Using VLSI Design Flow Outputs

EE241 Tutorial
Written by Brian Zimmer (2013)

1 Overview

In this tutorial, we will start with a fully place-and-routed 4-to-16 decoder created using the Synopsys VLSI design flow, import this design into Cadence Virtuoso, extract the design, and simulate it at the transistor level to verify predicted timing and energy results. The VLSI design flow uses a series of models and abstractions to generate circuits, so to truly understand the VLSI flow, it is instructive to remove these abstractions and compare the results in terms of timing and energy. Figure 1 shows a flowchart showing all of the conceptual steps in the tutorial.

2 Getting Started

2.1 Synopsys VLSI Flow

First, we will need to setup working directories for both the VLSI design flow. Run the commands below to setup the VLSI flow. On BWRC machines, replace ∼ee241 with /tools/designs/ee241 throughout this tutorial.

```
% source ~ee241/tutorials/ee241.bashrc
% cd /scratch/userA
% git clone ~ee241/tutorials/decoder_analysis
% cd decoder_analysis
% LABROOT=$PWD
```

This repository already contains a working 4-to-16 decoder, but you need to build the design yourself again. The following commands will run the entire flow to generate a place-and-routed design.

```
% cd $LABROOT/build-rvt
% make pt-pwr
```

Remember if you need to rerun any step in the process and the makefile claims that everything is up-to-date, you can run:

```
% make clean
```

2.2 Cadence Virtuoso

Next, run the commands below to setup a working directory for Cadence Virtuoso. Unlike the VLSI flow directory which will need to be copied for every different design, this directory only needs to be setup once and can be used for all of your projects.

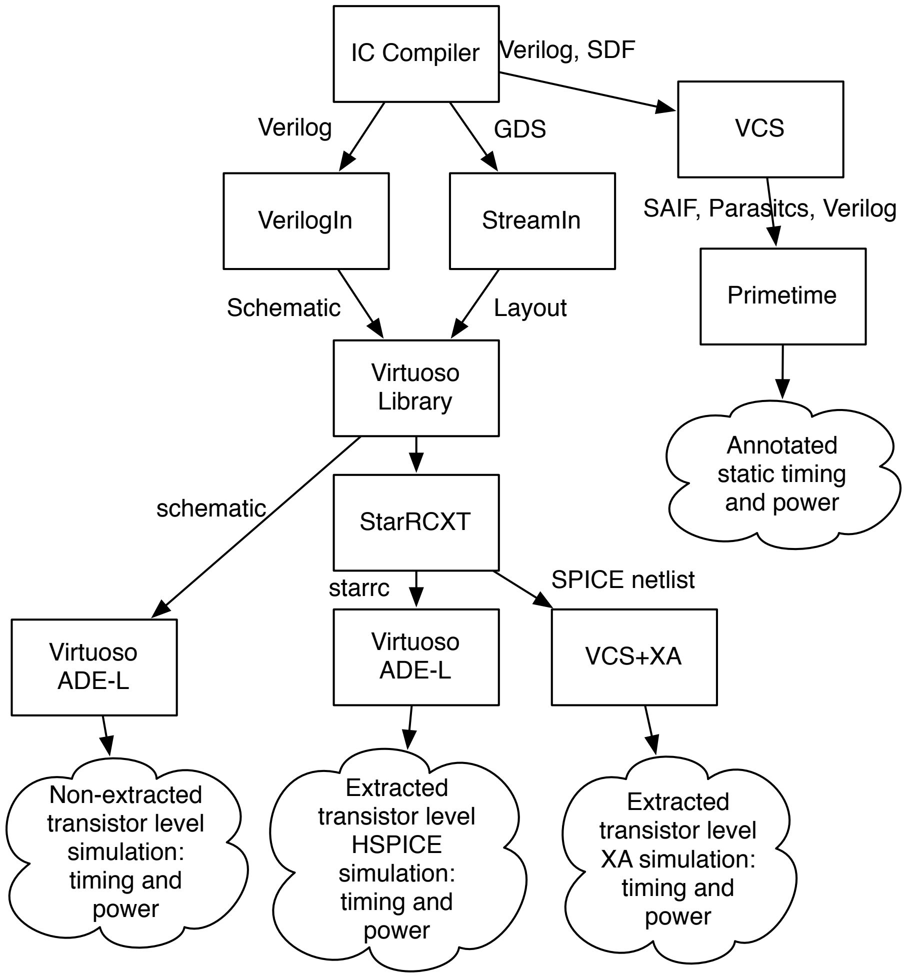

IC Compiler
Verilog, SDF
VCS
Verilog
GDS
VerilogIn
StreamIn
SAIF, Parasitcs, Verilog
Primetime
Schematic
Layout
Virtuoso Library
Annotated static timing and power
schematic
StarRCXT
starrc
SPICE netlist
Virtuoso ADE-L
Virtuoso ADE-L
VCS+XA
Non-extracted transistor level simulation: timing and power
Extracted transistor level HSPICE simulation: timing and power
Extracted transistor level XA simulation: timing and power

Figure 1: EE241 Toolflow for Lab 2

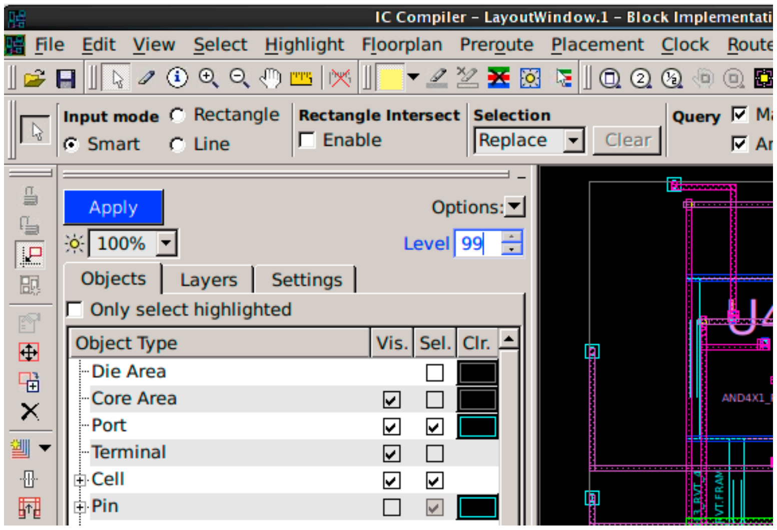
```
% cd /scratch/userA
% git clone ~ee241/ee241_virtuoso
% cd ee241_virtuoso
% VIRTUOSOLABROOT=$PWD
```

3 Translating layout

Open up the design in IC Compiler.

```
% cd $LABROOT/build-rvt/icc-par/current-icc
% ./start_gui
```

Notice that all can be seen is the wiring–the standard cells are all hidden. To expose the design, on the left panel increase the level to "99" and click "Apply", as shown in Figure 2.


