Using Magic VLSI to layout and simulate a Skywater 130nm CMOS inverter

1. Integrate skywater into magic

Since the skywater tech files are not installed in magic's library, we need to create a symbolic link to use the tech files for drawing layout.

```
sudo ln -s $PDK_ROOT/sky130A/libs.tech/magic/* $CAD_ROOT/magic/sys/
```

where:

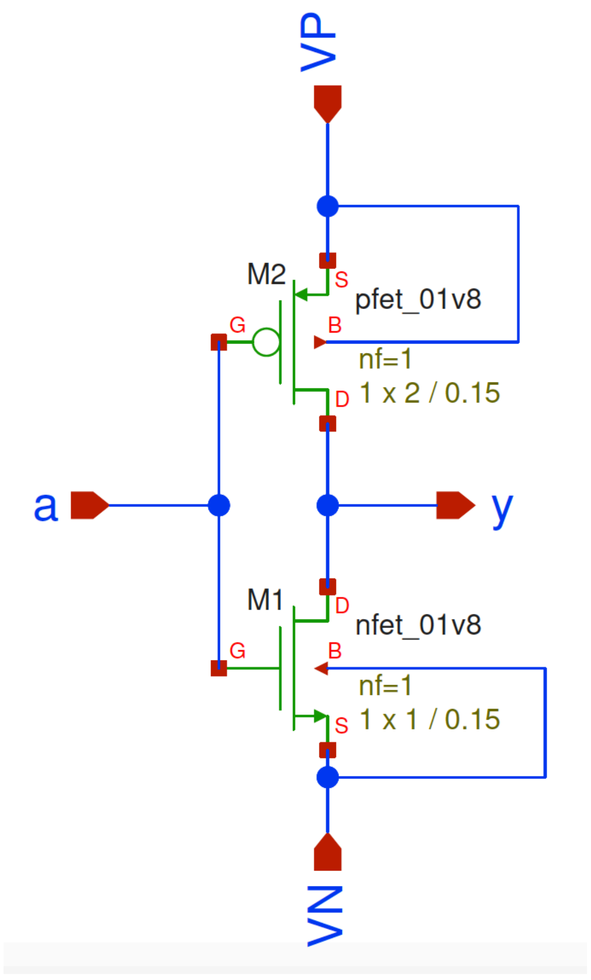
`$PDK_ROOT` is `/home/talarico/share/pdk`
`$CAD_ROOT` is `/home/talarico/opt/magic/lib`

2. Create a working directory

```
mkdir -p /home/talarico/ihome/ngs406/layout/tutorial_sky130
cd /home/talarico/ihome/ngs406/layout/tutorial_sky130
```

3. Create the schematic and the symbol of the inverter using xschem

```
xschem inv.xschem
```

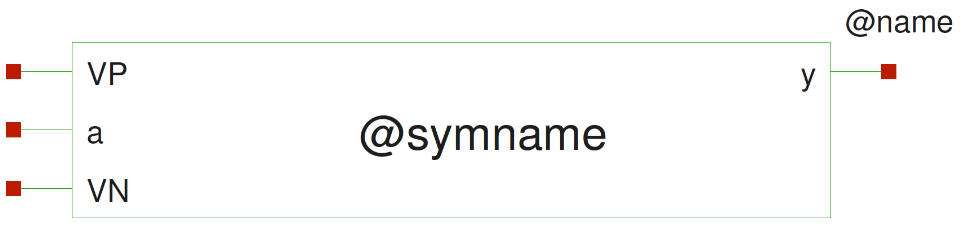


Once done drawing the schematic, the easiest way to generate the symbol is through the menu:
`Symbol > Make symbol from schematic`

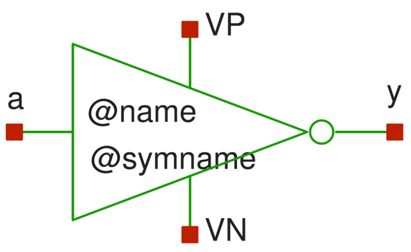
The spice netlist `inv.spice` associated to the schematic `inv.sch` is:

```
**.subckt inv y a VP VN
*.opin y
*.ipin a
*.ipin VP
*.ipin VN
XM1 y a VN VN sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2)
* W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 /
W'
+ sa=0 sb=0 sd=0 mult=1 m=1
XM2 y a VP VP sky130_fd_pr__pfet_01v8 L=0.15 W=2 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2)
* W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 /
W'
+ sa=0 sb=0 sd=0 mult=1 m=1
**.ends
.end
```

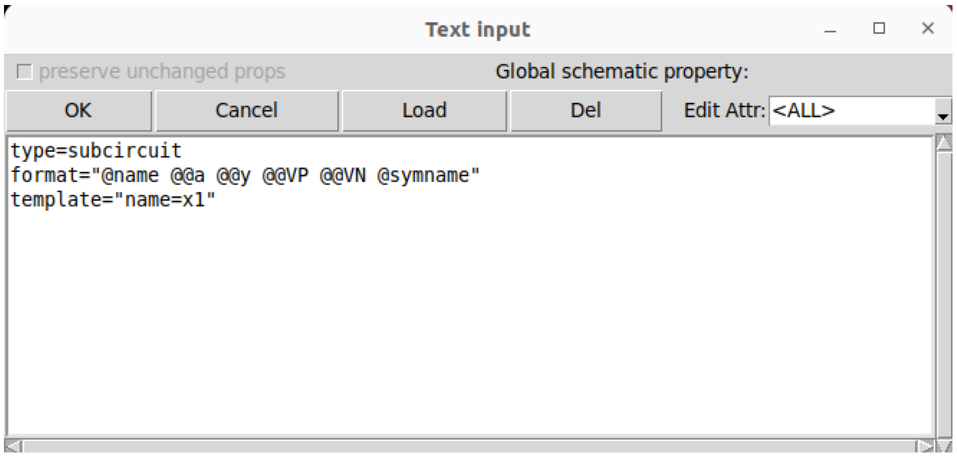
The next step is to open the symbol automatically generated:
`xschem inv.sym`



and edit its graphics. All graphic elements can be modified with the usual commands: `Edit > Move Objects` (shortcut `M`), `Edit > Flip selected objects` (shortcut `Alt+F`), `Edit > Rotate selected objects` (shortcut `Alt+R`). If needed, change the grid spacing and the snap value (`View > Grid spacing`, `View > snap value`). The grid can be toggled on/off using the command `Options > Draw grid` (shortcut `%`). The properties associated to the parts of the symbol (including fonts) can be edited by first selecting the part and then right clicking on it and selecting Edit attributes or more simply using the command `Properties > Edit` (shortcut `Q`)



Finally, edit the global symbol properties by click anywhere outside the symbol and select `Properties > Edit` (shortcut `Q`). This last step is very important, because it allows to specify the order of the ports of the associated subcircuit.

