Xschem slides [PDF version]

[Video] Xschem FSiC2022 presentation

All in one pdf documentation (repo.hu, github, sourceforge version with autoconfig)

INDEX

TUTORIALS

WHAT IS XSCHEM

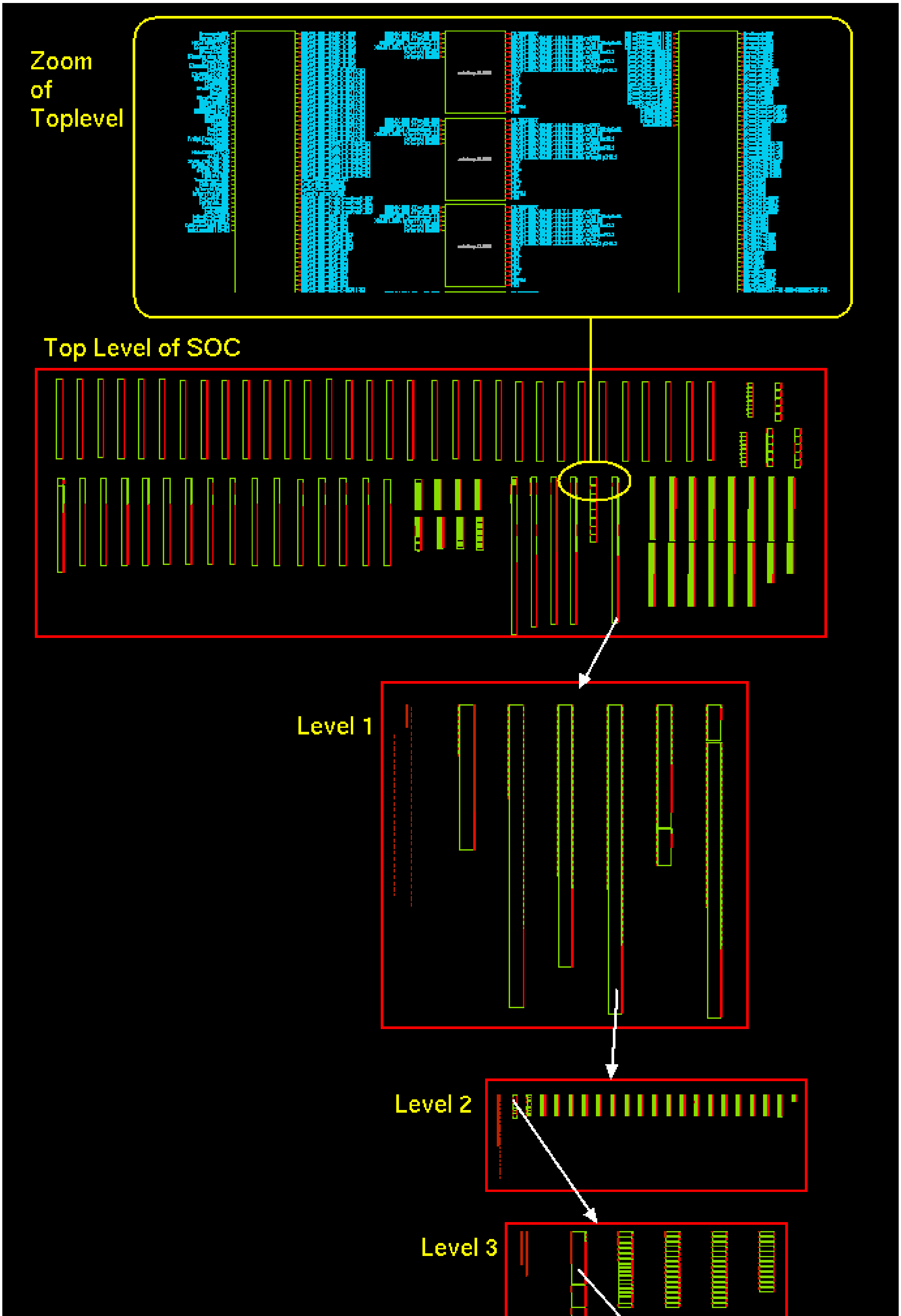
Electronic systems today tend to be generally very complex and a lot of work has to be done from circuit conception to the validation of the final product. One of the milestones of this process is the creation of the circuit schematic of the electronic system.

The circuit diagram has to be drawn using an interactive computer program called *schematic editor* , this is usually a very first step in the design cycle of the product. Once the schematic has been drawn on the computer, the circuit connectivity and device list *(netlist)* can be generated and sent to a circuit simulator (spice, hspice, eldo, just to mention some) for performing circuit simulation.

So, as you probably guessed, **XSCHEM** is a schematic capture program that allows to interactively enter an electronic circuit using a graphical and easy to use interface. When the schematic has been created a circuit netlist can be generated for simulation. Currently XSCHEM supports four netlist formats:

1. SPICE netlist
2. VHDL netlist
3. VERILOG netlist
4. tEDAx netlist for Printed board editing software like pcb-rnd.

XSCHEM was initially created for VLSI design, not for printed circuit board schematics (PCB), however the recently added tEDAx netlist format is used to export XSCHEM schematics to pcb-rnd or other tEDAx-aware PCB editors. The roadmap for XSCHEM development will focus more in the future to build a tight integration with pcb-rnd printed board editor, joining the CoralEDA ecosystem philosophy.
XSCHEM initial design goal was to handle Integrated Circuit (IC) design and generate netlists for Very Large Scale digital, analog or mixed mode simulations. While the user interface looks very simple, the netlisting and rendering engine in XSCHEM are designed from the ground-up to handle in the most efficient way very large designs. Also the user interaction has no bells and whistles but is the result of doing actual work on big projects in the most efficient way. This is why for example most of the work is done with bind keys, instead of using context menus or elaborate graphical actions, simply these things will slow your work if most of your schematics have 5-8 levels of hierarchy and 1000K+ transistors.
Here under a picture of a VLSI SOC (System On Chip) imported in XSCHEM. As you can see the ability of XSCHEM is to handle really big designs. This has been the primary goal during the whole development of the program. The sample design showed has more than 10 levels of hierarchy and really big schematics. For each hierarchy level one component is expanded until the leaf of the tree is reached. :-)


Zoom
of
Toplevel
Top Level of SOC
Level 1
Level 2
Level 3

It is also worth to point out that XSCHEM has nothing to do with GSCHEM, the name similarity is just coincidence. GSCHEM is another powerful Schematic Capture program, primarily focused on board level (PCB) system design. See gEDA for more information.

DOWNLOAD XSCHEM

You should download the xschem sources from one of these repositories:

- From repo.hu with subversion:

```
svn checkout svn://repo.hu/xschem/trunk xschem-src
```

- From sourceforge.net with subversion:

```
svn checkout https://svn.code.sf.net/p/xschem/code/ xschem-src
```

- From github.com with git:

```
git clone https://github.com/StefanSchippers/xschem.git xschem-src
```

Attention

If you have a binary xschem package installed on the system you should remove it. Packaged xschem versions are too old and they conflict with the one you are building from source. To remove: (this command is for Debian/Ubuntu, use similar commands for other distributions)
`sudo apt-get purge xschem`

INSTALL XSCHEM

in order to install the program run the following command:

```
user:~$ cd xschem-src; ./configure
```

This will make all the necessary checks for required libraries and system tools.

for Debian and Ubuntu systems these are the packages you should check to be installed. Tck/Tk versions may vary on different systems, 8.4, 8.5, 8.6 versions are all good.

```
  LIBRARIES                               DEVELOPMENT FILES
-----------------------------------------------------------
  libx11-6                                libx11-dev
  libxrender1                             libxrender-dev
  libxcb1                                libx11-xcb-dev
  libcairo2                              libcairo2-dev
  tcl8.6                                 tcl8.6-dev
  tk8.6                                  tk8.6-dev
  flex                                   bison
  libxpm4                                libxpm-dev
  libjpeg62-turbo or libjpeg             libjpeg-dev

# terminal program and editor used by default by xschem:
# alternative programs can be specified in xschemrc by
# setting tcl variables 'terminal' and 'editor', respectively.
  xterm vim-gtk3
# tools that should be available on all systems by default:
  gawk or mawk
# Suggested (not mandatory for using xschem) packages:
  tcl-tclreadline
```

If configure ends with no errors we are ready to compile:

```
user:~$ make
```

If we want to install xschem and its required files (execute as root if you plan to do a system-wide installation, for example in /usr/local):

```
user:~$ sudo make install
```

This will install all the runtime needed files into the locations previously configured (can be found in Makefile.conf). To change the default installation prefix (/usr/local), please replace the configure step shown above with:

```
 ./configure --prefix=new/prefix/path
```

For testing purposes **`xschem`** can be run and invoked from the build directory **`xschem-<version>/src/`** without installation.

```
user:~$ cd xschem-2.7.0/src && ./xschem
```

When xschem is running, type **`puts $XSCHEM_LIBRARY_PATH`** in the xschem tcl prompt to know the library search path.
Type **`puts $XSCHEM_SHAREDIR`** to see the installation path.

Sample user design libraries are provided and installed systemwide under **`${XSCHEM_SHAREDIR/xschem_library/`**. The XSCHEM_START_WINDOW specifies a schematic to preload at startup, to avoid absolute paths use a path that is relative to one of the **`XSCHEM_LIBRARY_PATH`** directories. XSCHEM will figure out the actual location. You may comment the definition if you don't want any schematic on startup.

If you need to override system settings, create a **`~/.xschem/xschemrc`**. The easiest way is to copy the system installed version from ${prefix}/share/xschem/xschemrc and then make the necessary changes.

```
user:$ mkdir ~/.xschem
```

```
user:$ cp <install root>/share/xschem/xschemrc ~/.xschem/xschemrc
```

When xschem is run the first time it will do the above operations, create a **`${HOME}/.xschem/`** directory and place a **`xschemrc`** into it.

- Build xschem with a custom tcl-tk installation

If you need to build xschem against a tcl-tk installation located in a non-standard place you must provide additional options to the .configure script.
Suppose there is a tcl-tk version 8.4 installation in **`/home/schippes/x/tcltk`** then the following commands must be given before runninkg make and make install:

```
export LD_LIBRARY_PATH=/home/schippes/x/tcltk/lib

./configure \
--prefix=/home/schippes \
/arg/tcl-version=8.4 \
/arg/tk-version=8.4 \
--prefix/libs/script/tcl=/home/schippes/x/tcltk \
--prefix/libs/script/tk=/home/schippes/x/tcltk \
--debug
```

This is the command I run to build and test xschem with tcl-tk 8.4 which was released 20 years ago.

-Technical information - Detailed XSCHEM startup sequence

Information here under is not meant to be executed by the user

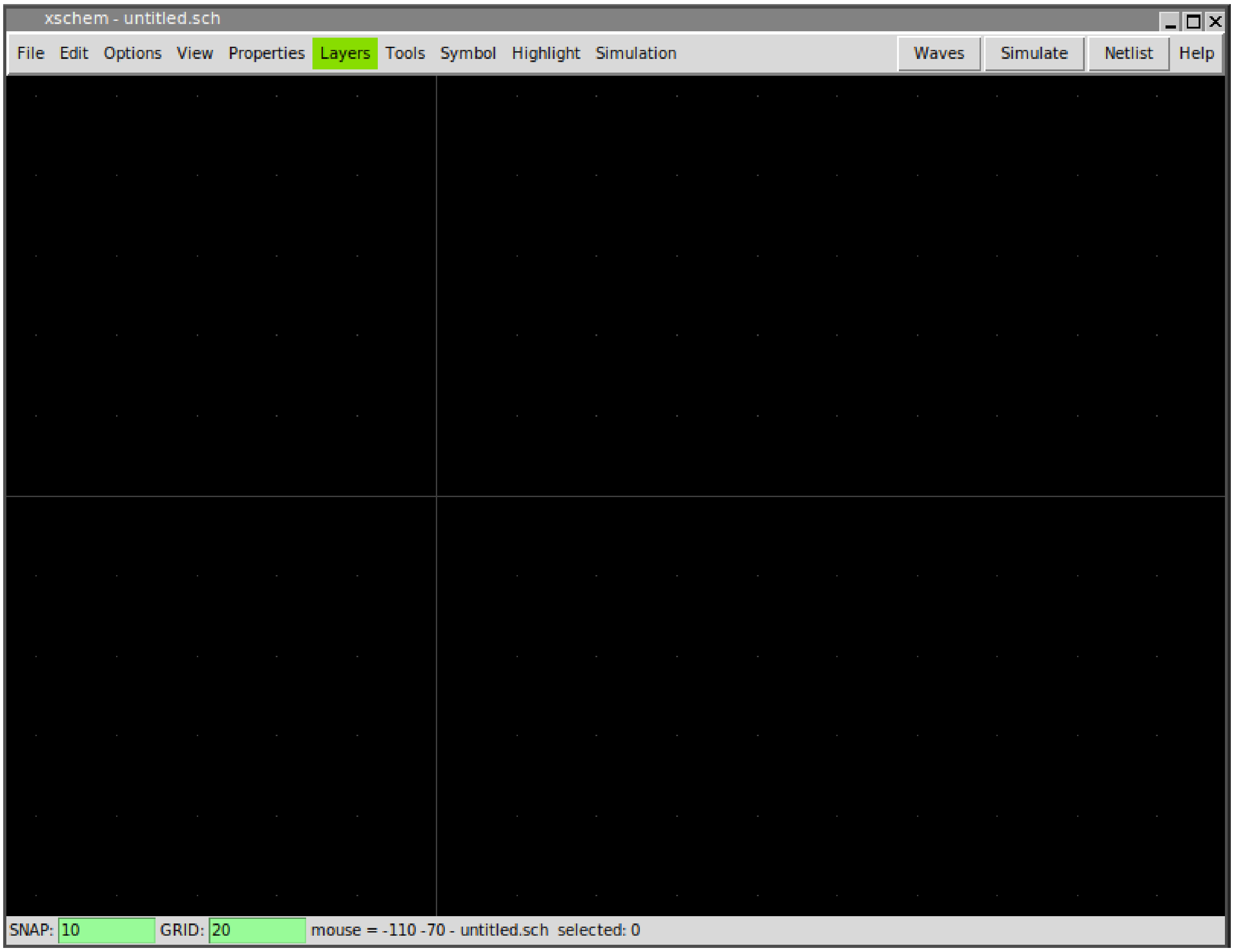
1. Source system-wide xschemrc if existing: **`XSCHEM_SHAREDIR/xschemrc`**. This file is by default all commented so it does nothing. See below how **`XSCHEM_SHAREDIR`** path is generated.
2. If **`--rcfile=<rcfile>`** is given then source the specified rcfile. Do not load any other rcfile.
3. If **`../src/xchem.tcl`** with respect to current dir is existing and **`../xschem_library`** is also existing then we are starting from a build directory, set **`XSCHEM_SHAREDIR`** to **`<current dir>`** and also set **`XSCHEM_LIBRARY_PATH`** to **`../xschem_library/devices`**.
4. Else use compile-time (generated from configure script) provided **`XSCHEM_SHAREDIR`**. (default is **`/usr/local/share/xschem`**).
5. If in current dir there is a **`xschemrc`** file source it.
6. Else if there is a **`USER_CONF_DIR/xschemrc`** file source it. **`XSCHEM_SHAREDIR`** and **`USER_CONF_DIR`** are preprocessor macros passed at compile time by the configure script. The first one will be overridden only if executing from a build directory, see item 3.
7. If **`XSCHEM_SHAREDIR`** not defined --> error and quit.
8. Start loading user provided schematic file or start with empty window (or filename specified in **`XSCHEM_START_WINDOW`** tcl variable).

RUN XSCHEM

Assuming **`xschem`** is installed in one of the **`${PATH}`** search paths just execute:

```
user:~$ xschem
```

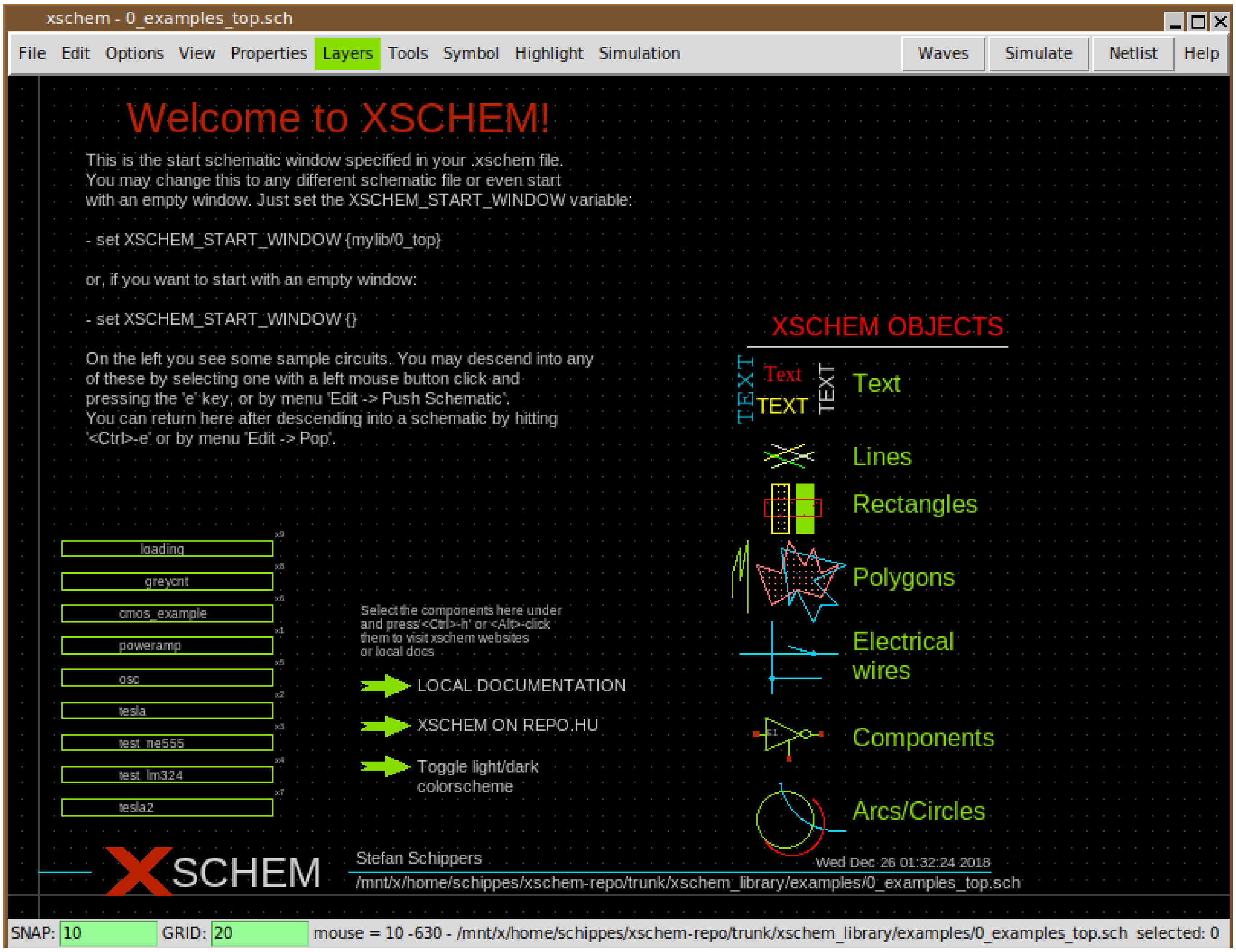
the xschem window should appear. If **`xschem`** is not in the search path then specify its full pathname.



SNAP: 10 GRID: 20 mouse = -110 -70 - untitled.sch selected: 0

if a filename is given that file will be loaded on startup:

```
user:~$ xschem .../xschem_library/examples/0_examples_top.sch
```



Welcome to XSCHEM!

This is the start schematic window specified in your .xschem file.
You may change this to any different schematic file or even start
with an empty window. Just set the XSCHEM_START_WINDOW variable:

- set XSCHEM_START_WINDOW {mylib/0_top}

or, if you want to start with an empty window:

- set XSCHEM_START_WINDOW {}

On the left you see some sample circuits. You may descend into any
of these by selecting one with a left mouse button click and
pressing the 'e' key, or by menu 'Edit -> Push Schematic'.
You can return here after descending into a schematic by hitting
'<Ctrl>-e' or by menu 'Edit -> Pop'.

Select the components here under
and press '<Ctrl>-h' or <Alt>-click
them to visit xschem websites
or local docs

LOCAL DOCUMENTATION

XSCHEM ON REPO.HU

Toggle light/dark
colorscheme

SNAP: 10 GRID: 20 mouse = 10 -630 - /mnt/x/home/schippes/xschem-repo/trunk/xschem_library/examples/0_examples_top.sch selected: 0

XSCHEM COMMAND LINE OPTIONS

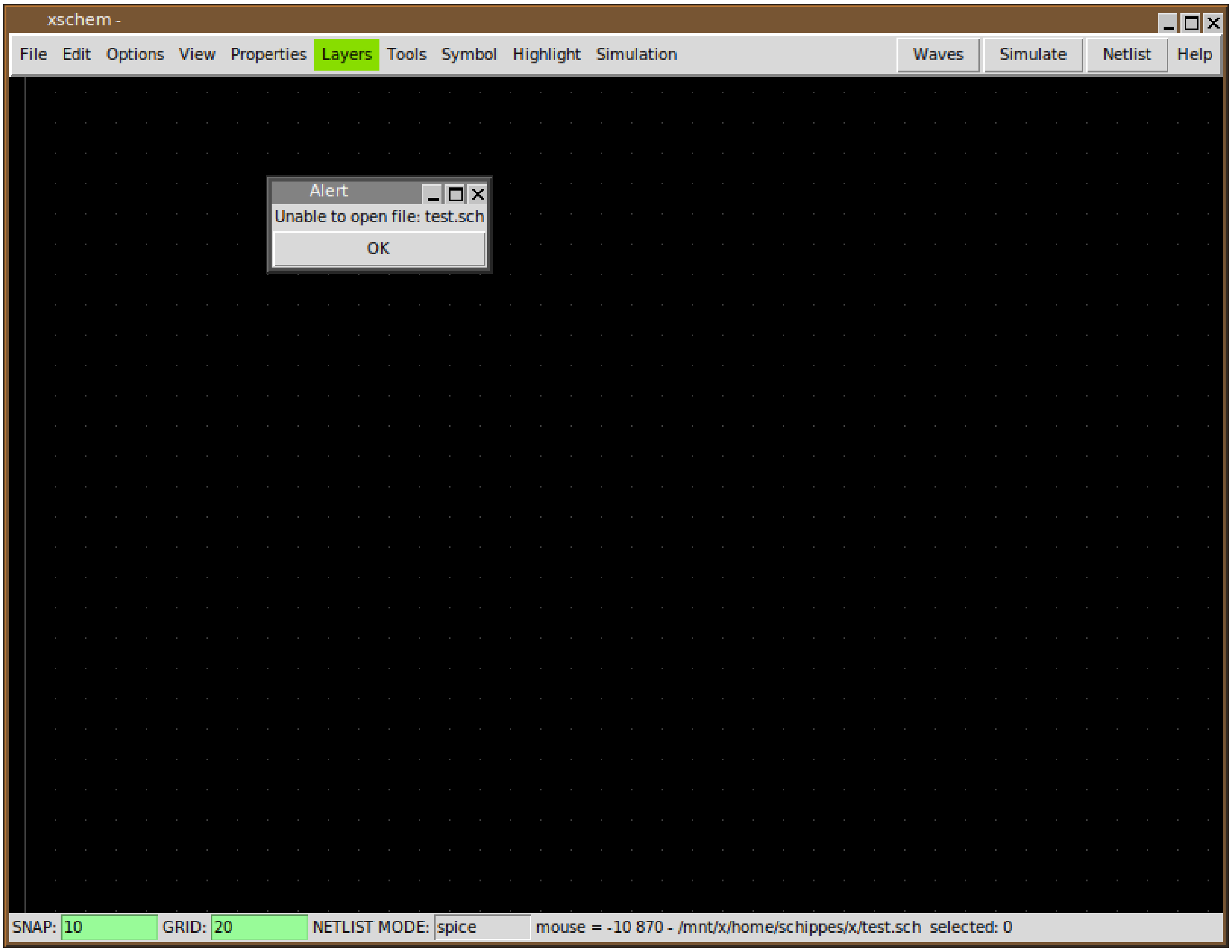
xschem accepts short (-h) or long (--help) options:

```
usage: xschem [options] [schematic | symbol ]
Options:
  -h  --help            Print this help.
  -b  --detach          Detach Xschem from console (no output and no input from console)
  -n  --netlist         Do a netlist of the given schematic cell.
  -v  --version         Print version information and exit.
  -V  --vhdl            Set netlist type to VHDL.
  -S  --simulate        Run a simulation of the current schematic file
                        (spice/Verilog/VHDL, depending on the netlist
                        type chosen).
  -w  --verilog         Set netlist type to Verilog.
  --tcl <tcl_cmd>       Execute specified tcl instructions before any other action,
                        after sourcing xschemrc, this can be used to change xschemrc variables.
  --preinit <tcl_cmd>   Execute specified tcl instructions before any other action,
                        and before loading xschemrc.
  --script <file>       Execute specified tcl file as a command script (perhaps with xschem  comman
  --command <tcl_cmd>   Execute specified tcl commands after completing startup.
  --diff <file>         Show differences with given file.
  --tcp_port <number>   Listen to specified tcp port for client connections. (number >=1024).
  -i  --no_rcload       Do not load any xschemrc file.
  --netlist_path <path>
  -o <path>            Set output path for netlist.
  --netlist_filename <file>
  -N <file>            Set name (only name or full path) of top level netlist file.
  -t  --tedax          Set netlist type to tEDAx.
  -s  --spice          Set netlist type to SPICE.
  -y  --symbol         Set netlist type to SYMBOL (used when drawing symbols)
  -x  --no_x           Don't use X (only command mode).
  -z  --rainbow        Use a rainbow-looking layer color table.
  -W  --waves          Show simulation waveforms.
  -f  --flat_netlist   Set flat netlist (for spice format only).
  -r  --no_readline    Start without the tclreadline package, this is necessary
      --pipe           if stdin and stdout are to be redirected. This also prevents xschem
                       from closing stdin / stdout / stderr even if invoked from pipes.
  -c  --color_ps       Set color postscript.
  --plotfile <file>    Use <file> as output for plot (png, svg, ps).
  --rcfile <file>      Use <file> as a rc file for startup instead of the
                       default xschemrc.
  -p  --postscript
      --pdf            Export pdf schematic.
      --png            Export png schematic.
      --svg            Export svg schematic.
  -q  --quit           Quit after doing things (no interactive mode).
  -l <file>
  --log <file>         Set a log file.
  -d <n>
  --debug <n>          Set debug level:  1,  2,  3,..  C program debug.
                                        -1, -2, -3...  TCL frontend debug.

xschem: interactive schematic capture program

Example:   xschem counter.sch
the schematic file `counter.sch' will be loaded.
```

CREATING A NEW SCHEMATIC

To create a new schematic run xschem and give a non existent filename:

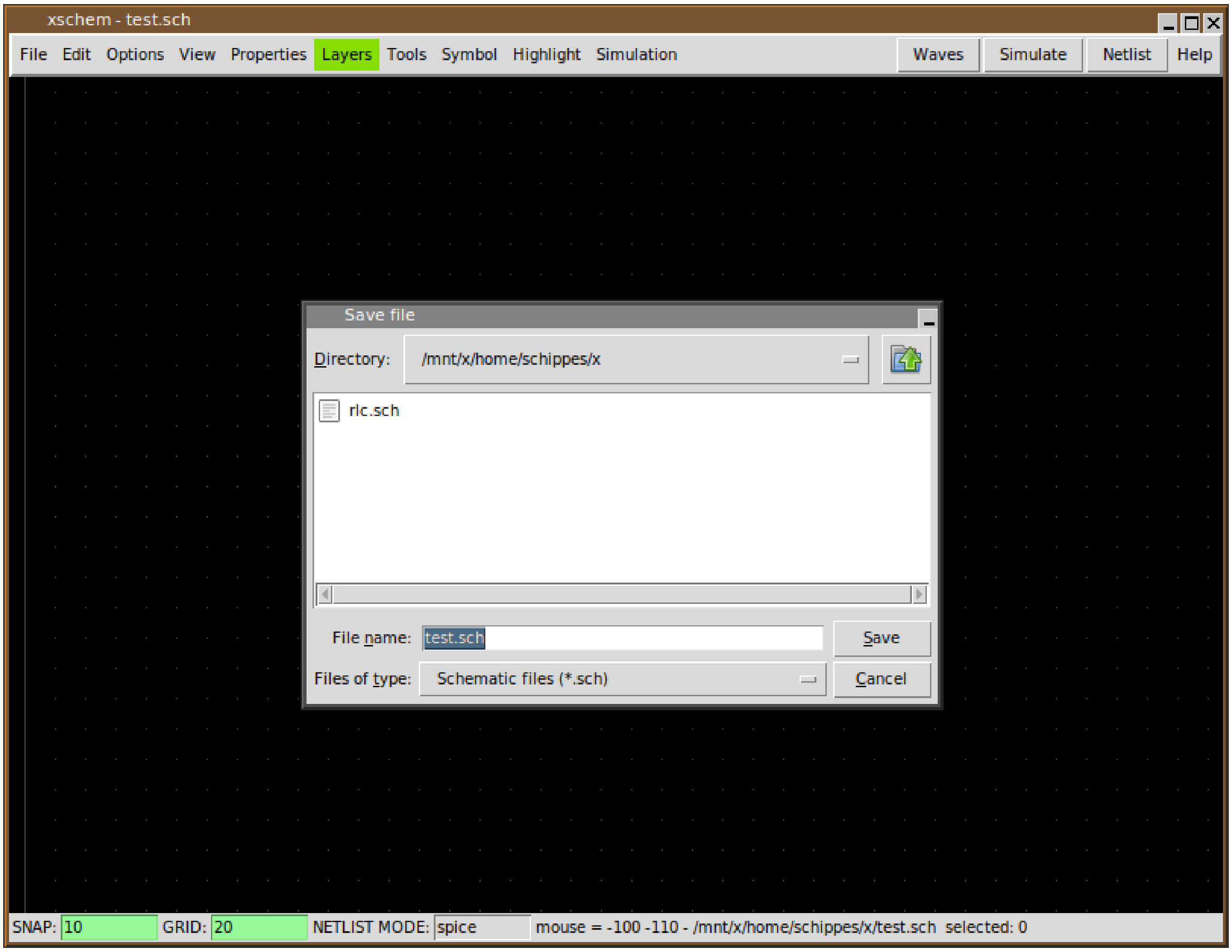
`xschem aaa.sch`


xschem -
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
Alert
OK

Unable to open file: test.sch

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -10 870 - /mnt/x/home/schippes/x/test.sch selected: 0

You can save the schematic by pressing **'<ctrl shift>s'** or by using the menu **File - Save As**:



SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -100 -110 - /mnt/x/home/schippes/x/test.sch selected: 0

If no filename change is needed you can just use **File - Save**. Now a new empty schematic file is created. You can use this **test.sch** for testing while reading the manual. After exiting XSCHEM you can load directly this schematic with the following commands, they are all equivalent.

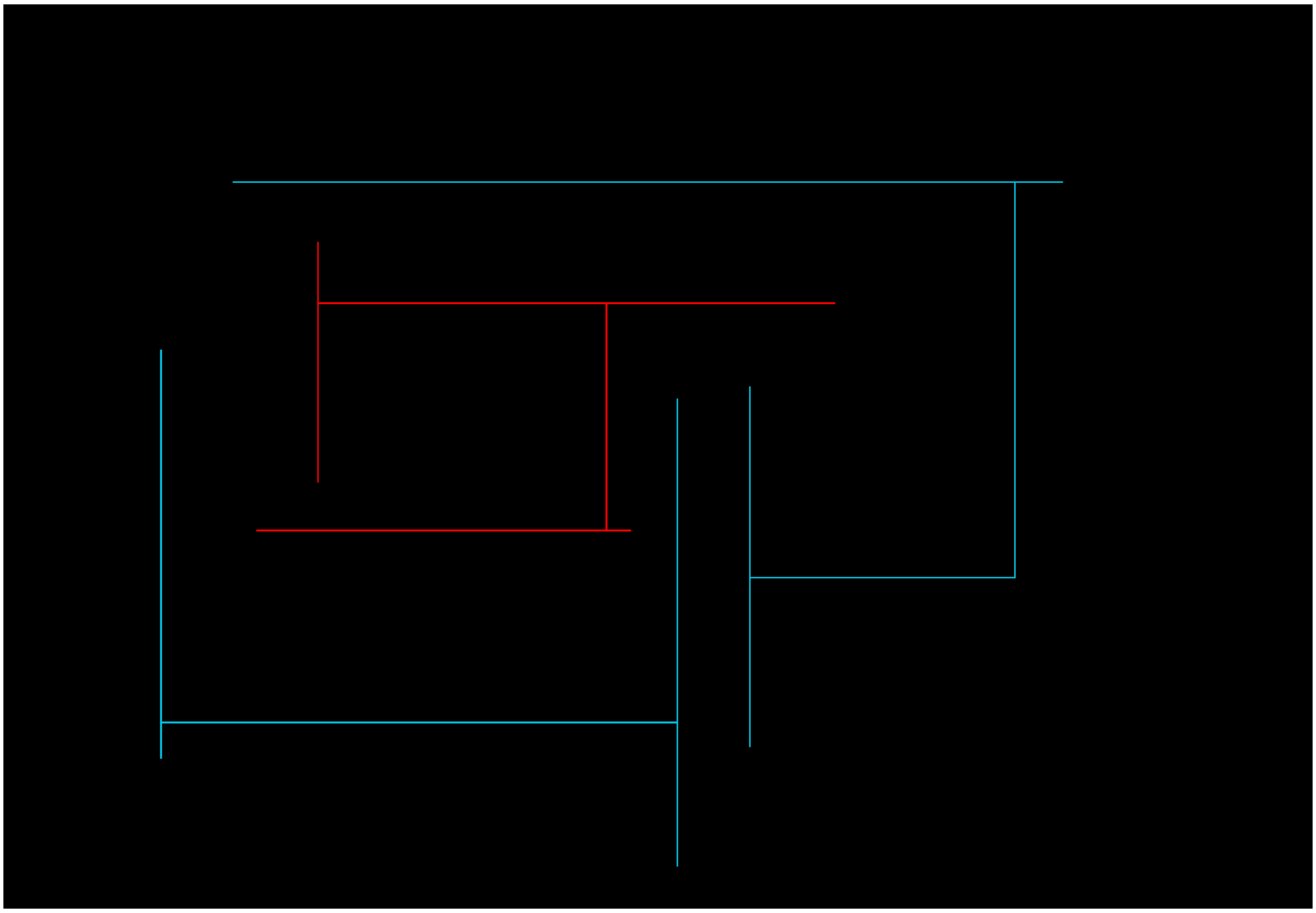
```
xschem /home/schippes/x/test.sch
# or ...
xschem ${HOME}/schippes/x/test
```

you can load **test.sch** when xschem is running by using the load command **'<ctrl>o'** key or by menu **Open** command. Use the file selector dialog to locate the schematic and load it in. When loading a new file XSCHEM asks to save the currently loaded schematic if it has been modified.

XSCHEM ELEMENTS

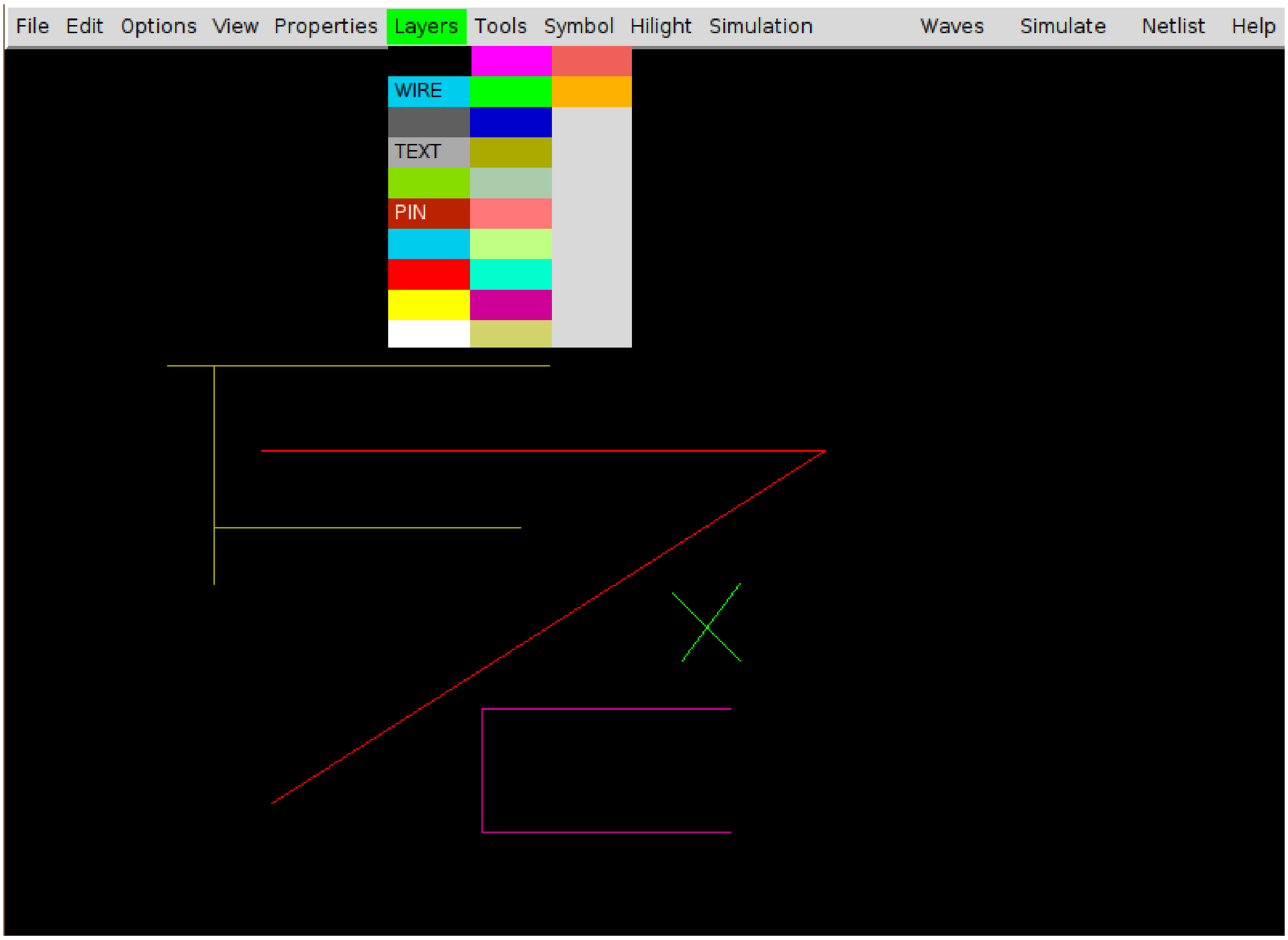
WIRES

Wires in XSCHEM are the equivalent of copper traces in printed circuit boards or electrical conductors. Wires are drawn as lines but the electrical connectivity graph is built by XSCHEM. To draw a wire segment point the mouse somewhere in the drawing window and press the **`'w'`** key. A rubber wire is shown with one end following the mouse. Clicking the left mouse button finishes the placement. The following picture shows a set of connected wires. There are many wire segments but only 3 electrical nodes. XSCHEM recognizes connection of wires and uses this information to build up the circuit connectivity. All wires are drawn on the 'wire' layer. One electrical node in the picture below has been highlighted in red (this is a XSCHEM function we will cover later on).



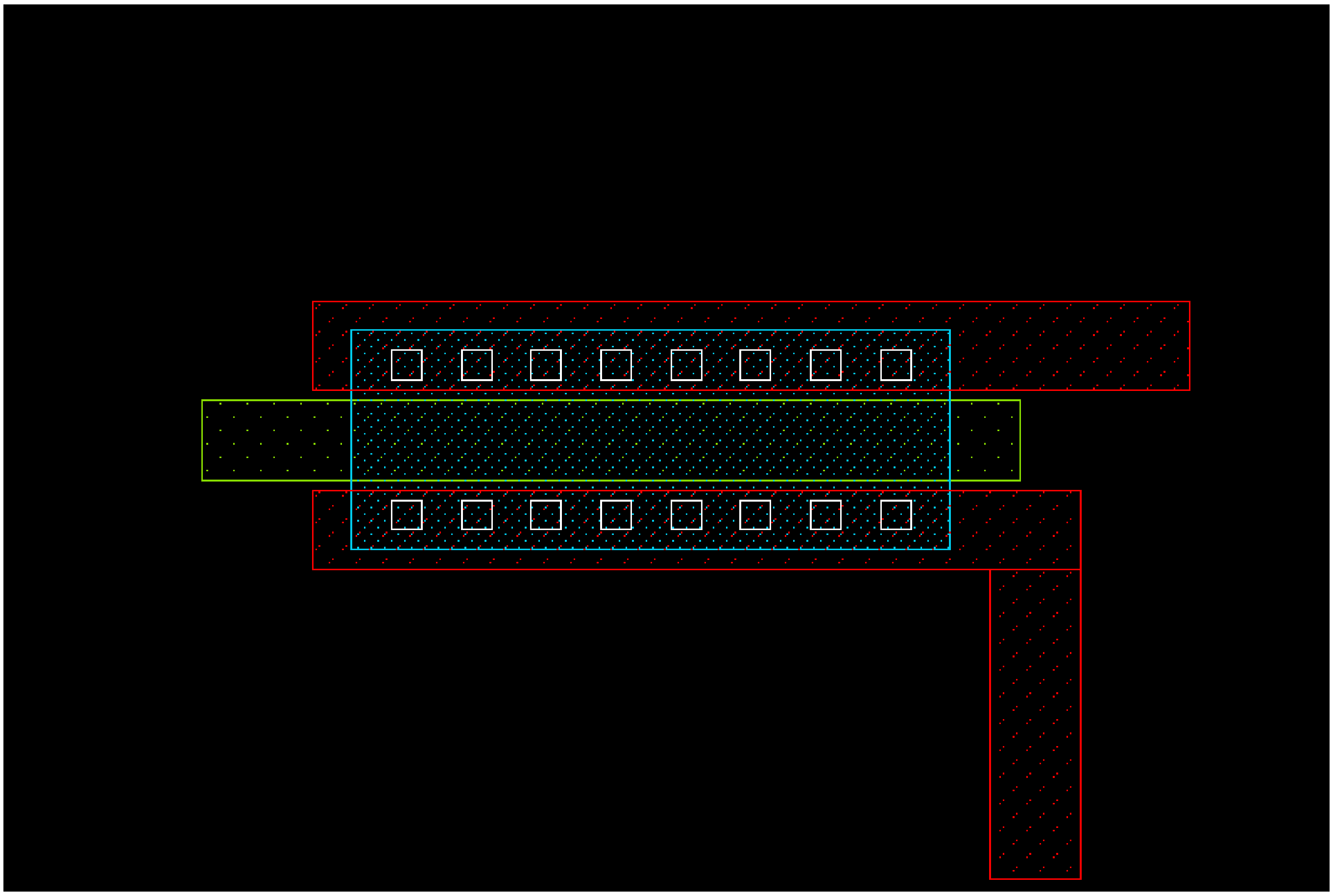
LINES

Lines are just segments that are used for drawing. Lines do not have any electrical meaning, in fact when building the circuit netlist, lines are completely ignored. XSCHEM uses different layers to draw lines. Each layer has its own color, allowing to draw with different colors. Lines are placed like wires, but using the **`'l'`** key. The 'Layers' menu allows to select various different layers (colors) for the line.



RECTANGLES

Rectangles like Lines are drawable on multiple layers, and also do not carry any electrical information. A specific 'PIN' layer is used to make pins that are used to interconnect wires and components. Different fill styles (or no fill) can be defined for each layer. Rectangles are placed with the **'r'** bindkey

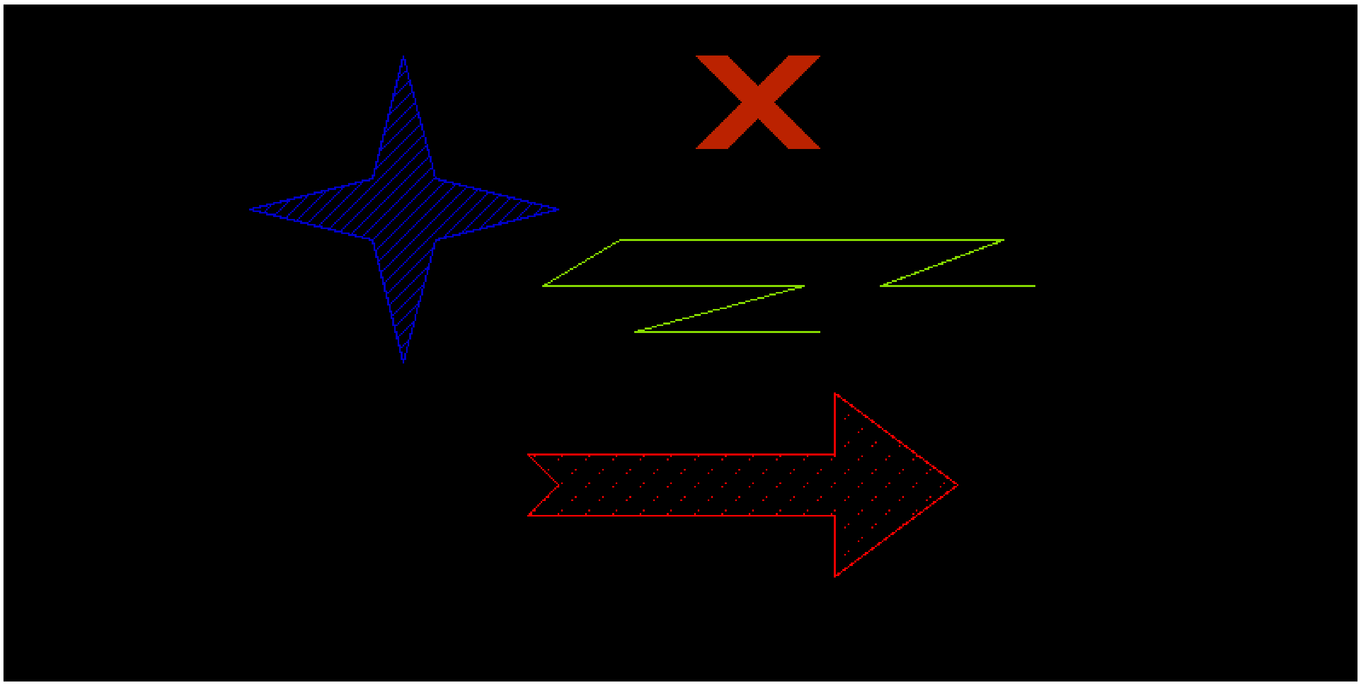


POLYGONS

Polygons are paths that can be drawn on any layer. Placements begins with the **`'p'`** key and continues as long as the user clicks points on the drawing area. Placement ends when:

- the last point is coincident to the first point.
- or by clicking the **`right mouse button`**, for an open polygon.
- or by hitting the **`Return`** key, for a closed polygon (this can be done also by clicking the last point coincident to the first polygon point).

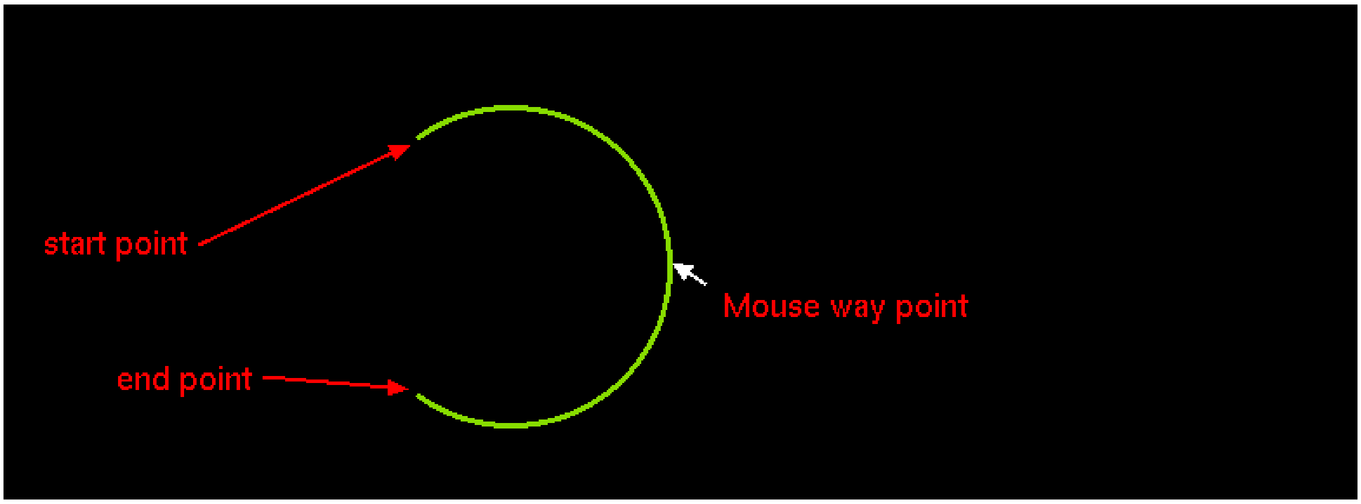
A **`fill=true`** attribute may be given to have the shape filled with the layer fill style.



CIRCLES / ARCS

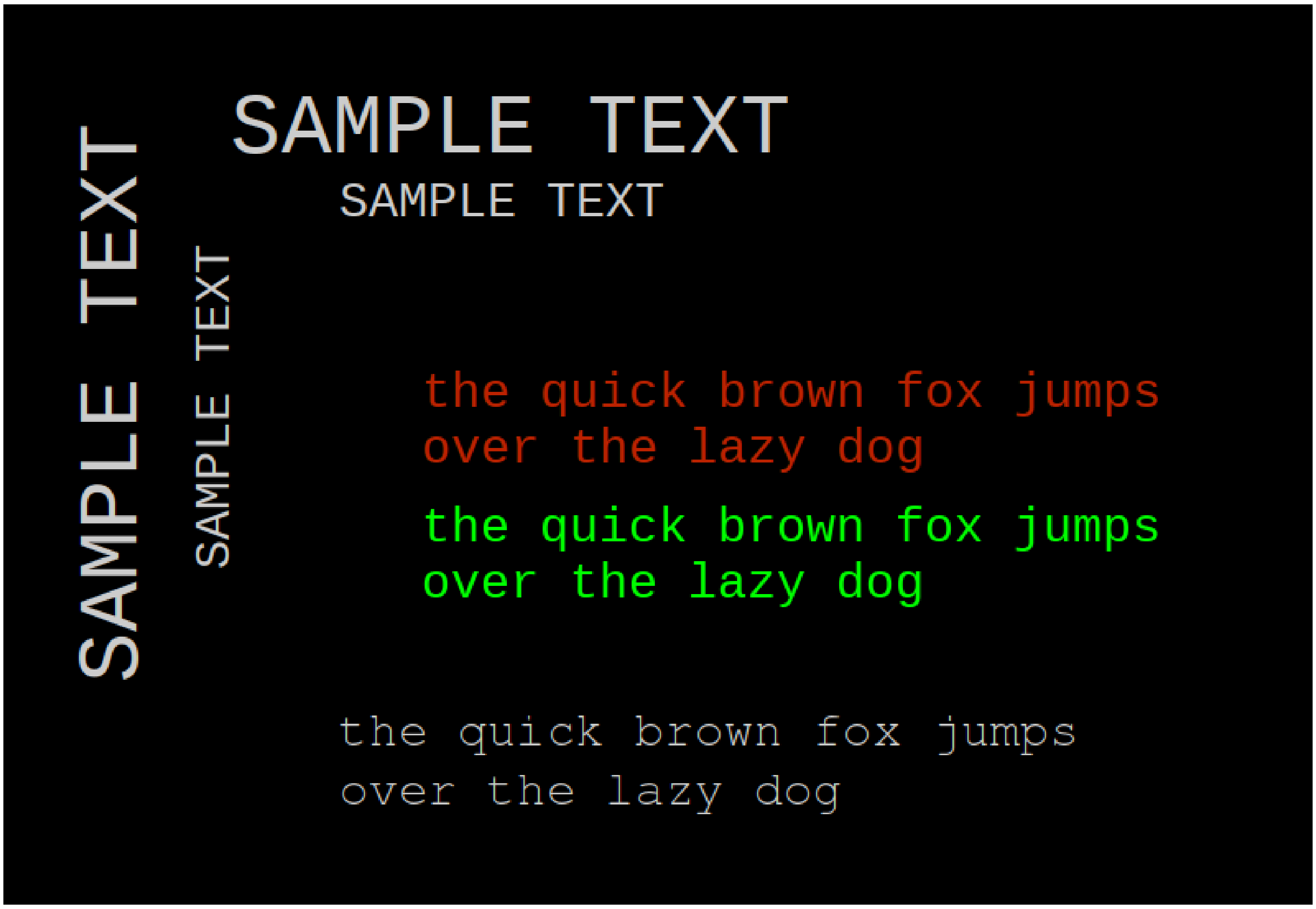
Arcs may be placed by hitting the **`Shift-C`** key. First click the start point, then the end point. Moving the mouse will show the arc passing thru the 2 points and the mouse waypoint. Clicking will place the arc. Arcs may be modified after creation by selecting in stretch mode (**`Ctrl-Button1-drag`**) one of the arc ends or the arc center:
- (end point selected in stretch mode): by starting a move (**`m`**) operation and moving the mouse the arc sweep may be changed.
- (start point selected in stretch mode):by starting a move (**`m`**) operation and moving the mouse the start arc angle may be changed.
- (arch center selected in stretch mode): by starting a move (**`m`**) operation and moving the mouse the arc radius may be changed.

If a circle is needed then use the **`Ctrl-Shift-C`** key combination.
A **`fill=true`** attribute may be given to have the shape filled with the layer fill style.

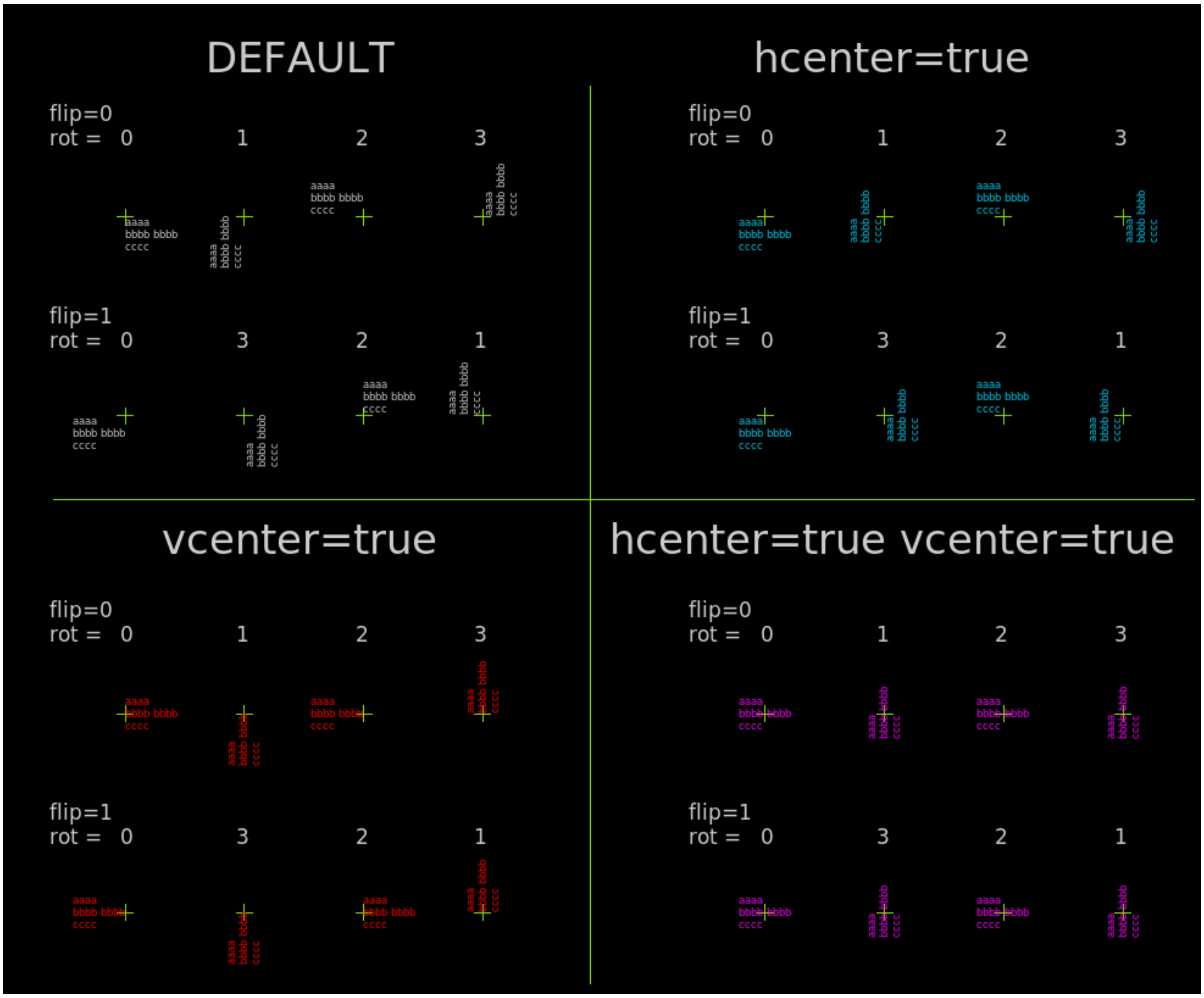


TEXT

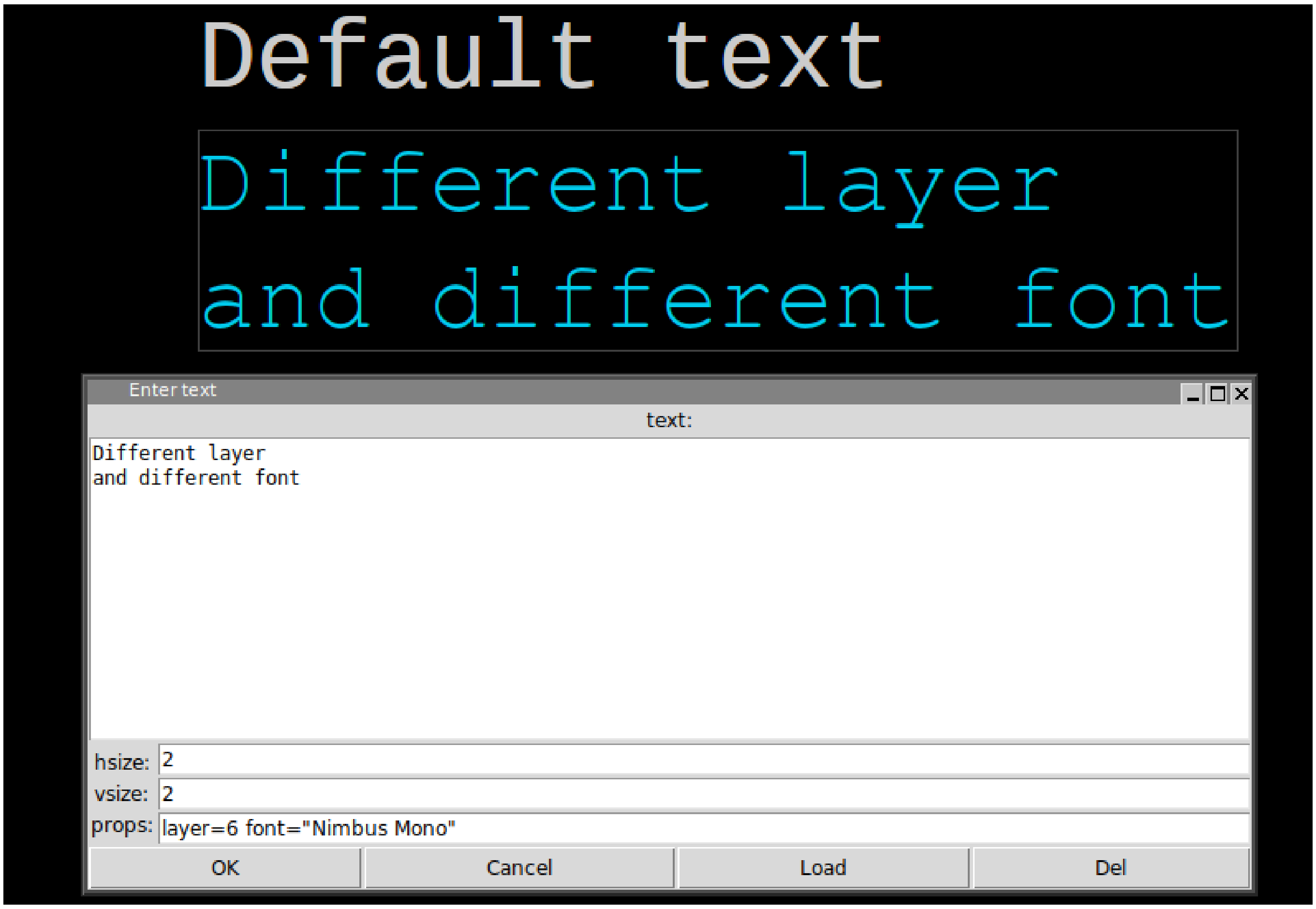
Text can be placed with the `'t'` bindkey. A dialog box appears where the user inputs the text and text size.


SAMPLE TEXT
SAMPLE TEXT
SAMPLE TEXT
SAMPLE TEXT
the quick brown fox jumps over the lazy dog
the quick brown fox jumps over the lazy dog
the quick brown fox jumps over the lazy dog

The **`layer`** property can be used to draw text on a different layer, for example, setting **`layer=6`** will draw on cyan color. A **`font`** property is defined to change the default font. A **`hcenter=true`** attribute may be set to center text in the reading direction, while **`vcenter=true`** centers text in the perpendicular (to reading) direction. the 2 attributes may be set both to get full centered text box.
A **`weight=bold`** attribute may be given for bold text, while a **`slant=italic`** or **`slant=oblique`** may specify italic or slanted text.
A **`hide=true`** will make the specified text invisible unless the **`View->Show hidden texts`** option is enabled. If **`hide=instance`** is given the text will be invisible in placed instances of the symbol, but visible when descending into the symbol.



You will learn in the xschem properties chapter how to set, edit and change object properties.

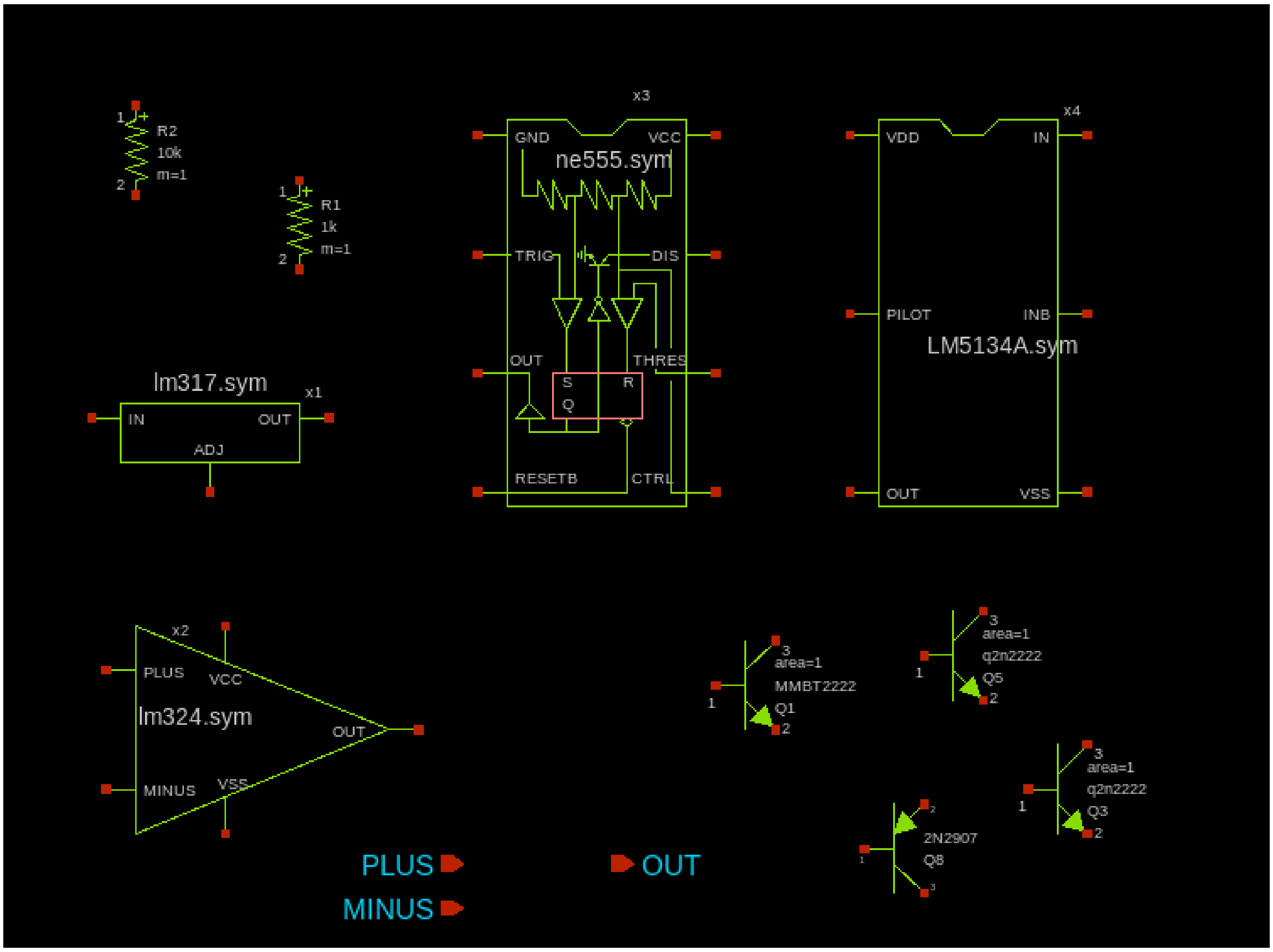


```
Different layer
and different font
```

hsize: 2
vsize: 2
props: layer=6 font="Nimbus Mono"

SYMBOLS

Symbols are graphical elements that represent electrical components. A symbol represents an electronic device, like for example a resistor, a bipolar transistor, an amplifier etc. As you can see graphically symbols are built with lines, rectangles, polygons and texts, the graphical primitives shown before. In the picture below some components are placed in a schematic window. Components are instances of symbols. For example you see three placements of the 'npn' bipolar transistor symbol. Like in C++, where objects are instances of classes, here components are instances of symbols.



Symbols (like schematic drawings) are stored in xschem libraries. For XSCHEM a library is just a directory placed under the XSCHEM_LIBRARY_PATH directory, see the installation slide. A symbol is stored in a .sym file.

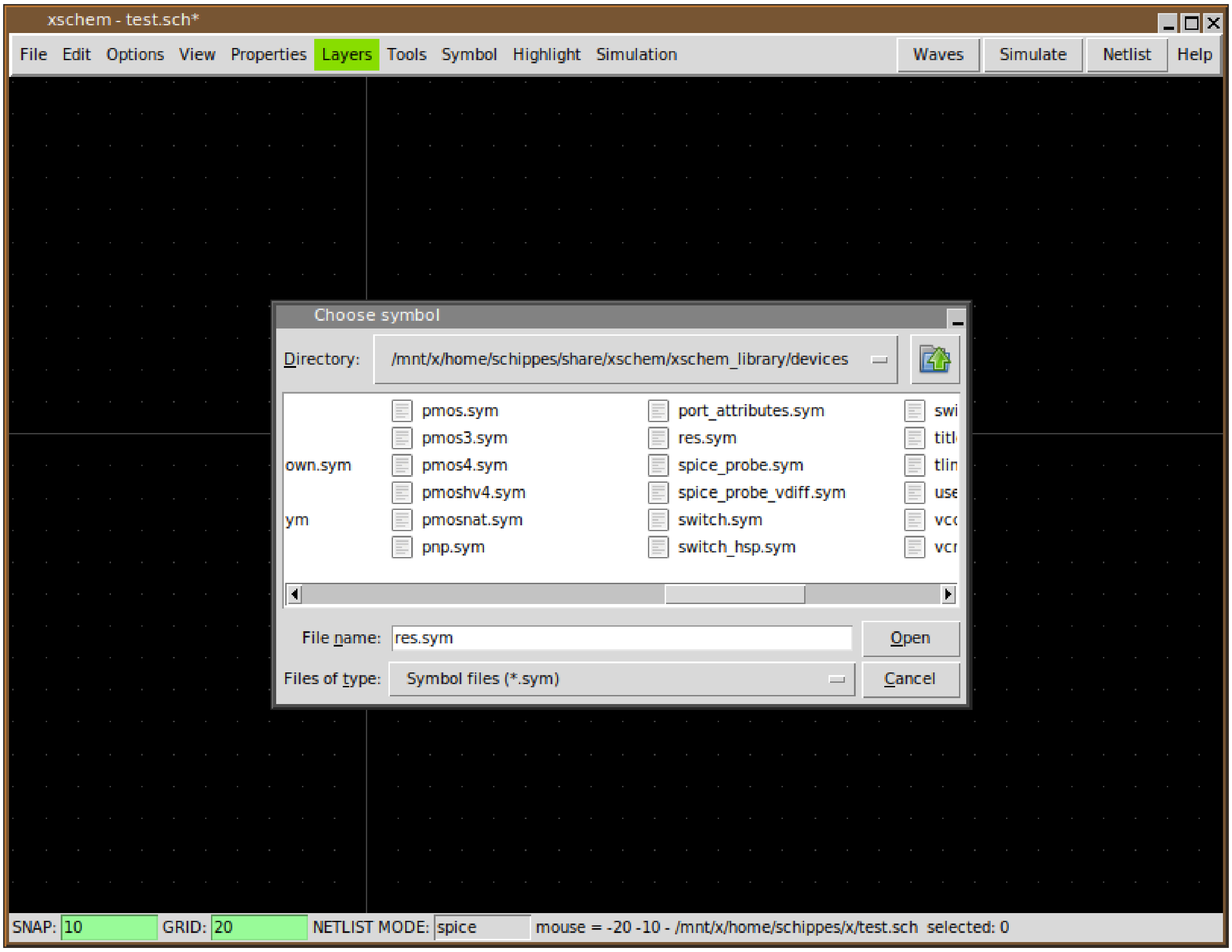
```
user:~$ cd .../share/xschem/xschem_library/
user:xschem_library$ ls
devices
user:xschem_library$ cd devices
user:devices$ ls *.sym
ammeter.sym                generic.sym            noconn.sym              switch_hsp.sym
arch_declarations.sym      gnd.sym                npn.sym                 switch.sym
architecture.sym           ind.sym                opin.sym                title.sym
assign.sym                 iopin.sym              package_not_shown.sym   tline_hsp.sym
attributes.sym             ipin.sym               package.sym             use.sym
bus_connect_not_shown.sym  isource_arith.sym      param_agauss.sym        vccs.sym
bus_connect.sym            isource_pwl.sym        param.sym               vcr.sym
capa.sym                   isource.sym            parax_cap.sym           vcvs.sym
cccs.sym                   k.sym                  pmos3.sym               vdd.sym
ccvs.sym                   lab_pin.sym            pmos4.sym               verilog_delay.sym
connect.sym                lab_wire.sym           pmosnat.sym             verilog_timescale.sym
delay_hsp.sym              launcher.sym           pnp.sym                 vsource_arith.sym
delay_line.sym             netlist_at_end.sym     port_attributes.sym     vsource_pwl.sym
delay.sym                  netlist_not_shown.sym  res.sym                 vsource.sym
diode.sym                  netlist.sym            spice_probe.sym         zener.sym
flash_cell.sym             nmos3.sym              spice_probe_vdiff.sym
generic_pin.sym            nmos4.sym              switch_hsp_pwl.sym
user:devices$ cd ...share/doc/xschem/
user:xschem$ ls
```

```
    examples  pcb
```

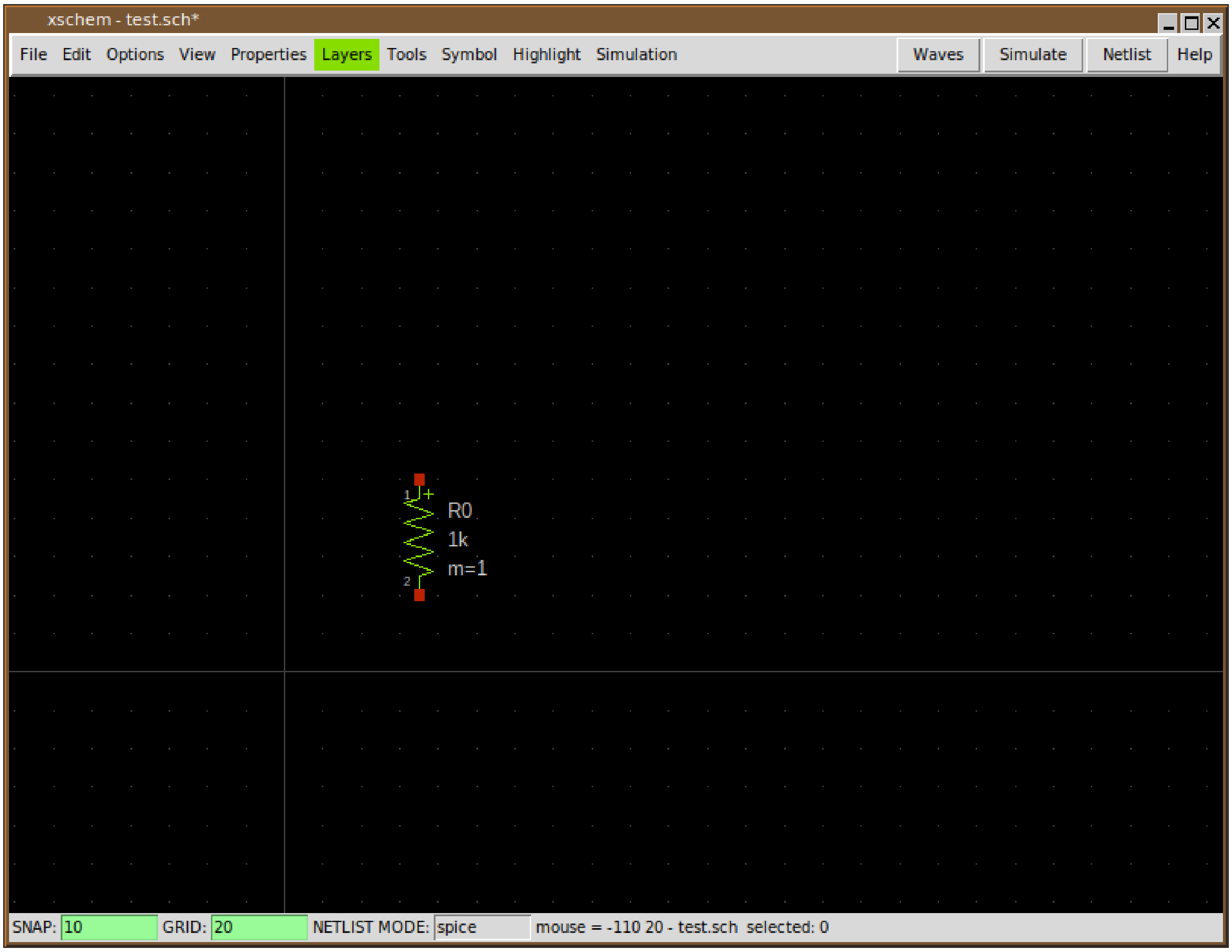
To place a symbol in the schematic window press the **`'Insert'`** key. A file chooser pops up, go to the xschem devices directory (**`.../share/xschem/xschem_library/devices`** in the distribution by default) and select a symbol (res.sym for example). The selected symbol will be instantiated as a component in the schematic at the mouse pointer coordinates.

SYMBOLS

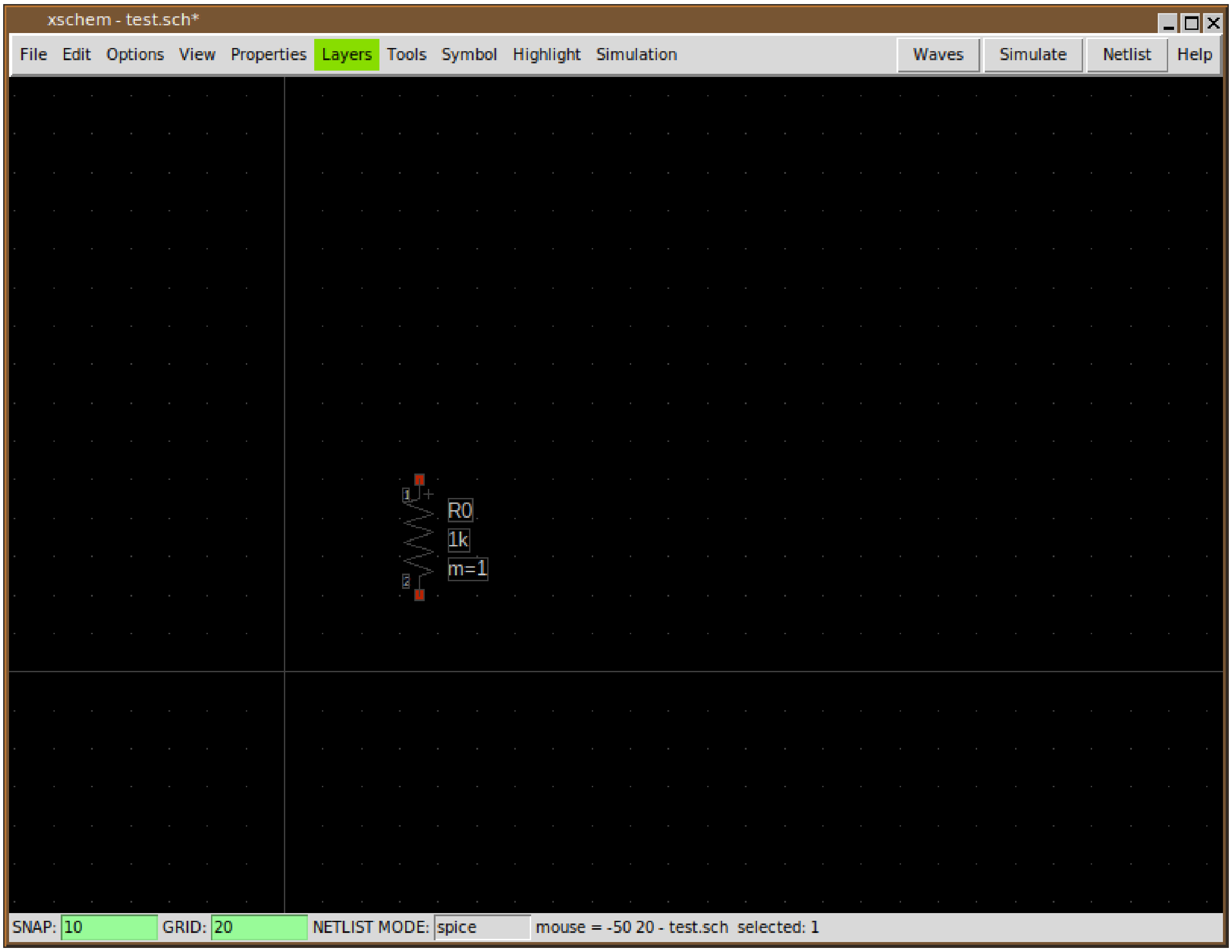
The best way to understand how a symbol is defined is to analyze an existing one. Load a test schematic (for example `test.sch`). Let's consider the resistor symbol. Use the **`Insert`** key to place the **`devices/res.sym`** symbol.


xschem - test.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
Choose symbol
Directory: /mnt/x/home/schippes/share/xschem/xschem_library/devices
own.sym
ym
pmos.sym
pmos3.sym
pmos4.sym
pmoshv4.sym
pmosnat.sym
pnp.sym
port_attributes.sym
res.sym
spice_probe.sym
spice_probe_vdiff.sym
switch.sym
switch_hsp.sym
swi
titl
tlin
use
vcc
vcr
File name: res.sym
Open
Files of type: Symbol files (*.sym)
Cancel
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -20 -10 - /mnt/x/home/schippes/x/test.sch selected: 0

Use the file selector dialog to locate **`res.sym`**.

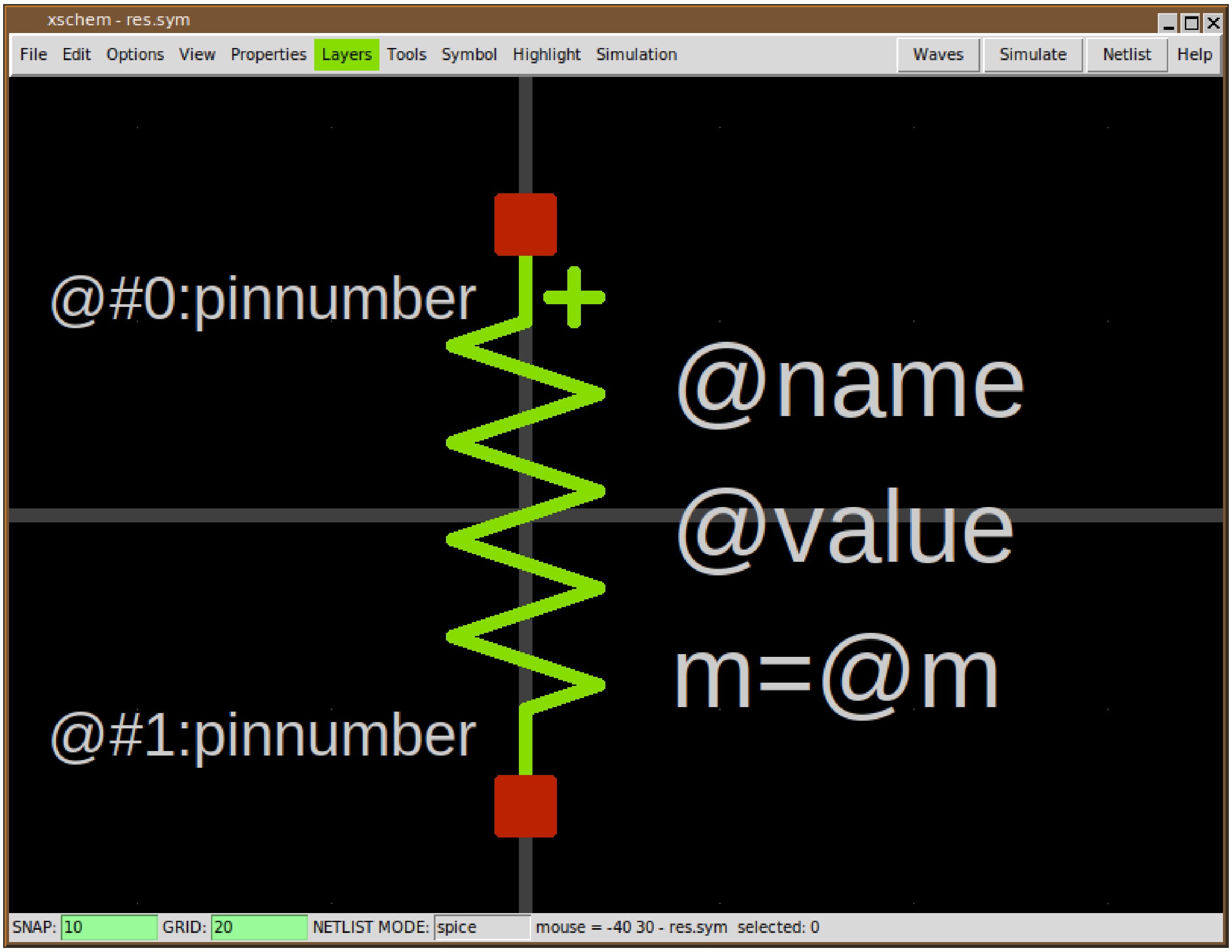

xschem - test.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
R0
1k
m=1
1
2
+
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -110 20 - test.sch selected: 0

Now select the resistor by left-clicking on it (it will turn to grey color)



mouse = -50 20 - test.sch selected: 1

After selecting the component (component is an instance of a symbol) descend into its symbol definition by pressing the **`'i'`** key. XSCHEM will load the **`devices/res.sym`** file and show it in the drawing window. Before descending it asks if you want to save the parent schematic drawing before loading the resistor symbol. Answer 'yes'.



SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -40 30 - res.sym selected: 0

The image above is the 'symbol definition', you can now select individual graphic elements that represent the symbol, lines, rectangles and text. Normally a symbol contains some pins, these are just rectangles drawn on the 'pin' layer, and some graphics / descriptive text. Another fundamental part of symbols are properties. Properties are text strings that define attributes of the symbol, for example:

- The name of the connection pins
- The type of the symbol (spice primitive, subcircuit, documentation)
- The format of the spice/verilog/VHDL netlist for the symbol

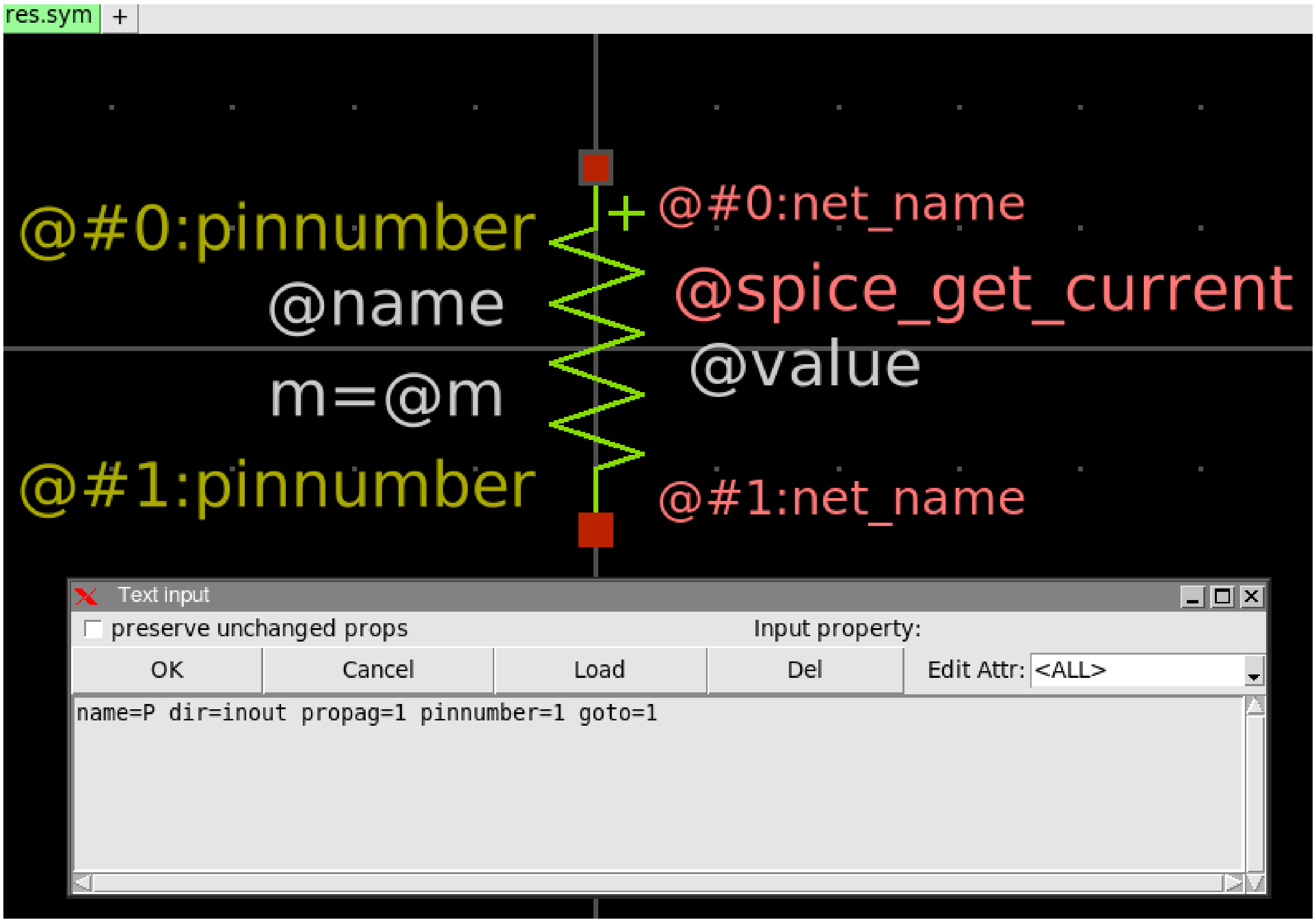
We will return on symbols after explaining properties.

XSCHEM PROPERTIES

Properties are text strings that are associated to XSCHEM objects. All graphic primitives support properties.

- Wires
- Lines
- Polygons
- Rectangles
- Circles/Arcs
- Texts
- Symbol references
- Global attributes

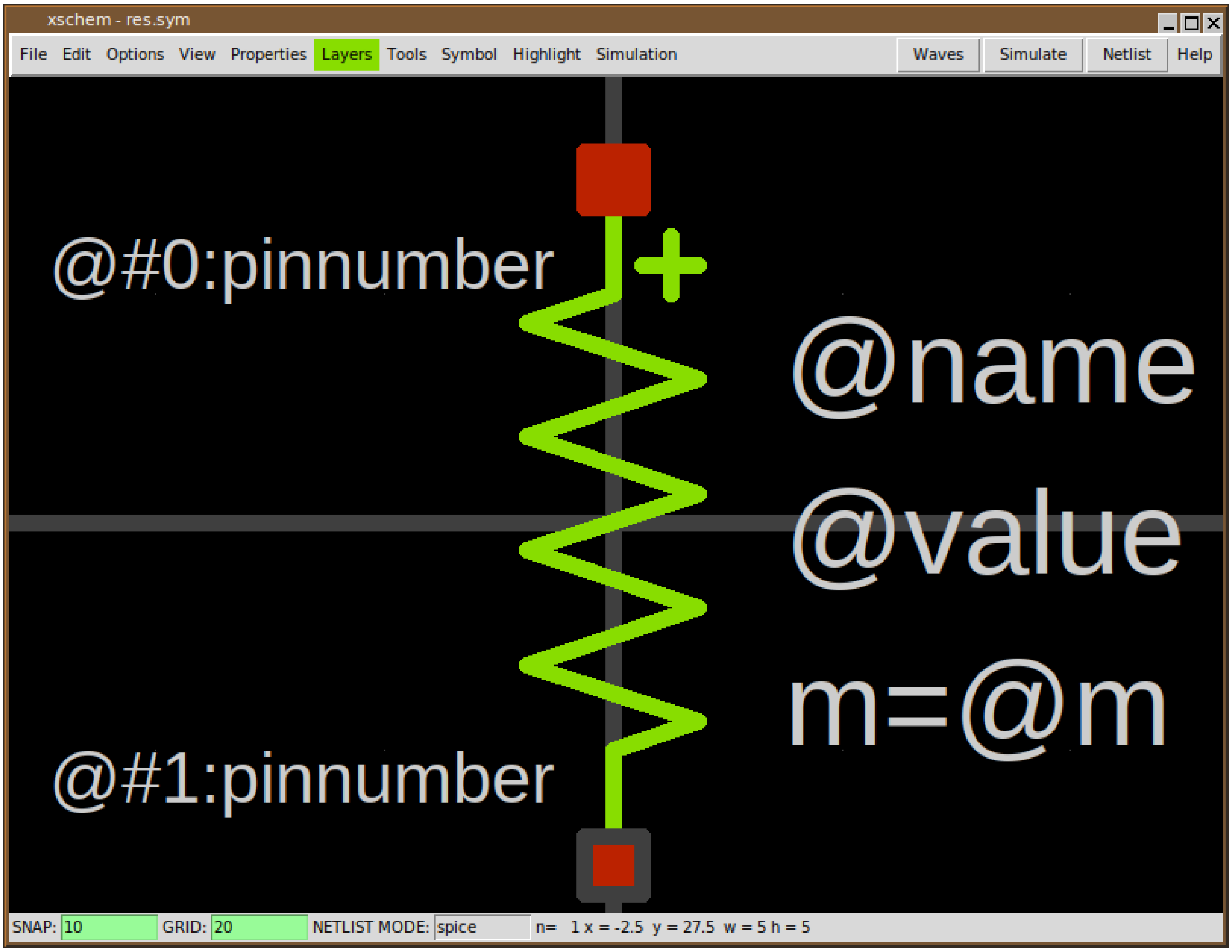
Consider for example the **`res.sym`** symbol (you may open it with the **`File->Open`** menu item) if you click inside one of the red pins and press the 'edit property' bindkey **`'q'`** a dialog box shows the property string associated with the selected pin:



```
name=P dir=inout propag=1 pinnumber=1 goto=1
```

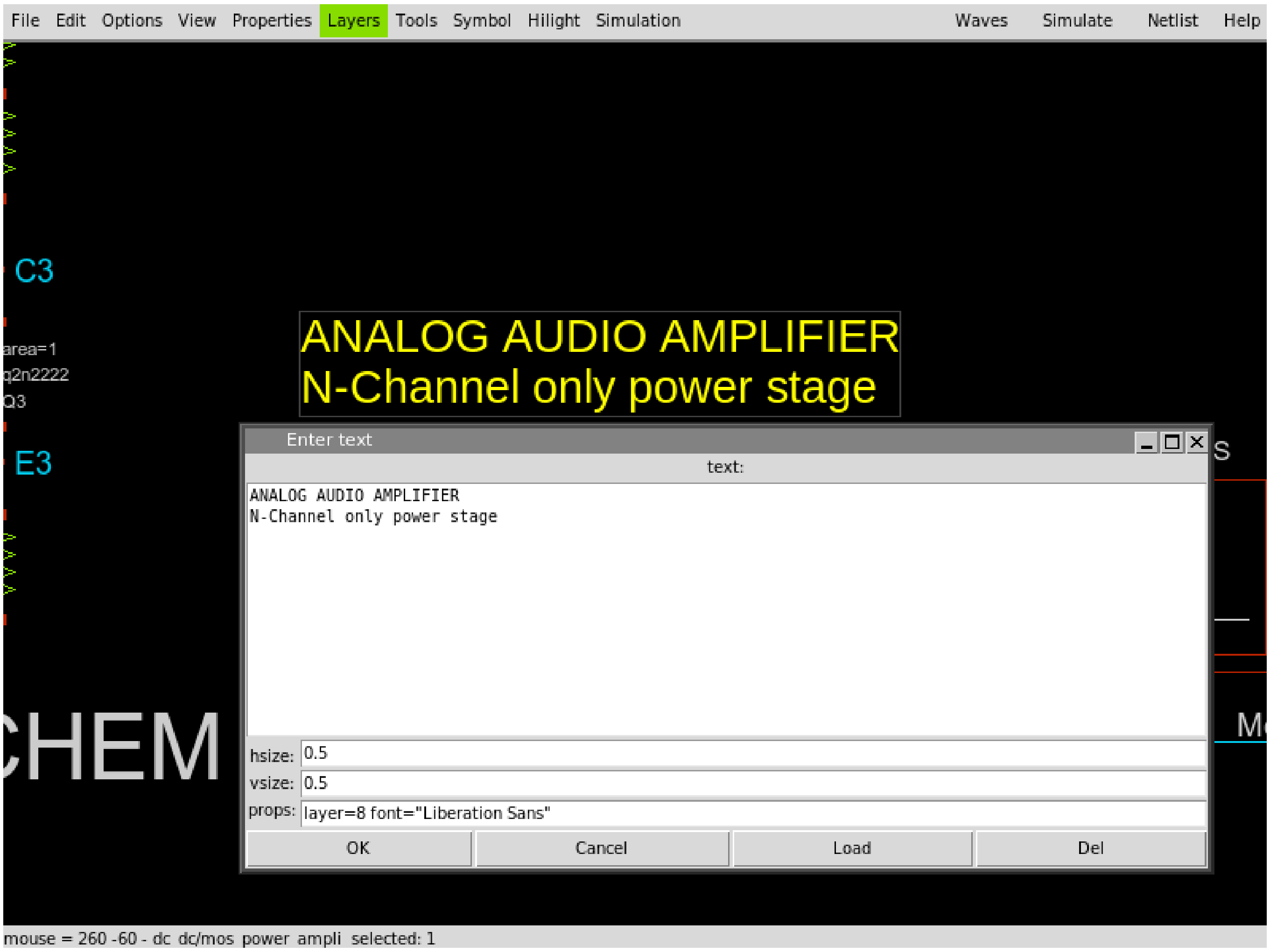
The **`name=p dir=inout propag=1 pinnumber=1`** property string tells that the selected pin name is **`'p'`**, this will be the symbol positive pin name in the produced netlist. The property string also defines a **`dir`** attribute with value

inout. This tells XSCHEM that electrically this is an input/output pin. This is important when producing VHDL/verilog netlists. The **propag=1** tells XSCHEM that when we select a wire attached to this pin (which is located at index 0 in xschem) the highlight will propagate to the other pin (with index 1). To view the xschem index of a pin click and hold the mouse on it, the index will be shown as **n= <number>** in the bottom status line:


xschem - res.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@#0:pinnumber
+
@name
@value
m=@m
@#1:pinnumber

SNAP: 10 GRID: 20 NETLIST MODE: spice n= 1 x = -2.5 y = 27.5 w = 5 h = 5

The **pinnumber=1** attribute is used when exporting to pcb software (via the tEDAx netlist) and tells to which pin number on the resistor footprint this positive pin is bound. The second (bottom) pin property string is **name=m dir=inout propag=0 pinnumber=2** and this defines the negative pin. The text primitives also have properties. For texts the property string may be used to specify font and the layer to use for displaying text.



```
ANALOG AUDIO AMPLIFIER
N-Channel only power stage
```

hsize: 0.5
vsize: 0.5
props: layer=8 font="Liberation Sans"

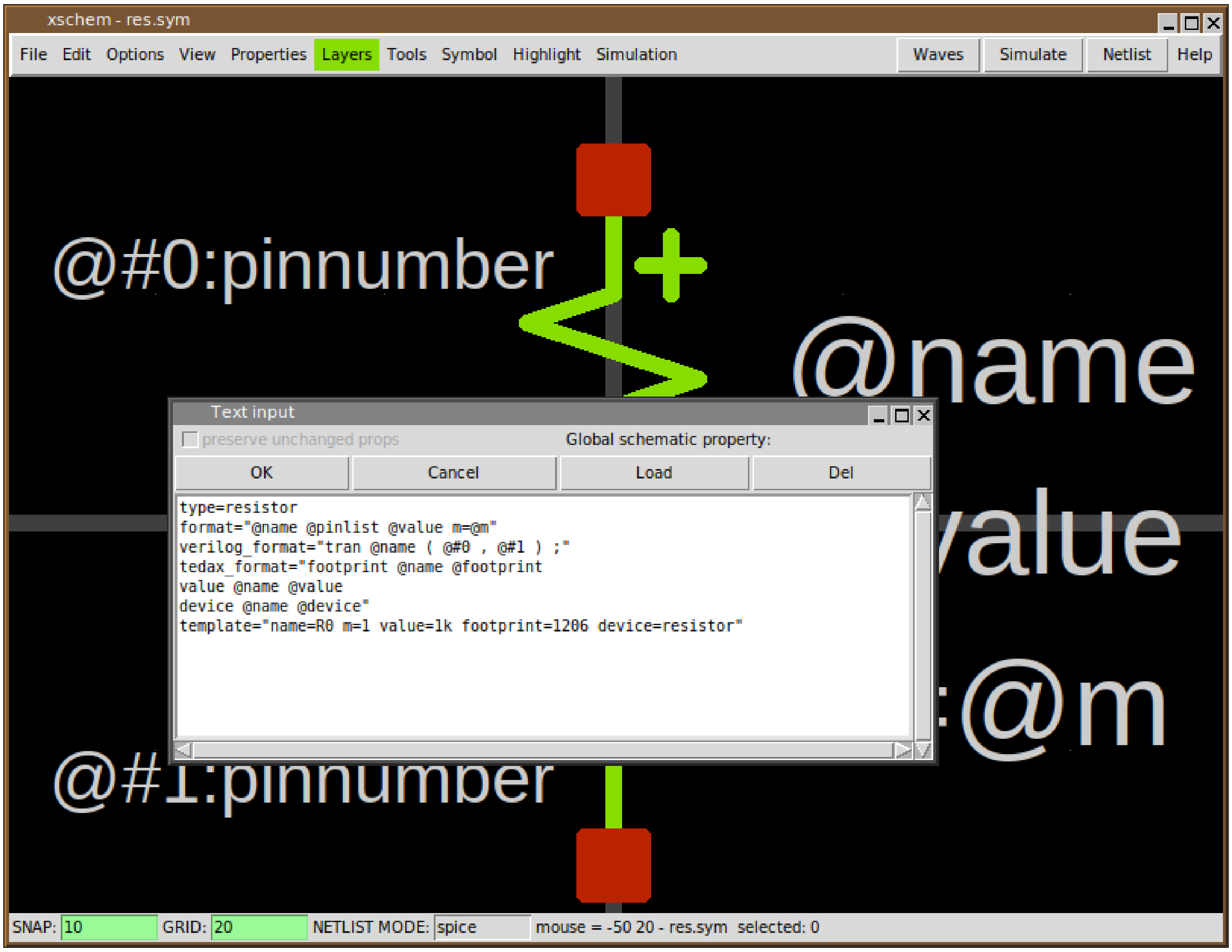
mouse = 260 -60 - dc dc/mos power ampli selected: 1

GLOBAL PROPERTIES

If you click outside of any displayed graphics in XSCHEM the selection set will be cleared. Clicking the edit property `'q'` key when nothing is selected will display the global property string of the schematic (.sch) or symbol window (.sym).

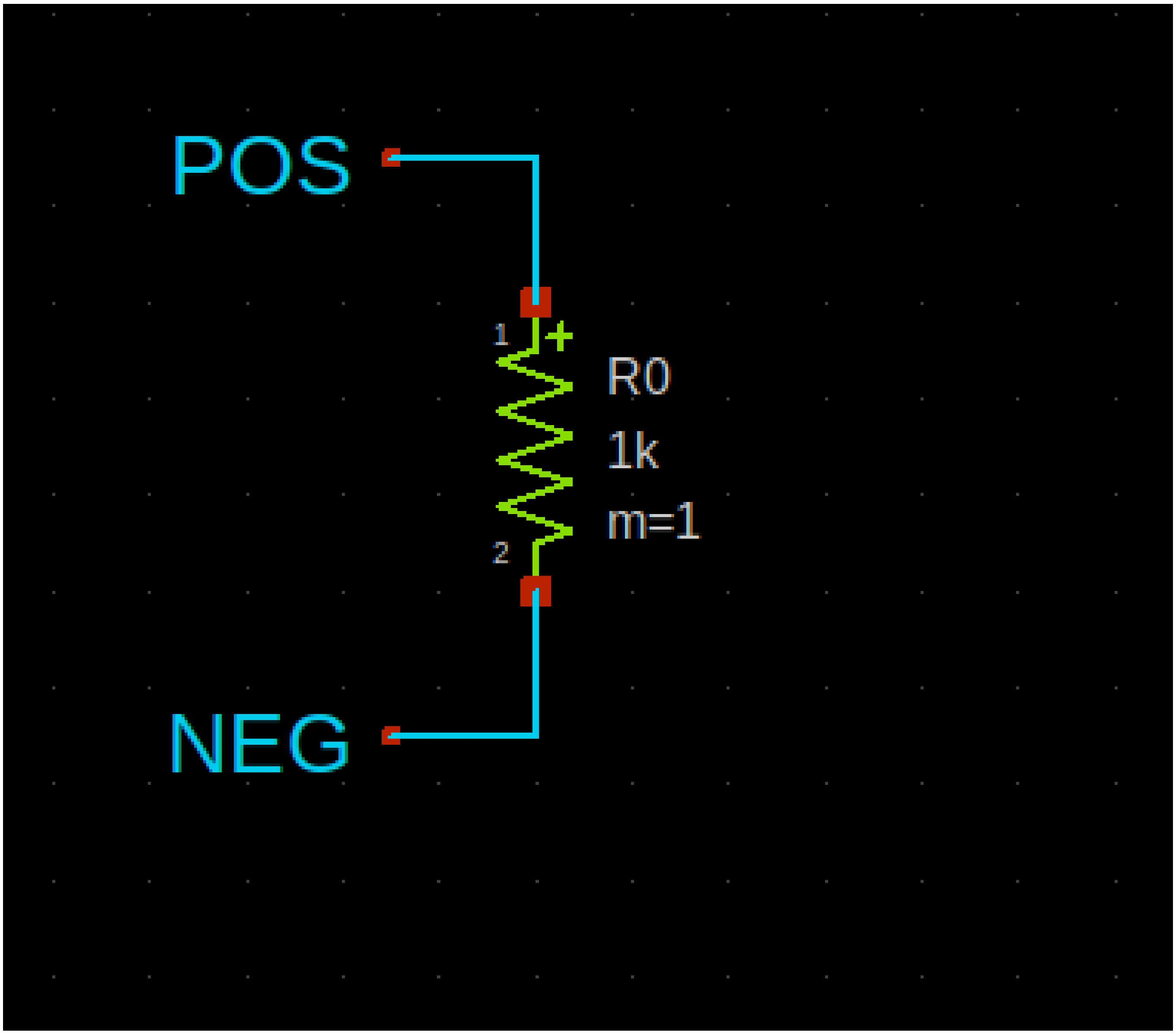
There is actually one different global property string defined for any available netlisting modes plus one global property string for symbol definition (file format 1.2), so if XSCHEM is set to produce SPICE netlists the SPICE global property string is displayed.

So, in addition to properties associated to graphical objects and symbols, we also have properties associated to schematic (.sch) and symbol files (.sym)

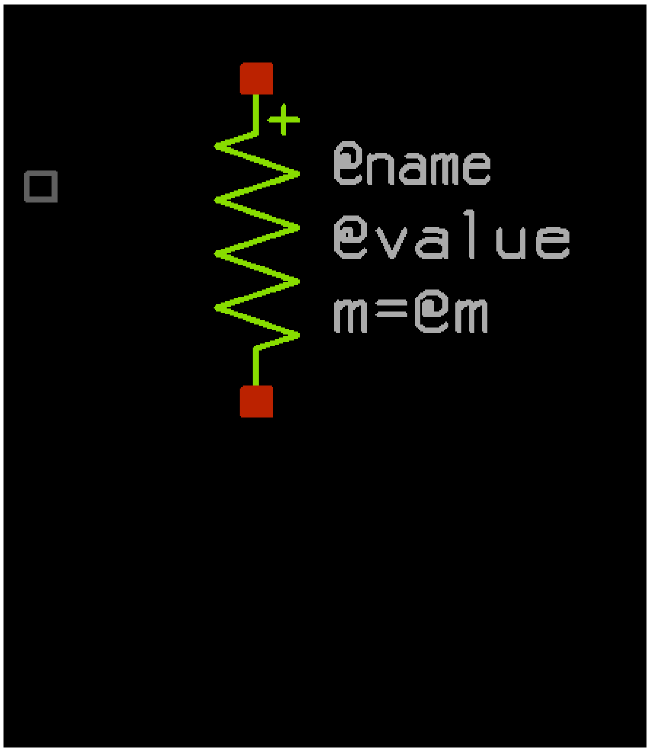

xschem - res.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@#0:pinnumber
@name
@value
m=@m
@#1:pinnumber
Text input
preserve unchanged props
Global schematic property:
OK Cancel Load Del
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -50 20 - res.sym selected: 0

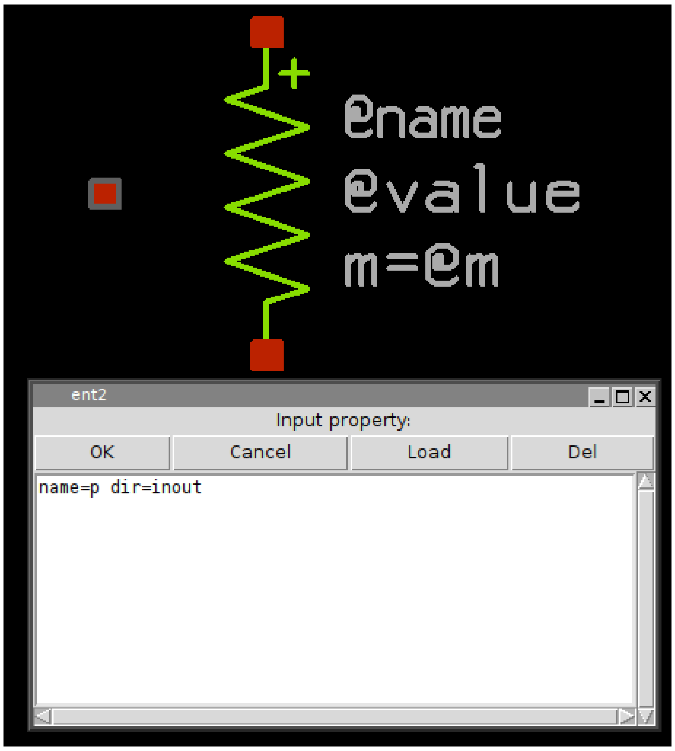
```
type=resistor
format="@name @pinlist @value m=@m"
verilog_format="tran @name ( @#0 , @#1 ) ;"
tedax_format="footprint @name @footprint
value @name @value
device @name @device"
template="name=R0 m=1 value=1k footprint=1206 device=resistor"
```

In the above 'Symbol' global property string, the **`format`** attribute defines the format of the SPICE netlist. The SPICE netlist element line starts with the symbol name (in this case a resistor so 'rxxxxx'), the list of pins, the resistor value and a multiplicity factor (m).
`@pinlist` will resolve to the parent nets attached to the resistor nodes, in the order they appear in the symbol (in this example; first node = 'p', second node = 'm').
We will return on component instantiation later, but for now, considering the following picture:



The **@name** will expand to R0, **@pinlist** for the **R0** component will expand to **POS NEG**.
@value resolves to the resistor value assigned in component instantiation. The **template** attribute defines default values if component instantiation does not define values for them.
If you want to add a pin to an existing symbol you may copy one of these. Select a pin, press the copy **'c'** bindkey and place a new copy of it somewhere.



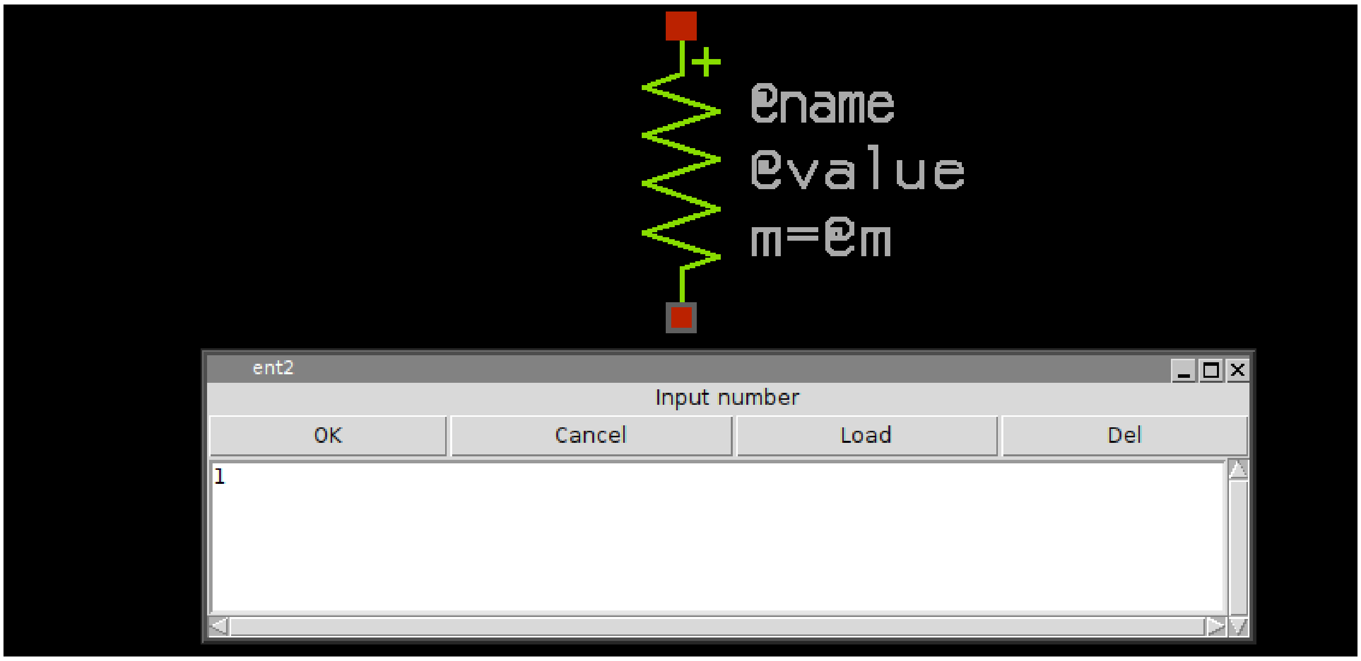


```
name=p dir=inout
```

After copying the pin you may change its properties, for example you will change its property string to something like: **`name=body dir=in`** (just as an example).
Note that pins in symbols are nothing more than rectangles drawn with the **`pin`** layer; instead of copying an existing one you may create it from scratch, select the pin layer from the **`Layers`** menu, point the mouse where you want to place the pin, press the **`'r'`** bindkey and drag the mouse to the desired pin size. There is no inherent limit or assumption on pin sizes, you are allowed to create any rectangular/square sizes. After placing the rectangle you must create a property string by selecting it and pressing the **`'q'`** bindkey. An empty string is shown in the dialog. Add a valid string as explained and you are all done.

PIN ORDERING

An important aspect for symbols is the order of the pins when producing the netlist. There are some rules in the order for example in SPICE netlist syntax; for example a Bipolar transistor has 3 pins and should be in a specific order (collector, base, emitter). When done placing pins on a newly created symbol you can specify the order by selecting the one that must be the first in the netlist and hitting the **`'<shift>S'`** bindkey; set the number to zero; this will make the selected pin the first one. Next, select the second pin and again hit **`'<shift>S'`**, set its number to 1 and so on. By doing so you have defined a specific pin ordering of the symbol.



PRIMITIVE OBJECT PROPERTIES

The following attribute may be set on lines, arcs, polygons, rectangles:

- **`dash=n`**, where n = integer. This specifies dashed mode drawing for the specified object.

The following attribute may be set on arcs, polygons, rectangles:

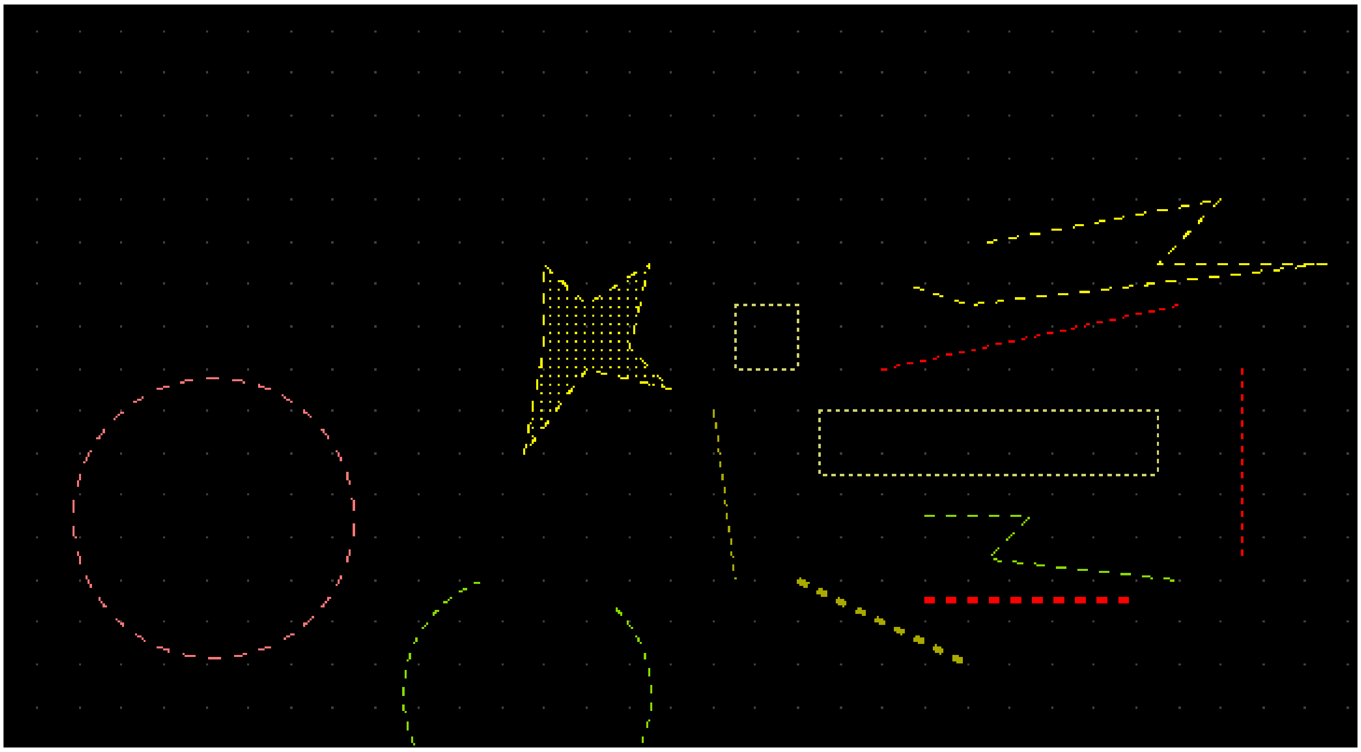
- **`fill=true`**. This specifies to fill the object with the layer predefined fill style. The default for rectangles (for historical reasons) is to use fill style if not specified. For arcs and polygons the default is to not use fill if unspecified.

The following attribute may be set on rectangles and instances:

- **`lock=true`** (or 1) can be set to disallow selecting the object. You always can double click on it to edit the attributes and reset lock to false (or 0). This is useful for title objects or frames you don't want to select while editing inside them.

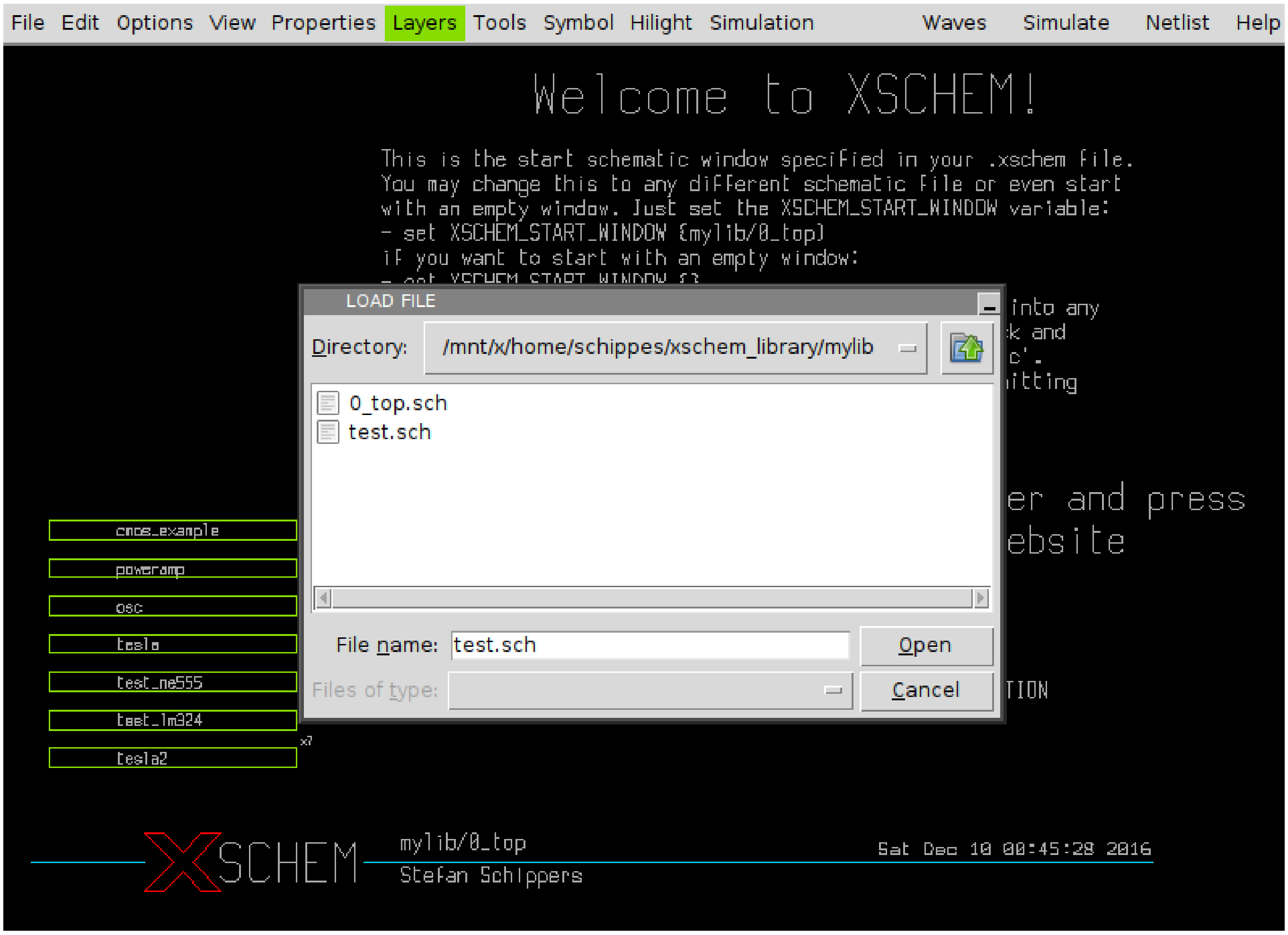
The following attribute may be set on wires and lines:

- **`bus=true`**. This specifies to draw a wider line. Mostly used to display wire buses.



COMPONENT INSTANTIATION

In the RUN XSCHEM slide some instructions were provided as examples to place a component in the schematic. Now we will cover the topic in more detail with emphasis on component properties. Start by opening a test schematic window (you may delete any existing stuff in it if any).



Welcome to XSCHEM!

This is the start schematic window specified in your .xschem file.
You may change this to any different schematic file or even start
with an empty window. Just set the XSCHEM_START_WINDOW variable:
- set XSCHEM_START_WINDOW {mylib/0_top}
if you want to start with an empty window:

LOAD FILE

Directory: /mnt/x/home/schippes/xschem_library/mylib

0_top.sch
test.sch

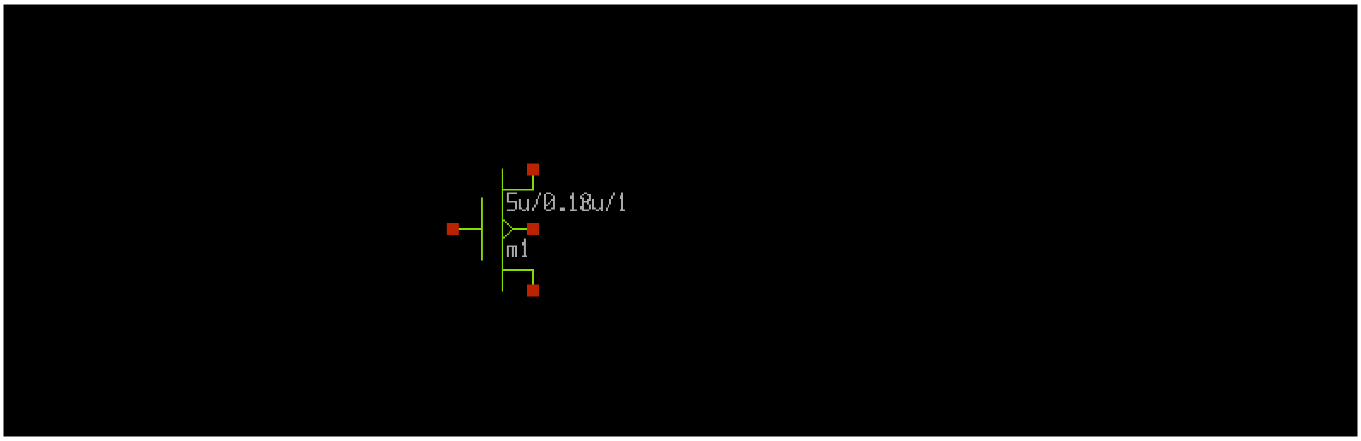
File name: test.sch

Files of type:

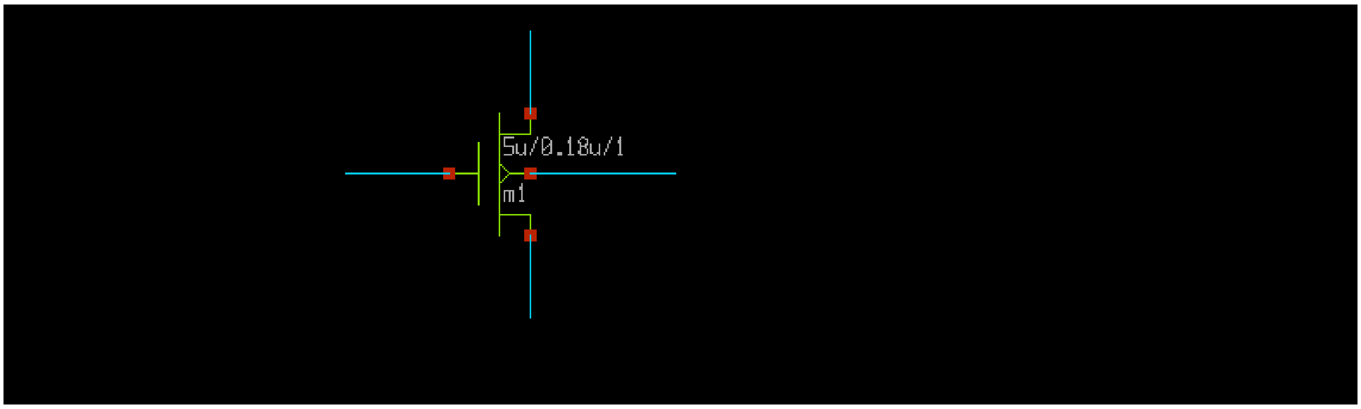
Open
Cancel

mylib/0_top
Stefan Schippers
Sat Dec 10 00:45:28 2016

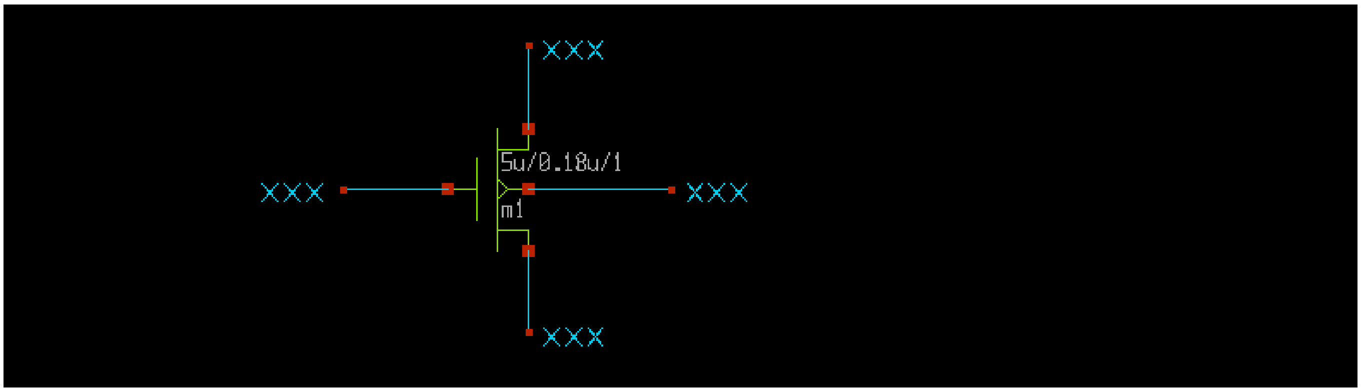
Now start by inserting a component, consider for example **`devices/nmos4.sym`**; press the **`Insert`** key, navigate to the **`devices`** design library and open the **`nmos4.sym`** symbol.



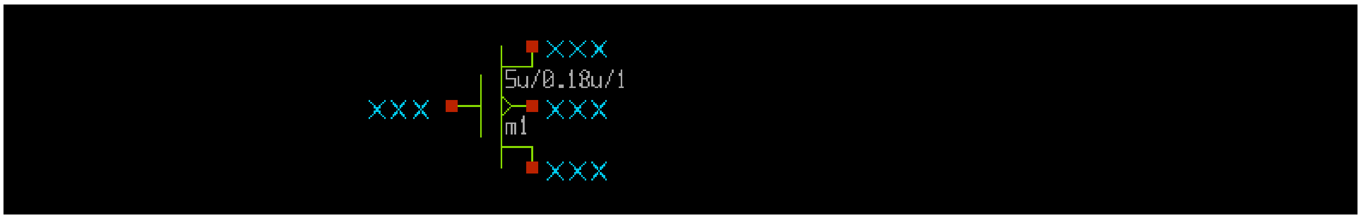
Now draw some wires on each pin of the nmos; place the mouse pointer on the component pins and use the **'w'** bindkey.



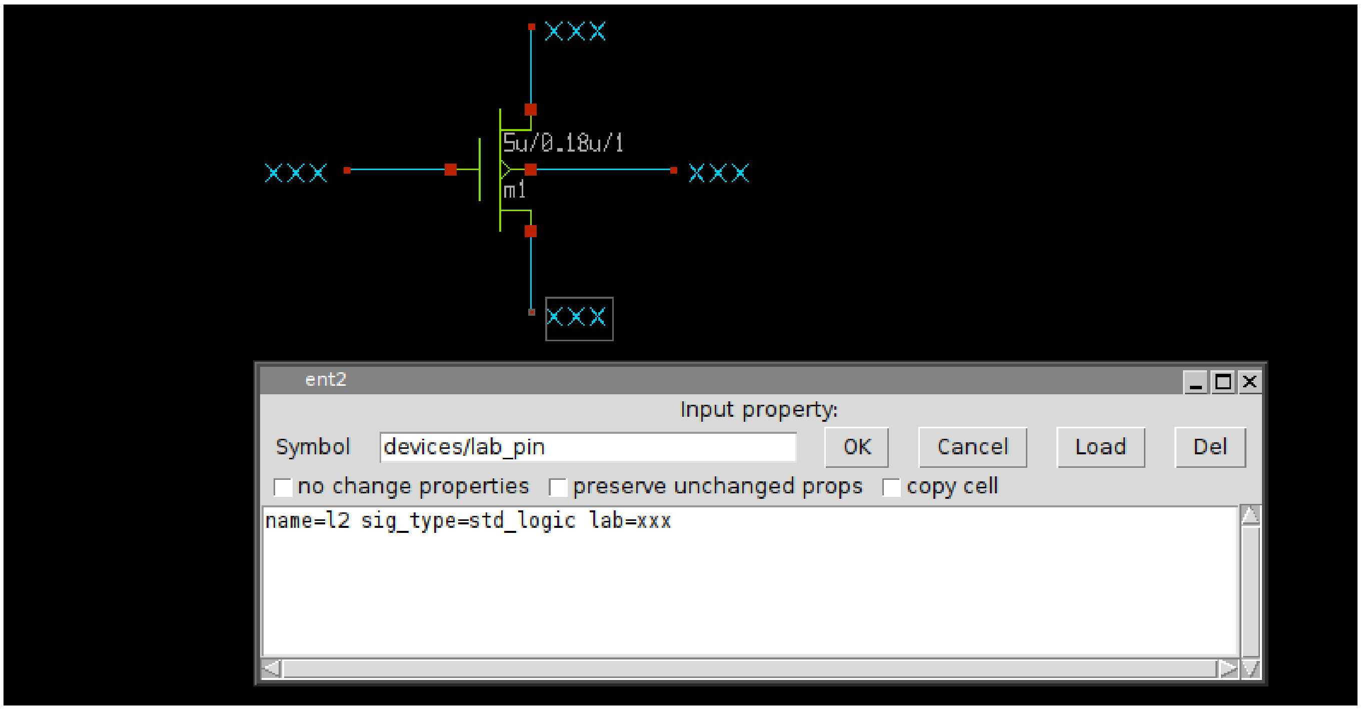
we need now to put labels on wire ends: use the **Insert** key and locate the **devices/lab_pin.sym** symbol. After the **lab_pin** symbol is placed you can move it by selecting it with the mouse and pressing the **'m'** bindkey. You can also flip (**'F'**) and rotate while moving (**'R'**) to adjust the orientation. After placing the first one you may copy the others from it (**'c'** bindkey). The end result should look like this:



This is what an electrical circuit is all about: a network of wires and components. In this schematic we have 5 components (4 labels and one mos) and 4 nets. It is not mandatory to put a wire segment between component pins; we could equally well do this:



This circuit is absolutely equivalent to the previous one: it will produce the same device connectivity netlist.
Now we need to set appropriate labels on the NMOS terminals. This is -again- accomplished with component properties.
Select the wire label on the nmos source pin and press the `'q'` bindkey:

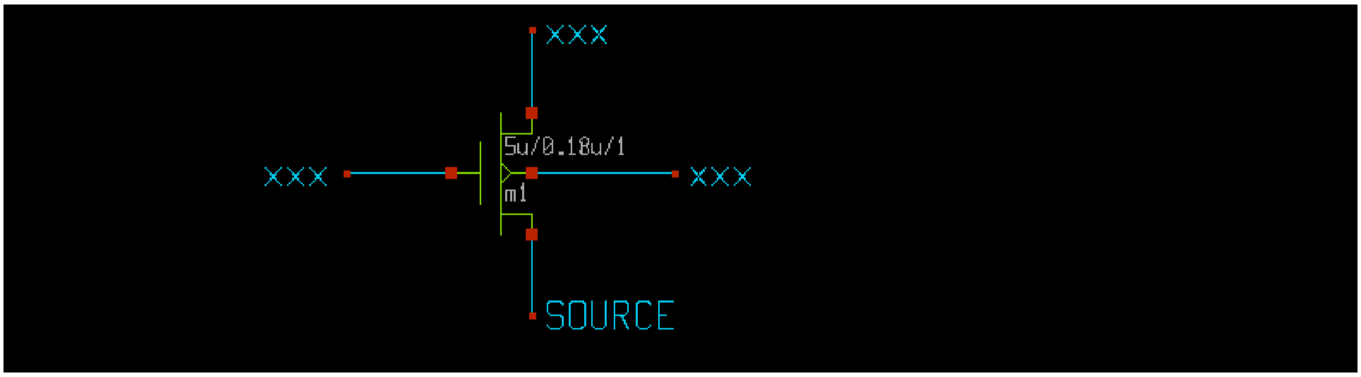


Symbol: devices/lab_pin

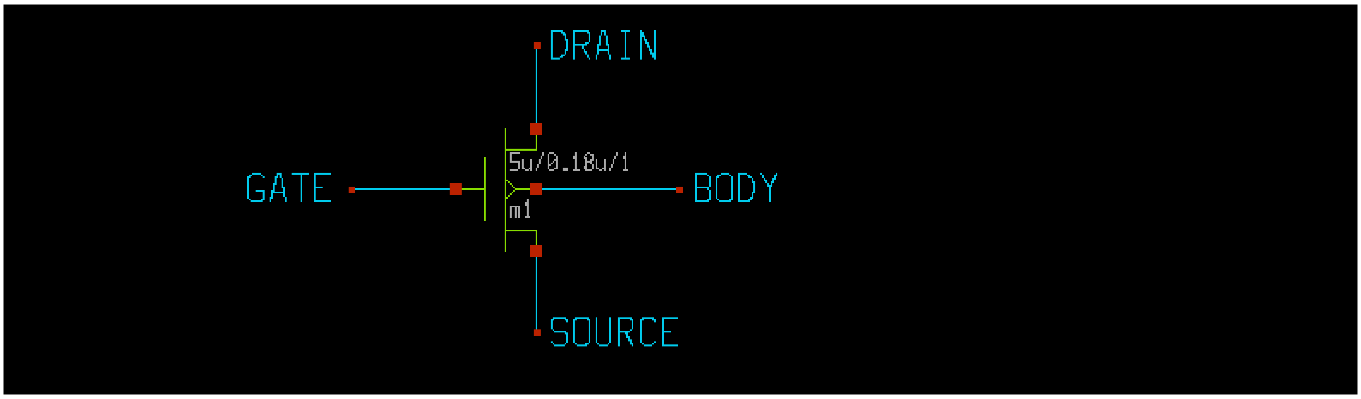
☐ no change properties ☐ preserve unchanged props ☐ copy cell

```
name=l2 sig_type=std_logic lab=xxx
```

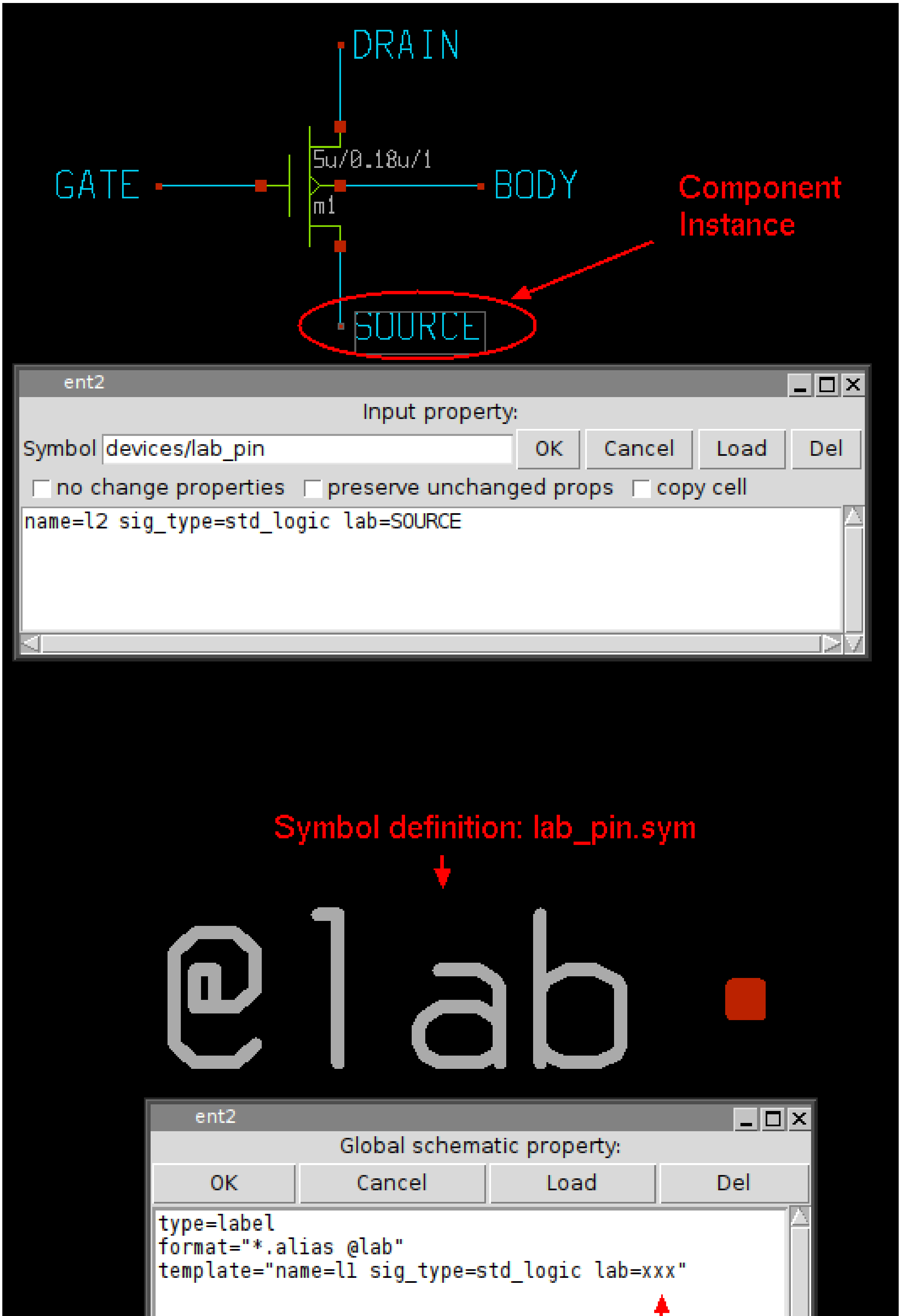
Now, replace the 'xxx' default string in the dialog with a different name (example: SOURCE) After clicking **OK** the source terminal will have the right label.



repeat the process for the remaining GATE, DRAIN, BODY terminals;



The following picture shows the **lab_pin** component with its properties and the corresponding symbol definition with its global properties (remember global properties in the xschem_properties slide)



Symbol: devices/lab_pin

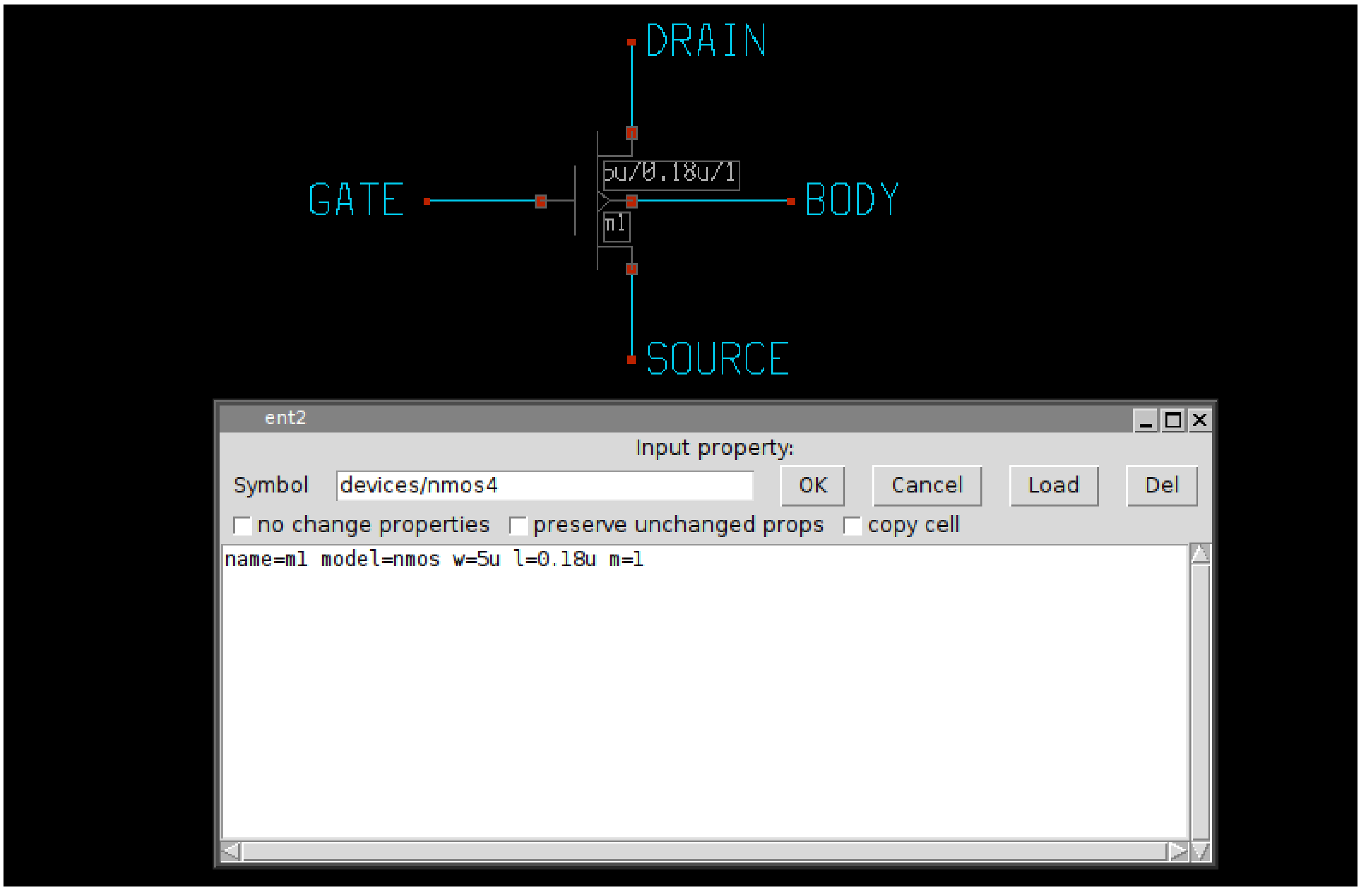
```
name=l2 sig_type=std_logic lab=SOURCE
```

```
type=label
format="*.alias @lab"
template="name=l1 sig_type=std_logic lab=xxx"
```

when building the netlist XSCHEM will look for wires that touch the red square of the lab_pin component and name that wires with the component 'lab' property. for example the SPICE netlist of the circuit will be:

```
m1 DRAIN GATE SOURCE BODY nmos w=5u l=0.18u m=1
```

We need now to edit the nmos properties. Select it and press the **`'q'`** bindkey

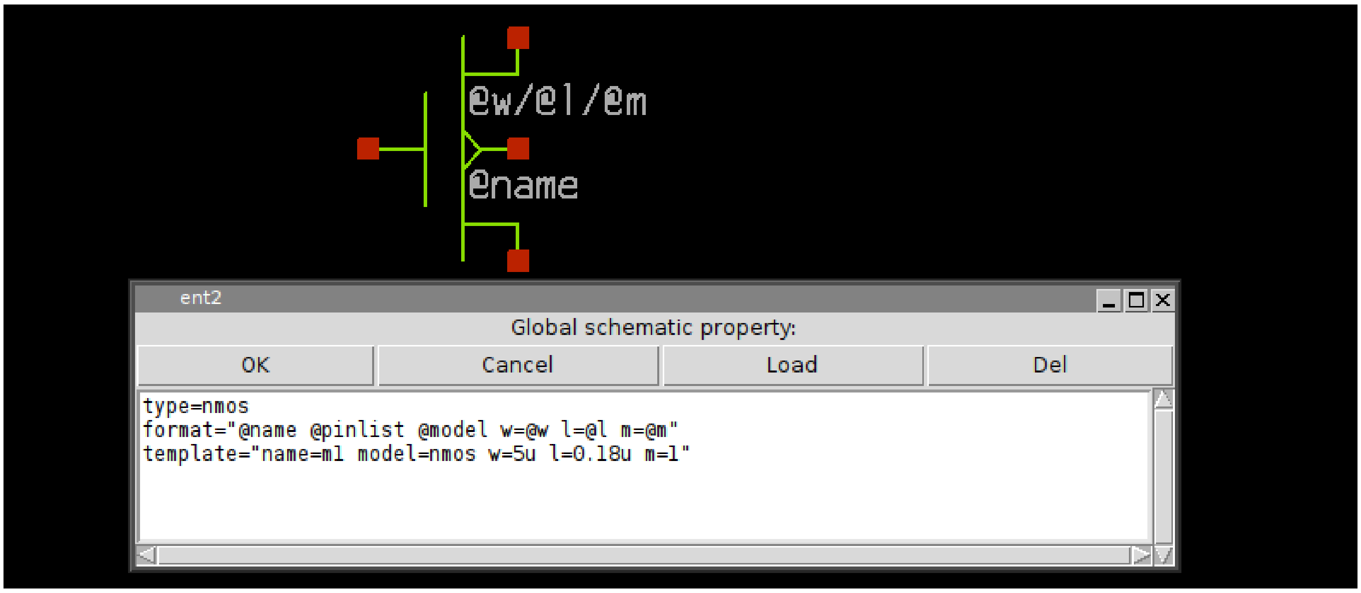


```
name=m1 model=nmos w=5u l=0.18u m=1
```

from the edit properties dialog you see there are 5 attributes with values defined:

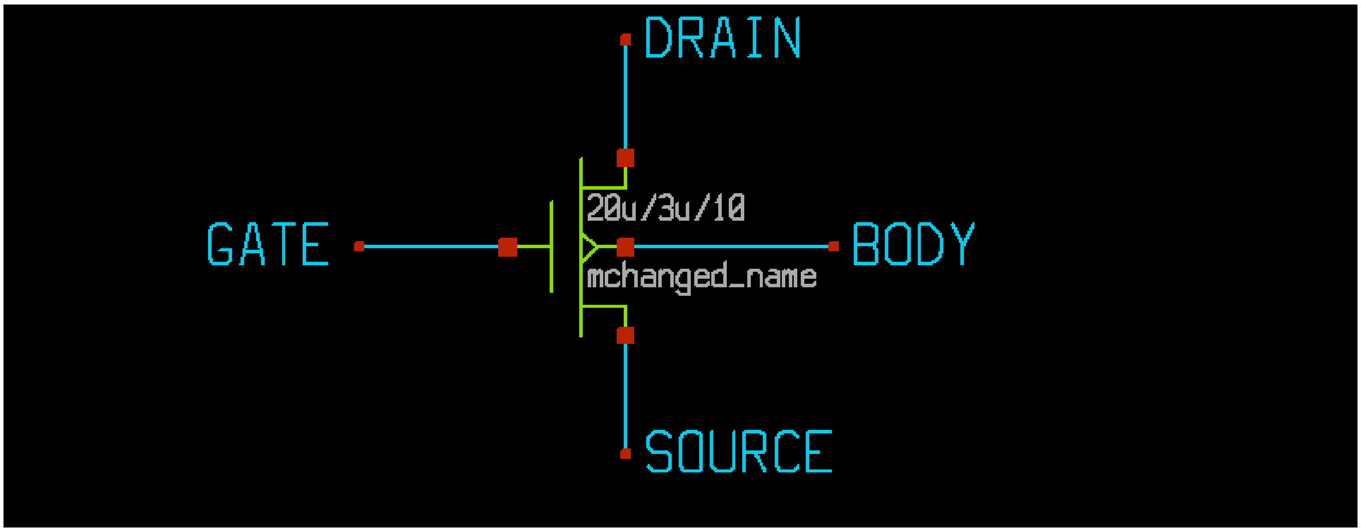
- The component name **`name=m1`**.
- The spice model to be used in simulation **`model=nmos`**.
- The transistor width **`w=5u`**.
- The transistor channel length **`l=0.18u`**.
- The number of parallel transistors (multiplicity) **`m=1`**.

We have never defined a value for these properties. These are the default values defined in the **`template`** attribute in the global **`nmos4.sym`** property string.



```
type=nmos
format="@name @pinlist @model w=@w l=@l m=@m"
template="name=m1 model=nmos w=5u l=0.18u m=1"
```

We may want to change the dimensions of the transistor; simply change the **`w`** and **`l`** attribute values.
Also the component name may be changed as long as it is unique in the current schematic window. All simulators require that components are unique, it is not permitted to have 2 components with identical name, so XSCHEM enforces this.



If a name is set that matches an existing component xschem will rename it keeping the first letter (**`m`** in this example) and appending a number (so you might end up in something like **`m23`** if there are many devices).

the **`name`** attribute is unique in the schematic window, and must be placed first in the property string. The name is also used by xschem to efficiently index it in the internal hash tables.

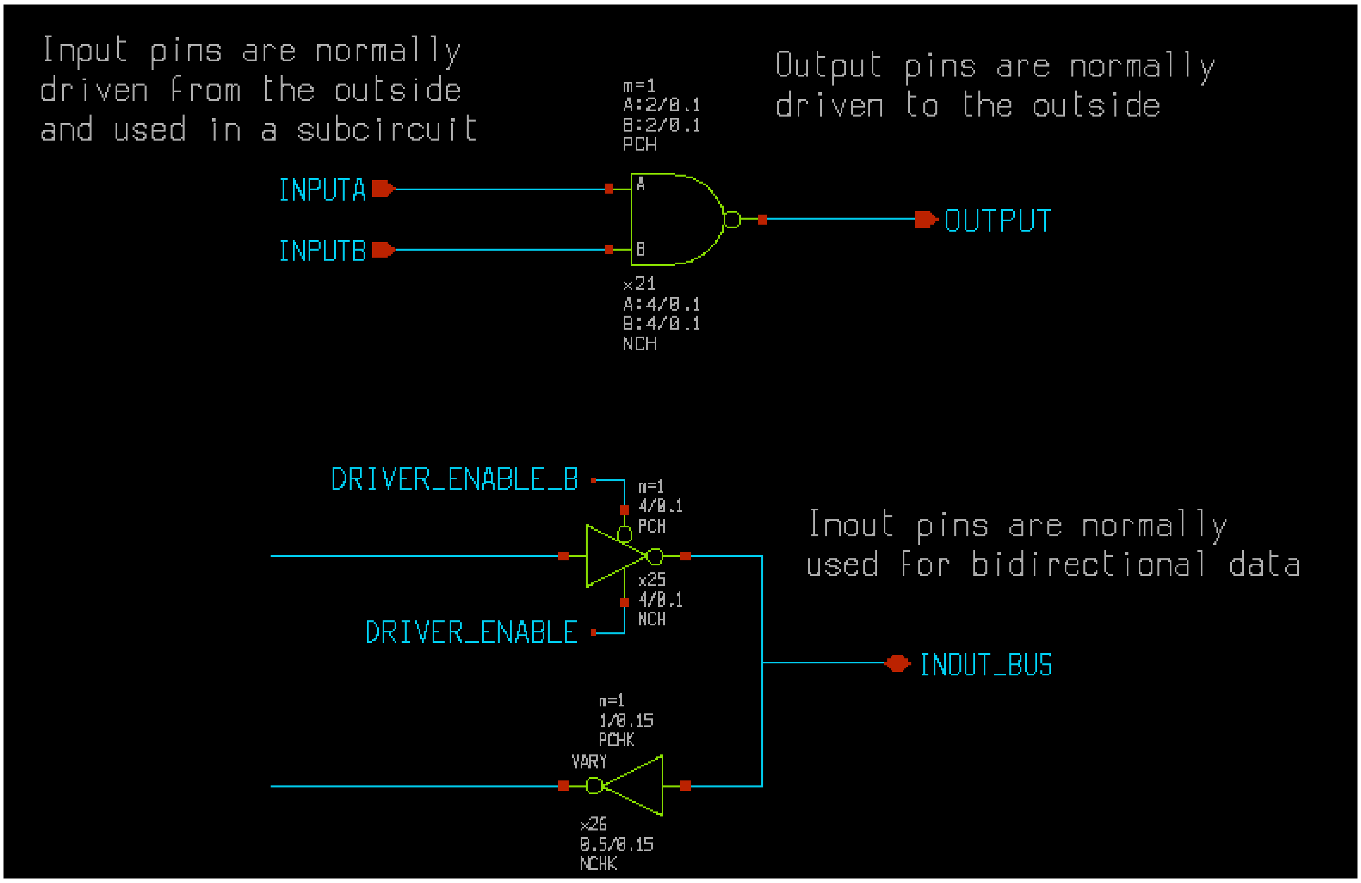
SPECIAL COMPONENTS

General purpose

- **`devices/ipin.sym`**
- **`devices/opin.sym`**

- **`devices/iopin.sym`**

These components are used to name a net or a pin of another component. They do not have any other function other than giving an explicit name to a net.


Input pins are normally driven from the outside and used in a subcircuit
Output pins are normally driven to the outside
INPUTA
INPUTB
OUTPUT
m=1
A:2/0.1
B:2/0.1
PCH
x21
A:4/0.1
B:4/0.1
NCH
DRIVER_ENABLE_B
DRIVER_ENABLE
Inout pins are normally used for bidirectional data
INOUT_BUS
m=1
4/0.1
PCH
x25
4/0.1
NCH
m=1
1/0.15
PCHK
VARY
x26
0.5/0.15
NCHK

- **`devices/lab_pin.sym`**
- **`devices/lab_wire.sym`**
- **`devices/launcher.sym`**
- **`devices/architecture.sym`**

This prints global attributes of the schematic. Attributes of this symbol should not be set. It is a readonly symbol printing top-level schematic properties.

Spice netlist special components

- **`devices/code.sym`**
- **`devices/code_shown.sym`**

these symbols are used to place simulator commands or additional netlist lines as text into the schematic.

