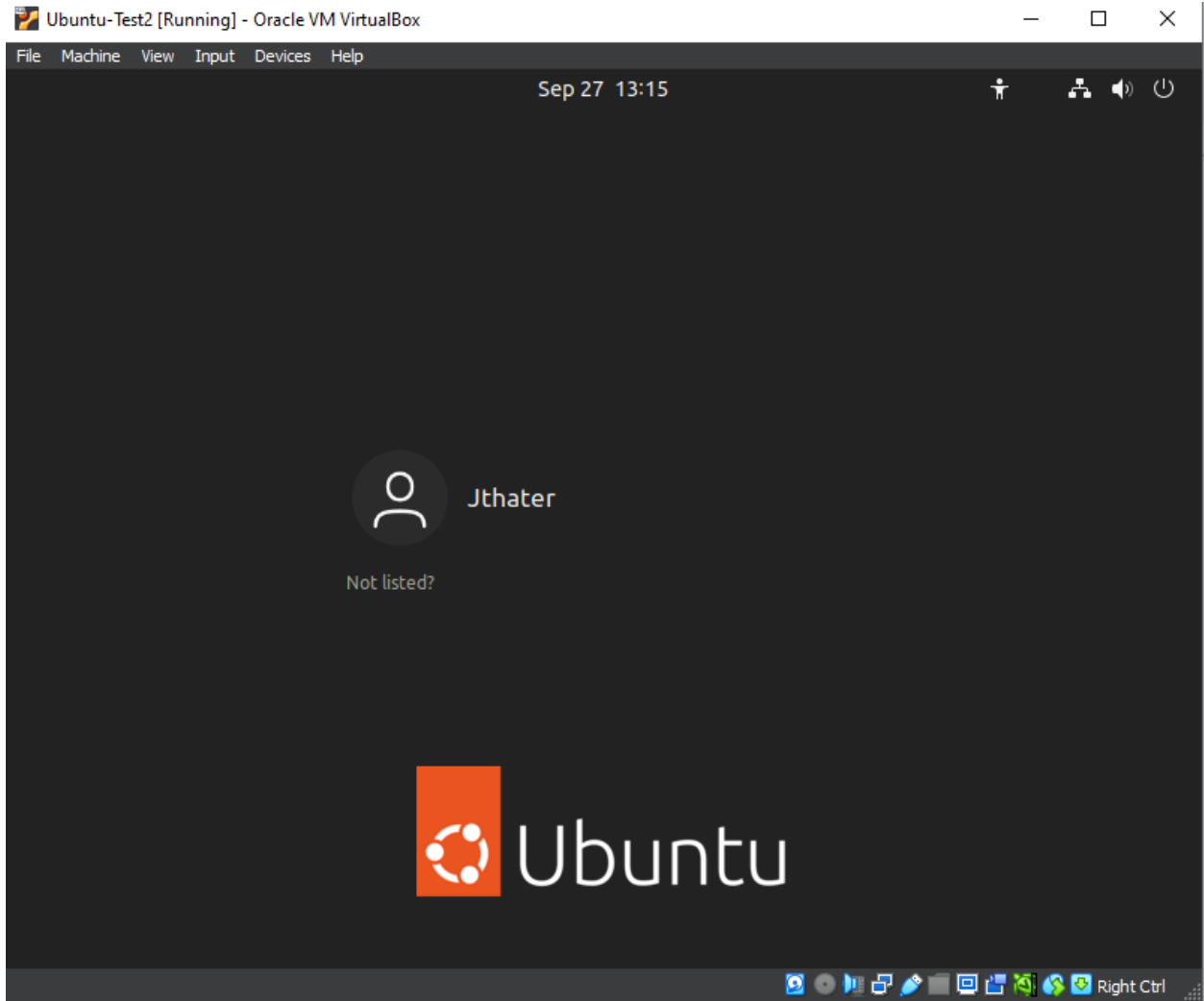


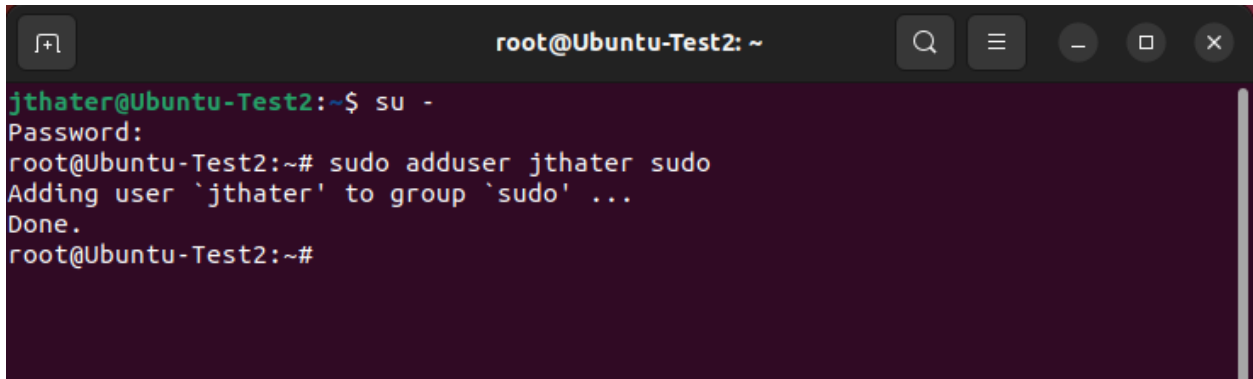
Figure 8: VM Setup Login

Go ahead and log in with your password. The last step is to ensure that the guest additions are downloaded correctly. Go full screen, and if the virtual machine also goes full screen, that means the guest additions have downloaded correctly! (NOTE: It seems that the virtual machine will not go full screen on 1440p monitors, only 1080p and smaller resolutions). With all of this done, you have successfully set up the virtual machine environment!

Also note, for some reason, sometimes you will not be added to the “sudoers” file. There are many ways to add yourself to this file, but I will outline a way that worked for me.

- Open up the terminal and type: **su -**
- Enter your password
- Type: **sudo adduser [your username] sudo**

If you did this correctly, you should see a message that you have been added to group sudo. Your terminal should look something like the figure below.

A terminal window titled "root@Ubuntu-Test2: ~" with standard window controls. The terminal shows a user named "jthater" switching to root via "su -", then running "sudo adduser jthater sudo". The output indicates the user was successfully added to the sudo group.

```
root@Ubuntu-Test2: ~
jthater@Ubuntu-Test2:~$ su -
Password:
root@Ubuntu-Test2:~# sudo adduser jthater sudo
Adding user `jthater' to group `sudo' ...
Done.
root@Ubuntu-Test2:~#
```

Figure 9: VM Sudo Setup

You can test if this worked by exiting out of the terminal, opening it back up, and typing something like “sudo ls.” If you don’t get an error, then you have successfully added yourself to the sudo group and can run commands as root.

NOTE: you may have to reboot the system to see the change go into effect (power off the machine).

Guide to Setting up Tool Environment for Analog Design Flow

Now that you have set up your virtual machine environment, it is time to begin downloading the tools needed for the analog design flow. It should be noted that since this whole process is still essentially in “alpha,” things are changing quite a bit, so by the time you read this, some of this information may be outdated. I recommend future teams to update this information as it changes.

This part of the guide will just give a brief overview of the tools and how to install them; it will not go in-depth about their use and how to use them to go through the analog process flow. This information can be found in the next section of this document.

One last reminder that the rest of this guide is written assuming you are using an Ubuntu system. If you are not running an Ubuntu system, then you will have to figure out how to run these commands on your own system.

Before starting to download things, it is probably a good idea to update/upgrade your “*apt-get*” command as it may not be the most recent after a new download of a virtual machine. To do this, run the commands:

- ***sudo apt-get update***
- ***sudo apt-get upgrade***

If you run these commands and they give you errors, look up the error numbers you are getting, but it most likely means that you are running an outdated version of Ubuntu and need to upgrade your machine. I won’t go into that here, but there are plenty of guides on how to do this.

Once these commands are run, you should have no problem using “*sudo apt-get install [xxx]*” to download packages and dependencies for these tools.

Required Packages

In this section, I will list out some needed/useful packages that you will want to download for your use in this environment. These packages are not dependencies for the software you will be downloading, but they are helpful to have as you work. The first package you will need is the Git package. This will allow you to clone repositories, which is vital for installing some of the software. To install Git, simply type this command in the terminal:

- **sudo apt-get install git**

The next package you want to install is some type of text editor that will allow you to read and edit files within Linux. There are many text editors to choose from, but I recommend Vim. To install Vim, simply type the following command:

- **sudo apt-get install vim**

Magic VLSI

The first tool that you want to download and configure will be a tool called Magic VLSI. This tool is essential for creating a lot of libraries and will have you download many of the dependencies that will be needed for other tools. Magic will be a tool that you can use for your layouts and to perform device extractions for LVS and post-layout parasitics. It can also generate GDS and LEF files.

Documentation of Magic can be found here:

<http://opencircuitdesign.com/magic/index.html>

This is the page that has all the information you would want to know about Magic, so look around.

In order to install Magic, navigate to the install page. Here, you will find instructions on the dependencies, configurations, and setup of this tool. For the sake of simplicity, I will list all of the dependencies and the commands to get them for an Ubuntu system. Some of these dependencies are optional, depending on how you want to configure Magic. However, I recommend getting all of them because they do not take up much space, and some of the other software will need these dependencies installed anyway. These dependencies and commands to get them are as follows:

- **sudo apt-get install m4**
- **sudo apt-get install tcsh**
- **sudo apt-get install csh**
- **sudo apt-get install libx11-dev**
- **sudo apt-get install tcl-dev tk-dev**
- **sudo apt-get install libcairo2-dev**

Optional:

- **sudo apt-get install mesa-common-dev libglu1-mesa-dev**
- **sudo apt-get install libncurses-dev**

NOTE: You can download all of these dependencies at once by typing them all in a single line. Such as:

- **Sudo apt-get install m4 tcsh csh libx11-dev tcl-dev tk-dev libcairo2-dev**

Once you have all of the dependencies downloaded successfully, you should be able to compile and install Magic successfully. I recommend installing it from the GitHub repository as it contains the most recent version. To do this, simply clone the git repository using the command:

- **git clone <https://github.com/RTimothyEdwards/magic>**
- **ls**

This will clone the GitHub repository into a directory on your virtual machine. This directory should show up in your current directory. You should be able to run the “ls” command to see it. The figure below illustrates this cloning and where the repository got cloned.

```
jthater@Ubuntu-Example:~$ git clone https://github.com/RTimothyEdwards/magic
Cloning into 'magic'...
remote: Enumerating objects: 12009, done.
remote: Counting objects: 100% (2069/2069), done.
remote: Compressing objects: 100% (887/887), done.
remote: Total 12009 (delta 1174), reused 1903 (delta 1030), pack-reused 9940
Receiving objects: 100% (12009/12009), 9.88 MiB | 25.48 MiB/s, done.
Resolving deltas: 100% (7390/7390), done.
jthater@Ubuntu-Example:~$ ls
Desktop    Downloads  Music      Public     Templates
Documents  magic      Pictures   snap       Videos
jthater@Ubuntu-Example:~$
```

Figure 10: Magic Repository Cloned

Once you have confirmed that you have cloned the Magic repository successfully, simply move into the directory using “cd”. I.e.:

- **cd magic**

Doing this will move you into the Magic directory, and from here, you can manually configure and install Magic. To configure Magic, simply type the following commands:

- **./configure --enable-cairo-offscreen**
- **make**
- **sudo make install**

With this specific configuration it should help to alleviate any glitches that may occur with the software when using the OpenGL graphics interface if you choose to use it (read more about this in the Magic documentation).

It should be noted that the default path for the installation places all of the files in the directory /usr/local/bin/magic.

Once this is done, you can verify that you have installed Magic successfully by simply typing “magic” into the console. You should see the Magic tool come up, and you should be able to navigate around it. The figure below shows what you should be seeing.

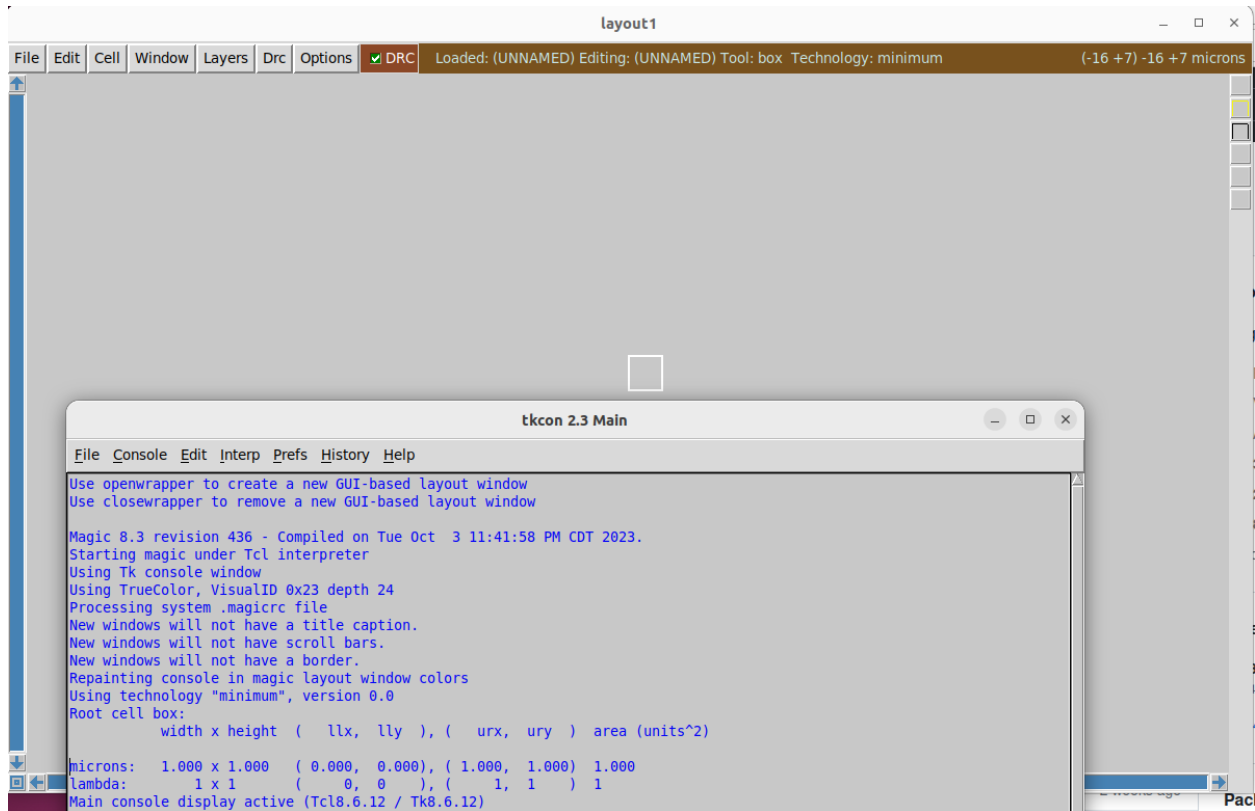


Figure 11: Magic Startup Screen

If you are able to open Magic with no issue, then you have successfully installed the software and are ready to move on to the next step.

Xschem

The next tool that you will be downloading is Xschem. Xschem is a schematic editor that allows you to create schematics, symbols, test benches and will create SPICE netlists for testing. I recommend this over other open-source schematic editors, such as Xcircuit, because it feels and operates closer to commercial-grade tools (like Cadence).

Documentation for Xschem can be found here:

<https://xschem.sourceforge.io/stefan/index.html>

Just like with Magic, this page will contain all information about how to install and configure Xschem. The documentation on how to use this software is very in-depth, so I highly recommend going through and reading the documentation once you get everything set up.

This software also requires some dependencies to be installed, but luckily, most of them have already been installed when Magic was installed, so only a few more need to be added. To get the rest of the dependencies, simply type:

- **sudo apt-get install flex**
- **sudo apt-get install bison**
- **sudo apt-get install libxpm-dev**

With all of these dependencies installed, Xschem can now be configured and installed. If you are following this guide section by section, then make sure you are back in your home directory before running this next step. This can simply be done with:

- **cd**
 - This will move you back to your home directory and should be done each time you begin installing new software

Now, simply clone this repository using the command:

- **git clone <https://github.com/StefanSchippers/xschem>**

Just like with Magic, if you did everything properly, you should be able to “ls” and see the Xschem directory. To install this software, simply type the commands from your home directory:

- **cd xschem**
- **./configure**
- **make**
- **sudo make install**

The default installation path of this software should be `/usr/local/bin/xschem`. NOTE: You can change this default directory when you run the `./configure` command if you so choose.

To ensure that everything is installed correctly, simply type the command `xschem` into the terminal, and you should be greeted with a similar-looking screen, as seen in the figure below.



Figure 12: Xschem Startup Screen

If you see this screen, then you have successfully installed Xschem and are ready to move on to the next step.

Ngspice

The next tool that you will install is Ngspice. Ngspice is an analog/mixed-signal simulation tool. This will allow you to read in SPICE netlists (and, with more recent versions, more complex netlists) and run different types of circuit analysis on them, such as DC sweeps, transient simulations, AC tests, and many, many more. It also allows you to simulate digital design through Xspice, which is installed as part of this software.

The Ngspice documentation can be found here:

<https://ngspice.sourceforge.io/>

To get a fully featured Ngspice, there are a few dependencies that you must download. They include:

- **sudo apt-get install libxaw7-dev**
- **sudo apt-get install lib64readline8 libreadline-dev**
 - It should be noted that this dependency can be very finicky, so you may have to change the 64 to a 32, or there might be a new version, so you might need to change the 8 to a 9. I wouldn't worry too much if you can't get this to work, as it is highly unlikely that you will use readlines for Ngspice for anything that you will work on.

You can download the Ngspice software from a Git repository like the other software, but this is more complicated than downloading it from a tar file in this case. So, I would recommend downloading this software from the tar file found here:

<https://sourceforge.net/projects/ngspice/files/ng-spice-rework/41/>

Download that tar file and navigate to the downloads file from the terminal. This can be achieved by being in your home directory and typing the command

- **cd Downloads**

Once in this directory, type `ls`, and you should see a file named `ngspice-41.tar.gz` or something similar. Once you have confirmed the file is there, type the following command to unzip the file.

- **tar -xvzf ngspice-41.tar.gz**

If you've done everything correctly, it should look something like the figure below.

```

jthater@Ubuntu-Example:~$ cd Downloads/
jthater@Ubuntu-Example:~/Downloads$ ls
ngspice-41.tar.gz
jthater@Ubuntu-Example:~/Downloads$ tar -xvzf ngspice-41.tar.gz
ngspice-41/

```

Figure 13: Unzipped Ngspice File

Unzipping the file will result in a bunch of lines being output to the terminal. Once everything is done, I recommend moving the file into your home directory. This can be done with the command:

- **mv ngspice-41 ..**

If done correctly, you can navigate back to your home directory, and you should be able to use “ls” to see it. Once you have confirmed that the file is there, you can run the following commands:

- **cd ngspice-41**
- **mkdir release**
- **cd release**
- **./configure --with-x --enable-xspice --disable-debug --enable-cider --with-readlines=yes --enable-predictor --enable-osdi --enable-openmp**
 - Must configure with osdi in order to support ReRAM simulations using Ngspice
- **make 2>&1 | tee make.log**
- **sudo make install**

Running these commands will take longer than any of the setups for the other software, so be prepared to wait a few minutes for this installation. To ensure that the download has been successful, simply type “ngspice” into the terminal. If you do so, you should see something similar to the figure seen below.

```

jthater@Ubuntu-Example:~$ ngspice
*****
** ngspice-41 : Circuit level simulation program
** The U. C. Berkeley CAD Group
** Copyright 1985-1994, Regents of the University of California.
** Copyright 2001-2023, The ngspice team.
** Please get your ngspice manual from https://ngspice.sourceforge.io/docs.html
** Please file your bug-reports at http://ngspice.sourceforge.net/bugrep.html
** Creation Date: Wed Oct 4 06:54:37 UTC 2023
*****
ngspice 2 ->

```

Figure 14: Ngspice Startup Screen

If you see something similar to this screen, then you have successfully installed Ngspice and are ready to move on to the next software.

GAW (optional)

GAW is a way to view the waveforms that are generated from Xschem. There are many options to view waveforms, including through Xschem directly, through Ngspice, or other waveform viewers. I recommend looking up waveform viewers yourself and finding something that fits your needs. I only include GAW since it is the one that I have worked with and am used to.

To download GAW, go here and download the latest version:

<https://download.tuxfamily.org/gaw/download/>

To download GAW, you will need this dependency:

- **sudo apt-get install libgtk-3-dev**

Optional:

- **sudo apt-get install xterm**
 - This is an emulator that will allow a Ngspice terminal to open when doing simulations in Xschem. This will allow you to use Ngspice to view waveforms. I recommend installing this as well.

Then, from your home directory, enter the following commands:

- **cd Downloads**
- **tar -xvzf gaw3-20220315.tar.gz**
- **mv gaw3-20220315 ..**
- **cd**
- **cd gaw3-20220315**
- **./configure**
- **make**
- **sudo make install**

To confirm that GAW has been installed correctly, type gaw into the terminal, and you should see something similar to the figure below pop up.

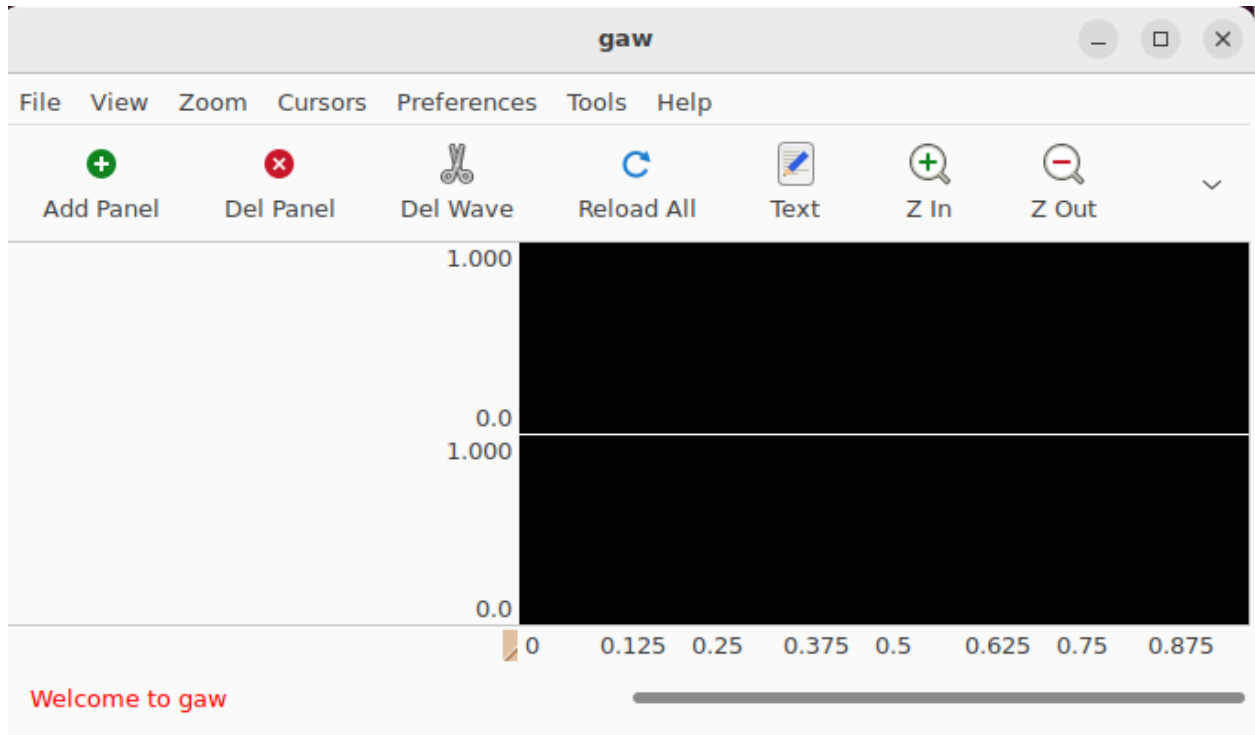


Figure 15: GAW Startup Screen

Netgen

The next tool that must be installed is Netgen. Netgen is a tool that will be able to perform LVS on two netlists, whether it be SPICE or Verilog.

Documentation for Netgen can be found here:

<http://opencircuitdesign.com/netgen/index.html>

This documentation page is much like Magic (as it is made by the same person). There are guides on how to install this software, and there are some tutorials on how to use the software. It's a good idea to look around in the "Tutorials" and "References" pages for very detailed information about all of the different things Netgen can do.

The installation of Netgen is very simple. Make sure you are in your home directory before performing this next command. Simply copy the Netgen GitHub repository using the command:

- **git clone** <https://github.com/RTimothyEdwards/netgen>

Once you have successfully cloned the repository, enter into the Netgen directory and run the following commands.

- **./configure**
- **make**
- **sudo make install**

If you have successfully installed Netgen, then simply type "*netgen*" into the terminal, and you should see something similar to what is shown below.

