0. Pre-requesit:

As will be seen, the vloga compiler compiles the vloga file into a .osdi file. In order for ngspice to be able to read this, it must be enabled during the install with the below line:

../configure --with-x --enable-xspice --disable-debug --enable-cider --with-readline=yes --enable-openmp **--enable-osdi** --enable-klu

1. Install openVAF:

vlogA files must be compiled before they can be run. The opensource vlogA compiler we use is called openVAF. The latest version of this can be installed from:

https://openvaf.semimod.de/download/

Once installed, you need to create the below path in your .bashrc file:

PATH to run openvaf from any terminal
export PATH="/home/slice/Desktop/openvaf:$PATH"

In this example, openVAF was installed to directory /home/slice/Desktop/openvaf. This line creates a path to that directory from any terminal window. The reason we need this is that to compile the vlogA files you need to type openvaf <name>.va. That openvaf command then calls openvaf installed at the given directory. Without the export path option you would need to always copy your vlogA file to the directory where openVAF was installed and compile from there.

2. Create a vlogA file:

A basic 0th-order OTA model (i.e. a vcvs!) is created below:

****

```
////////////////////////////////////////////////////////////////
// Netlist generated by : Diarmuid Collins
// Date : Mon Feb 05
////////////////////////////////////////////////////////////////

// Library Name : xxx
// Cell Name : OTA_vcvs
// View Name : veriloga
////////////////////////////////////////////////////////////////

`include "constants.vams"
`include "disciplines.vams"

module OTA_vcvs (vdda, vout, vinn, vinp, ibias_1u, vssa);

input vdda;
input ibias_1u;
input vinn;
input vinp;
```

```
input vssa;
output vout;

electrical vdda, vout, vinn, vinp, ibias_1u, vssa;

// Parameters
parameter real Aol_0 = 80; // Open loop gain in dB
parameter real GBW = 10e6; // Gain Bandwidth in Hz

analog begin

        V(vout,vssa) <+  pow(10,Aol_0/20)*V(vinp,vinn);
      $display("OTA_vcvs: vinn=%g voutp=%g --> vout=%g", V(vinn), V(vinp), V(vout));

     end

endmodule
```

****

Save this to some directory as OTA_vcvs.va as per below:

2. Compile the vlogA file:

In the directory you saved the .va file in step 2, run the following command:

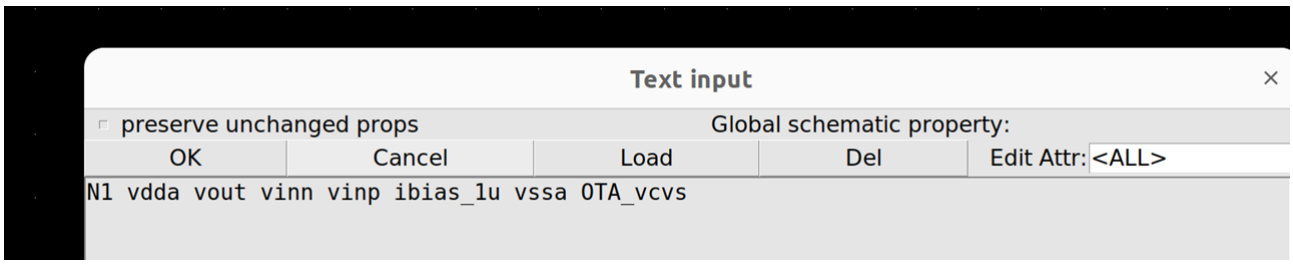
openvaf OTA_vcvs.va

This creates a .osdi file which ngspice runs as the .vlogA file.

Any syntax errors will be reported in this stage. Only when all errors have been resolved will a .osdi file be created.

3. Load the model file into ngspice through xschem:

In xschem, create a symbol matching the port names of the .va file. Inside that symbol add the below lines to create the subckt:



These lines follow the below syntax:

```
N<instance name> <nodes>* <model name> <instance parameters>*
```

It is important to start the instance name with “N”. After that as you can see you simply add the port names and finally the model name (OTA_vcvs).

Next you need to load the model into your netlist. To do this, include the following in your stim file:

```
15
16 .model OTA_vcvs OTA_vcvs
17 * vlogA instantiation
18
```

The above follows the below syntax:

```
.model <model name> <Verilog-A module name> <model parameters>*
```

I have called my model name = OTA_vcvs, which is the same as my vlogA module name. This is why OTA_vcvs appears twice.

