Introduction to the Custom Design Flow: Building a standard cell

EE241 Tutorial 3
Written by Brian Zimmer (2013)

Overview

In Tutorial 1 (GCD: VLSI's Hello World), you used the digital design flow to place-and-route a pre-existing library of standard cells based on an RTL description. In Tutorial 2 (Using VLSI Flow Outputs), you place-and-routed a 4-to-16 decoder, imported the design into Cadence Virtuoso, and investigated the difference in timing and power measurements from the digital design tools, non-parasitic transistor-level simulation, and parasitic transistor-level simulation. In this tutorial, you will use Cadence Virtuoso to create your own standard cell that can be used in the digital design flow, and learn how to run Monte Carlo simulations in order to investigate the effects of variability.

The custom design flow is shown in Figure 1. You start with a schematic representation of the circuit, and run simulations to verify functionality and performance. Then, you layout your design, and run Design Rule Checks (DRC) to verify that the layout is manufacturable, and Layout-Versus-Schematic (LVS) to verify that your layout matches your schematic. Last, you run parasitic extraction on your design to account for the contribution of wires and vias, and run simulation again on the extracted design.

To analyze the effect that variability has on your design, you can run a Monte Carlo analysis, which simulates your circuit hundreds of times while varying process parameters according to expected process variability.

In order to use your standard cell in a digital flow, you will need to describe the cell's pre-computed timing characteristics (so it can be synthesized) and physical attributes (so it can be place-and-routed). We will use Synopsys Library NCX to automatically simulate the cell at different temperatures, voltages, and process corners for different input slopes and output loads. NCX will create a .lib file that can be compiled into a binary .db file for use in the flow. We will use Cadence Virtuoso to generate an abstract view that only contains pins and metal layers, export this to a textual .lef file, then use Synopsys Library Compiler to generate a binary Milkyway file for use in the flow.

Getting Started

Load the EE241 environment, then download a set of scripts that will allow you to characterize your own standard cell. On BWRC machines, replace ∼ee241 with /tools/designs/ee241 throughout this tutorial.

```
% source ~ee241/tutorials/ee241.bashrc
% cd /scratch/userA
% git clone ~ee241/tutorials/gen_stdcells
```

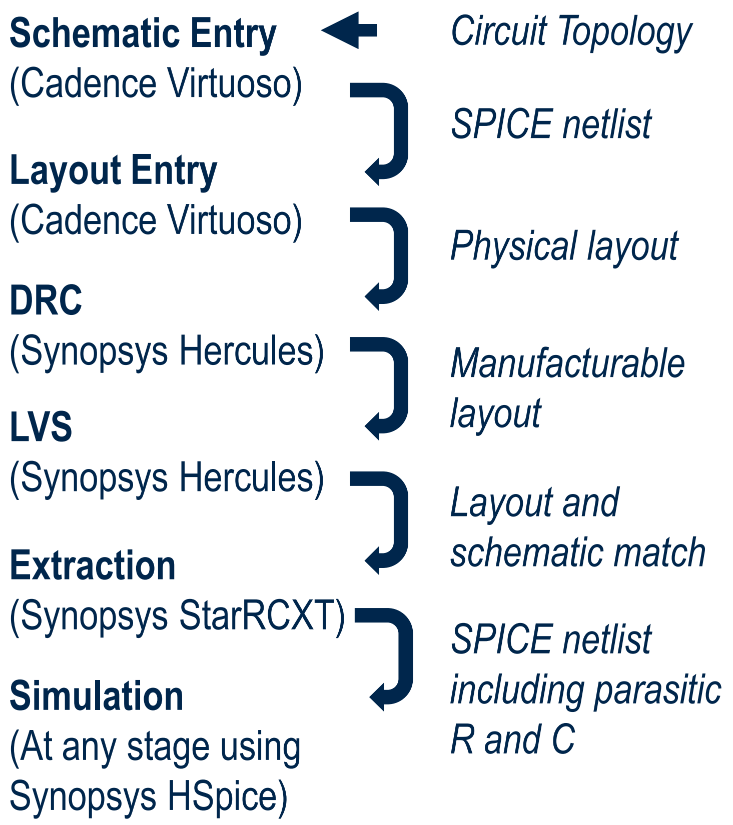


Figure 1: Custom design "flow."

Also, if you have not completed tutorial 1 and tutorial 2, you need to download a directory to run Virtuoso inside, and the decoder place-and-route directory.

```
% git clone ~ee241/tutorials/ee241_virtuoso
% git clone ~ee241/tutorials/decoder_analysis
```

Schematic Entry

Start Cadence Virtuoso.

```
% cd ee241_virtuoso
% virtuoso &
```

In Virtuoso, make a new library by going to File — New — Library. Call it (customcells), then select "Attach to an existing technology library" and choose SAED_PDK_32_28.

Then go to File — New — Cell, and name the cell NAND2X1B_RVT, and set the view to "Schematic."

First, add the pins to the file. Go to Create — Pins (or press p). Enter ”A1 A2” under ”Pin Names”, then click on the schematic to place them. Next, change the ”Direction” to ”Output” and place ”Z”. Last, change ”Direction” to ”inputOutput” and place ”VDD” and ”VSS.”

Now you need to enter the devices in the NAND2. Go to Create — Instance (or press i), set the Library to ”SAED_PDK_32_28”, the cell to ”nmos4t”, and the view to ”symbol.”

Change the width of each NMOS to 630n M (note: ”n M”, not ”nM”). Click in another box, and notice how the source/drain area and periphery change automatically. Virtuoso will try to guess parasitics from the source and drain before layout based on the length and width and design rules. Post-layout extraction will replace these values in the netlist (AS, AD, PS, PD) with the actual values. For example, in a NAND gate the diffusion for the NMOS can be shared, and no contact is needed, decreasing the parasitics from what the schematic would estimate.

Place two PMOS with a width of 760n. Now, go to Create — Wire, then click on the starting and ending point of a wire. If a diamond appears over the desired terminal, press ”s” to snap the wire to that location. Finish wiring up the design until it looks like Figure 2. Note that both NMOS bulk terminals must be connected to VSS. Go to File — Check and Save to finish.