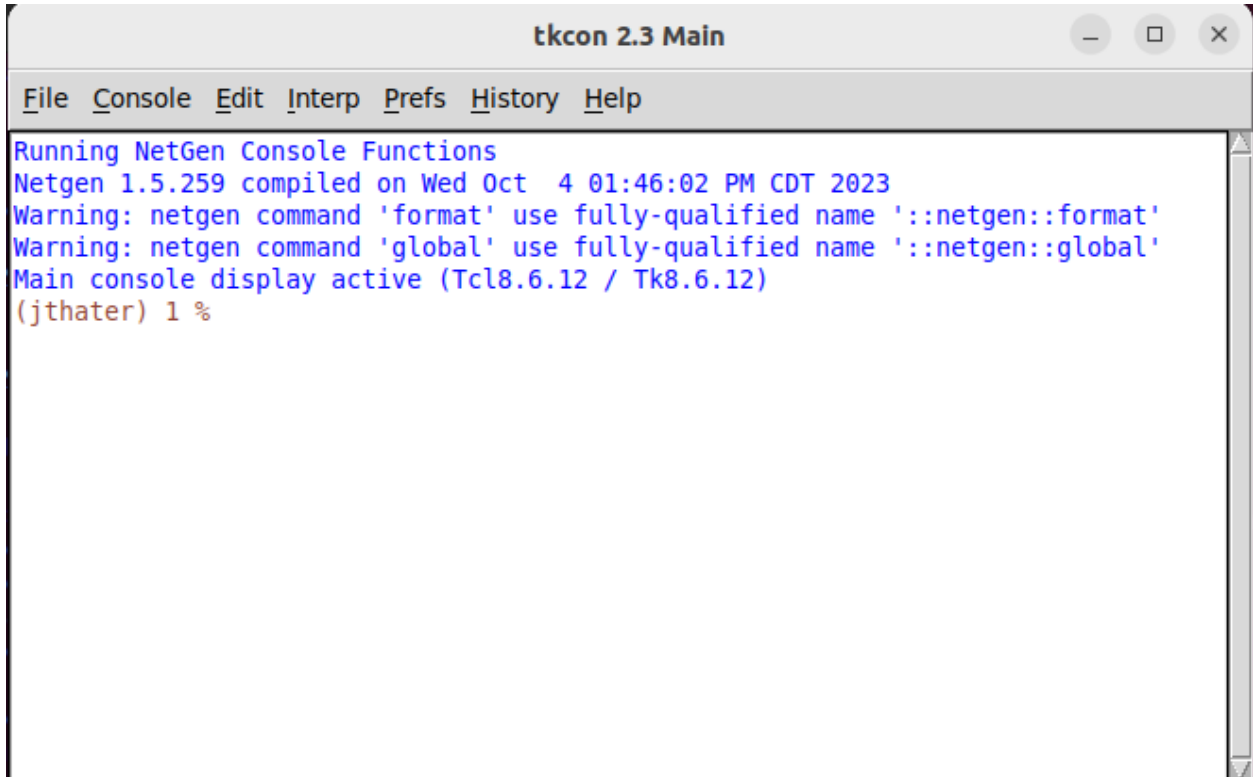


Figure 16: Netgen Startup Screen

Once you have verified that you have installed Netgen, you are ready to move on. If you are planning on doing simple analog designs, then you can most likely stop installing software here and move on to the Open PDKs Install section. The next two software are optional.

Open PDKs Install

The next install is the PDK that you will be designing devices with. As mentioned earlier, the installation will be done through Open PDKs as it has simplified the process of configuring the PDK for open-source tools and files. All you have to do is clone the repository, configure your installation for whichever PDK you want to download - SkyWater or GF180 MCU - and install the files, cells, and ruleset for that PDK. For this installation, I will be assuming that you are installing the SkyWater 130 nm PDK.

Documentation for Open PDKs can be found here:

https://opencircuitdesign.com/open_pdk/

Clicking through the different tabs will walk you through configurations and installation steps for your respective PDK. This page, however, does not contain much information about the specific PDK you are downloading (like the different cells, rules, etc.).

To find this information, navigate to the documentation for the SkyWater 130 nm process, which can be found here:

<https://skywater-pdk.readthedocs.io/en/main/>

I heavily recommend reading through some of the documentation for the SkyWater 130 nm PDK, so you get an idea of what you are downloading and the different cells and primitive devices that are available to you.

Once you have read through a little bit of the documentation, you can download and install Open PDKs. This will operate much in the same way as all of the other installations from the source code in GitHub. However, this install has many different configurations depending on which libraries from the PDK you want installed or not. Looking through both the SkyWater documentation as well as the install page on Open Circuit design should show you how to specifically configure these installs for you. It should be noted that you can always add libraries later if you find out you need something from them.

For most general users, the standard build of the SkyWater 130 nm PDK should be fine. To do this, simply type this into the terminal. NOTES: For this install, you need to have Magic and all of the dependencies installed. This is also a very large install, so make sure you have plenty of disk space available. As this is a large download, it will take quite some time, so be patient.

- **git clone** https://github.com/RTimothyEdwards/open_pdk
- **cd open_pdk**
- **./configure --enable-sky130-pdk --enable-sram-sky130**
- **make**
- **sudo make install**
- **make distclean**

While installing this massive script, you may see some warning and errors that pop up throughout the installation process - this is normal. The only thing you need to worry about is if, after going through any of these steps, you encounter “Error 2”. If you see this, then something went wrong, and it will direct you to a logfile where you can see what went wrong.

Once you have gone through this installation process, you can check if everything is installed correctly by using the commands shown below to navigate to the pdk installation to view the files:

- `cd /usr/local/share/pdk`
- `ls`

If you do this, you should see something similar to the figure below.

```
jthater@Ubuntu-Example:~$ cd /usr/local/share/pdk/  
jthater@Ubuntu-Example:/usr/local/share/pdk$ ls  
scripts sky130A sky130B  
jthater@Ubuntu-Example:/usr/local/share/pdk$
```

Figure 17: Open PDKs File Directory

If you see this, then it is likely that everything was installed correctly. You may notice that there are two Sky130 directories. One is “A” and the other is “B”. You don’t need to know all about the minute differences between these two. The only difference that matters is that the “sky130A” is the standard process, and the “sky130B” is a modified process that supports the manufacturing of ReRAM. **So, if you plan to use ReRAM in any part of your design, you must use the sky130B process for all of your designs. Otherwise, if you have no plans to use ReRAM, then you can go ahead and use the standard sky130A process.**

It is a good idea to move into these files and look around to see all that was downloaded. Each directory will contain a libs.ref and a libs.tech directory. The libs.ref directory houses all of the libraries that were downloaded and contains the files for the cells contained in these libraries (read the SkyWater 130 nm PDK documentation for more information about these libraries). The libs.tech directory will house important files for analog design and configuration of the tools for the SkyWater process. This will be covered in the next section of this guide.

Tool Usage + Example “Hello World” Type Project

In this part of the guide, I will cover how to use the tools in the analog process flow. Along the way, I will try to link out to videos/documentation that are helpful. I will also include a basic “hello world” type project - which will just be a basic inverter. I will take this inverter through the analog process flow of schematic creation, testbench creation, simulation, netlist extraction for layout, layout, DRC/LVS, post-layout netlist extraction, and finally, post-layout simulation. I will be using the sky130B process variant to design this inverter.

First things first, before proceeding into design, there are a few things to set up still. The first is to create your project directory. This project directory will contain all files relevant to your project. A recommended project directory setup would look something like this:

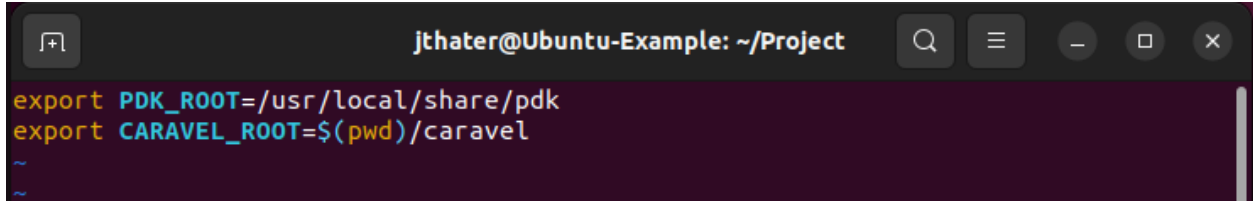
```
<myProject>  
  caravel/ gds/ mag/ netgen/ xshcem/ docs/ openlane/ verilog/
```

These different directories will house the different files of your project and will be required for when you submit your project design through Efabless.

Another thing to set up is global variables. You will need to have these set to successfully use the tools with the SkyWater 130 nm PDK and go through the precheck process that Efabless has. There are many ways to create the global variables, but I will list out the way that I do it (although there is probably a better way to do it). Simply follow these steps:

- **cd <myProject>**
- **touch setup.sh**
- **vim setup.sh**
 - This should open up a file called setup.sh. Press “i”, which will put you in insert mode and type the following lines in the file:
- **export PDK_ROOT=/usr/local/share/pdk**
 - This should be the default installation of the SkyWater PDK you downloaded. If you move where these directories were installed, then you need to point this variable where they have been moved to
- **export CARAVEL_ROOT=\$(pwd)/caravel**
 - Add text here

If you have done this successfully, it should look like the file shown in the figure below.



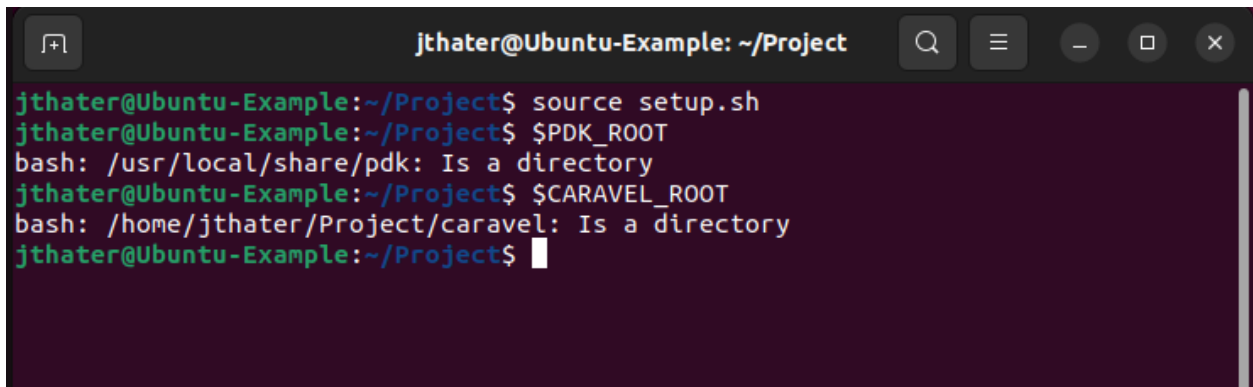
```
jthater@Ubuntu-Example: ~/Project
export PDK_ROOT=/usr/local/share/pdk
export CARAVEL_ROOT=$(pwd)/caravel
~
~
```

Figure 18: Setup File for Open-Source Analog Design Flow

Once this is done, press “ESC”, on your keyboard, which will place you outside of “insert” mode. Then, in subsequent order, press “:wq”, which will exit the file while saving the changes you made to it. Then, to make these global variables go into effect, in the terminal type:

- **source setup.sh**
 - To verify that these global variables were updated, type into the terminal:
- **\$PDK_ROOT**
- **\$CARAVEL_ROOT**

If you did everything correctly, you should see something like the figure below.



```
jthater@Ubuntu-Example: ~/Project
jthater@Ubuntu-Example:~/Project$ source setup.sh
jthater@Ubuntu-Example:~/Project$ $PDK_ROOT
bash: /usr/local/share/pdk: Is a directory
jthater@Ubuntu-Example:~/Project$ $CARAVEL_ROOT
bash: /home/jthater/Project/caravel: Is a directory
jthater@Ubuntu-Example:~/Project$
```

Figure 19: Verification that Global Variables Were Set

This should verify that the global variables have been updated. **NOTE: with the method outlined here, each time you open up a new terminal window, you will have to run the “source setup.sh” code for the global variables to update (which is why I mentioned this way might be inefficient).**

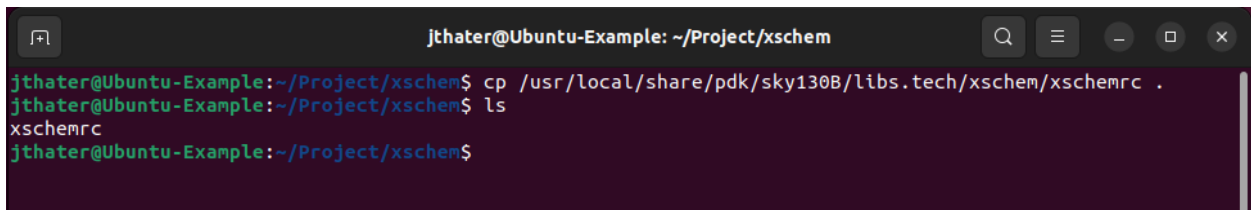
With all of this done, you are finally ready to start the open-source analog design flow.

Schematic Capture/Creation + Simulation

The first step of the analog design flow is schematic/testbench creation. The next step is to simulate your schematic. As these two steps are closely related for the open-source analog design flow, I have decided to put them in the same category. For this guide, I will focus on Xschem for schematic creation. To begin using Xschem to design with the SkyWater 130 nm PDK, you will need to copy the `xschemrc` into your working directory where you plan to your schematics. The `xschemrc` file should be in the `xschem` directory in the process variant you want to work with (`sky130A` or `sky130B`). To copy this file, first, be in the directory where you plan to work on these schematics, then type:

- `cp /usr/local/share/pdk/sky130B/libs.tech/xschem/xschemrc .`
- `ls`
 - This verifies that the `xschemrc` file has been properly copied into your current directory.

If you have followed these steps, you should see something similar to the figure below.

A terminal window titled "jthater@Ubuntu-Example: ~/Project/xschem" showing the following commands and output:

```
jthater@Ubuntu-Example:~/Project/xschem$ cp /usr/local/share/pdk/sky130B/libs.tech/xschem/xschemrc .
jthater@Ubuntu-Example:~/Project/xschem$ ls
xschemrc
jthater@Ubuntu-Example:~/Project/xschem$
```

Figure 20: Xschemrc File in Folder

Once you have the `xschemrc` file in your working directory, simply type:

- `xschem`

This will open up Xschem into a top-level file for the SkyWater PDK, as seen in the figure below.

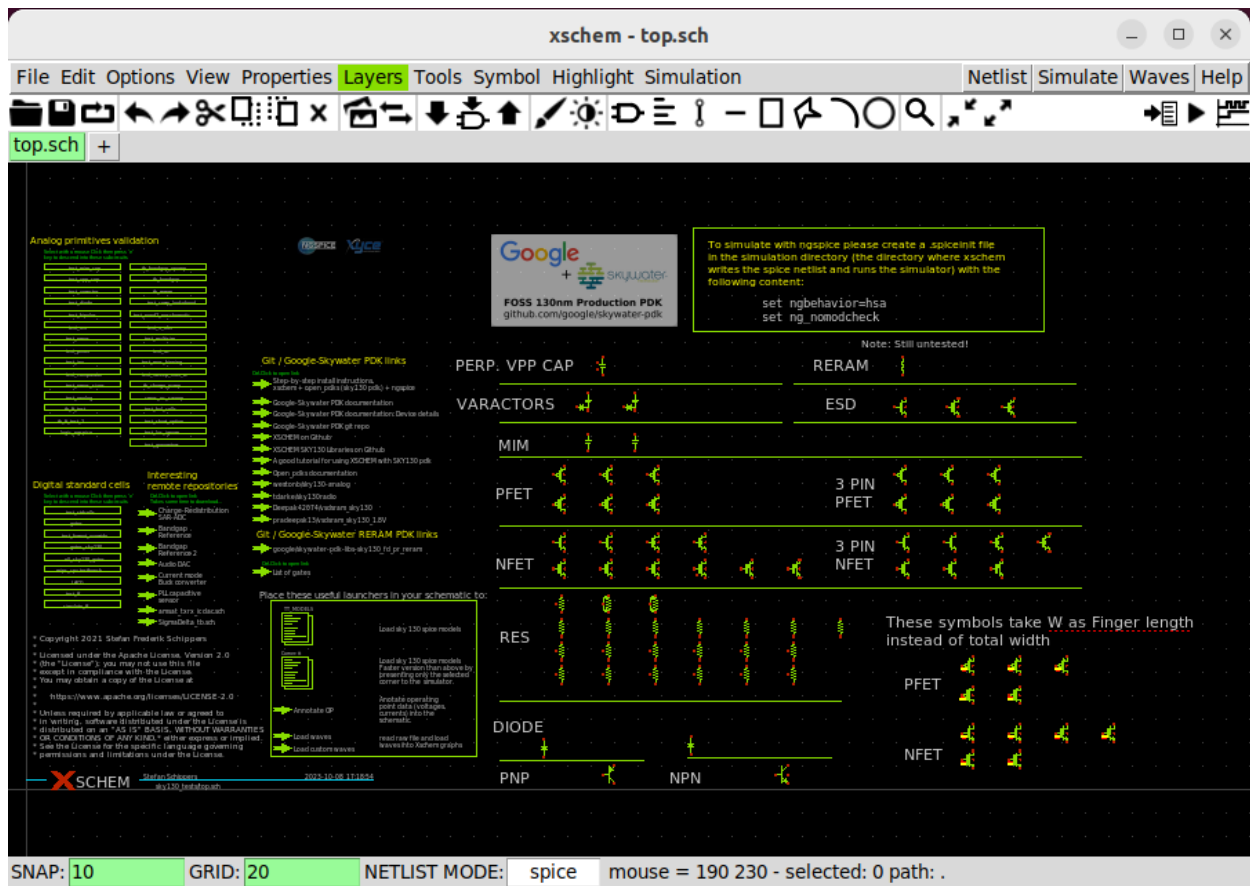


Figure 21: Xschem Top Level

This top-level file houses different primitive cells seen on the right-hand side. On the left-hand side, it has different example testbenches and schematics. Navigating through these and looking at some of the tutorials is probably a good place to get started with getting used to Xschem.

Another good place to look is the Xschem tutorial page found here:
https://xschem.sourceforge.io/stefan/xschem_man/xschem_man.html

For simulation advice with Ngspice, you can look at the user documentation found here:
<https://ngspice.sourceforge.io/docs/ngspice-manual.pdf>

This is a very lengthy document that covers everything, but it can also be very hard to find information, so I would just recommend looking up how to do something using Ngspice on the internet before resorting to the manual.

As for some videos that are helpful, there are quite a few good tutorials that you can find on YouTube for analog simulation. Some good videos are:
https://www.youtube.com/watch?v=YUA_I55k-tM&t=383s (Basic tutorial for using Xschem and simulation)

<https://www.youtube.com/watch?v=bm3l21ExLOY&t=834s> (More in-depth tutorial about simulations - a little outdated)

https://www.youtube.com/watch?v=BpPP2hE_eK8&t=1630s (More in-depth tutorials dealing with hierarchical design - a little outdated) - Example is heavily based on this video

<https://www.youtube.com/watch?v=bYbkz8FXnsQ&t=2s> (More tutorials - from the creator of Xschem)

Through these tutorials, you should have a good feel of Xschem and how to create schematics, test benches, and how to simulate designs.

[Example Inverter Schematic Creation and Simulation](#)

For the example of how to create schematics, I will be creating a simple inverter. I will create this schematic with a hierarchical design (the inverter will be its own cell, and the testbench a separate cell) so that it will be easier to take it into the layout. As mentioned before, I will be using the sky130B variation of the process.

Start by opening xschem from inside your project directory. Make sure that the *xschemrc* file is in this directory. If you have everything set up correctly, the top-level file should show up, as shown in the figure above. From here, navigate to the top menu and select file. This should open a drop-down menu. Find “Create new window/tab” and select that. This selection is seen in the figure below

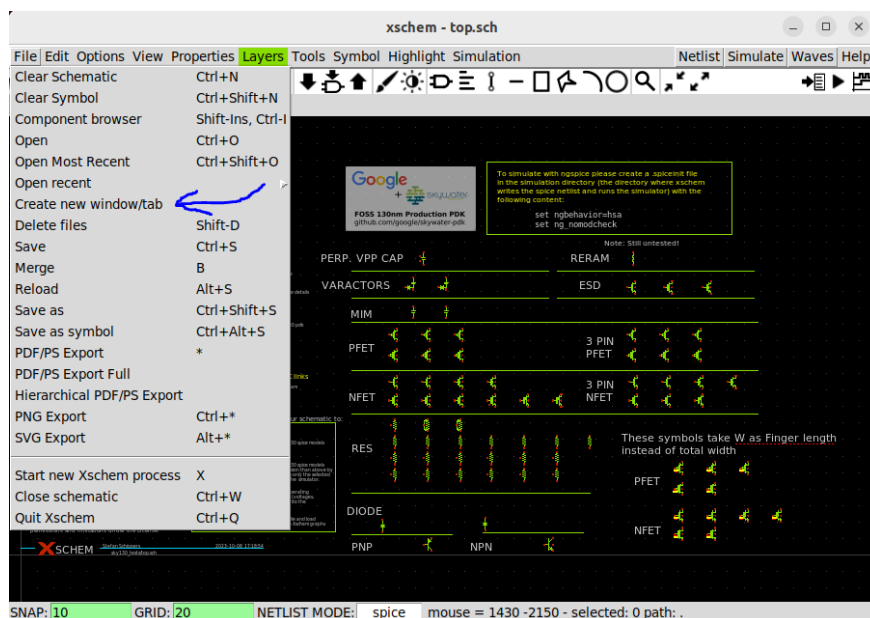


Figure 22: Xschem Tutorial - Creating New Window

This will open up a blank schematic where the inverter will be created. To begin placing components, press either “*Shift + i*” or right-click and select “*insert symbol*”. If you do this, you should see three separate directories to choose from, as seen in the figure below.

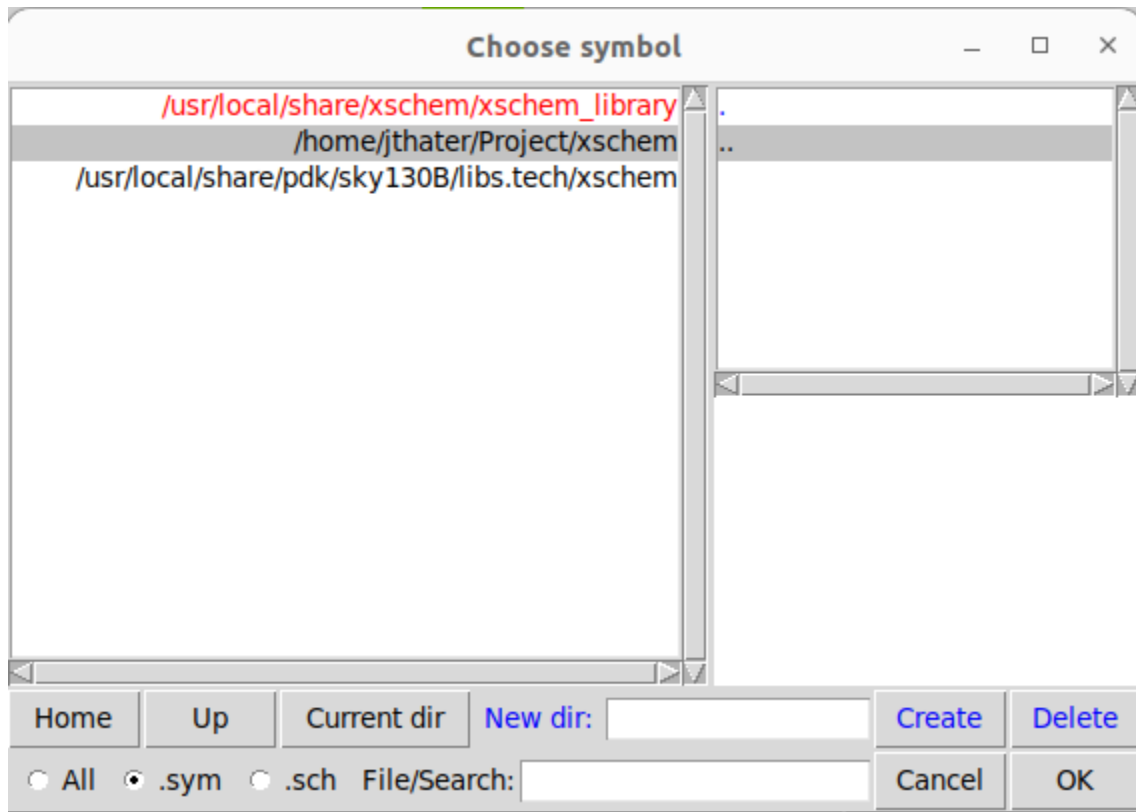


Figure 23: Xschem Symbol Insertion

I will briefly explain what each one contains, but I recommend that you look through each of them as well.

- /usr/local/share/xschem/xschem_library
 - This houses all of the inherent xschem devices that would be a part of any circuit simulation. Think voltage sources, input/output pins, etc.
- /home/jthater(user)/Project/xschem
 - This is your current directory and where all of your schematics/symbols will be placed once you create them
- /usr/local/share/pdk/sky130B/libs.tech/xschem
 - This is the PDK library, and it houses all the cells that can be used in the SkyWater process. It also houses test benches and digital cells as well.

First, place down the two transistors that will be used for the pull-up and pull-down network. These will be in the PDK library under a directory called “*sky130_fd_pr*”. For my inverter, I want to have a VDD higher than 1.8 V, so I will need to pick a transistor that can support a higher voltage. As such, for my pull-down network, I will pick the device “*nfet_g5v0d10v5.sym*”. This selection can be seen in the figure below.