Verilog netlist special components

- **`devices/verilog_timescale.sym`**
- **`devices/verilog_preprocessor.sym`**

VHDL netlist special components

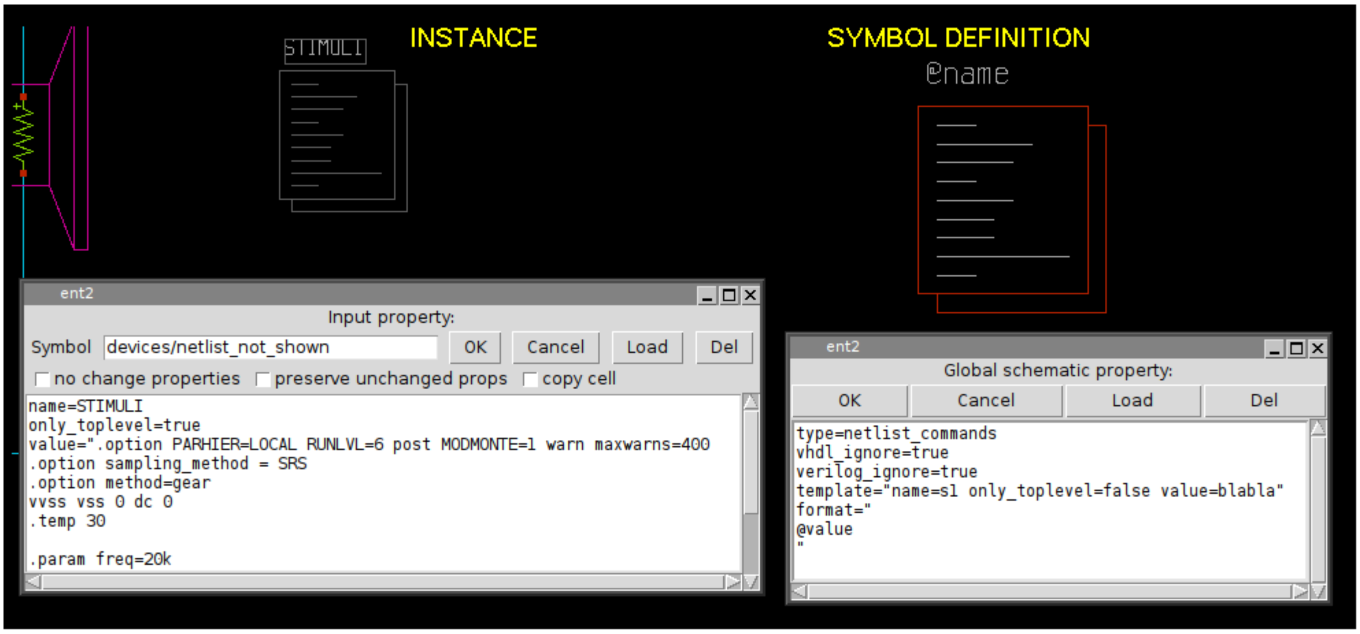
- **`devices/use.sym`**
- **`devices/package.sym`**
- **`devices/package_not_shown.sym`**
- **`devices/arch_declarations.sym`**
- **`devices/attributes.sym`**
- **`devices/port_attributes.sym`**
- **`devices/generic_pin.sym`**
- **`devices/lab_generic.sym`**

SYMBOL PROPERTY SYNTAX

GENERAL RULES

For symbols a global property string (to show it press `'q'` when nothing is selected and **`Options->Symbol global attrs`** is selected) defines at least 3 attributes:

- **`type`** defines the the type of symbol. Normally the type attribute describes the symbol and is ignored by XSCHEM, but there are some special types:
    - **`subcircuit`**: the symbol has an underlying schematic representation, when producing the netlist XSCHEM has to descend into the corresponding schematic. This will be covered in the subcircuits chapter.
    - **`primitive`**: the symbol has a schematic representation, you can descend into it but the netlister will not use it. This is very useful if you want to netlist a symbol using only the **`format`** (or **`vhdl_format`** or **`verilog_format`** depending on the netlist type) attribute or use the underlying schematic. By setting the attribute back to **`subcircuit`** and deleting (or setting to **`false`**) the **`verilog_format`** of **`vhdl_format`** attribute you can quickly change the behavior. For spice netlists the **`format`** attribute is always used also for subcircuits instantiation so always leave it there.
    - Any value different from **`subcircuit`** or **`primitive`** will cause xschem to not use any schematic file even if it exists. Xschem will not allow to descend into an existing schematic.
    - **`label`**: the symbol is used to label a net. These type of symbols must have one and only one pin, and the template string must define a **`lab`** attribute that is passed at component instantiation to name the net it is attached to.
    - **`probe`**: this denotes a probe symbol that may be backannotated with a backannotation script (example: ngspice_backannotate.tcl).
    - **`ngprobe`**: This is a probe element that uses a 'pull' method to fetch simulation data and display it in current schematic. The data displayed is thus dynamic, multiple instances of the same symbol with annotators will display operating point data for that particular instance without the need to update the backannotation as is required for annotators using the 'push' annotation method.
    - **`netlist_commands`**: the symbol is used to place SPICE commands into a spice netlist. It should also have a **`value`** attribute that may contain arbitrary text that is copied verbatim into the netlist. More on this in the netlist slide.



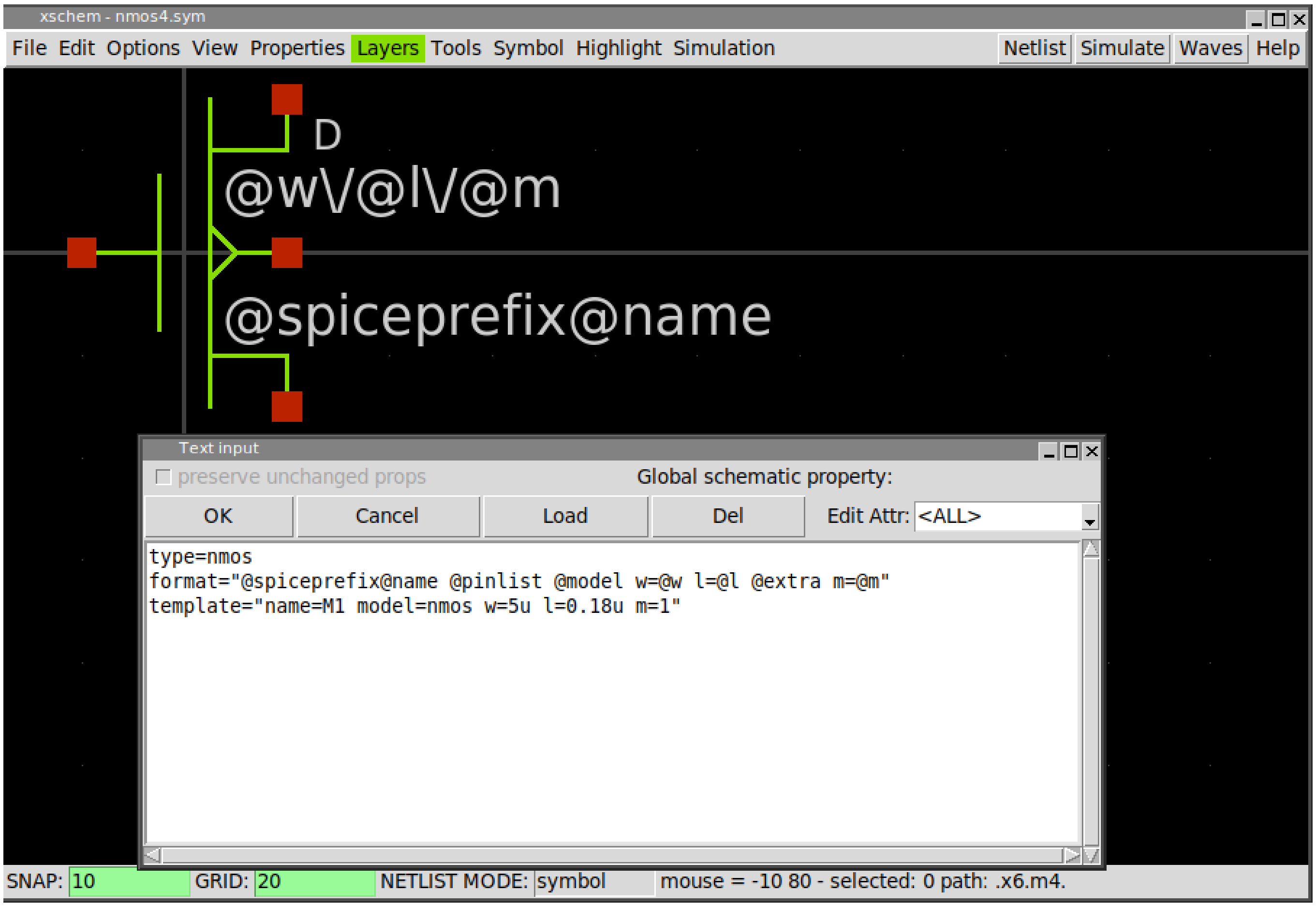
Symbol devices/netlist_not_shown

```
name=STIMULI
only_toplevel=true
value=".option PARHIER=LOCAL RUNLVL=6 post MODMONTE=1 warn maxwarns=400
.option sampling_method = SRS
.option method=gear
vvss vss 0 dc 0
.temp 30

.param freq=20k
```

```
type=netlist_commands
vhdl_ignore=true
verilog_ignore=true
template="name=s1 only_toplevel=false value=blabla"
format="
@value
"
```

Only symbols of type **`subcircuit`** or **`primitive`** may be descended into with the **`'e'`** bindkey if they have a schematic view.

- **`format`**: The format attribute defines the syntax for the SPICE netlist. the **`@`** character is a 'substitution character', it means that the token that follows is a parameter that will be substituted with the value passed at component instantiation. If no value is given there a value will be picked from the attribute declared in the **`template`** string.
  The **`@pinlist`** is a special token that will be substituted with the name of the wires that connect to symbol pins, in the order they are created in the symbol. See the pin ordering section in the xschem properties slide. if the order of pins for a NMOS symbol is for example, d,g,s,b, then @pinlist will be expanded when producing a netlist to the list of nets that connect to the symbol drain, gate, source, body respectively. There is also a special way to define single pins: **`@@d`** for example will be replaced by XSCHEM with the net that connects to the **`d`** pin of the symbol. so for example **`@pinlist`** is equivalent to **`@@d @@g @@s @@b`**. However using **`@pinlist`** and setting the correct pin ordering in the symbol pins will make netlist generation faster. This is important for very big components with lot of pins, and **`@pinlist`** is the default when symbol is generated automatically (**`Symbol ->Make symbol`** menu of **`<Shift>A`** key).
  The **`format`** attribute may contain a **`@spiceprefix`** string immediately preceding (with no spaces) the **`@name`** attribute.. This will be substituted with value given in instance (example: **`spiceprefix=X`**) but *ONLY* if **`Simulation->Use 'spiceprefix' attribute`** is set. This allows to create different netlists for simulation (example: all MOS are defined as subcircuits) or LVS (no device subcircuits).

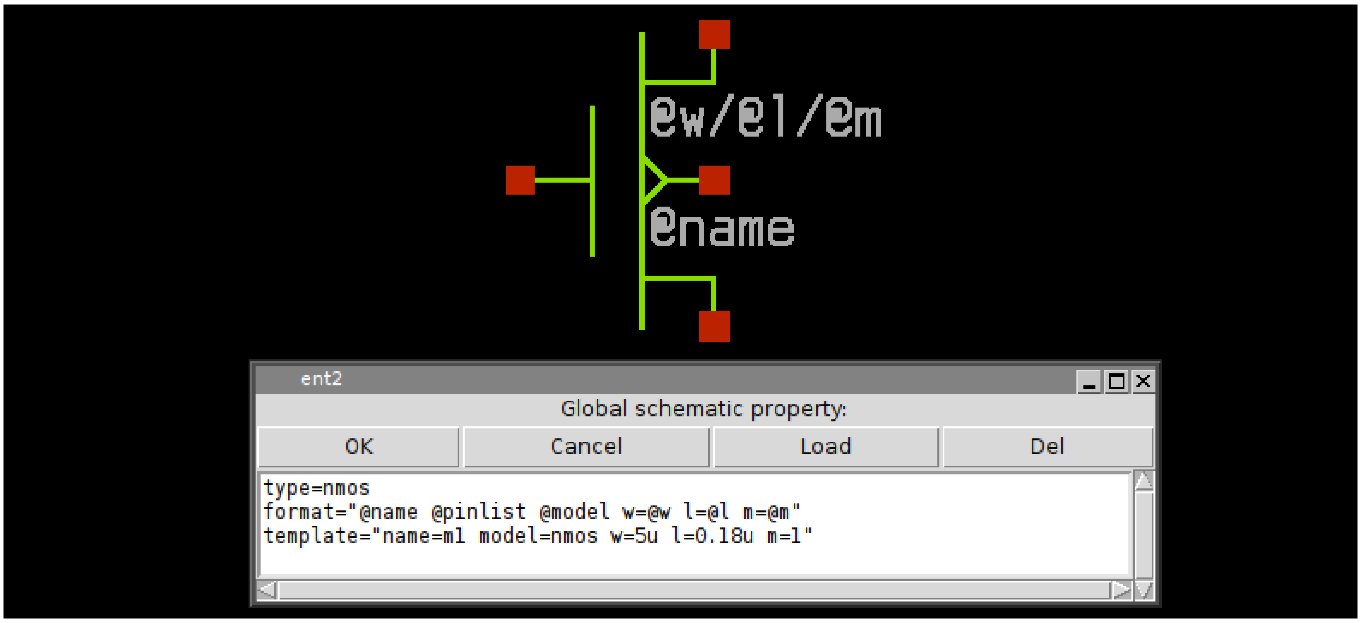

xschem - nmos4.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
D
@wV@lV@m
@spiceprefix@name
Text input
preserve unchanged props
Global schematic property:
OK
Cancel
Load
Del
Edit Attr: <ALL>

```
type=nmos
format="@spiceprefix@name @pinlist @model w=@w l=@l @extra m=@m"
template="name=M1 model=nmos w=5u l=0.18u m=1"
```

SNAP: 10 GRID: 20 NETLIST MODE: symbol mouse = -10 80 - selected: 0 path: .x6.m4.

- **`lvs_format`**: This is the netlisting format attribute that is automatically selected if Xschem is set to produce a LVS netlist (**`Simulation->LVS netlist: top level is a subckt`**). This means that a symbol may have two different attributes for netlisting: **`format`** use dfor spice simulations and **`lvs_format`** for schematic to layout (LVS) comparison. More in general the xschem command **`xschem set format my_format`** will instruct xschem to use **`my_format`** as netlisting rule for components that have this attribute defined. If symbols do not have the **`my_format`** attribute the default fallback (**`format`** for spice netlist) is used.

- **`template`**: Specifies default values for symbol parameters



```
type=nmos
format="@name @pinlist @model w=@w l=@l m=@m"
template="name=m1 model=nmos w=5u l=0.18u m=1"
```

The order these attributes appear in the property string is not important, they can be on the same line or on different lines:

```
type=nmos format="@name @pinlist @model w=@w l=@l m=@m" template="name=m1 model=nmos w=5u l=0.18u r
```

```
format="@name @pinlist @model w=@w l=@l m=@m"
template="name=m1 model=nmos w=5u l=0.18u m=1"
type=nmos
```

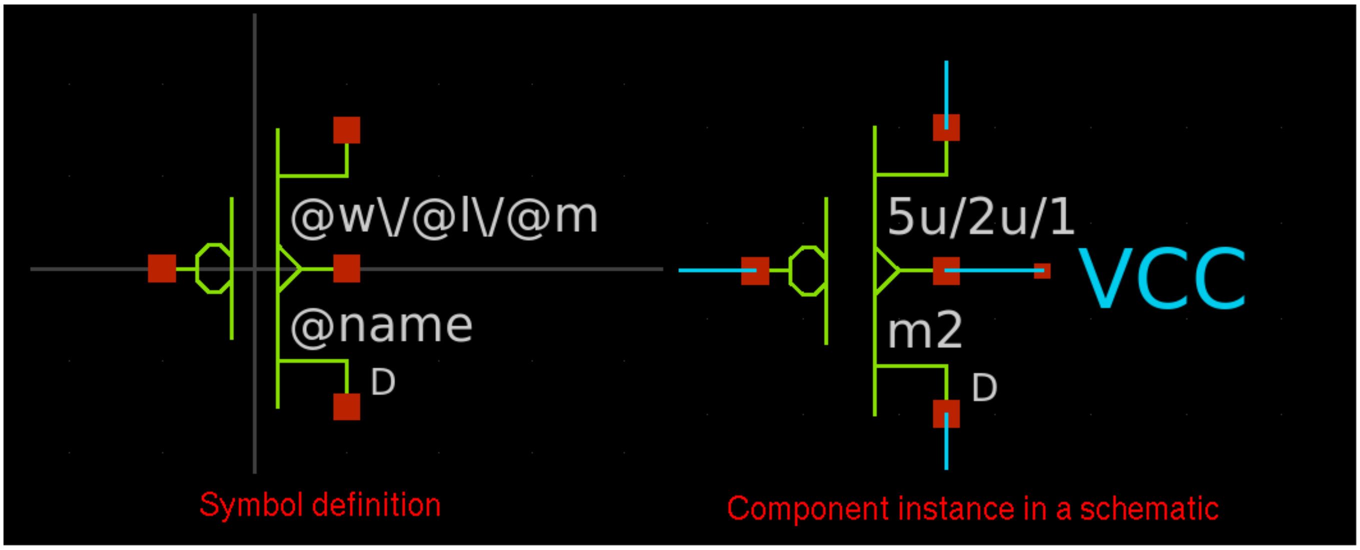
As you see double quotes are used when attribute values have spaces. For this reason if double quotes are needed in an attribute value they must be escaped with backslash `\"`

since the symbol global property string is formatted as a space separated list of **`attribute=value`** items, if a **`value`** has spaces in it it must be enclosed in double quotes, see for example the symbol template attribute: **`template="name=m1 model=nmos w=5u l=0.18u m=1"`** or the the format attribute: **`format="@name @pinlist @model w=@w l=@l m=@m"`**. As a direct consequence a literal double quote in property strings must be escaped (`\"`)

ATTRIBUTE SUBSTITUTION

XSCHEM uses a method for attribute substitution that is very similar to shell variable expansion done with the **`$`** character (for example **`$HOME --> /home/user`**) The only difference is that XSCHEM uses the `'@'` character. The choice of '@' vs '$' is simply because in some simulation netlists shell variables are passed to the simulator for expansion, so to avoid the need to escape the '$' in property strings a different and less used character was chosen.
A literal `@` must be escaped to prevent it to be interpreted as the start of a token to be substituted (`\@`). If a non space character (different than `@`) ends a token it must be escaped. Attribute substitution with values defined in instance attributes takes place in symbol format attribute and in every text, as shown in below picture.

In recent xschem versions a `%` prefixed attribute (example: **`%var`**) can be used instead of a @ prefix. The only difference is that if no matching attribute is defined in instance the **`%var`** resolves to **`var`** instead of an empty string.



If no matching attribute is defined in instance (for example we have **`@W`** in symbol and no **`W=...`** in instance) the **`@W`** string is substituted with an empty string.

OTHER PREDEFINED SYMBOL ATTRIBUTES

- **`vhdl_ignore`**
- **`spice_ignore`**
- **`verilog_ignore`**
- **`tedax_ignore`**

  These 4 attributes tell XSCHEM to ignore completely all instances of the symbol in the respective netlist formats. Allowed values for these attributes are **`true`** (or **`open`**), **`false`** and **`short`** If **`short`** is specified all symbol instances will short together all their pins. For this to work only one of the nets connected to the symbol may have a net label attached to it. All other nets will take this name. If more labeled nets connect to the shorted symbol a net short error is reported. Shorted symbol instances are displayed in the pin color (red) layer. See some images in the component properties man page when describing the same instance based attributes.
  Disabled symbols (**`spice_ignore=true`** or **`spice_ignore=open`**) are displayed in grey.

- **`lvs_ignore`**

  This attribute works in the same way as above attributes, may take the values **`true`** (or **`open`**), **`false`** or **`short`**, and will affect the symbol behaviour in the same way, but only if tcl variable **`lvs_ignore`** is set to **`1`**. This can be done in the Simulation menu: **`Set 'lvs_ignore' variable`**. If this **`lvs_ignore`** is set on the symbol it will be shorted / ignored or kept as is depending on its **`lvs_ignore`** attribute and will be effective in all netlisting formats. This is mostly used to modify the produced netlist automatically when doing schematic vs layout (LVS) comparison.

- **`vhdl_stop`**
- **`spice_stop`**
- **`verilog_stop`**
- **`tedax_stop`**

These 4 attributes will avoid XSCHEM to descend into the schematic representation of the symbol (if there is one) when building the respective netlist format. For example, if an analog block has a schematic (.sch) file describing the circuit that is meaningless when doing a VHDL netlist, we can use a **`vhdl_stop=true`** attribute to avoid descending into the schematic. Only the global property of the schematic will be netlisted. This allows to insert some behavioral VHDL code in the global schematic property that describes the block in a way the VHDL simulator can understand.

- **`spice_primitive`**
- **`vhdl_primitive`**
- **`verilog_primitive`**

Same as above **`_stop`** attributes, but in this case the schematic subcircuit is completely ignored, only the 'format' string is dumped to netlist. No component/entity is generated in vhdl netlist, no module declaration in verilog, no .subckt in spice, no schematic global attributes are exported to netlist.

- **`default_schematic`**

If set to **`ignore`** xschem will not descend into the symbol associated schematic and will not complain if this schematic does not exists. To descend into a schematic instances must specify a **`schematic`** attribute, otherwise no descending and expansion occurs.

- **`spice_sym_def`**
- **`verilog_sym_def`**
- **`vhdl_sym_def`**

If any of these attributes are present and not empty and the symbol type is set to **`subcircuit`** the corresponding netlister will ignore the schematic subcircuit and dump into the netlist the content of this attribute. The typical usage is to include a file, example:

```
verilog_sym_def="tcleval(`include \"[abs_sym_path verilog_include_file.v]\")"
```

For spice netlists if **`@pinlist`** is used in format string and **`spice_sym_def`** attribute is defined the port order will be derived from the subcircuit referenced by the **`spice_sym_def`** attribute.

In this example a **`verilog_include_file.v`** is included using the verilog **`` `include ``** directive. In order to generate a full path for it the **`abs_sym_path`** TCL function is used that searches for this file in any of the **`XCHEM_LIBRARY_PATH`** directories. Since TCL is used the attribute is wrapped into a tcleval(...),
The following will appear in the generated netlist:

```
// expanding   symbol:  verilog_include.sym # of pins=3
// sym_path: /home/schippes/.xschem/xschem_library/verilog_include.sym
`include "/home/schippes/.xschem/xschem_library/verilog_include_file.v"
```

- **`highlight`**

If set to **`true`** the symbol will be highlighted when one of the nets attached to its pins are highlighted.

- **`net_name`**

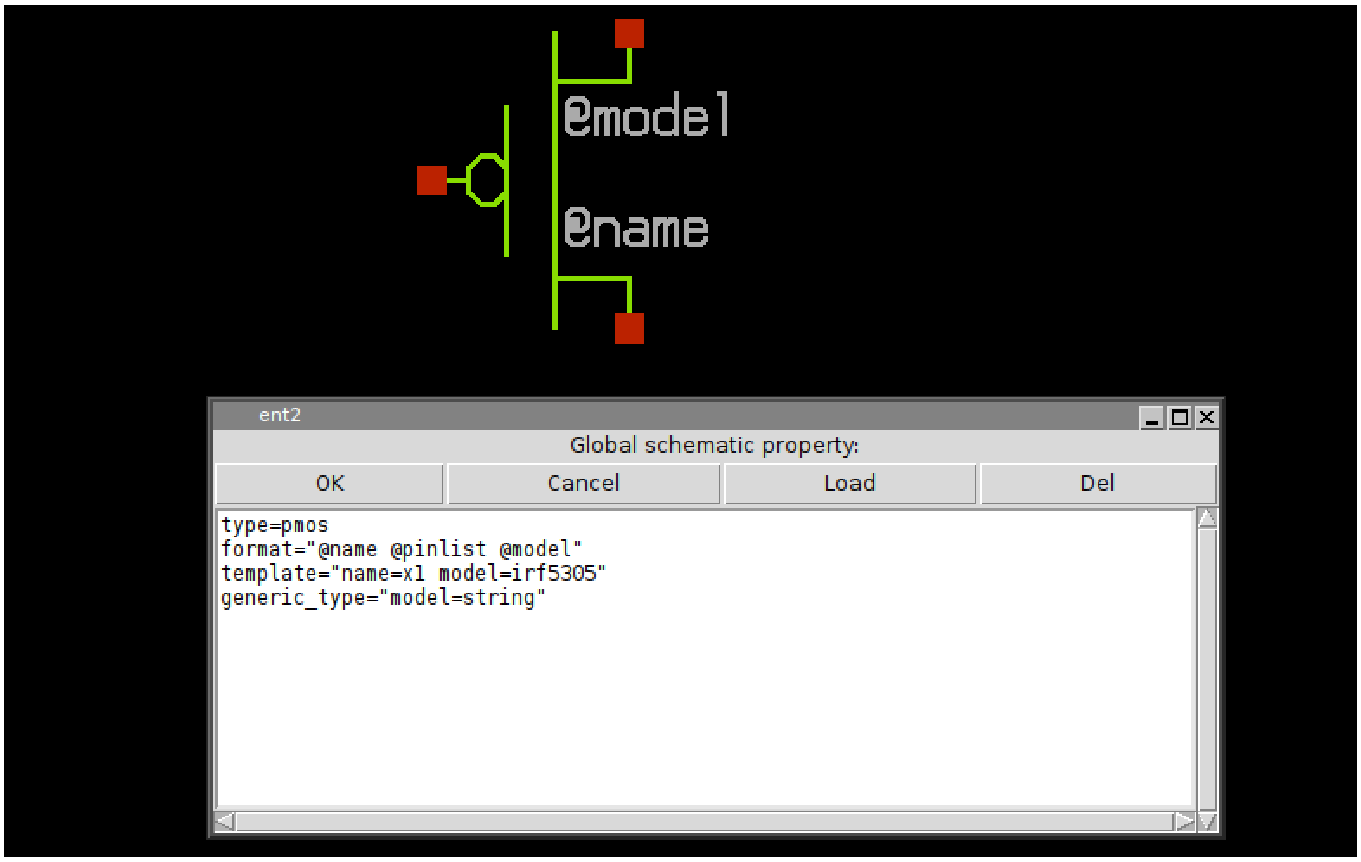
If set to **`true`** the **`#n:net_name`** symbol attributes will display the net names attached to pin terminals. the **`n`** is a pin number or name.

- **place**

This attribute is only useable in **netlist_commands** type symbols (**netlist.sym, code.sym, ...**) if set to **end** it tells XSCHEM that the component instance of that symbol must be netlisted at the end, after all the other elements. This is sometimes needed for SPICE commands that must given at the end of the netlist. This will be explained more in detail in the netlisting slide.
The **place=header** attribute is only valid only for netlist_commands type symbols and spice netlisting mode, it tells XSCHEM that this component must be netlisted in the very first part of a spice netlist. This is necessary for some spice commands that need to be placed before the rest of the netlist.

- **generic_type**

generic_type defines the type of parameters passed to VHDL/Verilog components. Consider the following MOS symbol definition; the **model** attribute is declared as **string** and it will be quoted in VHDL netlists.



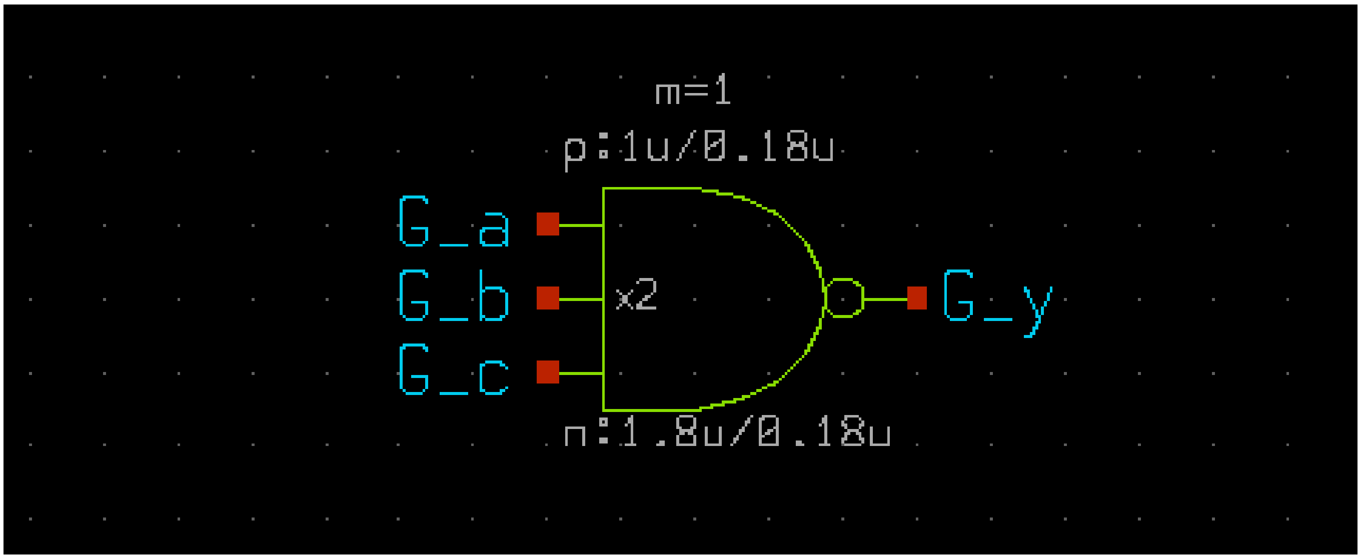
```
type=pmos
format="@name @pinlist @model"
template="name=x1 model=irf5305"
generic_type="model=string"
```

the resulting netlist is shown here, note that without the **generic_type** attribute the **irf5305** string would not be quoted.

```
entity test2 is
end test2 ;

architecture arch_test2 of test2 is
signal d : std_logic ;
signal s : std_logic ;
signal g : std_logic ;
begin
x3 : pmos3
generic map (
   model => "irf5305"
)
port map (
   d => d ,
   g => g ,
   s => s
);

end arch_test2 ;
```

- **extra**

This property specifies that some parameters defined in the **format** string are to be considered as additional pins. This allows to realize inherited connections, a kind of hidden pins with connections passed as parameters. Example of a symbol definition for the following cmos gate:



the symbol property list defines 2 extra pins , VCCPIN and VSSPIN that can be assigned to at component instantiation. The **extra** property tells XSCHEM that these 2 parameters are connection pins and not parameters and thus must not be declared as parameters in the .subckt line in a spice netlist:

```
type=subcircuit
vhdl_stop=true
format="@name @pinlist @VCCPIN @VSSPIN @symname wn=@wn ln=@ln wp=@wp lp=@lp m=@m"
template="name=x1 m=1
+ wn=30u ln=2.4u wp=20u lp=2.4u
+ VCCPIN=VCC VSSPIN=VSS"
extra="VCCPIN VSSPIN"
generic_type="m=integer wn=real ln=real wp=real lp=real VCCPIN=string VSSPIN=string"
verilog_stop=true
```

with these definitions the above schematic will be netlisted as:

```
**.subckt proval
x2 G_y G_a G_b G_c VCC VSS lvnand3 wn=1.8u ln=0.18u wp=1u lp=0.18u m=1
**.ends
* expanding symbol: customlogicLib/lvnand3 # of pins=4
.subckt lvnand3 y a b c VCCPIN VSSPIN wn=30u ln=2.4u wp=20u lp=2.4u
*.opin y
*.ipin a
```

```
*.ipin b
*.ipin c
m1 net2 a VSSPIN VSSPIN nlv w=wn l=ln geomod=0 m=1
m2 y a VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm2 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m3 y b VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm3 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m6 y c net1 VSSPIN nlv w=wn l=ln geomod=0 m=1
m4 y c VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm4 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m5 net1 b net2 VSSPIN nlv w=wn l=ln geomod=0 m=1
.ends
```

Without the **`extra`** property in the cmos gate symbol the following incorrect netlist will be produced:

```
**.subckt prova1
x2 G_y G_a G_b G_c VCC VSS lvnand3 wn=1.8u ln=0.18u wp=1u lp=0.18u m=1
**** begin user architecture code
**** end user architecture code
**.ends


* expanding symbol: customlogicLib/lvnand3 # of pins=4


.subckt lvnand3 y a b c wn=30u ln=2.4u wp=20u lp=2.4u VCCPIN=VCC VSSPIN=VSS
*.opin y
*.ipin a
*.ipin b
*.ipin c
m1 net2 a VSSPIN VSSPIN nlv w=wn l=ln geomod=0 m=1
m2 y a VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm2 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m3 y b VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm3 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m6 y c net1 VSSPIN nlv w=wn l=ln geomod=0 m=1
m4 y c VCCPIN VCCPIN plv w=wp l=lp geomod=0 m=1
dxm4 0 VCCPIN dnwell area='(wp + 57u)*(lp + 31u)' pj='2*(wp +57u)+2*(lp +31u)'
m5 net1 b net2 VSSPIN nlv w=wn l=ln geomod=0 m=1
**** begin user architecture code
**** end user architecture code
.ends
```

as you can see the VSSPIN and VCCPIN are listed as parameters in addition as pins in the netlist.

- **`verilog_extra`**

This attribute is similar to the **`extra`** attribute and is used for verilog netlist. Nodes listed in this attribute value will be used as additional pin connections.

the **`extra`** attribute is still used in verilog netlist as a list of attributes NOT to pass as symbol parameters

You may assign the following attributes to an instance: **`name=X1 VPWR=VCC VGND=GND subckt=NOR2_1`** and you want to have VCC and GND connections to the symbol in the Verilog netlist but do not want any of these attributes to be passed as symbol parameters. In this case you set: **`verilog_extra="VPWR VGND"`** and **`extra="VPWR VGND subckt"`** since **`subckt`** is probably a spice attribute and you don't want it in verilog.

- **`verilog_extra_dir`**

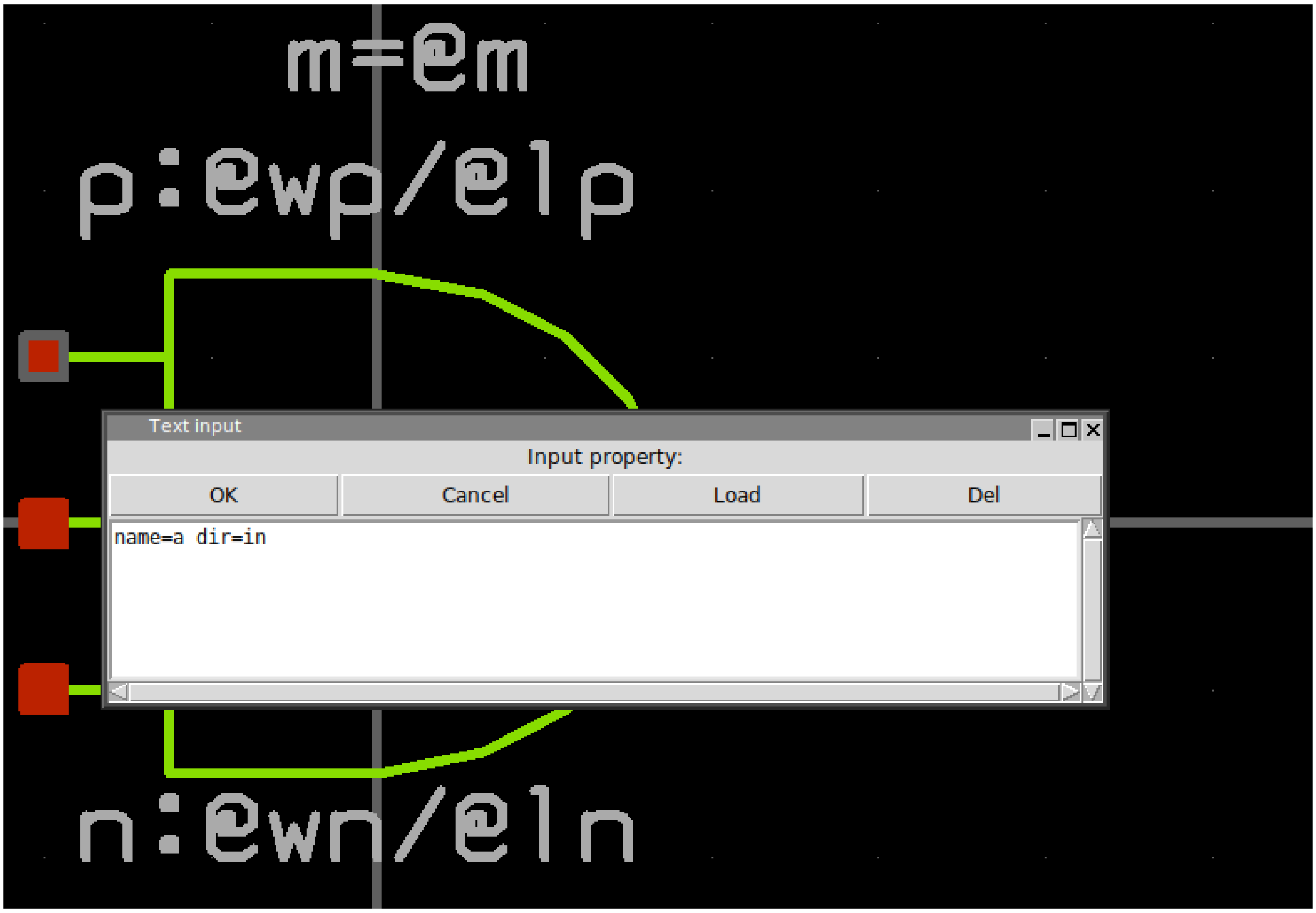
  This attribute allows to define the pin directions of **`verilog_extra`** symbol ports. If unspecified the default is **`inout`**. Allowed values are **`input`**, **`output`**, **`inout`**.
  Example: **`verilog_extra_dir="VPWR=input VGND=input"`**

- **`verilogprefix`**

  If this attribute is defined in symbol it will be used as a prefix to the symbol name and subcircuit expansion in verilog netlists.

- **`dir`**

  Defines the direction of a symbol pin. Allowed values are **`in`**, **`out`**, **`inout`**.

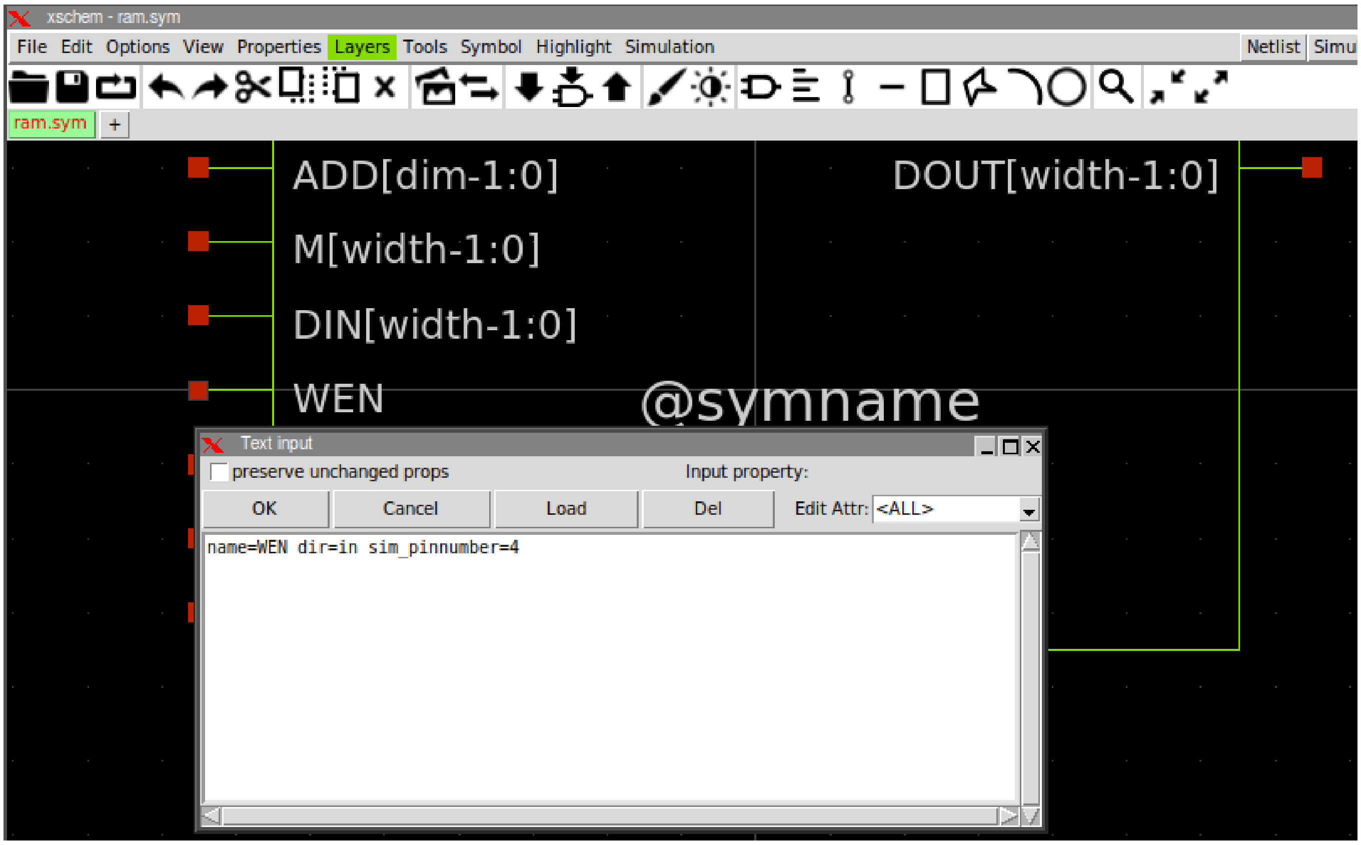


```
name=a dir=in
```

- **`pinnumber`**
  For packaged devices (tEDAx netlists) : indicate the position of the pin on the package. This can be overriden at instance level by attributes **`pinnumber(name)`** set in the instance for tEDAx netlists.

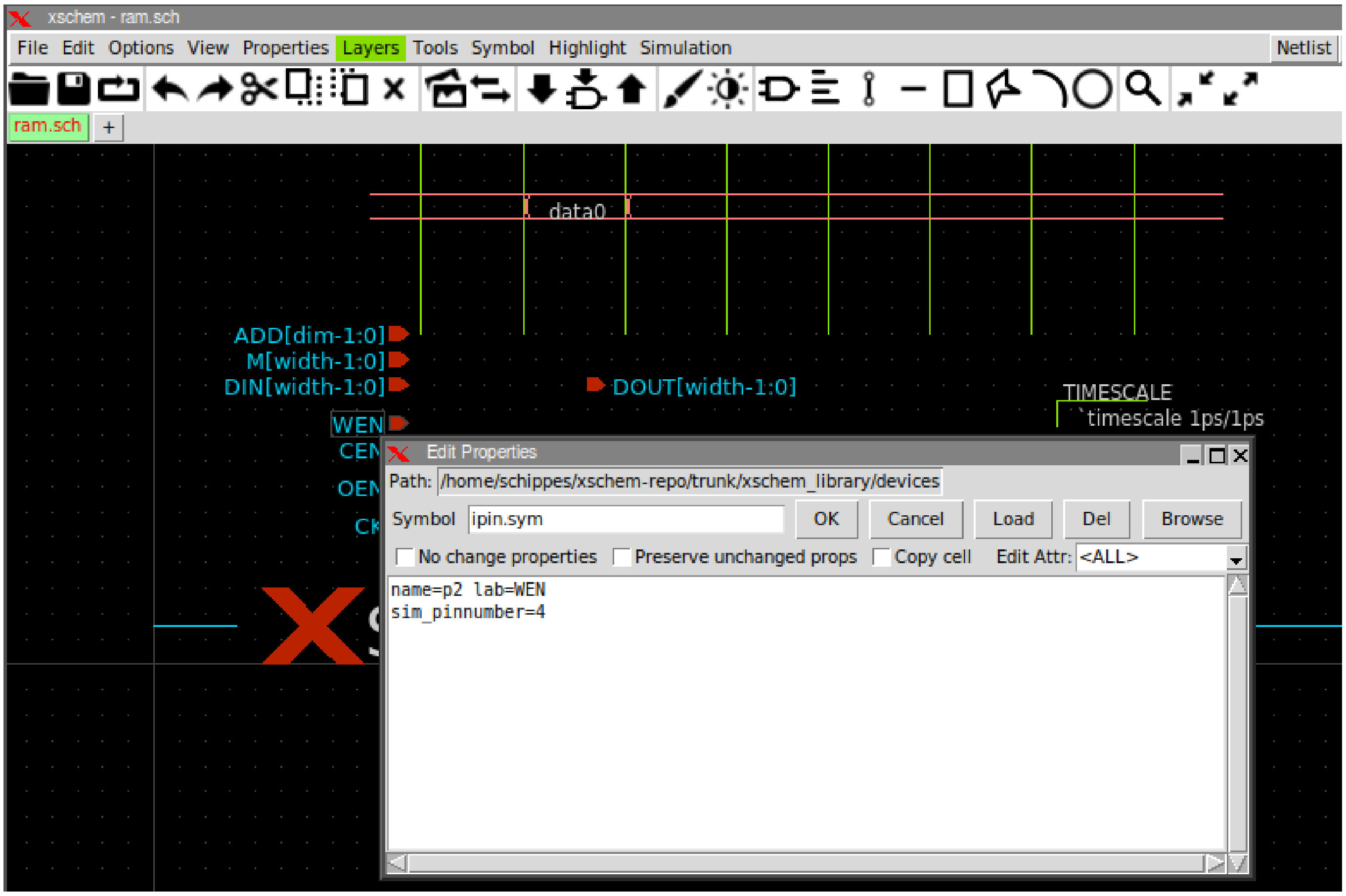
- **`sim_pinnumber`**

  For VHDL, SPICE, Verilog netlists: define the ordering of symbol ports in netlist. If all symbol pins have a sim_pinnumber attribute this symbol will be netlisted (in all netlist formats) with pins sorted in ascending order according to sim_pinnumber value. Start value of sim_pinnumber does not matter (may start at 1 or 0) , it is used as the sort key. You can assign the sim_pinnumber attribute directly in the symbol...



```
name=WEN dir=in sim_pinnumber=4
```

... Or you can assign these in the schematic pins, if you use the "Make symbol from schematic" function ('a' key) these attributes will be transferred to the symbol.
The sim_pinnumber attributes that determine the netlist port ordering are those defined in the symbol.



Path: /home/schippes/xschem-repo/trunk/xschem_library/devices

Symbol ipin.sym

```
name=p2 lab=WEN
sim_pinnumber=4
```

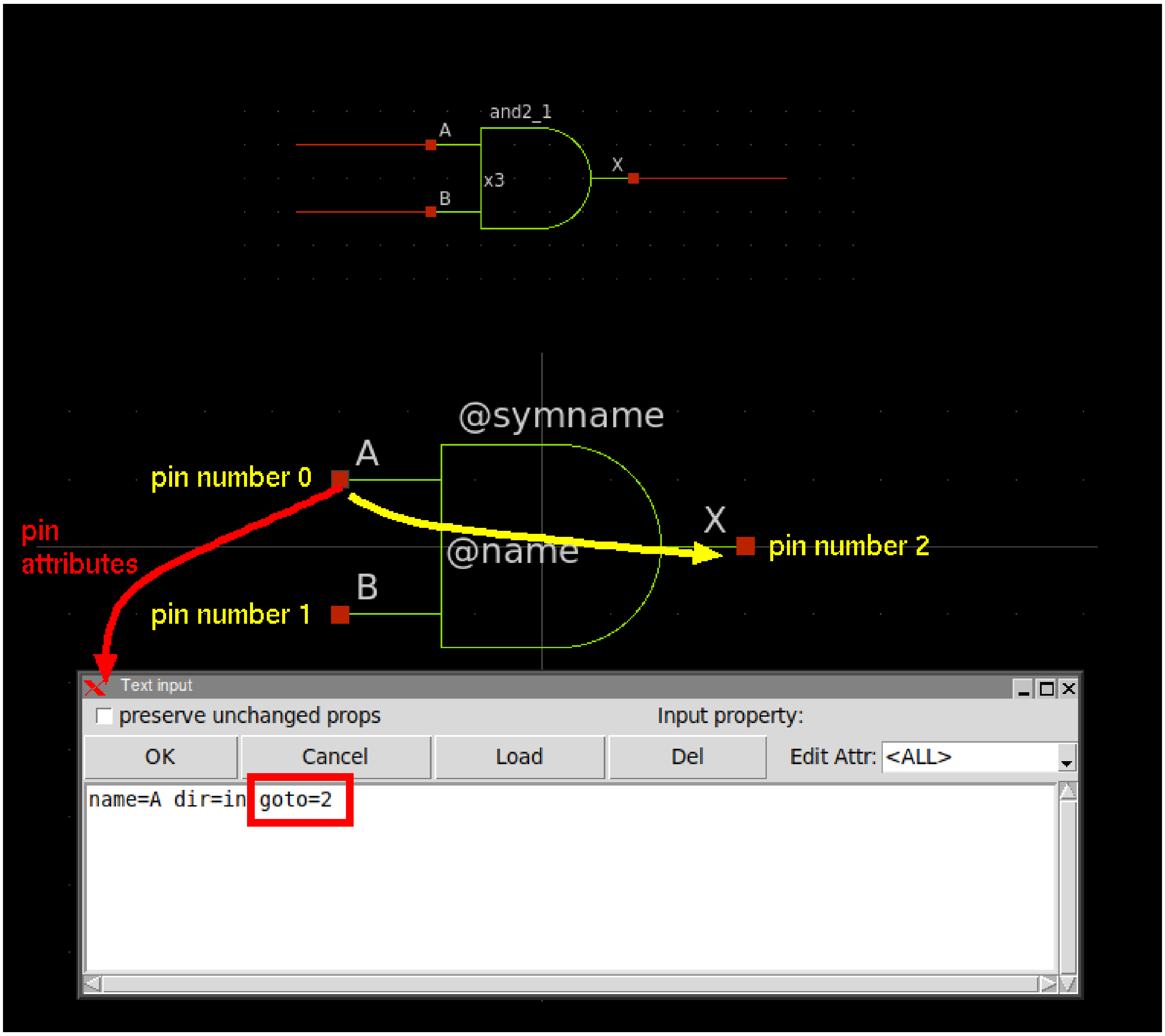
For sorting to happen all symbol pins must have a sim_pinnumber attribute. If some pins miss this attribute no sorting is done and pin ordering will be unchanged, the stored order of symbol pins will be used (first created pin netlisted first).
If there are duplicate sim_pinnumber attributes (but all pins have this attribute) sorting will happen but relative ordering or pins with identical sim_pinnumber is undefined.
As an example you may give sim_pinnumber=9999 on a symbol output and sim_pinnumber=1 on all other pins if you only require the output pin to be netlisted at the end, and don't care about the other pin ordering.

- **`propag=n`**

This attribute instructs xschem to do a 'propagate highlight' from the pin with this attribute to the pin **`n`**. The number 'n' refers to the pin sequence number (do a **`shift-S`** after selecting destination pin to know this information).

- **`goto=n[,m,...]`**

This attribute is used in the xschem embedded digital simulation engine: propagate logic simulation to the output pins **`n, [m, ...]`**. The logic function is defined via the 'function**n**' global attribute. There is one 'function**n**' for each **`n`** output pin. see 'function**n**' attribute for more info.



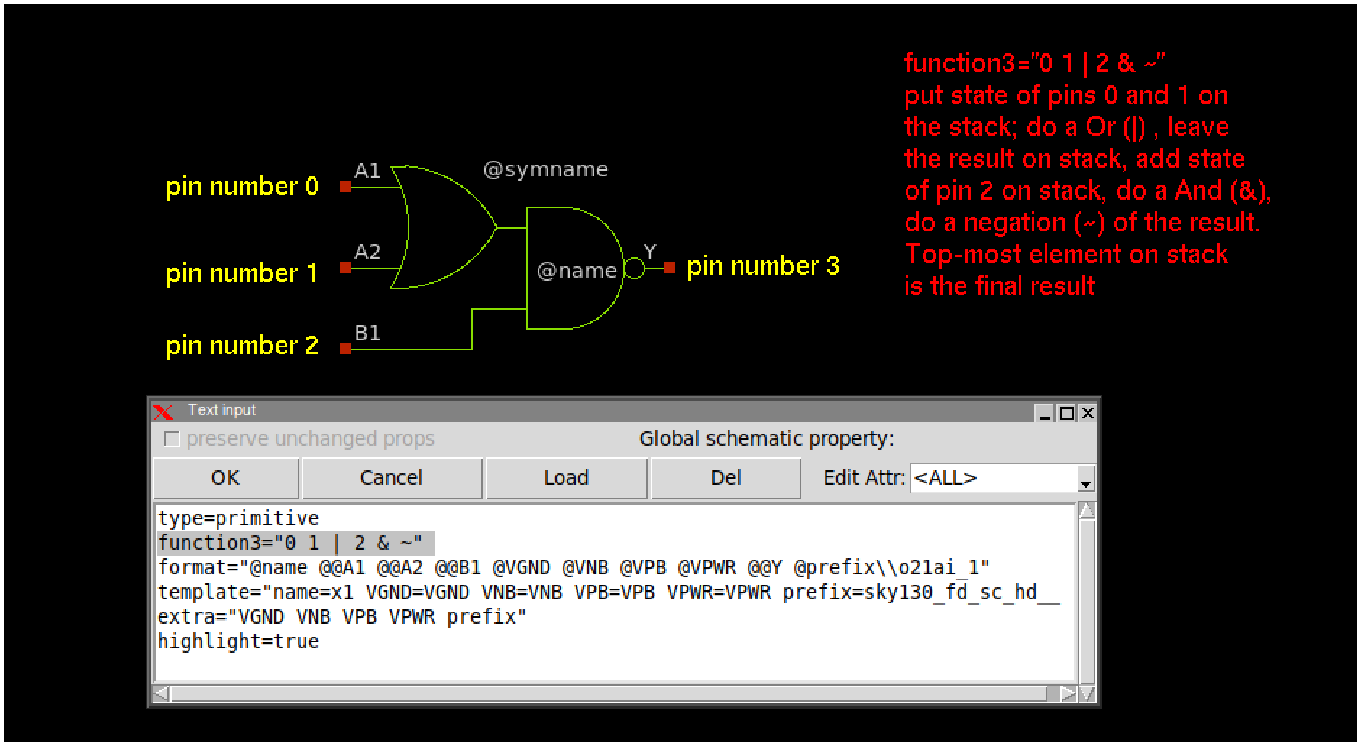
```
name=A dir=in goto=2
```

- **`clock=n`**

  A **`clock`** attribute defined on input pins add some information on the pin function as follows:

  - **`clock=0`** This indicates an 'active low' clock signal for flip-flops
  - **`clock=1`** This indicates an 'active high' clock signal for flip-flops
  - **`clock=2`** This indicates an 'active low' reset signal for flip-flops
  - **`clock=3`** This indicates an 'active high' reset signal for flip-flops
- **`function`**

  This attribute is set in the symbol global attributes and specifies the logic function to be applied to the associated output pin. The format is: **`function<n>="...logic function..."`** where the number **`<n>`** refers to the sequence number of the output pin (do a 'Shift-S' after selecting the pin to know its sequence number). Multiple functions (function3="...", function4="...") can be defined in case of elements with multiple outputs.



function3="0 1 | 2 & ~"
put state of pins 0 and 1 on the stack; do a Or (|) , leave the result on stack, add state of pin 2 on stack, do a And (&), do a negation (~) of the result. Top-most element on stack is the final result

```
type=primitive
function3="0 1 | 2 & ~"
format="@name @@A1 @@A2 @@B1 @VGND @VNB @VPB @VPWR @@Y @prefix\\o21ai_1"
template="name=x1 VGND=VGND VNB=VNB VPB=VPB VPWR=VPWR prefix=sky130_fd_sc_hd__
extra="VGND VNB VPB VPWR prefix"
highlight=true
```

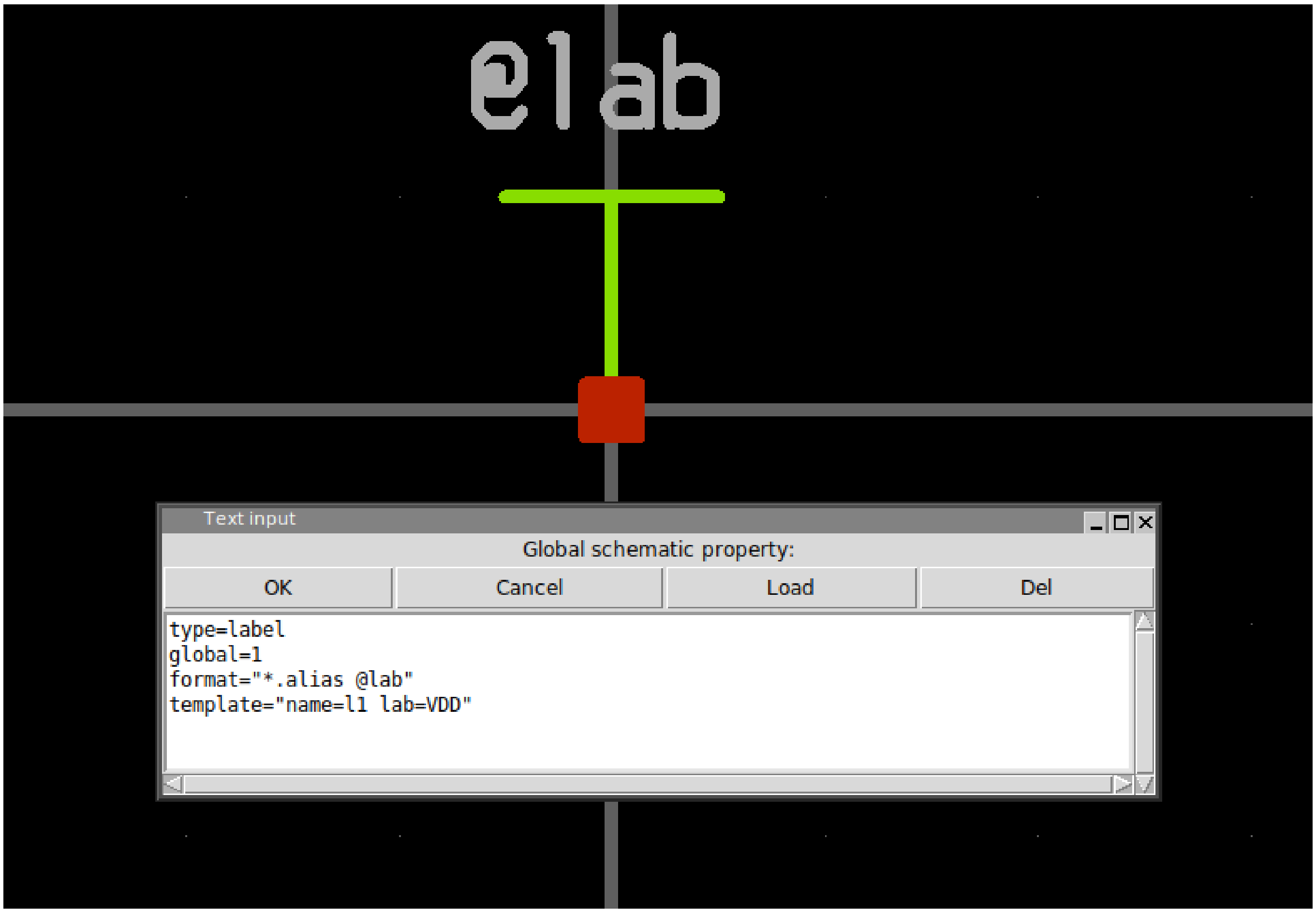
Commands that can appear in functions are:

- **n**: A digit indicates to put on the stack the logic value (0, 1, X) of pin with sequence number **n** The sequence number of a pin my be obtained by clicking the red square of the pin and pressing **Shift-S**.
- **&**: Does a logical AND operation of the last 2 elements on top of the stack, the result is left on the stack
- **|**: Does a logical OR operation of the last 2 elements on top of the stack, the result is left on the stack
- **^**: Does a logical XOR operation of the last 2 elements on top of the stack, the result is left on the stack
- **~**: Does a logical Negation operation of the last element on top of the stack, the result is left on the stack
- **M**: preceeded by 3 element 'a', 'b', 'm', return 'a' if 'm' == 0, 'b' if 'm'==1, else 'X'
- **m**: same as above, but don't update if 'm' not 1 or 0. Used to avoid deadlocks.
- **z**: preceeded by 2 elements, 'a', 'e', return 'a' if 'e' == 1 else Z (hi-Z)
- **d**: Duplicates top element on the stack
- **x**: Exchanges the 2 top elements on the stack
- **r**: Rotate down: bottom element of stack goes to top
- **H**: Puts a Logic '1' on the stack
- **L**: Puts a Logic '0' on the stack
- **Z**: Puts a Logic 'Z' on the stack
- **U**: Puts a Logic 'U' on the stack (do not assign to node)

The remaining value on the stack is the value that is returned and assigned to the output pin.

- **global**

a **global=true** property in a **label** type symbol will declare the corresponding net as 'global'. Global nets in spice netlists are like global variables in a C program, these nets are accessible at any hierarchical level without the need of passing them through pin connections.



```
type=label
global=1
format="*.alias @lab"
template="name=l1 lab=VDD"
```

- **`spice_netlist`**
- **`verilog_netlist`**
- **`vhdl_netlist`**

  If any of these 3 properties if set to **`true`** the symbol will be netlisted in the specified format. This is only valid if the split file netlisting mode is active (**`Options -> Split netlist`**). This is very rarely used but is required in mixed mode simulations, where part of the system will be handled by an analog simulator (spice) and another part of the system by a digital Verilog / VHDL simulator.
- **`verilog_format`**

  This is the Verilog equivalent of the **`format`** property for Spice primitives. This is a valid definition for a 2 input inverted XOR gate:

  ```
  verilog_format="xnor #(@risedel , @falldel ) @name ( @@Z , @@A , @@B );"
  ```

- **`vhdl_format`**

  same as above for VHDL primitives.

- **`tedax_format`**

  same as above for tEDAx netlists.

- **`device_model`**

This attribute contains a SPICE .model or .subckt specification (**`device_model=".model D1N4148 D ...."`**) that will be printed at end of netlist only once for the specified component (D1N4148 in the example).

- **`schematic`**

  This attribute specifies an alternate schematic file to open when descending into the subcircuit:

  ```
  schematic=inv_2.sch
  ```

  It is possible to call a TCL procedure to decide the schematic to descend into:

  ```
  schematic="tcleval([hierarchy_config @symname])"
  ```

  The above **`schematic`** attribute will be evaluated by a **`hierarchy_config`** TCL procedure (which must be defined) and the @symname attribute will be expanded to the name of the symbol before passing the argument to the TCL procedure. This allows user defined schematic selection in the hierarchy to simulate the design at different details/abstraction levels.
  One suggested approach is to define for a given **`opamp_65nm.sym`** symbol several schematics like **`opamp_65nm.sch`**, **`opamp_65nm_pex.sch`**, **`opamp_65nm_aged.sch`**, **`opamp_65nm_empty.sch`**, ... and define some user accessible method in hierarchy_config procedure to select one of these 'switch' schematics.

- **`symbol_ignore`**

  This attribute can be attached to symbol elements, like lines, rectangles, polygons, arcs, texts, wires and instances (in case of lcc symbols). If set to true (**`symbol_ignore=true`**) the corresponding element will not be displayed when the symbol is instantiated.

PREDEFINED SYMBOL VALUES

- **`@symname`**

  This expands to the name of the symbol

- **`@symref`**

  This expands to the symbol reference exactly as specified in the instance (the **`Symbol`** textbox if you edit the symbol attributes with **`q`** key).

- **`@symname_ext`**

  This expands to the name of the symbol, keeping the extension (usually .sym)

- **`@path`**

  This expands to the hierarchy path the symbol instance is placed in (example: **`xcontrol.xdec[3].xinv`**)

- **`@pinlist`**

  This expands to the list of nets that connect to symbol pins in the order they are set in the symbol

- **`@@pin`**

This expands to the net that connect to symbol pin named **`pin`**. This substitution takes place only when producing a netlist (Spice, Verilog, VHDL, tEDAx) so it is allowed to use this value only in **`format`**,**`vhdl_format`**, **`tedax_format`** or **`verilog_format`** attributes (see Netlisting slide)

- **`@#n`**

This expands to the net that connect to symbol pin at position **`n`** in the XSCHEM internal storage. This substitution takes place only when producing a netlist (Spice, Verilog, VHDL, tEDAx) so it is allowed to use this value only in **`format`**,**`vhdl_format`**, **`tedax_format`** or **`verilog_format`** attributes (see Netlisting slide)
This method of accessing a net that connects to a pin is much faster than previous one since XSCHEM does not need to loop through symbol pin names looking for a match.
Example: **`@#2`**: return net name that connects to the third pin of the symbol (position 2).

- **`@#n:pin_attribute`**

This expands to the value or property **`pin_attribute`** defined in the pin at position **`n`** in the XSCHEM internal storage. This method of looking up properties is very fast.
Example: **`@#0:pinnumber`**: This expands to the value of the pinnumber defined in pin object at position 0 in the xschem internal ordering. This format is very useful for slotted devices where the actual displayed pin number depends on the slot information defined in the instance name (example: U1:2, slot number 2 of IC U1). These tokens may be placed as text in the symbol graphic window, not in format strings.

- **`@#pin_name:pin_attribute`**

This expands to the value or property **`pin_attribute`** defined in the pin named **`pin_name`** This method of looking up properties is a bit slower since xschem has to do string matching to find out the pin.
Example: **`@#A:pinnumber`**: This expands to the value of the pinnumber defined in pin **`A`**. This format is very useful for slotted devices where the actual displayed pin number depends on the slot information defined in the instance name (example: U1:2, slot number 2 of IC U1). These tokens may be placed as text in the symbol graphic window, not in format strings.

- **`@#pin_name:net_name`**
- **`@#n:net_name`**

these expand to the net name attached to pin with name **`pin_name`** or with sequence number **`n`**.

- **`@#pin_name:resolved_net`**
- **`@#n:resolved_net`**

these expand to the full hierarchy name of the net attached to pin with name **`pin_name`** or with sequence number **`n`**.

- **`@#pin_name:spice_get_voltage`**
- **`@#n:spice_get_voltage`**

these expand to the voltage of the net attached to the pin with name **`pin_name`** or with sequence number **`n`**, extracted from simulation raw file (operating point or cursor **`b`** position)

- **`@sch_last_modified`**

this indicates the last modification time of the **`.sch`** file of the symbol.

- **`@sym_last_modified`**

  this indicates the last modification time of the **`.sym`** file of the symbol.

- **`@time_last_modified`**

  this indicates the last modification time of the schematic (.sch) **containing** the symbol instance.

- **`@schname_ext`**

  this expands to the name of the schematic (.sch) **containing** the symbol instance.

- **`@schname`**

  this expands to the name of the schematic **containing** the symbol instance, without the extension (no .sch).

- **`@topschname`**

  this expands to the name of the tol level schematic (.sch) **containing** the symbol instance.

- **`@prop_ptr`**

  this expands to the **entire** property string passed to the component.

- **`@schprop`**

  this expands to the **spice** global property string of the schematic containing the symbol

- **`@schvhdlprop`**

  this expands to the **VHDL** global property string of the schematic containing the symbol

- **`@schverilogprop`**

  this expands to the **Verilog** global property string of the schematic containing the symbol

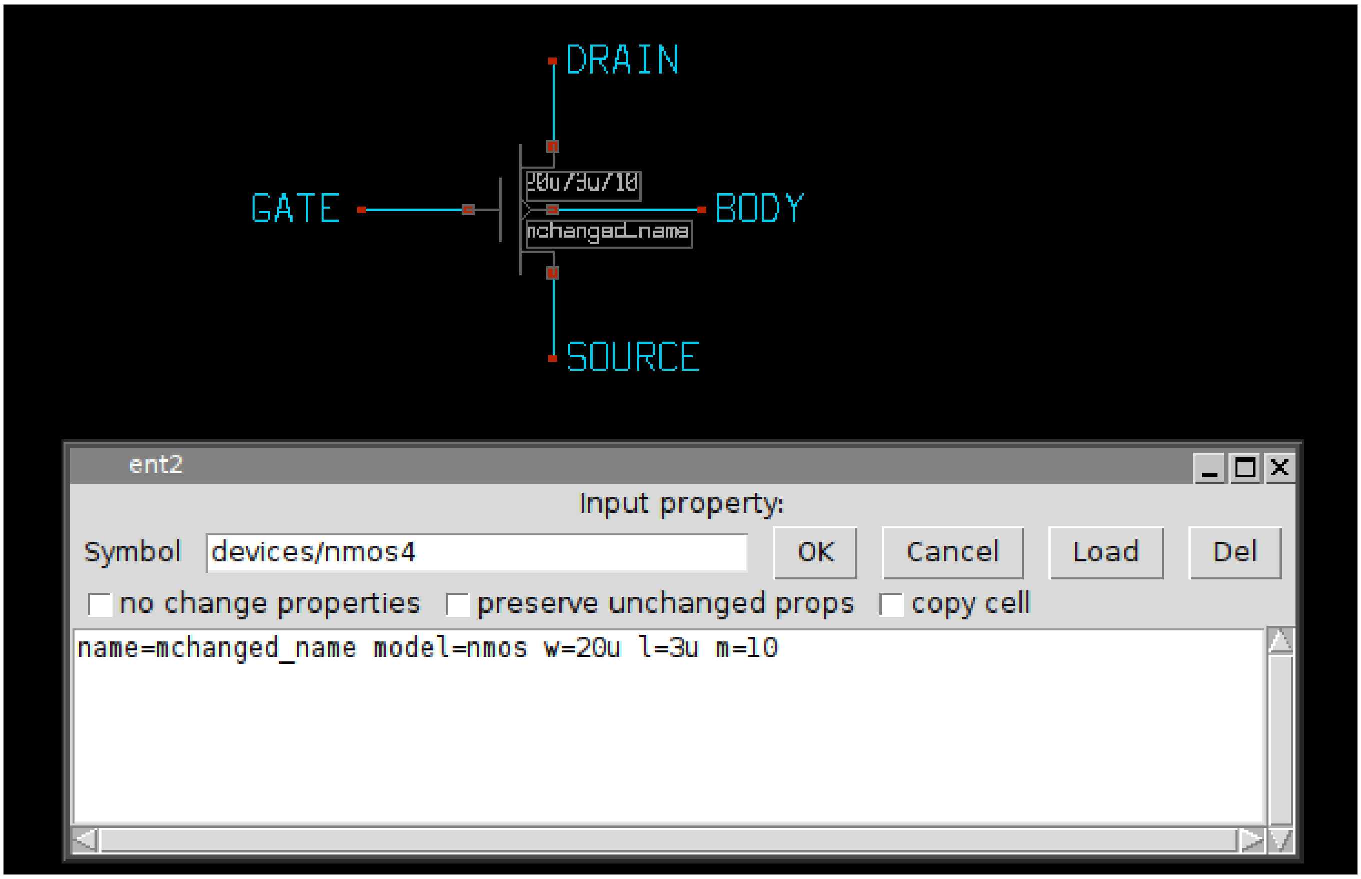
TCL ATTRIBUTE SUBSTITUTION

Any attribute and symbol text can be embedded in a **`tcleval(....)`** construct, the string inside the parentheses will be passed to the tcl interpreter for evaluation. This allows to use any tcl variable/command/expression. Example:
`spice_ignore="tcleval($::ignore_symbol)"`
will cause the symbol to be ignored by the spice netlister if the **`ignore_symbol`** tcl variable is existing and set to **`true`**

PREV UP NEXT

COMPONENT PROPERTY SYNTAX

Component property strings can be set in the usual way with the `'q'` on a selected component instance or by menu **`Properties --> Edit`**



```
name=mchanged_name model=nmos w=20u l=3u m=10
```

The dialog box allows to change the property string as well as the symbol reference. The property string is essentially a list of **`attribute=value`** items. As with symbol properties if a **`value`** has white space it should be double-quoted. The following property definitions are identical:

```
name=mchanged_name model=nmos w=20u l=3u m=10
```

```
name="mchanged_name" model="nmos" w="20u" l="3u" m="10"
```

Given the role of the `"` character, if quoted values are needed escapes must be used, like in the following example where the model name will be with quotes in netlist:

```
name="mchanged_name" model="\"nmos\"" w="20u" l="3u" m="10"
```

or

```
name="mchanged_name" model=\"nmos\" w="20u" l="3u" m="10"
```

the resulting SPICE netlist will be:
`mchanged_name DRAIN GATE SOURCE BODY "nmos" w=20u l=3u m=10`

There is no limit on the number of **`attribute=value`** items, each attribute should have a corresponding **`@attribute`** in the symbol definition format, but this is not a requirement. There are a number of special attributes as we will see later.

Important: a **`name=<inst_name>`** item is mandatory and must be placed in component property string to get a valid netlist, as this is the partname or so-called refdes (reference designator). If **`<inst_name>`** is already used in another component XSCHEM will auto-rename it to a unique name preserving the first letter (which ts a device type indicator for SPICE like netlists).

PREDEFINED COMPONENT ATTRIBUTES

- **`name`**

  This defines the name of the instance. Names are unique, so if for example multiple MOS components are placed in the design one should be named **`m1`** and the second **`m2`** or anything else, provided the names are different. XSCHEM enforces this, unless **`Options -> allow duplicated instance names`** is set. If a name is given that already exist in the current schematic it will be renamed. Normally the template string defines a default name for a given component, and especially for SPICE compatibility, the first character must NOT be changed. For example, the default name for a MOS transistor is **`m1`**, it can be renamed for example to **`mcurr_source`** but not for example to **`dcurr_source`**. XSCHEM does not enforce that the first character is preserved, it's up to the designer to keep it consistent with the component type.

- **`embed`**

  When the **`embed=true`** is set on a component instance the corresponding symbol will be saved into the schematic (.sch) file on the next save operation. This allows to distribute schematic files that contain the used symbols so these will not depend on external library symbols. When this attribute is set on a component instance, all instances in the schematic referring to the same symbol will use the embedded symbol definition. When descending into an embedded symbol, any changes will be local, meaning that no library symbol will be affected. The changes will be saved using the embedded tag (**`[...]`**) into the schematic file. Removing this attribute will revert to external symbols after saving and reloading the schematic file.

- **`url`**

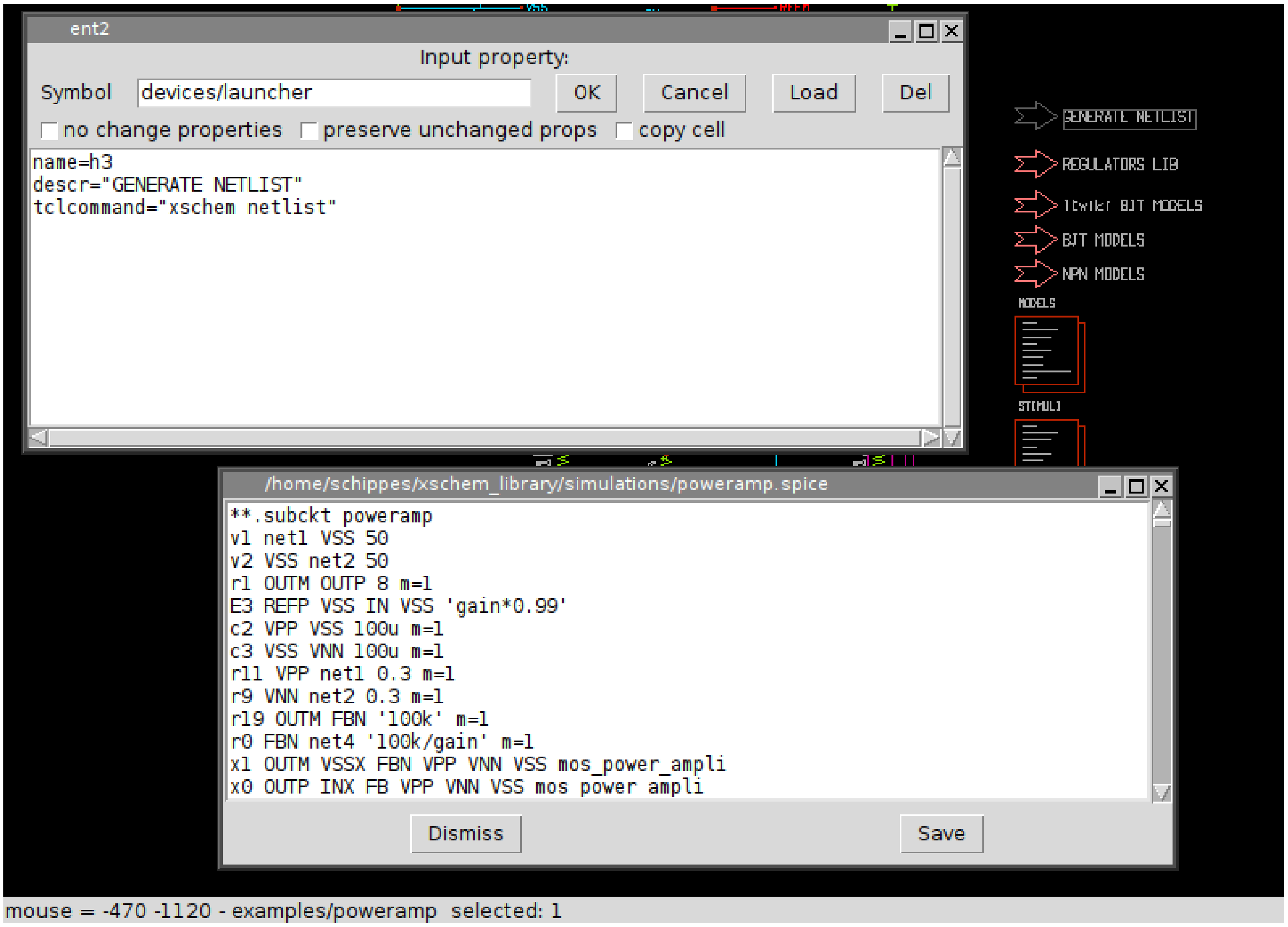
  This attribute defines a location (web page, file) that can be viewed when hitting the **`<shift>H`** key (or **`<Alt> left mouse button`**) on a selected component. This is very useful to link a datasheet to a component, for example. The default program used to open the url is **`xdg-open`**. this can be changed in the **`~/xschemrc`** configuration file with the **`launcher_default_program`** variable. **`url`** can be an http link or a local file that has a default association known to xdg-open.

- **`program`**

this attribute can be used to specify an application to be used to open the **url** link, if the default application has to be changed or the file type is unknown. for example **program=evince** may be given to specify an application for a pdf file specified with **url**

- **tclcommand**

  this can be any tcl statement (or group of statements separated by semicolons) including all xschem-specific commands, the statement will be executed when pressing the **<shift>H** key (or **<Alt> left mouse button**) on the selected instance.
  The **tclcommand** and **url** properties are mutually exclusive.



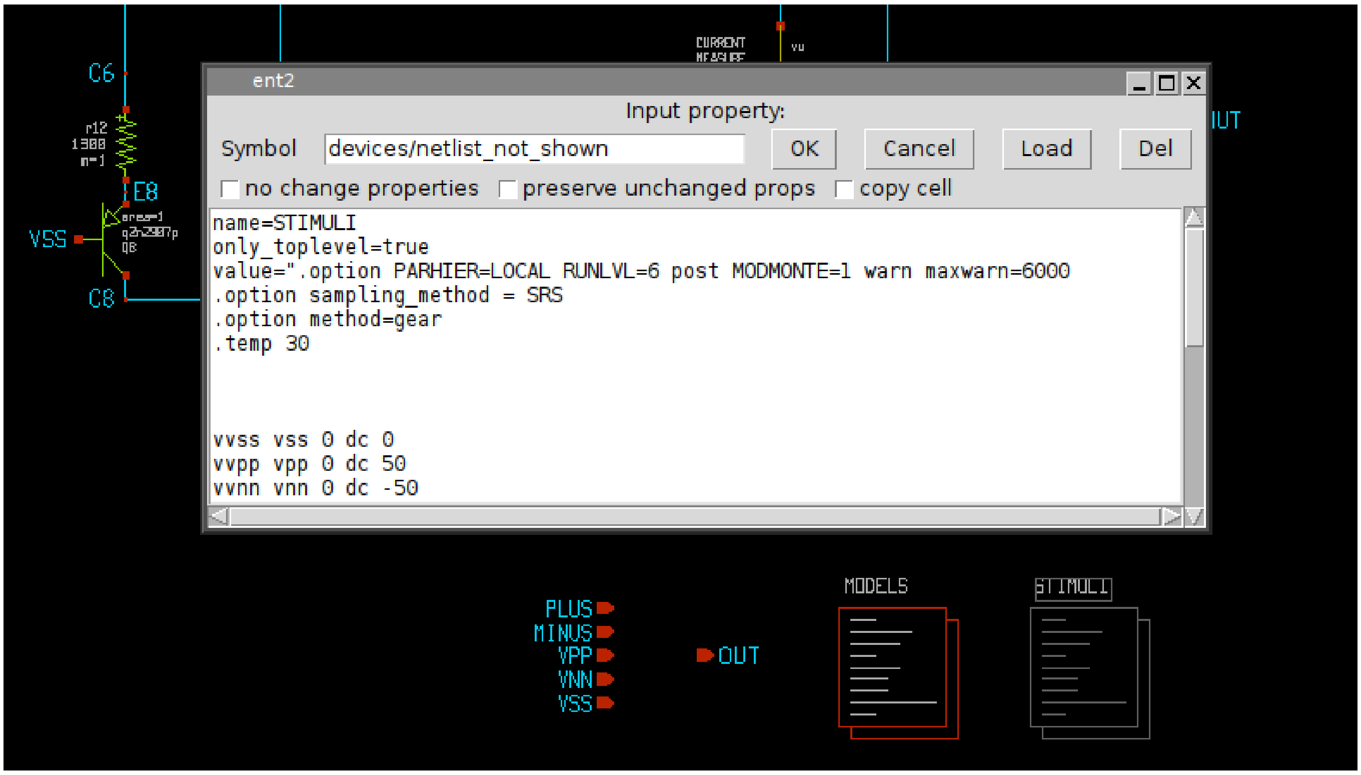
```
name=h3
descr="GENERATE NETLIST"
tclcommand="xschem netlist"
```

/home/schippes/xschem_library/simulations/poweramp.spice

```
**.subckt poweramp
v1 net1 VSS 50
v2 VSS net2 50
r1 OUTM OUTP 8 m=1
E3 REFP VSS IN VSS 'gain*0.99'
c2 VPP VSS 100u m=1
c3 VSS VNN 100u m=1
r11 VPP net1 0.3 m=1
r9 VNN net2 0.3 m=1
r19 OUTM FBN '100k' m=1
r0 FBN net4 '100k/gain' m=1
x1 OUTM VSSX FBN VPP VNN VSS mos_power_ampli
x0 OUTP INX FB VPP VNN VSS mos power ampli
```

- **only_toplevel**

  this attribute is valid only on **netlist_commands** type symbols and specifies that the symbol should be netlisted only if it is instantiated in the top-most hierarchy. This is very useful for spice commands. Spice commands are placed in a special **netlist** component as we will see and are meaningful only when simulating the block, but should be skipped if the component is simulated as part of a bigger system which has its own (at higher hierarchy level) **netlist**component for Spice commands.



Symbol: devices/netlist_not_shown

```
name=STIMULI
only_toplevel=true
value=".option PARHIER=LOCAL RUNLVL=6 post MODMONTE=1 warn maxwarn=6000
.option sampling_method = SRS
.option method=gear
.temp 30



vvss vss 0 dc 0
vvpp vpp 0 dc 50
vvnn vnn 0 dc -50
```

- **`global`**

  A **`global=true`** attribute on instances of **`label`** type symbols (like **`lab_pin.sym`**, **`lab_net.sym`**, **`vdd.sym`**, **`gnd.sym`**) will set the specified node to global in SPICE netlists, adding a **`.GLOBAL`** statement line for the node. This will override symbol **`global=...`** setting if any.

- **`lock`**

  A **`lock=true`** attribute will make the symbol not editable. the only way to make it editable again is to double click on it to bring up the edit attributes dialog box and set to false. This is useful for title symbols.

- **`hide`**

  A **`hide=true`** attribute will only display the symbol bounding box.

- **`hide_texts`**

  A **`hide_texts=true`** attribute will hide all symbol texts.

- **`text_size_<n>`**

  This attribute sets the size of symbol text item number **`n`**. This allows instance based symbol text sizing.

- **`text_layer_<n>`**

  This attribute sets the layer of symbol text item number **`n`**. This allows instance based symbol text color customization.

- **`highlight`**

If set to **`true`** the symbol will be highlighted when one of the nets attached to its pins are highlighted.

- **`net_name`**

  If set to **`true`** the **`#n:net_name`** symbol attributes will display the net names attached to pin terminals. the **`n`** is a pin number or name.

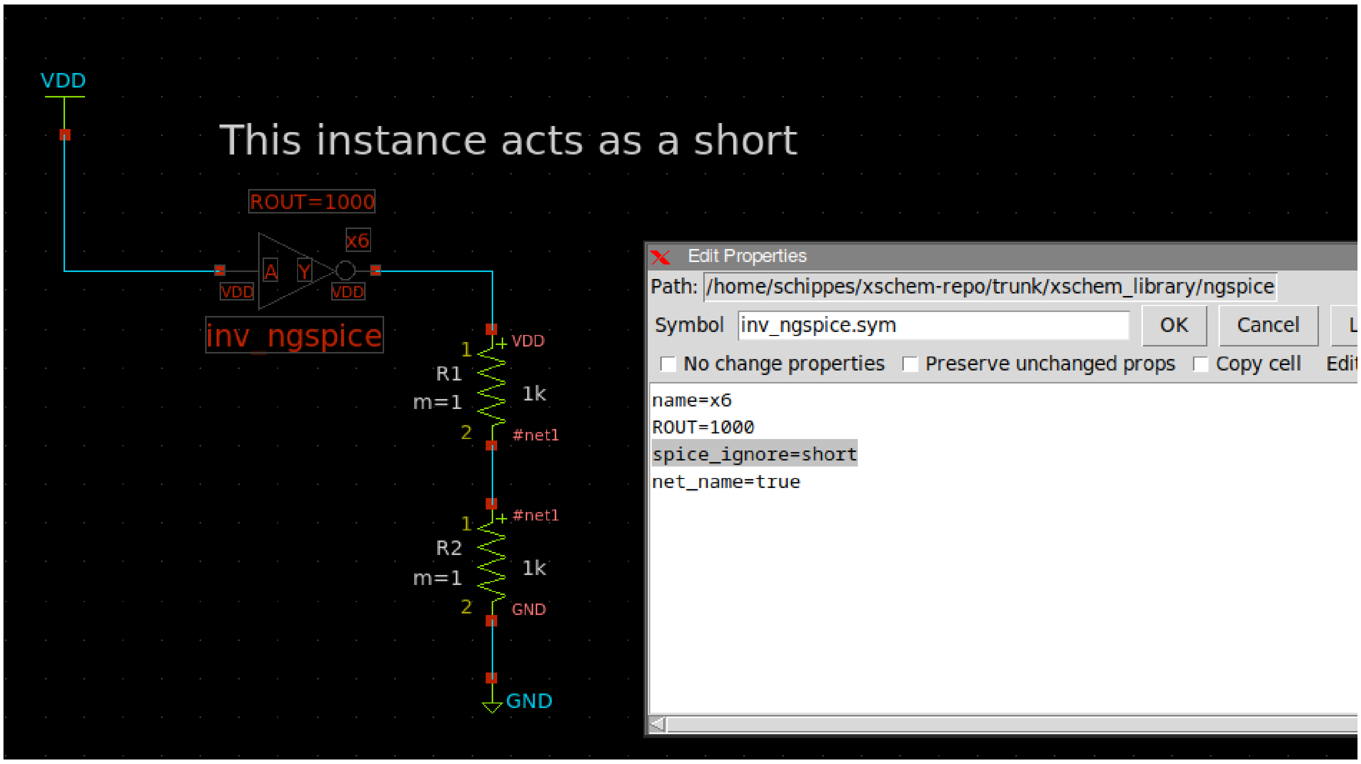
- **`place`**

  The **`place=end`** attribute is only valid only for **`netlist_commands`** type symbols, and tells XSCHEM that this component must be netlisted last. This is necessary for some spice commands that need to be placed **after** the rest of the netlist.

  The **`place=header`** attribute is only valid only for **`netlist_commands`** type symbols and spice netlisting mode, it tells XSCHEM that this component must be netlisted in the very first part of a spice netlist. This is necessary for some spice commands that need to be placed **before** the rest of the netlist.

- **`vhdl_ignore`**
- **`spice_ignore`**
- **`verilog_ignore`**
- **`tedax_ignore`**

  These 4 attributes tell XSCHEM to ignore completely the instance in the respective netlist formats. Allowed values for these attributes are **`true`** (or **`open`**), **`false`** and **`short`** If **`short`** is specified the instance will short together all its pins. For this to work only one of the nets connected to the symbol may have a net label attached to it. All other nets will take this name. If more labeled nets connect to the shorted symbol a net short error is reported. Shorted instances are displayed in the pin color (red) layer. See in below image the upper netname of R1 is **`VDD`**.

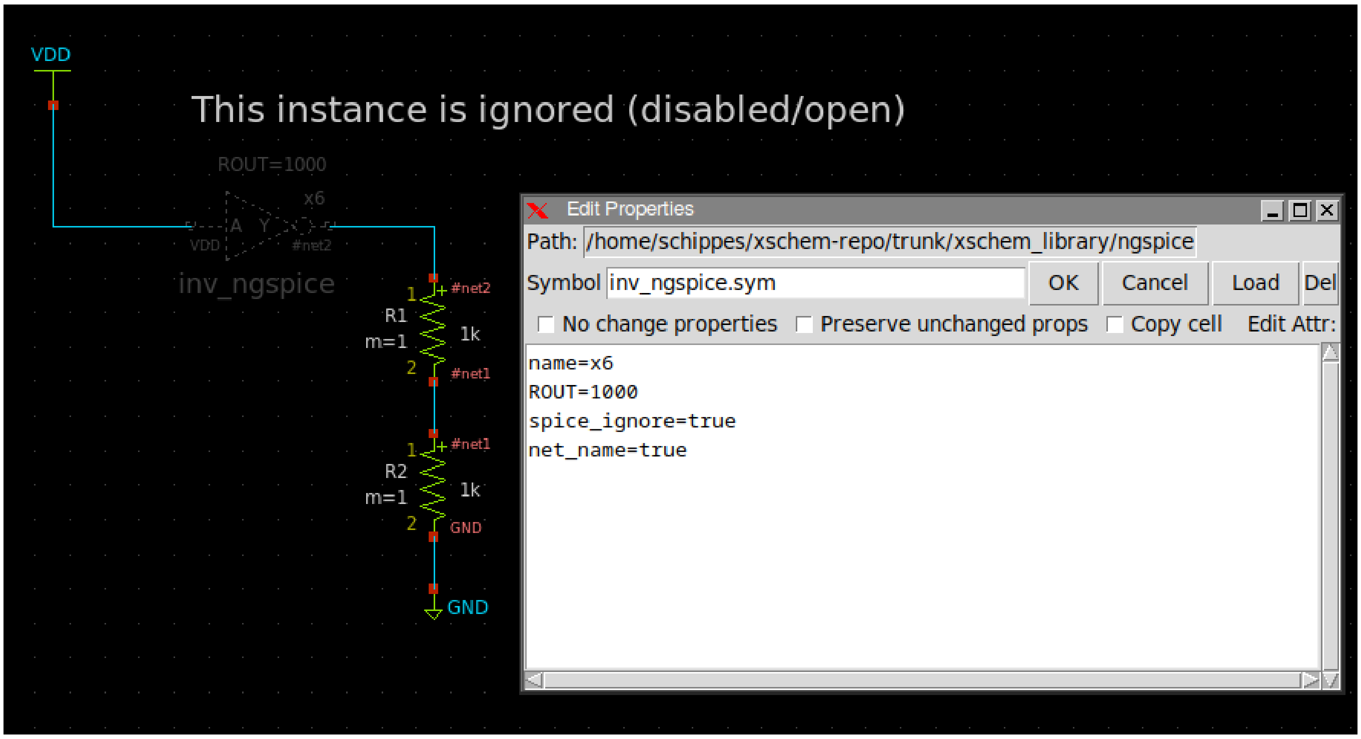


Path: /home/schippes/xschem-repo/trunk/xschem_library/ngspice

Symbol inv_ngspice.sym

```
name=x6
ROUT=1000
spice_ignore=short
net_name=true
```

Disabled components (**`spice_ignore=true`** or **`spice_ignore=open`**) are displayed in grey.

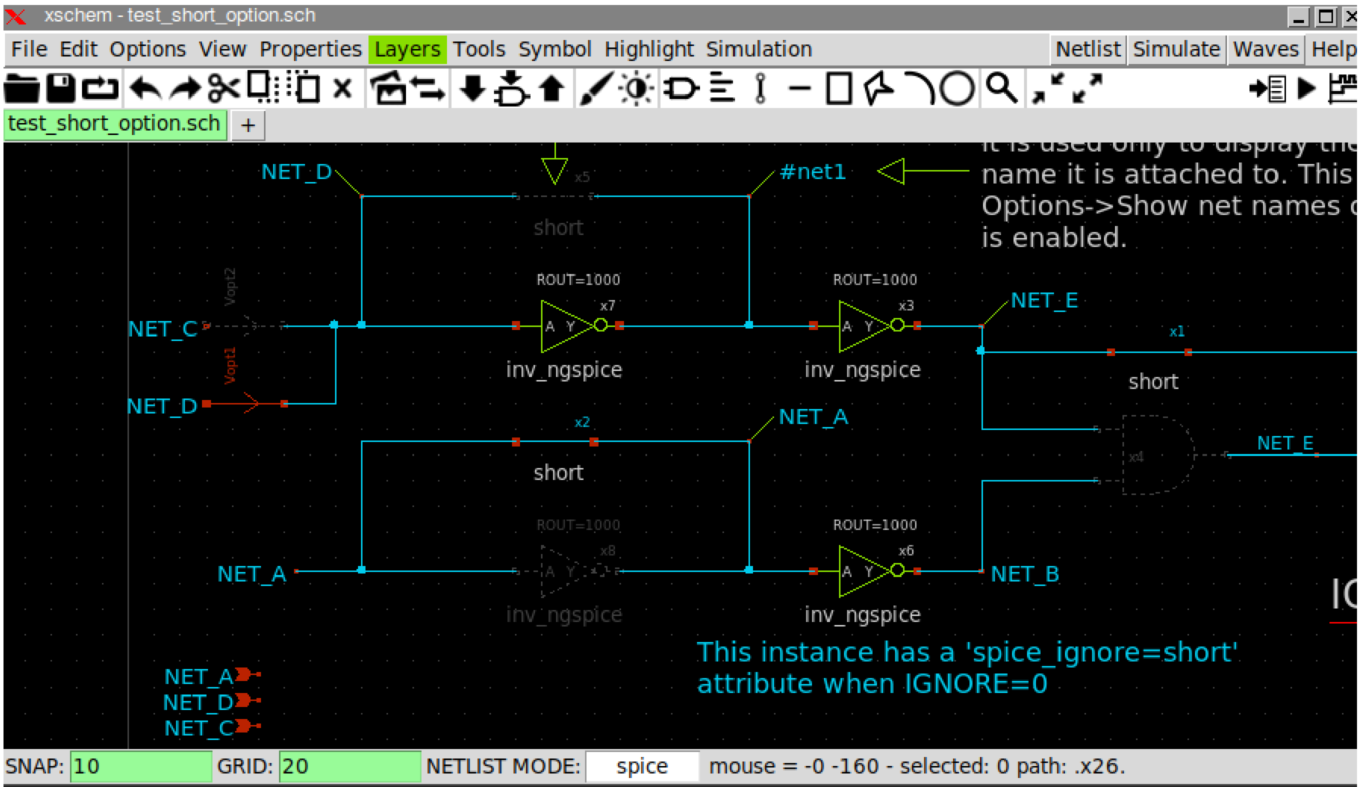


```
name=x6
ROUT=1000
spice_ignore=true
net_name=true
```

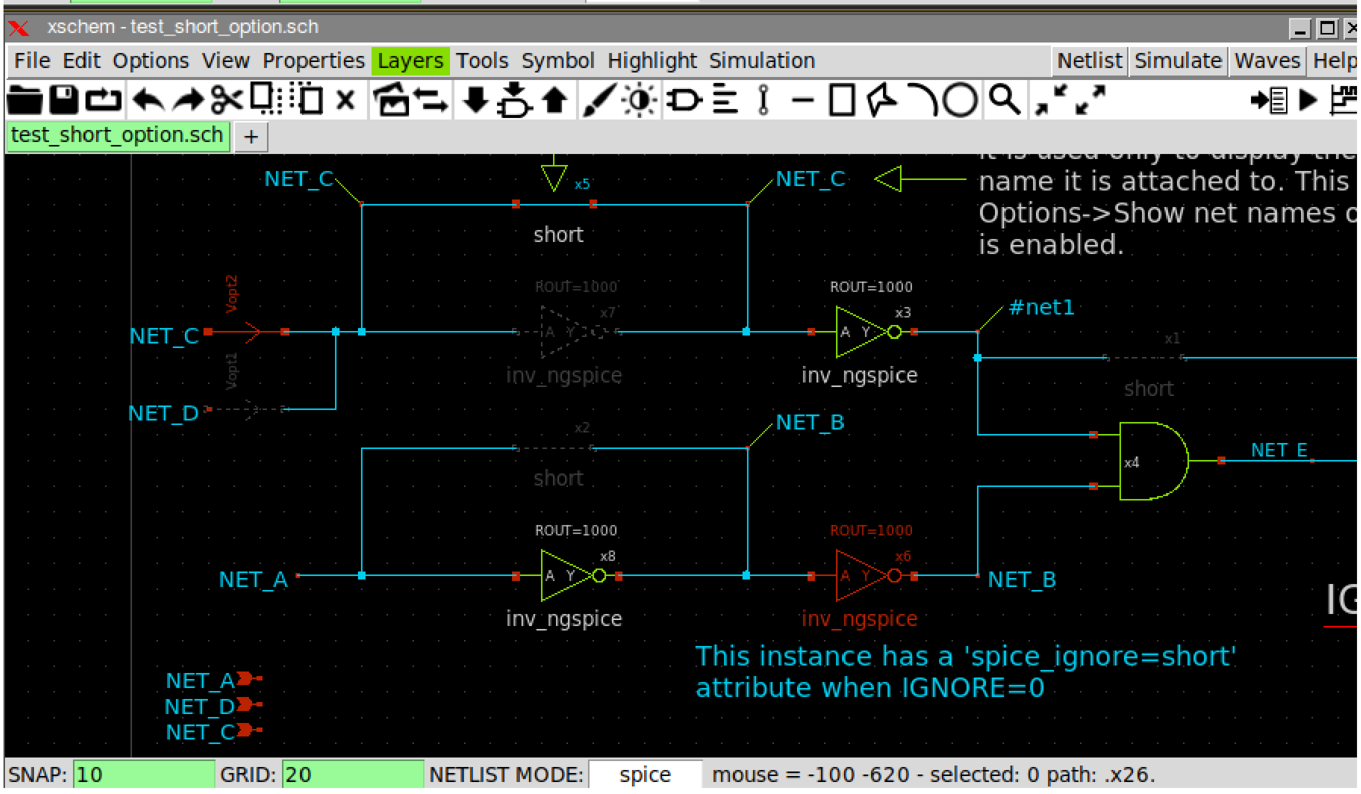
- **`lvs_ignore`**

This attribute works in the same way as above attributes, may take the values **`true`** (or **`open`**), **`false`** or **`short`**, and will affect the specific instance behaviour in the same way, but only if tcl variable **`lvs_ignore`** is set to **`1`**. This can be done in the Simulation menu: **`Set 'lvs_ignore' variable`**. If this **`lvs_ignore`** is set on the instance it will be shorted / ignored or kept as is depending on its **`lvs_ignore`** attribute and will be effective in all netlisting formats. This is mostly used to modify the produced netlist automatically when doing schematic vs layout (LVS) comparison.

By using the **`*_ignore`** attributes you can modify the circuit depending on the value of a tcl variable:


xschem - test_short_option.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_short_option.sch
NET_D
#net1
short
ROUT=1000
x7
x3
NET_C
inv_ngspice
NET_E
x1
NET_D
x2
NET_A
short
NET_E
NET_A
NET_B
name it is attached to. This
Options->Show net names
is enabled.
This instance has a 'spice_ignore=short'
attribute when IGNORE=0
NET_A
NET_D
NET_C

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -0 -160 - selected: 0 path: .x26.


xschem - test_short_option.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_short_option.sch
NET_C
NET_C
short
ROUT=1000
#net1
NET_C
inv_ngspice
NET_D
NET_B
short
NET_E
NET_A
NET_B
name it is attached to. This
Options->Show net names
is enabled.
This instance has a 'spice_ignore=short'
attribute when IGNORE=0
NET_A
NET_D
NET_C

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -100 -620 - selected: 0 path: .x26.

just set the attribute to something like:

```
spice_ignore="tcleval([if {$IGNORE == 1} {return {true}} else {return {false}}])"
```

or:

```
spice_ignore="tcleval([if {$IGNORE == 1} {return {short}} else {return {false}}])"
```

- **`spice_sym_def`**
- **`verilog_sym_def`**
- **`vhdl_sym_def`**

  If any of these attributes are present and not empty and the symbol type is set to **`subcircuit`** the corresponding netlister will ignore the schematic subcircuit for this specific instance and dump into the netlist the content of this attribute. This attribute must be paired with a **`schematic=...`** attribute set on the instance that tells the subcircuit name to use for this particular instance. The typical usage is to include a file, example:

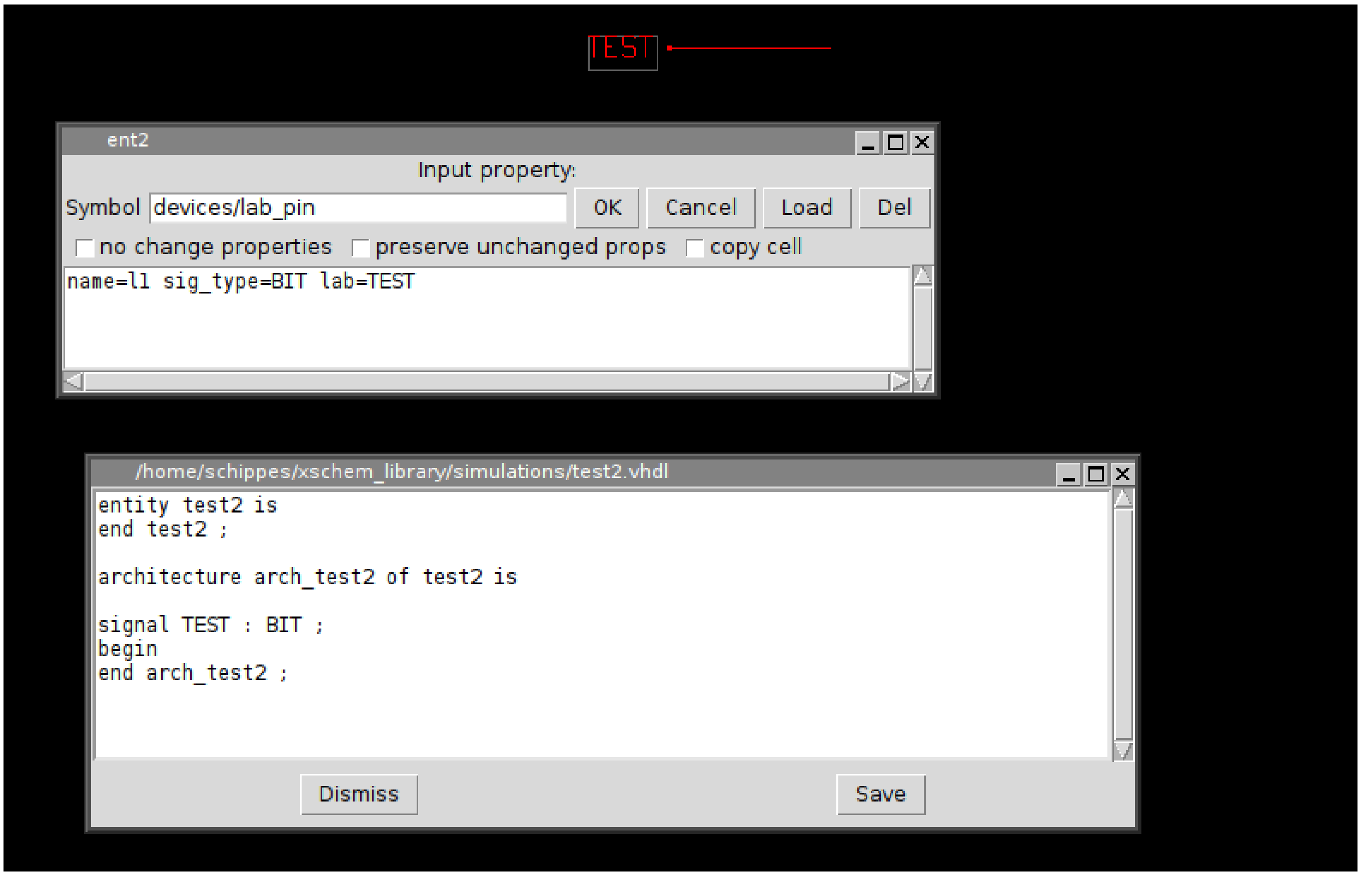
```
verilog_sym_def="tcleval(`include \"[abs_sym_path verilog_include_file.v]\")"
```

  In this example a **`verilog_include_file.v`** is included using the verilog **`` `include ``** directive. In order to generate a full path for it the **`abs_sym_path`** TCL function is used that searches for this file in any of the **`XCHEM_LIBRARY_PATH`** directories. Since TCL is used the attribute is wrapped into a tcleval(...),
  The following will appear in the generated netlist:

```
// expanding   symbol:  verilog_include.sym # of pins=3
// sym_path: /home/schippes/.xschem/xschem_library/verilog_include.sym
`include "/home/schippes/.xschem/xschem_library/verilog_include_file.v"
```

- **`sig_type`**

  For VHDL type netlist, this tells that the current label names a signal (or constant) of type **`sig_type`**. For example a label can be placed with name **`TEST`** and **`sig_type=BIT`**. The default type for VHDL if this property is missing is **`std_logic`**. The following picture shows the usage of **`sig_type`** and the resulting VHDL netlist. This property is applicable only to **`label`** type components: **`ipin.sym`**, **`iopin.sym`**, **`opin.sym`**, **`lab_pin.sym`**, **`lab_wire.sym`**.



```
devices/lab_pin
```

```
name=l1 sig_type=BIT lab=TEST
```

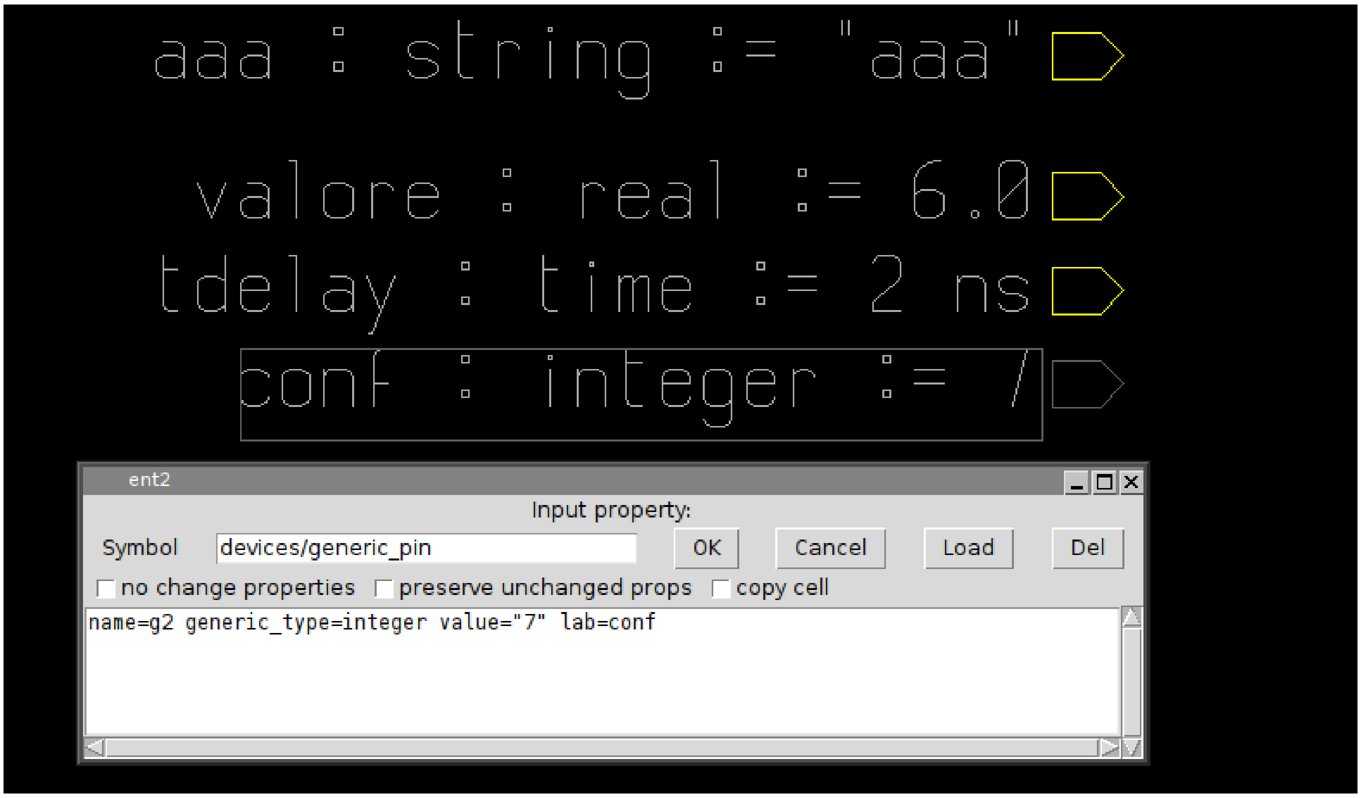
```
entity test2 is
end test2 ;

architecture arch_test2 of test2 is

signal TEST : BIT ;
begin
end arch_test2 ;
```

- **`verilog_type`**

  This is the same as sig_type but for verilog netlisting: can be used to declare a **`wire`** or a **`reg`** or any other datatype supported by the verilog language.

- **`generic_type`**

  `generic_type` defines the type of parameters passed to VHDL components. Consider the following examples of placement of **`generic_pin`** components in a VHDL design:



Symbol: devices/generic_pin

```
name=g2 generic_type=integer value="7" lab=conf
```

As you will see in the parameters slide, generics (they are just parameters passed to components) can be passed also via property strings in addition to using **`generic_pin`** components.

- **`class`**

  The **`class`** attribute is used to declare the class of a VHDL signal, most used classes are **`signal`** and **`constant`**. Default if missing is **`signal`**.

- **`device_model`**

  This attribute contains a SPICE .model or .subckt specification (**`device_model=".model D1N4148 D ...."`**) that will be printed at end of netlist only once for the specified component (D1N4148 in the example). **`device_model`** attributes defined at instance level override the **`device_model`** set in the symbol if any.

- **`schematic`**

  This attribute specifies an alternate schematic file to open when descending into the subcircuit. This is done only for the specific instance allowing to differentiate implementation ona specific instance of a given subcircuit. The specified schematic must have the same interface (in/out/inout pins) as the base schematic (that is inferred from the symbol name).
  Example: **`schematic=sky130_tests/inv2.sch`**

- **`pinnumber(name)`**

  This will override at instance level the value of attribute **`pinnumber`** of pin **`name`** of the symbol. This is mainly used for tedax, where by back annotation a connection to a symbol must be changed.

- **`pinnumber(index)`**

This will override at instance level the value of attribute **`pinnumber`** of **`index`**th pin of the symbol. This is mainly used for tedax, where by back annotation a connection to a symbol must be changed. This notation is faster since xschem does not have to find a pin by string matching.

- **`pin_attr(name|index)`**

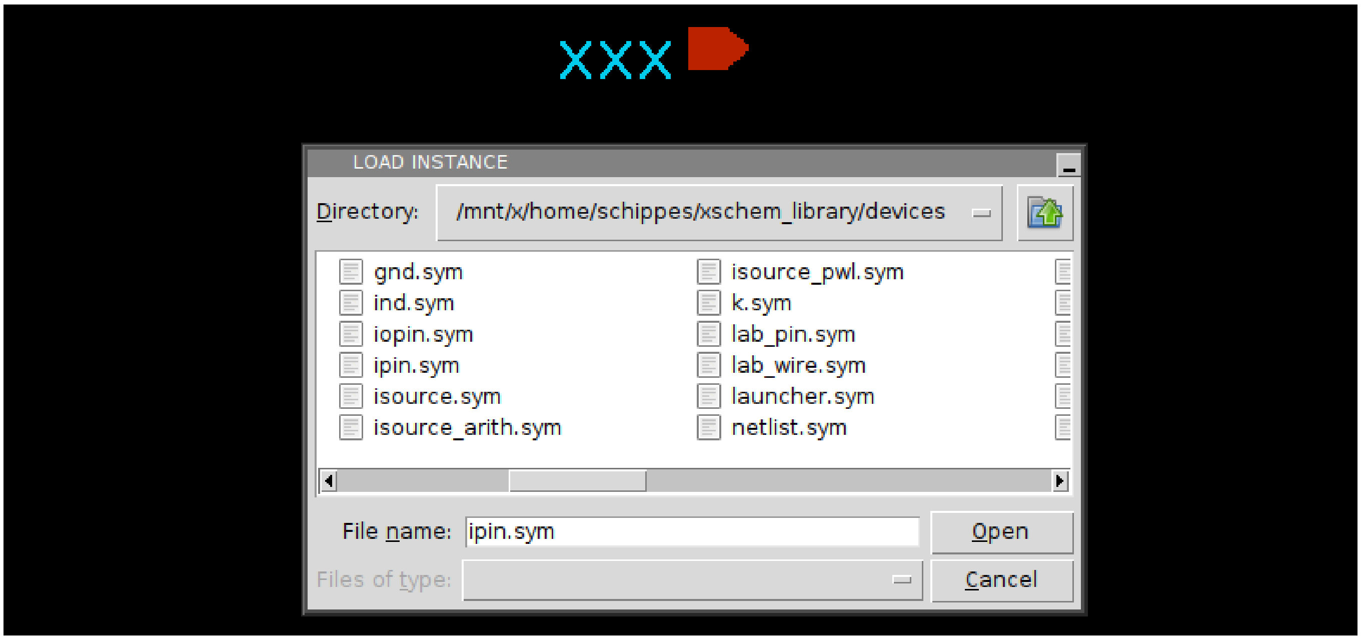
  This is a general mechanism where at instance level a pin attribute may be overridden for netlisting. Example: **`sig_type(OUT)=bit_vector`** (set VHDL type of pin OUT to bit_vector).

TCL ATTRIBUTE SUBSTITUTION

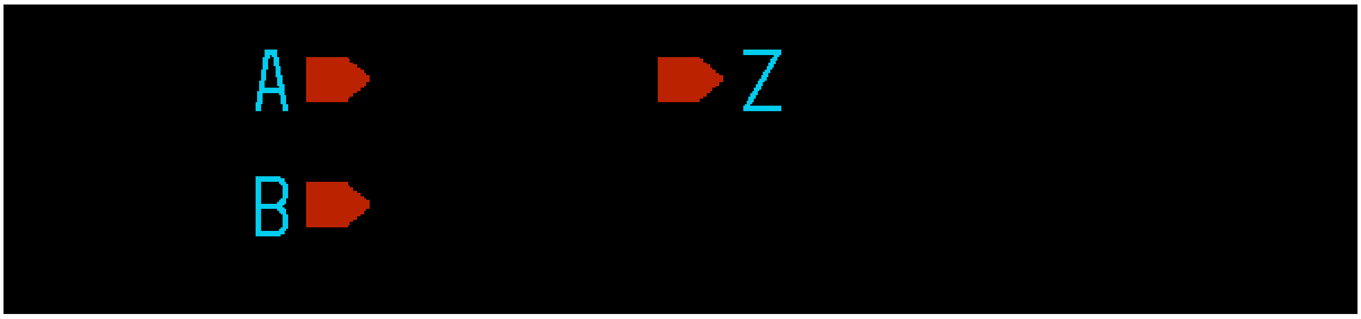
Any attribute and symbol text can be embedded in a **`tcleval(....)`** construct, the string inside the parentheses will be passed to the tcl interpreter for evaluation. This allows to use any tcl variable/command/expression. Example:
`value="tcleval([expr {[info exists ::resval] ? $::resval : {100k}}])"`
this attribute will set **`value`** (example: value of a resistor) to 100k if global tcl variable **`resval`** is not set or to the value of **`resval`** if set.

CREATING A CIRCUIT SCHEMATIC

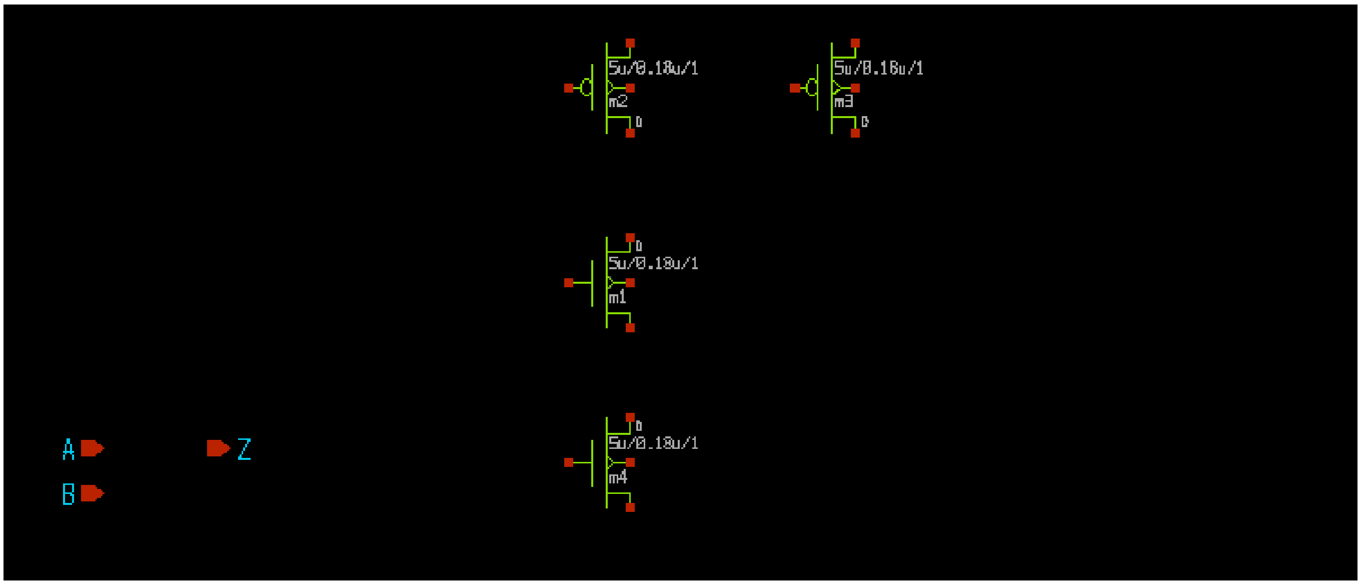
To create a new circuit start from an empty window, run xschem and select **`New Schematic`** in the **`File`** menu. Suppose we want co create a NAND gate, with two inputs, A and B and one output, Z. Lets start placing the input and output schematic pins; use the **`Insert`** key and locate the **`devices/ipin.sym`** symbol. After placing it change its lab attribute to **`'A'`**



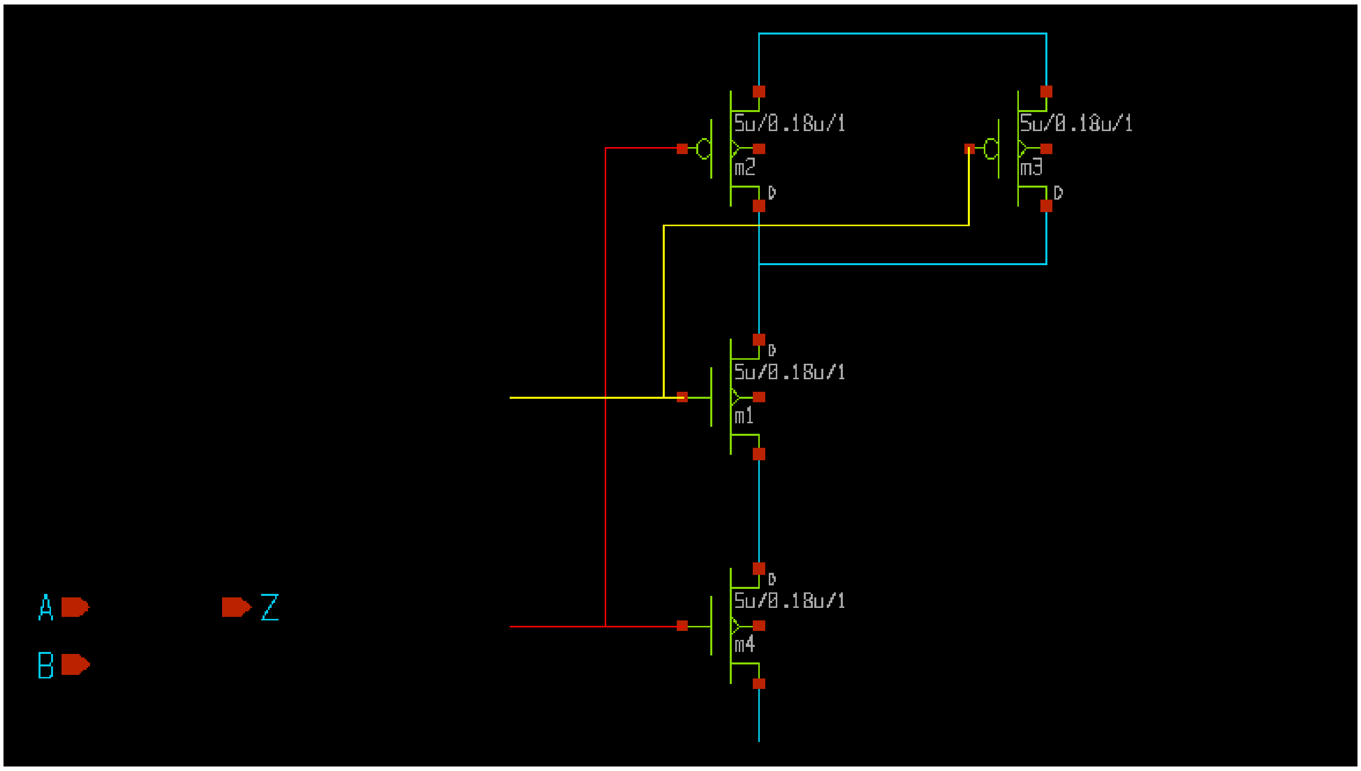
Copy another instance of it and set its lab attribute to **`B`**. Next place an output pin **`devices/opin.sym`** and set its lab to **`Z`**. The result will be as follows:



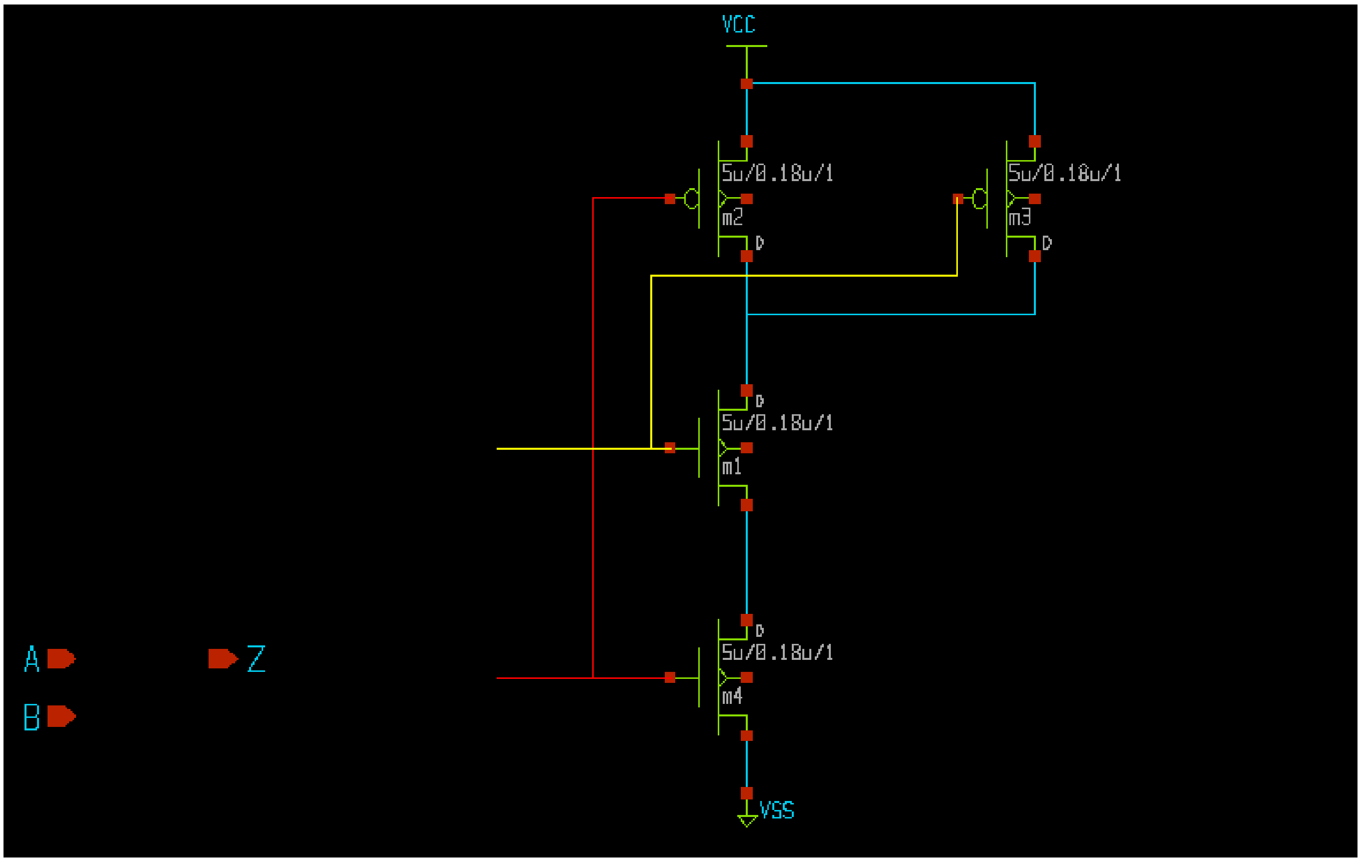
Now we need to build the actual circuit. Since we plan to do it in CMOS technology we need nmos and pmos transistors. Place one nmos from **`devices/nmos4.sym`** and one pmos from **`devices/pmos4.sym`** By selecting them with the mouse, moving (**`m`** bindkey), copying (**`'c'`** bindkey) place 4 transistors in the following way (the upper ones are pmos4, the lower ones nmos4):



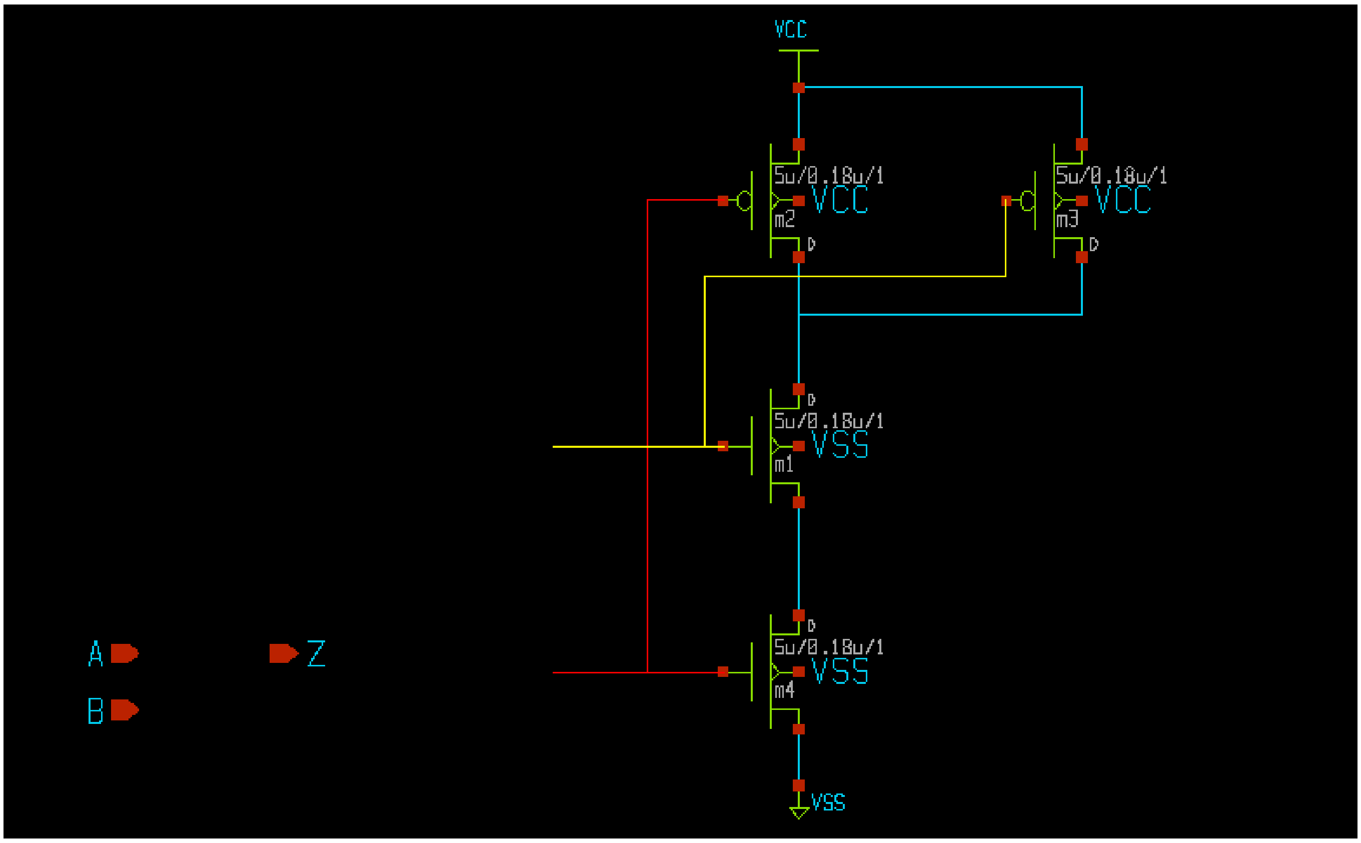
now draw wires to connect together the transistor to form a NAND gate; in the picture i have highlighted 2 electrical nodes by selecting one wire segment of each and pressing the **'k'** bindkey.



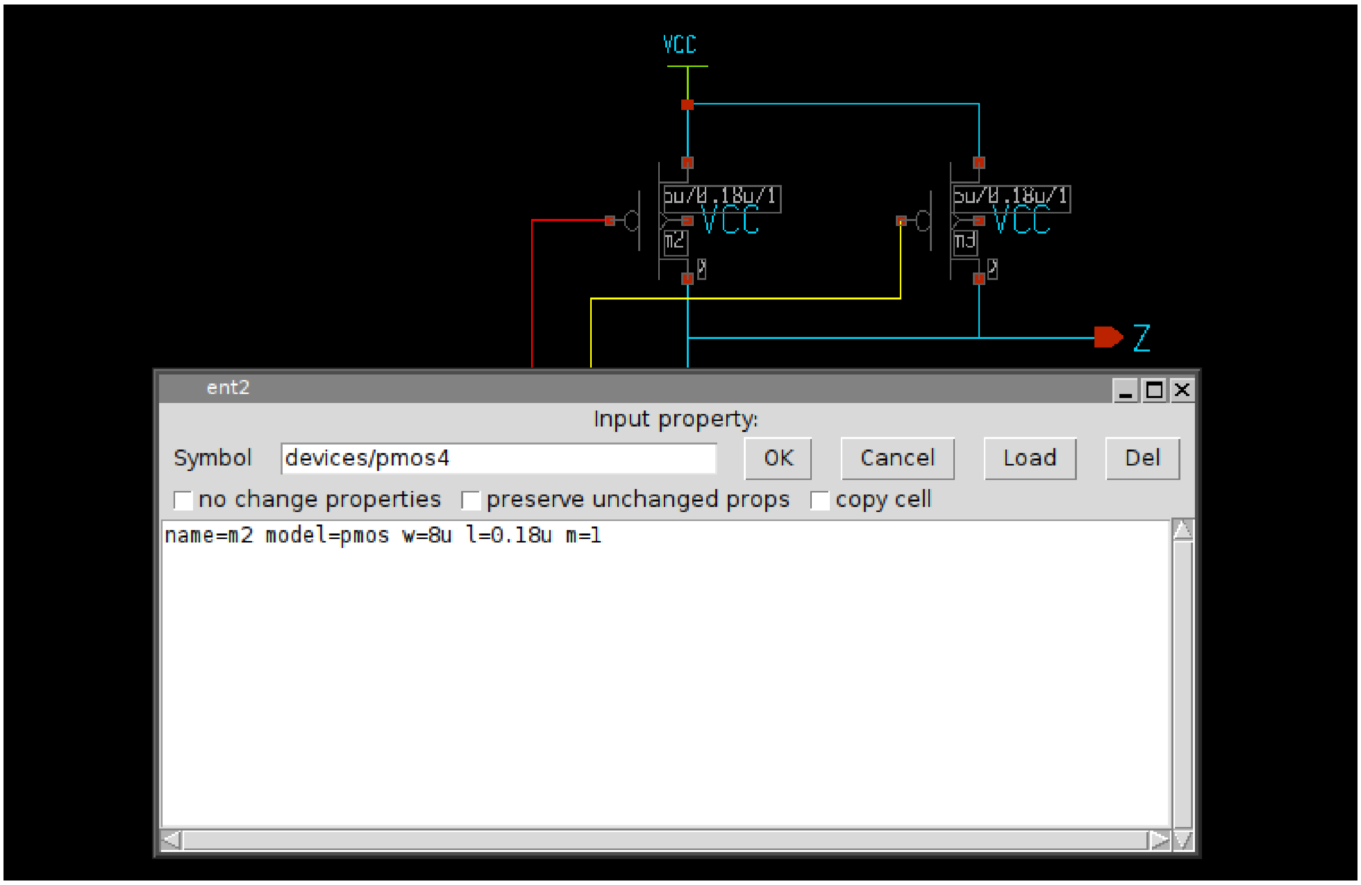
Next we need to place the supply nodes , VCC and VSS. we decide to use global nodes. Global nodes in SPICE semantics are like global variables in C programs, they are available everywhere, we do not need to propagate global nodes with pins. We could equally well use regular pins , as used for the A and B inputs, I am just showing different design styles. Use the **Insert** key and place both **devices/vdd.sym** and **devices/gnd.sym** Since the default names are respectively VDD and GND use the edit property bindkey **'q'** to change these to VCC and VSS.



we still need to connect the body terminals of the mos transistors. One possibility is to hookup the two upper pmos transistor terminals to VCC with wires, and the two bottom nmos terminals to VSS with wires, but just to show different design styles i am planning to use "by name" connection with labels. So place a wire label **`devices/lab_pin.sym`** and use 4 instances of it to name the 4 body terminals. Remember, while moving (select and press the **`'m'`** key) you can flip/rotate using the **`R`**/**`F`** keys.



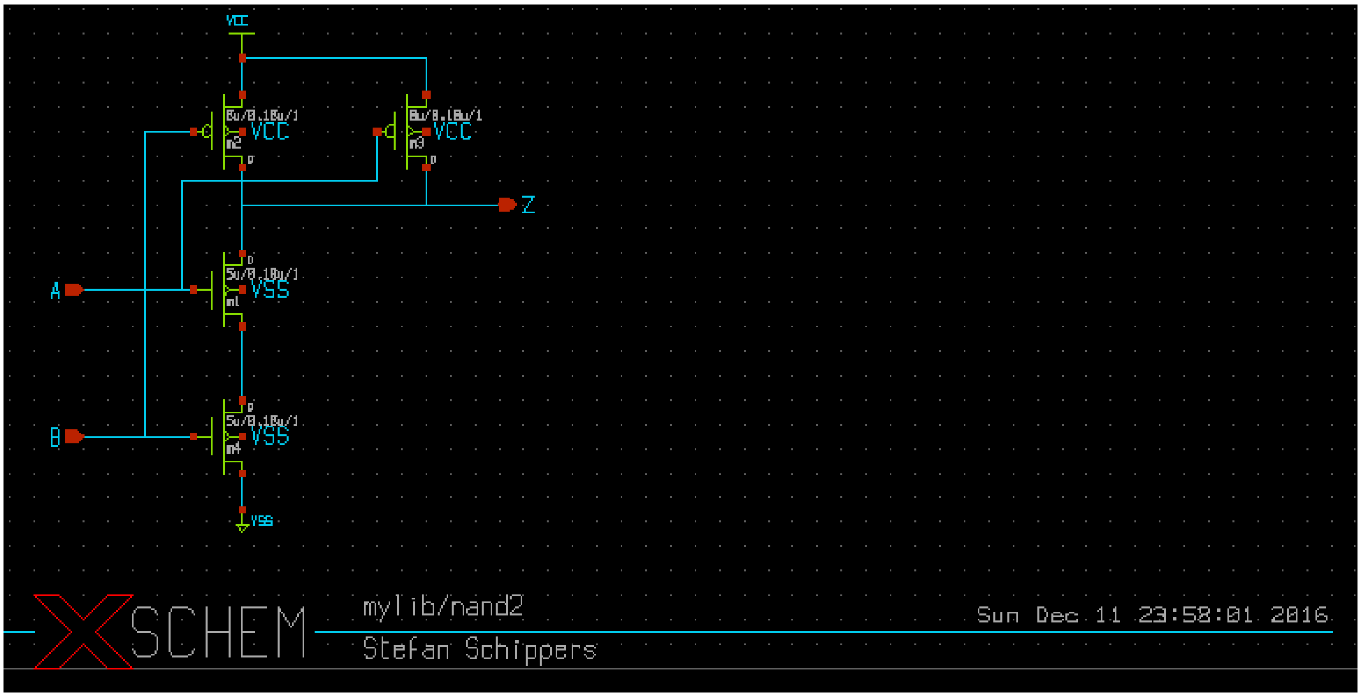
Finally we must connect the input and output port connectors, and to complete the gate schematic we decide to use W=8u for the pmos transistors. Select both the pmos devices and press the edit property **`'q'`** key; modify from 5u (default) to 8u.



devices/pmos4

```
name=m2 model=pmos w=8u l=0.18u m=1
```

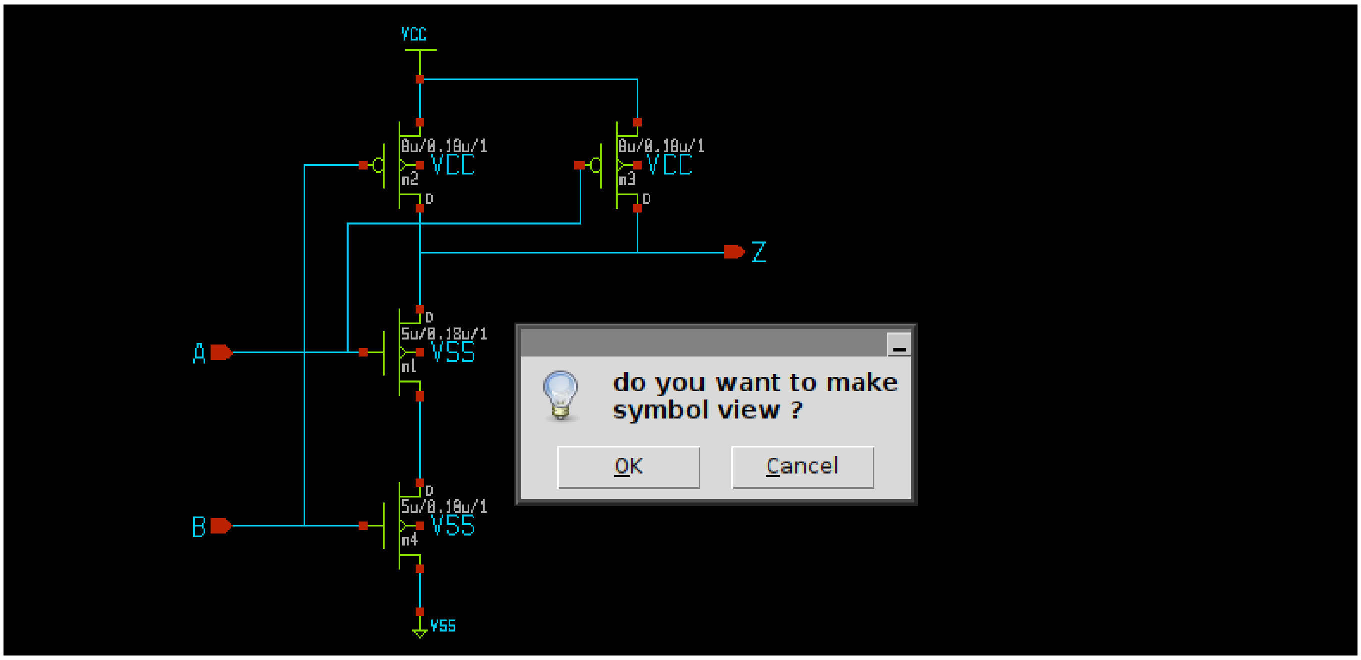
Now do a Save as operation, save it for example in **`mylib/nand2.sch`**.
To make the schematic nicer we also add the title component. This component is not netlisted but is useful, it reports the modification date and the author. Place the **`devices/title.sym`** component. The NAND gate is completed! (below picture also with grid, normally disabled in pictures to make image sizes smaller).



Normally a cmos gate like the one used in this example is used as a building block (among many others) for bigger circuits, therefore we need to enclose the schematic view above in a symbol representation.

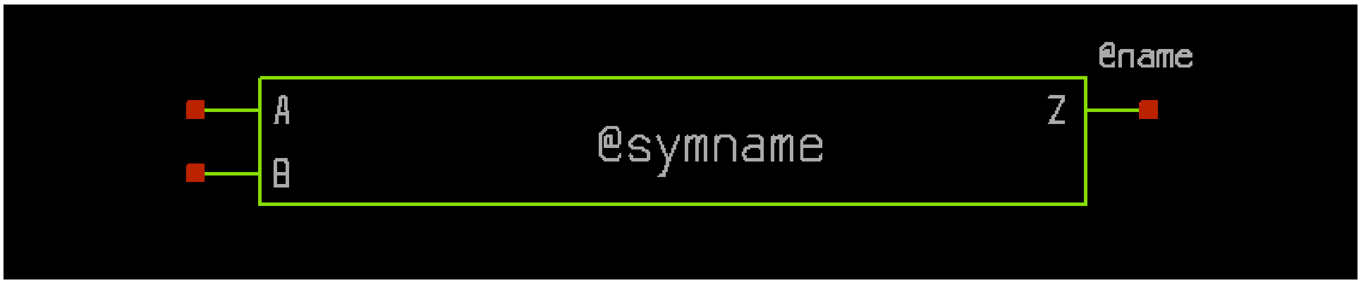
Automatic symbol creation

XSCHEM has the ability to automatically generate a symbol view given the schematic view. Just press the **'a'** bindkey in the drawing area of the nand2 gate.

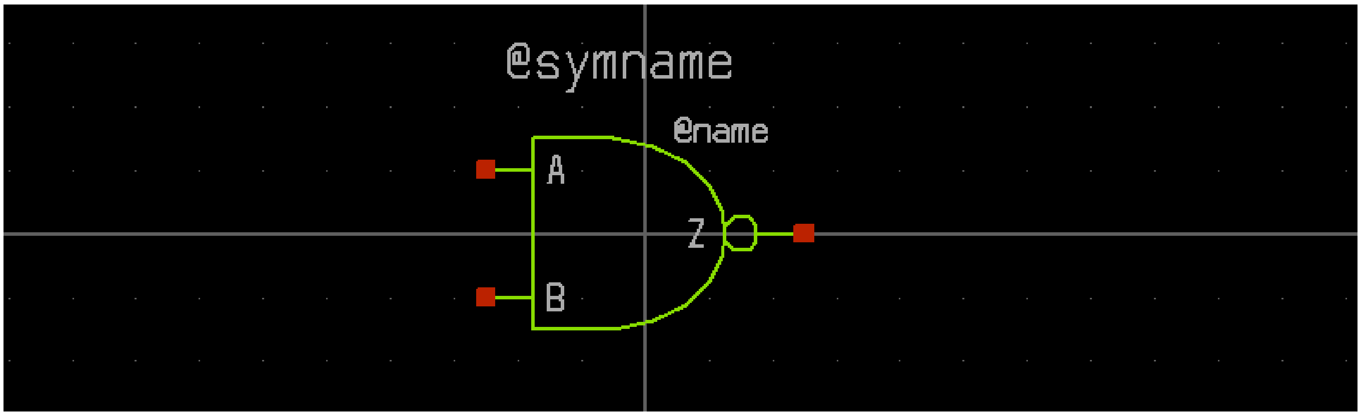


do you want to make symbol view ?

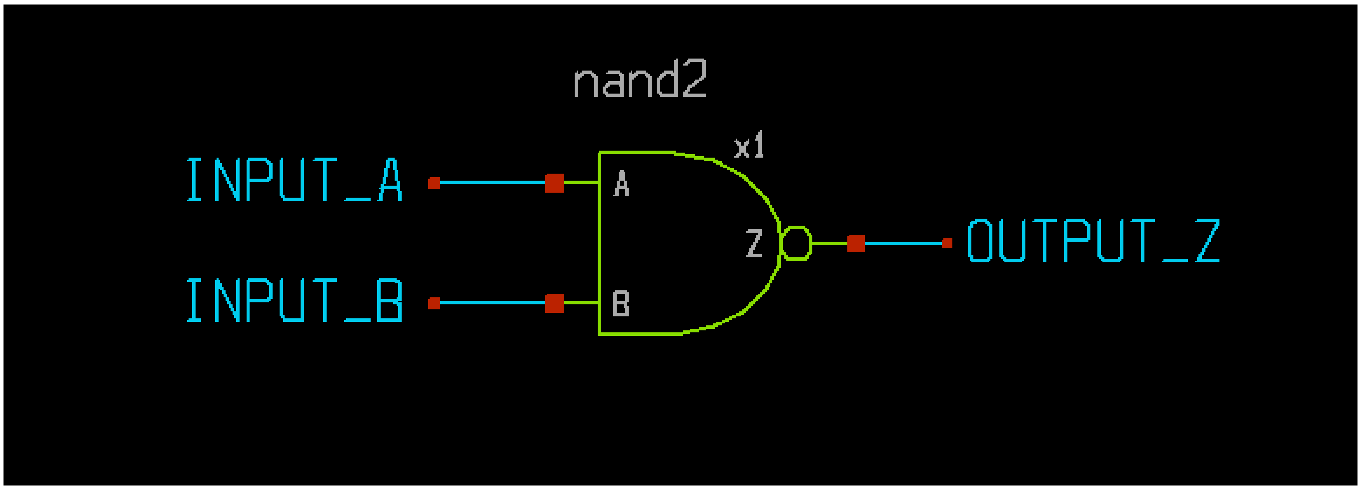
After pressing 'OK' a **mylib/nand2.sym** file is generated. try opening it (**File->Open**):



As you can see a symbolic view of the gate has been automatically created using the information in the schematic view (specifically, the input/output pins). Now, this graphic is not really looking like a nand gate, so we may wish to edit it to make it look better. Delete (by selecting and pressing the **Delete** key) all the green lines, keep the red pins, the pin labels and the @symname and @name texts, then draw a nand shape like in the following picture. To allow you to draw small segments you may need to reduce the snap factor (menu **View->Half snap threshold**) remember to reset the snap factor to its default setting when done.



This completes the nand2 component. It is now ready to be placed in a schematic. Open a test schematic (for example **`mylib/test.sch`** (remember to save the nand2.sym you have just created), press the **`Insert`** key and locate the **`mylib/nand2.sym`** symbol. Then insert **`devices/lab_pin.sym`** components and place wires to connect some nodes to the newly instantiated nand2 component:

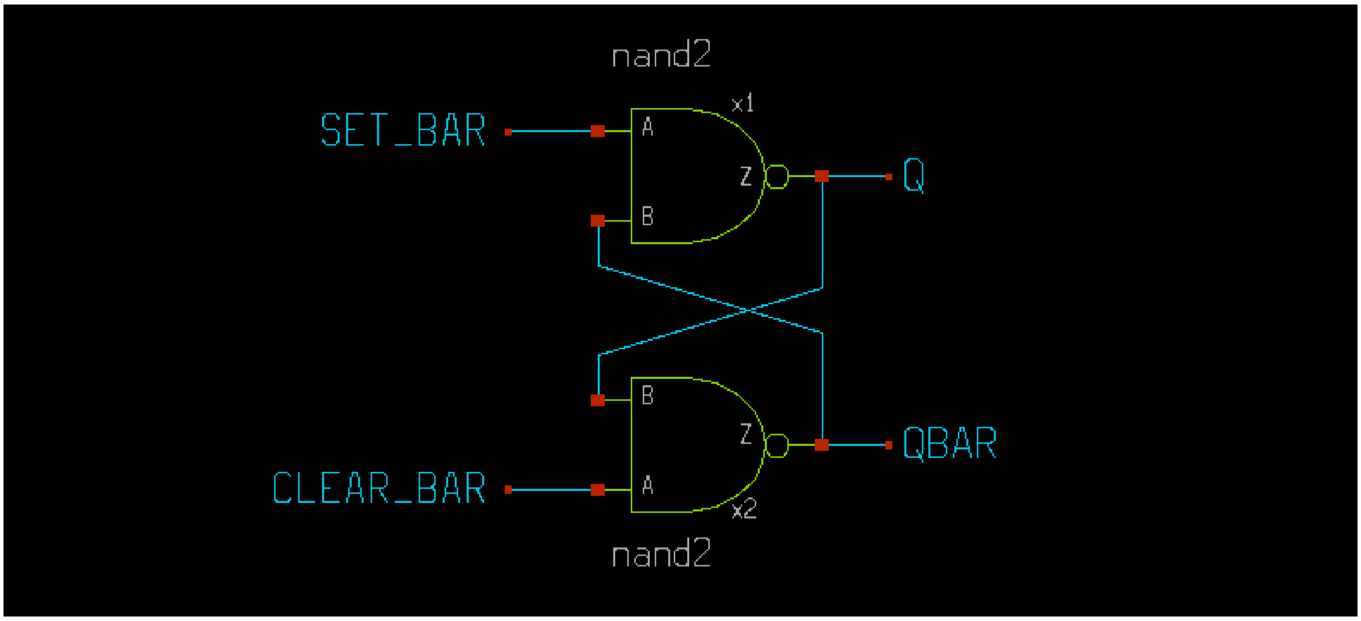


This is now a valid circuit. Let's test it by extracting the SPICE netlist. Enable the showing of netlist window (**`Options -> Show netlist win`**, or **`'A'`** key). Now extract the netlist (**`Netlist`** button on the right side of the menu bar, or **`'N'`** key). the SPICE netlist will be shown.

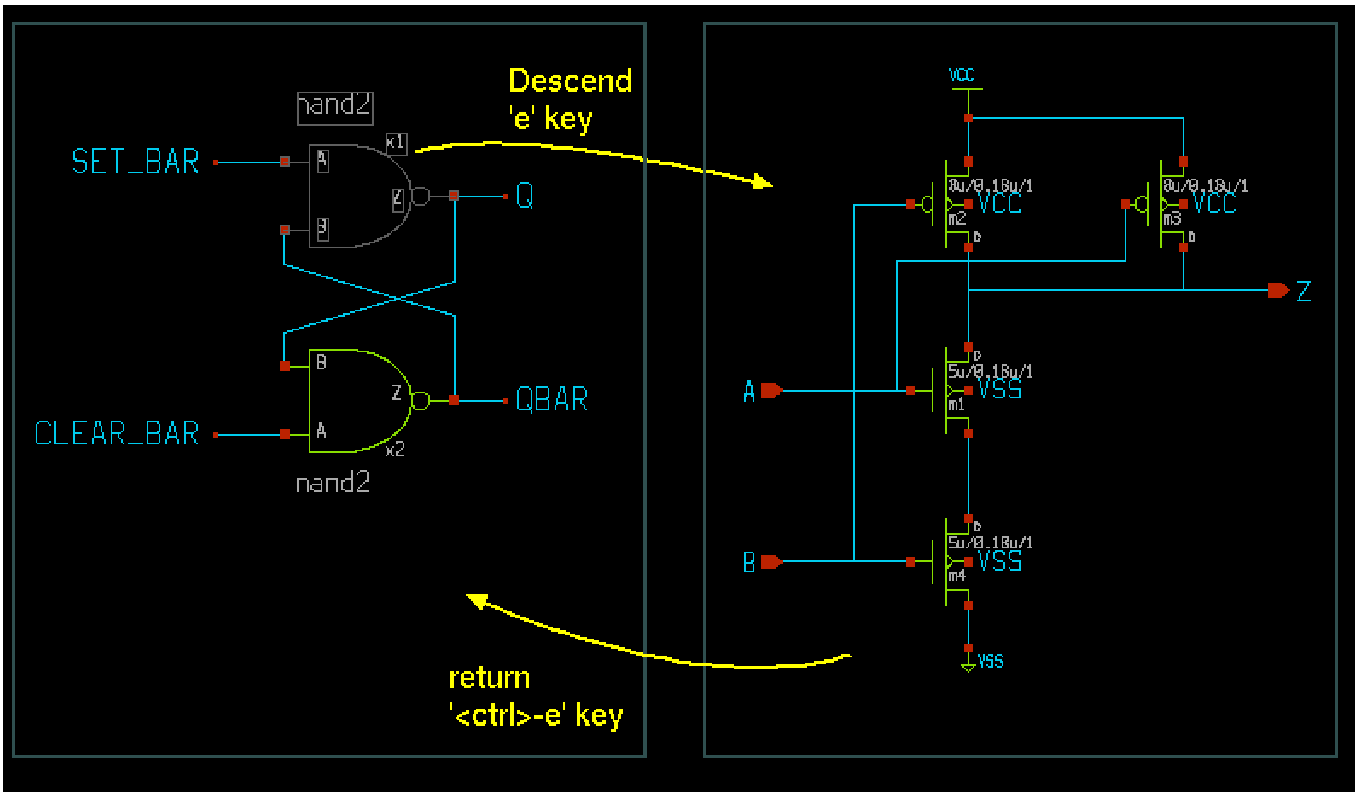
```
**.subckt test
x1 OUTPUT_Z INPUT_A INPUT_B nand2
**** begin user architecture code
**** end user architecture code
**.ends

* expanding symbol: mylib/nand2 # of pins=3
.subckt nand2 Z A B
*.ipin A
*.opin Z
*.ipin B
m1 Z A net1 VSS nmos w=5u l=0.18u m=1
m2 Z B VCC VCC pmos w=8u l=0.18u m=1
m3 Z A VCC VCC pmos w=8u l=0.18u m=1
m4 net1 B VSS VSS nmos w=5u l=0.18u m=1
**** begin user architecture code
**** end user architecture code
.ends

.GLOBAL VCC
```

```
.GLOBAL VSS
.end
```

This is an example of a hierarchical circuit. The nand2 is a symbol view of another lower level schematic. We may place multiple times the nand2 symbol to create more complex circuits.



By selecting one of the nand2 gates and pressing the **`'e'`** key or menu **`Edit -> Push schematic`** we can 'descend' into it and navigate through the various hierarchies. Pressing **`<ctrl>e`** returns back to the upper level.



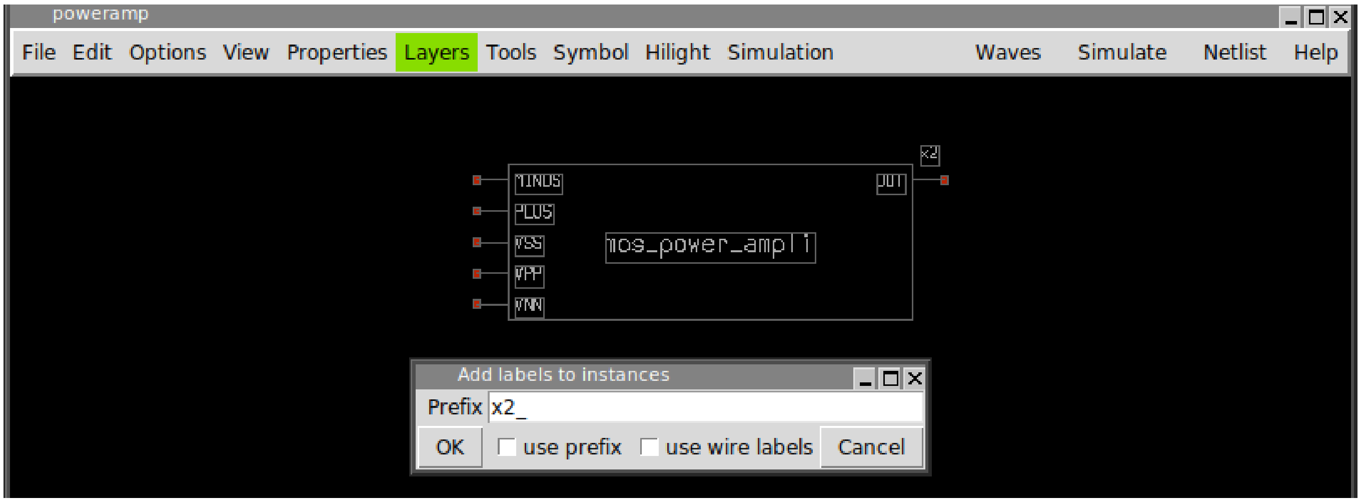
This is the corresponding netlist:

```
**.subckt test
x1 Q SET_BAR QBAR nand2
x2 QBAR CLEAR_BAR Q nand2
**** begin user architecture code
**** end user architecture code
**.ends

* expanding symbol: mylib/nand2 # of pins=3
.subckt nand2 Z A B
*.ipin A
*.opin Z
*.ipin B
m1 Z A net1 VSS nmos w=5u l=0.18u m=1
m2 Z B VCC VCC pmos w=8u l=0.18u m=1
m3 Z A VCC VCC pmos w=8u l=0.18u m=1
m4 net1 B VSS VSS nmos w=5u l=0.18u m=1
**** begin user architecture code
**** end user architecture code
.ends

.GLOBAL VCC
.GLOBAL VSS
.end
```

The advantage of using hierarchy in circuits is the same as using functions in programming languages; avoid drawing many repetitive blocks. Also the netlist file will be much smaller.

