HSIM^{plus}® Reference Manual

Version C-2009.06, June 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of ____ and its employees. This is copy number __________."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, the Synplicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{sm})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

Contents

About This Manual

This manual describes how to use the Synopsys HSIM^{plus} tool. The following sections provide a guide to this manual, as well as to other documentation that accompanies this tool.

Audience

Users are expected to be familiar with personal computers and workstations running an operating system (OS) with a graphic user interface.

The purpose of this manual is to enable users to produce the best nanometer IC design, verification, and analysis results.

Related Publications

For additional information about HSIM^{plus}, see

- The HSIM^{plus} Release Notes, available on SolvNet (see Accessing SolvNet on page ii)
- Documentation on the Web, which provides HTML and PDF documents and is available on SolvNet (see Accessing SolvNet on page ii)

You might also want to refer to the documentation for the following related Synopsys products:

- HSPICE
- NanoSim

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.

Convention	Description
Italic	Indicates a user-defined value, such as *object_name*.
Bold	Indicates user input—text you type verbatim—in syntax and examples.
[ ]	Denotes optional parameters, such as: write_file [-f *filename*]
...	Indicates that parameters can be repeated as many times as necessary: *pin1 pin2 ... pinN*
|	Indicates a choice among alternatives, such as low | medium | high
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services, which include downloading software, viewing Documentation on the Web, and entering a call to the Support Center.

To access SolvNet:

1. Go to the SolvNet Web page at http://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click Help on the SolvNet menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com/EnterACall (Synopsys user name and password required).
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

Introduction

Describes tool features and provides recommendations for use.

HSIM^{plus}

HSIM^{plus} expands and builds upon the production-proven HSIM simulator to address the most critical problems associated with the physical effects of interconnect wiring and short-channel effects in nanometer IC designs.

HSIM^{plus} is a complete transistor-level simulation and analysis platform for the design and verification of nanometer integrated circuits. Figure 1 on page 1 shows the HSIM^{plus} structure.

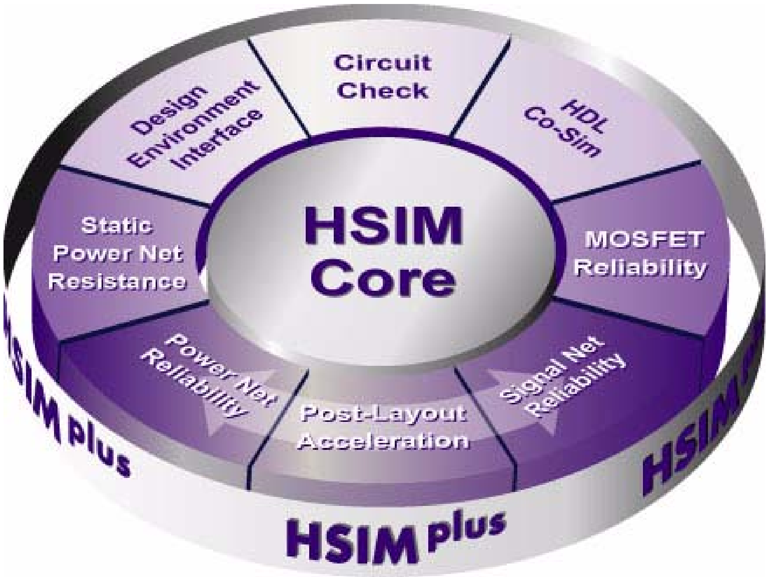


Figure 1 HSIM^{plus} Structure

The HSIM^{plus} structure is designed to maximize the designer's ability to simulate designs of various sizes and complexities with a cohesive set of HSIM tools. The HSIM^{plus} core platform and suite contain the following collection of features.

HSIM^{plus} Features

As shown in Figure 1, the HSIM^{plus} structure provides these features:

Post-Layout Acceleration

> Post-Layout Acceleration (PLX) enhances simulation performance and memory efficiency by providing hierarchical back-annotation of parasitics from extracted DSPF and SPEF files. This option also provides the ability to back-annotate a huge number of coupling capacitors into the hierarchical simulation data structures, while retaining the circuit hierarchy and maintaining performance and precision for transient simulation. Refer to Chapter 2, Post-Layout Acceleration (PLX) and SP2DSPF Utility for detailed user information.

Power Net Reliability Analysis

> Power Net Reliability Analysis (PWRA) incorporates technology for analysis of IC power networks, including the effect of dynamic IR drop on design characteristics, and adds capabilities for analyzing and physical visualization of potential chip failure from electro-migration due to high current densities. Refer to Chapter 3, Power Net Reliability Analysis (PWRA) for detailed user information.

Static Power Net Resistance

> Static Power Net Resistance (SPRES) performs the complex calculation of all pad-to-pin and pad-to-internal power net node resistances, generating a graphical map of the result. This screens power nets for electrical problems and eliminates the need to complete lengthy simulations in order to analyze gross errors in the layout of the power network. Refer to Chapter 4, Static Power Net Resistance (SPRES) for detailed user information.

Signal Net Reliability Analysis

> Signal Net Reliability Analysis (SIGRA) is used for calculation and physical visualization of potential chip failure from electro-migration in signal nets, correctly considering bi-directional current flow. Refer to Chapter 5, Signal Net Reliability Analysis (SIGRA) for detailed user information.

The SP2DSPF Utility

All the above HSIM^{plus} options may also access the SPICE-to-DSPF utility, which can extract semantically equivalent device and interconnect back-annotation files from a flat or hierarchical full-chip extracted SPICE netlist. With sp2dspf, users whose extraction tools do not adequately support netlisting of DSPF may access the post-layout options of HSIM^{plus}. Refer to Chapter 2, Post-Layout Acceleration (PLX) and SP2DSPF Utility for detailed user information.

MOSFET Reliability Analysis

MOSFET Reliability Analysis (MOSRA) is used for modeling of device aging induced by hot carrier injection (HCI) and negative bias temperature instability (NBTI). Both standard aging and user-specified aging models are supported. Refer to Chapter 6, MOSFET Reliability Analysis (MOSRA) for detailed user information.

CircuitCheck

Provides information on the Circuit Check (CCK) option and its capabilities including: reporting potential circuit problems during the early stage of simulation, detecting incorrect input data or tool usage, analyzing latch condition before simulation starts, monitoring node voltages, generating reports, comparing waveforms to determine when and why a circuit node is triggered to change its value, computing static rise and fall delays to transistor gates, and analyzing crosstalk noise voltage caused by coupling capacitors. Refer to Chapter 8, CircuitCheck for detailed user information.

ADMS Interface

HSIM-ADMS is the single-kernel integration of HSIM into the Mentor's multilanguage ADvanced Mixed-signal Simulator (ADMS). This integration adds capability for co-simulation of circuit blocks represented in SPICE format in HSIM, along with VHDL, Verilog, VHDL-AMS, and Verilog-AMS in ADMS. HSIM-ADMS allows designers to verify large mixed-signal designs with the flexibility of simulating various blocks at different levels of abstraction. Refer to Chapter 9, HSIM-ADMS Integration for detailed user information.

HSIM-Virtuoso Analog Design Environment Interface

Provides information on the HSIM-Virtuoso Interface including installation and un-installation information and how to install and use HSIM^{plus} with the Cadence® Virtuoso® Analog Design Environment software integration interfaces. Refer to Chapter 10, HSIM-Virtuoso Analog Design Environment Interface for detailed user information.

Verilog / VHDL / HSIM Co-Simulation

Provides information on how Verilog / VHDL Co-simulation works using HSIM. Refer to Chapter 11, Verilog/VHDL/HSIM Co-Simulation for detailed user information.

HSIM

The HSIM simulator is the core of the HSIM^{plus} platform. HSIM performs transient analysis, DC analysis, AC analysis, and Monte Carlo analysis supporting the following circuit elements:

- MOSFET (Metal-Oxide Semiconductor Field-Effect Transistors)
- Bipolar transistors
- Diodes
- Junction field-effect transistors
- Resistors
- Capacitors
- Self and Mutual inductors
- Independent voltage and current sources
- Linear and nonlinear controlled voltage and current sources
- Lossless and lossy transmission lines

Note:

Some special elements are not currently supported in AC analysis. Refer to the *HSIM Simulation Reference Manual: Chapter 10, AC Small-Signal Analysis* for details.

Interactive Circuit Analysis

HSIM's interactive circuit analysis provides a circuit debugging environment that interrupts simulation and performs interactive circuit diagnosis at selected points in time. HSIM analysis provides circuit information, such as:

- Node voltage
- Node capacitance
- Element current
- Element conductance and capacitance
- Fan-in and fan-out elements to a node
- Element terminal nodes
- Active element drivers to a node
- Active loading elements to a node
- Excessive current checks
- DC path between two nodes

Applications

HSIM provides analysis for many applications:

- Full-chip transistor-level functionality verification at pre-layout and post-layout stages
- High-speed circuit simulation for memory circuits, including:
  - DRAM
  - SRAM
  - ROM
  - EPROM
  - EEPROM
  - Flash memory
- Timing and power characterization for memory circuits with post-layout parasitics. Microprocessor, DSP, MPEG, and other large IP cores can also be characterized provided a reasonable number of user-selected input stimuli are made available.

- Cross-talk noise simulation
- High-speed analog and mixed-signal circuit simulation
- Functionality, timing, and power analysis report
- Full-chip post-layout simulation including all layout parasitics to determine the following:
  - Circuit performance under the influence of power net IR drop
  - Effect of coupling capacitance on delay, delay noise, functionary, or glitch power.

Input/Output Data

Simulation results stemming from the analysis allow probing designs to obtain the following:

- Nodal analog voltage waveforms
- Nodal digital logic-state waveforms
- Element branch current waveforms through a transistor, a resistor, a capacitor, an inductor or an independent voltage source

These output data can be displayed with either of the following waveform viewers:

- nWave
- SimWave
- WaveView

Check commands are used to provide detailed timing and power measurements. Refer to the *HSIM Simulation Reference Manual: Chapter 13, Timing and Power Analysis* for detailed information about check commands.

HSIM's input data format is generally compatible with the input format of industrial standard circuit simulators such as HSPICE.

Hierarchical Simulation Technology

HSIM's high capacity is provided by the innovative use of hierarchical technology—simulation that stores the circuit netlist hierarchically in memory instead of flattening the netlist.

This minimizes memory requirements for identical cells and subcircuits. HSIM's isomorphic matching techniques eliminates redundant computation for identical sub-circuits and provides greatly enhanced throughput.

To further improve throughput for post-layout simulation, HSIM^{plus} options include parasitic reduction for both signal and power nets. as well as hierarchical back-annotation. These combined capabilities give HSIM the ability to accurately and efficiently simulate circuits of tens to hundreds of millions of transistors.

HSIM simulates and analyzes

- Large circuit blocks
- Groups of large interacting circuit blocks
- Full-chip designs

Limitations and Recommendations

It is recommended that HSIM be employed to investigate and analyze nanometer effects prevalent among high-speed nanometer circuits, such as:

- The influence of power net IR drop on circuit performance
- Cross-coupling
- Full-chip post-layout simulation with layout parasitics

Many of these problems may not be observable while running independent block-level simulations and will require full-chip circuit simulation for detection and correction.

Caution!

> HSIM is not recommended for circuits containing fewer than 100 transistors. HSIM is more effective for simulating VLSI circuits.
>
> In addition, HSIM should not be used as an exhaustive simulation tool to verify large designs with a huge number of test vectors. HSIM is not designed to handle such applications. It is suggested that test vectors be intelligently selected for key functionality and critical circuit behavior to be tested without using exhaustive vector sets. If it is necessary that large sets of test vectors be simulated, dedicate a powerful computer to execute HSIM for the required time.

2

Post-Layout Acceleration (PLX) and SP2DSPF Utility

This chapter describes how to use the HSIM^{plus} PLX Post-Layout Acceleration option for hierarchical back-annotation (HBA) technology, structural back-annotation (SBA) technology, and the SP2DSPF utility.

With the majority of design starts now being targeted to processes at 130nm and below, post-layout analysis for nanometer effects is now a necessity. In dynamic circuit simulation, post-layout analysis can be performed by simulating a circuit with all the parasitic RCs and coupling capacitors extracted from the layout. A brute force approach to this problem—simulation of the flat extracted netlist with all the parasitic RCs and coupling capacitors—is not practical because of the huge size of the extracted netlist. Direct simulation of the extracted netlist consumes an enormous amount of CPU time and computer memory. To solve this problem, HSIM^{plus} provides the Post-Layout Acceleration option (PLX), including an optimized hierarchical back-annotation technology (HBA).

Back-Annotation Without Post-Layout Acceleration

Back-annotation in HSIM and HSIM^{plus} is done from the net definitions of the DSPF/SPEF files. Refer to the *HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation* for a detailed description of the HSIM parameters required to set up back-annotation from DSPF/SPEF files. An example to illustrate back-annotation without the HSIM^{plus} PLX option is shown in Figure 2 on page 10.

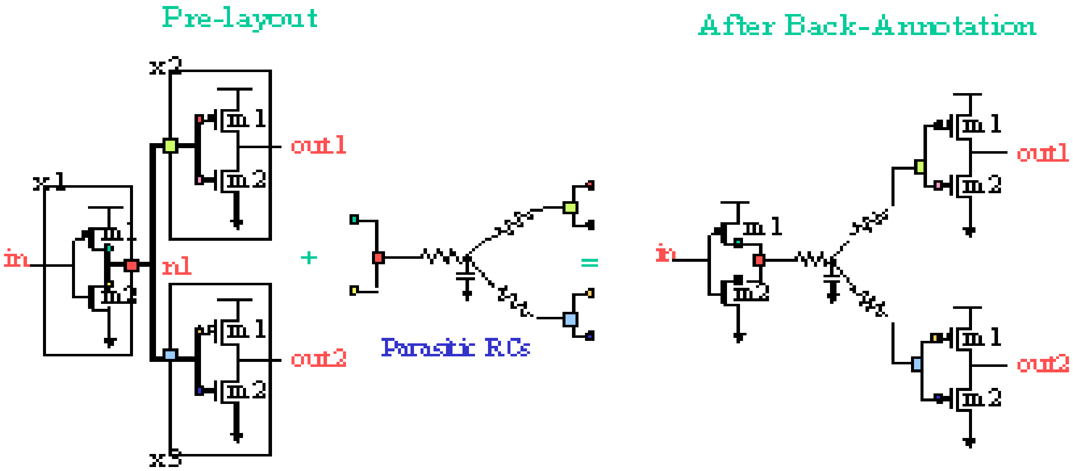


Figure 2 Back-Annotation Without the HSIM^{plus} PLX Option

The node being back-annotated is n1 in the DSPF file, which contains all the parasitic RCs for that node. In order to back-annotate the node n1, HSIM disconnects all the devices from the node n1 and reconnects them to the corresponding ports of the net. In doing this, HSIM has to flatten the circuit to bring all the net connections on the same hierarchy level. This may not be a problem when only a small number of nets are back-annotated, or there are no coupling capacitors in the back-annotation. In that case, HSIM may partially flatten the circuit, but the overall hierarchy is preserved. By maintaining the hierarchy in the circuit, HSIM can apply its hierarchical simulation methodology to achieve high simulation speed.

However, when the back-annotation involves many nets (full chip extraction), or there are a great number of coupling capacitors, the hierarchy is destroyed with many random connections between the nets. HSIM must then flatten virtually the entire circuit, such that the advantages of hierarchical simulation cannot be used and the simulation speed degrades significantly. When the power and ground nets are back-annotated, which almost every device in the circuit has a connection to, HSIM also must flatten the circuit to bring all the connections onto the same level of the hierarchy. This again results in a huge flat circuit to be simulated, thus making the simulation run times unacceptable in many cases.

As a result, the standard flat back-annotation approach can be efficiently applied only to either of the following:

- Block level post-layout simulations.
- Designs with low signal net coupling or low coupling through power nets.

When there is a need for full chip post-layout analysis, or when the power and ground nets are to be back-annotated, the HSIM^{plus} PLX option provides for hierarchical back-annotation to maintain the pre-layout circuit hierarchy in the presence of coupling, retaining memory efficiency, and simulation performance.

HSIM^{plus} Back-Annotation with Post-Layout Acceleration

With Post-Layout Acceleration, HSIM does not flatten the circuit in order to add the parasitic RCs. Applying a sophisticated optimization algorithm, HSIM distributes the parasitic RCs among different levels of hierarchy and sub-circuits. The hierarchy of the pre-layout netlist is completely preserved. The effect of hierarchical back-annotation in the HSIM^{plus} PLX option is shown in Figure 3. In the schematic example, the back-annotation of the power net with hierarchical back-annotation is shown.

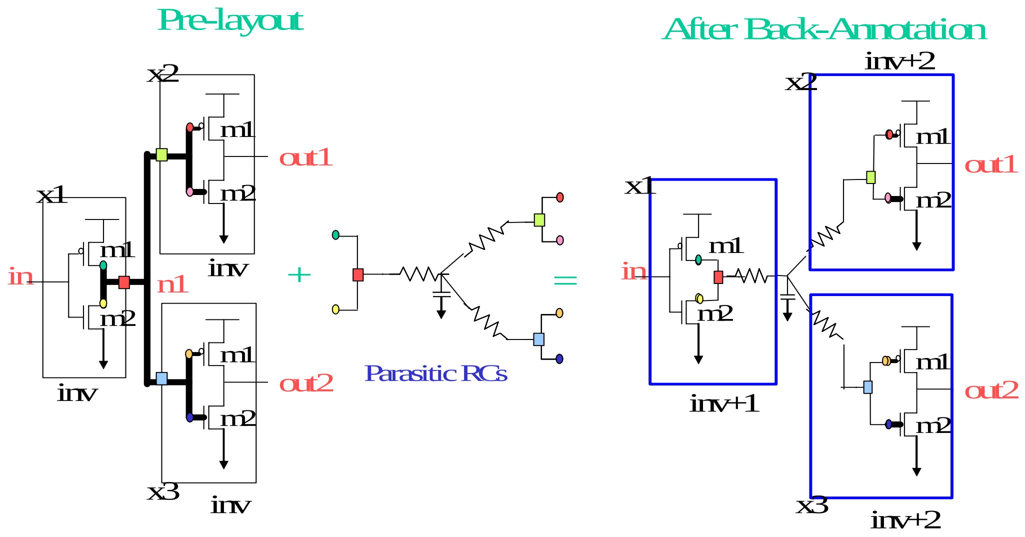


Figure 3 Effects of HSIM^{plus} PLX on Back-Annotation

In addition to saving the hierarchy of the circuit, the HSIM^{plus} PLX option processes all coupling capacitors using file operations. During these operations, the coupling capacitors are handled as ungrounded capacitors. Refer to the *HSIM Simulation Reference Manual: Chapter 6, Simulation Parameters*, for additional information on coupling capacitors. Many of the coupling capacitors may be split with no effect on the accuracy, thus the actual number of the capacitors that are added to the circuit is small. By doing filtering of the coupling capacitors on a file basis, HSIM HBA can accept virtually any number of coupling capacitors without big computer memory usage. The coupling capacitors being added to the circuit are also distributed over the hierarchy, so all the benefits of the HSIM hierarchical simulation will be fully utilized.

HSIMPFPLX

HSIMPFPLX is a flag that enables or disables post-layout acceleration and the HBA flow.

Syntax

In order to activate the HSIM^{plus} PLX option, use the syntax:

```
.param HSIMSPFPLX=1
```

where:

- HSIMSPFPLX=0 (the default) turns PLX off.
- HSIMSPFPLX=1 sets HBA processes incoming DSPF/SPEF file parasitic RCs.

HSIM^{plus} Structural Back-Annotation (SBA)

Structural back-annotation is designed to handle structural mismatches allowed by LVS between schematic and extracted netlists.

The SBA flow offers the following capabilities:

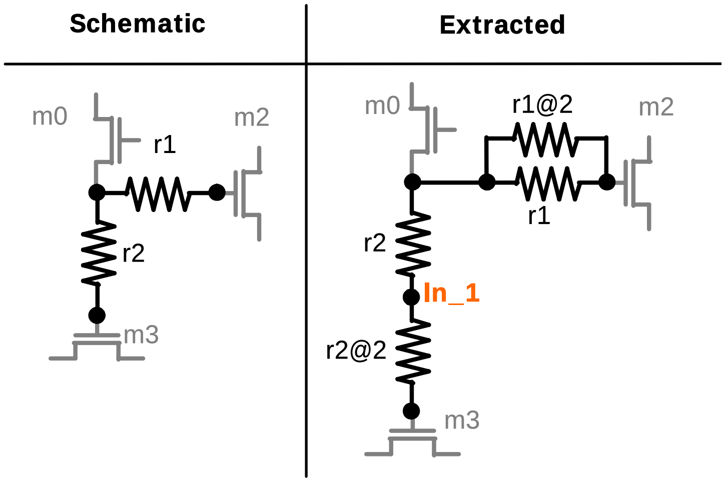
- SBA matches nets and devices by name and connectivity and therefore, it allows HSIM to identify structural differences between pre and post-layout netlists.
- SBA is performed before the simulation database is built and as a result, it can back-annotate structural differences, as well as devices with non-generic physical parameters.

- When the SBA flow is used, DPF back-annotation is not required and should not be used.
- SBA back-annotates structural differences and results in more accurate DSPF back-annotation with fewer warnings.

The following examples demonstrate the features of SBA.

1. SBA back-annotates functional resistors with extra post-layout nodes.

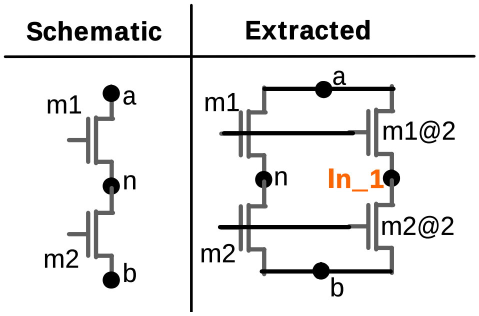
   In the figure below, the schematic netlist contains resistors, r1 and r2. In contrast, the extracted netlist contains two additional resistors, r1@2 and r2@2, and an extra node In_1.



   The SBA algorithm recognizes that schematic netlist is electrically equivalent to extracted one (in other words they are LVS clean) and subsequently back-annotates the extra In_1 node and the additional r1@2 and r2@2 devices into original schematic netlist.

2. SBA back-annotates complex finger devices.

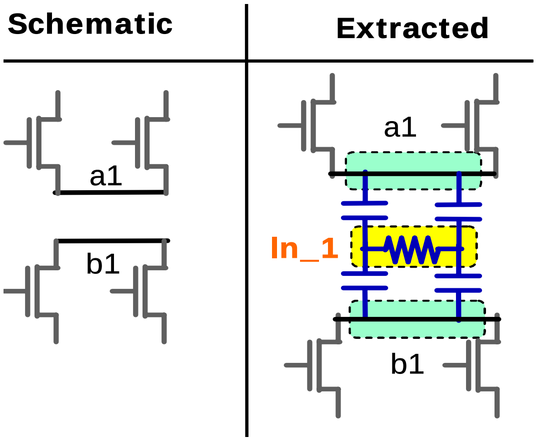
   In the figure below, the schematic netlist contains two MOSFETs, m1 and m2 in series. During layout, the m1 and m2 transistors were fingered, which resulted in an extracted netlist that contains two additional transistors, m1@2 and m2@2, as well as an extra node In_1.



The SBA algorithm matches these two structure and back-annotates the missing ln_1 net and the m1@2 and m2@2 devices from the extracted netlist.

3. SBA back-annotates post-layout fill in nets.

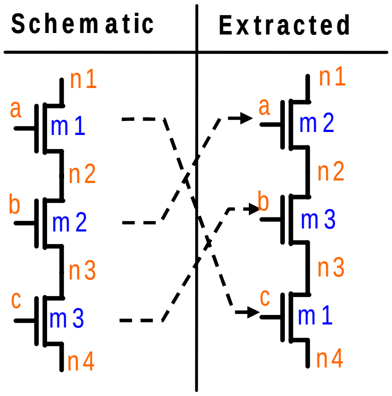
   Fill in nets are often inserted in the layout to planarize metal layers and reduce the risk of manufacturing failures. As a result, fill in nets appear only in the extracted netlist. In the figure below, net ln_1 is a fill in net that only appears in the extracted netlist, however, because ln_1 was placed in close proximity to nets a1 and b1, it is coupled to these nets through parasitic capacitance.



SBA identifies post-layout fill in nets and back-annotates them to the original schematic netlist.

4. SBA back-annotates swapped devices.

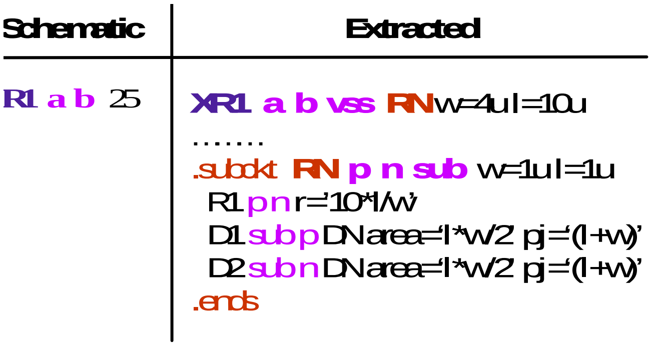
   Schematic devices can often be swapped or rotated during layout. The resulting extracted netlist is LVS clean, however, the device connections might differ. The figure below demonstrates the swapping of the devices m1, m2 and m3 between the schematic and extracted netlists.



SBA detects swapped devices and back-annotates them into the original schematic netlist.

5. SBA back-annotates sub-circuit or Verilog-A modeled devices.

   To model the nanometer effects of modern day designs, extracted netlists may contain devices that are modeled in Verilog-A or are wrapped inside of sub-circuits. In the figure below, the schematic netlists contains ideal resistor R1. However, during layout, the R1 resistor is implemented using NWELL. Furthermore, to accurately model the PN junction behavior of the NWELL resistor, it is extracted as sub-circuit instance that contains a resistor and two diodes.



Schematic	Extracted
R1 a b 25	XR1 a b vss RN w=4u l=10u .... .subckt RN p n sub w=1u l=1u R1 p n r=’10*l/w’ D1 sub p DN area=‘l*w/2’ pj=‘(l+w)’ D2 sub n DN area=‘l*w/2’ pj=‘(l+w)’ .ends

   SBA can be configured using the HSIMSBAPARAM parameter to recognize the XR1 extracted instance as a resistor and back-annotate it to the schematic netlist.

Below is a diagram that summarizes the SBA flow.

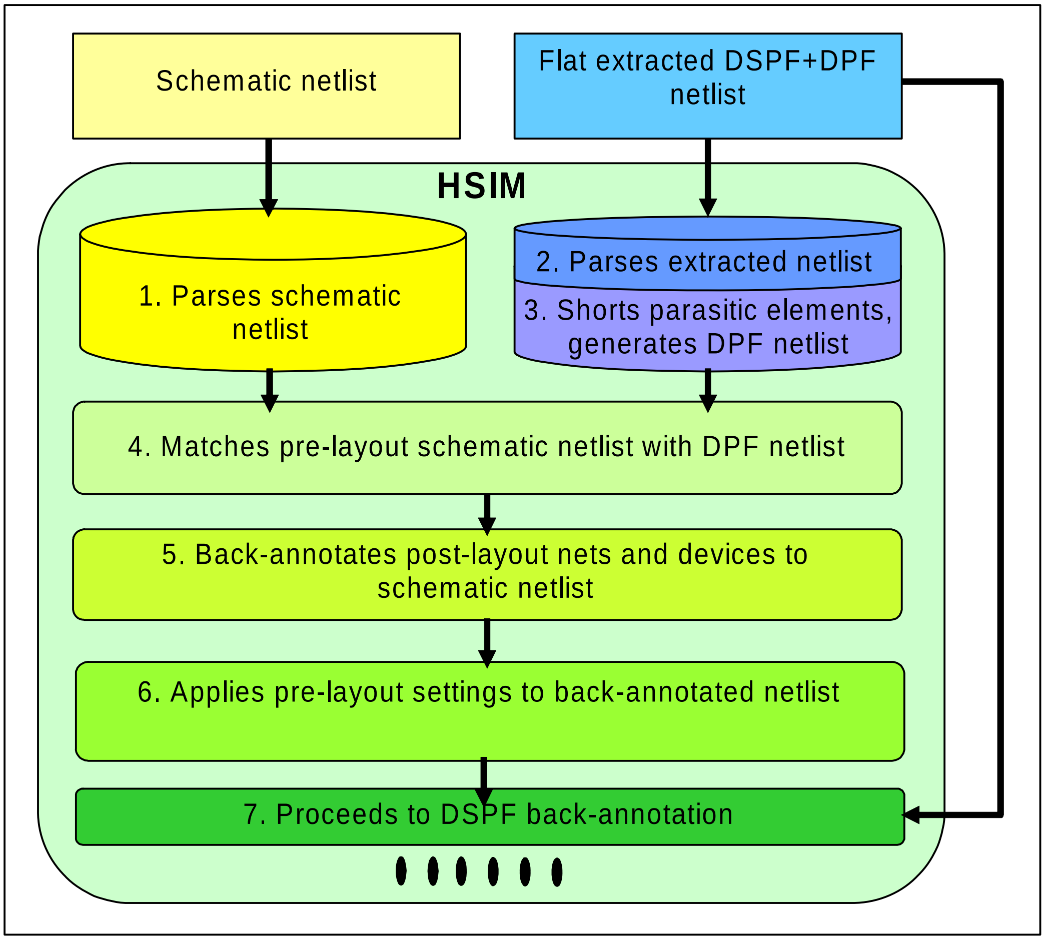

Schematic netlist
Flat extracted DSPF+DPF netlist
HSIM
1. Parses schematic netlist
2. Parses extracted netlist
3. Shorts parasitic elements, generates DPF netlist
4. Matches pre-layout schematic netlist with DPF netlist
5. Back-annotates post-layout nets and devices to schematic netlist
6. Applies pre-layout settings to back-annotated netlist
7. Proceeds to DSPF back-annotation

Figure 4 Structural Back-Annotation Flow

As shown in Figure 4, the SBA flow works as follows:

1. HSIM parses the pre-layout schematic netlist.
2. Parses the extracted DSPF+DPF netlists.
3. Shorts all parasitic RC's from the SPF section of the extracted netlist and internally generates flat extracted DPF netlist that contains only functional devices.
4. Matches pre-layout schematic netlist with generated flat DPF netlist. The netlist-to-netlist comparison starts with top level anchor nodes and propagates through entire design. As a result, nets and devices are matched by name and connectivity.

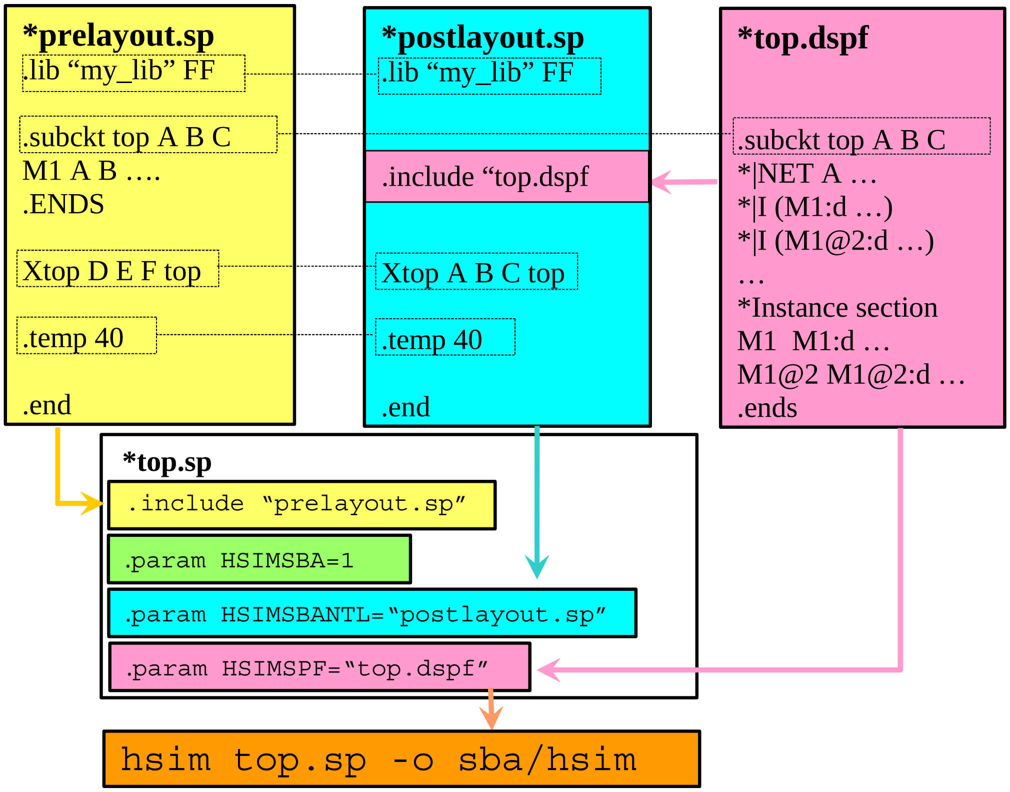
5. After the netlist-to-netlist comparison is complete, HSIM back-annotates the mapped devices from the extracted DPF netlist into the original pre-layout schematic netlist.
6. Once extracted devices and nets are back-annotated into the original pre-layout schematic netlist, the pre-layout accuracy settings (.print, .measure and .IC) are re-applied to the structurally back-annotated netlist.
7. If the HSIMSPF parameter is set, HSIM performs DSPF back-annotation.
8. Finally, HSIM builds the netlist database, performs DC initialization and proceeds to transient simulation.

Note:

SBA requires a clean LVS database.

Invoking SBA

To invoke SBA, set HSIMSBA=1 in the top level netlist. In addition, the HSIMSBANTL parameter is also required. The HSIMSBANTL parameter must point to a postlayout.sp file that has to be generated manually. The postlayout.sp file should be a self-contained SPICE netlist with the instance of extracted subcircuit, device libraries, and the appropriate temperature setting as demonstrated in the diagram below.



```
*prelayout.sp
.lib "my_lib" FF

.subckt top A B C
M1 A B ….
.ENDS

Xtop D E F top

.temp 40

.end
```

```
*postlayout.sp
.lib "my_lib" FF

.include "top.dspf

Xtop A B C top

.temp 40

.end
```

```
*top.dspf

.subckt top A B C
*|NET A …
*|I (M1:d …)
*|I (M1@2:d …)
…
*Instance section
M1  M1:d …
M1@2 M1@2:d …
.ends
```

```
*top.sp
.include "prelayout.sp"
.param HSIMSBA=1
.param HSIMSBANTL="postlayout.sp"
.param HSIMSPF="top.dspf"
```

```
hsim top.sp -o sba/hsim
```

Figure 5 Invoking SBA

SBA Parameters

HSIMSBA

HSIMSBA is a flag that enables or disables the SBA flow.

Syntax

```
.param HSIMSBA=0|1
```

If HSIMSBA is set to 0, all SBA parameters are ignored and the SBA is disabled. If it is set to 1, the SBA flow is enabled. The default is 0.

Note:

SBA requires the HSIM-PLX license feature.

HSIMSBANTL

HSIMSBANTL is used to specify the post-layout netlist required for the SBA flow as shown in Figure 5.

Syntax

```
.param HSIMSBANTL=<filename>
```

Where <filename> is the name of the post-layout netlist file to use. The post-layout netlist should contain the following:

- The top level instance of the extracted subcircuit.
- An .INCLUDE statement of the extracted subcircuit in DSPF+DPF or DPF format.
- A device library or models that are used in the extracted subcircuit.
- A .TEMP setting, otherwise HSIM will use default temperature when parsing post-layout netlist.
- An .END statement.

HSIMSBAPARAM

The HSIMSBAPARAM parameter is used to help match sub-circuit instances (parse) for the SBA flow.

Syntax

```
.param HSIMSBAPARAM="sub=<subckt> parse {<device>}"
```

Syntax Description

`<subckt>`

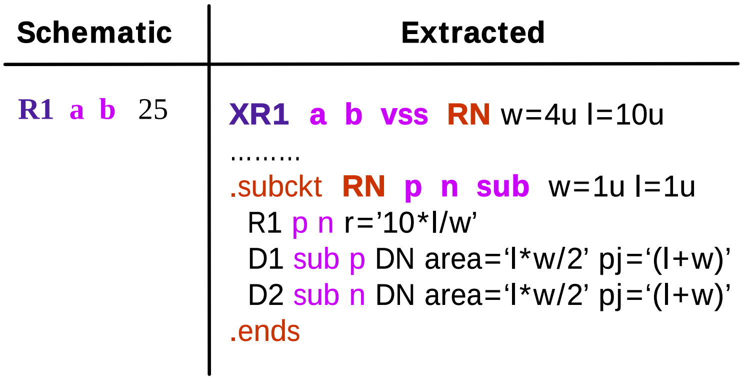
The name of the wrapper subcircuit or Verilog-A module that requires special treatment during SBA.

`<device>`

The discrete device that is substituted by SBA during matching in place of wrapper subcircuit or Verilog-A module.

Example One

In the example below, the resistor R1 in the schematic netlist was extracted as a subcircuit instance XR1.



Schematic	Extracted
R1 a b 25	XR1 a b vss RN w=4u l=10u ……… .subckt RN p n sub w=1u l=1u R1 p n r='10*l/w' D1 sub p DN area='l*w/2' pj='(l+w)' D2 sub n DN area='l*w/2' pj='(l+w)' .ends

The R1 and XR1 are entirely different elements. R1 is a discrete resistor with two terminals. XR1 is an instance of a subcircuit "RN" with three connections. Subcircuit RN has one resistor and two diodes. For SBA to map the schematic "R1" resistor with the extracted "XR1" instance HSIMSBAPARAM must be set in the post-layout netlist as follows:

```
.param HSIMSBAPARAM="sub=RN parse {Rext p n 10}"
```

This parameter instructs SBA to map any instance of subcircuit "RN" in the extracted netlist to a discrete resistor "Rext p n 10". Observe that the name and the value of Rext resistor are irrelevant since they will be used only to match the type of the device. However, to establish proper connectivity, the node names "p" and "n" of "Rext" resistor must match the port names in the ".SUBCKT RN p n sub" definition.

After "R1" and "XR1" devices are matched by SBA, the "XR1" instance from the extracted netlist is back-annotated in place of the ideal "R1" schematic resistor.

Example Two

In this example, the subcircuit instance XR1 in the schematic netlist was extracted as an ideal R1 resistor.

Schematic	Extracted
XR1 a b vss RN w=4u l=10u ……… .subckt RN p n sub w=1u l=1u R1 p n r=’10*l/w’ D1 sub p DN area=‘l*w/2’ pj=‘(l+w)’ D2 sub n DN area=‘l*w/2’ pj=‘(l+w)’ .ends	R1 a b 25

For SBA to map schematic the "XR1" instance with the ideal extracted "R1" resistor, HSIMSBAPARAM must be set in the pre-layout netlist as follows:

```
.param HSIMSBAPARAM="sub=RN parse {Rsch p n 10}"
```

After SBA matches the "XR1" instance in the schematic netlist with the extracted "R1" resistor, it back-annotates the extracted "R1" resistor. The final simulation netlist contains resistor "R1 a b 25".

Example Three

In this example, the XM1 subcircuit instance in the schematic netlist was extracted as the NHC subcircuit instance.

Sche	Extra
XM1 a b c d NCH w=0.45u l=0.18u ... .subckt NCH d g s b w=0 l=0	XM1 a b c d sub NCHEX w=0.5u l=0.2u ... .subckt NCHEX d g s b sub w=0 l=0 .MB d g s b NMOS w=w l=l

XM1 in the schematic netlist is an instance of the NCH subcircuit with 4 terminals, whereas in the extracted netlist it is an instance of the subcircuit NCHEX with 5 terminals. For SBA to map the XM1 schematic instance to the extracted one, HSIMSBAPARAM must be set in both netlists as follows:

Schematic:

```
.param HSIMSBAPARAM="sub=NCH parse {Mx d g s b N w=0.3u l=0.18u}"
```

This parameter instructs SBA to map any instance of NCH subcircuit to a discrete MOSFET device, Mx d g s b N w=0.3u l=0.18u. Observe that the device name Mx and device model N, as well as values of w and l, are arbitrary since they are used only to match the type of devices. However, to establish proper connectivity, the node names d, g, s and b of Mx must match the port names in the .subckt NCH d g s b definition.

Extracted:

```
.param HSIMSBAPARAM="sub=NCHEX parse {Mx d g s b N w=0.3u l=0.18u}"
```

This parameter instructs SBA to map any instance of the NCHEX subcircuit to a discrete MOSFET device, Mx d g s b N w=0.3u l=0.18u. Observe that the device name Mx and device model N, as well as values of w and l, are arbitrary since they are used only to match the type of devices. However, to establish proper connectivity, the node names d, g, s and b of Mx must match the port names in the .subckt NCHEX d g s b sub definition. The extra sub port in the subckt definition is ignored.

After these mapping steps, a correspondence between the XM1 instances in schematic and extracted netlists is established, and the extracted XM1 instance is back-annotated in place of the schematic XM1 instance.

HSIMSBAMSGLEVEL

HSIMSBAMSGLEVEL controls the number of warnings printed to the log file during SBA back-annotation.

Syntax

```
.param HSIMSBAMSGLEVEL=0|1|2|3|4|5
```

Syntax Description

HSIMSBAMSGLEVEL can be set to any integer between 0-5. If it is set to 0, no SBA warnings will be reported. When set to 5, detailed SBA warnings are reported to the HSIM log. The default value is 1.

HSIMSBAMSGLIMIT

HSIMSBAMSGLIMIT sets the limit on the total number of warning messages issued during the matching stage of SBA.

Syntax

```
.param HSMSBAMSGLEVEL=<value>
```

Syntax Description

<value> is an integral value that specifies the absolute limit on the number of warnings generated by SBA. The default value is 500. This setting does not override the global warning limit settings (like '.opt warnlimit'). SBA stops generating warning messages as soon as either of these limits is reached.

HSIMSBAHIERID

HSIMSBAHIERID is used in the post-layout file to set the proper hierarchical separator in the device name.

If we have the following device In the extracted netlist:

```
*Instance Section
M_X1/M3#1 A B C D NMOS ….
```

The following setting should be used in the post-layout netlist:

```
.param HSIMSBAHIERID="/"
```

HSIMSBASFX

HSIMSBASFX is used in the post-layout file to set the proper finger delimiter in the device name.

If we have the following device In the extracted netlist:

```
*Instance Section
M_X1/M3#1 A B C D NMOS ….
```

The following setting should be used in the post-layout netlist:

```
.param HSIMSBASFX="#"
```

HSIMSBAPFX

HSIMSBAPFX is used in the post-layout file to set the proper prefix in the name of extracted devices.

If we have the following devices In the extracted netlist:

```
*Instance Section
M_X1/M3#1 A B C D NMOS …
R_X1/R1   F N 100
X_X1/XR2  K L RN W=1 L=10 …
```

The following setting should be used in the post-layout netlist:

```
.param HSIMSBAPFX="M_"
```

SP2DSPF Utility

Generating a DSPF File From the Flat Extracted Netlist

Back-annotation of parasitic RCs from detailed standard-parasitic format[1] (DSPF) or standard-parasitic extended format[2] (SPEF) files form the core technology for HSIM's signal integrity and power/signal net reliability analysis.

Some extraction tools have problems outputting their results as DSPF files. These tools extract flat SPICE netlists with parasitic RCs so that making flow changes that permit the use of DSPF files may require additional effort from HSIM users. In HSIM, simulating large flat extracted netlists is much less efficient when compared to simulating the hierarchical netlist back-annotated from a DSPF file. Moreover, HSIM's reliability features can only be used if back-annotation is applied.

The SP2DSPF utility generates the following files from a pre-layout netlist and flat or hierarchical extracted netlist with parasitic RCs:

- DSPF file
- DPF (Device Parameter Format) file (optional)

Non-parasitic devices from the extracted netlist, such as MOSFETs, are matched with their pre-layout counterparts and can be output in the DPF file. The resulting DSPF and DPF files can then be used for back-annotating the pre-layout netlist.

Running SP2DSPF

To run SP2DSPF, use the following syntax:

```
hsim -sp2dspf [<parameter file>] [<parameter setting> ...]
```

A list of parameters for the current run are contained in either of the following:

- Parameter file: For parameters used in numerous runs.
- Specified on the command line: For parameters with one-time or limited usage.

  Note:

  Parameters read from the file or from the command line are identical.

The following example shows the hsim -sp2dspf syntax that refers to a parameter file.

```
hsim -sp2dspf p1_set.txt -anan gnd
```

Each parameter setting is a keyword prefixed with a dash (-) character and may be followed by parameter value(s). A single or multiple parameters can be inserted in a single syntax line when creating a parameter file.

SP2DSPF Utility Parameters

-pre

```
-pre <file name>
```

`-pre` is a required parameter that specifies a pre-layout netlist.

-fpre

```
-fpre <format> (default: hspice)
```

`-fpre` specifies the pre-layout netlist format where `format` is the netlist format name (for example, `spectre`). The list of available formats is the same as for HSIM. `hspice` is the default value.

-pretop

```
-pretop <subcircuit name>
```

`-pretop` specifies which pre-layout netlist sub-circuit is used as the top-level sub-circuit. The top-level netlist instance is the default used as the top-level sub-circuit.

-post

```
-post <file name>
```

`-post` is a required parameter that specifies the name of the extracted netlist.

-fpost

```
-fpost <format> (default: hspice)
```

`-fpost` specifies the extracted netlist format where `format` is the netlist format name (for example, `spectre`). The list of available formats is the same as for HSIM. `hspice` is the default value.

-posttop

```
-posttop <cell name>
```

`-posttop` specifies which extracted netlist sub-circuit should be used as the top-level sub-circuit. The default is the top-level netlist instances which are used as the top-level sub-circuit.

-an

```
-an <node name> <node name> [<node name> <node name> ...]
```

`-an` specifies anchor node pairs used as starting points for the matching process. The pairs have the following form:

- First Node: The first node of each pair is a pre-layout netlist node.
- Second Node: The second node is an extracted netlist node.

Only top-level nodes are allowed and at least one pair of anchor nodes is required. `-an` can be specified multiple times, as shown in the following example:

```
-an           0              gnd! pwr vdd
-an           vcc            vss
-anan <node name> [<node name> ...] (default: *)
```

-anan

`-anan` specifies anchor nodes with identical names in pre-layout and extracted netlists. Specifying `-anan node_name` is equivalent to specifying `-an node_name node_name`, for example, `-anan` provides a shorter and more convenient way to specify anchor nodes when they are named identically in both input netlists.

Additionally, `-anan` supports wildcard characters such as (*) and (?). When wildcards are used as a node name in `-anan`, the anchor node selection is performed in accordance with the following:

1. All top-level nodes whose names match the wildcard are found in both pre-layout and extracted nelists.
2. The names of pre-layout and extracted nodes are compared and every pair consisting of a pre-layout and an extracted node with identical names is used as a pair of anchor nodes.

The following example shows a wildcard used with the `-anan` parameter.

```
-anan *
```

`-anan *` is the default for `-anan`. In this example, the program searches all top-level nodes in both the pre-layout and extracted netlists for nodes with identical names. These nodes are then used as anchor nodes for matching process.

-out

```
-out <file name prefix> (default: hsim -sp2dspf)
```

`-out` specifies the default output file name prefix. If the name is concatenated, it will have the following properties:

- log extension: it is used as the log file name
- .dpf: DPF output
- .dspf: DSPF

  Note:

  > If different names are explicitly assigned using the -outdpf and -outdspf parameters, -out is overridden for these parameters.

-dpf

```
-dpf on/off (default: off)
```

Enables/disables DPF file output.

-outdpf

```
-outdpf <file name>
```

`-outdpf` specifies output DPF file name and enables DPF file output, provided it is not disabled explicitly by `-dpf` parameter. `-outdpf` has no effect if DPF output is disabled by explicit `-dpf` parameter. If `-outdpf` is not specified, DPF file name is built in accordance with output file name prefix setting `-out`. If output DPF file name is exactly the same as output DSPF file name, the output is directed to the DSPF file to form its instance section. Otherwise, a standalone DPF file is generated.

-dspf

```
-dspf on/off (default: on)
```

`-dspf` enables or disables DSPF file output.

-outdspf

```
-outdspf <file name>
```

`-outdspf` specifies the output DSPF file name and enables DSPF file output, provided it is not disabled explicitly by `-dspf` parameter. If `-outdspf` is not specified, the DSPF file name is built in accordance with output file name prefix setting in `-out`.