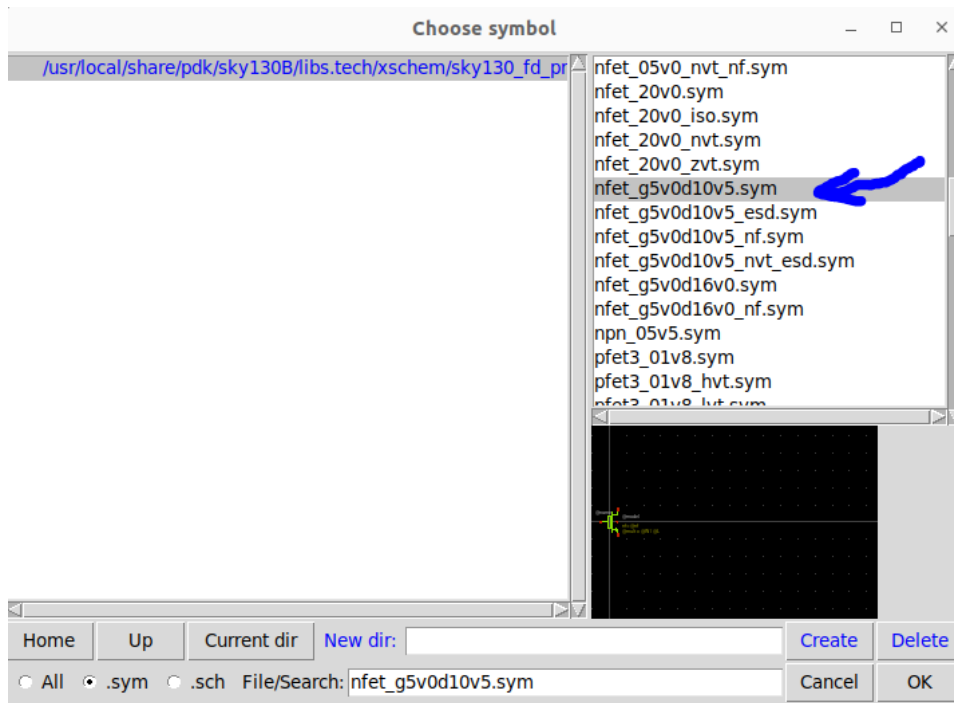


Figure 24: Xschem Nfet Device

This will allow you to place this device in the schematic window. I will grab a pfet of the same name for my pull-up network. If you have done this correctly, it should look something like the figure below.

If you need to move the device, or any device, after it has been placed, simply highlight over it and press the “m” key on your keyboard. To move where you are looking in Xschem, press the middle mouse button and move your mouse around. Pressing the “f” key will put your design in a zoomed-in view.

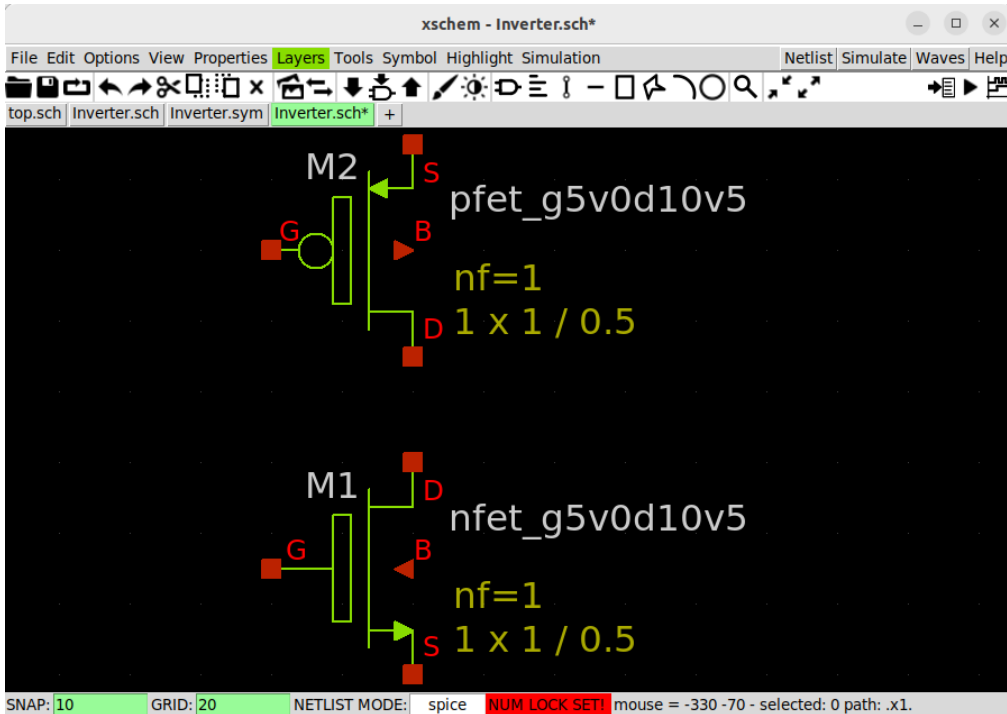


Figure 25: Nfet and Pfet Devices Placed

Once both devices have been placed, they can be wired up. This is done the same as it is in Cadence, so simply press “w” on your keyboard, and a wire will appear. Simply left-click where you want the wire to end. If you made a mistake, highlight the wire and press “DEL” on your keyboard. Hook up the device as shown in the figure below.

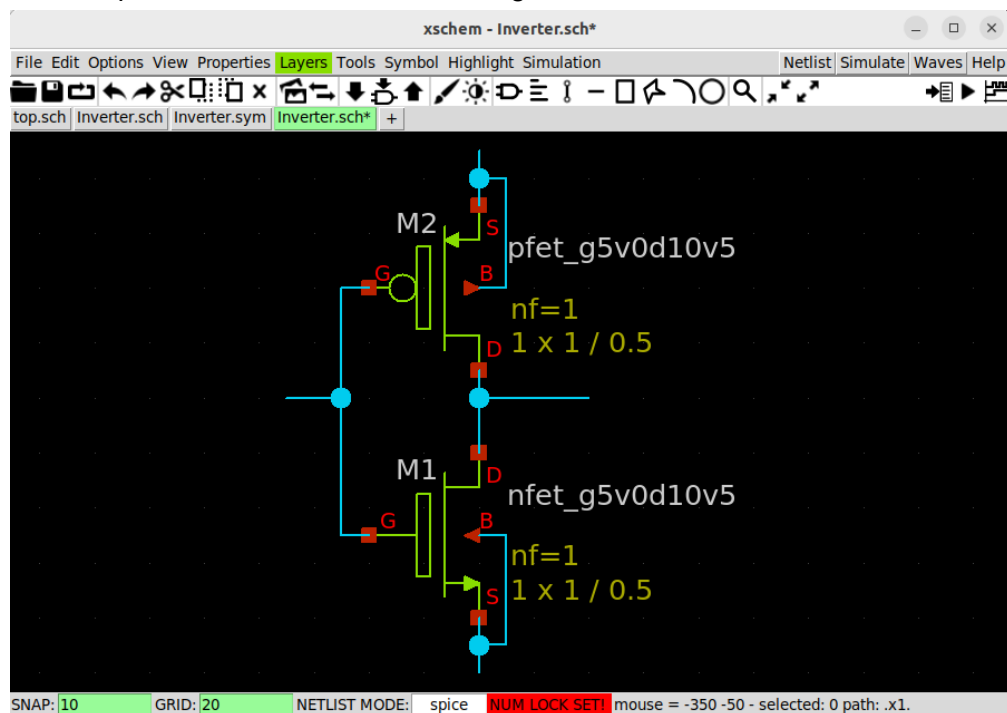


Figure 26: Nfet and Pfet Devices Hooked-up

From here, all that needs to be done is to add the correct pins to the schematic. To do this, open up the “choose symbol” window again and, this time, navigate to /usr/local/share/xschem/xschem_library. From here, choose the “opin.sym” for the Vout, the “ipin.sym” for Vin, and the “iopin.sym” for VDD and VSS. Some useful shortcuts that may come in handy when placing these pins are to press “Shift + r” to rotate an object. Another useful shortcut is to select an object and press the “c” key to copy it.

Once these pins have been placed, they will all have the name “XXX”. To change this, simply double-click on the pin, and a window will pop up, which will allow you to edit the name of the pin. Once you have placed all the pins and renamed them, it should look similar to the figure seen below.

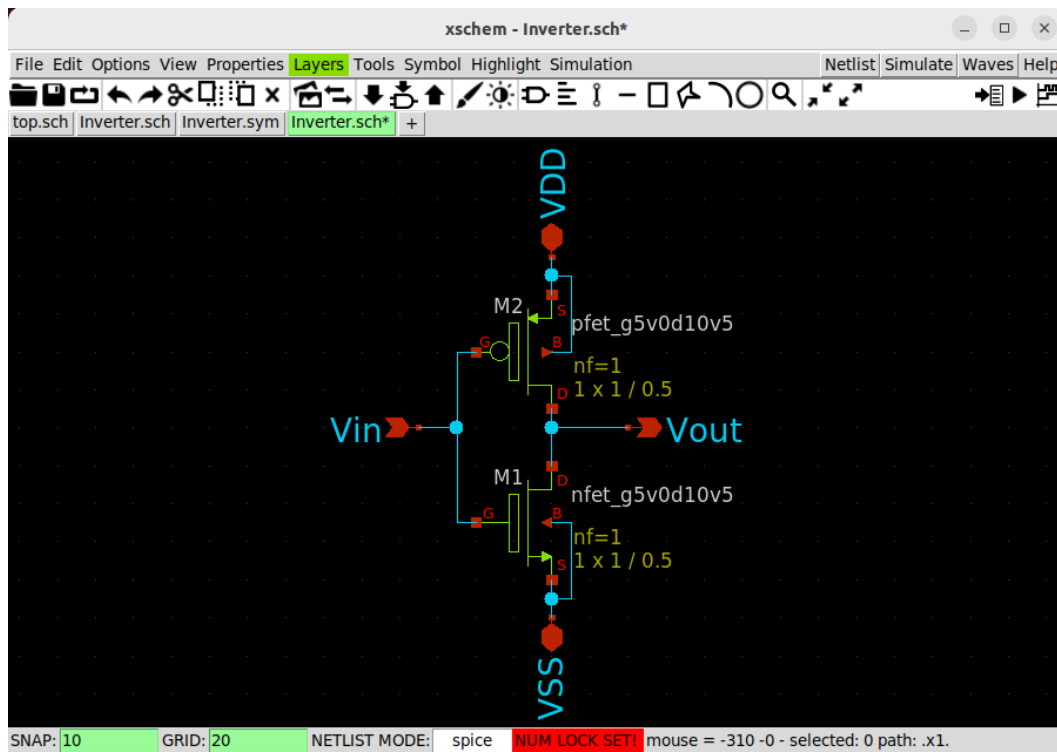


Figure 27: Pins Added to Device

Once you have a schematic that looks similar to mine, save the design in your project directory under an appropriate name. I named mine “inverter.sch”. Once you have saved your design, it is time to make a symbol of your schematic. Before, you would have to create the symbol by hand, but now you can make a symbol directly from the schematic. To do this, navigate to the top menu and select “Symbol”. A drop-down menu will appear, and you will want to select “Make symbol from schematic”. This selection is shown in the figure below.

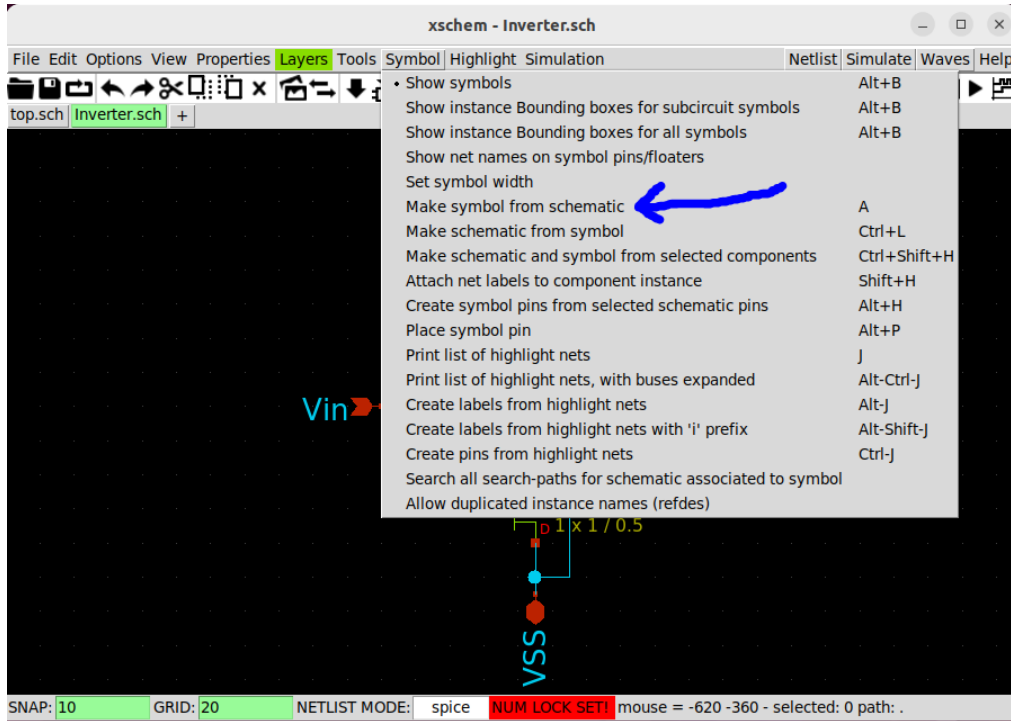


Figure 28: Making Symbol From Schematic

Once you have selected this option, it will appear as if nothing happened. However, the symbol file will have been created. To view this symbol, open up a new window/tab, as you have done before. Once you are in this window/tab, navigate the top menu to file again. This time, in the drop-down menu, select the "open" option. Then, open your project directory. You should see a new file called "inverter.sym". It should look like the figure below.

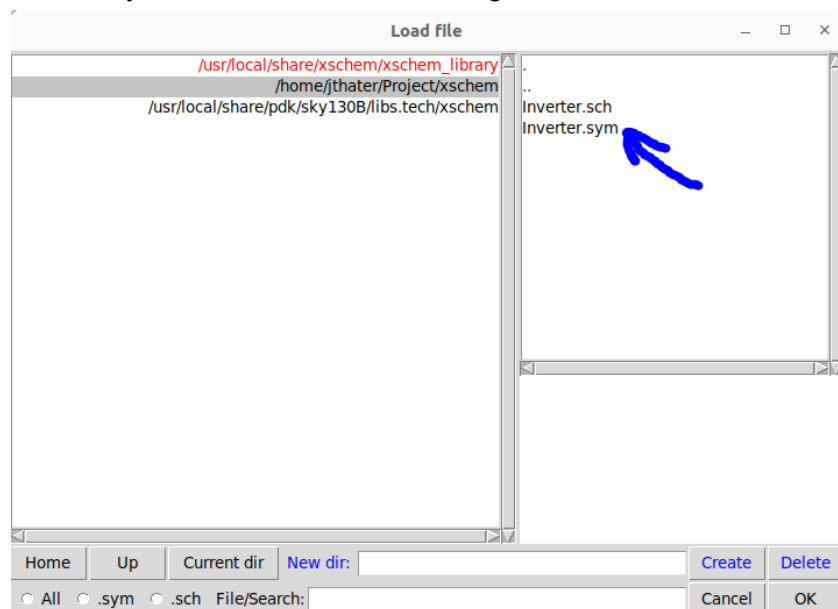


Figure 29: Inverter Symbol Load

Once you open up this symbol file, you should be greeted with the symbol view of your schematic, which should look like the figure below.

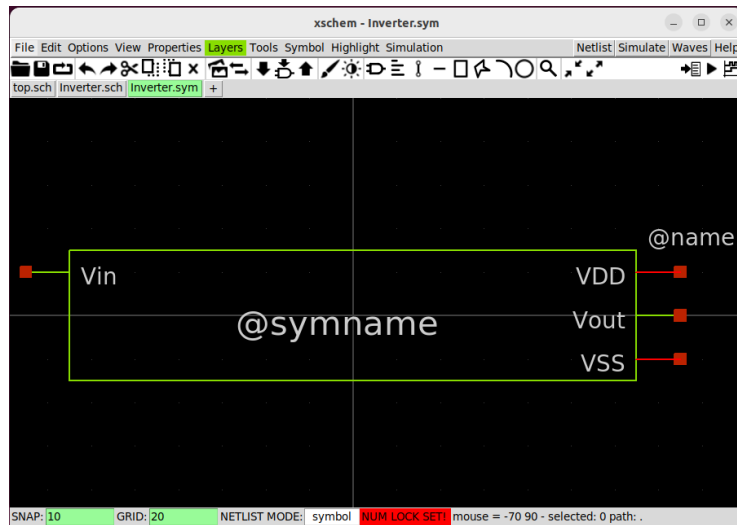


Figure 30: Inverter Symbol

You can change the shape of your symbol, as well as where the pins are located, but as that is just decorative, I will not cover that here.

Once you have verified that the symbol has been created successfully, then the test bench can be created. To begin creating this test bench, open up a new window/tab. From the new schematic window, open up the “choose symbol” and select the newly created “inverter.sym” in your project directory. It should look like the figure below when inserted.

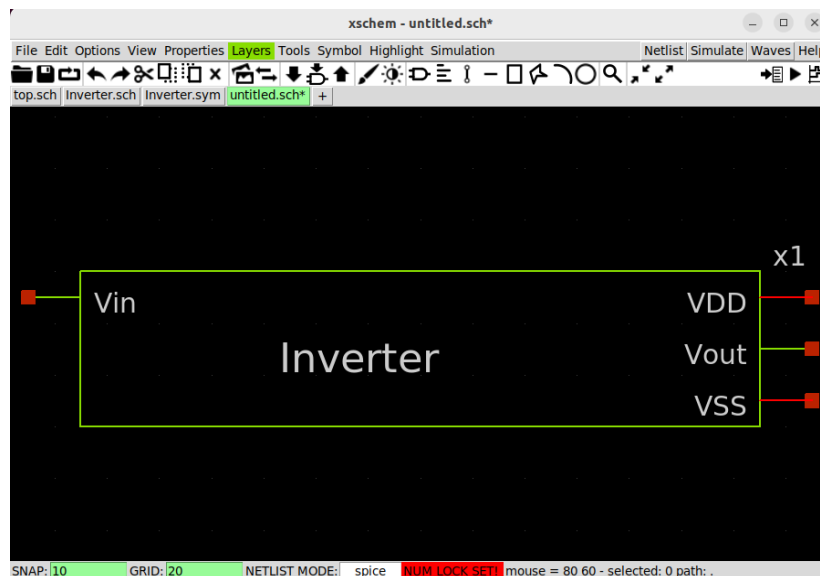


Figure 31: Inverter Symbol Placed

Notice that the symbol now has the name of your schematic as well as a name to refer to it (x1). Once you have placed your inverter symbol, it is time to place components around it to simulate

it. This is essentially the same idea as how you would create a test bench for a schematic in Cadence, although there are a few differences.

To begin, navigate to the Xschem library and add a “vsource.sym”. Once you have placed it down, copy another one down as well. One should be next to the Vin pin, and one should be next to the VDD pin. In the Xschem library, also find and place “gnd.sym” near the VSS pin and the two vsource devices. Next, in the Xschem library, add a device called “code_shown.sym” (this is what will be used to run different simulations like dc, ac, transient, etc.). Finally, from the sky130 library, in the “sky130_fd_pr” directory, add a device called “corner.sym” (this is the corner test that will be run on your device; by default, it is set to typical typical). With all of these devices placed, go ahead and hook up everything up, as shown in the figure below.

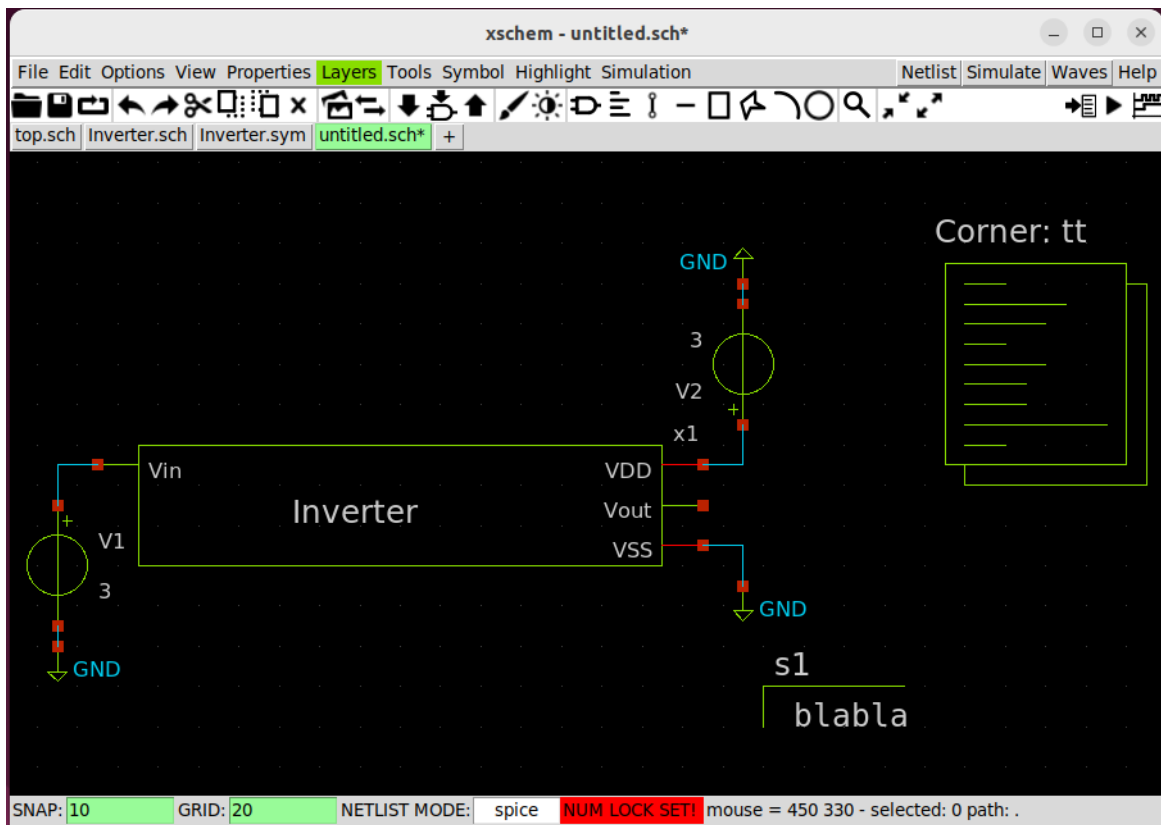


Figure 32: Inverter Testbench Wired

With everything hooked up, there is still a bit more to go. We have to describe what each voltage source is doing and, in the code_shown device, write out the code that will run a simulation of the circuit. We also have to give pin names to each net. For the voltage source connected to VDD, double-click on it and change the value from 3 to 2.5. This will supply a DC voltage of 2.5 V to VDD. For the voltage source connected to Vin, we want to create a repeating square wave. This can be achieved by using a pulse in Ngspice. The pulse in Ngspice follows this rule set:

- **pulse(VL VH TD TR TF PW PER PHASE)**
 - VL = Lower voltage

- VH = Higher voltage
- TD = Delay
- TF = Fall time
- TR = Rise time
- PW = Pulse width
- PER = Period
- Phase = Phase shift

To create a square wave for the Vin pin, I am using the following pulse command:

- **pulse(0 1.8 1ns 1ns 1ns 5ns 10ns)**

This command has to be put after “value=” and must be enclosed with quotation marks. It should look like the figure below.

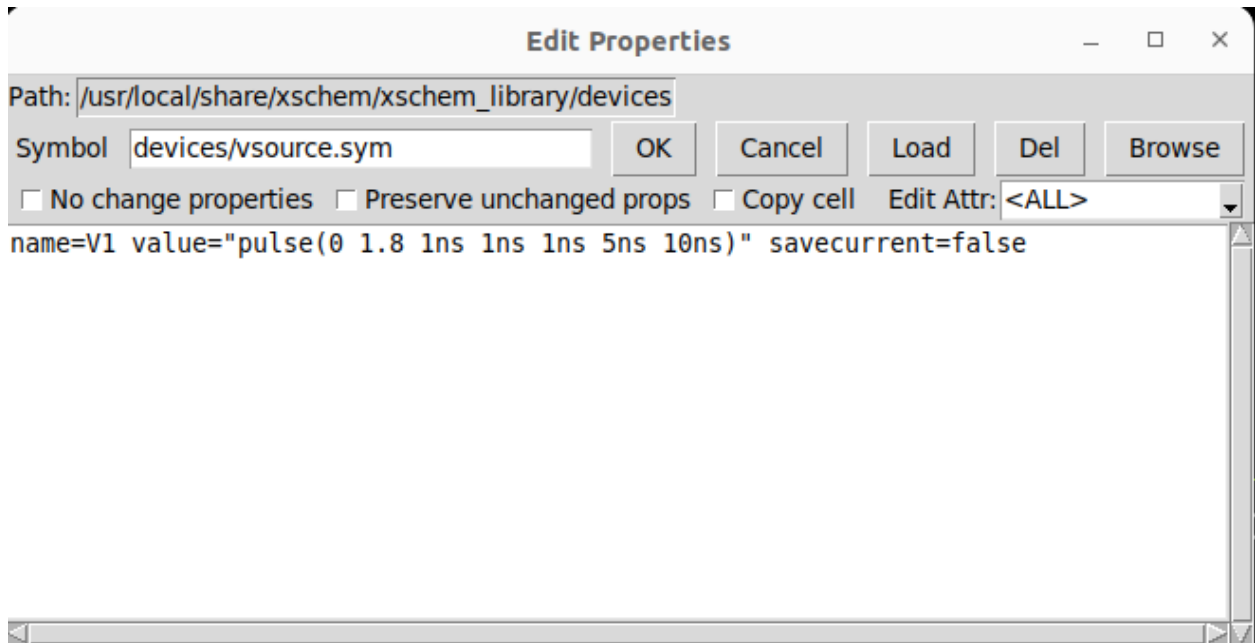


Figure 33: Voltage Pulse

Now that the voltage sources are set up, pins need to be added to our symbol so that the simulation will know the nets it has to run simulations on. To do this, navigate to the “choose symbol” window and go into the Xschem library. From here, find the device called “lab_pin.sym” and place it down on the VDD net. Copy this device and place one on the Vin net and on the Vout net. Double-click these pins and change their names so they match the pin on the symbol. If you have followed these steps so far, you should have a test bench that looks similar to the figure below.