IC Compiler - LayoutWindow.1 - Block Implementati
File Edit View Select Highlight Floorplan Preroute Placement Clock Route
Apply
Options:
Objects Layers Settings
Clear
AND4X1_

Input mode: Rectangle [✓ Smart] Line
Rectangle Intersect: ☐ Enable
Selection: Replace
Query: ☑ M ☑ Ar
100%
Level 99
☐ Only select highlighted

Object Type	Vis.	Sel.	Clr.
Die Area		☐	■
Core Area	☑	☐	■
Port	☑	☑	■
Terminal	☑	☐	
Cell	☑	☑	
Pin	☐	☑	■

Figure 2: Exposing additional levels.

You can now see the metalization inside each standard cell, as shown in Figure 3 (use "z" and the mouse to the zoom). But notice you still cannot see any devices–there are wells and poly but no active area. This will also be true of an IP blocks you place, such as memories. These views are only "black-box" views and only contain enough information to perform routing. To tape-out this design or perform transistor level extraction and simulation, we need all of the layers.

While IC Compiler is open, there are a few nice tools to use to better understand the output. For example, to highlight a path, go to Timing — New Timing Analysis Window. You can click "Apply" to see the worst paths, or set a specific path by setting the "From:" pin and "To:" pin. The TimingWindow will show the actual arrival time vs. the required arrival time, and report the difference as the "slack." Negative slack will mean there is a violated setup time. Click on a path, and click "Inspector" to see the path as part of the schematic. Then, to highlight the path on the layout, click on Highlight — Inspected Path (or Selected Path).

Now, open Cadence Virtuoso.

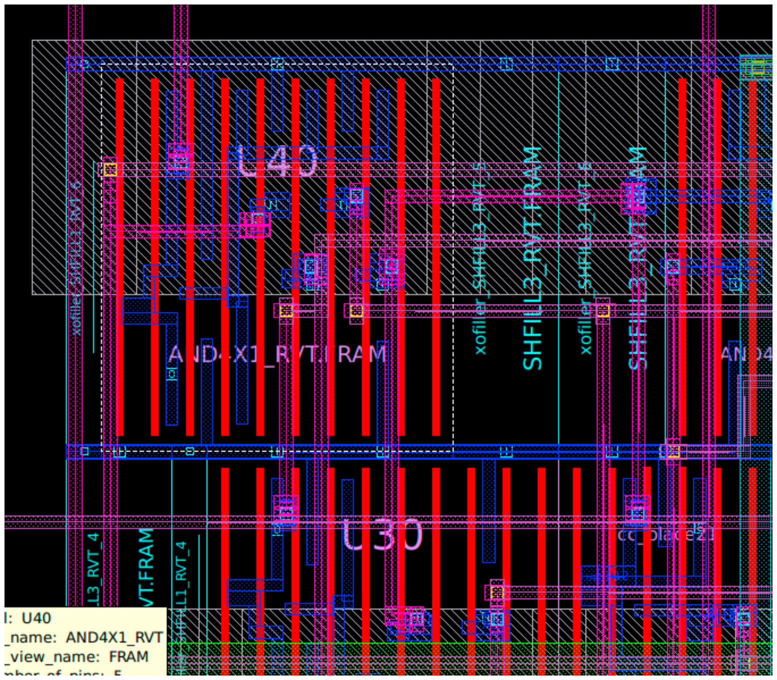


Figure 3: Blackbox view in IC Compiler.

```
% cd $VIRTUOSOLABROOT
% virtuoso &
```

Open the library saed32nm_rvt, then open the "layout" view of AND2X1_RVT. The layout is shown in Figure 4. Now press the ∼ key to view only the frontend layers. The views available inside Virtuoso include all of the needed layers and are called "full-views." You can cycle through other layers using the keys 1, 2, 3, etc to show that layer of metal and the via above it. Click "AV" to show all of the layers again.

We will use a GDS file format as an intermediate format to move from IC Compiler to Virtuoso. The GDS file will include instantiations of every standard cell, and we will intercept these instantiations and point them to use the "full-views" available in Virtuoso. The GDS file has already been generated inside icc-par/current-icc/results/decoder.gds.

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

Then go to File — Import Stream, then enter:

- File — Import — Stream
- "Stream File": icc-par/current-icc/results/decoder.gds
- "Library": The library you just created
- "Top Level Cell": change_names_icc
- "Attach Technolog Library": SAED_PDK_32_28
- Then click "Show Options"

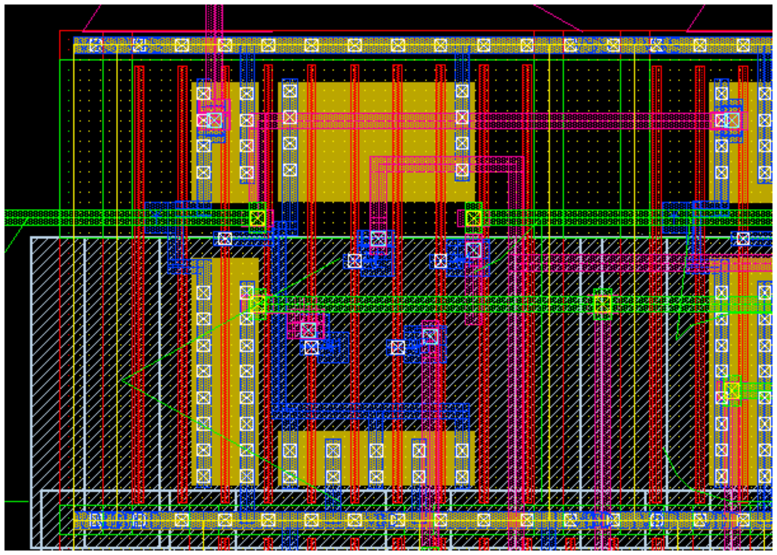


Figure 4: Full view in Virtuoso.