-pinports

```
-pinports on/off (default: on)
```

`-pinports` specifies whether SP2DSPF should report all top post-layout top-level ports as pins in the resultant DSPF file. This option is only effective when post-layout top-level subcircuit is explicitly specified by `-posttop` parameter.

-ms

```
-ms <subcircuit name> <subcircuit name> [<subcircuit name>
   <subcircuit name> ...]
```

`-ms` specifies isomorphic sub-circuit pairs in both pre-layout and extracted netlists.

- The first sub-circuit in the pair is a sub-circuit from the pre-layout netlist.
- The second sub-circuit in the pair is a sub-circuit from the extracted netlist.

This initial matching of sub-circuits is used by `hsim -sp2dspf` as a hint.

Caution!

> Use `-ms` with CAUTION. Providing initial matching information to the program can greatly increase its performance and reduce its peak memory requirements, however if the initial matching is wrong, caused by different sub-circuits being specified as matched, it may take MORE time and memory for the program to determine the correct answer.

There is usually no need to use `-ms` because the hierarchies of the pre-layout and extracted netlists are substantially different; the pre-layout netlist is mostly hierarchical while the extracted netlist is flat. However, for specific gate level circuits in ASIC designs, `-ms` may help, such as when an ASIC has a single (top) level with many instances of the cells. Netlists for cells with parasitic RCs are usually available, so the extracted netlist has additional parasitics only on the top-level. In this case,`-ms` can be used to set up the initial correspondence between the cells as shown in the following example:

```
-ms inv inv
-ms nand2 nand2
-ms nor2 nor2
```

-mm

```
-mm <model name> <model name> [<model name> <model name> ...]
```

`-mm` specifies pairs of device models that should be considered to be the same in both pre-layout and extracted netlists.

- The first name in the pair is a device model from the pre-layout netlist.
- The second name in the pair is a device model from the extracted netlist.

-opt outnf

```
-opt outnf <reverse|prefix> (default: prefix)
```

`-opt outnf` specifies output DPF/DSPF device naming convention as follows:

reverse

> In reverse mode, device names are built in accordance with reverse hierarchy (bottom-to-top) convention; e.g., `-m12.x32.x4.xcell`.

prefix

In prefix mode device names are built in accordance with straight hierarchy (top-to-bottom) convention and prepended with corresponding device type prefix; e.g., `m_xcell.x4.x32.m12`. The separator character used between the device type prefix and the actual hierarchical name can be changed using the `-opt outprefc` parameter.

-opt outprefc

```
-opt outprefc <prefix character>|none> (default: '_')
```

`-opt outprefc` specifies a separator character, which is used in extracted device names between device type prefix and the hierarchical name of the device. Refer to the `-opt outnf` parameter above for additional information. `-opt prefc` has no effect in modes other than `-opt outnf prefix`.

-opt outhierc

```
-opt outhierc <separator character> (default: '.')
```

`-opt outhierc` specifies instance name separator character used in hierarchical names in the output files.

-opt outsubc

```
-opt outsubc <separator character> (default: '@')
```

`-opt outsubc` specifies the subscript separator character used in fingered device names in output files.

-opt serial

```
-opt serial <any|flip|none> (default: any)
```

`-opt serial` controls processing of parallel-serial transistor groups. `any` allows transistors in serial chains to be permuted in an arbitrary manner. `flip` requires that transistors in all serial chains are arranged in the same order, but some chains can be reversed. `none` disables processing of serial transistor chains completely.

-opt capnet

```
-opt capnet <empty|node|pins> (default: node)
```

`-opt capnet` controls the format in which single-ground-capacitor nets are written to the DSPF file.

empty mode

> In empty mode only *|NET statements with corresponding capacitance values are written. Neither net pin, sub-nodes, nor any other devices are reported for the net.

node mode

> The node mode writes a single grounded capacitor element to DSPF, with no net pin/sub-node list, in addition to the information written in the empty mode.

pins mode

> The pins mode creates a complete net pin/sub-node list for each single-ground-capacitor net in addition to the information written in the node mode.

-opt dupcc

```
-opt dupcc on/off (default: off)
```

-opt dup`cc` specifies whether coupling capacitors connected between two parasitic nets should be reported in the resultant DSPF file as follows:

on

> Both nets.

off

> Default; only one net.

-opt rpref

```
-opt rpref <prefix string> (default: "_")
```

`-opt rpref` specifies the prefix used in the resultant DSPF file in parasitic resistor names. Resistor names are built as follows: r<prefix><numerical index>.

-opt ccpref, -opt gcpref

```
-opt ccpref <prefix string> (default: "_c")
-opt gcpref <prefix string> (default: "_g")
```

`-opt ccpref` and `-opt gcpref` specify prefaces used in the resultant DSPF file in coupling an ground capacitor names respectively. Capacitor names are built as follows: c<prefix><numerical index>.

-opt vsr

```
-opt vsr <resistance value> (default: 1e-3)
```

`-opt vsr` specifies the very small resistance (VSR) value used to implement connectivity whenever there is a need to introduce an additional net or instance pin in a DSPF file. For example, if two nodes that would be essentially shorted are required, then the resistor of this small value between the two nodes will be generated.

References

[1] A Cadence® developed format based on SPF. DSPF is similar to Spice but includes comments and a structure making it easier to organize netlist information into the original circuit with additional information used by RC trees.

[2] SPEF has extended capability and a small format compared to SPF and is defined in the Delay-Calculation-System (DCS) standard of the Open Verilog International.

3

Power Net Reliability Analysis (PWRA)

Provides information on performing IR drop and electro-migration analysis of power nets with the HSIM^{plus} PWRA option.

Power nodes in simulated circuits are considered to have constant voltage. In reality, resistive networks between the power node pad and the devices always exist. When devices draw current from the power source, the voltage or IR drop appears on the device terminals connected to power nodes. If significant, this IR drop may affect the performance of the circuit; such as the delay value.

The IR drop value depends on the value of the resistive path from the power pad, to the device connection. The current flowing through each power net RC network resistor may cause electro-migration (EM) problems in the circuit if the current density is greater than a specified threshold.

HSIM^{plus} provides power net reliability analysis (PWRA) to address these problems.

- Uses a DSPF file representation of the power net to calculate all the pad-to-pin resistances. The number of resistors in the power nets of modern circuits makes it impossible to simulate the circuit with back-annotated power nets as is, so power net RC reduction must be applied.
- Accepts power net RC networks by back-annotating the resistors and capacitors from a DSPF file. Refer to the *HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation* for detailed information on signal node back-annotation parameters.
- Simulates the circuit with the reduced power net RC network added to the circuit devices, providing dynamic IR drop on circuit behavior.
- Calculates the maximum IR drop at each connection between the power net and circuit devices.

- Calculates the maximum, average, or RMS current density through each power net resistor and compares it with the threshold for the purpose of electro-migration analysis.
- Displays the violation map highlighting locations where IR drop or current density exceeds the threshold.

HSIM^{plus} performs power net reliability analysis in two phases:

- Phase I: A reduced power net is back-annotated to the circuit and transient simulation is performed. This simulation determines the impact of dynamic voltage drop on circuit behavior and captures degradation of delay or characteristics induced by IR drop.
- Phase II: Node voltage and branch current density values are computed for the raw, unreduced power net and graphical visualization data is created.

Power Net Reduction

Specifics of Power Net Back-Annotation

Nanometer system-on-chip (SoC) designs incorporate incredibly complex power distribution systems in an attempt to minimize the resistance from power pads to transistor contacts in an area- and layer-efficient manner. This complexity results in DSPF files containing millions to hundreds of millions of resistors, each typically having a very small value.

HSIMPOSTL

To improve post-layout simulation performance, HSIM^{plus} signal net reduction reduces these networks to a near-equivalent circuit with a small and controllable loss in accuracy. This produces significant simulation throughput and memory efficiency benefits. Power net RC reduction is different from signal net RC reduction which is controlled by HSIMPOSTL. Refer to the *HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation*.

HSIMSPFPWNET

Power net RC reduction is applied only to power nets. By default, HSIM^{plus} PWRA defines a power net as an RC extracted net from a DSPF file, connected to a constant voltage supply. Setting HSIMSPFPWNET to a positive integer invokes power net reduction, as shown below:

```
.param HSIMSPFPWNET=5|4|3|2|1|0|-1
```

HSIMSPFPWNET=-1 (default)

Generic RC reduction algorithms controlled by HSIMPOSTL are applied to both signal and power nets.

HSIMSPFPWNET=0

Only performs parallel-series power reduction.

HSIMSPFPWNET=1,2,3

Provides varying speed and accuracy trade offs between memory usage and simulator precision. The lower the parameter value, the less aggressive the reduction.

HSIMSPFPWNET=4

Option 4 is the most aggressive power net reduction mode, applying the greatest effort and strongest reduction techniques. It has the following characteristics:

- Most significantly reduces the number of power net parasitic components
- Does not preserve the underlying topology of the power network
- Maintains a fairly close correlation for all impedances between the power pads and the device source or drain connections.

HSIMSPFPWNET=5

The power network is de-coupled.

HSIMPWNAME

To explicitly specify which nets should be treated as the power nets, use HSIMPWNAME as shown below:

```
.param HSIMPWNAME=vdd
.param HSIMPWNAME=vss*
```

In this example, HSIM^{plus} treats nets that match "VDD" and "VSS*" patterns as power-nets. This means that HSIMSPFPWNET reduction and power net reliability analysis applies only to specified "VDD" and "VSS*" power nets.

Note:

> This command should be used in conjunction with the PWRA option only (HSIMPWRA=1). Without this option, any net specified in HSIMRANET is considered a signal net, which may lead to unexpected warning and simulation results.

Use the following syntax for VSRC nets:

```
.param HSIMPWNAME="pwnet_name"
```

where *pwnet_name* is in the power net name. You can use wildcard characters in the name.

Use the following syntax for internal power nets:

```
.param HSIMPWNAME="pwnet_name [intvref=vref]
    [source=vsrcnet]"
```

where *pwnet_name* is in the power net name, *vref* is the Internal reference voltage used for PWRA analysis to compute IR-drop for internal power nets, and *vsrcnet* is the name of the parent external power net that connects to constant voltage source and is separated from internal power net by pre-layout resistors, inductors or MOSFET transistors. This field is used for guiding the HSIMPWTRACERL command.

When a net is connected to constant voltage source, HSIMPWNAME helps HSIM to distinguish power nets from signal nets. By default, if HSIMPWNAME is not set, HSIM^{plus} considers all nets connected to constant voltage sources as power nets. To explicitly specify which nets should be processed as power nets, specify:

```
.param HSIMPWNAME="vdd"
```

```
.param HSIMPWNAME="vss*"
```

In the previous example HSIM^{plus} processes nets that match VDD and VSS* patterns as power-nets and all other nets as signal nets. This means that HSIMSPFPWNET reduction and power net reliability analysis applies only to specified VDD and VSS* power nets.

To enable processing of internal power nets, you need to set HSIMPWNAME in conjunction with either HSIMPWTRACERL or HSIMSPFNETPPIN. Otherwise all internal power nets are processed as signal nets. See more information about internal power nets, see HSIMPWTRACERL on page 98.

Note:

> The HSIMPWNAME command is applicable when performing PWRA analysis (HSIMPWRA=1) or enabling power net reduction (HSIMSPFPWNET=0-5).

HSIMSPFPWRMIN

HSIMSPFPWRMIN prefilters small resistors in power nets. Resistors with values less than HSIMSPFPWRMIN are deleted during DSPF file parsing. The deleted resistor terminals become electrically equivalent.

Syntax

```
HSIMSPFPWRMIN=0.01 [default]
```

Power nets can be incorporated into the circuit by back-annotation from the DSPF files with HSIMSPF as defined in the *HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation*. The default value is 0.01ohm. A value of 1e-9ohm is the minimum that this parameter accepts. Values smaller than 1e-9ohm are reset to1e-9ohm.

HSIMSPFNETPIN

The power net should contain the power supply connections identified with the DSPF *|P statement, to which external VDD and GND connections are made. However, if PWRA is being used to simulate a core or IP block, not the full-chip circuit, the extracted DSPF may not contain the *|P statements. In this case, a temporary net pin can be specified using HSIMSPFNETPIN. When this parameter is specified, PWRA connects power voltage sources to a sub-node or instance pin in the DSPF power net, which is declared as a temporary net pin.

Syntax

```
HSIMSPFNETPIN="power_net powernet_subnode"
```

Arguments

power_net

> Power net name that adds a net pin.

powernet_subnode

> Sub-node name that serves serve as a new net pin.

Example

```
.param HSIMSPFNETPIN=”VDD VDD:1127”
```

HSIMTMPDIR

```
HSIMTMPDIR="directory name"
```

While processing power nets, HSIM^{plus} saves some data in temporary files. By default, those files are created in the current directory. The location of the temporary files can be changed by setting HSIMTMPDIR to any other directory as shown in the following syntax example:

```
.param HSIMTMPDIR='/usr/tmp'
```

To increase file operation performance and to avoid over-the-network access, set the location for temporary files on the local disk drive.

HSIMSPFPWFLAT

Note:

This parameter is now obsolete. If you are using it, replace it with the parameter HSIMSPFTLV defined below.

HSIMSPFTLV

The HSIMSPFTLV parameter is required when the extracted DSPF file is hierarchical (contains .subckt definition) and HSIM reliability analysis is enabled. HSIMSPFTLV instructs HSIM to handle extracted data in the global design context. As a result HSIM generates one top-level view (TLV) SPF file: `<hsim_output_prefix>_tlv.spf.gz`. The new TLV SPF file is placed in the same directory with all other HSIM output files.

Syntax

```
.param HSIMSPFTLV=<0|1>
```

Syntax Definitions

HSIMSPFTLV=0

Heirarchical SPF files are processed as is. This is the default.

HSIMSPFTLV=1

Heirarchical SPF files are abstracted to a top-level view.

When enabled, the HSIMSPFTLV parameter can handle both power and signal nets for all types of post-layout netlists with hierarchical DSPF: single-instance of single-level heirarchical DSPF, multiple-instances of single-level heirarchical DSPF, single-instance of multilevel hierarchical DSPF, and multiple-instances of multilevel hierarchical DSPF.

Note:

To use the HSIMSPFTLV parameter, at least one of the following must apply:

Power Net Reliability Analysis (PwRa) is enabled (HSIMPWRA=1).

Signal Net Reliability Analysis (SigRa) is enabled (HSIMSIGRA=1).

If neither PwRa or SigRa are enabled, the HSIMSPFTLV parameter checks out a PLX license, unless the PLX feature is already checked out during the simulation by some other option, such as HSIMSPFPLX or HSIMSBA.

Package Modeling

In addition to the extracted power net, it is sometimes necessary to model the package pin and related circuit, including bond wire resistance and inductance. These package models are not present in the pre-layout netlist or in the extracted DSPF. They may however, be described separately using SPICE syntax. HSIM can add packaging models using HSIMSPFNETIPIN.

HSIMSPFNETIPIN

HSIMSPFNETIPIN is used to add new instance pin connections to an SPF net. The syntax is:

```
.param HSIMSPFNETIPIN=”<net_name> <connect_point> \
   <new_instance_pin>”
```

Syntax Definitions

<net_name>

The SPF net name.

<connect_point>

Can be a sub-node, instance pin, or pin of the SPF net.

<new_instance_pin>

The instance pin name that needs to be connected to the connect_point. For example: .param HSIMSPFNETIPIN=”vdd vdd:3 iv1:p”; that connects the positive terminal of current source iv1 to sub-node vdd:3 after SPF back-annotation. The current source iv1 must be specified in the netlist file.

To illustrate this methodology, Example 1 provides a simple example of a pre-layout statement.

Example 1

```
* prelayout.sp
.param HSIMSPF=vdd.dspf
VVDD VDD 0 1.8
MP A B VDD VDD p
...
```

and in the `vdd.dspf` file, the net VDD is specified as:

```
*|NET VDD 10ff
*|P (VDD_1 X 0.0)
*|P (VDD_2 X 0.0)
*|I (MP:S MP S X 0ff)
R1 VDD MP:S 10.0
R2 VDD_2 MP:S 10.0
```

Figure 6 on page 42 shows the resulting post-layout netlist in which the `VDD_1` and `VDD_2` pins are merged to the `VDD` node.

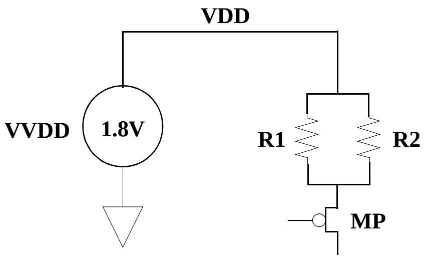


Figure 6 Post-layout Netlist Circuit

To model the power net with packaging effects, the pads are connected to separate voltage sources with 1.79V and 1.81V values. This is accomplished via package instances defined by the `rlc` sub-circuit, instead of connecting the VDD_1 and VDD_2 pads directly to the 1.8V voltage source. To achieve this, the following netlist file is used:

```
* prelayout_and_packaging.sp
.param HSIMSPF=vdd.dspf
VVDD VDD 0 1.8
MP A B VDD VDD p
.subckt rlc in out r=r l=l c=c
R in i r
C i 0 c
L i out l
.ends
vvdd_1 vpad_1 0 1.79
vvdd_2 vpad_2 0 1.81
xvdd_1 vpad_1 vdd rlc r=1 l=1n c=10f
xvdd_2 vpad_2 vdd rlc r=1 l=1n c=10f
.param HSIMSPFMERGEPIN=0
.param HSIMSPFNETIPIN='vdd vdd_1 xvdd_1:out'
.param HSIMSPFNETIPIN='vdd vdd_2 xvdd_2:out'
...
```

HSIMSPFMERGEPIN

In this example, setting HSIMSPFMERGEPIN=0 prevents merging pins VDD_1 and VDD_2 together. HSIMSPFNETIPIN parameters are used to reconnect pads to the out terminals of the `rlc` packaging model and connect the in ports of `XVDD_1` and `XVDD_2` to the respective `VVDD_1` and `VVDD_2` voltage sources.

The resulting netlist used during the simulation is shown on Figure 7 on page 43.

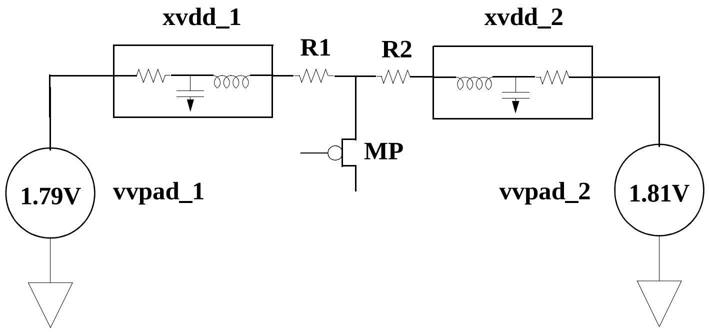


Figure 7 Post-layout Circuit with Packaging

HSIMSPFNETPPIN

```
.param HSIMSPFNETPPIN=”<net_name> <connect_point> \
   <new_instance_pin>”;
```

By default, HSIM^{plus} defines nets connected to variable voltage sources and nets which do not have direct connections to pads as signal nets. To treat such nets as power nets, you must use `HSIMSPFNETPPIN`. This parameter specifies which DSPF pins must be considered as voltage sources.

`HSIMSPFNETPPIN` is also used when connecting packaging to power net pads.

Note:

> `HSIMSPFNETPPIN` uses the same syntax as `HSIMSPFNETIPIN`. Refer to HSIMSPFNETIPIN on page 41 for additional information and examples.

Power Net IR Drop and EM Analysis Flow

If the entire power net is back-annotated in the circuit, it is possible to calculate the IR drop at each connection to the power net and the current through each resistor in the power net during transient simulation. However, such fully coupled simulation is practically infeasible because of the large power net size. For this reason, the typical approach consists of two phases.

Phase I

In Phase I, simulation is performed on the pre-layout netlist, without power net back-annotation (decoupled simulation). The currents of devices connected to the power net are stored in a file for future use.

Phase II

In Phase II, only the power net is simulated and the stored currents are injected into the power net and the IR drop and resistor current values are calculated. While this approach may give reasonable results in some situations, a large inaccuracy is introduced because of the effect of power nets on circuit simulation (i.e. on the currents stored) is completely ignored. PWRA implementation of power net reliability analysis overcomes this accuracy problem while keeping the simulation time reasonable.

In PWRA, the power net is reduced to a degree that the coupled simulation may be performed. Then the strongly reduced power net is back-annotated into the pre-layout circuit.

This power net has desired feedback, i.e. effect of power net on the circuit behavior. The currents stored during Phase I (coupled simulation) are used in Phase II, where those currents are injected into the original, unreduced power net, Figure 8 on page 45.

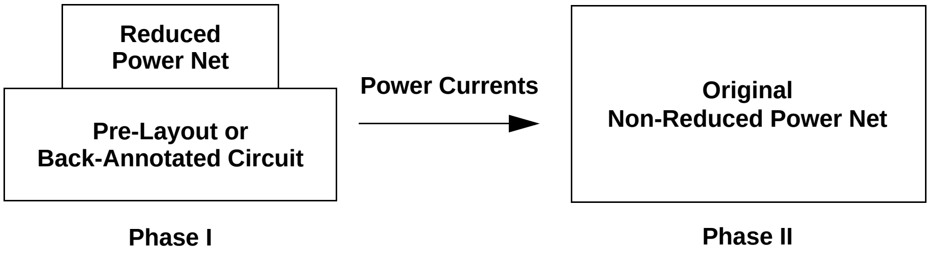

Reduced Power Net
Pre-Layout or Back-Annotated Circuit
Power Currents
Original Non-Reduced Power Net
Phase I
Phase II

Figure 8 Coupled Power Net Flow Diagram

Phase I Control Parameters

HSIMSPF

To perform power net reliability analysis, the DSPF file with the power net is specified using HSIMSPF. Refer to the HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation.

HSIMSPFPWNET

HSIMSPFPWNET is used to control the degree of power net reduction. Refer to HSIMSPFPWNET on page 36.

HSIMPWRA

HSIMPWRA activates the Reliability Analysis (RA) flow using the following syntax:

```
.param HSIMPWRA=1 (default HSIMPWRA=0)
```

To instruct HSIM that power net reliability analysis should be performed during the transient simulation, set HSIMPWRA=1 which activates the current saving process during Phase I of RA. When Phase I simulation is finished, HSIM^{plus} PWRA automatically executes Phase II of RA. To prevent this and manually start Phase II at a later time, use HSIMRAP2AUTO.

HSIMRAP2AUTO

To prevent PWRA from automatically executing Phase II of RA and manually starting Phase II at a later time, use HSIMRAP2AUTO=0.

Note:

> The HSIMRAP2AUTO command has the same functionality as and replaces the HSIMRADUMP command.

```
.param HSIMRAP2AUTO (default HSIMRAP2AUTO=0)
```

HSIMRAKEEPSERIESR

`HSIMRAKEEPSERIESR` restricts collapsing of series resistors. To collapse resistors, set the value to 0 (the default). To report the precise voltage (drop) values, set the value to 1.

```
.param HSIMRAPKEEPSERIESR <0|1>
```

HSIMRATAU

During Phase I simulation, PWRA saves device currents with every RA simulation step. The RA step value is controlled by HSIMRATAU with a default time interval of 1ns.

```
.param HSIMRATAU=1n (default)
```

HSIMRATCL

In Phase II, HSIM^{plus} PWRA does not read the circuit input netlist. A TCL script file is used to specify the Phase II control values. The script file can be named using HSIMRATCL.

Example 2

```
.param HSIMSPF='power_vdd.spf'
.param HSIMSPF='power_vss.spf'
.param HSIMSPFPWNET=4
.param HSIMPWRA=1
.param HSIMRAP2AUTO=0
.param HSIMRATAU=0.25e-9
.param HSIMRATCL='ra_vdd_vss.tcl'
```

As a result of Phase I, HSIM^{plus} generates some files with the results that are going to be used on Phase II of RA. For the example above, if the output name is test, then these files are generated:

- test.log: Usual log file.
- vdd.npf, vss.npf: Binary representation of the power nets.
- test-vdd.ranet, test-vss.ranet: Nodes, resistors, pins, connectivity, and coordinates.
- test-vdd.rasim, test-vss.rasim: Binary files created during transient simulation.
- test-vdd.ratcl, test-vss.ratcl: Phase II control files.

All parameters that can be used to control the execution of Phase I are listed below:

HSIMRARMIN

Resistors with value less than RMIN are deleted during parsing of the DSPF file. Terminals of deleted resistor become electrically equivalent.

```
.param HSIMRARMIN= <val>
```

HSIMSPFPWRMIN

HSIMSPFPWRMIN is used for simulation during Phase I, while value of HSIMRARMIN is used in Phase II simulation.

By default, the same value is used for HSIMSPFPWRMIN and HSIMRARMIN:

0.1: HSIMSPFPWNET=4 or 5

0.05: HSIMSPFPWNET=3

0.025: HSIMSPFPWNET=2

0.01: HSIMSPFPWNET=1 or 0

HSIMRAP2AUTO

```
.param HSIMRAP2AUTO=0,1 (default HSIMRAP2AUTO=0)
```

controls how intermediate files are dumped if Phases I and II are combined.

Note:

The HSIMRAP2AUTO command has the same functionality as and replaces the HSIMRADUMP command.

Syntax Definitions

HSIMRAP2AUTO=0

Only Phase 1 is performed and the .ranet/.rasim files are stored on disk. hsim -ra test-vdd should be run manually afterwards.

HSIMRAP2AUTO=1

hsim -ra test-vdd is executed automatically by the original hsim test.net -o test process

HSIMRATAU

```
.param HSIMRATAU=1n [default]
```

- Phase I: The average currents injected from transistor terminals into the original power nets are accumulated during this time interval.
- Phase II: The de-coupled original power net is simulated with this time step.

HSIMRATCL

```
.param HSIMRATCL=<file name>
```

Puts the line source filename into .ratcl command file created for each net. This way, TCL commands specified in filename are executed for each power/ground net.

HSIMOUTPUT

```
.param HSIMOUTPUT=format
```

Specifies the output format(s) of waveforms for nodes given in `printv` parameters and/or resistors specified in `printi` commands. PWRA currently supports FSDB, WDF, NASSDA, and EPIC formats. The default is FSDB format. Several waveform can be requested as shown in the following example:

Example 3

```
.param HSIMOUTPUT="wdf&epic&fsdb"
.param HSIMCOILIB=<path>/<lib name>
```

In this example, the syntax specifies the path and file name for the shared library used for output format generation. If this parameter is skipped, HSIM^{plus} PWRA searches for lib<out_format>.so in the following directories:

- Current working directory;
- $HOME directory;
- HSIM^{plus} PWRA run directory;
- LD_LIBRARY_PATH on Solaris and Linux, and SHLIB_PATH on HP

HSIMRAIRMIN

```
.param HSIMRAIRMIN=0,1 (0 by default)
```

During the trimming operation the following are performed:

- Reversible parallel and series reduction.
- Removal of all resistors smaller than RMIN.

Reversible Reduction For reversible reduction, EM current values are restored from the .ranet2npf cross-reference files. However, if resistors has been removed completely, no current information is available. To obtain the current values through the deleted resistors, use the following syntax:

```
.param HSIMRAIRMIN=1
```

Currents passed through the deleted resistors are restored based on the currents of the remaining resistors.

Phase II Control Parameters

To start Phase II, use the following syntax:

```
hsim -ra <output file name>-<power net name1> ..
   <output file name>-<power net nameN>
```

To start the Phase II for the VDD net only, use the following syntax:

```
hsim -ra test-vdd
```

HSIM^{plus} PWRA reads the .ratcl file for the correspondent power net and performs the specified types of analysis in Phase II. The following analysis types may be performed during the Phase II:

- 1|vmax: Maximum node IR drop.
- 2|imax: Peak resistor current.
- 3|irms: RMS resistor current.
- 4|iabs: Average absolute magnitude of resistor current.
- 5|iavg: Average resistor current.

During the Phase II, the currents stored from Phase I are injected into the entire power net. For each time interval equal to the HSIMRATAU, the following are recalculated and updated:

- Voltages at every power net node
- Currents through each power net resistor

If vmax analysis is specified, HSIM^{plus} PWRA generates the following files from Phase II:

- test-vdd.ralog: VDD log file
- test-vdd.radb: VDD net results
- test-vdd_vmax.ragds: vmax analysis violation map of the VDD net in GDSII format.

Defining Net Pins by Specifying X/Y Coordinates

You can define "*|P" net pins by specifying XY coordinates during Phase I, Phase II, and in static power net resistance (SPRES):

- In Phase I, use the HSIMSPFADDNETPINXY command. (See the *HSIM Simulation Reference Manual, Chapter 7, Post-Layout Back-Annotation* for details about this command.)
- In Phase II, use the addnetpinxy command in the RATCL file.
- In SPRES, use the addnetpinxy command in the SPRES TCL file.

The addnetpinxy command has the following syntax in the RATCL and SPRESTCL files:

addnetpinxy="<net> X=<xcoord> Y=<ycoord> layer=<malyer> pinxydist=<xydist>"

<net>

Specifies the net name.

X=<xcoord>

Specifies the X coordinate (in microns).

Y=<xcoord>

Specifies the Y coordinate (in microns).

layer=<malyer>

Specifies the metal layer name.

pinxydist=<xydist>

Specifies that only nodes that are within xydist distance from the user-specified XY coordinates are searched. The default xydist=1u.

The following formula is used to calculate the distance.

d = sqrt(dX2+dY2)

where dX = Xn-Xu; dY = Yn-Yu

Xn and Yn - node coordinates

Xu and Yu - user-specified coordinates

Output Files

HSIM uses the file names introduced in the previous section for output file names.

RALOG

The `RALOG` log file is always created using the following syntax:

```
<prefix>-<net name>.ralog
```

In the `RALOG` file, <prefix> is either hsim or name. It is user-specified on the command line after -o. This file contains the following information:

- Notification of program step completion.
- TCL commands read from *.ratcl files and processed by the program into internal structures.
- A simulation results table for each power net simulation timestep, containing the following is injected into the power net:
  - Worst IR drop (dVmax)
  - Node number

  - Maximum pin current (Imax)
  - Pin number
  - Total current (Itot)
- Additional statistics including: worst cases, total resistance, currents through PADs, etc.
- Information about files generated as a result of simulation. This information is presented in *.ralog only if ralayout was specified in*.ratcl file, otherwise they are located in the *.ltlog file.

RADB

The `RADB` simulation results file is always created consisting of voltages and currents. It is a binary file name composed of a prefix, net name, and extension as shown in the following syntax:

```
<prefix>-<net name>.radb
```

In the `RALOG` file, <prefix> is either hsim or name. It is user-specified in on the command line after -o. This binary file contains results of analyses requested with ra. In combination with *.ranet and *.npf files it is used for generation of various output files, including: *.raout, *.ragds, *.ascii, etc.

RAOUT

The `RAOUT` connectivity and results file is created if raout is specified in the *.ratcl file. The file name is composed of a prefix, net name, and extension as shown in the following syntax:

```
<prefix>-<net name>.raout
```

In the `RAOUT` file, <prefix> is hsim or name and is user-specified on the command line after -o. This ASCII file has the following sections:

Header

Header contains: net name, nominal voltage, start/stop times of the simulation, number of steps, and time interval of each step (i.e. HSIMRATAU)

Nodes

Nodes is created if peak IR drop analysis (type 1) is requested. This section has the following structure:

- #Node is the node index.
- max IRD is the maximum IR drop (mV).
- When IRD is the time when max IR drop occurred (ns).

- x, y are the node's coordinates (um).
- Node name is *.spf file name.

Resistors

Resistors is created if any type of EM analysis has been requested. This section has the following structure:

- #Resistor is the resistor index.
- #Term1, #Term2 is the resistor's terminals, specified as node indexes from the Nodes table.
- Resistance is the resistance value in Ohms.
- Imax - maximum current through resistor (uA).
- When Imax is the time when max current occurred (ns).
- Irms is the RMS current (uA).
- |I|avg is the average current magnitude (uA).
- Iavg is the average current (uA).
- Resistor name is the name of resistor from *.spf file.

Example 4

```
.BEGIN NODES
     #Node max IRdrop when IRD x y Node name
     #0 0.000mV 0ns 411.16 13.59 vdd
     #1 -16.840mV 4ns 10.42 10.41 vdd_pin18
     #2 -16.840mV 4ns 12.5 11.45 vdd_pin10
     …
.END NODES
.BEGIN RESISTORS
     #Resistor #Term1 #Term2 Resistance Imax When Imax Irms /
     Resistor name
     %1 3145 3146 6ohm 76.207uA 4ns 63.982uA R1
     %2 1582 3146 0.00663061ohm -76.207uA 4ns -63.982uA R2
     %3 1582 1590 0.0265224ohm -94.596uA 4ns -79.421uA R3
     …
.END RESISTORS
```

RAGDS

The `RAGDS` violation map layout file is created in either of the following ways:

- If ralayout is specified in the *.ratcl file.
- Using -ralayout as a parameter in hsim invocation.

The `RAGDS` file name is composed of a prefix, net name, analysis type, and extension as shown in the following syntax:

```
<prefix>-<net name>_<analysis type>.ragds
```

In the `RAGDS` file, <prefix> is hsim or name and is user-specified on the command line after -o. If -o is used in the ralayout file name, the command file name is shown in the following syntax:

```
<prefix>_<analysis type>.ragds.
```

The `RAGDS` file is binary and contains shapes (rectangle or path) and text labels in GDSII format. Each shape may have the following properties:

- Resistor Name
- Layer Name/Number from the original DSPF file
- Voltage Drop or Current Density
- Resistance Value
- Current through resistor (EM only)
  - Time: The time when the maximum value or usage occurs.
  - Ratio: Current density over the threshold (EM only).

The number and sequence of `RAGDS` file properties is controlled by the gdsprops TCL command.

TECH

The `TECH` technology file is always generated if any *.ragds file is requested for a net. The `TECH` file ASCII file has the same name as *.ragds file however, it has a different extension. `TECH` contains Virtuoso®-specific technology information necessary to correctly import GDSII files into a Virtuoso database.

DRF

The `DRF` Display Resources file is always generated if GDSII is requested for a net. The `DRF` file name is always hsim.drf. `DRF` defines ten colors for violation map layers as follows:

Forest Green

Green

Lime

Yellow

Gold

Orange

Brown

Maroon

Red

Bi kn R d

The smallest IR drop or EM corresponds to Forest Green and values greater than the threshold correspond to Blinking Red.

FSDB, OUT, WDF

The FSDB[1], OUT[2], WDF[3] waveforms are in various formats however, `FSDB` is the default format. One waveform format is always output and the format type is specified by HSIMOUTPUT. The file name is composed of the following, as shown in the syntax below:

- Net name
- Analysis Type
- Extension

```
<prefix>-<net name>_<analysis type>.fsdb
```

In the `FSDB`, `OUT`, and `WDF` waveforms, <prefix> is hsim or name and is user-specified on the command line after -o. Each waveform file contains at least 3 signals stored as follows:

- dVmax(vdd): Worst IR drop value reported at given time.
- Ipin_tot(vdd): Total current injected from all pins to the power net.
- Ipin_max(vdd): Maximum pin current injected to the power net.

To specify which nodes and resistors should be in the waveform files, the following TCL commands are used:

- printv
- printi
- printipad
- printvmode

ASCII

If requested, the `ASCII` violation map file is generated and presented in text form. The file is generated using the raformat ascii command. The `ASCII` file name may be composed of prefix, net name, analysis type, and extension as shown in the following syntax:

```
<prefix>-<net name>_<analysis type>.ascii
```

In the `ASCII` file, <prefix> is the name specified on the command line after -o. If -o is used in ralayout, the file name is as follows:

```
<prefix>_<analysis type>.ascii
```

The `ASCII` file is in the form of a text table and its content depends on the analysis type. For IR drop analysis, the *.ascii file may contain the following columns:

- Resistor name
- Voltage drop
- Original layer name/number
- x coordinate
- y coordinate
- Node name
- The time of the IR drop occurrence
- Actual voltage
- Level of violation

For EM analysis, the *.ascii file may have the following columns:

- Resistor name
- Current through resistor
- Current density through resistor
- Ratio: current density over threshold
- Original layer name/number
- Resistor length
- Time of maximum current occurrence
- Level of violation
- x coordinate
- y coordinate
- First width

- Second width
- Slack or threshold - current density
- Area: Calculated for VIA's or layers with a specified height using layerh.

The number, sequence, and sorting criteria for any ASCII file can be changed using either of the following commands:

```
asciicols
asciicolsort
```

Refer to asciicols on page 89 and asciicolsort on page 90.

Violation Map Visualization

Visualization marks different values of IR drop or EM current density with different colors producing a violation map. More precisely, a GDSII file is generated containing geometry elements, e.g. rectangles, tracks, etc., with different colors ranging from Forest Green for minor violations to Blinking Red for values exceeding a specified threshold.

The color map illustrates the distribution of IR drops and current densities within a maximum of 10 ranges determined by the redv and rediw commands. The color map, violation levels, and threshold ranges for 0.45V are shown below.

Forest Green	Level 1	|dV| < 0.05
Green	Level 2	0.05 <= |dV| < 0.1
Lime	Level 3	0.1<= |dV| < 0.15
Yellow	Level 4	0.15 <= |dV| < 0.2
Gold	Level 5	0.2<= |dV| < 0.25
Orange	Level 6	0.25 <= |dV| < 0.3
Brown	Level 7	0.3<= |dV| < 0.35
Maroon	Level 8	0.35 <= |dV| < 0.4
Red	Level 9	0.4<= |dV| < 0.45

Bi kn Rd	Level 10	0.45 <= |dV|

Note:

> Color mapping is defined in the hsim.drf file, which can be edited to change the colors.

Shapes correspond to an appropriate resistor from the *.dspf file. The coordinates of geometry elements are taken from the pin/sub-node statements and/or physical back-annotation portion of resistor statements in DSPF files. Element color is determined by the maximum values of voltage drop on resistor's nodes or current density through corresponding resistor.

Generating a Violation Map

Assuming that ra vmax irms is specified in the .ratcl command file and the maximum IR drop (vmax) with RMS current density (irms) had been calculated and stored into .radb files during the power net analysis phase, a violation map showing maximum IR drop (vmax) can be generated using the following syntax:

```
hsim -ralayout vmax test-vdd
```

The resulting test-vdd_vmax.ragds file is created. To change the name of output file, use the following syntax:

```
hsim -ralayout vmax test-vdd -o vdd
```

The layout is generated in the vdd_vmax.ragds file.

Generating Multiple GDSII Files with One Command

It is possible to generate several GDSII files using one ralayout command where, the names or numbers of the required analysis are specified as follows:

```
hsim -ralayout vmax imax irms test-vdd
```

Generating GDSII for All Analyses

To generate GDSII for all analyses, use the following syntax:

```
hsim -ralayout all test-vdd -o vdd
```

Generating a Violation Map over the Original Layout

In order to generate a violation map imposed over the original layout, specify ragds in .ratcl file as shown in the following syntax:

```
ragds -reflib <libName> -refcell <topCellName>
```

Syntax Definitions

libName

> The library name to which the original layout belongs or into which the original layout is loaded;

topCellName

> The name of the top cell in the original layout.

Displaying a Map Legend

To display a map legend indicating the voltage/current range for each layer number, use -legend in ragds as shown in the following syntax:

```
ragds -reflib <libName> -refcell <topCellName> -legend
```

User-Specified Layer Numbers

Layer numbers may be user-specified when displaying violation maps to eliminate map element conflicts with elements from the original layout.

Note:

> Do not specify more than 10 layers.

Layer numbers may be specified using ralayers in the .ratcl file as shown in the following syntax:

```
ralayers 51,53,57-60
```

This syntax distributes analysis results among six layers from 51 to 60 as follows:

`51`

> Layer 51 represents the lowest values.

`53`

> Layer 53 represents the next higher values.

```
57-60
```

Layer-60 represents values greater than the threshold specified in redv or rediw.

Layer Filtering

You can specify the original layers that should be placed into violation map using the `outlayers` command in the .ratcl file as shown in the following example:

```
outlayers 10,3,6
```

This command generates a violation map containing only the original layers numbered 10, 3, and 6. By default, HSIM^{plus} PWRA uses all layers.

Names Inserted into Geometry

HSIM^{plus} PWRA may put some names into geometry such as:

- Labels for pad points
- Pin names
- Node names specified in `printi` and `printv` within *.ratcl file.

gdslabels specifies which labels to be generated. It generates pad point(s) and node name labels specified in `printv` as shown in the following syntax:

```
gdslabels pad print
```

Generating Layout Formats

The `raformat` command is used to specify output formats for violation map display. Format selections are gds2, ascii, and rve.

```
raformat gds2 ascii
```

This command generates layouts in both GDSII and ASCII formats.

Automatically Generating Violation Maps

Violation maps can be automatically generated after power net analysis using the following syntax in the .ratcl file:

```
ralayout <analysis> [-o <prefix>]
```

In this syntax, <analysis> is the required analysis names or numbers (e.g., vmax, imax, etc.) separated by spaces or ALL for all analysis.

Loading GDSII Files into the Cadence Virtuoso Layout Editor

There are two ways to load GDSII files into Cadence® Virtuoso®:

- Load IR drop/EM violation map only.
- Load IR drop/EM violation map over original layout.

Loading the IR Drop/EM Violation Map Only

To load only a violation map, perform the following steps.

1. Select File/Import/Stream from the CIW menu. The dialog fields displayed include the following:
   - Run Directory: The working directory containing the .ragds, .tech, and hsim.drf files.
   - Input File: <fileName>.ragds.
   - Top Cell: <fileName> if specified in ralayout.
   - Library Name: The library into which <fileName>.ragds is loading.
   - ASCII Technology File Name: <netName>.tech
2. Load hsim.drf if it has not been loaded during the current session.
3. Input the following syntax in the CIW input command area:

   ```
   drLoadDrf("<working directory>/hsim.drf", nil)
   ```

4. Open the following:
   - view: layout
   - cell: <fileName>
   - library: The library into which <fileName>.ragds is loaded.

Loading a Violation Map Over the Original Layout

To load a violation map over original layout, perform the following steps:

1. Load the original layout if not loaded by selecting File/Import/Stream from the CIW menu. The dialog fields displayed include the following:
   - Run Directory: The working directory containing the original layout.
   - Input File: The file containing the original layout.
   - Top Cell: The top cell within the original layout.
   - Library Name: The library into which the original layout is loading.
   - ASCII Technology File Name: The technology file, if any.
2. Select File/Import/Stream from the CIW menu. The dialog fields displayed include the following:
   - Run Directory: The working directory containing the .ragds, .tech, and hsim.drf files.
   - Input File: <fileName>.ragds
   - Top Cell: <fileName>, if it was specified ralayout.
   - Library Name: The library containing original layout.
   - ASCII Technology File Name: <fileName>.tech
3. Load the hsim.drf file, if it was not loaded within the current session. In CIW input command area type:

   ```
   drLoadDrf("<working directory>/hsim.drf", nil)
   ```

4. Open the following:
   - view: layout
   - cell: <fileName>
   - library: The library containing the original layout.

.ratcl File Commands for Phase II Control

ra val

`ra val` specifies the analysis types to be performed during Phase II.

```
ra [-pwra|-sigra] val val=[all] [vmax] [imax] [irms] [iabs]
   [iavg]
```

-pwra

Executes Tcl for power net analysis only. Default is Tcl execution for both power and signal analyses.

-sigra

Executes Tcl for signal net analysis only. Default is Tcl execution for both power and signal analyses.

raout

`raout` specifies creates a file containing nodes an/or, resistors using the following syntax:

```
raout val
```

Syntax Definitions

val=1

The .raout file is created containing only nodes.

val=2

The .raout file is created containing only resistors.

val=3

The default. The .raout file is created containing nodes and resistors.

ralayout

`ralayout` specifies the analysis types to be dumped into output files. The layout is stored in the <prefix>_<analysisName>.<ext> file using the following syntax:

```
ralayout [-pwra|-sigra] <val> [-o <prefix>]
val == [all] [1|vmax] [2|imax] [3|irms] [4|iabs] [5|iavg]
```

Syntax Definitions

-o prefix

Specifies part of file name.

<prefix>

User-specified name.

<analysisName>

User-specified analysis name.

<ext>

File file extension (e.g. ragds, ascii, etc.).

-pwra

Executes Tcl for power net analysis only. Default is Tcl execution for both power and signal analyses.

-sigra

Executes Tcl for signal net analysis only. Default is Tcl execution for both power and signal analyses.

If <prefix> is omitted, HSIM uses <radb name> (the name of the *.radb file without the filename extension).

coordunit

`coordunit` is used to define the proper unit scale in generating IR drop and EM violation map(VM) in gdsii format.

```
coordunit [val]
```

By default, PWRA treats the process geometry as if it is in 1e-9m. If coordunit is not specified, the output geometry might have an incorrect scale. The default `coordunit` value is 1e-9.

Note:

Do not use a SPICE-formatted scaling number, such as 1n, in the `val` field.

redv

`redv` is used to control the IR Drop threshold for violation map generation as shown in the following syntax:

```
redv [val] [-lb val2] [-v | -r [{usage}]]
```

`redv` has 3 types of parameters that perform the following functions:

- Specify the IR Drop upper boundary threshold.
- Specify the IR Drop lower boundary threshold.
- Vary the map scale.

Specifying the IR Drop Upper Boundary Threshold val specifies the upper violation map boundary. This causes elements with the maximum IR drop at any terminal larger than val to be displayed in Blinking Red text. All other elements are distributed across the other 9 violation map layers according to their IR drop values. val must be specified in Volts. If val is skipped, HSIM^{plus} PWRA uses the maximum IR drop as val.

Example

Setting redv 0.09 distributes all resistors across the colored layers using voltage steps equal to 10mV. The Forest Green first layer contains resistors with maximum IR drop in the range of 0mV ~ 10mV. The Green layer contains resistors with IR drops of 10mV~20mV, through to the other end of the scale where the Blinking Red layer contains all resistors with IR drop greater then 90mV.

Specifying the IR Drop Lower Boundary Threshold The second parameter specifies the lower boundary for IR Drop threshold. No nodes with IR Drops lower than this value are output into the violation map.

Varying Map Scale A third type of parameter is used for varying colored map scale:

-v

Specifies the variable voltage scale as a % of voltage belonging to a given violation level. 100% voltage is the maximum IR drop.

-r

Specifies the distribution of resistors across the layers as a % of resistors belonging to a given violation level. 100% of the resistors is the entire set, with layout that compose a net.

Voltage Layout Distribution Assuming there are 10 color map layers and the maximum IR drop is 100mV, the following syntax generates a layout having the voltage distribution that follows.

```
redv -v 5 5 10 20
```

Voltage distribution

10th

Blinking Red. The maximum IR drop where the voltage distribution is 5% or 95mV ~ 100mV.

9th

Red. The voltage distribution is 5% or 90mV ~ 95mV.

8th

Maroon. The voltage distribution is 10% or 80mV ~ 90mV.

7th and all other layers

Since it is difficult to select voltage distributions that result in useful color maps, it is recommended that the resistor distribution be specified among colored layers of violation map. -r and %s specify how many resistors belong to each layer of the color map.

Resistor Layout Distribution

Assuming there are 10 color map layers and 1000 resistors in a power net, the following syntax generates a layout having the resistor distribution that follows.

```
redv -r 5 10 15
```

Resistor distribution

10th

Blinking Red. 5% or 50 resistors.

9th

Red. 10% or 100 resistors.

8th

Maroon. 15% or 150 resistors.

7th

Brown. The voltage distribution is 20% or 60mV ~ 80mV.

All other Layers

100 resistors.

PWRA automatically calculates the voltage scale.

rediw

```
rediw [default_val] [[(w_1@v_1 w_2@v_2 ...)]lval_1@layer_1]
   [lval_2@layer_2] ...
```

`rediw` specifies the current density (I/W) threshold for non-VIA layers. Layer threshold values can be specified as follows:

- default value for all layers
- default value for a given layer
- A set of values for specific widths of a layer. Width specific values must be specified within brackets () before the layer value.

The following syntax construct sets thresholds for several ranges:

```
(w1@v1 w2@v2)lval@layer
```

Syntax Definitions

v1

Current density value for resistors with a width within the range 0 - w1.

v2

Current density value for resistors with a width within the range w1 - w2.

lval

Current density value for resistors with a width greater than w2 have threshold lval.