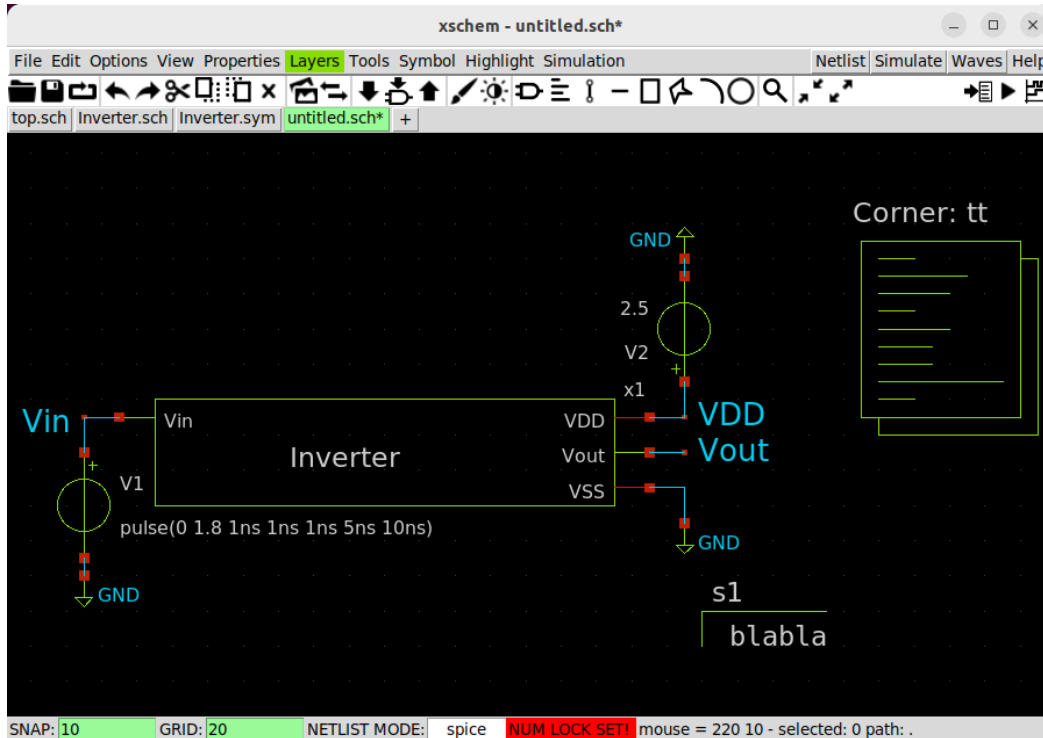


Figure 34: Control Block Placed

The final thing to add before we can begin simulating this test bench is add code to the code_shown device so Ngspice knows what simulation to run. The way the test bench is set up, a transient simulation needs to be run. We also want to save the waveforms that are produced so we can look at them later after the simulation is run. So, to do all this, type this in the code_shown device after the “value=”:

- “.tran 0.01n 1u
.save all”

If you have done it correctly, it should look something like the figure below.

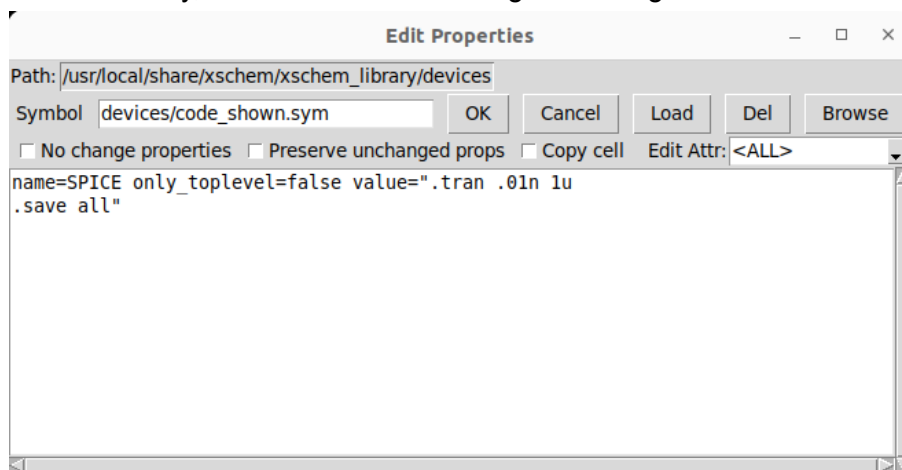


Figure 35: Control Block Code

This will set up a transient simulation from 0 to 1us in steps of 0.01ns. It will also save all the data. From here, save this schematic under an appropriate name in your project directory. I named mine “inverterTB.sch”.

A cool thing that you can do once you have saved this testbench schematic is you can easily descend into your inverter schematic. To do this, select the inverter symbol and press “e” on your keyboard. To ascend back up to your inverter testbench, press “CTRL + e”. This will be more helpful as your designs get more complex.

With everything set up in the testbench, it is time to simulate it. There are two ways to do this - I will show both.

The first way is to run the simulation as a live Ngspice simulation. Xschem is already configured for this setup, so you actually don't have to do much. You want to ensure that the simulation you will be running will be on a Spice netlist. To do this, navigate to options in the top menu. In the drop-down menu, ensure that “Spice netlist” is checked and nothing else. The second thing you should do is, under the Simulation drop-down menu, select the option “Show netlist after netlist command”. This will allow you to view the netlist that is created from your testbench.

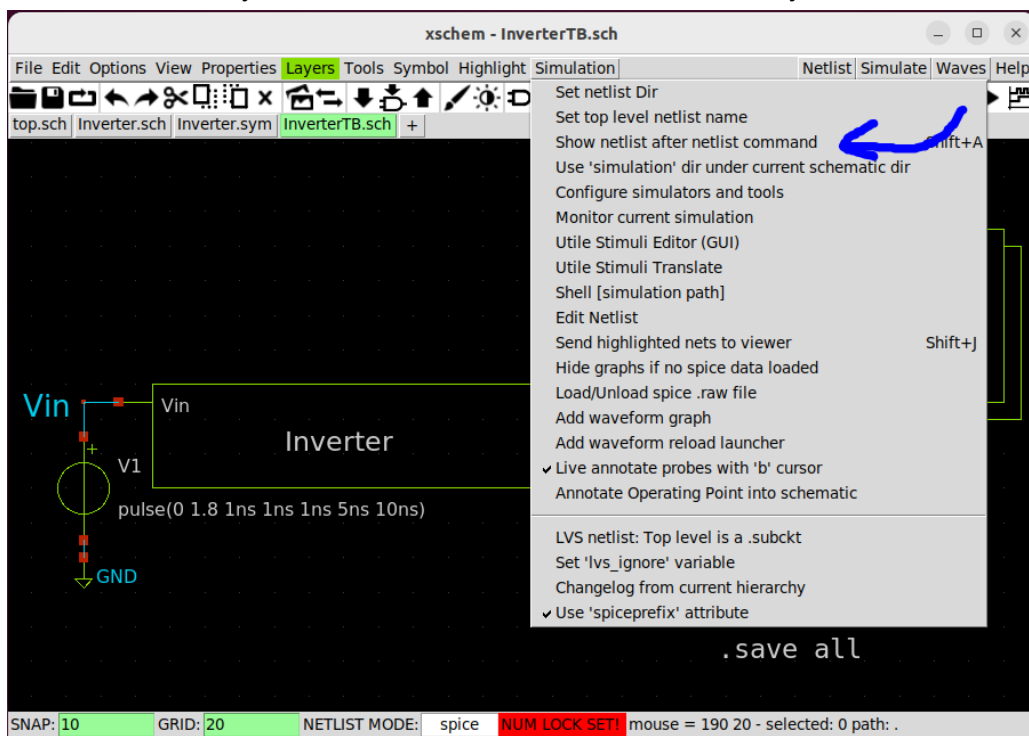


Figure 36: Netlist Command

From here, all you have to do is press the “Netlist” button in the top right corner. If you selected the option to show the netlist window, then the netlist window will pop up. I wouldn't worry too much about what it looks like now, but going forward, it is a very good idea to learn how to read these, as it will make your life a lot easier when it comes to troubleshooting layout issues. Make sure to save the netlist file. After you have saved the file, hit the “Simulate” button, also located

in the top right corner. Since you are running a live simulation, a Ngpsice terminal will pop up and run the simulation. After it is done, it should look something like the figure seen below.

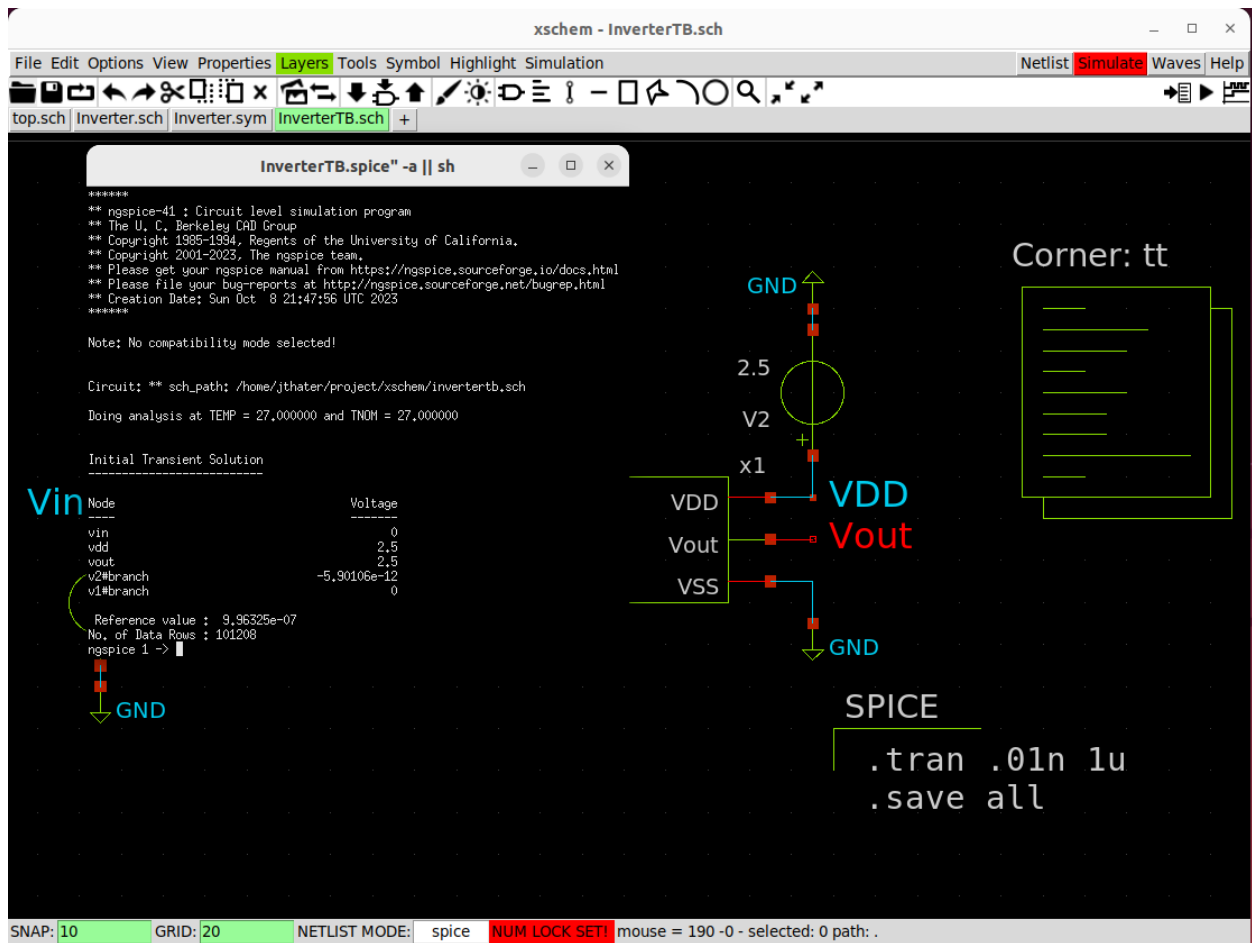


Figure 37: "Active" Simulation

To view the waveforms, you have to type commands into the terminal that popped up. To plot the voltage of Vout and Vin, you can use a command like:

- **plot v(Vout) v(Vin)**

This will plot the voltage of Vout and Vin on the same graph and should look something like the figure below.

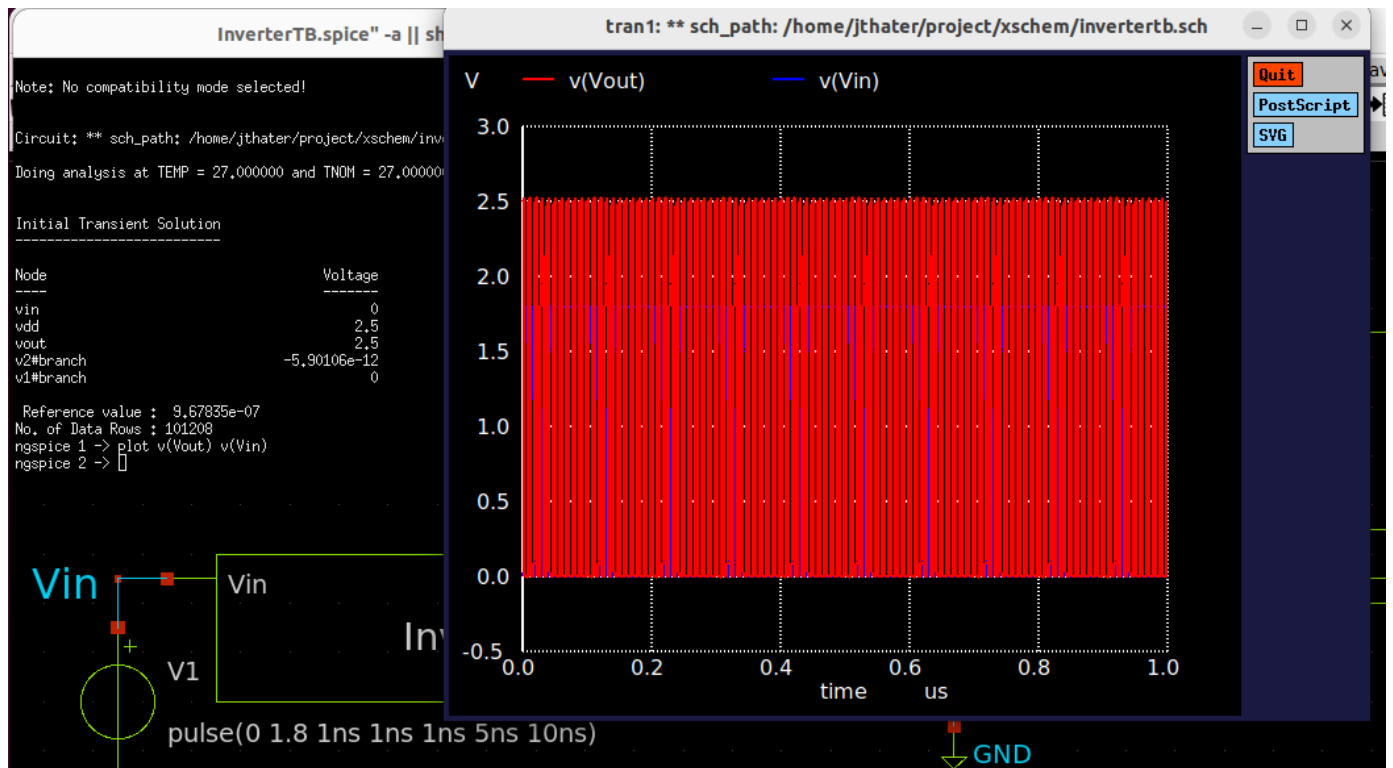


Figure 38: "Active" Simulation Waveforms

As you can see, the simulation ran too long to read anything out of it. I have not found a way to zoom in on the graph - I assume it would be some command in the terminal where you view the waveform from a specific time to a specific time. So, with this method, you have to be smart about how long you run your transient; otherwise, you will end up in a situation like this.

The other method, the one I am biased towards, is to run the simulation as a batch simulation and view the waveform using GAW. To do this, exit out of the Ngspice terminal and waveform viewer, and go back into the Xschem testbench. Navigate the top menu, and under the "Simulation" top-down menu, select "Configure simulators and tools". This will open up a window for simulation configuration. In this window, you will want to check the box next to Ngspice batch instead of Ngspice. You will also want to ensure that Gaw viewer is checked as well. Your setting should look like the figure below. Make sure you press "accept and close" for your settings to take effect.

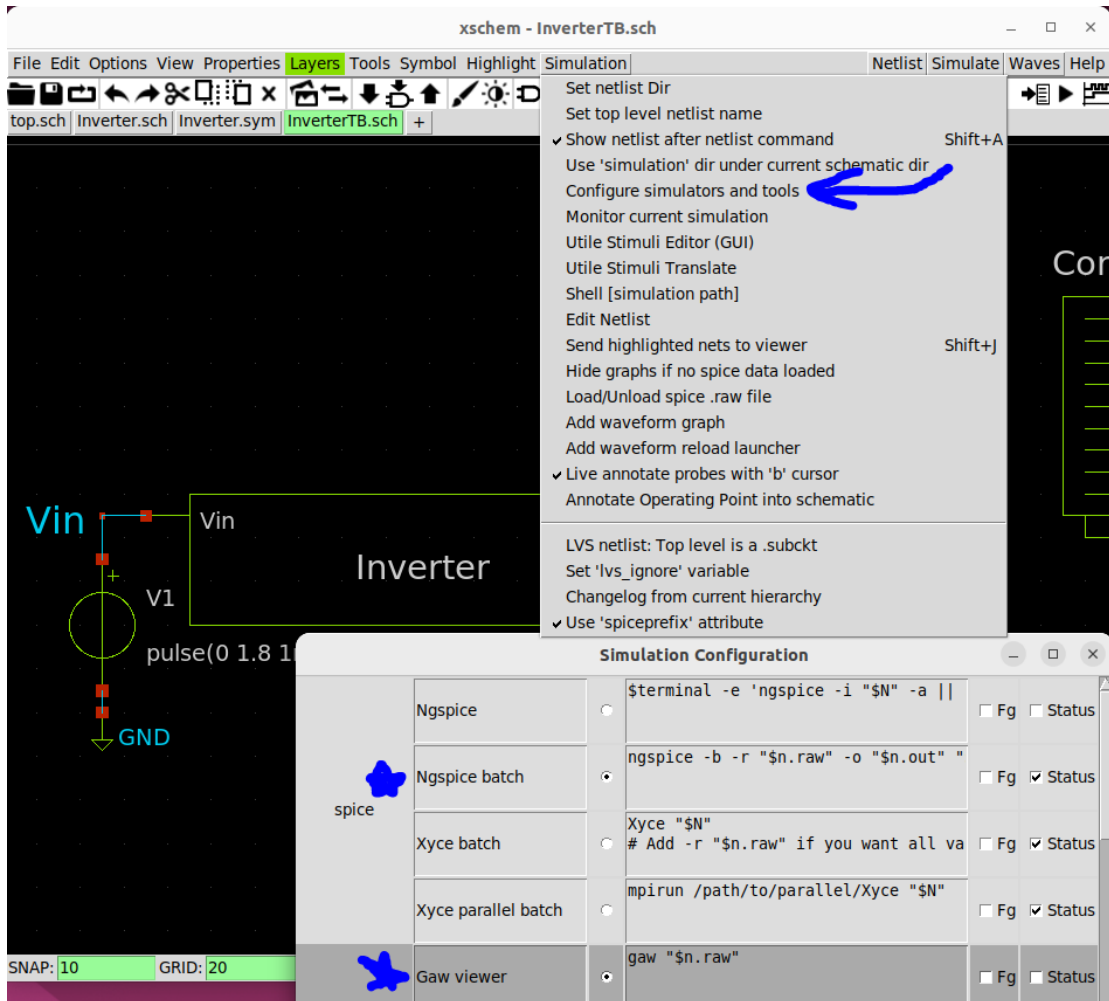


Figure 39: Setup Batch Simulations

With this set up, press the “Netlist” button again and save the resulting netlist that is generated. Then, hit the “Simulate” button. You will notice that the Ngspice terminal does not show up - this is expected. Instead, the “Simulation” button will be highlighted red for a little bit. This is because Ngspice is running the simulation in the background. Once the simulation is done, you will be given a status screen (if you have kept it checked in the “Simulation Configuration” screen). You can simply exit out of the status screen. **NOTE: If any errors occur during the simulation process, the errors will show up in this status screen.**

Now, to view the waveforms hit the third button in the top right labeled “Waves”. If you have configured Gaw correctly, as well as the “Simulation Configuration” in Xschem, Gaw should open with all of the waves that were generated from the simulation. Gaw is reading the “.raw” file of your test bench. This “.raw” file essentially is just a text version of a graph with all of the data points. If you have done everything correctly, it should look like the figure below.

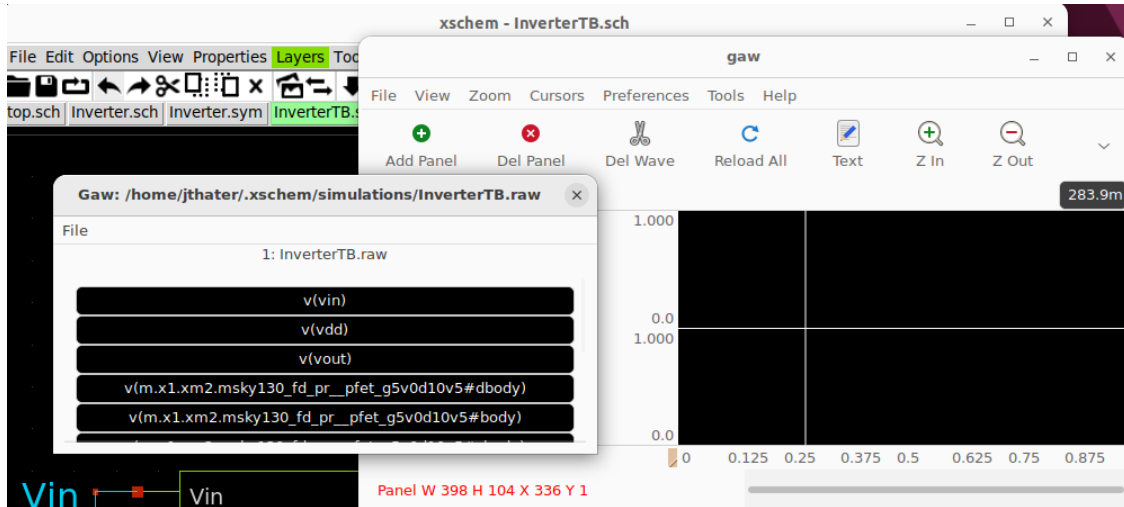


Figure 40: Batch Simulations Raw

To view the waveform, simply drag the boxes from the “.raw” window to the “gaw” window. For this example, I will drag “v(vin)” to the top window and “v(vout)” to the bottom window. I will also utilize the “Z in” option at the top of the window to zoom into the waveform. If you did this, you should get something similar to the figure below.

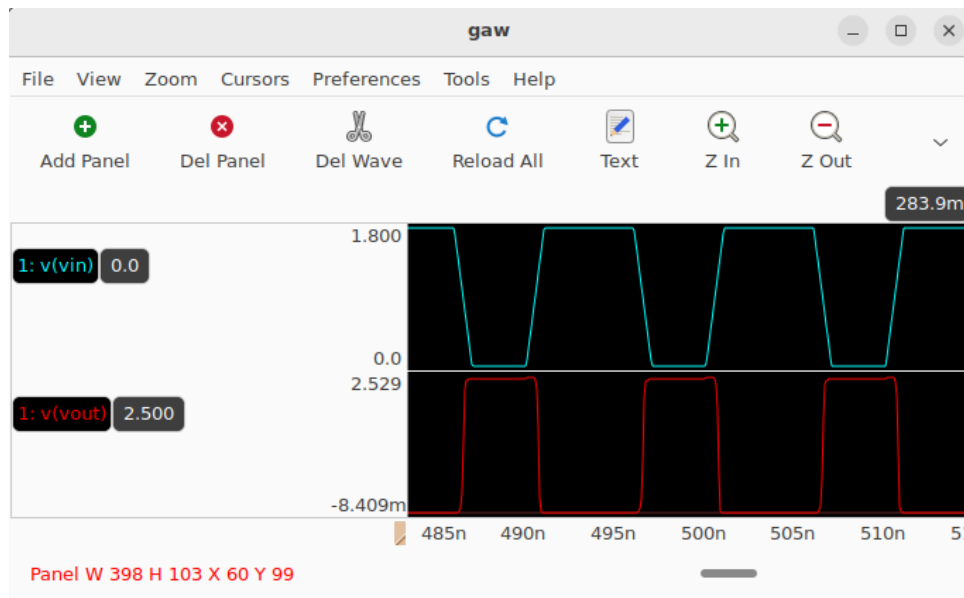


Figure 41: Batch Simulations Waveforms

I would recommend playing around with Gaw to see all that it can do because it is very useful. The reason I like using Gaw over the Ngpsice waveform viewer is because you can see all of the things that you can plot, it is more configurable, and it is easier to view the data. It is up to you which one you prefer more, so I recommend experimenting with both.

You can run more simulations on this inverter if you would like, but I will leave that up to you. With all of these steps complete, you have completed the schematic creation and simulation steps of the analog process flow. Next up will be netlist extraction and making the layout.

