XSCHEM : schematic capture and netlisting EDA tool

Xschem is a schematic capture program, it allows creation of hierarchical representation of circuits with a top down approach . By focusing on interfaces, hierarchy and instance properties a complex system can be described in terms of simpler building blocks. A VHDL or Verilog or Spice netlist can be generated from the drawn schematic, allowing the simulation of the circuit. Key feature of the program is its drawing engine written in C and using directly the Xlib drawing primitives; this gives very good speed performance, even on very big circuits. The user interface is built with the Tcl-Tk toolkit, tcl is also the extension language used.

Features

- hierarchical schematic drawings, no limits on size
- any object in the schematic can have any sort of properties (generics in VHDL, parameters in Spice or Verilog)
- new Spice/Verilog primitives can be created, and the netlist format can be defined by the user
- tcl extension language allows the creation of scripts; any user command in the drawing window has an associated tcl comand
- VHDL / Verilog / Spice netlist, ready for simulation
- Behavioral VHDL / Verilog code can be embedded as one of the properties of the schematic block,

Xschem runs on UNIX systems with X11 and Tcl-Tk toolkit installed.

Documentation

XSCHEM manual

Download

Current release
Old XSCHEM releases on Sourceforge
SVN: svn checkout svn://repo.hu/xschem/trunk

License

Contact

Anyone interested in this project please contact me at the following address:

STEFAN.SCHIPPERS@GMAIL.COM

Software requirements:

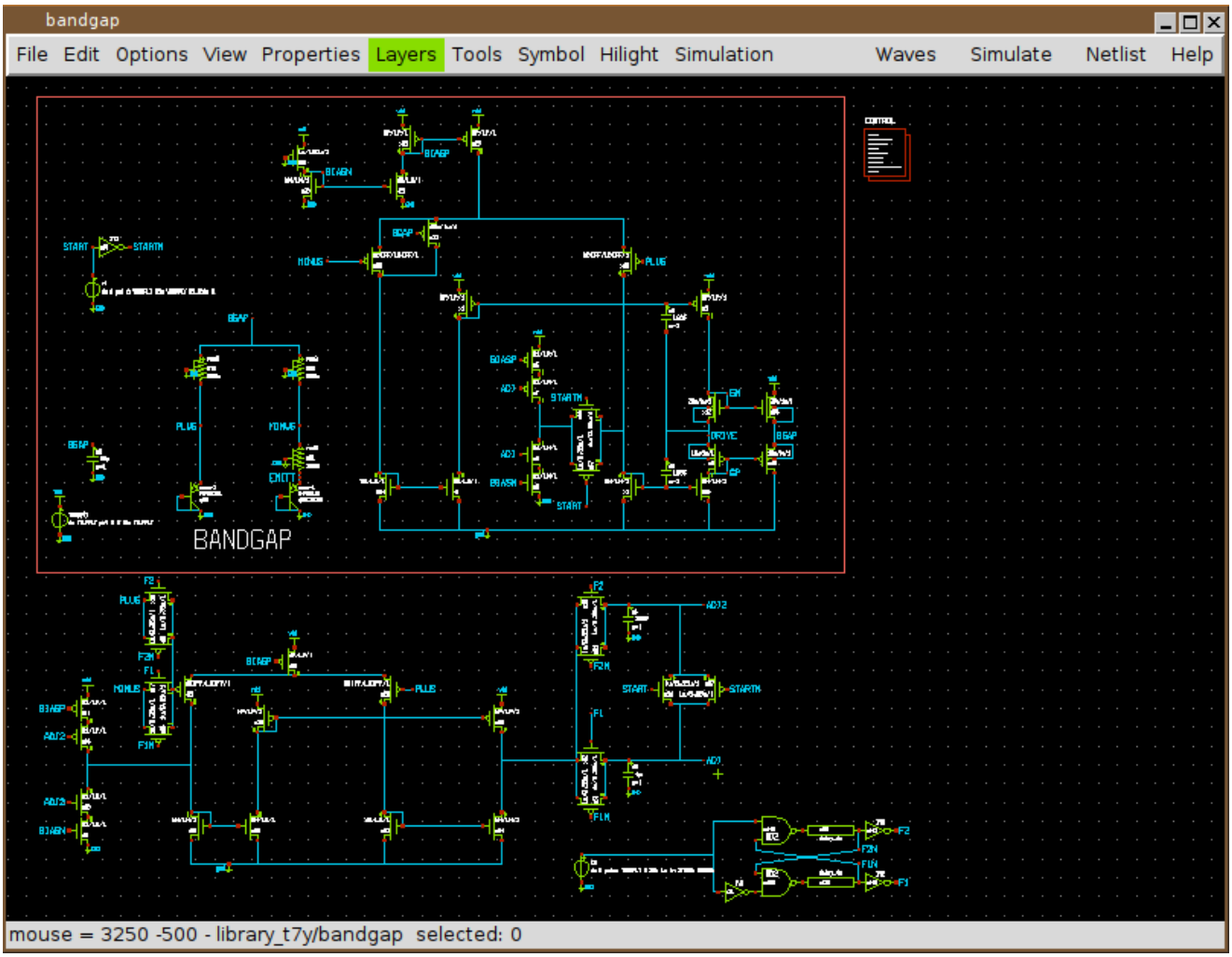
- X11
- tcl-tk libs and developent files
- c99 compiler
- bison (only for compiling the grammar parser)
- flex (only for compiling the lexical analyzer
- Xpm library and -dev header files
- awk (tested with gawk and mawk)

Systems tested:

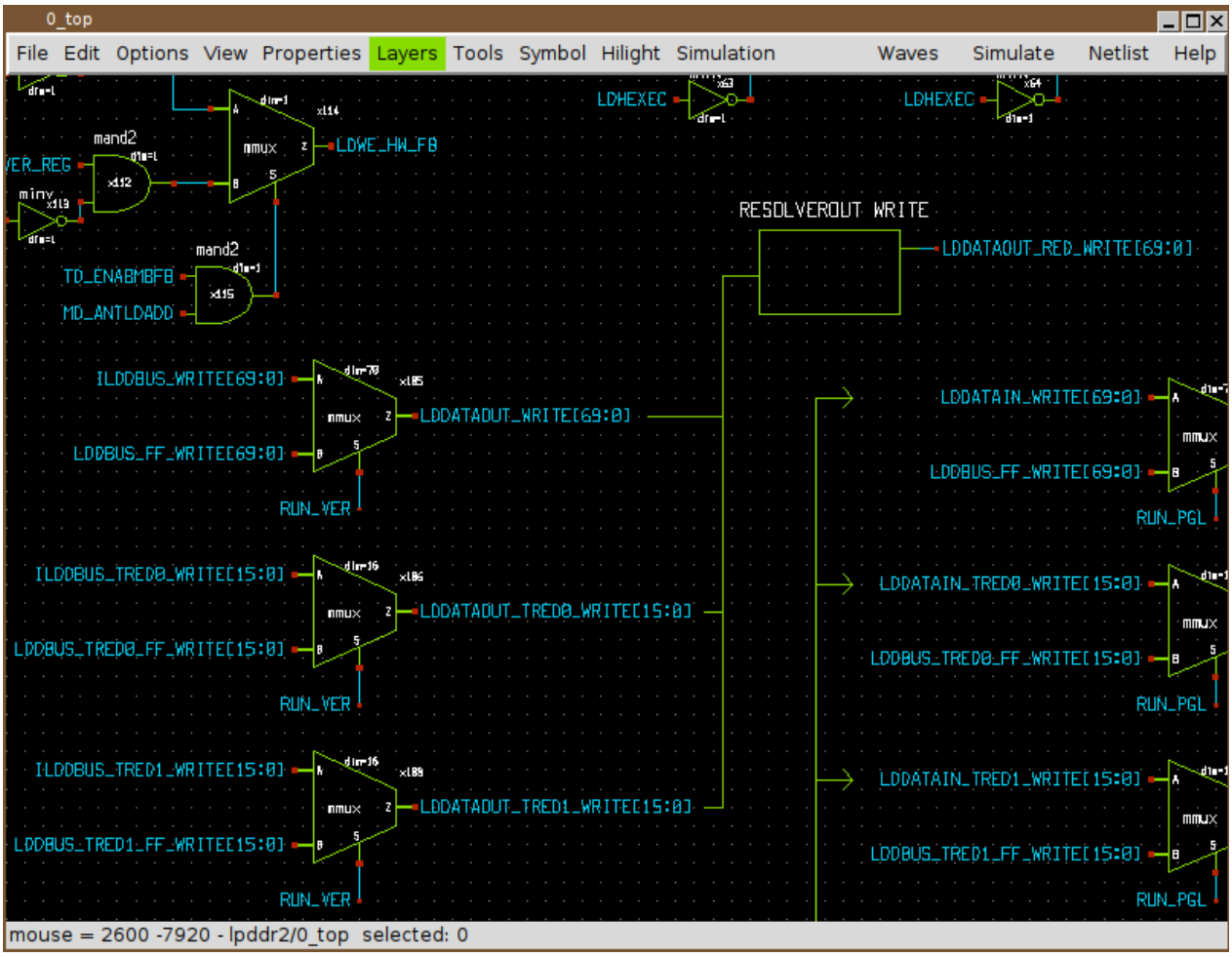
- Linux debian / Redhat
- Solaris sparc
- Windows (with the cygwin layer and cygwin/Xorg X11 server, plus the tcl/tk toolkit and the -dev libraries)

Screenshots

- analog circuit example



- digital system for VHDL simulation


0_top
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
LDHEXEC
LDHEXEC
mand2
LDWE_HW_FB
mand2
RESDLVEROUT_WRITE
LDDATAOUT_RED_WRITE[69:0]
TD_ENABMBFB
MD_ANTLDADD
ILDDBUS_WRITE[69:0]
LDDATADUT_WRITE[69:0]
LDDBUS_FF_WRITE[69:0]
RUN_VER
LDDATAIN_WRITE[69:0]
LDDBUS_FF_WRITE[69:0]
RUN_PGL
ILDDBUS_TRED0_WRITE[15:0]
LDDATADUT_TRED0_WRITE[15:0]
LDDBUS_TRED0_FF_WRITE[15:0]
RUN_VER
LDDATAIN_TRED0_WRITE[15:0]
LDDBUS_TRED0_FF_WRITE[15:0]
RUN_PGL
ILDDBUS_TRED1_WRITE[15:0]
LDDATADUT_TRED1_WRITE[15:0]
LDDBUS_TRED1_FF_WRITE[15:0]
RUN_VER
LDDATAIN_TRED1_WRITE[15:0]
LDDBUS_TRED1_FF_WRITE[15:0]
RUN_PGL
mouse = 2600 -7920 - lpddr2/0_top selected: 0

Contents

INDEX

TUTORIALS

FAQ

WHAT IS XSCHEM

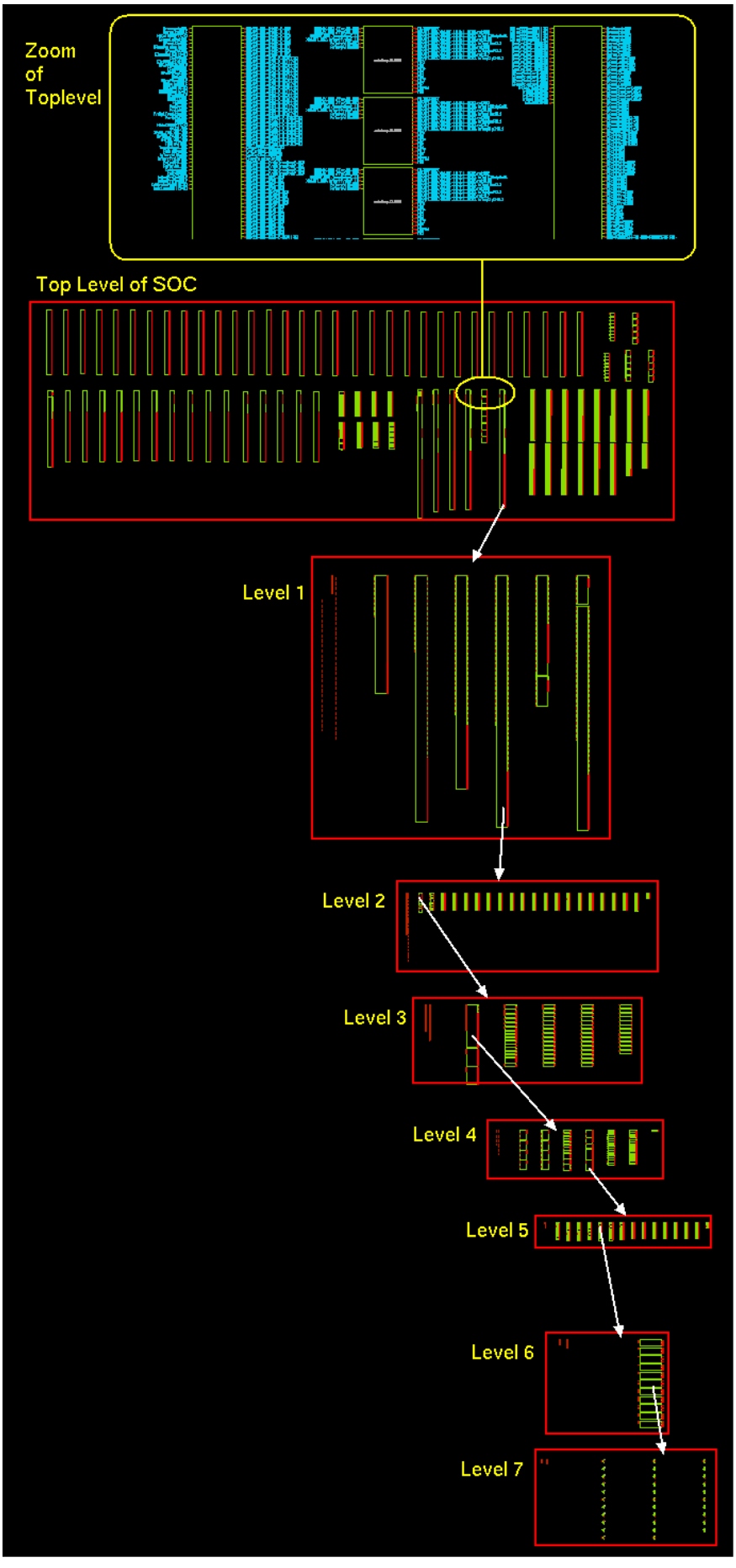
Electronic systems today tend to be generally very complex and a lot of work has to be done from circuit conception to the validation of the final product. One of the milestones of this process is the creation of the circuit schematic of the electronic system.

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
Level 4
Level 5
Level 6
Level 7

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.

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. 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
```

Detailed XSCHEM startup sequence

1. 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 `` `pwd` `` and also set `XSCHEM_LIBRARY_PATH` to `` `pwd`/../xschem_library ``.
2. else use compile-time (generated from configure script) provided `XSCHEM_SHAREDIR`.
3. if in current dir there is a `xschemrc` file source it.
4. else if there is a `USER_CONF_DIR/xschemrc` file source it.
5. else if there is a `XSCHEM_SHAREDIR/xschemrc` file then 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 point 1.
6. if `XSCHEM_SHAREDIR` not defined --> error and quit.
7. source `$XSCHEM_SHAREDIR/xschem.tcl`.
8. start loading user provided schematic file or start with empty window (or filename specified in `XSCHEM_START_WINDOW` tcl variable).

RUN XSCHEM

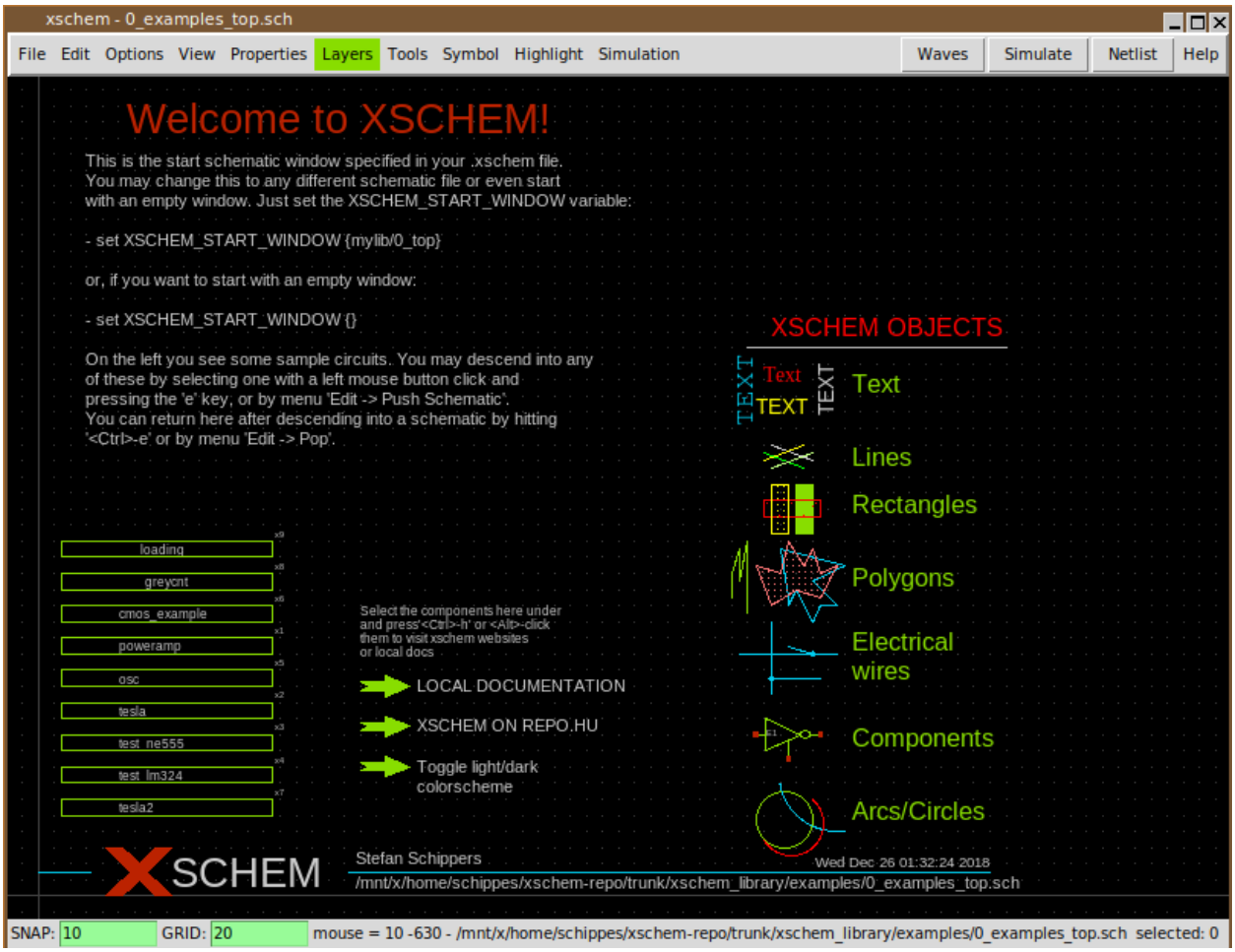
Assuming `xschem` is installed in one of the `${PATH}` search paths just execute:

```
user:~$ xschem
```

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

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
  -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
  -i  --no_rcload       do not load any xschemrc file
  -o  --netlist_path    set output for netlist
  -t  --tedax           set netlist type to tEDAx
  -s  --spice           set netlist type to SPICE
  -3  --a3page          set page size for pdf export to A3
  -x  --no_x            dont 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
  --rcfile <file>       use <file> as a rc file for startup instead of the
                        default xschemrc.
  -p  --postscript      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

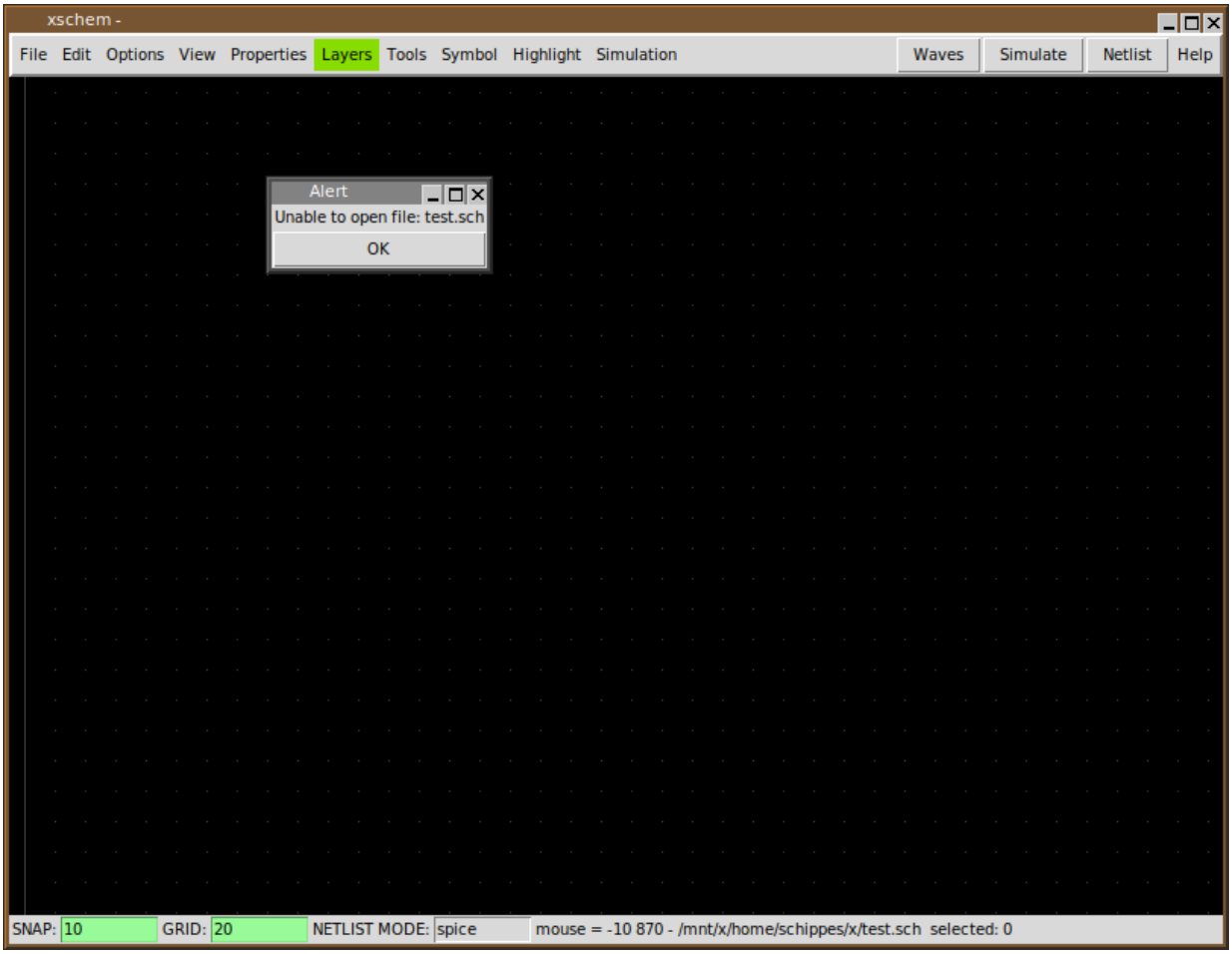
xschem: interactive schematic capture program

Example:   xschem counter.sch
the schematic file `counter' will be loaded searching in the
list of library paths provided by TCL variable XSCHEM_LIBRARY_PATH
This variable is set in user or system xschem startup file
(<install root>/share/xschem/xschemrc or ~/.xschem/xschemrc)
```

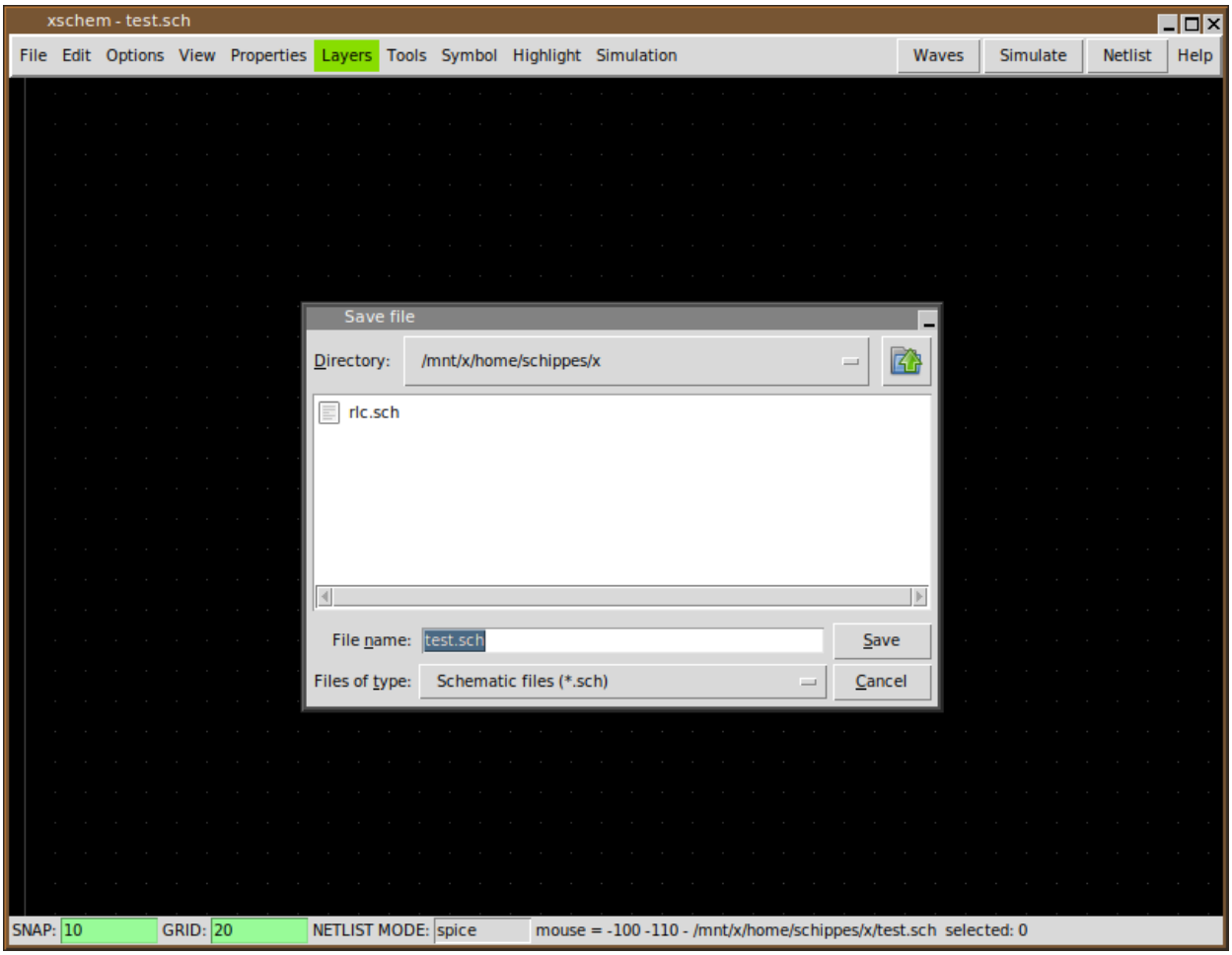
CREATING A NEW SCHEMATIC

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.

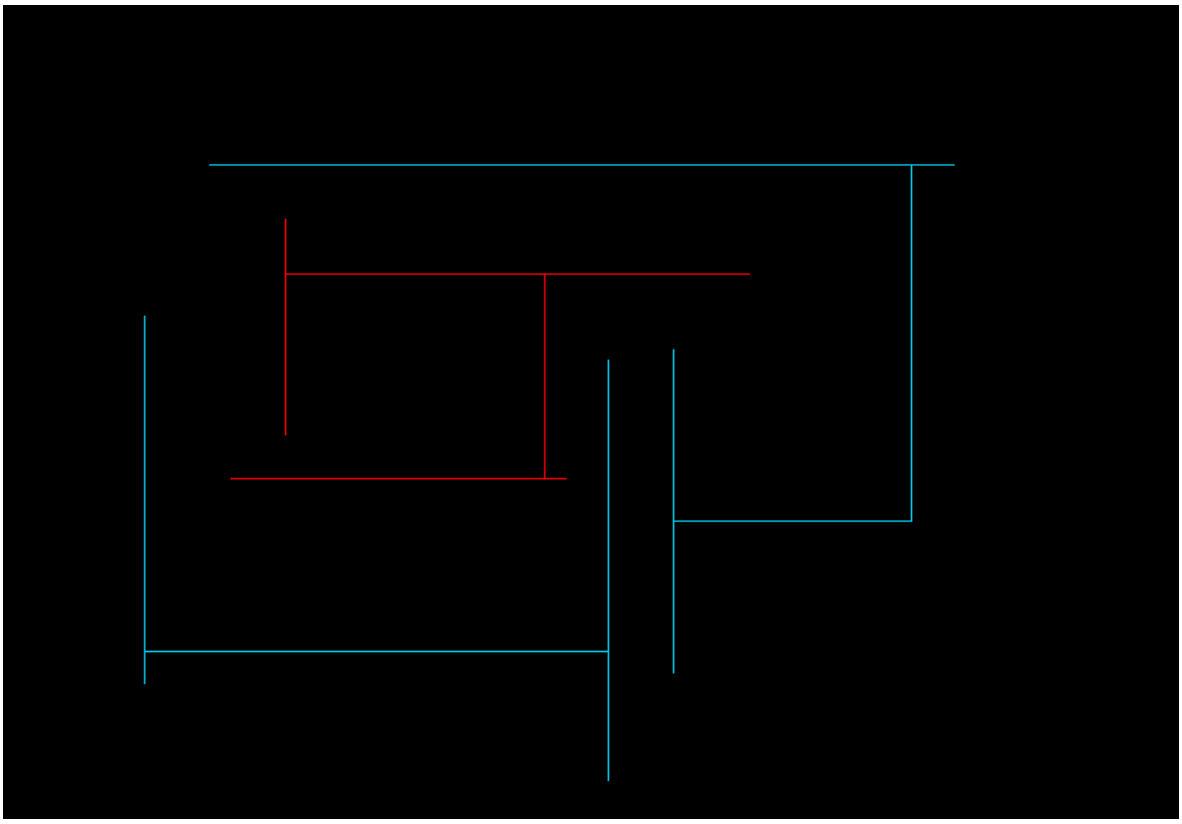
```
# xschem defaults its search to the
# XSCHEM_LIBRARY_PATH root path
# .sch may be omitted, it is added by default ...
xschem test
# or ...
xschem /home/schippes/x/test.sch
# or ...
xschem ${HOME}/schippes/x/test
```

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

XSCHEM ELEMENTS

WIRES

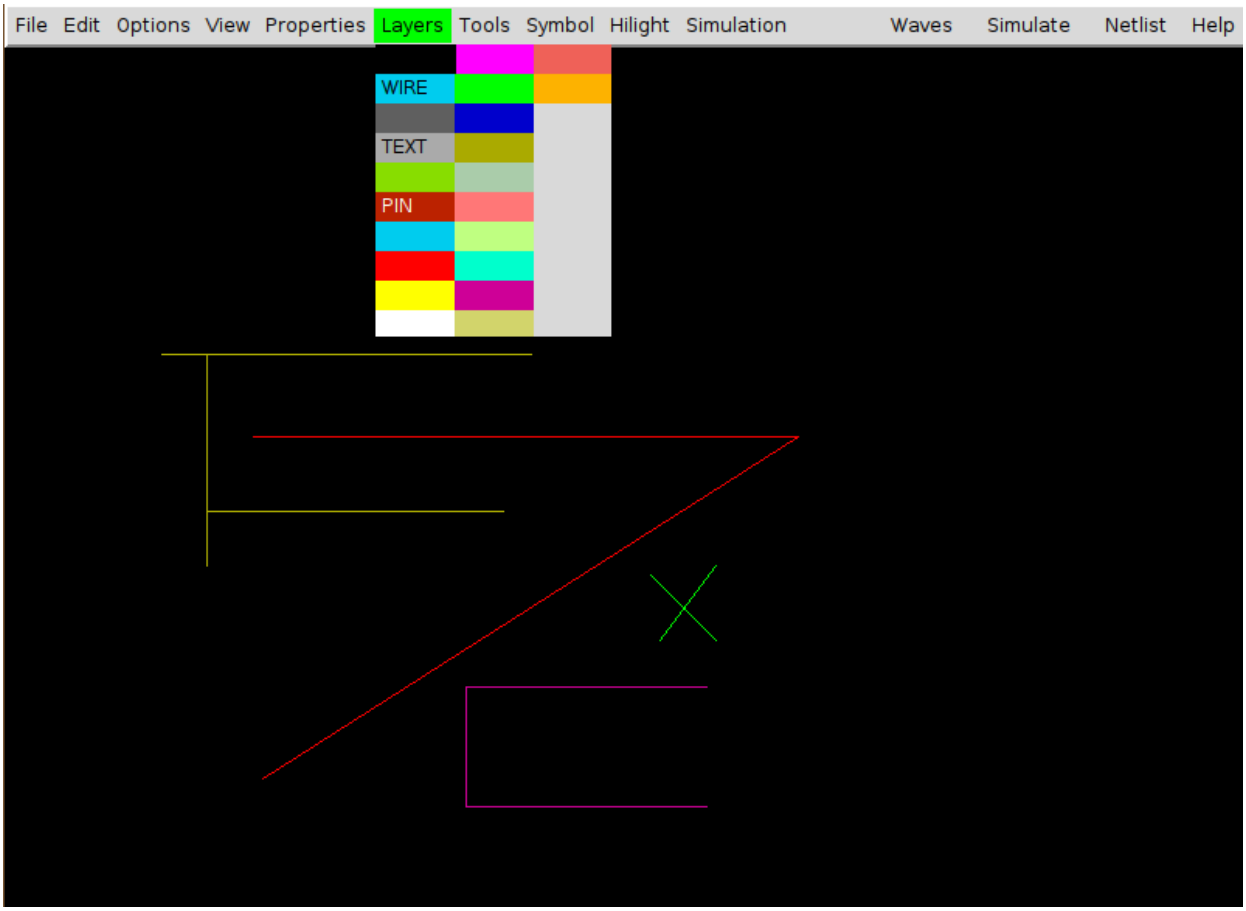
Wires in XSCHEM are the equivalent of copper traces in printed circuit boards or electrical conductors. Wires are drawn as lines but the electrical connectivity graph is built by XSCHEM. To draw a wire segment point the mouse somewhere in the drawing window and press the `'w'` key. A rubber wire is shown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



LINES

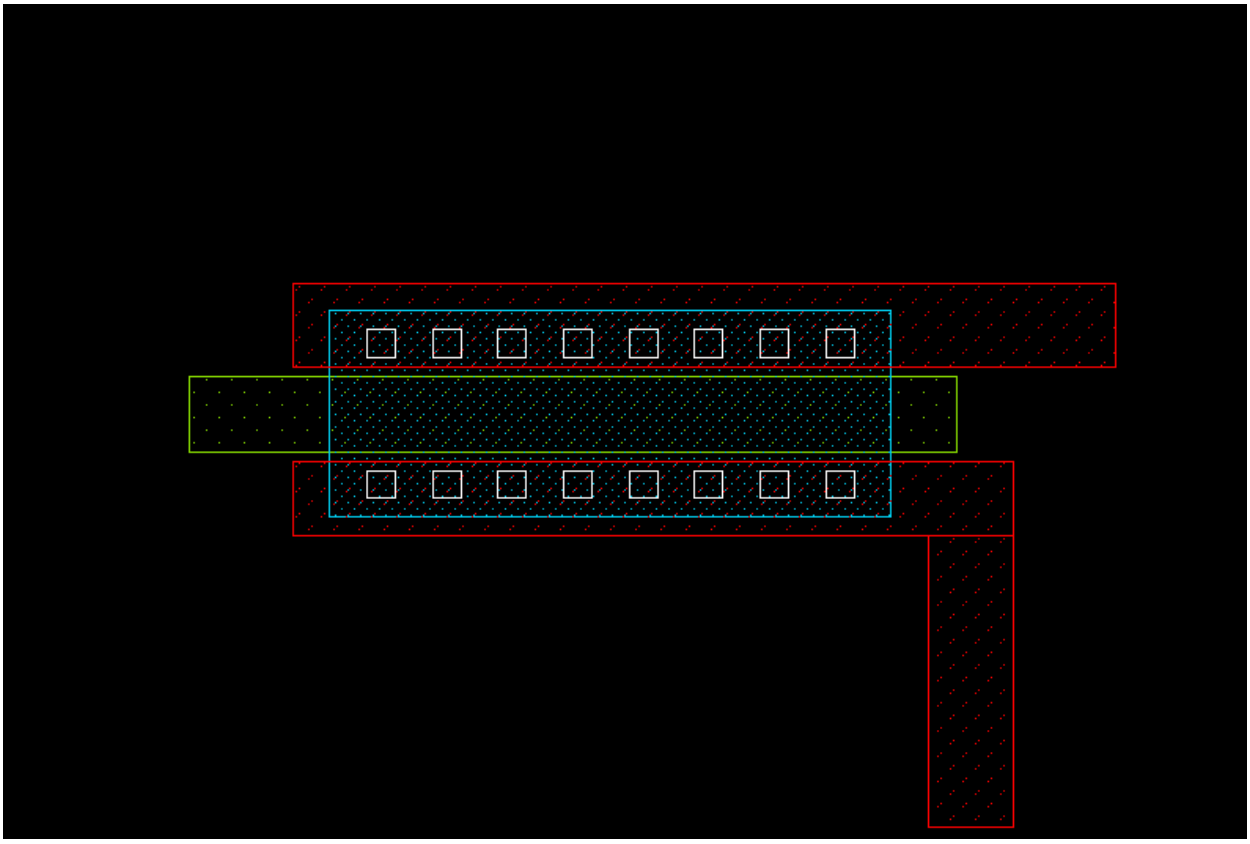
Lines are just segments that are used for drawing. Lines do not have any electrical meaning, in fact when building the circuit netlist, lines are completely ignored. XSCHEM uses different layers to draw lines. Each layer has its own color, allowing to draw with different colors. Lines

are placed like wires, but using the `'l'` key. The 'Layers' menu allows to select various different layers (colors) for the line.



RECTANGLES

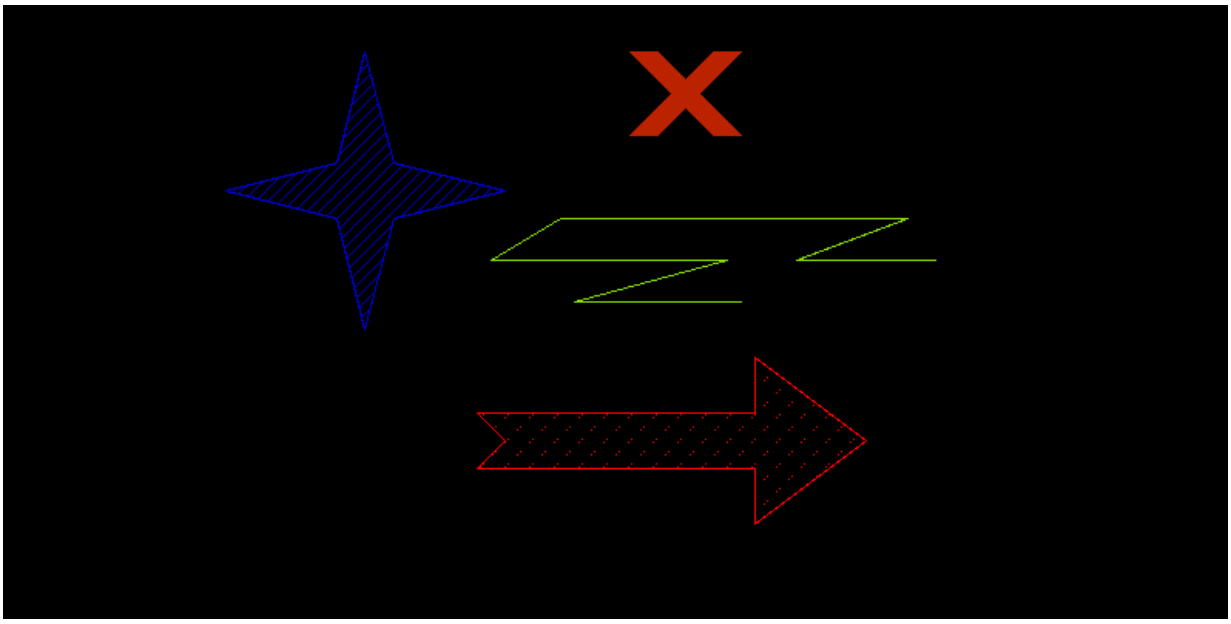
Rectangles like Lines are drawable on multiple layers, and also do not carry any electrical information. A specific 'PIN' layer is used to make pins that are used to interconnect wires and components. Different fill styles (or no fill) can be defined for each layer. Rectangles are placed with the `'r'`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



POLYGONS

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

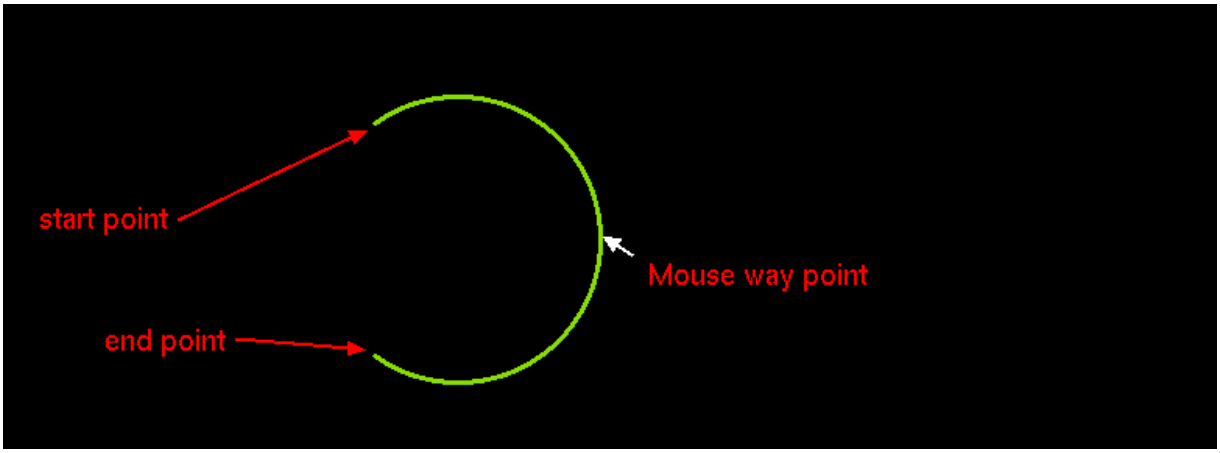
- the last point is coincident to the first point.
- or by clicking the `right mouse button`, for an open polygon.
- or by hitting the `Return` key, for a closed polygon (this can be done also by clicking the last point coincident to the first polygon point).



CIRCLES / ARCS

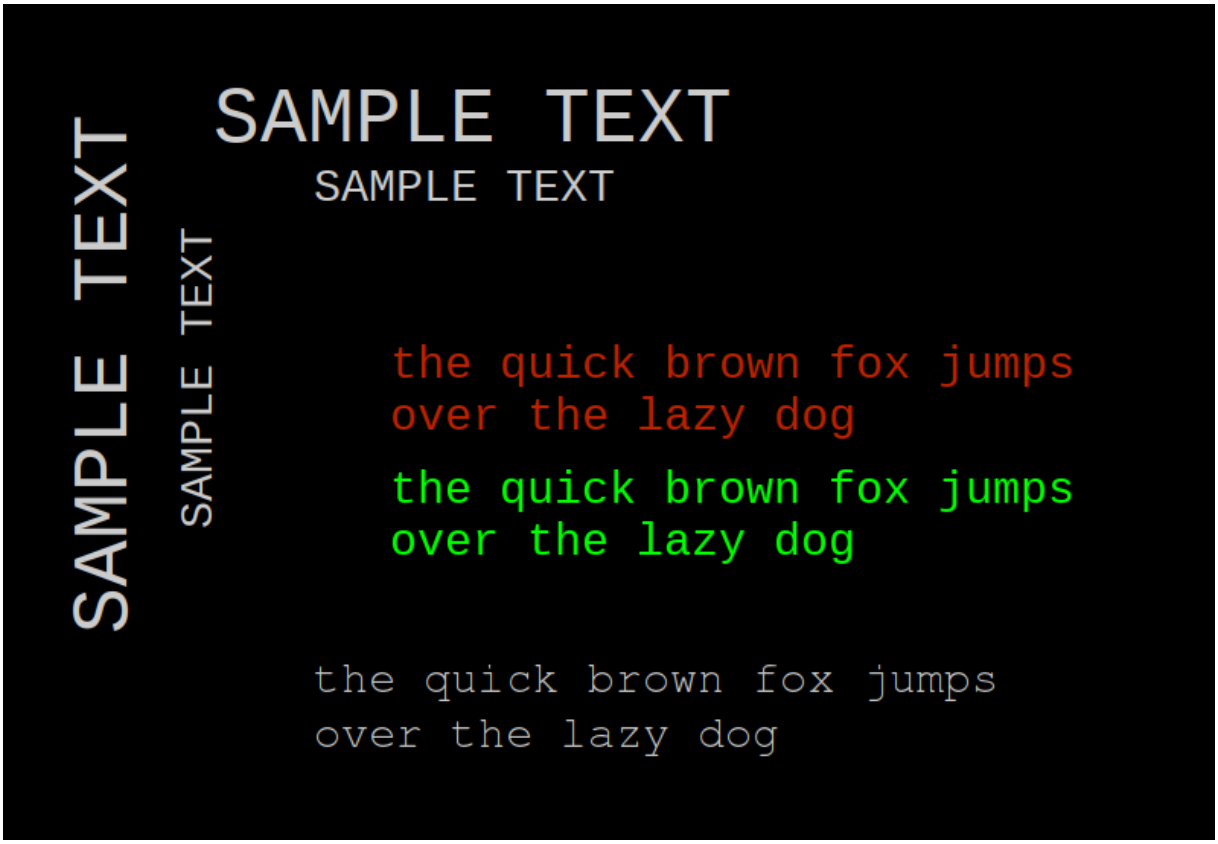
Arcs may be placed by hitting the `Shift-C` key. First click the start point, then the end point. Moving the mouse will show the arc passing thru the 2 points and the mouse waypoint. Clicking will place the arc. Arcs may be modified after creation by selecting in stretch mode (`Ctrl-Button1-drag`) one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

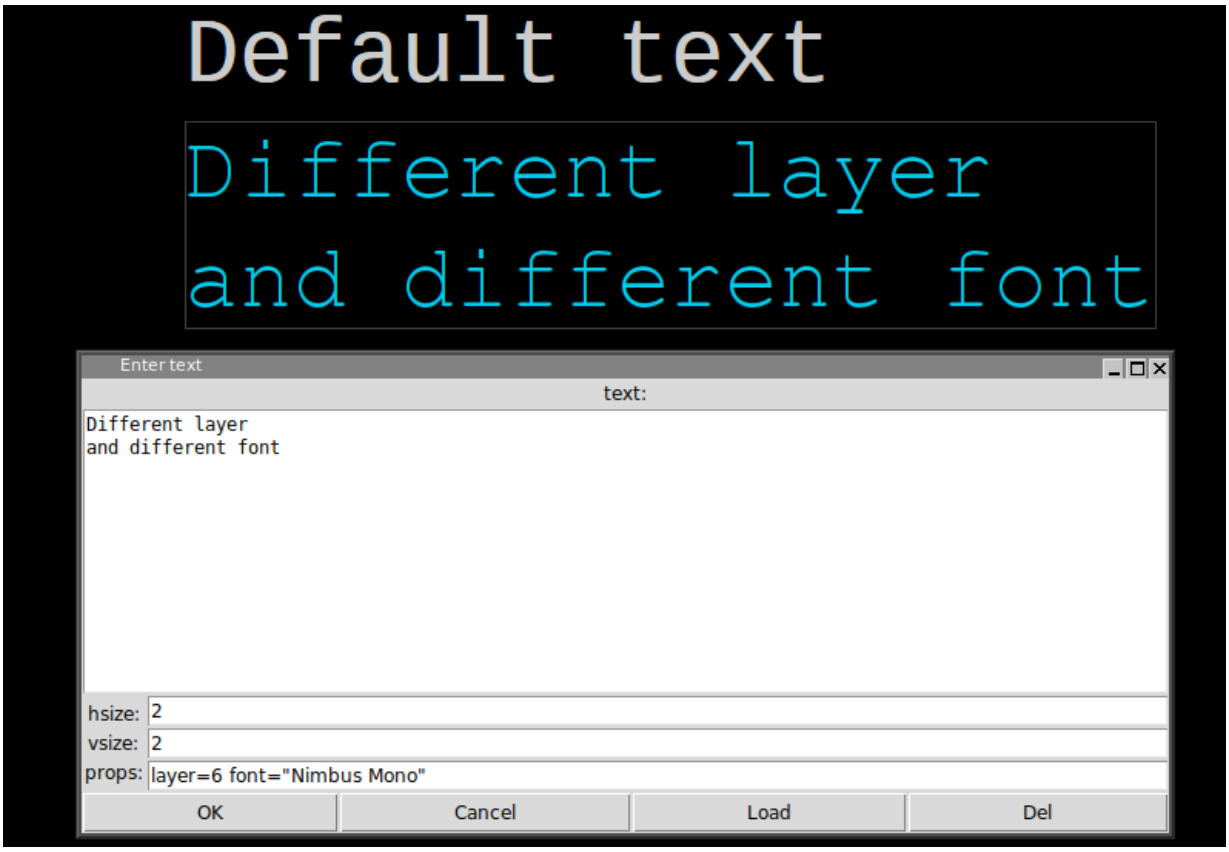


TEXT

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

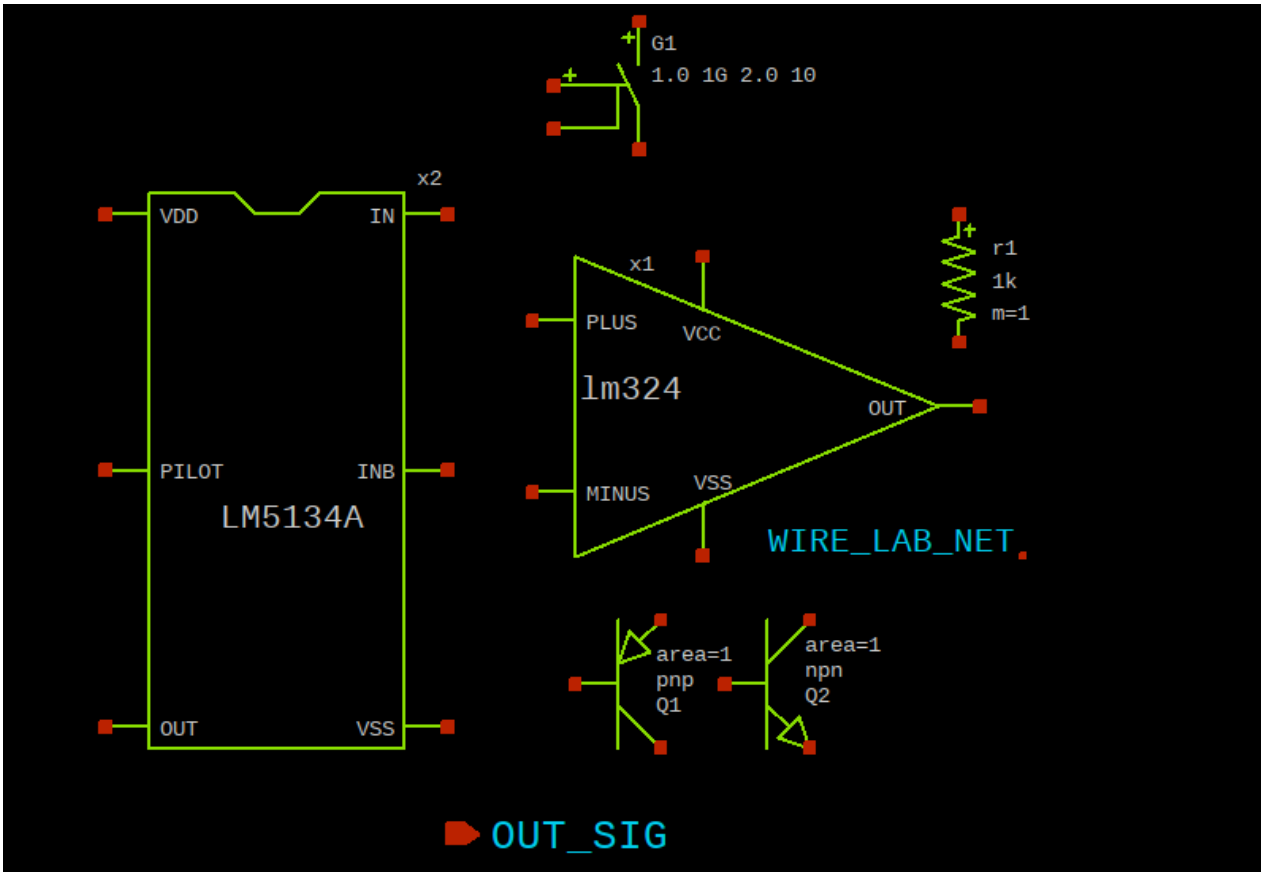


The `layer` property can be used to draw text on a different layer, for example, setting `layer=6` will draw on cyan color. A `font` property is defined to change the default font. Use only Monospaced fonts because bounding box is not correctly calculated by XSCHEM for proportional typefaces. You will learn in the xschem properties chapter how to set, edit and change object properties.


