Introduction to the Custom Design Flow: Building a standard cell

EE241 Tutorial 3
Written by Brian Zimmer (2013)
Updated by Sean Huang (2019)

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
```

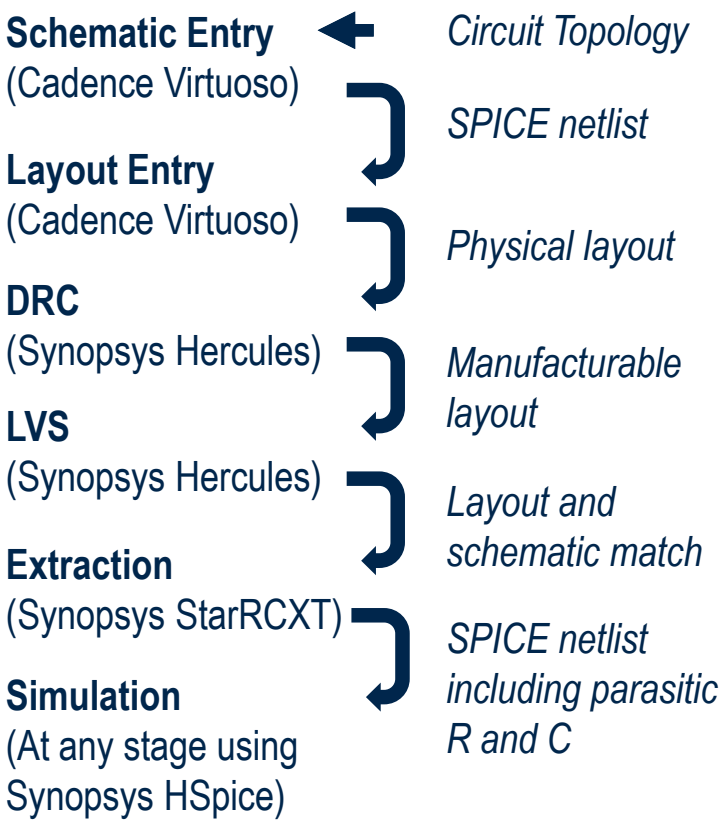


Figure 1: Custom design "flow."

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 "Y". 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.

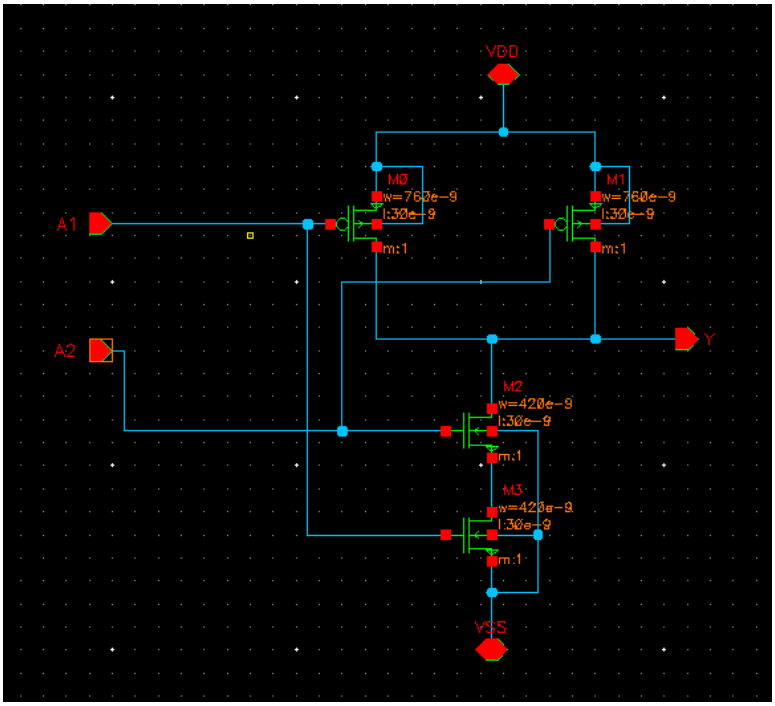


Figure 2: Schematic of the 2-input NAND gate.

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 Netlist Export

We will be running LVS against this schematic later, so we will need to export this schematic as a netlist. Most physical verification tools perform verification on generic filetypes, such as GDS files for layout and SPICE netlists for schematics. To export the NAND2 schematic you have just drawn, go to the Virtuoso main log window and go to File — Export — CDL...