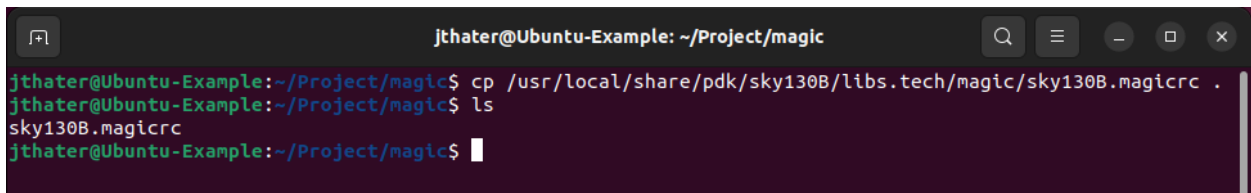
Layout Creation + DRC/LVS

The next part of the analog process flow is the most complicated and will cause some headaches at first. In this part, you will be importing the Spice netlist of your schematic into Magic. You will then make your layout, extract the layout netlist, and compare it to see if it matches the schematic netlist. Unlike Xschem, Magic is not like a lot of commercial tools, so it will take a while to get used to. I will try my best to explain the process as thoroughly as possible and mention where there are differences between what you may be used to with Cadence and what is new in Magic, but I can not cover all of the complexities that exist. Out of all of the areas in the project, this is where we ran into the most trouble, so I can not recommend enough that if you get stuck on this part, do not be afraid to ask questions in the Slack channel.

To get started with making your layout, you have to move a configuration file from the PDK library into your current directory (I am assuming you are now in your “magic” directory). This file is much like the `xschemrc` file. Except in this case, it houses important links and files for the rules for the layout of the SkyWater 130 nm PDK. To move this configuration to your current directory, type the following commands into the terminal (remember I am using the sky130B variant, so if you are using the “A” variant, switch the letters as needed):

- `cp /usr/local/share/pdk/sky130B/libs.tech/magic/sky130B.magicrc .`
- `ls`

If you have followed these steps, you should see something similar to the figure below.

A terminal window titled 'jthater@Ubuntu-Example: ~/Project/magic'. The terminal shows the following commands and output:

```
jthater@Ubuntu-Example:~/Project/magic$ cp /usr/local/share/pdk/sky130B/libs.tech/magic/sky130B.magicrc .
jthater@Ubuntu-Example:~/Project/magic$ ls
sky130B.magicrc
jthater@Ubuntu-Example:~/Project/magic$
```

Figure 42: Magic Setup File in Folder

With this file, you can run Magic with the correct technology file for your project. To test if it is working, type the following command into the terminal:

- `magic -rcfile sky130B.magicrc`

With this, it should open magic with the Sky130B technology file loaded. You should be able to verify this by looking at what technology Magic is using. The figure below shows you where to look and the correct Sky130B technology being used.

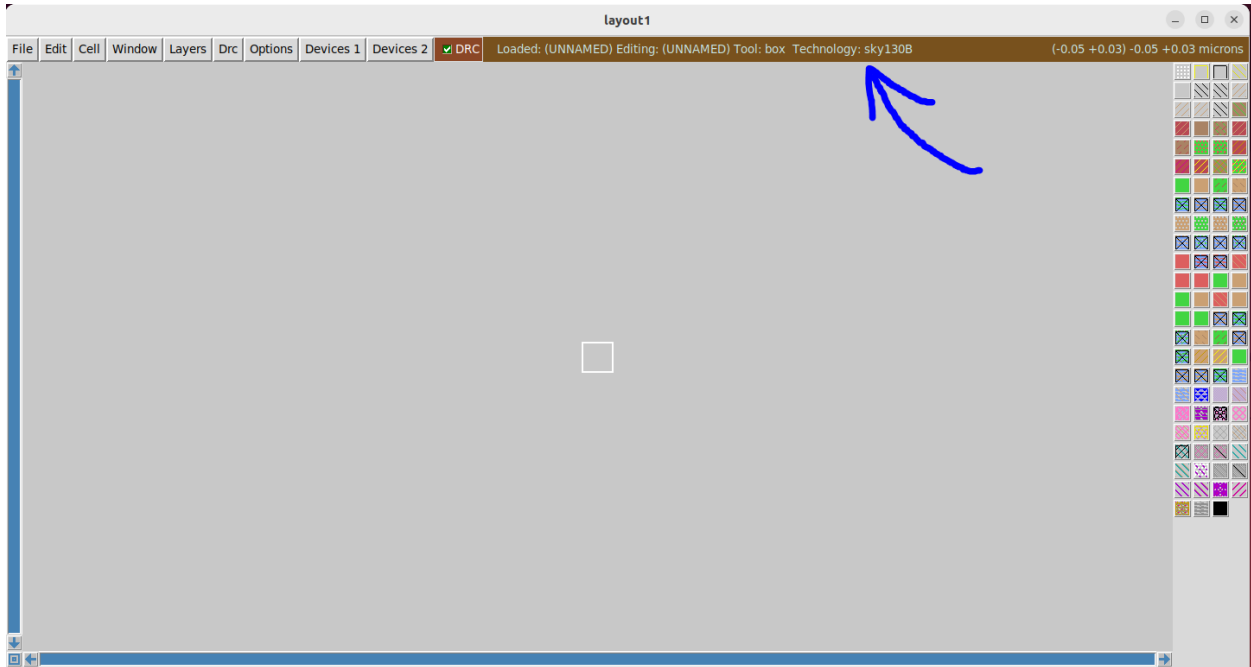


Figure 43: Magic Correct Technology File

Each time you run Magic, you will want to run it with that specific command; otherwise, you will not load in the correct library, and your layout will not follow the rules set out in the SkyWater 130 nm PDK.

As for tutorials on how to use Magic, the Magic documentation is a decent place to start, although it may be more helpful once you are more familiar with Magic. The documentation can be found here:

<http://opencircuitdesign.com/magic/>

Once on the landing page, navigate to “Using Magic” and then “Magic tutorials”.

A helpful cheat sheet (short-cuts and commands) for Magic, which was created by Harald Pretl in the Slack channel, can be found here:

https://github.com/iic-jku/osic-multitool/blob/main/magic-cheatsheet/magic_cheatsheet.pdf

As for videos, there is not a lot that covers specifically how we will need to do layouts or basic layouts for tutorials. A few videos:

<https://www.youtube.com/watch?v=RPppaGdjbj0> (Guide on how to create inverter - does not import Spice model, so not recommended, unless you know what you are doing)

These three videos are from Efabless and are a great place to get started. However, they are a little outdated.

<https://www.youtube.com/watch?v=ORw5OaY33A4&t=9s> (part 1)

<https://www.youtube.com/watch?v=NUahmUtY814> (part 2)

https://www.youtube.com/watch?v=OKWMM1D0_fPI (part 3)

<https://www.youtube.com/watch?v=XvBpgKwzrFY> (Video on how to create op-amp layout) - **This video is incredibly helpful for getting tutorials for this entire process, so I heavily recommend checking it out.**

As for running LVS, there is even fewer helpful videos. However, this tutorial is very helpful: <https://www.youtube.com/watch?v=NCaNF4EunYU>

Hopefully, with these tutorials and the example I show, you will get a decent grasp of using Magic to create layouts and how to run LVS. I am going to repeat myself, but I can not stress that out of all areas in this design flow, this is most likely where you will get stuck, so don't be afraid to ask questions in the Slack channel.

Example Inverter Layout + LVS

At this point, we have verified that our schematic meets our expectations through the simulations we did. Now, it's time to create a layout of our schematic. To begin doing this, navigate back to the original inverter schematic. Under the "Options" drop-down menu, ensure that "Spice netlist" is checked. Then navigate to the "Simulation" drop-down menu and ensure that "Show netlist after netlist command" is selected as well as "LVS netlist: Top level is a .subckt". The figure below shows where they are located.

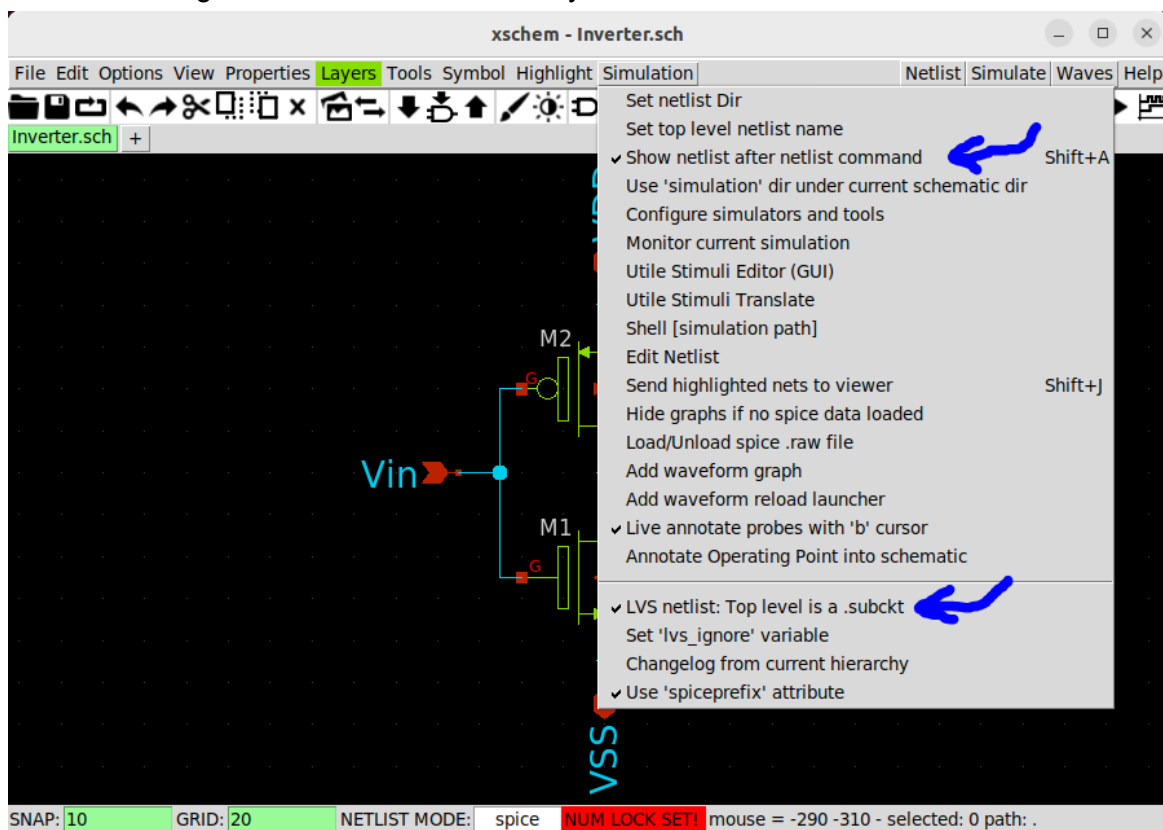
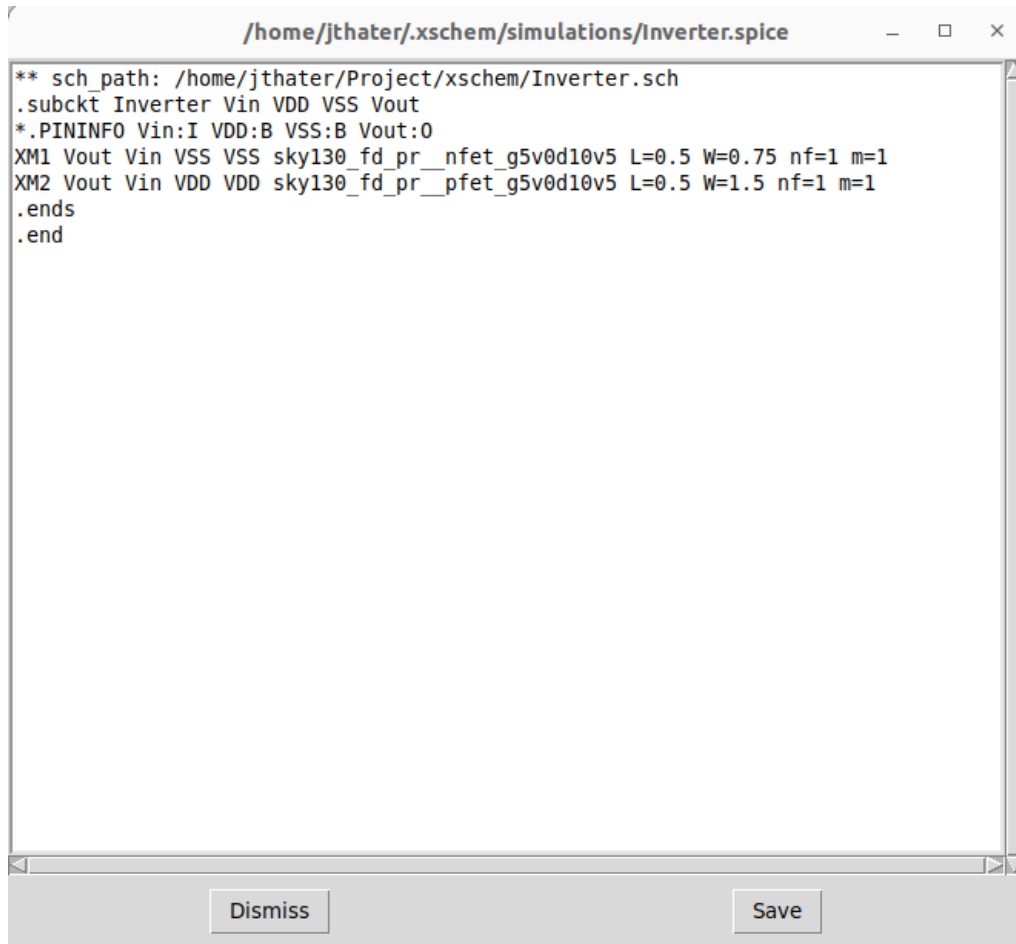


Figure 44: Obtaining Correct Netlist

Once all of these options are checked, you can extract the proper netlist from your schematic. To extract the netlist, click the “Netlist” button on the top right corner of the XSchem software. This will open up a window. The window should look something like the figure below.



```

** sch_path: /home/jthater/Project/xschem/Inverter.sch
.subckt Inverter Vin VDD VSS Vout
*.PININFO Vin:I VDD:B VSS:B Vout:0
XM1 Vout Vin VSS VSS sky130_fd_pr_nfet_g5v0d10v5 L=0.5 W=0.75 nf=1 m=1
XM2 Vout Vin VDD VDD sky130_fd_pr_pfet_g5v0d10v5 L=0.5 W=1.5 nf=1 m=1
.ends
.end

```

Figure 45: Correct Netlist for Layout

The key thing you want to check is that the lines “.subckt Inverter Vin VDD VSS Vout” and “.ends” are not commented out. If both of them are uncommented, then you most likely have extracted the proper netlist. Make sure to save this updated netlist. You may have also noticed that I slightly changed my transistor widths. I changed them to get better characteristics from my simulations.

The next step is opening Magic correctly. In order to open Magic correctly, you have to open it with the correct rcfile from the PDK you are trying to create a layout for. You should have set this up, so all you have to do is navigate to your “magic” directory and type the command:

- **magic -rcfile sky130B.magicrc**

If you see the sky130B technology, then you have opened up Magic correctly and are ready to create your layout. Luckily, there is an easy way to begin your layout. If you have followed all the

steps thus far, then you can import your Spice netlist from your schematic using the “Import SPICE” option. You can navigate to this option from the “File” menu at the top of Magic. The figure below shows where this option is.

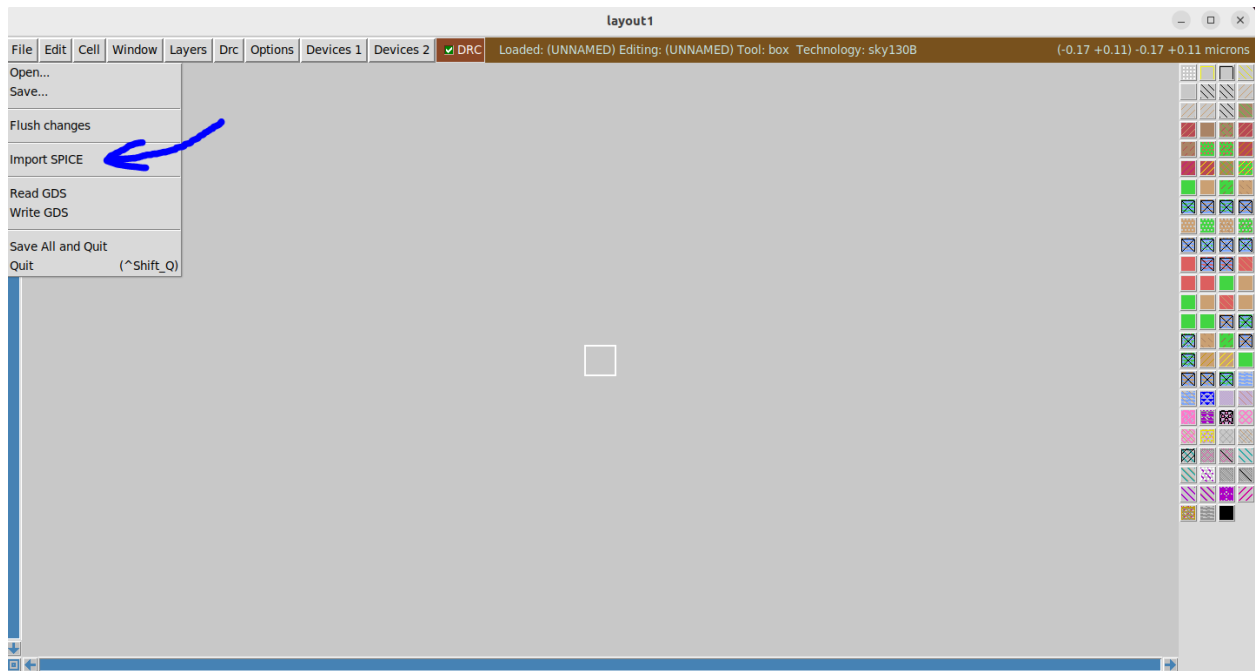


Figure 46: Importing Netlist

Once you click on this, it will open a file navigator where you will have to navigate to your netlist extracted from your schematic. The destination of this file may be different if you have configured settings differently, but by default, it should be in a file path contained in your home directory. For me, this file is located in /home/jthater/.xschem/simulations. The figure below shows this file navigation.

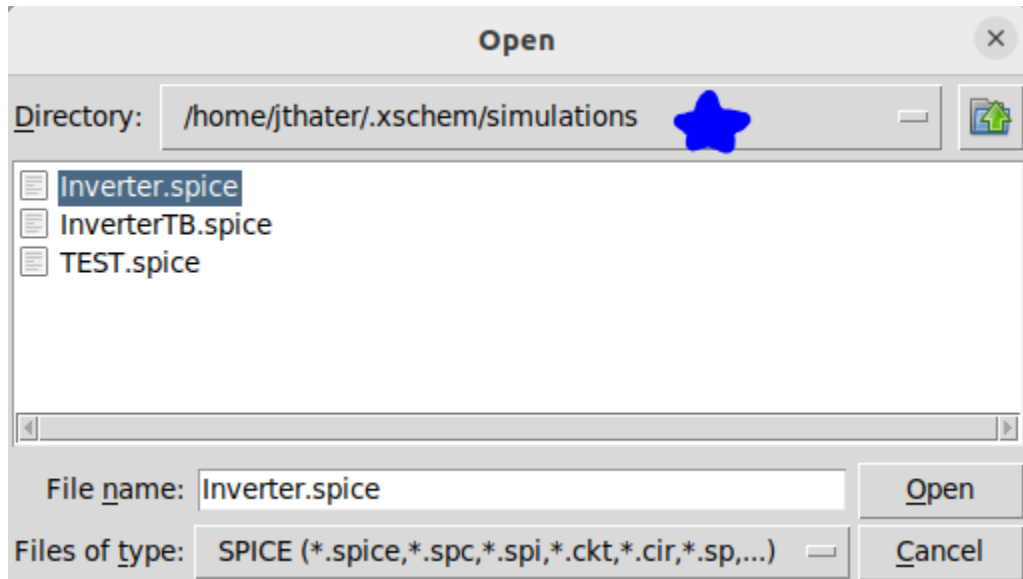


Figure 47: Netlist Location

Make sure to select the correct “.spice” file to import. Once you open up this Spice netlist, you should see something similar to the figure below. Press “v” on your keyboard to see the full view.

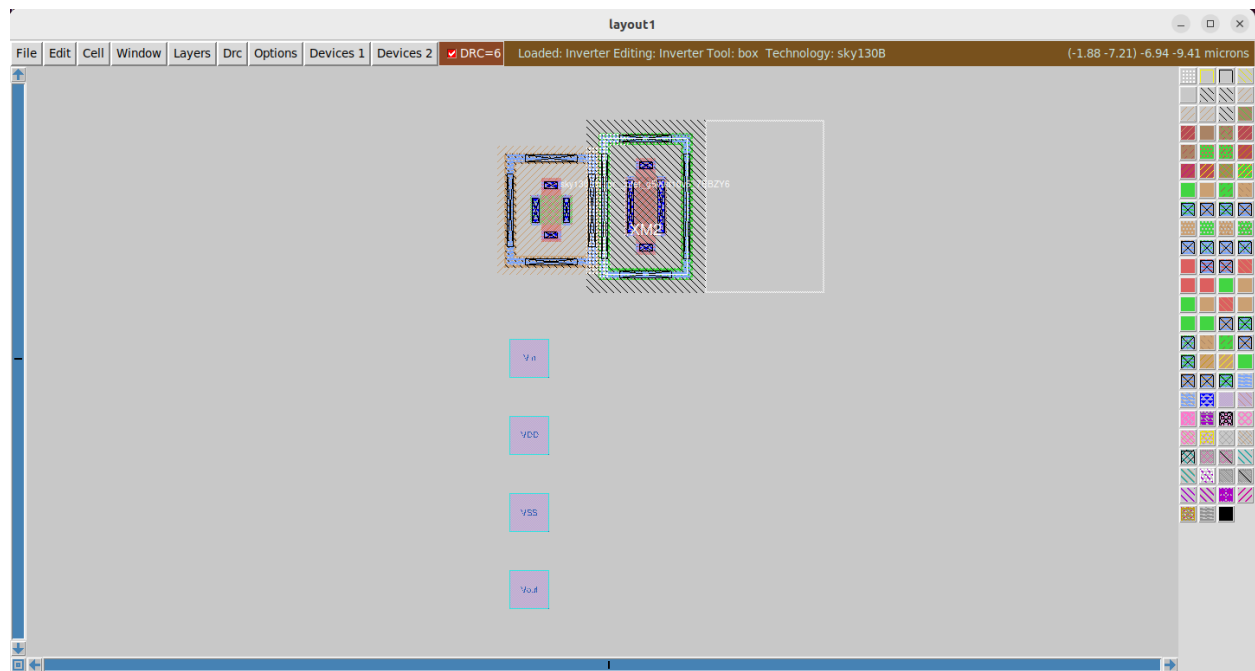


Figure 48: Imported Devices

If you have done everything correctly, you should see the devices from your schematic instantiated as well as the pins from your schematic placed. If you do not see this, the most likely culprits are that you did not extract the netlist from your schematic correctly or do not have the correct technology file selected.

Now, it is time to begin doing the layout. This process is a lot more complicated and much harder to describe in text format than creating the schematics. As such, I will not be able to guide you very well through how to make all of the connections and move things around. However, I will be able to describe what you should have in your layout and guide you through good practices.

What I recommend to do, is to watch the video linked above on how to create an op-amp layout, as it does a tremendous job of explaining basic commands/shortcuts of Magic so that you can create your own layout. It also talks about best practices when doing layout. Next, I recommend playing around with Magic yourself a little bit, trying out different shortcuts/commands to see what they do. Below, I have included some of the more useful shortcuts, commands, and settings that I use when creating layouts. Once you feel comfortable with Magic, then I recommend following along with the rest of this example to see if you can successfully create a layout of an inverter and extract its netlist to pass LVS.

There are many more shortcuts and commands that can be used with Magic, so I recommend checking out both the cheat sheet and the Magic documentation linked above. This list only encapsulates some of my most used shortcuts/commands when using Magic

- Helpful Shortcuts/General Use
 - In box mode, pressing left click will start the corner of a selection box, and right click sets the opposite corner of the selection box (this will take some playing around with to get used to).
 - To create nets, there are two main ways. The first is to create the desired net with the cursor box, then middle-click the layer you want on the right-hand side of the software. You can also middle-click a layer that has already been established in the layout. The other way is to use the terminal to paint the layer you want - using "paint [layer]."
 - Using the arrow keys will move the screen around.
 - Scrolling moves the window up + down
 - Shift + scrolling moves the window side to side.
 - Using the arrow keys on the num pad will move devices/nets around by the smallest increment. Very useful for doing precise placements.
 - "v" will put your window into full view, allowing you to see all devices in your layout.
 - "z" will zoom in.
 - "Shift + z" will zoom out.
 - "x" will expand the cells in your layout. You will have to select the entire device to expand its view. This will be helpful when you close Magic and reopen a layout you were working on previously.
 - "i" selects the device your cursor is over.
 - After selecting a device with "i", press "CTRL + p" to open the settings window on a device.
 - "s" selects the tile that the cursor is over. Repeatedly pressing "s" in the same spot will show what thing you highlighted over is connected to (this takes some playing around with). Very useful to see how your nets are connected.