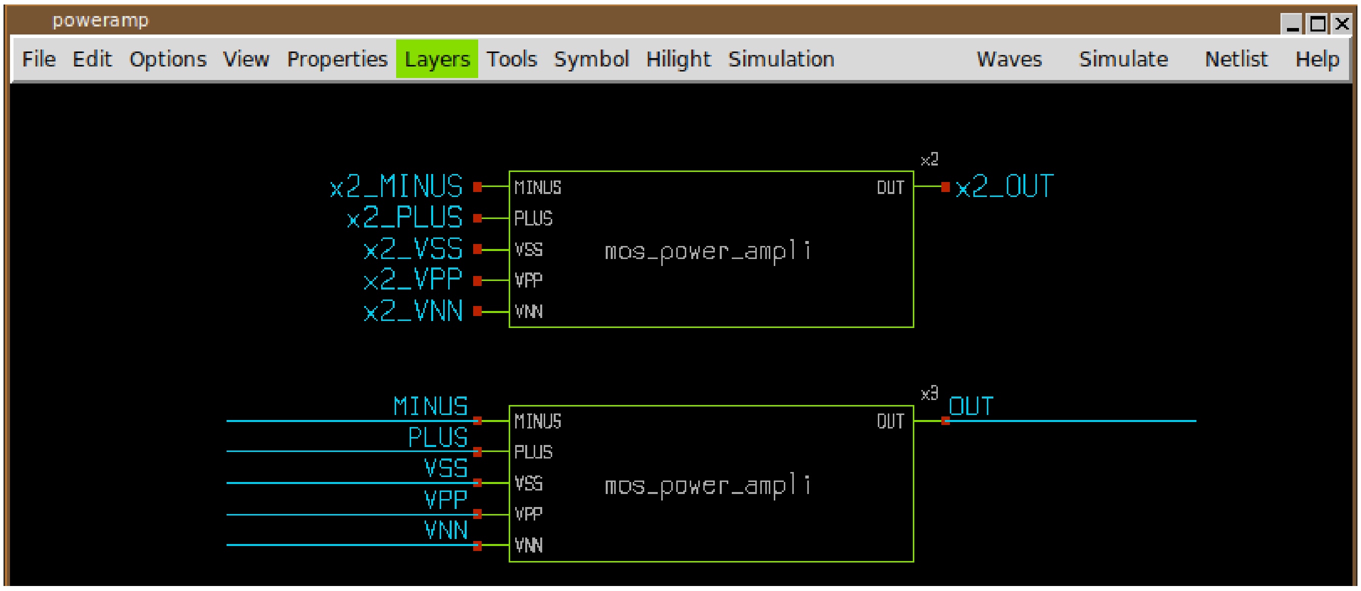
Automatic Component Wiring

When a new symbol is placed there is a function to connect its pins to auto-named nets: select the symbol, then Press the **'H'** key or the **Symbol->Attach net labels to component instance** menu entry.



The **use prefix** will prepend the shown prefix to the wire names to be attached to the component. The default value for the prefix is the instance name followed by an underscore.
The **use wire labels** will use wire labels instead of pin labels. Wire labels have the text name field offset vertically to allow a wire to pass through without crossing the wire name. in the picture below, the first component is wired with

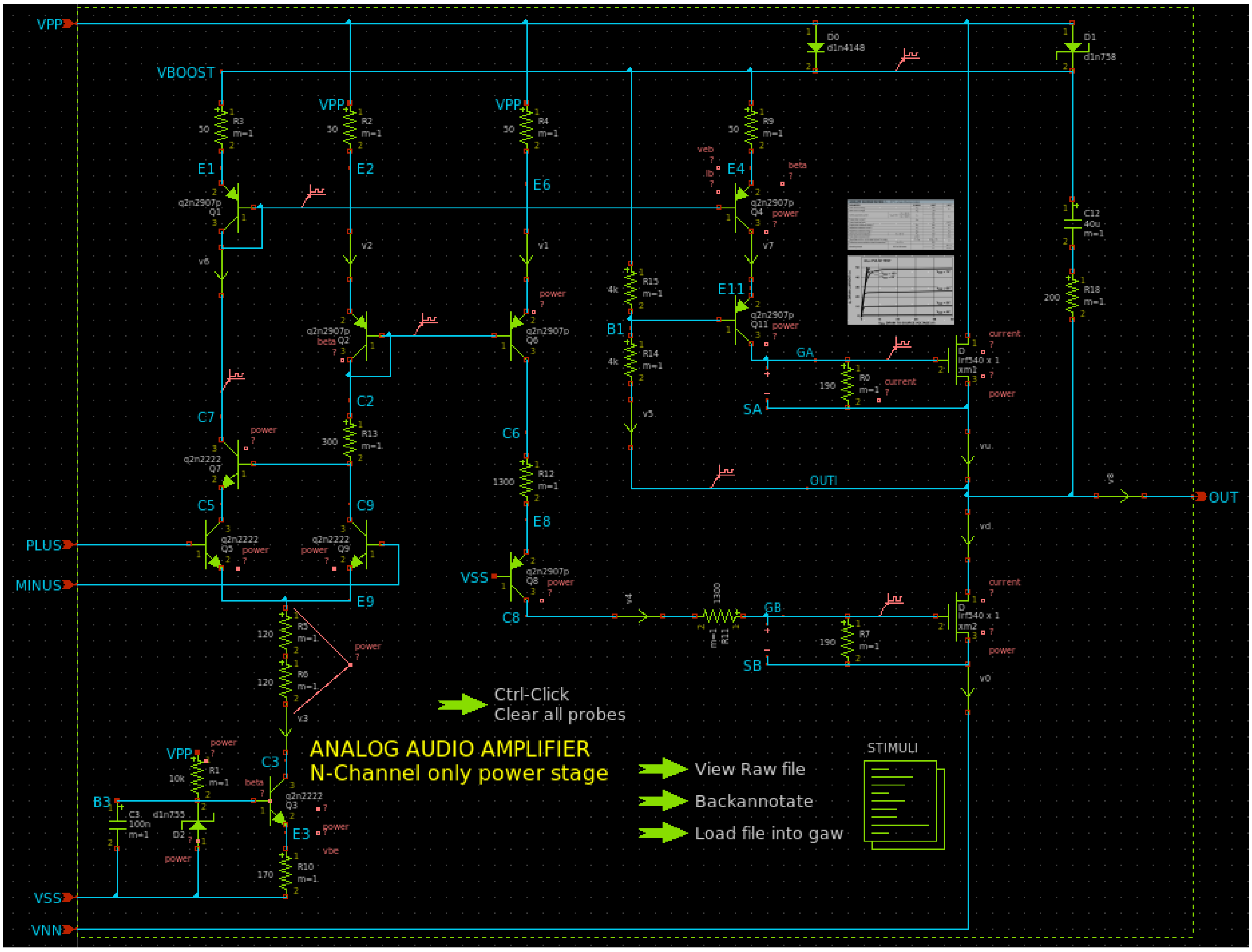
use prefix selected and **use wire labels** not selected, the second example with **use prefix** not selected and **use wire labels** selected. As you can see in the second example you may draw wires without overstriking the labels.



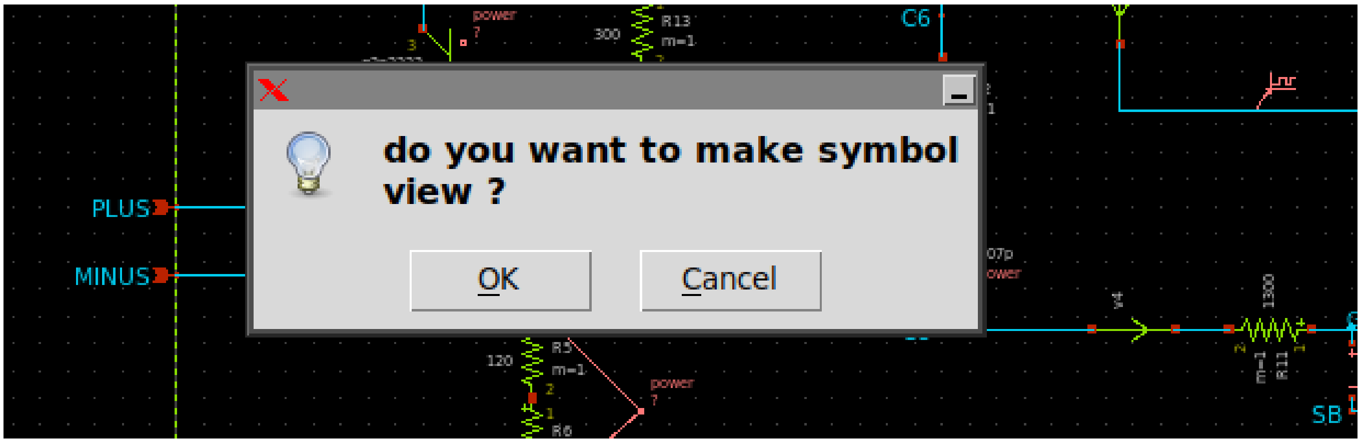
CREATING SYMBOLS

Creating a subcircuit symbol

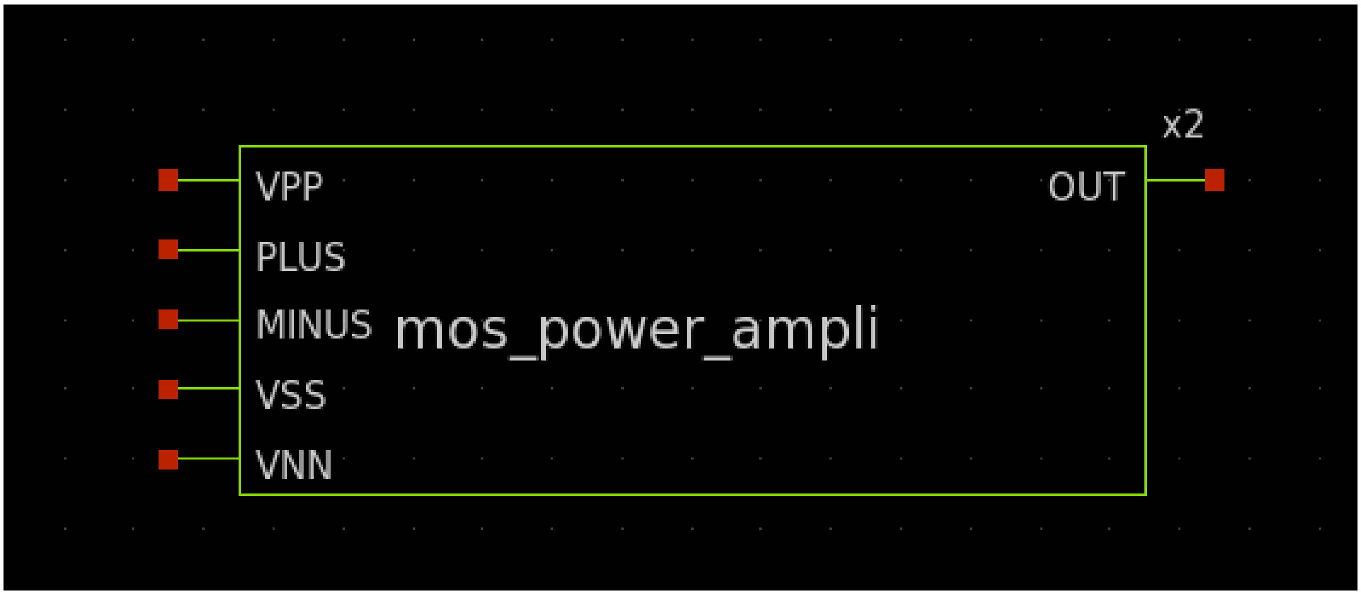
Suppose you have just finished creating a circuit and you now want to create a symbol for it so you can use this circuit as a sub block in other schematics:



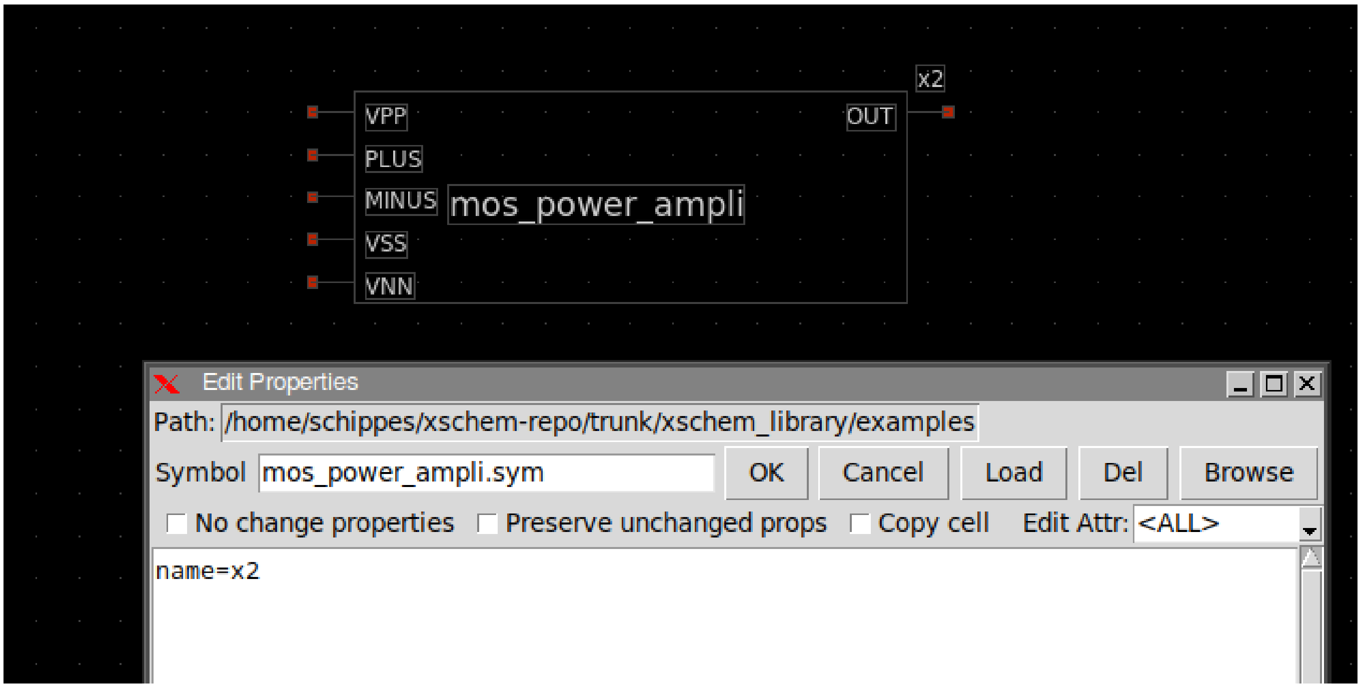
Above schematic contains **VPP**, **PLUS**, **MINUS**, **VSS**, **VNN** input pins and **OUT** output pin.
If you press the **a** key xschem will generate a symbol automatically.


do you want to make symbol view ?

If your schematic is called **`mos_power_amplifier.sch`** the symbol will be saved in the same place as the schematic and named **`mos_power_amplifier.sym`**.
If you open a new empty schematic and use the **`Insert`** or **`Shift-I`** key to insert a symbol and select the **`mos_power_amplifier.sym`** you get this:



If you select the symbol instance and press **`q`** you see the instance **`name`** attribute; the name attribute specifies an unique name in current schematic. There can not be two **`x2`** instances in a schematic. If you copy the placed instance to get two of them the new one will be automatically renamed (to **`x3`** or **`x1`**, or any available unique name).



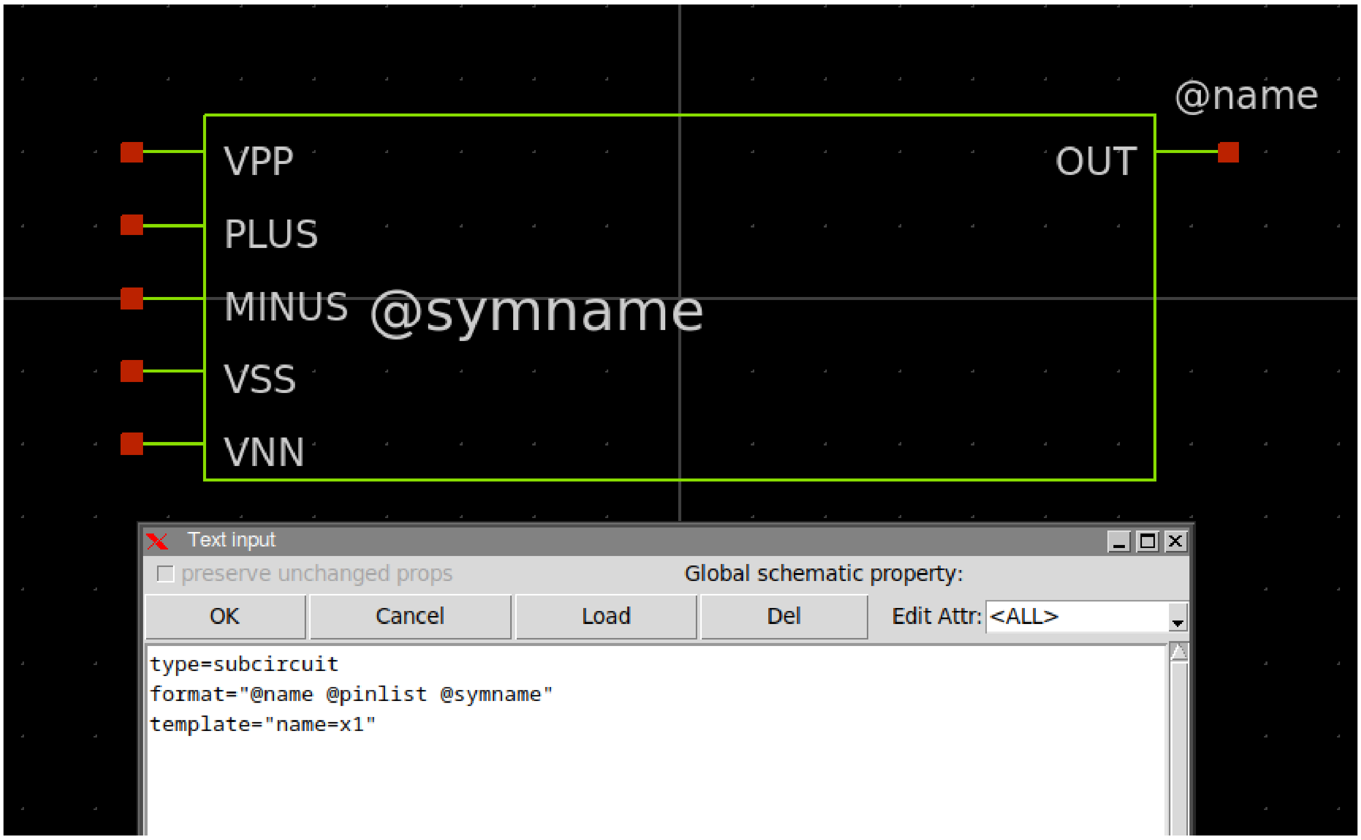
Path: /home/schippes/xschem-repo/trunk/xschem_library/examples

Symbol mos_power_ampli.sym

☐ No change properties ☐ Preserve unchanged props ☐ Copy cell Edit Attr: <ALL>

```
name=x2
```

If you descend into the symbol and press **q** you see the following attributes:



☐ preserve unchanged props Global schematic property:

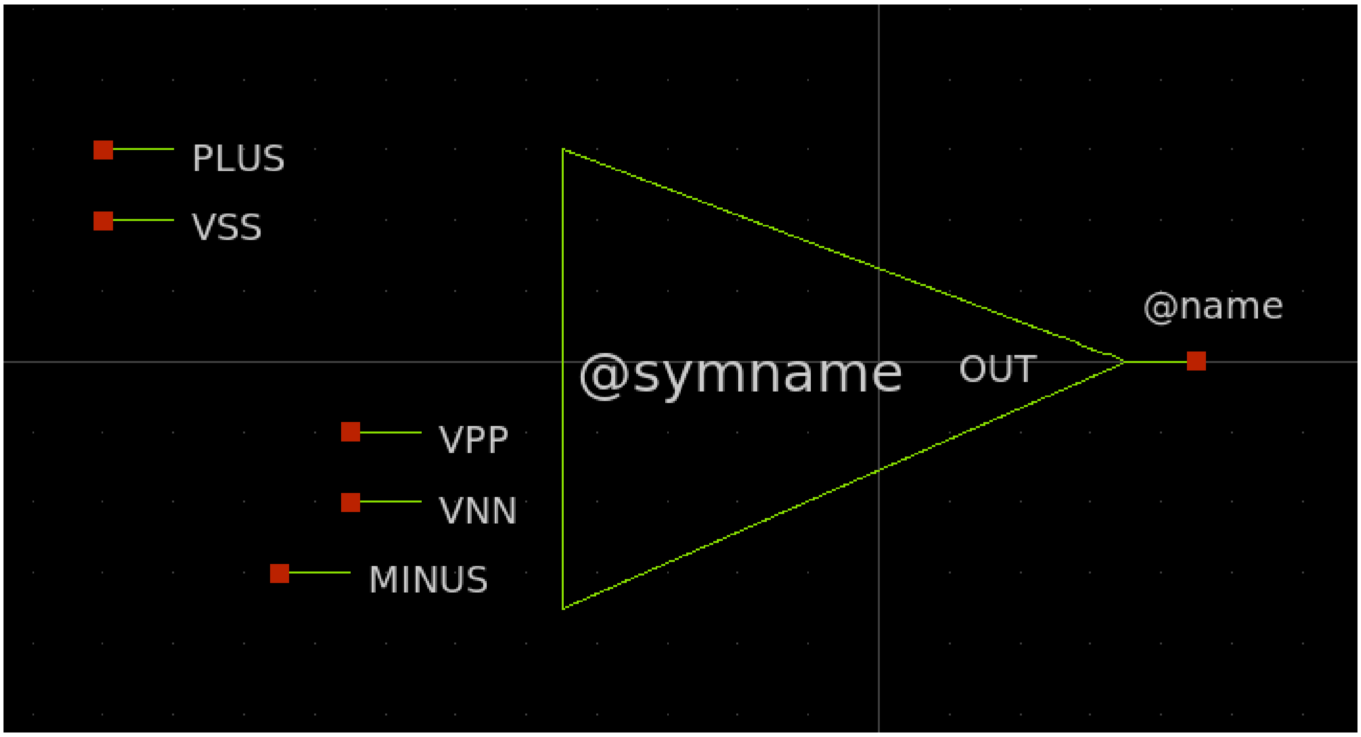
Edit Attr: <ALL>

```
type=subcircuit
format="@name @pinlist @symname"
template="name=x1"
```

```
type=subcircuit
format="@name @pinlist @symname"
template="name=x1"
```

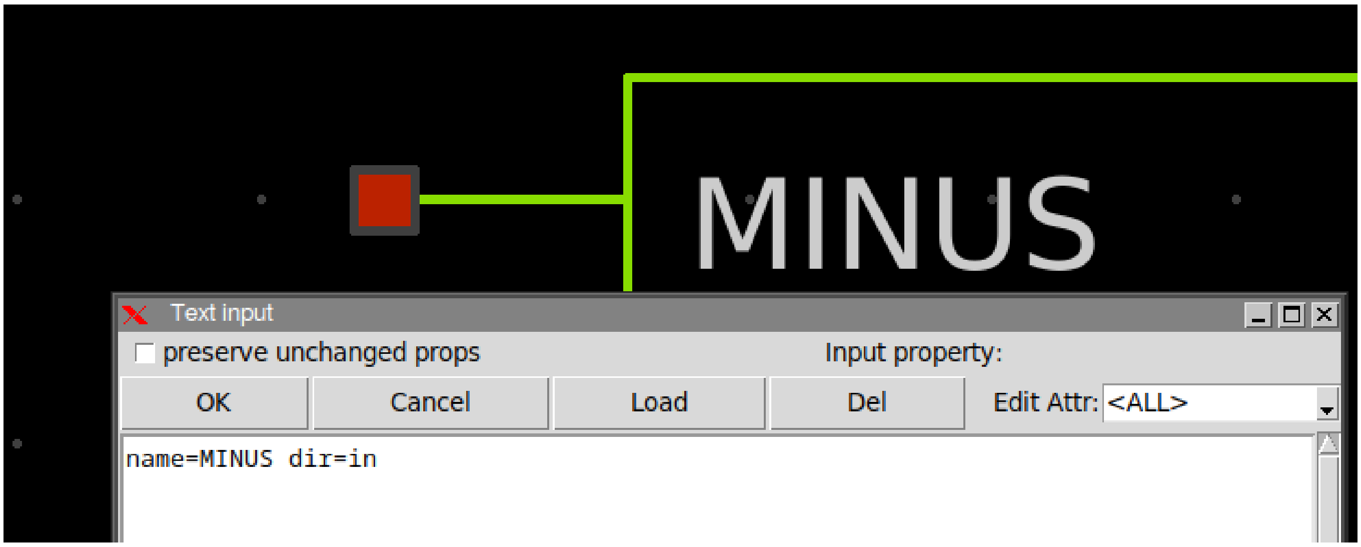
These attributes are using by xschem to generate the subcircuit netlist line. the **`format`** attribute tells xschem that a line containing the instance name (**`@name`**, replaced by **`x2`**), the list of attached nets (**`@pinlist`**, replaced by the nets attached to the symbol i/o ports in the order they are declared in the subcircuit) and the symbol name (**`@symname`**, replaced by **`mos_power_amplifier`**).
The **`type`** attribute tells xschem that the symbol is a subcircuit (not a terminal symbol) and netlister should further descend into the corresponding schematic to complete the netlist.
The **`template`** attribute defines default values for attributes when the symbol is placed in a schematic. For example if you place an instance of this symbol in an empty schematic the instance **`name`** attribute will be set to **`x1`**. If there is already an **`x1`** instance xschem will automatically rename the instance to a unique name.

You can manually edit the symbol to change its shape or change the pin ordering.
If you change the pin positions always move the pin (the red square) and the label together.



If you select one pin (the small red square box) and press 'q' you see the pin attributes:
`name` specifies the pin name.
`dir` specifies the pin direction (**`in, out, inout`**).
It is good practice to verify that the pin **`name`** attribute matches the name of the text label next to it.

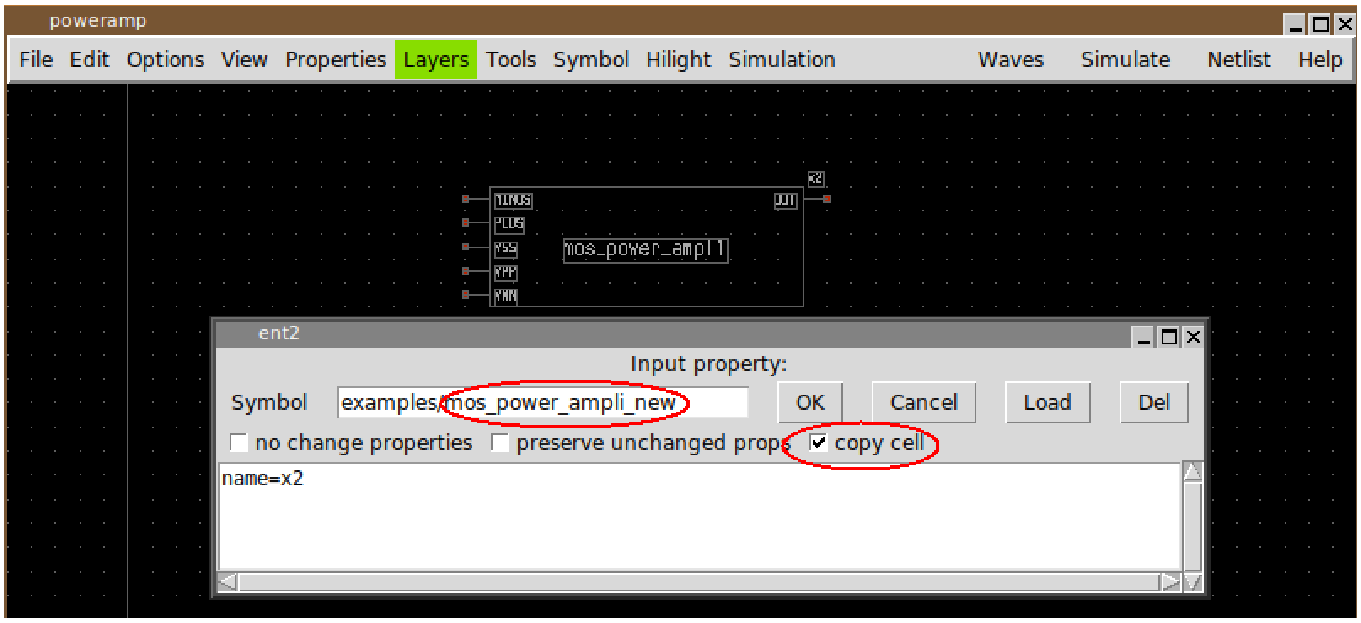
If you edit the text label next to a pin the pin name attribute will be changed automatically.



```
name=MINUS dir=in
```

Creating a new symbol and schematic by cloning

Another useful approach to create a new component (both symbol and schematic view) is to 'clone' it from a similar existing component: after copying a component to a different place in the schematic, press the edit property bindkey (**`q`** key) and set a new name for the symbol, set also the **`copy cell`** checkbox:

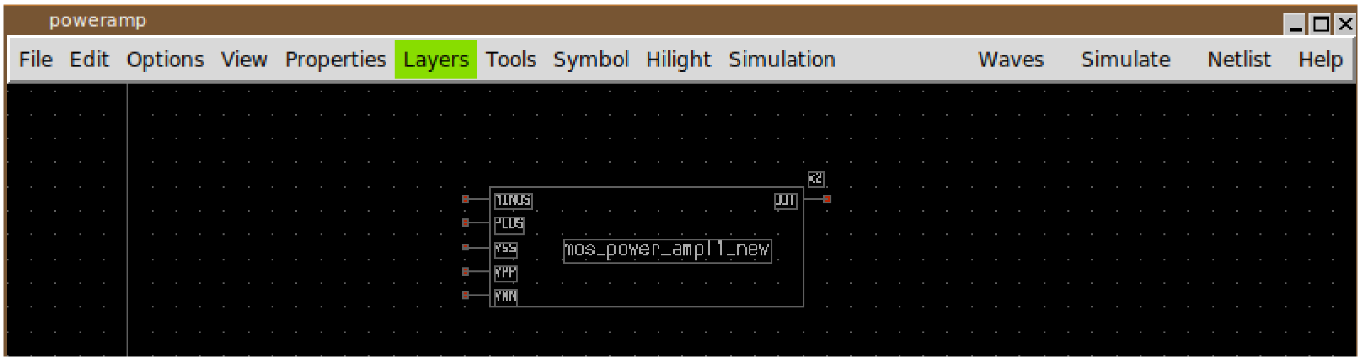

poweramp
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
x2
MINUS
PLUS
VSS
VPP
VNN
OUT
mos_power_ampli
ent2
Input property:
OK
Cancel
Load
Del

Symbol: examples/mos_power_ampli_new

[ ] no change properties [ ] preserve unchanged props [✓ copy cell]

```
name=x2
```

After pressing **`OK`** a copy (both schematic and symbol views) of the previously selected component will be created. After this clone operation modifications can be made on the newly created schematic and symbol views without affecting the original component.

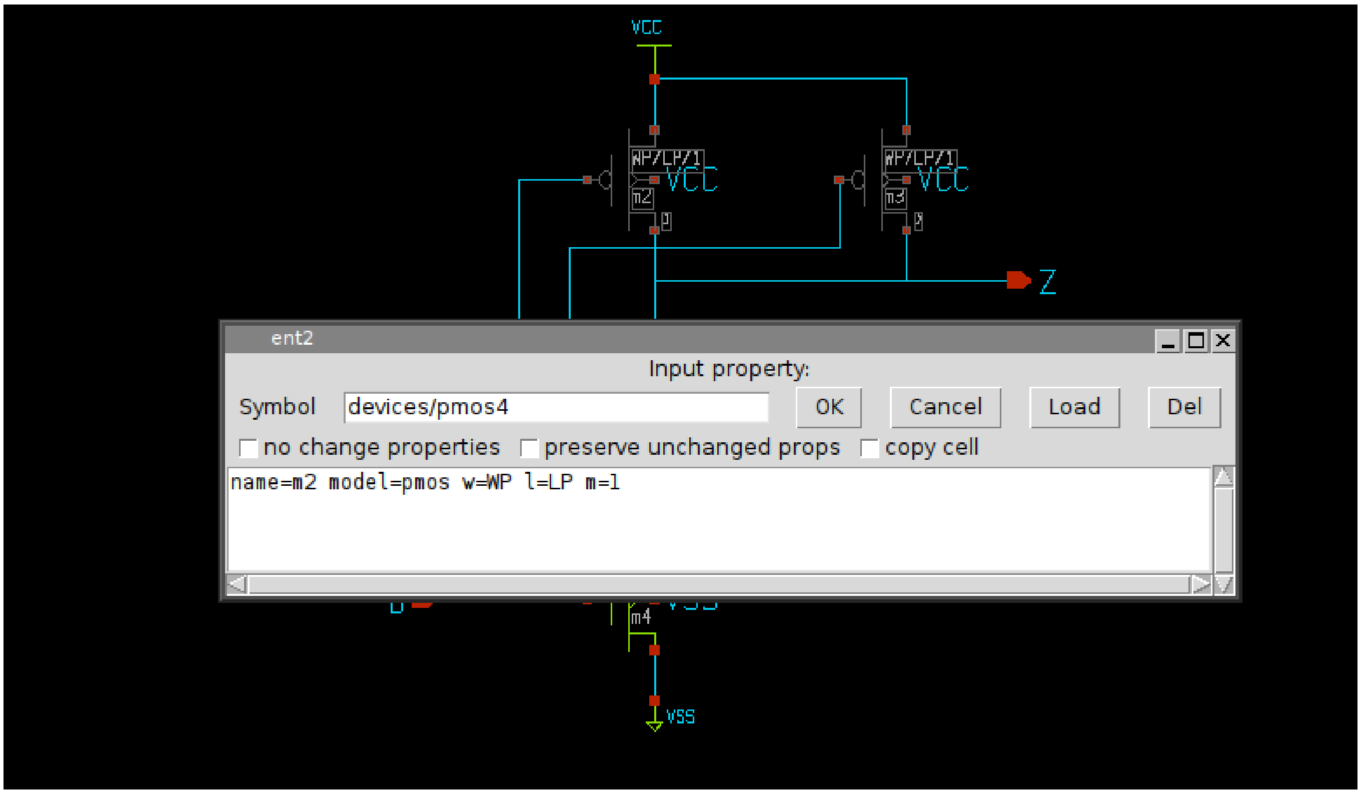

poweramp
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
MINUS
PLUS
VSS
VPP
VNN
OUT
x2
mos_power_ampli_new

for more info on symbols see the Tutorial

COMPONENT PARAMETERS

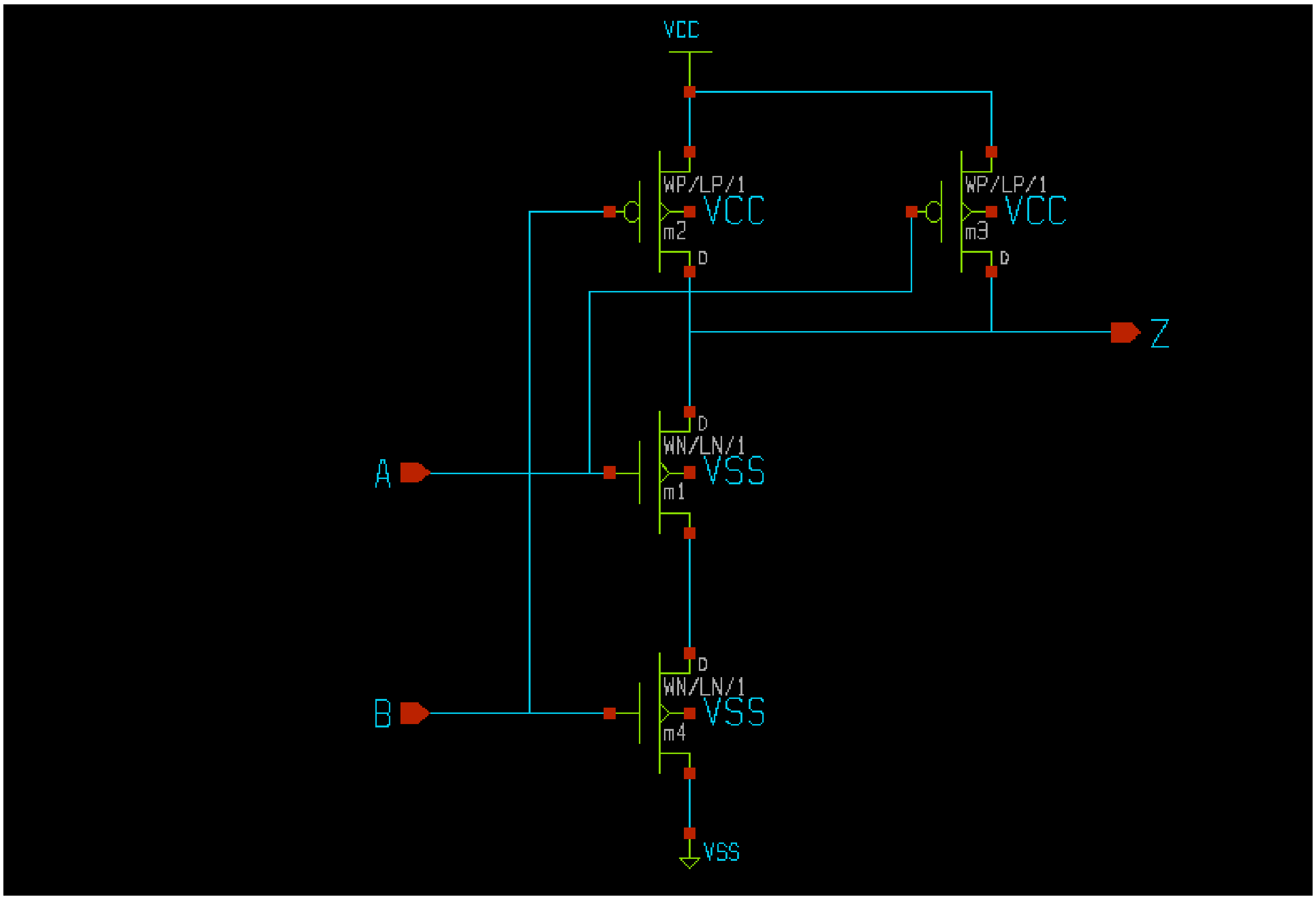
What makes subcircuits really useful is the possibility to pass parameters. Parametrized subcircuits are like functions with arguments in a programming language. One single component can be instantiated with different parameters. Recall the NAND2 gate we designed. It is made of four MOS transistors. A MOS transistor has at least 2 parameter, channel length (L) and transistor width (W) that define its geometry. we have 2 NMOS transistors and 2 PMOS transistors, so we would like to have 4 parameters passed to the NAND gate: P-channel with/length (WP/LP) and N-channel with/length (WN/LN). So open again the **`mylib/nand2.sch`** nand gate and replace the w=, l= properties with: **`w=WN l=LN`** for the two NMOS and **`w=WP l=LP`** for the two PMOS.

TIP: you can select two PMOS at the same time by clicking the second one with the **`shift`** key pressed, so with edit property **`'q'`** key you will change properties for both.

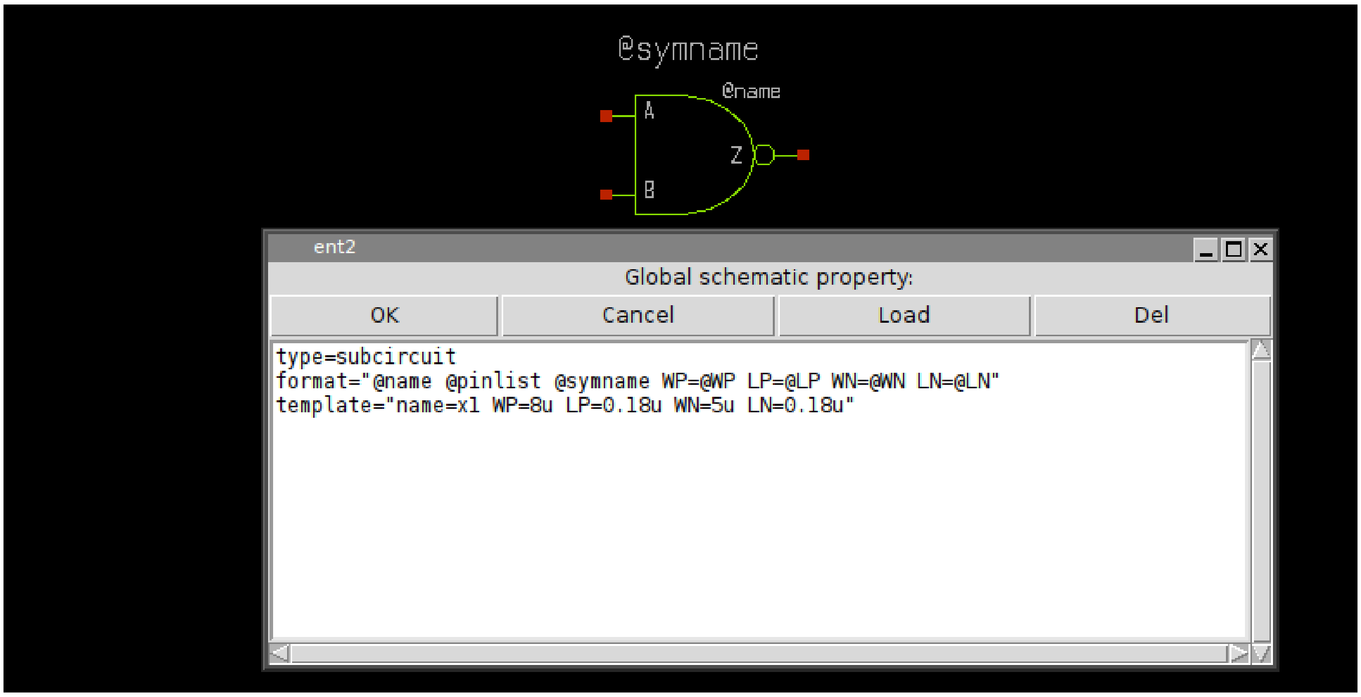


```
name=m2 model=pmos w=WP l=LP m=1
```

By doing the same for the NMOS transistors we end up with a schematic with fully parametrized transistor geometry.

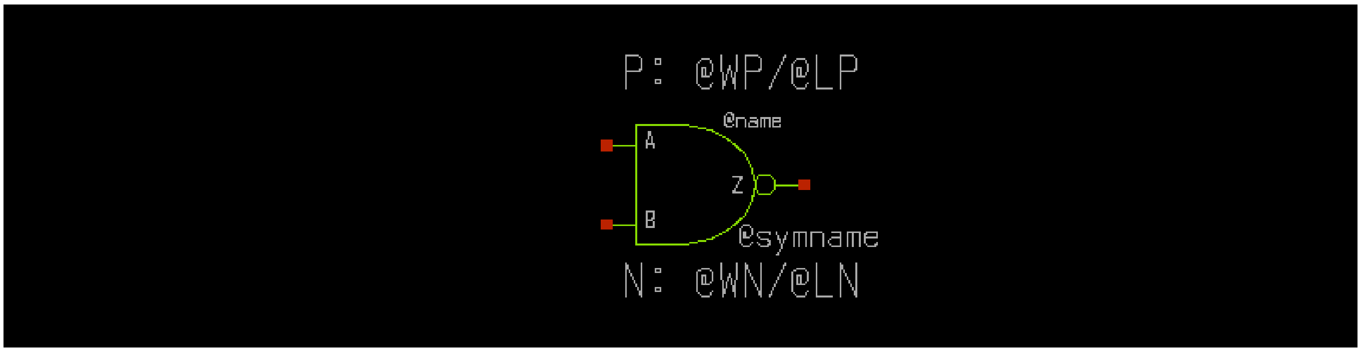


Now we have to change the **`mylib/nand2.sym`** symbol. Save the changes in the nand2 schematic (**`<shift>S`**) and load (**`Ctrl-o`**) the nand2 symbol. without selecting anything hit the **`'q'`** key to edit the symbol global property string. make the changes as shown in the picture.

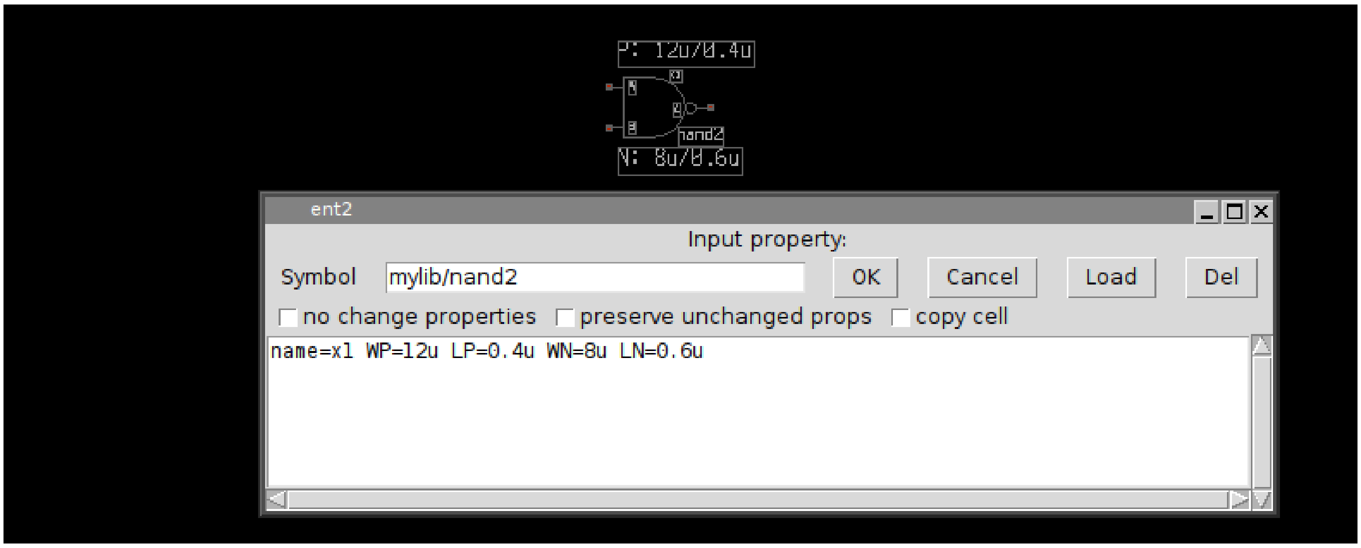


```
type=subcircuit
format="@name @pinlist @symname WP=@WP LP=@LP WN=@WN LN=@LN"
template="name=x1 WP=8u LP=0.18u WN=5u LN=0.18u"
```

The **`template`** attribute defines the default values to assign to WN, LN, WP, LP. The **`format`** string is updated to pass parameters, the replacement character `@` is used to substitute the parameters passed at component instantiation. You may also add some descriptive text (**`'t'`**) so you will visually see the actual value for the parameters of the component:



Now close the modified symbol saving the changes. Let's test the placement of the new modified symbol. Start a new schematic (menu **`File -> New`**) and insert (**`Insert key`**) the NAND2 gate. by pressing **`'q'`** you are now able to specify different values for the geometric parameters:

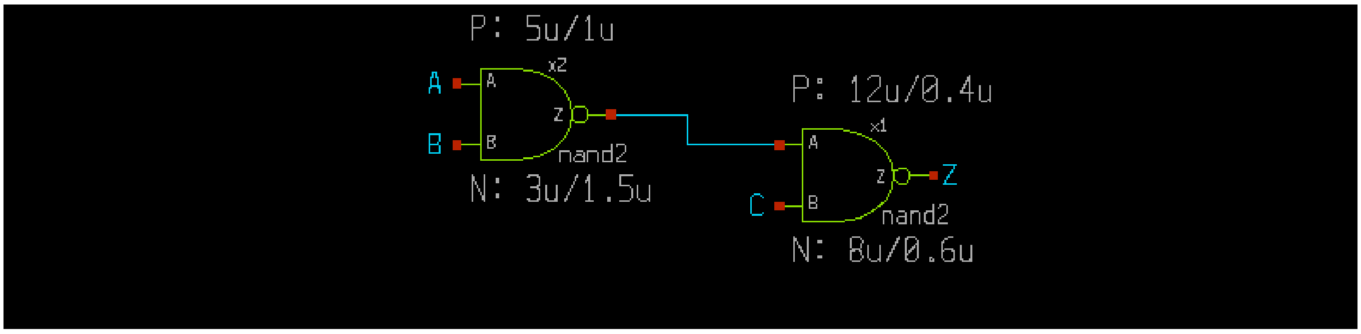


mylib/nand2

```
name=x1 WP=12u LP=0.4u WN=8u LN=0.6u
```

let's place a second instance (select and **`'c'`** copy key) of the nand gate. set for the second NAND gate different WN, LN, WP, LP parameters. place some labels on input and outputs and connect the output of the first NAND gate to one of the inputs of the second NAND gate. Name the pin labels as in the picture using the edit property **`'q'`** key on selected **`lab_pin`** instance

TIP: XSCHEM can automatically place pin labels on a component: just select it and press the **`Shift-h`** key.



now save the new schematic (**`'s'`** key, save in **`mylib/test2.sch`**) If you enable the netlist window, menu **`Options->Show netlist win`** and press the **`Netlist`** button in the menu bar you get the following netlist:

```
**.subckt test2
x1 Z net1 C nand2 WP=12u LP=0.4u WN=8u LN=0.6u
x2 net1 A B nand2 WP=5u LP=1u WN=3u LN=1.5u
**** begin user architecture code
**** end user architecture code
**.ends

* expanding symbol: mylib/nand2 # of pins=3

.subckt nand2 Z A B WP=8u LP=0.18u WN=5u LN=0.18u
*.ipin A
*.opin Z
*.ipin B
m1 Z A net1 VSS nmos w=WN l=LN m=1
m2 Z B VCC VCC pmos w=WP l=LP m=1
m3 Z A VCC VCC pmos w=WP l=LP m=1
m4 net1 B VSS VSS nmos w=WN l=LN m=1
**** begin user architecture code
**** end user architecture code
.ends

.GLOBAL VCC
.GLOBAL VSS
.end
```

As you can see there are 2 components placed passing parameters to a **`nand2`** subcircuit. There is complete freedom in the number of parameters. Any kind parameters can be used in subcircuits as long as the simulator permits these.

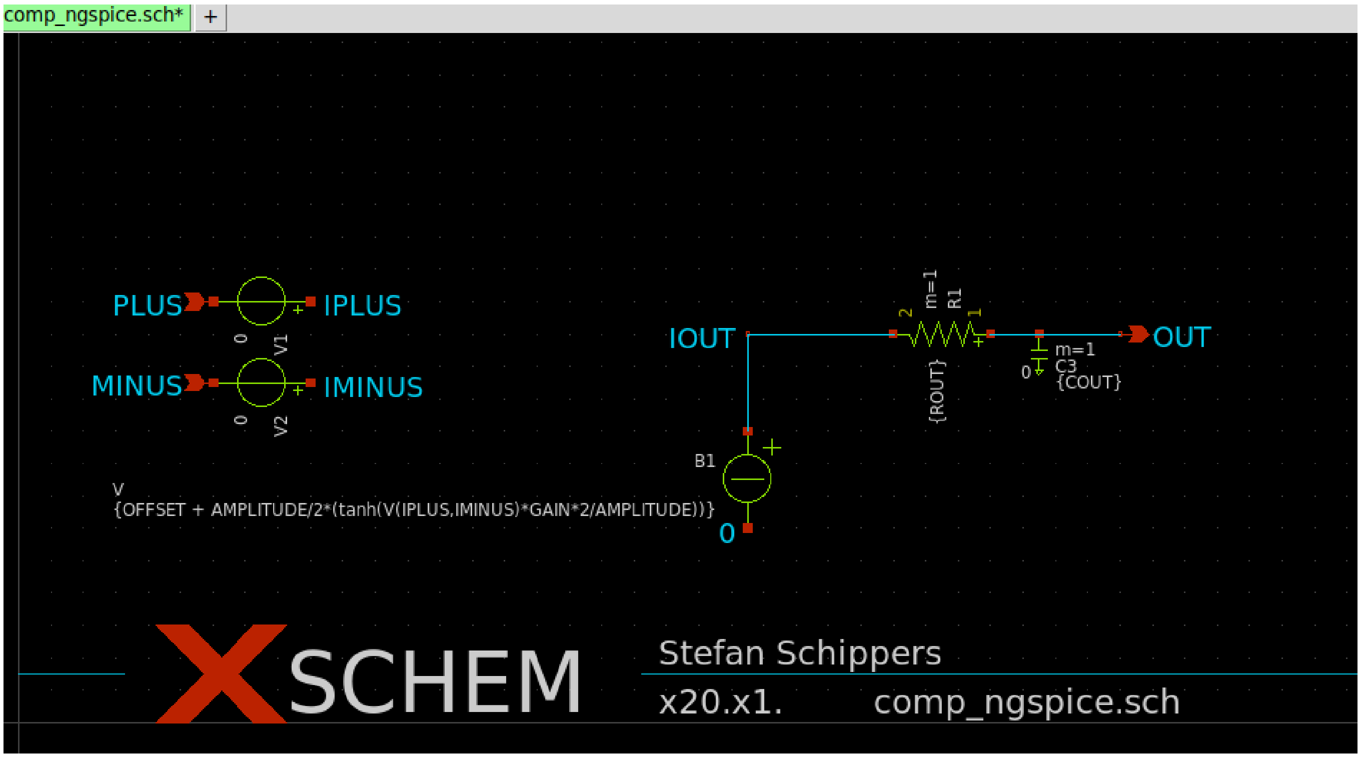
CREATING A PARAMETRIC SUBCIRCUIT

Let's suppose we want to design an OPAMP macromodel taking the following parameters:

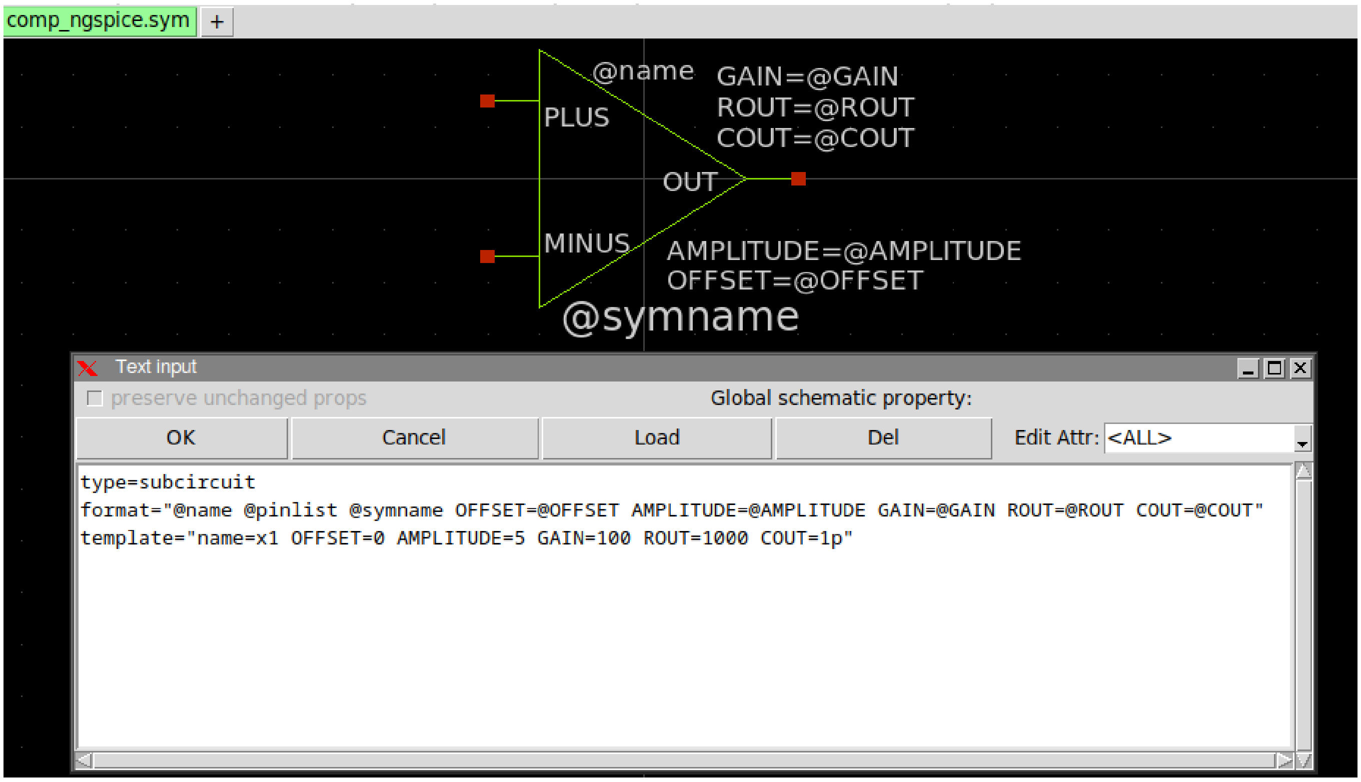
- GAIN: The differential maximum small signal gain of the opamp.
- AMPLITUDE: The peak to peak swing of the opamp output.
- OFFSET: the offset of the output when input differential signal is zero.
  For example giving AMPLITUDE=10 and OFFSET=5 will result in an output swing from 0 to +10V.
- ROUT: the output resistance.
- COUT: the output capacitance. Together with ROUT defines a RC time constant (dominant pole).

Parameters and expressions should be enclosed in curly braces or single quotes:
`value={ROUT}` or **`value='ROUT'`**

The image below shows the circuit. A 'B' voltage-type source with an hyperbolic tangent function is used because it has continuous derivative and a realistic shape.



after drawing the schematic a symbol is created. The easiest way is to press the 'a' key in the schematic to automatically create the symbol, then descend into the symbol and do some artwork to reshape it to represent an opamp.
After reshaping the symbol edit its global attributes and add handling of subcircuit parameters in the **`format`** and **`template`** attributes as shown below:



```
type=subcircuit
format="@name @pinlist @symname OFFSET=@OFFSET AMPLITUDE=@AMPLITUDE GAIN=@GAIN ROUT=@ROUT COUT=@COUT"
template="name=x1 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p"
```

the symbol has the following global attributes:

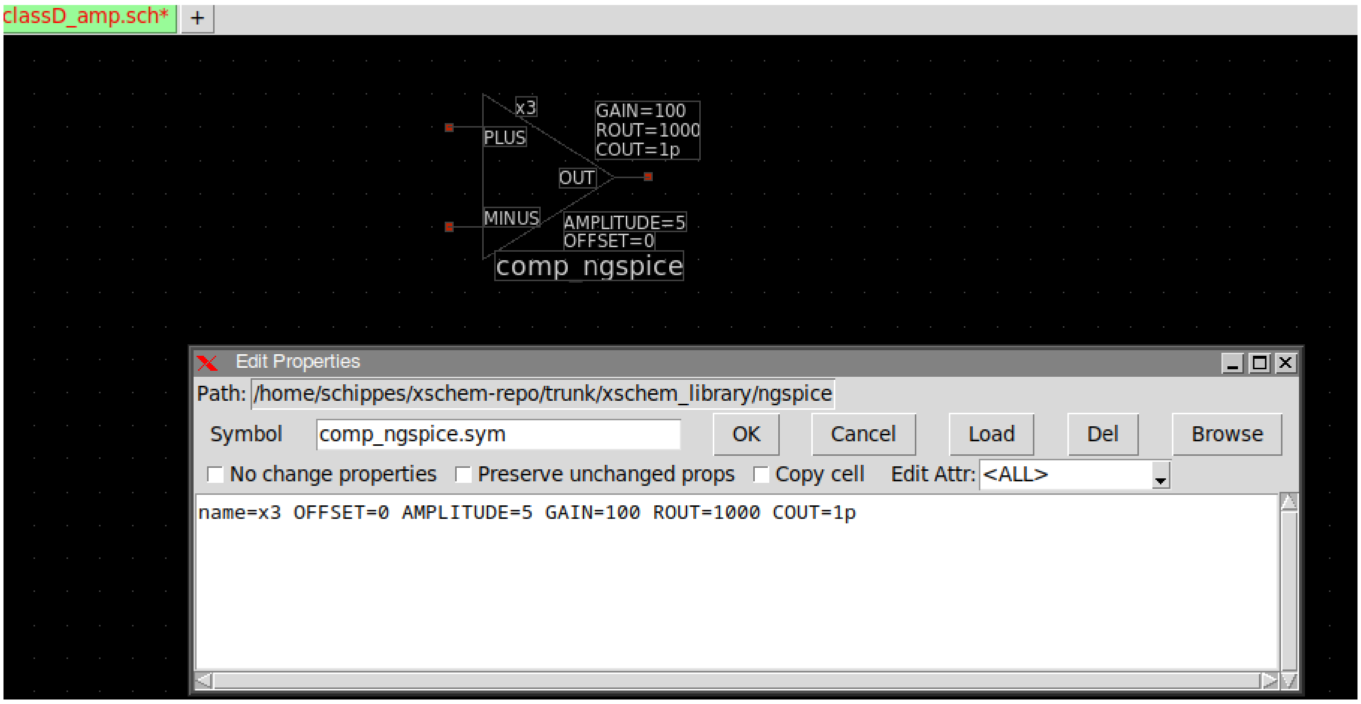
```
type=subcircuit
format="@name @pinlist @symname OFFSET=@OFFSET AMPLITUDE=@AMPLITUDE GAIN=@GAIN ROUT=@ROUT COUT=@COU
template="name=x1 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p"
```

The **`format`** string defines how the instantiation will look in the spice netlist, The following is the resulting spice netlist line and how it is generated from the format string:

```
  x1 REFD DRIVED IN comp_ngspice OFFSET=5 AMPLITUDE=10 GAIN=100 ROUT=1000 COUT=1p
  -- -------------- ------------ -------- ------------ -------- --------- -------
  |        |             |         |           |          |         |         |
  |        |             |         |           |          |         |     COUT=@COUT
  |        |             |         |           |          |     ROUT=@ROUT
  |        |             |         |           |      GAIN=@GAIN
  |        |             |         |           |
  |        |             |         |  AMPLITUDE=@AMPLITUDE
  |        |             |         |
  |        |             |   OFFSET=@OFFSET
  |        |        @symname
  |        |
  |      @pinlist
@name
```

The **`template`** string defines initial values for these parameters when you first instantiate this component:

```
template="name=x1 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p"
```



Path: /home/schippes/xschem-repo/trunk/xschem_library/ngspice

Symbol comp_ngspice.sym

```
name=x3 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p
```

As you can see in above image the placed component has instance parameters set to the same values listed in the **`template`** string. You may then change these values according yo your needs. The values set in the instance affect that specific component instance behavior. Multiple instances can be placed, each with it's own set of parameter values. As you can see the @param values in the format string are replaced with the actual value set in the instance attributes.

When a netlist is generated the following lines are generated regarding this comp_ngspice symbol:

```
* sch_path: /home/schippes/xschem-repo/trunk/xschem_library/examples/classD_amp.sch
...
... other components ....
...
x1 REFD DRIVED IN comp_ngspice OFFSET=5 AMPLITUDE=10 GAIN=100 ROUT=1000 COUT=1p
...
...
*  expanding   symbol:  comp_ngspice.sym # of pins=3
** sym_path: /home/schippes/xschem-repo/trunk/xschem_library/ngspice/comp_ngspice.sym
** sch_path: /home/schippes/xschem-repo/trunk/xschem_library/ngspice/comp_ngspice.sch
.subckt comp_ngspice PLUS OUT MINUS  OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p
*.ipin PLUS
*.ipin MINUS
*.opin OUT
B1 IOUT 0 V = {OFFSET + AMPLITUDE/2*(tanh(V(IPLUS,IMINUS)*GAIN*2/AMPLITUDE))}
R1 OUT IOUT {ROUT} m=1
C3 OUT 0 {COUT} m=1
V1 IPLUS PLUS 0
.save i(v1)
V2 IMINUS MINUS 0
.save i(v2)
.ends
...
...
.end
```

You see the **`.subckt`** line contains the subcircuit parameters and their values: The values present in the **`.subckt`** line are overridden by instance attribute values if given.

```
.subckt comp_ngspice PLUS OUT MINUS  OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p
        ------------ --------------  ----------------------------------------------
             |             |                              |
             |             |                              |
             |             |               parameters from template attributes:
             |             |                              |
             |             |                  _______________________________________________
             |             |    template="name=x1 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p"
             |         port list
        symbol name
```

EDITOR COMMANDS

Most editing commands are available in the menu, but definitely key-bindings and Mouse actions are the most effective way to build and arrange schematics, so you should learn at least the most important ones.

INTUITIVE INTERFACE

This is a recent interface that uses less keyboard commands and more mouse actions It is therefore possible to click and drag objects directly and do many more actions by just using mouse actions. This interface is enabled by enabling the menu checkbutton:
 `Options -> Intuitive click & Drag Interface`
Or by adding the following line:
 `set intuitive_interface 1`
in your **`xschemrc`** file.

The following cheatsheet image shows the intuitive_interface commands

The standard interface is described below. All below description applies also if intuitive_interface is enabled.

STANDARD INTERFACE

The basic principle in XSCHEM is that first you select something in the circuit then you decide what to do with the selection. For example, if you need to change an object property you first select it (mouse click) and then you press the edit property (**`'q'`**) key. It you need to move together multiple objects you select them (by area or using multiple mouse clicks with the **`Shift`** key), then you press the move (**`'m'`**) key.

EDITOR COMMAND CHEATSHEET

This list is available in XSCHEM in the **`Help`** menu

```
                        XSCHEM MOUSE BINDINGS
----------------------------------------------------------------------
LeftButton              Clear selection and select a graphic object
                        (line, rectangle, symbol, wire)
                        if clicking on blank area: clear selection

shift + LeftButton      Select without clearing previous selection

ctrl + LeftButton       if an 'url' or 'tclcommand' property is defined on
                        selected instance open the url or execute the
                        tclcommand

LeftButton drag         Select objects by area, clearing previous selection
                        "[shift] left button drag" and "[shift] ctrl-left
                        button drag" commands are swapped if enable_stretch
                        is set.
```

```
Ctrl + LeftButton drag  Select objects by area to perform a
                        subsequent   'stretch'  move operation

shift + LeftButton drag Select objects by area, without clearing
                        previous selection

Shift +                 Select objects by area without unselecting
Ctrl + LeftButton drag  to perform a subsequent   'stretch'  move operation

Shift + RightButton     Select all connected wires/labels/pins

Ctrl + RightButton      Select all connected wires/labels/pins, stopping at
                        wire junctions

Alt + RightButton       Cut wire at mouse position (creates 2 adjacent wires)
                        aligns the cut point to current snap setting.

Alt + Shift + RightButton
                        Cut wire at mouse position (creates 2 adjacent wires)
                        does not align cut point to current snap setting.


Mouse Wheel             Zoom in / out

MidButton drag          Pan viewable area

Alt + LeftButton        Unselect selected object

Alt + LeftButton drag
                        Unselect objects by area

RightButton drag        Zoom area

RightButton Release     Context menu

LeftButton Double click Terminate Polygon placement
                        Edit object attributes


                        XSCHEM KEY BINDINGS
----------------------------------------------------------------------
-          BackSpace    Back to parent schematic
-          Delete       Delete selected objects
-          Insert       Insert element from library
-          Print Scrn   Grab screen area
-          Escape       Abort, redraw, unselect
ctrl       Enter        Confirm closing dialog boxes
-          Down         Move down
-          Left         Move right
-          Right        Move left
-          Up           Move up
ctrl       Left         Previous tab (if tabbed interface enabled)
ctrl       Right        Next tab (if tabbed interface enabled)
-           '\'         Toggle fullscreen
-           '!'         Break selected wires at any wire or component pin
                        connection
-           ' '         Pan schematic
-           ' '         When drawing lines or wires toggle between
                        manhattan H-V, manhattan V-H or oblique path.
-           '#'         Highlight components with duplicated name (refdes)
ctrl        '#'         Rename components with duplicated name (refdes)
-           '5'         View only probes
ctrl        '0-9'       set current layer (4 -13)
            '0'         set selected net or label to logic value '0'
```

	'1'	set selected net or label to logic value '1'
	'2'	set selected net or label to logic value 'X'
	'3'	set selected net or label to logic value 'Z'
	'4'	toggle selected net or label: 1->gt;0, 0->gt;1, X->gt;X
-	'a'	Make symbol from pin list of current schematic
ctrl	'a'	Select all
shift	'A'	Toggle show netlist
-	'b'	Merge file
Shift	'B'	Edit/add header/license metadata to the schematic/symbol file.
ctrl	'b'	Toggle show text in symbol
alt	'b'	Toggle show symbol details / only bounding boxes
-	'c'	Copy selected objects, 'c' and 'alt-c' commands are swapped if enable_stret
Alt	'c'	Copy selected objects, insert wires when separating touching instance pins,
ctrl	'c'	Save to clipboard
shift	'C'	Start arc placement
shift+ctrl	'C'	Start circle placement
alt	'C'	Toggle dim/brite background with rest of layers
shift	'D'	Delete files
ctrl	'e'	Back to parent schematic
-	'e'	Descend to schematic
alt	'e'	Edit selected schematic in a new window
	'\'	Toggle Full screen
shift	'F'	Horizontal flip selected objects
alt	'f'	Horizontal flip selected objects around their anchor points
ctrl	'f'	Find/select by substring or regexp
-	'f'	Full zoom
shift+ctrl	'F'	Zoom full selected elements
shift	'G'	Double snap factor
-	'g'	Half snap factor
ctrl	'g'	Set snap factor
alt	'g'	Hilight selected nets and send to gaw waveform viewer
-	'h'	Constrained horizontal move/copy of objects
alt	'h'	create symbol pins from schematic pins
ctrl	'h'	Follow http link or execute command (url, tclcommand properties)
shift	'H'	Attach net labels to selected instance
ctrl+shift	'H'	Make schematic and symbol from selected components
-	'i'	Descend to symbol
alt	'i'	Edit selected symbol in a new window
alt+shift	'J'	Create labels with 'i' prefix from highlighted nets/pins
alt	'j'	Create labels without 'i' prefix from highlighted nets/pins
ctrl	'j'	Create ports from highlight nets
alt+ctrl	'j'	Print list of highlighted nets/pins with label expansion
shift	'J'	create xplot plot file for ngspice in simulation directory
		(just type xplot in ngspice)
-	'j'	Print list of highlighted nets/pins
-	'k'	Hilight selected nets
ctrl+shift	'K'	highlight net passing through elements with 'propag' property set on pins
shift	'K'	Unhilight all nets
ctrl	'k'	Unhilight selected nets
alt	'k'	Select all nets attached to selected wire / label / pin.
-	'l'	Start line
ctrl	'l'	Make schematic view from selected symbol
alt+shift	'l'	add lab_wire.sym to schematic
alt	'l'	add lab_pin.sym to schematic
ctrl+shift	'o'	Load most recent schematic
ctrl	'o'	Load schematic
-	'm'	Move selected objects. 'm' and 'ctrl-m' commands are swapped if enable_stre
ctrl	'm'	Move selected objects, stretching wires attached to them
Alt	'm'	Move selected objects, insert wires when separating touching instance pins,
shift	'M'	Move selected objects, insert wires when separating touching instance pins,
ctrl+shift	'M'	Move selected objects, combine Shift-M and Ctrl-m
shift	'N'	Top level only netlist
-	'n'	Hierarchical Netlist
ctrl	'n'	Clear schematic

ctrl+shift	'N'	Clear symbol
shift	'O'	Toggle Light / Dark colorscheme
ctrl	'o'	Load schematic
-	'p'	Place polygon. Operation ends by placing last point over first.
alt	'p'	Add symbol pin
ctrl	'p'	Pan schematic view
shift	'P'	Pan, other way to.
alt	'q'	Edit schematic file (dangerous!)
-	'q'	Edit prop
shift	'Q'	Edit prop with vim
ctrl+shift	'Q'	View prop
ctrl	'q'	Exit XSCHEM
alt	'r'	Rotate objects around their anchor points
shift	'R'	Rotate
-	'r'	Start rect
shift	'S'	Change element order
ctrl+shift	'S'	Save as schematic
-	's'	run simulation (asks confirmation)
ctrl	's'	Save schematic
alt	's'	Reload current schematic from disk
ctrl+alt	's'	Save-as symbol
-	't'	Place text
shift	'T'	Toggle *_ignore flag on selected instances
alt	'u'	Align to current grid selected objects
shift	'U'	Redo
-	'u'	Undo
-	'v'	Constrained vertical move/copy of objects
shift	'V'	Vertical flip selected objects
alt	'v'	Vertical flip selected objects around their anchor point
ctrl	'v'	Paste from clipboard
ctrl+shift	'V'	Toggle spice/vhdl/verilog netlist
-	'w'	Place wire
ctrl	'w'	close current schematic
shift	'W'	Place wire, snapping to closest pin or net endpoint
ctrl	'x'	Cut into clipboard
-	'x'	New cad session
shift	'X'	Highlight discrepancies between object ports and attached nets
alt	'x'	Toggle draw crosshair at mouse position
-	'y'	Toggle stretching wires
-	'z'	Zoom box
shift	'Z'	Zoom in
ctrl	'z'	Zoom out
-	'?'	Help
-	'&'	Join / break / collapse wires
shift	'*'	Postscript/pdf print
ctr+shift	'*'	Xpm/png print
alt+shift	'*'	Svg print
	'-'	dim colors
ctrl	'-'	Test mode: change line width
ctrl	'+'	Test mode: change line width
	'+'	brite colors
-	'_'	Toggle change line width
-	'%'	Toggle draw grid
ctrl	'='	Toggle fill rectangles
-	'$'	Toggle pixmap saving
ctrl	'$'	Toggle use XCopyArea vs drawing primitives for drawing the screen
-	':'	Toggle flat netlist

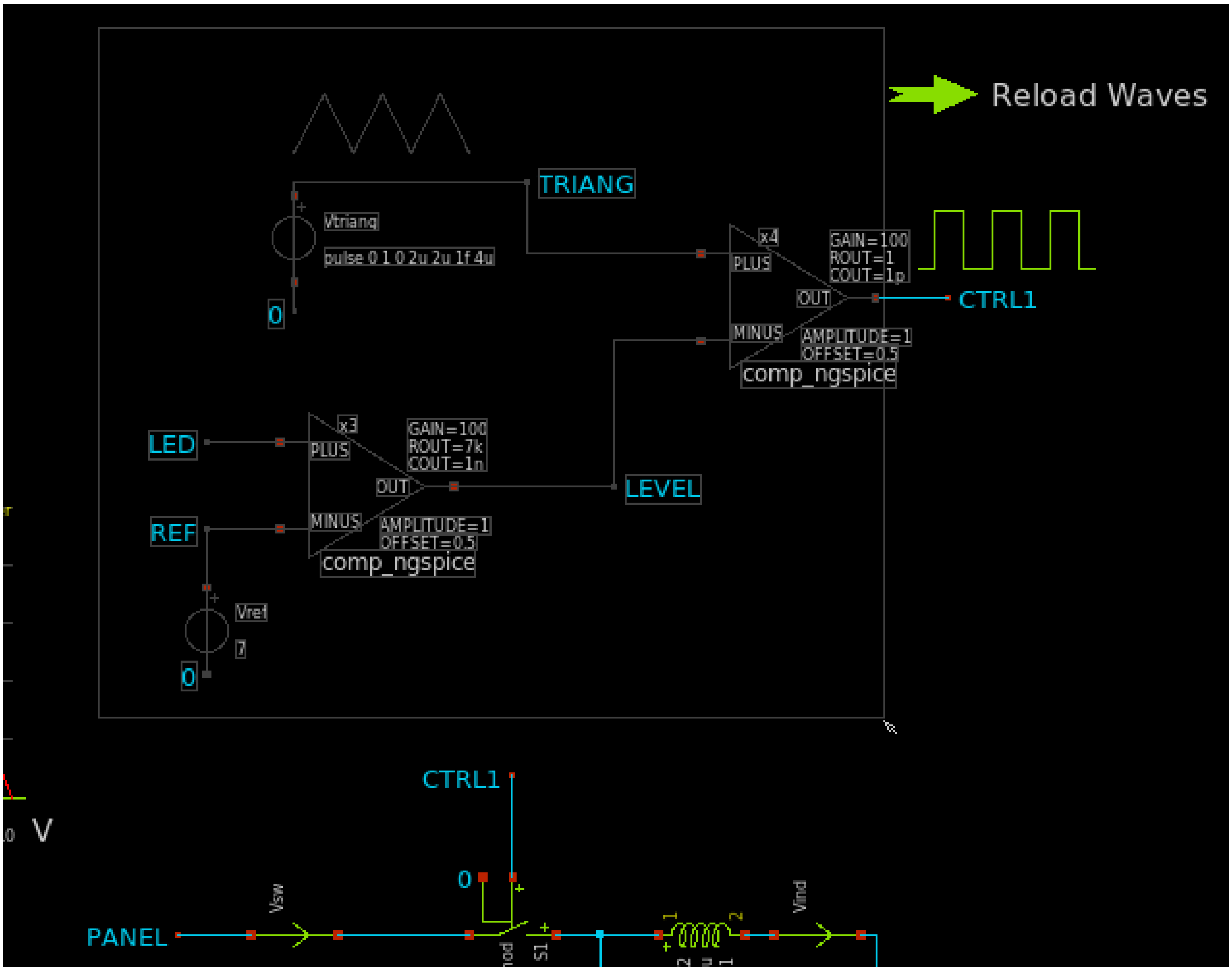
KEYBIND CUSTOMIZATION

changes to default keybindings may be placed in the **`~/.xschem/xschemrc`** file as in the following examples:

```
## replace Ctrl-d with Escape (so you won't kill the program :-))
set replace_key(Control-d) Escape
## swap w and W keybinds; Always specify Shift for capital letters
set replace_key(Shift-W) w
set replace_key(w) Shift-W
```

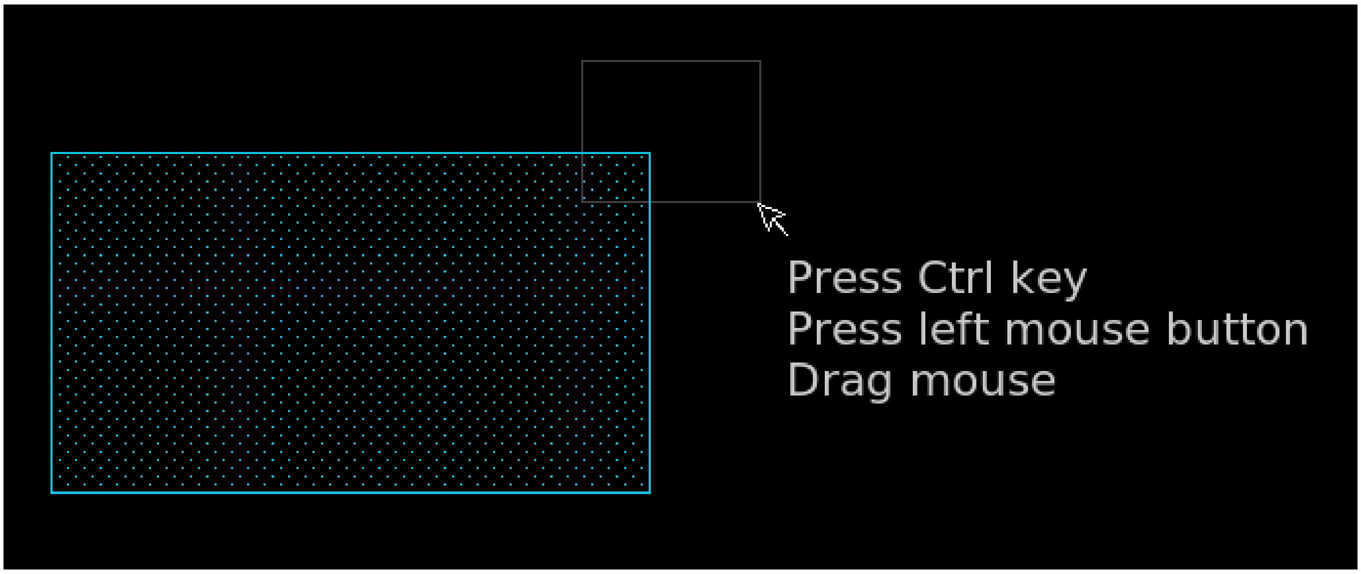
SELECT OBJECTS

Objects can be selected by clicking the left mouse button when the pointer is very close to the object. For rectangle objects the best point to select it is the internal side, close to one of the corners. More objects can be selected by pressing the **`Shift`** key and clicking another object. Once objects are selected they can be copied (**`c`** key), moved (**`m`** key), deleted (**`Delete`** key) or attributes changed (**`q`** key).
Objects can also be selected by area, by dragging with the left mouse button pressed a rectangle around the objects you want to select.

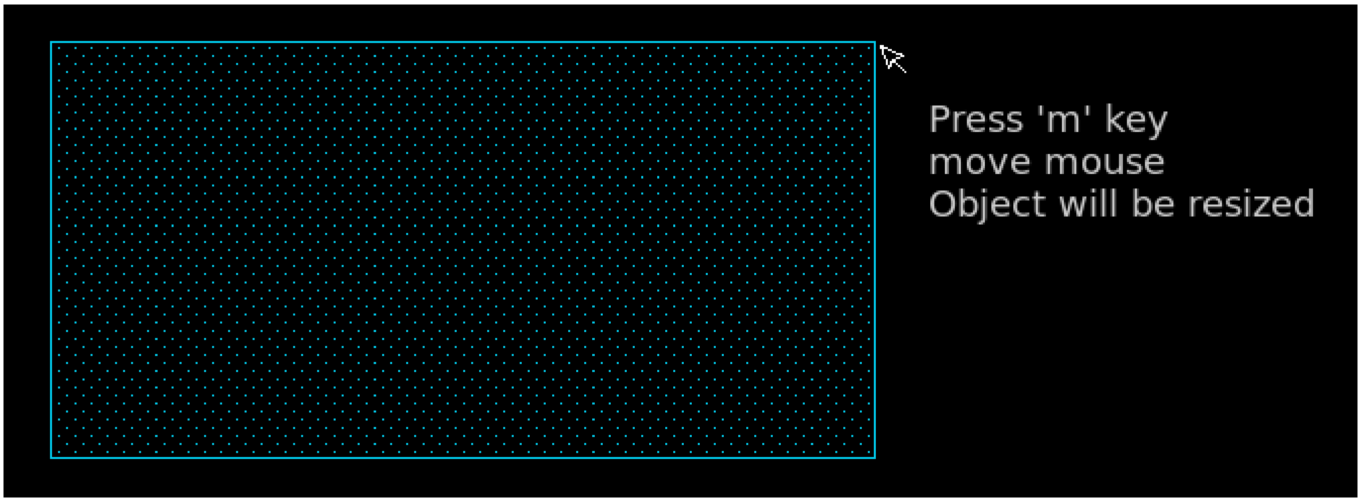


RESIZE OBJECTS

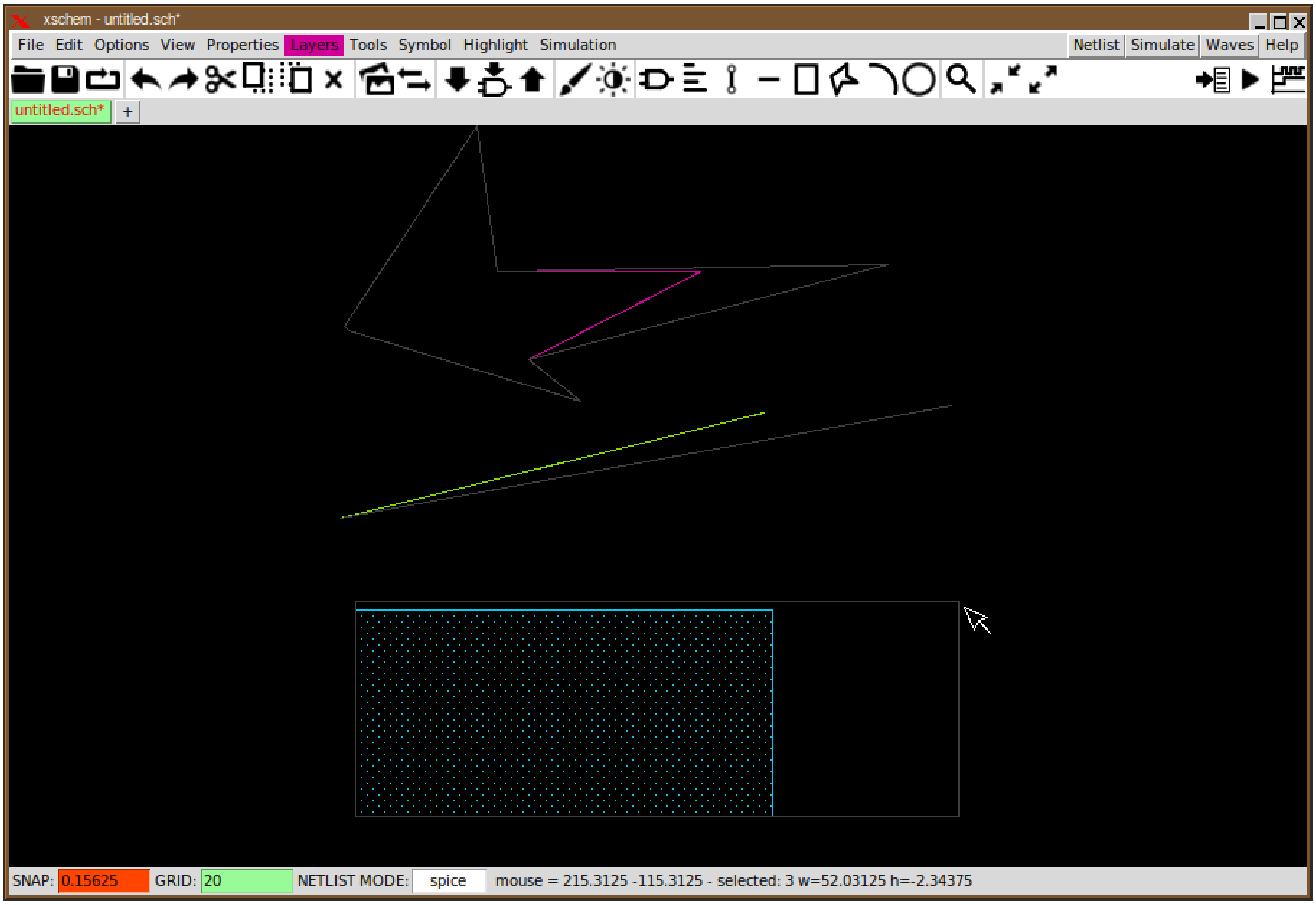
All Xschem base objects can be resized. For lines, rectangles, polygons you need to drag the mouse with left button pressed and Ctrl key pressed over one vertex/endpoint.



After releasing the mouse button the object will become selected and a subsequent move operation (`m` key) will move the selected vertex/endpoint.



More objects can be resized. You can add vertex/endpoints by pressing Ctrl and Shift and dragging the mouse to enclose another vertex/endpoint. After selecting all desired elements pressing the `m` key will resize all objects.

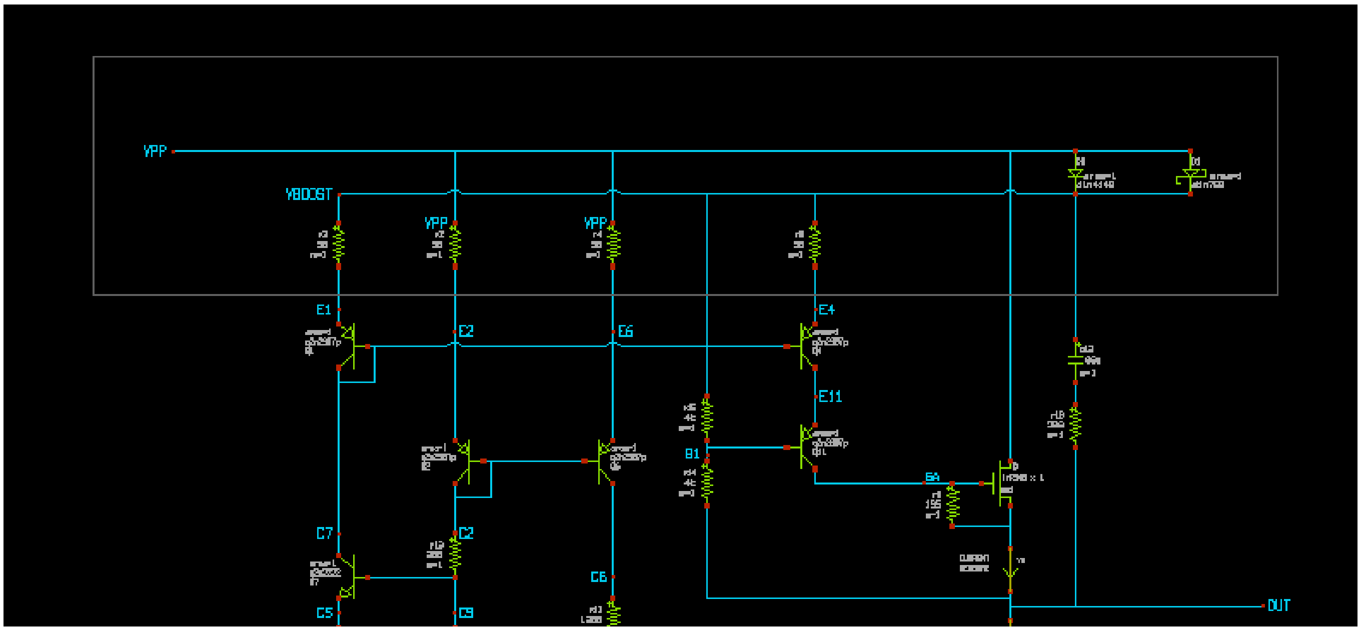

xschem - untitled.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
untitled.sch*
+

SNAP: 0.15625 GRID: 20 NETLIST MODE: spice mouse = 215.3125 -115.3125 - selected: 3 w=52.03125 h=-2.34375

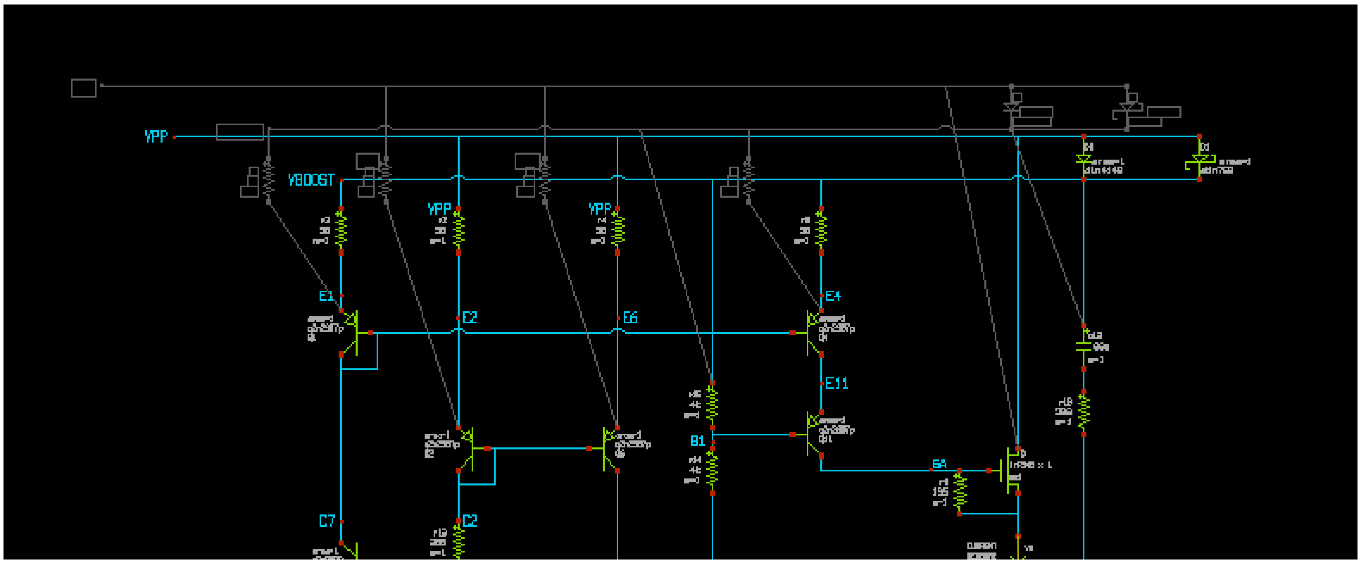
Circles can be resized as well. Capture the center of the circle with the above described mouse drag operation, the radius can be changed. For arcs you can capture the center (to modify the radius) or the endpoints to change the start/end angle or the arc angle.

STRETCH OPERATIONS

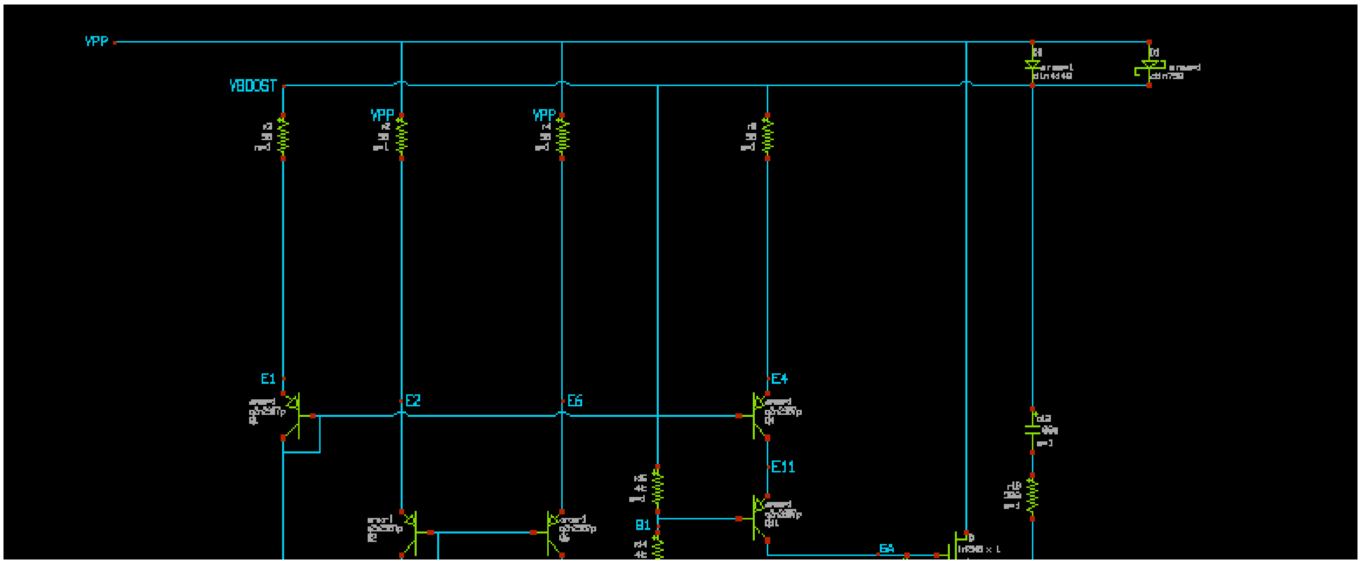
An important operation that deserves a special paragraph is the **`Stretch`** operation. There is frequently the need to move part of the circuit without breaking connections, for example to create more room for other circuitry or just to make it look better. The first thing to do is to drag a selection rectangle with the mouse holding down the **`Ctrl`** key, cutting wires we need to stretch:



After selection is done hit the move (`'m'`) key. You will be able to move the selected part of the schematic keeping connected the wires crossing the selection rectangle:

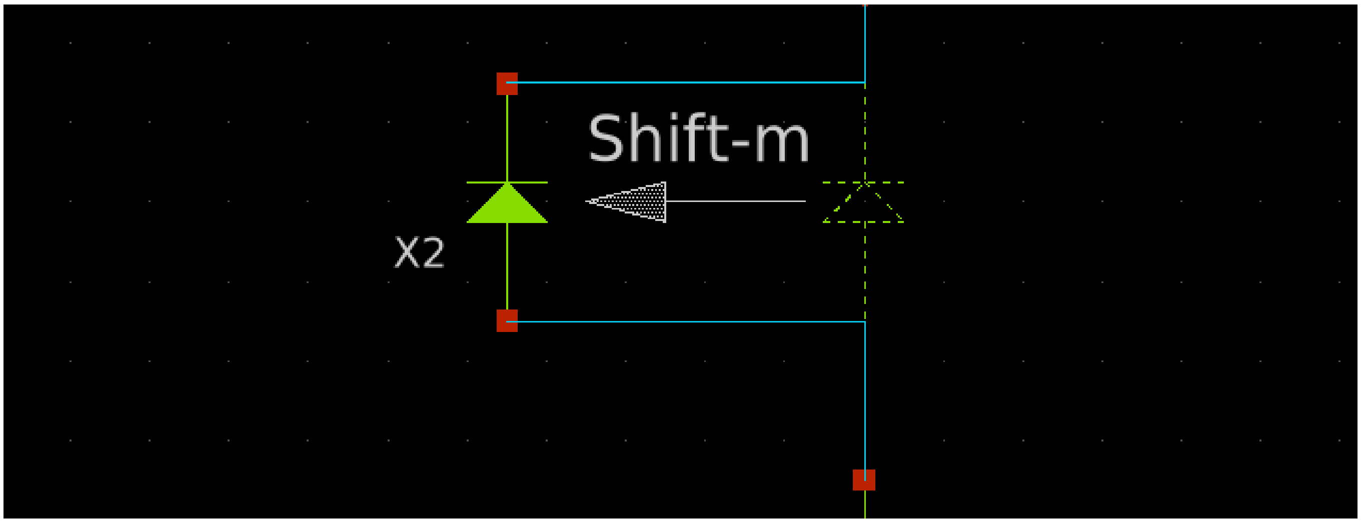


In our example we needed to move up part of the circuit, the end result is shown in next picture. Multiple stretch rectangles can be set using the **`Shift`** key in addition to the **`Ctrl`** key after setting the first stretch area.



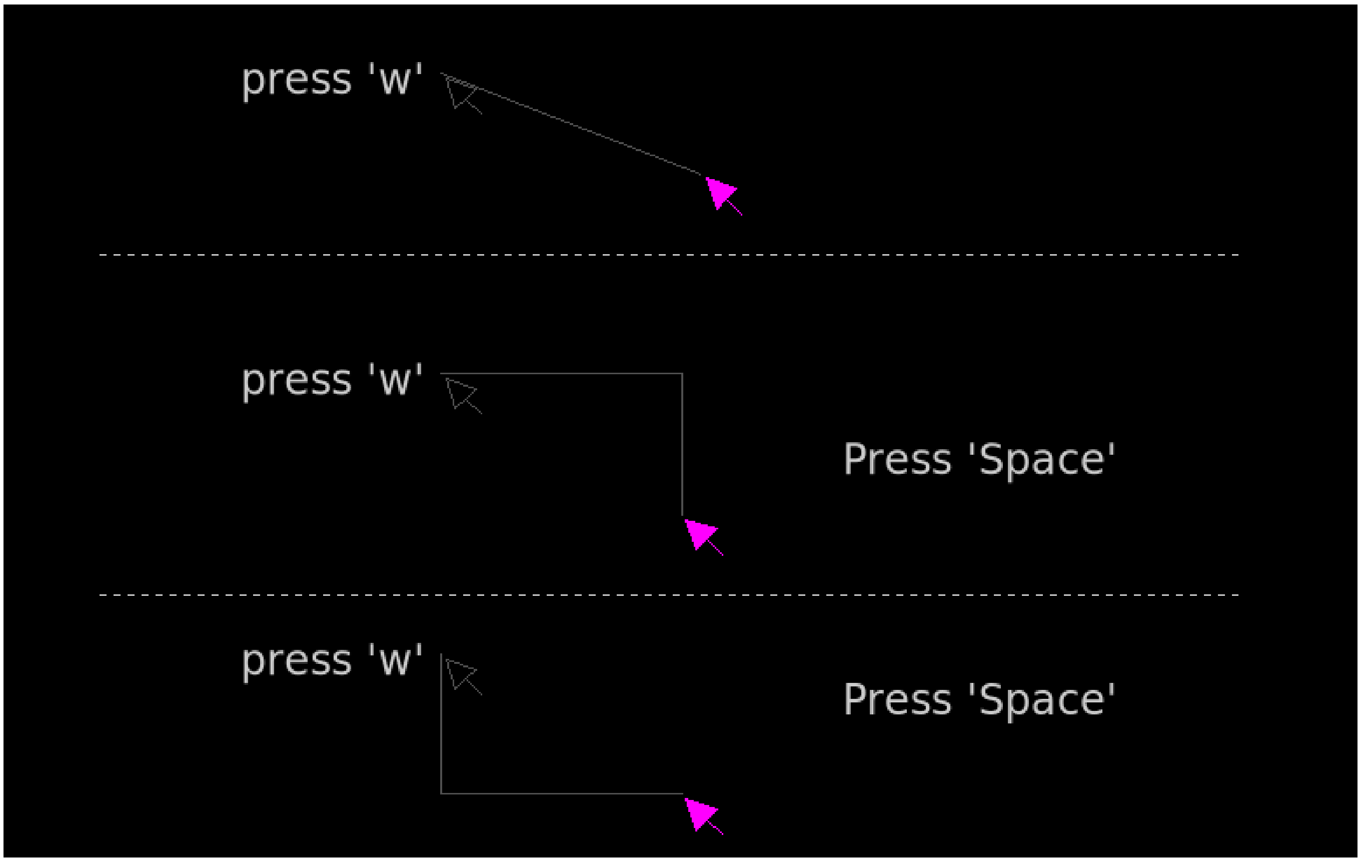
Another way to move objects stretching attached wires is to press **`Ctrl-m`** instead of **`m`** This way you don't have to remember to press **`Ctrl`** when doing the selection.

Pressing **`Shift-m`** instead of **`m`** will create new wires while moving the selected objects.



PLACE, WIRES MANHATTAN PATHS

When you press the **`w`** key a wire placement begins. Moving the mouse a rubber wire is displayed. Clicking the left mouse button will end the wire placement. If the space bar is pressed you toggle between Horizontal-Vertical, Vertical-Horizontal and oblique placement mode.

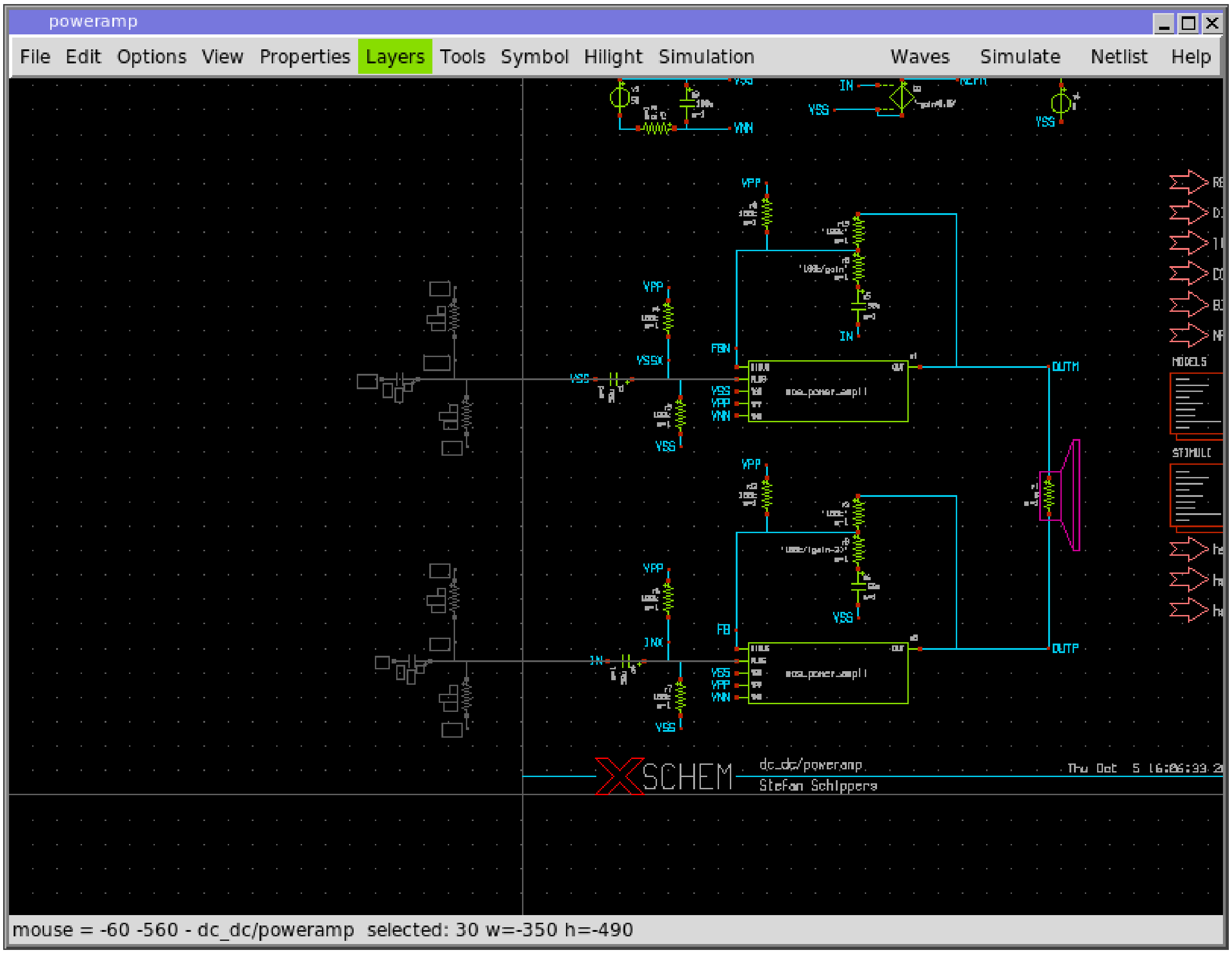


PLACE WIRES SNAPPING TO CLOSEST PIN OR NET ENDPOINT

The (uppercase) **`'W'`** bindkey allows to place a wire putting start (and end point, later) to the closest pin or wire endpoint, this will make it easier to connect precisely without the need to zoom in all times.

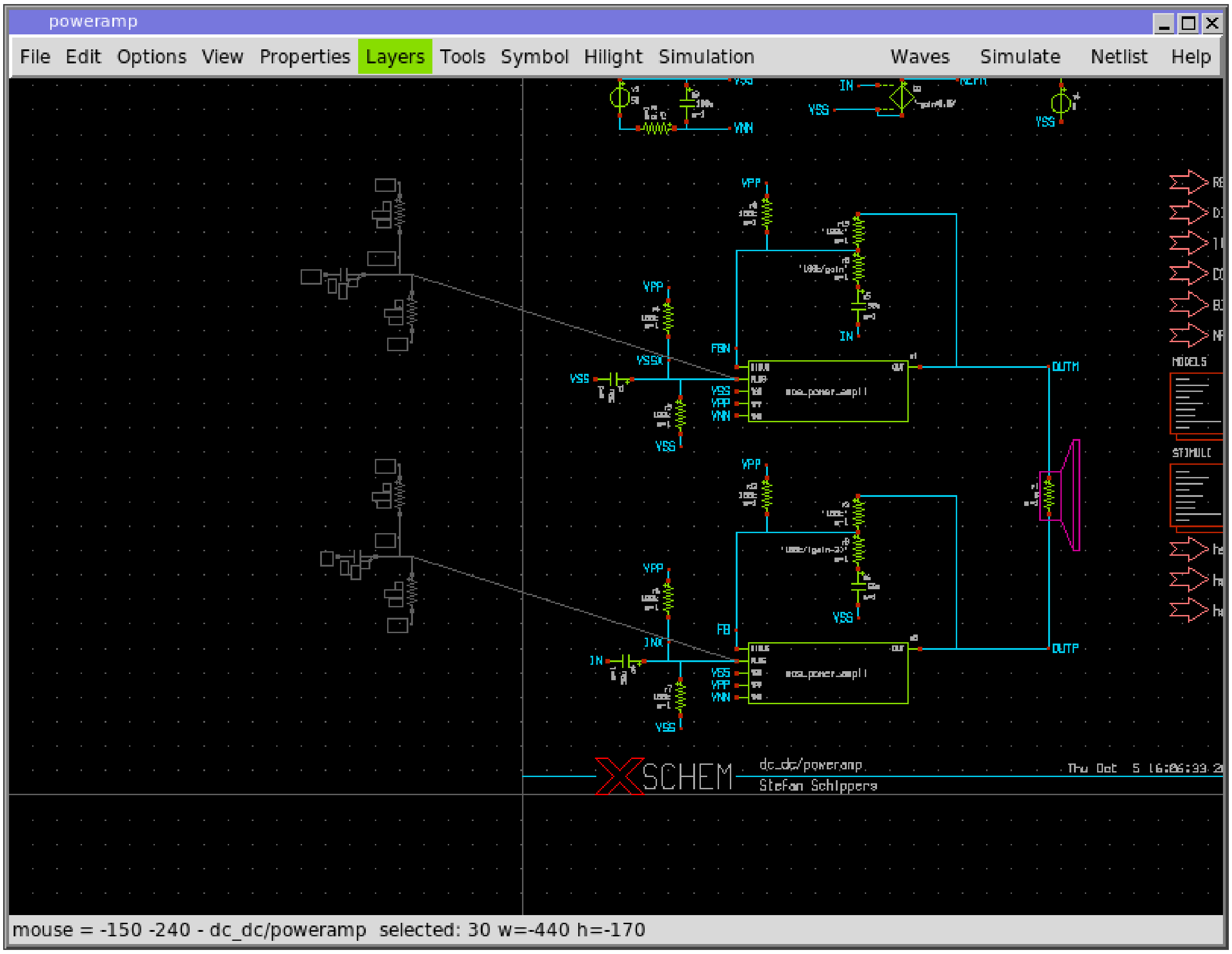
CONSTRAINED MOVE

while creating wires, lines, and moving, stretching, copying objects, pressing the **`'h'`** or **`'v'`** keys will constrain the movement to a horizontal or vertical direction, respectively.


poweramp
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
VPP
VSS
VNN
IN
FBN
FB
OUTM
OUTP
MODELS
STIMULI
mos_power_ampli
XSCHEM
dc_dc/poweramp
Stefan Schippers

mouse = -60 -560 - dc_dc/poweramp selected: 30 w=-350 h=-490

Constrained horizontal move: regardless of the mouse pointer Y position movement occurs on the X direction only.

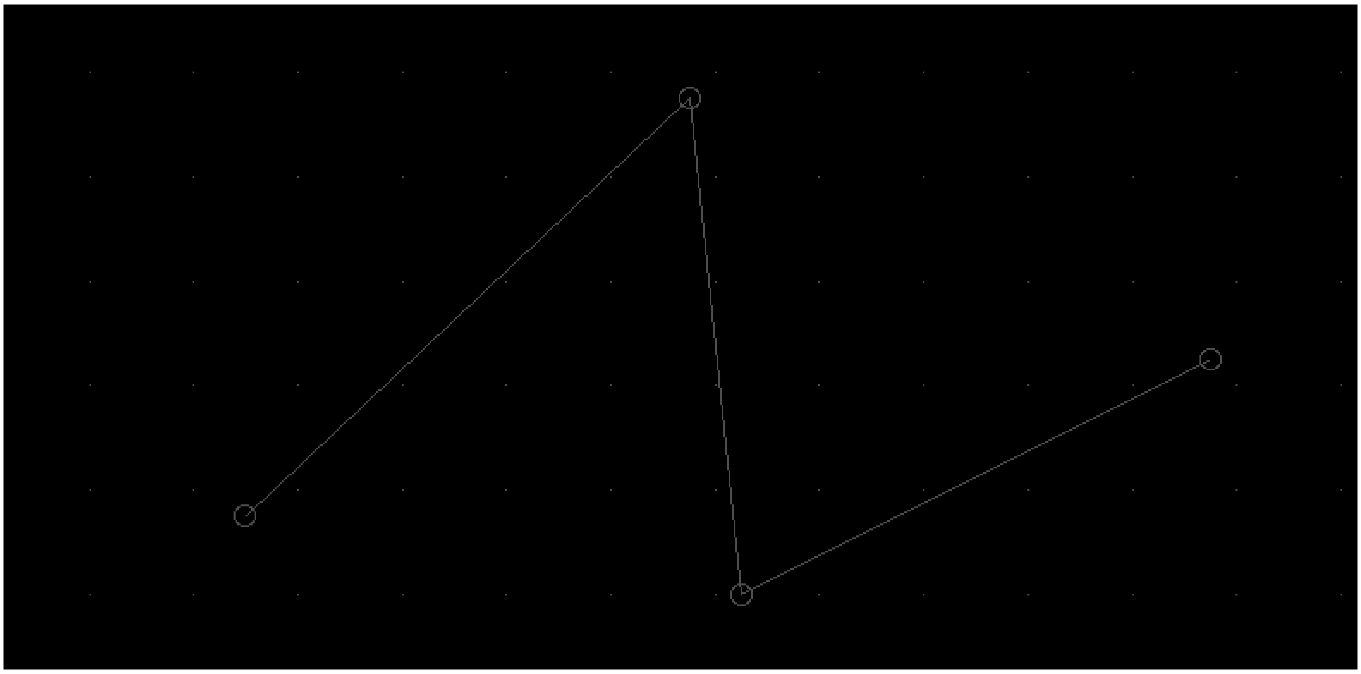

poweramp
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
VPP
VSS
VNN
IN
FBN
FB
OUTM
OUTP
MODELS
STIMULI
mos_power_ampli
XSCHEM
dc_dc/poweramp
Stefan Schippers

mouse = -150 -240 - dc_dc/poweramp selected: 30 w=-440 h=-170

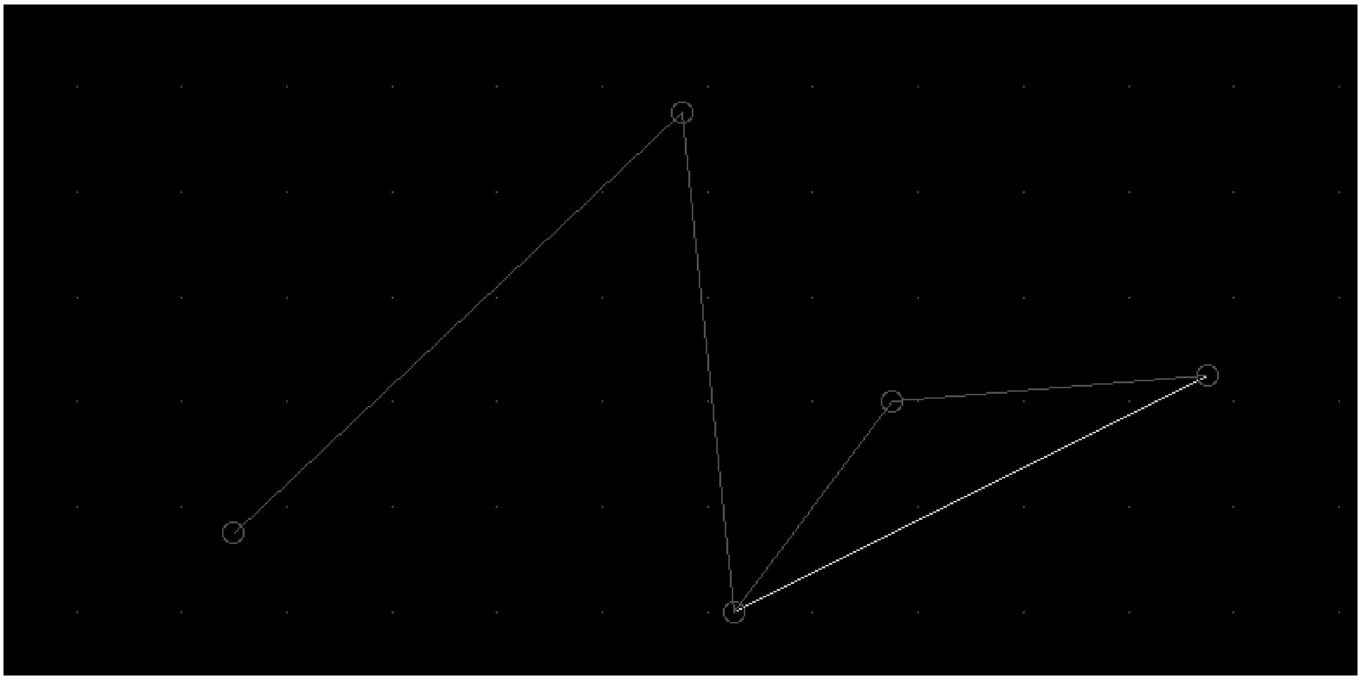
Unconstrained move: objects follow the mouse pointer in X and Y direction.

POLYGON EDITING COMMANDS

There are some specific editing modes for polygons. A polygon is created by pressing the **p** key. After dragging the first segment a mouse button clock will create the second point and so on. A double click ends the placement.
If a polygon is selected the control points are shown with circles. These circles can be dragged with the mouse directly.

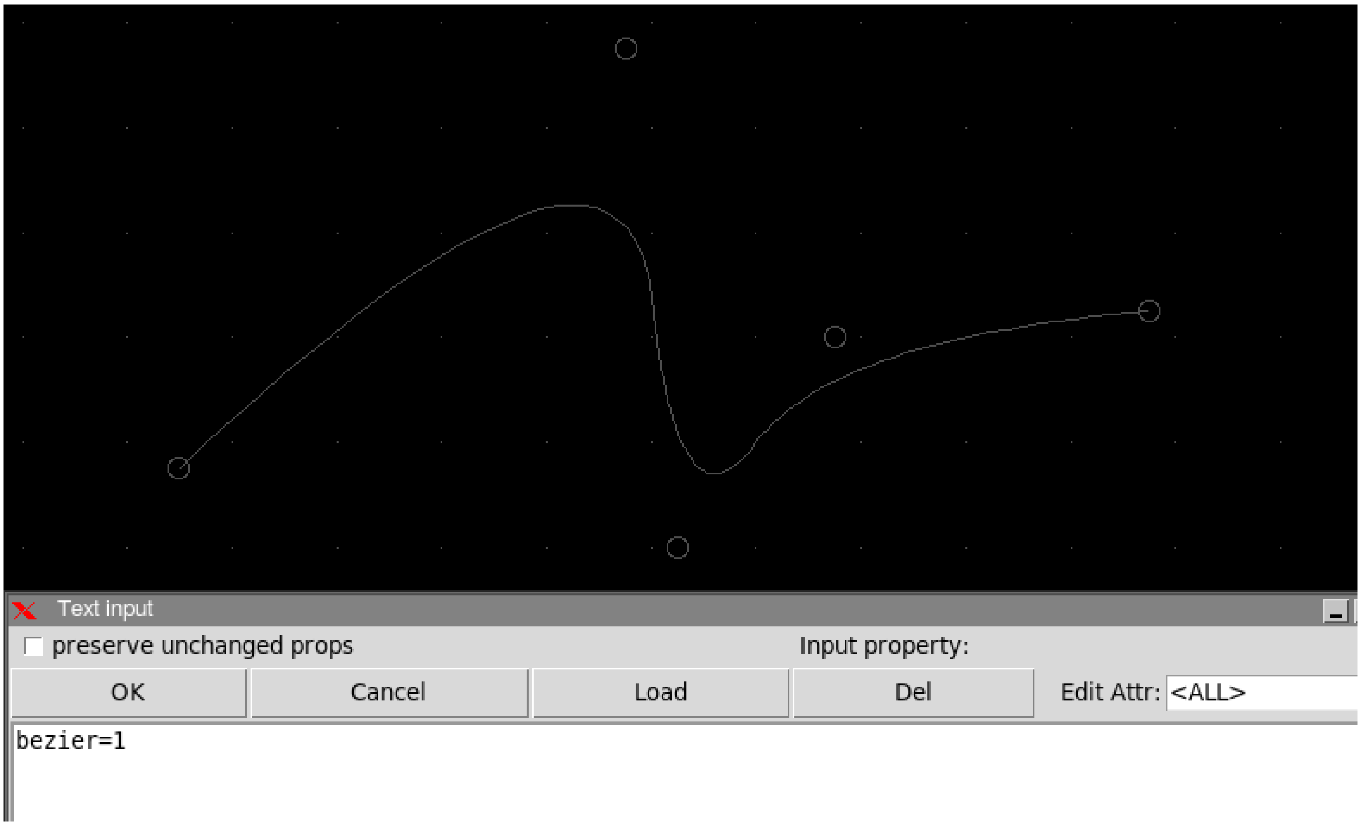


Clicking one control point with the **`Shift`** key will add a new point in the polygon shape:



Clicking one control point with the **`Ctrl`** key will delete a point in the polygon shape.

Adding attribute **`bezier=true`** or **`bezier=1`** will transform the polygon into a bezier curve with the polygon points acting as control points.


Text input
preserve unchanged props
Input property:
OK
Cancel
Load
Del
Edit Attr: <ALL>

```
bezier=1
```

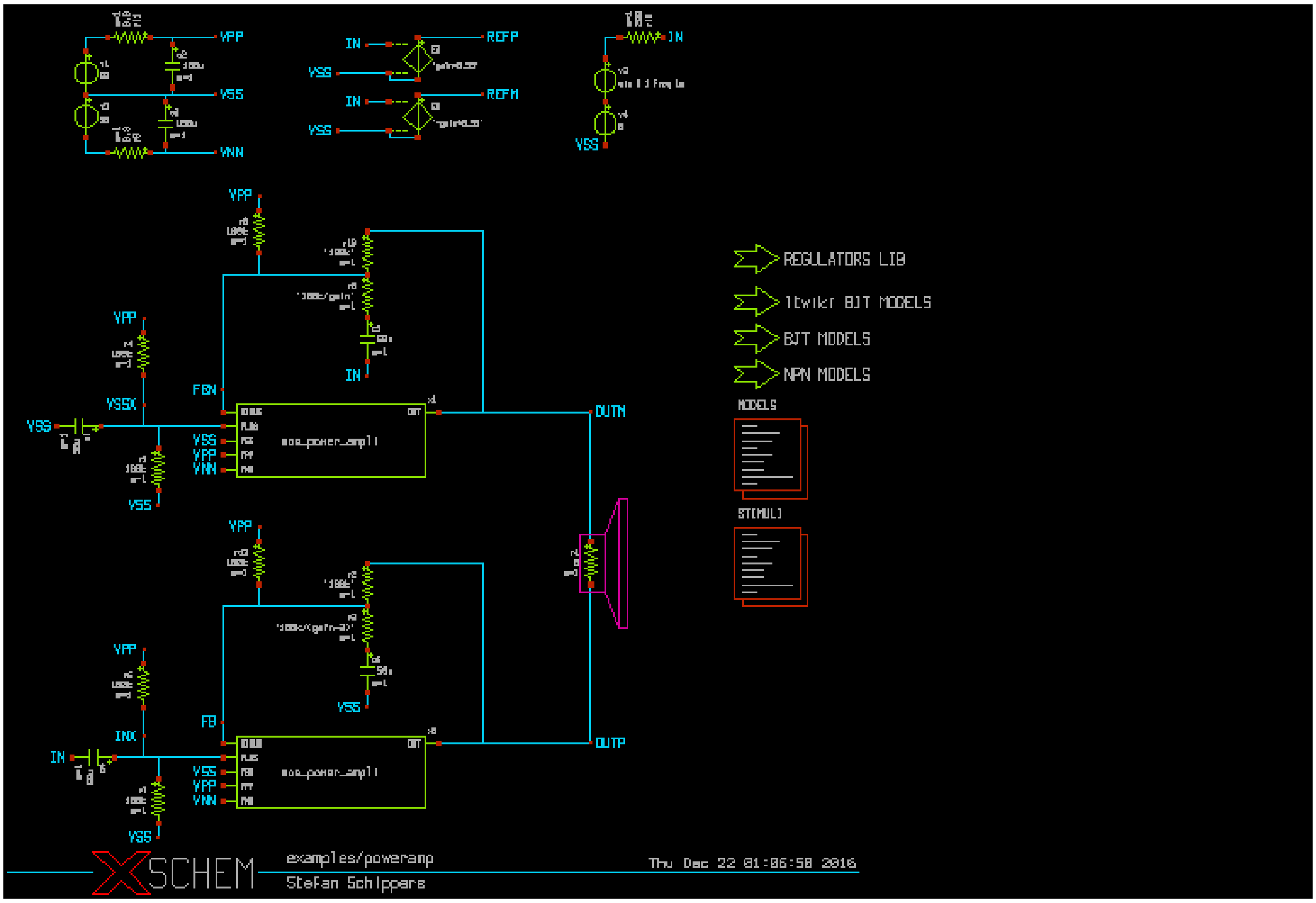
NETLISTING

XSCHEM has 3 predefined netlisting modes, **`Spice`**, **`Verilog`** and **`VHDL`**. Netlisting mode can be set in the **`Options`** menu (**`Vhdl`**, **`Verilog`** **`Spice`** radio buttons) or with the **`<Shift>V`** key. Once a netlist mode is set, hitting the **`Netlist`** button on the top-right of the menu bar or the **`n`** key will produce the netlist file in the defined simulation directory.
The simulation directory is one important path that is specified in the **`xschemrc`** file with the tcl variable **`netlist_dir`** (default if unset is **`~/.xschem/simulations`**) if **`netlist_dir`** is set to empty value (**`set netlist_dir {}`**) xschem will prompt user the first time the netlist is created.
The path where netlists are produced can be changed with the **`Simulation->Set netlist dir`** menu entry or simply by changing the **`netlist_dir`** variable, either in the xschemrc file or interactively by giving tcl commands.
If you use xschem interactively by giving tcl commands you may do something like:
`set netlist_dir [xschem get current_dirname]/spice; xschem netlist`
to have the netlist saved into a **`spice/`** folder into the directory containing the current schematic (this directory is not necessarily the current directory, like 'pwd'). There is also a **`local_netlist_dir`** variable. If this variable is set to 1 (default setting if unspecified is 0) instead of using **`~/.xschem/simulations`** the netlist will be saved in **`(directory of current schematic)/simulation`** The netlist filename is **`cellname.ext`** where **`cellname`** is the name of the top-level schematic from which the netlist has been generated, and **`ext`** is the file extension:

- **`spice`** for spice netlist.
- **`vhdl`** for vhdl netlist.
- **`v`** for verilog netlist.

EXAMPLE

Consider the following top level schematic, part of the XSCHEM distribution (**`examples/poweramp.sch`**).



This schematic is made of some **`leaf`** components and some **`subcircuit`** components:

- **`leaf`**: these components are 'known' to the simulator, netlist of these blocks is done by specifying a 'format' attribute in the symbol property string. Examples of leaf components in the schematic above are voltage sources, resistors, capacitors, dependent sources. The following are examples of leaf component instantiations in a SPICE netlist:

  ```
  c3 VSS VNN 100u m=1
  r11 VPP net1 0.3 m=1
  r9 VNN net2 0.3 m=1
  r19 OUTM FBN '100k' m=1
  ```

  The format of resistor (and capacitor) SPICE netlist is defined in the format attribute of the symbol global property:

  ```
  format="@name @pinlist @value m=@m"
  ```

- **`subcircuit`**: these components are not base blocks known to the simulator, but are representation of a more complex block. These components have in addition to the symbol a schematic representation. In the picture example the **`mos_power_ampli`** is a subcircuit block. These type of components also have a 'format' property that defines a subcircuit call. A subcircuit call specifies the connections of nets to the symbol pins and the symbol name. The following two subcircuit calls are present in the SPICE netlist:

```
        x1 OUTM VSSX FBN VPP VNN VSS mos_power_ampli
        x0 OUTP INX FB VPP VNN VSS mos_power_ampli
```

The format of subcircuit type components is also defined in the symbol **`format`** attribute:

```
        format="@name @pinlist @symname"
```

For subcircuits, after completing the netlist of the top level the XSCHEM' netlister will recursively generate all the netlists of subcircuit components until leaf schematics are reached that do not instantiate further subcircuits.

```
...
... (end of top level netlist)
...
* expanding symbol: examples/mos_power_ampli # of pins=6

.subckt mos_power_ampli OUT PLUS MINUS VPP VNN VSS
*.ipin PLUS
*.ipin MINUS
*.ipin VPP
...
...
```

Other netlist formats

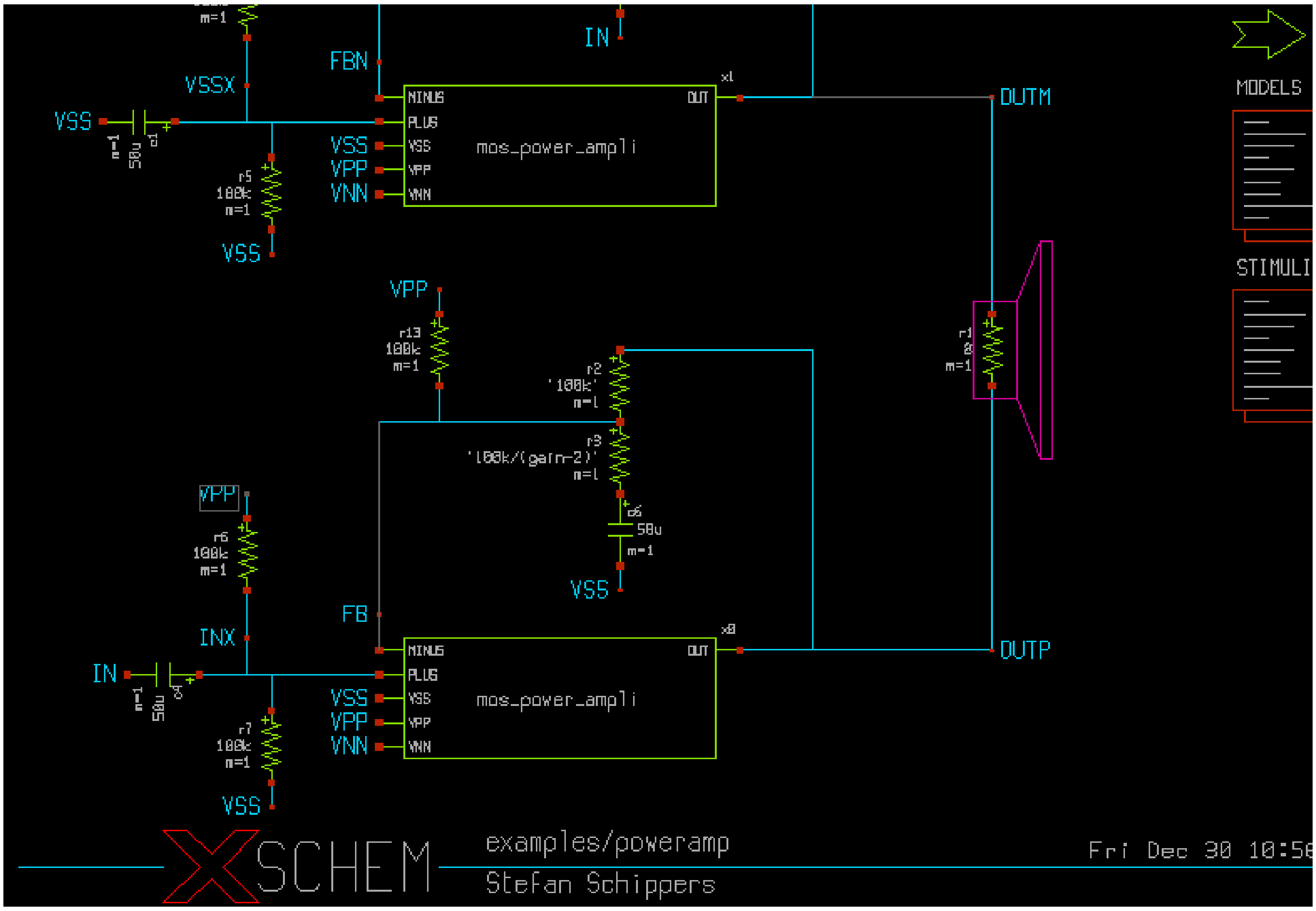
All the concepts explained for SPICE netlist apply for Verilog and VHDL formats. Its up to the designer to ensure that the objects in the schematic are 'known' to the target simulator. For example a resistor is normally not used in VHDL or Verilog designs, so unless an appropriate 'format' attribute is defined (for example a **`rtran`** device may be good for a verilog resistor with some limitations). The format attribute for Verilog is called **`verilog_format`** and the attribute for VHDL is **`vhdl_format`**
The following example shows two attributes in a NMOS symbol that define the format for SPICE and for Verilog and some valid default (**`template`**) values:

```
type=nmos
format="@name @pinlist @model w=@w l=@l m=@m"
verilog_format="@verilog_gate #(@del ) @name ( @@d , @@s , @@g );"
template="name=x1 verilog_gate=nmos del=50,50,50 model=NCH w=0.68 l=0.07 m=1"
generic_type="model=string"
```

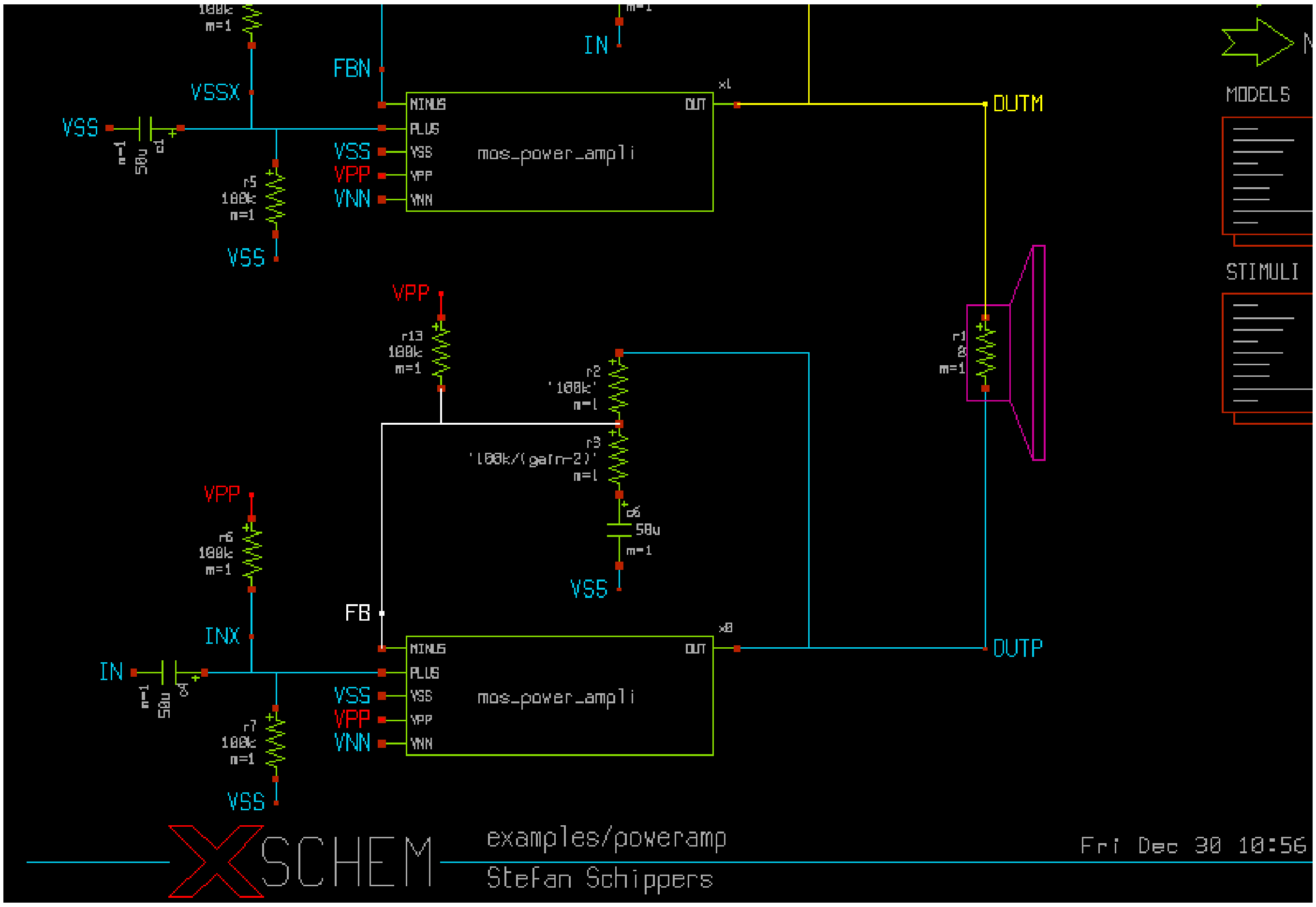
NET PROBES

XSCHEM has the ability to highlight a net and propagate the highlight color to all nets or instance pins attached to the net. It has the ability to follow this net through the hierarchy. This is very useful in large designs as it makes it easy to see where a net is driven and were the net goes (fan-out). Highlighting a net is straightforward, click a net and press the **`'k'`** key. If more nets are selected all nets will be colored with different colors. **`<Shift>K`** clears all highlight nets, **`<Ctrl>k`** clears selected nets.

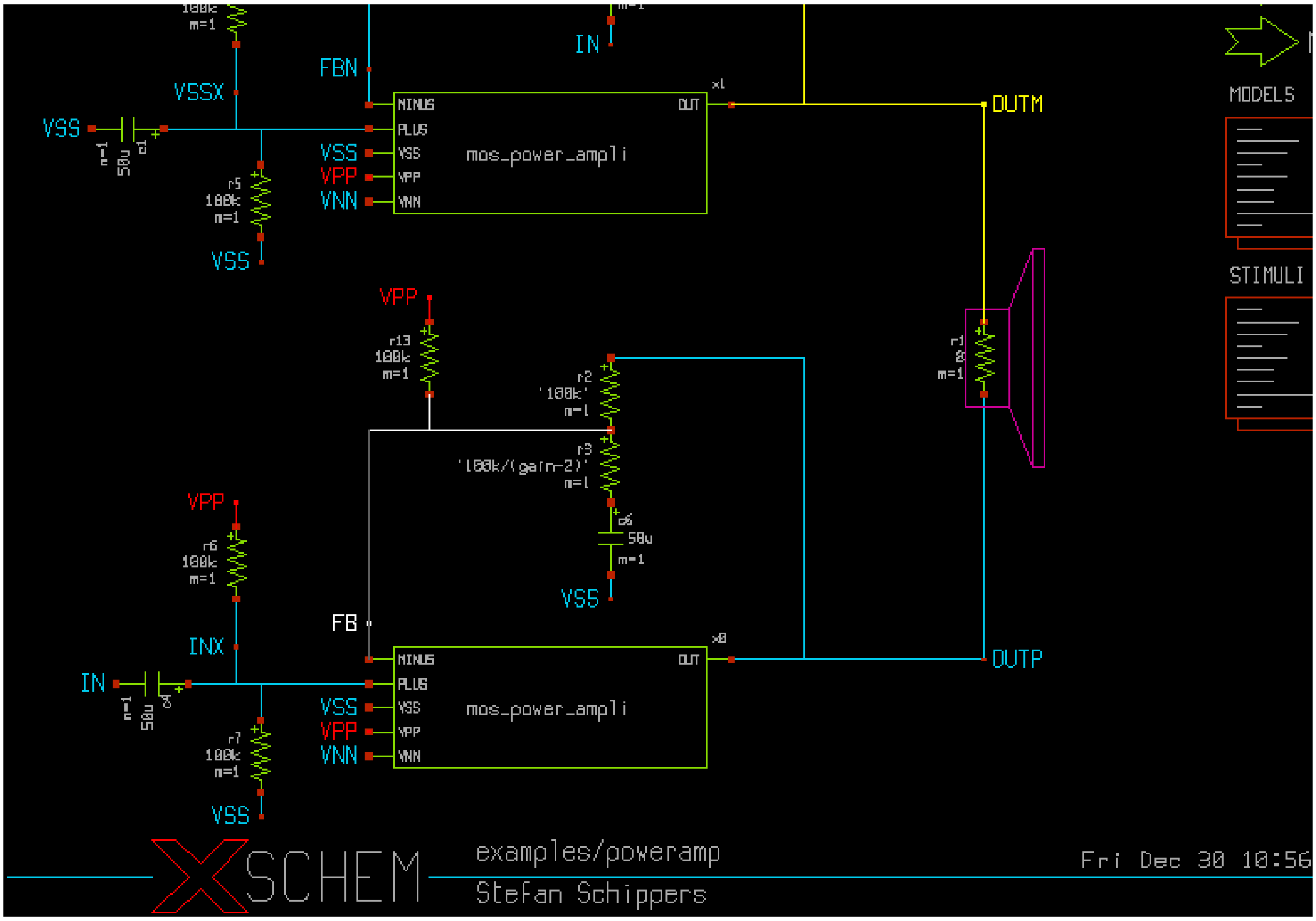
Select some nets...



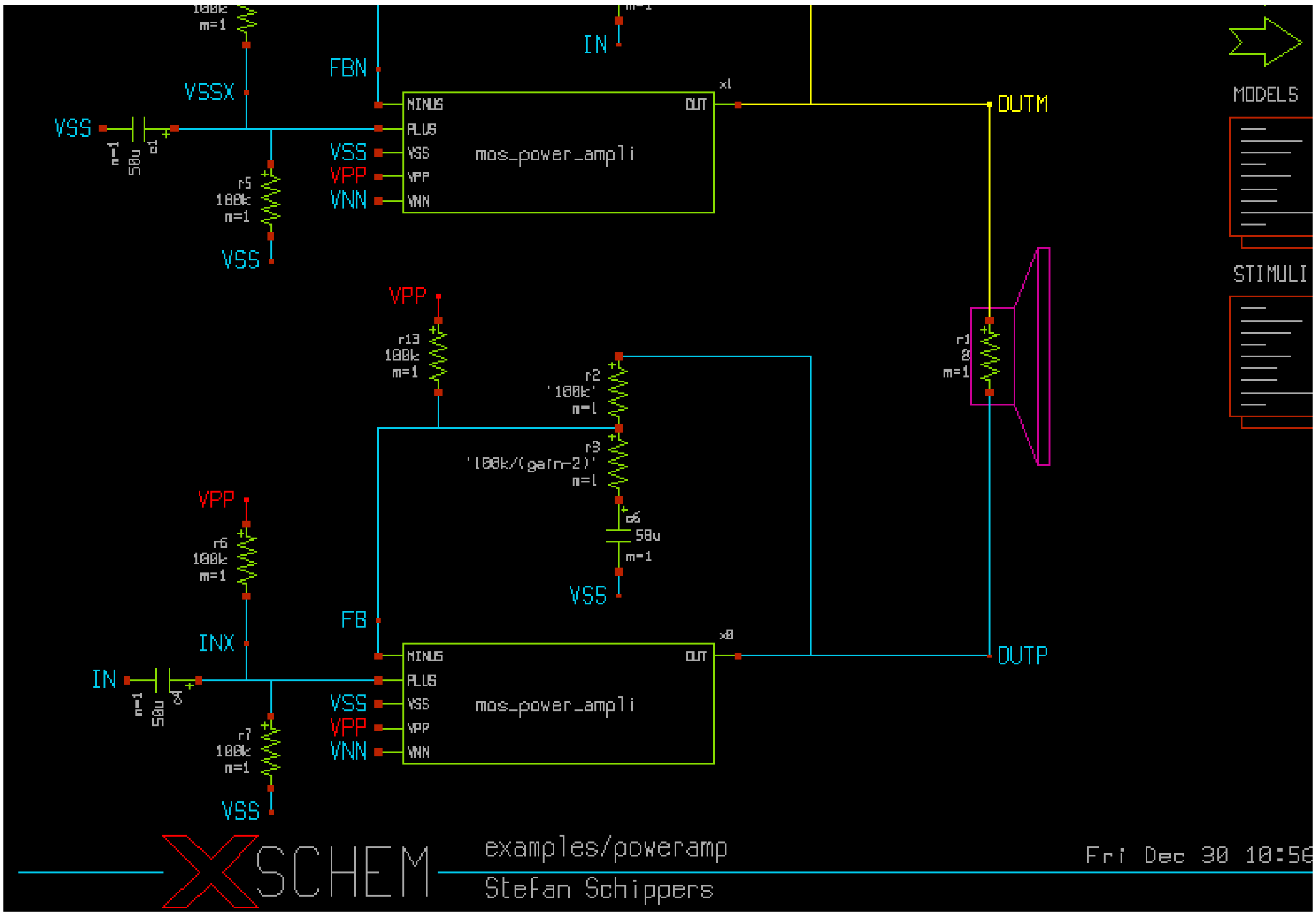
...press the **`'k'`** key...



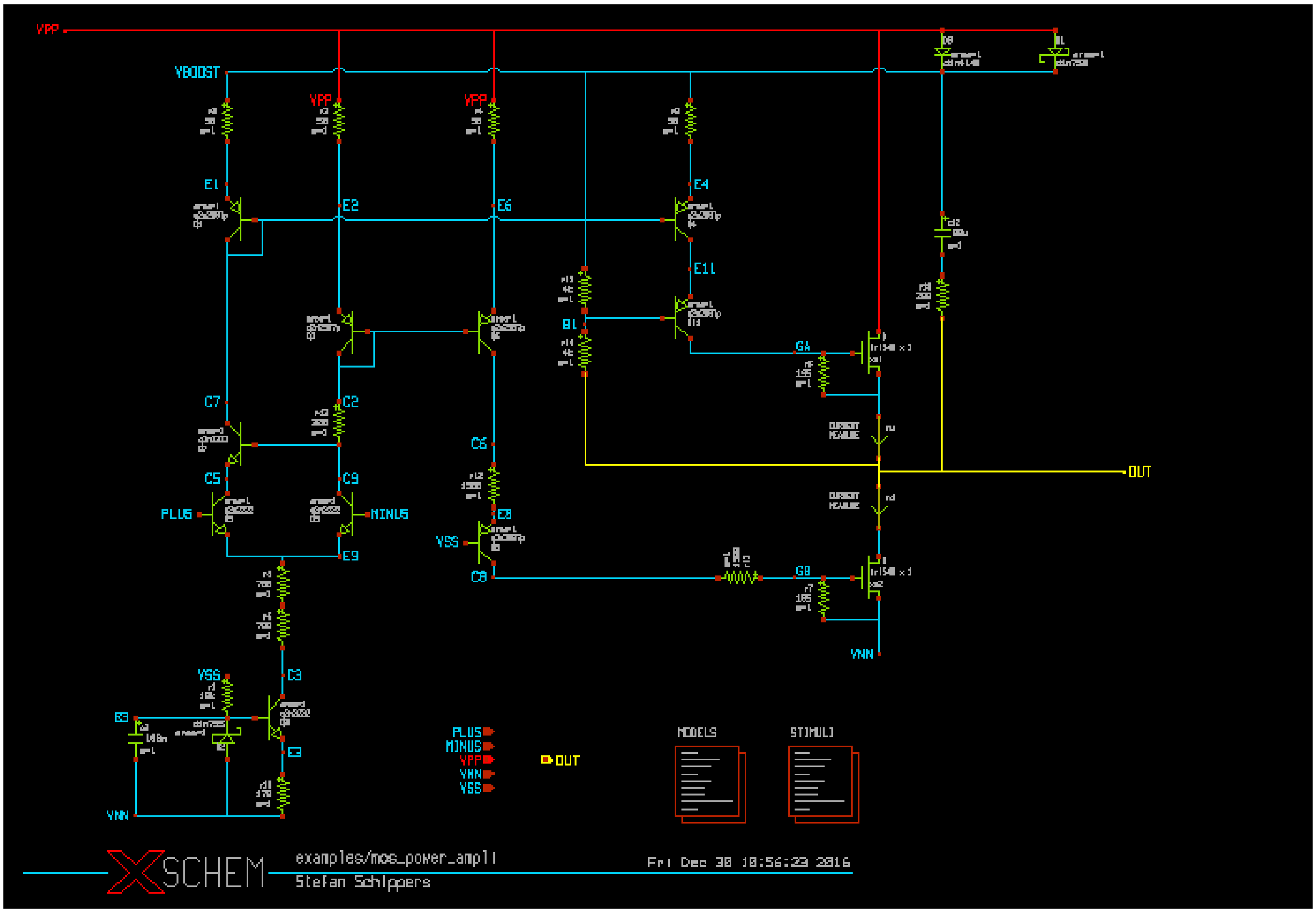
...all nets are highlighted, select the white net...



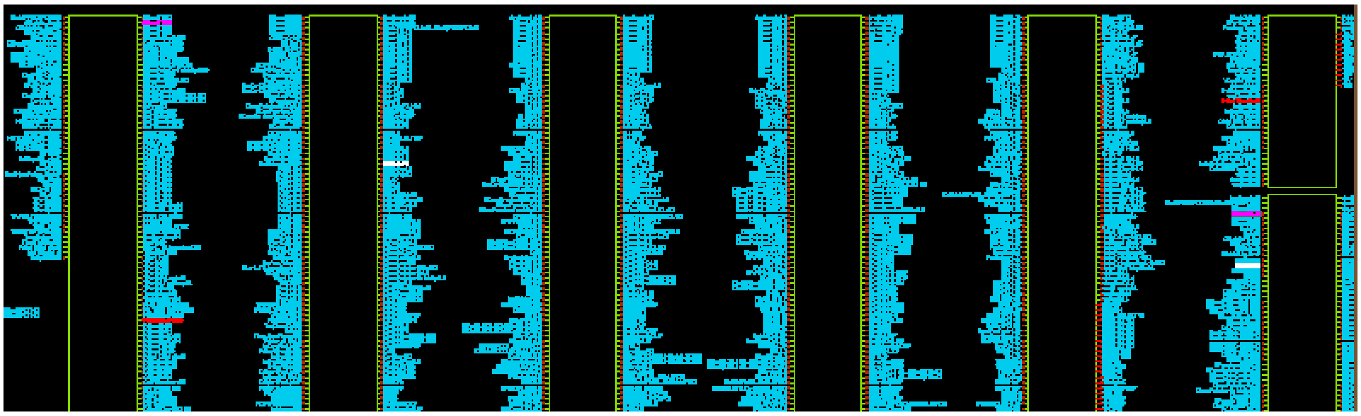
..press the **<Ctrl>k** key and white net is un-highlighted...



if you descend into component instance **x1** (mos_power_ampli) (**'e'** key) you will see the highlight nets propagated into the child component.



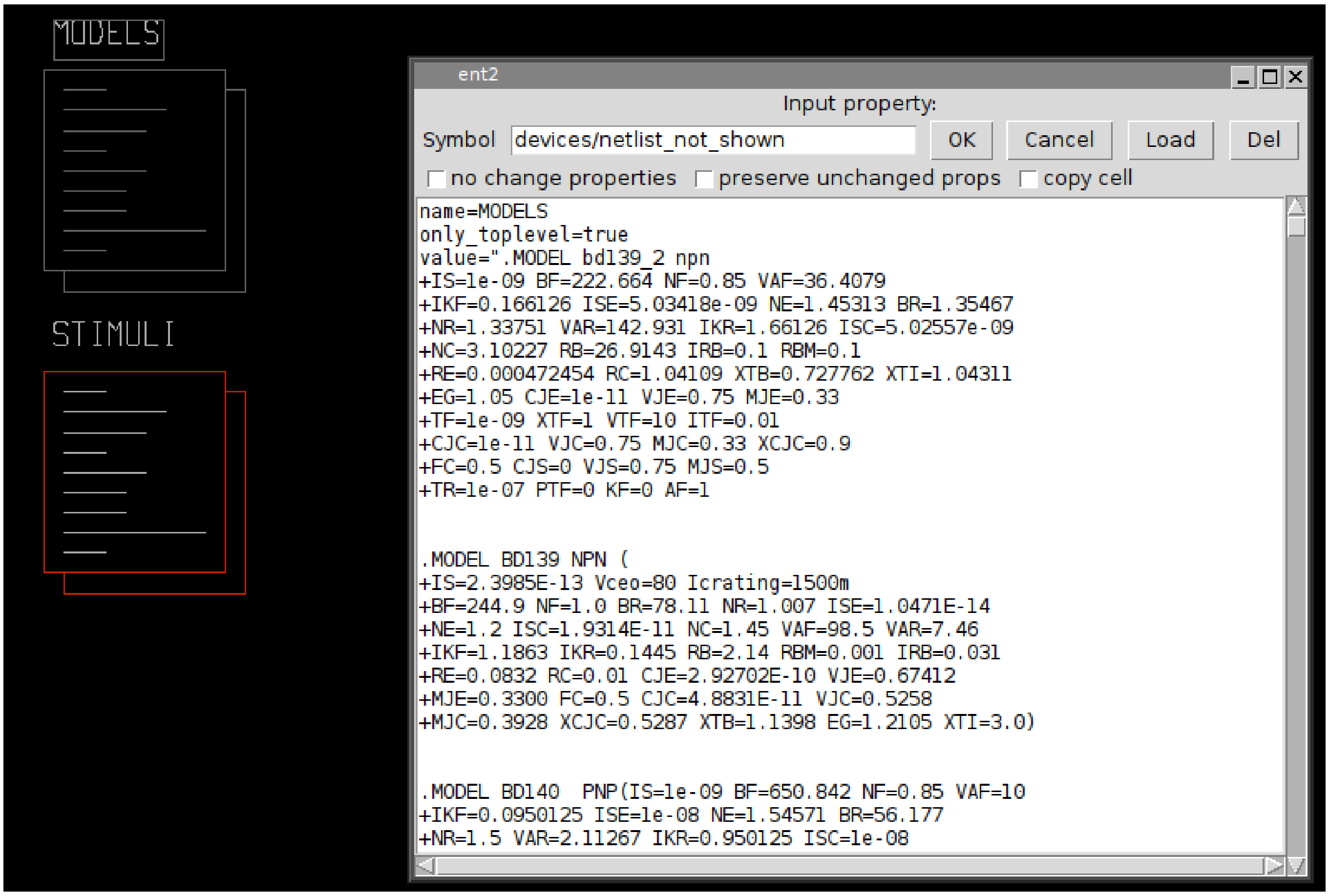
A very useful function is the 'View only probes' mode, (`'5'` key) that hides everything but the highlight probes. This is useful in very big VLSI designs to quickly locate start and end point of nets. Pressing again the `'5'` key restores the normal view.



PREV UP NEXT

SIMULATION

One of the design goals of XSCHEM is the ability to launch a simulation without additional manual file editing. For this purpose XSCHEM stores in a schematic not only the circuit but also the simulator settings and the additional files that are needed. For example there is a **`devices/netlist.sym`** and **`devices/netlist_not_shown.sym`** symbol that can be placed in a schematic acting as a container of text files for all the needed SPICE models and any additional information to make the schematic ready for simulation.



Symbol devices/netlist_not_shown

```
name=MODELS
only_toplevel=true
value=".MODEL bd139_2 npn
+IS=1e-09 BF=222.664 NF=0.85 VAF=36.4079
+IKF=0.166126 ISE=5.03418e-09 NE=1.45313 BR=1.35467
+NR=1.33751 VAR=142.931 IKR=1.66126 ISC=5.02557e-09
+NC=3.10227 RB=26.9143 IRB=0.1 RBM=0.1
+RE=0.000472454 RC=1.04109 XTB=0.727762 XTI=1.04311
+EG=1.05 CJE=1e-11 VJE=0.75 MJE=0.33
+TF=1e-09 XTF=1 VTF=10 ITF=0.01
+CJC=1e-11 VJC=0.75 MJC=0.33 XCJC=0.9
+FC=0.5 CJS=0 VJS=0.75 MJS=0.5
+TR=1e-07 PTF=0 KF=0 AF=1


.MODEL BD139 NPN (
+IS=2.3985E-13 Vceo=80 Icrating=1500m
+BF=244.9 NF=1.0 BR=78.11 NR=1.007 ISE=1.0471E-14
+NE=1.2 ISC=1.9314E-11 NC=1.45 VAF=98.5 VAR=7.46
+IKF=1.1863 IKR=0.1445 RB=2.14 RBM=0.001 IRB=0.031
+RE=0.0832 RC=0.01 CJE=2.92702E-10 VJE=0.67412
+MJE=0.3300 FC=0.5 CJC=4.8831E-11 VJC=0.5258
+MJC=0.3928 XCJC=0.5287 XTB=1.1398 EG=1.2105 XTI=3.0)


.MODEL BD140  PNP(IS=1e-09 BF=650.842 NF=0.85 VAF=10
+IKF=0.0950125 ISE=1e-08 NE=1.54571 BR=56.177
+NR=1.5 VAR=2.11267 IKR=0.950125 ISC=1e-08
```

The **`devices/netlist_not_shown`** symbol shown in the picture (with name MODELS) for example contains all the spice models of the components used in the schematic, this makes the schematic self contained, no additional files are needed to run a simulation. After generating the netlist (for example poweramp.spice) the resulting SPICE netlist can be sent directly for simulation (for example **`hspice -i poweramp.spice`** for the Hspice(TM) simulator).