Command parameters:

default_val

Threshold for all layers not mentioned in this command presented in uA/um.

lval_i

Threshold for specific layer presented in uA/um.

layer_i

Layer name.

w_i

Width presented in um.

v_i

Threshold for the given width presented in uA/um.

`rediw` sets the threshold for all types of current analysis. To set specific values for the given analysis use commands:

redimaxw, redirmsw, rediabsw, rediavgw

Syntax Definitions

redimaxw (imax)

Peak current

redirmsw (irms)

RMS current

rediabsw (iabs)

Average current magnitude

rediavgw (iavg)

Average current

All `redimaxw` commands have the same syntax as rediw. If no command is specified in the .ratcl file, HSIM^{plus} PWRA uses the maximum value for each layer for each analysis.

redia

```
redia val1@layer1 val2@layer2...
```

`redia` specifies the current density threshold for VIA layers. Since all layers specified by this command are treated as VIA, HSIM^{plus} PWRA calculates the current density through VIA resistors using their area, defined by width*width, and compares it with the threshold.

The threshold for each layer is presented in uA/um^{2}. There is no default value for all VIA layers as in rediw. Each VIA layer must have its own val@layer construction or it is considered to be a non-VIA layer with I/W threshold. redia covers all EM analysis. To set threshold for specific EM analysis, use the following commands:

redimaxa, redirmsa, rediabsa, rediavga

Syntax Definitions

`redimaxa`

Peak current

`redirmsa`

RMS current

`rediabsa`

Average current magnitude

`rediavga`

Average current

All analysis-specific commands have the same syntax as redia.

redj

```
redj val1 val2 val3...
```

`redj` is used for electro-migration analyses only. It specifies the Violation Map levels in J/Jmax ratios beginning with the lowest level.

Syntax Definitions

J

Current density.

Jmax

User-specified current threshold or the maximum current density calculated by HSIM.

To set Jmax use the following commands:

- rediw on page 66
- redia on page 68

emthreshproc, emldlayers

The `emthreshproc` and `emldlayers` commands provide a flexible capability to specify EM rules in SIGRA and PWRA to address multiple ways of calculating the threshold values. HSIM calculates the current through the resistor and compares the current against the maximum allowed current, which is the threshold Ith (not the current density).

The value of Ith for each resistor is calculated during post-processing through the call to the user-defined RA Tcl procedure. This procedure contains the formula/data necessary to calculate the Ith based on technology specific parameters, as well as the resistor width and/or length attributes.

The `emthreshproc` command assigns procedures to appropriate layers. The procedures are called during post-processing to calculate the Ith for each resistor. The values from HSIM are passed to the procedures as global parameters. HSIM supports the following parameters:

- `hs_width` for the resistor width
- `hs_length` for the resistor length

When used, the values of these parameters are passed to the procedures as global parameters. The units for these global parameters are microns. The procedures are expected to return the Ith value in uA.

For some EM rules, it is necessary to calculate the wire length, which is defined differently for metal and via resistors. For metal resistors, the wire length is defined as a maximum path between any two nodes on the wire. For via resistors, the wire length is the maximum wire lengths of wires that connect to the given via resistor. To correctly calculate/assign the wire length to the resistors, HSIM needs to know which layers are metal and which are via layers. You must use the `emldlayers` command to specify the layer types.

```
emthreshproc analysis proc_name@ layer_name
   [proc_name@layer_name...]
```

```
emldlayers <-m|-v> layer_name
```

Syntax Definitions

analysis

Specifies the one of the following types of EM analysis: imax, iavg, iabs or irms.

proc_name

Name of the user-defined Tcl procedure.

layer_name

Layer name

`-m`

Option to specify metal layer names.

`-v`

Option to specify via layer names.

Example of an RA Tcl File

The following example of an RA Tcl file contains commands that define the EM rules. Note that the example includes procedures in a separate `65nm.ratcl` file.

```
source 65nm.ratcl
emldlayers -m almi iaa ipdef -v cxx cww
emthreshproc imax t65_m1@almi t65_mx@iaa t65_my@ipdef
emthreshproc imax t65_vx@cxx t65_vy@cww
emthreshproc irms t65_irms_m1@almi t65_irms_mx1@iaa
t65_irms_mx3@ipdef
```

The following example show a Tcl file that defines procedures for EM analyses based on 65nm rules.

```
set UB 20
set LB 5

proc t65_metals {cW cI} {
global hs_length hs_width UB LB

  if { $hs_length >= $UB } {
    set threshold [expr $cI * ($hs_width - $cW) ]
  } elseif { $hs_length <= $LB } {
    set threshold [expr 4 * $cI * ($hs_width - $cW) ]
  } else {
    set threshold [expr ($UB / $hs_length) * $cI * ($hs_width - $cW) ]
  }
  if {$threshold > 0} {
    set threshold [expr 1000 * $threshold ]
  }
  return $res_threshold
}

proc t65_m1 {} {

set cW 0.02
set cI 1.8
set threshold 0

  set threshold [t65_metals $cW $cI]
  return $threshold
}

proc t65_mx {} {

set cW 0.02
set cI 1.9
set threshold 0

  set threshold [t65_metals $cW $cI]
  return $threshold
}

proc t65_my {} {

set cW 0.03
set cI 4.6
set threshold 0
```

```
  set threshold [t65_metals $cW $cI]
  return $threshold
}

proc t65_mz {} {

set cW 0.03
set cI 10.0

set threshold 0

  set threshold [t65_metals $cW $cI]
  return $threshold
}

proc t65_vias {cI} {
global hs_length hs_width UB LB

  if { $hs_length >= $UB } {
    set threshold $cI
  } elseif { $hs_length <= $LB } {
    set threshold [expr 4 * $cI ]
  } else {
    set threshold [expr ($UB / $hs_length) * $cI ]
  }
  if {$threshold > 0} {
    set threshold [expr 1000 * $threshold ]
  }
  return $res_threshold
}

proc t65_vx {} {

set cI 0.3
set threshold 0

  set threshold [t65_vias $cI]
  return $threshold
}

proc t65_vy {} {

set cI 0.9
set threshold 0

  set threshold [t65_vias $cI]
  return $threshold
}
```

```
proc t65_vz {} {

set cI 4.0
set threshold 0

  set threshold [t65_vias $cI]
  return $threshold
}

#-----  Irms

set dT 5

proc t65_irms_metals {cI cW1 cW2 cW3} {
global hs_width dT

set W $hs_width

set threshold [expr sqrt ($cI * $dT * pow (($W - $cW1), 2) * ($W + $cW2)
/ ($W - $cW1 + $cW3))]
return $threshold
}

proc t65_irms_m1 {} {
set cI 20.0
set cW1 0.02
set cW2 0.4
set cW3 0.06

set threshold [t65_irms_metals $cI $cW1 $cW2 $cW3]
return $threshold
}

proc t65_irms_mx1 {} {
set cI 7.0
set cW1 0.02
set cW2 0.5
set cW3 0.06

set threshold [t65_irms_metals $cI $cW1 $cW2 $cW3]
return $threshold
}

proc t65_irms_mx2 {} {
set cI 5.0
set cW1 0.02
set cW2 0.9
set cW3 0.06

set threshold [t65_irms_metals $cI $cW1 $cW2 $cW3]
return $threshold
}

proc t65_irms_mx3 {} {
set cI 3.0
set cW1 0.02
set cW2 1.5
set cW3 0.06

set threshold [t65_irms_metals $cI $cW1 $cW2 $cW3]
return $threshold
}
```

emlmaxiv, emlmaxim

`emlmaxiv` and `emlmaxim` are used to specify length-dependent electromigration rules for the maximum current density of via and metal layers respectively.

Note:

> The `emlmaxiv` and `emlmaxim` commands are more restrictive than the emthreshproc, emldlayers commands that you can specify in a Tcl procedure for greater flexibility.

```
emlmaxiv [-llb <val_l>] [-lub <val_u>] {cI@via_layer }
```

```
emlmaxim [-llb <val_l>] [-lub <val_u>] {cW@cI@metal_layer }
```

Syntax Definitions

val_l

> Length of the lower boundary in microns.

val_u

> Length of the upper boundary in microns.

cW

> Width correction value in microns.

cI

> Current value for threshold calculation in mA.

Several commands can be specified in one file. Each command can have information for several layers. The lower and upper boundaries can be specified once in any command.

jjmaxlog

For signal net RA analysis, there can be many nets and it can be difficult to view each net to find EM violations. To create a log report (one per net) on the J/Jmax value in order address identify nets with EM violations, use the following syntax:

```
jjmaxlog [<def_val>] [<val>@imax] [<val>@iavg] [<val>@iabs]
   [<val>@irms]
```

All of the options are used for filtration. Therefore, only nets with J/Jmax values that are greater than `<val>` or `<def_val>` are added to the log file.

ragds

```
ragds [-reflib <lib_name> -refcell <cellname>] [-legend]
```

`ragds` specifies the generation of VM over original layout, and presence of the legend in a layout. Command has the following parameters:

Syntax Definitions

lib_name

Library name with the original layout

cellname

Top cell name in the original layout

raformat

```
raformat [all] [1|gds2] [3|ascii] [4|rve]
```

`raformat` specifies the output layout format. The default is gds2(1). To generate output in ascii format, use raformat ascii. The raformat `rve` command option generates an error database that can be loaded into the Mentor Graphics RVE tool for visualization.

ralayers

```
ralayers [layer_1[{,layer_i|-layer_j}]]
```

`ralayers` specifies the layer numbers of the violation map layout using the following syntax:

Syntax Definitions

, (comma)

Specifies a single layer number.

- (dash)

Specifies a set of layer numbers, e.g. 51-54 equal to 51,52,53,54.

printi

```
printi <net_name>:<resistor_name>
   [{<net_name>:<resistor_name>}]
```

`printi` specifies the resistor currents inserted into wave output files. Currents are calculated at the Phase II of simulation. Several resistors can be specified in one command. Several `printi` commands can be specified in one .ratcl file. The asterisk (*) wildcard can be used for the <resistor_name>.

printv

```
printv <node_name> [{<node_name>}]
```

`printv` specifies the node voltage as either IR drop or the actual value inserted into the wave output files. Voltages are calculated at Phase II simulation. Several nodes can be specified in one command. Several `printv` commands can be specified in one .ratcl file. The asterisk (*) wildcard can be used for the <node_name>.

printvmode

```
printvmode [1|dv] [2|v]
```

`printvmode` specifies the type of dats output by the `printv` command.

Syntax Definitions

[1|dv]

Voltage drop

[2|v]

Actual voltage at a node.

The default is actual voltage at a node (2 | v).

printipad

```
printipad <pad_name> [{<printipad_name>}]
```

`printipad` specifies the net pad currents inserted into wave output files. Pads are specified by *P in the SPEF file. Currents are injected at Phase II simulation. Several pads may be specified in one command. Several printipad commands may be specified in one .ratcl file. The asterisk (*) wildcard can be used for the <pad_name>.

alterpad

```
alterpad <mode> dspf_net_pin dspf_net_node
```

`alterpad` specifies the new pad connections during second phase simulation. It is used for “what if” analysis.

Syntax Definitions

`<mode>`

Specifies whether you want to move or add a pad.

dspf_net_pin

Specifies the original/new pad name

dspf_net_node

Specifies the node you want to move the original/new pad to

Example 5

```
alterpad move vdd vdd:4
```

HSIM performs move of the original pad vdd to the node vdd:4.

```
alterpad add vdd xa1/mp:s
```

HSIM adds a new pad to the instance pin xa1/mp:s.

gdslabels

```
gdslabels [all] [1|pad] [2|print] [3|pins] [-layer <number>]
```

`gdslabels` specifies what text labels are generated and put into *.ragds file. By default, no labels are generated.

Syntax Definitions

[1|pad]

All pad names are specified as *|P in DSPF files,

[2|print]

Node names are specified in `printv` command.

[3|pins]

All pin names are specified as *|I in DSPF files.

[all]

Text labels for all objects mentioned in the previous bullets.

[-layer <number>]

Specifies the layer number for label visualization, where <number> is integer between 1 and 255. Note that the layer number should not be in the range of `ralayers`.

outlayers

```
outlayers [anType] layer_1[{,layer_i}] [-f]
```

`outlayers` specifies the original layers that are used to generate the violation map. Commas (,) are used to separate layer numbers or names.

Syntax Definitions

[anType]

Specifies one of the following analysis types: vmax, imax, irms, iavg, or iabs.

-f

Specifies that each layer must be in separate file.

If `outlayers` is skipped, HSIM^{plus} PWRA uses all layers.

skiplayers

```
skiplayers [anType] layer_1[{,layer_i}]
```

`skiplayers` specifies the physical design layers to exclude from GDSII or ASCII reporting. Skiplayers is opposite to "outlayers" command.

Syntax Definitions

[anType]

Specifies one of the following analysis types: vmax, imax, irms, iavg, or iabs. If no value is set, skiplayers applies to all analysis types

layer_1 [{,layer_i}]

Specifies the physical design layers to be excluded from output reporting

RA TCL may contain several “skiplayers” commands for different types of analysis. If both "skiplayers" and "outlayers" commands are used for the same analysis, HSIM uses the latest command in the specification.

Example

```
skiplayers  poly, nwell
skiplayers vmax via1, via2
skiplayers iavg metal, metal2
```

In the above example, poly and n-well layers are not reported for all RA analysis. In addition, via1 and via2 layers are skipped during vmax analysis and metal1 and metal2 layers are skipped during iavg analysis.

outfiltres

```
outfiltres w1@r1 w2@r2...
```

`outfiltres` is used to filter out resistors of specific width-resistance pairs in the violation map(VM). By default, HSIMplus recognizes and filters non-physical resistors generated by STAR-RCXT in the VM. For DSPF/SPEF generated by other RC extractors, you can use `outfiltres` to filter out resistors with a specific width-value pair.

Syntax Definitions

w1, w2

The resistor widths in microns.

r1, r2

The resistances in Ohms.

If `outfiltres` command is used, and the DSPF/SPEF is generated from Star-RCXT, the width-value pair defined in `outfiltres` overrides the default.

gdsmapvmax

```
gdsmapvmax -v float_val | -l int_val
```

`gdsmapvmax` specifies the filter for the number of resistors stored in the GDSII file resulting from IR drop analysis (vmax).

Syntax Definitions

-v

Value threshold. float_val specifies that resistors with an IR drop lower than this value is not stored in the GDSII file.

-l

Level threshold. int_val specifies that the number of mostly violated levels where resistors are stored in the GDSII file.

If this command is skipped, HSIM^{plus} PWRA stores all resistors.

gdsmapi

```
gdsmapi {<max|avg|rms|abs>} {-v|-l} [default_val]
   val1@layer1 [{val2@layer2}]
```

`gdsmapi <max|avg|rms|abs>` specifies the filter for the number of resistors stored in the GDSII file as a results of one of the following EM analyses: imax, irms, iavg, or iab.

Syntax Definitions

-v

Specifies that the filter threshold is determined by a value, current density (uA/um).

-l

Sets the filter threshold by level.

Resistors are placed into the GDSII file if its current density is greater than the filter threshold value (-v) or if it belongs to the level satisfied by the following condition: (10 - level) < threshold_level (-l). Each original layer may have its own val@layer threshold specification. If default_val is set, all layers without their own specification are treated as if they have the default_val value. If default_val is not set, and a layer does not have its own specification, all resistors in this layer are in the GDSII file. If `gdsmapi` is skipped, HSIM^{plus} PWRA stores all resistors.

gdsoutmode

```
gdsoutmode [ird@<<1|v>|<0|dv>>] [em@<<1|i>|<0|iw>>]
```

`gdsoutmode` specifies the type of information within the GDSII file as follows:

Syntax Definitions

For IR drop:

ird@1 or ird@v

Actual voltages

ird@0 or ird@dv

Voltage drop [default]

For Electro-migration analysis:

em@1 or em@i

Actual currents

em@0 or em@iw

Current density [default]

If `gdsoutmode` is skipped, HSIM^{plus} PWRA generates a voltage drop and current density map.

If actual voltages are specified, HSIM^{plus} PWRA ignores redv and builds the following maps:

- A regularly distributed voltage map using maximum voltage as the threshold.
- A regularly distributed current map using maximum current as the threshold.

gdsmag

```
gdsmag <[+]|-><val>
```

`gdsmag` specifies the magnification of shapes within VM layout. All DSPF file coordinates are handled as follows:

Syntax Definitions

+ (plus)

Sorts in ascending order.

- (minus)

Sorts in descending order.

`<val>` is positive

Multiplies by `<val>`.

`<val>` is negative

Divides by `<val>`.

tstart

```
tstart <time>
```

`tstart` specifies the start time for RA simulation. The default unit time value is seconds.

tstop

```
tstop <time>
```

`tstop` specifies the stop time for RA simulation. The default time value is seconds.

swin

```
swin -apw <t1 t2> [<t3 t4> ...] [net_pattern_1 net_pattern2
    ...]
```

`swin` specifies multiple tstart-tstop windows for RA power net analysis. t1 and t2 are used to specify the first simulation window.

If you simulate 100ns (100n) for the Phase I simulation, and you are only interested in 10n-20n and 50n-60n for the Phase II RA analysis, you can specify the following command in the .ratcl file.

```
swin 10n 20n 50n 60n
```

Without the -apw option, HSIM^{plus} PWRA calculates the overall IAVG, IRMS and IABS currents for all of the defined simulation windows. In contrast, when swin -apw is used, the IAVG, IRMS and IABS currents are calculated separately for each defined simulation window and then the peak value is reported. For example:

```
swin t1 t2 t3 t4 …
```

If IAVG analysis is performed, PWRA uses the following formula to compute IAVG for the defined simulation windows t1-t2, t3-t4, …

$$IAVG = \frac{1}{(t2 - t1) + (t4 - t3) + \ldots}\left(\int_{t1}^{t2} i(t)\,dt + \int_{t3}^{t4} i(t)\,dt + \ldots\right)$$

In contrast, if the -apw option is used as shown below:

```
swin -apw t1 t2 t3 t4
```

PWRA uses the following formula to compute IAVG for defined simulation windows t1-t2, t3-t4, …

$$IAVG = PEAK\left\{\begin{array}{l}\dfrac{1}{t2 - t1}\displaystyle\int_{t1}^{t2} i(t)\,dt \\ \dfrac{1}{t4 - t3}\displaystyle\int_{t3}^{t4} i(t)\,dt\end{array}\right.$$

Similarly, if IRMS and IABS analysis are selected, HSIM-RA uses following formulas for calculating the corresponding currents.

$$IABS = PEAK\left\{\begin{array}{l}\dfrac{1}{t2 - t1}\displaystyle\int_{t1}^{t2} |i(t)|dt \\ \dfrac{1}{t4 - t3}\displaystyle\int_{t3}^{t4} |i(t)|dt\end{array}\right.$$

Note:

The maximum number of simulation windows is 20.

$$IRMS = PEAK \begin{matrix} \frac{1}{t2 - t1} \sqrt{\int_{t1}^{t2} i\ t\ ^{2} dt} \\ \frac{1}{t4 - t3} \sqrt{\int_{t3}^{t4} i\ t\ ^{2} dt} \end{matrix}$$

Note:

> Due to the limitations of the viewers currently used, the waveform points which correspond to the end of previous window and the beginning of the new one will be connected by a straight line. However, since windows are specified by you, it should not be a problem, i.e., you know that the intervals between the windows must be ignored.

Note:

> Outside the specified windows, no RA analysis is performed in Phase II, and the result is set to zero.

Note:

> The PEAK function chooses the peak result from all swin windows, ignoring the sign during comparison. Please observe that IMAX and VMAX results are not affected by the -apw option.

You can also specify a pattern for one or more nets. For example:

```
swin 10ns vdd vss
```

This command specifies one window, 10ns-40ns, for two nets, vdd and vss.

```
swin -apw 5ns 15ns 25ns 35ns a*
```

This command specifies two windows, 5ns-15ns and 25ns-35ns, for all nets that start with "a". Also, averaging is done in each window.

```
swin 5ns 15ns 25ns 35ns i* o*
```

This command specifies two windows, 5ns-15ns and 25ns-35ns, for two sets of nets with names that start with "i" or "o". Averaging is done over the whole time of simulation.

twin

```
twin <w>
```

`twin` specifies a window of duration <w> for RA power net and signal net analyses around the time of peak current through the power source.

For example, if you simulate 100ns in the Phase I simulation, and the current peaks at 35ns, then by specifying the following command in the .ratcl file:

```
twin 10n
```

the Phase II RA analysis is automatically carried out in the window between 30ns and 40ns.

Note:

> Typically, the time of current peak does not necessarily correspond to the time of the maximum IR drop.

tau

```
tau <timestep>
```

`tau` permits setting a different value for the Phase II time step. In Phase I, the average currents injected from transistor terminals into the original power net are accumulated during the time interval specified by HSIMRATAU. By default, the same value is used as the time step for the Phase II.

layerh

```
layerh [<def_val>] <h1>@<layer1> [{<h2>@<layer2>}]
```

`layerh` specifies the height (thickness) of non-VIA layers. Height is used for current density calculation as shown in the following syntax:

```
J=I/(w * h) uA/um2
```

If height is not specified in <h@layer>, <def_val> is used as the default value for all layers. If <def_val> is omitted, h=1 is used. Height is displayed in um (microns). Several heights may be specified in a single command. Several `layerh` commands can be specified in a single .ratcl file. If several layerh commands have <def_val>, only the latest <def_val> will be used as default value.

layerea

```
layerea [<def_val>] <ea1>@<layer1> [{<ea2>@<layer2>}]
```

`layerea` specifies the coefficient of the effective area (EA) of VIA layers. EA is used for current density calculation as shown in the following syntax

```
J=I/((w * w) * ea) uA/um2
```

If EA is not specified in <ea@layer>, <def_val> is used as the default value for all layers. If <def_val> is omitted, ea=1 is used. Several EAs may be specified in one command. Several `layerea` commands may be specified in one .ratcl file. If several layerea commands have <def_val>, the <def_val> from the latest layerea command will be used as the default value.

layermap

```
layermap <numb_1>@<name_1>[<numb_i>@<name_i> …]
```

`layermap` specifies the mapping between layers numbers and names. Several layer number and name pairs can be put in a single command. Several commands may be used in one file.

rvemapvmax

```
rvemapvmax [-l] <val>
```

`rvemapvmax` specifies the filter for the number of nodes to be stored in the RVE output file resulting from IR drop analysis (vmax). This command supports two types of filtering: by by level of violation or IR drop value. To request filtering by level of violation, the -l parameter must precede <val>. In this case <val> must be an integer specifying the number of most violated levels to be outputted in the RVE file. In case of filtering only by <val>, the value must be in volts specifying minimum value of node IR drop to be outputted into RVE file. If this command is skipped, HSIMplus PWRA stores all nodes.

rvemapi

```
rvemapi{max | avg | rms | abs} [-l] [<default_val>]
   <val_1>@<layer_1> <val_2>@<layer_2> …
```

`rvemapi` <max|avg|rms|abs> specifies the filter for the number of resistors to be stored in the RVE output file as a result of any of the following EM analysis: imax, irms, iavg, iabs. This command supports two types of filtering: by by level of violation or value of current density. To request filtering by level of violation -l parameter must precede all other parameters. In this case < default_val> and all other <val_i> must be integers specifying the number of most violated levels to be outputted into RVE file. In case of filtering by value <default_val> and all <val_i> the value must be in uA/um specifying minimum value of resistor current density to be outputted into RVE file.

Each layer can have its own <val>@<layer> threshold specification, where <layer> is layer name/number. If <default_val> is specified, layers without individual specifications are treated as they have this <default_val> value. If <default_val> is not specified and a layer does not have its own specification, all resistors on this layer are listed in the output file. If rvemapi is skipped, HSIMplus PWRA stores all resistors.

asciimapvmax

```
asciimapvmax [-l] <val>
```

asciimapvmax specifies the filter for the number of nodes to be stored in the ASCII output file resulting from IR drop analysis (vmax). This command supports two types of filtering: by IR drop value or by level of violation. To request filtering by level of violation, the “-l” parameter must precede <val>. In this case <val> must be an integer specifying the number of mostly violated levels to be output into the ASCII file. In case of filtering by value, <val> must be in volts specifying minimum value of the node’s IR drop to be output into the ASCII file. If this command is skipped, HSIMplus PWRA stores all nodes.

asciimapi

```
asciimapi{max | avg | rms | abs} [-pwra|-sigra] [-l]
   [<default_val>] <val_1>@<layer_1> <val_2>@<layer_2>
```

asciimapi{<max|avg|rms|abs>} specifies the filter for the number of resistors to be stored in the ASCII output file as a result of any of the following EM analysis: imax, irms, iavg, or iabs. This command supports two types of filtering: by value of current density or by level of violation. To request filtering by level of violation, the “-l” parameter must precede all other parameters. In this case <default_val> and all other <val_i> must be integers specifying the number of mostly violated levels to be output into ASCII file. In case of filtering by value

<default_val> and all <val_i> must be in uA/um specifying the minimum value of the resistor's current density to be output into the ASCII file.

Each layer can have its own "<val>@<layer>" threshold specification, where <layer> is layer name/number. If <default_val> is specified, layers without individual specifications are treated as if they have this <default_val> value. If <default_val> is not specified and a layer does not have its own specification, all resistors belonging to this layer are listed in the output file. If asciimapi is skipped, HSIMplus PWRA stores all resistors.

-pwra

> Executes Tcl for power net analysis only. Default is Tcl execution for both power and signal analyses.

-sigra

> Executes Tcl for signal net analysis only. Default is Tcl execution for both power and signal analyses.

asciicols

```
asciicols <ird|em> {<column name>}
```

`asciicols` is a TCL command that adds a column and specifies which columns, and in what sequence they should be listed in an ASCII file.

Syntax Definitions

ird

> For IR drop, the following columns are selectable:
>
> - resname: Resistor name
> - dv: Value of IR drop (mV)
> - volt: Node voltage (V)
> - layer: Layer name or number
> - x, x coordinate
> - y, y coordinate
> - nodename: Name of node
> - time: Time when IR drop occurred
> - level: Level of violation

em

For EM analysis, the following columns are selectable:

- resname: Resistor name
- current: Current through resistor (uA)
- curdns: Current density through resistor (uA/um)
- j_jmax: Ratio J/Jmax. Jmax is user specified threshold for layer or maximum J calculated by program if threshold was not set.
- layer: Layer name or number
- x, x coordinate
- y, x coordinate
- w1, width_1
- w2, width_2
- slack: Result of subtracting <current density> - <threshold>
- time: Time when the maximum current occurred (IMAX analysis only)
- level: Level of violation

asciicolsort

```
asciicolsort <ird|em> [<[+]|-><name>|<[+]|-><numb]
```

`asciicolsort` specifies the number and sequence of columns used as sorting keys for sorting ASCII files. Columns used for sorting specifications must be specified in asciicols. Column names and numbers are the same as in asciicols. The sign controls the type of sorting as follows:

+ (plus): Sorts in ascending order.

- (minus): Sorts in descending order.

The syntax shown in Example 6 generates and sorts an ASCII file for IR drop analysis:

Example 6

```
asciicols ird dv time layer nodename
asciicolsort ird -dv time
```

The output from the above syntax appears as follows:

```
IRDrop(mV)                                Time(ns)       LayerNode name
2.254                                     2.9ns          6vdd:385
2.253                                     0.2ns          3vdd:161
2.253                                     0.2ns          3vdd:162
2.253                                     2.9ns          3vdd:182
2.218                                     2.9ns          6vdd:387
```

gdsprops

```
gdsprops [all] [none] {[<prop_numb>|<prop_name>]}
```

`gdsprops` specifies the set and sequence of properties added to each shape within the GDSII file. To set properties, use either of the following:

<prop_numb> is used as a GDSII file attribute. It can be specified as a number from 1 to 12. <prop_name> is the corresponding property.

`gdsprops` supports the following properties:

1|resname

Resistor name.

2|layer

Resistor's original layer number or name.

3|value

Max IR voltage (mV) drop or EM current density (uA/um or uA/um2).

4|resistance

Resistor nominal value in Ohms.

5|time

Time when value occurred in nanoseconds.

6|j_jmax

Ratio J/Jmax. Jmax is user specified threshold for layer or maximum J calculated by the program if threshold was not set.

7|current

EM current (uA).

8|nodes

Resistor node names.

9|group

Number of the group to which the resistor belongs. This property is used with the via grouping command from StarRCXT.

10|width

Resistor width.

11|length

Resistor length.

12|area

Resistor area.

13|rescur

Current through resistor when maximum IR drop.

14|resird

IR drop on resistor.

Set all properties with the following commands:

gdsprops all

Sets all properties. The sequence is the same as shown above.

gdsprops none

Eliminates all properties.

Note:

If gdsprops is skipped, HSIM^{plus} PWRA generates the first 4 properties.

gdstiming

```
gdstiming <-start start time> <-stop stop time> <-tau step>
```

`gdstiming` specifies a set of time steps for the dynamic visualization of power net analysis.

Syntax Definitions

-start

Starting point

-stop

End point

-tau

Step size for dumping dynamic visualization information

All parameters use nanoseconds as units.

Caution!

Dumping of this information is time and disk space consuming. Installing the minimum number of time points necessary to accomplish the task is highly recommended.

gdsdefrw

```
gdsdefrw <val>
```

`gdsdefrw` specifies the default resistor width. The value are used for visualization purpose only so, if a resistor has no width, current density is not calculated.

gdsdatatype

```
gdsdatatype <val>
```

`gdsdatatype` specifies data types for violation map layers in the GDSII file.

Syntax Definitions

<val>

Any integer between 0 and 256. The default value is 0.

gdsfilechsymb

```
gdsfilechsymb <symb_orig><symb_new>
```

The original symbol in the gds file is replaced with the new symbol.

Syntax Definitions

symb_orig

the original symbol

symb_new

the new symbol

gdsfilechsymb

@$

This substitutes @ with $ in all file names which contain GDSII format.

Note:

There is no space between <symb_orig> and <symb_new>.

There may be several symbol pairs (<symb_orig><symb_new>) in one command. Symbol pairs must be separated by space.

deflayer

```
deflayer <number|name>
```

`deflayer` specifies the default layers name and number for cases when layer is not output by the extractor. By default it is unknown or "" for some special cases.

rarve

```
rarve [-topcell <name>] [-diagtype {<2|poly> | <1|edge>}]
   [-defwidth <val>]
```

`rarve` sets several parameters used to generate RVE files.

Syntax Definitions

-topcell

Sets the name of top cell in RVE file. The default name is TOP_CELL. If <name> is the word "file", the `topcell` name will be: <prefix>-<net name>.

-diagtype

Specifies the type of a shape which will be generated for non-orthogonal resistor. The default value is <2|poly>.

-defwidth

Specifies the default width for all resistors which have zero width or which have no width statement in SPF/DSPF file.

raviewer

```
raviewer [1|virtuoso] [2|calibre]
```

`raviewer` selects the viewer that PWRA uses to generate technology and/or display resource files.

Syntax Definitions

1|virtuoso

default Virtuoso Layout Editor[4]

2|calibre

Calibre DESIGNrev[5]

rediac

```
rediac val_1@layer_1 val_2@layer_2...
```

`rediac` specifies the AC operation threshold for peak current analysis.

Syntax Definitions

val_i

The coefficient specified in a semiconductor foundries technology.

layer_i

The name of an SPF/DSPF file layer.

The `Ipeak_dc` value is a current density factor that specifies the limit of Amps/ micron of width. It is typically derived from the manufacturing specification of semiconductor foundries, and is often stated as a coefficient that is multiplied by the width for the appropriate layer. The current must be in Amps and the width in microns. For periodic signals, there is a peak current at which a metal line undergoes excessive Joule heating and can begin to melt. The limit for the peak current, Ipeak, is calculated using the following formula:

```
Ipeak = Ipeak_DC / sqrt(dratio)
```

- Ipeak_DC = coefficient * width;
- dratio = td / tau
  - td: signal width
  - tau: signal period

The `<prefix>-<net name>_acpc.dratio` file is automatically generated when the `rediac` command is used. This file contains a table with the following columns:

- Resistor name
- Signal width

- Signal period
- Peak AC operating current

imax analysis is required when `rediac` is used. HSIM calculates the currents through all resistors and generates violation map.

redirmst

```
dT cT1@cW1@layer_1 cT2@cW2@layer_2 ... cTi@cWi@layer_i
```

The `redirmst` command specifies the temperature dependent parameters to determine current density threshold used in RMS analysis. This command is specifically applied to current density threshold setting with the following equation:

Current Density Threshold = sqrt [cT x dT x resistor_width x (resistor_width + cW)]

The redirmst command does not work if the current density threshold equation is different from above.

If the current density threshold equation is different from above, use the flexible procedure-base method to set current threshold. See the emthreshproc command description.

`redirmst` uses the following parameters:

Syntax Definitions

dT

Temperature rise in celsius due to Joule heating.

cTi

Temperature coefficient for a specific layer. Unit: mA*mA/um*um.

cWi

Width coefficient(or adjustment) for a specific layer.

layer_i

Layer name for a specific layer.

redtrms

```
redtrms dT1@cT1@cW1@layer1 dT2@cT2@cW3@layer2...
```

`redtrms` specifies temperature analysis parameters using the following parameters:

Syntax Definitions

dTi

Temperature threshold for a specific layer.

cTi

Temperature coefficient for a specific layer.

cW

Width coefficient for a specific layer.

layer_i

Layer name.

avgviagrp

`avgviagrp <0|1>`

`avgviagrp` averages out all currents for parasitic resistors within via layers that have matching $grp attributes within an extracted DSPF netlist for power and signal nets.

Syntax Definitions

0

`avgviagrp` is not used. This is the default.

1

`avgviagrp` is turned on.

Example: DSPF Netlist

```
R1 net1:1 net1:2 10 $grp1 ...
R2 net1:2 net1:3 10 $grp1 ...
R3 net1:4 net1:5 10 $grp2 ...
R4 net1:5 net1:6 10 $grp2 ...
```

Using the above example, if avgviagrp is set, HSIM measures the currents through all four resistors and averages the results across each individual group.

Currents through R1==R2==((i(R1) + i(R2)) / 2) and R3==R4==((i(R3) + i(R4)) / 2).

Internal Power Nets

In HSIM^{plus} Power Net Reliability Analysis, external power nets are defined as any net directly connected to a constant voltage source. An internal power net is a net separated from the external power net by pre-layout resistors, inductors or MOSFET transistors and powers the rest of the circuit. To process internal power nets HSIM^{plus} needs to know the name of the internal power net and its source points. For internal power nets these source points are physical locations where power is sourced to the internal power net. Typically such points are located at terminals of pre-layout resistors, inductors or MOSFET transistors that separate internal power net from external one.

Use the HSIMPWNAME command to specify the name of internal power net and the HSIMPWTRACERL command to auto-trace the source point location.

When PWRA analyses are enabled, the voltage drop is calculated with respect to the constant voltage source of the parent external power net. The reference voltage can be changed by using the intvref statement. For example, If an internal power net vdda is connected to a vdd parent external power net with a 3V source, by default, the voltage drop in the vdda net is calculated with respect to 3V. To change the reference voltage to 1V, use the following command:

```
.param HSIMPWNAME=”vdda intvref=1V”
```

Given the name of the internal power net, HSIMPWTRACERL can usually detect the parent external power net. However, in some cases you need to provide further guidance to HSIMPWTRACERL by explicitly setting the source field in the HSIMPNAME option. For example:

```
.
.param HSIMPWNAME=”vdda intvref=1V source=vdd”
```

Note:

The internal power net source point locations can be set manually with the HSIMSPFNETPPIN command. HSIMSPFNETPPIN overrides HSIMPWTRACERL for specified internal power nets.

HSIMPWTRACERL

Additionally, when HSIMPWTRACERL is set using the syntax options described below, it automatically detects the following:

Syntax Definitions

1

Traces internal power net source points that connect to external power nets through pre-layout resistors or inductors.

2

Traces internal power net source points that connect to external power nets through pre-layout MOSFET transistors.

3

Traces internal power net source points through pre-layout resistors, inductors or MOSFET transistors.

HSIMPWTRACERL enables auto tracing to locate source point locations in internal power nets.

When PWRA analyses are enabled, the voltage drop is calculated with respect to the constant voltage source of the parent external power net. The reference voltage can be changed by using the intvref statement.

Example

If an internal power net vdda is connected to a vdd parent external power net with a 3V source, by default, the voltage drop in the vdda net is calculated with respect to 3V. To change the reference voltage to 1V, use the following command:

```
.param HSIMPWNAME=”vdda intvref=1V”
```

Given the name of the internal power net, most times HSIMPWTRACERL can detect an parent external power net. However, in some cases you need to provide further guidance to HSIMPWTRACERL by setting the source field explicitly in the HSIMPNAME command. For example:

```
.param HSIMPWNAME=”vdda intvref=1V source=vdd”
```

Note:

The internal power net source point locations can be set manually with the HSIMSPFNETPPIN command. The HSIMSPFNETPPIN command overrides HSIMPWTRACERL for specified internal power nets.

References

[1] FSDB is a binary file recognized by nWave from Novas Software Inc.

[2] OUT is ASCII EPIC format.

[3] WDF is a binary file recognized by SandWork Design, Inc.

[4] The Virtuoso Layout Editor is a product of Cadence Design Systems.

[5] The Calibre DESIGNrev is a product of Mentor Graphics.

4

Static Power Net Resistance (SPRES)

Provides information on performing a DC analysis of power net resistance with the HSIM^{plus} SPRES option.

Power Net Resistance Calculator

The HSIM^{plus} Power Net-RA option provides for high accuracy analysis of dynamic IR drop. During the layout of complex nanometer ICs, it can be valuable to quickly assess the parasitic effects of power net wiring, without performing a complete simulation. To provide this capability, the HSIM^{plus} Static Power Net Resistance option performs the complex calculation of all pad-to-pin and/or pad-to-internal instance pin resistances.

More specifically, for each instance pin (*|I) in the DSPF file, an effective resistance is calculated to all pads (*|P) connected together. To perform such calculations, no pre-layout netlist is necessary, only the net description in DSPF format is needed. The result of the calculation is output as an ordered list of instance pins and their resistances (in the order from largest to smallest values). In addition to a text output, a graphical display of the resistance distribution on the layout can be generated.

hsim -r

To run the static resistance calculation, use the command:

```
hsim -r <resistance_command_file> [-o <output_prefix>]
```

The <resistance_command_file> contains setup commands for resistance calculation as shown in Example 7. Refer to Net Resistance Calculator Commands on page 103 for a complete list of the commands.

Example 7

```
hsim -r r.tcl -o res
```

The command file should specify a DSPF file and net name for the calculation shown in Example 8.

Example 8

```
file extracted.spf
net gnd
rmin 0.05
report -nr 1000 -minr 100
```

As a result of the run, the following files will be created:

```
“res-gnd.rlog” -- log file
“res-gnd.rout” -- resistances from pads to pins, sorted in
decreasing order, pin coordinates (if available) and names
“res-gnd.minr100nr1000.report” -- subset of “res-gnd.rout” file,
filtered with “report” command
“res-gnd.png” -- colormap of resistances (red for maximum
resistance, green for zero) for quick viewing
```

hsim -rout

When *.rout file has been created, additional reports may be produced running following command where <report_options> are described in Report File Generation Options on page 108:

```
hsim -rout <prefix_of_rout_file> <report_options>
```

Example 9

```
hsim -rout res-gnd -pat “*:s” -minr 200
```

This will print reports of all pins with resistance larger than 200 ohms and names matching pattern “*:s” (i.e. source terminals).

routcols

```
routcols <column name>
```

The routcols command is used to control all of the columns within the `*.rout` file except for the first “Count” column, which contains each resistor number within the output table.

The routcols command is specified in the SPRES input file `(resistance_command_file).`

Syntax Description

`<column name>` may have the following number or name:

Number	Name	Description
1	resval	resistance value
2	wl	W/L ratio
3	xy	X, Y coordinates
4	gdsxy	X, Y coordinates within the GDSII file
5	name	instance pin name
6	hname	hierarchical instance pin name
all	—	all of the above mentioned columns

Net Resistance Calculator Commands

The following commands are used with the net resistance calculator.

file

Specifies the name of the DSPF file containing a power net description.

file

<DSPF_file_name>

Example 10

```
file gnd.dspf
```

net

net specifies the DSPF net name.

```
net <net_name>
```

Example 11

```
net gnd
```

addnetpin

This command is used to specify a temporary net pin.

```
addnetpin <net_name> <node_name>
```

Where <net_name> is the power net name that adds a pin, and <node_name> is instance pin or subnode.

Example 12

```
addnetpin vdd vdd:21
addnetpin vdd XI0|XI10|MP1:S
```

Restrictions:

Only one <node_name> in each command.

netdeletepad

This command permits removal of some net pads from the resistance calculation.

```
netdeletepad <net_name> <pad_name1> [<pad_name2> ...]
```

Example 13 The pads removed in either of the following examples are reported both on-screen and in the log file.

```
netdeletepad vdd VDD_1
netdeletepad vdd VDD_1 VDD_5 VDD_7
```

netinclude

netinclude inserts lines from <file.name> to the DSPF file after the *INET <net.name> statement. It allows the inclusion of extra statements without touching the DSPF file.

```
netinclude <net_name> <file_name>
```

Example 14

```
netinclude vdd vdd_added_ipins.txt
```

gds

gds output in GDSII format is performed.

```
gds <box_width> [-spfunits <val>] [-mag <m_val>] [-texttype
   <val>]
```

Every pin is output as a square of <box_width>. -spfunits is an optional parameter that specifies geometry units within the SPF/DSPF file. The default value for units is 1e-6. So if this option is skipped, SPRES will process all coordinates as if they are in microns. Optional parameter -mag specifies magnification coefficient for geometry in SPF/DSPF file. Negative <m_val> means that coordinates will be divided by <m_val>. Positive <m_val> multiplies coordinates. If you specify a -texttype value, the specified value is used for GDSII output.

gdslayer0

In the gds output, pins with the smallest resistance are output as <layer_num> in gdslayer0.

```
gdslayer0 <layer_num>
```

The <layer_num> default is 100.

gdsthresh

gdsthresh is an optional command used to control the distribution of resistors within levels of a violation map. If this command is skipped, all resistors are regularly distributed among 10 levels.

```
gdsthresh {[-lobnd <l_val>] [-upbnd <u_val>]} |
   {[-nproc <p_val_10> <p_val_9> <p_val_8>…<p_val_1>]}
```

gdsthresh has two forms as follows:

1. In its first form, gdsthresh specifies the threshold for the lowest and highest violation levels. Resistors between these thresholds will be distributed at regular intervals among 8 other levels.
   - -lobnd: Specifies lower boundary. All resistors with value lower than <l_val> will be placed into the lowest violated level (dark green color).
   - -upbnd: Specifies upper boundary. All resistors with value higher than <u_val> will be placed into the most violated level (blinking red color).
2. In its second form, gdsthresh specifies the percentage distribution of resistors among violation levels as follows:

-nproc specifies the percentage of all resistors that must be presented within the particular level, starting from most violated (blinking red color). All levels without specification (skipped value <p_val_i>) will be regularly distributed

ipin

`ipin` explicitly specifies instance pins for resistance calculation. By default, resistance is calculated for every instance pin. Using ipin reduces the number of resistance calculations required by specifying the list of instance pins.

```
ipin <pin_pattern>
```

Example 15

```
ipin vdd:f46 vdd:f8? vdd: f97*
```

layer0ohm

Resistors with indicated layer names and/or numbers will be shorted if resistor layer information is available in the DSPF file. Layer names in layer0ohm should be cross referenced to layer numbers in layer_map section of the DSPF file.

```
layer0ohm <layer_nameornum1> [<layer_nameornum2> ...]
```

Example 16

```
layer0ohm 5 mt1 cont
```

layerfactor

Resistor values belonging to an indicated layer in layerfactor are scaled by a factor. Either of the examples shown in Example 17 are acceptable.

```
layerfactor <layer_nameornum> <scale_factor>
```

Example 17

```
layerfactor mt1 0.1
layerfactor 5 100
```

rmin

`rmin <value>` specifies the minimum resistor values read from the DSPF file. Resistors with smaller values will be shorted so that there is a reduction of resistor elements with a corresponding accuracy trade-off. In the example below, any resistor smaller than 1e-9 Ohm will be shorted.

```
rmin 1e-9
```

```
rmin 0.5
```

png

If png is not set or <size_x> is less than 100, the colormap dimensions are automatically calculated. If coordinates are available, instance pins are displayed as pixels with colors ranging from red (maximum resistance) to green (zero resistance), e.g. png 1000 generates a colormap of size 1000 X - 1000 times the aspect_ratio. Graphic output is generated using a publicly available GD library.

```
png <size_x>
```

subnode

subnode causes the resistance to be calculated for a sub-node such as vdd:53 vdd:6*. By default, resistance to sub-nodes is not calculated.

```
subnode <subnode_pattern> [<node_pattern2> ...]
```

sortby

sortby specifies the method for sorting entries in the .rout file using the following values:

```
sortby <sortkey>
```

Syntax Options

r

Default. Generates the .rout file with entries sorted by resistance from the pads.

w/l

Sorts by the W/L ratio of MOSFET transistors.

rw/l

Sorts by the product of resistance and the W/L ratio of the transistor corresponding to the source or drain pin.

Report Generation Commands and Options

Filtered information can be obtained using either of the following methods:

Method 1: TCL File Command

report

Insert the report command into the TCL file as shown in Example 18.

```
report <report_options>
```

Example 18 TCL File Command.

```
report -nr 11 -minr 100 -xmin 0 -layer 24 -pat “*:s”
```

Method 2: Command Line Execution

hsim -rout

Execute hsim -rout from the command line using the shown in Example 19.

```
hsim -rout <rout_prefix> <rout_options>
```

Example 19 Command Line Execution

```
hsim -rout r-vdd -nr 11 -minr 100 -xmin 0 -layer 24 -pat “*:s”
```

Report File Generation Options

The following options are filters used to produce the *.report file from information in *.rout file.

-nr

No more than nr nodes are output.

-minr

Only nodes with res > minr are output.

-maxr

Only nodes with res < maxr are output.

-layer n

Only nodes with layer n (or no layer info) are output.

-xmin xmin, -ymin ymin

Only nodes specified coordinate region (or no coordinate info) are output.

-pat “pattern_with_wildcards”

Only nodes with hierarchical name:terminal matching given pattern are output.

-png x_size_in_pixels

Controls X-size of png image, use -png 0 to disable png output.

-gds <box_width>

Output in GDSII format is performed. Every pin is output as a square of <box_width>.

gdslobnd <val>

Sets the lower boundary/threshold for the distribution of resistors among violation levels. All resistors with values lower than <val> are placed into the lowest violated level (dark green color).

-gdsupbnd <val>

Sets the upper boundary/threshold for the distribution of resistors among violation levels. All resistors with values higher than <val> are placed into the most violated level (blinking red color).

-gdsmag <val>

Specifies geometry magnification. Negative <val> means coordinates are divided by <val>. Positive <val> multiplies coordinates.

-gdsunits <val>

Specifies geometry units within the SPF/DSPF file. The default value for units is 1e-6. If this option is skipped, SPRES will process all coordinates as if they are in microns.

-gdslayer0 <val>

Specifies the starting layer number for violation map generation. Default is 100.

Note:

GDS report file options, such as -gdsmag, for the report command are different from GDS commands, such as gdslayer0, for net resistance calculation. Do not use GDS commands as GDS options for the report file options.

Power Net IR Drop and EM Analysis Flow

If the entire power net is back-annotated in the circuit, it is possible to calculate the IR drop at each connection to the power net and the current through each resistor in the power net during transient simulation. However, such fully coupled simulation is practically infeasible because of the large power net size. For this reason, the typical approach consists of two phases.

Phase I

In Phase I, simulation is performed on the pre-layout netlist, without power net back-annotation (decoupled simulation). The currents of devices connected to the power net are stored in a file for future use.

Phase II

In Phase II, only the power net is simulated and the stored currents are injected into the power net and the IR drop and resistor current values are calculated. While this approach may give reasonable results in some situations, a large inaccuracy is introduced because of the effect of power nets on circuit simulation (i.e. on the currents stored) is completely ignored. PWRA implementation of power net reliability analysis overcomes this accuracy problem while keeping the simulation time reasonable.

In PWRA, the power net is reduced to a degree that the coupled simulation may be performed. Then the strongly reduced power net is back-annotated into the pre-layout circuit.

This power net has desired feedback, i.e. effect of power net on the circuit behavior. The currents stored during Phase I (coupled simulation) are used in Phase II, where those currents are injected into the original, unreduced power net, Figure 9 on page 111.