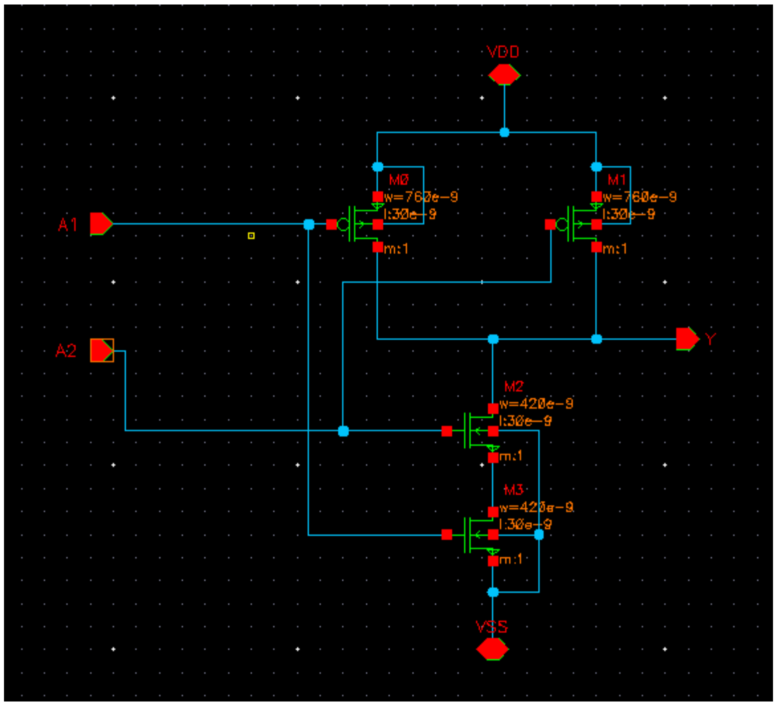


Figure 2: Schematic of the 2-input NAND gate.

Now make a symbol view so your cell can be instantiated in other designs and netlisted. Go to Create — Cellview — From Cellview, and click ok in the dialog box. You can leave the symbol as is, or use the drawing tools to draw a NAND symbol. When you are done, save and close the window.

Schematic Simulation

The best way to test design is to make a ”testbench” schematic, then instantiate your design within it. This allows different tests for the same design, and won’t cause LVS conflicts. Go to File — New — Cell, and name the cell nand2_tb, and set the view to ”Schematic.”

Press i, and place your cell (NAND2X1b_RVT). Using the “vdc” and “vpwl” components in analogLib and the “vdd”, “gnd”, “noConn” components in basic, hook up VDD, VSS, and the inputs as shown in Figure 3. Add a 1fF capacitor as a load. Set the vpwl source on your input as follows

- A: 3 pairs of points, T1: 0, V1: 0, T2: 100p, V2: 0, T3: 110p, V3: VDD_val

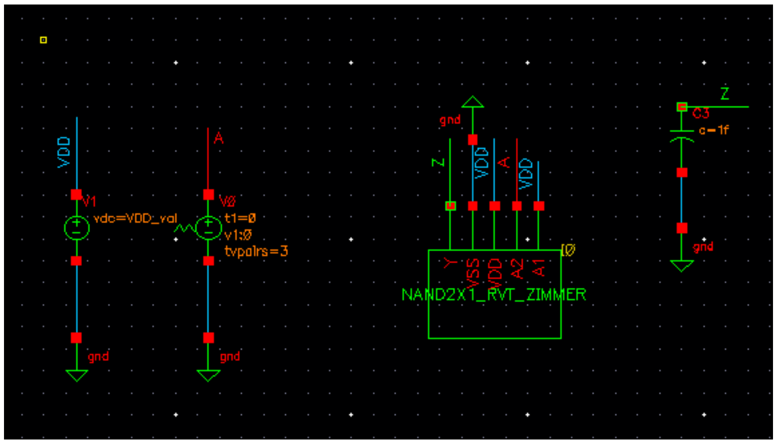


Figure 3: Testbench to measure the delay of an inverter.

Note that ”VDD_val” is not a typo–we are defining it as a parameter so it can be changed from the testbench. Go to File — Check and Save and fix any warnings or errors.

Now to simulate, go to Launch — ADE L.

Click on Variables — Copy from Cellview, then in the right pane, enter 1.05 as the value for VDD_val. Go to Analysis — Choose, and set the stop time to 4n, and click ok. Then go to Outputs — To be Plotted — Select on Schematic, and click on both the A and Z bus (and select all of their signals), then press “Esc” when finished. Last, go to Simulation — Netlist and Run, and make sure the design simulated as expected.

The default process corner for simulation is TT. You can change the corner by going to Setup — Model Libraries, and changing the section to ”SF”, ”FS”, ”SS”, or ”FF”. You can change the temperature by going to Setup — Temperature.

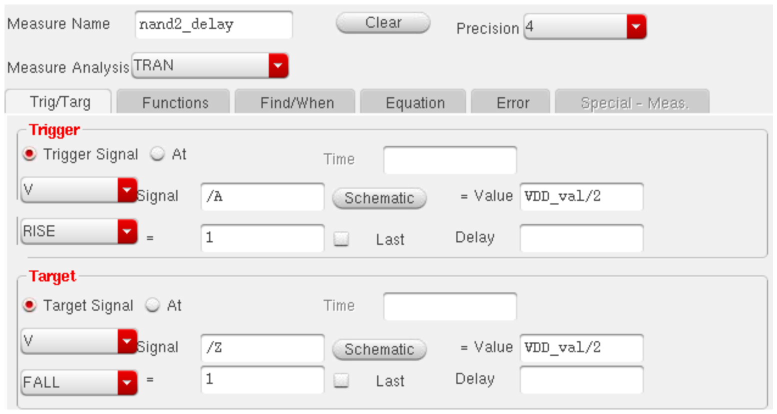
Monte Carlo Analysis

While a real design kit will use measured silicon data to calibrate variability for many device parameters, our educational design kit will only vary the threshold voltage of a device. Threshold voltage variation has been found to be Gaussian, so during Monte Carlo simulation, your circuit will be simulated hundreds of times, but each time the threshold of each device will be shifted by an amount determined from sampling a Gaussian distribution for each device. The standard deviation of each device is inversely proportional to its size. In our process, Avt is assumed to be 2 mV·um.

$$\sigma_{Vth} = \frac{2}{\sqrt{W * L * 1e6 * 1e6}} mV \tag{1}$$

Cadence Virtuoso cannot perform Monte Carlo simulation in ADE L. Virtuoso will run Monte Carlo in ADE XL, but only with the Spectre simulator, not HSPICE. Therefore to run Monte Carlo, there are two options: 1) Export the netlist and run HSPICE manually from the command line or 2) use HSPICE built in utility in ADE-L to run Monte Carlo. The downside of the 2nd approach is that it is no longer possible to run sweeps of a variable (such as the supply voltage), so you will need to run on the command line to both sweep a variable and run Monte Carlo analysis.

To run Monte Carlo, in your open ADE L window, go to Tools — HSPICE Statistical. Under "Number of runs" enter 300, and check the box next to "Enabled." Under the outputs tab in the Measurements section, click on "Open." Enter the settings shown in Figure 4, and click "Add", then "Close." Now inside the HSPICE Monte Carlo Analysis window, under the "Simulation" section click "Netlist-Run." You should see different delays for different iterations as shown in Figure 5.


Measure Name: nand2_delay
Precision: 4
Measure Analysis: TRAN

Trigger
Trigger Signal [✓]
V Signal: /A = Value: VDD_val/2
RISE = 1

Target
Target Signal [✓]
V Signal: /Z = Value: VDD_val/2
FALL = 1

Figure 4: Settings to measure the delay through an inverter.