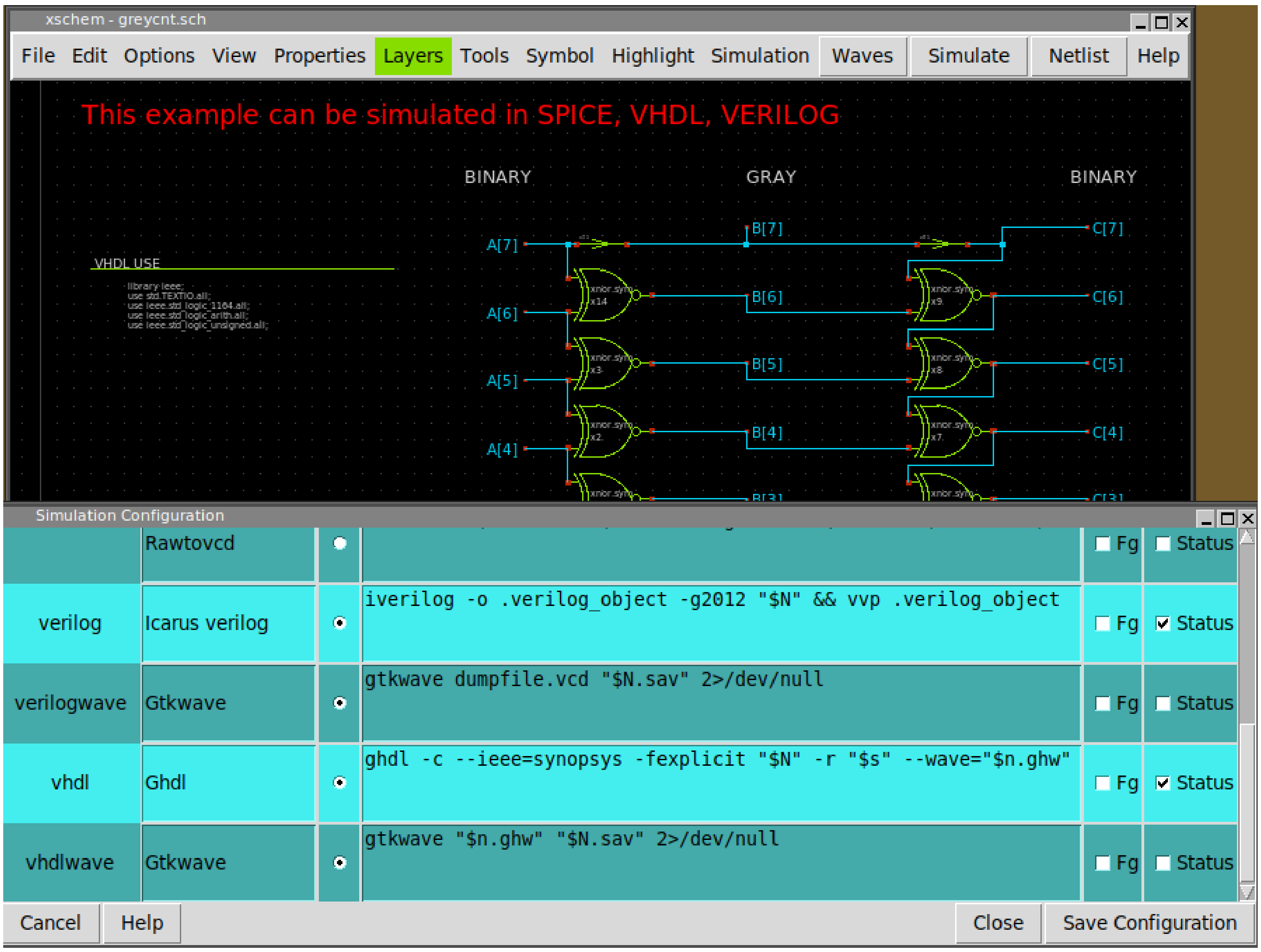
HELPFUL XTERM CONFIGURATION

By default, xschem runs all simulators in an xterm window. Here is some configuration to make xterm more usable:

```
# enable copy/paste with Ctrl-Shift-C and Ctrl-Shift-V.
# set font size to 18.
echo '
XTerm*VT100.Translations: #override \
    Ctrl Shift <Key>C: copy-selection(CLIPBOARD) \n\
    Ctrl Shift <Key>V: insert-selection(CLIPBOARD)
xterm*font:      *-fixed-*-*-*-18-*
' >> ~/.Xresources

# reload the configuration
xrdb -merge ~/.Xresources
```

VERILOG SIMULATION

This is a tutorial showing how to run a simulation with XSCHEM. The first important thing to note is that XSCHEM is just a schematic editor, so we need to setup valid bindings to simulators. For this tutorial we plan to do a **`Verilog`** simulation since there is a very good open source simulator available, called Icarus Verilog. There is also a good waveform viewer called gtkwave that is able to show simulator results. Install these two valuable tools and setup simulator invocation by using the Simulator configurator (**`Simulation->Configure Simulators and tools`**).

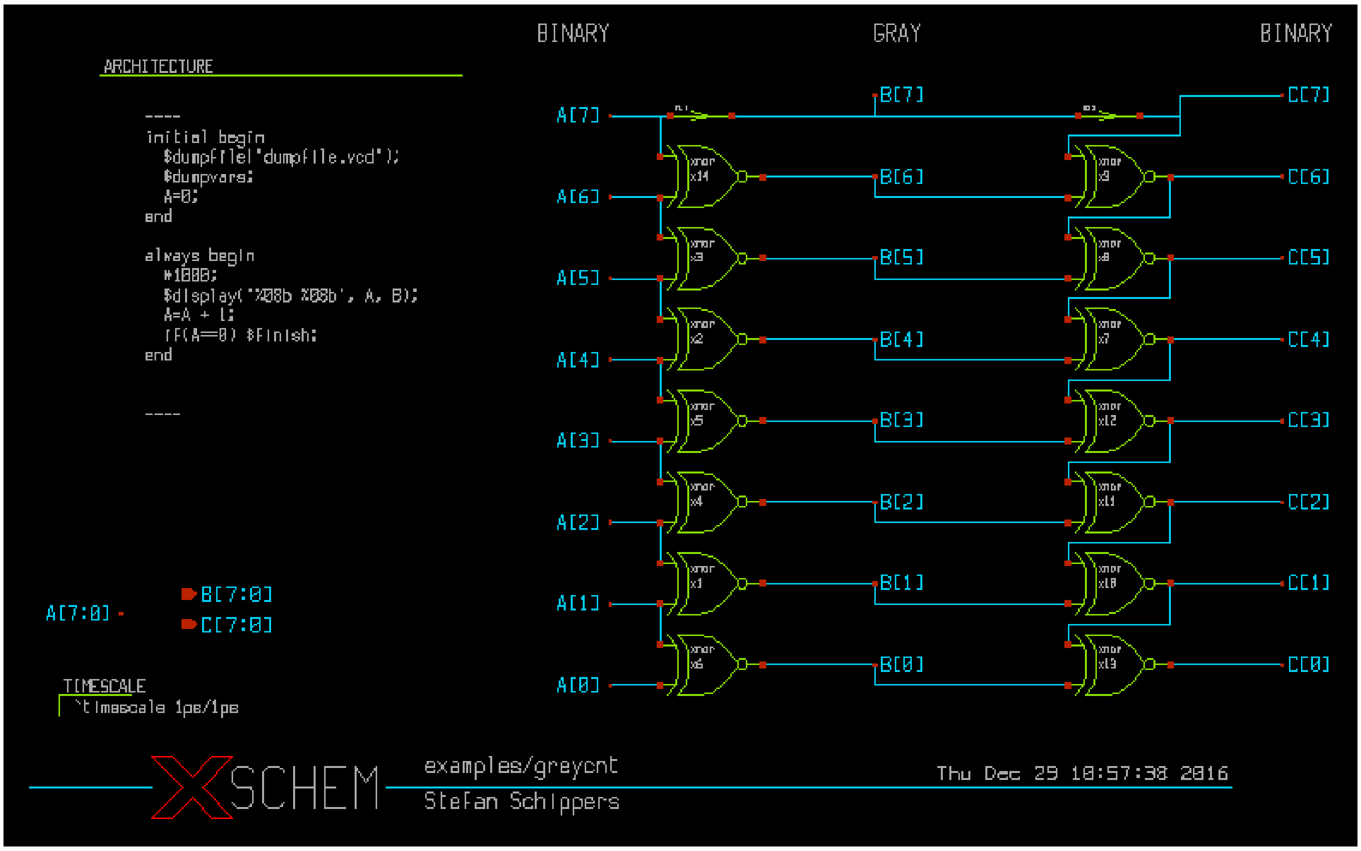


	Rawtovcd	○		☐ Fg	☐ Status
verilog	Icarus verilog	●	`iverilog -o .verilog_object -g2012 "$N" && vvp .verilog_object`	☐ Fg	✓ Status
verilogwave	Gtkwave	●	`gtkwave dumpfile.vcd "$N.sav" 2>/dev/null`	☐ Fg	☐ Status
vhdl	Ghdl	●	`ghdl -c --ieee=synopsys -fexplicit "$N" -r "$s" --wave="$n.ghw"`	☐ Fg	✓ Status
vhdlwave	Gtkwave	●	`gtkwave "$n.ghw" "$N.sav" 2>/dev/null`	☐ Fg	☐ Status

The text entry on the verilog line is the command to invoke icarus verilog simulation. **`$N`** will be expanded to the netlist file (**`$netlist_dir/greycnt.v`**), while **`$n`** will be replaced with the circuit name without extension (**`$netlist_dir/greycnt`**). Note also the command to invoke gtkwave on the vcd file generated by the verilog simulation. If **`Save Configuration`** button is pressed the changes are made permanent by saving in a **`~/.xschem/simrc`** file.

In the XSCHEM distribution there is one example design, **`examples/greycnt.sch`**.
Load this design:

```
user:~$ xschem .../share/doc/xschem/examples/greycnt.sch
```

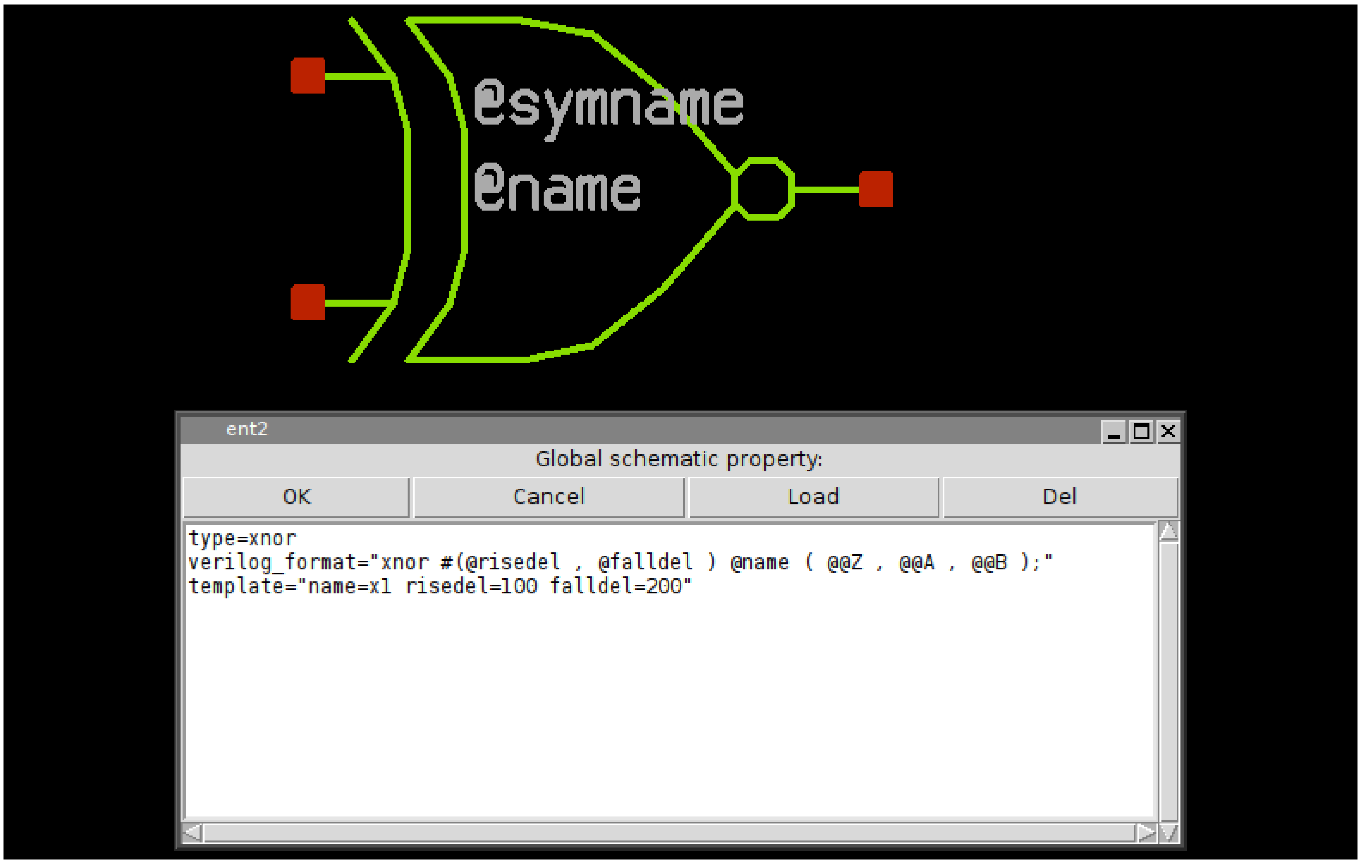


```
----
initial begin
  $dumpfile("dumpfile.vcd");
  $dumpvars;
  A=0;
end

always begin
  #1000;
  $display("%08b %08b", A, B);
  A=A + 1;
  if(A==0) $finish;
end


----
```

```
`timescale 1ps/1ps
```

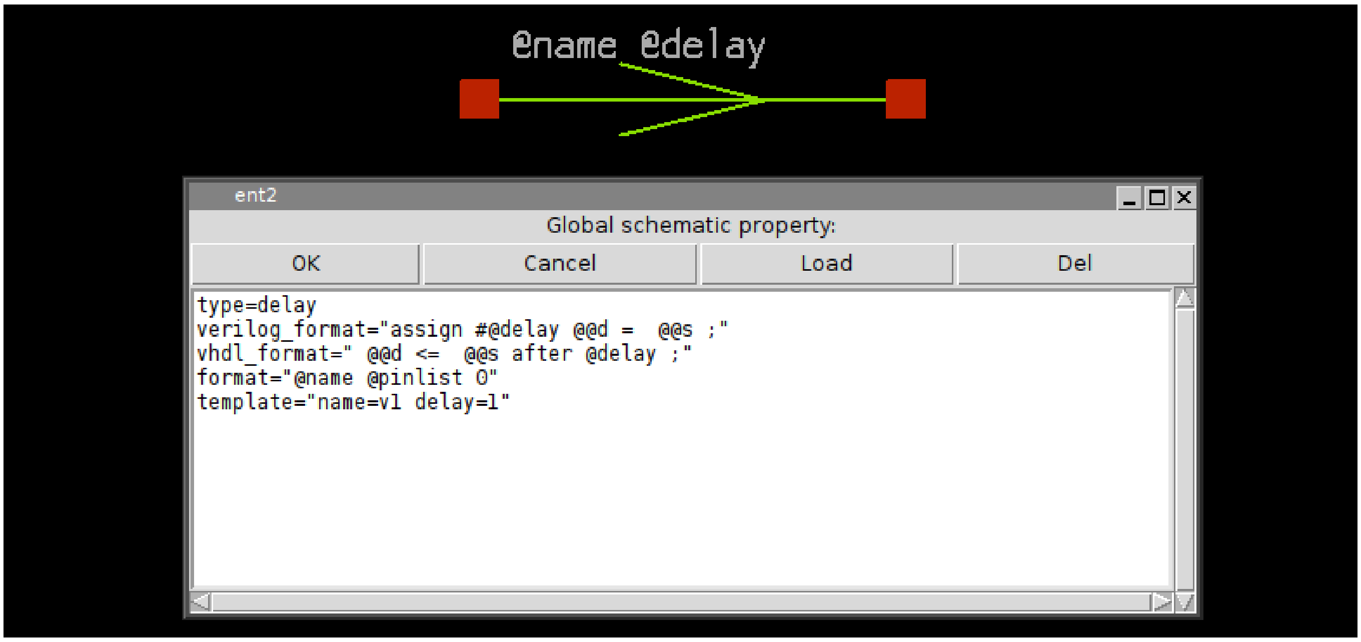
This testbench has a 8 bit input vector A[7:0] and two output vectors, B[7:0] and C[7:0]. B[7:0] is a grey coded vector, this mean that if A[7:0] is incremented as a binary number B[7:0] will increment by changing only one bit at a time. The C[7:0] vector is the reverse transformation from grey-code to binary, so at the end if simulation goes well C[7:0] == A[7:0]. In this schematic there are some components, the first one is the **`xnor`** gate, the second one is the **`assign`** element. The 'xnor' performs the logical 'Not-Xor' of its inputs, while 'assign' just propagates the input unchanged to the output, optionally with some delay. This is useful if we want to change the name of a net (putting two labels with different names on the same net is not allowed, since this is normally an error, leading to a short circuit).

An Ex-Nor gate can be represented as a verilog primitive, so for the xnor gate we just need to setup a **`verilog_format`** attribute in the global property string of the **`xnor.sym`** gate:



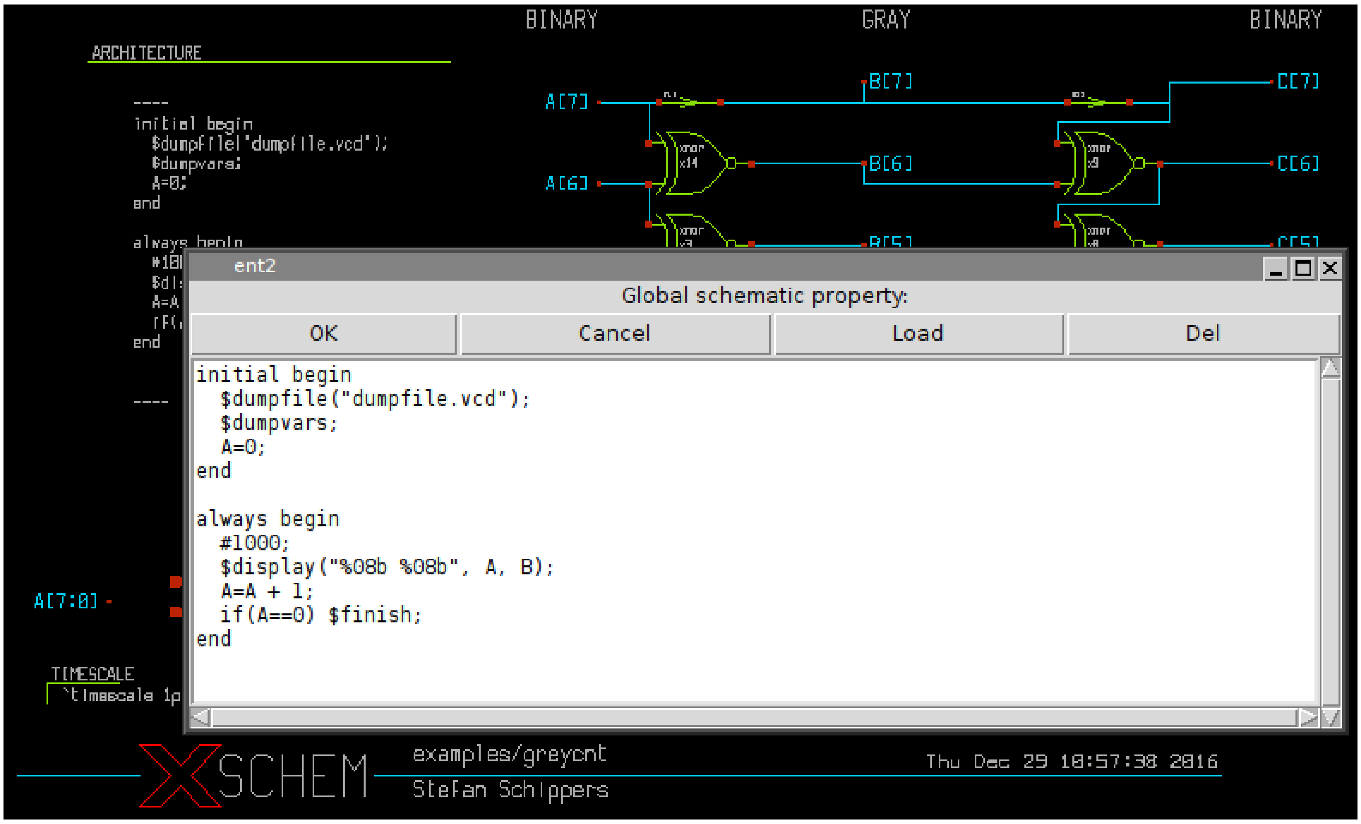
```
type=xnor
verilog_format="xnor #(@risedel , @falldel ) @name ( @@Z , @@A , @@B );"
template="name=x1 risedel=100 falldel=200"
```

the 'assign' symbol is much simpler, in this property string you see the definition for SPICE (**`format`** attribute), Verilog (**`verilog_format`**) and VHDL (**`vhdl_format`**). This shows how a single symbol can be used for different netlist formats.



```
type=delay
verilog_format="assign #@delay @@d =  @@s ;"
vhdl_format=" @@d <=  @@s after @delay ;"
format="@name @pinlist 0"
template="name=v1 delay=1"
```

While showing the top-level testbench **`greycnt`** set XSCHEM in Verilog mode (menu **`Options->Verilog`** radio button, or **`<Shift>V`** key) and press the edit property **`'q'`** key, you will see some verilog code:



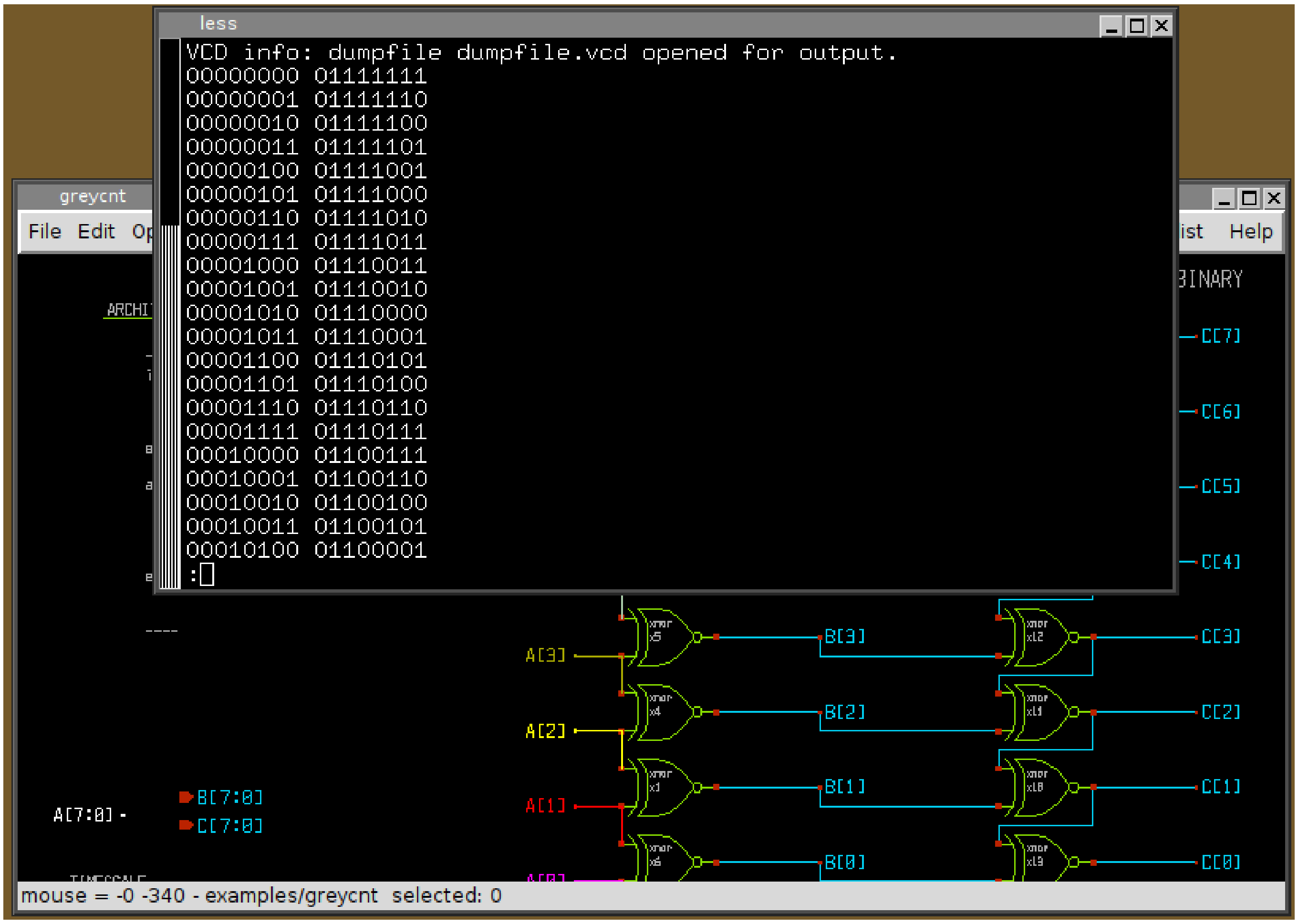
```
initial begin
  $dumpfile("dumpfile.vcd");
  $dumpvars;
  A=0;
end

always begin
  #1000;
  $display("%08b %08b", A, B);
  A=A + 1;
  if(A==0) $finish;
end
```

This is the testbench behavioral code that generates stimuli for the simulation and gives instructions on where to save simulation results. If you generate the verilog netlist with the **Netlist** button on the right side of the menu bar (or **n** key) a **greycnt.v** file will be generated in the simulation directory (**${HOME}/xschem_library/simulations** is the default path in the XSCHEM distribution, but can be changed with the **set netlist_dir $env(HOME)/simulations** in **xschemrc** file):

```
`timescale 1ps/1ps
module greycnt (
  output wire [7:0] B,
  output wire [7:0] C
);

reg [7:0] A  ;

xnor #(1 , 1 ) x2 ( B[4] , A[5] , A[4] );
xnor #(1 , 1 ) x3 ( B[5] , A[6] , A[5] );
xnor #(1 , 1 ) x14 ( B[6] , A[7] , A[6] );
assign #1 B[7] = A[7] ;
xnor #(1 , 1 ) x1 ( B[1] , A[2] , A[1] );
xnor #(1 , 1 ) x4 ( B[2] , A[3] , A[2] );
xnor #(1 , 1 ) x5 ( B[3] , A[4] , A[3] );
xnor #(1 , 1 ) x6 ( B[0] , A[1] , A[0] );
xnor #(1 , 1 ) x7 ( C[4] , C[5] , B[4] );
xnor #(1 , 1 ) x8 ( C[5] , C[6] , B[5] );
xnor #(1 , 1 ) x9 ( C[6] , C[7] , B[6] );
assign #1 C[7] = B[7] ;
xnor #(1 , 1 ) x10 ( C[1] , C[2] , B[1] );
xnor #(1 , 1 ) x11 ( C[2] , C[3] , B[2] );
xnor #(1 , 1 ) x12 ( C[3] , C[4] , B[3] );
xnor #(1 , 1 ) x13 ( C[0] , C[1] , B[0] );
initial begin
  $dumpfile("dumpfile.vcd");
  $dumpvars;
```

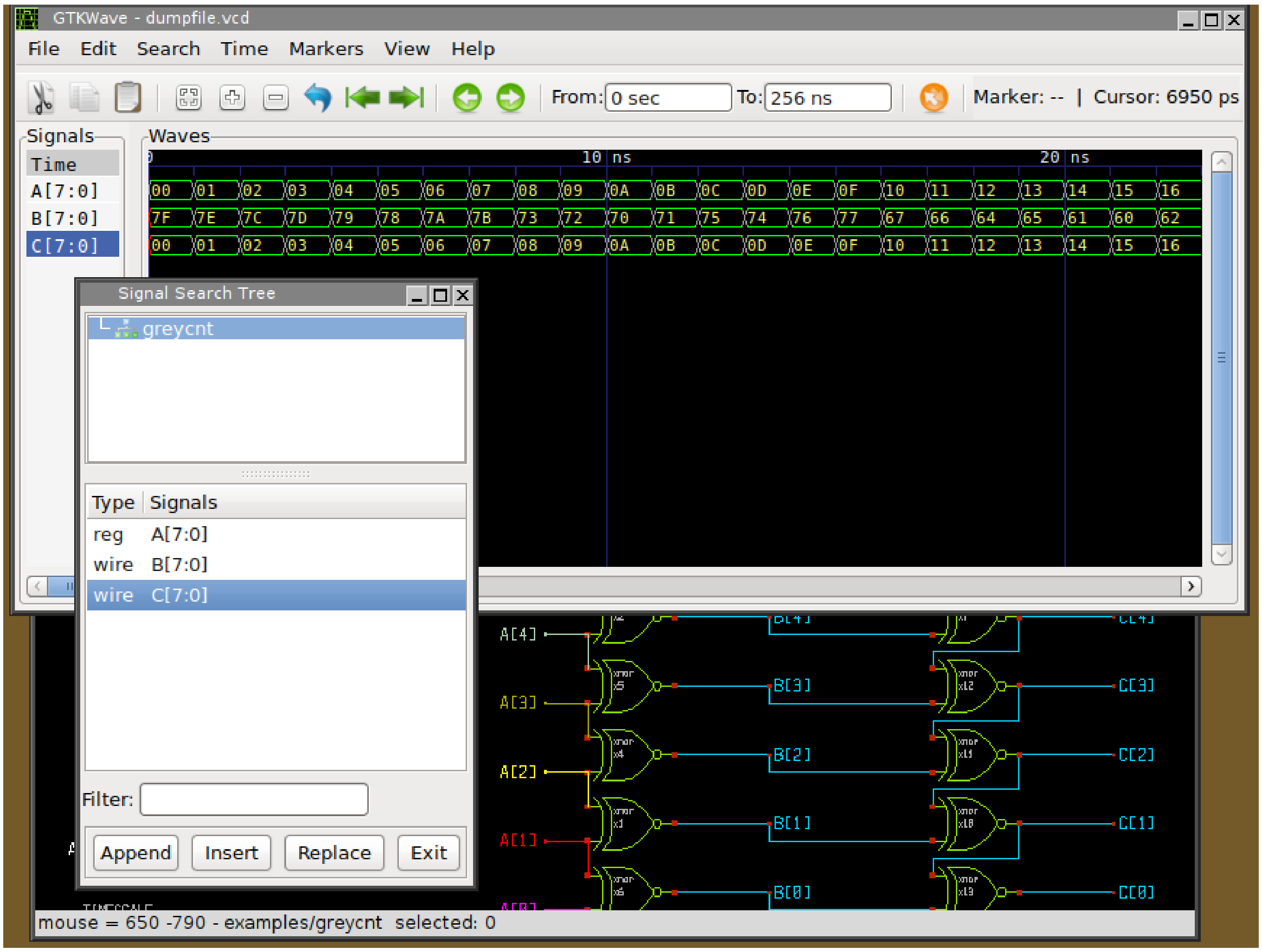
```
    A=0;
  end

  always begin
    #1000;
    $display("%08b %08b", A, B);
    A=A + 1;
    if(A==0) $finish;
  end
  endmodule
```

you will recognize the behavioral code right after the netlist specifying the connection of nets to the xnor and assign gates and all the necessary verilog declarations. If you press the **Simulation** button the **Icarus Verilog** simulator will be executed to compile (iverilog) and run (vvp) the simulation, a terminal window will show the simulation output, in this case the input vector A[7:0] and the grey coded B[7:0] vectors are shown. You can quit the simulator log window by pressing **'q'**.



```
VCD info: dumpfile dumpfile.vcd opened for output.
00000000 01111111
00000001 01111110
00000010 01111100
00000011 01111101
00000100 01111001
00000101 01111000
00000110 01111010
00000111 01111011
00001000 01110011
00001001 01110010
00001010 01110000
00001011 01110001
00001100 01110101
00001101 01110100
00001110 01110110
00001111 01110111
00010000 01100111
00010001 01100110
00010010 01100100
00010011 01100101
00010100 01100001
:
```

If simulation completes with no errors waveforms can be viewed. Press the **Waves** button in the top-right of the menu bar, you may add waveforms in the gtkwave window:

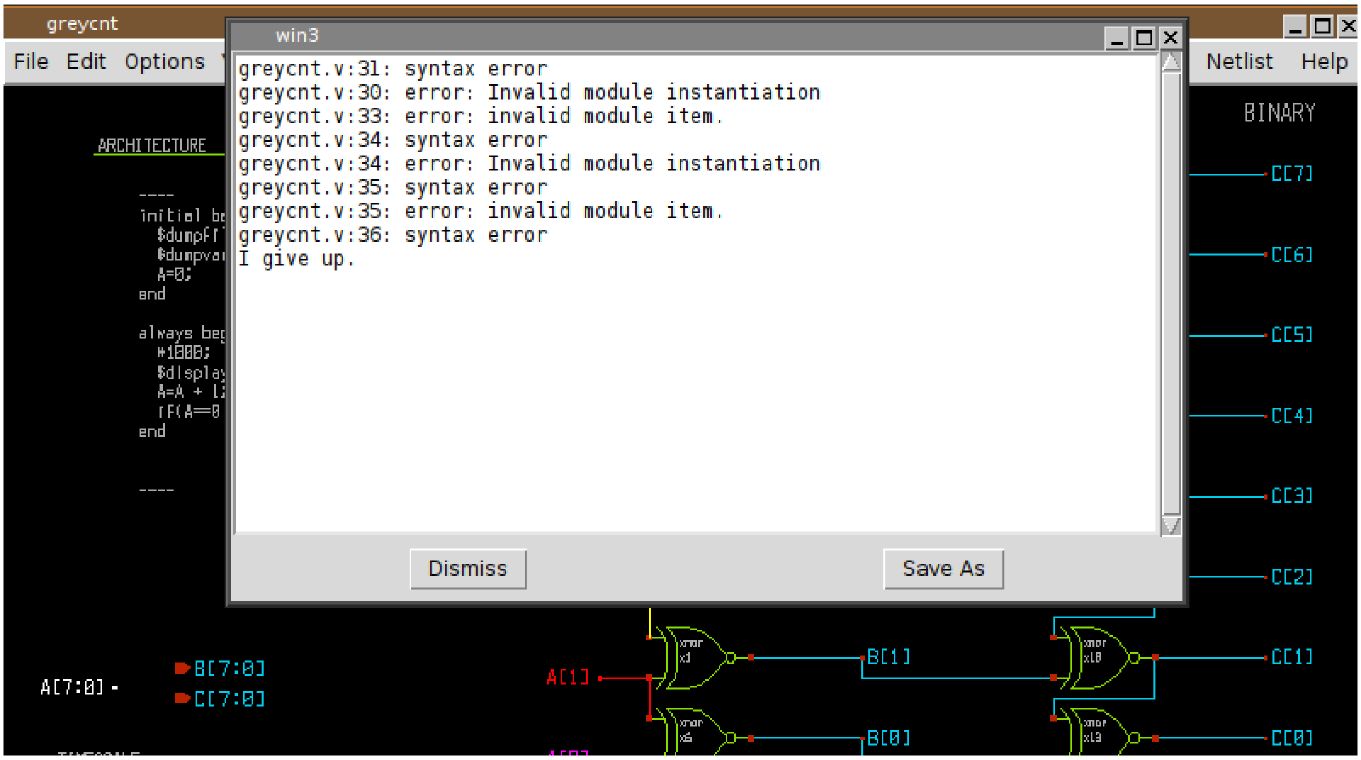


```
Time              10 ns                                                  20 ns
A[7:0]  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
B[7:0]  7F 7E 7C 7D 79 78 7A 7B 73 72 70 71 75 74 76 77 67 66 64 65 61 60 62
C[7:0]  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
```

Type	Signals
reg	A[7:0]
wire	B[7:0]
wire	C[7:0]

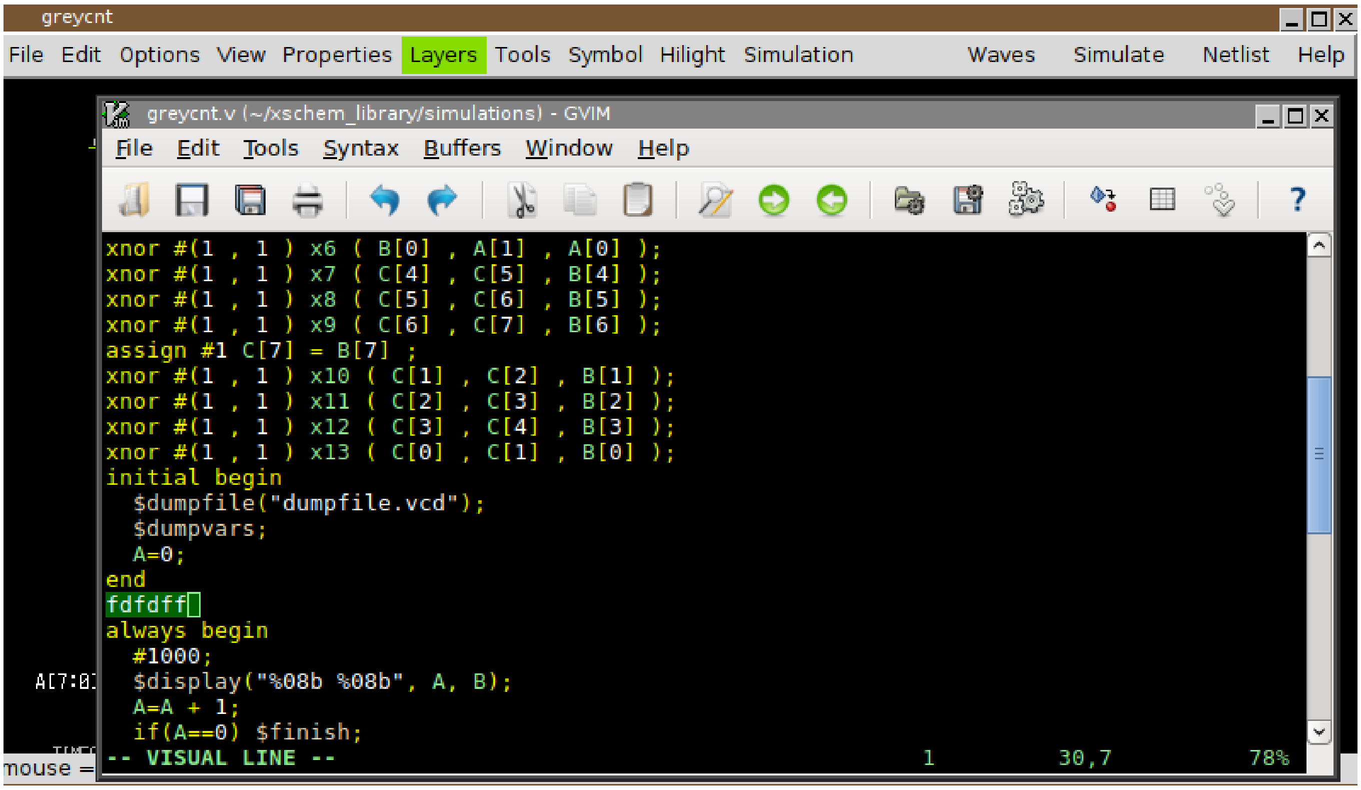
```
mouse = 650 -790 - examples/greycnt  selected: 0
```

If the schematic contains errors that the simulator can not handle instead of the simulation log a window showing the error messages from the simulator is shown:



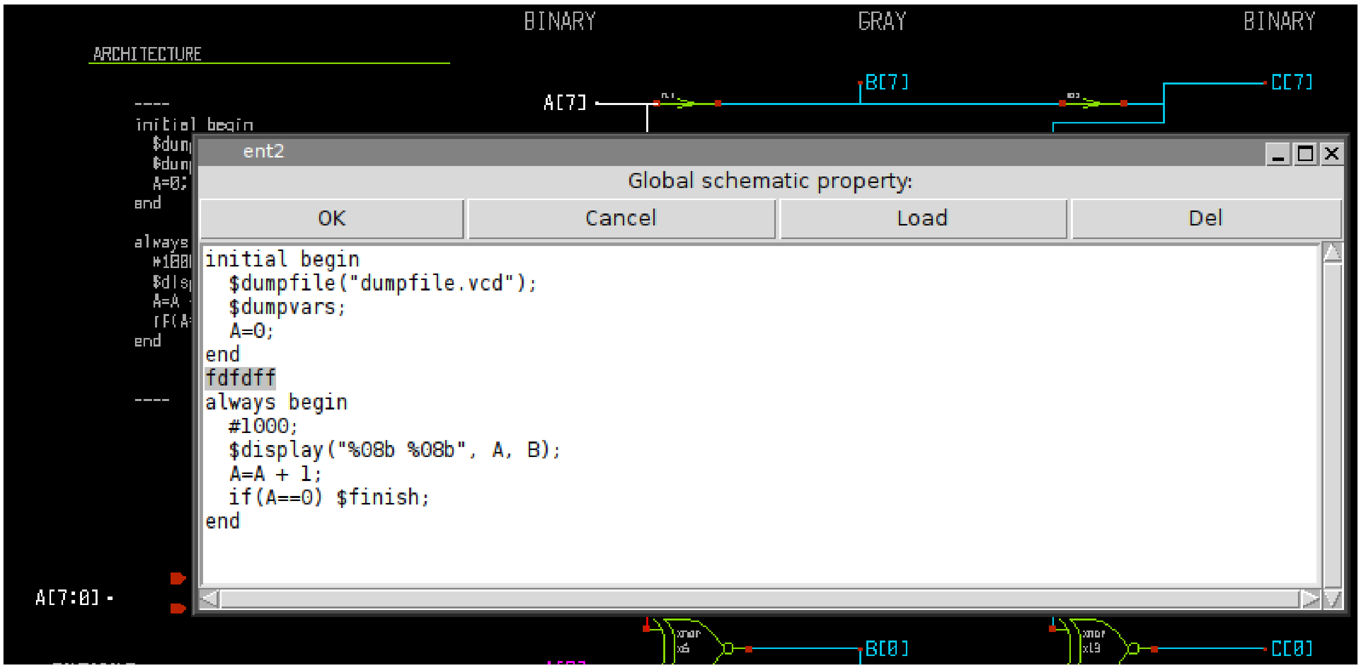
```
greycnt.v:31: syntax error
greycnt.v:30: error: Invalid module instantiation
greycnt.v:33: error: invalid module item.
greycnt.v:34: syntax error
greycnt.v:34: error: Invalid module instantiation
greycnt.v:35: syntax error
greycnt.v:35: error: invalid module item.
greycnt.v:36: syntax error
I give up.
```

To facilitate the debug you may wish to edit the netlist (**Simulation->Edit Netlist**) to locate the error, in the picture below i inserted deliberately a random string to trigger the failure:


greycnt
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
greycnt.v (~/xschem_library/simulations) - GVIM
File Edit Tools Syntax Buffers Window Help

```
xnor #(1 , 1 ) x6 ( B[0] , A[1] , A[0] );
xnor #(1 , 1 ) x7 ( C[4] , C[5] , B[4] );
xnor #(1 , 1 ) x8 ( C[5] , C[6] , B[5] );
xnor #(1 , 1 ) x9 ( C[6] , C[7] , B[6] );
assign #1 C[7] = B[7] ;
xnor #(1 , 1 ) x10 ( C[1] , C[2] , B[1] );
xnor #(1 , 1 ) x11 ( C[2] , C[3] , B[2] );
xnor #(1 , 1 ) x12 ( C[3] , C[4] , B[3] );
xnor #(1 , 1 ) x13 ( C[0] , C[1] , B[0] );
initial begin
  $dumpfile("dumpfile.vcd");
  $dumpvars;
  A=0;
end
fdfdff
always begin
  #1000;
  $display("%08b %08b", A, B);
  A=A + 1;
  if(A==0) $finish;
-- VISUAL LINE --                               1         30,7          78%
```

As you can see the error is in the behavioral code of the top level greycnt schematic, so edit the global property (**'q'** key with no component selected) and fix the error.

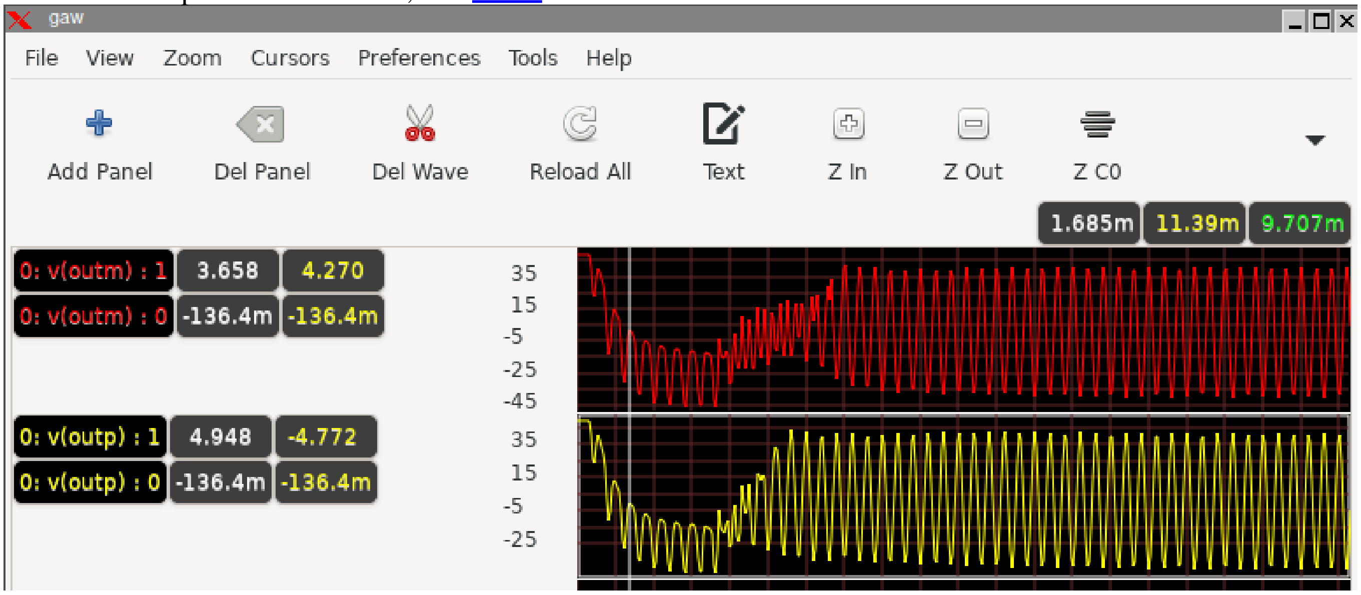


```
initial begin
  $dumpfile("dumpfile.vcd");
  $dumpvars;
  A=0;
end
fdfdff
always begin
  #1000;
  $display("%08b %08b", A, B);
  A=A + 1;
  if(A==0) $finish;
end
```

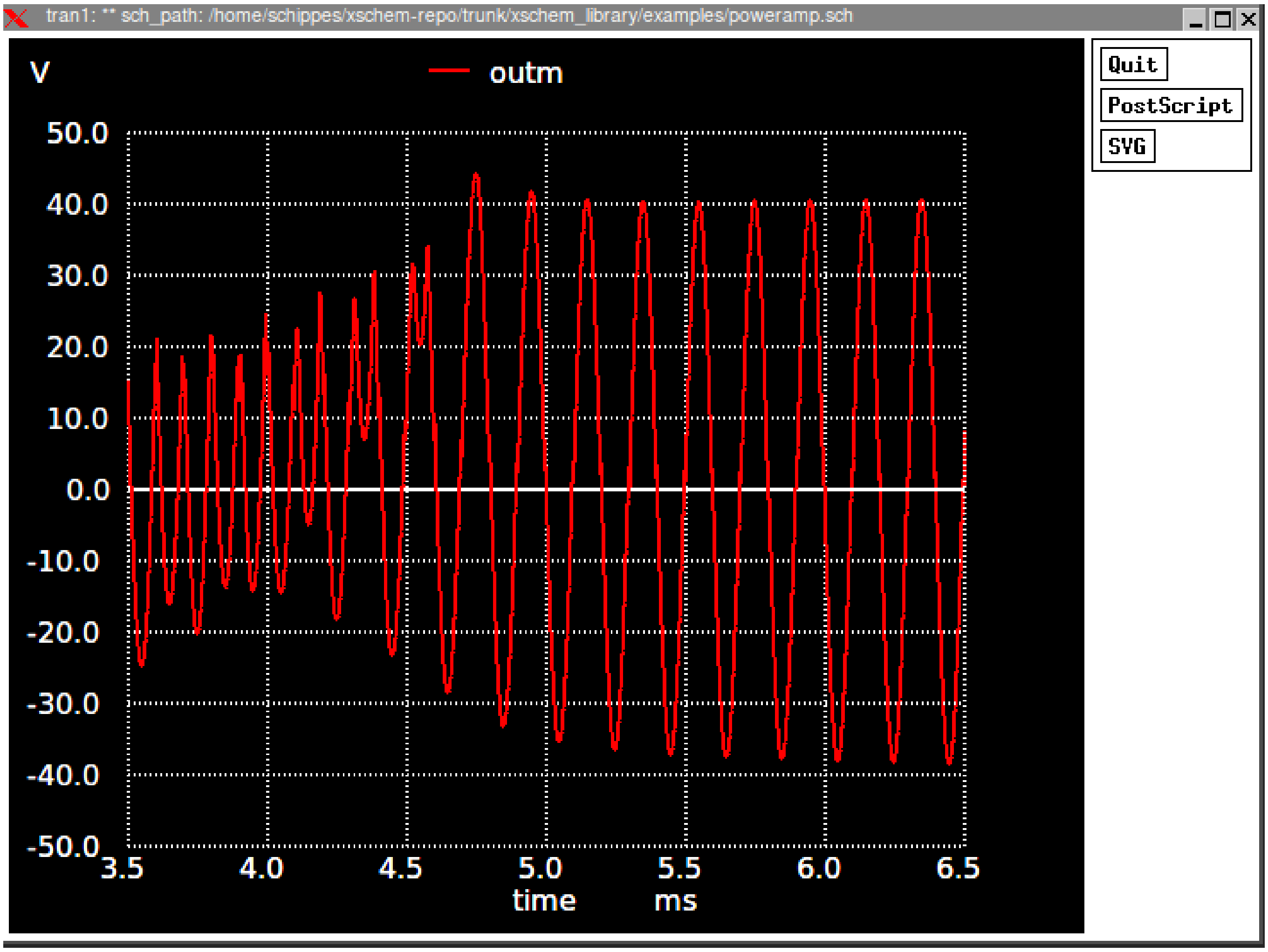
PREV UP NEXT

VIEWING SIMULATION DATA WITH XSCHEM

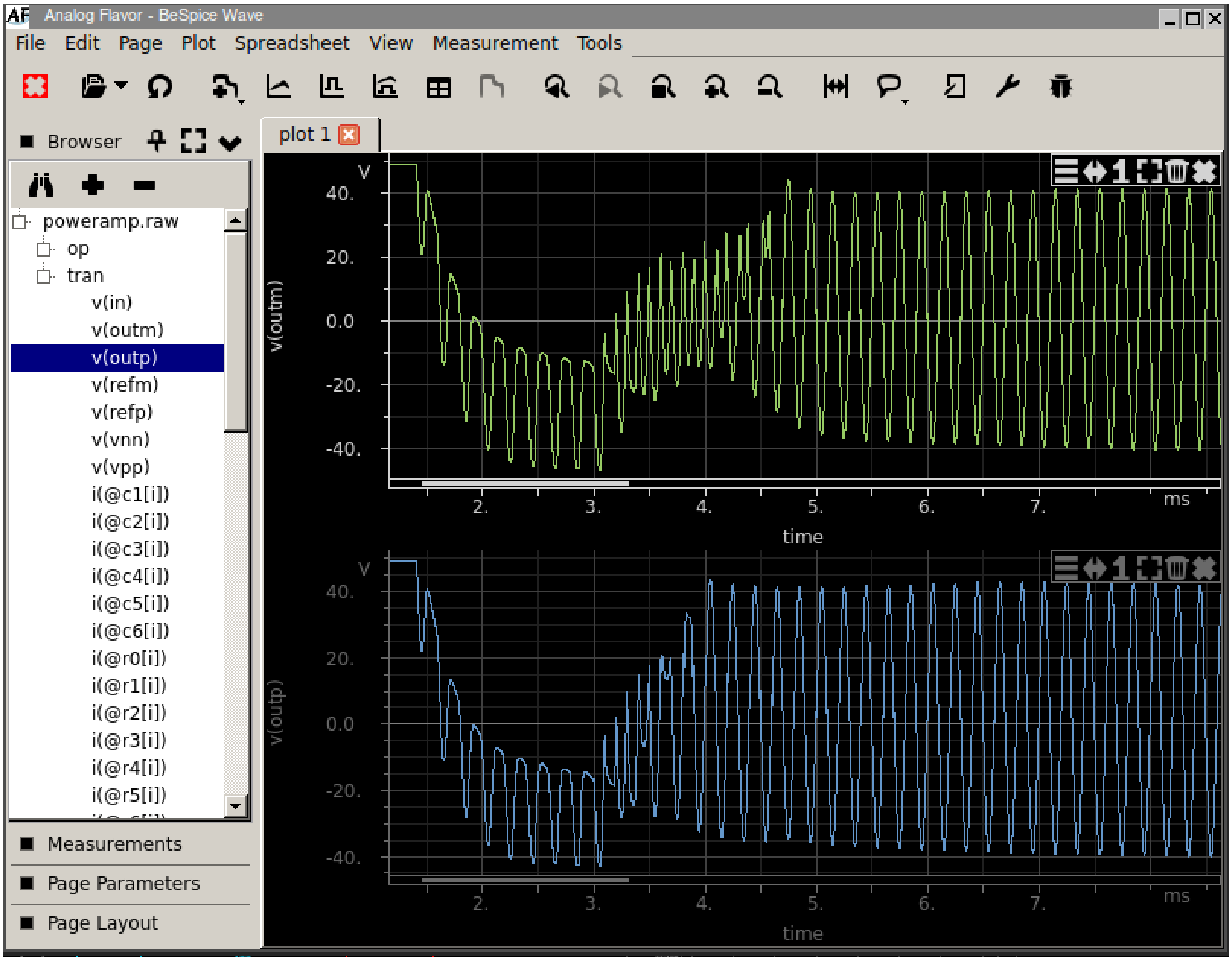
Usually when a spice simulation is done you want to see the results, this is usually accomplished with a waveform viewer. There are few open source viewers, like GAW...



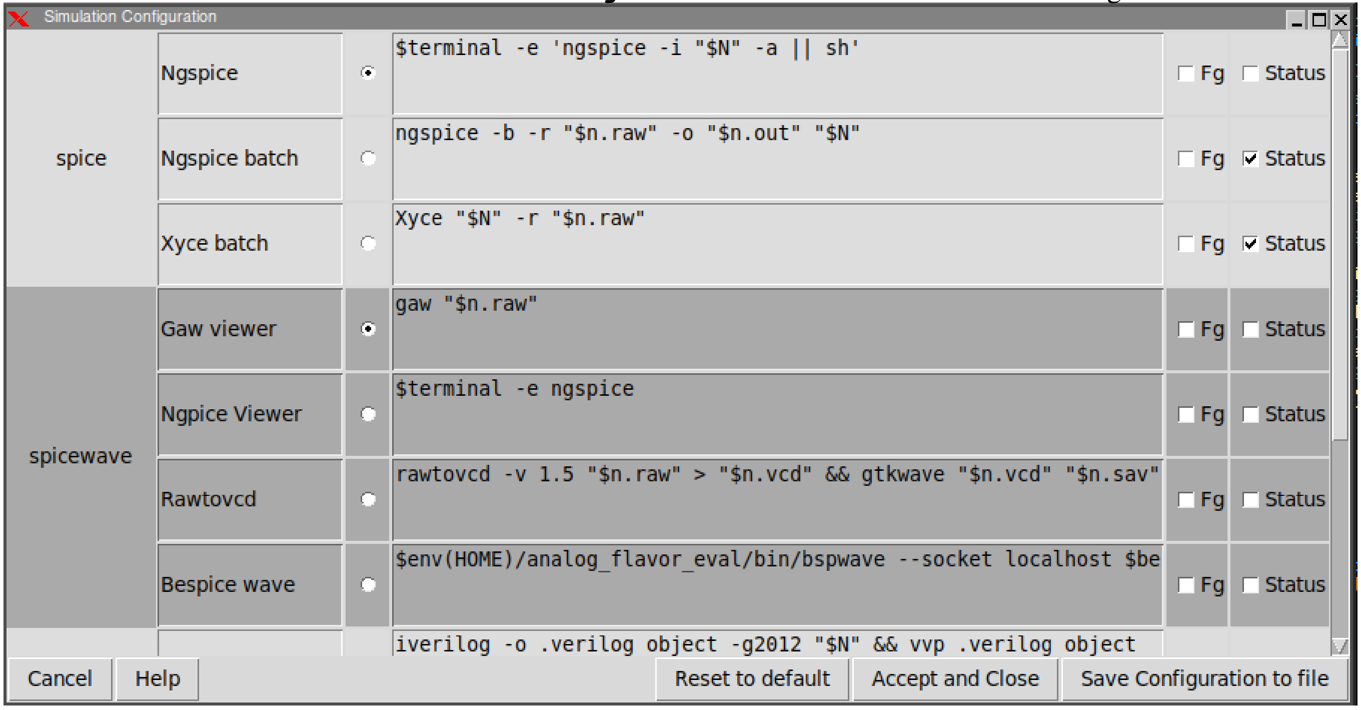
...Or ngspice internal plotting facilities:



There is also an interesting commercial product from Analog Flavor, called BeSpice (bspwave) that offers a free of charge one year evaluation license for non commercial use:

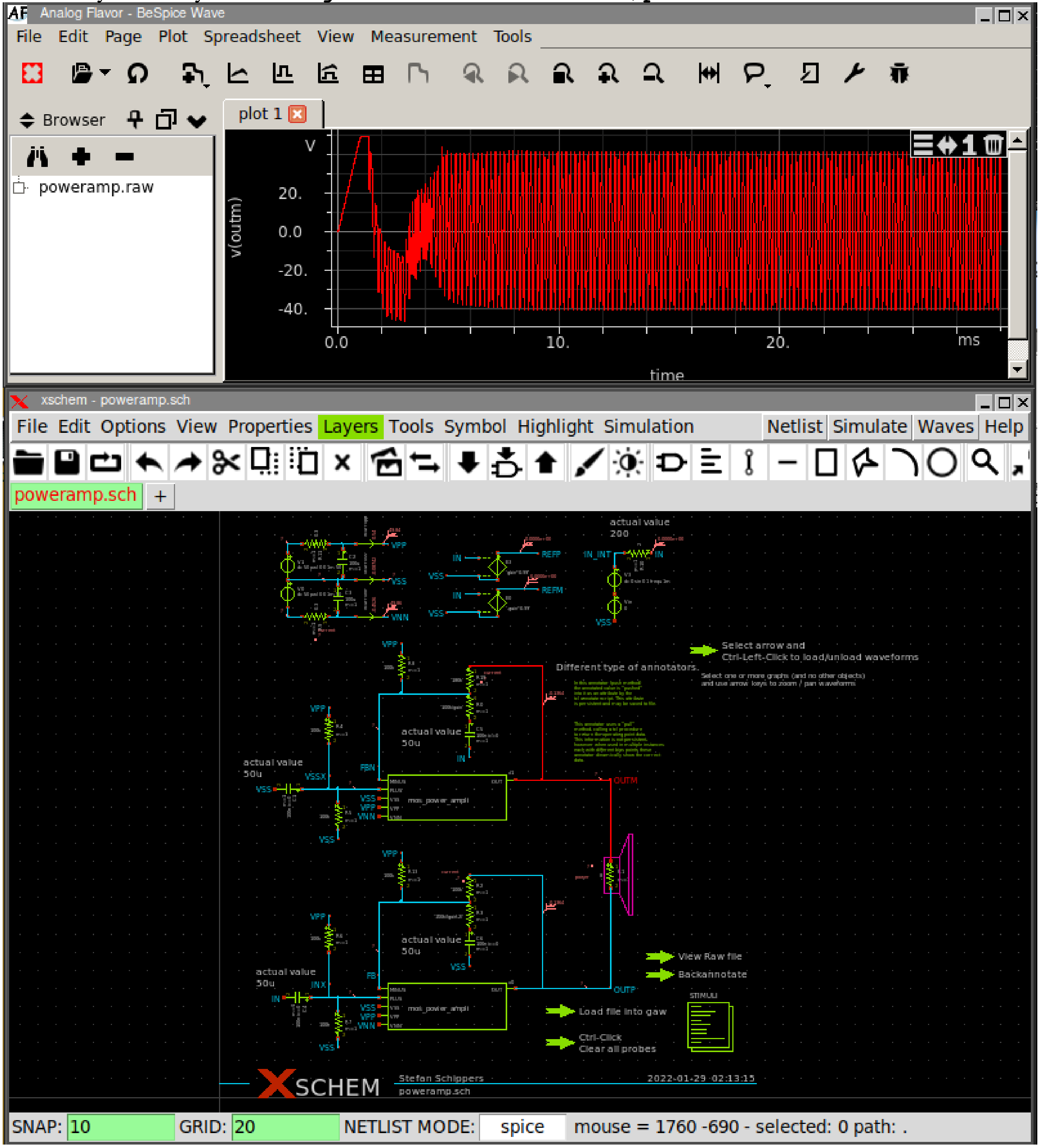


All these waveform viewers are supported by xschem and more can be added, just by giving the command line to start the viewer to xschem in the **`Simulation-> Configure simulators and tools`** dialog:


Simulation Configuration

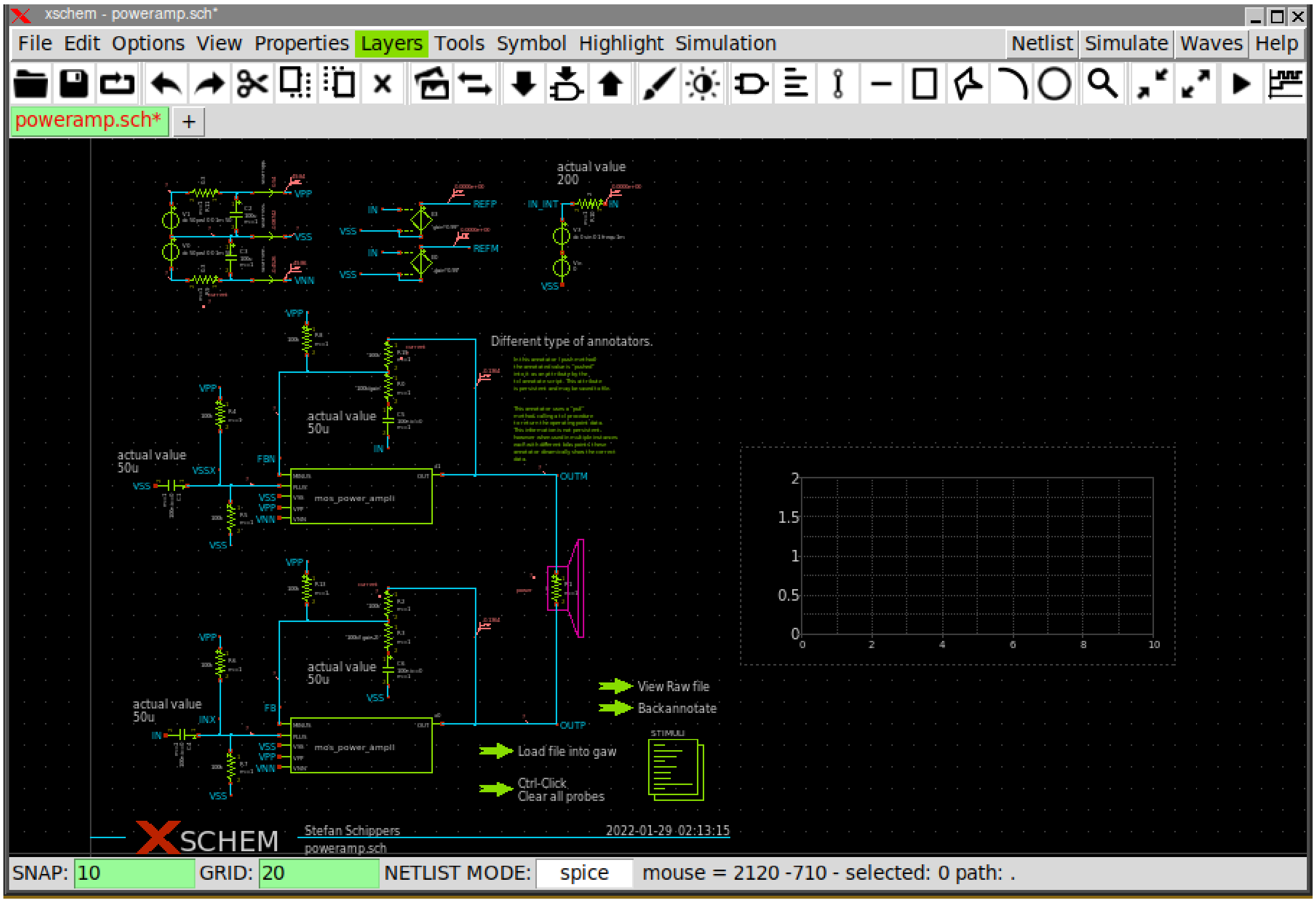
spice	Ngspice	●	`$terminal -e 'ngspice -i "$N" -a || sh'`	☐ Fg	☐ Status
	Ngspice batch	○	`ngspice -b -r "$n.raw" -o "$n.out" "$N"`	☐ Fg	✓ Status
	Xyce batch	○	`Xyce "$N" -r "$n.raw"`	☐ Fg	✓ Status
spicewave	Gaw viewer	●	`gaw "$n.raw"`	☐ Fg	☐ Status
	Ngpice Viewer	○	`$terminal -e ngspice`	☐ Fg	☐ Status
	Rawtovcd	○	`rawtovcd -v 1.5 "$n.raw" > "$n.vcd" && gtkwave "$n.vcd" "$n.sav"`	☐ Fg	☐ Status
	Bespice wave	○	`$env(HOME)/analog_flavor_eval/bin/bspwave --socket localhost $be`	☐ Fg	☐ Status
			`iverilog -o .verilog_object -g2012 "$N" && vvp .verilog_object`		

For gaw and bespice xschem can automatically send nets to the viewer by clicking a net on the schematic and pressing the `Alt-G` key bind or by menu `Hilight->Send selected nets/pins to Viewer`

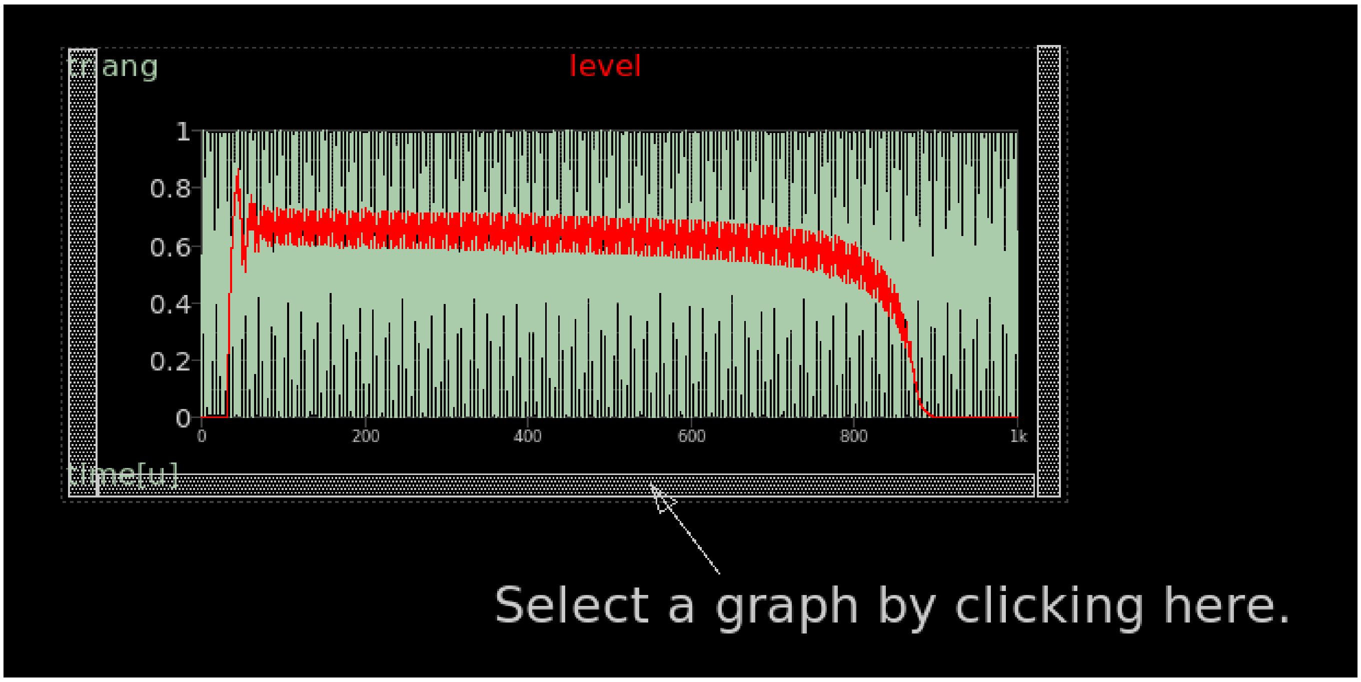

Analog Flavor - BeSpice Wave
File Edit Page Plot Spreadsheet View Measurement Tools
Browser
poweramp.raw
plot 1
v(outm)
time
ms
xschem - poweramp.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Simulate Waves Help
poweramp.sch
Select arrow and Ctrl-Left-Click to load/unload waveforms
Different type of annotators.
View Raw file
Backannotate
Load file into gaw
Ctrl-Click Clear all probes
SCHEM
Stefan Schippers
poweramp.sch
2022-01-29 02:13:15
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 1760 -690 - selected: 0 path: .

Using XSCHEM's internal graph functions

Xschem can now display waveforms by itself in the drawing area. in the Simulation menu there is an entry to add a graph: `Graph -> Add waveform graph`. When this menu is pressed a box can be placed in the schematic:


xschem - poweramp.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Simulate Waves Help
poweramp.sch*
actual value 200
Different type of annotators.
actual value 50u
View Raw file
Backannotate
Load file into gaw
Ctrl-Click Clear all probes
STIMULI
Stefan Schippers
poweramp.sch
2022-01-29 02:13:15
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 2120 -710 - selected: 0 path: .

Xschem graphs are embedded into a rectangle object. Resizing graphs is done in the same way as resizing a rectangle object. See the related page. To select a graph (to delete it or move it to a different position) click the left mouse button while in the area shown in this picture:

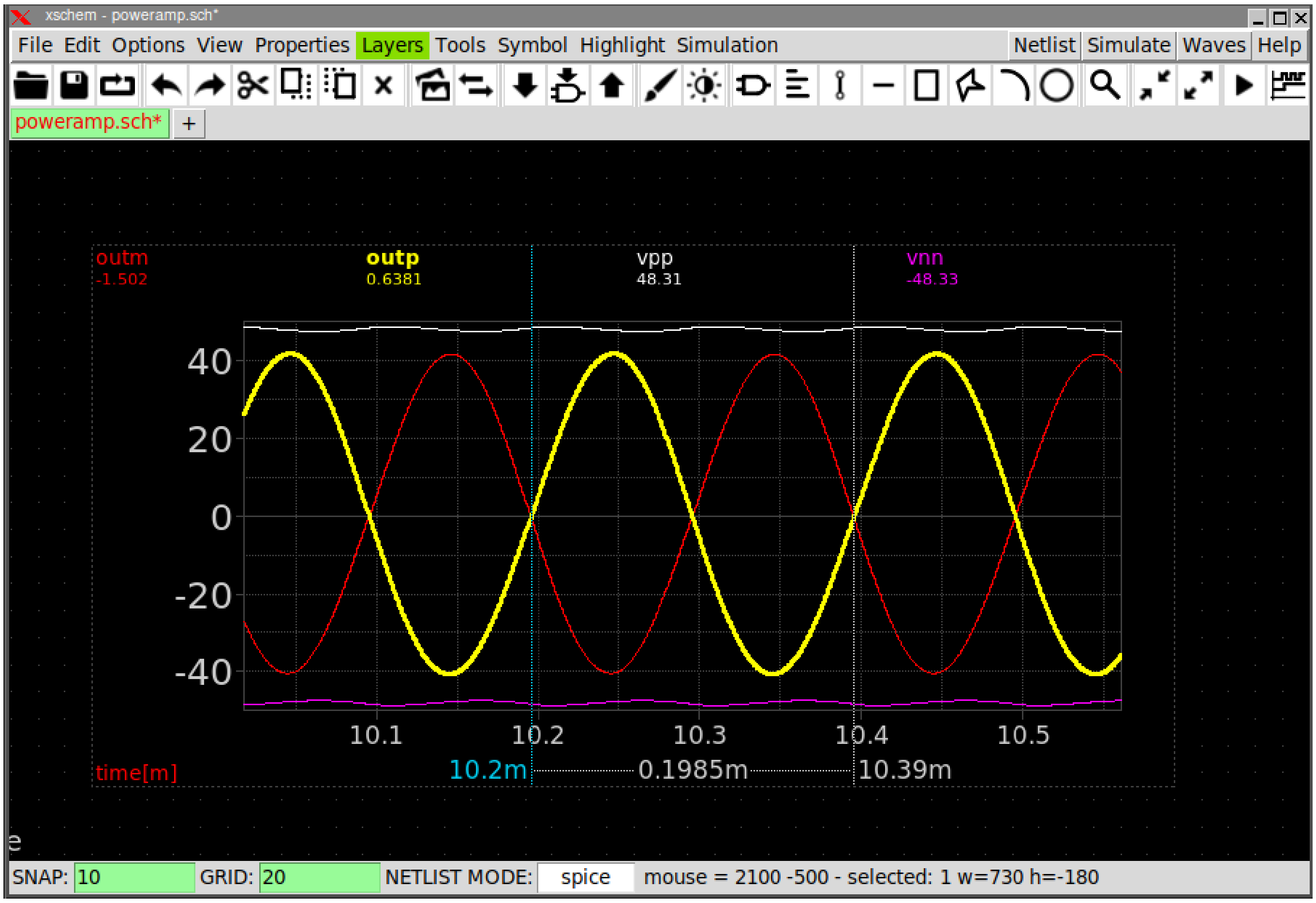


The next step is loading the simulation data, This is done by menu **`Waves->Op | Ac | Dc | Tran | Tran | Noise | Sp`** . This command loads the user selected .raw file produced by a ngspice/Xyce simulation.

Ensure the **`circuit.raw`** is saved in binary format (no **`set filetype=ascii`** in your testbench)

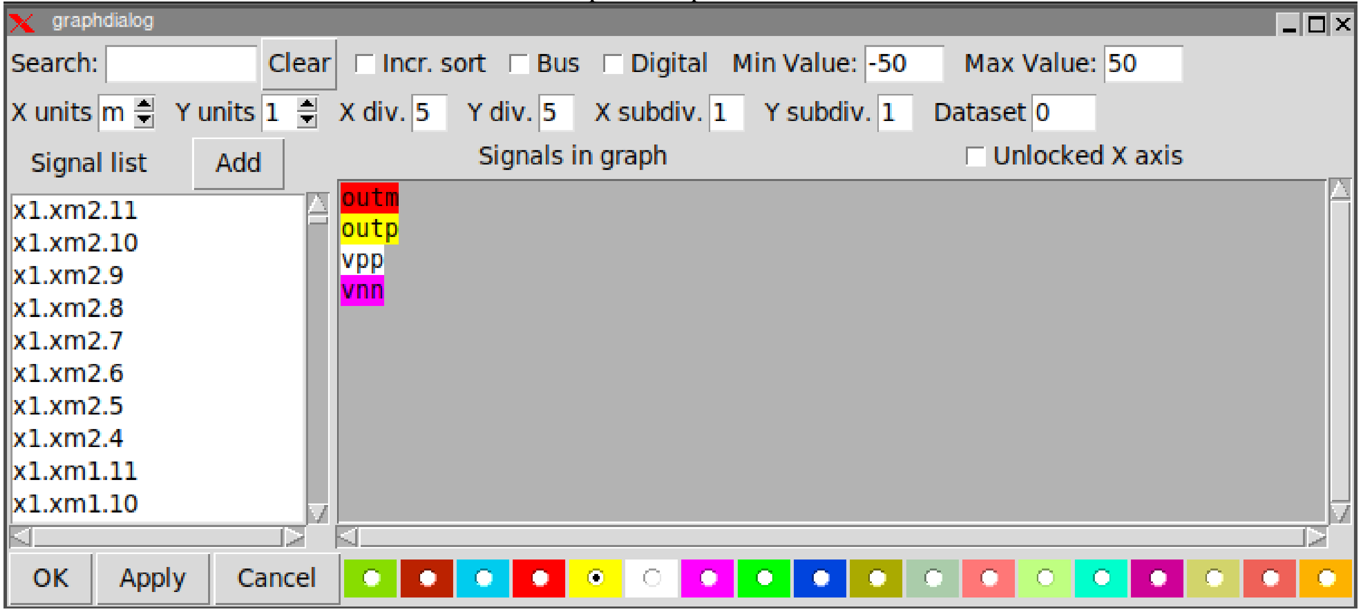
The raw file is usually located in the simulation/netlisting directory **`Simulation ->set netlist dir`**.
After placing a graph box and loading simulation data a wave can be added. If you place the mouse on the inside of the box, close to the bottom/left/right edges and click the graph will be selected. You can also select a graph by dragging a selection rectangle all around it. This tells xschem where new nodes to be plotted will go, in case you have multiple graphs. Then, select a node or a net label, press 'Alt-G', the net will be added to the graph. Here after a list of commands you can perform in a graph to modify the viewport. These commands are active when the mouse is Inside the graph (you will notice the mouse pointer changing from an arrow to a **`+`**). if the mouse is outside the graph the usual Xschem functions will be available to operate on schematics:

- Pressing **`f`** with the mouse in the middle of the graph area will do a full X-axis zoom.
- Pressing **`f`** with the mouse on the left of the Y axis will do a full Y-axis zoom.
- Pressing **`Left/Right`** or **`Up/Down`** arrow keys while the mouse is inside a graph will move the waveforms to the left/right or zoom in/zoom out respectively.
- Pressing **`Left/Right`** or **`Up/Down`** arrow keys while the mouse is on the left of the Y-axis will move the waveforms or zoom in/zoom out in the Y direction respectively.
- Pressing the **`left`** mouse button while the pointer is in the center of the graph will move the waves left or right following the pointer X movement.
- Pressing the **`left`** mouse button while the pointer is on the left of the Y-axis will move the waves high or low following the pointer Y movement.
- Doing the above with the **`Shift`** key pressed will zoom in/out instead of moving.
- pressing **`a`** and/or **`b`** will show a vertical cursor. The sweep variable difference between the **`a`** and the **`b`** cursor is shown and the values of all signals at the X position of the **`a`** cursor is shown.
- Double clicking the **`left`** mouse button with the pointer above a wave label will allow to change its color.
- Pressing the **`right`** mouse button with the pointer above a wave label will show it in bold.
- Double clicking the **`left`** mouse button with the pointer in the middle of the graph will show a configuration dialog box, where you can change many graph parameters.
- Pressing the **`right`** mouse button in the graph area and dragging some distance in the X direction will zoom in the waveforms to that X range.



SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 2100 -500 - selected: 1 w=730 h=-180

The graph configuration dialog box which is shown by **left** button double clicking inside the graph, allows to change many graph attributes, like number of X/Y labels, minor ticks, wave colors, add waves from the list of waves found in the raw file, select the dataset to show in case of multiple sweep simulations and more.



Min Value: -50 Max Value: 50
X units m Y units 1 X div. 5 Y div. 5 X subdiv. 1 Y subdiv. 1 Dataset 0

x1.xm2.11
x1.xm2.10
x1.xm2.9
x1.xm2.8
x1.xm2.7
x1.xm2.6
x1.xm2.5
x1.xm2.4
x1.xm1.11
x1.xm1.10

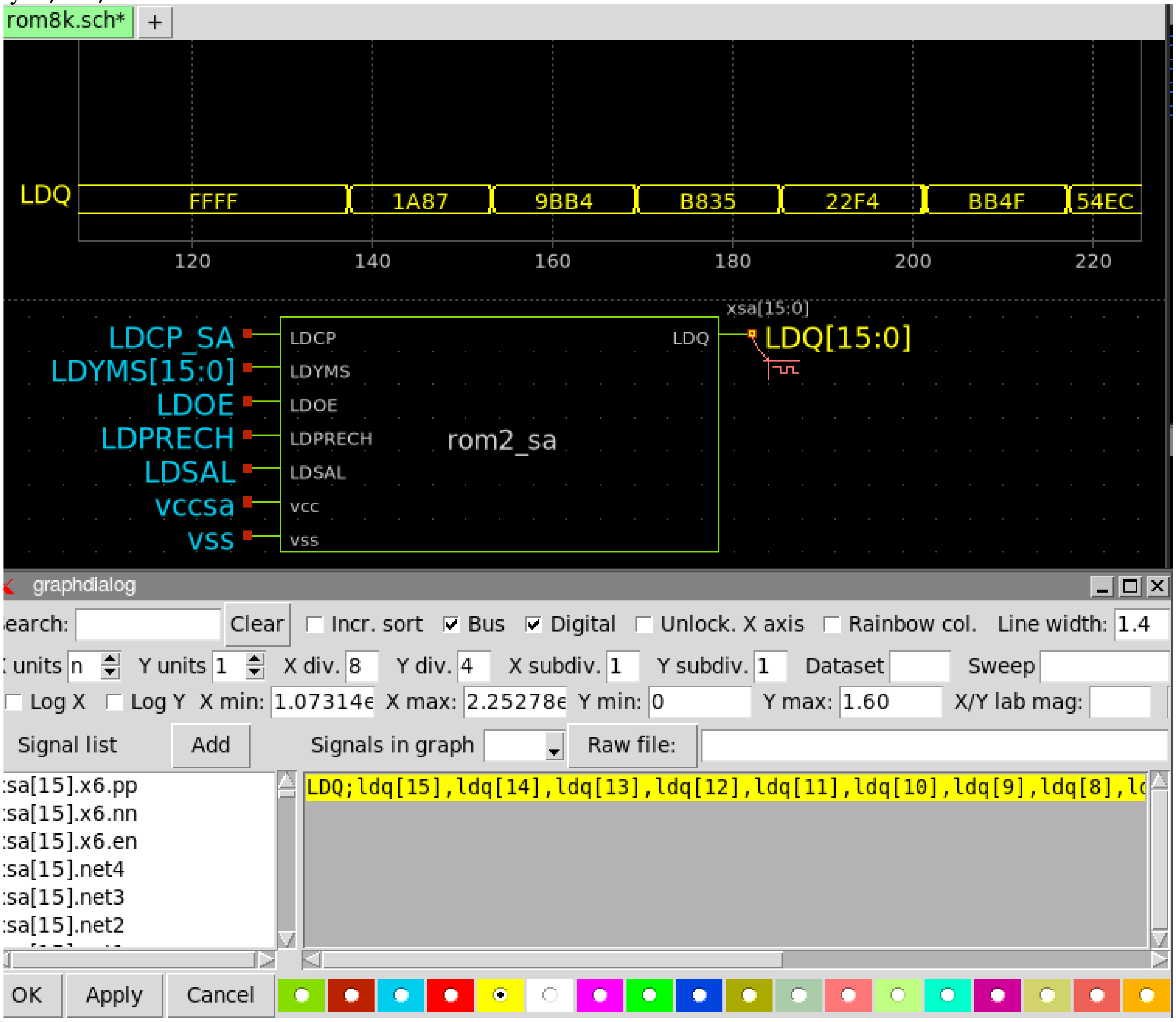
outm
outp
vpp
vnn

The text area with the colored wave names is just a text widget. You can manually edit it to add / remove waves, or you can place the cursor somewhere in the text, select some waves from the listbox on the left, press the **Add** button to have these waves added. If you place the insertion cursor in the middle of a node name in the text area, you can click the color radio buttons on the bottom to change the color. The **Search** entry can be used to restrict the list of nodes displayed in the listbox. The Search entry supports regular expression patterns. For example, **^X** will match all nodes that begin with

X, **xm[0-9]\.** will match all nodes containing xm followed by one digit and a dot.

Display bus signals

If you have a design where digital signals are present you might want to group some of these to form a bus and display these bundled signals. After placing a graph box and loading the simulation data as explained above, left-double click the graph to show the configuration dialog, check the **bus** and **digital** check boxes, use the **Search** text entry to restrict the list of signals, then select all the signals you want to show as a bus and click the **Add** button. Also set the **Min value** and the **Max value** of the signals in the bus. This information is needed by Xschem to calculate the logic high and logic low thresholds. Currently the logic '1' is set at 80% of the signal min-max range and the logic '0' level is set at 20% of the signal range. After pressing the **Add** button a bus is shown in the text area. The first field is a template **BUS_NAME** that you should change to give a meaningful name to the bus. The bus name is separated from the rest of bits by a **,** or **;** character.



earch: [ ] ☐ Incr. sort ☑ Bus ☑ Digital ☐ Unlock. X axis ☐ Rainbow col. Line width: 1.4

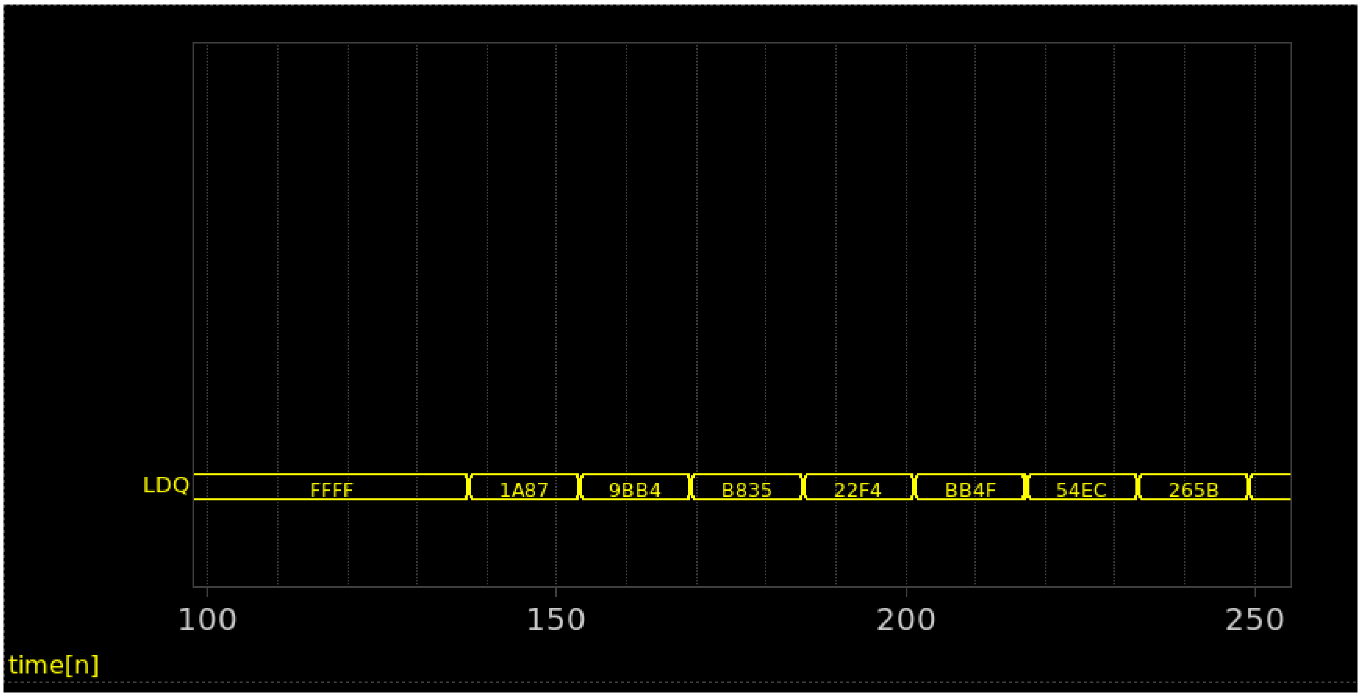
units n Y units 1 X div. 8 Y div. 4 X subdiv. 1 Y subdiv. 1 Dataset Sweep

☐ Log X ☐ Log Y X min: 1.07314e X max: 2.25278e Y min: 0 Y max: 1.60 X/Y lab mag:

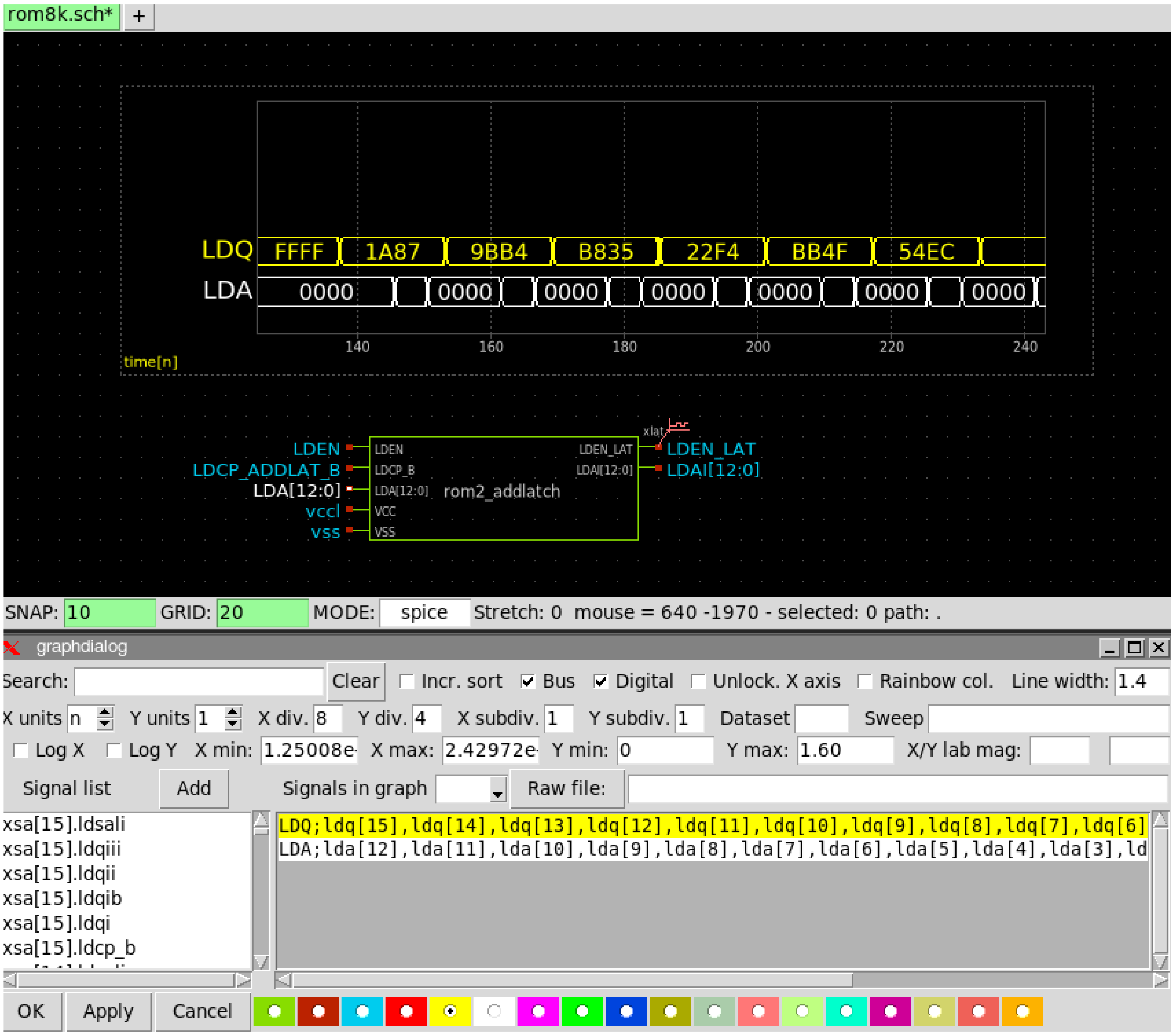
```
:sa[15].x6.pp
:sa[15].x6.nn
:sa[15].x6.en
:sa[15].net4
:sa[15].net3
:sa[15].net2
```

```
LDQ;ldq[15],ldq[14],ldq[13],ldq[12],ldq[11],ldq[10],ldq[9],ldq[8],lc
```

You will then see your bussed signal in the graph:



If you have bussed signals in the schematic , like **`LDA[12:0]`** and your graph has the **`Digital`** and **`Bus`** checkboxes set you can simply add the LDA bus to the graph by clicking the net in the schematic (with the configuration dialog open) and pressing **`Alt-G`**:



SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 640 -1970 - selected: 0 path: .

Search: Clear ☐ Incr. sort ☑ Bus ☑ Digital ☐ Unlock. X axis ☐ Rainbow col. Line width: 1.4

X units n Y units 1 X div. 8 Y div. 4 X subdiv. 1 Y subdiv. 1 Dataset Sweep

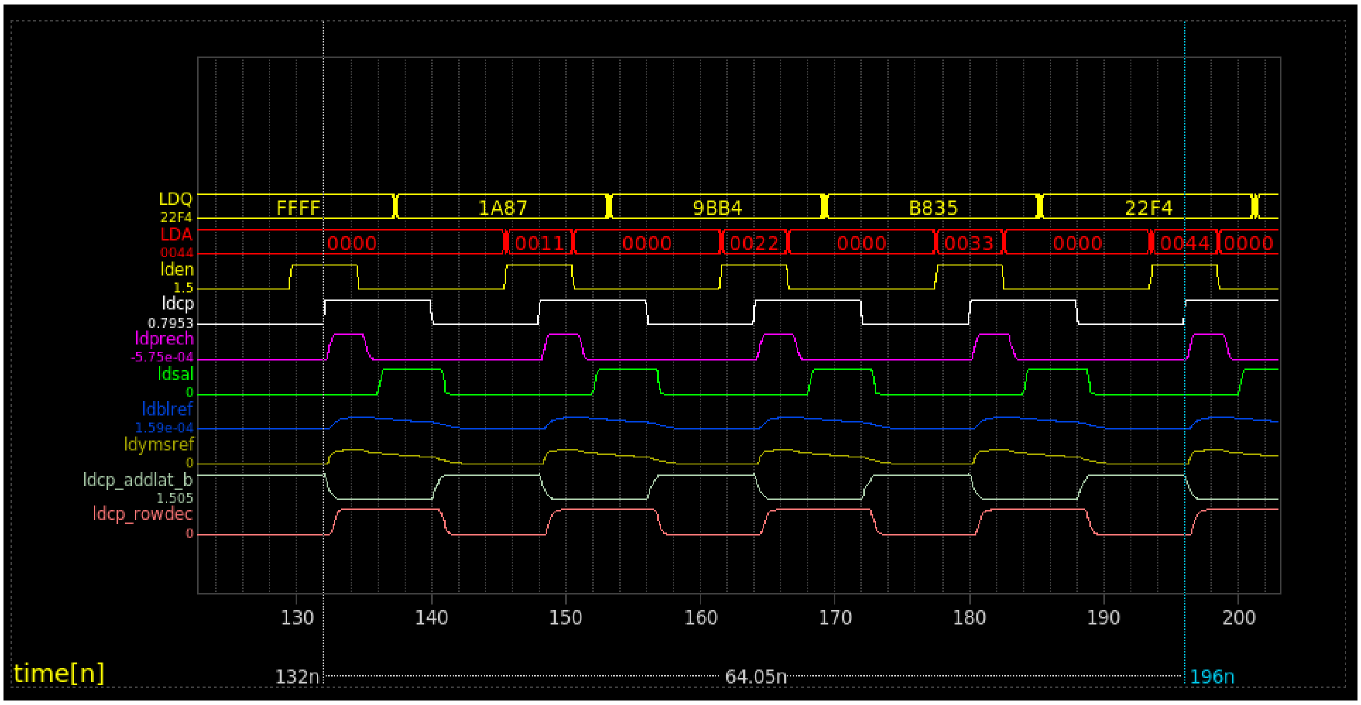
☐ Log X ☐ Log Y X min: 1.25008e X max: 2.42972e Y min: 0 Y max: 1.60 X/Y lab mag:

Signal list Add Signals in graph Raw file:

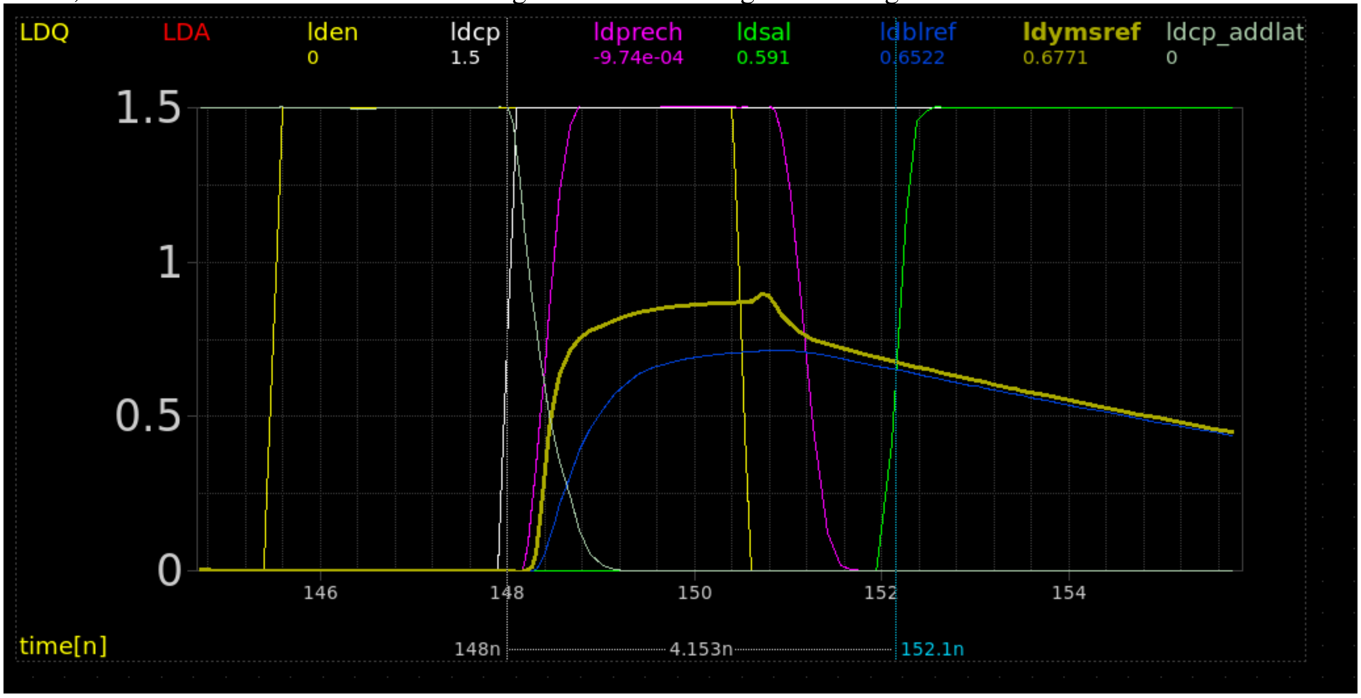
xsa[15].ldsali
xsa[15].ldqiii
xsa[15].ldqii
xsa[15].ldqib
xsa[15].ldqi
xsa[15].ldcp_b

```
LDQ;ldq[15],ldq[14],ldq[13],ldq[12],ldq[11],ldq[10],ldq[9],ldq[8],ldq[7],ldq[6]
LDA;lda[12],lda[11],lda[10],lda[9],lda[8],lda[7],lda[6],lda[5],lda[4],lda[3],ld
```

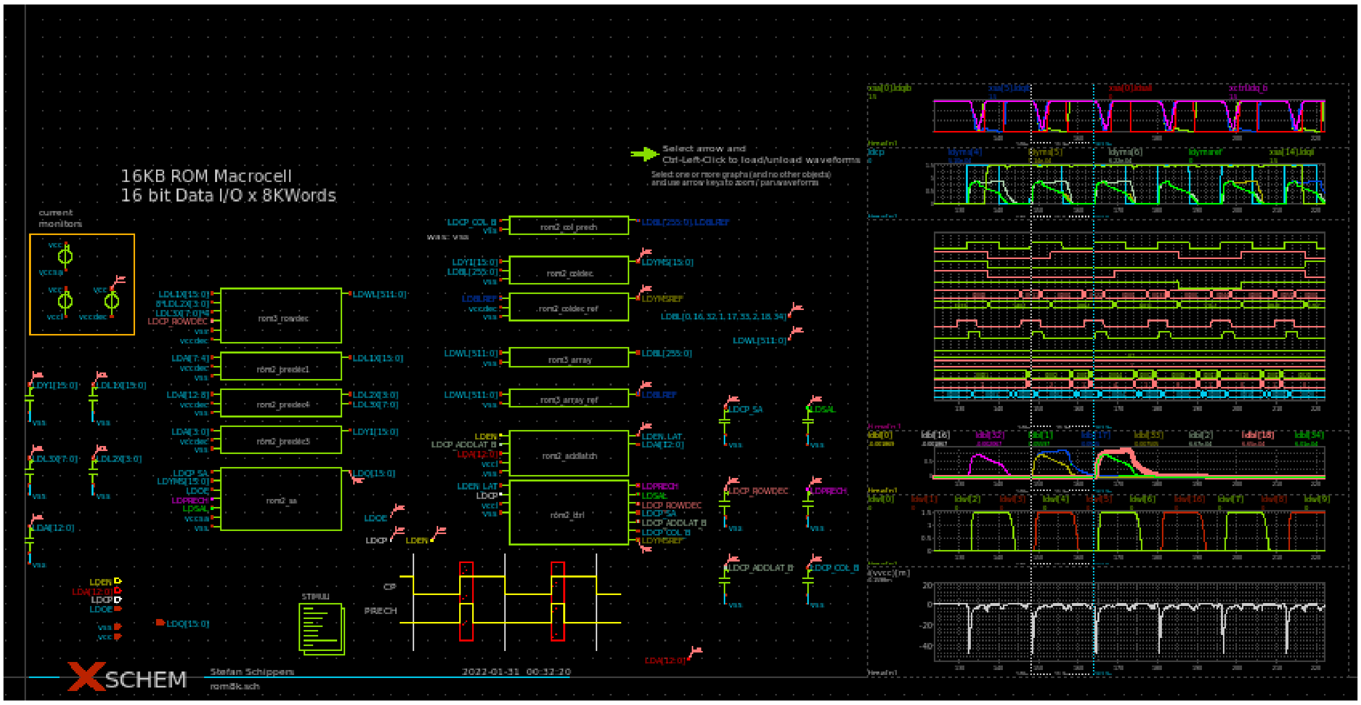
You can add many signals to see them stacked in a very compact view:



It is possible to switch the graph to analog mode, by unchecking the **Digital** checkbox in the graph configuration dialog, to better see the waveforms. Switching back to Digital yields the previous view. In analog mode buses are not shown, but are not lost. You will see them again when switching back to Digital mode.



Many graphs can be created in a schematic, and the configuration of all graphs (viewport, list of signals, colors) is saved together with the schematic. If you re-run a simulation just unloading/loading the data from the simulation menu will update the waveforms.



Expression evaluation on waves

It is possible to enter math expressions combining simulation data, for example multiply current and voltage to get the power. The syntax of expressions uses postfix (RPN) notation. When entering an expression use double quotes in the graph edit attribute dialog box, so the expression will be considered as a single new wave to display. Operands are loaded onto a stack like structure and then evaluated. The syntax is:

```
"alias_name;operand operand operator ..."
```

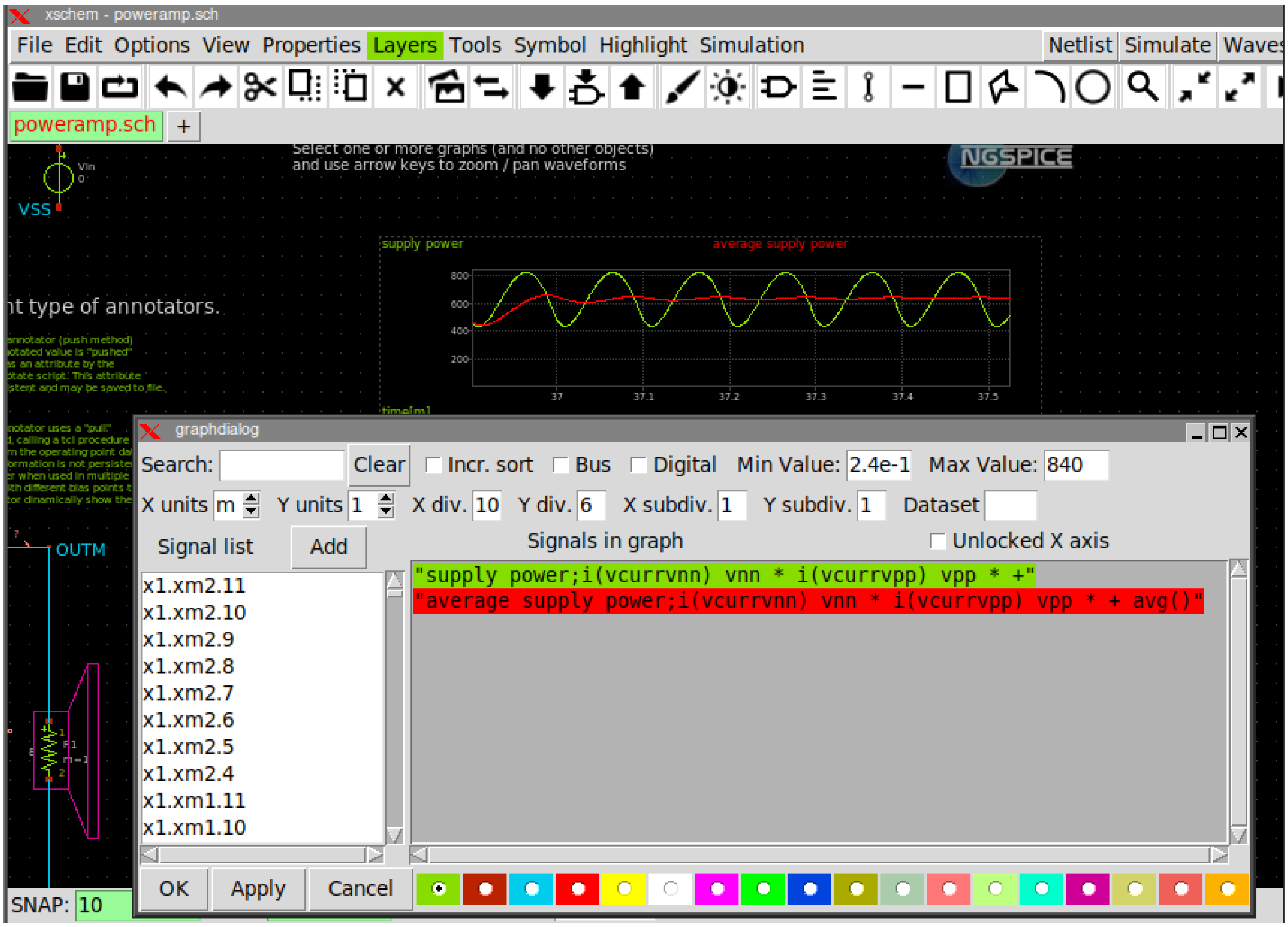
Example:

```
"supply power;i(vcurrvnn) vnn * i(vcurrvpp) vpp * +"
```

that means: i(vcurrvnn) * vnn + i(vcurrvpp) * vpp.

```
"i(vcurrvnn) 1e6 *"
```

that means: i(vcurrvnn) * 1e6.



Select one or more graphs (and no other objects)
and use arrow keys to zoom / pan waveforms

Search: Incr. sort Bus Digital Min Value: 2.4e-1 Max Value: 840
X units m Y units 1 X div. 10 Y div. 6 X subdiv. 1 Y subdiv. 1 Dataset
Unlocked X axis

Signal list:
x1.xm2.11
x1.xm2.10
x1.xm2.9
x1.xm2.8
x1.xm2.7
x1.xm2.6
x1.xm2.5
x1.xm2.4
x1.xm1.11
x1.xm1.10

Signals in graph:
```
"supply power;i(vcurrvnn) vnn * i(vcurrvpp) vpp * +"
"average supply power;i(vcurrvnn) vnn * i(vcurrvpp) vpp * + avg()"
```

The optional **alias_name** is just a string to display as the wave label instead of the whole expression. The following operators are defined:
3 argument operators:

- **?** Conditional expression: X cond Y ? --> return X if cond = 1 else Y

2 argument operators:

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **<** Lower than
- **>** Greater than
- **==** Equal
- **!=** Not equal
- **<=** Lower or equal
- **>=** Greater or equal
- **==** Equal
- **==** Equal
- ****** Exponentiation
- **max()** Take the maximum of the two operators
- **min()** Take the minimum of the two operators
- **exch()** Exchange top 2 operands on stack

- **`ravg()`** Running average of over a specified time window
- **`del()`** Delete waveform by specified quantity on the X-axis
- **`re()`** Return real part of complex number specified as magnitude and phase (in deg)
- **`im()`** Return imaginary part of complex number specified as magnitude and phase (in deg)

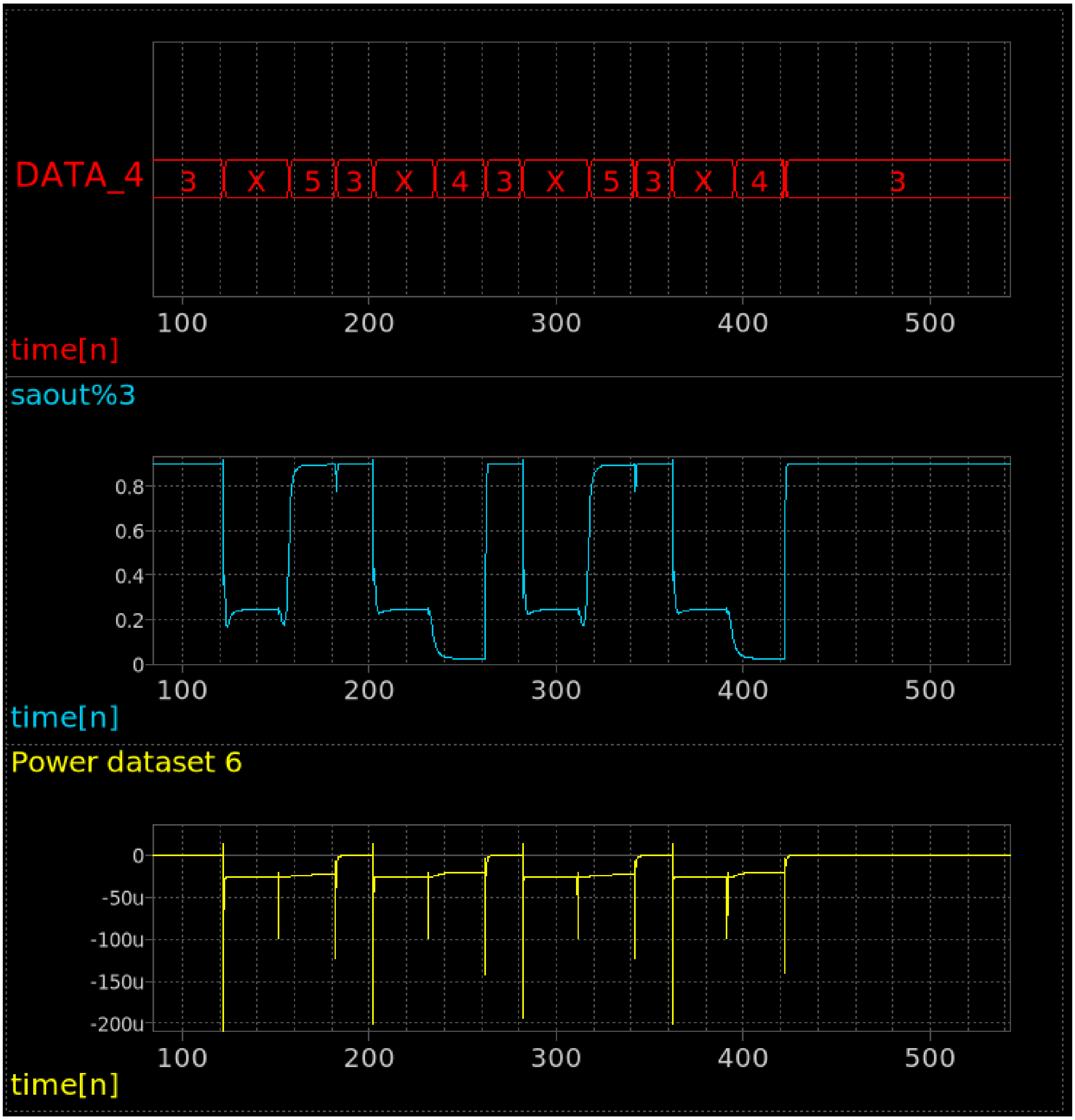
1 argument operators:

- **`sin()`** Trig. sin function
- **`cos()`** Trig. cos function
- **`tan()`** Trig. tan function
- **`sinh()`** Hyp. sin function
- **`cosh()`** Hyp. cos function
- **`tanh()`** Hyp. tan function
- **`asinh()`** Inv hyp. sin function
- **`acosh()`** Inv hyp. cos function
- **`atanh()`** Inv hyp. tan function
- **`asin()`** Inverse trig. sin function
- **`acos()`** Inverse trig. cos function
- **`atan()`** Inverse trig. tan function
- **`cph()`** Continuous phase. Instead of [-180, +180] discontinuities make phase continuous
- **`sqrt()`** Square root
- **`sgn()`** Sign
- **`abs()`** Absolute value
- **`exp()`** Base-e Exponentiation
- **`ln()`** Base-e logarithm
- **`log10()`** Base 10 logarithm
- **`idx()`** point number (0, 1, 2, ...) in vector
- **`db20()`** Value in deciBel (20 * log10(n))
- **`avg()`** Average
- **`prev()`** Delay waveform by one point (at any x-axis position take the previous value)
- **`deriv()`** Derivative w.r.t. graph sweep variable
- **`deriv0()`** Derivative w.r.t. simulation (index 0) sweep variable
- **`integ()`** Integration
- **`dup()`** Duplicate last element on stack

Display a specific dataset for a node

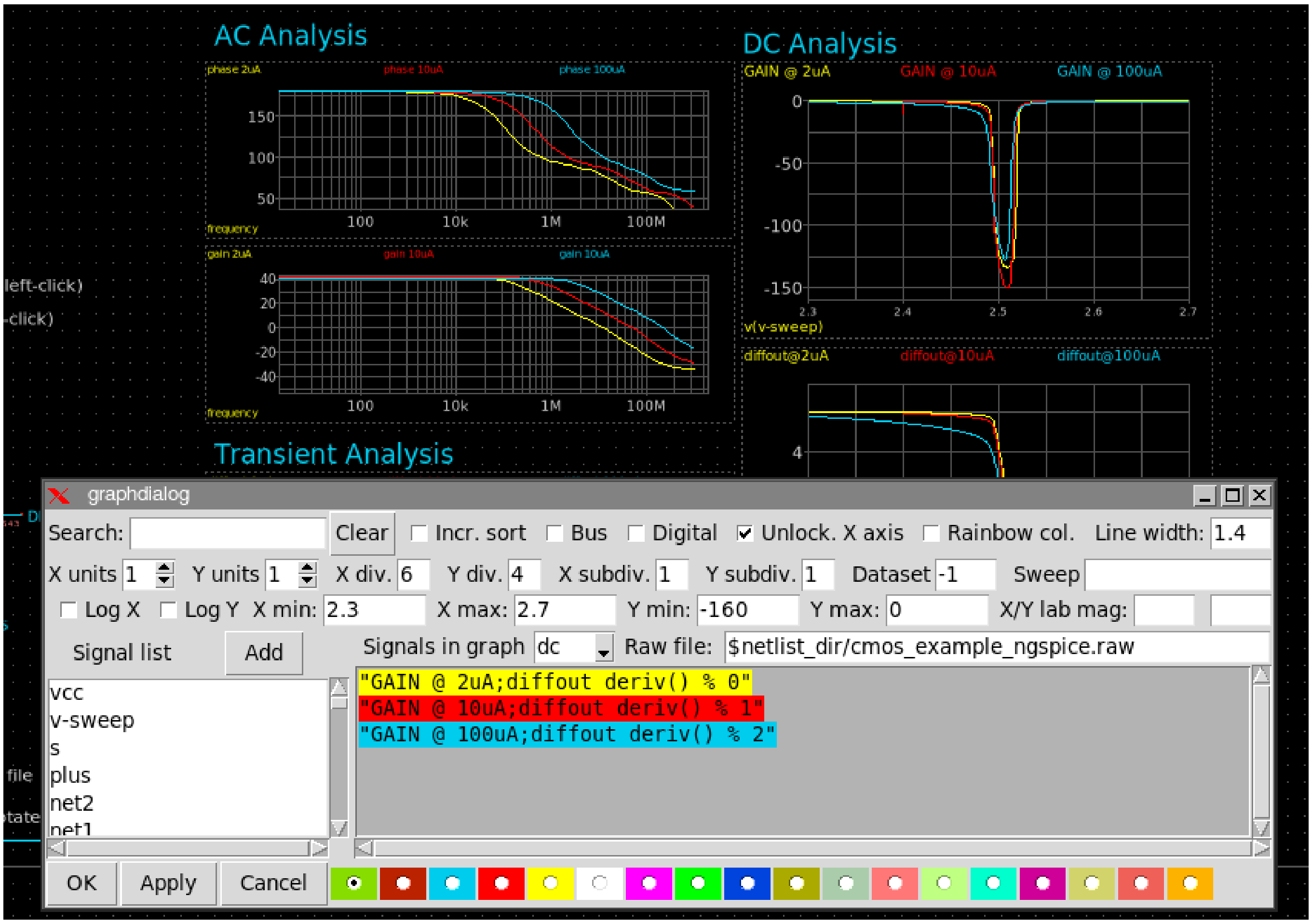
The following syntax: **`node%n`** where **`node`** is a saved node or a bus or an expression and **`n`** is an integer number will plot only the indicated dataset number. Dataset numbers start from 0. This syntax is accepted for single nodes, bus names and expressions:

```
"DATA_4; en, cal, saout % 4"
saout%3
"Power dataset 6; I(VVCC) VCC * %6"
```



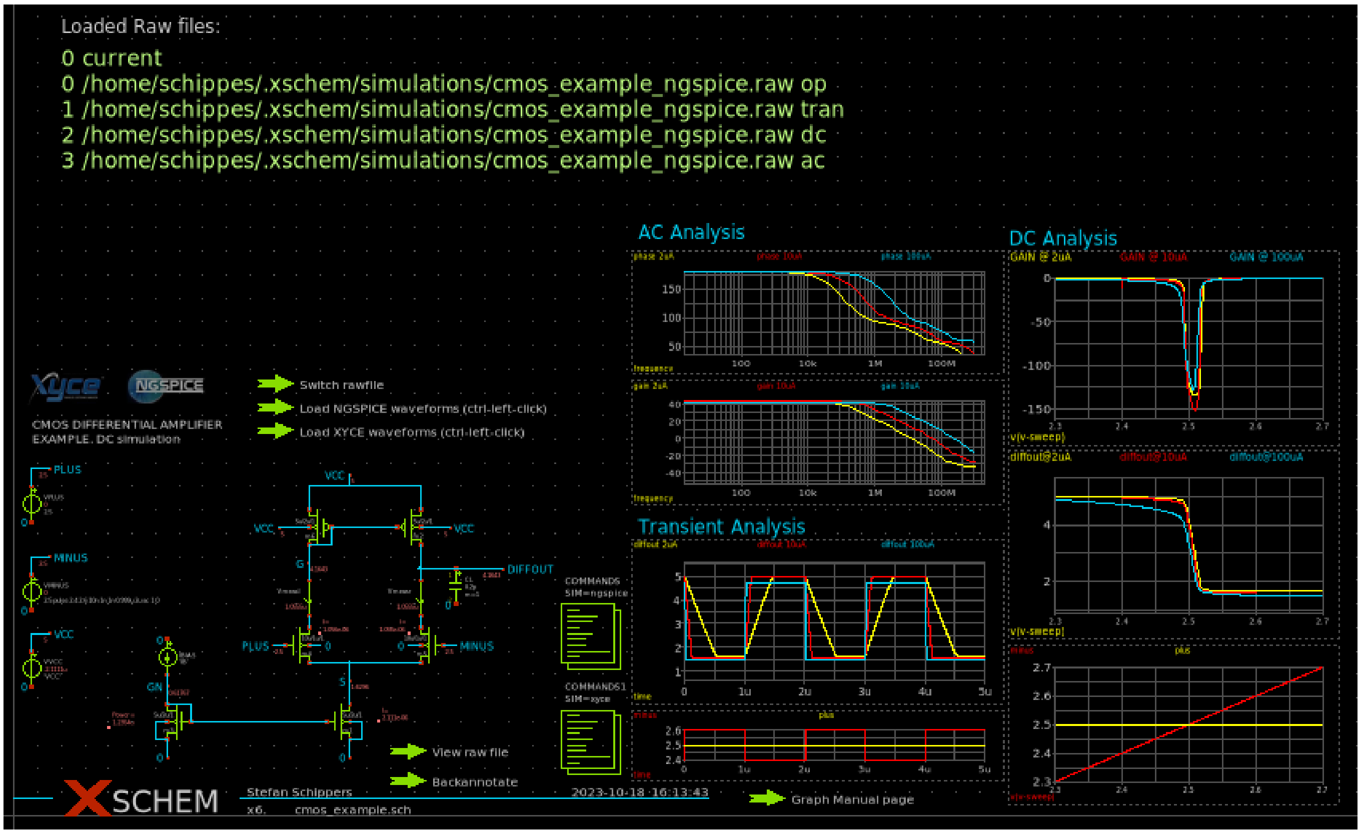
Specify a different raw file in a graph

It is now possible (xschem 3.4.5+) to load more raw files for a schematic in xschem.
Each graph may optionally refer to a different raw file. If you double click on a graph while waveforms are loaded you see the graphdialog dialog box:



```
"GAIN @ 2uA;diffout deriv() % 0"
"GAIN @ 10uA;diffout deriv() % 1"
"GAIN @ 100uA;diffout deriv() % 2"
```

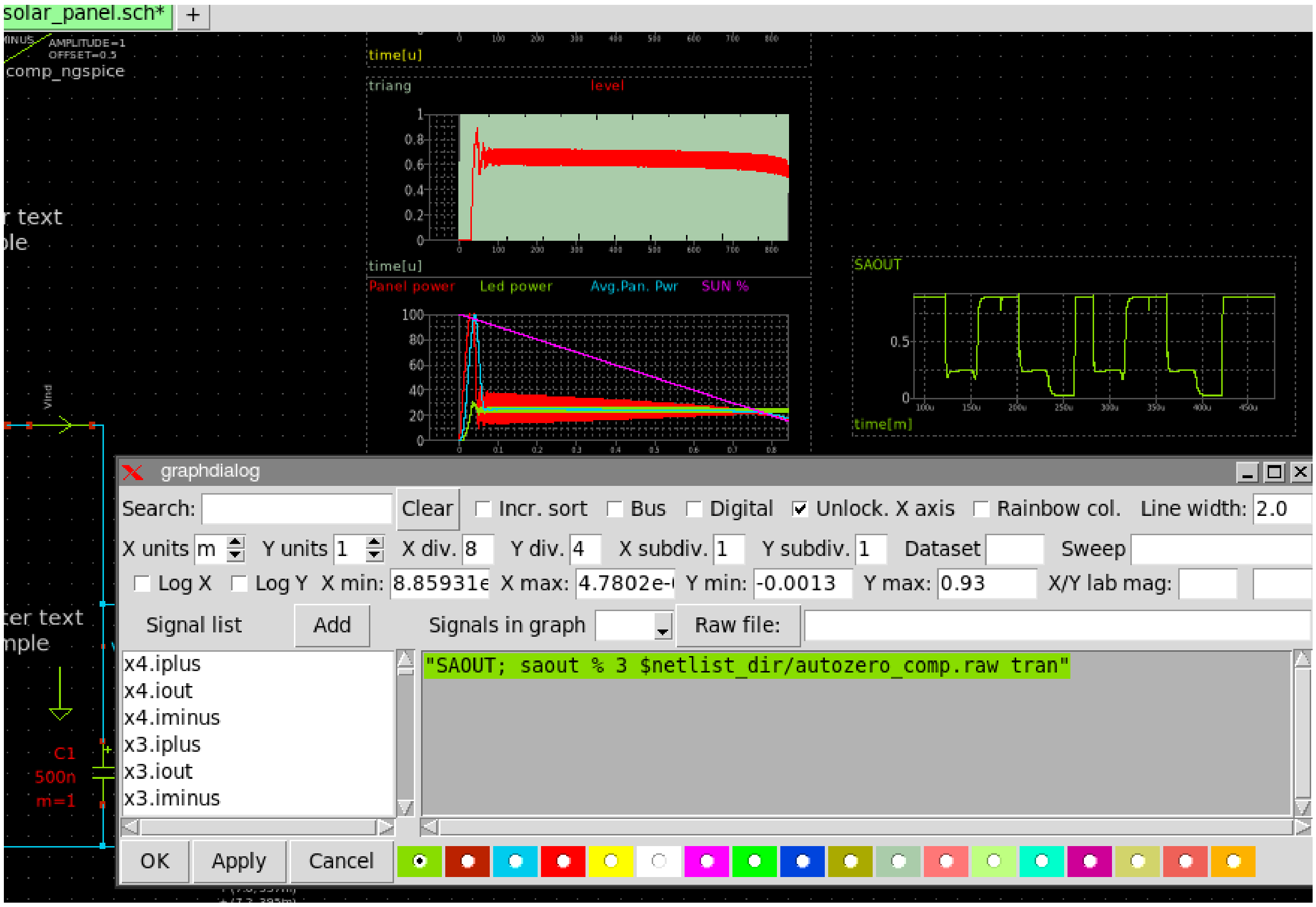
The dialog box has now a simulation type listbox and a **Raw file:** text entry box. Specifying an existing .raw file name and a simulation type from the listbox will override the 'base' raw file loaded in xschem. (the 'base' raw file is the only one that was loaded previously and applied to all graphs) If no raw file is specified in a graph it will use the loaded 'base' raw file if any, like it used to work before. It is also possible to have multiple graphs all referring to the same raw file, each graph showing a different simulation. It is for example possible to show a dc, tran, ac simulation all loaded from a common .raw file. Image below shows an example: DC, AC, Transient simulation each one done with 3 runs varying the Bias current. In addition the **xschem annotate_op** is also used to annotate the operating point into the schematic. Ctrl-Left-button-clicking the **Backannotate** launcher will instantly update all graphs with data taken from the updated raw file(s).



```
Loaded Raw files:
0 current
0 /home/schippes/.xschem/simulations/cmos_example_ngspice.raw op
1 /home/schippes/.xschem/simulations/cmos_example_ngspice.raw tran
2 /home/schippes/.xschem/simulations/cmos_example_ngspice.raw dc
3 /home/schippes/.xschem/simulations/cmos_example_ngspice.raw ac
```

Specify a different raw file for a single signal in a graph

The general syntax for a signal in a graph is the following:
 `"alias_name; signal_name % dataset# raw_file sim_type`
where:
 `dataset#` is the dataset index to display (only meaningful and needed if multiple datasets are present like in Montecarlo / Mismatch simulations). If empty or -1 then show all datasets.
 `raw_file` is the location and name of the raw file to load. You can use **`$netlist_dir`** to quickly reference the simulation directory where usually such raw files are located.
 `sim_type` is the simulation type, like **`ac`**, **`sp`**, **`spectrum`**, **`dc`**, **`op`**, **`tran`**, **`noise`**.
Example:
 `"SAOUT; saout % 2 $netlist_dir/autozero_comp.raw tran`



Search: Clear ☐ Incr. sort ☐ Bus ☐ Digital ☑ Unlock. X axis ☐ Rainbow col. Line width: 2.0

X units m Y units 1 X div. 8 Y div. 4 X subdiv. 1 Y subdiv. 1 Dataset Sweep

☐ Log X ☐ Log Y X min: 8.85931e X max: 4.7802e- Y min: -0.0013 Y max: 0.93 X/Y lab mag:

Signal list Add Signals in graph Raw file:

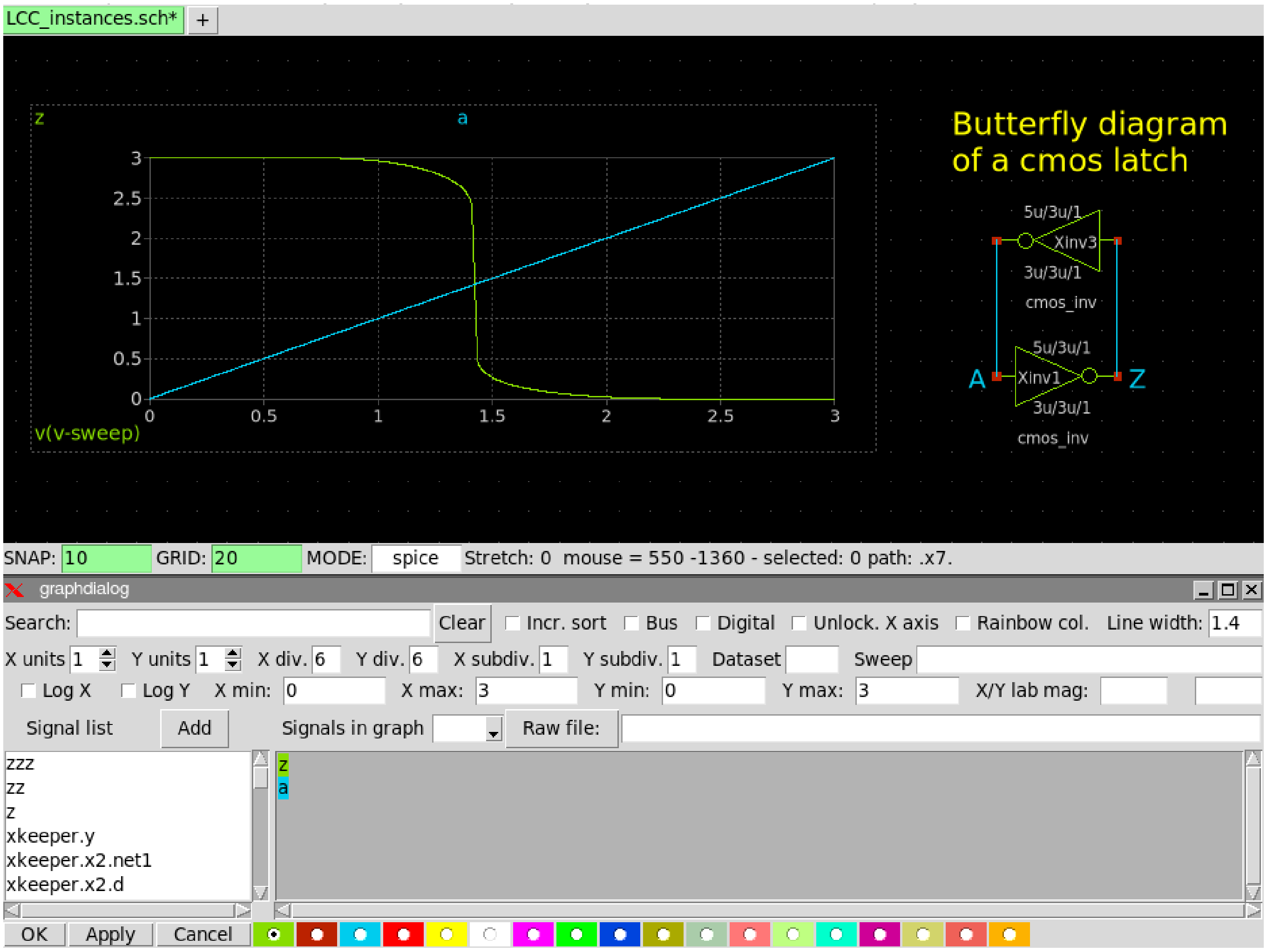
```
x4.iplus
x4.iout
x4.iminus
x3.iplus
x3.iout
x3.iminus
```

```
"SAOUT; saout % 3 $netlist_dir/autozero_comp.raw tran"
```

OK Apply Cancel

Change sweep variable

The graph dialog box has a **Sweep** textbox where you can write the X-axis variable. By default xschem uses the first variable in the raw file for the X-axis, and this is the sweep variable the simulation was done, so time for transients, frequency for AC sims, voltage or current sweep for DC sims. Example below shows a cmos latch where a DC simulation has been done sweeping the voltage generator on the **a** input from 0 to 3V.



SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 550 -1360 - selected: 0 path: .x7.

Search: [ ] Incr. sort [ ] Bus [ ] Digital [ ] Unlock. X axis [ ] Rainbow col. Line width: 1.4

X units 1 Y units 1 X div. 6 Y div. 6 X subdiv. 1 Y subdiv. 1 Dataset Sweep

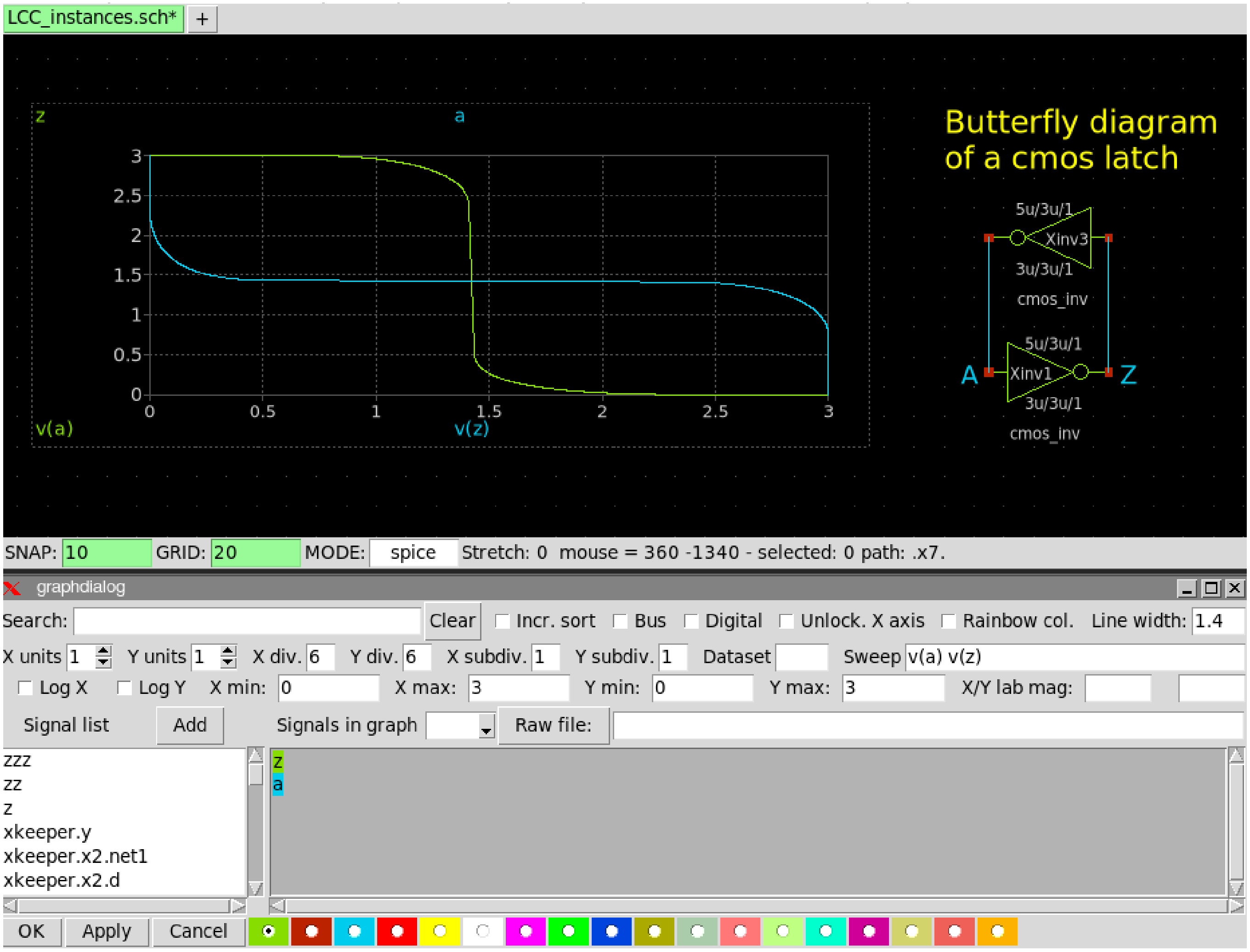
[ ] Log X [ ] Log Y X min: 0 X max: 3 Y min: 0 Y max: 3 X/Y lab mag:

Signal list Signals in graph Raw file:

```
zzz
zz
z
xkeeper.y
xkeeper.x2.net1
xkeeper.x2.d
```

```
z
a
```

If **v(a) v(z)** is specified in the Sweep textbox (or **a z**) the **z** signal will be plotted vs **a** and the **a** signal will be plotted vs **z**.



SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 360 -1340 - selected: 0 path: .x7.

Search: Incr. sort Bus Digital Unlock. X axis Rainbow col. Line width: 1.4

X units 1 Y units 1 X div. 6 Y div. 6 X subdiv. 1 Y subdiv. 1 Dataset Sweep v(a) v(z)

Log X Log Y X min: 0 X max: 3 Y min: 0 Y max: 3 X/Y lab mag:

Signal list

Signals in graph

```
zzz
zz
z
xkeeper.y
xkeeper.x2.net1
xkeeper.x2.d
```

```
z
a
```

DEVELOPER INFO

GENERAL INFORMATION

XSCHEM uses layers for its graphics, each layer is a logical entity defining graphic attributes like color and fill style. There are very few graphical primitive objects:

1. Lines
2. Rectangles
3. Open / close Polygons
4. Arcs / Circles
5. Text

These primitive objects can be drawn on any layer. XSCHEM number of layers can be defined at compile time, however there are some predefined layers (from 0 to 5) that have specific functions:

0. Background color
1. Wire color (nets)
2. Selection color / grid
3. Text color
4. Symbol drawing color
5. Pin color
6. General purpose
7. General purpose
8. General purpose

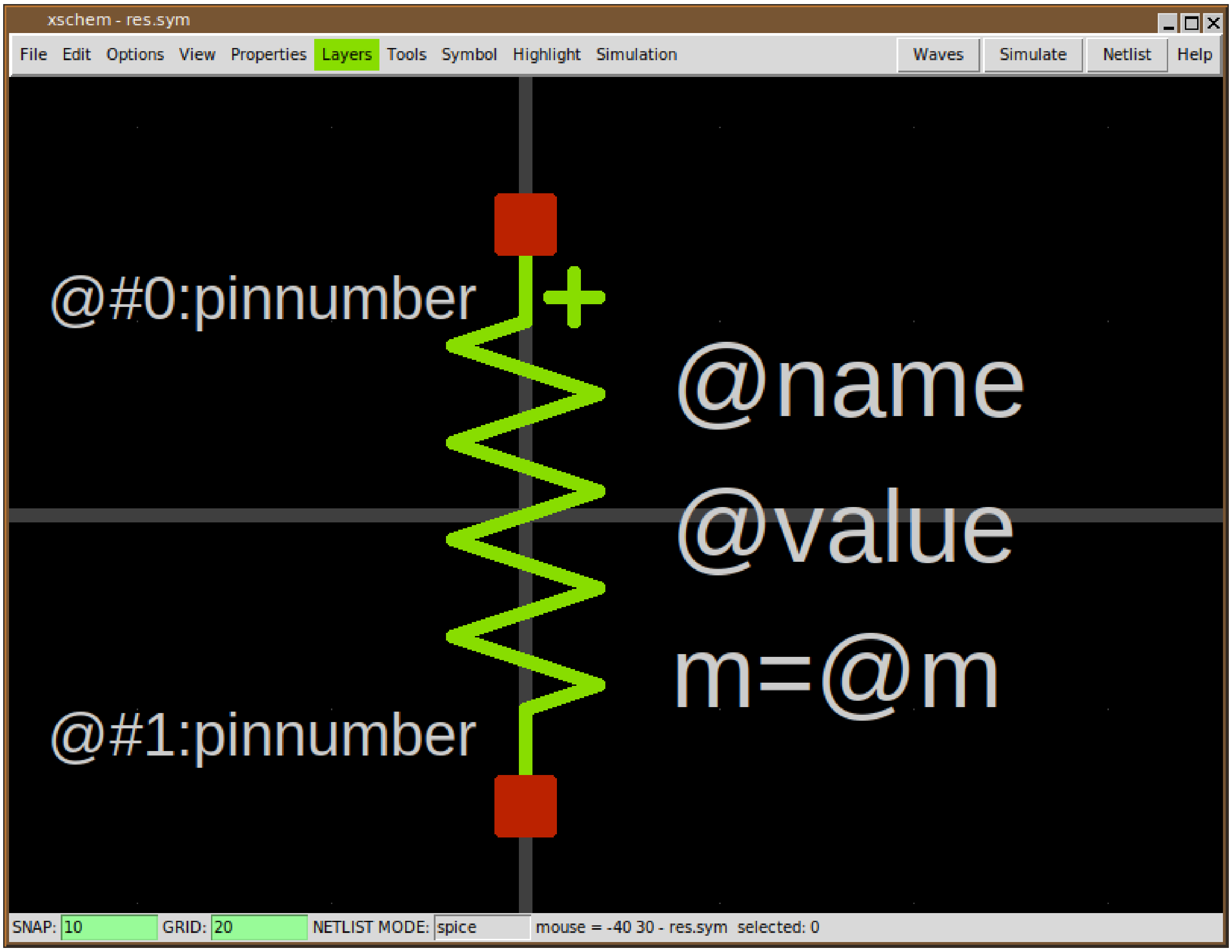
....

20. General purpose
21. General purpose

Although any layer can be used for drawing it is strongly advisable to avoid the background color and the selection color to avoid confusion. Drawing begins by painting the background (layer 0), then drawing the grid (layer 1) then drawing wires (nets) on layer 2, then all graphical objects (lines, rectangles, polygons) starting form layer 0 to the last defined layer.

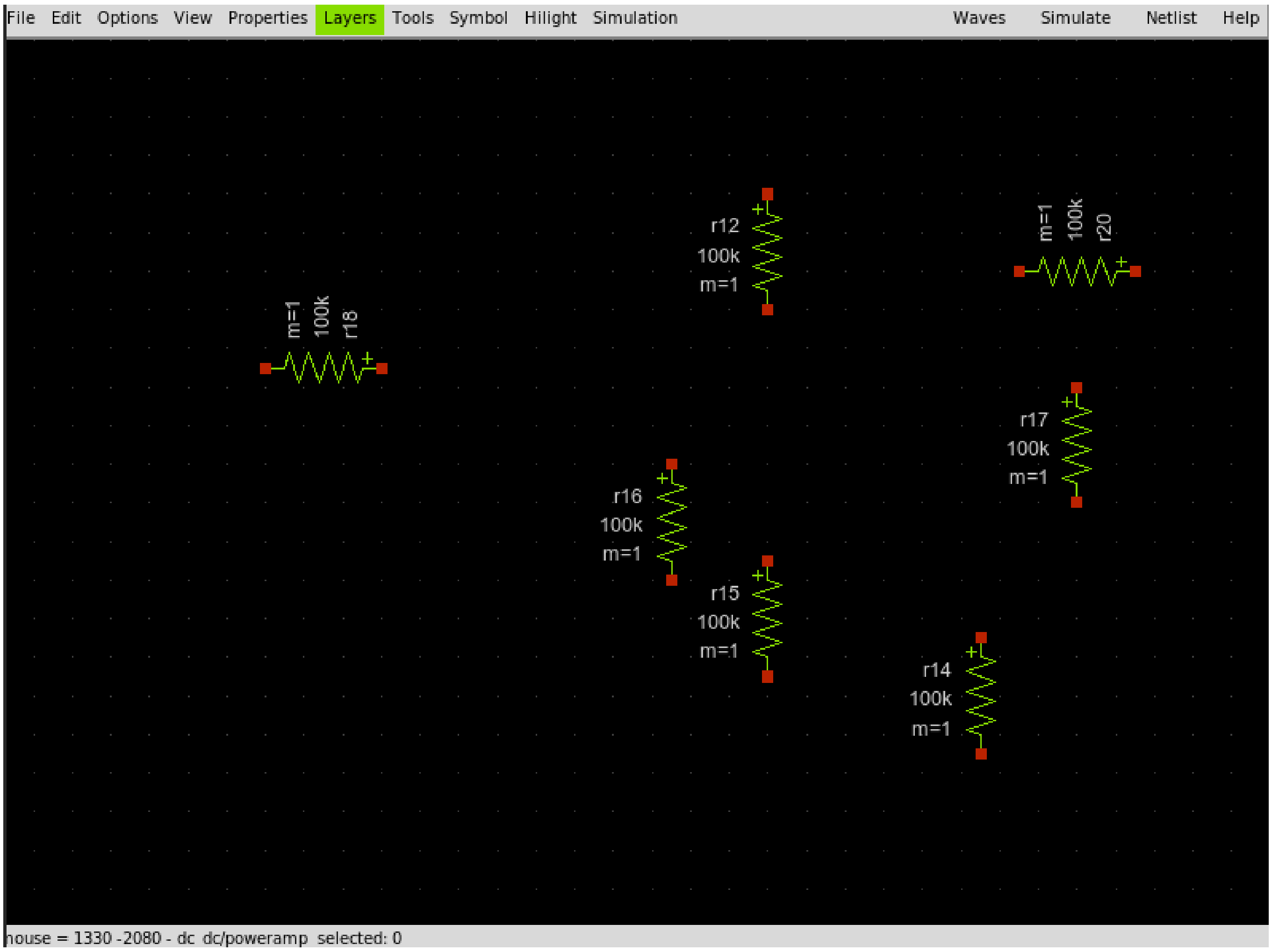
SYMBOLS

There is a primitive object called symbol. Symbols are just a group of primitive graphic objects (lines, polygons, rectangles, text) that can be shown as a single atomic entity. Once created a symbol can be placed in a schematic. The instantiation of a symbol is called 'component'.


xschem - res.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@#0:pinnumber
@name
@value
m=@m
@#1:pinnumber

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -40 30 - res.sym selected: 0

The above picture shows a resistor symbol, built drawing some lines on layer 4 (green), some pins on layer 5 (red) and some text. Symbols once created are stored in libraries (library is just a UNIX directory known to XSCHEM) and can be placed like just any other primitive object multiple times in a schematic window with different orientations.



WIRES

Another special primitive object in XSCHEM is 'Wire', Graphically it is drawn as a line on layer 1 (wires). Wires are drawn only on this layer, they are treated differently by XSCHEM since they carry electrical information. Electrical connection between components is done by drawing a connecting wire.

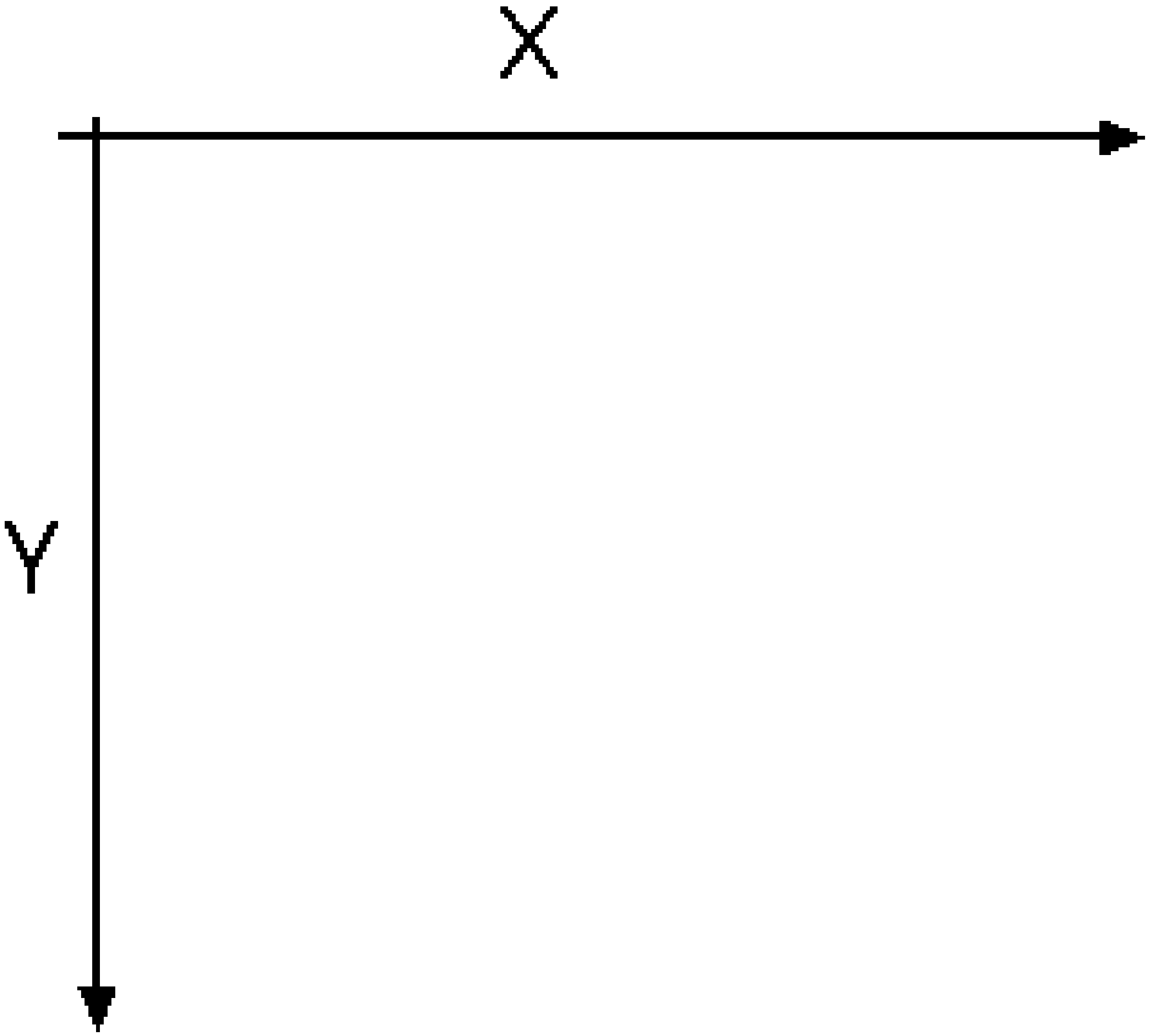
Since wires are used to build the circuit connectivity it is best to avoid drawing lines on layer 1 to avoid confusion, since they would appear like wires, but ignored completely for electrical connectivity.

PROPERTIES

All XSCHEM objects (wires, lines, rectangles, polygons, text, symbol instance aka component) have a property string attached. Any text can be present in a property string, however in most cases the property string is organized as a set of **`key=value`** pairs separated by white space. In addition to object properties the schematic or symbol view has global properties attached. There is one global property defined per netlisting mode (currently SPICE, VHDL, Verilog, tEDAx) and one additional global property for symbols (containing the netlisting rules usually). See the XSCHEM properties section of the manual for more info.

COORDINATE SYSTEM

XSCHEM coordinates are stored as double precision floating point numbers, axis orientation is the same as Xorg default coordinate orientation:



When drawing objects in XSCHEM coordinates are snapped to a multiple of 10.0 coordinate units, so all drawn objects are easily aligned. The snap level can be changed to any value by the user to allow drawing small objects if desired. Grid points are shown at multiples of 20.0 coordinate units, by default.

XSCHEM FILE FORMAT SPECIFICATION

XSCHEM schematics and symbols are stored in .sch and .sym files respectively. The two file formats are identical, with the exception that symbol (.sym) files usually do not contain wires and component instantiations (although they can).

every schematic/symbol object has a corresponding record in the file. A single character at the beginning of a line, separated by white space from subsequent fields marks the type of object:

- **`v`** : XSCHEM Version string

- **S** : Global property associated to the .sch file for SPICE netlisting
- **V** : Global property associated to the .sch file for VERILOG netlisting
- **G** : Global property associated to the .sch file for VHDL netlisting OR Global property associated to the .sym file for netlisting (in 1,2 file format **K** is used, although backward compatibility is guaranteed)
- **E** : Global property associated to the .sch file for tEDAx netlisting
- **K** : Global property associated to the .sch/sym file for netlisting.
  For schematic it is used if instantiated as a component (file format 1.2 and newer)
- **L** : Line
- **B** : Rectangle
- **P** : Open / Closed polygon
- **A** : Arc / Circle
- **T** : Text
- **N** : Wire, used to connect together components (only in .sch files)
- **C** : Component instance in a schematic (only in .sch files)
- **[** : Start of a symbol embedding, the symbol refers to the immediately preceding component instance. This tag must immediately follow a component instance (**C**). See the example here under. A component symbol is embedded into the schematic file when saving if the **embed=true** attribute is set on one of the component instances. Only one copy of the embedded symbol is saved into the schematic and all components referring to this symbol will use the embedded definition. When a component has an embedded symbol definition immediately following, a **embed=true** is added to the component property string if not already present.

```
C {TECHLIB/PCH} 620 -810 0 0 {name=x5 model=PCHLV w=4 l=0.09  m=1 embed=true}
[
v {xschem version=2.9.7 file_version=1.2}
G {}
K {type=pmos
format="@name @pinlist @model w=@w l=@l m=@m"
verilog_format="@verilog_gate #(@del ) @name ( @@d , @@s , @@g );"
template=" name=x1 verilog_gate=pmos del=50,50,50 model=PCH w=0.68 l=0.07 m=1 "
generic_type="model=string"
}
V {}
S {}
E {}
L 4 5 20 20 20 {}
L 4 20 20 20 30 {}
L 4 5 -20 20 -20 {}
L 4 20 -30 20 -20 {}
L 4 -20 0 -10 0 {}
L 4 5 -27.5 5 27.5 {11}
L 4 5 -5 10 0 {}
L 4 5 5 10 0 {}
L 4 10 0 20 0 {}
L 18 -2.5 -15 -2.5 15 {}
B 5 17.5 27.5 22.5 32.5 {name=d dir=inout}
B 5 -22.5 -2.5 -17.5 2.5 {name=g dir=in}
B 5 17.5 -32.5 22.5 -27.5 {name=s dir=inout}
B 5 17.5 -2.5 22.5 2.5 {name=b dir=in}
A 4 -6.25 0 3.75 270 360 {}
T {@w/@l*@m} 7.5 -17.5 0 0 0.2 0.2 {}
T {@name} 7.5 6.25 0 0 0.2 0.2 {999}
T {@model} 2.5 -27.5 0 1 0.2 0.2 {layer=8}
T {D} 25 17.5 0 0 0.15 0.15 {layer=13}
T {NF=@nf} -5 -15 0 1 0.15 0.15 {}
]
```

- **]** : End of an embedded symbol.

the object tag in column 1 is followed by space separated fields that completely define the corresponding object.

VERSION STRING

Example:
v {xschem version=2.9.7 file_version=1.2}

Two attributes are defined, the xschem version and the file format version. Current file format version is 1.2. This string is guaranteed to be the first one in XSCHEM .sch and .sym files. A comment can be added (by manually editing the xschem schematic or symbol file) as shown below:

```
v {xschem version=3.1.0 file_version=1.2
* Copyright 2022 Stefan Frederik Schippers
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     https://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
}
```

GLOBAL SCHEMATIC/SYMBOL PROPERTIES

Example:
K {type=regulator
format="x@name @pinlist r@symname"
verilog_format="assign @#2 = @#0 ;"
tedax_format="footprint @name @footprint
device @name @symname"
template="name=U1 footprint=TO220"}

Global properties define a property string bound to the parent schematic/symbol file, there is one global property record per netlisting mode, currently SPICE, VHDL, Verilog, tEDAx.
In addition (only in file_format 1.2 and newer) for schematics and symbols there is a global attribute ('K') that defines how to netlist the schematic/symbol if placed as a symbol into another parent schematic (should be set in the same way as the 'G' global attribute for symbols in pre-1.2 file format). Normally only 'G' ('K' in 1.2 file format) type property strings are used for symbols and define attributes telling netlisters what to do with the symbol, while global property strings in schematic files corresponding to the active netlisting mode of XSCHEM are copied verbatim to the netlist.
the object tag (S, V, G, E, K) is followed by the property string enclosed in curly braces (**{...}**). This allows strings to contain any white space and newlines. Curly braces if present in the string are automatically escaped with the '\' character by XSCHEM when saving data.
Example of the 4 property string records for a schematic file:
G {}
V {assign #1500 LDOUT = LDIN +1;
}

```
E {}
S {}
```

in this case only the verilog-related global property has some definition. This is Verilog code that is copied into the output netlist.

