Jazz Digital Design Manual

Rev. 9.0
Doc Number: NPB PS-0402

Table of Contents

1 Revision History

Date	Rev	Author	Purpose
03/10/03	0.1	Sunvir Gujral	• 1.0 Release Candidate
03/12/03	0.2	Sunvir Gujral	• Added section on SE-to-dfII
03/19/03	0.3	Sunvir Gujral	• Added section on Post-Feb03 M/S LVS methodology
03/24/03	0.4	Sunvir Gujral / Winnie Ng	• Cleaned up section on Assura-LVS flow
04/08/03	0.5	Sunvir Gujral / Kashyap Bellur	• Cleaned up section on Characterization • Added to Memory section
04/21/03	0.6	Sunvir Gujral	• Added section on Verilog In
05/06/03	0.7	Sunvir Gujral / Winnie Ng	• Added Inherited Connections flow
06/04/03	0.8	Sunvir Gujral	• Added FAsCOT flow, added patch installation instructions
06/06/03	1.0	Sunvir Gujral	• Initial release under QSI document mgmt system
07/10/03	2.0	Sunvir Gujral / Felix Tom	• Changed library names to 'A'-style naming convention • Added A35 library documentation • Added support for Apollo • Updated FAsCOT flow diagram • Updated Silicon Ensemble Flow sub-section • Updated Inherited Connections sub-section
08/25/03	2.1	Sunvir Gujral	• Updated SE and Wroute methodologies; Verilog Import; Calibre LVS; Memory sections • Added Power guidelines
02/05/04	3.0	Felix Tom / Troy Rossean	• Added table of core and I/O library names • Updated supported process technologies list • Updated tools version list • Added Apollo P&R Flow sub-section • Updated Place and Route, Verilog Import, Mixed-signal Simulation, Calibre LVS, and Assura LVS sections • Miscellaneous text editing and formatting changes
04/13/04	4.0	Winnie Ng	• Added chapter on Mixed-Signal LVS using Assura 3.1.2
05/14/04	5.0	Sunvir Gujral	• Fixed FAsCOT diagram on page 8
11/29/04	6.0	Felix Tom	• Updated supported process technologies list • Updated tools version list • Fixed net name inconsistencies in Verilog Import section • Updated SE GDSII→dfII sub-section • Updated Assura LVS section with additional memory-related steps • Updated Memory Generation section

Date	Rev	Author	Purpose
01/27/05	7.0	Felix Tom	• Updated supported process technologies table • Updated tools version table • Updated section 8 with details on SDF version and options required for compatibility with Verilog models • Updated section 12.2 with Verilog netlisting options • Updated section 13.1 with Calibre Interactive LVS • Updated section 13.2 with SE Verilog netlist export • Added new section 16 on simulation and LVS with 2- and 3-rail library cells on the same design. • Updated section 17.2 with work-arounds for missing LEF routing blockages and CDL netlist pin order mismatches
02/08/06	8.0	Felix Tom	• Updated supported process technologies table (section 3) • Removed the requirement for SE 5.4.35 to export DEF (sections 5 and 9.1) • Added timing check diagrams (section 6) • Added more details on generating SDF files (section 8) • Added more details on making an imported Verilog netlist ready for LVS (section 11.1) • Added information on displaying available inherited connection properties (section 11.2) • Added a description of the conditions that would lead to Poly antenna violations on a compiled ROM (section 17.2.5) • Removed index (section 19)
03/12/07	9.0	Felix Tom	• Updated core and I/O library names table (section 2.1) • Updated supported process technologies table (section 3) • Updated DEF import procedure to add step to change maskLayout units (section 9.1.3.1)

2 Introduction

This document is targeted towards users of Jazz digital libraries. There are separate deliverables for each of Jazz's major process technology nodes based on minimum transistor size: 0.18µm (A18, A30), 0.25µm (A25H, A25L), and 0.35µm (A35, A50H).

NOTE: The A18 and A35 libraries were previously referred to as the "cg18" and "sbc35" libraries. They are the same libraries, but the 'A' designation is being used now to prevent confusion with Jazz process names.

2.1 Library Contents & Directory Structure

A library deliverable is comprised of views for most of the major digital/ASIC flows. Jazz has verified the views with a subset of these flows and will provide support to the customer on these flows only. Supported tools and versions are listed in Table 2 of section 5. Supported flows and recommended methodologies are outlined throughout this document.

A library is delivered in a single tarred and compressed file. The naming convention for this file is *<libname>_<version>.tgz*. For example, the d09 version of the A18 library is named *A18_d09.tgz*.

The standard core cell and I/O buffer components within a library are identified with the names given in the following table.

Table 1. Core Cell and I/O Buffer Library Names

Library	*Core Cell Libraries*	*I/O Buffer Libraries*
A13	cscd	io60u3v
A13LP	fs130	
A18	scgp scgp_analog	io60u5v io80u5v io160u5v
A18LP	scm	
A25B	ri54sy215	ri54io200
A25H	ri25sy100 ri25sy100_analog	ri25io800
A25L	ri28sy100	ri28io800
A30	scd scd_analog	io60u5v
A35	ri35sy101 ri35sy101_analog	ri35io111
A50 / A50H	ri54sy215	n/a
A50A		ri54io200

where the *_analog nomenclature denotes a library with separate NMOS source and substrate connections.

2.1.1 Directory Structure

A library is organized into separate sub-directories for each of the tools and/or views. There is also a *RELEASE_INFO* directory that contains release information and other important information for the current and previous releases.

The following top-level directories could be present in a library. Note that all directories might not be present in some libraries.

- apollo (or astro)
- calibre
- dfII
- doc
- dp
- gds
- hspice
- lef
- mgc_dft
- process
- spectre
- star
- synopsys
- techs
- tlf
- verilog
- vital

2.1.2 Using dfII Libraries

If the dfII libraries in *$JAZZLIBS/$LIBNAME/dfII* are to be used, the path for each of them must be added to the appropriate cds.lib file. In earlier releases (prior to mid-2003) these libraries also had to be untarred before using them.

2.2 Installation

To install a library for the first time, the downloaded file must be uncompressed and untarred into the directory where it will be installed (*$JAZZLIBS* is used throughout this document).

- cd $JAZZLIBS
- mkdir A18
- cd A18
- mkdir d09 ; cd d09 (optional)
- gtar –xzf A18_d09.tgz

To install patches, please see section 2.3 below.

Note that the tarfile will overwrite the current directory. If different versions of the same library are being maintained, the previous version should be moved out of the way or a new directory created in which to untar the new release.

2.3 Patch Installation

Patches to Jazz libraries are delivered in between major releases and are identified by a letter and ".patch" following the major release number. For example, the first patch after the d11 release of the A18 library is called *A18_d11a.patch*. A patched library will always have a letter following the release number.

A patch release only contains files that have changed since the previous release. The first time a library is to be installed, a fully patched version can be requested from Jazz. If a library is already installed and a new patch becomes available, any previous patches that have been released but not installed should be applied before the latest patch is installed.

For example, if the version of the currently installed A18 library is d11 and a d11c patch becomes available, the following files must be downloaded and installed in order: *A18_d11a.patch*, *A18_d11b.patch*, and finally *A18_d11c.patch*. The patched library should now be referred to as A18_d11c.

3 Process Technologies

A library contains process-specific directories for all cell files and views that have process-dependent information in them. The processes currently supported by each library are:

Jazz **A13** Library (0.12µm gate length, 1.2v core VDD)

- `CA13HC` – 6-metal RF CMOS with thick top metal (ca13)

Jazz **A13LP** Library (0.12µm gate length, 1.2v core VDD, high-density/low-power)

- `CA13HC` – 6-metal RF CMOS with thick top metal (ca13)

Jazz **A18** Library (0.18µm gate length, 1.8v core VDD)

- `CA18QW1` – 4-metal analog/mixed-signal CMOS (cg18a)
- `SBC18QB` – 4-metal SiGe BiCMOS with thick top metal (sbc18pt)
- `CA18PW1` – 5-metal analog/mixed-signal CMOS (cg18a)
- `CR18PW54` – 5-metal RF CMOS with thick top metal (ca18)
- `SBC18PT` – 5-metal SiGe BiCMOS with 2 thick top metals (sbc18pt)
- `CA18HW1` – 6-metal analog/mixed-signal CMOS (cg18a)
- `CA18HR` – 6-metal RF CMOS with thick top metal (ca18)
- `SBC18HX` – 6-metal SiGe BiCMOS with 2 thick top metals (sbc18/sbc18_techlib)

Jazz **A18LP** Library (0.18µm gate length, 1.8v core VDD, high-density/low-power)

- `SBC18QB` – 4-metal SiGe BiCMOS with thick top metal (sbc18pt)
- `SBC18PT` – 5-metal SiGe BiCMOS with 2 thick top metals (sbc18pt)
- `CA18HR` – 6-metal RF CMOS with thick top metal (ca18)
- `SBC18HX` – 6-metal SiGe BiCMOS with 2 thick top metals (sbc18/sbc18_techlib)

Jazz **A25B** Library (0.6µm gate length, 5.0v core VDD)

- `BCD25MB` – 3-metal BiCMOS/DMOS with thick top metal (bcd25)

Jazz **A25H** Library (0.36µm gate length, 3.3v core VDD)

- `CI25MW` – 3-metal imager CMOS with A18 metallization (ci25)
- `CA25QW3` – 4-metal analog/mixed-signal CMOS (ch25a)
- `CA25QLD` – 4-metal high voltage analog/mixed-signal CMOS (ch25a)
- `CH25QW` – 4-metal digital CMOS (ch25a)

Jazz **A25L** Library (0.24µm gate length, 2.5v core VDD)

- `CL25QW` – 4-metal digital CMOS (ch25a)

Jazz **A30** Library (0.36µm gate length, 3.3v core VDD)

- `SBC18MTV` – 3-metal SiGe BiCMOS with thick top metal (sbc18w)
- `SBC18MW` – 3-metal SiGe BiCMOS with thick top metal (sbc18w)
- `SBC18QB` – 4-metal SiGe BiCMOS with thick top metal (sbc18pt)
- `SBC18QTD` – 4-metal SiGe BiCMOS with thick top metal (sbc18d)
- `SBC18QW` – 4-metal SiGe BiCMOS with thick top metal (sbc18w)
- `SBC18PT` – 5-metal SiGe BiCMOS with 2 thick top metals (sbc18pt)
- `SBC18HX` – 6-metal SiGe BiCMOS with 2 thick top metals (sbc18/sbc18_techlib)

Jazz **A35** Library (0.35µm gate length, 3.3v core VDD)

- `BC35MW` – 3-metal BiCMOS with thick top metal (bc35/bc35_techlib)
- `SBC35MW` – 3-metal SiGe BiCMOS with thick top metal (sbc35)
- `SBC35MX` – 3-metal SiGe BiCMOS with thick top metal (sbc35x/sbc35x_techlib)
- `BC35QW` – 4-metal BiCMOS with thick top metal (bc35/bc35_techlib)
- `SBC35QTX` – 4-metal SiGe BiCMOS with thick top metal (sbc35x/sbc35x_techlib)

Jazz **A50** Library (0.6µm gate length, 5.0v core VDD)

- `CP05M` – 3-metal RF CMOS with thick top metal (cp05)
- `CP05MF` – 3-metal RF CMOS with thick top metal (cp05)

Jazz **A50A** Library (0.6µm gate length, 5.0v core VDD)

- `C05HMA` – 3-metal analog/mixed-signal CMOS (c05ha)

Jazz **A50H** Library (0.6µm gate length, 5.0v core VDD)

- `SBC35QTP` – 4-metal SiGe BiCMOS with thick top metal (sbc35x/sbc35x_techlib)
  this process was formerly known as SBCP35Q

The names in parentheses after the process descriptions identify the design kit (and dfII technology library if different than design kit name) associated with each process.

Process-specific files that apply to the library as a whole are generally found in the *$JAZZLIBS/$LIBNAME/process* directory. For example, the technology section of LEF files is stored in *$JAZZLIBS/$LIBNAME/process/<process name>/lef/tech.lef*.

4 Jazz Foundry Assisted COT (FAsCOT) Flow

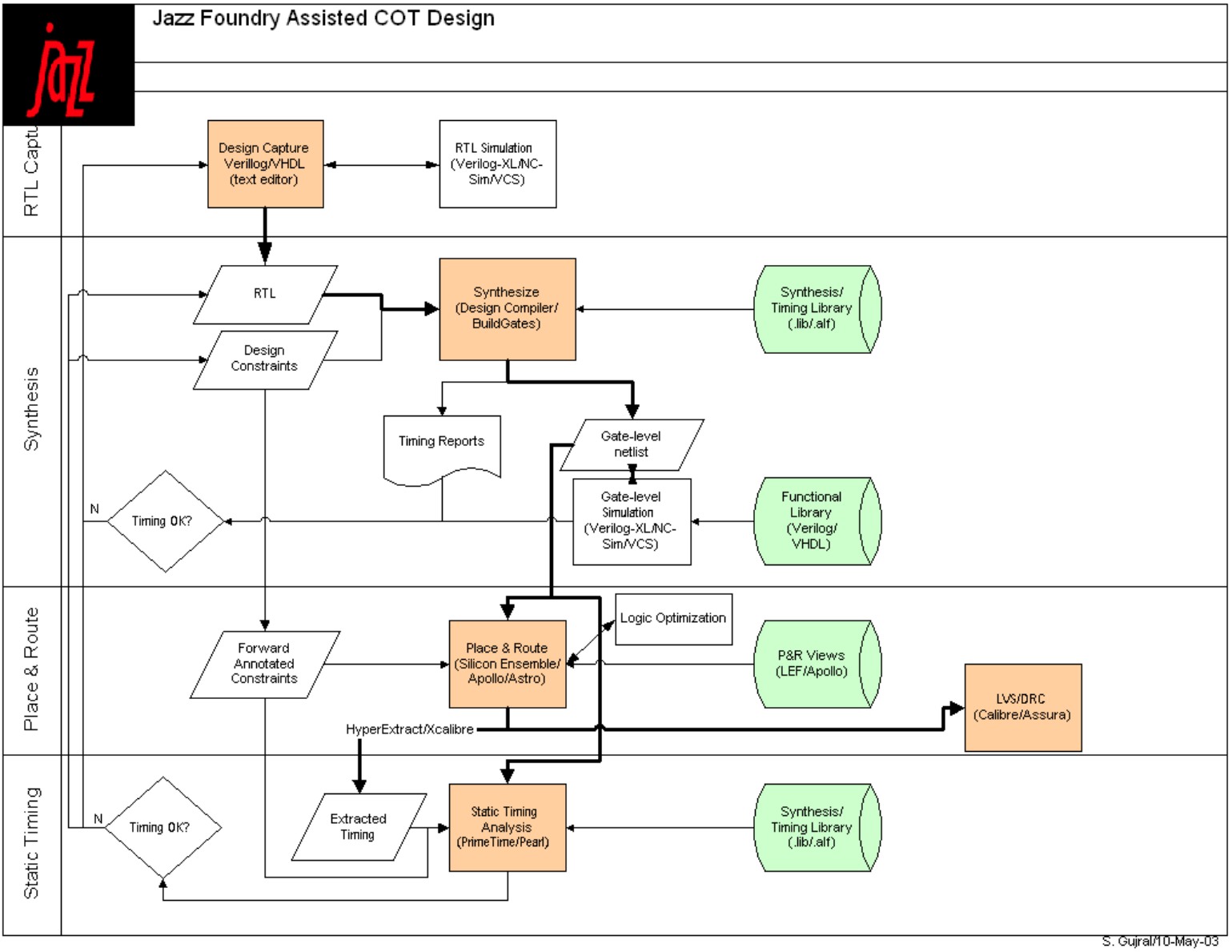

Figure 1.

5 Software Versions

Please note that all flows described in this document have been tested using the following tools and versions. This does not, however, imply that other versions may not work also.

Table 2. Supported Tools and Versions

Task	*Tool Version*
Simulation	Verilog-XL 05.10.004-s NC-Verilog 05.10-s015
Synthesis	Design Compiler V-2004.06-SP1
Static timing analysis	PrimeTime V-2004.06-1
Floorplanning	Silicon Ensemble 5.4.134 SoC Encounter v04.10-s415_1 Apollo U-2003.06-SP1 Astro V-2004.06
Cell placement (QPLACE)	
I/O placement	
Power-planning / Special route	
Routing	
Antenna fixing	
Physical verification (DRC / LVS)	Calibre v2004.04_11 Assura 3.1.4 USR2
Parasitic Extraction	HyperExtract 3.4E.29 Star-RCXT V-2004.06

6 Characterization

Simulations are done at the following process corners for slow, typical, and fast models.

Table 3. Characterization Corners

Corner	*Temp*	*Voltage*			
		A13	**A18**	**A30, A35**	**A50H**
Slow (ss)	125C	1.08V	1.65V	3.0V	2.7 / 4.5V
Typical (tt)	25C	1.20V	1.80V	3.3V	2.8 / 5.0V
Fast (ff)	-40C	1.32V	1.95V	3.6V	2.9 / 5.5V

2D nonlinear table-lookup – input slew vs. output cap

Once the library design constraints (which include the slowest input transition time and the smallest output load) are provided, the characterization tool has the ability to automatically determine the range of a cell's operation across the input transition time and output load axes. This ensures that every cell is appropriately characterized for all anticipated loads and slews. (To obtain the fastest transition time, a large buffer from the library is chosen to drive the input of every cell. This gives a measure of how fast the inputs can transition. The slowest transition at the output is an indicator of the maximum load the cell can handle.)

Since the delay is quite non-linear at the lower end of the input transition time and output load ranges, more data is sampled at these corners. At the upper end, the delay is observed to be close to linear.

Delay measurement (7x7 tables)

- Rising at 40% (vdd) → Falling at 60% (vdd)
- Rising at 40% (vdd) → Rising at 40% (vdd)
- Falling at 60% (vdd) → Rising at 40% (vdd)
- Falling at 60% (vdd) → Falling at 60% (vdd)
- Could be negative

The reason for choosing measurement points of 40% for rising and 60% for falling edges is to minimize the negative delay values which could occur due to fast switching output of some cells. Although this measurement may still result in negative timing delays, the number of such occurrences is minimized.

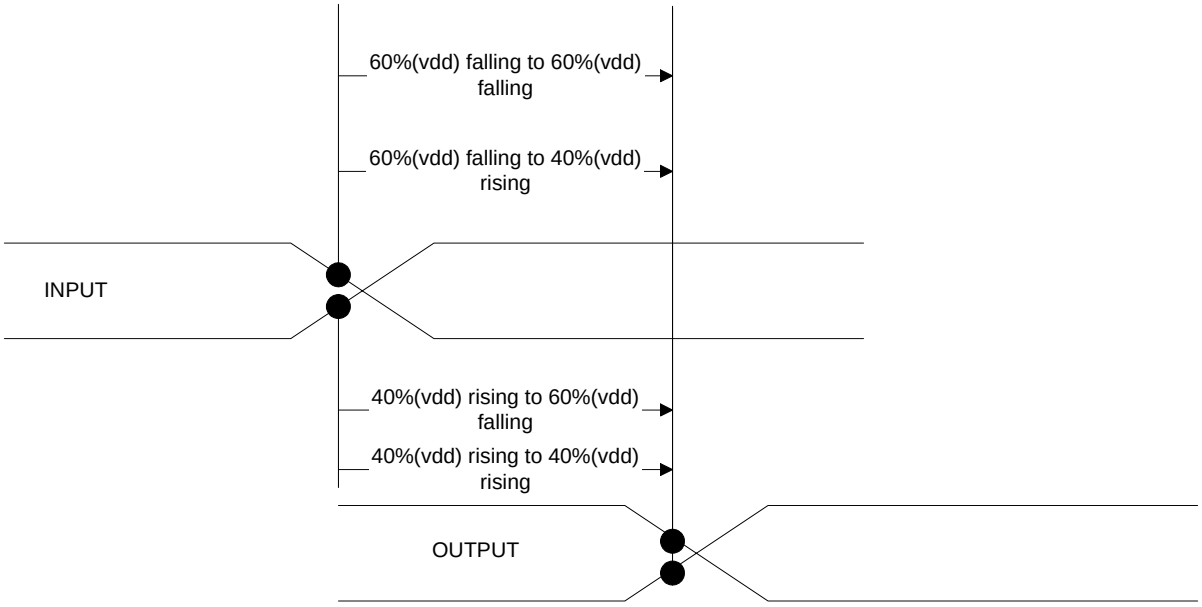

Figure 2. Delay measurement points

Transition Thresholds (7x7 tables)

- Output transition at 10% (vdd) to 90% (vdd)
- Positive, monotonically increasing

The measurement of slew from 10% to 90% of VDD captures the linear region of the output curve. Based on an S shaped waveform, the slew would become very pessimistic if the measurement was made from 0% to 100% of VDD.