Default text
Different layer
and different font
Enter text
text:
Different layer
and different font
hsize: 2
vsize: 2
props: layer=6 font="Nimbus Mono"
OK
Cancel
Load
Del

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.



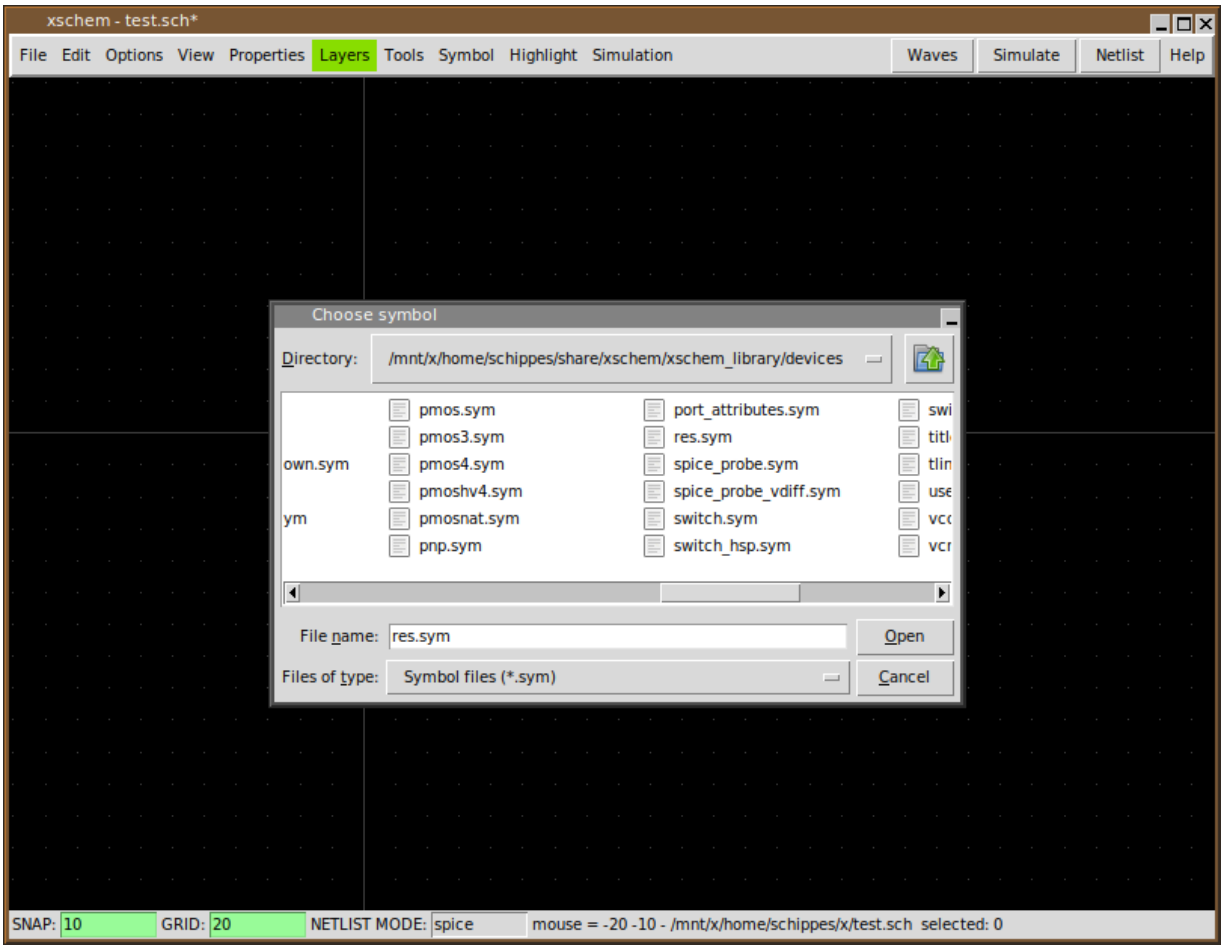
Symbols (like schematic drawings) are stored in xschem libraries. For XSCHEM a library is just a directory placed under the XSCHEM_LIBRARY_PATH directory, see the installation slide. A symbol is stored in a .sym file.

```
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
```

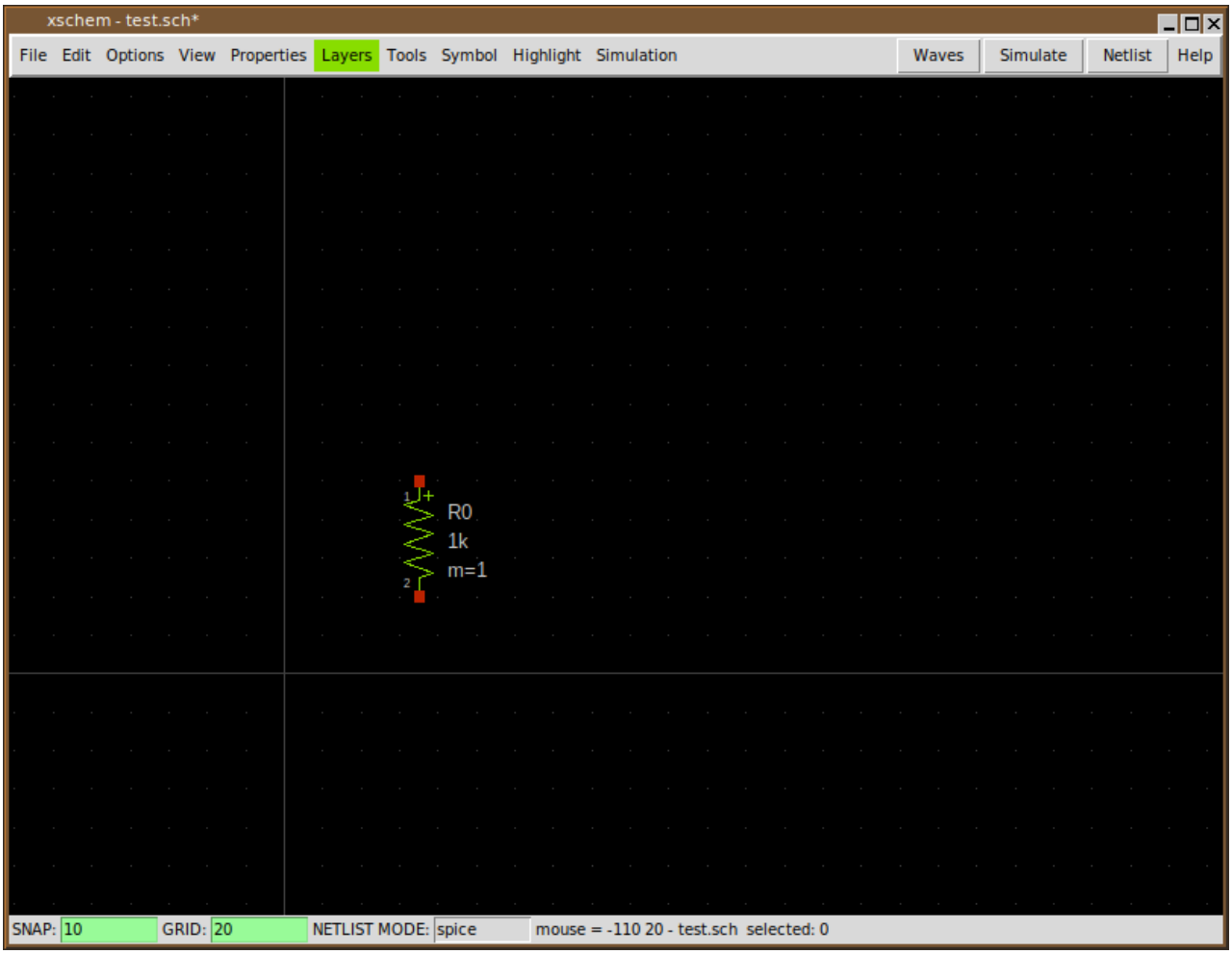
```
arch_declarations.sym      gnd.sym                npn.sym
switch.sym
architecture.sym           ind.sym                opin.sym
title.sym
assign.sym                 iopin.sym              package_not_shown.sym
tline_hsp.sym
attributes.sym             ipin.sym               package.sym
use.sym
bus_connect_not_shown.sym  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.

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.

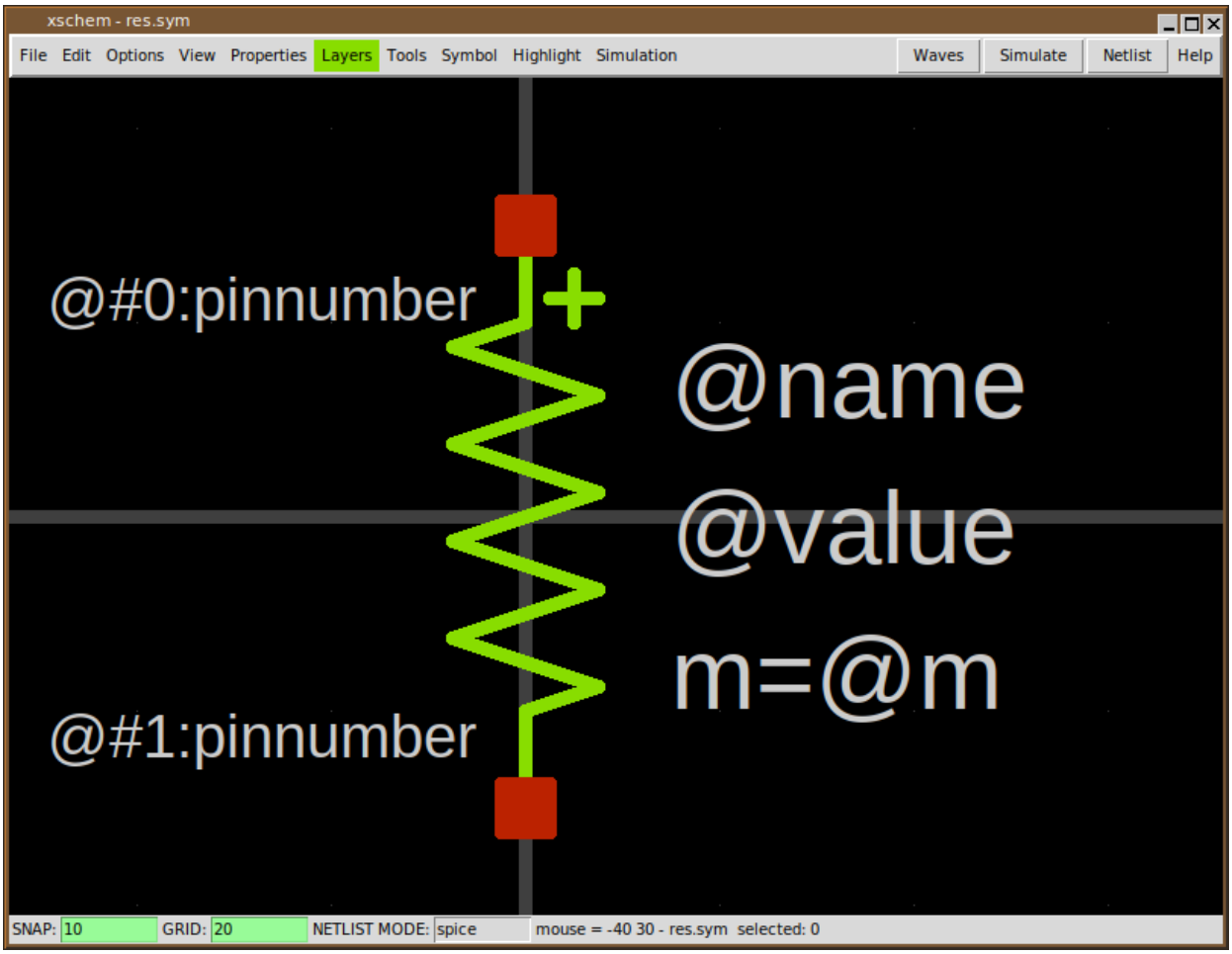


Use the file selector dialog to locate `res.sym`.



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.

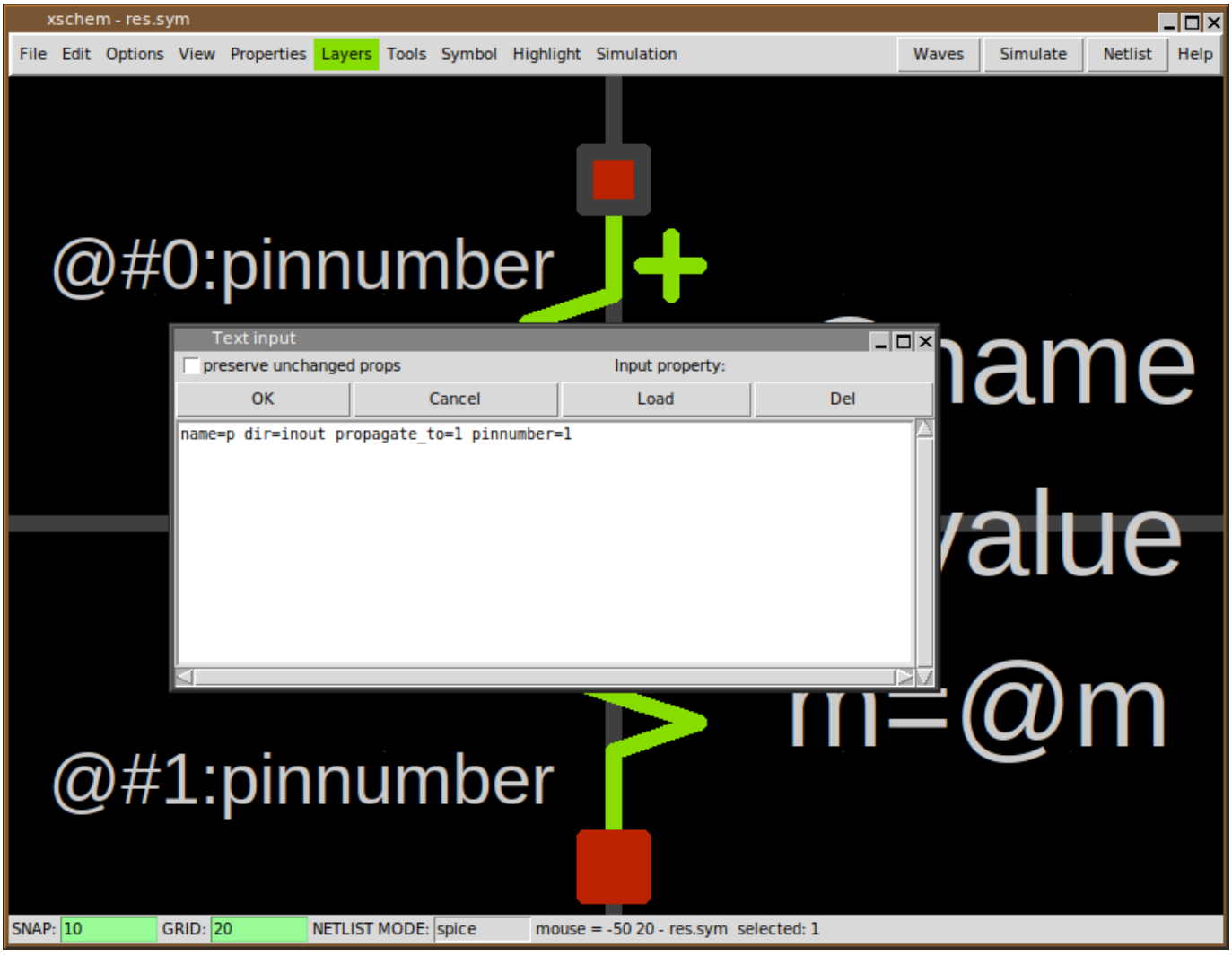
</body> </html>

XSCHEM PROPERTIES

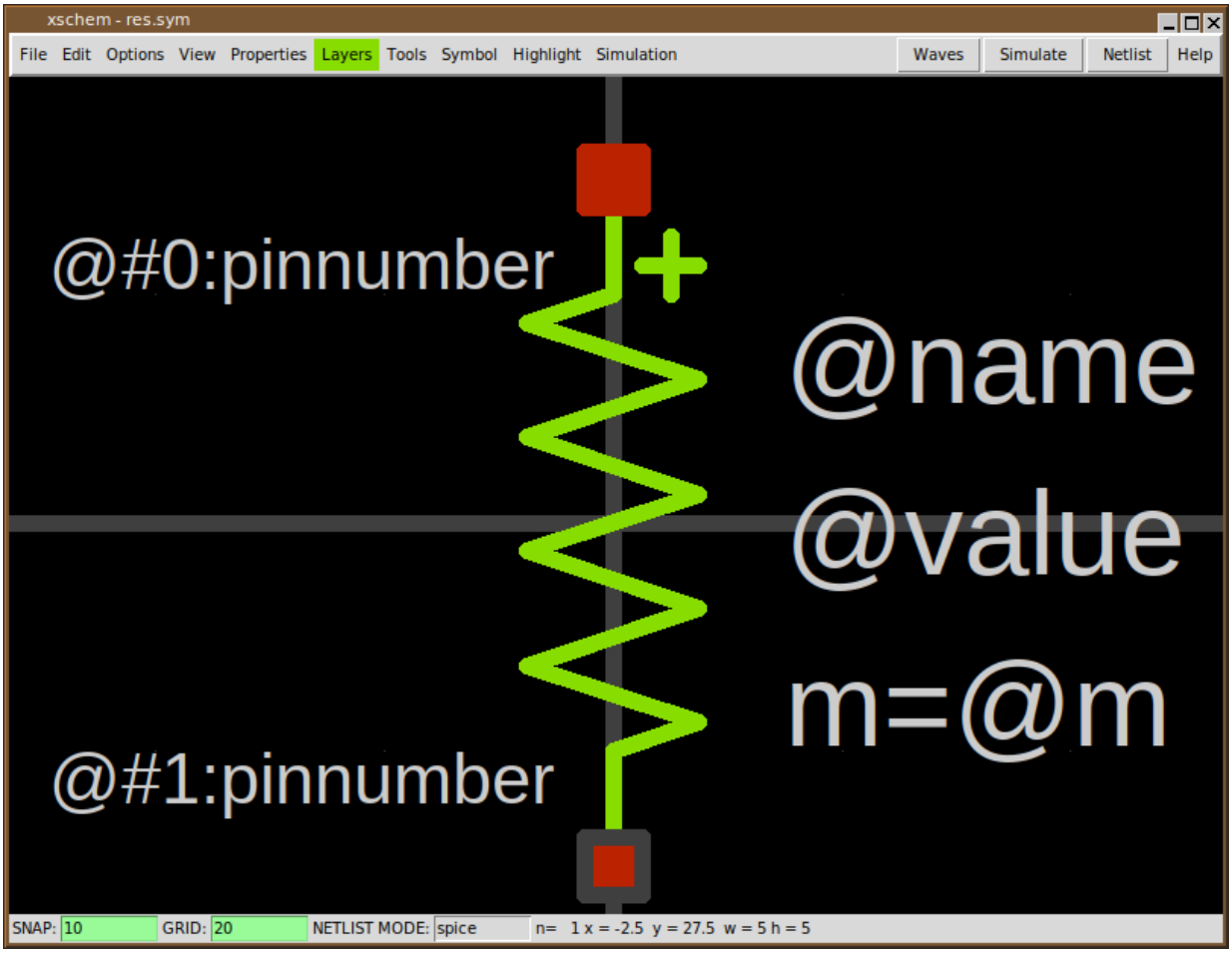
Properties are text strings that are associated to XSCHEM objects. All graphic primitives support properties.

- Wires
- Lines
- Rectangles
- Texts
- Symbols

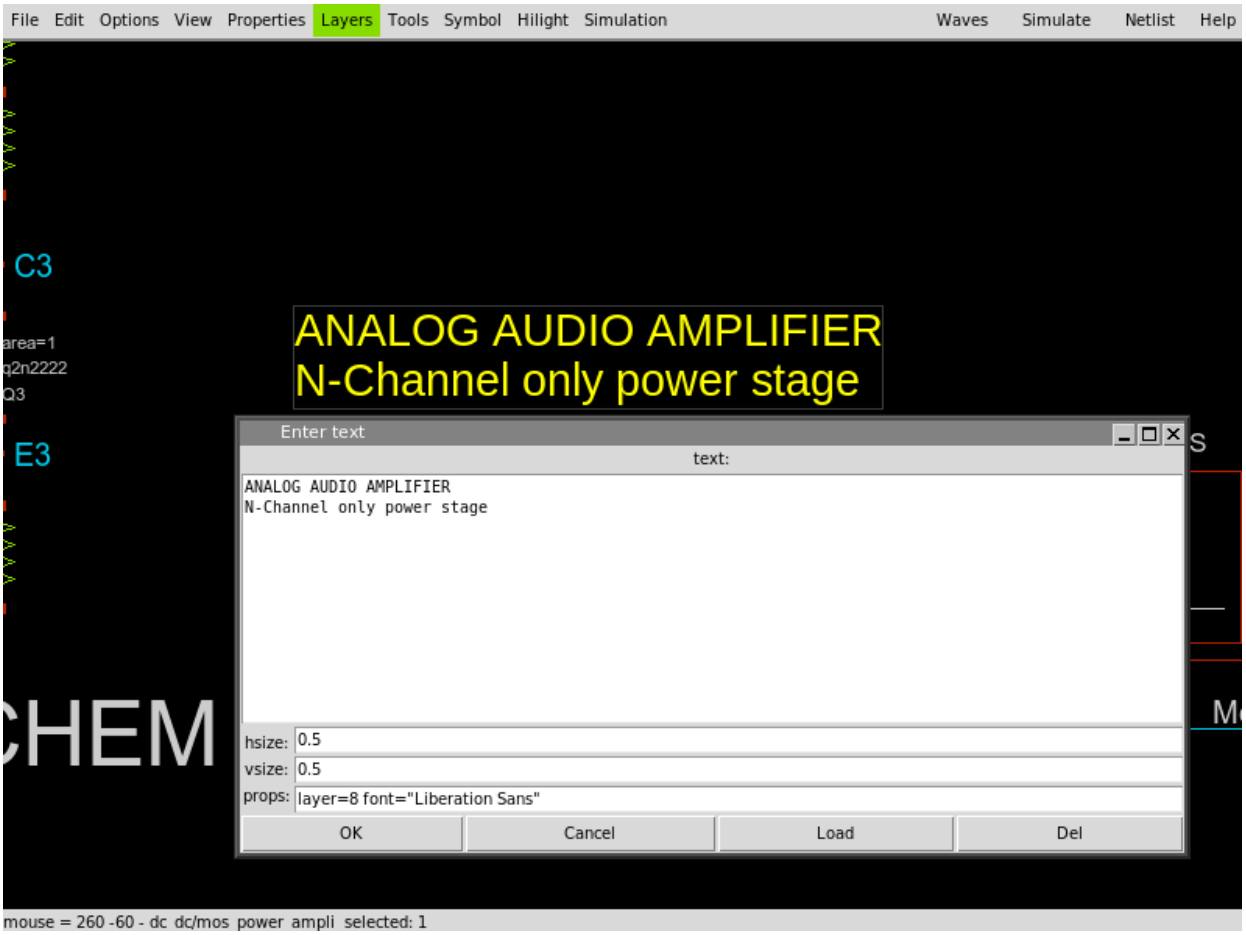
Consider for example the `res.sym` symbol (you may open it with the `File->Open` menu item) if you click inside one of the red pins and press the 'edit property' bindkey `'q'` a dialog box shows the property string associated with the selected pin:



The `name=p dir=inout propagate_to=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 `propagate_to=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 propagate_to=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.

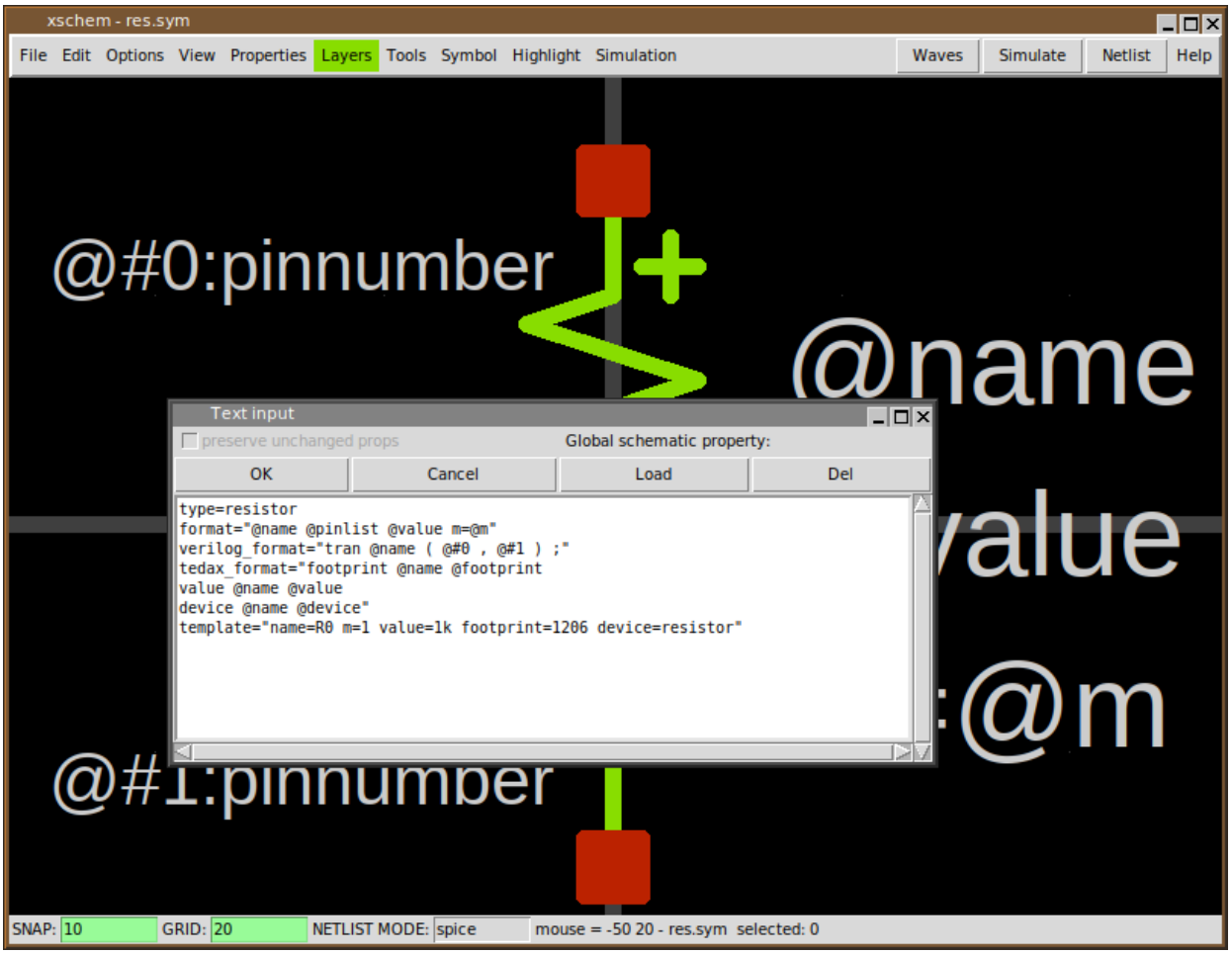


GLOBAL PROPERTIES

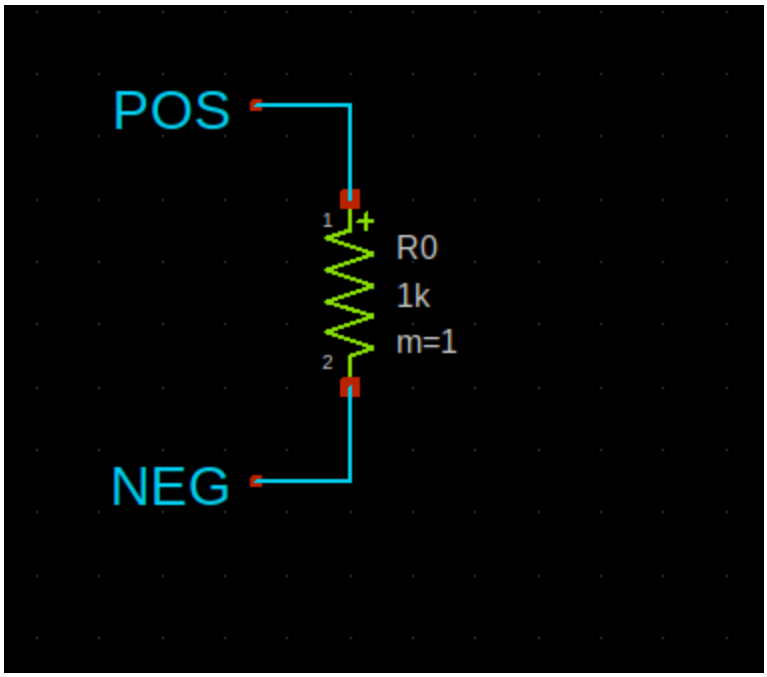
If you click outside of any displayed graphics in XSCHEM the selection set will be cleared. Clicking the edit property `'q'` key when nothing is selected will display the global property string of the schematic (.sch) or symbol window (.sym).

There is actually one different global property string defined for any available netlisting modes, so if XSCHEM is set to produce SPICE netlists the SPICE global property string is displayed.

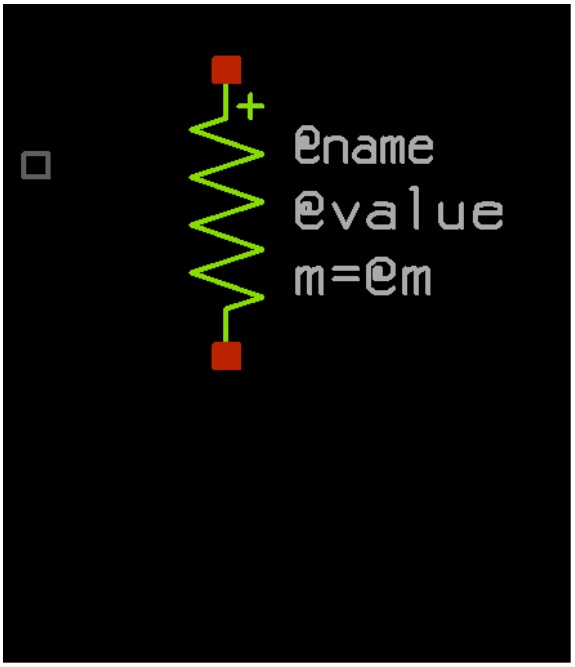
So, in addition to properties associated to graphical objects and symbols, we also have properties associated to schematic (.sch) and symbol files (.sym)

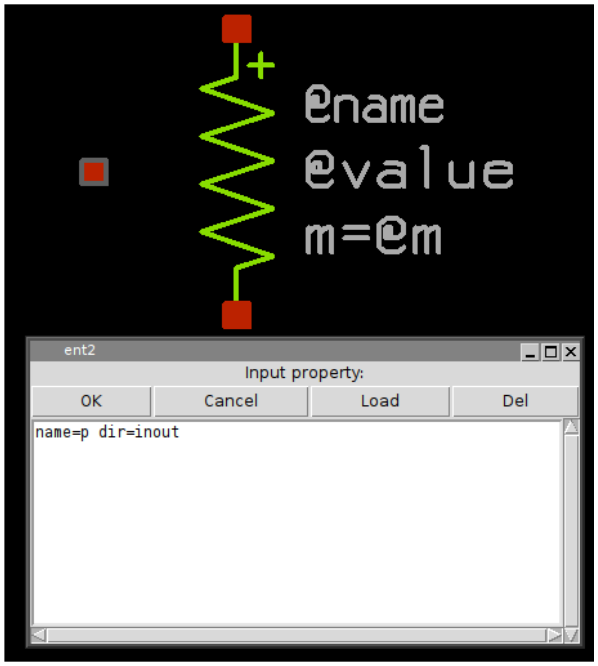


The `format` attribute defines the format of the SPICE netlist. The SPICE netlist element line starts with the symbol name (in this case a resistor so 'rxxxxx'), the list of pins, the resistor value and a multiplicity factor (m).
`@pinlist` will resolve to the parent nets attached to the resistor nodes, 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.



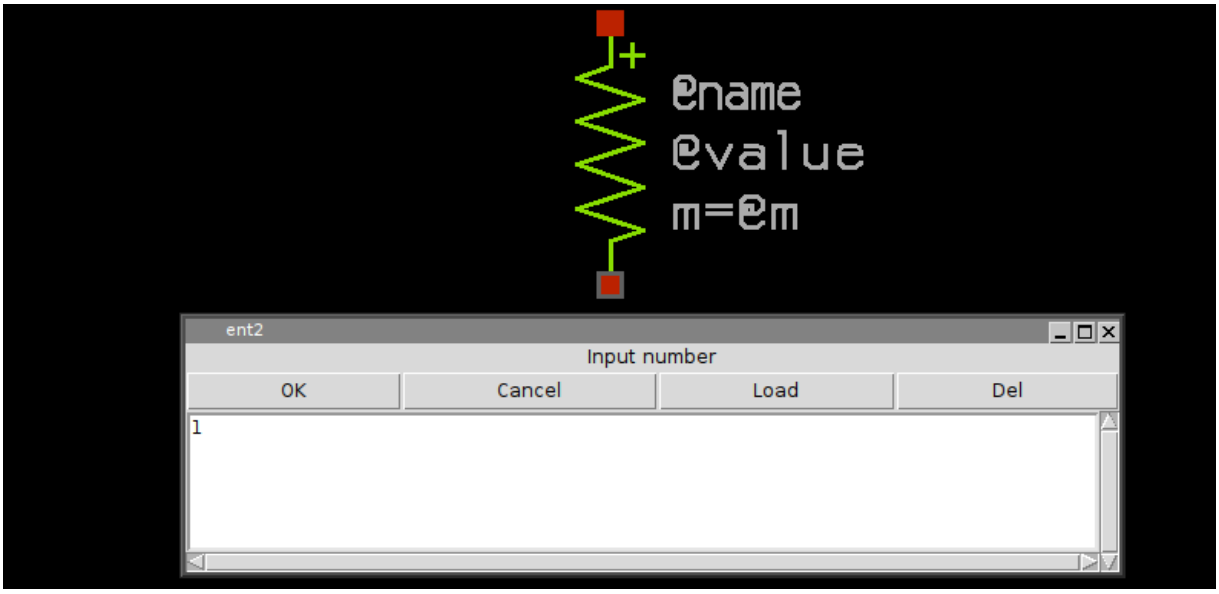


After copying the pin you may change its properties, for example you will change its property string to something like: `name=body dir=in` (just as an example).

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.

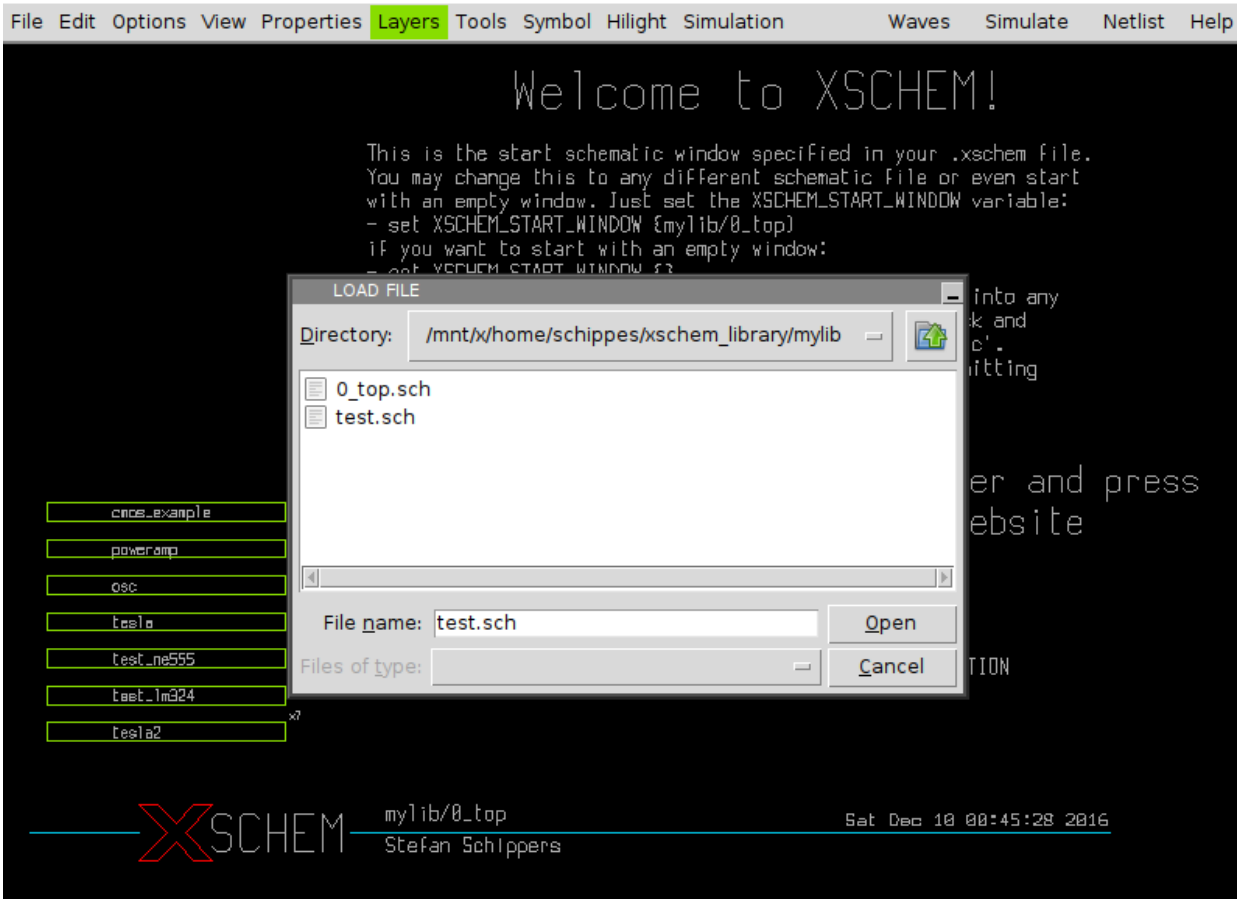
PIN ORDERING

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.

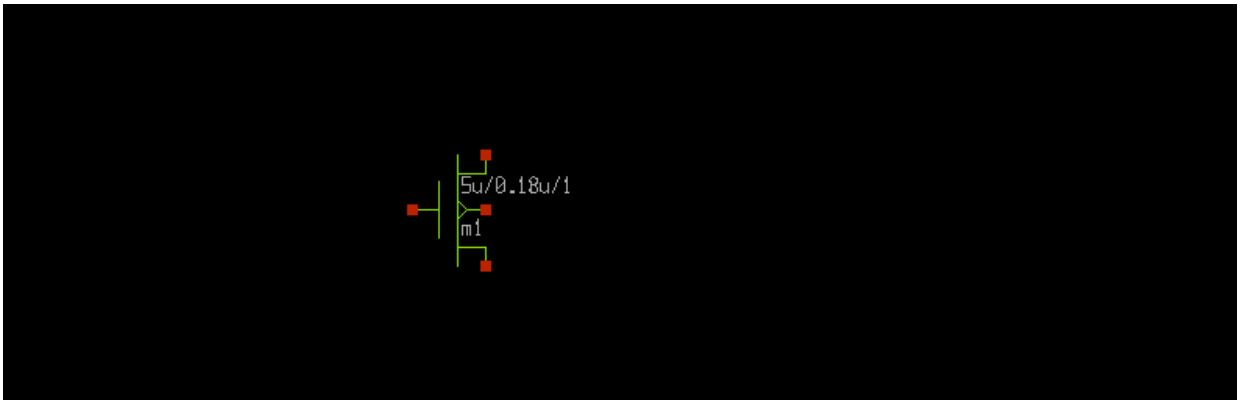


COMPONENT INSTANTIATION

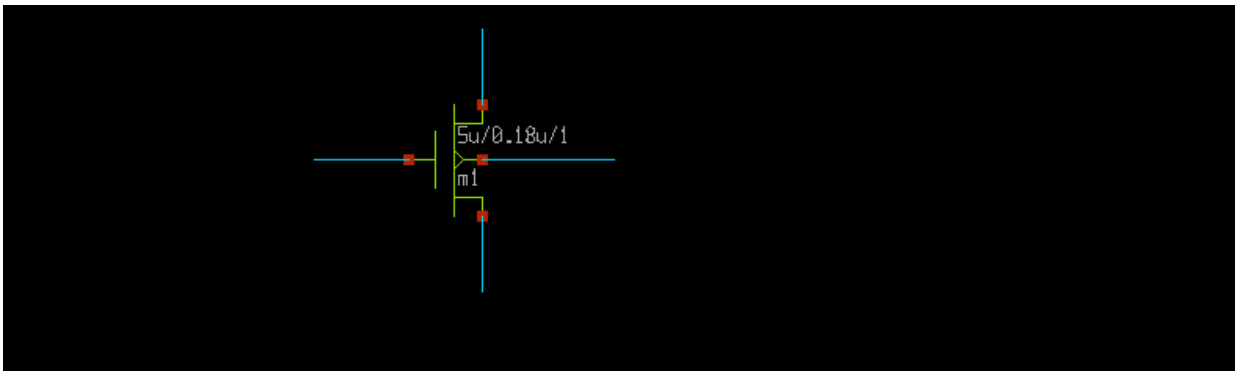
In the RUN XSCHEM slide some instructions were provided as examples to place a component in the schematic. Now we will cover the topic in more detail with emphasis on component properties. Start by opening a test schematic window (you may delete any existing stuff in it if any).



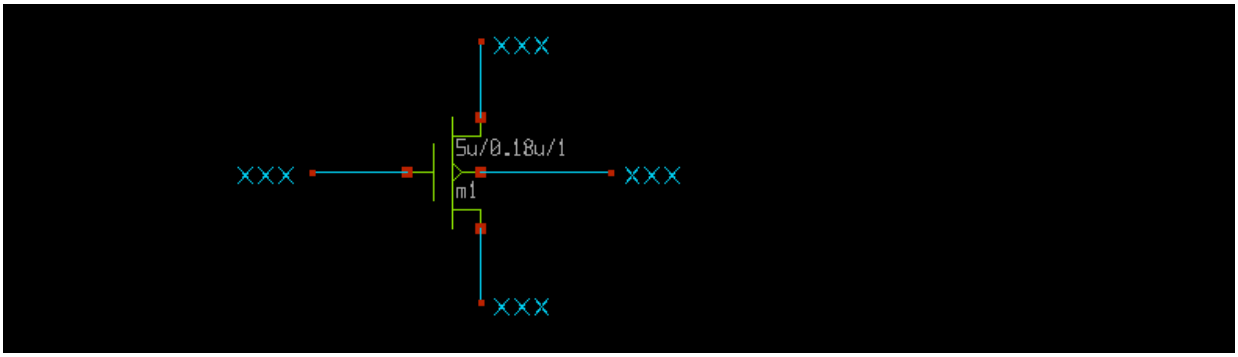
Now start by inserting a component, consider for example `devices/nmos4.sym`; press the `Insert` key, navigate to the `devices` design library and open the `nmos4.sym` symbol.



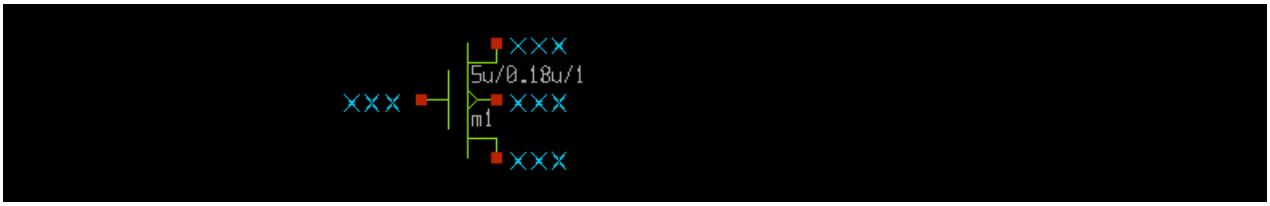
Now draw some wires on each pin of the nmos; place the mouse pointer on the component pins and use the `'w'` bindkey.



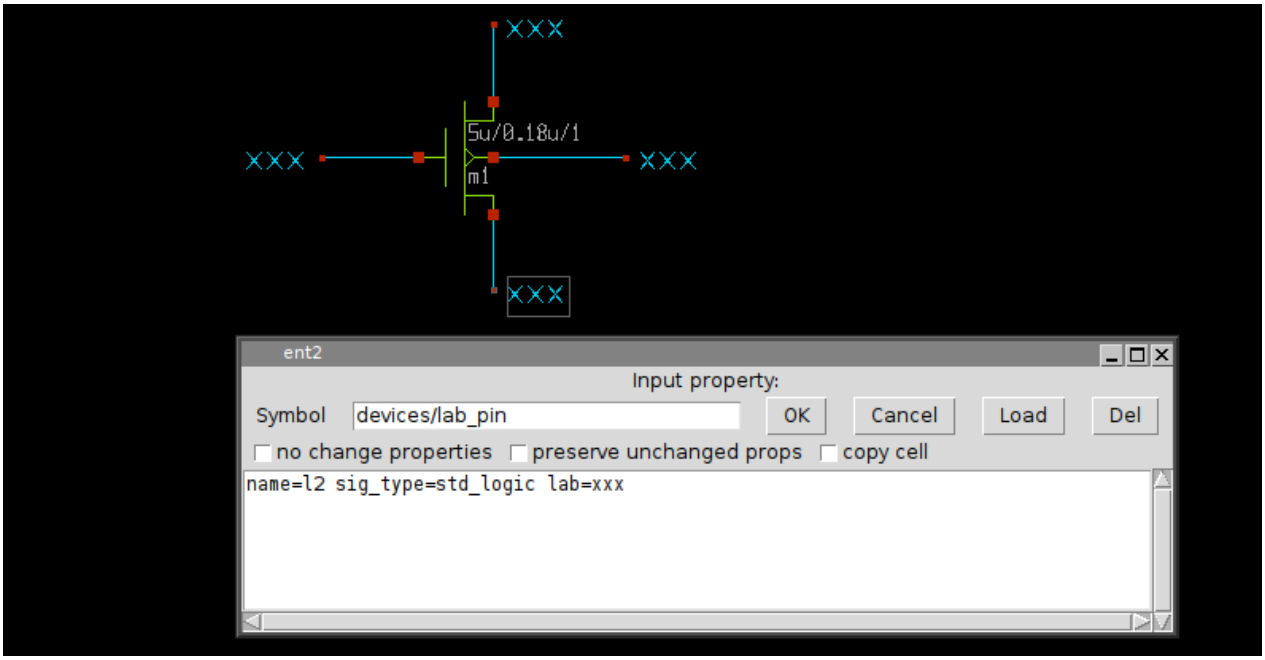
we need now to put labels on wire ends: use the `Insert` key and locate the `devices/lab_pin.sym` symbol. After the `lab_pin` symbol is placed you can move it by selecting it with the mouse and pressing the `'m'` bindkey. You can also flip (`'F'`) and rotate while moving (`'R'`) to adjust the orientation.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



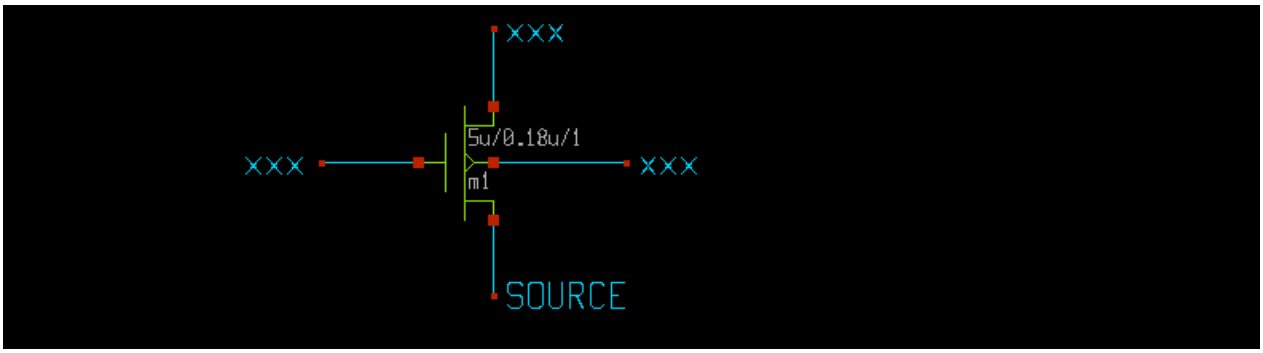
This is what an electrical circuit is all about: a network of wires and components. In this schematic we have 5 components (4 labels and one mos) and 4 nets. It is not mandatory to put a wire segment between component pins; we could equally well do this:



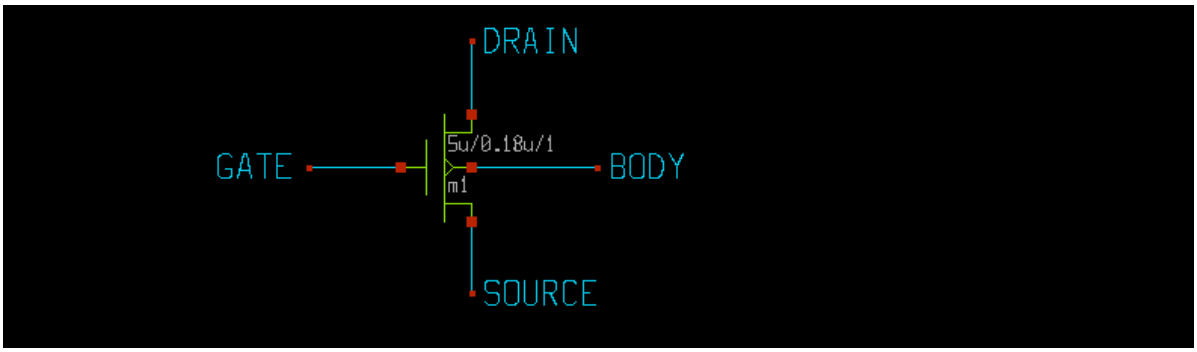
This circuit is absolutely equivalent to the previous one: it will produce the same device connectivity netlist.
Now we need to set appropriate labels on the NMOS terminals. This is -again- accomplished with component properties. Select the wire label on the nmos source pin and press the `'q'` bindkey:



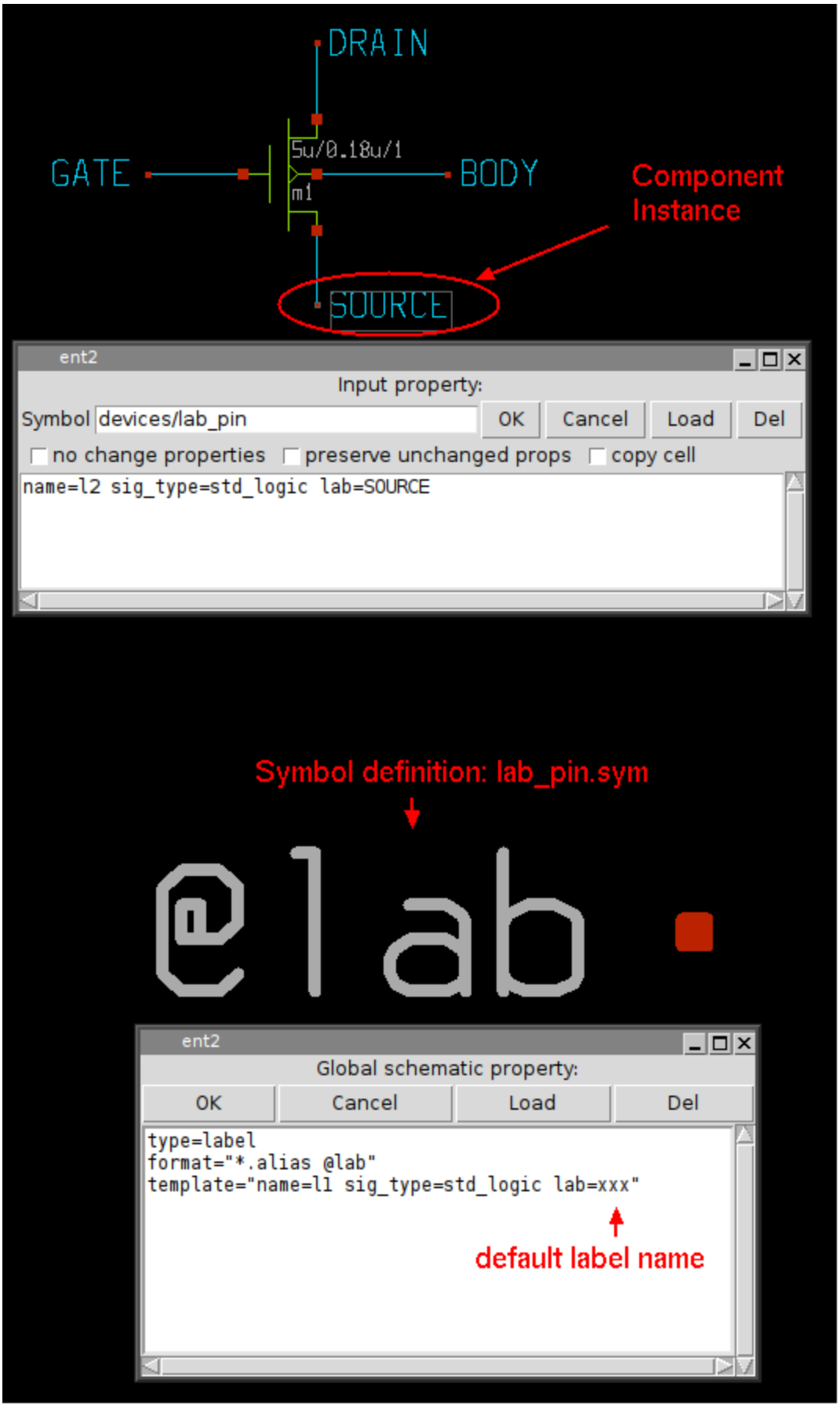
Now, replace the 'xxx' default string in the dialog with a different name (example: SOURCE) After clicking `OK` the source terminal will have the right label.



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



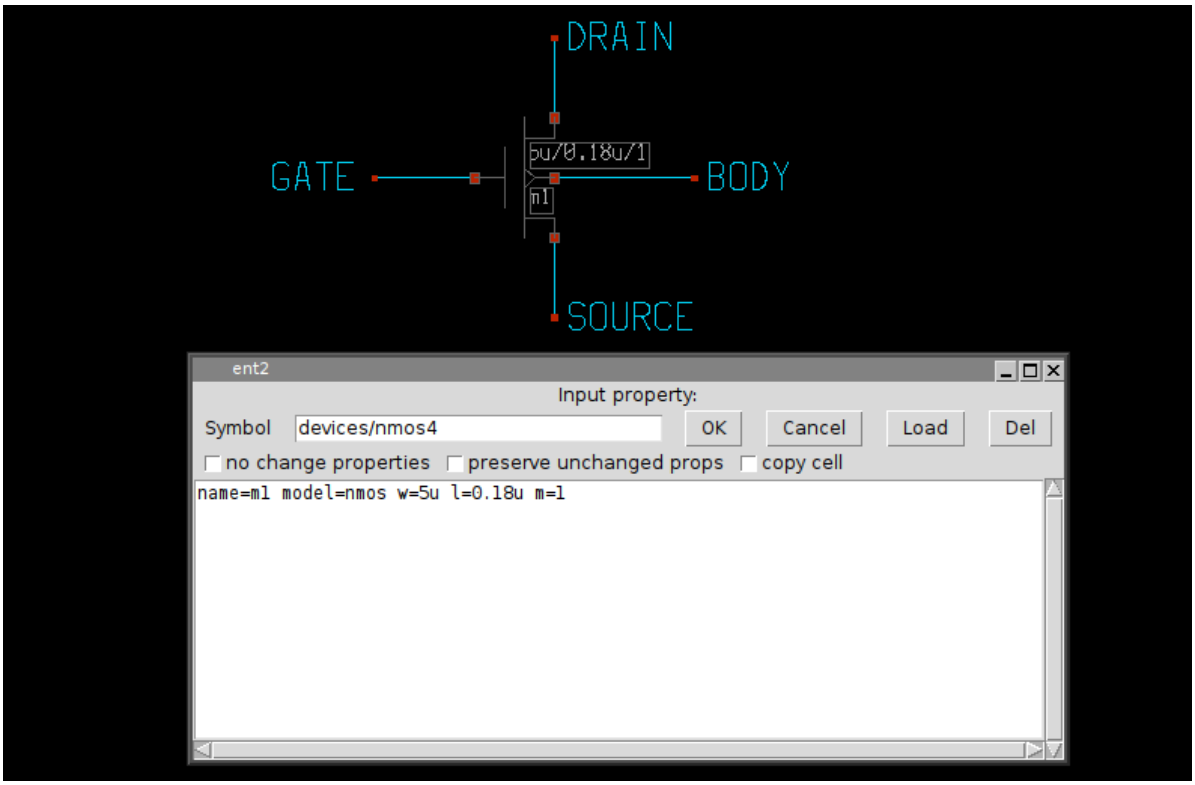
The following picture shows the `lab_pin` component with its properties and the corresponding symbol definition with its global properties (remember global properties in the xschem_properties slide)


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
@lab
ent2
Global schematic property:
OK
Cancel
Load
Del
type=label
format="*.alias @lab"
template="name=l1 sig_type=std_logic lab=xxx"
default label name

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
```

