PREV UP NEXT

# Xschem slides [PDF version]

# [Video] Xschem FSiC2022 presentation

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

# INDEX



# TUTORIALS

- [Video] Editing component attributes
- [Video] Copying objects across xschem windows
- [Video] Symbols with inherited connections
- [Video] Search / replace function
- [Video] How to stretch objects
- [Video] Parameters in subcircuits
- [Video] Create pins from net labels, fix grid align issues, wires
- [Video] Link documentation to components/symbols
- [Video] Use rawtovcd to show ngspice waveforms in gtkwave
- [Video] Run a Verilog simulation with XSCHEM and icarus Verilog
- [Video] See logic propagation of nets live in xschem without using a backend simulator
- [Video] View Ngspice/Xyce simulation data inside XSCHEM
- [Video] Probe xschem nets into the GAW waveform viewer
- [Video] Probe xschem nets into the BESPICE waveform viewer
- [Video] Creating a symbol
- [Video] Instantiating schematics instead of symbols (LCC, Local Custom Cell)
- [Video] Using more schematic views of a symbol to do simulation at different abstraction levels
- [Video] Let components display the name of the net attached to their pins

# FAQ

- Common questions about XSCHEM

PREV UP NEXT

# WHAT IS XSCHEM

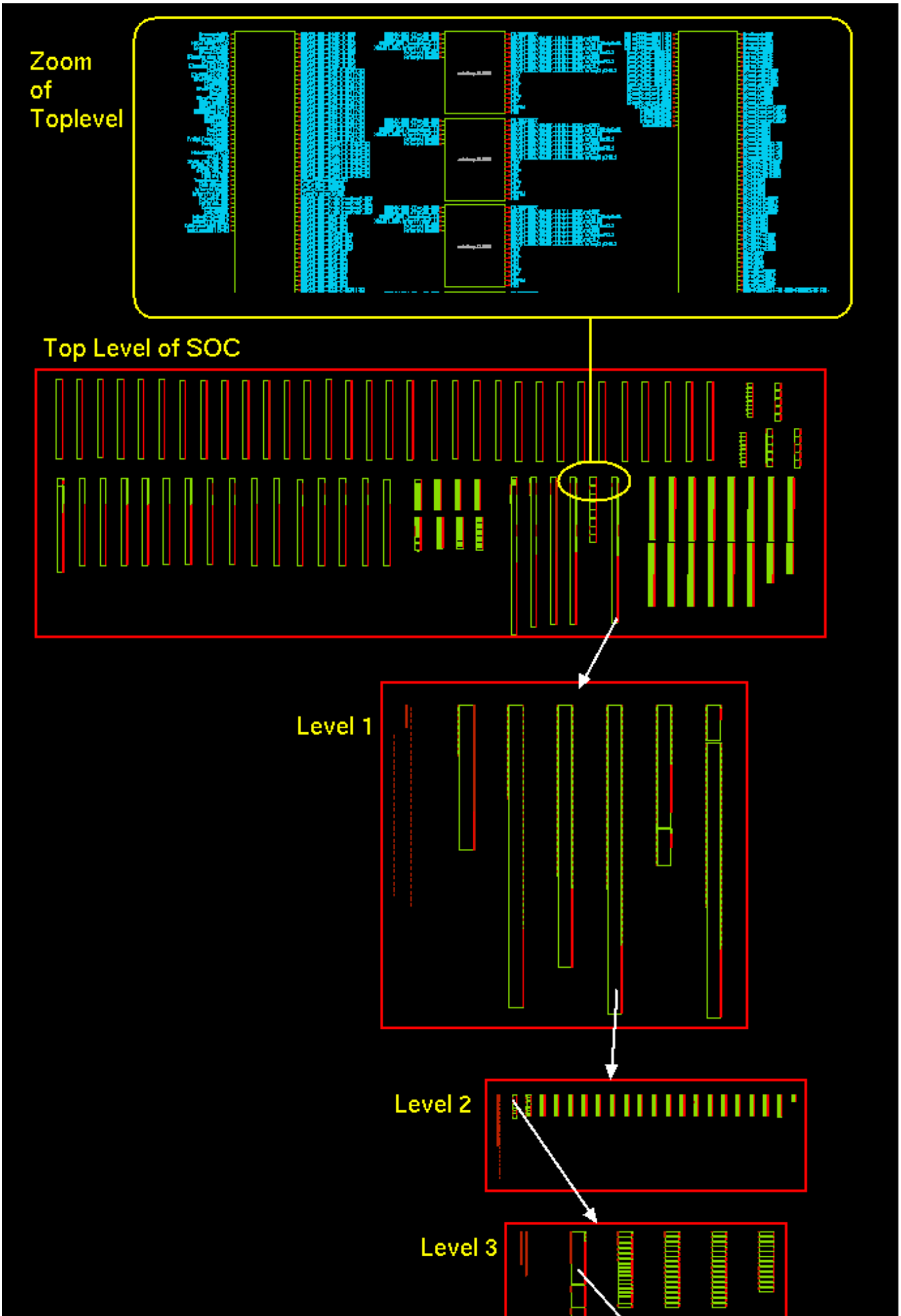
Electronic systems today tend to be generally very complex and a lot of work has to be done from circuit conception to the validation of the final product. One of the milestones of this process is the creation of the circuit schematic of the electronic system.

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.

PREV UP NEXT

# INSTALL XSCHEM

in order to install the program run the following command:

```
user:~$ cd xschem-<version>; ./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.

```
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
          gawk or mawk
```

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:~$ 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
```

DESTDIR is supported.

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
```

# -Technical information - Detailed XSCHEM startup sequence

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

1. If **--rcfile=<rcfile>** is given then source the specified rcfile. Do not load any other rcfile.
2. 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**.
3. Else use compile-time (generated from configure script) provided **XSCHEM_SHAREDIR**.
4. Source system-wide xschemrc if existing: **XSCHEM_SHAREDIR/xschemrc**
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 2.
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).

PREV UP NEXT

# RUN XSCHEM

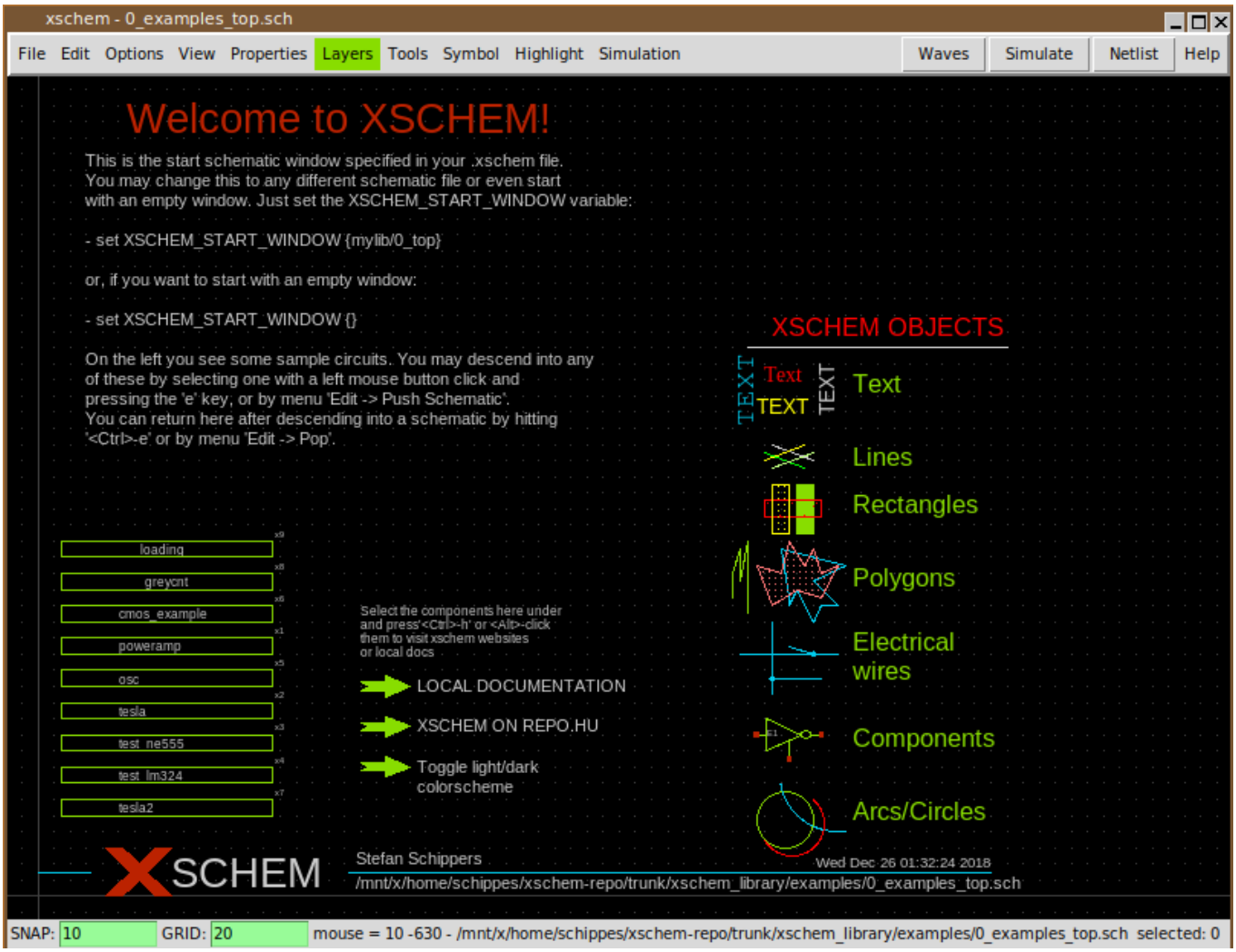
Assuming **xschem** is installed in one of the **${PATH}** search paths just execute:

```
user:~$ xschem
```

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


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

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




## XSCHEM COMMAND LINE OPTIONS

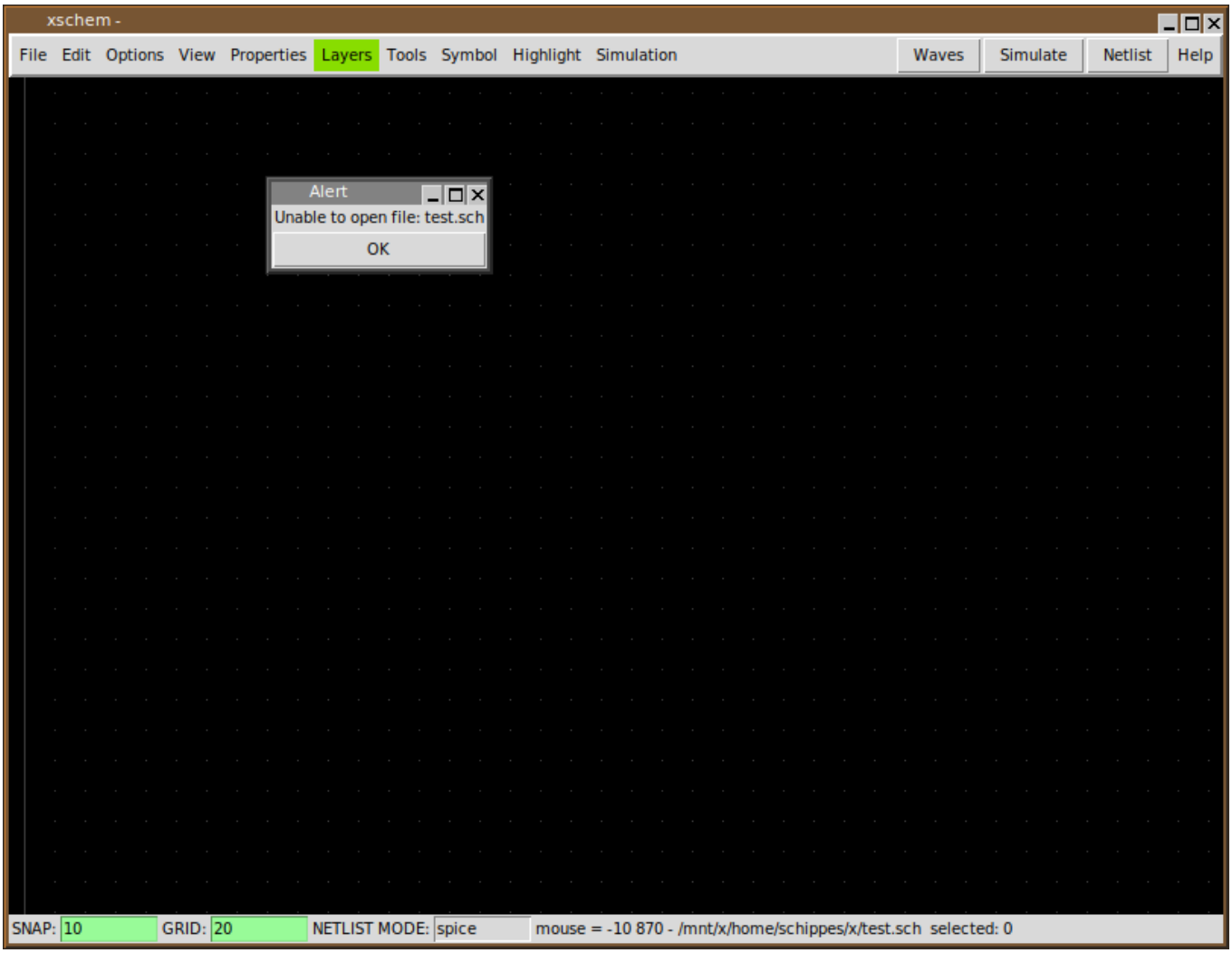
xschem accepts short (-h) or long (--help) options:

```
usage: xschem [options] [schematic | symbol ]
Options:
  -h  --help            Print this help.
  -b  --detach          Detach Xschem from console and fork in the background.
  -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 schematc file
                        (spice/Verilog/VHDL, depending on the netlist
                        type chosen).
  -w  --verilog         Set netlist type to Verilog.
  --tcl <tcl_script>    Execute specified tcl instructions before any other action,
                        this can be used to change xschemrc variables.
  --command <tcl_cmd>   Execute specified tcl commands after completing startup.
  --script <file>       Execute specified tcl file as a command script (perhaps with xschem  comman
  --tcp_port <number>   Listen to specified tcp port for client connections. (number >=1024).
  -i  --no_rcload       Do not load any xschemrc file.
  -o <file>             Set output path for netlist.
  --netlist_path <file>
  -N <file>             Set name (only name, not path) of top level netlist file.
  --netlist_filename <file>
```

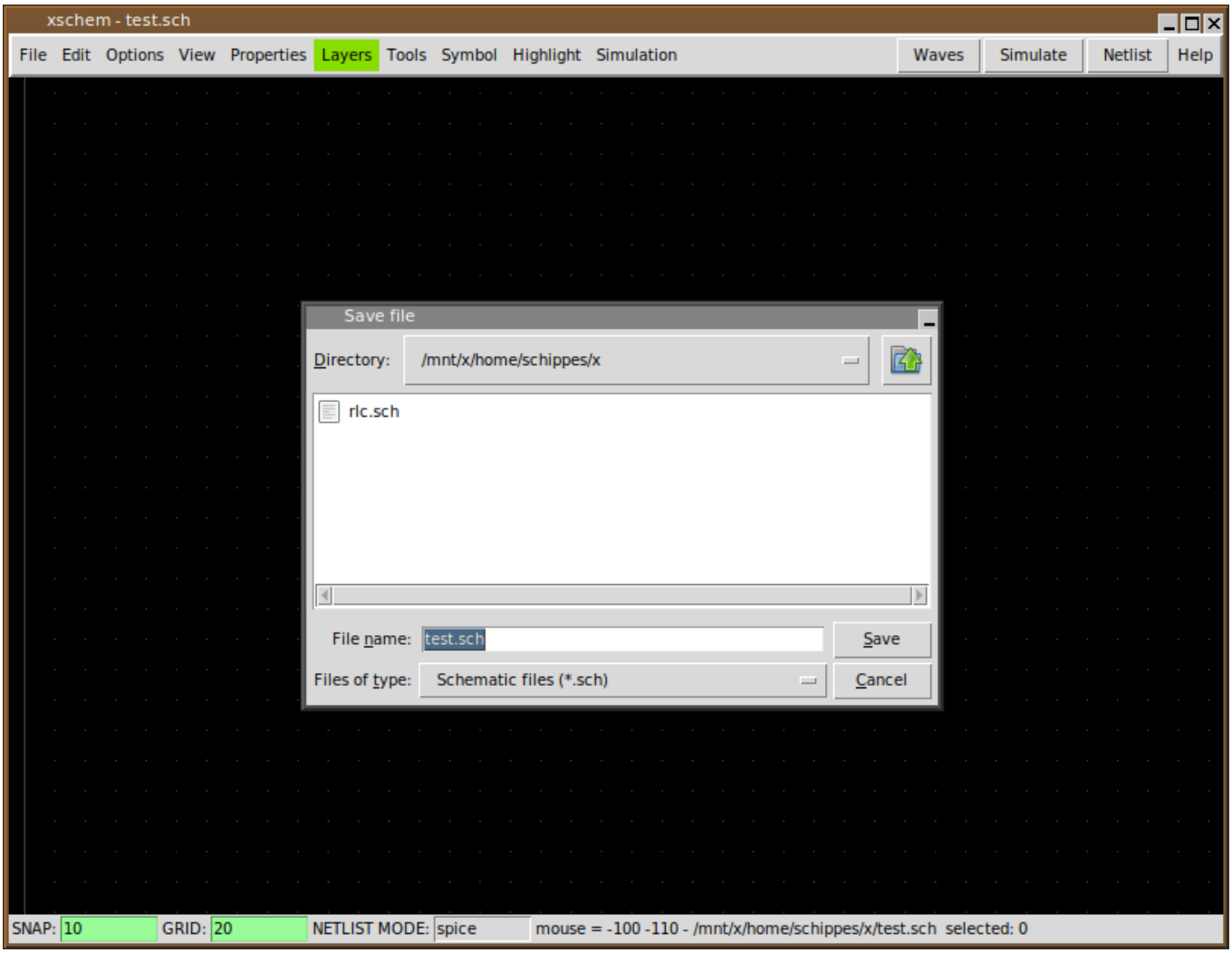
```
  -t  --tedax          Set netlist type to tEDAx.
  -s  --spice          Set netlist type to SPICE.
  -y  --symbol         Set netlist type to SYMBOL (used when drawing symbols)
  -x  --no_x           Don't use X (only command mode).
  -z  --rainbow        Use a raibow-looking layer color table.
  -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 if stdin and stdout are to be redirected
                       for example to /dev/null).
  -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`**




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




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
```

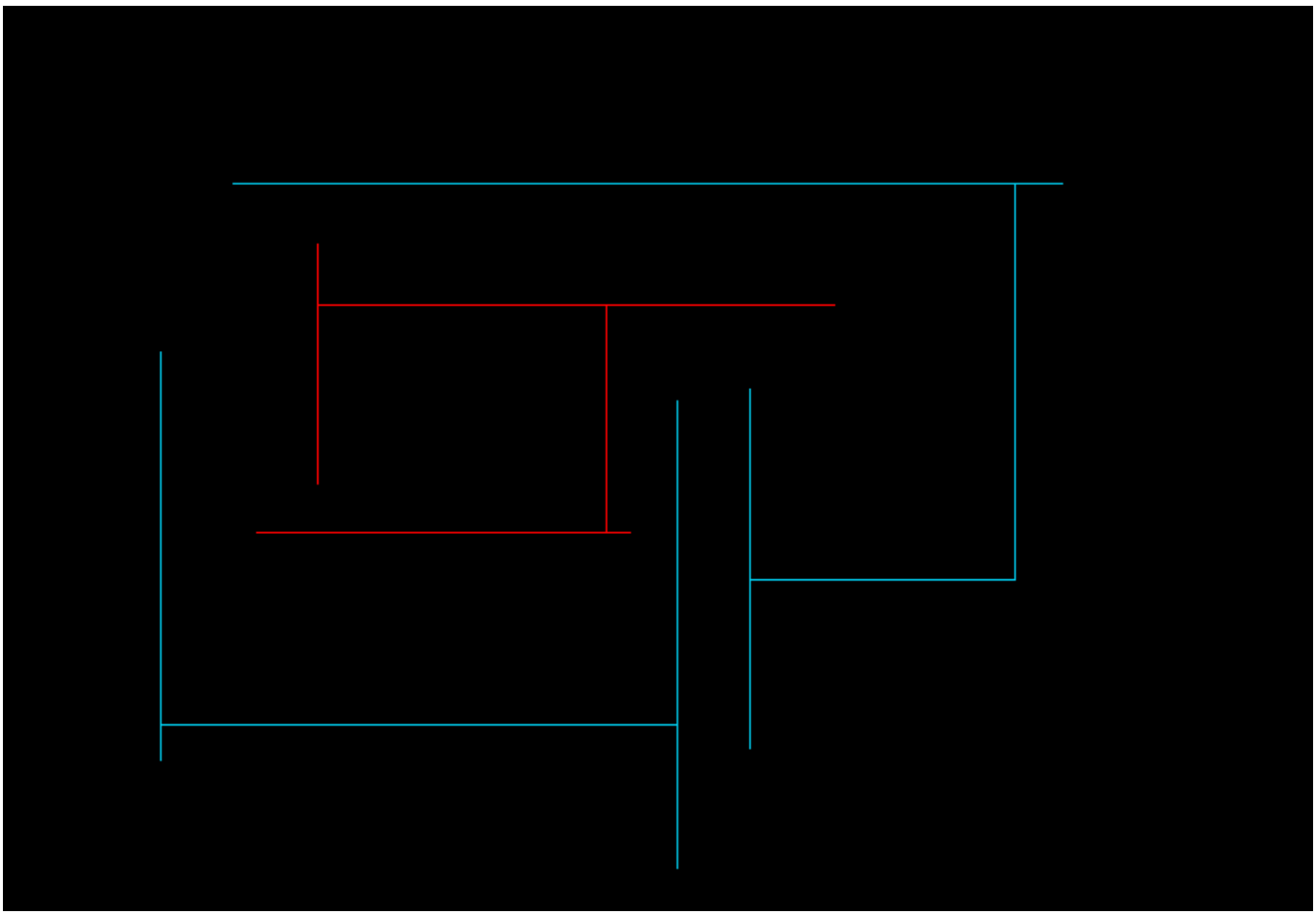
you can load **`test.sch`** when xschem is running by using the load command **`'<ctrl>o'`** key or by menu **`Open`** command. Use the file selector dialog to locate the schematic and load it in. When loading a new file XSCHEM asks to save the currently loaded schematic if it has been modified.

PREV UP NEXT

# XSCHEM ELEMENTS

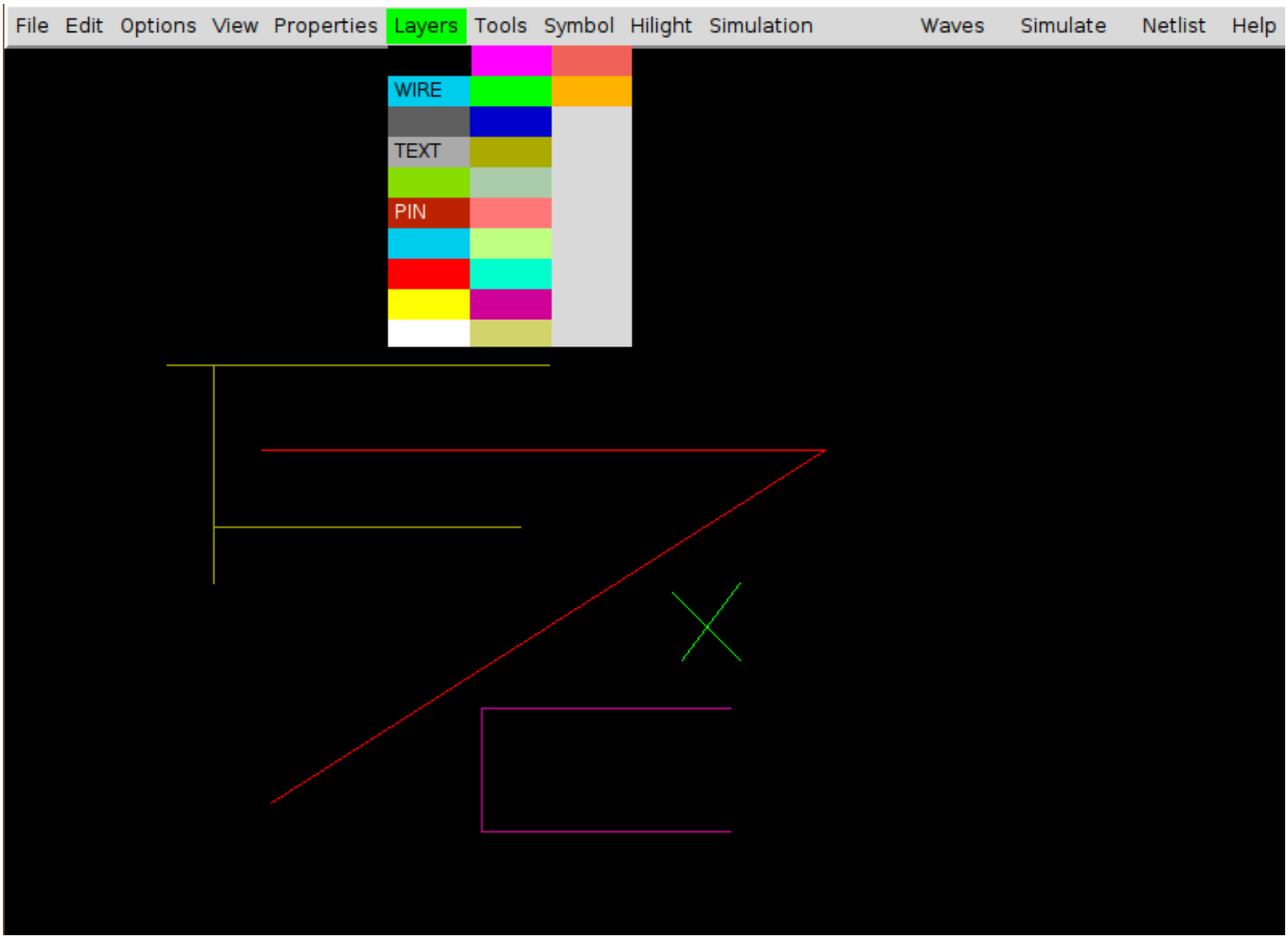
## WIRES

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).



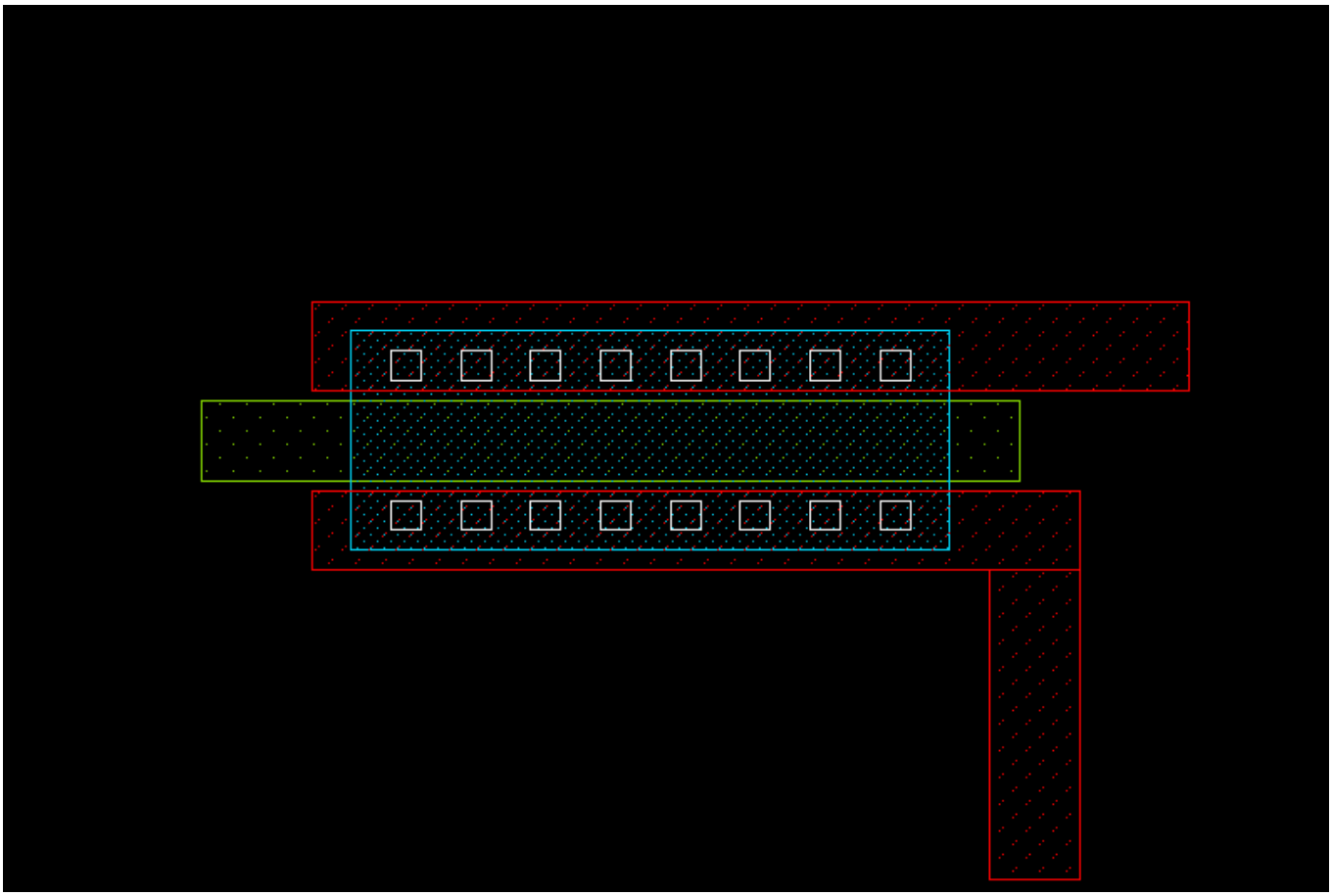
## LINES

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

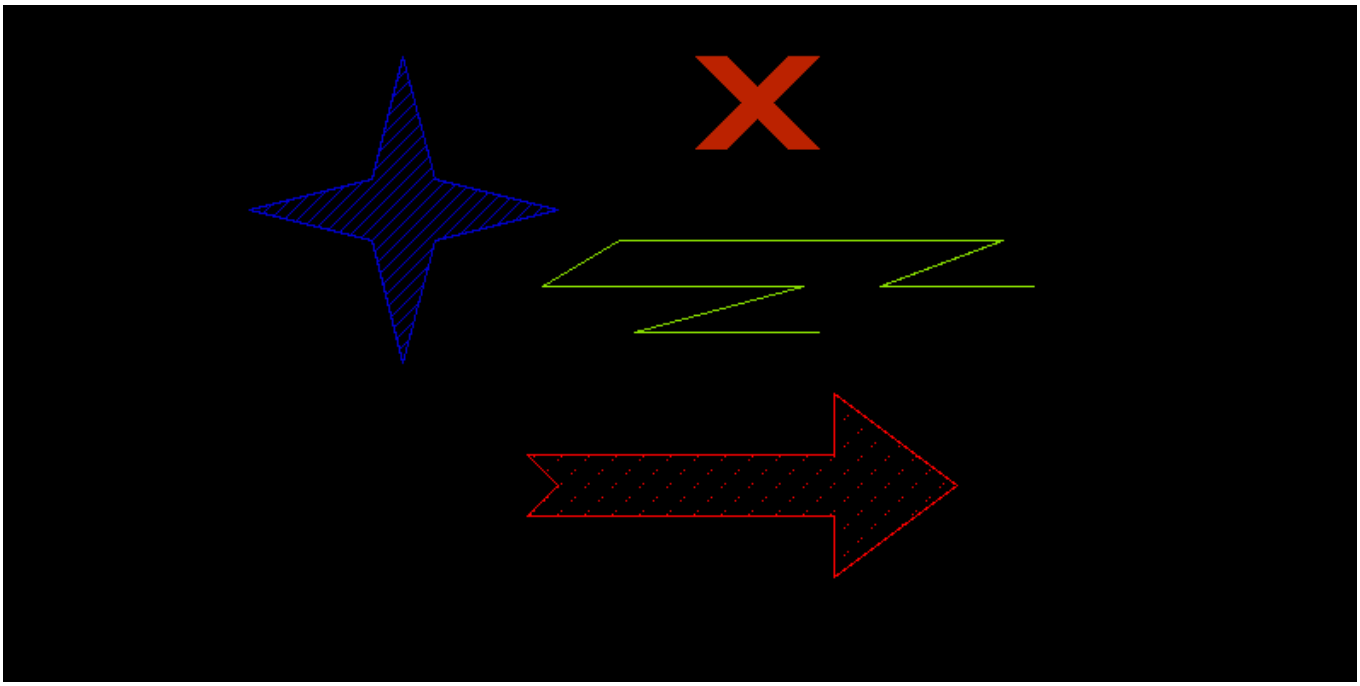


## POLYGONS

Polygons are paths that can be drawn on any layer. Placements begins with the **'ctrl-w'** 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).

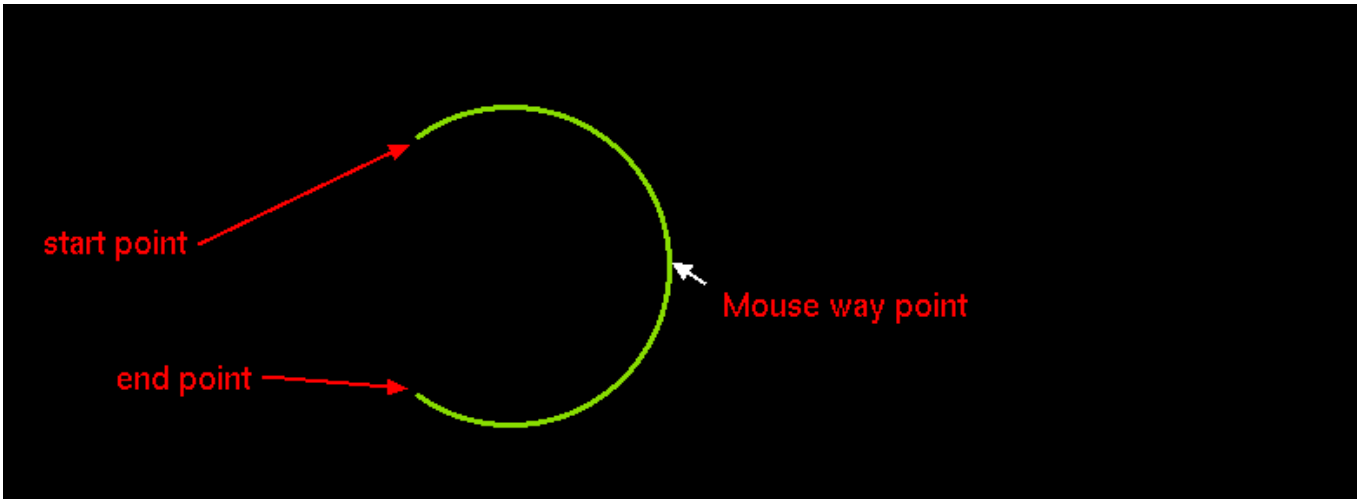
A **fill=true** attribute may be given to have the shape filled with the layer fill style.



## CIRCLES / ARCS

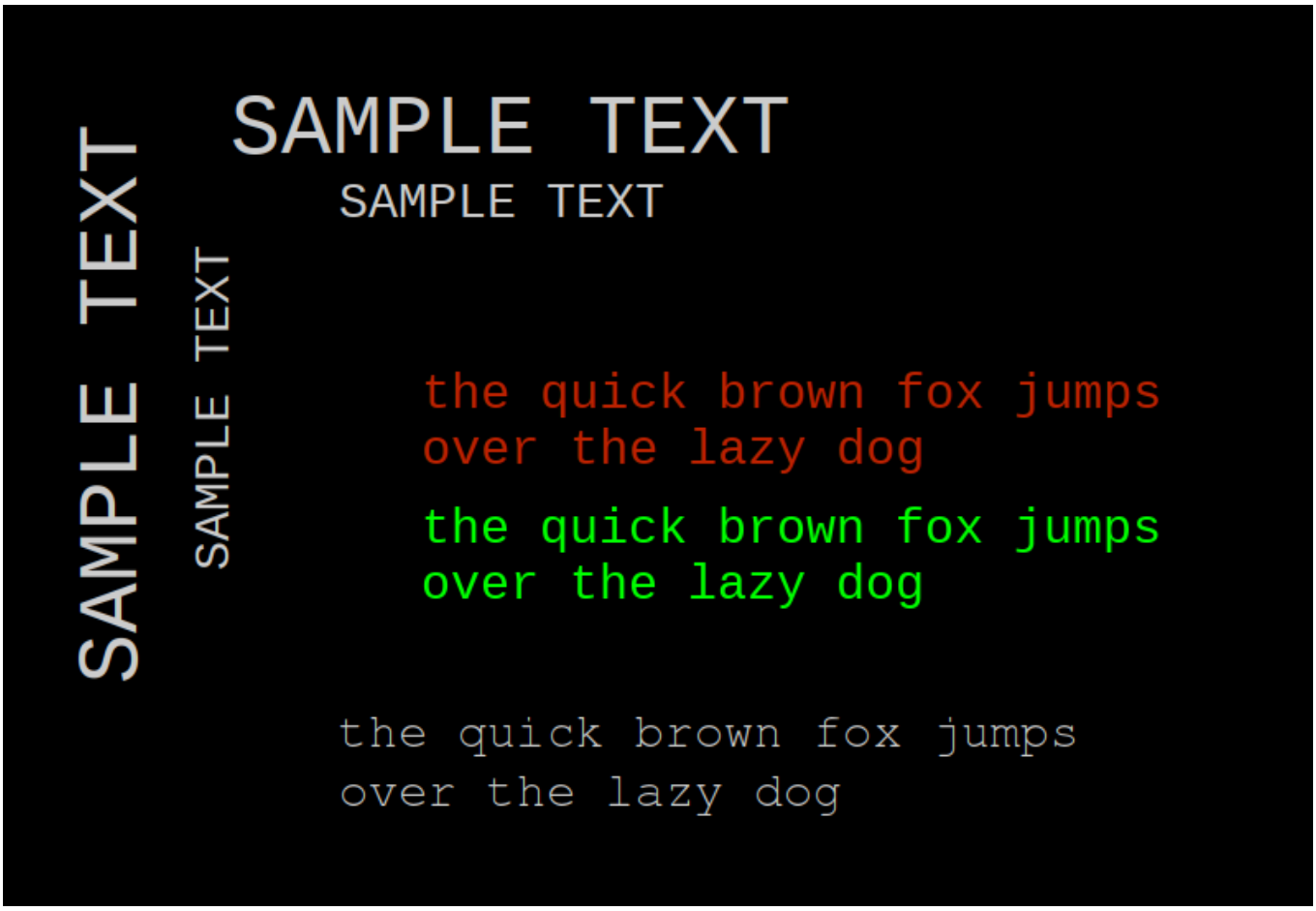
Arcs may be placed by hitting the **`Shift-C`** key. First click the start point, then the end point. Moving the mouse will show the arc passing thru the 2 points and the mouse waypoint. Clicking will place the arc. Arcs may be modified after creation by selecting in stretch mode ( **`Ctrl-Button1-drag`** ) 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.

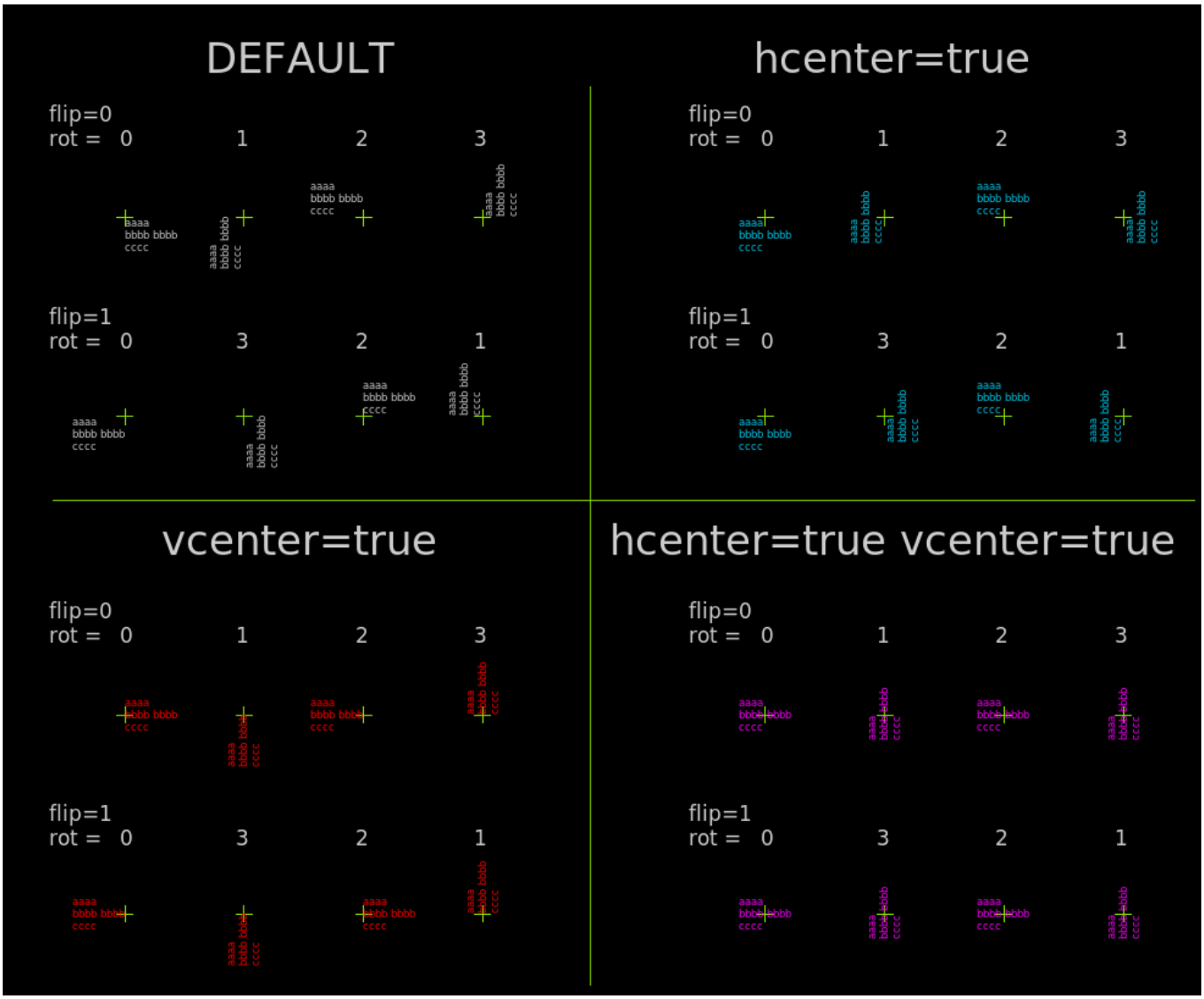


## TEXT

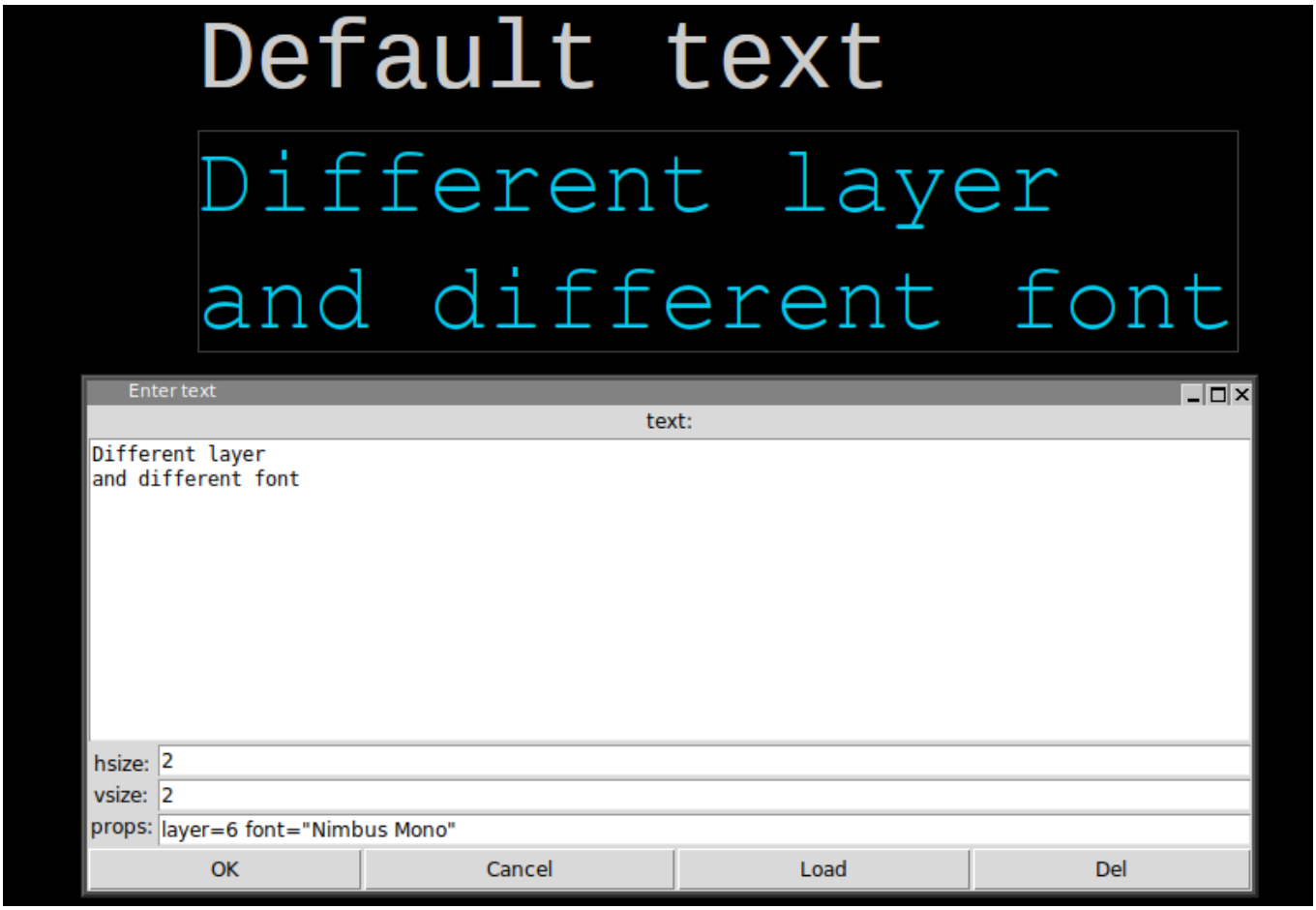
Text can be placed with the **'t'** bindkey. A dialog box appears where the user inputs the text and text size.




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 when the symbol is displayed as a component in a schematic.

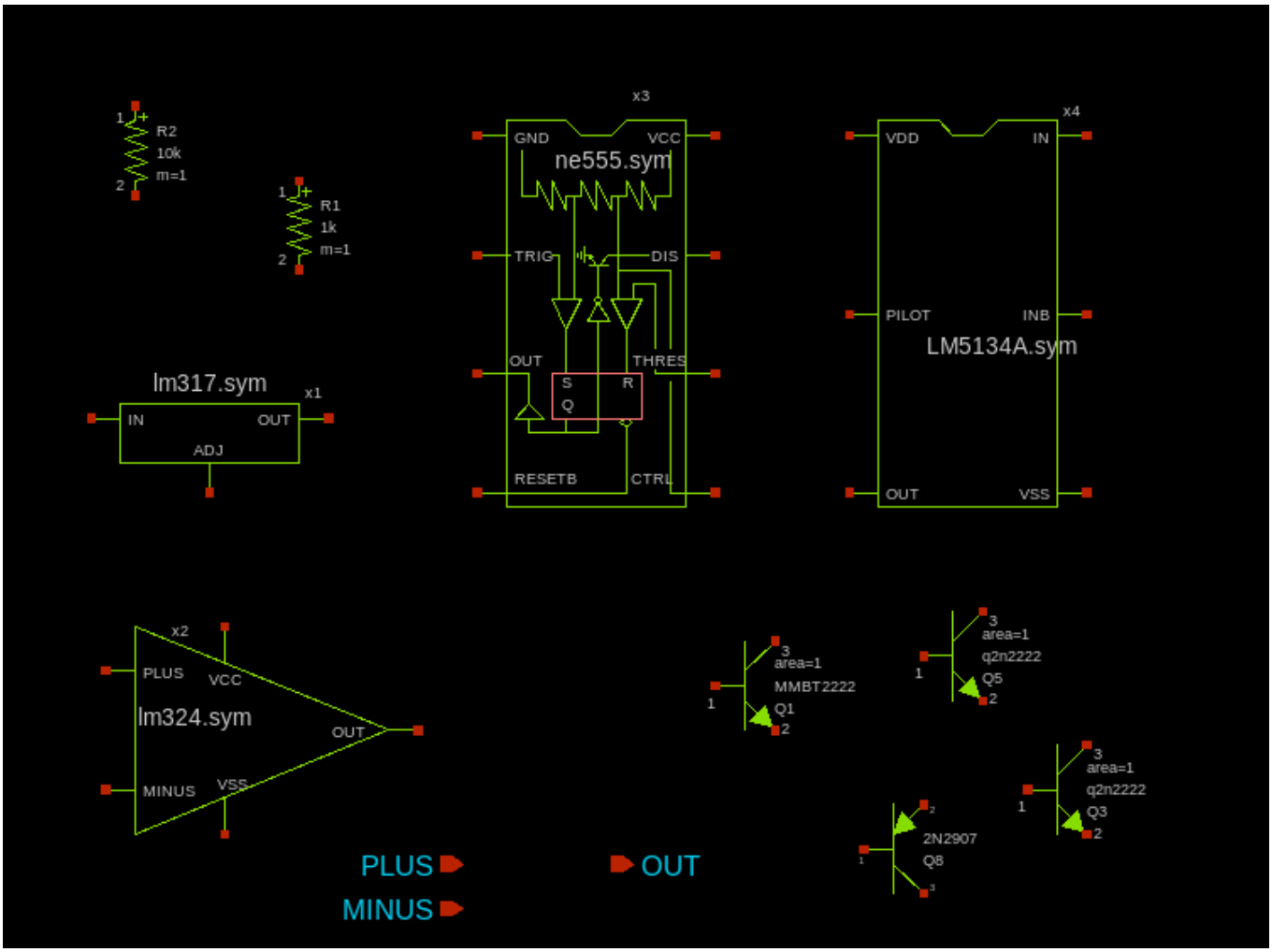



You wil learn in the [xschem properties chapter] how to set, edit and change object properties.




## 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
```

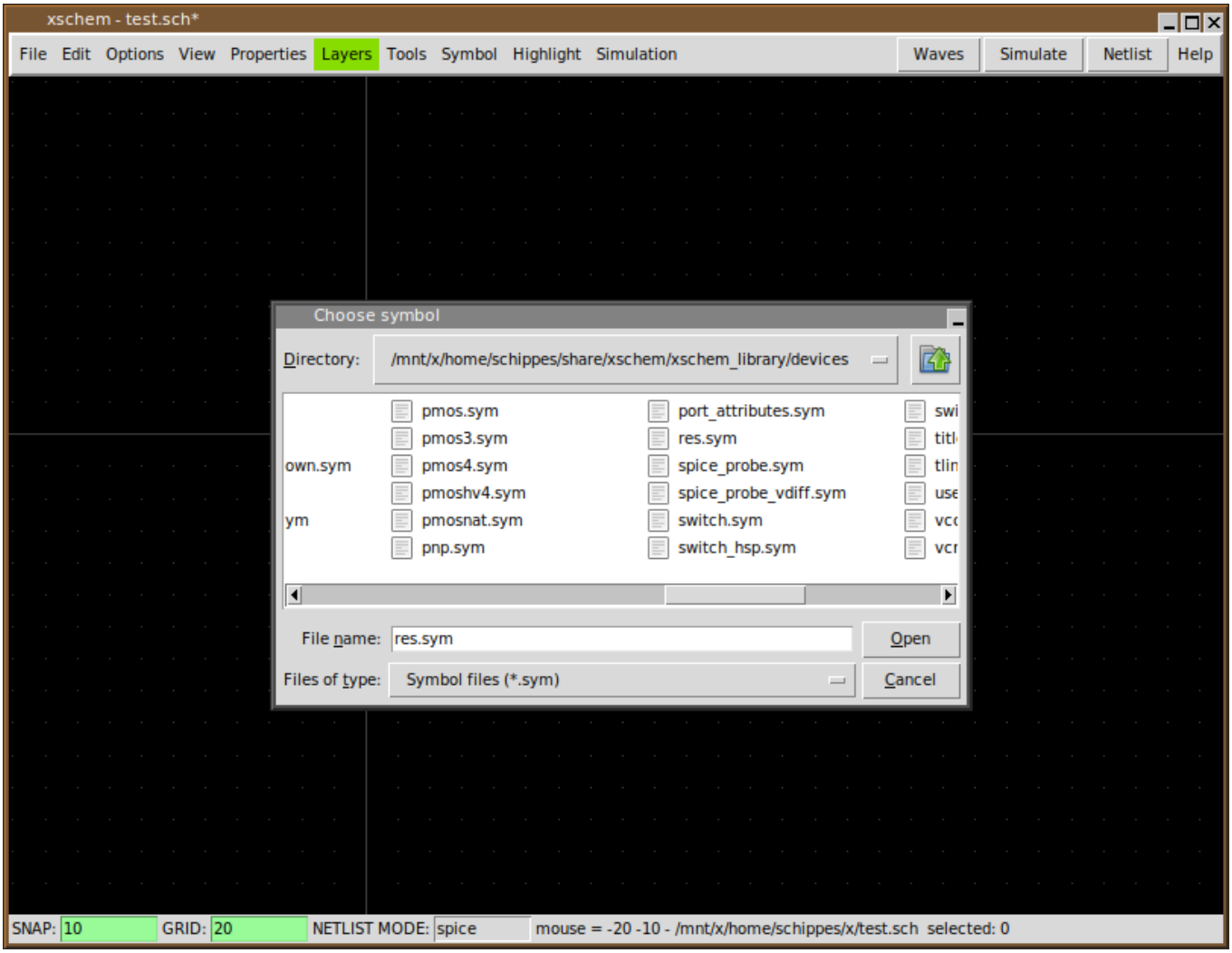
```
examples  pcb
```

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.

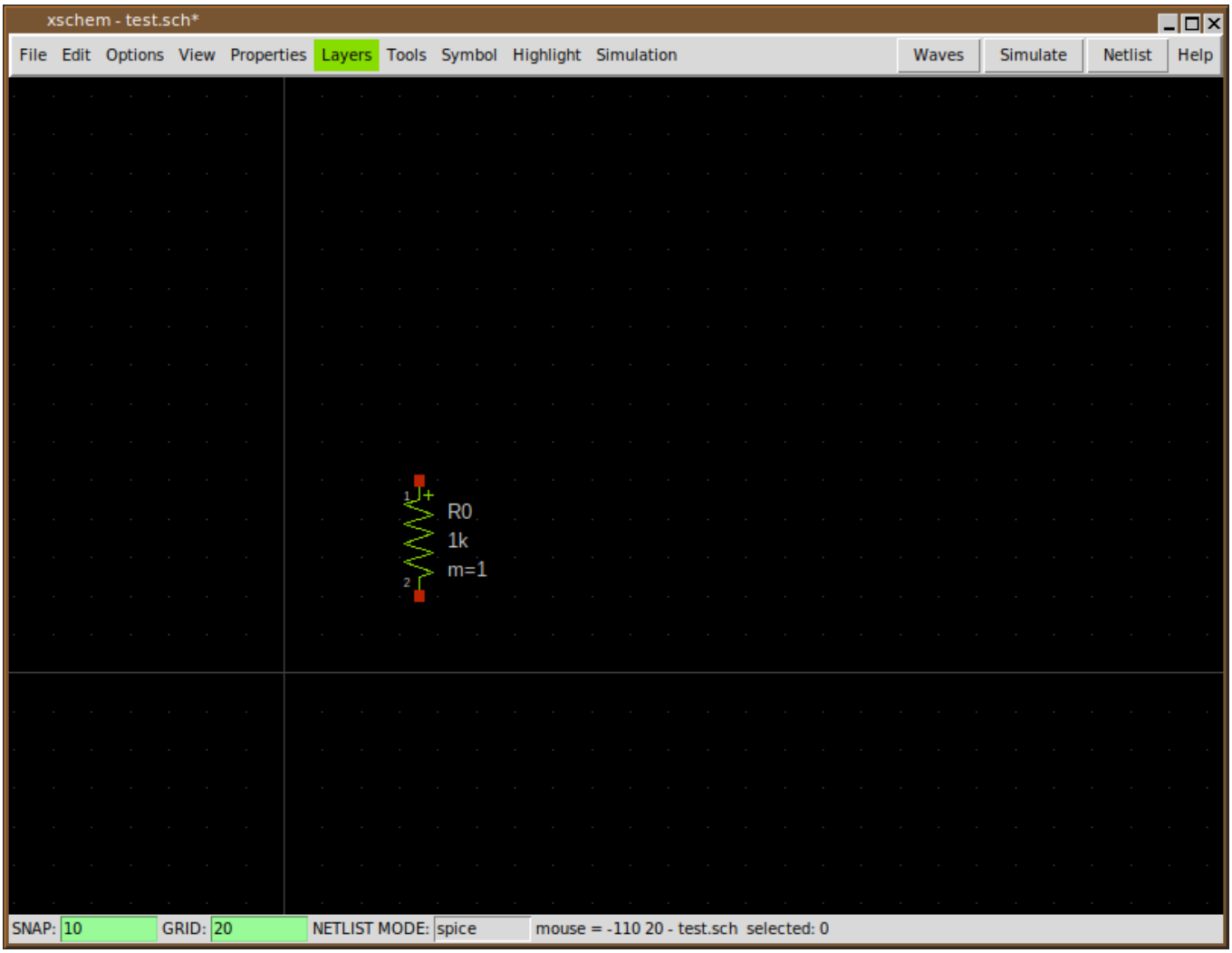
PREV UP NEXT

# 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.

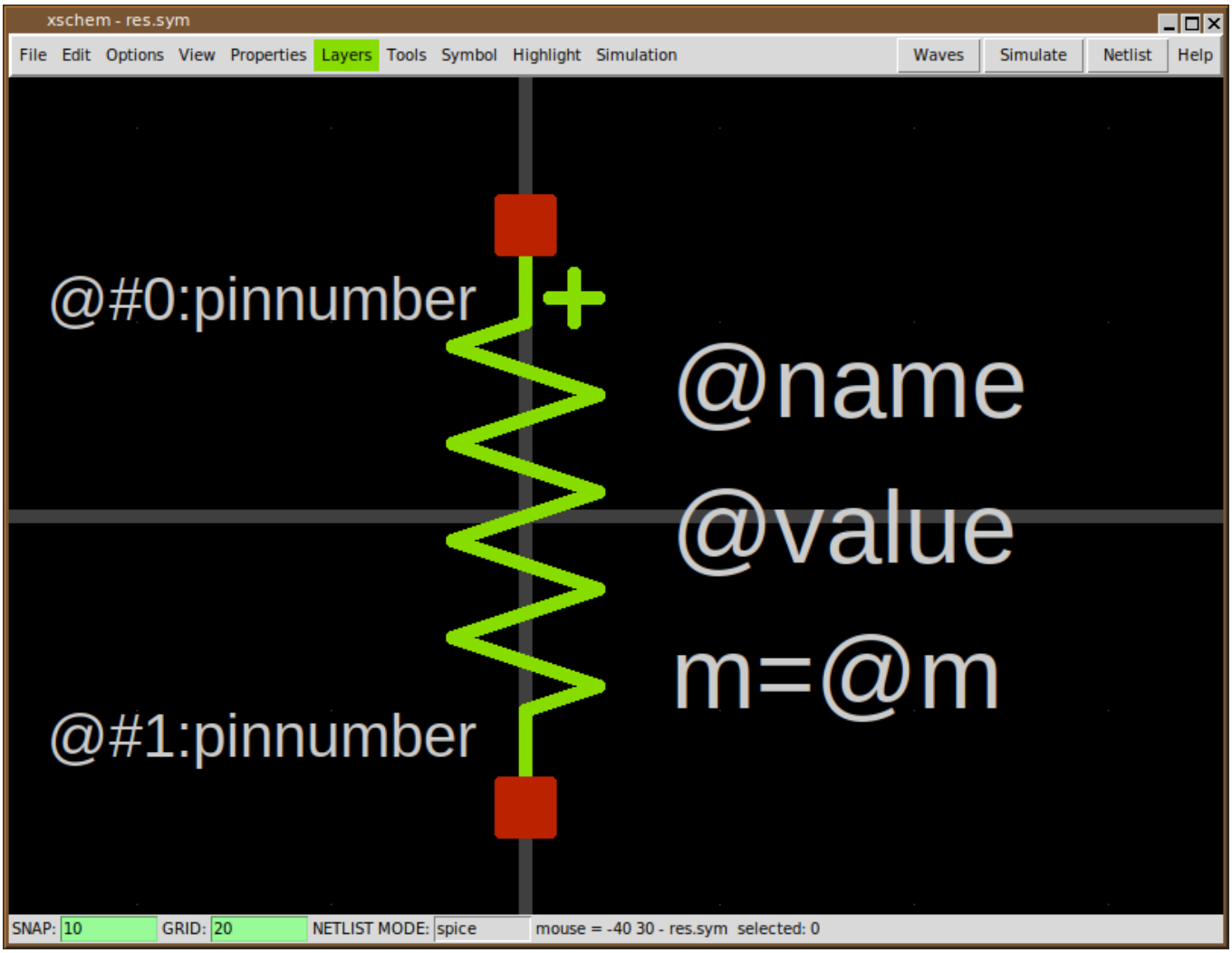



Use the file selector dialog to locate **`res.sym`**.




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


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'.




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.

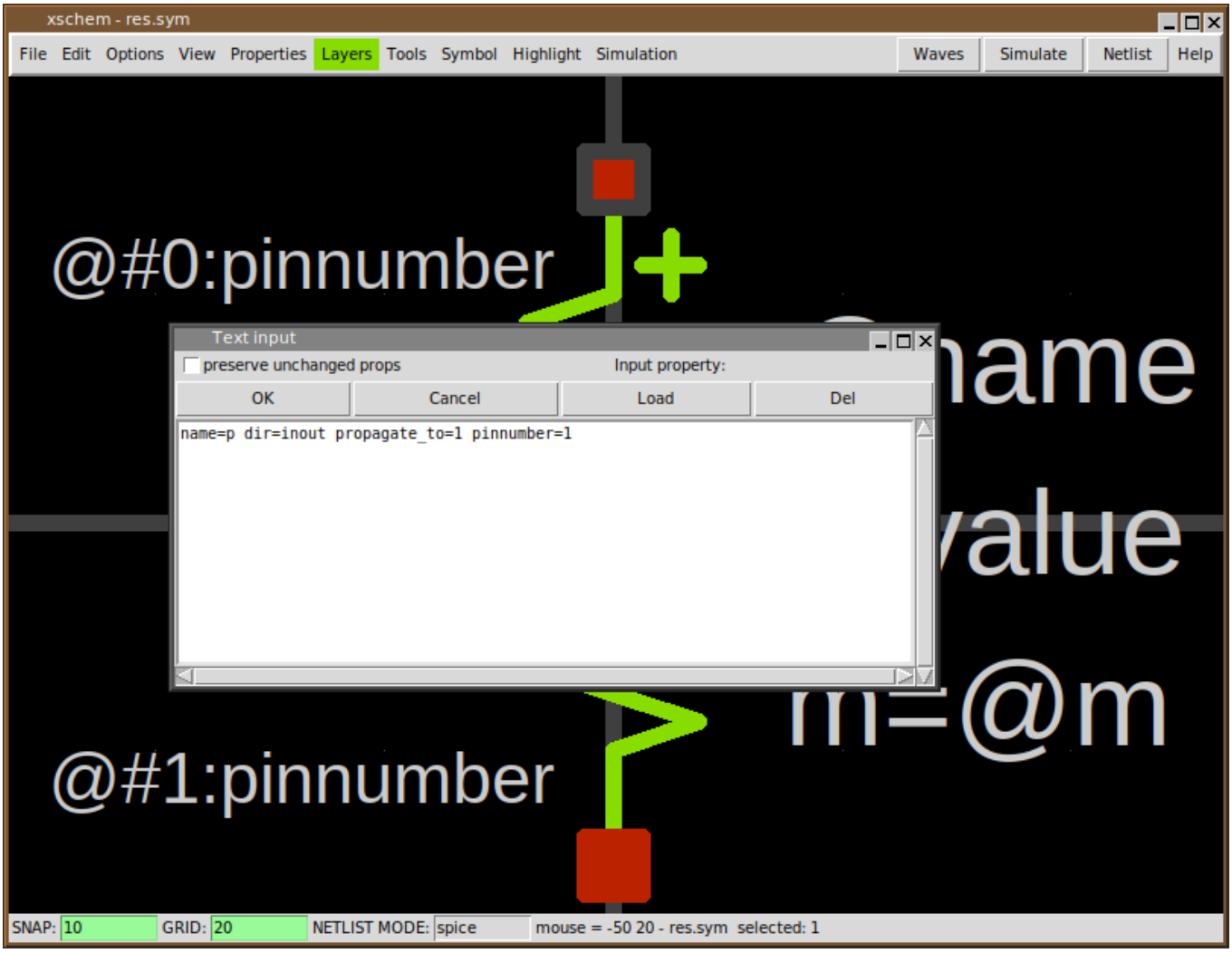
PREV UP NEXT

# XSCHEM PROPERTIES

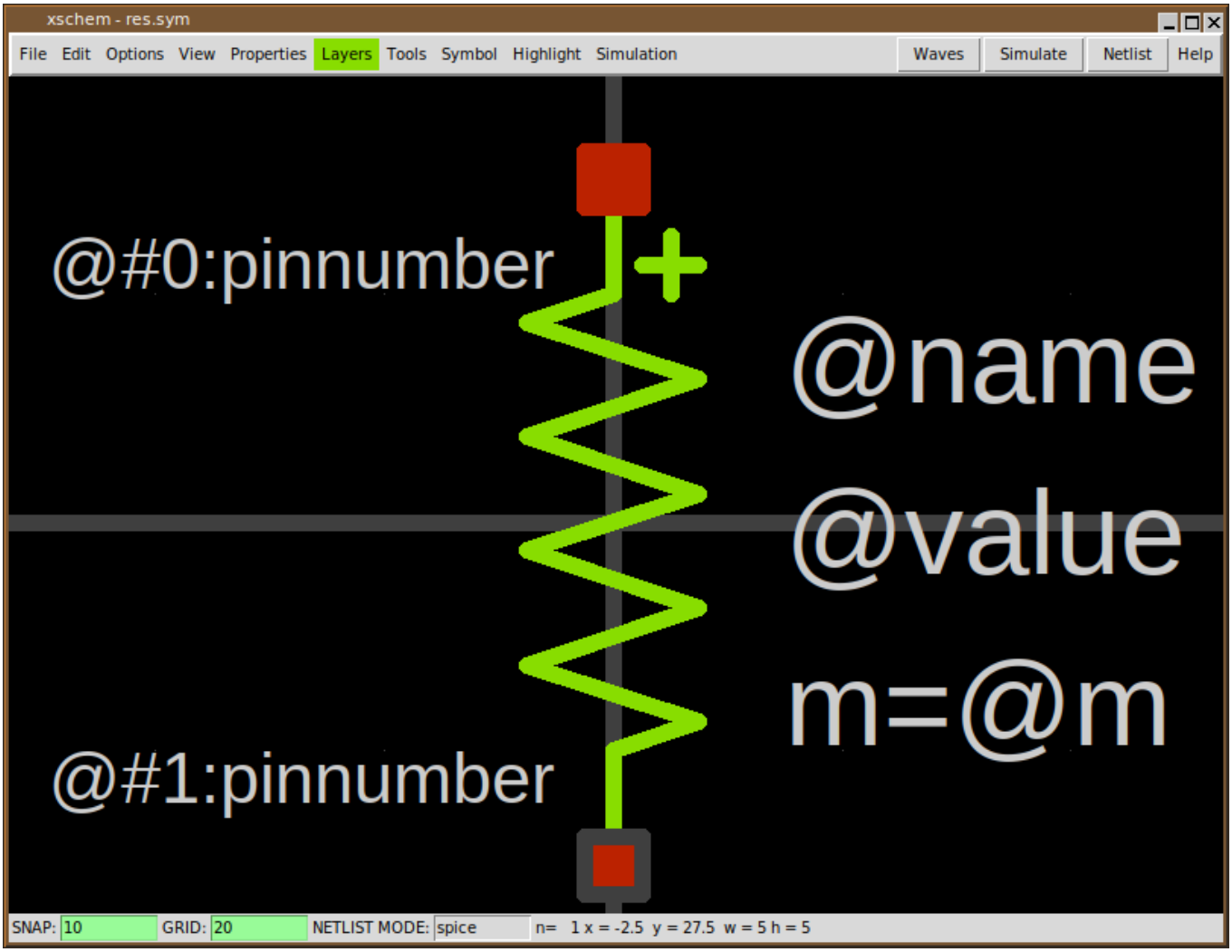
Properties are text strings that are associated to XSCHEM objects. All graphic primitives support properties.

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

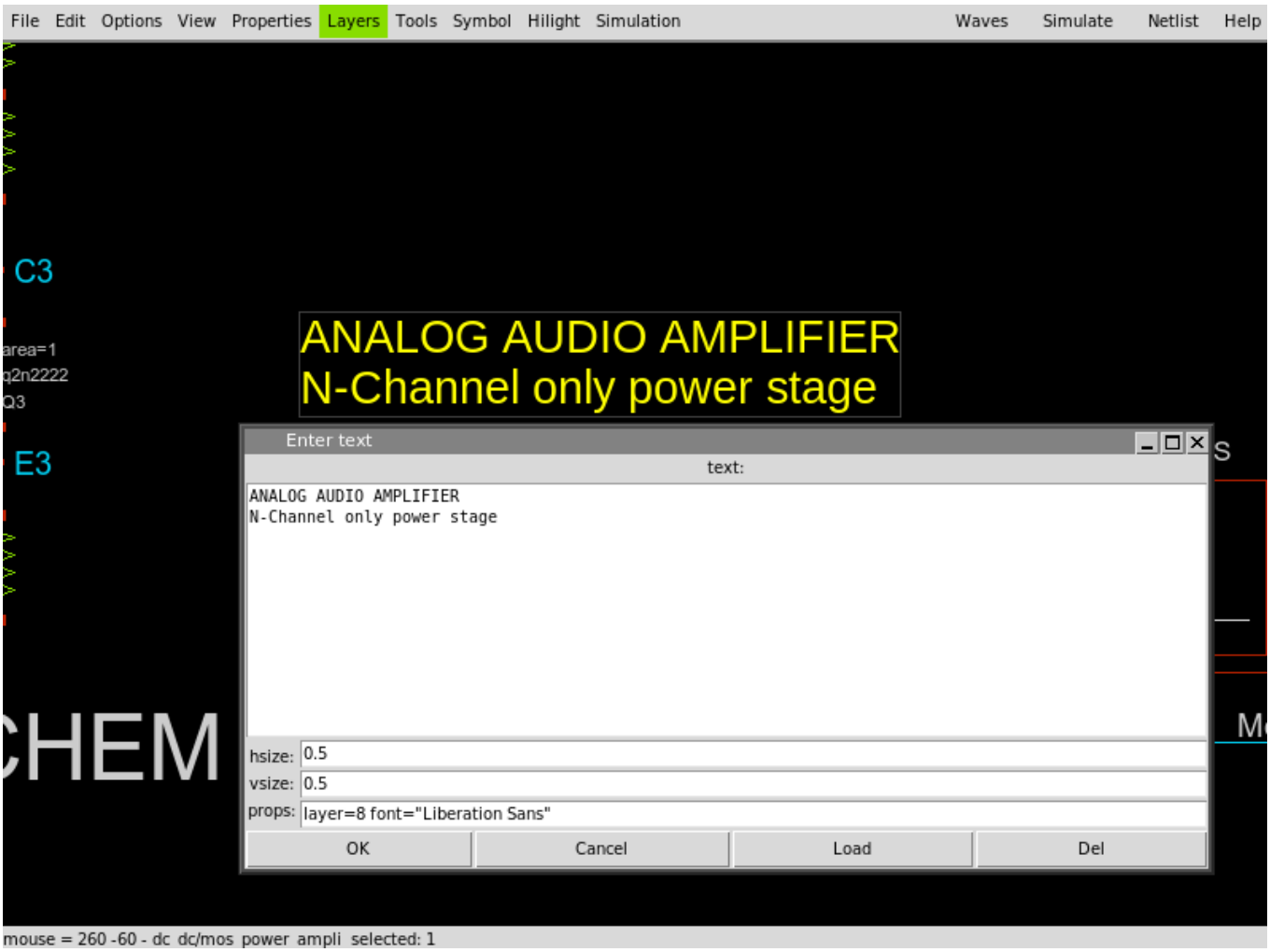
Consider for example the **`res.sym`** symbol (you may open it with the **`File->Open`** menu item) if you click inside one of the red pins and press the 'edit property' bindkey **`'q'`** a dialog box shows the property string associated with the selected pin:



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 attaced 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:




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.

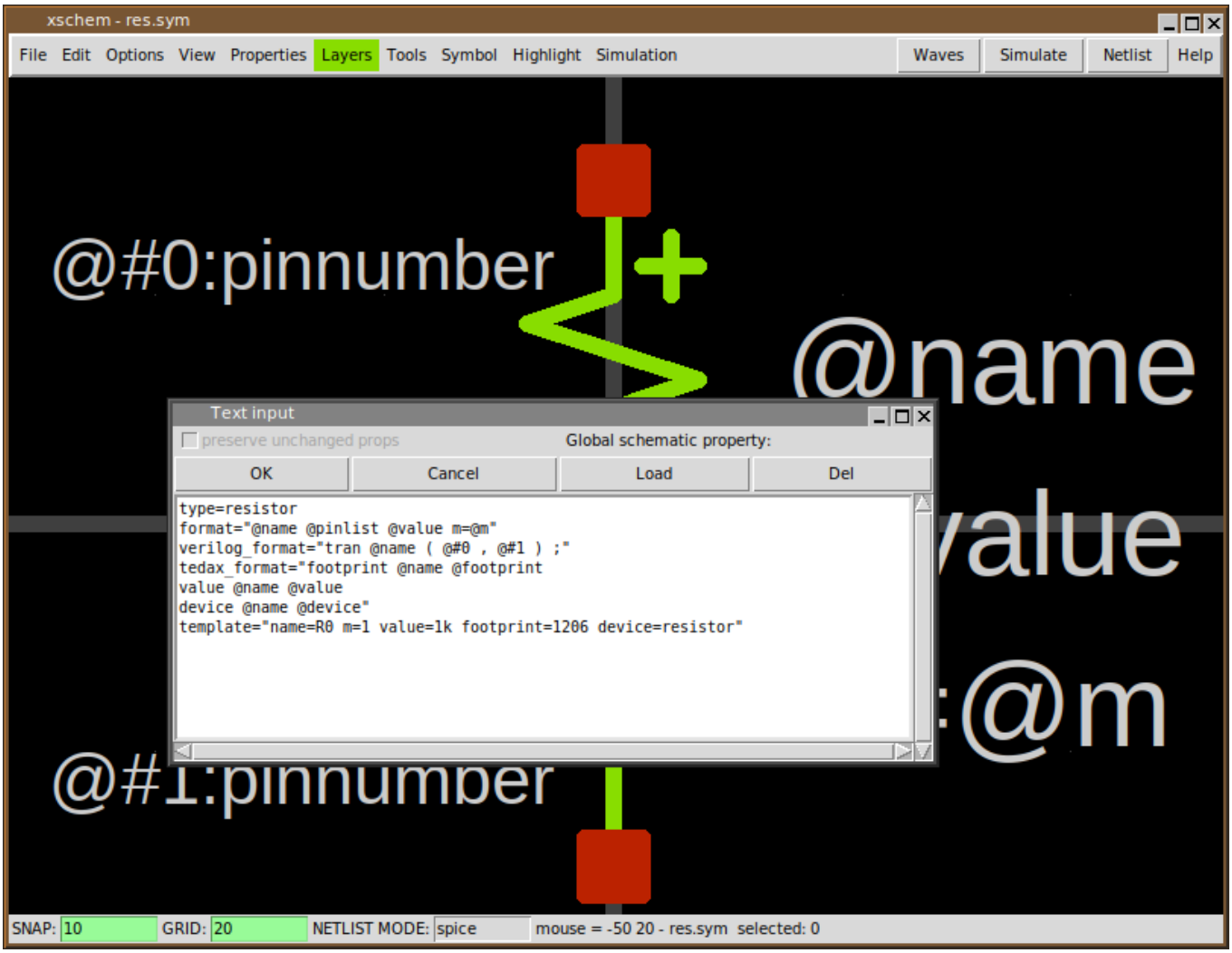



## GLOBAL PROPERTIES

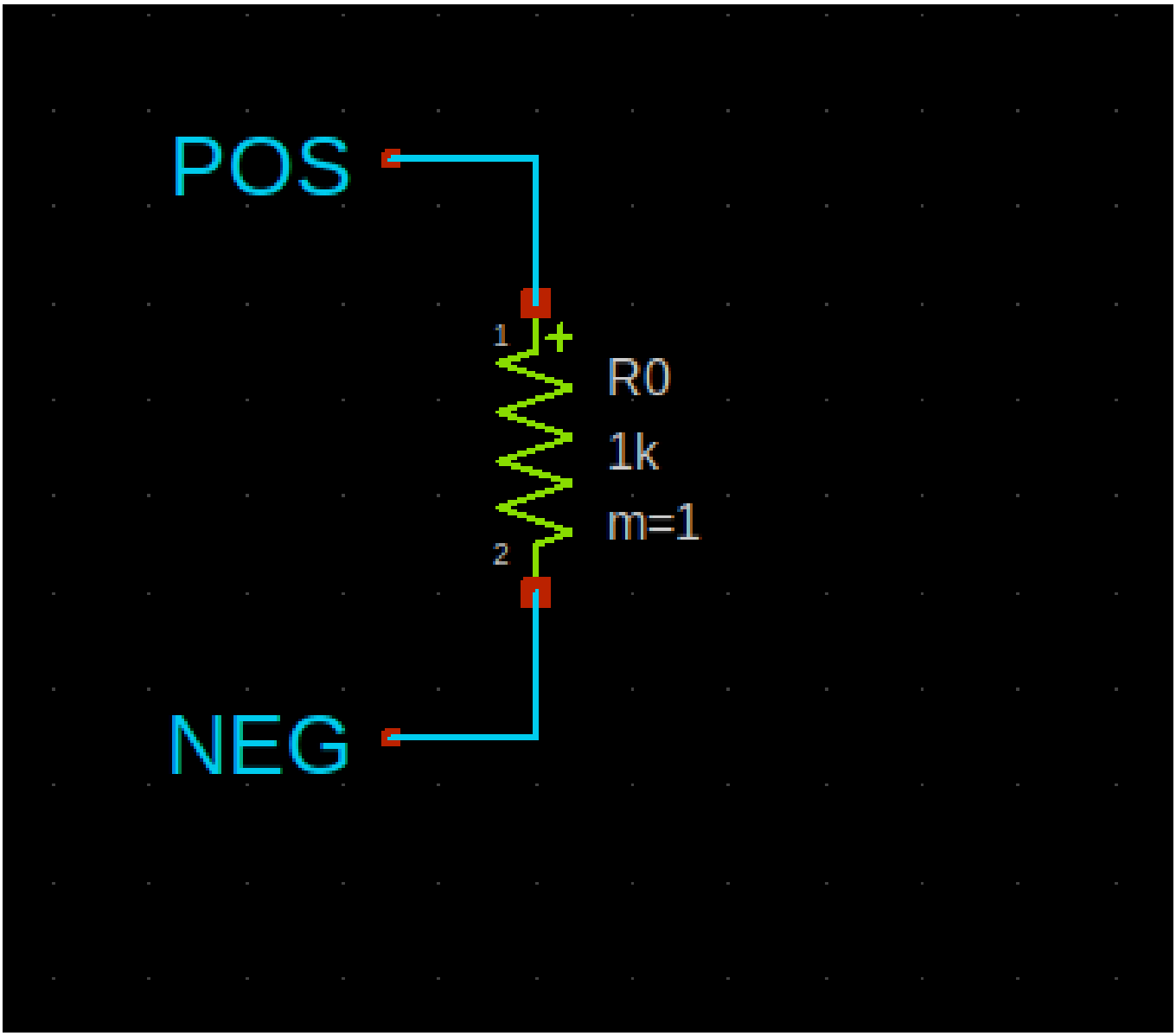
If you click outside of any displayed graphics in XSCHEM the selection set will be cleared. Clicking the edit property **`'q'`** key when nothing is selected will display the global property string of the schematic (.sch) or symbol window (.sym).

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.

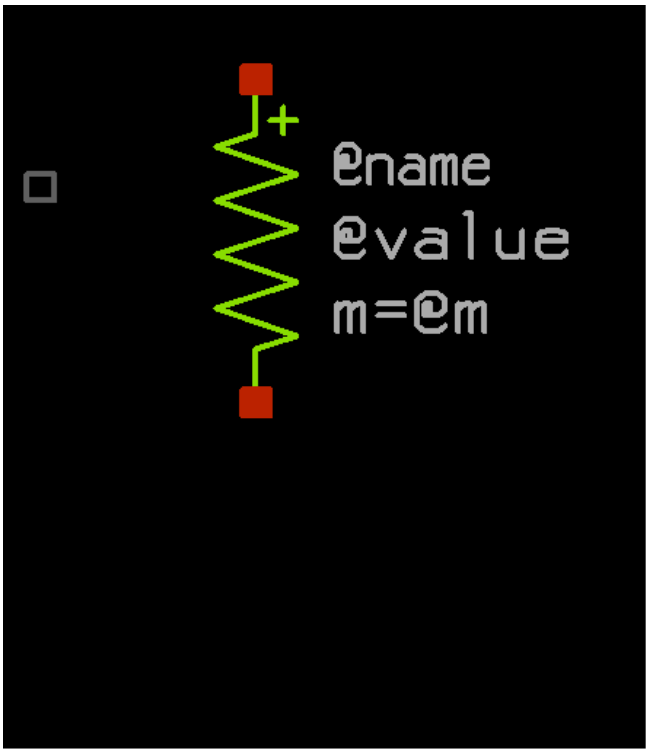
So, in addition to properties associated to graphical objects and symbols, we also have properties associated to schematic (.sch) and symbol files (.sym)

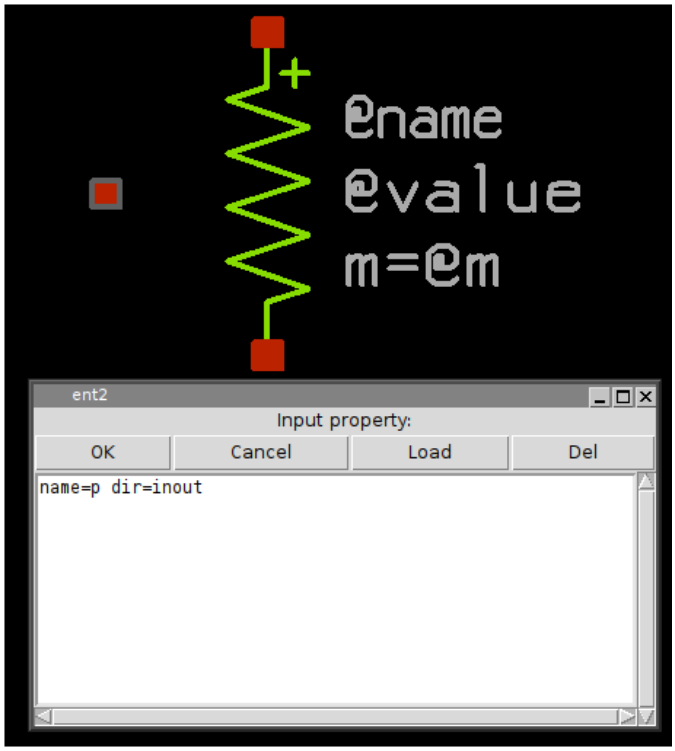



In the above 'Symbol' global property string, the **`format`** attribute defines the format of the SPICE netlist. The SPICE netlist element line starts with the symbol name (in this case a resistor so 'rxxxxx'), the list of pins, the resistor value and a multiplicity factor (m).
**`@pinlist`** will resolve 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.