For any measurement, statistics are automatically calculated in the Measurement Variation Statistics section as shown in Figure 6. Here you can get a quick mean and sigma measurement. To view the distribution, click "Get Variation Statistics", then click "Plotting as." To export this data for more detailed processing, change the mode from "Hist" to "Scatter" and set Y as your variable and X as index as shown in Figure Then in the plotting window, right click on the signal name — Send To — Export as shown in Figure 7. You can also find the .mt file in /simulation/inverter_tb/HSPICE/schematic/psf/input.mt0.

In the HSPICE window, go to File — Save Setup so it can be easily loaded later. If you do this from the ADE L window, you will lose HSPICE-specific information.

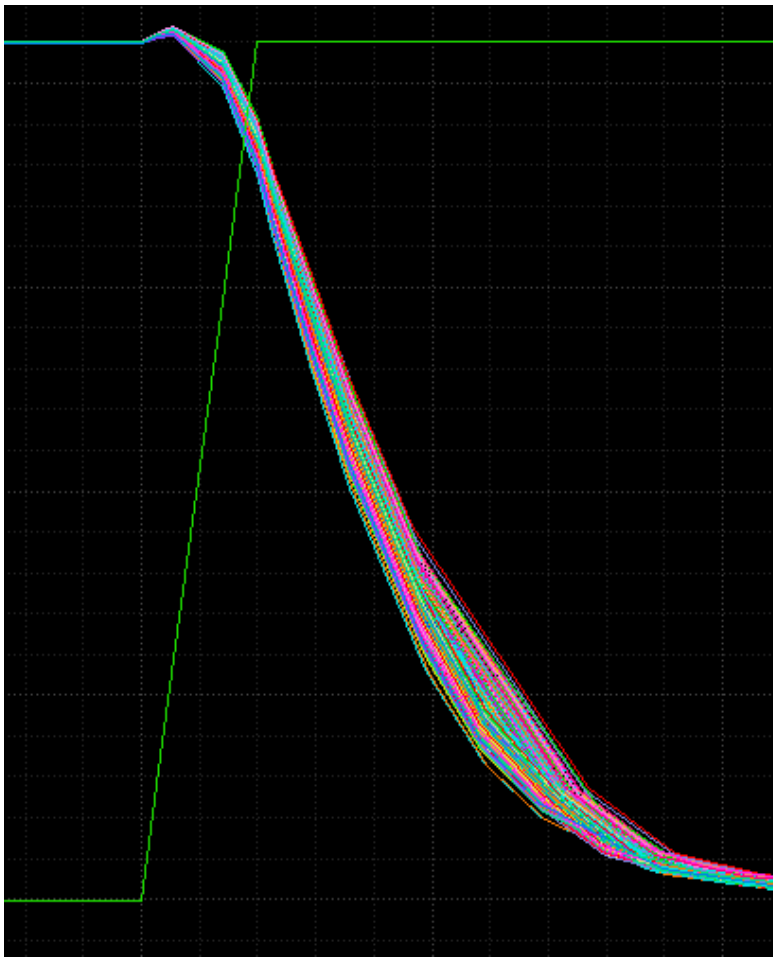
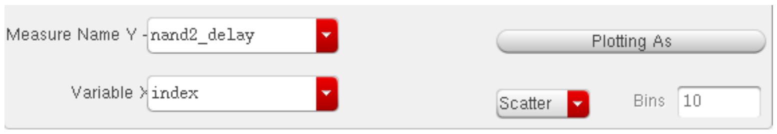


Figure 5: Different inverter delays due to local variation.



Measure Name Y nand2_delay
Variable X index
Plotting As
Scatter
Bins 10

Figure 6: Generate a scatterplot of a measured value.

Layout Entry

In this tutorial, we will design a new NAND2 gate. To save time, we will use an existing cell as a template. In the CIW, go to Tools — Library Manager, choose the saed32nm_rvt library , and choose the cell as NAND2X1_RVT, then right click on "Layout" and choose "Copy". Change the "To" library to "customcells" and cell to NAND2X1B_RVT and click ok. Note that you are only copying the layout view–do not override the schematic you just created!

Now open the layout view NAND2X1B_RVT in the customcells library, and you should see the layout in Figure 8. Based on the naming of the cell, the "X1" means that the drive strength should

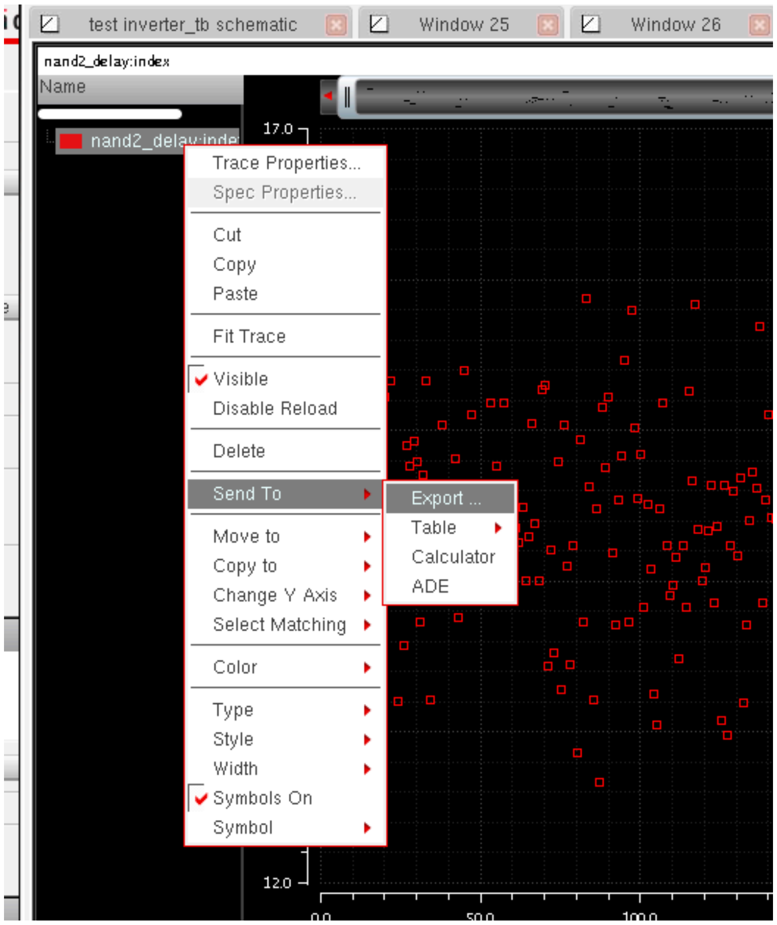


Figure 7: Export variable to a CSV file.

be larger than NAND2X0_RVT. However, the designers accomplished this by adding 2 extra inverter stages as a buffer. The fanout between the 1st/2nd stage and the 2nd/3rd stage is less than 4, and therefore is not optimal.

We can improve this gate both in terms of performance and area by reverting it to a single stage while retaining drive strength by increasing the NMOS and PMOS size. To accomplish this, each PMOS will need to be 760nm wide and the NMOS will need to be about 630nm wide (1.5 increase in size because of the stack). However, these devices will not fit on the standard cell pitch, so we need to multi-finger both the NMOS and PMOS as shown in Figure 9. Following the guidelines below, modify your layout until it looks like the figure.