- **“m”** will allow you to move nets/devices after they have been selected with **“s”** or **“j”**.
- **“space”** will open up the wiring tool. This tool acts much like how wiring nets up in Cadence works. Very useful to get used to. I would recommend playing around with it. Pressing **“space”** again will put you back in the box tool.
- In the wiring tool, if connecting two metal layers together, when ending them can, press **“shift + left click”**; this will connect the two metal layers and place a via down as well.
- Helpful Commands for Magic Terminal
 - After selecting a port, typing **“port index”** in the terminal shows where the port is in the netlist created by Magic. Helpful for LVS and post-layout.
 - After selecting a port, typing **“port class [direction]”** will allow you to change the port direction (input, output, input/output). To see what the port is classified as simply type **“port class”**.
 - Typing **“rotate [number]”** in the terminal will rotate a device by that number.
 - Using **“clock [number]”** will rotate the device clockwise by that number.
 - Typing **“box”** in the terminal will show you the size of your box.
 - Under **“Devices 1”** in the top menu - use the **“vias[number]”** there to create vias for your devices. It is recommended to create your vias this way, as it will give you the proper sizing that is needed for them. Via 1 will connect metal 1 to metal 2. Via 2 will connect metal 2 to metal 3.
 - Pins are defaulted to metal 1. If you want to change this, then select the pin, and in the terminal, type **“setlabel layer [layer]”**. After you have done this, you will also want to type **“erase [original layer]”**.
 - Typing **“fill [direction (n, e, s, w)] [layer]”** will fill in the layer to the farthest point your box is at. Hard to describe, so I would recommend playing around with this. Described at 46:00 in the op-amp layout video.
- Helpful Settings
 - Under **“options”**, select the crosshair. This will make it easier to see exactly where the cursor is and allow you to line things up easier.
 - Under **“window”**, set **“Grid on”**, and set your grid size. This will make it even easier for you to line things up in your layout and will be good to use when you

are just starting out. You can even turn on “**Snap-to-grid on**”, if you feel like it will be beneficial to you.

Once you feel comfortable enough navigating Magic, you can continue with making a layout for the inverter. The first thing that you should do is decide if you want to change any of the settings of the devices. This setting window is opened after pressing “*i*” to select a device and “**CTRL + p**”. The settings window should look similar to the figure below.

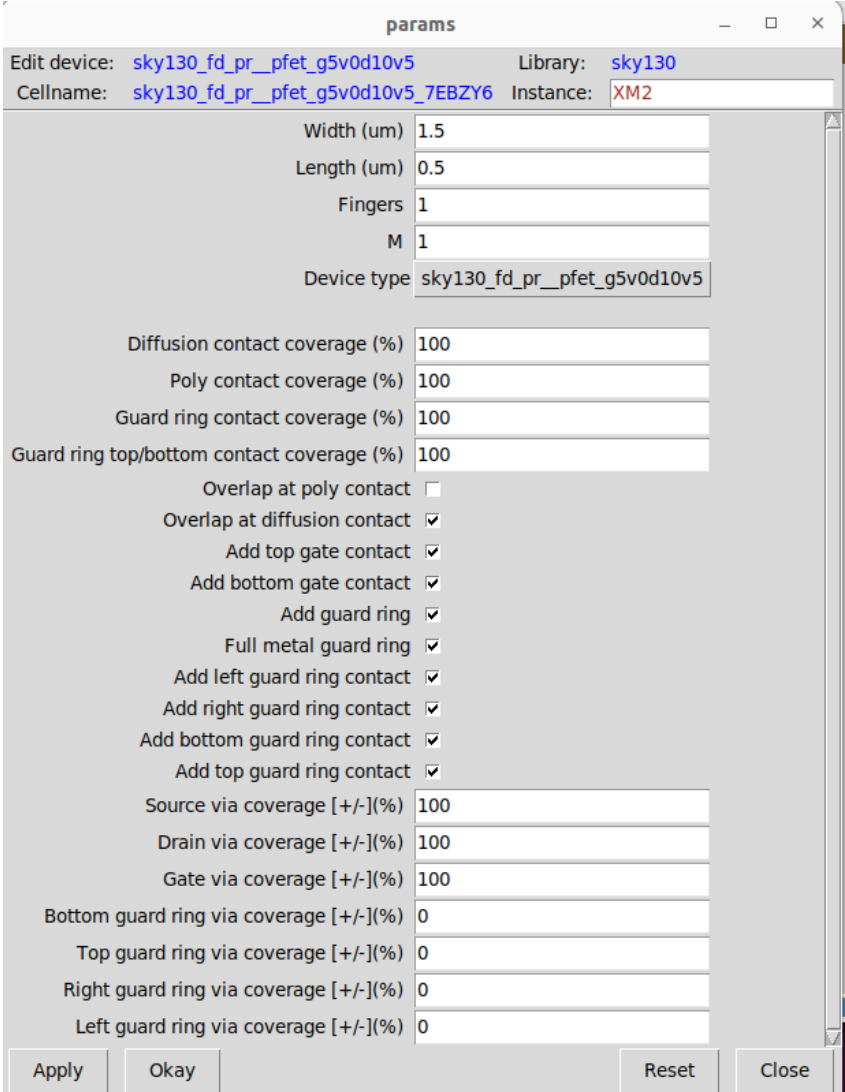


Figure 49: Device Settings

Here, you can change a lot of settings, such as if there is a top or bottom gate contact or if there is a guard ring around the device. I do not recommend changing any of these settings unless you know what you are doing or you are sure you aren't worried about noise/gradient effects on a device. For this example, I will not be changing any of these settings.

The next thing that you should do is move the PMOS device above your NMOS device. It is a good idea to put them fairly close together but ensure that they are not close enough to have

DRC errors. Another good idea is to line up the edges of the guard rings of each device so that it will make connections between them easier. In our classes, we have not covered guard rings, so I recommend looking up their use. Essentially, they will act to isolate the device from outside noise. Next, after you have lined everything up, it is a good idea to use metal 1 to connect the two gates together. This is a good idea for analog devices because it will help to reduce the electric field gradient. Make sure that when you make these connections, you do not have any DRC errors. If you have followed these instructions so far, you should have something that looks similar to the figure below.

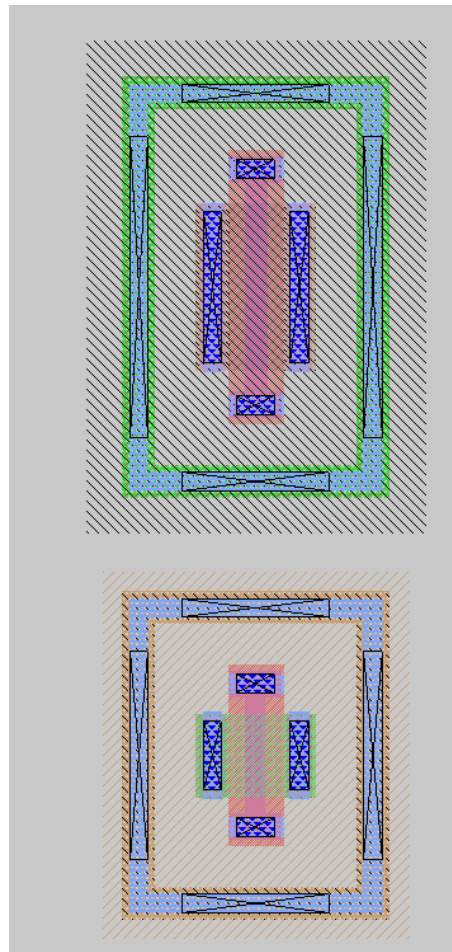


Figure 50: Nfet and Pfet Devices Ready to be Connected

Next, it is time to start making connections to power and ground. For analog layouts, you will want to use the layer “*locali*” to connect up your guard rings and create a large area so that the power and ground planes will have low resistance. Next, you will want to wire up metal 1 over part of your guard ring and on top of the “*locali*” layer. This will double up your connection for power and ground and essentially means they will be more well-connected. This is what digital standard cells use as well. Finally, on top of the metal 1 layer, place down a smaller strip of “*viali*” layer. This is where contact cuts will be made on your device. Your layout should look similar to the figure below if you have been following these steps.

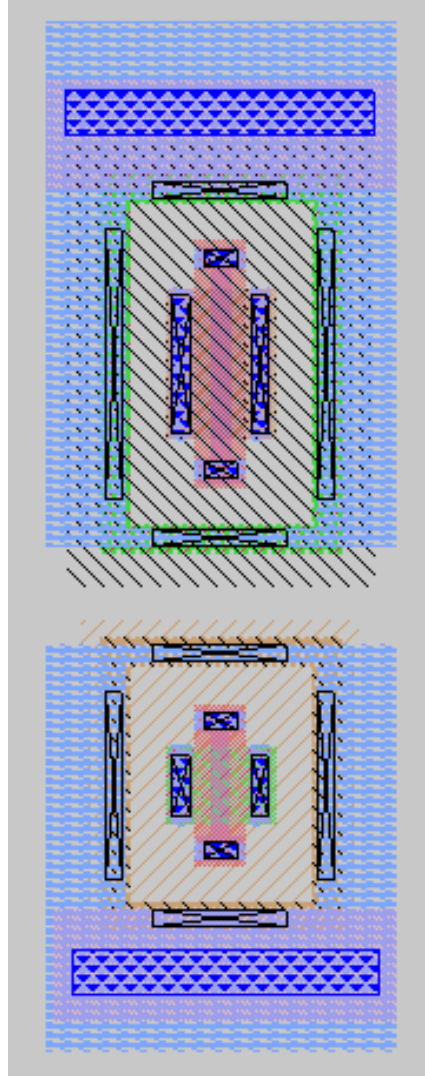


Figure 51: Nfet and Pfet Power and Ground Connections with Contact Cuts

At this point, it is a good idea to save. Navigate to the top menu and select “file”, then in the drop-down menu, select save. You will get a prompt asking you what you will want to do. Just press “autowrite”, as this will save this layout and the devices that were created for this layout.

Now, it is time to make the power connections. For the power and ground connections, you can place them on the metal 1 layer connected to the guard ring. You will want to remove the “*viali*” layer from above it before you place them down. It is also a good idea to fully place the metal 1 layer above the “*locali*” layer on the top part of the PMOS and the bottom part of the NMOS. Once you have made enough room, you can move the “VDD” port to the top part of the PMOS and the “VSS” port to the bottom part of the NMOS. In my layout, I moved some of my layers around so they look more flush with my overall design. After these changes, your layout should look similar to the figure below.

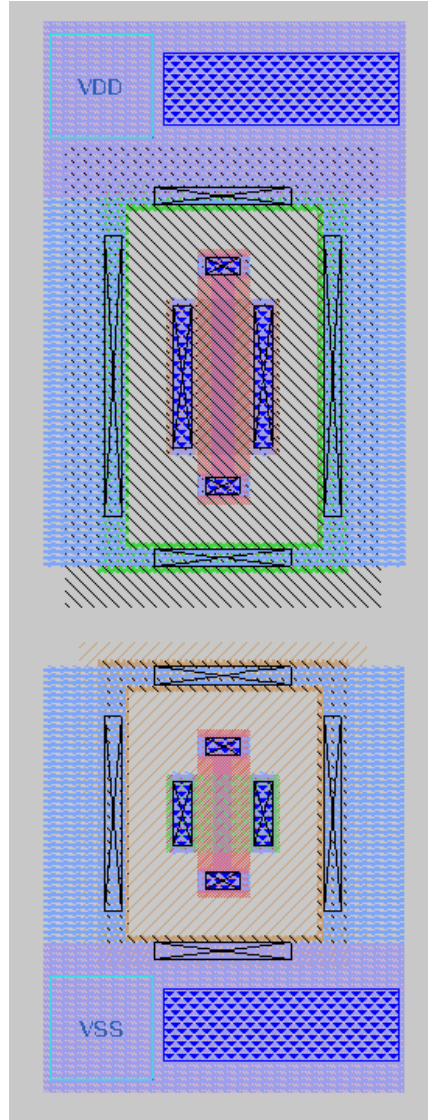


Figure 52: Ports Placed in Correct Spots

Next, it is time to make the final connections between the devices and hook up the input and output ports. It should be noted that it does not matter which side of the transistor you use for the source/drain, as they are interchangeable in the layout. For this layout I will use the left side of the gates for the connections to power/ground. I will then connect the gates and hook them up to the input. Finally, I will connect the right sides of the gates together and hook them up to the output. You should make sure that your nets are large enough so they can carry enough current that is required for your device. Make sure that you are checking and fixing any DRC issues that you run into when making these connections. Your final layout should look something like the figure below.

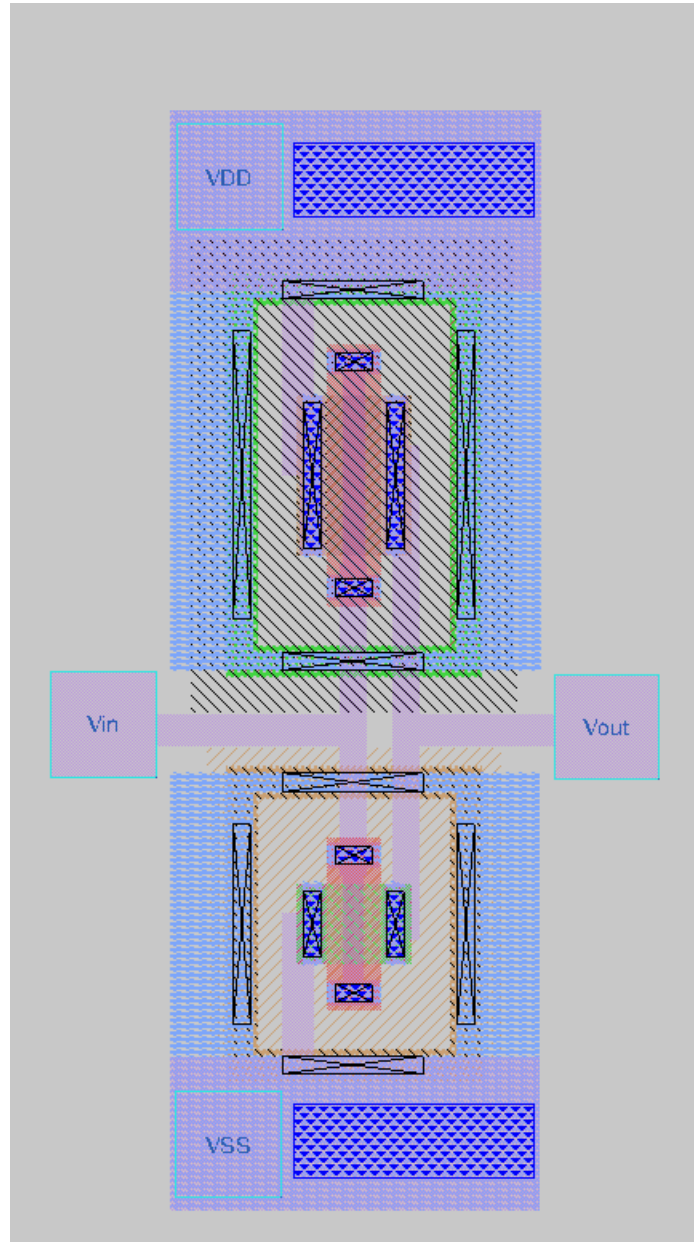


Figure 53: Final Hookups

With this, your layout is finished. As this is a simple design there are a lot of potential problems that we did run into. As you start scaling up your designs you may run into more issues with pins and their ordering, making connections over gates with higher-level metals, etc. These are covered pretty well in the op-amp layout video linked above. Also, many layout issues and their solutions are covered in the Slack channel that was mentioned earlier, so take a look around, as there is very good advice for more complex layout designs.

For now, with this design, our layout is complete and we can move on to running LVS. First, make sure to save your design. Then, to extract the proper netlist to make comparisons between layout and schematic, type the following commands into the Magic terminal:

- **extract all**
- **ext2spice lvs**
- **ext2spice**

These commands will essentially extract the nets of your devices into a .ext file. Then it will convert the .ext file into a spice netlist that will have the proper specifications set so it can be compared in an LVS test. Both the .ext file and the .spice file, by default, are saved in the current directory you are in. For now, you can exit out of Magic.

Once you are back in the terminal type “ls” and you should see a bunch of new files. They should look similar to my files seen in the figure below.

```

jthater@Ubuntu-Example: ~/Project
jthater@Ubuntu-Example:~/Project/magic$ magic -rcfile sky130B.magicrc
jthater@Ubuntu-Example:~/Project/magic$ ls
Inverter.ext  Inverter.spice  sky130_fd_pr_nfet_g5v0d10v5_6XHARQ.ext  sky130_fd_pr_pfet_g5v0d10v5_7EBZY6.ext
Inverter.mag  sky130B.magicrc  sky130_fd_pr_nfet_g5v0d10v5_6XHARQ.mag  sky130_fd_pr_pfet_g5v0d10v5_7EBZY6.mag
jthater@Ubuntu-Example:~/Project/magic$

```

Figure 54: Updated Files in Directory

These are the files that were generated from the commands that were run above. The main one that we are concerned about is the one titled “Inverter.spice”, as this is the extracted spice netlist of our layout. However, the way that I did things, this file will have the same name as my schematic spice netlist. To correct this, I will change the extracted magic spice netlist to a more appropriate name. To do this run a command similar to:

- **mv Inverter.spice InverterMag.spice**

This will rename the file to InverterMag.spice, so when we grab the schematic spice netlist the two files will not clobber each other. Before we run LVS we need to grab two more files. One is the schematic spice netlist and the other is the setup file for Netgen to properly compare the netlists for the sky130B process. If you have everything in their default locations you can run the following commands to copy the files into your current directory:

- **cp /usr/local/share/pdk/sky130B/libs.tech/netgen/sky130B_setup.tcl .**
- **cp /home/jthater(user)/xschem/simulations/Inverter.spice .**
- **ls**
 - To verify files have been copied over properly.

Once this is done there is one more thing you need to learn. With the netlists you extracted, you have extracted the “top-level view” of them. This is important because if you don’t do this, the comparison between the two spice netlists will not be comparable; thus, an LVS test cannot be passed. In these top-level netlists, they make calls to subcircuits. It is a good idea to check both

the schematic and Magic extracted spice netlists to understand what is happening. My netlists are shown in the figures below.

```
* NGSPICE file created from Inverter.ext - technology: sky130B
.subckt sky130_fd_pr__nfet_g5v0d10v5_6XHARQ a_n108_n75# a_n50_n163# a_n242_n297# a_50_n75#
X0 a_50_n75# a_n50_n163# a_n108_n75# a_n242_n297# sky130_fd_pr__nfet_g5v0d10v5 ad=0.218 pd=2.08 as=0.218 ps=2.08 w=0.75 l=0.5
.ends

.subckt sky130_fd_pr__pfet_g5v0d10v5_7EBZY6 a_50_n150# a_n50_n247# a_n108_n150# w_n308_n447#
X0 a_50_n150# a_n50_n247# a_n108_n150# w_n308_n447# sky130_fd_pr__pfet_g5v0d10v5 ad=0.435 pd=3.58 as=0.435 ps=3.58 w=1.5 l=0.5
.ends

.subckt Inverter Vin VDD VSS Vout
XXM1 VSS Vin VSS Vout sky130_fd_pr__nfet_g5v0d10v5_6XHARQ
XXM2 Vout Vin VDD VDD sky130_fd_pr__pfet_g5v0d10v5_7EBZY6
.ends
```

Figure 55: Extracted Magic Spice Netlist

```
** sch_path: /home/jthater/Project/xschem/Inverter.sch
.subckt Inverter Vin VDD VSS Vout
*.PININFO Vin:I VDD:B VSS:B Vout:O
XM1 Vout Vin VSS VSS sky130_fd_pr__nfet_g5v0d10v5 L=0.5 W=0.75 nf=1 m=1
XM2 Vout Vin VDD VDD sky130_fd_pr__pfet_g5v0d10v5 L=0.5 W=1.5 nf=1 m=1
.ends
.end
```

Figure 56: Schematic Netlist

As can be seen both the schematic and Magic netlist have a subcircuit reference to “Inverter Vin VDD VSS Vout”. When LVS is run we will be comparing these two subcircuits to see if they match. This subcircuit name will be important when running LVS, so make sure to take note of what your subcircuit names are.

With all of this done, LVS can finally be run. There are many ways to run Netgen to do an LVS check, but I prefer doing it directly in the terminal. A general guide to running LVS is a command like this:

- **netgen -batch lvs “schematic.spice schematic_subcircuit_name” “magic.spice magic_subcircuit_name” setup_file.tcl**

With the way I have set up my naming and subcircuit names, I will run the following command to do LVS:

- **netgen -batch lvs “Inverter.spice Inverter” “InverterMag.spice Inverter” sky130B_setup.tcl**

Running this LVS will generate a lot of output, but all of will be logged to a file called “comp.out”. Opening up this file using a text editor will allow you to see the results of your LVS check. My generated “comp.out” file is shown in the figure below.

```

Cell pin lists are equivalent.
Device classes sky130_fd_pr__nfet_g5v0d10v5 and sky130_fd_pr__nfet_g5v0d10v5 are equivalent.

Circuit 1 cell sky130_fd_pr__pfet_g5v0d10v5 and Circuit 2 cell sky130_fd_pr__pfet_g5v0d10v5 are black boxes.
Warning: Equate pins: cell sky130_fd_pr__pfet_g5v0d10v5 is a placeholder, treated as a black box.
Warning: Equate pins: cell sky130_fd_pr__pfet_g5v0d10v5 is a placeholder, treated as a black box.

Subcircuit pins:
Circuit 1: sky130_fd_pr__pfet_g5v0d10v5 | Circuit 2: sky130_fd_pr__pfet_g5v0d10v5
-----|-----
1 | 1
2 | 2
3 | 3
4 | 4
-----

Cell pin lists are equivalent.
Device classes sky130_fd_pr__pfet_g5v0d10v5 and sky130_fd_pr__pfet_g5v0d10v5 are equivalent.
Flattening unmatched subcell sky130_fd_pr__nfet_g5v0d10v5_6XHARQ in circuit Inverter (1)(1 instance)
Flattening unmatched subcell sky130_fd_pr__pfet_g5v0d10v5_7EBZY6 in circuit Inverter (1)(1 instance)

Subcircuit summary:
Circuit 1: Inverter | Circuit 2: Inverter
-----|-----
sky130_fd_pr__nfet_g5v0d10v5 (1) | sky130_fd_pr__nfet_g5v0d10v5 (1)
sky130_fd_pr__pfet_g5v0d10v5 (1) | sky130_fd_pr__pfet_g5v0d10v5 (1)
Number of devices: 2 | Number of devices: 2
Number of nets: 4 | Number of nets: 4
-----

Netlists match uniquely.

Subcircuit pins:
Circuit 1: Inverter | Circuit 2: Inverter
-----|-----
VSS | VSS
Vin | Vin
Vout | Vout
VDD | VDD
-----

Cell pin lists are equivalent.
Device classes Inverter and Inverter are equivalent.

Final result: Circuits match uniquely.

```

Figure 57: LVS Pass

As can be seen in the figure, the number of devices and nets match both my schematic and layout. The order of my pins also matches (VSS, Vin, Vout, and VDD respectively). As such, my two “circuits match uniquely” which means that it passes the LVS check.

If you did not get a passed LVS, then you have done something wrong along the way. If you made it to the layout with all the devices generated and ports placed when you imported the Spice netlist, then you most likely have something incorrect with your layout or the way you extracted it. Using the comp.out file is a great place to begin troubleshooting why your two circuits may not be matching.

With this, LVS is completed, and there is only one more step left in the analog design flow before we begin integrating our design into the Efabless environment. That step is the extracted post-layout simulation.

Parasitic Netlist Extraction + Post-Layout Simulation

For the last step in the analog design flow, a simulation must be done on the post-layout parasitic netlist to ensure that after layout, your device will still work as simulated. The parasitic netlist will be extracted using magic. This netlist will then be read into the symbol for the circuit that you created in Xschem. This process is actually fairly straightforward compared to the other steps involved in the analog design flow, but there are still things to look out for to make sure that this process is done correctly. If you have been following this guide, then Magic and Xschem will already be set up with the correct configuration files, so there is no need to copy any more files over to your directories.

As for tutorials on a full R+C extraction of a netlist and then subsequent simulation, there are not any that exist at the time of writing. There is a video tutorial that shows how to take a netlist and have Xschem/Ngspice run a simulation on it. However, it does not show how to perform a full R+C extraction. This video is here:

<https://www.youtube.com/watch?v=8SMBSYiLbHM&t=114s> (In the video, he types “primitive”, in all of my tests, this does not work. Use “primitive” instead.)

As for how to do more complex extractions, your best bet would be to search the Slack channel with terms such as “parasitic extraction”, “cthresh”, “extresist”, or “extract”. Using these terms, you should find posts talking about how to perform netlist extraction and some issues that may come along with it. For the example, in the next section, I will show what I have used to perform full R+C extractions.

Example Inverter R+C Extraction + Post-Layout Simulation

Now that you have created a layout and ensured there were no DRC or LVS issues, it is time to do a post-layout simulation of the inverter. Before proceeding I recommend creating a copy of your layout in case something goes awry while extracting the parasitic netlist. I copied my layout file to another directory. Once you have done this, reopen up magic with your inverter layout. You can do this in the terminal by simply typing:

- **magic Inverter.mag**

This will open up Magic with your device, as well as the correct technology file that was used to create your layout. From here you will type commands in the Magic terminal to generate the correct netlist and extract it. For the commands listed below, I will assume you are doing a full R+C extraction. The commands to type in the terminal are as follows:

- **select top cell**
- **expand**
- **flatten Inverter_Flat**

- **load Inverter_Flat**
- **cellname delete Inverter**
- **cellname rename Inverter_Flat Inverter**
- **select top cell**
- **extract do local**
- **extract all**
- **ext2sim labels on**
- **ext2sim**
- **extresist tolerance 10**
- **extresist**
- **ext2spice lvs**
- **ext2spice cthresh 0**
- **ext2spice extresist on**
- **ext2spice**

These commands created a flattened version of your layout, which you will then load. This is done so that the parasitic resistance can be properly extracted. You will then delete the original inverter layout (why I said to copy your layout file). You do this so that the name of your subcircuit from the extracted netlist will match the subcircuit name in your schematic. You can skip the “*cellname ...*” steps if you plan to rename the subcircuit in your newly extracted netlist once it is generated.