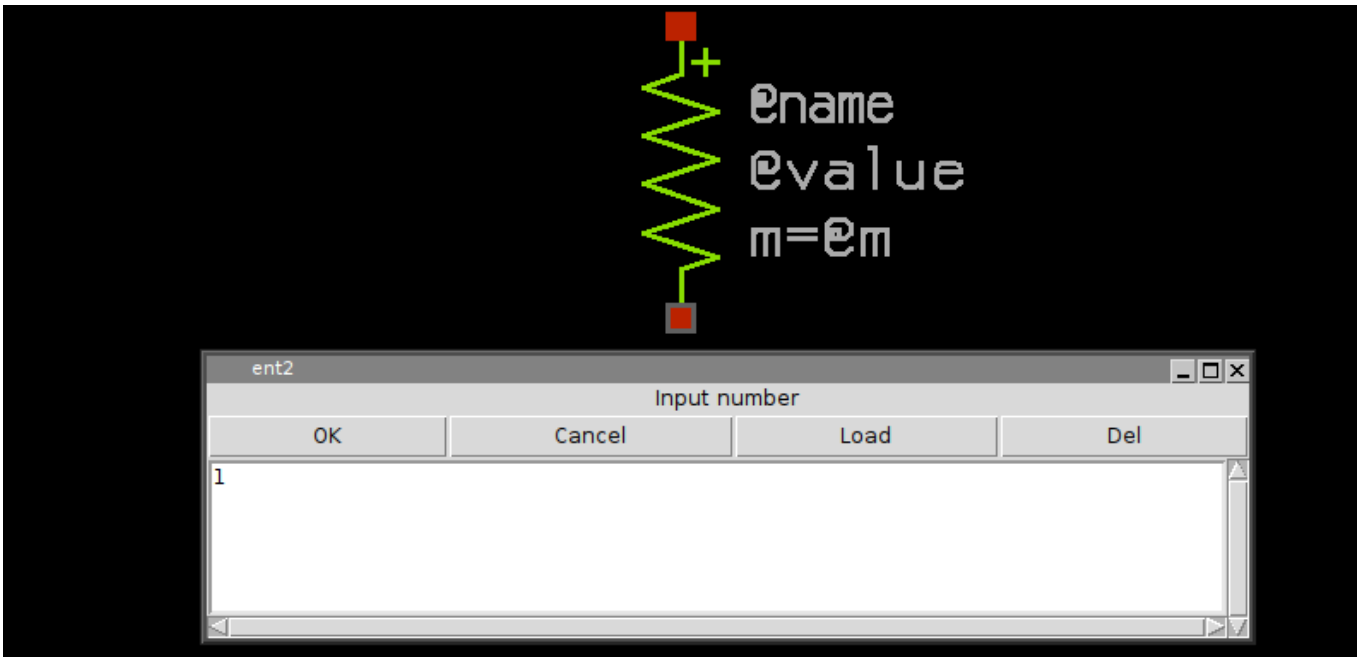



After copying the pin you may change its properties, for example you will change its property string to something like: **name=body dir=in** (just as an example).
Note that pins in symbols are nothing more than rectangles drawn with the **pin** layer; instead of copying an existing one you may create 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:

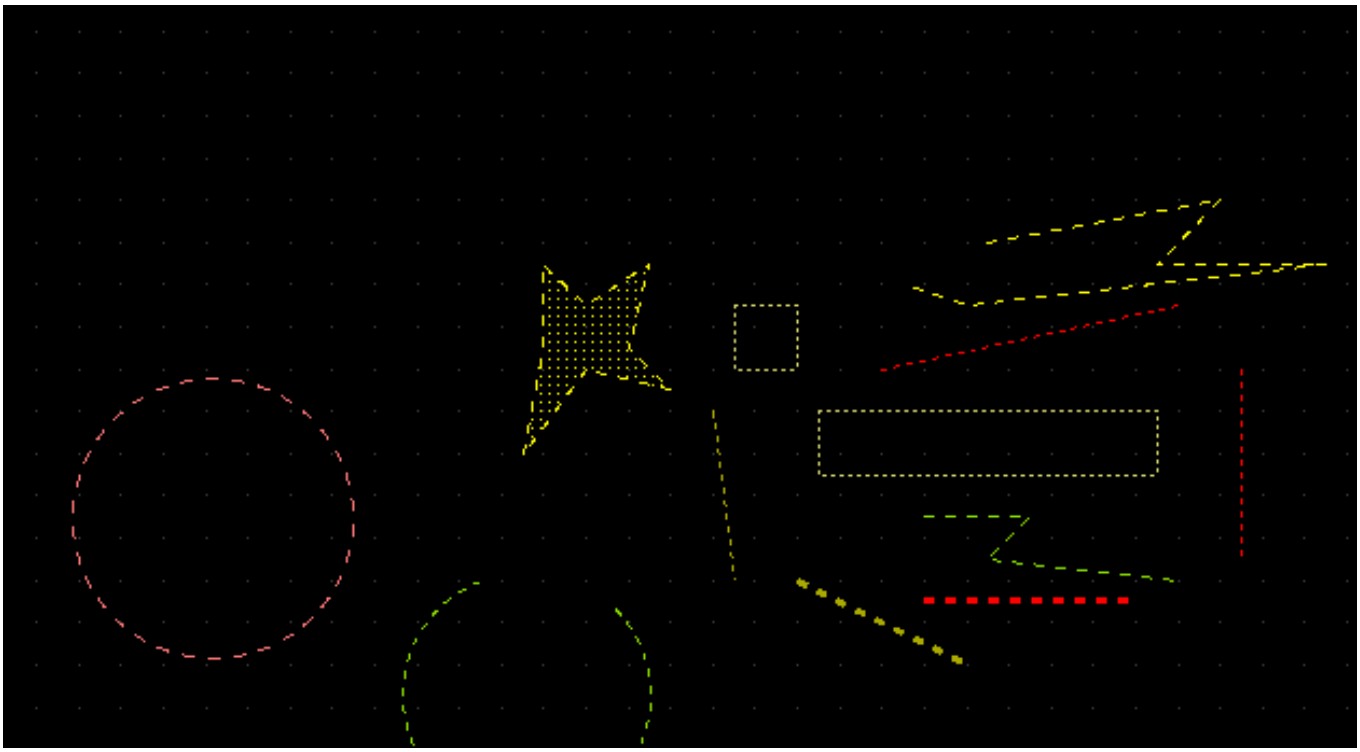
- **`dash=n`**, where n = integer. This specifies dashed mode drawing for the specified object.

The following attribute may be set on arcs and polygons:

- **`fill=true`**. This specifies to fill the object with the layer predefined fill style.

The following attribute may be set on wires and lines:

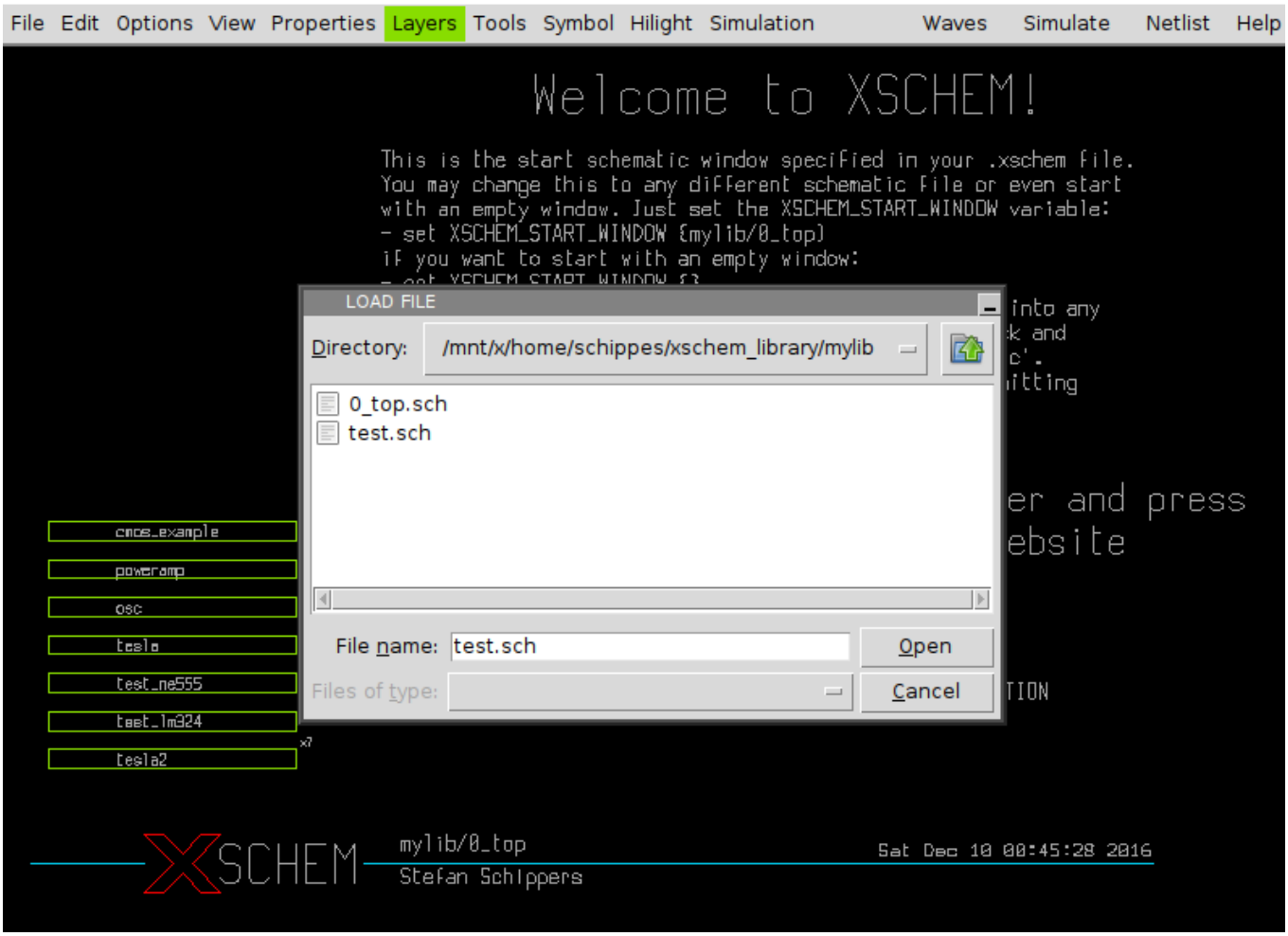
- **`bus=true`**. This specifies to draw a wider line. Mostly used to display wire buses.



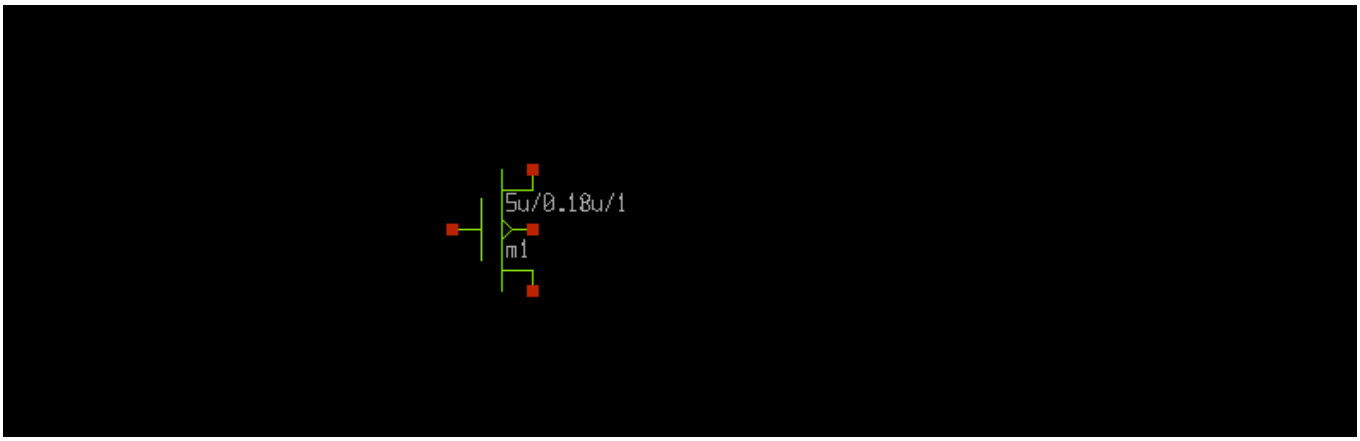
PREV UP NEXT

# COMPONENT INSTANTIATION

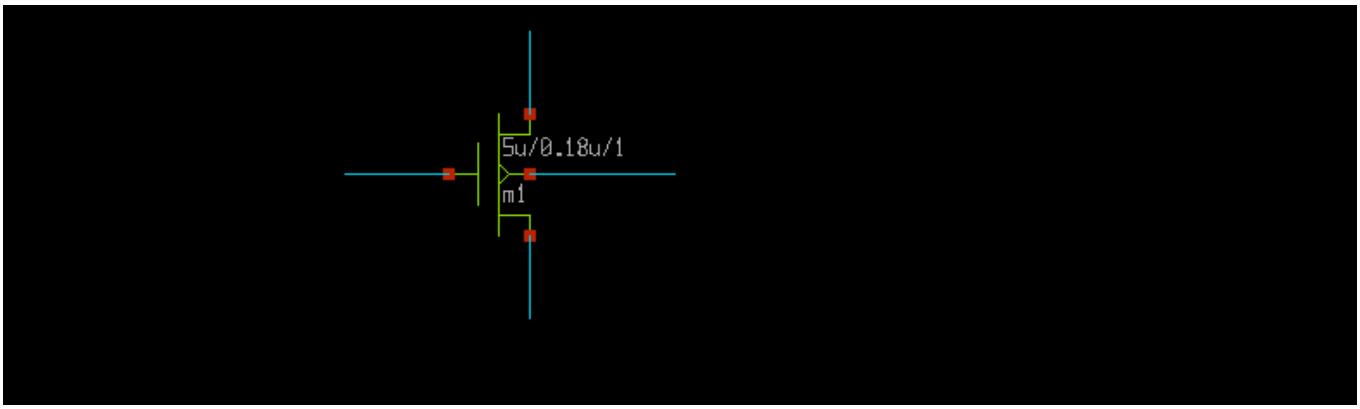
In the RUN XSCHEM slide some instructions were provided as examples to place a component in the schematic. Now we will cover the topic in more detail with emphasis on component properties. Start by opening a test schematic window (you may delete any existing stuff in it if any).



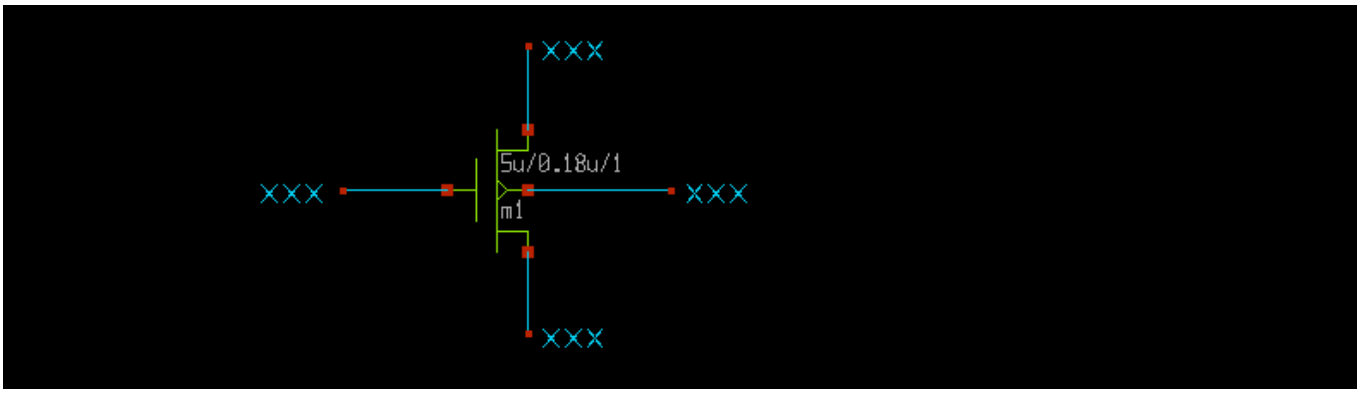

Now start by inserting a component, consider for example **`devices/nmos4.sym`**; press the **`Insert`** key, navigate to the **`devices`** design library and open the **`nmos4.sym`** symbol.



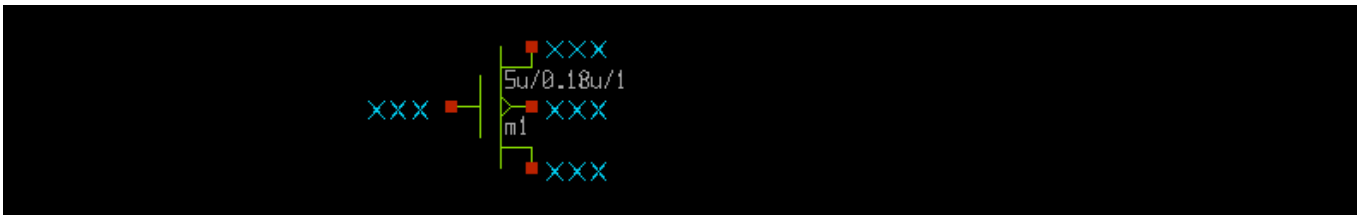

Now draw some wires on each pin of the nmos; place the mouse pointer on the component pins and use the **'w'** bindkey.



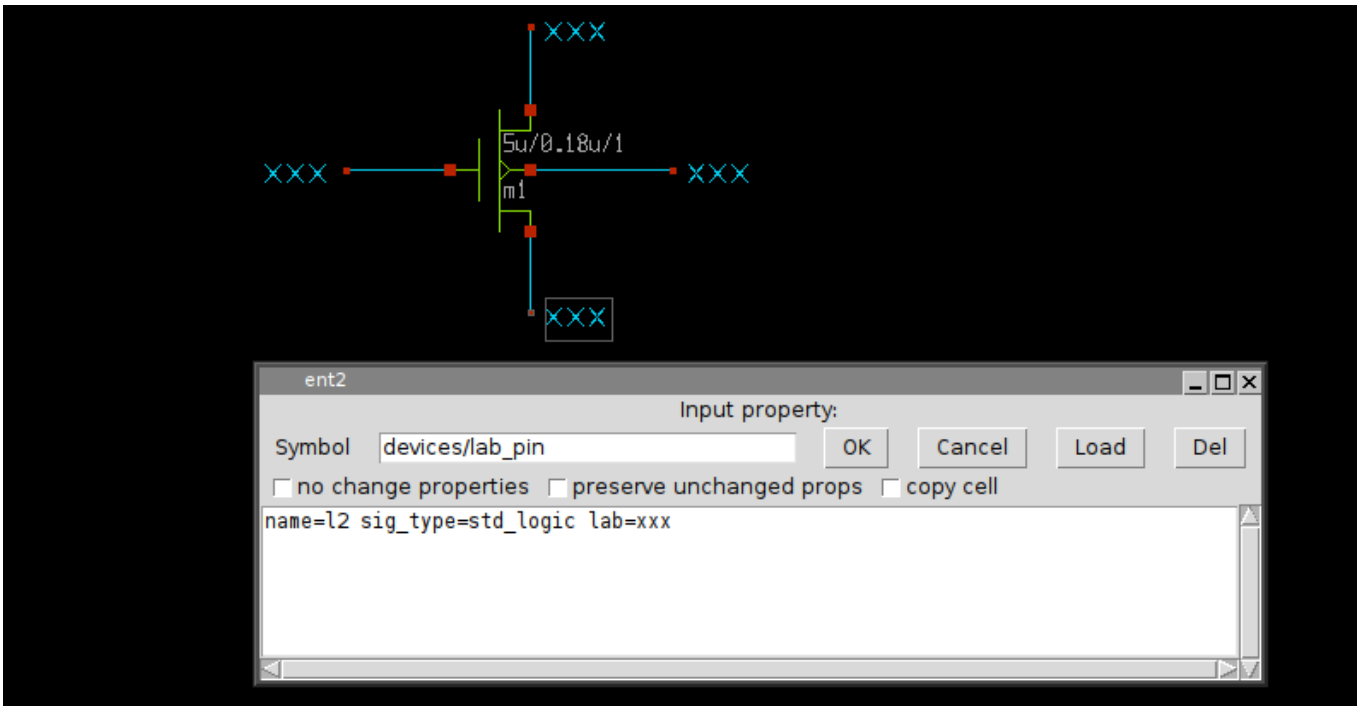

we need now to put labels on wire ends: use the **Insert** key and locate the **devices/lab_pin.sym** symbol. After the **lab_pin** symbol is placed you can move it by selecting it with the mouse and pressing the **'m'** bindkey. You can also flip ( **'F'**) and rotate while moving (**'R'**) to adjust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



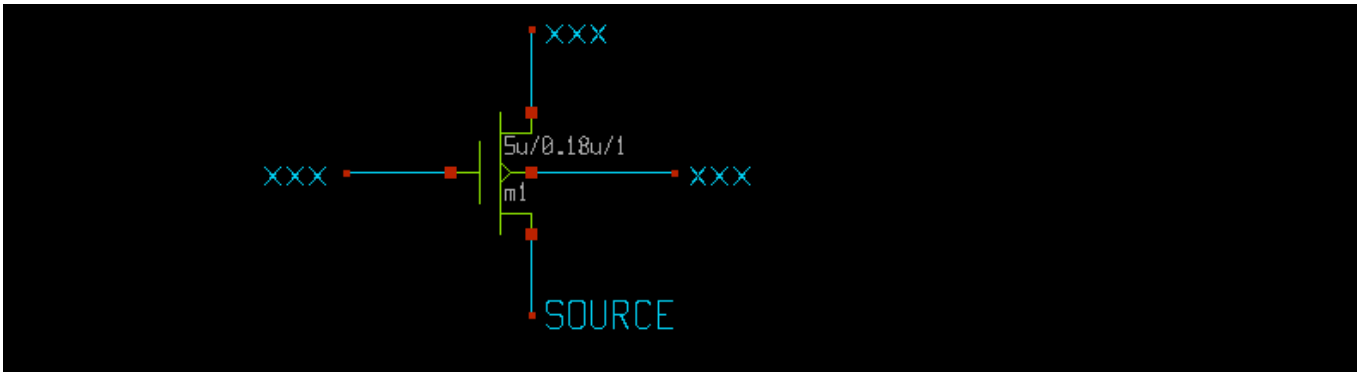

This is what an electrical circuit is all about: a network of wires and components. In this schematic we have 5 components (4 labels and one mos) and 4 nets. It is not mandatory to put a wire segment between component pins; we could equally well do this:



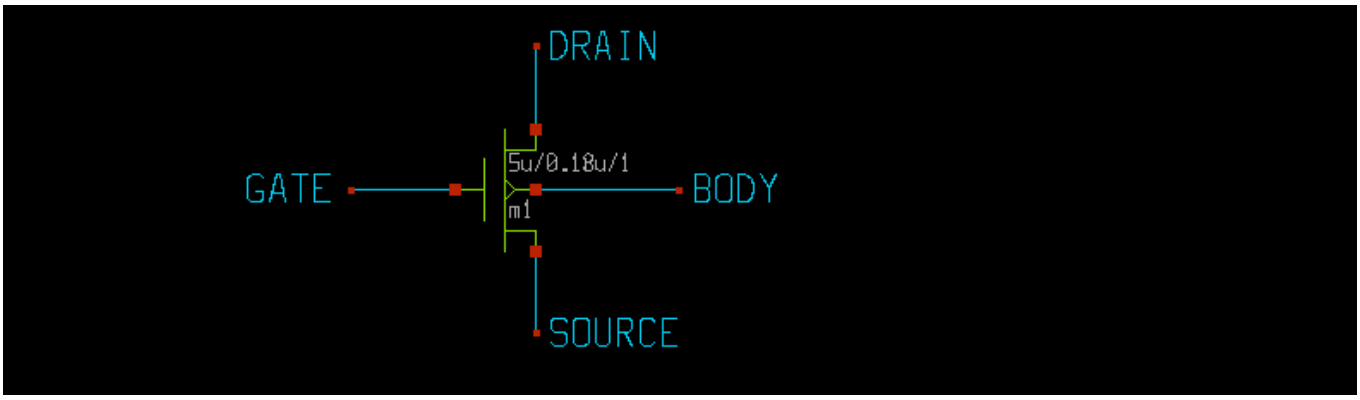

This circuit is absolutely equivalent to the previous one: it will produce the same device connectivity netlist.
Now we need to set appropriate labels on the NMOS terminals. This is -again- accomplished with component properties.
Select the wire label on the nmos source pin and press the **'q'** bindkey:



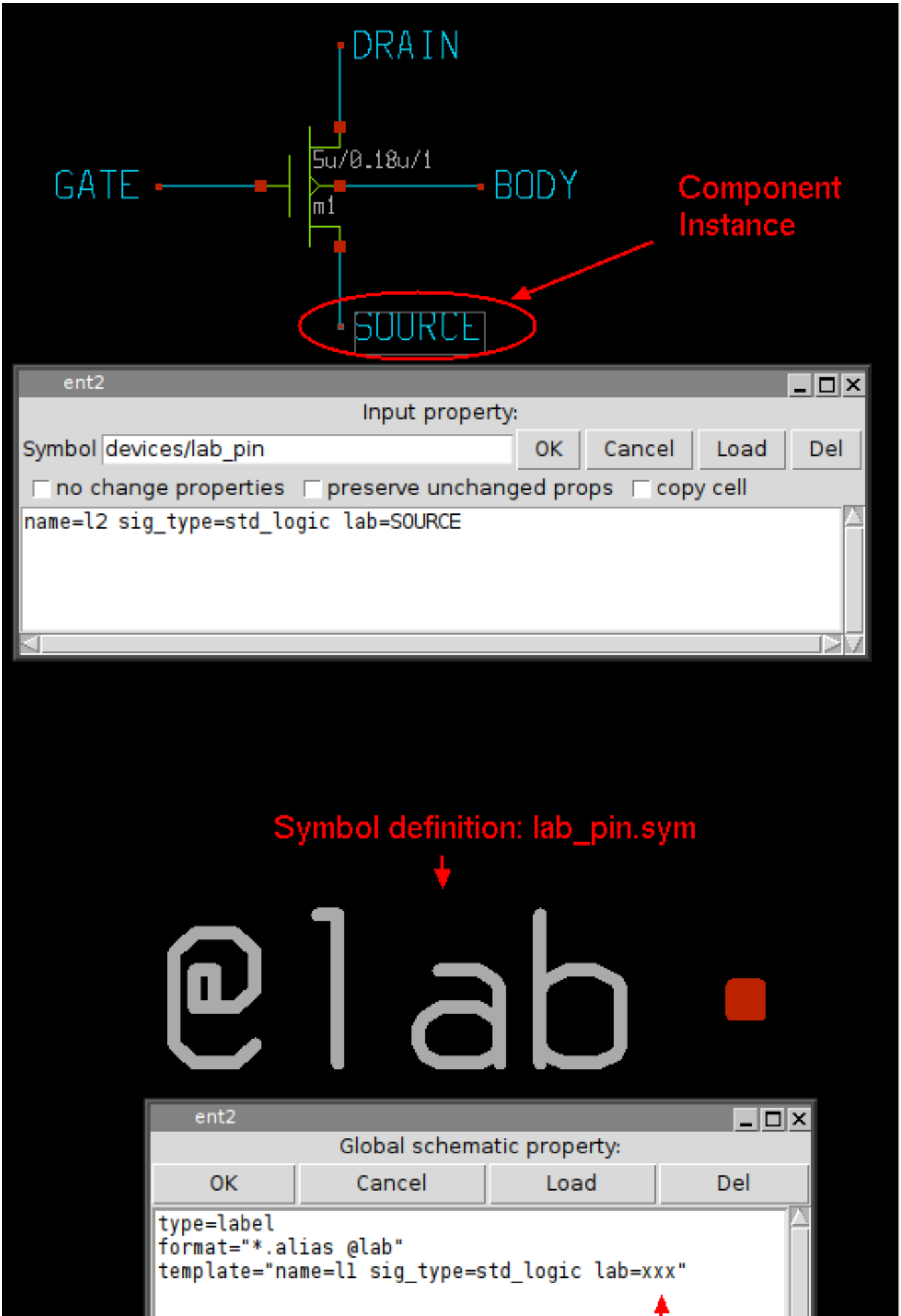

Now, replace the 'xxx' default string in the dialog with a different name (example: SOURCE) After clicking **OK** the source terminal will have the right label.




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



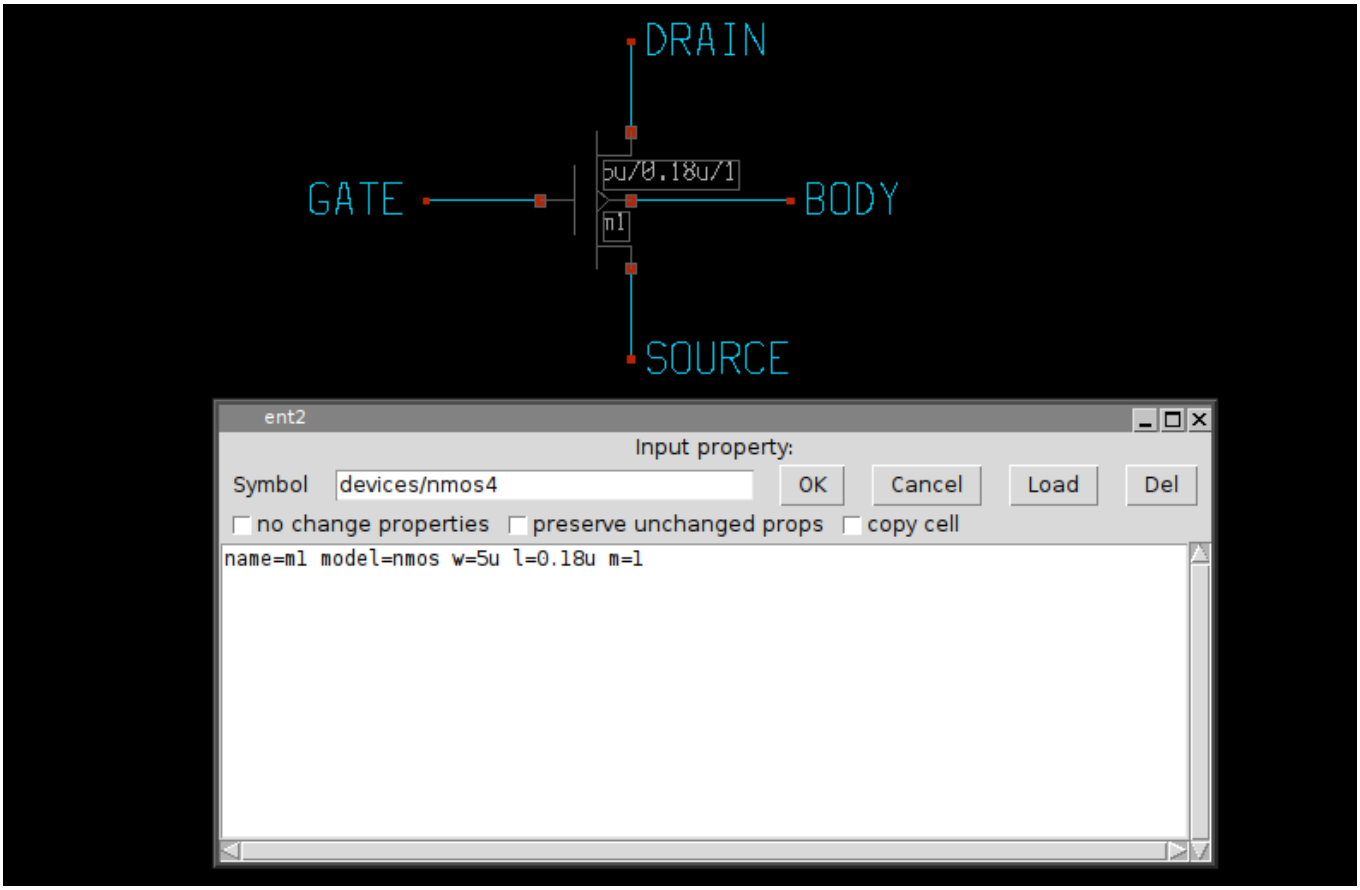

The following picture shows the **`lab_pin`** component with its properties and the corresponding symbol definition with its global properties (remember global properties in the xschem_properties slide)


DRAIN
GATE
5u/0.18u/1
m1
BODY
Component Instance
SOURCE
ent2
Input property:
Symbol devices/lab_pin
OK
Cancel
Load
Del
no change properties
preserve unchanged props
copy cell
name=l2 sig_type=std_logic lab=SOURCE
Symbol definition: lab_pin.sym
ent2
Global schematic property:
OK
Cancel
Load
Del
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
```

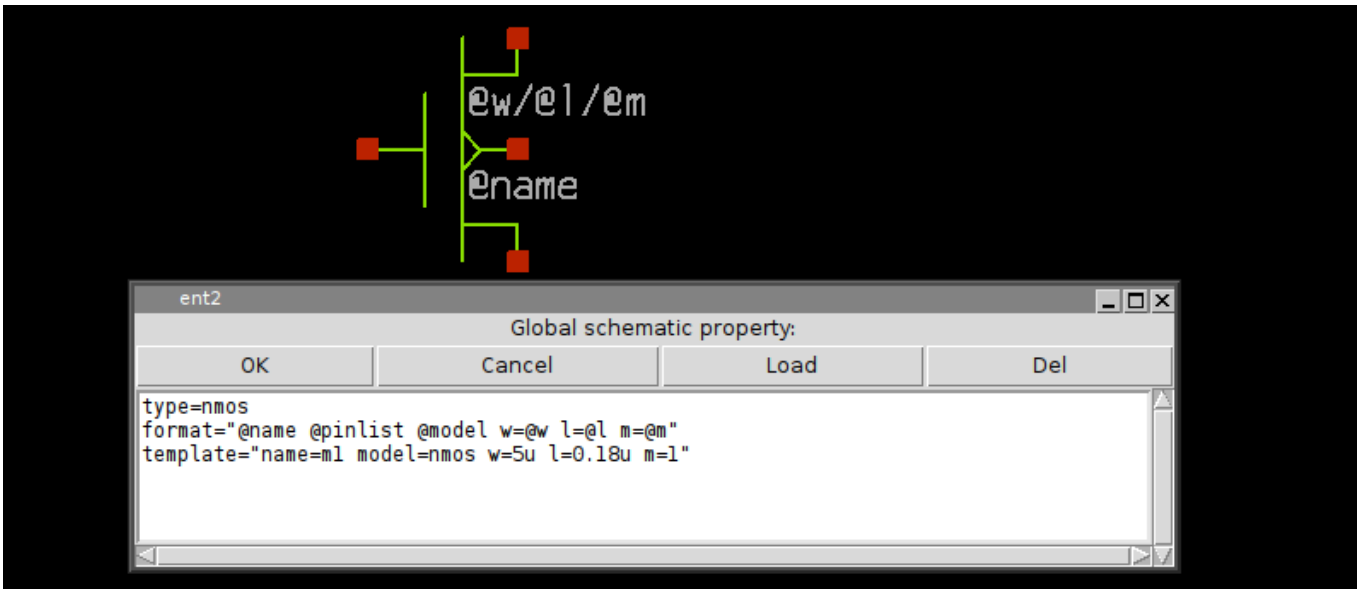
We need now to edit the nmos properties. Select it and press the **'q'** bindkey



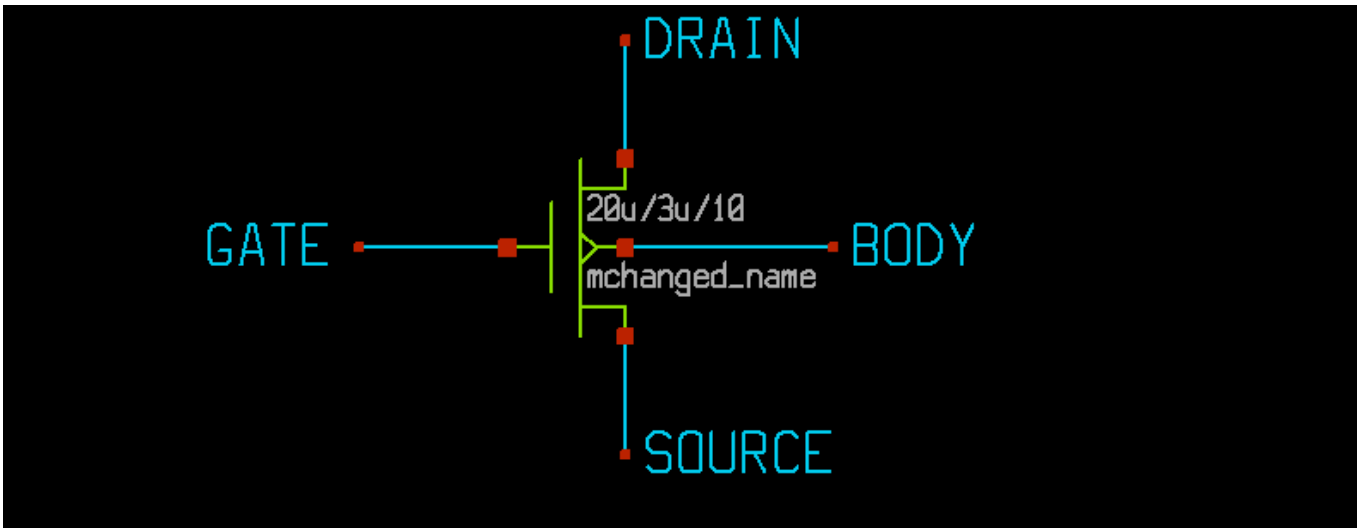

from the edit properties dialog you see there are 5 attributes with values defined:

- 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.




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.

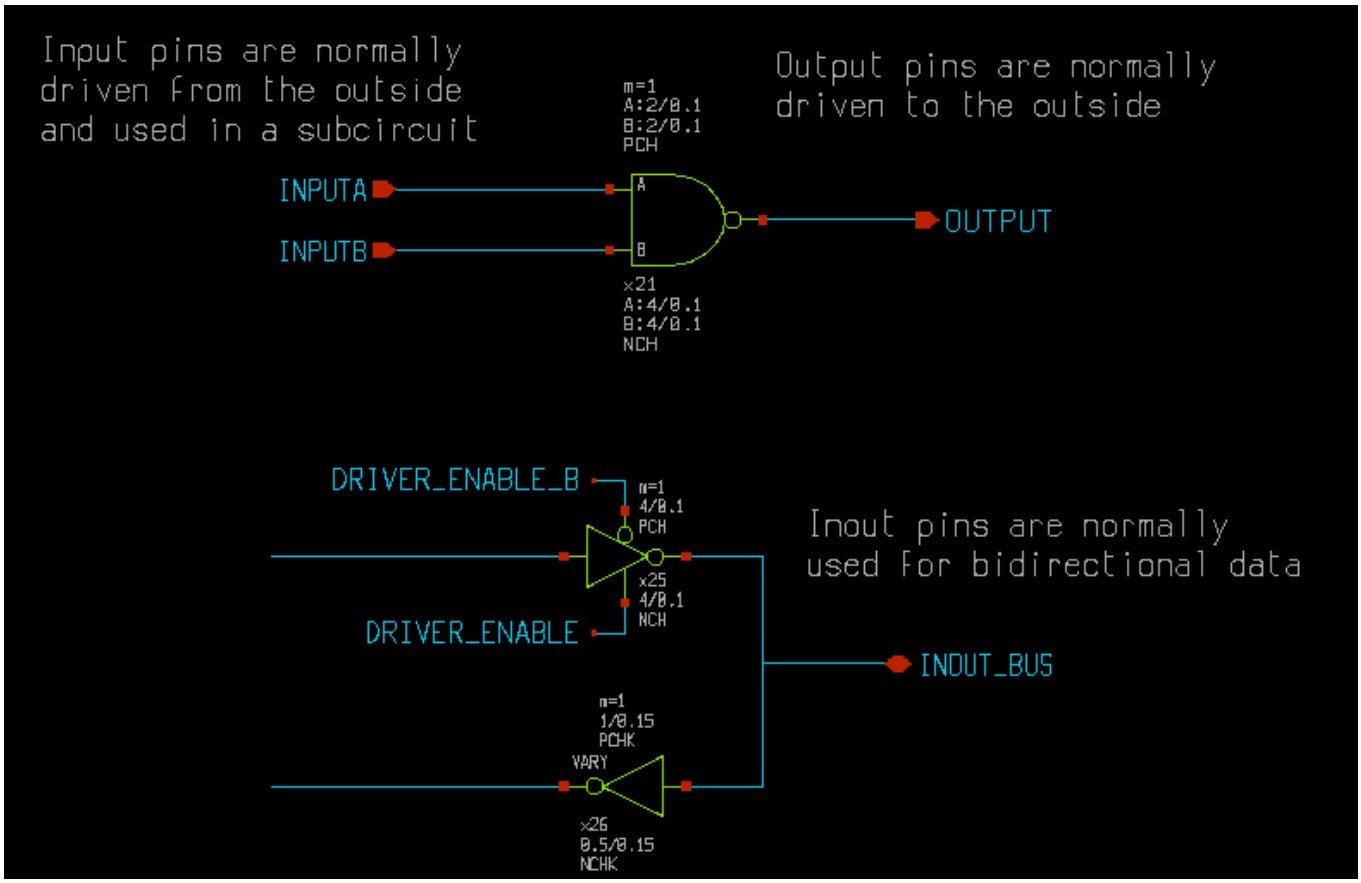
## SPECIAL COMPONENTS

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.




- **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**

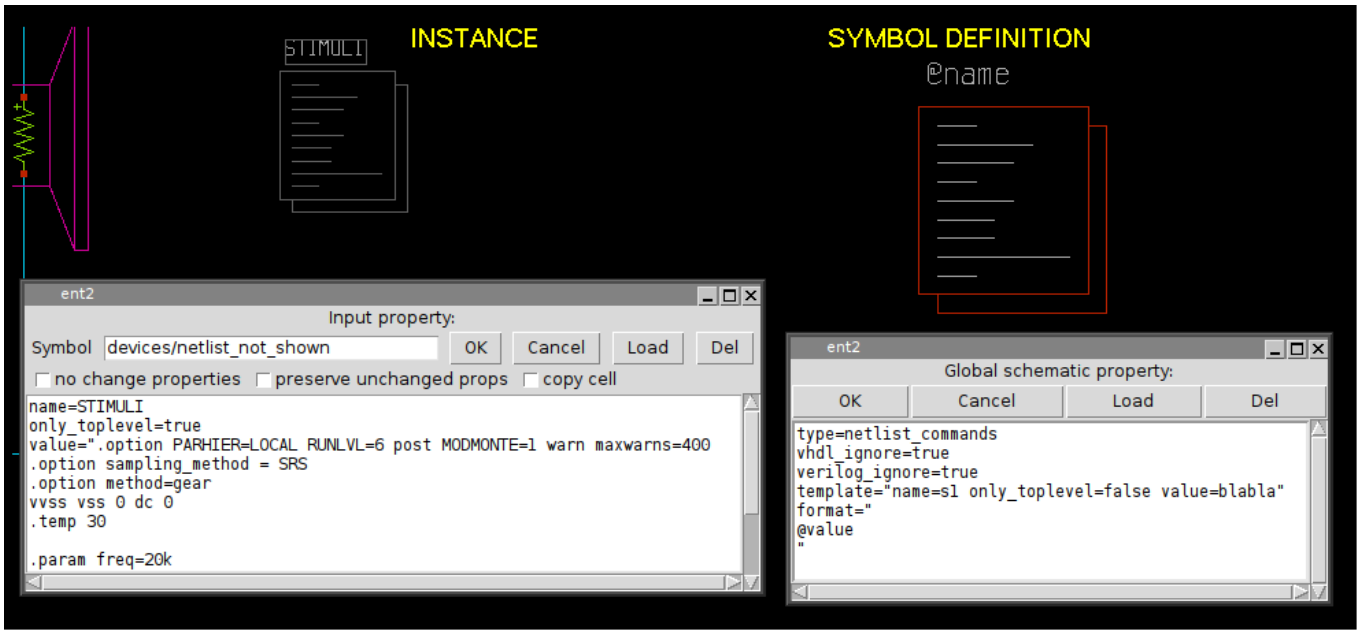
PREV UP NEXT

# SYMBOL PROPERTY SYNTAX

## GENERAL RULES

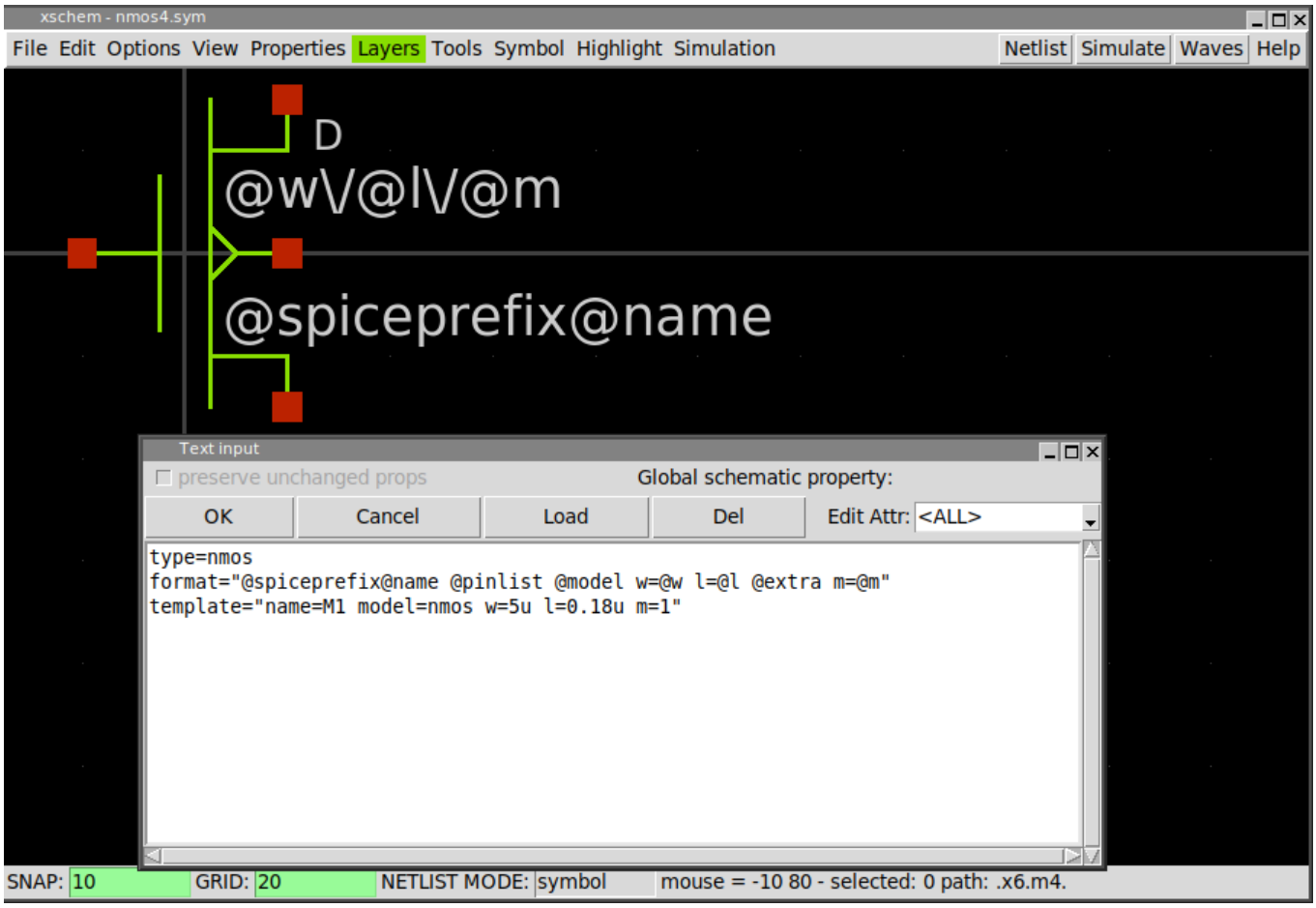
For symbols a global property string (to show it press **'q'** when nothing is selected and **Options->Symbol global attrs** is selected) defines at least 3 attributes:

- **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 instantiationi 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 instancs 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 methid.
  - **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.

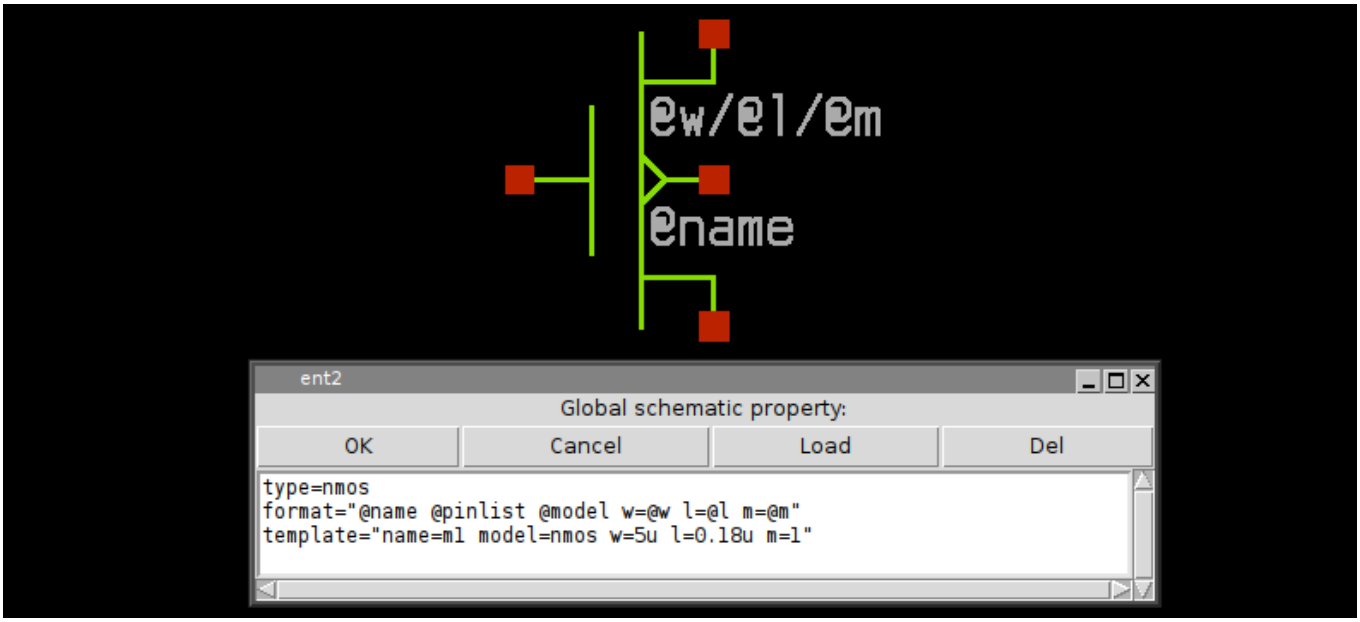



Only symbols of type **`subcircuit`** or **`primitive`** may be descended into with the **`'e'`** bindkey if they have a schematic view.

- **`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).




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




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 m
```

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

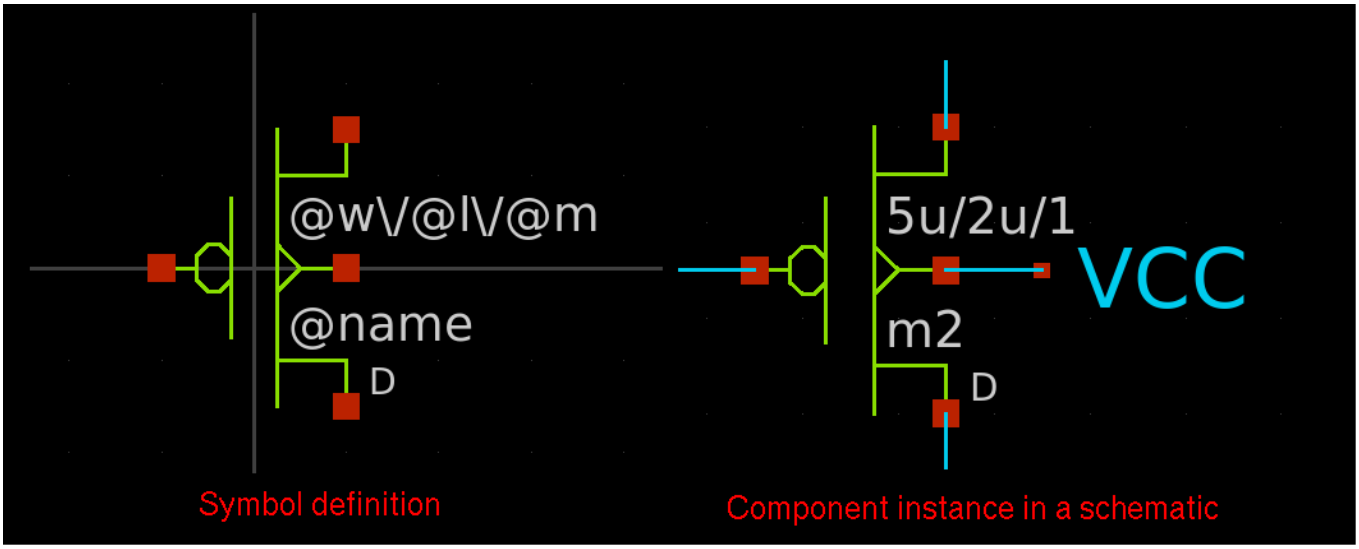
As you see double quotes are used when attribute values have spaces. For this reason if double quotes are needed in an attribute value they must be escaped with backslash **\"**

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**

  These 3 attributes tell XSCHEM to ignore completely the symbol in the respective netlist formats.

- **vhdl_stop**
- **spice_stop**
- **verilog_stop**

  These 3 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.

- **highlight**

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

- **net_name**

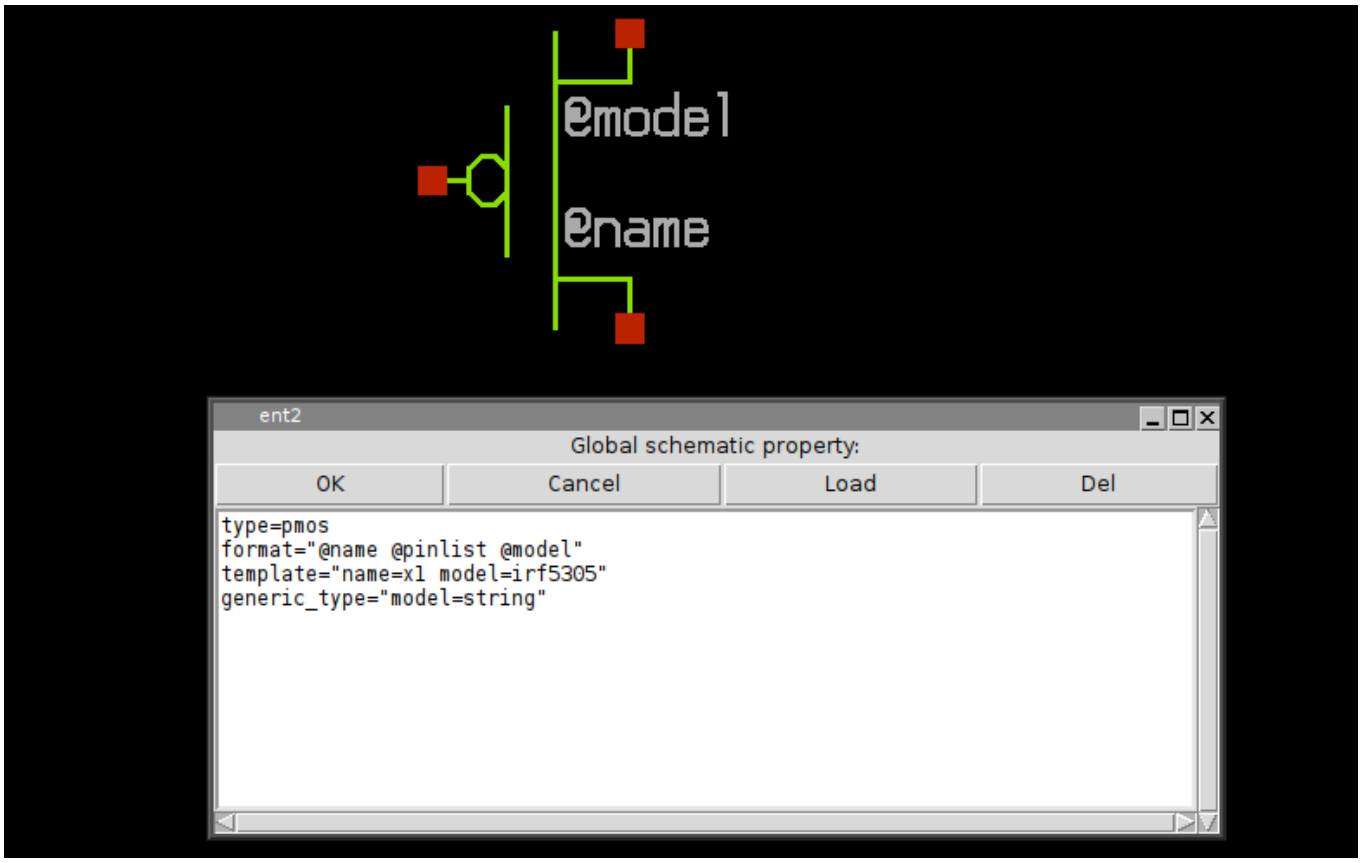
  If set to **true** the **#n:net_name** symbol attributes will display the net names attached to pin terminals. the **n** is a pin number or name.

- **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 ge 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 components. Consider the following MOS symbol definition; the **model** attribute is declared as **string** and it will be quoted in VHDL netlists.

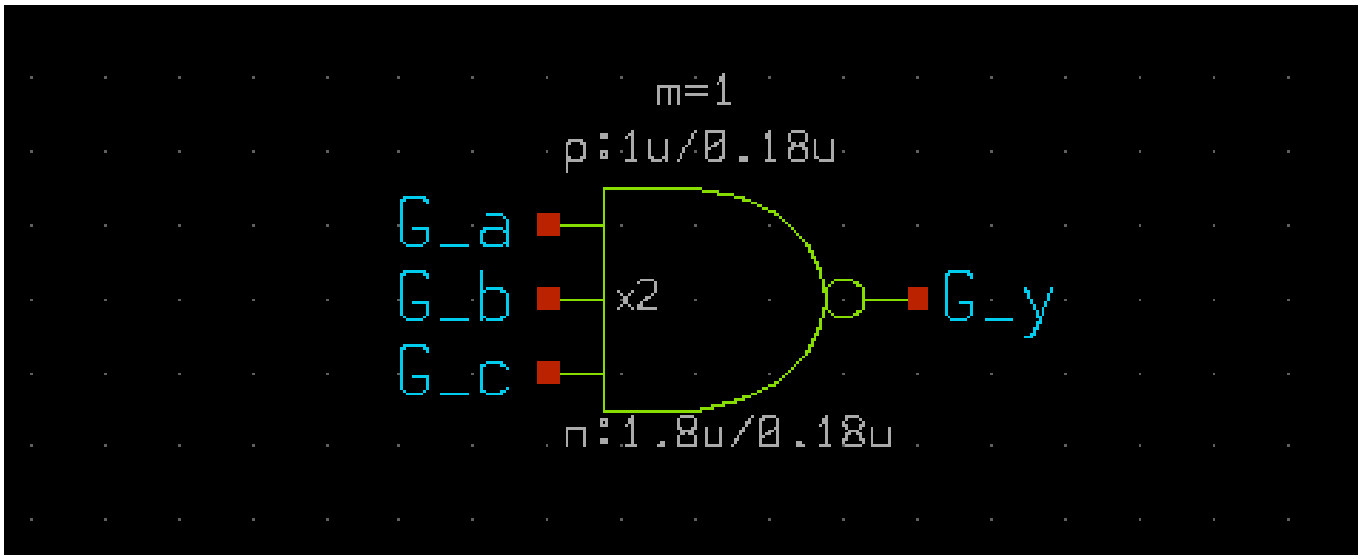



the resulting netlist is shown here, note that without the **generic_type** attribute the **irf5305** string would not be quoted.

```
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 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
**.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
```

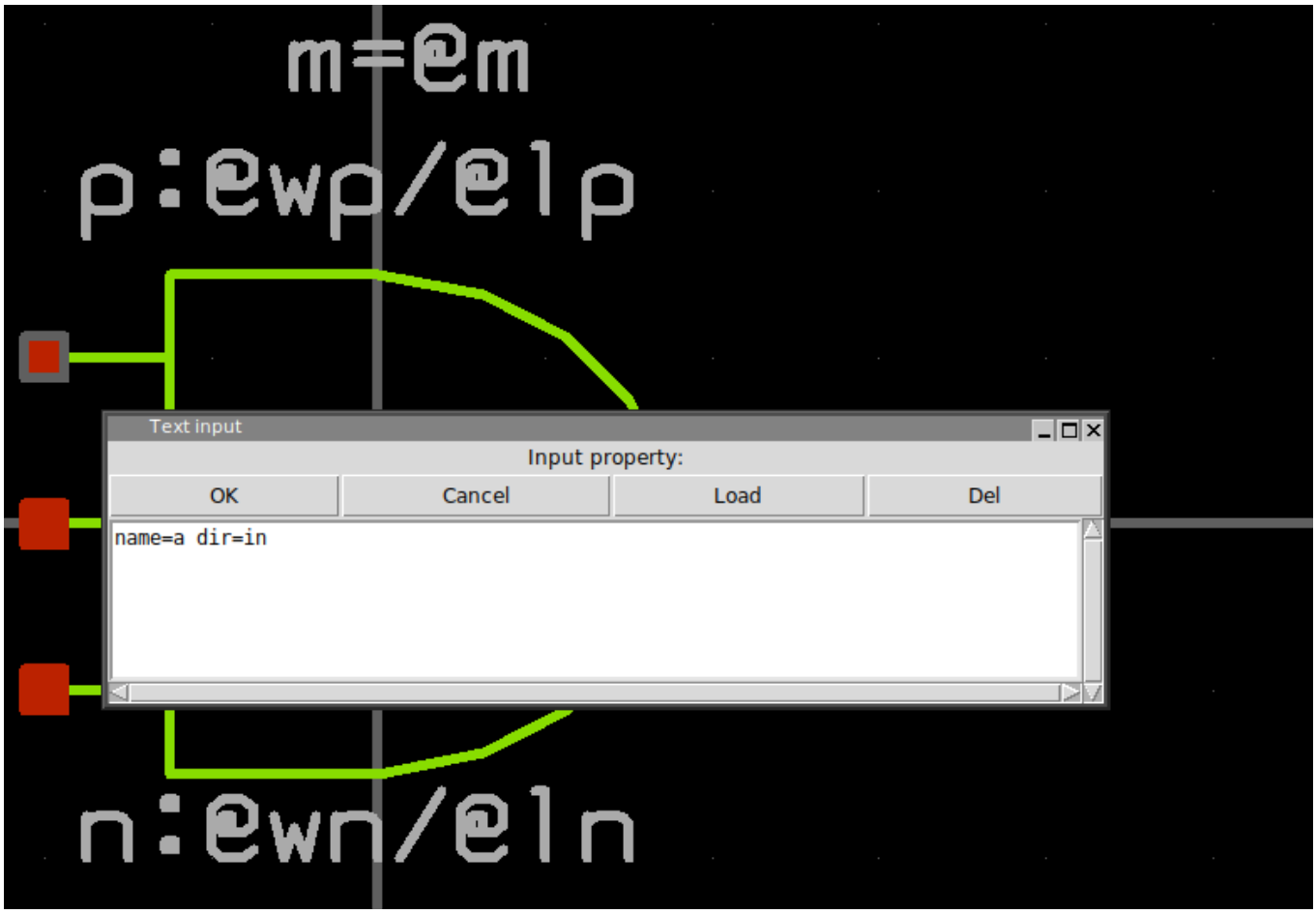
Without the **extra** property in the cmos gate symbol the following incorrect netlist will be produced:

```
**.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 and not as pins in the netlist.

- **dir**

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

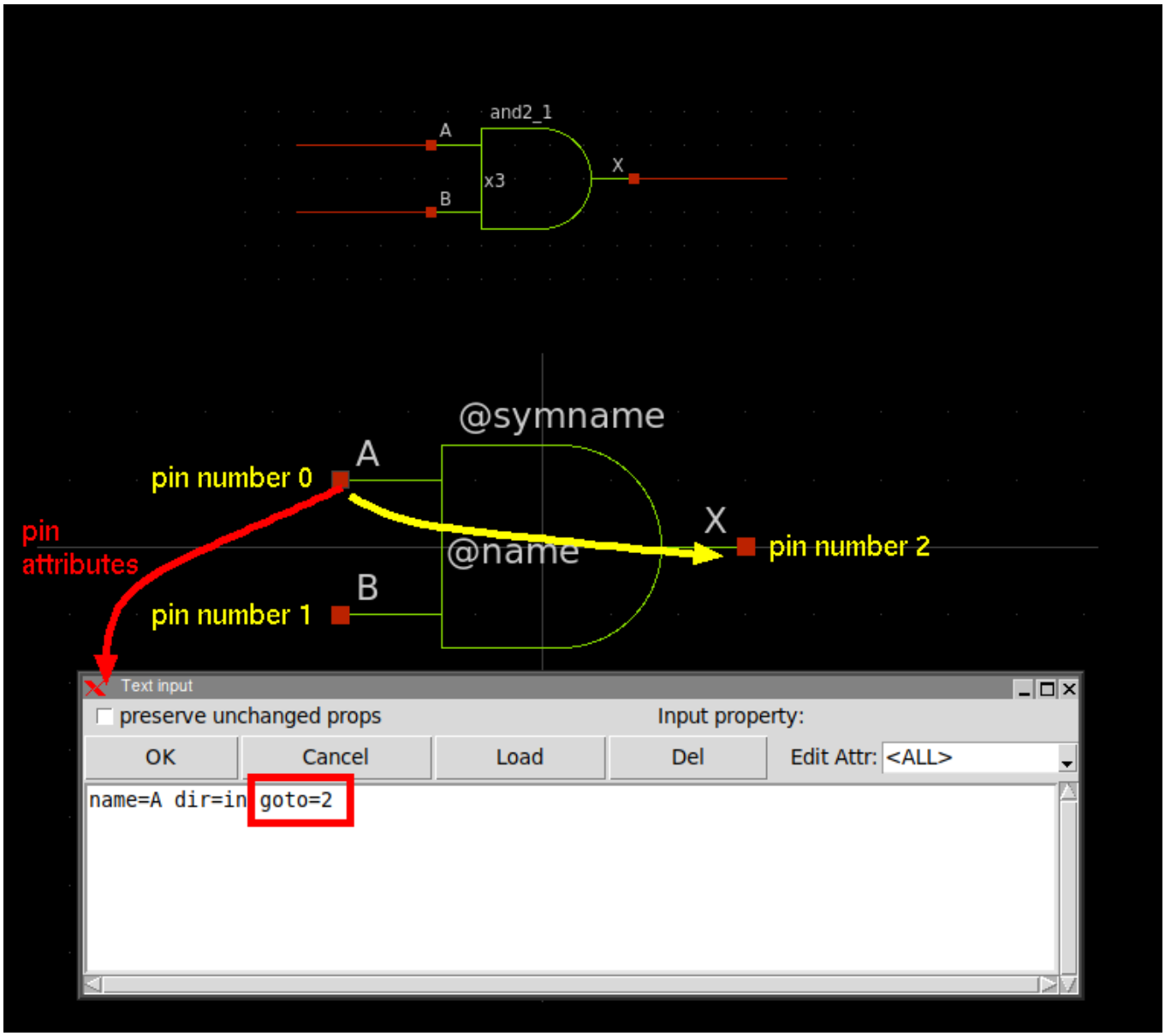



- **propag=n**

  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 'funtion**n**' for each **n** output pin. see 'function**n**' attribute for more info.



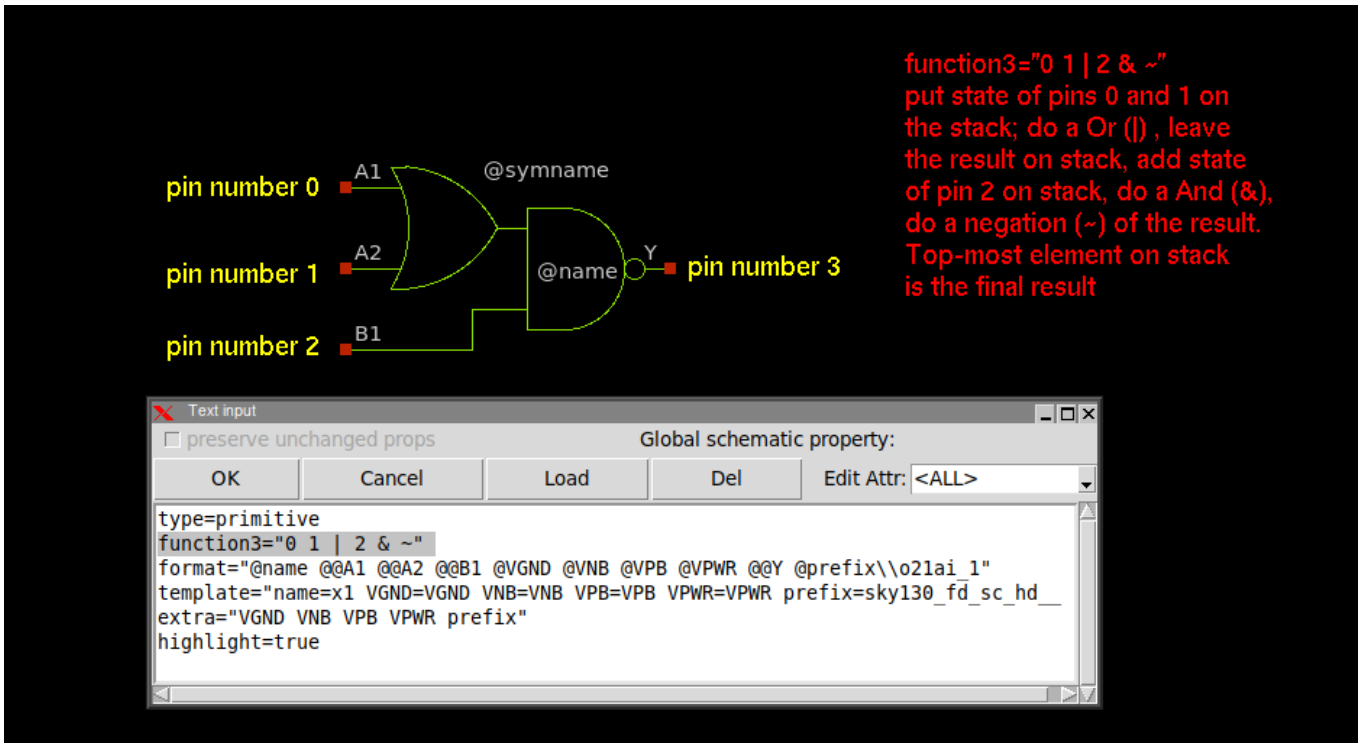

- **`clock=n`**

  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.



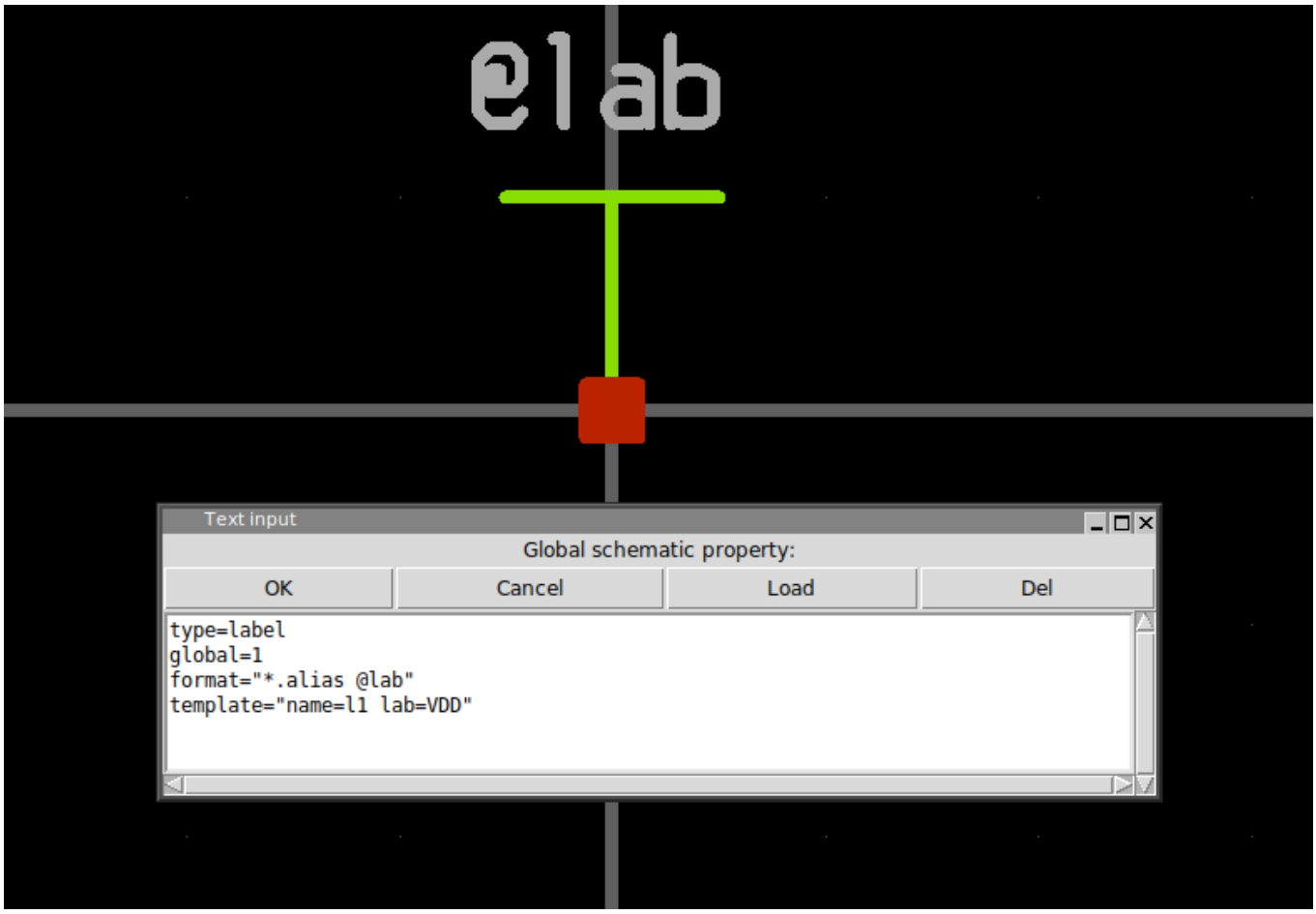

Commands that can appear in functions are:

- **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**: Exhanges 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.




- **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
  ```

## PREDEFINED SYMBOL VALUES

- **`@symname`**

  This expands to the name of the symbol

- **`@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**.

- **@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**

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

- **@prop_ptr**

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

- **@schprop**

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

- **@schvhdlprop**

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

- **@schverilogprop**

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

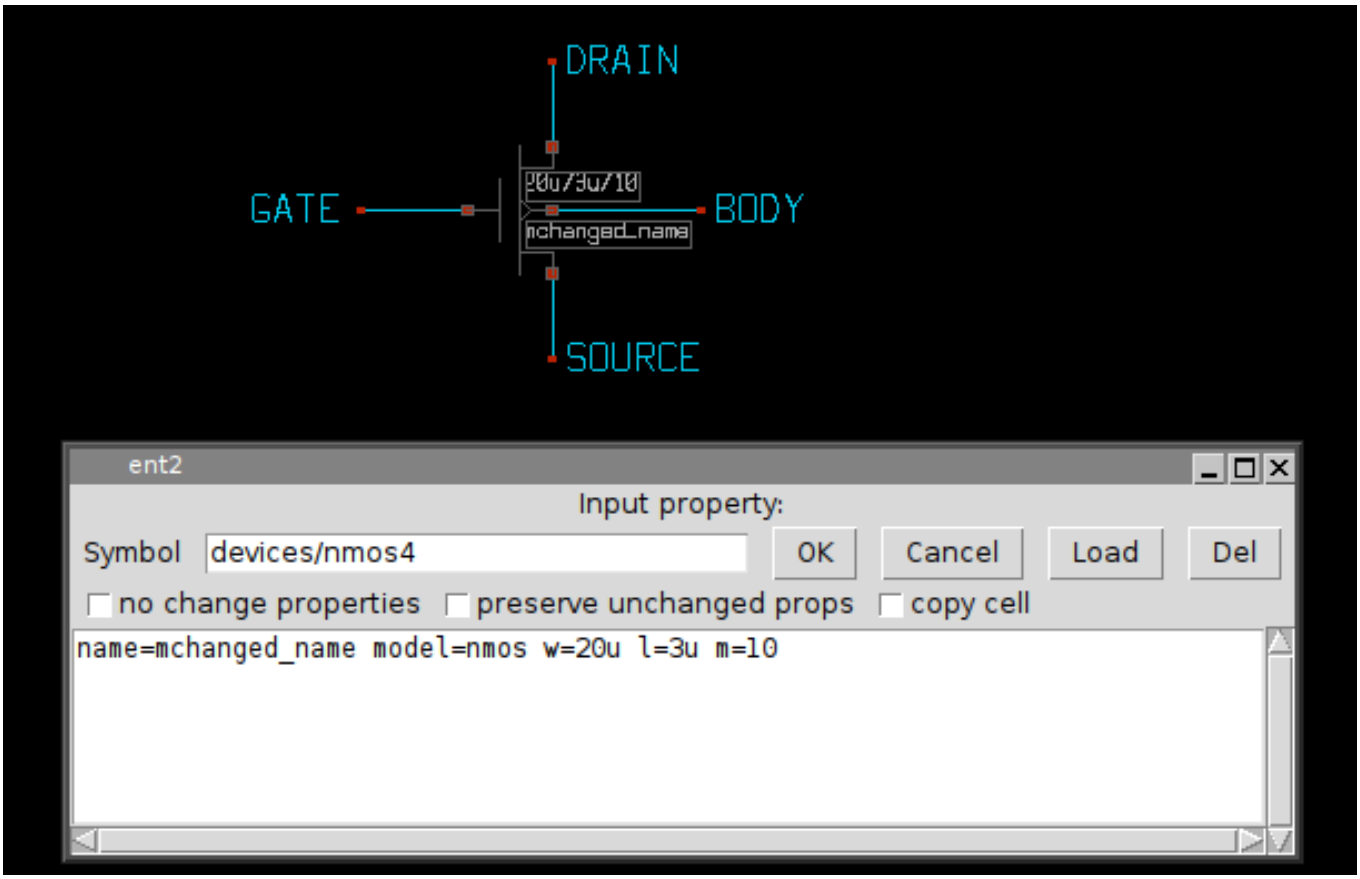
## TCL ATTRIBUTE SUBSTITUTION

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**




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 expecially 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**

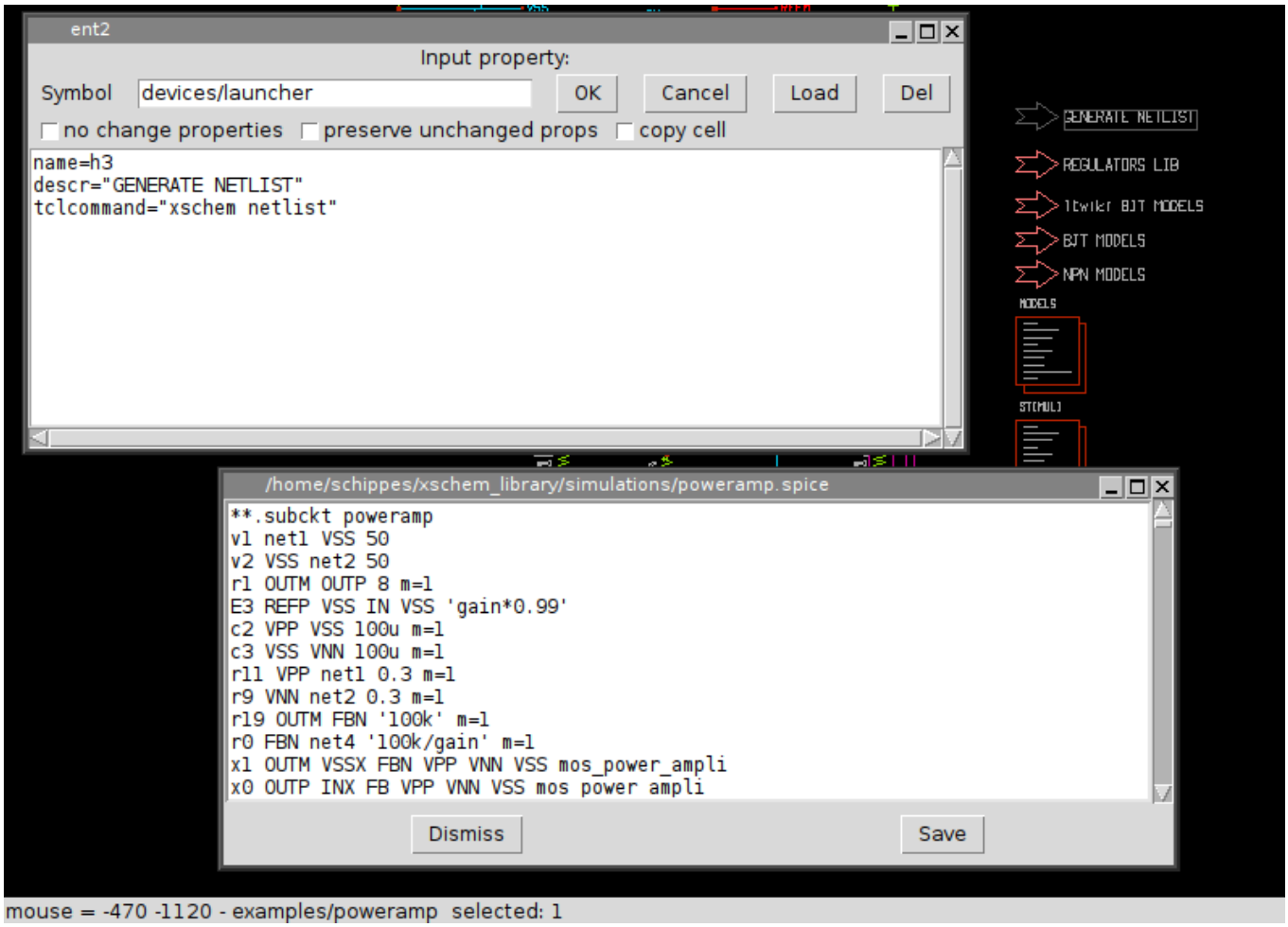
  This attribute defines a location (web page, file) that can be viewed when hitting the **<shift>H** key (or **<Alt> left mouse buttoni**) on a selected component. This is very useful to link a datasheet to a component, for example. The default program used to open the url is **xdg-open**. this can be 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**

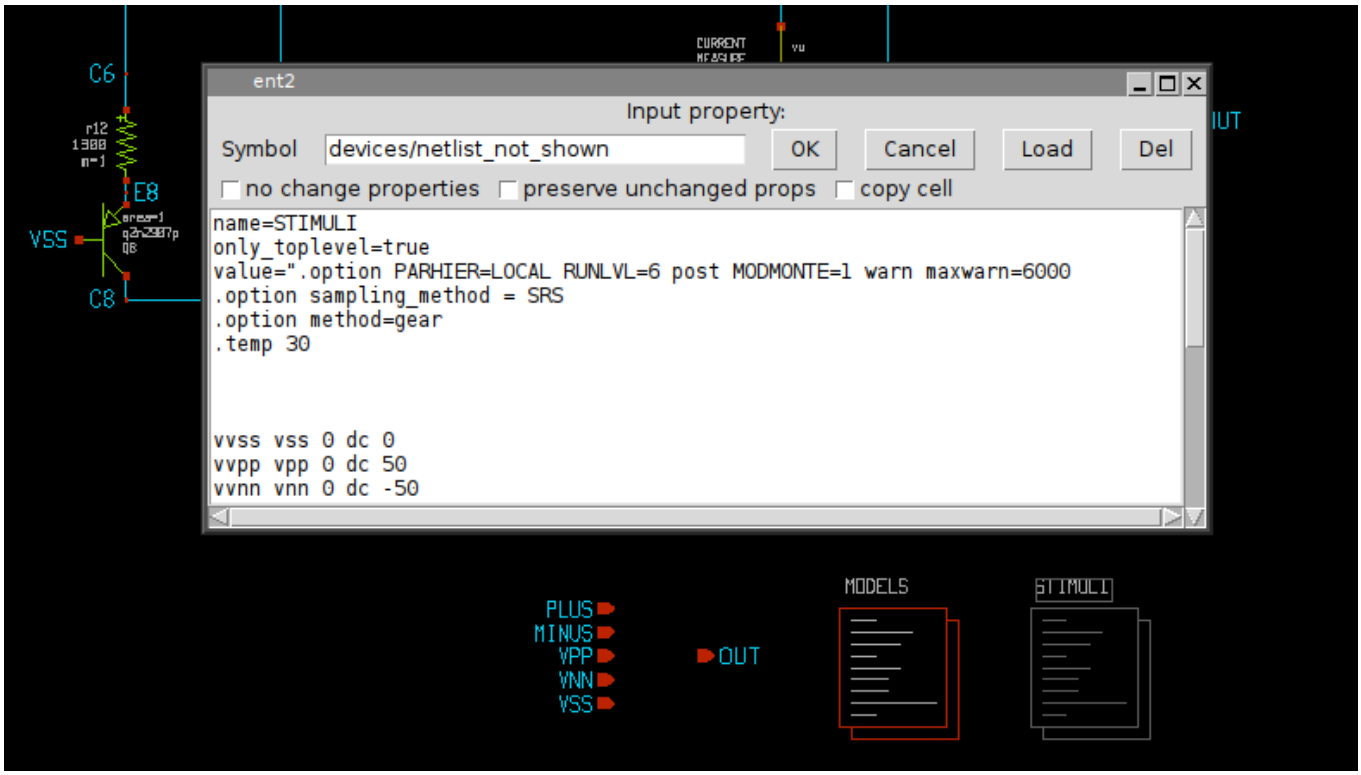
- **tclcommand**

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.