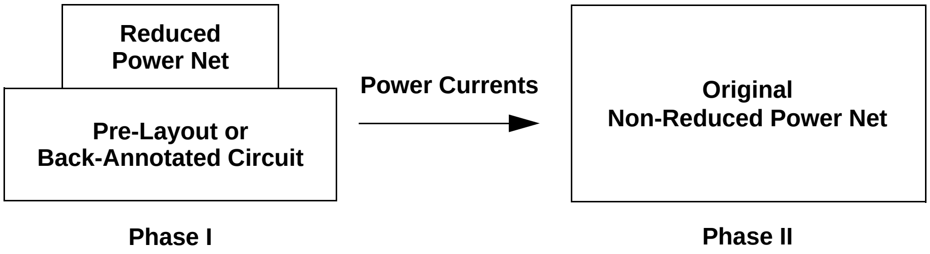


Figure 9 Coupled Power Net Flow Diagram

Phase I Control Parameters

HSIMSPF

To perform power net reliability analysis, the DSPF file with the power net is specified using HSIMSPF. Refer to the HSIM Simulation Reference Manual: Chapter 7, Post-Layout Back-Annotation.

HSIMSPFPWNET

HSIMSPFPWNET is used to control the degree of power net reduction.

HSIMPWRA

HSIMPWRA activates the Reliability Analysis (RA) flow using the following syntax:

```
.param HSIMPWRA=1 (default HSIMPWRA=0)
```

To instruct HSIM that power net reliability analysis should be performed during the transient simulation, set HSIMPWRA=1 which activates the current saving process during Phase I of RA. When Phase I simulation is finished, HSIM^{plus} PWRA automatically executes Phase II of RA. To prevent this and manually start Phase II at a later time, use HSIMRAP2AUTO.

HSIMRAP2AUTO

To prevent PWRA from automatically executing Phase II of RA and manually starting Phase II at a later time, use HSIMRAP2AUTO=2.

Note:

The HSIMRAP2AUTO command has the same functionality as and replaces the HSIMRADUMP command.

```
.param HSIMRAP2AUTO (default HSIMRAP2AUTO=1)
```

HSIMRATAU

During Phase I simulation, PWRA saves device currents with every RA simulation step. The RA step value is controlled by HSIMRATAU with a default time interval of 1ns.

```
.param HSIMRATAU=1n (default)
```

HSIMRATCL

In Phase II, HSIM^{plus} PWRA does not read the circuit input netlist. A TCL script file is used to specify the Phase II control values. The script file can be named using HSIMRATCL.

Example 20

```
.param HSIMSPF='power_vdd.spf'
.param HSIMSPF='power_vss.spf'
.param HSIMSPFPWNET=4
.param HSIMPWRA=1
.param HSIMRAP2AUTO=2
.param HSIMRATAU=0.25e-9
.param HSIMRATCL='ra_vdd_vss.tcl'
```

As a result of Phase I, HSIM^{plus} generates some files with the results that are going to be used on Phase II of RA. For the example above, if the output name is test, then these files are generated:

- test.log: Usual log file.
- vdd.npf, vss.npf: Binary representation of the power nets.
- test-vdd.ranet, test-vss.ranet: Nodes, resistors, pins, connectivity, and coordinates.
- test-vdd.rasim, test-vss.rasim: Binary files created during transient simulation.
- test-vdd.ratcl, test-vss.ratcl: Phase II control files.

All parameters that can be used to control the execution of Phase I are listed below:

HSIMRARMIN

Resistors with value less than RMIN are deleted during parsing of the DSPF file. Terminals of deleted resistor become electrically equivalent.

```
.param HSIMRARMIN= <val>
```

HSIMSPFPWRMIN

HSIMSPFPWRMIN is used for simulation during Phase I, while value of HSIMRARMIN is used in Phase II simulation.

By default, the same value is used for HSIMSPFPWRMIN and HSIMRARMIN:

0.1: HSIMSPFPWNET=4 or 5

0.05: HSIMSPFPWNET=3

0.025: HSIMSPFPWNET=2

0.01: HSIMSPFPWNET=1 or 0

HSIMRAP2AUTO

```
.param HSIMRAP2AUTO=2,1 (default HSIMRAP2AUTO=1)
```

controls how intermediate files are dumped if Phases I and II are combined.

Note:

> The HSIMRAP2AUTO command has the same functionality as and replaces the HSIMRADUMP command.

Syntax Definitions

HSIMRAP2AUTO=2

> Only Phase 1 is performed and the .ranet/.rasim files are stored on disk. hsim -ra test-vdd should be run manually afterwards.

HSIMRAP2AUTO=1

> hsim -ra test-vdd is executed automatically by the original hsim test.net -o test process

HSIMRATAU

```
.param HSIMRATAU=1n [default]
```

- Phase I: The average currents injected from transistor terminals into the original power nets are accumulated during this time interval.
- Phase II: The de-coupled original power net is simulated with this time step.

HSIMRATCL

```
.param HSIMRATCL=<file name>
```

Puts the line source filename into .ratcl command file created for each net. This way, TCL commands specified in filename are executed for each power/ground net.

HSIMOUTPUT

```
.param HSIMOUTPUT=format
```

Specifies the output format(s) of waveforms for nodes given in `printv` parameters and/or resistors specified in `printi` commands. PWRA currently supports FSDB, WDF, NASSDA, and EPIC formats. The default is FSDB format. Several waveform can be requested as shown in the following example:

Example 21

```
.param HSIMOUTPUT="wdf&epic&fsdb"
.param HSIMCOILIB=<path>/<lib name>
```

In this example, the syntax specifies the path and file name for the shared library used for output format generation. If this parameter is skipped, HSIM^{plus} PWRA searches for lib<out_format>.so in the following directories:

- Current working directory;
- $HOME directory;
- HSIM^{plus} PWRA run directory;
- LD_LIBRARY_PATH on Solaris and Linux, and SHLIB_PATH on HP

HSIMRAIRMIN

```
.param HSIMRAIRMIN=0,1 (0 by default)
```

During the trimming operation the following are performed:

- Reversible parallel and series reduction.
- Removal of all resistors smaller than RMIN.

Reversible Reduction For reversible reduction, EM current values are restored from the .ranet2npf cross-reference files. However, if resistors has been removed completely, no current information is available. To obtain the current values through the deleted resistors, use the following syntax:

```
.param HSIMRAIRMIN=1
```

Currents passed through the deleted resistors are restored based on the currents of the remaining resistors.

Phase II Control Parameters

To start Phase II, use the following syntax:

```
hsim -ra <output file name>-<power net name1> ..
   <output file name>-<power net nameN>
```

To start the Phase II for the VDD net only, use the following syntax:

```
hsim -ra test-vdd
```

HSIM^{plus} PWRA reads the .ratcl file for the correspondent power net and performs the specified types of analysis in Phase II. The following analysis types may be performed during the Phase II:

- 1|vmax: Maximum node IR drop.
- 2|imax: Peak resistor current.
- 3|irms: RMS resistor current.
- 4|iabs: Maximum absolute current magnitude.
- 5|iavg: Average resistor current.

During the Phase II, the currents stored from Phase I are injected into the entire power net. For each time interval equal to the HSIMRATAU, the following are recalculated and updated:

- Voltages at every power net node
- Currents through each power net resistor

If vmax analysis is specified, HSIM^{plus} PWRA generates the following files from Phase II:

- test-vdd.ralog: VDD log file
- test-vdd.radb: VDD net results
- test-vdd_vmax.ragds: vmax analysis violation map of the VDD net in GDSII format.

Defining Net Pins by Specifiying X/Y Coordinates

You can define "*|P" net pins by specifying XY coordinates during Phase I, Phase II, and in static power net resistance (SPRES):

- In Phase I, use the HSIMSPFADDNETPINXY command. (See the *HSIM Simulation Reference Manual, Chapter 7, Post-Layout Back-Annotation* for details about this command.)
- In Phase II, use the addnetpinxy command in the RATCL file.
- In SPRES, use the addnetpinxy command in the SPRES TCL file.

The addnetpinxy command has the following syntax in the RATCL and SPRESTCL files:

addnetpinxy="<net> X=<xcoord> Y=<ycoord> layer=<malyer> pinxydist=<xydist>"

<net>

Specifies the net name.

X=<xcoord>

Specifies the X coordinate (in microns).

Y=<xcoord>

Specifies the Y coordinate (in microns).

layer=<malyer>

Specifies the metal layer name.

pinxydist=<xydist>

Specifies that only nodes that are within xydist distance from the user-specified XY cordinates are searched. The default xydist=1u.

The following formula is used to calculate the distance.

d = sqrt(dX2+dY2)

where dX = Xn-Xu; dY = Yn-Yu

Xn and Yn - node coordinates

Xu and Yu - user-specified coordinates

5

Signal Net Reliability Analysis (SIGRA)

Describes the Phase I and Phase II control parameters used in signal net reliability analysis (SIGRA).

Overview

Signal net reliability analysis (SIGRA) is performed in $^{\text{HSIMplus}}$ similar to the power net reliability analysis described in Chapter 3, “Power Net Reliability Analysis (PWRA).” The analysis consists of two phases:

- Phase I: HSIM simulates a circuit with back-annotated signal nets and saves currents flowing through the net ports.
- Phase II: The saved currents are injected into the non-reduced signal nets and the current through each individual resistor of the net is computed.

Only electro-migration (EM) analysis is performed for signal nets so that the current density for each resistor is calculated, which then may be displayed in the form of a violation map in a GDSII file. The spots where the current density exceeds the specified threshold value are highlighted.

If HSIMRANET is used, only the `net(s)` or `net_pattern` specified in this parameter will be considered for RA.

```
HSIMRANET= <net(s) or net_pattern>
```

Analyzing EM effects on a design's signal nets is dependent on the input vector stimulus required to switch or activate these signal nets. If a less-than-complete vector is supplied, not all signal nets may be simulated. Vectorless signal net reliability analysis (VSIGRA) can be used to address incomplete vectors. Refer to Vectorless Signal Net Reliability Analysis (VSIGRA) on page 122.

Phase I Control Parameters

HSIMSIGRA

To perform EM Analysis for signal nets, set HSIMSIGRA=1.

HSIMRASIGCONLY

When HSIMSIGRA=1, the HSIMRASIGCONLY parameter controls whether signal nets selected with the HSIMRANET command are back-annotated as C or RC. In the default mode when SIGRA analyses are performed (HSIMSIGRA=1), HSIM back-annotates all HSIMRANET-selected nets as lumped capacitance to ground. If HSIMRASIGCONLY=0, HSIM back-annotates all selected signal nets as RC and reduces them using the HSIMPOSTL algorithm.

HSIMRANET

HSIM saves the currents for signal nets having the same names, according to a pattern set by HSIMRANET as shown in Example 22.

Example 22

```
.param HSIMSIGRA=1
```

`.param HSIMRANET=”x*”`: All signal nets with names that fit pattern x* are analyzed.

`.param HSIMRANET=”x2/x5*”`: A single net x2/x5 where the slash (/) character is the DSPF divider, will be the subject to RA analysis.

All information related to signal or power nets in the reliability analysis is stored and analyzed on a per-pattern basis. Only the nets specified in this parameter will go to RA Files created during simulation contain the HSIMRANET pattern as a part of their names as shown in Example 23. Refer to the *HSIM Simulation Reference Manual: Chapter 8, Simulation Output*. for additional information.

Example 23

```
.param HSIMRANET=”x2/x5”
.param HSIMRANET=”x3*”
.param HSIMRANET=”x4/x1/x2*”
```

Example 23 shows the results of EM analysis for the nets corresponding to the specified patterns. This information is written into files with the following elements in their names:

- x2.x5
- x3@
- x4.x1.x2@

A specific hierarchy can be set to specify the level for which EM analysis of signal nets is to be performed as shown in Example 24.

Example 24

```
.param HSIMRANET=”x* level=2”
```

When level is specified, quotation marks are required. Without the quotes, the level=2 string will be recognized as a comment. It is possible to have different levels set for different patterns. Example 25, illustrates a reliability analysis simulation run for `level=1` and `level=3` signal nets.

Example 25

```
.param HSIMRANET=”x* level=1”
.param HSIMRANET=”x*/* level=3”
```

HSIMSKIPRANET

```
HSIMSKIPRANET=<pattern>
```

If HSIMSKIPRANET is specified, the nets that fit the pattern will not be sent to RA as shown in the following example.

Let the circuit has SPF nets a1,a2,a3,b,c,d. If the following inputs are true, then RA will only be performed for nets b, c, and d:

```
.param HSIMRANET=*
.param HSIMSKIPRANET=a*
```

Phase II Control - .ractl File Commands

iavmin val

`iavmin val` specifies the threshold value for the average current injected into a signal net. The signal net is sent to reliability analysis only if the average injected current in the net exceeds this threshold. This allows the nets with very small currents that do not require analysis to be filtered out.

If you do not specify a unit type, the default is Amperes.You can also use symbols such as m (milli), u (micro), n (nano), p (pico), and so on. For example:

```
iavmin 10u
```

means that `iavmin` equals to 10 micro-Amperes. The default value of the threshold is 100 uA.

nnetmax num

nnetmax num specifies the number of signal nets with the maximum average injected current requiring analysis. This parameter allows the output and analysis time to be limited when there is a large total number of nets. The default is 100.

selectsignets

```
selectsignets net pattern [net pattern ...]
```

`selectsignets` command allows you to filter the signal net analysis during Phase II of the sigra analysis. For example:

```
selectsignets a* in
```

This will select for analysis at Phase 2 net in and all nets which fit pattern a*.

Vectorless Signal Net Reliability Analysis (VSIGRA)

Vectorless Signal Net Reliability Analysis can be used to quickly and accurately capture a design's signal net resistors' current densities to determine any potential electro-migration (EM) threshold violations without depending on the design's input vector set.

VSIGRA also accurately and dynamically captures the currents flowing through device instance pins connected to signal nets. However, analyzing EM effects on a design's signal nets highly depends on the input vector stimulus required to switch or activate these signal nets, thus if a less-than-complete vector is supplied, not all signal nets may be simulated thus not fully analyzing all nets in any particular design.

By building a representative circuit that represents each signal net, including functional driving and receiving devices as well as all parasitic elements connected to each signal net, VSIGRA is independent of a design's vector set, thus overcoming any coverage limitations. These individual signal net circuits are then simulated through the standard SIGRA flow to calculate any potential EM violations.

VSIGRA Flow

The VSIGRA flow consists of a netlist generation phase as well as the reliability analysis phase (simulation and analysis). VSIGRA generates a flat netlist (vsigra.sp) consisting of a unique circuit for each net defined within the extracted DSPF netlist which is also found within the pre-layout schematic netlist. You can also choose which nets are processed or ignored using the existing HSIMRANET command. Another way to select the nets for processing is to specify the minimum net capacitance. Only the net matching the pattern(s) of HSIMRANET and with net capacitance greater than a specified value are selected for processing.

VSIGRA reads in a pre-layout schematic netlist and an extracted DSPF netlist as well as configuration parameters to help drive the simulation. In addition, the VSIGRA flow also requires its own configuration file (HSIMVSIGRATCL) to control the netlist generation phase.

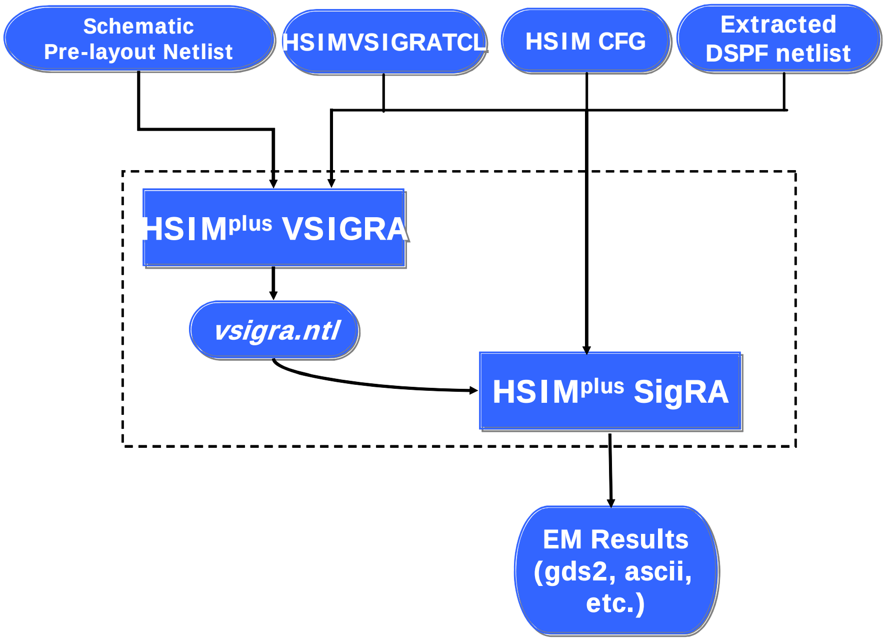


Figure 10 VSIGRA Flow

VSIGRA's netlist generation phase consists of reading in both the pre-layout and DSPF netlists and generating a new simulation netlist achieving full coverage on all specified signal nets for reliability analysis. The input DSPF netlist must be extracted up to the transistor level, it means its instance pin statements should correspond to the MOS transistor terminals.

To turn on VSIGRA the you must set the HSIMVSIGRA and the HSIMVSIGRATCL parameters.

Within the VSIGRA flow, you must provide HSIM with the following:

- Pre-layout SPICE netlist(s) (schematic netlist)
- DSPF netlist containing signal nets
- SPICE device models
- Configuration files:
  - A set of parameters to set up the SIGRA run as if the run was executed for the Phase I of SIGRA. This might include HSIMRANET parameter to filter all the nets that should be a subject for RA

- A VSIGRA CFG TCL File.

VSIGRA generates a new flat pre-layout netlist containing an individual circuit for each parasitic net processed. The new output netlist is then fed into SIGRA for the simulation and reliability analysis phase.

HSIMVSIGRA

```
.param HSIMVSIGRA=0|1
```

To initiate VSIGRA, set HSIMVSIGRA=1. The default is 0.

HSIMVSIGRATCL

```
.param HSIMVSIGRATCL=<vsigra_cfg_tcl_file>
```

Use the HSIMVSIGRATCL parameter to identify the VSIGRATCL configuration file.

VSIGRA CFG TCL File

The VSIGRA_CFG_TCL file read in by the HSIMVSIGRATCL parameter supports the following syntax definitions:

outfile <filename>

> Controls the name of the resulting output netlist. If this command is not set, the resulting output netlist name is vsigra.sp. All nets are in this one file.

autorun_outfile <filename>

> Controls the name of the output file(s) for the run initiated with the autorun command (see autorun below). If this command is not set, the resulting output name is outfile file name without the file extension.

include <filename>

> Specifies the name of the file to be included into the output file. This file should contain all of the necessary models and simulation setup information.

pwr <nodename>, gnd <nodename>, driver <nodename>

> Specifies the names and values for the nodes of the power, ground, and driver accordingly. It is assumed that the include file contains the voltage sources statements connected to those nodes.

cmin <value>

Specifies the minimum value of the net capacitance (in farads) that is used to select the nets for VSIGRA along with the HSIMRANET parameter.

autorun <1|0>

1: (default) Continue on to simulation and reliability analysis (SIGRA) phase immediate following netlist generation phase.

0: Exit after netlist generation phase is complete (do not run SIGRA).

A brief example of a VSIGRA_CFG_TCL file is provided in Example 26 below.

Example 26

```
.param HSIMVSIGRA=1
.param HSIMVSIGRATCL=myfile.vsigra

myfile.vsigra:
----------
outfile vsigra_tt108_nets.sp
include vsigra_setup.sp
driver in
pwr vdd
gnd vss
cmin 1e-15
autorun 0
```

The output VSIGRA netlist is named vsigra_tt108_nets.sp and each unique circuit within this netlist is driven by voltage sources defined in the file vsigra_setup.sp.

6

MOSFET Reliability Analysis (MOSRA)

This chapter describes how to use MOSFET reliability analysis (MOSRA) for both fresh and post-stress simulations. It also describes the HSIM API for defining custom HCI equations and for defining stress model equations. The Overview of the Unified MOSRA Solution section describes the unified MOSRA solution with HSPICE®.

Overview of MOSFET Reliability Analysis (MOSRA)

Hot-carrier injection (HCI) is a key reliability concern for MOSFETs in 90-nm technologies and below.

Device aging effects include changes in the following:

- Current-driving capability
- Threshold voltage (Vth)
- Trans-conductance value (gm)
- Sub-threshold slope (S)

A MOSRA license is required for the two-stage MOSRA flow:

- Stage 1: Fresh simulation; computes the stress (EAge) of each transistor in a circuit.
- Stage 2: Post-stress simulation; simulates a design with degraded model parameters based on the expected life-time of circuit usage.

Figure 11 shows the HSIM reliability simulation flow.

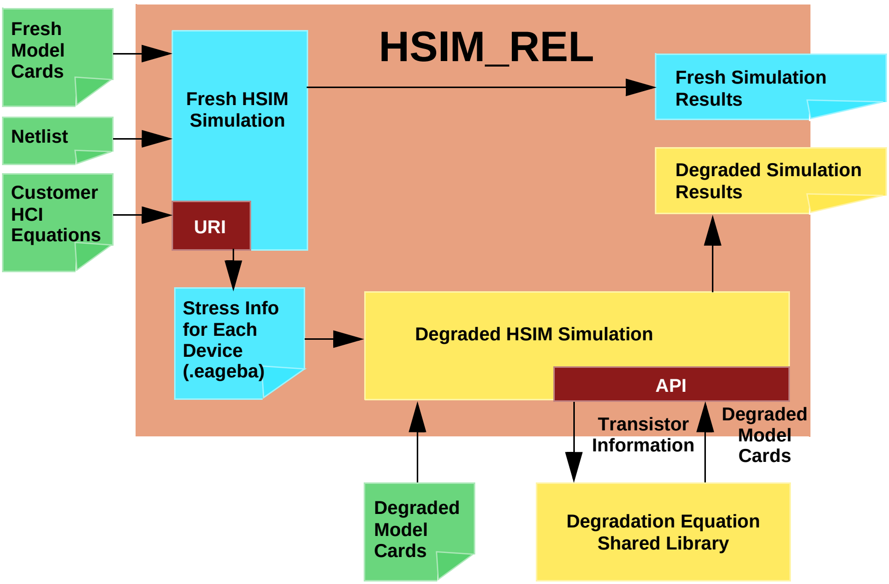


Figure 11 HSIM Reliability Simulation Flow Chart

Fresh Simulation

The fresh simulation is similar to a normal HSIM simulation except that MOSFET reliability analysis (MOSRA) is enabled for devices with transistor models containing the MOSRA parameters. At the end of the fresh simulation, the stress (EAge) of the devices for the duration of the simulation are reported in the .eage file and stored in the .eageba back-annotation file. The electron age (EAge) is a measurement of the amount of stress level induced in the devices.

Post-Stress Simulation

The aim of the fresh simulation is to compute the electron age of each transistor in a circuit. The aim of post-stress simulation is to simulate the degradation of the circuit, based on the stress information of each transistor produced by the fresh simulation and a degraded set of model cards. Refer to Figure 11 on page 128.

User Reliability Interface (URI)

In addition to the built-in HCI reliability model equation, HSIM provides an application procedure interface (API) that permits users to define custom HCI equations and define stress model equations. Refer to API Access on page 137.

Simulation Control Parameters

HSIMAGINGINST

HSIMAGINGINST enables user-specified instances for reliability analysis for both fresh and post-stress simulation using the following syntax:

```
.param HSIMAGINGINST=<instance_name>
```

The following example shows the command required to get a report of the electronic ages of the transistors for the xcore sub-circuit:

```
.param HSIMAGINGINST=x3m.x0.x0.xcore.*
```

HSIMAGINGSTART, HSIMAGINGSTOP

HSIMAGINGSTART and HSIMAGINGSTOP report device aging values for a specified time period. These commands are used for both fresh and post-stress simulation as shown in the following syntax:

```
.param HSIMAGINGSTART=<start_time>
.param HSIMAGINGSTOP=<stop_time>
```

HSIMHCIEAGEREFINST

HSIMHCIEAGEREFINST reports relative age values instead of absolute age values. This parameter is used for both fresh and post-stress simulation of HCI.

```
.param HSIMHCIEAGEREFINST=m1
```

The reported HCI age values of each element as a ratio of this element, m1.

HSIMHCIEAGETHRESHOLD

`HSIMHCIEAGETHRESHOLD` is used with the stressed model library to allow you to define whether a new model card is needed for each transistor, based on the HCI threshold. The default is value=0. This parameter is only used with URI in post-stress simulation. The following example illustrates the syntax.

```
.param HSIMHCIEAGETHRESHOLD=1e-12
```

HSIMHCIEAGESAMPLING

`HSIMHCIEAGESAMPLING` is used with the stressed model library to allow users to group new models cards. This parameter is only used with URI in post-stress simulation. The default is value=0. The following example illustrates the syntax.

```
.param HSIMHCIEAGESAMPLING=10
```

HSIMMOSRASIM

HSIMMOSRASIM is used to control both fresh and stress simulation runs using the following syntax:

```
.param HSIMMOSRASIM=<parameter>
```

Parameters

0

[default] Performs fresh simulation only.

1

Performs post-stress simulation only.

2

Performs both fresh and post-stress simulation.

HSIMRELMODE

HSIMRELMODE is used for both fresh and stress simulation as a selector for controlling whether a simulation will account for HCI effects as follows:

The following example illustrates the HSIMRELMODE syntax:

```
.param HSIMRELMODE=1
```

HSIMRELTOTALTIME

HSIMRELTOTALTIME permits users to specify the time length of circuit usage prior to post-stress simulation in seconds. This value typically corresponds to the expected life-time of the circuit, prorated with active usage. The default value is the transient simulation time at the fresh simulation. HSIMRELTOTALTIME is only used for post-stress simulation. The following example illustrates the hsimreltotaltime syntax:

```
.param HSIMRELTOTALTIME=6.3e+7     *corresponds to 2 years
```

The .eageba file generated by the fresh simulation will be scaled by the total reliability time in order to determine the stress level for the post-stress simulation.

HSIMURILIB

HSIMURILIB defines the stressed model library for both fresh simulation and post-stress simulation. The default is value=NULL. The example below illustrates the syntax.

Note:

HSIMURILIB is similar to HSIMCOILIB.

```
.param HSIMURILIB=”../../general/mylib.so”
```

Note:

Appendix 17, User Reliability Interface provides a description of the URI to define the HCI equations. Appendix 16, MOSRA Stressed Model Application defines the extensions required to include the stressModel equation API.

Modeling

Fresh Simulation Models

HCI

n-MOSFET model card must be extended to support reliability simulation for fresh simulation runs.

HCI model parameters are defined as follows:

hcim

Exponent for substrate current-to-drain current ratio. default is value=3.0.

hcih

Pre-factor coefficient for HCI effect. The default is value=1.0. hcih cannot be zero.

mosralevel

The type of MOSRA reliability model that should be used:

- 0 - no MOSRA reliability effect
- 1 - build-in equation, HCI effect only

These parameters can be added to a model card as shown in the following syntax:

```
.model model_name mosra
* reliability flag
+ mosralevel=1
* HCI parameters
+ hcim=val_1 hcih=val_2
```

Note:

mosralevel=1 indicates that the HCI effect is calculated for all MOSFET devices linked to this model card.

appendmodel

Since the model cards are typically contained in model libraries, it is often inconvenient (or prohibited) to edit the model libraries. The appendmodel command can be used to append the MOSRA parameters to the model cards.

To minimize explicit changes to the existing model card libraries, appendmodel uses the following syntax:

```
.appendmodel src_mod [model_keyword1] dest_mod
   [model_keyword2] [paramorder = val]
```

appendmodel appends the parameter values from the src_mod_name source model card to the dest_mod_name destination model card.

src_mod

The source model name, which should be the name of the MOSRA model.

model_keyword1

The model type for src_mod. This field is optional and it can be the keyword "mosra".

dest_mod

The destination model name, which should be the name of the original model in the model library.

model_keyword2

The model type for dest_mod. This field is optional and it can be ‘nmos’.

paramorder

The order of the model parameters in the modified model. The default value is 0.

Note:

model_keyword1, model_keyword2 and paramorder are optional and do not have to be specified for the fresh simulation model.

Note:

MOSRA supports binning parameters. It does not support any wild cards.

The appendmodel syntax can be used in either of the following ways:

```
.appendmodel hci_1 b3_nch
```

appendmodel appends the content of the model card hci_1 to the b3_nch BSIM3 model card. It can also be written as:

```
.appendmodel hci_1 mosra b3_nch nmos
```

Where mosra and nmos are the optional model type keywords.

Note:

HSIM supports user-specified reliability models in addition to the above built-in models. Refer to Appendix 17, User Reliability Interface for additional information.

Post-Stress Simulation Models

For post-stress simulation, the corresponding degraded models need to be applied to the devices based on the aging (EAge) of the device.

The following additional model card parameters are used in degraded model cards to group the similarly aged transistors together in a process similar to the binning approach.

HCIMIN

Minimum HCI electron age. [default] 0

HCIMAX

Maximum HCI electron age. [default] 1.0e32

Note:

Degraded model cards with the binning parameters HCIMIN and HCIMAX are associated with the original model names since the original model names are used in the transistor cards. The degraded model names are formed by appending eageN to the original model name, where N is an arbitrary number. For example, if the original model name is modn.5, the degraded model names can be modn.5eage1, modn.5eage2, modn.5eage3, ...

Example 27 provides an example of a degraded model card.

Example 27 Degraded Model Card

```
.model modn.1 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.6 u0=0.063
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.5
.model modn.2 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.58 u0=0.067
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.5
.model modp level=49
+ lmin=0.06u lmax=5u wmin=0.1u wmax=5u vth0=-0.6 u0=0.025
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.0e32
****** Additional model cards for stressed results ******
.model modn.1eage1 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.63 u0=0.061
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=1.5 hcimax=20
.model modn.1eage2 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.69 u0=0.06
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ iceman=20 hcimax=1.0e32
.model modn.2eage1 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.62 u0=0.064
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=1 hcimax=20
.model modn.2eage2 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.64 u0=0.062
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=20 hcimax=1.0e32
*****************
```

Typically, there are many device model parameters in the original model and there are only a few parameters that are added or changed in the post-stress binned models. The .appendmodel command can be used for the generation of the binned post-stress models. The binned models with the corresponding post-stress parameters must first be generated in a file. The following is an example to append the post-stress parameters to the standard model parameters for the model name appended with .eageN.

```
-------------------------------------------
.appendmodel n5v.1 nmos n5v.1eage0 mosra paramorder = 1
.appendmodel n5v.1 nmos n5v.1eage1 mosra paramorder = 1
-------------------------------------------
```

Since some of the post-stress model parameters, such as vtho, can be defined in both the standard model and also the post-stress model, the post-stress model parameters need to be appended to the standard model parameters. This can be accomplished by using "paramorder =1" in the .appendmodel command.

Output Files

Fresh Simulation

In addition to normal HSIM output files, two output files are generated from the MOSRA flow:

eage File

The eage file contains the HCI_EAGE values for devices in the HSIMAGINGINST list as shown in the following example.

```
hci_min=2.649110e-07 hci_max=5.367516e-07
m4
HCI_EAGE=5.367516e-07
m2
HCI_EAGE=2.649110e-07
```

In the eage file, the HCI stress value for each device is reported as the value of HCI_EAGE. The minimum and maximum values of all the devices are reported in the hci_min and hci_max values.

eageba File

The .eageba file is the Electron Age Back-Annotation file from a fresh simulation for post-stress model generation. Example 28 illustrates the syntax:

Example 28

```
.param HSIMRELFRESHTIME=5.000000e-08
.HSIMPARAM inst=m2 HSIMAGEMODEL=n.8
+ HSIMAGEHCI=2.649110e-07
+ HSIMAGE1=1.200000e-06 HSIMAGEW=1.500000e-05
.HSIMPARAM inst=m4 HSIMAGEMODEL=n.8
+ HSIMAGEHCI=5.367516e-07
+ HSIMAGEl=1.200000e-06 HSIMAGEW=1.500000e-05
```

The eageba file is used for post-stress simulation. The fresh simulation time, reported as the HSIMRELFRESHTIME values, is used to calculate the EAge value for the transistors at the circuit usage time specified by HSIMRELTOTALTIME.

The model name used for each device is reported as the value of the HSIMAGEMODEL in the eagaba file. The hci values reported in the eagaba file must be the same as those in the eage file.

Post-Stress Simulation

Similar to the fresh simulation, eage and eageba files are generated during post-stress simulation. These files have the same format as the fresh simulation eage and eageba files respectively. The eageba file contains the model name used for each transistor. This can be used to identify the model used for the post-stress simulation. The eage file contains the HCI aging values used for post-stress simulation.

API Access

URI for HCI Equations

In addition to the built-in reliability model equation for HCI, HSIM permits defining custom HCI equations using the URI model programming interface. The URI model is a dynamic library implemented with C language and the Synopsys-supplied API. Appendix 17, User Reliability Interface describes the procedure to define custom HCI equations.

URI Extension for Customized Stressed Model Equations

The URI for HCI equations includes custom stress model parameters. This flexible solution provides the ability to write custom equations and generate degraded transistor model card parameters based on length, width, and stress conditions. Appendix 16, MOSRA Stressed Model Application describes the MOSRA stressed model URI.

MOSRA Print Commands

```
.print LR91(<device instance>): hot-carrier static
   substrate current;
.print LR92(<device instance>): hot-carrier stress mileage;
```

Note:

The LX91, LX92, LX93 print commands that were sometimes used for MOSRA-relate prints are replaced by the LR91, LR92, LR93 print commands.

MOSRA Examples

Fresh Simulation Netlist Example

Example 29 Fresh simulation netlist of a two-inverter design.

```
* Netlist for MOSRA flesh simulation
vdd vdd 0 1.5
.subckt inv in out width=0.2u
m1 out in 0 0 modn w=width l=0.09u
+ as=4e-14 ad=4e-14 ps=1.0u pd-1.0u
m2 out in vdd vdd modp w='2*width' l=0.09u
+ as=4e-14 ad=4e-14 ps=1.0u pd-1.0u
.ends
x1        1         2         inv            width=0.2u
x2        2         3         inv            width=0.4u
.tran               0.1n 1u
.print              tran v(1) v(2) v(3)
.param HSIMAGINGINST=*
.param HSIMRELMOD=1
.inc "model/modellib"
```

Example 30 Model lib using width and length binning for the models.

```
.model modn.1 level=49
+ lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.6 uo=0.063
+ mosralevel=1 hcim=3 hcih=1 na=1 nhv=0.2 nga=1.0
+ ...
.model modn.2 level=49
+ lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.6 uo=0.063
+ mosralevel=1 hcim=3 hcih=1 na=1 nhv=0.2 nga=1.0
+ ...
.model modp level=49
+ lmin=0.06u lmax=5u wmin=0.1u wmax=5u vth0=-0.6 uo=0.025
+ mosralevel=1 hcim=3 hcih=1 na=1 nhv=0.2 nga=1.0
+ ...
```

The only difference between a MOSRA fresh simulation from a HSIM simulation is the additional HSIMAGINGINST and HSIMRELMOD parameters and the inclusion of the hcim and hcih parameters in the model cards.

In the fresh circuit, the binning model approach is used. Inside HSIM, the n-MOSFET transistor x1.m1 is linked to model modn.1, and the transistor x2.m1 is linked to model modn.2. Notice that the p-MOSFET transistors x1.m2 and x2.m2 are both linked to model modp.

Fresh Simulation Outputs

The eage and eageba files are generated in addition to normal HSIM output files.

eage file:

```
hci_min=2.649110e-07 hci_max=5.367516e-07
x1.m1               HCI_EAGE=5.367516e-07
x2.m1               HCI_EAGE=2.649110e-07
```

eagea file:

```
.param HSIMRELFRESHTIME=1.000000e-06
.hsimparam inst=x1.m1 HSIMAGEMODEL=n.8
+ HSIMAGEHCI=2.649110e-07
+ HSIMAGEL=9.000000e-08 HSIMAGEW=2.000000e-07
.hsimparam inst=x2.m1 HSIMAGEMODEL=n.8
+ HSIMAGEHCI=5.367516e-07
+ HSIMAGEL=9.000000e-08 HSIMAGEW=4.000000e-07
```

Note:

> In the example above, only HCI_EAGE values are reported because only HCI analysis was specified in the HSIMRELMOD parameter.

Post-Stress Simulation Netlist Example

The following syntax illustrates a MOSFET post-stress reliability simulation inverter chain.

```
*Netlist for MOSRA simulation with degraded model cards
vdd   vdd   0   1.5
.subckt inv in out width=0.2u
m1 out in 0 0 modn w=width l=0.09u
+ as=4e-14 ad=4e-14 ps=1.0u pd=1.0u
m2 out in   vdd vdd modp w='2 * width' l=0.09u
+ as=4e-14 ad=4e-14 ps=1.0u pd=1.0u
.ends
x1 1    2   inv width=0.2u
x2 2    3   inv width=0.4u
.tran 0.1n   1u
.print tran v(1) v(2) v(3)
.param HSIMAGINGINST=*
.param HSIMRELMOD=1
.param HSIMMOSRASIM=1
.param HSIMRELTOTALTIME=6.3e7
.inc “age0/hsim.eageba”
.inc “model/degraded.modellib”
```

Degraded model card:

```
.model modn.1 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.6 u0=0.063
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.5
.model modn.2 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.58 u0=0.067
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.5
.model modp level=49
+ lmin=0.06u lmax=5u wmin=0.1u wmax=5u vth0=-0.6 u0=0.025
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=0 hcimax=1.0e32
****** Additional model cards for stressed results ******
.model modn.1eage1 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.63 u0=0.061
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=1.5 hcimax=20
.model modn.1eage2 level=49
+lmin=0.06u lmax=0.1u wmin=0.1u wmax=0.3u vth0=0.69 u0=0.06
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ iceman=20 hcimax=1.0e32
.model modn.2eage1 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.62 u0=0.064
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=1 hcimax=20
.model modn.2eage2 level=49
+lmin=0.06u lmax=0.1u wmin=0.3u wmax=0.5u vth0=0.64 u0=0.062
+ mosralevel=1 hcim=3 hcih=1
+ .......
+ hcimin=20 hcimax=1.0e32
*****************
```

The post-stress simulation is specified by the HSIMMOSRASIM=1.

The hci_eage transistor values from the fresh simulation are specified in the age0/hsim.eageba file. hci_eage values will be converted to the corresponding values for the user-specified value of 6.3e7 seconds; approximately 2 years.

Associated Transistors Using the Degraded Model

If we assume that the converted value of HCI_EAGE for x1.m1 is 15, then transistor x1.m1 will be linked to the modn.1eage1 model for the degraded simulation. If we also assume that the converted value of HCI_eage for x2.m1 is 25, then transistor x2.m1 will be linked to the modn.2eage2 model for the degraded simulation. p-MOSFET transistors x1.m2 and x2.m2 will still be linked to the modp model.

Overview of the Unified MOSRA Solution

HSIM^{plus} and HSPICE introduced two unified MOSRA solutions for HCI and NBTI analysis:

- Unified built-in MOSRA equations

  The same MOSRA model equations are shared between HSIM^{plus} and HSPICE. The device threshold voltage and mobility are the two degradation monitoring parameters.

- Unified customized MOSRA equations

  The compiled customized MOSRA model equations in C-source can be shared and dynamically linked to HSIM^{plus} and HSPICE at runtime. The common set of API functions is shared by HSIM^{plus} and HSPICE.

Figure 12 shows an overview of the MOSRA unified simulation flow.

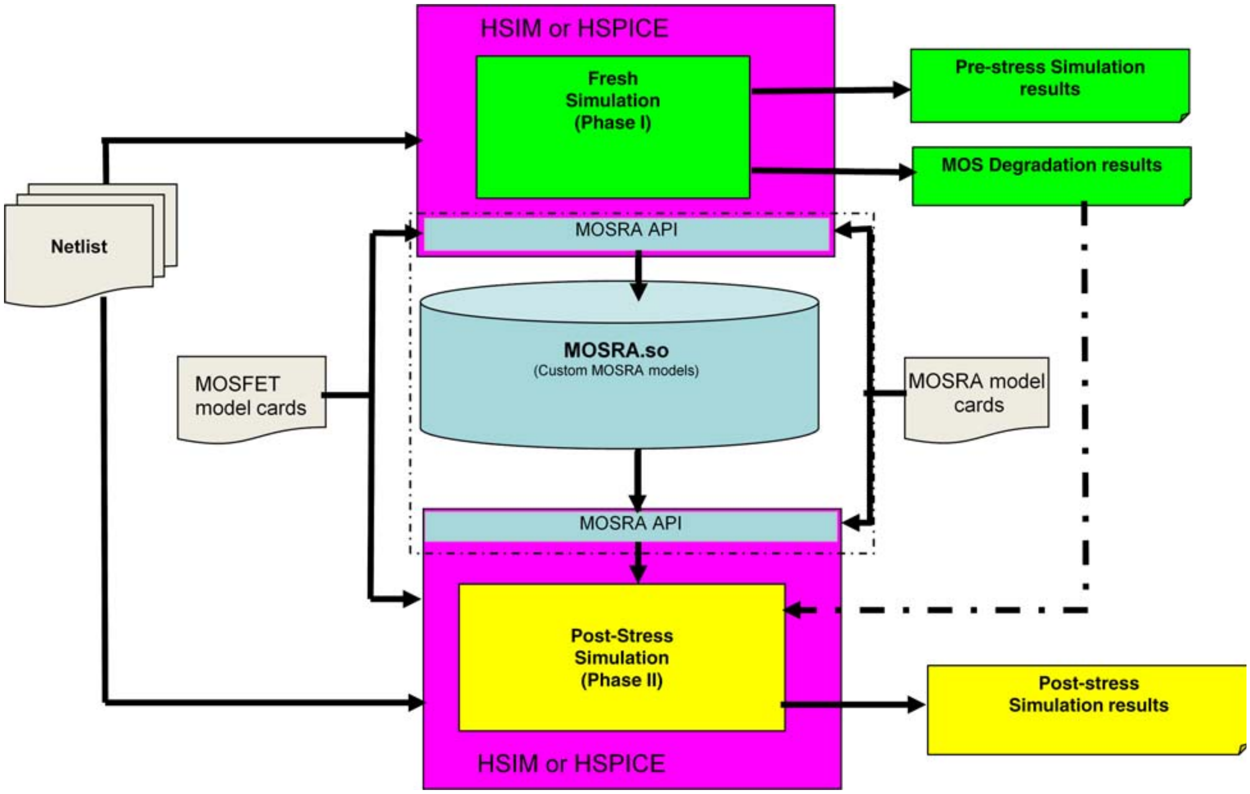


Figure 12 Simulation Overview

All the HSIM^{plus} existing MOSRA simulation control parameters listed in the Simulation Control Parameters section are supported in the unified MOSRA flow. In addition, HSPICE MOSRA syntax is also supported.

Note:

For more about the MOSRA flow, see the HSPICE User Guide: Simulation and Analysis, HSPICE MOSFET Model Reliability Analysis (MOSRA) chapter.

To set up the unified MOSRA flow, specify the following parameter and option:

HSIMUNIFIEDMOSRA

This command selects the unified built-in MOSRA equations or the MOSRA analysis using the following syntax:

```
.param HSIMUNIFIEDMOSRA=<0|1>
```

0

Uses the built-n MOSRA equations (the default).

1

Uses the unified HSIM^{plus} and HSPICE MOSRA equations.

Using Fresh and Post-Stress Simulation Models

To run the built-in MOSRA flow, define the model level as level=1. For example:

```
.model pch_age mosra level=1
```

After specifying the MOSRA model card, use .appendmodel to append MOSRA parameters to the model card. For example:

```
.appendmodel pch_age mosra pch pmos
```

During the post-stress simulation, HSIM^{plus} MOSRA automatically includes the changes from device stress effect based on the model parameter degradations or the aging of the device (`.eagba` file). No other model card is needed.

Output Files for Fresh and Post-Stress Simulations

For the fresh simulation, the default HSIM^{plus} output file, `hsim_fresh.log` and the waveform output file are created. HSIM^{plus} also output an `hsim_post.log` file for post-stress simulation. In addition, `.radeg` and `.eageba` files are created. The `.radeg` file contains the aging information for the devices defined in `HSIMAGEINGINST`. For example:

```
x2.m2
Device Type: PMOS
L       = 9.0000E-08
W       = 4.0000E-05
Bias Direction: bi-direction
DegFP    = 0.1
dvth     = 1.469055E-01
mulu0    = 100.000%
```

The `.eageba` file is the electron age back-annotation file from a fresh simulation for post-stress simulation. For example:

```
.hsimparam inst=x2.m2 hsimagemodel=pch hsimmosra_dvt    =
1.469055E-01
.hsimmosra_mulu0  = 1.000000E+00
```

API Access

HSIM^{plus} MOSRA allows the custom HCI and NBTI equations using a common set of modeling APIs with HSPICE. The custom models are implemented in the C language. For HSIM^{plus} to link to the compiled library at runtime, set the environment variable `setenv hspice_mosra_models` to point to the compiled library. For example:

```
setenv hspice_mosra_models /home/sim/lib
```

The supported platforms are Solaris 32/64 bit, Linux 32 bit, and AMD 64 bit.

Running the MOSRA Flow

The unified flow supports fresh and post-stress simulations. Both phases can run in one simulation or in two separate runs.

Running a MOSRA Example with Built-In Equations

The following example shows a MOS NBTI reliability simulation using a unified built-in model on an inverter chain.

```
* netlist for mosra simulation with degradation model card
vdd vdd 0 1.5
vin 1 0 pwl 0n 0v 1n 0v 1.01n 1.5 2n 1.5 2.01n 0 R
.subckt inv in out vdd width=0.2u
m1 out in 0 0 nch w=width l=0.09u as=4e-14 ad=4e-14 ps=1.0u pd=1.0u
m2 out in vdd vdd pch w='2*width' l=0.09u as=4e-14 ad=4e-14 ps=1.0u
pd=1.0u
.ends

x1 1 2 vdd inv width=10u
x2 2 3 vdd inv width=20u
cload 3 0 1p
.tran 0.1n 1u
.print tran v(1) v(3)
.PARAM HSIMAGINGINST='*'
.param hsimreltotaltime=6.3e+7
.param hsimunifiedmosra=1

.model nch nmos level=54 version=4.5
.model pch pmos level=54 version=4.5

.model nbti_p mosra level=1
+tit0 = 5e-7  titfd = 7.5e-10  tittd = 1.45e-20
+tn = 0.25
.appendmodel nbti_p mosra pch pmos
.end
```

After the MOSRA fresh and post-stress simulations run, check v(3) on the timing shift by comparing waveforms.

Running a MOSRA API Example

This section uses a MOSRA API netlist example to run the unified flow.

```
* test circuit using demo MOSRA API model
.option mraapi=1
.model n1 nmos level=54
+version = 4.4
+vth0    = 0.25
.model p1 pmos level=54
+version = 4.4
+vth0    = -0.25
* mos reliability model card, MRA demo models
.model n1_ra mosra level=101
+ rela=1e-4 relb=2 reln=0.25
.model p1_ra mosra level=101
+ rela=1e-4 relb=2 reln=0.25
* appendmodel command
.appendmodel n1_ra mosra n1 nmos
.appendmodel p1_ra mosra p1 pmos
.mosra reltotaltime = 3.1536e+8 aginginst = ‘*’
.tran 1n 100n
.print tran v(*)
.end
```

Correlating the MOSRA Output Files with HSPICE

To correlate the results between HSIM^{plus} and HSPICE simulations, run the `test.sp` netlist in HSPICE. Below are the fresh and post-stress sample timing outputs from the HSPICE runs.

HSPICE Fresh Simulation Results

HSPICE reports the signal delays from fresh simulation in hspice.mt0:

```
$DATA1 SOURCE='HSPICE' VERSION='A-2007.12'
.TITLE '* test circuit using demo mosra api model'
 delayr           delayf           periodr          periodf
 temper           alter#
  1.236e-10        1.236e-10        2.060e-11        2.059e-11
   27.0000           1.0000
```

HSPICE Post-Stress Simulation Results

The delays after post-stress simulation are reported in hspice.mt1:

```
$DATA1 SOURCE='HSPICE' VERSION='A-2007.12'
.TITLE '* test circuit using demo mosra api model'
 delayr           delayf           periodr          periodf
temper           alter#
  1.280e-10        1.281e-10        2.134e-11        2.134e-11

   27.0000           1.0000
```

The HSIM^{plus} MOSRA timing shift after post-stress simulation is within a 2% difference of HSPICE MOSRA simulation.