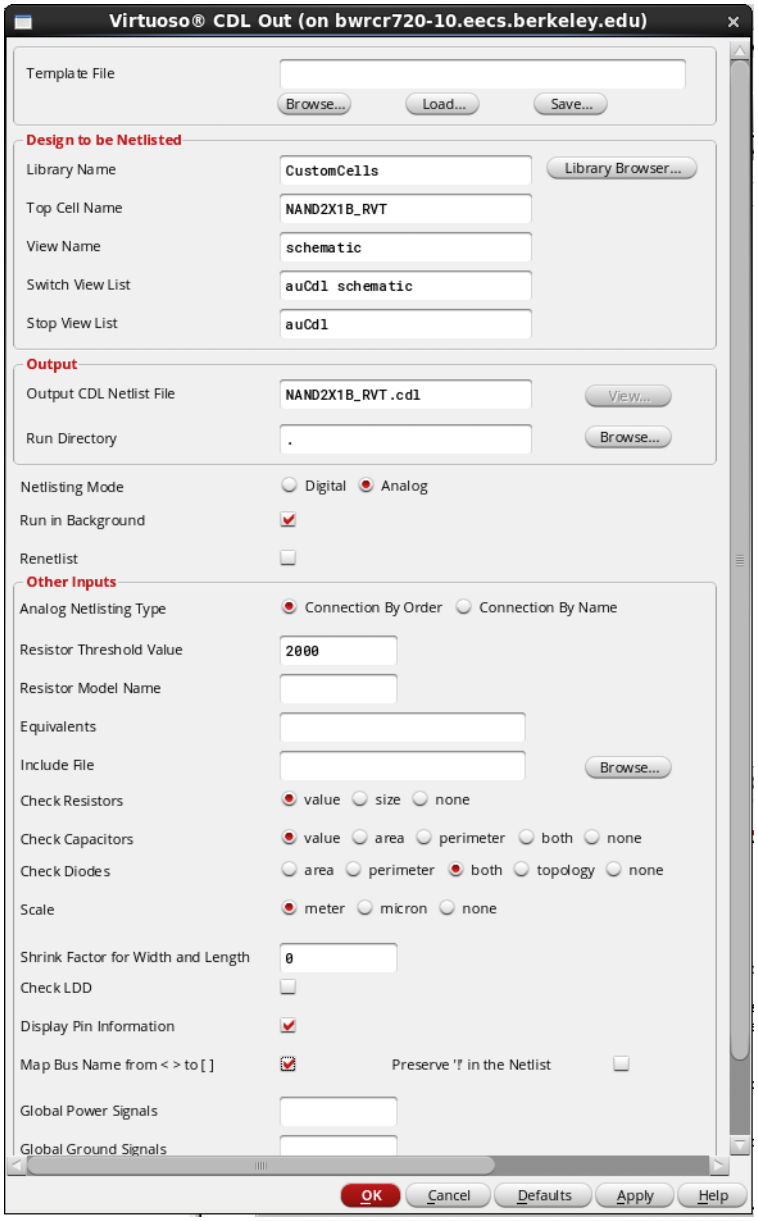


Figure 3: CDL Export Window

Populate the following fields:

- Library Name: Your new Custom Cell library
- Top Cell Name: NAND2X1B_RVT
- View Name: schematic
- Switch View List: auCdl schematic
- Stop View List: auCdl
- Output CDL Netlist File: NAND2X1B_RVT.cdl
- Make sure "Map Bus Name from <> to [ ]" is checked.

Hit Apply and wait for the CDL to be exported. Once this is done, open the CDL in a text editor. Notice that in the subcircuit definition, the transistor definitions (labeled `MMN*` or `MMP*`) have number of fingers definied. This is a limitation of the `SAED32_28_PDK` we are using and will cause this netlist to fail LVS as it expects this information. This technology does not support more than one finger, so we will just have to manually append `nf=1` to each transistor definition. Simple right?

Jokes aside, I have written a Python script to perform this hack so we can still do LVS with larger cells. The script should be in `gen_stdcells`. Fix your CDL file with this as follows.

```
From ee241_virtuoso,
  % python3 ../gen_stdcells/fixcdl.py NAN2X1B_RVT.cdl
```

Now the CDL is ready to be checked against the layout.

Schematic Simulation

The best way to test a 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 4. 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

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

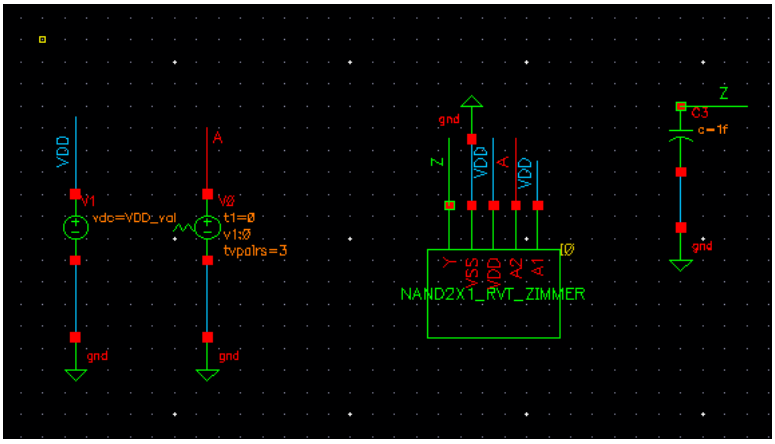


Figure 4: Testbench to measure the delay of an inverter.

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 5, 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 6.

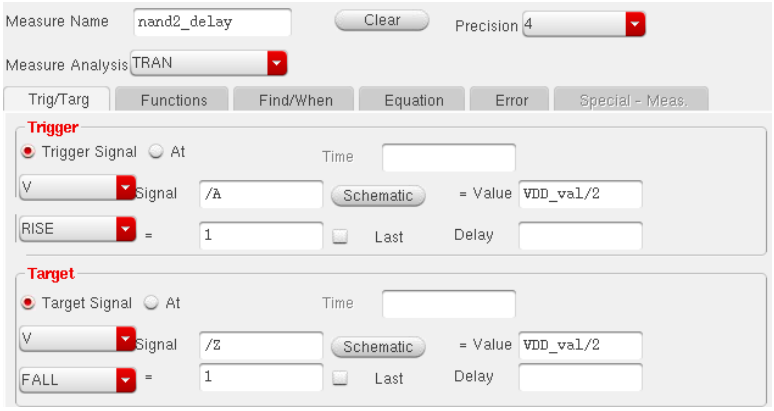


Figure 5: Settings to measure the delay through an inverter.