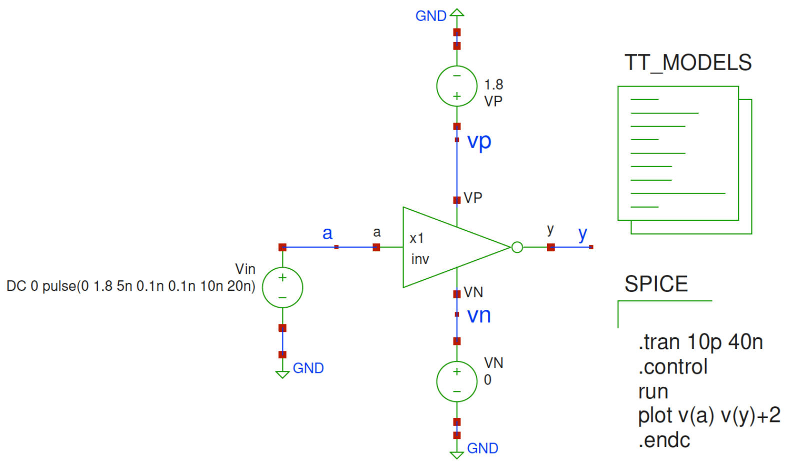


```
type=subcircuit
format="@name @@a @@y @@VP @@VN @symname"
template="name=x1"
```

4. Simulate the inverter with ngspice

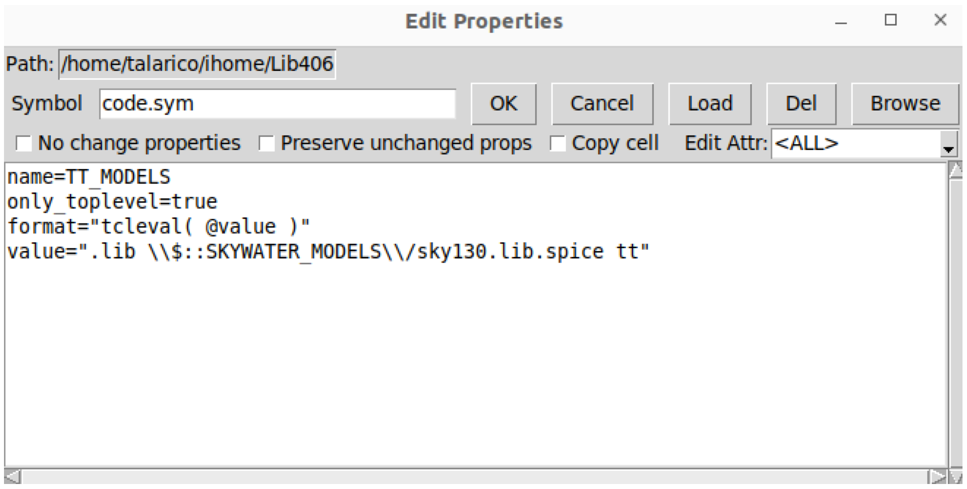
The most modular approach is to run the simulation through a testbench. It is possible to create the testbench directly as a gspice netlist or through xschem.

```
xschem tb_inv.sch
```



```
.tran 10p 40n
.control
run
plot v(a) v(y)+2
.endc
```

With the attributes of symbol code.sym set as follows:(see Appendix):



```
name=TT_MODELS
only_toplevel=true
format="tcleval( @value )"
value=".lib \\$::SKYWATER_MODELS\\/sky130.lib.spice tt"
```

The spice netlist associated to the testbench is `tb_inv.spice`:

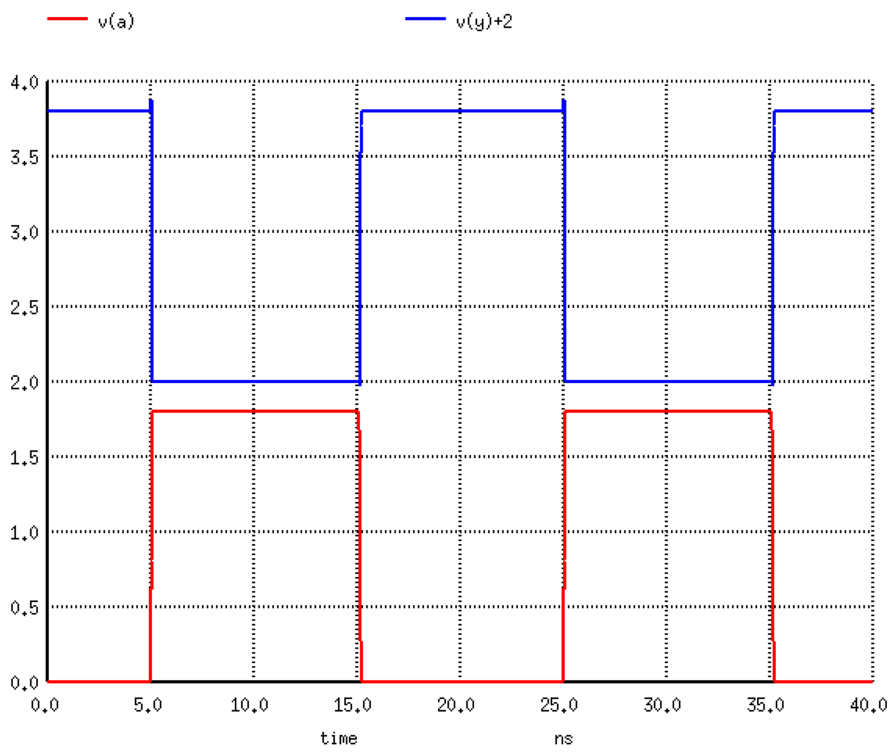
```
**.subckt tb_inv
x1 a y vp vn inv
Vin a GND DC 0 pulse(0 1.8 5n 0.1n 0.1n 10n 20n)
VN vn GND 0
VP vp GND 1.8

**** begin user architecture code
.lib /home/talarico/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.tran 10p 40n
.control
run
plot v(a) v(y)+2
.endc

**** end user architecture code



* expanding   symbol:  inv.sym # of pins=4
* sym_path: /home/talarico/ihome/ngs406/layout/tutorial_sky130/inv.sym
* sch_path: /home/talarico/ihome/ngs406/layout/tutorial_sky130/inv.sch
.subckt inv  a  y  VP  VN
*.opin y
*.ipin a
*.ipin VP
*.ipin VN
XM1 y a VN VN sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2)
* W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 /
W'
+ sa=0 sb=0 sd=0 mult=1 m=1
XM2 y a VP VP sky130_fd_pr__pfet_01v8 L=0.15 W=2 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2)
* W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 /
W'
+ sa=0 sb=0 sd=0 mult=1 m=1
.ends

.GLOBAL GND
.end
```

Before running the simulation create a spice initialization file `.spiceinit` with the following content:

```
set ngbehavior=hsa
set ng_nomodcheck
set color0=white
set color1=black
set xbrushwidth=2
```

The simulation can be run from within xschem or through the CLI:

```
ngspice tb_inv.spice
```



5. Inverter Layout

Start magic as follows:

```
magic -rcfile sky130A.magicrc
```

The default grid is a bit too "fine" for the drawn channel length (150nm) of the sky 130 process, so change it as follows:

```
% grid 50nm 50nm
```

Make the cursor snap to the user grid just defined

```
% snap user
```

Draw the gate of the transistors (poly).