You only need a few commands to change the layout. By pressing ∼, 1, 2, 3 etc you can show only certain layers. Press Shift-1 to add M1 to whatever is visible. To move something, hover over the object, then press ”m” on the keyboard. To make something bigger or smaller, hover over the edge of the object, then press ”s” on the keyboard, then click again at the final location. Press ”Esc” if you selected something incorrectly. To add new wire, press Ctrl-Shift-W (then F3 will change the options such as the width). To just add a rectangle, select the desired layer in the layer pallete, and press ”r”. Then click on the two corners to create.

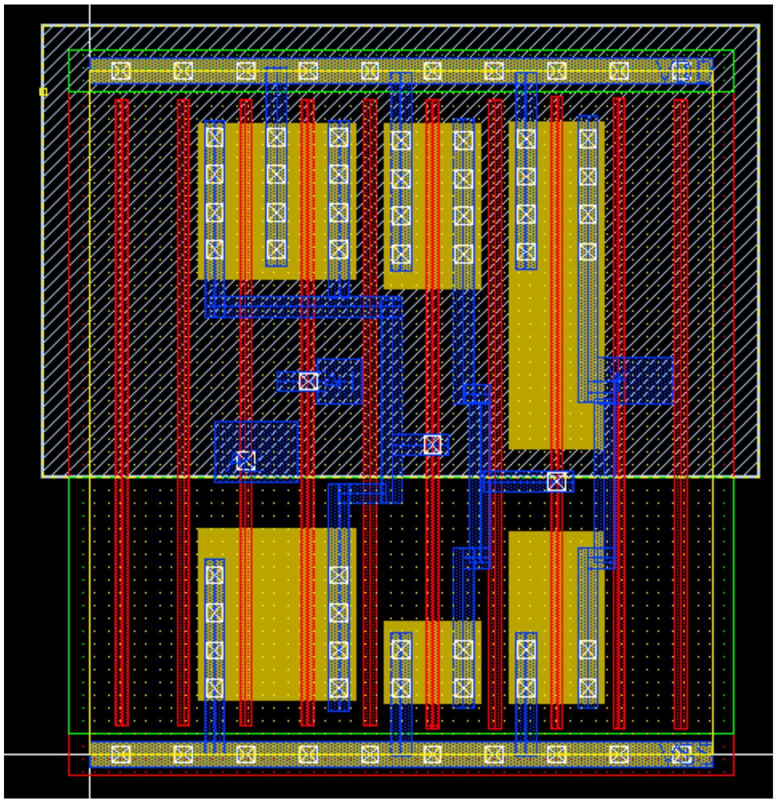


Figure 8: Educational standard cell library NAND2X1_RVT

Make sure you resize all of the wells as well as the yellow boundary to the new cell dimensions (try to emulate the left end of the cell.) Make all of the layers visible to ensure that you didn't miss anything. All of the pin labels should already exist, but if deleted one, go to Create — Label, enter the terminal name, then select the layer to be M1PIN.

LVS

When you believe the layout matches the schematic, go to Hercules — Run LVS. Wait a few seconds for the "Block" entry to be filled in automatically as shown in Figure 10, then press "Execute." In the options tab, choose both "General" and "LVS" from the menu, and make sure that the Block and Library entries are correct. If you are having problems running, close all of your designs in the CIW by going to File — Close Data — All — Ok, and make sure there are no abstract views in your design.

If you are having problems passing LVS, you will see an error like Figure 11. Click on the "LVS Errors" tab, then the design under "First Priority." The best approach for small circuits is to look directly at the netlists as shown in Figure 12. First check to make sure that all of the ports are defined and match, then look at the rest of the netlist and see if either the number of devices, the properties, or the connections are wrong.