- Check the box next to "Replace [ ] with <>" (Note: if you do not see this option, open the layout after streamin and replace all pin label text square brackets with angle brackets)
- Click the "layers" tab.
- Under "map file", click open, and open ee241/synopsys-32nm/techfiles/saed32nm_1p9m_gdsout.map.bck
- Click on the "libraries" tab
- Click on saed32nm_rvt, then click on the right arrow
- Click the "cells" tab
- Click "Add row"
- Enter: Stream name: "change_names_icc", library "test" (or the name of the library you created), "cell": decoder, view: "layout"

Click apply, then open the layout view of the design in your new library. Press Shift-f to expose the standard cells so you can see the complete design, as shown in Figure 5.

4 Translating schematics

In order to run extraction or LVS on this design, we need to generate schematics for this design.

Note that this design started as RTL level verilog. This is useless as a schematic, as there are no standard cells yet. We could use synthesis output because this includes structural Verilog, yet there is no clock tree and ICC can change cells in order to fix timing, so it will not match the layout. Therefore we need to use the structural Verilog generated by IC Compiler. Open this output in your favorite editor with the following commands

```
% cd $LABROOT/build-rvt/icc-par/current-icc/results
% vim decoder.output.v
```

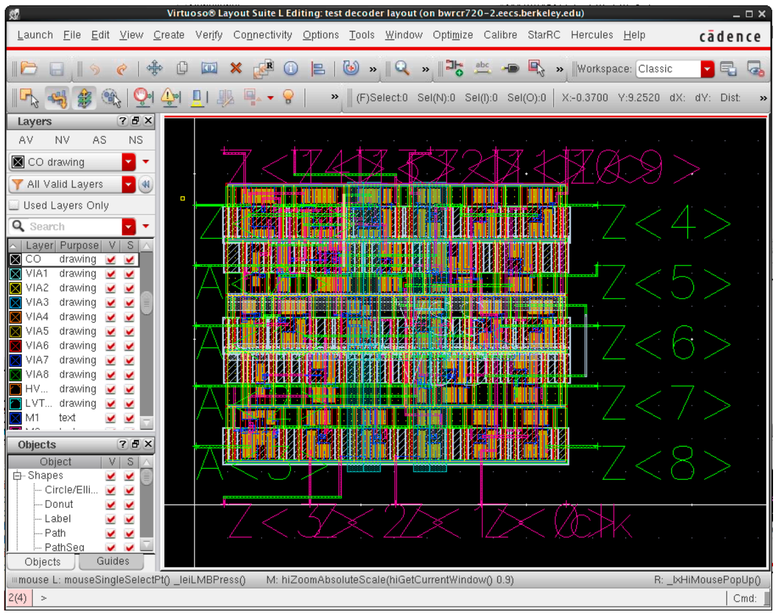

Virtuoso® Layout Suite L Editing: test decoder layout (on bwrcr720-2.eecs.berkeley.edu)
Launch File Edit View Create Verify Connectivity Options Tools Window Optimize Calibre StarRC Hercules Help
cādence
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:-0.3700 Y:9.2520 dX: dY: Dist:
Layers
AV NV AS NS
CO drawing
All Valid Layers
Used Layers Only
Search
Layer Purpose V S
CO drawing
VIA1 drawing
VIA2 drawing
VIA3 drawing
VIA4 drawing
VIA5 drawing
VIA6 drawing
VIA7 drawing
VIA8 drawing
HV... drawing
LVT... drawing
M1 text
Objects
Object V S
Shapes
Circle/Elli...
Donut
Label
Path
PathSeg
Objects Guides
Z<4>
Z<5>
Z<6>
Z<7>
Z<8>
mouse L: mouseSingleSelectPt() _leiLMBPress() M: hiZoomAbsoluteScale(hiGetCurrentWindow() 0.9) R: _lxHiMousePopUp()
2(4) > Cmd:

Figure 5: Entire decoder imported into Virtuoso.

Notice that each standard cell does not include power and ground. Power and ground was not in the input netlist, and power is treated specially in IC Compiler.

Go back to Virtuoso, and open the schematic and symbol views of the AND2X1_RVT cell in saed32nm_rvt. Notice that this view expects VDD and VSS, because a transistor level netlist needs to know what net is connected to the transistor sources. Therefore our Verilog description needs to include information about power as well. There is another file in the output of IC Compiler that includes this information.

```
% cd $LABROOT/build/icc-par/current-icc/results
% vim decoder.pg.lvs.v
```

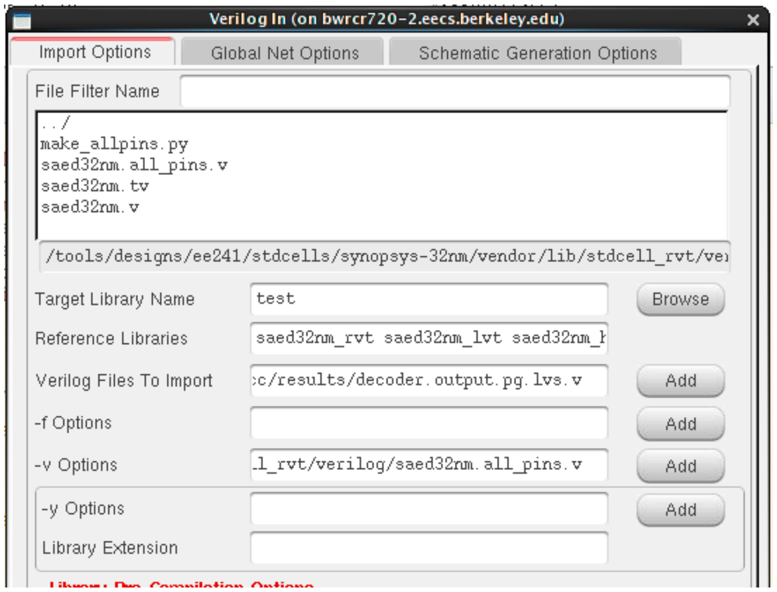
Notice that this is still a Verilog netlist. Just like the layout from IC Compiler has no information about the actual devices, this Verilog netlist has no information about transistors. However, Virtuoso's schematics are a "full-view" that include the devices, so during import to Virtuoso, we will need to reference the standard cell library as we did for the layout import.

Go to File — Import — Verilog, then and load "verilogin_template". The form should look like Figure 6.

You will need to change:

- Target Library Name: your new library name
- Verilog file to import: icc-par/current-icc/results/decoder.output.pg.lvs.v
- -v Options: ~ee241/stdcells/synopsys-32nm/vendor/lib/stdcell_rvt/verilog/saed32nm.all_pins.v

Click apply, then open the generated schematic. Virtuoso has instantiated all of the standard cells and routed all of the nets.