- **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 usefull for spice commands. Spice commands are placed in a special **netlist** component as we will see and are meaningfull 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.




- **`lock`**

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

- **`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.

- **`spice_ignore`**

  This tells XSCHEM that for SPICE netlist this component will be **completely** ignored.

- **`verilog_ignore`**

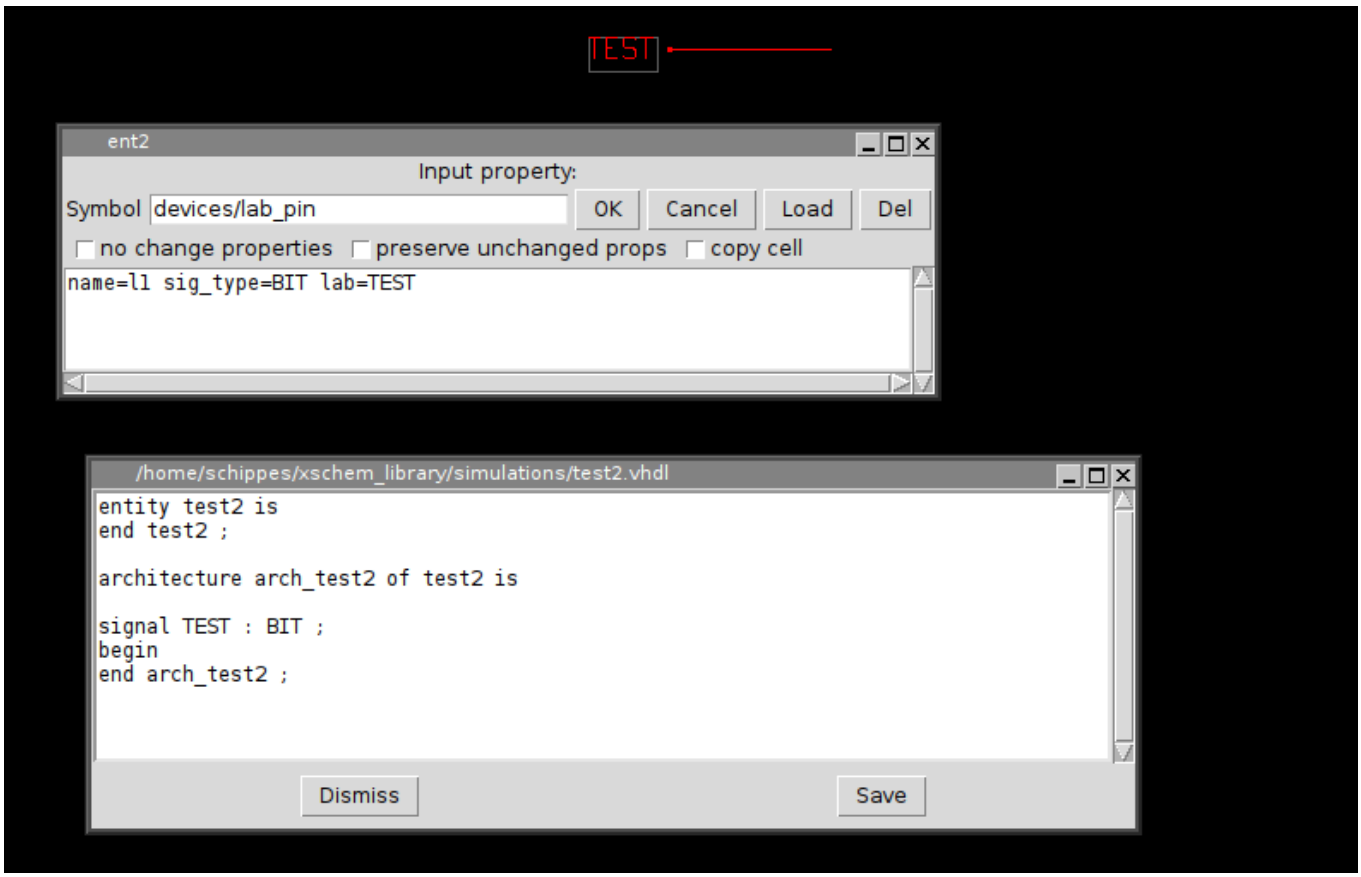
This tells XSCHEM that for Verilog netlist this component will be **completely** ignored.

- **vhdl_ignore**

  This tells XSCHEM that for VHDL netlist this component will be **completely** ignored.

- **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**.

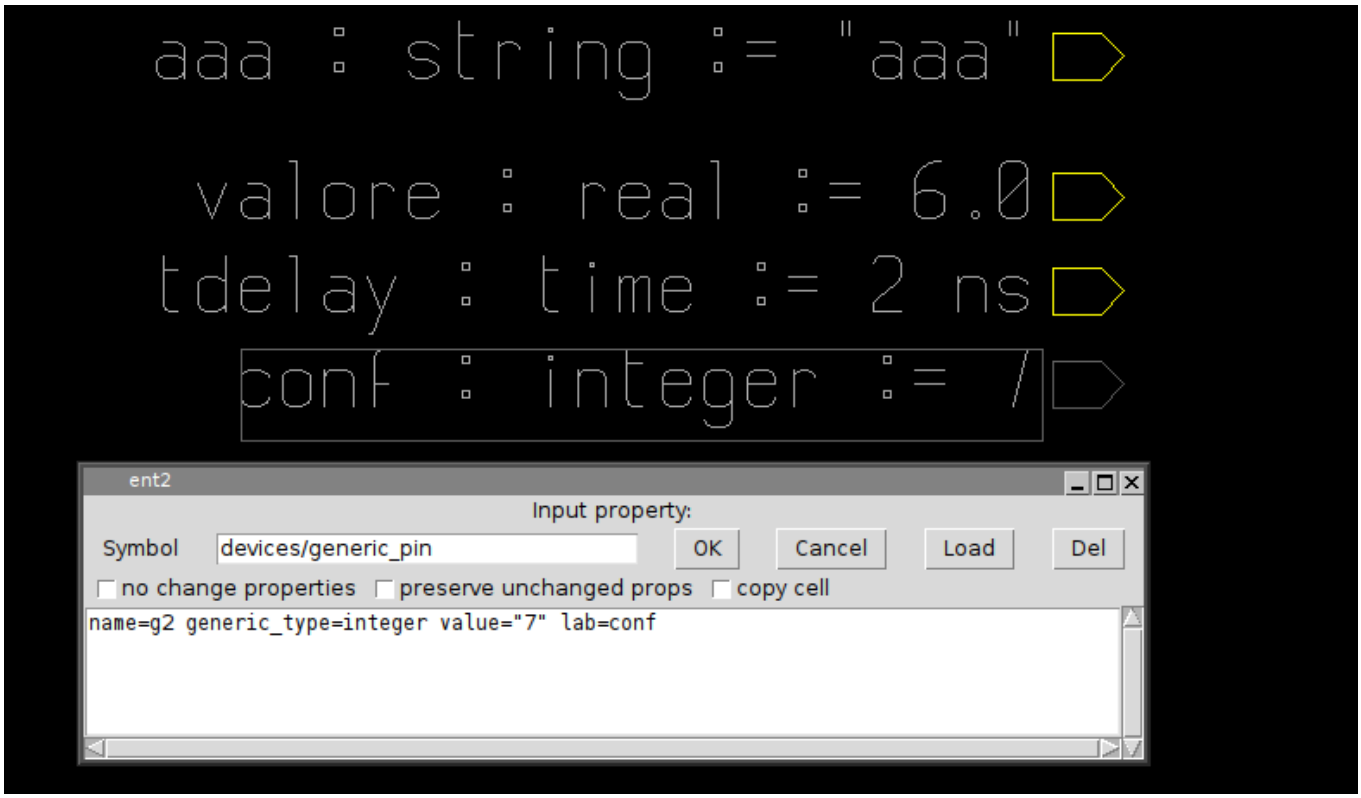



- **verilog_type**

  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:




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.

- **`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.

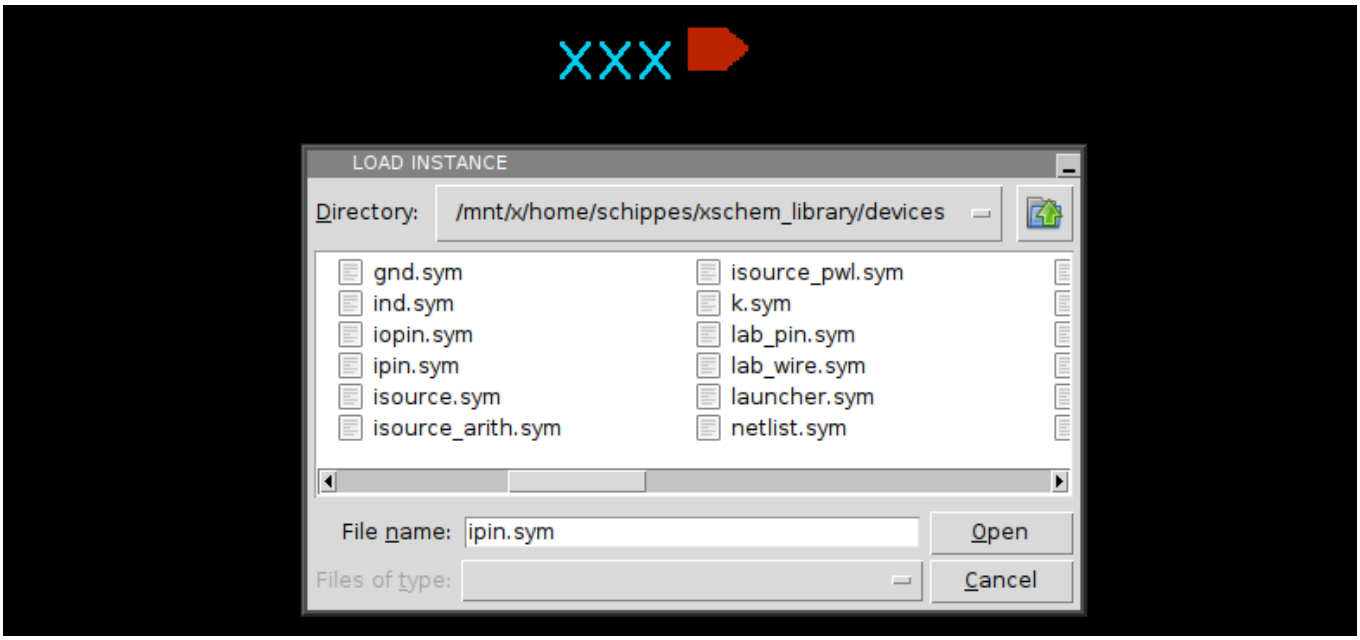
## TCL ATTRIBUTE SUBSTITUTION

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.

PREV UP NEXT

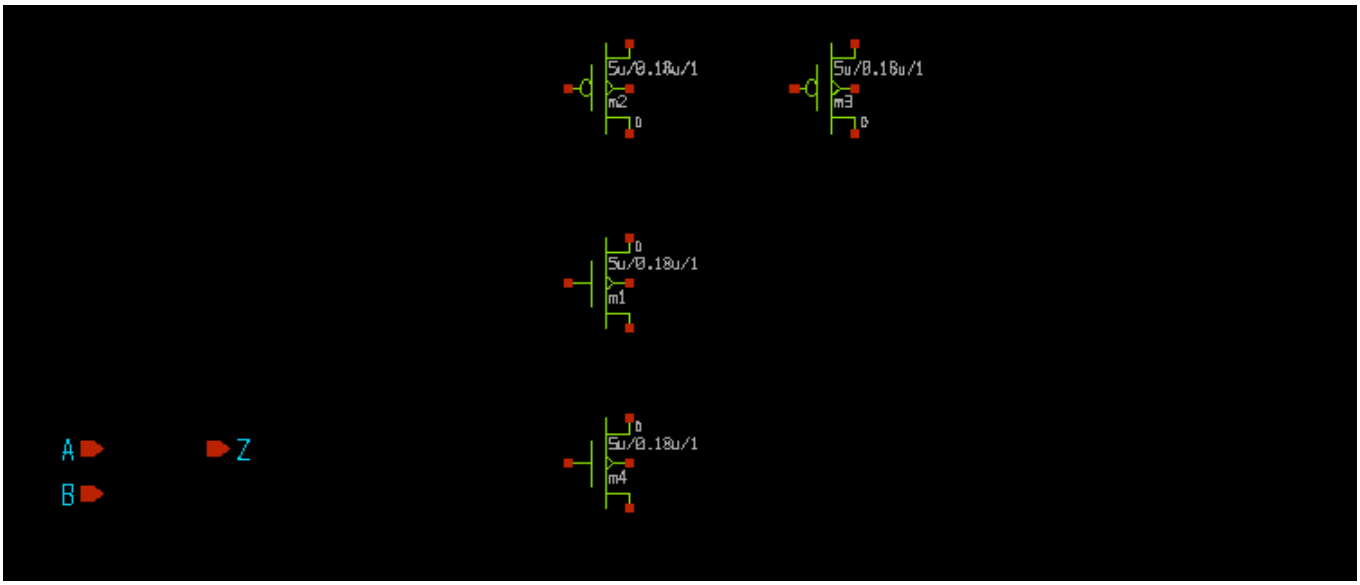
# CREATING A CIRCUIT SCHEMATIC

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'`**

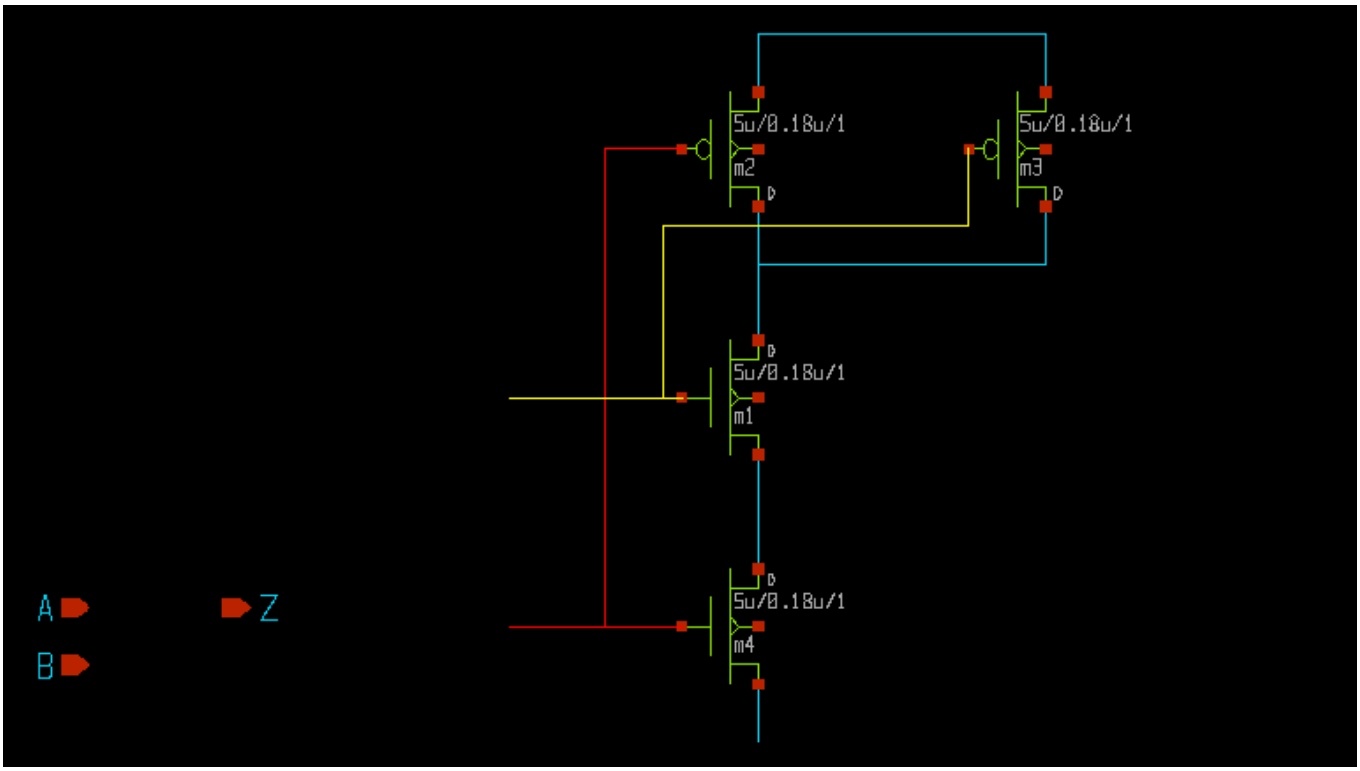



Copy another instance of it and set its lab attribute to **`B`**. Next place an output pin **`devices/opin.sym`** and set its lab to **`Z`**. The result will be as follows:

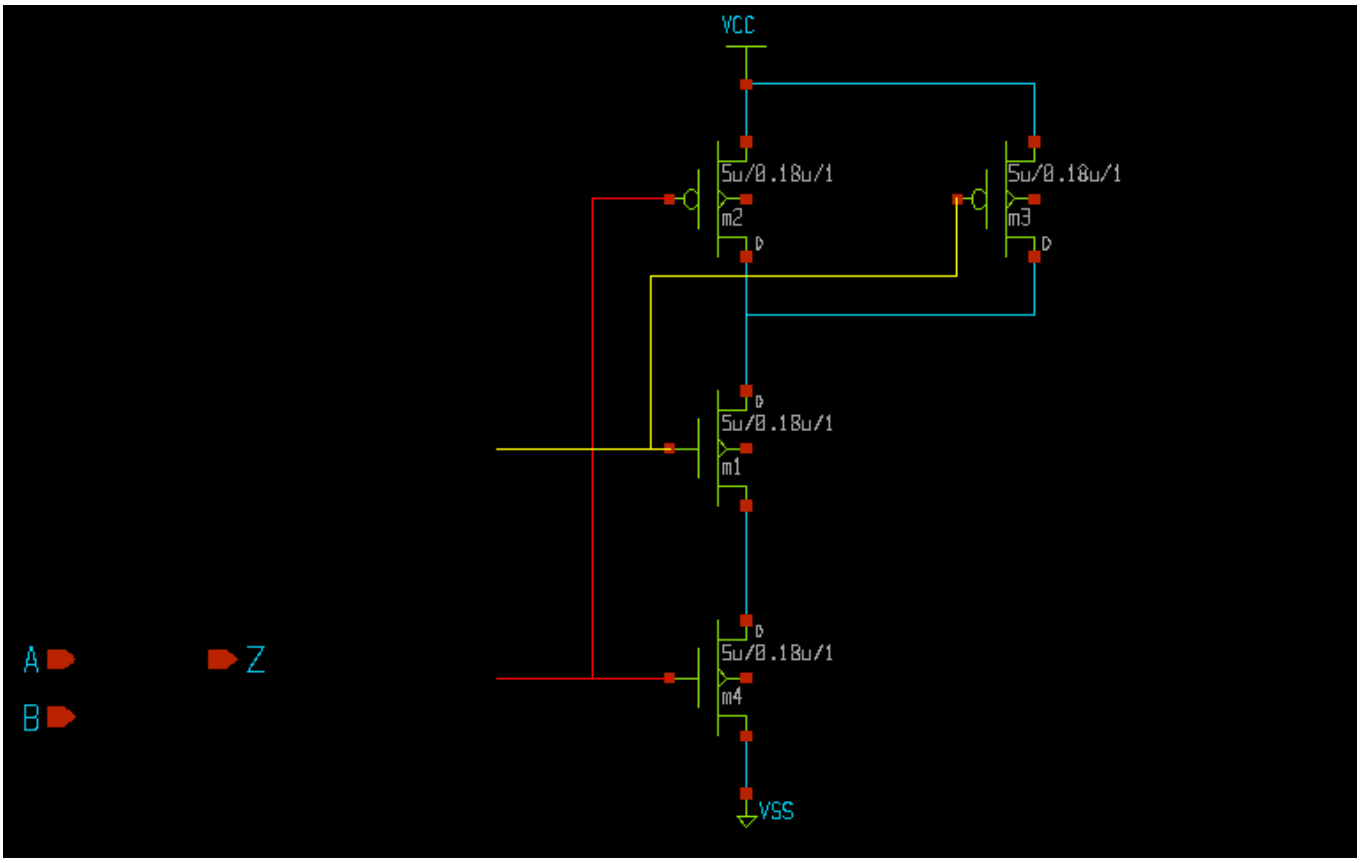

Now we need to build the actual circuit. Since we plan to do it in CMOS technology we need nmos and pmos transistors. Place one nmos from **`devices/nmos4.sym`** and one pmos from **`devices/pmos4.sym`** By selecting them with the mouse, moving (**`m`** bindkey), copying (**`'c'`** bindkey) place 4 transistors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



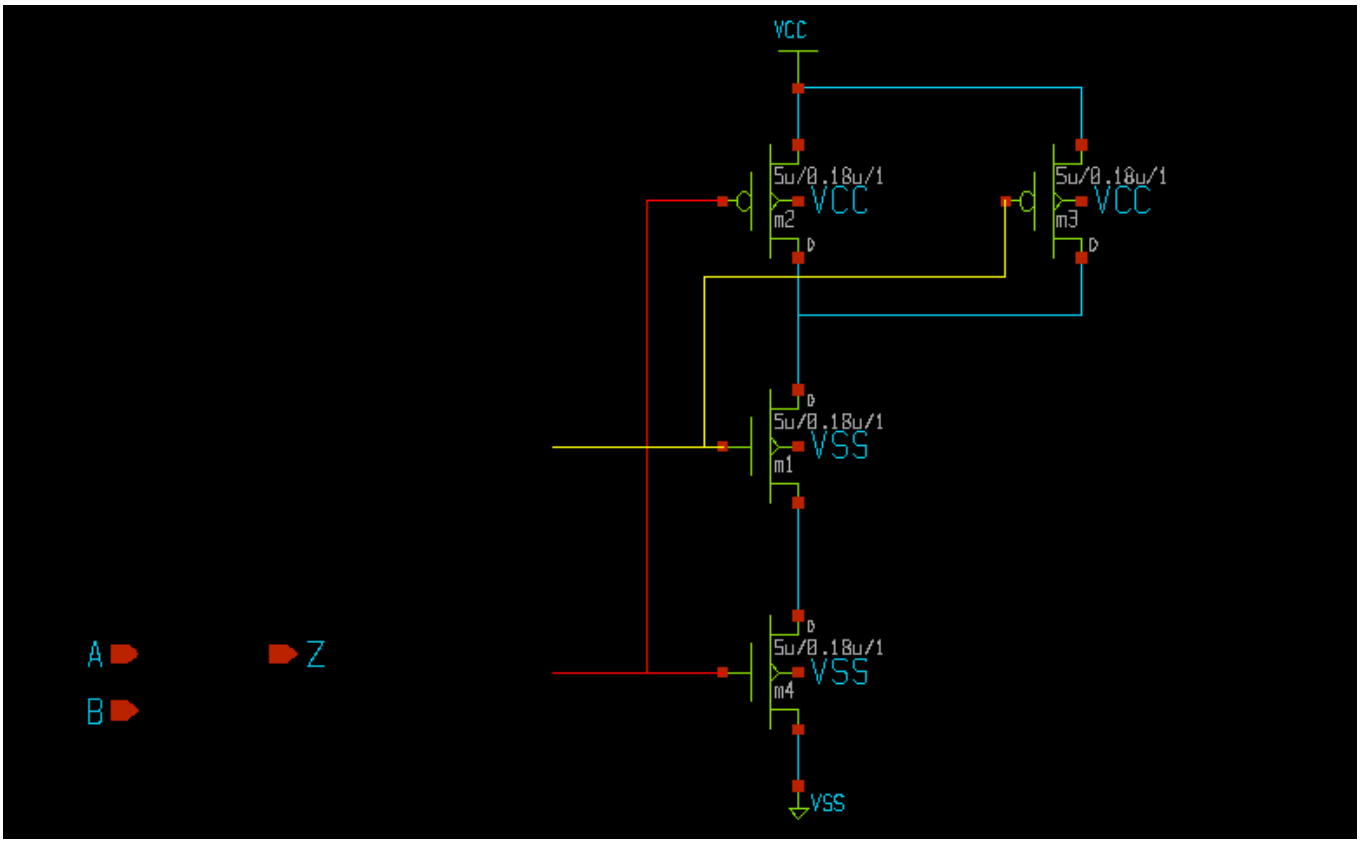

now draw wires to connect together the transistor to form a NAND gate; in the picture i have highlighted 2 electrical nodes by selecting one wire segment of each and pressing the **'k'** bindkey.



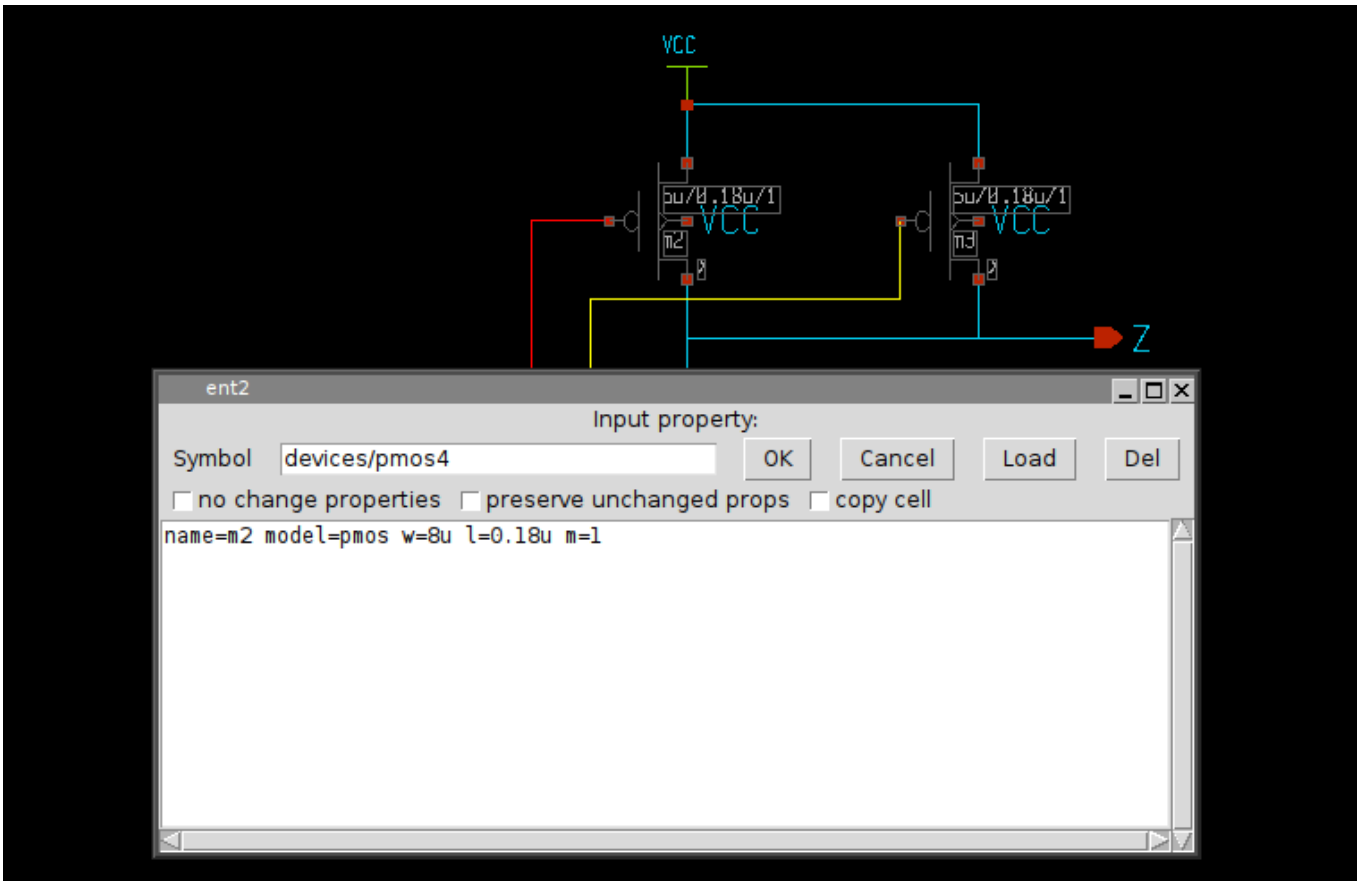

Next we need to place the supply nodes , VCC and VSS. we decide to use global nodes. Global nodes in SPICE semantics are like global variables in C programs, they are available everywhere, we do not need to propagate global nodes with pins. We could equally well use regular pins , as used for the A and 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.



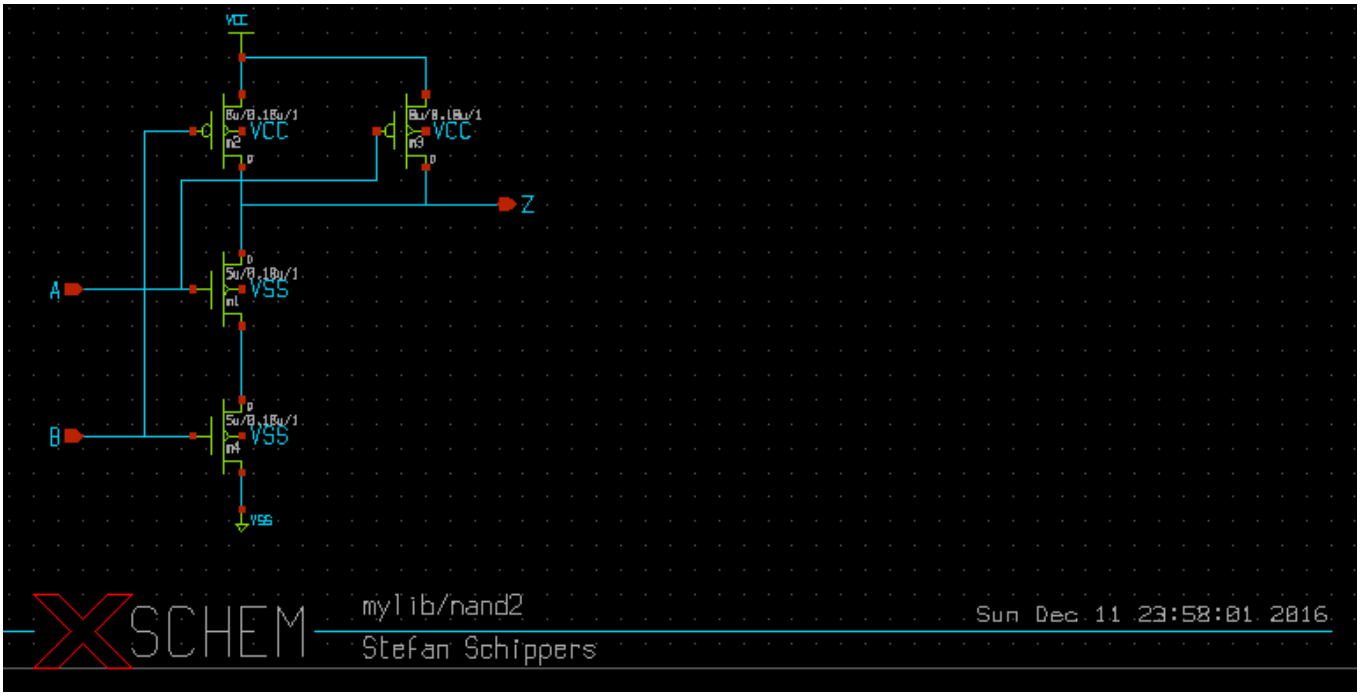

we still need to connect the body terminals of the mos transistors. One possibility is to hookup the two upper pmos transistor terminals to VCC with wires, and the two bottom nmos terminals to VSS with wires, but just to show different design styles i am planning to use "by name" connection with labels. 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 proprty **'q'** key; modify from 5u (default) to 8u.



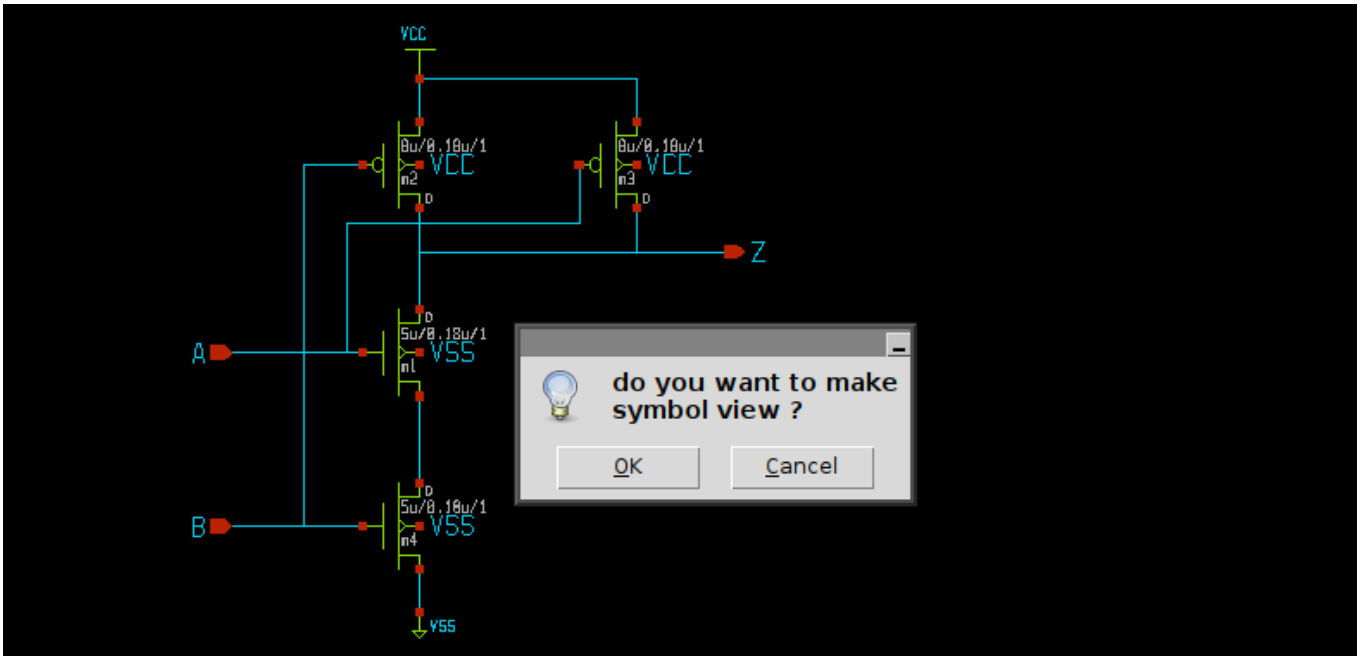

Now do a Save as operation, save it for example in **`mylib/nand2.sch`**.
To make the schematic nicer we also add the title component. This component is not netlisted but is useful, it reports the modification date and the author. Place the **`devices/title.sym`** component. The NAND gate is completed!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

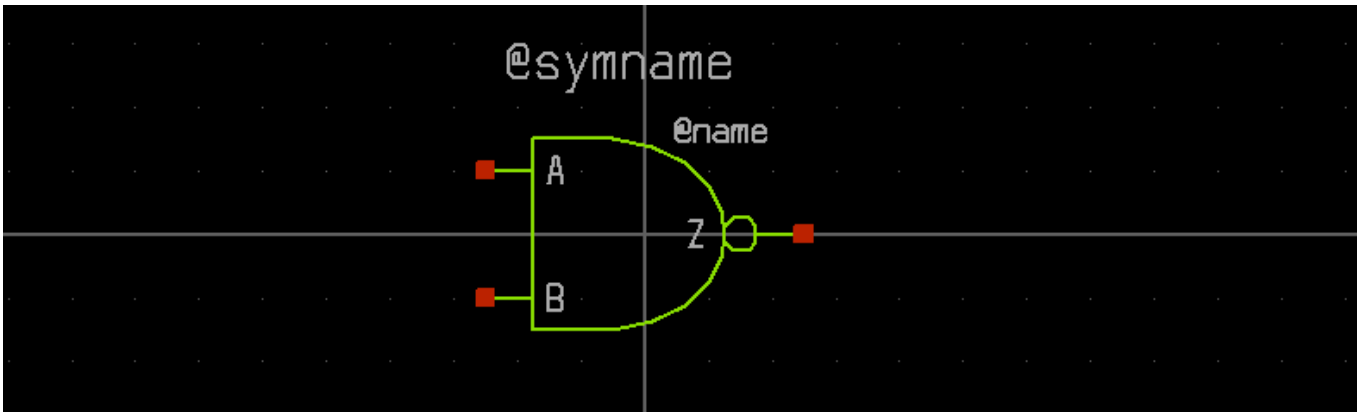
## Automatic symbol creation

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.

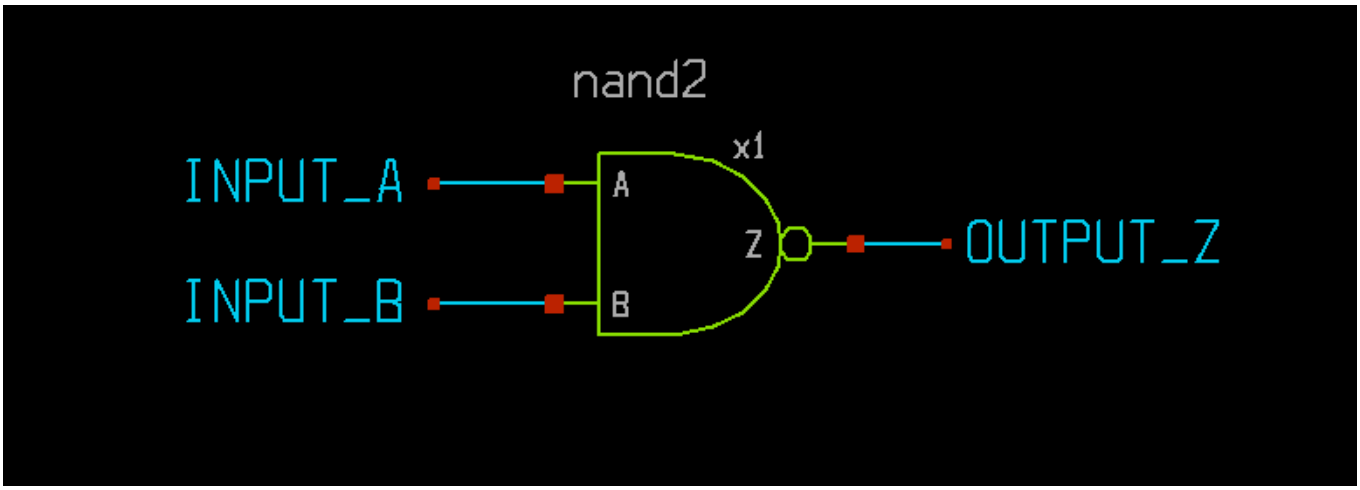



After pressing 'OK' a **mylib/nand2.sym** file is generated. try opening it (**File->Open**):


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 thresholf**) 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:

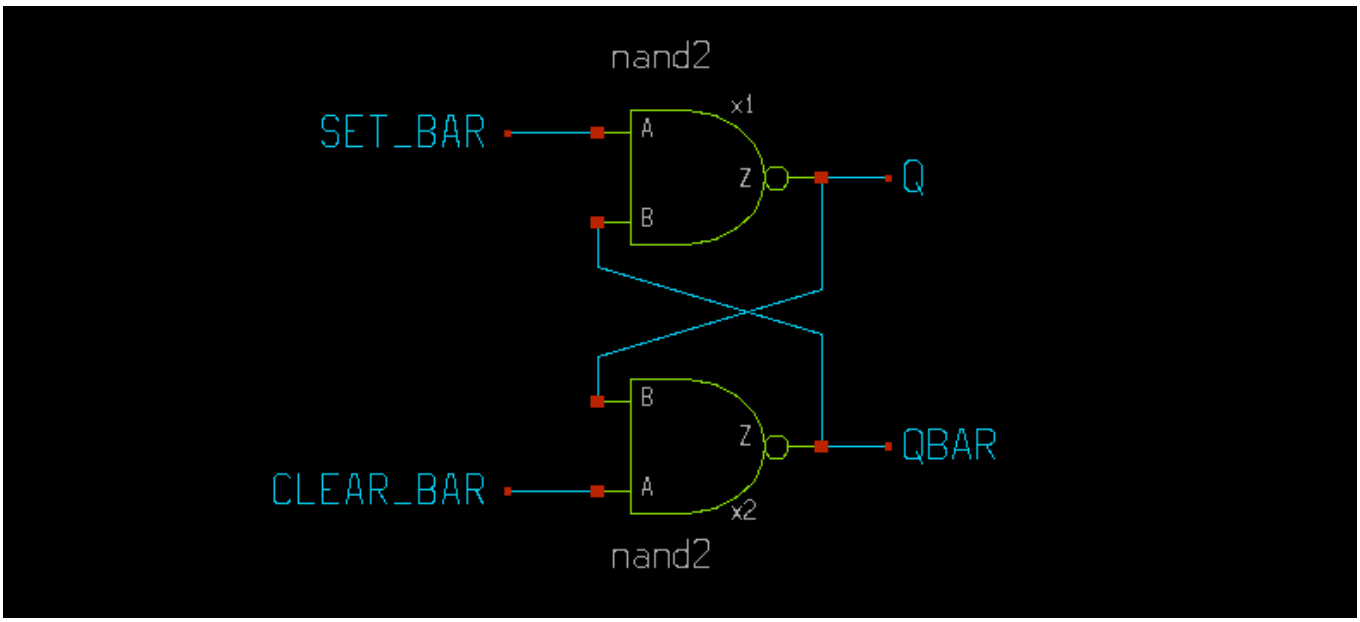



This is now a valid circuit. Let's test it by extracting the SPICE netlist. Enable the showing of netlist window (**`Options -> Show netlist win`**, or **`'A'`** key). Now extract the netlist (**`Netlist`** button on the right side of the menu bar, or **`'N'`** key). the SPICE netlist will be shown.

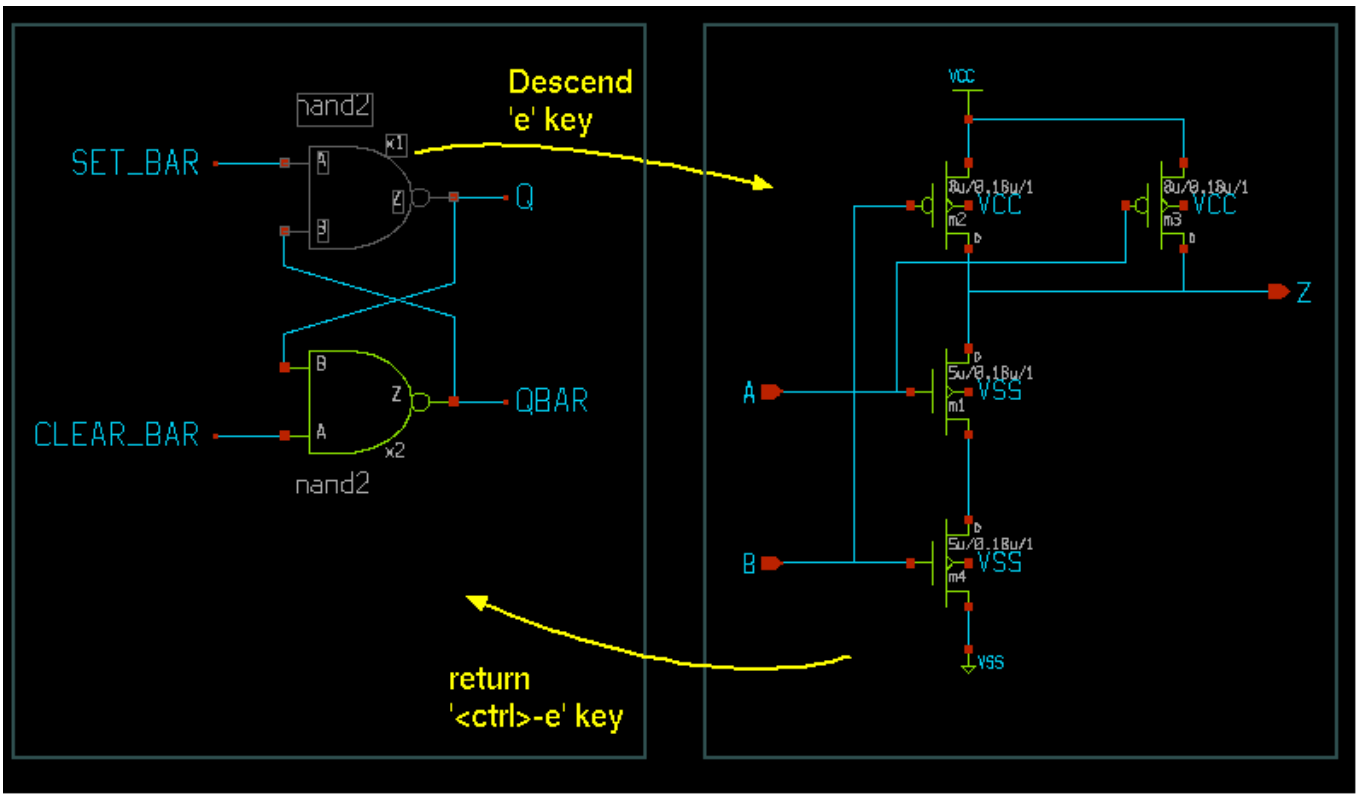
```
**.subckt test
x1 OUTPUT_Z INPUT_A INPUT_B nand2
**** begin user architecture code
**** end user architecture code
**.ends

* 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.



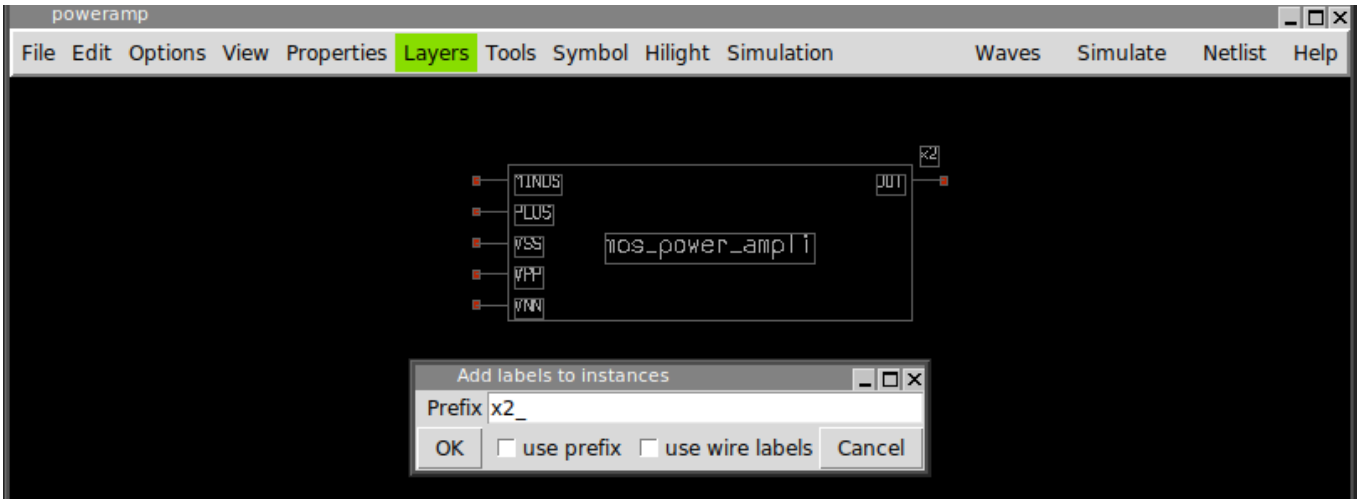
This is the corresponding netlist:

```
**.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.

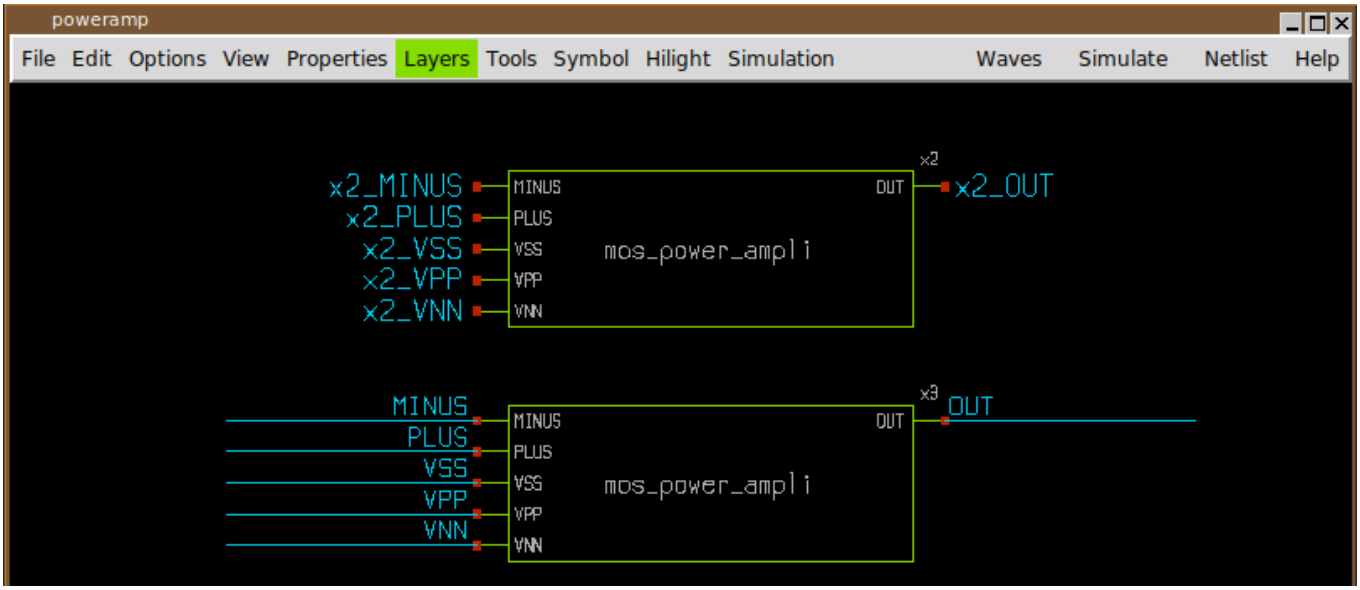
## Automatic Component Wiring

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.

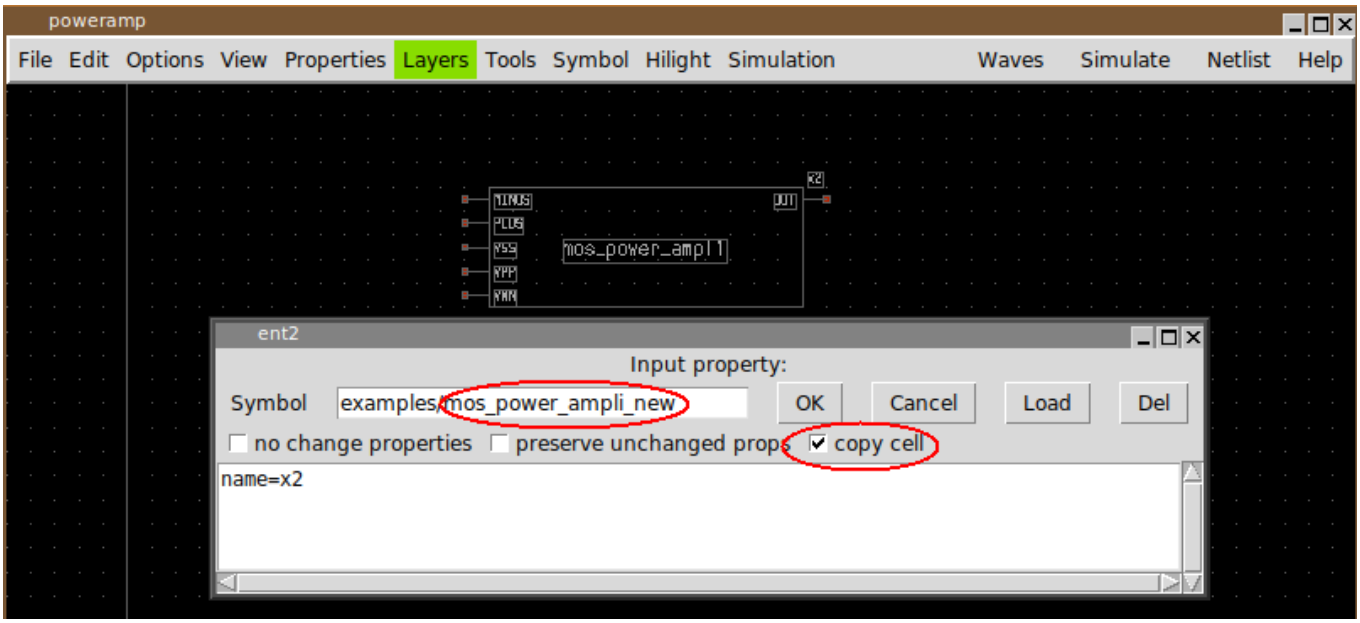


PREV UP NEXT

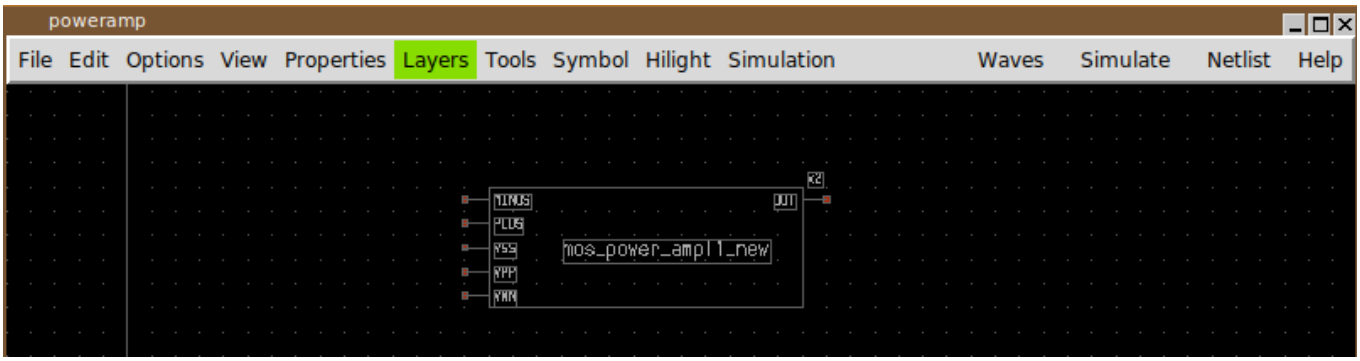
# CREATING SYMBOLS

## creating a new symbol and schematic by cloning

A 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:




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.



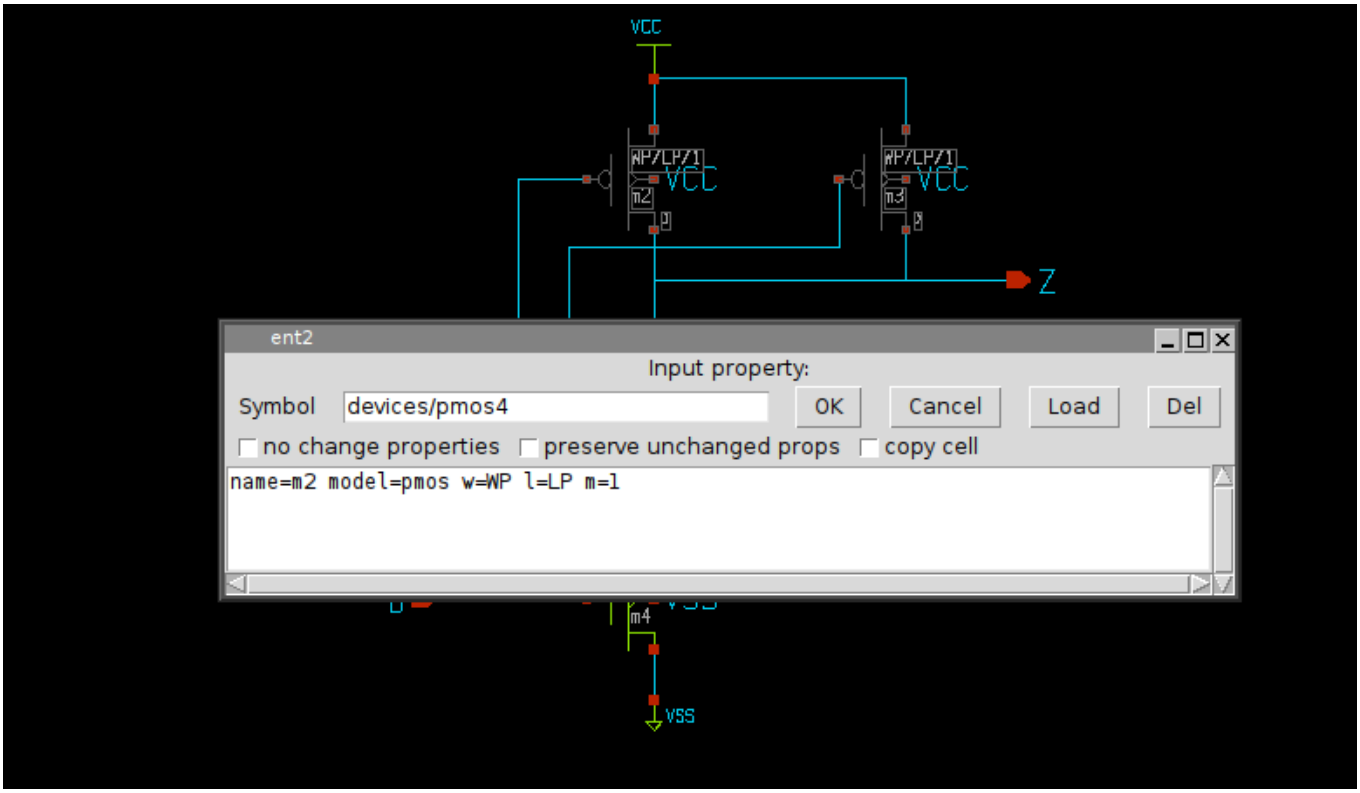

for more info on symbols see the Tutorial

PREV UP NEXT

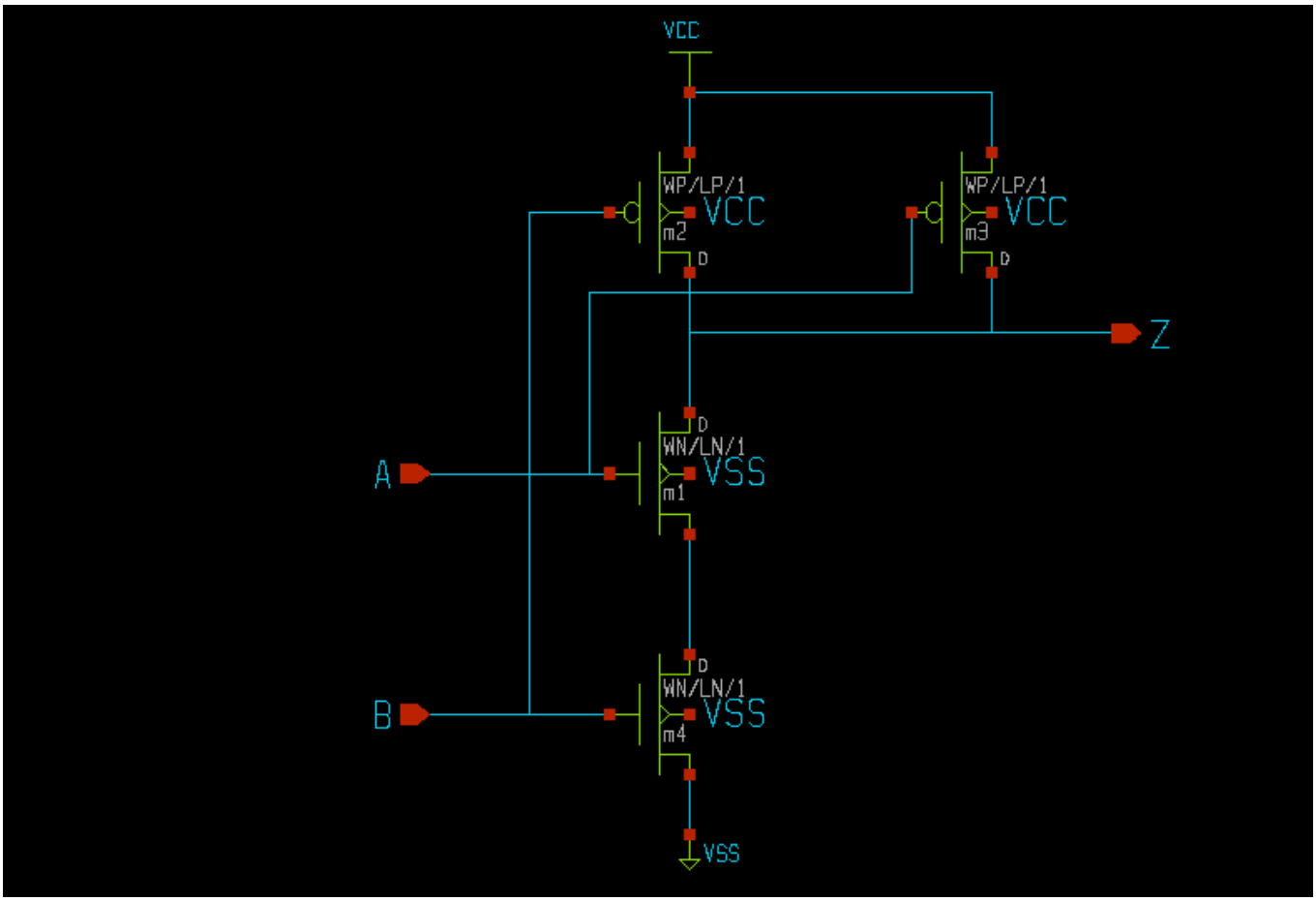
# COMPONENT PARAMETERS

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.

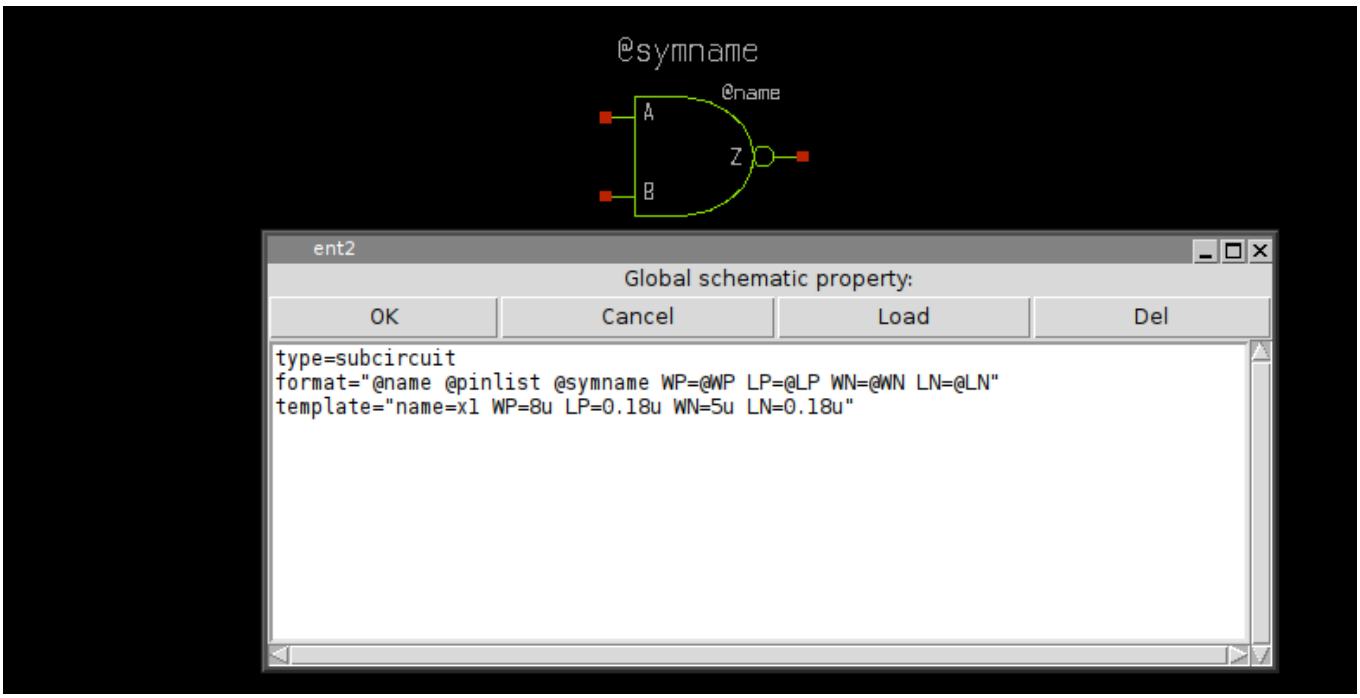
TIP: you can select two PMOS at the same time by clicking the second one with the **`shift`** key pressed, so with edit property **`'q'`** key you will change properties for both.



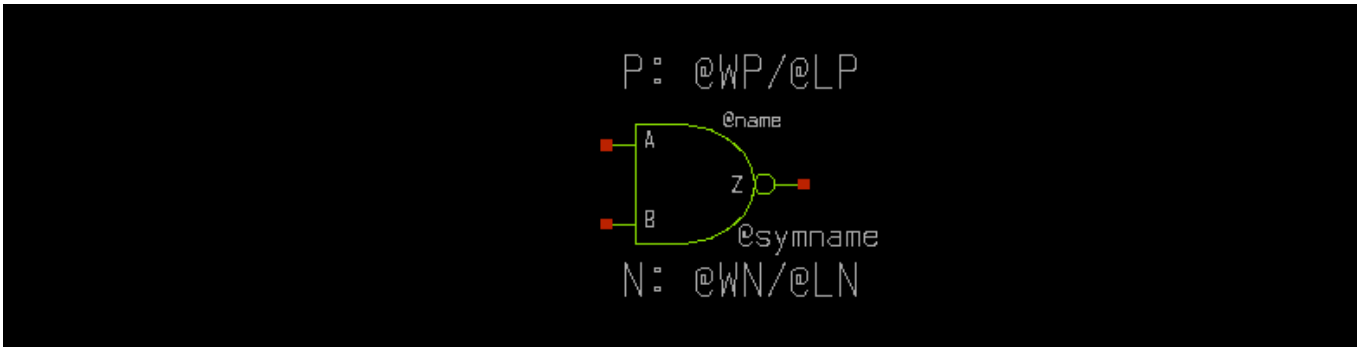

By doing the same for the NMOS transistors we end up with a schematic with fully parametrized transistor geometry.



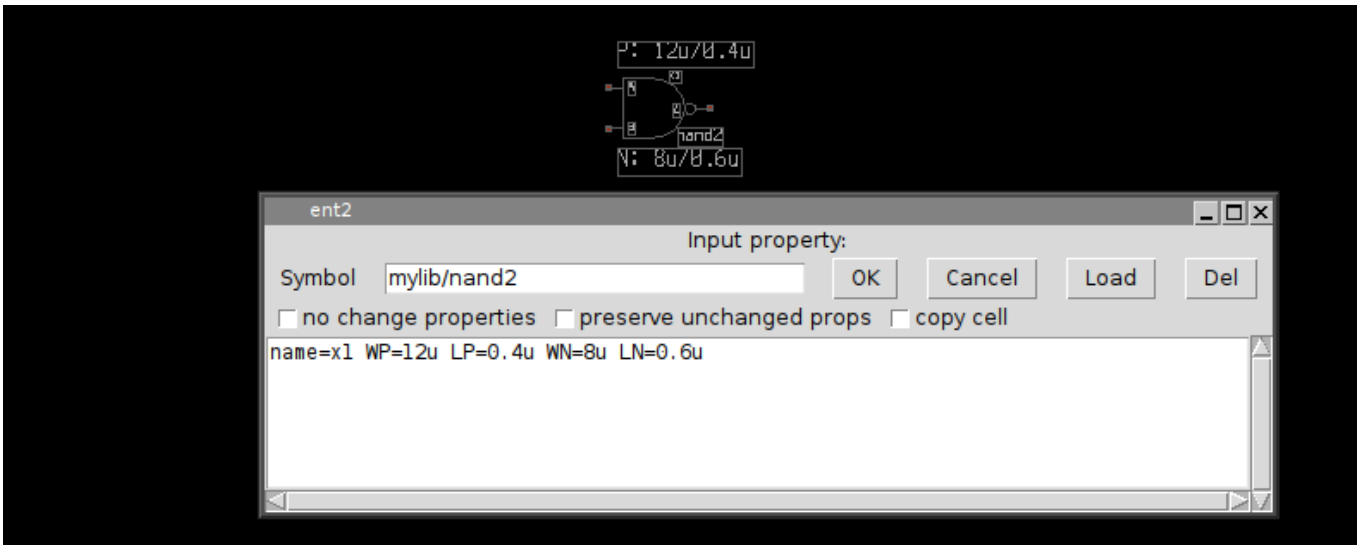

Now we have to change the **`mylib/nand2.sym`** symbol. Save the changes in the nand2 schematic (**`<shift>S`**) and load (**`Ctrl-o`**) the nand2 symbol. without selecting anything hit the **`'q'`** key to edit the symbol global property string. make the changes as shown in the picture.



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:

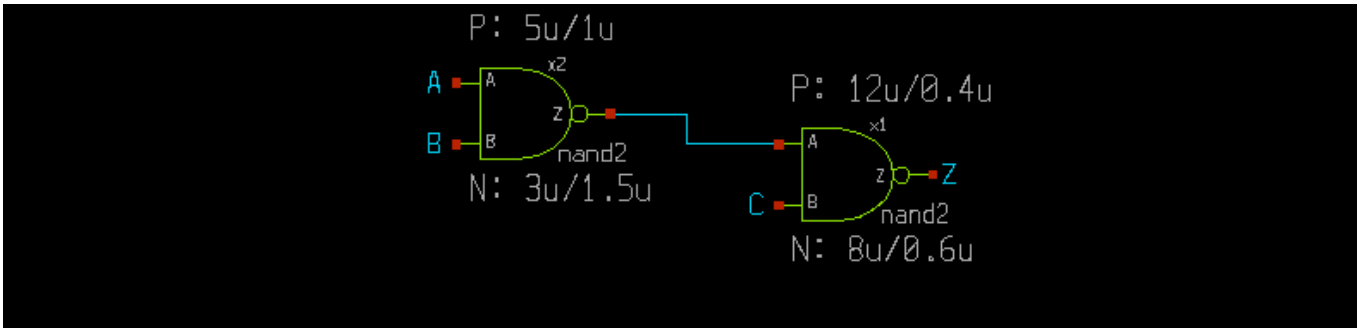



Now close the modified symbol saving the changes. Let's test the placement of the new modified symbol. Start a new schematic (menu **`File -> New`**) and insert (**`Insert key`**) the NAND2 gate. by pressing **`'q'`** you are now able to speciify different values for the geometric parameters:




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.

PREV UP NEXT

# 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.

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

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

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

Shift + Right Button      Select all connected wires/labels/pins

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

Mouse Wheel               Zoom in / out

MidButton drag            Pan viewable area

Alt + LeftButton          Unselect selected object

Alt + LeftButton drag
                          Unselect objects by area

RightButton               Context menu
```

| | | |
|---|---|---|
| Shift + RightButton | | Select object under the mouse and if label/pin select attached nets |
| Ctrl + RightButton | | Select object under the mouse and if label/pin select attached nets up to net junctions |
| LeftButton Double click | | Terminate Polygon placement<br>Edit object attributes |

XSCHEM KEY BINDINGS

| | | |
|---|---|---|
| - | BackSpace | Back to parent schematic |
| - | Delete | Delete selected objects |
| - | Insert | Insert element from library |
| - | 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->0, 0->1, X->X |
| - | 'a' | Make symbol from pin list of current schematic |
| ctrl | 'a' | Select all |
| shift | 'A' | Toggle show netlist |
| - | 'b' | Merge file |
| ctrl | 'b' | Toggle show text in symbol |
| alt | 'b' | Toggle show symbol details / only bounding boxes |
| - | 'c' | Copy selected obj. |
| 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' | Flip |
| alt | 'f' | Flip 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 |
| - | '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 obj. |
| shift | 'N' | Top level only netlist |
| - | 'n' | Hierarchical Netlist |
| ctrl | 'n' | New schematic |
| ctrl+shift | 'N' | New symbol |
| alt | 'n' | Empty schematic in new window |
| alt+shift | 'N' | Empty symbol in new window |
| shift | 'O' | Toggle Light / Dark colorscheme |
| ctrl | 'o' | Load schematic |
| 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 |
| ctrl | 's' | Save schematic |
| alt | 's' | Reload current schematic from disk |
| ctrl+alt | 's' | Save-as symbol |
| - | 't' | Place text |
| alt | 'u' | Align to current grid selected objects |
| shift | 'U' | Redo |
| - | 'u' | Undo |
| - | 'v' | Constrained vertical move/copy of objects |
| ctrl | 'v' | Paste from clipboard |
| shift | 'V' | Toggle spice/vhdl/verilog netlist |
| - | 'w' | Place wire |
| ctrl | 'w' | Place polygon. Operation ends by placing last point over first. |
| 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 |
| - | '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
```

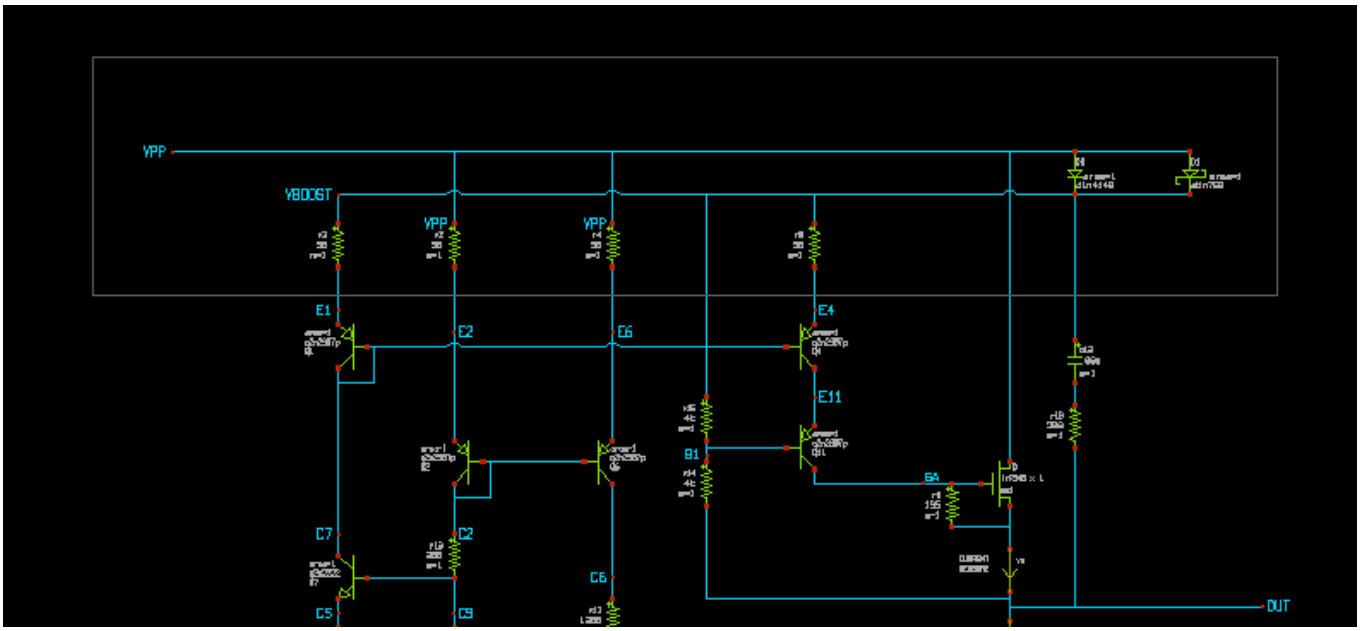
## KEYBIND CUSTOMIZATION

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

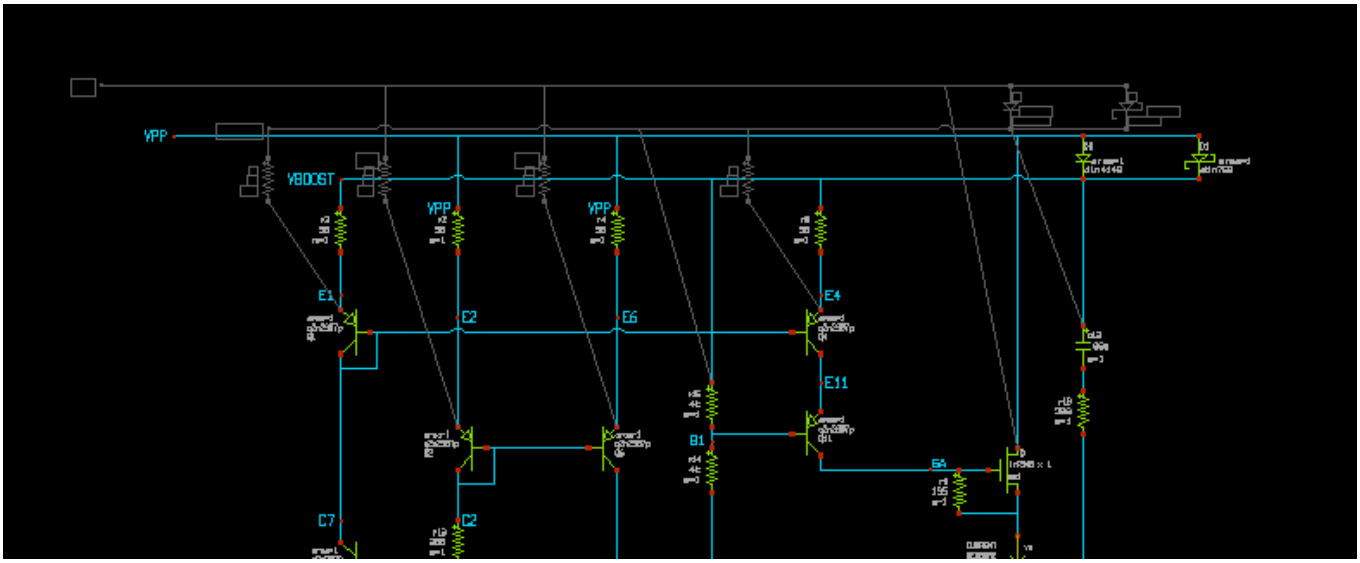
```
## replace Ctrl-d with Escape (so you won't kill the program :-))
set replace_key(Control-d) Escape
## swap w and W keybinds; Always specify Shift for capital letters
set replace_key(Shift-W) w
set replace_key(w) Shift-W
```

## STRETCH OPERATIONS

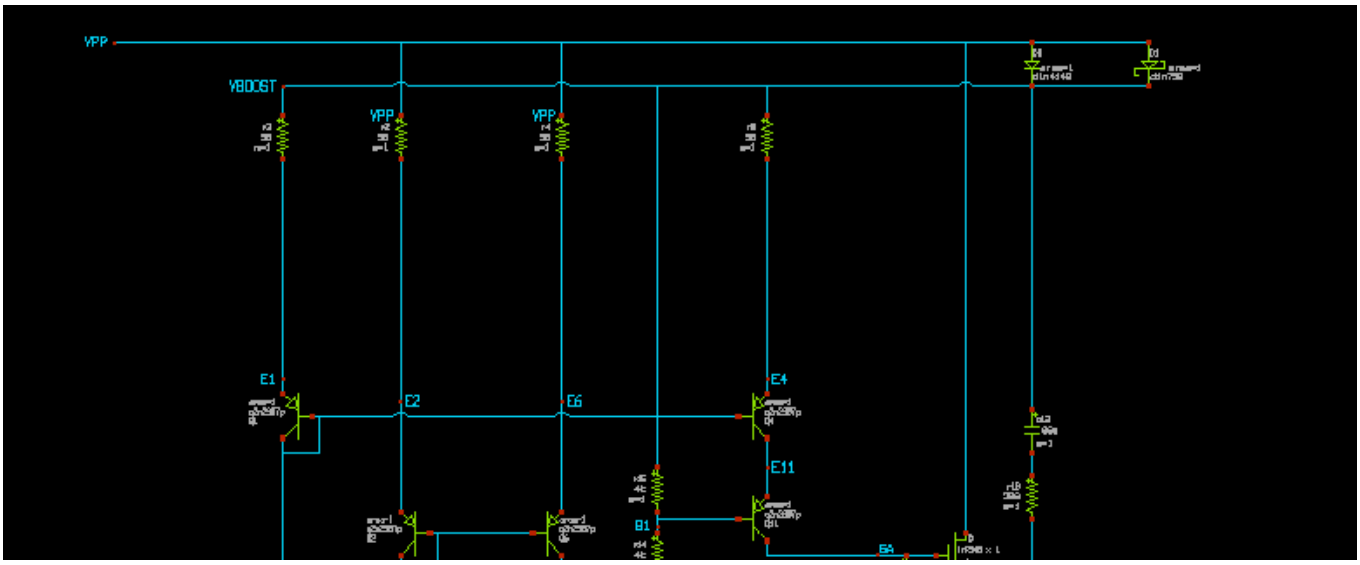
An important operation that deserves a special paragraph is the **Stretch** operation. There is frequently the need to move part of the circuit without breaking connections, for example to create more room for other circuitry or just to make it look better. The first thing to do is to drag a selection 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.