7

HSIM^{plus}-PrimeRail Interface

This chapter describes the flow, parameters, and available options for using the HSIM^{plus}-PrimeRail interface.

HSIM^{plus} PrimeRail Flow

The HSIM^{plus}-PrimeRail flow allows you to perform top-level rail analysis using HSIM^{plus} to simulate transistor-level blocks within the design. This flow accurately solves for all necessary dynamic data (currents, voltages), which is crucial during top-level power and rail analysis.

As shown in Figure 13, PrimeRail relies upon HSIM^{plus} for the transistor-level dynamic simulation results prior to solving the global matrix for top-level analysis.

The transistor-level designs are simulated using HSIM, which solves for all device currents. These dynamic results are then fed back to PrimeRail's top-level matrix solver for accurate top-level power and rail analysis results. PrimeRail gives users the capability of analyzing designs containing mixed gate- and transistor-level blocks. PrimeRail's transistor-level analysis portion relies on HSIM^{plus} for transistor-level SPICE simulation and Star-RCXT™ for transistor-level device and parasitic extraction.

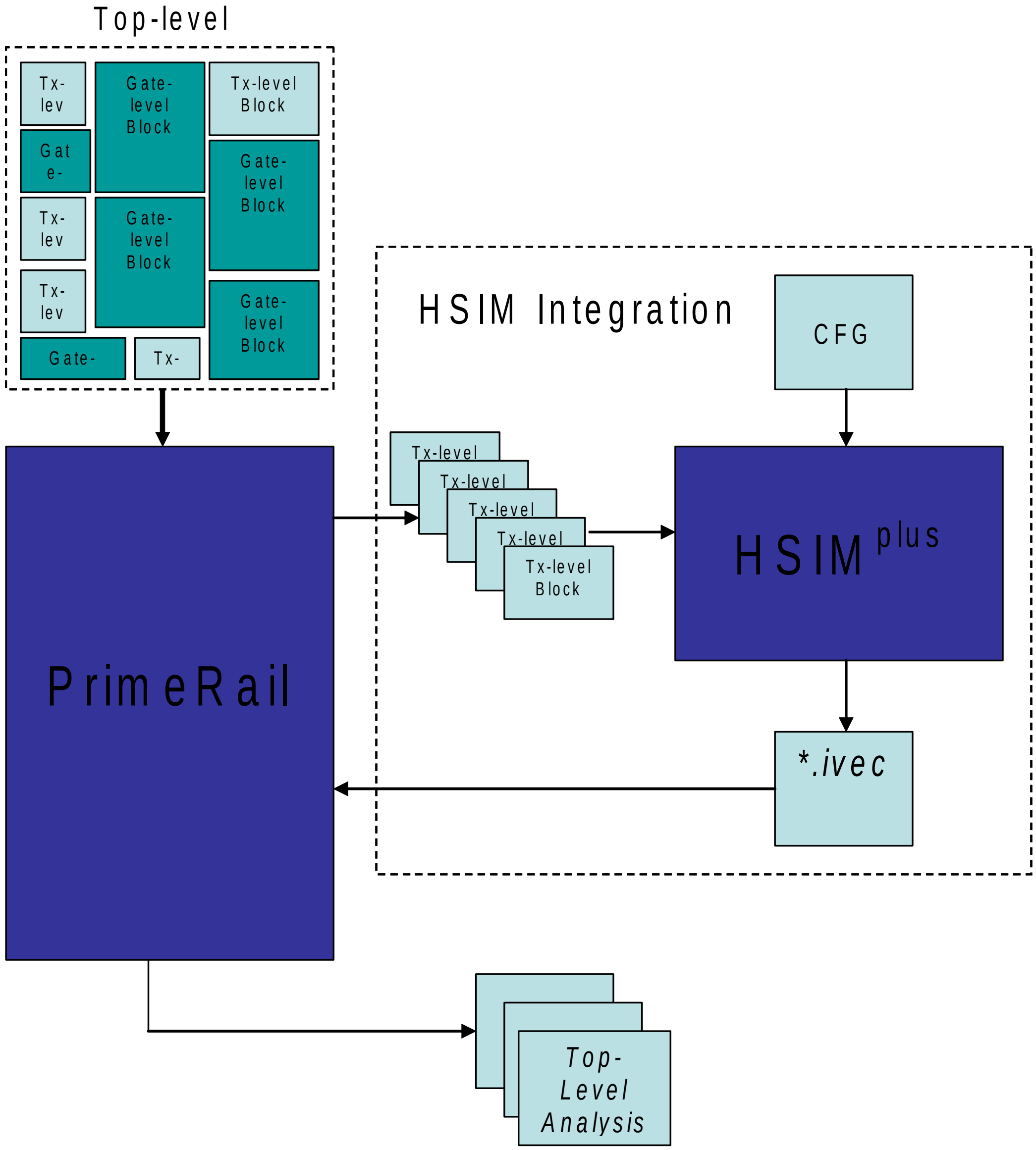


Figure 13 HSIMplus-PrimeRail Integration Flow

HSIM Simulation

The HSIM simulation portion of this flow follows that of a typical HSIM-RA (Reliability Analysis) Phase I simulation with two additional parameters activating the PrimeRail flow: HSIMPRIMERAIL and HSIMPRIMERAILTCL.

The IR drop analysis flow supports a choice of use models, depending on circuit size capacity that is required and accuracy/performance requirements. These flows, as depicted in Figure 14, are:

1. Recommended: HSIM simulation with back-annotation, compression and reduction of power/ground net parasitics. Post-Layout Acceleration (PLX) is optional.
2. Complete (Optional): HSIM simulation with HSIM^{plus} PWRA and PLX in addition to the Recommended flow above.
3. Minimal: HSIM standalone simulation with no back-annotation, compression or reduction of power/ground net parasitics. DLF files are required for power nets whose currents are desired as they are not back-annotated from the extracted DSPF netlist in this flow.

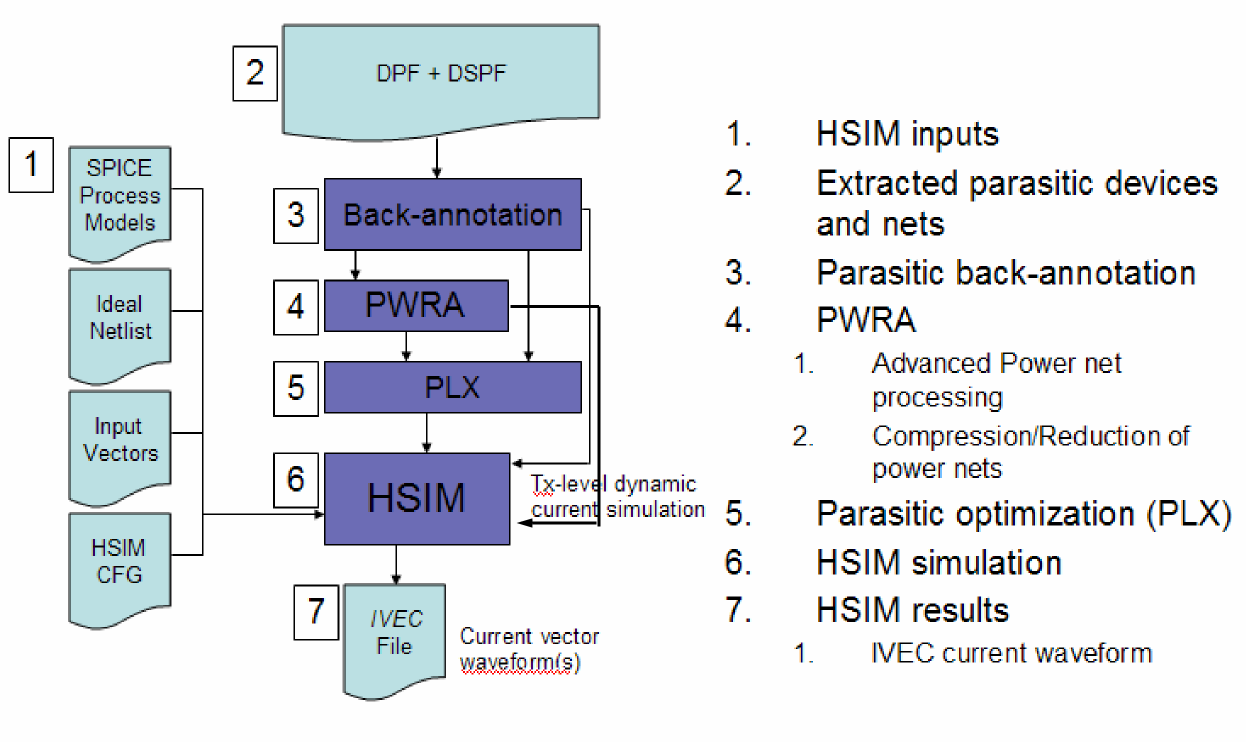

2
DPF + DSPF
1
SPICE Process Models
Ideal Netlist
Input Vectors
HSIM CFG
3
Back-annotation
4
PWRA
5
PLX
6
HSIM
Tx-level dynamic current simulation
7
IVEC File
Current vector waveform(s)
1. HSIM inputs
2. Extracted parasitic devices and nets
3. Parasitic back-annotation
4. PWRA
1. Advanced Power net processing
2. Compression/Reduction of power nets
5. Parasitic optimization (PLX)
6. HSIM simulation
7. HSIM results
1. IVEC current waveform

Figure 14 IR Drop Analysis Flows for the HSIM^{plus}-PrimeRail Interface

In the Preferred flow, for optimum accuracy and performance with power/ground bus parasitics, the following files must be input to HSIM:

- Ideal input SPICE netlist(s) (schematic netlist)
- Extracted parasitic netlist(s) (DSPF) containing both power and signal nets

- SPICE device models / technology file
- Configuration files, including the HSIM CFG and the IVEC CFG files
- Input vector stimulus / simulation vectors
- Device List file (DLF) — this file is optional

By default, HSIM back-annotates all devices and nets found in the DSPF netlist and collects all currents from devices connected to the nets defined in the IVEC CFG file. In the event that there are no extracted parasitic (DSPF) netlists available, PrimeRail supplies HSIM with a Device List File (DLF) containing all devices connected to a particular net of which currents are to be collected.

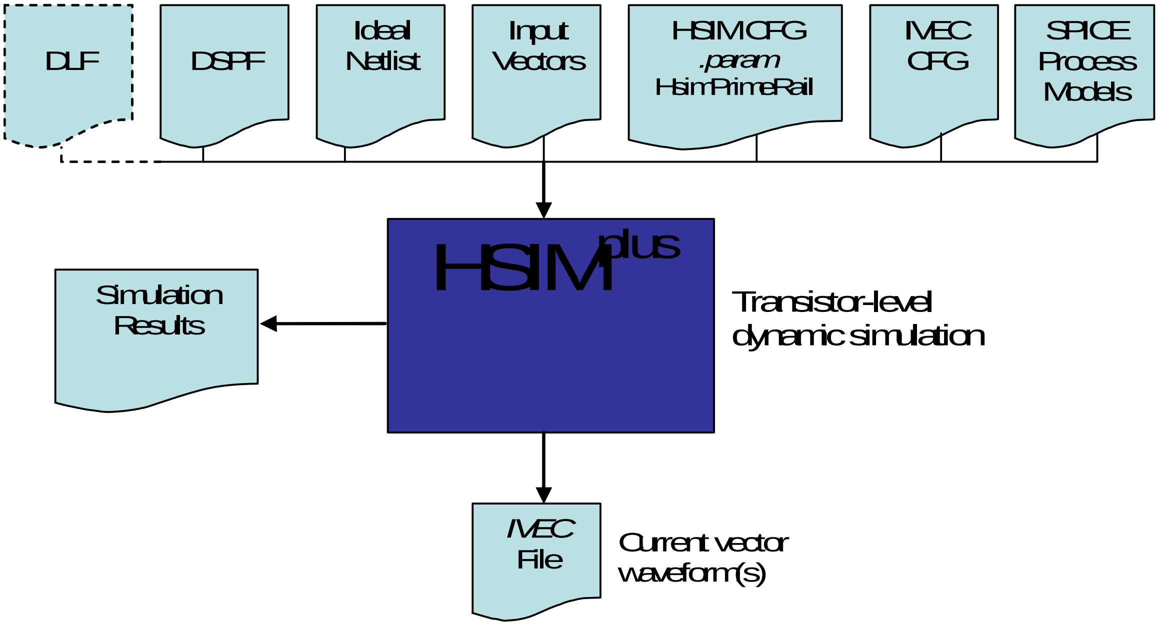


Figure 15 HSIM Simulation Flow

Note:

> HSIM parameters can be defined in the HSIM CFG file, the input SPICE deck read into HSIM, or in any file that is included (.inc) within the input SPICE deck. HSIM also reserves the hsim.ini file for defining any HSIM parameters to be used during a simulation.

In addition to all of the HSIM parameters used in the HSIM CFG file, the parameters HSIMPRIMERAIL and HSIMPRIMERAILTCL must also be included to activate HSIM in the PrimeRail flow.

HSIMPRIMERAIL

The HSIMPRIMERAIL parameter is used to indicate whether or not the HSIM^{plus} Primerail flow is activated.

Syntax

```
.param HSIMPRIMERAIL=0|1
```

HSIMPRIMERAILTCL

The HSIMPRIMERAILTCL is used to specify the name of the current vector waveform (IVEC) TCL configuration file that is generated by HSIM^{plus} and subsequently passed to PrimeRail.

Syntax

```
.param HSIMPRIMERAILTCL=<ivec_cfg_tcl_filename>
```

Combining HSIM^{plus}-PrimeRail Interface with PWRA Options

The preferred flow takes advantage of the capabilities in the HSIM^{plus} Power Net Reliability Analysis (PWRA) option. The following scenarios depict the expected behavior of the flow depending on which combination of HSIM RA parameters are used in conjunction with the PrimeRail interface.

1. HSIM^{plus}-PrimeRail Interface with Power Net Reduction (HSIMSPFPWNET) and no PWRA

   In this scenario, the CFG file contains the HSIMSPFPWNET parameter, which controls RC reduction on power nets. HSIM reduces the RC network of power nets but will not run Reliability Analysis. This command requires the hsim-pra license FEATURE but will not run Reliability Analysis unless HSIMPWRA is set.

   Here, if a net is defined in the SPF netlist, there is no need for a DLF file to be supplied for this net. HSIM back-annotates the corresponding parasitics from the SPF file and performs RC reduction to the degree set in the HSIMSPFPWNET parameter. All currents from devices connected to this net will be stored and written to the *.ivec file.

   If a net is defined in the -net section of the IVEC TCLCFG file, but it is not present in the SPF netlist, a DLF file for that net is required for HSIM to know which device currents to store.

This is the recommend flow as the design is simulated with the power net in place giving more accurate current results in the resulting *.ivec file.

For more information on the HSIMSPFPWNET parameter, refer to HSIMSPFPWNET on page 36.

2. HSIM^{plus}-PrimeRail Interface with HSIM-PWRA (HSIMSPFPWNET) Optional

   This flow is similar to the recommended flow described above, with the addition of the HSIMPWRA parameter. This parameter is set in the HSIM CFG file and it turns on the HSIM^{plus} Power Net Reliability option. You also have the choice of running with or without power net RC reduction controlled by the HSIMSPFPWNET parameter.

   If a net is defined in the IVEC CFG TCL file but is not found in the SPF netlist, a DLF file is required for that net so HSIM knows which device currents to store for devices connected to the net. Otherwise, if a net exists in the SPF netlist and is defined in the TCL file, HSIM back-annotates all parasitics, performs RC reduction if set by you, and stores currents of all of the devices connected to the net.

   In addition, HSIM also writes out the corresponding HSIM-RA files (*.ranet, *rasim, *.npf, etc.) expected after Phase I of RA simulation. These files are required if the customer has interest in running the HSIM Re-use Simulation Results Option (Phase I RA Results Re-use) described on page 155.

   Note:

   > Phase II of RA simulation is not performed if both of the HSIMPWRA and HSIMPRIMERAIL parameters are set in the HSIM CFG file.

   For more information on the HSIMPWRA parameter, refer to HSIMPWRA on page 45

3. HSIM^{plus}-PrimeRail Interface with no HSIM-RA Parameters or no HSIM-PRA License Available

   In this case, no hsim-pra license is available or the setup does not include either HSIMPWRA or HSIMSPFPWNET parameters.

   When there are not any HSIM PWRA parameters set in the HSIM CFG file, a Device List File (DLF) is required as an expected input file. HSIM reads in and back-annotates all of the nets found within the extracted SPF netlist, except those nets that are defined in the -net section of the IVEC CFG TCL file. A DLF file must be supplied for each of those nets. HSIM takes the information from the DLF file and stores currents from all of the devices connected to the net as described in the DLF file.

If a net happens to be defined in both the DLF and SPF files, HSIM does not back-annotate the net from the SPF data and runs the simulation as if no parasitics are connected to the net, instead relying on the information from the DLF file. This is similar to running HSIM-PWRA with an ideal power network (HSIMSPFPWNET=5).

The *.ivec file generated in this case only contains current information from the nets defined in the IVEC CFG TCL file and the respective devices found in the corresponding DLF files.

HSIM Re-use Simulation Results Option (Phase I RA Results Re-use)

The HSIM simulation portion of the HSIM^{plus}-PrimeRail flow is similar to that of the 1st phase of HSIM^{plus} PWRA. After Phase I of reliability analysis is completed, all of the necessary dynamic simulation results are written to files (*.ranet, *.rasim, *.npf), which are later used during Phase II RA.

When HSIM^{plus} PWRA has been previously run on a particular transistor-level design, and the dynamic simulation results are available, re-use is desirable. In this verification flow, top-level power and rail analysis using PrimeRail is desired, but rather than re-running the transistor-level simulation portion of the flow using HSIM, the re-use option can be chosen. This option simply converts the HSIM^{plus} PWRA Phase I results into *.ivec file syntax, which PrimeRail understands. The flow is shown in Figure 16 is different than the other HSIM/ PrimeRail flows, in that the re-use flow is a command line IVEC generation run. Please see usage below for more detail.

This flow is ideal if you use HSIM-RA for block-level reliability analysis and then scale-up to perform rail analysis on the top-level. Rather than rerunning the block-level transistor simulation, you can re-use your existing HSIM-RA results and ultimately feed the resulting *.ivec files into PrimeRail.

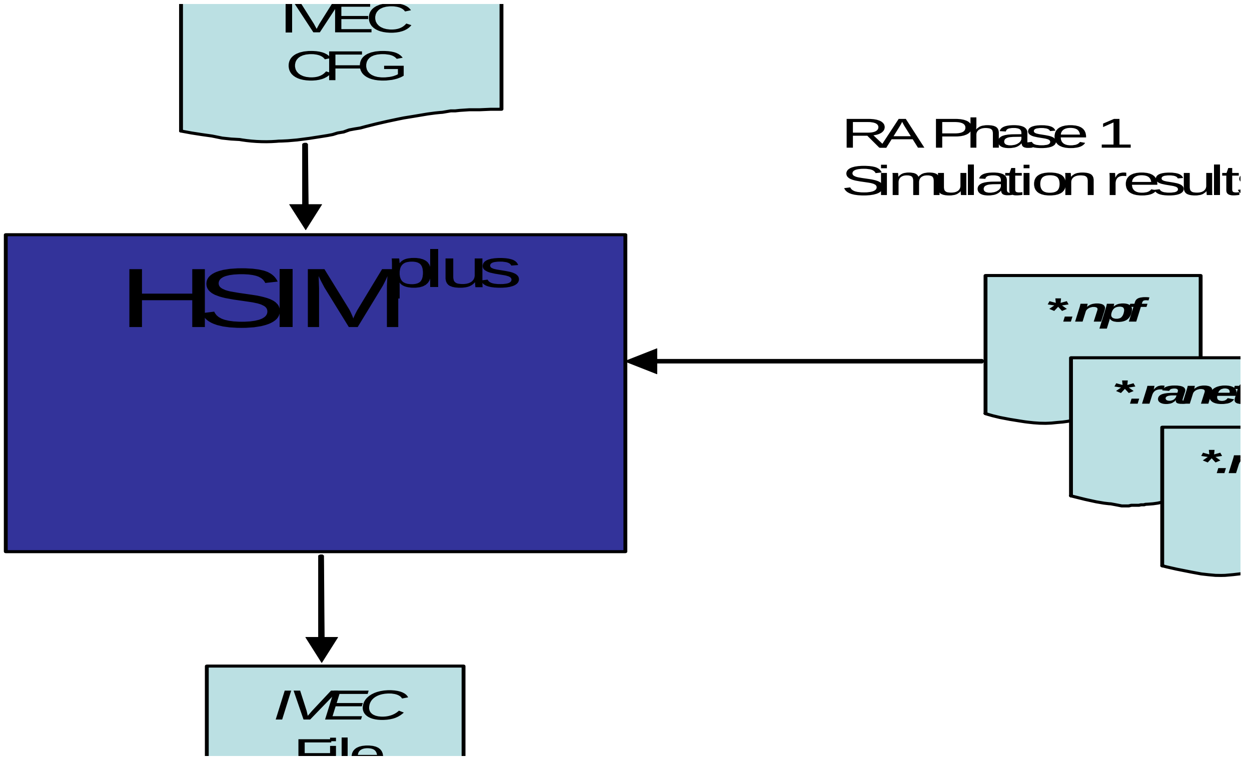


Figure 16 HSIM^{plus}-PrimeRail PWRA Re-use Option

To run HSIM with the PWRA re-use option, use the following syntax:

```
hsim -primerail <ivec_cfg_file> -i <rasim_prefix> -o
   <output_pfx>
```

IVEC CFG TCL File Syntax

The IVEC CFG TCL file read in by HSIMPRIMERAILTCL supports the following syntax:

```
time -start <start_time> -stop <end_time> -tau
   <ivec_step_size>
net -name <net_name> -dlf <DLF_filename>
input -case <0|1>
output -pres <1|10> -ctype <max|avg> -compress <0|1|2>
```

Syntax Definitions

time: (all times in ns)

start - transient start time of current information written to *.ivec file (default: 0ns)

stop - transient stop time of current information written to *.ivec file (default: stop time set by .tran statement)

tau - step size of currents written to *.ivec file (default: HSIMRATAU or 1ns)

net:

name - power net(s) to be analyzed (VDD, VSS, etc.)

dlf - PG netlist of connected device nodes (Device List File)

input:

case - automatically adjust case sensitivity to HSIM settings during IVEC generation. The possible values are 0 (case insensitive) and 1 (case sensitive). The default is 0.

output:

pres - print resolution time unit: 1(default) = 1ps; 10 = 10ps

ctype - current type: max or avg (default)

compress - 0: no compression (default); 1:UNIX compress; 2:gzip

Note:

For the Recommended, Complete and Minimal flows HSIM, by default, automatically adjusts all net names to match the case of the internal database settings. This is to account for running with case sensitive netlists in Spectre® format where case sensitivity is supported. The -case switch within the IVEC TCL file is not necessary for these flows.

However, for the RA Re-use flow, HSIM does not recall the case settings used during the simulation to generate the RA results. Therefore, you must provide HSIM with the correct case switch in the input section of the IVEC TCL file. By default, the switch is set to case insensitive (-case 0).

Example

An example of an IVEC CFG TCL file that is passed to the HSIMPRIMERAILTCL parameter is provided below. The IVEC CFG TCL file can only contain one time and one output section but can contain several net sections.

```
.param HSIMPRIMERAIL=1
.param HSIMPRIMERAILTCL=myfile.pr

myfile.pr:
----------
time -start 0 -stop 50 -tau 0.02
net -name VDD
input -case 0
output -pres 1 -ctype avg -compress 1
```

The example file measures all average currents of devices connected to net VDD, starting at time 0ns and ending at time 50ns with a time step of 20ps. The resulting *.ivec file will be compressed using UNIX compression.

HSIMRATAU vs. IVEC tau

If HSIMPWRA=0 or is not defined, HSIM takes the value from IVEC TCL -tau and uses it for the simulation and generation of the *.ivec file. HSIM does not print any warning message in this case.

If HSIMPWRA=1, HSIM takes the value from the HSIMRATAU parameter (default 1ns) and uses this value for the simulation and generation of the RA output files. If the value of IVEC TCL -tau is smaller than HSIMRATAU, HSIM may not be able to provide the desired granularity in the *.ivec file. In this case the value of .ivec tau is increased to equal HSIMRATAU and a warning message is printed.

8

CircuitCheck

Provides information on the HSIM^{plus} CircuitCheck (CCK) option and it's capabilities including: reporting potential circuit problems during the early stage of simulation, detecting incorrect input data or tool usage, analyzing latch condition before simulation starts, monitoring node voltages, generating reports, comparing waveforms to determine when and why a circuit node is triggered to change its value, computing static rise and fall delays to transistor gates, and analyzing crosstalk noise voltage caused by coupling capacitors.

Overview of CircuitCheck (CCK) Option

The CircuitCheck (CCK) option detects design problems that may lead to lengthy simulation times, and which automates circuit debugging and diagnostics checks. The capabilities include:

- For static DC paths, visual active circuit debugging (ACD) by designated DC path number through the supplied ACD browser.
- Reporting potential problems in the circuit before running simulation.
- Scanning the netlist for geometric and electrical parameter errors.
- Identifying uninitialized latches before simulation starts.
- Monitoring node voltages during a simulation and generate a report when they detect the constraints set by user.
- Tracing circuit paths to find the source of state changes.
- Computing the static rise and fall delays to transistor gates.
- Analyzing the crosstalk noise voltage caused by coupling capacitors.
- Locating leakage paths and monitor standby conditions.

The CircuitCheck commands are executed in different stages when running HSIM^{plus} and provide various reports, warning or error messages. Unless otherwise specified, the CircuitCheck commands are invoked by adding a list of CircuitCheck commands in a single file. The CircuitCheck command file must be specified in the HSIM input file using the HsimCktCheck parameter as illustrated below:

```
.param hsimCktCheck=cck_cmd_file
```

In this example, cck_cmd_file is the name of the CircuitCheck command file.

CircuitCheck Tutorial

For a comprehensive CircuitCheck Tutorial, see the CircuitCheck Tutorial on page 351.

Conventions

The following conventions are used for CircuitCheck commands and processing:

- The terms VDD (voltage source) and GND (ground) are used as generic terms. CircuitCheck does not look for VDD and GND, but rather the values they represent.
- In CircuitCheck only, a back slash character (\) indicates a continuation of the code onto the next line. When entering this code string, the back slash is omitted.
- The /* and */ comment delimiters are used in the CircuitCheck command language. The beginning is indicated by a /* and the end of a comment is indicated by a */. CircuitCheck treats everything between the /* and */ delimiters as part of the comment.

CircuitCheck Command Usage

The CircuitCheck commands provide a wide array of functions designed to support various simulation, processing, and testing options. The CircuitCheck commands have been developed in functional groups to provide various types of testing. Some of the commands are expected to be used in a group, and some provide stand-alone functions. The commands are presented in the following categories and in the following sections.

- Parametric Checks on page 162
- Design and Electrical Rules Check on page 172
- Digital Logic and Memory Diagnostics on page 234
- Timing Checks on page 250
- Dynamic Device Voltage Check on page 266
- Signal Integrity Checks on page 279
- Leakage Current Detection on page 296
- CircuitCheck Utilities on page 311

The following tables list the commands most likely to be used for the listed applications. The commands listed in the tables are not always exclusive to that application and may be used for a variety of applications.

Design and Electrical Rules Check	
cckAntGate on page 231	cckCapV on page 180
cckDiode on page 188	cckDioV on page 180
cckElemI on page 191	cckMatchSub on page 192
cckExiPath on page 192	cckFloatGateIsrc on page 194
cckMosV on page 173	cckNmosG_gt_DS on page 200
cckNmosB_gt_DS on page 196	cckNmosNodeToVdd on page 205
cckNodeVoltage on page 206	cckPathToVsrc on page 208
cckPmosB_lt_DS on page 213	cckPmosG_lt_DS on page 217
cckPmosNodeToGnd on page 223	cckResV on page 180
cckSOA on page 225	cckSubstrate on page 228
cckAntGate on page 231	tcheck mosv on page 266[a]

a. This command and its parameters are used for Dynamic Device Voltage Check during simulation .

Note:

For more information on parametric checks, see Design and Electrical Rules Check on page 172.

Digital Logic and Memory Diagnostics	
cckFlashcore on page 235	cckLatchUnInit on page 236
cckLatchInElem on page 238 cckLatchSkipElem on page 238[a]	cckMaxStackUpNmos on page 239
cckMaxStackUpPmos on page 239	cckMaxStuckAt on page 240
cckToggleCount on page 241	ntrig on page 242
intrig on page 247	

a. These are sub-commands for cckLatchUnInit

Note:

For more information on signal integrity checks, see Digital Logic and Memory Diagnostics on page 234.

Timing Checks	
cckMaxNmosToVdd on page 251	cckMaxPmosToGnd on page 252
cckMaxStackUpNmos on page 239	cckMaxStackUpPmos on page 239
cckMeasPathDelay on page 253	cckNodeMaxRF on page 255
cckParasiticRC on page 170	cckRCDlyPath on page 257[a]

a. See cckRCDlyPath for a description of the group of commands needed to support this function.

Note:

For more information on leakage current detection, see Timing Checks on page 250.

Parametric Checks	
cckParam on page 162	cckParasiticRC on page 170

Note:

For more information on signal integrity checks, see Signal Integrity Checks on page 279.

Signal Integrity Checks	
cckDXtalk on page 280[a]	cckParasiticRC on page 170
cckStaticXtalk_GroupCmd on page 287[b]	

[a] *This is the Dynamic Crosstalk command.*

[b] *This is a link to a series of commands that are used as a group to test Static Crosstalk.*

Note:

For more information on parametric checks, see Parametric Checks on page 162.

Leakage Current Detection	
cckAnalogPDown on page 305	cckAnalogPDownIth on page 305
cckElemI on page 191	cckExiPath on page 192
cckMaxStaticLeak on page 297	cckOffLeakI on page 298
cckStaticHZNode on page 308	cckStaticDCPath on page 310

Note:

For more information on leakage current detection, see Leakage Current Detection on page 296.

CircuitCheck Utilities	
cckBasic on page 311	cckCompareOp on page 312
cckMatchSub on page 314	cckPatternMatch on page 316
cckPatternConstraint on page 317	cckSetMosDir on page 259
cckTgPair on page 319	

Note:

For more information on CircuitCheck utilities, see CircuitCheck Utilities on page 338.

Table 1 lists commonly used commands for basic design, electrical parameters, and logic check as sorted by analysis type.

Table 1 Basic Design, Electrical Parameters, and Logic Check

Static Analysis	Dynamic Analysis	Multiple Mode
cckAntGate	cckElemI	cckDiode
cckCapV	cckExiPath	cckSubstrate
cckDioV	cckFlashCore	
cckFloatGateIsrc	cckLatchInElem	
cckMatchSub	cckLatchSkipElem	
cckMaxPmosToGnd	cckLatchUninit	
cckMaxStackUpNmos	cckMeasPathDelay	
cckMaxStackUpPmos	cckNodeVoltage	
cckMaxStuckAt	cckSOA	
cckMosV	cckToggleCount	
cckNmosB_gt_DS	Intrig	
cckNmosG_gt_DS	Ntrig	
cckNmosNodeToVdd	Tcheck mosv	
cckParasiticRC		
cckPathToVsrc		
cckPmosB_lt_DS		
CckPmosG_lt_DS		
cckPmosNodeToGnd		

Table 1 Basic Design, Electrical Parameters, and Logic Check

Static Analysis	Dynamic Analysis	Multiple Mode
cckRCDlypath		
cckRCDlypath		
ckMaxNmosToVdd		

Table 2 lists commonly used commands for device dependent characteristics, as sorted by analysis type.

Table 2 Device Dependent Characteristics

Static	Dynamic
cckBasic	cckAnalogPdown
cckMaxStaticLeak	cckAnalogPDownIth
cckParam	cckCompareOp
cckparasiticRC	cckDXtalk
cckPattern***	cckExiPath
cckPatternMatch	cckOffLeakI
cckSetMosDir	
cckStaticXtalk_GroupC	
cckTgPair	

Specifying Circuit Checks in Command Files

Circuit Check allows you to specify static or dynamic check commands to conduct specific analyses throughout the simulation. To use this capability, you need to group all the commands in a file and add the following command in the netlist:

```
.param hsimCktCheck=<cck_cmd_filename>
```

For example, suppose you want to conduct a CCK static floating gate check. Create a `cck.cmd` CCK command file and add the following content to the file:

```
cckFloatGateIsrc 1
```

Then in the netlist you also need to add a statement for CCK command file specification:

```
.param hsimCktCheck=cck.cmd
```

In addition to the static and dynamic CCK commands, there is a special group of commands that conduct dynamic device voltage checks (see the Dynamic Device Voltage Check on page 294 for details). You must group all the related dynamic device voltage check commands in a command file and specify it in the netlist with the following statement:

```
.param hsimDeviceV=<deviceV_cmd_filename>
```

For example, suppose you want to conduct a CCK dynamic mosv check. Create a `deviceV.cmd`CCK command file with the following content in the file:

```
.tcheck test1 mosv model=* lvgs=-0.6
```

Then in the netlist you also need to add a statement for CCK command file specification:

```
.param hsimDeviceV=deviceV.cmd
```

Running Circuit Check Operations without DC Initialization and Transient Simulation

Most of the static CCK commands conduct analyses are based on netlist connectivity data and do not require DC initialization or transient simulation results. By default, HSIM^{plus} always conducts full DC initialization and transient simulation whether CCK commands are used or not. However, sometimes you only want to focus on static CCK checks, so a `cckNoSimu` control parameter (described in the next section) is available to allow HSIM^{plus} to skip the DC initialization and transient steps.

cckNoSimu

Accelerates the static CCK commands by skipping DC initialization and transient simulation.

Syntax

```
cckNoSimu [0|1] level=[0|1|2]
```

Description

Lets you run static CCK commands without running DC initialization or transient simulation first.

Arguments

Argument	Description
`level=0`	Runs both DC initialization and transient simulation (same as `0` .)
`level=1`	Disables transient simulation (same as `1`.)
`level=2`	Disables DC initialization, transient simulation, and partitioning (building the circuit database).

Examples

The following CCK command file conducts static floating gate analysis based on the netlist connectivity database and skips transient simulation:

```
cckNoSimu 1

cckFloatGateIsrc 1
```

or

```
cckNoSimu level=1

cckFloatGateIsrc 1
```

The following CCK command file conducts static floating gate analysis based on the netlist connectivity database and goes directly to the end of the simulation process by skipping DC initialization, transient simulation, and partitioning (building the circuit database).

```
cckNoSimu level=2

cckFloatGateIsrc 1
```

Note:

> The cckNoSimu command has no effect with the cckStaticXTalk and cckRCDlyPath static CCK commands, which conduct static crosstalk glitch analysis and static path delay analysis, respectively. If you use one or both

commands in the CCK command file, HSIM^{plus} automatically disables transient simulation even without the cckNosimu command. Do not use the cckStaticXTalk and cckRCDlyPath commands with other static CCK commands to try to save memory and runtime. For example:

```
cckNoSimu level=2

cckFloatGateIsrc 1

cckRCDlyPath 1
```

Disables transient simulation only because cckRCDlyPath is specified, but still runs DC initialization and builds the circuit database. so these steps do not achieve the original goal to save memory and runtime.

You need to properly specify cckNoSimu to avoid design flow problems. For example, if you conduct a static floating gate check and dynamic device voltage check at the same and include cckNoSimu, HSIM^{plus} runs but does not perform a dynamic check because DC initialization and transient simulation are disabled.

Passing Parameters Into CircuitCheck Commands

To provide the freedom to organize HSIM^{plus} inputs and CircuitCheck command files, the parameter-passing capabilities in both HSIM^{plus} and CircuitCheck are provided in the CircuitCheck Command groups.

Parameter values are passed into CircuitCheck command files by specifying them in either the input netlist file or CircuitCheck command files.

Parameters specified in the netlist or CircuitCheck command file are subject to the following constraints:

1. The .param statement in the HSIM input netlist or the CircuitCheck Command file must follow spice format; i.e., a .param is only assigned a value or an expression. However, string is not allowed in a parameter specification.
2. If the same parameter is specified in the netlist file, via .param HSIMCKTCHECK=cck_cmd_File, or via .param HSIMDEVICEV=dev_v_file, there are priority rules for Circuit Check to pick the values:
   - In non-post-process cases:

.param in HSIM netlist > .param HSIMCKTCHECK=cck_cmd_File > .param HSIMDEVICEV=dev_v_file

- In post-process cases:

  .param in HSIM netlist > .param HSIMDEVICEV=dev_v_file > .param HSIMCKTCHECK=cck_cmd_File

When the following parameter definitions are added into the HSIM input file, the CircuitCheck command files use these values and not the CircuitCheck command values.

Example 31 Parameter Definitions

```
.param min=1.0
.param base='min + 0.5'
.param max='2.0*min + base'
.param duration='3*(-min+max)'
.param HsimCktCheck=CircuitCheck.cck
```

Where `min` is an assigned value, and `base`, `max`, and `duration` are all given with an expression. These expressions are converted to values by the HSIM parser.

CircuitCheck.cck

In this CircuitCheck command file example, the parameters vg and vs are defined using the parameter values passed from the netlist file. For example:

Example 32 CircuitCheck Command File

```
.param vg=base+min
.param vs=-min+2*max
cckMosv tag model=p vhth=min lvd=base uvd=max
cond=‘(uvg<vg || lvd>=vs)’
```

If the same parameter is specified in multiple places, priority rules of parameter-passing apply, and a Warning message is issued in the log file, such as:

```
WARNING '.param vdd1' has already been defined, ignore!!
```

Note:

In the cond expression, combinations of parameters such as base+min, -min+2*max are also supported.

In the above example, the parameter definitions are specified in either netlist file or CircuitCheck command file.

Include Statements

The include statement can be used to include files into CircuitCheck command files. Multiple include statements are allowed in a CircuitCheck command file. For example:

```
/* file: cck_cmd */
include cck_cmd1
cckPmosG_lt_DS

/* file: cck_cmd1 */
cckNmosG_gt_DS
include cck_cmd2

/* file: cck_cmd2 */
cckMosv plv_vgs1 model=plv cond='lvg-uvs < -1.7' rpttrace=1
```

Parametric Checks

These CircuitCheck commands are designed to help check min/max device dimensions, temperature, post-layout parasitic effects, etc.

Check Electrical Parameters

cckParam

The cckParam command checks for:

- Normal element size
- Correct model usage
- Reasonable temperature usage

CircuitCheck commands are executed in different stages when running HSIM. Figure 15 graphically illustrates the high and low limits where warning or error messages are generated and when required, CircuitCheck stops. When data being checked exceeds CircuitCheck's soft upper bound a warning message is issued. As the data abnormality exceeds CircuitCheck's absolute upper bound, an error message is generated and CircuitCheck stops.

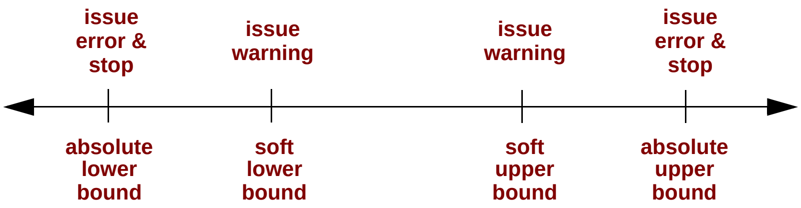


Figure 17 CircuitCheck Checking Methodology

When a very large MOSFET width is detected such as 1 meter which is above the absolute upper bound, it will be considered an error and simulation will stop. If the width appears to be larger than ordinary, but has not reached the absolute upper bound, a warning message will be generated and checking will continue.

CircuitCheck checks the following device parameters:

<erMaxCap=v>	<waMaxCap=v>
<erMaxMosW=v>	<waMaxMosW=v>
<erMinMosW=v>	<waMinMosW=v>
<erMaxMosL=v>	<waMaxMosL=v>
<erMinMosL=v>	<waMinMosL=v>
<erMaxMosAD=v>	<waMaxMosAD=v>
<erMaxMosAS=v>	<waMaxMosAS=v>
<erMaxMosPD=v>	<waMaxMosPD=v>
<erMaxMosPS=v>	<waMaxMosPS=v>
<erMaxMosTox=v>	<waMaxMosTox=v>
<erMinMosTox=v>	<waMinMosTox=v>
<erMaxTemp=v>	<waMaxTemp=v>
<erMinTemp=v>	<waMinTemp=v>

<erMaxDiodeW=v>	<waMaxDiodeW=v>
<erMaxDiodeL=v>	<waMaxDiodeL=v>
<erMaxDiodeA=v>	<waMaxDiodeA=v>
<erMinDiodeW=v>	<waMinDiodeW=v>
<erMinDiodeL=v>	<waMinDiodeL=v>
<erMinDiodeA=v>	<waMinDiodeA=v>
<erMinMfactor=v>	<waMinMfactor=v>
<model=m>	

Note:

Resistor size is checked by using the hsimrmin and hsimrmax commands.

CircuitCheck parameters are described in the following sections.

Capacitor Values

Cap (Capacitor) has two bounds:

- erMaxCap: Absolute upper bound
- waMaxCap: Soft-upper bound

The results of a capacitor value causes one of the following to occur:

- HSIM stops: If a capacitor exceeds the absolute upper bound.
- Prints warning message: If the capacitor is larger than the soft-upper bound, but not the absolute upper bound.

The error message and warning message will be reported in hsim.cck.

MOSFET Width

There are four bounds associated with MOSFET width:

- erMaxMosW: Absolute upper bound
- waMaxMosW: Soft-upper bound

- erMinMosW: Absolute lower bound
- waMinMosW: Soft-lower bound

MOSFET Width: Upper Bounds If the width of a MOSFET is larger than erMaxMosW, an ERROR message is reported and HSIM is stopped. If the width is between waMaxMosW and erMaxMosW, a Warning message is printed.

MOSFET Width: Lower Bounds If the width of a MOSFET is smaller than erMinMosW, an ERROR message is reported and HSIM is stopped. If the width of a MOSFET is between waMinMosW and erMinMosW, a Warning message is printed.

MOSFET Length

MOSFET length has the following bounds:

- erMaxMosL: Absolute upper bound
- waMaxMosL: Soft-upper bound
- erMinMosL: Absolute lower bound
- waMinMosL: Soft-lower bound

MOSFET Length: Upper Bounds If MOSFET length exceeds erMaxMosL, HSIM is stopped. If the length is between waMaxMosL and erMaxMosL, a Warning message is printed.

MOSFET Length: Lower Bounds If the length of a MOSFET is smaller than erMinMosL, HSIM is stopped. If the width of a MOSFET is between waMinMosL and erMinMosL, a Warning message is printed.

MOSFET Drain/Source Area and Drain/Source Perimeter

MOSFET Drain Area has the following bounds:

- erMaxMosAD: Absolute upper bound
- waMaxMosAD: Soft-upper bound

MOSFET Source Area has the following bounds:

- erMaxMosAS: Absolute upper bound
- waMaxMosAS: Soft-upper bound

MOSFET Drain Perimeter has the following bounds:

- erMaxMosPD: Absolute upper bound
- waMaxMosPD: Soft-upper bound

MOSFET Source Perimeter has the following bounds:

- erMaxMosPS: Absolute upper bound
- waMaxMosPS: Soft-upper bound

MOSFET Gate Oxide Thickness

The MOSFET tox is checked. The upper and lower bounds are set to detect design problems as follows:

- erMaxMosTox: Absolute upper bound
- waMaxMosTox: Soft-upper bound
- erMinMosTox: Absolute lower bound
- waMinMosTox: Soft-lower bound

Diode Width, Length, and Area

The twelve values associated with diode are:

- erMaxDiodeW: Absolute upper bound for diode width
- waMaxDiodeW: Soft upper bound for diode width
- erMaxDiodeL: Absolute upper bound for diode length
- waMaxDiodeL: Soft upper bound for diode length
- erMaxDiodeA: Absolute upper bound for diode area
- waMaxDiodeA: Soft upper bound for diode area
- erMinDiodeW: Absolute lower bound for diode width
- waMinDiodeW: Soft lower bound for diode width
- erMinDiodeL: Absolute lower bound for diode length
- waMinDiodeL: Soft lower bound for diode length

- erMinDiodeA: Absolute lower bound for diode area
- waMinDiodeA: Soft lower bound for diode area

Simulation Run Temperature

Simulation run temperature is checked. The upper and lower bounds are set to detect simulation problems as follows:

- erMaxTemp: Absolute upper bound
- waMaxTemp: Soft-upper bound
- erMinTemp: Absolute lower bound
- waMinTemp: Soft-lower bound

If the temperature exceeds the absolute upper bound defined as erMaxTemp, HSIM is stopped.

Model

Allow different parameter values for different models. Each line contains all the specific values for each individual model.

The default values are defined in the table below:

Table 3 Model Default Parameter Values

Default Value	**Measurement Unit**
erMaxCap=0.001	units=F
waMaxCap=1.e-8	units=F. This value is 10000 pF
erMaxMosW=0.01	units=meter. This value is 10000 um.
waMaxMosW=0.001	units=meter. This amounts to 1000 um.
erMinMosW=1.e-8	units=meter. This amounts to 0.01 um.
waMinMosW=1.e-7	units=meter. This amounts to 0.1um.
erMaxMosL=0.01	units=meter
waMaxMosL=0.001	units=meter. This amounts to 1000 um.
erMinMosL=1.e-8	units=meter.This value is 0.01 um.

Table 3 Model Default Parameter Values (Continued)

Default Value	**Measurement Unit**
waMinMosL=1.e-7	units=meter.This amounts to 0.1 um.
erMaxMosAD=0.0001	units=meter square
waMaxMosAD=1.e-6	units=meter square
erMaxMosAS=0.0001	units=meter square
waMaxMosAS=1.e-6	units=meter square
erMaxMosPD=0.01	units=meter. This value is 10000 um.
waMaxMosPD=0.001	units=meter
erMaxMosPS=0.01	units=meter
waMaxMosPS=0.001	units=meter
erMaxMosTox=5.e-8	units=meter
waMaxMosTox=3e-8	units=meter
erMinMosTox=5.e-10	units=meter
waMinMosTox=5.e-9	units=meter
erMaxDiodeW=1.e-2	units=meter
waMaxDiodeW=1.e-3	units=meter
erMaxDiodeL=1.e-2	units=meter
waMaxDiodeL=1.e-3	units=meter
erMaxDiodeA=1.e-4	units=meter square
waMaxDiodeA=1.e-6	units=meter square
erMinDiodeW=1.e-8	unit=meter
waMinDiodeW=1.e-7	unit=meter

Table 3 Model Default Parameter Values (Continued)

Default Value	Measurement Unit
erMinDiodeL=1.e-8	unit=meter
waMinDiodeL=1.e-7	unit=meter
erMinDiodeA=1.e-16	unit=meter square
waMinDiodeA=1.e-14	unit=meter square
erMaxTemp=500	units=Celsius
waMaxTemp=100	units=Celsius
erMinTemp=-200	units=Celsius
waMinTemp=-100	units=Celsius
erMinMfactor=-0.001	
waMinMfactor=0	

These defaults may be overridden by using the commands listed above.

The following is an example of the `cckParam` command.

Example 33 cckParam Command

```
cckParam erMaxCap=2000p waMaxCap=1.e-9
cckParam erMaxMosW=0.1 waMaxMosW=0.03 erMinMosW=0.01u
waMinMosW=0.1u
```

A capacitor with these characteristics is reported as follows:

- An ERROR message is reported for a value > 2000 pf (picofarad).
- A Warning message is printed for capacitor values between 1000 pF and 2000 pF.

A MOSFET with these widths is reported as follows:

- HSIM stops and an ERROR message is reported if its width is one of the following:
  - > 0.1 m

    - < 0.01 um.
- A Warning message is printed if it’s width is between either of the following:
    - 0.03 meter and 0.1 meter.
    - 0.01 micron and 0.1 micron.

The following is a sample output (hsim.cck or out_file.cck) resulting from the command shown in the previous example.

Example 34 Output File

```
**************************************
* Check Elem Size, Model, Temperature
**************************************
Warning: find a mos (mi8x) with unusual width (0.04 M) in subckt
(fuse)
Error: can't allow mos (mc2, width 0.155 M) in subckt (fuse10)
Warning: find a mos (mi8) with unusual width (0.05 M) in subckt
(cgdv)
Error: can't allow too large a capacitor (cext, 3e-6 F) in subckt
(TLC)
```

In the sample file shown above, the following definitions apply:

- mi8x: Element name
- fuse: Sub-circuit name
- cext: Capacitor name
- TLC: Top-Level Circuit

Limiting the Number of Violations Reported

The "num" parameter limits the number of violations reported, for avoiding lengthy output files. For example, "cckParam num=10" allows up to 10 warnings or errors in the violation report. The default number is 300.