Draw drain and source (n-diffusions) of the nMOS transistor (ndiff)

Extend the poly hangover over the n-diffusion to fix the DRC errors

Create the drain and source terminals of the nMOS transistor. Place local interconnect (li) in the diffusions and contact (ndc) the n-diffusions with the local interconnects.
Create the body-tap for the nMOS transistor. Place p-substrate diffusion (psd) and local interconnect (li) adjacent to the source of the nMOS transistor, and then contact (psc) together the psd and the li.

Draw the pMOS transistor. The easiest approach is to copy the nMOS transistor (including the body tap) and have magic turn it into a pMOS by surrounding the copy with n-well (nwell). The ndiff is turned into pdiff, the ndc is turned into pdc, the psd is turned in nsd, and the psc is turned into nsc.

Create the ground and power rails (metal1).

For convenience, the exact commands to build the layout up to this point have been collected in a .tcl script (`buildInvLayout.tcl`)

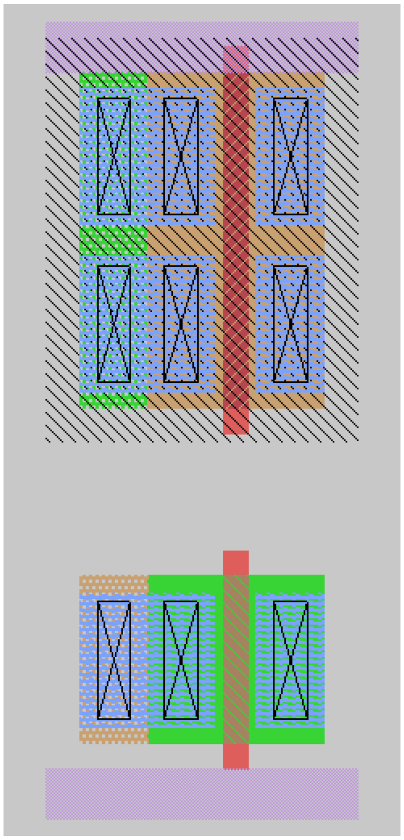
```
# File: Build_InvLayout.tcl

# set grid and snap
grid 50nm 50nm
snap user

# set crosshair at origin and draw gate of transistors (poly)
box size 0 0
box position 0 0
box size 150nm 1000nm
paint poly

# set crosshair at origin and draw drain and source diffusions (ndiff)
box size 0 0
box position 0 0
box position -450nm 0
box size 1050nm 1000nm
paint ndiff

# extend the poly over the diffusions to fix DRC errors
box size 0 0
box position 0 0
box size 150nm -150nm
paint poly
box size 0 0
box position 0 0
box position 0 1000nm
box size 150nm 150nm
paint poly

# create the drain and source terminals of the nMOS transistor.
# place local interconnect (li) in the diffusions and then contact the
# n-diffusions with the local interconnect (ndc).
# drain side:
box size 0 0
box position 200nm 100nm
box size 400nm 800nm
paint li
box size 0 0
box position 300nm 150nm
box size 200nm 700nm
paint ndc
# source side:
box size 0 0
box position 200nm 100nm
box size 400nm 800nm
select area
copy w 650nm
# tap the body terminal of the nMOS transistor
# place psubstrate diffusion (psd) and local interconnect
# (li) and contact them together (psc)
box size 0 0
box position -850nm 0nm
box size 400nm 1000nm
paint psd
box size 0 0
box position -850nm 900nm
box size 400nm -800nm
paint li
box size 0 0
box position -750nm 850nm
box size 200nm -700nm
paint psc

# draw the pMOS transistor
# 1. select and copy the nmos transistor
```

```
box size 0 0
box position -900nm 1150nm
box size 1550nm -1300nm
select area
copy north 3000nm
# zoom to full view
view
# 2. double the width of the transistor
copy down 1000nm
# 3. Wrap n-well around --> magic turns the nMOS into a pMOS
box size 0 0
box position -1050nm 4200nm
box size 1850nm -2400nm
paint nwell

# draw ground and power rails (metal 1)
box size 0 0
box position -1050nm -150nm
box size 1850nm -300nm
paint metal1
select area
copy up 4450nm
```

To execute the script run the command:

```
% source buildInvLayout.tcl
```

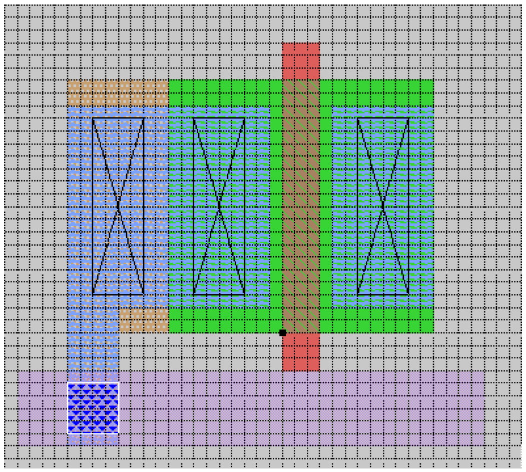
At this point the layout looks as follows:



The rest of the layout can be quickly completed interactively. In magic by default everything is a box. In BOX Tool mode (indicated by a crosshair cursor) the mouse button clicks are used as follows:

Left mouse button (LMB)	--> left bottom corner of box
Right mouse button (RMB)	--> right top corner of box
Middle mouse button (MMB)	--> paint the box with the same type of layer underneath the crosshair cursor

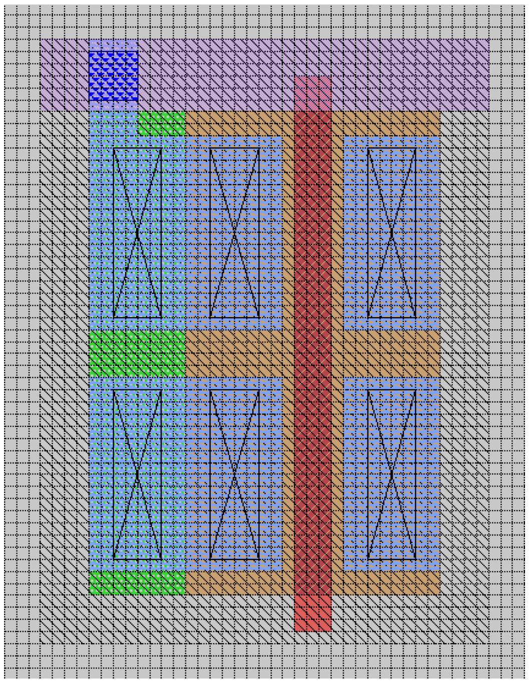
Connect the body of the nMOS transistor to the ground rail; place local interconnect (li) and then connect with `mcon` (a.k.a. `viali`) the local interconnect and the metal1 rail. Note: the body and the source of the nMOS transistor are already electrically connected (through local interconnect).