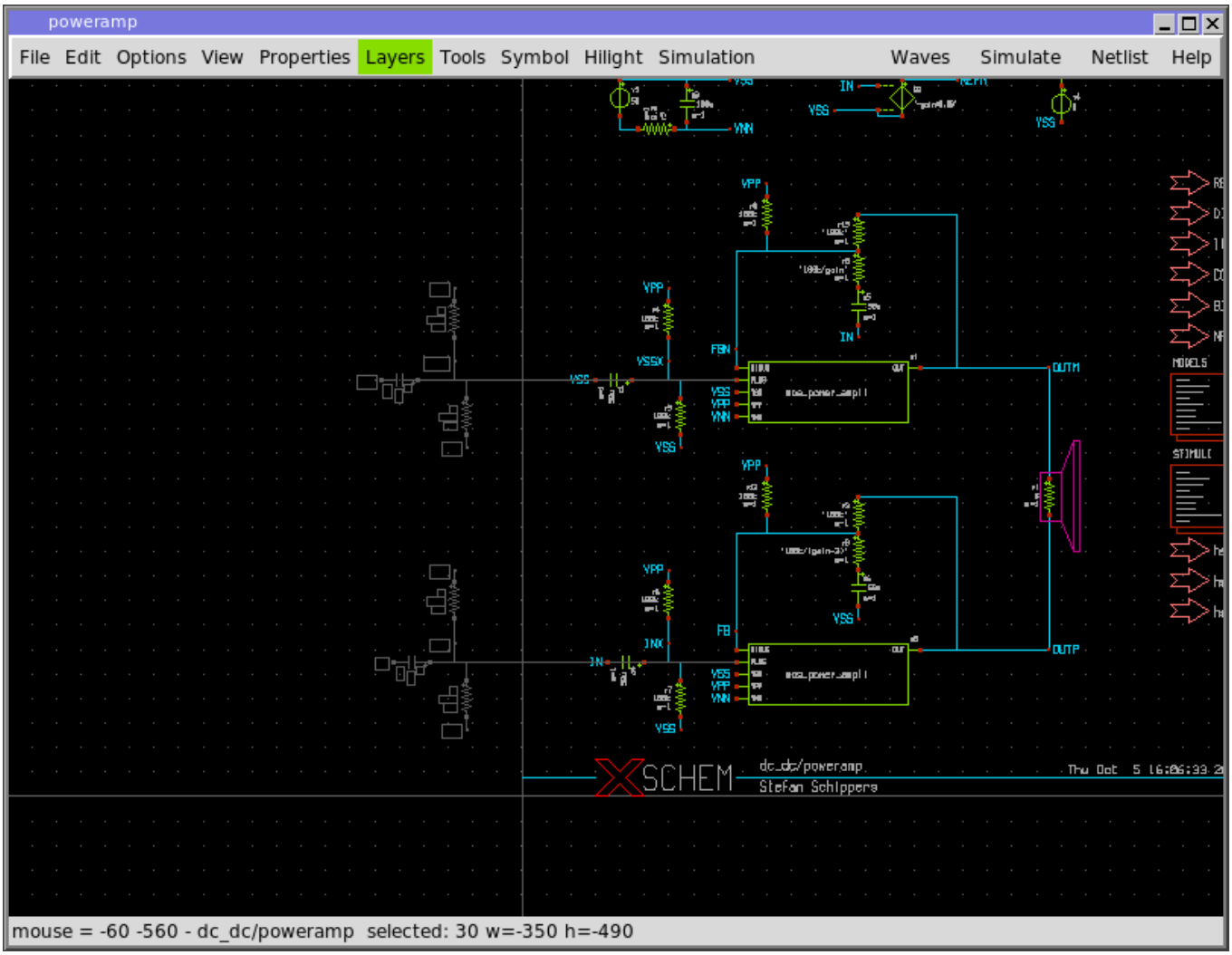



## PLACE WIRES SNAPPING TO CLOSEST PIN OT NET ENDPOINT

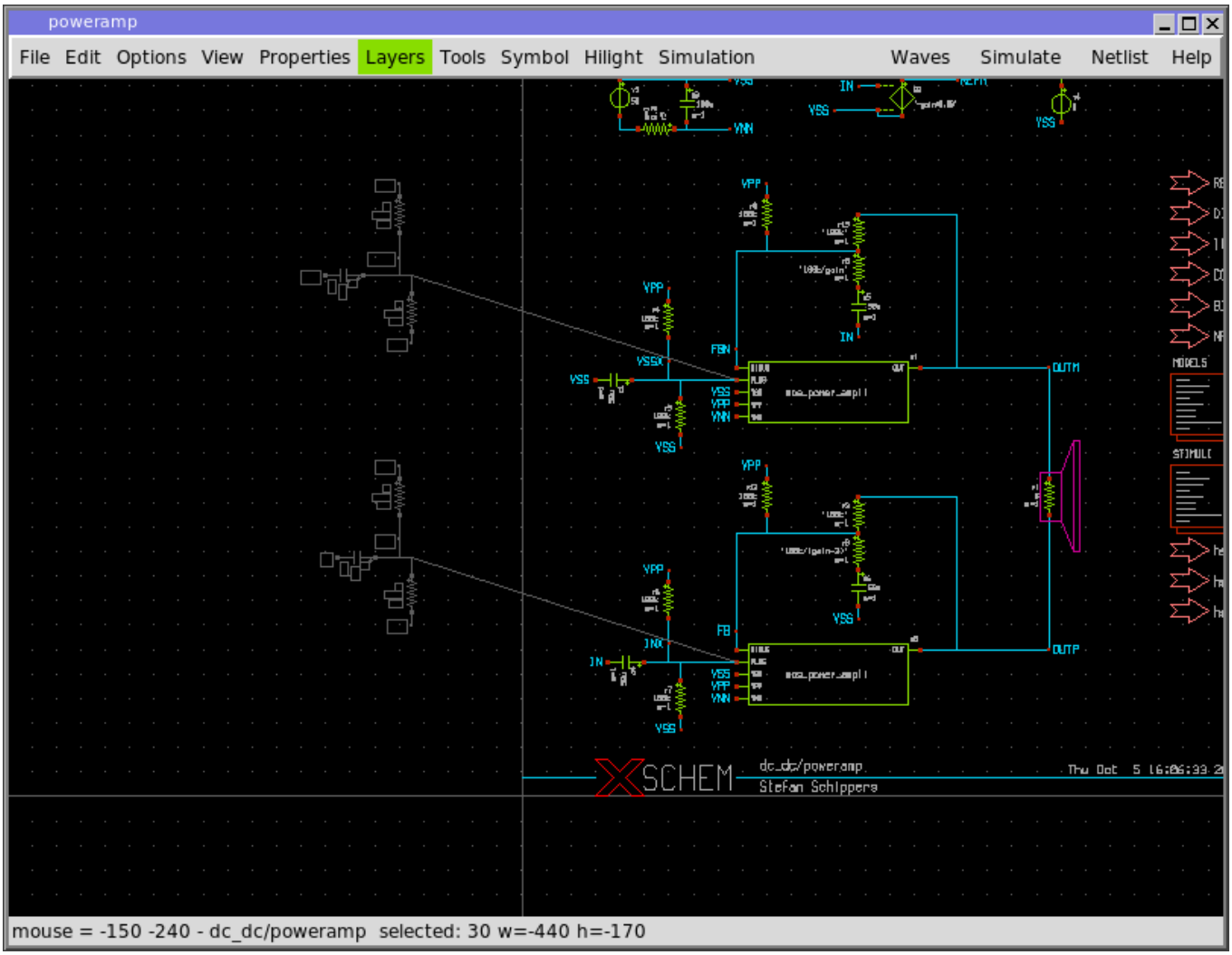
The (uppercase) **`'W'`** bindkey allows to place a wire putting start (and end point, later) to the closest pin or wire endpoint, this will make it easier to connect precisely without the need to zoom in all times.

## 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.




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




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

PREV UP NEXT

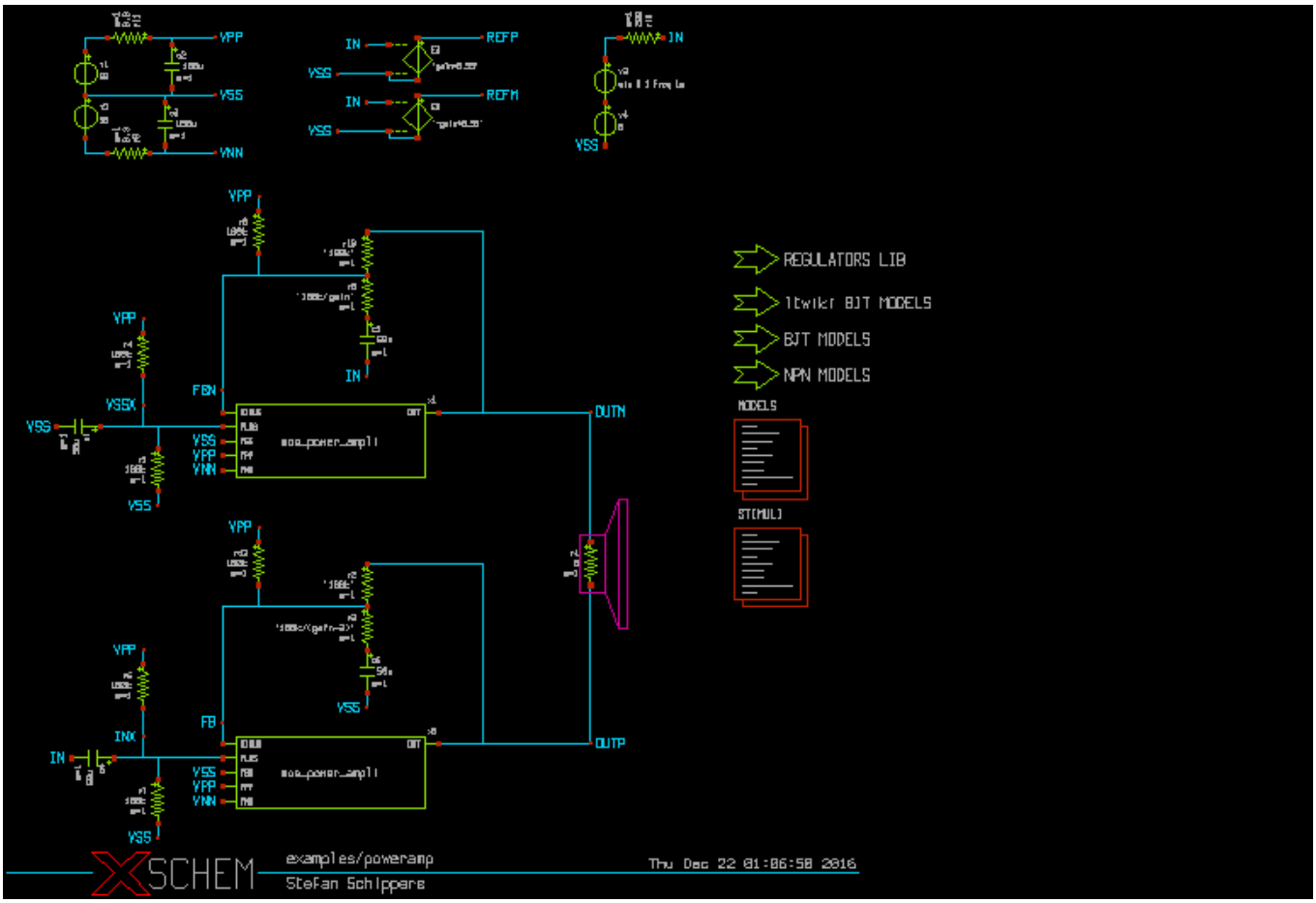
# NETLISTING

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 **<Shift>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, if no one is defined XSCHEM will prompt for a directory. The path where netlists are produced can be changed with the **Simulation->Set netlist dir** menu entry. 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 componens 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:

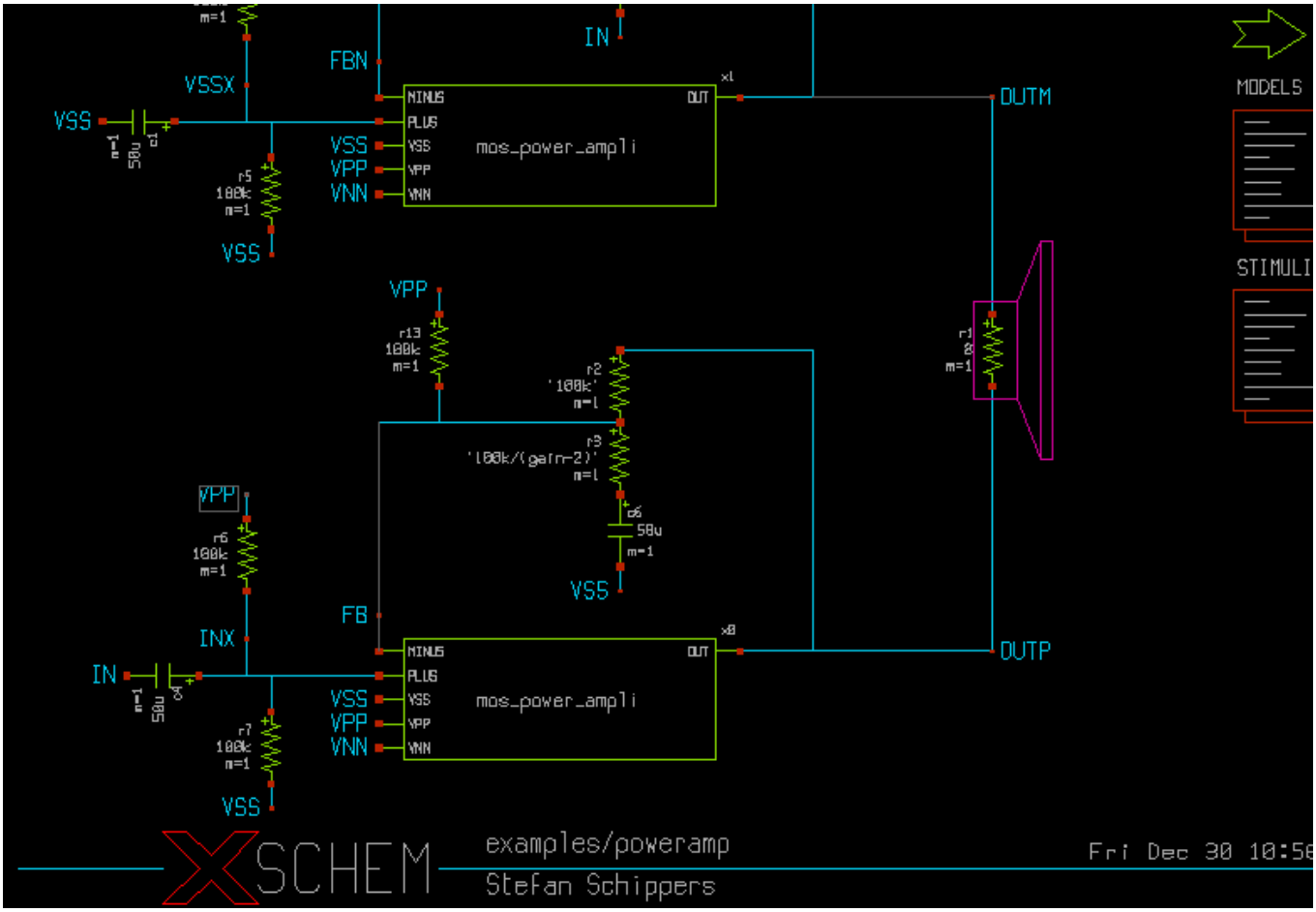
```
type=nmos
format="@name @pinlist @model w=@w l=@l m=@m"
verilog_format="@verilog_gate #(@del ) @name ( @@d , @@s , @@g );"
template="name=x1 verilog_gate=nmos del=50,50,50 model=NCH w=0.68 l=0.07 m=1"
generic_type="model=string"
```

PREV UP NEXT

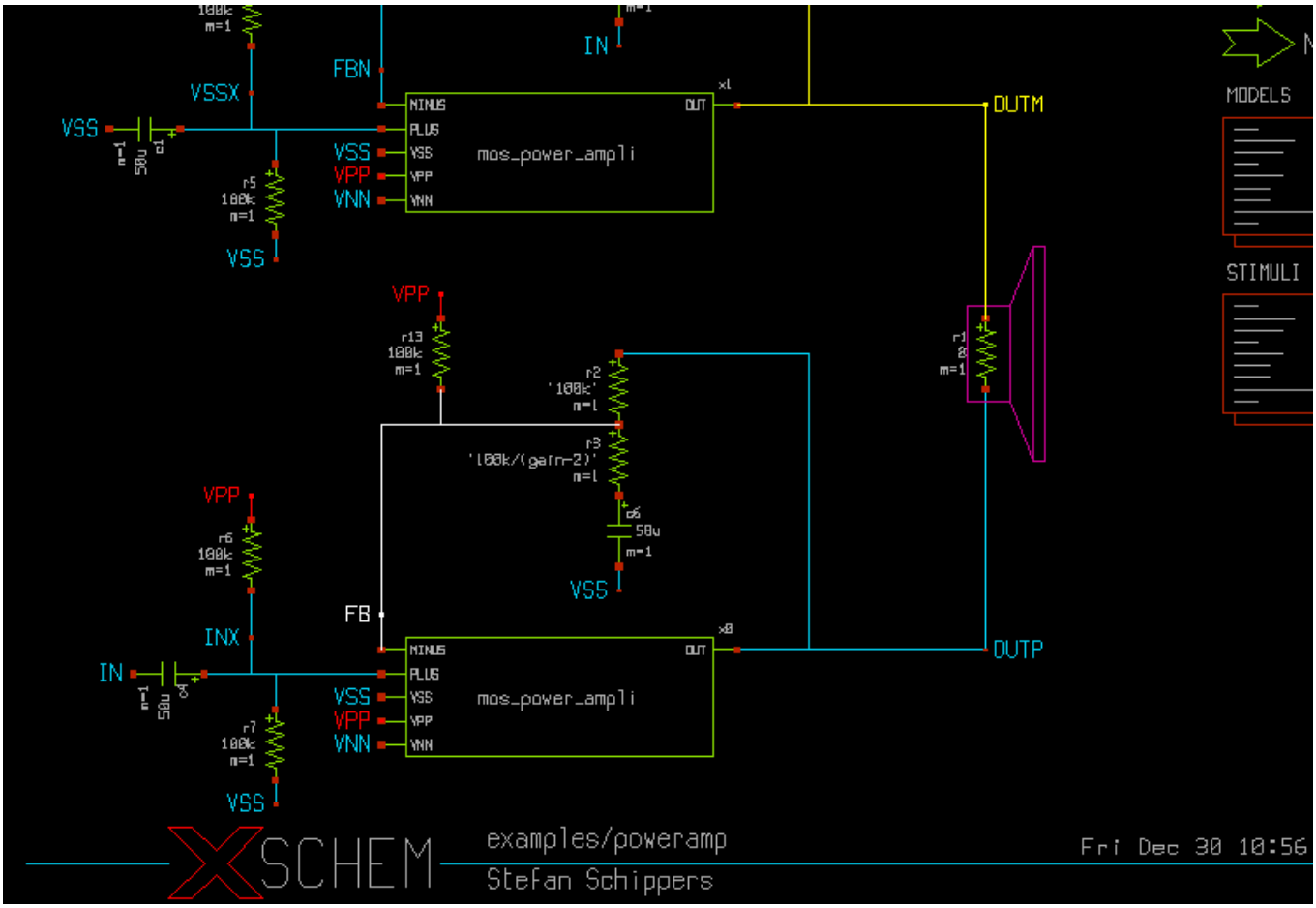
# NET PROBES

XSCHEM has the ability to hilight 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.

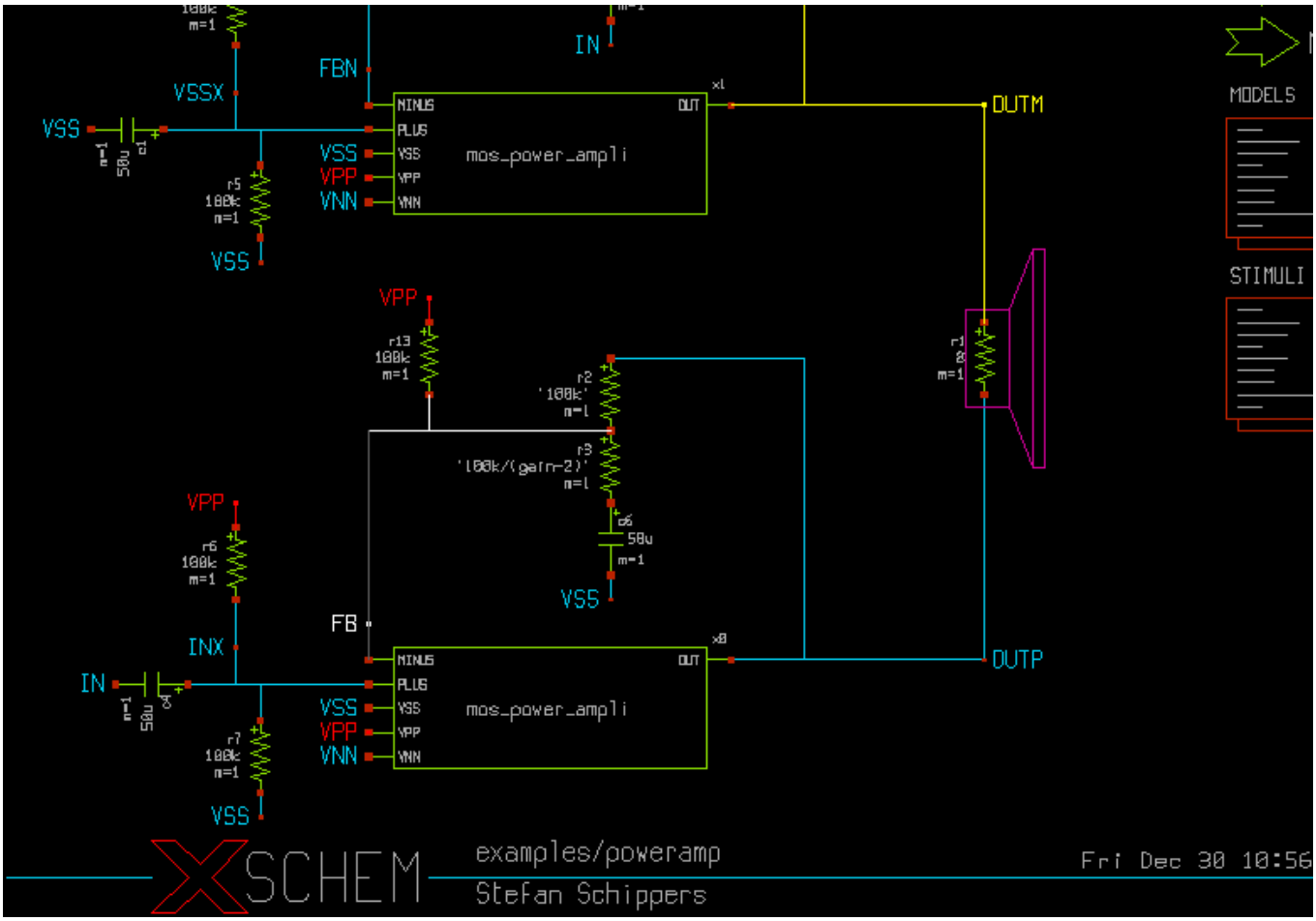
Select some nets...



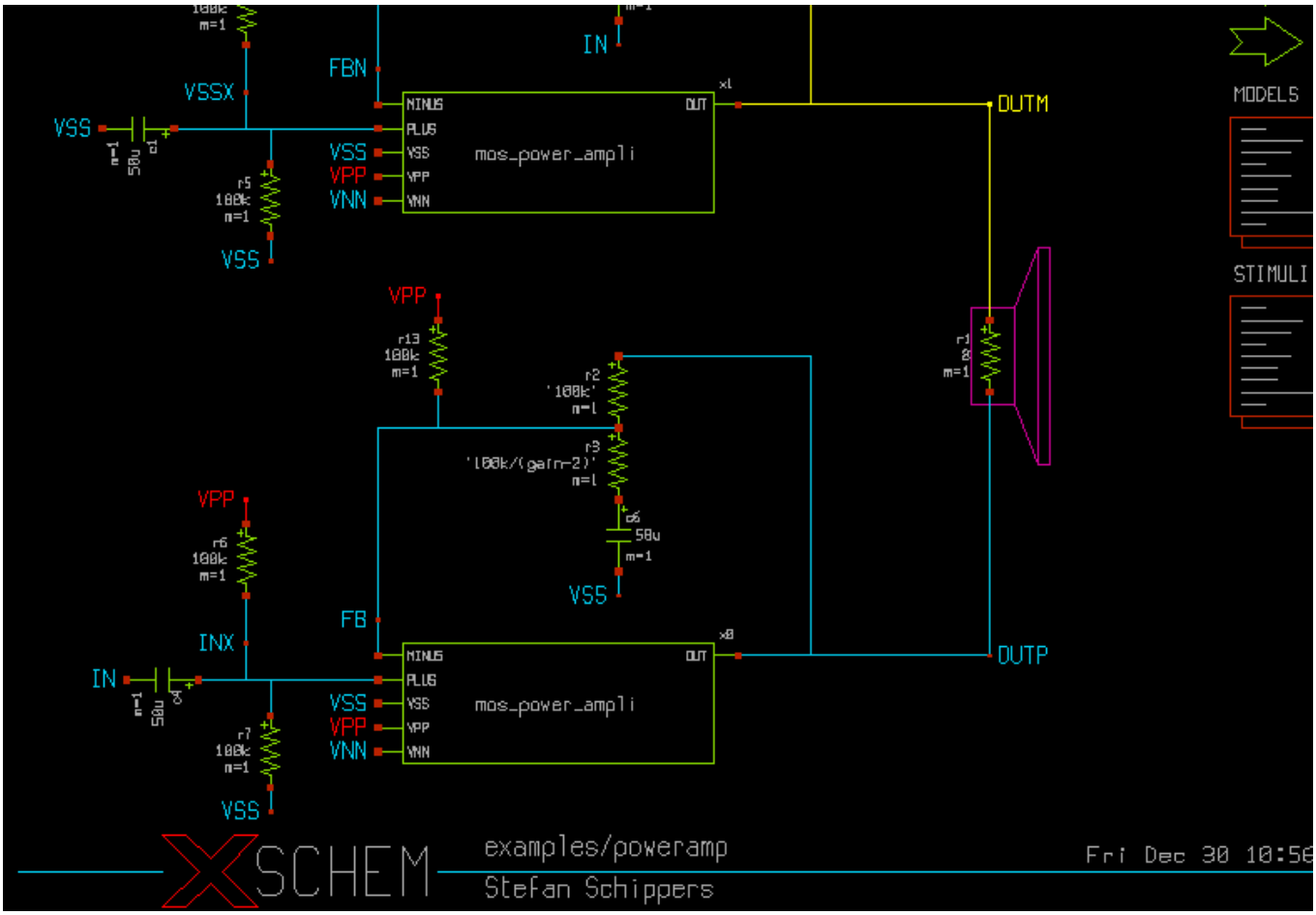

...press the **'k'** key...



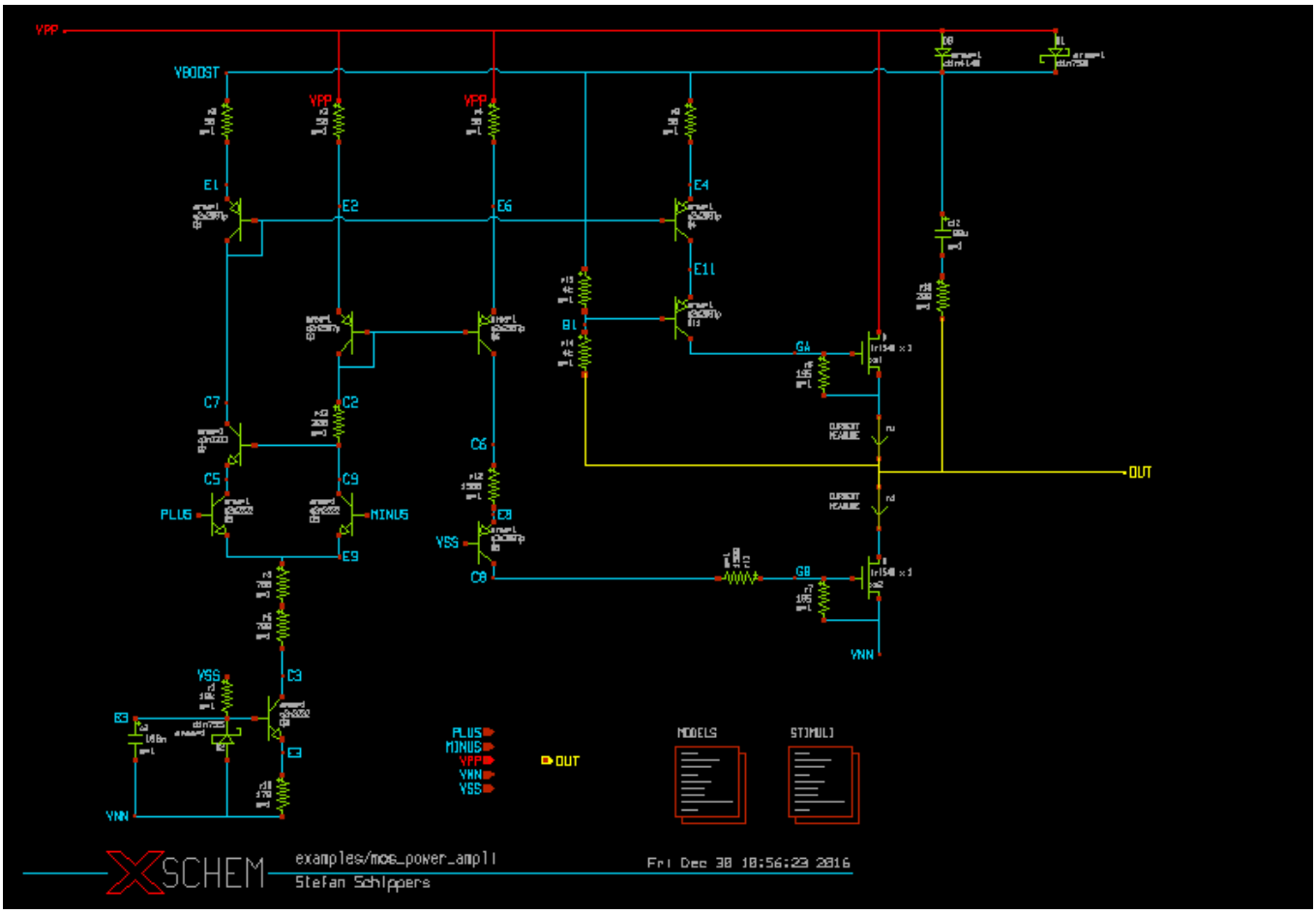

...all nets are highlighted, select the white net...



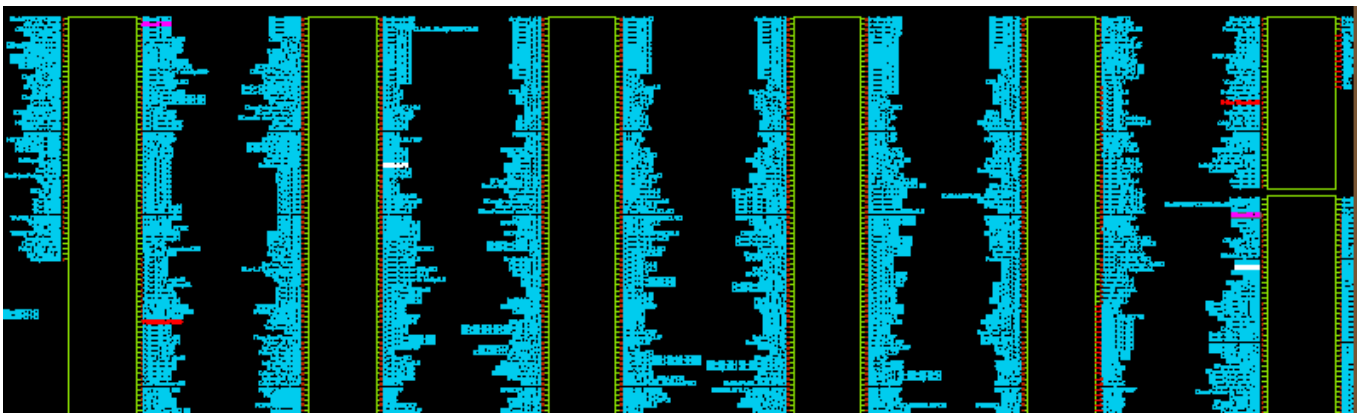

..press the **<Ctrl>k** key and white net is un-highlighted...




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



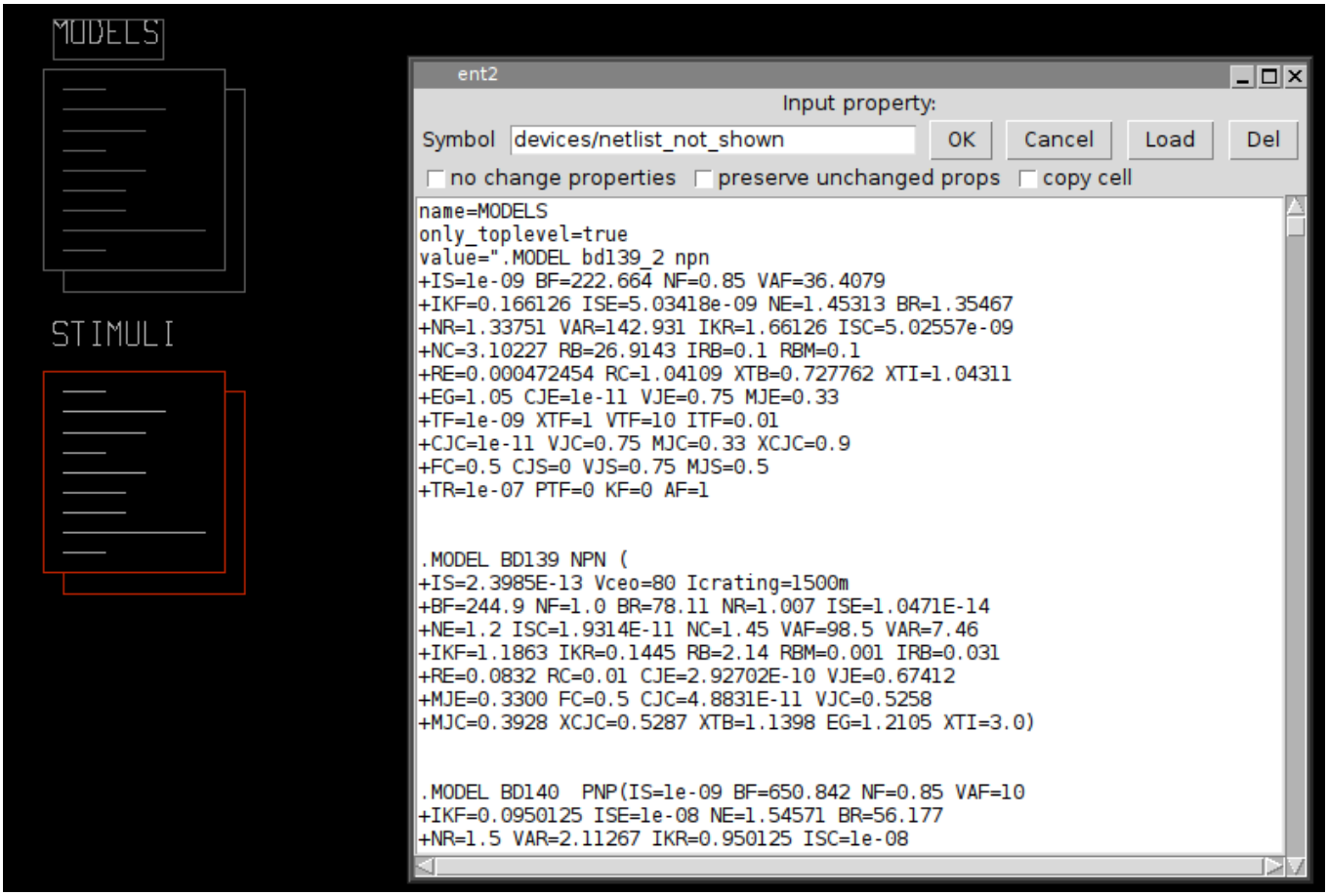

A very useful function is the 'View only probes' mode, (`'5'` key) that hides everything but the highlight probes. This is useful in very big VLSI designs to quickly locate start and end point of nets. Pressing again the `'5'` key restores the normal view.



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.

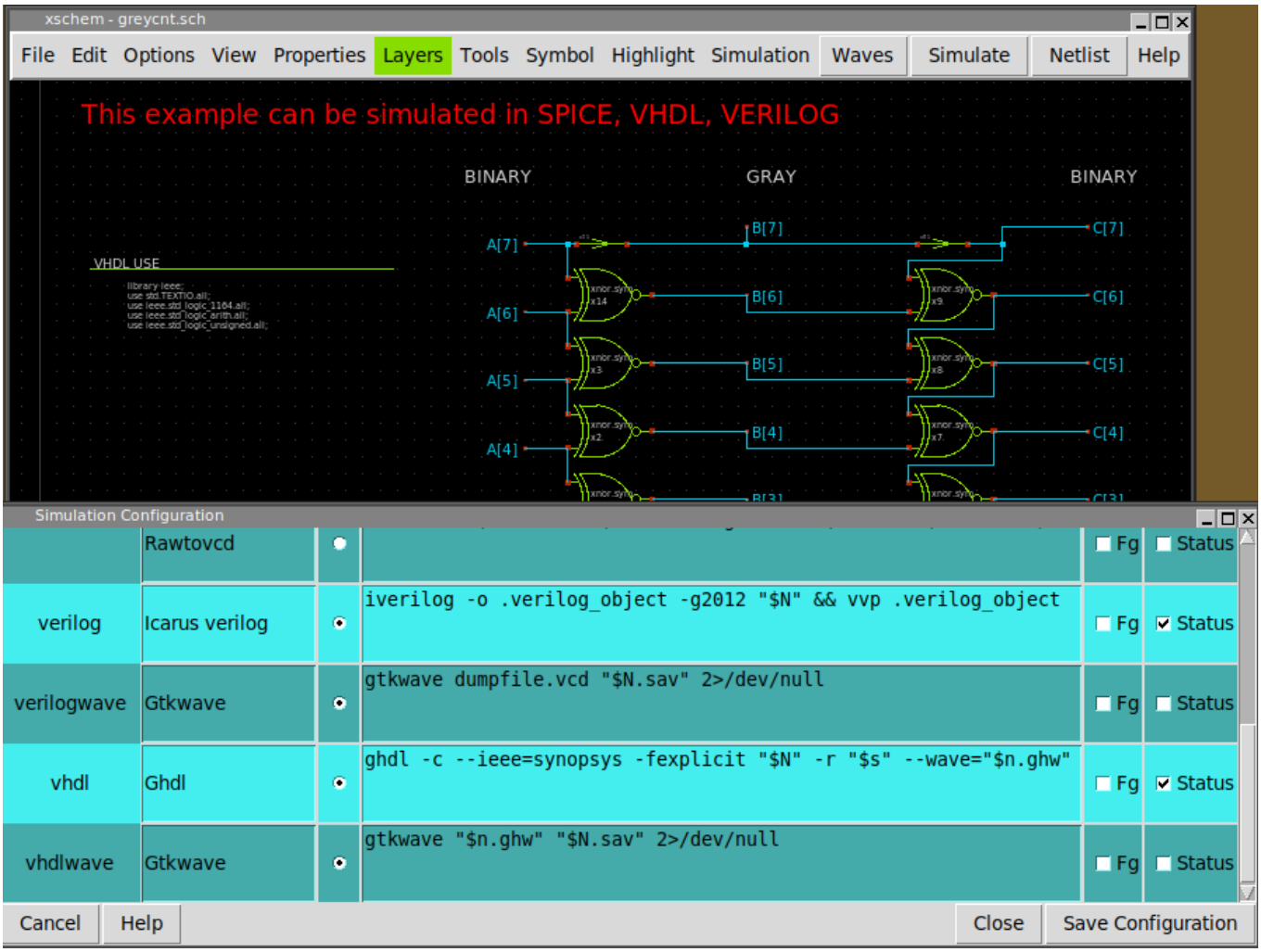



The **devices/netlist_not_shown** symbol shown in the picture (with name MODELS) for example contains all the spice models of the components used in the schematic, this makes the schematic self contained, no additional files are needed to run a simulation. After generating the netlist (for example poweramp.spice)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

## 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**

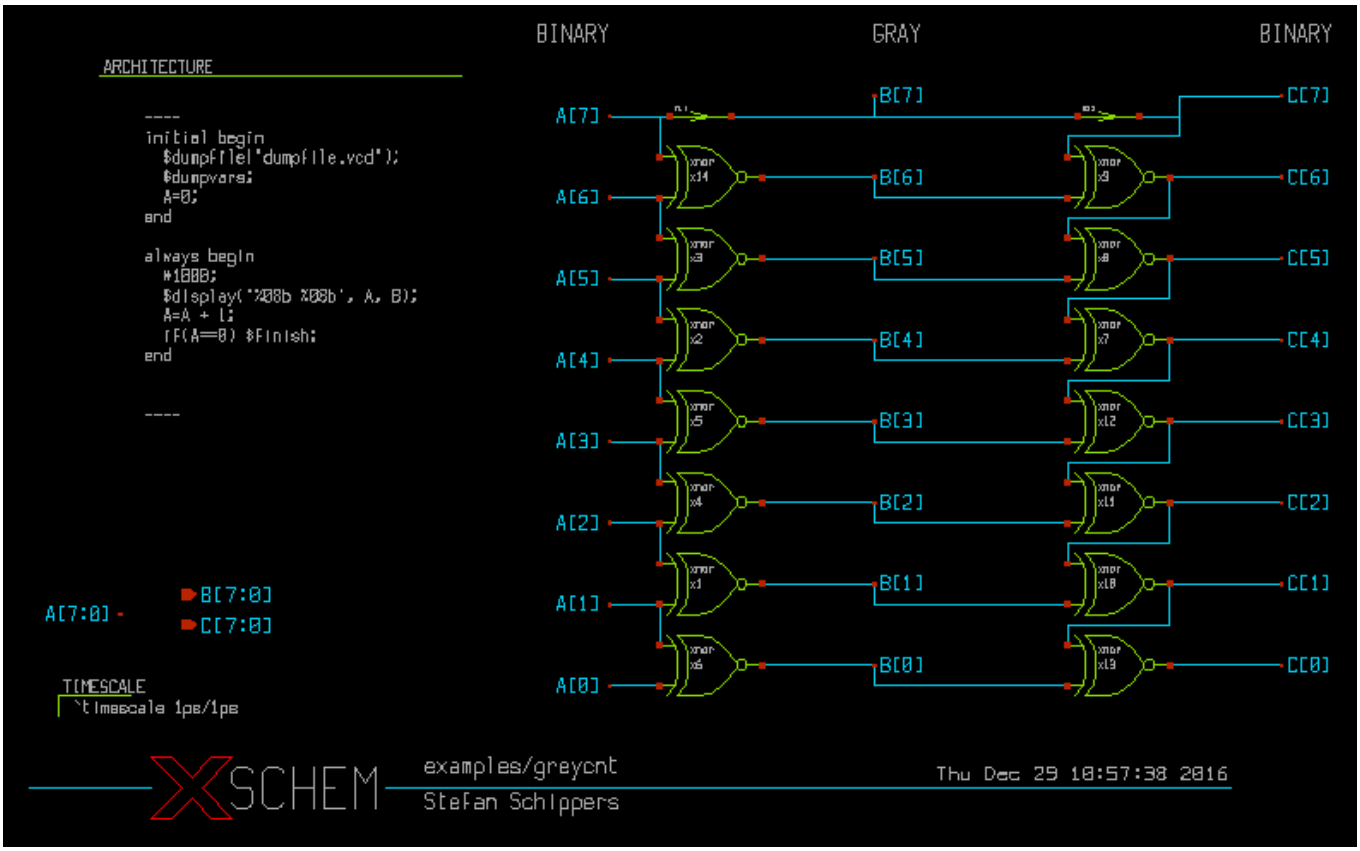
**tools**).




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 theverilog simulation. If **Save Configuration** button is pressed the changes are made permanent by saving in a **~/.xschem/simrc** file.

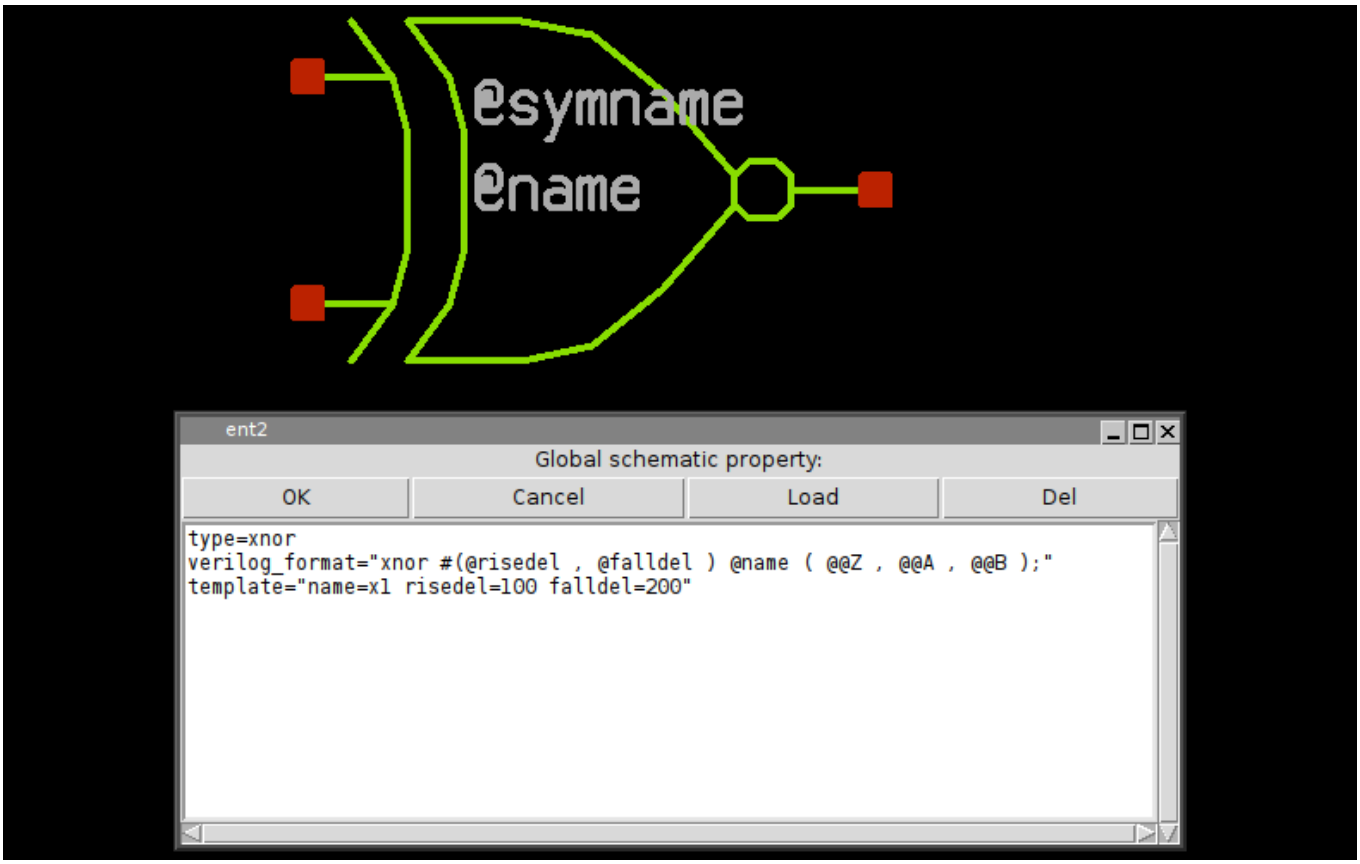
In the XSCHEM distribution there is one example design, **examples/greycnt.sch**.
Load this design:

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

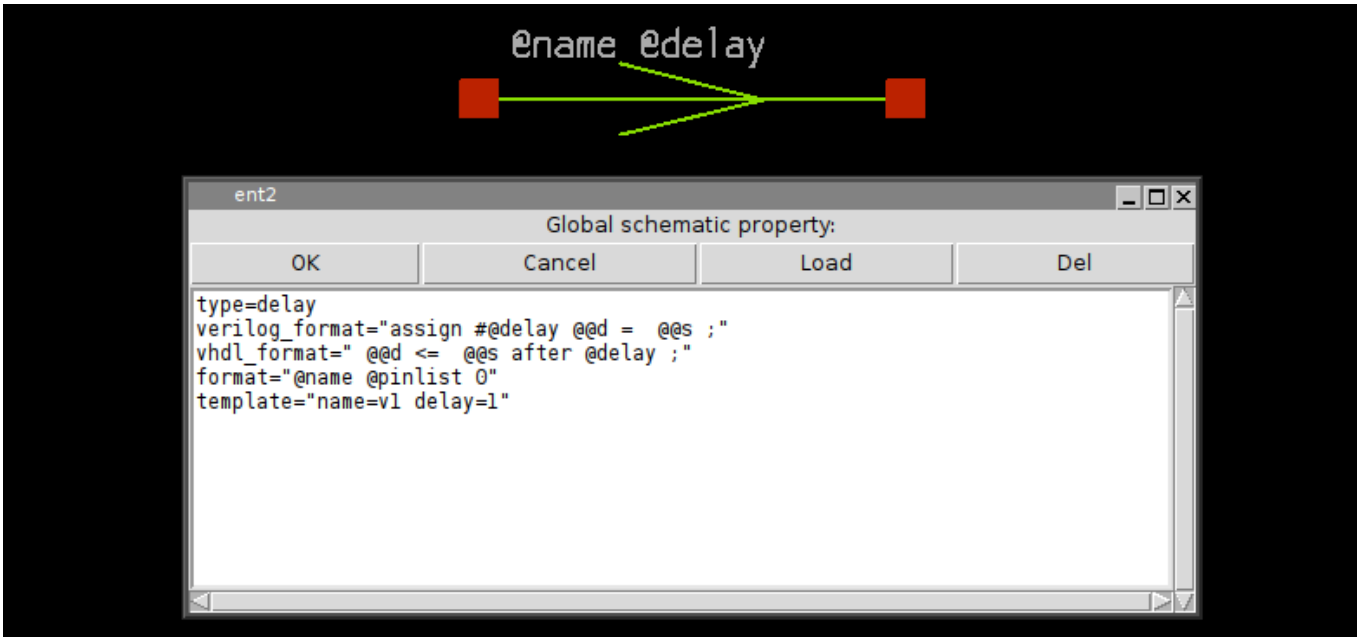



This testbench has a 8 bit input vector A[7:0] and two output vectors, B[7:0] and C[7:0]. B[7:0] is a grey coded vector, this mean that if A[7:0] is incremented as a binary number B[7:0] will increment by changing only one bit at a time. The C[7:0] vector is the reverse transformation from grey-code 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).

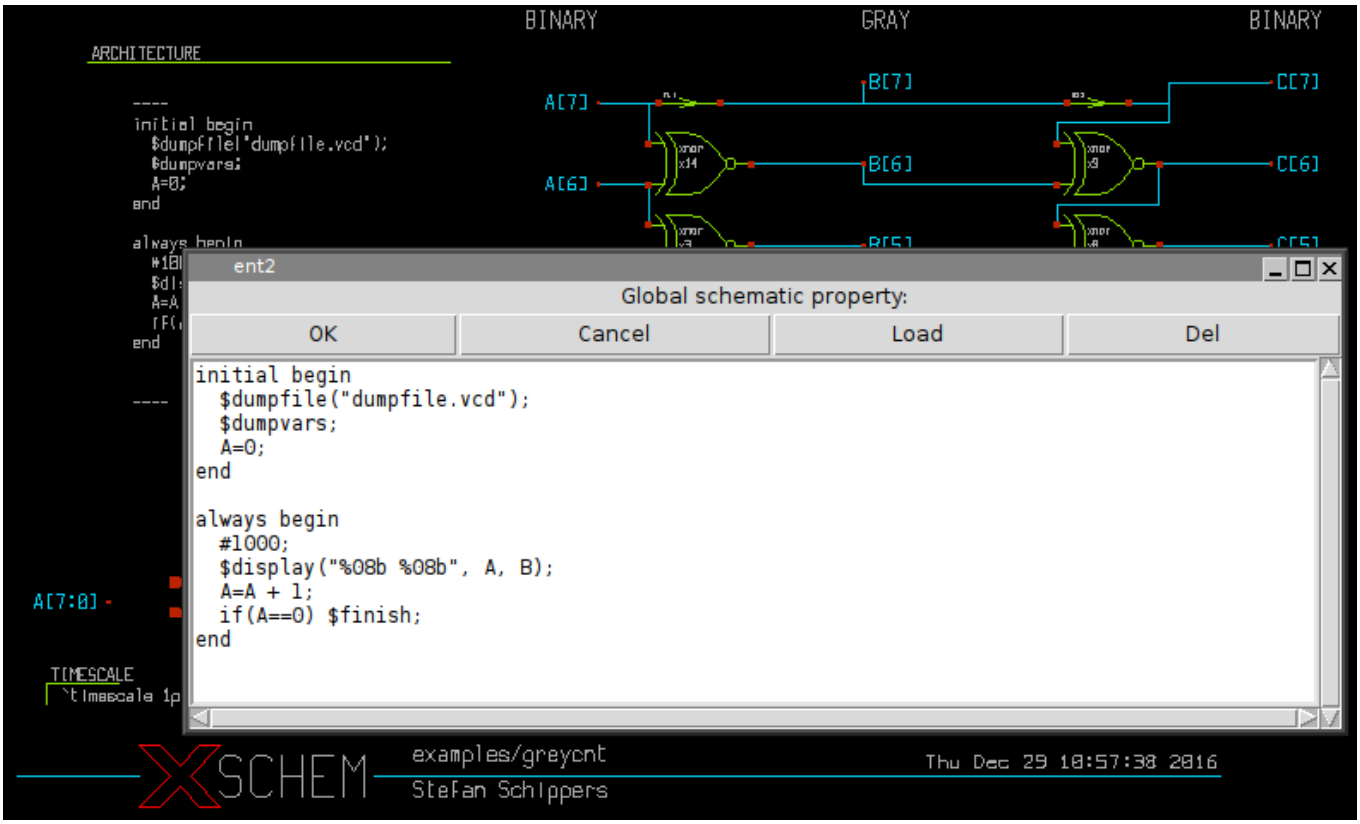
An Ex-Nor gate can be represented as a verilog primitive, so for the xnor gate we just need to setup a **`verilog_format`** attribute in the global property string of the **`xnor.sym`** gate:




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.




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:

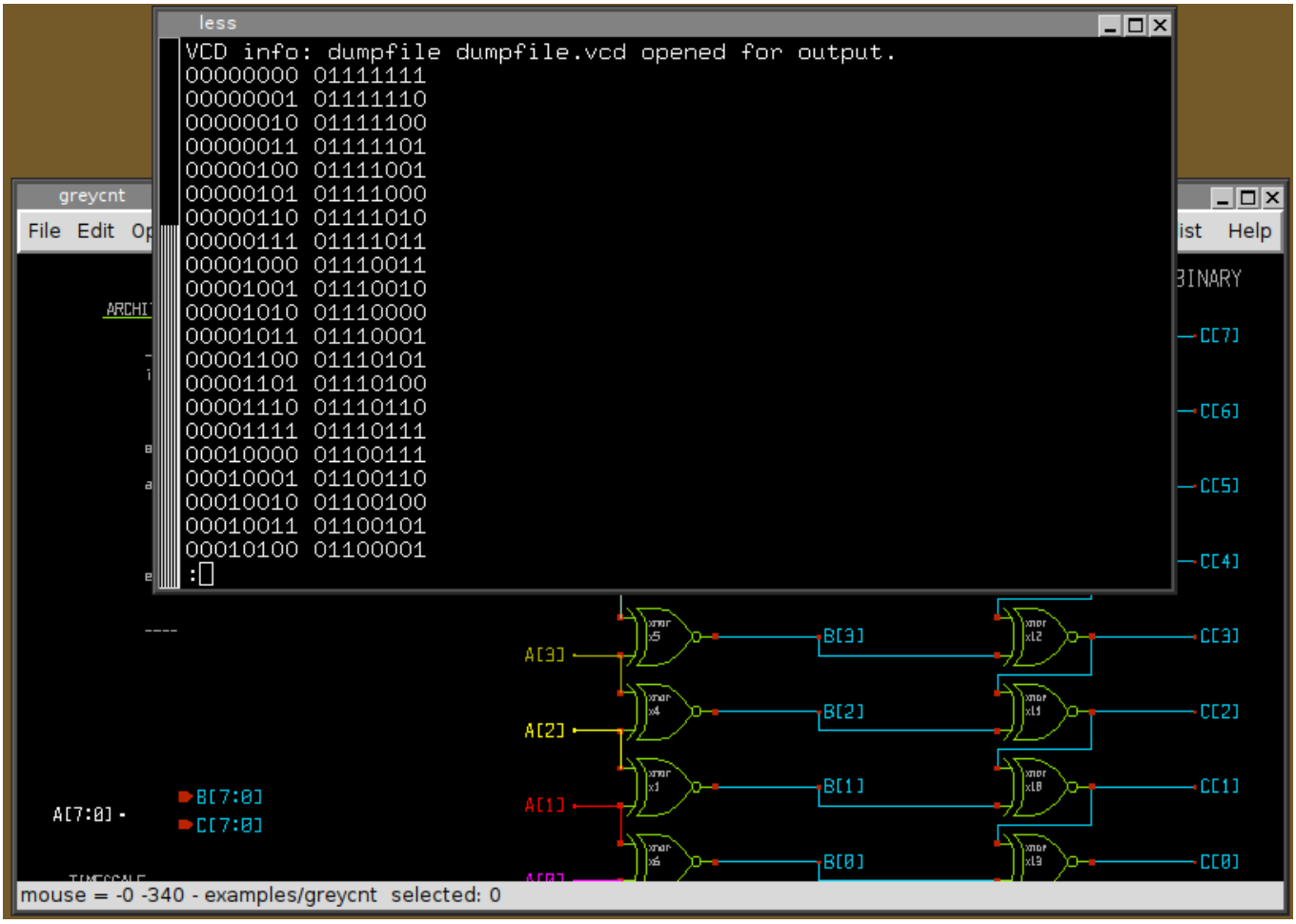



This is the testbench behavioral code that generates stimuli for the simulation and gives instructions on where to save simulation results. If you generate the verilog netlist with the **Netlist** button on the right side of the menu bar (or **n** key) a **greycnt.v** file will be generated in the simulation 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):

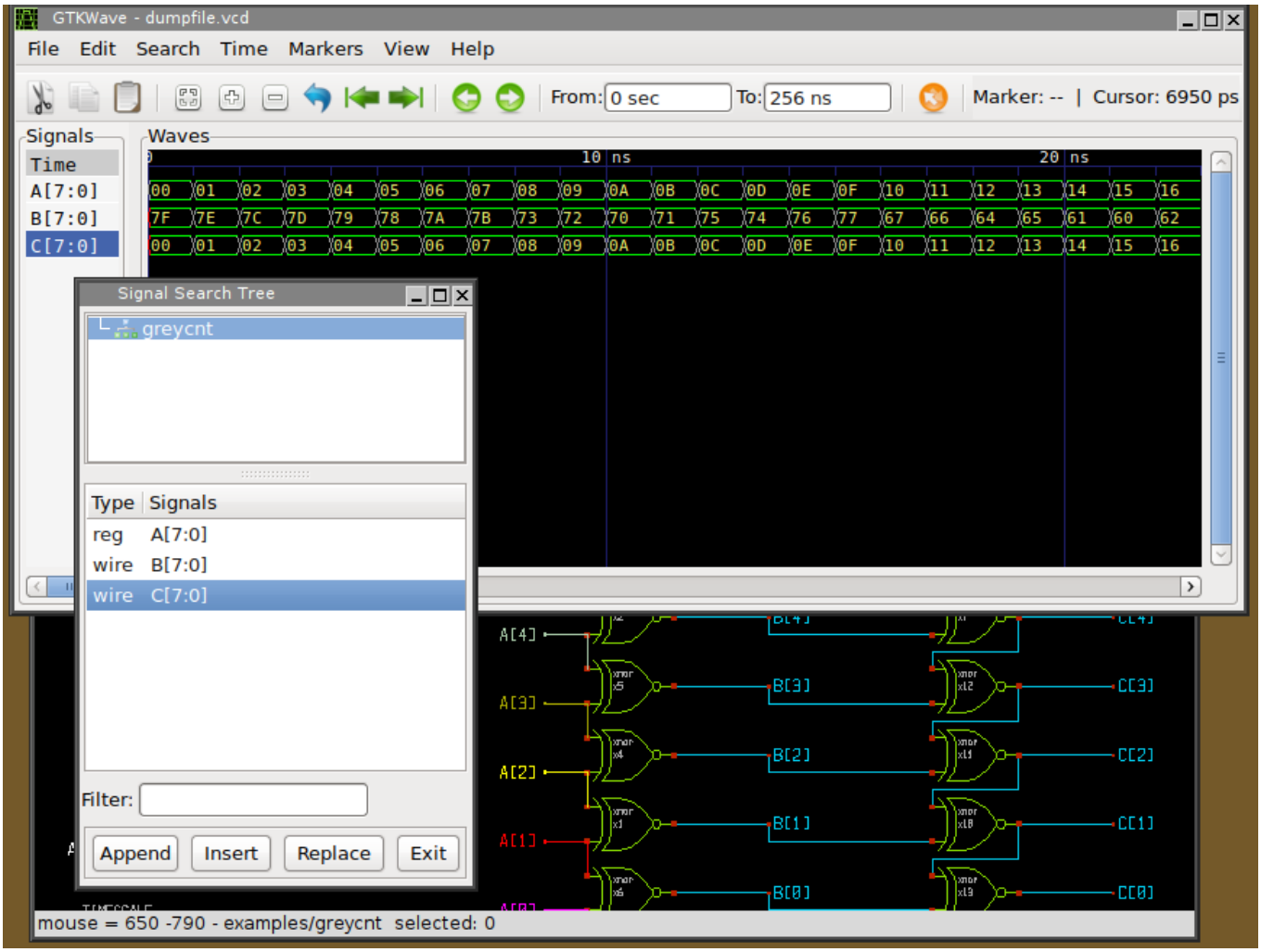
```
`timescale 1ps/1ps
module greycnt (
  output wire [7:0] B,
  output wire [7:0] C
);

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
```

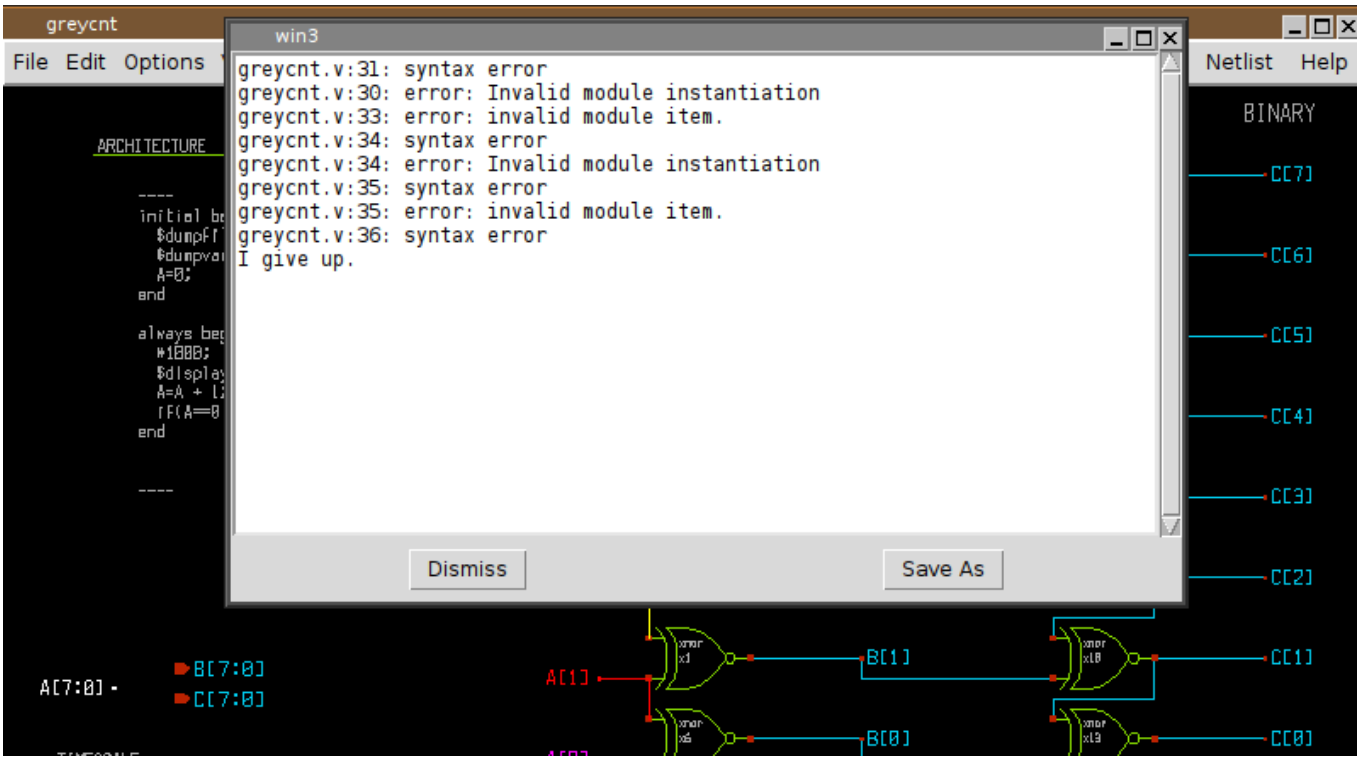
you will recognize the behavioral code right after the netlist specifying the connection of nets to the xnor and assign gates and all the necessary verilog declarations. If you press the **`Simulation`** button the **`Icarus Verilog`** simulator will be executed to compile (iverilog) and run (vvp) the 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'`**.



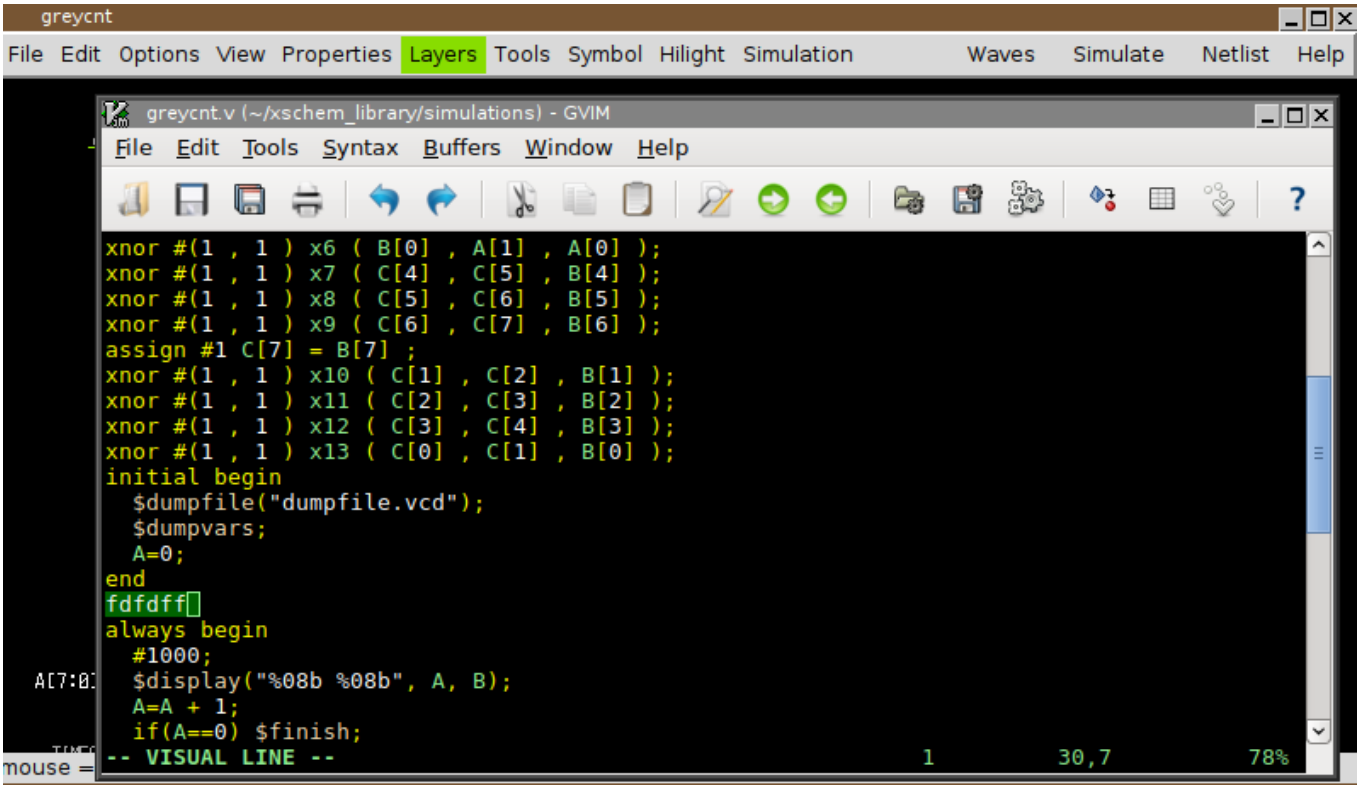

If simulation completes with no errors waveforms can be viewed. Press the **`Waves`** button in the top-right of the menu bar, you may add waveforms in the gtkwave window:



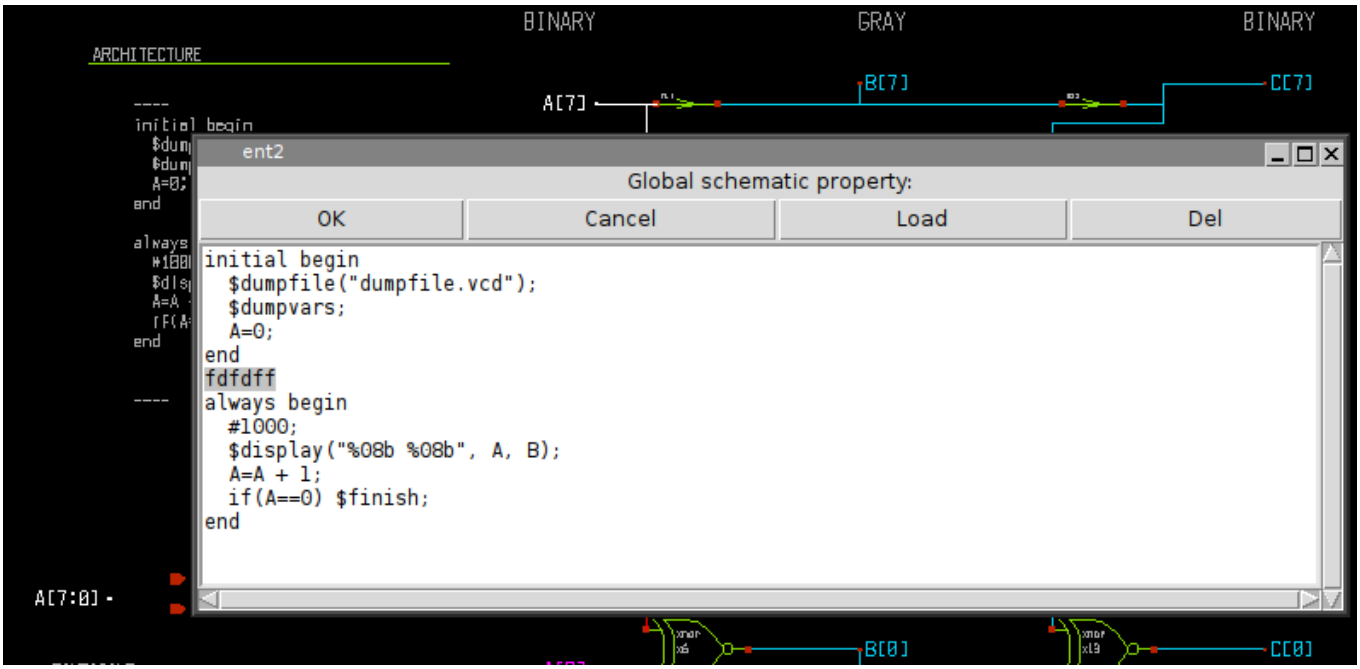

If the schematic contains errors that the simulator can not handle instead of the simulation log a window showing the error messages from the simulator is shown:




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:



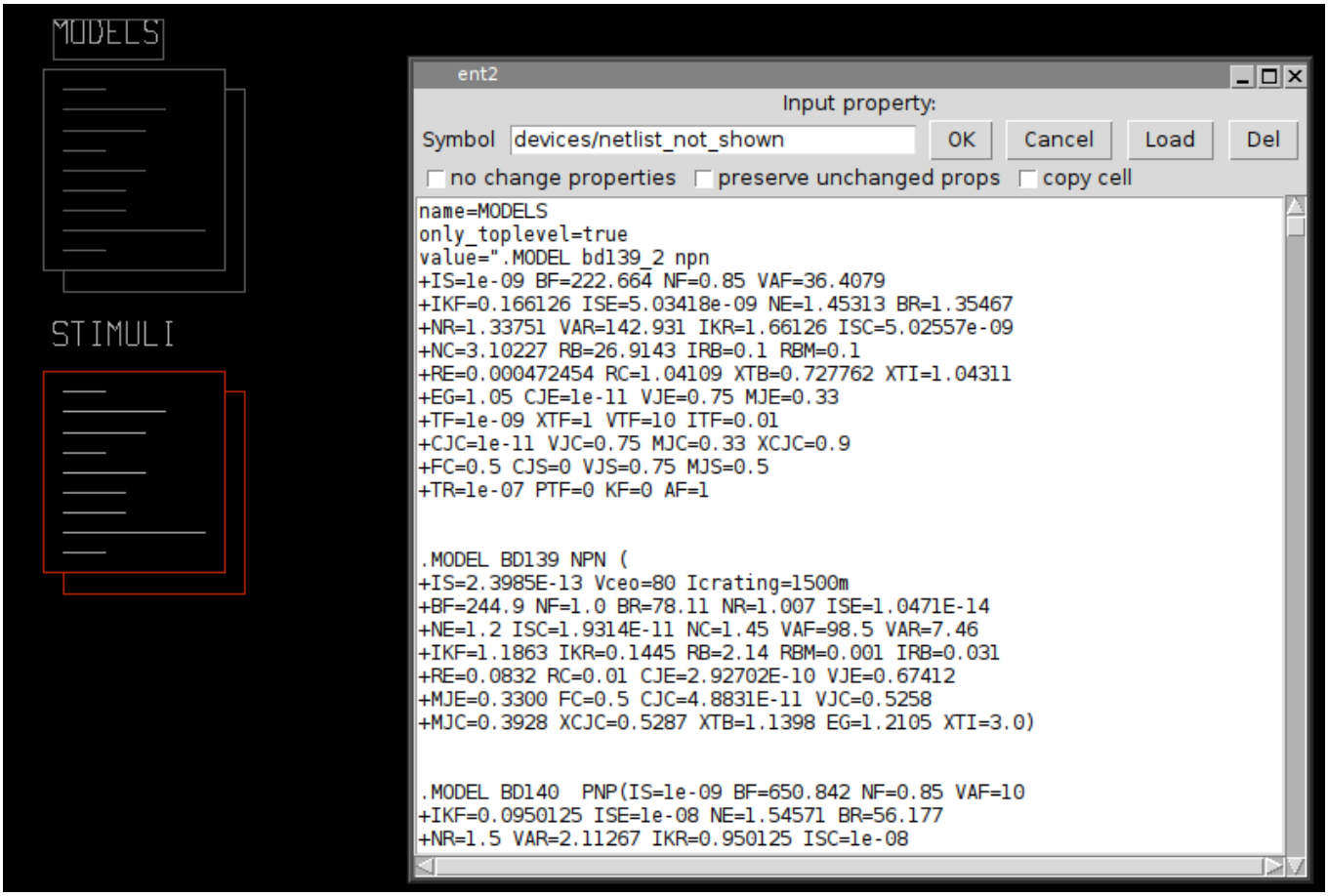

As you can see the error is in the behavioral code of the top level greycnt schematic, so edit the global property (**`'q'`** key with no component selected) and fix the error.


BINARY
GRAY
BINARY
ARCHITECTURE
A[7]
B[7]
C[7]
ent2
Global schematic property:
OK
Cancel
Load
Del
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
A[7:0]
B[0]
C[0]

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.