Verilog In (on bwrcr720-2.eecs.berkeley.edu)
Import Options
Global Net Options
Schematic Generation Options
Browse
Add
Add
Add
Add

File Filter Name

```
../
make_allpins.py
saed32nm.all_pins.v
saed32nm.tv
saed32nm.v
```

/tools/designs/ee241/stdcells/synopsys-32nm/vendor/lib/stdcell_rvt/ve

Target Library Name: test

Reference Libraries: saed32nm_rvt saed32nm_lvt saed32nm_h

Verilog Files To Import: c/results/decoder.output.pg.lvs.v

-f Options:

-v Options: l_rvt/verilog/saed32nm.all_pins.v

-y Options:

Library Extension:

Library: Pre-Compilation Options

Figure 6: VerilogIn dialog settings.

5 LVS

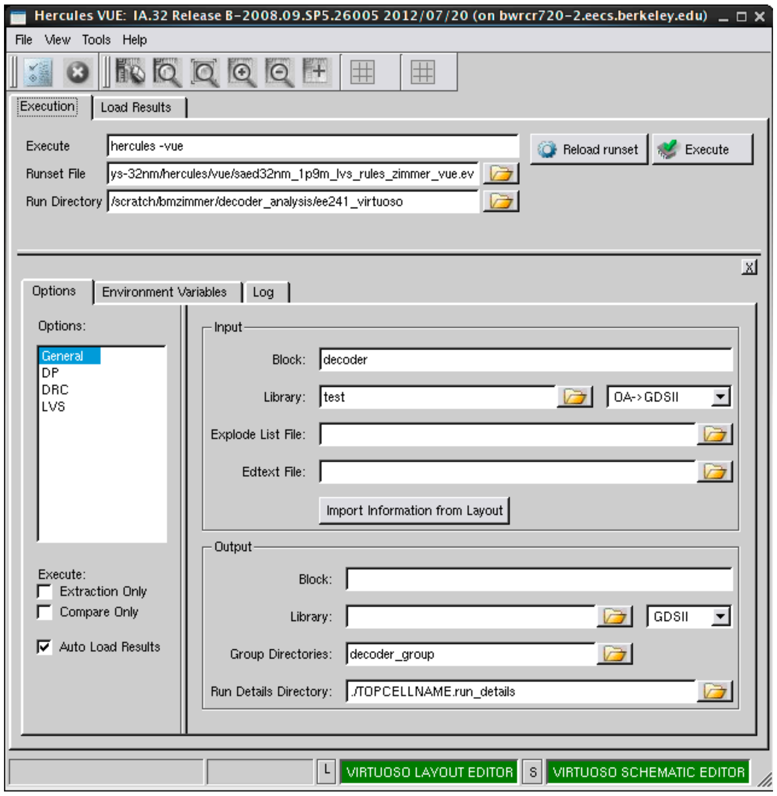
In order to perform parasitic extraction of a design, the design must pass layout-versus-schematic (LVS). This is because the parasitics caused by polygons in layout need to be annotated to their equivalent net in the schematic.

To run LVS, open the layout view and go to Hercules — Run LVS. Wait a few seconds for the "Block" entry to be filled in automatically as shown in Figure 7, then press "Execute." You should see LVS pass as shown in Figure 8. Once you have verified that the design has been imported correctly (and ICC has done its job correctly), you can run extraction.

6 Extraction

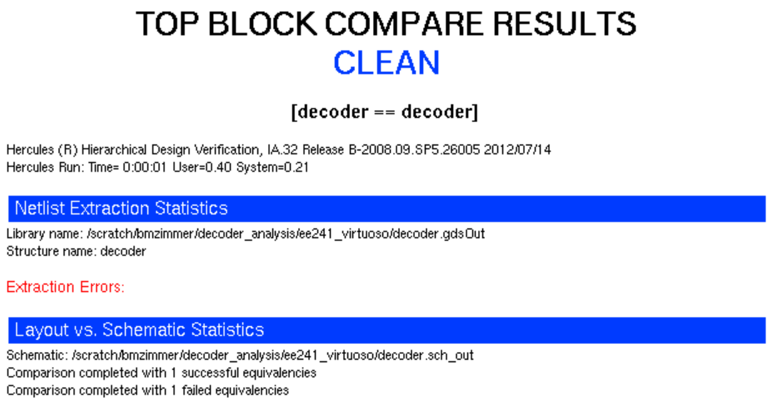
To run parasitic extraction in Virtuoso, open the layout view and go to StarRC — Parasitic Generation Cockpit. The setting should be automatically loaded for you as shown in Figure 9, but you will need to change one thing: Click on the "Output Parasitics" tab, and check the box next to "Ports Annotation", then enter in your library and cell name. If you do not do this, you will not be able to netlist the design. Some other 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
- Extract Parasitics Tab — Additional Options, allows you to set a threshold to try to reduce device count


Hercules VUE: IA.32 Release B-2008.09.SP5.26005 2012/07/20 (on bwrcr720-2.eecs.berkeley.edu)
File View Tools Help
Execution
Load Results
Reload runset
Execute
Options
Environment Variables
Log
General
DP
DRC
LVS
Import Information from Layout
VIRTUOSO LAYOUT EDITOR
VIRTUOSO SCHEMATIC EDITOR