Attribute strings for all Xschem objects are enclosed in curly braces. This allows attributes to span multiple lines. This component instance:

```
 C {capa.sym} 890 -160 0 0 {name=C4 m=1 value=10u device="tantalium capacitor"}
```

and this one:

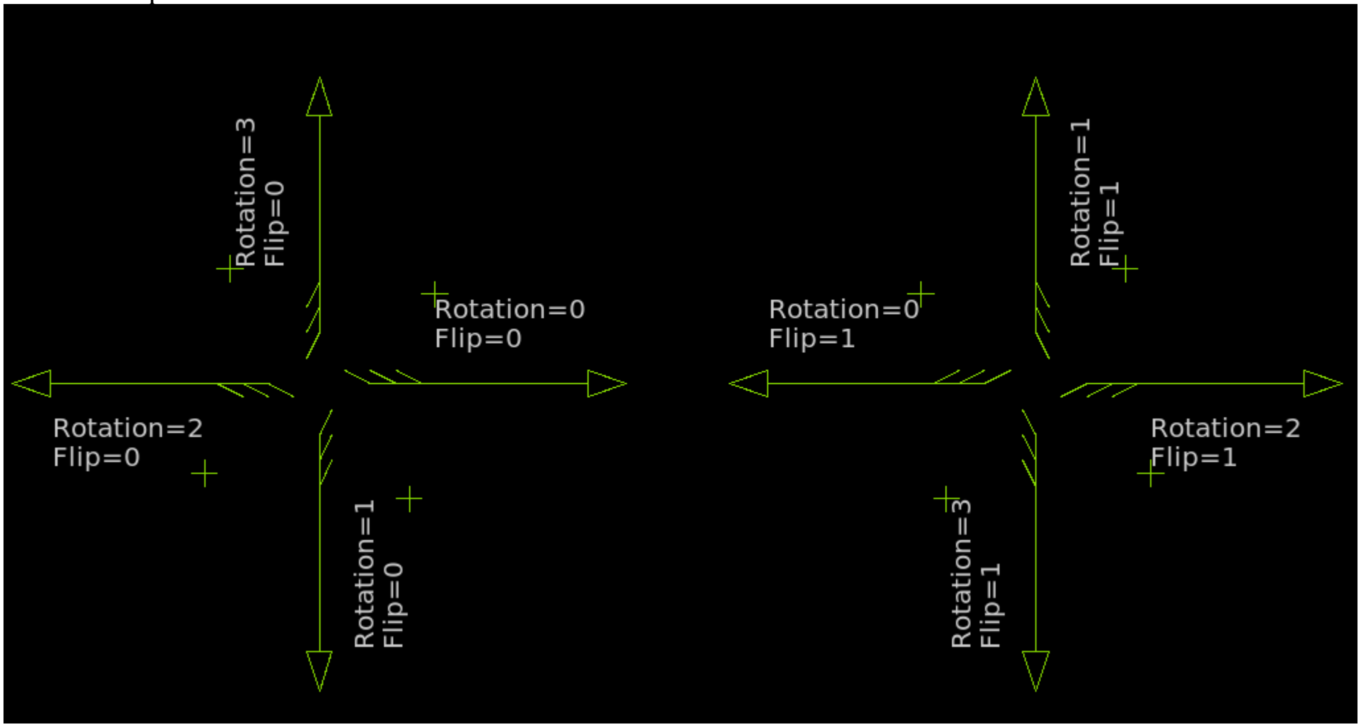
```
 C {capa.sym} 890 -160 0 0 {name=C4
m=1 value=10u
device="tantalium capacitor"
}
```

are perfectly equivalent.

TEXT OBJECT

Example: `T {3 of 4 NANDS of a 74ls00} 500 -580 0 0 0.4 0.4 {font=Monospace layer=4}`
This line defines a text object, the first field after the type tag is the displayed text, followed by X and Y coordinates,rotation, mirror, horizontal and vertical text size and finally a property string defining some text attributes.

- The displayed text is enclosed in curly braces (`{...}`) to allow white space. Literal curly braces must be escaped if present in the saved string. XSCHEM will automatically add the escapes where needed on save.
- X ad Y coordinates are saved and retrieved as double precision floating point numbers.
- Rotation and mirror are integers (range [0:3], [0:1] respectively) that define the orientation of text objects. Using rotation and mirror text can be aligned to any corner of its bounding box, so there are 4 different alignments for vertical text and 4 different alignments for horizontal text. Below picture shows how text is displayed with respect to its anchor point.



- text X and Y sizes are stored as floating point numbers.
- Finally a property string is available to attach attributes to the text object. Currently the following attributes are predefined for text objects:

- **`font`** Name of font to be used (ex: font=Arial)
- **`layer`** Number of layer to use for drawing (as in Xschem Layers menu)
- **`hcenter`** If set to **`true`** horizontal center text
- **`vcenter`** If set to **`true`** vertical center text
- **`weight`** If set to **`bold`** use bold style
- **`slant`** If set to **`italic`** or **`oblique`** use that style for text
- **`hide`** If set to **`true`** text will be hidden unless **`View->Show hidden texts`** is enabled
  If **`hide=instance`** is given the text will be invisible in placed instances of the symbol, but visible when descending into the symbol.

WIRE OBJECT

Example: **`N 890 -130 890 -110 {lab=ANALOG_GND}`**
Format: **`N x1 y1 x2 y2 {attributes}`**
The net 'N' tag is followed by the end point coordinates x1,y1 - x2,y2. (stored and read as double precision numbers) and a property string, used in this case to name the net. In most cases you don't need to specify attributes for nets (one exception is the **`bus`** attribute) as the **`lab`** attribute is set by xschem when creating a netlist or more generally when building the connectivity. This means that almost always nets in a xschem schematic are set as in following example:

```
N 890 -130 890 -110 {}
```

Xschem schematic files store only geometrical data and attributes of the graphic primitives, the connectivity and the logical network is obtained by xschem.

LINE OBJECT

Example: **`L 4 -50 20 50 20 {This is a line on layer 4}`**
Format: **`L layer x1 y1 x2 y2 {attributes}`**
The line 'L' tag is followed by an integer specifying the graphic layer followed by the x1,y1 - x2,y2 coordinates of the line and a property string.

RECTANGLE OBJECT

Example: **`B 5 -62.5 -2.5 -57.5 2.5 {name=IN dir=in pinnumber=1}`**
Format: **`B layer x1 y1 x2 y2 {attributes}`**
The 'Box' 'B' tag is followed by an integer specifying the graphic layer followed by the x1,y1 - x2,y2 coordinates of the rectangle and a final property string. This example defines a symbol pin.
A **`fill=true`** attribute may be given get a patterned fill (this is the default for rectangles).
A **`fill=false`** attribute may be given to avoid a fill pattern.
A **`fill=full`** attribute may be given to get a full solid fill.
Example: **`B 4 100 -300 400 100 {fill=false}`**

OPEN / CLOSED POLYGON OBJECT

Example: **`P 3 5 2450 -210 2460 -170 2500 -170 2510 -210 2450 -210 {}`**
Format: **`P layer npoints px1 py1 px2 py2 .... {attributes}`**
the Polygon 'P' tag is followed by an integer specifying the layer number, followed by the number of points (integer), the x,y coordinates of the polygon points and the property string (empty in this example). If the last point is coincident to the first point a closed polygon is drawn. A **`fill=true`** attribute may be given to fill a closed polygon, in this case a polygon line looks like:

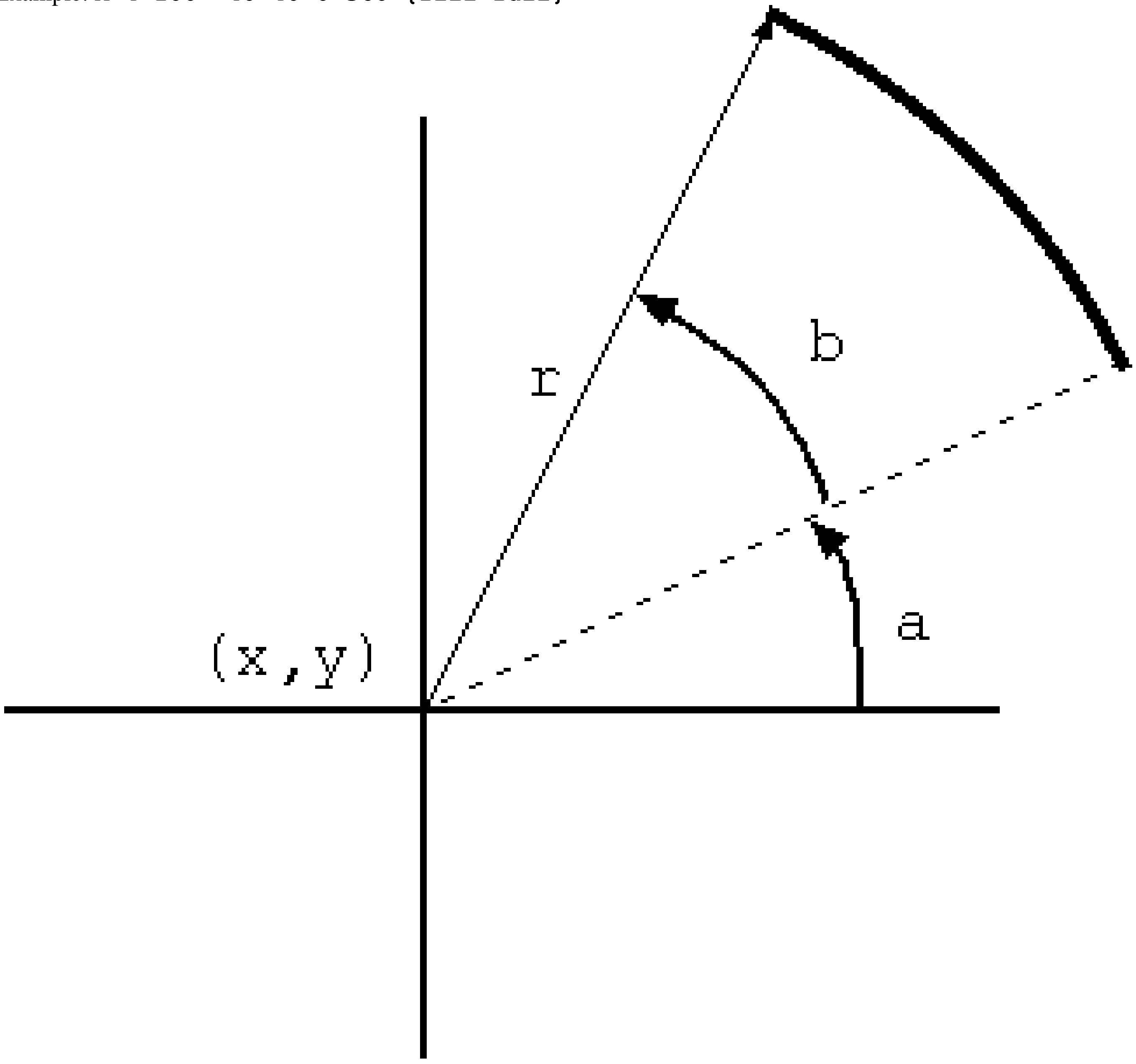
```
P 3 5 2450 -210 2460 -170 2500 -170 2510 -210 2450 -210 {fill=true}
```

A **`fill=full`** attribute will paint the polygon with a solid full color (instead of a patterned fill).

A **bezier=true** attribute will transform the polygon into a bezier curve. See the editor commands page on polygons.

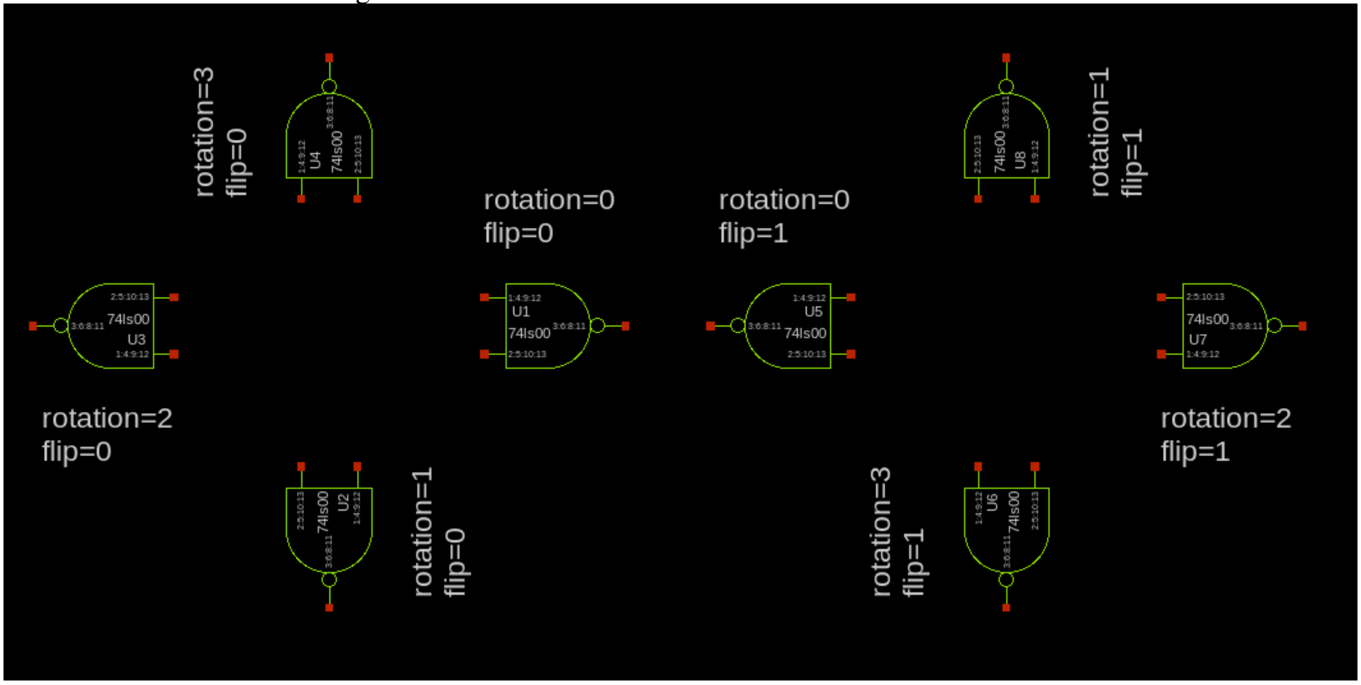
ARC OBJECT

Example: **`A 3 450 -210 120 45 225 {}`**
Format: **`A x y r a b {attributes}`**
The Arc 'A' tag is followed by an integer specifying the layer number, followed by the arc x, y center coordinates, the arc radius, the start angle (measured counterclockwise from the three o'clock direction), the arc sweep angle (measured counterclockwise from the start angle) and the property string (empty in this example). Angles are measured in degrees.
Arcs can be filled or not:
A **`fill=true`** attribute may be given get a patterned fill.
A **`fill=false`** attribute may be given to avoid a fill pattern. This is the default
A **`fill=full`** attribute may be given to get a full solid fill.
Circles are just arcs with a sweep angle of 360 degrees.
Example: **`A 4 100 -40 40 0 360 {fill=full}`**



COMPONENT INSTANCE

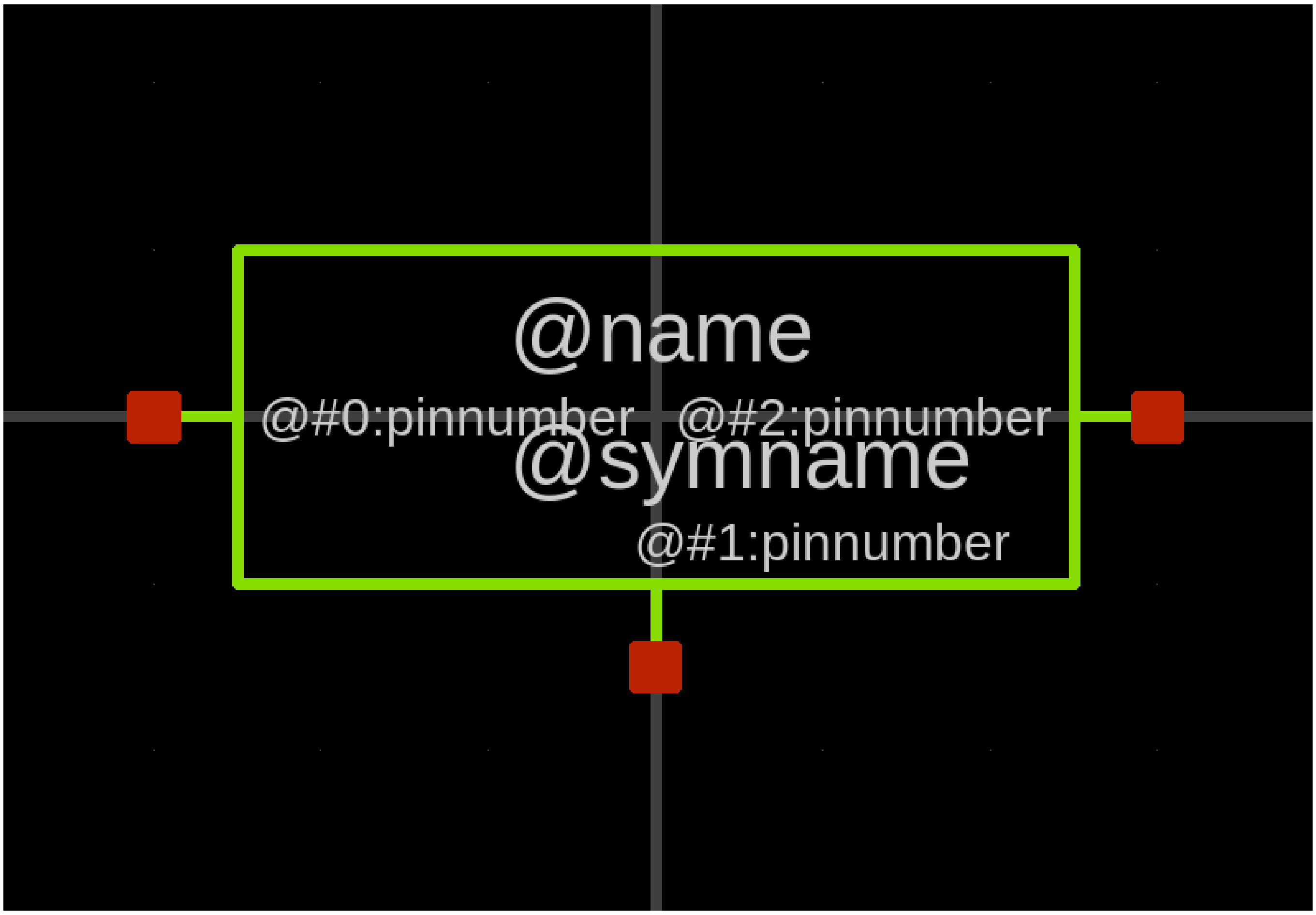
Example: **`C {capa.sym} 890 -160 0 0 {name=C4 m=1 value=10u device="tantalium capacitor"}`**
Format: **`C {<symbol reference>} <X coord> <Y coord> <rotation> <flip> {<attributes>}`**
The component instance tag C is followed by a string specifying **`library/symbol`** or only **`symbol`** (see This tutorial about symbol references) followed by the x,y coordinates, rotation (integer range [0:3]), mirror (integer range [0:1]), and a property string defining various attributes including the mandatory **`name=...`** attribute.
Orientation and mirror meanings are as follows:



EXAMPLE OF A COMPLETE SYMBOL FILE (7805.sym)

```
G {}
K {type=regulator
format="x@name @pinlist r@symname"
verilog_format="assign @#2 = @#0 ;"
tedax_format="footprint @name @footprint
device @name @symname"
template="name=U1 footprint=TO220"}
V {}
S {}
E {}
L 4 -60 0 -50 0 {}
L 4 50 0 60 0 {}
L 4 -50 -20 50 -20 {}
L 4 50 -20 50 20 {}
L 4 -50 20 50 20 {}
L 4 -50 -20 -50 20 {}
L 4 0 20 0 30 {}
B 5 -62.5 -2.5 -57.5 2.5 {name=IN dir=in pinnumber=1}
B 5 -2.5 27.5 2.5 32.5 {name=GND dir=inout pinnumber=2}
B 5 57.5 -2.5 62.5 2.5 {name=OUT dir=out pinnumber=3}
T {@name} -17.5 -15 0 0 0.2 0.2 {}
T {@symname} -17.5 0 0 0 0.2 0.2 {}
```

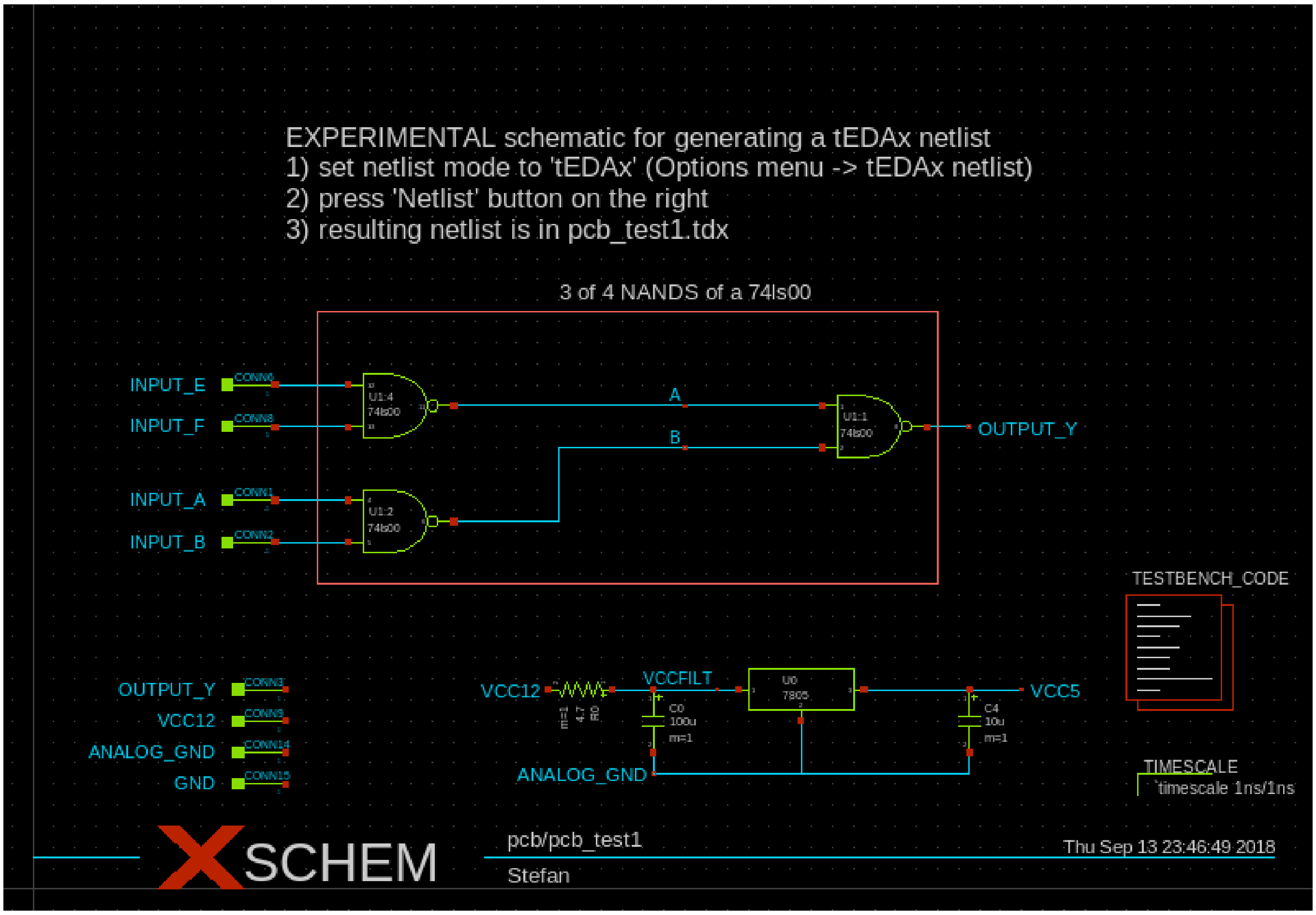
```
T {@#0:pinnumber} -47.5 -2.5 0 0 0.12 0.12 {}
T {@#1:pinnumber} -2.5 12.5 0 0 0.12 0.12 {}
T {@#2:pinnumber} 47.5 -2.5 0 1 0.12 0.12 {}
```



EXAMPLE OF A COMPLETE SCHEMATIC FILE (pcb_test1.sch)

```
G {}
K {}
V {}
S {}
E {}
B 20 270 -550 860 -290 {}
T {3 of 4 NANDS of a 74ls00} 500 -580 0 0 0.4 0.4 {}
T {EXPERIMENTAL schematic for generating a tEDAx netlist
1) set netlist mode to 'tEDAx' (Options menu -> tEDAx netlist)
2) press 'Netlist' button on the right
3) resulting netlist is in pcb_test1.tdx } 240 -730 0 0 0.5 0.5 {}
N 230 -330 300 -330 {lab=INPUT_B}
N 230 -370 300 -370 {lab=INPUT_A}
N 680 -420 750 -420 {lab=B}
N 680 -460 750 -460 {lab=A}
N 400 -350 440 -350 {lab=B}
N 850 -440 890 -440 {lab=OUTPUT_Y}
N 230 -440 300 -440 {lab=INPUT_F}
```

```
N 230 -480 300 -480 {lab=INPUT_E}
N 400 -460 440 -460 {lab=A}
N 550 -190 670 -190 {lab=VCCFILT}
N 590 -130 590 -110 {lab=ANALOG_GND}
N 790 -190 940 -190 {lab=VCC5}
N 890 -130 890 -110 {lab=ANALOG_GND}
N 730 -110 890 -110 {lab=ANALOG_GND}
N 730 -160 730 -110 {lab=ANALOG_GND}
N 590 -110 730 -110 {lab=ANALOG_GND}
N 440 -460 680 -460 {lab=A}
N 500 -420 680 -420 {lab=B}
N 500 -420 500 -350 {lab=B}
N 440 -350 500 -350 {lab=B}
C {title.sym} 160 -30 0 0 {name=l2 author="Stefan"}
C {74ls00.sym} 340 -350 0 0 {name=U1:2  risedel=100 falldel=200}
C {74ls00.sym} 790 -440 0 0 {name=U1:1  risedel=100 falldel=200}
C {lab_pin.sym} 890 -440 0 1 {name=p0 lab=OUTPUT_Y}
C {capa.sym} 590 -160 0 0 {name=C0 m=1 value=100u device="electrolitic capacitor"}
C {74ls00.sym} 340 -460 0 0 {name=U1:4 risedel=100 falldel=200 power=VCC5
url="http://www.engrcs.com/components/74LS00.pdf".sym}
C {LM7805.pdf"}
C {lab_pin.sym} 490 -190 0 0 {name=p20 lab=VCC12}
C {lab_pin.sym} 940 -190 0 1 {name=p22 lab=VCC5}
C {lab_pin.sym} 590 -110 0 0 {name=p23 lab=ANALOG_GND}
C {capa.sym} 890 -160 0 0 {name=C4 m=1 value=10u device="tantalium capacitor"}
C {res.sym} 520 -190 1 0 {name=R0 m=1 value=4.7 device="carbon resistor"}
C {lab_wire.sym} 620 -460 0 0 {name=l3 lab=A}
C {lab_wire.sym} 620 -420 0 0 {name=l0 lab=B}
C {lab_wire.sym} 650 -190 0 0 {name=l1 lab=VCCFILT}
C {connector.sym} 230 -370 0 0 {name=CONN1 lab=INPUT_A verilog_type=reg}
C {connector.sym} 230 -330 0 0 {name=CONN2 lab=INPUT_B verilog_type=reg}
C {connector.sym} 240 -190 0 0 { name=CONN3 lab=OUTPUT_Y }
C {connector.sym} 230 -480 0 0 {name=CONN6 lab=INPUT_E verilog_type=reg}
C {connector.sym} 230 -440 0 0 {name=CONN8 lab=INPUT_F verilog_type=reg}
C {connector.sym} 240 -160 0 0 { name=CONN9 lab=VCC12 }
C {connector.sym} 240 -130 0 0 { name=CONN14 lab=ANALOG_GND  verilog_type=reg}
C {connector.sym} 240 -100 0 0 { name=CONN15 lab=GND  verilog_type=reg}
C {code.sym} 1030 -280 0 0 {name=TESTBENCH_CODE only_toplevel=false value="initial begin
  $dumpfile(\\"dumpfile.vcd\\");
  $dumpvars;
  INPUT_E=0;
  INPUT_F=0;
  INPUT_A=0;
  INPUT_B=0;
  ANALOG_GND=0;
  #10000;
  INPUT_A=1;
  INPUT_B=1;
  #10000;
  INPUT_E=1;
  INPUT_F=1;
  #10000;
  INPUT_F=0;
  #10000;
  INPUT_B=0;
  #10000;
  $finish;
end

assign VCC12=1;

"}
C {verilog_timescale.sym} 1050 -100 0 0 {name=s1 timestep="1ns" precision="1ns" }
```


EXPERIMENTAL schematic for generating a tEDAx netlist
1) set netlist mode to 'tEDAx' (Options menu -> tEDAx netlist)
2) press 'Netlist' button on the right
3) resulting netlist is in pcb_test1.tdx
3 of 4 NANDS of a 74ls00
INPUT_E
INPUT_F
INPUT_A
INPUT_B
OUTPUT_Y
VCC12
ANALOG_GND
GND
VCCFILT
VCC5
TESTBENCH_CODE
TIMESCALE
`timescale 1ns/1ns
pcb/pcb_test1
Stefan
Thu Sep 13 23:46:49 2018
XSCHEM

XSCHEM COMMAND REFERENCE DOCUMENTATION

The following are xschem specific tcl commands. All commands are prefixed by the **xschem** keyword. Example:

```
xschem getprop instance x3 OFFSET
```

- **abort_operation**

```
Resets UI state, unselect all and abort any pending operation
```

- **add_symbol_pin [x y name dir [draw]]**

```
place a symbol pin.
x,y : pin coordinates
name = pin name
dir = in|out|inout
draw: 1 | 0 (draw or not the added pin immediately, default = 1)
if no parameters given start a GUI placement of a symbol pin
```

- **add_graph**

```
Start a GUI placement of a graph object
```

- **add_image**

```
Ask user to choose a png/jpg file and start a GUI placement of the image
```

- **align**

```
Align currently selected objects to current snap setting
```

- **annotate_op [raw_file] [level]**

```
Annotate operating point data into current schematic.
use <schematic name>.raw or use supplied argument as raw file to open
look for operating point data and annotate voltages/currents into schematic.
The optional 'level' integer specifies the hierarchy level the raw file refers to.
This is necessary if annotate_op is called from a sub schematic at a hierarchy
level > 0 but simulation was done at top level (hierarchy 0, for example)
```

- **arc**

```
Start a GUI placement of an arc.
User should click 3 unaligned points to define the arc
```

- **attach_labels**

```
Attach net labels to selected component(s) instance(s)
```

- **bbox begin|end**

```
Start/end bounding box calculation: parameter is either 'begin' or 'end'
```

- **break_wires [remove]**

```
Break wires at selected instance pins
if '1' is given as 'remove' parameter broken wires that are
all inside selected instances will be deleted
```

- **build_colors**

```
Rebuild color palette using values of tcl vars dim_value and dim_bg
```

- **callback winpath event mx my key button aux state**

```
Invoke the callback event dispatcher with a software event
```

- **case_insensitive 1|0**

```
Set case insensitive symbol lookup. Use only on case insensitive filesystems
```

- **change_elem_order n**

```
set selected object (instance, wire, line, rect, ...) to
position 'n' in its respective array
```

- **change_sch_path n <draw>**

```
if descended into a vector instance change inst number we are into to 'n',
(same rules as 'descend' command) without going up and descending again
if 'draw' string is given redraw screen
```

- **check_symbols**

```
List all used symbols in current schematic and warn if some symbol is newer
```

- **check_unique_names [1|0]**

```
Check if all instances have a unique refdes (name attribute in xschem),
highlight such instances. If second parameter is '1' rename duplicates
```

- **closest_object**

```
returns index of closest object to mouse coordinates
index = type layer n
type = wire | text | line | poly | rect | arc | inst
layer is the layer number the object is drawn with
(valid for line, poly, rect, arc)
n is the index of the object in the xschem array
example:
   $  after 3000 {set obj [xschem closest_object]}
(after 3s)
   $ puts $obj
   line 4 19
```

- **circle**

```
Start a GUI placement of a circle.
User should click 3 unaligned points to define the circle
```

- **clear [force] [symbol|schematic]**

```
Clear current schematic window. Resets hierarchy level. Remove symbols
the 'force' parameter will not ask to save existing modified schematic.
the 'schematic' or 'symbol' parameter specifies to default to a schematic
or symbol window (default: schematic)
```

- **clear_drawing**

```
Clears drawing but does not purge symbols
```

- **color_dim value**

```
Dim colors or brite colors depending on value parameter: -5 <= value <= 5
```

- **compare_schematics [sch_file]**

```
Compare currently loaded schematic with another 'sch_file' schematic.
if no file is given prompt user to choose one
```

- **connected_nets [0|1|2|3]**

```
Select nets/labels  connected to currently selected instance
```

```
if '1' argument is given, stop at wire junctions
if '2' argument is given select only wires directly
attached to selected instance/net
if '3' argument is given combine '1' and '2'
```

- **copy**

```
Copy selection to clipboard
```

- **copy_hierarchy to from**

```
Copy hierarchy info from tab/window "from" to tab/window "to"
Example: xschem copy_hierarchy .drw .x1.drw
```

- **copy_objects [deltax deltay [rot flip]]**

```
if deltax and deltay (and optionally rot and flip) are given copy selection
to specified offset, otherwise start a GUI copy operation
```

- **count_items string separator quoting_chars**

```
Debug command
```

- **create_plot_cmd**

```
Create an xplot file in netlist/simulation directory with
the list of highlighted nodes in a format the selected waveform
viewer understands (bespice, gaw, ngspice)
```

- **cursor n e**

```
enable or disable cursors.
cursor will be set at 0.0 position. use 'xschem set cursor[12]_x' to set position
n: cursor number (1 or 2, for a or b)
e: enable flag: 1: show, 0: hide
```

- **cut**

```
Cut selection to clipboard
```

- **debug n**

```
Set xschem in debug mode.'n' is the debug level
(0=no debug). Higher levels yield more debug info.
```

- **delete**

```
Delete selection
```

- **delete_files**

```
Bring up a file selector the user can use to delete files
```

- **descend [n] [notitle]**

```
Descend into selected component instance. Optional number 'n' specifies the
```

```
 instance number to descend into for vector instances (default: 0).
 0 or 1: leftmost instance, 2: second leftmost instance, ...
-1: rightmost instance,-2: second rightmost instance, ...
if integer 'notitle' is given pass it to descend_schematic()
```

- **descend_symbol**

```
Descend into the symbol view of selected component instance
```

- **destroy_all [force]**

```
Close all additional windows/tabs. If 'force' is given do not ask for
confirmation for changed schematics
Returns the remaining # of windows/tabs in addition to main window/tab
```

- **display_hilights [nets|instances]**

```
Print a list of highlighted objects (nets, net labels/pins, instances)
if 'instances' is specified list only instance highlights
if 'nets' is specified list only net highlights
```

- **draw_graph [n] [flags]**

```
Redraw graph rectangle number 'n'.
If the optional 'flags' integer is given it will be used as the
flags bitmask to use while drawing (can be used to restrict what to redraw)
```

- **drc_check [i]**

```
Perform DRC rulecheck of instances.
if i is specified do check of specified instance
otherwise check all instances in current schematic.
```

- **edit_file**

```
Edit xschem file of current schematic if nothing is selected.
Edit .sym file if a component is selected.
```

- **edit_prop**

```
Edit global schematic/symbol attributes or attributes
of currently selected instances
```

- **edit_vi_prop**

```
Edit global schematic/symbol attributes or
attributes of currently selected instances
using a text editor (defined in tcl 'editor' variable)
```

- **embed_rawfile raw_file**

```
Embed base 64 encoded 'raw_file' into currently
selected element as a 'spice_data'
attribute.
```

- **enable_layers**

```
Enable/disable layers depending on tcl array variable enable_layer()
```

- **escape_chars source [charset]**

```
escape tcl special characters with backslash
if charset is given escape characters in charset
```

- **exit [exit_code] [closewindow] [force]**

```
Exit the program, ask for confirm if current file modified.
if exit_code is given exit with its value, otherwise use 0 exit code
if 'closewindow' is given close the window, otherwise leave with a blank schematic
when closing the last remaining window
if 'force' is given do not ask before closing modified schematic windows/tabs
This command returns the list of remaining open windows in addition to main window
```

- **expandlabel lab**

```
Expand vectored labels/instance names:
xschem expandlabel {2*A[3:0]} --> A[3],A[2],A[1],A[0],A[3],A[2],A[1],A[0] 8
last field is the number of bits
since [ and ] are TCL special characters argument must be quoted with { and }
```

- **fill_reset [nodraw]**

```
After setting tcl array pixdata(n) reset fill patterns on all layers
If 'nodraw' is given do not redraw window.
```

- **fill_type n fill_type [nodraw]**

```
Set fill type for layer 'n', fill_type may be 'solid' or 'stipple' or 'empty'
If 'nodraw' is given do not redraw window.
```

- **find_nth string sep quote keep_quote n**

```
Find n-th field string separated by characters in sep. 1st field is in position 1
do not split quoted fields (if quote characters are given) and return unquoted.
xschem find_nth {aaa,bbb,ccc,ddd} {,} 2  --> bbb
xschem find_nth {aaa, "bbb, ccc" , ddd} { ,} {"} 2  --> bbb, ccc
```

- **flip [x0 y0]**

```
Flip selection horizontally around point x0 y0.
if x0, y0 not given use mouse coordinates
```

- **flip_in_place**

```
Flip selection horizontally, each object around its center
```

- **flipv [x0 y0]**

```
Flip selection vertically around point x0 y0.
if x0, y0 not given use mouse coordinates
```

- **flipv_in_place**

```
Flip selection vertically, each object around its center
```

- **`floaters_from_selected_inst`**

```
flatten to current level selected instance texts
```

- **`fullscreen`**

```
Toggle fullscreen modes: fullscreen with menu & status, fullscreen, normal
```

- **`get var`**

```
Get C variable/constant 'var'
```

  - **`backlayer`** number of background layer
  - **`bbox`** bounding box schematic
  - **`bbox_hilighted`** bounding box of highlinhted objects
  - **`bbox_selected`** bounding box of selected objects
  - **`cadlayers`** number of layers
  - **`case_insensitive`** case_insensitive symbol matching
  - **`color_ps`** color postscript flag
  - **`constr_mv`** color postscript flag
  - **`current_dirname`** directory name of current design
  - **`current_name`** name of current design (no library path)
  - **`current_win_path`** path of current tab/window (.drw, .x1.drw, ...)
  - **`currsch`** hierarchy level of current schematic (start at 0)
  - **`cursor1_x`** cursor 1 position
  - **`cursor2_x`** cursor 2 position
  - **`debug_var`** debug level (0 = no debug, 1, 2, 3,...)
  - **`draw_window`** direct draw into window
  - **`first_sel`** get data about first selected object
  - **`fix_broken_tiled_fill`** get drawing method setting (for broken GPUs)
  - **`fix_mouse_coord`** get fix_mouse_coord setting (fix for broken RDP)
  - **`format`** alternate format attribute to use in netlist (or NULL)
  - **`graph_lastsel`** number of last graph that was clicked
  - **`gridlayer`** layer number for grid
  - **`help`** command help
  - **`header_text`** header metadata (license info etc) present in schematic
  - **`infowindow_text`** ERC messages
  - **`instances`** number of instances in schematic
  - **`intuitive_interface`** ERC messages
  - **`last_created_window`** return win_path of last created tab or window
  - **`lastsel`** number of selected objects
  - **`line_width`** get line width
  - **`lines`** (xschem get lines n) number of lines on layer 'n'
  - **`netlist_name`** netlist name if set. If 'fallback' given get default name
  - **`netlist_type`** get current netlist type (spice/vhdl/verilog/tedax)
  - **`no_draw`** disable drawing
  - **`ntabs`** get number of additional tabs (0 = only one tab)
  - **`pinlayer`** layer number for pins
  - **`raw_level`** hierarchy level where raw file was loaded
  - **`rectcolor`** current layer number
  - **`rects`** (xschem get rects n) number of rectangles on layer 'n'
  - **`sellayer`** layer number for selection

    - **`semaphore`** used for debug
    - **`schname`** get full path of current sch. if 'n' given get sch of level 'n'
    - **`schprop`** get schematic "spice" global attributes
    - **`schvhdlprop`** get schematic "vhdl" global attributes
    - **`schverilogprop`** get schematic "verilog" global attributes
    - **`schsymbolprop`** get schematic "symbol" global attributes
    - **`schtedaxprop`** get schematic "tedax" global attributes
    - **`sch_path`** get hierarchy path. if 'n' given get hierpath of level 'n'
    - **`sch_to_compare`** if set return schematic current design is compared with
    - **`symbols`** number of loaded symbols
    - **`temp_dir`** get windows temporary dir
    - **`text_svg`** return 1 if using <text> elements in svg export
    - **`textlayer`** layer number for texts
    - **`top_path`** get top hier path of current window (always "" for tabbed if)
    - **`topwindow`** same as top_path but main window returned as "."
    - **`version`** return xschem version
    - **`wirelayer`** layer used for wires
    - **`wires`** number of wires
    - **`xschem_web_dirname`**
    - **`xorigin`** x coordinate of origin
    - **`yorigin`** y coordinate of origin
    - **`zoom`** zoom level
- **`get_additional_symbols what`**

```
create new symbols for instance based implementation selection
```

- **`get_cell cell n_dirs`**

```
return result of get_cell function
```

- **`get_cell_w_ext cell n_dirs`**

```
return result of get_cell_w_ext function
```

- **`getprop instance|instance_pin|symbol|text ref`**

```
    getprop instance inst
Get the full attribute string of 'inst'

    getprop instance inst attr
Get the value of attribute 'attr'
If 'attr has the form 'cell::sym_attr' look up attribute 'sym_attr'
of the symbol referenced by the instance.

    getprop instance_notcl inst attr
Same as above but do not perform tcl substitution

    getprop instance_pin inst pin
Get the full attribute string of pin 'pin' of instance 'inst'
Example: xschem getprop instance_pin x3 MINUS --> name=MINUS dir=in

    getprop instance_pin inst pin pin_attr
Get attribute 'pin_attr' of pin 'pin' of instance 'inst'
Example: xschem getprop instance_pin x3 MINUS dir --> in
```

```
    getprop symbol sym_name
Get full attribute string of symbol 'sym_name'
example:
xschem getprop symbol comp_ngspice.sym -->
  type=subcircuit
  format="@name @pinlist @symname
     OFFSET=@OFFSET AMPLITUDE=@AMPLITUDE GAIN=@GAIN ROUT=@ROUT COUT=@COUT"
  template="name=x1 OFFSET=0 AMPLITUDE=5 GAIN=100 ROUT=1000 COUT=1p"

    getprop symbol sym_name sym_attr [with_quotes]
Get value of attribute 'sym_attr' of symbol 'sym_name'
'with_quotes' (default:0) is an integer passed to get_tok_value()

    getprop rect layer num attr [with_quotes]
if '1' is given as 'keep' return backslashes and unescaped quotes if present in value
Get attribute 'attr' of rectangle number 'num' on layer 'layer'

    getprop text num attr
Get attribute 'attr' of text number 'num'
if attribute is 'txt_ptr' return the text

    getprop wire num attr
Get attribute 'attr' of wire number 'num'

    ('inst' can be an instance name or instance number)
    ('pin' can be a pin name or pin number)
```

- **get_sch_from_sym inst [symbol]**

```
get schematic associated with instance 'inst'
if inst==-1 and a 'symbol' name is given get sch associated with symbol
```

- **get_tok str tok [with_quotes]**

```
get value of token 'tok' in string 'str'
'with_quotes' (default:0) is an integer passed to get_tok_value()
```

- **get_tok_size**

```
Get length of last looked up attribute name (not its value)
if returned value is 0 it means that last searched attribute did not exist
```

- **globals**

```
Return various global variables used in the program
```

- **go_back [notitle]**

```
Go up one level (pop) in hierarchy
if integer 'notitle' given pass it to the go_back() function (1=do not update window titl
```

- **grabscreen**

```
grab root window
```

- **hash_file file [skip_path_lines]**

```
Do a simple hash of 'file'
'skip_path_lines' is an integer (default: 0) passed to hash_file()
```

- **hash_string str**

```
          Do a simple hashing of string 'str'
```

- **help**

```
         Print command help
```

- **hier_psprint [file]**

```
          Hierarchical postscript / pdf print
          if 'file' is not given show a fileselector dialog box
```

- **hilight [drill]**

```
          Highlight selected element/pins/labels/nets
          if 'drill' is given propagate net highlights through conducting elements
          (elements that have the 'propag' attribute on pins )
```

- **hilight_instname inst [fast]**

```
          Highlight instance 'inst'
              if 'fast' is specified do not redraw
          'inst' can be an instance name or number
```

- **hilight_netname net**

```
          Highlight net name 'net'
```

- **image [invert|white_transp|black_transp|transp_white|transp_black|write_back|**

```
               blend_white|blend_black]
          Apply required changes to selected images
          invert: invert colors
          white_transp: transform white color to transparent (alpha=0)
          black_transp: transform black color to transparent (alpha=0)
          transp_white: transform transparent to white color
          transp_black: transform transparent to black color
          blend_white:  blend with white background and remove alpha
          blend_black:  blend with black background and remove alpha
          write_back:   write resulting image back into `image_data` attribute
```

- **instance sym_name x y rot flip [prop] [n]**

```
          Place a new instance of symbol 'sym_name' at position x,y,
          rotation and flip  set to 'rot', 'flip'
          if 'prop' is given it is the new instance attribute
          string (default: symbol template string)
          if 'n' is given it must be 0 on first call
          and non zero on following calls
          It is used only for efficiency reasons if placing multiple instances
```

- **instance_bbox inst**

```
          return instance and symbol bounding boxes
          'inst' can be an instance name or number
```

- **instance_coord [instance]**

```
Return instance name, symbol name, x placement coord, y placement coord, rotation and fli
of selected instances
if 'instance' is given (instance name or number) return data about specified instance
Example:
  xschem [~] xschem instance_coord
  {R5} {res.sym} 260 260 0 0
  {C1} {capa.sym} 150 150 1 1
```

- **instance_list**

```
Return a list of 3-items. Each 3-item is
an instance name followed by the symbol reference and symbol type.
Example: xschem instance_list -->
  {x1} {sky130_tests/bandgap.sym} {subcircuit}} {...} {...} {...} ...
```

- **instance_net inst pin**

```
Return the name of the net attached to pin 'pin' of instance 'inst'
Example: xschem instance_net x3 MINUS --> REF
```

- **instance_nodemap inst [pin]**

```
Return the instance name followed by a list of 'pin net' associations
example:  xschem instance_nodemap x3
--> x3 PLUS LED OUT LEVEL MINUS REF
instance x3 pin PLUS is attached to net LED, pin OUT to net LEVEL and so on...
If 'pin' is given restrict map to only that pin
```

- **instance_number inst [n]**

```
Return the position of instance 'inst' in the instance array
If 'n' is given set indicated instance position to 'n'
```

- **instance_pin_coord inst attr value**

```
Return the name and coordinates of pin with
attribute 'attr' set to 'value' of instance 'inst'
'inst can be an instance name or a number
Example: xschem instance_pin_coord x3 name MINUS --> {MINUS} 600 -840
```

- **instance_pins inst**

```
Return list of pins of instance 'inst'
'inst can be an instance name or a number
```

- **instance_pos inst**

```
Get number (position) of instance name 'inst'
```

- **instances_to_net net**

```
Return list of instances names and pins attached to net 'net'
Example: xschem instances_to_net PANEL
 --> { {Vsw} {plus} {580} {-560} } { {p2} {p} {660} {-440} }
     { {Vpanel1} {minus} {600} {-440} }
```

- **is_symgen symbol**

```
tell if 'symbol' is a generator (symbol(param1,param2,...)
```

- **line [x1 y1 x2 y2] [pos] [propstring] [draw]**

```
if 'x1 y1 x2 y2'is given place line on current
layer (rectcolor) at indicated coordinates.
if 'pos' is given insert at given position in rectangle array.
if 'pos' set to -1 append to last element in line array.
'propstring' is the attribute string. Set to empty if not given.
if 'draw' is set to 1 (default) draw the new object, else don't
If no coordinates are given start a GUI operation of line placement
```

- **line_width n**

```
set line width to floating point number 'n'
```

- **list_hierarchy**

```
List all schematics at or below current hierarchy with modification times.
Example: xschem list_hiearchy
-->
20230302_003134  {/home/.../ngspice/solar_panel.sch}
20230211_010031  {/home/.../ngspice/pv_ngspice.sch}
20221011_175308  {/home/.../ngspice/diode_ngspice.sch}
20221014_091945  {/home/.../ngspice/comp_ngspice.sch}
```

- **list_hilights [sep | all | all_nets | all_inst]**

```
 Sorted list of non port or non top level current level highlight nets,
 separated by character 'sep' (default: space)
 if `all_inst` is given list all instance hilights
 if `all_nets` is given list all net hilights
 if `all` is given list all hash content
```

- **list_nets**

```
 List all nets with type (in / out / inout / net)
```

- **list_tokens str with_quotes**

```
List tokens in string 'str'
with_quotes:
0: eat non escaped quotes (")
1: return unescaped quotes as part of the token value if they are present
2: eat backslashes
```

- **load f [symbol|gui|noundoreset|nofullzoom]**

```
Load a new file 'f'.
'gui': ask to save modified file or warn if opening an already
open file or opening a new(not existing) file.
'noundoreset': do not reset the undo history
'symbol': do not load symbols (used if loading a symbol instead of a schematic)
'nofullzoom': do not do a full zoom on new schematic.
'nodraw': do not draw.
```

- **load_new_window [f]**

```
Load schematic in a new tab/window. If 'f' not given prompt user
if 'f' is given as empty '{}' then open untitled.sch
```

- **log f**

```
If 'f' is given output stderr messages to file 'f'
if 'f' is not given and a file log is open, close log
file and resume logging to stderr
```

- **log_write text**

```
write given string to log file, so tcl can write messages on the log file
```

- **logic_get_net net_name**

```
Get logic state of net named 'net_name'
Returns 0, 1, 2, 3 for logic levels 0, 1, X, Z or nothing if no net found.
```

- **logic_set_net net_name n [num]**

```
set 'net_name' to logic level 'n' 'num' times.
'n':
    0  set to logic value 0
    1  set to logic value 1
    2  set to logic value X
    3  set to logic value Z
   -1  toggle logic valie (1->0, 0->1)
the 'num' parameter is essentially useful only with 'toggle' (-1)  value
```

- **logic_set n [num]**

```
set selected nets, net labels or pins to logic level 'n' 'num' times.
'n':
    0  set to logic value 0
    1  set to logic value 1
    2  set to logic value X
    3  set to logic value Z
   -1  toggle logic valie (1->0, 0->1)
the 'num' parameter is essentially useful only with 'toggle' (-1)  value
```

- **make_sch**

```
Make a schematic from selected symbol
```

- **make_sch_from_sel**

```
Create an LCC instance from selection and place it instead of selection
also ask if a symbol (.sym) file needs to be created
```

- **make_symbol**

```
From current schematic (circuit.sch) create a symbol (circuit.sym)
using ipin.sym, opin.sym, iopin.sym in schematic
to deduce symbol interface pins.
```

- **merge [f]**

```
Merge another file. if 'f' not given prompt user.
```

- **move_instance inst x y rot flip [nodraw] [noundo]**

```
resets instance coordinates, and rotaton/flip. A dash will keep existing value
if 'nodraw' is given do not draw the moved instance
if 'noundo' is given operation is not undoable
```

- **move_objects [dx dy] [kissing] [stretch]**

```
Start a move operation on selection and let user terminate the operation in the GUI
if kissing is given add nets to pins that touch other instances or nets
if stretch is given stretch connected nets to follow instace pins
if dx and dy are given move by that amount.
```

- **my_strtok_r str delim quote keep_quote**

```
    test for my_strtok_r() function
```

- **net_label [type]**

```
Place a new net label
'type': 1: place a 'lab_pin.sym' label
        0: place a 'lab_wire.sym' label
User should complete the placement in the GUI.
```

- **net_pin_mismatch**

```
Highlight nets attached to selected symbols with
a different name than symbol pin
```

- **netlist [-messages] [filename]**

```
do a netlist of current schematic in currently defined netlist format
if 'filename'is given use specified name for the netlist
if 'filename' contains path components place the file in specified path location.
if only a name is given and no path ('/') components are given use the
default netlisting directory.
This means that 'xschem netlist test.spice' and 'xschem netlist ./test.spice'
will create the netlist in different places.
netlisting directory is reset to previous setting after completing this command
If -messages is given return the ERC messages instead of just a fail (1)
or no fail (0) code.
```

- **new_process [f]**

```
Start a new xschem process for a schematic.
If 'f' is given load specified schematic.
```

- **new_schematic create|destroy|destroy_all|switch winpath file [draw]**

```
Open/destroy a new tab or window
  create: create new empty window or with 'file' loaded if 'file' given.
          The winpath must be given (even {} is ok).
          non empty winpath ({1}) will avoid warnings if opening the
          same file multiple times.
  destroy: destroy tab/window identified by winpath. Example:
           xschem new_schematic destroy .x1.drw
  destroy_all: close all tabs/additional windows
           if the 'force'argument is given do not issue a warning if modified
           tabs are about to be closed.
  switch: switch context to specified 'winpath' window or specified schematic name
           If 'draw' is given and set to 0 do not redraw after switching tab
           (only tab i/f)
Main window/tab has winpath set to .drw,
Additional windows/tabs have winpath set to .x1.drw, .x2.drw and so on...
```

- **only_probes**

```
    dim schematic to better show highlights
```

- **origin x y [zoom]**

```
Move origin to 'x, y', optionally changing zoom level to 'zoom'
A dash ('-') given for x or y will keep existing value
```

- **parse_cmd**

```
debug command to test parse_cmd_string()
splits a command string into argv-like arguments
return # of args in *argc
argv[*argc] is always set to NULL
```

- **parselabel str**

```
Debug command to test vector net syntax parser
```

- **paste [x y]**

```
Paste clipboard. If 'x y' not given user should complete placement in the GUI
```

- **pinlist inst [attr]**

```
List all pins of instance 'inst'
if no 'attr' is given return full attribute string,
else return value for attribute 'attr'.
Example: xschem pinlist x3 name
-->  { {0} {PLUS} } { {1} {OUT} } { {2} {MINUS} }
Example: xschem pinlist x3 dir
-->  { {0} {in} } { {1} {out} } { {2} {in} }
Example: xschem pinlist x3
--> { {0} {name=PLUS dir=in } } { {1} {name=OUT dir=out } }
    { {2} {name=MINUS dir=in } }
```

- **place_symbol [sym_name] [prop]**

```
Start a GUI placement operation of specified 'sym_name' symbol.
If 'sym_name' not given prompt user
'prop' is the attribute string of the symbol.
If not given take from symbol template attribute.
```

- **place_text**

```
Start a GUI placement of a text object
```

- **polygon**

```
Start a GUI placement of a polygon
```

- **preview_window create|draw|destroy|close [winpath] [file]**

```
destroy: will delete preview schematic data and destroy container window
close: same as destroy but leave the container window.
Used in fileselector to show a schematic preview.
```

- **print png|svg|ps|pdf|ps_full|pdf_full img_file [img_x img_y] [x1 y1 x2 y2]**

```
If img_x and img_y are set to 0 (recommended for svg and ps/pdf)
they will be calculated by xschem automatically
if img_x and img_y are given they will set the bitmap size, if
area to export is not given then use the selection boundbox if
a selection exists or do a full zoom.
Export current schematic to image.
                         img x   y size    xschem area to export
   0     1    2    3         4   5             6    7   8   9
xschem print png file.png  [400 300]       [ -300 -200 300 200 ]
xschem print svg file.svg  [400 300]       [ -300 -200 300 200 ]
xschem print ps  file.ps   [400 300]       [ -300 -200 300 200 ]
xschem print eps file.eps  [400 300]       [ -300 -200 300 200 ]
xschem print pdf file.pdf  [400 300]       [ -300 -200 300 200 ]
xschem print ps_full  file.ps
xschem print pdf_full file.pdf
```

- **print_hilight_net show**

```
from highlighted nets/pins/labels:
show == 0   ==> create pins from highlight nets
show == 1   ==> show list of highlight net in a dialog box
show == 2   ==> create labels with i prefix from hilight nets
show == 3   ==> show list of highlight net with path and label
               expansion in a dialog box
show == 4   ==> create labels without i prefix from hilight nets
for show = 0, 2, 4 user should complete GUI placement
of created objects
```

- **print_spice_element inst**

```
Print spice raw netlist line for instance (number or name) 'inst'
```

- **propagate_hilights [set clear]**

```
Debug: wrapper to propagate_hilights() function
```

- **push_undo**

```
Push current state on undo stack
```

- **raw what ...**

```
  what = add | clear | datasets | index | info | loaded | list | new | points | rawfile |
         read | set | sim_type | switch | switch_back | table_read | value | values | pos

xschem raw read filename [type [sweep1 sweep2]]
  if sweep1, sweep2 interval is given in 'read' subcommand load only the interval
  sweep1 <= sweep_var < sweep2
  type is the analysis type to load (tran, dc, ac, op, ...). If not given load first four
  raw file.

xschem raw clear [rawfile [type]]
  unload given file and type. If type not given delete all type sfrom rawfile
  if no file is given unload all raw files.

xschem raw del name
  delete named vector from current raw file

xschem raw info
  print information about loaded raw files and show the currently active one.

xschem raw new name type sweepvar start end step
  create a new raw file with sweep variable 'sweepvar' with number=(end - start) / step c
  from start value 'start' and step 'step'

xschem raw list
  get list of saved simulation variables

xschem raw vars
  get number of simulation variables

xschem raw switch [n | rawfile type]
  make the indicated 'rawfile, type' the active one
  else if a number n is specified make the n-th raw data the active one.
  if no file or number is specified then switch to the next rawdata in the list.

xschem switch_back
  switch to previously active rawdata.

xschem raw datasets
  get number of datasets (simulation runs)

xschem raw value node n [dset]
  return n-th value of 'node' in raw file
  dset is the dataset to look into in case of multiple runs (first run = 0).
  if dset = -1 consider n as the absolute position into the whole data file
  (all datasets combined).
  If n is given as empty string {} return value at cursor b,
  dset not used in this case

xschem raw loaded
  return hierarchy level where raw file was loaded or -1 if no raw loaded

xschem raw rawfile
   return raw filename

xschem raw sim_type
   return raw loaded simulation type (ac, op, tran, ...)

xschem raw index node
  get index of simulation variable 'node'.
  Example:  raw index v(led) --> 46

xschem raw values node [dset]
  print all simulation values of 'node' for dataset 'dset' (default dset=0)
  dset= -1: print all values for all datasets

xschem raw pos_at node value [dset] [from_start] [to_end]
  returns the position, starting from 0 or from_start if given, to the end of dataset
  or to_end if given of the first point 'p' where node[p] and node[p+1] bracket value.
  If dset not given assume dset 0 (first one)
  This is usually done on the sweep (time) variable in transient sims where timestep is
  not uniform

xschem raw points [dset]
  print simulation points for dataset 'dset' (default: all dataset points combined)

xschem raw set node n value [dset]
  change loaded raw file data node[n] to value
  dset is the dataset to look into in case of multiple runs (first run = 0)
  dset = -1: consider n as the absolute position in the whole raw file
  (all datasets combined)

xschem raw table_read tablefile
  read a tabular data file.
  First line is the header line containing variable names.
  data is presented in column format after the header line
  First column is sweep (x-axis) variable
  Double empty lines start a new dataset
  Single empty lines are ignored
  Datasets can have different # of lines.
  new dataset do not start with a header row.
  Lines beginning with '#' are comments and ignored

     time    var_a   var_b   var_c
  # this is a comment, ignored
      0.0     0.0     1.8    0.3
    <single empty line: ignored>
      0.1     0.0     1.5    0.6
      ...     ...     ...    ...
    <empty line>
    <Second empty line: start new dataset>
      0.0     0.0     1.8    0.3
      0.1     0.0     1.5    0.6
      ...     ...     ...    ...

xschem raw add varname [expr] [sweep_var]
  add a 'varname' vector with all values set to 0 to loaded raw file if expr not given
  otherwise initialize data with values calculated from expr.
  if expr is given and also sweep_var is given use indicated sweep_var for expressions
  that need it. If sweep_var not given use first raw file variable as sweep variable.
  If varname is already existing and expr given recalculate data
  Example: xschem raw add power {outm outp - i(@r1[i]) *}
```

- **raw_clear**

```
Unload all simulation raw files
You can use xschem raw clear as well.
```

- **raw_read [file] [sim] [sweep1 sweep2]**

```
If a raw file is already loaded delete from memory
then load specified file and analysis 'sim' (dc, ac, tran, op, ...)
If 'sim' not specified load first section found in raw file.
if sweep1, sweep2 interval is given load only the interval
sweep1 <= sweep_var < sweep2
```

- **raw_read_from_attr [sim]**

```
If a simulation raw file is already loaded delete from memory
else read section 'sim' (tran, dc, ac, op, ...)
of base64 encoded data from a 'spice_data'
```

```
attribute of selected instance
If sim not given read first section found
```

- **rebuild_connectivity**

```
Rebuild logical connectivity abstraction of schematic
```

- **rebuild_selection**

```
Rebuild selection list
```

- **record_global_node n node**

```
call the record_global_node function (list of netlist global nodes)
```

- **rect [x1 y1 x2 y2] [pos] [propstring] [draw]**

```
if 'x1 y1 x2 y2'is given place recangle on current
layer (rectcolor) at indicated coordinates.
if 'pos' is given insert at given position in rectangle array.
if 'pos' set to -1 append rectangle to last element in rectangle array.
'propstring' is the attribute string. Set to empty if not given.
if 'draw' is set to 1 (default) draw the new object, else don't
If no coordinates are given start a GUI operation of rectangle placement
```

- **redo**

```
Redo last undone action
```

- **redraw**

```
redraw window
```

- **reload**

```
Forced (be careful!) Reload current schematic from disk
```

- **reload_symbols**

```
Reload all used symbols from disk
```

- **remove_symbols**

```
Internal command: remove all symbol definitions
```

- **replace_symbol inst new_symbol [fast]**

```
Replace 'inst' symbol with 'new_symbol'
If doing multiple substitutions set 'fast' to {}
 on first call and 'fast' on next calls
for faster operation.
do a 'xschem redraw' at end to update screen
Example: xschem replace_symbol R3 capa.sym
```

- **reset_caches**

```
Reset cached instance and symbol cached flags (inst->flags, sym->flags)
```

- **reset_inst_prop inst**

```
Reset instance attribute string taking it from symbol template string
```

- **reset_symbol inst symref**

```
This is a low level command, it merely changes the xctx->inst[...].name field.
It is caller responsibility to delete all symbols before and do a reload_symbols
afterward
```

- **resetwin create_pixmap clear_pixmap force w h**

```
internal command: calls resetwin()
```

- **resolved_net [net]**

```
if 'net' is given  return its topmost full hierarchy name
else returns the topmost full hierarchy name of selected net/pin/label.
Nets connected to I/O ports are mapped to upper level recursively
```

- **rotate [x0 y0]**

```
Rotate selection around point x0 y0.
if x0, y0 not given use mouse coordinates
```

- **rotate_in_place**

```
Rotate selected objects around their 0,0 coordinate point
```

- **save**

```
Save schematic if modified. Does not ask confirmation!
```

- **saveas [file] [type]**

```
save current schematic as 'file'
if file is empty ({}) use current schematic name
as defalt and prompt user with file selector
'type' is used used to set/change file extension:
  schematic: save as schematic (*.sch)
  symbol: save as symbol (*.sym)
  If not specified default to schematic (*.sch)
Does not ask confirmation if file name given
```

- **sch_pinlist**

```
List a 2-item list of all pins  and directions of current schematic
Example: xschem sch_pinlist
-->  {PLUS} {in} {OUT} {out} {MINUS} {in} {VCC} {inout} {VSS} {inout}
```

- **schematic_in_new_window [new_process] [nodraw] [force]**

```
When a symbol is selected edit corresponding schematic
in a new tab/window if not already open.
If nothing selected open another window of the second
```

```
schematic (issues a warning).
if 'new_process' is given start a new xschem process
if 'nodraw' is given do not draw loaded schematic
returns '1' if a new schematic was opened, 0 otherwise
```

- **search regex|exact select tok val [match_case]**

```
Search instances / wires / rects / texts with attribute string containing 'tok'
and value 'val'
search can be exact ('exact') or as a regular expression ('regex')
select:
   0 : highlight matching instances
   1 : select matching instances
  -1 : unselect matching instances
'tok' set as:
    propstring : will search for 'val' in the entire
    *instance* attribute string.
    cell::propstring : will search for 'val' in the entire
    *symbol* attribute string.
    cell::name : will search for 'val' in the symbol name
    cell::<attr> will search for 'val' in symbol attribute 'attr'
      example: xschem search regex 0 cell::template GAIN=100
 match_case:
   1 : Match case
   0 : Do not match case
   If not given assume 1 (Match case)
```

- **select instance|wire|text id [clear] [fast]**

```
    select rect|line|poly|arc layer id [clear] [fast]
    Select indicated instance or wire or text, or
    Select indicated (layer, number) rectangle, line, polygon, arc.
    For 'instance' 'id' can be the instance name or number
    for all other objects 'id' is the position in the respective arrays
    if 'clear' is specified does an unselect operation
    if 'fast' is specified avoid sending information to infowindow and status bar
    returns 1 if something selected, 0 otherwise
```

- **select_all**

```
Selects all objects in schematic
```

- **select_dangling_nets**

```
Select all nets/labels that are dangling, ie not attached to any non pin/port/probe compo
Returns number of selected items (wires,labels) if danglings found, 0 otherwise
```

- **select_hilight_net**

```
Select all highlight objects (wires, labels, pins, instances)
```

- **select_inside x1 y1 x2 y2 [sel]**

```
Select all objects inside the indicated area
if [sel] is set to '0' do an unselect operation
```

- **selected_set [what]**

```
Return a list of selected instance names
```

```
If what is not given or set to 'inst' return list of selected instance names
If what set to 'rect' return list of selected rectangles with their coordinates
If what set to 'text' return list of selected texts with their coordinates
```

- **`selected_wire`**

```
Return list of selected nets
```

- **`send_to_viewer`**

```
Send selected wires/net labels/pins/voltage source or ammeter currents to current
open viewer (gaw or bespice)
```

- **`set var value`**

```
Set C variable 'var' to 'value'
```

  - **`cadgrid`** set cad grid (default: 20)
  - **`cadsnap`** set mouse snap (default: 10)
  - **`color_ps`** set color psoscript (1 or 0)
  - **`constr_mv`** set constrained move (1=horiz, 2=vert, 0=none)
  - **`cursor1_x`** set graph cursor1 position
  - **`cursor2_x`** set graph cursor2 position
  - **`draw_window`** set drawing to window (1 or 0)
  - **`fix_broken_tiled_fill`** alternate drawing method for broken GPUs
  - **`fix_mouse_coord`** fix for wrong mouse coords in RDP software
  - **`format`** set name of custom format attribute used for netlisting
  - **`header_text`** set header metadata (used for license info)
  - **`hide_symbols`** set to 0,1,2 for various hiding level of symbols
  - **`hilight_color`** set hilight color for next hilight
  - **`infowindow_text`** ERC messages
  - **`intuitive_interface`** ERC messages
  - **`netlist_name`** set custom netlist name
  - **`netlist_type`** set netlisting mode (spice, verilog, vhdl, tedax, symbol)
  - **`no_draw`** set no drawing flag (0 or 1)
  - **`no_undo`** set to 1 to disable undo
  - **`raw_level`** set hierarchy level loaded raw file refers to
  - **`rectcolor`** set current layer (0, 1, ... , cadlayers-1)
  - **`sch_to_compare`** set name of schematic to compare current window with
  - **`schsymbolprop`** set global symbol attribute string
  - **`schprop`** set schematic global spice attribute string
  - **`schverilogprop`** set schematic global verilog attribute string
  - **`schvhdlprop`** set schematic global vhdl attribute string
  - **`schtedaxprop`** set schematic global tedax attribute string
  - **`text_svg`** set to 1 to use svg <text> elements
  - **`semaphore`** debug
  - **`show_hidden_texts`** set to 1 to enable showing texts with attr hide=true
  - **`sym_txt`** set to 0 to hide symbol texts
- **`set_different_tok str new_str old_str`**

```
Return string 'str' replacing/adding/removing tokens that are
different between 'new_str' and 'old_str'
```

- **`set_modify`**

```
Force modify status on current schematic
```

- **setprop instance|symbol|text|rect ref tok [val] [fast]**

```
    setprop instance inst [tok] [val] [fast]
set attribute 'tok' of instance (name or number) 'inst' to value 'val'
If 'tok' set to 'allprops' replace whole instance prop_str with 'val'
If 'val' not given (no attribute value) delete attribute from instance
If 'tok' not given clear completely instance attribute string
If 'fast' argument if given does not redraw and is not undoable

    setprop symbol name tok [val]
Set attribute 'tok' of symbol name 'name' to 'val'
If 'val' not given (no attribute value) delete attribute from symbol
This command is not very useful since changes are not saved into symbol
and netlisters reload symbols, so changes are lost anyway.

    setprop rect lay n tok [val] [fast|fastundo]
Set attribute 'tok' of rectangle number'n' on layer 'lay'
If 'val' not given (no attribute value) delete attribute from rect
If 'fast' argument is given does not redraw and is not undoable
If 'fastundo' s given same as above but action is undoable.

    setprop rect 2 n fullxzoom
    setprop rect 2 n fullyzoom
These commands do full x/y zoom of graph 'n' (on layer 2, this is hardcoded).

    setprop text n [tok] [val] [fast|fastundo]
Set attribute 'tok' of text number 'n'
If 'tok' not specified set text string (txt_ptr) to value
If "txt_ptr" is given as token replace the text txt_ptr ("the text")
If 'val' not given (no attribute value) delete attribute from text
If 'fast' argument is given does not redraw and is not undoable
If 'fastundo' s given same as above but action is undoable.
```

- **simulate [callback]**

```
Run a simulation (start simulator configured as default in
Tools -> Configure simulators and tools)
If 'callback' procedure name is given execute the procedure when simulation
is finished. all execute(..., id) data is available (id = execute(id) )
A callback prodedure is useful if simulation is launched in background mode
( set sim(spice,1,fg) 0 )
```

- **snap_wire**

```
Start a GUI start snapped wire placement (click to start a
wire to closest pin/net endpoint)
```

- **str_replace str rep with [escape]**

```
replace 'rep' with 'with' in string 'str'
if rep not preceeded by an 'escape' character
```

- **subst_tok str tok newval**

```
Return string 'str' with 'tok' attribute value replaced with 'newval'
```

- **symbol_in_new_window [new_process]**

```
When a symbol is selected edit it in a new tab/window if not already open.
If nothing selected open another window of the second schematic (issues a warning).
if 'new_process' is given start a new xschem process
```

- **swap_cursors**

```
swap cursor A (1)  and cursor B (2) positions.
```

- **swap_windows**

```
swap first and second window in window interface (internal command)
```

- **switch [window_path |schematic_name]**

```
Switch context to indicated window path or schematic name
returns 0 if switch was successfull or 1 in case of errors
(no tabs/windows present or no matching winpath / schematic name
found).
```

- **symbols [n | 'derived_symbols']**

```
if 'n' given list symbol with name or number 'n', else list all
if 'derived_symbols' is given list also symbols derived from base symbol
due to instance based implementation selection. This option must be used
after a netlist operation with 'keep_symbols' TCL variable set to 1
```

- **tab_list**

```
list all windows / tabs with window pathname and associated filename
```

- **table_read [table_file]**

```
If a simulation raw file is lodaded unload from memory.
else read a tabular file 'table_file'
First line is the header line containing variable names.
data is presented in column format after the header line
First column is sweep (x-axis) variable
Double empty lines start a new dataset
Single empty lines are ignored
Datasets can have different # of lines.
new dataset do not start with a header row.
Lines beginning with '#' are comments and ignored

   time    var_a   var_b   var_c
# this is a comment, ignored
    0.0     0.0     1.8    0.3
  <single empty line: ignored>
    0.1     0.0     1.5    0.6
    ...     ...     ...    ...
  <empty line>
  <Second empty line: start new dataset>
    0.0     0.0     1.8    0.3
    0.1     0.0     1.5    0.6
    ...     ...     ...    ...
```

- **test**

```
Testmode ...
```

- **text x y rot flip text props size draw**

```
Create a text object
  x, y, rot, flip specify the position and orientation
  text is the text string
  props is the attribute string
  size sets the size
  draw is a flag. If set to 1 will draw the created text
```

- **text_string n**

```
get text string of text object 'n'
```

- **toggle_colorscheme**

```
Toggle dark/light colorscheme
```

- **toggle_ignore**

```
toggle *_ignore=true attribute on selected instances
* = {spice,verilog,vhdl,tedax} depending on current netlist mode
```

- **touch x1 y1 x2 y2 x0 y0**

```
returns 1 if line {x1 y1 x2 y2} touches point {x0 y0}, 0 otherwise
```

- **translate n str**

```
Translate string 'str' replacing @xxx tokens with values in instance 'n' attributes
  Example: xschem translate vref {the voltage is @value}
  the voltage is 1.8
```

- **translate3 str eat_escapes s1 [s2] [s3]**

```
Translate string 'str' replacing @xxx tokens with values in string s1 or if
  not found in string s2 or if not found in string s3
  eat_escapes should be either 1 (remove backslashes) or 0 (keep them)
  Example: xschem translate3 {the voltage is @value} {name=x12} {name=x1 value=1.8}
  the voltage is 1.8
```

- **trim_chars str sep**

```
Remove leading and trailing chars matching any character in 'sep' from str
```

- **trim_wires**

```
Remove operlapping wires, join lines, trim wires at intersections
```

- **undo [redo [set_modify]]**

```
Undo last action. Optional integers redo and set_modify are passed to pop_undo()
```

- **undo_type disk|memory**

```
Use disk file ('disk') or RAM ('memory') for undo bufer
```

- **unhilight_all [fast]**

```
if 'fast' is given do not redraw
Clear all highlights
```

- **unhilight**

```
Unhighlight selected nets/pins
```

- **unselect_all [draw]**

```
Unselect everything. If draw is given and set to '0' no drawing is done
```

- **update_all_sym_bboxes**

```
Update all symbol bounding boxes
```

- **update_op**

```
update tcl ngspice::ngspice array data from raw file point 0
```

- **view_prop**

```
View attributes of selected element (read only)
if multiple selection show the first element (in xschem  array order)
```

- **warning_overlapped_symbols [sel]**

```
Highlight or select (if 'sel' set to 1) perfectly overlapped instances
this is usually an error and difficult to grasp visually
```

- **windowid topwin_path**

```
Used by xschem.tcl for configure events (set icon)
```

- **wire_coord n**

```
return 4 coordinates of wire[n]
```

- **wire [x1 y1 x2 y2] [pos] [prop] [sel]**

```
Place a new wire
if no coordinates are given start a GUI wire placement
```

- **wire_cut [x y] [noalign]**

```
start a wire cut operation. Point the mouse in the middle of a wire and
Alt-click right button.
if x and y are given cut wire at given point
if noalign is given and is set to 'noalign' do not align the cut point to closest snap po
```

- **xcb_info**

```
For debug
```

- **zoom_box [x1 y1 x2 y2] [factor]**

```
Zoom to specified coordinates, if 'factor' is given reduce view (factor < 1.0)
or add border (factor > 1.0)
If no coordinates are given start GUI zoom box operation
```

- **zoom_full [center|nodraw|nolinewidth]**

```
Set full view.
If 'center' is given center vire instead of lower-left align
If 'nodraw' is given don't redraw
If 'nolinewidth]' is given don't reset line widths.
```

- **zoom_hilighted**

```
Zoom to highlighted objects
```

- **zoom_in**

```
Zoom in drawing
```

- **zoom_out**

```
Zoom out drawing
```

- **zoom_selected**

```
Zoom to selection
```

XSCHEM TCL GLOBAL VARIABLES

```
# default command for first spice simulation command (interactive ngspice)
sim(spice,0,cmd) {$terminal -e 'ngspice -i "$N" -a || sh'}

# flag for foreground (1) or background (0) operation
sim(spice,0,fg) 0

# flag for status dialog box opening (1) at simulation end or not (0)
sim(spice,0,st) 0

sim(spice,1,cmd) {ngspice -b -r "$n.raw" -o "$n.out" "$N"}
sim(spice,1,fg) 0
sim(spice,1,st) 1
sim(spice,2,cmd) "Xyce \"\$N\"\n# Add -r \"\$n.raw\" if you want all variables saved"
sim(spice,2,fg) 0
sim(spice,2,st) 1
sim(spice,3,cmd) {mpirun /path/to/parallel/Xyce "$N"}
sim(spice,3,fg) 0
sim(spice,3,st) 1

# Number of configured spice simulation commands (4), [ sim(spice,0,...) ... sim(spice,3,...) ]
sim(spice,n) 4

# default spice command to use (0) --> sim(spice,0,...)
sim(spice,default) 0
sim(spicewave,0,cmd) {gaw "$n.raw" }
sim(spicewave,0,fg) 0
sim(spicewave,0,st) 0
sim(spicewave,1,cmd) {$terminal -e ngspice}
sim(spicewave,1,fg) 0
sim(spicewave,1,st) 0
sim(spicewave,2,cmd) {rawtovcd -v 1.5 "$n.raw" > "$n.vcd" && gtkwave "$n.vcd" "$n.sav" 2>/dev/nu
sim(spicewave,2,fg) 0
sim(spicewave,2,st) 0
sim(spicewave,3,cmd) {$env(HOME)/analog_flavor_eval/bin/bspwave --socket localhost $bespice_list
sim(spicewave,3,fg) 0
sim(spicewave,3,st) 0
sim(spicewave,n) 4
sim(spicewave,default) 0

# list of configured tools. For each of these there is a set of sim(tool,...) settings
sim(tool_list) spice spicewave verilog verilogwave vhdl vhdlwave

sim(verilog,0,cmd) {iverilog -o .verilog_object -g2012 "$N" && vvp .verilog_object}
sim(verilog,0,fg) 0
sim(verilog,0,st) 1
sim(verilog,n) 1
sim(verilog,default) 0
sim(verilogwave,0,cmd) {gtkwave dumpfile.vcd "$N.sav" 2>/dev/null}
sim(verilogwave,0,fg) 0
sim(verilogwave,0,st) 0
sim(verilogwave,n) 1
sim(verilogwave,default) 0
sim(vhdl,0,cmd) {ghdl -c --ieee=synopsys -fexplicit "$N" -r "$s" --wave="$n.ghw"}
sim(vhdl,0,fg) 0
sim(vhdl,0,st) 1
sim(vhdl,n) 1
sim(vhdl,default) 0
sim(vhdlwave,0,cmd) {gtkwave "$n.ghw" "$N.sav" 2>/dev/null}
sim(vhdlwave,0,fg) 0
sim(vhdlwave,0,st) 0
sim(vhdlwave,n) 1
sim(vhdlwave,default) 0

add_all_windows_drives 1
autofocus_mainwindow 1
auto_hilight 0
autotrim_wires 0
bespice_listen_port {}
bespice_server_getdata
big_grid_points 0
bus_replacement_char {} ;# use {<>} to replace [] with <> in bussed signals
cadlayers 22
cairo_font_line_spacing 1.0
cairo_font_name {Sans-Serif}
cairo_font_scale 1.0
cairo_vert_correct 0
case_insensitive 0
change_lw 1
color_ps 1
colors $dark_colors
compare_sch 0
component_browser_on_top 1
connect_by_kissing 0
constrained_move 0
copy_cell 0
dark_colors {
  "#000000" "#00ccee" "#3f3f3f" "#cccccc" "#88dd00" "#bb2200" "#00ccee" "#ff0000"
  "#ffff00" "#ffffff" "#ff00ff" "#00ff00" "#0044dd" "#aaaa00" "#aaccaa" "#ff7777"
  "#bfff81" "#00ffcc" "#ce0097" "#d2d46b" "#ef6158" "#fdb200"}
dark_colorscheme 1
dark_colors_save
debug_var 0
delay_flag
dim_bg 0.0
dim_value 0.0
dircolor(/share/doc/xschem/) {#338844}
dircolor(/share/xschem/) red
disable_unique_names 0
download_url_helper {curl -f -s -O}
draw_grid 1
draw_window 0
editor {gvim -f}
edit_prop_size 80x12
edit_symbol_prop_new_sel {}
enable_dim_bg 0
enable_layer($i) 1
enable_stretch 0
en_hilight_conn_inst 0
execute(cmd,<id>)
execute(data,<id>)
execute(status,<id>)
execute(cmd,last)
execute(data,last)
execute(status,last)
execute(error,last)
execute(exitcode,last)
execute(id)
flat_netlist 0
fullscreen 0
gaw_tcp_address {localhost 2020}
graph_bus 0
graph_logx 0
graph_logy 0
graph_rainbow 0
graph_raw_level -1 ;# hierarchy level where raw file has been loaded
graph_schname {}
graph_sel_color 4
graph_selected {}
graph_sel_wave {}
graph_sort 0
has_cairo 1
has_x
hide_empty_graphs 0 ;# if set to 1 waveform boxes will be hidden if no raw file loaded
hide_symbols 0
incr_hilight 1

# text saved into the ERC informational dialog box.
# netlist warnings and errors are shown here.
infowindow_text

initial_geometry {900x600}
launcher_default_program {xdg-open}
light_colors {
  "#ffffff" "#0044ee" "#aaaaaa" "#222222" "#229900" "#bb2200" "#00ccee" "#ff0000"
  "#888800" "#00aaaa" "#880088" "#00ff00" "#0000cc" "#666600" "#557755" "#aa2222"
  "#7ccc40" "#00ffcc" "#ce0097" "#d2d46b" "#ef6158" "#fdb200"}
light_colors_save
line_width 0
live_cursor2_backannotate 0

# if set use <sch_dir>/simulation for netlist and sims
local_netlist_dir 0
```

```
lvs_ignore 0
lvs_netlist 0
measure_text "y=\nx="
menu_debug_var 0
myload_files2 {}
myload_globfilter {*}
myload_index1 0
netlist_dir "$USER_CONF_DIR/simulations"
netlist_show 0
netlist_type spice
nocairo_font_xscale .85
nocairo_font_yscale .88
nocairo_vert_correct 0
no_change_attrs 0
nolist_libs {}
noprint_libs {}
only_probes 0  ; # 20110112
OS
persistent_command 0
preserve_unchanged_attrs 0
rainbow_colors 0
search_schematic 0
show_hidden_texts 0
show_infowindow 0
show_infowindow_after_netlist 0
show_pin_net_names 0
spiceprefix 1
split_files 0
svg_font_name {Sans-Serif}
symbol_width 150
sym_txt 1
tabbed_interface 0
tcl_files {}
tclstop 0
terminal xterm
text_line_default_geometry 80x12
textwindow_wcounter
toolbar_horiz   1
toolbar_list {  ... }
toolbar_visible 0
to_pdf {ps2pdf}
to_png {gm convert}
transparent_svg 0
undo_type disk
unzoom_nodrift 0
use_tclreadline 1 ;# use the tclreadline package for command prompt. default: 1
USER_CONF_DIR
verilog_2001 1
verilog_bitblast 0
viewdata_wcounter
xschem_libs {}
xschem_listen_port {}
xschem_server_getdata
XSCHEM_SHAREDIR
XSCHEM_START_WINDOW {}
XSCHEM_TMP_DIR {/tmp}
zoom_full_center 0
```

Simulator / waveform setup

In xschem a tcl array variable **`sim`** is used to specify external process commands, like simulators and waveform viewers. This variable is set in the GUI with the **`Simulation-> Configure simulators and tools`** menu entry. First of all you need to set the **`tool_list`** list of configured tools:

```
set sim(tool_list) { spice spicewave verilog verilogwave vhdl vhdlwave }
```

For each tool you need to define some sub elements:

```
# Number of configured spice simulation commands (4), [ sim(spice,0,...) ... sim(spice,3,...)
sim(spice,n) 4
# default spice command to use (0) --> sim(spice,0,...)
sim(spice,default) 0
# default command for first spice simulation command (interactive ngspice)
sim(spice,0,cmd) {$terminal -e 'ngspice -i "$N" -a || sh'}
# flag for foreground (1) or background (0) operation
sim(spice,0,fg) 0
# flag for status dialog box opening (1) at simulation end or not (0)
sim(spice,0,st) 0
sim(spice,1,cmd) {ngspice -b -r "$n.raw" -o "$n.out" "$N"}
sim(spice,1,fg) 0
sim(spice,1,st) 1
sim(spice,2,cmd) "Xyce \"\$N\"\n# Add -r \"\$n.raw\" if you want all variables saved"
sim(spice,2,fg) 0
sim(spice,2,st) 1
sim(spice,3,cmd) {mpirun /path/to/parallel/Xyce "$N"}
sim(spice,3,fg) 0
sim(spice,3,st) 1
```

XSCHEM TCL PROCEDURES

Commands in brackets are internal procedures, not supposed to be used by end users

```
# show xschem about dialog
about

# given a symbol reference 'sym' return its absolute path
# Example: % abs_sym_path devices/iopin.sch
#          /home/schippes/share/xschem/xschem_library/devices/iopin.sym
abs_sym_path sym

add_ext
add_lab_no_prefix
add_lab_prefix

# show an alert dialog box and display 'text'.
# if 'position' is empty (example: alert_ {hello, world} {}) show at mouse coordinates
# otherwise use specified coordinates example: alert_ {hello, world} +300+400
# if nowait is 1 do not wait for user to close dialog box
# if yesnow is 1 show yes and no buttons and return user choice (1 / 0).
# (this works only if nowait is unset).
alert_ text [position] [nowait] [yesno]

ask_save
attach_labels_to_inst
```

```
balloon
balloon_show
bespice_getdata
bespice_server
build_widgets
change_color
clear_simulate_button
color_dim
context_menu
convert_to_pdf
convert_to_png
create_layers_menu
create_pins

# pause execution for 'ms milliseconds, keeping event loop responding
delay [ms]
delete_ctx
delete_files
delete_tab
descend_hierarchy
download_url
edit_file
edit_netlist
edit_prop
edit_vi_netlist_prop
edit_vi_prop
enter_text

# evaluate 'expr'. if 'expr' has errors or does not evaluate return 'expr' as is
ev expr

every
execute
execute_fileevent
execute_wait
fill_graph_listbox

# find file into $paths directories matching $f
# use $pathlist global search path if $paths empty
# recursively descend directories
find_file f [paths]

# as above, return only first match found
find_file_first f [paths]

# process all symbols in current design, get full path of them if found in
# XSCHEM_LIBRARY_PATH search path, then transform them with exactly one 'n_dir'
# path components added.
# example: current design has an instance referencing 'lab_pin.sym'
#          after executing 'fix_symbols 1' the instance symbol reference
#          will be devices/lab_pin.sym. This will be done only on symbols
#          that are existing in the current search paths (there is
#          devices/lab_pin.sym in one of the search paths).
fix_symbols n_dirs

from_eng
gaw_cmd
gaw_echoline
get_cell
get_directory
get_file_path

# launch a terminal shell, if 'curpath' is given set path to 'curpath'
get_shell
```

```
graph_add_nodes
graph_add_nodes_from_list
graph_change_wave_color
graph_edit_properties
graph_edit_wave
graph_get_signal_list
graph_show_measure
graph_update_nodelist
hash_string
history
housekeeping_ctx
infowindow
input_line
inutile
inutile_alias_window
inutile_get_time
inutile_help_window
inutile_line
inutile_read_data
inutile_template
inutile_translate
inutile_write_data
is_xschem_file
key_binding
launcher
list_hierarchy
list_tokens
load_file_dialog
load_file_dialog_mkdir
load_file_dialog_up
load_recent_file
make_symbol
make_symbol_lcc

# find files into $paths directories matching $f
# use $pathlist global search path if $paths empty
# recursively descend directories
match_file f [paths]

myload_display_preview
myload_getresult
myload_place_symbol
myload_set_colors1
myload_set_colors2
myload_set_home
netlist
next_tab
no_open_dialogs
order
pack_tabs
pack_widgets
path_head
pin_label
prev_tab
print_help_and_exit
probe_net
property_search

# quit xschem closing all tabs/windows (including the first/main)
# user has the option to cancel the closing of modified tabs/windows
# if 'force' is given no confirmation is asked and modified content is lost.
# the number of schematic views  left over (in addition to main window)
#  is returned. If only one (the main view) is left command returns 0.
```

```
quit_xschem [force]

raise_dialog
read_data
read_data_nonewline
read_data_window
reconfigure_layers_button
reconfigure_layers_menu
rectorder
redef_puts

# Given an absolute path 'symbol' of a symbol/schematic remove the path prefix
# if file is in a library directory (a $pathlist dir)
# Example: rel_sym_path /home/schippes/share/xschem/xschem_library/devices/iopin.sym
#          devices/iopin.sym
rel_sym_path symbol

reroute_inst
reroute_net
reset_colors
restore_ctx
return_release
rotation
save_ctx
save_file_dialog
save_sim_defaults
schpins_to_sympins
select_inst
select_layers
set_bindings
set_env
set_graph_linewidth
set_initial_dirs
set_missing_colors_to_black

# set 'var' with '$val' if 'var' not existing
set_ne var val

# set_netlist_dir force [path]
# if path is given set as new netlist path where netlists and simulations are done.
# force should be always set to 1 unless you just want to query current path.
# select_netlist_dir 0 will return current path (you can get with $netlist_dir as well)
set_netlist_dir

set_old_tk_fonts

# when XSCHEM_LIBRARY_PATH is changed this function is called
# by 'trace_set_paths' refresh and cache new library search paths.
set_paths

set_replace_key_binding

# Initialize the tcl sim array variable (if not already set)
# setting up simulator / wave viewer commands
set_sim_defaults

set_tab_names
setglob
setup_recent_menu
setup_tabbed_interface
setup_tcp_bespice
setup_tcp_gaw
setup_tcp_xschem
setup_toolbar
```

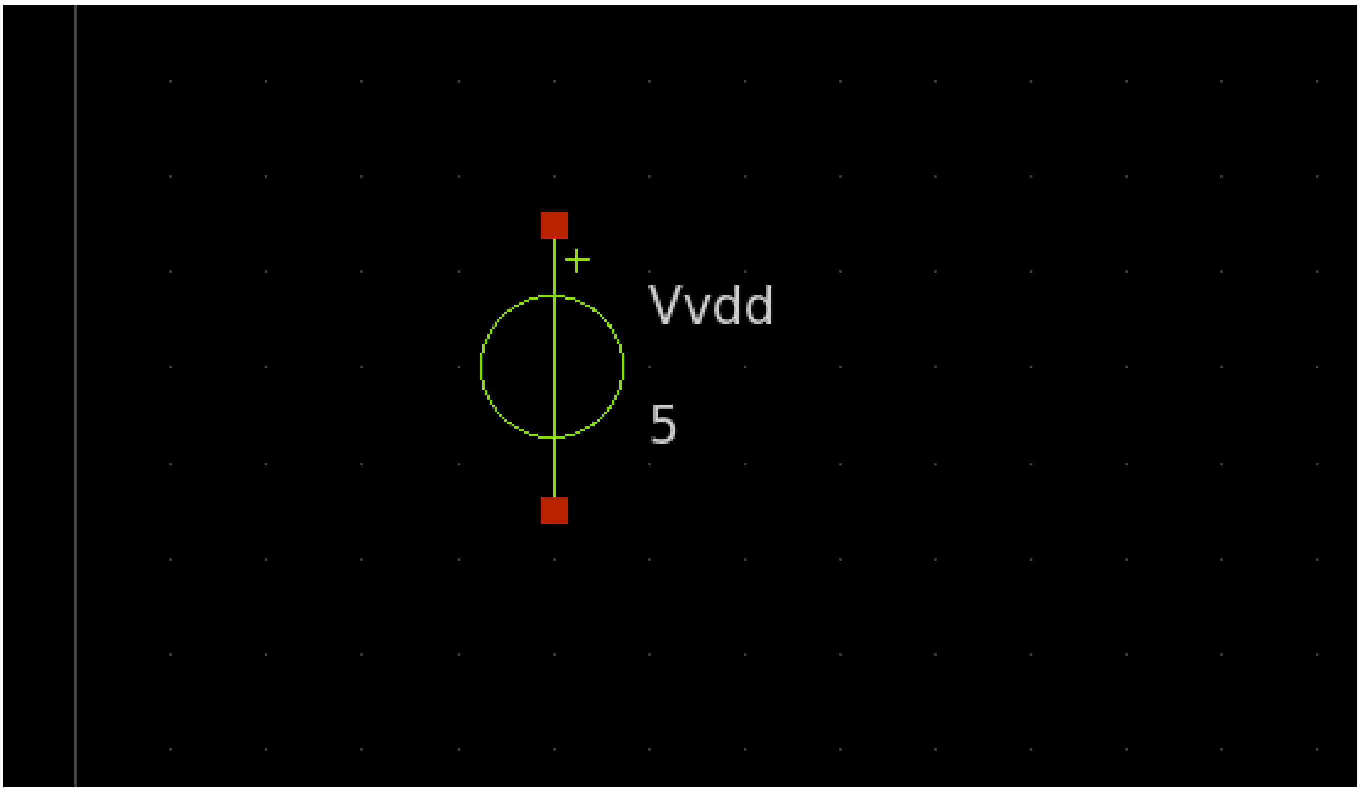
```
sframe

# show ERC (electrical rule check) dialog box
show_infotext

simconf
simconf_add
simconf_reset
simconf_saveconf
sim_is_ngspice
sim_is_xyce
sim_is_Xyce
simulate
simulate_button
simuldir
source_user_tcl_files
sub_find_file
sub_match_file
swap_compare_schematics
swap_tabs

# show a dialog box asking user to switch undo bguffer from memory to disk
switch_undo

# evaluate a tcl command from GUI
tclcmd

tclcmd_ok_button
tclpropeval
tclpropeval2
text_line
textwindow
to_eng
tolist
toolbar_add
toolbar_hide
toolbar_show

# this function executes whenever XSCHEM_LIBRARY_PATH changes (registered
# with a 'trace' command)
trace_set_paths

try_download_url
update_div
update_graph_node
update_recent_file
update_schematic_header
view_current_sim_output
waves
write_data
write_recent_file
xschem_getdata
xschem_server
```

SOME USEFUL SCRIPT EXAMPLES

The following examples show the xschem commands one by one. In general you should create small TCL procedures to perform these tasks. This way you can optimize things, for example creating temporary variables holding the output of the various **`xschem ...`** commands.

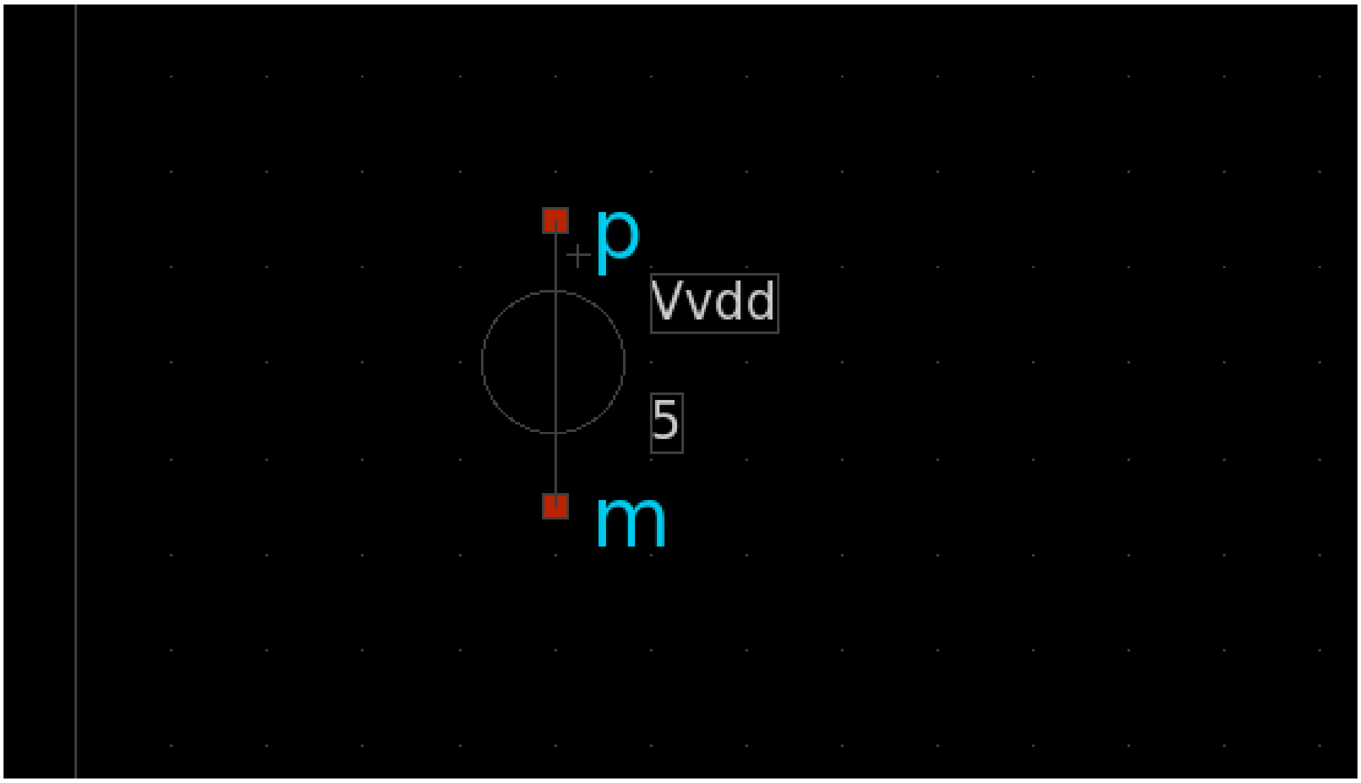
- **Instantiate a component and wire it up with specific nets on its terminals.**

```
# Create a 5V Vvdd voltage source
xschem instance vsource.sym 100 100 0 0 {name=Vvdd value=5}
```

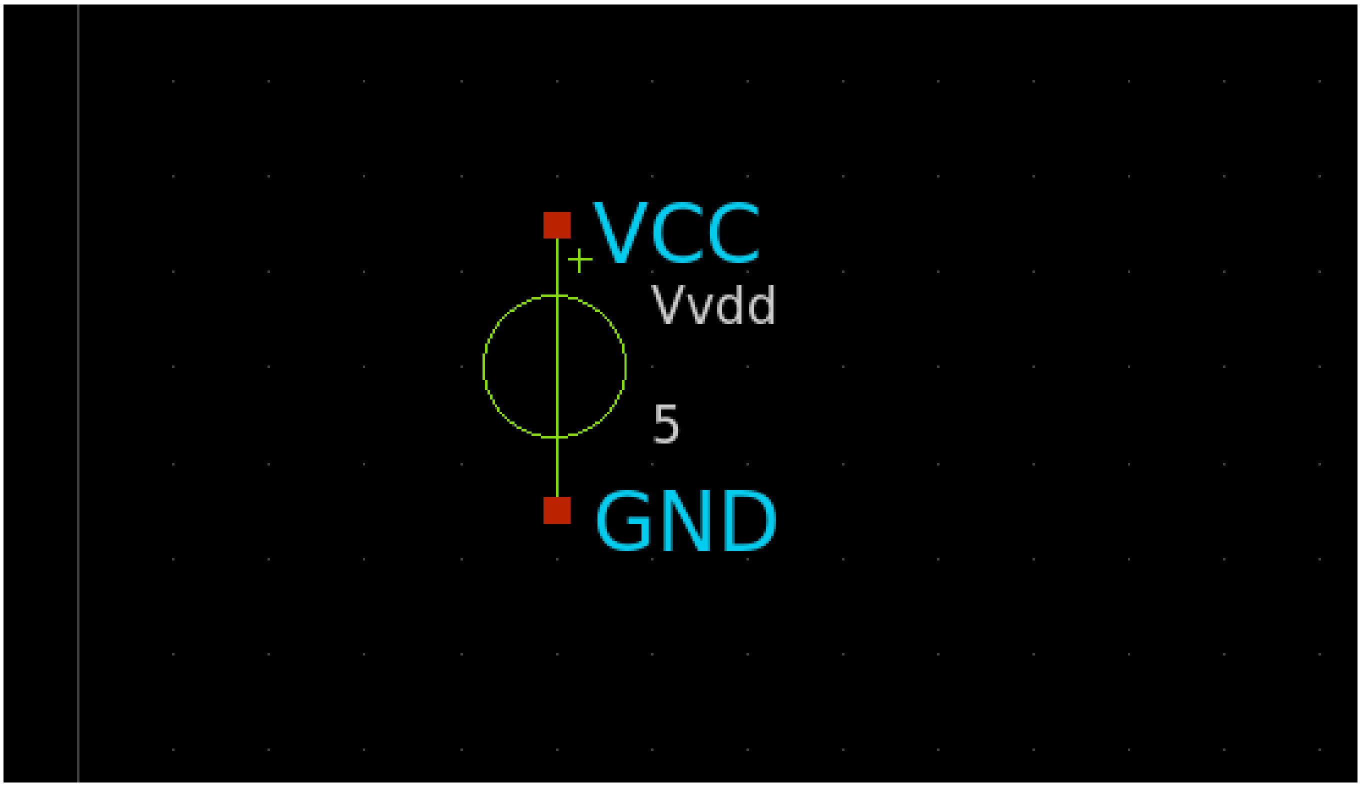


```
# Attach labels, They will get the symbol pin labels
xschem select instance vvdd
xschem attach_labels
```

Instantiate a component and wire it up with specific nets on its terminals.

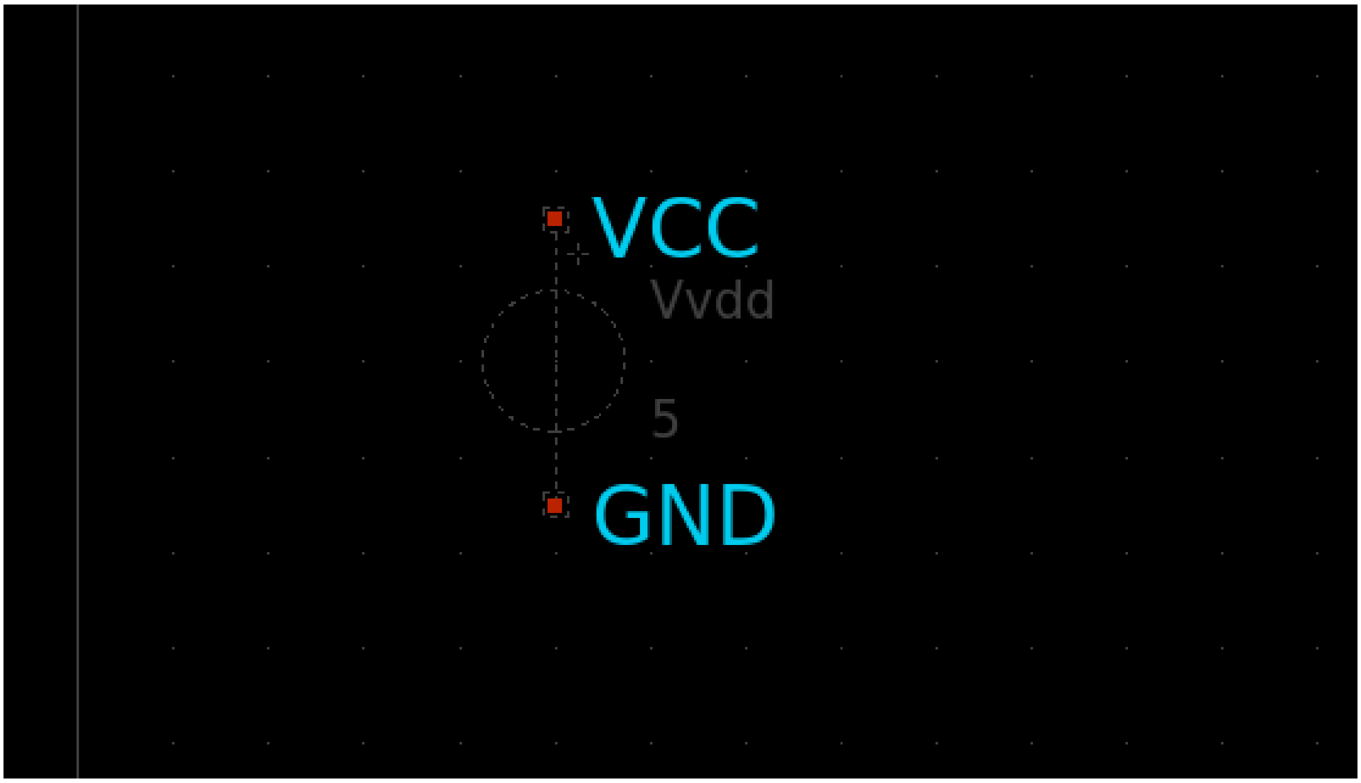


```
# Select labels, unselect vsource and change positive and negative terminal
# labels to VCC and GND respectively
# The first item in the selected_set list is the first vsource terminal
# (the positive terminal), the second one is the negative terminal.
# At the end unselect all
xschem connected_nets
xschem select instance Vvdd clear
xschem setprop instance [lindex [xschem selected_set] 0] lab VCC
xschem setprop instance [lindex [xschem selected_set] 1] lab GND
xschem unselect_all
```



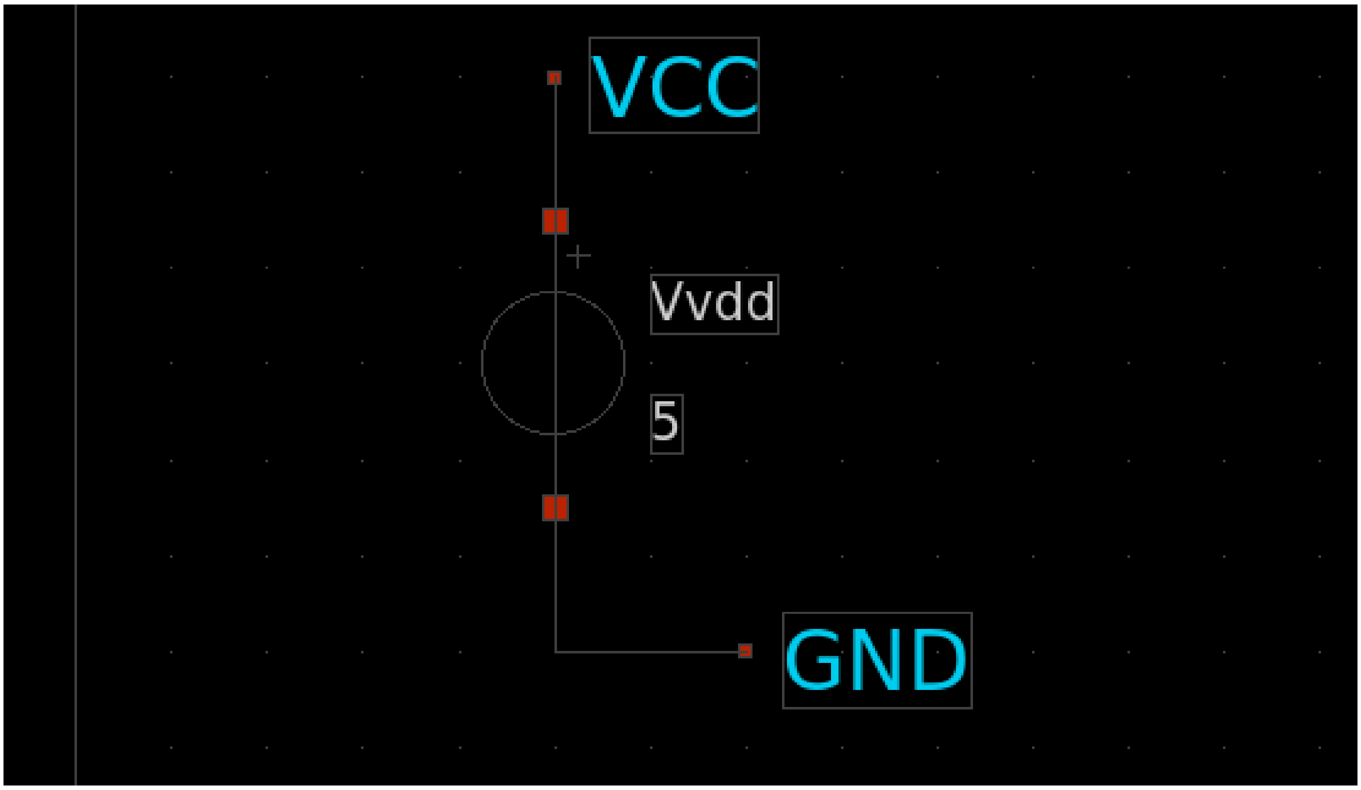
- **Disable a component in the schematic**

```
# Add spice_ignore=true attribute
# the component will be ignored in generated netlists.
xschem setprop instance Vvdd spice_ignore true
```

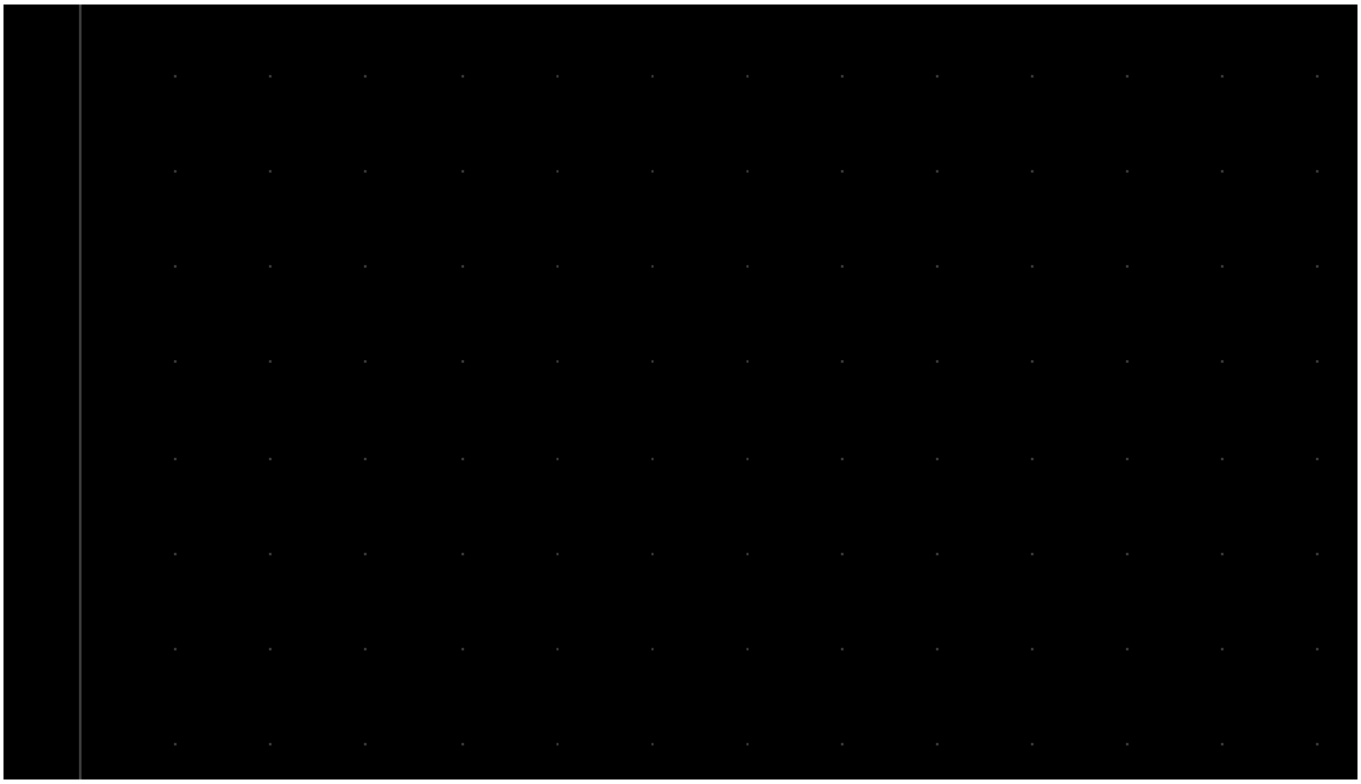


- **Delete a component together with its attached nets**

```
# select component, select attached nets and delete
# this will also select wire segments if labels are attached to selected instance with wires
xschem select instance Vvdd
xschem connected_nets
```

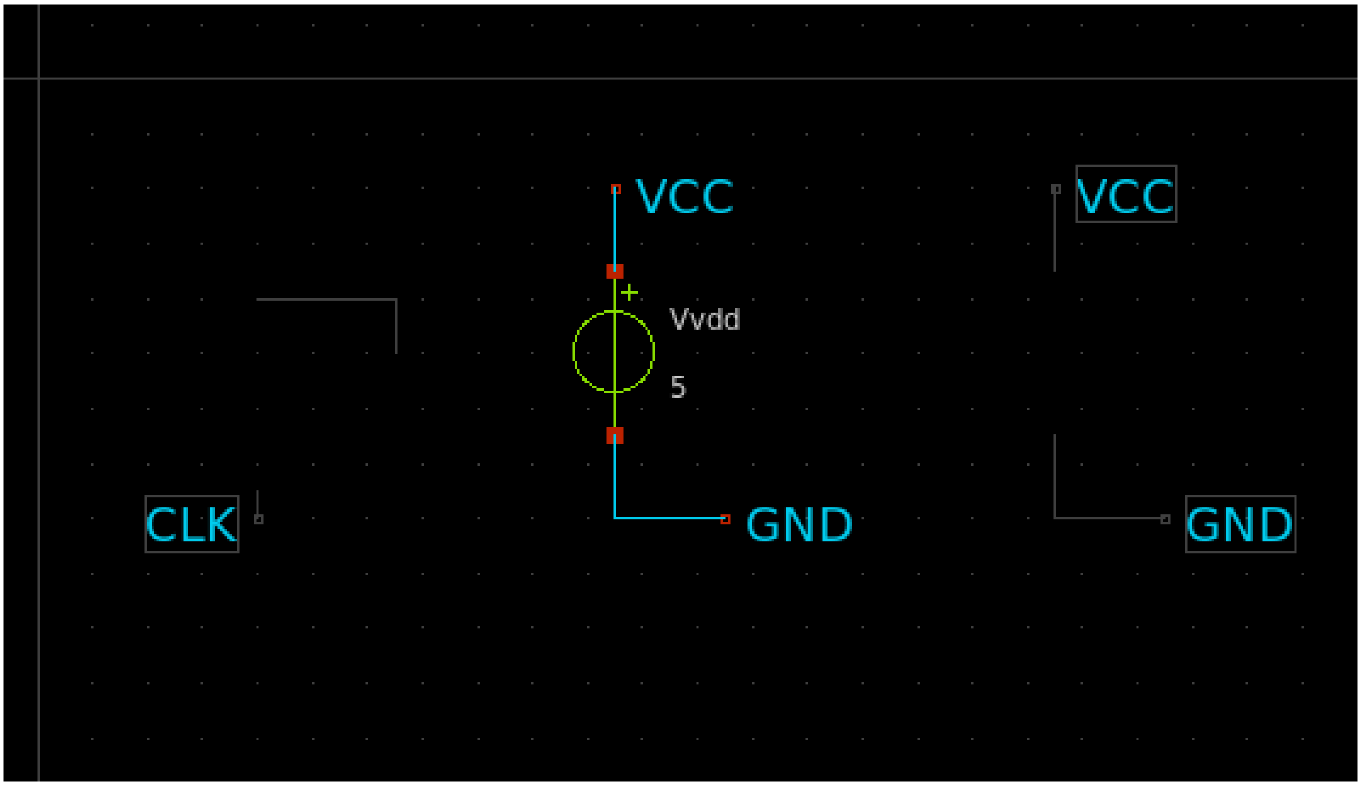


```
# Delete selection
xschem delete
```



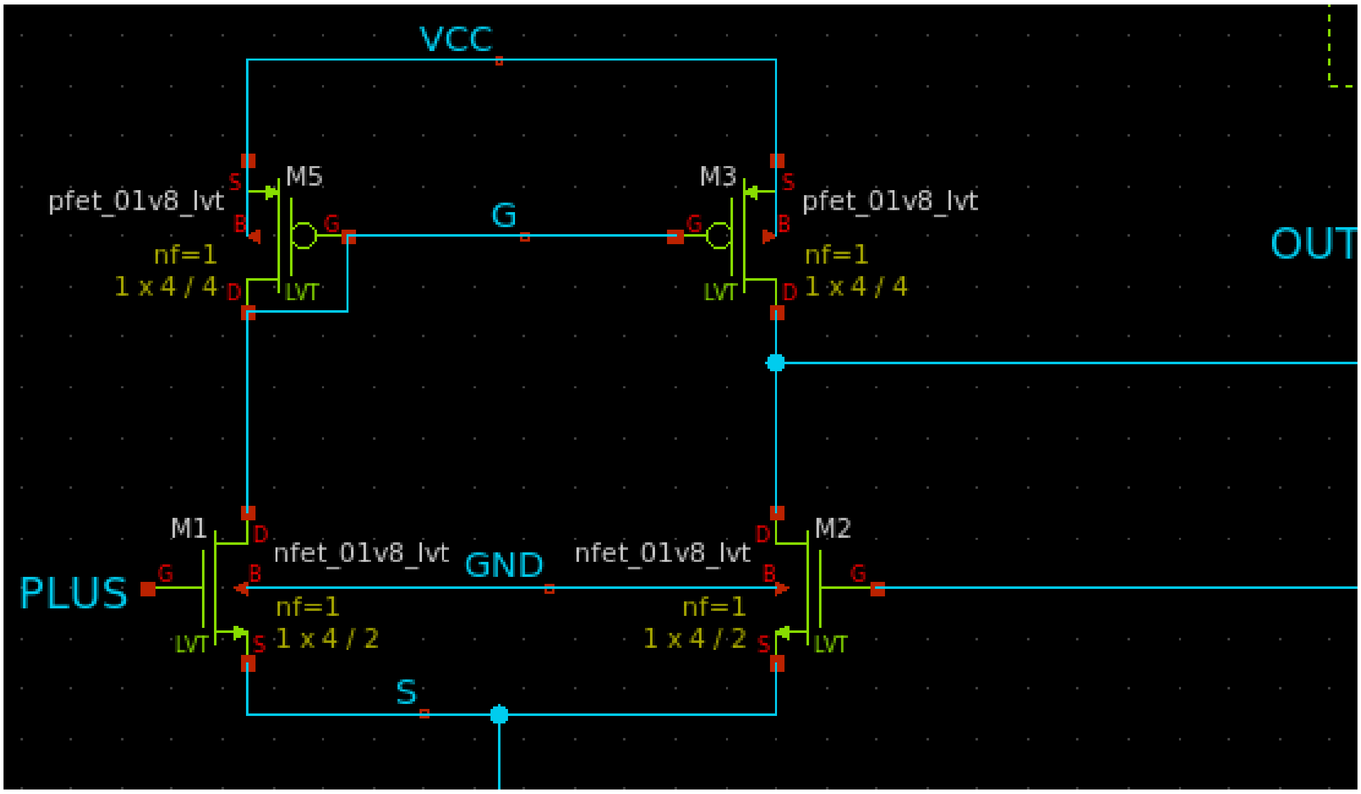
- **Delete dangling nets and labels**

```
# If after some editing or deletions dangling nets are present
# they can all be selected. Deletion may be done with a "xschem delete" command.
xschem select_dangling_nets
```

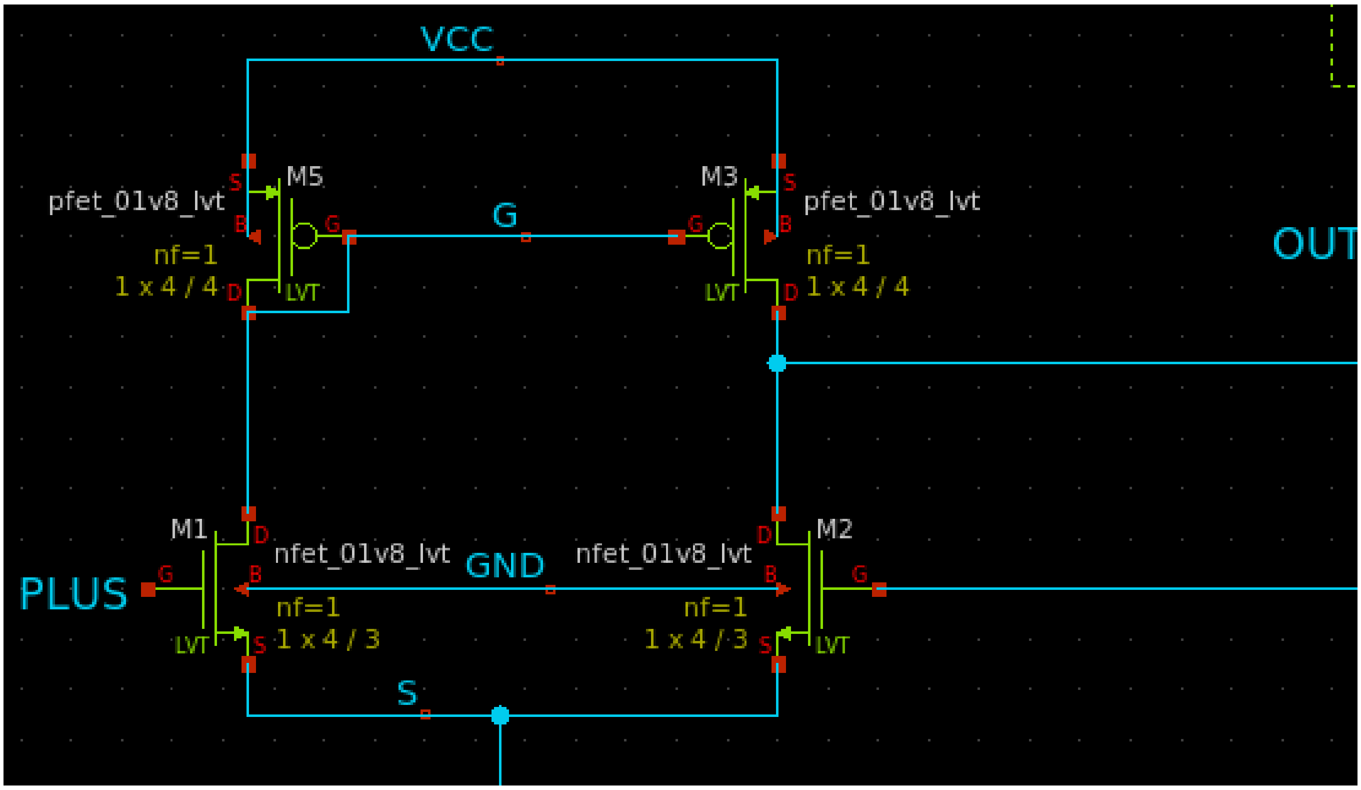


- **Change attributes of a group of components**

```
# From this situation we want to select all MOS elements with L=2
# and modify L (gate length) to 3
```

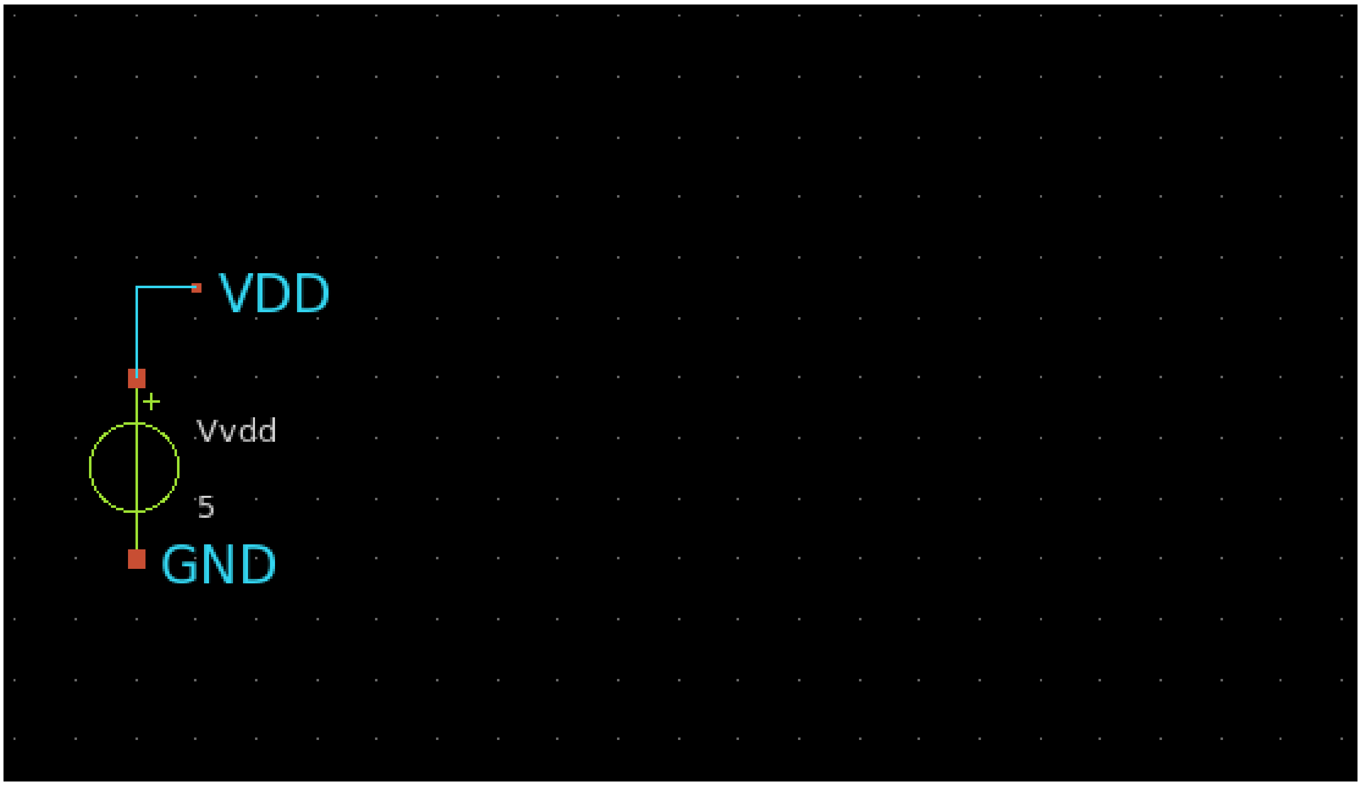


```
# Do an exact search of elements with L=2
xschem search exact 1 L 2
# a more precise search to avoid selecting unwanted elements might be:
# xschem search regex 1 propstring "L=2\[ \n\].*model=nfet"
# the above command will do a regular expression search on the whole
# instance property string (the special token propstring)
foreach i [xschem selected_set] { xschem setprop instance $i L 3}
xschem unselect_all
```

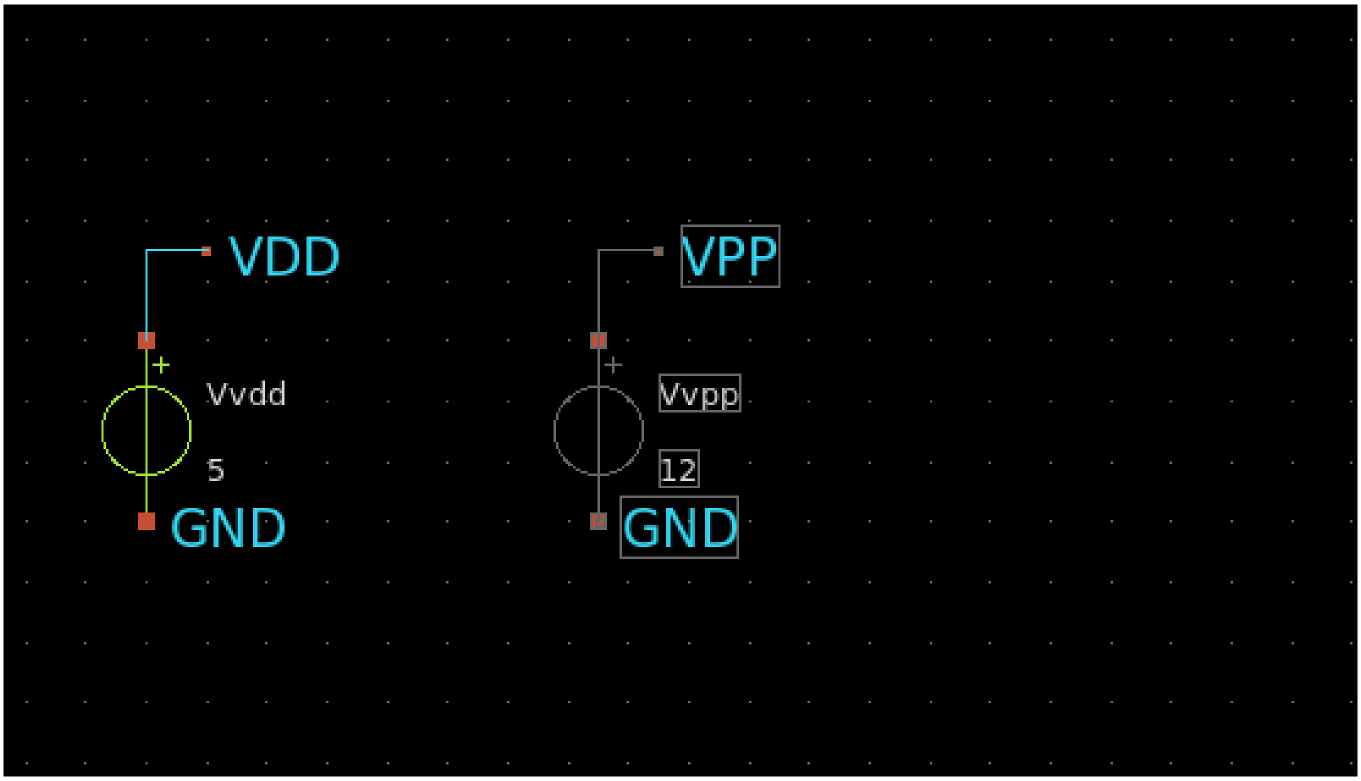


• Copy a components with its wired terminals

```
# From this situation we want to copy Vvdd to a different location
# and change the instance name, voltage value  and its positive terminal net name
```

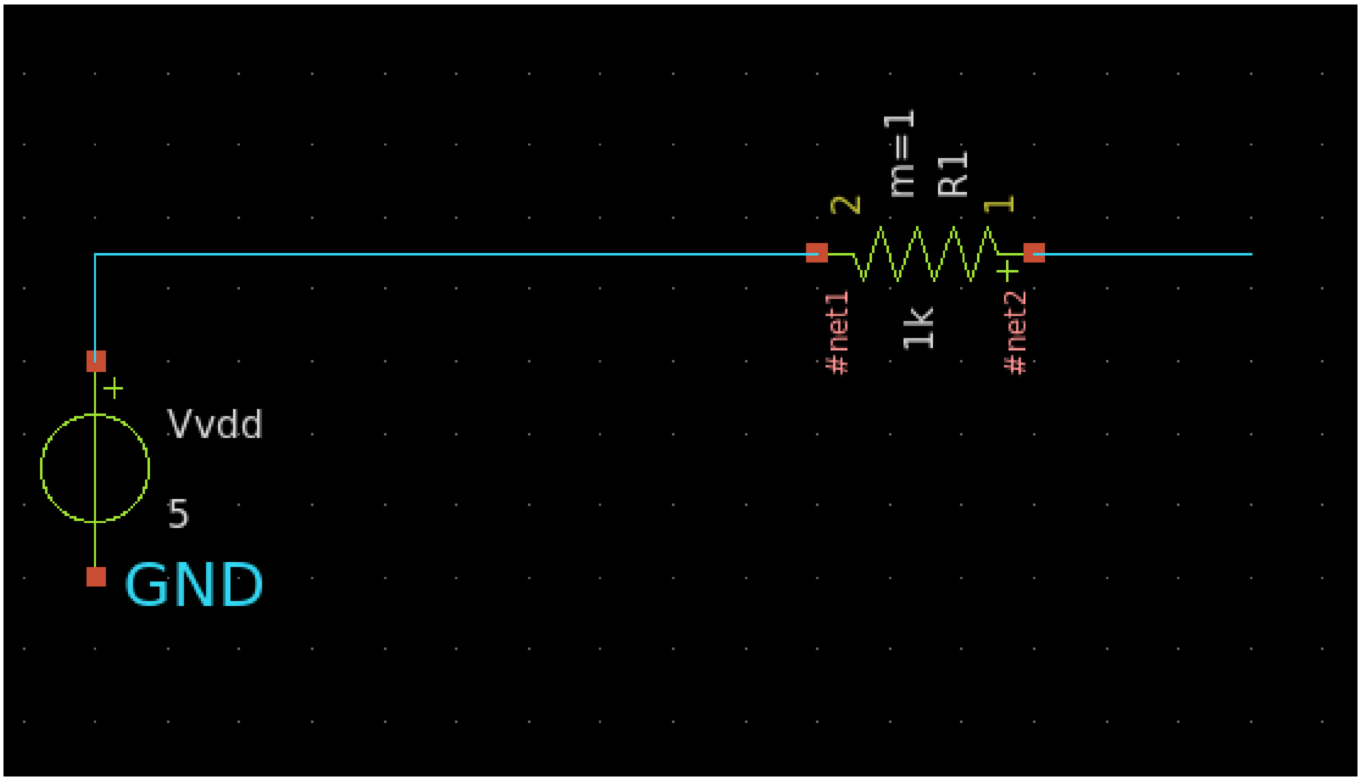


```
# select the desired instance
xschem select instance Vvdd
# select attached wires
xschem connected_nets
# Copy to clipboard
xschem copy
# Paste selection 150 x-axis units to the right
xschem paste 150 0
# First selected_set item is the voltage source (it was selected first)
xschem setprop instance [lindex [xschem selected_set] 0] name Vvpp
xschem setprop instance [lindex [xschem selected_set] 0] value 12
# Following item is the net label attached to the first symbol pin
xschem setprop instance [lindex [xschem selected_set] 1] lab VPP
```

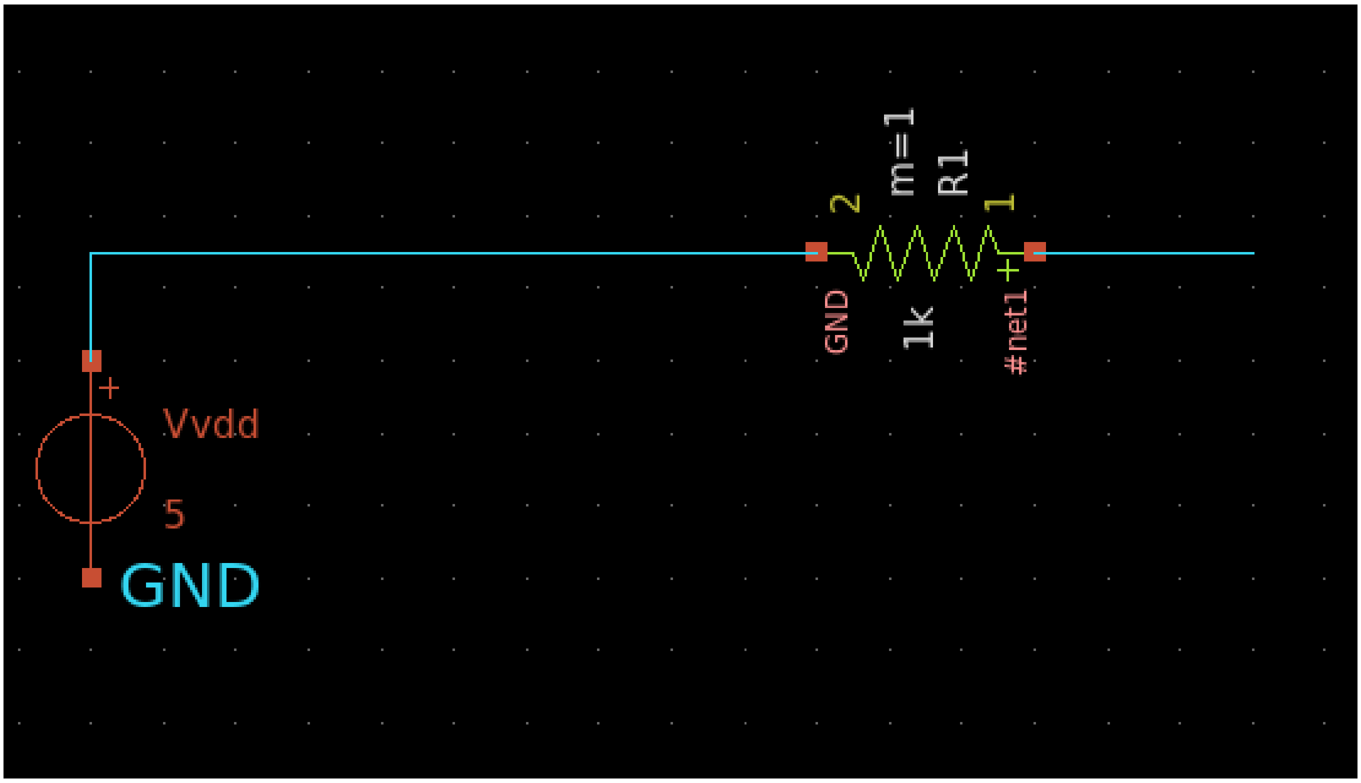


- **Transform a component into a short**

```
# We want to transform this voltage source into a short, passing the
# negative label onto the positive terminal.
# Warning: no net label must be present on the positive net, otherwise you end up with
# an ERC error (multiple differnet labels on the same net)
```

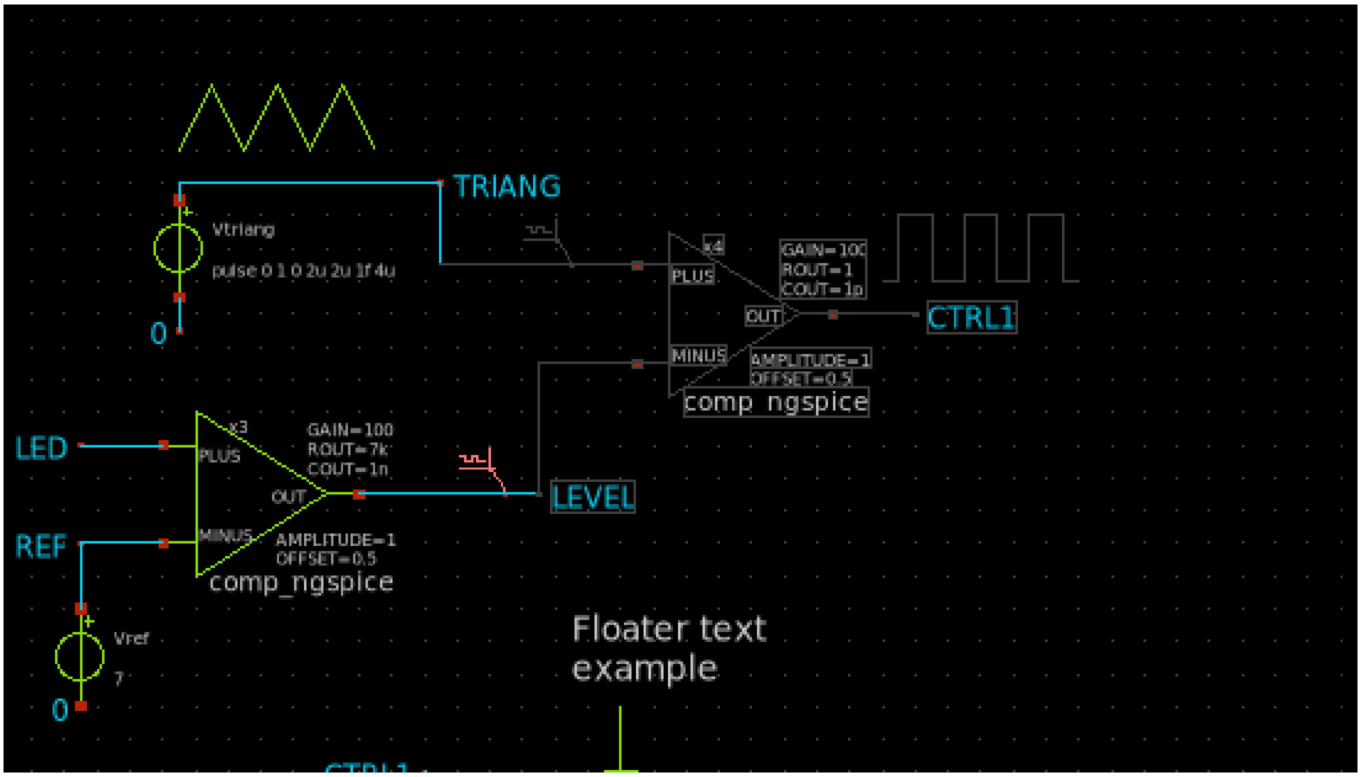


```
# add spice_ignore=short attribute to instance
# All pins of the instance will be shorted together to the same net.
# Instance will be shown in red to indicate the short condition.
# Option "Options->Show net name on symbol pins" is enabled and attribute
# net_name=true is set on the resistor to show net names.
# you see the left resistor terminal is GND now.

xschem setprop instance Vvdd spice_ignore short
```

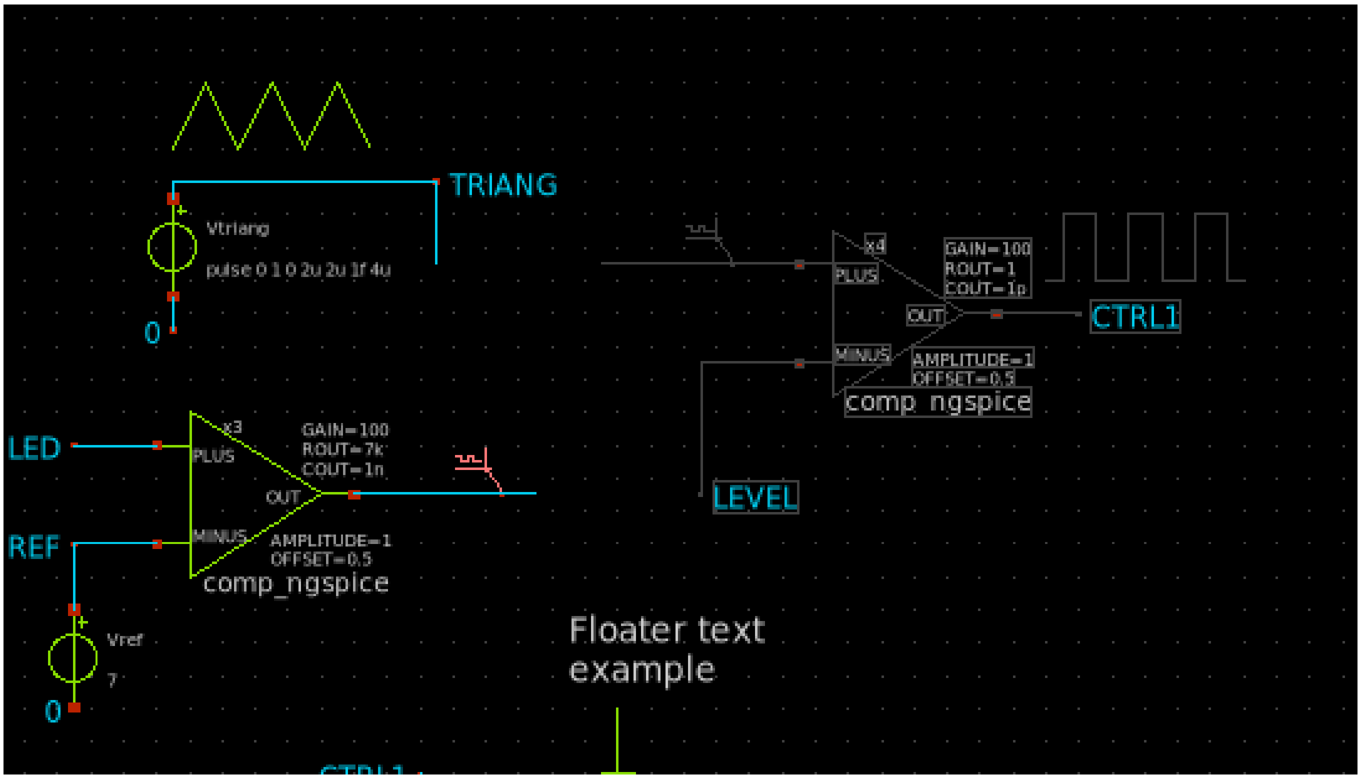


- **Move a selected portion of the schematic**

```
# After selecting some objects...
```

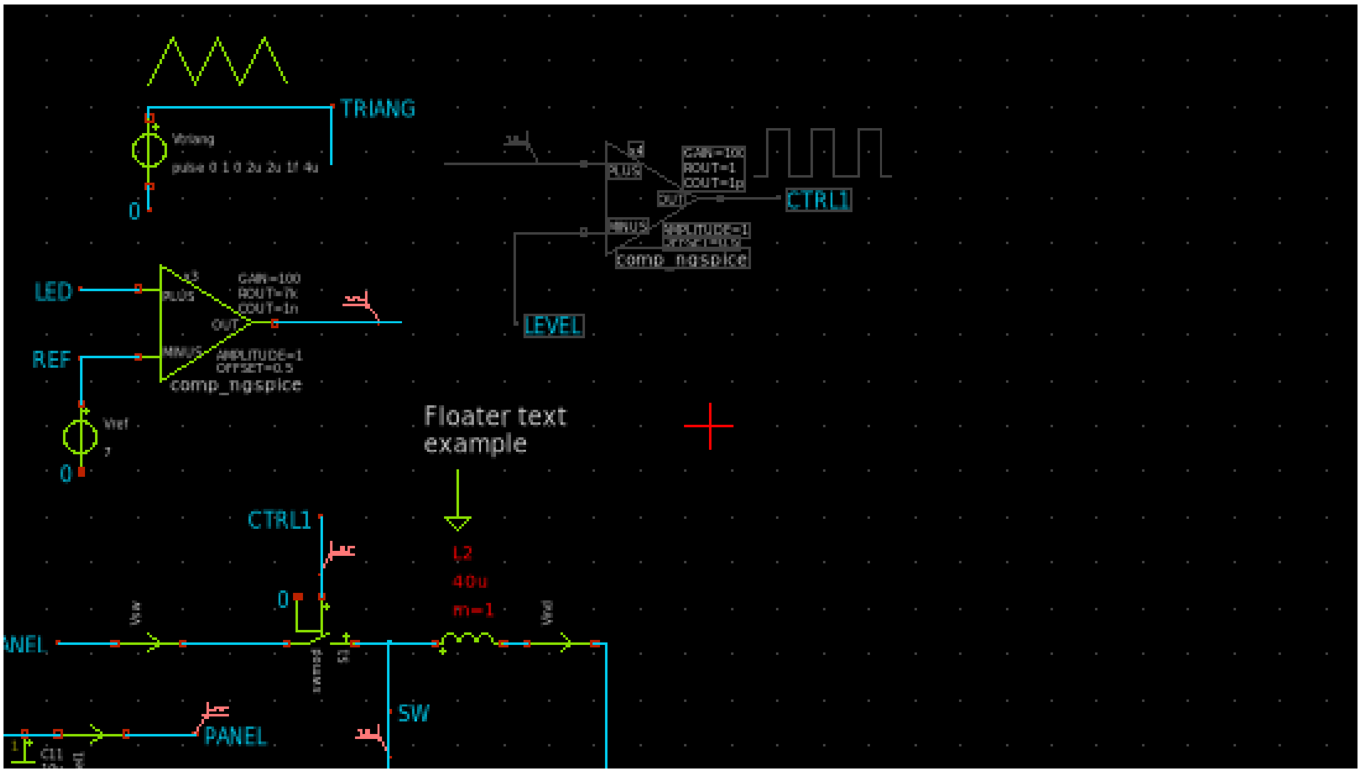


```
# ... We move them by some X / Y quantities.
xschem move_objects 100 0
```

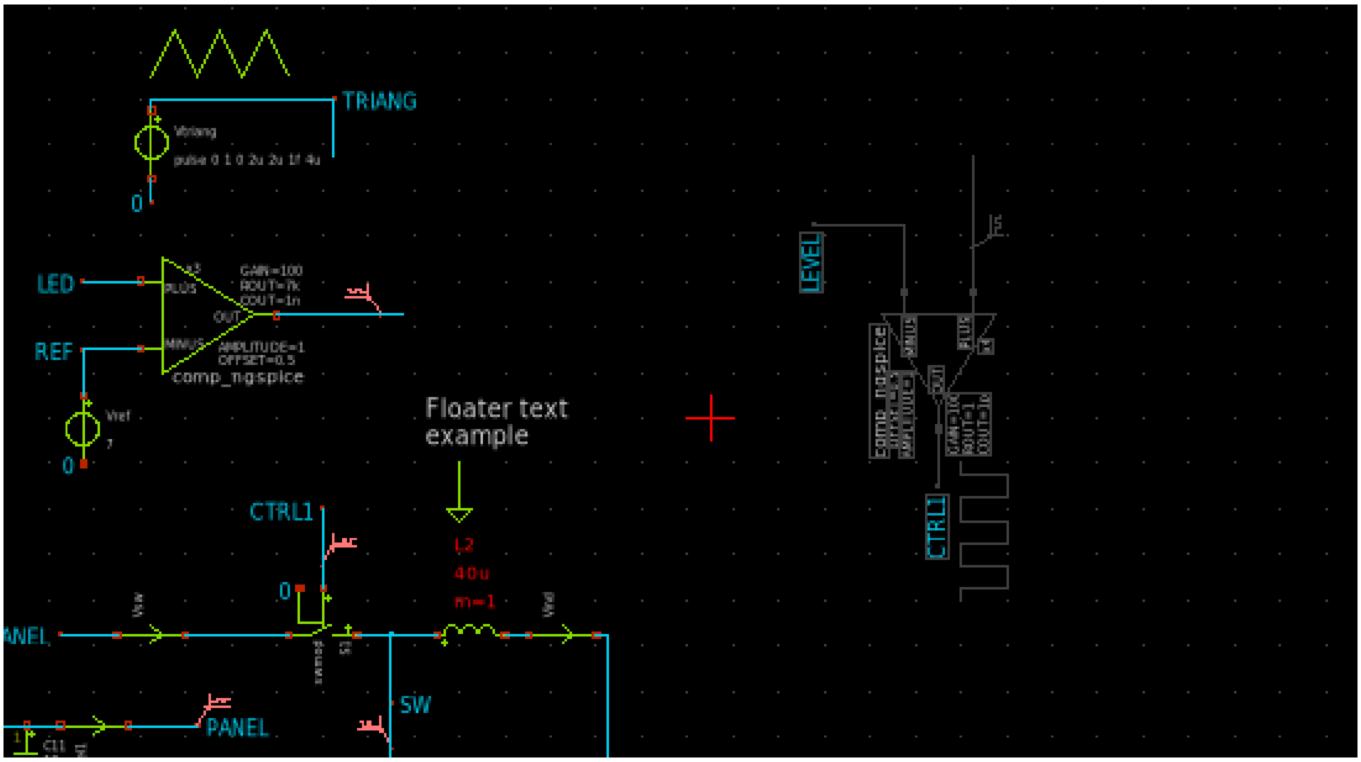


- **Rotate a selected portion of the schematic**

```
# After selecting some objects ...
```

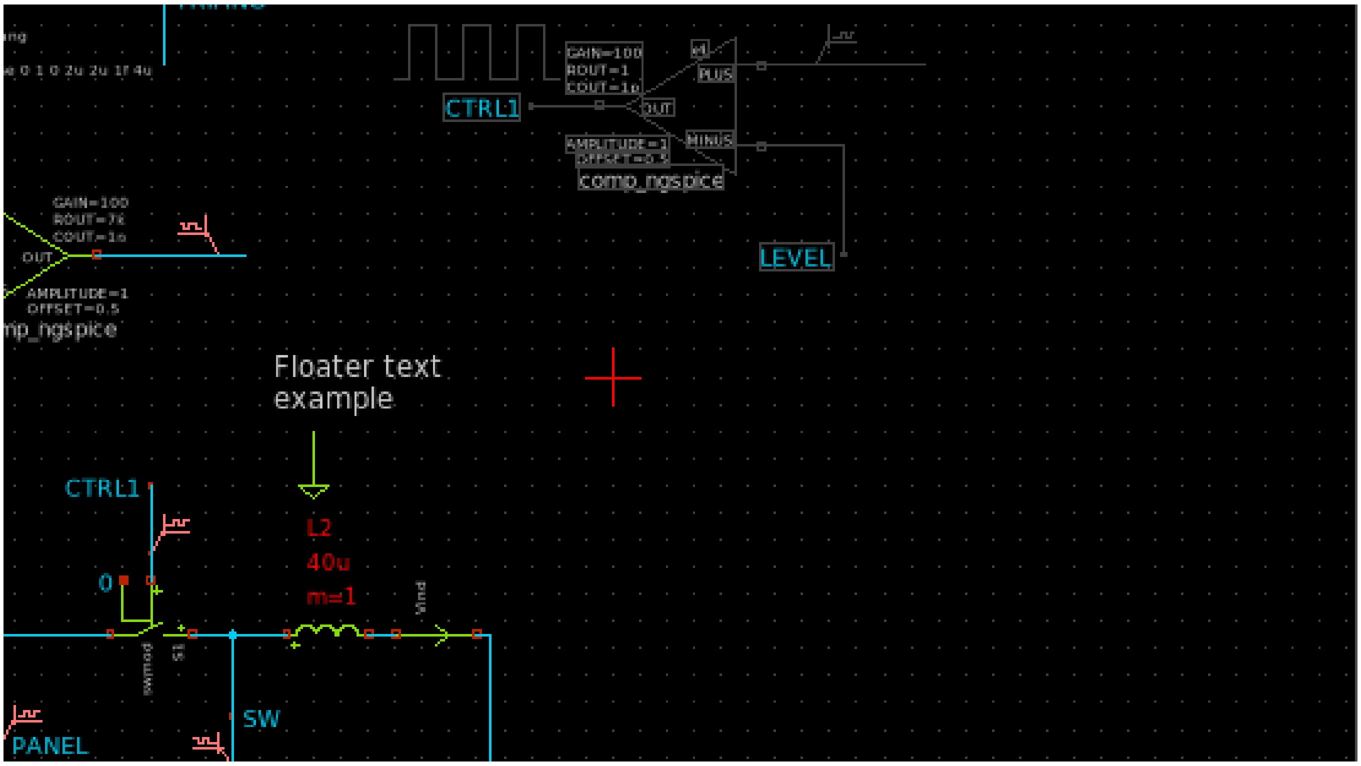


```
# ... We rotate them clockwise around point 1100,-800 (shown with the red cross)
xschem rotate 1100 -800
```



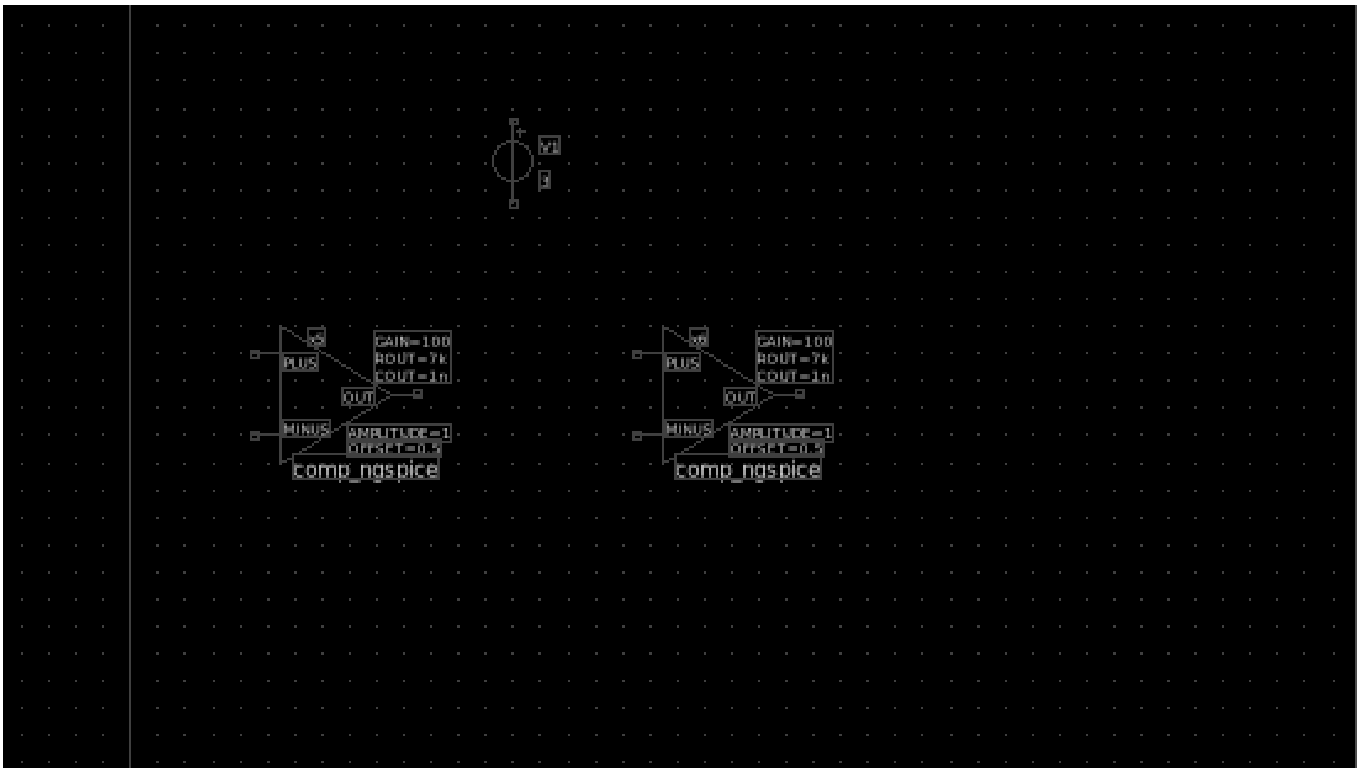
- **Flip a selected portion of the schematic**

```
# After selecting some objects as before ...
# ... We flip them horizontally around point 1100,-800 (shown with the red cross)
xschem flip 1100 -800
```