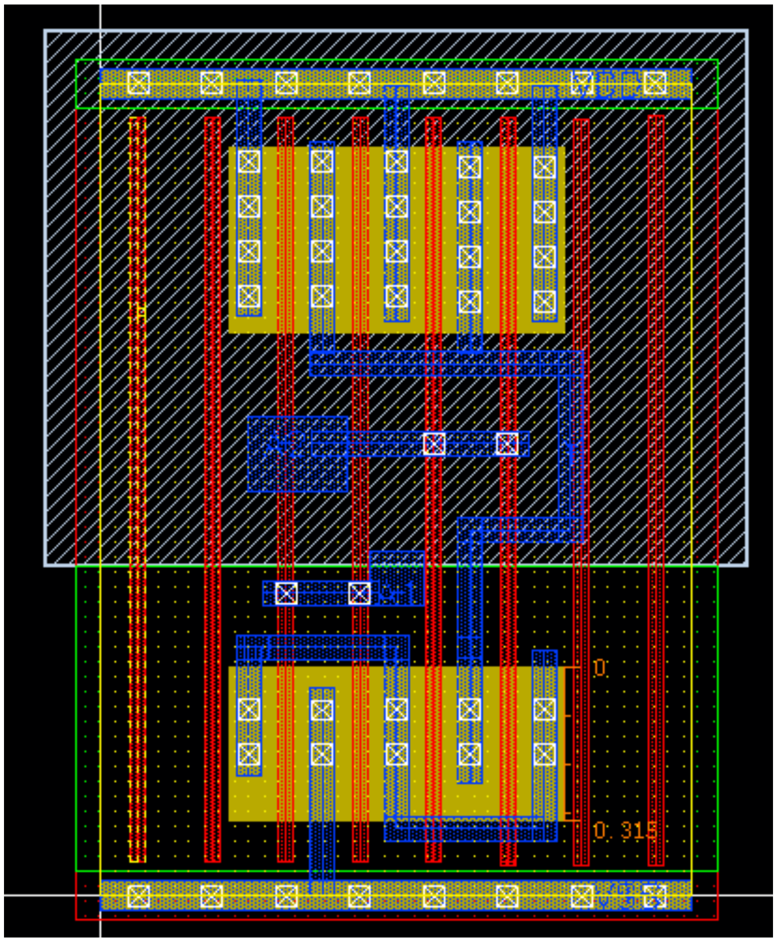


Figure 9: An improved 2-input NAND layout

DRC

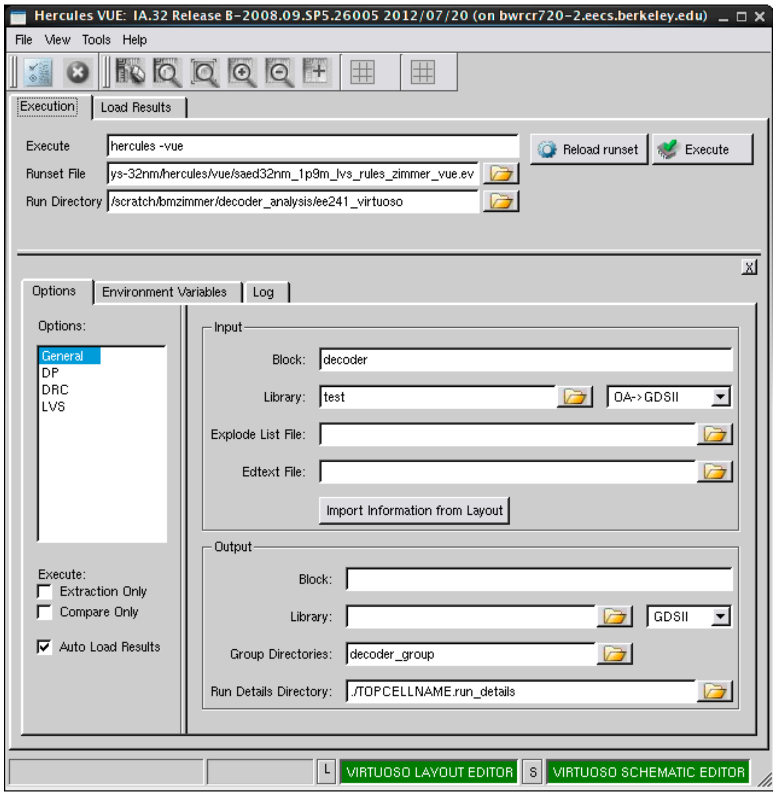
When you believe the layout matches the schematic, go to Hercules — Run DRC. Wait a few seconds for the "Block" entry to be filled in automatically then press "Execute."

Double click on the DRC error to see the error on the layout. Each error entry will provide additional details about what failed, as shown in Figure 13.

Extraction

To run parasitic extraction in Virtuoso, open the layout view and go to StarRC — Parasitic Generation Cockpit. The setting should be automatically loaded for you as shown in Figure 14, but some options are useful to understand.

- Extract Parasitics Tab — Extraction. RC will extract both resistance and capacitance.
- Extract Parasitics Tab — Couple to Ground = NO will include coupling between nets in your extraction


Hercules VUE: IA.32 Release B-2008.09.SP5.26005 2012/07/20 (on bwrcr720-2.eecs.berkeley.edu)

Execute: hercules -vue
Runset File: ys-32nm/hercules/vue/saed32nm_1p9m_lvs_rules_zimmer_vue.ev
Run Directory: /scratch/bmzimmer/decoder_analysis/ee241_virtuoso

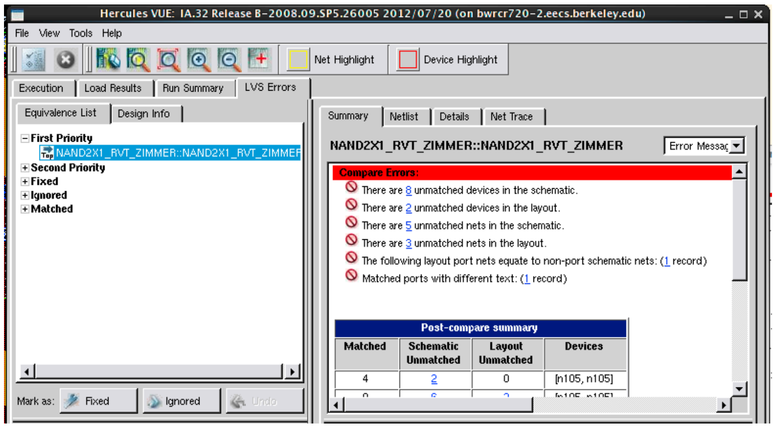
Options: General

Input
Block: decoder
Library: test — OA->GDSII
Explode List File:
Edtext File:

Output
Block:
Library: — GDSII
Group Directories: decoder_group
Run Details Directory: ./TOPCELLNAME.run_details

Execute:
☐ Extraction Only
☐ Compare Only
☑ Auto Load Results

Figure 10: Hercules LVS dialog settings.


Hercules VUE: IA.32 Release B-2008.09.SP5.26005 2012/07/20 (on bwrcr720-2.eecs.berkeley.edu)

First Priority
NAND2X1_RVT_ZIMMER::NAND2X1_RVT_ZIMMER
Second Priority
Fixed
Ignored
Matched

NAND2X1_RVT_ZIMMER::NAND2X1_RVT_ZIMMER

Compare Errors:
There are 8 unmatched devices in the schematic.
There are 2 unmatched devices in the layout.
There are 5 unmatched nets in the schematic.
There are 3 unmatched nets in the layout.
The following layout port nets equate to non-port schematic nets: (1 record)
Matched ports with different text: (1 record)

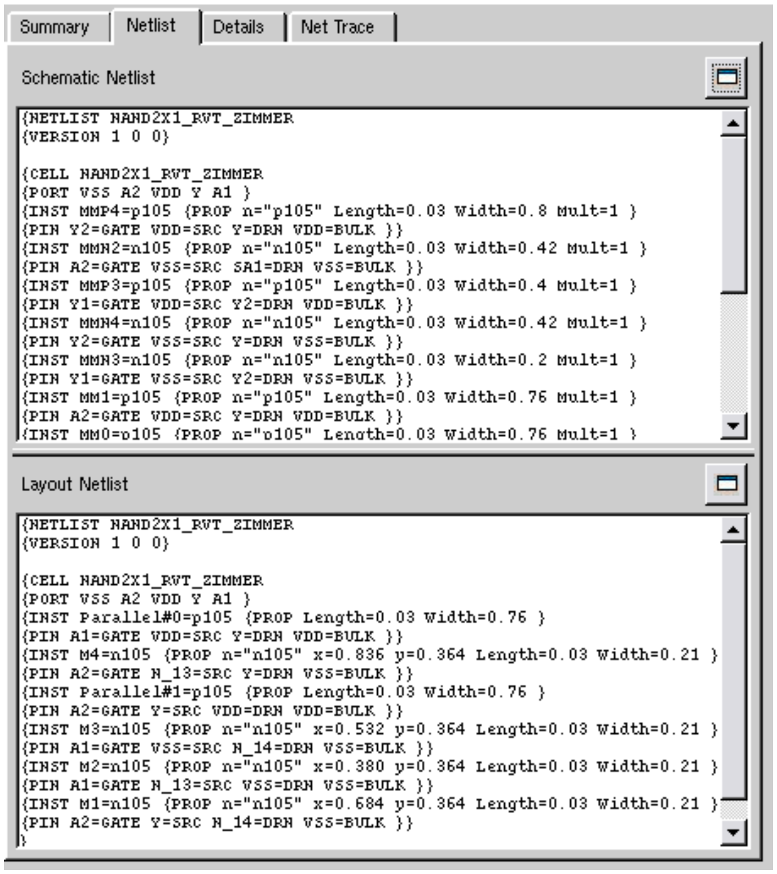
Post-compare summary

Matched	Schematic Unmatched	Layout Unmatched	Devices
4	2	0	[n105, n105]

Figure 11: Typical error dialog for LVS.