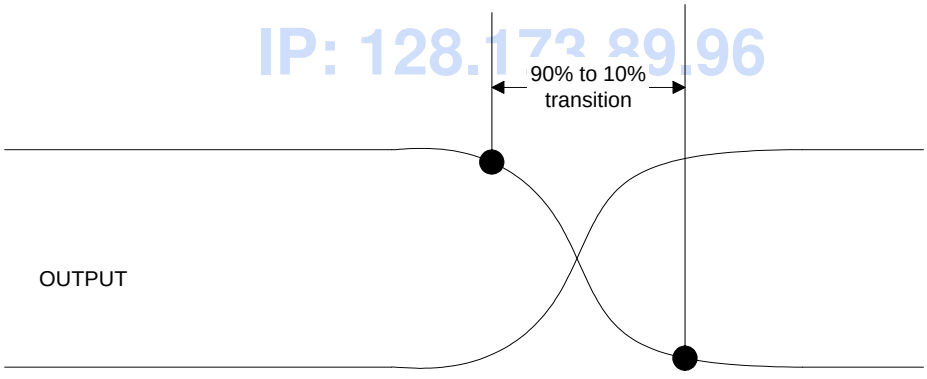

Figure 3. Transition measurement points

Constraint characterization

Constraints describe limits that guarantee predictable cell behavior. Degradation refers to the increase in delay as the input data edge is moved closer to the triggering or clock edge. A degradation parameter (dp) is used to evaluate setup/hold and recovery/removal values. If the delay is less than the target value (nominal delay × (1+dp)), the setup time

is decreased. If the delay exceeds the target value, the setup time is increased. Nominal delay refers to the delay when the data edge is far away from the triggering edge.

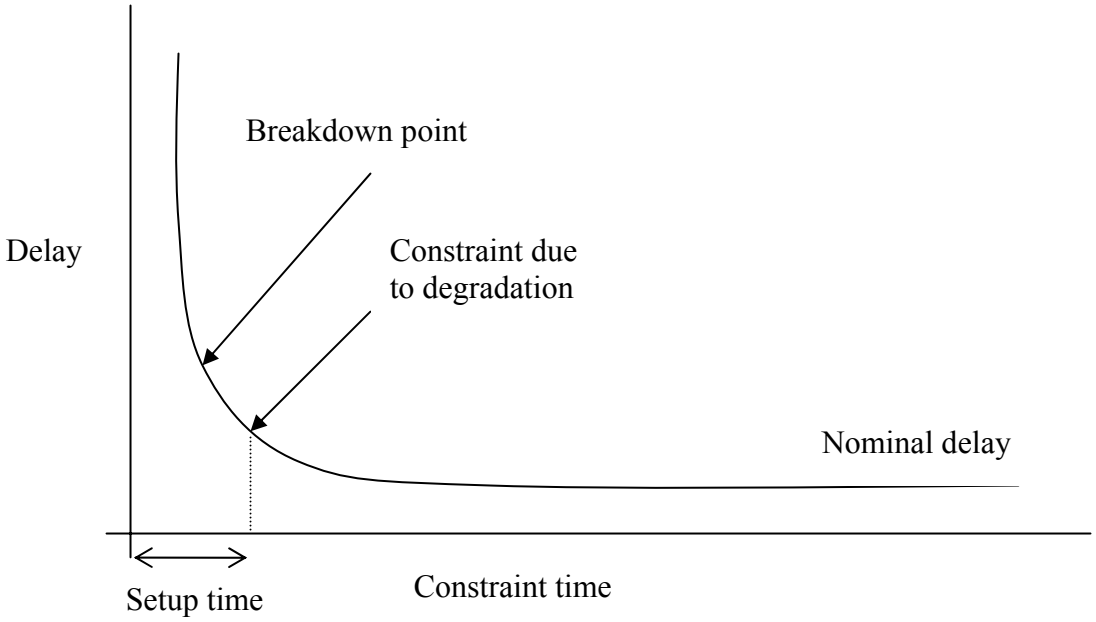

Figure 4. Relative degradation delay curve

Setup and Hold (Recovery/Removal) → (3x3 tables)
- Same as delay measurement

Recovery and removal are timing numbers of the asynchronous set and reset pins (in latches or flops) with respect to the clock edge.

Specifically, recovery refers to the time before the clock that the set/reset pin can go from active to inactive and the clock can still write valid data to the output. Removal refers to the time after the clock that the set/reset pin can go from active to inactive and the clock will not overwrite the set/reset value on the output.

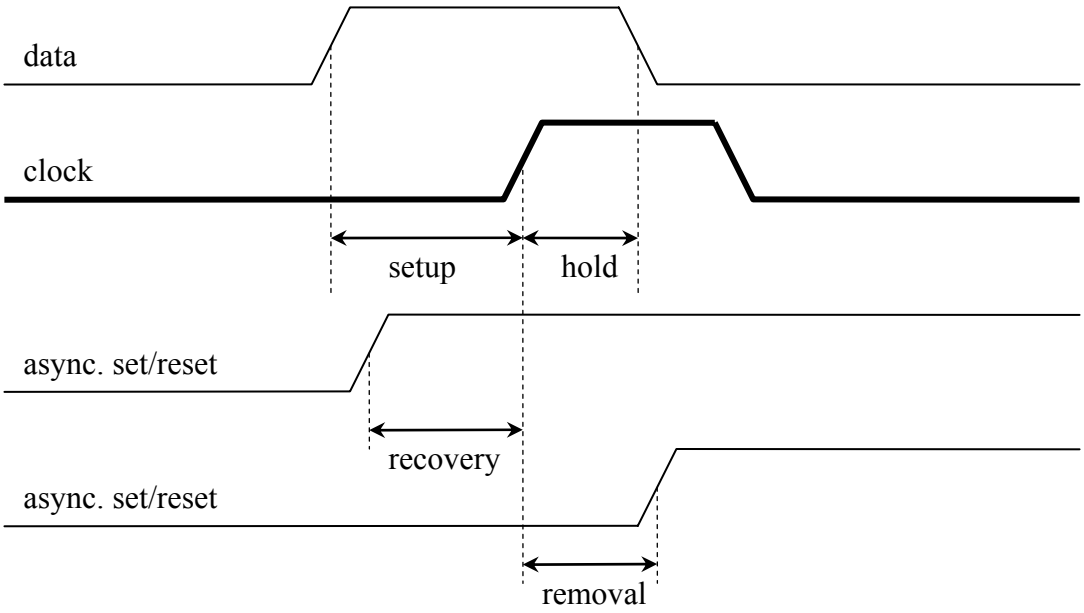

Figure 5. Timing checks

Power tables

Power modeling includes models for short circuit power and leakage power.
– 6x6 – input trans. vs. output cap.
– rise/fall power of output pins
– 1x6 – input transition
– rise/fall power of input pins
– 6x3x3 – input trans. vs. output cap. vs. like-pin output cap.
– QN load affects Q power for RS latches

7 Synthesis

The *$JAZZLIBS/$LIBNAME/synopsys/lib* and *$JAZZLIBS/$LIBNAME/synopsys/db* directories contain Synopsys Liberty (.lib) and compiled (.db) format files for all cells in the library. TLF for synthesis is being phased out, but earlier versions are available in the *$JAZZLIBS/$LIBNAME/tlf* directory. Please use the Cadence supplied syn2tlf utility to convert .lib files to TLF.

Slow, typical, and fast files can be found in the directories *$JAZZLIBS/$LIBNAME/synopsys/<lib,db>/ss*, *$JAZZLIBS/$LIBNAME/synopsys/<lib,db>/tt*, and *$JAZZLIBS/$LIBNAME/synopsys/<lib,db>/ff*, respectively.

8 Digital Simulation

The *$JAZZLIBS/$LIBNAME/verilog/src* directory contains Verilog functional models for all cells in the library, while the *$JAZZLIBS/$LIBNAME/vital/src* directory contains the corresponding VHDL models (if available).

Standard core cell models are included in the *.v (verilog) or *.vhd (vital) files, where the file names correspond to those given in Table 1 of section 2.1. These are the only files that should be used when running gate-level simulations.

Older Verilog models contain specify blocks that will annotate correctly with SDF version 2.1 generated by PrimeTime with the "-no_edge" option

```
write_sdf -no_edge <sdf_file_name>
```

to prevent edge specifications from being placed on IOPATHs. The delays and timing checks in these models have placeholder values of 1ns, and do not represent actual timing information.

Newer Verilog models make use of the combined SETUPHOLD and RECREM tasks in order to support negative timing check values, and require SDF version 3.0 generated by PrimeTime with the "-no_edge" and "-include [list SETUPHOLD RECREM]" options

```
write_sdf -include [list SETUPHOLD RECREM] -no_edge -version 3.0 <sdf_file_name>
```

9 Place and Route

The library has Place and Route views for both Silicon Ensemble/PKS and Apollo/Astro.

9.1 Cadence Silicon Ensemble Flow

The recommended methodology is to perform all tasks (floorplanning, cell & I/O placement, power-planning/special routing, antenna checking, diode insertion, detailed routing) within the SE 5.4.134 environment.

Note: Silicon Ensemble antenna checks should not be used for sign-off purposes. Always use the Jazz DRC/ANT verification decks for final tape-out.

9.1.1 Silicon Ensemble/PKS Files

The primary library inputs to SE/PKS are the LEF files. Jazz splits the LEF into technology and macro (core and I/O) files, with the latter found in *$JAZZLIBS/$LIBNAME/lef*. The technology file (*tech.lef*) used will depend on the particular process that is being designed for, and can be found in *$JAZZLIBS/$LIBNAME/process/<process name>/lef*. For example, with the SBC18HX process, the *tech.lef* that should be used is *$JAZZLIBS/$LIBNAME/process/sbc18hx/lef/tech.lef*.

The LEF files include antenna parameters and rules that are supported in SE 5.4.x but not in SE 5.3.x or earlier versions.

Timing information for the library (TLF or CTLF files) and timing constraints for the design (GCF files) must be supplied if a timing-driven flow is desired. The library timing data may be found in the *$JAZZLIBS/$LIBNAME/tlf* directory, or if not available, translated from the .lib files with the syn2tlf utility. The timing constraints can be translated from the Synopsys SDC format using BuildGates/PKS or Pearl.

An initialization file (*se.ini*) is also provided in *$JAZZLIBS/$LIBNAME/process/<process name>/lef*. This file should be linked to in the directory from which SE is started. Note that the technology LEFs specify a 2000 DBU/micron resolution, so macro files created for floorplanning and placement which refer to dimensions in DBUs must be adjusted accordingly.

9.1.2 Wroute Standalone Methodology

If desired, Wroute standalone can be used instead of the detailed routing step in SE. Before running Wroute, a DEF file with all the cells placed must be exported from Silicon Ensemble.

1. Run Wroute stand-alone
   - command used: wroute -q <cfg file>
   - see sample cfg file below
2. Read in routed DEF from Wroute back into SE

3. Add filler cells if they were not previously added in SE during placement
4. Stream out GDSII or write DEF (the variable OUTPUT.DEF.USEDVIA.ONLY should be set to FALSE so all vias are included in the DEF file)
5. Stream-in or DEF-in to Virtuoso
6. Run Calibre DRC/ANT

9.1.2.1 SAMPLE CFG FILE FOR WROUTE:

```
inputLefName "tech.lef tech_ant5.3.lef scgp.lef antenna.lef
scgp_ant.lef io8Ou5v6lm.lef"
inputDefName placed.def
routeGlobal true
routeFinal true
outputDbName wroute.wdb
antennaFullReport TRUE
frouteManufacturingGrid 10
timingDrivenRouting TRUE
outputAllVia TRUE
antennaReportName ant.rpt
inputLdefCommentChar "#"
frouteForceAntennaCell TRUE
frouteFixAntennaPass 2
frouteAntennaCellPass 2
frouteSearchRepair true
outputFullDef true
outputDefName routed.def
```


9.1.3 SE-to-dfII Methodology

To create a dfII cellview from a layout done in Silicon Ensemble, the design can be transferred in one of two ways, using LEF/DEF or using GDSII data. LEF/DEF is the preferred methodology as it preserves net names and is generally faster once a reference library has been created from LEF.

9.1.3.1 USING LEF/DEF

A new reference library must be created from the technology LEF file that was used in SE; once this reference library has been created, it can be used to import multiple DEF files. Latter versions of SE do not include all via definitions when exporting a DEF file, but can be forced to do so by setting the variable OUTPUT.DEF.USEDVIA.ONLY to FALSE.

The following steps illustrate the process of creating a library and attaching a technology file to it:

1. Select the *File→Import→LEF…* command in the CIW.
2. Enter the path to the technology LEF in “LEF File Name”.
3. Enter the name for the new library in “Target Library Name”.
4. Select “Silicon Ensemble” as the target P&R engine if not already set.
5. Click OK.

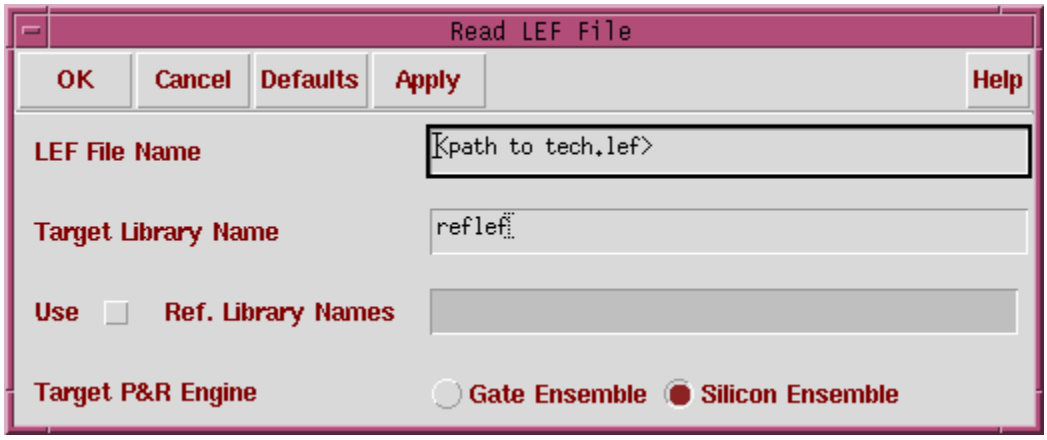

Figure 6. LEF In form

6. If version 5.0.33 or later of icfb is used, change the default layer numbers and the maskLayout units
   - dump the technology file for the newly created library using *Tools→Technology File Manager…→Dump…*
   - edit the layer numbers in the techLayers section of the technology file to be as follows (layer names may vary, and not all libraries will have 6 metal layers)

poly	5
contact	7
metal1	8
via1	17
metal2	18
via2	27
metal3	28
via3	37
metal4	38
via4	47
metal5	48
via5	57
metal6	58

   - set the maskLayout units parameter to 1000 if it has a different value
   - load the edited technology file back into the newly created library using *Tools→Technology File Manager…→Load…*

   In version 5.1.41 or later of icfb, a layer map file can be specified in the LEF-In form to accomplish the layer re-numbering.
7. Select the Technology *File→Attach To…* command in the CIW.
8. Choose the newly created library from the "Design Library" cyclic field.
9. Choose the technology library from the "Technology Library" cyclic field (see section 3 to determine the appropriate choice for each process).
10. Click OK.

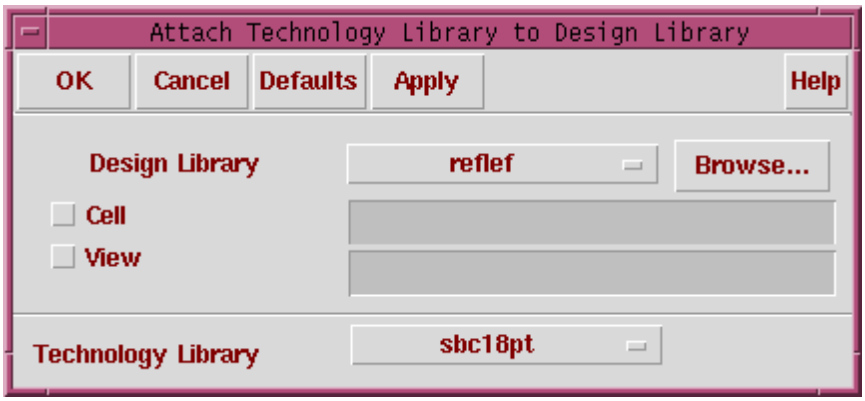

Figure 7. Technology Library attachment form

To import a DEF into a new dfII library, the following steps should be used:

1. Select *File→New→Library…* in the CIW.
2. Enter the name for the new library in "Name".
3. Select the option "Attach to an existing techfile" and specify the appropriate technology library in the subsequent pop-up window.
4. Click OK.
5. In the subsequent pop-up window, choose the technology library from the "Technology Library" cyclic field (see section 3 to determine the appropriate choice for each process).
6. Click OK.
7. Select *File→Import→DEF…* in the CIW.
8. Enter the name for the newly created library in "Library Name".
9. Enter the name for the design's top-level cell in "Cell Name".
10. Enter layout as the "View Name".
11. Enable the "Use Ref. Library Names" option and enter the name of the reference library created above, as well as the name(s) of the library(ies) containing all the leaf cells, "scgp" and "io80u5v6lm" in this example.
12. Enter the path to the DEF file in "DEF File Name".
13. Select "Silicon Ensemble" as the target P&R engine if not already set.
14. Click OK.

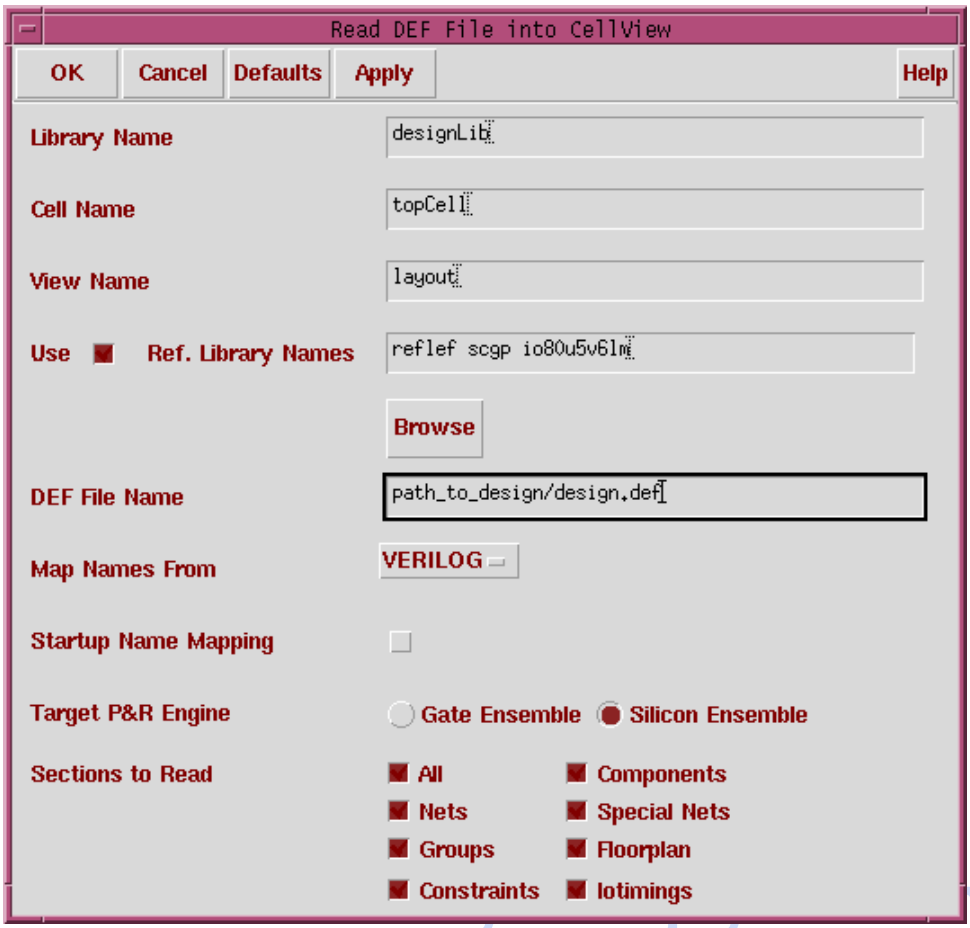

Figure 8. DEF In form

Once the design is imported, the layout view of the design will contain abstract views from the reference library(ies) for each of the leaf cells used in the design.