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

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
```

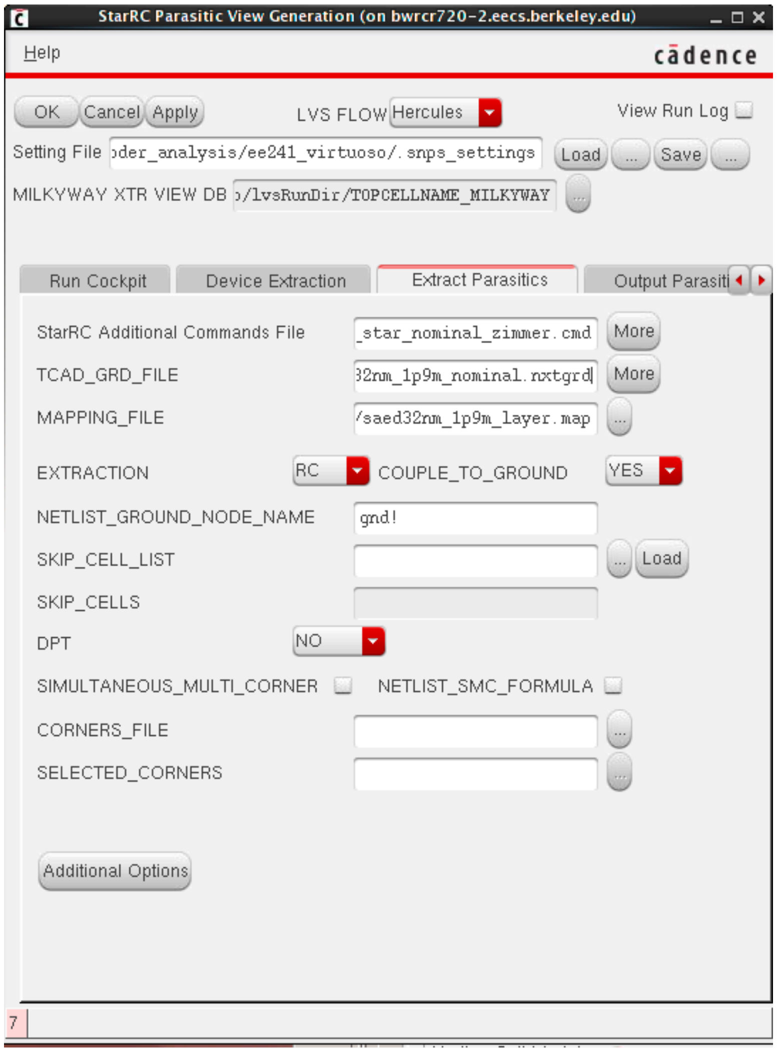
Figure 7: Hercules LVS dialog settings.



```
TOP BLOCK COMPARE RESULTS
CLEAN
[decoder == decoder]

Hercules (R) Hierarchical Design Verification, IA.32 Release B-2008.09.SP5.26005 2012/07/14
Hercules Run: Time= 0:00:01 User=0.40 System=0.21

Netlist Extraction Statistics
Library name: /scratch/bmzimmer/decoder_analysis/ee241_virtuoso/decoder.gdsOut
Structure name: decoder

Extraction Errors:

Layout vs. Schematic Statistics
Schematic: /scratch/bmzimmer/decoder_analysis/ee241_virtuoso/decoder.sch_out
Comparison completed with 1 successful equivalencies
Comparison completed with 1 failed equivalencies
```

Figure 8: Decoder passing LVS.

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 as shown in Figure 10. By going to StarRC — Parasitic Prober you can explore the parasitics in your design.



StarRC Parasitic View Generation (on bwrcr720-2.eecs.berkeley.edu)

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

7

Figure 9: StarRC settings.

7 Simulation in Virtuoso

Go to File — New — Cell View, call it decoder_tb, and make the view and type "schematic". Press 'i', then insert the symbol view of your newly imported decoder. 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 11. Note that you need to instantiate an array of noConn by pressing q while the noConn block is selected and changing the instance name, as shown in Figure 12. Set the VPWL as follows:

- A[3]: 3 pairs of points, T1: 0, V1: 1.05, T2: 500p, V2: 1.05, T3: 510p, V3: 1.05
- A[2]: 3 pairs of points, T1: 0, V1: 0, T2: 500p, V2: 0, T3: 510p, V3: 1.05
- A[1]: 3 pairs of points, T1: 0, V1: 1.05, T2: 500p, V2: 1.05, T3: 510p, V3: 1.05
- A[0]: 3 pairs of points, T1: 0, V1: 1.05, T2: 500p, V2: 1.05, T3: 510p, V3: 1.05

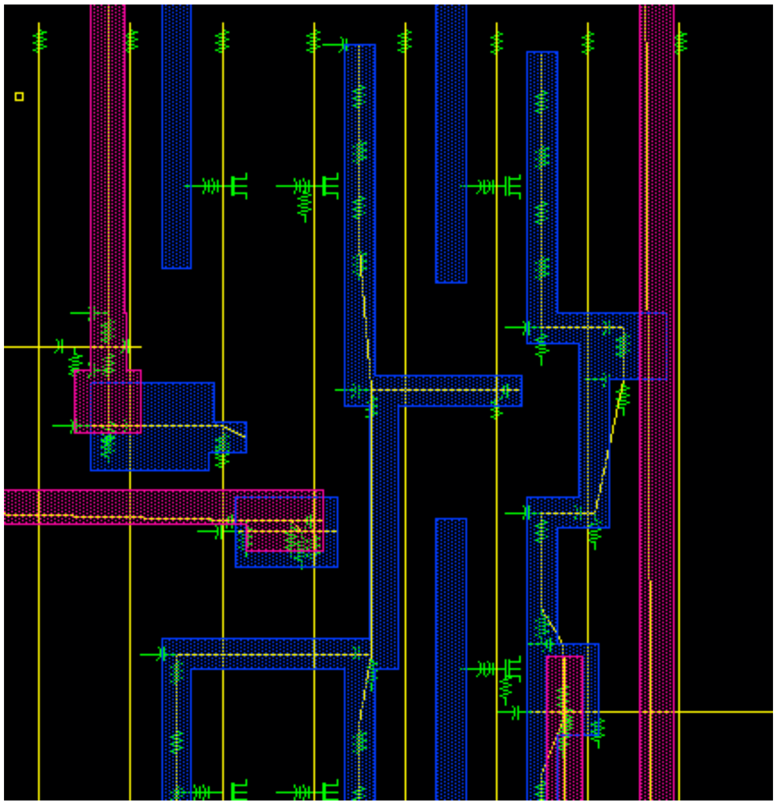


Figure 10: Extracted resistors and capacitors annotated on layout.

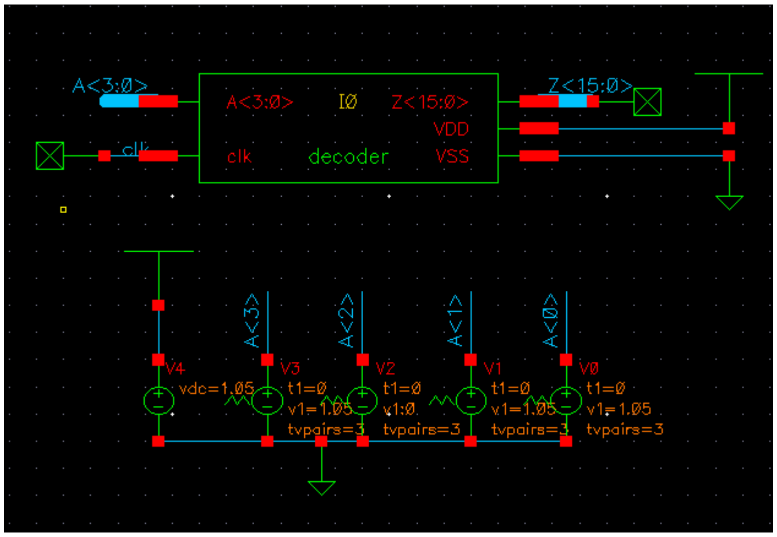
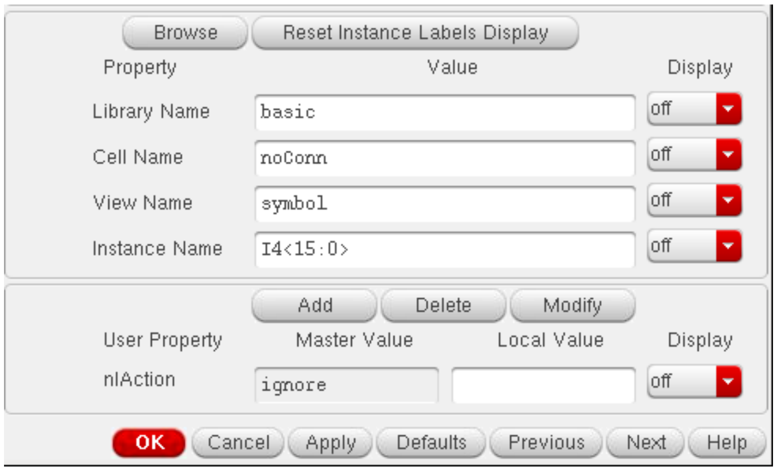


Figure 11: Testbench to measure a timing path.

Go to File — Check and Save, and there should be no errors or warnings. 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



Property	Value	Display
Library Name	basic	off
Cell Name	noConn	off
View Name	symbol	off
Instance Name	I4<15:0>	off

User Property	Master Value	Local Value	Display
nlAction	ignore		off

Figure 12: Creating an array of noConn instances.

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

Then click enter the settings shown in Figure 13. 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. Next, go to Analysis — Choose, and set the stop time to 1n, 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 you should see a result like Figure 14.

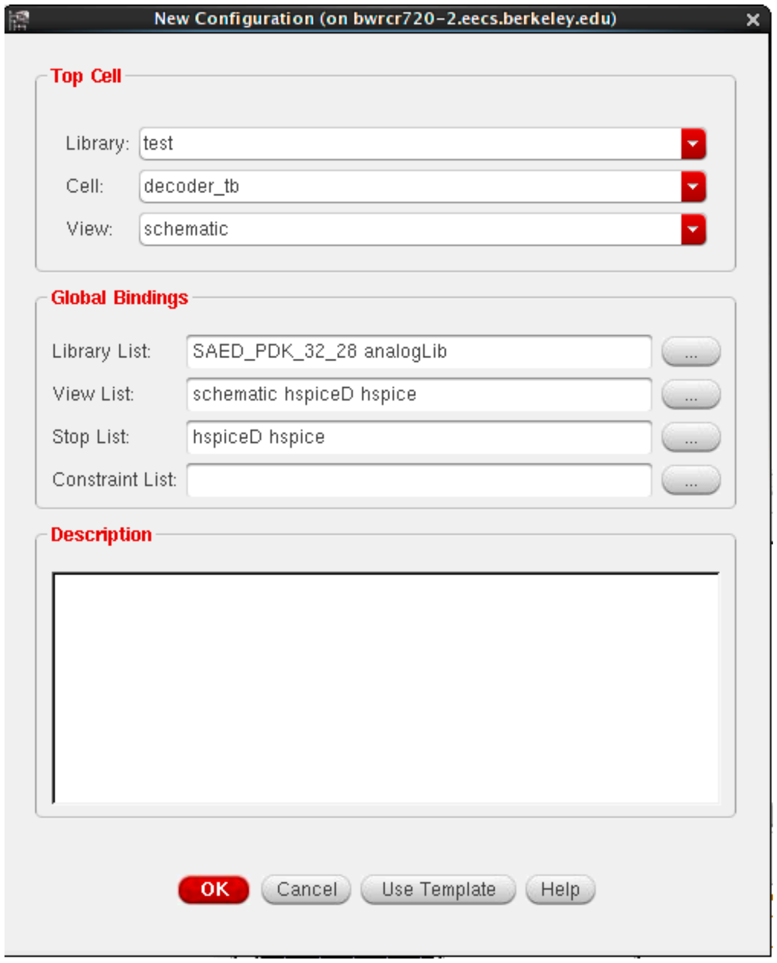
To measure the path we are measuring (A[2] rising to Z[15] rising), mouse over the first edge at the 50% point and press "a", then go to the second edge and press "b".

Your result is only measuring the devices. Now, we would like to simulate this same path, but also annotate the parasitics we calculated during extraction.

Open the config view of decoder_tb, (click "yes" for Configuration config), then in the "View to Use" column for the "decoder" instance, enter starrc as shown in Figure 15. Then go to File — Save.

Go back to ADE L and rerun the simulation. The delay should now increase because the parasitics of the wires are now included. Go to File — Save State, then save inside "cellview" so you can open this simulation later. By going to Simulation — Netlist — Display, you can make sure the parasitics are being included within the spice deck.

Note that IC Compiler and Primetime perform timing on all paths. However, many paths in the decoder are false paths. For example, the path A[3] falling to Z[2] falling is a false path. Because it is a one hot decoder, a high A[3] means that only Z[15:8] could be high, so there is no possible way for Z[2] to fall.



Top Cell

Library: test

Cell: decoder_tb

View: schematic

Global Bindings

Library List: SAED_PDK_32_28 analogLib

View List: schematic hspiceD hspice

Stop List: hspiceD hspice

Constraint List:

Description

Figure 13: Settings for the new config view.

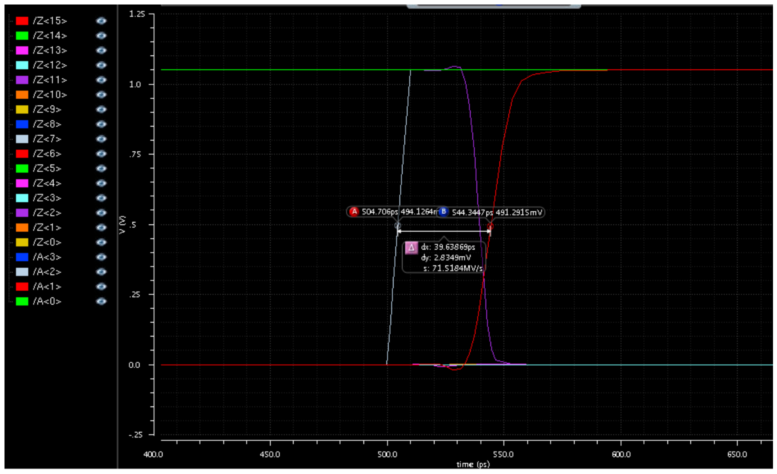
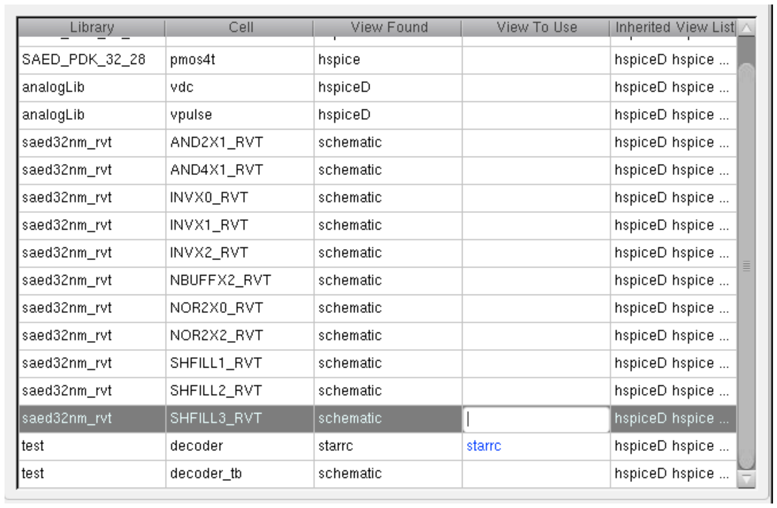


Figure 14: A[2] rise to Z[15] rising timing path for schematic simulation.



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 15: Change the config view to use the extracted netlist.

7.1 Energy measurement

At the beginning of the tutorial when you typed "make pt-pwr", you should simulated the post-place-and-route netlist with annotated parasitics and performed power analysis on your design. Go back to the simulation to see what activity the testbench generated.