- Extract Parasitics Tab — Additional Options, allows you to set a threshold to try to reduce device count

Now click "Apply" and wait a minute while extraction runs. When it finishes, go to your new library and open the "starrc" view of the decoder cell. If you zoom in, you can see the annotated parasitics overlaid on your layout. By going to StarRC — Parasitic Prober you can explore the



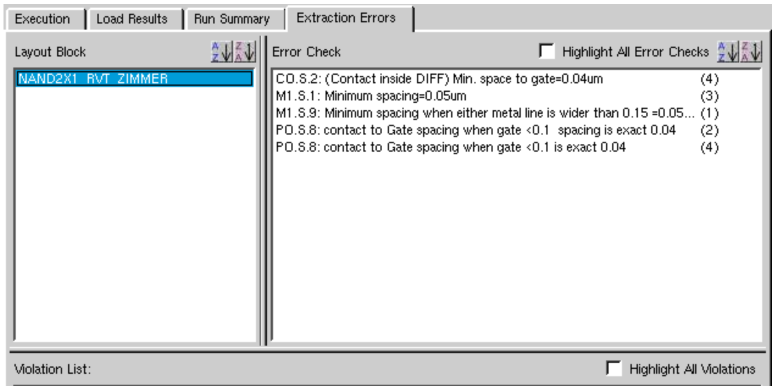
```
{NETLIST NAND2X1_RVT_ZIMMER
{VERSION 1 0 0}

{CELL NAND2X1_RVT_ZIMMER
{PORT VSS A2 VDD Y A1 }
{INST MMP4=p105 {PROP n="p105" Length=0.03 Width=0.8 Mult=1 }
{PIN Y2=GATE VDD=SRC Y=DRN VDD=BULK }}
{INST MMN2=n105 {PROP n="n105" Length=0.03 Width=0.42 Mult=1 }
{PIN A2=GATE VSS=SRC SA1=DRN VSS=BULK }}
{INST MMP3=p105 {PROP n="p105" Length=0.03 Width=0.4 Mult=1 }
{PIN Y1=GATE VDD=SRC Y2=DRN VDD=BULK }}
{INST MMN4=n105 {PROP n="n105" Length=0.03 Width=0.42 Mult=1 }
{PIN Y2=GATE VSS=SRC Y=DRN VSS=BULK }}
{INST MMN3=n105 {PROP n="n105" Length=0.03 Width=0.2 Mult=1 }
{PIN Y1=GATE VSS=SRC Y2=DRN VSS=BULK }}
{INST MM1=p105 {PROP n="p105" Length=0.03 Width=0.76 Mult=1 }
{PIN A2=GATE VDD=SRC Y=DRN VDD=BULK }}
{INST MM0=p105 {PROP n="p105" Length=0.03 Width=0.76 Mult=1 }
```

```
{NETLIST NAND2X1_RVT_ZIMMER
{VERSION 1 0 0}

{CELL NAND2X1_RVT_ZIMMER
{PORT VSS A2 VDD Y A1 }
{INST Parallel#0=p105 {PROP Length=0.03 Width=0.76 }
{PIN A1=GATE VDD=SRC Y=DRN VDD=BULK }}
{INST M4=n105 {PROP n="n105" x=0.836 y=0.364 Length=0.03 Width=0.21 }
{PIN A2=GATE N_13=SRC Y=DRN VSS=BULK }}
{INST Parallel#1=p105 {PROP Length=0.03 Width=0.76 }
{PIN A2=GATE Y=SRC VDD=DRN VDD=BULK }}
{INST M3=n105 {PROP n="n105" x=0.532 y=0.364 Length=0.03 Width=0.21 }
{PIN A1=GATE VSS=SRC N_14=DRN VSS=BULK }}
{INST M2=n105 {PROP n="n105" x=0.380 y=0.364 Length=0.03 Width=0.21 }
{PIN A1=GATE N_13=SRC VSS=DRN VSS=BULK }}
{INST M1=n105 {PROP n="n105" x=0.684 y=0.364 Length=0.03 Width=0.21 }
{PIN A2=GATE Y=SRC N_14=DRN VSS=BULK }}
}
```

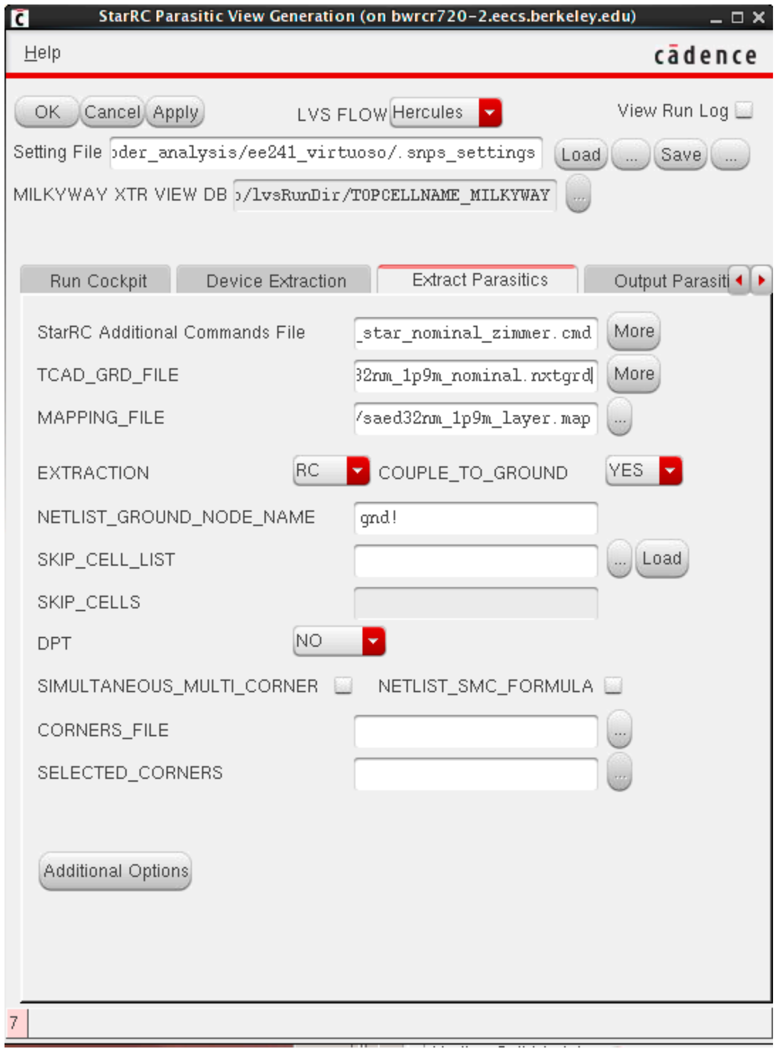
Figure 12: Debugging LVS using the netlist



```
CO.S.2: (Contact inside DIFF) Min. space to gate=0.04um                  (4)
M1.S.1: Minimum spacing=0.05um                                           (3)
M1.S.9: Minimum spacing when either metal line is wider than 0.15 =0.05... (1)
PO.S.8: contact to Gate spacing when gate <0.1  spacing is exact 0.04    (2)
PO.S.8: contact to Gate spacing when gate <0.1 is exact 0.04             (4)
```

Figure 13: Typical DRC errors.

parasitics in your design.