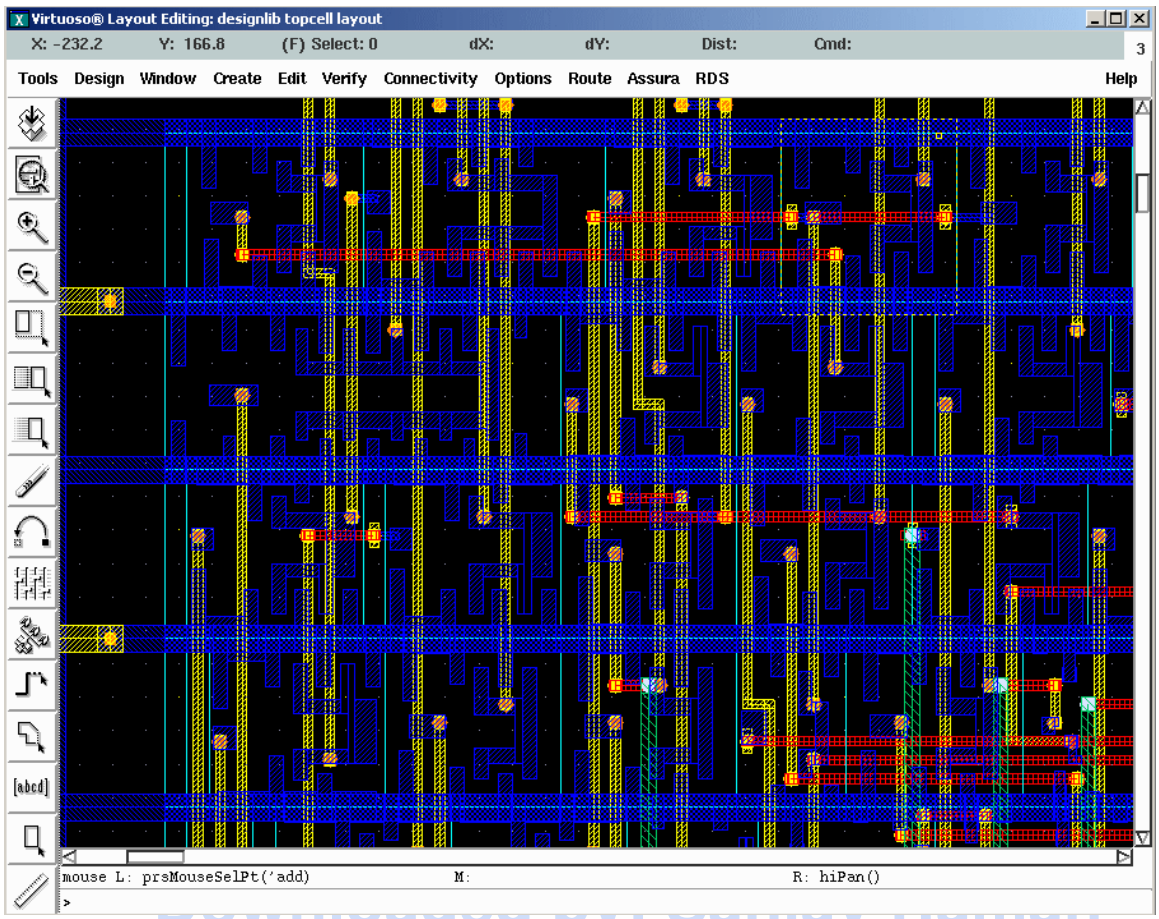

Figure 9. Imported DEF with abstract views

These views must be replaced with the corresponding layout views in order to get a complete layout. This can be done using the *Edit→Search* function in Virtuoso. **Alternatively, the changeLibView.il SKILL code can be used to change view names. This code runs significantly faster than Virtuoso's search-and-replace utility**.

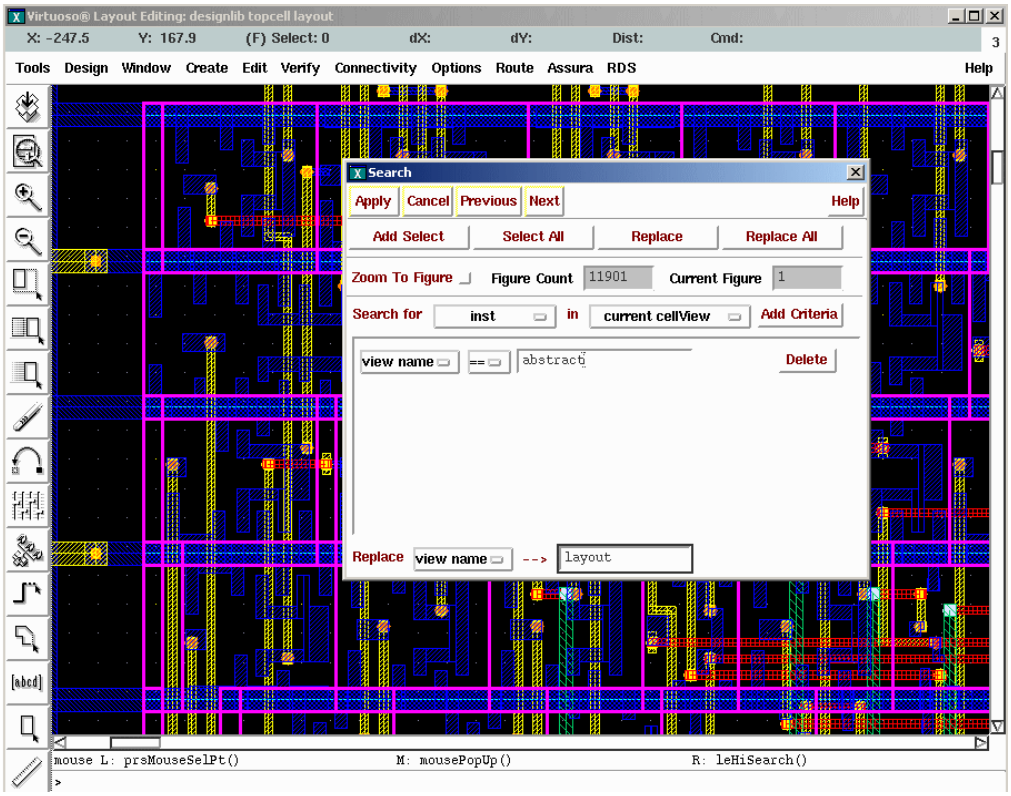

Figure 10. Virtuoso search-and-replace form

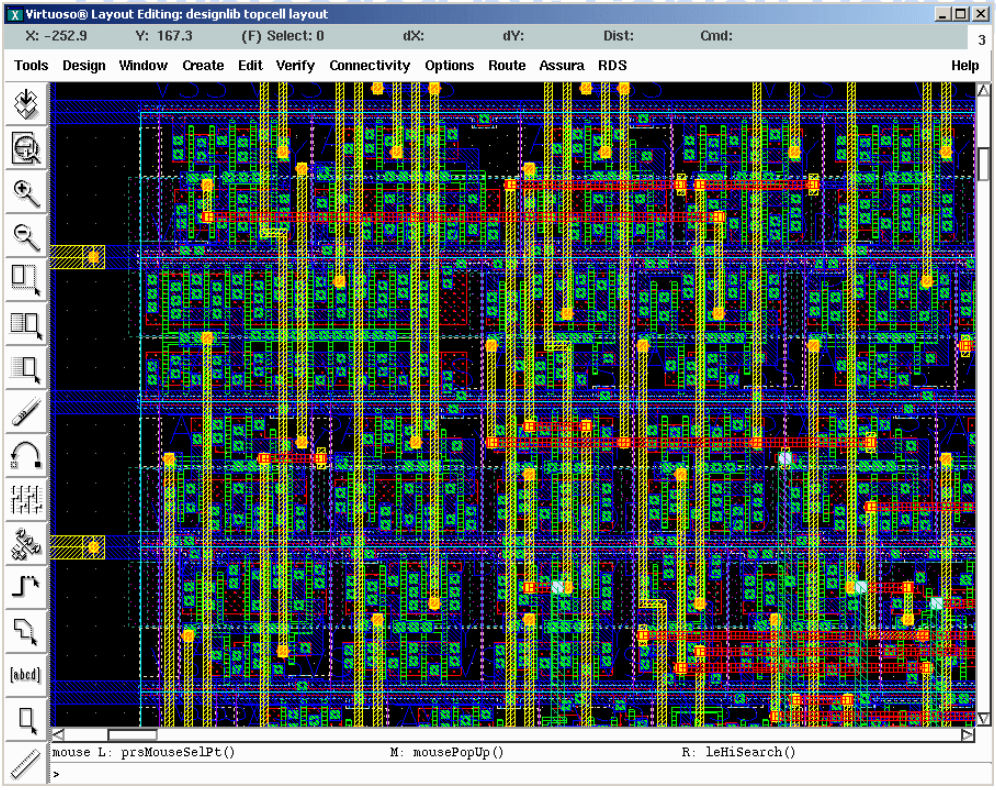

Figure 11. Design with abstract views replaced by layout views

9.1.3.2 USING GDSII

Silicon Ensemble can also write out a GDSII format file which can be read into dfII with appropriate references to the cell library. The following steps should be used to export a GDSII file:

1. Open the Export GDSII form using *File→Export→GDSII…*
2. Enter a name for the GDSII file.
3. Enter the path to the *gds2.map* file associated with the process being used (located in *$JAZZLIBS/$LIBNAME/process/<process name>/lef*).
4. Set "Units" to Thousands.
5. Click on "Variables" and set the following variables:
   GDSII.OUTPUT.ATTACHED.INSTANCENAMES TRUE
   GDSII.OUTPUT.IOPINNAMES TRUE
6. Click OK on both forms.

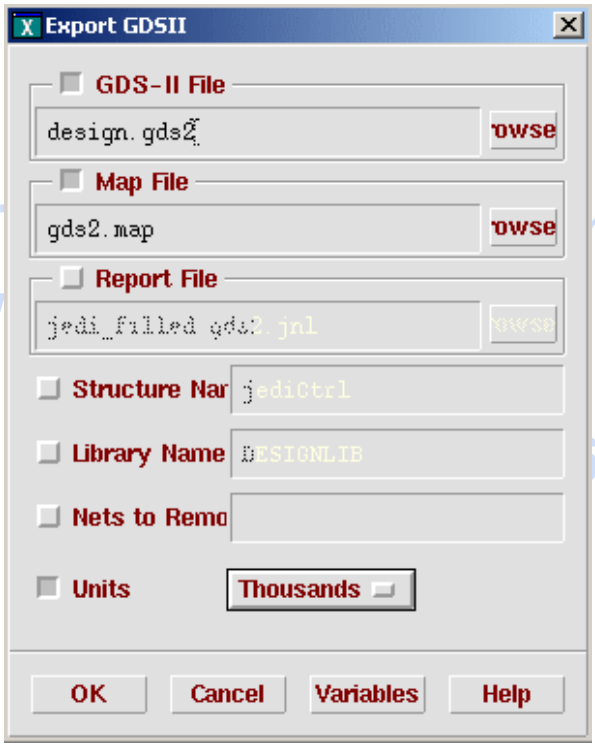

Figure 12. Export GDSII form

The GDSII file can then be imported into a dfII library using the following steps:

1. Open the Virtuoso Stream In form using *File→Import→Stream…* in the CIW.
2. Enter the path to the GDSII in "Input File".
3. Enter the top-level cell name in "Top Cell Name".
4. Enter the destination library in "Library Name".
5. Click on "Options" and check the "Retain Reference Library (No Merge)" option.
6. Check the "preserve" option for "Case Sensitivity".

7. Specify the "Reference Library Order" on the same form. This field can contain multiple dfII library names. Usually, these are the names of the Jazz-supplied standard cell and I/O libraries.
8. Click OK on both forms.

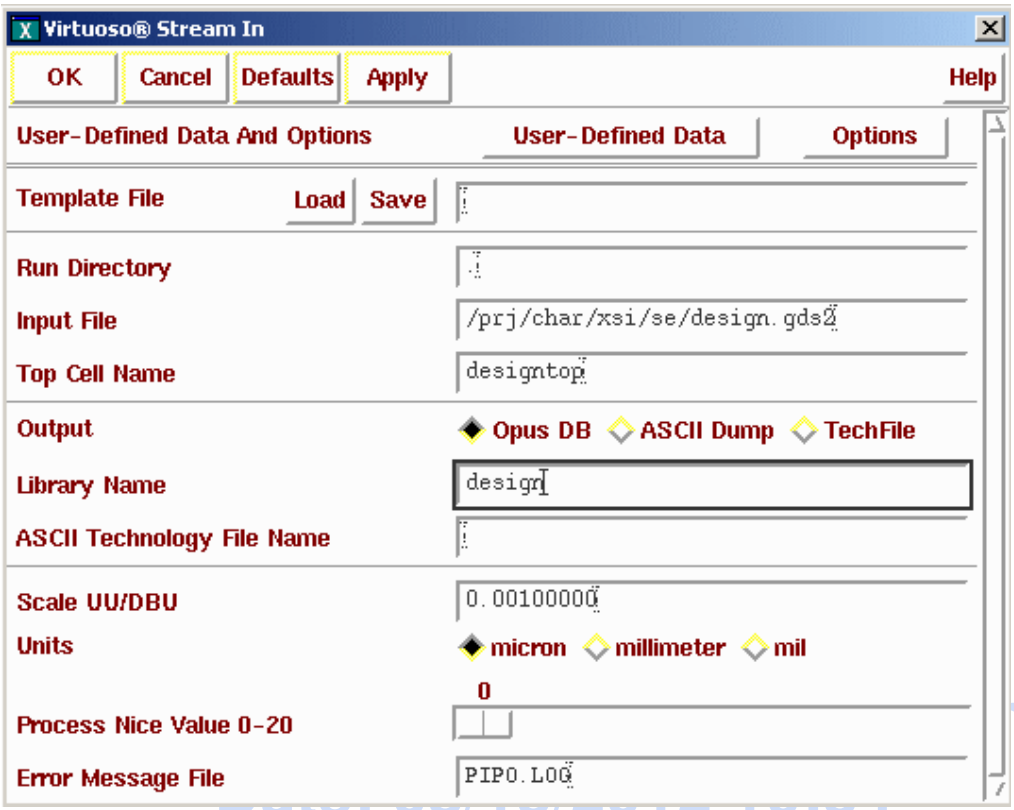

Figure 13. Stream In form

9.2 Apollo Place and Route Flow

This section identifies the files and reference libraries needed to import a design into Apollo. It also covers the steps to export the design out of Apollo and import it into dfII.

Note: Antenna violations reported by Apollo should not be used for sign off purposes. Always use the Jazz DRC/ANT verification decks for final tape-out.

9.2.1 Apollo Reference Libraries and Technology Files

The inputs into Apollo P&R are the reference libraries (standard cell and I/O) and technology files. The libraries include FRAM, CEL, NETL, PWR, and TIM views, and can be found in the directories *$JAZZLIBS/$LIBNAME/apollo/<cell_lib>* for standard cells or *$JAZZLIBS/$LIBNAME/apollo/<process name>/<cell_lib>* for I/Os. The technology files can be found in *$JAZZLIBS/$LIBNAME/process/<process name>/apollo*, with file names of *<cell_lib>.tf*.

The FRAM views include antenna parameters for the cells, while the rules defining antenna violations are specified in a Scheme command file that must be loaded prior to performing any antenna checking or fixing operations. This command file can be found in *$JAZZLIBS/$LIBNAME/process/<process name>/apollo*, with file name *antenna.rul*.

Timing information for the library is contained in the TIM views. These and a Synopsys SDC file must be provided if a timing-driven flow is desired.

9.2.2 GDSII out of Apollo

To create a dfII cellview for a design, the data can be transferred in GDSII format. GDSII is the recommended methodology for getting into dfII.

Export a GDSII file from Apollo using the following steps:

1. Invoke the auStreamOut utility in Apollo (select *Tools→Data Prep*, then *Output→Stream Out …*).
2. Enter a name for the GDSII file in "Stream File Name".
3. Enter the library name where the design is stored in "Library Name".
4. Set the "Child Extraction Depth" field to 20 (or any other number that will encompass all the physical hierarchy levels in the design).
5. Select "Specified Cell" for the "Convert" option.
6. Enter the top-level cell name to be streamed out in "Cell Name".
7. Enable the "Convert Reference Lib Child Cells" option.
8. Select "FILL" for the "Fill" option; requires the existence of a .FILL cell for the design, created using the geNewFillNG utility (*Route Utility→Fill Notch/Gap …*).
9. Select "Force output outdated Fill Cell" for the "Fill" option if the Apollo database has been edited after the .FILL data was generated, but the editing is known to have had no effect on the fill.
10. Click on the "Pin/Net Options" button.
11. Select both "As Text" and "As Geometry" for the "Output Pins" option.
12. Click "OK" in the "Stream Out Data File" form.

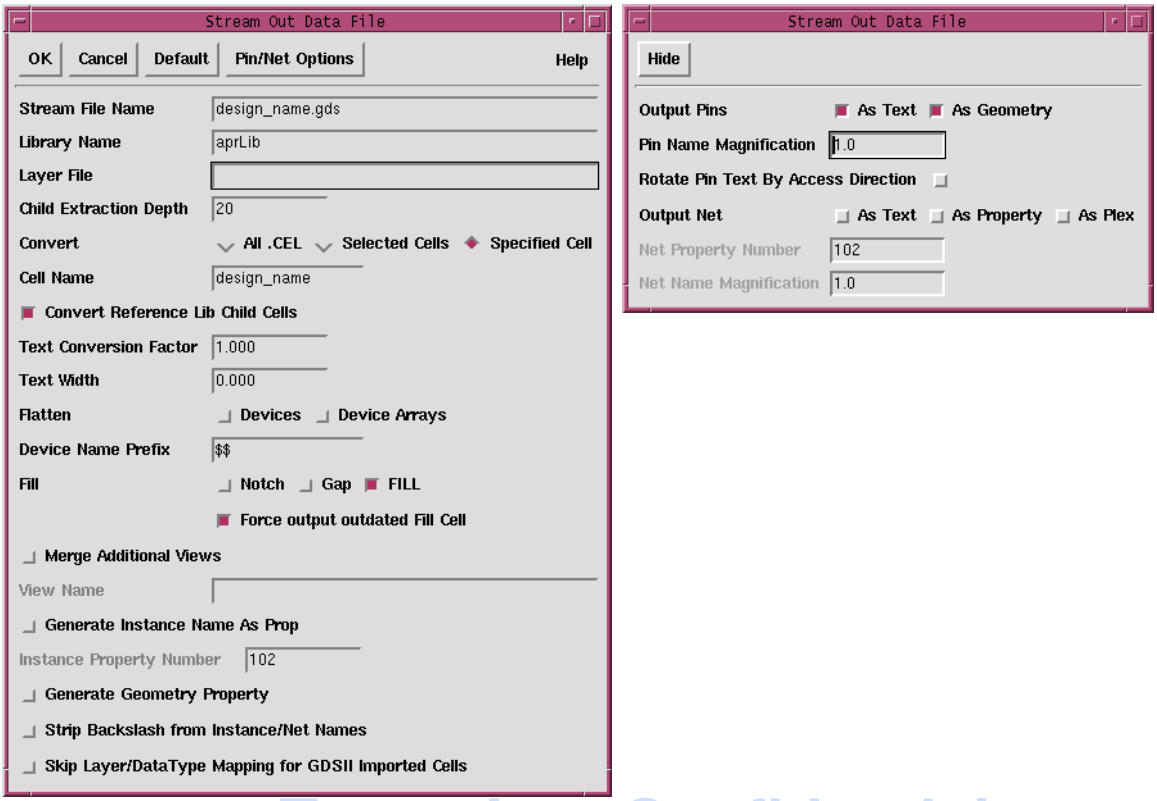

Figure 14. Stream Out Data File form

9.2.3 GDSII into dfII

The same flow described in section 9.1.3.2 can be used for stream in. Use the same form and all the same options.

10 Parasitic Extraction

Parasitic extraction is supported via HyperExtract in Silicon Ensemble, Fire & Ice QXC in SoC Encounter (may not be available on all processes), and Star-RCXT in Apollo/Astro. The extraction rules files are provided for each process variation in the following directories:

HyperExtract - *$JAZZLIBS/$LIBNAME/process/<process name>/he*
Fire & Ice QXC – *$JAZZLIBS/$LIBNAME/process/<process name>/soc*
Star-RCXT – *$JAZZLIBS/$LIBNAME/process/<process name>/star*

Raw process layer information for building your own parasitic extraction tables is available from Jazz Customer Solutions.

11 Verilog Import

Occasionally, it might be necessary to read the Verilog netlist into the Cadence dfII environment to create a schematic view. The Verilog Import utility can be used to accomplish this task.

11.1 Methodology

1. Set the SKILL variable `importHdlPlaceLabelOnAllSegs=t` in the CIW. This fixes some problems when reading in large netlists using ihdl.
2. Select *File→Import→Verilog* in the CIW.
3. Fill out the Verilog In form with the following information:
    a. **Target Library Name**: name of the dfII library that contains the layout for the design to be read in.
    b. **Reference Libraries**: list of dfII libraries that contain the leaf cells used in the design.
    c. **Verilog Files To Import**: path to the structural Verilog netlist of the design.
    d. **–v Options**: path(s) to the Verilog library(ies) for the cells used in the design, e.g., *$JAZZLIBS/$LIBNAME/verilog/src/scgp.v*. Multiple libraries are specified using a space to separate between them.
    e. **Import Structural Modules As**: set as "schematic" (cyclical field).
    f. **Overwrite Existing Views**: checked if a schematic view for this design already exists in the library.
    g. **Power Net Name:** specify inh_PVDD to connect tie-high nets to the inherited connection ports added by the netlister. Note that Verilog In automatically adds "!"s to the end of the power net names. These are removed by the CBR netlister used later to generate the netlist for LVS.
    h. **Ground Net Name:** specify inh_NVSS to connect tie-low nets to the inherited connection ports added by the netlister. Note that Verilog In automatically adds "!"s to the end of the ground net names. These are removed by the CBR netlister used later to generate the netlist for LVS.
4. Click OK.
5. Schematic and symbol cellviews will now be added to the library in a cell with the same name as the top-most Verilog module in the imported netlist. If the netlist is hierarchical, then multiple schematics/symbols will be added to the library reflecting the hierarchy.
6. If LVS is to be run from the *Calibre* menu, replace all inh_PVDD! and inh_NVSS! net names in all levels of hierarchy with inh_PVDD and inh_NVSS (stripping the "!" from the names).
7. If LVS is to be run on the stand-alone digital block and there are any inh_PVDD! or inh_NVSS! nets at the top level, replace these with the appropriate power and ground net names (by default VDD and VSS when using the *Jazz→Verification* menu, VDD! and VSS! when using the *Calibre* menu).