Finally, inside the .control statement of your stim file, load in the following line to load in the .osdi file

pre_osdi OTA_vcvs.osdi

When you netlist now you should see the correct subckt instantiation as per below:

```
178
179 * expanding   symbol:  OTA_ideal_order0_vloga.sym # of pins=6
180 ** sym_path: /home/slice/xschem/tb_OTA_1stage/Stefans_eg/ota_vloga/OTA.sym
181 ** sch_path: /home/slice/xschem/tb_OTA_1stage/Stefans_eg/ota_vloga/OTA_ideal_order0_vloga.sch
182 .subckt OTA_ideal_order0_vloga vdda vout vinn vinp ibias_1u vssa
183 *.ipin vssa
184 *.ipin vinp
185 *.ipin ibias_1u
186 *.ipin vinn
187 *.ipin vdda
188 R1 ibias_1u vssa 1MEG m=1
189 **** begin user architecture code
190 N1 vdda vout vinn vinp ibias_1u vssa OTA_vcvs
191 **** end user architecture code
192 .ends
193
```

4. Simulate the model:

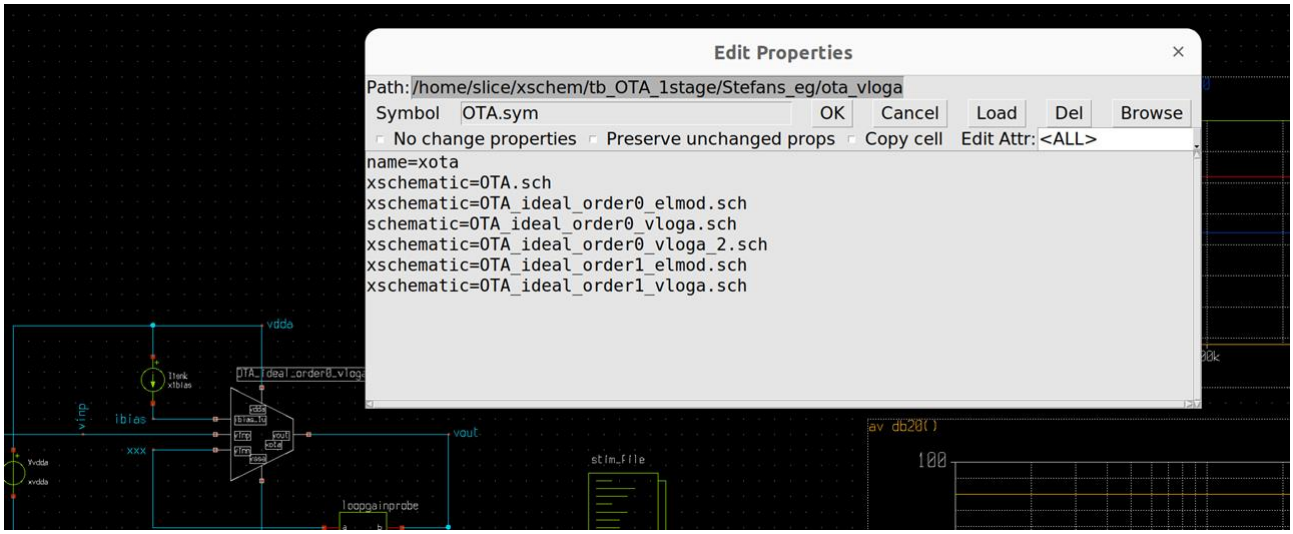
In xschem, run the sim and analyse the data using its waveviewer.

5. Example:

A good example can be found at the below link:

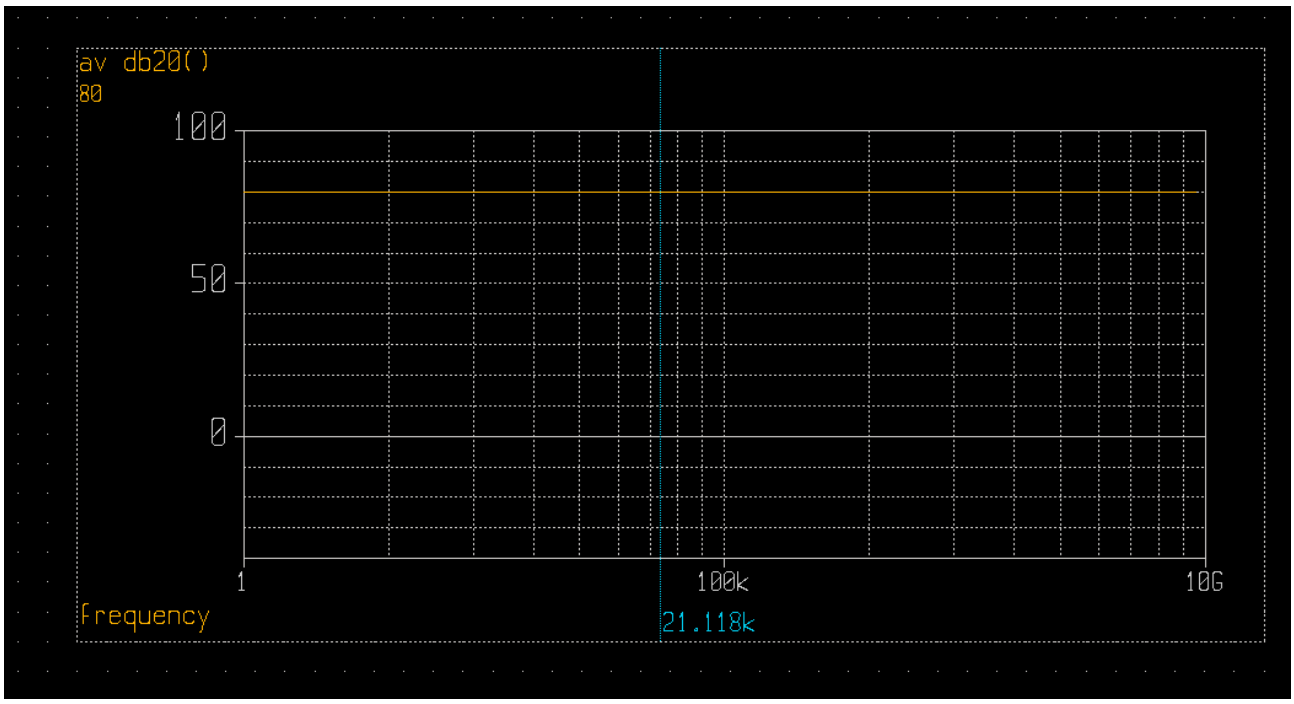
~/xschem/tb_OTA_1stage/Stefans_eg/ota_vloga