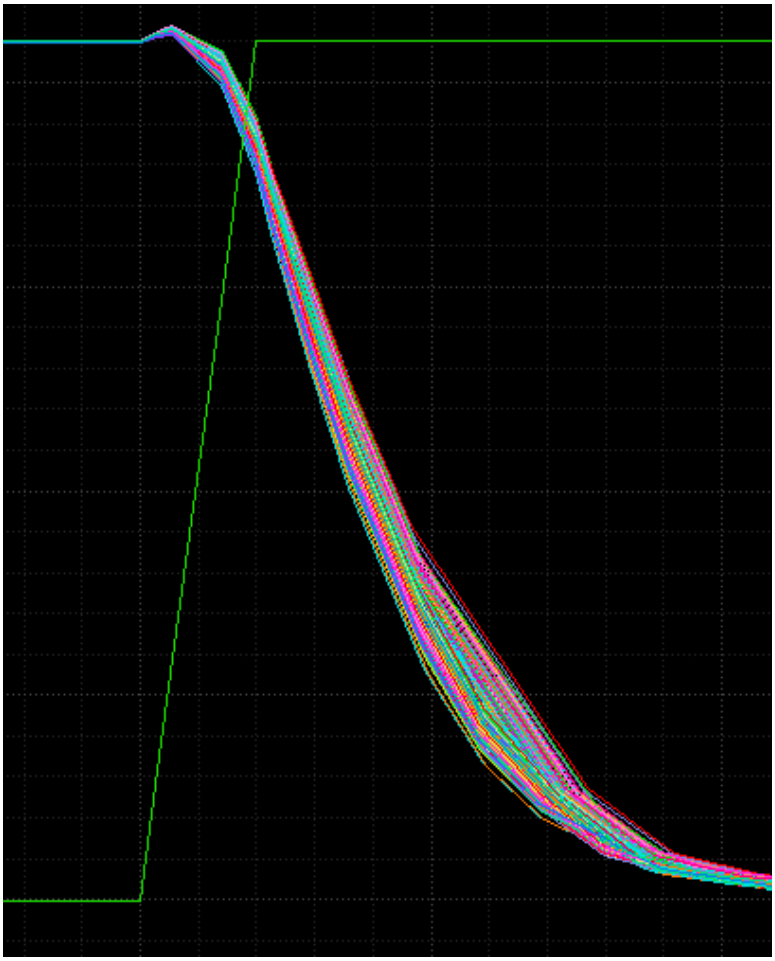


Figure 6: Different inverter delays due to local variation.

For any measurement, statistics are automatically calculated in the Measurement Variation Statistics section as shown in Figure 7. 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 8. You can also find the .mt file in /simulation/inverter_tb/HSPICE/schematic/psf/input.mt0.

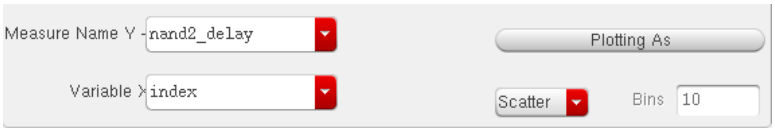


Figure 7: Generate a scatterplot of a measured value.

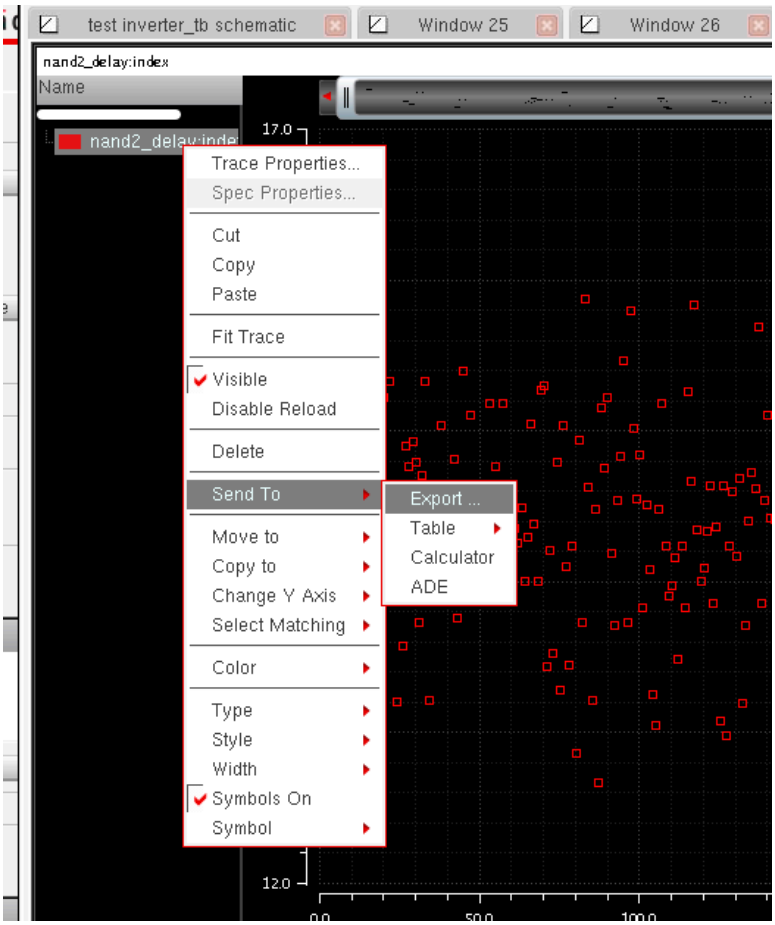


Figure 8: Export variable to a CSV file.

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.

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 9. Based on the naming of the cell, the "X1" means that the drive strength should 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.