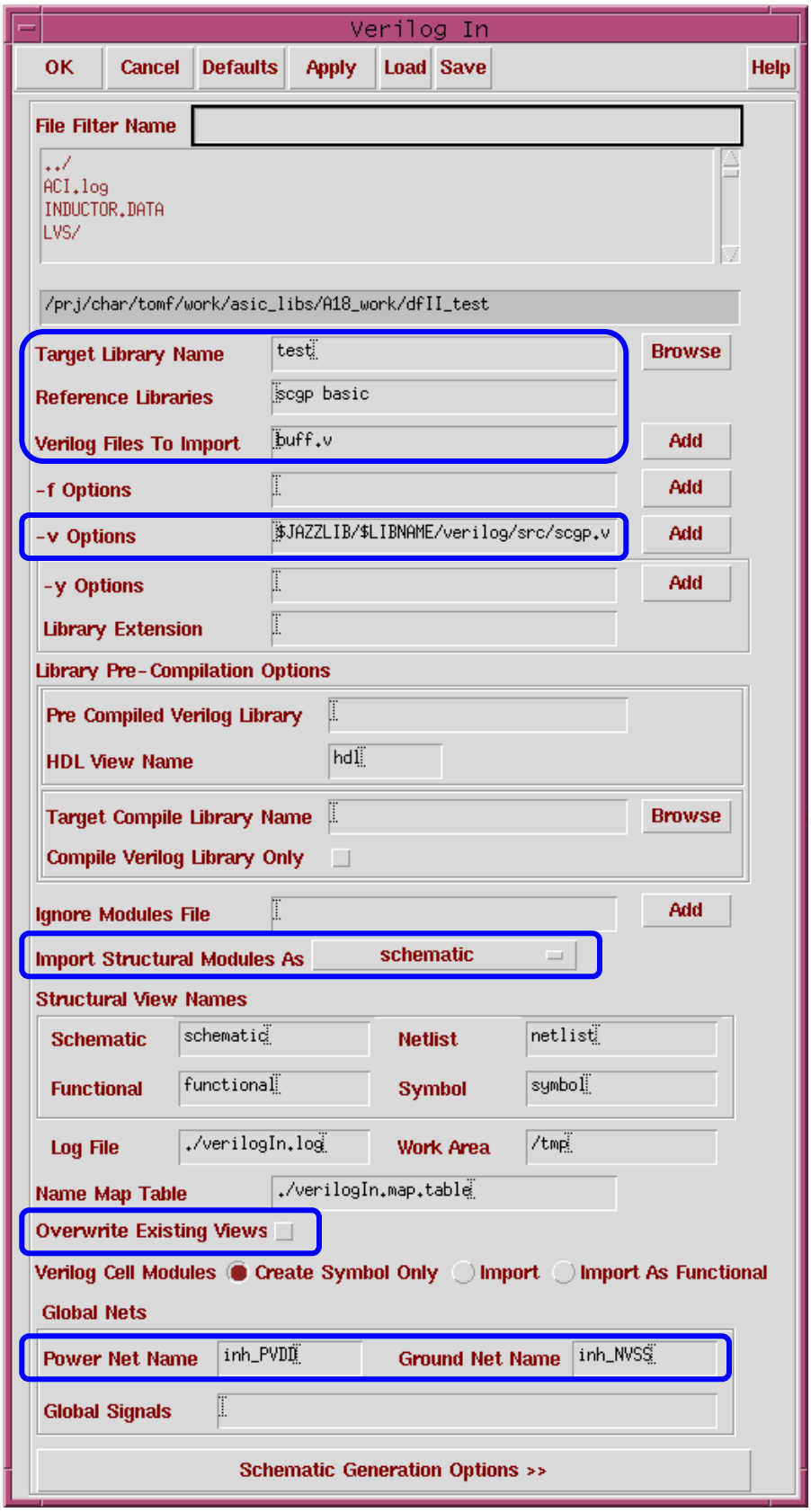

Figure 15. Verilog In form

11.2 Inherited Connections

Cadence's inherited connections flow specifies the supply nets at the level of hierarchy where different supply signals must be differentiated. If there are distinct analog and digital supplies, the properties that set the digital power and ground nets are placed on instances at the highest level of hierarchy where both analog and digital supplies co-exist. Below this level of hierarchy there is no need to set any supply signal names.

Cadence provides functionality to determine what inherited expressions are available, using the following steps:

1. Open a schematic.
2. Set the view list to include a view with inherited terminals (spectre for instance) by choosing *Options→Editor*, then putting spectre at the front of the "View Name List".
3. Choose *Edit→Net Expressions→Available Properties*, then select an instance. The form will show all the netSet property names that are available to be set, as well as the current values of those properties. Clicking on a primitive will show its available properties and their currently evaluated values. Clicking on a non-leaf cell when all lower-level cells already have netSets will not display any properties.
4. Choose *Edit→Net Expressions→Evaluated Names* to see all the names/values in a hierarchy and to show the hierarchy tree where these values are set.

If a design relies on inherited connections for power and ground, the steps outlined below should be followed to import a netlist with the correct connectivity.

Using the following Verilog structural netlist as an example:

```
// Verilog HDL for "test", "buff" "verilog"

module buff_sub (in, out);
input in;
output out;
wire n1;
 and2x1 U1 ( .A(in), .B(1'b1), .Y(n1) );
 or2x1 U2 ( .A(n1), .B(1'b0), .Y(out) );
endmodule

module buff (IN, OUT);
input IN;
output OUT;
  buff_sub U1 ( .in(IN), .out(OUT) );
endmodule
```

1. Select *File→Import→Verilog…* to create the schematic view for cell "buff" as described in the previous section.
2. Create a "buff_top" schematic which instantiates cell "buff", and has "buff_pwr" and "buff_gnd" pins.

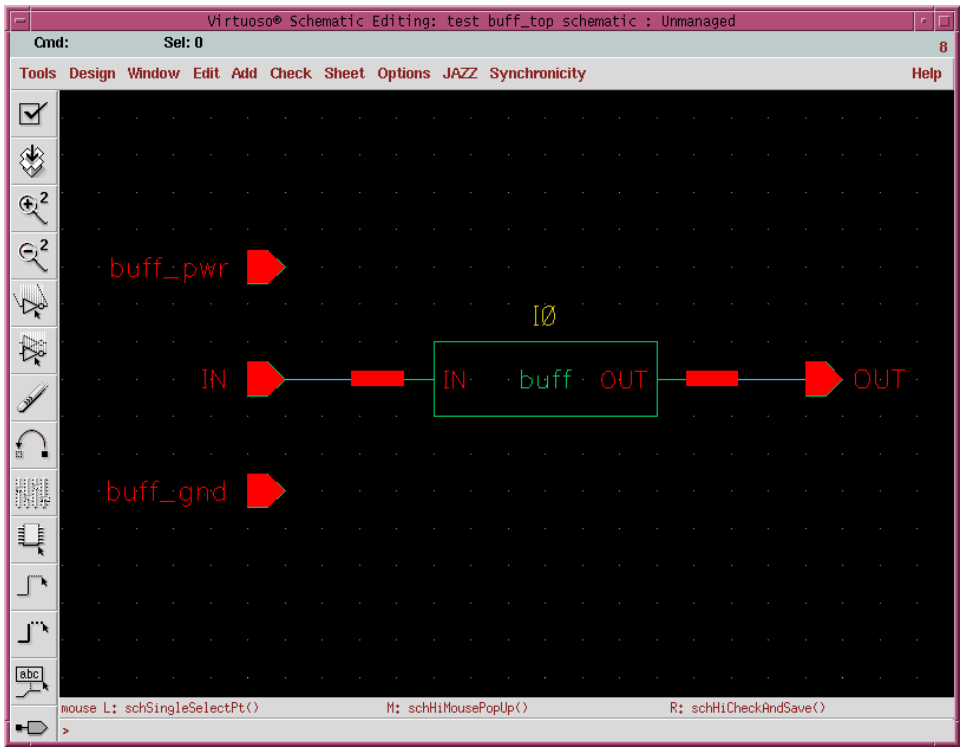

Figure 16. Schematic for cell "buff_top"

3. Edit the properties of the "buff" instance and click on the "Add" button.

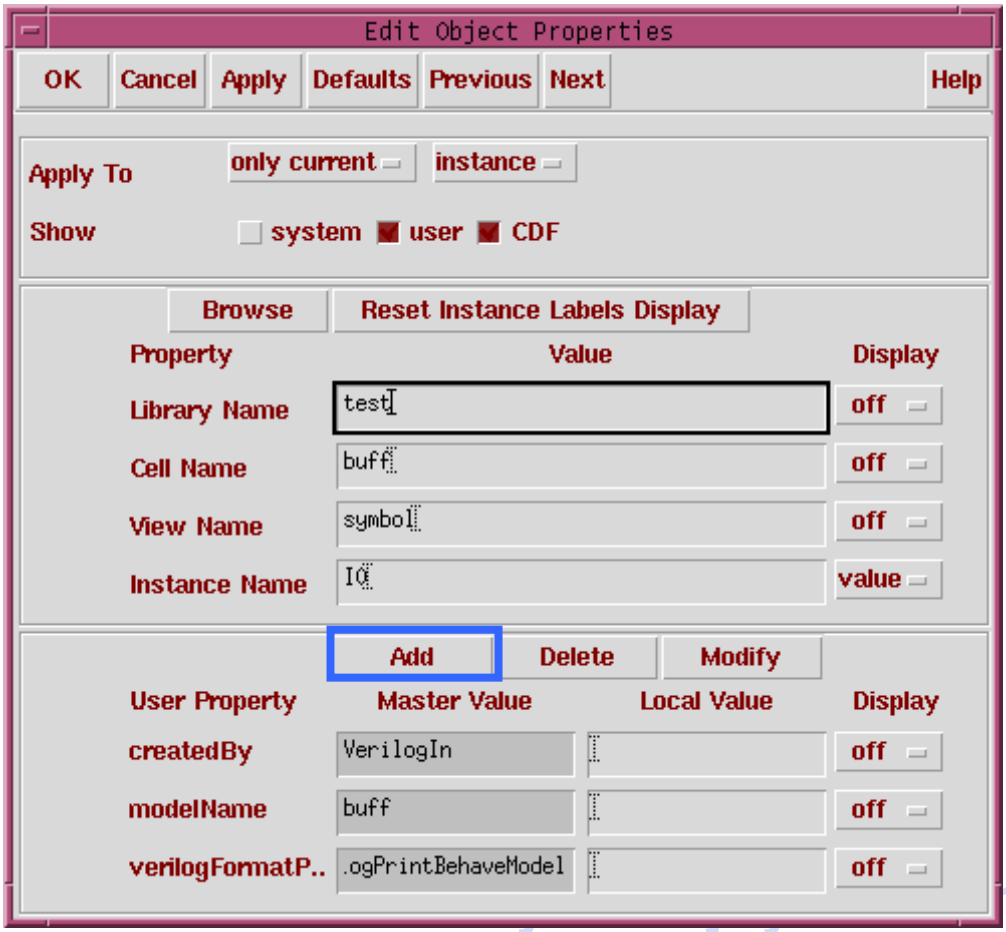

Figure 17. Schematic object properties form

4. Add the required netSet properties to specify that the inherited power and ground connections for all cells under the “buff” hierarchy should be tied to the “buff_pwr” and “buff_gnd” pins (cells with a separate substrate connection would also need an NSUB property).

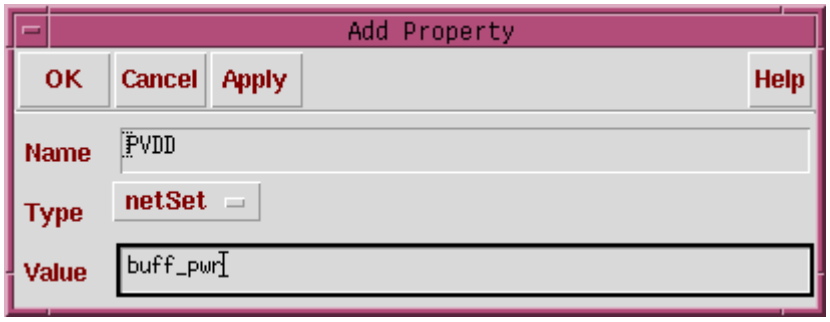

Figure 18. Inherited power connection property

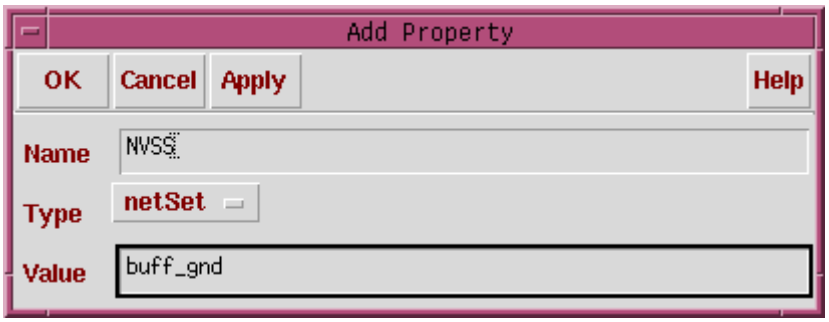

Figure 19. Inherited ground connection property

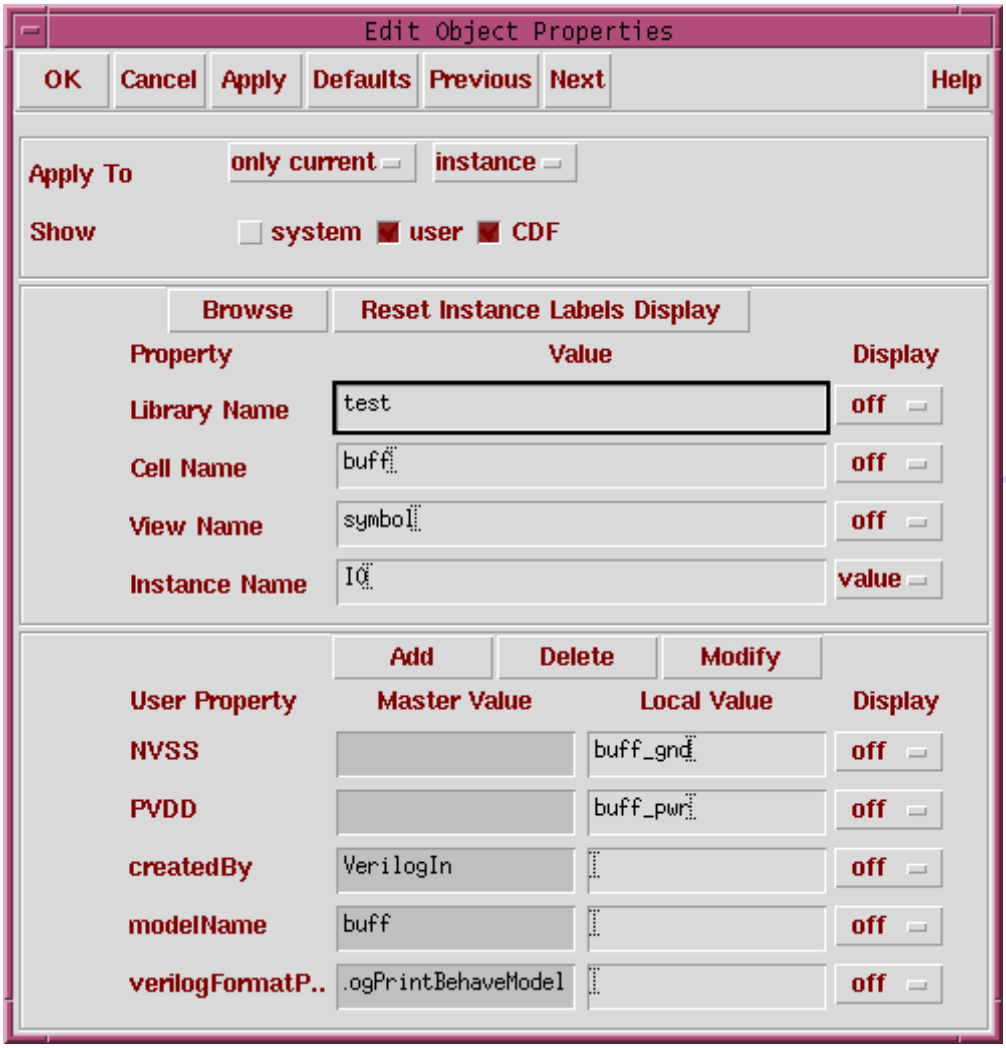

Figure 20. Schematic object properties after adding netSet properties

5. Select *JAZZ→Netlist Utilities→CBR for Calibre with Include files* to generate the LVS netlist. This step is optional, as the netlist can be generated automatically when Calibre LVS is run.

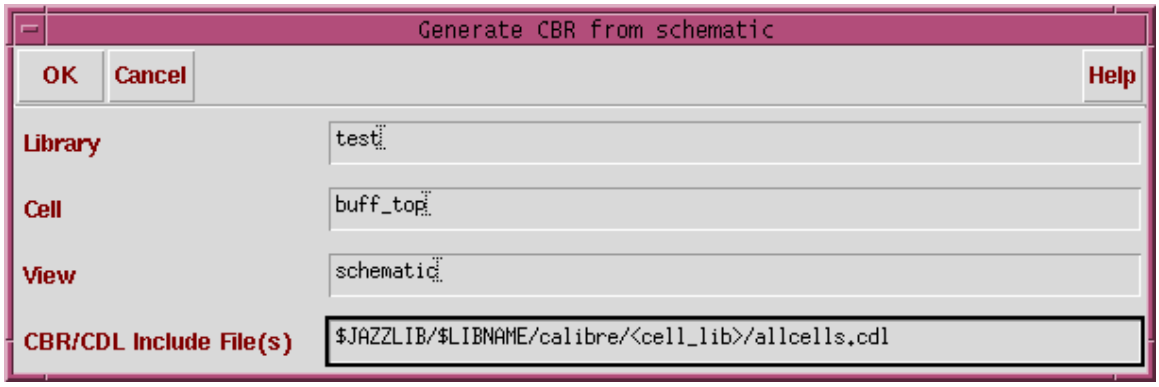

Figure 21. CBR netlister form

The following CBR netlist is generated with the desired power and ground nets.

```
* RSS DFII to Calibre netlister output for buff_top
* Date of buff_top netlist generation: 2004/08/26_10:52:45
* initialization file
/rds/prod/HOTCODE/amslibs/cds_default/cdsware/bin/cbrsbc18.ini and
./cbr.ini used
* CBR Netlister Include
.include "/prj/char/tomf/work/asic_libs/A18/calibre/scgp/allcells.cdl"
...
...
.SUBCKT buff_sub_1 in inh_NVSS inh_PVDD out
XU2 n1 inh_NVSS inh_PVDD inh_NVSS out or2x1
XU1 in inh_PVDD inh_PVDD inh_NVSS n1 and2x1
.ENDS buff_sub_1
.SUBCKT buff IN OUT inh_NVSS inh_PVDD
XU1 IN inh_NVSS inh_PVDD OUT buff_sub_1
.ENDS buff
.SUBCKT buff_top IN OUT buff_gnd buff_pwr
XI0 IN OUT VSS VDD buff
.ENDS buff_top
```


For I/O libraries there will typically be multiple power and ground nets that will need to be connected through inherited connections, as shown in the following table.

Table 4. Inherited connection properties for I/O libraries

Net Name	*Inherited Connection Property*
VDD	P
VDDO	POUT
VDDP	PPAD
VESD	PESD
VGG	PGG
VSEQ	PSEQ
VSS	N
VSSO	NOUT

12 Mixed-signal Simulation

The library contains dfII views that can be used to enable both transistor-level and mixed-signal simulations from inside the Cadence dfII environment. Step-by-step methodologies using Spectre or HSPICE are presented below.

The dfII libraries include the following views:

- abstract
- auCdl
- hspiceS
- layout
- spectre
- spectreS
- symbol
- verilog

Note that no schematic view is provided. Full spice-level netlists, however, are provided in the hspice and spectre directories. Also note that licensing terms for Jazz libraries do not allow end-user modifications of the cells.