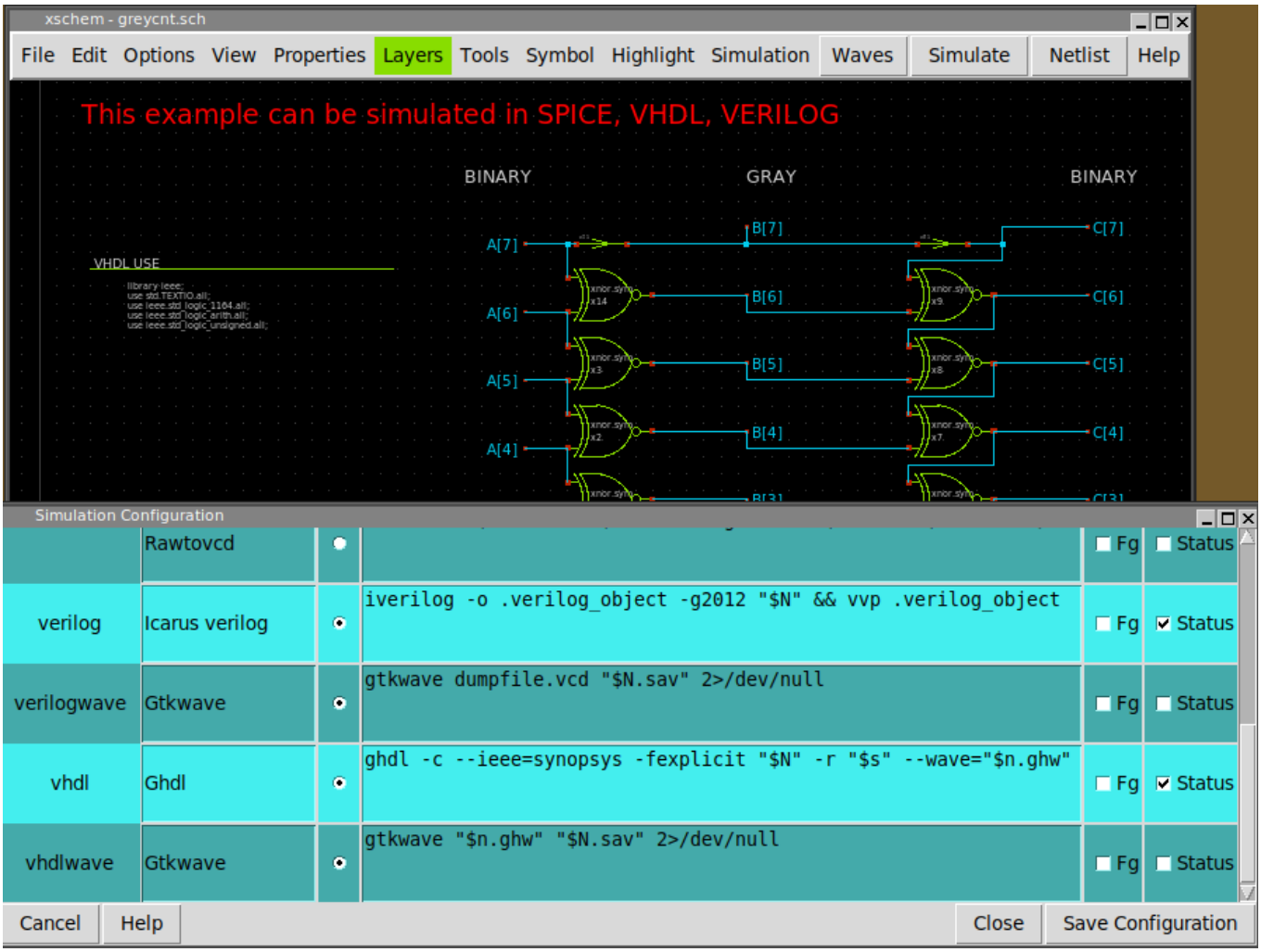



The **devices/netlist_not_shown** symbol shown in the picture (with name MODELS) for example contains all the spice models of the components used in the schematic, this makes the schematic self contained, no additional files are needed to run a simulation. After generating the netlist (for example poweramp.spice)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

## 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**

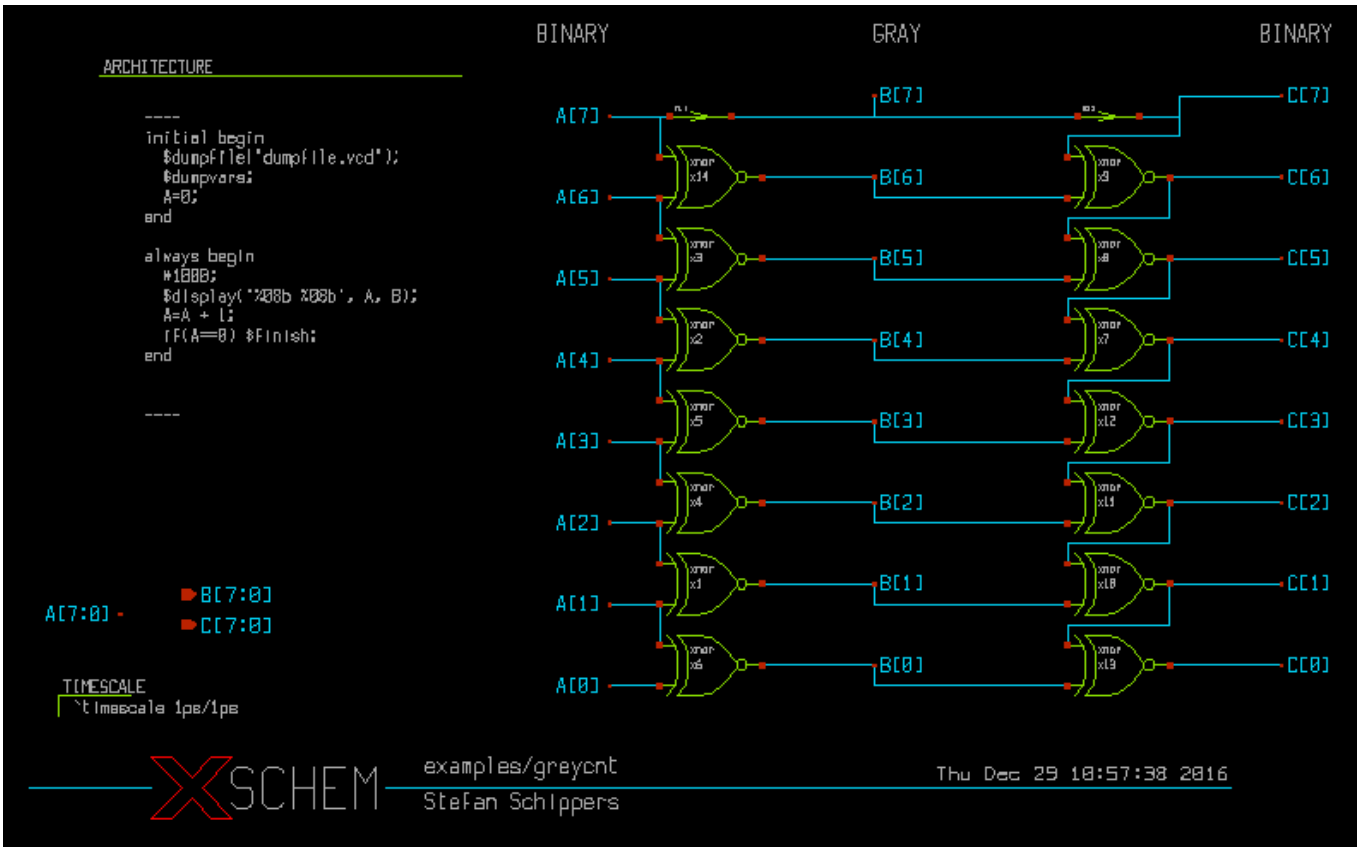
**tools**).




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 theverilog simulation. If **Save Configuration** button is pressed the changes are made permanent by saving in a **~/.xschem/simrc** file.

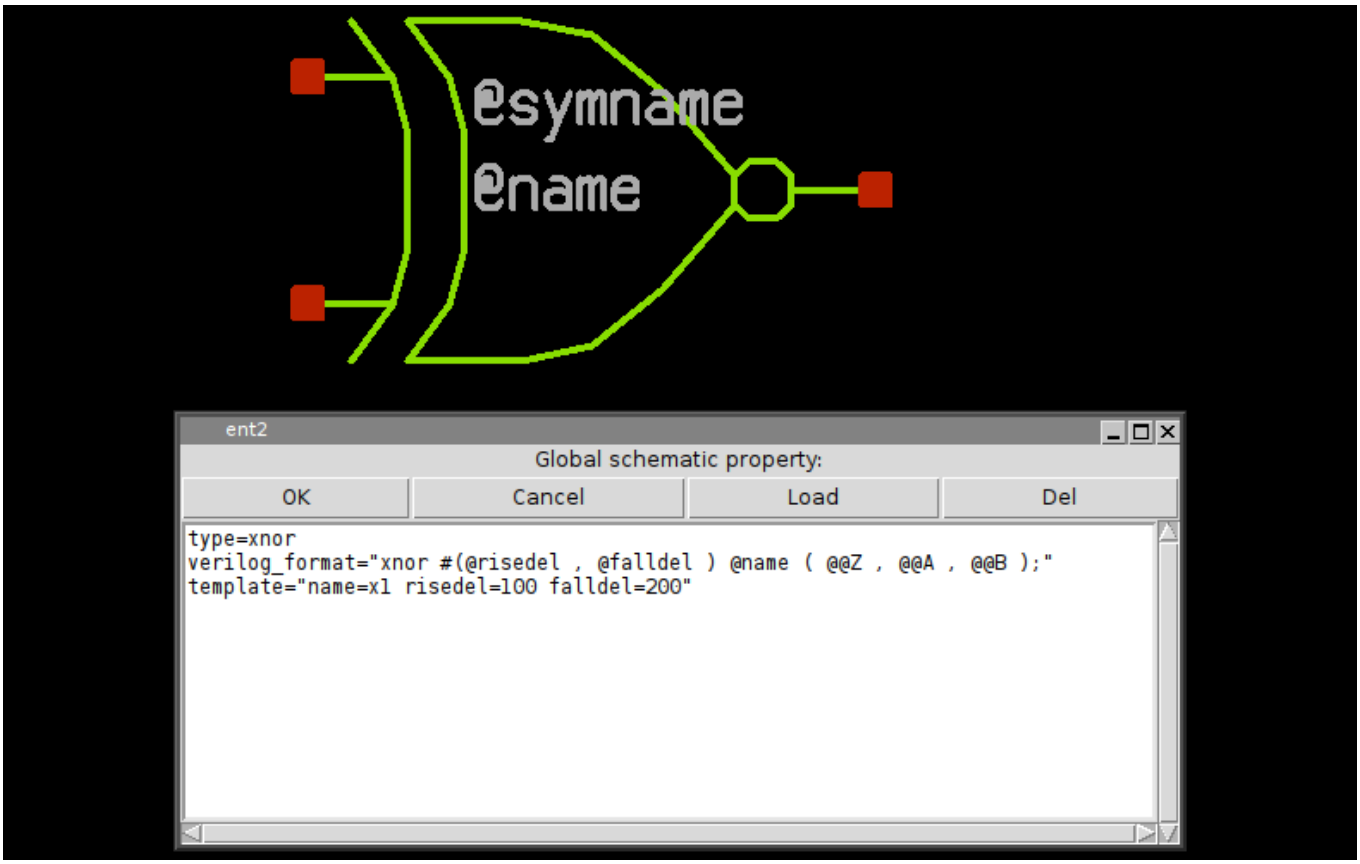
In the XSCHEM distribution there is one example design, **examples/greycnt.sch**.
Load this design:

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

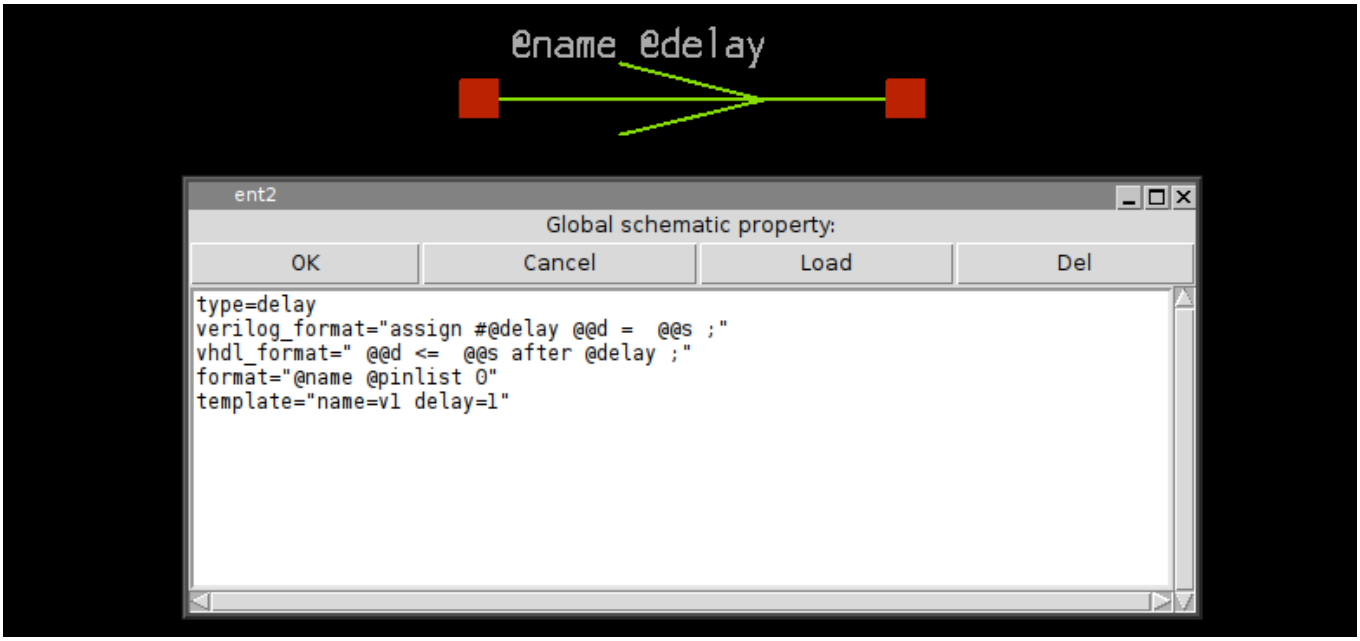



This testbench has a 8 bit input vector A[7:0] and two output vectors, B[7:0] and C[7:0]. B[7:0] is a grey coded vector, this mean that if A[7:0] is incremented as a binary number B[7:0] will increment by changing only one bit at a time. The C[7:0] vector is the reverse transformation from grey-code 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).

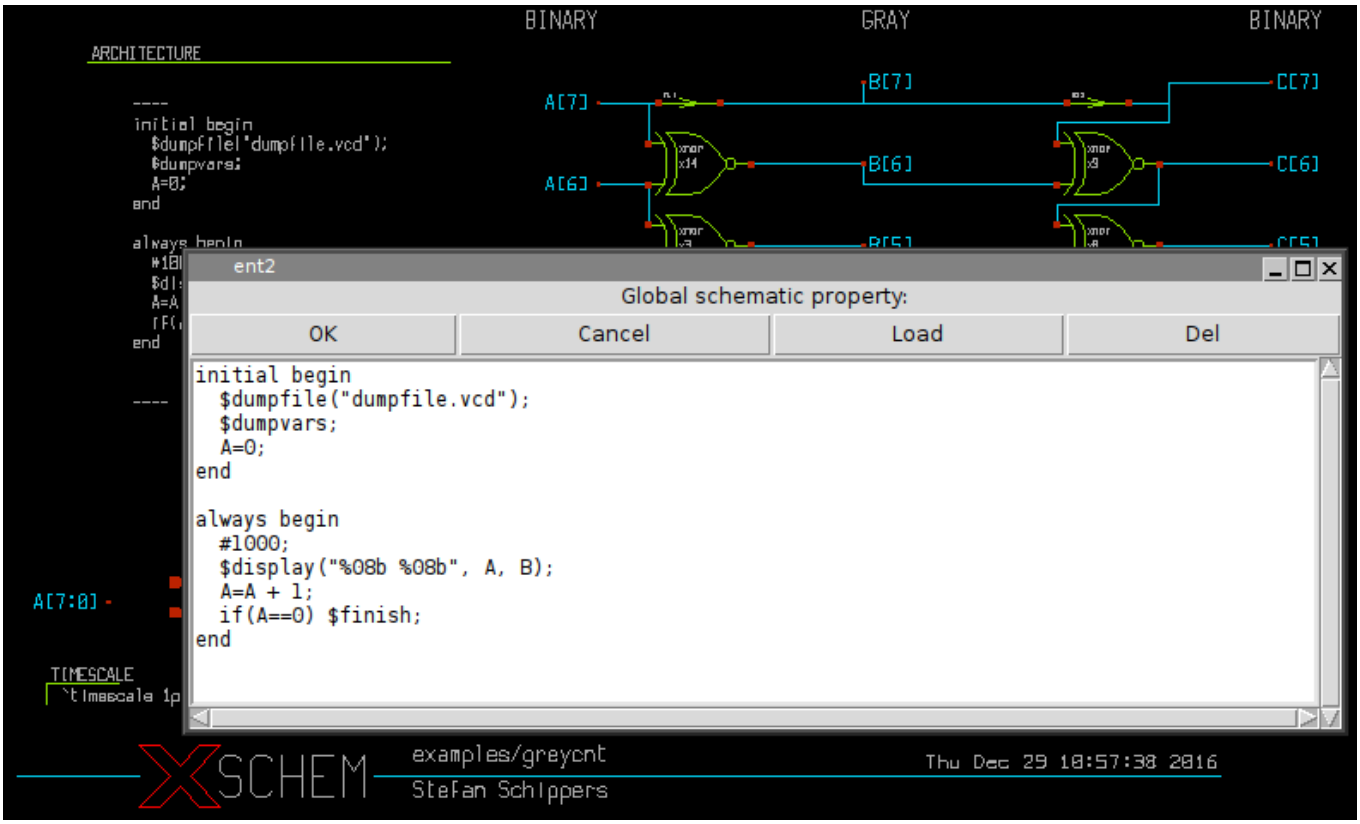
An Ex-Nor gate can be represented as a verilog primitive, so for the xnor gate we just need to setup a **`verilog_format`** attribute in the global property string of the **`xnor.sym`** gate:




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.




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:

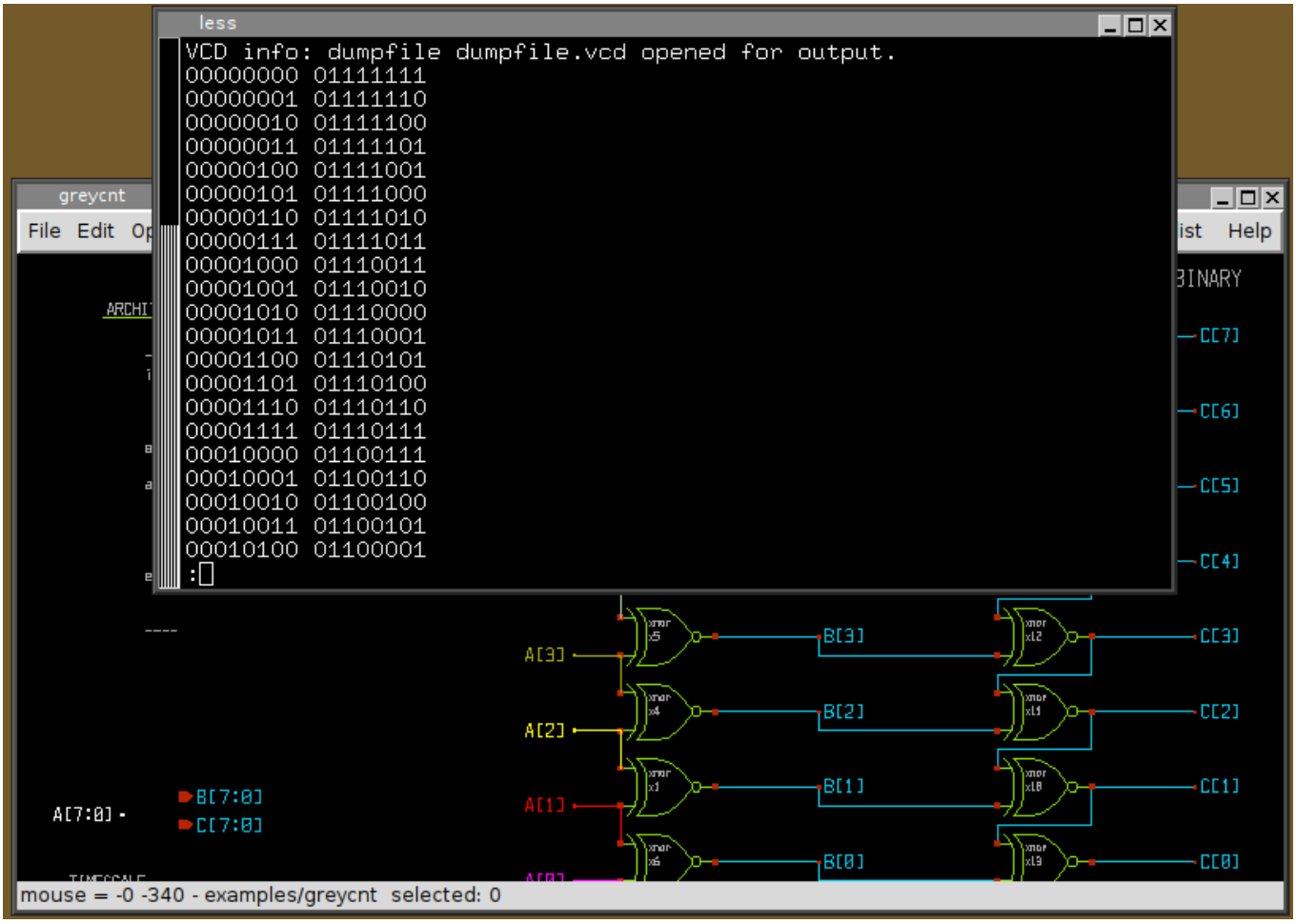



This is the testbench behavioral code that generates stimuli for the simulation and gives instructions on where to save simulation results. If you generate the verilog netlist with the **Netlist** button on the right side of the menu bar (or **n** key) a **greycnt.v** file will be generated in the simulation 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):

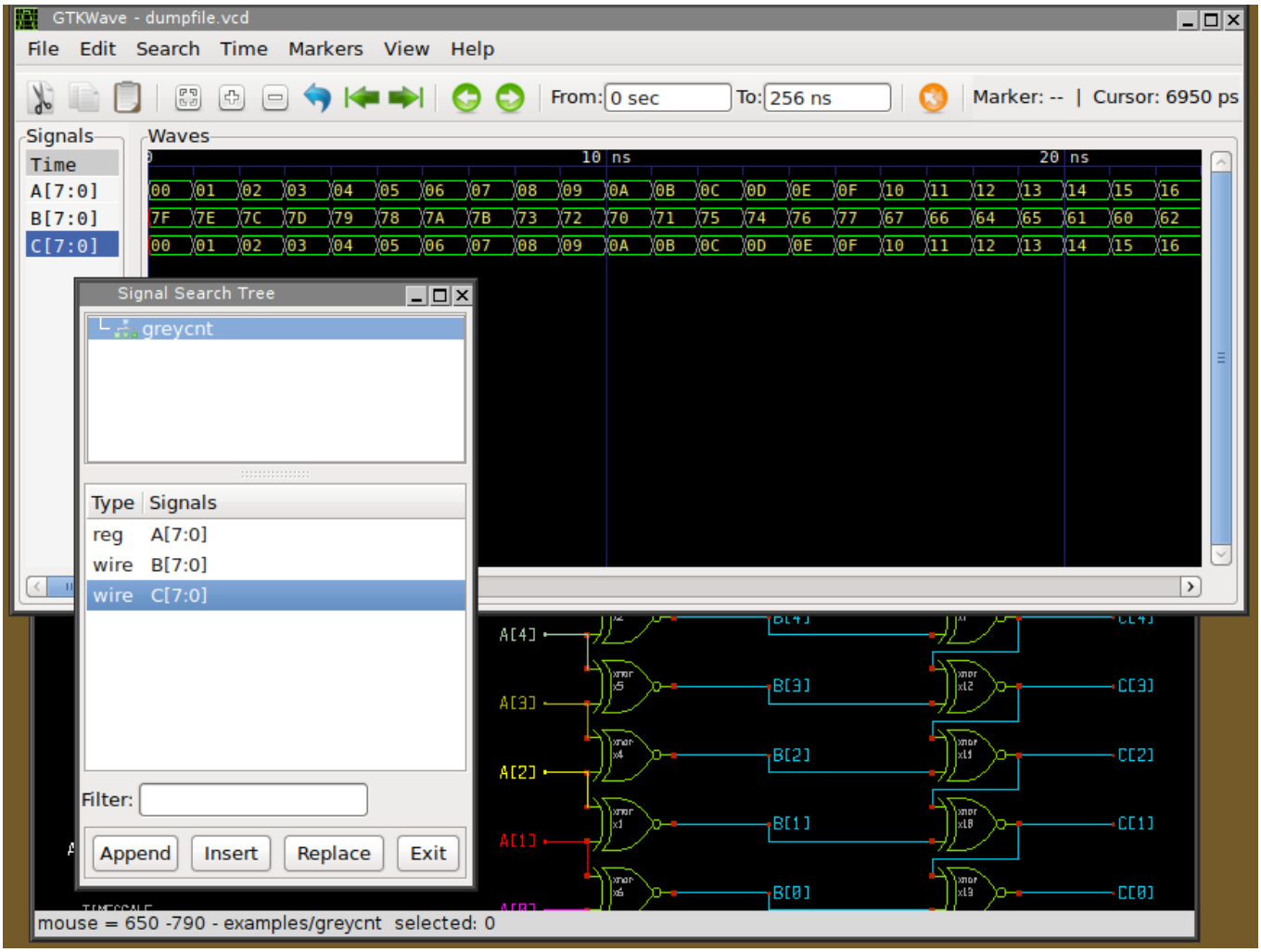
```
`timescale 1ps/1ps
module greycnt (
  output wire [7:0] B,
  output wire [7:0] C
);

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
```

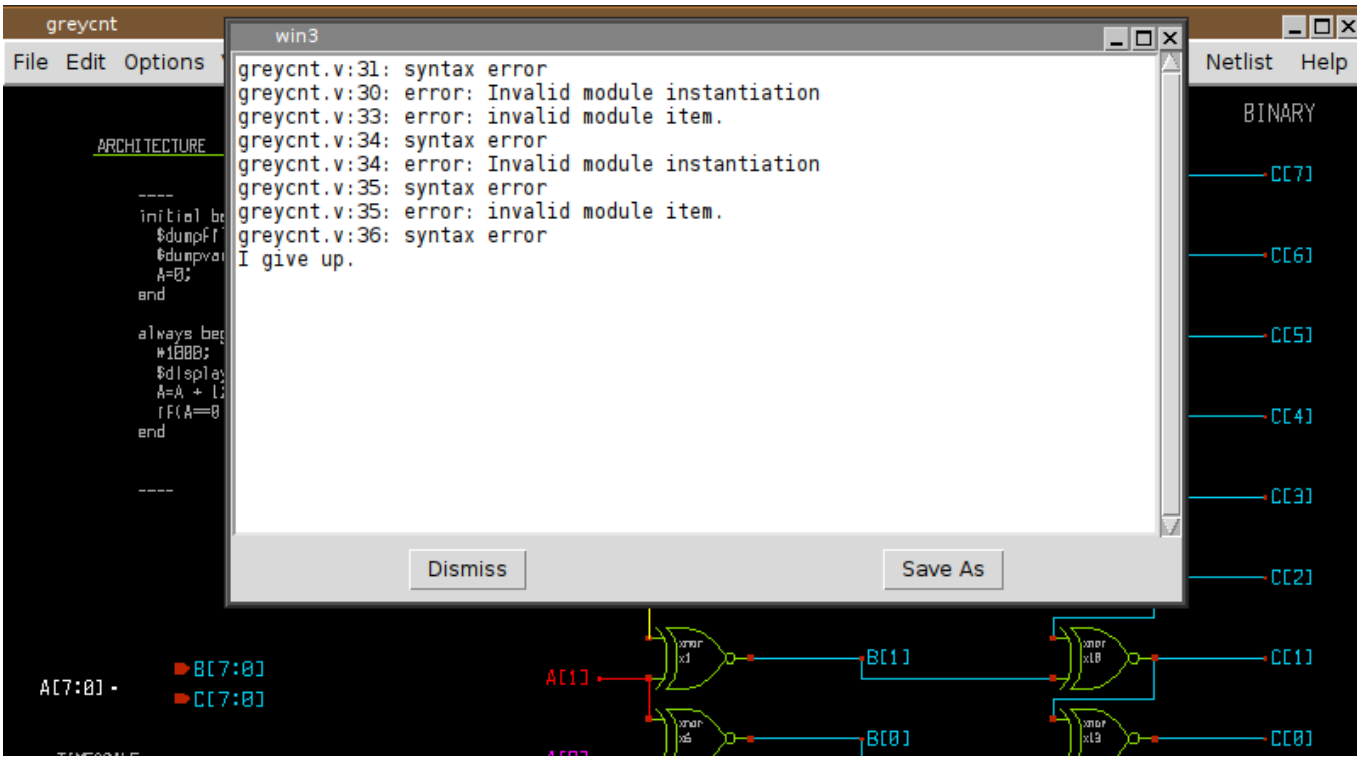
you will recognize the behavioral code right after the netlist specifying the connection of nets to the xnor and assign gates and all the necessary verilog declarations. If you press the **`Simulation`** button the **`Icarus Verilog`** simulator will be executed to compile (iverilog) and run (vvp) the 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'`**.



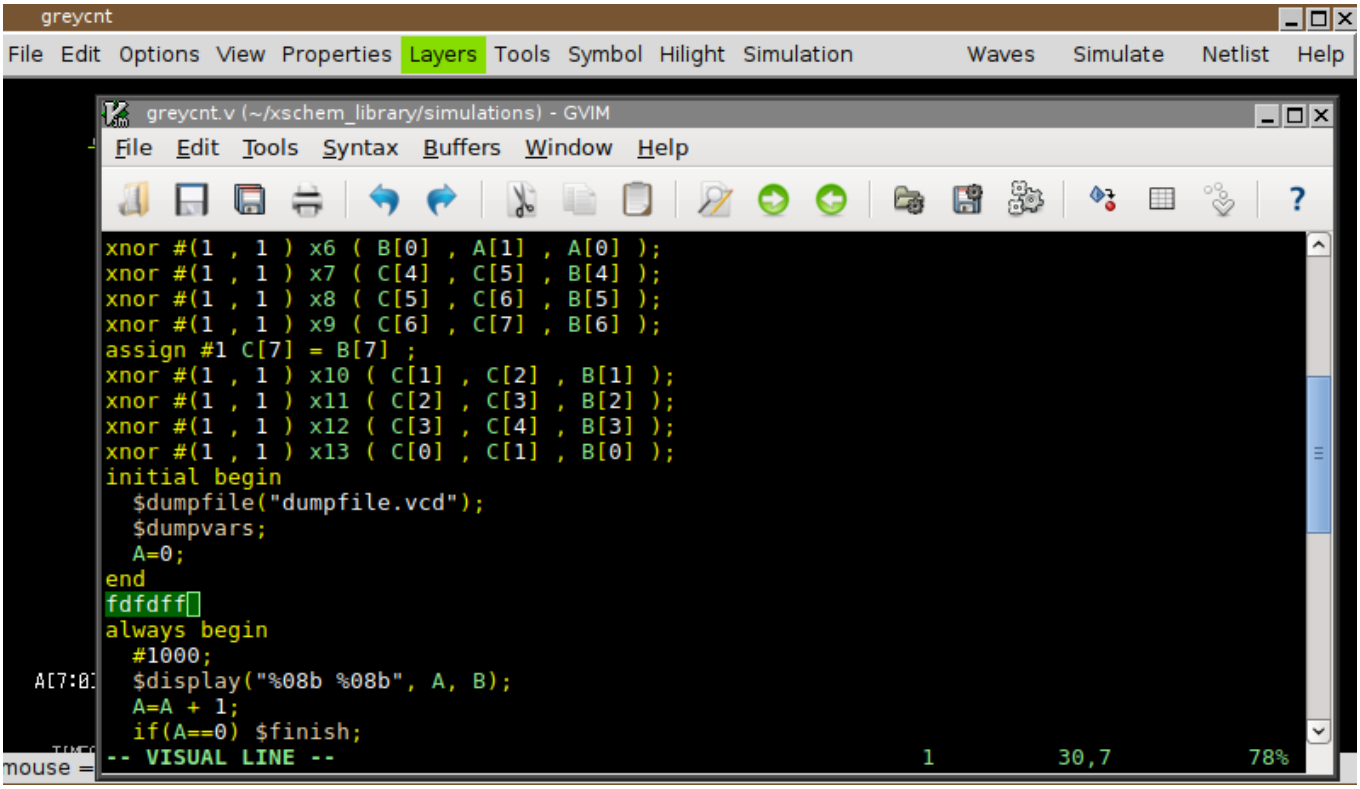

If simulation completes with no errors waveforms can be viewed. Press the **`Waves`** button in the top-right of the menu bar, you may add waveforms in the gtkwave window:



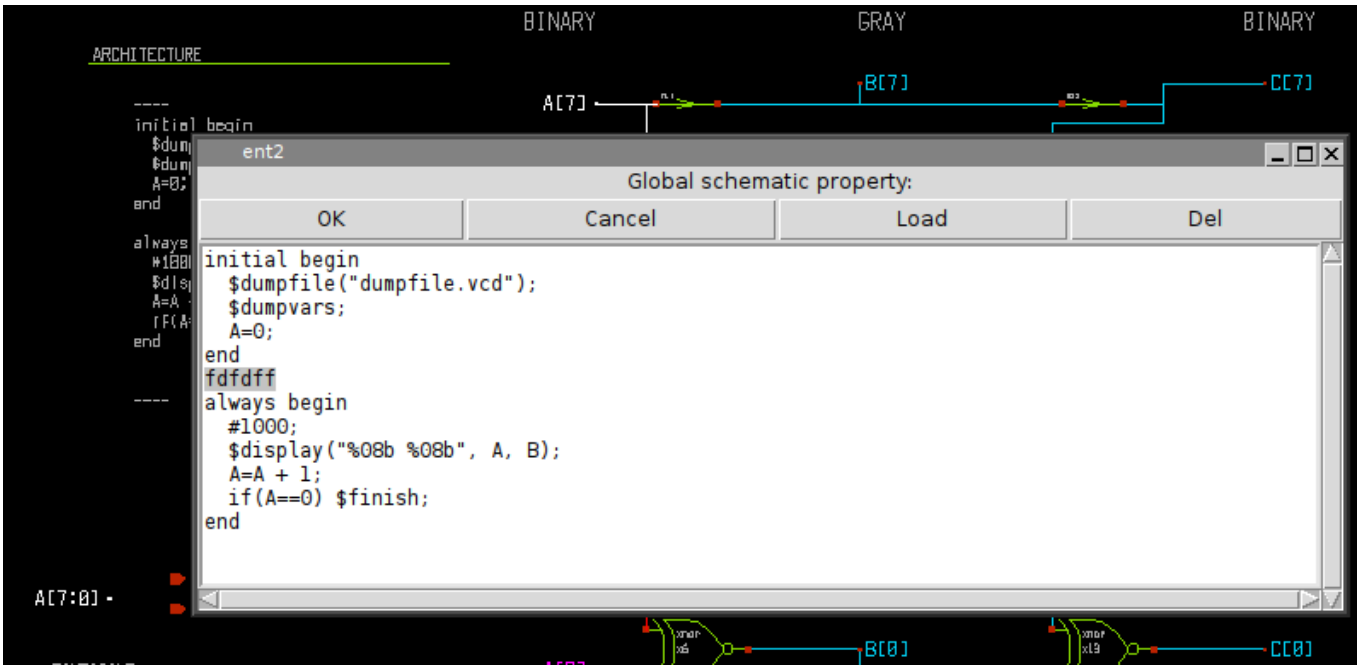

If the schematic contains errors that the simulator can not handle instead of the simulation log a window showing the error messages from the simulator is shown:




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:



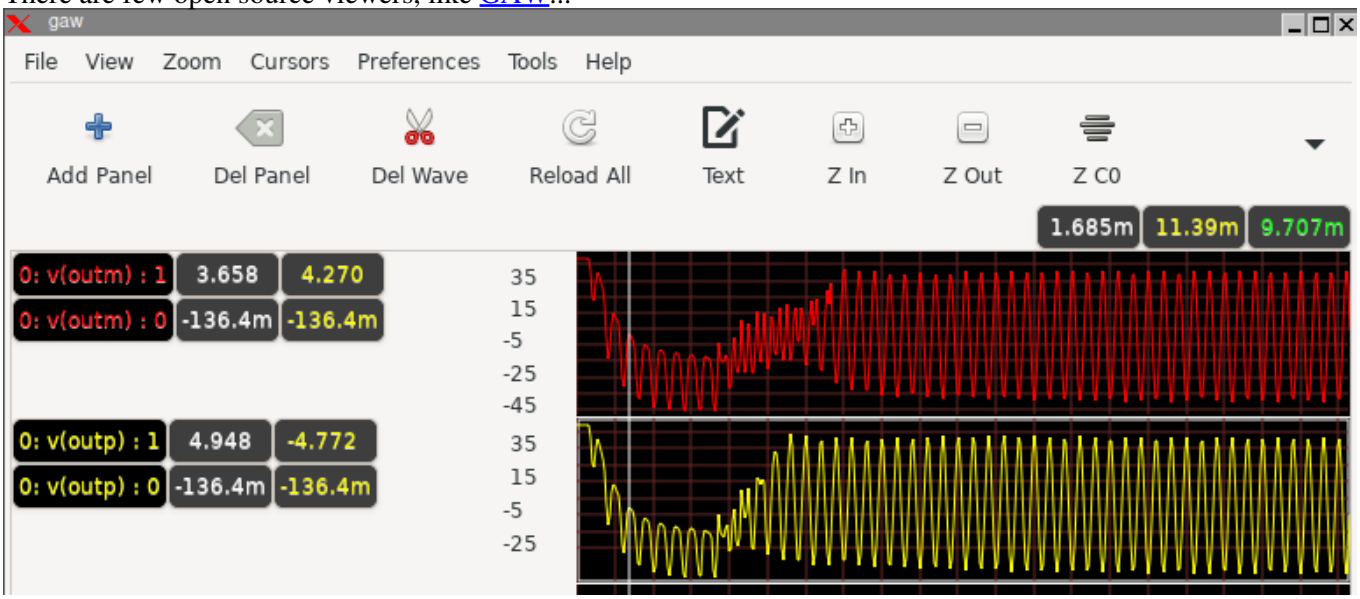

As you can see the error is in the behavioral code of the top level greycnt schematic, so edit the global property (**`'q'`** key with no component selected) and fix the error.


BINARY
GRAY
BINARY
ARCHITECTURE
A[7]
B[7]
C[7]
ent2
Global schematic property:
OK
Cancel
Load
Del
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
A[7:0]
B[0]
C[0]

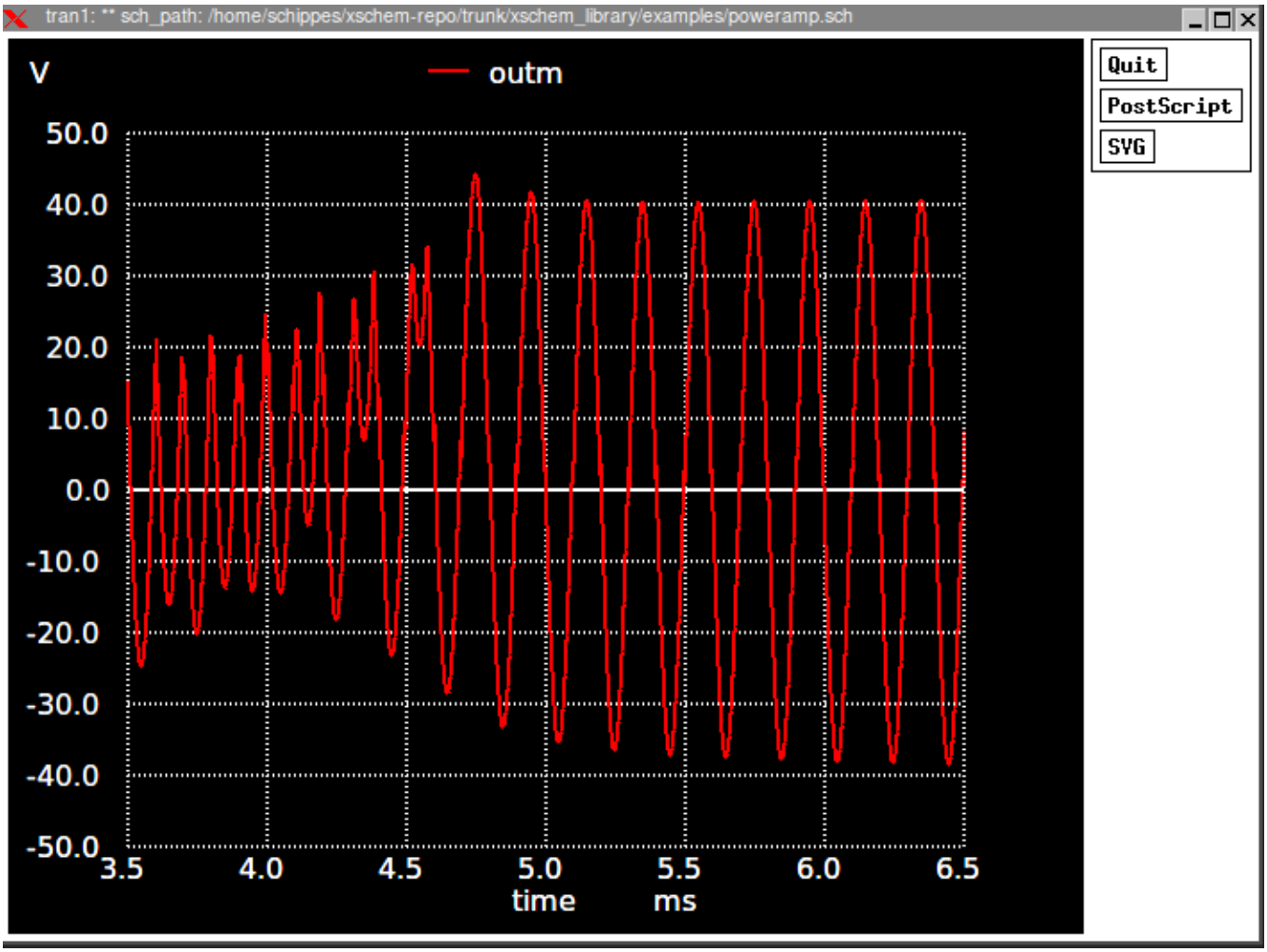
PREV UP NEXT

# VIEWING SIMULATION DATA WITH XSCHEM

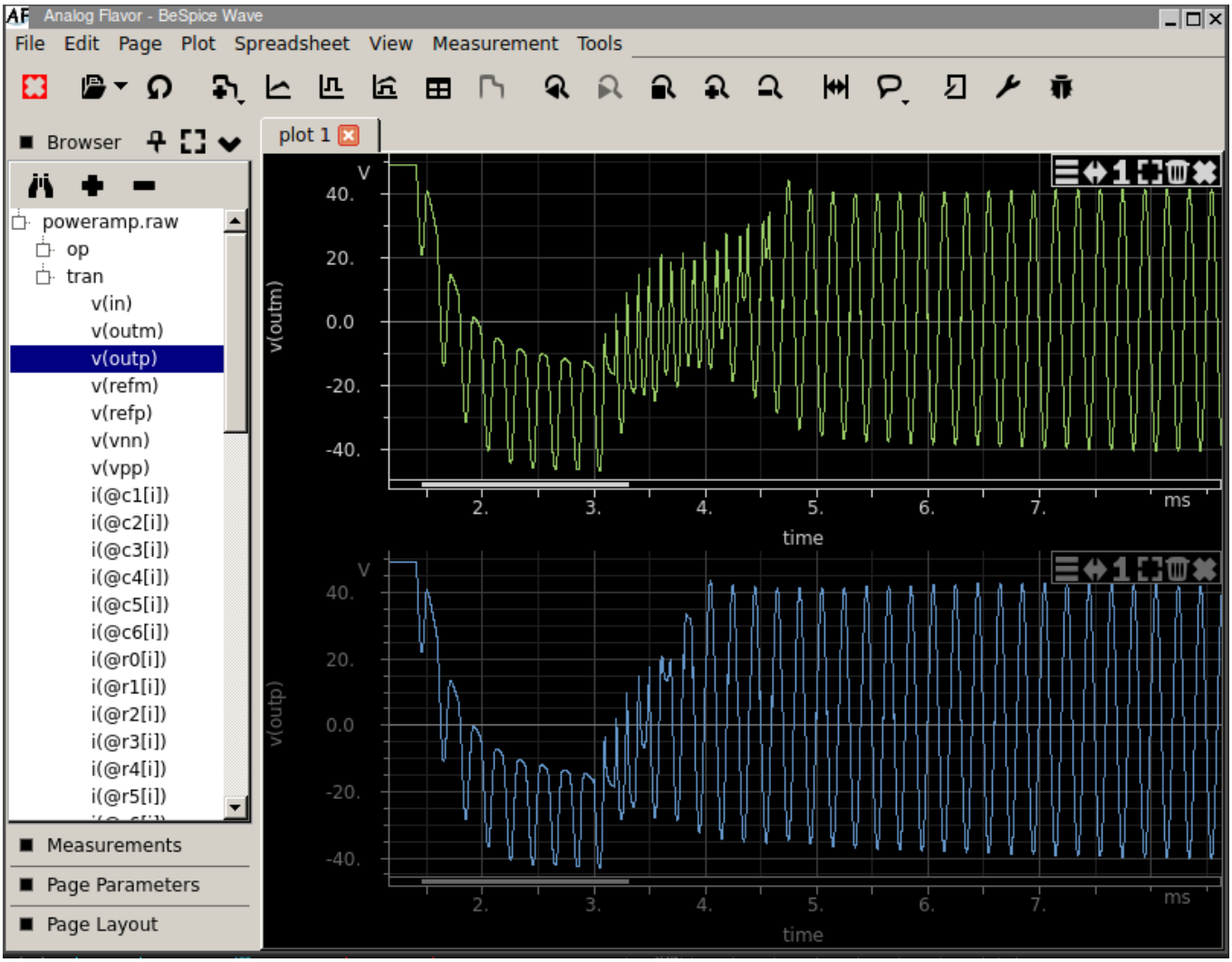
Usually when a spice simulation is done you want to see the results, this is usually accomplished with a waveform viewer. There are few open source viewers, like GAW...



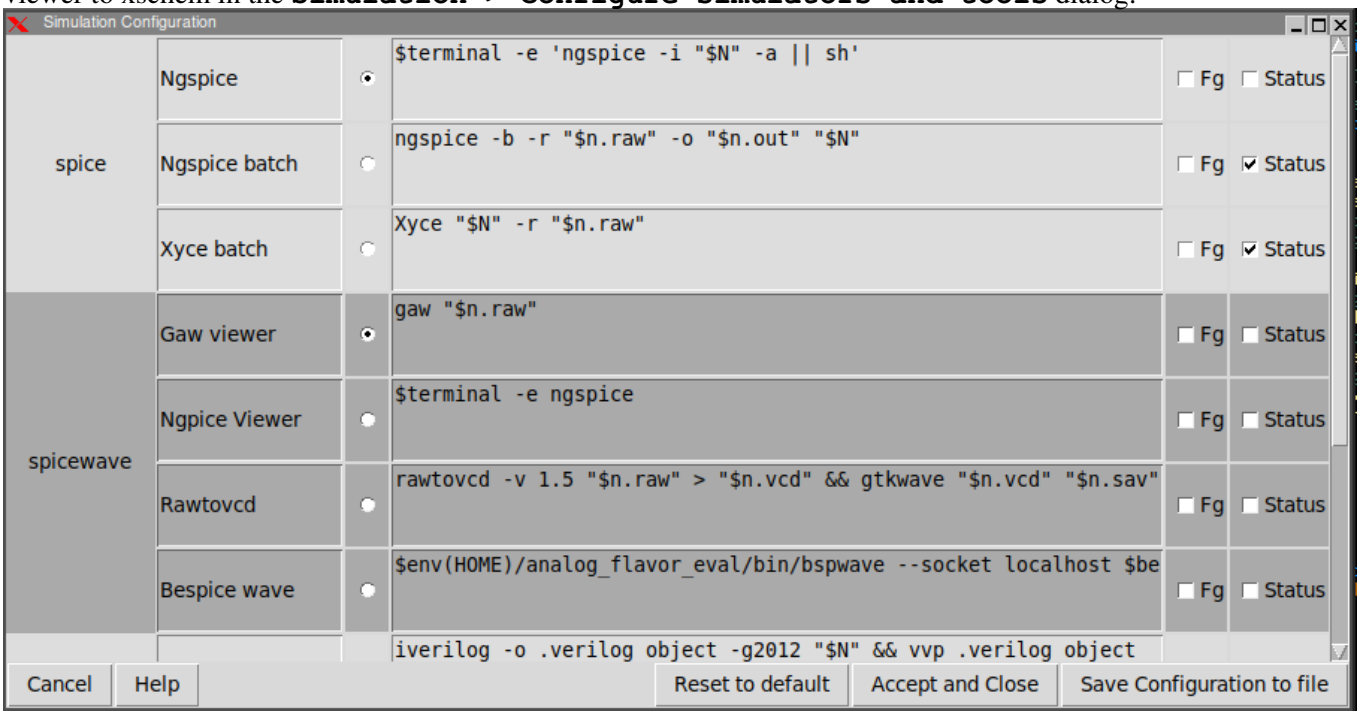

...Or ngspice internal plotting facilities:



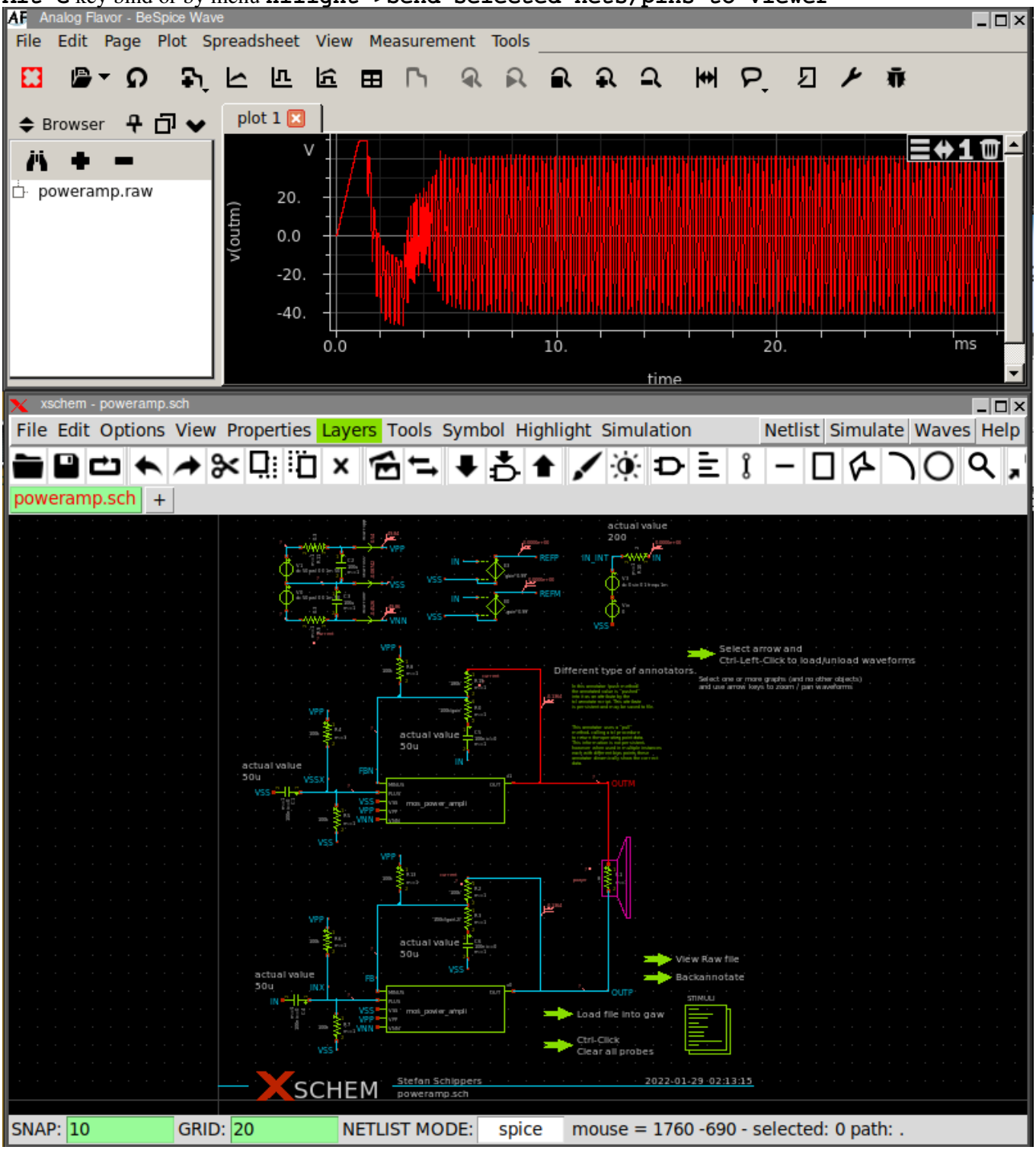

There is also an interesting commercial product from Analog Flavor, called BeSpice (bspwave) that offers a free of charge one year evaluation license for non commercial use:




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:

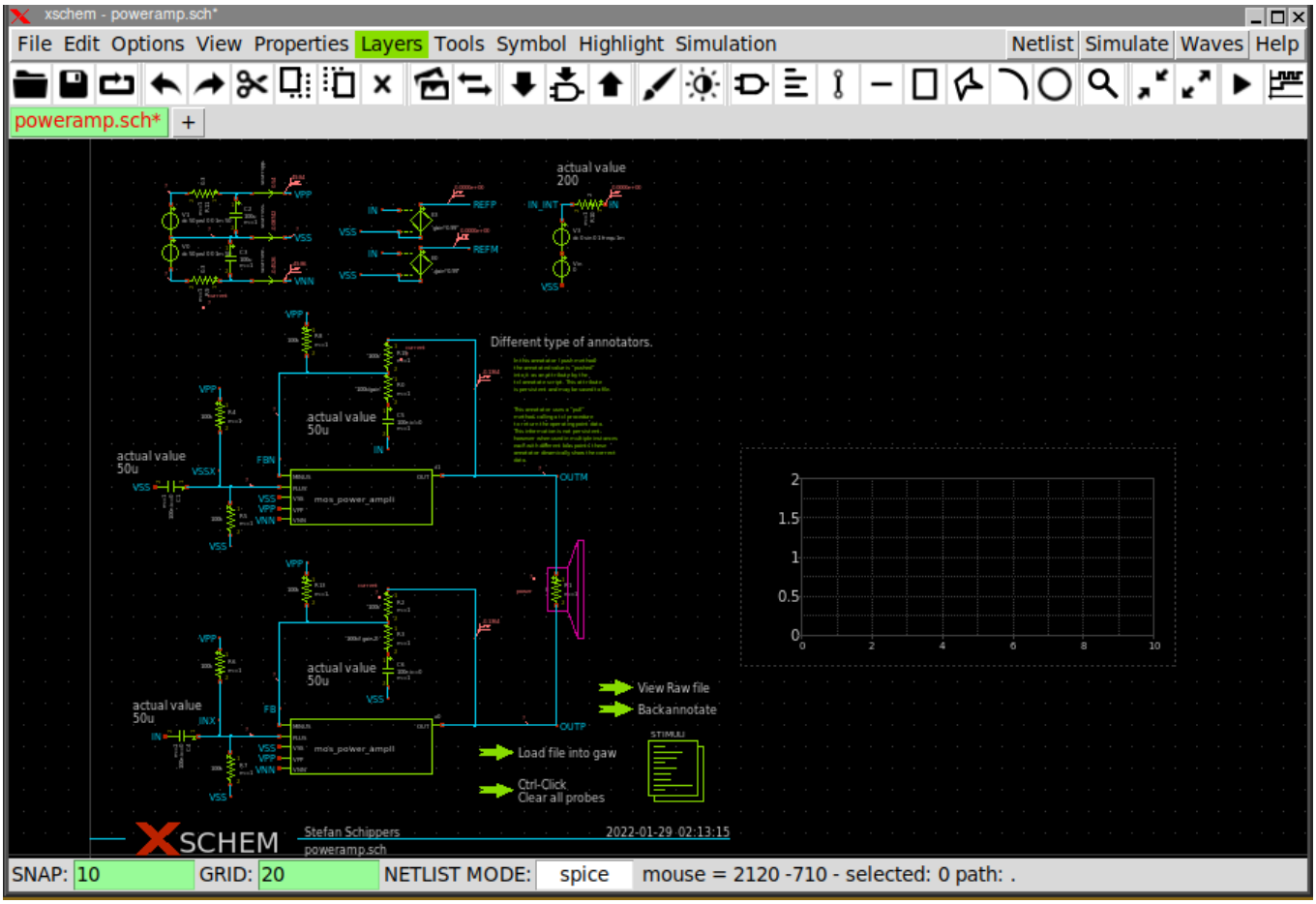


For gaw and bespice xschem can automatically send nets to the viewer by clicking a net on the schematic and pressing the **`Alt-G`** key bind or by menu **`Hilight->Send selected nets/pins to Viewer`**




# Using XSCHEM's internal graph functions

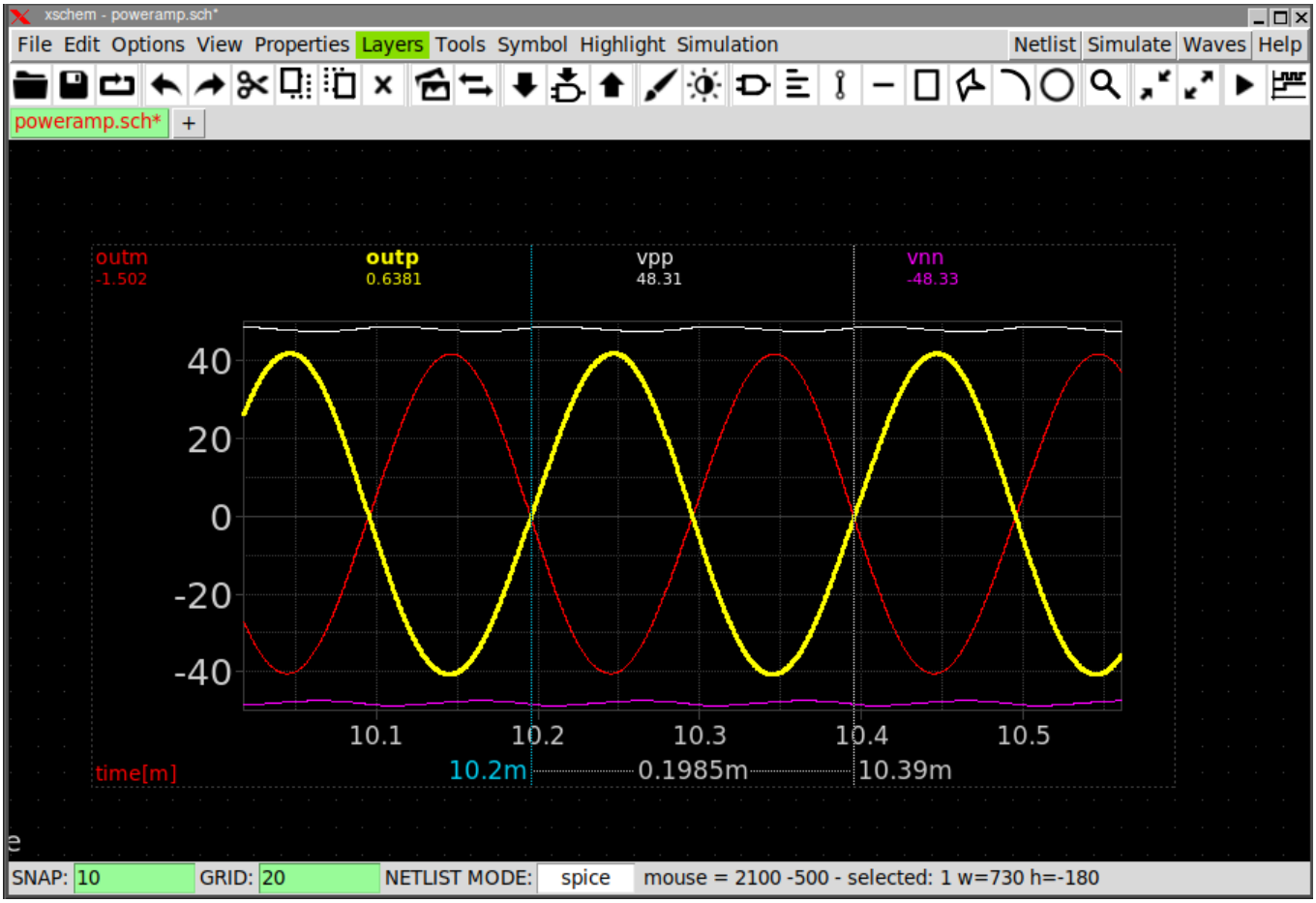
Xschem can now display waveforms by itself in the drawing area. in the Simulation menu there is an entry to add a graph: **`Add waveform graph`**. When this menu is pressed a box can be placed in the schematic:



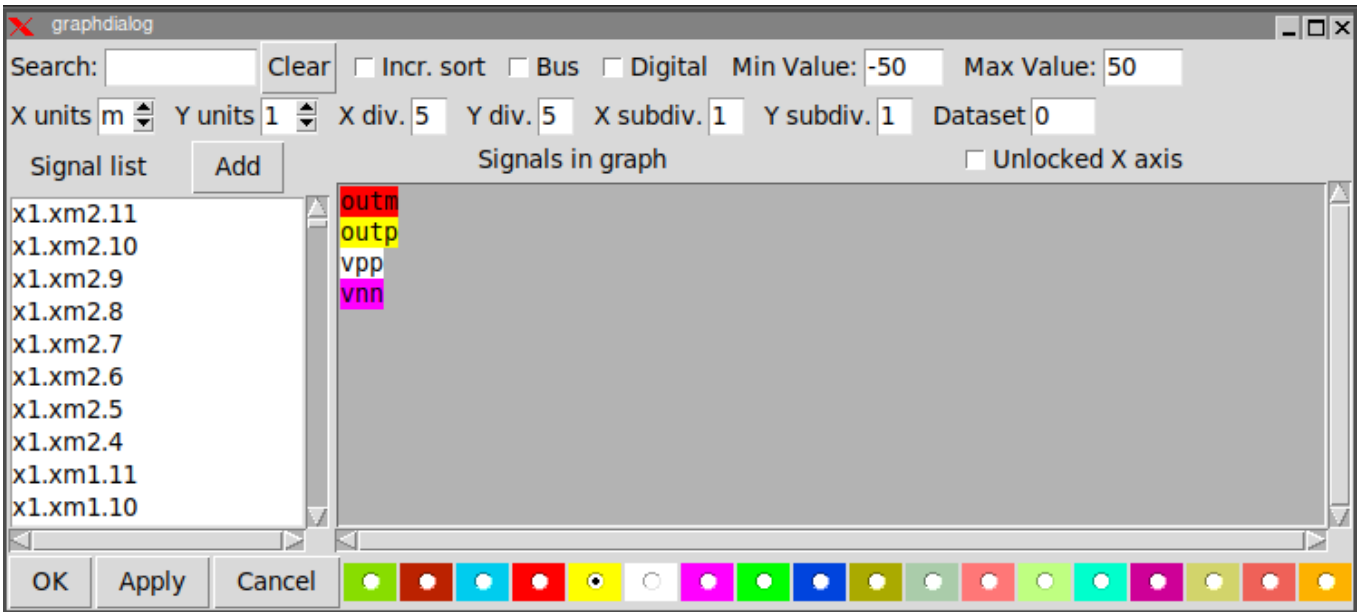

The next step is loading the simulation data, This is done by menu **`Simulation->Load/Unload ngspice .raw file`**. This command loads a .raw file produced by a ngspice/Xyce simulation. The file name is expected to be **`circuit.raw`** where **`circuit.sch`** is the name of the schematic opened in the drawing area. The raw file is searched for 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.



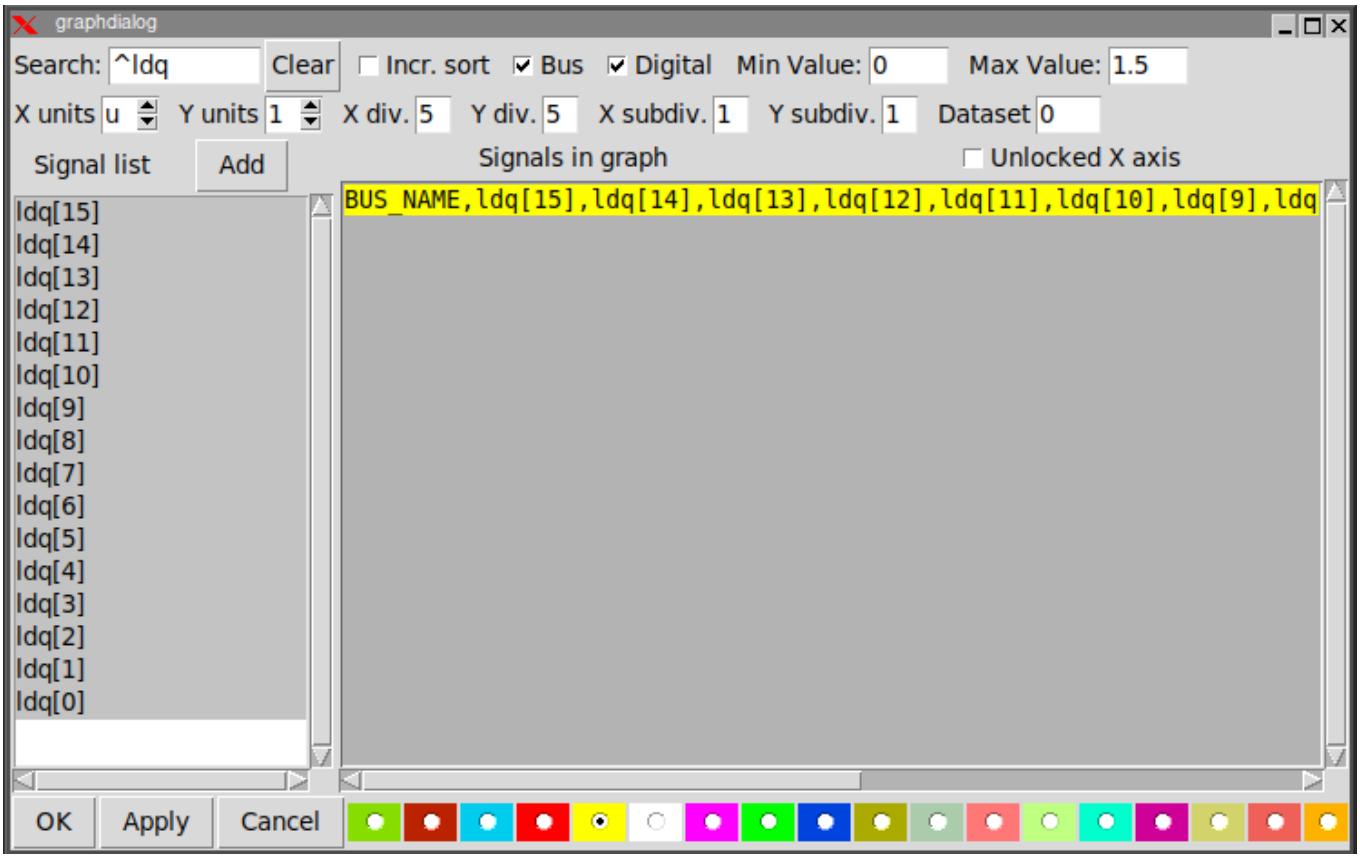

The graph configuration dialog box which is shown by **left** button double clicking inside the graph, allows to change many graph attributes, like number of X/Y labels, minor ticks, wave colors, add waves from the list of waves found in the raw file, select the dataset to show in case of multiple sweep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



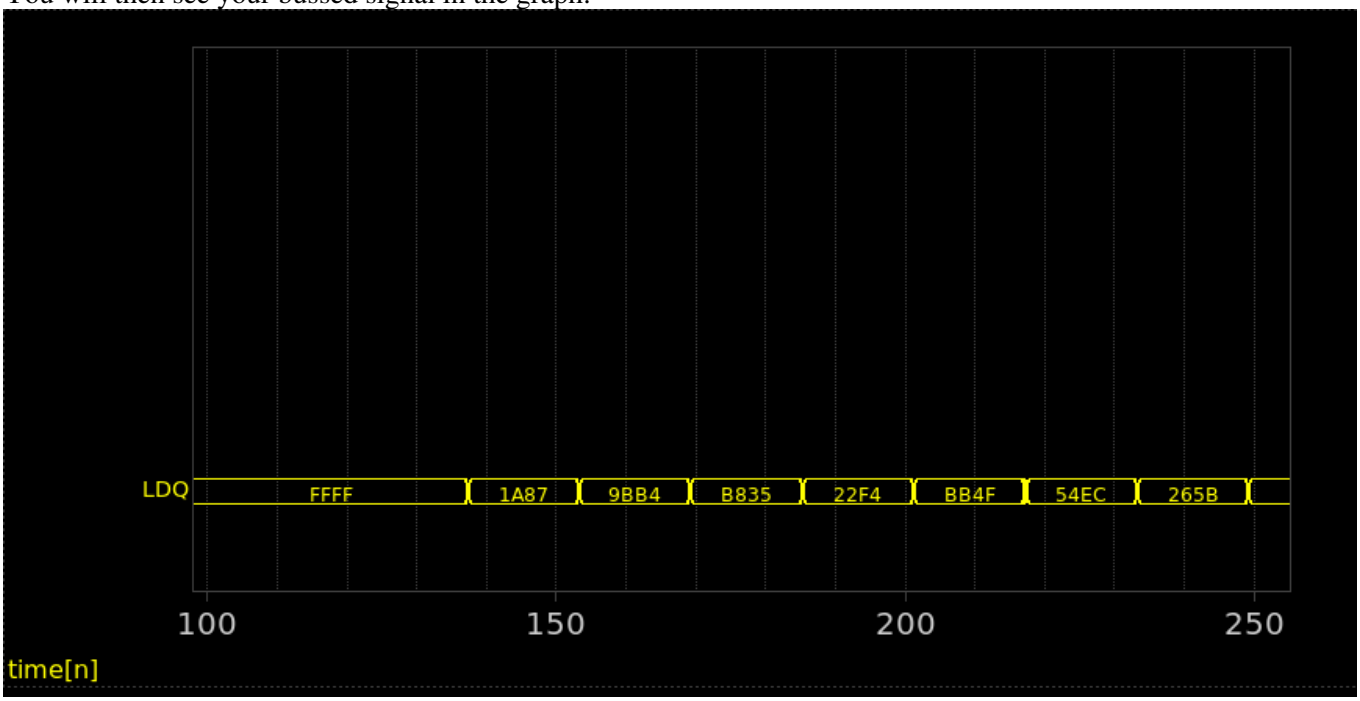

The text area with the colored wave names is just a text widget. You can manually edit it to add / remove waves, or you can place the cursor somewhere in the text, select some waves from the listbox on the left, press the **Add** button to have these waves added. If you place the insertion cursor in 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.

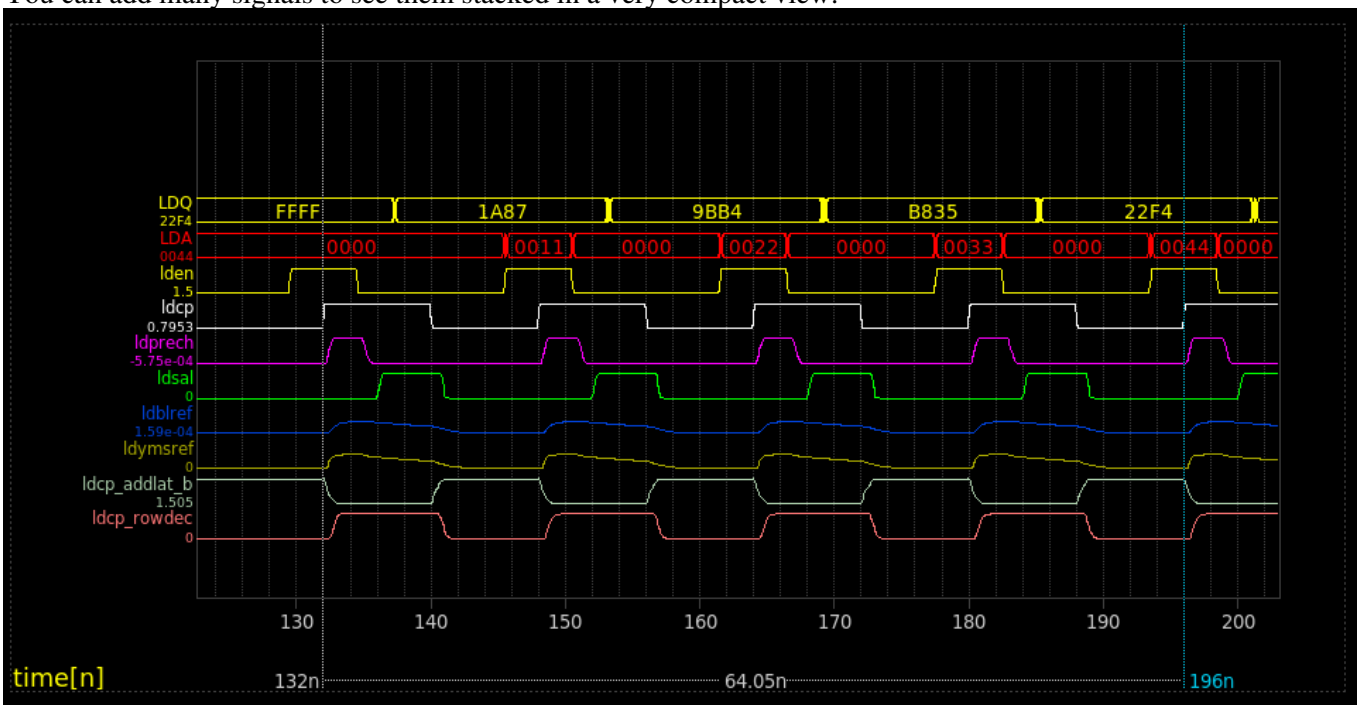



You will then see your bussed signal in the graph:




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`**:

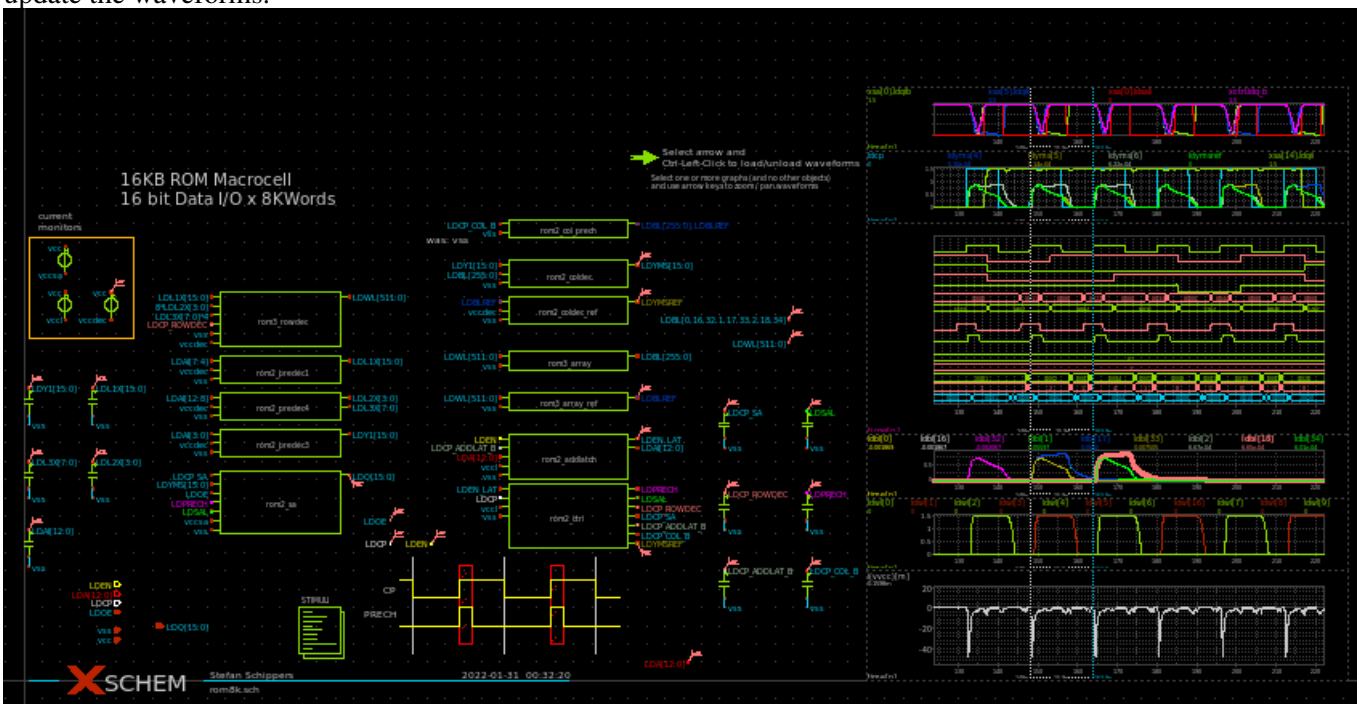

You can add many signals to see them stacked in a very compact view:




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.

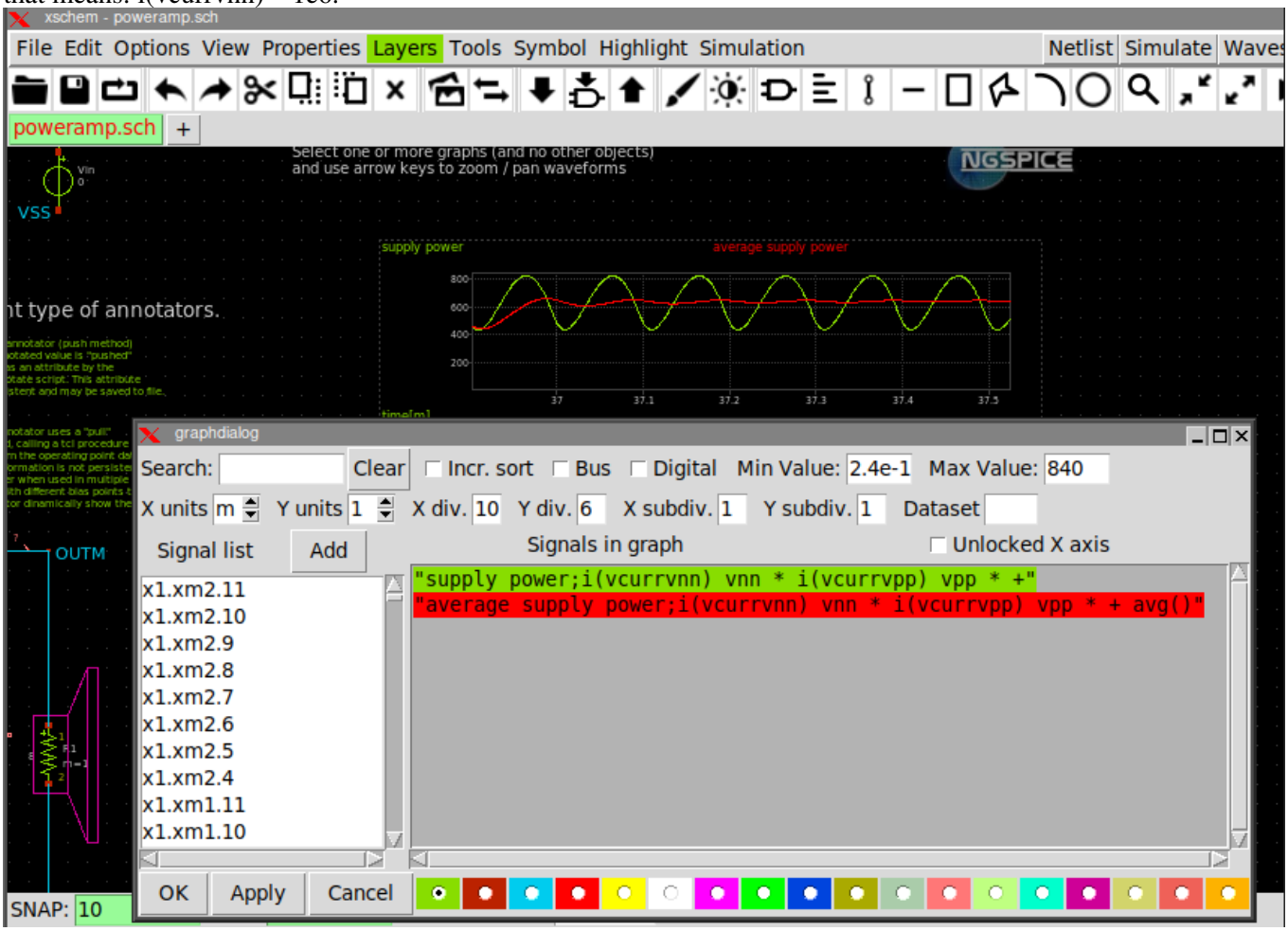

Many graphs can be created in a schematic, and the configuration of all graphs (viewport, list of signals, colors) is saved together with the schematic. If you re-run a simulation just unloading/loading the data from the simulation menu will update the waveforms.




## 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.




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

- **+** Addition
- **–** Subtraction
- ***** Multiplication
- **/** Division
- ****** Exponentiation
- **exch()** Exchange top 2 operands on stack

1 argument operators:

- **sin()** Trig. sin function
- **cos()** Trig. cos function
- **tan()** Trig. tan function
- **sqrt()** Square root
- **sgn()** Sign

- **`abs()`** Absolute value
- **`exp()`** Base-e Exponentiation
- **`ln()`** Base-e logarithm
- **`log10()`** Base 10 logarithm
- **`avg()`** Moving average
- **`deriv()`** Derivative
- **`integ()`** Integration
- **`dup()`** Duplicate last element on stack

PREV UP NEXT

# 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 predefiend 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

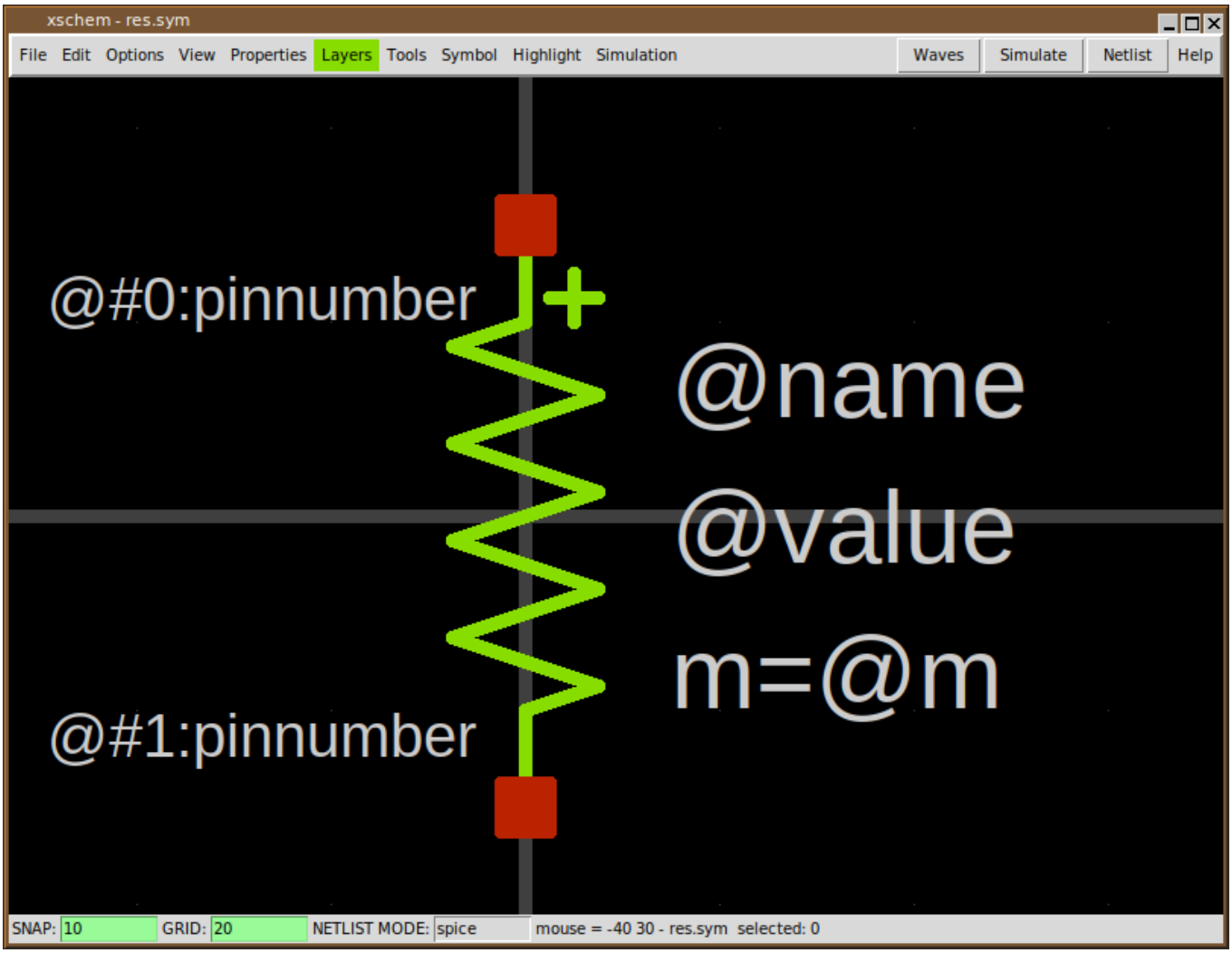
....