M-factor

If there are elements with a zero or negative m-factor, warnings are reported. The following is a sample .cck output:

```
Warning: find a mos (xt.xcore.x256m4) with unusual m factor (0)
in subckt (epju)
Warning: find a mos (xt.xbist.xm4) with unusual m factor (0) in
subckt (enll)
```

When HSIM checks parameters, it creates a default template containing all the default values, as indicated in Table on page 161. This template is used to examine all the transistors. If any default needs to be changed, simply specify a cckParam statement without any model keyword, as in the following example:

```
cckParam erMaxMosW=0.05
cckParam model=nch1 erMaxMosW=0.06
```

For example, the first statement of cckParam overwrites the value of erMaxMosW to 0.05 meter. The second statement has a keyword model=nch1, which creates a new template consisting of the default values, except that erMaxMosW is changed to 0.06 meter for model nch1.

It is suggested to group the parameters for general purpose checking first. Those values will overwrite the default settings. Then, create a new statement with model and the associated values. If the line is too long, a back slash (\) is used to indicate the continuation of the line. Parameters for each model must be on a separate line.

Post-Layout RC Checking

cckParasiticRC

Reports data about the parasitic RC during netlist loading to aid in post layout circuit debugging.

Syntax

```
cckParasiticRC <rmin=rmin_value> <cmin=cmin_value>
   <ccmin=ccmin_value> <numr=num_of_r> <numc=num_of_c>
   <numcc=num_of_cc> <outFile=output_file_name>
```

Parameters

rmin

Minimum value of parasitic resistors to report. The default is rmin=0.

cmin

Minimum value of parasitic of grounded capacitors to report The default is cmin=0.

ccmin

Minimum value of parasitic coupling capacitors to report. The default is ccmin=0.

numr

Maximum number of resistors to report. The default is numr=100.

numc

Maximum number of grounded capacitors to report. The default is numc=100.

numcc

Maximum number of coupling capacitors to report. The default is numcc=100.

outFile

Output file name for saving Warning messages. The default is parasiticRC.cck.

Description

`cckParasiticRC` reports data about the parasitic RC during netlist loading to aid in post layout circuit debugging. If the resistance, coupling capacitance and node capacitance are greater than the predetermined values specified in the rmin, cmin and ccmin respectively, cckParasiticRC will report the nodes along with the corresponding resistor or capacitor values. This report is stored in the file, output_file_name, which is specified in the outFile. The default output file name is parasiticRC.cck.

Example

The following is an example of adding cckParasiticRC to the CircuitCheck command file:

```
cckParasiticRC rmin=1 cmin=0.1p ccmin=0.1p numr=100 numc=200
numcc=200 outFile=”tt.out”
```

After simulation is completed, HSIM reports the resistors, capacitors, and coupling capacitors with values greater than 1 Ohm, 0.1pF, and 0.1pF respectively as shown in the following report. The number of resistor nodes and capacitor nodes are limited to 100 and 200 respectively due to the numr, numc and numcc setup. The statistical results are stored in the file, tt.out. The following shows a portion of the report from the above example.

```
* Check parasitic R
r3215 n_14:5944 n_14:f258 r=143.889
r1197 n_10:2332 n_10:f524 r=143.889
* Check parasitic C
c44127 vdd:4600 vdd:4522 c=3.49101e-017
c40397 vdd:8035 vdd:7957 c=3.49101e-017
* Check parasitic CC
cg8188 n_9:1473 0 cc=3.64899e-017
cg32864 n_15:7189 0 cc=3.70265e-017
cckParasiticRC rmin=1 cmin=0.1p ccmin=0.1p numr=100 numc=200
numcc=200 outFile=”tt.out”
```

Design and Electrical Rules Check

These CircuitCheck commands are designed to help check over-voltage conditions, excessive device currents, forward-biased diodes, shorts to VDD or GND, missing VDD and GND connections, floating gates, and voltage-domain interface errors.

Static Device Voltage Analysis

The concept of static analysis is the ability to propagate characteristics of the design, in this case voltages, throughout the whole design without the need of vectors or transient simulation. To propagate voltages through the design it is necessary to define rules associated with the propagation. These propagation rules are used to determine if a voltage (or voltage range) can be seen by a node through a network of design elements like MOSFETs, resistors, capacitors, and diodes. Using these rules HSIM^{plus}, along with the Circuit Check Option, can analyze voltage biasing conditions and report user-defined violations, with a minimum of a design netlist and supply voltages defined.

By performing static voltage propagation you can apply electrical rule checks to determine if damaging voltage biases exists. Because the voltage propagation is performed statically, the coverage is greater than what can be achieved via dynamic simulation, ensuring more violations are reported. Ultimately more corrections can be implemented to ensure design correctness.

Device Voltage Analysis for Transistors

cckMosV

cckMosV performs a static check on user-specified transistors. If a violation occurs, a warning is printed to the hsim.cckmosv or output.cckmosv report file. Also, refer to Device Voltage Analysis for Capacitor, Resistor and Diode on page 190.

Syntax

```
cckMosV tagName <model=model_name> <subckt=subckt_name>
   <inst=inst_name> <rmSub=subckt_name> <rmInst=inst_name>
   <skipSub=suckt_name> <skipInst=inst_name>
   <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1> <vpth=v2>
   <lvd=v7> <uvd=v8> <lvg=v9> <uvg=v10> <lvs=v11> <uvs=v12>
   <lvb=v13>  <uvb=v14><cond='expression'> <num=n>
   <separate_file=[0|1]> <rptTrace=[0|1]>
   <risePmosFallNmos=[0|1]> <subinfo=[0|1]>
   <filterAlert=[0|1]> <pwl_time=time>
   <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
   <-connSub=subckt_name> <-connInst=inst_name>
   <-connNode=node_name> <autoFvsrcnd=[0|1]>
```

Parameters

tagName

tagName is a label in the report file to make searching easier.

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck does not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

limitMos=<number>

limitMos defines the expansion limit from the given voltage source that is not to exceed <number> transistors. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vlth=<vl>

The low voltage threshold. If vlth is specified, then CircuitCheck will trace from the voltage sources with values less than or equal to <vl>. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vhth=<vh>

The high voltage threshold. If vhth is specified, then CircuitCheck will trace only from sources with values equal to or greater than <vh>. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vnth=<v1>

vnth specifies the n-MOSFET threshold voltage.The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n_MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vpth=<v2>

vpth specifies the p-MOSFET threshold voltage.The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvd=<v7>

The lower bound of drain node voltage to be <v7>. If a device's drain terminal is less than <v7> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

uvd=<v8>

The drain node voltage upper boundary. If a device's drain terminal is greater than <v8> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvg=<v9>

The gate node voltage lower boundary. If a device's gate terminal is less than <v9> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

uvg=<v10>

The gate node voltage upper boundary. If a device’s gate terminal is greater than <v10> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvs=<v11>

The source node voltage lower boundary. If a device's source terminal is less than <v11> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

uvs=<v12>

The source node voltage upper boundary. If a device's source terminal is greater than <v12> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvb=<v13>

The bulk node voltage lower boundary. If a device's bulk terminal is less than <v13> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

uvb=<v14>

The bulk node voltage upper boundary. If a device's bulk terminal is greater than <v14> a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

cond=expression

A voltage constraint can be a Boolean expression such as: `cond='(uvp < 1 || lvn >= 0.5)` or mathematical operations (+,-,*, /, and abs()). Operators, +,-,*, / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, || ('or' operator). When this expression is true, a warning is generated.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

separate_file=[0|1]

separate_file=0 is the default which causes all warning messages to be reported in the <hsim_output_prefix>.cck file. If separate_file=1, warning messages are reported in a separate <hsim_output_prefix>.cckmosv_<command_tag>file. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

risePmosFallNmos=[0|1]

risePmosFallNmos indicates whether (1) or not (0) Device Voltage Analysis propagation applies restrictions with positive, zero, or negative voltage sources relative to element type. If risePmosFallNmos is set to 1, positive voltage sources can only be propagated through p-MOSFET devices, and negative or zero voltage sources can only be propagated through n-MOSFET devices. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

–connSub=subckt_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified subcircuit name.

–connInst=inst_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

–connNode=node_name

Limits elements to be reported to those that have a direct connection to the specified node name.

autoFvsrcnd=[0|1]

If set to 1 enables the automatic pwl voltage range feature. This feature conducts the following additional steps before the static check:

1. Search for any pwl or pulse Vsrc definition in the netlist
2. For any found pwl or pulse Vsrc, obtain its voltage range info (Vmin, Vmax)
3. Based on the obtained voltage range information, perform an operation equivalent to "fvsrcnd(<VsrcNodeName>, Vmin, Vmax)"

The default is 0.

Note:

Among the -connSub, -connInst, and -connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Note:

vlth and vhth act like filters; not like checking criteria. Their purpose is to filter out other voltage sources so that cckMosV starts tracing only with voltage sources lower than vlth or higher than vhth. The actual checking criteria are specified in lvg, uvg, lvd, uvd, lvs, and uvs.

Note:

> Circuit Check can pass other variables defined by .param into CCK commands. If users want to define a variable, this variable must be defined by .param and then be passed into CCK commands.

Examples

The following is an example of passing user-defined parameters into a CCK command:

```
.param vxx=2.5
cckMosV tagName inst=inv cond='(lvd*3 - uvs >= -vxx)'
```

The following is an example of using the cond= options:

```
(a) cond='(lvd - 0.5*uvs >= 1.5)'
(b) cond='(lvd >= 0.5*uvg + 1.5)'
(c) cond ='(uvd-lvs*0.5 > 1.5 *2)'
(d) cond ='(uvd-abs(lvs-lvg) > 1.5 *2)'
```

The following are static device voltage checking examples:

```
cckMosv tn subckt=a0tc inst=* uvg=2.9 limitMos=2
risePmosFallNmos=1 skipInst=aa*
```

In the above example:

- All instances of a0tc are checked because subckt=a0tc inst=*.
- Since risePmosFallNmos=1, when a voltage source is traced then:
- Positive: Go through p-MOSFET then, only go through p-MOSFETS.
- GND or Negative: Only go through n-MOSFET.
- Since skipInst=aa*, when tracing, instances with the prefix aa are not gone through.
- Since limitMos=2, only nodes reached from a voltage source by going through no more than 2 transistors are considered.
- Transistor node voltages are then checked to see if their gate node exceeds 2.9 V. Transistors that meet this criteria are reported.

  ```
  cckMosv tn subckt=b1tr inst=* vlth=0.4 uvg=2.9
  ```

In the above example, since subckt=b1tr and inst=*, all transistors in all the instantiations of b1tr are checked. Since vlth=0.4, the power supplies with less than or equal t 0.4 V are traced. since limitMos is not set, any node can be traced. If a transistor's gate node exceeds 2.9 V, then it is reported.

```
cckMosV tag model=nch cond='(uvg>=0.4||lvd <-1.8)'
```

In the above example, the syntax checks all the nch transistors. If a nch MOSFET has gate node voltage larger or equal to 0.4 V or drain voltage smaller than -1.8 V, then it is reported.

```
cckMosV tag model=nch vlth=0.5 cond='(uvg >=0.4||lvd <-1.8)'
```

In the above example, the syntax traces from nch MOSFET to see if it will reach any logic-0 voltage sources. If the upper bound of voltage source reached by gate node is larger than 0.4 V or the lower bound of voltage sources seen by a drain node is less than -1.8 V, then it is reported.

```
cckMosV tpc model=pch inst=* vhth=0.7 cond='(lvg <=1.1||uvd >1.8)'
```

In the above example, the syntax checks pch MOSFETs to find the logic-1voltage source range using vhth=0.7 V as seen from the gate node. If the lower bound is less than or equal to 1.1 V, it is reported. Similarly, the upper voltage source bound seen from the drain node can be checked. If it is larger than 1.8 V, then it is reported.

```
cckMosV tn model=nch inst=* vlth=0.2 uvg=0.3 limitMos=2
risePmosFallNmos=1 skipInst=xram.xwr2_7*
```

In the above example, nch transistors are checked to determine if their nodes can reach logic-low voltage source with values less than 0.2V. When tracing, the process can only go through two MOSFETS and must go through a nch MOSFET. If a gate node voltage is greater than 0.3V, it is reported.

Note:

Tracing can not go through any MOSFETS named xram.xwr2_7*.

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx       0 pvdd
vvde vvdex       0 pwl 0ns 0 0.1ns pvdd
vshift shift       0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckMosv vhth=0.5 vnth=0.3 vt=0.1 autoFvsrcnd=1
```

CCK operates equivalently to following syntax:

```
cckMosv vhth=0.5 vnth=0.3 vt=0.1 fvsrcnd=(vvdex, 0, 1.11)
fvsrcnd=(shift, 0, 0.8)
```

The report header is updated to include the autoFvsrcnd setting:

```
*****************************************************
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*  vnth=0.300 vpth=-0.500
*  vhth=0.500
*
*  vt=0.10 num=300
*
*  autoFvsrcnd=1
*  format: instName gate_volt d/s_volt
*****************************************************
```

Device Voltage Analysis for Capacitor, Resistor and Diode

With some analog designs, it is important to check the voltage for 2-terminal devices such as capacitors, resistors, and diodes. cckCapV performs a static voltage check on user-specified capacitors similarly to the tests conducted by cckMosV. A warning message is printed to the hsim.cckcapv or output.cckcapv report file if a violation occurs.

cckResV and cckDioV perform static voltage checking for specified resistors and diodes. A warning message is reported in either the .cckResV or .cckDioV report files if a violation occurs.

The syntax and parameters for cckCapV, cckDioV, and cckResV are as follows:

cckCapV

cckCapv is used to perform analysis checks on capacitors. Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared against your defined rules.

Syntax

```
cckCapV tagName <model=m> <subckt=subckt_name>
   <inst=instName> <rmSub=subckt_name> <rmInst=instName>
   <skipSub=subckt_name> <skipInst=instName>
   <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1> <vpth=v2>
   <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6> <cond='expression'>
   <num=n> <separate_file=[0|1]> <rptTrace=[0|1]>
   <risePmosFallNmos=[0|1]> <subinfo=[0|1]>
   <filterAlert=[0|1]> <pwl_time=time>
   <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
```

cckDioV

cckDioV is used to perform analysis checks on diodes. Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared against your defined rules.

Syntax

```
cckDioV tagName <model=m> <subckt=subckt_name>
   <inst=instName> <rmSub=subckt_name> <rmInst=instName>
   <skipSub=subckt_name> <skipInst=instName>
   <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1> <vpth=v2>
   <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6> <cond='expression'>
   <num=n> <separate_file=[0|1]> <rptTrace=[0|1]>
   <risePmosFallNmos=[0|1]> <subinfo=[0|1]>
   <filterAlert=[0|1]> <pwl_time=time>
   <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
```

cckResV

cckResV is used to perform analysis checks on resistors. Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared against your defined rules.

Syntax

```
cckResV tagName <model=m0del_name> <subckt=subckt_name>
   <inst=instName> <rmSub=subckt_name> <rmInst=instName>
   <skipSub=subckt_name> <skipInst=instName>
   <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1> <vpth=v2>
   <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6> <cond='expression'>
   <num=n> <separate_file=[0|1]> <rptTrace=[0|1]>
   <risePmosFallNmos=[0|1]> <subinfo=[0|1]>
   <filterAlert=[0|1]> <pwl_time=time>
   <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
```

Parameters

The following parameters are used for cckCapV, cckDioV, and cckResV.

tagName

tagName is a label in the report file to make searching easier.

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

limitMos=<number>

limitMos defines the expansion limit from the given voltage source that is not to exceed <number> transistors. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vlth=<vl>

The low voltage threshold. If vlth is specified, then CircuitCheck will trace from the voltage sources with values less than or equal to <vl>. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vhth=<vh>

The high voltage threshold. If vhth is specified, then CircuitCheck will trace only from sources with values equal to or greater than <vh>. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vnth=<v1>

vnth specifies the n-MOSFET threshold voltage. The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n-MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

vpth=<v2>

vpth specifies the p-MOSFET threshold voltage.The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvp=<v3>

lvp specifies the lower bound of the anode voltage. If a resistor's, capacitor's, or diode's anode voltage is less than <v3>, a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

uvp=<v4>

lvp specifies the upper bound of the anode voltage. If a resistor's, capacitor's, or diode's anode voltage is greater than <v3>, a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 349.

lvn=<v5>

lvn specifies the lower bound of the cathode voltage. If a resistor's, capacitor's, or diode's cathode voltage is less than <v5>, a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

uvn=<v6>

uvn specifies the upper bound of the cathode voltage. If a resistor's, capacitor's, or diode's cathode voltage is greater than <v5>, a warning is generated. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

cond=expression

A voltage constraint can be a Boolean expression such as: `cond='(uvp < 1 || lvn >= 0.5)` or mathematical operations (+,-,*, /, and abs()). Operators, +,-,*, / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, || ('or' operator). When this expression is true, a warning is generated.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

separate_file=[0|1]

separate_file indicates whether (1) or not (0) warnings are diverted to a different output file. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

- For cckResV, if set to 1, warnings are diverted from <hsim output prefix>.cckresv to <hsim output prefix>.cckresv_<command tag> output file(s).
- For cckCapV, if set to 1, warnings are diverted from <hsim output prefix>.cckcapv to <hsim output prefix>.cckcapv_<command tag> output file(s).
- For cckDioV, if set to 1, warnings are diverted from <hsim output prefix>.cckdiov to <hsim output prefix>.cckdiov_<command tag> output file(s).

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

risePmosFallNmos=[0|1]

risePmosFallNmos indicates wheter (1) or not (0) Device Voltage Analysis propagation applies restrictions with positive, zero, or negative voltage sources relative to element type. If risePmosFallNmos is set to 1, positive voltage sources can only be propagated through p-MOSFET devices, and negative or zero voltage sources can only be propagated through n-MOSFET devices. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

Note:

Circuit Check can pass other variables defined by .param into CCK commands. If users want to define a variable, this variable must be defined by .param and then be passed into CCK commands.

Examples

The following example passes user-defined parameters into a CCK command:

```
.param vol=3
cckCapV check_bias cond='( uvp*2 - lvn + 2*2 > vol*3 )'
```

The following example uses the cond= options:

```
(a) cond='(lvp - 0.5*uvn >= 1.5)'
(b) cond='(lvp >= 0.5*uvn + 1.5)'
(c) cond ='(uvp-lvp*0.5 > 1.5 *2)
(d) cond ='(uvp-abs(lvn-0.5) > 1.5 *2)'
```

The following are examples of static device voltage checking:

Add the following syntax to the CircuitCheck command file:

```
cckCapV cap_report model=CMOD inst=* uvp=2.0 uvn=2.0 vnth=1 vpth=1
rptTrace=1
cckDioV dio_report model=DMOD inst=* uvp=2.0 uvn=2.0 vnth=2 vpth=2
rptTrace=1
cckResV res_report model=RMOD inst=* uvp=2.0 uvn=2.0 vnth=1 vpth=1
rptTrace=1 separate_file=1
```

Report File Examples

The resulting report files are shown in the following three examples.

Sample .cckcapv Report:

```
Sample .cckcapv Report
tag=cap_report vnth=1.000 vpth=1.000 brief=0 rptTrace=1
     extTrace=0
limitCap=0 separate_file=0 risePmosFallNmos=0 model=cmod
inst=*
uvg=2.00 uvb=2.00

format: tag instName constraint-violated-node-name
     (node voltage)
cap_report xi11.x2.c1
     (P) net277 (3.00)
          thruI xi10.mxmp1_1_1
          frmNd vddh (constant node)
cap_report xi12.x3.c1
     (P) xi12.net3 (3.00)
          thruI xi12.mxmp1_1_1
          thruI xi12.x1.r1
          frmNd vddh (constant node)
cap_report xi17.x3.c1
     (P) xi17.net3 (3.00)
          thruI xi17.mxmp1_1_1
          thruI xi17.x1.r1
          frmNd vddh (constant node)
```

Sample .cckdiov Report:

```
Sample .cckdiov Report
*    tag=dio_report vnth=2.000 vpth=2.000 brief=0 rptTrace=1
     extTrace=0
*    limitDio=0 separate_file=0 risePmosFallNmos=0 model=dmod
*    inst=*
*    uvg=2.00 uvb=2.00
*
*    format: tag instName constraint-violated-node-name
     (node voltage)
dio_report xi11.x2.d1
     (P) net277 (3.00)
          thruI xi10.mxmp1_1_1
          frmNd vddh (constant node)
dio_report xi12.x1.d1
     (P) vddh (3.00)
     (N) xi12.net1 (3.00)
          thruI xi12.x1.d1
          frmNd vddh (constant node)
```

Sample .cckresv Report:

```
Sample .cckresv Report
*    tag=res_report vnth=1.000 vpth=1.000 brief=0 rptTrace=1
     extTrace=0
*    limitRes=0 separate_file=0 risePmosFallNmos=0 model=rmod
*    inst=*
*    uvg=2.00 uvb=2.00
*
*    format: tag instName constraint-violated-node-name
     (node voltage)
res_report xi11.x2.r1
     (P) net277 (3.00)
          thruI xi10.mxmp1_1_1
          frmNd vddh (constant node)

res_report xi12.x1.r1
     (P) vddh (3.00)
     (N) xi12.net1 (3.00)
          thruI xi12.x1.r1
          frmNd vddh (constant node)
res_report xi12.x3.r1
     (P) xi12.net3 (3.00)
          thruI xi12.mxmp1_1_1
          thruI xi12.x1.r1
          frmNd vddh (constant node)
```

Subcircuit-Based Voltage Analysis Using the Static Approach

You use the cckSubV command to perform a subcircuit-based voltage analysis using the static approach.

cckSubV

This command performs a static subcircuit port voltage check, which can be considered an extension of the existing static cckMosV command. However, unlike cckMosV, which performs a voltage check at the device (for example, MOSFET) level, cckSubV provides the ability to check the port/pin voltage in a subcircuit with a regular expression rather than a device terminal- specific pattern.

Syntax

```
cckSubV tagName <subckt=subckt_name> <inst=inst_name>
   <rmsub=subckt_name> <rmInst=inst_name>
   <skipInst=inst_name> <limitMos=num> <vhth=vh> <vlth=vl>
   <vnth=v2> <vpth=v3> <cond='expression'> <num=n>
   <separate_file=[0|1]> <rptTrace=[0|1]>
```

```
<risePmosFallNmos=[0|1]> <subinfo=[0|1]>
<filterAlert=[0|1]> <pwl_time=time>
<fvsrc='(e_name,vmin,vmax)'>
<fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
```

Parameters

The cckSubV command uses the following parameters.

tagName

tagName is a label in the report file to make searching easier.

subckt=<subckt_name>, inst=<inst_name>

The subckt parameter enforces that the node being analyzed is associated with the specified subckt <subckt_name>. Similarly, the inst parameter enforces that the node being analyzed is associated with the defined instance <inst_name>. If “subckt” is not specified, cckSubV looks for all subcircuits in the netlist. “subckt” does not support a “*” wild card, but “inst” does.

rmSub=<subckt_name>, rmInst=<inst_name>

The rmSub parameter prevents nodes that reside in the subcircuit <subckt_name> from being reported. The rmInst parameter prevents nodes that reside in the instance matching <inst_name> expression from being reported. “rmsub” does not support a “*” wild card, but “rmInst” does.

skipInst=<inst_name>

If this parameter is used, instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. This parameter allows the use of a “*” wildcard.

limitMos=<number>

limitMos defines the expansion limit from the given voltage source that is not to exceed <number> transistors. This parameter can be specified with global settings.

vhth=<vh>

The high voltage threshold. If vhth is specified, then Circuit Check traces only from sources with values equal to or greater than <vh>. This parameter can be specified with global settings.

vlth=<vl>

The low voltage threshold. If vlth is specified, then Circuit Check traces from the voltage sources with values less than or equal to <vl>. This parameter can be specified with global settings.

vnth=<v2>

vnth specifies the n-MOSFET threshold voltage. The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n-MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings.

vpth=<v3>

vpth specifies the p-MOSFET threshold voltage.The value <v3> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings.

cond='expression'[

You can use a Boolean expression to define a constraint as violation reporting criteria. If the Boolean value returned is logically true, it means the analysis satisfies the reporting criteria and the violation is reported. An expression can be formed by voltage of the node using lv(<node_name>) or uv(<node_name>), which means the lower range and upper range of node voltage respectively.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings.

separate_file=[0|1]

separate_file indicates whether (1) or not (0) warnings are diverted to a different output file. This parameter can be specified with global settings.

If set to separate_file=1, warnings are diverted from <hsim output prefix>.ccksubv to the <hsim output prefix.ccksubv_<command tag> output file(s).

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a conductive transistor path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings.

risePmosFallNmos=[0|1]

risePmosFallNmos indicates wheter (1) or not (0) Device Voltage Analysis propagation applies restrictions with positive, zero, or negative voltage sources relative to element type. If risePmosFallNmos is set to 1, positive voltage sources can only be propagated through p-MOSFET devices, and negative or zero voltage sources can only be propagated through n-MOSFET devices. The default value is 0. This parameter can be specified with global settings.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) Circuit Check produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check only propagates from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these sources only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings.

Note:

Circuit Check can pass other variables defined by .param into CCK commands. If you want to define a variable, this variable must be defined by .param and then be passed into CCK commands.

Examples

```
cckSubV sub1 subckt=tg cond='lv(in) <0 || uv(in) > 1' rptTrace=1
```

This command performs a subcircuit voltage check with the tag name sub1. According to the subckt=tg and the “cond=” expression, it looks for the voltage of node “in” defined within the tg subcircuit. If this node has its lower end voltage less than 0 or upper end voltage greater than 1V, a violation is reported.

```
cckSubV sub3 subckt=tg  cond='lv(in)<1 || uv(out)>3'  rptTrace=1
```

Violations are reported if the lower end of voltage of node “in” is less than 1V or the upper end of voltage of node “out” is greater than 3V.

The following example shows a cckSubV report:

```
* Static Device-V Analysis
* command= cckSubV
* tag=tag2
* cond='lv(out)<0 || uv(out) > 2'
* vpth=-0.500 vnth=0.500
* rptTrace=1  limitMos=0  separate_file=0
* risePmosFallNmos=0  subinfo=0 FilterAlert=0
*
* subckt= tg
*
* format: tag instName constraint-violated-node-name (minv
- maxv)
tag2 x27
 (2) n73 (0.00 - 3.00)
   thruI x27.mn1 (V=3)
   thruI x25.mp1 (V=3)
   frmNd vdd (constant node
```

Diode Forward Bias Analysis

cckDiode is used to perform forward bias checks on diodes.

cckDiode

Syntax

```
cckDiode <mode=0|1|2> <num=n> <separate_file=0|1>
   <subinfo=[0|1]> <vt=vt0> <start=t1> <stop=t2>
   <model=model_name> <tag=t>
```

Parameters

mode=<0|1|2>

mode determines the mode of analysis. The possible values are:

- mode=0 performs a static check.
- mode=1 performs a dynamic check.
- mode=2 performs both a static and dynamic check.

This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

num=n

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default is 300. If num is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

separate_file=[0|1]

separate_file=0 is the default which causes all warning messages to be reported in the <hsim_output_prefix>.cck file. If separate_file=1, warning messages are reported in a separate <hsim_output_prefix>.cckDiode file. This parameter is valid only in static mode (mode=0). This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vt=<vt0>

This control parameter is used in both static and dynamic modes. After voltage propagation, <vt0> is used to determine if a violation is warranted based on the following equation:

```
V(anode) >= V(cathode) +<vt0>
```

The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

start=<t1>

Specifies the start time for vt checks.

stop=<t2>

Specifies the stop times for vt checks.

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices. The diode types to be checked using the model parameter are user-selected.

tag=<t>

t is printed in the report file at the beginning of the readability line for a forward-based diode.

Description

In transient simulation, it is necessary to determine if any diode becomes forward-biased. If an anode's voltage is greater than a cathode's voltage by a threshold value, a Warning message will be printed. The default threshold is overwritten using vt=parameter.

The Warning message is written into one of the following files:

- hsim.cck
- out_file.cck

Examples

```
cckDiode mode=2 num=100 start=100n stop=500n model=pdio tag=t1\
separate_file=1
```

Anode and cathode voltage are checked in static mode and dynamic mode because mode=2. While in dynamic mode, the checks occur between 100 ns and 500 ns for diode model pdio. If the anode voltage is larger than the cathode voltage by 0.5V, a Warning message is reported in the hsim.cck or out_file.cck file. When the number of warnings exceeds 100, the checking will be stopped. The following is the MOSFET Bulk and Diode Forward-Biased in Simulation (hsim.cck or out_file.cck) output sample resulting from the command shown in the example above.

Report from static mode:

```
****************************************************
* Diode forward-bias checking before DC
*
*   tag=t1
*   model=pdio
*   vt=0.5
*   ith=5e-05
*   seperate_file = 1
*   num = 100
*   start=0 stop=0
*
****************************************************
Diode (d1) is forward-biased (anode (dvdd_o, 3.3 volt) cathode
(net1, 0 volt)), via serial res/ind.
```

Report from dynamic mode:

```
****************************************************
* MOS-Bulk/Diode Forward Bias and Node Voltage Check
* diode forward bias: tag=t1 model=pdio vt=0.5
* start=1e-07 stop=5e-07
****************************************************
(t1) @100.00n, diode (d1) forward biased
        v(anode)=3.3  v(cathode)=1.67559
```

In this output sample derived from the previous example, it reads as follows: at time 100 ns, a diode (d1) becomes forward-biased with the following parameters.

- Anode voltage: 3.3V
- Cathode voltage: 1.67559V

Element Current Analysis

cckElemI

In transient simulation, CircuitCheck monitors the current through each element. If the absolute value of the current exceeds the threshold ith, CircuitCheck reports the element name, current, and time. If tag is specified, the Warning will be prefixed with the tag.

Syntax

```
cckElemI <start=t1> <stop=t2> <tag=t> <subckt=s> <inst=name>
   <ith=v> <model=m> <rms=[0|1]> <avg=[0|1]> <num=n>
   <ithAbs=[0|1]>
```

Parameters

subckt

Instance current to be checked by using subckt=s and inst=name is user-selectable.

inst

Instance current to be checked by using subckt=s and inst=name is user-selectable.

rms

The option `rms=1` is to permits checking the root-mean-square current of the specified element during a given time window [t1, t2]. If this rms current exceeds the threshold ith=v, then this element will be reported.

avg=1

avg=1 checks the average current in an element during a given time window. If the average current is larger than the ith=v threshold, the element is printed.

ithAbs

Setting ithAbs=1 report the current with sign value instead of reporting the absolute value. CircuitCheck reports absolute current values by default.

Example

```
cckElemI start=10n stop=76n tag=t1 subckt=aa inst=* ith=1.e-6
model=pch rms=1
```

Within a specified time window from 10n to 76n, all instances in all instantiations of subckt=aa are examined. The root-mean-square current of each pch transistor is then computed. If the current is greater than 1u Amp, the element is reported using a t1 tag.

Instance and Subcircuit Reference Check

cckMatchSub

Syntax

```
cckMatchSub <subckt=subckt_name> <ReptHierNode=1|0>
```

Parameters

subckt

Specifies the subckt name as the matching target.

ReptHierNode

When ReptHierNode=1, both the matched nodes and the associated upstream hierarchical nodes are reported.

Description

cckMatchSub conducts a reference check and lists all instances associated with a specified subckt, with the port mapping information. While cckPatternMatch on page 316 performs detailed pattern matching by traversing design hierarchy and/or flattening certain levels during the operation, its intensive approach may require a certain amount of computing resource. cckMatchSub is used as a supplement to cckPatternMatch to gain early and final design-phase matching results.

Excessive Current Path Detection

cckExiPath

cckExiPath detects excessive current paths from a power supply to the ground. Both current thresholds and time duration thresholds are specified by you.

Syntax Definitions

```
cckExiPath <ith=cur> <tth=time> <from=vsrc1 <from=vsrc2
   ...>> <to=gnd <to=vss ...>> <start=t1> <stop=t2>
```

Parameters

ith

ith is current threshold. All the elements in a path must have current larger than the ith value.

tth

tth is the time threshold. The excessive current path must exist for more than this time interval.

from

from defines the power supply. Multiple from nodes are allowed in a command.

to

to defines the ending nodes. Multiple to nodes are allowed in a command.

Note:

If "from" and "to" options are not specified, cckExiPath searches the dc paths with all combinations of all voltage sources (VSRCs).

Examples

Given a netlist with VSRC nodes, VDD, VCC and VSS, if "from" and "to" are not specified, cckExiPath searches the paths "from VDD to VSS" and "from VCC to VSS" and from "VDD to VCC".

The following is an example of excessive current path detection syntax:

```
cckExiPath ith=10u tth=0.6n from=vdd from=xam.d1 to=gnd start=10n
stop=100n
```

cckExiPath starts monitoring from 10 ns to 100 ns. If a path exists from VDD or node xam.d1 to GND and all the elements in this path have current greater than 10uAmp and last for more than 0.6 ns, then this path is reported. The output of the Warning goes to file hsim.cckexipath.

The following is a sample cckExiPath output produced using the syntax shown above.

```
****************************************************
* cckExiPath ith=1e-005 tth=6e-010 stop=1e-007
* from=xam.d1
* from=vdd
* to=gnd
****************************************************
path 0: from xam.x8.n to gnd, time 5e-009 - 6e-008 (duration 5.5e-
008)
     xam.xcr8.mn (eid=25508) I=0.000197307
          node gnd
path 1: from vdd to gnd, time 3.2384e-008 - 4.0598e-008 (duration
8.214e-009)
     xam.xrd.mu1 (eid=50071) I=0.000146329
          node xam.xu8.p1
     xam.xr.mu2 (eid=50072) I=0.000141375
          node xam.npf<1>
     xam.xrd1.mn (eid=49985) I=4.14077e-005
          node gnd
```

Floating Gates and Current Sources Analysis

cckFloatGateIsrc

Description

cckFloatGateIsrc is used to perform both floating gate checks on nmos/pmos transistors and checks to see if a current source is only connected to nmos/pmos transistor gates.

Syntax

```
cckFloatGateIsrc [1|0] <listall=0|1> <num=n> <subinfo=0|1>
   <skipport=0|1> <extTrace=0|1>
```

Parameters

0|1

Enter 0 to turn checking OFF

num=n

This parameter limits the total number of Warnings reported. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

listall= [0|1]

listall, when set to 0, reports only one representative floating gate per gate node, and whether it is a PMOS or NMOS. When set to 1, both NMOS and PMOS devices are reported when their gates are connected together and are floating. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

skipport=[0|1]

skipport indicates whether (1) or not (0) all top level ports and nodes connected to top level ports through resistors and/or inductors are excluded from the floating node report.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

Examples

```
cckFloatGateIsrc 1
```

The following is the Floating gate nodes and Floating isrc nodes (hsim.cck or out_file.cck) output sample resulting from the command example above.

Note:

Only representative devices are reported, since listall parameter is not used.

```
****************************
* Floating gate nodes
****************************
mos (x1.mn) gate node (x1.p) floats
****************************
* Floating isrc nodes
****************************
current src node (x2.i) floats
```

An intuitive example for `listall` is that three inverters, inv1, inv2, and inv3, have their gates connected together to an input node, but nothing drives the input.

- listall=0 (default): cckFloatGateIsrc reports only one representative MOS device from the inv1, inv2, and inv3 as a floating gate.
- listall=1: cckFloatGateIsrc reports all the MOS devices in inv1, inv2, and inv3 as floating gates.

Check NMOS Bulk Connections

cckNmosB_gt_DS

Note:

> See cckPmosG_lt_DS on page 217 for detailed explanation of the parameters and examples for using this command.

Syntax

```
cckNmosB_gt_DS <model=model_name> <subckt=subckt_name>
   <inst=inst_name> <rmSub=subckt_name> <rmInst=inst_name>
   <skipSub=suckt_name> <skipInst=inst_name> <vlth=vl>
   <vt=vt0> <vnth=v1> <vpth=v2> <num=n> <rptv=[0|1]>
   <rptTrace=[0|1]> <subinfo=[0|1]> <filterAlert=[0|1]>
   <pwl_time=time> <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
   <-connSub=subckt_name> <-connInst=inst_name> <-
   connNode=node_name> <autoFvsrcnd=[0|1]>
```

Default Values

vlth=0.4

vt=0.3

vnth=0.5

vpth=-0.5

Parameters

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

vlth=<vl>

The low voltage threshold. If vlth is specified, then CircuitCheck will trace from the voltage sources with values less than or equal to <vl>. The default value is 0.4. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vt=<vt0>

After voltage propagation <vt0> is used to determine if a violation is warranted based on the equation:

```
max(Vb) >= min(Vd/s)+<vt0>
```

The default value for this parameter is 0.3. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vnth=<v1>

The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n_MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vpth=<v2>

The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

rptv=[0|1]

rptv idicates whether (1) or not (0) voltage ranges are reported. The default value is 1.

Note:

This parameter is now obsolete, as this is now the default behavior. It will no longer be supported in future releases.

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

–connSub=subckt_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified subcircuit name.

–connInst=inst_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

–connNode=node_name

Limits elements to be reported to those that have a direct connection to the specified node name.

autoFvsrcnd=[0|1]

If set to 1 enables the automatic pwl voltage range feature. This feature conducts the following additional steps before the static check:

1. Search for any pwl or pulse Vsrc definition in the netlist
2. For any found pwl or pulse Vsrc, obtain its voltage range info (Vmin, Vmax)
3. Based on the obtained voltage range information, perform an operation equivalent to "fvsrcnd(<VsrcNodeName>, Vmin, Vmax)"

The default is 0.

Note:

Among the -connSub, -connInst, and -connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Description

cckNmosB_gt_DS is used to perform analysis checks on n-MOSFET transistors specifically to see if the maximum bulk voltage (maxVb) is greater than the minimum drain/source voltages (minVd/s). Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If `maxVb >= minVd/s + vt` warnings are generated. The num parameter is used to limit the number of warnings that are reported.

Example

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx       0 pvdd
vvde vvdex       0 pwl 0ns 0 0.1ns pvdd
vshift shift       0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckNmosB_gt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrcnd=1
```

CCK operates equivalently to following syntax:

```
cckNmosB_gt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrcnd=(vvdex, 0, 1.11)
fvsrcnd=(shift, 0, 0.8)
```

The report header is updated to include the autoFvsrcnd setting:

```
****************************************************
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*  vnth=0.300 vpth=-0.500
*  vhth=0.500
*
*  vt=0.10 num=300
*
*  autoFvsrcnd=1
*  format: instName gate_volt d/s_volt
****************************************************
```

Find Potentially Conducting NMOS Devices

cckNmosG_gt_DS

Syntax

```
cckNmosG_gt_DS> <model=model_name> <subckt=subckt_name>
   <inst=inst_name> <rmSub=subckt_name> <rmInst=inst_name>
   <skipSub=suckt_name> <skipInst=inst_name> <vlth=vl>
   <vt=vt0> <vnth=v1> <vpth=v2> <num=n> <rptv=[0|1]>
   <rptTrace=[0|1]> <subinfo=[0|1]> <filterAlert=[0|1]>
   <pwl_time=time> <fvsrc='(e_name,vmin,vmax)'>
```

```
<fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
<-connSub=subckt_name> <-connInst=inst_name> <-
connNode=node_name> <autoFvsrcnd=[0|1]>
```

Default Values

- vlth=0.4
- vt=0.3
- vnth=0.5
- vpth=-0.5

Parameters

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

vlth=<vl>

The low voltage threshold. If vlth is specified, then CircuitCheck will trace from the voltage sources with values less than or equal to <vl>. The default value is 0.4. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vt=<vt0>

After voltage propagation <vt0> is used to determine if a violation is warranted based on the equation:

```
max(Vg) >= min(Vd/s)+<vt0>
```

The default value for this parameter is 0.3. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vnth=<v1>

The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n_MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vpth=<v2>

The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

rptv=[0|1]

rptv idicates whether (1) or not (0) voltage ranges are reported. The default value is 1.

Note:

This parameter is now obsolete, as this is now the default behavior. It will no longer be supported in future releases.

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

–connSub=subckt_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified subcircuit name.

–connInst=inst_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

–connNode=node_name

Limits elements to be reported to those that have a direct connection to the specified node name.

autoFvsrcnd=[0|1]

If set to 1 enables the automatic pwl voltage range feature. This feature conducts the following additional steps before the static check:

1. Search for any pwl or pulse Vsrc definition in the netlist
2. For any found pwl or pulse Vsrc, obtain its voltage range info (Vmin, Vmax)
3. Based on the obtained voltage range information, perform an operation equivalent to "fvsrcnd(<VsrcNodeName>, Vmin, Vmax)"

The default is 0.

Note:

> Among the -connSub, -connInst, and -connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Description

cckNmosG_gt_DS is used to perform analysis checks on n-MOSFET transistors specifically to see if the maximum gate voltage (maxVg) is greater than the minimum drain/source voltages (minVd/s). Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If `maxVg >= minVd/s + vt` warnings are generated. Refer to Figure 16.

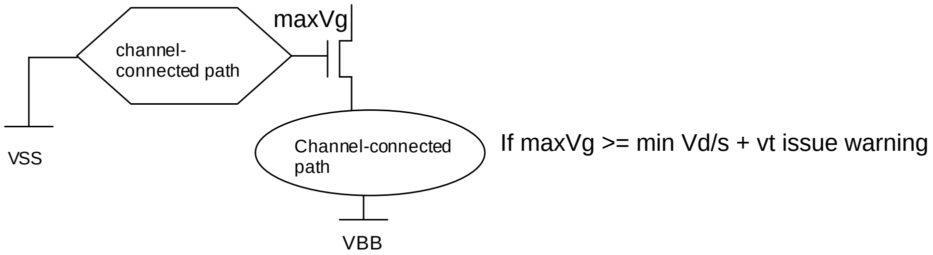


Figure 18 Report Warning Circuit

Examples

```
cckNmosG_gt_DS vlth=0.4 vt=0.3 inst=xam*
```

The following is the .cck file output sample resulting from the command example above.

```
****************************************************
* NMOS gate vsrc value greater than D/S vsrc: inst=xam*
* vlth=0.40   vt=0.30
* format: instName gate_volt d/s_volt
****************************************************
 xam.xctl.mu30      Vg (0.6) Vs (0)
 xam.xu22.mu2       Vg (0.6) Vs (0)
 xam.xi14.mu30      Vg (0.6) Vs (0)
```

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx       0 pvdd
vvde vvdex       0 pwl 0ns 0 0.1ns pvdd
vshift shift       0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
ckNmosG_gt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrcnd=1
```

CCK operates equivalently to following syntax:

```
ckNmosG_gt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrcnd=(vvdex, 0, 1.11)
fvsrcnd=(shift, 0, 0.8)
```

The report header is updated to include the autoFvsrcnd setting:

```
****************************************************
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*  vnth=0.300 vpth=-0.500
*  vhth=0.500
*
*  vt=0.10 num=300
*
*  autoFvsrcnd=1
*  format: instName gate_volt d/s_volt
****************************************************
```

Check NMOS Node to VDD Connection

cckNmosNodeToVdd

Description

Checks NMOS terminals (drain/source/gate/bulk). If any one is connected to logic-high voltage source (whose value is larger than vhth), report it.

Usage Syntax 1

```
cckNmosNodeToVdd  <tag=tagName> <num=n> <vhth=v>
   <inst=inst_name> <model=model_name> <node=drain>
   <node=source> <node=gate> <node=bulk>
```

Usage Syntax 2

```
cckNmosNodeToVdd <tag=tagName> <num=n> <vhth=v>
   <inst=inst_name> <model=model_name>
   <node='drain|source|gate|bulk'>
```

Note:

Usage Syntax 2 is a version of Usage Syntax 1 to simplify the "node" specification. Node='drain | source' is the same as node=drain node=source.

Parameters

tag

Specifies the label in the report.

num

Specifies the total number of violations to be reported

vhth

Specifies the threshold voltage to be checked. The default is 0.7V.

inst

Specifies the instance name in the design to be checked.

model

Specifies the model to be checked.

node

Specifies the nodes to be checked. The keyword, "*" and "all", are supported for checking on four terminals of a MOS.

Note:

Multiple cckNmosNodeToVdd commands can be specified in the CCK command file.

Examples

The following examples are equivalent. They check the node voltage inside instance, x03 with model=nmos. If the voltages of gate node and drain node are higher than 0.7V, they will be reported.

```
cckNmosNodeToVdd tag=tag2 vhth=0.7 node=gate node=drain
inst=x03.* model=nmos
cckNmosNodeToVdd tag=tag2 vhth=0.7 node='gate| drain' inst=x03.*
model=nmos
```

The following example checks the four terminals of the NMOS device in a design since "node=*" is specified.

```
cckNmosNodeToVdd tag=tag4 vhth=0.7 node=*
```

The following example checks the four terminals of the NMOS device inside instance x03 with model name, nmos, because the keyword, "all", is used in the "node=" control parameter.

```
cckNmosNodeToVdd tag=tag5 vhth=0.7 node=all inst=x03.* model=nmos
```

Check Node Voltage

cckNodeVoltage

Checks if the voltage of a node exceeds specified limits.

Syntax

```
cckNodeVoltage <num=n> <vmax=v1> <vmin=v2> <start=t1>
   <stop=t2> <model=m> <tag=t> <lvdb=v> <uvdb=v> <lvbd=v>
   <uvbd=v> <lvds=v> <uvds=v> <lvsd=v> <uvsd=v> <lvdg=v>
   <uvdg=v> <lvgd=v> <uvgd=v> <lvgs=v> <uvgs=v> <lvsg=v>
   <uvsg=v> <lvbs=v> <uvbs=v> <lvcs=v> <uvcs=v> <lvsc=v>
   <uvsc=v> <lves=v> <uves=v> <lvse=v> <uvse=v> <lvbc=v>
   <uvbc=v> <lvcb=v> <uvcb=v> <lvbe=v> <uvbe=v> <lveb=v>
   <uveb=v> <lvce=v> <uvce=v> <lvec=v> <uvec=v> <lvsb=v>
   <uvsb=v> <lvgb=v> <uvgb=v> <lvbg=v> <uvbg=v> <lvac=v>
   <uvac=v> <node=name>
```

Description

In transient simulation, CircuitCheck monitors node voltage. If any node voltage is greater than the maximum value (v1), or smaller than the minimum value (v2), then CircuitCheck reports the following:

- Node name
- Voltage
- Time

`num=m`: If the number of warnings exceeds a predetermined threshold (`num=n`), CircuitCheck stops printing. The time window to monitor the node voltage can also be specified.

`model=m`: The type of devices used to check node voltage can be selected. The following three model types are supported:

- MOSFET
- BJT
- Diode

cckNodeVoltage compares the voltage of a pair of terminals to user-defined thresholds. For example, MOSFET elements have four terminals:

- Drain
- Source
- Gate
- Bulk

For each of these four MOSFET nodes, the following bounds can be applied:

- lvdb: Lower bound of drain-to-bulk voltage difference.
- uvdb: Upper bound of drain-to-bulk voltage difference.
- lvgs: Lower bound of gate-to-source voltage difference.
- uvgs: Upper bound of gate-to-source voltage difference.

Example

The following example represents a sample device used to check node voltage:

```
cckNodeVoltage vmax=10 vmin=-3 model=nch
```

In the following example, starting from 500ns the gate-to-source voltage difference of ph1a MOSFET is checked. If it is smaller than 0.5V, it is reported and output tag ph1a is placed at the beginning of the line. The drain-to-bulk voltage difference is also checked to see if it is greater than 1.3V.

If the number of Warnings exceeds a predetermined threshold (num=n), CircuitCheck will stop printing. The time window to monitor the node voltage can also be specified.

```
cckNodeVoltage model=ph1a tag=ph1a lvgs=0.5 uvdb=1.3 start=500n
```

In the following example, starting from 100ns until end of simulation, CircuitCheck monitors node voltage. If the voltage is > 5 volt, or < –2.3 volt, a Warning is issued.

```
cckNodeVoltage    num=100 vmax=5 vmin=-2.3 start=100n
```

The following is the MOSFET Bulk/Diode Forward Bias and Node Voltage Check (from hsim.cck or out_file.cck) output sample resulting from the command example above.