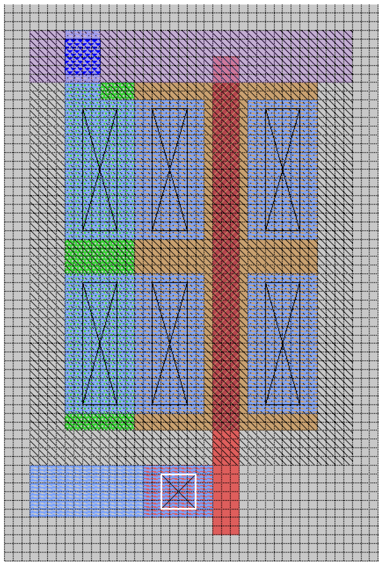
To verify the connectivity of various layers, place the crosshair on one of the layers of interest and then type `s` (macro for `select`) repeatedly.

Connect the body of the pMOS transistor to the power rail; place local interconnect (`li`) and then connect with `mcon` (a.k.a. `viali`) the local interconnect and the metal1 rail. Note: the body and the source of the pMOS transistor are already electrically connected (through local interconnect).

Extend the nwell to fix the DRC errors.

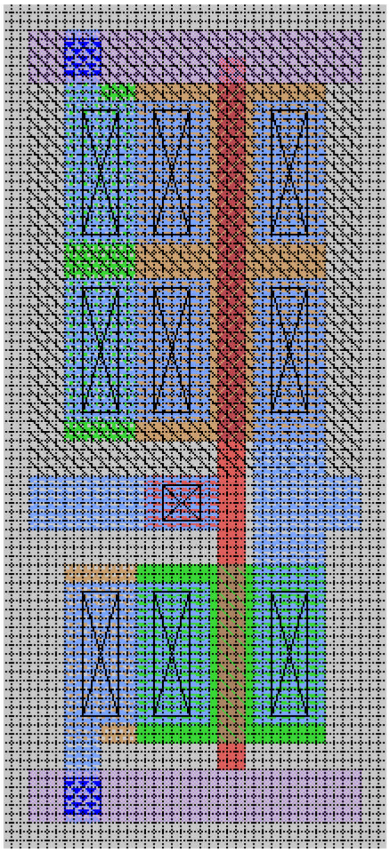


Create the input of the inverter. Extend the polysilicon (`poly`), add local interconnect (`li`), and contact (`pc`) the poly and the local interconnect:



Place a box on the area surrounding the nMOS transistor and the ground rail. Select the area through the command `% select area`. Move the selected area up and go as close as possible to the pMOS network. As long as the layout is DRC errors free, use repeatedly the command `% move up 1`.

Create the output of the inverter. Connect the drain of the pMOS transistor and the drain of nMOS transistor using local interconnect (li).



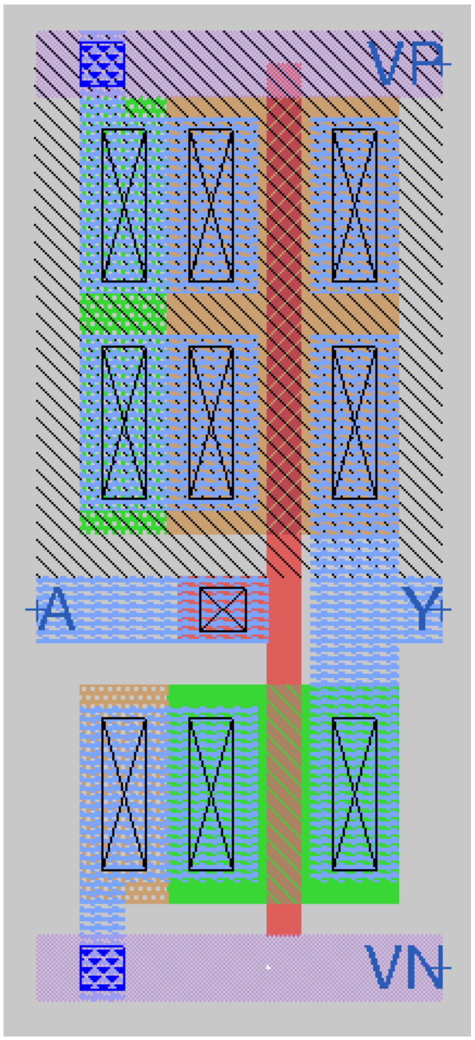
Finally label and define the input, output, power rail and ground rail of the inverter as ports. Associating ports to the labels facilitate doing hierarchical layouts. Put the crosshair on one terminal at the time and use the corresponding commands:

```
% label A w
% port make 1
% label Y e
% port make 2
% label VP w
% port make 3
% label VN w
% port make 4
```

Labels are yellow, and after making them ports become blue. The font is extremely small, so it may be useful to surround them with a box and then zoom box using the macro ^Z.
Magic's zoom commands are:
Zoom in – `z` (equivalent to `zoom 0.5`)
Zoom out – `Z` (equivalent to `zoom 2`)
Zoom Box – `^Z` (equivalent to `findbox zoom`)

Finally, save the layout:

```
% save magic_inv.mag
```



6. **Convert the layout into a spice netlist**
First, extract the circuit netlist associated to the layout using the command:

```
% extract all
```

The command `extract all` creates the intermediate format netlist file `magic_inv.ext.`

Second, apply all the setting appropriate for running LVS.

```
% ext2spice lvs
```

The settings for running LVS include:

```
% ext2spice hierarchy on
% ext2spice scale off
% ext2spice format ngspice
% ext2spice cthresh infinite
% ext2spice rthresh infinite
% ext2spice global off
% ext2spice blackbox on
% ext2spice subcircuit top auto
% ext2spice renumber off
```

Finally, covert the `.ext` netlist into a `.spice` netlist

```
% ext2spice -d
```

The `-d` option distribute junction areas and perimeters calculation per node rather than per device.