LVS FLOW Hercules
View Run Log
Setting File ›der_analysis/ee241_virtuoso/.snps_settings
MILKYWAY XTR VIEW DB ›/lvsRunDir/TOPCELLNAME_MILKYWAY

StarRC Additional Commands File	_star_nominal_zimmer.cmd
TCAD_GRD_FILE	32nm_1p9m_nominal.nxtgrd
MAPPING_FILE	/saed32nm_1p9m_layer.map
EXTRACTION	RC
COUPLE_TO_GROUND	YES
NETLIST_GROUND_NODE_NAME	gnd!
SKIP_CELL_LIST	
SKIP_CELLS	
DPT	NO
SIMULTANEOUS_MULTI_CORNER	☐
NETLIST_SMC_FORMULA	☐
CORNERS_FILE	
SELECTED_CORNERS	

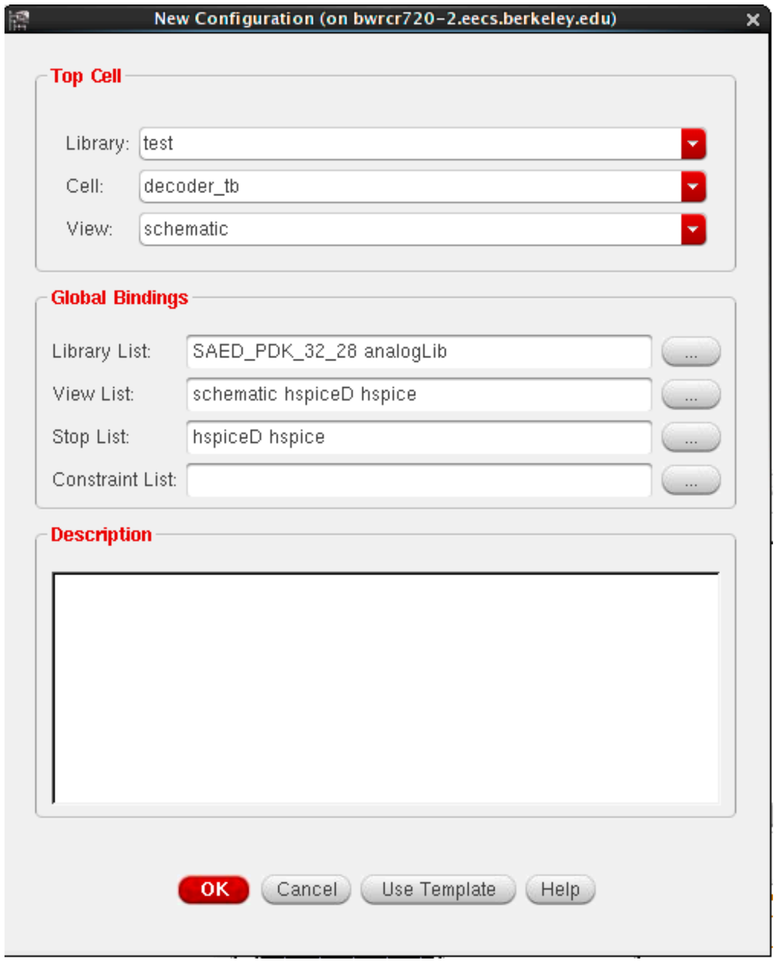
Figure 14: StarRC settings.

Extracted Simulation

Next, we need to make a "config" view that will let us tell the simulator whether to simulate with the schematic view or the starrc view. Go to File — New — Cellview, and enter

- Cell: nand2_tb (the testbench you just created)
- View: config

Then click enter the settings shown in Figure 15. When the next window opens, go to File — Save. Now in the schematic window, go to Launch — ADE L. Go to Setup — Simulator, and make sure HSPICE is chosen. Click on Setup — Design, and choose View: config, and press ok. Enter the same settings as your schematic level, and run the simulation.



Library: test
Cell: decoder_tb
View: schematic

Library List: SAED_PDK_32_28 analogLib
View List: schematic hspiceD hspice
Stop List: hspiceD hspice
Constraint List:

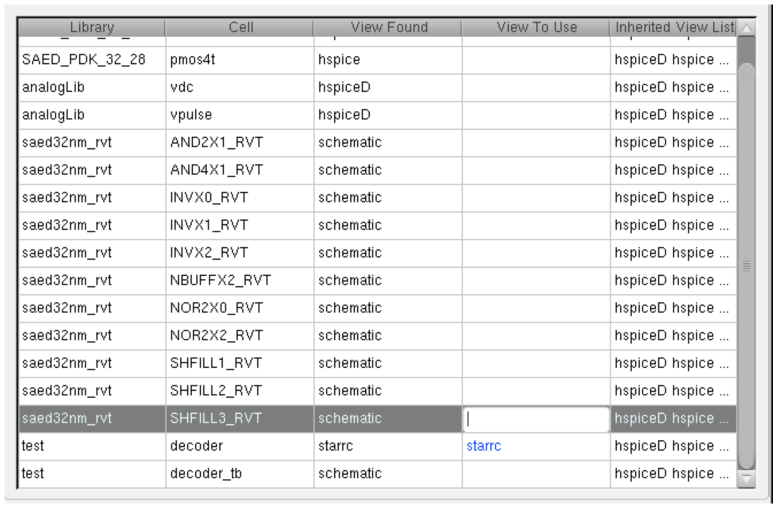
Figure 15: Settings for the new config view.

Next, you need to tell the netlister to choose the extracted view instead of the schematic view. Open the config view of nand2_tb, (click "yes" for Configuration config), then in the "View to Use" column for the "NAND2X1B_RVT" instance, enter starrc as shown in Figure 16. Then go to File — Save.

Cell Characterization

.lef/Milkyway generation

Open the layout view of your new cell, and go to Tools — Abstract Generation. Click on your cell, then "Move Cell" to the "Core" bin. Click on the Options tab, then click "..." next to Import Options, navigate to gen_stdcells/abstract.options, then click "Import". Now click "Start." In the CIW, go to File — Export — LEF, and create the file gen_stdcells/lefs/nand2x1b_rvt.lef. Set the library name to "customcells", output cell to "nand2x1b_rvt", and view to "abstract." Then check the box next to "no technology", and click "ok."



Library	Cell	View Found	View To Use	Inherited View List
SAED_PDK_32_28	pmos4t	hspice		hspiceD hspice ...
analogLib	vdc	hspiceD		hspiceD hspice ...
analogLib	vpulse	hspiceD		hspiceD hspice ...
saed32nm_rvt	AND2X1_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	AND4X1_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	INVX0_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	INVX1_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	INVX2_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	NBUFFX2_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	NOR2X0_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	NOR2X2_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	SHFILL1_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	SHFILL2_RVT	schematic		hspiceD hspice ...
saed32nm_rvt	SHFILL3_RVT	schematic		hspiceD hspice ...
test	decoder	starrc	starrc	hspiceD hspice ...
test	decoder_tb	schematic		hspiceD hspice ...