HSPICE & Spectre subcircuits are available in the following directories:

- *$JAZZLIBS/$LIBNAME/hspice/<cell_lib>*
- *$JAZZLIBS/$LIBNAME/hspice/<process name>/<cell_lib>* (for I/Os)
- *$JAZZLIBS/$LIBNAME/spectre/<cell_lib>*
- *$JAZZLIBS/$LIBNAME/spectre/<process name>/<cell_lib>* (for I/Os)

Use spectre, hspice, or symbol views in Composer. Power and ground nets are provided through inherited connections.

12.1 Transistor-level Methodology

1. Import the Verilog netlist(s) as described in section 11

With the spectre simulator option in the Analog Design Environment:

2. Select the *Setup→Model Libraries …* menu item
3. Add one or both of the following files to the list of model libraries:
   $JAZZLIBS/$LIBNAME/spectre/<cell_lib>/allcells.scs
   $JAZZLIBS/$LIBNAME/spectre/<process name>/<cell_lib>/allcells.scs (for I/Os)
4. Run the simulation

With the spectreS simulator option in the Analog Design Environment:

2. Create a file (e.g., stdcell_includes.scs) with one or both of the following lines in it:
   include "*$JAZZLIBS/$LIBNAME/spectre/<cell_lib>/allcells.scs*"
   include "*$JAZZLIBS/$LIBNAME/spectre/<process name>/<cell_lib>/allcells.scs*"
3. Select the *Setup→Environment* menu item in the simulation window
4. Enter the path to the file created in step (2) into the "Stimulus File" field

5. Set the “Include/Stimulus File Syntax” option to “spectre” (do not leave as “cdsSpice”)
6. Run the simulation

With the hspiceS simulator option in the Analog Design Environment:

2. Create a file (e.g., stdcell_includes.cir) with one or both of the following lines in it:
   .include “*$JAZZLIBS/$LIBNAME/hspice/<cell_lib>/allcells.cir*”
   .include “*$JAZZLIBS/$LIBNAME/hspice/<process name>/<cell_lib>/allcells.cir*”
3. Select the *Setup→Environment* menu item in the simulation window
4. Enter the path to the file created in step (2) into the “Stimulus File” field
5. Set the “Include/Stimulus File Syntax” option to “hspice” (do not leave as “cdsSpice”)
6. Run the simulation

12.2 Mixed-signal Methodology

1. Import the Verilog netlist(s) as described in section 11
2. Select *File→New→Cellview…* in the CIW to create a configuration view for the top-level cell in the simulation using the Hierarchy Editor

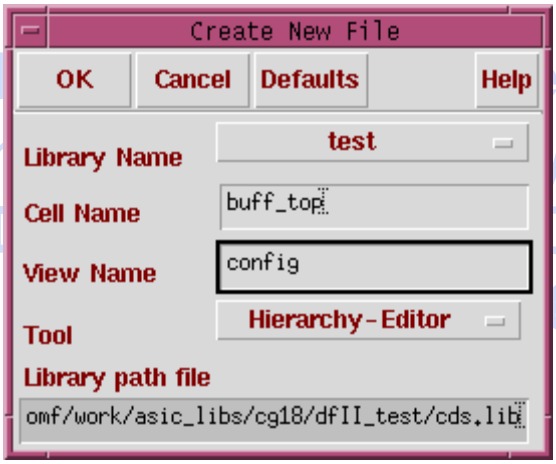

Figure 22. Creating a configuration view

3. Enter “schematic” in the “View” field, “schematic verilog spectre” in the “View List” field, and “verilog spectre” in the “Stop List” field (if using the spectreS or hspiceS simulator, replace “spectre” in the above fields with the corresponding view name)

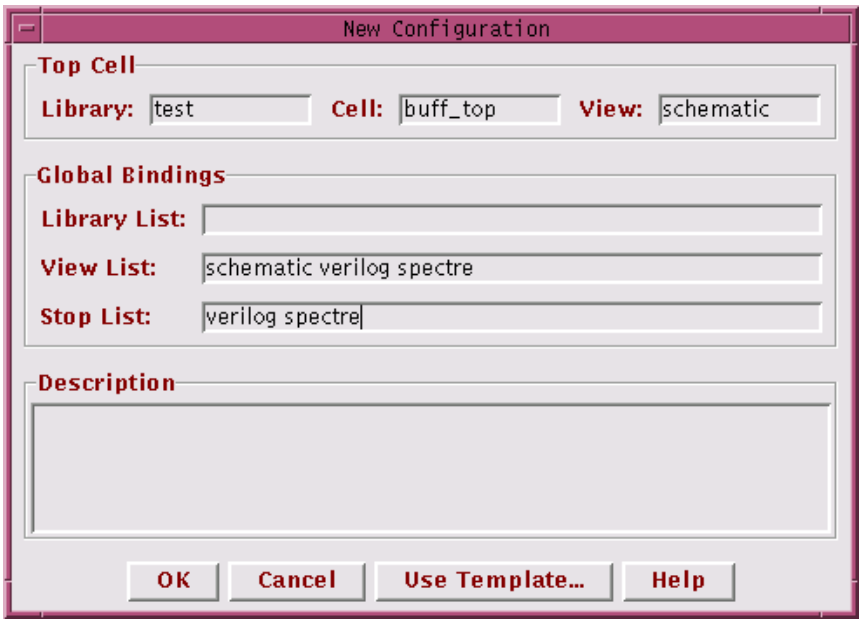

Figure 23. Configuration view parameters

4. Inspect the cell bindings to ensure that the appropriate views will be used during the simulation, then save the configuration view

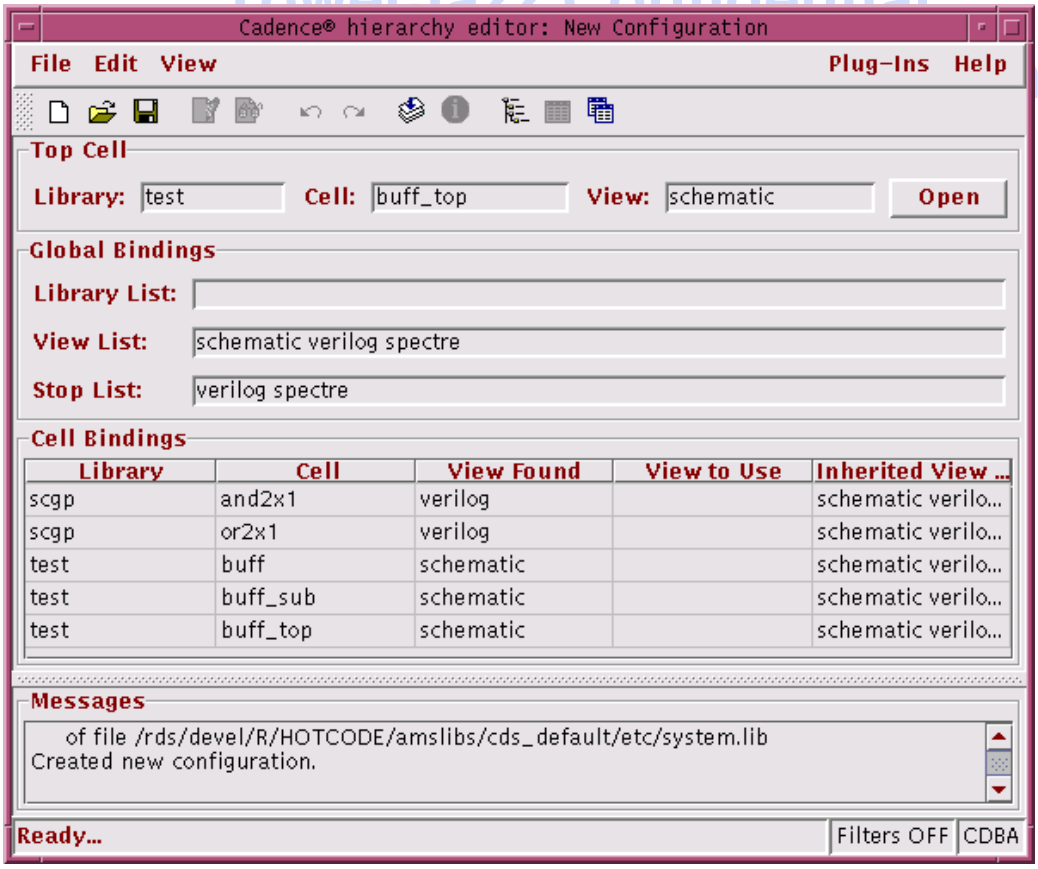

Figure 24. Configuration view

5. Open the schematic to be simulated by clicking the "Open" button in the configuration view (or selecting "yes" for the schematic view when opening an existing configuration view)
6. Select *Tools→Analog Environment* to bring up the Analog Design Environment window
7. Select *Setup→Simulator/Directory/Host …* and change the "Simulator" option to "spectreVerilog" (or "spectreSVerilog or hspiceSVerilog); note that opening the simulation schematic directly will result in an error message being generated at this step
8. Select *Setup→Environment …* and click the "Verilog Netlist Option …" button to check the "Netlist Explicitly" option; it must be enabled to avoid port order discrepancies between cell instances and their module definitions

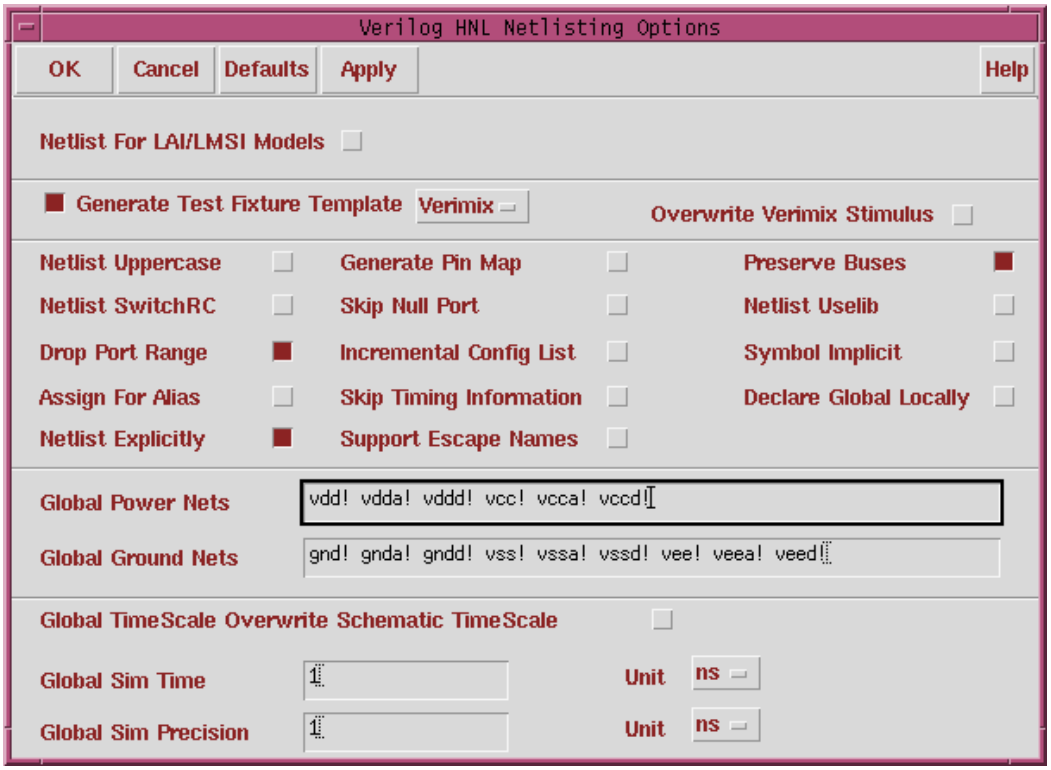

Figure 25. Verilog netlisting options

9. Select *Simulation→Options→Digital …* and enter the path to the Verilog models in the "Library Files" field (see section 8)
10. Run the simulation

13 Mixed-Signal LVS Using Calibre (post-Feb. 2003 Kits)

Note: This methodology works with Design Kits dated after February 18, 2003.
See section 14 titled* Mixed-Signal LVS Using Calibre (pre-Feb. 2003 Kits) *for using older design kits.

13.1 Verilog-In Methodology

LVS can be run directly from the Virtuoso Layout Editor using either the "JAZZ" or "Calibre" custom pull-down menus. A schematic view of the design must already exist in dfII. See section 11 for information on how to read in a Verilog netlist.

13.1.1 Using the JAZZ Menu

On the Virtuoso Layout Editor window select *Jazz→Verification→Calibre LVS→Run LVS...*

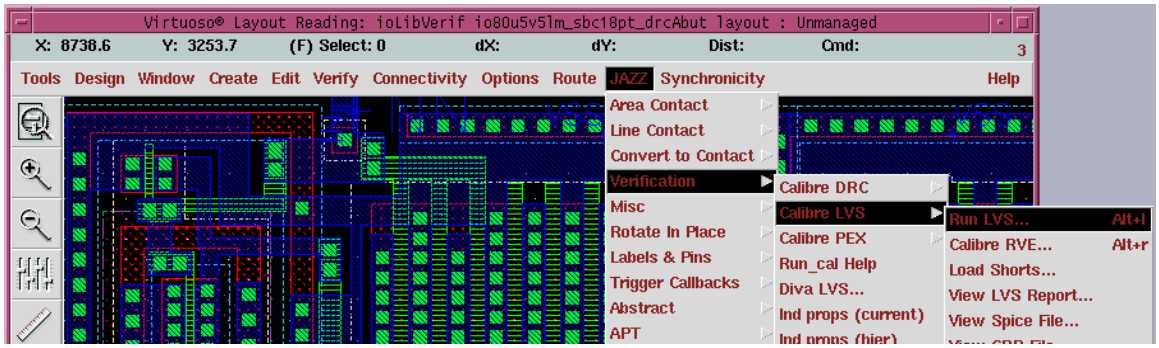

Figure 26. Running Calibre LVS from the JAZZ menu

On the Calibre LVS form (see below), only the "Source Includes (Opt.)" field should need to be filled with the path(s) to the *allcells.cdl* file(s), which can be found in *$JAZZLIBS/$LIBNAME/calibre/<cell_lib>* and/or *$JAZZLIBS/$LIBNAME/calibre/<process name>/<cell_lib>*.

After clicking on "OK", a cbr netlist is written to *<project dir>/verification/cbr_dir*. The exact location can be seen in the CIW. Inherited connections, if used, will be netlisted correctly by the netlister. This cbr netlist is compared to the created GDS file by Calibre LVS.

Since Calibre LVS is case-insensitive by default, case sensitivity may need to be enabled if the input Verilog netlist was case-sensitive, which is usually the case. To enable case sensitivity in Calibre LVS, a new file should be created with the following two lines in it:

SOURCE CASE YES
LAYOUT CASE YES

and the file name added to the Calibre form in the *RuleFile include(Opt.)* text field.

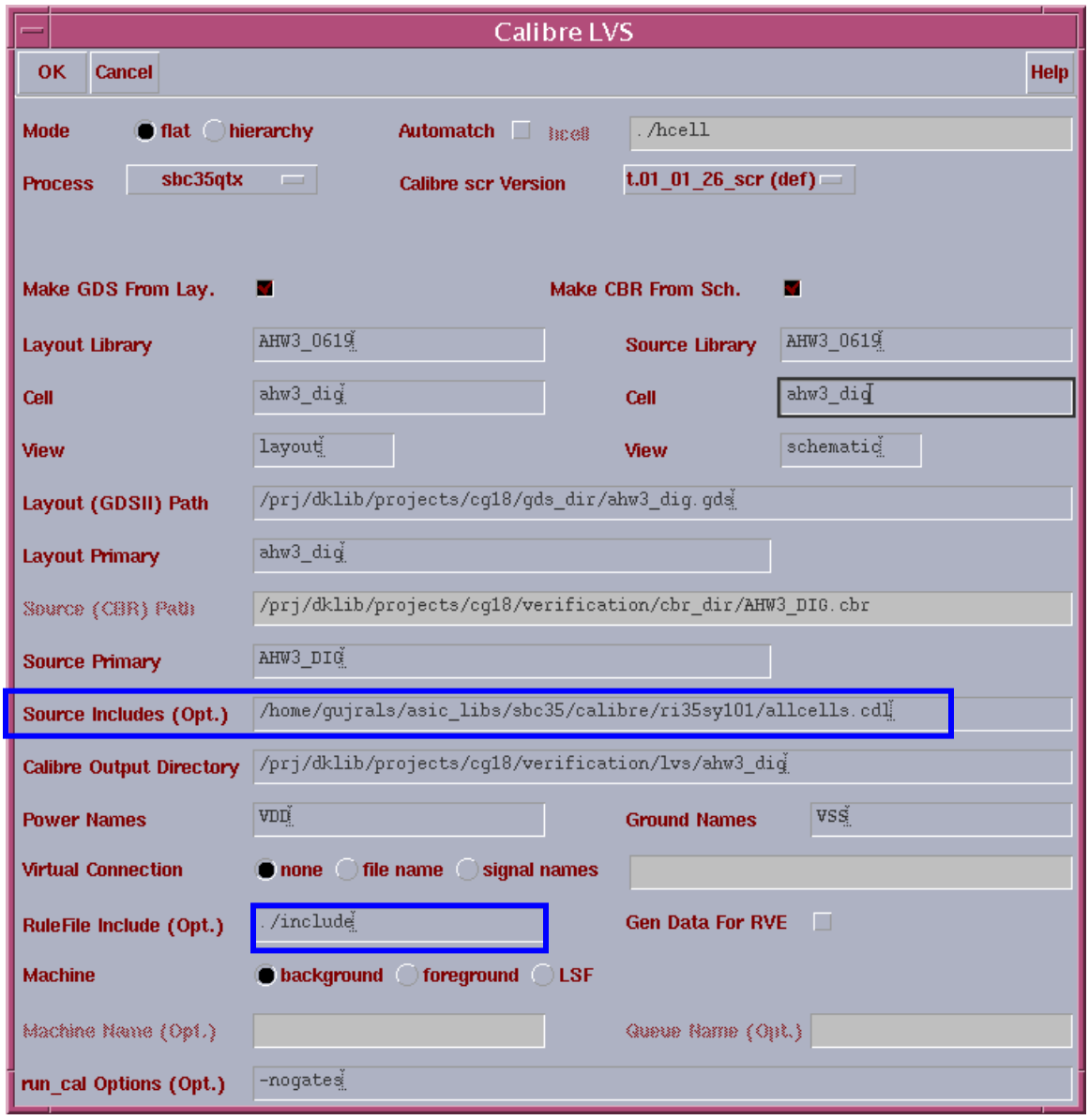

Figure 27. Calibre LVS form

13.1.2 Using the Calibre Menu

On the Virtuoso Layout Editor window select *Calibre→Setup→Netlist Export…*

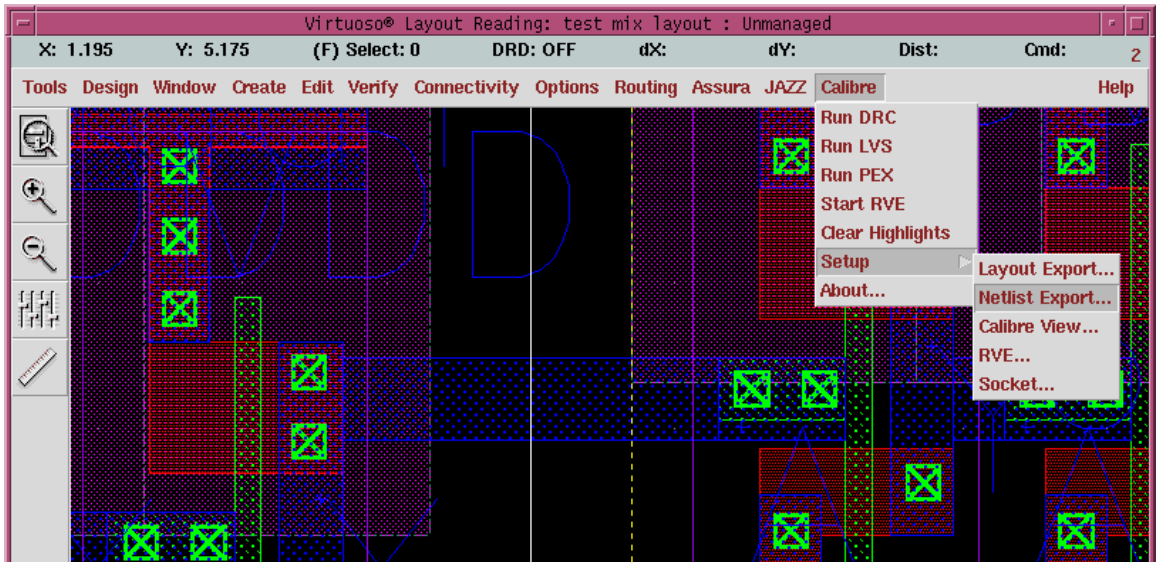

Figure 28. Invoking Calibre netlist export setup

and specify the path(s) to the *allcells.cdl* file(s) in the "Include File" field.

Figure 29. Calibre netlist export setup form

The Calibre Interactive tool can now be invoked by selecting *Calibre→Run LVS* on the Virtuoso Layout Editor window.