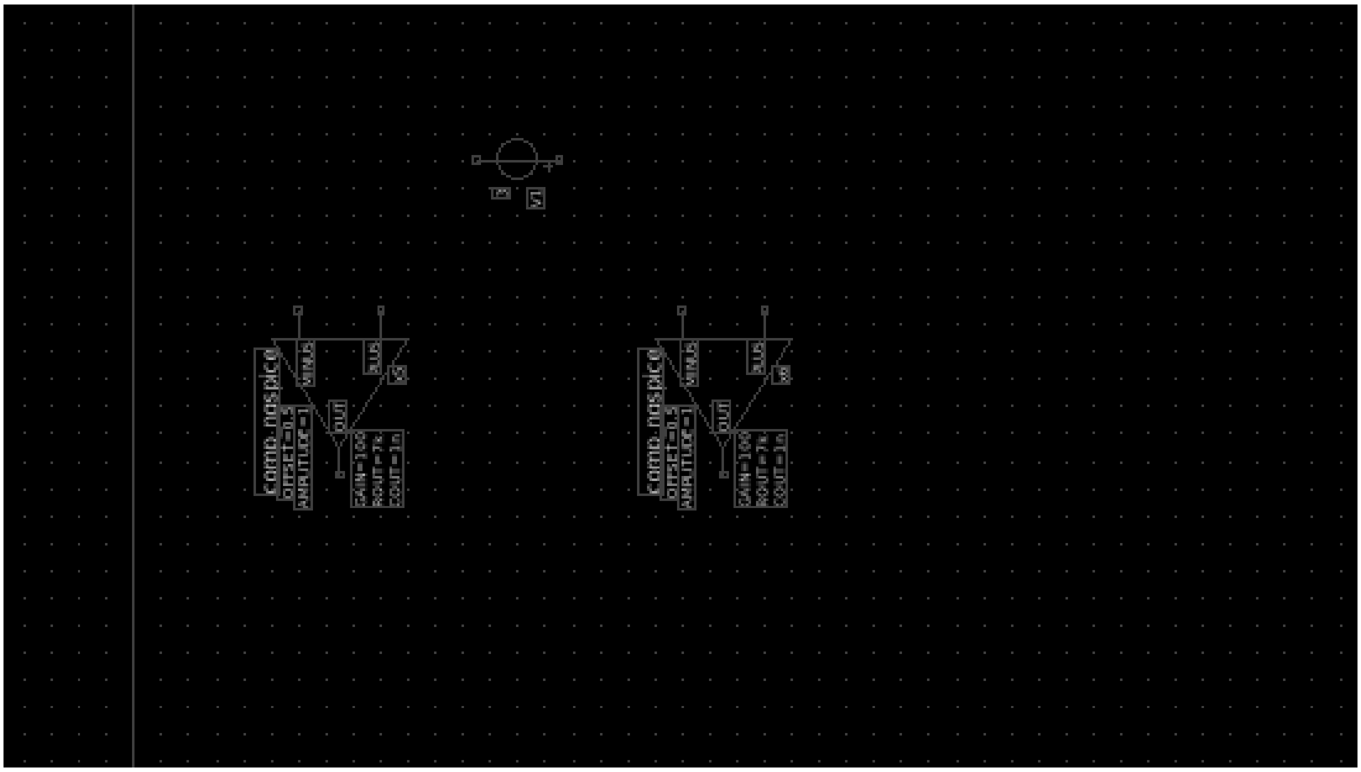


- **Rotate in place a selected portion of the schematic**

```
# After selecting some objects ...
```

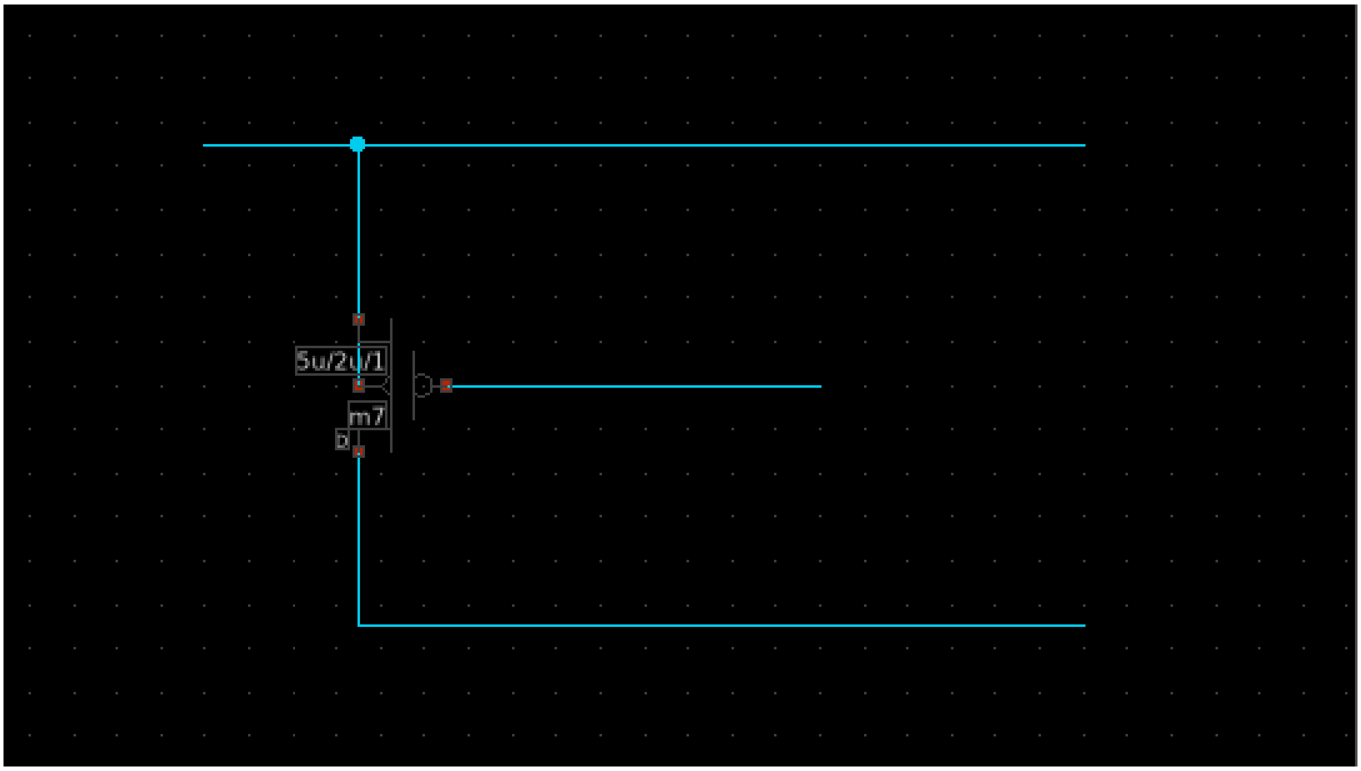


```
# ... We rotate clockwise each object around their origins
# the same command 'flip_in_place' is available for flipping horizontally.
xschem rotate_in_place
```

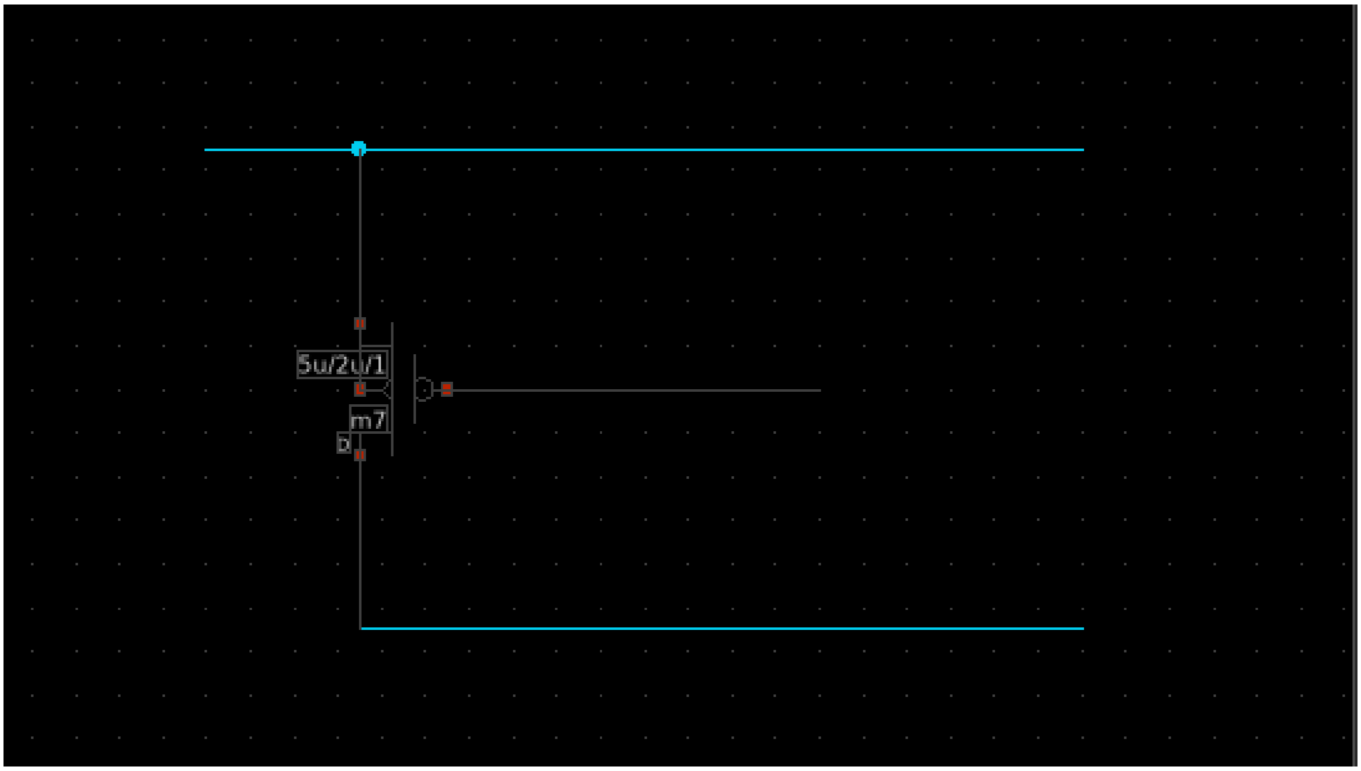


• Move a wired object

```
# After selecting some objects ...
```

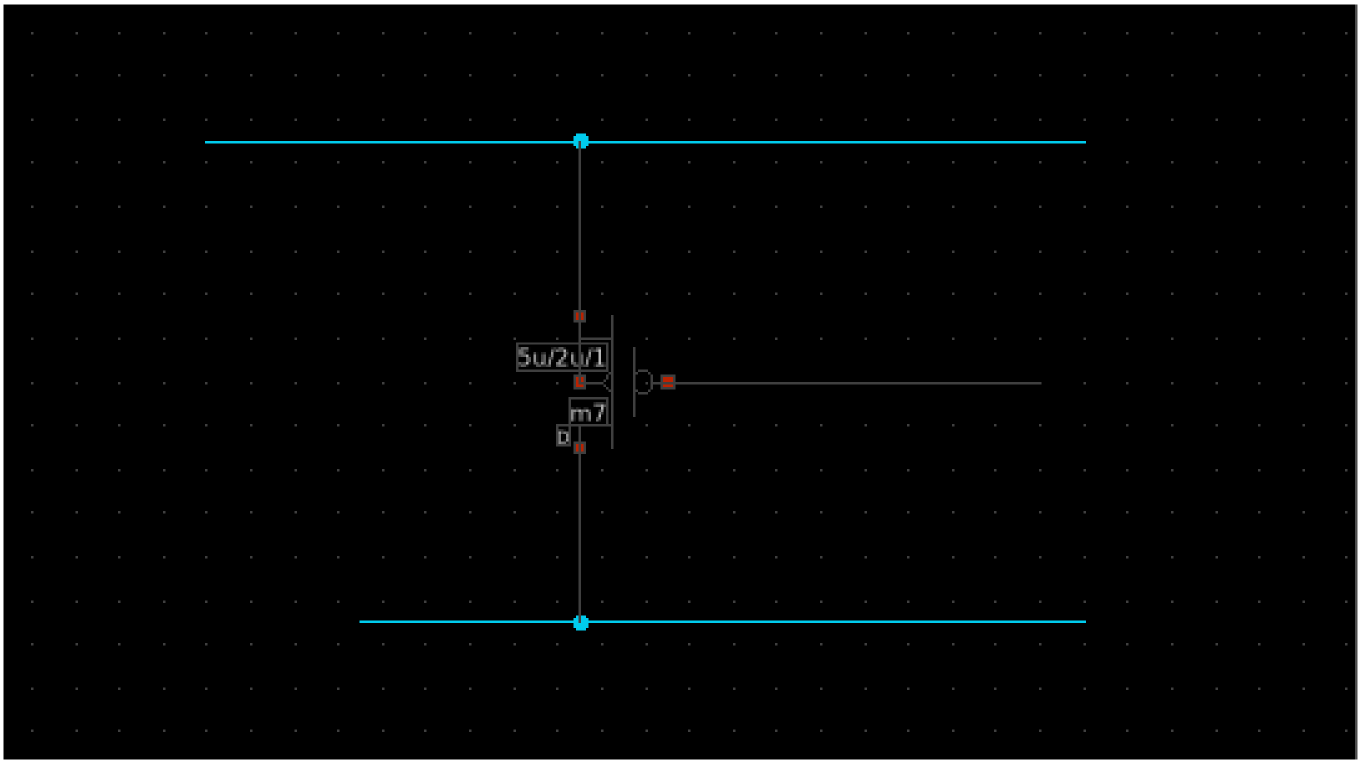


```
# ... we select only the first segments attached to their pins ...
xschem connected_nets 2
```



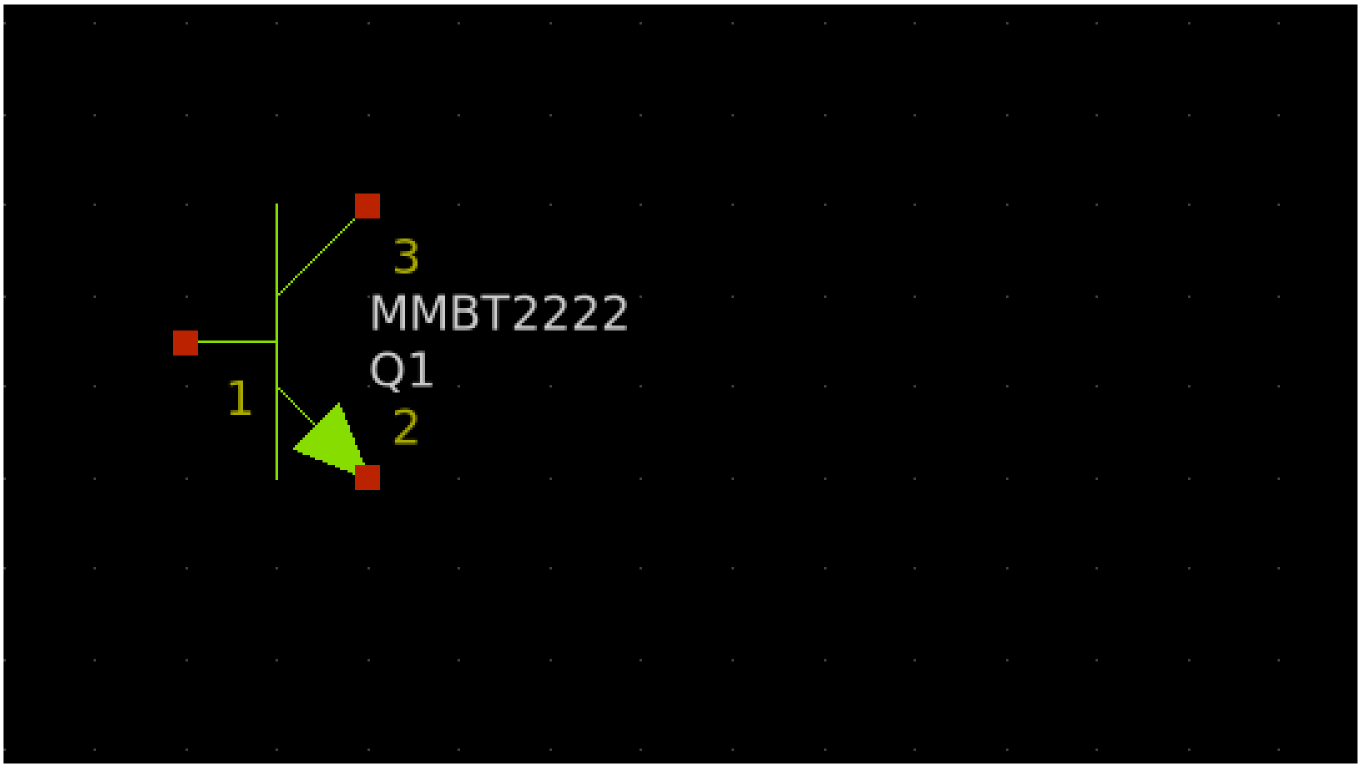
```
# ... And then move the selection.
```

```
xschem move_objects 100 0
```

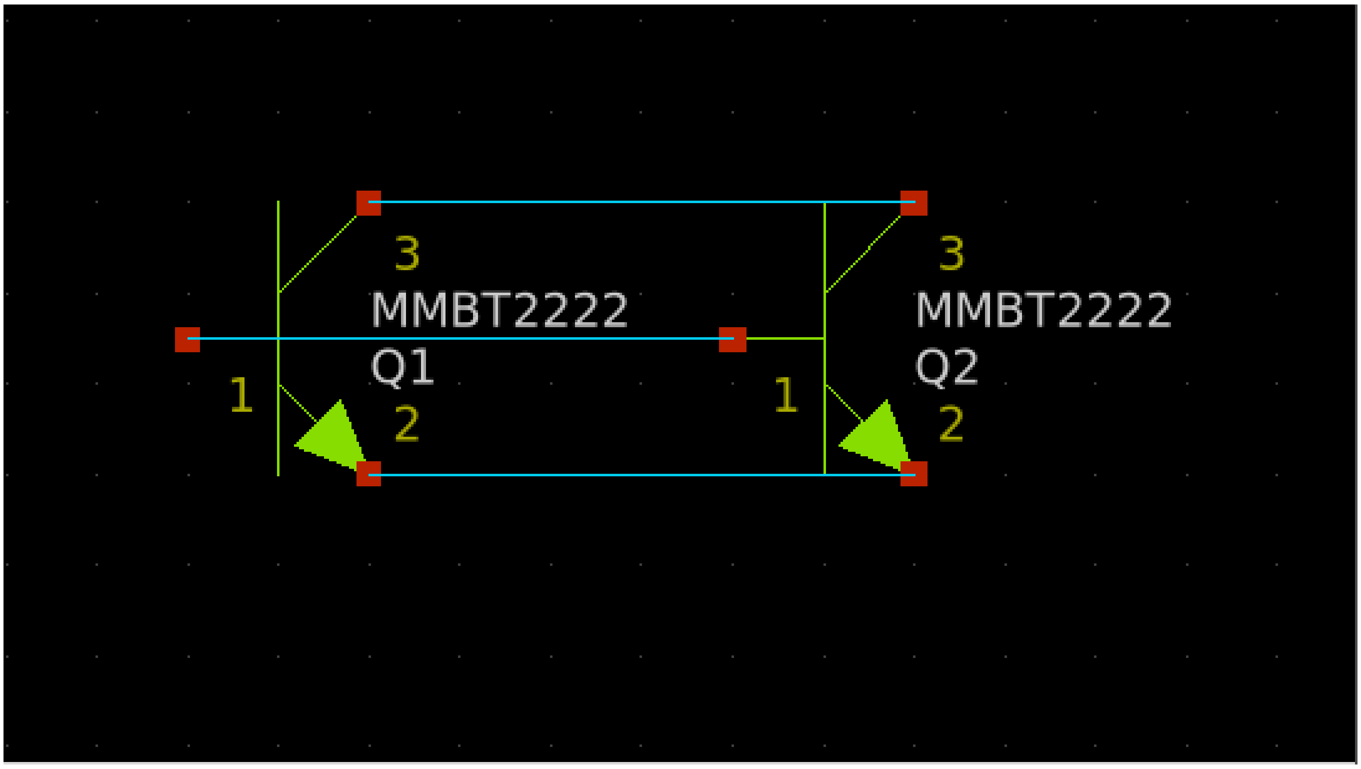


- **Add and wire parallel devices**

```
# Given this instance ...
```

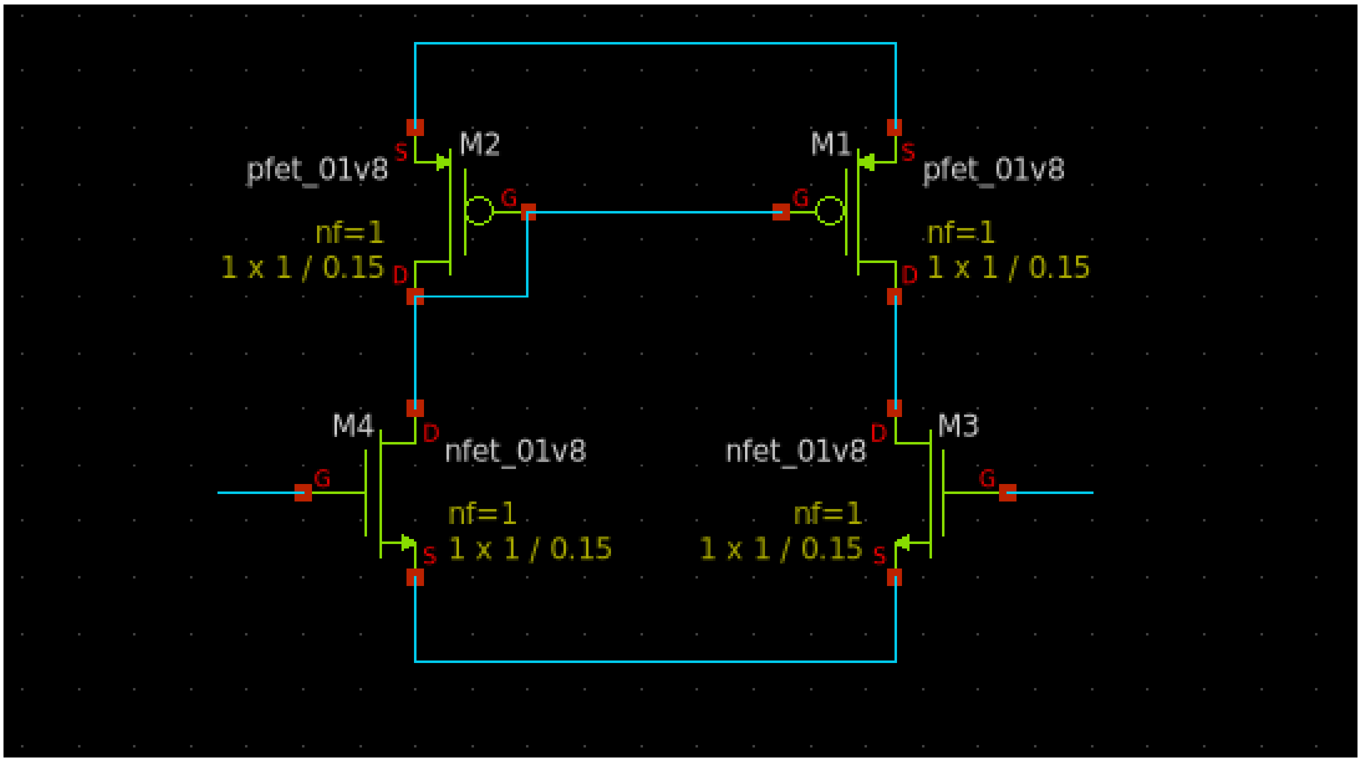


```
# ... The following commands will copy-paste the object and move it
# using the "connect by kissing" feature: when separating connected
#  instances a wire is added.
xschem select instance Q1
xschem copy
xschem paste 0 0
xschem move_objects 120 0 kissing
xschem unselect_all
```

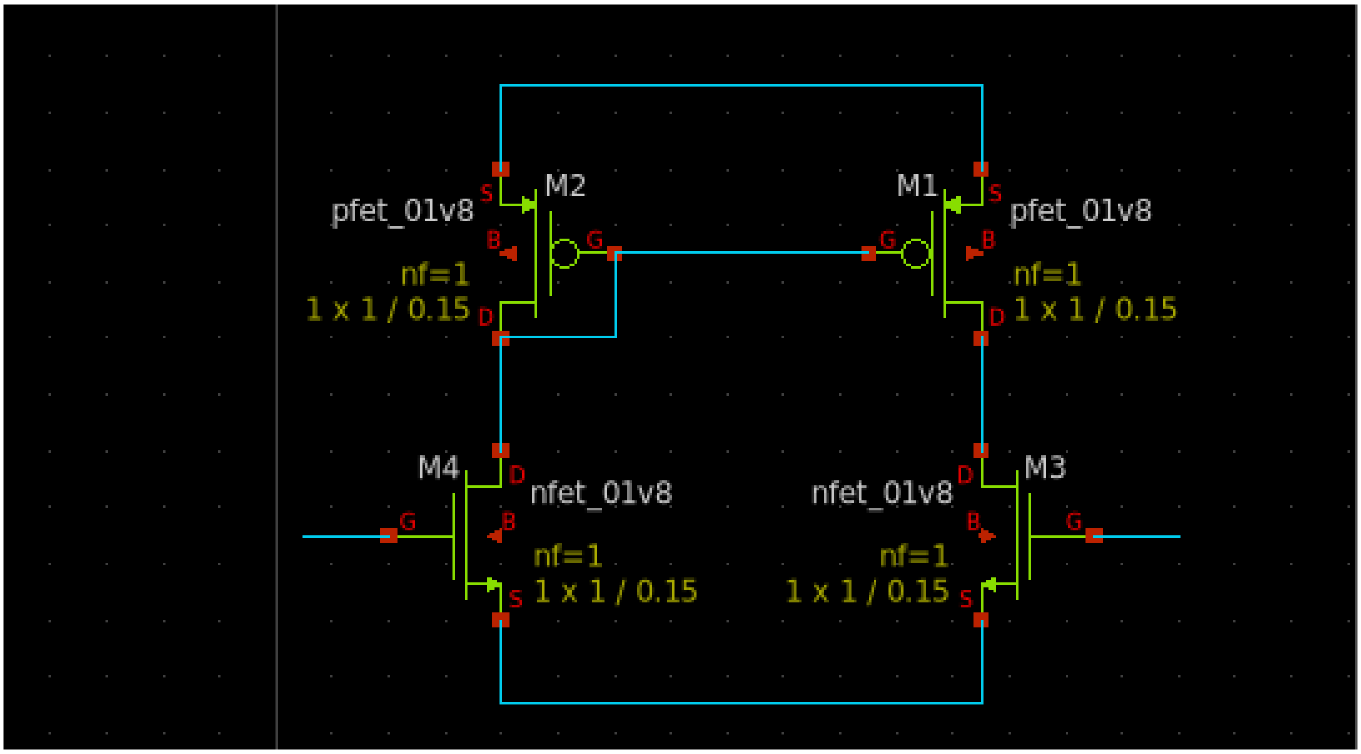


- **Replace symbols**

```
# In the following schematic we want to replace the nfet3/pfet3 with nfet and pfet
# that have the bulk connection pin.
```



```
# select all instances that match "fet_01v8" model name
xschem search regex 1 model {fet_01v8}
set f {}
foreach i [xschem selected_set] {
  # Replace fet3 with fet in symbol reference
  set newname [regsub {fet3} [xschem getprop instance $i cell::name] {fet}]
  xschem replace_symbol $i $newname $f
  # remove body attribute since it is now assigned to the bulk pin
  xschem setprop instance $i body fast
  # the f parameter is for optimzing (avoid pushing undo at each iteration)
  set f fast
}
xschem unselect_all
xschem redraw
```


M2
pfet_01v8
S
B
G
nf=1
1 x 1 / 0.15
D
M1
pfet_01v8
S
G
B
nf=1
1 x 1 / 0.15
D
M4
D
nfet_01v8
G
B
nf=1
1 x 1 / 0.15
S
nfet_01v8
D
M3
B
G
nf=1
1 x 1 / 0.15
S

XSCHEM REMOTE INTERFACE SPECIFICATION

GENERAL INFORMATIONS

XSCHEM embeds a tcl shell, when running xschem the terminal will present a tcl prompt allowing to send commands through it. Most user actions done in the drawing window can be done by sending tcl commands through the tcl shell. A tcp socket can be activated to allow sending remote commands to xschem, for this to work you must the **`xschem_listen_port`** tcl variable in xschemrc, specifying an unused port number. Xschem will listen to this port number for commands and send back results, as if commands were given directly from the tcl console.

XSCHEM implements a TCL **`xschem`** command that accepts additional arguments. This command implements all the XSCHEM remote interface. Of course all Tck-Tk commands are available, for example, if this command is sent to XSCHEM: '**`wm withdraw .`**' the xschem main window will be withdrawn by the window manager, while '**`wm state . normal`**' will show again the window.
This command: '**`puts $XSCHEM_LIBRARY_PATH`**' will print the content of the **`XSCHEM_LIBRARY_PATH`** tcl variable containing the search path.

Handling TCP connection with multiple XSCHEM instances

Since the same TCP port can not be used in more than one process a mechanism is provided to handle multiple xschem processes.
A **`setup_tcp_xschem <port>`** command is provided to set up another TCP port xschem will listen to, freeing the initial port number set in the **`xschem_listen_port`** TCL variable, in the **`xschemrc`** configuration file.

- If port is given and is an unused TCP port it will be used for following TCP communications.
- If port is not given use the port number defined in **`xschem_listen_port`**.
- If port number is given and is equal to **`0`** a free port number will be used.

In all cases the **`xschem_listen_port`** returns the new port number that will be used and set the global **`xschem_listen_port`** variable accordingly.

The following shell script fragment shows the commands to be used to negotiate with xschem another tcp port.
The **`nc`** (netcat) utility is used to pipe the commands to the tcp socket.
When starting xschem a fixed initial port number is always used (2021 by default), so it is always possible to remotely communicate with xschem using this TCP port. Then the following commands can be sent to setup a new port number for further communications, freeing the initial (2021) port number. If another xschem process is started it will again use the initial port number, so no port number collisions occur.

```
# start an xschem instance in background
schippes@asus:~$ xschem -b &
[1] 9342
# negotiate a new port number instead of default 2021
schippes@asus:~$ a=$(echo 'setup_tcp_xschem 0' |nc localhost 2021)
schippes@asus:~$ echo "$a"
34279
# Send a command using the new port number
schippes@asus:~$ b=$(echo 'xschem get current_name' |nc localhost "$a")
schippes@asus:~$ echo "$b"
untitled.sch
```

```
## repeat above steps if you want additional xschem instances each listening to a different free tc
```

TUTORIAL: INSTALL XSCHEM

This short tutorial will illustrate all the steps needed to install XSCHEM on a linux system, getting the files from the SVN repository.

1. Remove all previous xschem related data from old installs, i assume here previous stuff was in **`/usr/local`**, if not change the root prefix accordingly:

   ```
   schippes@mazinga:~$ sudo rm -rf /usr/local/share/xschem/ /usr/local/share/doc/xschem/
   schippes@mazinga:~$ rm -f  ~/xschemrc ~/.xschem/xschemrc
   ```

2. Checkout xschem from the git repository into a build directory (I use xschem_git here):

   ```
   git clone https://github.com/StefanSchippers/xschem.git xschem_git
   ```

3. Configure xschem. In this tutorial we want xschem to be installed in **`/usr/local/bin`**, xschem data installed in **`/usr/local/share/xschem`**, xschem documentation and example circuits installed in **`/usr/local/share/doc/xschem`**, xschem system-wide component symbols installed in **`/usr/local/share/xschem/xschem_library/devices`**, xschem user configuration stored in user's home directory under **`~/.xschem`** and xschem user libraries installed in **`~/.xschem/xschem_library`**:

   ```
   schippes@mazinga:~/xschem_git$ ./configure
   ```

   which sets all default paths, it is equivalent to doing:

   ```
   schippes@mazinga:~/xschem_git$ ./configure --prefix=/usr/local --user-conf-dir=~/.xschem \
   --user-lib-path=~/.xschem/xschem_library \
   --sys-lib-path=/usr/local/share/xschem/xschem_library/devices
   ```

4. If all required libraries, header files and tools that are needed to build xschem are present on the system, the configuration will end with this message (details may vary depending on the host system):

   ```
   ...
   ...
   --- Generating build and config files
   config.h:      ok
   Makefile.conf: ok
   src/Makefile:  ok


   =====================
   Configuration summary
   =====================

   Compilation:
   ```

```
         CC:        gcc
         debug:     no
         profiling: no

        Paths:
         prefix:        /usr/local
         user-conf-dir: ~/.xschem
         user-lib-path: ~/share/xschem/xschem_library
         sys-lib-path:  /usr/local/share/xschem/xschem_library/devices

        Libs & features:
         tcl:       -ltcl8.6
         tk:        -ltcl8.6  -ltk8.6
         cairo:     yes
         xrender:   yes
         xcb:       yes

        Configuration complete, ready to compile.

        schippes@mazinga:~/xschem_git$
```

5. Build xschem by running 'make'

```
        schippes@mazinga:~/xschem_git$ make
```

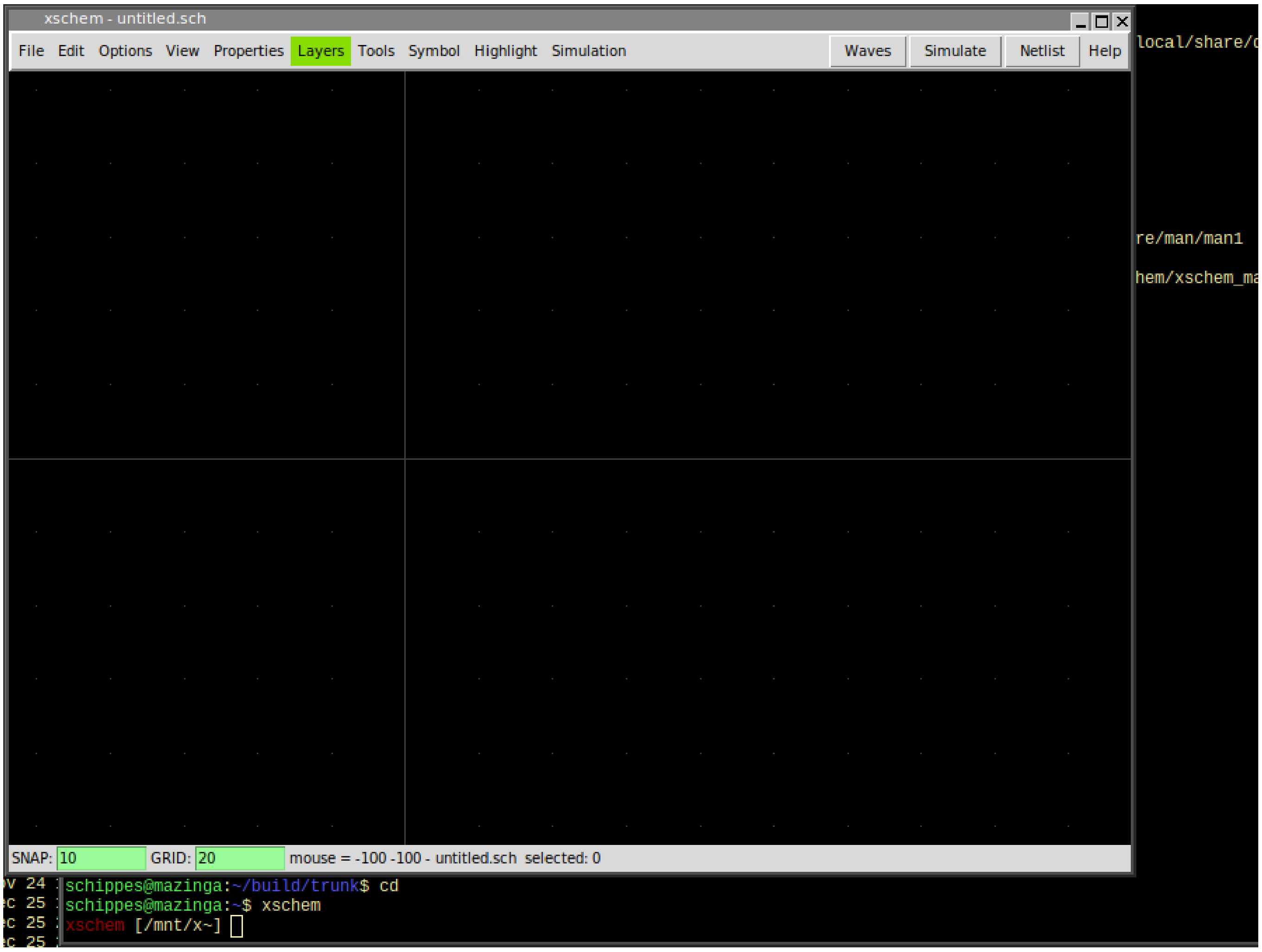
6. If compilation of source files completed with no errors xschem will be ready for installation:

```
        schippes@mazinga:~/xschem_git$ sudo make install
```

   Note that since we are installing in /usr/local we need root rights (sudo) for doing the installation.

7. Test xschem by launching 'xschem' from the terminal:

```
        schippes@mazinga:~/xschem_git$ cd
        schippes@mazinga:~$ xschem
```


xschem - untitled.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
SNAP: 10 GRID: 20 mouse = -100 -100 - untitled.sch selected: 0

```
local/share/c
re/man/man1
hem/xschem_ma
v 24 schippes@mazinga:~/build/trunk$ cd
c 25 schippes@mazinga:~$ xschem
c 25 xschem [/mnt/x~]
c 25
```

if /usr/local/bin is not in your PATH variable use the full xschem path:

```
schippes@mazinga:~$ /usr/local/bin/xschem
```

8. Close xschem (menu **`File - Exit`**)

9. Copy the xschemrc file in the **`trunk/src`** directory to the **`~/.xschem`** directory. If **`~/.xschem`** does not exist create it with **`mkdir ~/.xschem`**

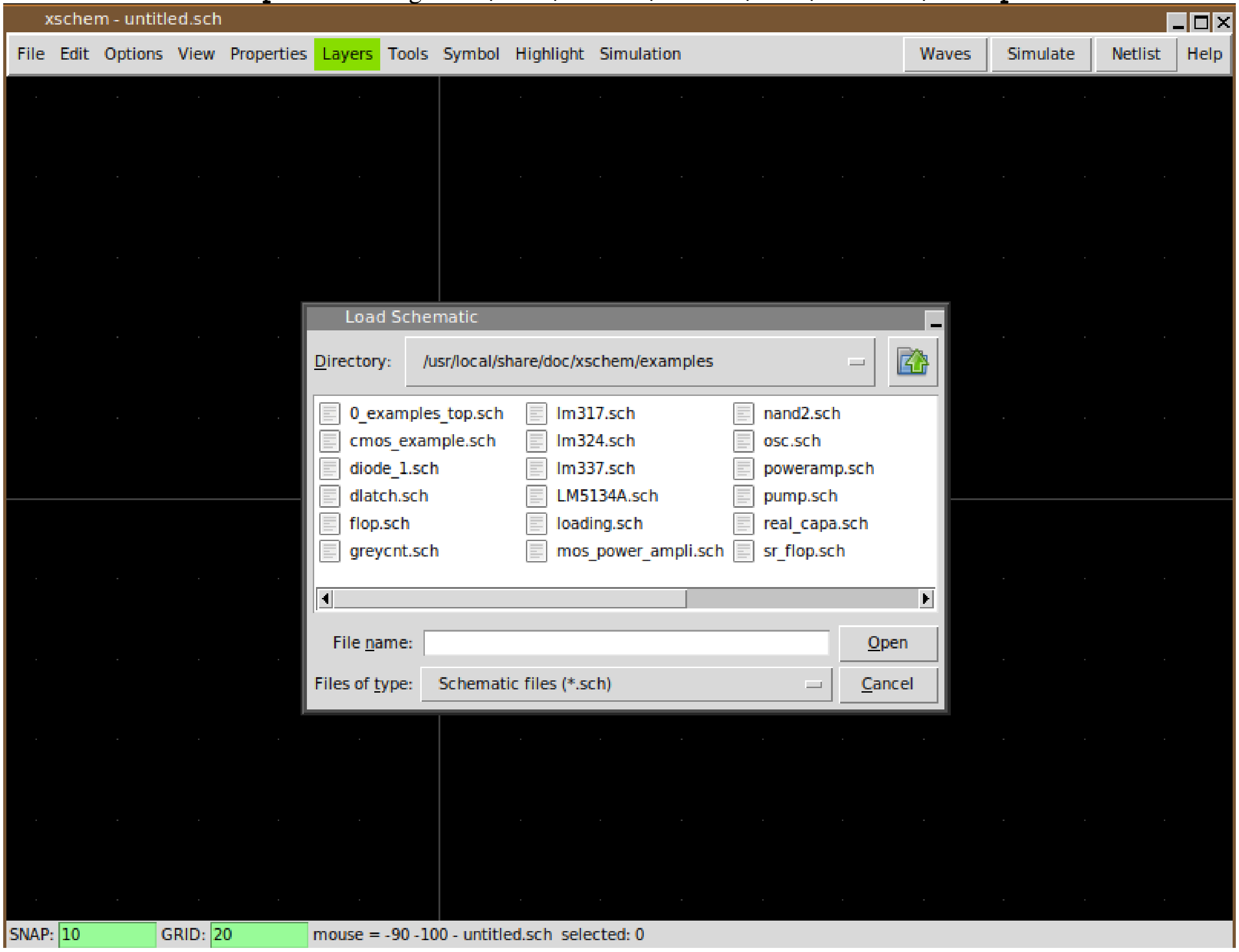
```
schippes@mazinga:~$ cp build/trunk/src/xschemrc ~/.xschem
```

The **`~/.xschem/xschemrc`** is the user xschem configuration file. You may change it later to change xschem defaults or add / remove / change component and schematic directories. For first tests it is recommended to leave xschemrc as it is.

10. Run xschem again to try some schematic load tests:

```
schippes@mazinga:~$ xschem
```

11. Select menu **File - Open** and navigate to **/usr/local/share/doc/xschem/examples**:


xschem - untitled.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
Load Schematic
Directory:
File name:
Open
Files of type:
Cancel

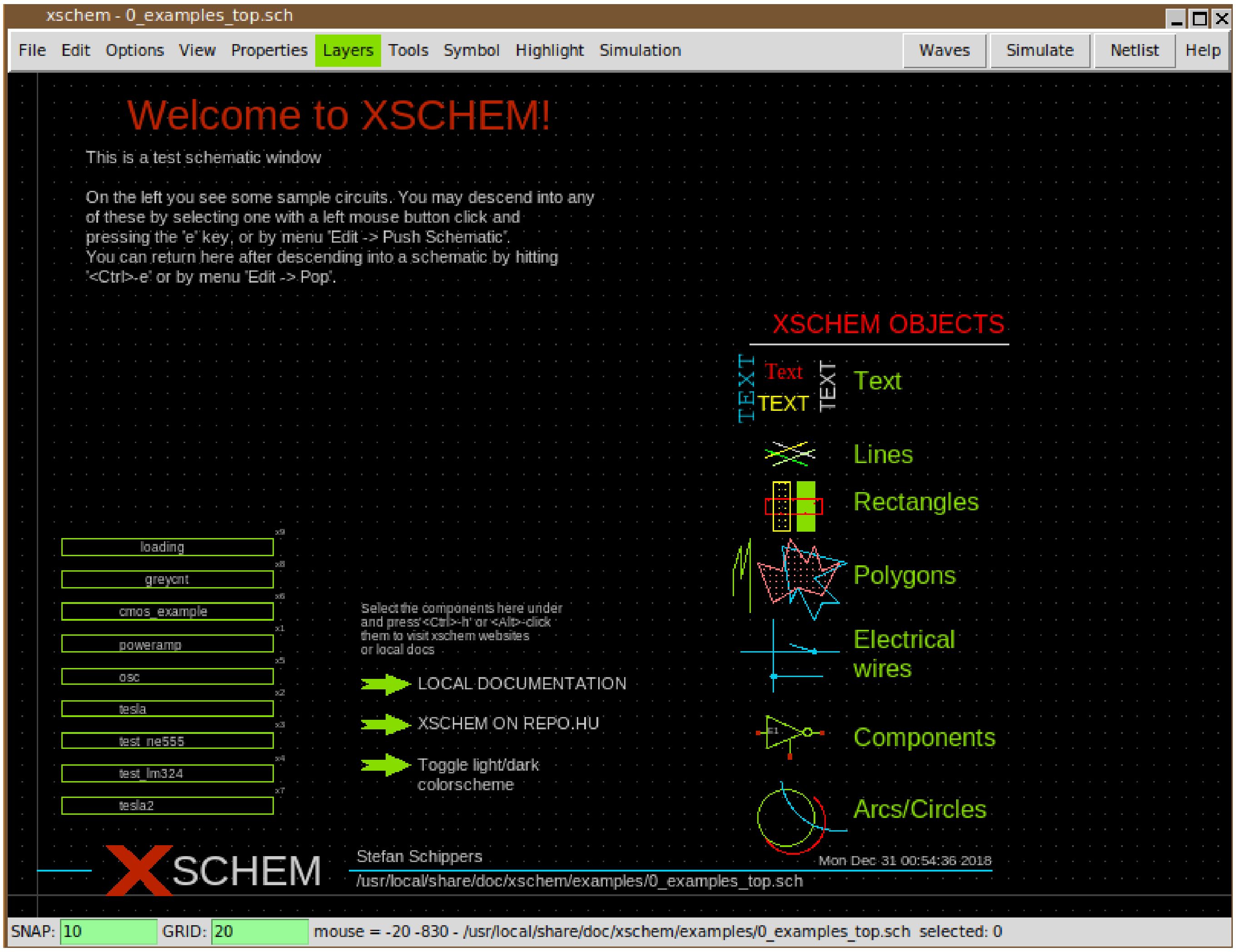
Directory: /usr/local/share/doc/xschem/examples

```
0_examples_top.sch    lm317.sch             nand2.sch
cmos_example.sch      lm324.sch             osc.sch
diode_1.sch           lm337.sch             poweramp.sch
dlatch.sch            LM5134A.sch           pump.sch
flop.sch              loading.sch           real_capa.sch
greycnt.sch           mos_power_ampli.sch   sr_flop.sch
```

Files of type: Schematic files (*.sch)

SNAP: 10 GRID: 20 mouse = -90 -100 - untitled.sch selected: 0

12. Select **0_examples_top.sch** and press 'OK':



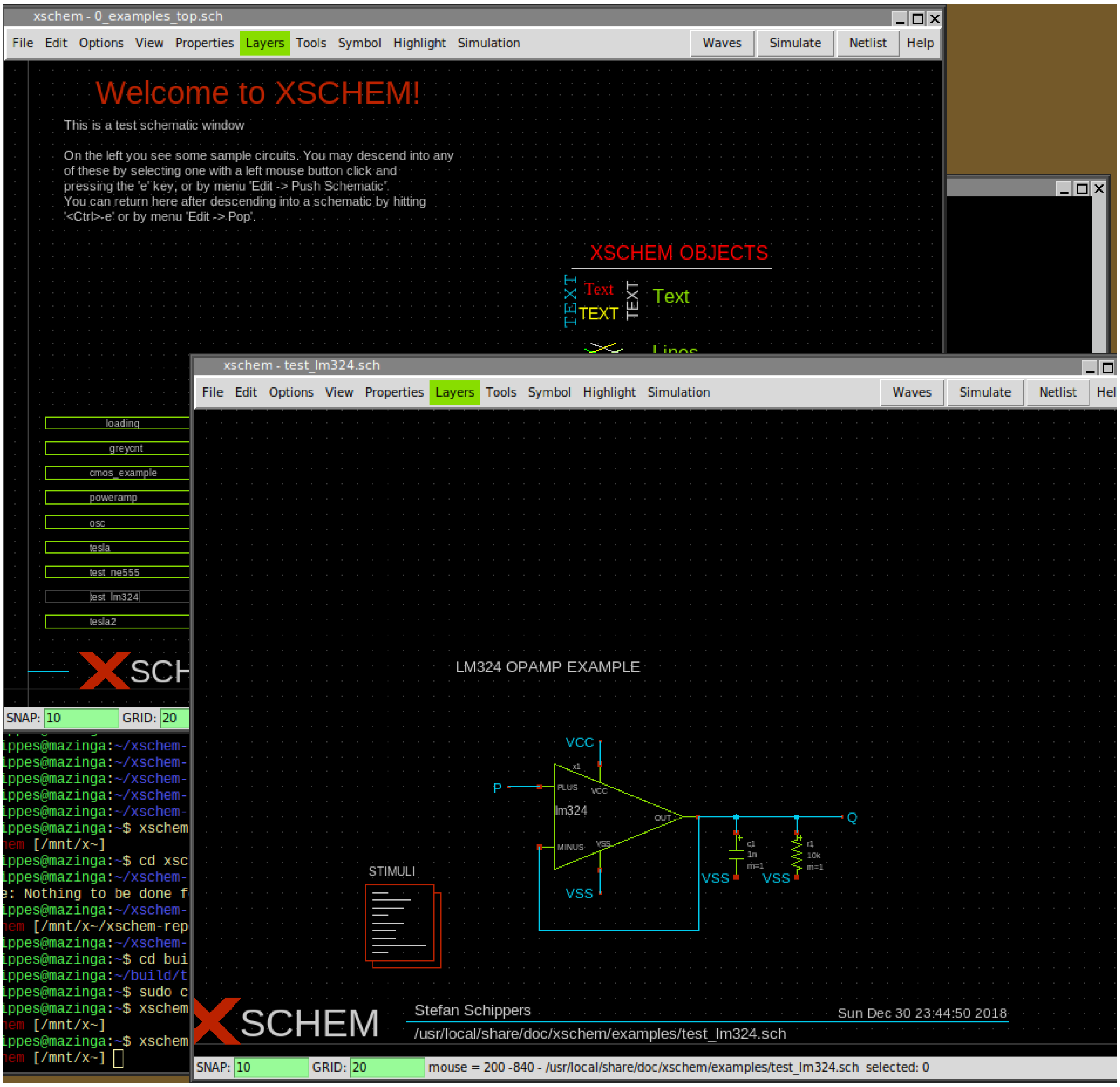
Welcome to XSCHEM!

This is a test schematic window

On the left you see some sample circuits. You may descend into any of these by selecting one with a left mouse button click and pressing the 'e' key, or by menu 'Edit -> Push Schematic'. You can return here after descending into a schematic by hitting '<Ctrl>-e' or by menu 'Edit -> Pop'.

Select the components here under and press '<Ctrl>-h' or <Alt>-click them to visit xschem websites or local docs

13. This schematic contains a set of sub-schematics. Select one of them by clicking it with the left mouse button (test_lm324 in this example) and press the **`Alt-e`** key combination: another xschem window will be opened with the schematic view of the selected symbol:


xschem - 0_examples_top.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
XSCHEM OBJECTS
TEXT Text TEXT Text
Lines
loading
greycnt
cmos_example
poweramp
osc
tesla
test_ne555
test_lm324
tesla2
XSCH
xschem - test_lm324.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Hel
LM324 OPAMP EXAMPLE
VCC
P
PLUS
VCC
lm324
OUT
MINUS
VSS
Q
c1 1n m=1
r1 10k m=1
VSS
VSS
VSS
STIMULI
XSCHEM
Stefan Schippers
Sun Dec 30 23:44:50 2018
/usr/local/share/doc/xschem/examples/test_lm324.sch

Welcome to XSCHEM!

This is a test schematic window

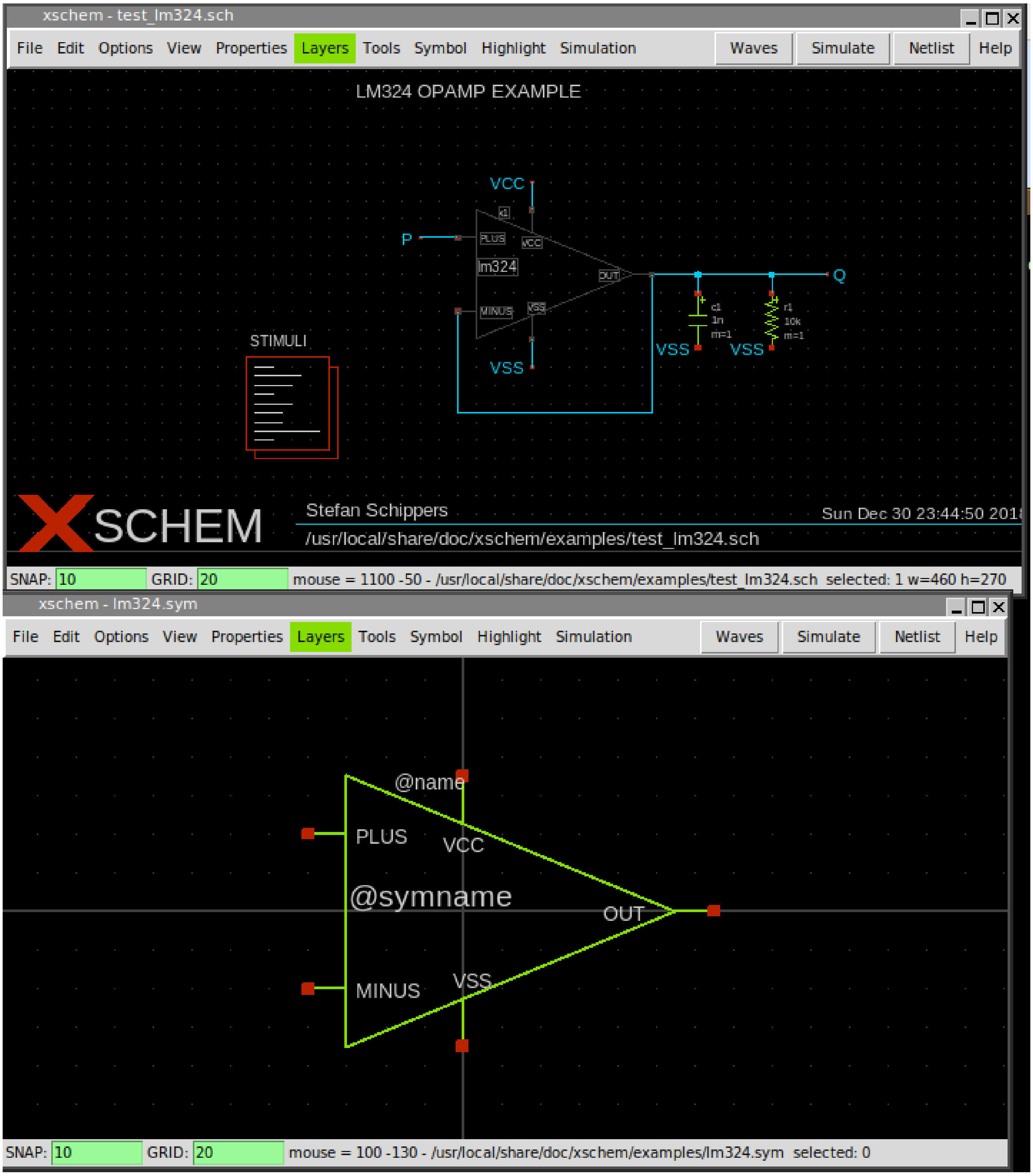
On the left you see some sample circuits. You may descend into any of these by selecting one with a left mouse button click and pressing the 'e' key, or by menu 'Edit -> Push Schematic'.
You can return here after descending into a schematic by hitting '<Ctrl>-e' or by menu 'Edit -> Pop'.

SNAP: 10 GRID: 20

```
ippes@mazinga:~/xschem-
ippes@mazinga:~/xschem-
ippes@mazinga:~/xschem-
ippes@mazinga:~/xschem-
ippes@mazinga:~/xschem-
ippes@mazinga:~$ xschem
hem [/mnt/x~]
ippes@mazinga:~$ cd xsc
ippes@mazinga:~/xschem-
e: Nothing to be done f
ippes@mazinga:~/xschem-
hem [/mnt/x~/xschem-rep
ippes@mazinga:~/xschem-
ippes@mazinga:~$ cd bui
ippes@mazinga:~/build/t
ippes@mazinga:~$ sudo c
ippes@mazinga:~$ xschem
hem [/mnt/x~]
ippes@mazinga:~$ xschem
hem [/mnt/x~] 
```

SNAP: 10 GRID: 20 mouse = 200 -840 - /usr/local/share/doc/xschem/examples/test_lm324.sch selected: 0

14. Click on the lm324 symbol, it can now be edited using the **`Alt-i`** key combination:


xschem - test_lm324.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
LM324 OPAMP EXAMPLE
VCC
P
Q
lm324
PLUS
MINUS
OUT
VSS
c1
1n
m=1
r1
10k
m=1
STIMULI
SCHEM
Stefan Schippers
Sun Dec 30 23:44:50 2018
/usr/local/share/doc/xschem/examples/test_lm324.sch
xschem - lm324.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
@name
PLUS
VCC
@symname
OUT
MINUS
VSS

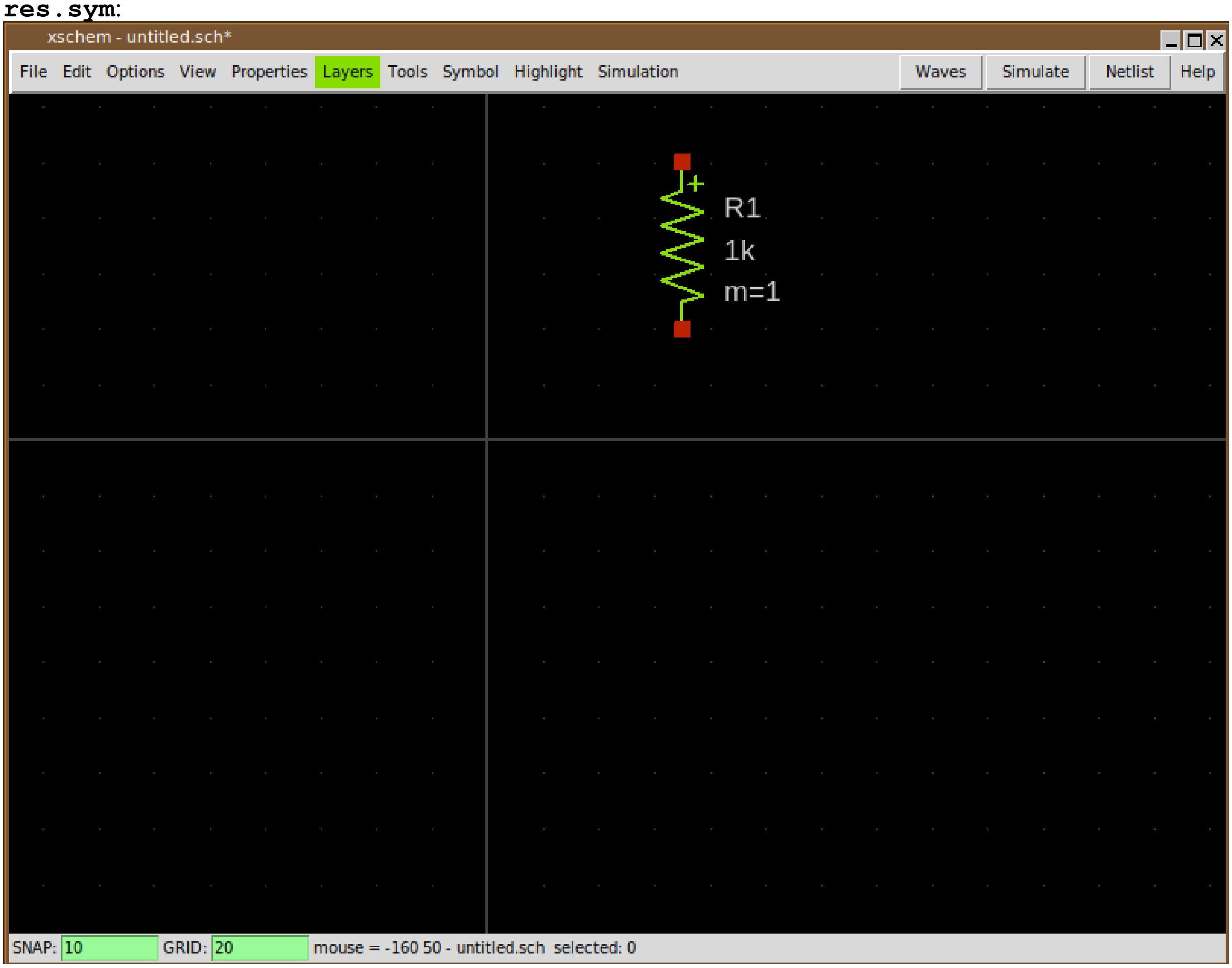
SNAP: 10 GRID: 20 mouse = 1100 -50 - /usr/local/share/doc/xschem/examples/test_lm324.sch selected: 1 w=460 h=270

SNAP: 10 GRID: 20 mouse = 100 -130 - /usr/local/share/doc/xschem/examples/lm324.sym selected: 0

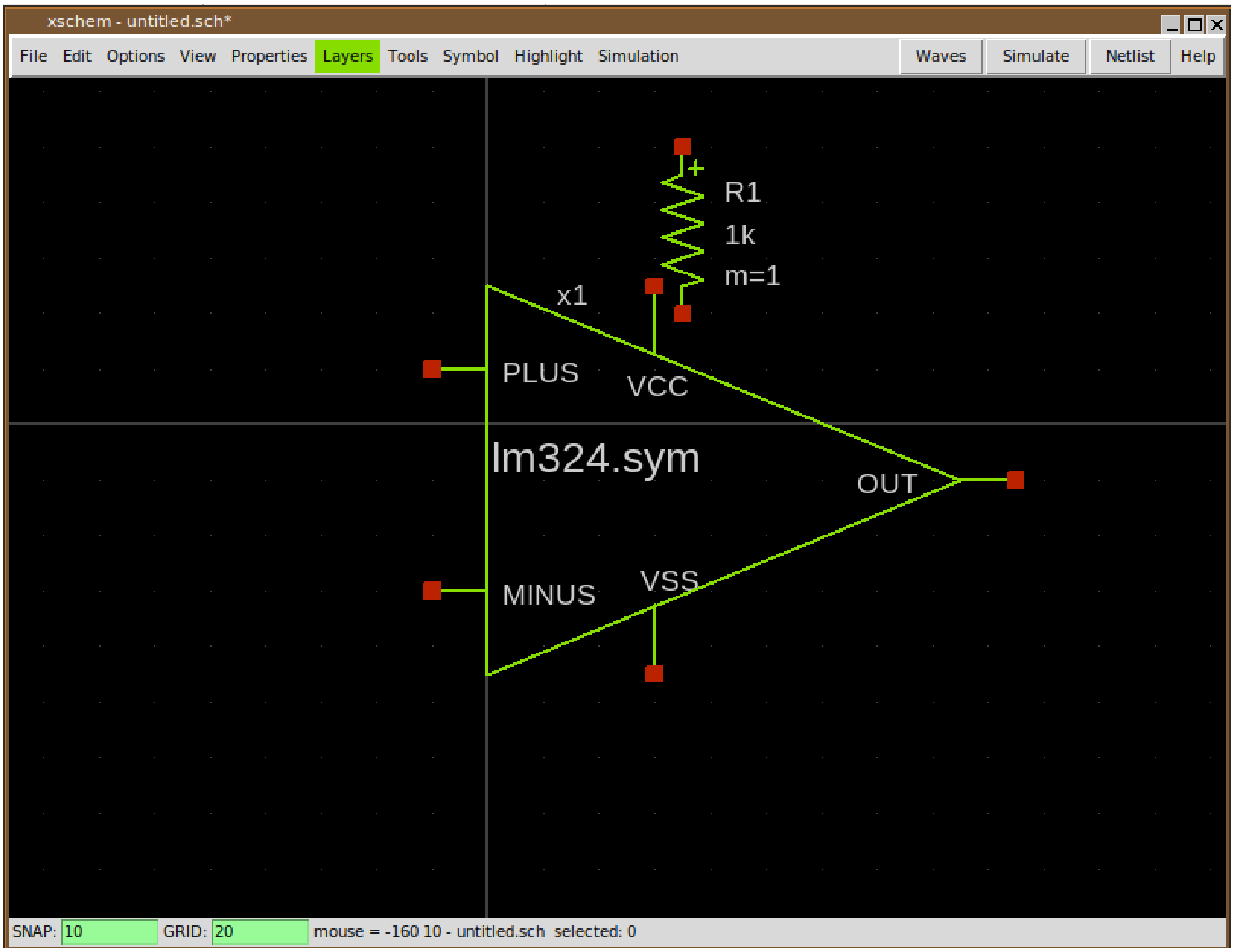
15. Now close all xschem windows and restart a new xschem instance from terminal:

```
schippes@mazinga:~$ xschem
```

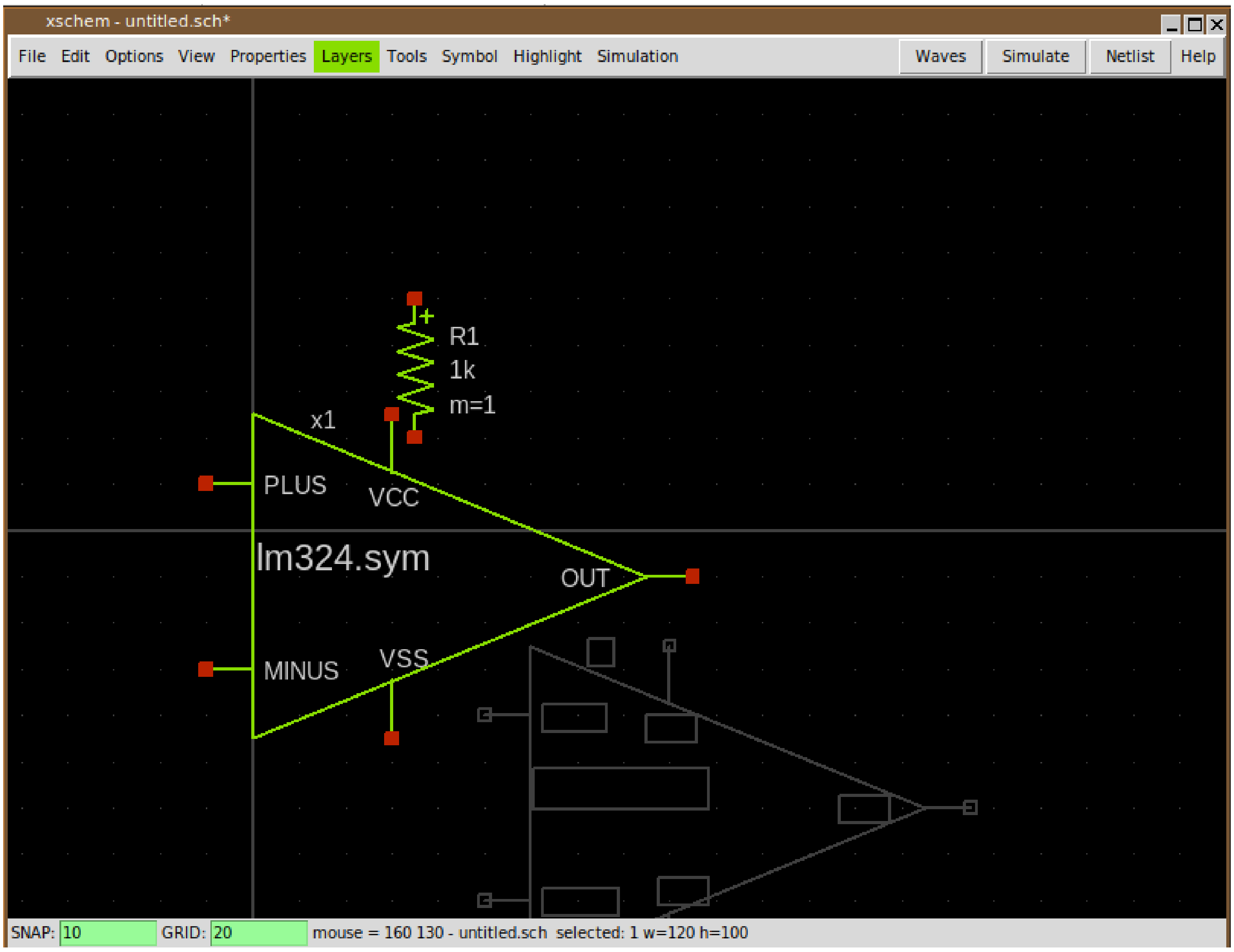
16. We want to create a simple circuit in this empty schematic window: press the **`Insert`** key (this is used to place components) in the file selector navigate to **`/usr/local/share/xschem/xschem_library`** and select **`res.sym`**:



17. Lets add another component: press **`Insert`** key again and navigate to **`/usr/local/share/doc/xschem/examples`** and select **`lm324.sym`**:

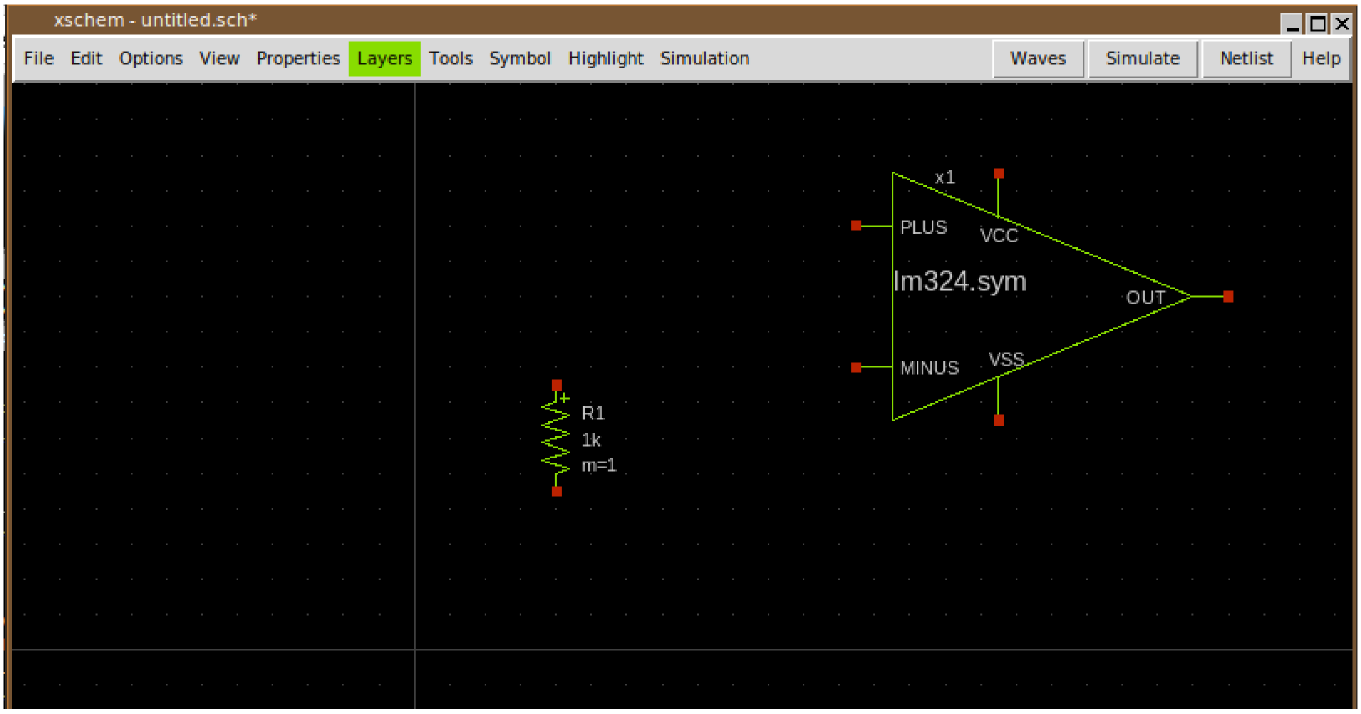


18. Select (click on it) the lm324 symbol and move it by pressing the `m` key:

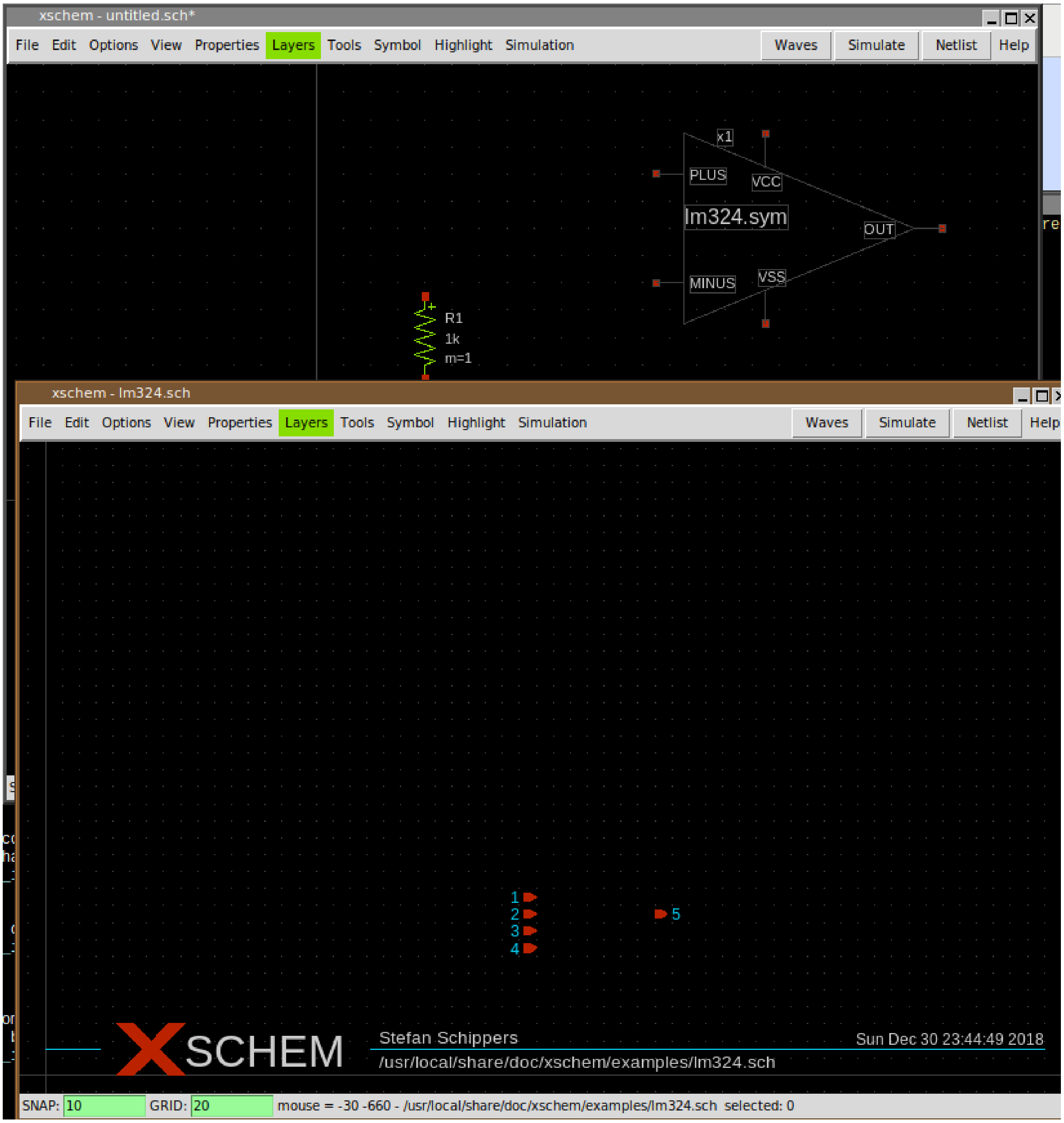

xschem - untitled.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
R1
1k
m=1
x1
PLUS
VCC
lm324.sym
OUT
MINUS
VSS
SNAP: 10 GRID: 20

mouse = 160 130 - untitled.sch selected: 1 w=120 h=100

19. Place the lm324 component where you want in the schematic by placing the mouse and clicking the left button:

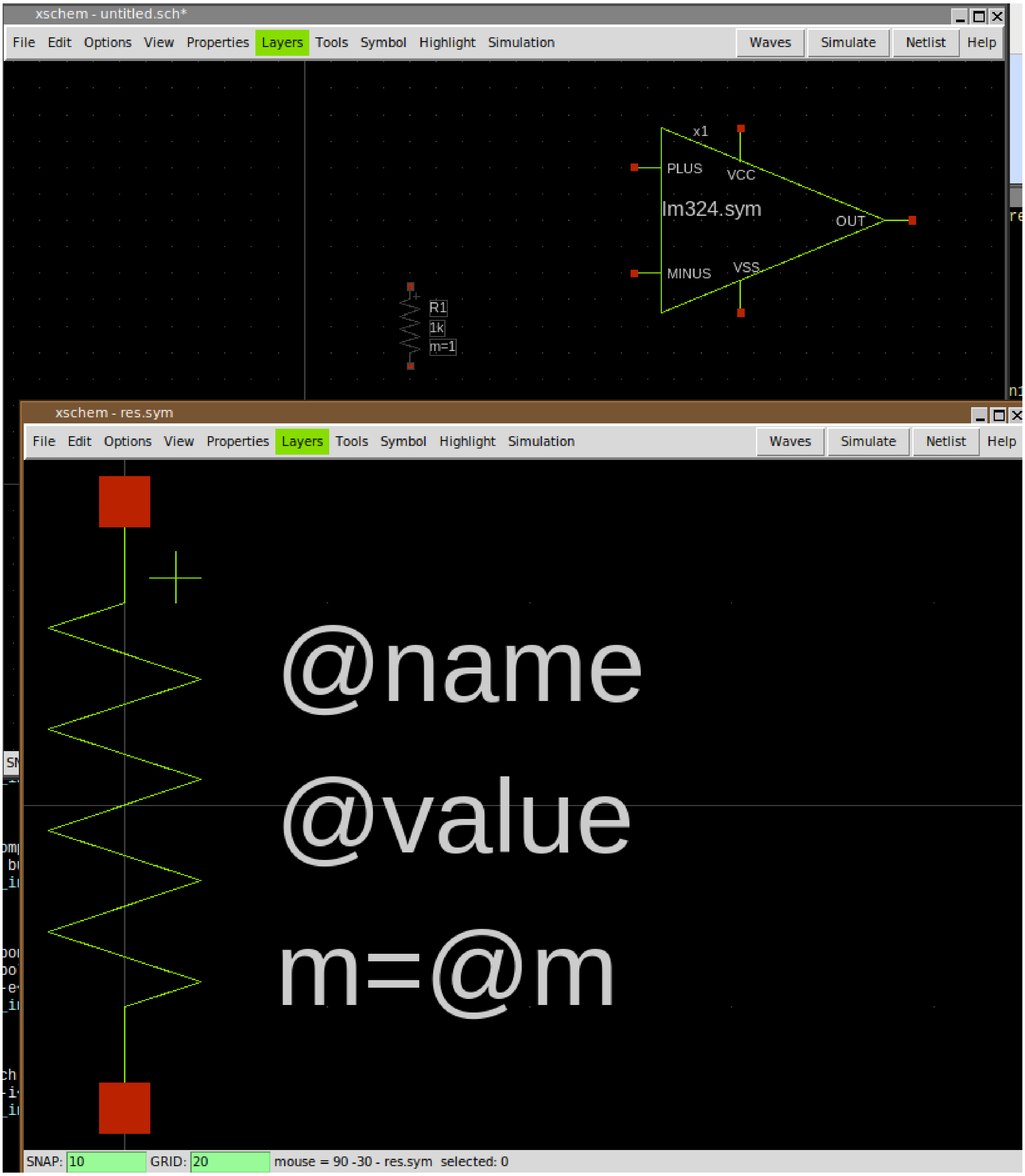


20. The lm324.sym component has a schematic (**.sch**) representation, while the resistor is a primitive, it has only a symbol view (**.sym**). you can see the schematic of the lm324 by selecting it and pressing **Alt-e**:


xschem - untitled.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
x1
PLUS
VCC
lm324.sym
OUT
MINUS
VSS
R1
1k
m=1
xschem - lm324.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
1
2
3
4
5
XSCHEM
Stefan Schippers
Sun Dec 30 23:44:49 2018
/usr/local/share/doc/xschem/examples/lm324.sch
SNAP: 10 GRID: 20 mouse = -30 -660 - /usr/local/share/doc/xschem/examples/lm324.sch selected: 0

21. Close the lm324.sch window and view the symbol view of the resistor by selecting it and pressing **`Alt-i`**:

e tutorial, if all the steps were successful there is a good probability that xschem is correctly installed on your system.


xschem - untitled.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
x1
PLUS
VCC
lm324.sym
OUT
MINUS
VSS
R1
1k
m=1
xschem - res.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@name
@value
m=@m
SNAP: 10 GRID: 20 mouse = 90 -30 - res.sym selected: 0

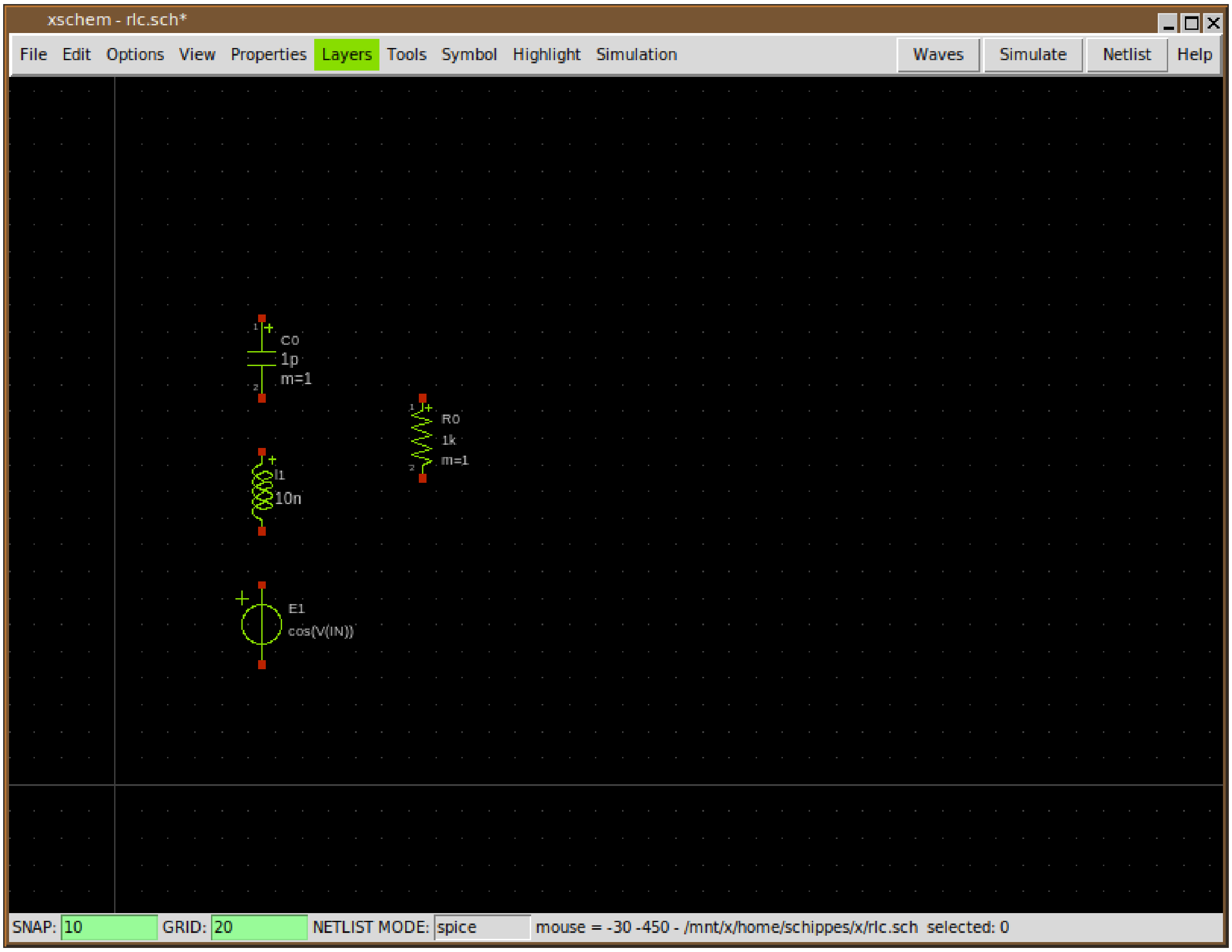
This concludes the tutorial, if all the steps were successful there is a good probability that xschem is correctly installed on your system.

UP

TUTORIAL: RUN A SIMULATION WITH XSCHEM

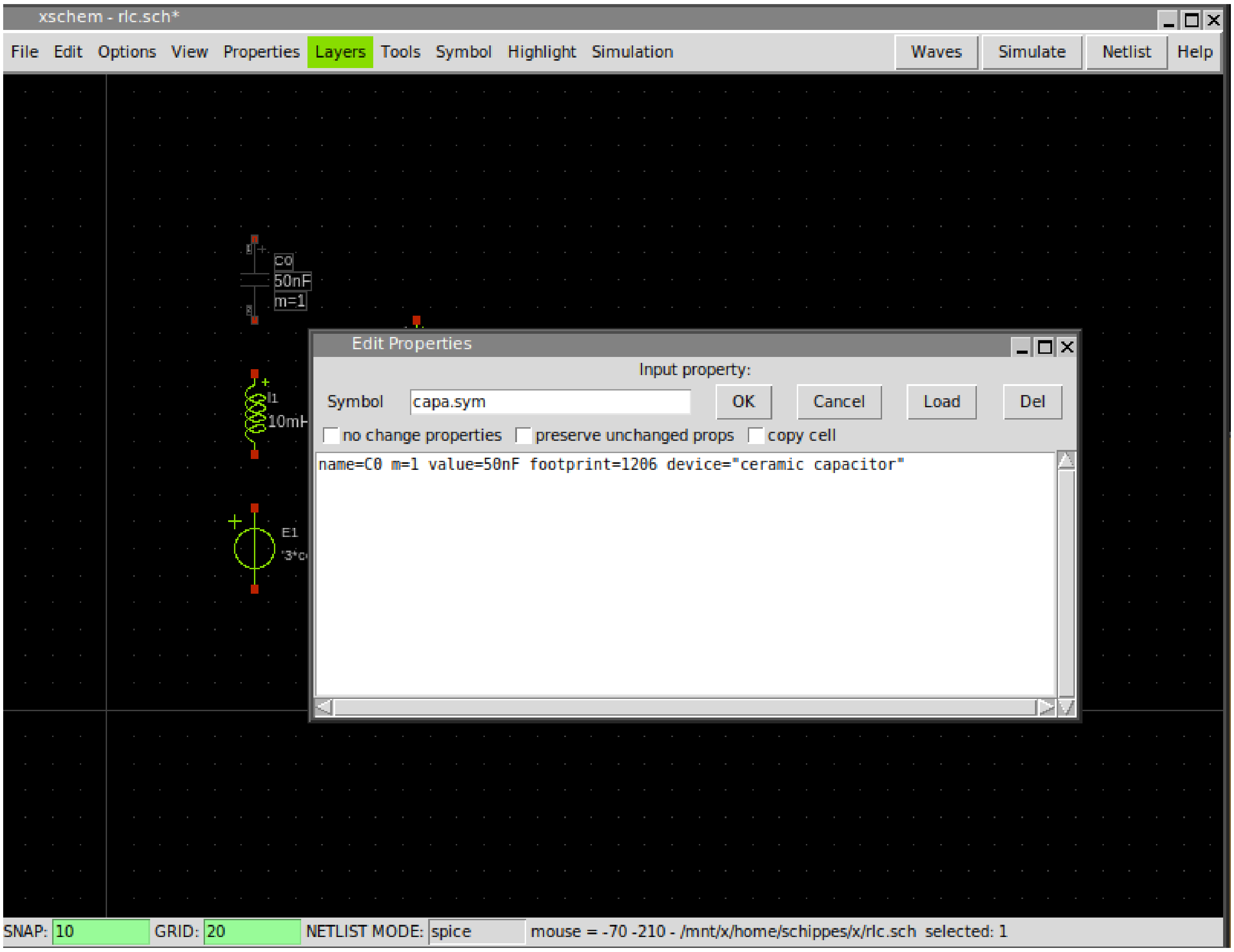
here some instructions to create a schematic and run a ngspice transient sim in XSCHEM:

1. Build and install xschem from svn head.
2. Create some empty directory (in my examples i use ~/x)
3. cd ~/x
4. ~/bin/xschem rlc.sch (use the actual xschem install path). xschem will warn you that the rlc.sch file does not exist. No problem.
5. Press Insert key
6. Navigate in the file selector to .../share/xschem/xschem_library/devices
7. Select 'capa.sym' and press 'Open'
8. Select the capacitor, press 'm' and place it somewhere
9. Press 'Insert' again and place 'res.sym' and then again 'ind.sym'
10. Again, press 'Insert' and place 'vsource_arith.sym'
11. By selecting (left btn click) and moving ('m') place the components like in this picture:


xschem - rlc.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
C0
1p
m=1
R0
1k
m=1
l1
10n
E1
cos(V(IN))

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -30 -450 - /mnt/x/home/schippes/x/rlc.sch selected: 0

12. Press the right mouse button on the capacitor and set its 'value=' attribute to 50nF:

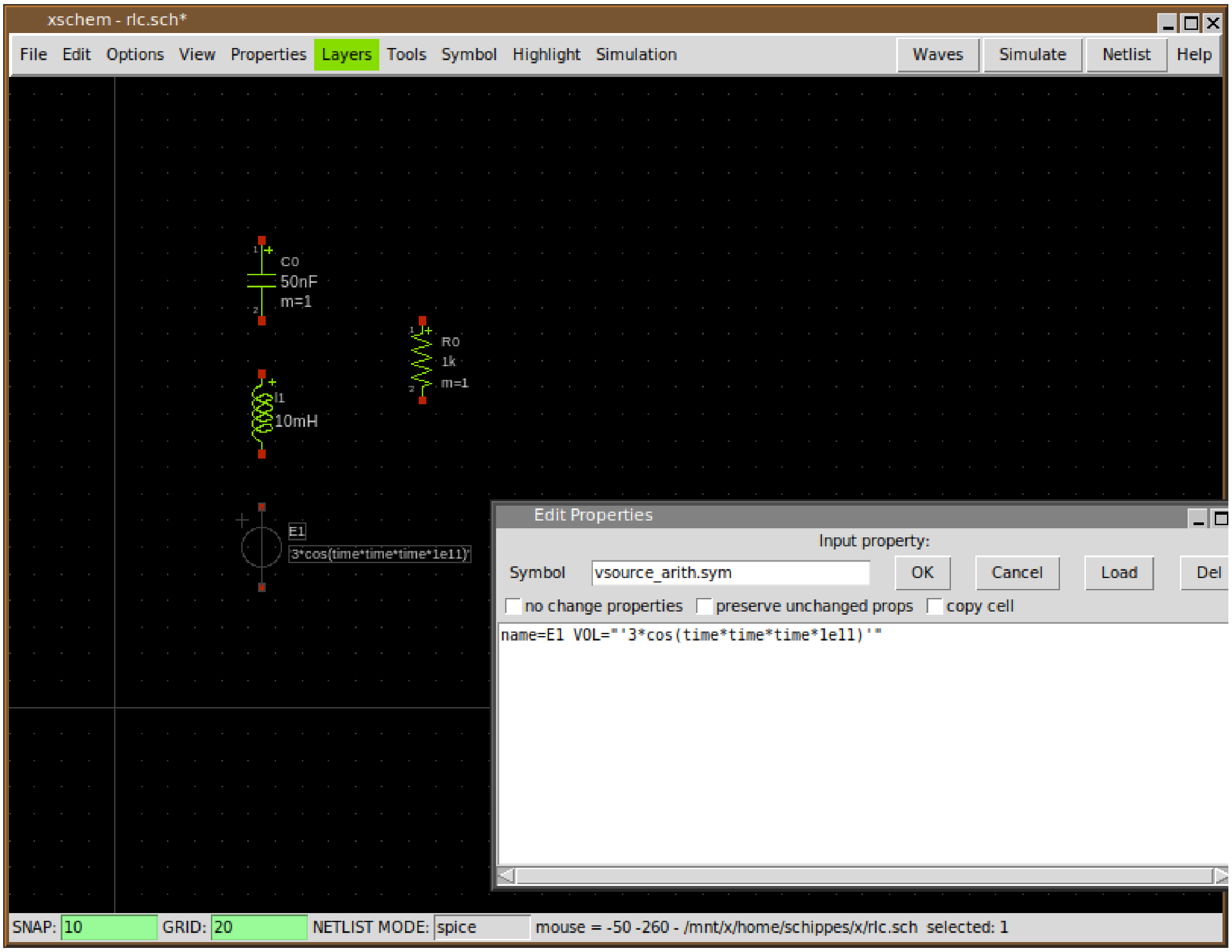


```
name=C0 m=1 value=50nF footprint=1206 device="ceramic capacitor"
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -70 -210 - /mnt/x/home/schippes/x/rlc.sch selected: 1

Symbol: capa.sym

13. Do the same for the inductor (10mH) and the resistor (1k)
14. Set the voltage source VOL to: "'3*cos(time*time*time*1e11)'" (include quotes, single and double):

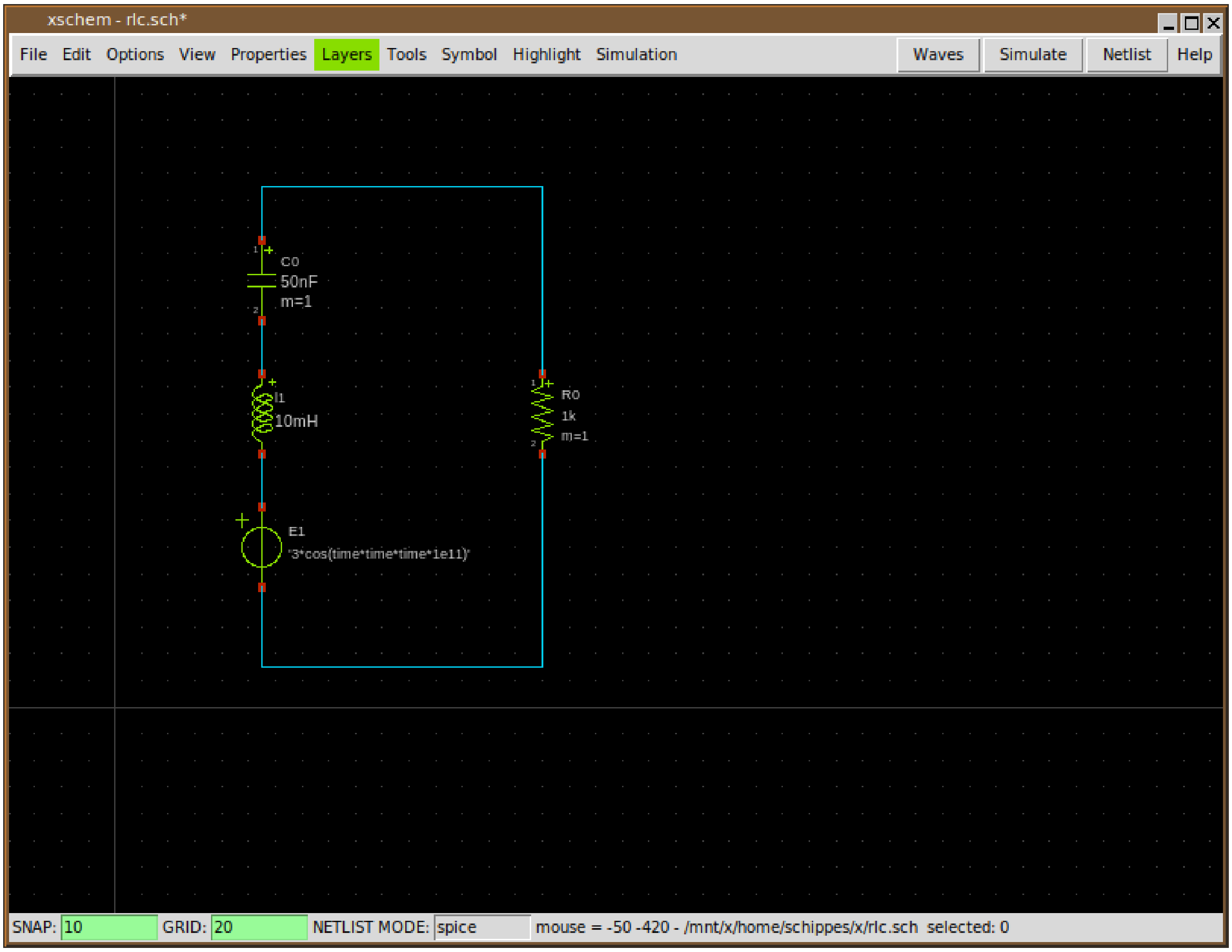


Symbol: vsource_arith.sym

```
name=E1 VOL="'3*cos(time*time*time*1e11)'"
```

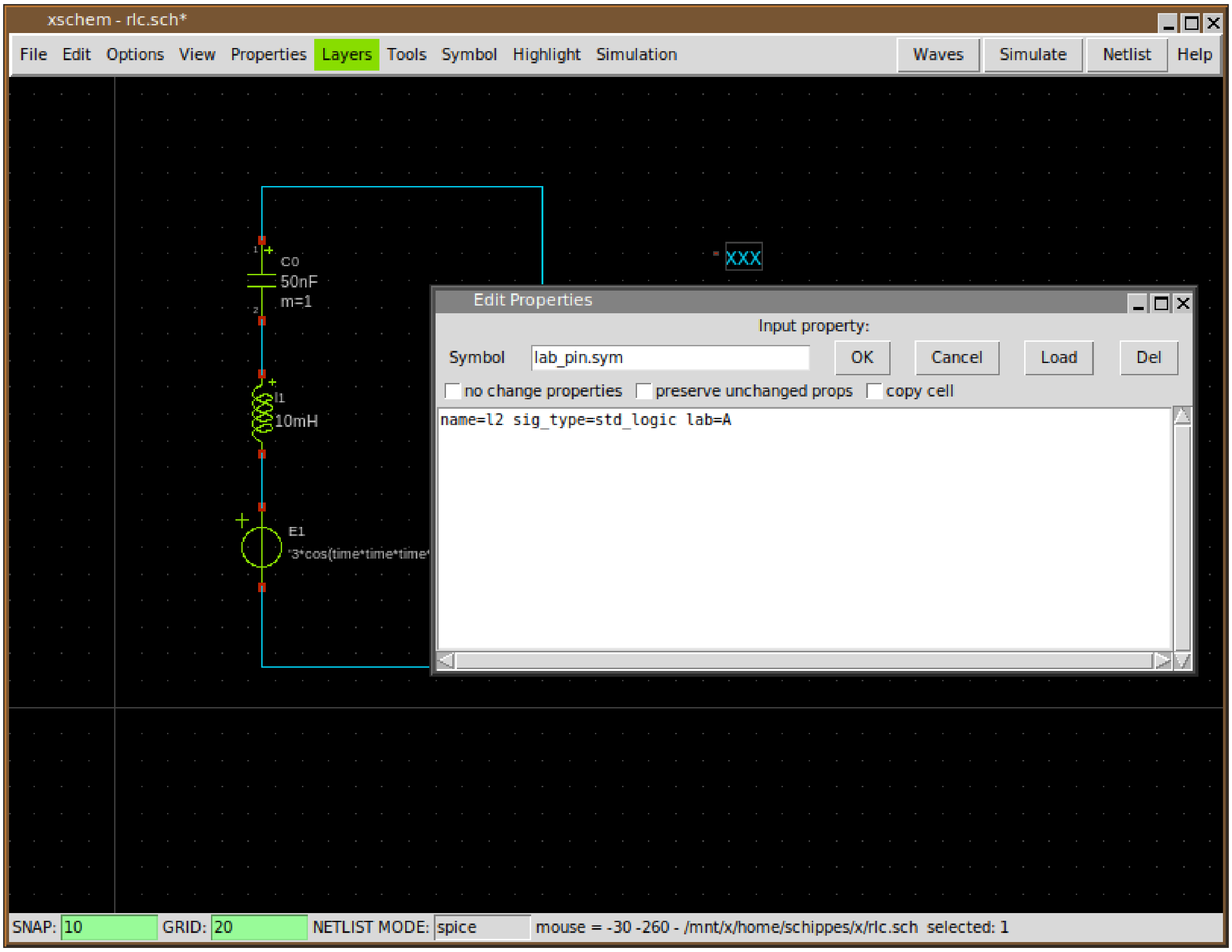
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -50 -260 - /mnt/x/home/schippes/x/rlc.sch selected: 1

15. Pressing the 'w' key and moving the mouse you draw wires, wire the components as shown (press 'w', move the mouse and click, this draws a wire segment):


xschem - rlc.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
C0
50nF
m=1
l1
10mH
R0
1k
m=1
E1
'3*cos(time*time*time*1e11)'

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -50 -420 - /mnt/x/home/schippes/x/rlc.sch selected: 0

16. Press 'Insert key and place one instance of 'lab_pin', then use the right mouse button to change its 'lab' attribute to A:

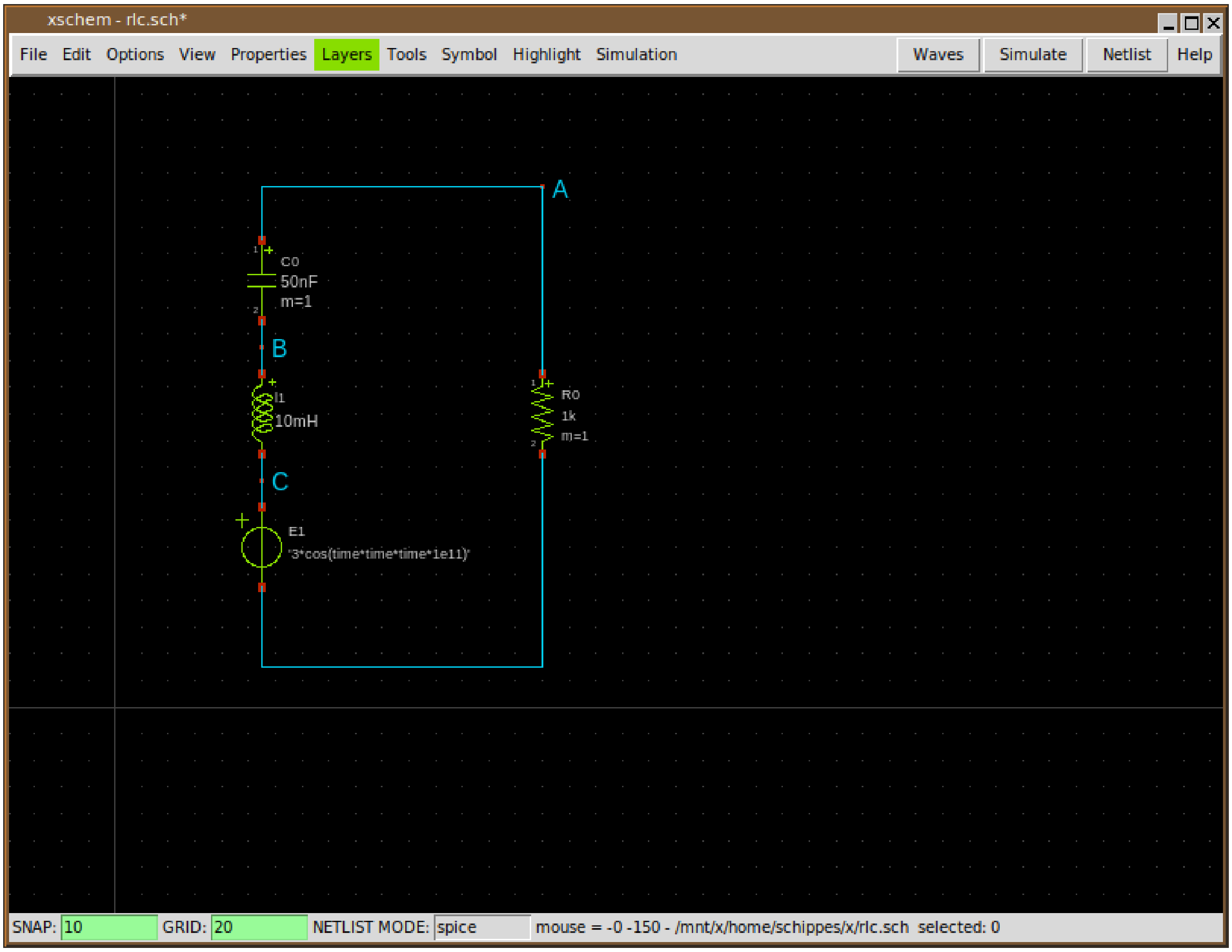


Symbol: lab_pin.sym

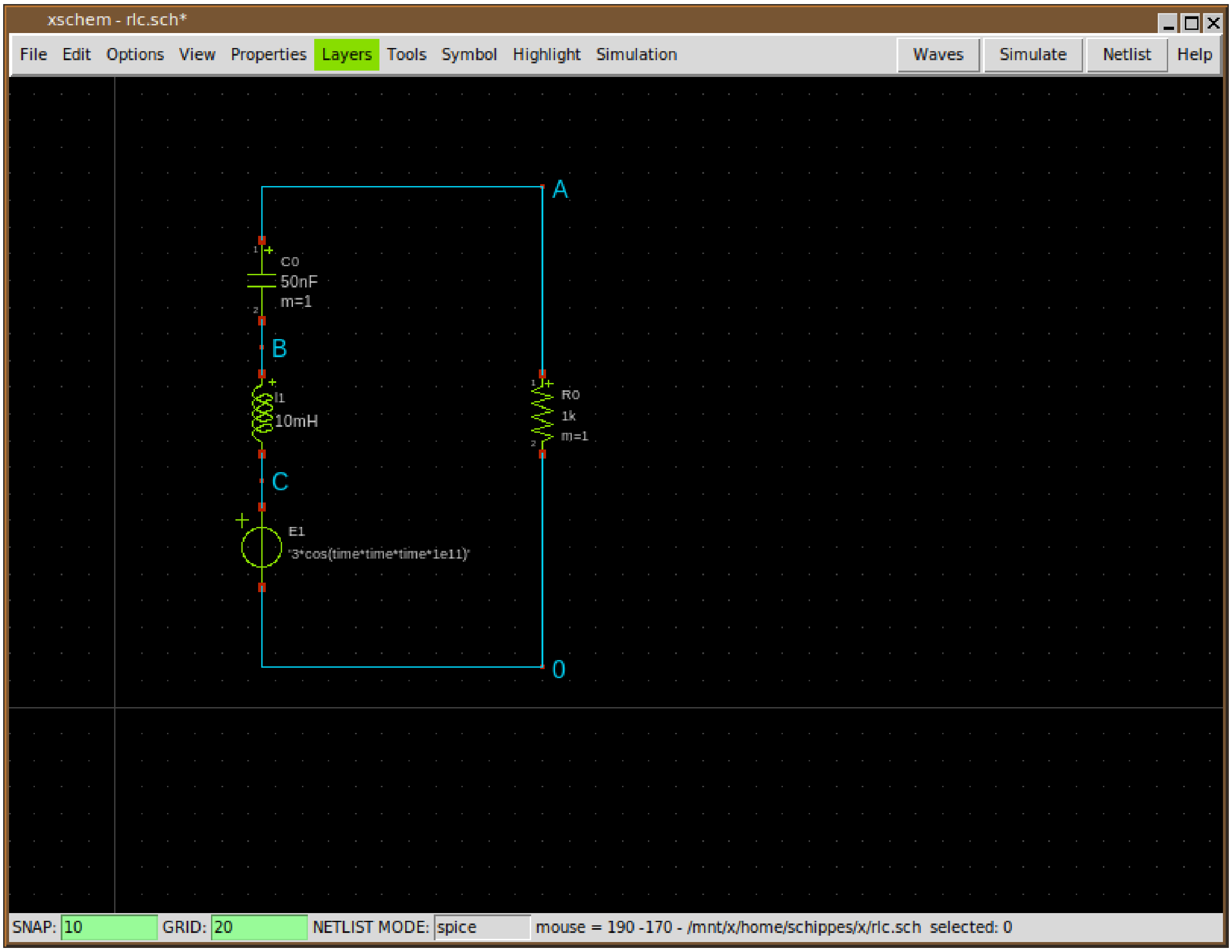
```
name=l2 sig_type=std_logic lab=A
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -30 -260 - /mnt/x/home/schippes/x/rlc.sch selected: 1

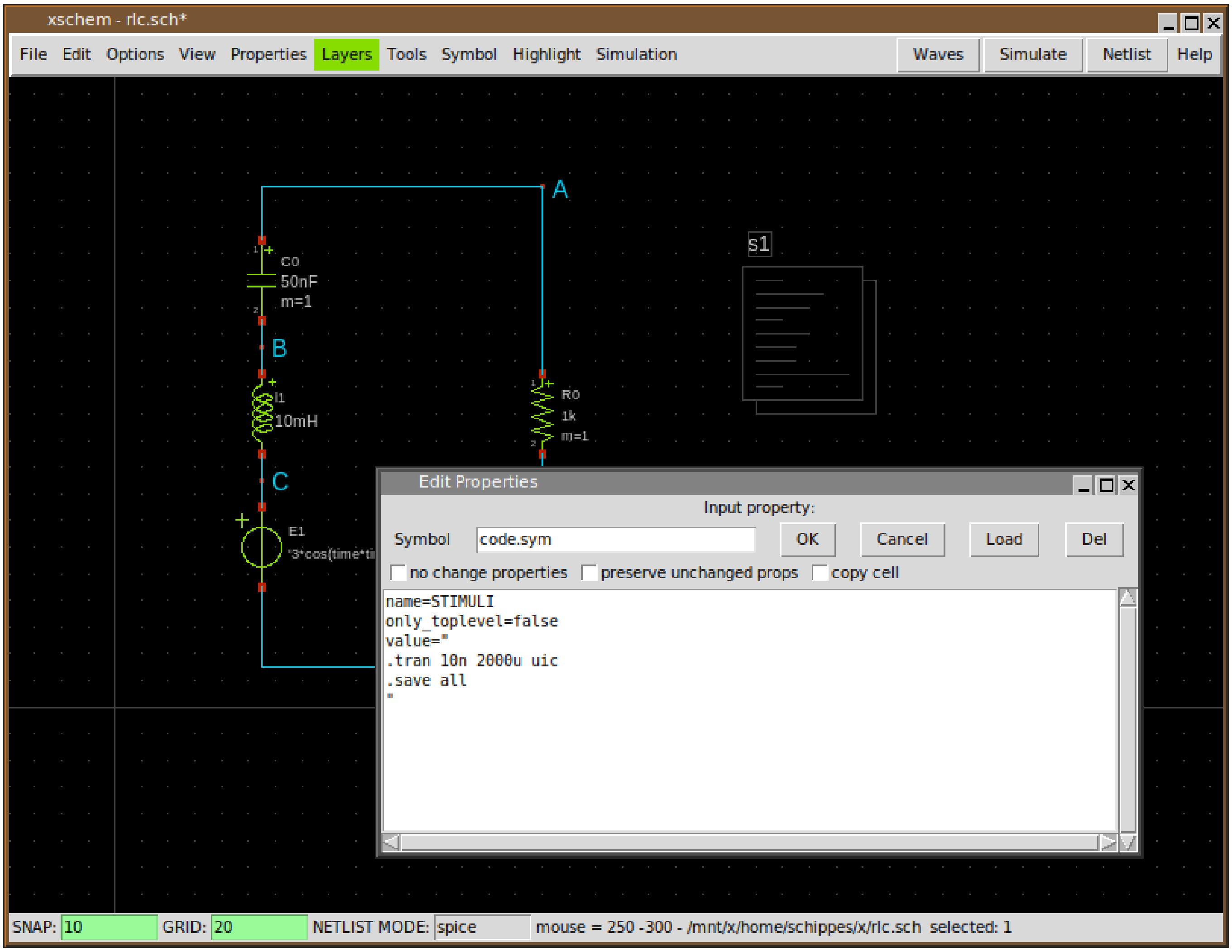
17. Move the label as shown, (you can use 'Shift+F' to flip and 'Shift+R' to rotate), then using 'c' copy this pin label and edit attributes to create the B and C labels, place all of these as shown:


xschem - rlc.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
A
B
C
C0
50nF
m=1
l1
10mH
R0
1k
m=1
E1
'3*cos(time*time*time*1e11)'
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = -0 -150 - /mnt/x/home/schippes/x/rlc.sch selected: 0

18. Select the 'C' label and copy it as shown here, set its lab attribute to 0 (this will be the 0V (gnd node))


xschem - rlc.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
A
C0
50nF
m=1
B
l1
10mH
C
E1
'3*cos(time*time*time*1e11)'
R0
1k
m=1
0
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 190 -170 - /mnt/x/home/schippes/x/rlc.sch selected: 0

19. Press 'Insert key, place the 'code.sym' symbol, set name and value attributes as follows:

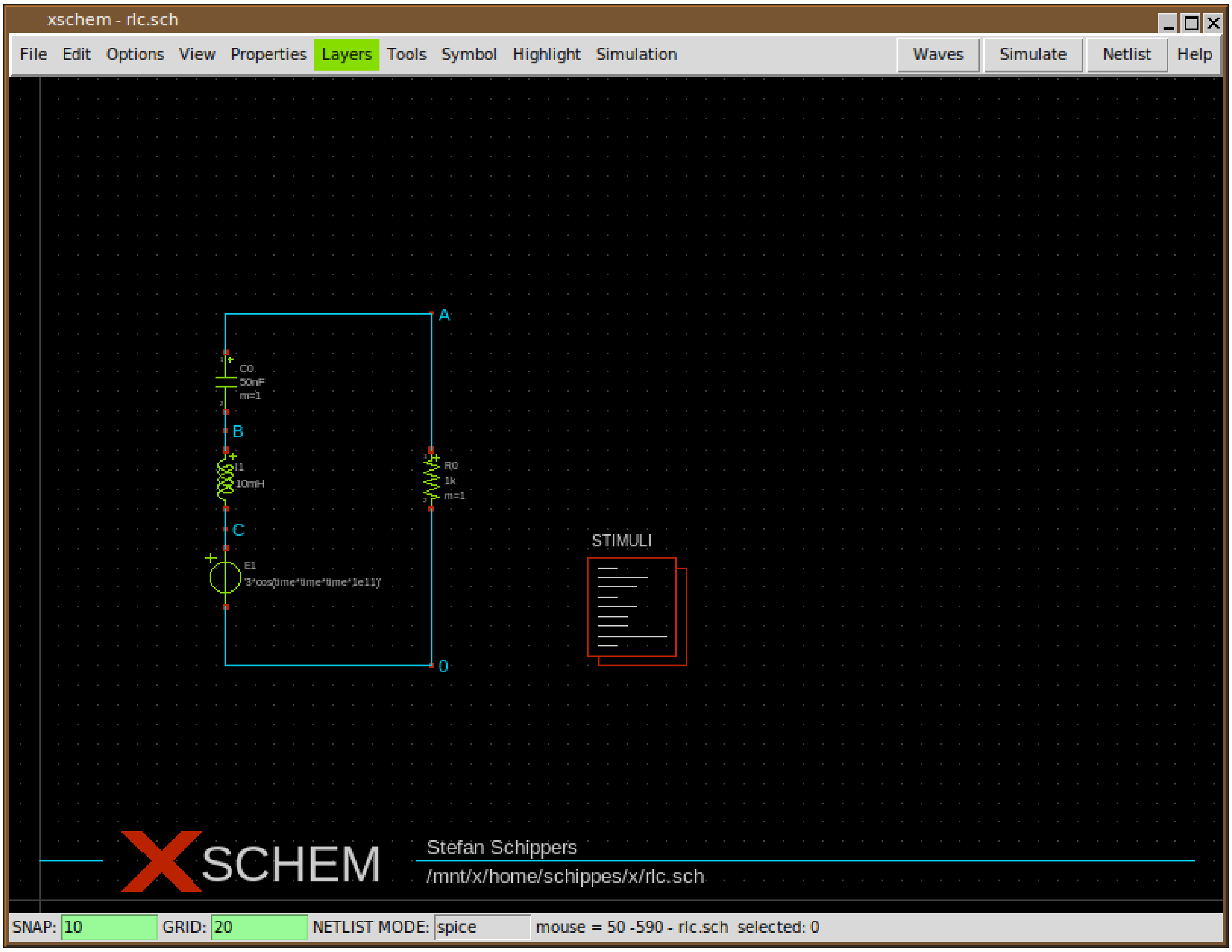


Symbol code.sym

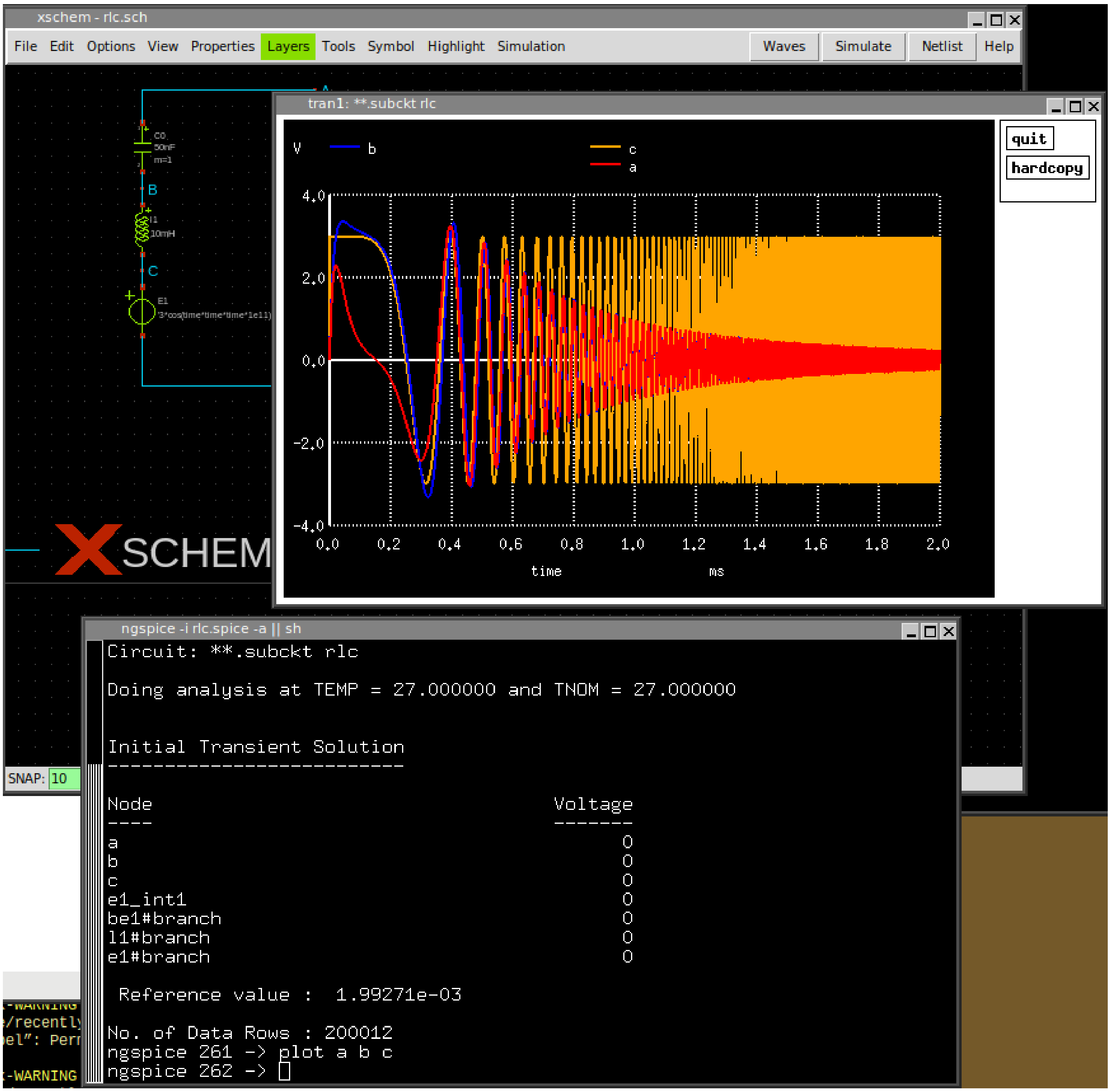
```
name=STIMULI
only_toplevel=false
value="
.tran 10n 2000u uic
.save all
"
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 250 -300 - /mnt/x/home/schippes/x/rlc.sch selected: 1

20. Cosmetics: add 'title.sym' move the circuit (by selecting it dragging the mouse and pressing 'm', if needed). Note that you can do a 'stretch move' operation if you need move components keeping the wires attached; refer to the xschem manual here


xschem - rlc.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
A
C0
50nF
m=1
B
l1
10mH
R0
1k
m=1
C
E1
'3*cos(time*time*time*1e11)'
0
STIMULI
XSCHEM
Stefan Schippers
/mnt/x/home/schippes/x/rlc.sch
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = 50 -590 - rlc.sch selected: 0

21. The circuit is ready for simulation: press 'netlist' the 'rlc.spice' will be generated in current dir.
22. If ngspice is installed on the system press 'Simulate':
23. In the simulator window type 'plot a b c':


xschem - rlc.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
C0
50nF
m=1
B
l1
10mH
C
E1
3*cos(time*time*time*1e11)
SCHEM
SNAP: 10
tran1: **.subckt rlc
quit
hardcopy
V
b
c
a
4.0
2.0
0.0
-2.0
-4.0
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
time
ms
ngspice -i rlc.spice -a || sh

```
Circuit: **.subckt rlc

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


Initial Transient Solution
--------------------------

Node                                   Voltage
----                                   -------
a                                            0
b                                            0
c                                            0
e1_int1                                      0
be1#branch                                   0
l1#branch                                    0
e1#branch                                    0

 Reference value :  1.99271e-03

No. of Data Rows : 200012
ngspice 261 -> plot a b c
ngspice 262 ->
```

24. If you set 'Simulation -> Configure simulators and tools -> Ngspice Batch' and press 'Simulate' again the sim will be run in batch mode, a 'rlc.raw' file will be generated and a 'rlc.out' file will contain the simulator textual output.

UP

TUTORIAL: INSTANCE BASED SELECTION OF SYMBOL IMPLEMENTATION

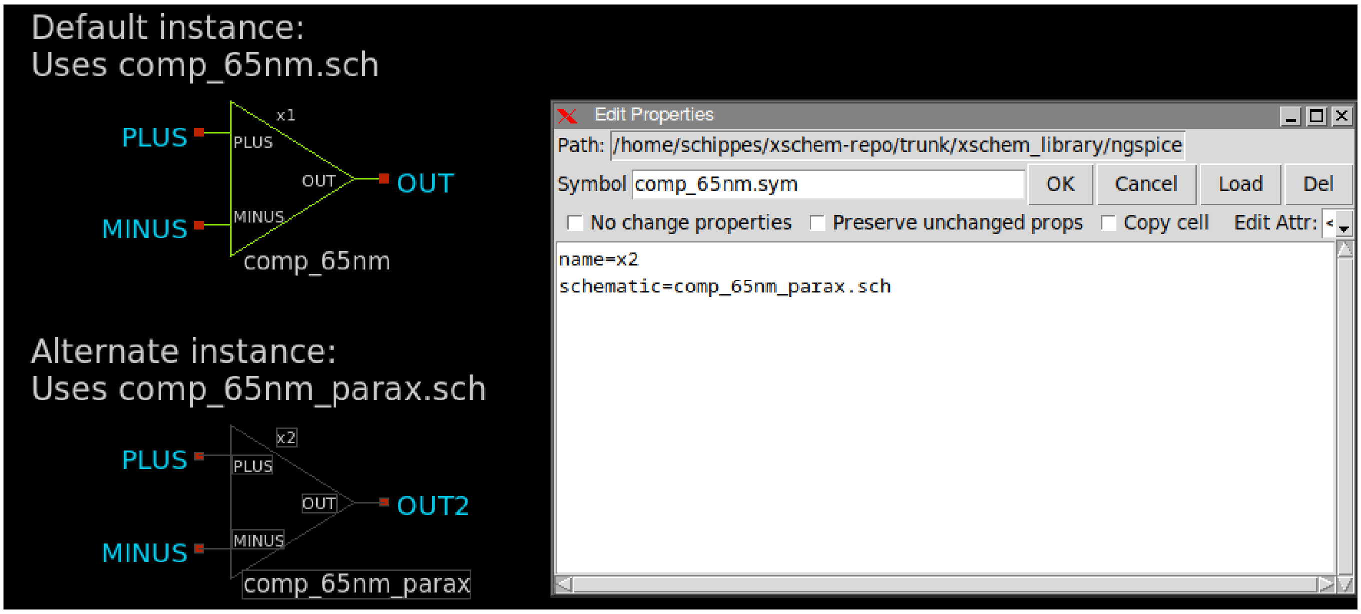
It is quite common to have in a design multiple instances of the same subcircuit. Think for example of memory arrays and decoder circuits. In some cases there are numerous instances of the same identical circuit. This leads to a very large netlist and heavy simulation loads (both in time and space).
On the other hand typically only a small portion of these repetitive circuits are exercised in simulation. For example you might want to simulate the selection only of the first 2 wordlines and the last 2 wordlines in a 1024 wordlines memory array.

In these situations it might be useful to keep the full subcircuit implementation for the circuit parts that are exercised and provide a simplified subcircuit for the parts that are idle during simulation. The simplified parts may just do the 'essential work' like driving the idle value on the outputs and loading the inputs with an equivalent of the input capacitance, in order to not alter the full circuit behavior.

schematic attribute on instance

Inside a symbol it is possible to specify an alternate **`schematic`** to descend into. For example if symbol **`inv.sym`** has attribute **`schematic=inv2.sch`** then xschem will descend into **`inv2.sch`** instead of the default **`inv.sch`**. See symbol property syntax man page. However these attributes at symbol level are applied to all instances of that symbol. To enable instance based differentiation it is now possible to use this attribute in the instance.
A **`schematic=<schematic reference>`** attribute attached to an instance will specify the schematic implementation to be used for that (and only that) instance of the subcircuit.
Example:

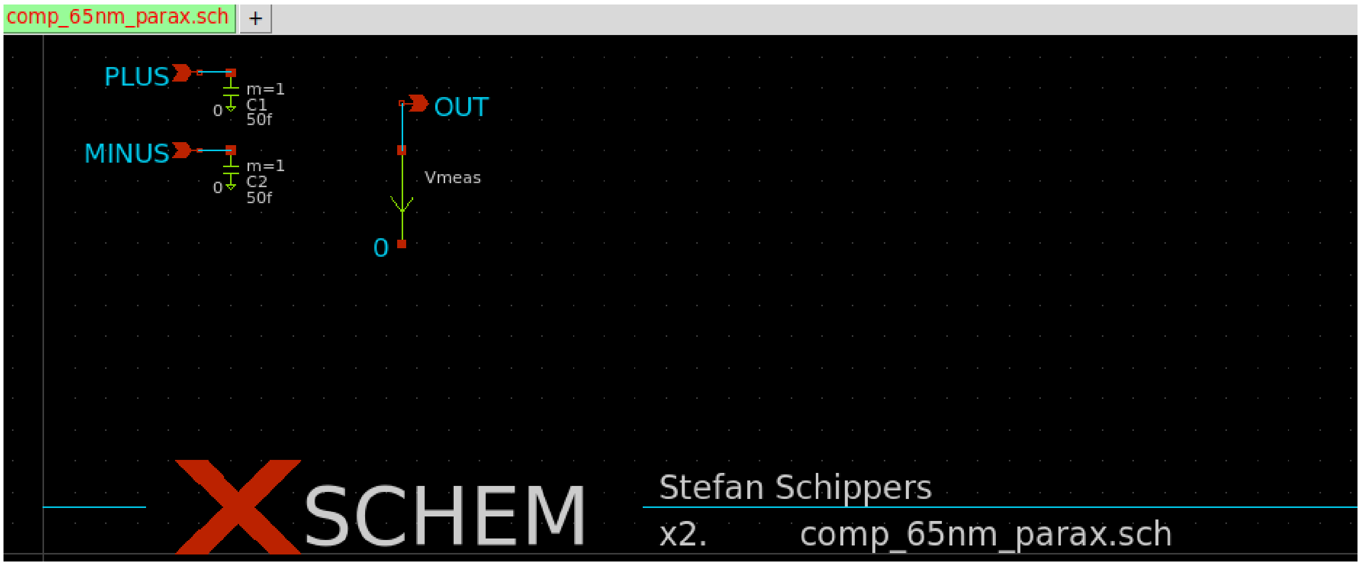
```
schematic=comp_65nm_parax.sch
```



Path: /home/schippes/xschem-repo/trunk/xschem_library/ngspice
Symbol comp_65nm.sym

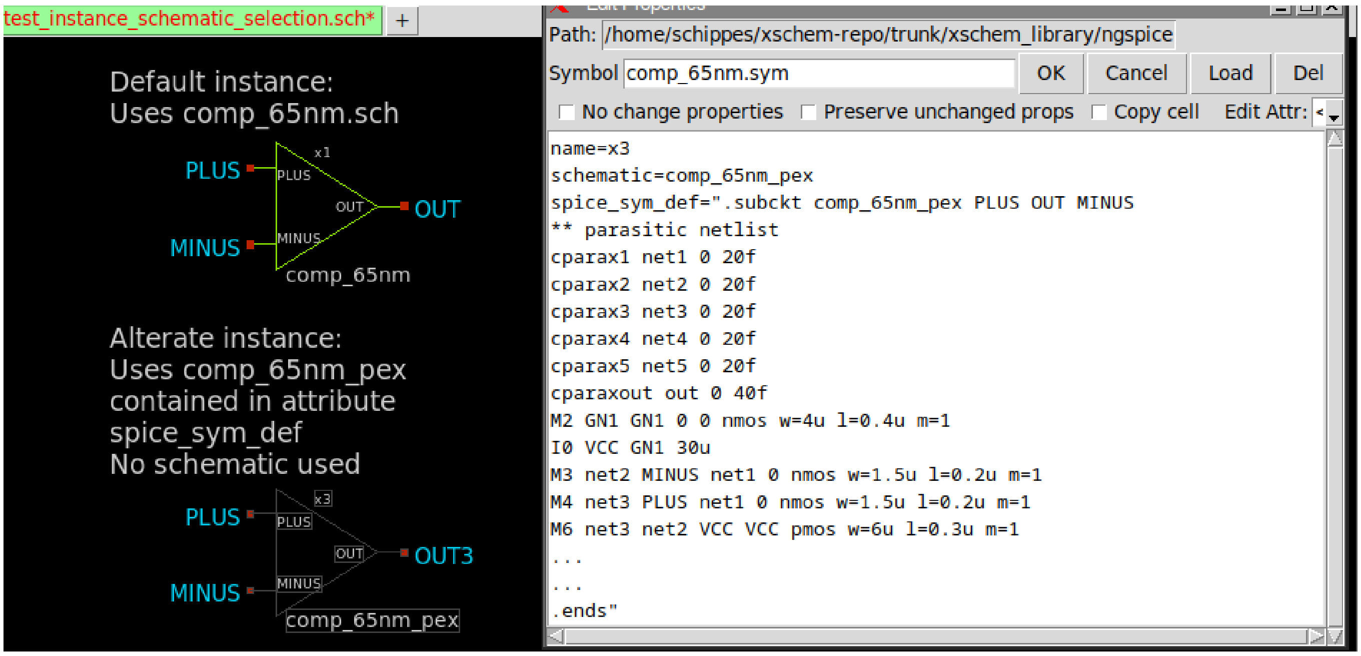
```
name=x2
schematic=comp_65nm_parax.sch
```

The **`comp_65nm_parax.sch`** schematic may be something like this, that is a simplified circuit that just keeps a known value on the outputs and adds some parasitic capacitance to the inputs.



spice_sym_def attribute on instance

A **`spice_sym_def=<...text...>`** attribute attached to an instance will specify some text that describes the subcircuit (it can be a simplified spice subcircuit netlist or a spice .include line that gets the subcircuit from an external file). This attribute on an instance must always be paired with a matching **`schematic`** attribute that specifies the subcircuit name the instance is linked to.



```
name=x3
schematic=comp_65nm_pex
spice_sym_def=".subckt comp_65nm_pex PLUS OUT MINUS
** parasitic netlist
cparax1 net1 0 20f
cparax2 net2 0 20f
cparax3 net3 0 20f
cparax4 net4 0 20f
cparax5 net5 0 20f
cparaxout out 0 40f
M2 GN1 GN1 0 0 nmos w=4u l=0.4u m=1
I0 VCC GN1 30u
M3 net2 MINUS net1 0 nmos w=1.5u l=0.2u m=1
M4 net3 PLUS net1 0 nmos w=1.5u l=0.2u m=1
M6 net3 net2 VCC VCC pmos w=6u l=0.3u m=1
...
...
.ends"
```

Another possibility is to specify these attributes so the actual netlist will be included by the simulator.

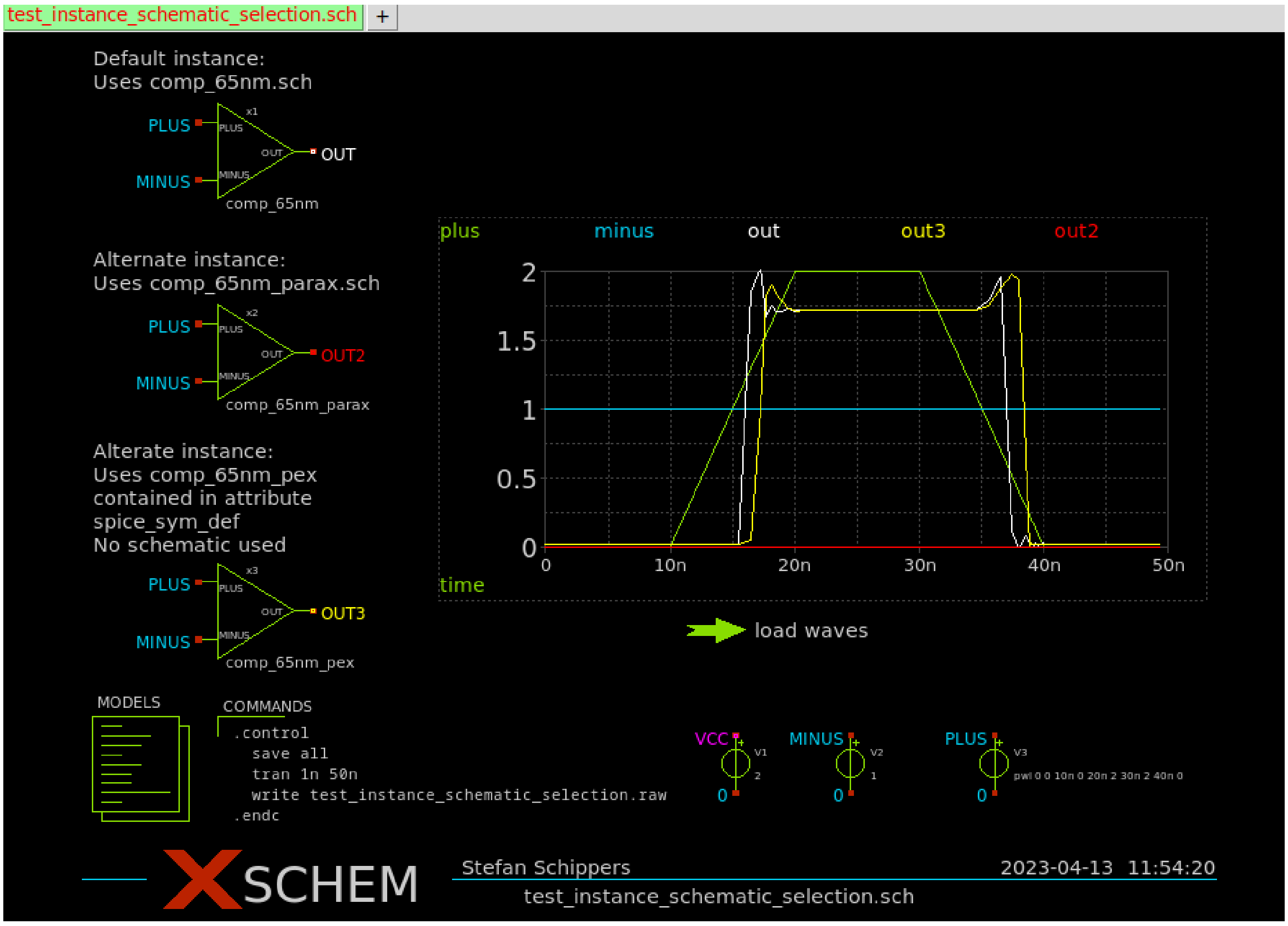
```
schematic=comp_65nm_pex
spice_sym_def=".include /path/to/comp_65nm_pex.cir"
```

When a **`spice_sym_def`** is specified no alternate schematic is used for this instance. The definition is provided as text (a piece of netlist, like for example a parasitic spice netlist extraction).

Putting this all together here is a schematic with 3 instances of **`comp_65nm.sym`**.

- **`x1`** is the standard instance using the default **`comp_65nm.sch`**
- **`x2`** is a simplified instance that just keeps the output low.
- **`x3`** uses a parasitic extraction netlist (output will move slower).

See the waveforms of the OUT, OUT2, OUT3 signals that behave accordingly.



```
.control
  save all
  tran 1n 50n
  write test_instance_schematic_selection.raw
.endc
```

Automatic port order setting from provided subcircuit netlist (Spice netlists only)

If a **`spice_sym_def`** attribute is defined and has one of the following forms:

```
spice_sym_def="
.subckt opamp PLUS MINUS OUT VCC VSS
...
...
...
.ends
"
```

Or:

```
spice_sym_def=".include /path/to/subckt_file"
```

Xschem will use the port order provided in the subckt line, either by looking directly into the attribute value or by loading the file specified by the .include line. This way there will not be inconsistencies between instance line and subckt definition in the circuit netlist. If for some reason the port list can not be read or pin names do not match xschem will use the port order drom the **`.sym`** file.

Note: all the above concepts are valid for VHDL, Verilog and tEDAx netlists by replacing the **`spice_sym_def`** attribute with **`vhdl_sym_def`**, **`verilog_sym_def`** and **`tedax_sym_def`** respectively.

Instance based SPICE model

In some cases a device is specified by a model and if model parameters can not be set in the instance line we need multiple models if we want to use multiple devices with different model parameters. This can be done by specifying a model in the following way:

```
type=mechanical_rotational
format="@name @pinlist inertia@name
.model inertia@name inertia_omega_tau J=@J"
template="name=N1 J=1.1"
```

Note the model name is given as **`inertia@name`**, this will make each model instance have a different and unique name. This will generate an instance line:

```
N1 A B C inertiaN1
.model inertiaN1 inertia_omega_tau J=1.4
```

A better way hat handles also vectored instances is the following:

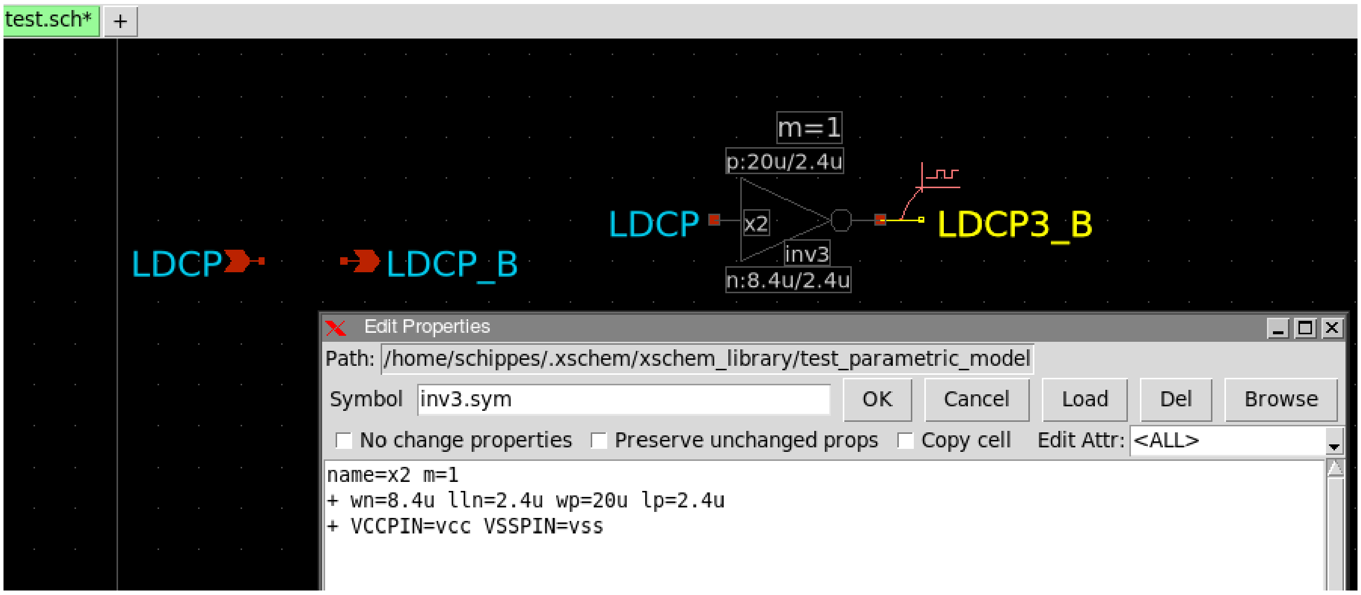
```
type=mechanical_rotational
format="@name @pinlist #inertia#@name
.model #inertia#@name inertia_omega_tau J=@J"
template="name=N1 J=1.1"
```

This way if you place a vectored instance name=N1[3:0] it will expand in netlist as:

```
N1[3] XAA XBB XCC inertiaN1[3]
N1[2] XAA XX XCC inertiaN1[2]
N1[1] XAA XX XCC inertiaN1[1]
N1[0] XAA XX XCC inertiaN1[0]
.model inertiaN1[3] inertia_omega_tau J=1.2
.model inertiaN1[2] inertia_omega_tau J=1.2
.model inertiaN1[1] inertia_omega_tau J=1.2
.model inertiaN1[0] inertia_omega_tau J=1.2
```

Subcircuits with SPICE models given as parameters

In general SPICE allows parameters to be passed to subcircuits. This is the case for dimensions, like **`W=2u`**, **`L=0.15u`** that are passed to a subcircuit. The subcircuit uses these parameters (W, L) instead of numbers, making the subcircuit truly parametric. However transistor models in a subcircuit can not be passed as parameters, the following inverter instantiation is illegal:
`X1 A Y inverter W=2u L=0.15u modn=cmosn modp=cmosp`
To overcome this problem Xschem must generate multiple subcircuits. Consider the following inv3.sym symbol:



Path: /home/schippes/.xschem/xschem_library/test_parametric_model

Symbol inv3.sym

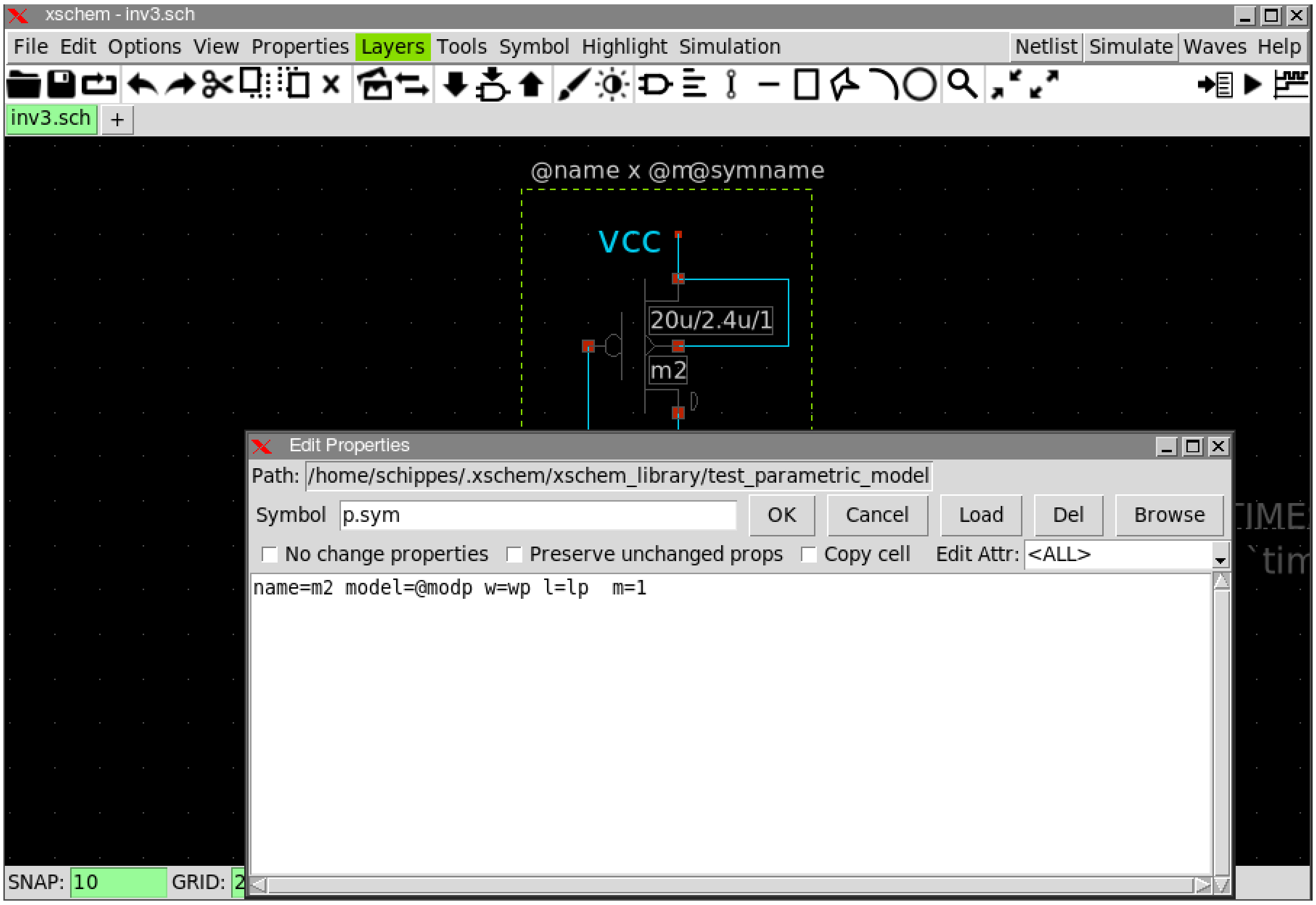
No change properties Preserve unchanged props Copy cell Edit Attr: <ALL>

```
name=x2 m=1
+ wn=8.4u lln=2.4u wp=20u lp=2.4u
+ VCCPIN=vcc VSSPIN=vss
```

the symbol has the following attributes:

```
type=subcircuit
format="@name @pinlist @VCCPIN @VSSPIN @symname wn=@wn lln=@lln wp=@wp lp=@lp m=@m"
template="name=x1 m=1 modn=xmodn modp=xmodp
+ wn=10u lln=1.2u wp=10u lp=1.2u
+ VCCPIN=VCC VSSPIN=VSS"
extra="VCCPIN VSSPIN modn modp"
```

In above attributes two parameters are defined that specify transistor models, **`modn`** and **`modp`**, with default values (if unspecified in instance) **`xmodn`** and **`xmodp`**. The inverter subcircuit transistors will use the **`@modn`** and **`@modp`** as SPICE models:



Path: /home/schippes/.xschem/xschem_library/test_parametric_model

Symbol: p.sym

Edit Attr: <ALL>

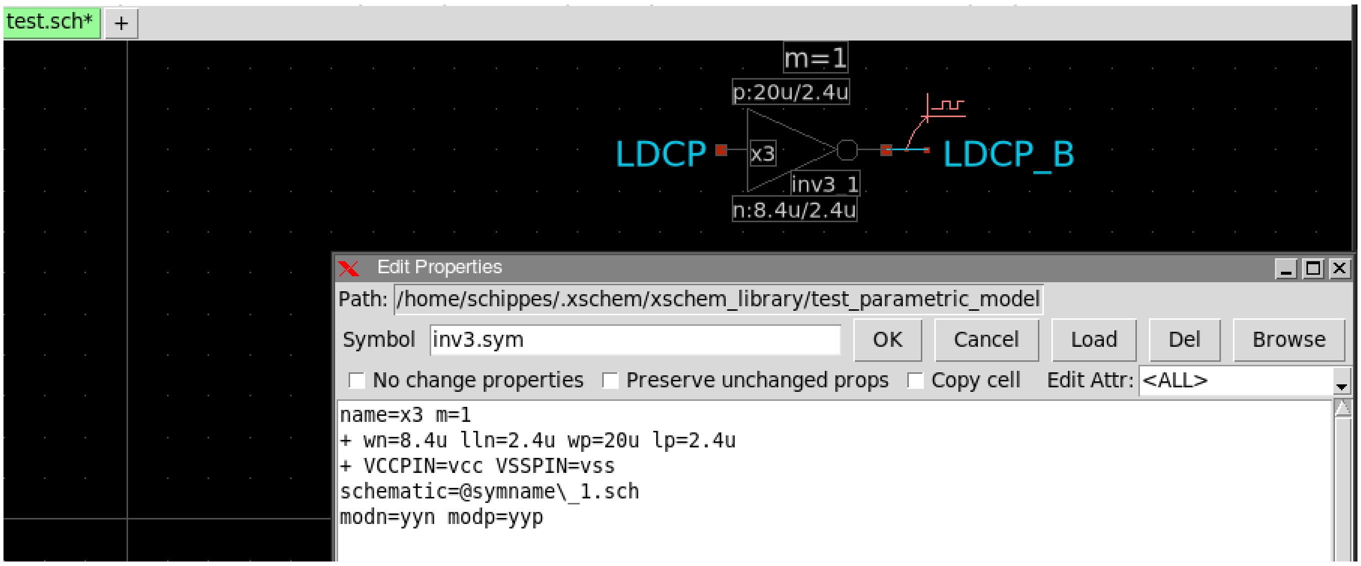
```
name=m2 model=@modp w=wp l=lp  m=1
```

If an inv3.sym is placed n the schematic and no **`schematic=...`** parameter is given to create an instance based subcircuit specialization:

The following netlist will be produced:

```
x2 LDCP3_B LDCP vcc vss inv3 wn=8.4u lln=2.4u wp=20u lp=2.4u m=1
...
...
.subckt inv3 y a VCCPIN VSSPIN      wn=10u lln=1.2u wp=10u lp=1.2u
*.opin y
*.ipin a
m2 y a VCCPIN VCCPIN xmodp w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u' m=
m1 y a VSSPIN VSSPIN xmodn w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u' r
.ends
```

However if another instance is placed:



Path: /home/schippes/.xschem/xschem_library/test_parametric_model

Symbol inv3.sym

No change properties Preserve unchanged props Copy cell Edit Attr: <ALL>

```
name=x3 m=1
+ wn=8.4u lln=2.4u wp=20u lp=2.4u
+ VCCPIN=vcc VSSPIN=vss
schematic=@symname\_1.sch
modn=yyn modp=yyp
```

with following attributes:

```
name=x3 m=1
+ wn=8.4u lln=2.4u wp=20u lp=2.4u
+ VCCPIN=vcc VSSPIN=vss
schematic=@symname\_1.sch
modn=yyn modp=yyp
```

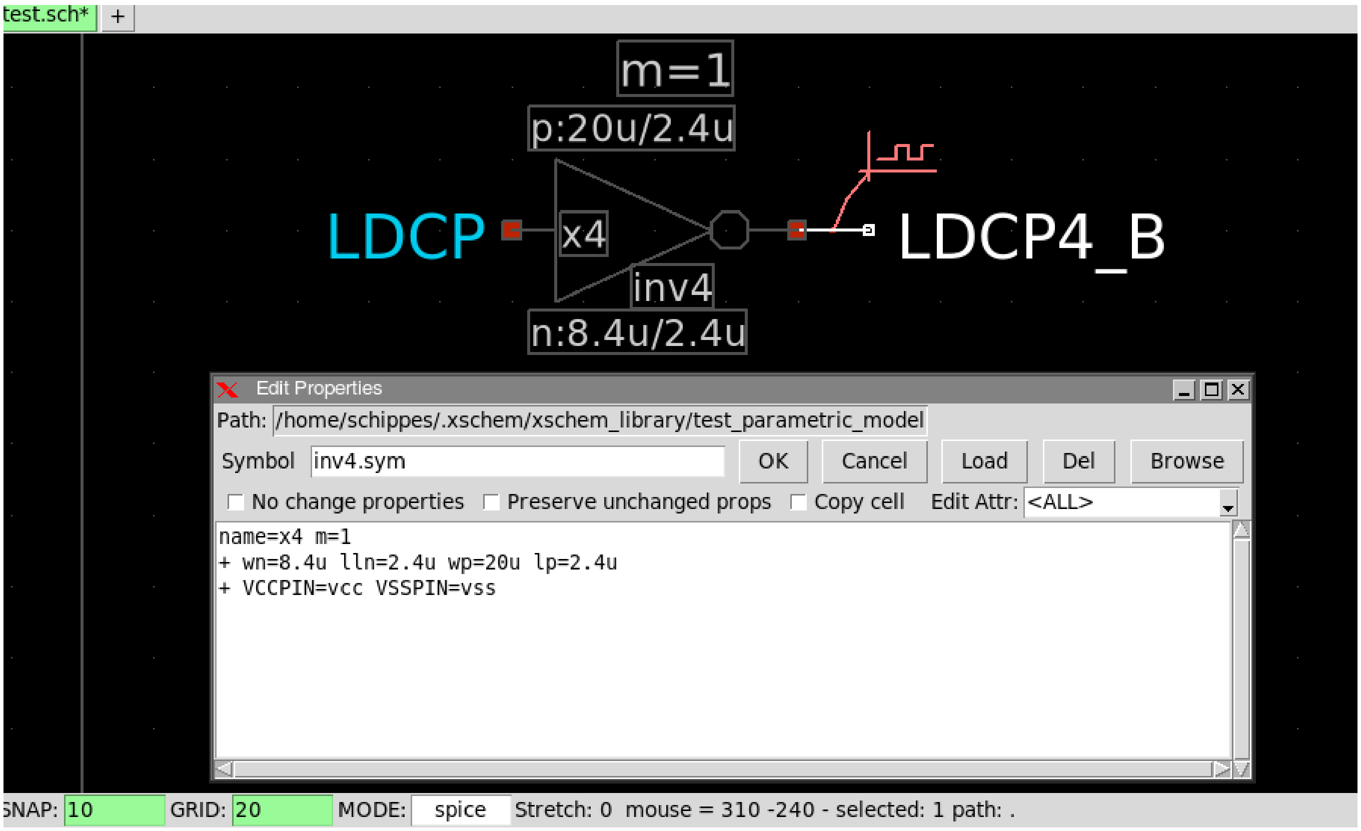
the following netlist is generated:

```
x3 LDCP_B LDCP vcc vss inv3_1 wn=8.4u lln=2.4u wp=20u lp=2.4u m=1
...
...
.subckt inv3_1 y a VCCPIN VSSPIN      wn=10u lln=1.2u wp=10u lp=1.2u
*.opin y
*.ipin a
m2 y a VCCPIN VCCPIN yyp w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u' m=1
m1 y a VSSPIN VSSPIN yyn w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u' m=
.ends
```

You see that a second **`inv_1.sym`** is generated with changed models (**`yyn`** and **`yyp`**). This allows you to reuse the same symbol with different model names. Xschem does the necessary work to duplicate the subcircuit, since model names can not be set as parameters.

Another example of spice models given as parameters

Consider the following symbol instance:



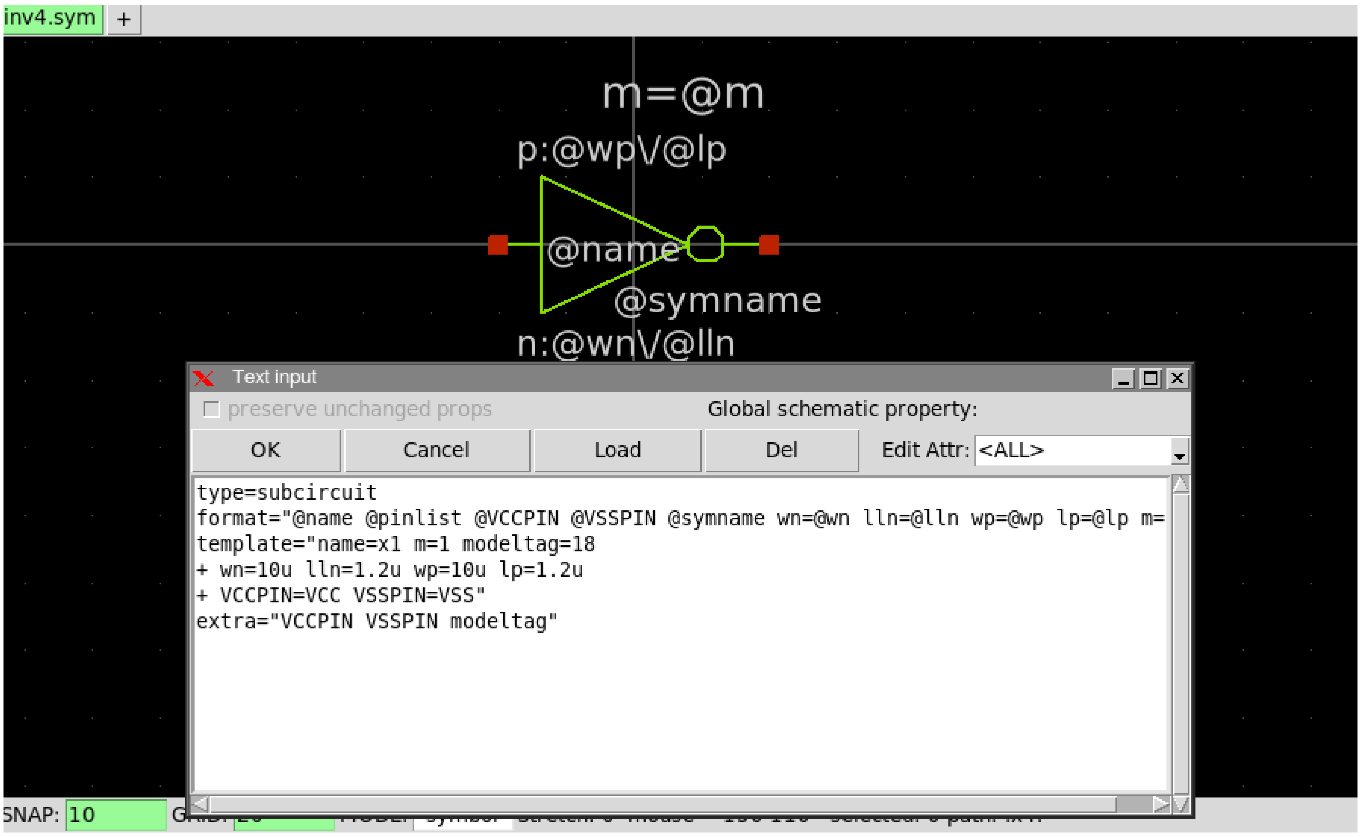
Path: /home/schippes/.xschem/xschem_library/test_parametric_model

Symbol inv4.sym

```
name=x4 m=1
+ wn=8.4u lln=2.4u wp=20u lp=2.4u
+ VCCPIN=vcc VSSPIN=vss
```

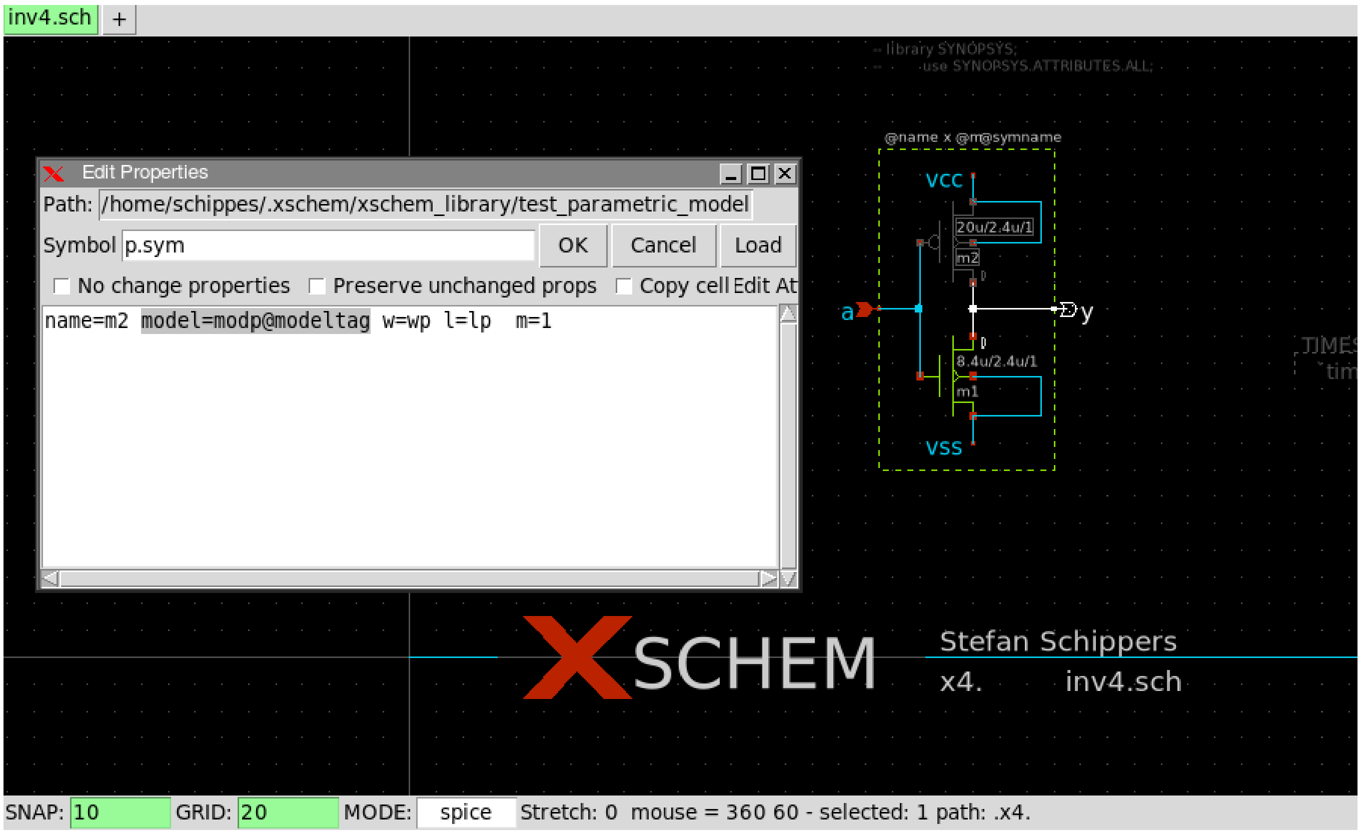
SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 310 -240 - selected: 1 path: .

with the following symbol definition



```
type=subcircuit
format="@name @pinlist @VCCPIN @VSSPIN @symname wn=@wn lln=@lln wp=@wp lp=@lp m=
template="name=x1 m=1 modeltag=18
+ wn=10u lln=1.2u wp=10u lp=1.2u
+ VCCPIN=VCC VSSPIN=VSS"
extra="VCCPIN VSSPIN modeltag"
```

And the following schematic definition. Note the **model** syntax for the p-channel transistor (the n-channel transistor has a similar **model=modn@modeltag** definition):



Path: /home/schippes/.xschem/xschem_library/test_parametric_model
Symbol p.sym
No change properties Preserve unchanged props Copy cell Edit At

```
name=m2 model=modp@modeltag w=wp l=lp  m=1
```

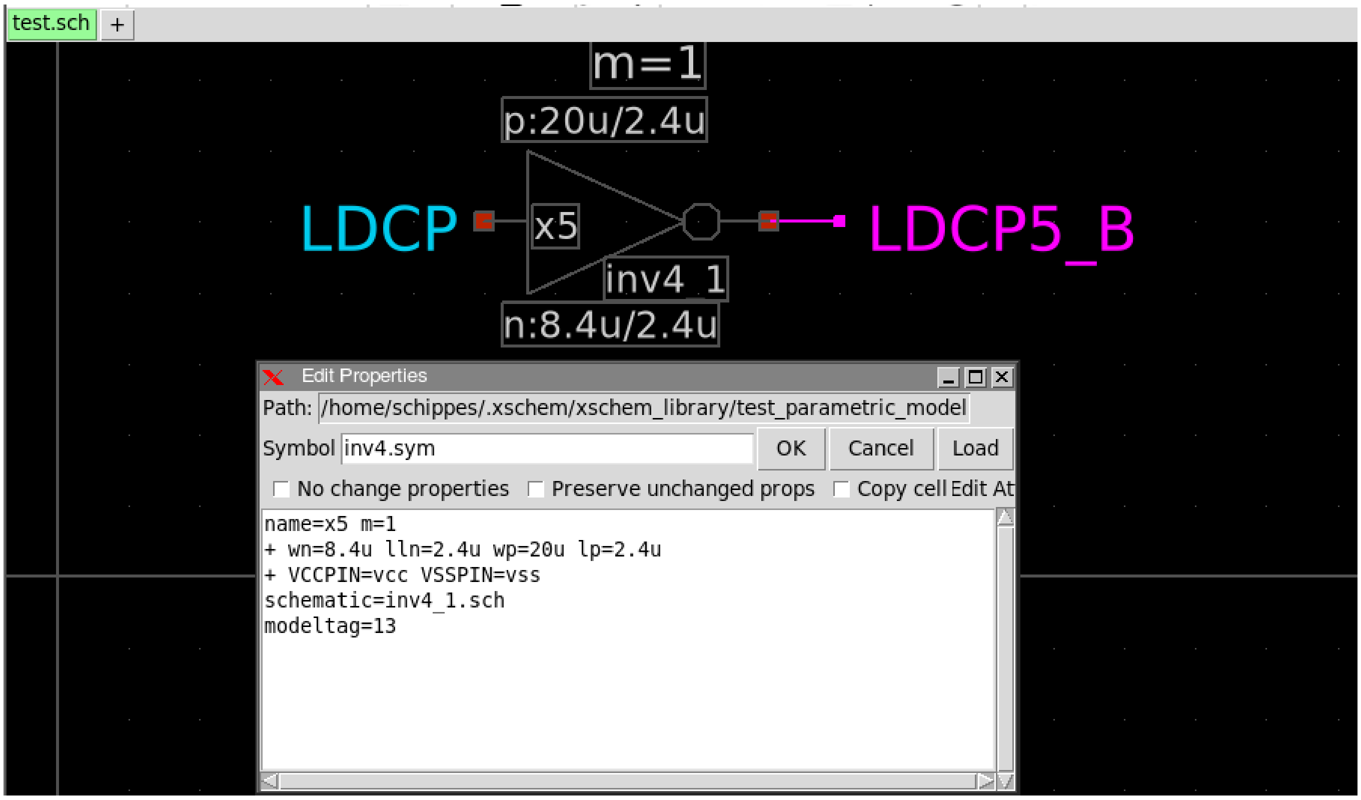
SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 360 60 - selected: 1 path: .x4.

The following netlist will be produced

```
...
...
x4 LDCP4_B LDCP vcc vss inv4 wn=8.4u lln=2.4u wp=20u lp=2.4u m=1
...
...
* expanding   symbol:  inv4.sym # of pins=2
** sym_path: /home/schippes/.xschem/xschem_library/test_parametric_model/inv4.sym
** sch_path: /home/schippes/.xschem/xschem_library/test_parametric_model/inv4.sch
.subckt inv4 y a VCCPIN VSSPIN     wn=10u lln=1.2u wp=10u lp=1.2u
*.opin y
*.ipin a
m2 y a VCCPIN VCCPIN modp18 w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u'
m1 y a VSSPIN VSSPIN modn18 w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u'
.ends
...
...
```

You see the **@modeltag** will be substituted looking first in the mos transistor attributes (but there is no definition there), then in the containing symbol **template** attributes (and there is a **modeltag=18** definition).

Now suppose you want to place another instance of **inv4.sym** but with a different modeltag: Since we know that spice does not allow model names to be passed as parameters we need to specialize the **inv4.sch** subcircuit to a new **inv_1.sch** subcircuit. Therefore we give the attribute **schematic=inv4_1.sch** to the second inv4 instance. We also set there a different modeltag: **modeltag=13**



Path: /home/schippes/.xschem/xschem_library/test_parametric_model

Symbol inv4.sym

```
name=x5 m=1
+ wn=8.4u lln=2.4u wp=20u lp=2.4u
+ VCCPIN=vcc VSSPIN=vss
schematic=inv4_1.sch
modeltag=13
```

The netlist for this additional instance will be:

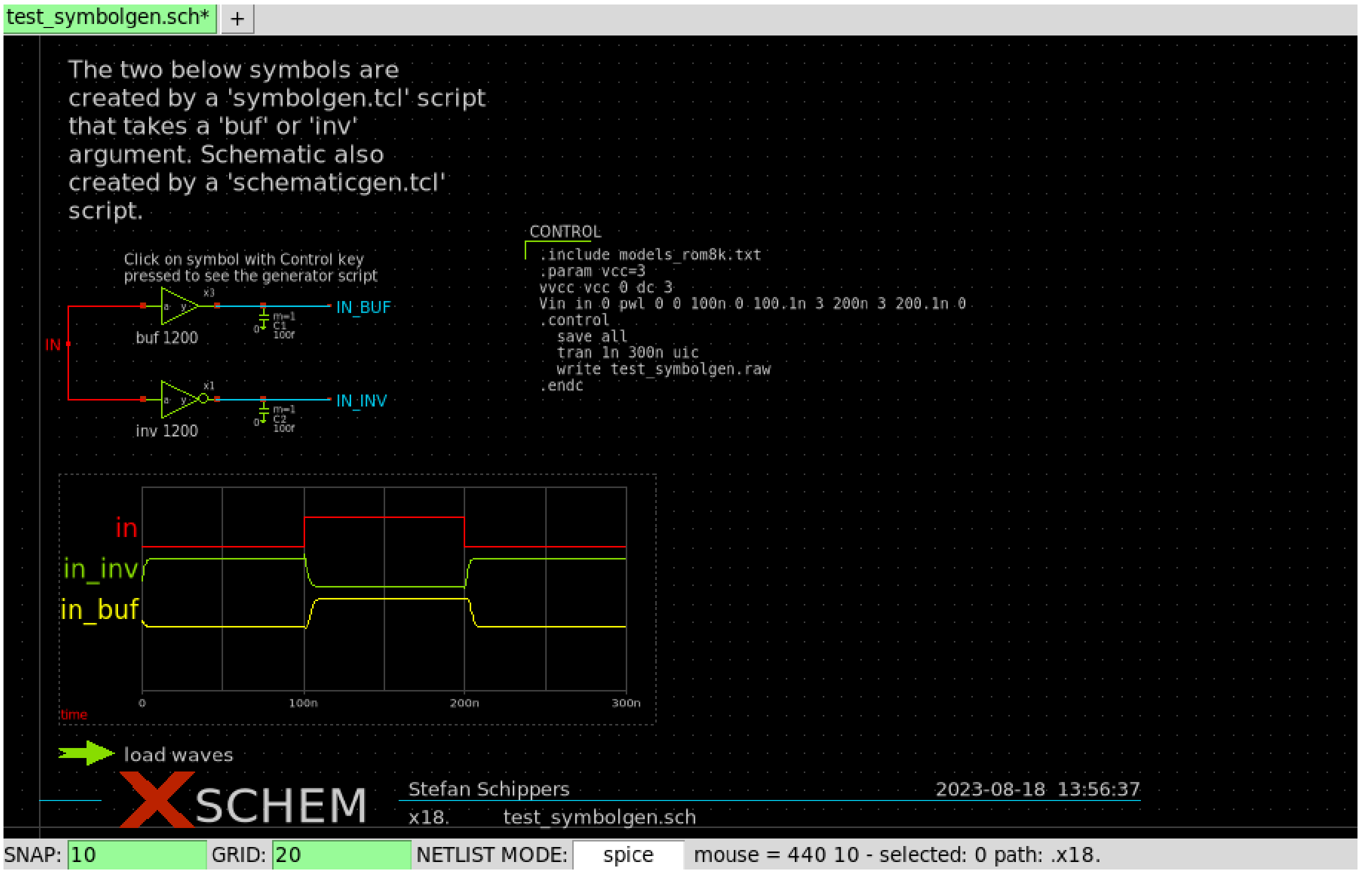
```
...
...
x5 LDCP5_B LDCP vcc vss inv4_1 wn=8.4u lln=2.4u wp=20u lp=2.4u m=1
...
...
* expanding   symbol:  inv4_1.sym # of pins=2
** sym_path: /home/schippes/.xschem/xschem_library/test_parametric_model/inv4.sym
** sch_path: /home/schippes/.xschem/xschem_library/test_parametric_model/inv4.sch
.subckt inv4_1 y a VCCPIN VSSPIN     wn=10u lln=1.2u wp=10u lp=1.2u
*.opin y
*.ipin a
m2 y a VCCPIN VCCPIN modp13 w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u'
m1 y a VSSPIN VSSPIN modn13 w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u'
.ends
...
...
```

This way it is possible from a single symbol (**`inv4.sym`** in the example) to netlist multiple instances of it with different models, in the example using a **`modeltag`** variable.

UP

TUTORIAL: SYMBOL AND SCHEMATIC GENERATORS (aka PCELLS)

It is possible to insert a symbol by referencing a generator script instead of a **.sym** file. When inserting the symbol select the **All** checkbox to see all files , select the generator script, then in the File/Search textbox add two parenthesis **()** (or put required parameters in between, like **(buf,250)**). If you don't append the **()** xschem will do that for you.



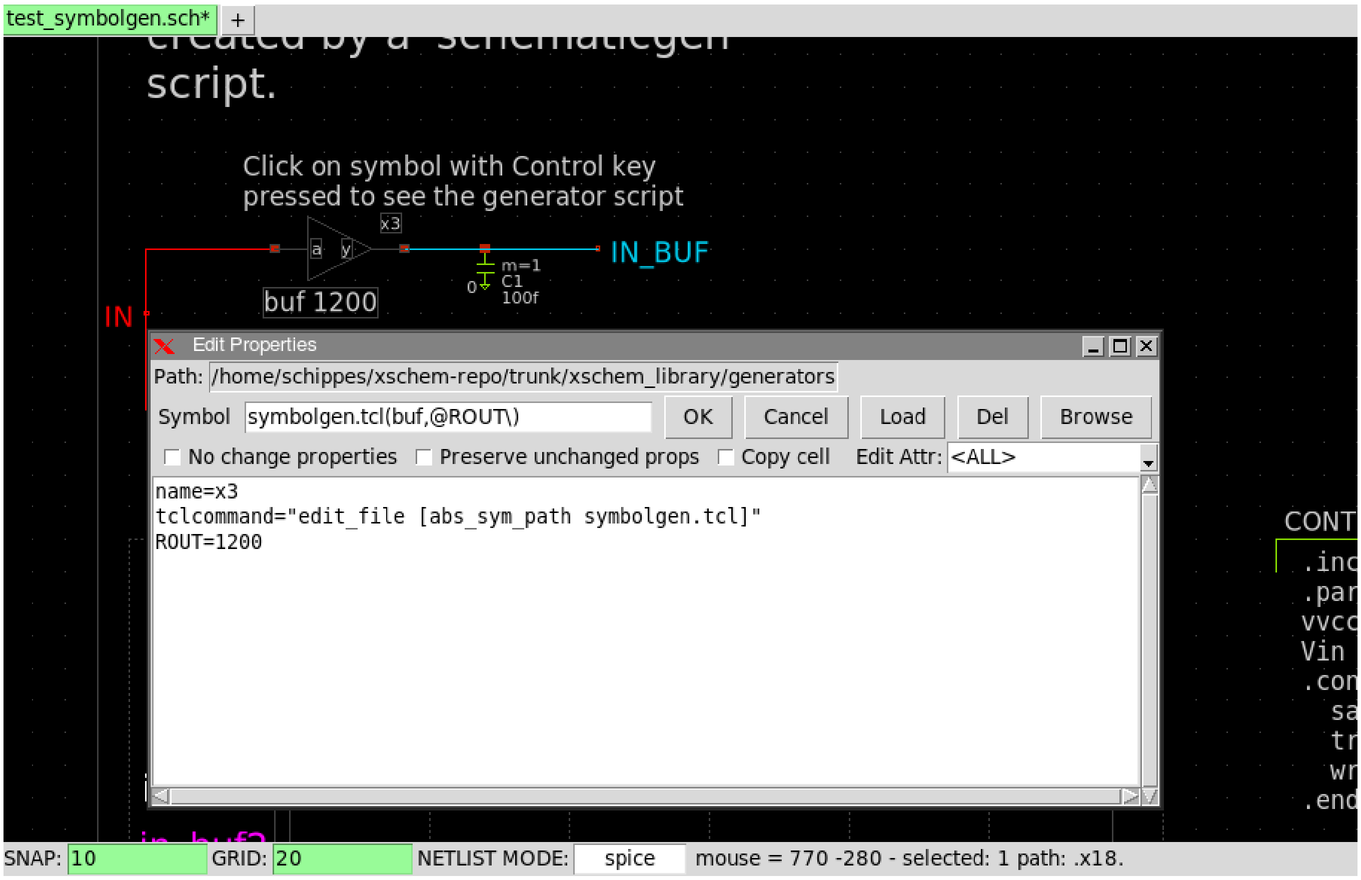
The two below symbols are created by a 'symbolgen.tcl' script that takes a 'buf' or 'inv' argument. Schematic also created by a 'schematicgen.tcl' script.

Click on symbol with Control key pressed to see the generator script

CONTROL
```
.include models_rom8k.txt
.param vcc=3
vvcc vcc 0 dc 3
Vin in 0 pwl 0 0 100n 0 100.1n 3 200n 3 200.1n 0
.control
  save all
  tran 1n 300n uic
  write test_symbolgen.raw
.endc
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 440 10 - selected: 0 path: .x18.