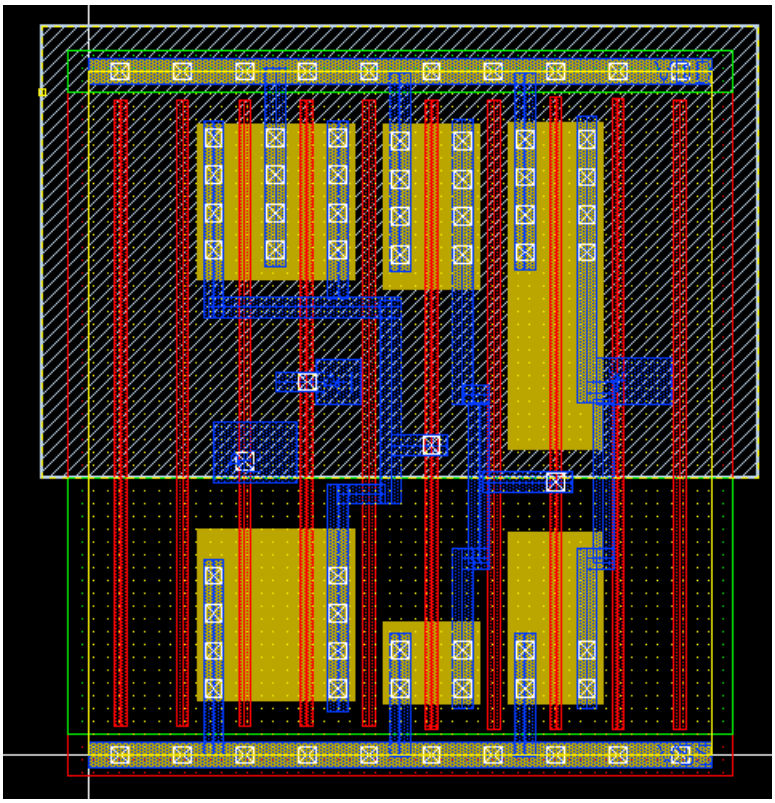


Figure 9: Educational standard cell library NAND2X1_RVT

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 10. 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

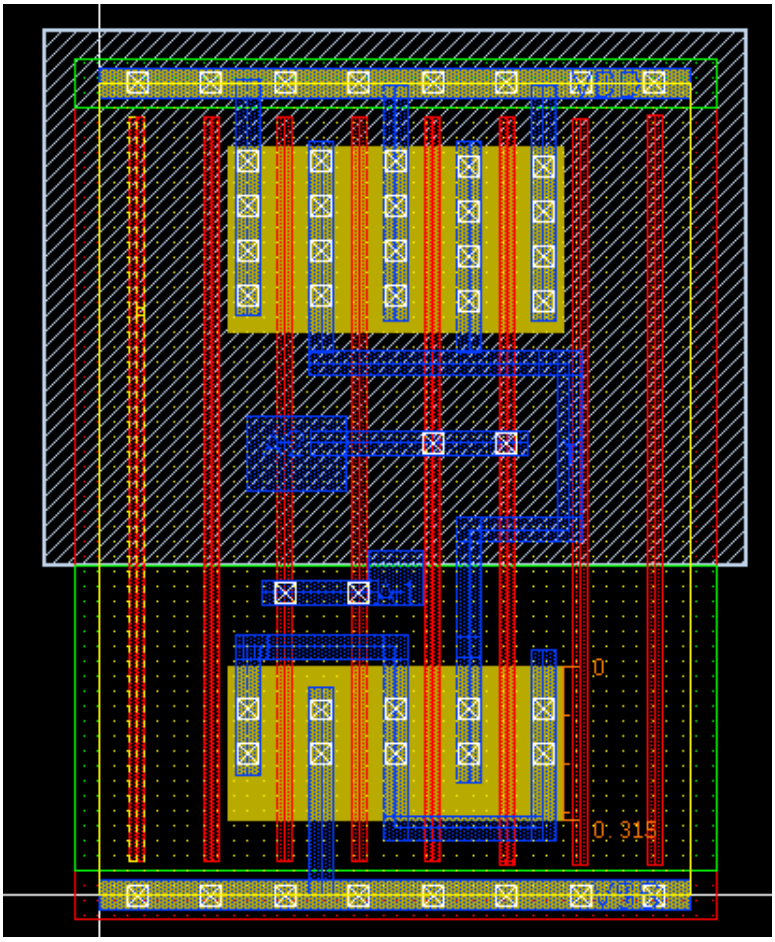


Figure 10: An improved 2-input NAND layout

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.

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, make sure both the layout and schematic views are open in Virtuoso. Go to IC Validator — Run VUE. Change the runset file to "/home/ff/ee241/hammer-stuff/SAED_PDK32nm/icv/drc/drc/saed32nm_1p9m_lvs_rules_sehuang.rs", then go to the Option tab. Go to the LVS settings and make sure the following settings are given:

- Source: Single Input
- Cell: NAND2X1B_RVT
- Filetype: SPICE
- File: NAND2X1B_RVT.cdl

Do NOT press "Import information from Schematic". This will tell IC Validator to extract its own CDL netlist, which as we have discussed before is missing the number of fingers per transistor. Once you have made sure all of these fields are filled in, hit Execute.

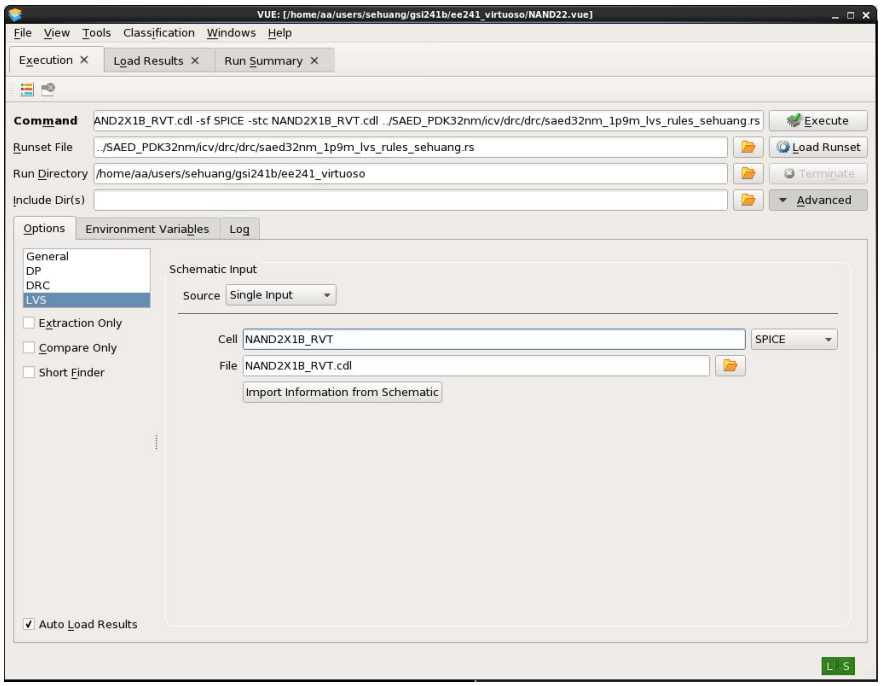


Figure 11: ICV LVS dialog settings.