The spice netlist generated is `magic_inv.spice:`

```
* NGSPICE file created from magic_inv.ext - technology: sky130A

.subckt magic_inv A Y VP VN
X0 Y A VN VN sky130_fd_pr__nfet_01v8 ad=4.5e+11p pd=2.9e+06u as=4.5e+11p ps=2.9e+06u
w=1e+06u l=150000u
X1 Y A VP VP sky130_fd_pr__pfet_01v8 ad=9e+11p pd=4.9e+06u as=9e+11p ps=4.9e+06u w=2e+06u
l=150000u
.ends
```

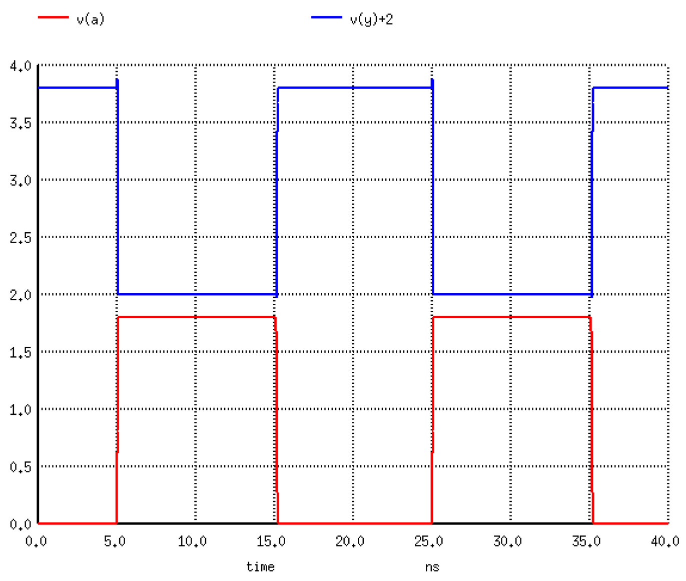
7. **Double check the spice netlist generated through magic (run the same simulation you have run on the schematic netlist)**

Create a spice testbench to verify the behavior of the spice netlist generated. The testbench file `tb_magic_inv.sp` looks as follows:

```
* tb_magic_inv.sp
.lib ~/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.include magic_inv.spice

Xinv A Y VP VN magic_inv

Vdd VP VN 1.8
Vgnd VN 0 0
Vin A VN DC 0 pulse(0  1.8 5n 100p 100p 10n 20n)
*                   V1  V2 TD  TR   TF  PW  PER
.tran 10p 40n
```

```
.control
run
plot v(A) v(Y)+2
.endc
```

```
ngspice tb_magic_inv.sp
```



8. Run LVS using netgen

The netlist to compare are `inv_xschem.cir`:

```
** inv_xschem.cir

x1 a y vp vn inv

.subckt inv  a  y  VP  VN
XM1 y a VN VN sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad='int((nf+1)/2) * W/nf * 0.29'
as='int((nf+2)/2) * W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 /
W' nrs='0.29 / W'
+ sa=0 sb=0 sd=0 mult=1 m=1
XM2 y a VP VP sky130_fd_pr__pfet_01v8 L=0.15 W=2 nf=1 ad='int((nf+1)/2) * W/nf * 0.29'
as='int((nf+2)/2) * W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 /
W' nrs='0.29 / W'
+ sa=0 sb=0 sd=0 mult=1 m=1
.ends
.end
```

and inv_magic.cir:

```
** inv_magic.cir
.include magic_inv.spice
Xinv A Y VP VN magic_inv

.end
```

where `magic_inv.spice` is the spice netlist generated via magic:

```
* NGSPICE file created from magic_inv.ext - technology: sky130A

.subckt magic_inv A Y VP VN
X0 Y A VN VN sky130_fd_pr__nfet_01v8 ad=4.5e+11p pd=2.9e+06u as=4.5e+11p ps=2.9e+06u
w=1e+06u l=150000u
X1 Y A VP VP sky130_fd_pr__pfet_01v8 ad=9e+11p pd=4.9e+06u as=9e+11p ps=4.9e+06u w=2e+06u
l=150000u
.ends
```

Run netgen's lvs interactively:

```
netgen
% lvs inv_magic.cir inv_xschem.cir
```

The lvs passes (`circuits match uniquely`) but there are property errors.

```
...
Final result:
Circuits match uniquely.
Property errors were found.

The following cells had property errors:
 inv_magic.cir
...
```

The detailed results of the lvs comparison are saved in the file: `comp.out`

The reason for the property errors is that in the xschem's spice netlist `inv_xschem.cir` many of the MOS quantities are given in parametric form rather than as absolute values.
The PDK provides a tcl script to solve this issue. The script is located at:
`~/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl`

To take advantage of this script, instead of running netgen and lvs interactively we run netgen in batch mode:

```
netgen -batch lvs inv_magic.cir inv_xschem.cir ~/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl
```

Since the command is quite lengthy, it is worth to put the command in a python script `LVS.py`.

```
#!/usr/bin/env python3
import os, sys

if len(sys.argv) != 3:
    print(' LVS: you must specify two netlist filenames to compare!')
    sys.exit(1)

os.system('netgen -batch lvs {} {} \
        ~/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl'.format(sys.argv[1],
sys.argv[2]))

sys.exit(0);
```

For convenience let's also make the script executable, and move it in a directory that is already part of the $PATH:

```
chmod ugo+x LVS.py
mv LVS.py LVS
mv LVS.py ~/scripts/.
```

This way we can execute the lvs comparison as follows:

```
LVS inv_magic.cir inv_xschem.cir

...
Circuit 1 contains 2 devices, Circuit 2 contains 2 devices.
Circuit 1 contains 4 nets,    Circuit 2 contains 4 nets.

Final result:
Circuits match uniquely.
...
```

9. **Parasitic extraction**
   Open the layout of the inverter and extract the spice netlist with parasitic.

```
magic
% tech load sky130A
```

Select the point in the Layout window where you want to load the layout of the cell, and then run parasitic extraction:

```
% load magic_inv.mag
% grid 5
% snap user

% extract all
% ext2spice hierarchy on
% ext2spice scale off
% ext2spice cthresh 0
% ext2spice -d -o postlayout.spice -f ngspice
```

Change the name of the extracted netlist to be a bit more descriptive.

```
mv postlayout.spice magic_inv_post.sp
```

The content of the extracted spice netilist `magic_inv_post.sp` is:

```
* NGSPICE file created from magic_inv.ext - technology: sky130A

.subckt magic_inv A Y VP VN
X0 Y A VN VN sky130_fd_pr__nfet_01v8 ad=4.5e+11p pd=2.9e+06u as=4.5e+11p ps=2.9e+06u
+ w=1e+06u l=150000u
X1 Y A VP VP sky130_fd_pr__pfet_01v8 ad=9e+11p pd=4.9e+06u as=9e+11p ps=4.9e+06u
+ w=2e+06u l=150000u
C0 A VP 0.15fF
C1 VP Y 0.16fF
C2 A Y 0.06fF
.ends
```