Figure 16: Change the config view to use the extracted netlist.

Now you need to convert the LEF file to a MW file. There is a Makefile that performs this process. First look at the generated LEF file to make sure all of the pins exist and have the correct direction.

```
% cd /scratch/userA/gen_stdcell/lefs/
% vim nand2x1b_rvt.lef
% make
```

.lib/.db generation

You will need a netlist of devices to characterize. Open the schematic of your design, go to Launch — ADE L, then Setup — Environment, and check the box next to "setTopLevelAsSubckt" and "HSPICE Case Sensitivity". Now, go to Simulation — Netlist — Create, then File — Save As, and place it in the gen_stdcells/netlists directory as nand2x1b_rvt.sp. Then, open this file in an editor, and delete everything before the .subckt line (but maintain the first line as a comment!)

Library NCX automatically recognizes the functions of different cells, and based on the existing technology settings, creates a set of HSPICE simulations that will generate a .lib file for you. After that, the Makefile calls IC Compiler to convert the .lib file to a .db file. Generate timing information for your cell, then open the .lib file to see what happened.

```
% cd /scratch/userA/gen_stdcell/
% make
% vim customcells.lib
...
pin (Y) {
      direction : "output";
      function : "!(A1 * A2)";
      related_power_pin : "VDD";
      related_ground_pin : "VSS";
      timing () {
        related_pin : "A1";
        timing_type : "combinational";
        timing_sense : "negative_unate";
        cell_rise ("del_1_7_7") {
          index_1("0.0160000, 0.0320000, 0.0640000, 0.1280000, 0.256000 ...
          index_2("0.1000000, 0.2500000, 0.5000000, 1.0000000, 2.000000,...
          values("0.0158369, 0.0167964, 0.0182212, 0.0211338, 0.0268709,...
            "0.0229815, 0.0243130, 0.0263408, 0.0299642, 0.0359516, 0.04...
            "0.0341857, 0.0360079, 0.0388393, 0.0438080, 0.0522559, 0.06...
            "0.0526853, 0.0552465, 0.0589251, 0.0657167, 0.0774758, 0.09...
            "0.0862680, 0.0890297, 0.0939580, 0.1024084, 0.1180048, 0.14...
            "0.1502805, 0.1536840, 0.1593136, 0.1690508, 0.1877911, 0.22...
            "0.2778792, 0.2797484, 0.2856589, 0.2970177, 0.3184407, 0.35...
        }

...
```

The lib file holds a 2d table indexed by index_1 (rise/fall time on the input) and index_2 (load capacitance). It is important to have a feeling for how this was generated. Open the spice file that was used to generate this data

```
% cd sim_dir/lib/customcells/NAND2X1B_RVT/tY_A1_0000f
% vim tY_A1_0000f.sp
%
.DATA stimdata
index nc_td adrv_cap iptr cap_Y
+ t_A10 v_A10 t_A11 v_A11 t_A12 v_A12 t_A13 v_A13 t_A14 v_A14 t_A15 v_A15 t_A16 v_A16 t_A17
+ t_A110 v_A110 t_A111 v_A111 t_A112 v_A112 t_A113 v_A113 t_A114 v_A114 t_A115 v_A115
+ t_A20 v_A20 t_A21 v_A21
  1 4.01e-09 5e-12 1.6e-11 1.0000000000e-16
  + 0 0 4e-09 0 4.01e-09 0 4.012666667e-09 0.05 4.015333333e-09 0.2
  + 4.018e-09 0.3325197896 4.020666667e-09 0.44801579 4.023333333e-09 0.55 4.026e-09 0.641259
  + 4.031333333e-09 0.8 4.034e-09 0.8706299474 4.036666667e-09 0.9370039475 4.039333333e-09
  + 4.044666667e-09 1 0 1 4e-07 1
    2 4.01e-09 5e-12 1.6e-11 2.5000000000e-16
```

This structure will run a simulation for each line in the .DATA section and effectively sweeps both dimensions of the table. Notice the end of each line 1.000000000e-16 and 2.5000000000e-16. for the parameter cap_Y. This matches the values in index_1.

Now use a waveform viewer to see what happened when the simulation ran.

```
% cd sim_dir/lib/customcells/NAND2X1B_RVT/tY_A1_0000f
% wv tY_A1_0000f.tr0 &
```

Then plot v(n_a1) and v(n_y), and you should see something similar to Figure 17.

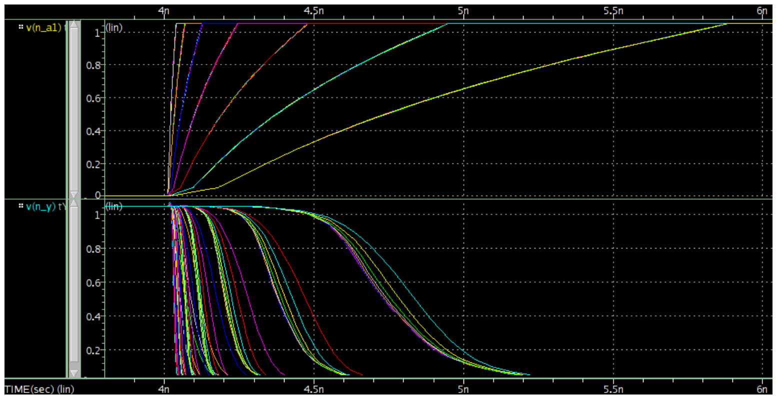


Figure 17: NCX characterization of a NAND2 gate

Place-and-route

First, optimize the timing for the existing 4-to-16 decoder and note the critical path.

```
% cd /scratch/userA/decoder_analysis/build-rvt/
% vim Makefrag
% ....
clock_period = 0.08
...
# add
```

Then run place-and-route and record the critical path length and the gates that it goes through.

Now you need to let Design Compiler and IC Compiler know about your new library. Add the following lines to both dc-syn/Makefile and icc-par/Makefile (replace it with the correct path)

```
% cd /scratch/userA/decoder_analysis/build-rvt/dc-syn
% vim Makefile
% ....
toplevelinst =  ...

# Hack for custom cells
mw_sram_libs = /scratch/userA/gen_stdcells/lefs/customcells
db_sram_libs = /scratch/userA/gen_stdcells/customcells.db

% cd /scratch/userA/decoder_analysis/build-rvt/icc-par
% vim Makefile
% ....
toplevelinst =  ...

# Hack for custom cells
mw_sram_libs = /scratch/userA/gen_stdcells/lefs/customcells
db_sram_libs = /scratch/userA/gen_stdcells/customcells.db
```

Now rerun both synthesis and place-and-route, and check the Verilog output to make sure your gate was instantiated. If it was not, check the log file for errors (search for your cell name: NAND2X1B_RVT).