If you are having problems passing LVS, you will see an error like Figure 12. Click on the "LVS Errors" tab. In the Equivalence List, you can see all the inequalities that LVS has found. Expanding this list and selecting an entry under Unmatched will populate the Summary tab to the right. This will display a table of run statistics. Selecting the Errors list and selecting one of the errors will update the Summary to show a list of the offending devices. Clicking one of these will bring up the device in the Netlist Visualizer in the bottom half of the GUI. This on its own is not particularly useful, however, when selecting the errors, ICV should highlight the offending devices on the layout or schematic, depending on which one you have selected.

DRC

When you believe the layout matches the schematic, go to the layout window and select IC Validator — Run VUE. This time the runset file should already have the correct file selected ("/home/ff/ee241/hammer-stuff/SAED_PDK32nm/icv/drc/drc/saed32nm_1p9m_drc_rules.rs").

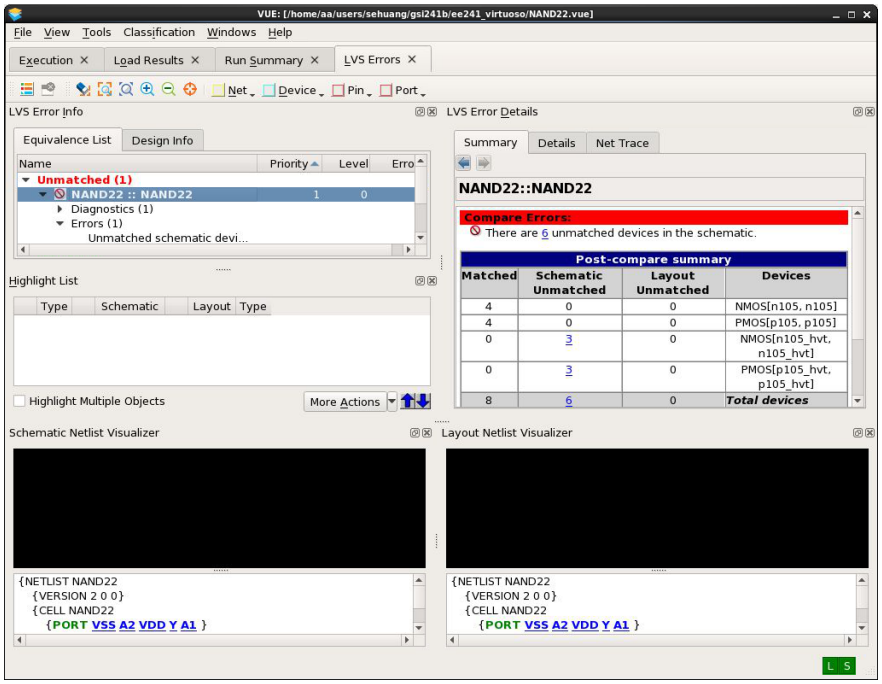


Figure 12: Typical error dialog for LVS.

Double click on the DRC error to highlight the error on the layout. Each error entry will provide additional details about what failed, as shown in Figure 14.

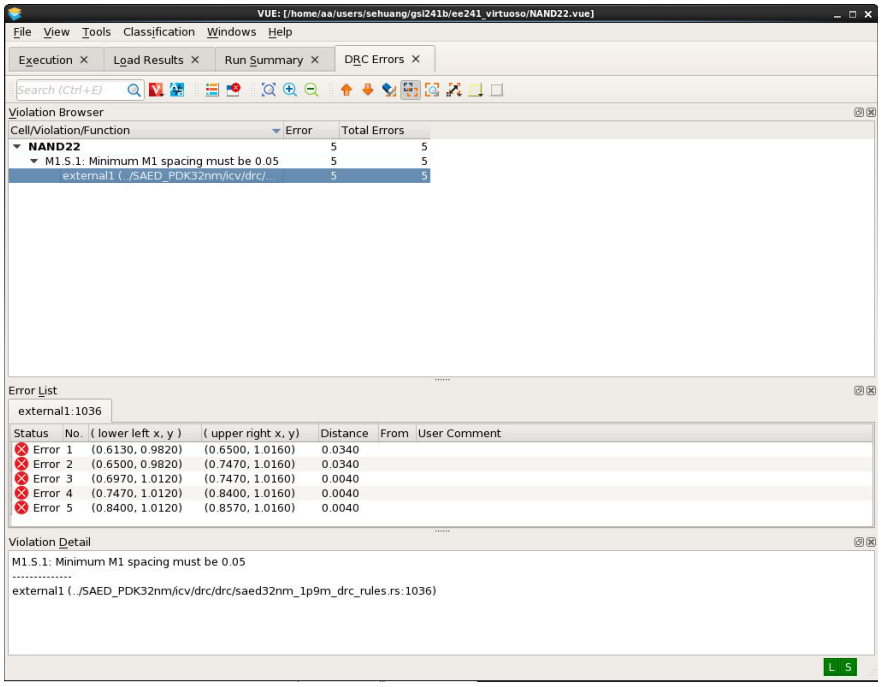


Figure 13: Typical DRC errors.