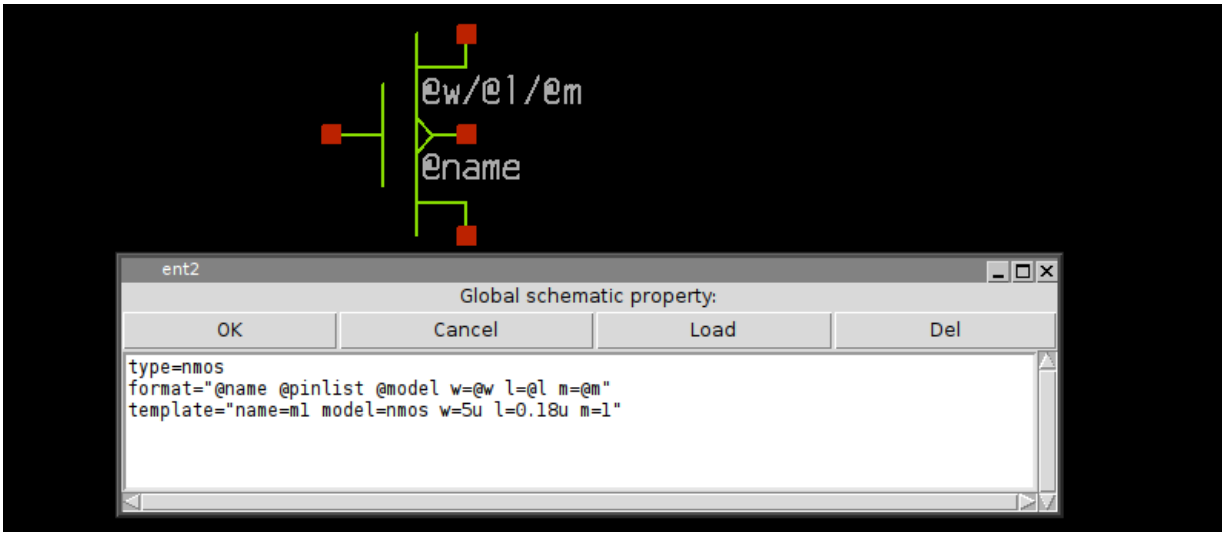
We need now to edit the nmos properties. Select it and press the `'q'` bindkey



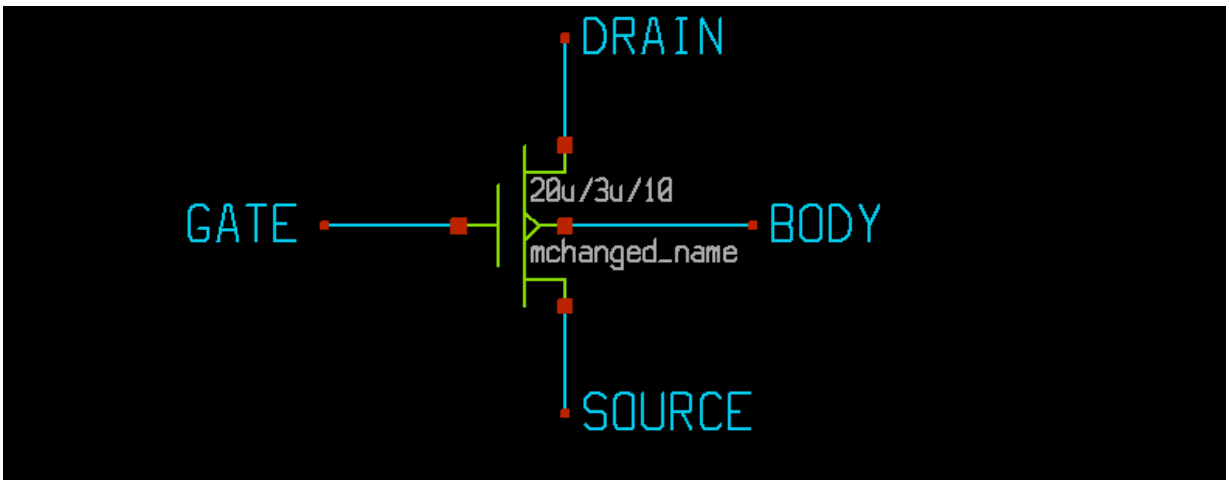
from the edit properties dialog you see there are 5 attributes with values defined:

- 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).

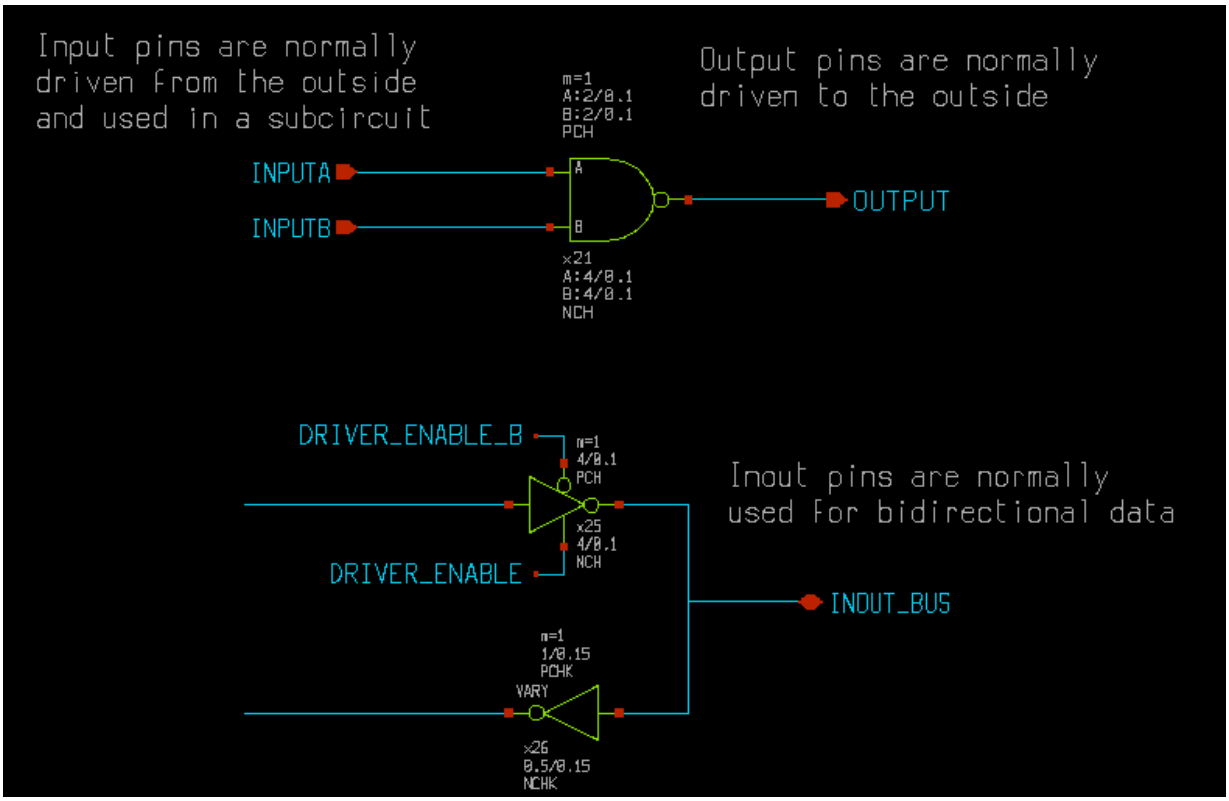
the `name` attribute is unique in the schematic window, and must be placed first in the property string. The name is also used by xschem to efficiently index it in the internal hash tables.

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
- devices/code.sym

This symbol is used to place simulator commands into a schematic.

Spice netlist special components

- devices/netlist.sym
- devices/netlist_not_shown.sym

Verilog netlist special components

- devices/verilog_timescale.sym

VHDL netlist special components

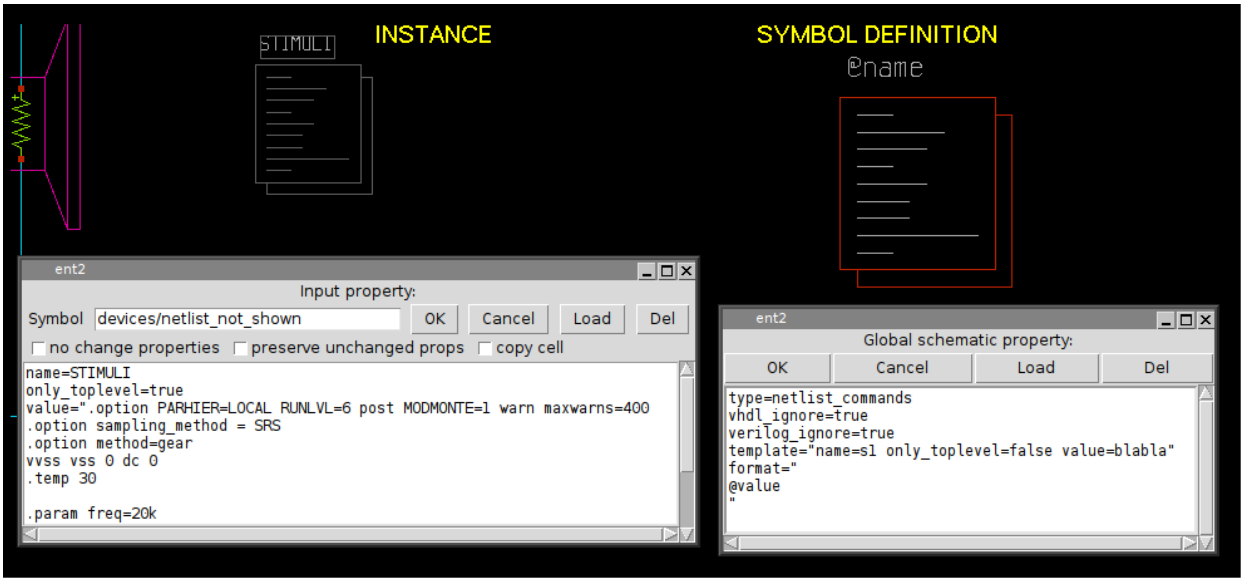
- devices/use.sym
- devices/package.sym
- devices/package_not_shown.sym
- devices/arch_declarations.sym
- devices/attributes.sym
- devices/port_attributes.sym
- devices/generic_pin.sym
- devices/generic.sym

SYMBOL PROPERTY SYNTAX

GENERAL RULES

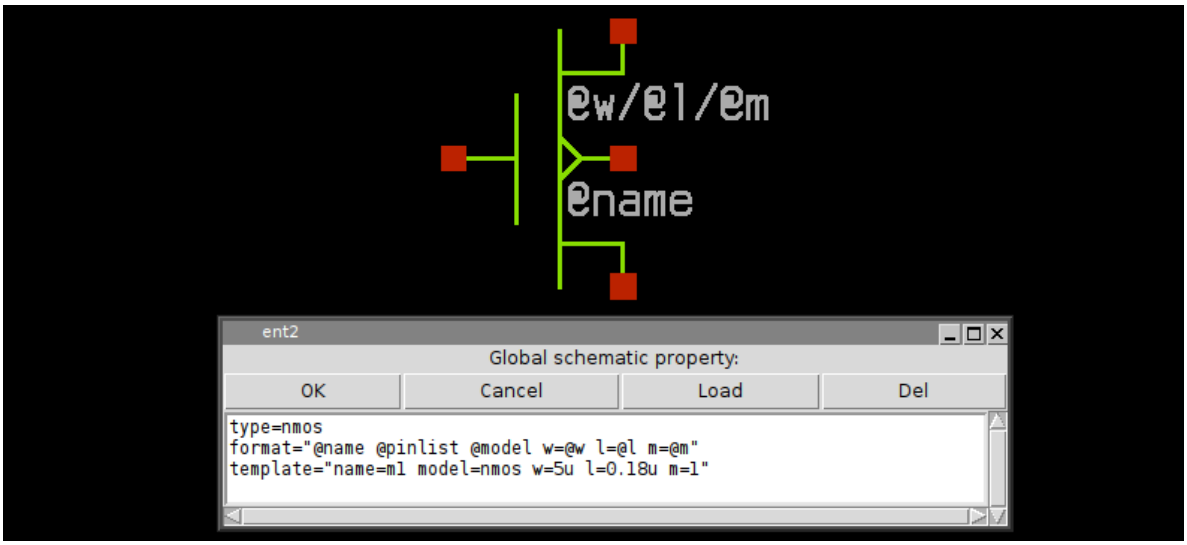
For symbols a global property string (to show it press `'q'` when nothing is selected). defines at least 3 attributes:

- `type` defines the the type of symbol. Normally the type attribute describes the symbol and ignored by XSCHEM, but there are some special types:
  - `subcircuit`: the symbol has an underlining 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 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` you can quickly change the behavior.
  - `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 from component instantiation
  - `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).
- `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=1"
```

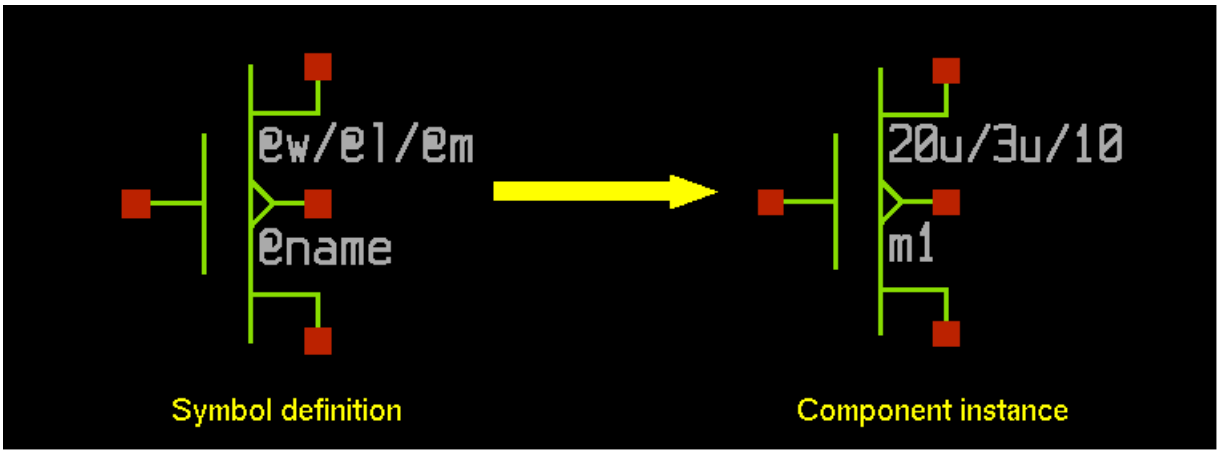
```
format="@name @pinlist @model w=@w l=@l m=@m"
template="name=m1 model=nmos w=5u l=0.18u m=1"
type=nmos
```

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 (`\@`). Attribute substitution takes place in symbol format attribute and in every text, as shown in below picture.



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`

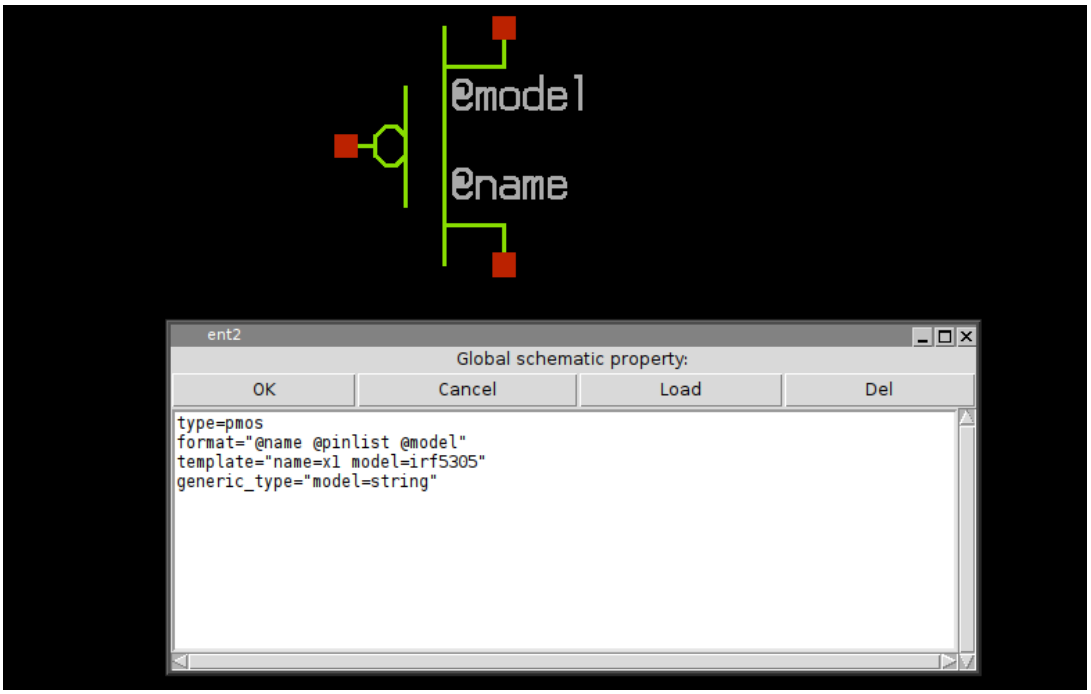
  These 3 attributes will avoid XSCHEM to descend into the schematic representation of the symbol (if there is one) when building the respective netlist format. For example, if an analog block has a schematic (.sch) file describing the circuit that is meaningless when doing a VHDL netlist, we can use a `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.

- `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.

- `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.

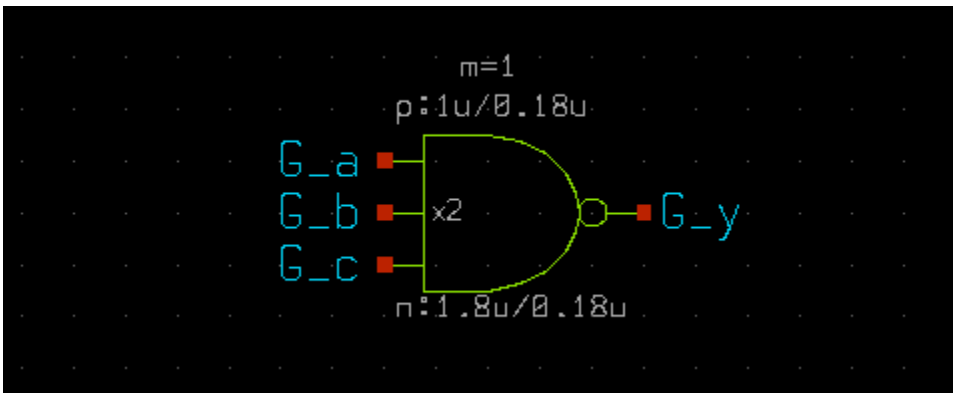
```
entity test2 is
end test2 ;

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

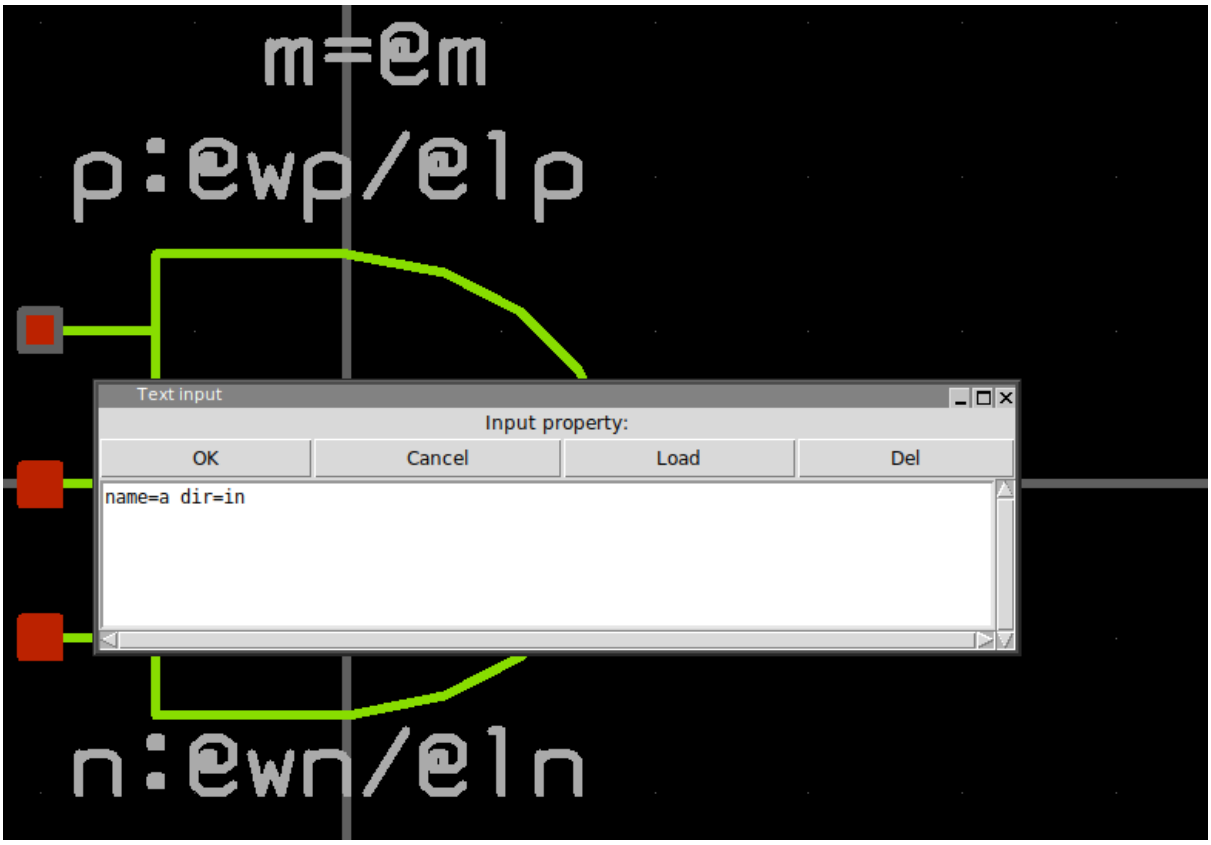

.subckt lvnand3 y a b c
wn=30u ln=2.4u wp=20u lp=2.4u
VCCPIN=VCC VSSPIN=VSS
*.opin y
*.ipin a
*.ipin b
*.ipin c
m1 net2 a VSSPIN VSSPIN nlv w=wn l=ln geomod=0 m=1
```

```
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.

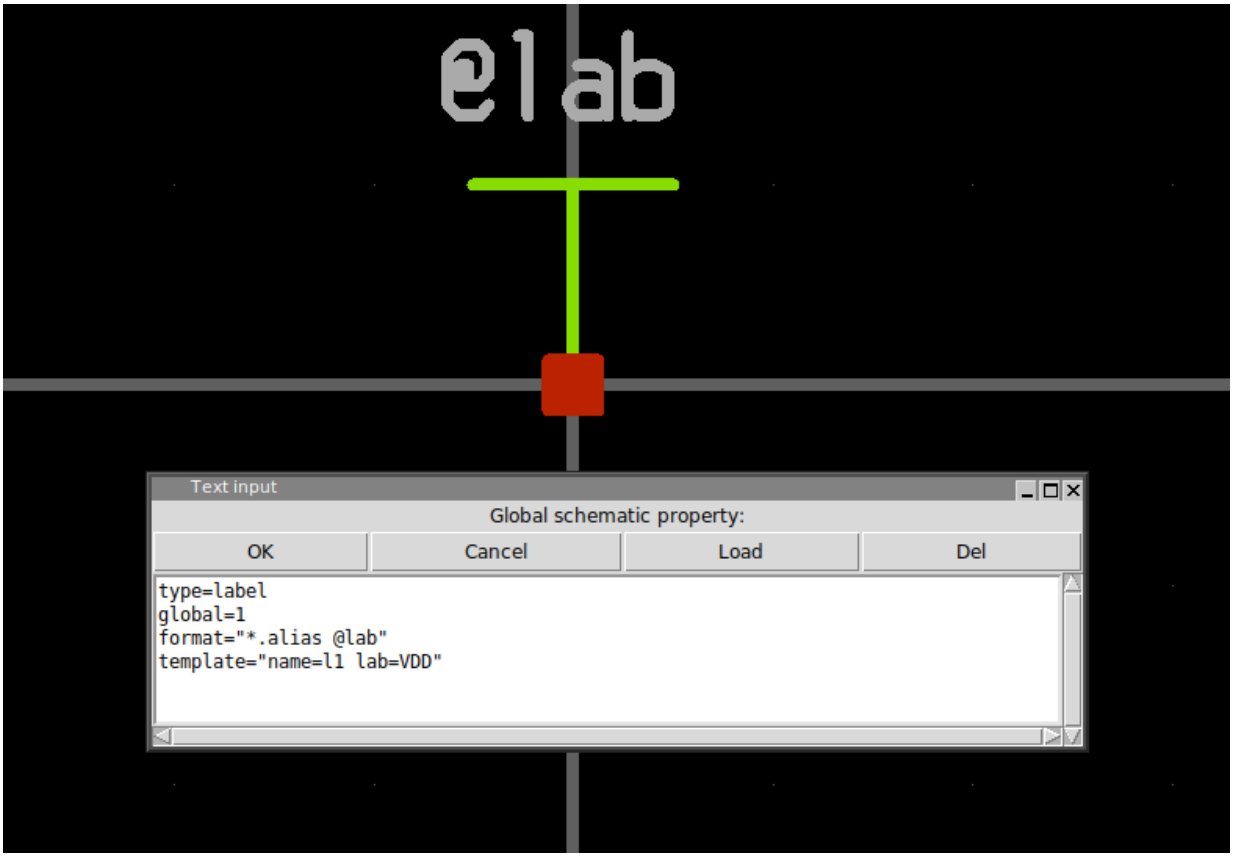
- `dir`

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



- `global`

a `global=1` 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.

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.

- `@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`

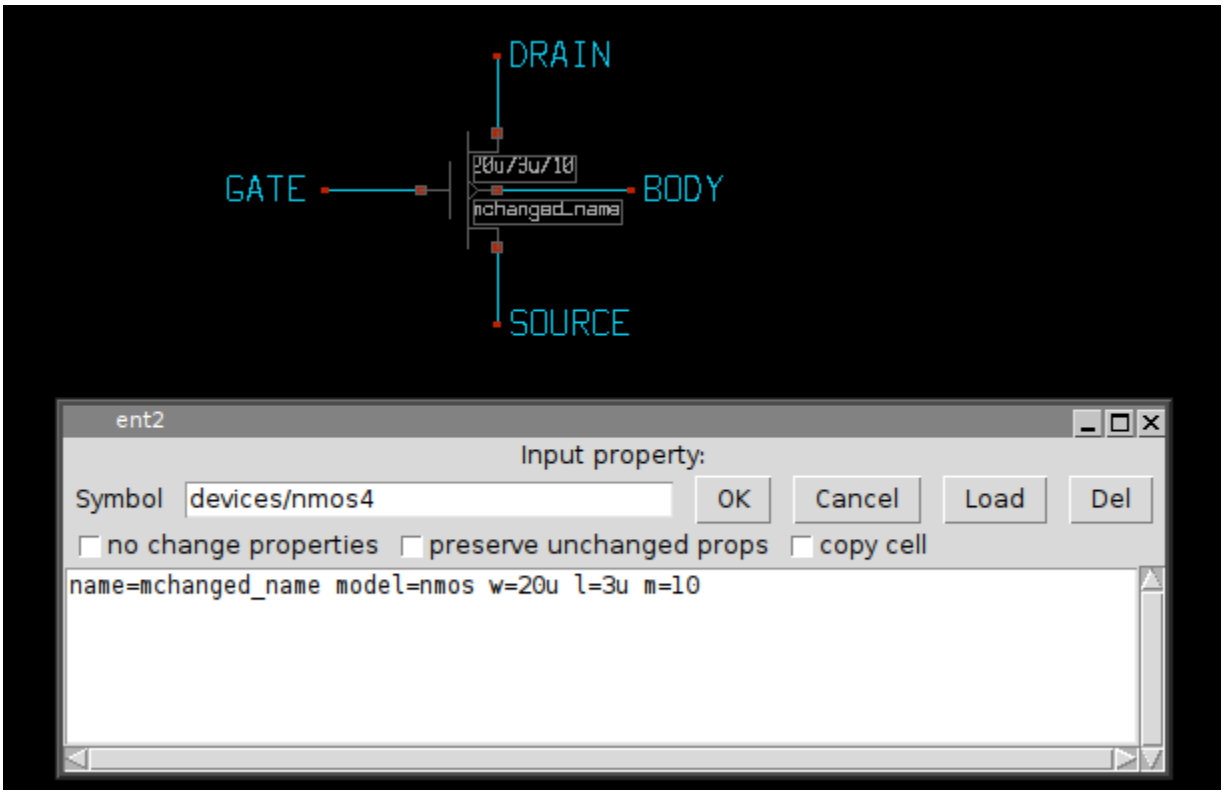
  this expandes to the **VHDL** global property string of the schematic containing the symbol

- `@schverilogprop`

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

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 before any other attributes in component property string. The `name` attribute is used by XSCHEM -among other things- for fast indexing the component. 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. Must be the first `attribute=value` in the component property string. 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, 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`

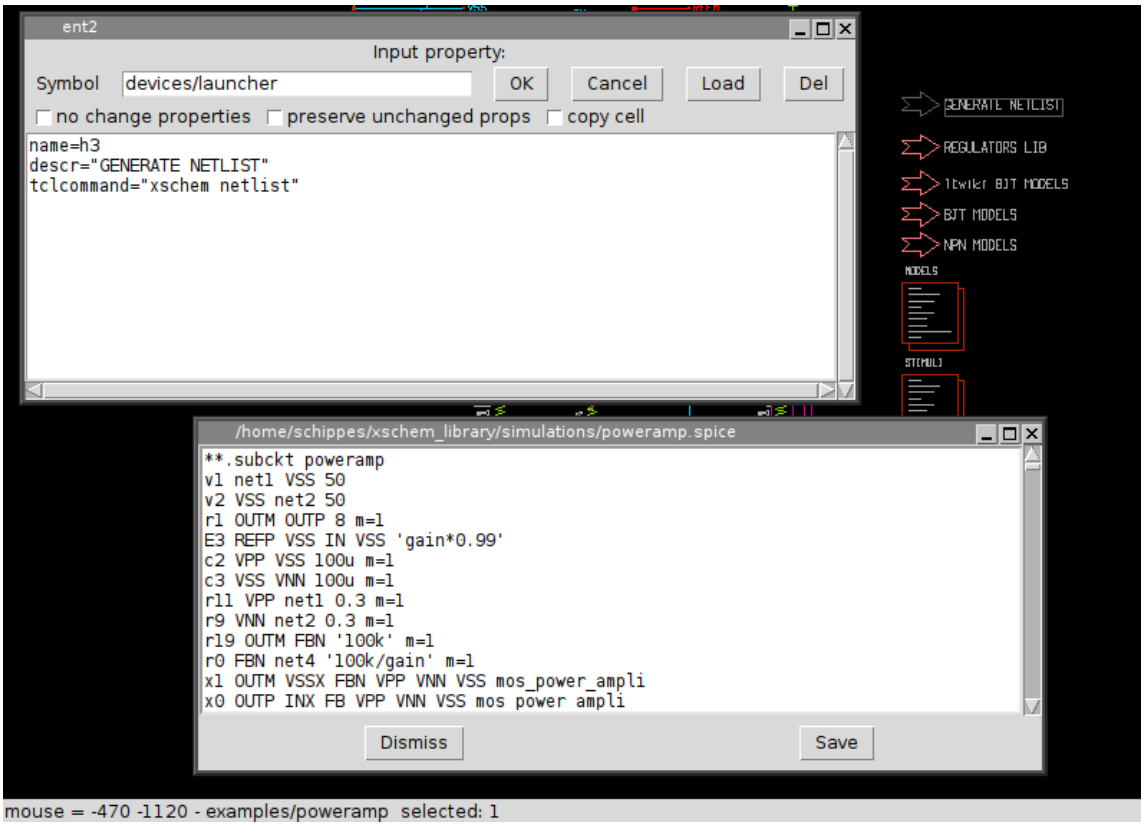
This attribute defines a location (web page, file) that can be viewed when hitting the `<shift>H` key (or `<Alt> left mouse buttoni`) on a selected component. This is very useful to link a datasheet to a component, for example. The default program used to open the url is `xdg-open`. this can be changed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

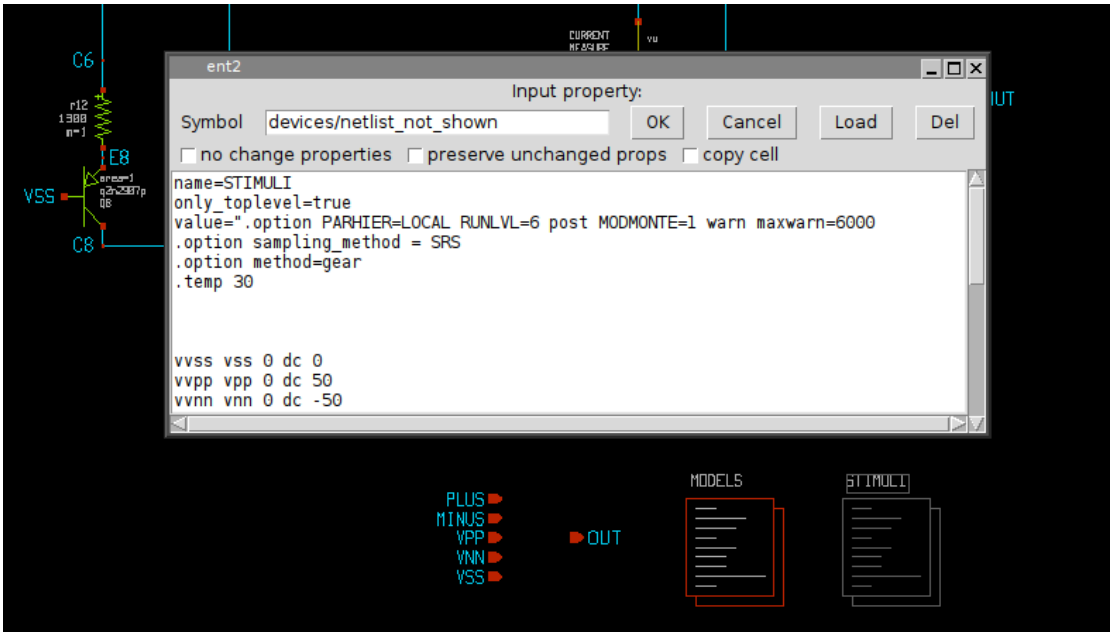
- `tclcommand`

  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.



- `place`

  This `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.

- `spice_ignore`

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

- `verilog_ignore`

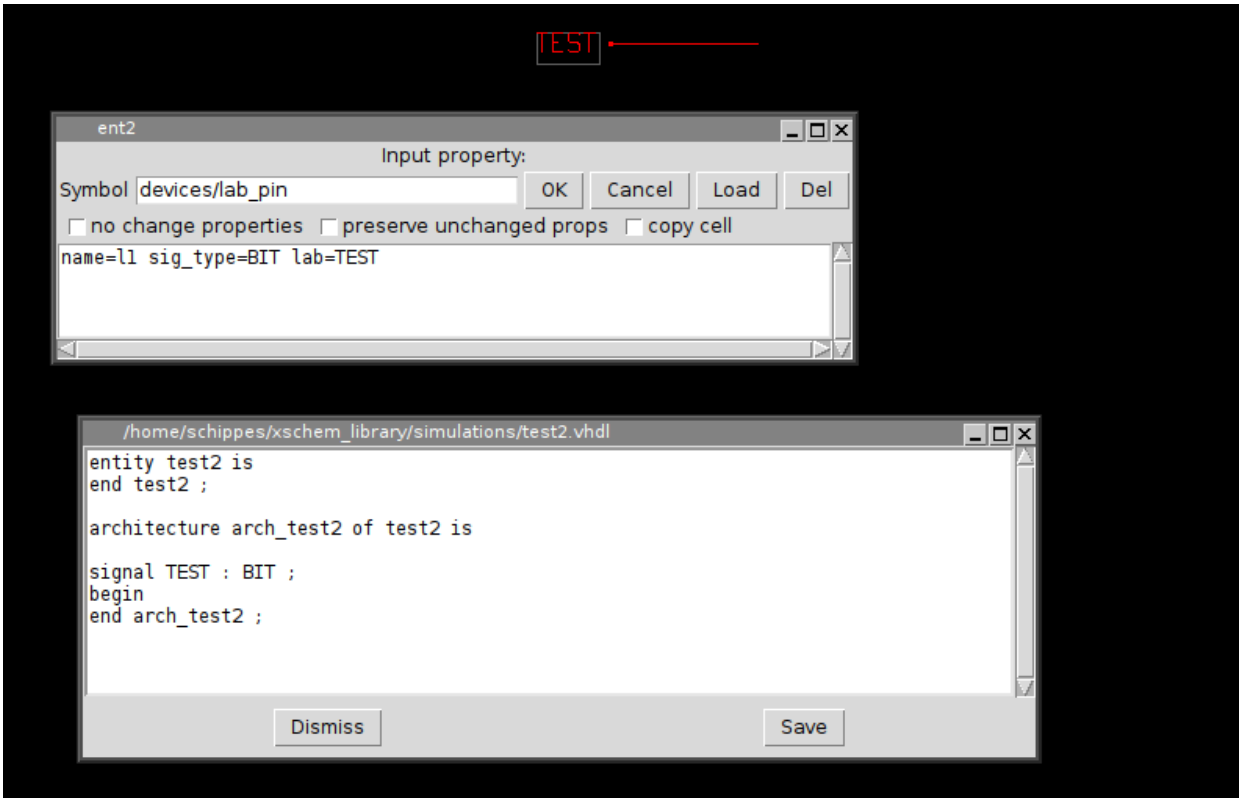
  This tells XSCHEM that for Verilog netlist this component will be **completely** ignored.

- `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`.

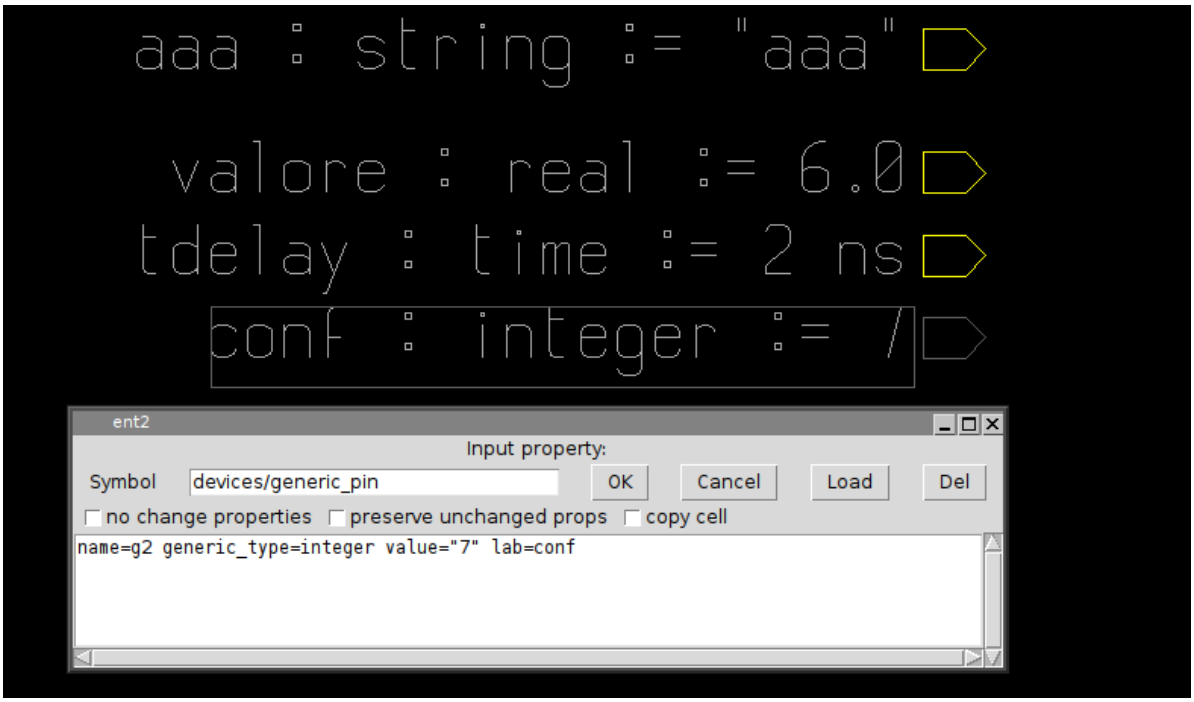


- `verilog_type`

  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:



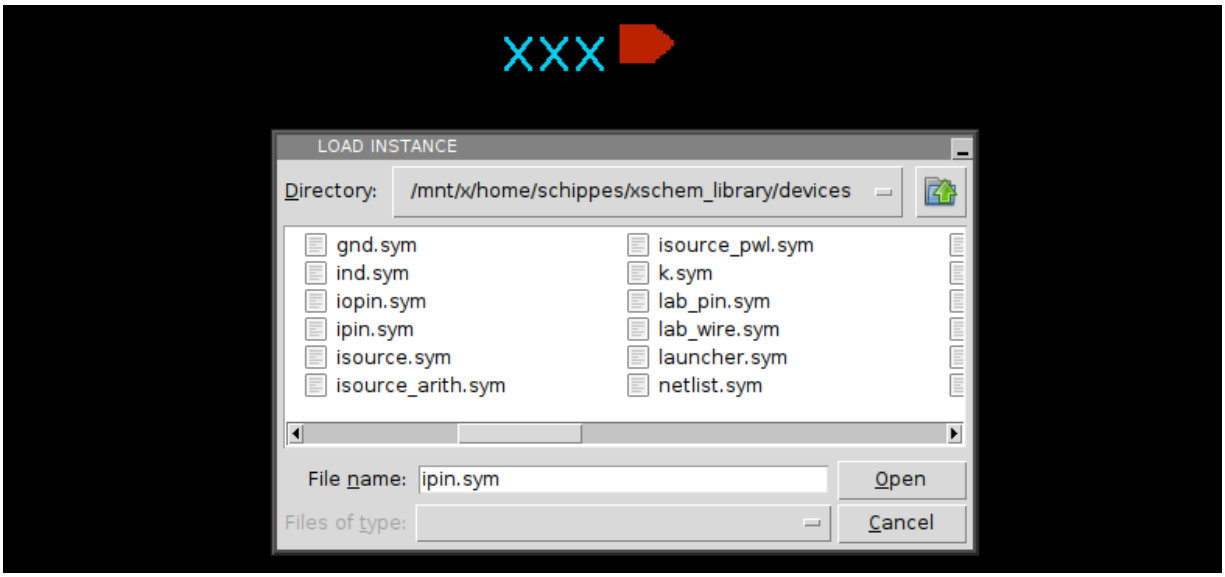
As you will see in the parameters slide, generics (they are just parameters passed to components) can be passed also via property strings in addition to using `generic_pin` components.

- `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`.