The symbolgen.tcl generator in this example takes two parameters, a **(buf)** or a **(inv)** parameter to generate a buffer or an inverter, respectively, and a output resistance value (a number) If no parameters are given (empty parentheses) a buffer is generated with a default ROUT.



Path: /home/schippes/xschem-repo/trunk/xschem_library/generators

Symbol symbolgen.tcl(buf,@ROUT\)

No change properties Preserve unchanged props Copy cell Edit Attr: <ALL>

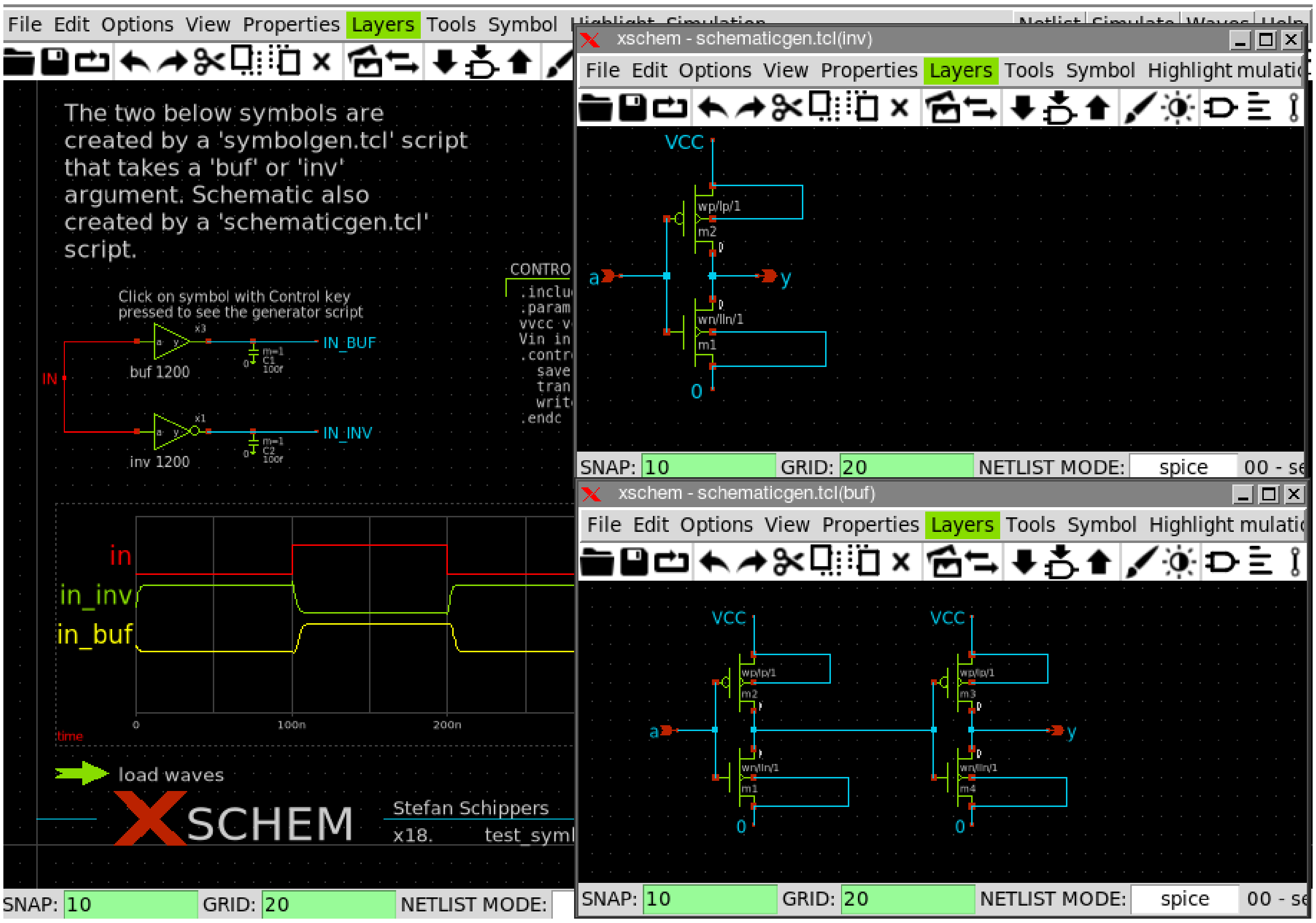
```
name=x3
tclcommand="edit_file [abs_sym_path symbolgen.tcl]"
ROUT=1200
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 770 -280 - selected: 1 path: .x18.

In this example a tcl script is used, you can use any language you like. The script only needs to parse the parameters (if any) and outputs on standard output a regular xschem symbol file.

```
#!/bin/sh
# the next line restarts using wish \
exec tclsh "$0" "$@"

set arg1 [lindex $argv 0]
set rout [lindex $argv 1]
# puts stderr "arg1=|$arg1| $rout=|$rout|"
if { $arg1 eq {inv}} {
puts "v {xschem version=3.1.0 file_version=1.2}
K {type=subcircuit
xvhdl_primitive=true
xverilog_primitive=true
xvhdl_format=\"@@y <= not @@a after 90 ps;\"
xverilog_format=\"assign #90 @@y = ~@@a ;\"
format=\"@name @pinlist @symname wn=@wn lln=@lln wp=@wp lp=@lp\"
template=\"name=x1 wn=1u lln=2u wp=4u lp=2u\"
schematic=schematicgen.tcl(inv)}
L 4 -40 0 -20 0 {}
L 4 -20 -20 20 0 {}
L 4 -20 -20 -20 20 {}
L 4 -20 20 20 0 {}
L 4 30 -0 40 -0 {}
B 5 37.5 -2.5 42.5 2.5 {name=y dir=out }
B 5 -42.5 -2.5 -37.5 2.5 {name=a dir=in }
A 4 25 -0 5 180 360 {}
T {$arg1 $rout} -47.5 24 0 0 0.3 0.3 {}
T {@name} 25 -22 0 0 0.2 0.2 {}
T {y} 7.5 -6.5 0 1 0.2 0.2 {}
T {a} -17.5 -6.5 0 0 0.2 0.2 {}
"
} else {
puts "v {xschem version=3.1.0 file_version=1.2}
K {type=subcircuit
xvhdl_primitive=true
xverilog_primitive=true
xvhdl_format=\"@@y <= @@a after 90 ps;\"
xverilog_format=\"assign #90 @@y = @@a ;\"
format=\"@name @pinlist @symname wn=@wn lln=@lln wp=@wp lp=@lp\"
template=\"name=x1 wn=1u lln=2u wp=4u lp=2u\"
schematic=schematicgen.tcl(buf)}
L 4 20 0 40 0 {}
L 4 -40 0 -20 0 {}
L 4 -20 -20 20 0 {}
L 4 -20 -20 -20 20 {}
L 4 -20 20 20 0 {}
B 5 37.5 -2.5 42.5 2.5 {name=y dir=out }
B 5 -42.5 -2.5 -37.5 2.5 {name=a dir=in }
T {$arg1 $rout} -47.5 24 0 0 0.3 0.3 {}
T {@name} 25 -22 0 0 0.2 0.2 {}
T {y} 7.5 -6.5 0 1 0.2 0.2 {}
T {a} -17.5 -6.5 0 0 0.2 0.2 {}
"
}
```

The **`generators/test_symbolgen.sch`** is a test schematic that places two instances of this symbol generator, one as **`symbolgen.tcl(buf,@ROUT\)`** and one as **`symbolgen.tcl(inv,@ROUT\)`**. The **`buf,@ROUT`** indicates two parameters, one indicates if it is a buffer or an inverter, the second passes an additional parameter. Instead of using a numeric literal the instance value ROUT is passed to the generator. A backslash is needed before the closing parenthesis to avoid this parenthesis to be considered as part of the parameter. The schematic implementations of these symbols are defined by the generator using a **`schematic`** attribute. The buffer will use **`schematicgen.tcl(buf)`** and the inverter will use **`schematicgen.tcl(inv)`**, these schematic names are referencing a schematic generator script instead of regular schematic files. You see different schematics (see below picture) when descending the buf or inv generator. See next section about schematic generators.



The two below symbols are created by a 'symbolgen.tcl' script that takes a 'buf' or 'inv' argument. Schematic also created by a 'schematicgen.tcl' script.

Click on symbol with Control key pressed to see the generator script

File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Simulate Waves Help
xschem - schematicgen.tcl(inv)
xschem - schematicgen.tcl(buf)
IN
IN_BUF
IN_INV
buf 1200
inv 1200
CONTRO
VCC
a
y
0
wp/lp/1
wn/lln/1
m1
m2
m3
m4
in
in_inv
in_buf
time
load waves
XSCHEM
Stefan Schippers
SNAP: 10
GRID: 20
NETLIST MODE: spice

The following is the extracted netlist from this example:

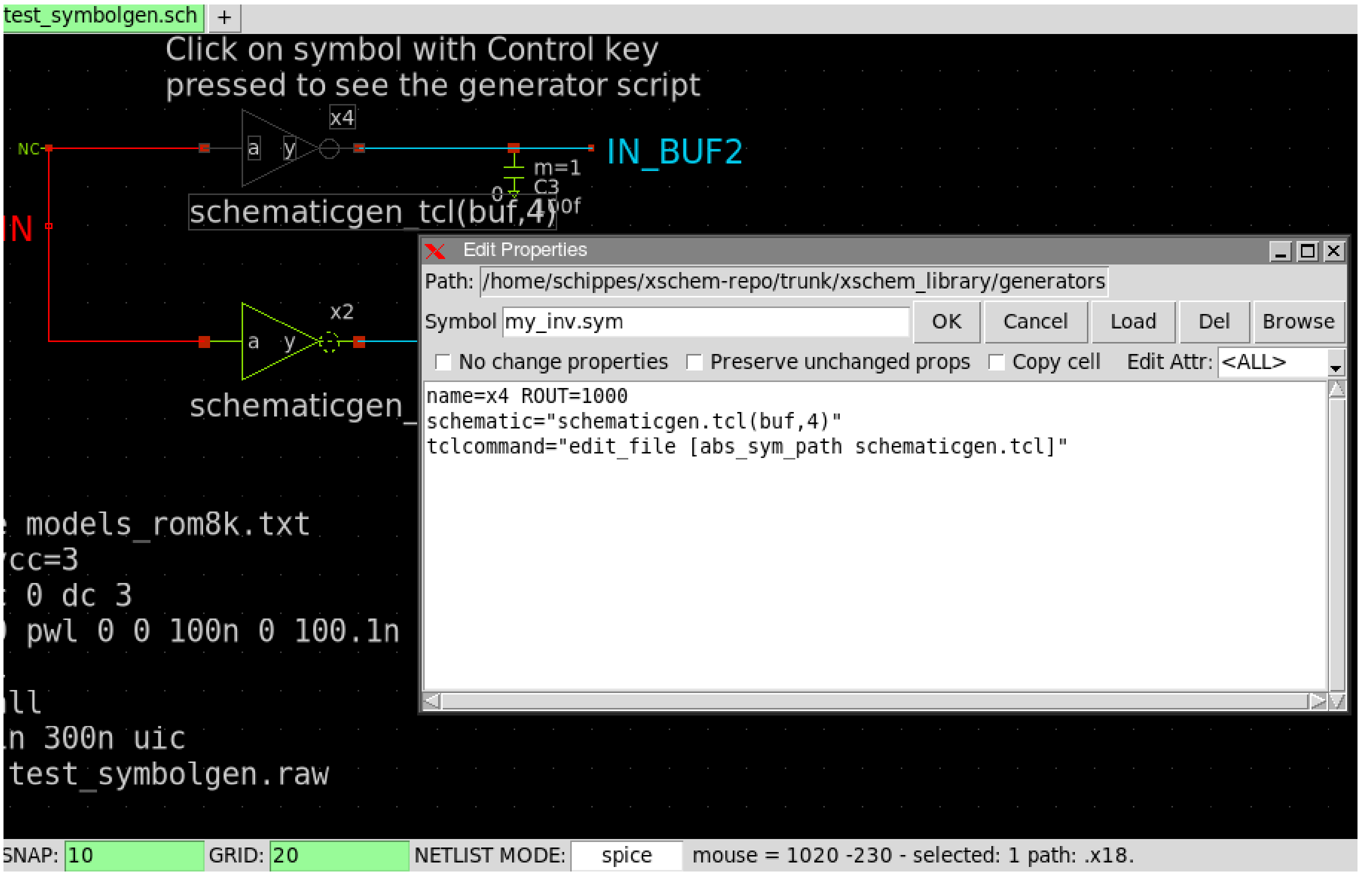
```
** sch_path: /home/schippes/xschem-repo/trunk/xschem_library/generators/test_symbolgen.sch
**.subckt test_symbolgen
x1 IN_INV IN symbolgen_tcl_inv_1200 wn=1u lln=2u wp=4u lp=2u
x3 IN_BUF IN symbolgen_tcl_buf_1200 wn=1u lln=2u wp=4u lp=2u
C1 IN_BUF 0 100f m=1
C2 IN_INV 0 100f m=1
**** begin user architecture code

.include models_rom8k.txt
.param vcc=3
vvcc vcc 0 dc 3
Vin in 0 pwl 0 0 100n 0 100.1n 3 200n 3 200.1n 0
.control
  save all
  tran 1n 300n uic
  write test_symbolgen.raw
.endc


**** end user architecture code
**.ends

* expanding   symbol:  symbolgen.tcl(inv,1200) # of pins=2
** sym_path: /home/schippes/xschem-repo/trunk/xschem_library/generators/symbolgen.tcl
** sch_path: /home/schippes/xschem-repo/trunk/xschem_library/generators/schematicgen.tcl
.subckt symbolgen_tcl_inv_1200 y a  wn=1u lln=2u wp=4u lp=2u
*.opin y
```

```
*.ipin a
m2 y a VCC VCC cmosp w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u' m=1
m1 y a 0 0 cmosn w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u' m=1
.ends


* expanding   symbol:  symbolgen.tcl(buf,1200) # of pins=2
** sym_path: /home/schippes/xschem-repo/trunk/xschem_library/generators/symbolgen.tcl
** sch_path: /home/schippes/xschem-repo/trunk/xschem_library/generators/schematicgen.tcl
.subckt symbolgen_tcl_buf_1200 y a  wn=1u lln=2u wp=4u lp=2u
*.opin y
*.ipin a
m2 net1 a VCC VCC cmosp w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u' m=1
m1 net1 a 0 0 cmosn w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u' m=1
m3 y net1 VCC VCC cmosp w=wp l=lp ad='wp *4.6u' as='wp *4.6u' pd='wp *2+9.2u' ps='wp *2+9.2u' m=1
m4 y net1 0 0 cmosn w=wn l=lln ad='wn *4.3u' as='wn *4.3u' pd='wn *2+8.6u' ps='wn *2+8.6u' m=1
.ends

.end
```

This approach allows to create polymorphic symbols. Multiple parameters may be given to the generator script, like **`symbolgen.tcl(inv,hv,100)`**. Xschem will call the symbolgen.tcl script with the following command: **`symbolgen.tcl inv hv 100`** and take the standard output from the script as the symbol file to load and display.

Schematic generators (pcells)

The same approach used for symbol generators can be used for schematic generators. If you add a **`schematic=schematicgen.tcl(buf,4)`** attribute to an instance xschem will look for a script named **`schematicgeni.tcl`** in the search paths and call it with the given parameters (that is, execute the command **`schematicgen.tcl buf 4`**) and read the produced output as a schematic file.



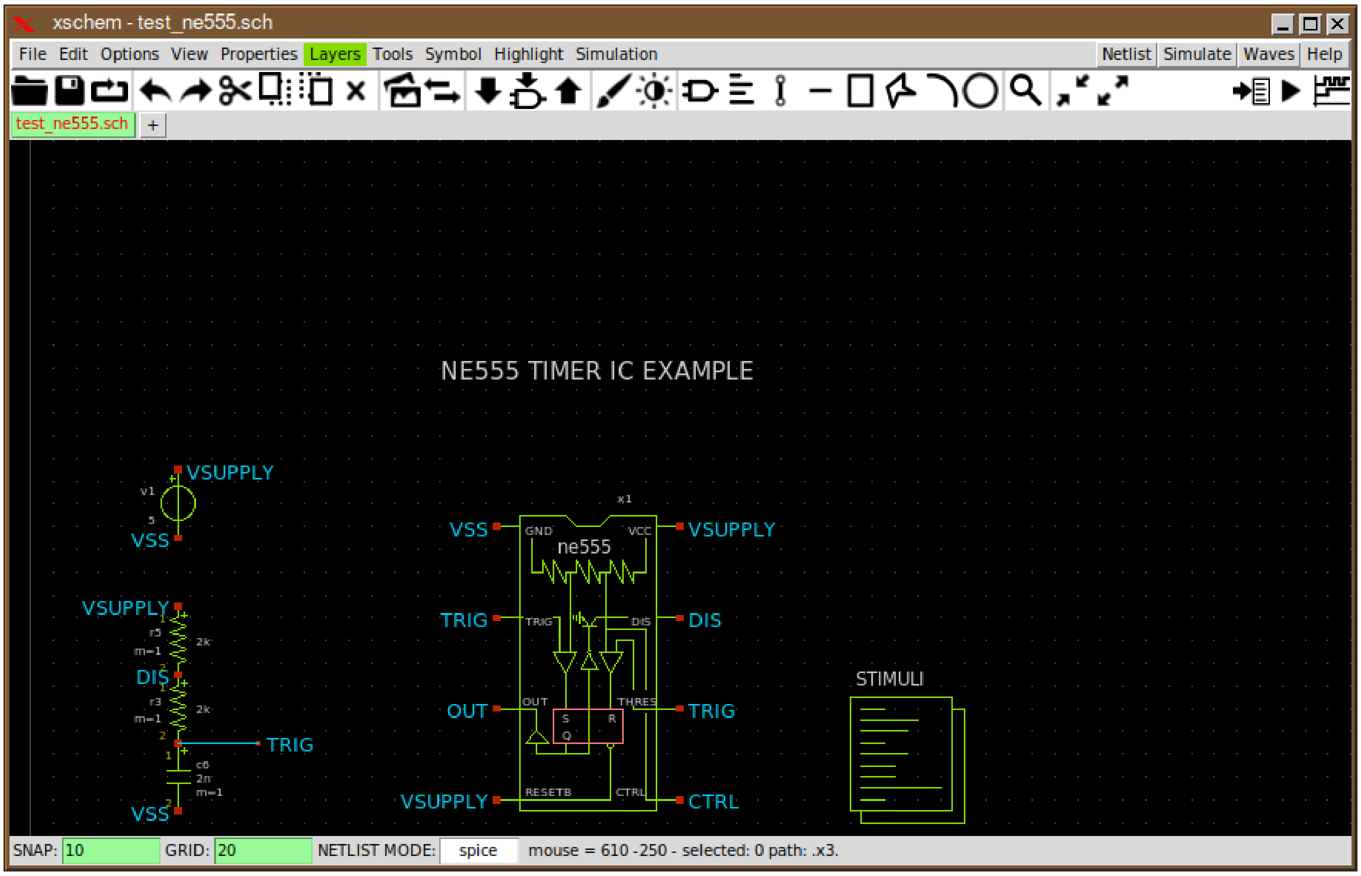
```
name=x4 ROUT=1000
schematic="schematicgen.tcl(buf,4)"
tclcommand="edit_file [abs_sym_path schematicgen.tcl]"
```

```
 models_rom8k.txt
cc=3
 0 dc 3
 pwl 0 0 100n 0 100.1n

ll
n 300n uic
 test_symbolgen.raw
```

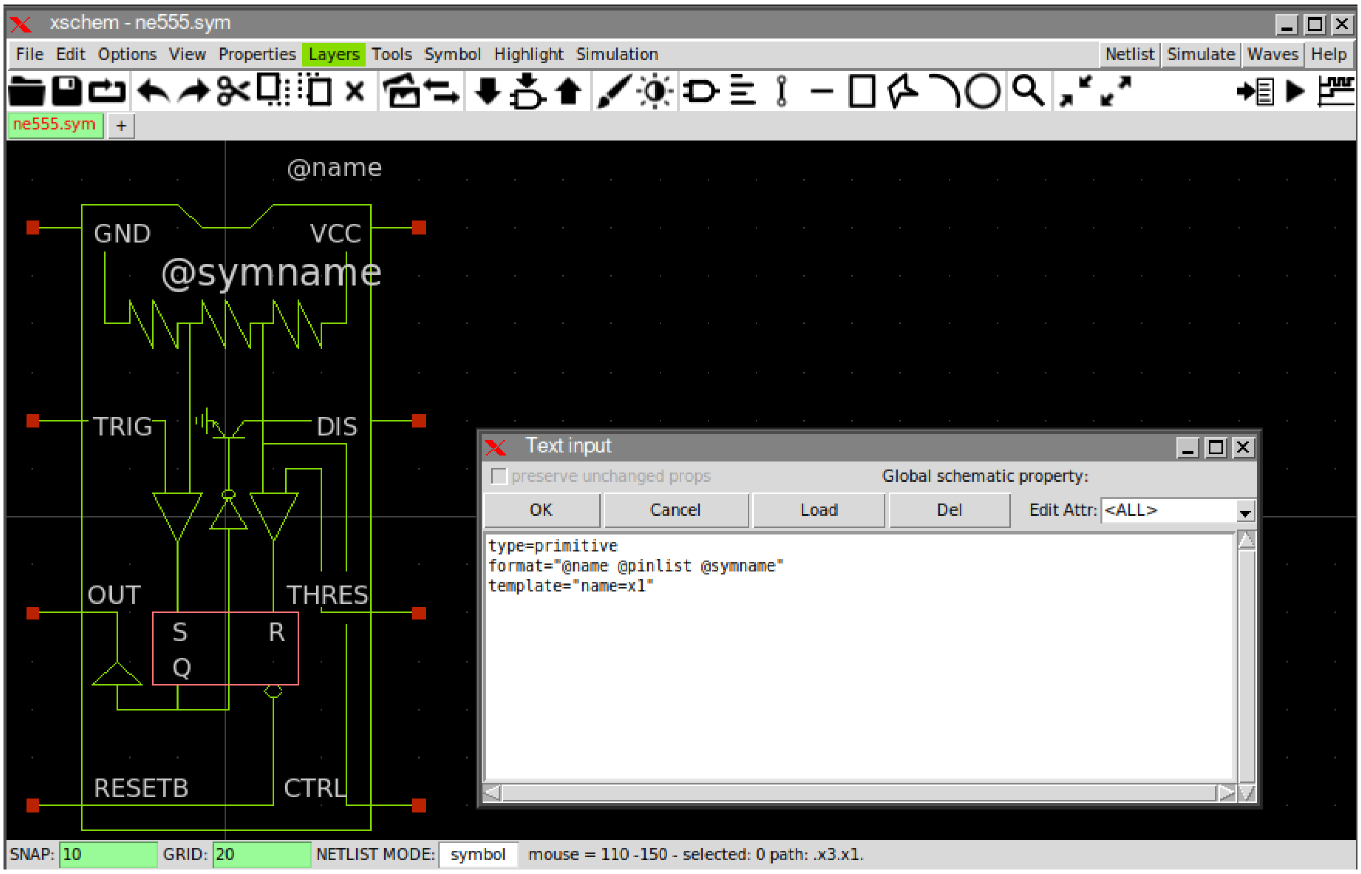
UP

TUTORIAL: CREATE A SYMBOL AND USE AN EXISTING NETLIST

In some cases you have an existing netlist for a circuit block, perhaps from a previous design or from a layout parasitic netlist extraction. In order to use this netlist in your design you might consider creating a symbol for it in xschem. This symbol should match the I/O interface and name of the block netlist and does not need to have a corresponding schematic since we want to use the existing netlist. One such example in the standard xschem distribution is the test_ne555.sch circuit. The test schematic contains a symbol for the popular NE555 timer. The symbol does not provide any implementation, the implementation is included in the top design as a .subckt netlist.


xschem - test_ne555.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_ne555.sch
NE555 TIMER IC EXAMPLE
VSUPPLY
VSS
DIS
TRIG
OUT
CTRL
ne555
x1
GND
VCC
RESETB
THRES
STIMULI
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 610 -250 - selected: 0 path: .x3.

The symbol is implemented in the following way: the symbol attributes are:

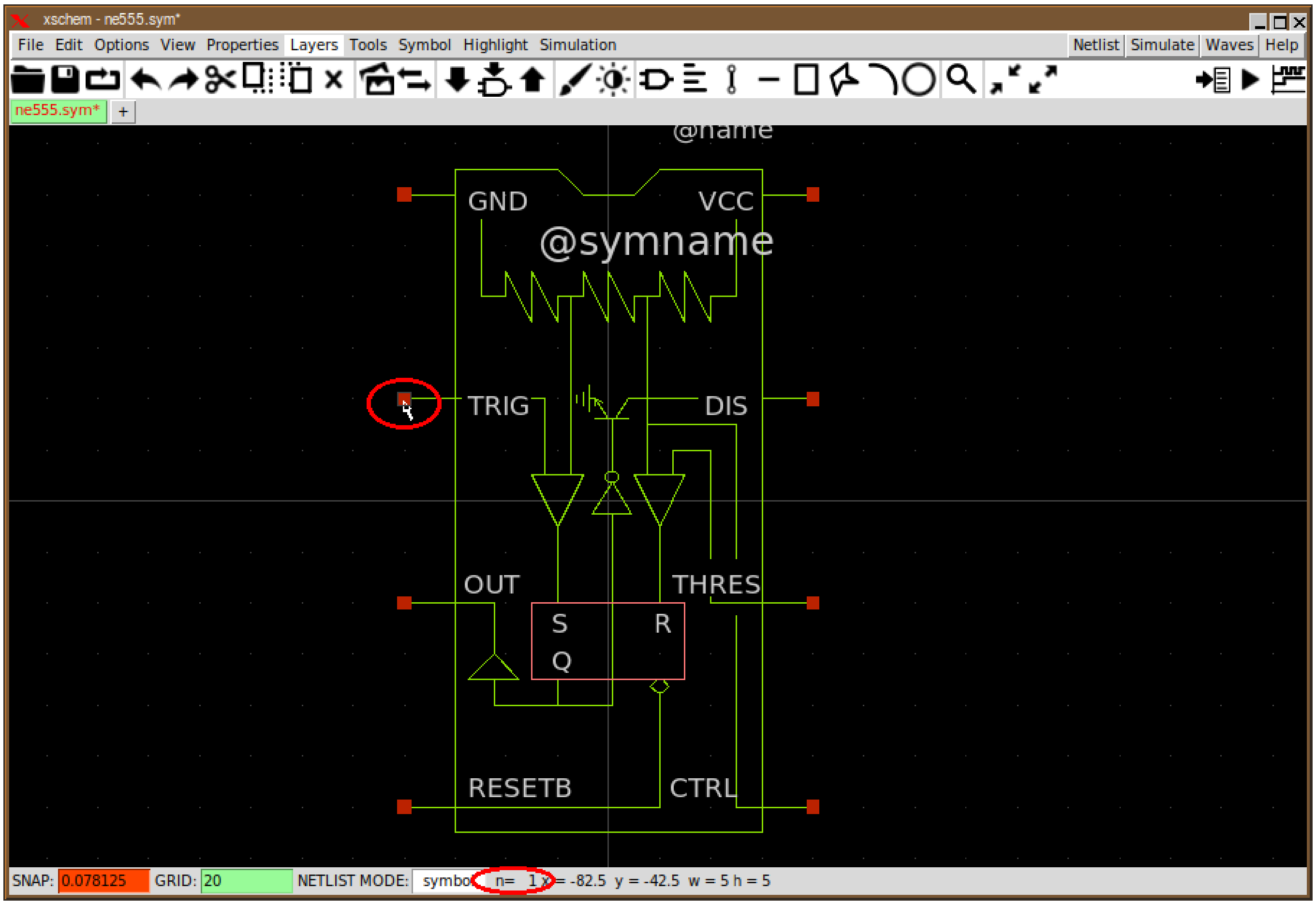


```
type=primitive
format="@name @pinlist @symname"
template="name=x1"
```

```
type=primitive
format="@name @pinlist @symname"
template="name=x1"
```

the **`primitive`** value for the **`type`** attribute (instead of the more used **`subcircuit`** for symbols with a corresponding implementation schematic) tells xschem to generate only the instance calls (the X lines for spice netlists) and not descend into the symbol and not generate a .subckt for it.

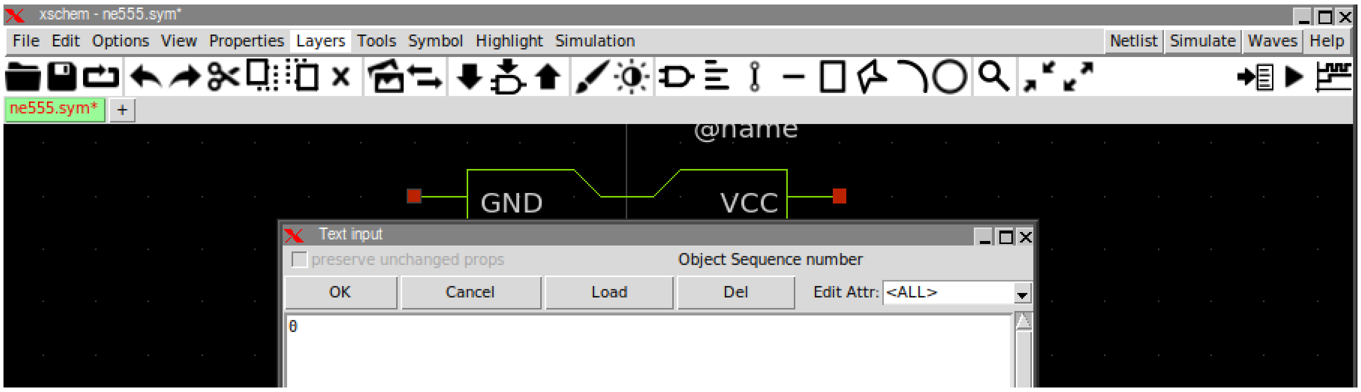
The **`@pinlist`** is expanded into the netlist to the list of I/O ports. The order of the ports in this case is the order these pins are created in the symbol. If you click a pin (the small red square box) a "**`n = <number>`**" appears in the status line. This is the index of the pin. The first created pin starts from 0.


xschem - ne555.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
ne555.sym*
@name
GND
VCC
@symname
TRIG
DIS
OUT
THRES
S
R
Q
RESETB
CTRL

SNAP: 0.078125 GRID: 20 NETLIST MODE: symbol n= 1 x= -82.5 y = -42.5 w = 5 h = 5

1. Changing the pin ordering by altering the object sequence number

You can change the order the pins are stored into the .sym file. Start by clicking the pin that you want to have first in the netlist, then press **Shift-s**, set the number to 0.



Object Sequence number

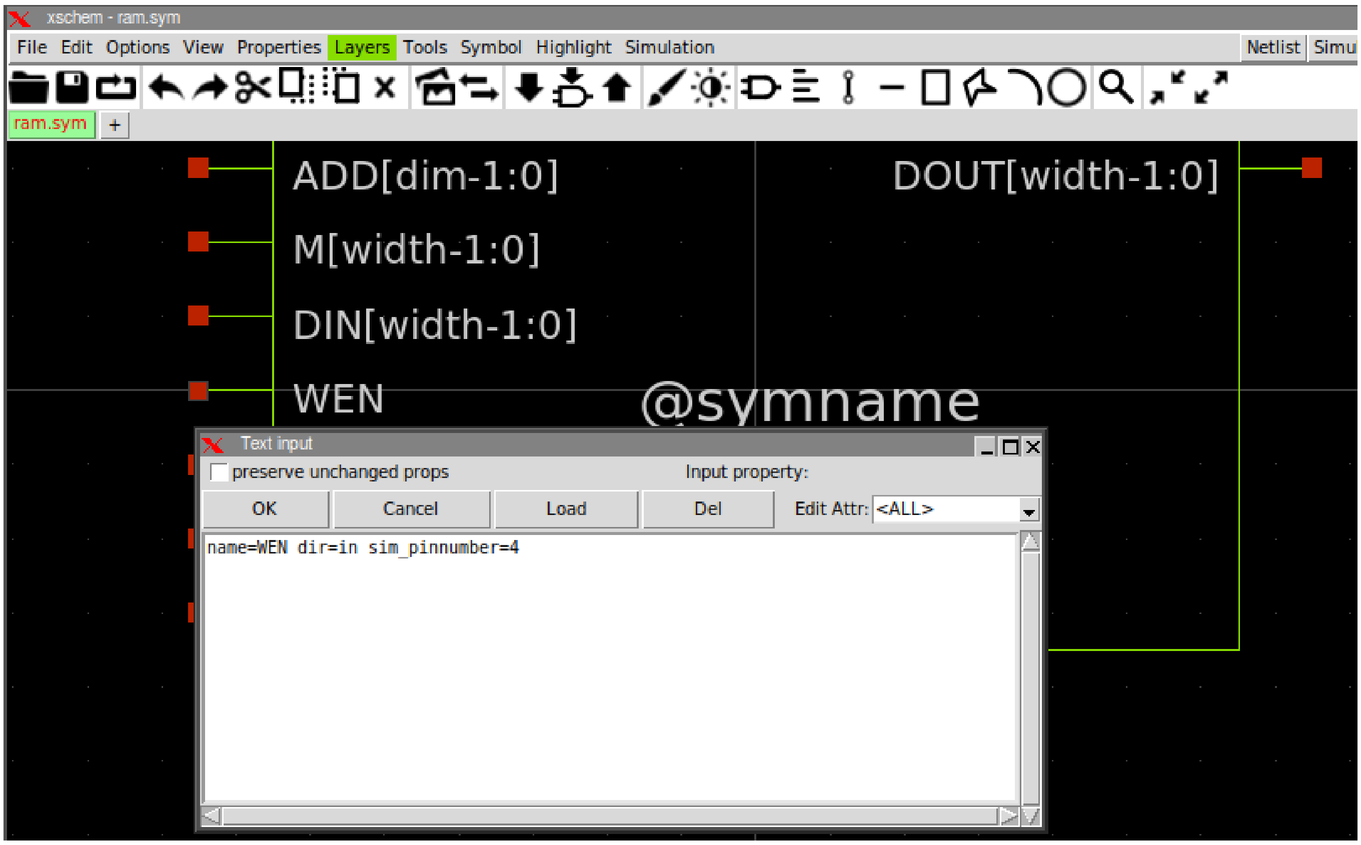
```
0
```

This will put the selected pin in first position. Then move to the pin you want in second position, repeat above steps and assign to it index number 1, and so on for all the symbol pins. At the end save your symbol and this will be the pin ordering in netlists. When netlist is produced this order will be used. If left pins in above example have sequence numbers of (starting from the top) 0, 1, 2, 3 and right pins have sequence numbers (starting from the bottom) 4, 5, 6, 7 the instance line in the netlist will be (check the net names with the schematic in the first image above):

```
x1 VSS TRIG OUT VSUPPLY CTRL TRIG DIS VSUPPLY ne555
```

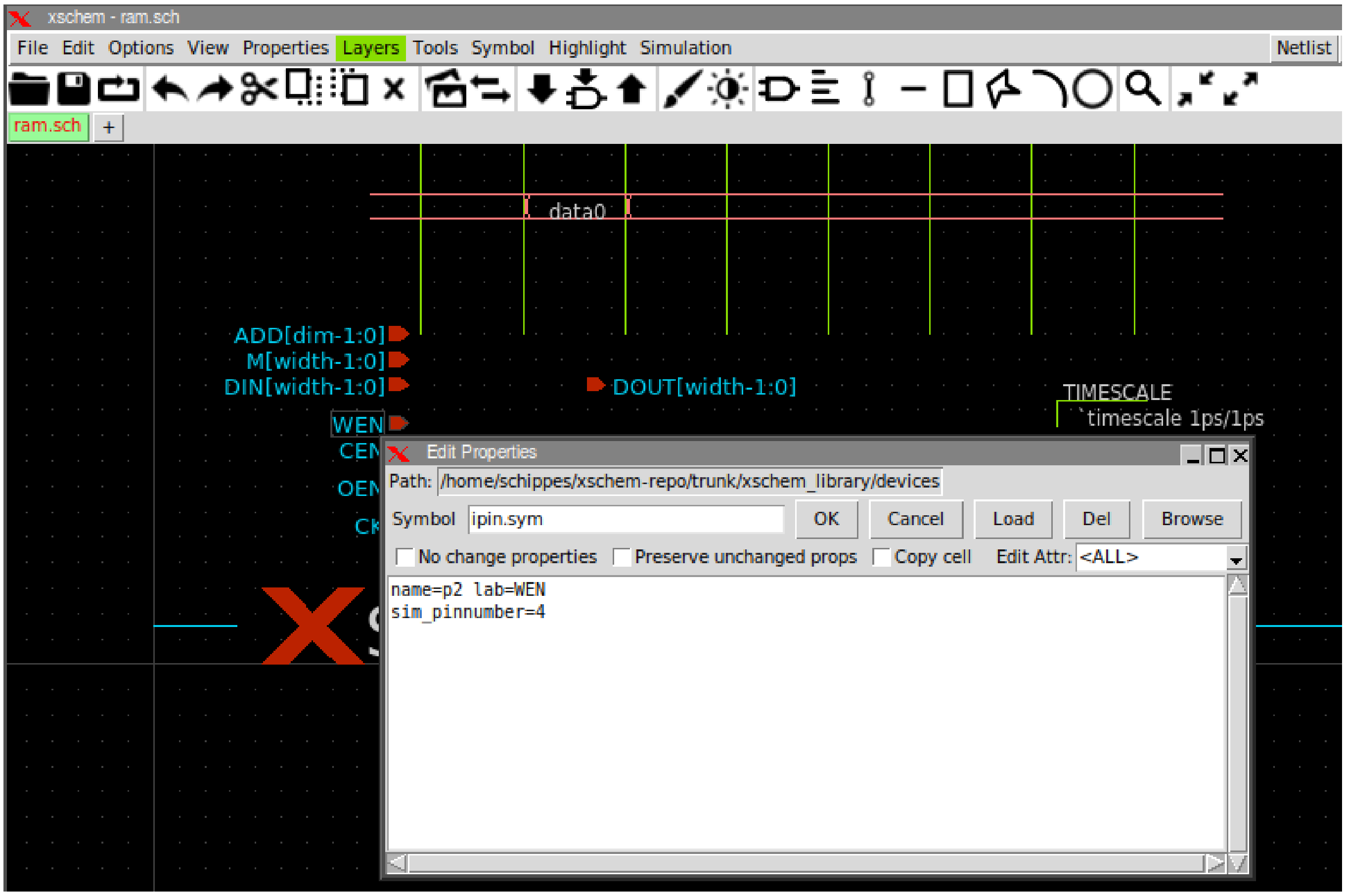
2. Changing the pin ordering by using the `sim_pinnumber` attribute

If all symbol pins have a **`sim_pinnumber`** attribute this symbol will be netlisted (in all netlist formats) with pins sorted in ascending order according to **`sim_pinnumber`** value. Start value of sim_pinnumber does not matter (may start at 1 or 0) , it is used as the sort key. You can assign the sim_pinnumber attribute directly in the symbol...



```
name=WEN dir=in sim_pinnumber=4
```

... Or you can assign these in the schematic pins, if you use the **`Make symbol from schematic`** function ('a' key) these attributes will be transferred to the symbol. The **`sim_pinnumber`** attributes that determine the netlist port ordering are those defined in the symbol.



```
name=p2 lab=WEN
sim_pinnumber=4
```

For sorting to happen all symbol pins must have a sim_pinnumber attribute. If some pins miss this attribute no sorting is done and pin ordering will be unchanged, the stored order of symbol pins will be used (first created pin netlisted first). If there are duplicate sim_pinnumber attributes (but all pins have this attribute) sorting will happen but relative ordering or pins with identical sim_pinnumber is undefined.
As an example you may give **`sim_pinnumber=9999`** on a symbol output and **`sim_pinnumber=1`** on all other pins if you only require the output pin to be netlisted at the end and don't care about the other pin ordering.

3. **Explicitly specify port ordering in `format` (or `verilog_format` or `vhdl_format`) string**

Instead of the following format string that defines the netlist instance line syntax:

```
format="@name @pinlist @symname"
```

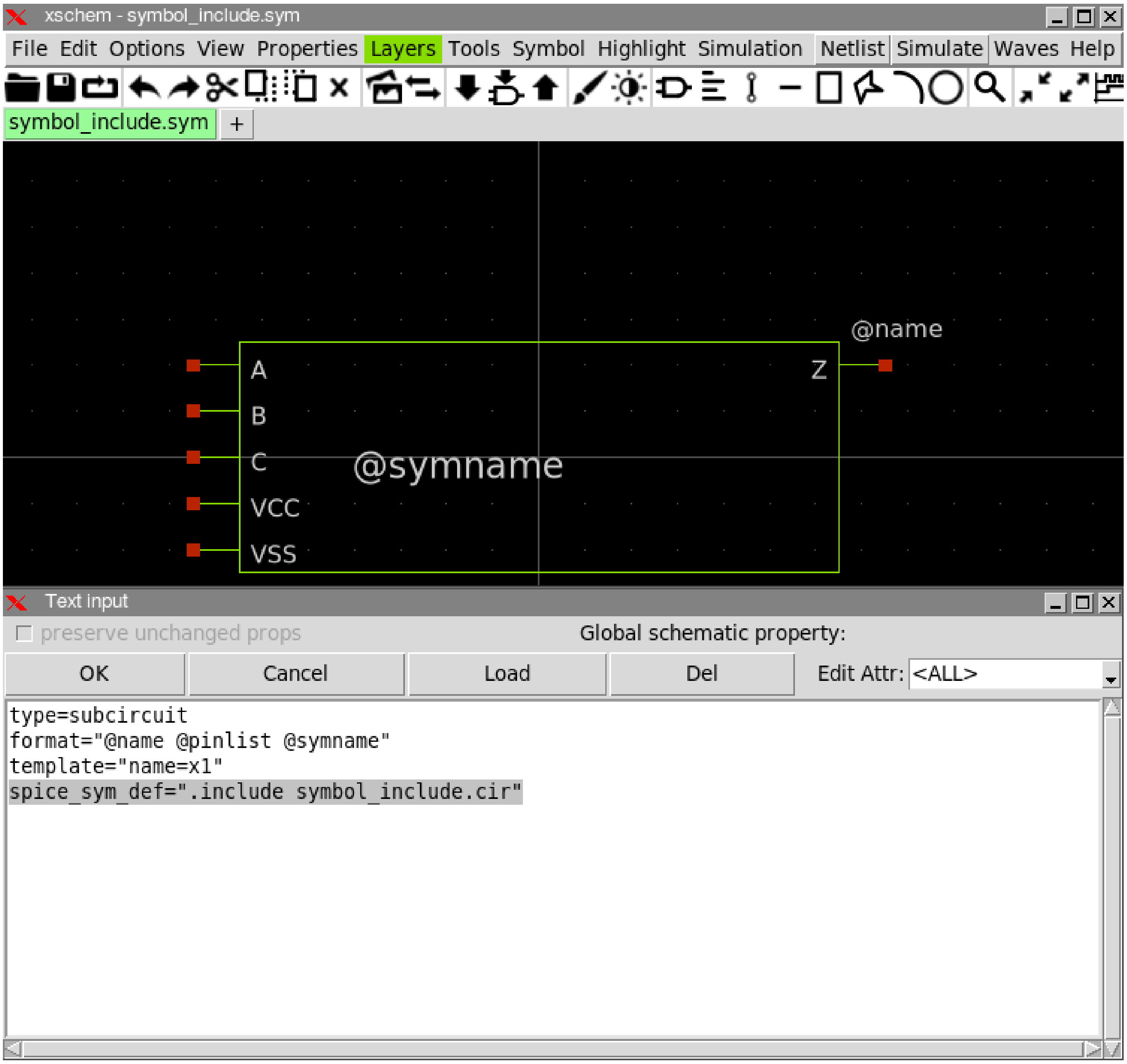
You can use the following:

```
format="@name @@GND @@TRIG @@OUT @@RESETB @@CTRL @@THRES @@DIS @@VCC @symname"
```

In this case you specify the port order one by one explicitly. This can be used for spice primitive devices, spice subcircuits (like this example), VHDL and Verilog primitives. This method can NOT be used for VHDL and verilog subcircuits since for these you do not provide a **`vhdl_format`** or **`verilog_format`** string. For these use one of the first two methods. In general for VHDL and Verilog port order is not important since port-net association is **named** and not **positional**.

Obtaining the pin ordering from the subcircuit definition specified via `spice_sym_def`

For spice netlists if **`@pinlist`** is specified in format string and a symbol **`spice_sym_def`** attribute is used then the order of the symbol ports will be obtained from the **`.subckt`** specified by **`spice_sym_def`**, either directly or via a **`.include`** statement


xschem - symbol_include.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Simulate Waves Help
symbol_include.sym
+
@name
A
B
C
VCC
VSS
Z
@symname
Text input
preserve unchanged props
Global schematic property:
OK
Cancel
Load
Del
Edit Attr:
<ALL>

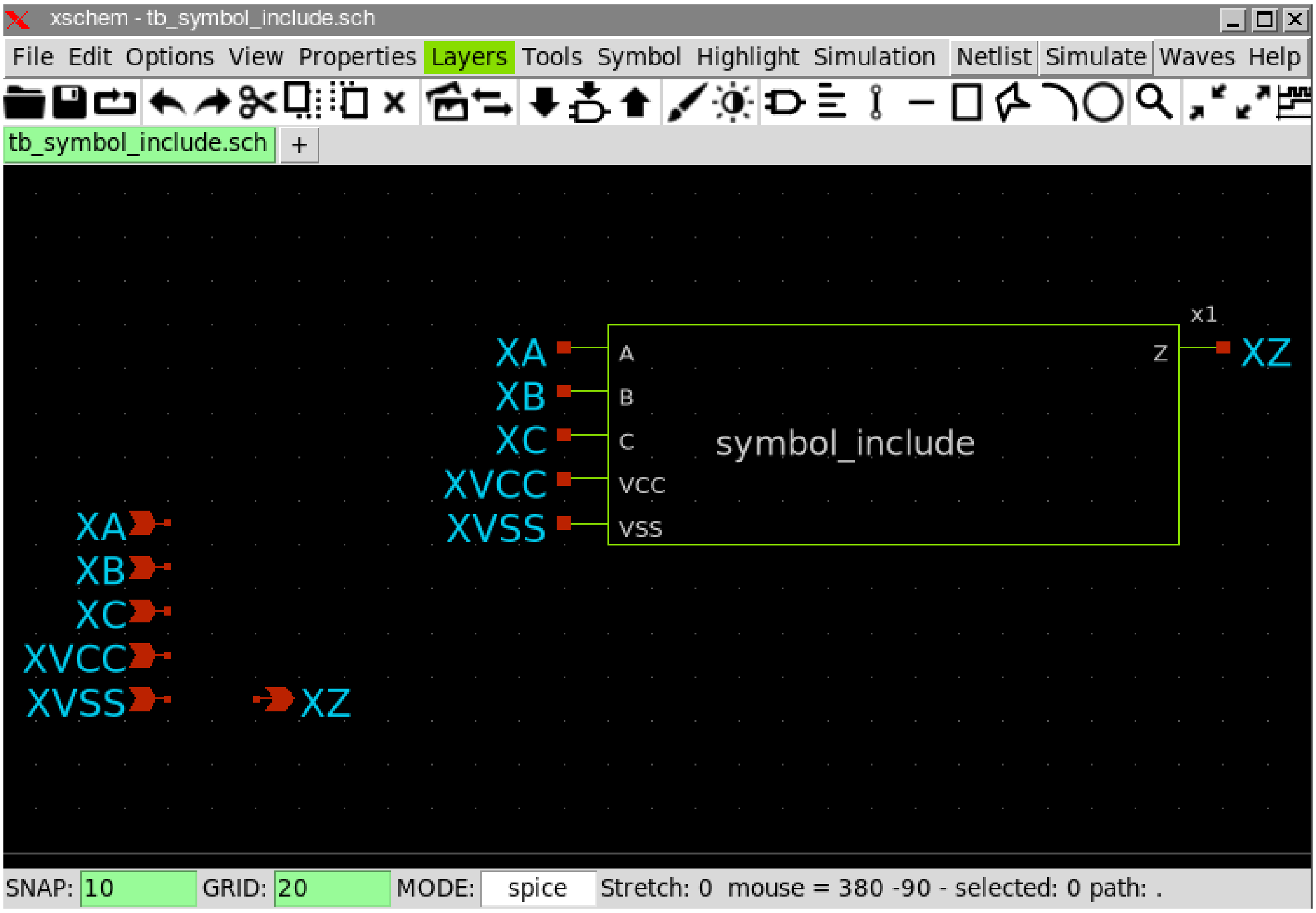
```
type=subcircuit
format="@name @pinlist @symname"
template="name=x1"
spice_sym_def=".include symbol_include.cir"
```

The **`symbol_include.cir`** file has the following content:

```
* example of a subcircuit contained in a file

.subckt symbol_include Z VCC VSS
+ A B C W=10 L=1
...
...
.ends
```

4.

And as a result the following circuit:


xschem - tb_symbol_include.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Simulate Waves Help
tb_symbol_include.sch
+
x1
XA
XB
XC
XVCC
XVSS
A
B
C
VCC
VSS
Z
XZ
symbol_include
XA
XB
XC
XVCC
XVSS
XZ
SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 380 -90 - selected: 0 path: .

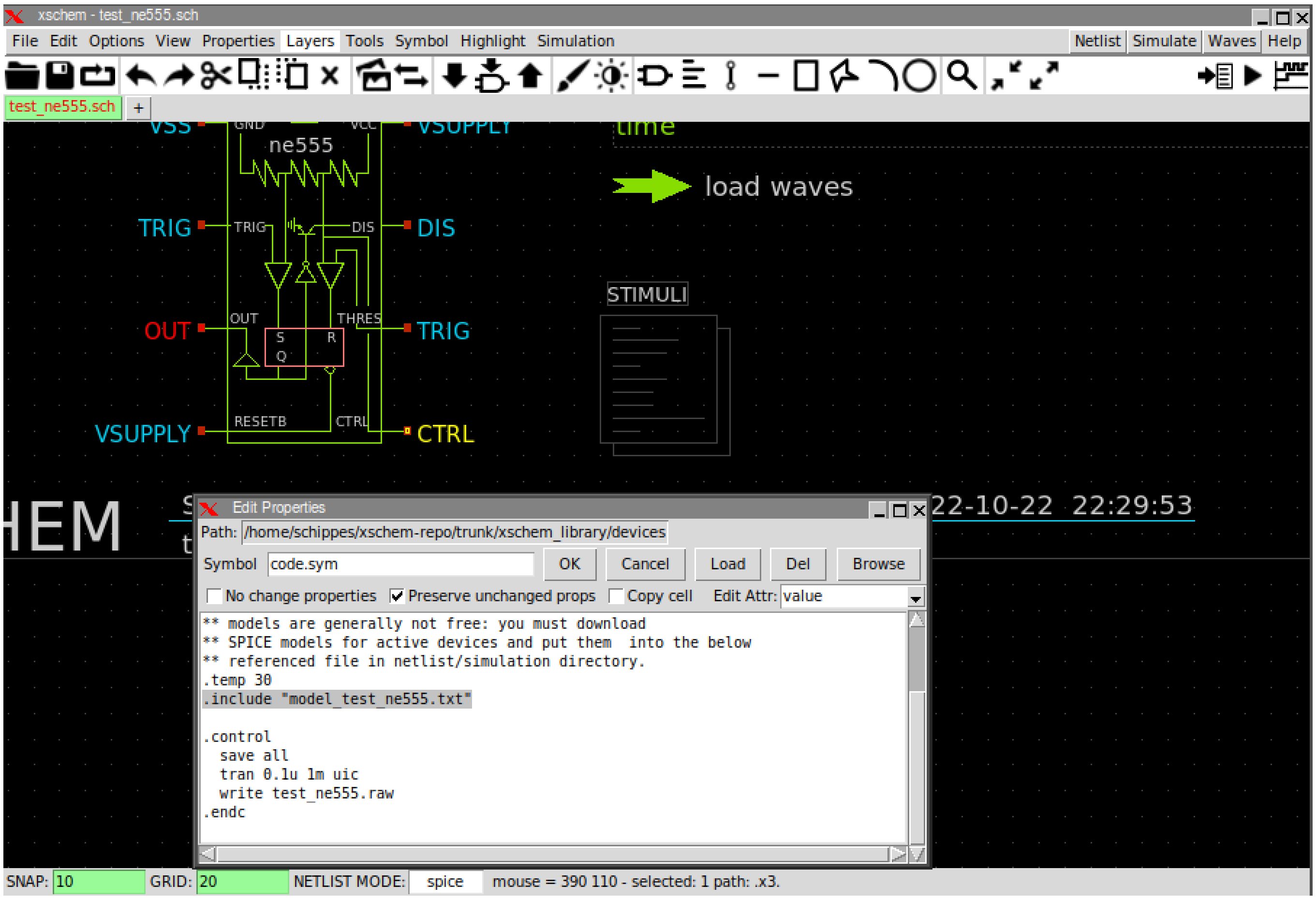
is netlisted in the following way, notice the net assignment in the **x1** subcircuit call matches the order in the **symbol_include.cir** file:

```
** sch_path: /home/schippes/.xschem/xschem_library/symbol_include/tb_symbol_include.sch
**.subckt tb_symbol_include XZ XVSS XVCC XC XB XA
*.opin XZ
*.ipin XVSS
*.ipin XVCC
*.ipin XC
*.ipin XB
*.ipin XA
x1 XZ XVCC XVSS XA XB XC symbol_include
**.ends

* expanding   symbol:  symbol_include.sym # of pins=6
** sym_path: /home/schippes/.xschem/xschem_library/symbol_include/symbol_include.sym
.include symbol_include.cir
.end
```

Specifying subcircuit netlist

1. Add a `.include <file>` line in the top level

The first method is to declare the symbol as **`type=primitive`** (this is the case in all images above) and simply add a **`.include /path/to/subcircuit.spice`** in the top level netlist:



Path: /home/schippes/xschem-repo/trunk/xschem_library/devices

Symbol: code.sym

☐ No change properties ☑ Preserve unchanged props ☐ Copy cell Edit Attr: value

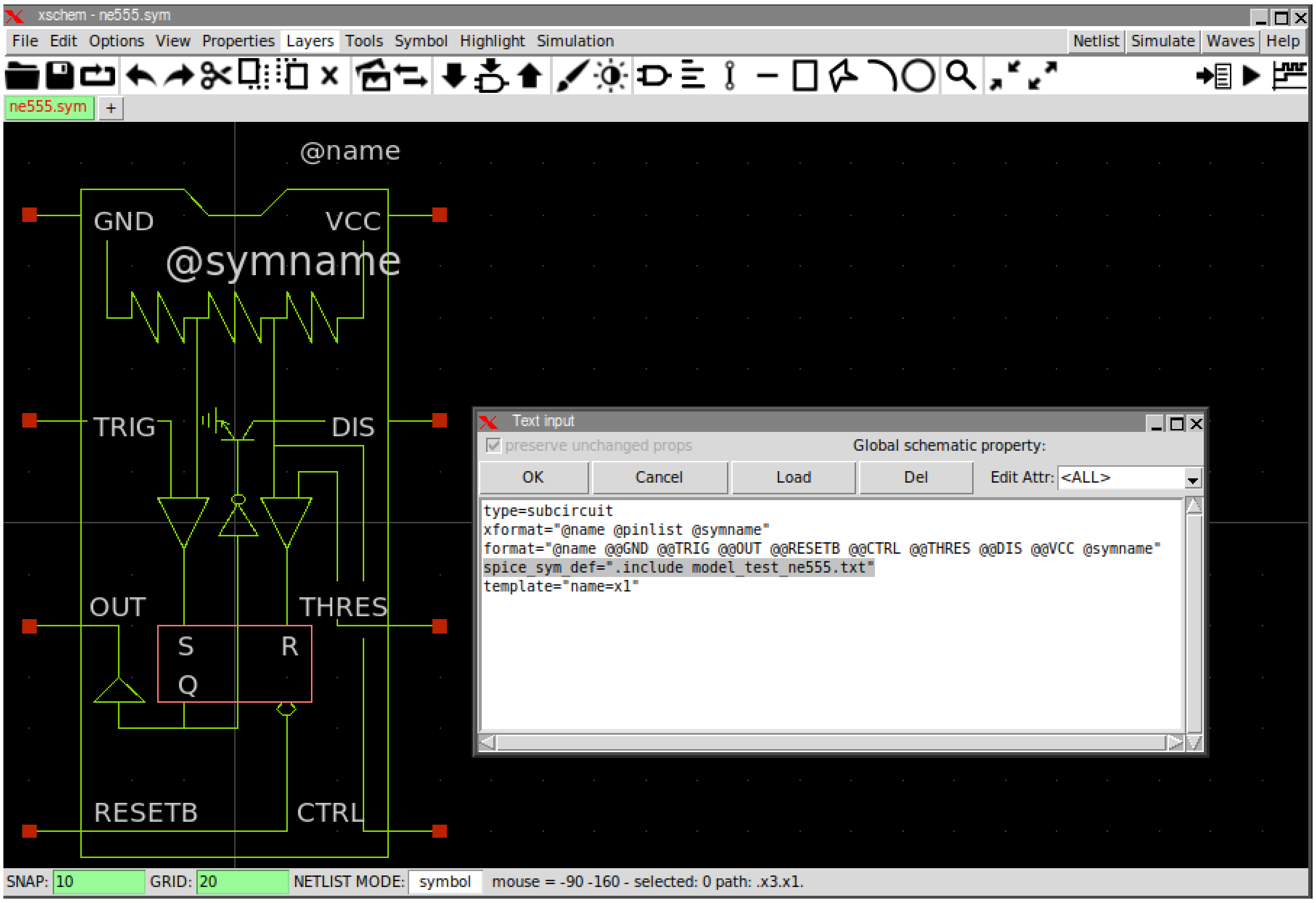
```
** models are generally not free: you must download
** SPICE models for active devices and put them  into the below
** referenced file in netlist/simulation directory.
.temp 30
.include "model_test_ne555.txt"

.control
  save all
  tran 0.1u 1m uic
  write test_ne555.raw
.endc
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 390 110 - selected: 1 path: .x3.

2. Use a `spice_sym_def=".include <file>"` line in the symbol

The second method is to declare the symbol **`type`** as **`subcircuit`** and add a **`spice_sym_def`** attribute in the symbol. the value of this attribute will be copied verbatim to the netlist, so for the example shown here this should do the job:

```
spice_sym_def=".include model_test_ne555.txt"
```



```
type=subcircuit
xformat="@name @pinlist @symname"
format="@name @@GND @@TRIG @@OUT @@RESETB @@CTRL @@THRES @@DIS @@VCC @symname"
spice_sym_def=".include model_test_ne555.txt"
template="name=x1"
```

The produced netlist will be:

```
** sch_path: /home/schippes/xschem-repo/trunk/xschem_library/examples/test_ne555.sch
**.subckt test_ne555
x1 VSS TRIG OUT VSUPPLY CTRL TRIG DIS VSUPPLY ne555
...
...
* expanding   symbol:  ne555.sym # of pins=8
** sym_path: /home/schippes/xschem-repo/trunk/xschem_library/examples/ne555.sym
.include model_test_ne555.txt
.end
```

The advantage of this method is that the reference of the subcircuit is embedded in the symbol and if the symbol is reused in another design the .include line travels with the symbol and you don't have to add the line in the top level.

3. **Completely specify a subcircuit in the `format` attribute of the symbol**

The following set of symbol attrtibutes:

```
type=source
format="X@name @@in @@out sub_@name
.subckt sub_@name in out
@name out 0 V=@func
.ends sub_@name"
template="name=B1 FUNC="pow(V(in),2)""
```

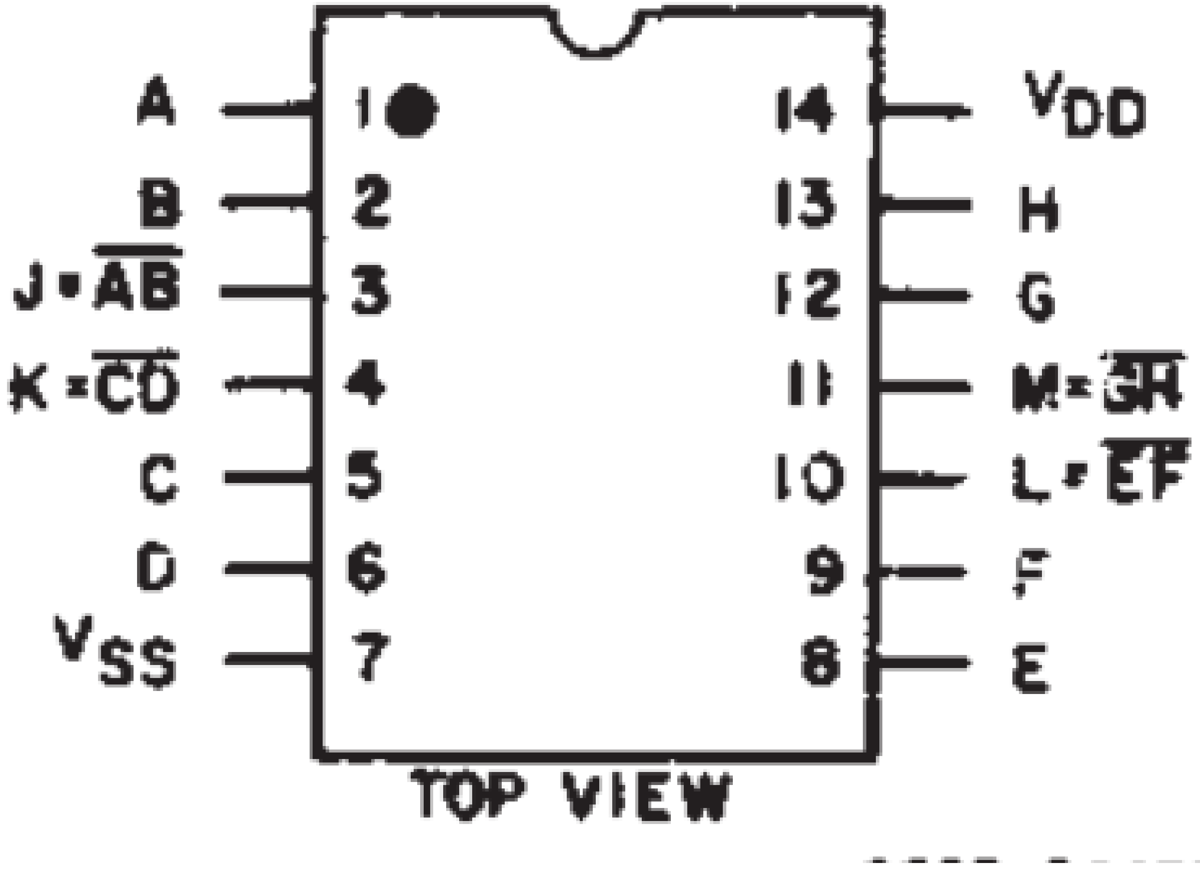
will create a sub_xxx subcircuit with a unique name for every symbol instance using the @name attribute (which is indeed unique). This allows to build subcircuits with arbitrary parameters (a math expression in the example).

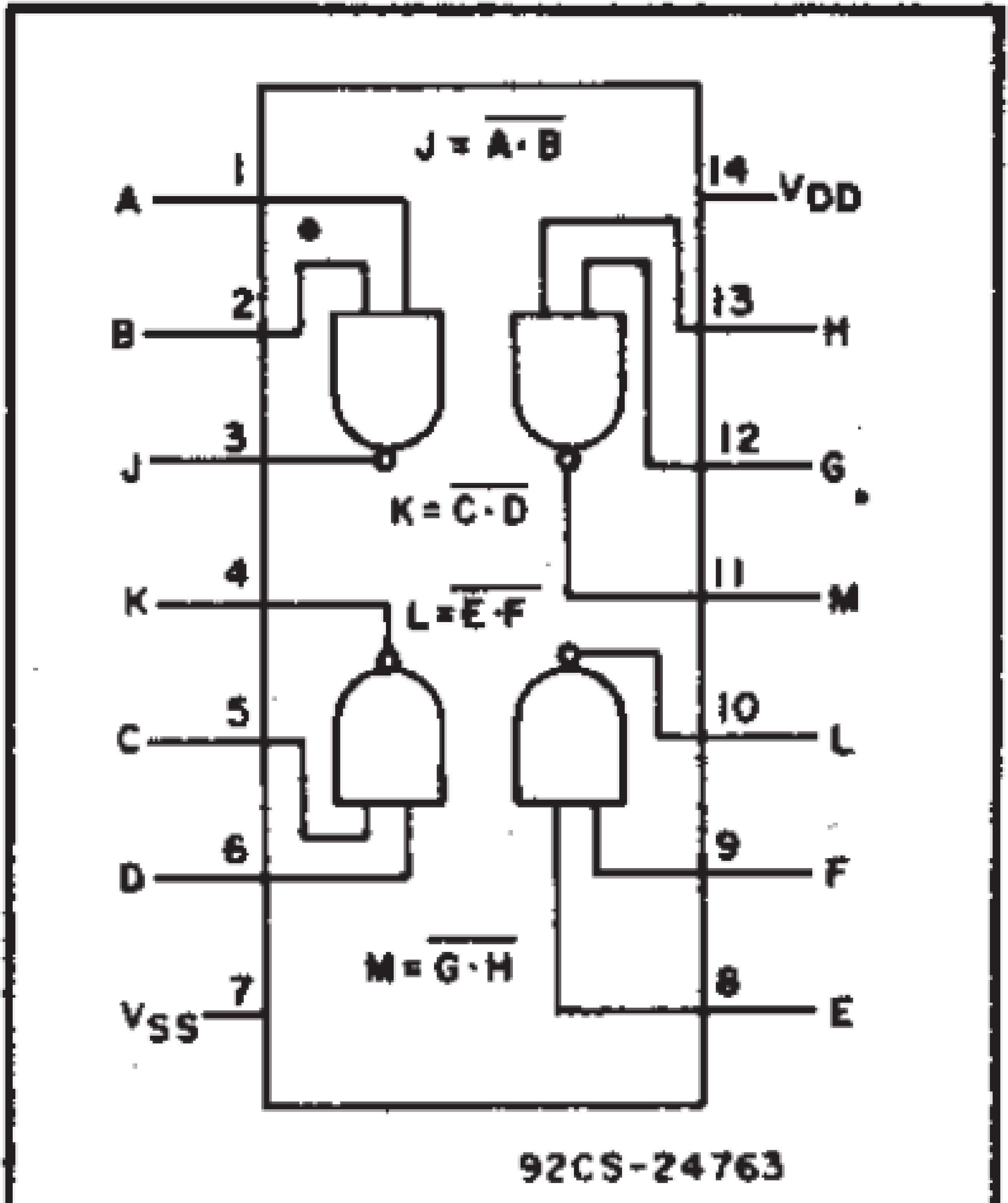
The problem of this approach is that it works by creating nested .subckt inside the parent schematic (which could itself be a .subckt). Not all simulators support this (although Ngspice and Xyce seem to work OK with this).

UP

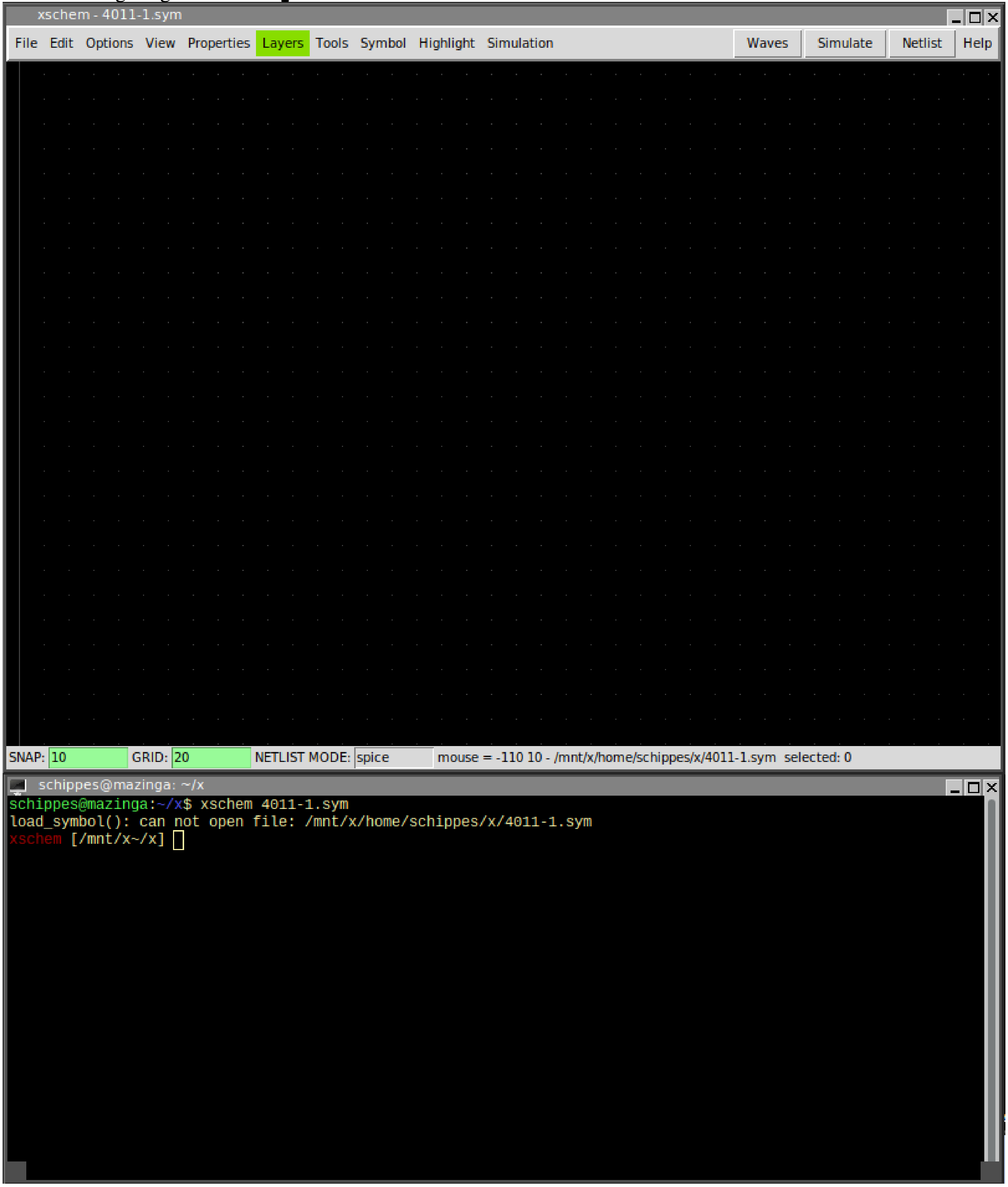
TUTORIAL: CREATE AN XSCHEM SYMBOL

In this tutorial we will build a 4011 CMOS quad 2-input NAND symbol. This IC has 4 nand gates (3 pins each, total 4*3=12 pins + VDD,VSS power pins) This device comes in a dual in line 14 pin package.




J = $\overline{A \cdot B}$
K = $\overline{C \cdot D}$
L = $\overline{E \cdot F}$
M = $\overline{G \cdot H}$
A
B
J
K
C
D
V_{SS}
1
2
3
4
5
6
7
14
13
12
11
10
9
8
V_{DD}
H
G
M
L
F
E
92CS-24763
CD4011B
FUNCTIONAL DIAGRAM

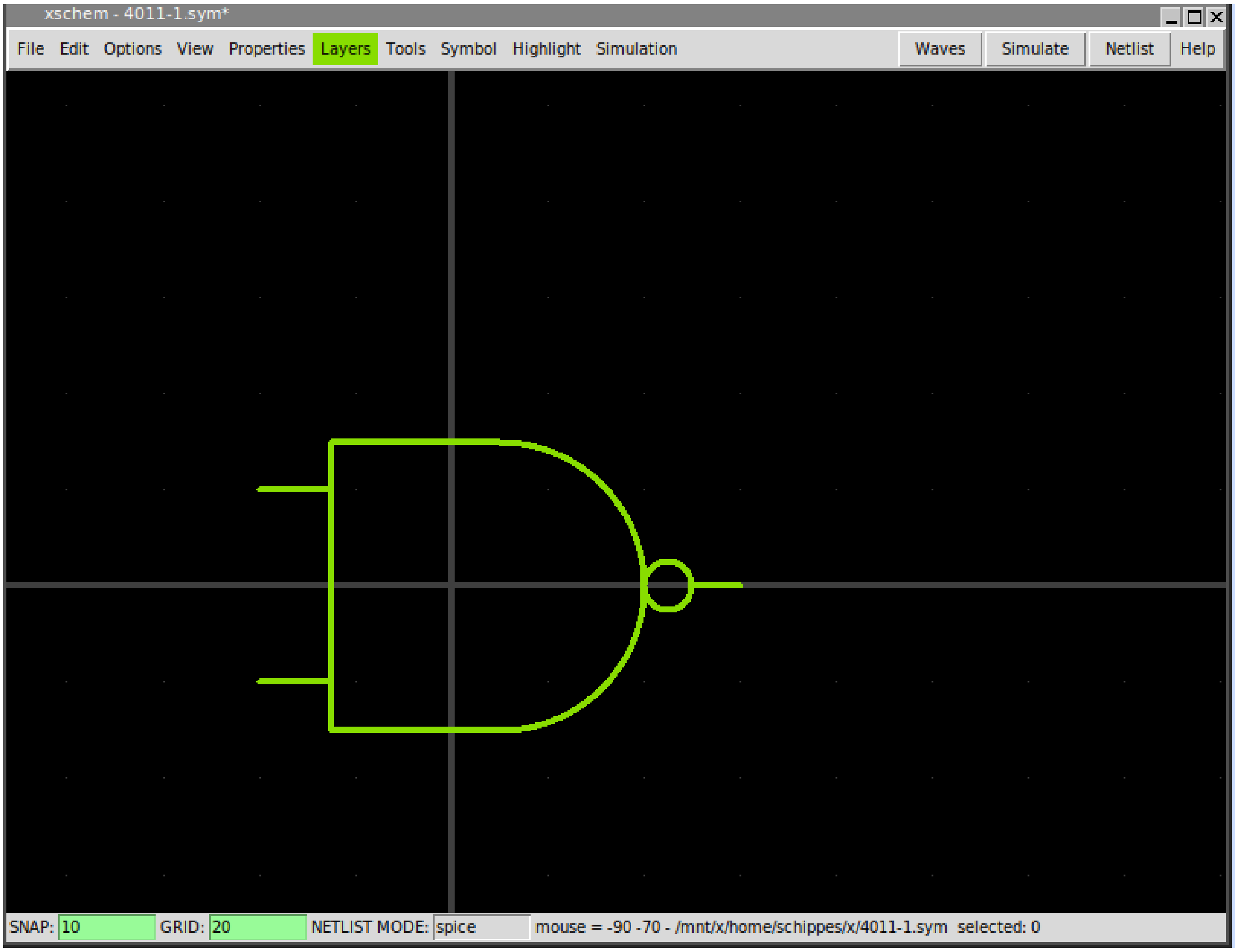
1. Start xschem giving **`4011-1.sym`** as filename:


xschem - 4011-1.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
schippes@mazinga: ~/x

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -110 10 - /mnt/x/home/schippes/x/4011-1.sym selected: 0

```
schippes@mazinga:~/x$ xschem 4011-1.sym
load_symbol(): can not open file: /mnt/x/home/schippes/x/4011-1.sym
xschem [/mnt/x~/x]
```

2. use layer 4 (the default) to draw the following shapes, use **`l`** to draw lines and use **`Shift-c`** to draw arcs, use **`Ctrl-Shift-c`** to draw circles. Arcs and circles are drawn by specifying start - end point and a 3rd way point. You will need to change the grid snap to '5' for drawing the smallest objects using the **`g`** key. Be sure to restore the grid snap to the default value with **`Shift-g`** as soon as you are done. Also ensure that the gate terminals are on grid with the default '10' snap setting. Use the **`m`** key after selecting objects to move them around.

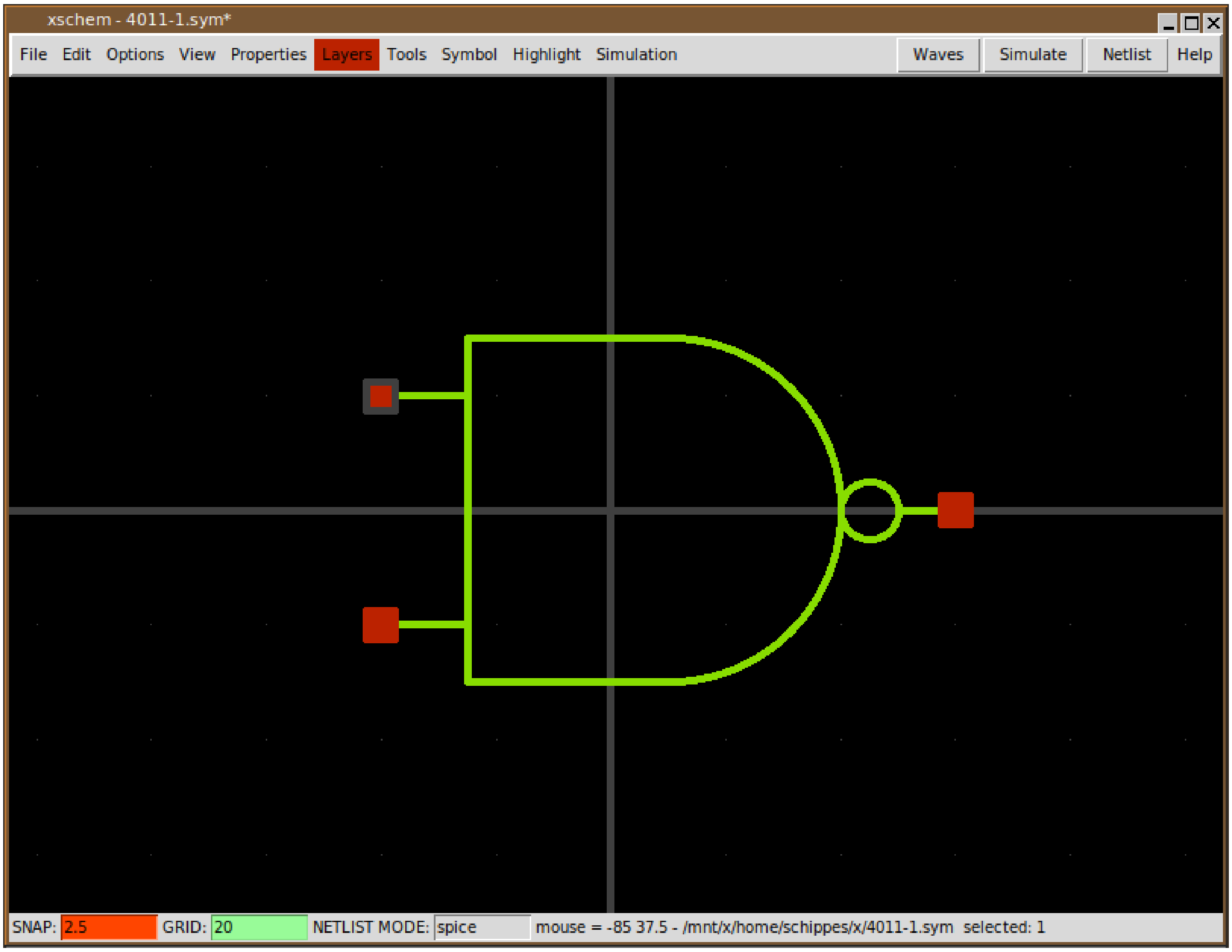

xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -90 -70 - /mnt/x/home/schippes/x/4011-1.sym selected: 0

Do **NOT** forget to reset the grid setting to the default (10) value as soon as you finished drawing small objects, otherwise the rest of the objects will be all off grid making the symbol unusable

3. Create pins, select layer 5 from the **`Layers`** menu. Set grid snap to 2.5 to allow drawing small rectangles centered on gate terminals. Start from the 'A' input of the nand gate (we assume A to be the left-top input), then the 'B' input (the lower left input terminal), then the 'Z' output (the right terminal). If you click and hold the mouse selecting the rectangles the 'w' and 'h' dimensions are shown. They should be equal to 5. remember to reset the grid to default 10 when done.

   Update: a more advanced command is now available to place a symbol pin: **`Alt-p`**


xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help

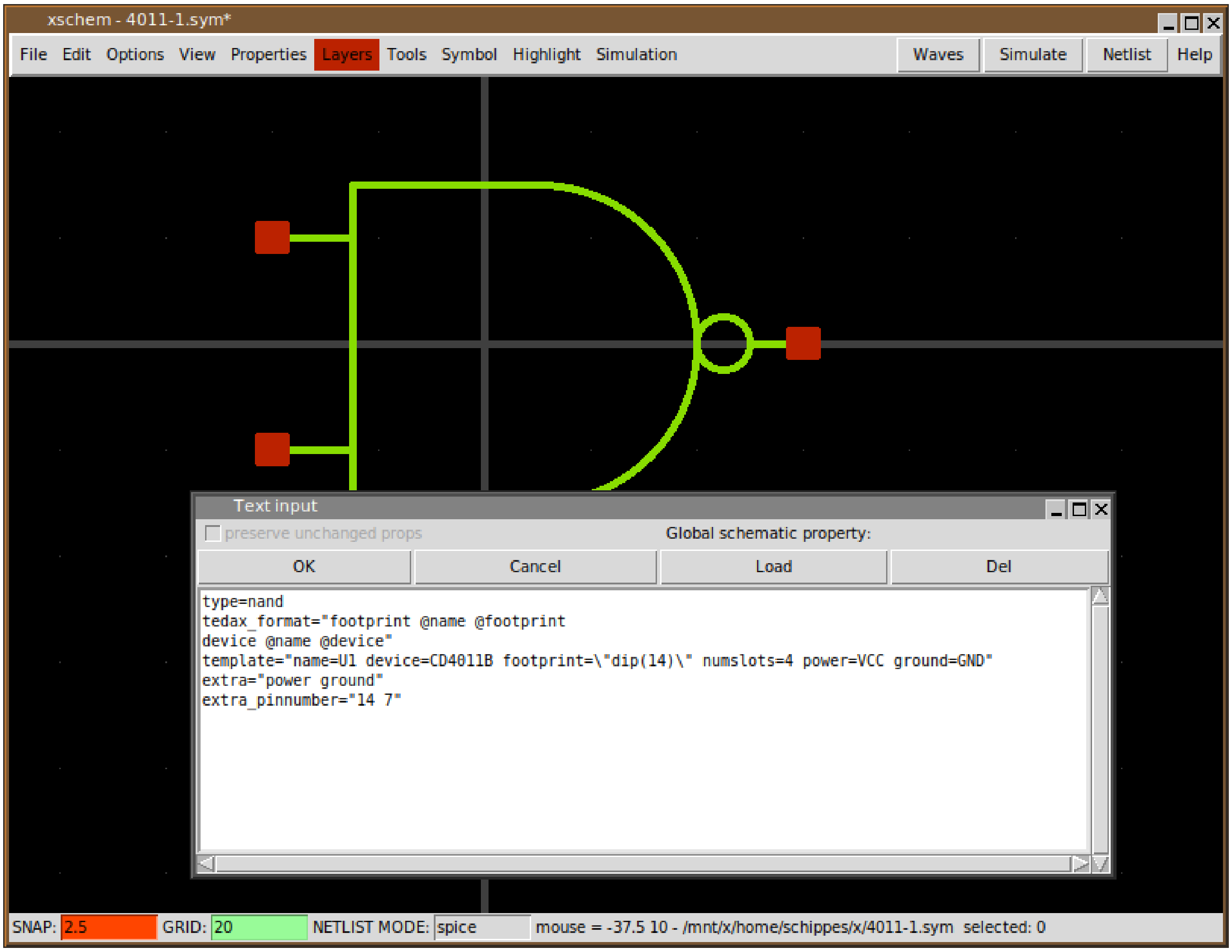
SNAP: 2.5 GRID: 20 NETLIST MODE: spice mouse = -85 37.5 - /mnt/x/home/schippes/x/4011-1.sym selected: 1

4. Now when **no object is selected** press **q** to edit the symbol global attributes. Type the following text:

```
type=nand
tedax_format="footprint @name @footprint
device @name @device"
template="name=U1 device=CD4011B footprint=\"dip(14)\" numslots=4 power=VCC ground=GND"
extra="power ground"
extra_pinnumber="14 7"
```

Instead of the **q** key the attribute dialog box can also be displayed by **double clicking** the left mouse button

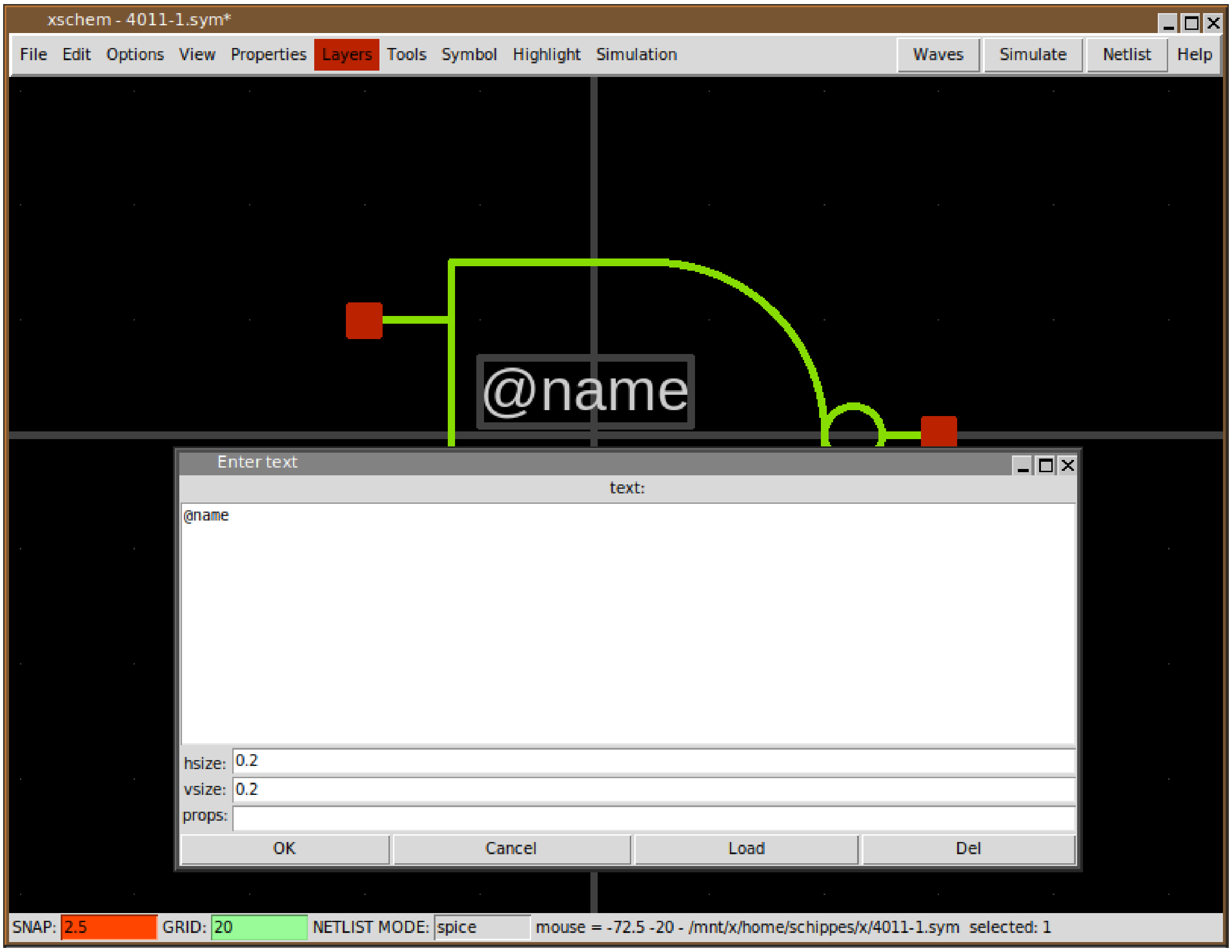
these attributes specify the gate type, the format for tedax netlist, the **template** attribute specifies default values for attributes and defines pin connection for VDD and VSS that are associated to package pins 14 and 7. The **device** attribute specifies the component name to be used in the tEDAx netlist (this is usually the name of the IC as shown in the datasheet). The **extra** and **extra_pinnumber** attributes specify extra pin connections that are implicit, not drawn on the symbol. This is one of the possible styles to handle power connections on slotted devices.


xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
Text input
preserve unchanged props
Global schematic property:
OK Cancel Load Del

```
type=nand
tedax_format="footprint @name @footprint
device @name @device"
template="name=U1 device=CD4011B footprint=\"dip(14)\" numslots=4 power=VCC ground=GND"
extra="power ground"
extra_pinnumber="14 7"
```

SNAP: 2.5 GRID: 20 NETLIST MODE: spice mouse = -37.5 10 - /mnt/x/home/schippes/x/4011-1.sym selected: 0

5. Press the **`t`** to place some text; set text v and h size to 0.2 and write **`@name`**; this will be replaced with the instance name (aka refdes) when using the symbol in a schematic. Place a similar string with text **`@symname`** and place it under the @name string.

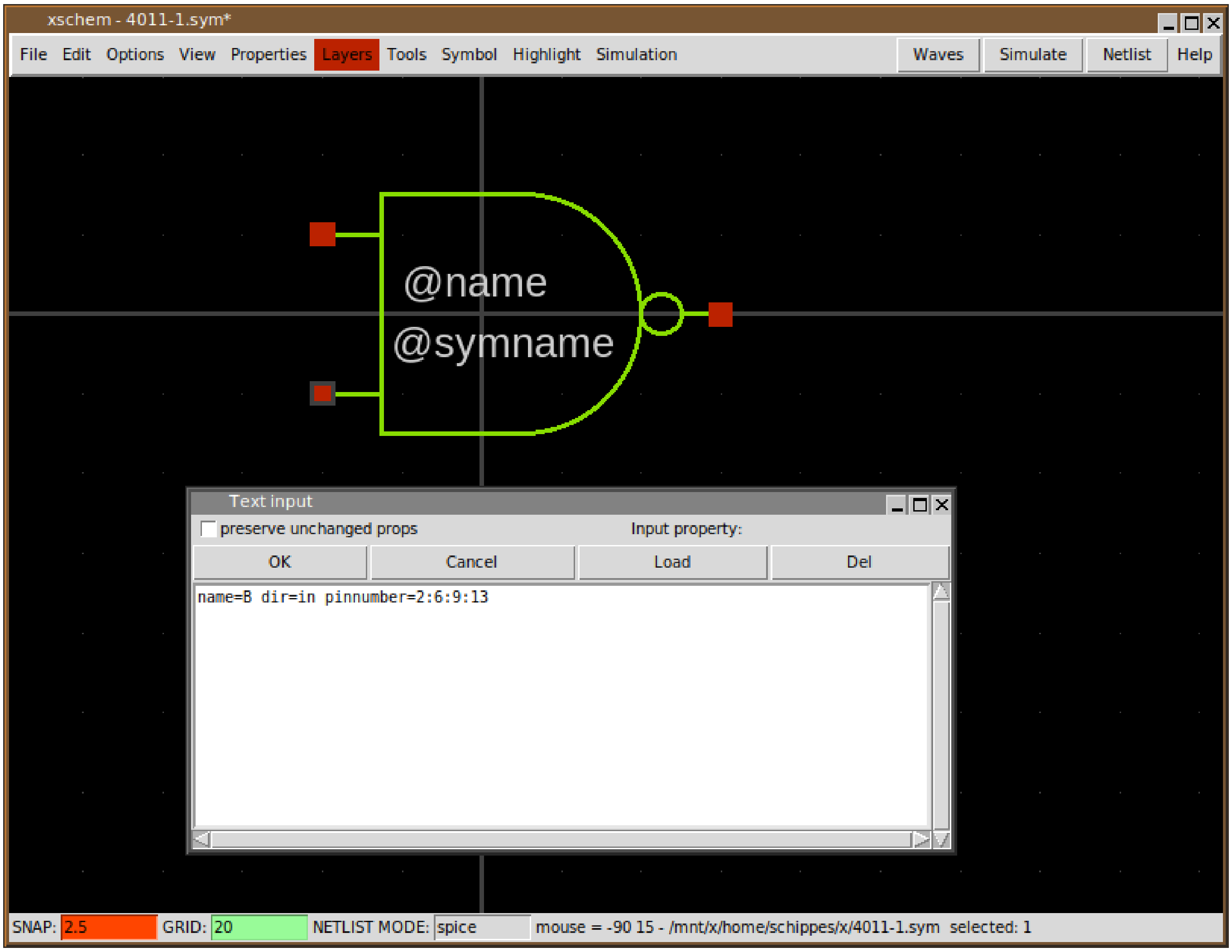

xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@name
Enter text
text:
OK Cancel Load Del

```
@name
```

hsize: 0.2
vsize: 0.2
props:

SNAP: 2.5 GRID: 20 NETLIST MODE: spice mouse = -72.5 -20 - /mnt/x/home/schippes/x/4011-1.sym selected: 1

6. select the red pins (click the mouse close to the interior side of the rectangle corners) and press **`q`**, set attribute **`name=A dir=in pinnumber=1:5:8:12`** for the upper left pin, **`name=B dir=in pinnumber=2:6:9:13`** for the lower left pin, **`name=Z dir=out pinnumber=3:4:10:11`** for the right output pin. As you can see pin numbers 7 and 14 are missing from the list of pins; they used for VSS and VDD power supplies, which are implicit (no explicit pins). Since we are creating a slotted device (an IC containing 4 identical nand gates) the **`pinnumber`** attribute for each pin specifies the pin number for each slot, so the following: **`name=A dir=in pinnumber=1:5:8:12`** specifies that pin A of the nand gate is connected to package pin 1 for nand slot 1, to package pin 5 for nand slot 2 and so on.i The **`dir`** attribute specifies the direction of the pin; XSCHEM supports **`in`**, **`out`** and **`inout`** types. These attributes are used mainly for digital simulators (Verilog and VHDL), but specifying pin direction is good practice anyway.

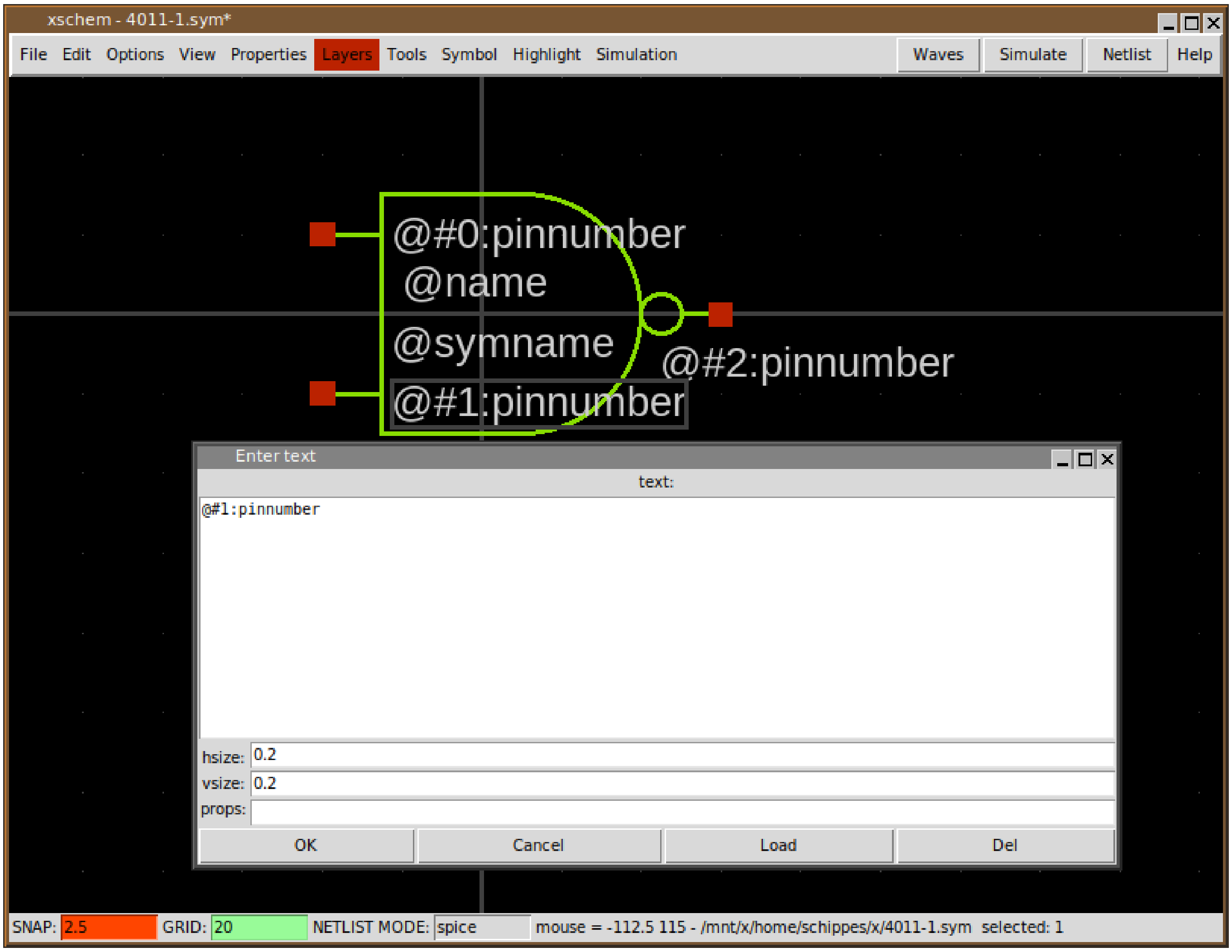

xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Waves Simulate Netlist Help
@name
@symname
Text input
preserve unchanged props
Input property:
OK Cancel Load Del

```
name=B dir=in pinnumber=2:6:9:13
```

SNAP: 2.5 GRID: 20 NETLIST MODE: spice mouse = -90 15 - /mnt/x/home/schippes/x/4011-1.sym selected: 1

Instead of the **`q`** key the attribute dialog box can also be displayed by placing the mouse pointer over the pin object and pressing the **`right`** mouse button

7. We want now to place some text near the gate pins to display the pin number: again, use the **`t`** key and place the following text, with hsize and vsize set to 0.2:


xschem - 4011-1.sym*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
@#0:pinnumber
@name
@symname
@#1:pinnumber
@#2:pinnumber
Enter text
text:
hsize:
vsize:
props:
OK Cancel Load Del
SNAP: 2.5 GRID: 20 NETLIST MODE: spice mouse = -112.5 115 - /mnt/x/home/schippes/x/4011-1.sym selected: 1

text: @#1:pinnumber

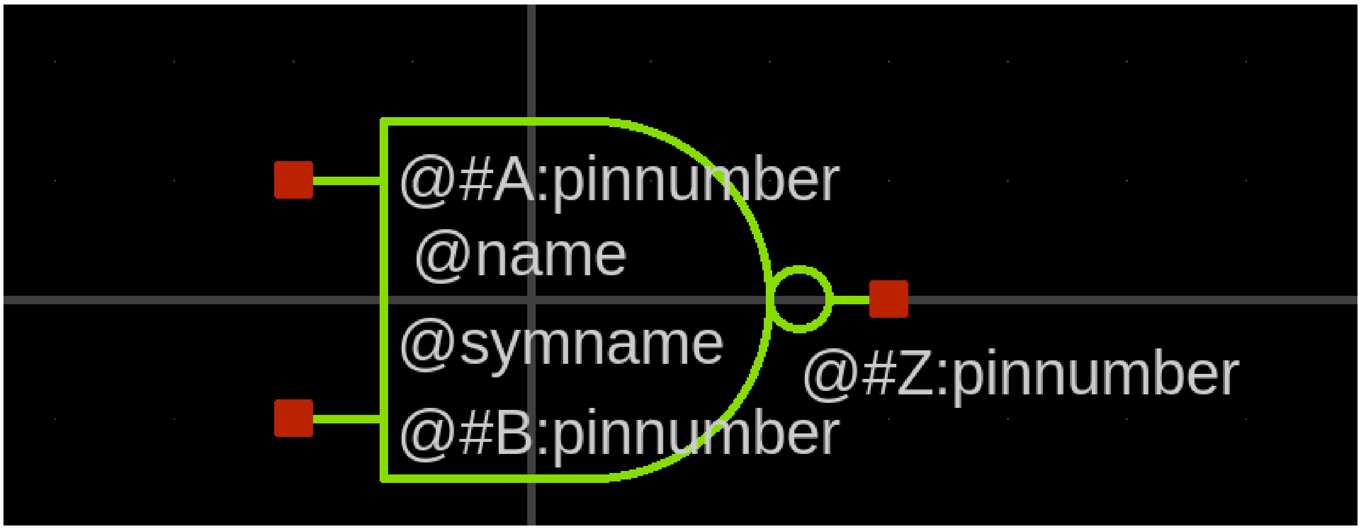
hsize: 0.2

vsize: 0.2

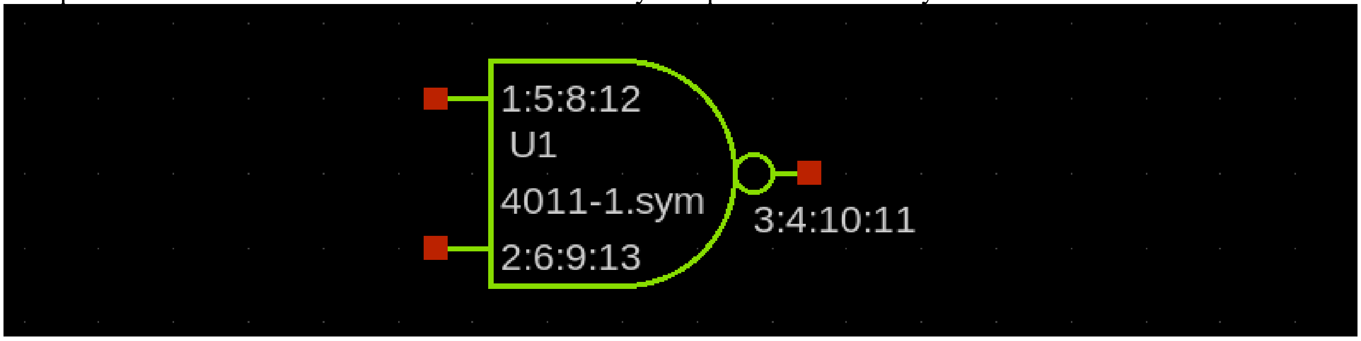
The complicated syntax of these text labels has the following meaning:

- The `@` is the variable expansion (macro) identifier, as usual.
- The `#0` specifies pin with index 0, this is the first pin we have created, the upper left nand input. The index of a pin can be viewed by selecting the pin and pressing **`Shift-s`**.
- The **`pinnumber`** specifies the attribute we want to be substituted with the actual value when placing the gate in a schematic as we will see shortly.

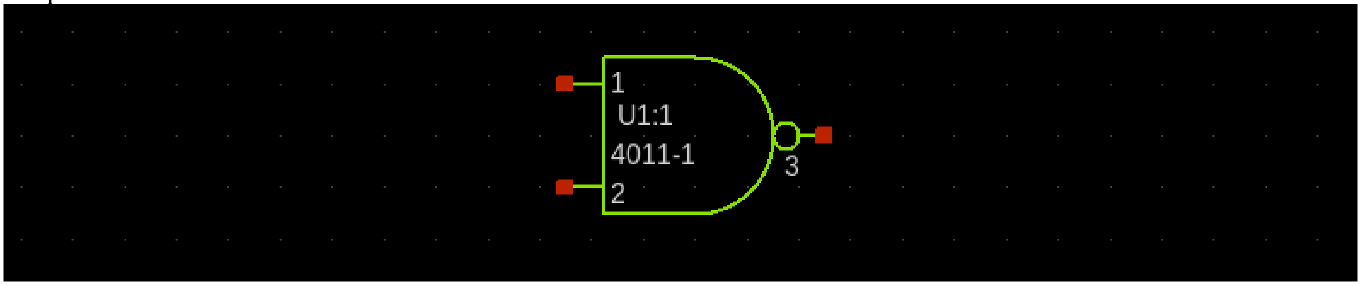
8. There is another syntax that can be used to display pin numbers, instead of specifying the pin index in XSCHEM list (that reflects the creation order) you can reference pins by their name; The only reason to use the previous syntax with pin index numbers is efficiency when dealing with extremely big symbols (SoC or similar high pin count chips).



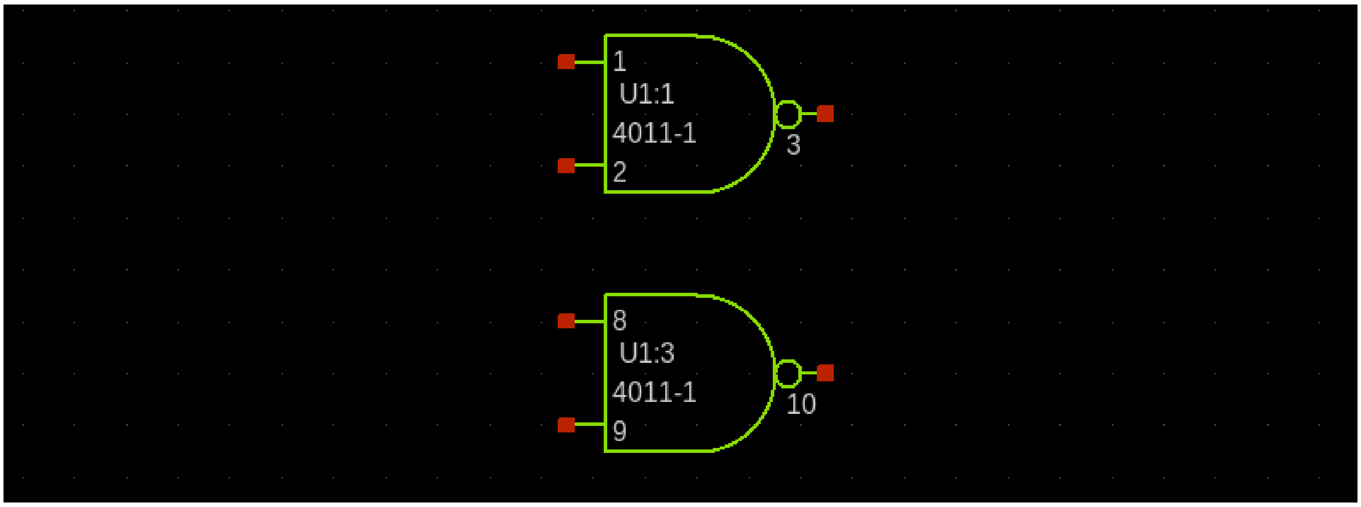
9. The symbol is now complete; save it and close XSCHEM. Now open again xschem with an empty schematic, for example **`xschem test.sch`**. Press the **`Insert`** key and place the 4011-1 symbol:



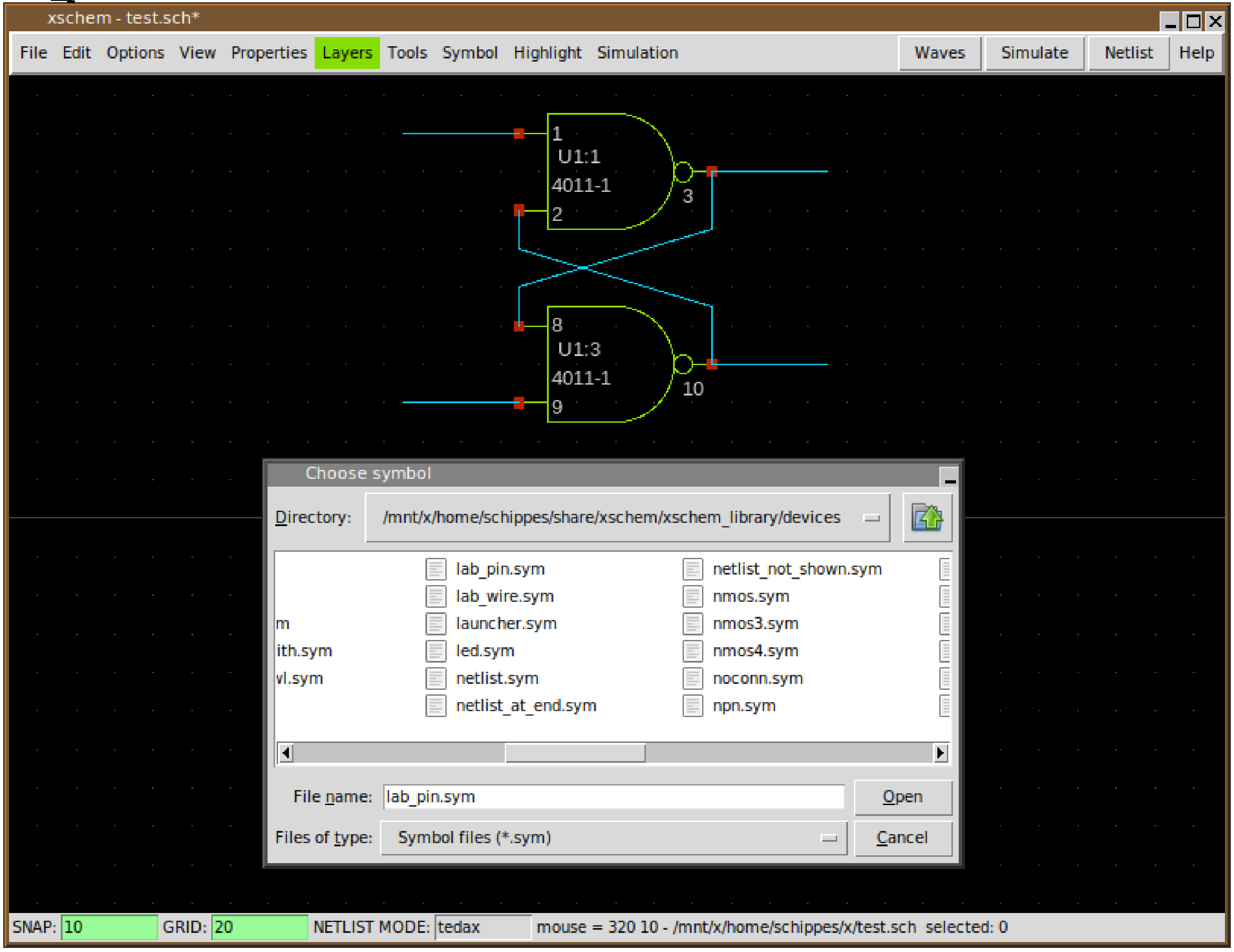
We see that all pin numbers are shown for each pin; this reminds us that this is a slotted device! slotted devices should specify the slot number in the instance **`name`** so, select the component, press **`q`** and change the **`U1`** name attribute to **`U1:1`**. You can also remove the **`.sym`** extension in the 'Symbol' entry of the dialog box, for more compactness:



As you can see now the slot is resolved and the right pin numbers are displayed. Now select and copy the component (use the **`c`** key), and change the **`name`** attribute of the new copy to **`U1:3`**:



10. Now draw some wires, for example to create an SR latch as shown, use the **`w`** key to draw wires; when done with the wiring insert a net label by pressing the **`Insert`** key and navigating to **`.../share/xschem/xschem_library/devices`** (the XSCHEM system symbol library) and selecting **`lab_pin`**:



Directory: /mnt/x/home/schippes/share/xschem/xschem_library/devices

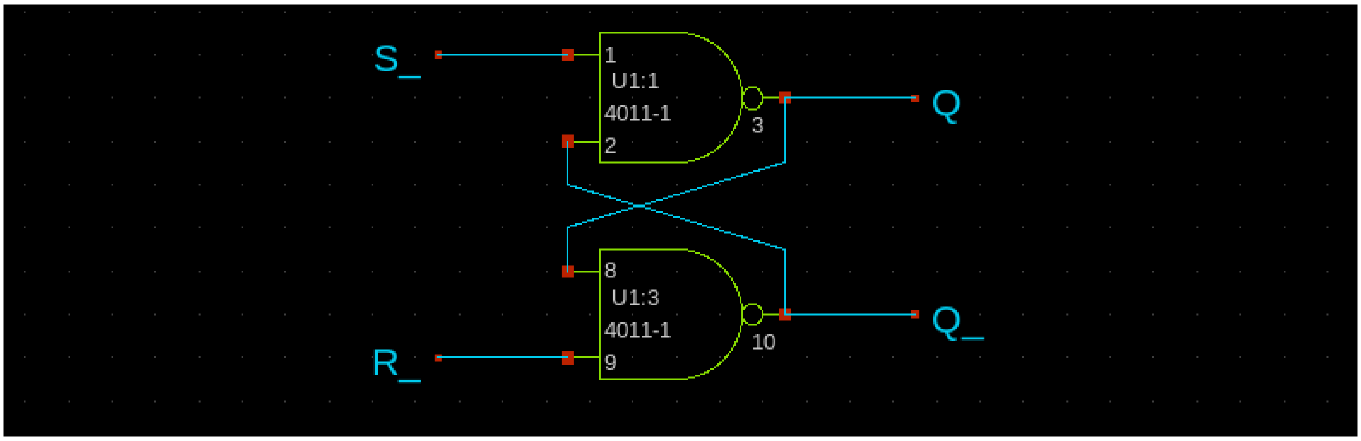
```
m          lab_pin.sym          netlist_not_shown.sym
ith.sym    lab_wire.sym         nmos.sym
vl.sym     launcher.sym         nmos3.sym
           led.sym              nmos4.sym
           netlist.sym          noconn.sym
           netlist_at_end.sym   npn.sym
```

File name: lab_pin.sym

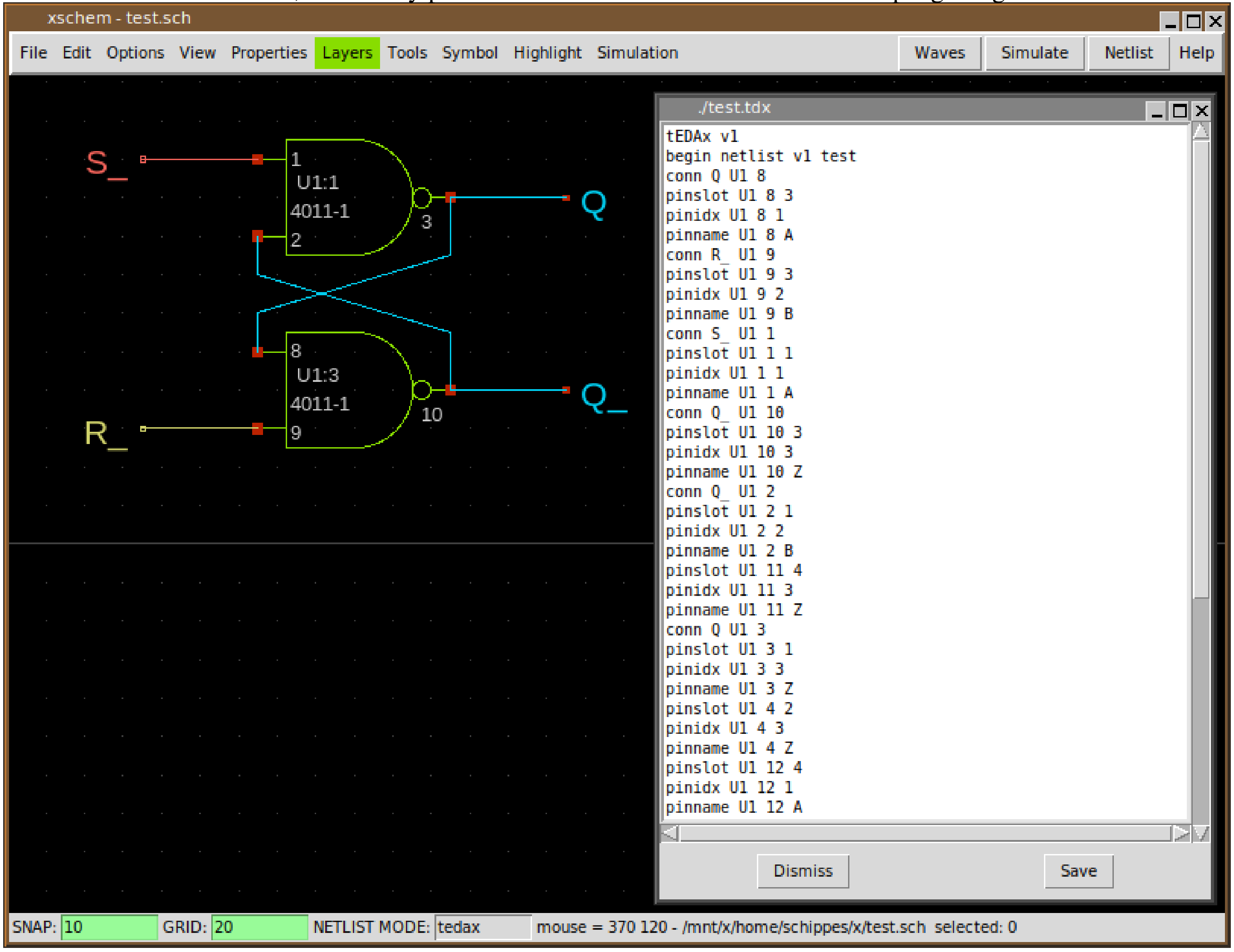
Files of type: Symbol files (*.sym)

SNAP: 10 GRID: 20 NETLIST MODE: tedax mouse = 320 10 - /mnt/x/home/schippes/x/test.sch selected: 0

Place 4 of these **`lab_pin`** symbols and set their **`lab`** attribute to **`S_, R_, Q, Q_`** respectively; place the 4 labels as shown (use the **`Shift-f`** key to flip the **`Q, Q_`** labels):



11. The test circuit for this tutorial is now complete: its time to extract the tEDAx netlist; press the **`Shift-A`** key to enable showing the netlist window, press **`Shift-v`** multiple times to set the netlisting mode as shown in the bottom status bar to **`tedax`**, and finally press the **`Netlist`** button located in the top-right region of the window:


xschem - test.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Waves Simulate Netlist Help
./test.tdx
Dismiss
Save

```
tEDAx v1
begin netlist v1 test
conn Q U1 8
pinslot U1 8 3
pinidx U1 8 1
pinname U1 8 A
conn R_ U1 9
pinslot U1 9 3
pinidx U1 9 2
pinname U1 9 B
conn S_ U1 1
pinslot U1 1 1
pinidx U1 1 1
pinname U1 1 A
conn Q_ U1 10
pinslot U1 10 3
pinidx U1 10 3
pinname U1 10 Z
conn Q_ U1 2
pinslot U1 2 1
pinidx U1 2 2
pinname U1 2 B
pinslot U1 11 4
pinidx U1 11 3
pinname U1 11 Z
conn Q U1 3
pinslot U1 3 1
pinidx U1 3 3
pinname U1 3 Z
pinslot U1 4 2
pinidx U1 4 3
pinname U1 4 Z
pinslot U1 12 4
pinidx U1 12 1
pinname U1 12 A
```

SNAP: 10 GRID: 20 NETLIST MODE: tedax mouse = 370 120 - /mnt/x/home/schippes/x/test.sch selected: 0

This is the resulting netlist you should get:

```
tEDAx v1
begin netlist v1 test
conn Q U1 8
```

```
pinslot U1 8 3
pinidx U1 8 1
pinname U1 8 A
conn R_ U1 9
pinslot U1 9 3
pinidx U1 9 2
pinname U1 9 B
conn S_ U1 1
pinslot U1 1 1
pinidx U1 1 1
pinname U1 1 A
conn Q_ U1 10
pinslot U1 10 3
pinidx U1 10 3
pinname U1 10 Z
conn Q_ U1 2
pinslot U1 2 1
pinidx U1 2 2
pinname U1 2 B
pinslot U1 11 4
pinidx U1 11 3
pinname U1 11 Z
conn Q U1 3
pinslot U1 3 1
pinidx U1 3 3
pinname U1 3 Z
pinslot U1 4 2
pinidx U1 4 3
pinname U1 4 Z
pinslot U1 12 4
pinidx U1 12 1
pinname U1 12 A
pinslot U1 13 4
pinidx U1 13 2
pinname U1 13 B
pinslot U1 5 2
pinidx U1 5 1
pinname U1 5 A
conn VCC U1 14
pinname U1 14 power
pinslot U1 6 2
pinidx U1 6 2
pinname U1 6 B
conn GND U1 7
pinname U1 7 ground
footprint U1 dip(14)
device U1 CD4011B
end netlist
```

This concludes the tutorial; of course this is not a complete circuit, connectors are missing among other things, but the basics of creating a new component should now be less obscure.

TUTORIAL: Manage XSCHEM design / symbol libraries

There are various ways to describe symbol locations in xschem,

- first approach: define a **`XSCHEM_LIBRARY_PATH`** that is a list of paths to last level directories containing .sym /.sch files
- second approach: define a **`XSCHEM_LIBRARY_PATH`** that is a list of paths one or more levels above the directories containing .sym/.sch files
- Third approach: define a **`XSCHEM_LIBRARY_PATH`** that is a hierarchy of paths, zero, one or more levels above the directories containing .sym/.sch files. If you have a directory tree where each directory level may contain .sch and .sym files you should list the deepest directories first so xschem will start searching for a symbol reference in the deepest levels first.

In the first approach a '**`npn.sym`**' symbol placed in a schematic will be saved as '**`npn.sym`**' in the .sch file, when loading back the parent schematic xschem will go through the elements of **`XSCHEM_LIBRARY_PATH`** and look for a directory containing **`npn.sym`**.

In the second approach the '**`npn.sym`**' will be saved as '**`devices/npn.sym`**' (assuming **`devices/`** is the directory containing this symbol) . This is because the **`XSCHEM_LIBRARY_PATH`** is pointing to something like **`/some/path/xschem_library/`** and **`xschem_library/`** contains **`devices/`** (names are just given as examples, any dir name is allowed for **`xschem_library/`** and **`devices/`**)

In the third approach '**`npn.sym`**' or some other **`dir/symbol.sym`** will be searched in all path elements listed in **`XSCHEM_LIBRARY_PATH`**, by appending the symbol reference to each path element until a file is found. the first match is used. This is the reason you should put the deepest directories first in **`XSCHEM_LIBRARY_PATH`**. If **`/a/b/c/dir/symbol.sym`** is inserted in the design and XSCHEM_LIBRARY_PATH contains the following definitions:

```
set XSCHEM_LIBRARY_PATH /a/b/c /a/b /a
```

the symbol reference will be just **`dir/symbol.sym`**, since appending the symbol reference to the first path an existing file is found. If the following definition for XSCHEM_LIBRARY_PATH is given instead:

```
set XSCHEM_LIBRARY_PATH /a /a/b/ /a/b/c
```

then the symbol reference will be **`/b/c/dir/symbol.sym`** since the first path component was found in the absolute path of the inserted symbol and the only matching prefix is removed from the relative symbol reference that will be saved in the schematic.

The first approach is preferred by pcb hobbysts, people working on small designs. the second approach is preferred for big designs where a one or more directory level indirection is desired for symbols, so any symbol in xschem is given as '**`libname/symname.sym`**' (one level directory specification in symbol references) or '**`libgroup/libname/symname.sym`**' (2 level directory specification in symbol references) instead of just '**`symname.sym`**'

SYMBOL LOOKUP (ie when loading a schematic):
The absolute path of the symbol reference is obtained by appending the symbol reference to the XSCHEM_LIBRARY_PATH paths in the order they are listed until the resulting file is found in the machine filesystem. The first match is used.
SYMBOL INSERTION (ie when drawing a schematic and inserting a component):
The relative symbol reference that is saved in the schematic file is obtained by removing the first occurrence of a matching path prefix from the ones listed in `XSCHEM_LIBRARY_PATH` in the order they are listed. The first matching prefix is used to determine the relative symbol reference. This is the reason deepest path elements must be

listed first in XSCHEM_LIBRARY_PATH if you want the shortest possible symbol relative reference to be saved in the schematic file.

For VLSI / big designs I **strongly** suggest using the second approach, just as an example i have the following dirs:

```
~/share/xschem/xschem_library/
  containing:
  devices/
  TECHLIB/

~/xschem_library/
  containing:
  stdcell_stef/

~/share/doc/xschem/
  containing:
  library_t9/
  dram/
```

then in my xschemrc i have the following:

```
set XSCHEM_LIBRARY_PATH \
$env(HOME)/share/xschem/xschem_library:$env(HOME)/share/doc/xschem/:$env(HOME)/xschem_
```

You may choose either method, but please be consistent throughout your design.

Change project setup runtime

Since Xschem now handles multiple windows or tabs, it is desirasble to load schematics from different projects into a single running instance of xschem. This is not difficult to do and you might want to write your own procedure into your xschemrc to automate this. Lets suppose you open a new schematic tab. After opening the new tab go to the xschem prompt in the terminal you launched Xschem from, and redefine your XSCHEM_LIBRARY_PATH:

```
set XSCHEM_LIBRARY_PATH {} ;# clear previous definitions
append XSCHEM_LIBRARY_PATH :${XSCHEM_SHAREDIR}/xschem_library  ;# for devices/
append XSCHEM_LIBRARY_PATH :/home/schippes/share/pdk/sky130A/libs.tech/xschem ;# for sky130 lil
# project specific variables (either tcl variables or shell variables via the TCL env() array)
set PDK_ROOT /home/schippes/share/pdk
set PDK sky130A
set SKYWATER_MODELS ${PDK_ROOT}/${PDK}/libs.tech/ngspice
set SKYWATER_STDCELLS ${PDK_ROOT}/${PDK}/libs.ref/sky130_fd_sc_hd/spice
```

At this point your new tab will work with the new defnitions while the previous tab will continue with its previous settings.

you should create a small procedure and put int into your xschemrc so you will just need to type the procedure name:

```
proc set_sky130 {} {
    ## XSCHEM_SHAREDIR points to XSCHEM install path, example: /usr/local/share/xschem
    ## USER_CONF_DIR is usually ~/.xschem
    ## env may be used to set environment variables, like:
```

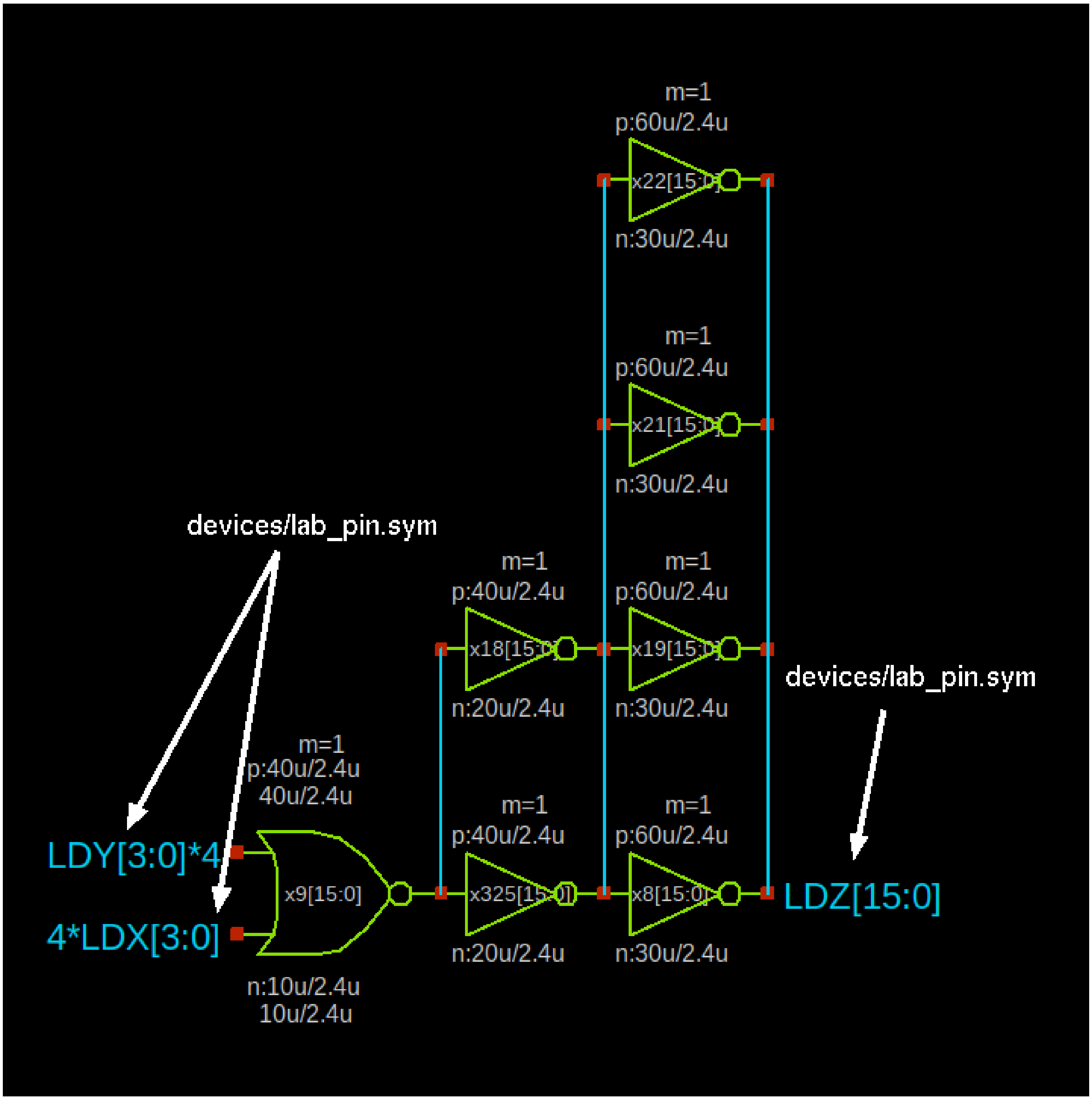
```
    ## set env(PDK_ROOT) .....
    global XSCHEM_LIBRARY_PATH XSCHEM_SHAREDIR USER_CONF_DIR env
    ## Other global TCL variables listed here depend on the project setup.
    global PDK_ROOT PDK SKYWATER_MODELS SKYWATER_STDCELLS

    # project specific variables (either tcl variables or shell variables via the TCL env() arra
    set PDK_ROOT /home/schippes/share/pdk
    set PDK sky130A
    set SKYWATER_MODELS ${PDK_ROOT}/${PDK}/libs.tech/ngspice
    set SKYWATER_STDCELLS ${PDK_ROOT}/${PDK}/libs.ref/sky130_fd_sc_hd/spice

    set XSCHEM_LIBRARY_PATH {} ;# clear previous definitions
    append XSCHEM_LIBRARY_PATH :${XSCHEM_SHAREDIR}/xschem_library  ;# for devices/
    append XSCHEM_LIBRARY_PATH :${PDK_ROOT}/${PDK}/libs.tech/xschem
}
```

TUTORIAL: Use Bus/Vector notation for signal bundles / arrays of instances

XSCHEM has the ability to use a compact notation to represent signal bundles. There is no specific 'bus' entity, in XSCHEM a bus is simply a wire with a label representing a bundle of bits, the syntax is explained below. Normally a net label assigns a name to a wire, for example 'ENABLE', 'RESET', 'CLK' and so on, however more complex formats are available to describe multiple bits.

- **`AAA,BBB,CCC`**: described a bundle of 3 signals, AAA, BBB, CCC.
- **`AAA[3:0]`**: describes the set **`AAA[3],AAA[2],AAA[1],AAA[0]`**. The form **`AAA[3:0]`** and **`AAA[3],AAA[2],AAA[1],AAA[0]`** are exactly equivalent.
- **`AAA[1:0],BBB[5:4]`**: describes the bundle: **`AAA[1],AAA[0],BBB[5],BBB[4]`**.
- **`AAA[6:0:2]`**: describes the bundle **`AAA[6],AAA[4],AAA[2],AAA[0]`**.
- **`AAA[0:1:4:3]`**: describes the bundle **`AAA[0],AAA[1],AAA[4],AAA[5],AAA[8],AAA[9]`**. The meaning of the 4 parameters are: start:end:offset:repetitions.
- **`2*AAA[1:0]`**: describes the bundle **`AAA[1],AAA[0],AAA[1],AAA[0]`**.
- **`AAA[1:0]*2`**: describes the bundle **`AAA[1],AAA[1],AAA[0],AAA[0]`**.
- **`2*(AAA[1:0],BBB)`**: describes the bundle **`AAA[1],AAA[0],BBB,AAA[1],AAA[0],BBB`**.
- **`(AAA[1:0],BBB)*2`**: describes the bundle **`AAA[1],AAA[1],AAA[0],AAA[0],BBB,BBB`**.

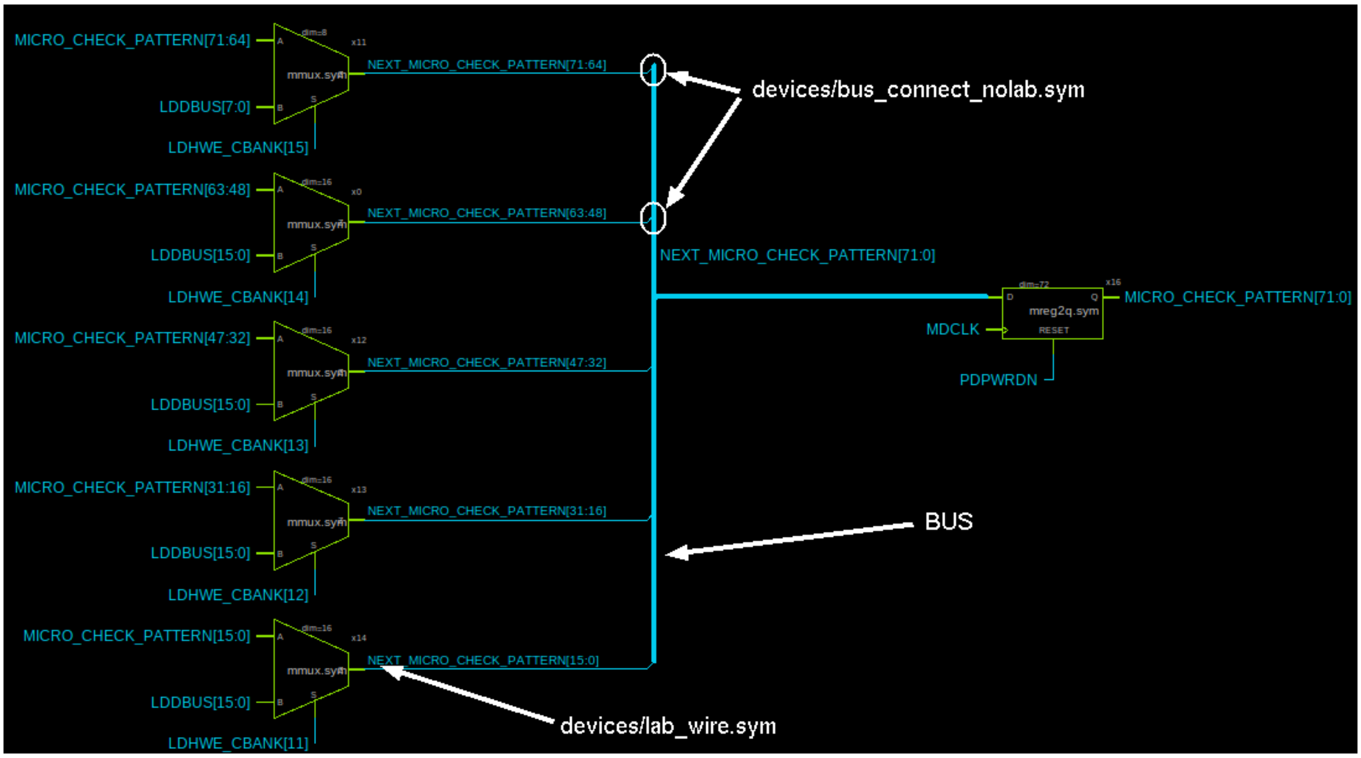
All the above notations are perfectly valid label net name attributes.
In a very similar way multiple instances can be placed in a schematic setting the 'name' attribute to a vector notation.
For example in picture below **`x22[15:0]`** represents 16 inverters with names **`x22[15],x22[14],...,x22[0]`**.



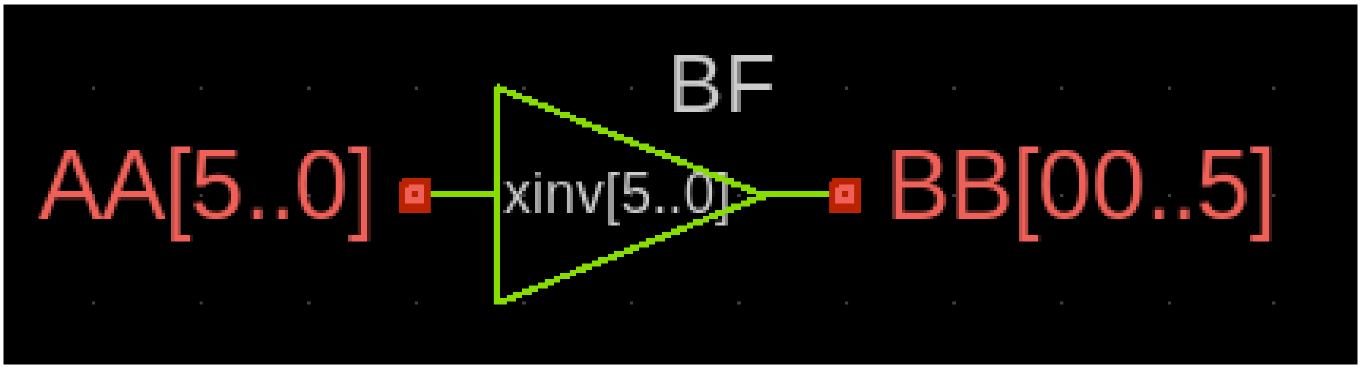
Recently a new notation has been added for buses that expands without putting brackets:

- **`AAA[3..0]`**: describes the set **`AAA3,AAA2,AAA1,AAA0`**. The form **`AAA[3..0]`** and **`AAA3,AAA2,AAA1,AAA0`** are exactly equivalent.
- **`AAA[1..0],BBB[5..4]`**: describes the bundle: **`AAA1,AAA0,BBB5,BBB4`**.
- **`AAA[6..0..2]`**: describes the bundle **`AAA6,AAA4,AAA2,AAA0`**.
- **`2*AAA[1..0]`**: describes the bundle **`AAA1,AAA0,AAA1,AAA0`**.
- **`AAA[1..0]*2`**: describes the bundle **`AAA1,AAA1,AAA0,AAA0`**.
- **`2*(AAA[1..0],BBB)`**: describes the bundle **`AAA1,AAA0,BBB,AAA1,AAA0,BBB`**.
- **`(AAA[1..0],BBB)*2`**: describes the bundle **`AAA1,AAA1,AAA0,AAA0,BBB,BBB`**.