NOTE: the setting `ext2spice hierarchy off` makes the parasitic extraction more conservative

```
% extract all
% ext2spice hierarchy off
% ext2spice scale off
% ext2spice cthresh 0
% ext2spice -d -o postlayout.spice -f ngspice
```

In this case the content of the extracted spice netlist is:

```
* NGSPICE file created from magic_inv.ext - technology: sky130A

.subckt magic_inv A Y VP VN
X0 Y A VN VN sky130_fd_pr__nfet_01v8 ad=4.5e+11p pd=2.9e+06u as=4.5e+11p ps=2.9e+06u
+ w=1e+06u l=150000u
X1 Y A VP VP sky130_fd_pr__pfet_01v8 ad=9e+11p pd=4.9e+06u as=9e+11p ps=4.9e+06u
+ w=2e+06u l=150000u
C0 A VP 0.15fF
C1 Y A 0.06fF
C2 Y VP 0.16fF
C3 Y VN 0.27fF
C4 A VN 0.18fF
C5 VP VN 0.80fF
.ends
```

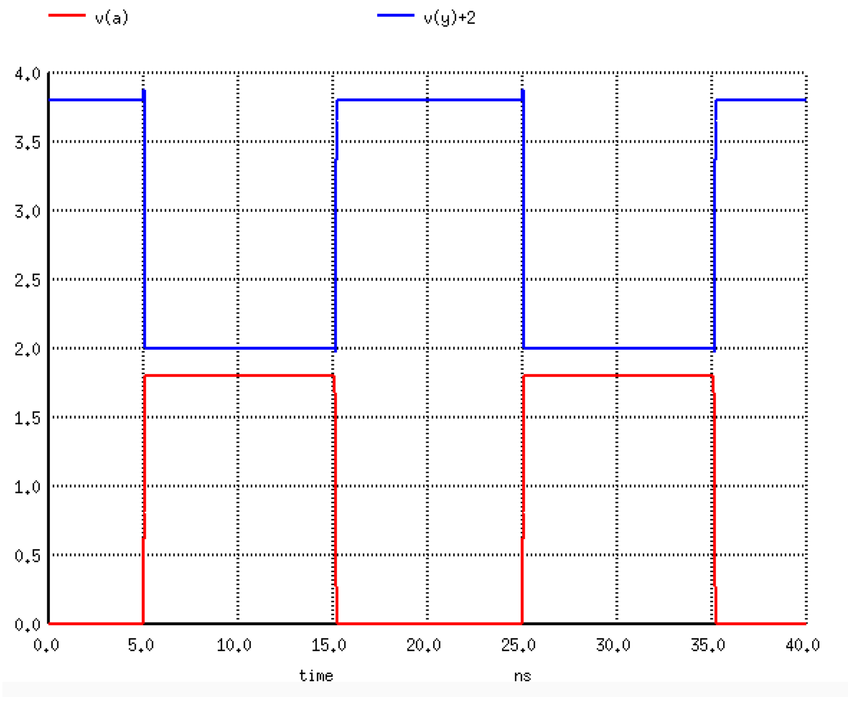
10. Post-layout simulation

Create a test harness around the netlist with parasitic and name it `tb_magic_inv_post.sp`:

```
* tb_magic_inv_post.sp
.lib ~/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.include magic_inv_post.sp
 Xinv A Y VP VN magic_inv
 Vdd VP VN 1.8
Vgnd VN 0 0
Vin A VN DC 0 pulse(0  1.8 5n 100p 100p 10n 20n)
*                   V1 V2  TD  TR   TF   PW PER
 .tran 10p 40n
 .control
  run
  plot v(A) v(Y)+2
.endc
```

And finally run the simulation:

```
ngspice tb_magic_inv_post.sp
```



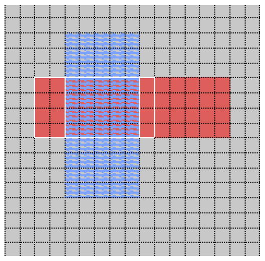
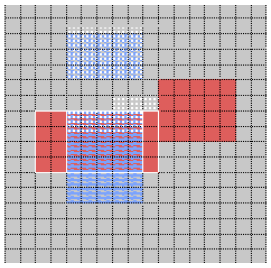
11. Frequently used BOX tool commands

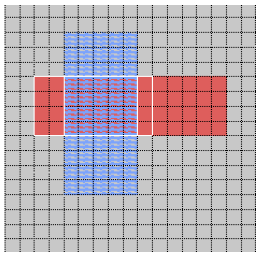
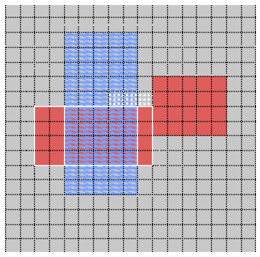
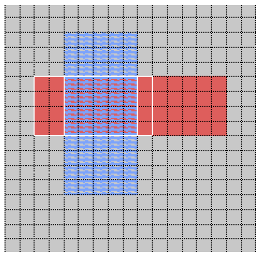
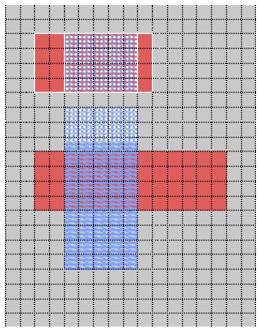
command	**macro**	**description**
	click on lmb	set/move left lower corner coordinate (llx, lly) of the box
	click on rmb	set/move right upper corner coordinate (urx, ury) of the box
	click on mmb	paint box with material under the cursor
`help`		Short description of all commands
`help` *string*		Short description of all commands containing string
`grid`	`g`	toggle grid on/off
`grid 0.05um 0.05um`		set the grid to 0.05um × 0.05um
`snap user`		cause the box to snap to the selected grid
`zoom` *amount*		zoom in/out by the factor *amount*
`zoom 0.5`	`z`	zoom (out) by a factor of 1/2
`zoom 2`	`Z`	zoom (in) by a factor of 2
	`^Z`	box zoom in
`view`	`v`	zoom window out so everything is visible
`macro help`		shows all available macros
`tech layer`		lists all available layers
`paint` *layer*		Paint the defined box with the specified layer
`select`	`s`	Select region under the cursor. Toggles between connected regions under the cursor
`select area`	`a`	select regions in area under box
`select cell`	`i`	select topmost cell in the window
`select less cell`	`^I`	select topmost cell in the window
`select clear`	`,`	clear out selections
`what`		Tells what's in the selected area
`erase` *layer*		erase layer under box
`erase`		erase everything under box
`delete`	`d`	erase everything under selected area
	`^D`	delete material**s** inside box that match what's under the cursor
`undo`	`u`	undo
`redo`	`U`	redo
`findlabel` *label*		set box to the location of *label*
`erase label`		erase the label(s) in the box
`label` *string*		create label at box location
`label B FreeSans 8`		create label B with font FreeSans and size 8 (*)
`label B w`		create label B with justification to the west of the box (*)
`port make` *index*		make the label under the cursor box a port and assign to it the desired numerical *index*
`save`		save the current edit on disk NOTE: save does not descend the hierarchy
`save` *filename*		save the current edit on disk to *filename*.mag NOTE: save does not descend the hierarchy
`quit`		exit magic
`tech load` *techname*		load specified *techname* (e.g., *sky130A*)
`tech layers`		list all the layers in the technology
`load` *filename.mag*		load the cell *filename.mag* in the magic window
	Arrow Keys	"panning" the layout view
`box size`		return box size (in lambda units)
`box position`		return the box llx, lly coordinates (in lambda units)
`box`	`b`	shows box dimensions in all available units
`drc find` *[nth]*		locate next or (*n-th*) error in the layout
`drc why`	`?`	print out reasons for drc error under box