```
% cd $LABROOT/build-rvt/vcs-sim-gl-par
% make convert
% dve -vpd vcdplus.vpd &
```

Then plot both the A and Z signals. You should see the decoder is driven with A counting from 0000 to 1111 as shown in Figure 16. By running "make convert", the switching activity of every node is annotated in a SAIF file. Then Primetime (in pt-pwr) combines this with the icc-par netlist and parasitics file (SBPF) to accurately measure energy consumption. For example, look at the generated SAIF file (switching activity).

```
% cd $LABROOT/build-rvt/vcs-sim-gl-par
% make convert
% cat vcdplus.saif
(INSTANCE U34
      (NET
        (A1
          (T0 1600) (T1 1400) (TX 0)
          (TC 1) (IG 0)
        )
        (A2
          (T0 1400) (T1 1590) (TX 10)
          (TC 14) (IG 0)
 ....
```

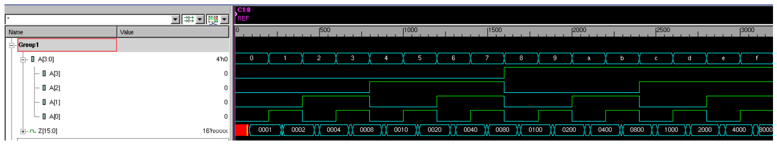


Figure 16: Test vector generated by Verilog testbench.

TC counts the number of toggles, and is used to calculate dynamic energy. T1, T0, and TX correspond to time at logic 1, 0, and X and are used to calculate leakage energy (leakage will be state dependent).

Now look at the Primetime energy report:

```
% cd $LABROOT/build-rvt/pt-pwr
% make
% cat current-pt/reports/vcdplus.timing.avg.max.report