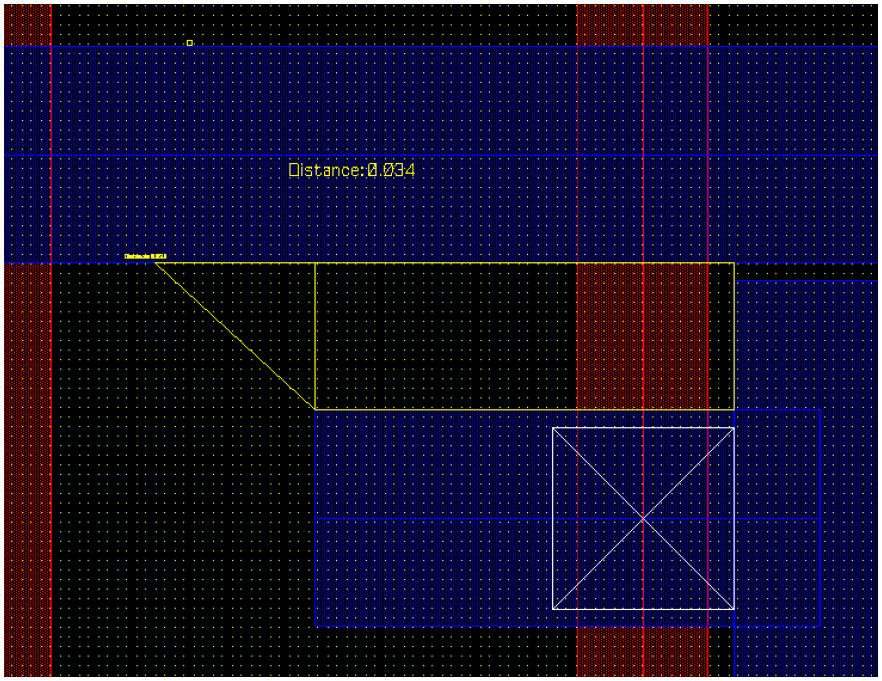


Figure 14: Highlighted DRC Error.

Extraction

To run parasitic extraction in Virtuoso, open the layout view and go to StarRC — Parasitic Generation Cockpit. The settings for this extraction are the same as in the last lab, so fill out the StarRC dialog in the same way.

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 NAND2 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 parasitics in your design.

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.

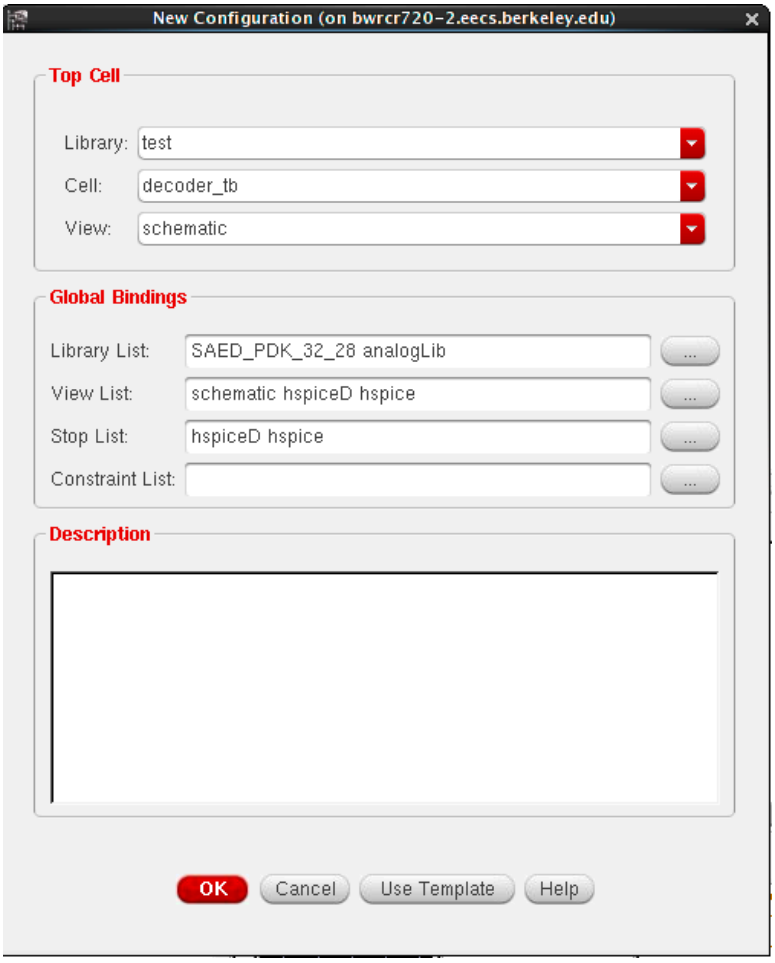


Figure 15: Settings for the new config view.

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.

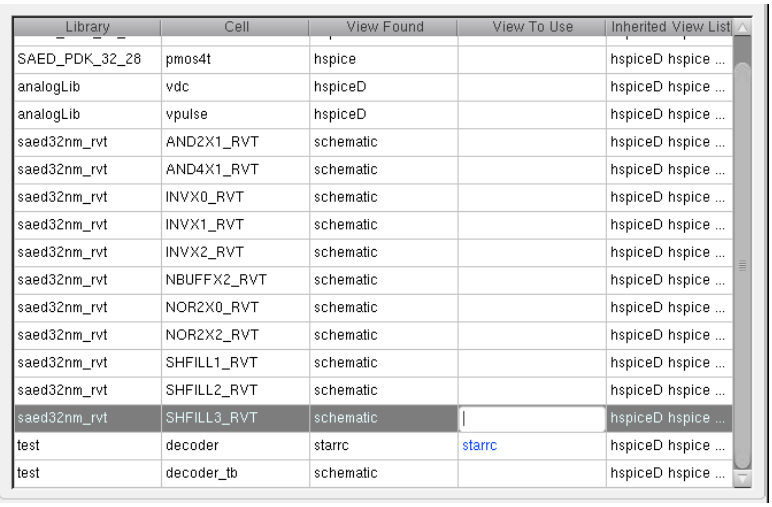
Cell Characterization

.lef/Milkyway generation

To generate a Library Exchange Format (LEF) representation of the custom NAND2 cell you've just made, use the Cadence Abstract Generation Tool.

```
% abstract &
```

This will open a new dialog box where we will select the cell we want to export. To make explaining this tool easier, I've added color-coded boxes over the important option buttons.



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.