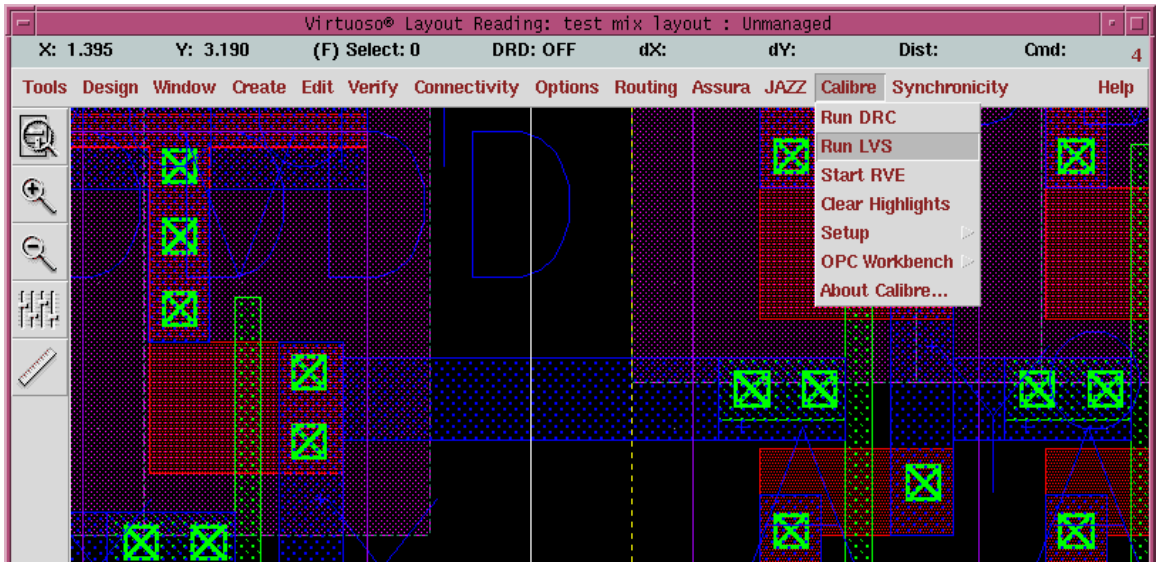

Figure 30. Running Calibre LVS from the Calibre menu

Case sensitivity can be enabled as described in the previous section. To specify the include file in the Calibre Interactive GUI, select *Setup→LVS Options*

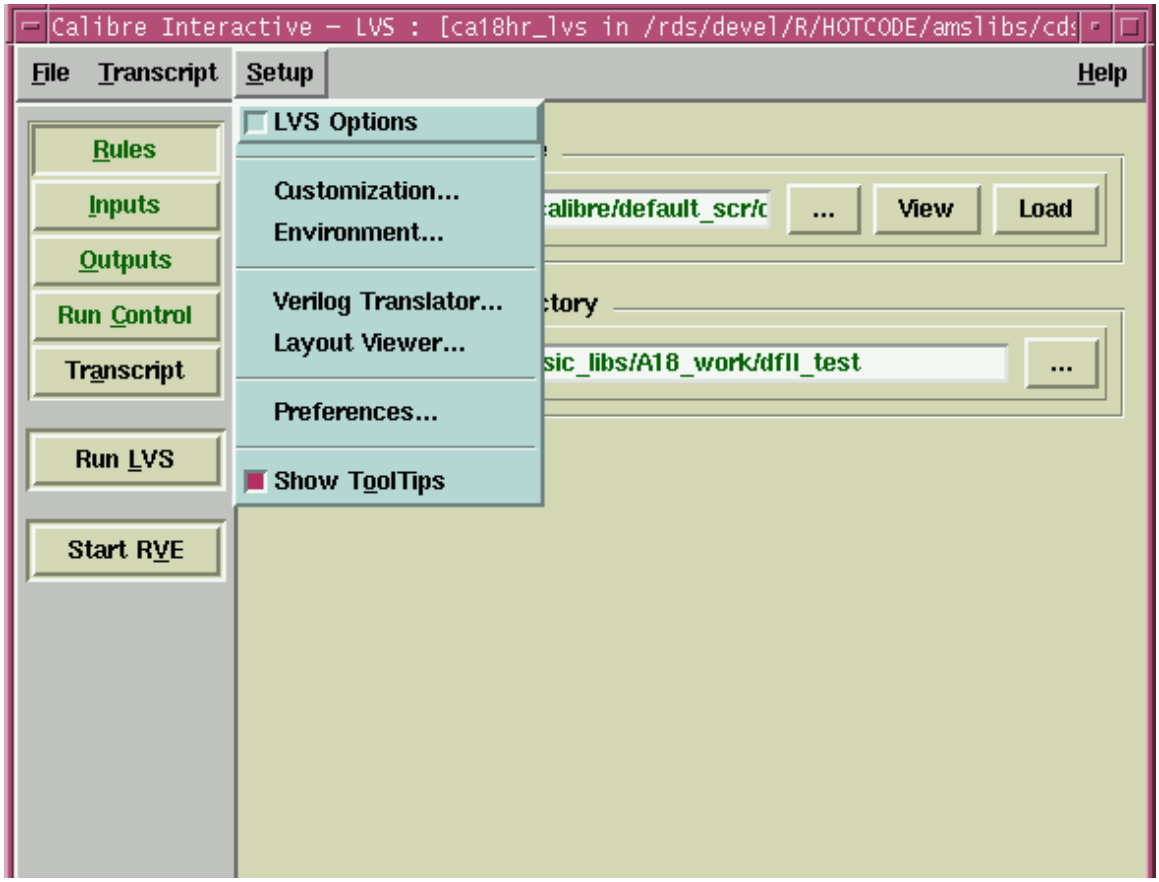

Figure 31. Calibre Interactive LVS form

and go to the “Includes” tab in the new “LVS Options” window pane.

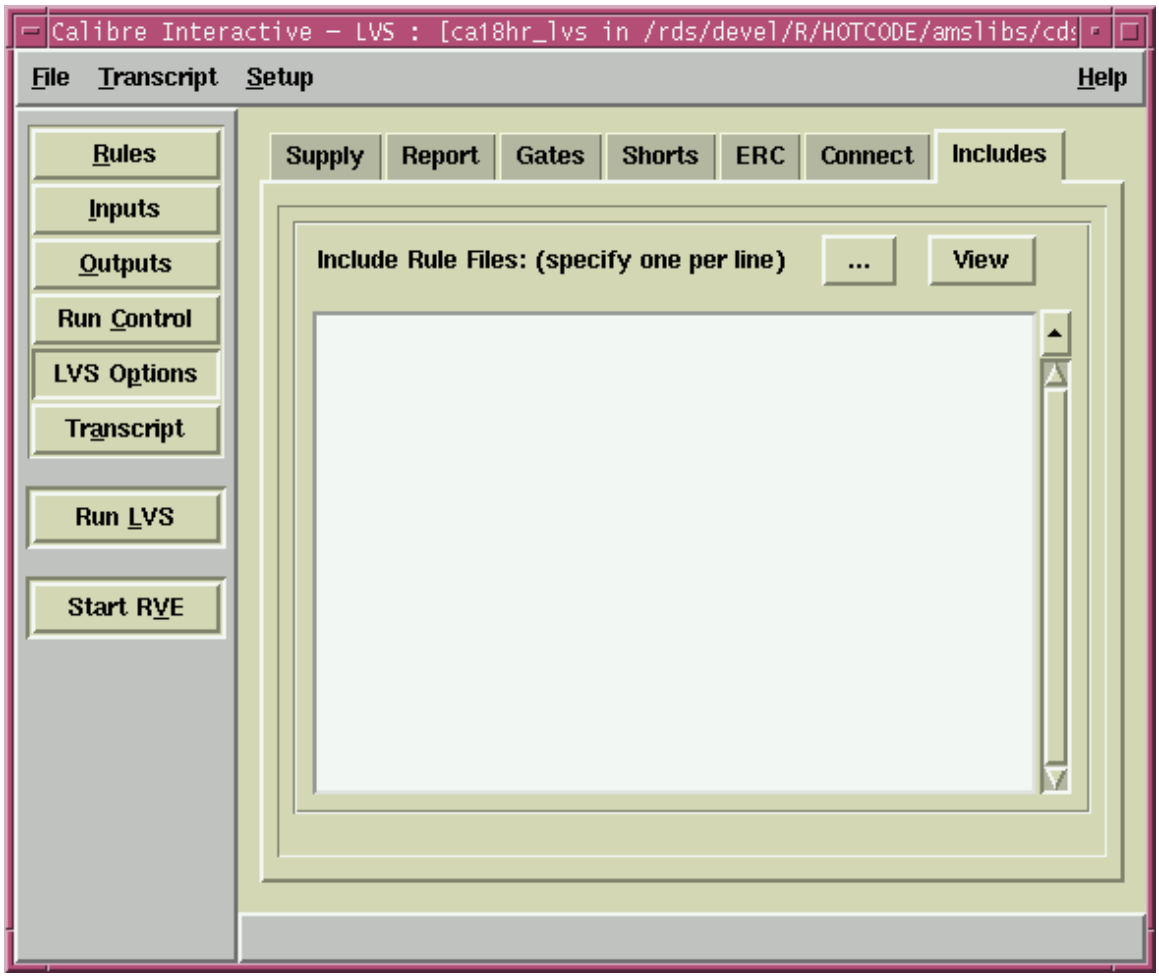

Figure 32. Calibre Interactive LVS Options

13.2 v2lvs Methodology

An alternative approach to creating the LVS netlist for the digital portion of a design is to export a Verilog netlist from the P&R tool and use the v2lvs translator.

From the Silicon Ensemble GUI select *File→Export→Verilog…* to export a post-routed netlist. The “Output Power & Ground Ports” option should be enabled.

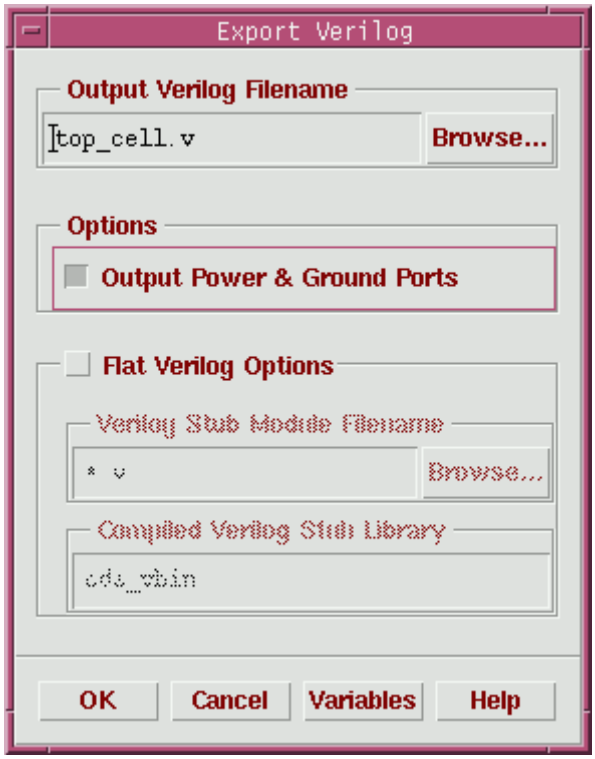

Figure 33. Silicon Ensemble Export Verilog form

From the Apollo GUI select *Tools→Data Prep* and then *Output→Verilog Out …* to export a post-routed netlist. Refer to the figure below to see which options need to be enabled or disabled.

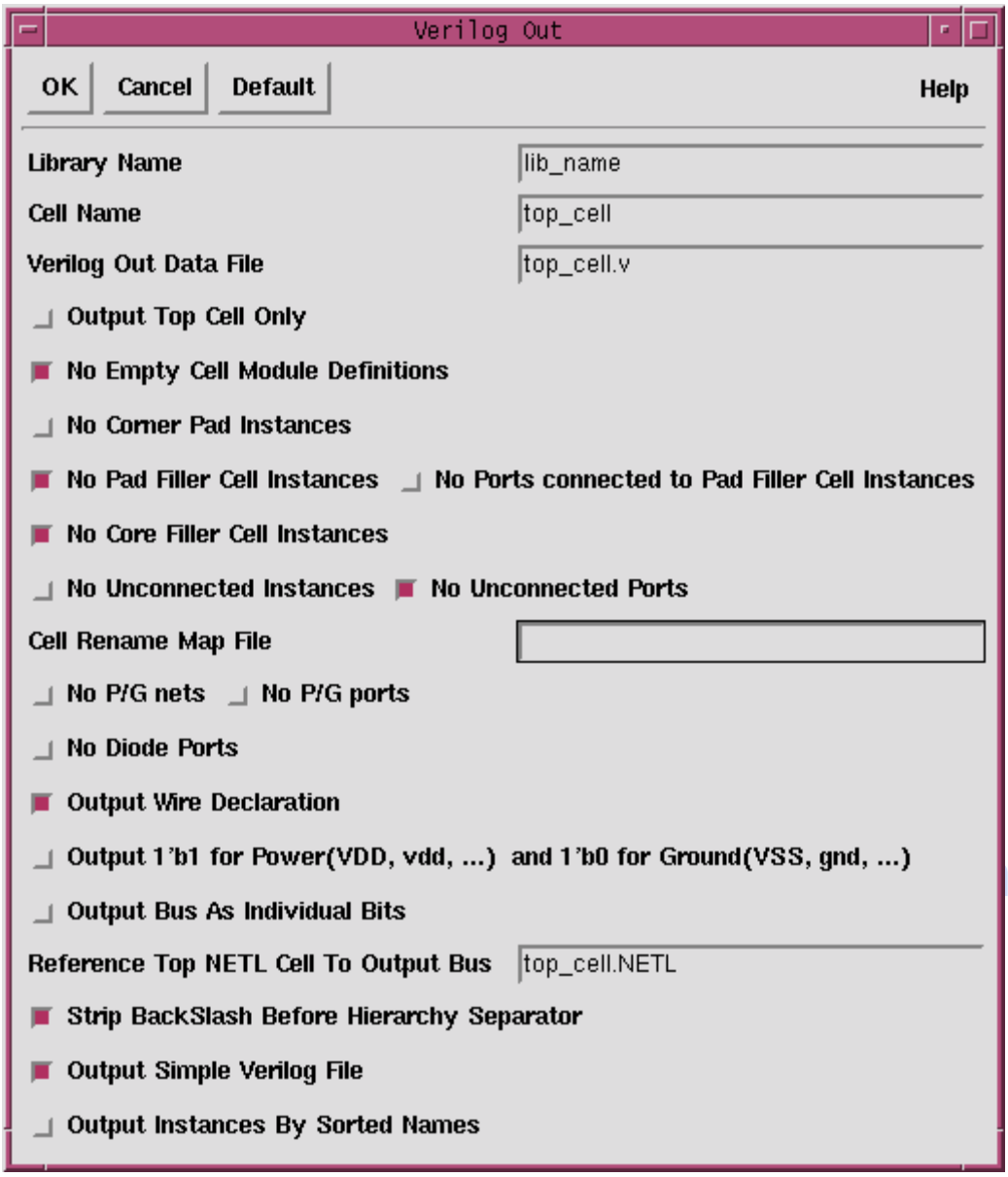

Figure 34. Apollo Verilog Out form

Run the v2lvs translator on the resulting Verilog netlist using the following options:

```
-l <filename> :  Defines the Verilog library file name (the library is
                 not translated).  The library file is parsed for inter-
                 face pin configurations (see -s).
-s <filename> :  Causes the string .INCLUDE "filename" to be put at the
                 beginning of the generated spice file (-o).  v2lvs does
                 NOT parse the SPICE file (see -l).
-o <filename> :  Defines the file for the resulting Spice-like netlist
                 to be used in the LVS (layout versus schematic) appli-
                 cation.  This result will include a translation of the
                 Verilog netlist.
-v <filename> :  Defines the Verilog design netlist file name.
```

After the Verilog netlist has been translated, bring up the Calibre LVS form as described in sections 13.1.1 or 13.1.2.

If the design has both analog and digital components, symbol and auCdl views (the latter just a copy of the symbol) for the digital block(s) should be created, the digital block(s) instantiated in the schematic, and the netlist(s) created with v2lvs entered in the "Source Includes (Opt.)" field (see Figure 27). With Calibre Interactive the netlist(s) should be entered in the "Include File" field (see Figure 29).

If the design is all digital, the "Make CBR from Sch." option can be disabled, and the path to the netlist created with v2lvs entered in the "Source (CBR) Path" field (see Figure 27). With Calibre Interactive the option "Export from schematic viewer" should be disabled and the path to the netlist created with v2lvs should be entered in the "Files:" field.

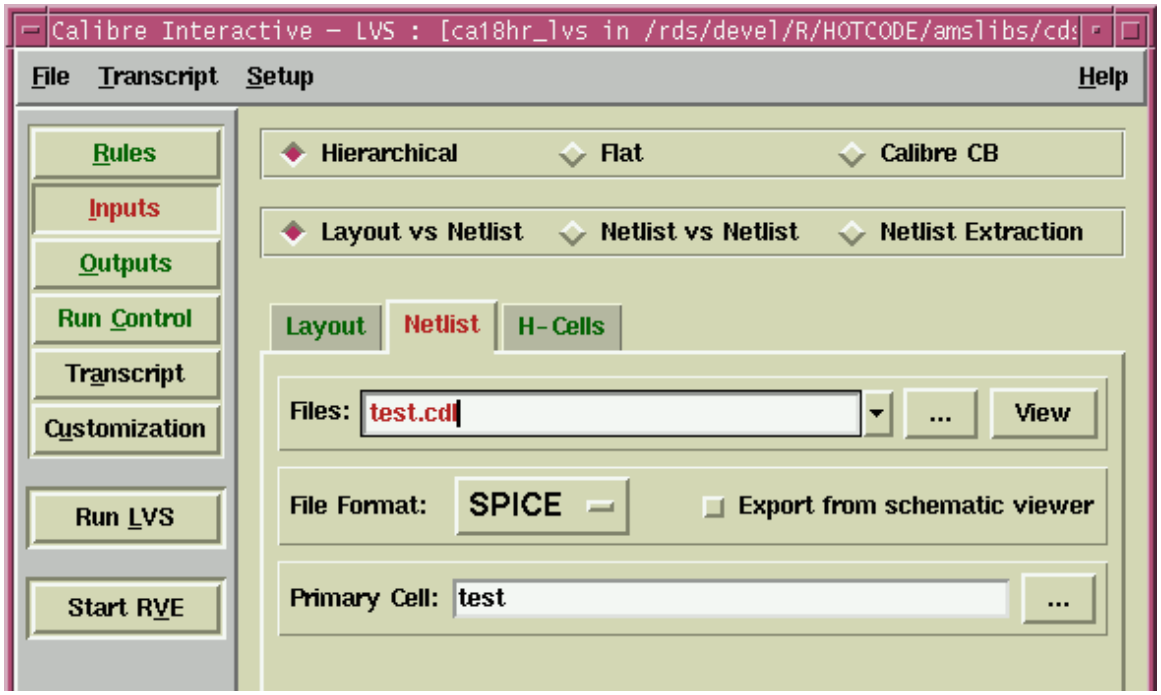

Figure 35. Calibre Interactive LVS netlisting options

The top-level pin order in a v2lvs netlist may not match that of the corresponding instance in the Cadence netlist. If this is the case, the Perl script shown in section 17.2.4 may be used to generate the SKILL code required to define the pin order for the CDL netlister.

14 Mixed-Signal LVS Using Calibre (pre-Feb. 2003 Kits)

Note: This methodology is required for Design Kits dated before February 18, 2003. See section 13 titled* Mixed-Signal LVS Using Calibre (post-Feb. 2003 Kits) *for the newer, streamlined methodology.

14.1 Methodology

Running LVS requires 3 steps.

1. Write a cbr netlist using the RDS netlister in the Schematic Editor.
2. Modify the netlist to include cdl netlists for the cells used in the schematic.
3. Run Calibre LVS.

Step 1: Write a cbr netlist using the RDS netlister.

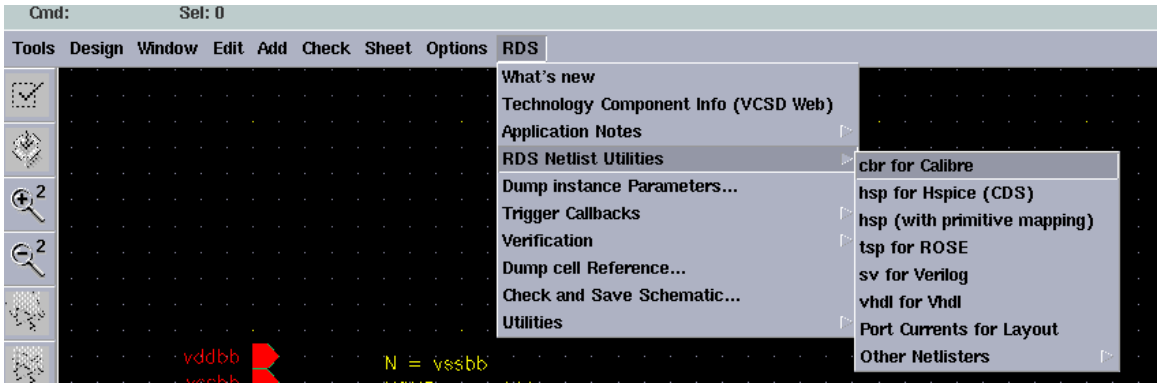

Figure 36. Writing a cbr netlist from the RDS menu

No additional windows are shown after "cbr for Calibre" is selected. A cbr netlist is written to the verification/cbr_dir directory. The exact location can be seen in the CIW. Inherited connections, if used, will be netlisted correctly by the netlister.