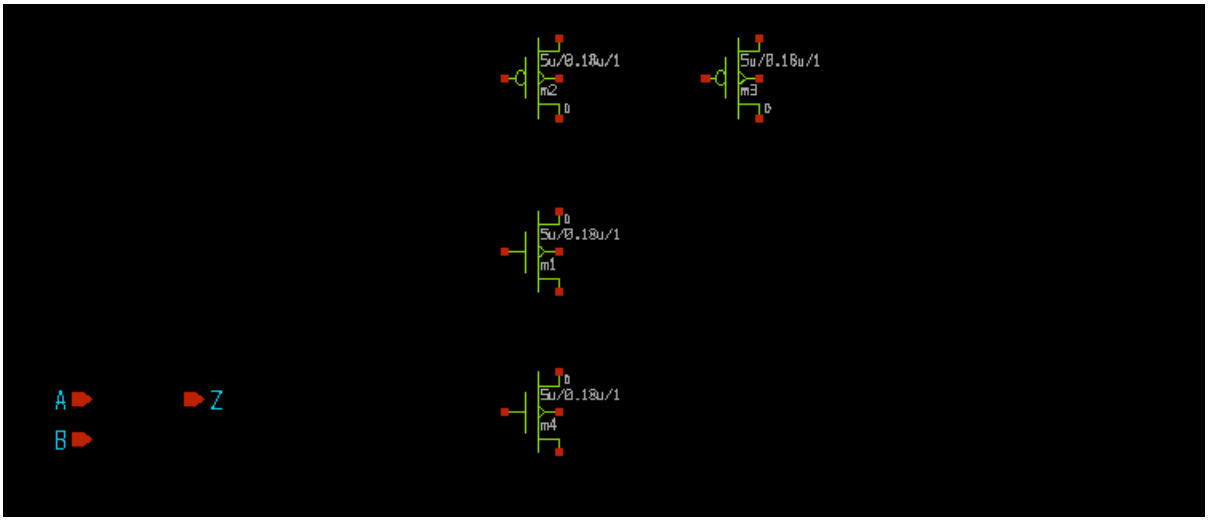
CREATING A CIRCUIT SCHEMATIC

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'`

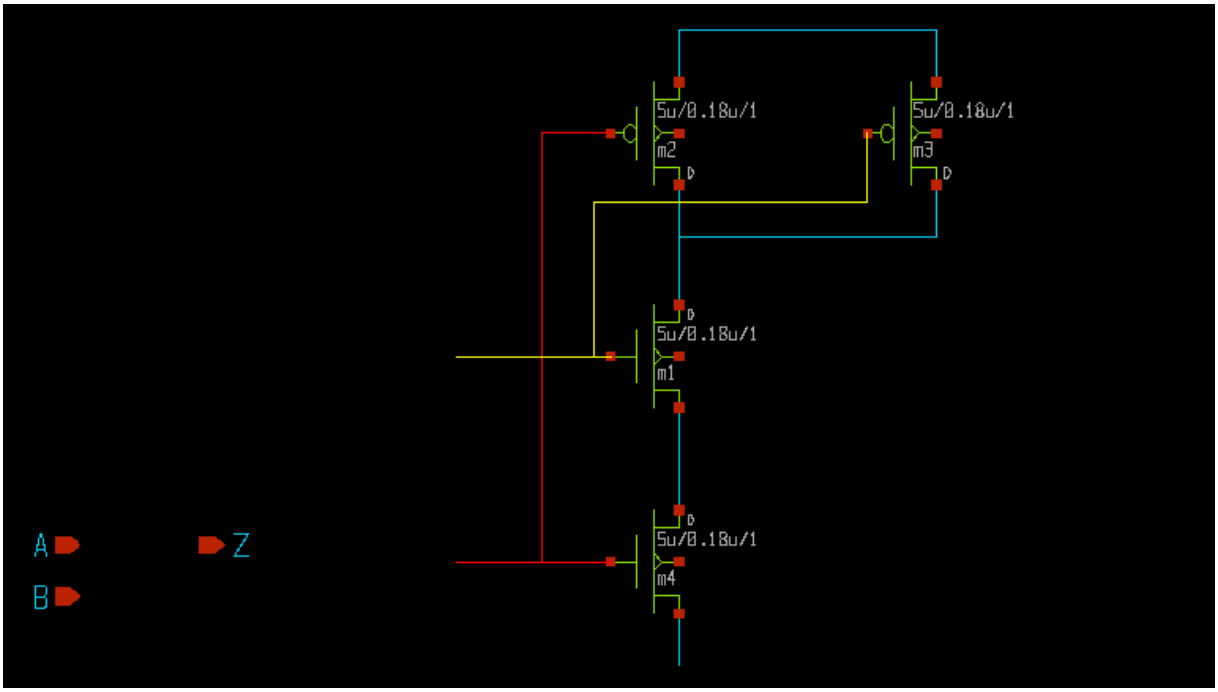


Copy another instance of it and set its lab attribute to `B`. Next place an output pin `devices/opin.sym` and set its lab to `Z`. The result will be as follows:

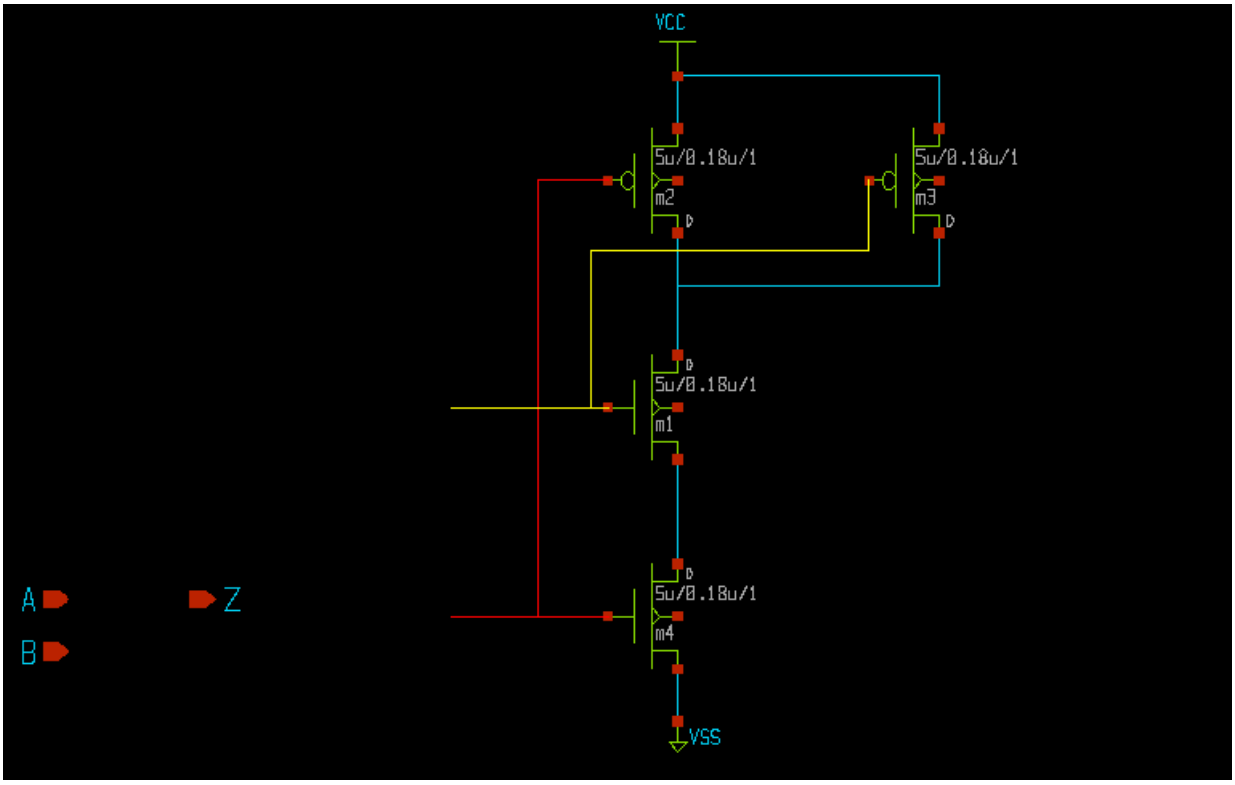
Now we need to build the actual circuit. Since we plan to do it in CMOS technology we need nmos and pmos transistors. Place one nmos from `devices/nmos4.sym` and one pmos from `devices/pmos4.sym` By selecting them with the mouse, moving (`m` bindkey), copying (`'c'` bindkey) place 4 transistors in the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



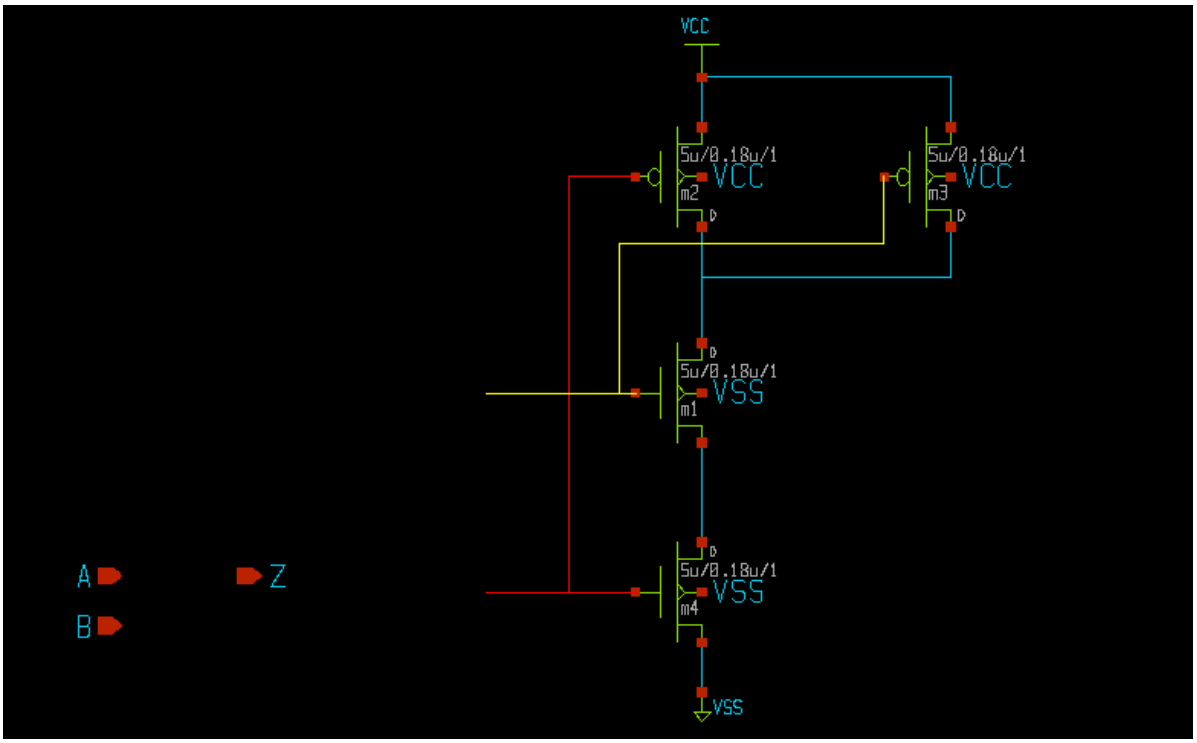
now draw wires to connect together the transistor to form a NAND gate; in the picture i have highlighted 2 electrical nodes by selecting one wire segment of each and pressing the `'k'` bindkey.



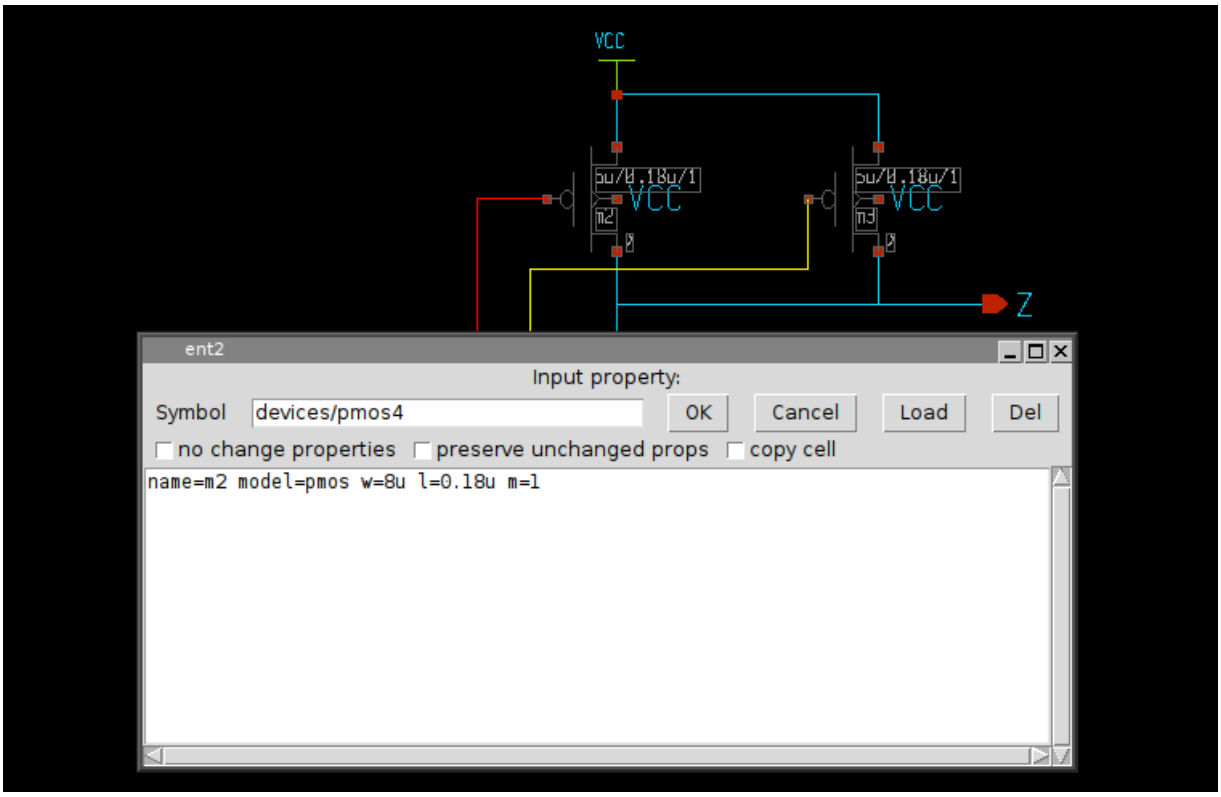
Next we need to place the supply nodes , VCC and VSS. we decide to use global nodes. Global nodes in SPICE semantics are like global variables in C programs, they are available everywhere, we do not need to propagate global nodes with pins. We could equally well use regular pins , as used for the A and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



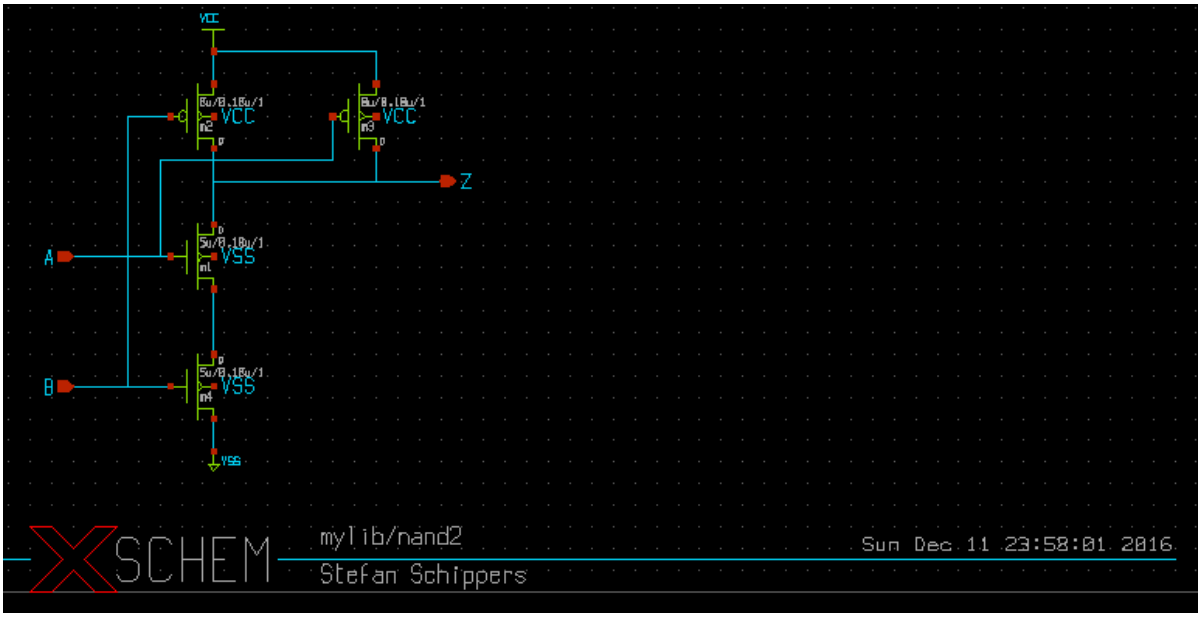
we still need to connect the body terminals of the mos transistors. One possibility is to hookup the two upper pmos transistor terminals to VCC with wires, and the two bottom nmos terminals to VSS with wires, but just to show different design styles i am planning to use "by name" connection with labels. 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.



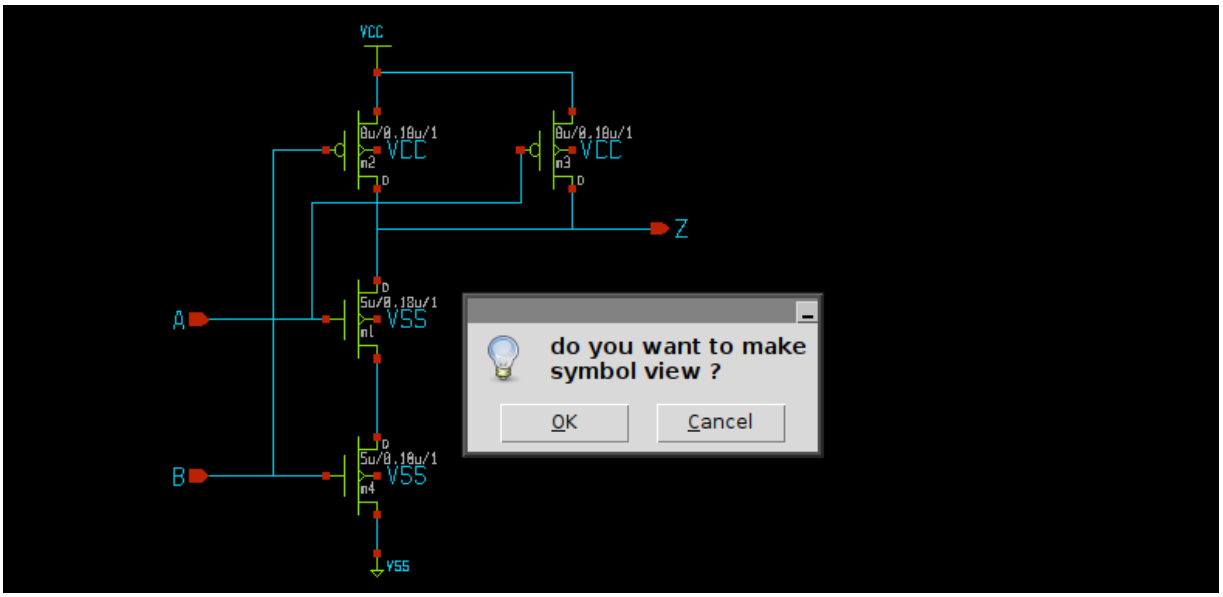
Now do a Save as operation, save it for example in `mylib/nand2.sch`.
To make the schematic nicer we also add the title component. This component is not netlisted but is useful, it reports the modification date and the author. Place the `devices/title.sym` component. The NAND gate is completed! (below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

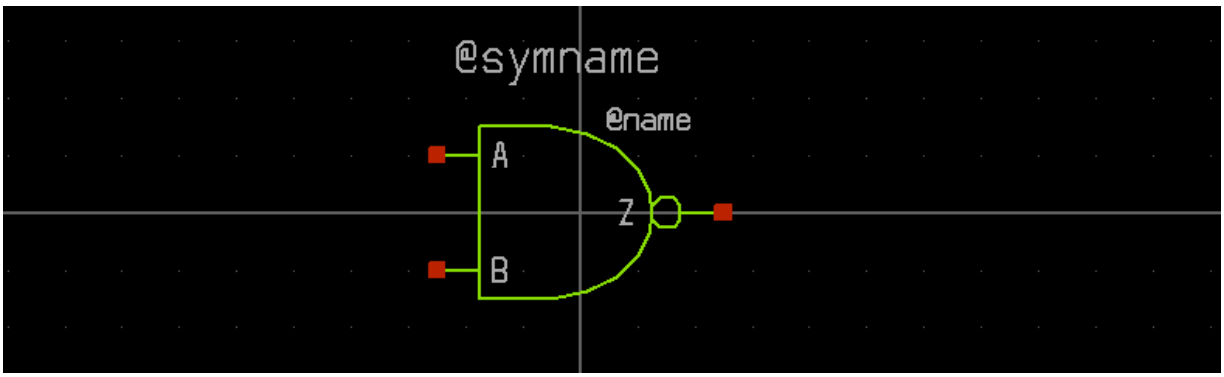
Automatic symbol creation

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.

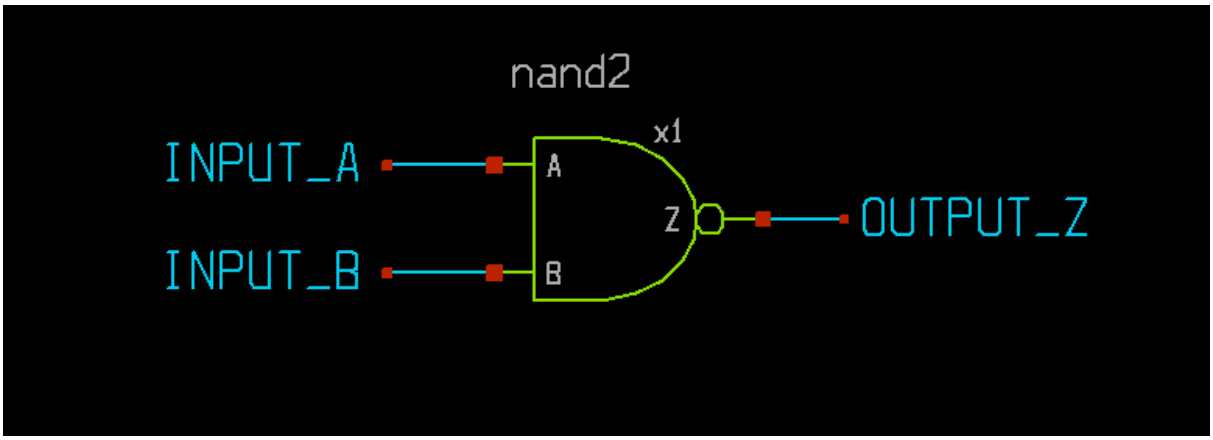


After pressing 'OK' a `mylib/nand2.sym` file is generated. try opening it (`File->Open`):

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:



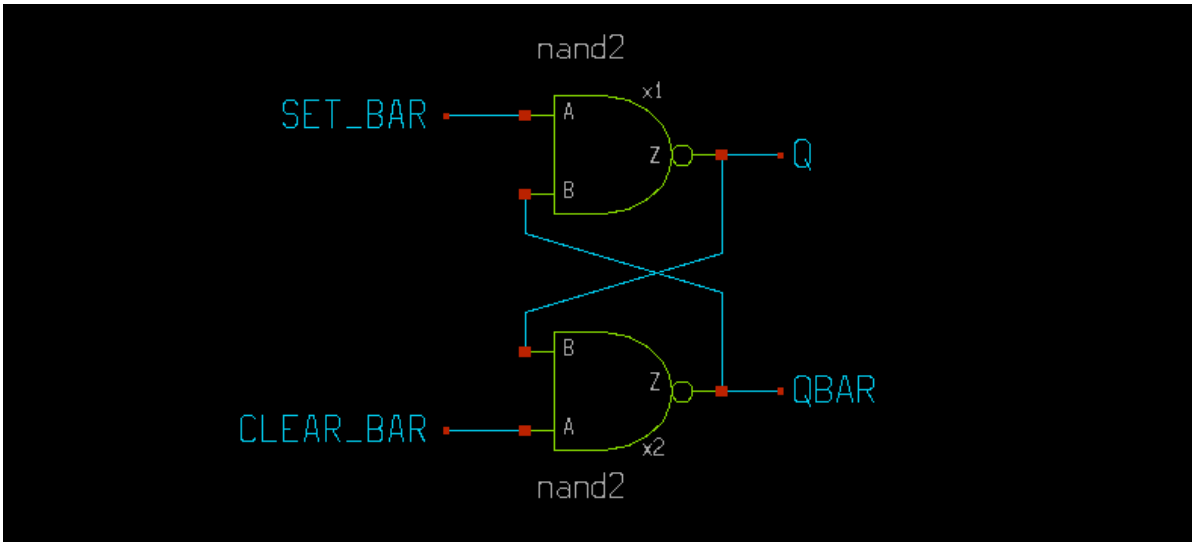
This is now a valid circuit. Let's test it by extracting the SPICE netlist. Enable the showing of netlist window (`Options -> Show netlist win`, or `'A'` key). Now extract the netlist (`Netlist` button on the right side of the menu bar, or `'N'` key). the SPICE netlist will be shown.

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

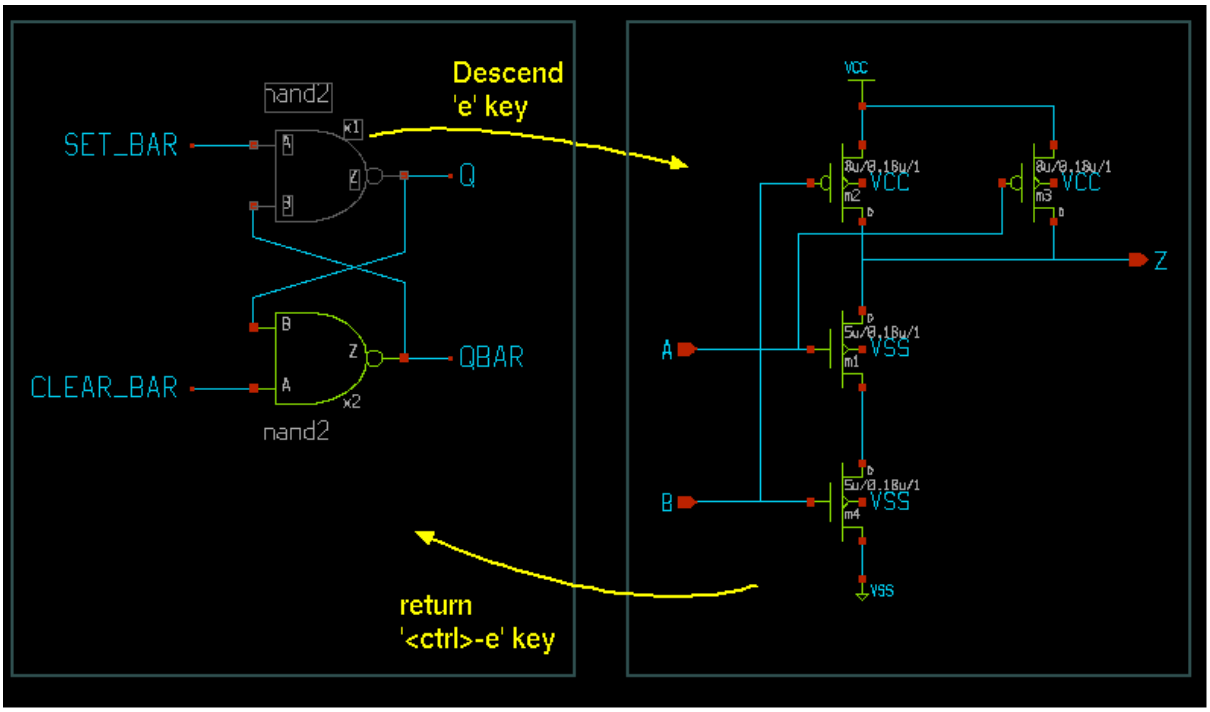
* expanding symbol: mylib/nand2 # of pins=3
.subckt nand2 Z A B
*.ipin A
*.opin Z
*.ipin B
m1 Z A net1 VSS nmos w=5u l=0.18u m=1
m2 Z B VCC VCC pmos w=8u l=0.18u m=1
m3 Z A VCC VCC pmos w=8u l=0.18u m=1
m4 net1 B VSS VSS nmos w=5u l=0.18u m=1
**** begin user architecture code
**** end user architecture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

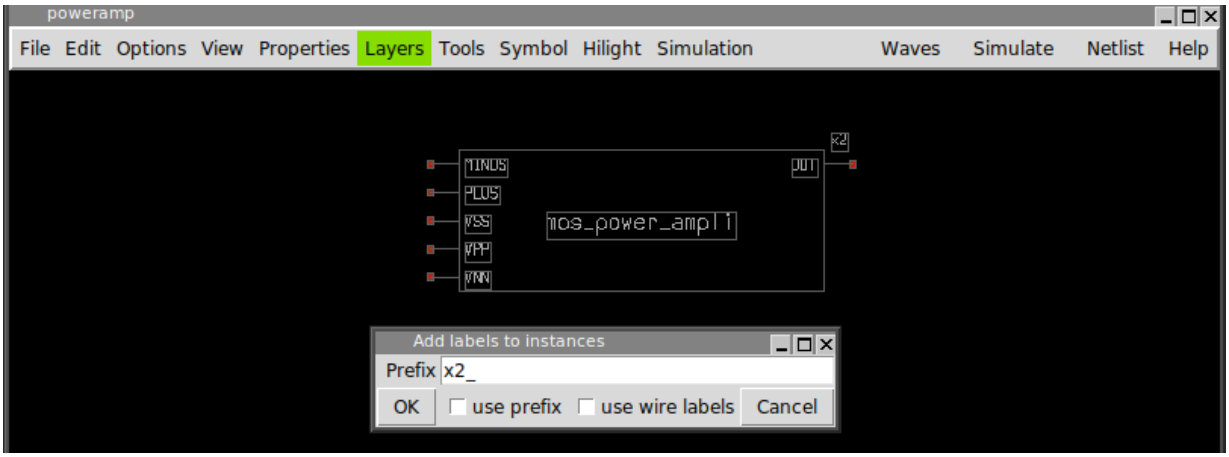
```
* expanding symbol: mylib/nand2 # of pins=3
.subckt nand2 Z A B
*.ipin A
*.opin Z
*.ipin B
m1 Z A net1 VSS nmos w=5u l=0.18u m=1
m2 Z B VCC VCC pmos w=8u l=0.18u m=1
m3 Z A VCC VCC pmos w=8u l=0.18u m=1
m4 net1 B VSS VSS nmos w=5u l=0.18u m=1
**** begin user architecture code
**** end user architecture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

.GLOBAL VCC
.GLOBAL VSS
.end
```

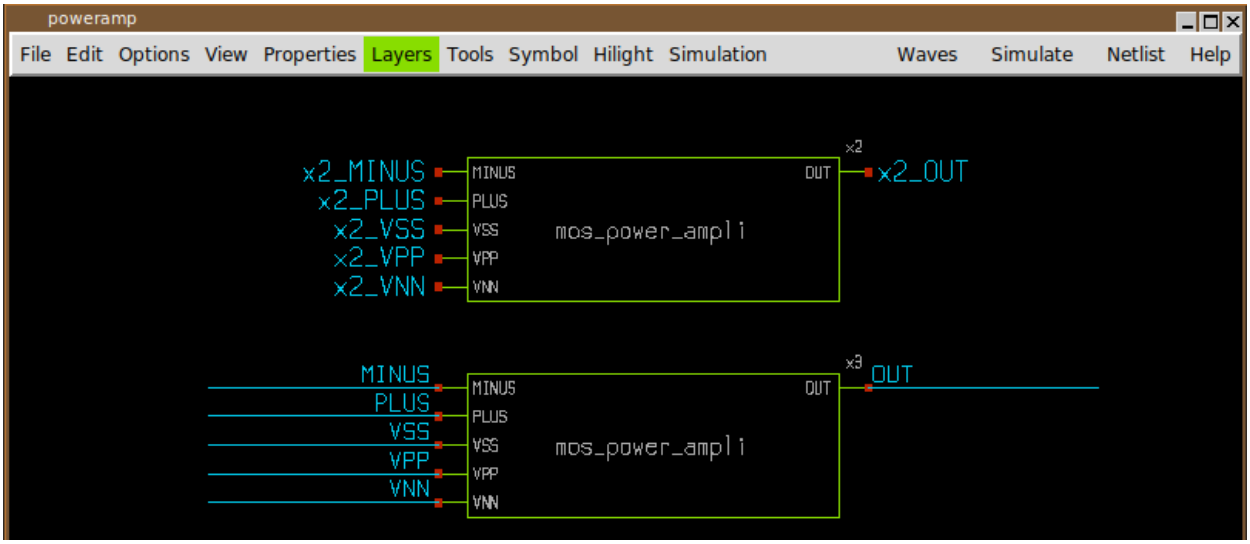
The advantage of using hierarchy in circuits is the same as using functions in programming languages; avoid drawing many repetitive blocks. Also the netlist file will be much smaller.

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 pins to component instance` menu entry.



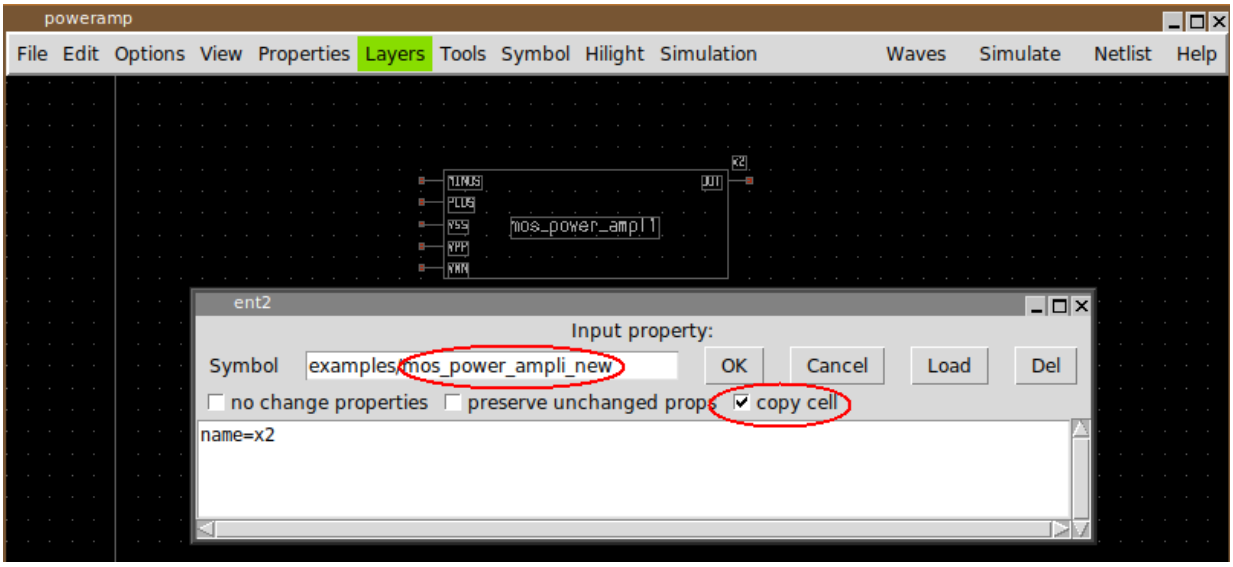
The `use prefix` will prepend the shown prefix to the wire names to be attached to the component. The default value for the prefix is the instance name followed by an underscore. The `use wire labels` will use wire labels instead of pin labels. Wire labels have the text name field offset vertically to 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.


poweramp
File Edit Options View Properties Layers Tools Symbol Hilight Simulation Waves Simulate Netlist Help
x2_MINUS
x2_PLUS
x2_VSS
x2_VPP
x2_VNN
MINUS
PLUS
VSS
VPP
VNN
mos_power_ampli
OUT
x2
x2_OUT
MINUS
PLUS
VSS
VPP
VNN
mos_power_ampli
OUT
x3
OUT

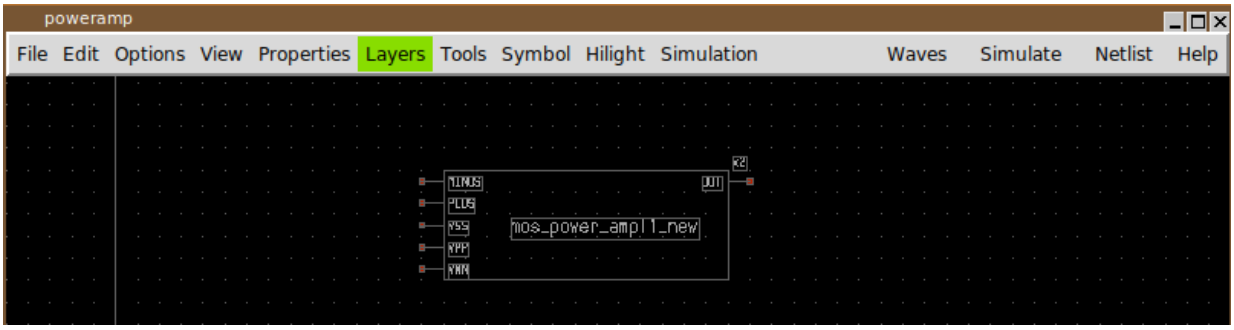
CREATING SYMBOLS

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.

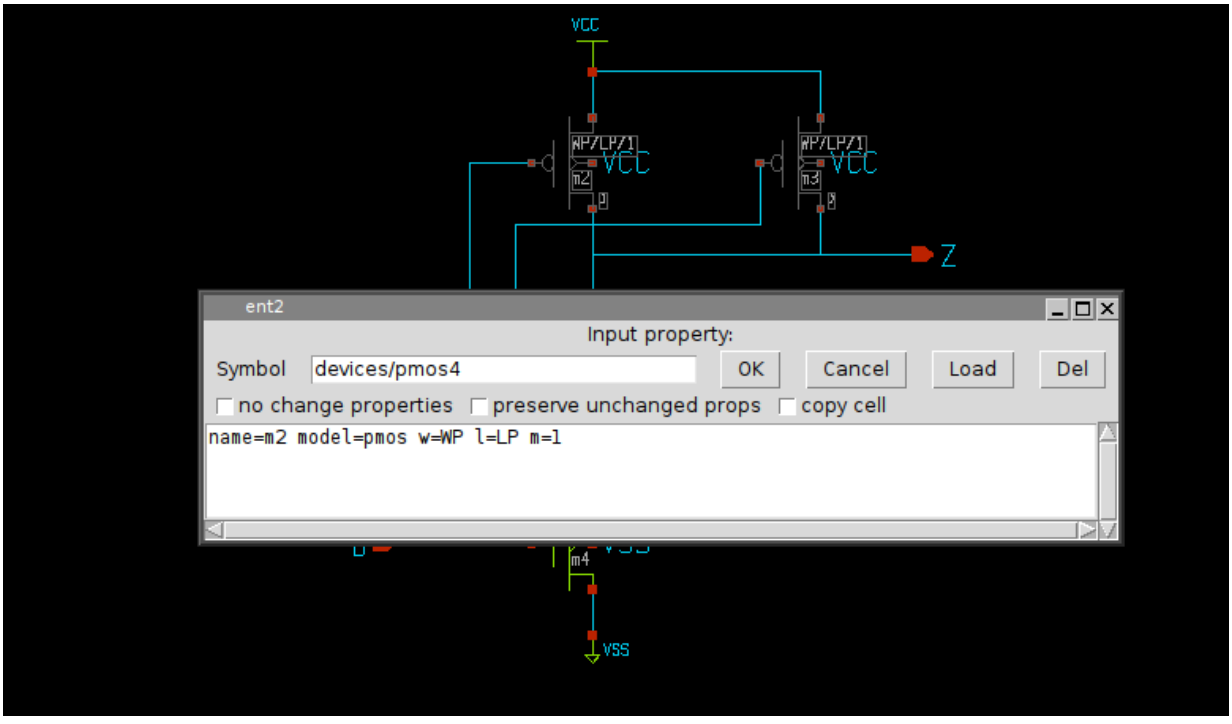


for more info on symbols see the Tutorial

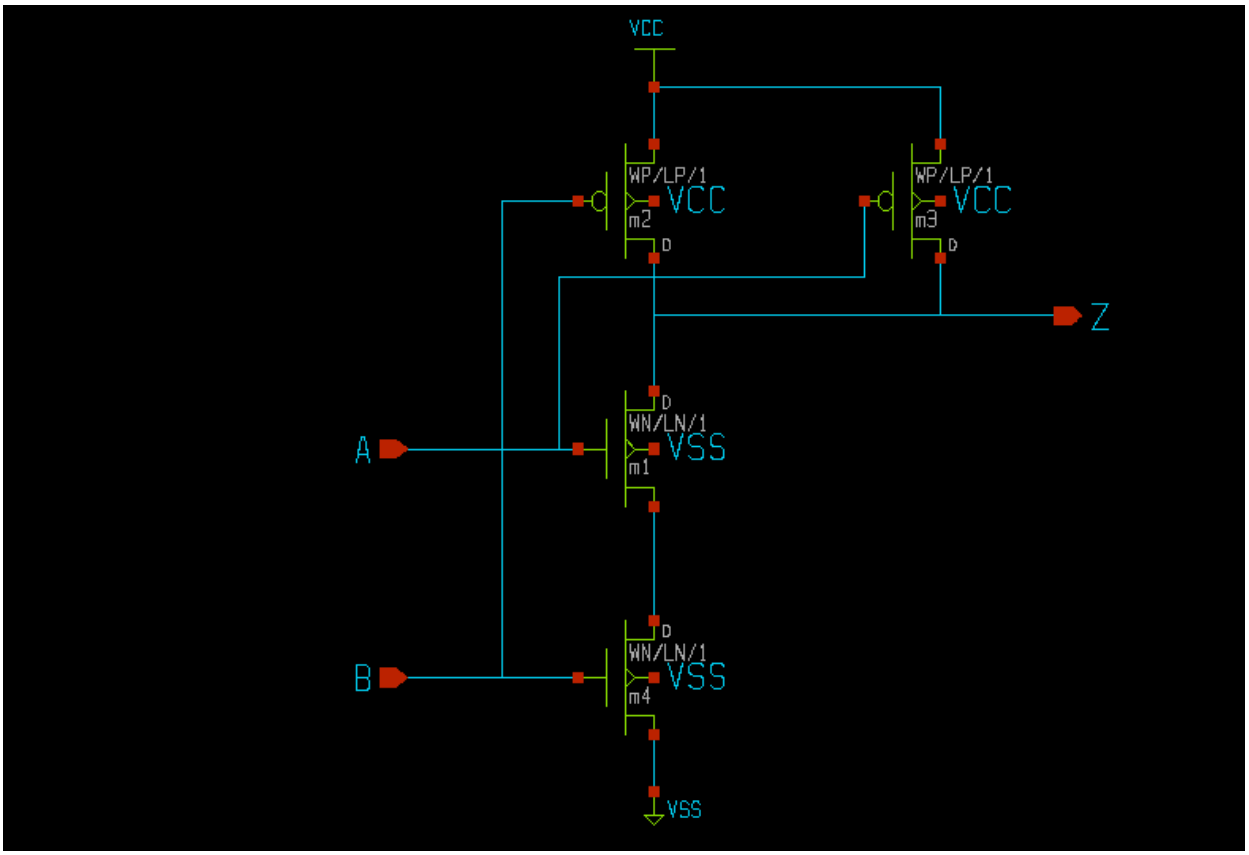
COMPONENT PARAMETERS

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.

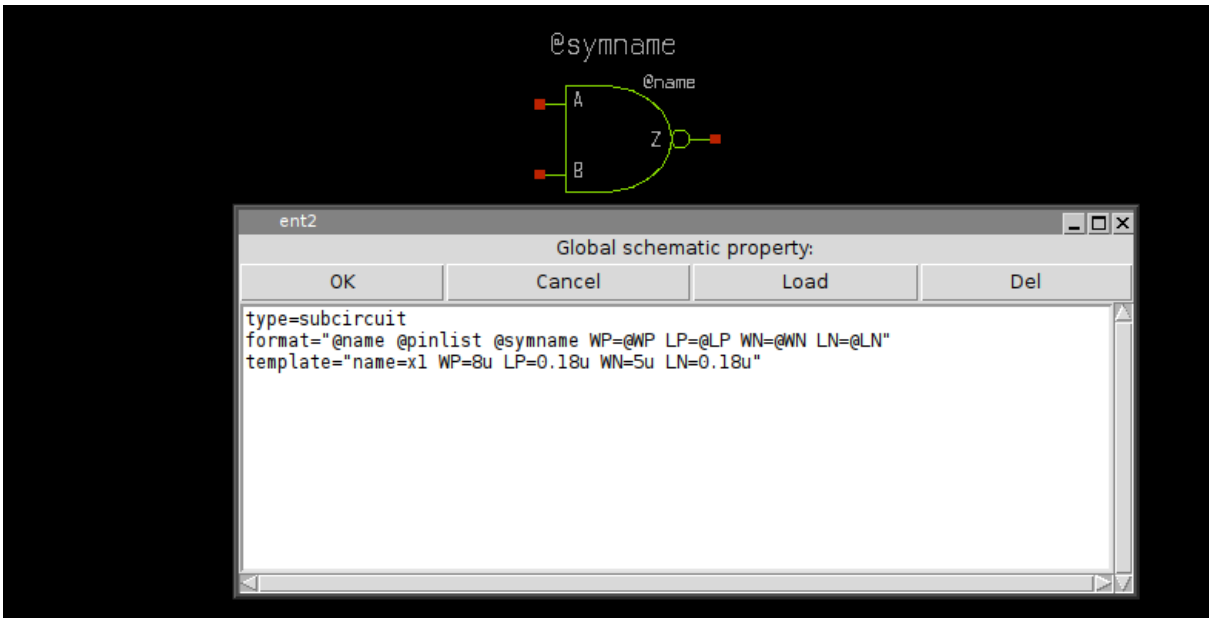
TIP: you can select two PMOS at the same time by clicking the second one with the `shift` key pressed, so with edit property `'q'` key you will change properties for both.



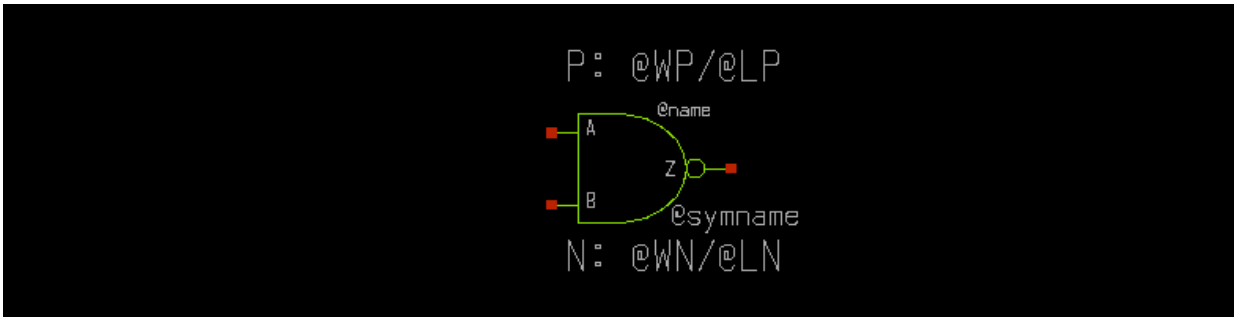
By doing the same for the NMOS transistors we end up with a schematic with fully parametrized transistor geometry.



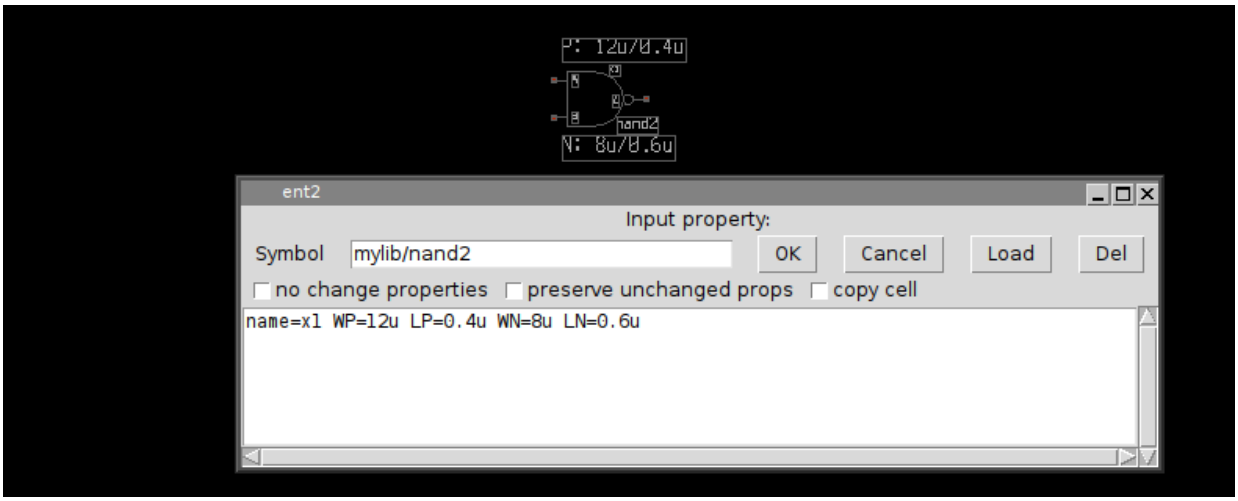
Now we have to change the `mylib/nand2.sym` symbol. Save the changes in the nand2 schematic (`<shift>S`) and load (`Ctrl-o`) the nand2 symbol. without selecting anything hit the `'q'` key to edit the symbol global property string. make the changes as shown in the picture.



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:

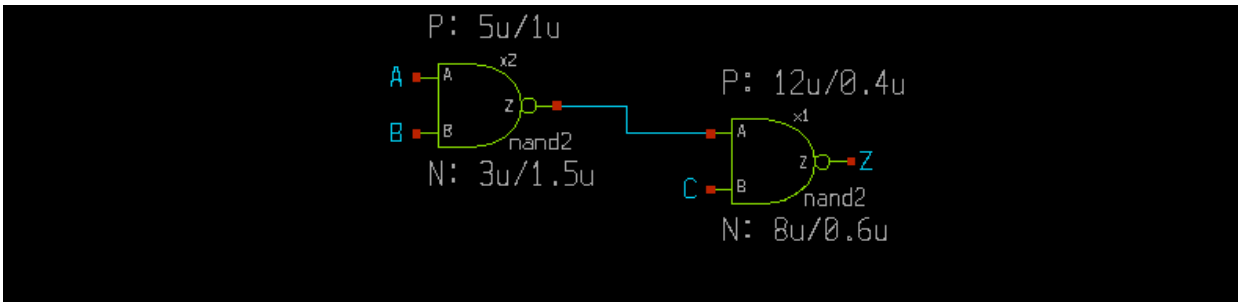


Now close the modified symbol saving the changes. Let's test the placement of the new modified symbol. Start a new schematic (menu `File -> New`) and insert (`Insert key`) the NAND2 gate. by pressing `'q'` you are now able to speciify different values for the geometric parameters:



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.

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

Mouse Wheel             Zoom in / out

MidButton drag          Pan viewable area

Alt + LeftButton        Unselect selected object

Alt + LeftButton drag
                        Unselect objects by area
```

```
RightButton             Edit property of object under the mouse
                        else edit global schematic / symbol property string

Shift + RightButton     Edit property of object under the mouse
                        else edit global schematic / symbol property string
                        using the specified (or default) text editor.