20. General purpose
21. General purpose

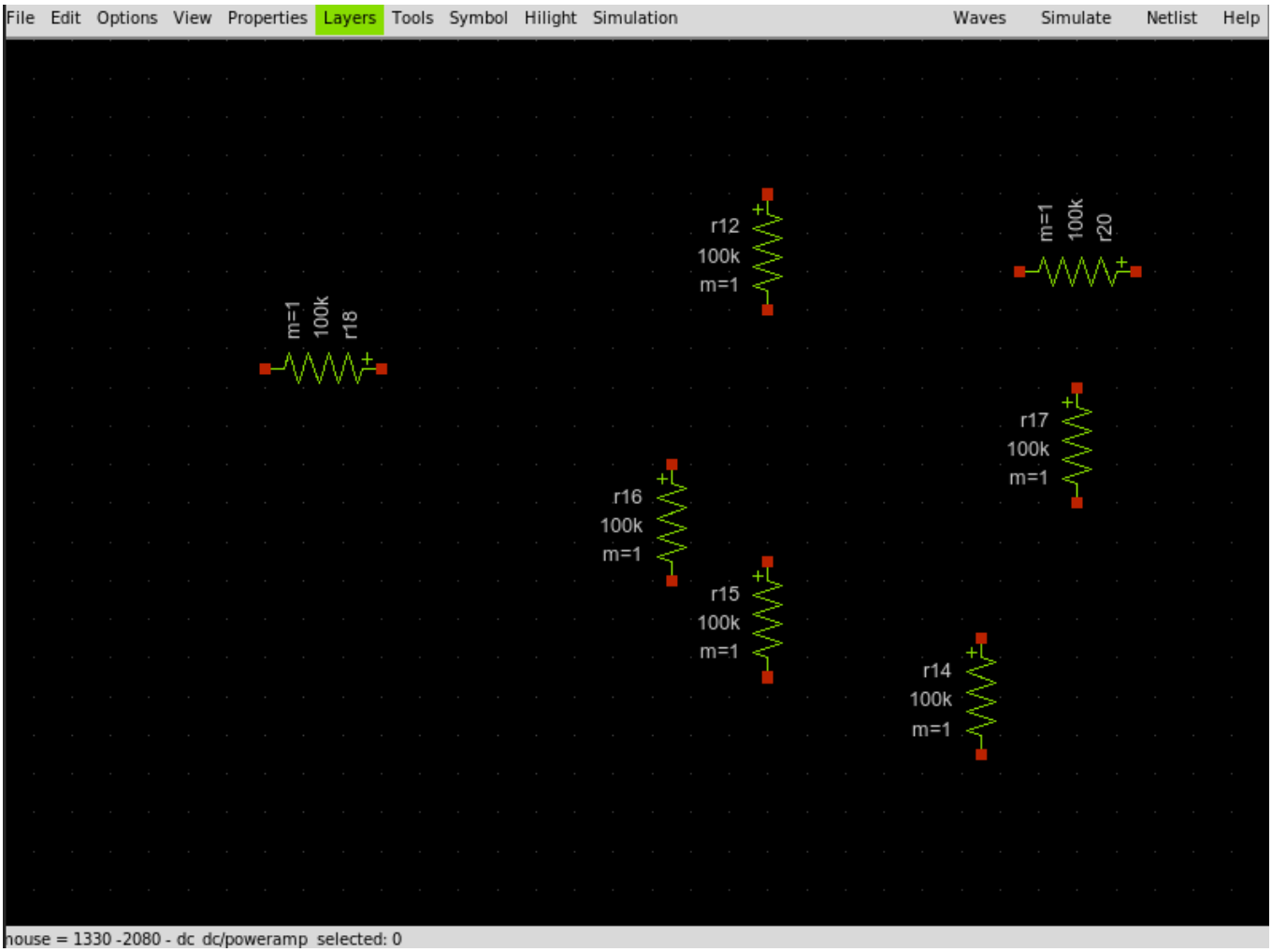
Although any layer can be used for drawing it is strongly advisable to avoid the background color and the selection color to avoid confusion. Drawing begins by painting the background (layer 0), then drawing the grid (layer 1) then drawing wires (nets) on layer 2, then all graphical objects (lines, rectangles,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




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.

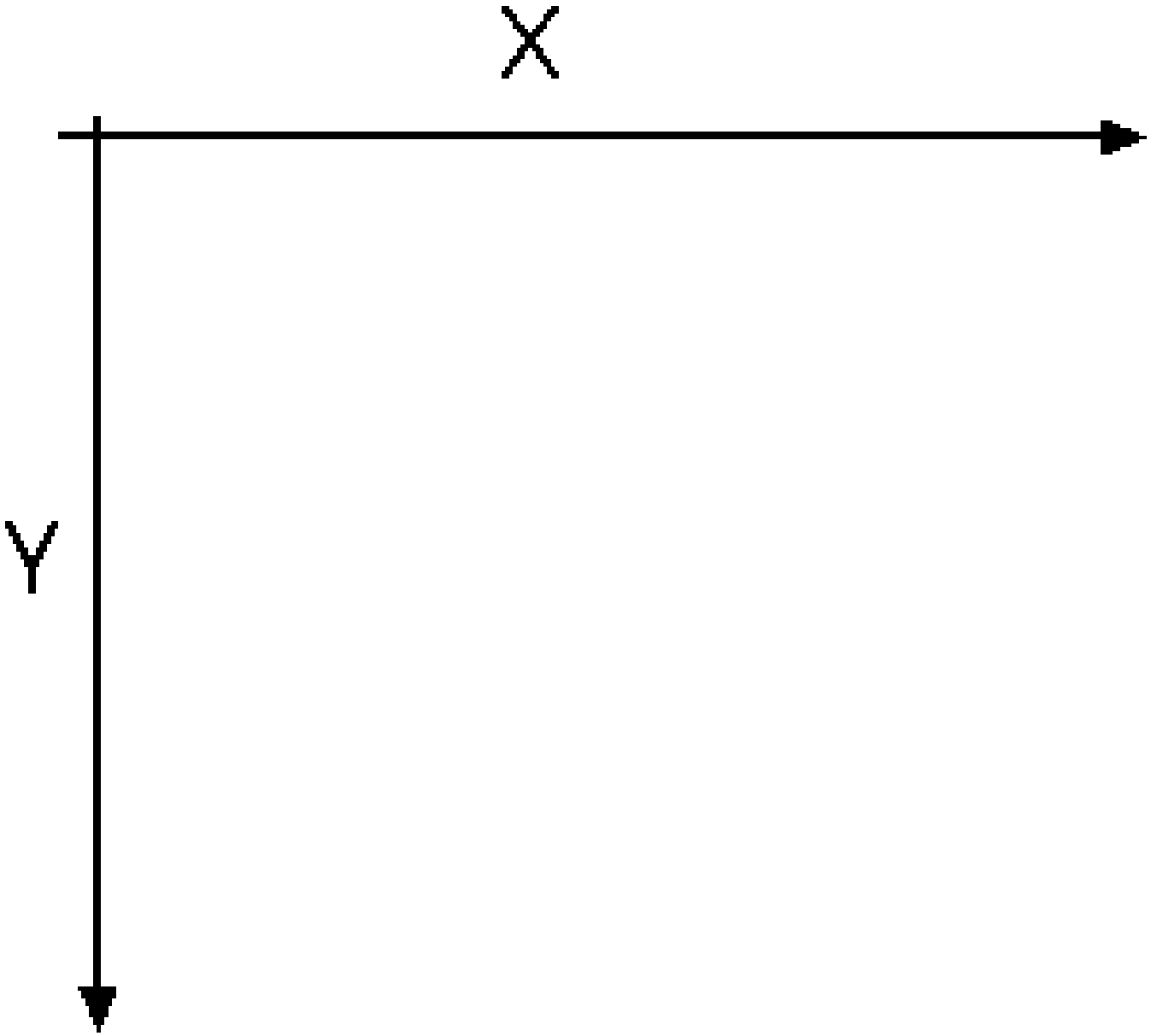
## PROPERTIES

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 objecs 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}
[
G {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:
```
G {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:

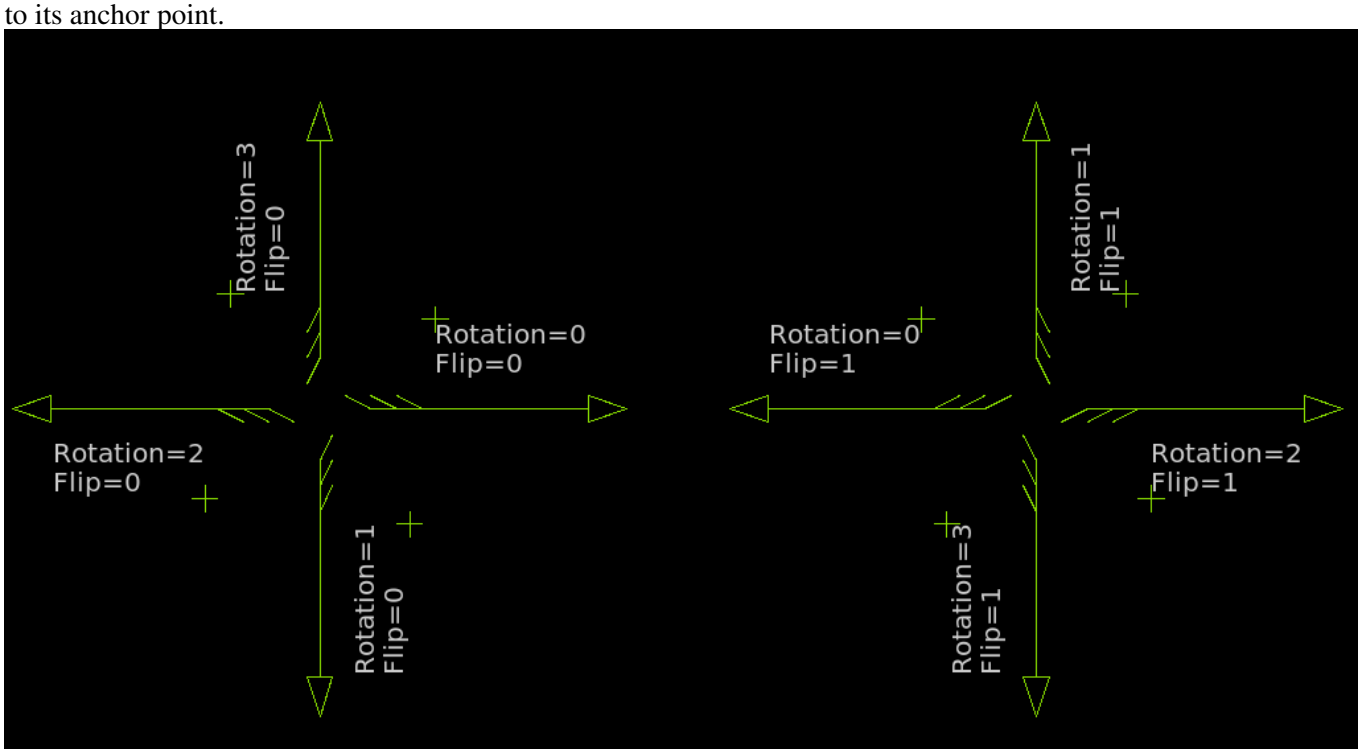
```
C {capa.sym} 890 -160 0 0 {name=C4
m=1 value=10u
device="tantalium capacitor"
}
```

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 stored with the same syntax as the displayed text field. 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

## WIRE OBJECT

Example: **N 890 -130 890 -110 {lab=ANALOG_GND}**
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}**
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

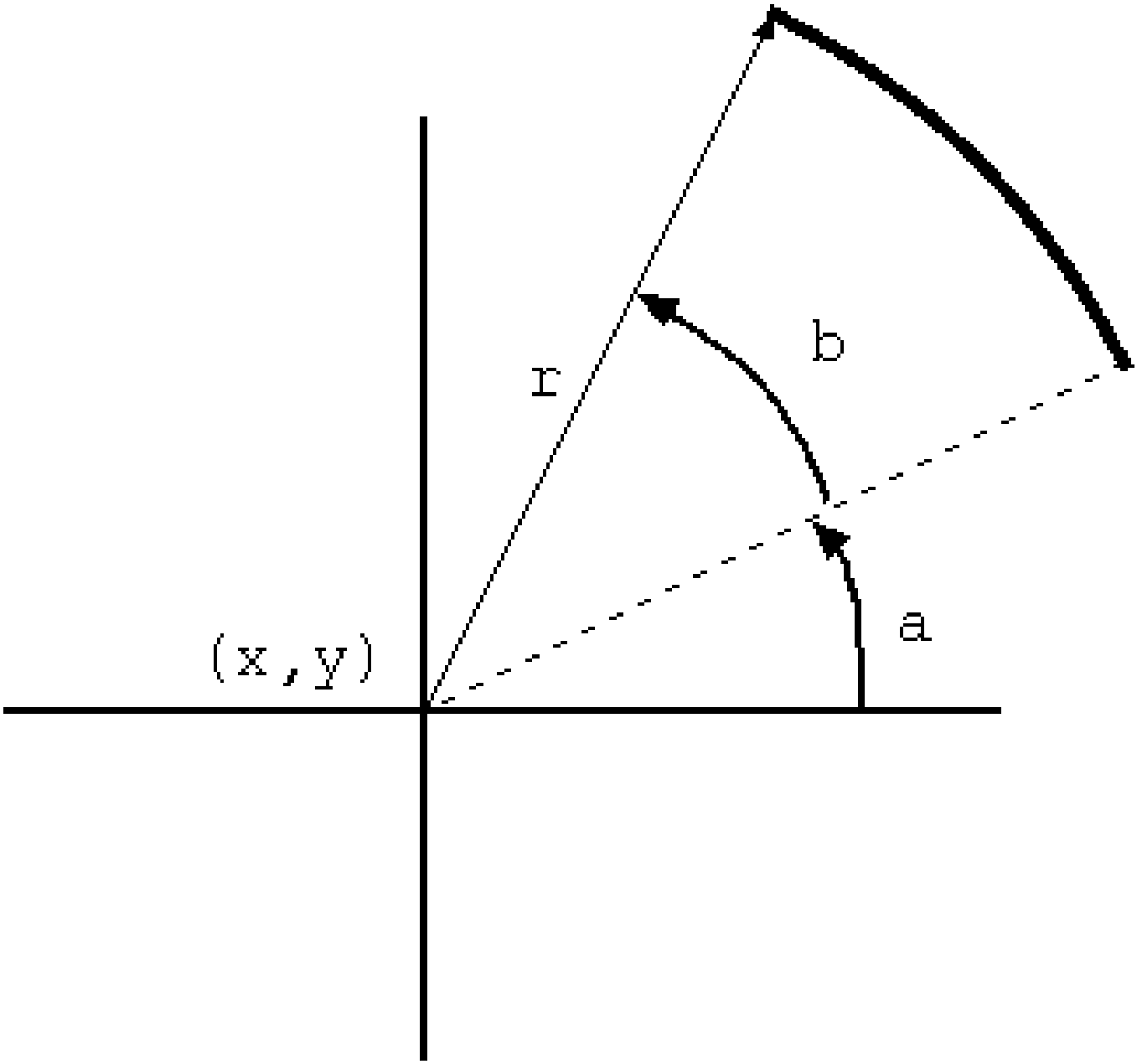
Example: **B 5 -62.5 -2.5 -57.5 2.5 {name=IN dir=in pinnumber=1}**
The 'Box' 'B' tag is followed by an integer specifying the graphic layer followed by the x1,y1 - x2,y2 coordinates of the rectangle and a final property string. This example defines a symbol pin.

## OPEN / CLOSED POLYGON OBJECT

Example: **P 3 5 2450 -210 2460 -170 2500 -170 2510 -210 2450 -210 {}**
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' arribute 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}**

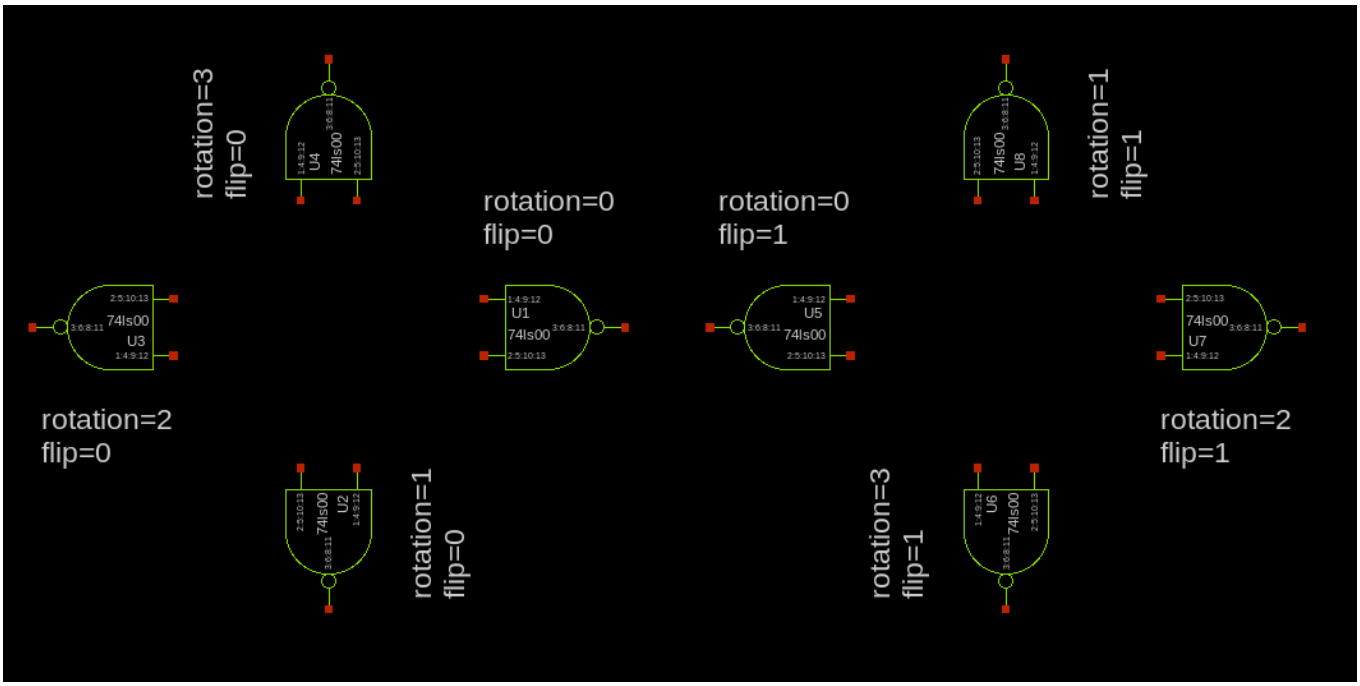
## ARC OBJECT

Example: **A 3 450 -210 120 45 225 {}**
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.



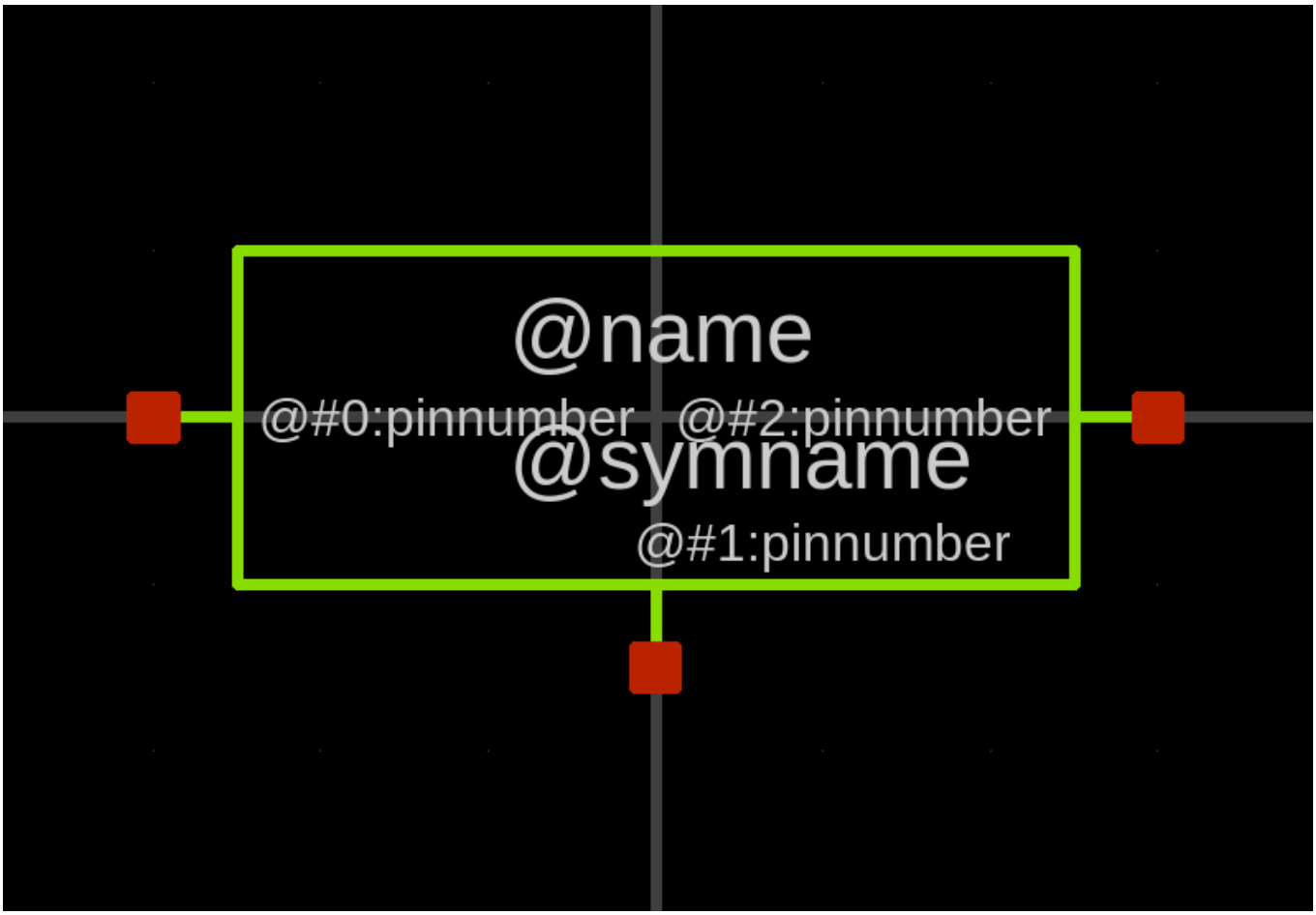

## COMPONENT INSTANCE

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)

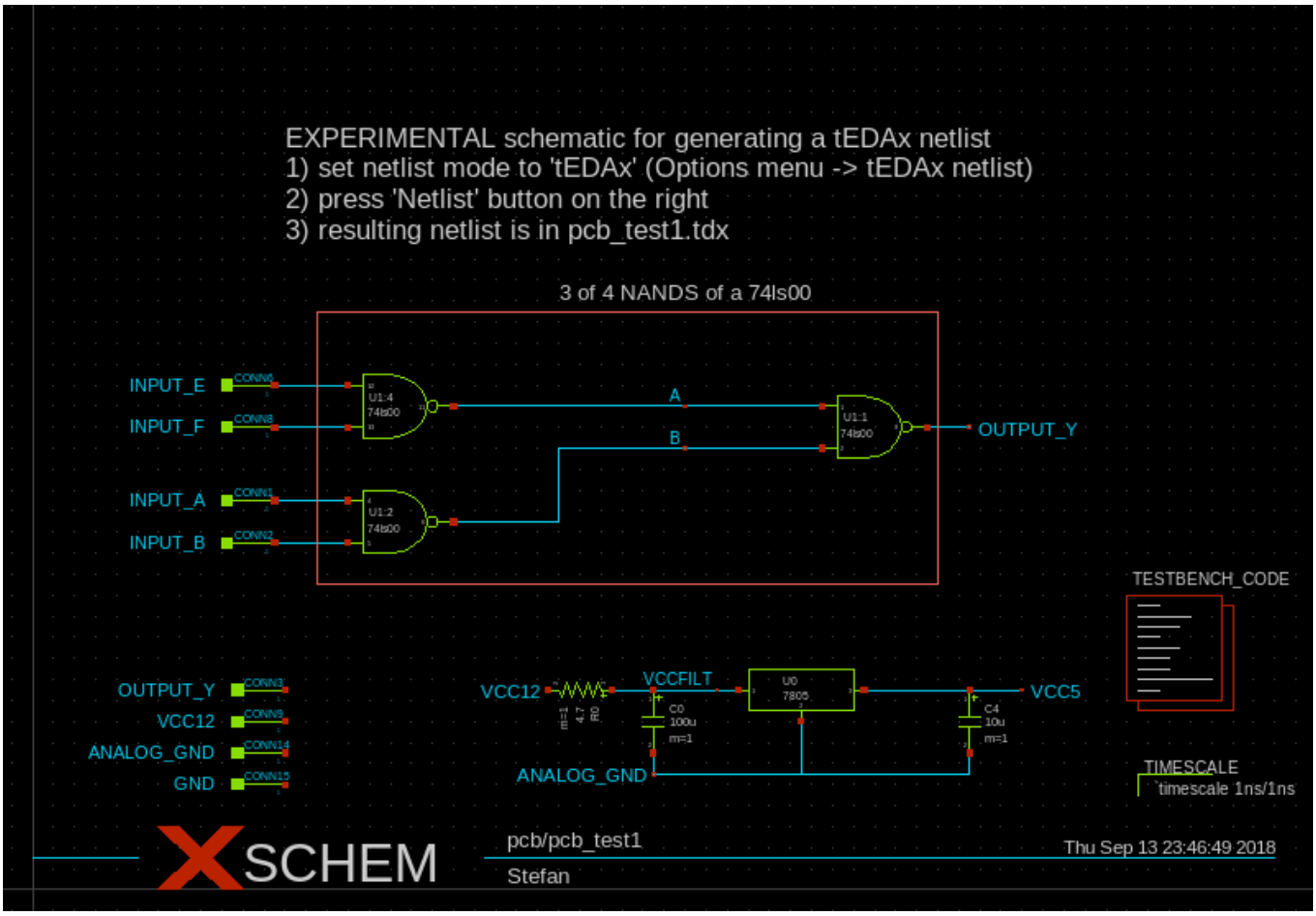
```
G {type=regulator
format="x@name @pinlist r@symname"
verilog_format="assign @#2 = @#0 ;"
tedax_format="footprint @name @footprint
device @name @symname"
template="name=U1 footprint=TO220"}
V {}
S {}
E {}
L 4 -60 0 -50 0 {}
L 4 50 0 60 0 {}
L 4 -50 -20 50 -20 {}
L 4 50 -20 50 20 {}
L 4 -50 20 50 20 {}
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 {}
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
A
B
OUTPUT_Y
TESTBENCH_CODE
OUTPUT_Y
VCC12
ANALOG_GND
GND
VCCFILT
VCC5
TIMESCALE
timescale 1ns/1ns
XSCHEM
pcb/pcb_test1
Stefan
Thu Sep 13 23:46:49 2018

PREV UP NEXT

# XSCHEM REMOTE INTERFACE SPECIFICATION

## GENERAL INFORMATIONS

XSCHEM embeds a tcl shell, when running xschem the terminal will present a tcl prompt allowing to send / get 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.

UP

# 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** and xschem user configuration stored in user's home directory under **~/.xschem**:

```
schippes@mazinga:~/xschem_git$ ./configure --prefix=/usr/local --user-conf-dir=~/.xschem \
--user-lib-path=~/share/xschem/xschem_library \
--sys-lib-path=/usr/local/share/xschem/xschem_library
```

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

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
```

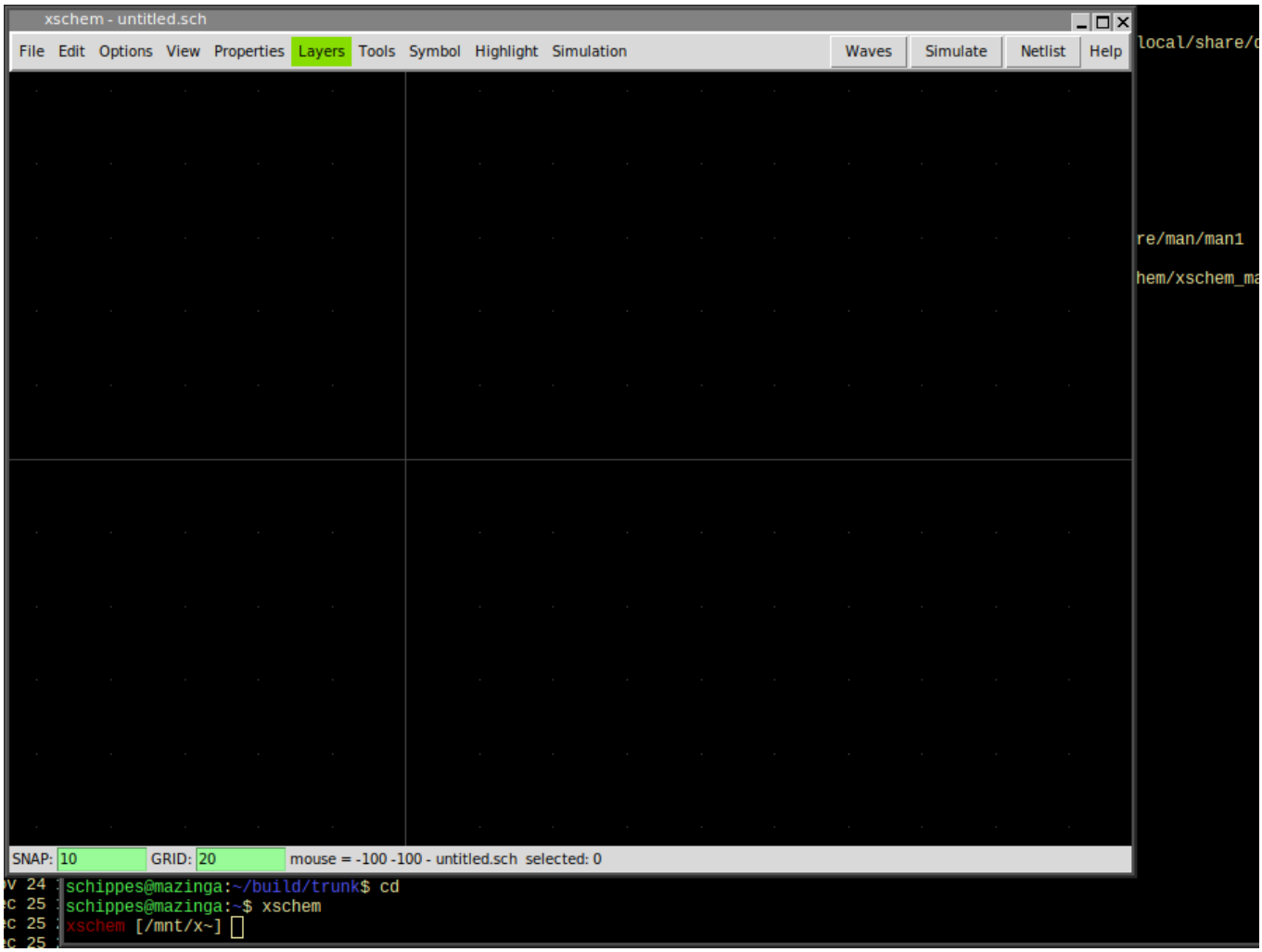
6. If compilation of source files completed with no errors xschem will be ready for installation:

```
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
```




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`**

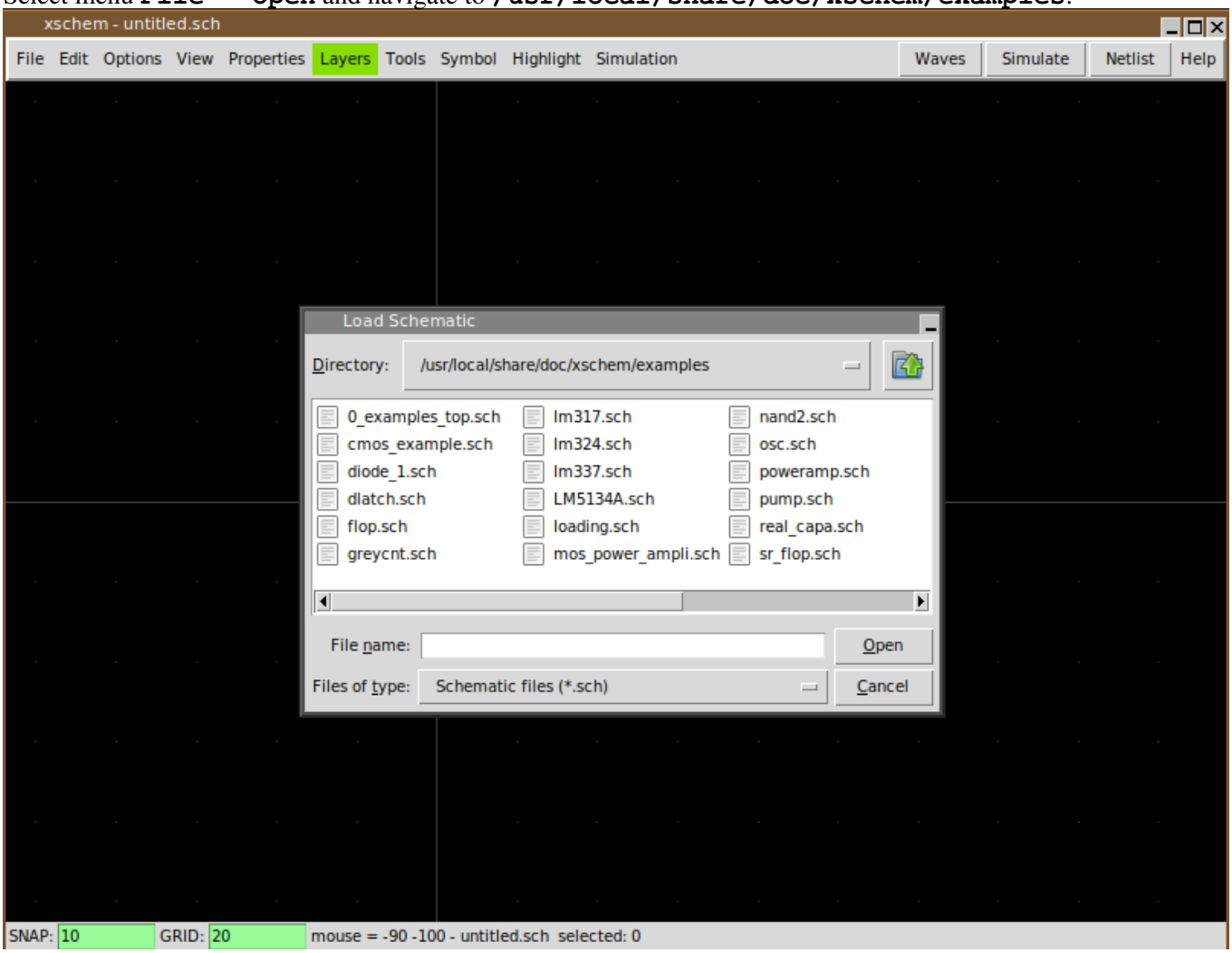
```
schippes@mazinga:~$ cp build/trunk/src/xschemrc ~/.xschem
```

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.

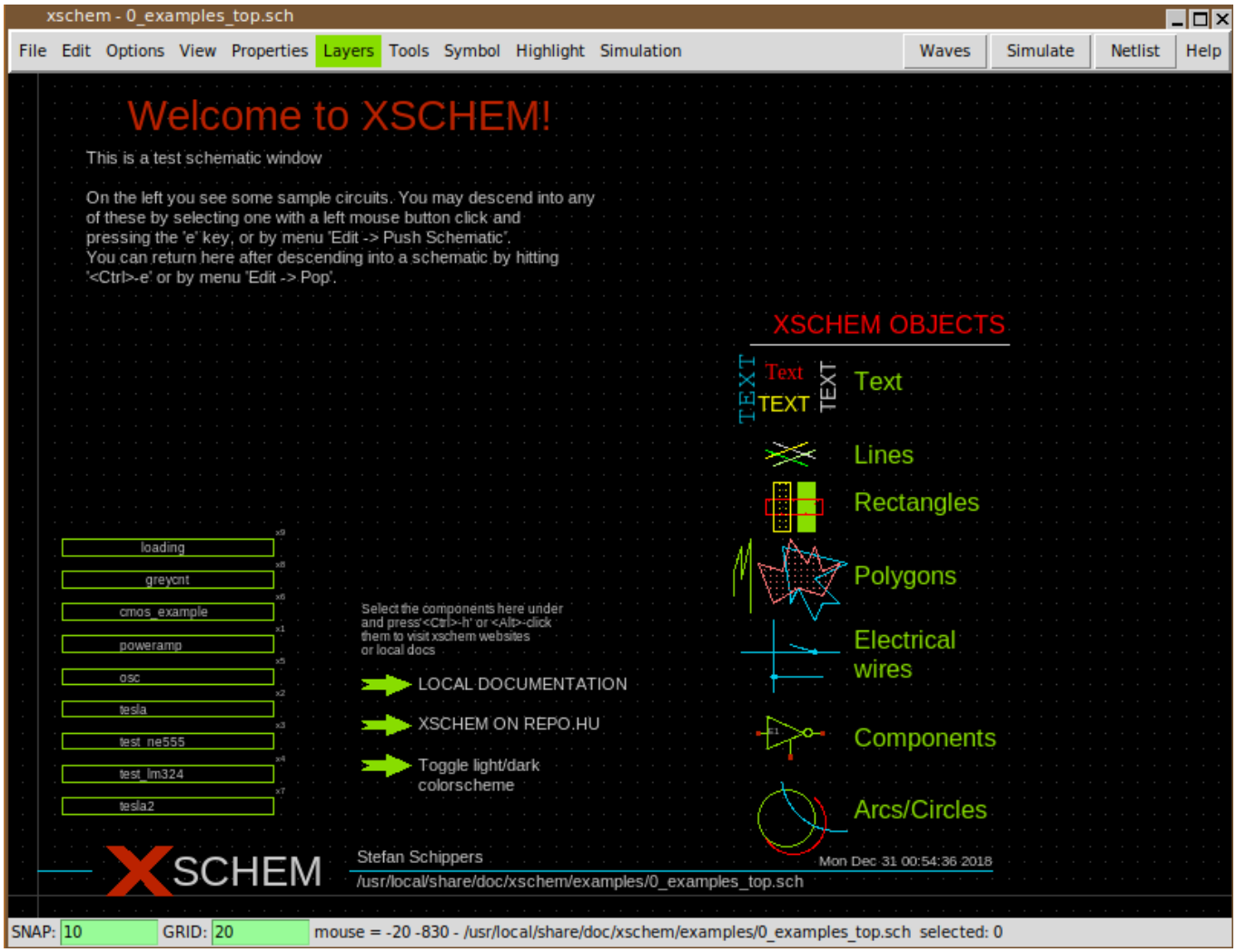
10. Run xschem again to try some schematic load tests:

```
schippes@mazinga:~$ xschem
```

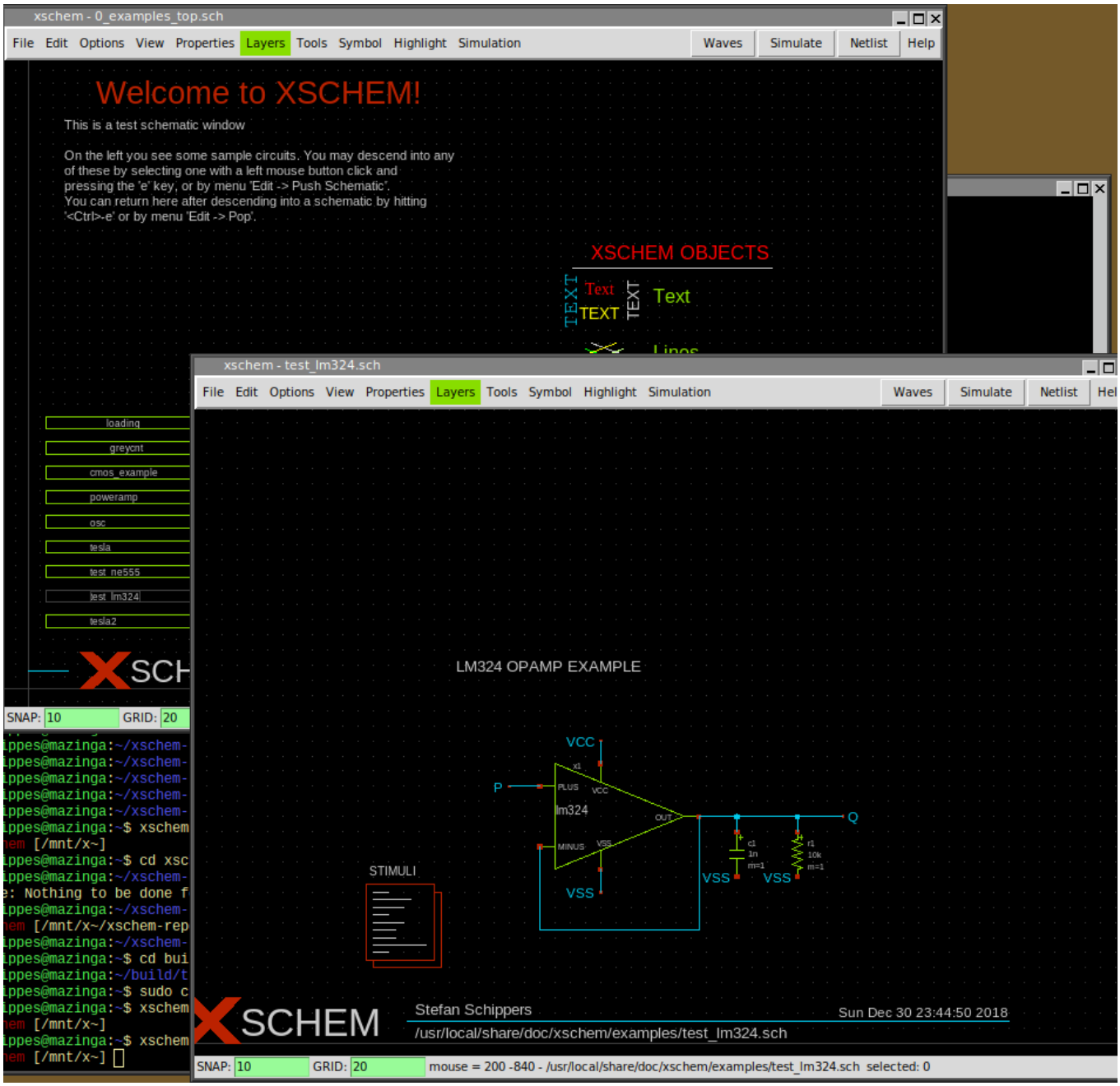
11. Select menu **`File - Open`** and navigate to **`/usr/local/share/doc/xschem/examples`**:



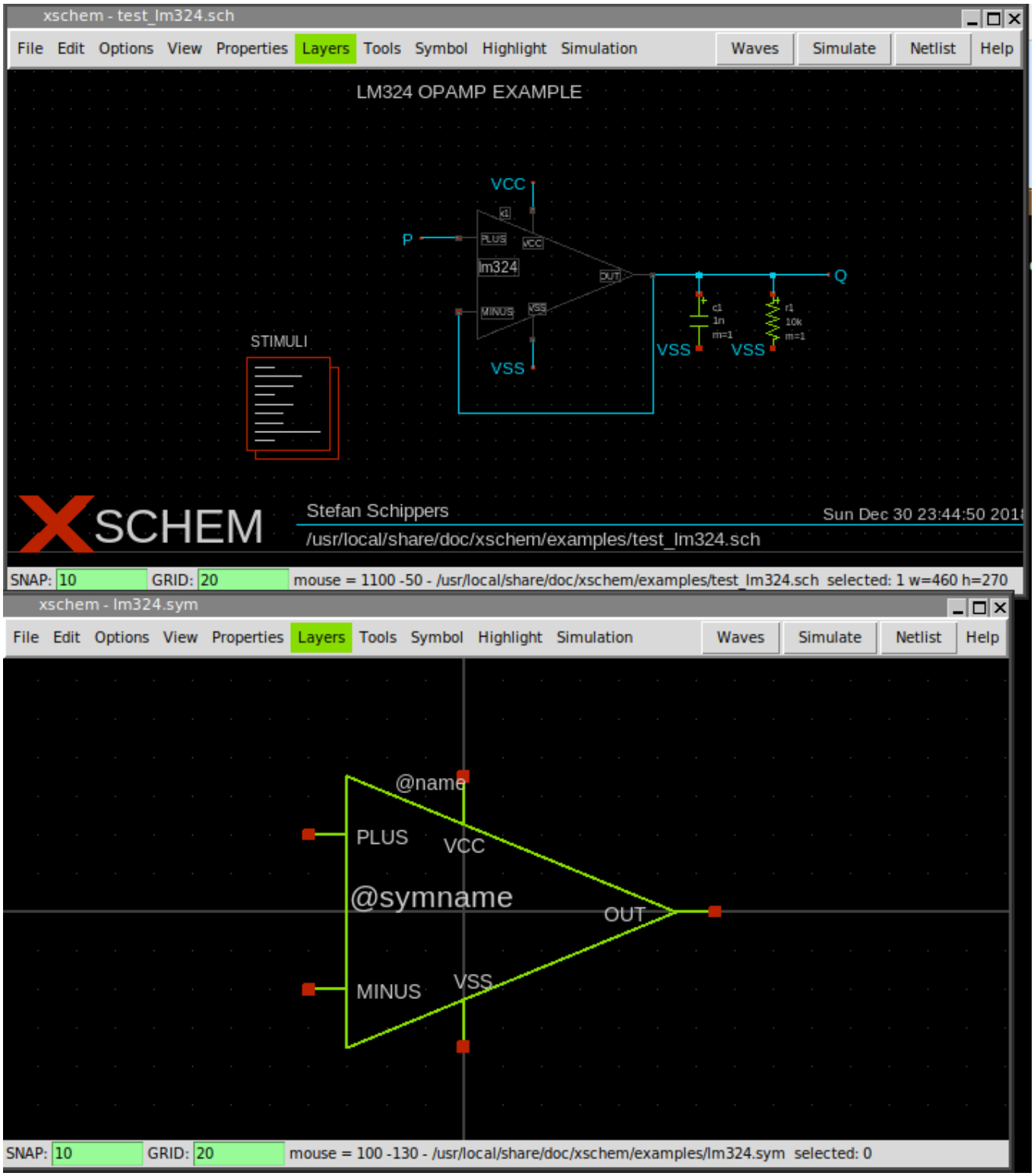

12. Select **`0_examples_top.sch`** and press 'OK':




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:




14. Click on the lm324 symbol, it can now be edited using the **`Alt-i`** key combination:

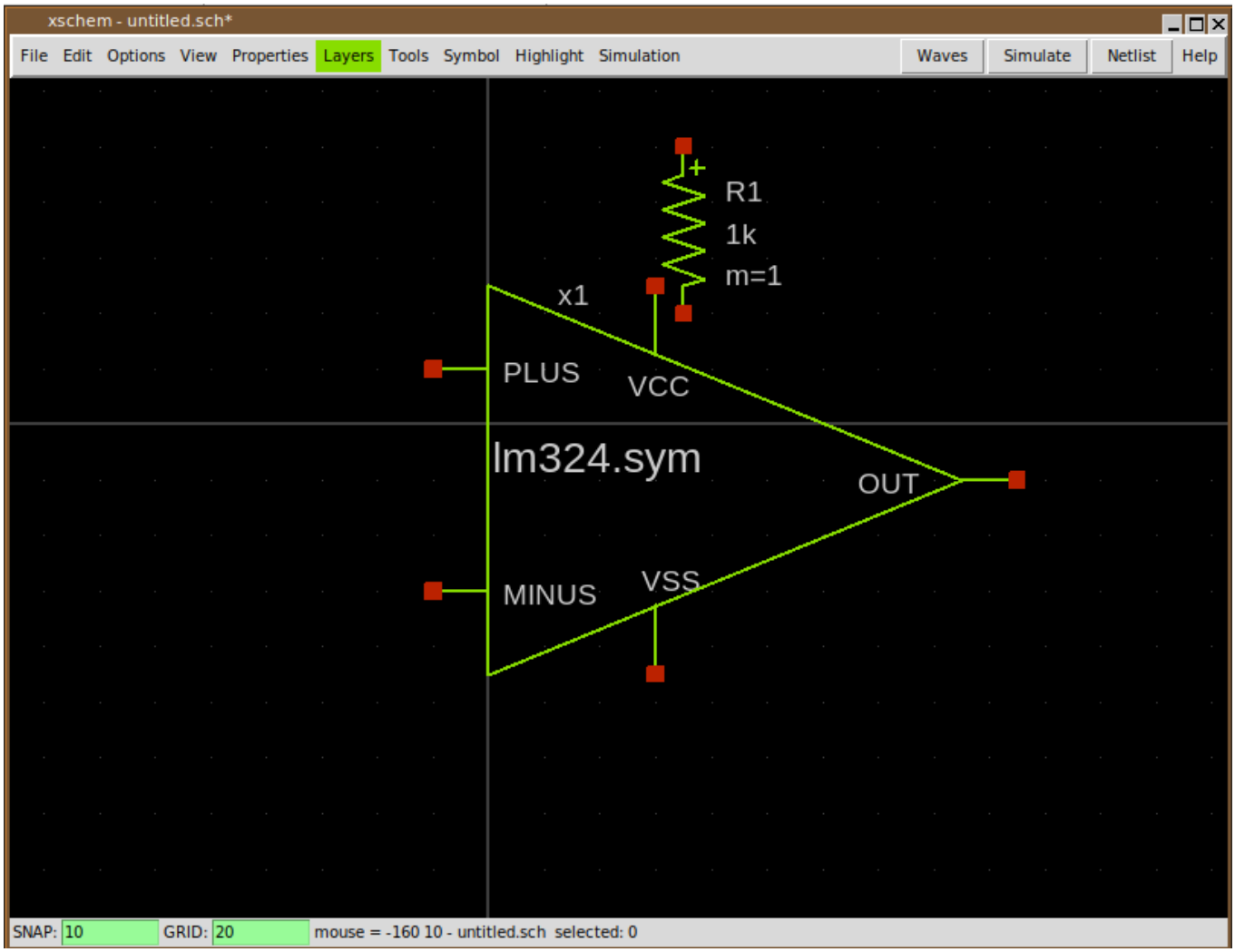



15. Now close all xschem windows and restart a new xschem instance from terminal:

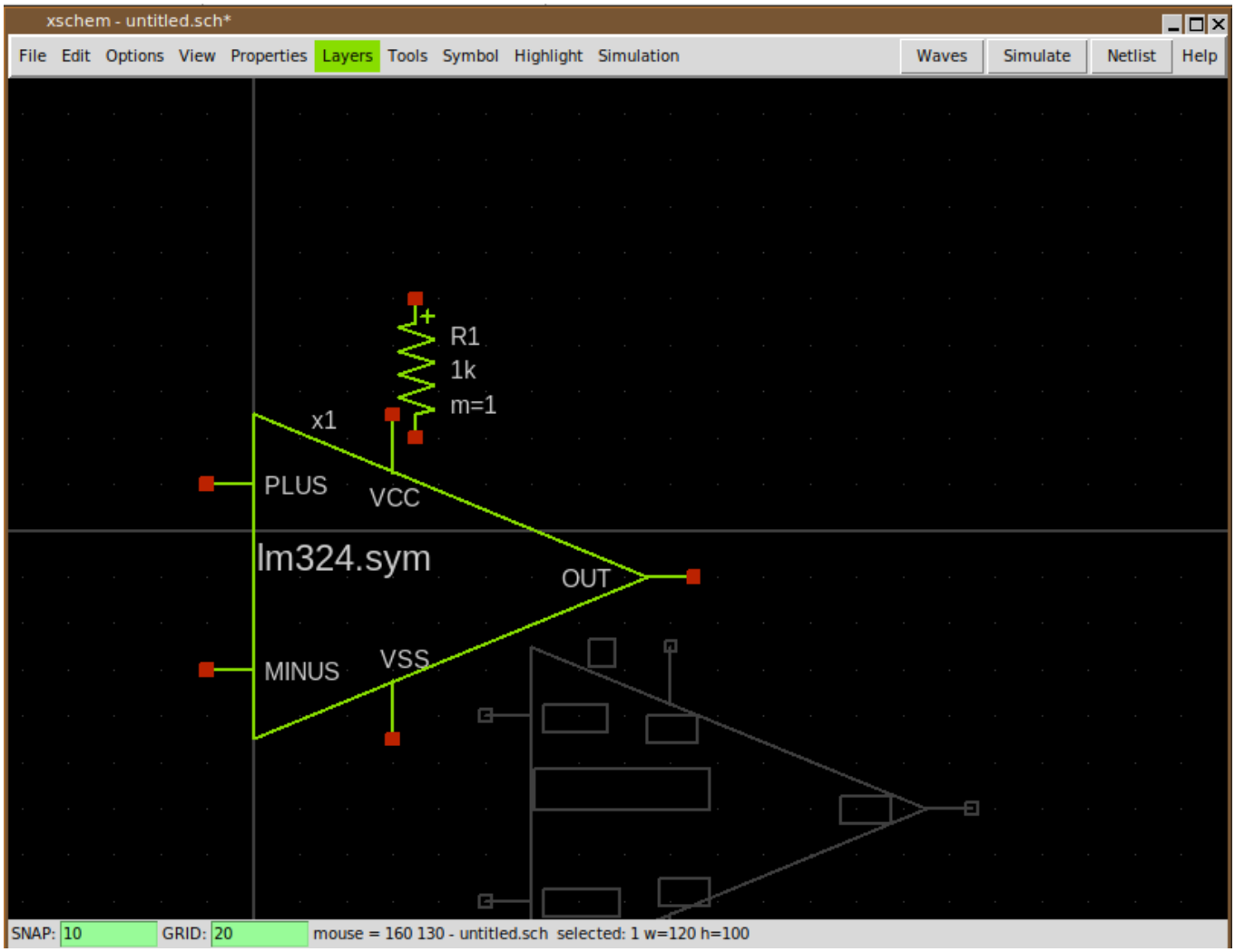
```
schippes@mazinga:~$ xschem
```

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`**:

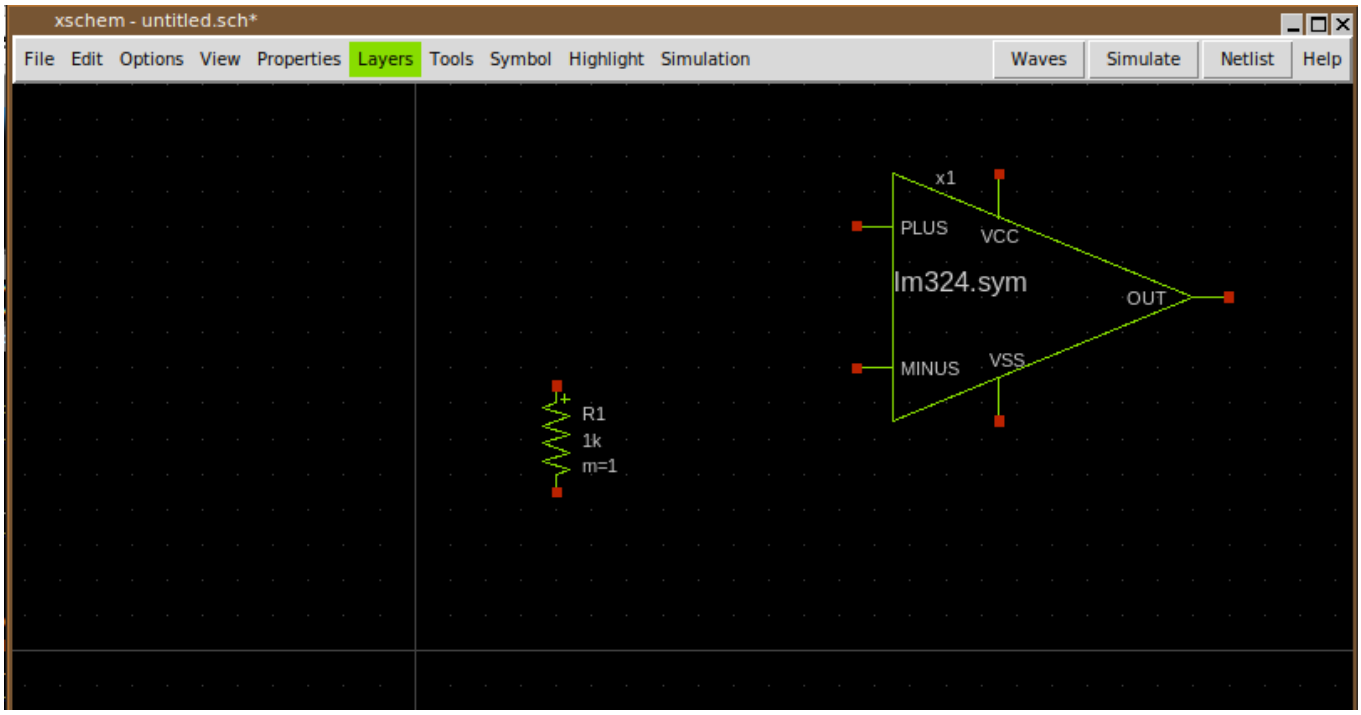

17. Lets add another component: press **`Insert`** key again and navigate to **`/usr/local/share/doc/xschem/examples`** and select **`lm324.sym`**:




18. Select (click on it) the lm324 symbol and move it by pressing the **m** key:



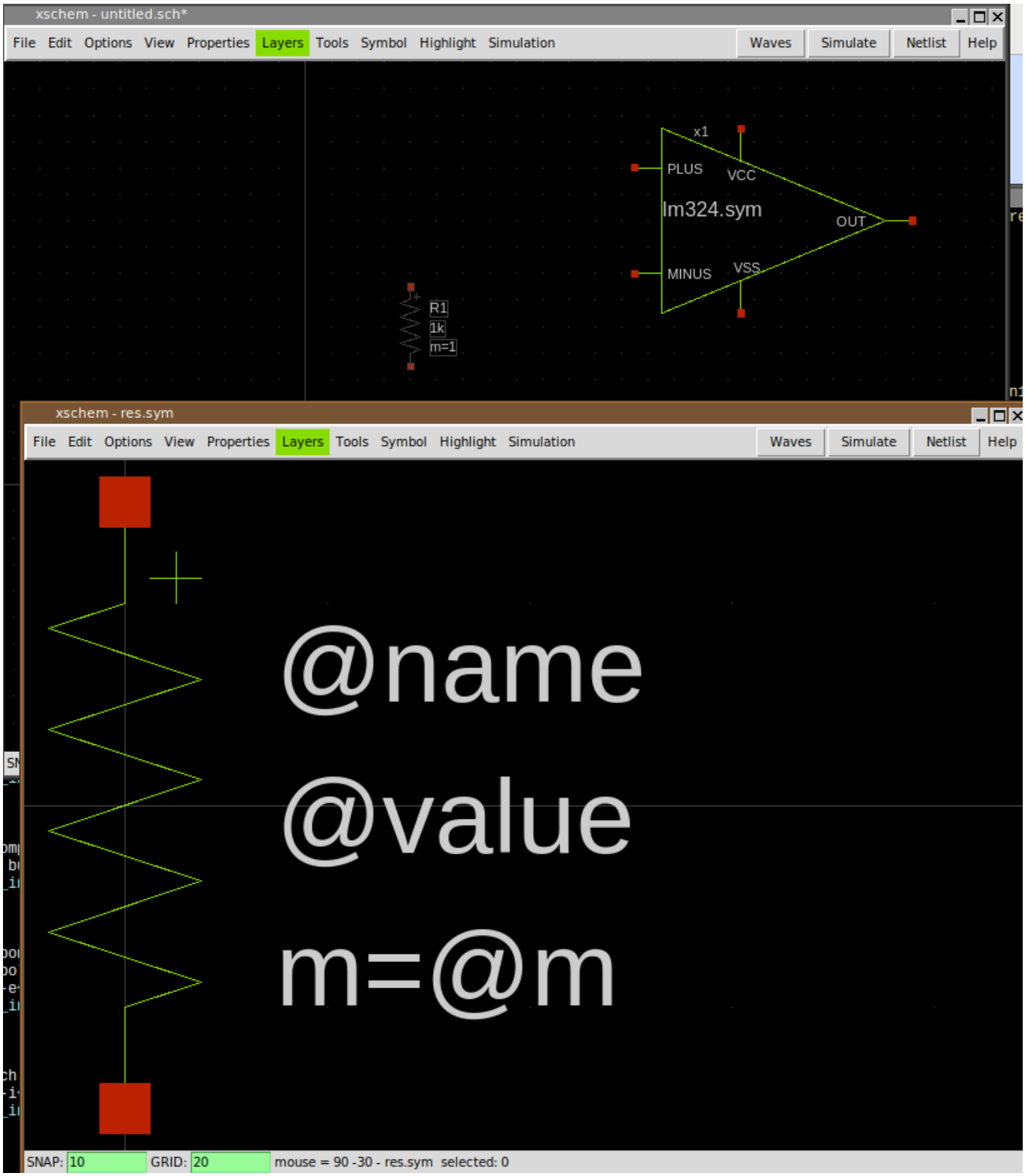

19. Place the lm324 component where you want in the schematic by placing the mouse and clicking the left button:




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`**:


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.




**This concludes the tutorial, if all the steps were successful there is a good probability that xschem is correctly installed on your system.**

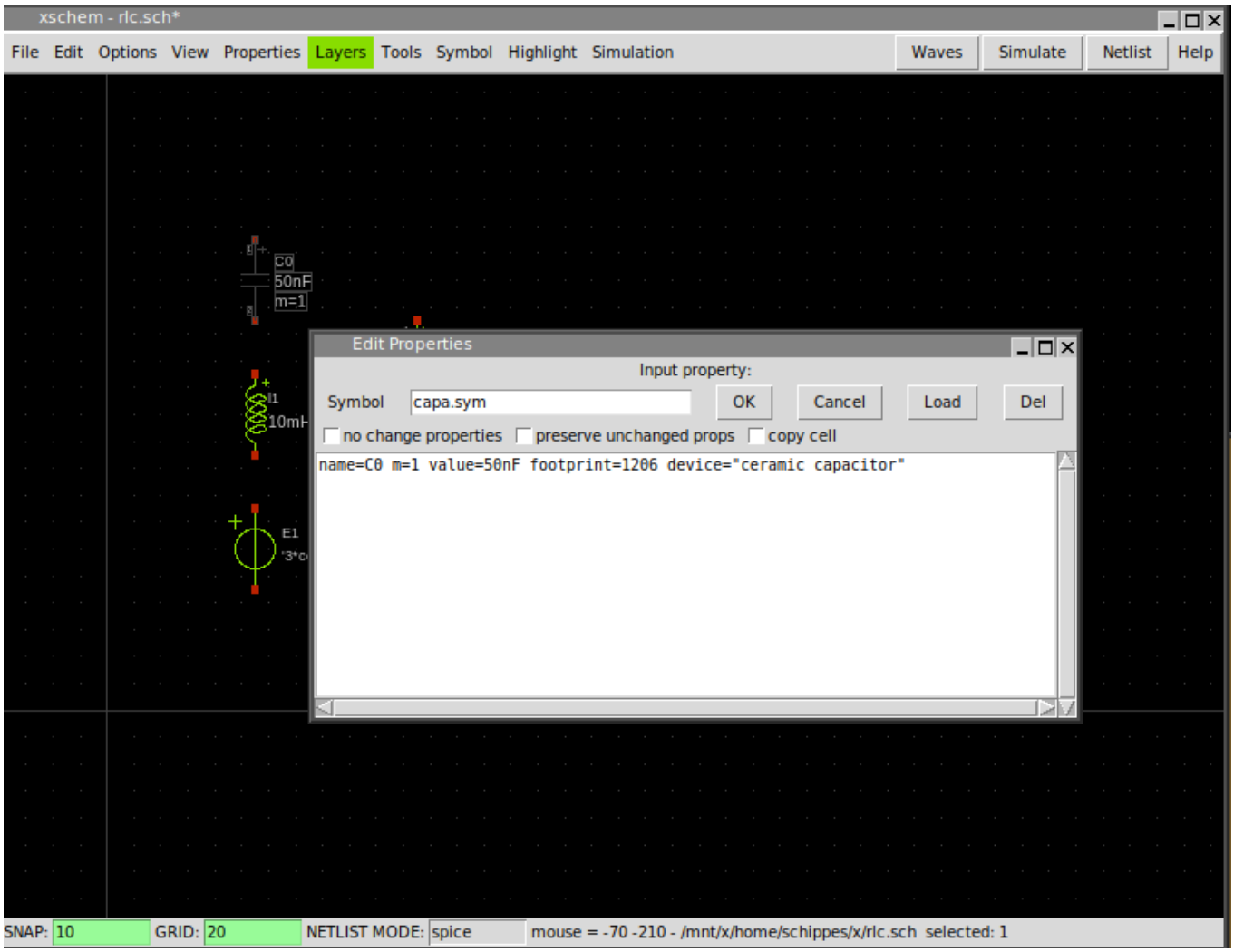
UP

# TUTORIAL: RUN A SIMULATION WITH XSCHEM

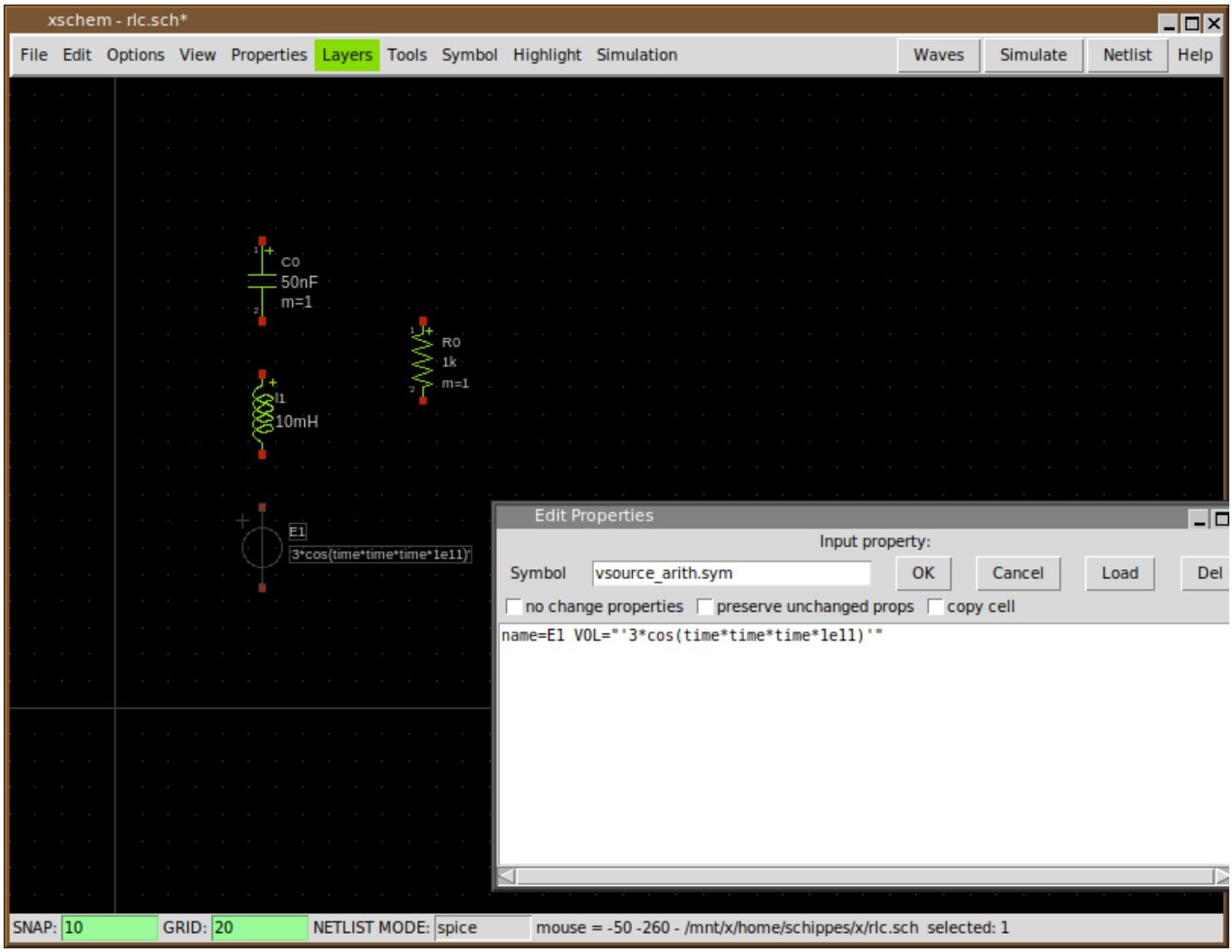
here some instructions to create a schematic and run a ngspice transient sim in XSCHEM:

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:

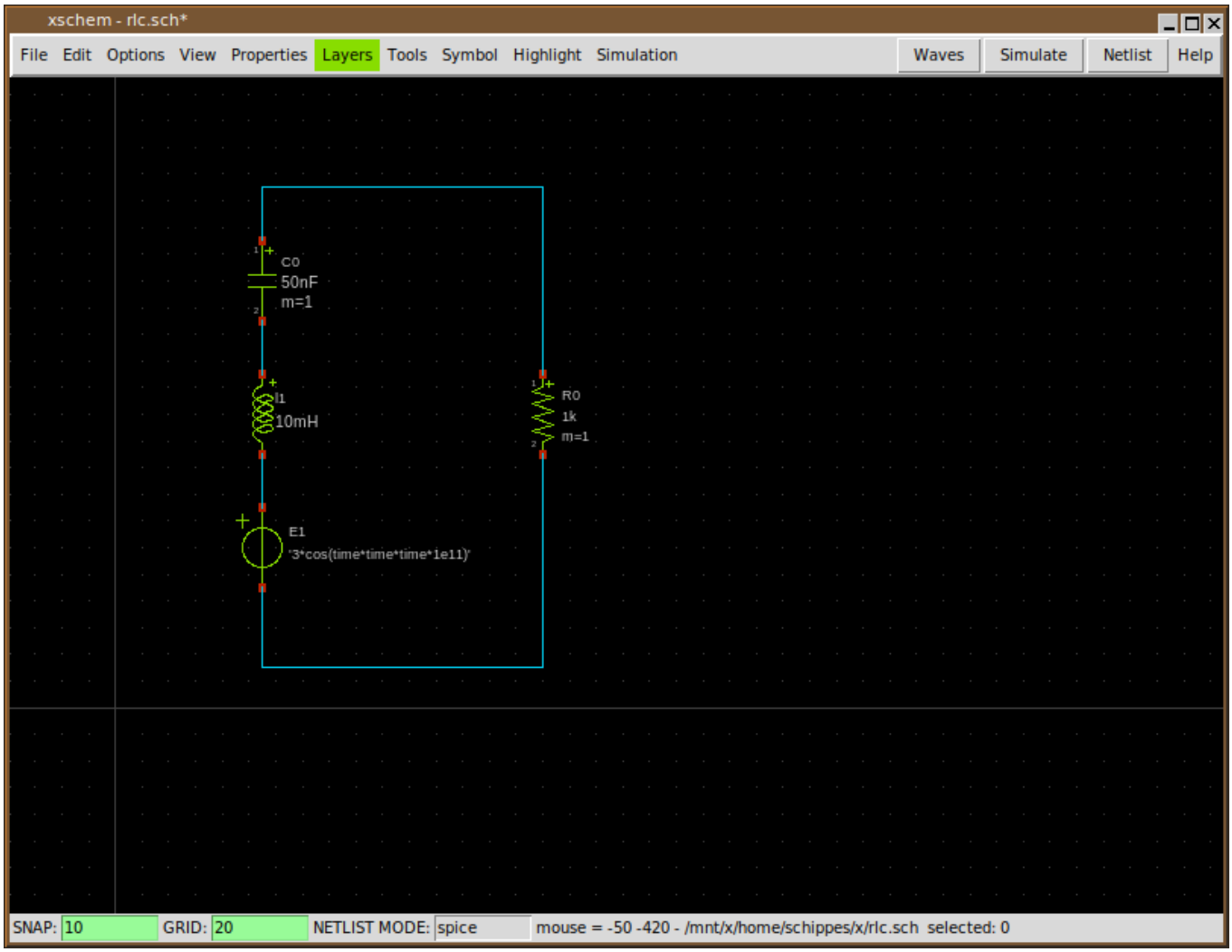

12. Press the right mouse button on the capacitor and set its 'value=' attribute to 50nF:



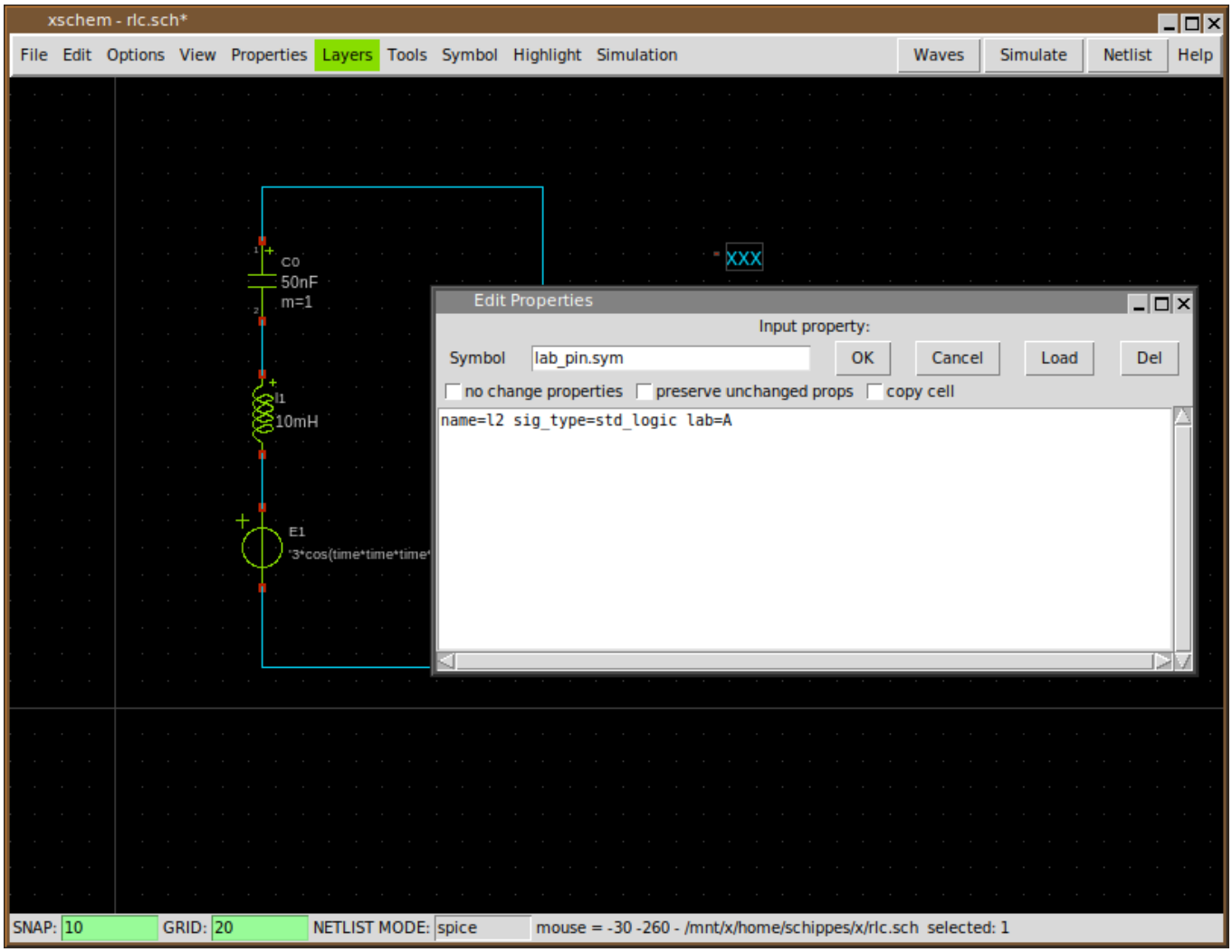

13. Do the same for the inductor (10mH) and the resistor (1k)
14. Set the voltage source VOL to: "'3*cos(time*time*time*1e11)'" (include quotes, single and double):



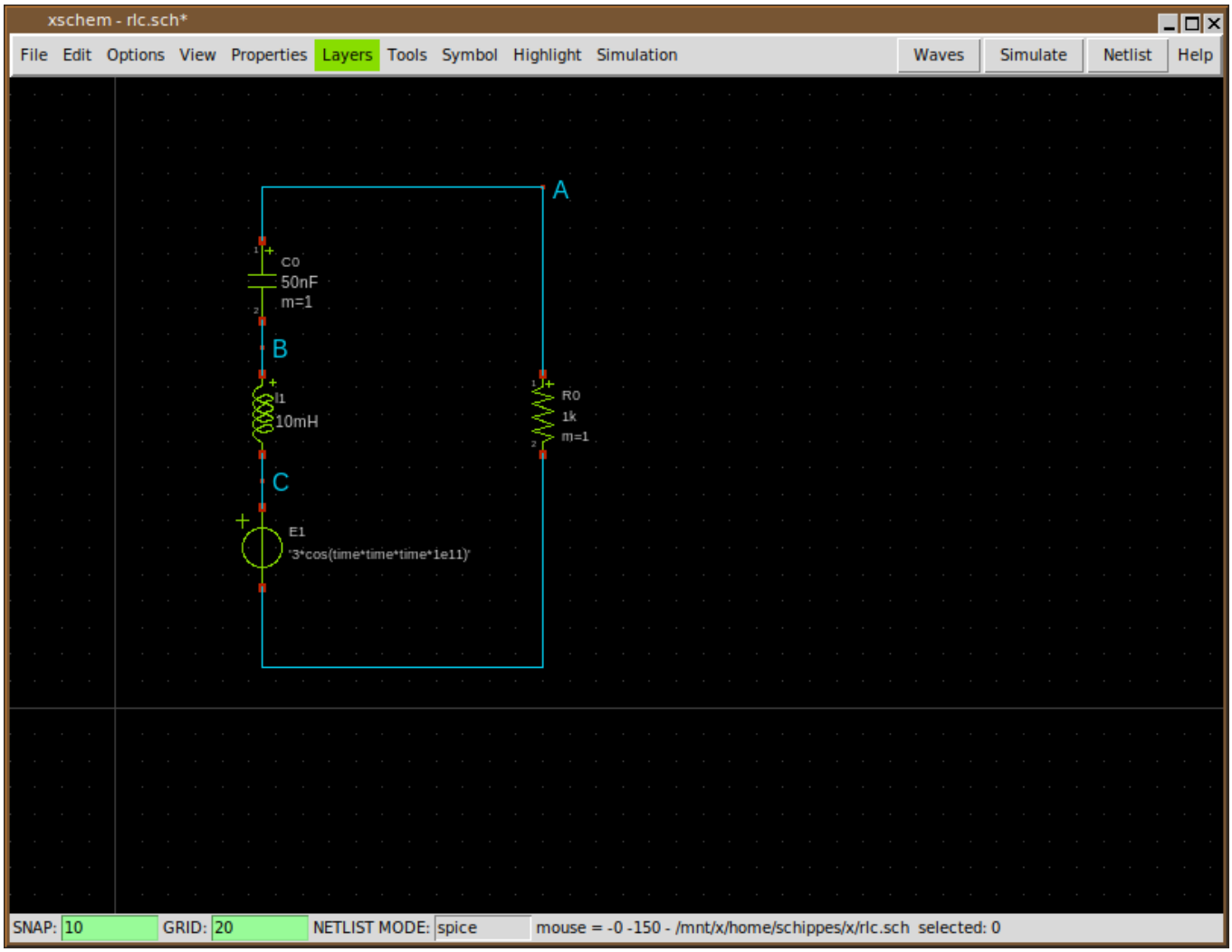

15. Pressing the 'w' key and moving the mouse you draw wires, wire the components as shown (press 'w', move the mouse and click, this draws a wire segment):



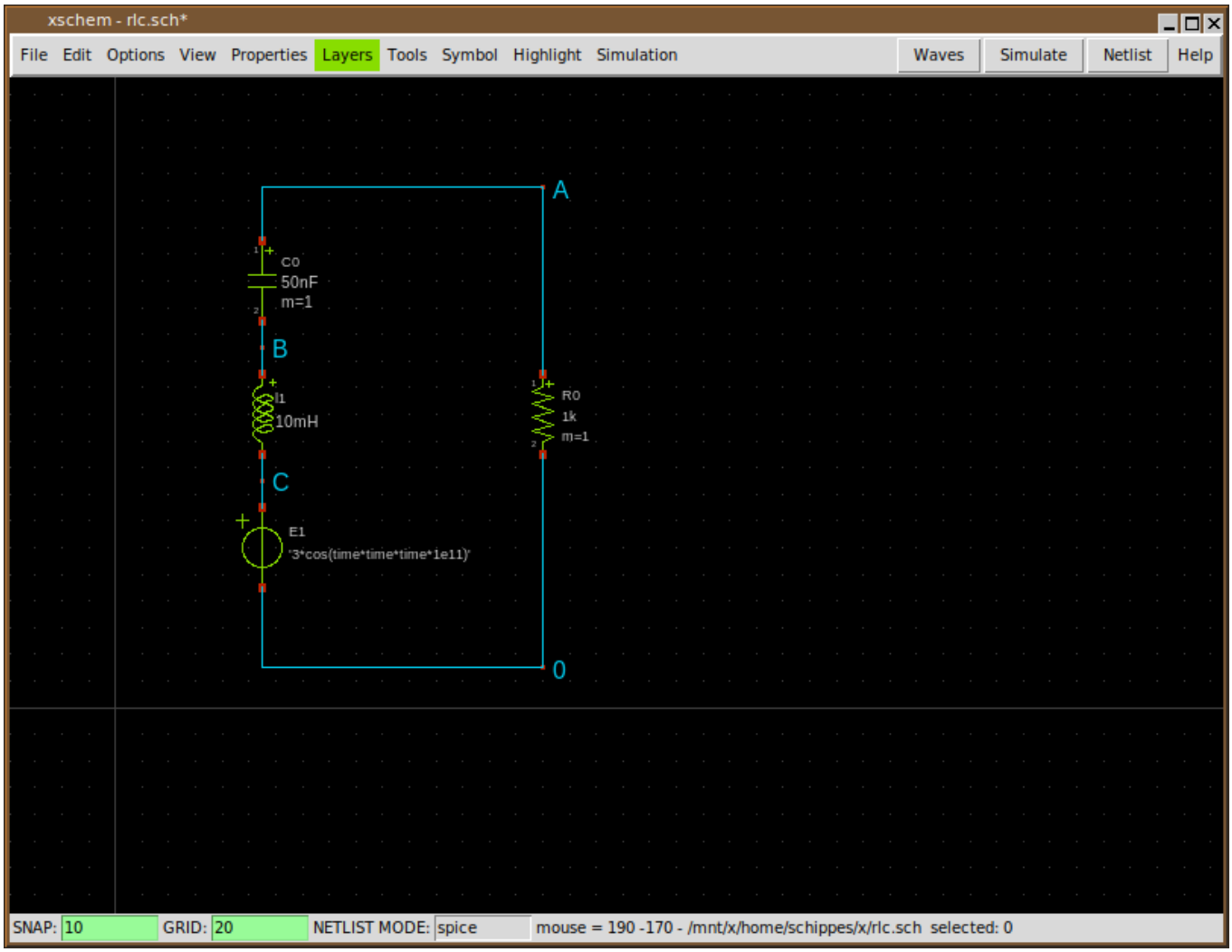

16. Press 'Insert key and place one instance of 'lab_pin', then use the right mouse button to change its 'lab' attribute to A:



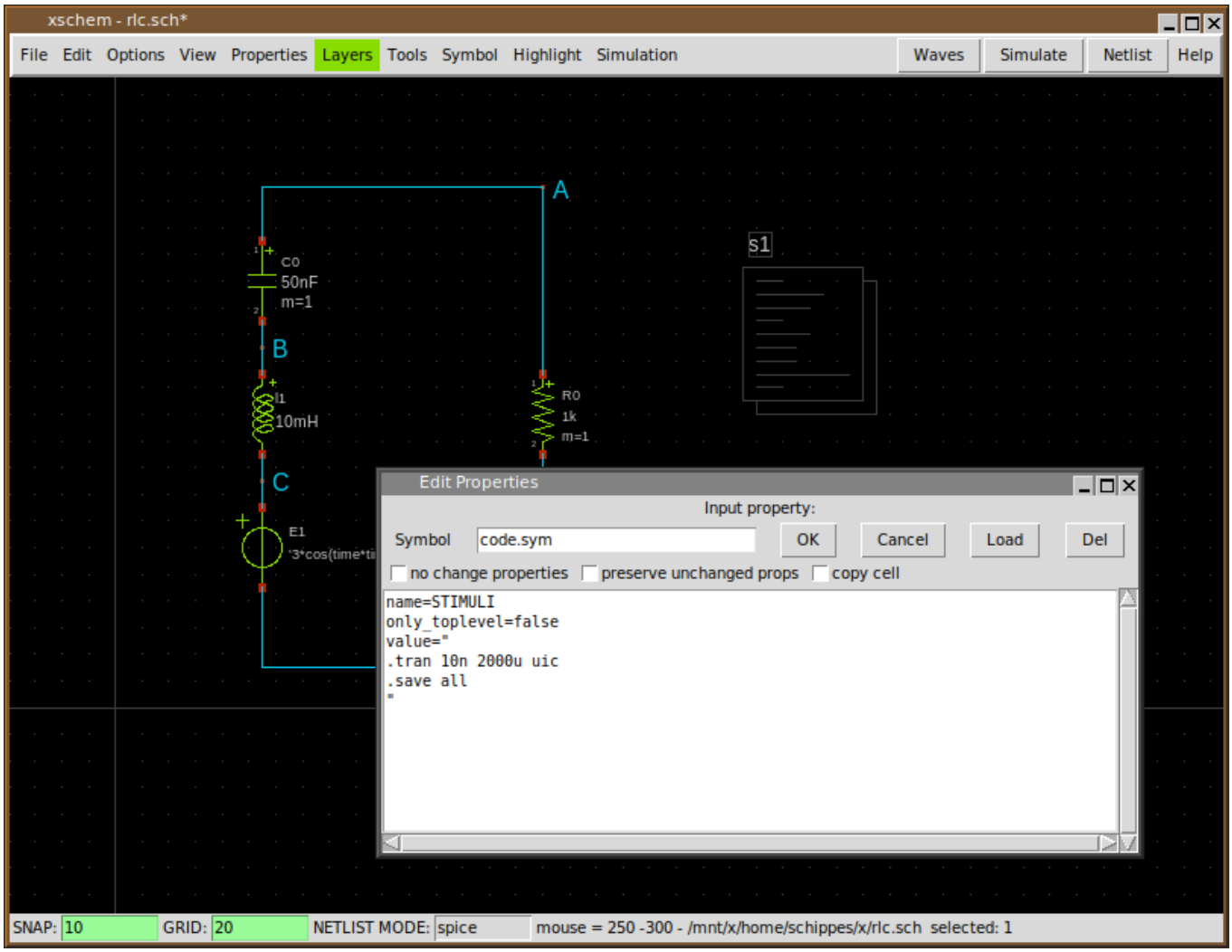

17. Move the label as shown, (you can use 'Shift+F' to flip and 'Shift+R' to rotate), then using 'c' copy this pin label and edit attributes to create the B and C labels, place all of these as shown:



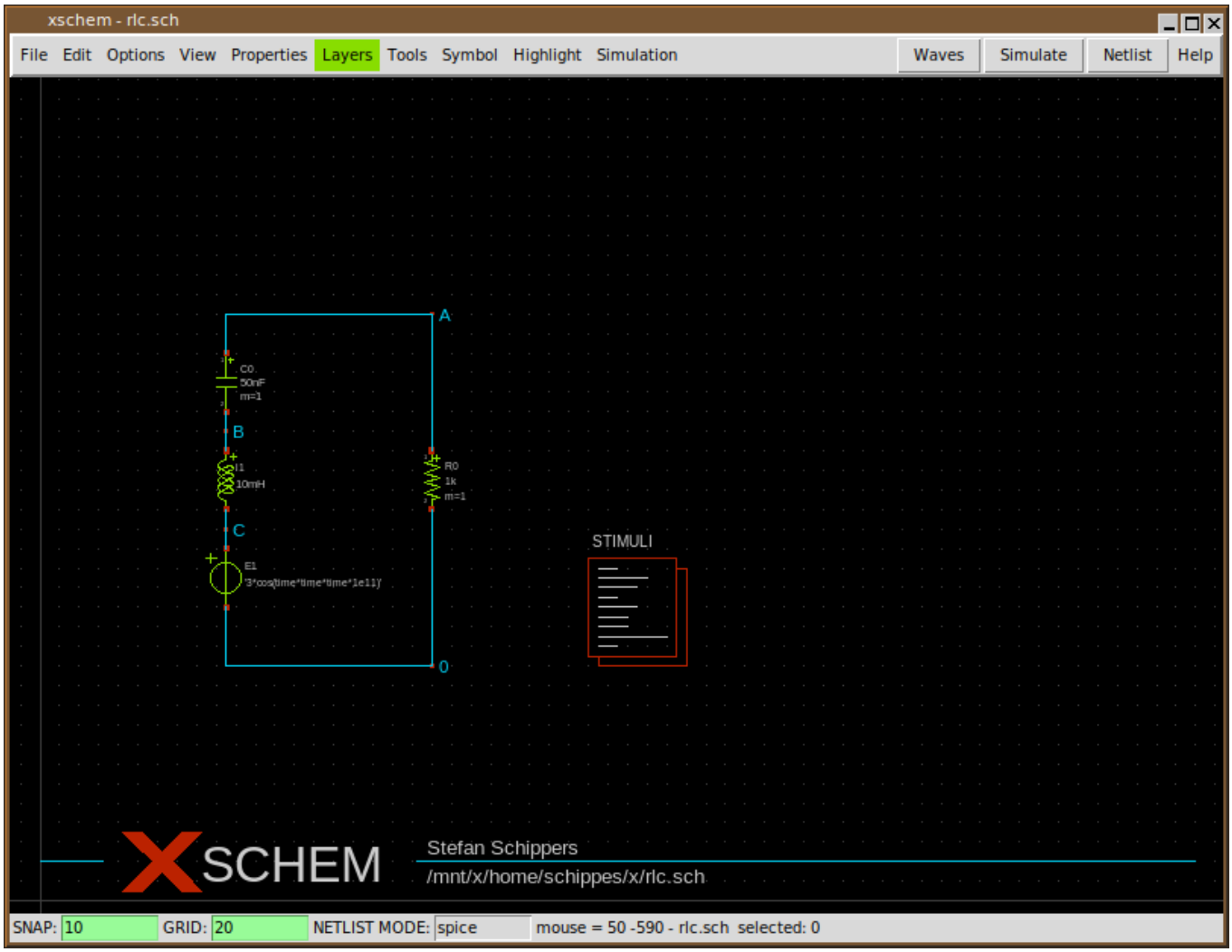

18. Select the 'C' label and copy it as shown here, set its lab attribute to 0 (this will be the 0V (gnd node))



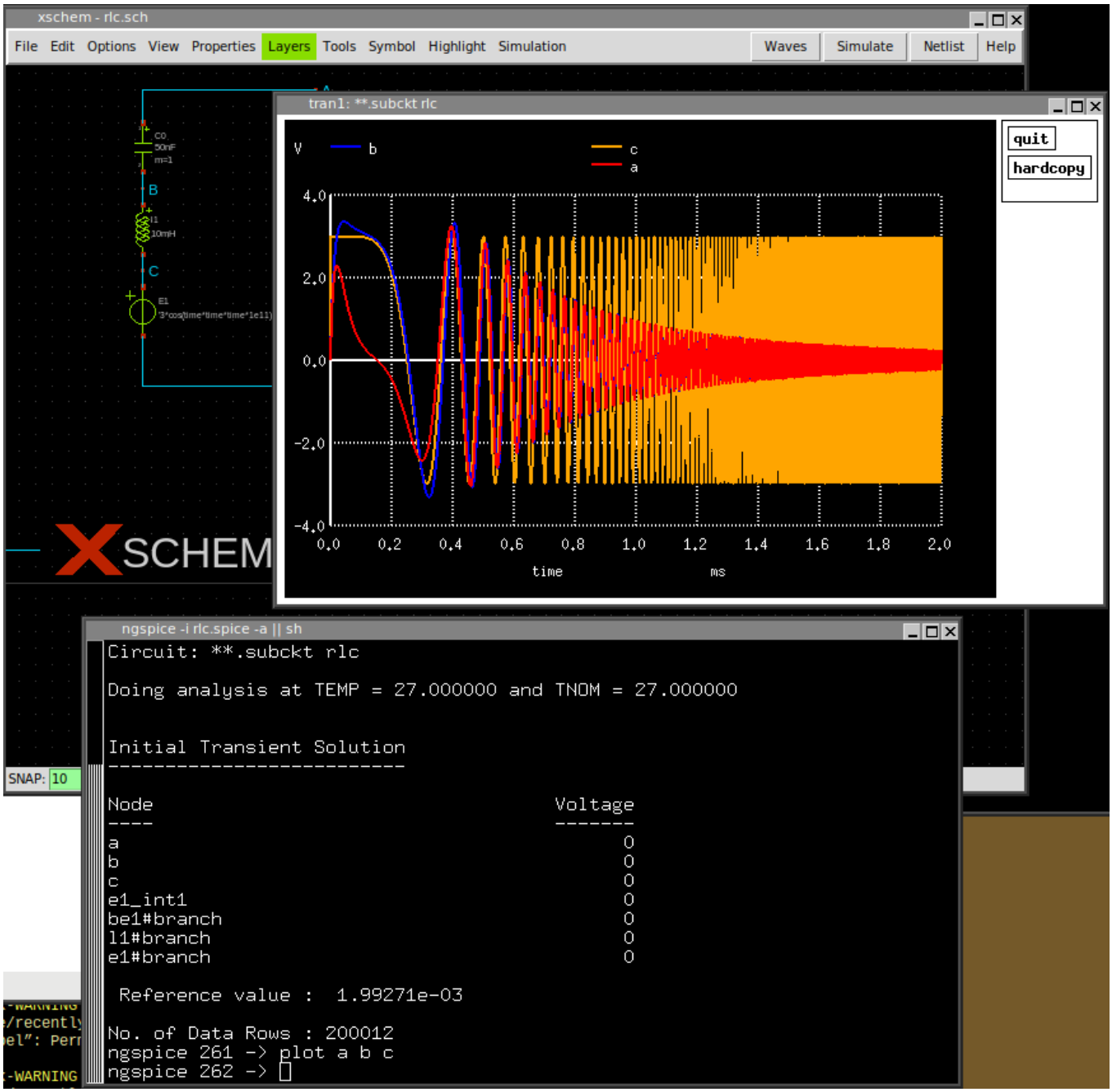

19. Press 'Insert key, place the 'code.sym' symbol, set name and value attributes as follows:




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




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':

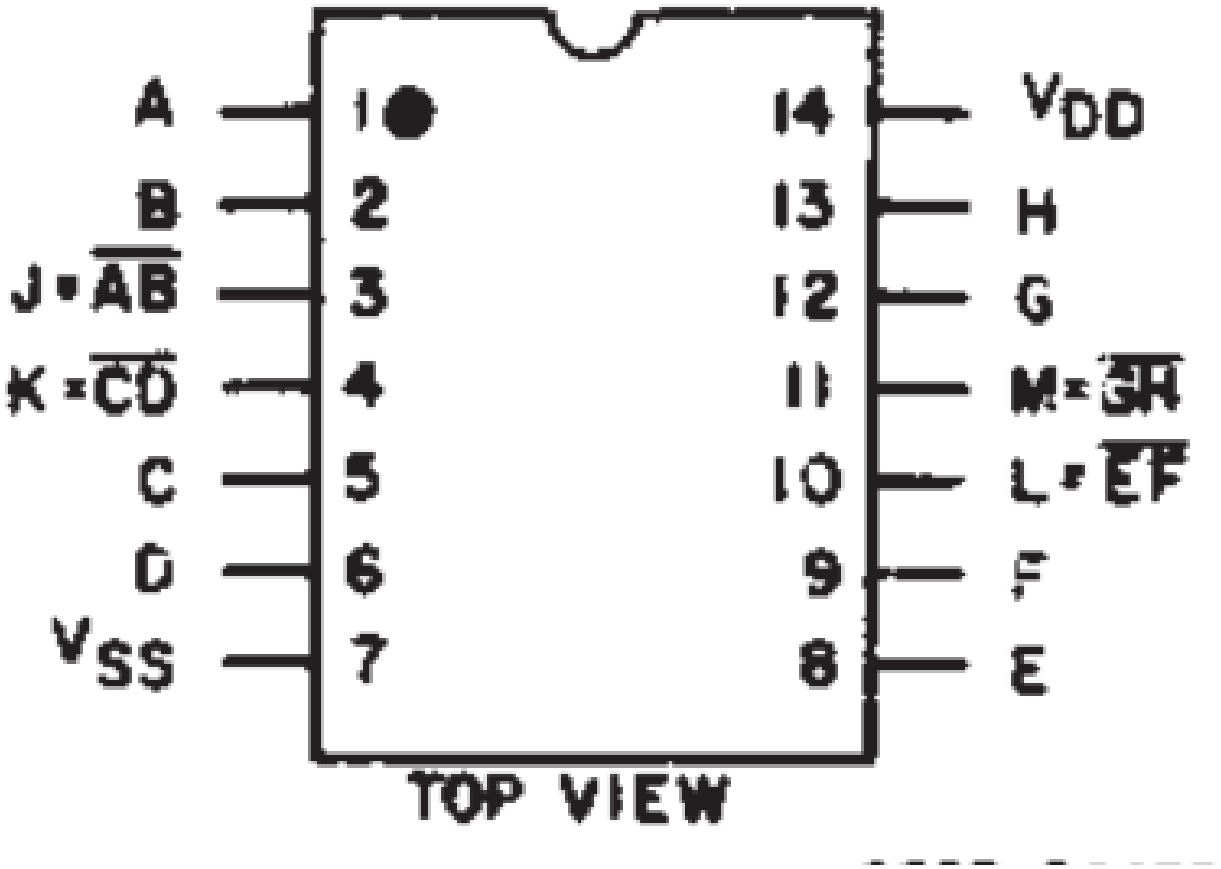


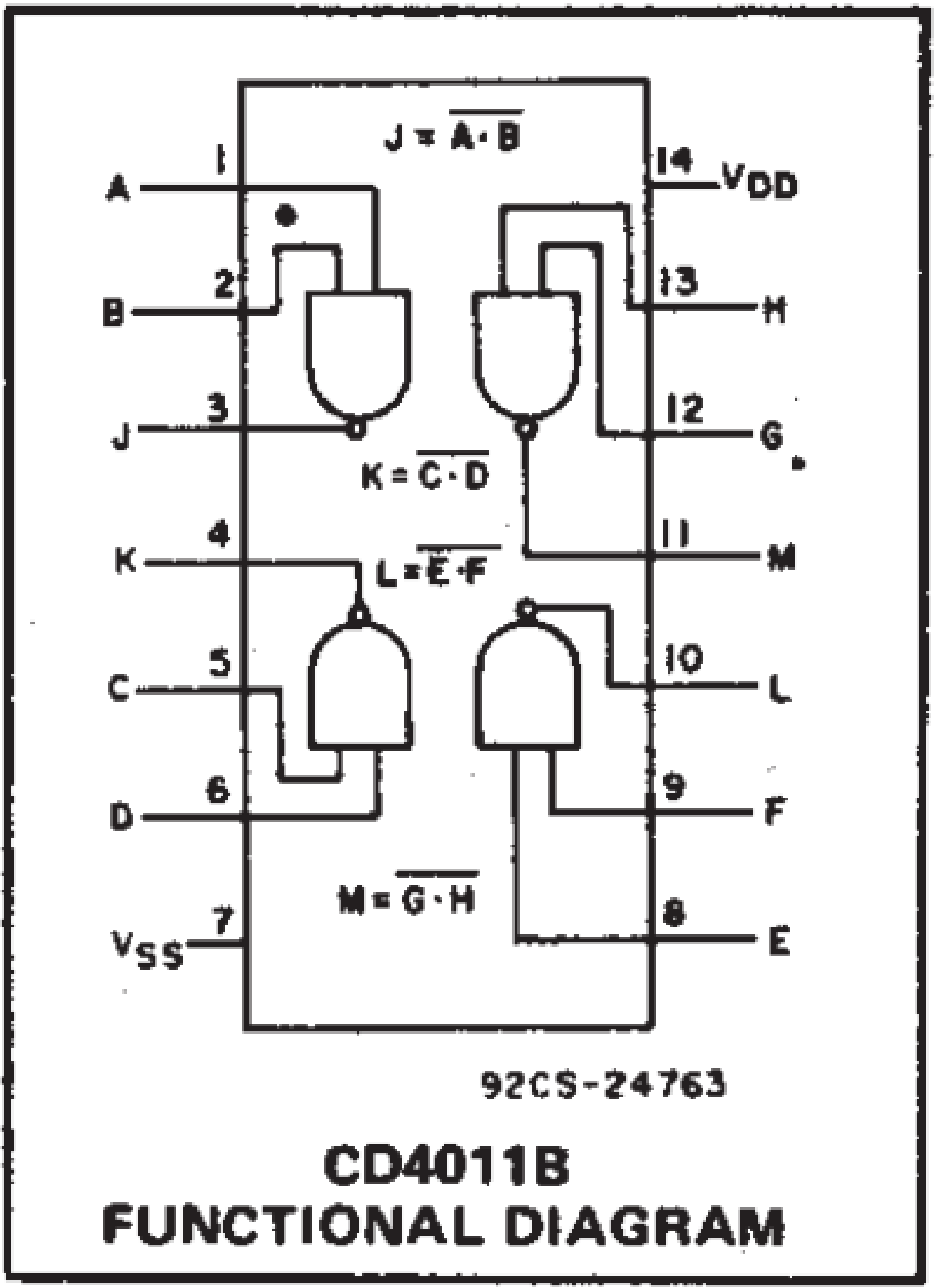

24. If you set 'Simulation -> Configure simulators and tools -> Ngspice Batch' and press 'Simulate' again the sim will be run in batch mode, a 'rlc.raw' file will be generated and a 'rlc.out' file will contain the simulator textual output.

UP

# TUTORIAL: CREATE AN XSCHEM SYMBOL

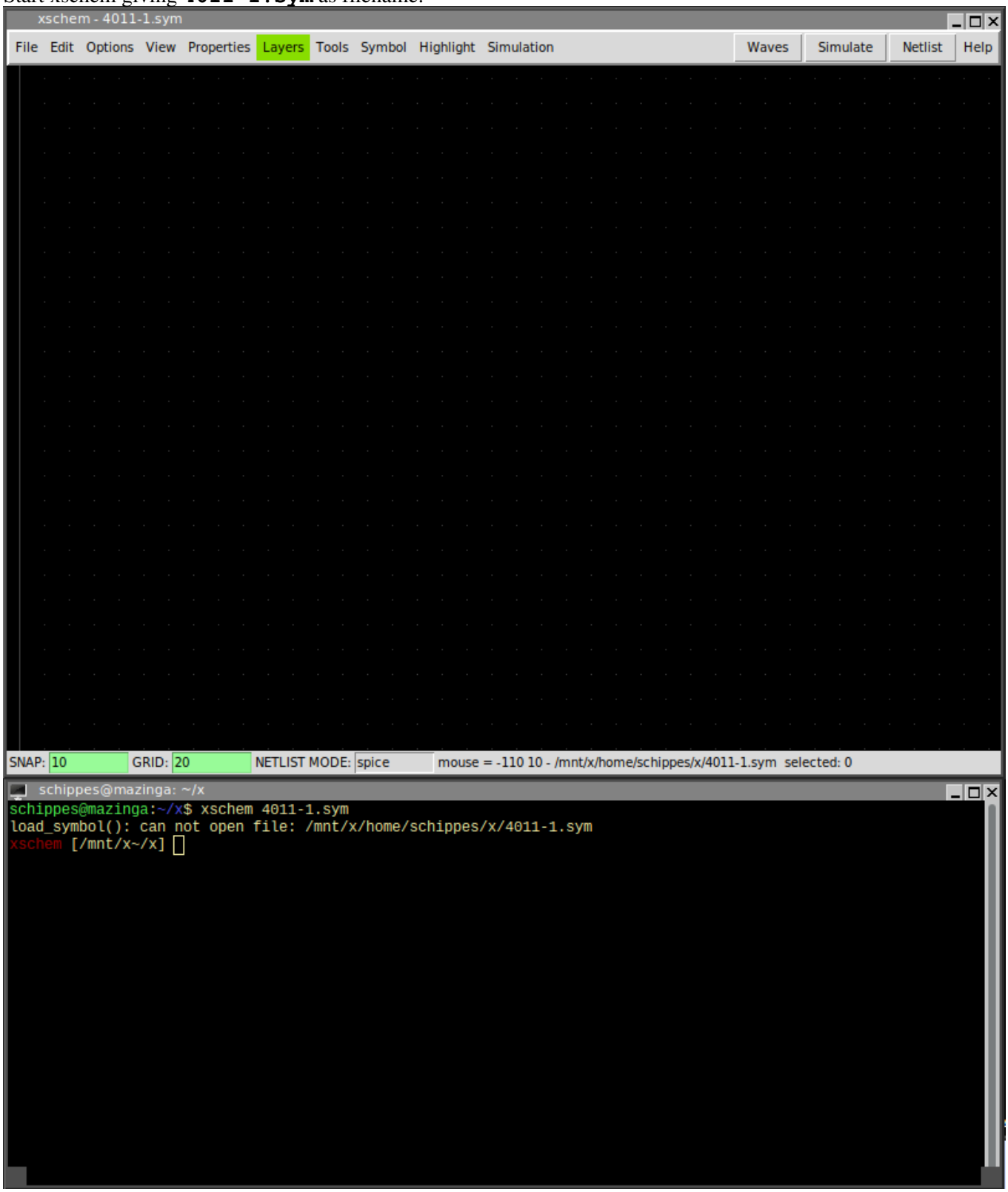
In this tutorial we will build a 4011 CMOS quad 2-input NAND symbol. This IC has 4 nand gates (3 pins each, total 4*3=12 pins + VDD,VSS power pins) This device comes in a dual in line 14 pin package.



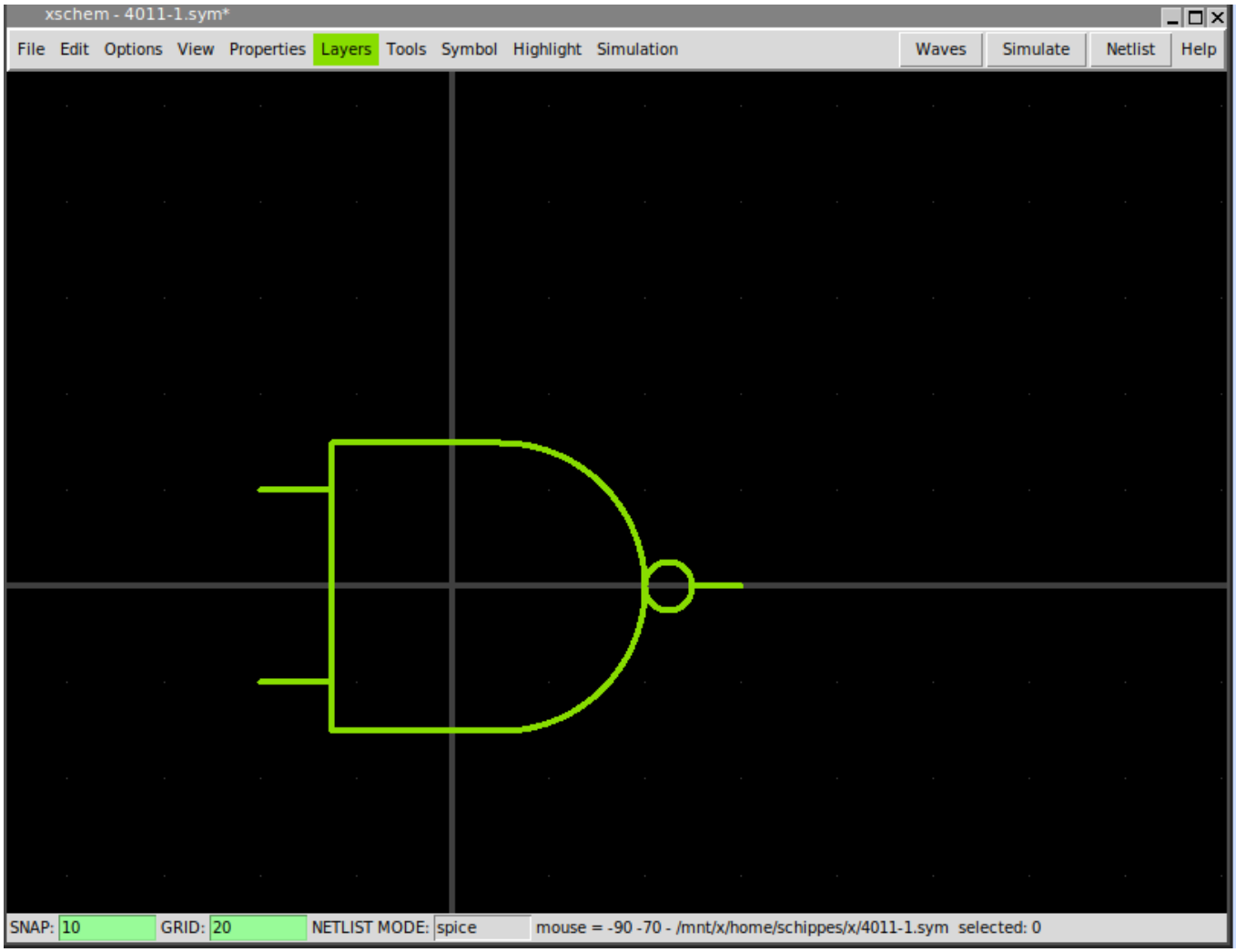



**CD4011B**
**FUNCTIONAL DIAGRAM**

1. Start xschem giving **`4011-1.sym`** as filename:



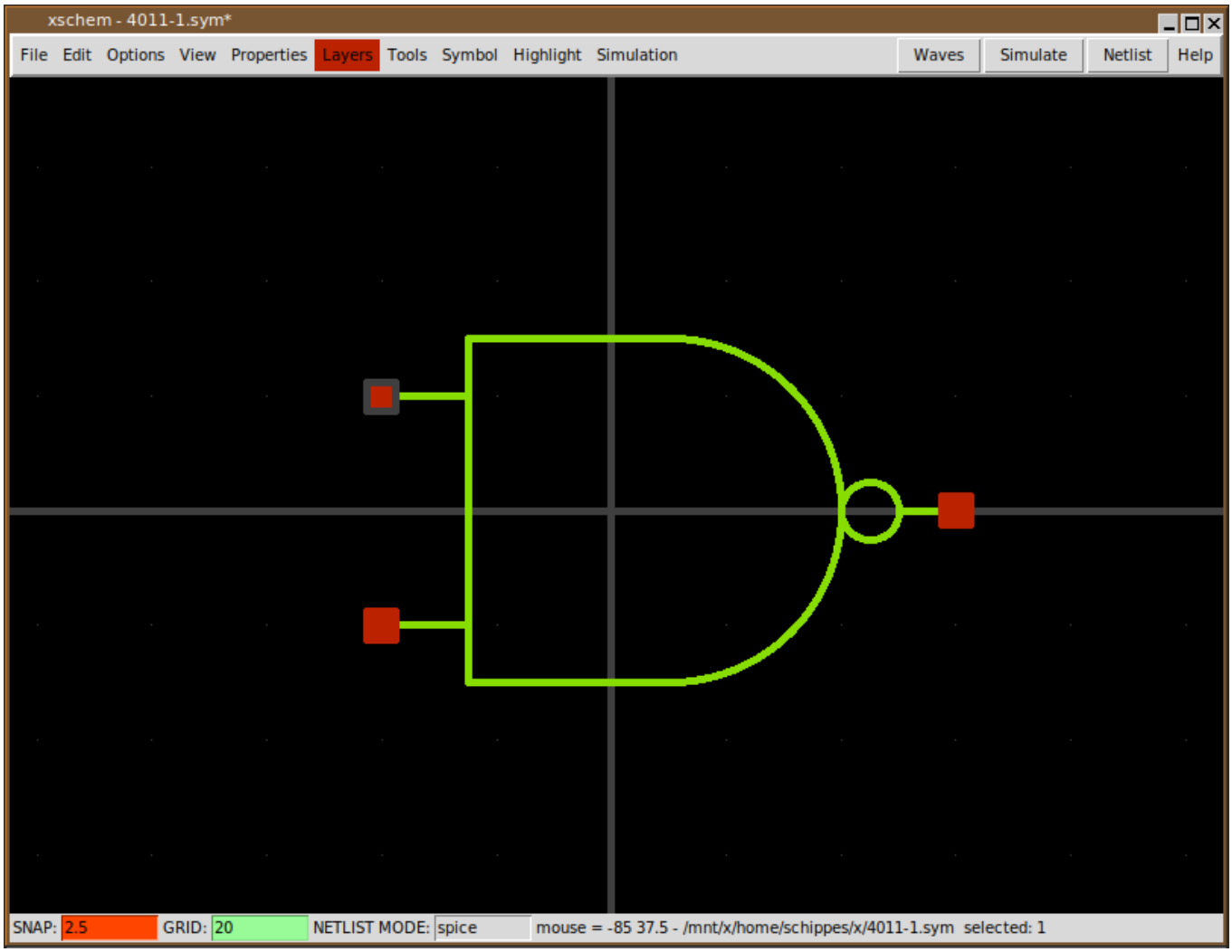

2. use layer 4 (the default) to draw the following shapes, use **`l`** to draw lines and use **`Shift-c`** to draw arcs, use **`Ctrl-Shift-c`** to draw circles. Arcs and circles are drawn by specifying start - end point and a 3rd way point. You will need to change the grid snap to '5' for drawing the 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.




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`**

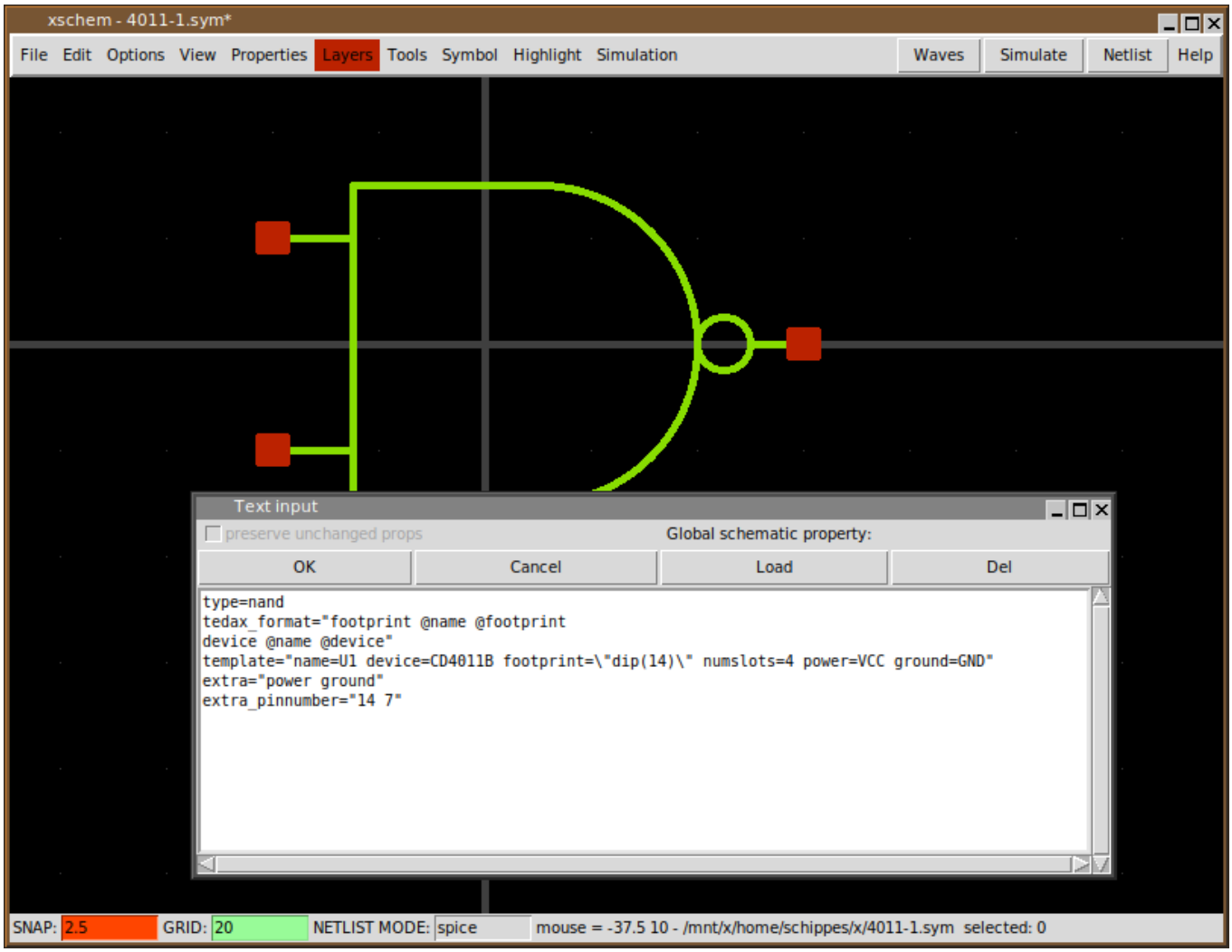



4. Now when **no object is selected** press **q** to edit the symbol global attributes. Type the following text:

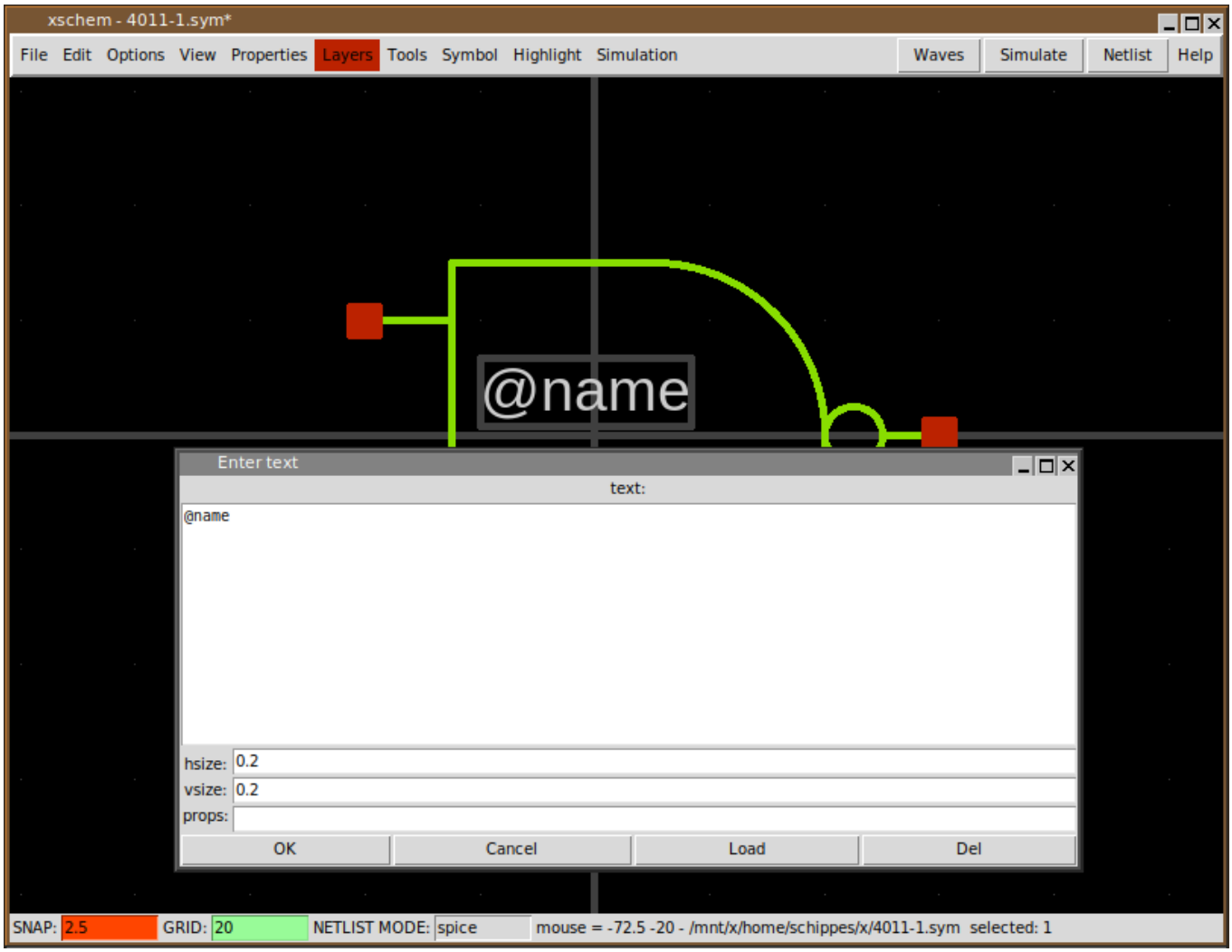
```
type=nand
tedax_format="footprint @name @footprint
device @name @device"
template="name=U1 device=CD4011B footprint=\"dip(14)\" numslots=4 power=VCC ground=GND"
extra="power ground"
extra_pinnumber="14 7"
```

Instead of the **q** key the attribute dialog box can also be displayed by pressing the **right** mouse button

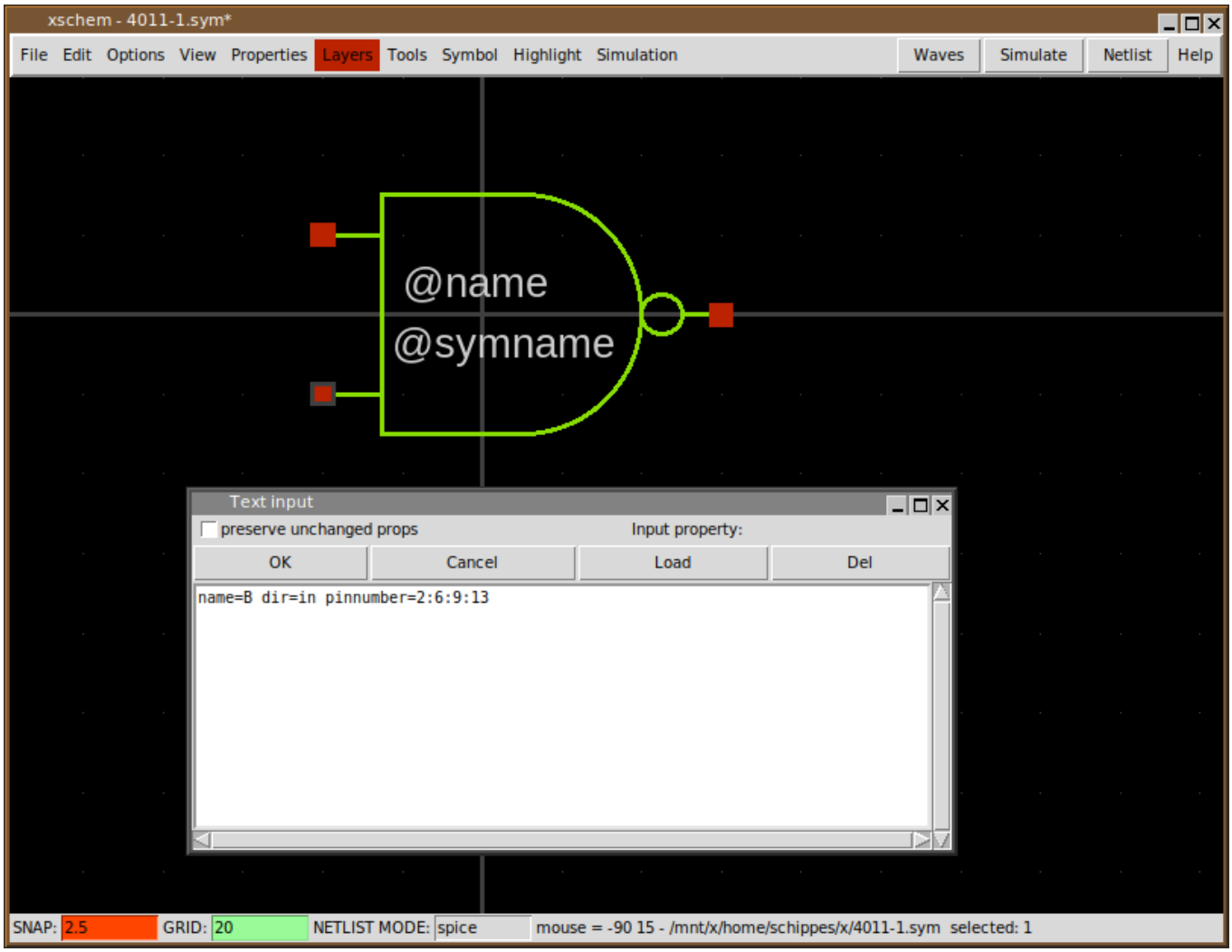
these attributes specify the gate type, the format for tedax netlist, the **template** attribute specifies default values for attributes and defines pin connection for VDD and VSS that are associated to package pins 14 and 7. The **device** attribute specifies the component name to be used in the tEDAx 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.




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.

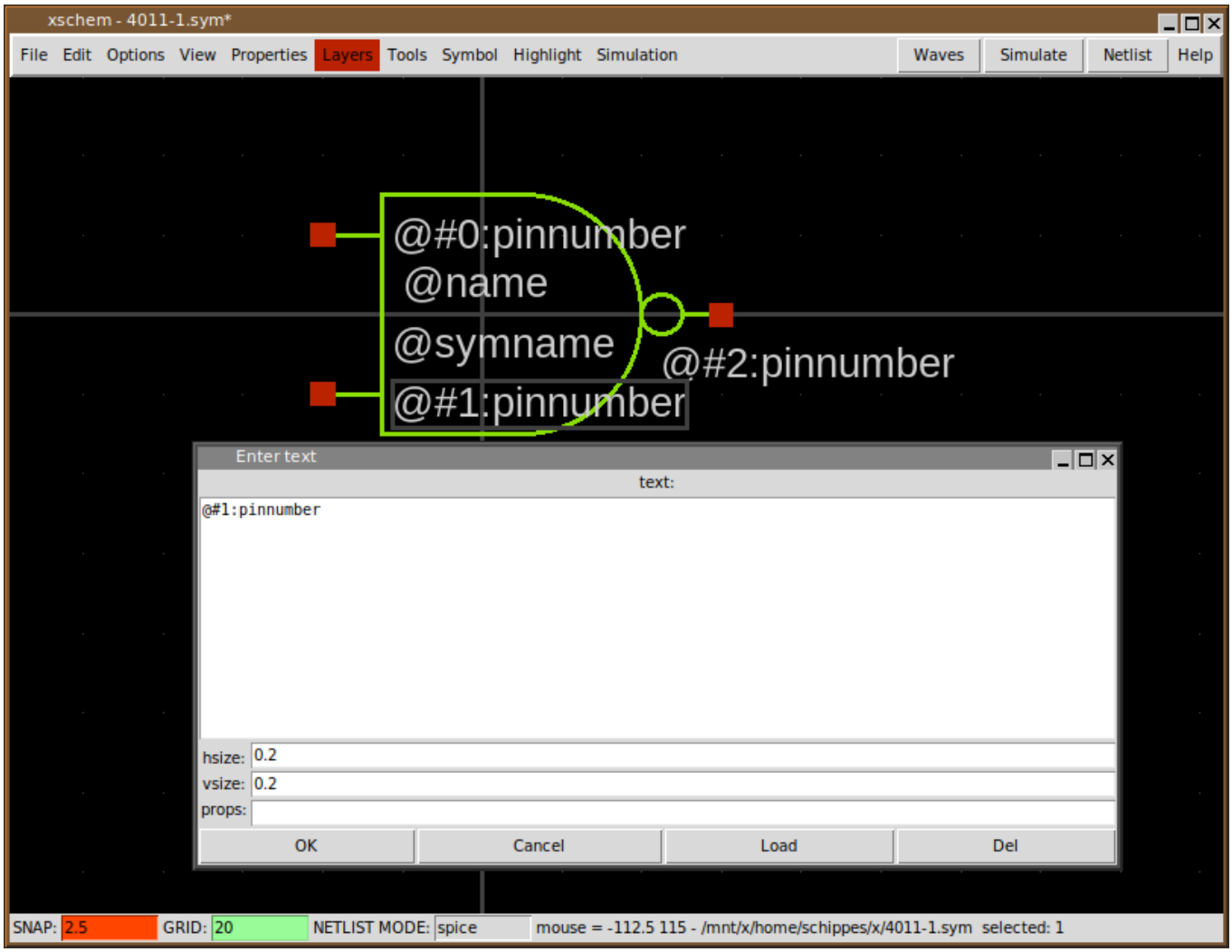



6. select the red pins (click the mouse close to the interior side of the rectangle corners) and press **q**, set attribute **name=A dir=in pinnumber=1:5:8:12** for the upper left pin, **name=B dir=in pinnumber=2:6:9:13** for the lower left pin, **name=Z dir=out pinnumber=3:4:10:11** for the right output 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.




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

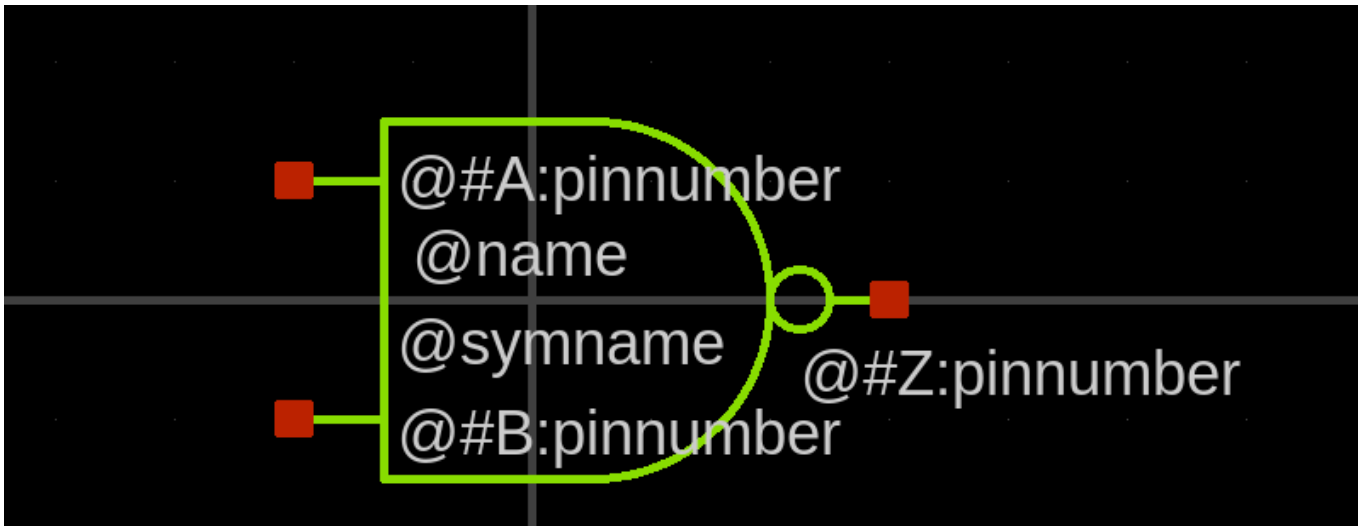
7. We want now to place some text near the gate pins to display the pin number: again, use the **t** key and place the following text, with hsize and vsize set to 0.2:



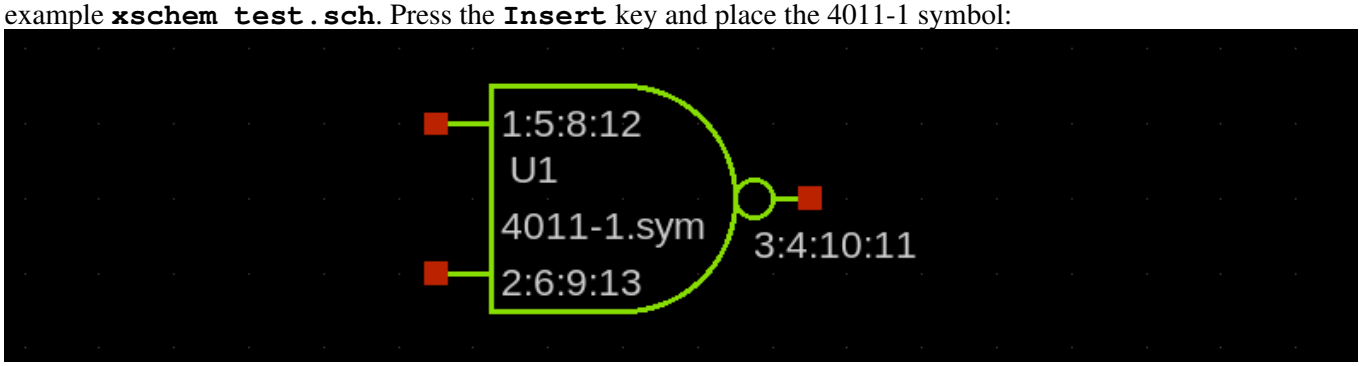

The complicated syntax of these text labels has the following meaning:

- 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.

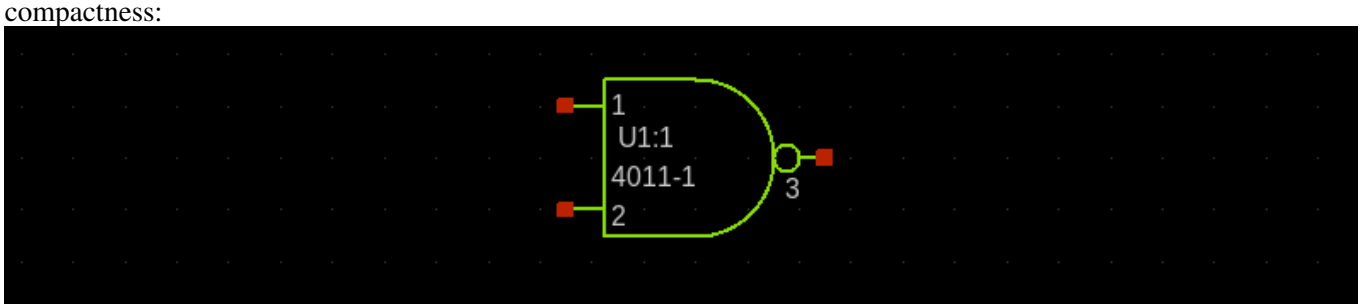
8. There is another syntax that can be used to display pin numbers, instead of specifying the pin index in XSCHEM list (that reflects the creation order) you can reference pins by their name; The only reason to use the previous syntax with pin index numbers is efficiency when dealing with extremely big symbols (SoC or similar high pin count chips).



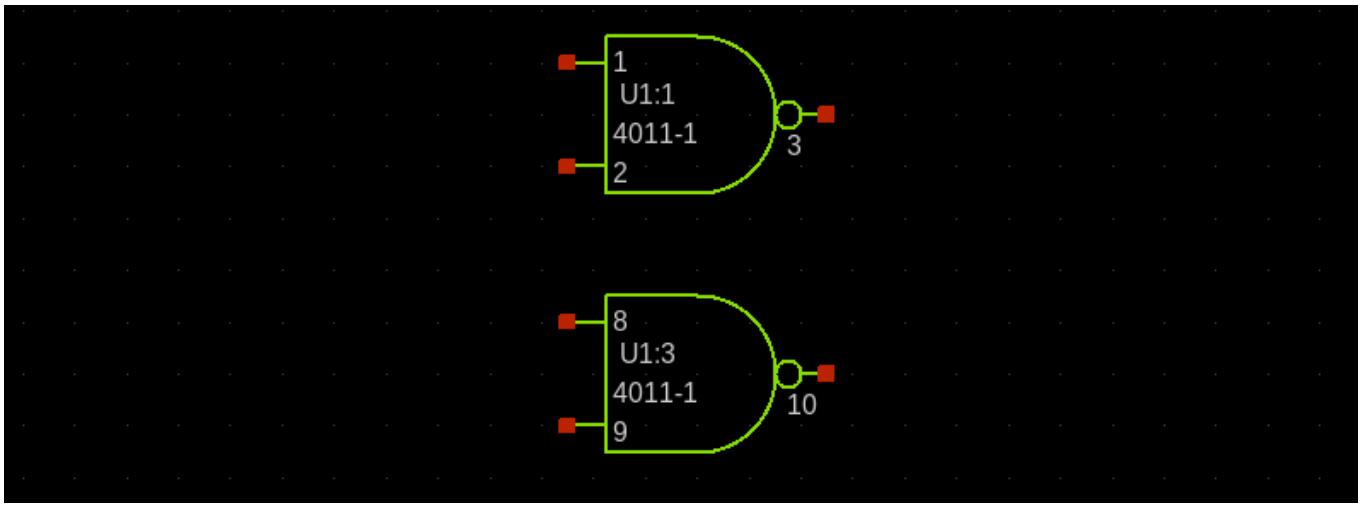

9. The symbol is now complete; save it and close XSCHEM. Now open again xschem with an empty schematic, for example **`xschem test.sch`**. Press the **`Insert`** key and place the 4011-1 symbol:



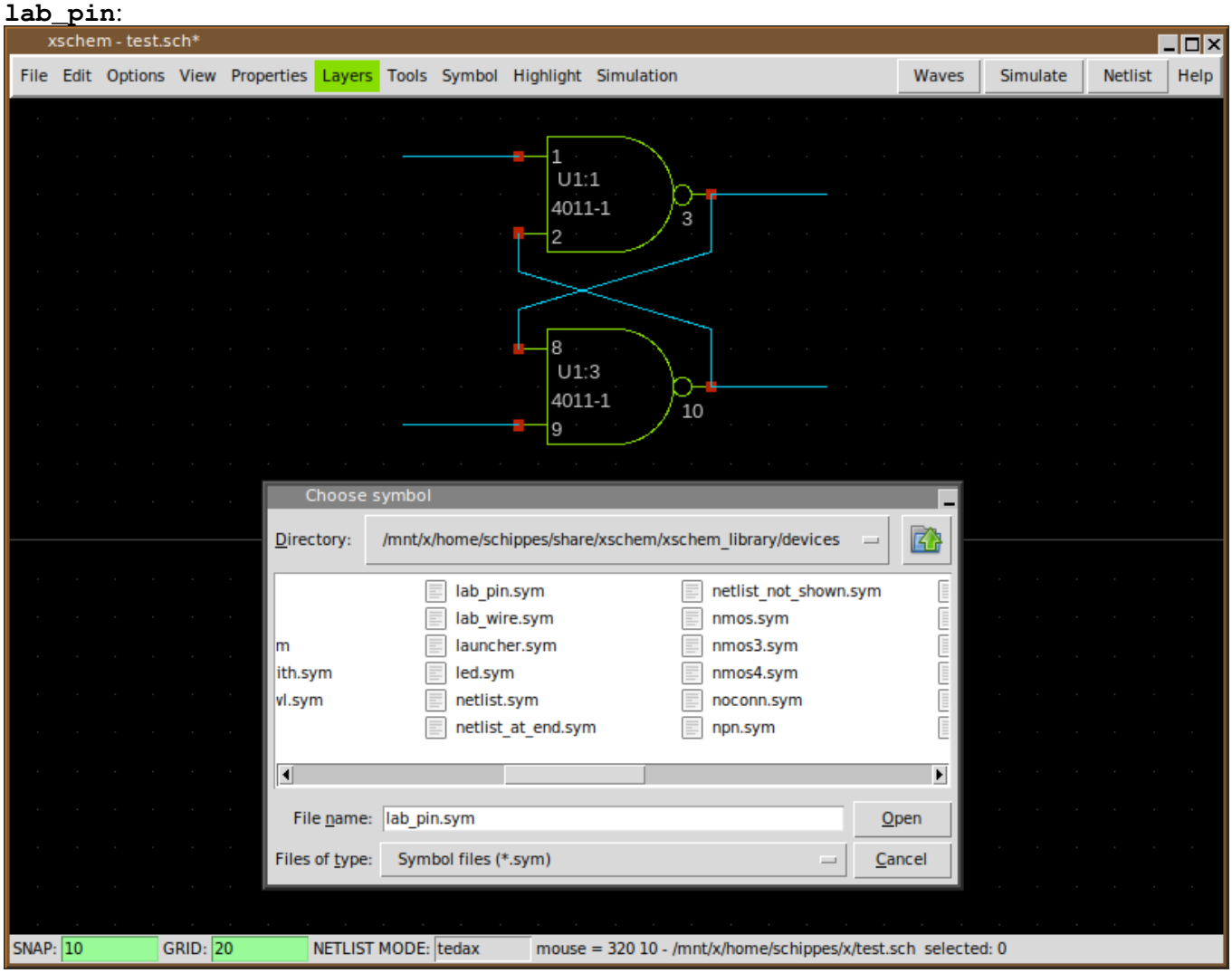

We see that all pin numbers are shown for each pin; this reminds us that this is a slotted device! slotted devices should specify the slot number in the instance **`name`** so, select the component, press **`q`** and change the **`U1`** name attribute to **`U1:1`**. You can also remove the **`.sym`** extension in the 'Symbol' entry of the dialog box, for more compactness:



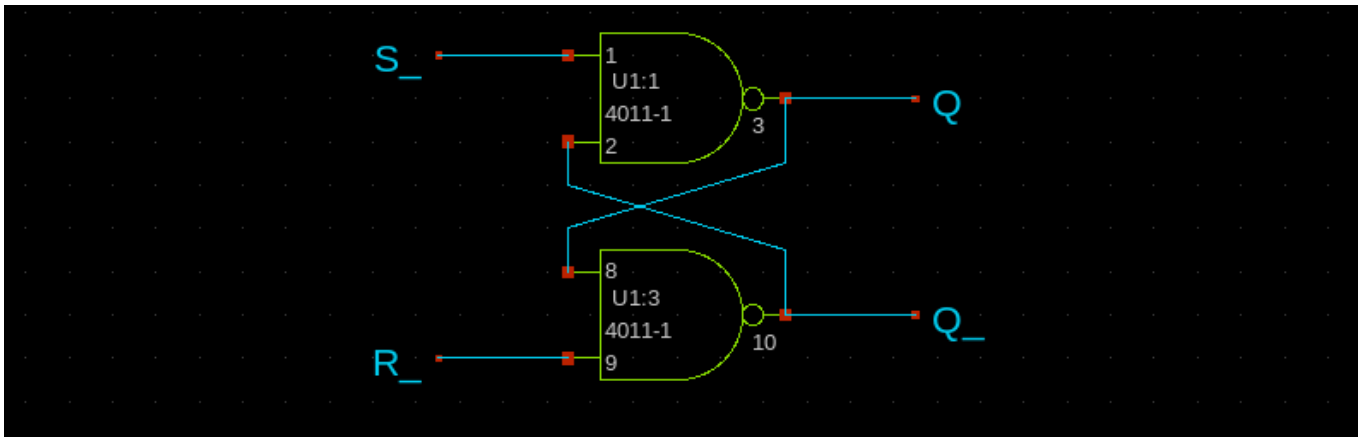

As you can see now the slot is resolved and the right pin numbers are displayed. Now select and copy the component (use the **`c`** key), and change the **`name`** attribute of the new copy to **`U1:3`**:



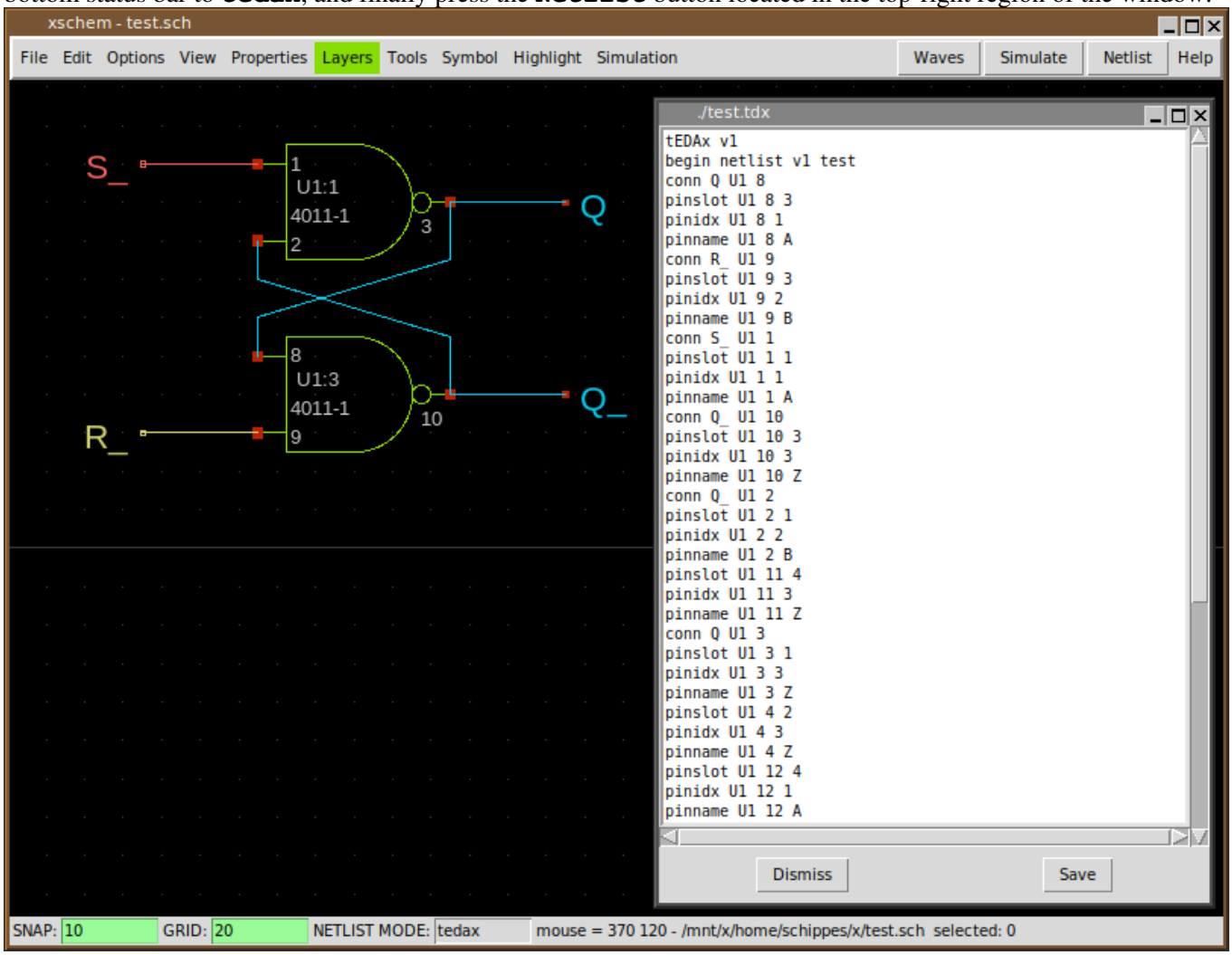

10. Now draw some wires, for example to create an SR latch as shown, use the **`w`** key to draw wires; when done with the wiring insert a net label by pressing the **`Insert`** key and navigating to **`.../share/xschem/xschem_library/devices`** (the XSCHEM system symbol library) and selecting **`lab_pin`**:




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:




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.

UP

# TUTORIAL: Manage XSCHEM design / symbol libraries

There are 2 ways to describe symbols 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

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/**)

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**'

In any case the real path of the symbol reference is obtained by prepending the XSCHEM_LIBRARY_PATH paths to the symbol reference until the resulting file is found in the machine filesystem.

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.**

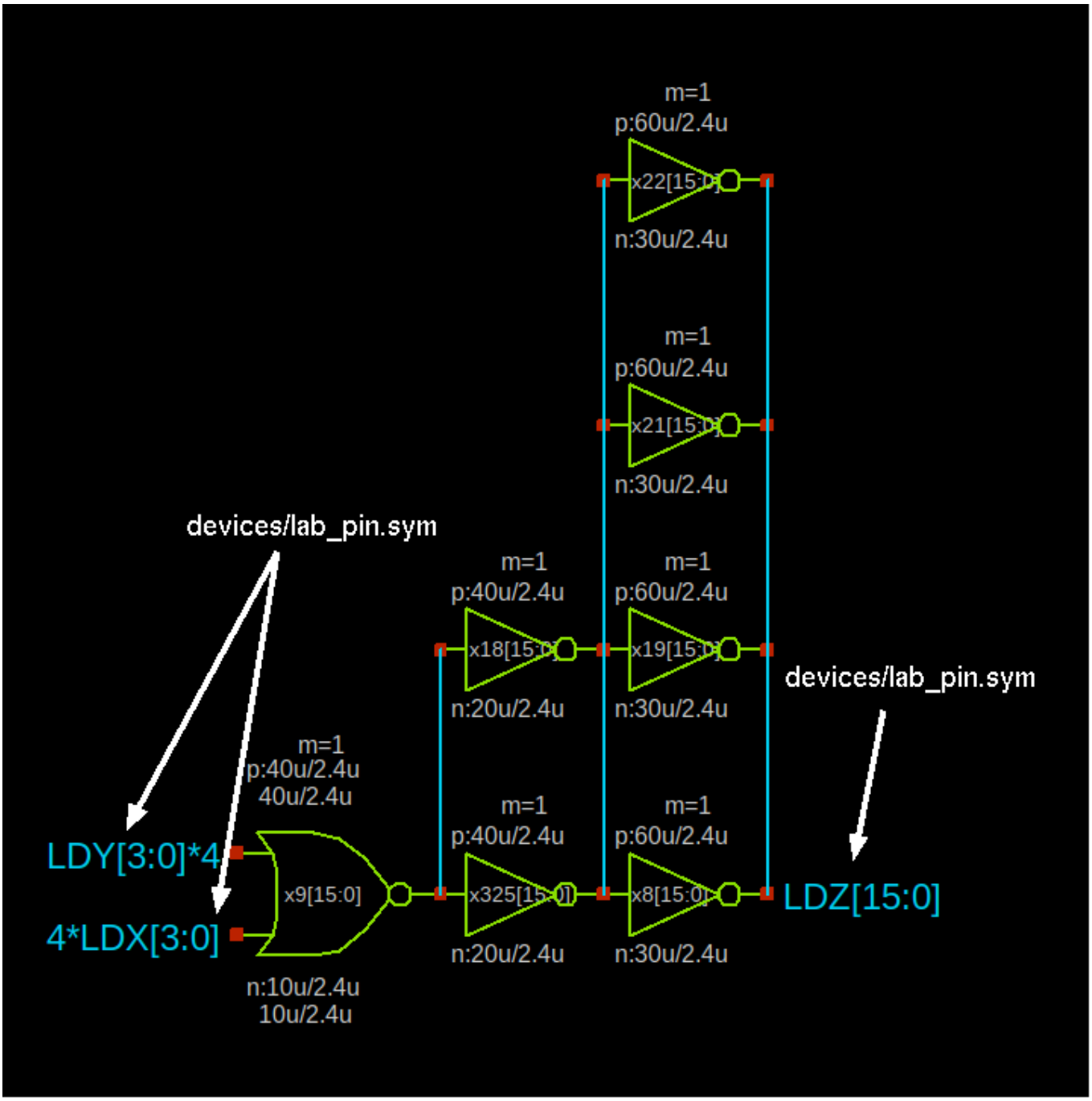
UP

# 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]`**.
- **`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`**.

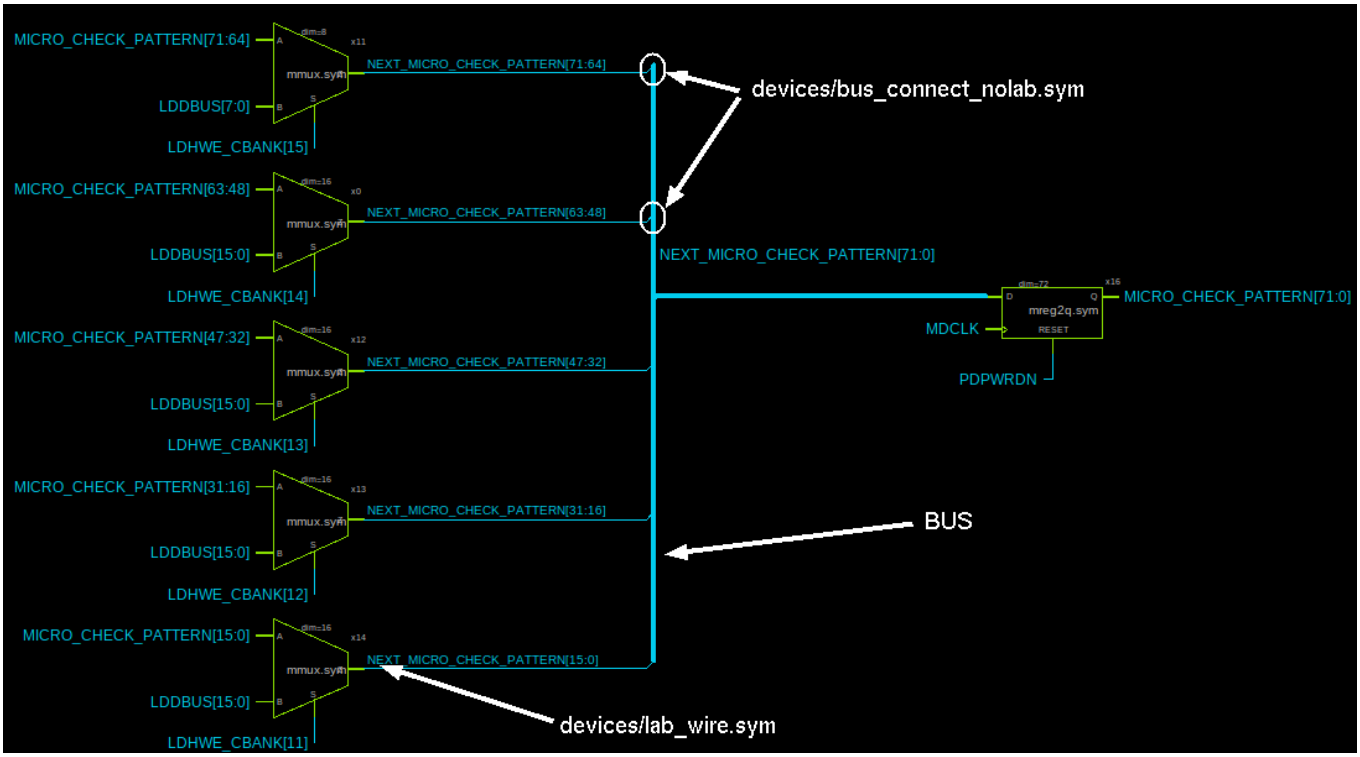
All the above notations are perfectly valid label net name attributes.
In a very similar way multiple instances can be placed in a schematic setting the 'name' attribute to a vector notation. For example in picture below **`x22[15:0]`** represents 16 inverters with names **`x22[15],x22[14],...,x22[0]`**.



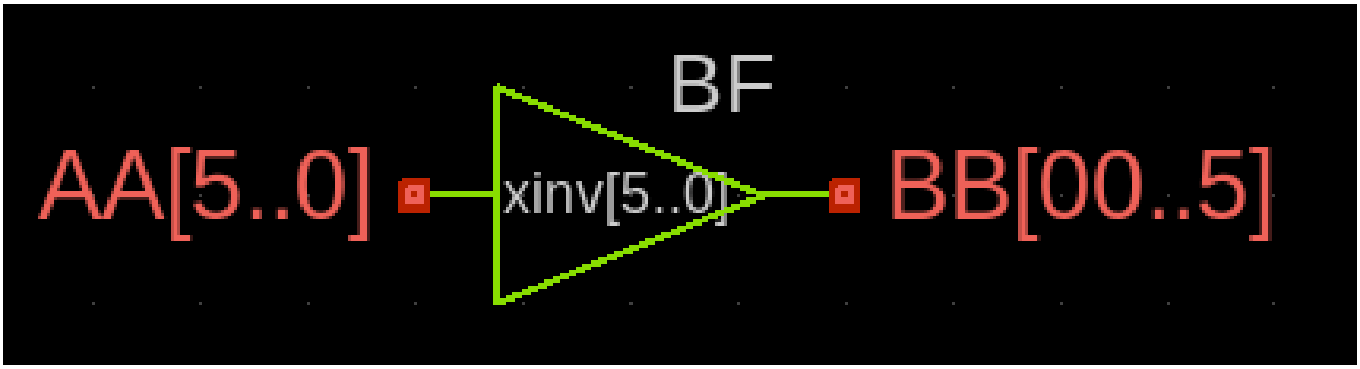

Recently a new notation has been added for buses that expands without putting brackets:

- **`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`**.



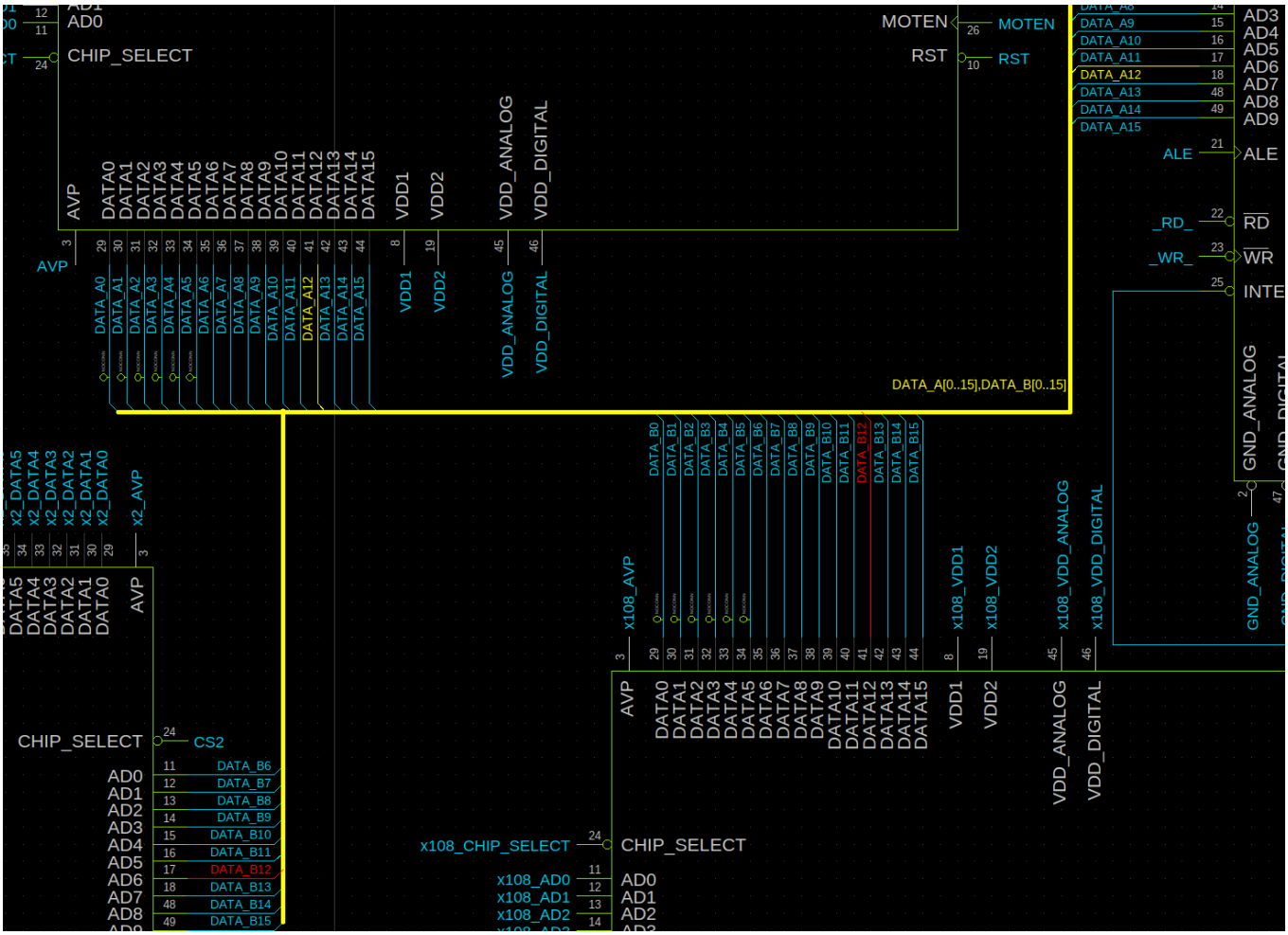

following picture shows an istantiation of 6 inverters:




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
...
```

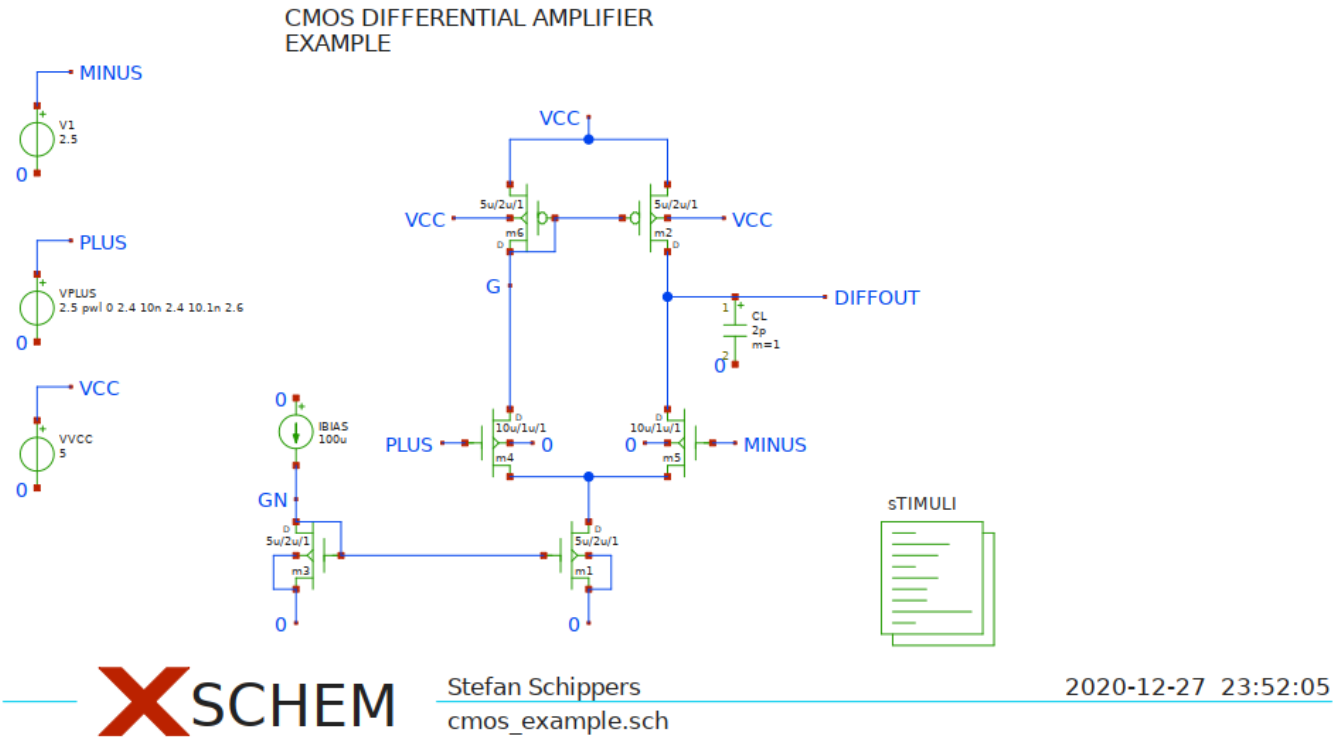
Example of a more complex bus routing. main bus is a bundle of 2 buses: DATA_A[0..15] and DATA_B[0..15]



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.