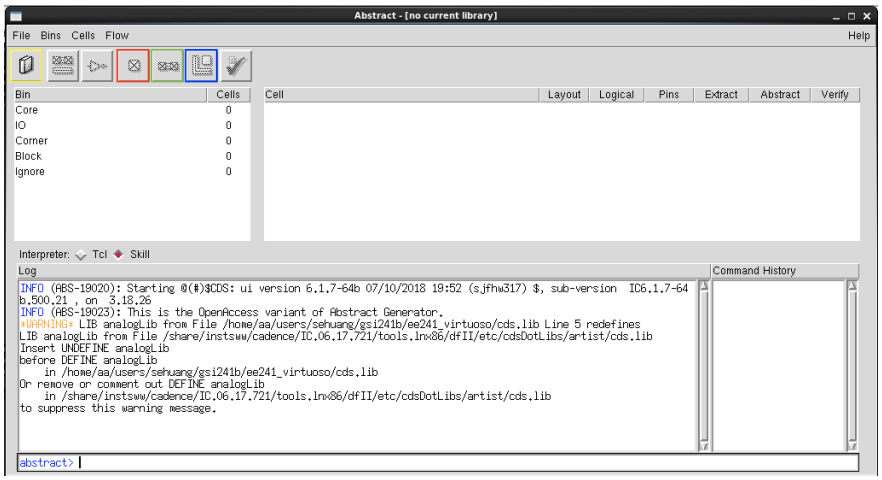


Figure 17: Abstract Generator

To load the NAND2X1B_RVT cell, select Open Library (yellow box), and select your Custom Cells library. You should see your NAND2X1B_RVT cell automatically loaded into the list on the right, with a green check mark under Layout. To process the cell, we will need to get the pins of the design and extract the rest of the layout.

Start by selecting Pins (red box). In the Output pin names field, enter the names of your output pins as a Regular Expression. In this case we only have one output, so put "Y". Once this is done, hit Run. Watch the output in the main window and make sure it has created a label for every pin in your design, including the power pins.

Next, we need to extract the signal layers of the layout. Select Extract (green box) In this case, we don't need to do anything special, but in future designs you may wish to deselect certain layers from your abstract design generation. For now, we can move on and hit Run. See that the tool has extracted all the nets in your design.

Next, we take the extracted nets and pins and create an abstract representation of our circuit. This step creates an abstract view that Virtuoso can read, but is not yet the more generic LEF format. Select Abstract (blue box). Again, there is nothing for us to do here, but if your design is more complex and has certain features that need to be accounted for, here is where you would set the tool to recognize them. Hit Run and see how the tool has found the power rails and generated blockages for all the existing routing in the cell.

Now to create a LEF file from the abstract view we made, go to File — Export — LEF and name your file "NAND2X1B_RVT.lef", then hit OK.

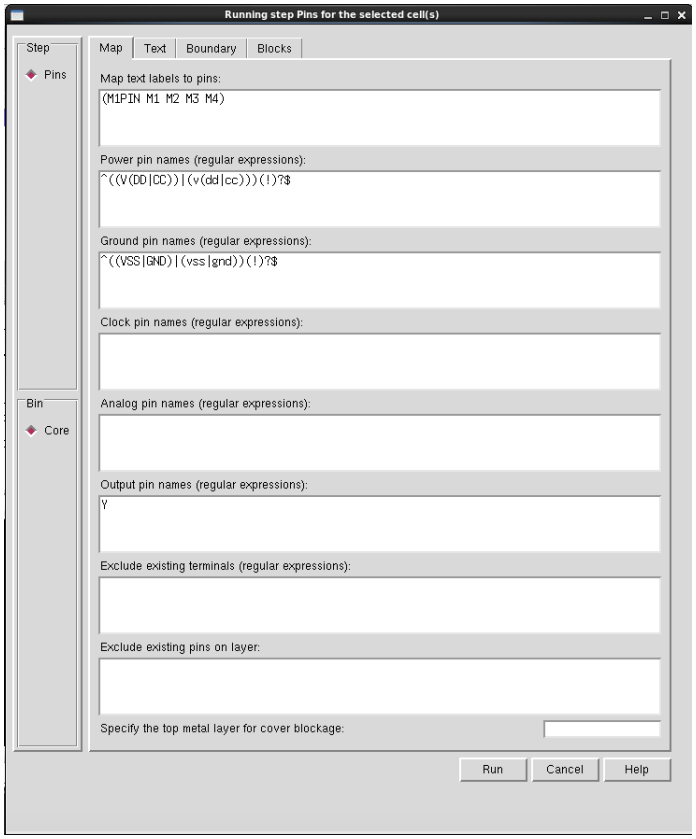


Figure 18: Pins Run Settings

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

Given the issues with StarRC, the library generation part of this lab cannot be completed. However, with a PDK that can be properly extracted, the following information is still useful, so I am leaving this in the lab.

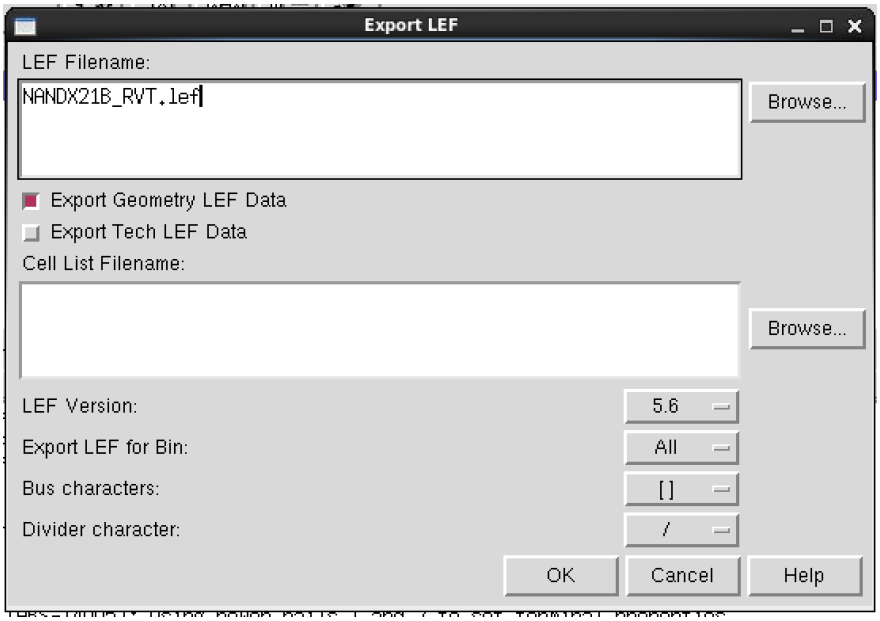


Figure 19: Abstract Export

You will need a netlist of devices to characterize. Open the starrc view 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) {
  related_power_pin : VDD ;
  related_ground_pin : VSS ;
  direction : "output" ;
  output_voltage : "DC_0" ;
  power_down_function : "!VDD + VSS" ;
  function : "(A2*A1)'" ;
  max_capacitance : 8 ;
  max_transition : 0.4527 ;
...