Next, you do different types of extractions that will generate the correct R+C netlist. The two key lines of this extraction are “*extresist tolerance ...*” and “*ext2spice cthresh ...*”. These two lines will set the bottom threshold of the resistance/capacitances that will be in the extracted netlist. You can play around with these values if you like, but the values listed here will give you the best results the majority of the time. After running this extraction with no warnings you should get a spice netlist that looks similar to the one shown in the figure below.

```

* NGSPICE file created from Inverter.ext - technology: sky130B

.subckt Inverter Vin VDD VSS Vout
X0 Vout.t1 Vin.t0 VSS.t1 VSS.t0 sky130_fd_pr__nfet_g5v0d10v5 ad=0.218 pd=2.08 as=0.218 ps=2.08 w=0.75 l=0.5
X1 Vout.t0 Vin.t1 VDD.t1 VDD.t0 sky130_fd_pr__pfet_g5v0d10v5 ad=0.435 pd=3.58 as=0.435 ps=3.58 w=1.5 l=0.5
R0 Vin.n0 Vin.t1 133.161
R1 Vin.n0 Vin.t0 112.722
R2 Vin Vin.n0 0.93369
R3 VSS.n6 VSS.n2 1488.48
R4 VSS.n6 VSS.n3 1488.48
R5 VSS.n7 VSS.n3 1488.48
R6 VSS.n7 VSS.n2 1488.48
R7 VSS.n4 VSS.n2 1047.64
R8 VSS.n4 VSS.n3 1047.64
R9 VSS.n6 VSS.n5 146.25
R10 VSS.t0 VSS.n6 146.25
R11 VSS.n8 VSS.n7 146.25
R12 VSS.n7 VSS.t0 146.25
R13 VSS.n10 VSS.t1 119.695
R14 VSS.n2 VSS.n0 97.5005
R15 VSS.n3 VSS.n1 97.5005
R16 VSS.n5 VSS.n0 95.7785
R17 VSS.n5 VSS.n1 94.2726
R18 VSS.n8 VSS.n1 32.431
R19 VSS.n9 VSS.n0 27.6091
R20 VSS.t0 VSS.n4 20.2778
R21 VSS.n9 VSS.n8 4.99776
R22 VSS.n10 VSS.n9 0.869897
R23 VSS VSS.n10 0.100931
R24 Vout.n0 Vout.t0 168.077
R25 Vout.n0 Vout.t1 119.984
R26 Vout Vout.n0 0.664228
R27 VDD.n8 VDD.n7 1053.52
R28 VDD.n5 VDD.n3 1053.52
R29 VDD.n3 VDD.n2 194.792
R30 VDD.n7 VDD.n6 194.792
R31 VDD.n11 VDD.t1 167.714
R32 VDD.n4 VDD.n0 102.305
R33 VDD.n4 VDD.n1 101.165
R34 VDD.n5 VDD.n4 46.2505
R35 VDD.n9 VDD.n8 46.2505
R36 VDD.n9 VDD.n1 37.2504
R37 VDD.n8 VDD.n2 36.1142
R38 VDD.n6 VDD.n5 36.1142
R39 VDD.n10 VDD.n0 32.9612
R40 VDD.n7 VDD.n1 23.1255
R41 VDD.n3 VDD.n0 23.1255
R42 VDD.n6 VDD.t0 8.55241
R43 VDD.t0 VDD.n2 8.55241
R44 VDD.n10 VDD.n9 4.12253
R45 VDD.n11 VDD.n10 0.856491
R46 VDD VDD.n11 0.085929
C0 VDD Vin 0.652f
C1 VDD Vout 0.3f
C2 Vout Vin 0.537f
.ends

```

Figure 58: Parasitics Generated from Layout

If you have a similar-looking netlist, then you have most likely extracted everything correctly. In the netlist in the figure above, you can see 47 different parasitic resistances and 3 parasitic capacitances that were generated.

From here, you can navigate back to Xschem. In Xschem, go ahead and open up the symbol file that you created. In this symbol file, click on an empty space and press the letter “q” on your keyboard. This will open up a window with text describing the circuit. In this file, you will want to change one line. You should see a line that says “*type=subcircuit*”. You will want to change this line to “*type=primitive*”. Your text file should look like the figure below.

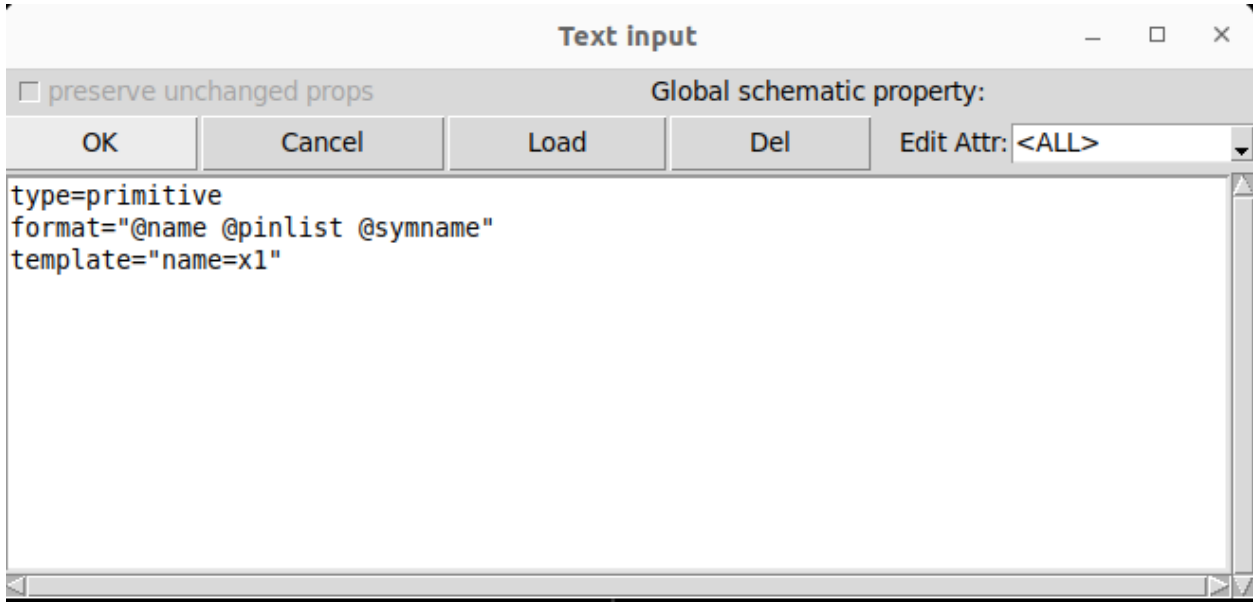


Figure 59: Editing Symbol File

Once this line has been written, press “OK” and save the symbol file. Next, you will want to open up your inverter testbench. In your inverter testbench, all you have to do to get it to run your post-layout parasitic netlist is add a “.include ...” line in your “code_shown” block. This include statement will link to the path of your extracted netlist. For the way I have configured my directories, the added line will be:

- **.include /home/jthater/Project/magic/Inverter.spice**

In the “code_shown” block, the updated code is shown in the figure below.

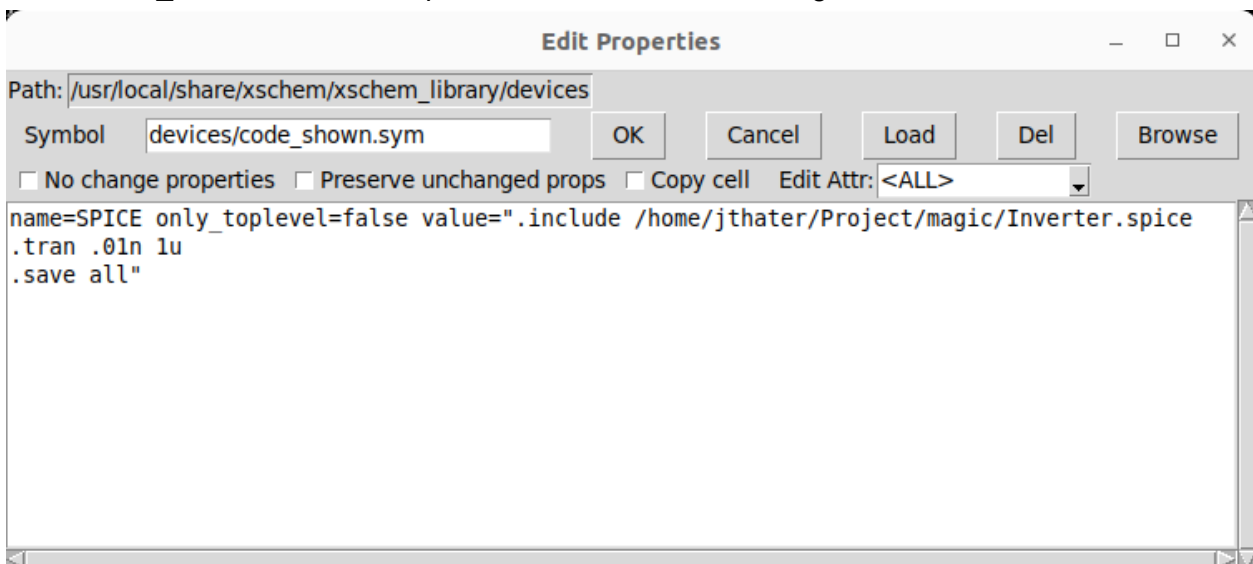


Figure 60: Updated Code in Control Block

With all of this done, there is only one more very important thing to check. You have to ensure that the pin orders in your testbench layout match the pin orders of your extracted netlist. Whenever you change the symbol file type from “*subcircuit*” to “*primitive*,” the testbench loses connection with the original schematic. As a result, there is a chance that the pin order for the testbench will get scrambled. If this happens and you do not correct it, then the pins in your extracted netlist will get the wrong input/output voltage and will almost certainly make your simulation turn out incorrect. To check if these netlists are incorrect, simply view the extracted netlist pin order and your testbench pin order. I will show an example of mine in the two figures below. I have underlined in green what you should be paying attention to.

```

jthater@Ubuntu-Example: ~/Project/magic
* NGSPICE file created from Inverter.ext - technology: sky130B
.subckt Inverter Vin VDD VSS Vout
X0 vout.t1 Vin.t0 VSS.t1 VSS.t0 sky130_fd_pr__nfet_g5v0d10v5 ad=0.218 pd=2.08 as
=0.218 ps=2.08 w=0.75 l=0.5
X1 Vout.t0 Vin.t1 VDD.t1 VDD.t0 sky130_fd_pr__pfet_g5v0d10v5 ad=0.435 pd=3.58 as
=0.435 ps=3.58 w=1.5 l=0.5
R0 Vin.n0 Vin.t1 133.161
R1 Vin.n0 Vin.t0 112.722
R2 Vin Vin.n0 0.93369
R3 VSS.n6 VSS.n2 1488.48
R4 VSS.n6 VSS.n3 1488.48
R5 VSS.n7 VSS.n3 1488.48
R6 VSS.n7 VSS.n2 1488.48
R7 VSS.n4 VSS.n2 1047.64
R8 VSS.n4 VSS.n3 1047.64

```

Figure 61: Magic Netlist Pin Order

```

/home/jthater/xschem/simulations/InverterTB.spice
** sch_path: /home/jthater/Project/xschem/InverterTB.sch
** .subckt InverterTB
V1 Vin GND pulse(0 1.8 1ns 1ns 1ns 5ns 10ns)
V2 VDD GND 2.5
x1 VDD Vin Vout GND Inverter
**** begin user architecture code

.include /home/jthater/Project/magic/Inverter.spice
.tran .01n lu
.save all

.param mc_mm switch=0
.param mc_pr switch=0
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/corners/tt.spice
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/r+c/res_typical_cap_typ
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/r+c/res_typical_cap_typ
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/corners/tt/specialized_c

**** end user architecture code
** .ends
.GLOBAL GND
.end

```

Figure 62: Testbench Netlist Pin Order

As can be seen in the figure, the extracted netlist has a pin order of “Vin VDD VSS Vout” while the testbench has a pin order of “VDD Vin Vout GND” (GND is the same as VSS). So, from this it can be seen that the pin orders do not match and must be correct.

To correct this, simply edit the testbench netlist pin order to match that of the extracted netlist pin order. This does not work the other way, so do not try to change the extracted netlist pin order. For my example, once I correct the pin order, my updated testbench netlist looks like the figure below.

```

/home/jthater/.xschem/simulations/InverterTB.spice
** sch_path: /home/jthater/Project/xschem/InverterTB.sch
** .subckt InverterTB
V1 Vin GND pulse(0 1.8 1ns 1ns 1ns 5ns 10ns)
V2 VDD GND 2.5
x1 Vin VDD GND Vout Inverter
**** begin user architecture code

.include /home/jthater/Project/magic/Inverter.spice
.tran .01n 1u
.save all

.param mc_mm_switch=0
.param mc_pr_switch=0
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/corners/tt.spice
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/r+c/res_typical_cap_typ
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/r+c/res_typical_cap_typ
.include /usr/local/share/pdk/sky130B/libs.tech/ngspice/corners/tt/specialized_c

**** end user architecture code
** .ends
.GLOBAL GND
.end

```

Figure 63: Updated Testbench Netlist Pin Order

With this correct netlist, all that needs to be done is to run a simulation, just like you would any other simulation. It should be noted that each time you regenerate the testbench netlist, you will have to change the pin order again. My post-layout results are shown in the figure below.

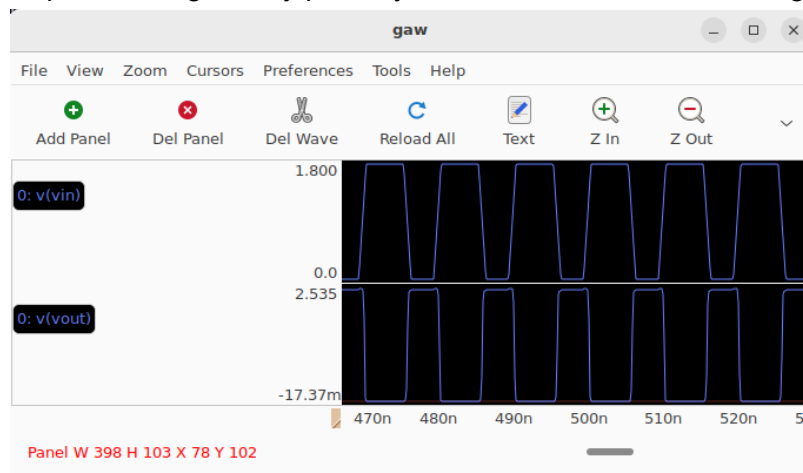


Figure 64: Post-Layout Waveforms

Once you get a simulation - that looks correct - you have successfully completed the last step of the analog design flow. All that is left is to integrate your design into the Caravan harness through Efabless.

Integration of Design with Harness Through Efabless

This is the final step in the open-source analog design flow through Efabless. At this point, your circuit has been created, laid out, had post-layout simulations done, and is ready to be fabricated. In order to have it fabricated, it has to be hooked up to the Caravel harness and pass certain precheck that Efabless has to ensure that your design is up to par to actually be fabricated.

It is important to understand the harness and all of the pins you can connect your circuit design to. It is also important to understand what prechecks will be run on your circuit. There is not much documentation for this process. The best way to understand this process is to read the GitHub repository READMEs and to view the Slack channel, where many questions about this process have been answered. The GitHub repository can be found at the link below:

https://github.com/efabless/caravel_user_project_analog

I heavily recommend going through all of the files that are made available to you in this repository, as it houses an example project that is properly hooked - both in the schematic and the layout. It also passes all of the main prechecks that Efabless runs. A lot of the things done may not make sense until you have done them yourself, so following along with the example done below will help out.

Before being able to successfully set up and run precheck, you will need to install Docker on your machine. The installation process is slightly more complicated than just running a single line to install everything. The process for installing Docker on an Ubuntu machine can be found at the link below:

<https://docs.docker.com/engine/install/ubuntu/>

This page will walk you through how to install Docker in many different ways. It will also walk through troubleshooting tips, so I will not cover how to install Docker in this document. I installed Docker using the *“Install using the apt repository”* way, and everything worked for me. Once you have Docker installed and have familiarized yourself with the analog framework, you can move toward the example below.

Example Inverter Harness Hookup

Now that your inverter has gone through all the steps of the analog design flow, it is time to begin hooking it up with the Harness and then run precheck on your design. There is quite a bit of setup and installation for this process. The first step in this process is to clone the Caravel analog framework from the GitHub repository. You can clone the repository by using this command:

- **git clone** https://github.com/efabless/caravel_user_project_analog

Once you have cloned the repository, go ahead and `cd` into it. It is a good idea to look around at all the files that are here. Inside this repository is an example project. It is a good idea to look at this example project and see how it is hooked up to the harness through Xschem and Magic.

As of the time of writing this, the way to open the layouts is slightly janky. You may have noticed that you are unable to open the layouts to view their contents. To fix this, you have to edit a variable or manually edit the files to open them. I have tried to use the `$PDK` variable way and have been unsuccessful, so I just manually edit the magic files so they open with the correct library. To do this, use whatever text editor you have - for me, this is Vim - and open up one of the magic files. For example:

- `vim user_analog_project_wrapper_empty.mag`

This will open up a file that should look similar to the one seen below.

```
magic
tech $PDK
timestamp 1632839657
<< checkpoint >>
rect -680 351370 292680 352680
rect -680 630 630 351370
rect 291370 630 292680 351370
rect -680 -680 292680 630
<< metal2 >>
rect 262 -400 318 240
rect 853 -400 909 240
rect 1444 -400 1500 240
rect 2035 -400 2091 240
rect 2626 -400 2682 240
rect 3217 -400 3273 240
rect 3808 -400 3864 240
rect 4399 -400 4455 240
rect 4990 -400 5046 240
rect 5581 -400 5637 240
rect 6172 -400 6228 240
rect 6763 -400 6819 240
rect 7354 -400 7410 240
rect 7945 -400 8001 240
rect 8536 -400 8592 240
rect 9127 -400 9183 240
rect 9718 -400 9774 240
rect 10309 -400 10365 240
rect 10900 -400 10956 240
rect 11491 -400 11547 240
rect 12082 -400 12138 240
```

Figure 65: Wrapper File

The part that needs to be changed is in the second line. You will want to delete the “`$PDK`” part and add the PDK you will be using. For this example, change it to “`sky130B`”. If you have done this correctly, it should be updated to look like the figure below.

```
magic
tech sky130B
timestamp 1632839657
<< checkpoint >>
rect -680 351370 292680 352680
rect -680 630 630 351370
rect 291370 630 292680 351370
rect -680 -680 292680 630
<< metal2 >>
rect 262 -400 318 240
rect 853 -400 909 240
rect 1444 -400 1500 240
```

Figure 66: Updated Wrapper File

Remember to save the updated file, for Vim, you do this by typing “:wq”. You can verify that this has worked by opening up the magic file you just edited, and you should be able to see its contents. Once this is verified, you can go ahead and do the same for the other layouts in the folder.

Once you get an understanding of the files and the example project that is shown in this repository, you can start hooking up and integrating your own design. First, you will want to hook up your device to the correct pins using Xschem. To do this, I recommend copying the project wrapper over to the file where you have all of your designs. The file you will want to copy over has the name “*user_analog_project_wrapper.sch*”. Once you have copied the file over, go ahead and open it with Xschem. It should look like the figure below.

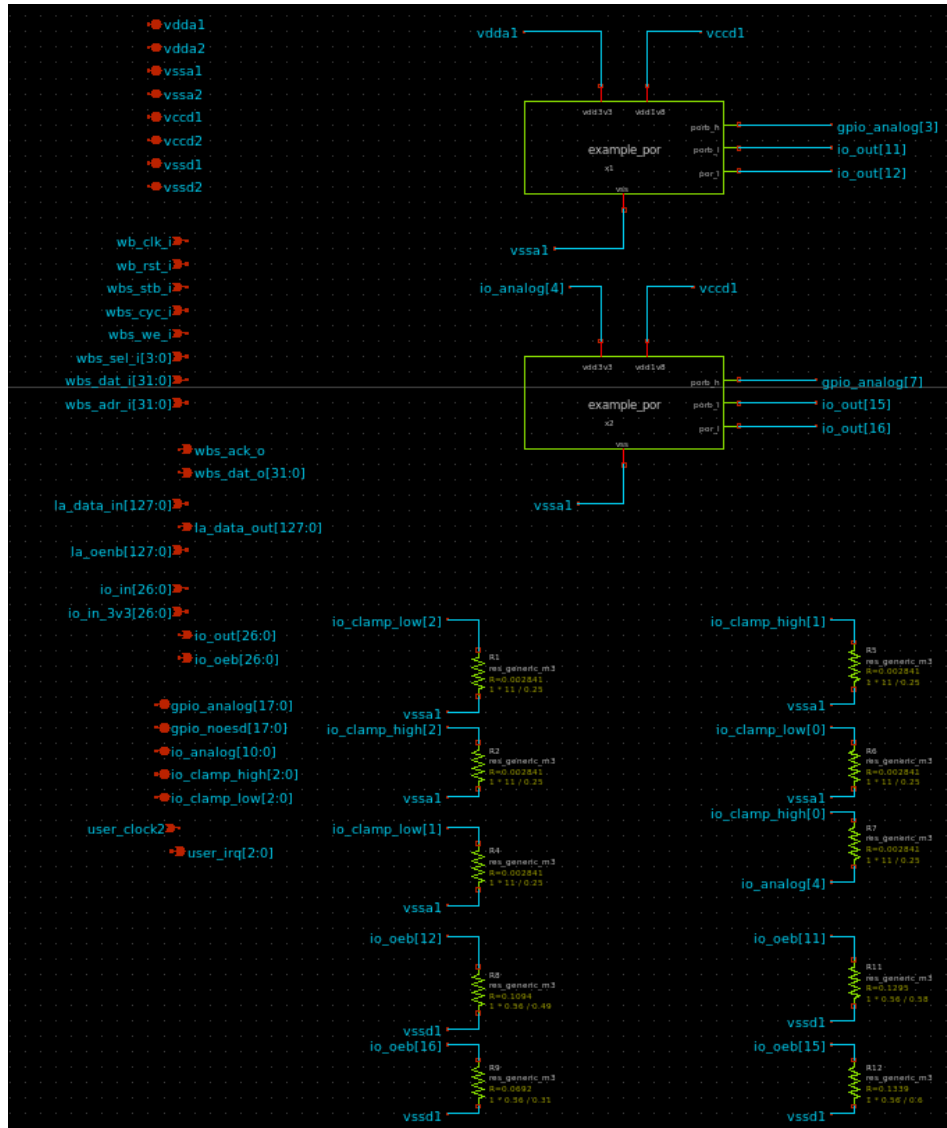


Figure 67: Example Schematic Wrapper

In this schematic, you can see the example project, some resistors that will be used as clamps on some of the I/O ports, and all of the pins that are available to you. It should be noted that not all of these pins are *actually* available to you. Most of these pins overlap with each other, so you will have to pick and choose which ones you want to use. To get an understanding of the pins that are available to you, how to use them, and what they map to in the management area, refer to the documentation found in the files here:

https://github.com/efabless/caravel_user_project_analog/tree/main/verilog/rtl

I would recommend before you hook anything up double check that pins that you are using do not overlap with each other, they operate as you expect them to, and know what they input/expect at the output. I am not too sure how many of these mistakes, if any, will be caught at precheck, so the onus is on you to ensure that everything is hooked up correctly.

With all of that out of the way, it is time to start hooking up the example inverter that was created. You can go ahead and delete everything that is in this schematic except for the pins that are on the left-hand side. Once you do this, you can place your inverter into the schematic. Next, go ahead and hook the inverter, as you see in the figure below.