In following picture there is a main 72 bit bus (the vertical thick wire) and bus ripper symbols (**`devices/bus_connect_nolab.sym`**) are used to take slices of bits from the main bus. Wire labels are used to define bus slices. To display thick wires for busses, select all wire segments, then press 'q' and add attribute **`bus=true`**.


MICRO_CHECK_PATTERN[71:64]
LDDBUS[7:0]
LDHWE_CBANK[15]
mmux.sym
NEXT_MICRO_CHECK_PATTERN[71:64]
devices/bus_connect_nolab.sym
MICRO_CHECK_PATTERN[63:48]
LDDBUS[15:0]
LDHWE_CBANK[14]
NEXT_MICRO_CHECK_PATTERN[63:48]
NEXT_MICRO_CHECK_PATTERN[71:0]
mreg2q.sym
MICRO_CHECK_PATTERN[71:0]
MDCLK
PDPWRDN
MICRO_CHECK_PATTERN[47:32]
LDHWE_CBANK[13]
NEXT_MICRO_CHECK_PATTERN[47:32]
MICRO_CHECK_PATTERN[31:16]
LDHWE_CBANK[12]
NEXT_MICRO_CHECK_PATTERN[31:16]
BUS
MICRO_CHECK_PATTERN[15:0]
LDHWE_CBANK[11]
NEXT_MICRO_CHECK_PATTERN[15:0]
devices/lab_wire.sym

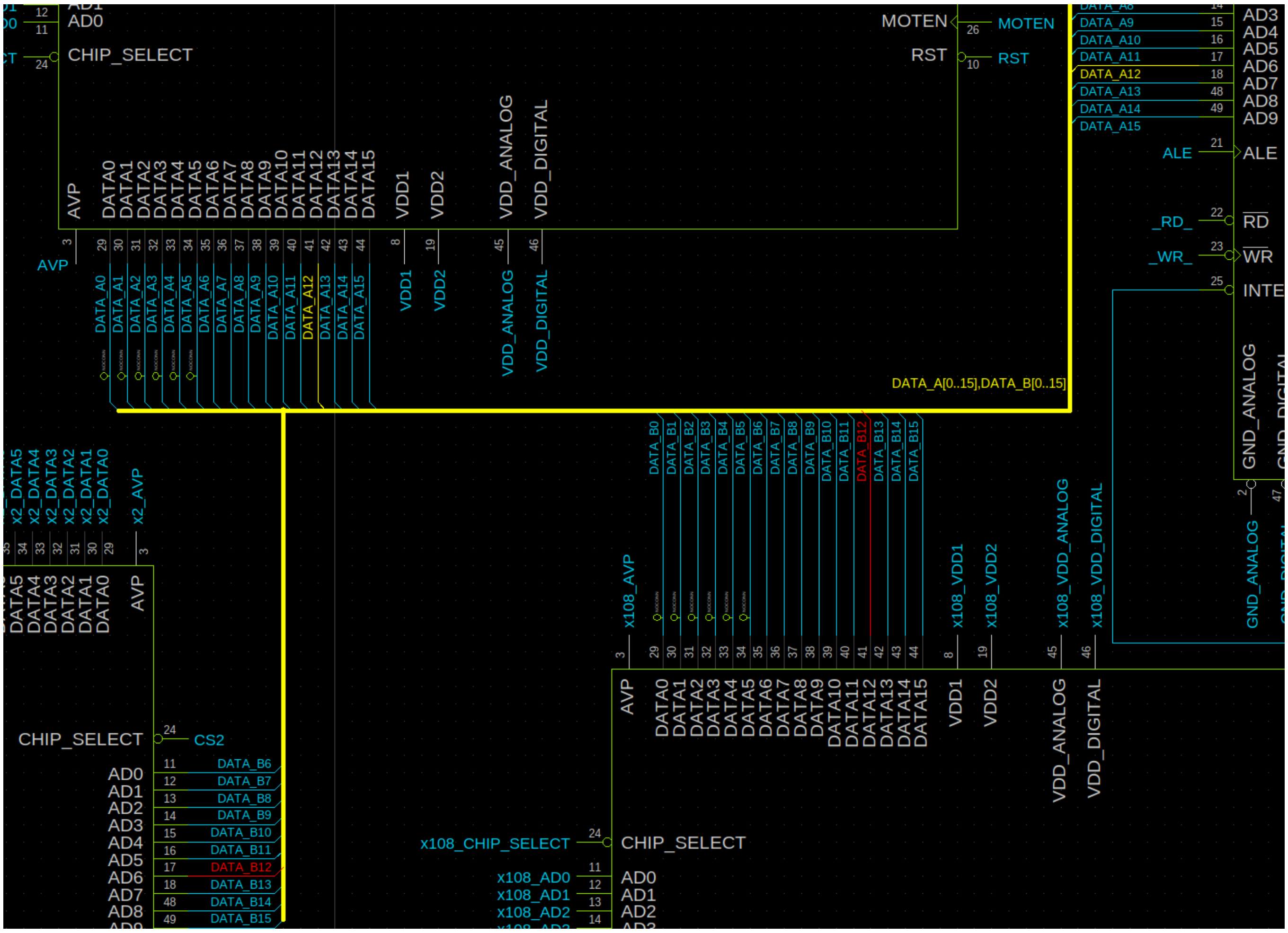
following picture shows an istantiation of 6 inverters:



The generated spice netlist is the following:

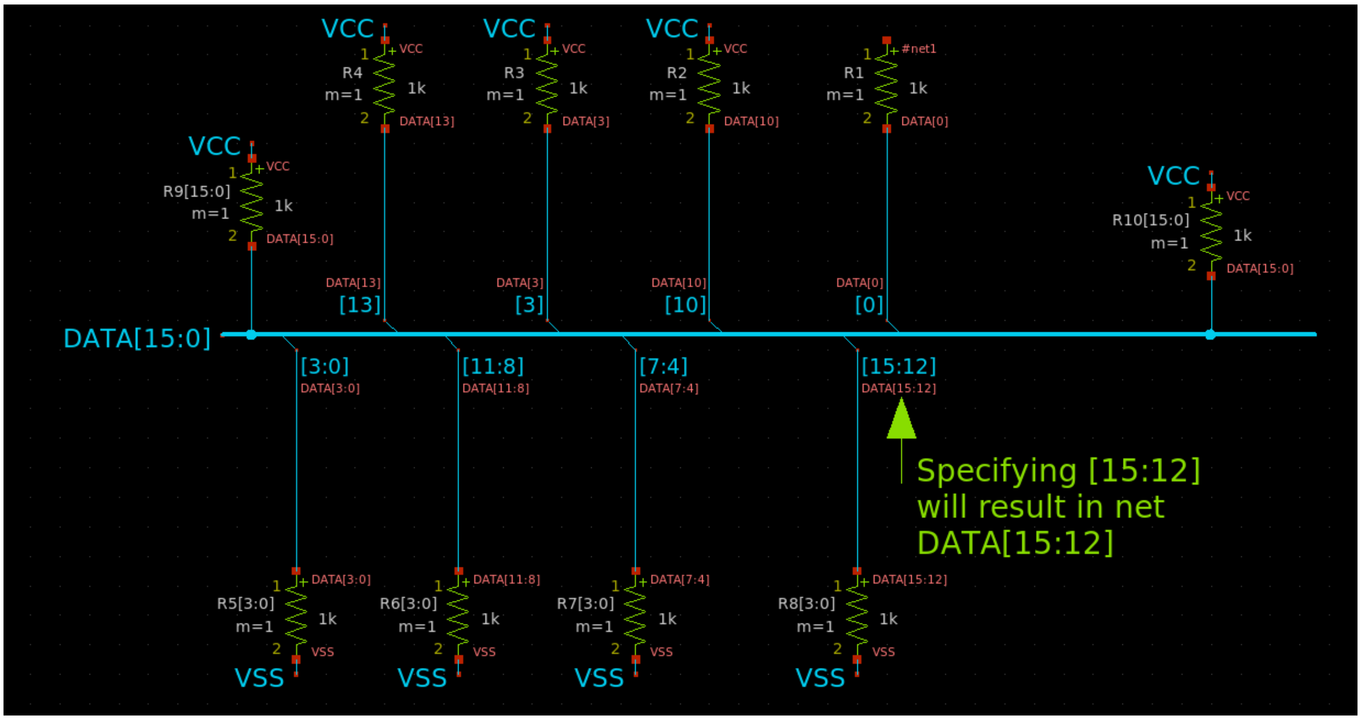
```
...
xinv5 BB0 AA5 bf
xinv4 BB1 AA4 bf
xinv3 BB2 AA3 bf
xinv2 BB3 AA2 bf
xinv1 BB4 AA1 bf
xinv0 BB5 AA0 bf
...
```

Example of a more complex bus routing. main bus is a bundle of 2 buses: DATA_A[0..15] and DATA_B[0..15]

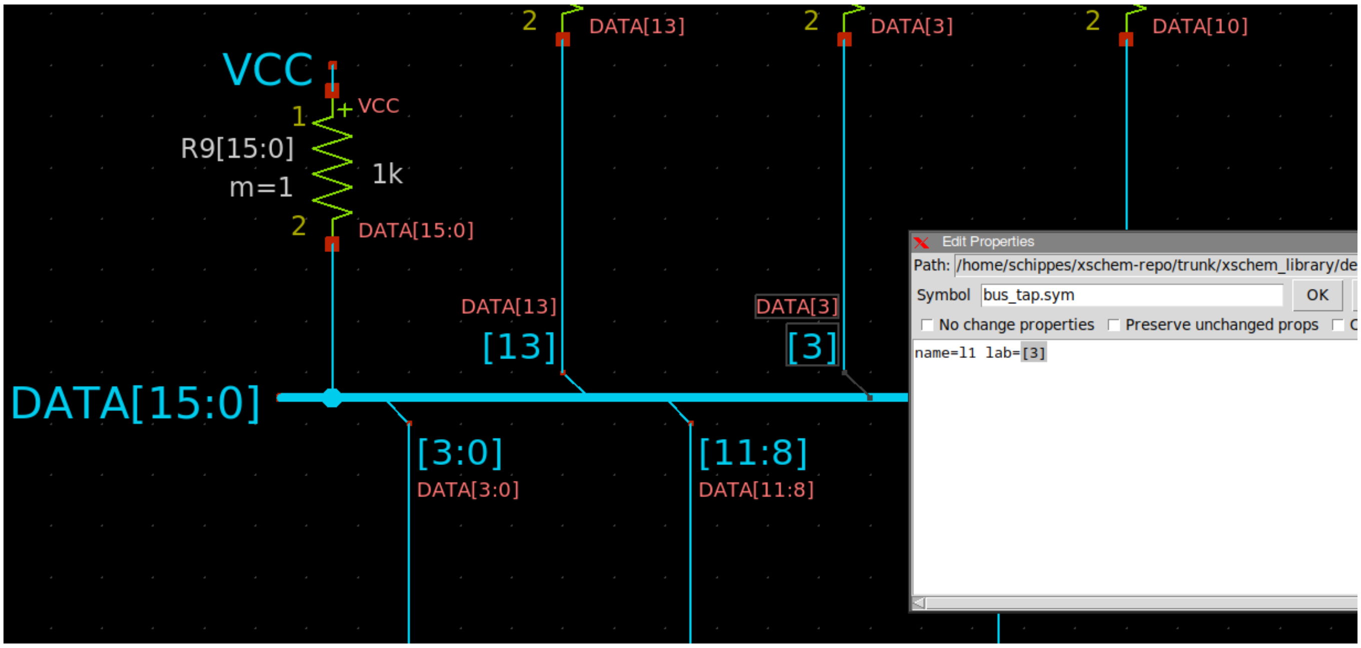


BUS TAPS

A new symbol, **`devices/bus_tap.sym`** has been creted to make bus connections more flexible. This is a 2 pin symbol, one pin must be connected to the bus wire, the other pin only defines the bus slice, indicating only the range of bits and not the complete bus name:



As you see in the picture a **lab** attribute is given that specifies only a bit range, like **[13]** or **[7:0]**. The net attached to the 'bus slice' end of the **bus_tap.sym** will get the base name of the bus (**DATA** in the example) and the index, that is **DATA[13]** In the example below the menu **Options->Show net names on symbol pins / floaters** has been enbled to see (the pink texts) the resulting net names.



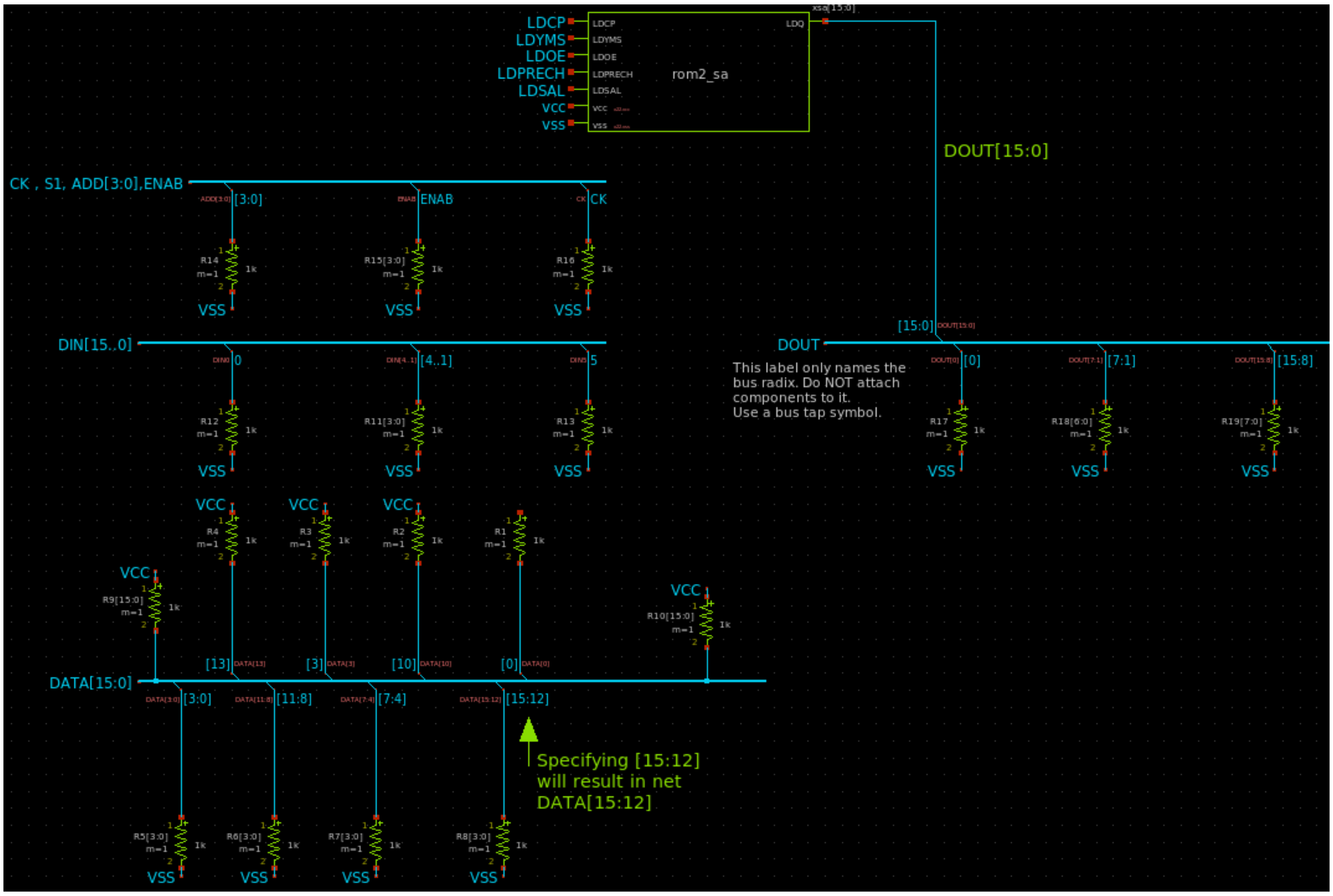
Path: /home/schippes/xschem-repo/trunk/xschem_library/de

Symbol bus_tap.sym

No change properties Preserve unchanged props

```
name=l1 lab=[3]
```

A complete example **examples/test_bus_tap.sch** shows various possible **bus_tap.sym** use cases.



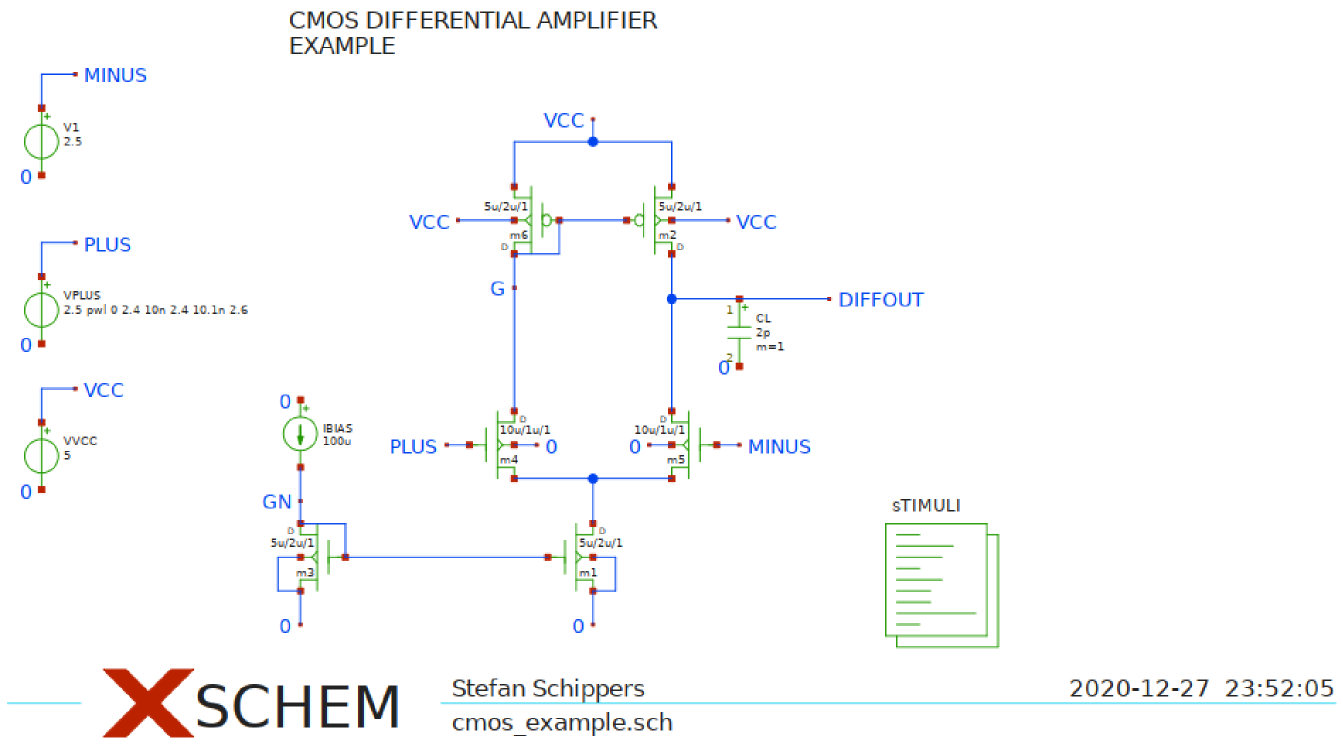
This label only names the bus radix. Do NOT attach components to it.
Use a bus tap symbol.

Specifying [15:12] will result in net DATA[15:12]

UP

TUTORIAL: Backannotation of NGSPICE simulation operating point data into an XSCHEM schematic

The objective of this tutorial is to show into the schematic the operating point data (voltages currents, other electrical parameters) of a SPICE simulation done with the Ngspice simulator. This tutorial is based on the **`cmos_example.sch`** example schematic located in the **`examples/`** directory. Start Xschem from a terminal since we need to give some commands in this tutorial.



SETUP

Select the 'STIMULI' code block (click on it) and edit its attributes (press **`q`** or **`Shift-q`**):

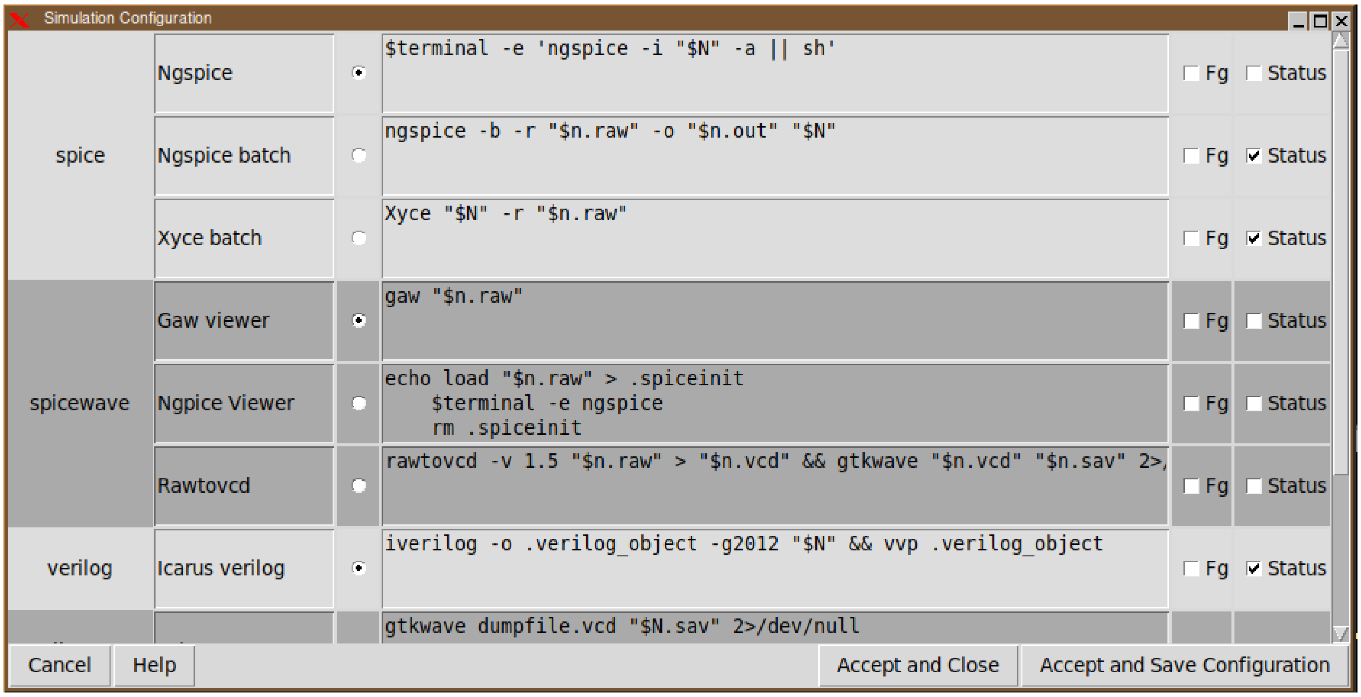
```
.temp 30
** models are generally not free: you must download
** SPICE models for active devices and put them into the below
** referenced file in netlist/simulation directory.
.include "models_cmos_example.txt"
.control
op
save all
write cmos_example.raw
.endc
```

The important parts are in red in above text. This ensures all variables are saved into the raw file. These instructions are for an interactive ngspice run.

You may have other simulations saved in the raw file (dc, tran, ac) however one operating point must also be present:

```
.temp 30
** models are generally not free: you must download
** SPICE models for active devices and put them into the below
** referenced file in netlist/simulation directory.
.include "models_cmos_example.txt"
.control
save all
dc vplus 2.3 2.7 0.001
write cmos_example.raw
set appendwrite
op
save all
write cmos_example.raw
.endc
```

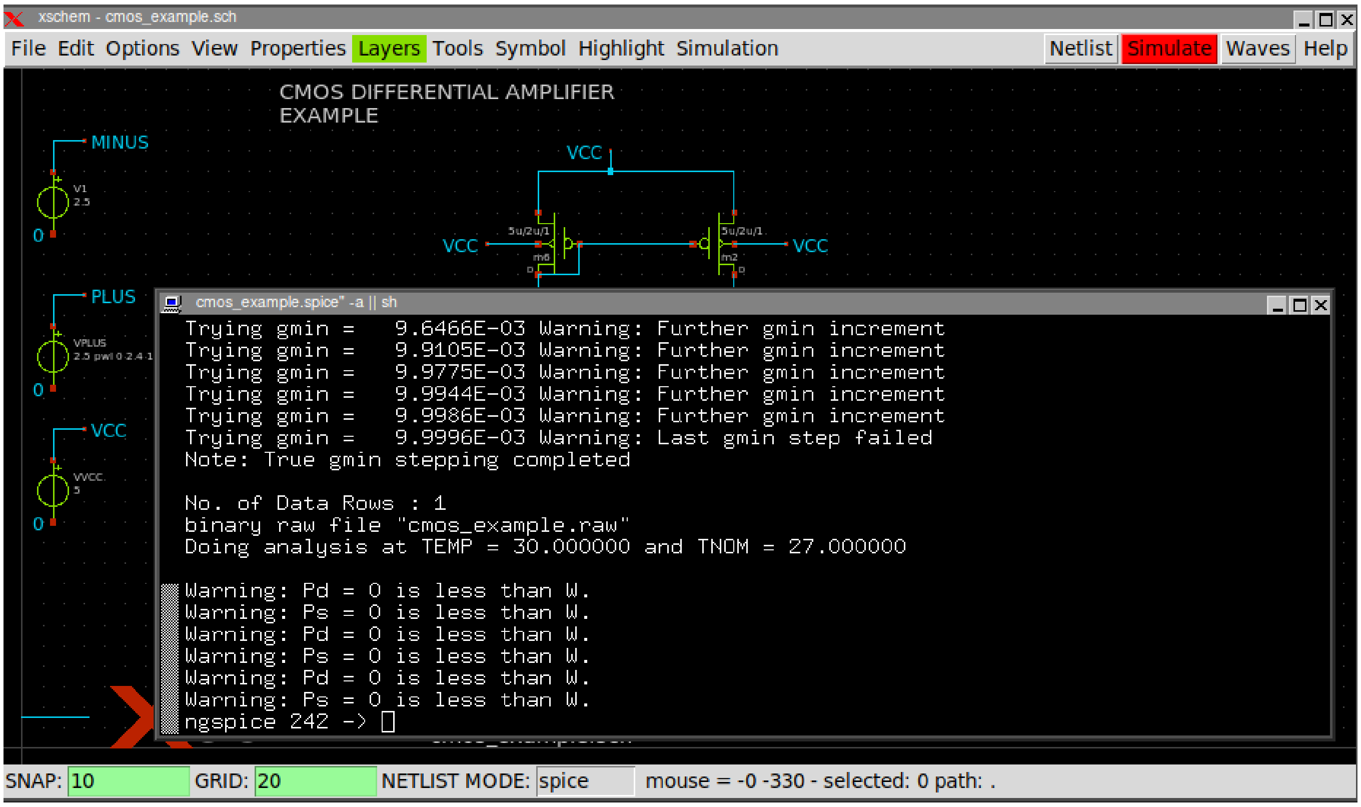
When done open the **`Simulation-> Configure simulators and tools`** dialog box and ensure the **`Ngspice`** simulator is selected (not Ngspice batch). Also ensure the spice netlist mode is selected (**`Options -> Spice netlist`**).


Simulation Configuration

spice	Ngspice	●	`$terminal -e 'ngspice -i "$N" -a || sh'`	☐ Fg	☐ Status
	Ngspice batch	○	`ngspice -b -r "$n.raw" -o "$n.out" "$N"`	☐ Fg	☑ Status
	Xyce batch	○	`Xyce "$N" -r "$n.raw"`	☐ Fg	☑ Status
spicewave	Gaw viewer	●	`gaw "$n.raw"`	☐ Fg	☐ Status
	Ngpice Viewer	○	`echo load "$n.raw" > .spiceinit` `$terminal -e ngspice` `rm .spiceinit`	☐ Fg	☐ Status
	Rawtovcd	○	`rawtovcd -v 1.5 "$n.raw" > "$n.vcd" && gtkwave "$n.vcd" "$n.sav" 2>/`	☐ Fg	☐ Status
verilog	Icarus verilog	●	`iverilog -o .verilog_object -g2012 "$N" && vvp .verilog_object`	☐ Fg	☑ Status
			`gtkwave dumpfile.vcd "$N.sav" 2>/dev/null`		

SIMULATION

If you now press the **`Netlist`** followed by the **`Simulate`** button simulation should complete with no errors.


xschem - cmos_example.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
CMOS DIFFERENTIAL AMPLIFIER
EXAMPLE
MINUS
PLUS
VCC
cmos_example.spice" -a || sh
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = -0 -330 - selected: 0 path: .

```
Trying gmin =   9.6466E-03 Warning: Further gmin increment
Trying gmin =   9.9105E-03 Warning: Further gmin increment
Trying gmin =   9.9775E-03 Warning: Further gmin increment
Trying gmin =   9.9944E-03 Warning: Further gmin increment
Trying gmin =   9.9986E-03 Warning: Further gmin increment
Trying gmin =   9.9996E-03 Warning: Last gmin step failed
Note: True gmin stepping completed

No. of Data Rows : 1
binary raw file "cmos_example.raw"
Doing analysis at TEMP = 30.000000 and TNOM = 27.000000

Warning: Pd = 0 is less than W.
Warning: Ps = 0 is less than W.
Warning: Pd = 0 is less than W.
Warning: Ps = 0 is less than W.
Warning: Pd = 0 is less than W.
Warning: Ps = 0 is less than W.
ngspice 242 ->
```

You can close the simulator since we need only the **`cmos_example.raw`** file that is now saved in the simulation directory (usually **`~/.xschem/simulations/cmos_example.raw`**).
Now verify that xschem is able to read the raw file: issue this command in the xschem console:
`xschem annotate_op`

```
xschem [~] xschem annotate_op
Raw file data read: /home/schippes/.xschem/simulations/cmos_example.raw
points=1, vars=38, datasets=1
0
xschem [~]
```

If there are no errors we are ready and set.
you can load also a specific file:

```
xschem [~] xschem annotate_op $netlist_dir/cmos_example_ngspice.raw
```

ANNOTATION

The annotation procedure is based on a **pull** method: the probe objects have atributes or tcl commands embedded that fetch simulation data from a table that has been read by Xschem. In addition to specific probe elements also net labels will show voltage values and ammeters / voltage sources will show currents.

To ensure all currents are saved modify the **`STIMULI`** attributes as follows:

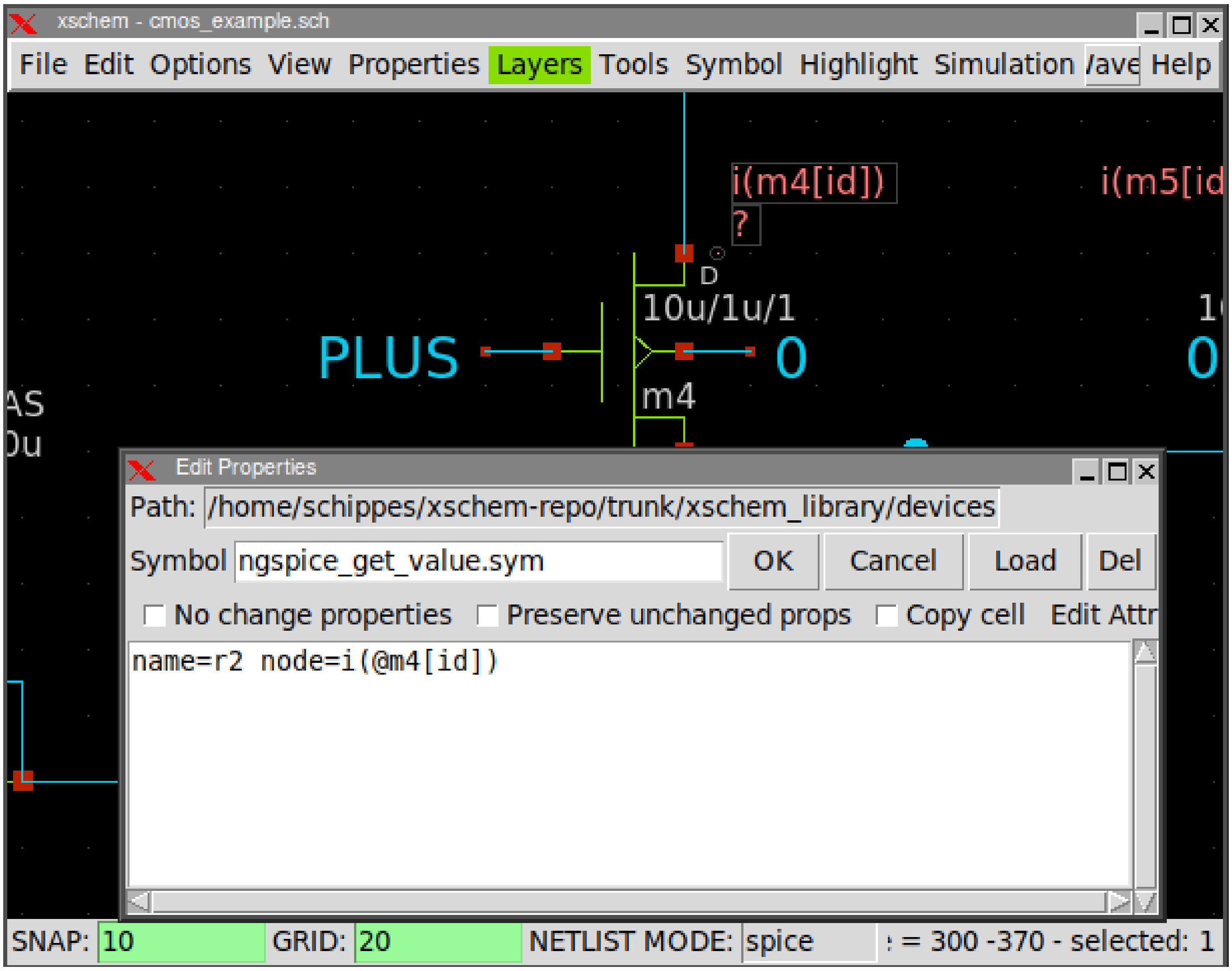
```
.temp 30
** models are generally not free: you must download
** SPICE models for active devices and put them into the below
** referenced file in netlist/simulation directory.
.include "models_cmos_example.txt"
```

```
.option savecurrents
.save all
.control
op
write cmos_example.raw
.endc
```

Remove all previous probe elements and place some **`devices/ngspice_probe.sym`** components and some **`devices/ngspice_get_value.sym`** components. the ngspice_probe.sym is a simple voltage viewer and must be attached to a net. The ngspice_get_value.sym displays a generic variable stored in the raw file. This symbol is usually placed next to the referenced component, but does not need to be attached to any specific point or wire. Edit its attributes and set its **`node`** attribute to an existing saved variable in the raw file.

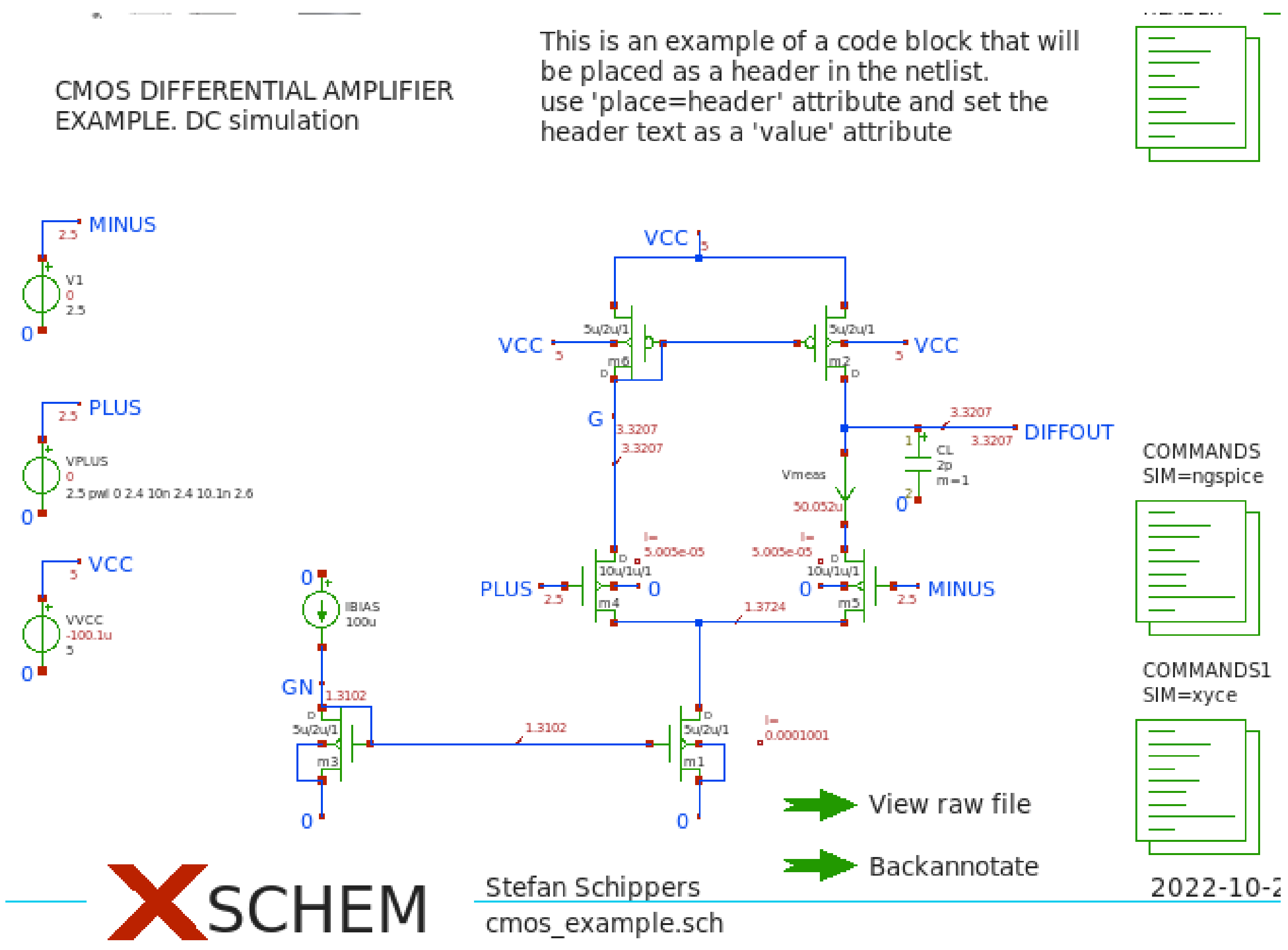
`ngspice_get_value.sym` debugging tips:

- See all available devices in ngspice with **`display`**
- See all values for a device with **`print @somedevice`**
    - eg **`print @m.xm6.xmain1.msky130_fd_pr__nfet_g5v0d16v0__base[vth]`**
- **Some values must explicitly be saved** before the analysis to be available for annotation.
    - eg **`save @m.xm6.xmain1.msky130_fd_pr__nfet_g5v0d16v0__base[vth]`**
- Usually you'll need to wrap your value with **`v()`** in the symbol properties in xschem.
    - eg **`node=v(@m.xm6.xmain1.msky130_fd_pr__nfet_g5v0d16v0__base[vth])`**


xschem - cmos_example.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation /ave Help
i(m4[id])
?
i(m5[id
D
10u/1u/1
PLUS
0
m4
Edit Properties
Path: /home/schippes/xschem-repo/trunk/xschem_library/devices
Symbol ngspice_get_value.sym
OK
Cancel
Load
Del
No change properties
Preserve unchanged props
Copy cell
Edit Attr
SNAP: 10
GRID: 20
NETLIST MODE: spice
= 300 -370 - selected: 1

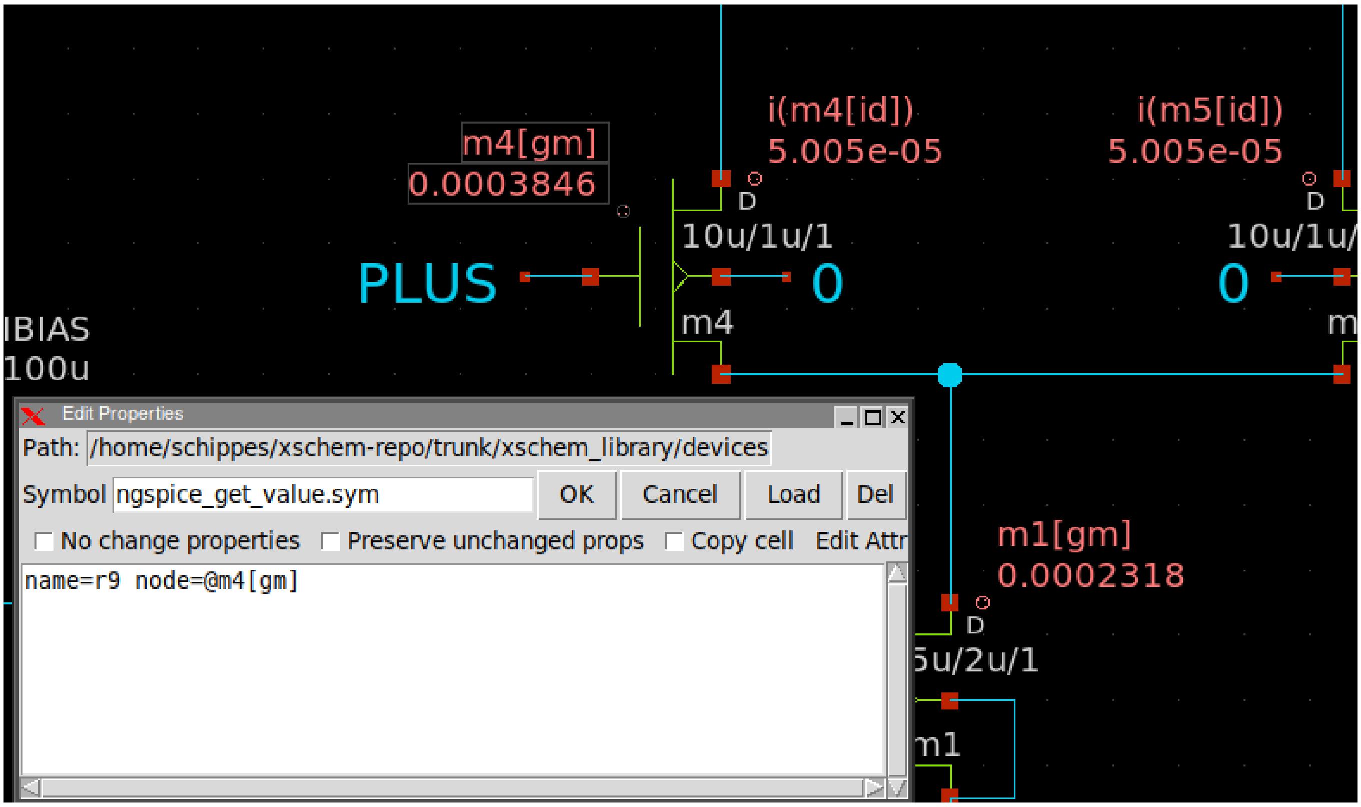
```
name=r2 node=i(@m4[id])
```

Run again the simulation and the **`xschem annotate_op`** command and values will be updated.



You can add additional variables in the raw file , for example modifying the .save instruction:

```
.save all @m4[gm] @m5[gm] @m1[gm]
```



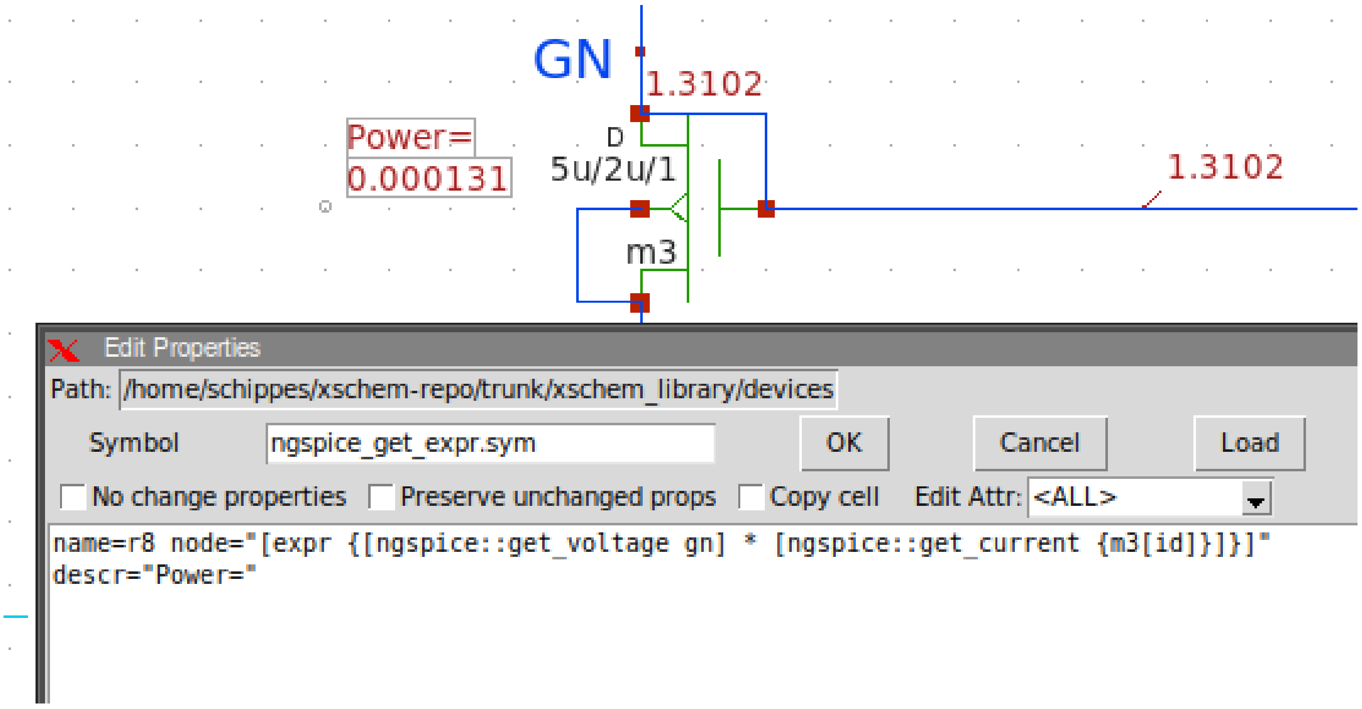
Path: /home/schippes/xschem-repo/trunk/xschem_library/devices

Symbol ngspice_get_value.sym

```
name=r9 node=@m4[gm]
```

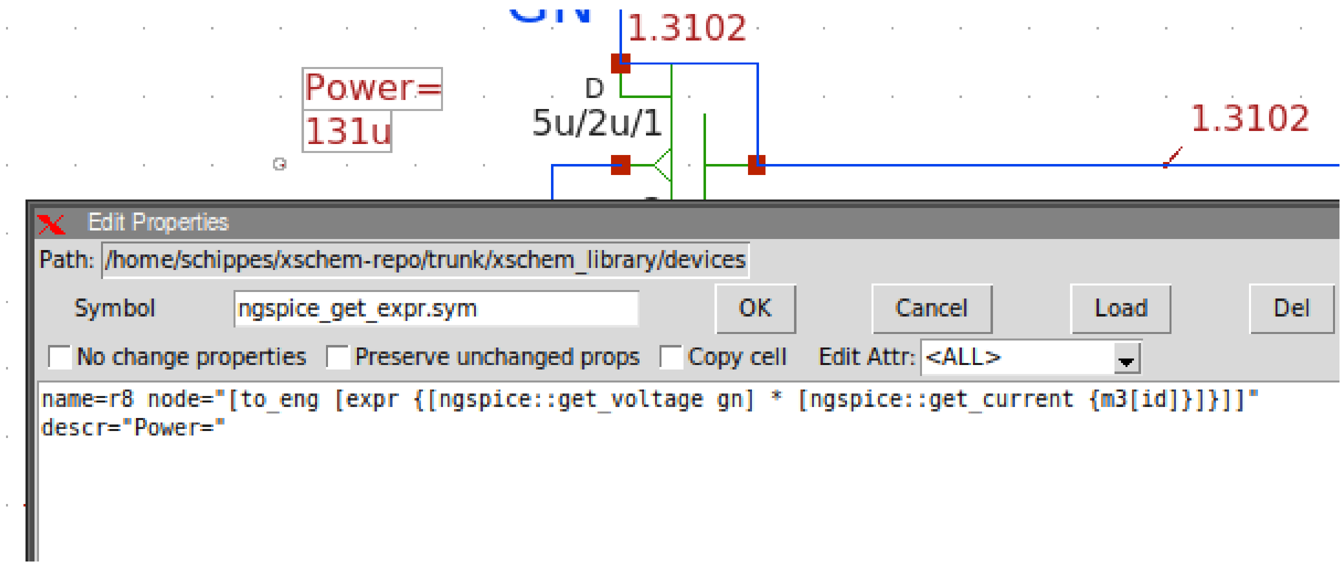
Data annotated into the schematic using these components allows more simulation parameters to be viewed into the schematic, not being restricted to currents and voltages. Since these components get data using a pull method data is not persistent and not saved to file. After reloading the file just do a **`xschem annotate_op`** to view data again.

There is one last probe component, the **`devices/ngspice_get_expr.sym`**. This is the most complex one, and thus also the most flexible. It allows to insert a generic tcl expression using spice simulated data to report more complex data. In the example below this component is used to display the electrical power of transistor m3, calculated as **`V(GN) * Id(m3)`**.



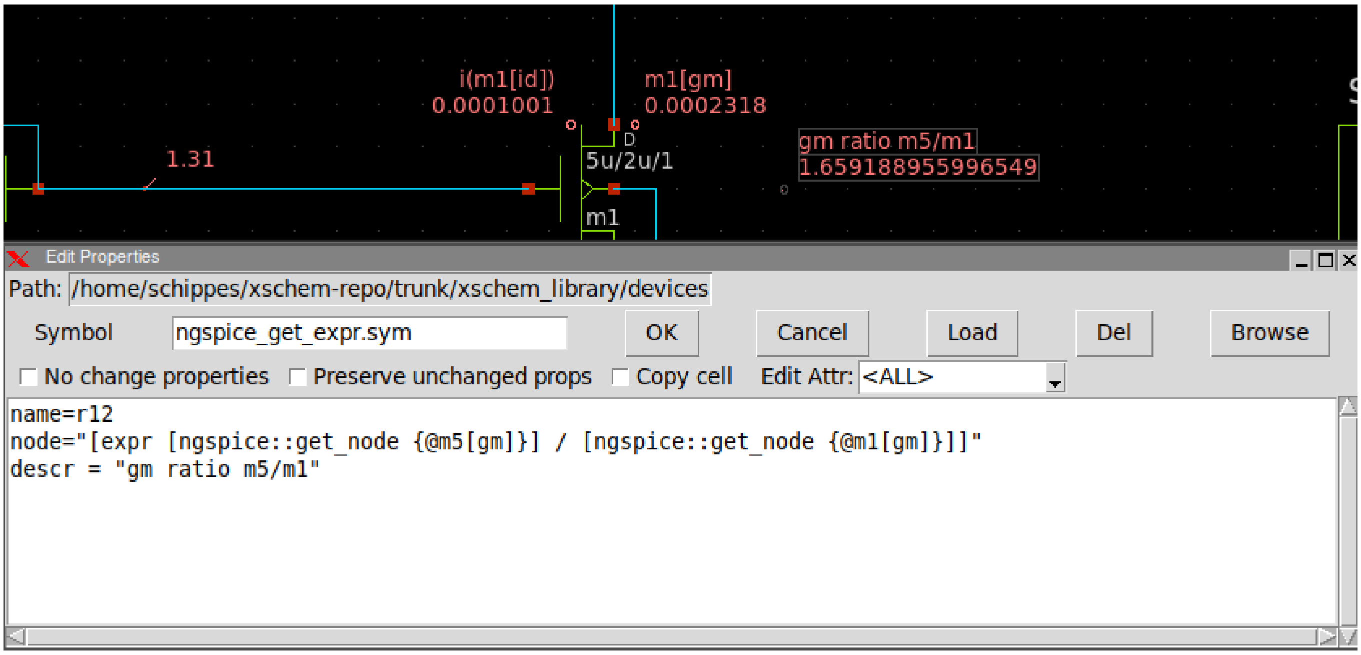
```
name=r8 node="[expr {[ngspice::get_voltage gn] * [ngspice::get_current {m3[id]}]}]"
descr="Power="
```

you can wrap the whole expression inside a **[to_eng ...]** to have the value displayed in engineering form using the usual SPICE suffixes (example: 131u for 131e-6)



```
name=r8 node="[to_eng [expr {[ngspice::get_voltage gn] * [ngspice::get_current {m3[id]}]}]]"
descr="Power="
```

The example shown below uses this component to display a (meaningless, but shows the usage) gm ratio of 2 transistors:

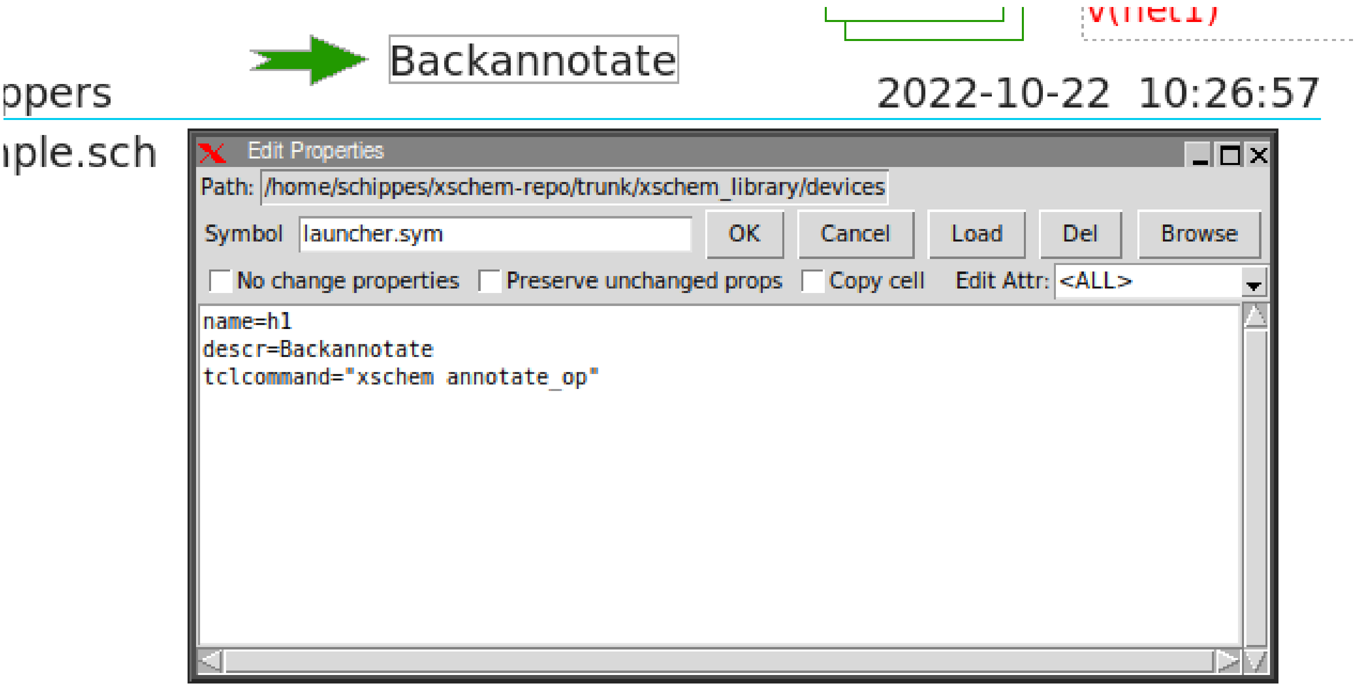


Symbol: ngspice_get_expr.sym

```
name=r12
node="[expr [ngspice::get_node {@m5[gm]}] / [ngspice::get_node {@m1[gm]}]]"
descr = "gm ratio m5/m1"
```

The syntax is a bit complex, considering the verbosity of TCL and the strange ngspice naming syntax, however once a working one is created changing the expression is easy.

To avoid the need of typing commands in the xschem console a launcher component **`devices/launcher.sym`** can be placed with the tcl command for doing the annotation. Just do a **`Ctrl-Click`** on it to trigger the annotation.



Symbol: launcher.sym

```
name=h1
descr=Backannotate
tclcommand="xschem annotate_op"
```

UP

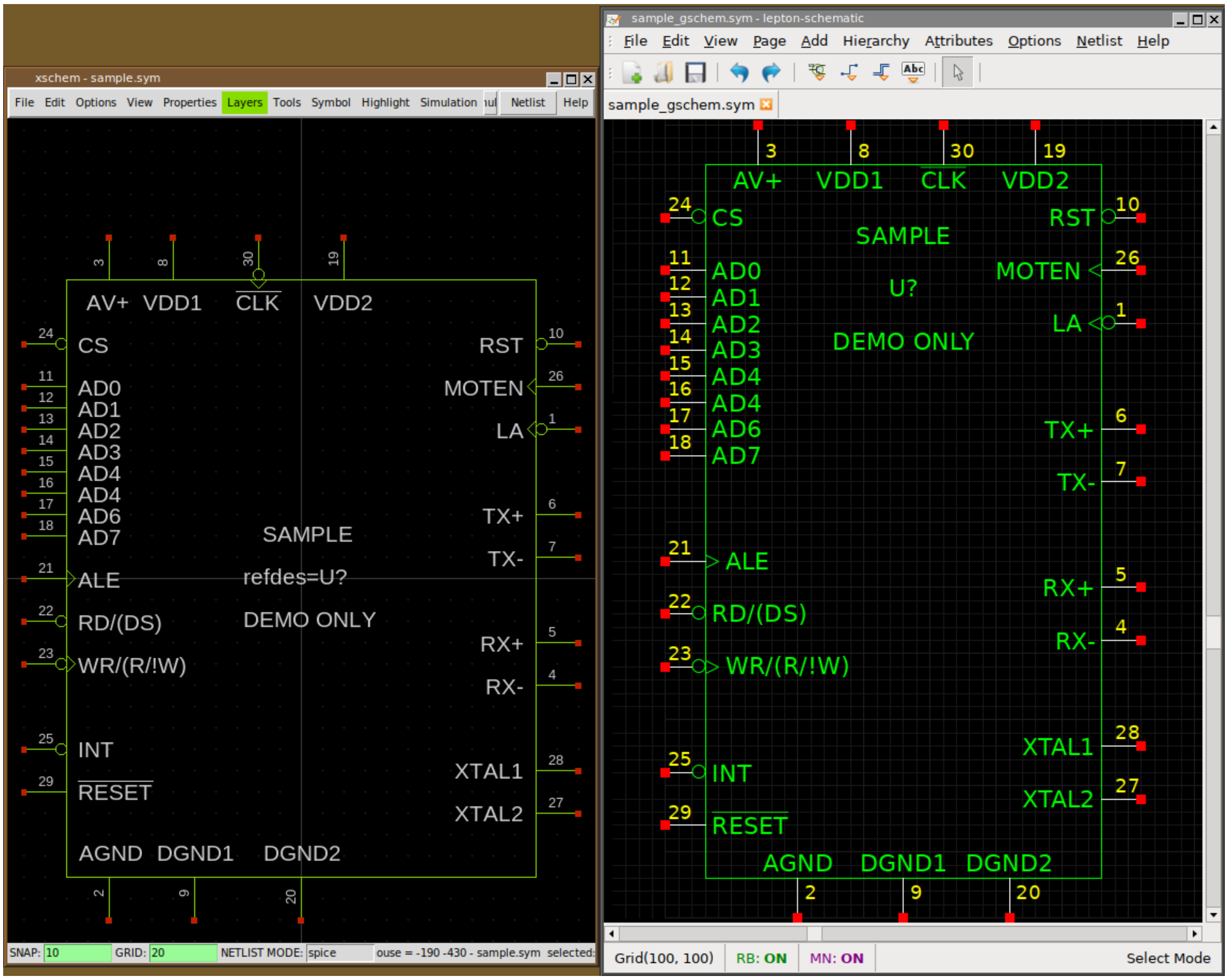
TUTORIAL: Use symgen.awk to create symbols from 'djboxsym' compatible text files

The **`symgen.awk`** utility (installed in **`(install_root)/share/xschem`**) generates xschem symbol files from a textual description that is backward compatible to DJ Delorie's perl djboxsym symbol generator for the geda schematic editor (gschem, lepton-schematic). A sample **`sample.symdef`** file is the following:

```
# This is a sample symbol definition for documenting djboxsym.  Some
# of the pins have been intentionally mistyped in order to demonstrate
# all combinations of flags.  DO NOT USE AS A CP2201 REFERENCE!

[labels]

SAMPLE
refdes=U?
DEMO ONLY
! copryright=2006 DJ Delorie
! author=DJ Delorie
! uselicense=unlimited
! distlicense=GPL
! device=sample device
! description=ethernet controller
! footprint=QFN-28

[left]
24 ! CS
.bus
11  AD0
12  AD1
13  AD2
14  AD3
15  AD4
16  AD4
17  AD6
18  AD7

21 > ALE
22 ! RD/(DS)
23 !> WR/(R/!W)

25 ! INT
29  \_RESET\_

[right]
10 ! RST
26 > MOTEN
1 !> LA

6 TX+
7 TX-

5 RX+
4 RX-

28 XTAL1
27 XTAL2
[top]
```

```
3 AV+
8 VDD1
30 !> \_CLK\_
19 VDD2

[bottom]
2 AGND
9 DGND1
20 DGND2
```

Creating the symbol is simple:
`<install_path>/share/xschem/symgen.awk sample.symdef > sample.sym`
The resulting symbol is shown here under, side-compared with the same symbol generated by djboxsym for gschem:

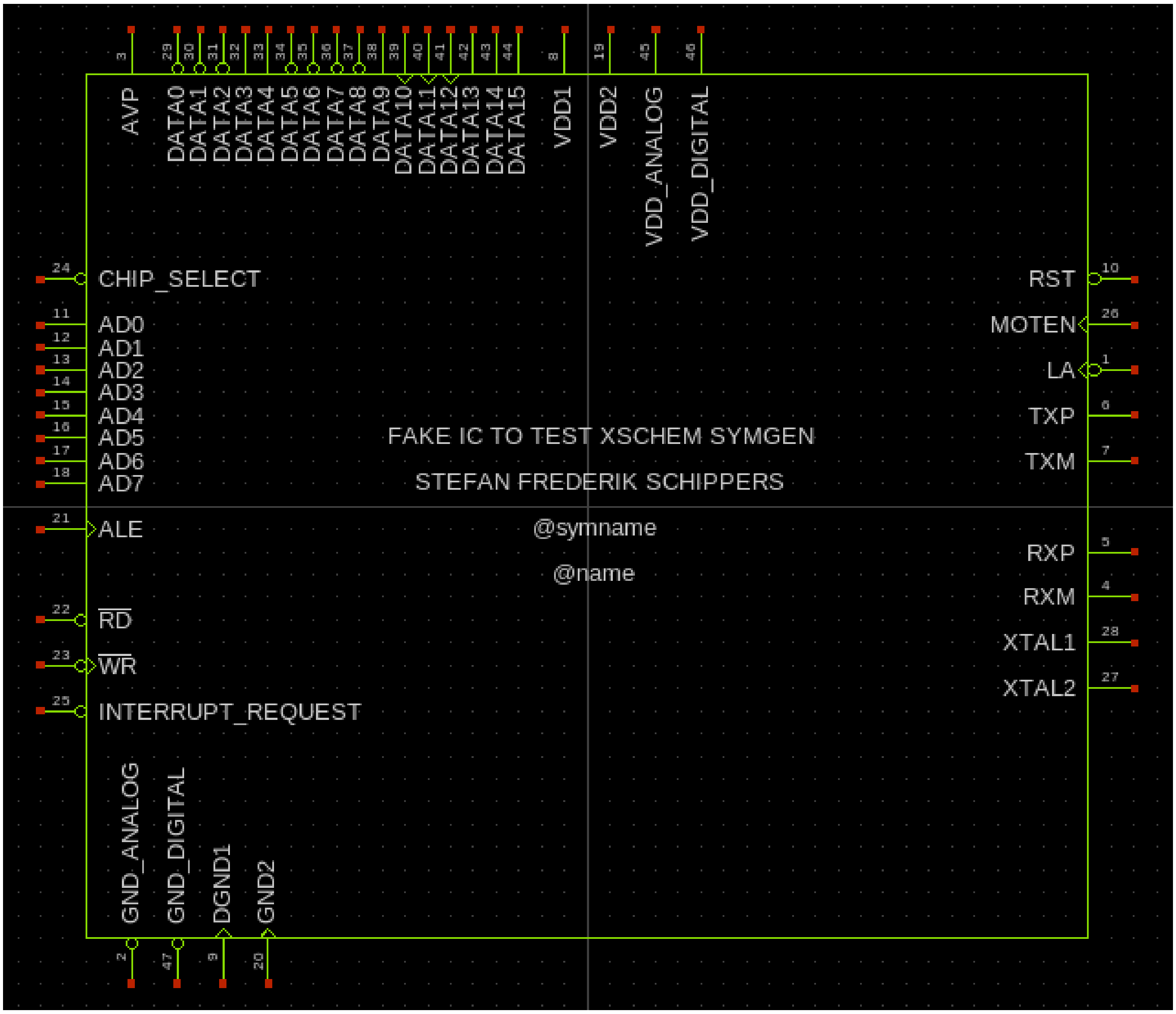

xschem - sample.sym
File Edit Options View Properties Layers Tools Symbol Highlight Simulation Netlist Help
3 8 30 19
AV+ VDD1 CLK VDD2
24 CS
RST 10
11 AD0 12 AD1 13 AD2 14 AD3 15 AD4 16 AD4 17 AD6 18 AD7
MOTEN 26
LA 1
TX+ 6
TX- 7
SAMPLE
refdes=U?
DEMO ONLY
21 ALE
22 RD/(DS)
23 WR/(R/!W)
RX+ 5
RX- 4
25 INT
29 RESET
XTAL1 28
XTAL2 27
AGND DGND1 DGND2
2 9 20
SNAP: 10 GRID: 20 NETLIST MODE: spice
ouse = -190 -430 - sample.sym selected:
sample_gschem.sym - lepton-schematic
File Edit View Page Add Hierarchy Attributes Options Netlist Help
sample_gschem.sym
3 8 30 19
AV+ VDD1 CLK VDD2
24 CS
SAMPLE
U?
DEMO ONLY
RST 10
11 AD0 12 AD1 13 AD2 14 AD3 15 AD4 16 AD4 17 AD6 18 AD7
MOTEN 26
LA 1
TX+ 6
TX- 7
21 ALE
22 RD/(DS)
23 WR/(R/!W)
RX+ 5
RX- 4
XTAL1 28
XTAL2 27
25 INT
29 RESET
AGND DGND1 DGND2
2 9 20
Grid(100, 100) RB: ON MN: ON Select Mode

Another **`sample2.symdef`** file specifically created to generate a perfectly valid xschem symbol (including attributes for spice netlisting) is the following:

```
#  <pinnumber>  <direction>[<circle><edge_trigger>] <name>
# circle: !
# edge_trigger: >
# direction is mandatory: i=input, o=output, b=bidirectional (inout)
```

```
[labels]
FAKE IC TO TEST XSCHEM SYMGEN
STEFAN FREDERIK SCHIPPERS
@symname
@name
! type=subcircuit
! format="@name @pinlist @symname"
! template="name=x1"
--vmode
[left]
24   i!   CHIP_SELECT
.bus
11   i    AD0
12   i    AD1
13   i    AD2
14   i    AD3
15   i    AD4
16   i    AD5
17   i    AD6
18   i    AD7

21   i>   ALE

22   i!   \_RD\_
23   i!>  \_WR\_
25   i!   INTERRUPT_REQUEST
[right]
10   i!   RST
26   i>   MOTEN
1    i!>  LA
6    o    TXP
7    o    TXM

5    i    RXP
4    i    RXM
28   i    XTAL1
27   i    XTAL2
[top]
3    io    AVP
.bus
29  o!    DATA0
30  o!    DATA1
31  o!    DATA2
32  o     DATA3
33  o     DATA4
34  o!    DATA5
35  o!    DATA6
36  o!    DATA7
37  o!    DATA8
38  o     DATA9
39  o>    DATA10
40  o>    DATA11
41  o>    DATA12
42  o     DATA13
43  o     DATA14
44  o     DATA15

8    io    VDD1
19   io    VDD2
45   io    VDD_ANALOG
46   io    VDD_DIGITAL
[bottom]
2    io!   GND_ANALOG
47   io!   GND_DIGITAL
```

```
9    io>   DGND1
20   io>   GND2
```



some extensions of xschem's symdef text file format with respect to original djboxsym format:

- In addition to optional **!** (inversion bubble) and **>** (edge trigger) specifiers XSCHEM's **symgen.awk** accepts a pin direction specifier, **i**, **o**, **io** and **p** (latter one for power pins, treated by xschem as inout) for 'input', 'output', 'inout' (bidirectional) direction and 'power'. These attributes are fundamental for digital simulations (Verilog, Vhdl). If this specifier is missing (as it is in djboxsym .symdef files) then the direction is assumed as **b** (inout). XSCHEM does not have any specific direction for power pins so they are treated as 'inout'
Port direction specifiers are indeed supported also by 'djboxsym' but not documented.
- Option **--vmode** given **before** any pin declaration like in djboxsym sets vertical orientation for top / bottom pins.
- **.bus** specifier can be used for all pin orientations, left, top, right, bottom if **--vmode** is enabled, otherwise it will affect only spacing of left/right pins.
- Option **--auto_pinnumber** given **before** any pin declaration lets **symgen.awk** automatically add pin numbers, so the first field may be omitted

- Edge trigger (>) and inversion bubble (!) specifiers are drawn on all sides, not only left/right.
- Option **`--hide_pinnumber`** given **before** any pin declaration avoids pin numbers in generated symbol. If this option is used it is mostly done together with **`--auto_pinnumber`** to get rid of pin numbers completely.

UP

TUTORIAL: Translate GEDA gschem/lepton-schematic schematics and symbols to xschem

The **gschemtoxschem.awk** utility (installed in **(install_root)/share/xschem**) generates xschem schematic and symbol files from their GEDA equivalents.

First of all, note that xschem comes with all geda symbols already translated to xschem.

Create an empty directory where you want your xschem schematics/symbols, inside this directory create an **xschemrc** file with the following path added, if not already done in your **~/.xschem/xschemrc** file:

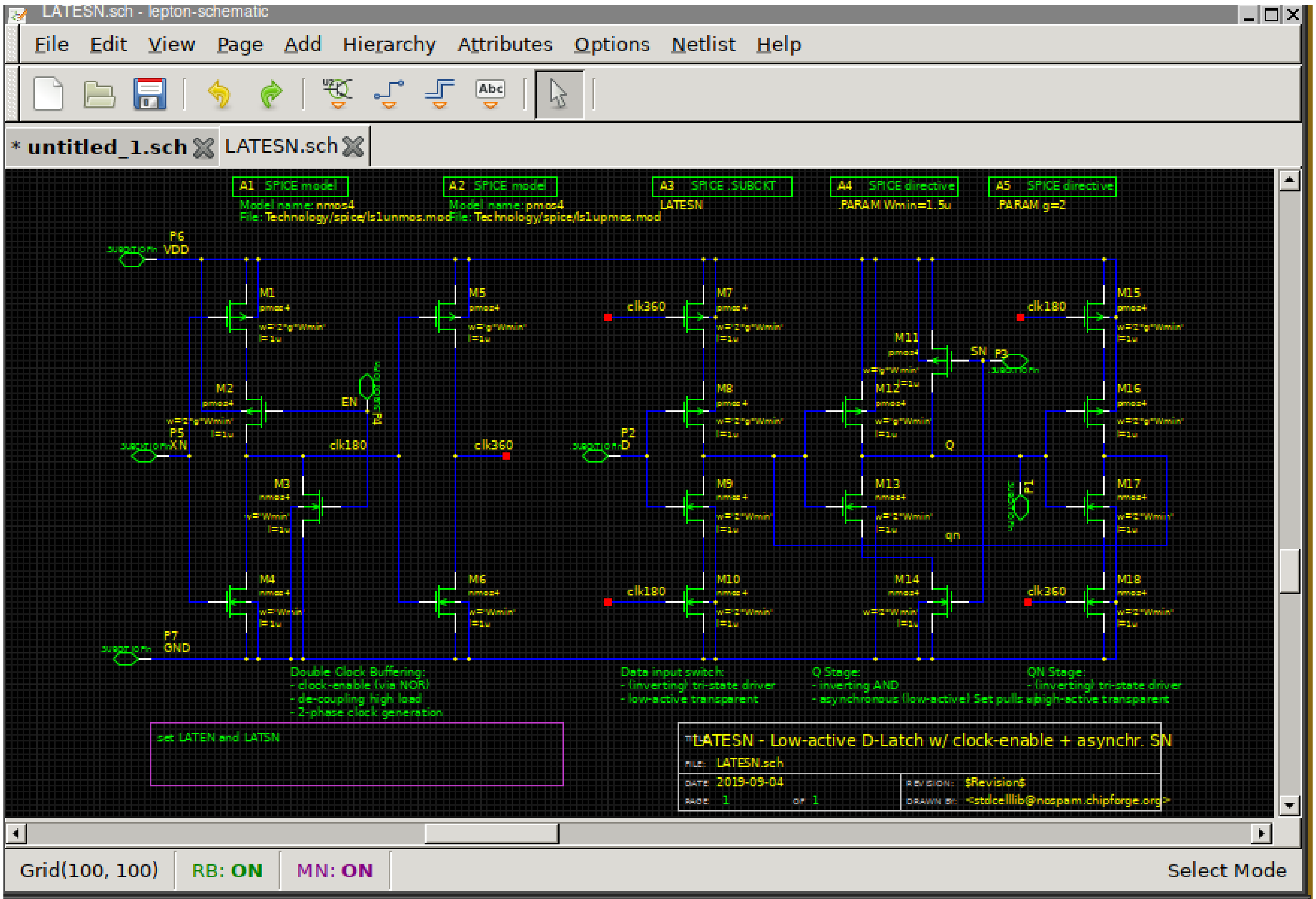
```
append XSCHEM_LIBRARY_PATH :${XSCHEM_SHAREDIR}/../doc/xschem/gschem_import/sym
```

Next, in this directory create a **convert.sh** script and make it executable:

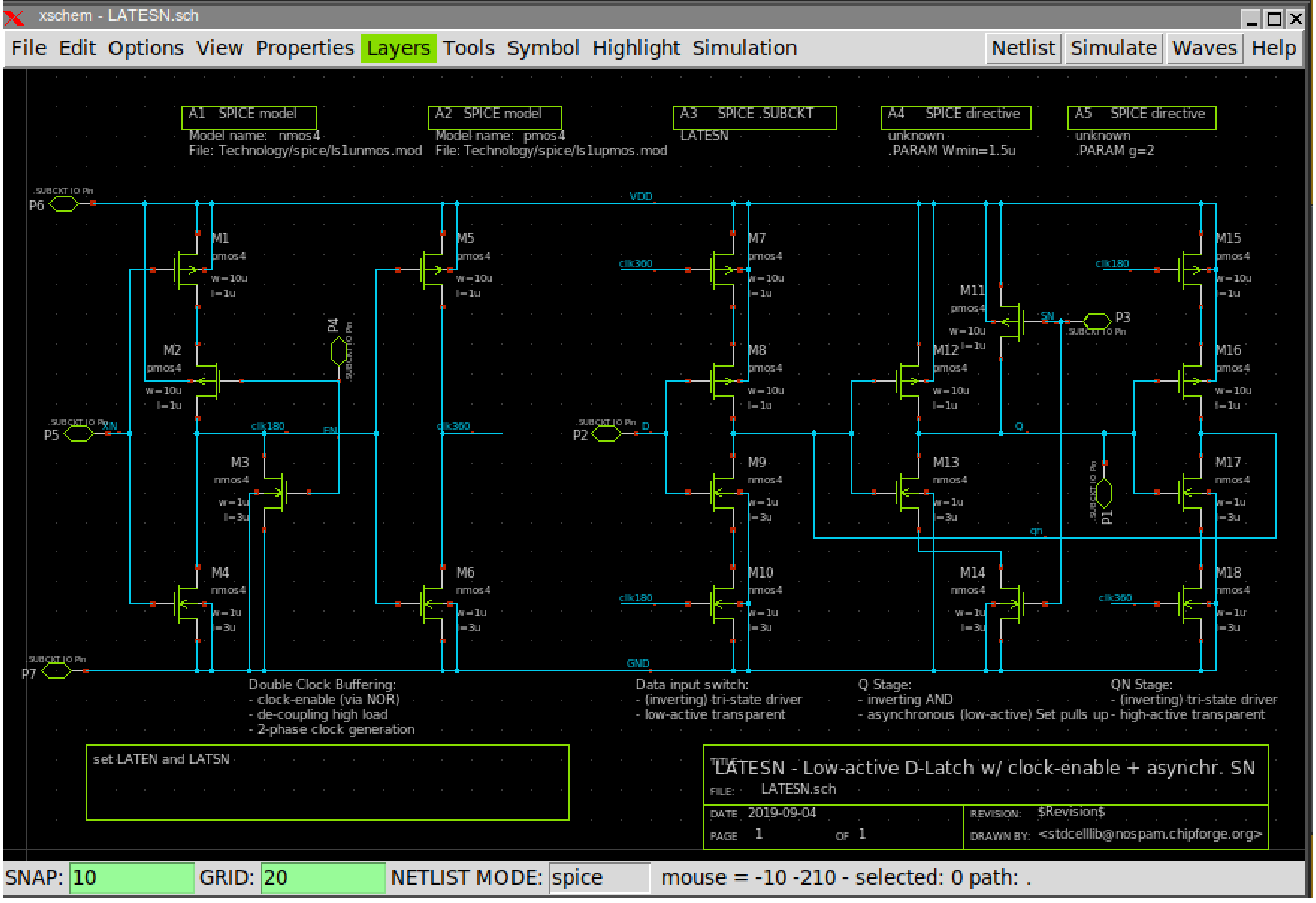
```
#!/bin/bash

# remove empty glob specifications *.sym or *.sch
shopt -s nullglob

for file in directory_with_geda_files/*.{sym,sch}
do
  /path_to_xschem_install_root/share/xschem/gschemtoxschem.awk $file >  $(basename -- $file)
done
```

Note that you have to set the correct path for **gschemtoxschem.awk** depending on your xschem installation and set the correct path for the directory (**directory_with_geda_files** in above example) containing the geda files.
The current directory will be populated with xschem schematics/symbols with the same name as their GEDA equivalents. Incidentally xschem and gschem use the same file extensions (.sym, .sch), so be careful not to mix xschem and gschem files.

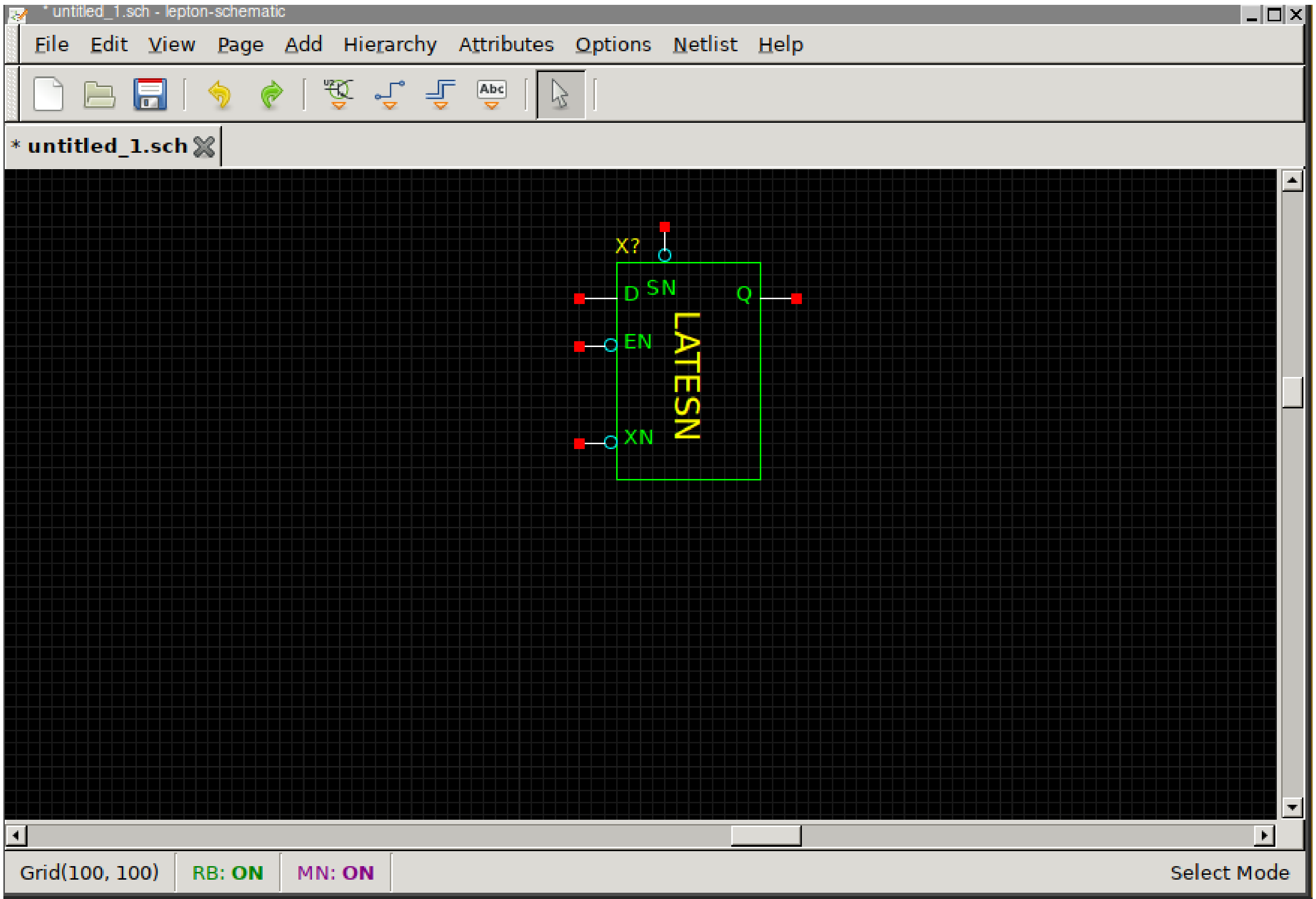
Below an example of a schematic and a symbol shown both in xschem and lepton-schematic (gschem fork)

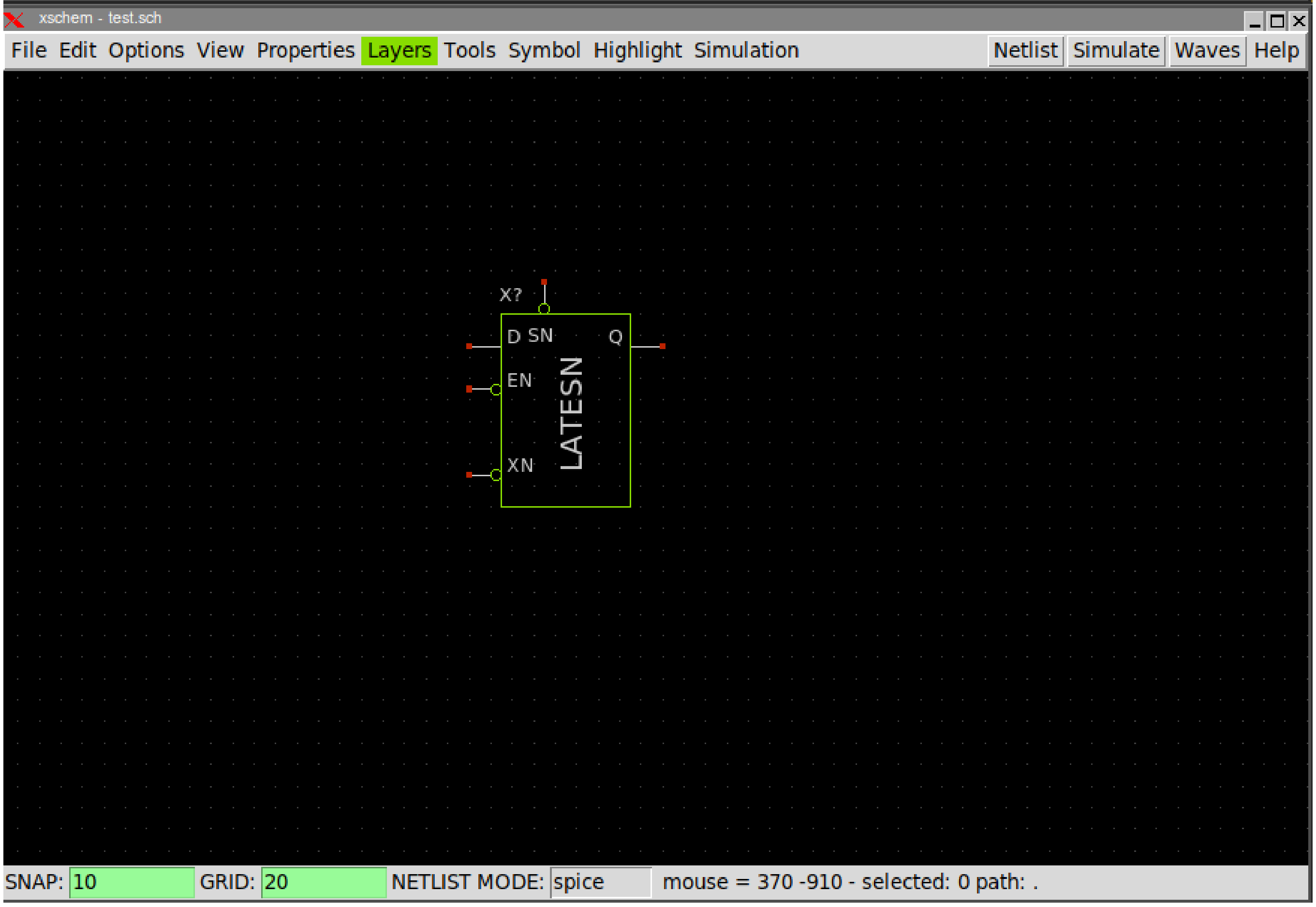


Grid(100, 100) RB: ON MN: ON Select Mode



SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -10 -210 - selected: 0 path: .





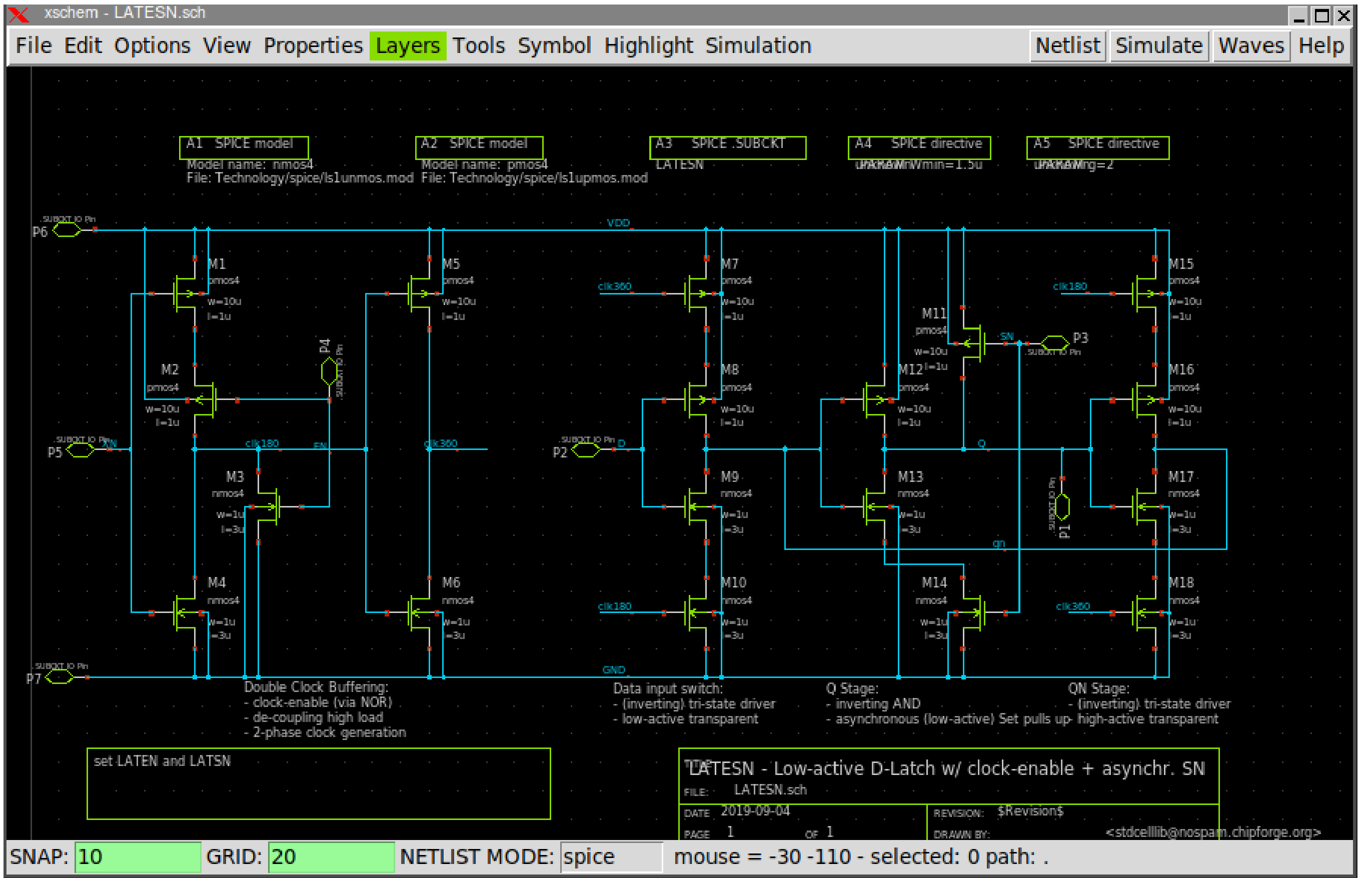
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 370 -910 - selected: 0 path: .

Notes for schematics targeted for spice simulations

Most of geda schematics do not define precise rules for spice netlisting. primitive symbols are symbols that do not have a schematic representation, examples are the nmos and pmos transistors in first schematic. They should have a **`format`** property that defines how the symbol should be translated to spice netlist. See the relevant schem manual page. Subcircuit symbols are symbols that translate to spice as a .subckt calls. An example is the LATESN symbol in above picture. Xschem convention is that subcircuit symbol instances have a name attribute that begins with 'X' or 'x'. As with primitive symbols they also have a **`format`** global attribute, but the **`type=subcircuit`** attribute states it is a subcircuit instance. After producing the instance call (for example **`X1 net1 net2 net3 ... subcircuit_name)`**) for all instances of this symbol a .subckt expansion is also produced:

```
.subckt subcircuit_name pin1 pin2 pin3 ...
...
...
.ends
```

After doing the conversion with **`gschemtoxschem.awk`** you should check your schematics and symbols and make the necessary corrections.
In particular you should check that schematic pins match symbol pins, regarding pin name and direction. Xschem standard way is to use **`ipin.sym, opin.sym, iopin.sch`** for input, output, inout pins, respectively. Following image shows the original converted schematic and the hand-modified schematic with the proper pins. Note that VDD/GND pins have been removed since the LATESN symbol does not have such supply pins.
In spice netlist VDD/GND to the subcircuit is in this particular case passed via net-assign.



Model name: nmos4
File: Technology/spice/ls1unmos.mod

Model name: pmos4
File: Technology/spice/ls1upmos.mod

LATESN

.PARAM Wmin=1.5u

.PARAM ng=2

Double Clock Buffering:
- clock-enable (via NOR)
- de-coupling high load
- 2-phase clock generation

Data input switch:
- (inverting) tri-state driver
- low-active transparent

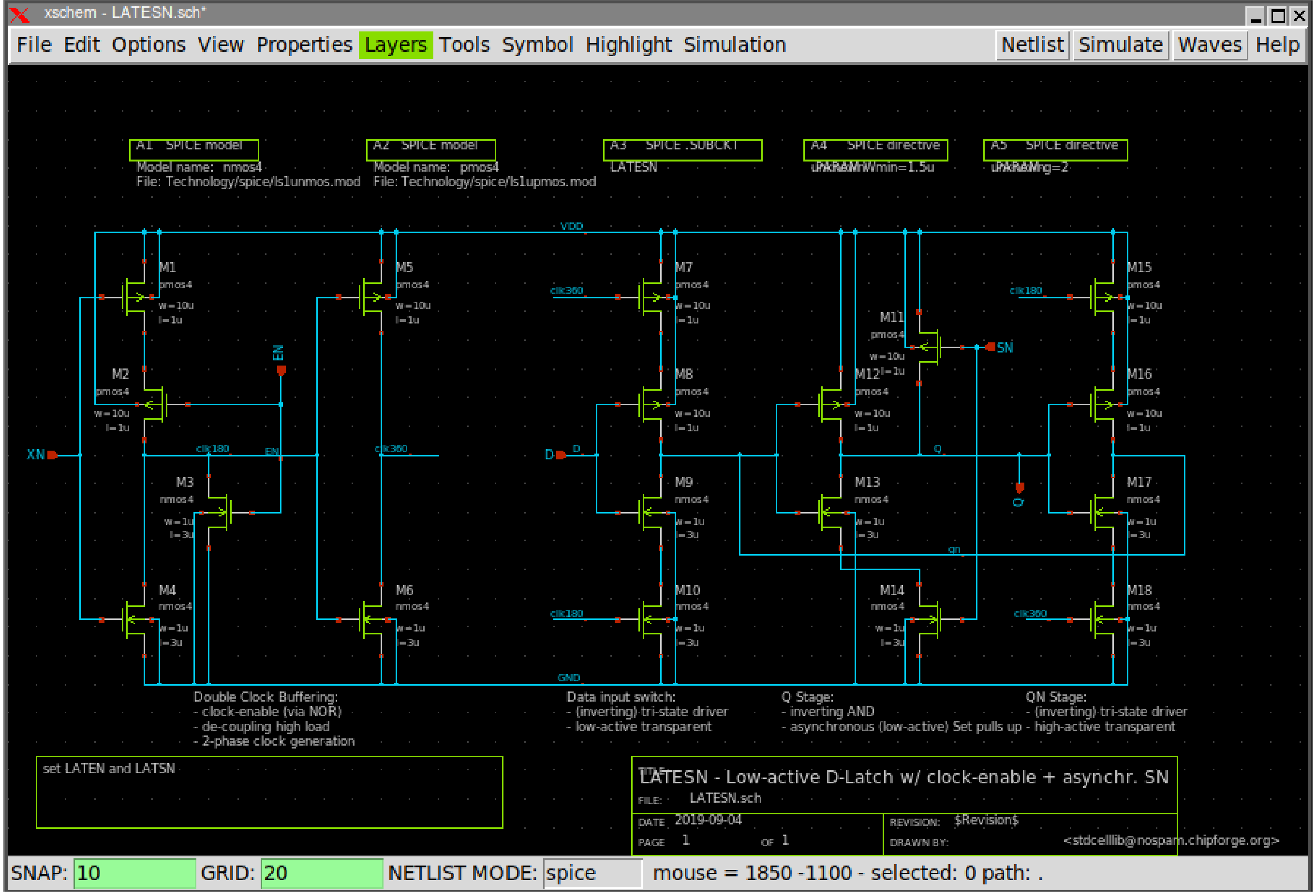
Q Stage:
- inverting AND
- asynchronous (low-active) Set pulls up

QN Stage:
- (inverting) tri-state driver
- high-active transparent

set LATEN and LATSN

TITLE LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
PAGE 1 OF 1
REVISION: $Revision$
DRAWN BY: <stdcelllib@nospam.chipforge.org>

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -30 -110 - selected: 0 path: .



Model name: nmos4
File: Technology/spice/ls1unmos.mod

Model name: pmos4
File: Technology/spice/ls1upmos.mod

LATESN

.PARAM Wmin=1.5u

.PARAM ng=2

Double Clock Buffering:
- clock-enable (via NOR)
- de-coupling high load
- 2-phase clock generation

Data input switch:
- (inverting) tri-state driver
- low-active transparent

Q Stage:
- inverting AND
- asynchronous (low-active) Set pulls up

QN Stage:
- (inverting) tri-state driver
- high-active transparent

set LATEN and LATSN

TITLE LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
PAGE 1 OF 1
REVISION: $Revision$
DRAWN BY: <stdcelllib@nospam.chipforge.org>

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 1850 -1100 - selected: 0 path: .

XSCHEM SKY130 INTEGRATION

To use Xschem with the Google-Skywater 130nm process (here: Sky130) The following items must be followed:

- Install Xschem. Follow the Manual Install instructions

  If you install xschem from sources ensure no xschem package is already installed in your linux system. Packaged xschem versions are too old so you should remove the installed package. The command for ubuntu/Debian systems is **`sudo apt-get remove --purge xschem`**

- Install the Magic VLSI layout editor. Instructions here.
- Install ngspice, by cloning the git source repository and building the program.

```
## clone the source repository into a local ngspice_git directory
git clone https://git.code.sf.net/p/ngspice/ngspice ngspice_git
cd ngspice_git
mkdir release
./autogen.sh
cd release
## by default if no --prefix is provided ngspice will install under /usr/local/{bin,share,ma
## you can add a --prefix=/home/username to install into your home directory.
../configure --with-x --enable-xspice --disable-debug --enable-cider --with-readline=yes --e
## build the program
make
## install the program and needed files.
sudo make install
```

IMPORTANT!!

You need to create the following **`.spiceinit`** file in the directory where simulations are run (typically **`~/.xschem/simulations`**) or in your home directory. This file sets some default behavior for reading .lib files and speeds up loading pdk model files.

```
set ngbehavior=hsa
set ng_nomodcheck
```

- Install Open_Pdks that will provide among other things all the sky130 PDK data, including standard cells, SPICE models, layout data, timing information, design rules and provides also also the Xschem symbols of available

silicon primitive devices and the set of locic standard cells built on top of these primitive devices. Instructions are here.
Please ensure sufficient disk space is available (Open_pdks uses several GB, a lot of space can be recovered after installation by removing the source files if needed). Also keep in mind that the installation takes considerable time. The following steps are needed:

```
## fetch the repository with git:
git clone git://opencircuitdesign.com/open_pdks
cd open_pdks
## configure the build, a --prefix option can be given to install
## in a different place, by default after installation a
## /usr/local/share/pdk directory is created if no --prefix is provided.
## Below line for example requests installation in my home directory
## (/home/schippes/share/pdk):
## ./configure --enable-sky130-pdk --prefix=/home/schippes
## Do the following steps one at a time and ensure no errors are
##  reported after each step.
./configure --enable-sky130-pdk
make
sudo make install
```

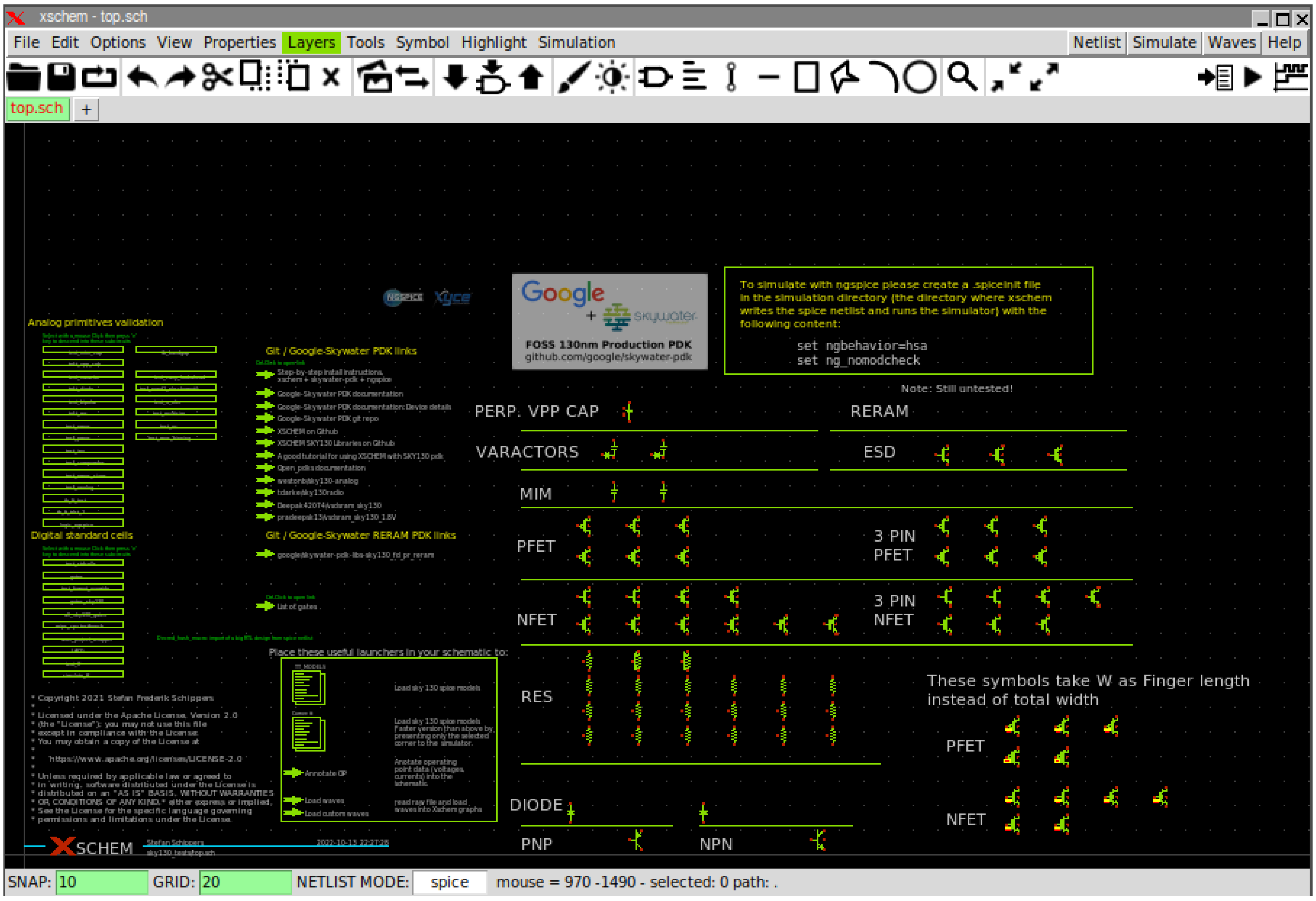
If you want to install also the gf180mcu pdk replace the above ./configure command with the following:

```
 ./configure --enable-sky130-pdk --enable-gf180mcu-pdk
```

- At this point the complete PDK has been installed in **`/usr/local/share/pdk`** (or **`<prefix>/share/pdk`** if --prefix was provided).
Xschem libraries also have been installed and are located under **`<prefix>/share/pdk/sky130A/libs.tech/xschem/`** or **`<prefix>/share/pdk/sky130B/libs.tech/xschem/`**.
the **`sky130B`** directory contains the ReRAM Sky130 process option in addition to all **`Sky130A`** devices.
- After completing the above steps you can do a test run of xschem and use the Sky130 devices. You need to create a new empty directory, create a new xschemrc file with the following content: (**`source <prefix>/share/pdk/sky130B/libs.tech/xschem/xschemrc`**) and run xschem:

```
mkdir test_xschem_sky130
cd test_xschem_sky130
echo 'source /usr/local/share/pdk/sky130B/libs.tech/xschem/xschemrc' > ./xschemrc
xschem
```

- If all went well the following welcome page will be shown. The page contains some example circuits on the left and shows all the available silicon devices on the right. You can descend into the example circuits on the left by clicking the symbols (they will turn to grey meaning they are selected) and press the **`e`** key or by menu **`Edit->Push schematic`**. You can return to the parent level by pressing **`Ctrl-e`** or by menu **`Edit->Pop`**.


xschem - top.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
top.sch +
Analog primitives validation
Digital standard cells
Git / Google-Skywater PDK links
Git / Google-Skywater RERAM PDK links
Google + skywater
FOSS 130nm Production PDK
github.com/google/skywater-pdk
PERP. VPP CAP
VARACTORS
MIM
PFET
NFET
RES
DIODE
PNP
NPN
RERAM
ESD
3 PIN PFET
3 PIN NFET
PFET
NFET
Place these useful launchers in your schematic to:
Annotate OP
Load waves
Load custom waves
XSCHEM

To simulate with ngspice please create a .spiceinit file in the simulation directory (the directory where xschem writes the spice netlist and runs the simulator) with the following content:

```
set ngbehavior=hsa
set ng_nomodcheck
```

Note: Still untested!

These symbols take W as Finger length instead of total width

```
* Copyright 2021 Stefan Frederik Schippers
* Licensed under the Apache License, Version 2.0
* (the "License"); you may not use this file
* except in compliance with the License.
* You may obtain a copy of the License at
*
*   https://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to
* in writing, software distributed under the License is
* distributed on an "AS IS" BASIS, WITHOUT WARRANTIES
* OR CONDITIONS OF ANY KIND, * either express or implied.
* See the License for the specific language governing
* permissions and limitations under the License.
```

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 970 -1490 - selected: 0 path: .

You can disable the welcome page by commenting the following line in the xschemrc file:

```
set XSCHEM_START_WINDOW {sky130_tests/top.sch}
```

or:

```
unset XSCHEM_START_WINDOW
```

PDK_ROOT and PDK environment variables

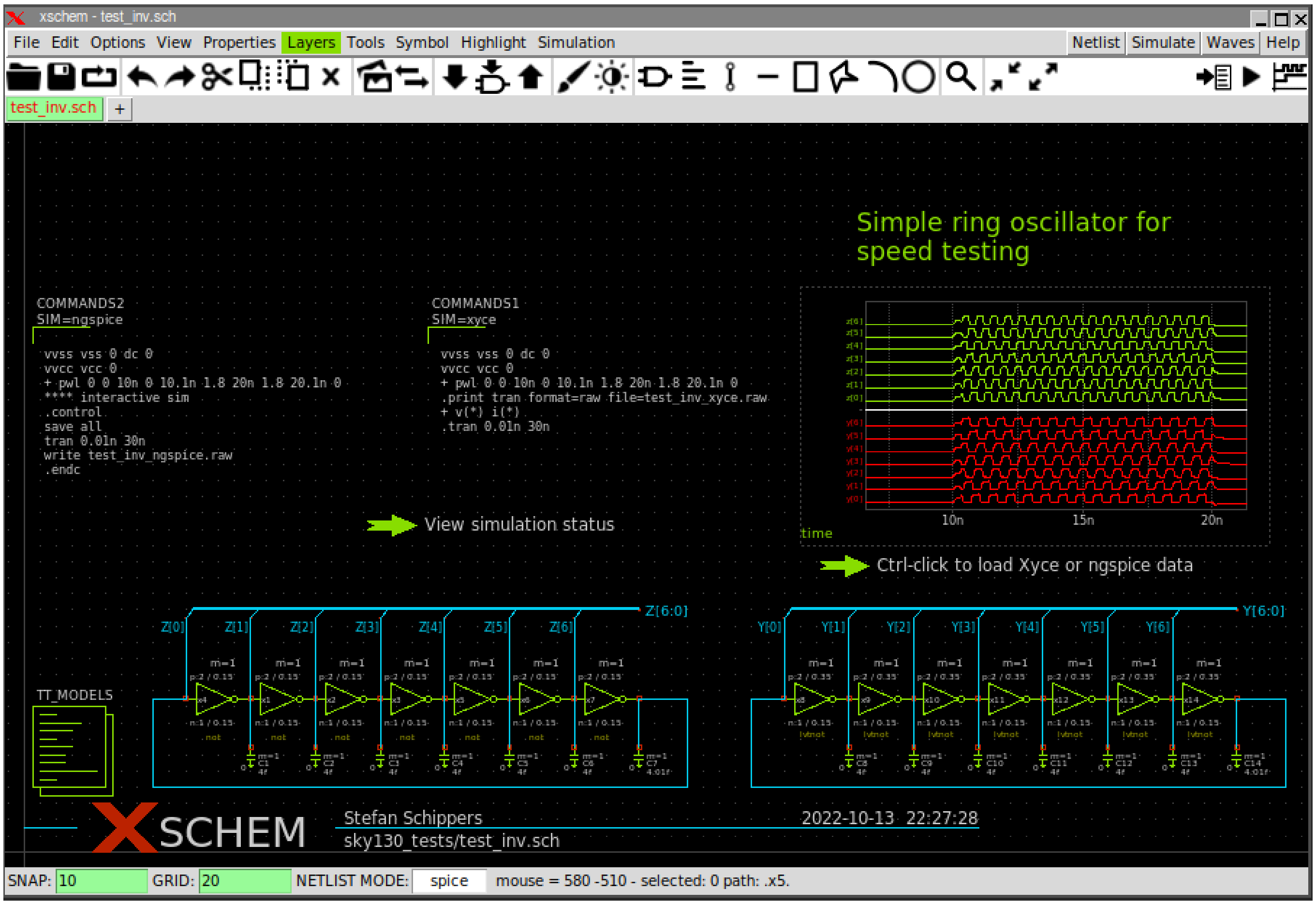
Xschem (via the xschemrc file) looks for a **PDK_ROOT** environment variable that points to the installed pdk to use. This is expecially useful if multiple or different versions of the pdk are installed. If the pdk is installed in **/usr/local/share/pdk** PDK_ROOT should be set to **/usr/local/share/pdk**. For **Sky130** another variable **PDK** tells the process variant to use (currently **sky130A**) or **sky130B**). If **PDK** is unset the default **sky130A** will be used. If no PDK_ROOT variable is defined xschem will look into the following locations and pick the first existing found in the order listed below:

1. **/usr/share/pdk**
2. **/usr/local/share/pdk**
3. **~/share/pdk**

If no pdk is found a warning message is displayed on the xschem launching terminal.

Simulating a circuit with sky130 devices

The best way to quickly set up a simulation with Xschem is to look at some of the provided examples. If you descend into the **`test_inv`** component you see a working circuit ready for simulation.



```
COMMANDS2
SIM=ngspice

 vvss vss 0 dc 0
 vvcc vcc 0
 + pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n 0
 **** interactive sim
 .control
 save all
 tran 0.01n 30n
 write test_inv_ngspice.raw
 .endc
```

```
COMMANDS1
SIM=xyce

 vvss vss 0 dc 0
 vvcc vcc 0
 + pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n 0
 .print tran format=raw file=test_inv_xyce.raw
 + v(*) i(*)
 .tran 0.01n 30n
```

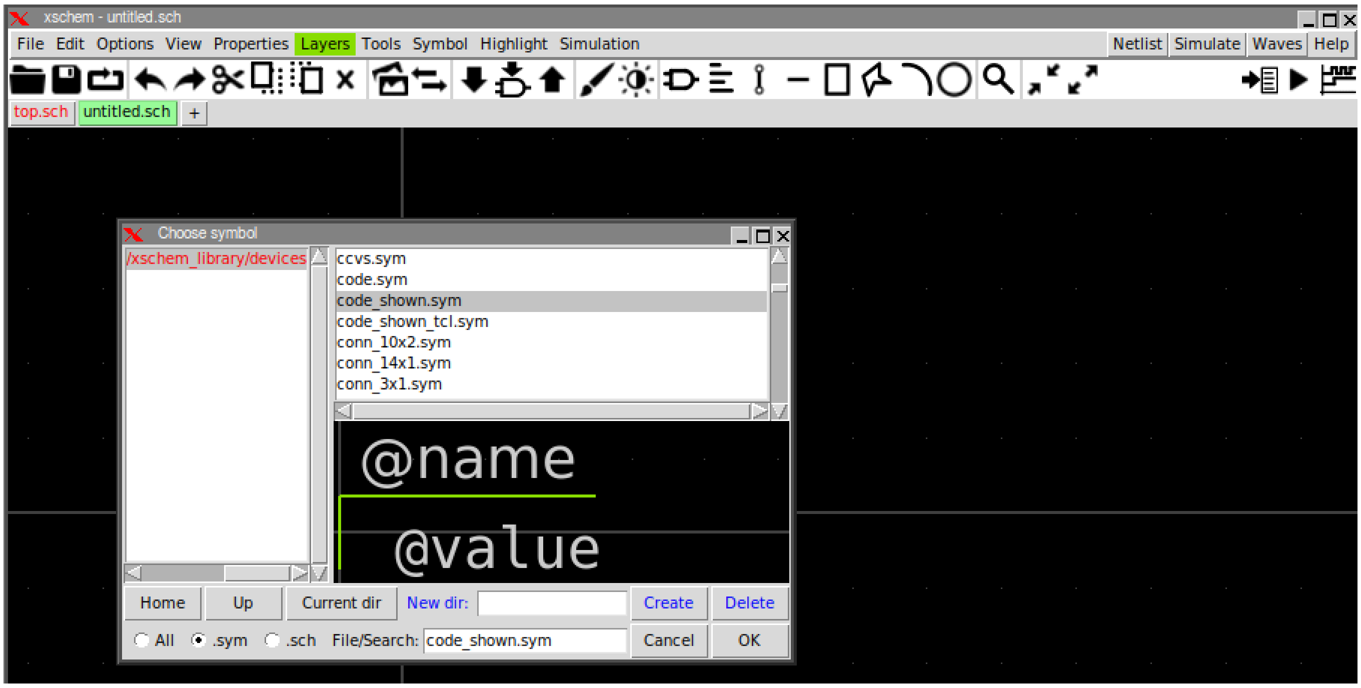
xschem - test_inv.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_inv.sch
Simple ring oscillator for speed testing
View simulation status
Ctrl-click to load Xyce or ngspice data
10n
15n
20n
time
Z[6:0]
Y[6:0]
TT_MODELS
SCHEM
Stefan Schippers
2022-10-13 22:27:28
sky130_tests/test_inv.sch
SNAP: 10 GRID: 20 NETLIST MODE: spice
mouse = 580 -510 - selected: 0 path: .x5.

One line is needed in the spice netlist to load the spice models:

```
.lib /usr/local/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
```

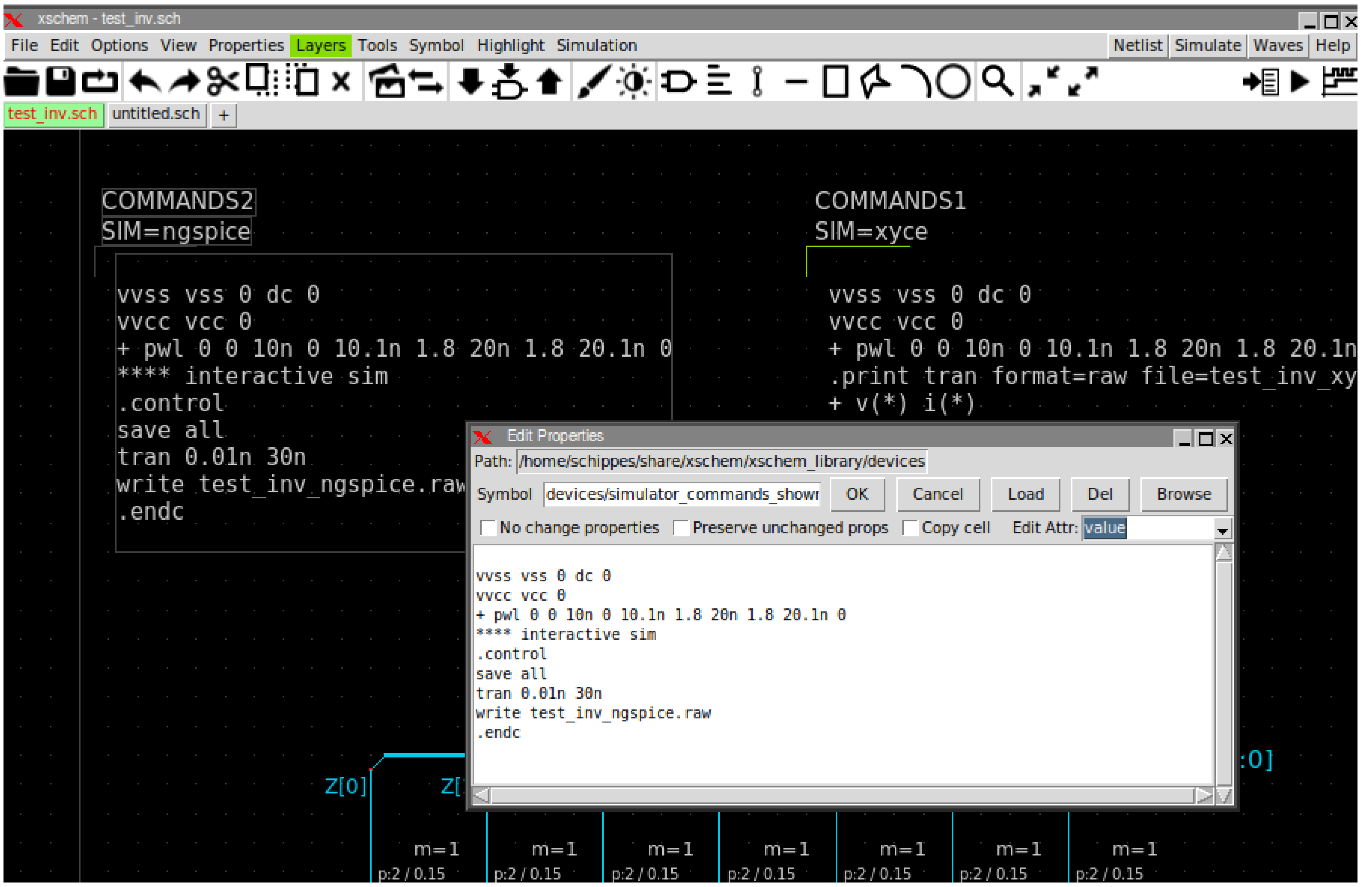
The exact path depends on the install location of the pdk as explained above. In the picture above the **`TT_MODELS`** component takes care of generating the .lib line in the netlist. the **`tt`** at the end of the **`.lib`** line is the process corner (tt = typical n, typical p transistors). You can change the corner to **`ss, sf, fs, ff`** to verify your design across process variations.

You see in the circuit a **`COMMANDS2`** component. This component allows to enter text to specify the simulation to run, giving simulator commands and options. You place this component by pressing the **`Insert`** or **`i`** key, browsing into the standard xschem **`devices`** directory and placing **`code_shown.sym`** or **`code.sym`** into the schematic.



All [✓ .sym] .sch File/Search: code_shown.sym

Once placed in the schematic, you may click the component, press **q** to edit its attributes, set the **Edit attr.** listbox on the right to **value** and enter the simulator commands to run the simulation. You can give a reference name to this component by setting the **Edit attr.** listbox to **name** and give it a name that will be diplayed in the schematic. (**COMMANDS2** in the example).


xschem - test_inv.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_inv.sch untitled.sch +
Edit Properties
OK Cancel Load Del Browse
No change properties Preserve unchanged props Copy cell
Z[0]
m=1
p:2 / 0.15

```
COMMANDS2
SIM=ngspice

vvss vss 0 dc 0
vvcc vcc 0
+ pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n 0
**** interactive sim
.control
save all
tran 0.01n 30n
write test_inv_ngspice.raw
.endc
```

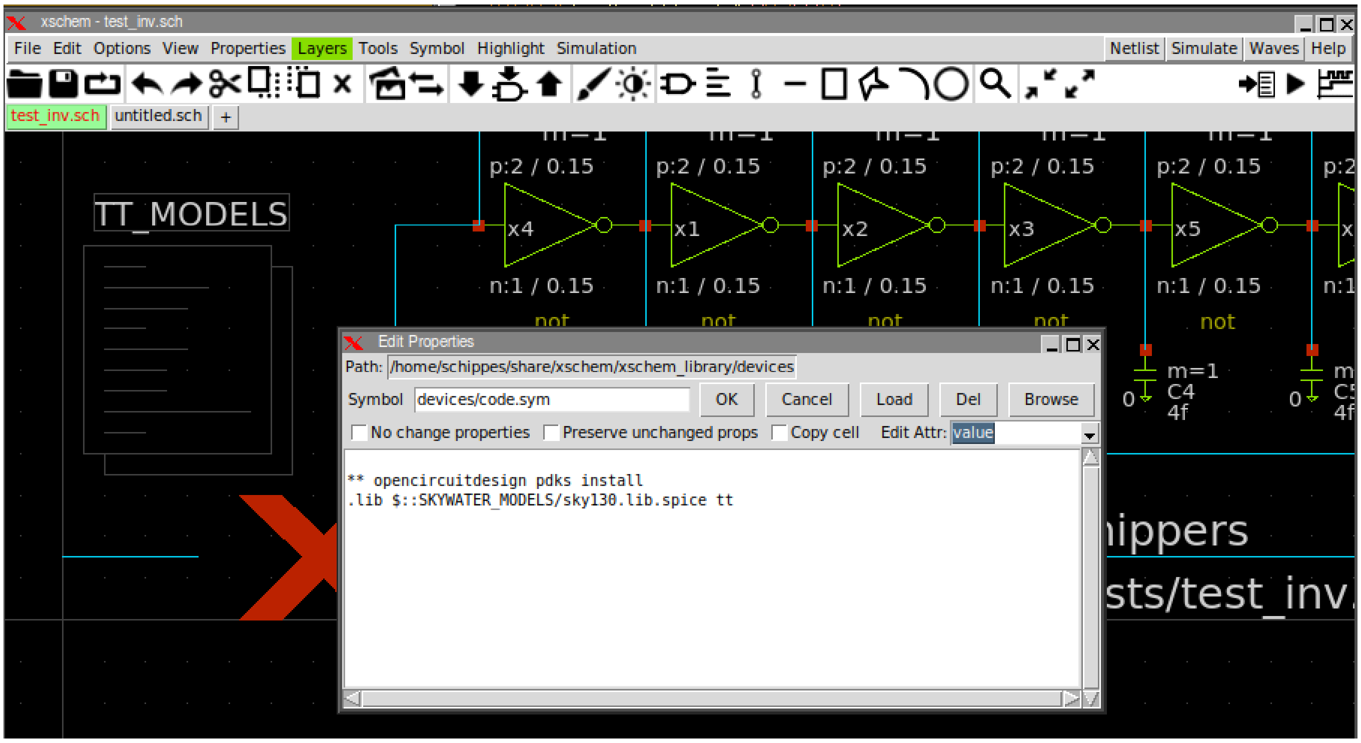
```
COMMANDS1
SIM=xyce

vvss vss 0 dc 0
vvcc vcc 0
+ pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n
.print tran format=raw file=test_inv_xy
+ v(*) i(*)
```

Path: /home/schippes/share/xschem/xschem_library/devices
Symbol devices/simulator_commands_showr
Edit Attr: value

```
vvss vss 0 dc 0
vvcc vcc 0
+ pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n 0
**** interactive sim
.control
save all
tran 0.01n 30n
write test_inv_ngspice.raw
.endc
```

Note in above commands a **`write test_inv_ngspice.raw`** command. This example runs simulation with both Xyce and ngspice so the output raw file is differentiated. If you just plan to use one simulator a good suggestion is to write a raw file with the same name as the circuit, so **`write test_inv.raw`**.

If you select the TT_MODELS component and press **`q`** you see the reference to the PDK top library SPICE file. The path is specified using TCL variables that have been generated by xschem when the pdk installation was looked up. This allows to have portable schematics, no absolute path is hardcoded in the schematic files.


xschem - test_inv.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_inv.sch untitled.sch +
TT_MODELS
p:2 / 0.15
n:1 / 0.15
x4 x1 x2 x3 x5
not
m=1 C4 4f
Edit Properties
No change properties Preserve unchanged props Copy cell Edit Attr: value
OK Cancel Load Del Browse

Path: /home/schippes/share/xschem/xschem_library/devices

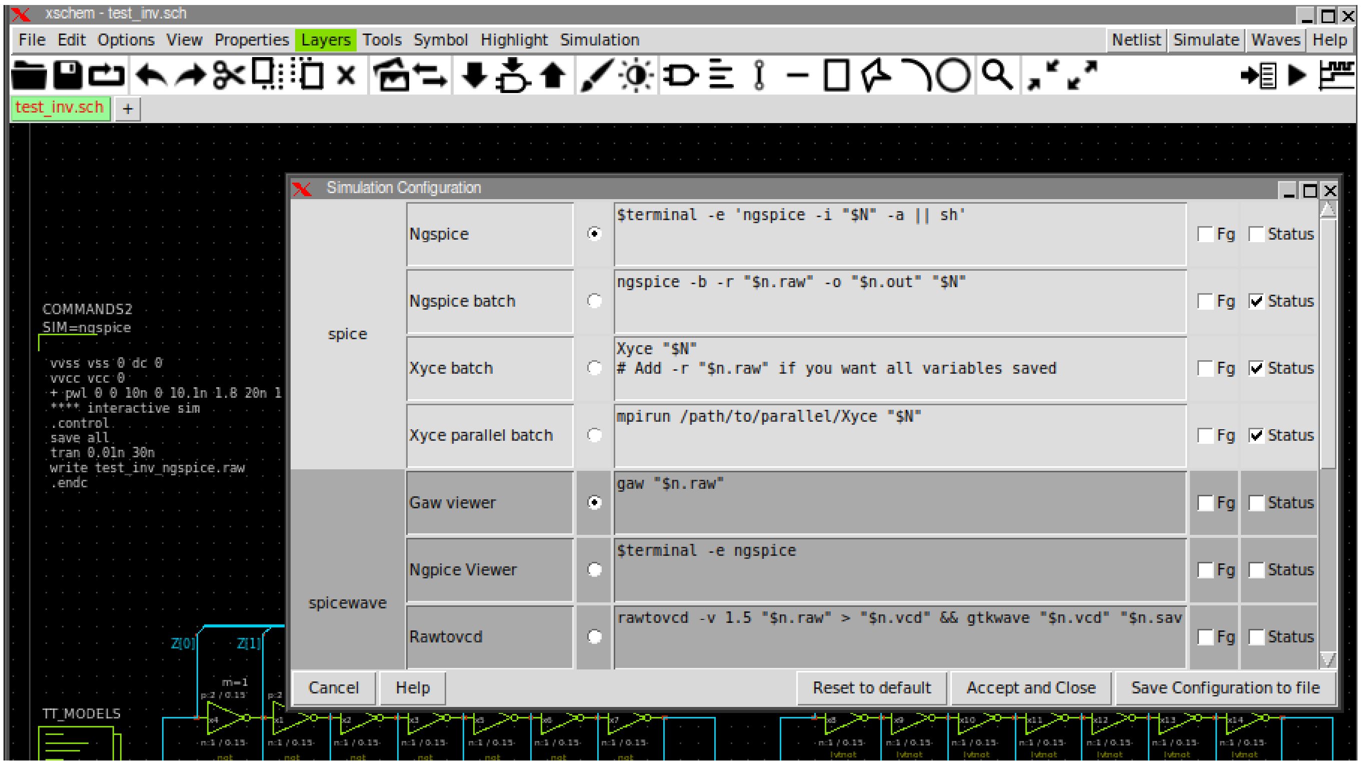
Symbol devices/code.sym

```
** opencircuitdesign pdks install
.lib $::SKYWATER_MODELS/sky130.lib.spice tt
```

If everything is set up correctly pressing the **`Netlist`** button or hitting the **`n`** key will produce a spice netlist of the circuit. The netlist location is by default set to your home directory: **`~/.xschem/simulations`**

```
schippes@mazinga:~/x/test_open_pdks$ ls -ltr ~/.xschem/simulations/
...
...
-rw-r--r-- 1 schippes schippes      3266 ott 18 15:26  test_inv.spice
```

You can then simulate the circuit. Select the simulator to use by clicking menu **`Simulation->Configure simulators and tools`** and selecting (for this example) **`ngspice`**



xschem - test_inv.sch

Simulation Configuration

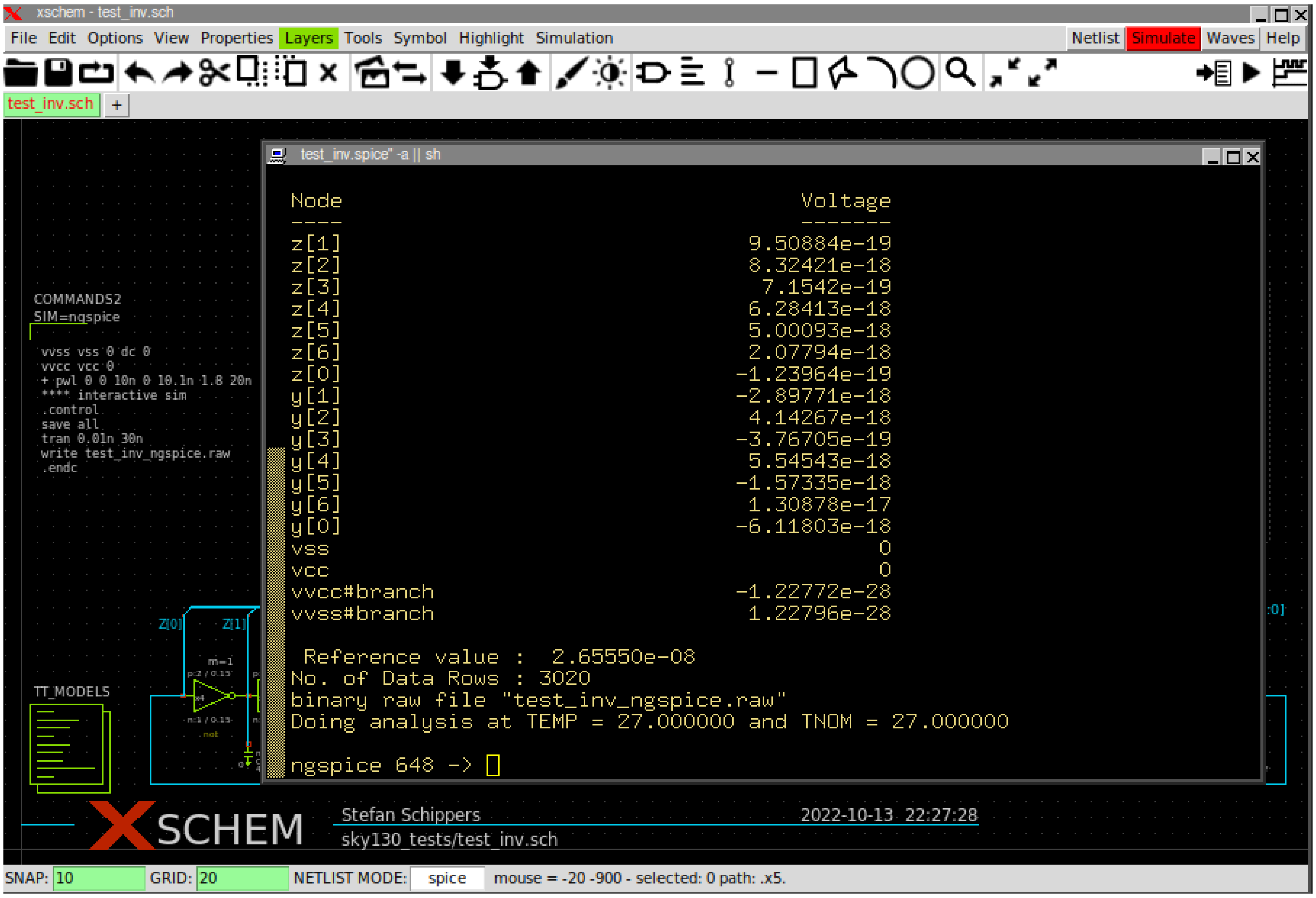
spice	Ngspice	●	`$terminal -e 'ngspice -i "$N" -a || sh'`	☐ Fg	☐ Status
	Ngspice batch	○	`ngspice -b -r "$n.raw" -o "$n.out" "$N"`	☐ Fg	✓ Status

	Xyce batch	○	`Xyce "$N"`
			`# Add -r "$n.raw" if you want all variables saved`
				☐ Fg	✓ Status

	Xyce parallel batch	○	`mpirun /path/to/parallel/Xyce "$N"`	☐ Fg	✓ Status
spicewave	Gaw viewer	●	`gaw "$n.raw"`	☐ Fg	☐ Status
	Ngpice Viewer	○	`$terminal -e ngspice`	☐ Fg	☐ Status
	Rawtovcd	○	`rawtovcd -v 1.5 "$n.raw" > "$n.vcd" && gtkwave "$n.vcd" "$n.sav`	☐ Fg	☐ Status

```
COMMANDS2
SIM=ngspice

 vvss vss 0 dc 0
 vvcc vcc 0
 + pwl 0 0 10n 0 10.1n 1.8 20n 1
 **** interactive sim
 .control
 save all
 tran 0.01n 30n
 write test_inv_ngspice.raw
 .endc
```

Press the **`Simulation`** button and see the ngspice running in a terminal:



xschem - test_inv.sch

test_inv.spice" -a || sh

```
Node                                  Voltage
----                                  -------
z[1]                              9.50884e-19
z[2]                              8.32421e-18
z[3]                               7.1542e-19
z[4]                              6.28413e-18
z[5]                              5.00093e-18
z[6]                              2.07794e-18
z[0]                             -1.23964e-19
y[1]                             -2.89771e-18
y[2]                              4.14267e-18
y[3]                             -3.76705e-19
y[4]                              5.54543e-18
y[5]                             -1.57335e-18
y[6]                              1.30878e-17
y[0]                             -6.11803e-18
vss                                         0
vcc                                         0
vvcc#branch                      -1.22772e-28
vvss#branch                       1.22796e-28

 Reference value :  2.65550e-08
No. of Data Rows : 3020
binary raw file "test_inv_ngspice.raw"
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000

ngspice 648 ->
```

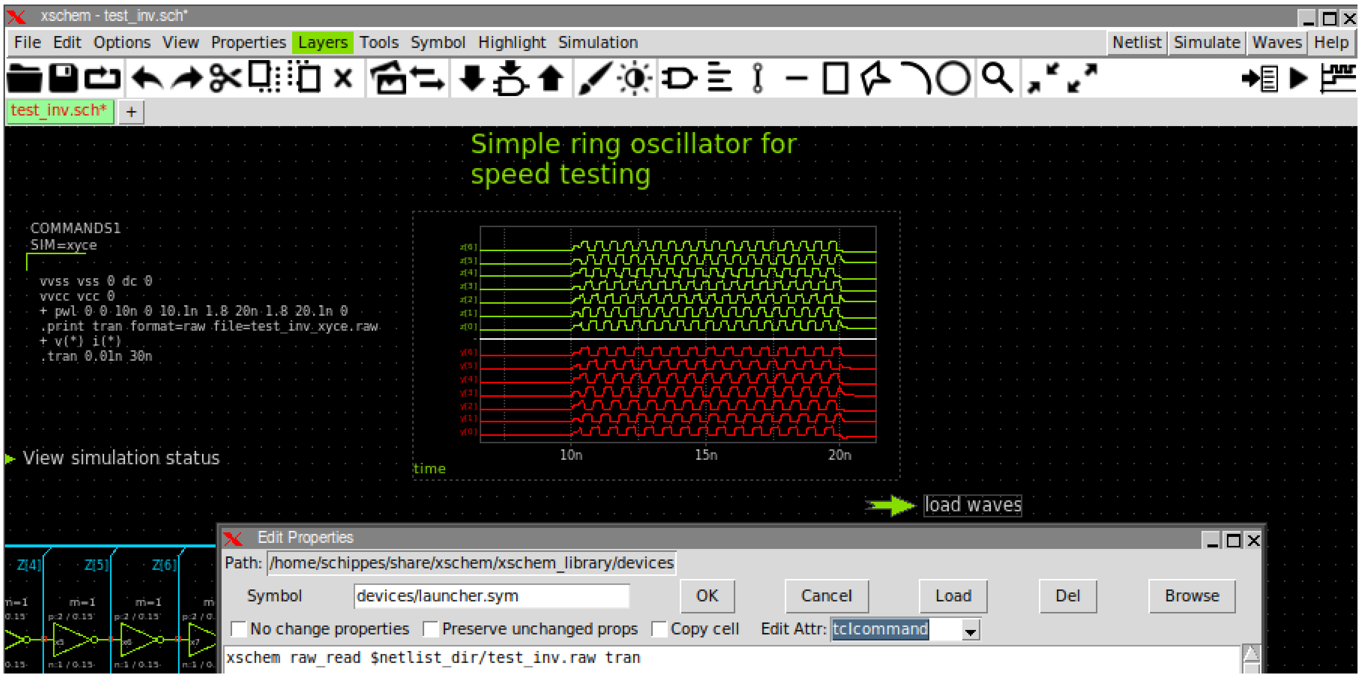
Stefan Schippers 2022-10-13 22:27:28
sky130_tests/test_inv.sch

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = -20 -900 - selected: 0 path: .x5.

The default terminal used by xschem to run the simulator is **`xterm`**. I strongly suggest you to install xterm (on ubuntu/debian Linux: **`sudo apt-get install xterm`**) since it is a very small package and is not a broken terminal like most Gnome/KDE/LXDE stuff. You can however use any terminal editor by specifying the one to use in your xschemrc. If not specified xschem defaults to **`xterm`**

```
## set terminal xterm
set terminal gnome-terminal
```

After completing simulation you can add into the schematic a graph (**`Simulation->Add waveform graph`**) and a waveform reload launcher (**`Simulation->Add waveform reload launcher`**). The launcher has a **`tclcommand`** attribute that loads the simulator data file (**`test_inv.raw`**) and specifies the type of analysis (**`op, dc, ac, tran`**)

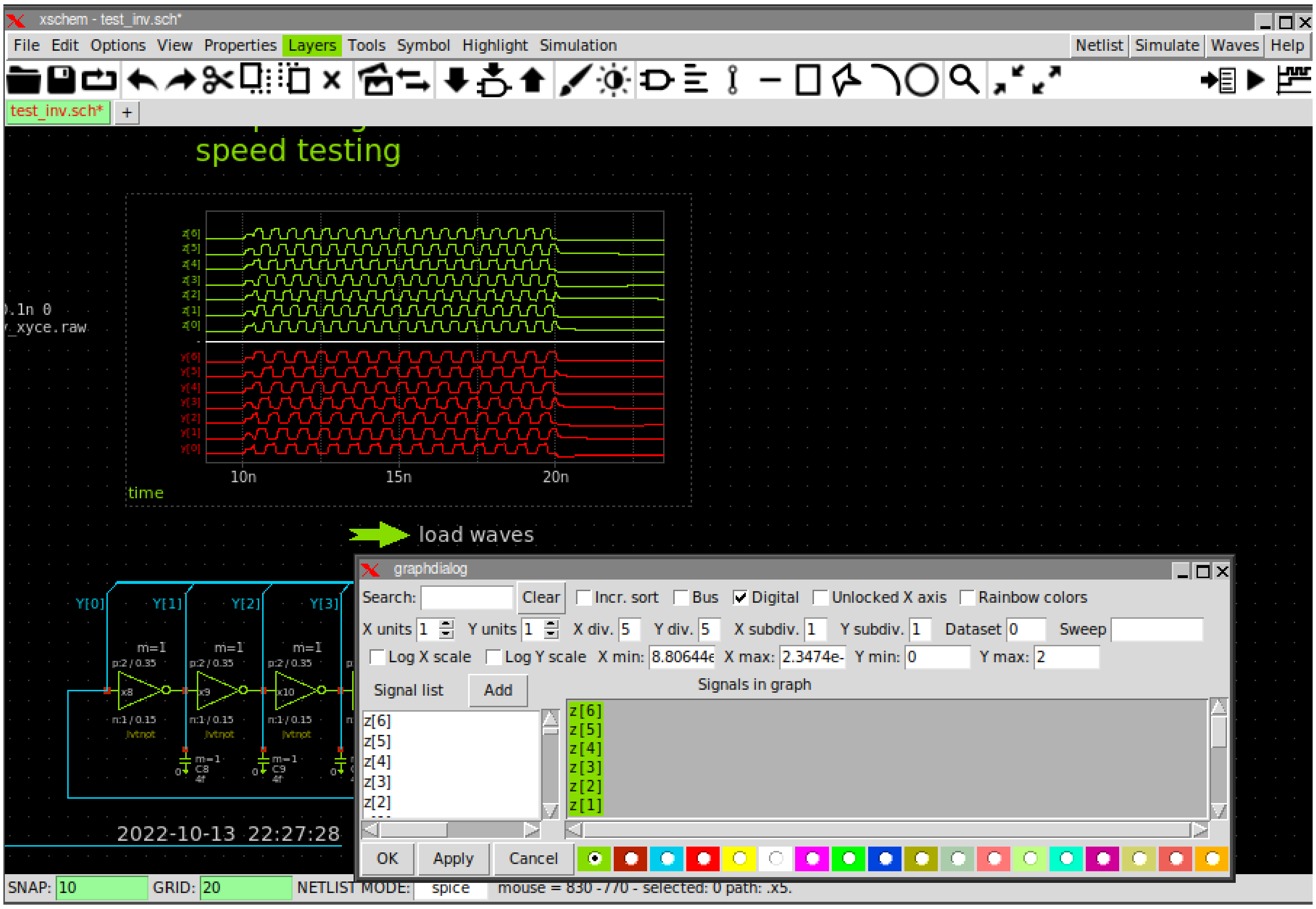

xschem - test_inv.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
test_inv.sch*
Simple ring oscillator for speed testing
View simulation status
time
10n
15n
20n
load waves
Edit Properties
OK
Cancel
Load
Del
Browse

```
COMMANDS1
SIM=xyce

 vvss vss 0 dc 0
 vvcc vcc 0
 + pwl 0 0 10n 0 10.1n 1.8 20n 1.8 20.1n 0
 .print tran format=raw file=test_inv_xyce.raw
 + v(*) i(*)
 .tran 0.01n 30n
```

Path: /home/schippes/share/xschem/xschem_library/devices
Symbol: devices/launcher.sym
No change properties ☐ Preserve unchanged props ☐ Copy cell ☐ Edit Attr: tclcommand

```
xschem raw_read $netlist_dir/test_inv.raw tran
```

See the manual for details



Search: Clear Incr. sort Bus [✓ Digital] Unlocked X axis Rainbow colors

X units 1 Y units 1 X div. 5 Y div. 5 X subdiv. 1 Y subdiv. 1 Dataset 0 Sweep

Log X scale Log Y scale X min: 8.80644e X max: 2.3474e- Y min: 0 Y max: 2

Signal list: z[6] z[5] z[4] z[3] z[2]

Signals in graph: z[6] z[5] z[4] z[3] z[2] z[1]

SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 830 -770 - selected: 0 path: .x5.

UP

FAQ

I want new instances to get assigned a new unique name automatically.

Add this to your xschemrc file:

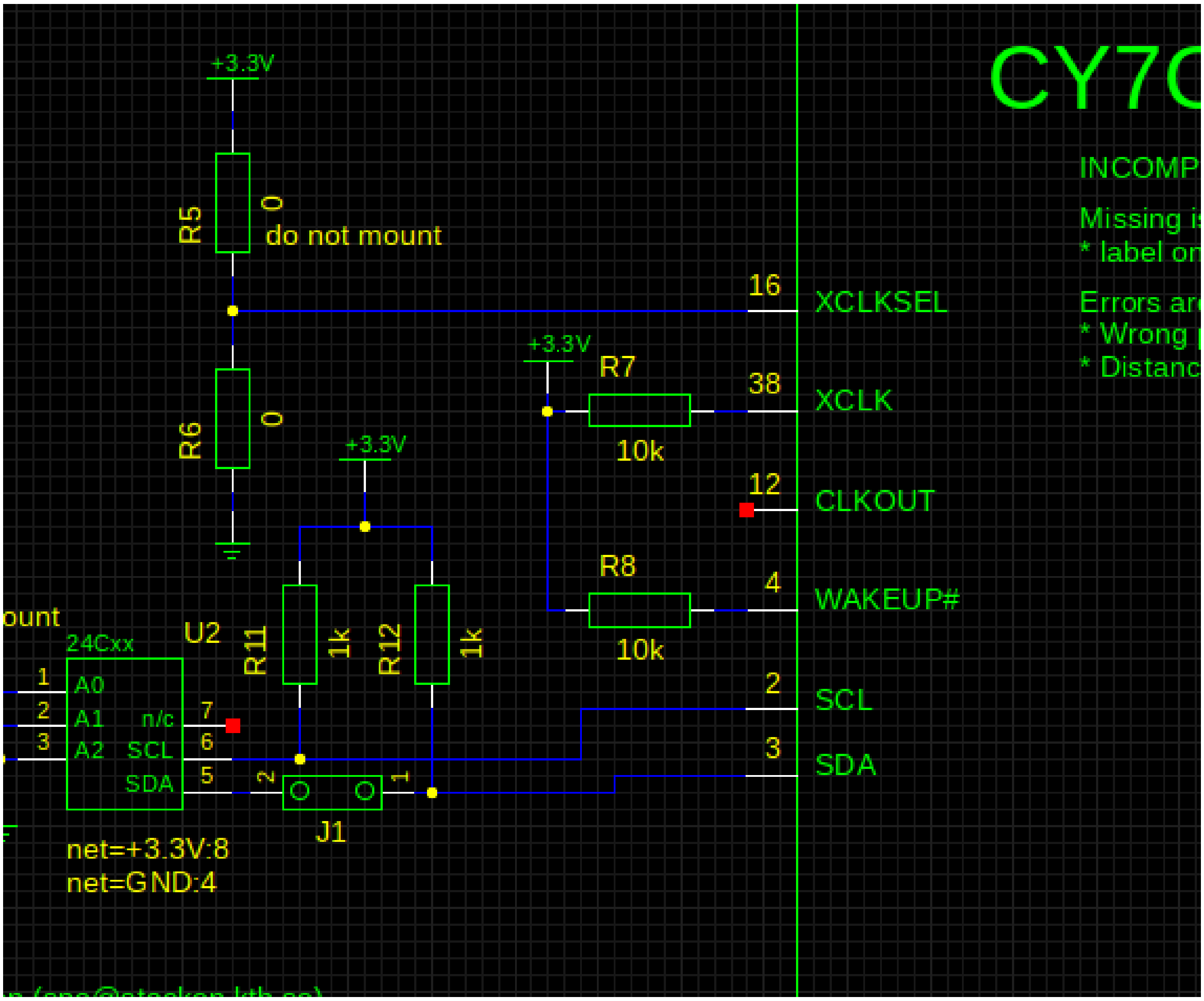
```
      set disable_unique_names 0
```

By default XSCHEM allows instance name (Refdes) duplicates in the schematic. This must be resolved by the user normally, before exporting any netlist. The **`Hilight - Highlight duplicate instance names`** (**`k`** key) menu entry can be used to mark the components that need to be renamed. The **`Highlight - Rename duplicate instance names`** menu entry can be used to automatically rename the last added components so that they have an unique name. Using the above mentioned xschemrc option will automatically rename any added refdes that clashes with existing names.

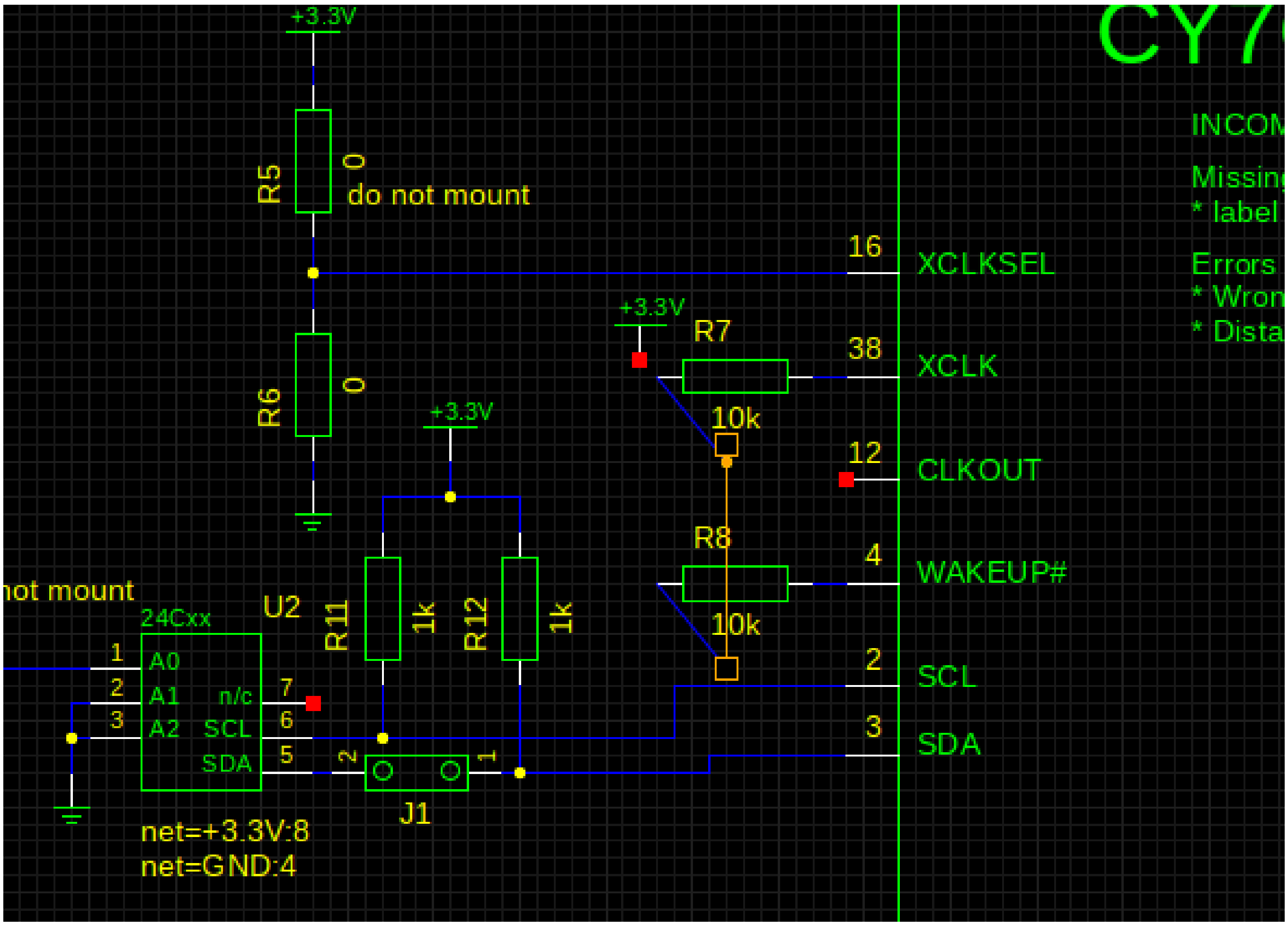
Why do i have to press 'm' to move a component instead of just click and drag?

XSCHEM is intended to handle very big schematics, mouse drags are used to select a rectangular portion of the circuit to move / stretch, if a mouse click + drag moves components it would be very easy to move things instead of selecting things. This happens with geda-gschem for example:

Why do i have to press 'm' to move a component instead of just click and drag?



Here i want to select the R7 and R8 resistors, so i place the mouse close to the upper-left R7 boundary and start dragging, but since clicking also selects nearby objects the wire gets selected and moving the mouse will move the wire.



This behavior is considered not acceptable so clicking and dragging will never modify the circuit. Pressing 'm' (for move) or 'c' (for copy) makes the behavior more predictable and safer. A new user just needs to get used to it.

I start xschem in the background and it freezes. Why?

XSCHEM is usually launched from a terminal, the terminal becomes a TCL shell where commands can be sent to xschem. For this reason XSCHEM should not be launched in background, as any I/O operation to/from the terminal will block the program. If you don't plan to use the terminal just start XSCHEM with the -b option: **`xschem -b &`**. XSCHEM will fork itself in the background detaching from the terminal.

Using Xschem (also for skywater-pdk users): a checklist in case of problems:

- Xschem by itself (as well as ngspice and open_pdks) does not require a docker container if you build from sources.
- The whole skywater pdk is in rapid evolution, and so is xschem. Do not use packaged versions of xschem provided by linux distributions, the xschem version provided is far too old. Same consideration for ngspice. Please build xschem from sources by cloning from git: git clone git@github.com:StefanSchippers/xschem.git xschem-src, then running ./configure with optional --prefix parameter, see instructions here. In particular please verify you have all the required packages installed. refer to the install page in the xschem manual.
- To install xschem and ngspice follow this video, but DO NOT follow this video for skywater spice models installation, there is a second video for this, the default and highly recommended procedure is to install open_pdks.
- After installing open_pdks you can run simulations by including the top skywater model file: .lib /your/path/to/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt.
- The recommended way to design and simulate a circuit is to create a new empty directory and copy the open_pdks provided xschemrc: mkdir my_example ; cp /your/path/to/share/pdk/sky130A/libs.tech/xschem/xschemrc

my_example/, then cd into that directory and start xschem.

- Xschem writes netlists in a directory defined by the tcl 'netlist_dir' variable. You can change the location by editing the xschemrc file (locate the 'set netlist_dir' line and change according to your needs). By default the netlist directory is set to ~/.xschem/simulations. Always verify you have write permissions in the directory you are using for netlist generation. The spice simulator will be invoked by xschem and will also be running in this directory, so all spice generated files will also be in this directory.
- Xschem uses a terminal and an editor to allow editing some files or displaying some content. For this there are two variables defined in xschemrc: editor and terminal. By default editor is set to 'gvim -f' and terminal is set to 'xterm'. I suggest to install xterm on your system, it is a very small package and has much less problems than 'modern' terminal emulators, and verify 'editor' is set to an existing editor installed on the system. Please note that for gvim a -f option is added to avoid gvim forking in the backgound. If your editor of choice forks itself in the background please provide an option do avoid doing so. Xschem needs for the editor sub-process to finish before going forward.
- Xschem is able to produce Spice, Verilog and VHDL netlists, the default open source tools for simulating these are by default ngspice, icarus verilog and ghdl respectively. If you plan to simulate verilog designs in addition to spice, please install icarus verilog (i recommend building from git, git clone git://github.com/steveicarus/iverilog.git verilog-src), for VHDL simulations install ghdl from git, git clone https://github.com/ghdl/ghdl.git ghdl-src. xschem can invoke these simulator by pressing the 'Simulate' button, this works if the paths for the simulators are correctly configured. To verify the configuration go to xschem Simulation menu and click 'Configure simulators and tools'. A dialog box appears with the various command lines xschem uses to invoke the simulator. There is a 'Help' button giving more information. The Configure simulators and tools dialog box can be used to invoke different simulators, even commercial tools. Xschem has been used with HSPICE, cadence NCSIM digital simulator and Mentor Modelsim.
- For ngspice specific issues please read the manual! it has lot of very useful information.
- Please note that skywater-pdk has a .option scale=1.0u in the spice files, that means that all transistor dimensions you give (L=0.18, W=2) will be scaled down by 1e6. so a '1' means 1 micro-meter. DO not use l=0.18u, since that will reduce to 0.18 pico-meters!!

XSCHEM GRAPHICS PERFORMANCE CONSIDERATIONS

For 2D graphic workloads software rendering on framebuffer device is the fastest option on almost all laptops i have tested xschem on. The only exception is the little Samsung N220 due to the slow and tiny Atom N450 CPU.
Framebuffer graphics is also the most precise and reliable since the X11 specification has exact rules for pixelization/rasterization of primitives.
All drawings are thus exactly identical down to the pixels on any machine using fbdev.
Following table summarizes the test times for a xschem BIST routine, doing lot of graphic (among other) operations.

```
HOST                CPU                      GPU                          X11 Driver    Test time
=================================================================================================
Asus F556U          Intel core i7-7500U      Intel HD 620                 fbdev         19.8 -bes
Asus F556U          Intel core i7-7500U      Intel HD 620                 modesetting   28.7
Asus F556U          Intel core i7-7500U      Intel HD 620                 intel         20.6

Samsung N220        Intel Atom N450          Integrated N450              fbdev         96.5
Samsung N220        Intel Atom N450          Integrated N450              modesetting   N/A
Samsung N220        Intel Atom N450          Integrated N450              intel         95.0 -bes

Samsung R540        Intel Core i3 M 380      AMD/ATI Radeon HD 54xx       fbdev         31.9 -bes
Samsung R540        Intel Core i3 M 380      AMD/ATI Radeon HD 54xx       modesetting   64.3
Samsung R540        Intel Core i3 M 380      AMD/ATI Radeon HD 54xx       radeon        32.3

HP Pavilion DV6000  Intel Core Duo T5450     NVIDIA GeForce 8400M GS      fbdev         41.5 -bes
HP Pavilion DV6000  Intel Core Duo T5450     NVIDIA GeForce 8400M GS      modesetting  216.1
HP Pavilion DV6000  Intel Core Duo T5450     NVIDIA GeForce 8400M GS      nouveau       41.6
```

Following video shows the testing in action (OS: Devuan/testing, Arch: amd64):

Your browser does not support the video tag.