  timing() {
      related_pin : "A1" ;
      timing_sense : "negative_unate" ;
```

```
      cell_rise(del_1_7_7) {
        index_1("0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1.024");
        index_2("0.1, 0.25, 0.5, 1, 2, 4, 8");
        values("0.01203, 0.01257, 0.01344, 0.01501, 0.0181, 0.02438, 0.03723",\
               "0.01584, 0.0166, 0.01781, 0.02005, 0.02371, 0.03007, 0.0424",\
               "0.02184, 0.02271, 0.02417, 0.02706, 0.03172, 0.04, 0.05359",\
               "0.02858, 0.02928, 0.03026, 0.03353, 0.04053, 0.05237, 0.07068",\
               "0.04518, 0.04657, 0.04883, 0.05299, 0.06135, 0.0764, 0.0999",\
               "0.07155, 0.07241, 0.07387, 0.07782, 0.0875, 0.1041, 0.1339",\
               "0.128, 0.13, 0.1312, 0.1337, 0.1395, 0.1558, 0.187");
...
```

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 /scratch/userA/gen_stdcells/customcells/runtime/spice/NAND2X1B_RVT
% tar -xf delay__A1__hl__Y__lh__ACQ_1.tar.gz
% cd delay__A1__hl__Y__lh__ACQ_1.sif_0
% vim deck.cir
%
...
.data arc_data
+ slew_a1 load_y temperature_tag __param_vdd __param_vss
+ 1.6e-11 1e-16 25 1.05 0
+ 1.6e-11 2.5e-16 25 1.05 0
+ 1.6e-11 5e-16 25 1.05 0
+ 1.6e-11 1e-15 25 1.05 0
+ 1.6e-11 2e-15 25 1.05 0
+ 1.6e-11 4e-15 25 1.05 0
+ 1.6e-11 8.000000000000001e-15 25 1.05 0
+ 3.2e-11 1e-16 25 1.05 0
+ 3.2e-11 2.5e-16 25 1.05 0
+ 3.2e-11 5e-16 25 1.05 0
+ 3.2e-11 1e-15 25 1.05 0
+ 3.2e-11 2e-15 25 1.05 0
+ 3.2e-11 4e-15 25 1.05 0
+ 3.2e-11 8.000000000000001e-15 25 1.05 0
+ 6.4e-11 1e-16 25 1.05 0
+ 6.4e-11 2.5e-16 25 1.05 0
+ 6.4e-11 5e-16 25 1.05 0
+ 6.4e-11 1e-15 25 1.05 0
+ 6.4e-11 2e-15 25 1.05 0
+ 6.4e-11 4e-15 25 1.05 0
+ 6.4e-11 8.000000000000001e-15 25 1.05 0
...
```

This structure will run a simulation for each line in the .data section and effectively sweeps both dimensions of the table. Notice the second value in each line (1e-16, 2.5e-16, 5e-16, etc.) for the parameter load_y. This matches the values in index_2.

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

```
% cd /scratch/userA/gen_stdcells/
% cd customcells/runtime/spice/NAND2X1B_RVT/delay__A1__hl__Y__lh__ACQ_1.sif_0/
% wv deck.mt0 &
```

Then plot delay__a1__hl__y__lh.trig and delay__a1__hl__y__lh.targ, overlay them in the same plot, and you should see something similar to Figure 20.

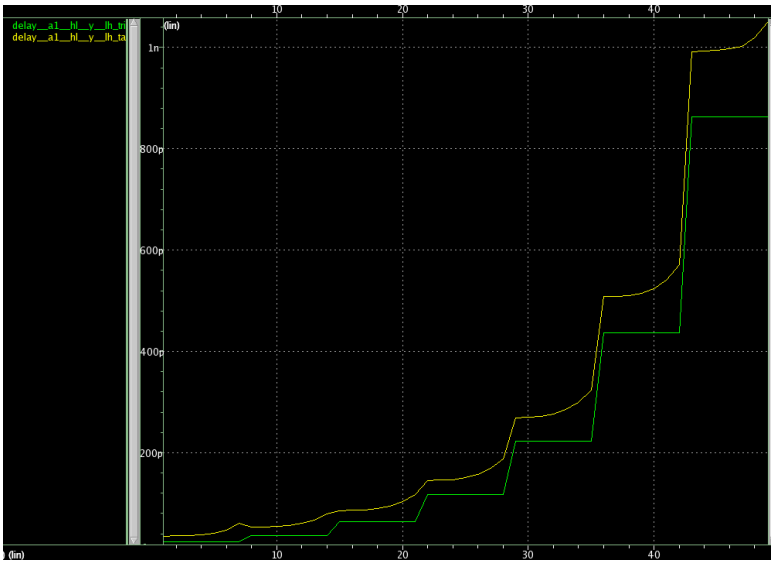


Figure 20: NCX characterization of a NAND2 gate

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).