Step 2: Modify the netlist to include cdl netlists for the cells used in the schematic.

Modify the cbr netlist that is created by adding a .include statement to include subcircuit definitions for all the standard cells, e.g.,

.include "*path_to_asic_libs/A35/calibre/ri35sy101/allcells.cdl*"

These definitions can be found in the *calibre* directory delivered as part of the ASIC library. The file that you need to include is called *allcells.cdl*. This file has the ports listed in the same order as those that will be output by the cbr netlister. Note that individual cdl files for each cell might not have the ports listed in the same order.

Step 3: Run Calibre LVS.

Use the RDS menu in the Virtuoso Layout Editor to go to *RDS→Verification→Calibre LVS→Run LVS*.

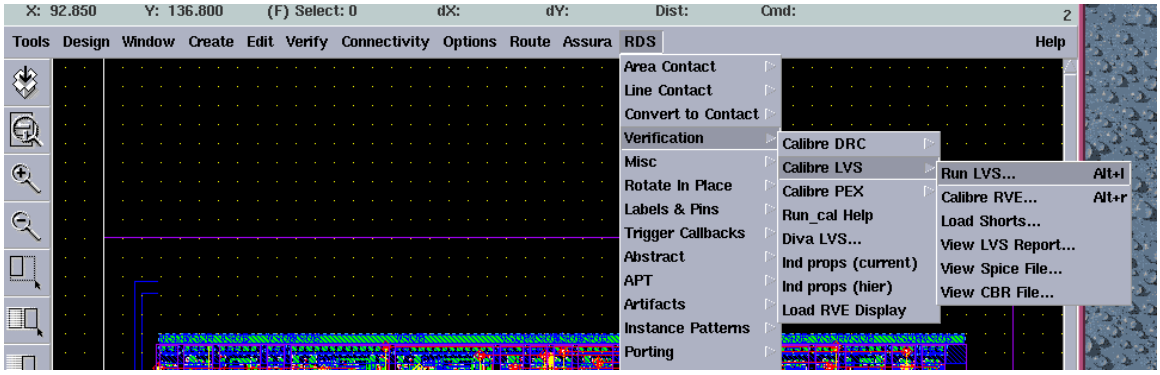

Figure 37. Running Calibre LVS from the RDS menu

On the Calibre LVS form, make sure *Make CBR from Schematic* is OFF and the *Source (CBR) Path* field contains the name of the file edited in Step 2. Also make sure that the Mode is set to *hierarchy* and *Automatch* is ON.

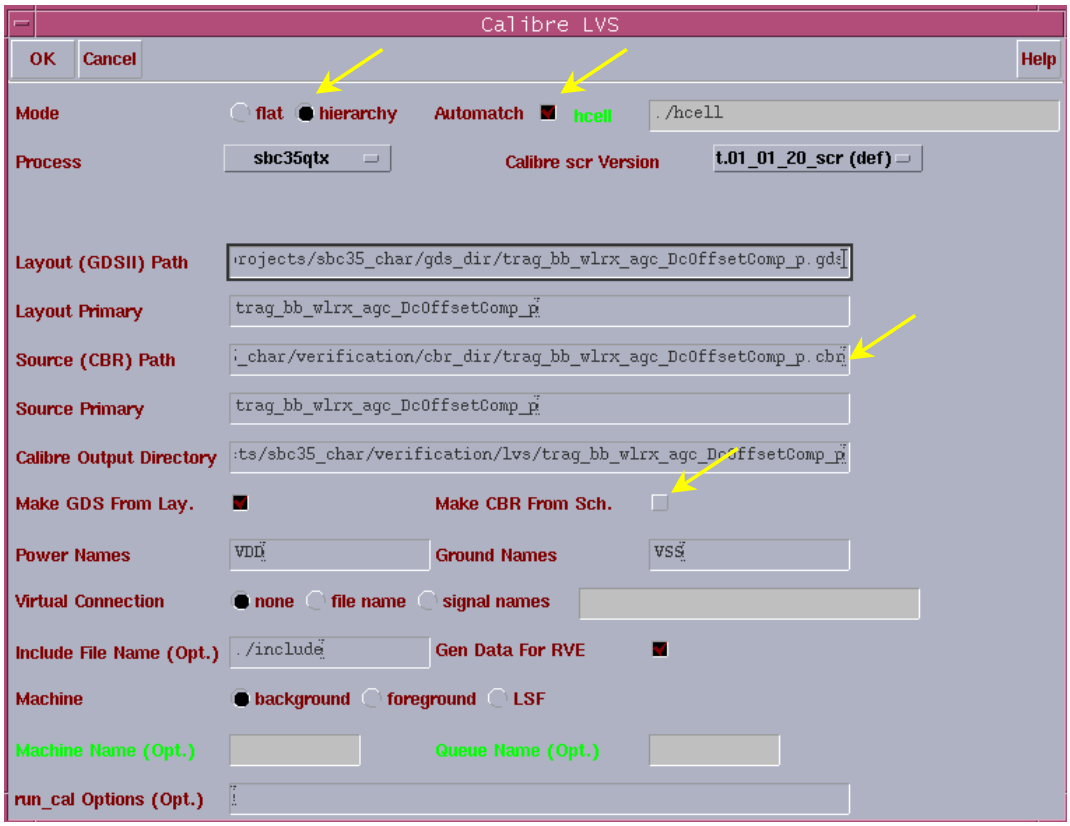

Figure 38. Calibre LVS form

Click on OK and progress with Calibre as usual.

15 Mixed-Signal LVS Using Assura

15.1 Assura 3.1.2 Methodology

This methodology can be used to verify designs with analog blocks and one or more digital blocks.

1. From top cell layout view, invoke Assura→run LVS...
   Run Assura LVS form appears.

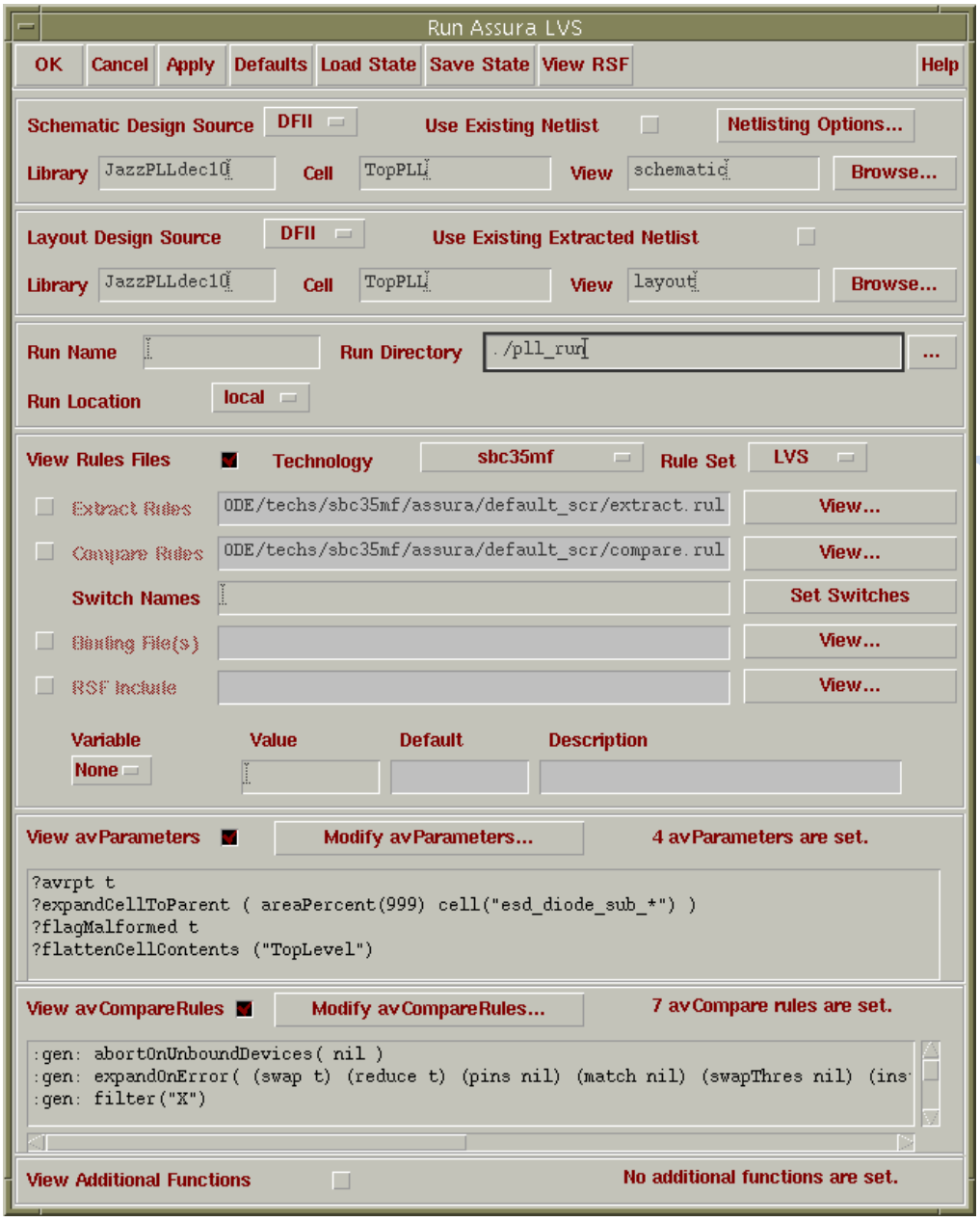

Figure 39. Running Assura LVS

2. Select **DFII** for **Schematic Design Source**.
3. Select **Netlisting Options** button and Schematic Netlisting Options Form appears.

   Select **CDL** for **Netlist Type**.
   Enter path to transistor level netlist of digital cells (allcells.cdl) in text field next to netlist type pulldown menu, and click **Add** button. The path to digital cells netlist appears in **CDL/SPICE Files For Netlisting** text display field.
   Click **OK** button on Schematic Netlisting Options Form to accept netlisting options.

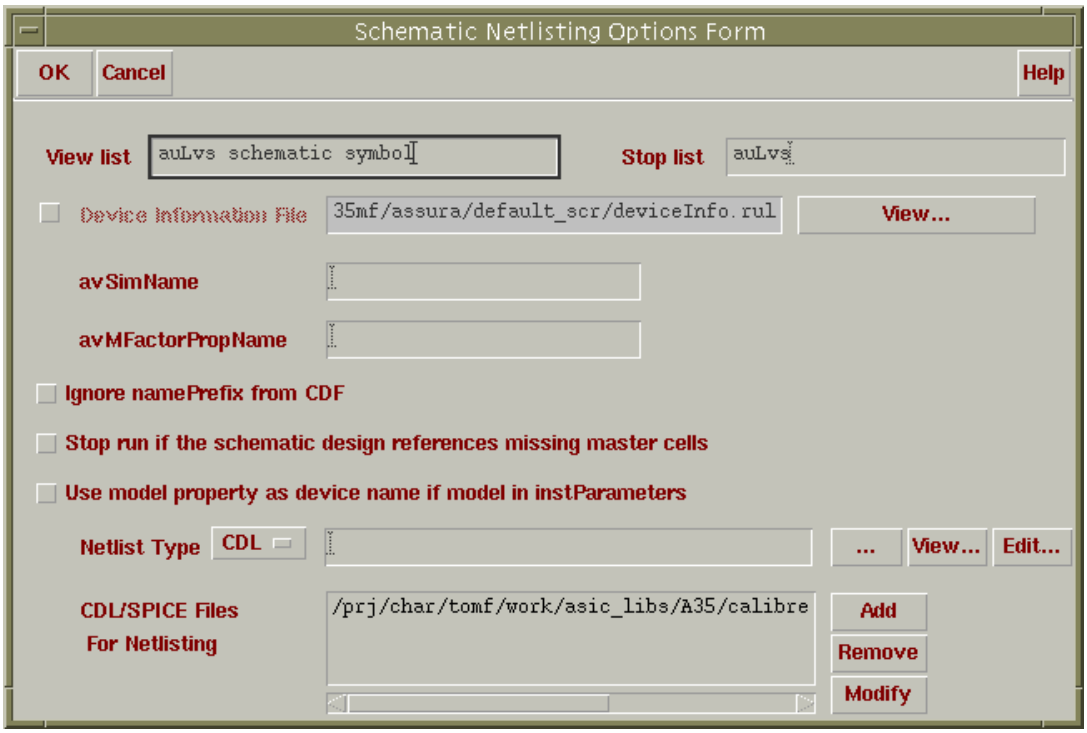

Figure 40. Assura Schematic Netlisting Options form

4. Select appropriate technology from **Technology** pulldown menu on Assura LVS form.
5. Select **LVS** for **Rule Set** pulldown menu.
6. Select avCompareRules button to set formGate(none) compare rule.
   Check **Use in Run** checkbox.
   Select **none** with pulldown menu.
   Select **Cell Name** with pulldown menu.
   Click + button, formGate(none) parameter appears in message box.

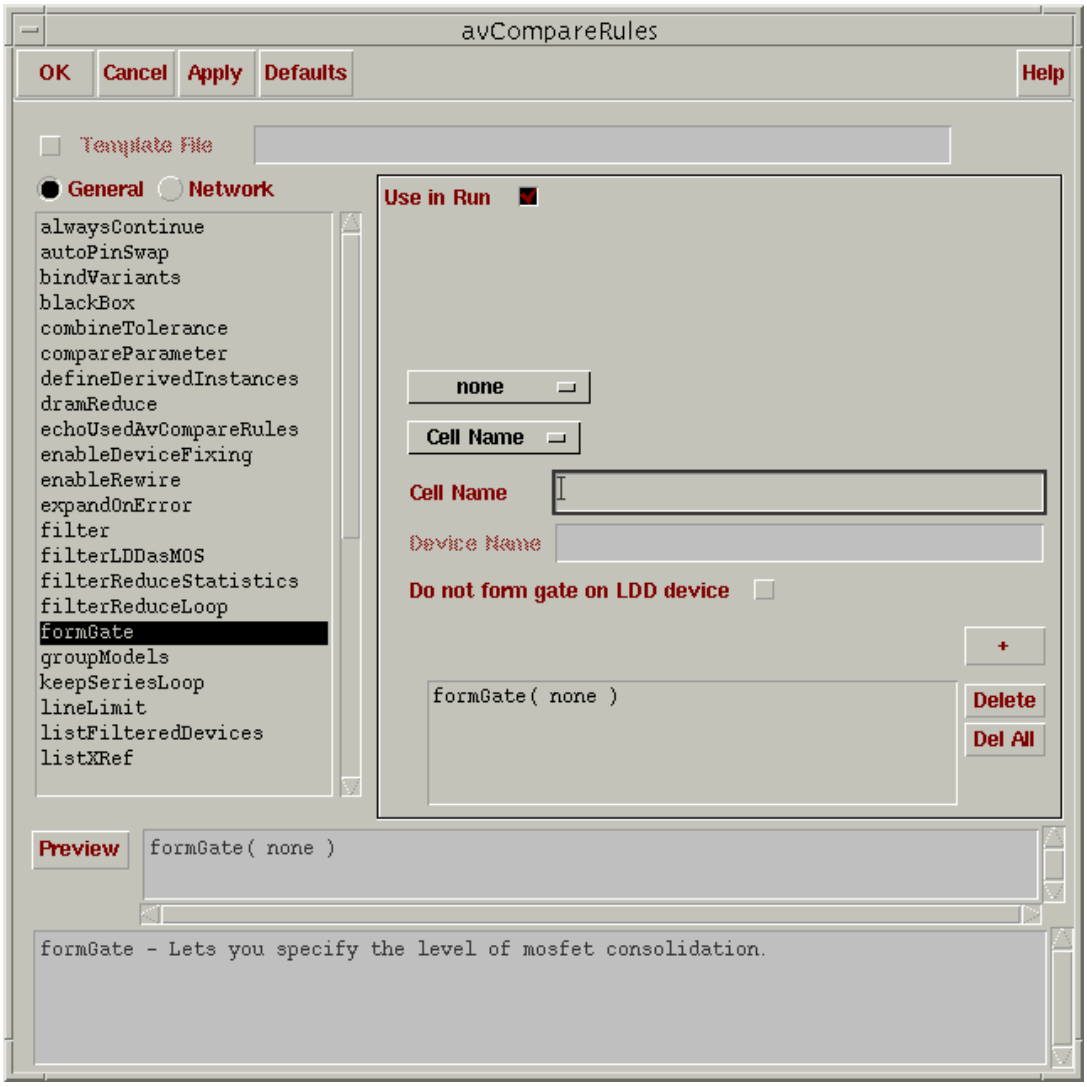

Figure 41. Assura LVS compare rules

7. Click **OK** button to start LVS run.
8. ***Useful tips***: If the design contains symmetrical circuitry, run LVS in flat mode.
   From Assura LVS form, select **Modify avParameters…** button.
   Select **?flattenCellContents** in parameter display list in avParameters form.
   Check **Use in Run** checkbox.
   Enter ***** in Cells text field.
9. ***Useful tips***: If the design contains dummy devices (devices with all terminals connected to substrate or ground), set filter("X") compare parameter.
   From Assura LVS form, select **Modify avComparRules…** button.
   Select **filter** in parameter display list in avCompareRules form.
   Check **Use in Run** checkbox.
   Enter **filter("X")** in text field.

Memory blocks generated by a compiler may require some additional steps in order to achieve a clean LVS run:

10. Delete or comment out the "*.*SCALE meter*" statement (if present) in the CDL netlist for the memory block.
11. Set the environment variable *RDS_ASSURA_REDUCE_PARALLEL_MOS* to *YES* prior to invoking icfb.
12. Label all the top-level pins in the layout view of the memory block according to section 3 (Pin Labeling) of chapter 3 in the "Assura Verification Flow" application note found under *JAZZ→Verification* in the CIW.
13. Select avCompareRules button to set autoPinSwap compare rule.
    Check **Use in Run** checkbox.
    Select **Enable autoPinSwap**.
    Enter 100000 for the **Threshold Value**.
    Click + button, autoPinSwap(t 100000) appears in message box.

For details on how to use Assura physical verification tool, please refer to Assura Physical Verification User Guide / Assura Physical Verification Developer Guide.

15.2 Assura 3.0 Methodology

This methodology can be used to verify designs with analog blocks and one or more digital blocks.

1. Make sure the digital block(s) do not have auCdl view(s) so the Calibre netlister will be able to netlist the digital block(s).
2. For each digital block, open the schematic view and invoke *JAZZ→Netlist Utilities→cbr for Calibre*. (Do not use *CBR for Calibre with Include files*, as Assura cannot handle the .include statement.)
3. Edit the cbr netlist (*DESIGN.cbr*) generated in *<project dir>/verification/cbr_dir*, adding pin names for global power and ground nets to the subcircuit definition. Note the addition of VSS and VDD pins to the subckt definition below.

```
.SUBCKT DESIGN Atb<0> Atb<1> Atb<2> Atb<3>
+ Ch1FltDACEN<0> h1FltDACEN<1> Ch1FltDACEN<2> Ch1FltDAC_0<0>
+ Ch1FltDAC_0<1> h1FltDAC_0<2> h1FltDAC_0<3> Ch1FltDAC_0<4>
+ Ch1FltDAC_0<5> Ch1FltDAC_0<6> Ch1FltDAC_0<7> Ch1FltDAC_1<0>
...
+ ch2e_DacMX<1> ch2e_DacMX<2> ch2e_DacMX<3> VSS VDD rfcbClk
+ rfcbDin rfcbDout
```

4. Copy the digital block's symbol view to a new view, e.g., mysymbol. (It doesn't need to be mysymbol; it's anything but auCdl, auLvs, or symbol.)
5. Add global power and ground pins with net expressions to mysymbol view, and save the edits.
6. Open the top cell schematic, and change the view for the digital block instance from symbol to mysymbol.