```
**** ********************************************
* MOS-Bulk/Diode Forward Bias and Node Voltage Check
* node voltage: vmax=5, vmin=-2.3
**** ********************************************
@20.4n, node (xx1.pg.a1) voltage (6.71) exceeds vmax
@100.7n, node (x1.asrc.p1) voltage (-3.4) is below vmin
```

In the following example, nodes in different areas have different voltage ranges. Node names are allowed in the command. CircuitCheck will examine those specific nodes to see if their voltages meet the constraint. Otherwise, a Warning is created. For example, nodes in x1 and x2 instances are limited to be within -1V and 3V. The nodes in x4 instance need to be within -2V to 4V.

```
cckNodeVoltage   vmax=3   vmin=-1 node=x1* node=x2*
cckNodeVoltage   vmax=4   vmin=-2 node=x4*
```

Check Paths to Voltage Sources

cckPathToVsrc

Checks whether a node has a path to reach VDD, GND, both, or specified nodes, and reports if no such path is found.

Syntax Usage 1

```
cckPathToVsrc <fanout=0|1]> <num=N> <node=node_name(s)>
   <bjton=0|1> <bjtonc2e=0|1> <bjtonb2e=0|1> <bjtonb2c=0|1>
```

Syntax Usage 2

```
cckPathToVsrc <fanout=0|1]> <num=N> <node=node_name(s)>
   <ckt_node=internal_node_name(s)>
   <vsrc_node=vsrc_node_name(s)> <bjton=0|1>
   <bjtonc2e=0|1> <bjtonb2e=0|1> <bjtonb2c=0|1>
```

Description

cckPathToVsrc checks whether a node has a path to reach VDD, GND, both, or specified nodes, and reports if no such path is found.

There are two uses for cckPathToVsrc, and the reporting styles are slightly different in both. When syntax usage 1 is issued, nodes are reported if they can not reach a constant voltage source, ground, or both. The report has three categories:

- No path to any generic voltage sources or ground node(s)
- No path to any generic voltage sources
- No path to any ground nodes

The generic voltage source refers to all of the possible voltage sources in the design or netlist, regardless of the names and the voltage level of the voltage sources.

Syntax usage 2 is more enhanced. When this version is issued, it checks if a node has a path to voltage source, ground source, or specific nodes. The report contains two categories:

- No path to the vsrc_node `specified_vsrc_node_name`
- No path to the circuit node `specified_ckt_node_name`

The `specified_vsrc_node_name` and `specified_ckt_node_name` parameters are the node names specified in the control parameters, `vsrc_node=` and `ckt_node=` respectively. Version (2) is more beneficial in a design with multiple voltage sources. For example, when a design has multiple voltage sources, (VDD1, VDD2, etc.) the particular VSRC nodes can be reported in the banner if there are violations.

Parameters

fanout=0|1

To only check nodes that have a direct connection to the transistor gate, set fanout=1. To check all nodes, set fanout=0. The default is 0.

num=N

num limits the number of Warnings in each category to N

node=node_name(s)

cckPathToVsrc only reports nodes specified in the node parameter.

vsrc_node=vsrc_node_name(s)

This parameter specifies the names of the voltage source node(s). cckPathToVsrc traces if the specified nodes in node= could reach the vsrc_node. The vsrc_node must be a voltage source. If non-vsrc-nodes are chosen, Warnings are issued in the log file.

ckt_node=cke_node_name(s)

This parameter is used to check if there are paths from the nodes to the internal nodes specified by ckt_node. ckt_node must be internal node(s) rather than voltage sources.

bjton=0|1

This parameter is used to include all BJT devices in the checking. The default is 0 which excludes all BJT devices.

bjtonc2e=0|1

This parameter is used to include collector to emitter paths with bjtonc2e=1 for all BJT devices in the checking. The default is 0.

bjtonb2e=0|1

This parameter is used to include base to emitter paths with bjtonb2e=1 for all BJT devices in the checking. The default is 0.

bjtonb2c=0|1

This parameter is used to include base to collector paths with bjtonb2c=1 for all BJT devices in the checking. The default is 0.

Note:

If a node has a resistive path to the current source, it will not be reported.

Note:

There is a limitation that the total number of ckt_node and vsrc_node can not exceed 30 nodes. If the number of given nodes is exceeded, Warning will be issued in the log file.

Note:

A header is generated in the report file with the information specified in this CircuitCheck command.

Examples

In the following example, all nodes with fanout are checked to see if constant voltage sources can be reached. The number of Warnings for each category will not exceed 300.

```
cckPathToVsrc num=300 node=*
```

The following is the Check Path to Voltage Sources (from hsim.cck or out_file.cck) output sample resulting from the command example above.

```
****************************************************
* Check Path to Voltage Sources
*    fanout = 0
*    num    = 300
*    node   = *
****************************************************
No path to any generic voltage sources or ground node(s):
 node (to_inv3) has no path to any voltage sources or ground node(s)

No path to any generic voltage sources:
  node (low) has no path to any generic voltage sources

No path to ground node(s):
  node (high) has no path to ground node(s)
~



~
```

Note:

> The report style of version (1) and version (2) are slightly different. See the variations in the following report examples.

When the command in the example above is changed to the following:

```
cckPathToVsrc node=* vsrc_node=vdd vsrc_node=vss ckt_node=to_inv3
```

The report looks like this:

```
No path to circuit node 'to_inv3':
  node (high) has no path to to_inv3
  node (low) has no path to to_inv3
  node (out) has no path to to_inv3
  node (to_inv3) has no path to to_inv3

No path to specified vsrc_node 'vss':
  node (high) has no path to vss
  node (to_inv3) has no path to vss

No path to specified vsrc_node 'vdd':
  node (low) has no path to vdd
  node (to_inv3) has no path to vdd
```

In the third example, all transistor nodes including BJTs with fanout=1 are checked to determine if constant voltage sources can be reached. The number of Warnings for each category will not exceed 10000.

```
cckPathToVsrc fanout=1 node=* num=10000 BJTON=1
```

The following is the Check Path to Voltage Sources (from hsim.cck or out_file.cck) output sample resulting from the command example above.

```
****************************************************
* Check Path to Voltage Sources
*    fanout    = 1
*    num       = 10000
*    bjton     = 1
*    bjtonb2c  = 1
*    bjtonb2e  = 1
*    bjtonc2e  = 1
*    node   = *
****************************************************
No path to any generic voltage sources or ground node(s):
node(xids3234a1.xoscillator.xosc_biassource.xosc_biasres.net26)
has no path to any voltage sources or ground node(s)
node(xids3234a1.xoscillator.xosc_biassource.xosc_biasres.net28)
has no path to any voltage sources or ground node(s)
node (xids3234a1.xpads_east.dinr) has no path to any voltage
sources or ground node(s)
~
~
```

Note:

> The report includes BJTs on in path checking. You can apply any of the combinations of bjtonc2e/bjtonb2e/bjtonb2c=0 in this example. However, HSIM issues a warning when the combination of bjtonc2e/bjtonb2e/bjtonb2c=1 is used with bjton=0

Check PMOS Bulk Connections

cckPmosB_lt_DS

Syntax

```
cckPmosB_lt_DS <model=model_name> <subckt=subckt_name>
   <inst=inst_name> <rmSub=subckt_name> <rmInst=inst_name>
   <skipSub=suckt_name> <skipInst=inst_name> <vhth=vh>
   <vt=vt0> <vnth=v1> <vpth=v2> <num=n> <rptv=[0|1]>
   <rptTrace=[0|1]> <subinfo=[0|1]> <filterAlert=[0|1]>
   <pwl_time=time> <fvsrc='(e_name,vmin,vmax)'>
```

```
<fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
<-connSub=subckt_name> <-connInst=inst_name> <-
connNode=node_name> <autoFvsrcnd=[0|1]>
```

Default Values

vhth=0.7

vt=0.3

vnth=0.5

vpth=-0.5

Parameters

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

vhth=<vh>

The high voltage threshold. If vhth is specified, then CircuitCheck traces only from sources with values equal to or greater than <vh>. The default value for this parameter is 0.7. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vt=<vt0>

After voltage propagation <vt0> is used to determine if a violation is warranted based on the equation:

```
min(Vb) <= max(Vd/s)-<vt0>
```

The default value for this parameter is 0.3. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vnth=<v1>

The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n_MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vpth=<v2>

The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

rptv=[0|1]

rptv idicates whether (1) or not (0) voltage ranges are reported. The default value is 1.

Note:

> This parameter is now obsolete, as this is now the default behavior. It will no longer be supported in future releases.

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

–connSub=subckt_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified subcircuit name.

–connInst=inst_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

–connNode=node_name

Limits elements to be reported to those that have a direct connection to the specified node name.

autoFvsrcnd=[0|1]

If set to 1 enables the automatic pwl voltage range feature. This feature conducts the following additional steps before the static check:

1. Search for any pwl or pulse Vsrc definition in the netlist
2. For any found pwl or pulse Vsrc, obtain its voltage range info (Vmin, Vmax)
3. Based on the obtained voltage range information, perform an operation equivalent to "fvsrcnd(<VsrcNodeName>, Vmin, Vmax)"

The default is 0.

Note:

> Among the -connSub, -connInst, and -connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Description

cckPmosB_lt_DS is used to perform analysis checks on p-MOSFET transistors specifically to see if the minimum bulk voltage (minVb) is less than the maximum drain/source voltages (maxVd/s). Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If `minVb <= maxVd/s - vt` warnings are generated. Setting num=n limits the number of warnings.

Example

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx       0 pvdd
vvde vvdex       0 pwl 0ns 0 0.1ns pvdd
vshift shift       0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckPmosB_lt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrcnd=1
```

CCK operates equivalently to following syntax:

```
cckPmosB_lt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrcnd=(vvdex, 0, 1.11)
fvsrcnd=(shift, 0, 0.8)
```

The report header is updated to include the autoFvsrcnd setting:

```
****************************************************
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*  vnth=0.300 vpth=-0.500
*  vhth=0.500
*
*  vt=0.10 num=300
*
*  autoFvsrcnd=1
*  format: instName gate_volt d/s_volt
****************************************************
```

Find Potentially Conducting PMOS Devices

cckPmosG_lt_DS

Syntax

```
cckPmosG_lt_DS <model=model_name> <subckt=subckt_name>
   <inst=inst_name> <rmSub=subckt_name> <rmInst=inst_name>
   <skipSub=suckt_name> <skipInst=inst_name> <vhth=vh>
   <vt=vt0> <vnth=v1> <vpth=v2> <num=n> <rptv=[0|1]>
   <rptTrace=[0|1]> <subinfo=[0|1]> <filterAlert=[0|1]>
   <pwl_time=time> <fvsrc='(e_name,vmin,vmax)'>
   <fvsrcnd='(vsrc_node_name,vmin,vmax)'> <extTrace=[0|1]>
   <-connSub=subckt_name> <-connInst=inst_name>
   <-connNode=node_name> <autoFvsrcnd=[0|1]>
```

Default Values

vhth=0.7

vt=0.3

vnth=0.5

vpth=-0.5

Parameters

model=<model_name>

model causes CircuitCheck to go through all instances of <model_name> during path tracing. CircuitCheck will not consider other model devices.

subckt=<subckt_name>, inst=<inst_name>

subckt=subckt_name and inst=inst_name examine all instances of <inst_name> in all of the instances of sub-circuit <subckt_name>. Instance names may contain wild cards; the sub-circuit name can not.

rmSub=<subckt_name>, rmInst=<inst_name>

If one or both of these parameters are used, elements that satisfy the defined conditions are not checked. The parameter rmSub prevents devices which reside in the subcircuit <subckt_name> to be reported. The parameter rmInst prevents devices which match the instance <inst_name> to be reported. Both parameters allow the use of wildcards.

skipSub=<subckt_name>, skipInst=<inst_name>

If one or both parameters are used, subcircuits matching <subckt_name> and instances matching <inst_name> prevent voltage propagation from occurring through elements within their coverage. Both parameters allow the use of wildcards.

vhth=<vh>

The high voltage threshold. If vhth is specified, then CircuitCheck traces only from sources with values equal to or greater than <vh>. The default value for this parameter is 0.7. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vt=<vt0>

After voltage propagation <vt0> is used to determine if a violation is warranted based on the equation:

```
min(Vg) <= max(Vd/s)-<vt0>
```

The default value for this parameter is 0.3

vnth=<v1>

The value <v1> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a n_MOSFET transistor channel. The default value for this parameter is 0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

vpth=<v2>

The value <v2> is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value for this parameter is -0.5. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

num=<n>

num limits the number of warnings generated by a particular analysis command defining it. Only <n> warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <n> is set to less than or equal to 0, the warnings are unlimited. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

rptv=[0|1]

rptv idicates whether (1) or not (0) voltage ranges are reported. The default value is 1.

Note:

This parameter is now obsolete, as this is now the default behavior. It will no longer be supported in future releases.

rptTrace=[0|1]

rptTrace indicates whether (1) or not (0) Circuit Check reports a transistor's conductive path that leads from its node(s) to a voltage source. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

subinfo=[0|1]

subinfo indicated whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

filterAlert=[0|1]

filterAlert indicates whether (1) or not (0) CircuitCheck produces an error message and the entire process is terminated when the scope of a given command, defined by the 'model', 'subckt' and/or 'inst' parameters, is found to be empty. The default is 0. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

pwl_time=<time>

pwl_time, when specified, is used to propagate time specific voltage value(s) from a piece-wise-linear (pwl) voltage source(s). The voltage value (v(time)) used for propagation is defined by the pwl voltage source at time=<time>. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source of value=v(time). By default Circuit Check only propagates from constant voltage sources. All <time> units are in seconds (s). For example: `pwl_time=10n`.

fvsrc='(<e_name>,<vmin>,<vmax>)'

fvsrc is used to propagate from voltage source element <e_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

fvsrcnd='(<vsrc_node_name>,<vmin>,<vmax>)'

fvsrcnd is used propagate from voltage source node <vsrc_node_name> with values <vmin> and <vmax>. If vhth is set, constant voltage sources greater than or equal to the value set for vhth is used as a starting point for propagation. If vlth is set, constant voltage sources less than or equal to the value set for vlth is used as a starting point for propagation. By default Circuit Check will only propagate from constant voltage sources.

extTrace=[0|1]

extTrace indicates whether (1) or not (0) Circuit Check selects two constant voltage sources and starts propagation from these only. If a violation is found a warning is generated. The default 0, meaning propagation starts from all constant voltage sources, satisfying the rules set by vhth and/or vlth. This parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

–connSub=subckt_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified subcircuit name.

–connInst=inst_name

Limits elements to be reported to those that have a direct connection to the ports (except Power/Ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

–connNode=node_name

Limits elements to be reported to those that have a direct connection to the specified node name.

autoFvsrcnd=[0|1]

If set to 1 enables the automatic pwl voltage range feature. This feature conducts the following additional steps before the static check:

1. Search for any pwl or pulse Vsrc definition in the netlist
2. For any found pwl or pulse Vsrc, obtain its voltage range info (Vmin, Vmax)
3. Based on the obtained voltage range information, perform an operation equivalent to "fvsrcnd(<VsrcNodeName>, Vmin, Vmax)"

The default is 0.

Note:

Among the -connSub, -connInst, and -connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Description

The cckPmosG_lt_DS command examines every p-MOSFET's channel-connected paths from drain/source and gate. If the threshold for logic-high voltage is defined by vhth=vh. If there is a possible channel-connected path from source or drain to a logic-high voltage source VPP, check the channel-connected path from gate to another logic-high voltage source VDD. If the minimum of gate voltage, minVg, is less than the maximum of D/S voltage, maxVd/s, CircuitCheck reports a warning about this p-MOSFET and the resulting VPP and VDD, because the p-MOSFET may cause a leakage path. Refer to Figure 17 on page 221.

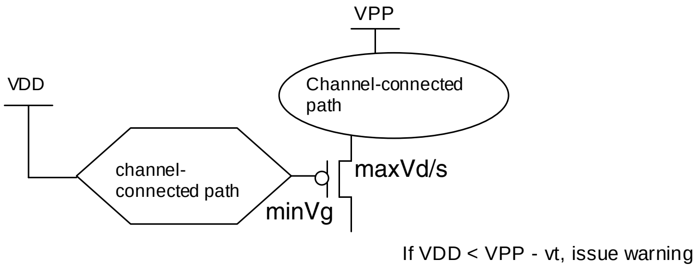


Figure 19 VDD < (VPP – vt) = Report Warning

Note:

The following values are used in voltage drop calculations along with path tracing:

- vnth: n-MOSFET turn-on threshold.
- vpth: p-MOSFET turn-on threshold.

The command allows some portions of the design to be examined by using subckt and inst options. During the channel-connect path tracing, it allows some specific instances to be skipped by using skipSub and skipInst. In the reporting, some instances are repressed using rmSub and rmInst. In addition, it reports the voltage range at certain nodes with rptv=[0|1].

When rptTrace=1, two paths of transistors are included for each device to be printed. The first one is a path from a reported device gate node to a voltage source; the 2nd list is a path from its drain or source node to lead to another power supply.

cckPmosG_lt_DS and related commands are used immediately after HSIM netlist parsing as described in the following:

- Operation parsing is static based and vt in the model file is not used.
- vnth and vpth in the command option are used for voltage drop consideration along the path tracing process.

Examples

The following example checks all the p-MOSFETs in instance xcm. Find the channel-connected paths from drain or source to any high voltage source whose value is larger than 0.8V. Let the highest voltage of source node be maxVd/s. Then, look for the channel-connected path from gate to a positive voltage source whose value is larger than vhth=0.8V. Let the lowest voltage source value seen from gate be minVg. If maxVg is smaller than minVd/s by 0.3V, issue a warning.

```
cckPmosG_lt_DS vhth=0.8 inst=xcm* vt=0.3 vnth=0.6 vpth=-0.5
```

The following is the output sample resulting from the command example above.

```
****************************************************
* PMOS gate vsrc value less than D/S vsrc: inst=xcm
* vhth=0.80    vt=0.30
* format: instName gate_volt d/s_volt
****************************************************
 xcm.xdec.xu1.mp1            Vg (1) Vs (1.65)
 xcm.xdec.xu6.mp2            Vg (1) Vs (1.65)
 xcm.xctl.mu2                Vg (1) Vs (1.65)
```

In the following example, all of the p-MOSFET in subckt cpu and the p-MOSFET whose names starting with x1 and x2 in subckt fifo are checked. The command traces to all the logic-high voltage sources, whose values are larger than 0.8V. During the traversing, it will skip all the transistors in subckt mux2.

```
cckPmosG_lt_DS vhth=0.8 vt=0.3 vnth=0.6 vpth=-0.5 subckt=cpu
inst=* subckt=fifo inst=x1* inst=x2* skipSub=mux2 skipInst=*
rmInst=x4*
```

When the tracing is completed, it compares the gate node voltage with those of drain and source. If the gate voltage is smaller than drain or source by 0.3V, it reports this device, unless this device's name is prefixed with x4.

The following example uses the rptTrace parameter:

```
cckPmosG_lt_DS subckt=aa inst=* vhth=0.7 vnth=0.5 vpth=-0.6
vt=0.3 rptTrace=1
```

For each transistor reported, a transistor path is printed which leads its gate node to a voltage source.

A sample of the above command example's output is:

```
xi.mp1     Vg (2.4) Vs (2.8)
     (G) wctrl
          thruI xi10.m2 (propagate 2.400)
          thruI xi10.r3 (propagate 2.400)
          thruI xia1.mp1 (propagate 2.400)
          frmNd vdd
     (S) wa
          thruI xi9.m1 (propagate 2.8)
          frmNd exvdd
```

The gate node of transistor xi.mp1 has a potential voltage of 2.4 volt, which is less than the source node's potential voltage (2.8 volt). Hence, this device can be partially conducting and cause large leakage current. To facilitate debugging the design, a path is shown connecting its gate node to 2.4 volt; a path connecting its source node to 2.8 volt. In this output, it shows that the gate node wctrl can reach VDD of 2.4 volt through xi10.m2, xi10.r3 and xia1.mp1. How voltage is propagated is also shown. If there is a voltage drop, it will be printed. For its source node wa, it is connected to exvdd which is at 2.8 volt through xi9.m1. By going through these two lists, the cause for potential large leakage current can be found.

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx       0 pvdd
vvde vvdex       0 pwl 0ns 0 0.1ns pvdd
vshift shift       0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckPmosG_lt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrcnd=1
```

CCK operates equivalently to following syntax:

```
cckPmosG_lt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrcnd=(vvdex, 0, 1.11)
fvsrcnd=(shift, 0, 0.8)
```

The report header is updated to include the autoFvsrcnd setting:

```
****************************************************
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*  vnth=0.300 vpth=-0.500
*  vhth=0.500
*
*  vt=0.10 num=300
*
*  autoFvsrcnd=1
*  format: instName gate_volt d/s_volt
****************************************************
```

Check PMOS Node To GND Connection

cckPmosNodeToGnd

Description

Checks a p-MOSFET's four nodes (drain/source/gate/bulk). If any one is connected to ground or logic-low voltage source (whose value is less than vhth), report it.

Usage Syntax 1

```
cckPmosNodeToGnd <tag=tagName> <num=n> <vlth=v>
   <inst=inst_name> <model=model_name> <node=drain>
   <node=source> <node=gate> <node=bulk>
```

Usage Syntax 2

```
cckPmosNodeToGnd <tag=tagName> <num=n> <vlth=v>
   <inst=inst_name> <model=model_name>
   <node='drain|source|gate|bulk'>
```

Note:

Usage Syntax 2 is a version of Usage Syntax 1 to simplify the "node" specification. Node='drain | source' is the same as node=drain node=source.

Parameters

tag

Specify the label in the report.

num

Specify the total number of violations to be reported

vlth

Specify the threshold voltage to be checked. The default is 0.4V.

inst

Specify the instance name in the design to be checked.

model

Specify the model to be checked.

node

Specify the nodes to be checked. The keyword, "*" and "all", are supported for checking on four terminals of a PMOS device.

Note:

Multiple cckPmosNodeToGnd commands can be specified in the CCK command file.

Examples

The following two examples are equivalent. They check the node voltage inside instance, x03 with model=pmos. If the voltages of the gate node and drain node are less than default vlth (0.4V), they will be reported.

```
cckPmosNodeToGnd tag=tag2  node=gate node=drain inst=x03.*
model=pmos
cckPmosNodeToGnd tag=tag2  node='gate| drain' inst=x03.*
model=pmos
```

The following example checks the four terminals of the PMOS device in a design, since "node=*" is specified and reports the violations if the terminal voltage is less than 0.3V.

```
cckPmosNodeToGnd tag=tag4 vlth=0.3 node=*
```

The following example checks the four terminals of the PMOS device inside instance x03 with model name, pmos, because the keyword, "all", is used in the "node=" control parameter.

```
cckPmosNodeToGnd tag=tag5 vlth=0.7 node=all inst=x03.* model=pmos
```

Safe Operating Area Check

cckSOA

The cckSOA command checks for any violation against the safe operating area (SOA) limits specified via the cckSOA constraint expression. If a default is discovered, a Warning message is printed.

Syntax

```
cckSOA <inst=instanceScope> <label="labelName">
   <model=typeOfModel|VA_model_name>
   <num=numberOfViolationTimeWindow>
   <constraint=ConstraintExpression> <start=start_time>
   <stop=stop_time> <filterAlert 0|1>
```

Parameters

inst

Specifies the scope of devices in the circuit within which the cckSOA check will be applied.

label

Specifies a label/tag to be used in the report file for distinguishing each different cckSOA commands used.

model

Limits the inst scope to be checked one step further so that the check is applied just on devices with the same model as specified in this option.

You can also specify a Verilog-A module name, which means cckSOA performs a port voltage check on the instances instantiated from the matched Verilog-A module.

num

Tells cckSOA how many time windows of violation to be checked and reported. The num default is 1. If num is negative or 0, then the check continues until the simulation stops.

constraint

Expression uses the form: '(condition_expression, lower_bound, upper_bound, time_duration)'

You can also specify a constraint expression to perform analysis on a targeted Verilog-A module port. For example:

```
cckSOA label=test model="resVA1" constraint=’(v(in1_b) <
2.0, 0, 0, 5n)’
```

In this example, cckSOA looks for voltage of “in1_b” port defined in “resVA1” Verilog-A module. If the port voltage value is less than 2v and the condition lasts longer than 5ns, it is reported as violation.

Use '(and)' to delimit the constraint expression. Where condition_expression can be any [+ – * /] combination of cckSOA constraint functions listed below. The cckSOA constraint functions presently supported include:

- I: get current of a DIODE
- IB: get BULK current of MOSFET or BASE current of BJT
- IC: get COLLECTOR current of BJT

- ID: get DRAIN current of MOSFET or JFET
- IE: get EMITTER current of BJT
- IG: get GATE current of MOSFET/JFET
- IS: get SOURCE current of MOSFET/JFET/BJT
- VB: get BULK/BASE voltage of MOSFET/BJT
- VBC: get BASE/COLLECTOR voltage difference of BJT
- VBD: get BULK/DRAIN voltage difference of MOSFET
- VBE: get BASE/EMITTER voltage difference of BJT
- VBS: get BULK(BASE)/SOURCE voltage difference of MOSFET(BJT)
- VC: get COLLECTOR voltage of BJT
- VCE: get COLLECTOR/EMITTER voltage difference of BJT
- VCS: get COLLECTOR/SOURCE voltage difference of BJT
- VD: get DRAIN voltage of MOSFET/JFET
- VDIP: get ANODE/CATHODE voltage difference of DIODE
- VDS: get DRAIN/SOURCE voltage difference of MOSFET/JFET
- VE: get EMITTER voltage of BJT
- VES: get EMITTER/SOURCE voltage difference of BJT
- VG: get GATE voltage of MOSFET/JFET
- VGB: get GATE/BULK voltage difference of MOSFET
- VGD: get GATE/DRAIN voltage difference of MOSFET/JFET
- VGS: get GATE/SOURCE voltage difference of MOSFET/JFET
- VNEG: get CATHODE voltage of DIODE
- VPOS: get ANODE voltage of DIODE
- VS : get SOURCE voltage of MOSFET/JFET/BJT

start

Specifies the start time for the SOA check.

stop

Specifies the stop time for the SOA check.

filterAlert

If set to 0 (the default) issues a warning if there are no matched devices. If set it to 1, issues an error message and terminates the CircuitCheck analysis if there are no matched devices.

When the given "condition_expression" is found to be lower than the "lower_bound" or higher than the "upper_bound" *and* it lasts for a time period longer than the "time_duration," then this time window of violation is reported. If this kind of violation happens frequently and exceeds the option "num" specified, then cckSOA will cease to check or report more time windows of violation.

Example

The following commands check the device element x3m.x0.x0.xsap.x1.mp1 with model type p with the constraint '(vgs, -1.5, 0.0, 15n)' expression; check if the function value vgs (Voltage difference between Gate and Source) is outside the range (-1.5, 0.0) consecutively for a time period longer than 15 nanoseconds. If such a violation occurs, it is reported. It also checks and reports occurrences of the violation up to 2 times then ceases checking. At the same time the commands also check on the same device with constraint expression '(id, -500u, 5u, 15n)' which focuses its drain current.

```
cckSOA inst=x3m.x0.x0.xsap.x1.mp1 Label="vgs" model=p num=2\
constraint = '(vgs, -1.5, 0.0, 15n)'
cckSOA inst=x3m.x0.x0.xsap.x1.mp1 Label="id" model=p num=2 \
constraint = '(id, -500u, 5u, 15n)'
```

Its violation report looks like the following:

```
***-----------------------------------------------------***
***-----------------      vgs     ----------------------***
Element: (x3m.x0.x0.xsap.x1.mp1) of model: 'p',
violates constraint: '(vgs, -1.5, 0, 15)'
@ the following time window(s):
{
     Time Window #1:(121.8300, 140.3690)--> time span=18.5390 ns
     Peak constraint values: low @(-2.4094), high @(0.0000)
}
{
     Time Window #2:(172.0070, 190.3690)--> time span=18.3620 ns
     Peak constraint values: low @(-2.4094), high @(0.0000)
}
***-----------------------------------------------------***
***---------------           id     --------------------***
Element: (x3m.x0.x0.xsap.x1.mp1) of model: 'p',
violates constraint: '(id, -0.0005, 5e-006, 15)'
@ the following time window(s):
{
     Time Window #1:(120.8400, 140.5300)--> time span=19.6900 ns
     Peak constraint values: low @(0.0000), high @(0.0010)
}
{
     Time Window #2:(170.8420, 190.5300)--> time span=19.6880 ns
     Peak constraint values: low @(0.0000), high @(0.0010)
}
```

Subcircuit-Based Voltage Analysis Using the Dynamic Approach

The cckDynSubV command lets you evaluate the signal degradation or the correct biasing, such as a signal propagating along parasitic nets or a chain of diode. It also provides the flexibility to check or monitor the voltage difference on multiple specified nodes during transient simulation.

cckDynSubV

Syntax

```
cckDynSubV tag subckt=<subckt_name> inst=<inst_name>
   constraint='expression' num=<n> duration=<val>
   start=<time> stop=<time> <separate_file=<0|1>
   <filterAlert 0|1>
```

Parameters

tag

Each cckDynSubV command must have a unique tag name as its first parameter. When several cckDynSubV commands are in one command file, these unique tags distinguish violations reported from the different commands. In the output file, each tag name is the leading string of each line, so you can easily identify which line is output by which command. Also, the tag name is referenced as part of output filename if separate_file is set to 1. For example:

```
cckDynSubV sub1_chk  subckt= sub1 constraint= '…' …
cckDynSubV inst1_chk  inst= subInst1 constraint= '…' …
```

subckt

Specifies the subcircuit name in which the node voltages are to be checked. If a subcircuit containing the nodes to be checked is instantiated more than once, and if the violations occur, these violations are all reported. This parameter is optional and does not support a “*” wildcard. If a subckt name matches the specified subcircuit and the constraint expression of nodes is applicable, then the expression in constraint is evaluated to determine if a violation occurs.

inst

Specifies the instance of a subcircuit or the instantiation of specified subckt in which nodes to be checked dwell. You can specify a hierarchical name of instances delimited by “.” character. This parameter is optional and does not support a “*” wildcard. If an instance matches with this inst scope and the constraint expression of node voltages specified is applicable on it, then the expression in constraint is evaluated to determine if a violation occurs.

constraint= ‘expression’

An expression is formed by voltages of nodes, logical operators (&&, ||, >, <, >=, <=, ==, !=) and mathematical operators (+,-,*, /, and abs()). A simple example is: constraint='((v(nodeA)–v(nodeC)) < v(nodeB) || (v(nodeB) >= 0.5 && v(nodeD)<1.0))'. When this expression is logically true, a violation is reported. The argument of “V” is a “node name”, where “node name” can be one of the following:

- An absolute (hierarchical) node name if no “subckt” or “inst” is specified. For example, “x0.x1.x2.x3.net4”. No wildcard is allowed in such an absolute node name expression.

- A (hierarchical) node name “relative” to “subckt” or specified “inst”. For example, if given “inst=x0.x1.x2.*” and there are lower-level subckt instances “x3” and “x4”, then “V(nd1)” means check node “nd1” of subckt instance “x0.x1.x2.x3” and “nd1” of subckt instance “x0.x1.x2.x4”. For example, if given “subckt=sub0” then “V(nd2)” means check voltage at “nd2” under subckt “sub0”.
- When inst= “x0.x1.x2.*” is given as the pattern, it matches with subckt instances like “x0.x1.x2.x3” as well as “x0.x1.x2.x5.x6”. That is, the wildcard “*” matches “x3” and “x5.x6” as a regular expression would normally do.

Note:

An expression must be quoted by single quotation marks “ ‘ “.

num=<n>

Limits the number of violations output by the cckDynSubV command. If the number of violations exceeds <n>, only <n> violation are generated. The default value is 300. If <n> is set to less than or equal to 0, it means unlimited.

duration

Specifies the time duration that the constraint expression sustains the logically true condition to trigger an error report.

start and stop

Specifies the start and stop time span to perform checking.

separate_file

When set to 1, specifies that errors to be reported are directed to separate file(s) per each tag in cckDynSubV command statement. The default value is 0. When separate_file=1, output files are named according to the following rules:

- If t e -o option (output_prefix) is issued in the HSIM^{plus} command line, it is used as the output file prefix. Otherwise, `hsim` is used as the default output file prefix.
- `nodev` is the first portion of default extension for cckDynSubV output files. For example:

  `output_prefix.nodev` or `hsim.nodev`
- When separate_file=1, an additional <tag> is attached at the end of output file name :

```
output_prefix.nodev_<tag>
```

For the following commands:

```
cckDynSubV abc subckt= abc constraint= ‘v(nd1)>3.0’
separate_file=1
```

```
cckDynSubV ijk subckt= ijk constraint= ‘v(nd2)<0.’
separate_file=0
```

The following output files, respectively, are created:

```
output_prefix.nodev_abc
```

```
output_prefix.nodev
```

filterAlert

If set to 0 (the default) issues a warning if there are no matched devices. If set it to 1, issues an error message and terminates the CircuitCheck analysis if there are no matched devices.

Examples

```
cckDynSubV test subckt=inv constraint= ‘( (V(nd1)-V(nd2)>3) ||
(V(nd1)-V(nd2)<1)’  duration=1n start=5n stop=10n
```

Performs a dynamic node voltage check from transient time 5ns to 10ns. The violations are reported if the voltage difference (V(nd1) – V(nd2)) is greater than 3v or less than 1v and such conditions remains longer than 1ns. The target nodes “nd1” and “nd2” are defined inside subckt “inv”.

```
cckDynSubV test1 inst=hsio* constraint= ‘(V(nd1) >=0 && V(nd2)-
V(nd3) > 0.5)’ duration=2n start=5n stop=10n num=200
separate_file=1
```

Performs a dynamic node voltage check from transient time 5ns to 10ns. A violation occurs if the constraint expression : “V(nd)>=0V AND V(nd2)-V(nd3) < 0.5” is satisfied. Only subckt instances with names starting with “hsio” are checked and reported. The total number of violations to be reported is limited to 200. The violations are in a separate file with extension nodev_test1.

```
cckDynSubV test2 subckt=fulladd inst=xful* constraint='(v(o1)> 0
&& v(c1)-v(c2) > 0.2)' duration=3n num=100 separate_file=1
```

Performs a dynamic voltage check throughout the entire transient simulation cycle. The expression looks for signal names such as “o1”, “c1”, and “c2” defined within subckt “fulladd”. The checking targets are on subckt “fulladd” instantiation with its instance name starting with “xful”.

The following example shows a cckDynSubV report:

```
* -------------------------------------------------------------
* Dynamic Node Voltage Check
* tag      =test2
* subckt   =fulladd
* inst     =xful*
* constraint= '(v(o1)> 0 && v(c1)-v(c2) > 0.2)'
* duration  =3n
* num       = 100
* separate_file = 1
* twindow   = (0, max)

* ---------------------------------------------------------
Instance: xful2
Violation Data:
Duration: (41.000, 51.961) --> time span = 10.961 ns
  at 41.000 ns: xful2.o1(v=4.95532) xful2.c1(v=2.4666)
xful2.c2(v=2.2018)
  at 51.961 ns: xful2.o1(v=0.861179) xful2.c1(v=5.12874)
  xful2.c2(v=4.92276)

Instance: xful3
Violation Data:
Duration: (42.395, 52.203) --> time span = 9.808 ns
  at 42.395 ns: xful3.o1(v=4.05138) xful3.c1(v=4.61114)
  xful3.c2(v=4.37931)
  at 52.203 ns: xful3.o1(v=0.801633) xful3.c1(v=5.0937)
  xful3.c2(v=4.87613)

Instance: xful2
Violation Data:
Duration: (80.967, 91.961) --> time span = 10.994 ns
  at 80.967 ns: xful2.o1(v=4.98497) xful2.c1(v=2.40586)
  xful2.c2(v=2.17932)
  at 91.961 ns: xful2.o1(v=0.864535) xful2.c1(v=5.12876)
  xful2.c2(v=4.9235)

** Total number of violations = 3
```

Substrate Forward Bias Check

cckSubstrate

Checks whether a MOSFET substrate becomes forward-biased.

Syntax

```
cckSubstrate <mode=[2|1|0]> <num=n> <vt=v> <ith=iv>
   <start=t1> <stop=t2> <model=m> <tag=t>
```

Description

The cckSubstrate command checks whether a MOSFET substrate becomes forward-biased. This check is performed before DC initialization and during transient simulation. In simulation, once a MOSFET's substrate becomes forward-biased by more than a threshold, a Warning message is printed.

Parameters

mode

Since checking is performed before DC initialization or during transient simulation, the mode parameter is used to control when checking is accomplished.

- Mode Set to 0

  When the mode is set to 0, every MOSFET's substrate connection is checked before DC analysis is performed. A warning message is issued if a MOSFET's bulk is connected as shown in the following:

  p-MOSFET: Ground or negative constant voltage source

  n-MOSFET: Positive constant voltage source

  The warning message is saved in one of the following files:

  hsim.cck: default

  out_file.cck: If -o out_file is used in HSIM invocation.

- Mode Set to 1

  When the mode is set to 1, a MOSFET's substrate is checked to see if it becomes forward-biased during simulation.

- Mode Set to 2

  When the mode is set to 2, both types of checking are accomplished.

num

Controls the number of warning messages issued.

vt

Set the voltage threshold (applicable to dynamic check only) to avoid printing out an excessive number of messages. If the forward bias voltage exceeds vt, a Warning message is printed. Default is 0.5V.

ith

> If ith is specified, the bulk current is computed under the forward bias condition. If the absolute value of this bulk current is greater than ith, this element will be reported. Default is 0 Amp.

start

> Specifies the start time for vt checks.

stop

> Specifies the stop times for vt checks.

model

> MOSFET type to be checked model is user-selectable.

tag

> t is printed in the report file at the beginning of the readability line.

Examples

```
cckSubstrate mode=2 num=300 vt=0.8 start=10n stop=50n start=90n
stop=120n
```

Since mode=2, the entire MOSFET substrate is checked before performing DC initialization and during transient simulation.

Note:

> There is a limit of 300 Warning messages that can be specified as shown in the example above.

During simulation, if a substrate becomes forward-biased by more than 0.8V, a Warning message is printed. The following explanation illustrates when a Warning message is printed.

At 10 ns, a p-MOSFET’s drain is 3V and its substrate is 2.1V. This p-MOSFET is forward-biased by 3-2.1=0.9V which is larger than vt=0.8V. Hence, a Warning message is printed. Checking is accomplished between 10 ns and 50 ns and between 90 ns and 120 ns.

The following is the MOSFET Substrate Checking Before DC output sample resulting from the command example above.

```
*********************************
* MOS Substrate Checking Before DC
*********************************
in subckt (TLC), pmos (xi.pg)'s bulk (gnd) is connected to 0.00 volt
in subckt (TLC), nmos (xi2.mi10)'s bulk (vdd) is connected to
3.00 volt
```

In the above example, the following definitions apply:

- xi.pg: p-MOSFET element name
- gnd: Node name
- xi2.mi10: Element name
- vdd: Node name
- TLC: Top-level circuit

The following is the MOSFET Bulk and Diode Forward-biased In Simulation output sample resulting from the same command example:

```
**** ***********************************************
* MOS Bulk and Diode forward-biased In Simulation
*     bulk forward bias: model=n1a, threshold=0.8 tag=n1
*********************************************
(n1)@20.86n, nmos (xu5.mn) bulk forward-biased
v(b)=0.0000 v(d)=-1.51 v(s)=0.0000
```

This sample reads: At 20.86ns, n-MOSFET xu5.mn's bulk becomes forward-biased with the following parameters.

- Bulk voltage (v(b)) is 0V
- Drain voltage n1a(v(d)) is –1.51V.

This element is reported since the difference derived by the following formula is greater than the default of 0.8V.

```
(0-(-1.51))=1.51V
```

Note:

In some case, the substrate forward bias check may produce a huge file. num is used to limit the amount of output.

Note:

In transient simulation, both the substrate and diode forward bias are checked at each time interval and shows both results under the same header.

Unprotected Antenna Node Check

cckAntGate

Checks if an antenna node is protected by diode.

Syntax

```
cckAntGate <num=n>
```

Parameters

num

limit the number of warnings, default is 300

Description

This command checks if an antenna node is protected by diode. If an antenna node has no connection to any reversed-bias diode, a warning is issued. An antenna node is defined as primary the input at top level, driven by ideal voltage sources, and it has direct connection or through resistor/inductor to transistor's gate.

Static Voltage Propagation Sharing

Multiple commands can share the same static voltage propagation. The commands that can share static voltage propagation are:

- cckMosV
- cckCapV
- cckResV
- cckDioV
- cckPmosG_lt_DS
- cckNmosG_gt_DS
- cckPmosB_lt_DS
- cckNmosG_gt_DS

Propagation Parameters

Propagation parameters are the command parameters that affect the result of the static voltage propagation. The propagation parameters are:

- exttrace
- limitmos
- risepmosfallnmos
- vpth

- vnth
- off_vpth
- off_vnth
- vlth
- vhth

Propagation Sharing

Among all of the commands in the command file, groups of command that meet all the following conditions can share the same propagation:

1. Have the same command name
2. Use identical propagation parameters
3. Use exttrace=0

Note:

Command sharing does not depend on the ordering of the commands in the command file.

Example

Consider the following CircuitCheck command file:

```
cckStaticDCPath nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
cckPmosG_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
rpttrace=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
rpttrace=1
cckPmosB_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
cckNmosB_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
cckDiode mode=0 subinfo=1 num=1000000
cckfloatgateisrc 1 subinfo=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3
pcap=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3
pcap=1
cckMosV mn1 model=n7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mn2 model=n_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn3 model=na inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn4 model=n  inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.02 vnth=0 vpth=0 subinfo=1 num=1000000
rpttrace=1
cckMosV mp1 model=p7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mp2 model=p_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mp4 model=p  inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
```

There are 17 commands. 10 separate static voltage propagations are necessary in this case.

Propagations 1 to 5 (commands not supported for voltage propagation sharing):

```
cckStaticDCPath nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
cckDiode mode=0 subinfo=1 num=1000000
cckfloatgateisrc 1 subinfo=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
fanout=3 pcap=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
fanout=3 pcap=1
```

Propagations 6-8 (supported, but different command names are used):

```
cckPmosG_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1
num=1000000 rpttrace=1
cckPmosB_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1
num=1000000
cckNmosB_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1
num=1000000
```

Propagation 9:

```
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1
num=1000000  rpttrace=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.02 vnth=0 vpth=0 subinfo=1
num=1000000  rpttrace=1
```

Propagation 10:

```
cckMosV mn1 model=n7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3
subinfo=1
cckMosV mn2 model=n_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2
subinfo=1
cckMosV mn3 model=na inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2
subinfo=1
cckMosV mn4 model=n  inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mp1 model=p7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3
subinfo=1
cckMosV mp2 model=p_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2
subinfo=1
cckMosV mp4 model=p  inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
```

Digital Logic and Memory Diagnostics

These CircuitCheck commands are designed to help check stuck-at nodes, un-initialized latches, series MOSFET stack-up, Flash memory over-erase conditions, toggle count, and trace event triggers. These commands are presented in alphabetic order to make it easier to find them.

Flash Memory Check

Over-erasing and involuntary operation condition checking in flash memory cells.

cckFlashcore

Automatically detects over-erasing condition in flash memory cell and helps prevent involuntary operations during read cycle

Syntax

```
cckFlashcore <vtlow=vtlow_value> <vdsrdmax=vdsrdmax_value>
```

Parameters

vtlow

CircuitCheck will check if Vth < vtlow during the phase of erasing; the check will be disabled otherwise. If Vth < vtlow, then Warning messages will be reported.

vdsrdmax

CircuitCheck will check if Vds > vdsrdmax during the phase of reading; the check will be disabled otherwise. If Vds < vdsrdmax, then Warning messages will be reported.

Note:

If vtlow and vdsrdmax are used together, both checks will launch simultaneously.

Note:

If the vtlow or vdsrdmax is not specified, the corresponding check will be disabled.

Description

This command is especially helpful in flash memory cell design. It provides an automatic detection of over-erasing condition in flash memory cell and ensures that there is no risk of involuntary operation during the read cycle. With this CircuitCheck command, CCK generates Warning messages and reports the flash core cell instance if Vth becomes less than a specified threshold during the erasing cycle or if Vds is greater than a threshold during read cycle.

Note:

The Warning messages are stored in the hsim.cck or file_name.cck, where file_name is the output file name specified.

Examples

Adding cckFlashCore to the CircuitCheck command file: When the following command is added in the CircuitCheck command file after simulation is completed, a Warning messages is issued and stored in the .cck file if Vds > 4.5 V.

```
cckFlashcore vdsrdmax=4.5
```

The following is a typical warning message:

```
*
*
High Vds during reading.
     model mos1, vds=4.911000, time=1.500004e-005.
High Vds during reading.
     model mos1, vds=5.060000, time=1.500005e-005.
High Vds during reading.
     model mos1, vds=5.805000, time=1.500005e-005.
High Vds during reading.
     model mos1, vds=6.550000, time=1.500005e-005.
High Vds during reading.
     model mos1, vds=6.848000, time=1.500006e-005.
```

Note:

In the Warning message shown above, Vt check is disabled because the parameter, vtlow, is not specified.

Find Un-initialized Latch

cckLatchUnInit

Syntax

```
cckLatchUnInit 0|1
```

Parameters

1

Detects uninitialized latches (default)

0

Turns off `cckLatchUnInit`

Examples

```
cckLatchUnInit 1
```

Sample output from cckLatchUnInit:

```
***********************************
* Un-initialized Latch
***********************************
isolated latch node (xicg.xeset.n2)
isolated latch node (xicg.xeset.n1)
```

Description

A latch usually has a circuit loop. Figure 18 shows two circuits. The first circuit has a pass gate in the loop and the second circuit does not. C and C are signals connecting to GATE nodes of pass gate transistors. The initial conditions of the A and B nodes in the loop will affect the simulation result. If no node is driven at the initial stage, CircuitCheck reports it.

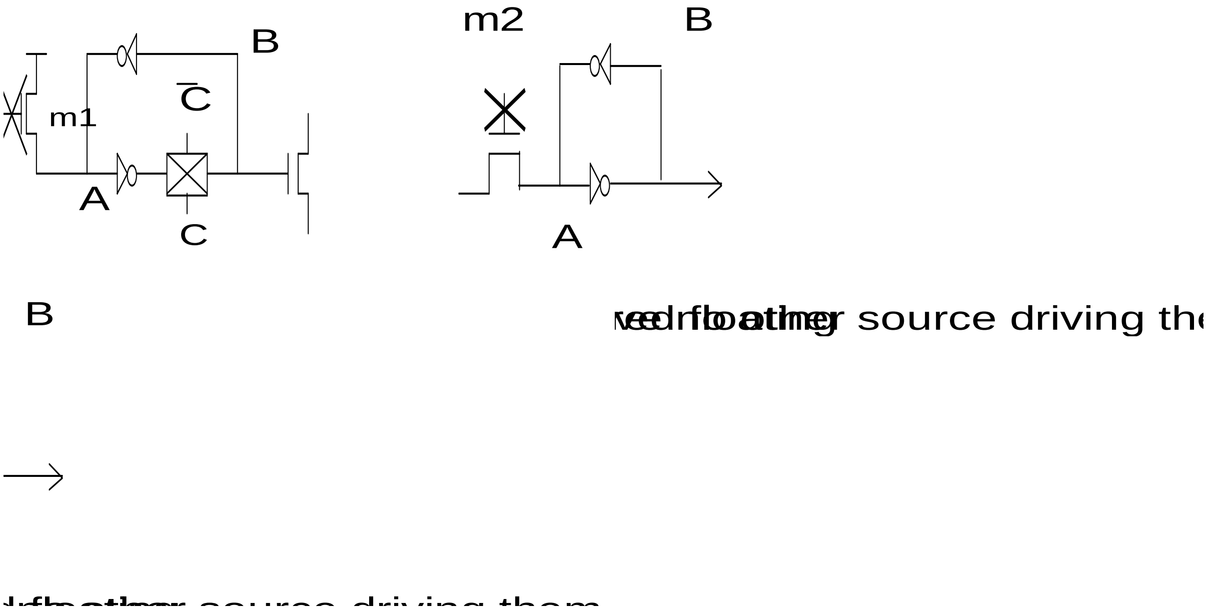


Figure 20 cckLatchUnInit: Signals connecting the Pass Gate through Nodes A & B

cckLatchUnInit checking is performed after DC initialization. In the first circuit shown in Figure 18 on page 237, if m1 is OFF and the pass gate is ON due to C/C. A and B have no other elements driving them. Therefore, they are considered to be floating.

cckLatchUnInit has a group of sub-commands which are described in the following sections:

- cckLatchInElem on page 238
- cckLatchSkipElem on page 238

cckLatchInElem

This is a sub-command for cckLatchUnInit and directs it to only check latches in a portion of the design

Syntax

```
cckLatchInElem <subckt=s> <inst=e>
```

Examples

The following command directs cckLatchUnInit to check latches in only a portion of the design, instances xm and xcpu:

```
cckLatchInElem <inst=xm*> <inst=xcpu*>
```

In the following command, CircuitCheck checks latches in all the instances of subckt xlatch1 and xlatch2:

```
cckLatchUnInit 1
cckLatchInElem subckt=xlatch1 subckt=xlatch2
```

In the following command, the options subckt and inst specify instance and/or instance(s) instantiated from a specific subckt for cckLatchUnInit to check the latches.