`move`	m	move selected area to the crosshair cursor position
`move` *direction*		move selected area of 1 unit of distance in the defined direction. Distances are normally interpreted as *lambda* values. However, this depends on the last use of the **snap** command; **snap internal** changes the interpretation of units to the internal grid, while **snap user** changes the interpretation of units to the user grid. The units of distance can be specified explicitly by appending to the distance the value i or l or um (with no intervening spaces). The material left behind by the move operation is erased.
`move` *direction distance*		move selected area of *distance* in the defined *direction.* Distances are normally interpreted as *lambda* values. However, this depends on the last use of the **snap** command; **snap internal** changes the interpretation of units to the internal grid, while **snap user** changes the interpretation of units to the user grid. The units of distance can be specified explicitly by appending to the distance the value i or l or um (with no intervening spaces). The material left behind by the move operation is erased.
`stretch` *direction*		the **stretch** command moves the current selection from its current position according to the command arguments and fills in behind with material such that electrical connections are maintained across the area in between the original and final positions. (**)
`stretch` *direction distance*		
`copy`	c	copy selected area to the crosshair cursor position
`copy` *direction*		copy selection at unit distance and specified *direction* from coordinate (llx, lly)
`copy` direction distance		copy selection at *distance* and specified *direction* from coordinate (llx, lly)
`sideways`	f	flip selection and box sideways
`upsidedown`	F	flip selection and box upside down
`rotate`	r	rotate the selection and box clockwise of 90 degrees
`rotate` *[+/-] degrees*		rotate the selection and box by [+/-] *degrees* (the value of *degrees* must be 90, 180 or 270)
`clockwise`	r	it is the same as rotate
`clockwise` *degrees*		rotate the selection and box clockwise by *degrees* (the value of *degrees* must be 90, 180 or 270)

(*) The easiest way to set labels is to use the menu Edit > Text in the layout window
For executing a macro, the focus must be on the “Layout window”
For executing a command, the focus must be on the “tcl-tk console”

(**) Example:

	`% move down 2` 

	`% stretch down 2` 
	`% copy north 8` 

Magic consists of four tools. The space bar in the Layout Window allows to switch among the tools (BOX Tool, WIRING tool, NETLIST tool, and PICK tool).

To draw a layout without continuously incurring in DRC errors it is important to have a reasonable knowledge of the design rules. The design rules can be obtained directly from within magic using the `tech` command.

command	**macro**	**description**
`tech drc width` *layer*		Return the minimum allowed width for the indicated *layer*
`tech drc spacing` *layer1 [layer2]*		Return the minimum allowed spacing between *layer1* and *layer2*, if *layer2* is specified, or between *layer1* and itself, if not.

Example

```
% tech drc width poly
15
```

The value returned is in internal units.

For the sky130A the units are mapped as follows:

```
microns:   0.010 x 0.010
lambda:     1.00 x 1.00
```

The command `tech lambda` shows the internal units per lambda unit:

```
% tech lambda
```

1 1

1 lambda unit corresponds to 1 internal unit

Since 1 internal unit corresponds to 1 lambda unit and 1 lambda unit corresponds to 0.01um (=10nm), the min width of the poly is 150nm

```
% tech drc width nwell
84
```

```
% tech drc spacing li li
17
```

```
% tech drc spacing ndiff nwell
34
```

If needed, is possible to scale the ratio between internal units and lambda units using the `scalegrid` command

```
% scalegrid a b
```

After the command is executed *a* lambda units corresponds to *b* internal units.

```
scalegrid 1 2
```

means 2 internal units per lambda.

Grid scaling is interpreted relative to the current scale, so

```
scalegrid 1 2
scalegrid 1 2
```

results in 4 internal units per lambda.
The grid scaling can be queried using the **tech lambda** command.

12. Appendix

Whenever we put together an xschem's testbench for a circuit using the sky130 process, we need to include the libraries where the device models are located. Rather than including a code.sym component and editing it by hand, it's more convenient to build a specific symbol (e.g., TT_models). The fastest way to build a new symbol is to copy an existing symbol that does something similar and then modify it with a text editor rather than through xschem.

Here is the content of the symbol TT_models.sym:

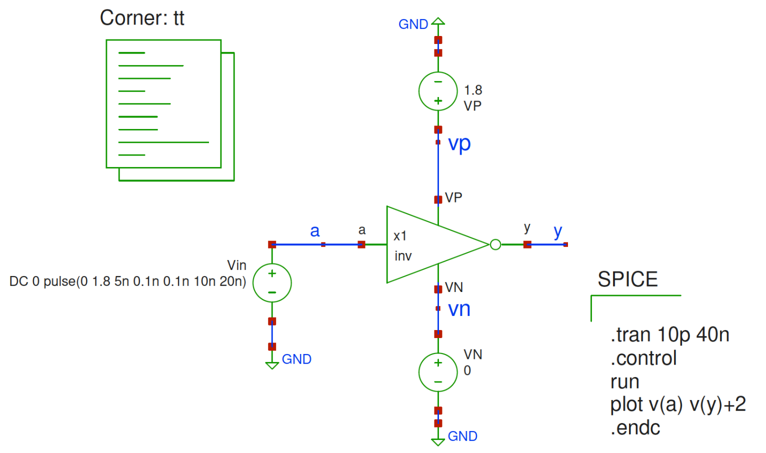
```
v {xschem version=3.1.0 file_version=1.2 }
G {}
K {type=netlist_commands
template="name=TT_models only_toplevel=false"
format="tcleval(
\\.lib $::SKYWATER_MODELS/sky130.lib.spice tt\\
)"
}

V {}
S {}
E {}
L 4 20 30 60 30 {}
L 4 20 40 40 40 {}
L 4 20 50 60 50 {}
L 4 20 60 50 60 {}
L 4 20 70 50 70 {}
L 4 20 80 90 80 {}
L 4 20 90 40 90 {}
L 4 20 20 70 20 {}
L 4 20 10 40 10 {}
L 4 100 10 110 10 {}
L 4 110 10 110 110 {}
L 4 20 110 110 110 {}
L 4 20 100 20 110 {}
L 4 100 0 100 100 {}
L 4 10 100 100 100 {}
L 4 10 0 10 100 {}
L 4 10 0 100 0 {}
T {@name} 5 -25 0 0 0.3 0.3 {}
```

Other common process corners for which we may want to run simulations include `ff`, `fs`, `sf` and `ss`.