LeftButton Double click Terminate Polygon placement


                        XSCHEM KEY BINDINGS
-------------------------------------------------------------------
-          BackSpace    Back to parent schematic
-          Delete       Delete selected objects
-          Insert       Insert element from library
-          Down         Move down
ctrl       Enter        Confirm closing dialog boxes
-          Escape       Abort, redraw, unselect
-          Left         Move right
-          Right        Move left
-          Up           Move up
-           '!'         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
-           '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
-           '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
ctrl        'e'         Back to parent schematic
-           'e'         Descend to schematic
alt         'e'         Edit selected schematic in a new window
shift+alt   'F'         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       'G'         Double snap factor
-           'g'         Half snap factor
ctrl        'g'         Set snap factor
-           '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
```

```
shift       'J'         Create symbol from pin list
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
-           'j'         Print list of highlighted nets/pins
-           'k'         Hilight selected nets
ctrl+shift  'K'         highlight net passing through elements with
'propagate_to' property set on pins
shift       'K'         Unhilight all nets
ctrl        'k'         Unhilight selected nets
-           'l'         Start line
ctrl        'l'         Make schematic view from selected symbol
ctrl        'o'         Load schematic
-           'm'         Move selected obj.
shift       'N'         Hierarchical netlist
-           'n'         Netlist
Ctrl        'n'         New schematic
Ctrl+Shift  'N'         New symbol
shift       'O'         Toggle Light / Dark colorscheme
ctrl        'o'         Load schematic
-           'p'         Pan
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
-           '='         Toggle fill rectangles
-           '$'         Toggle pixmap  saving
ctrl        '$'         Toggle use XCopyArea vs drawing primitives for
drawing the screen
-           ':'         Toggle flat netlist
```

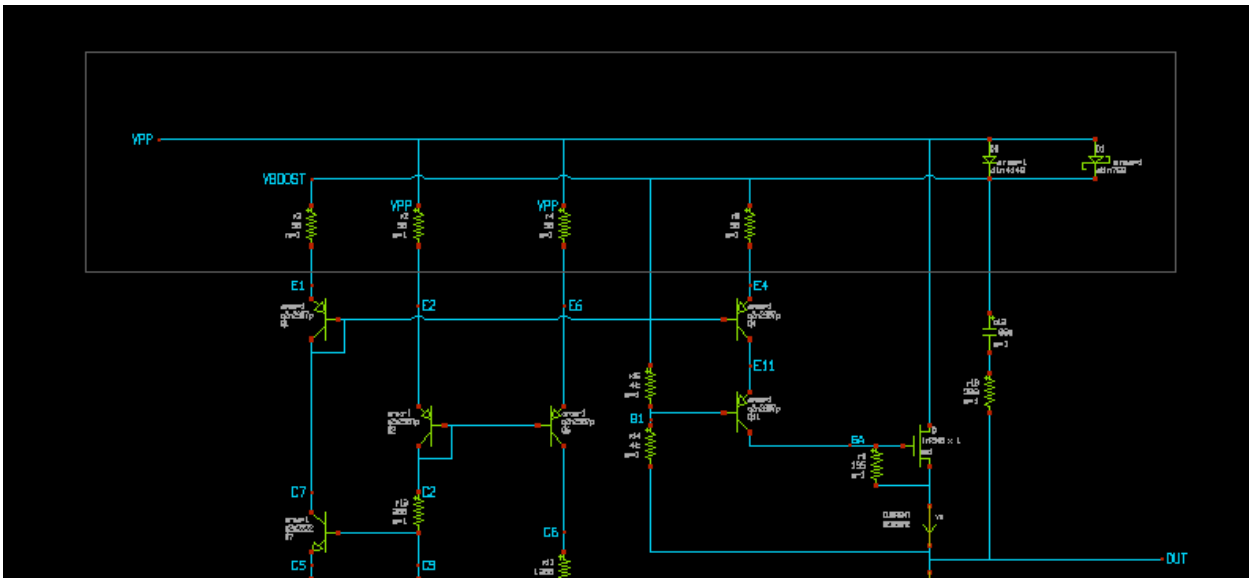
KEYBIND CUSTOMIZATION

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

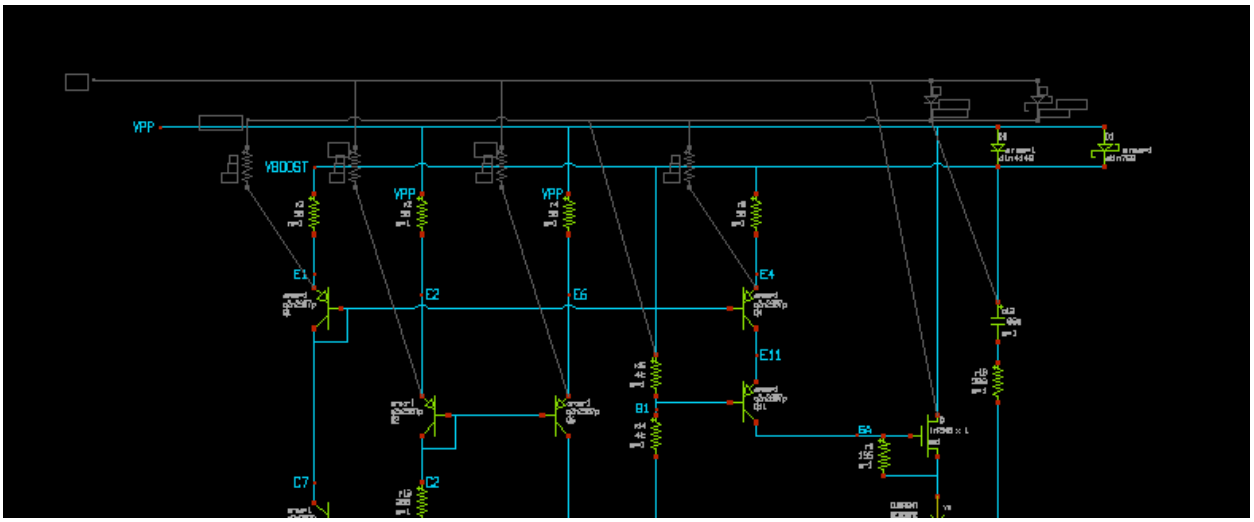
```
## replace Ctrl-d with Escape (so you won't kill the program :-))
set replace_key(Control-d) Escape
## swap w and W keybinds; Always specify Shift for capital letters
set replace_key(Shift-W) w
set replace_key(w) Shift-W
```

STRETCH OPERATIONS

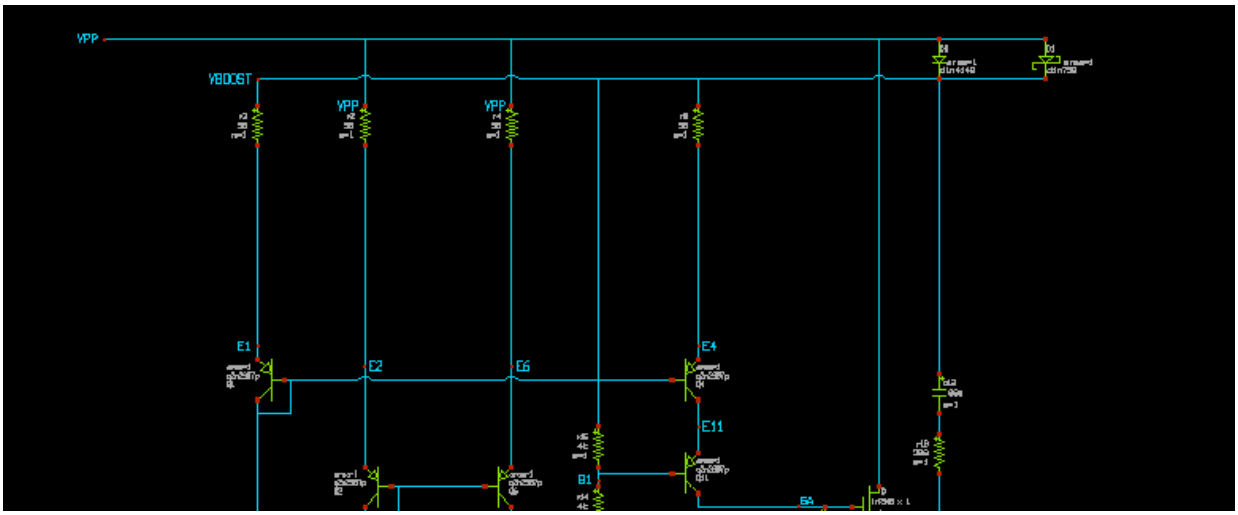
An important operation that deserves a special paragraph is the `Stretch` operation. There is frequently the need to move part of the circuit without breaking connections, for example to create more room for other circuitry or just to make it look better. The first thing to do is to drag a selection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

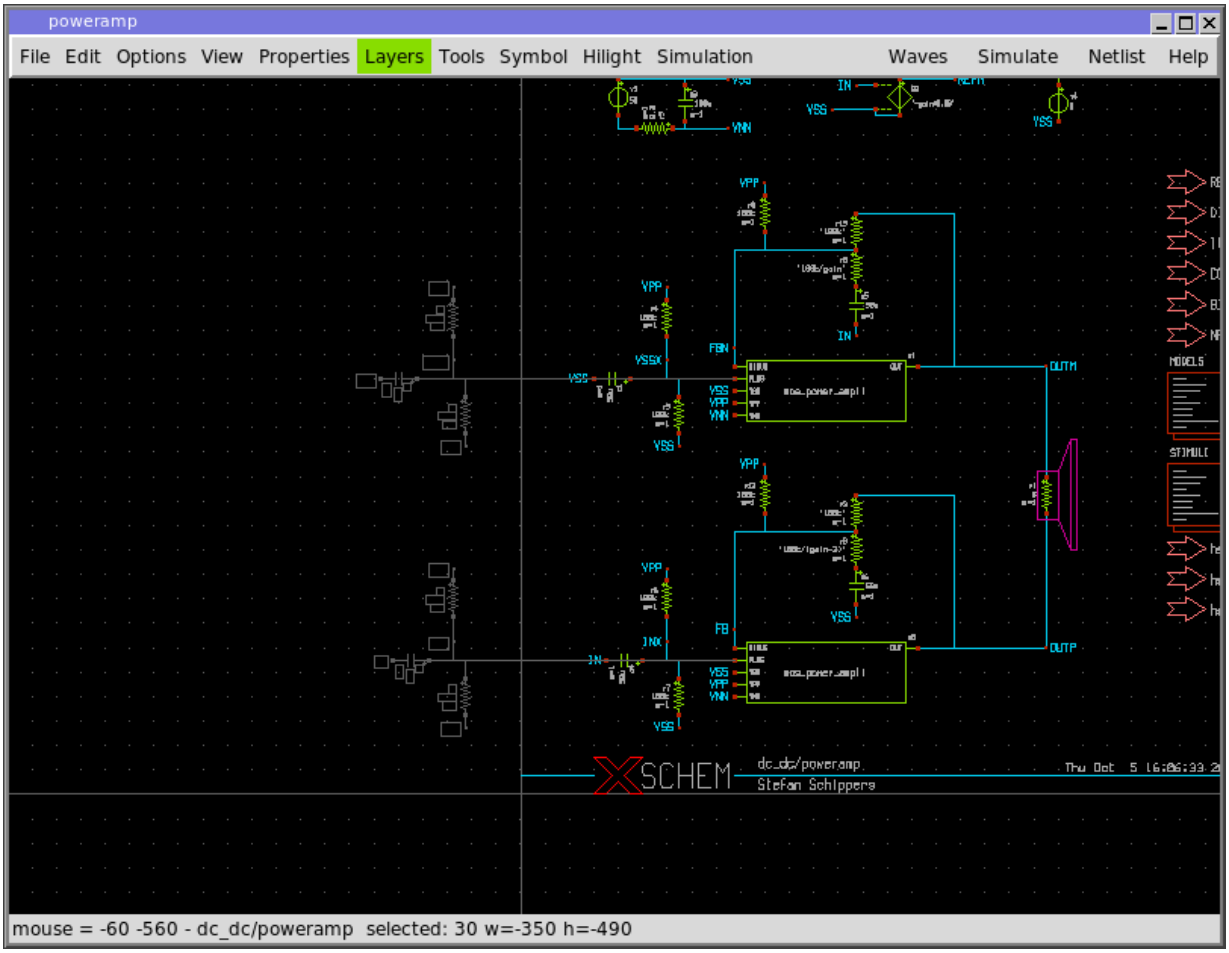


PLACE WIRES SNAPPING TO CLOSEST PIN OT NET ENDPOINT

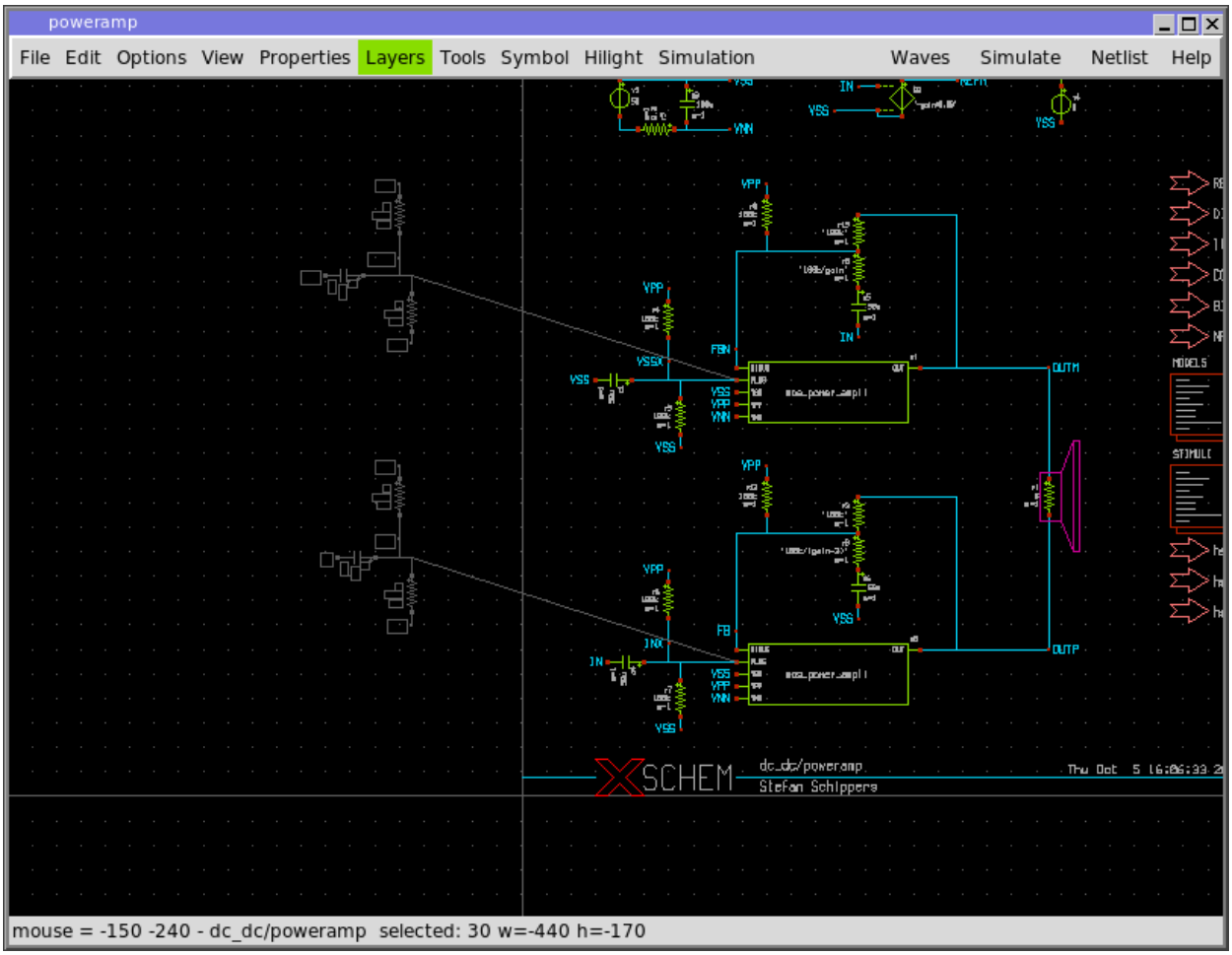
The (uppercase) `'W'` bindkey allows to place a wire putting start (and end point, later) to the closest pin or wire endpoint, this will make it easier to connect precisely without the need to zoom in all times.

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.

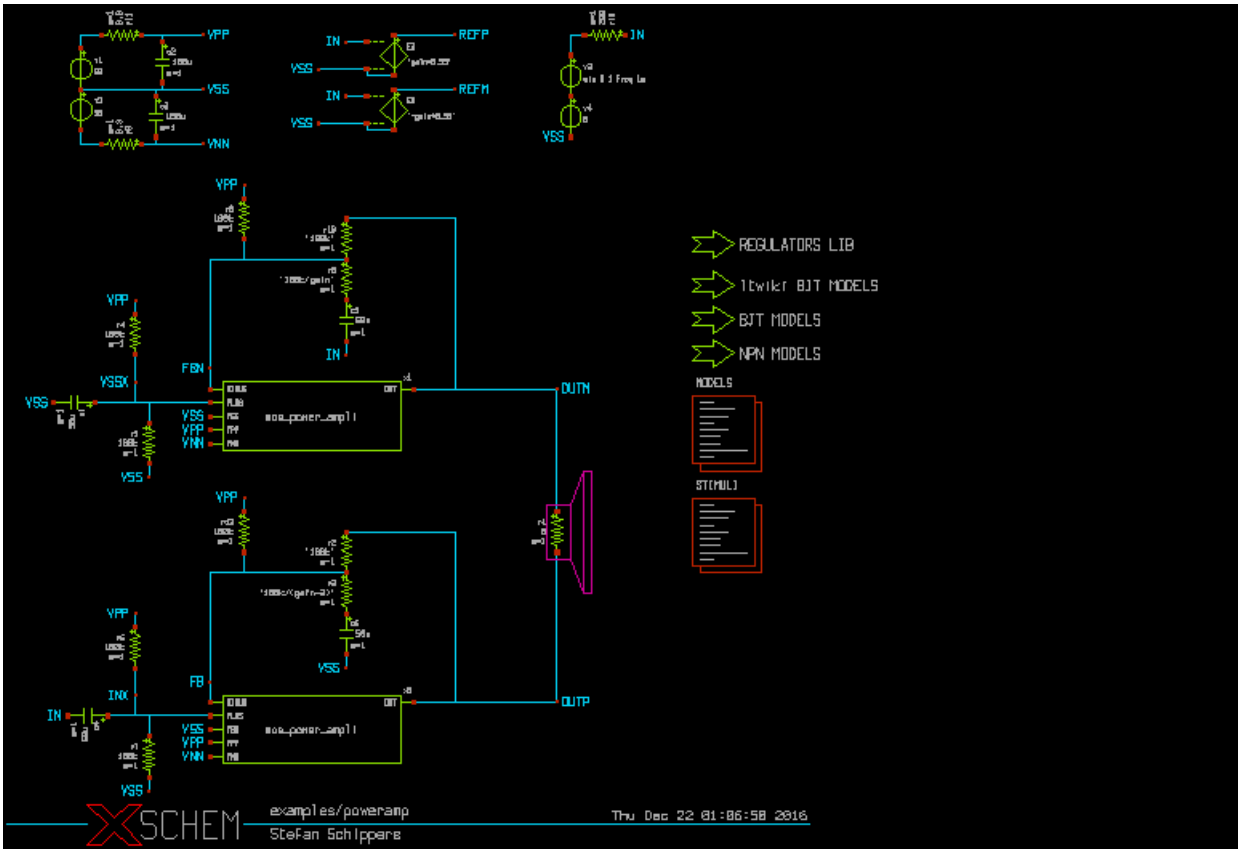
NETLISTING

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

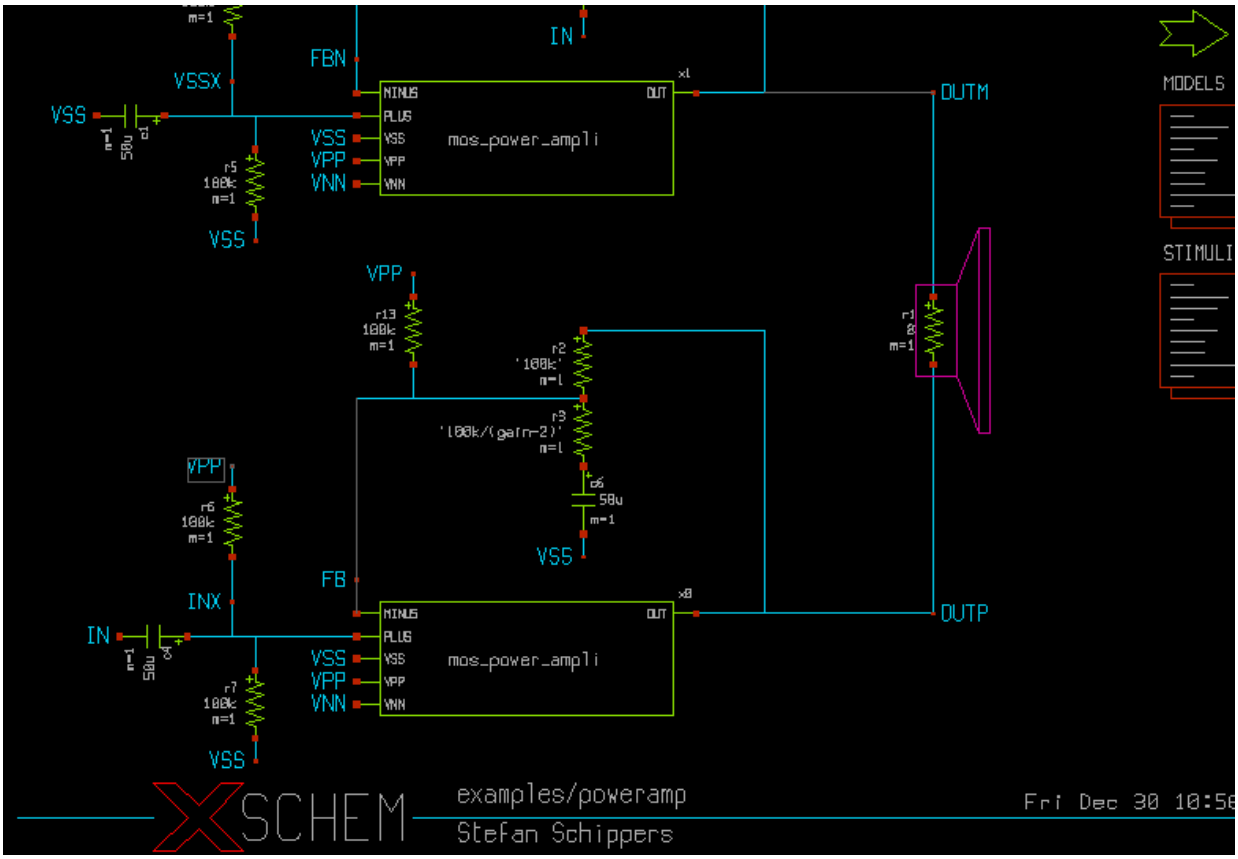
All the concepts explained for SPICE netlist apply for Verilog and VHDL formats. Its up to the designer to ensure that the objects in the schematic are 'known' to the target simulator. For example a resistor is normally not used in VHDL or Verilog designs, so unless an appropriate 'format' attribute 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"
```
```

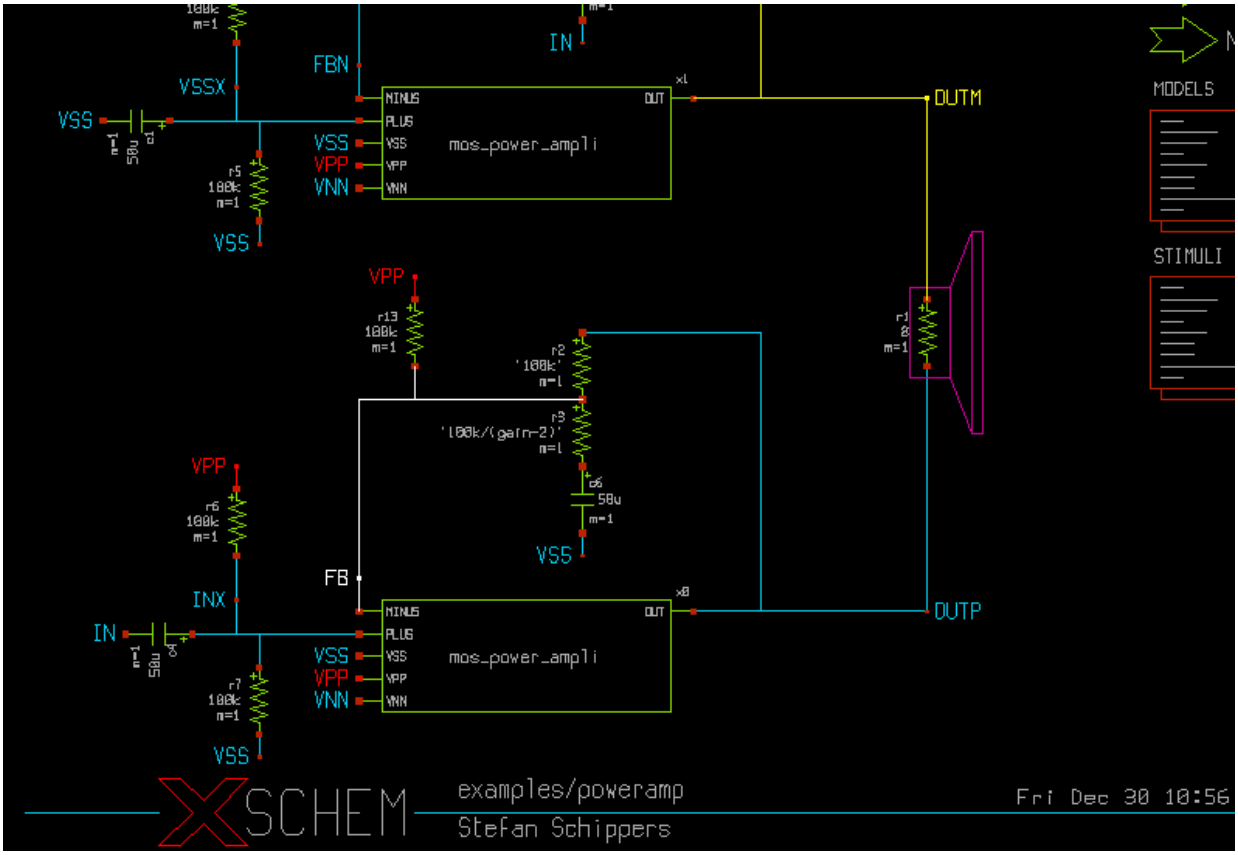
NET PROBES

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.

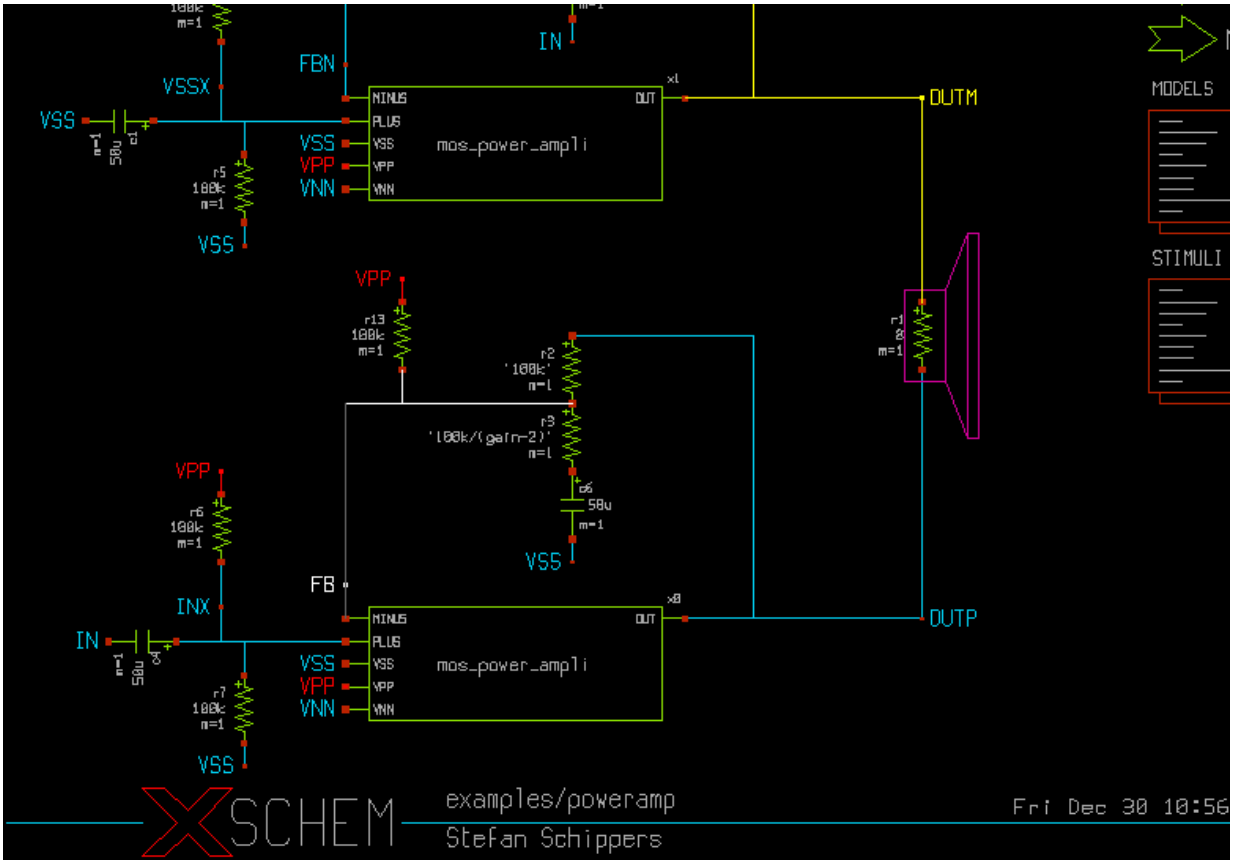
Select some nets...



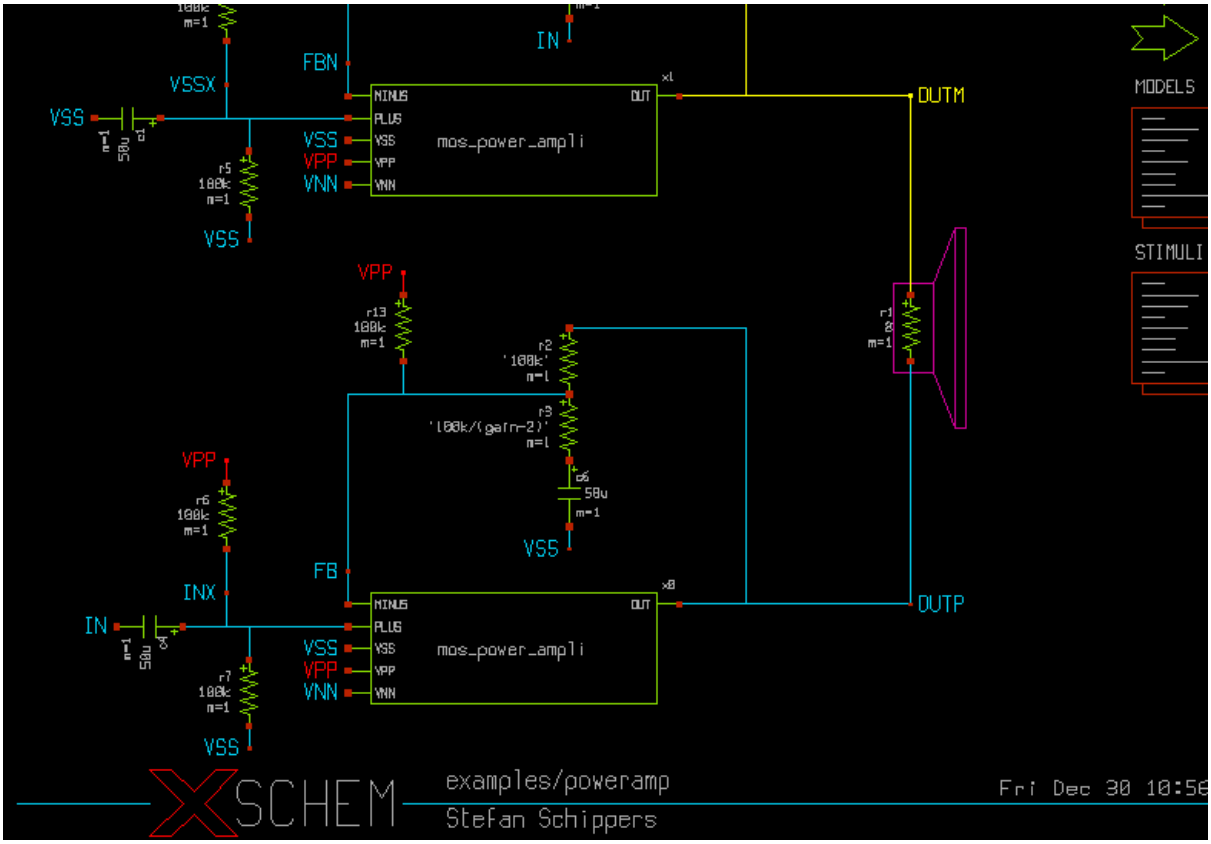
...press the `'k'` key...



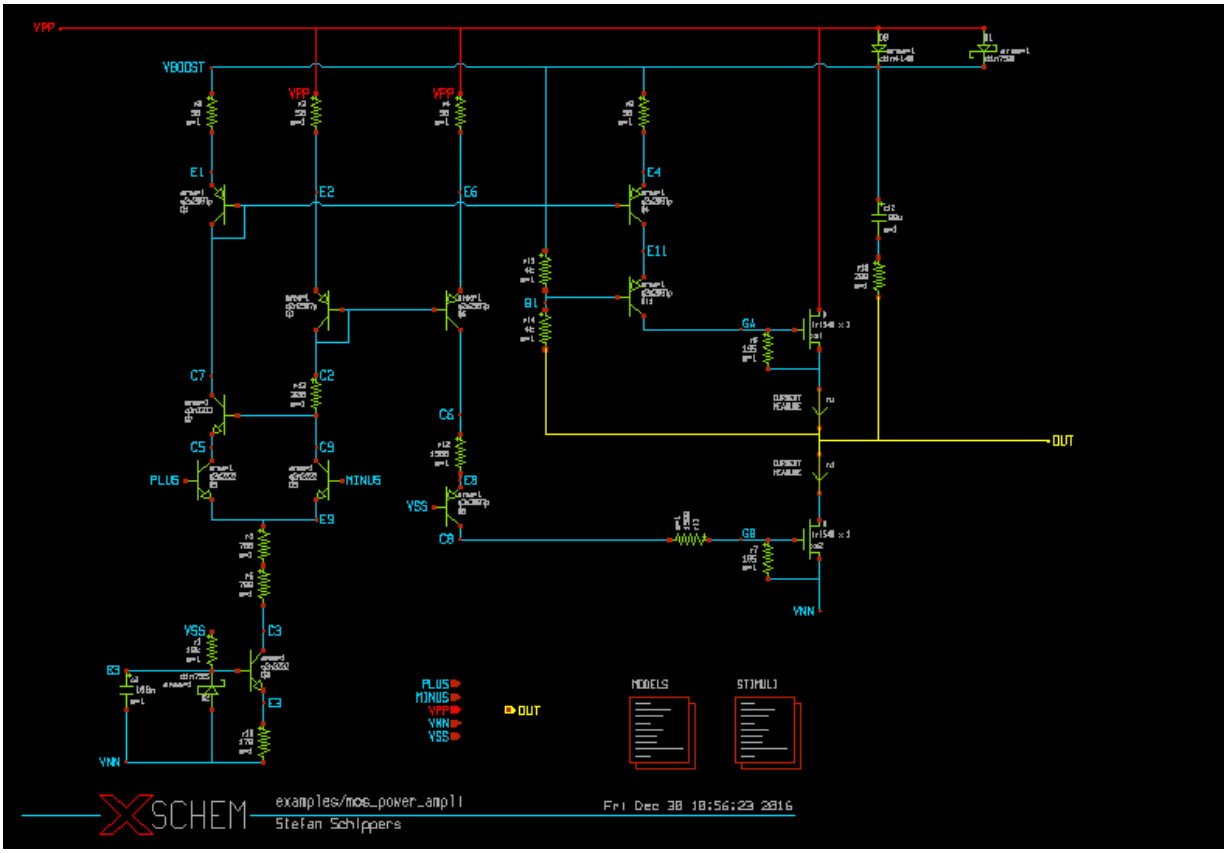
...all nets are highlighted, select the white net...



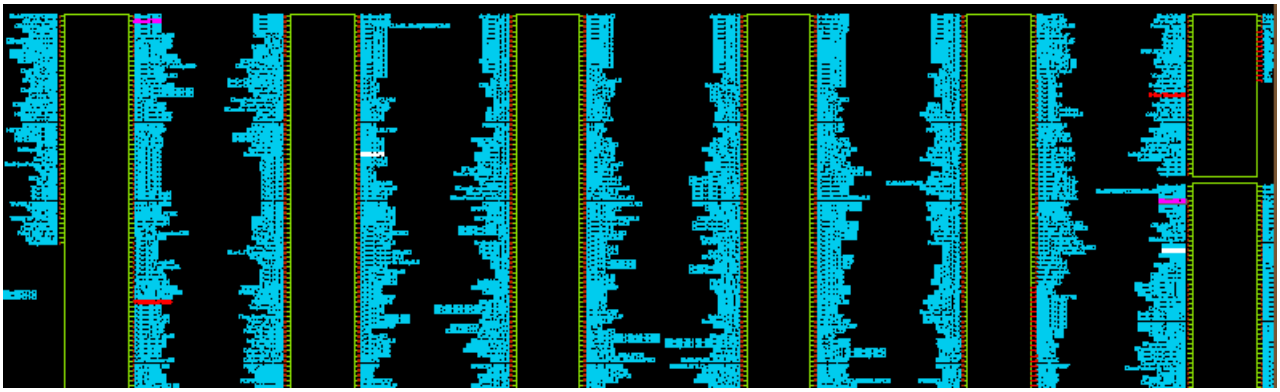
..press the `<Ctrl>k` key and white net is un-highlighted...



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

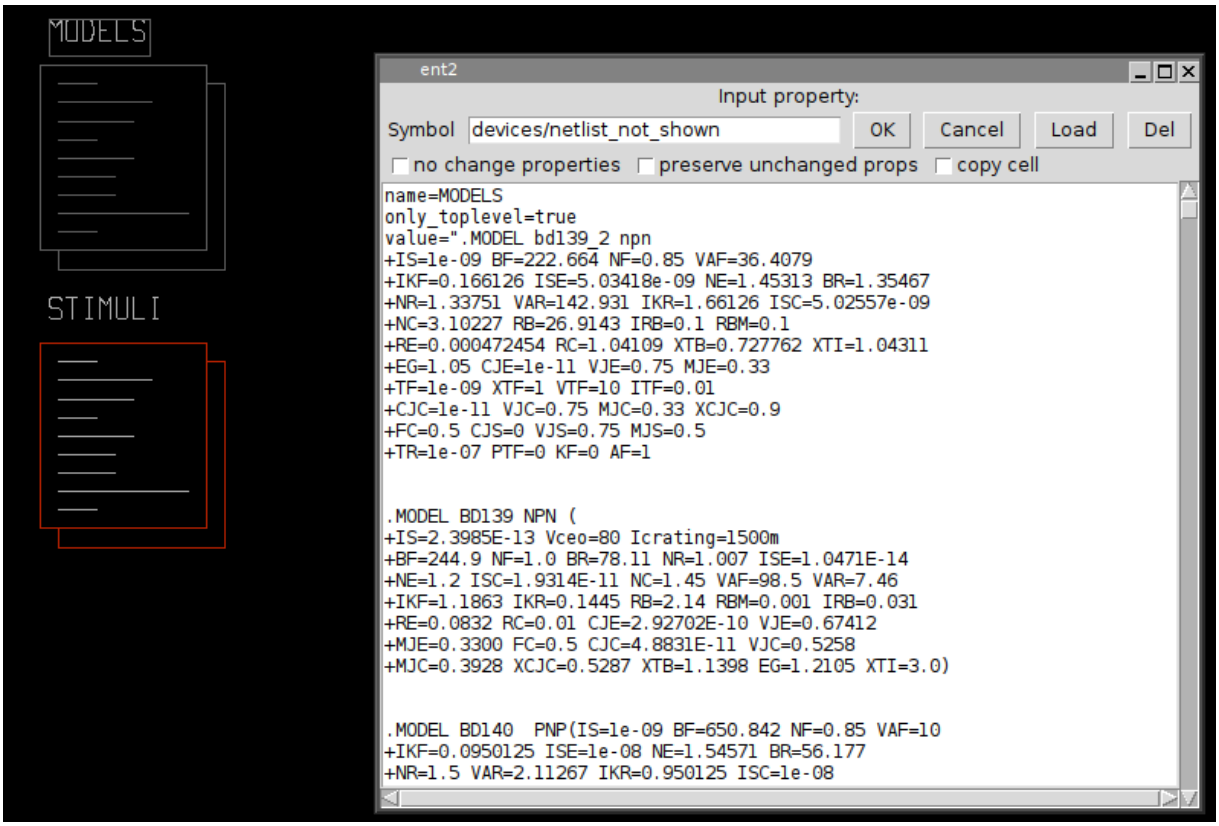


A very useful function is the 'View only probes' mode, (`'5'` key) that hides everything but the highlight probes. This is useful in very big VLSI designs to quickly locate start and end point of nets. Pressing again the `'5'` key restores the normal view.



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

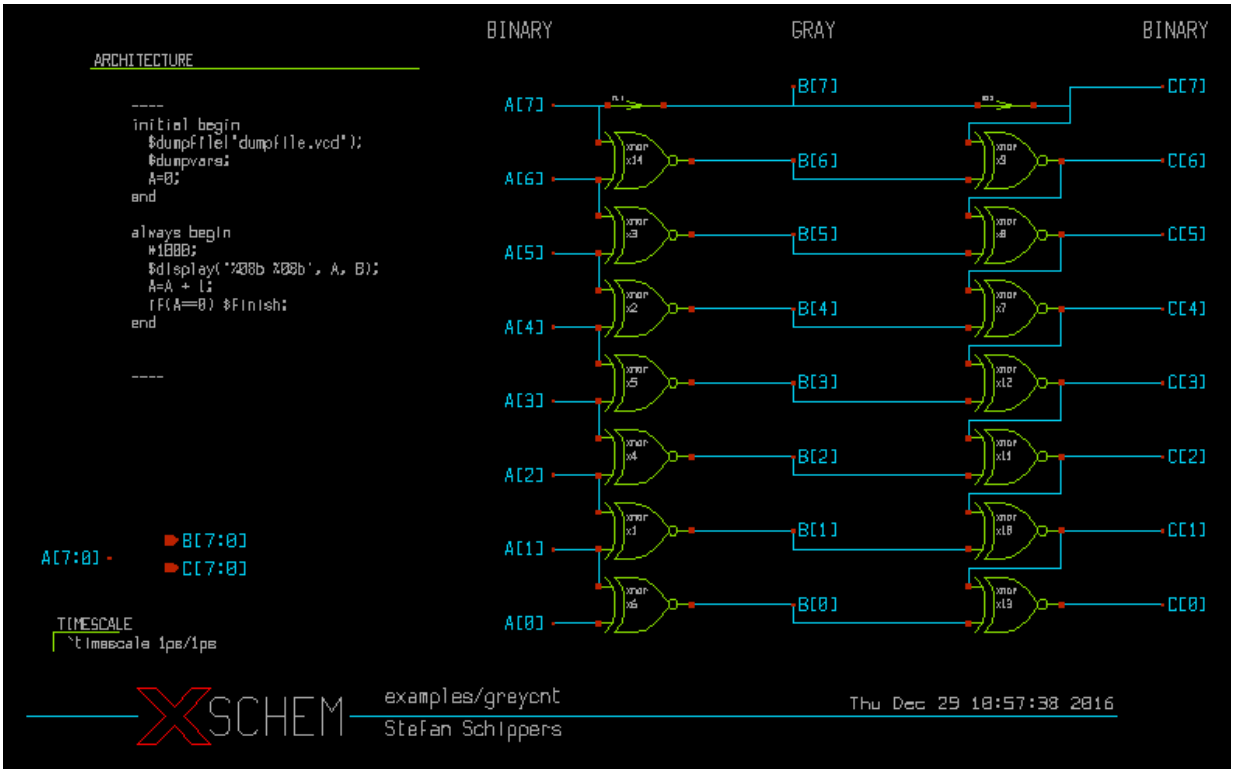
This is a tutorial showing how to run a simulation with XSCHEM. The first important thing to note is that XSCHEM is just a schematic editor, so we need to setup valid bindings to simulators in the `~/xschemrc` file. For this tutorial we plan to do a `Verilog` simulation since there is a very

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 installation paths in `xschemrc`:

```
...
## icarus verilog
set iverilog_path $env(HOME)/verilog/bin/iverilog
set vvp_path $env(HOME)/verilog/bin/vvp
set iverilog_opts {-g2012}
...
## gtkwave
set gtkwave_path $env(HOME)/gtkwave/bin/gtkwave
...
```

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

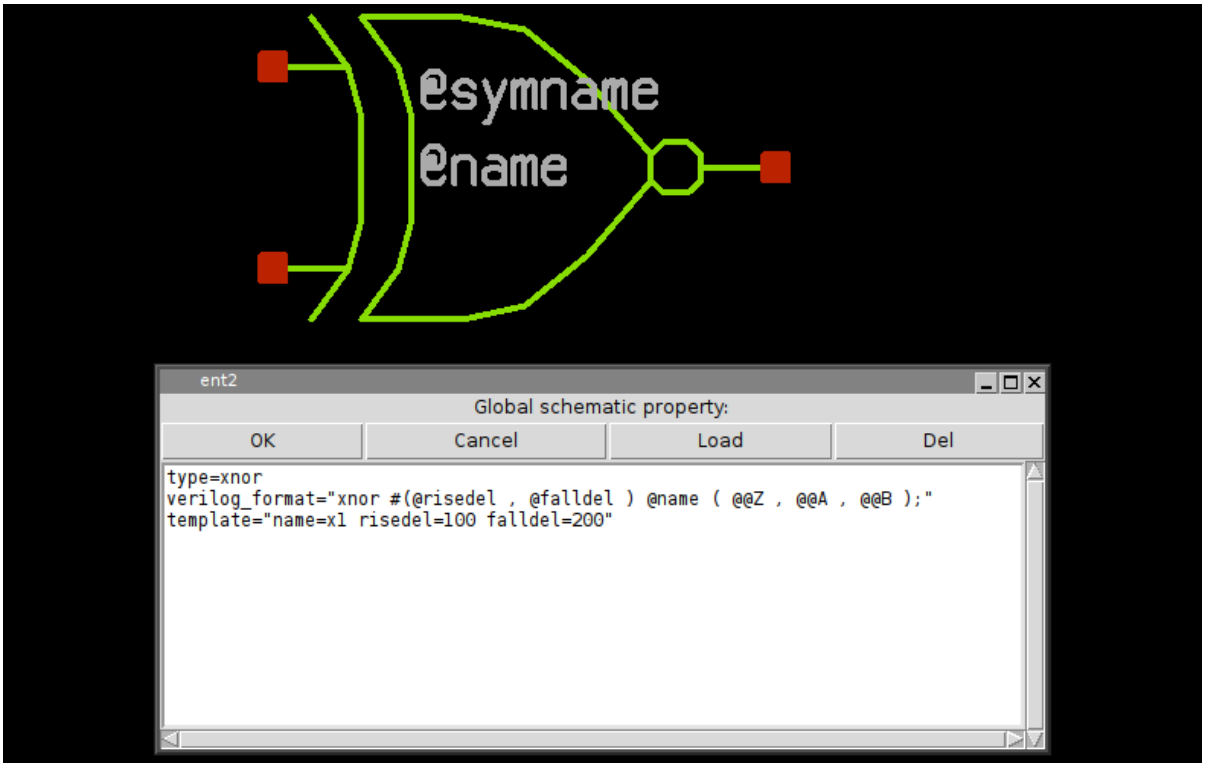
```
user:~$ xschem examples/greycnt
```



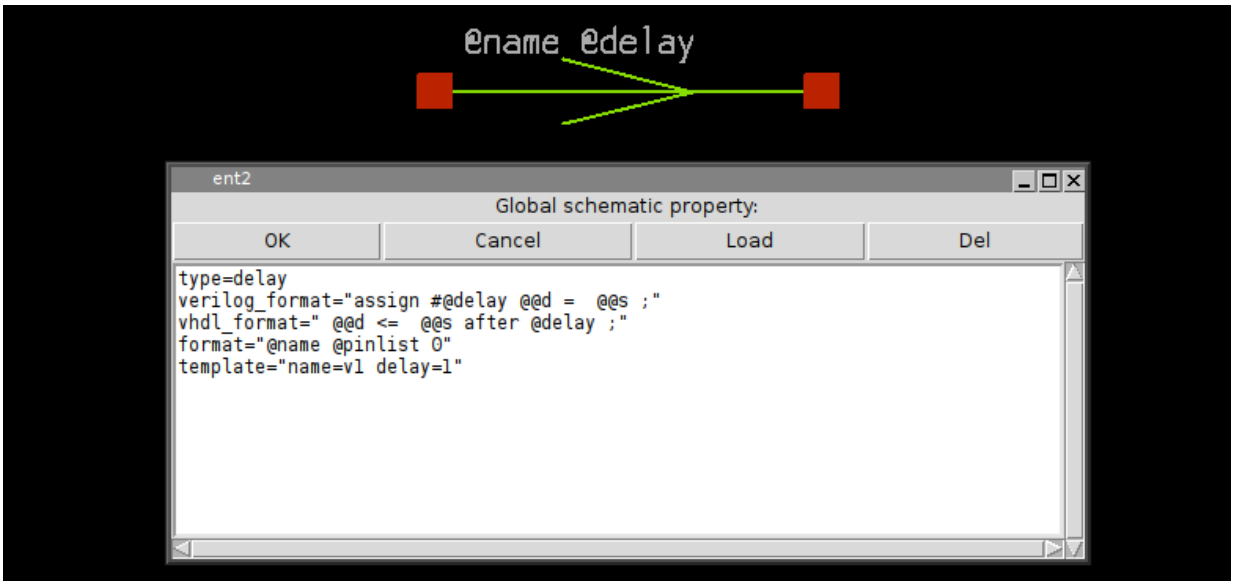
This testbench has a 8 bit input vector A[7:0] and two output vectors, B[7:0] and C[7:0]. B[7:0] is a grey coded vector, this mean that if A[7:0] is incremented as a binary number B[7:0] will increment by changing only one bit at a time. The C[7:0] vector is the reverse transformation from grey-code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

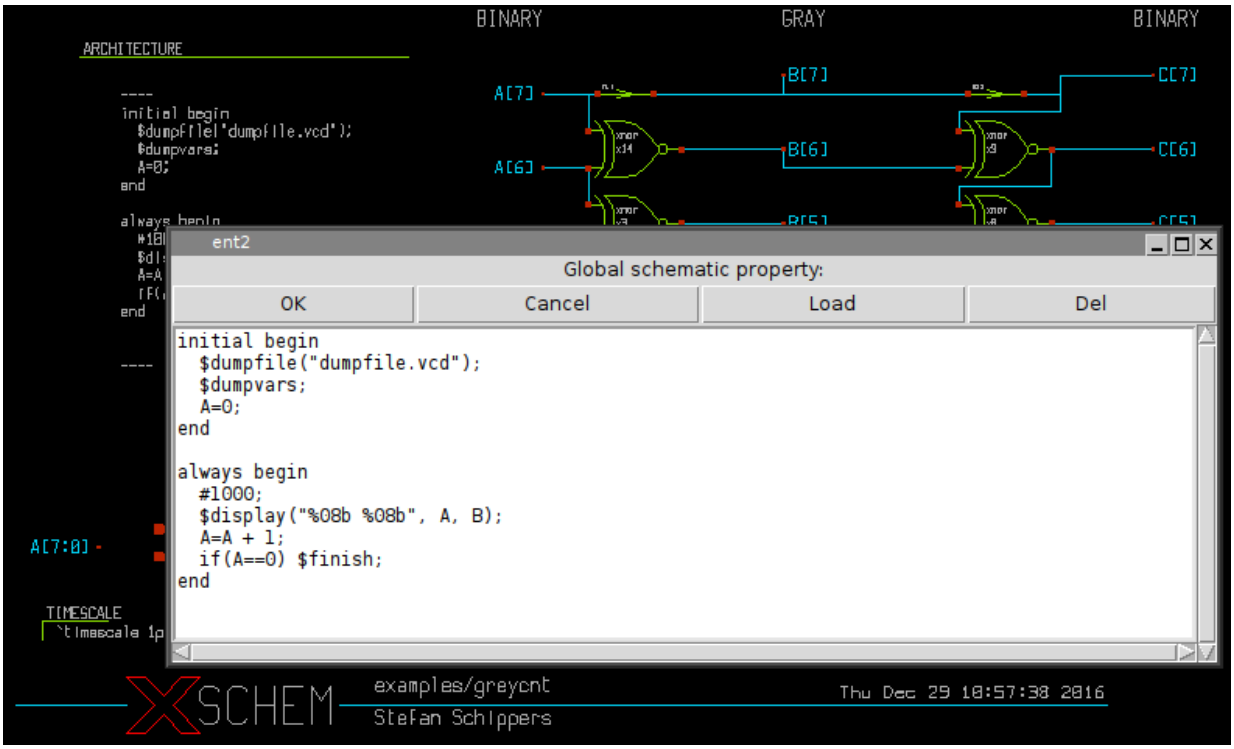
An Ex-Nor gate can be represented as a verilog primitive, so for the xnor gate we just need to setup a `verilog_format` attribute in the global property string of the `xnor.sym` gate:



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 `<Shift>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):

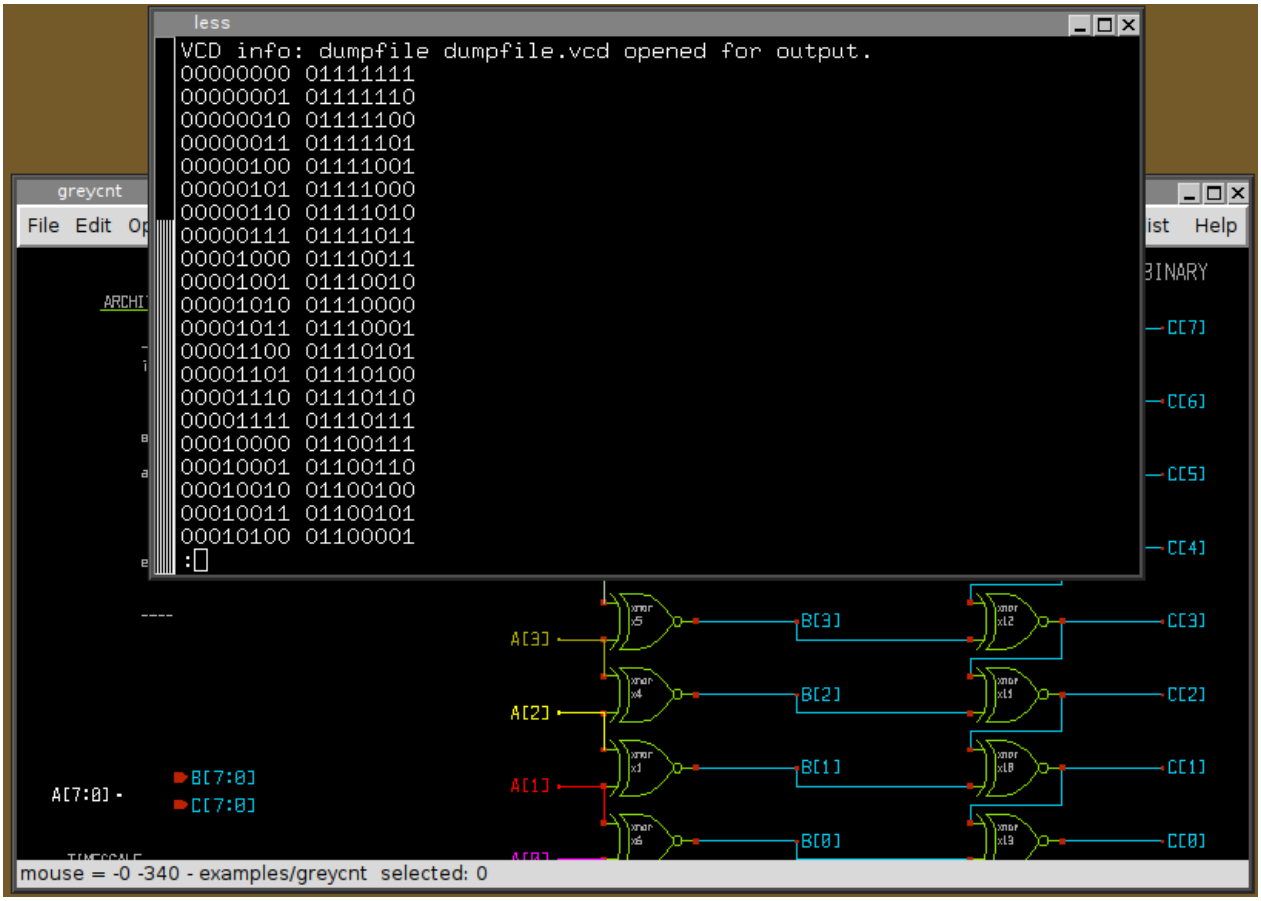
```
`timescale 1ps/1ps
module greycnt (
  output wire [7:0] B,
  output wire [7:0] C
);

reg [7:0] A  ;