                                          Switch   Int      Leak      Total
Hierarchy                                 Power    Power    Power     Power    %
--------------------------------------------------------------------------------
decoder                                   3.09e-05 3.41e-05 2.22e-06  6.72e-05 100.0
```

So for the 3.2ns testbench, the average power was 67uW for the decoder.

Now, back in Virtuoso, recreate the same testbench inside decoder_tb as shown in Figure 17 using vpulse with periods of 400ps, 800ps, 1600ps, and 3200ps for A[0], A[1], A[2], and A[3], and pulse time as half of the period. You might want to backup your old testbench. Launch ADE again, and run a transient simulation for 3.2ns. Probe the current through the VDD current source. Then go to Outputs — Setup, click New Expression, and set the Name to be "power", then next to calculator click Open. Click the "it" button, then select the positive port of the VDD voltage source. Then in the function panel, click "Average", then append "1.05" to the expression in the textbox. Without closing the calculator, go back to the Setup Output Options dialog and click "Get Expression". The box will look like Figure 18. Click "Add", then "Ok". Now run the simulation. In the outputs pane, you should see a number for your power. This is the power measured by a spice simulation of the extracted netlist.

8 Mixed-mode simulation

For small designs, full HSPICE simulation in Virtuoso using handcrafted testbenches is easy. However, for larger designs, it is very useful to stimulate the simulation with a Verilog testbench, using a faster simulator, such as XA. Then critical analog components such as PLLs, SRAMS, and asynchronous interfaces can be simulated at the transistor level while the basic digital logic can be simulated more quickly with VCS.

Another reason to run mixed-mode simulations is to measure the effects of voltage scaling on energy. While Primetime works well for measuring power of digital logic, voltage scaling studies require

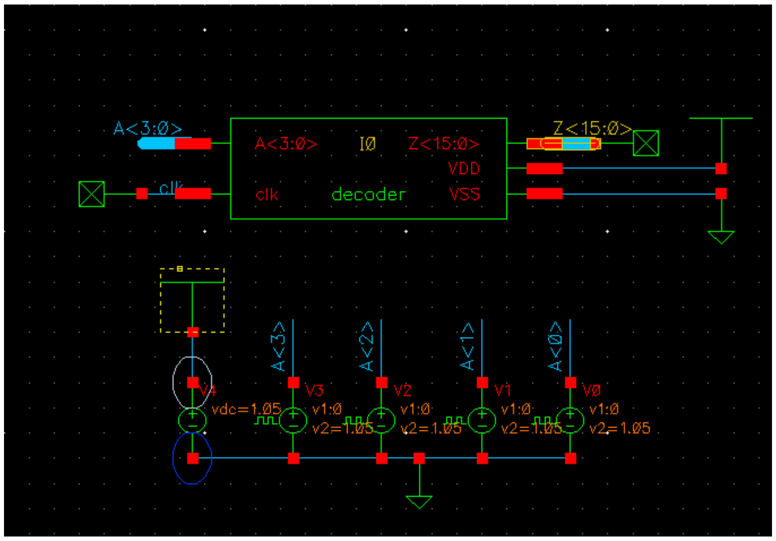
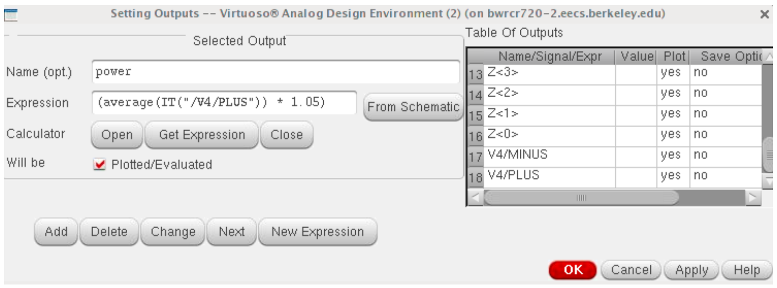


Figure 17: Test bench inside Virtuoso that emulates the Verilog testbench.


Setting Outputs -- Virtuoso® Analog Design Environment (2) (on bwrcr720-2.eecs.berkeley.edu)

Selected Output

Name (opt.)	power
Expression	(average(IT("/V4/PLUS")) * 1.05)
Calculator	
Will be	✓ Plotted/Evaluated

Table Of Outputs

	Name/Signal/Expr	Value	Plot	Save Optio
13	Z<3>		yes	no
14	Z<2>		yes	no
15	Z<1>		yes	no
16	Z<0>		yes	no
17	V4/MINUS		yes	no
18	V4/PLUS		yes	no

Figure 18: Create a measure statement to calculate average power.

all of the standard cells to be characterized at different voltages, and these cells are not always available.

In ADE L, go to Simulation — Netlist — Create. A netlist with parasitics should appear. Now to go File — Save As, and place this design in the $LABROOT/build-rvt/vcs-sim-tl-pex directory as decoder.sp. This directory is a new directory that will run a mixed simulation.

Look at the Makefile in this directory.

```
% cd $LABROOT/build-rvt/vcs-sim-tl-pex
% cat Makefile

srcdir = $(basedir)/src
vsrcs = \
  $(srcdir)/decoder_tb.v

...
```

```
$(vcs_sim) : Makefile $(vsrcs)
  $(VCS) $(VCS_OPTS) +incdir+$(srcdir) -o $(vcs_sim) \
         +define+CLOCK_PERIOD=$(vcs_clock_period) \
        $(vsrcs)

  ...

  VCS_OPTS = -notice -PP -line +lint=all,noVCDE,noTFIPC,noIWU,noOUDPE \
    +v2k -timescale=1ns/1ps \
    -P ../icc-par/current-icc/access.tab -debug \
    +neg_tchk \
    -ad=vcsAD.init -lca
```

The -ad=vcsAD.init line is the magic line that enables mixed signal simulation. Think of the vcsAD.init file as the config file we used in Virtuoso to choose between schematic only and parasitic simulation. You can replace $(vcs_clock_period) to define a clock period in terms of ns.

The only Verilog source is the same testbench as before, which instantiates the decoder with:

```
  decoder DUT0 (.A(A), .Z(Z), .clk(clk));
```

Now look at the configuration file. This defines the analog to digital interface, tells VCS to use the Synopsys XA simulator (a transistor level simulator like HSPICE but optimized for speed), and to use spice on the "decoder" cell corresponding to the instantiation above. Also, Verilog has bus format for signals (eg. A[3:0]) but SPICE has no buses, so the tool must know how to convert between formats to hook up the ports correctly. xa_options tells the simulator to merge all probed signals with the VPD file so you can see digital and analog signals together in the same tool.

```
  % cat vcsAD.init
  choose xa -hspice decoder.sp -c xa_options -o xa;
  use_spice -cell decoder;
  bus_format <%d>;
  d2a rf_time=10p rise_time=20p fall_time=20p delay=0 hiv=1.05 lov=0.0 node=*;
  a2d loth=0.3v hith=0.7v node=*;
```

Next we need to hookup the supplies within the decoder. While there are cleaner ways to do this, we will do something easy for now. Edit decoder.sp. Delete all lines (except the first line) before the .subckt definition, and all lines after the .ends at the end of the circuit. *Make sure the first line is a comment.* Note that if your power measurment fails, you need to make sure your FROM and TO values are within the simulation time. Then add a path to the library, and place voltage sources as shown below:

```
  % cat decoder.sp

  *  Decoder
  .lib /tools/designs/ee241/synopsys-32nm/hspice/saed32nm.lib TT

  .subckt decoder a<3> a<2> a<1> a<0> vdd vss z<15> z<14> z<13> z<12> z<11>
```

```
+ z<10> z<9> z<8> z<7> z<6> z<5> z<4> z<3> z<2> z<1> z<0> clk
Vvdd vdd 0 DC=1.05
Vvss vss 0 DC=0
.probe tran i(Vvdd)
.measure pavg AVG i(Vvdd) FROM=0 TO=3.2n

    cg1431 u26|mp3|drn 0 c=85.1142e-18
```

Last, run the simulation and see the power measured by XA.

```
% make
% make run
% cat xa.meas
% dve -vpd vcdplus.vpd
```

9 Conclusion

In this tutorial, you investigated the accuracy of IC Compiler and Primetime reports. In order to run a fully-extracted transistor-level simulation, you need to import both the GDS and Verilog netlist generated by IC Compiler into a Virtuoso schematic and layout. Then using Hercules you run LVS to verify the import, and using Synopsys StarRCXT you extract parasitics from the layout. With this information, you can simulate a path and compare its delay between schematic-only simulation, extracted simulation, and IC Compiler estimates. Also, you can measure energy based on extracted simulation results, primetime, and a mixed signal simulation of an extracted netlist through a Verilog testbench.