This example simulates an OTA. However, I have created multiple views for this OTA as per below:

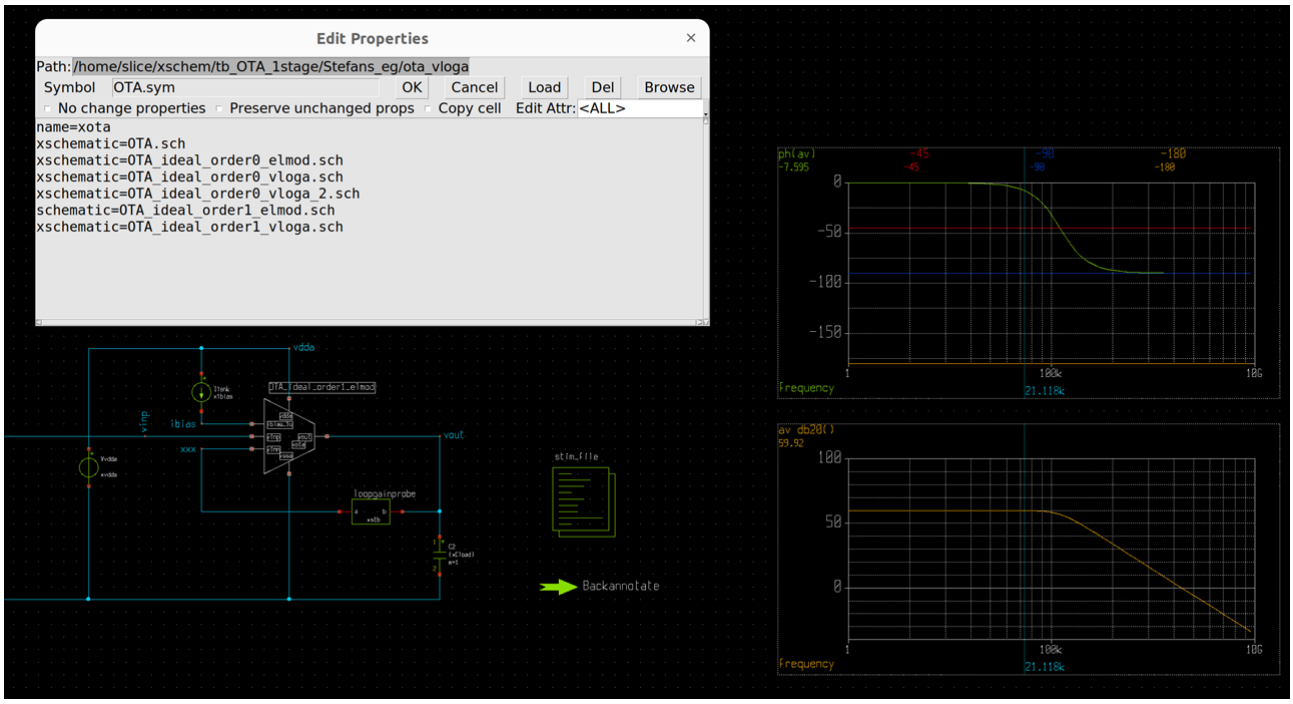


You can select which view to use by simply hitting q on the symbol and removing the “x” before the view name you want to select. Note: This is how I do it which I think is a pretty efficient way in the absence of a config view editor.

As the vcvs only gives 80dB gain when you view the magnitude plot, you will see the below:



This is in contrast to running the order1 element model where a single pole response is modelled as shown below.



This example can also be found on the below public repository in github:

https://github.com/SLICESemiconductor/OpenSourceTool_Examples/tree/main/Running_vloga_in_ngspice

6. Conclusion:

This shows the basic setup of creating a veriloga file, compiling it (openVAF) and converting to a form which ngspice will understand before re-running it inside there (through xschem). We can

build from this example to make more complex ones but they should all follow this general flow (e.g. a proper order 1 OTA model, bin2therm converters, etc.)