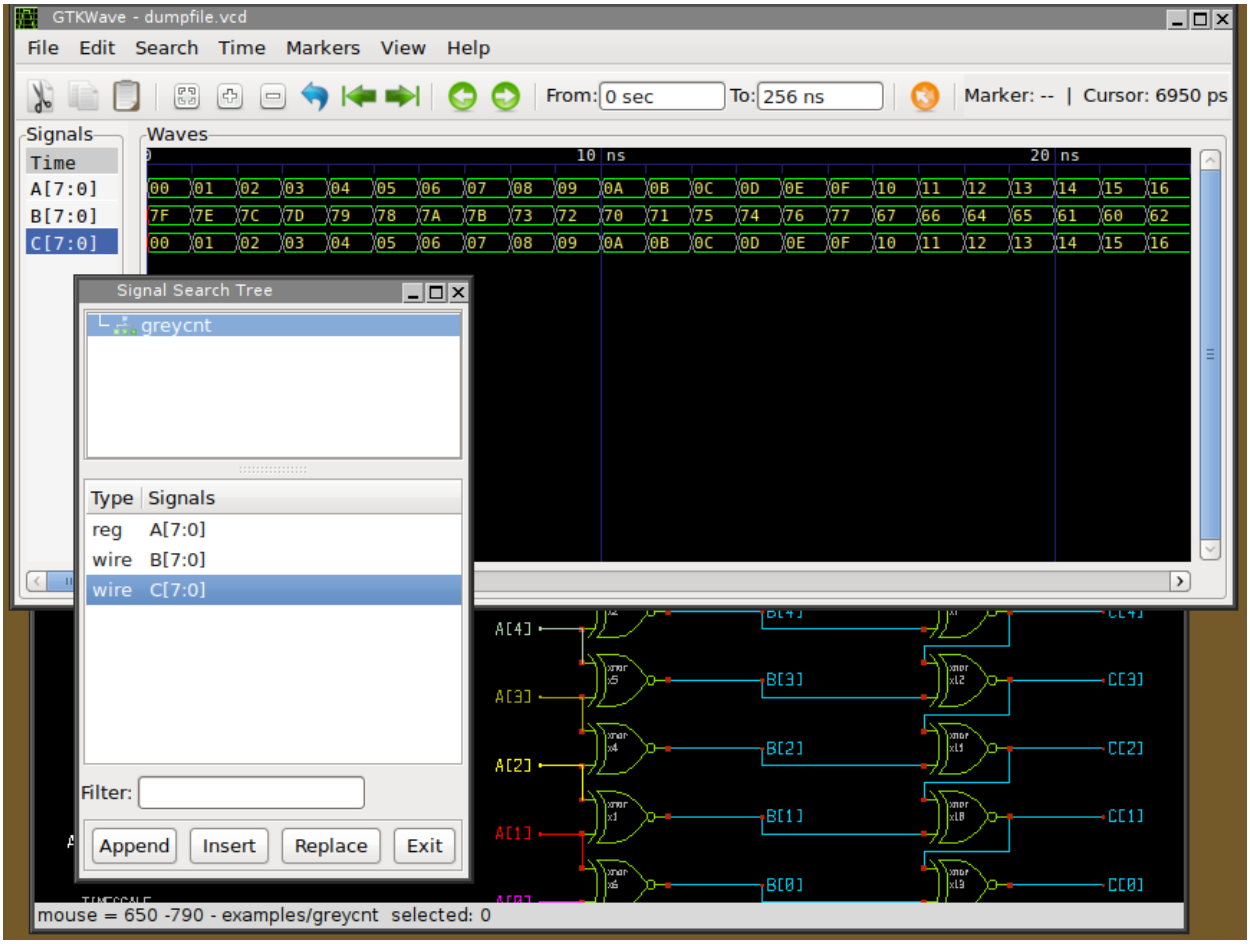
xnor #(1 , 1 ) x2 ( B[4] , A[5] , A[4] );
xnor #(1 , 1 ) x3 ( B[5] , A[6] , A[5] );
xnor #(1 , 1 ) x14 ( B[6] , A[7] , A[6] );
assign #1 B[7] = A[7] ;
xnor #(1 , 1 ) x1 ( B[1] , A[2] , A[1] );
xnor #(1 , 1 ) x4 ( B[2] , A[3] , A[2] );
xnor #(1 , 1 ) x5 ( B[3] , A[4] , A[3] );
xnor #(1 , 1 ) x6 ( B[0] , 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
```

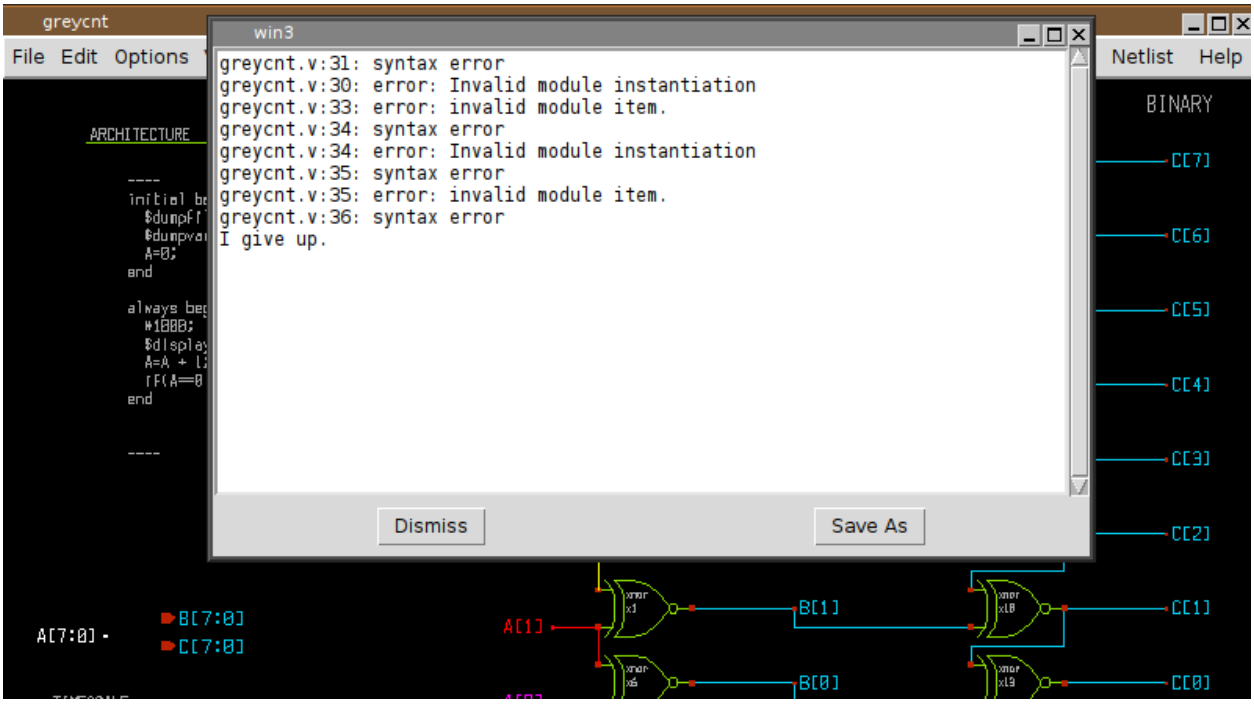
you will recognize the behavioral code right after the netlist specifying the connection of nets to the xnor and assign gates and all the necessary verilog declarations. If you press the `Simulation` button the `Icarus Verilog` simulator will be executed to compile (iverilog) and run (vvp) the simulation,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



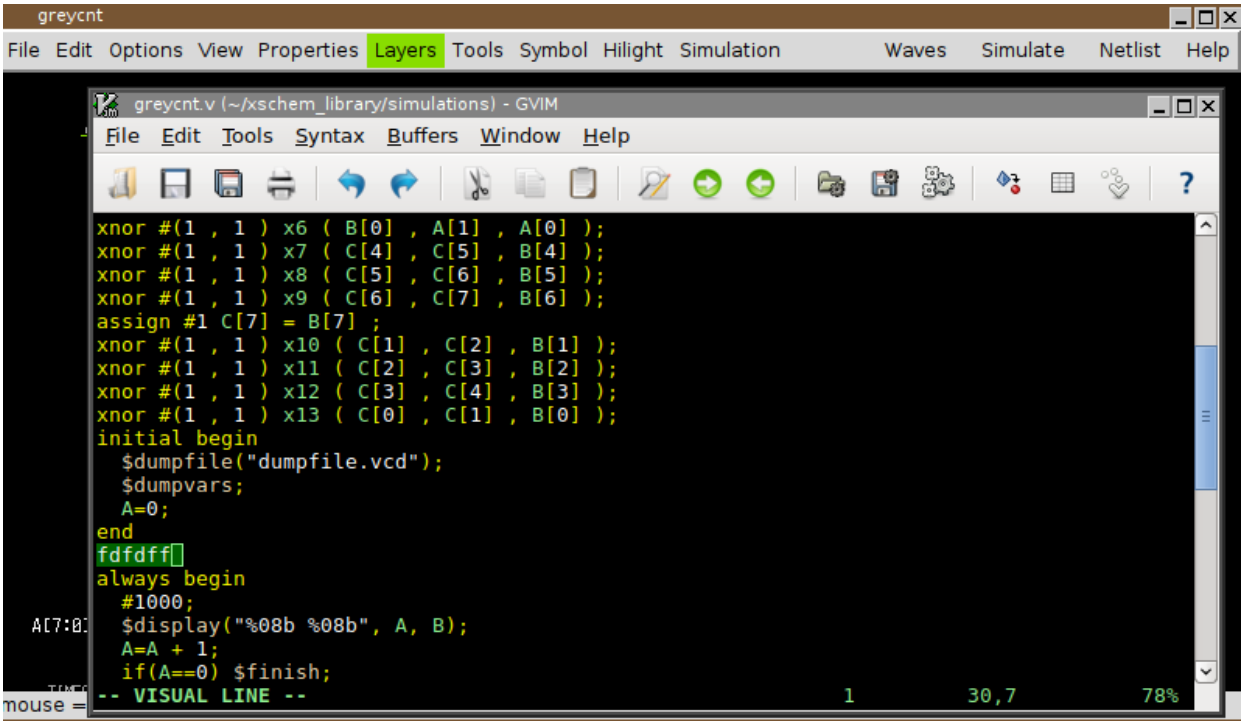
If simulation completes with no errors waveforms can be viewed. Press the `Waves` button in the top-right of the menu bar, you may add waveforms in the gtkwave window:



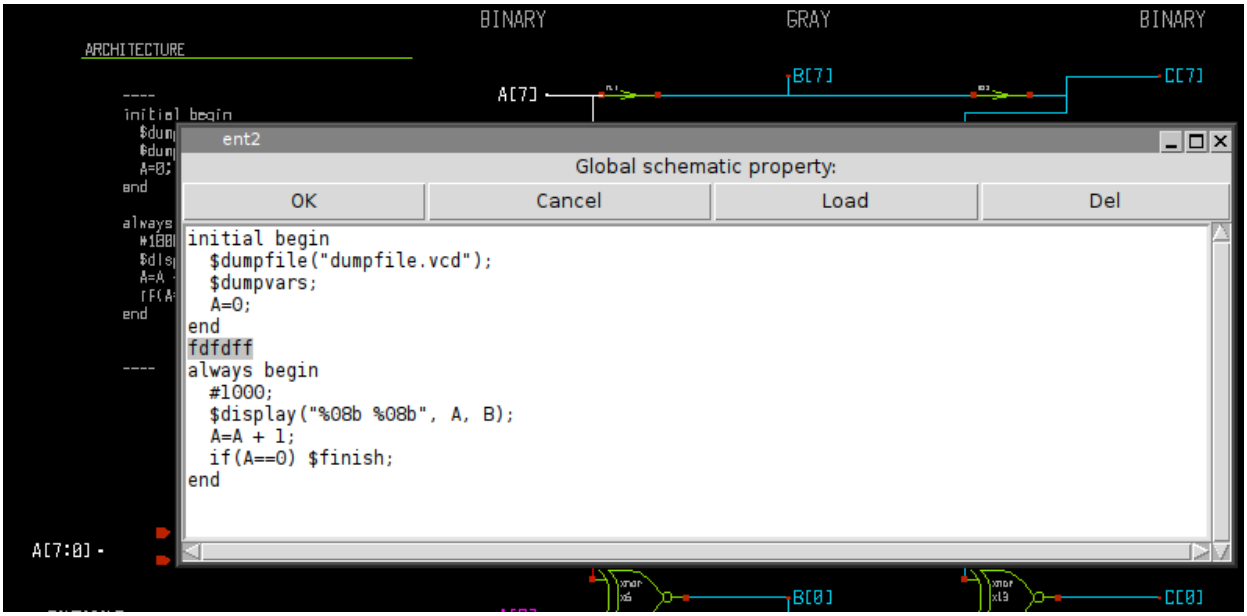
If the schematic contains errors that the simulator can not handle instead of the simulation log a window showing the error messages from the simulator is shown:



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]

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:

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

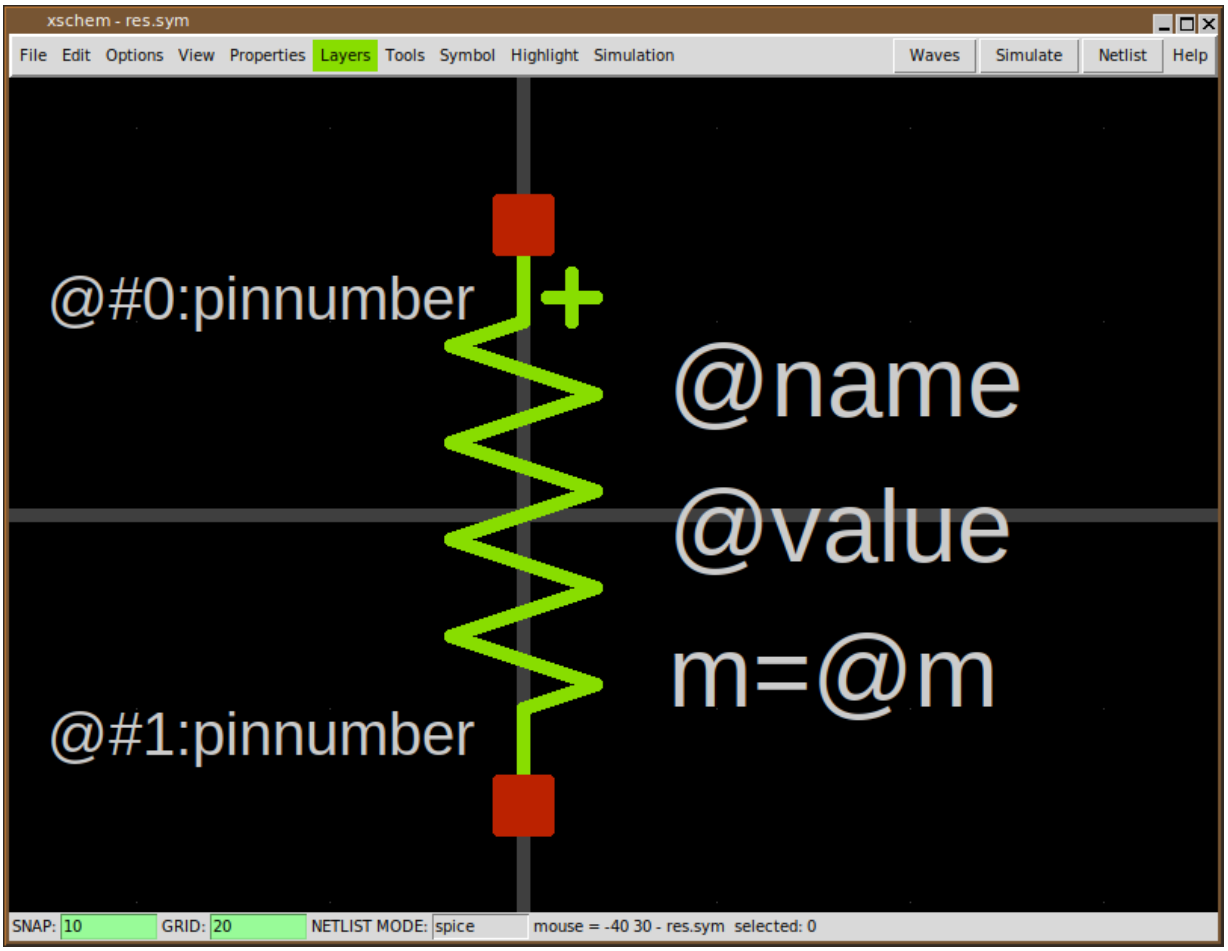
....

20. General purpose
21. General purpose

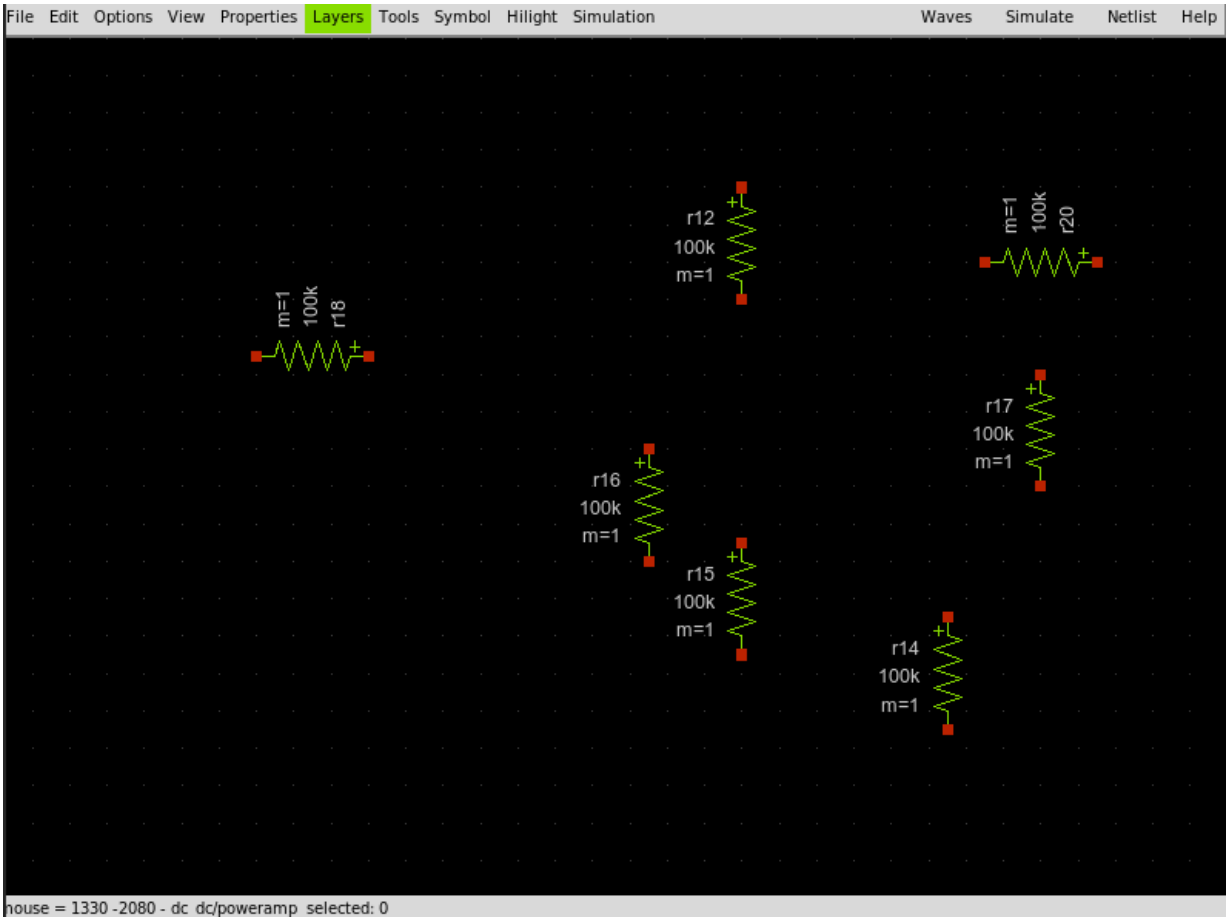
Although any layer can be used for drawing it is strongly advisable to avoid the background color and the selection color to avoid confusion. Drawing begins by painting the background (layer 0), then drawing the grid (layer 1) then drawing wires (nets) on layer 2, then all graphical objects (lines, rectangles,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

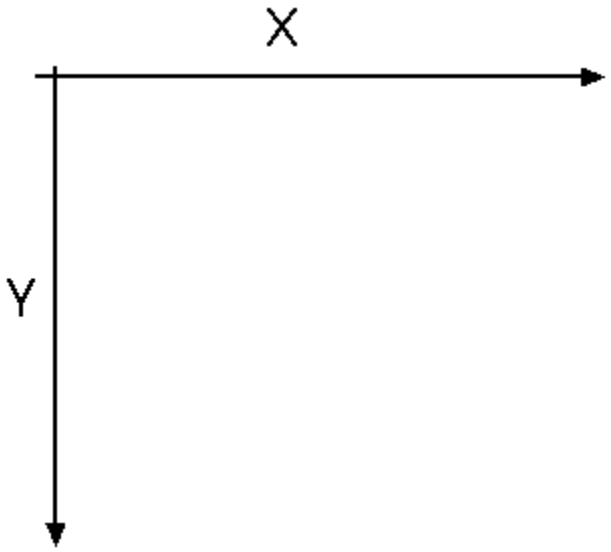
Since wires are used to build the circuit connectivity it is best to avoid drawing lines on layer 1 to avoid confusion, since they would appear like wires, but ignored completely for electrical connectivity.

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). 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 do not contain wires and component instantiations.

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 the .sch/.sym file for SPICE netlisting
- `V` : Global property associated the .sch/.sym file for VERILOG netlisting
- `G` : Global property associated the .sch/.sym file for VHDL netlisting
- `E` : Global property associated the .sch/.sym file for tEDAx netlisting
- `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=2.8.2_RC3 file_version=1.0}
```

Two attributes are defined, the xschem version and the file format version. Current file format version is 1.0. This string is guaranteed to be the first one in XSCHEM .sch and .sym files.

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"}
```

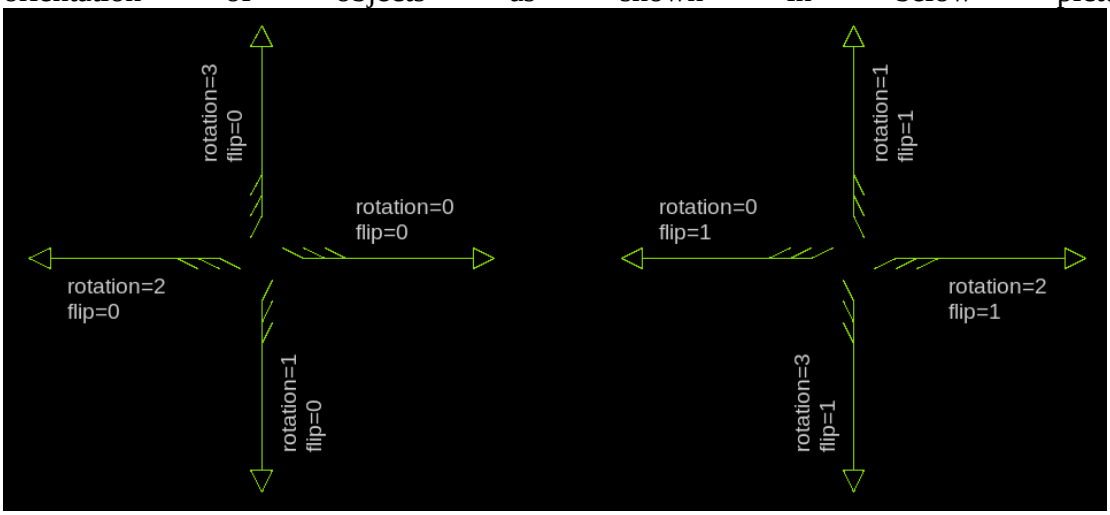
Global properties define a property string bound to the parent schematic/symbol file, there is one global property record per netlisting mode, currently SPICE, VHDL, Verilog, tEDAx.
Normally only 'G' type property strings are used for symbols and define attributes telling netlisters what to do with the 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) 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.

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 objects as shown in below picture:



- 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.

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.

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.

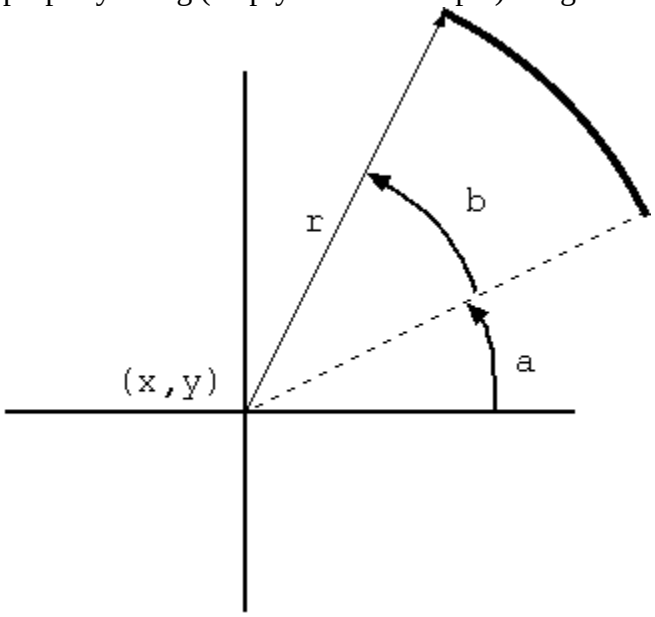
OPEN / CLOSED POLYGON OBJECT

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 cloded 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

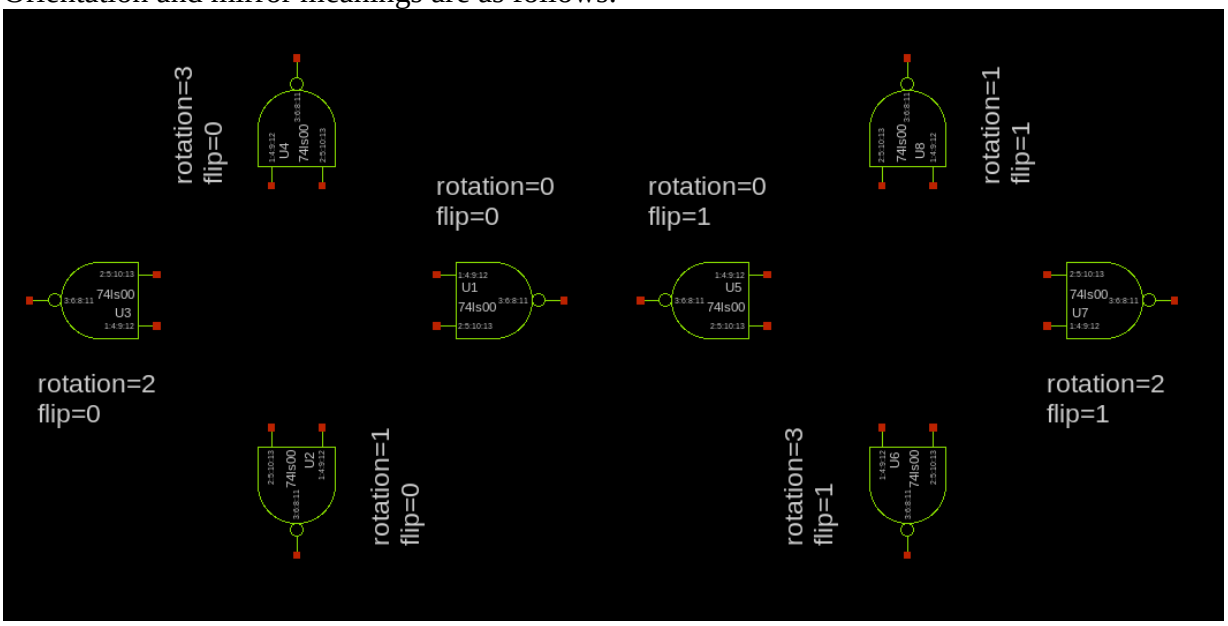
Example: `A 3 450 -210 120 45 225 {}`
The Arc 'A' tag is followed by an integer specifying the layer number, followed by the arc x, y center coordinates, the arc radius, the start angle (measured counterclockwise from the three o'clock direction), the arc sweep angle (measured counterclockwise from the start angle) and the property string (empty in this example). Angles are measured in degrees.



COMPONENT INSTANCE

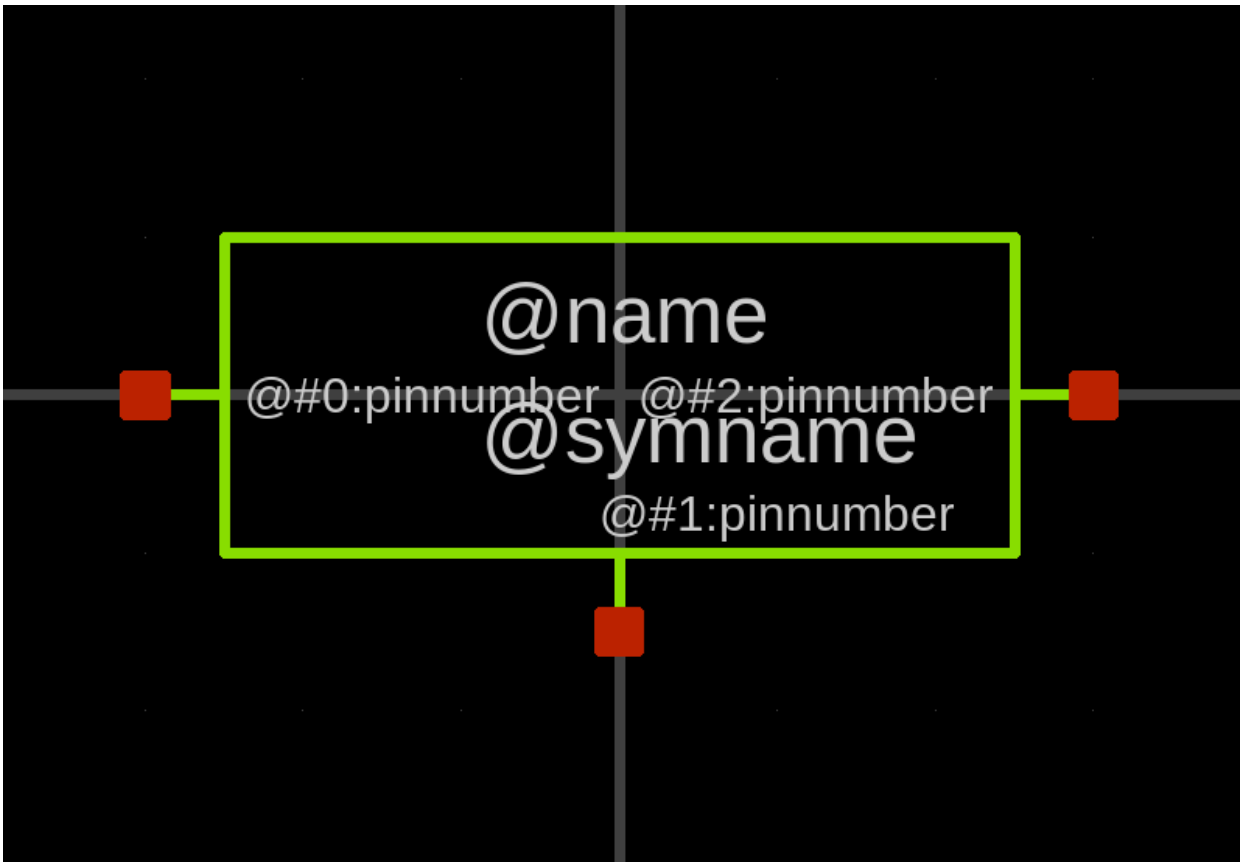
Example: `C {pcb/capa} 890 -160 0 0 {name=C4 m=1 value=10u device="tantalium capacitor"}`
Format: `C {<symbol reference>} <X coord> <Y coord> <rotation> <flip> {<attributes>}`
The component instance tag C is followed by a string specifying `library/symbol`, followed by the x,y coordinates, rotation (integer 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 {}
```



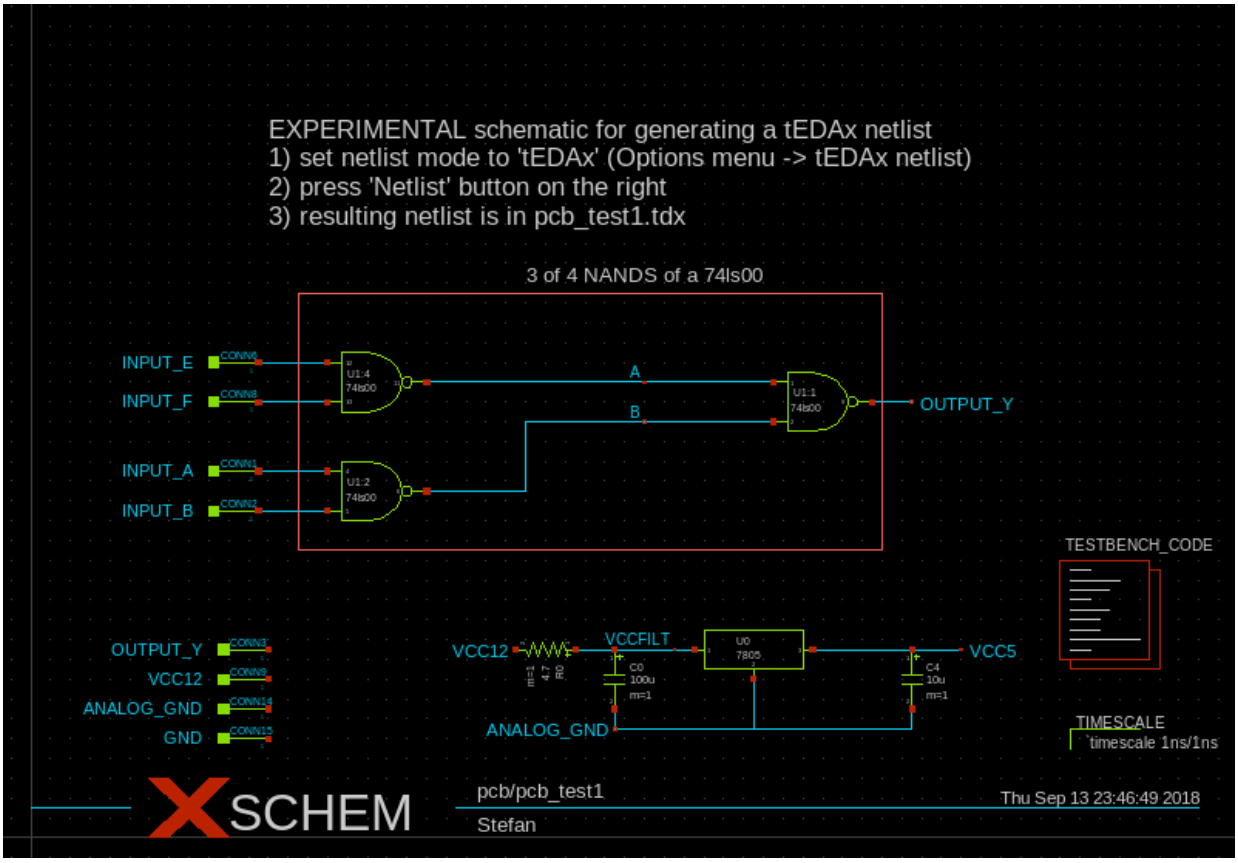
EXAMPLE OF A COMPLETE SCHEMATIC FILE (pcb_test1.sch)

```
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 {devices/title} 160 -30 0 0 {name=l2 author="Stefan"}
C {pcb/74ls00} 340 -350 0 0 {name=U1:2  risedel=100 falldel=200}
C {pcb/74ls00} 790 -440 0 0 {name=U1:1  risedel=100 falldel=200}
C {devices/lab_pin} 890 -440 0 1 {name=p0 lab=OUTPUT_Y}
C {pcb/capa} 590 -160 0 0 {name=C0 m=1 value=100u device="electrolitic
capacitor"}
C {pcb/74ls00} 340 -460 0 0 {name=U1:4 risedel=100 falldel=200 power=VCC5
url="http://www.engrcs.com/components/74LS00.pdf"}
C {pcb/7805} 730 -190 0 0 {name=U0
url="https://www.sparkfun.com/datasheets/Components/LM7805.pdf"}
C {devices/lab_pin} 490 -190 0 0 {name=p20 lab=VCC12}
C {devices/lab_pin} 940 -190 0 1 {name=p22 lab=VCC5}
C {devices/lab_pin} 590 -110 0 0 {name=p23 lab=ANALOG_GND}
C {pcb/capa} 890 -160 0 0 {name=C4 m=1 value=10u device="tantalium
capacitor"}
C {pcb/res} 520 -190 1 0 {name=R0 m=1 value=4.7 device="carbon resistor"}
C {devices/lab_wire} 620 -460 0 0 {name=l3 lab=A}
C {devices/lab_wire} 620 -420 0 0 {name=l0 lab=B}
C {devices/lab_wire} 650 -190 0 0 {name=l1 lab=VCCFILT}
C {pcb/connector} 230 -370 0 0 {name=CONN1 lab=INPUT_A verilog_type=reg}
C {pcb/connector} 230 -330 0 0 {name=CONN2 lab=INPUT_B verilog_type=reg}
C {pcb/connector} 240 -190 0 0 { name=CONN3 lab=OUTPUT_Y }
C {pcb/connector} 230 -480 0 0 {name=CONN6 lab=INPUT_E verilog_type=reg}
C {pcb/connector} 230 -440 0 0 {name=CONN8 lab=INPUT_F verilog_type=reg}
C {pcb/connector} 240 -160 0 0 { name=CONN9 lab=VCC12 }
C {pcb/connector} 240 -130 0 0 { name=CONN14 lab=ANALOG_GND
verilog_type=reg}
C {pcb/connector} 240 -100 0 0 { name=CONN15 lab=GND  verilog_type=reg}
C {devices/code} 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 {devices/verilog_timescale} 1050 -100 0 0 {name=s1 timestep="1ns"
precision="1ns" }
```



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. Since I/O is simply done through stdin/stdout this allows XSCHEM to be controlled by another application, by piping XSCHEM commands to XSCHEM stdin and getting XSCHEM answers piped from XSCHEM stdout.

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.

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
```

4. Create a build directory (here i use ~/build, choose whatever name you like):

```
schippes@mazinga:~$ mkdir build
```

6. Go into the build directory

```
schippes@mazinga:~$ cd build
```

8. Checkout xschem from the svn repository:

```
schippes@mazinga:~/build$ svn checkout svn://repo.hu/xschem/trunk
```

10. Cd into trunk directory:

```
schippes@mazinga:~/build$ cd trunk
```

12. 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:~/build/trunk$ ./configure --prefix=/usr/local --
user-conf-dir=~/.xschem \
 --user-lib-path=~/share/xschem/xschem_library \
 --sys-lib-path=/usr/local/share/xschem/xschem_library
```

16. 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:~/build/trunk$
```

50. Build xschem by running 'make'

```
schippes@mazinga:~/build/trunk$ make
```

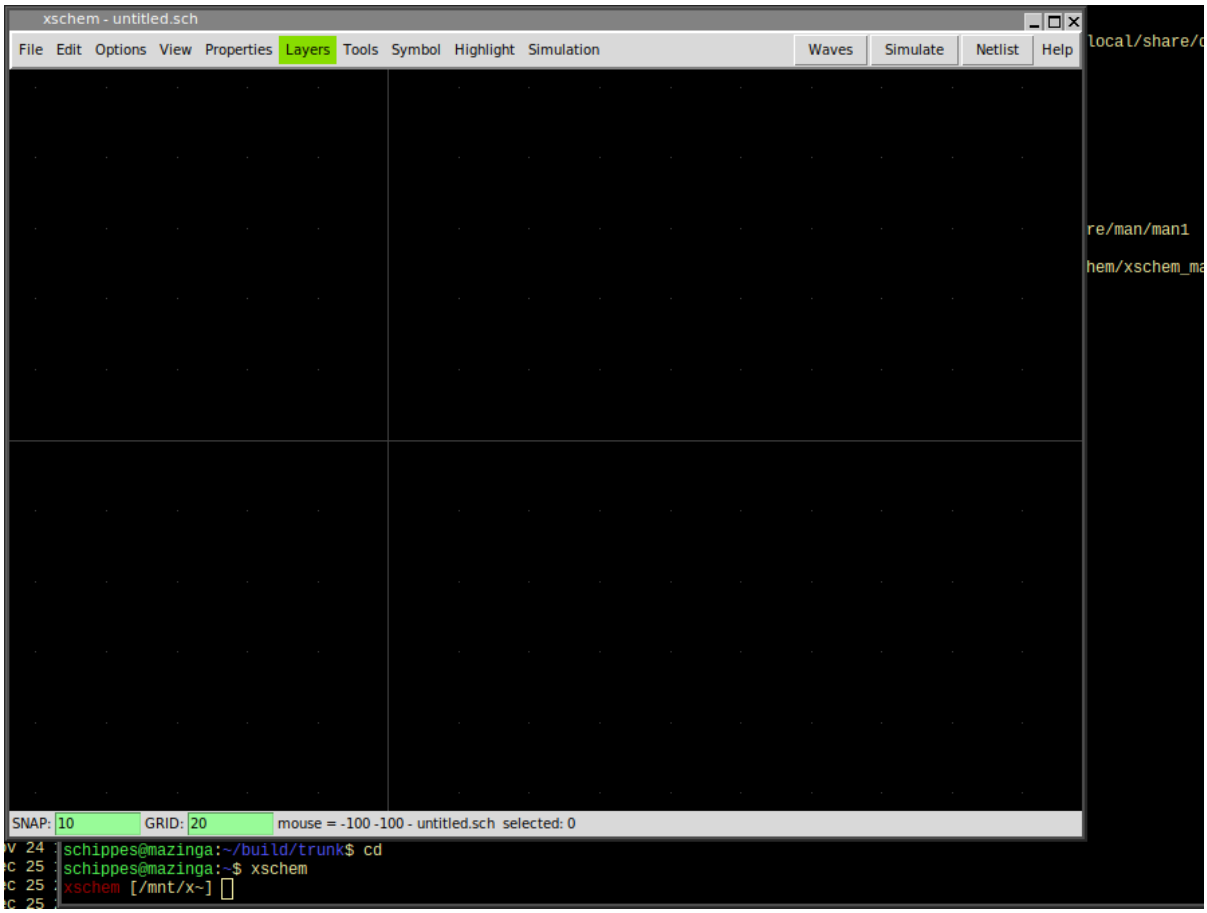
52. If compilation of source files completed with no errors xschem will be ready for installation:

```
schippes@mazinga:~/build/trunk$ sudo make install
```

Note that since we are installing in /usr/local we need root rights (sudo) for doing the installation.

54. Test xschem by launching 'xschem' from the terminal:

```
55.  schippes@mazinga:~/build/trunk$ cd
56.  schippes@mazinga:~$ xschem
```



if /usr/local/bin is not in your PATH variable use the full xschem path:

```
schippes@mazinga:~$ /usr/local/bin/xschem
```

57. Close xschem (menu `File - Exit`)

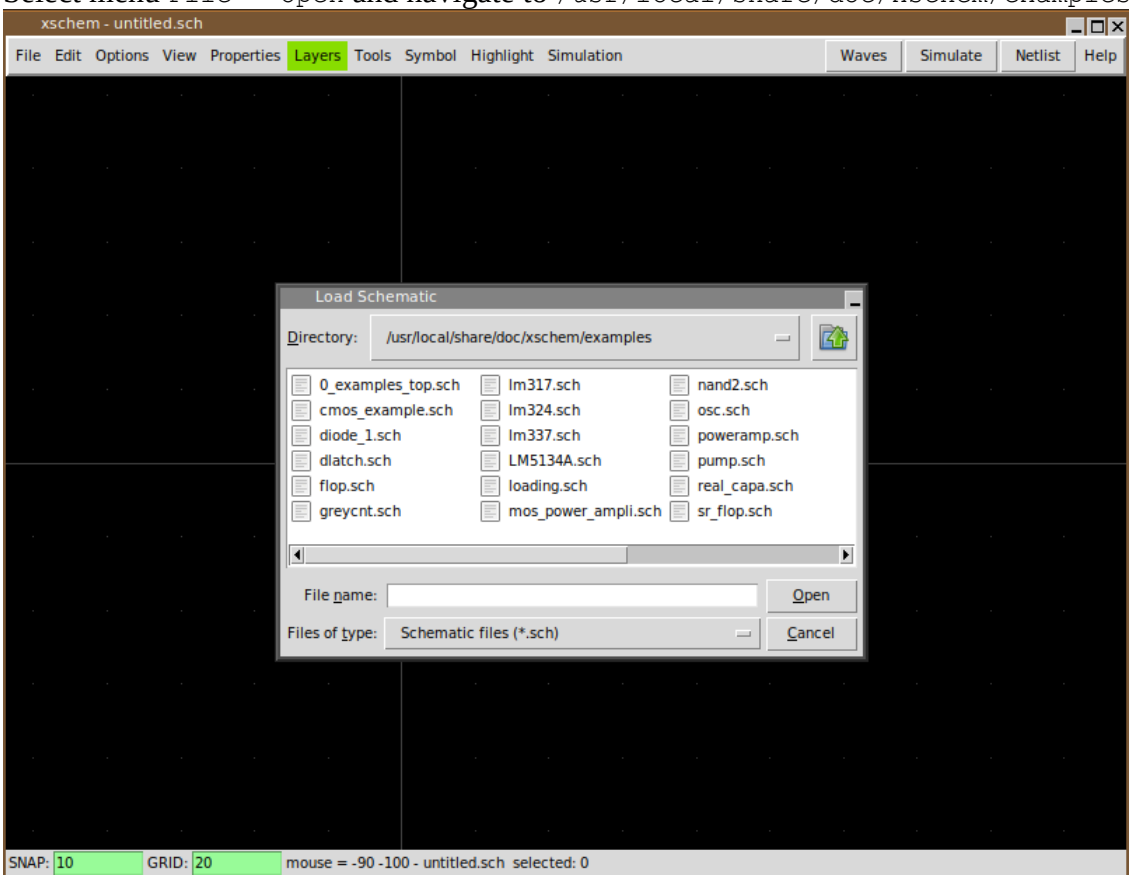
58. Copy the xschemrc file in the `trunk/src` directory to the `~/.xschem` directory. If `~/.xschem` does not exist create it with `mkdir ~/.xschem`

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

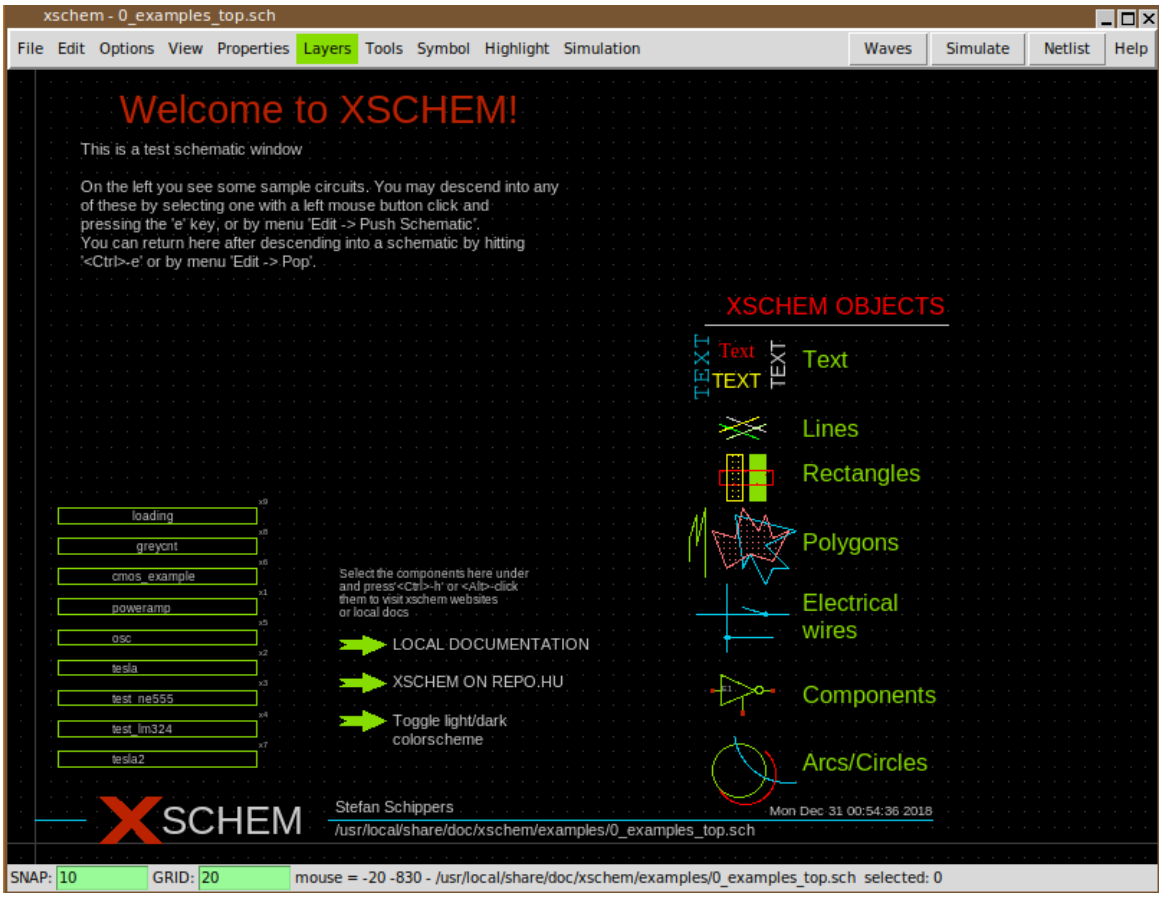
The `~/.xschem/xschemrc` is the user xschem configuration file. You may change it later to change xschem defaults or add / remove / change component and schematic directories. For first tests it is recommended to leave xschemrc as it is.

60. Run xschem again to try some schematic load tests:
61. `schippes@mazinga:~$ xschem`

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

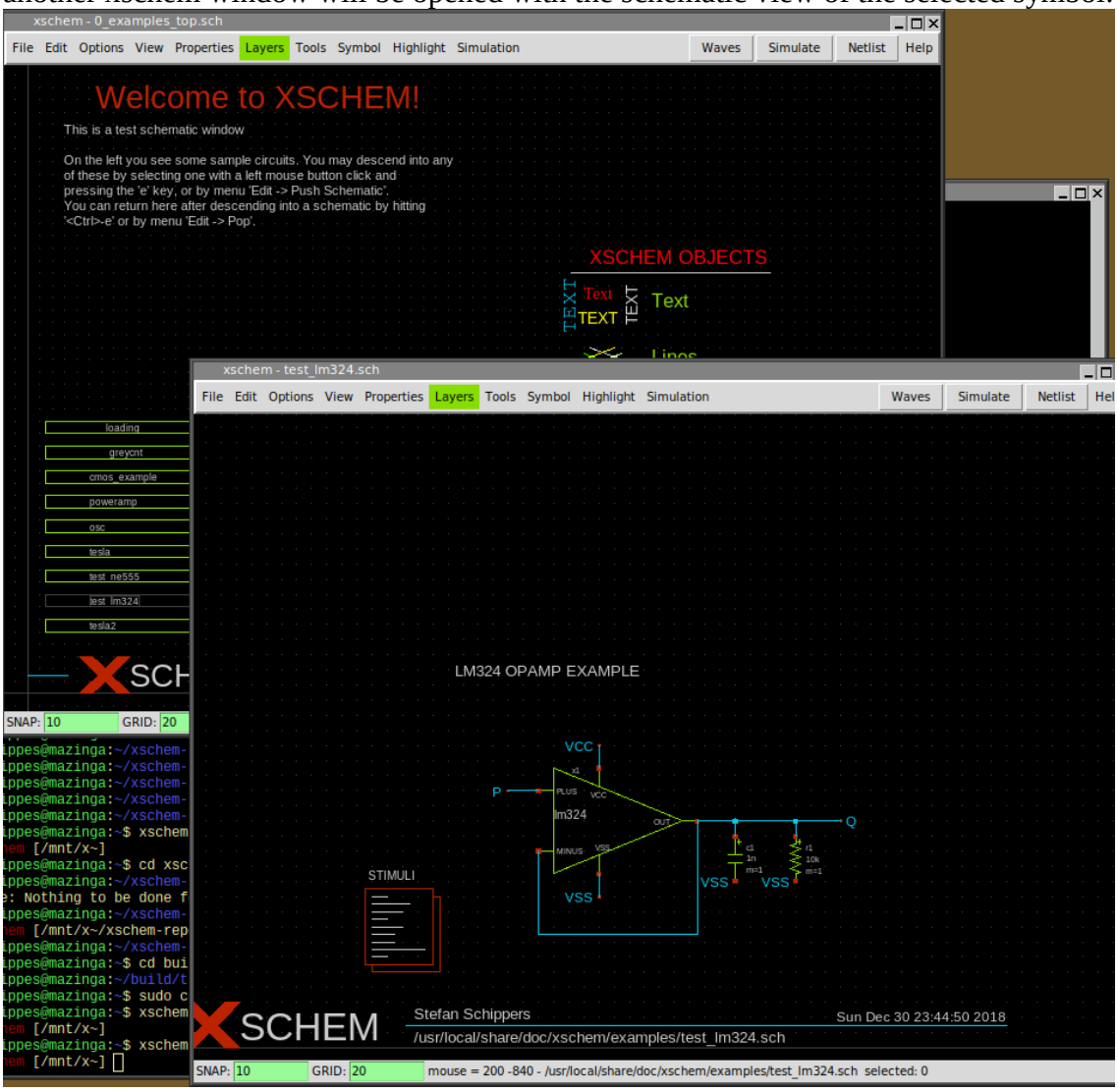


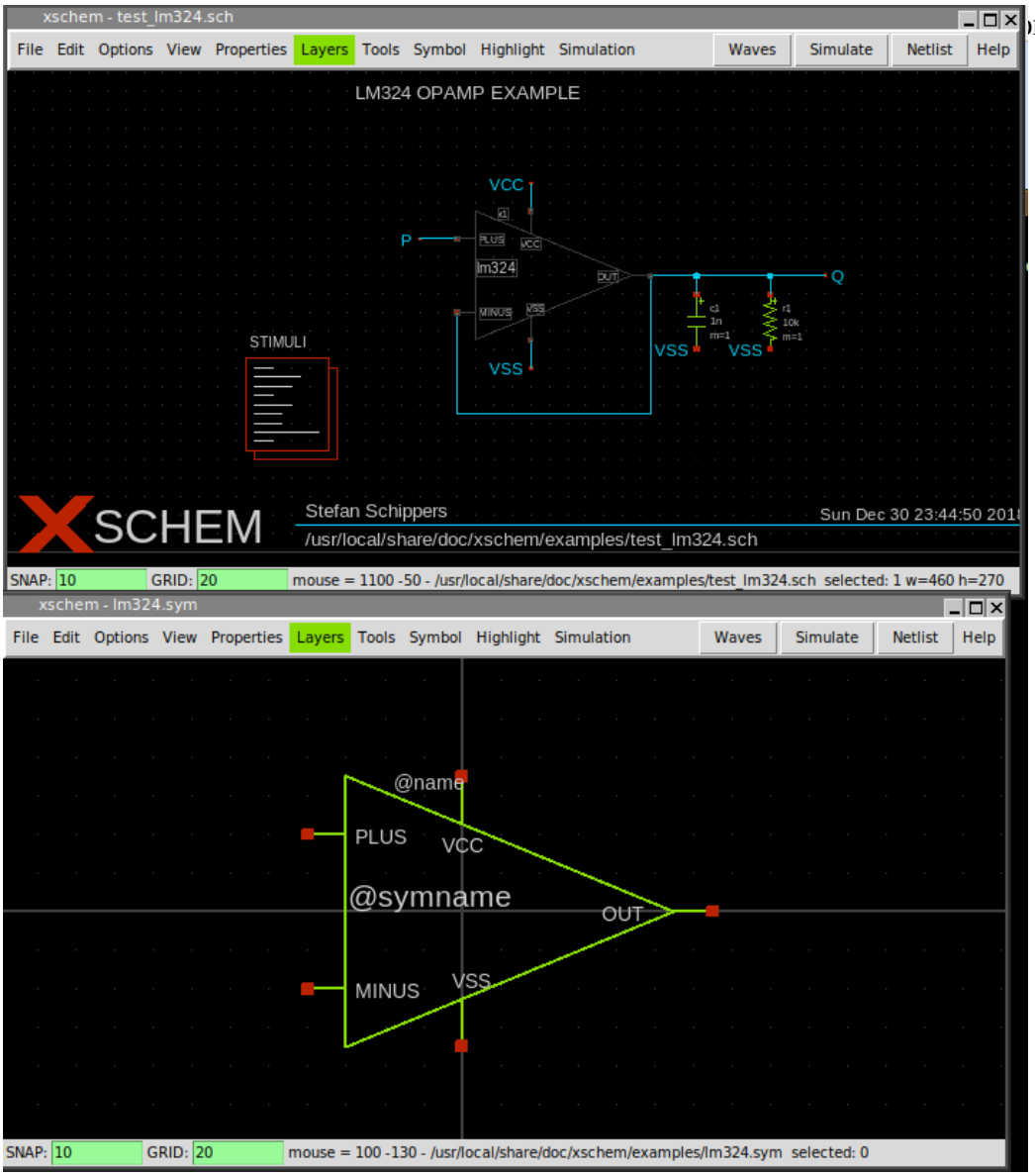
63.



64. 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:



65. 

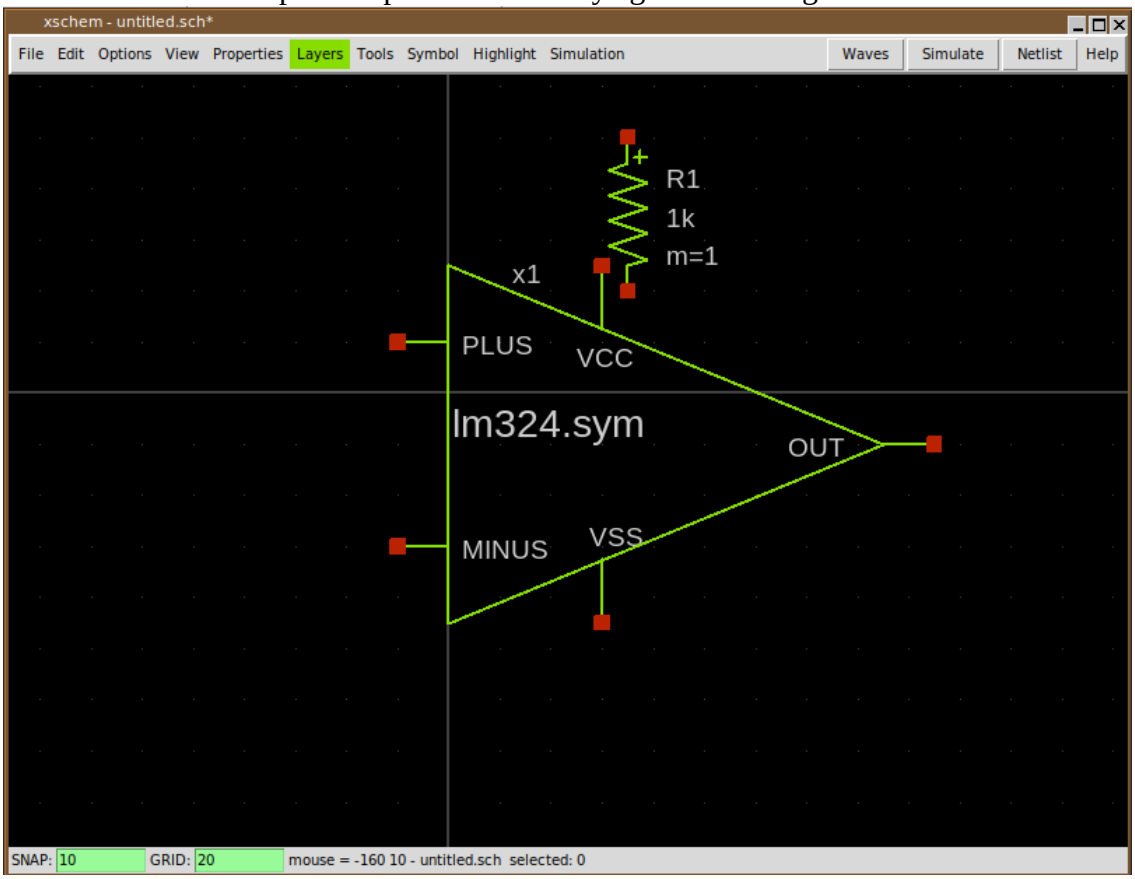
66. Now close all xschem windows and restart a new xschem instance from terminal:

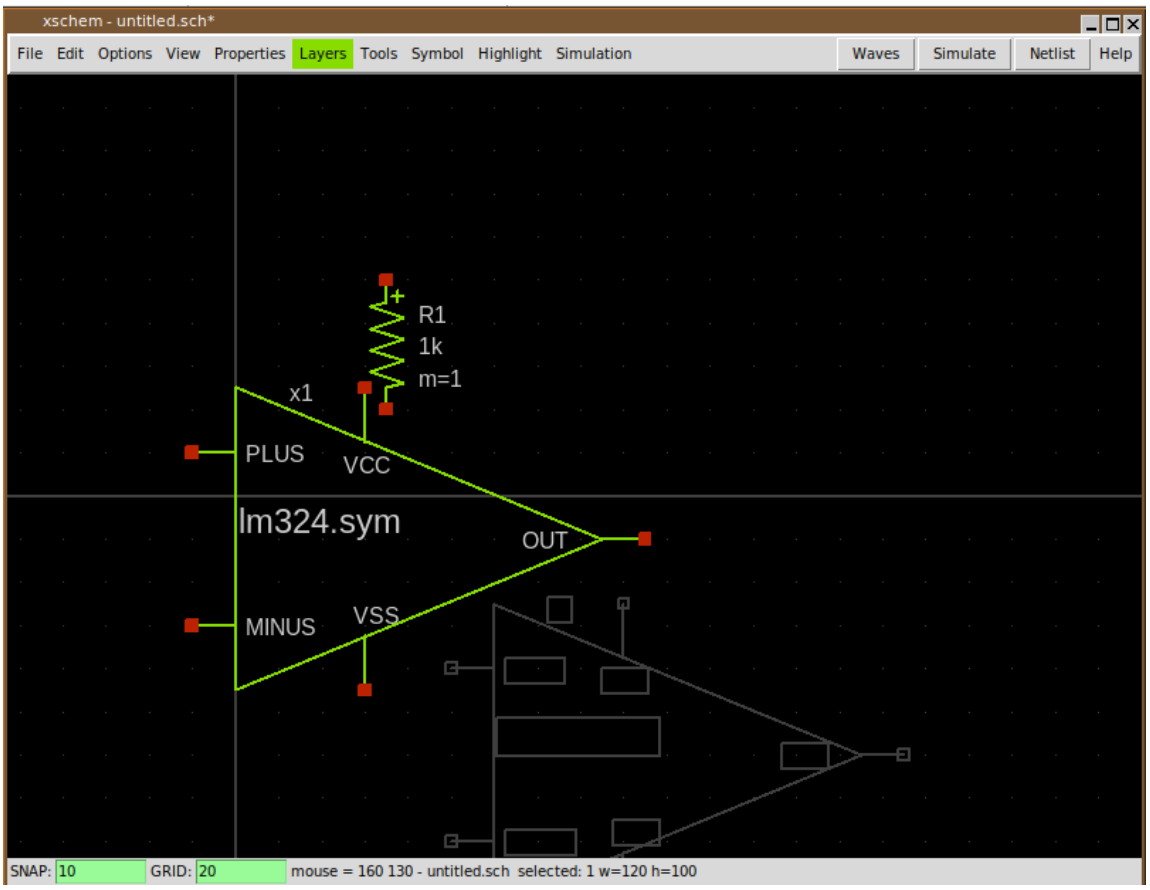
```
67.  schippes@mazinga:~$ xschem
```

68. 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

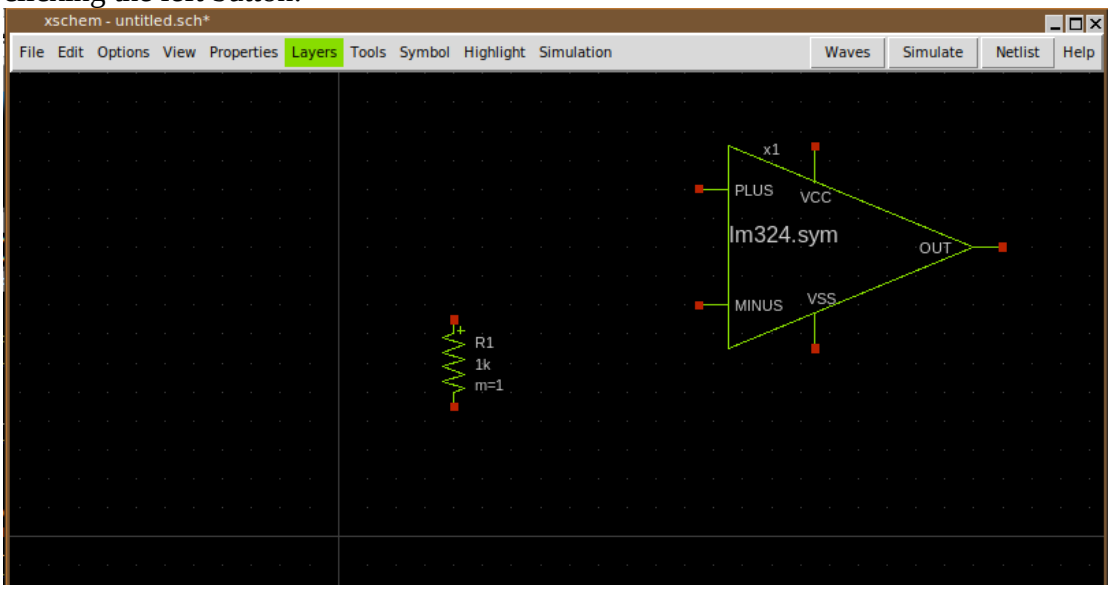
`/usr/local/share/xschem/xschem_library` and select `res.sym`:

69. Lets add another component: press `Insert` key again and navigate to

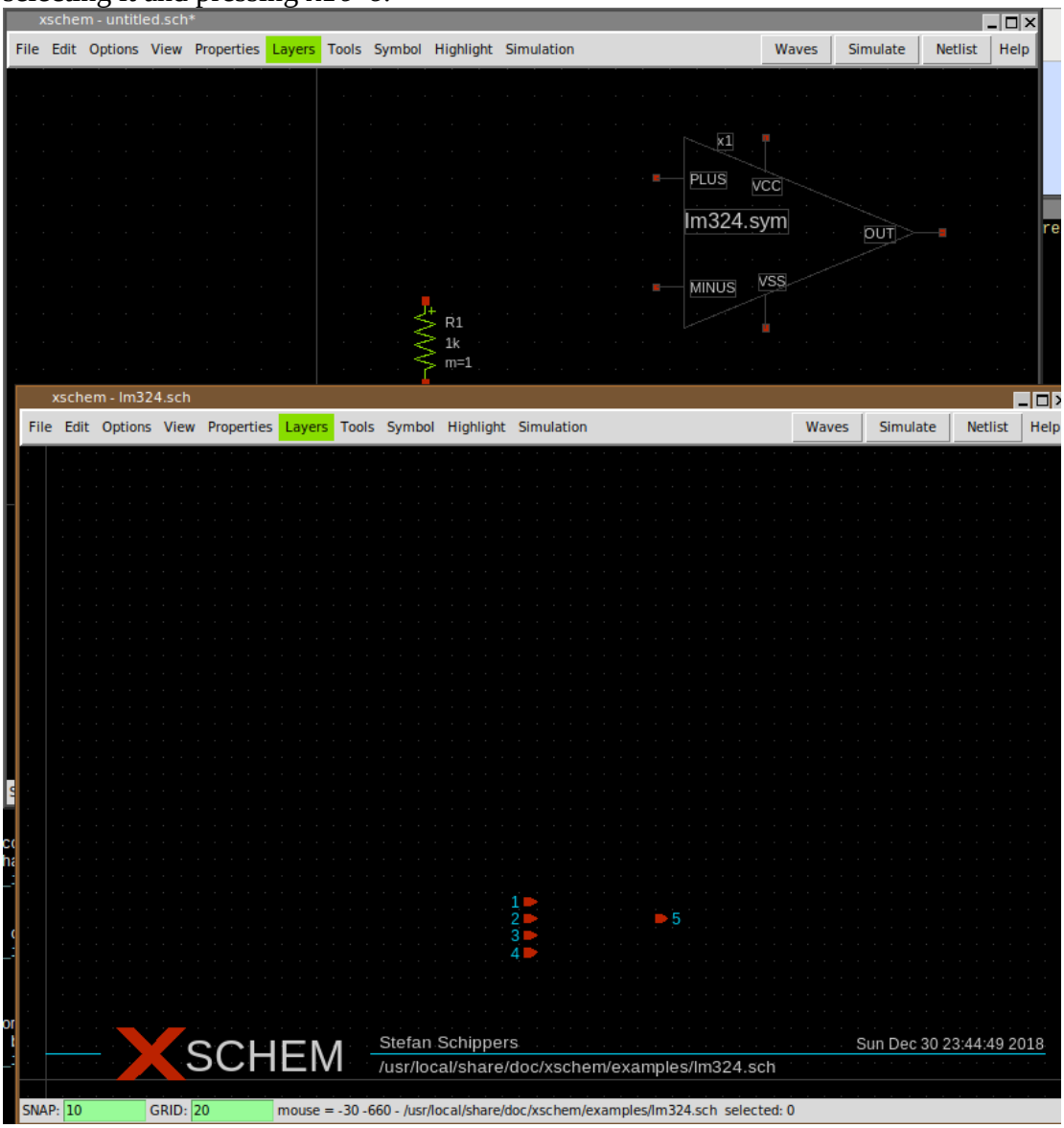


70. 

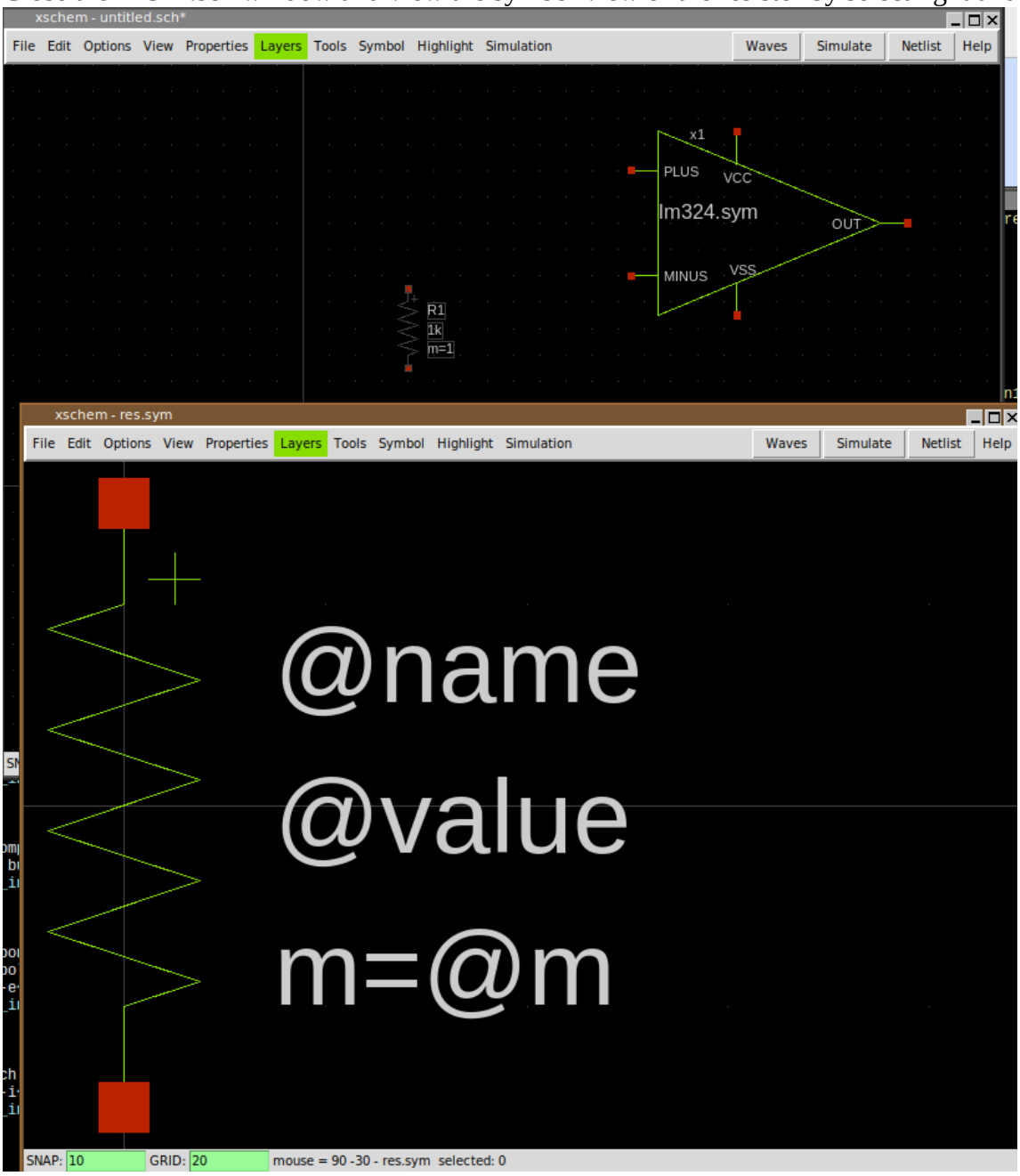
71. Place the lm324 component where you want in the schematic by placing the mouse and clicking the left button:



72. 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`:



73. Close the lm324.sch window and view the symbol view of the resistor by selecting it and



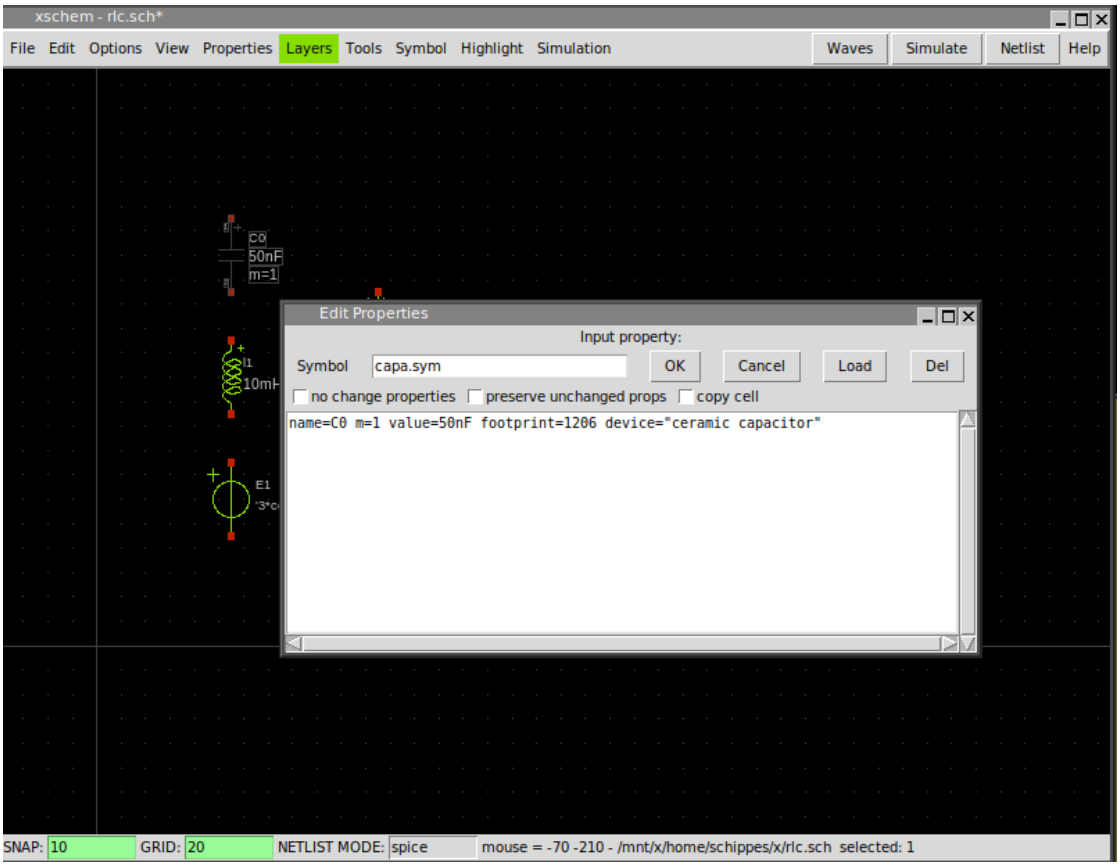
This concludes the tutorial, if all the steps were successful there is a good probability that xschem is correctly installed on your system.

TUTORIAL: RUN A SIMULATION WITH XSCHEM

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

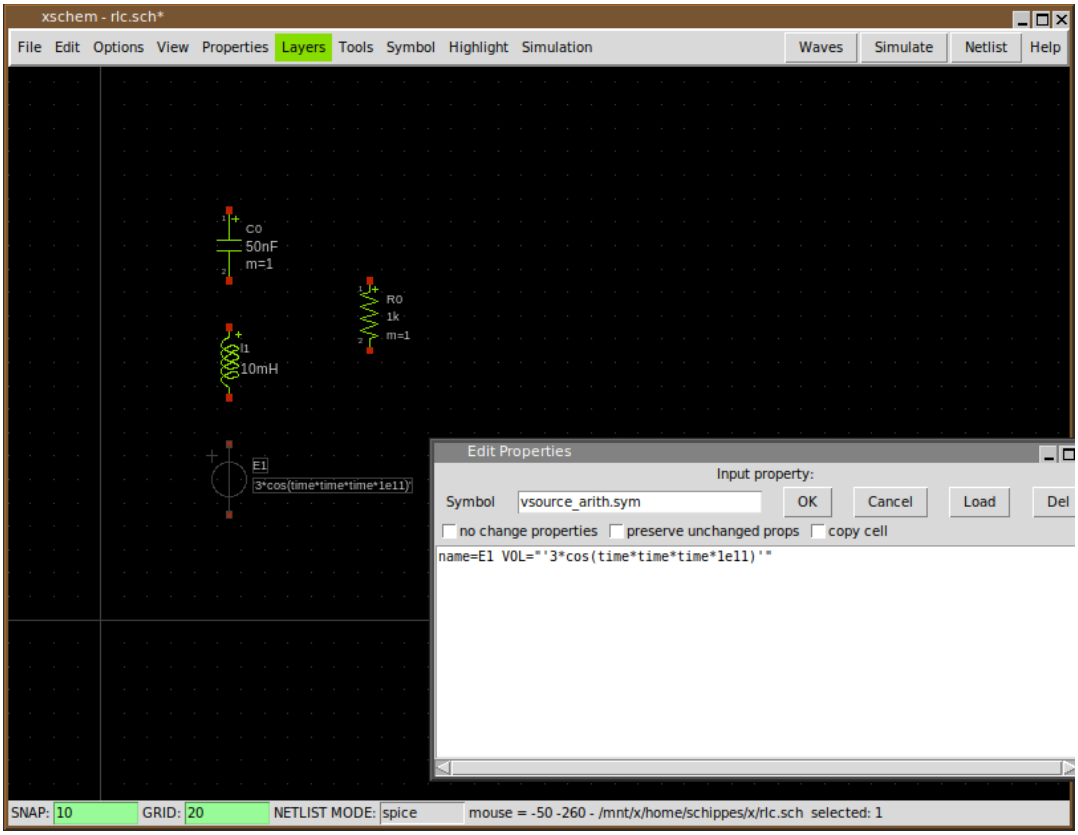
1. Build and install xschem from svn head.
2. Create some empty directory (in my examples i use ~/x)
3. cd ~/x
4. ~/bin/xschem rlc (use the actual xschem install path). xschem will warn you that the rlc.sch file does not exist. No problem.
5. Press Insert key
6. Navigate in the file selector to .../share/xschem/xschem_library/devices
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:

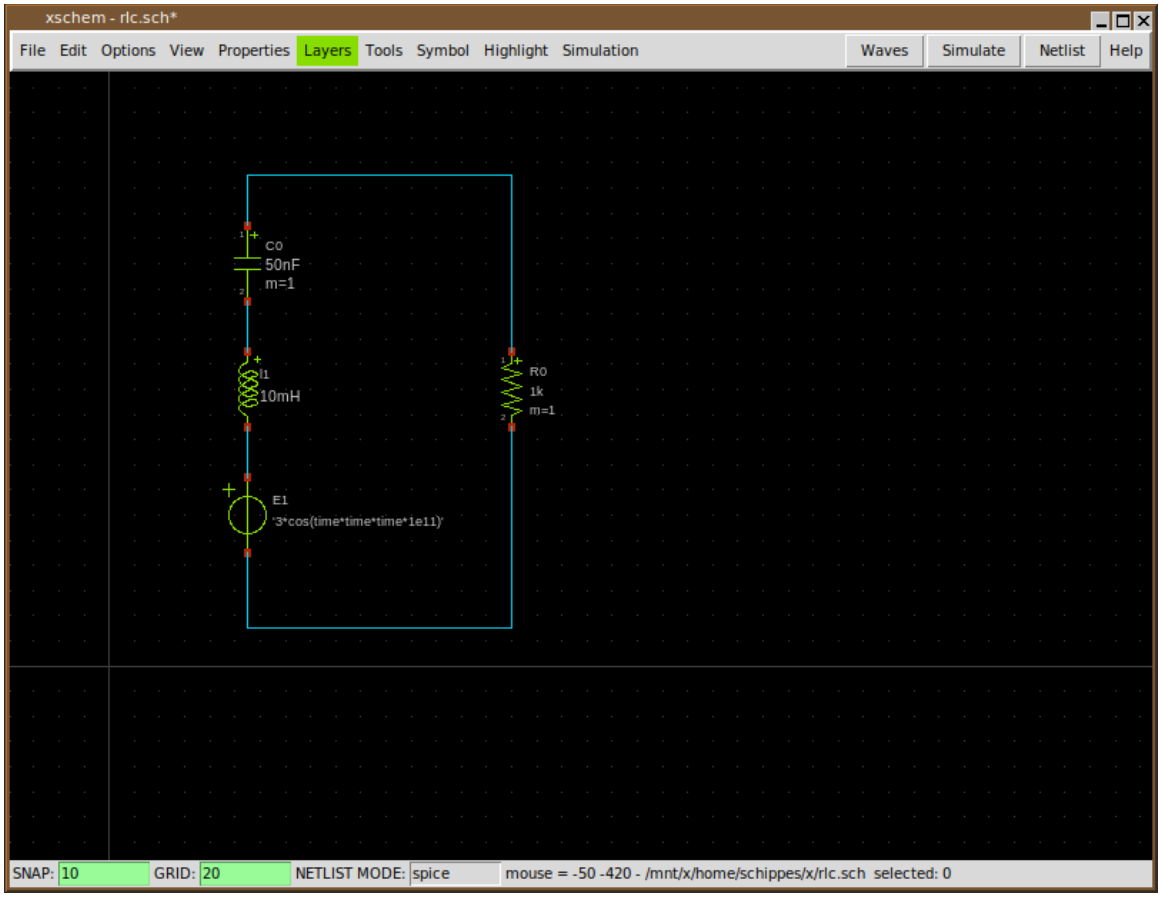


13. Do the same for the inductor (10mH) and the resistor (1k)

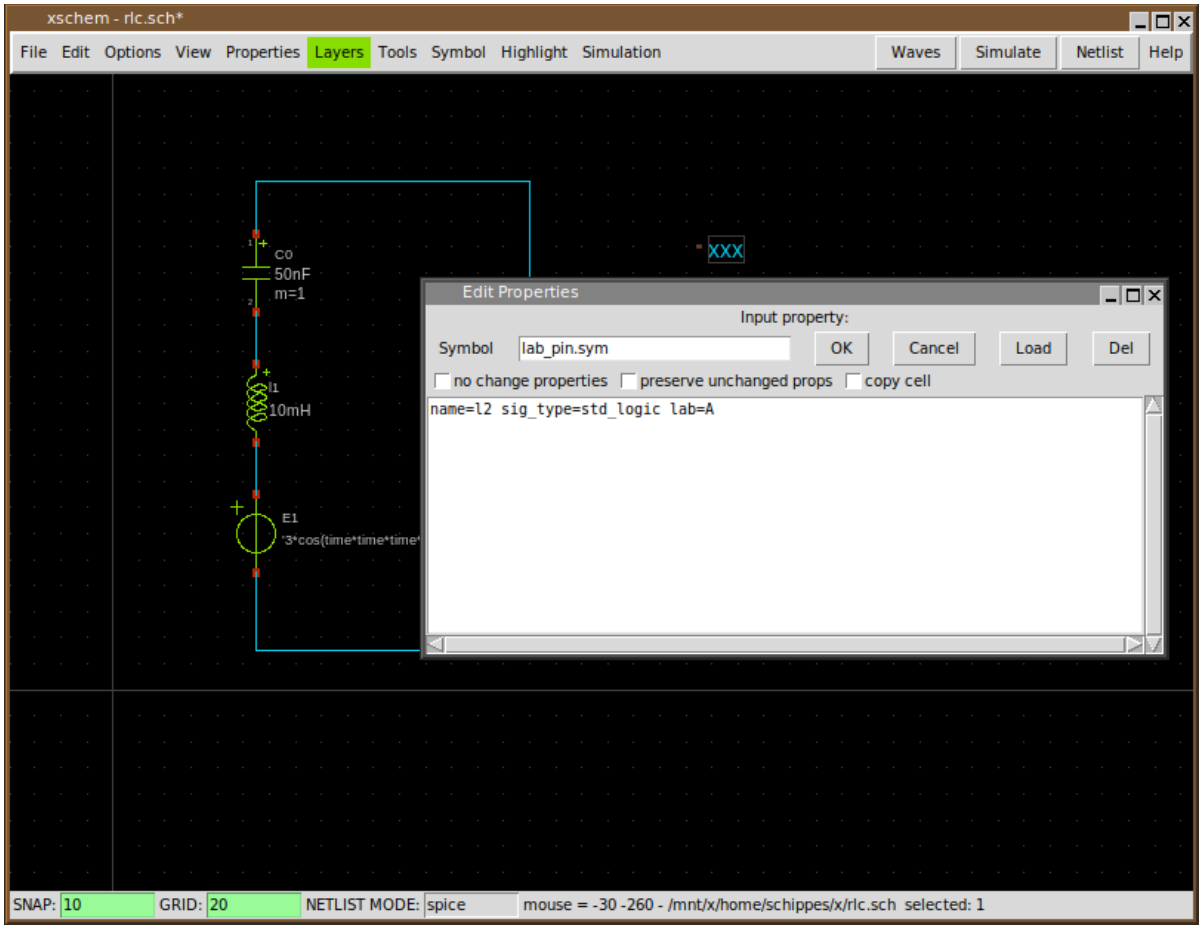
14. Set the voltage source VOL to: "'3*cos(time*time*time*1e11)'" (include quotes, single and double):



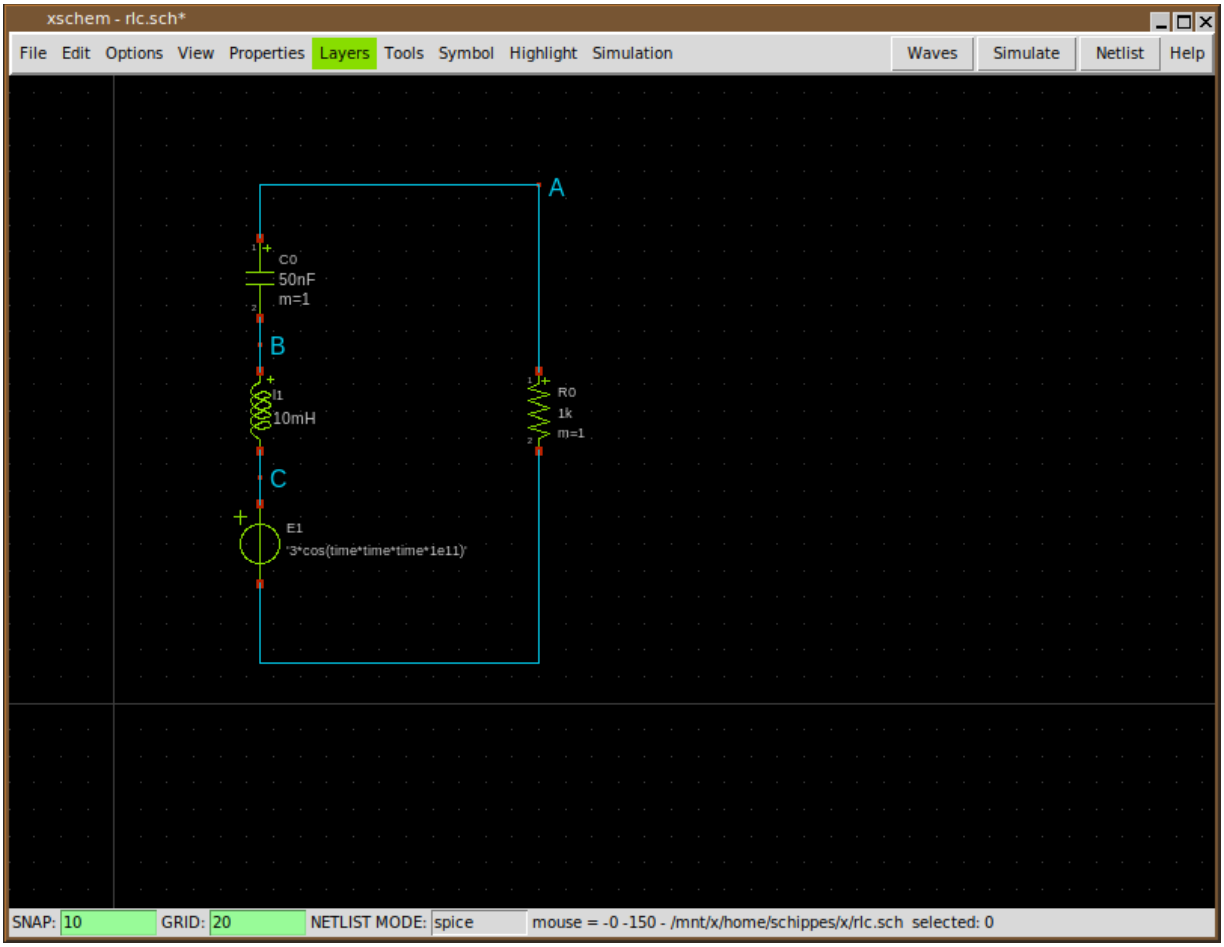
15. Pressing the 'w' key and moving the mouse you draw wires, wire the components as shown (press 'w', move the mouse and click, this draws a wire segment):



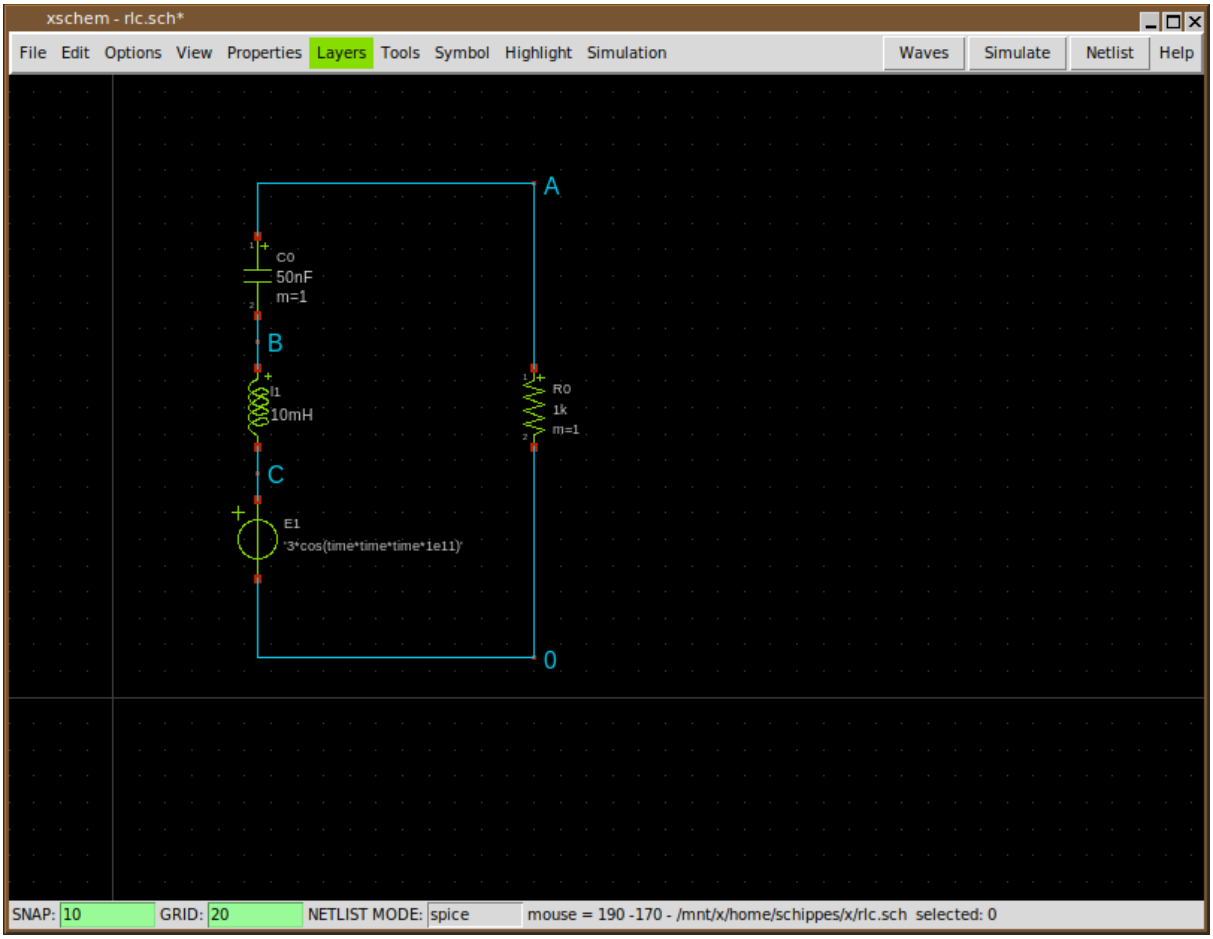
16. Press 'Insert key and place one instance of 'lab_pin', then use the right mouse button to change its 'lab' attribute to A:



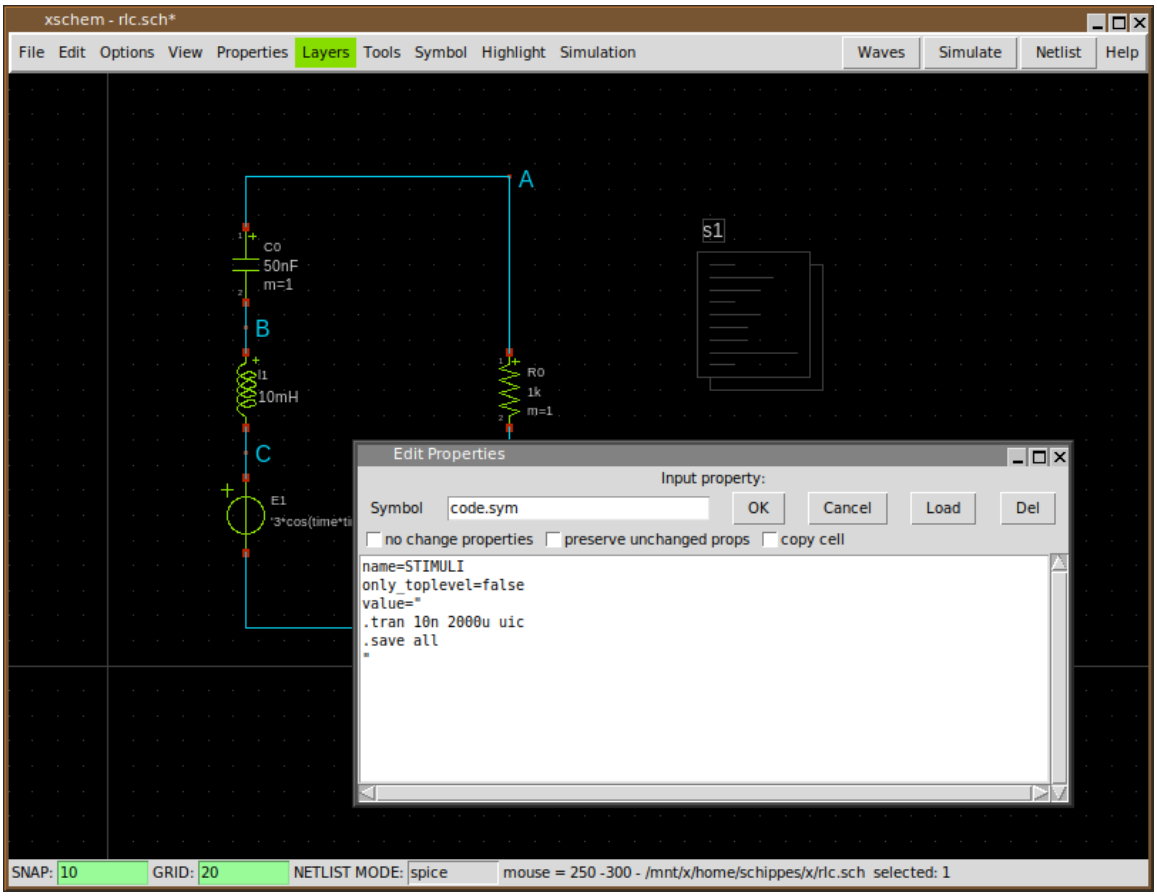
17. Move the label as shown, (you can use 'F' to flip and 'R' to rotate), then using 'c' copy this pin label and edit attributes to create the B and C labels, place all of these as shown:



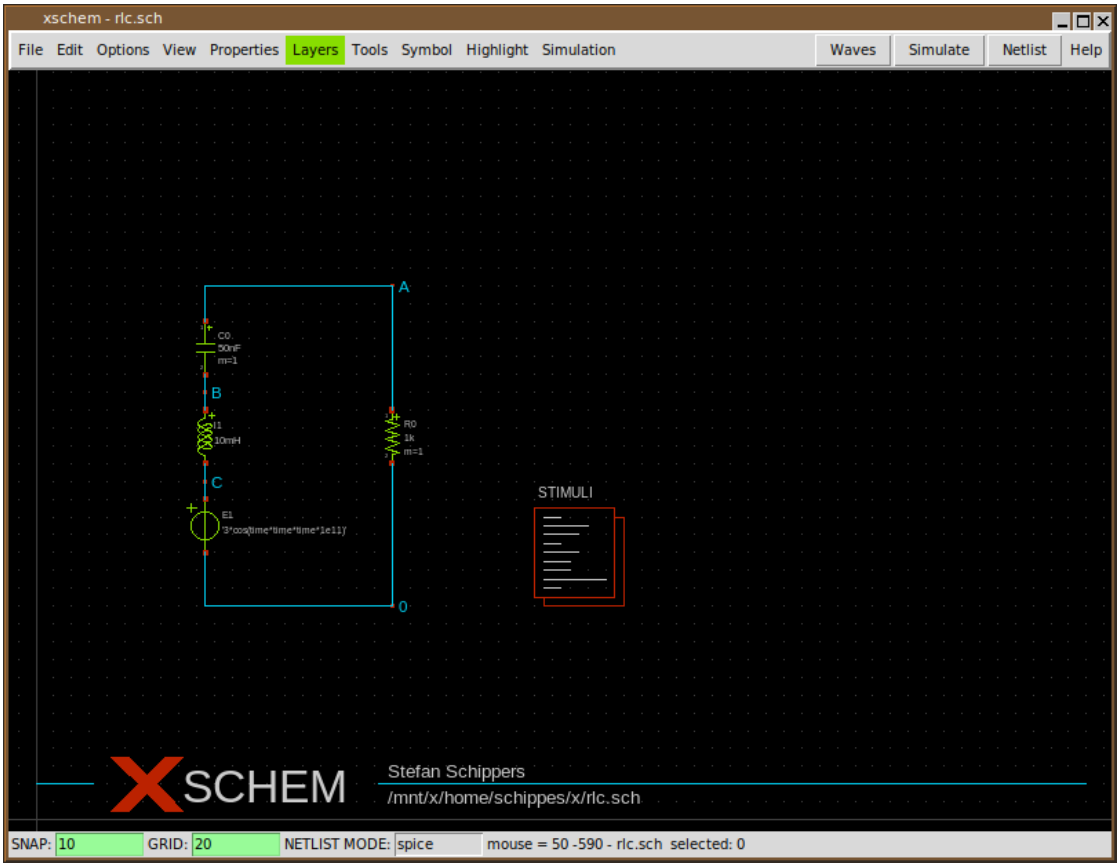
18. Select the 'C' label and copy it as shown here, set its lab attribute to 0 (this will be the 0V (gnd node))



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

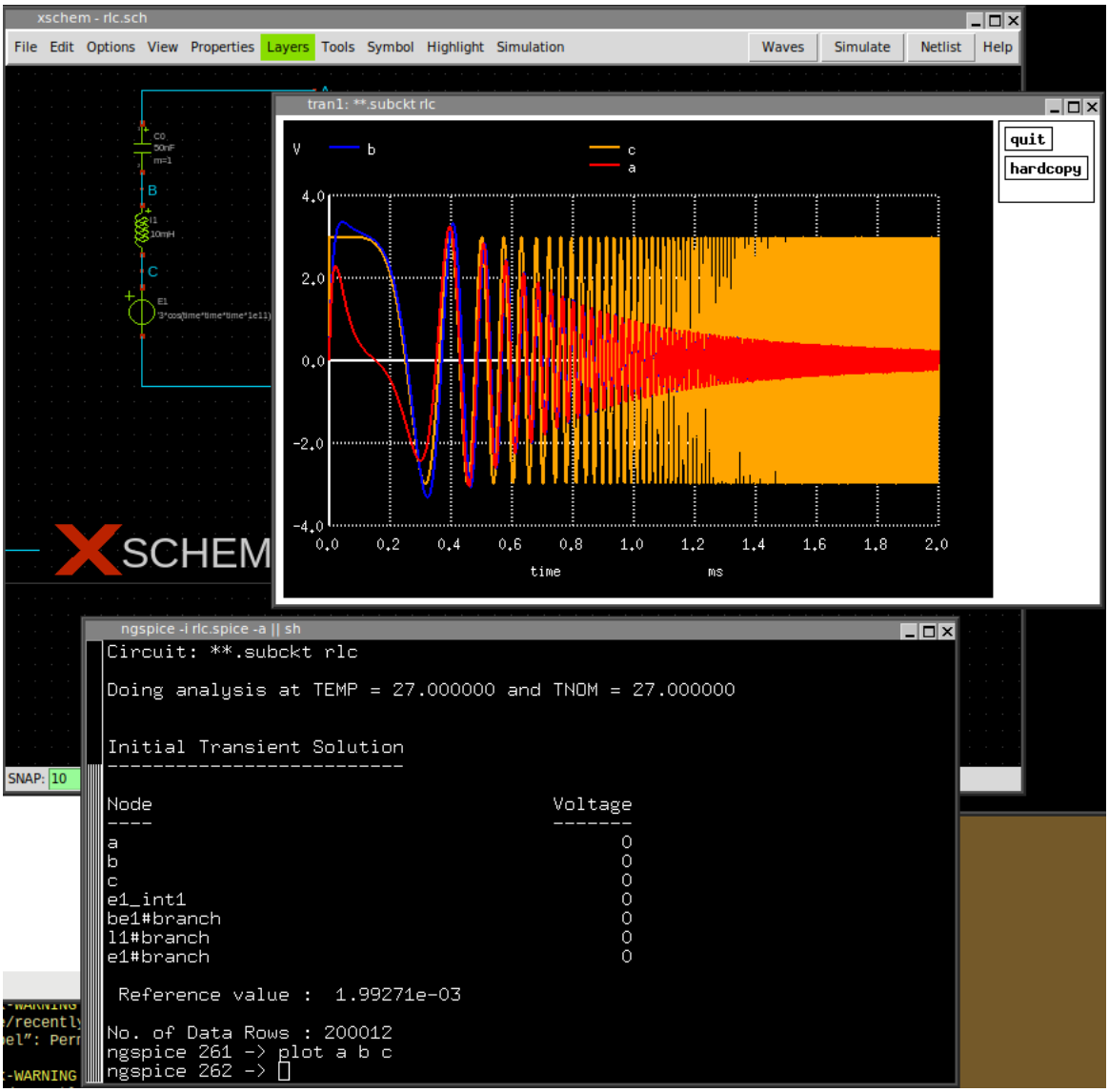


20. Cosmetics: add 'title.sym' move the circuit (by selecting it dragging the mouse and pressing 'm', if needed). Note that you can do a 'stretch move'operation if you need move components keeping the wires attached; refer to the xschem manual here



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':

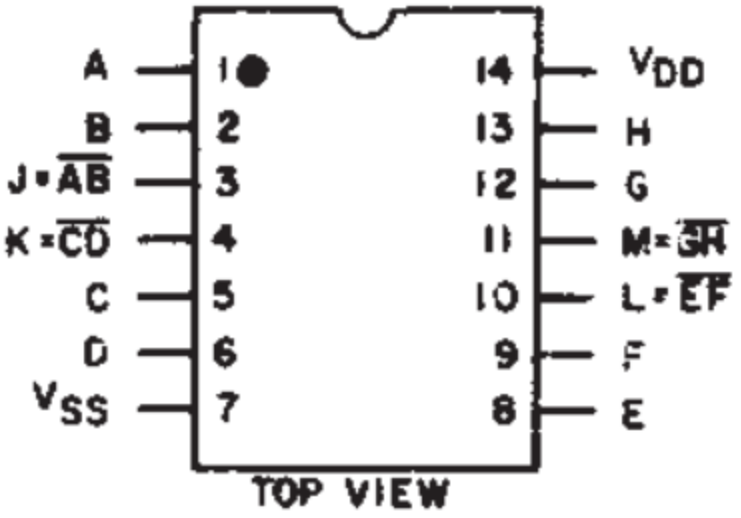
23. In the simulator window type 'plot a b c':

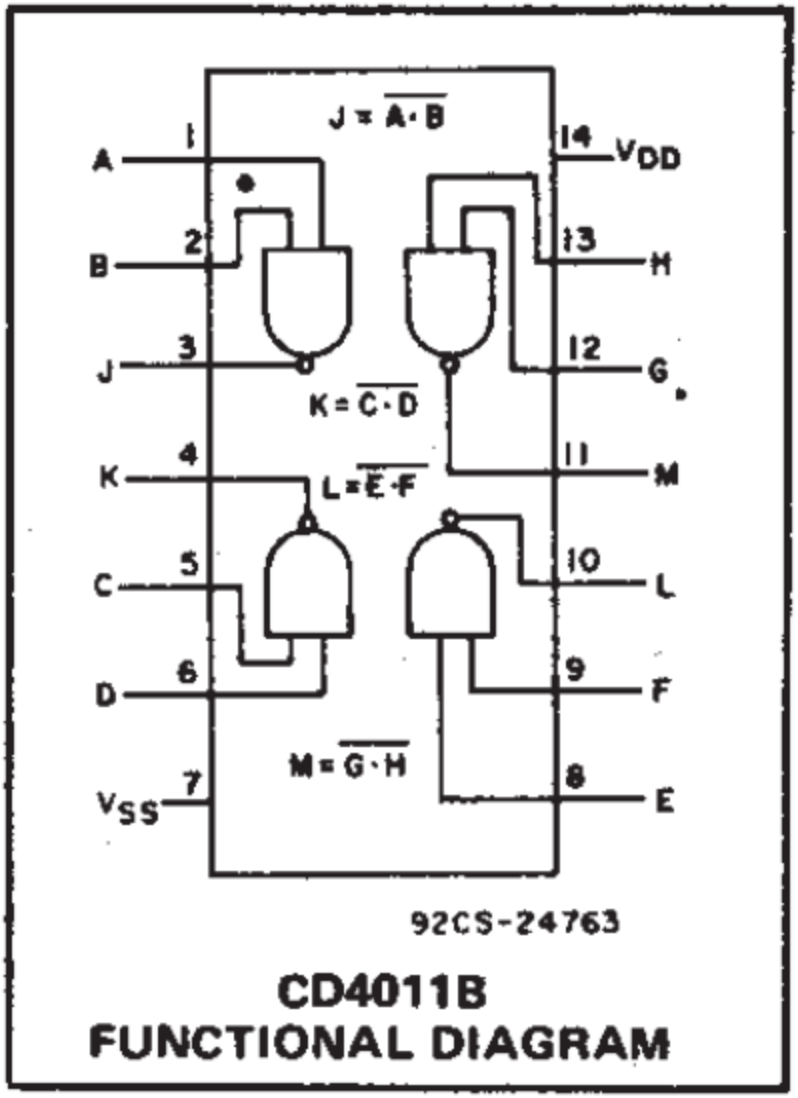


24. If you set 'Simulation -> Ngspice Batch Spice Simulator' and press 'Simulate' again the sim will be run in batch mode, a 'rlc.raw' file will be generated and a 'rlc.out' file will contain the simulator textual output.

TUTORIAL: CREATE AN XSCHEM SYMBOL

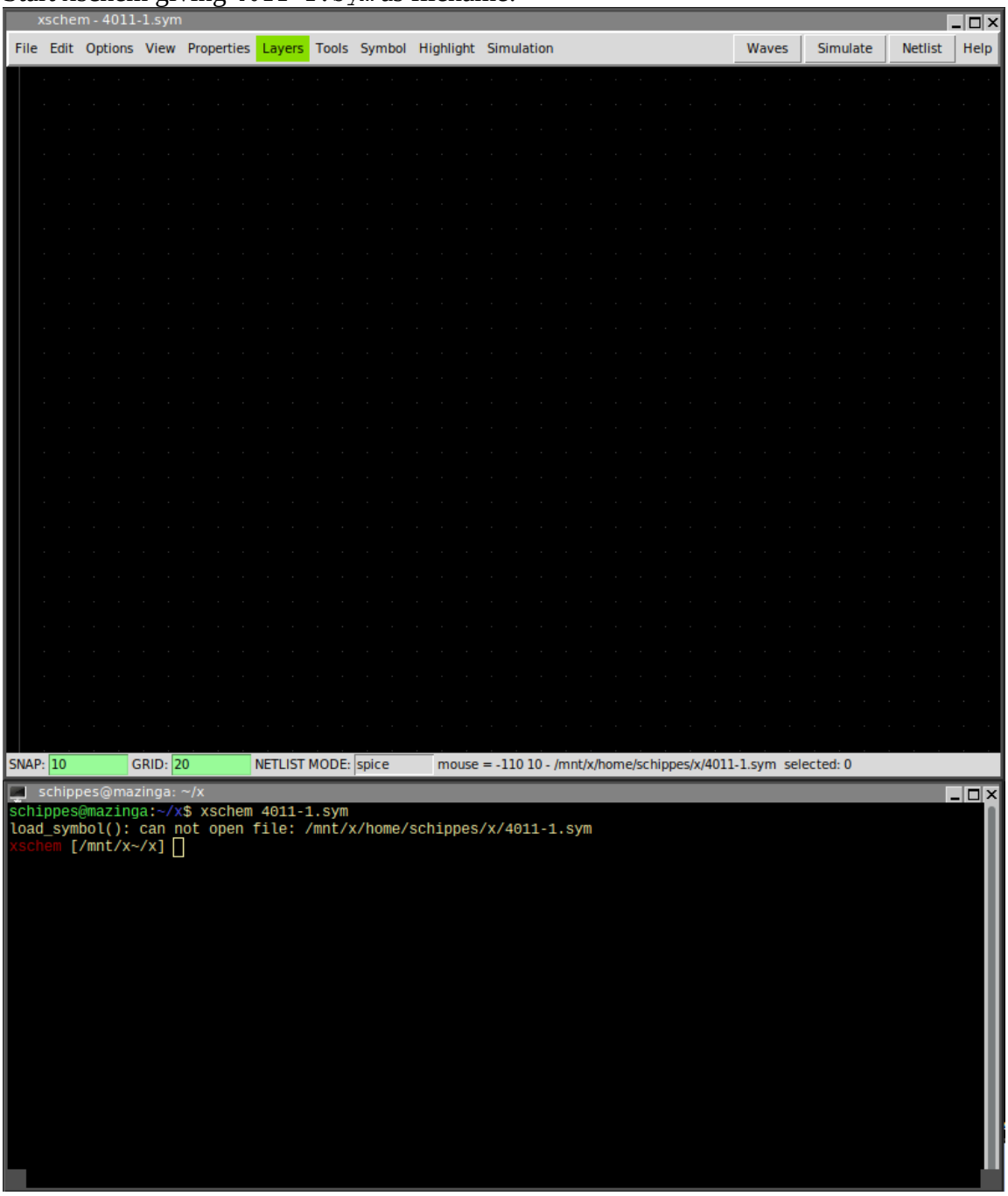
In this tutorial we will build a 4011 CMOS quad 2-input NAND symbol. This IC has 4 nand gates (3 pins each, total 4*3=12 pins + VDD,VSS power pins) This device comes in a dual in line 14 pin package.



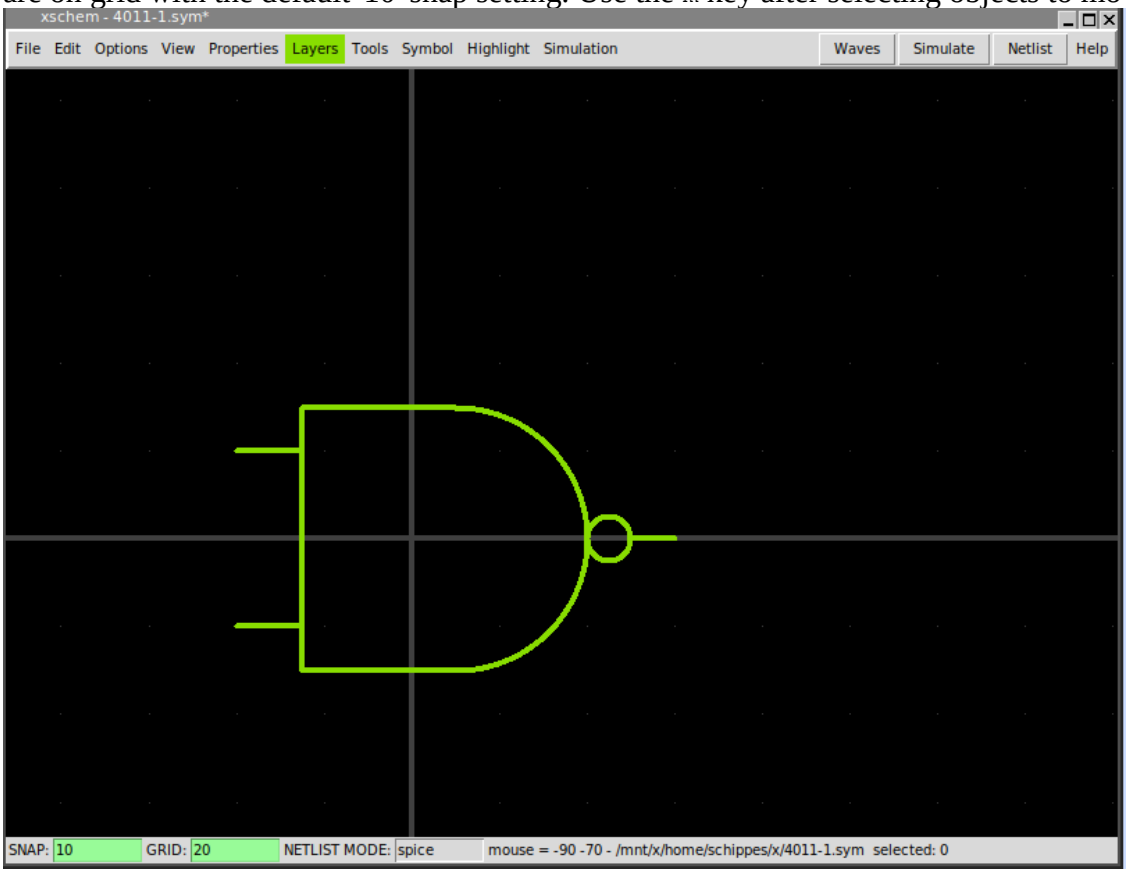


CD4011B
FUNCTIONAL DIAGRAM

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

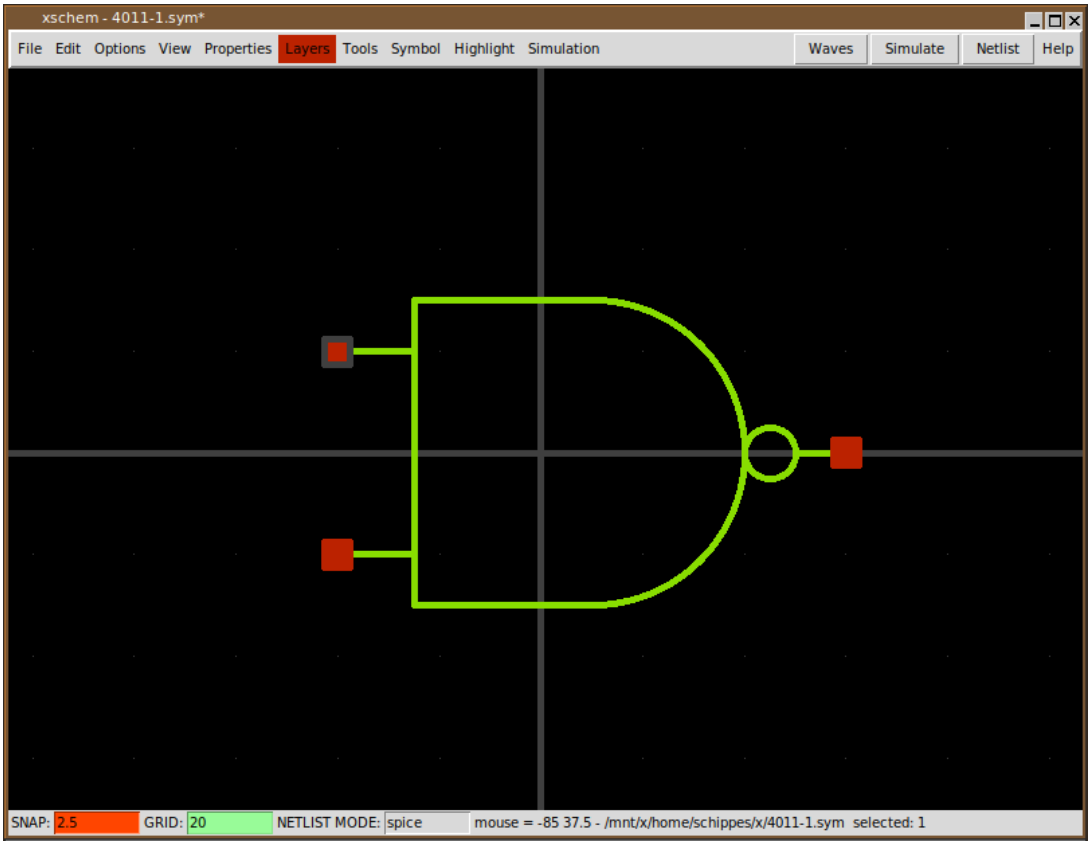


2. use layer 4 (the default) to draw the following shapes, use `l` to draw lines and use `Shift-c` to draw arcs, use `Ctrl-Shift-c` to draw circles. Arcs and circles are drawn by specifying start - end point and a 3rd way point. You will need to change the grid snap to '5' for drawing the smallest objects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



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.

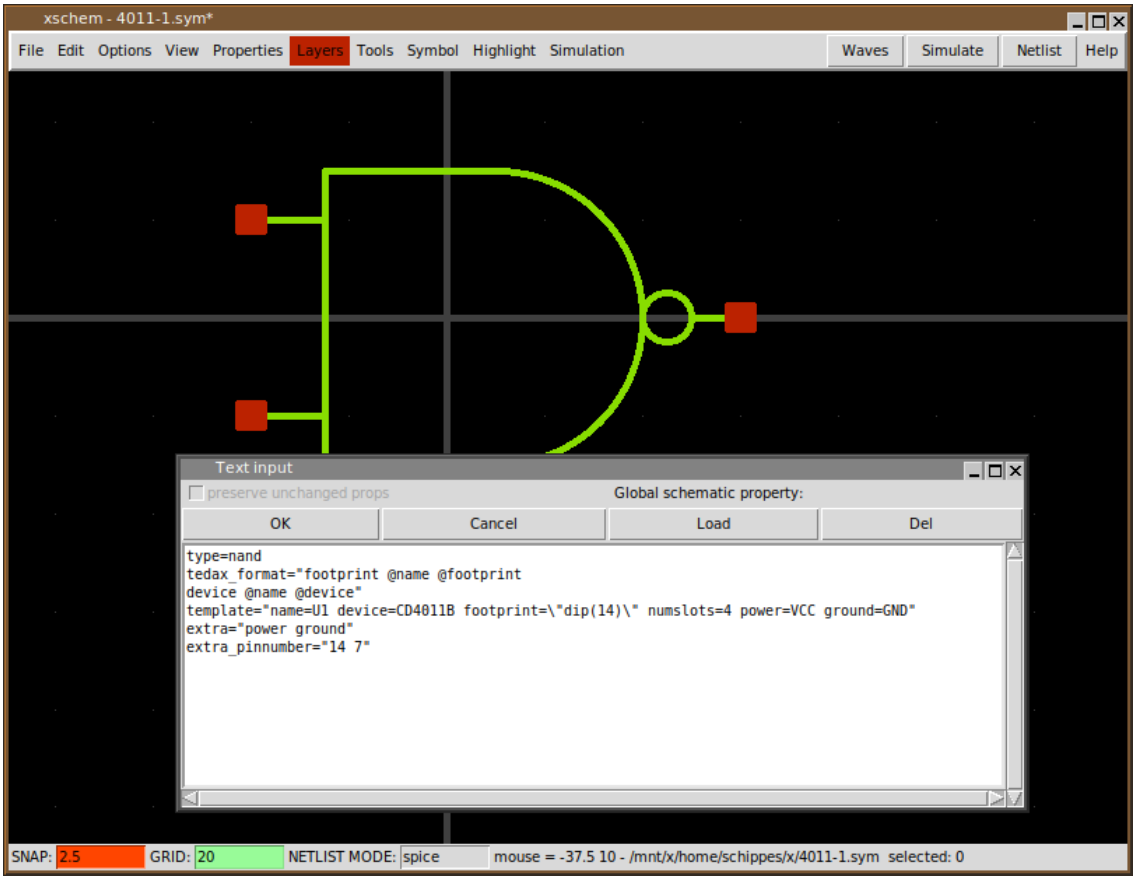


4. Now when **no object is selected** press `q` to edit the symbol global attributes. Type the following text:

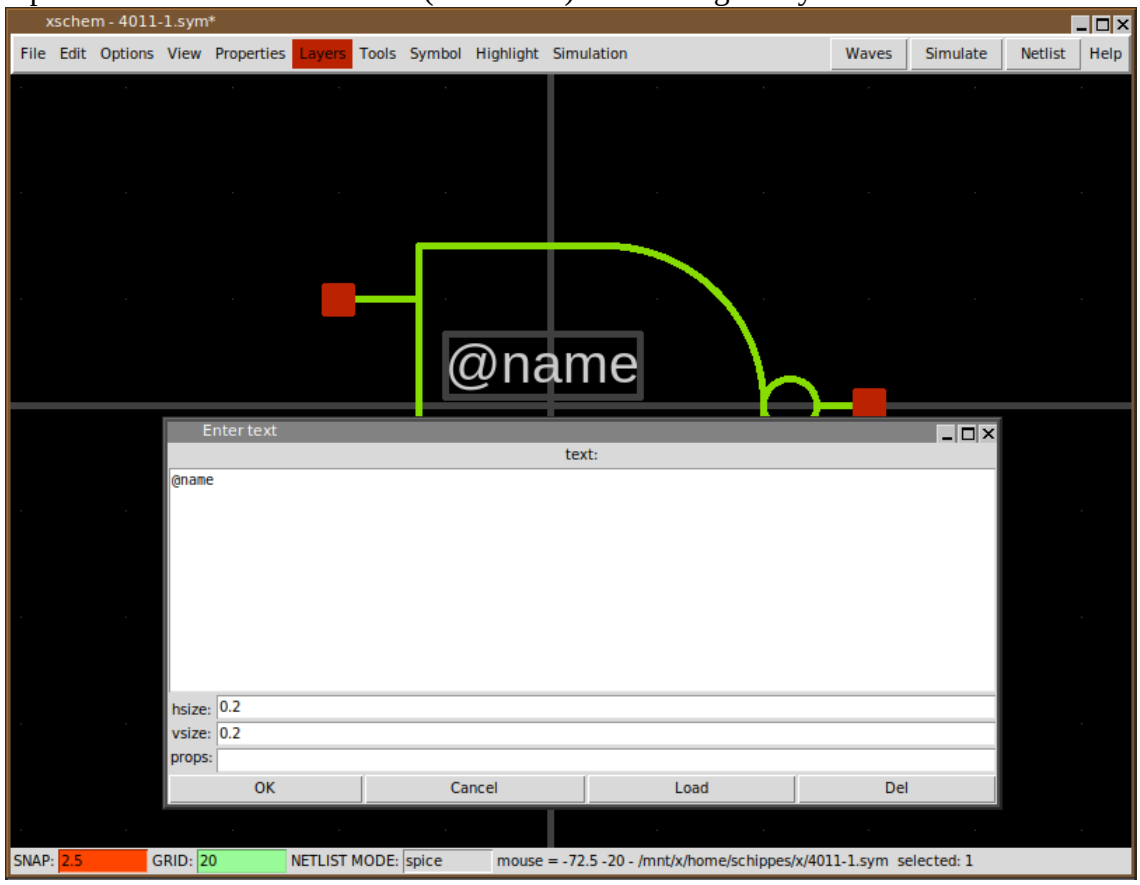
```
5. type=nand
6. tedax_format="footprint @name @footprint
7. device @name @device"
8. template="name=U1 device=CD4011B footprint=\"dip(14)\" numslots=4
   power=VCC ground=GND"
9. extra="power ground"
10.  extra_pinnumber="14 7"
```

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

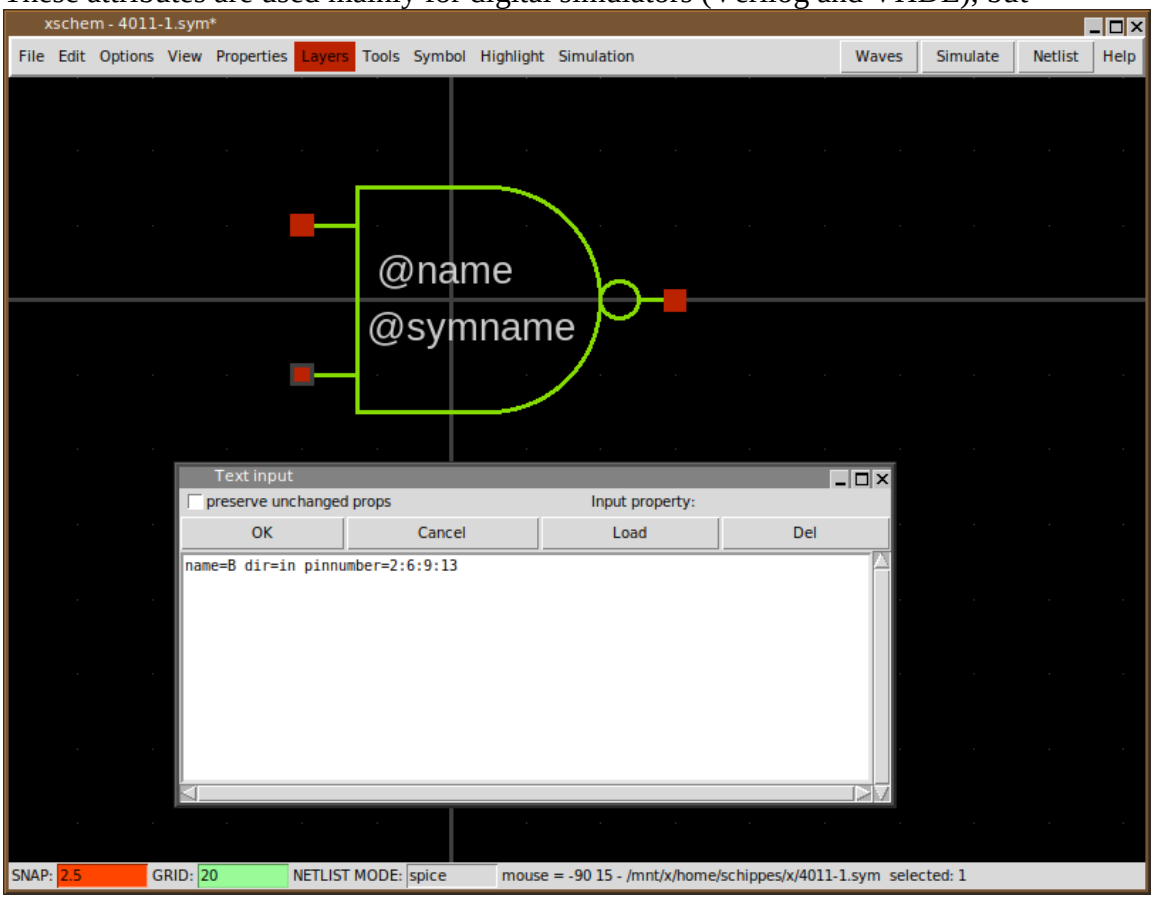
these attributes specify the gate type, the format for tedax netlist, the `template` attribute specifies default values for attributes and defines pin connection for VDD and VSS that are associated to package pins 14 and 7. The `device` attribute specifies the component name to be used in the tEDAx netlist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



11. 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

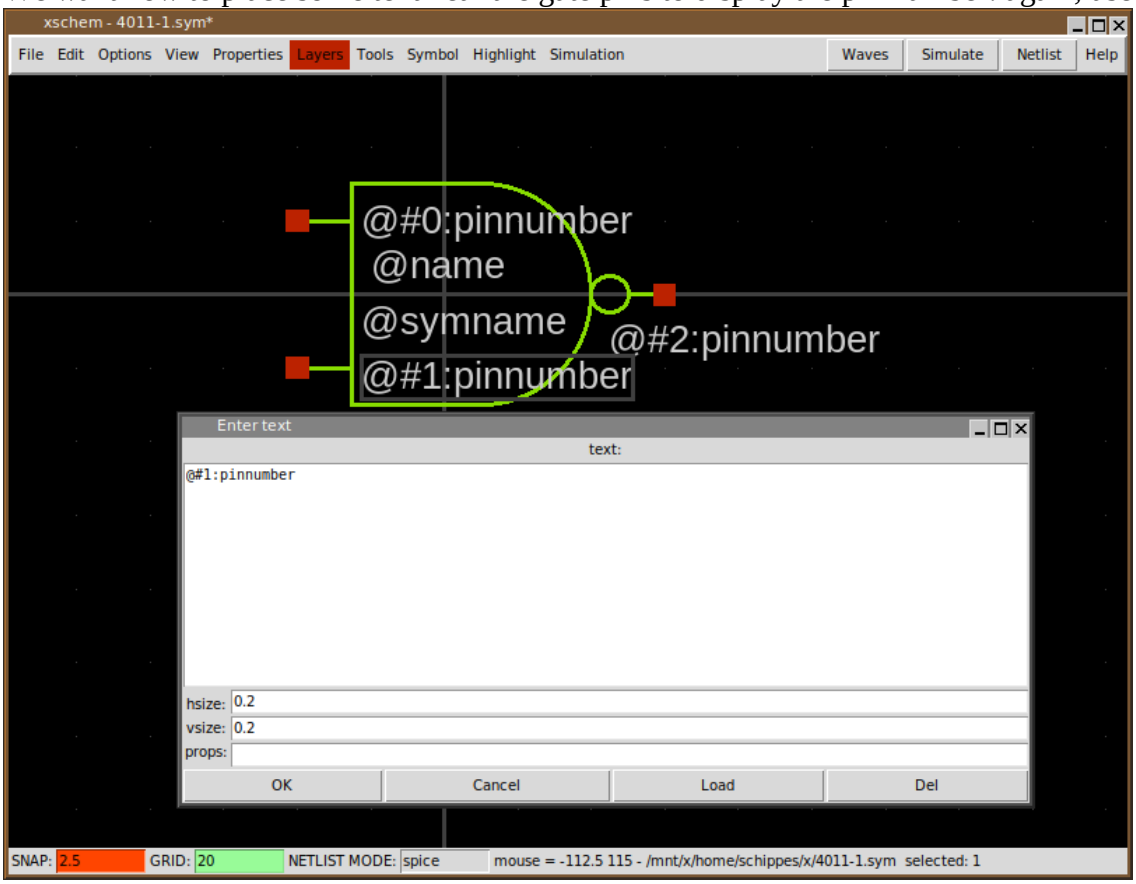


12. select the red pins (click the mouse close to the interior side of the rectangle corners) and press `q`, set attribute
`name=A dir=in pinnumber=1:5:8:12` for the upper left pin, `name=B dir=in pinnumber=2:6:9:13` for the lower left pin, `name=Z dir=out pinnumber=3:4:10:11` for the right output pin.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



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

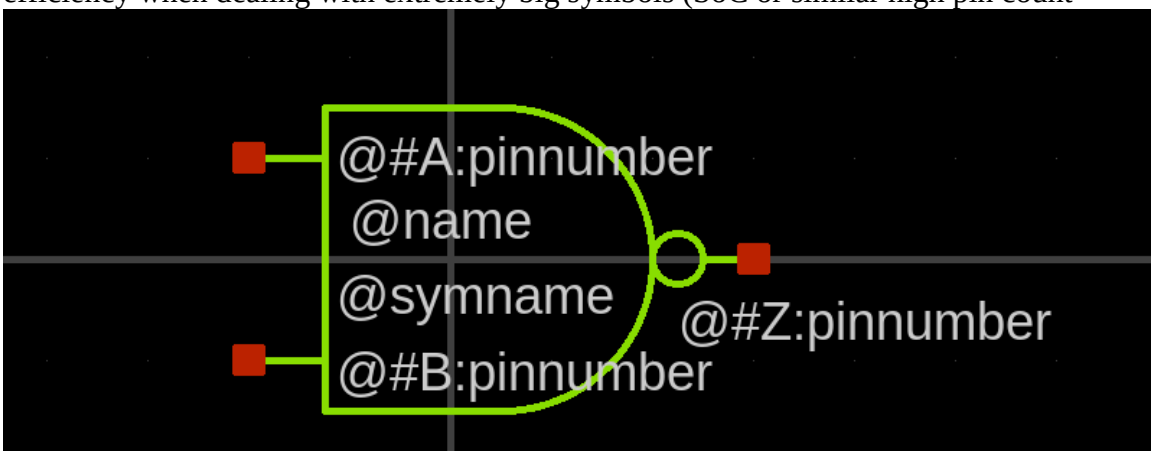
13. We want now to place some text near the gate pins to display the pin number: again, use



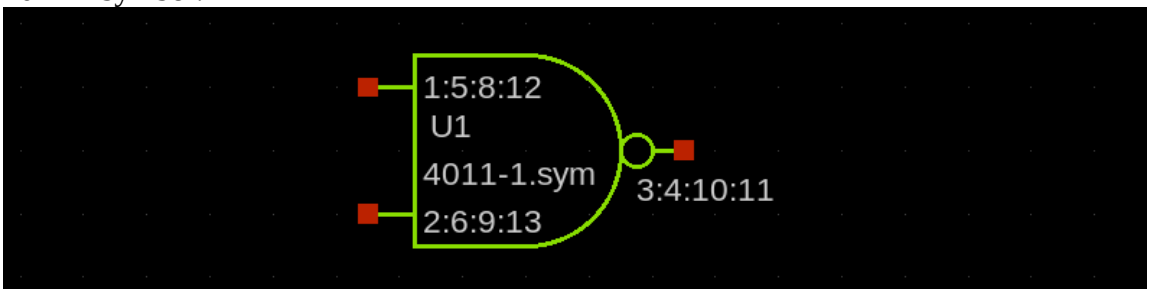
The complicated syntax of these text labels has the following meaning:

- 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.

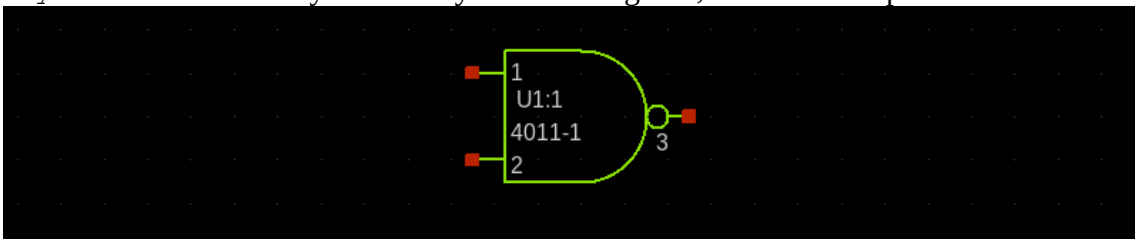
14. There is another syntax that can be used to display pin numbers, instead of specifying the pin index in XSCHEM list (that reflects the creation order) you can reference pins by their name; The only reason to use the previous syntax with pin index numbers is efficiency when dealing with extremely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



15. 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:

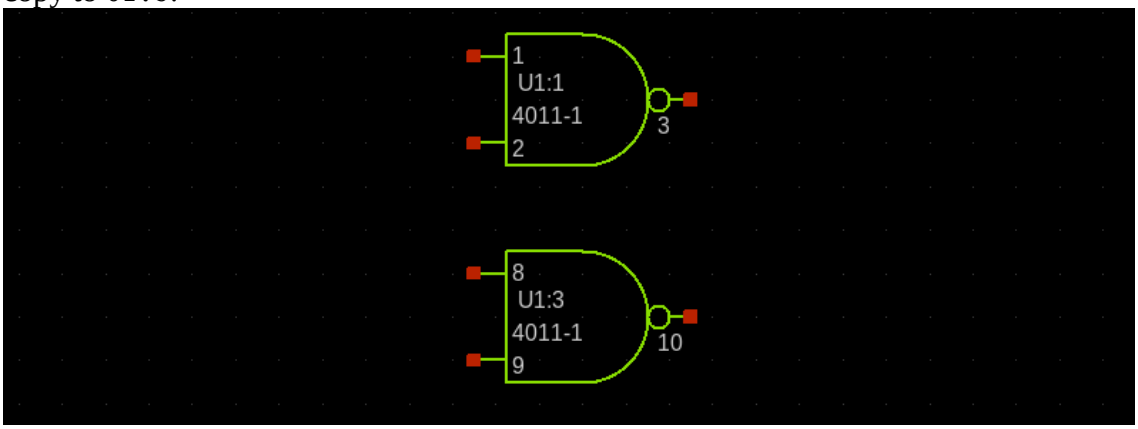


We see that all pin numbers are shown for each pin; this reminds us that this is a slotted device! slotted devices should specify the slot number in the instance `name` so, select the component, press `q` and change the `U1` name attribute to `U1:1`. You can also remove the `.sym` extension in the 'Symbol'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

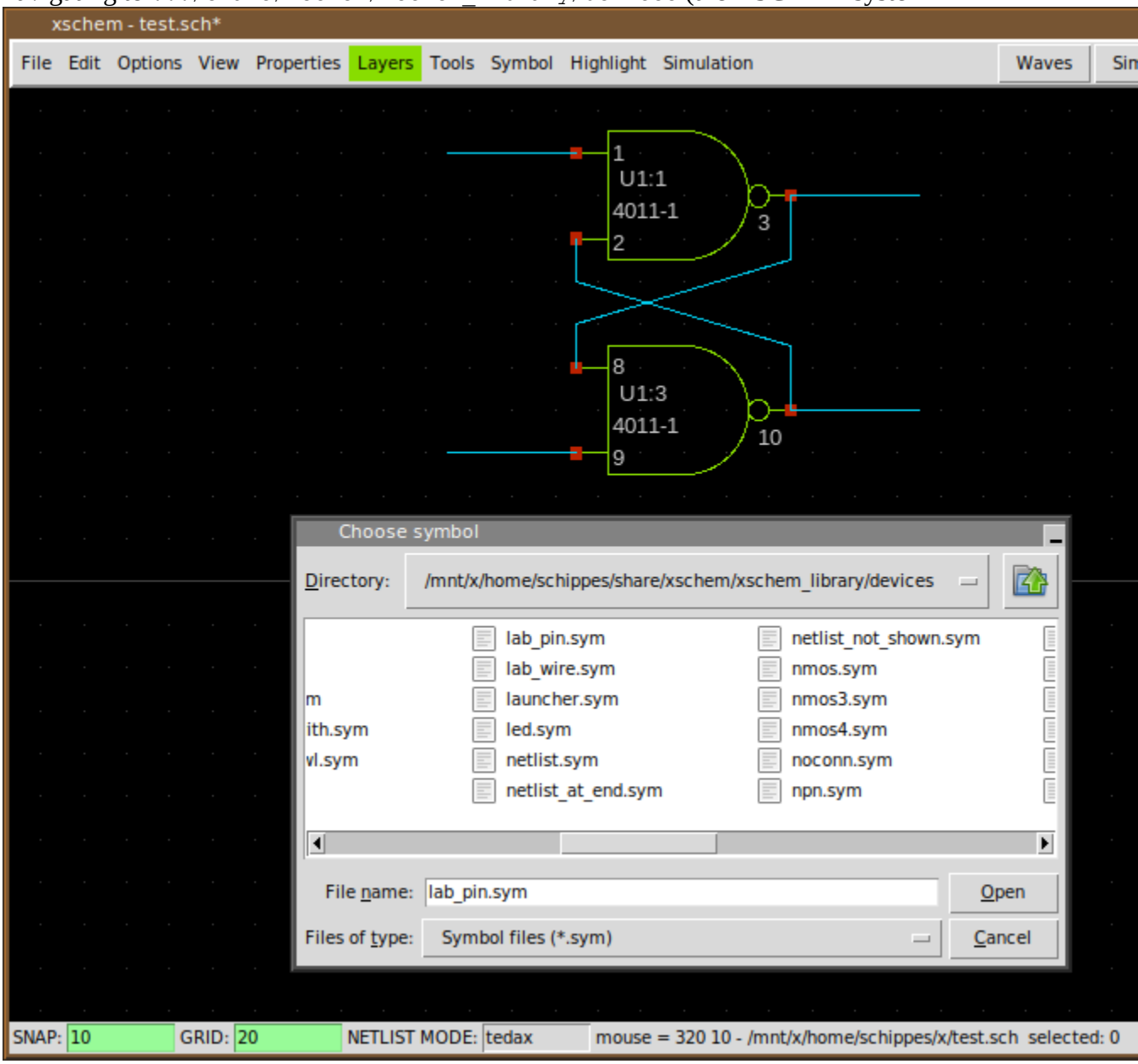


As you can see now the slot is resolved and the right pin numbers are displayed. Now select and copy the component (use the `c` key), and change the `name` attribute of the new

copy to `U1:3`:

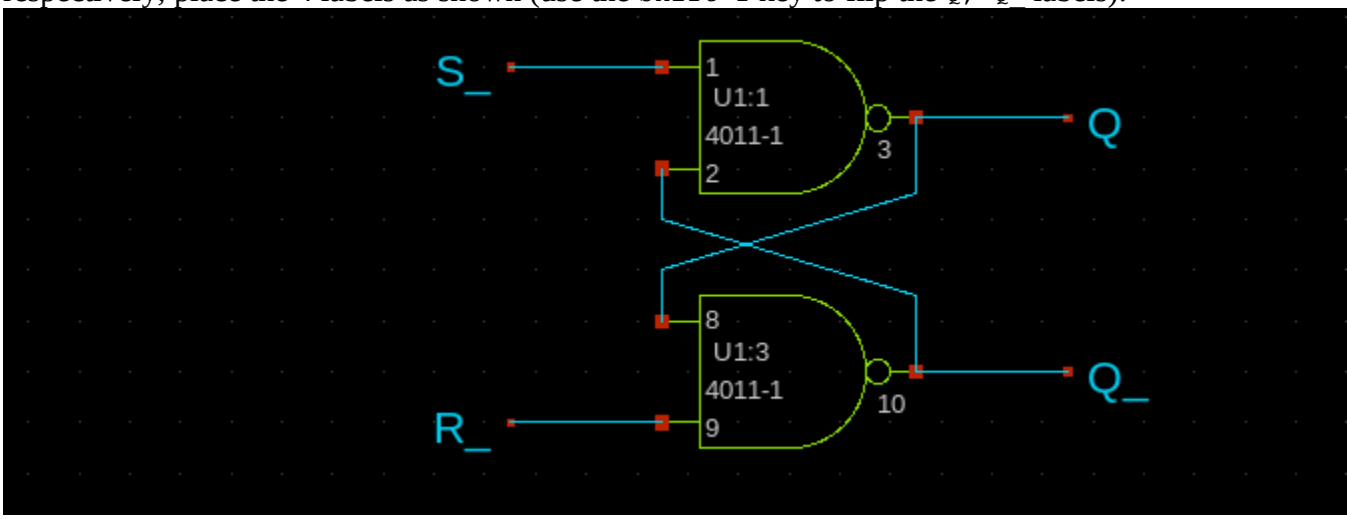


16. Now draw some wires, for example to create an SR latch as shown, use the `w` key to draw wires; when done with the wiring insert a net label by pressing the `Insert` key and navigating to `. . ./share/xschem/xschem_library/devices` (the XSCHEM system

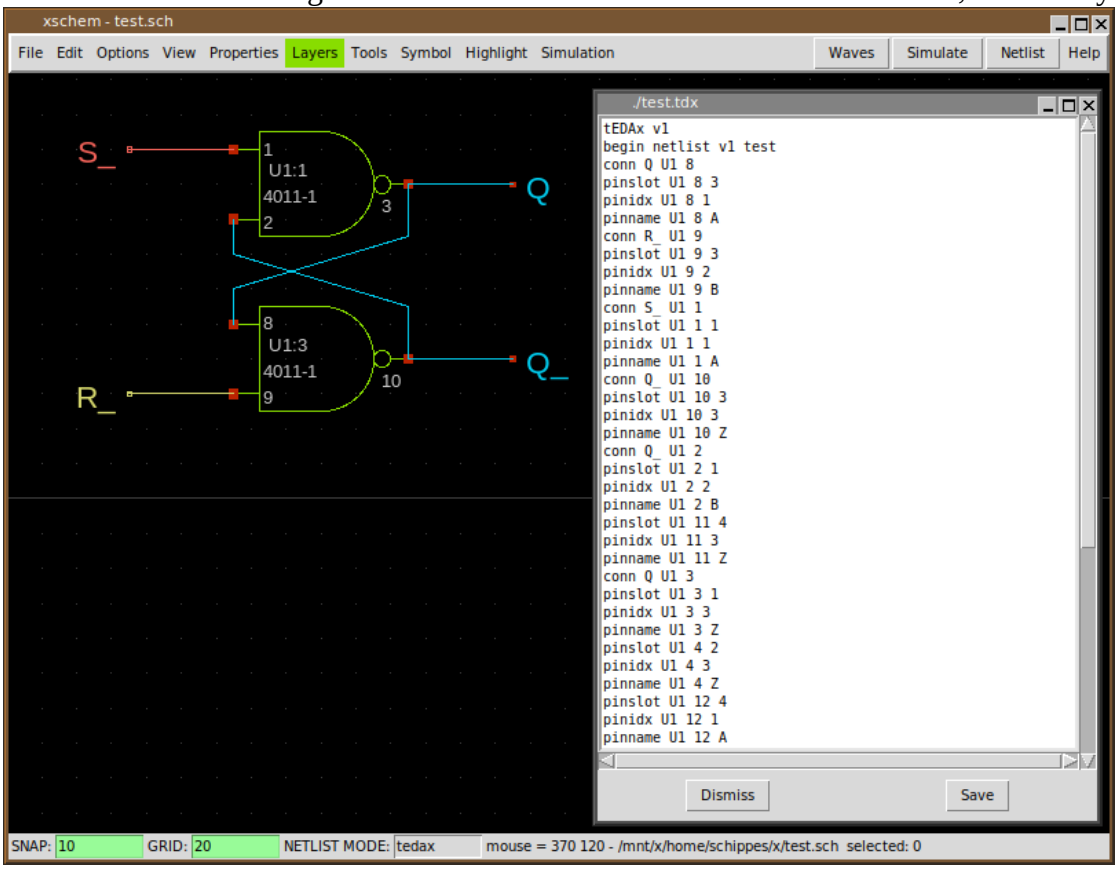


Place 4 of these `lab_pin` symbols and set their `lab` attribute to `S_, R_, Q, Q_`

respectively; place the 4 labels as shown (use the `Shift-f` key to flip the `Q, Q_` labels):



17. 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



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
```

```
39.  pinname U1 2 B
40.  pinslot U1 11 4
41.  pinidx U1 11 3
42.  pinname U1 11 Z
43.  conn Q U1 3
44.  pinslot U1 3 1
45.  pinidx U1 3 3
46.  pinname U1 3 Z
47.  pinslot U1 4 2
48.  pinidx U1 4 3
49.  pinname U1 4 Z
50.  pinslot U1 12 4
51.  pinidx U1 12 1
52.  pinname U1 12 A
53.  pinslot U1 13 4
54.  pinidx U1 13 2
55.  pinname U1 13 B
56.  pinslot U1 5 2
57.  pinidx U1 5 1
58.  pinname U1 5 A
59.  conn VCC U1 14
60.  pinname U1 14 power
61.  pinslot U1 6 2
62.  pinidx U1 6 2
63.  pinname U1 6 B
64.  conn GND U1 7
65.  pinname U1 7 ground
66.  footprint U1 dip(14)
67.  device U1 CD4011B
68.  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.

FAQ

When placing a new component i want a dialog showing the defined libraries before opening the TCL file selector

Add this to your xschemrc file:

```
set use_list_dirs 1
```

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

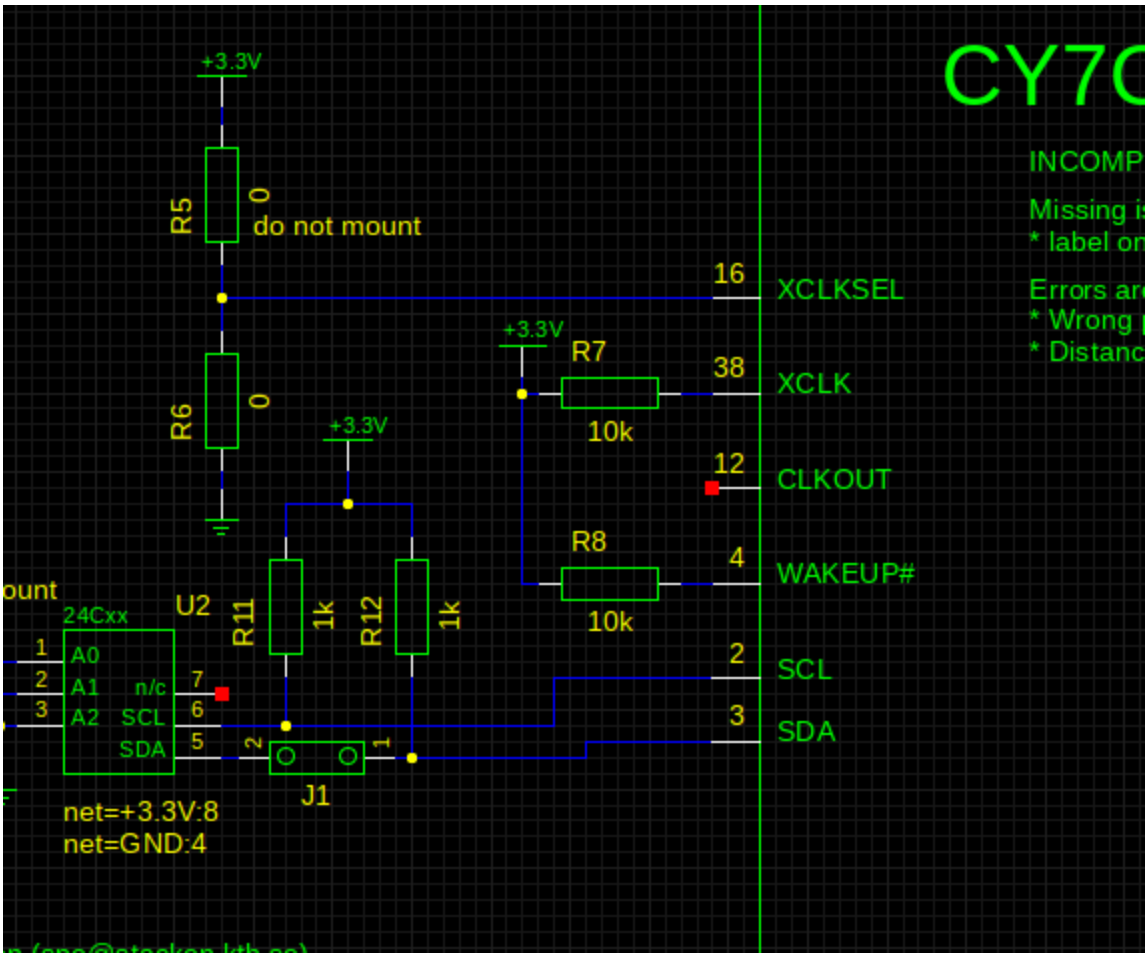
Add this to your xschemrc file:

```
set disable_unique_names 0
```

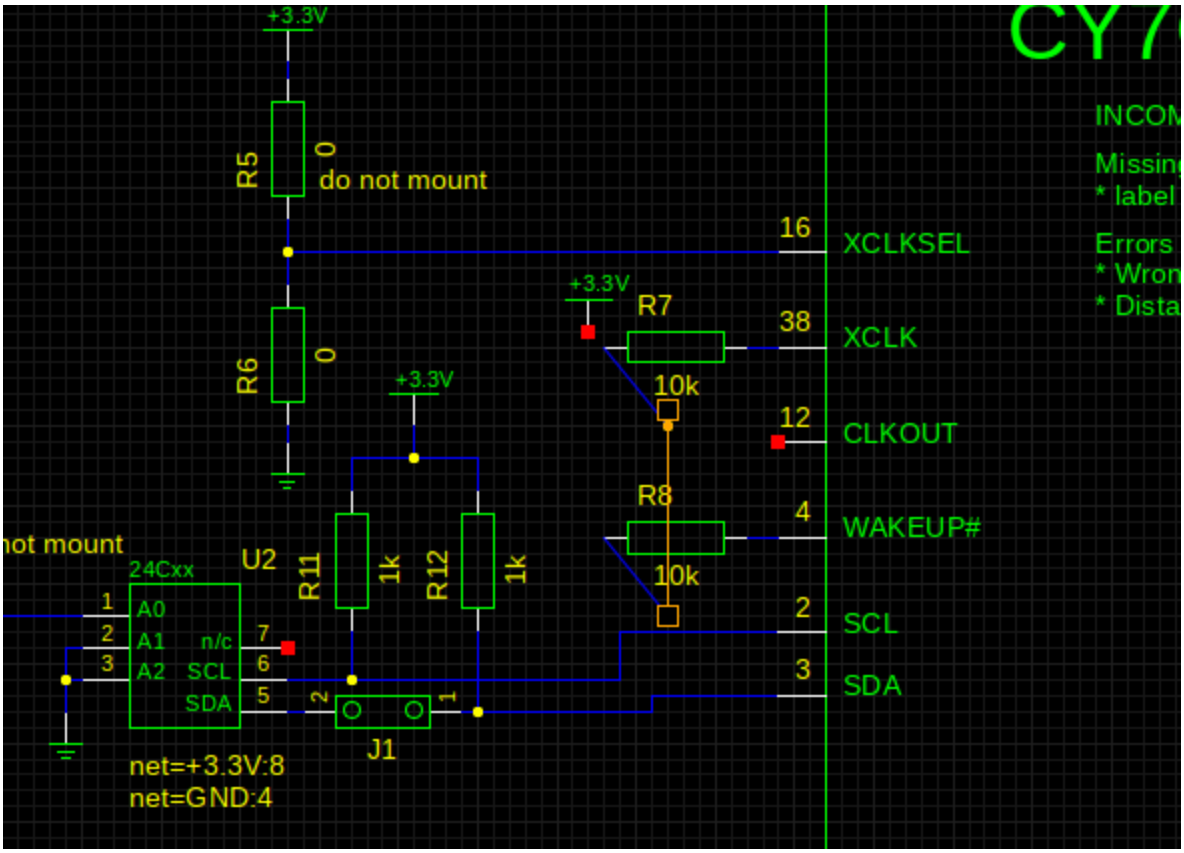
By default XSCHEM allows instance name (Refdes) duplicates in the schematic. This must be resolved by the user normally, before exporting any netlist. The `Hilight - Highlight duplicate instance names` (`k` key) menu entry can be used to mark the components that need to be renamed. The `Highlight - Rename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