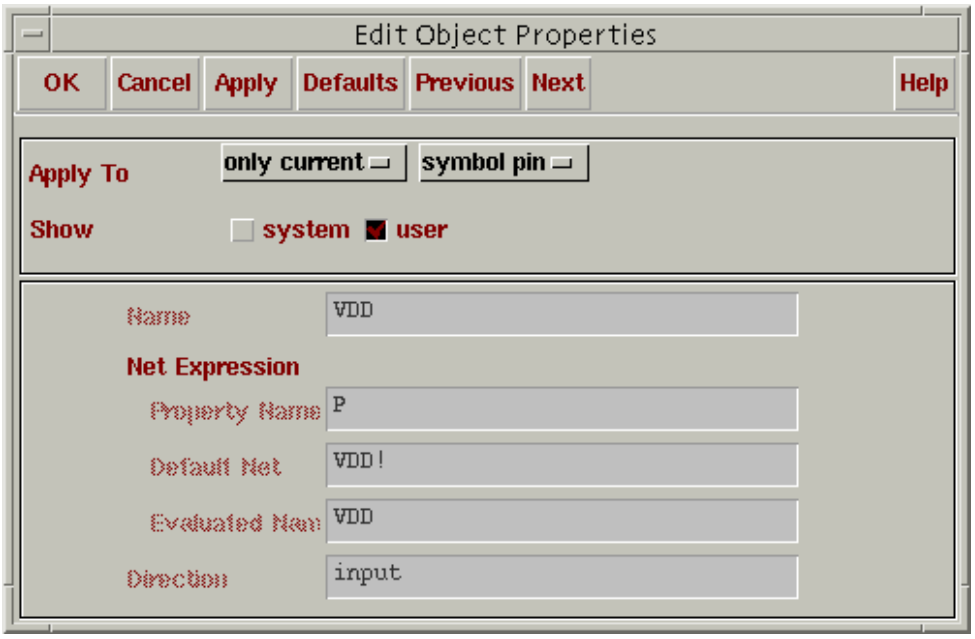

Figure 42. Pin with net expression property

7. Add 2 netSet properties to the digital block instance:
   Name: N, Type: netSet, Value: <global ground net>
   Name: P, Type: netSet, Value: <global power net>

Repeat steps 3-7 for each digital block if more than one is present in the top cell.

8. Check and Save the top cell schematic.
9. Rename the schematic view for all digital blocks in the top cell, to prevent the CBR netlister from descending into them when netlisting the full design.
10. From the top cell layout view, invoke *Assura→Run LVS...*
11. Select the **Netlisting Options…** button to set the View and Stop lists, and the paths to the CDL files:
    – set the View list to: schematic auLvs
    – set the Stop list to: auLvs
    – if using Assura 3.03 or higher, enter the path to the cbr netlist(s) generated for the digital block(s) in the **CDL/SPICE File** field, and click the **Add** button
    – enter the path to the transistor level netlists of the library cells (allcells.cdl) in the same field, and click the **Add** button again
    – click the **OK** button on the Schematic Netlisting Options Form to accept these netlisting options.
12. Select the **Modify avCompareRules…** button to set the **formGate** compare rule:
    – select **formGate** from the rules list
    – check the **Use in Run** checkbox
    – select **none** from the first pulldown menu
    – select **Cell Name** from the second pulldown menu
    – click the + button; formGate(none) parameter appears in the message box.

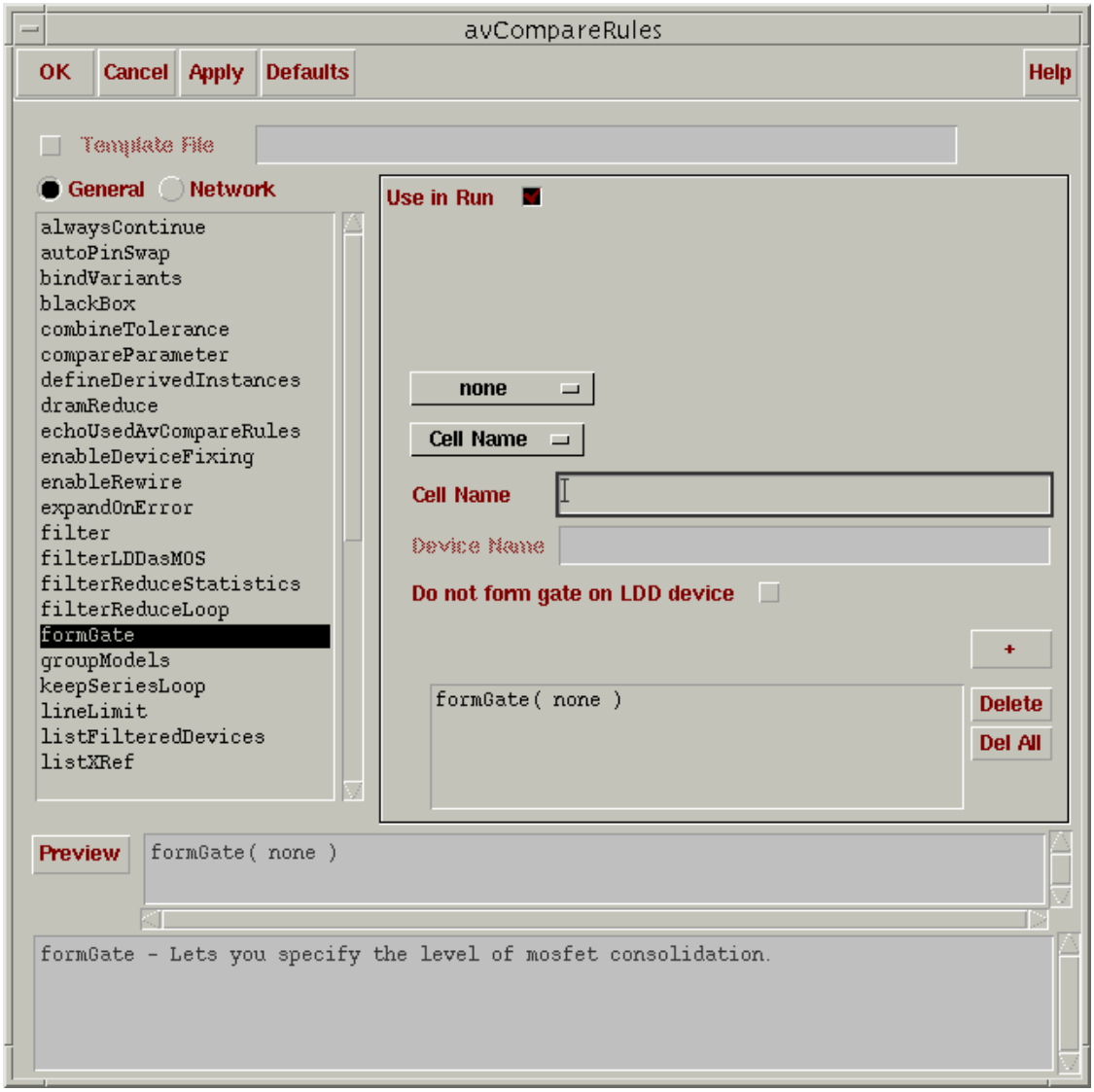

Figure 43. Assura LVS compare rules

13. Click the **OK** button to start the LVS run.

Useful tips: If the design contains symmetrical circuitry, run LVS in flat mode:
- from the Assura LVS form, select the **Modify avParameters…** button
- select **?flattenCellContents** from the parameter list
- check the **Use in Run** checkbox
- enter ***** in the Cells text field.

Useful tips: If the design contains dummy devices (devices with all terminals connected to substrate or ground), set the **filter** compare rule:
- from the Assura LVS form, select the **Modify avCompareRules…** button
- select **filter** from the parameter list

– check the **Use in Run** checkbox
– enter **filter("X")** in the text box.

For details on how to use the Assura physical verification tool, please refer to the Assura Physical Verification User Guide / Assura Physical Verification Developer Guide.

Proceed to steps 14-16 if using Assura 3.0 - 3.02.

14. Select the **Stop Run** button in the Progress window to stop the run. This step is necessary to generate the rsf and vlr files.
15. Edit the vlr file, adding avReadCDL commands to include *<digital block>.cbr* and allcells.cdl files.
    avReadCDL("*<path to cbr netlist generated for digital block>*")
    avReadCDL("*<path to allcells.cdl>*")
16. Run Assura in batch mode in the current working directory:
    assura <rundir>/<runName>.rsf > <rundir>/<runName>.log

16 Concurrent Use of 2- and 3-rail Libraries

The A18 library allows for the concurrent use of cells from the 2- and 3-rail libraries (scgp and scgp_analog) with only minor differences in the transistor-level simulation and Calibre LVS flows (support for Assura LVS has not been implemented).

Section 12.1 describes the methodology for transistor-level simulation with a single digital library. The only difference when using both the 2- and 3-rail libraries together is that two include files corresponding to the two libraries need to be specified:

…/spectre/scgp/allcells.scs and …/spectre/scgp_analog/allcells_a.scs

or

…/hspice/scgp/allcells.cir and …/hspice/scgp_analog/allcells_a.cir.

Note that …/spectre/scgp_analog/allcells.scs and …/hspice/scgp_analog/allcells.cir are also included in the library. These files should only be used when the 3-rail library is the only library being referenced.

Sections 13 and 14 describe the methodologies for Calibre LVS verification with a single digital library. As with simulation, the only difference when using both the 2- and 3-rail libraries together is that two include files corresponding to the two libraries need to be specified:

…/calibre/scgp/allcells.cdl and …/calibre/scgp_analog/allcells_a.cdl

Again, a …/calibre/scgp_analog/allcells.cdl file is available but should only be used when the 3-rail library is the only library being referenced.

Proper LVS verification relies on the cell names from the 3-rail library being mapped from <cellName> to <cellName>_a. A streamout cell name mapping table is automatically defined when a cellview from the 3-rail library is opened for the first time during a session (a message in the CIW provides confirmation this has happened). If a block is streamed out without a cell from the 3-rail library having been opened previously or the mapping table having been defined manually, LVS will fail.

The Cadence CDL netlister will also perform the cell name mapping automatically, but any netlist generated outside the Cadence environment will require that this mapping be performed by the user.

17 Memories

17.1 Memory Generation

The following memory compilers are available for the Jazz 0.18µm 6-level metal and 5-level metal (with thin metal4) processes.

Table 5. Jazz memory compilers

	Max Size	Min Size
Single-port Static RAM	8192×64 mux16 4096×128 mux8 2048×128 mux4	64×2 mux16 32×2 mux8 16×2 mux4
Dual-port Static RAM	8192×64 mux16 4096×128 mux8 2048×128 mux4	64×2 mux16 32×2 mux8 16×2 mux4
Single-port Register File	256×64 mux4 512×128 mux2 256×128 mux1	16×2 mux4 8×4 mux2 4×8 mux1
Dual-port Register File	1024×64 mux4 512×128 mux2 256×128 mux1	16×2 mux4 8×2 mux2 4×2 mux1
Diffusion ROM	32768×64 mux32 16384×128 mux16 8192×128 mux8	256×2 mux32 128×2 mux16 64×2 mux8

17.2 Compiled Instance Post-processing

17.2.1 LEF File Layer Names

The layer names in the LEF files (*.vclef) generated by the Single-port SRAM and Single-port Register File compilers will need to be changed as follows in order to match the names in the technology LEF files:

- METAL# → metal#
- VIA12 → via1
- VIA23 → via2
- VIA34 → via3

A short sed script that reads “file1”, makes the desired substitutions, and writes the results to “file2” is shown below:

```
sed –e s/METAL/metal/ -e 's/VIA\([1-3]\)[2-4]/via\1/' file1 > file2
```

17.2.2 LEF File Routing Blockages

The routing blockages are missing from the LEF file (*.vclef) generated by the Single-port Register File compiler. A quick fix for this is to add the following lines to the OBS section of the file:

```
#core
LAYER metal1 ;
RECT B+1.2 B+1.64 X-B-1.2 Y-B-1.64 ;
LAYER metal2 ;
RECT B+1.2 B+1.64 X-B-1.2 Y-B-1.64 ;
LAYER metal3 ;
RECT B+1.2 B+1.2 X-B-1.2 Y-B-1.2 ;
```

where B is equal to two times the power bus width, and X and Y represent the dimensions of the macro given in the SIZE statement.

Note that the metal3 obstruction is slightly larger to prevent any routes from shorting supply connections, but it also prevents the metal3 pins from being accessible. The metal1 and metal2 pins are still available, however. This work-around may not be 100% effective in all situations, but it should work most of the time.

17.2.3 CDL Netlist Device Names

The CDL netlists will also need to be edited to change the device model names in order to match the names in the LVS rules:

- P → PFET
- N → NFET
- DN → NDIODE

As with the LEF files, an sed script can be used to accomplish this:

```
sed –e ’s/ \([NP]\) / \1FET /’ –e ’s/ DN / NDIODE /’ file1 > file2
```

Note that there is a blank space before and after each model name in the script.

The CDL edits above apply to all supported 0.18µm processes except for CA18HR, where the diode model name is DN. When using this process the sed script should be modified as follows:

```
sed –e ’s/ \([NP]\) / \1FET /’ –e ’s/ NDIODE / DN /’ file1 > file2
```

17.2.4 CDL Netlist Pins

The netlists generated by the Dual-port SRAM, Dual-port Register File, and Diffusion ROM compilers make use of global supply pins. It may be desirable to make these supply pins local in order to ease the process of integrating the memories with the rest of the design.

Another potential issue is the pin order that the CDL netlister produces for the memory instance. It may be necessary to define this pin order so it will match that in the CDL netlist generated by the compiler.

The following Perl script can be used to automate these tasks (the first line will need to be modified if the Perl executable is in a location other than /usr/bin). In addition to converting the supply pins, this script will also write a *.il file with the SKILL code required to define the netlisting pin order for the memory instance; the LIBRARY variable in this file must be updated prior to loading it into the CIW with the command

```
load “fileName”
```

Executing the Perl script without any arguments outputs a brief description.

```
#!/usr/bin/perl
$progName = substr($0, rindex($0, "/")+1);
if ($#ARGV < 0) {
  print <<"  END";
  \n   Usage: $progName <file_name> [cell_name]\n
          <file_name> is a CDL netlist whose global VDD & VSS
          pins are to be converted to local pins.  The original
          file is renamed to <file_name>_orig.\n
          [cell_name] is an optional argument that identifies
          the top-level subcircuit (if omitted, last subcircuit
          read will be taken as top level); SKILL code defining
          the CDL netlisting pin order for this subcircuit is
          written out to a [cell_name]_pins.il file (value of
          LIBRARY variable must be updated prior to use).
  END
  exit;
}
$origFile = $ARGV[0]."_orig";
$cellName = $ARGV[1];
if (-e $origFile) {
  print "\n   The file \"$origFile\" already exists.";
  print "\n   Do you want to overwrite it (y/n)? ";
  $answer = getc;
  unless ($answer eq "y" || $answer eq "Y") {exit}
}
if (open(iFile, $ARGV[0])) {
  $subcktDef = $subcktCall = 0;
  $outFile = $ARGV[0]."_temp";
  open(oFile, ">$outFile");
  while ($line = <iFile>) {
    if ($line =~ /^\.global/i) {
      $line =~ s/^(\.global)/\**$1/i;
      print(oFile $line);
    }
    elsif ($line =~ /^\.subckt\s+(\S+)\s+/i) {
      $subcktDef = 1;
      $savedLine = $line;
      if ($cellName) {
        if ($cellName eq $1) {$topCell = 1}
        else {$topCell = 0}
```

```
    }
    else {$topCellPins = ""}
  }
  elsif ($line =~ /^x/i) {
    if ($subcktCall == 1) {
      $savedLine =~ s/(\s+)(\S+\s*$)/$1VDD VSS $2/;
      print(oFile $savedLine);
    }
    $subcktCall = 1;
    $savedLine = $line;
  }
  elsif ($line =~ /^\+/) {
    print(oFile $savedLine);
    if ($subcktDef == 1) {
      if ($cellName) {
        if ($topCell == 1) {$topCellPins = $topCellPins.$savedLine}
      }
      else {$topCellPins = $topCellPins.$savedLine}
    }
    $savedLine = $line;
  }
  elsif ($subcktDef == 1) {
    $subcktDef = 0;
    $savedLine =~ s/($)/ VDD VSS$1/;
    print(oFile $savedLine.$line);
    if ($cellName) {
      if ($topCell == 1) {$topCellPins = $topCellPins.$savedLine}
    }
    else {$topCellPins = $topCellPins.$savedLine}
  }
  elsif ($subcktCall == 1) {
    $subcktCall = 0;
    $savedLine =~ s/(\s+)(\S+\s*$)/$1VDD VSS $2/;
    print(oFile $savedLine.$line);
  }
  else {print(oFile $line)}
}
close(iFile);
close(oFile);
rename($ARGV[0], $origFile);
rename($outFile, $ARGV[0]);
$topCellPins =~ /^\.subckt\s+(\S+)\s+/i;
$cellName = $1;
$topCellPins =~ s/^\.subckt\s+\S+\s+//i;
$topCellPins =~ s/\+\s+//g;
$topCellPins =~ s/\[(\d+)\]/<$1>/g;
$topCellPins =~ s/[\ +]/\" \"/g;
$topCellPins =~ s/(.+)/              \"$1\"/g;
$topCellPins =~ s/^              //;
chomp($topCellPins);
open(oFile, ">${cellName}_pins.il");
print oFile <<"  END";
/*************************************************/
 LIBRARY = "libName"
 CELL    = "$cellName"
/*************************************************/
let( ( libId cellId cdfId )
```

```
        unless( cellId = ddGetObj( LIBRARY CELL )
            error( "Could not get cell %s." CELL )
        )
        unless( cdfId = cdfGetBaseCellCDF( cellId )
            cdfId = cdfCreateBaseCellCDF( cellId )
            cdfId->simInfo = list( nil )
        )
        ;;; Simulator Information
        cdfId->simInfo->auCdl = '( nil
            termOrder (
                $topCellPins
            )
            netlistProcedure ansCdlSubcktCall
            namePrefix "X"
        )
        cdfSaveCDF( cdfId )
    )
    END
    close(oFile);
}
else {print "\n   Could not read the file \"$ARGV[0]\".\n"}
```

If the pin order mis-match is limited to the order of individual bits within a bus, another solution may be to have the memory compiler "bit-blast" the busses into scalar pins. To do this, select *Utilities→Advanced Options* in the compiler GUI and delete any bus delimiter characters specified (square brackets by default).

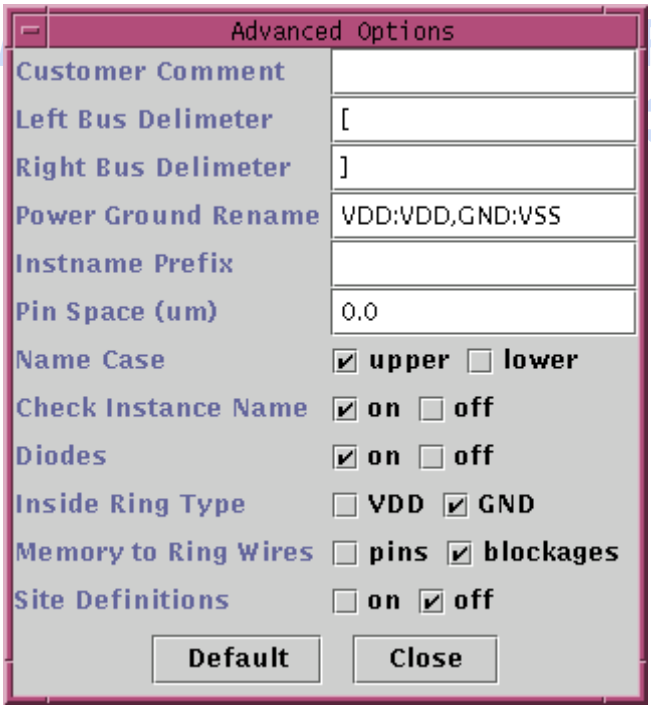

Figure 44. Memory compiler options

The resulting bus pin notation will be XYZ_1_ instead of XYZ[1].

17.2.5 ROM Code

The ROM coding may result in Poly antenna violations under the conditions described below:

1. A multiplexer width of 16 or 32 has been selected.

2. When divided into consecutive 16-word segments, the ROM data in one or more pair of adjacent columns (2^{nd}-3^{rd}, 4^{th}-5^{th}, 6^{th}-7^{th}, etc.) includes only one bit set to "0". In the example shown below, the 4^{th}-5^{th} column pair of data would result in an antenna violation.

   x 11 11 11 x
   x 11 11 11 x
   x 11 11 10 x
   x 11 11 11 x
   x 11 11 11 x
   x 11 01 11 x
   x 11 11 11 x
   x 11 11 11 x
   x 11 11 01 x
   x 11 11 11 x
   x 11 11 11 x
   x 11 11 11 x
   x 11 11 11 x
   x 11 11 01 x
   x 11 11 11 x
   x 11 11 11 x

18 Design Guidelines and Techniques

18.1 Power Estimation

18.1.1 Calculating Junction Temperature

Given θ_{ja} from the packaging vendor, the expected junction temperature can be calculated for a device at different ambient temperatures. The junction temperature should remain within the limits of the fast and slow corners. Section 6, "Characterization", lists the minimum and maximum characterized temperatures for each process library.

Junction temperature can be calculated using the equation below:

$T_j = T_a + (\theta_{ja} \times P_d)$

where

T_j = junction temperature in °C
T_a = ambient temperature in °C
θ_{ja} = package thermal impedance in °C/W
P_d = chip power dissipation in W