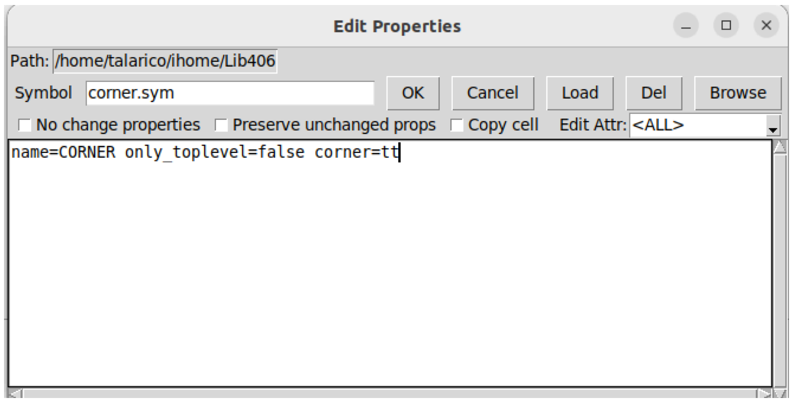
NOTE: xschem already provides an ad hoc symbol for this purpose: it is corner.sym. There is neither the need to use the symbol code.sym nor the need to create the symbol TT_models.

Example

The testbench test_inv.sch shows how to take advantage of corner.sym:



With the attributes of corner.sym set as follows:



Symbol: corner.sym

```
name=CORNER only_toplevel=false corner=tt
```

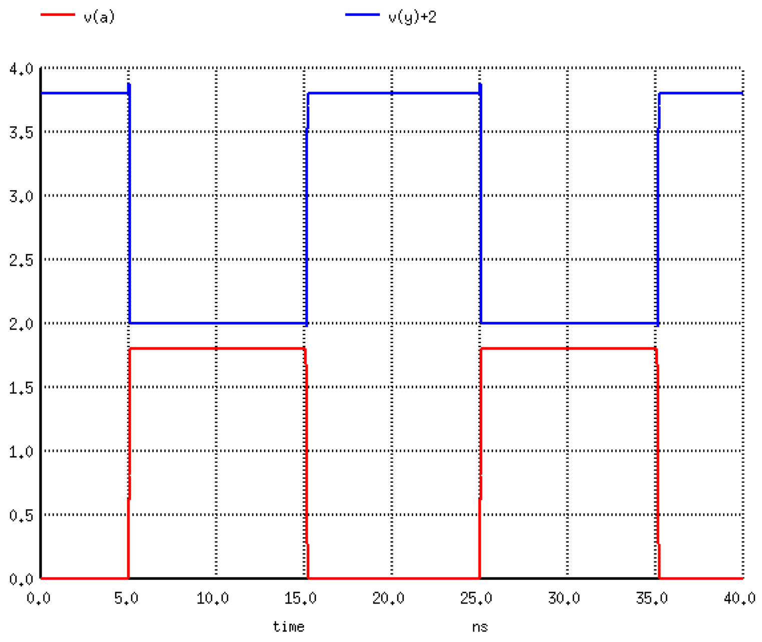
The corresponding spice netlist `test_inv.spice` is:

```
**.subckt test_inv
x1 a y vp vn inv
Vin a GND DC 0 pulse(0 1.8 5n 0.1n 0.1n 10n 20n)
VN vn GND 0
VP vp GND 1.8
**** begin user architecture code

.tran 10p 40n
.control
run
plot v(a) v(y)+2
.endc

.param mc_mm_switch=0
.param mc_pr_switch=0
```

```
.include /home/talarico/share/pdk/sky130A/libs.tech/ngspice/corners/tt.spice
.include /home/talarico/share/pdk/sky130A/libs.tech/ngspice/r+c/res_typical__cap_typical.spice
.include /home/talarico/share/pdk/sky130A/libs.tech/ngspice/r+c/res_typical__cap_typical__lin.spice
.include /home/talarico/share/pdk/sky130A/libs.tech/ngspice/corners/tt/specialized_cells.spice

**** end user architecture code
**.ends
* expanding   symbol:  inv.sym # of pins=4
* sym_path: /home/talarico/ihome/ngs406/layout/tutorial_sky130/inv.sym
* sch_path: /home/talarico/ihome/ngs406/layout/tutorial_sky130/inv.sch
.subckt inv  a  y  VP  VN
*.opin y
*.ipin a
*.ipin VP
*.ipin VN
XM1 y a VN VN sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2) * W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 / W'
+ sa=0 sb=0 sd=0 mult=1 m=1
XM2 y a VP VP sky130_fd_pr__pfet_01v8 L=0.15 W=2 nf=1 ad='int((nf+1)/2) * W/nf * 0.29' as='int((nf+2)/2) * W/nf * 0.29'
+ pd='2*int((nf+1)/2) * (W/nf + 0.29)' ps='2*int((nf+2)/2) * (W/nf + 0.29)' nrd='0.29 / W' nrs='0.29 / W'
+ sa=0 sb=0 sd=0 mult=1 m=1
.ends


.GLOBAL GND

.end
```

Invoke the simulation from CLI:

```
ngspice test_inv.spice
```



References:

[1] Magic VLSI user's guide (Tim Edwards)
http://opencircuitdesign.com/magic/userguide.html

[2] CMOS Inverter VTC and Transient Simulation Tutorial Using Xschem and Ngspice (Bradley A. Minch, @bminch)
https://www.youtube.com/watch?v=bm3l21ExLOY

[3] Creating a Hierarchical Schematic in Xschem (Bradley A. Minch)
https://www.youtube.com/watch?v=BpPP2hE_eK8

[4] Creating a Hierarchical Layout in Magic Using the SKY130 PDK (Bradley A. Minch)
https://www.youtube.com/watch?v=RPppaGdjbj0

[5] Layout Versus Schematic Tutorial Using Netgen – Part 1 (Bradley A. Minch)
https://www.youtube.com/watch?v=NCaNF4EunYU&t=134s

[6] Layout Versus Schematic Tutorial Using Netgen – Part 2 (Bradley A. Minch)
https://www.youtube.com/watch?v=_xsZbaTBEEA&t=41s