## CONFIGURATION

Open your **`xschemrc`** file (usually **`~/.xschem/xschemrc`**), go to the end of the file, Ensure the following tcl file is appended to the list of scripts loaded on startup by Xschem:

```
#### list of tcl files to preload.
lappend tcl_files ${XSCHEM_SHAREDIR}/ngspice_backannotate.tcl
```

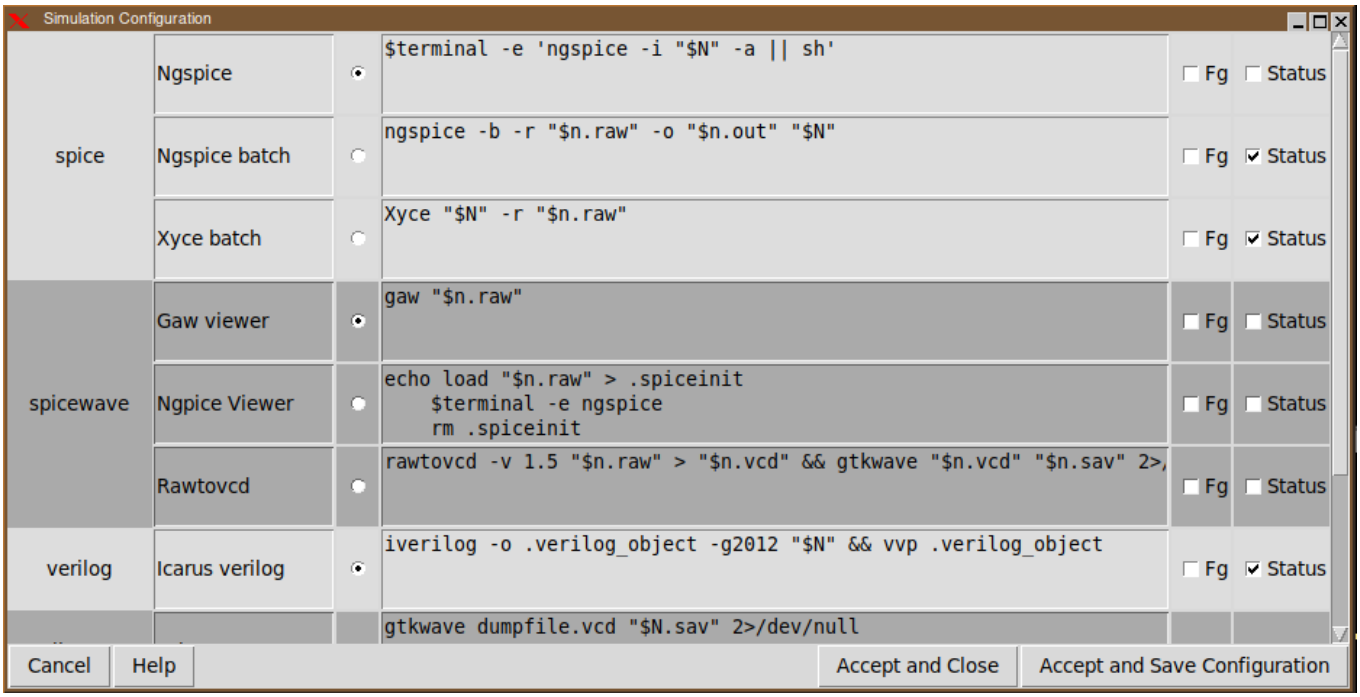
## SETUP

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.
```

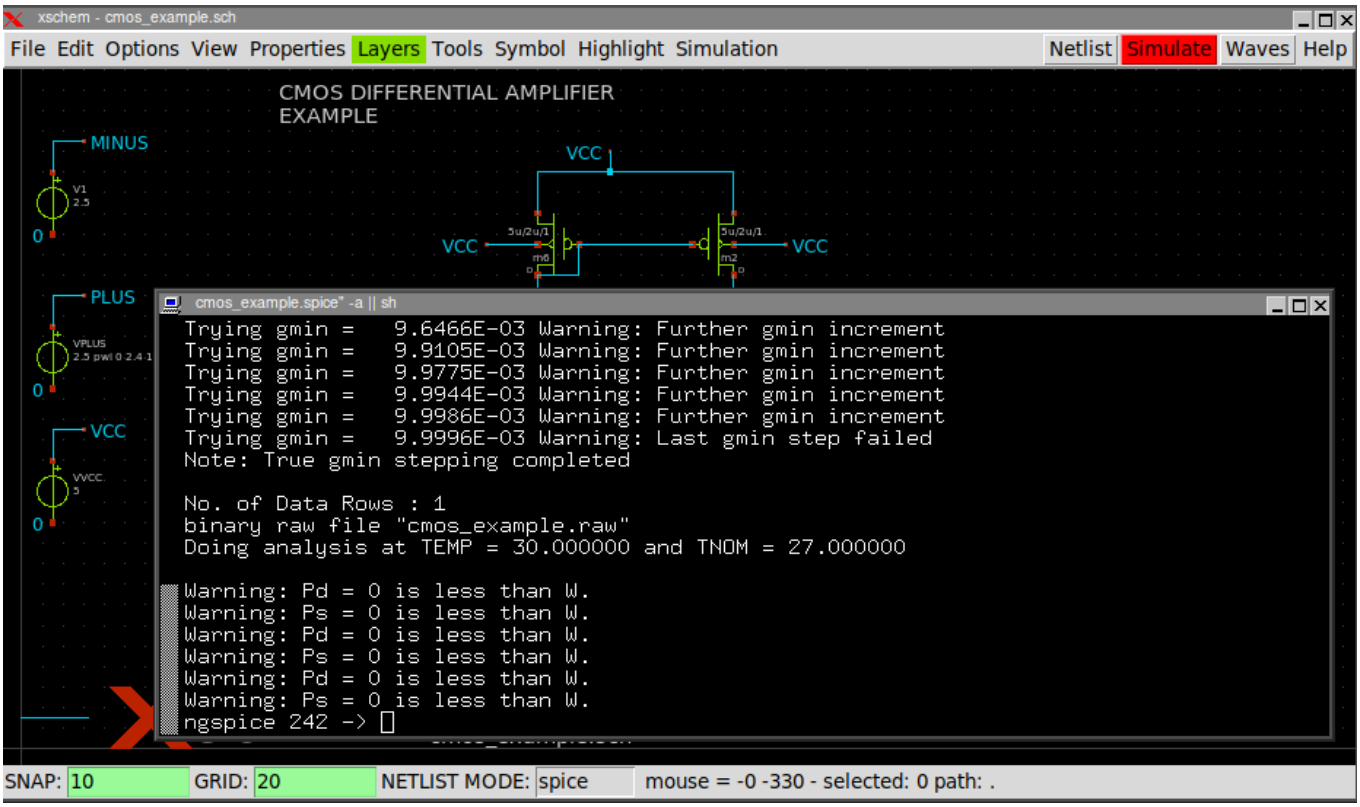
```
.include "models_cmos_example.txt"
.control
 op
 save all
 write cmos_example.raw
.endc
```

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.
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

If you now press the **`Netlist`** followed by the **`Simulate`** button simulation should complete with no errors.



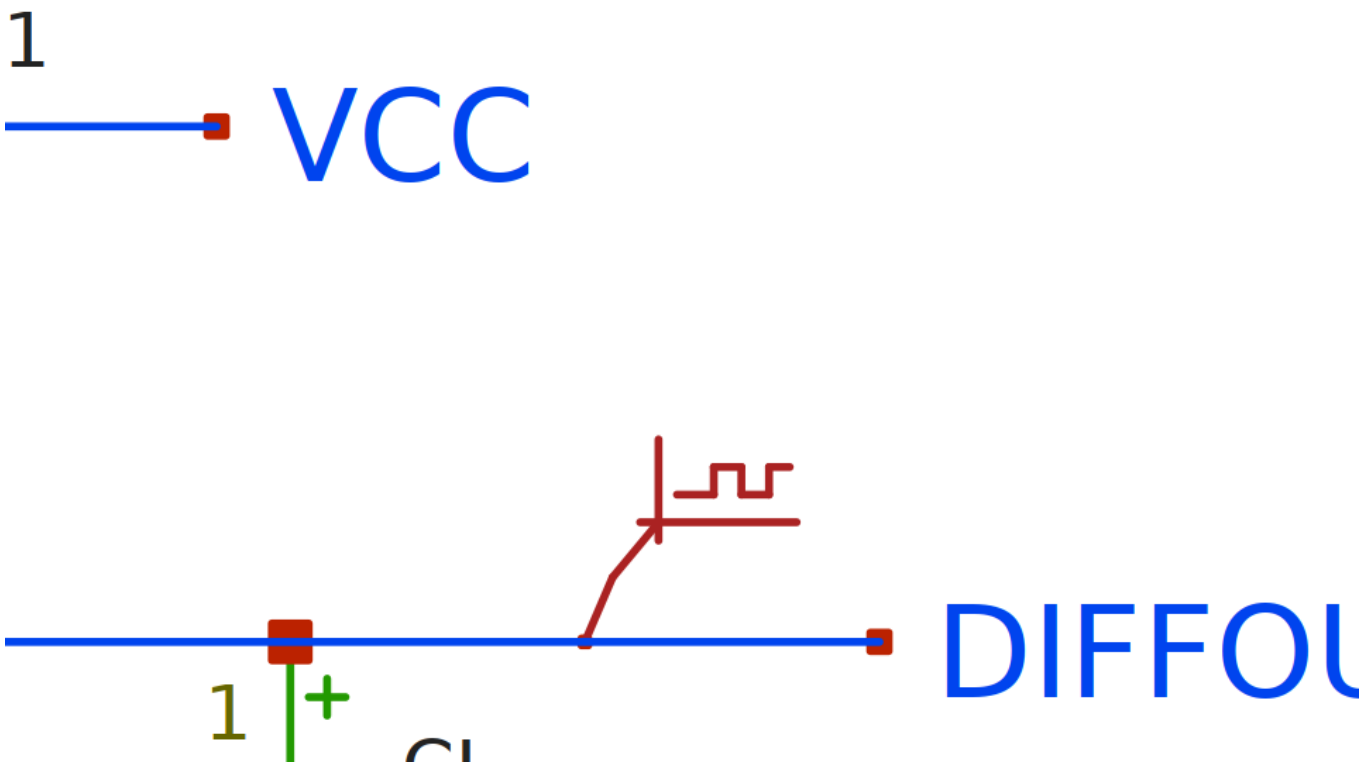

You can close the simulator since we need only the **`cmos_example.raw`** file that is now saved in the simulation directory (usually **`~/.xschem/simulations/cmos_example.raw`**).
Now verify that xschem is able to read the raw file: issue this command in the xschem console:
**`ngspice::annotate`**

```
xschem [~] ngspice::annotate
Raw file read ...
xschem [~]
```

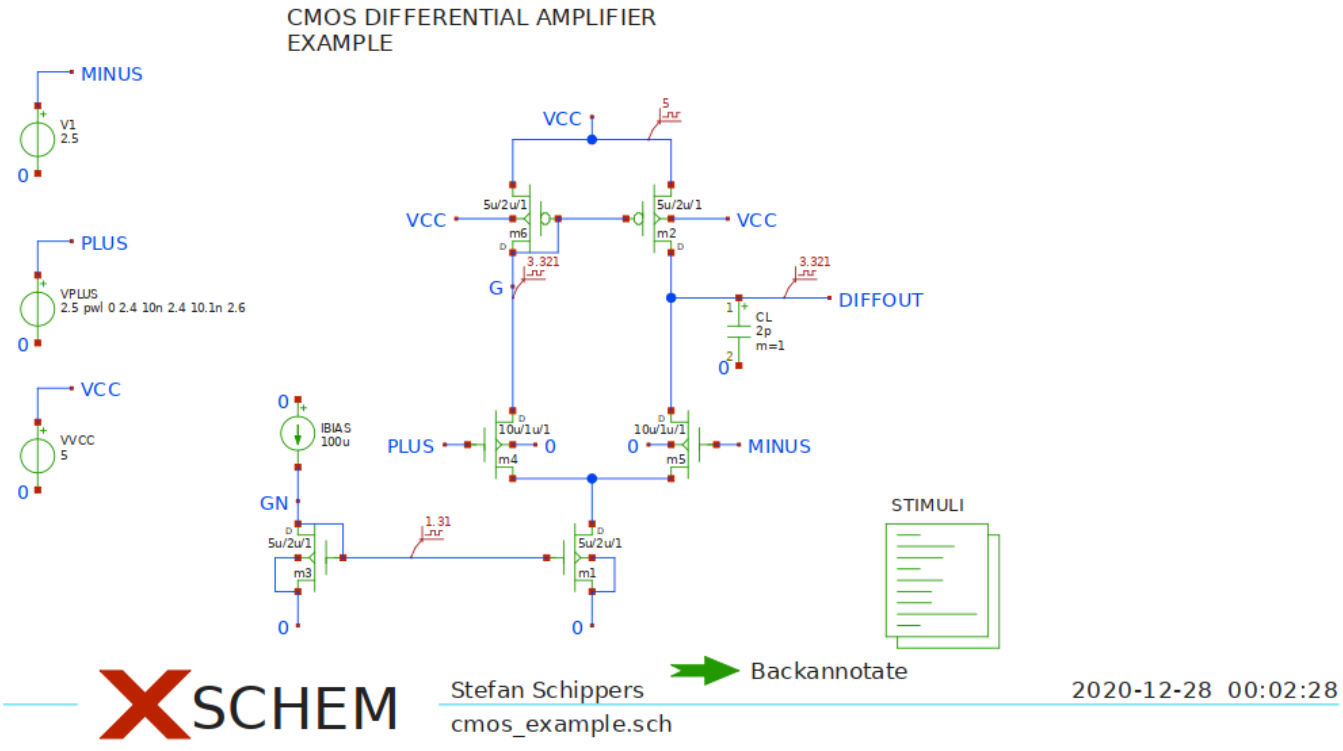
If there are no errors we are ready and set.

## PUSH ANNOTATION METHOD

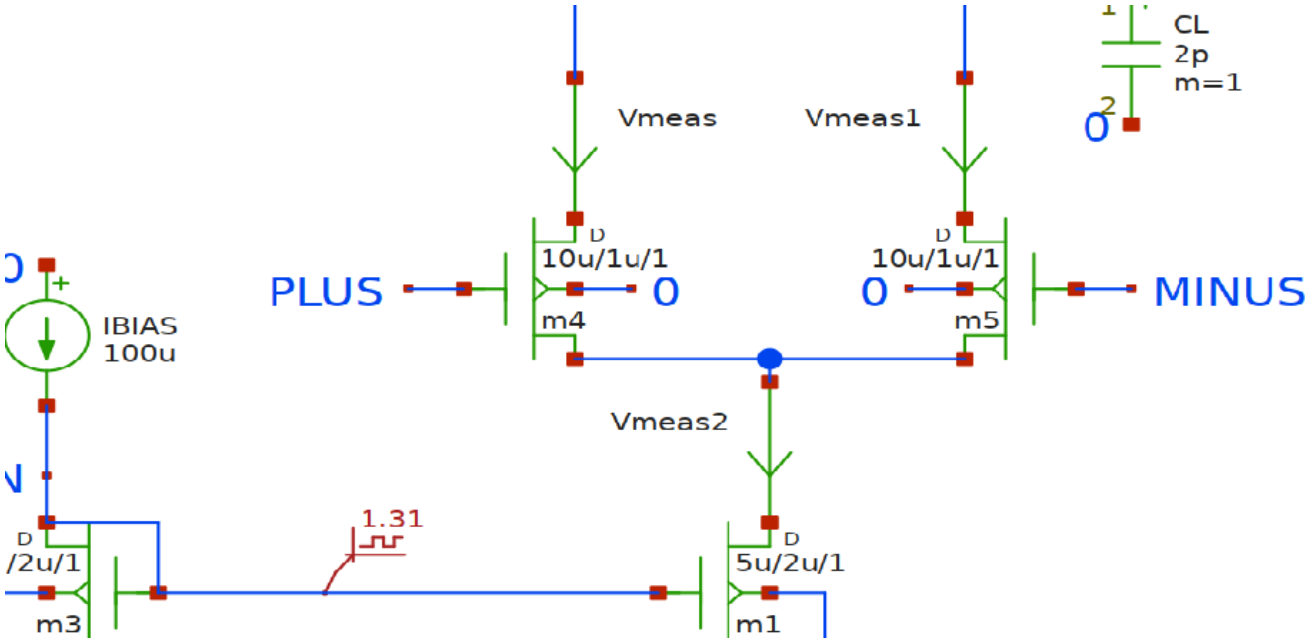
Start placing some probe elements into the schematic. The first element is the **`devices/spice_probe.sym`** component. This must be attached to some schematic wires to show the voltage value.




Place some of these elements on various nets, issue the above mentioned **`ngspice::annotate`** command and see the voltage values in the schematic.

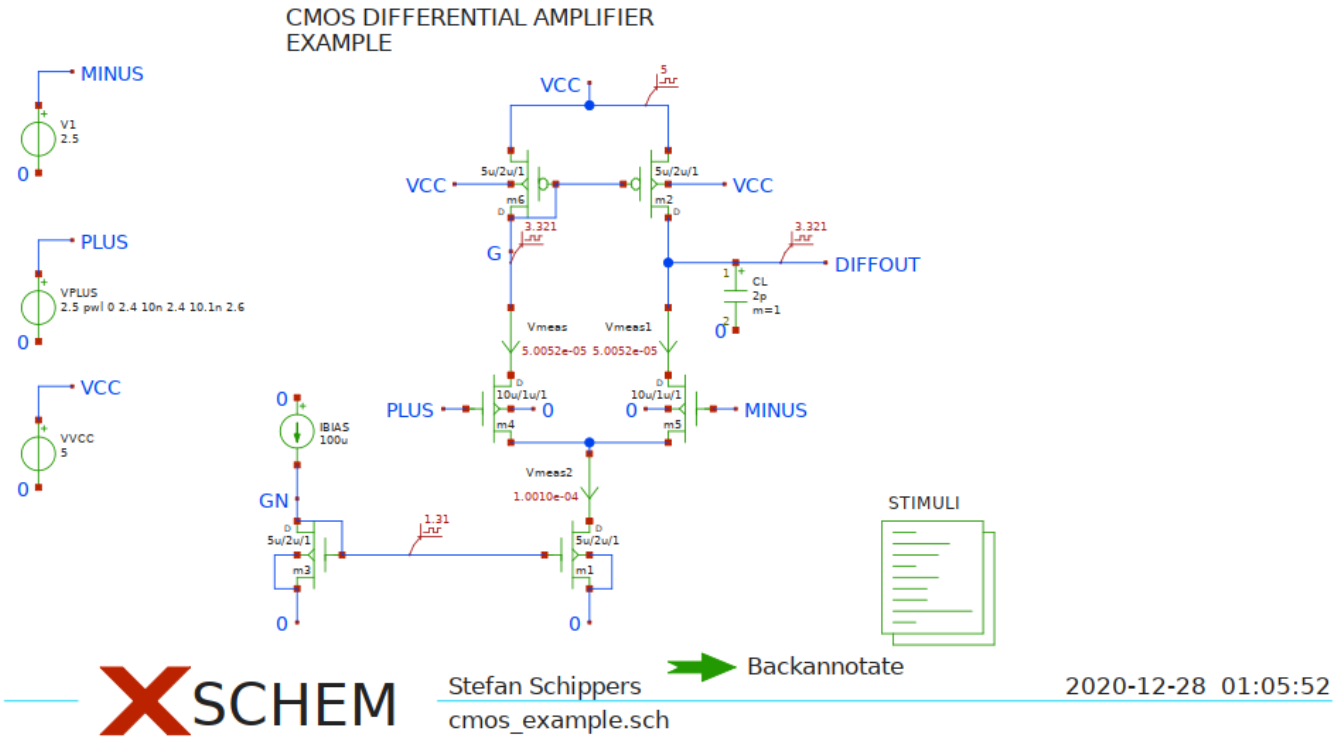



Another useful component is the **`devices/ammeter.sym`** one which allow to monitor branch currents. Break some wires and insert this component as shown here:



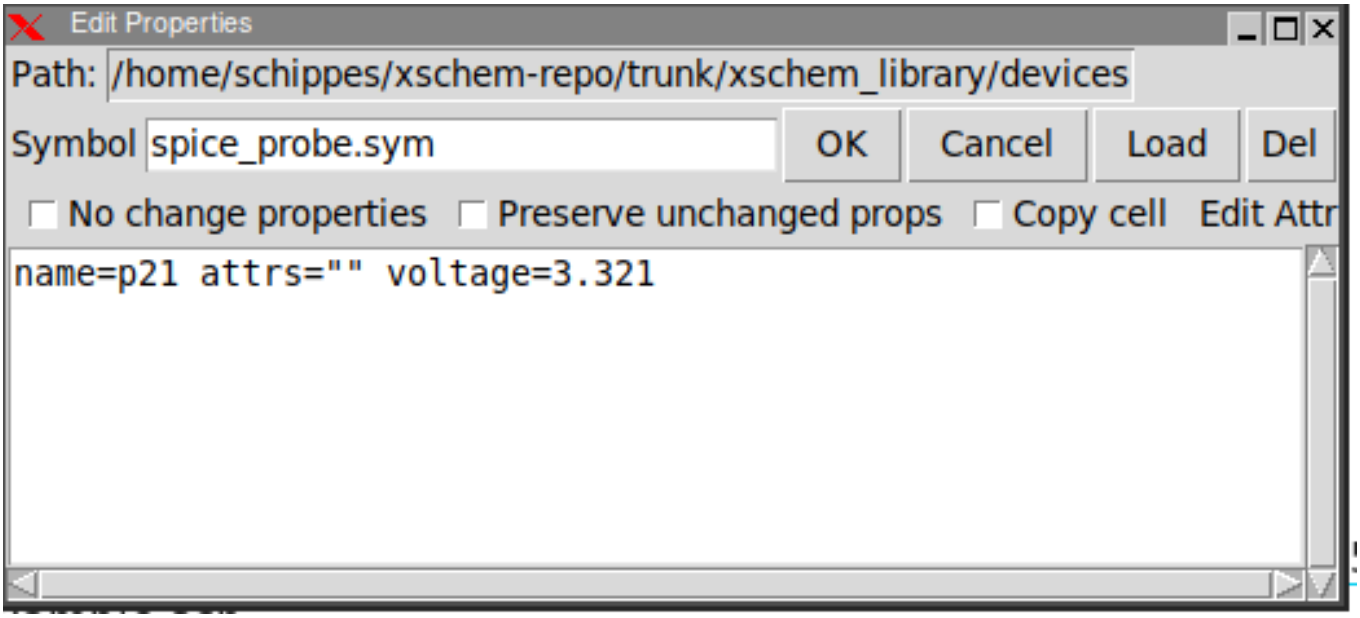

IMPORTANT: When inserting current probes the circuit topology changes (new nodes are created) so you need to re-create the netlist and re-run the simulation

Doing again the **`ngspice::annotate`** command after simulation will update the ammeters showing the branch currents.




These voltage and current values are inserted in the probe components as **attributes** and thus can be saved to file. Remember that if you change the circuit the values shown in the probe elements are no longer valid, you should update the values with a new simulation + annotate operation when done with the changes.

What i have described so far is the simplest annotation procedure based on a **push** method: a tcl script reads the simulation raw file and 'pushes' voltage and current values into the probe components as instance attributes. If you do an edit attribute on one of these elements you see the attribute 'pushed' into it by the annotate script. The advantage of this method is that values pushed into probes can be saved to file and are thus persistent.



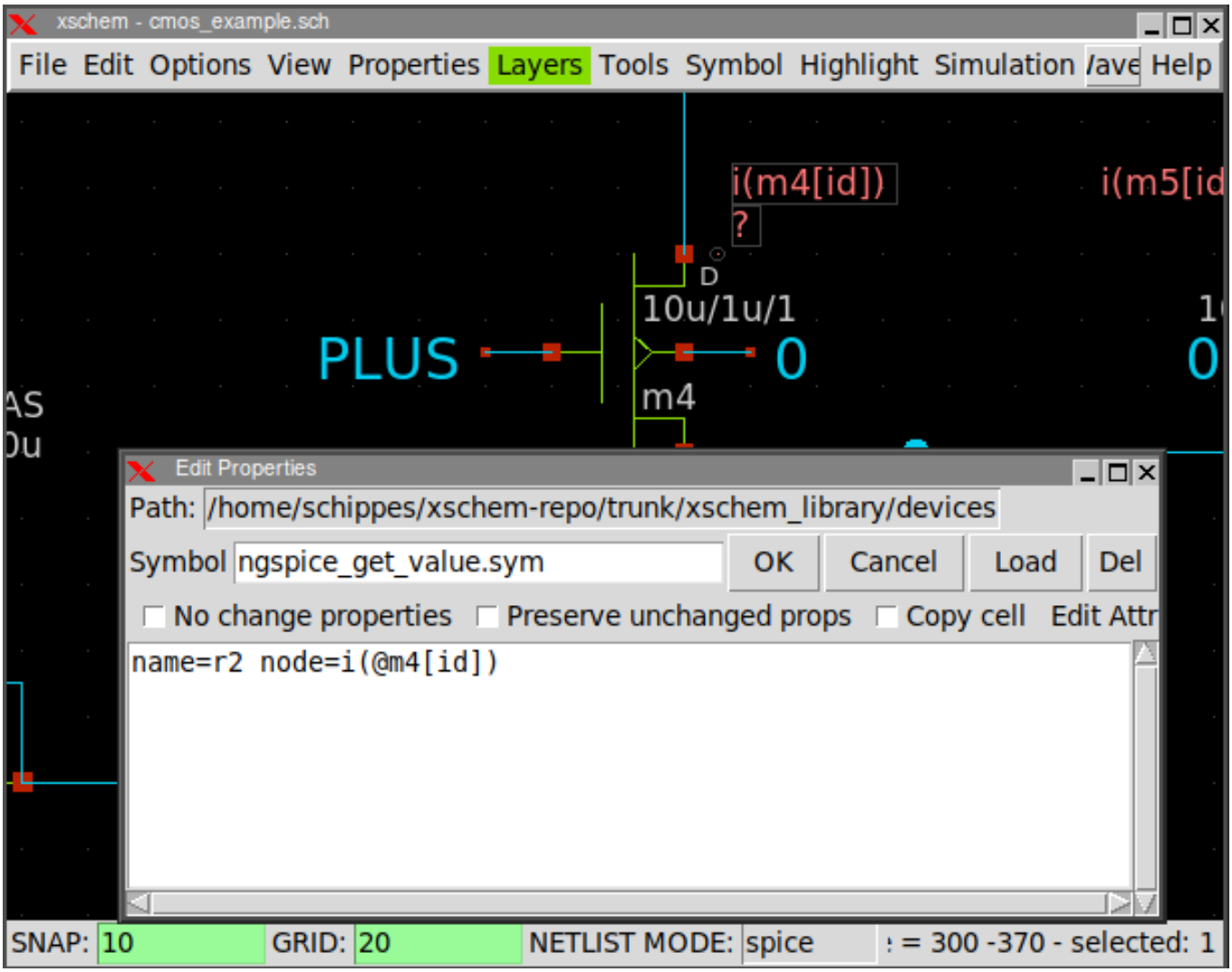

## PULL ANNOTATION METHOD

There is another annotation procedure that is based on a **pull** method: the probe objects have tcl commands embedded that fetch simulation data from a table that has been read by the annotate script.

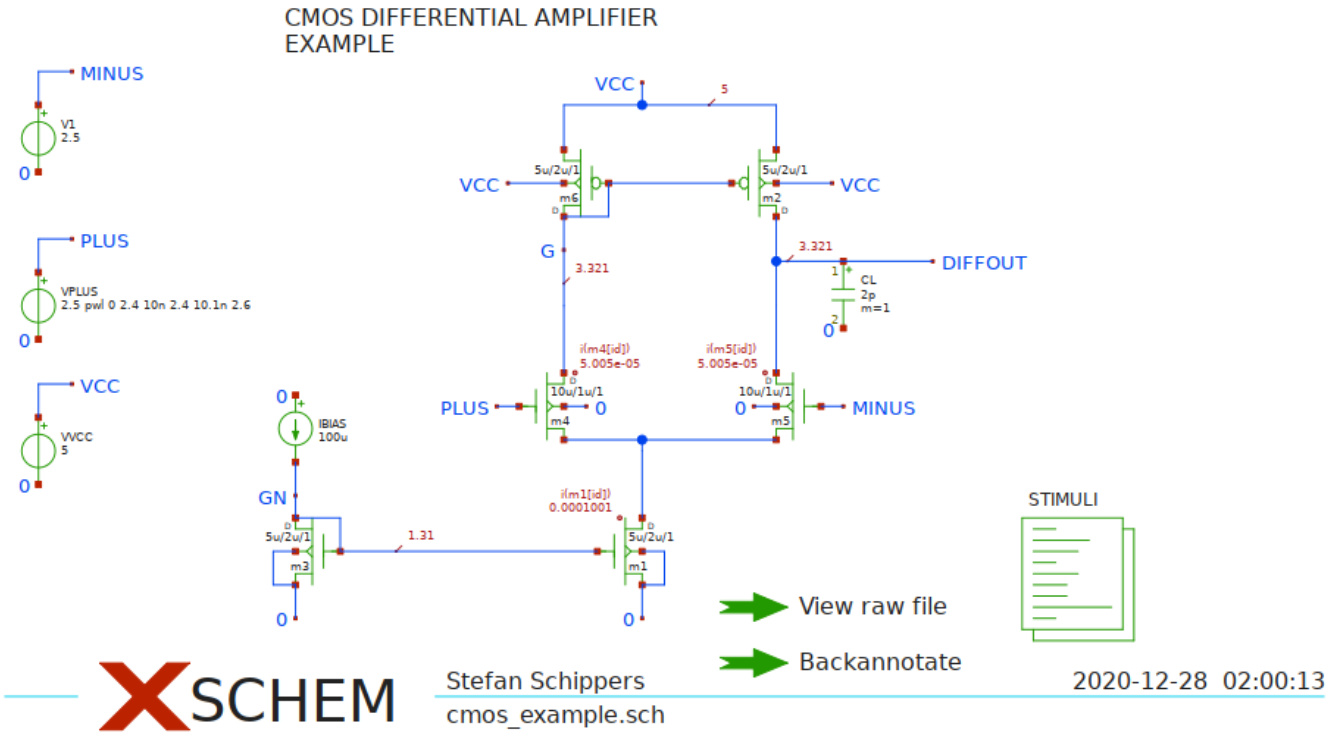
To ensure all currents are saved modify the **`STIMULI`** attributes as follows:

```
.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.

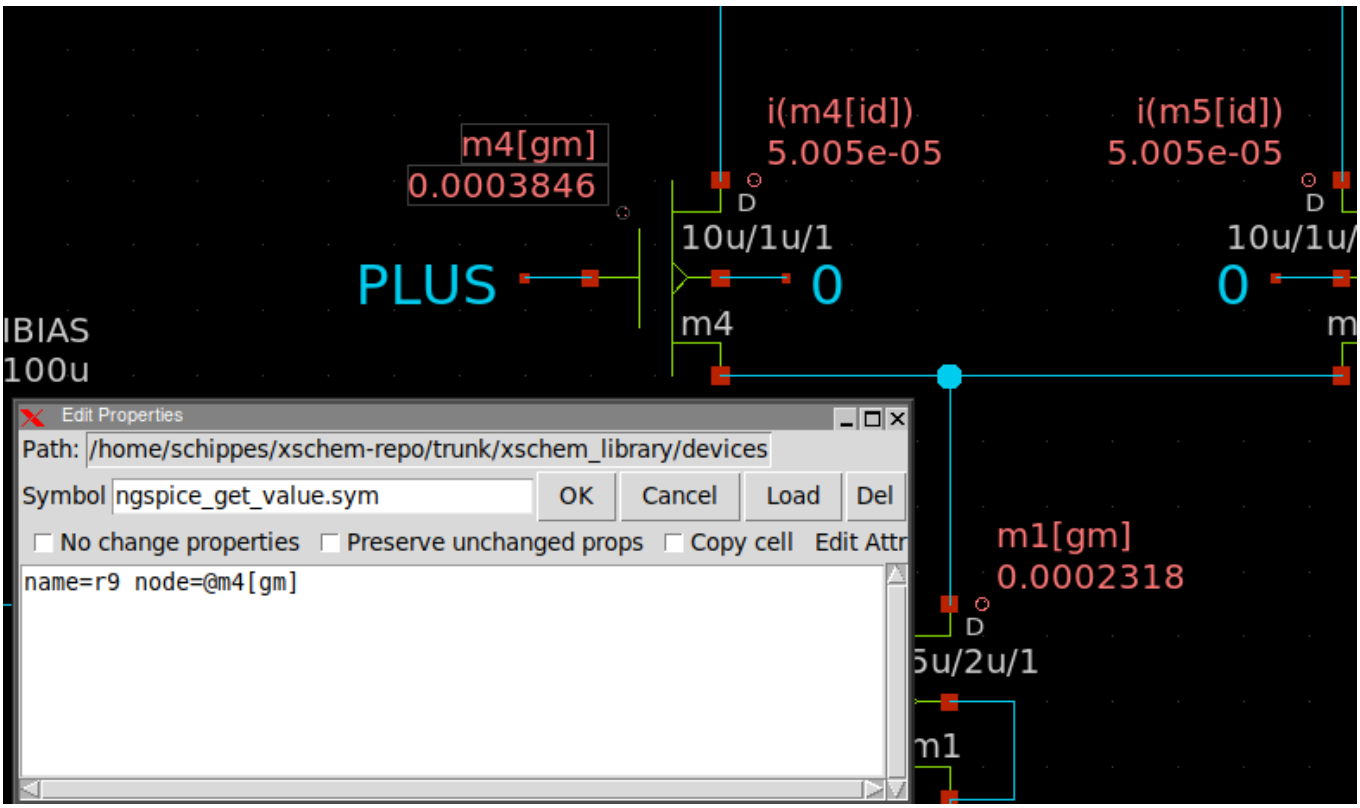



Run again the simulation and the **`ngspice::annotate`** command and values will be updated.




You can add additional variables in the raw file , for example modifying the .save instruction:

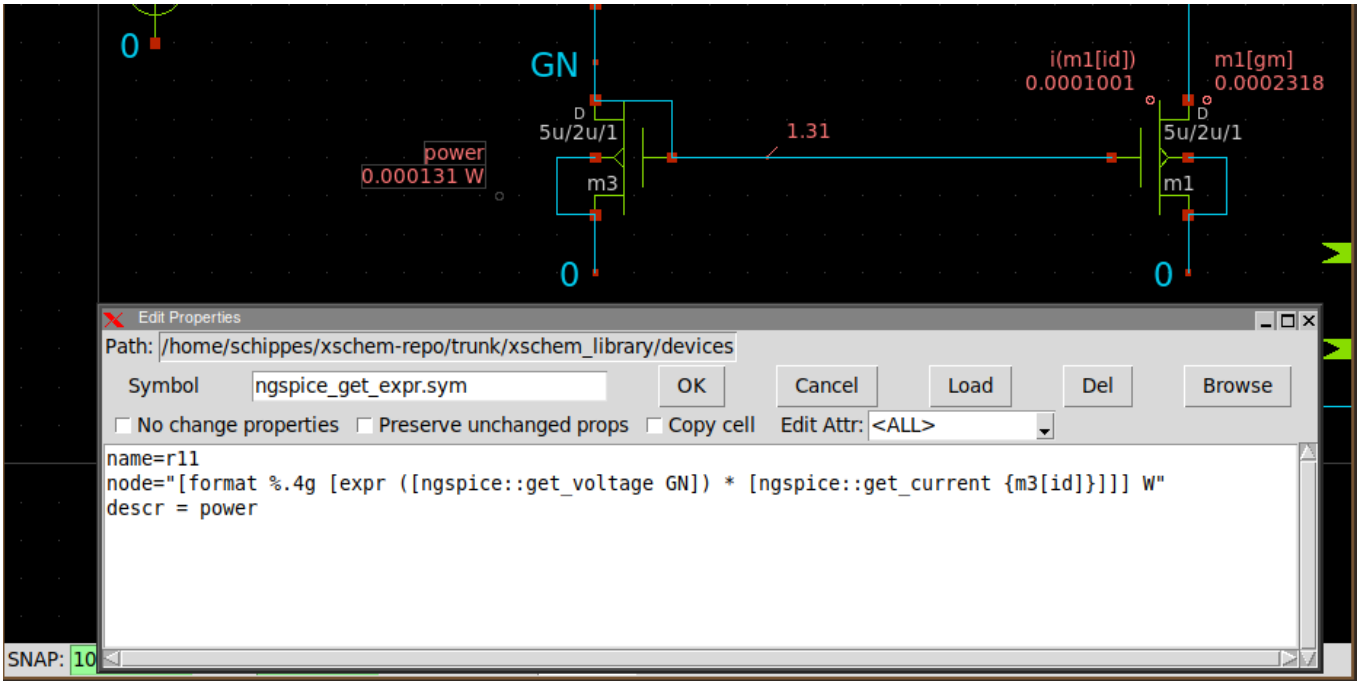
```
.save all @m4[gm] @m5[gm] @m1[gm]
```



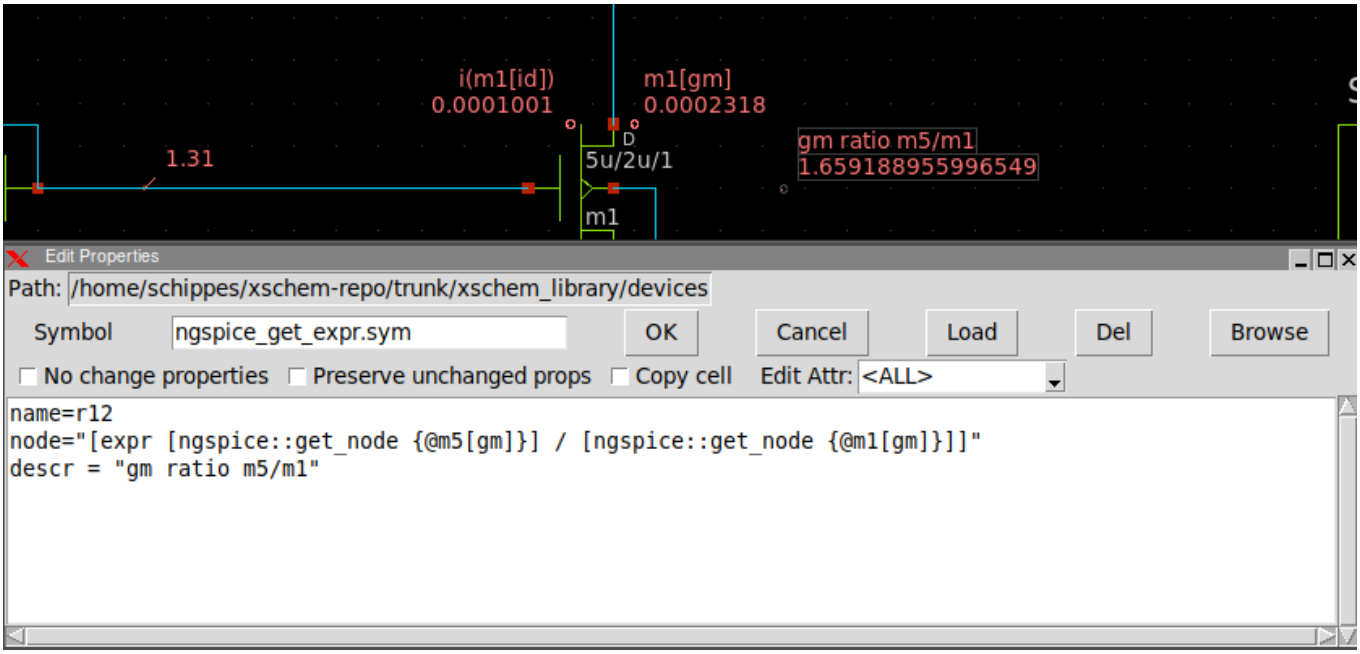

Data annotated into the schematic using these components allows more simulation parameters to be viewed into the schematic, not being restricted to currents and voltages. Since these components get data using a pull method data is not

persistent and not saved to file. After reloading the file just do a **`ngspice::annotate`** 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)`**.

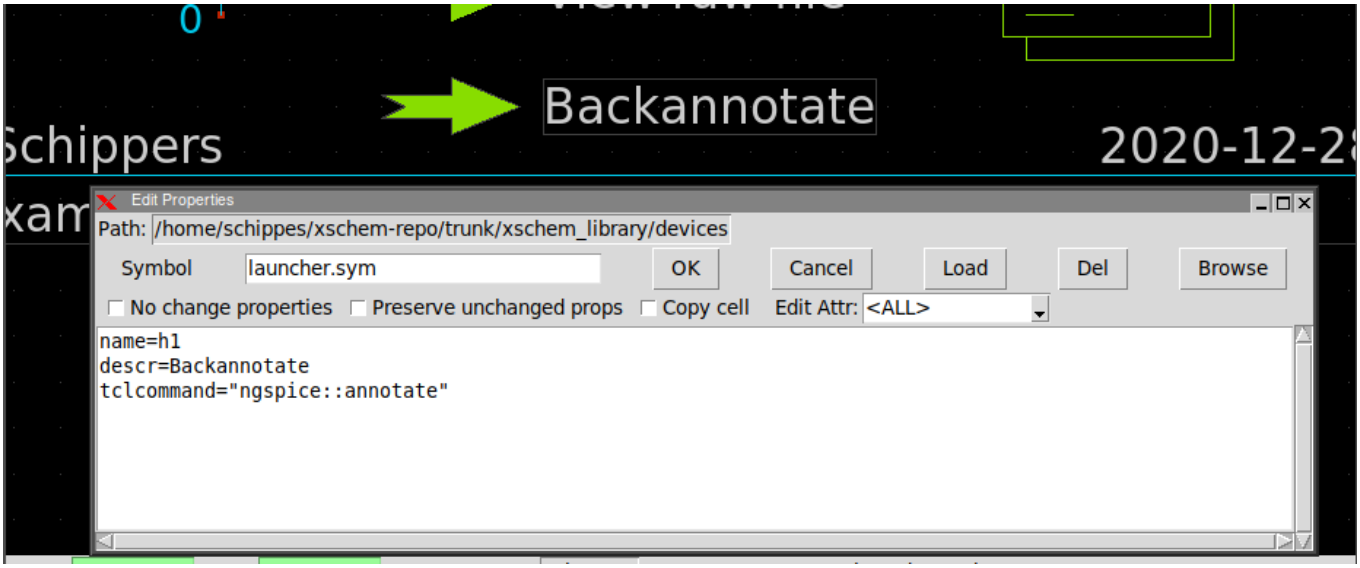



The example shown uses this component to display a (meaningless, but shows the usage) gm ratio of 2 transistors:




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.



UP

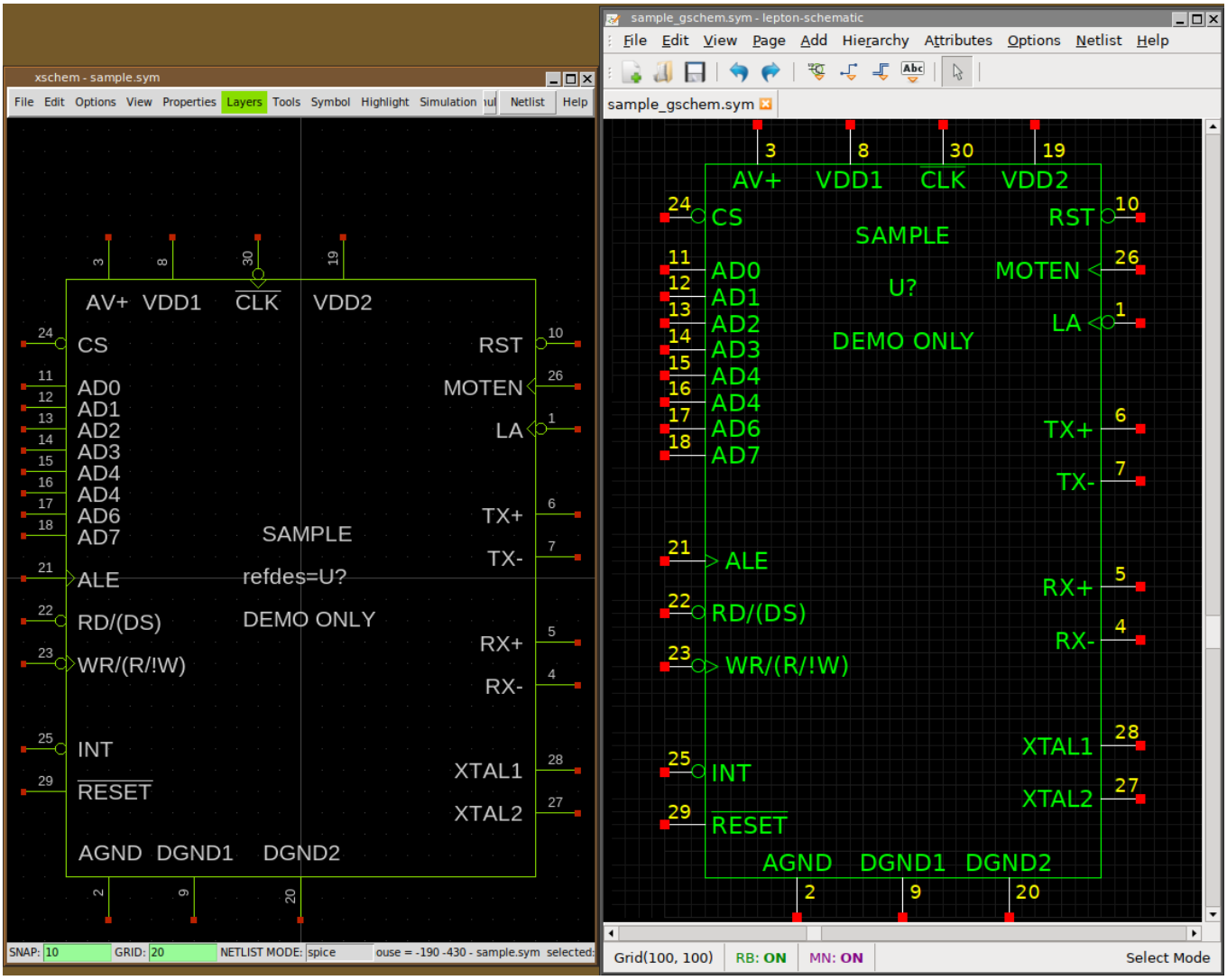
# TUTORIAL: Use symgen.awk to create symbols from 'djboxsym' compatible text files

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:

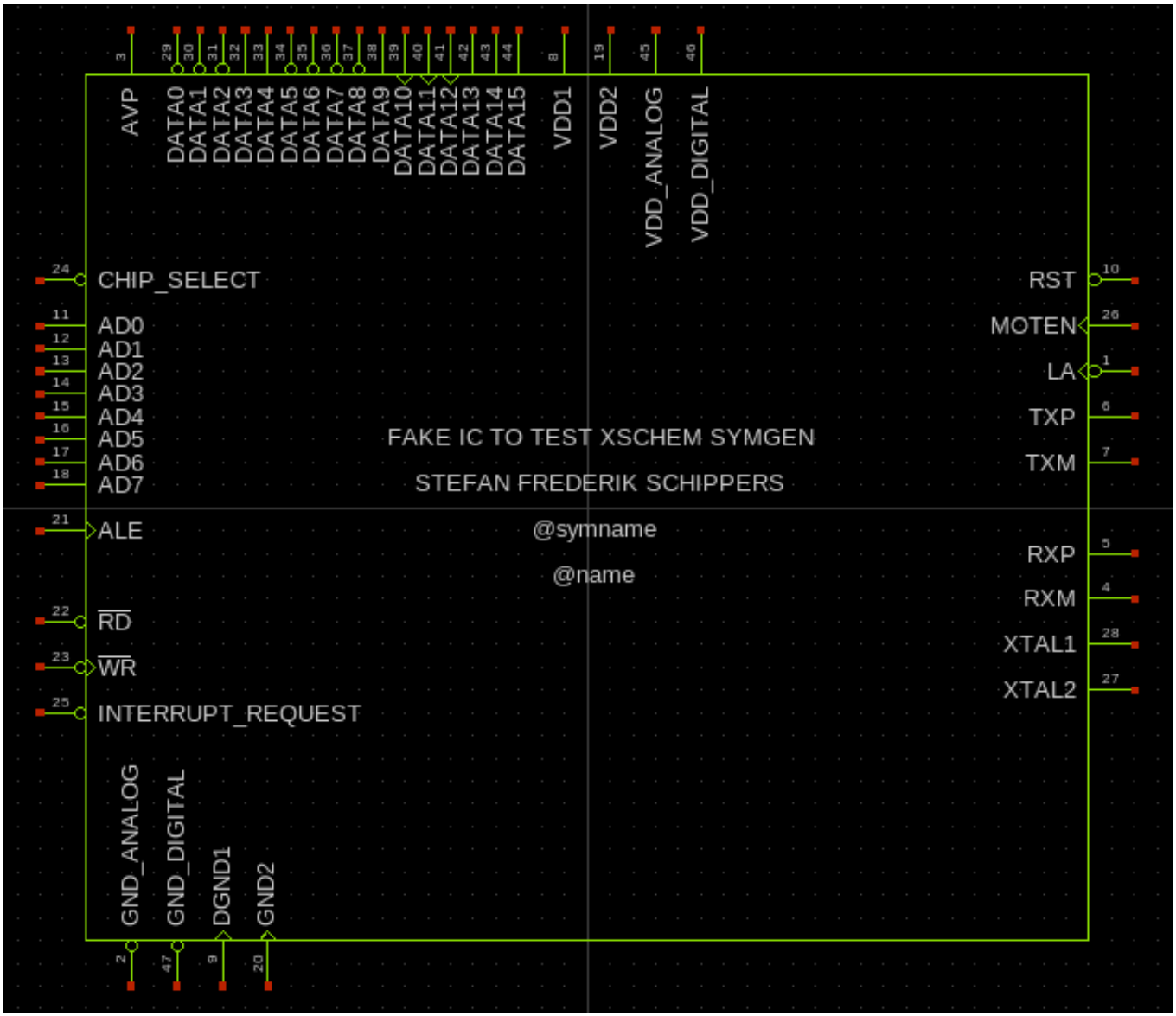



Another **`sample2.symdef`** file specifically created to generate a perfectly valid xschem symbol (including attributes for spice netlisting) is the following:

```
#  <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:

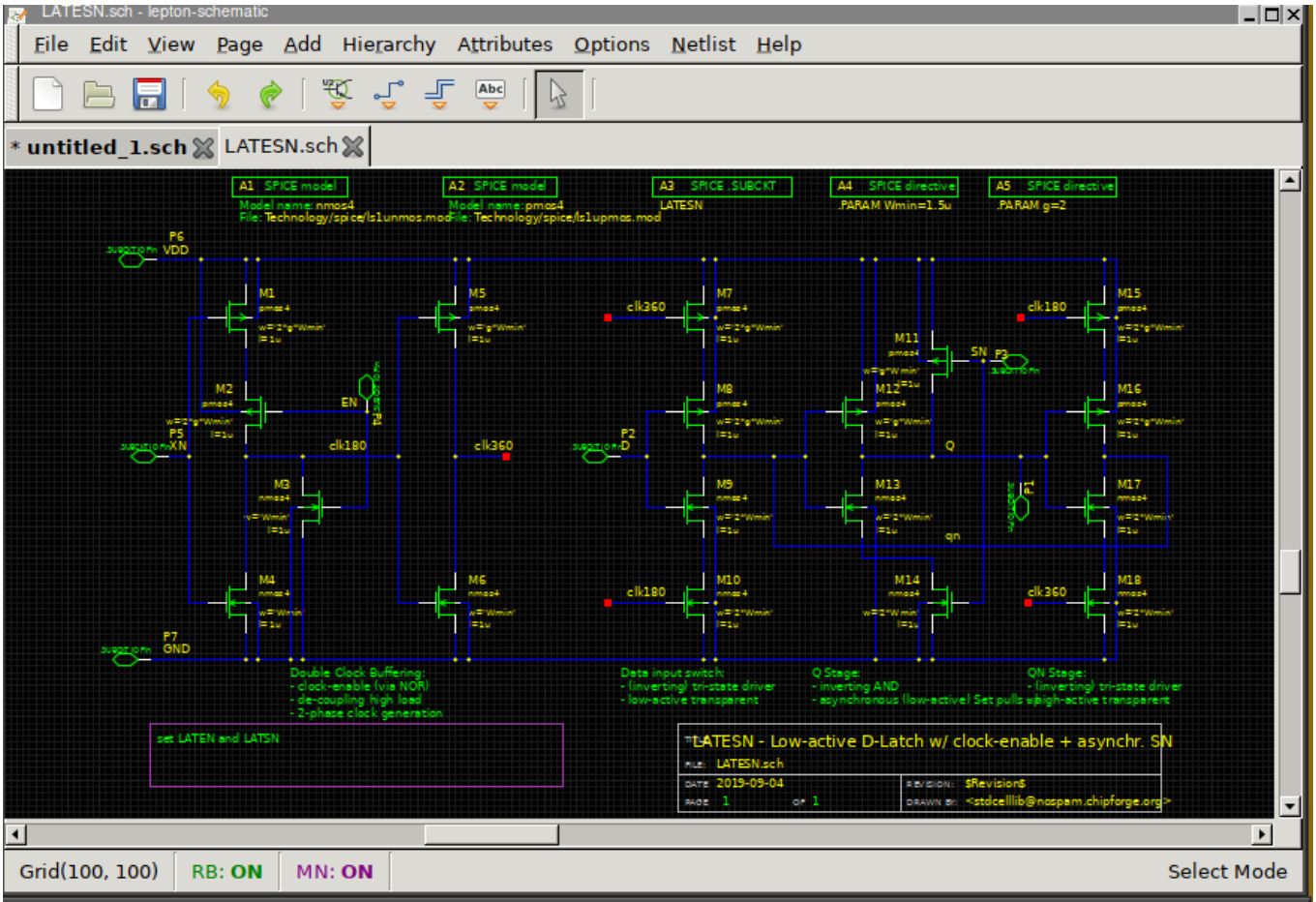
```
append XSCHEM_LIBRARY_PATH :${XSCHEM_SHAREDIR}/../doc/xschem/gschem_import/sym
```

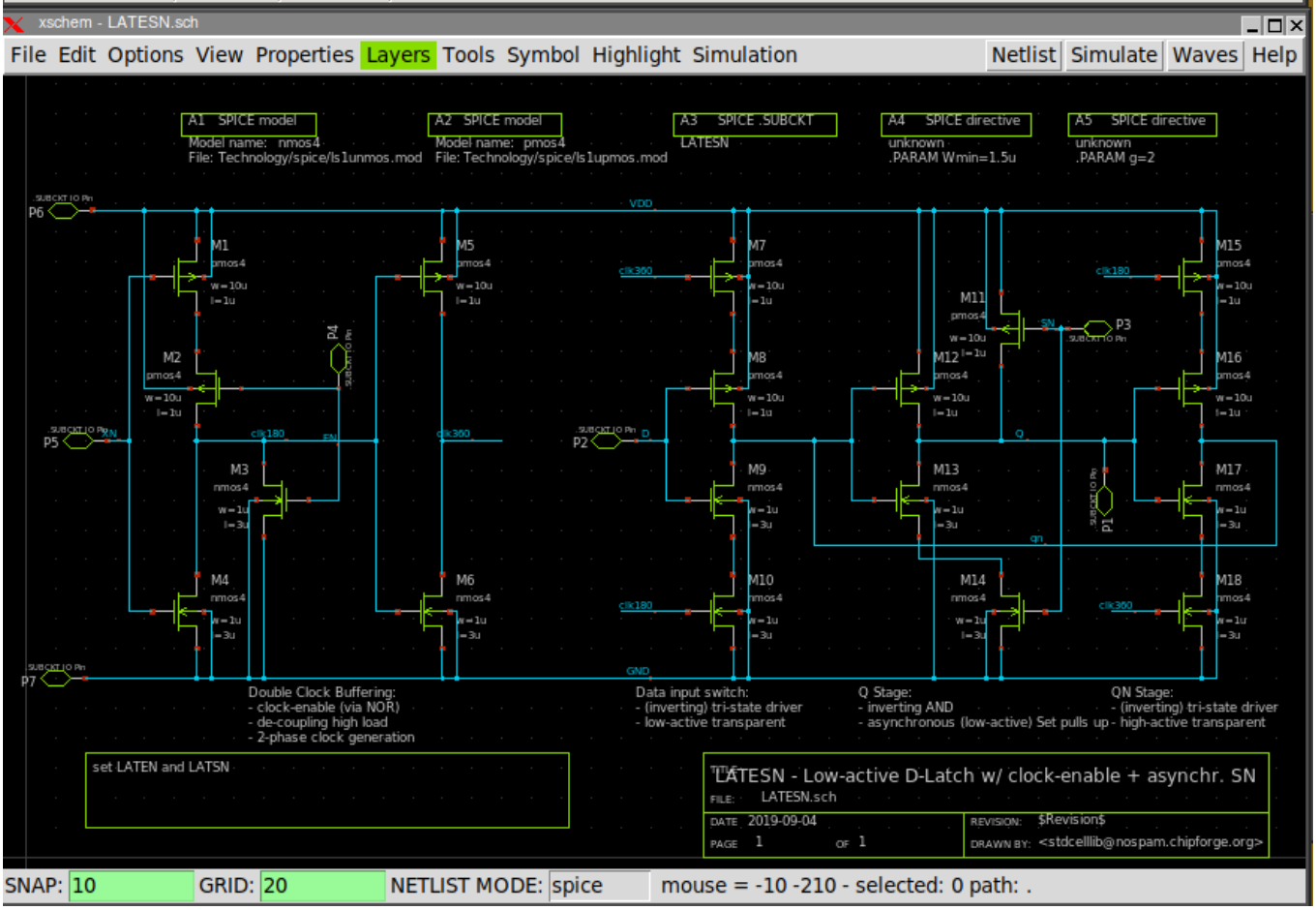
Next, in this directory create a **convert.sh** script and make it executable:

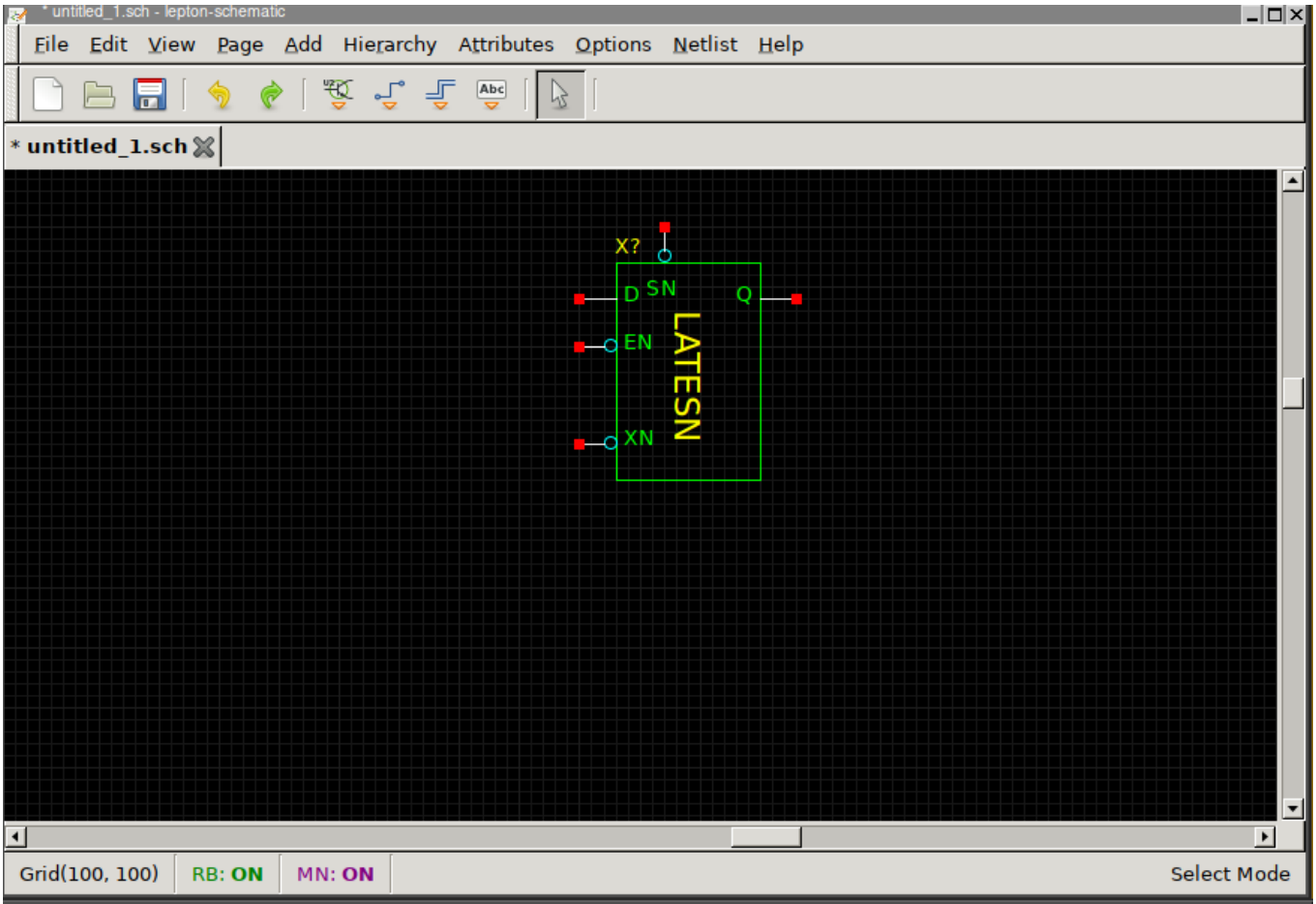
```
#!/bin/bash

# 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
```

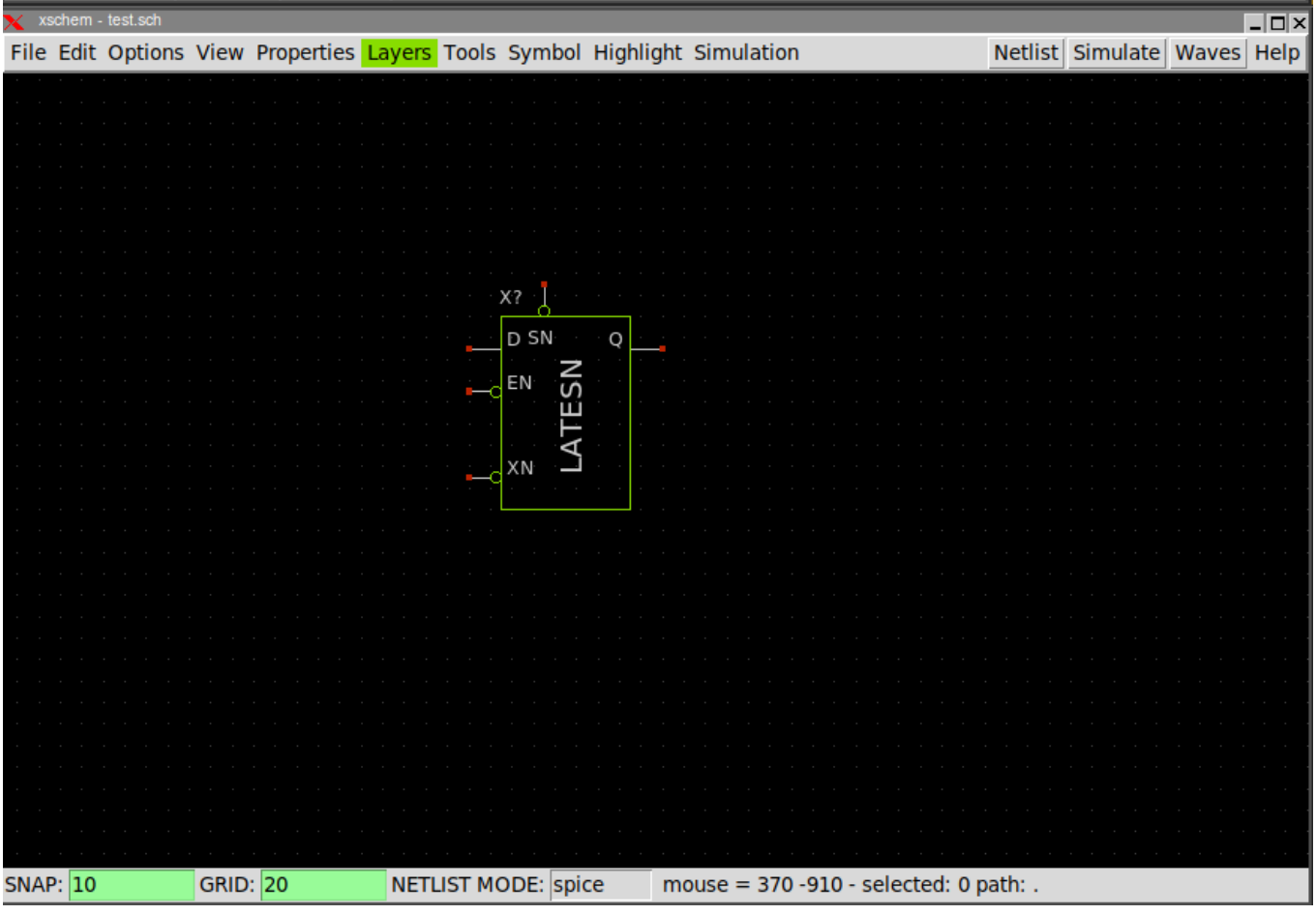
Note that you have to set the correct path for **gschemtoxschem.awk** depending on your xschem installation and set the correct path for the directory (**directory_with_geda_files** in above example) containing the geda files.
The current directory will be populated with xschem schematics/symbols with 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)


LATESN.sch - lepton-schematic
File Edit View Page Add Hierarchy Attributes Options Netlist Help
untitled_1.sch
LATESN.sch
A1 SPICE model
Model name: nmos4
File: Technology/spice/ls1unmos.mod
A2 SPICE model
Model name: pmos4
File: Technology/spice/ls1upmos.mod
A3 SPICE .SUBCKT
LATESN
A4 SPICE directive
.PARAM Wmin=1.5u
A5 SPICE directive
.PARAM g=2
P6 VDD
P7 GND
P5 XN
P2 D
P3 SN
P4 EN
M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18
clk180
clk360
Q
qn
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
LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
REVISION: $Revision$
PAGE 1 OF 1
DRAWN BY: <stdcelllib@nospam.chipforge.org>
Grid(100, 100)
RB: ON
MN: ON
Select Mode


xschem - LATESN.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
A1 SPICE model
Model name: nmos4
File: Technology/spice/ls1unmos.mod
A2 SPICE model
Model name: pmos4
File: Technology/spice/ls1upmos.mod
A3 SPICE .SUBCKT
LATESN
A4 SPICE directive
unknown
.PARAM Wmin=1.5u
A5 SPICE directive
unknown
.PARAM g=2
P6 P5 P2 P7 P3 P4
M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18
VDD
GND
clk180
clk360
Q
qn
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
LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
REVISION: $Revision$
PAGE 1 OF 1
DRAWN BY: <stdcelllib@nospam.chipforge.org>
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = -10 -210 - selected: 0 path: .


* untitled_1.sch - lepton-schematic
File Edit View Page Add Hierarchy Attributes Options Netlist Help
* untitled_1.sch
X?
D SN
Q
EN
XN
LATESN
Grid(100, 100)
RB: ON
MN: ON
Select Mode

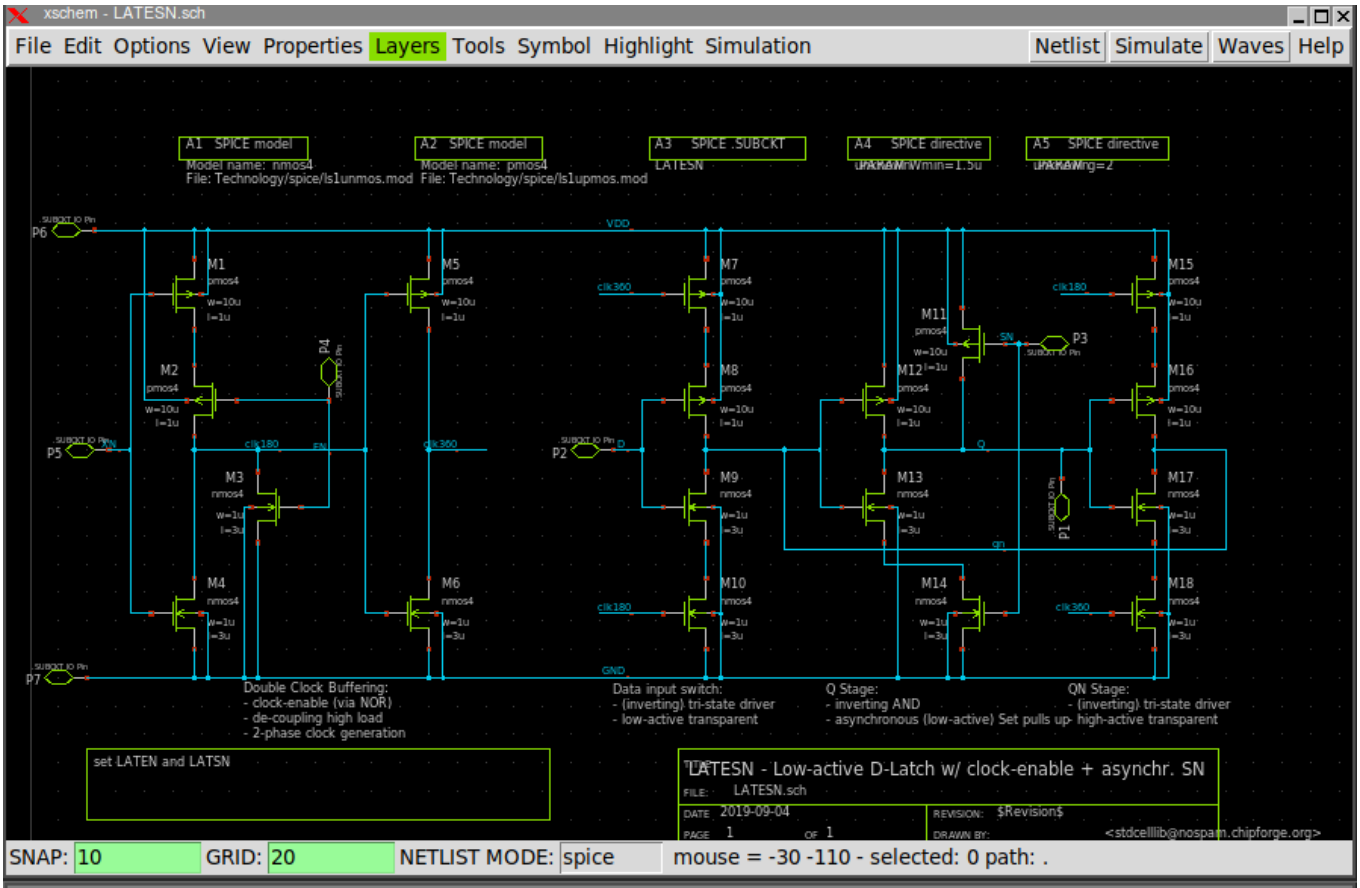

xschem - test.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
X?
D SN
Q
EN
XN
LATESN
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = 370 -910 - selected: 0 path: .

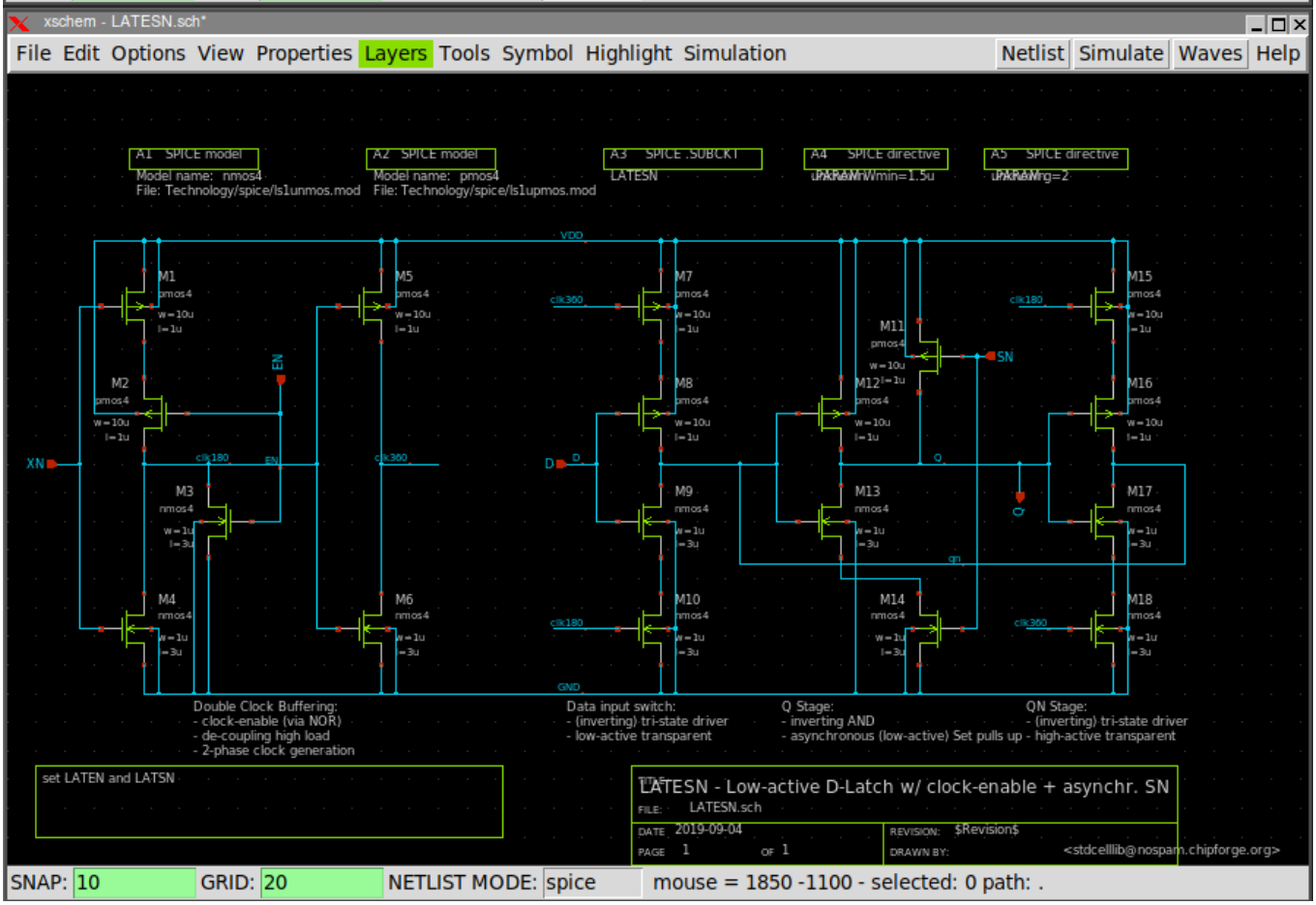
## Notes for schematics targeted for spice simulations

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.


xschem - LATESN.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
A1 SPICE model
Model name: nmos4
File: Technology/spice/ls1unmos.mod
A2 SPICE model
Model name: pmos4
File: Technology/spice/ls1upmos.mod
A3 SPICE .SUBCKT
LATESN
A4 SPICE directive
A5 SPICE directive
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
LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
REVISION: $Revision$
PAGE 1 OF 1
DRAWN BY: <stdcelllib@nospam.chipforge.org>
SNAP: 10 GRID: 20 NETLIST MODE: spice
mouse = -30 -110 - selected: 0 path: .


xschem - LATESN.sch*
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
A1 SPICE model
Model name: nmos4
File: Technology/spice/ls1unmos.mod
A2 SPICE model
Model name: pmos4
File: Technology/spice/ls1upmos.mod
A3 SPICE .SUBCKT
LATESN
A4 SPICE directive
A5 SPICE directive
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
LATESN - Low-active D-Latch w/ clock-enable + asynchr. SN
FILE: LATESN.sch
DATE 2019-09-04
REVISION: $Revision$
PAGE 1 OF 1
DRAWN BY: <stdcelllib@nospam.chipforge.org>
SNAP: 10 GRID: 20 NETLIST MODE: spice
mouse = 1850 -1100 - selected: 0 path: .

UP

# FAQ

## I want new instances to get assigned a new unique name automatically.

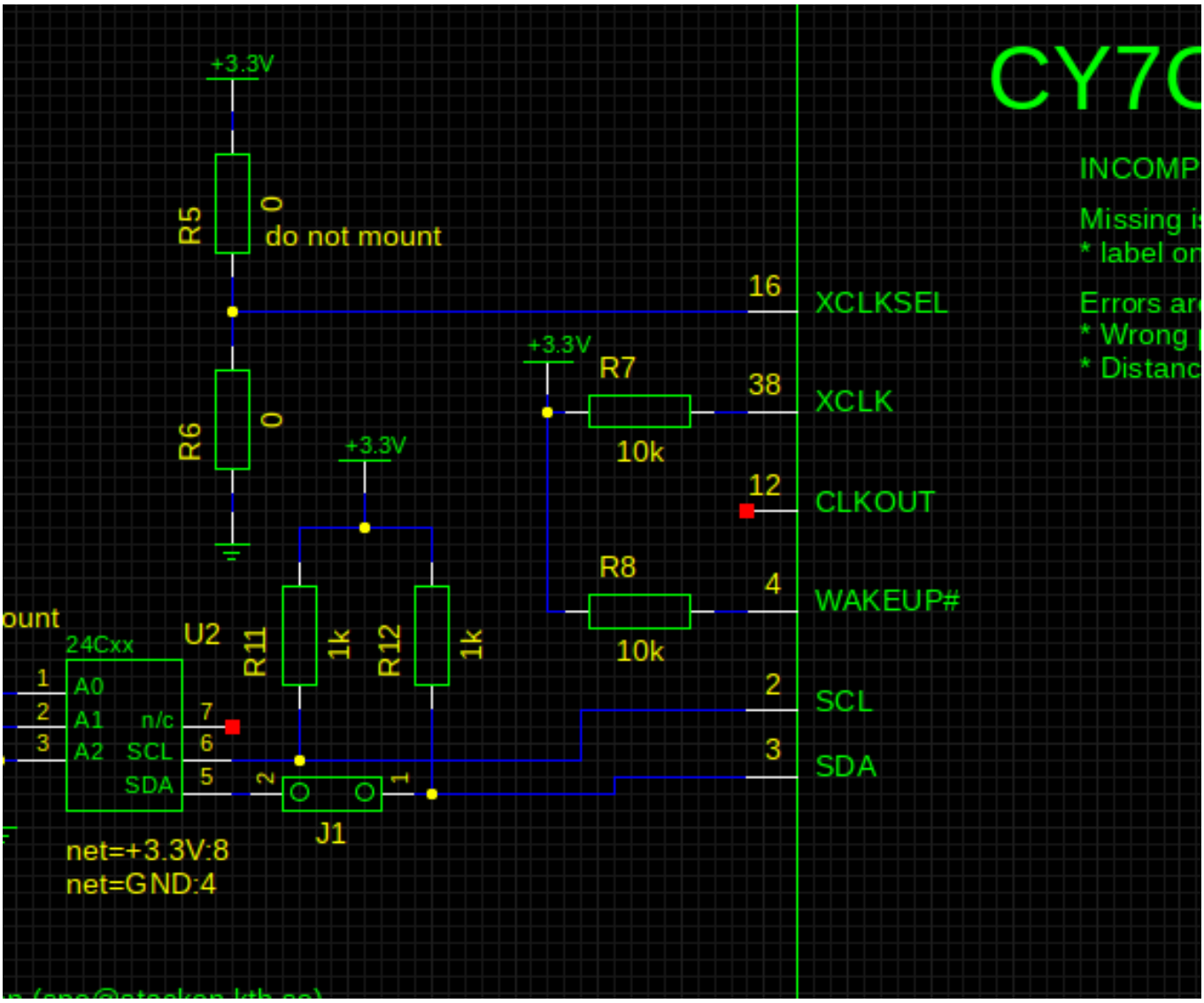
Add this to your xschemrc file:

```
set disable_unique_names 0
```

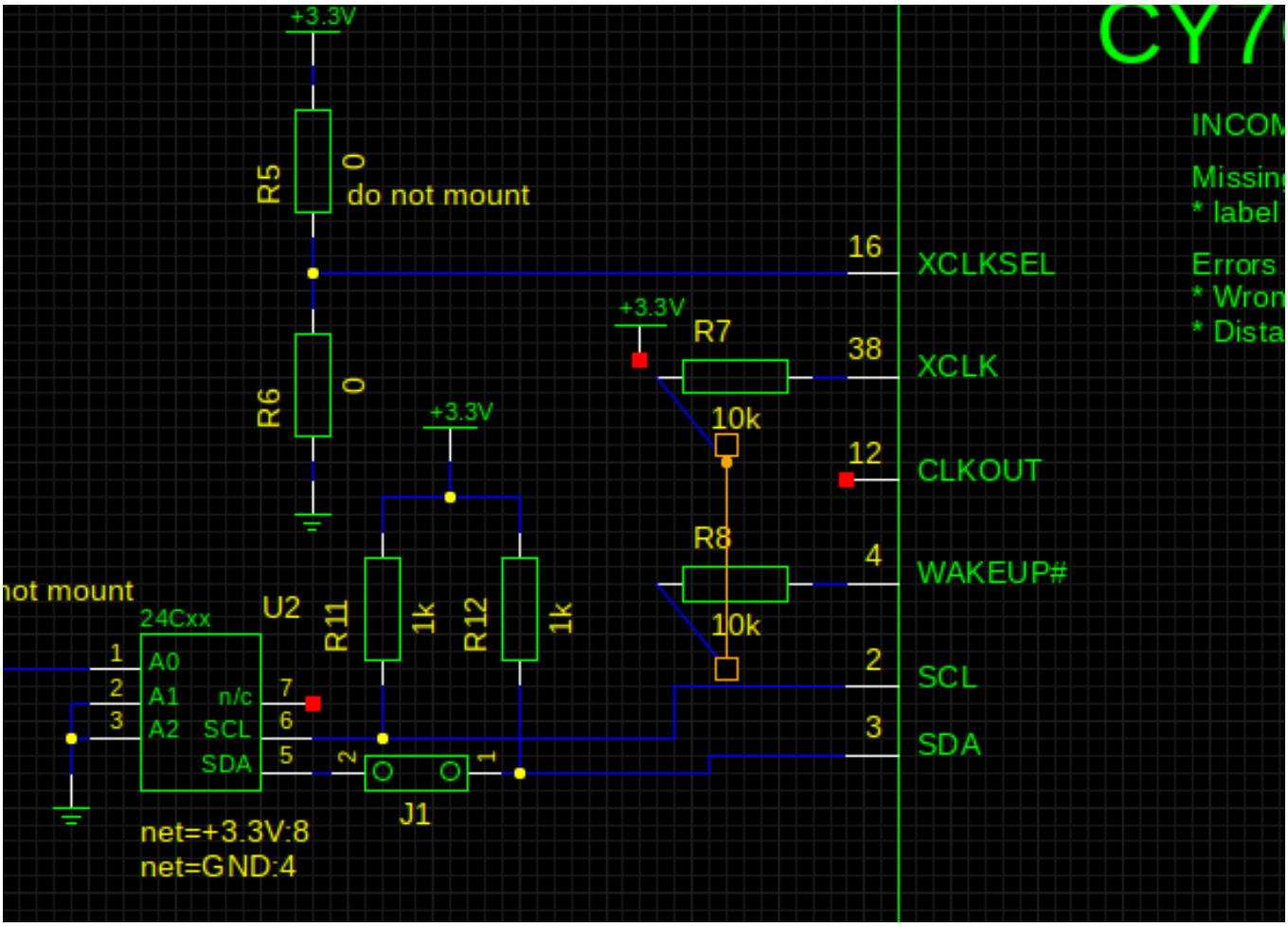
By default XSCHEM allows instance name (Refdes) duplicates in the schematic. This must be resolved by the user normally, before exporting any netlist. The **`Hilight - Highlight duplicate instance names`** (**`k`** key) menu entry can be used to mark the components that need to be renamed. The **`Highlight - 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:




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!!