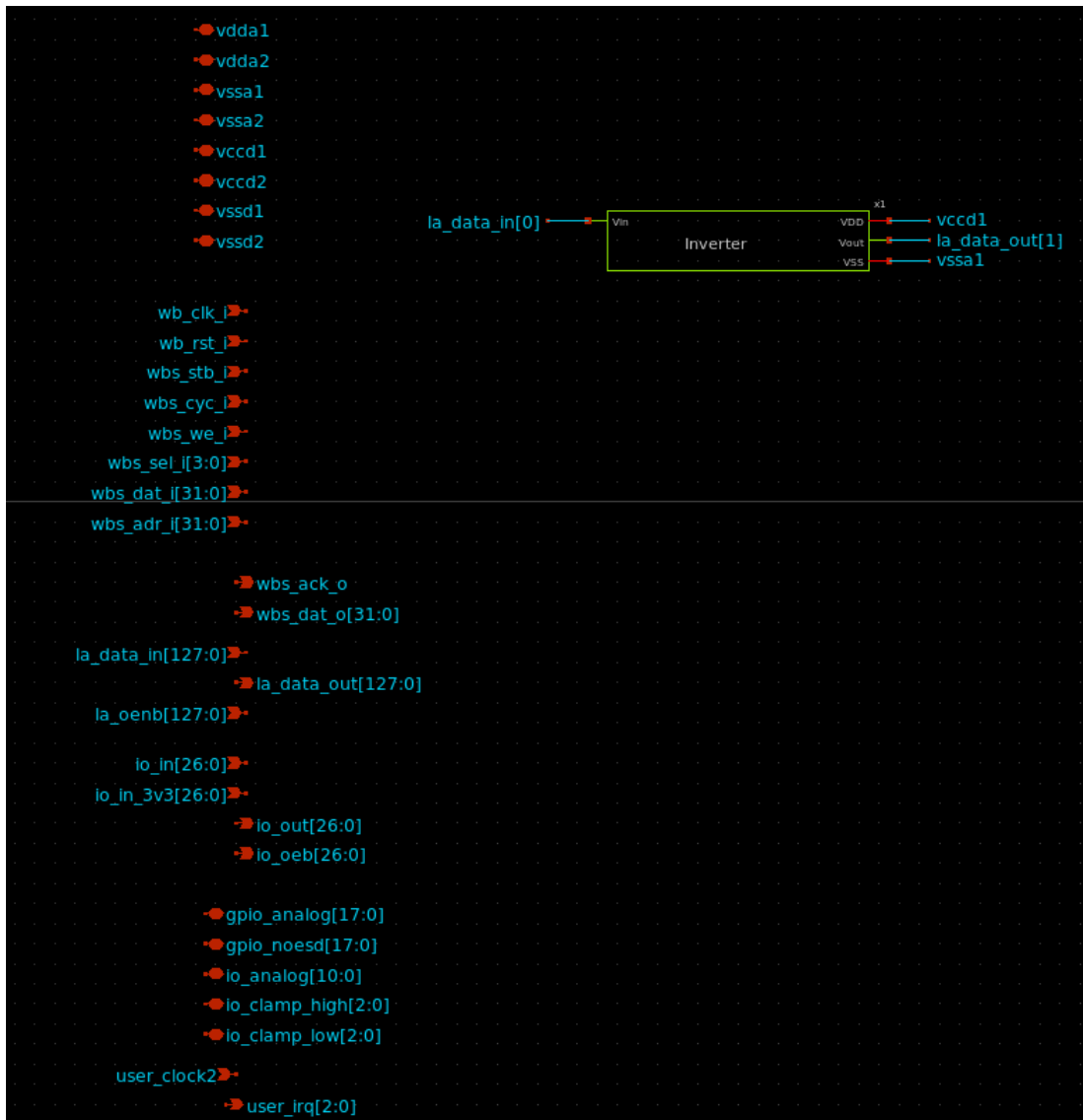


Figure 68: Inverter Hook-up in Schematic Wrapper

Hooking the device up this way will connect the VDD to the digital power supply of 1.8 V, and the VSS will be connected to the analog ground. The input will come from the logic analyzer, and the output will go to the logic analyzer.

Next, go ahead and generate the LVS netlist as you have done previously and save it. When you do this, you will get some warnings, but these can be safely ignored.

Once you have done all of this, it is time to hook the circuit up the harness in the layout. To do this, you will want to copy over the empty wrapper to your current working file with all of your

layouts. The file you want to copy over is *“user_analog_project_wrapper_empty.mag”*. Once you have this file copied over, go ahead and open it. While you can just open it using the command *“magic”*, I recommend always opening up layouts with the *“-rcfile sky130B.magicrc”* addition, as that, will give you more options, like premade vias, through the top toolbar (These options will be under dropdown menus title *“Devices 1”* & *“Devices 2”*). Once you open the empty wrapper, you should see something like the figure below.

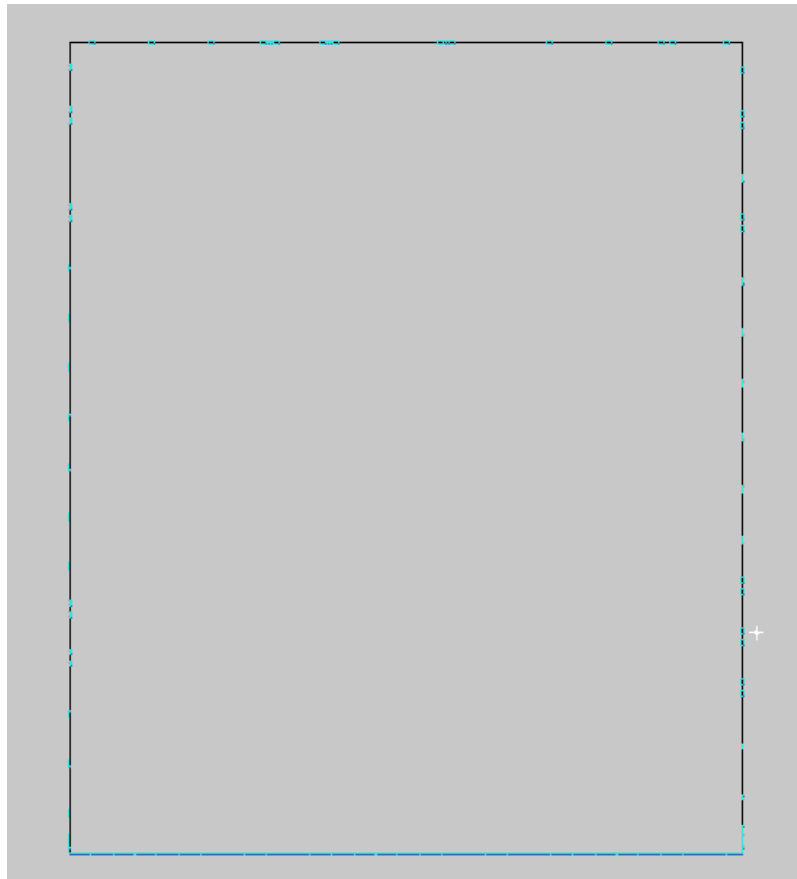


Figure 69: Empty Layout Wrapper

Once you open the empty wrapper, you will notice that there is a giant hollow box with some smaller blue boxes all around it. The area inside this box is the room you have to place your layout - it is massive, so unless you have hundreds of components, you shouldn't run out of room. The blue boxes you see are the pins that you can connect to. Remember that this is deceptive, and not every pin you see you will be able to connect to, as some will overlap in the management area. You can zoom in on these different blue boxes so you can get an idea of where all the pins are. I would recommend going through and drawing out a general guide of where all the pins are on a piece of paper so you don't have to search for a pin each time you want to connect to it.

Once you get an idea of where all the pins are, you can place your inverter into the layout and begin hooking it up to all the pins. Before you place your inverter, it may be a good idea to zoom in on an area and place the crosshair there so that you can easily find the inverter once placed.

To place your inverter into this layout, navigate to the top menu, and under “Cell”, click on “Place Instance”. This can be seen in the figure below.

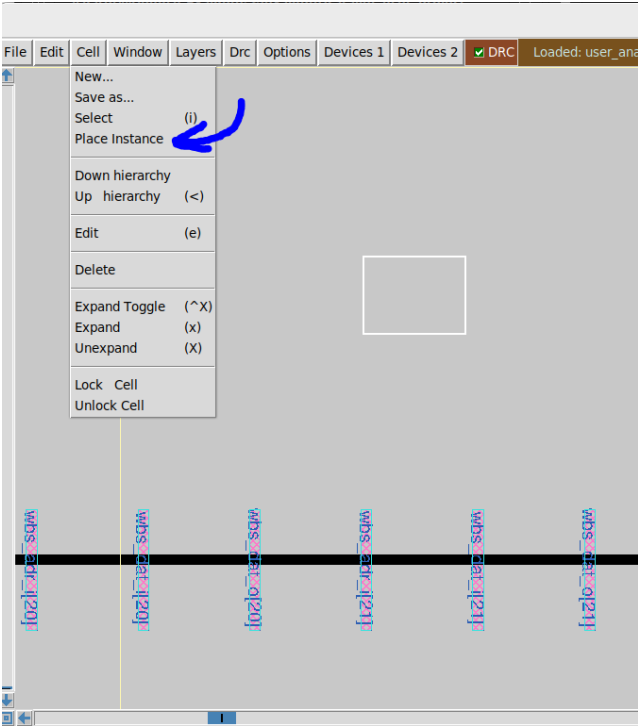


Figure 70: Place Inverter Layout

Navigate to where your inverter is using the menu and select it to be placed. Once you have placed your inverter, you should see something similar to the figure below.

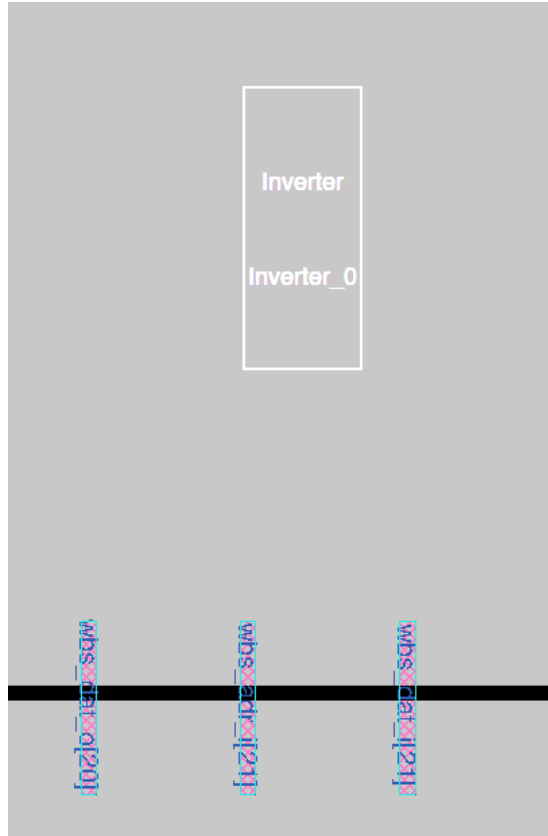


Figure 71: Inverter Layout Placed

In order to actually view the inverter, you have to “expand” it. To do this, select the inverter and press “x” on your keyboard. Doing this should reveal the full layout of your inverter. It should look like the figure below if you have done everything correctly.

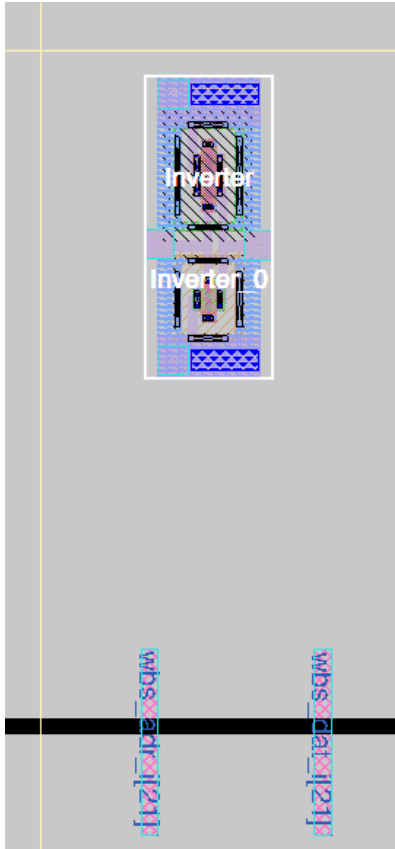


Figure 72: Inverter Layout Placed and Expanded

Once you have your inverter placed, go ahead and move it above the “*la_data*” pins so that connecting everything is easy. Since I’ve already gone over how to do layouts, I won’t go in-depth on how to do everything. Once you get to this stage where your circuits have been built, all you have to do is connect your circuit to the correct pins on the correct metal layer.

With the connections we made in Xschem, our input/output goes to the “*la_data*” pins, which are at the bottom. VSS is connected to the analog ground, which is at the top right, and VDD is connected to the digital power, which is also at the top. The “*la_data*” pins are on metal 2, and the ground/power pins are on metal 3. Go ahead and connect everything up to your circuit as was outlined in the schematic.

While doing this, you will definitely notice how massive the area is for your designs and may have trouble creating boxes that are even. I recommend learning some of the magic terminal commands, as it can make this process much easier and quicker. The link to the cheatsheet is found in the Magic tutorial earlier in this document.

Once you have everything hooked up, it should look similar to the figure below.

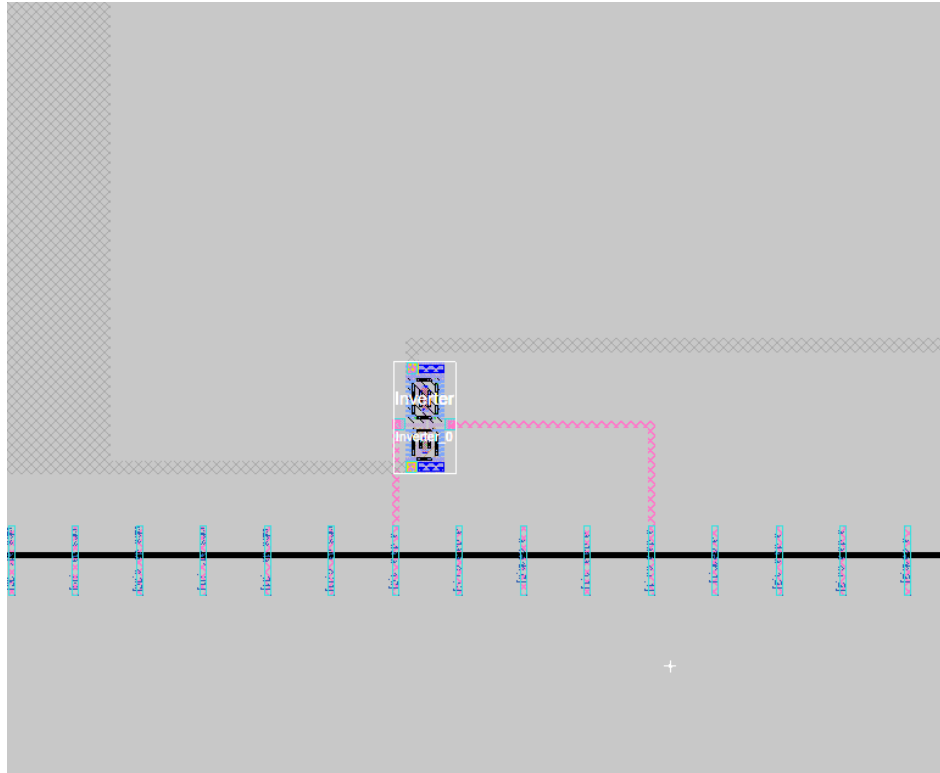


Figure 73: Inverter Hooked-up in Layout to Wrapper

Once you have everything hooked up, save a copy of it under the name “*user_analog_project_wrapper*”. Once this is done, go ahead and extract an LVS-ready netlist from it. In case you forgot these commands, they are:

- **extract all**
- **ext2spice lvs**
- **ext2spice**

Running these commands should generate the correct netlist. You may notice some warnings, but once again, they can be safely ignored. Once you have the spice netlist, rename it to something else so that when you copy the schematic-generated netlist into this file, it does not get overwritten. Once you do that, copy over the schematic-generated netlist and run LVS as you have done previously. In case you forgot that command, it is:

- **netgen -batch lvs “*user_analog_project_wrapperMag.spice* *user_analog_project_wrapper*” “*user_analog_project_wrapper.spice* *user_analog_project_wrapper*” sky130B_setup.tcl**

It should be noted that I renamed the magic-generated netlist to “*user_analog_project_wrapperMag.spice*” so that I could run the LVS check with no issue. If you have hooked everything up correctly and run LVS, you should get a passed check that looks similar to the figure below.

```
wbs_stb_i |wbs_stb_i
wbs_we_i |wbs_we_i
-----
Cell pin lists are equivalent.
Device classes user_analog_project_wrapper and user_analog_project_wrapper are equivalent.
Final result: Circuits match uniquely.
█
```

Figure 74: Wrapper Passing LVS

With a passed LVS screen, there is one final file that must be generated for precheck. This file is a GDS version of your hooked-up layout. This GDS file will allow for the precheck to check your design and run different checks, such as more robust DRC. To generate this GDS file, open up your layout once more. In the top menu, navigate to “File” and select “Write GDS”. This can be seen in the figure below.

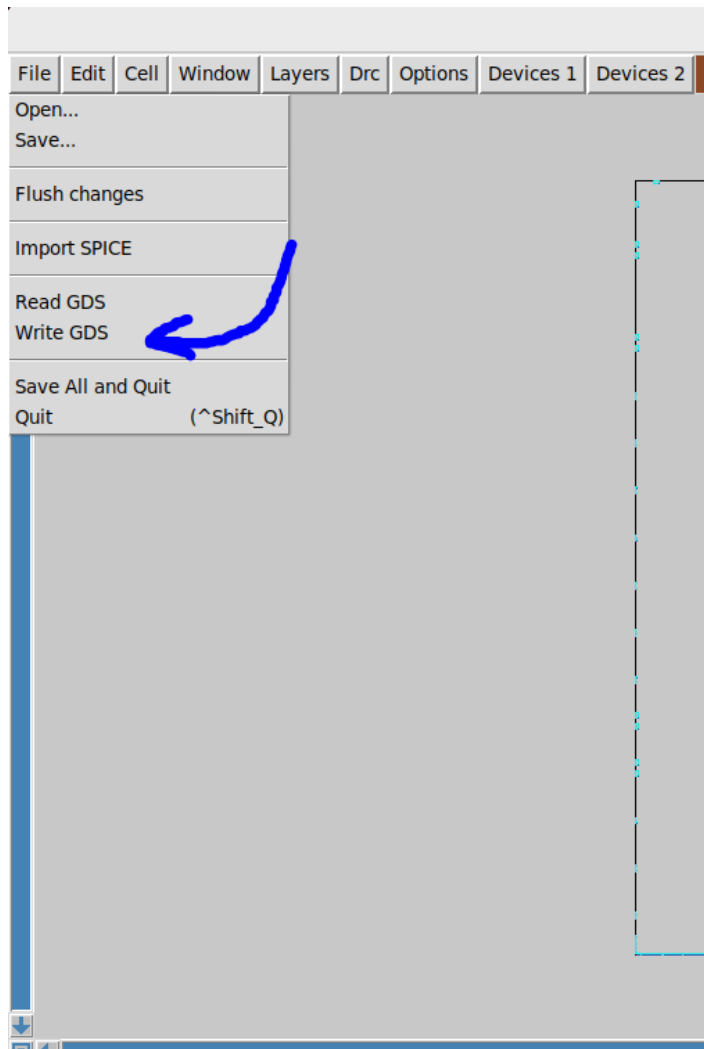


Figure 75: Creation of GDS file

This will open up a window and allow you to save it. Make sure to save it as *“user_analog_project_wrapper”*. Once this is done, you can navigate to the top menu and select *“Read GDS”* under *“File”* to ensure that it generated correctly.

With this done, you have successfully gone through the open-source analog design process flow, as well as generated all the needed files for precheck. It is recommended for precheck that you place all of your schematic, symbols, and layouts (including the generated ones like MOSFETs) into the Caravel Project repository so that precheck can be successfully run.

For steps on how to run precheck, follow the instructions titled *“Pre-check Guide,”* which should be linked on our senior design website.

BONUS: ReRAM Setup

In order to set ReRAM up for simulation, you must do the following. First, open up Xschem with the top-level view of the SkyWater 130 nm cells and tests. On the left-hand side, under the analog primitive section, look for a box called “tb_reram”. Open up this testbench by clicking on the box and pressing “e” on your keyboard. This should open up a view similar to the one seen in the figure below.

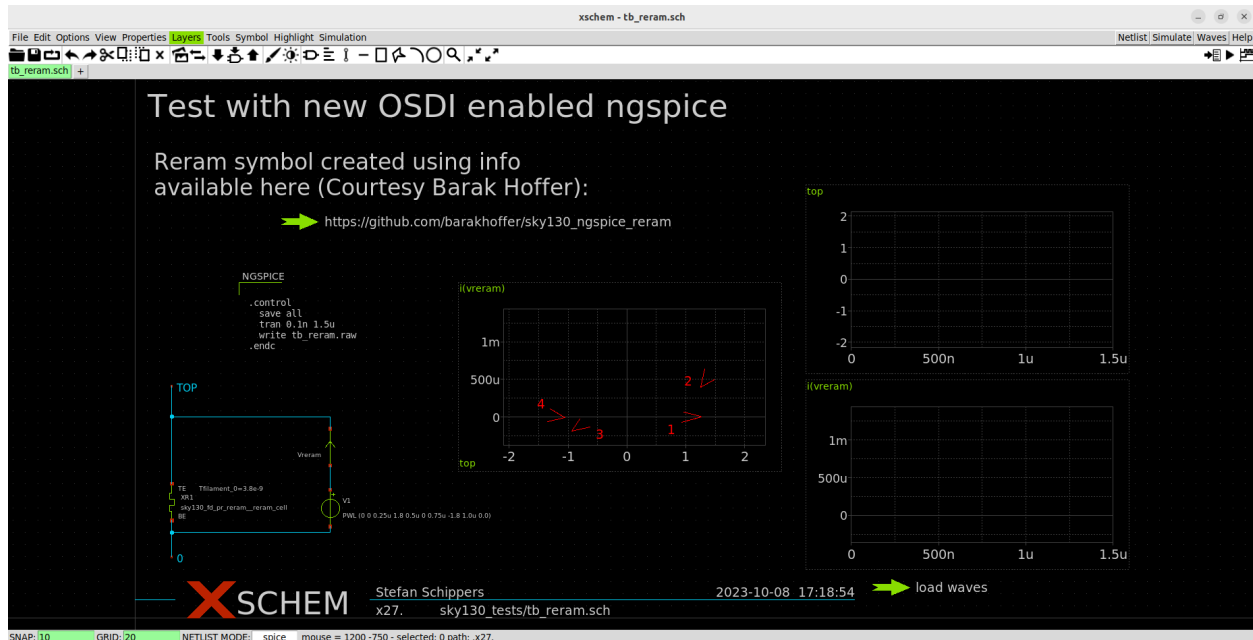


Figure 76: ReRAM Testbench in Open PDKs

You may notice some text at the top of this document talking about a ReRAM symbol and a link to a GitHub page. This GitHub page contains an updated model and extra files for the ReRAM device. These files and an updated model are needed because the default symbol model for ReRAM in the SkyWater PDK is not able to be simulated with Ngspice. With these updated files, the ReRAM device is able to be simulated using Ngspice. If you want to learn more about how all of this works, I recommend joining the ReRAM channel in the Slack that was linked above. It is a good idea to scroll up and read through all of the posts to get a better idea of how ReRAM works in the SkyWater 130 PDK.

In order to begin simulating ReRAM using Ngspice, the files in the GitHub page must be cloned, and an install script must be run. Be sure that you have your global variable “PDK_ROOT” set; otherwise, the install file will not place the files into the correct location. To get these files and the updated model, type the following commands:

- **git clone** https://github.com/barakhoffer/sky130_ngspice_reram
- **cd sky130_ngspice_reram**

- **source install.sh**

This will place the updated ReRAM model and additional files in the correct locations so that an Ngspice simulation can be run. To test if this worked, open up the “*tb_reram*” file in Xschem again. Then simply press “netlist” and “simulate”. With the way the testbench is set up, you can do the simulation as a batch, so just leave everything default. Note that for the simulation to work, you must have configured Ngspice with OSDI support. If you have followed this guide, you will have already done this. Once the simulation is done, you can “*Ctrl + left click*” the arrow that says “load waves”. If everything has been done correctly and the simulation was successful, you should see results like the figure below.

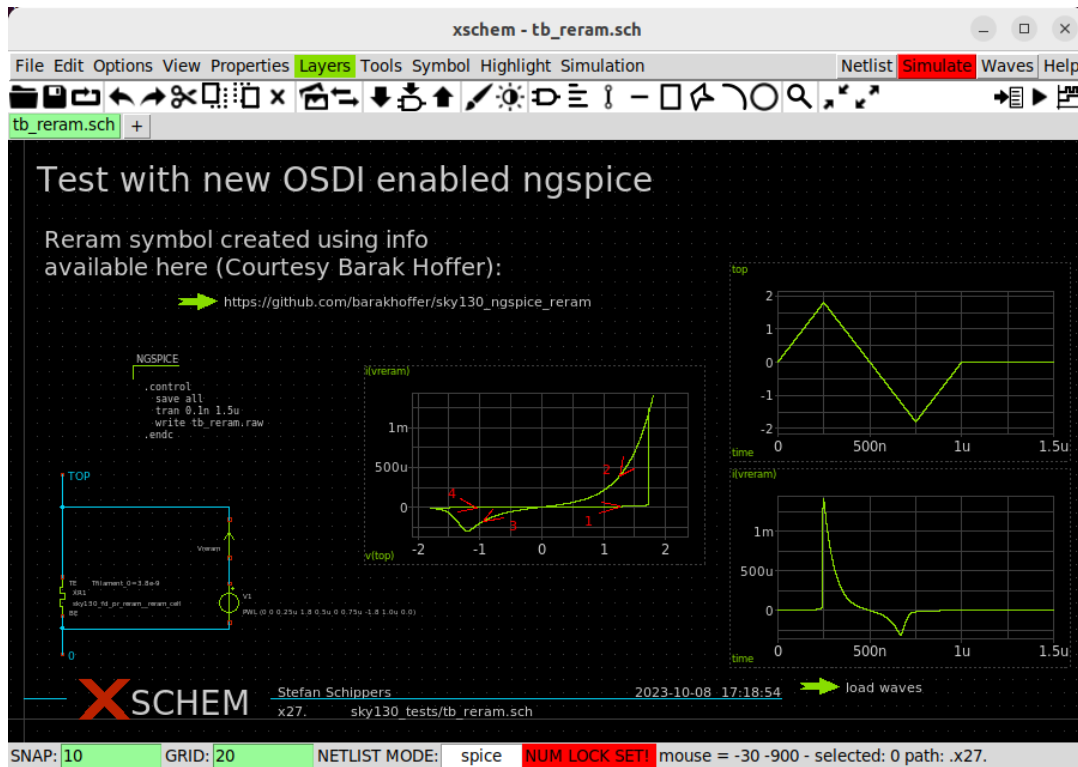


Figure 77: ReRAM Simulation in Xschem using Ngspice

If you see results similar to this, then you have successfully installed the updated model plus additional files and can readily simulate the ReRAM device using Ngspice!