```
cckLatchUnInit 1
cckLatchInElem <subckt=s> <inst=e>
```

cckLatchSkipElem

Directs CircuitCheck to check all the latches except the specified portion.

Syntax

```
cckLatchSkipElem <subckt=subckt_name> <inst=inst_name>
```

Description

This is a subcommand for cckLatchUnInit and when cckLatchUnInit 1, this subcommand directs CircuitCheck to check all the latches except the specific portion defined in the cckLatchSkipElem command. The options `subckt` and `inst` specify specific instance and/or instance(s) instantiated from a specific subcircuit for `cckLatchUnInit` to skip during its checking operation:

Example

In the following example, CircuitCheck checks all the other specified latches except those in instance xram:

```
cckLatchSkipElem inst=xram.*
cckLatchUnInit 1
```

Check Stack-up Transistors

CircuitCheck can verify the number of MOSFETs in series. If it exceeds the stack-up length, CircuitCheck reports the MOSFET stack path.

cckMaxStackUpNmos

Reports any path of more than predetermined n-MOSFET transistors in series.

Syntax

```
cckMaxStackUpNmos <length=len> <num=n>
```

Example

The following CircuitCheck command reports any path of more than three n-MOSFET transistors in series.

```
cckMaxStackUpNmos length=3
```

The following is the NMOS stack-up limit output sample resulting from the command example above.

```
*************************************
* NMOS stack up limit: 3
*************************************
Largest nmos stack: 4
     (1) stack length: 4
        xam.xd.xu_0.mn1
        xam.xd.xu_0.mn2
        xam.xd.xu_0.mn3
        xam.xd.xu_0.mn4
```

Note:

These four nmos transistors are in series.

cckMaxStackUpPmos

Syntax

```
cckMaxStackUpPmos length=n
```

Example

```
cckMaxStackUpPmos length=2
```

The following is the PMOS stack up limit output sample resulting from the command example above.

```
**************************************
* PMOS stack up limit: 2
**************************************
Largest pmos stack: 3
     (1) stack length: 3
          xm.xu_6.xu51.mp
          xm.xwr_6.xu.mnoen
          xm.xwr_6.xu.mpi
```

Check and Classify the Stuck Nodes

CircuitCheck reports any node stuck at a certain voltage. If the stuck value is positive, CircuitCheck treats it as stuck-at-1. Otherwise, it is stuck-at-0. This type of Warning will be reported in .cckstuck0 and .cckstuck1 files.

cckMaxStuckAt

Syntax

```
cckMaxStuckAt0 <num=n> <node=nd> <skipSub=skip_sub_name>
   <skipnode=skip_node_name>
cckMaxStuckAt1 <num=n> <node=nd> <skipSub=skip_sub_name>
   <skipnode=skip_node_name>
```

Parameters

num=n

Limits the number of Warnings in each category to n.

node=nd

Defines the portion of the design to be checked for stuck-at nodes.

skipSub

Skips the specified sub-circuit.

skipnode

Skips the specified node.

Description

The number of warnings can be limited using `num`. CircuitCheck also permits selection of which portion of a design is checked using `node`. Refer to the example below.

Example

```
cckMaxStuckAt0 <num=300> <node=xm.*>
```

CircuitCheck checks the stuck-at nodes in instance xm.

cckToggleCount

Checks every node in the design.

Syntax

```
cckToggleCount <vref=v> <skipSub=skip_sub_name>
   <skipnode=skip_node_name> <start=start+tim1
   <stop=stop_tim1 <start=start_tim2 <stop=stop_tim2...
   <tc0=0|1>>>>
```

Parameters

vref=v

Specifies the reference voltage to be used.

skipSub

Skips the specified sub-circuit.

skipnode

Skips the specified node.

start, stop

start and stop specify the time window(s) in which cckToggleCount is checked. Multiple start and stop parameters can be specified.

tc0=0|1

A value of 0 (the default) disables the additional zero-toggling report, so only the active signals are reported. A value 1 enables the additional zero-toggling report.

Examples

```
cckToggleCount vref=5.0 skipsub=cnand start=20n stop=40n
start=60n stop=80n
```

If any node crosses the vref=v reference voltage, the toggle count at this node is increased by 1. At the end of the simulation, the toggle count is reported for every node. This report is used to compute the toggle frequency at each node and then multiply with the node capacitance to estimate the power consumption. The output is located in either the hsim.ccktoggle or output.ccktoggle file.

Example 35 Sample hsim.ccktoggle File Output.

```
*Synopsys Corporation.
*HSIM Win32 Debug Version 5.0 - 175607242004
*Tracking No - HSIM 2004.30.3
*Copyright (C) 1998 - 2004. All rights reserved.
*
*
* cckToggleCount vref=0.7
*   format: node_name toggle_count
aa<0> 1
aa<12> 4
aa<16> 1
dd<0> 1
xic5m29f016b_verilog.xibaaps.i25_d 1
xic5m29f016b_verilog.xibaaps.i26_d 2
```

cckConnReport

Performs a high connectivity node check.

Syntax

```
cckConnReport <vconnth=<value> <subinfo=[0|1]>
```

Parameters

vconnth=value

Specifies the device count threshold value. CCK reports node names with connected devices greater than the specified threshold value. The default is 500.

subinfo=[0|1]

If set to 1 specifies that the additional subcircuit information is printed along with the device full hierarchy name. The default is 0.

Note:

The reported node names are in the full hierarchy naming convention with the corresponding subcircuit name information (if subinfo=1 is set), and the node names are sorted in descending order by the associated connected device count value.

Examples

```
cckConnReport  connth=1000
```

CCK reports the node name with a connected device count greater than 1000 and generates the following report:

```
*
* CCK Command : cckConnReport
* connth=1000 subinf
o=1
*
*
Node #1: vss
Subinfo : TLC
Data :
  Connections: 50064
Node #2: vdd
Subinfo : TLC
Data :
  Connections: 31965
Node #3: vcci
Subinfo : TLC
Data :
  Connections: 4780
Node #4: xids3234a1.vdd_tempcore
Subinfo : d32a34a1_g171
Data :
  Connections: 4210
Node #5: xids3234a1.xcontrol.net138
Subinfo : d32a34a1_g171 control_g167
Data :
  Connections: 3320
```

Interactive Circuit Debugging Command for Tracking Circuit

CircuitCheck traces the cause of a digital state change at a given node. Refer to the HSIM Simulation *Reference Manual: Chapter 12, Interactive Debugging, List of Interactive Mode Commands* for information on obtaining the interactive debugging environments.

Note:

> It is strongly recommended that all the logic signals are printed out before using this feature. This is accomplished by using the lprint v(*) command. The hsimvdd value must also be set to specify the low and high voltage logic thresholds. The commands used to track circuits are based on FSDB files and must be specified as FSDB format to use the commands.

Note:

In a large design, the FSDB file can be created by first running the HSIM simulation. At a later time in the process, enter the Interactive Mode and use the previously created FSDB files for the ntrig commands. This reduces the number of times that simulations for ntrig commands must be rerun.

Finding a Node’s First State Change After a Specified Time

ntrig

Syntax

```
ntrig node_name <-t time> <-f file1> <-mt time_interval>
```

Description

`file1` is used to find the first state change for the node after the specified time in nanoseconds. This command only finds the last signal to trigger a change in a given node, unless -mt is specified. If -f is not specified, the current FSDB file will be used. If -t is not used, the default time is 0.

Very often a state change is caused by a combination of several other signals. For example, the output of a NAND gate is changed due to the changes of input signals. When –mt is given, this command expands the search to find more signals causing this given node to change. The time_interval in nanoseconds is the period of backward time travel specified to find multiple inputs.

Examples

Example 36

```
HSIM> ntrig -t 9     m188:gate
time=9.000000 ns
 fsdb file #1=hsim.fsdb.
 signal=m188(10)
 Only 1 possibility.
 possibility: #1
   m188:gate from 0 to 1 in [9.3260n, 9.3550n]
<- m177(1) from 1 to 0 in [9.2870n, 9.3090n]
<- m189(2) from 0 to 1 in [9.2470n, 9.2760n]
<- m429(3) from 1 to 0 in [9.2070n, 9.2290n]
<- m432(4) from 0 to 1 in [9.1680n, 9.1970n]
<- n_276(5) from 1 to 0 in [9.1280n, 9.1500n]
<- m173(6) from 0 to 1 in [9.0870n, 9.1180n]
<- i_22221.m31(7) from 1 to 0 in [9.0380n, 9.0750n]
<- i_22221.m28:gate from 0 to 1 in [8.9790n, 9.0200n]
 i_22221.m28:gate is a voltage source!
```

Example 37

```
HSIM > ntrig out -t 50 -mt 12
Node out (4):

Total number of possibilities: 2

possibility: #1
   out (4) from 1 to 0 in [52.06n, 52.57n]
<- s1 (6) from 0 to 1 in [51.03n, 51.62n]
<- in2 (2) from 1 to 0 in [50.29n, 50.70n]
in2 is a voltage source!

possibility: #2
   out (4) from 1 to 0 in [52.06n, 52.57n]
<- s3 (8) from 0 to 1 in [43.37n, 44.22n]
<- s2 (7) from 1 to 0 in [41.60n, 43.08n]
<- in3 (3) from 0 to 1 in [40.29n, 40.70n]
in3 is a voltage source!
```

The out node changed its state from 52.06 ns to 52.57 ns. Since -mt is specified, the search will travel backward to 40.67 ns (=52.57 – 12). It finds nodes s1 and s3 changed their states at 51.03 ns and 43.37 ns, respectively. These two nodes will be reported. Also node s1 was triggered to change by node in2 at 50.29 ns. Since in2 is a voltage source, the tracing from s1 is stopped. The other possibility is from s3 to trace to s2 and finally to in3.

Using ntrig to Trace a Circuit for State Change Causes In addition, the ntrig command traces the circuit for the cause of a state change according to one FSDB file and retrieve the corresponding signals of another user- specified FSDB file for comparison.

Syntax

```
ntrig <-t time> <-f file1> <-cf file2> node_name
```

Note:

file1 and file2 are fsdb files.

Description

file1 is used to find the first state change for the node after the specified time. If -f is not set, the FSDB file of the current run is the default. If -t is not set, the default time is 0. file2, needs to be specified in order to compare values from two FSDB files.

Examples

Example 38

```
HSIM> ntrig -t 3000    -cf try.fsdb d0
     time=3000.000000 ns
     fsdb file #1=hsim.fsdb.
     fsdb file #2=try.fsdb.
     signal=d0
     possibility: #1
          d0
               file 1: 1 to 0 in [3053.8080n, 3053.8410n]
               file 2: 1 to 0 in [3053.8060n, 3053.8390n]
          <- xi259.net86
               file 1: 0 to 1 in [3053.7510n, 3053.7900n]
               file 2: 0 to 1 in [3053.7490n, 3053.7880n]
          <- xi259.net71
               file 1: 1 to 0 in [3053.7020n, 3053.7250n]
               file 2: 1 to 0 in [3053.7000n, 3053.7230n]
          <- xi259.net75
               file 1: 0 to 1 in [3053.6560n, 3053.6760n]
               file 2: 0 to 1 in [3053.6540n, 3053.6740n]
          <- d0i
               file 1: 1 to 0 in [3053.6370n, 3053.6550n]
               file 2: 1 to 0 in [3053.6350n, 3053.6530n]
          d0i is a voltage source.
```

A compare.rc file will be created each time the ntrig command is used to assist in bring up all the identified digital signals in the nWave environment. The compare.rc file is used as follows:

- Select File->Restore Signals in nWave
- Choose compare.rc

Using ntrig to Find the Difference in Two Initial Conditions Sometimes it is useful to find out why at a certain time specified by t1, a node is at a specific digital state while in another simulation run, the same node is at a different state. It is valuable to know what other signals caused this node to be at different states at a given time. This is achieved using ntrig -ic, as follows:

Syntax

```
ntrig -ic <-t t1> <-f file1> <-cf file2> node_name
```

Description

This feature compares two files (file1 and file2) to find out at time t1 what other signals may cause this node to have different states. Time is measured in nanoseconds. If -t is not set, the default time is 0. If -f is not set, the current FSDB is a default file. The second FSDB file (file2), needs to be specified for comparison. In this case, the program only examines at time t1 to determine what made the given node to be at different states.

Example

```
HSIM > ntrig c –ic –t 10 -cf w.fsdb
Only 1 possibility.
possibility: #1
c (6) voltage: 0 vs 1.
<- a (9) voltage: 0 vs 1.
<- d (11) voltage: 1 vs 0.
```

Node c at time 10 ns has two different states: 0 in current fsdb; 1 in w.fsdb. It is because two input signals a and d affecting node c are in different states. Node a is 0 in current fsdb; but it is 1 in w.fsdb. Node d is 1 in current fsdb; but it is at 0 in w.fsdb.

Using ntrig to Find the Difference in Digital Value at a Given Time
Sometimes it is beneficial to trace down what causes a signal to have different digital values in two runs. The ntrig command is able to accomplish this. The ntrig -diff command finds the difference in a signal’s digital value at a given time:

Syntax

```
ntrig node_name -diff <-t time> <-f file1> <-cf file2>
```

Description

The second fsdb file (file2), needs to be specified for comparison. If this given node, at the given time, has different values from these files, it is traced backward in time domain to report the cause of the difference.

Example

```
HSIM > ntrig out -t 70 -diff -cf w.fsdb
     time=70.000000 ns
     fsdb file #1=hsim.fsdb.
     fsdb file #2=w.fsdb.
     Node out (6):
Only 1 possibility.
possibility: #1
     out (6)
          file 1: 0 to 1 in [66.3940n, 66.7470n]
          file 2: 0 to 0 in [66.3940n, 66.7470n]
     <- s2 (9)
          file 1: 0 to 1 in [64.8010n, 66.7470n]
          file 2: 0 to 0 in [45.8420n, 66.7470n]
     <- in3 (5)
          file 1: 1 to 0 in [45.0300n, 45.0350n]
          file 2: 1 to 1 in [45.0300n, 45.0350n]
     in3 is a voltage source!
```

Node out changed from 0 to 1 in current fsdb from 66.3940 ns. But in the other simulation run (w.fsdb) it stayed at 0. The program traced and found at 64.8010 ns, node s2 changed to 1 in current fsdb. It was due to node in3 which changed from 1 to 0 at 45.0300 ns. From the w.fsdb file, nodes s2 and in3 had different states. This helps to find out what triggered a node to change its state.

intrig

Another way to probe a node is to provide the node ID. The usage and functionalities are same as ntrig, except that the node hierarchical ID is used, instead of node names.

```
intrig node_id <-t time> <-f file1> <-mt time1>
intrig node_id <-t time> <-f file1> <-cf file2>
intrig node_id -ic <-t time> <-f file1> <-cf file2>
intrig node_id -diff <-t time> <-f file1> <-cf file2>
```

intrig Tutorial The following is an example of an interactive circuit debugging commands for tracking a circuit. Figure 19 on page 248 illustrates this example.

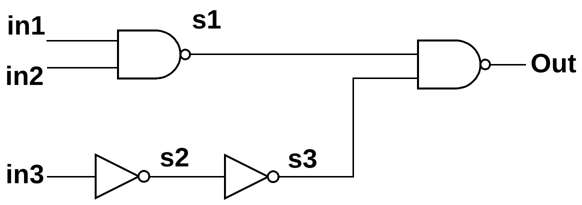


Figure 21 intrig Circuit

This example is stored in directory /$HSIM_HOME/tutorial/logic. The three primary inputs are in1, in2 and in3. Different choice of input pattern may lead to different output voltage, v(out). For this circuit, there are two different input vectors.

In the first run of testb.sp, the following syntax is used:

```
Vvdd vdd 0 5
r1 vdd in1 1
Va2 in2 0 pwl (0 5 5e-08 5 5.1e-08 0 10e-08 0)
Va3 in3 0 pwl (0 0 4e-08 0 4.1e-08 5 10e-08 5)
```

In the second run of testw.sp, the following syntax is used:

```
Vvdd vdd 0 5
r1 vdd in1 1
Va2 in2 0 pwl (0 5 5e-08 5 5.1e-08 0 10e-08 0)
Va3 in3 0 pwl (0 0 4e-08 0 4.1e-08 0 10e-08 0)
```

The only difference in the two cases is the in3 signal. The following 2 cases to demonstrate how to specify different options in ntrig command.

The steps for running this demonstration follow.

1. Run the runw script.

   Run script runw (top netlist is testw.sp). An FSDB file called w.fsdb will be generated. This file is required for later usage.

2. Enter the interactive mode.

   At the command line, type the following:

   ```
   hsim testb.sp
   ```

   HSIM will stop at 80 ns and enter the interactive mode.

3. Find the latest path.

When bringing up the waveform for the out node in the hsim.fsdb file, it shows that v(out) goes from 1 to 0 at around 52 ns. To find out the cause, type the following command at the interactive simulation mode.

```
HSIM> ntrig -t 50 out
```

The following signal list will be displayed:

```
out (4) has only one possibility:
possibility: #1
out (4) from 1 to 0 in [52.0600n, 52.5760n]
<- s1 (6) from 0 to 1 in [51.0370n, 51.6290n]
<- in2 (2) from 1 to 0 in [50.2990n, 50.7000n]
in2 is a voltage source!
```

4. Find multiple paths.

   Based on the schematics Figure 19 on page 248, the state change of the out may be influenced by both s1 and s3. The previous ntrig command only shows one path. In order to find out both paths triggering v(out) to digital low stage after 50 ns, the -mt option is required. Type the following command at the interactive mode:

   ```
   HSIM> ntrig -t 50 -mt 10 out
   ```

   The following will be displayed:

   ```
   out (4) has two possibilities:
   possibility: #1
   out (4) <2> from 1 to 0 in [52.0600n, 52.5760n]
   <- s1 (6) from 0 to 1 in [51.0370n, 51.6290n]
   <- in2 (2) from 1 to 0 in [50.2990n, 50.7000n]
   in2 is a voltage source!
   possibility: #2
   out (4) <2> from 1 to 0 in [52.0600n, 52.5760n]
   <- s3 (8) from 0 to 1 in [43.3740n, 44.2230n]
   <- s2 (7) from 1 to 0 in [41.6020n, 43.0830n]
   <- in3 (3) from 0 to 1 in [40.2990n, 40.7000n]
   in3 is a voltage source.
   ```

   Both paths are displayed. s3 changes its state at about 9 ns earlier than out. If the following command is issued and only one path is listed:

   ```
   HSIM> ntrig -t 50 -mt 8
   ```

5. Trace the difference in a node.

V(out) is at different states at time 60 ns in the first run testw.sp and current run testb.sp. To find out why they are different, type the following command at the interactive simulation mode:

```
HSIM> ntrig -t 60 out -diff -cf w.fsdb
```

The following will be displayed:

```
out (4) has only one possibility:
possibility: #1
out (4)
     file 1: 1 to 0 in [52.0600n, 52.5760n]
     file 2: 1 to 1 in [0.0000n, 52.5760n]
<- s3 (8)
     file 1: 0 to 1 in [43.3740n, 44.2230n]
     file 2: 0 to 0 in [0.0000n, 44.2230n]
<- s2 (7)
     file 1: 1 to 0 in [41.6020n, 43.0830n]
     file 2: 1 to 1 in [0.0000n, 43.0830n]
<- in3 (3)
     file 1: 0 to 1 in [40.2990n, 40.7000n]
     file 2: 0 to 0 in [0.0000n, 40.7000n]
in3 is a voltage source.
```

CCK outputs a comparison list of signals back-to-back.

6. Find the difference in two initial conditions.

   Type the following command at the interactive simulation mode:

```
HSIM> ntrig -t 60 -ic -cf w.fsdb out
```

The following will be displayed:

```
out (4)has only one possibility:
possibility: #1
out (4) voltage: 0 vs
<- s3 (8) voltage: 1 vs 0.
<- s2 (7) voltage: 0 vs 1.
<- in3 (3) voltage: 1 vs 0.
in3 is a voltage source.
```

Timing Checks

These CircuitCheck commands are designed to help check RC parasitic delay estimation, charge/discharge path delays, input slew rate, inter-nodal delays, and event times.

Check Number of n-MOSFET in Charging Path to VDD

Check the number of n-MOSFETs on every rising path to VDD. If it is larger than the specified length leng, CircuitCheck will issue a Warning until the number of Warnings exceeds num=n. This command is to make sure the charging-up paths will not have too many n-MOSFETs.

cckMaxNmosToVdd

Checks NMOS limit on charging-up path to VDD.

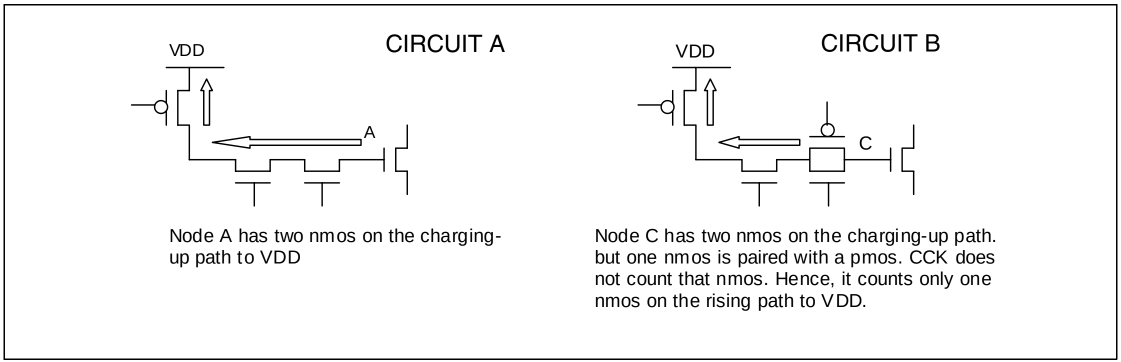
Syntax

```
cckMaxNmosToVdd <length=leng> <num=n>
```

Description

This command checks NMOS limit on charging-up path to VDD. The default value for the `length` parameter is 0.

In Figure 20 on page 251, if length=1, a Warning will be issued for Circuit A. Circuit B has a p-MOSFET accompanying a n-MOSFET, CircuitCheck will not count that particular n-MOSFET. Hence, no Warning is reported for Circuit B.



Node A has two nmos on the charging-up path to VDD

Node C has two nmos on the charging-up path. but one nmos is paired with a pmos. CCK does not count that nmos. Hence, it counts only one nmos on the rising path to VDD.

Figure 22 Example p-MOSFET and n-MOSFET Circuits

Example

```
cckMaxNmosToVdd length=2 num=30
```

The following is the NMOS limit on charging-up path output sample resulting from the command example above:

```
****************************************************
* NMOS limit on charging up path: 2, num=30
****************************************************
Largest number of nmos on rising path: 4
(1) number of nmos on path: 4
     end node (xam.xrd.nout)
          xam.xrd.xu8.mn
          xam.xrd.xu6.mn
          xam.xrd.xu7.mn
          xam.xrd.xu9.mn
          xam.xrd.xu24.mp
     from node (vdd) with 1.65 volt
```

Check Number of p-MOSFET in Discharging Path to GND

Check number of p-MOSFETs on discharging path to ground. If it exceeds the specified limit length=leng, a Warning will be reported, until the number of Warnings reaches num=n. Similarly to the previous command, if a p-MOSFET is paired with another n-MOSFET, this p-MOSFET will not be counted. See Figure 21 on page 252.

cckMaxPmosToGnd

Checks PMOS limit on discharging path to GND.

Syntax

```
cckMaxPmosToGnd <length=leng> <num=n>
```

The default value of length is 0.

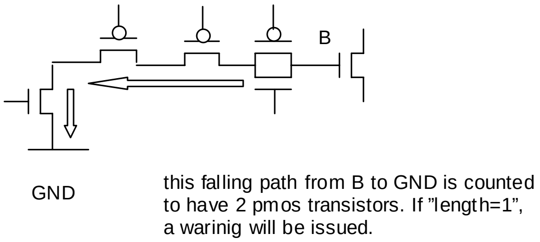

B
GND
this falling path from B to GND is counted
to have 2 pmos transistors. If ”length=1”,
a warinig will be issued.

Figure 23 Example p-MOSFET Circuit

cckMaxStackUpNmos

cckMaxStackUpNmos checks the number of n-MOSFET transistors in series. If it exceeds the stack-up length, CircuitCheck reports the MOSFET stack path. See cckMaxStackUpNmos on page 239 for detailed description.

cckMaxStackUpPmos

cckMaxStackUpPmos checks the number of p-MOSFET transistors in series. If it exceeds the stack-up length, CircuitCheck reports the MOSFET stack path. See cckMaxStackUpPmos on page 239 for detailed description.

Checking Path Delay Between Two Nodes

cckMeasPathDelay

Description

This command traces the path delay between the source node and target node. Differing from the .measure command that only measures the delay between source node and target node, cckMeasPathDelay has the path tracking capability that reports all delay paths between the source node and target nodes.

When it is necessary to measure the path between node A and node D, the source measures incremental paths between node A to node B, then node B to node C and finally node C to node D. This segmented path A to B, B to C, and C to D from the source to the target is reported using cckMeasPathDelay as shown in the following syntax:

Syntax

```
cckMeasPathDelay <source=source_node_name>
   <target=target_node_name> <outFile=output_file_name>
   <srcEdge=[1|-1|0]> <targEdge=[1|-1|0]>
   <start=start_time> <stop=stop_time>
   <post=fsdb_file_name>
```

Parameters

source

Specifies the source node name, permits wild cards in the expression, and searches all trigger nodes satisfying the parameter setting. For *, it will report all path triggering the target nodes.

target

Specifies the target node name. Wild cards are permitted in the expression however, they can not be as simple as a single asterisk (*). Wild cards must be specified similarly to the following example: target=data*.

outFile

Specifies the output file name for storing check results. If it is not provided, hsim.cck is the default and will be used.

srcEdge 1|-1|0

1: Looks for rising trigger nodes only.

-1: Looks for falling trigger nodes only.

0: (default) Looks for both rising and falling trigger nodes.

targEdge

Same as srcEdge.

start

Specifies the time span allowed for cckMeasPathDelay to perform its checks. If start and stop are not specified, it checks the entire transient time. If a value for start is specified, cckMeasPathDelay checks from specified start_time time to the end of transient simulation.

stop

Specifies the time span allowed for cckMeasPathDelay to perform its checks. If start and stop are not specified, it checks the entire transient time. If a value for start is specified, cckMeasPathDelay checks from 0 to the specified stop_time.

post

The analysis is conducted in the post-process mode. This means that the program directly analyzes the path measurement through the specified existing HSIM FSDB output file instead of looking into the current simulation data.

Note:

When using cckMeasPathDelay, lprint must be specified in the netlist file because cckMeasPathDelay uses the FSDB file to generate the report.

Examples

Adding cckMeasPathDelay to the CircuitCheck command file:

If the cckMeasPathDelay is added to the CircuitCheck command file as shown in the following syntax,

```
cckMeasPathDelay source=a_* target=out_* outFile=result.out
```

This command reports the delay paths with the node name beginning with a_ to the target node name with the prefix, out_. Because the srcEdge is not specified, it traces both the rising edge and falling edge.The result is written into the file called result.out. The example below shows a typical report.

```
* Measurement of Path Delay.
* source nodes: “a_*”.
* target nodes: “out_*”.
* source edge: don't care.
* target edge: don't care.
* Path delay measurement for target node: (out_a)
Path: #1
     out_a (525) rising in [1721.0000n, 1721.0200n]
          <- zz_i1/1 (4995) falling in [1720.9700n, 1721.0100n]
          <- 146 (12) rising in [1720.9200n, 1720.9800n]
          <- m2_i1/7 (3516) <2> falling in [1720.8400n,
1720.8800n]
          <- 147 (13) rising in [1720.7900n, 1720.8300n]
          <- div_i0/4 (1903) falling in [1720.1500n, 1720.1800n]
          <- 144 (10) rising in [1720.0800n, 1720.1500n]
          <- cp_i1/1 (1406) falling in [1720.0100n, 1720.0500n]
          <- a_a (4023) rising in [1719.9900n, 1720.0000n]
     a_a is a voltage source!
* total number of path found=1.
```

Estimating the Rise and Fall Delay at a Node

cckNodeMaxRF

Performs a quick estimate of the rising and falling delay at a given node.

Syntax

```
cckNodeMaxRF <node=fullPathNodeName> <tth=t1>
cckNodeMaxRF <subckt=subcktName> <node=nodeNameInSub>
   <tth=t1>
cckNodeMaxRF <skipNode=fullNodeName1>
   <skipNode=fullNodeName2> <tth=t1>
```

Parameters

tth=t1

tth is the time threshold. If a node has a Rise/Fall delay longer than the specified using tth, it will be reported.

Description

cckNodeMaxRF performs a quick estimate of the rising and falling delay at a given node. cckNodeMaxRF examines all the adjacent elements of this node to find the most resistive elements and calculate its effective resistance (Rmax). Then find its node capacitance (Cnode) which is multiplied by Rmax. The resulting product is an estimate of rising and falling delay to this node, since very likely, both the rising and falling paths will go through the most resistive element. This function does not trace from this node to VDD or ground.

When the estimated delay is larger than the time threshold (t1), it reports this node and its capacitance, its most resistive adjacent element, and the estimated delay. The output file is in either the hsim.cck or output.cck files. The nodes are selected by their full path names or by subckt with its nodes. Also, the command is used to check all the nodes except the skipped nodes.

The following command examines the nodes in the xam module and all wordline* nodes in all the instantiations of subcircuit atc. Estimated delays larger than 4ns are reported.

```
cckNodeMaxRF node=xam.* subckt=atc node=wordline* tth=4n
```

Sample cckNodeMaxRF output contents generated using the syntax in the example above.

```
 ****************************************************
* estimate node max rise/fall delay
* tth=4e-009
* subckt=atc node=wordline*
* node=xam.*
* format: node_name est_delay(ns) (elem_with_max_R ohm)
* (note: if an elem is always off, its ohm is 1e15)
****************************************************
xam.xctl.nlined2<5> 30.2 ns (max_R xcam.xctl.xi6_5.mu2 1e+5 ohm)
xam.xctl.nonseqc2 40.5 ns (max_R xcam.xctl.xi10.mu1 3e+5 ohm)
```

cckParasiticRC

cckParasiticRC performs statistical analysis of the parasitic RC during netlist loading to aid in post layout circuit debugging. See cckParasiticRC on page 170 for detailed description.

Static RC Delay Analysis – Estimate Slew Rate

Static RC Delay Analysis finds the RC delay to the GATE nodes of MOSFETs. The charging paths from VDD and the discharging paths from GND to each GATE node is found, and their RC delays are then computed. This analysis provides the following information for each signal:

- Circuit loading
- Slew rate

cckRCDlyPath

This command and its Delay Path Sub-Commands are used together to provide Static RC Delay Analysis.

Syntax

```
cckRCDlyPath [0|1] fanoutonly=[0|1]
```

When cckRCDlyPath is set to 1, CircuitCheck performs RC delay analysis after DC operating point analysis. Two files are created:

```
hsim.cckriseFor rise RC delay
hsim.cckfallFor fall RC delay
```

When the default fanoutonly=0 is set, the RC delay path can reach to either gate or output node. fanoutonly=1 only reports the RC delay path to the gate node.

```
cckRCDelayPath 1 fanoutOnly=1
```

If the HSIM –o out_file is issued, the delay files will be:

```
out_file.cckrise
out_file.cckfall
```

Note:

> When cckRCDlyPath is issued, CircuitChck automatically disables HSIM transient simulation.

Delay Path Sub-Commands

The following sub-commands are needed to report delay paths:

- cckDlyAtNode
- cckDlySkipElem

- cckDlySkipNode
- cckLimitRisePmosFallNmos
- cckRCFallDelay
- cckRCRiseDelay
- cckSetMosDir

cckDlyAtNode

Syntax

```
cckDlyAtNode <subckt=s> <node=nd>
```

Description

With this command, CircuitCheck analyzes the nodes specified in subckt and node.

Examples

```
cckDlyAtNode node=xcpu.*
```

CircuitCheck analyzes only the nodes in instance xcpu.

```
cckDlyAtNode subckt=xdp node=*
```

CircuitCheck analyzes the nodes in all the instantiations of subckt xdp.

cckDlySkipElem

Syntax

```
cckDlySkipElem <subckt=s> <inst=e> <pattern=p>
```

Description

This command directs cckRCDlypath to skip the elements specified in subckt, inst and pattern.

Example

```
CckDlySkipElem inst=xram.*
```

CircuitCheck analyses all the nodes except those in instance xram.

cckDlySkipNode

Syntax

```
cckDlySkipNode <subckt=s> <node=nd>
```

Description

With this command, CircuitCheck skips the nodes specified in subckt and node.

cckLimitRisePmosFallNmos

Syntax

```
cckLimitRisePmosFallNmos [1|0]
```

Description

Tracing paths: When the rising paths to VDD are traced, the default is to go through p-MOSFET only. Therefore, CircuitCheck ignores n-MOSFET elements, except for the transmission gates consisting of both p-MOSFET and n-MOSFET. When the falling path to GND is traced, the default is to limit the search within n-MOSFET elements only. Set cckLimitRisePmosFallNmos to 0 to change the default and expand the search through both p-MOSFET and n-MOSFET.

cckRCFallDelay

Syntax

```
cckRCFallDelay <min=dd3> <max=dd4> <inside=[0|1]>
```

Description

This command directs cckRCDlypath to check the rising delay.

For falling paths, the following are reported to the .cckfall file:

- Paths to GND
- Falling paths with a delay greater than max or less than min

A report on paths with delays within the <min, max> (=<dd3, dd4>) range is obtained by setting inside=1.

cckRCRiseDelay

Syntax

```
cckRCRiseDelay <min=dd1> <max=dd2> <inside=[0|1]>
```

Description

This command directs cckRCDlypath to check the rising delay.

For rising delay, the RC delays are computed along the transistor paths to VDD. The default is to report a rising path with a delay greater than max or smaller than min. If a report on rising paths with delays within a range such as <min, max> (=<dd1, dd2>) is required, set inside=1.

cckSetMosDir

Syntax

```
cckSetMosDir [1|0]
```

Description

Since RC delay analysis is a static circuit analysis, the methods used to eliminate false paths are:

- Transistor direction
- Inverter relationship to trim the paths

The default direction of MOSFETs must be determined before tracing the paths to compute the delay such as cckSetMosDir 1. Figure 22 on page 260 shows the direction. Transistors with hard-to-find directions are set as undefined.

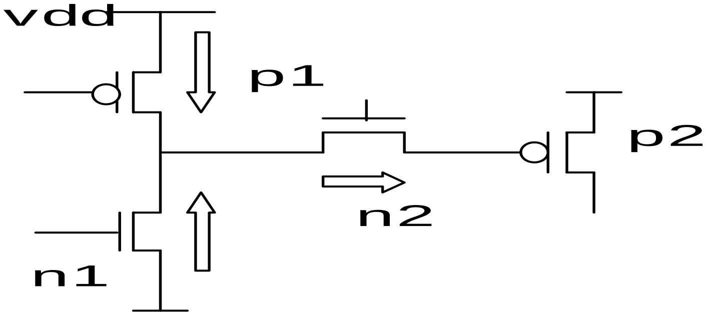


Figure 24 MOSFET Direction

In Figure 22 on page 260 in the transistor paths are:

- p-MOSFET (p1): Leaving VDD
- n-MOSFET (n1): Leaving GND
- Pass gate (n2): Entering the p2 gate

The paths may become invalid and ignored if they fall into one of the following criteria:

1. If two pass transistors are in the same subcircuit with an inverted relationship on the same path. Figure 23 on page 260 illustrates this condition.

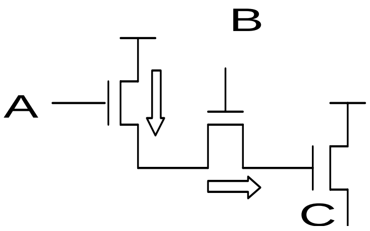


Figure 25 Transistors with an Inverted Relationship

If both pass transistors reside in the same subcircuit structure, and B is the reverse of A, then both transistors have an inverted relationship causing the path to be ignored

2. One or more pass transistors on the path is considered statically off.

   For statically off pmos: if the pmos gate is connected to a constant voltage and its Vg >= Max(constant voltage source in the circuit). If not applicable then the value of HSIMVDD is used as the threshold.

   For statically off nmos: if the nmos gate is connected to a constant voltage and its Vg <= 0.

Computing the Resistance of MOSFET

When CircuitCheck computes the resistance of a MOSFET it considers the parallel transistors and treats them as parallel resistors. This makes their effective resistance smaller in computing the delay. Figure 24 on page 261 illustrates a parallel transistor circuit.

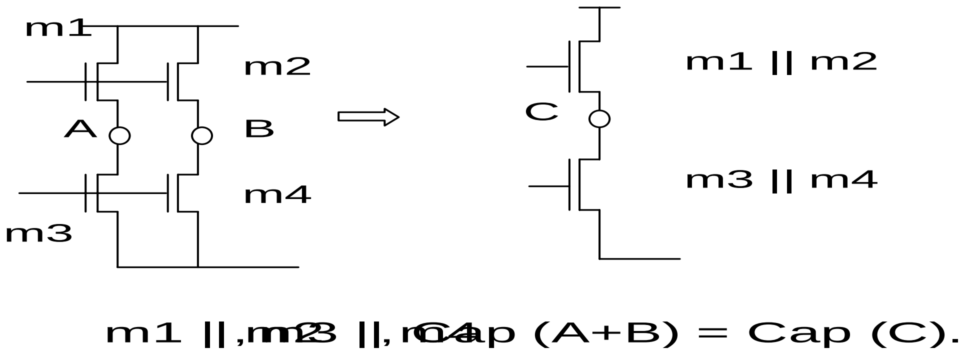


Figure 26 MOSFET Resistance

In Figure 24 on page 261 m1 and m2 are in parallel since they share the same input gate and drain node. m3 and m4 are also parallel. In addition, CircuitCheck also computes the effective capacitance at C. The capacitance value of C is the sum of the capacitance of nodes A and B.

Rising and Falling Path Delays

Elmore Delay Model Based on the Elmore Delay Model, the rising and falling delays are computed as illustrated in Figure 25 on page 262 and Figure 26 on page 262.

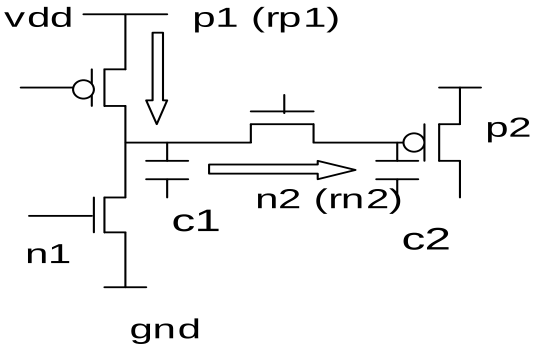


Figure 27 Rising Delay

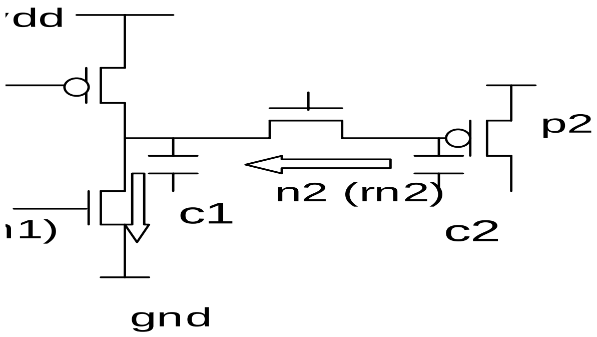


Figure 28 Falling Delay

```
cckRCDlyPath   1
cckRCRiseDelay min=0.8n max=3.e-9
cckRCFallDelay min=0.4n max=3.2e-9
cckLimitRisePmosFallNmos1
cckSetMosDir   1
```

This check finds rising/falling paths with the following delays:

- Rising: Delays > 3 ns or < 0.8 ns
- Falling: Delays > 3.2 ns or < 0.4 ns.

```
; Path format
; Serial_num Node_name Node_capacitance(pf) Rise_time(ns)
;             Elem_name Elem_type L/W or Resistance

Total number of paths=2

User defined value:
     rise_time: Ton-Time < 0.8ns || 3ns < Ton-Time

     S.N. Node_name C(pf)Rise_time(ns)
1    Node:xic.xib.fr880.1164.37
          xi10.mp1pmos2.6/8.2
          xi32.mp1pmos2.2/8.2
          * xi32.mn1nmos1.55/3.1
          x88inst.mi22pmos1.6/8.2

2    Node:xiaba.xi27.w160b0.1750.68
          xic.xiba.xib.mi49pmos1.8/8
          xic5m.ri50res38.4
```

Explanation of this Rising Path Report

Rising delay paths are sorted with descending delay values. Each rising path indicates the following:

- Ending node
- Node capacitance
- Rising delay to ending node

CircuitCheck lists the elements on a path from VDD to the ending node and the following element characteristics are printed:

- Element name
- Element type
- Transistor length
- Transistor width

If there is a parallel element, an asterisk (*) is printed before the element name.

Path 1 Referring to line item 1 in the example provided for Rising and Falling Path Delays on page 261:

Rising paths are sorted with the descending delay values.

```
Node:    xic.xib.fr88     0.116    4.37
```

This means that node xic.xib.fr88 has 0.116 pF capacitance. The rising delay from VDD to this node is 4.37 ns. The path is as follows:

```
xi10.mp1  pmos 2.6/8.2
xi32.mp1  pmos 2.2/8.2
* xi32.mn1nmos 1.55/3.1
x88inst.mi22pmos1.6/8.2
```

Note:

When an asterisk (*) is the first character in the line, it indicates it is a parallel transistor.

1st MOSFET From VDD, the path goes through a p-MOSFET xi10.mp1 with the following dimensions:

- Length: 2.6 um
- Width: 8.2 um

2nd MOSFET p-MOSFET xi10.mp1 goes through p-MOSFET xi32.mp1 and n-MOSFET parallel transistor xi32.mn1 with the following dimensions:

- Length: 2.2 um
- Width: 8.2 um

3rd MOSFET From p-MOSFET xi10.mp1, the path goes through n-MOSFET xi32.mn1 with the following dimensions:

- Length: 1.55 um
- Width: 3.1 um

Note:

These two transistors usually form a transmission gate.

4th MOSFET From n-MOSFET xi32.mn1, the path goes through p-MOSFET x88inst.mi22 with the following dimensions:

- Length: 1.6 um
- Width: 8.2 um

The signal then reaches the ending node.

Path 2 Referring to line item 2 in the example provided for Rising and Falling Path Delays on page 261:

1st MOSFET The second path goes through a p-MOSFET xic.xiba.xib.mi49 whose dimensions are:

- Length: 1.8 um
- Width: 8.0 um

2nd Resistor After passing through p-MOSFET xic.xiba.xib.mi49, the path goes through a 38.4 ohm resistor to reach this ending node. The ending node’s capacitance is 0.175 pF and the rising delay is 0.68ns.

Example 39 Falling Delay Output Sample hsim.cckfall or out_file.cckfall.

```
; Path format
; Serial_num Node_name Node_capacitance(pf) Fall_time(ns)
; Elem_name Elem_type L/W or Resistance
Total number of paths=2
User defined value:
fall_time: Toff-Time < 0.4ns || 3.2ns < Toff-Time

   S.N.  Node_name         C(pf)    Fall_time(ns)
1 Node:  xiba_out          0.150    4.15
         xi3.mn1           nmos     0.55/6.3
         xi25.mn1          nmos     0.55/11.58
2 Node:  xic5m2.i12_out    0.150    3.65
         xc.xi99.xi11.mn1  nmos     0.55/6.3
         xc.xim4.xi10.mn1  nmos     0.55/11.58
```

Explanation of this Falling Path Report:

Falling delay paths are sorted with descending delay values similar to the rising delay report. Each falling path indicates the following:

- Ending node
- Node capacitance
- Falling delay to ending node

CircuitCheck lists the elements on a path from GND to the ending node and the following element characteristics are printed:

- Element name
- Element type
- Transistor length
- Transistor width

If there is a parallel element, an asterisk (*) is printed before element name.

1st Element The first path's ending node is xiba_out) and has a capacitance of 0.15 pF. Starting from the GND node, the element is traced through n-MOSFET xi3.mn1) with the following dimensions:

- Length: 0.55 um
- Width: 6.3 um

2nd ELEMENT From n-MOSFET xi3.mn1), the path goes through n-MOSFET xi25.mn1) with the following dimensions:

- Length: 0.55 um
- Width: 11.58 um

Dynamic Device Voltage Check

A single command tcheck followed by numerous parameters are used to check device voltages during the simulation. When a condition is met, the device voltage is recorded and reported later. These commands are stored in a command file, which is invoked by adding the following to an input file of HSIM:

```
.param hsimDeviceV=dev_v_file
```

There are four types of devices allowed: MOSFET, BJT, diode, and capacitor.

tcheck mosv

Checks MOSFET device voltage.

Syntax

```
.tcheck tag_name mosv <model=name> <subckt=subckt_name>
   <mos=inst_name> <lvgd=val> <uvgd=val> <lvds=val>
   <uvds=val> <lvdb=val> <uvdb=val> <lvgs=val> <uvgs=val>
   <lvgb=val> <uvgb=val> <lvsb=val> <uvsb=val> <minL=val>
   <maxL=val> <cond='expression'> <report=#> <time=val>
   <parallel=0|1> <separate_file=1>
   <sort=tag|t1|el_name|node1|node2|err_v> <start=time>
   <stop=time> <step=time>
```

Parameters

tcheck

Keyword for transient analysis voltage check.

tag_name

A tag in the checking result file used to distinguish it from other information.

mosv

A key word for MOSFET node voltages checking.

subckt

Specify sub-circuit to be checked.

mos

Specify the instance to be checked.

model

Specify the model to be checked.

report

Specify how many times elements with errors will be reported for the associated tag. If not limited, CCK reports all violations by default.

time

Specify the time duration that the checking condition sustains before triggering an error report.

parallel

Used to reduce the number of errors reported for the parallel devices.

separate_file

Causes errors to be reported with a tag to a separate file.

start and stop

The time span allowed for HSIM to perform checking is specified.

step

Controls which time step to check.

Note:

mos, model, subckt, or any combination of these commands may be used to specify elements to be checked.

Note:

The output of a device voltage check is reported in either of the following files:

- hsim.mosv
- out_file.mosv (if hsim -0 out_file is used)

tag_name

Identifies a specific range of values that are specified in the hsim.mosv file.

Example

```
.tcheck tag1 mosv lvgs=1
```

Device voltages with a gate-to-source voltage difference < 1V will contain a tag1 prefix in the hsim.mosv file. For example, the hsim.mosv file will have a similar entry to those shown below:

```
tag1: 1 40.00n 40.91n n lvgs x4.x3.mn2 b[0] x4.bn 0.50 1.00
tag1: 2 53.00n 60.91n n lvgs x4.x3.mn2 b[0] x4.bn 0.70 1.00
```

mosv

A key word for MOSFET node voltages checking.

Examples

Example 40

```
.tcheck tag1 mosv model=nch lvgs=1 uvgs=5 lvgb=3 uvds=10
```

Checks all of the MOSFETs using the nch model. An error message is reported to one of the following files:

- hsim.mosv
- out_file.mosv

when any of the following conditions is met:

- vgs <1 V
- vgs > 5 V
- vgb < 3 V
- vds > 10 V

Example 41

```
.tcheck t1 mosv model=nch cond='(vgs < -3 || vgs > 7) || (vdb <
1 && vdb > -1)'
```

Checks all MOSFETs using the nch model. If vgs and vdb meet the user-defined conditional expression, then an error is reported to the hsim.t1 or out_file.t1.

Example 42

```
.tcheck tag mosv model=n minL=1.1e-6 maxL=5.e-6 lvgs=-1 uvgs=1
lvgd=-1 uvgd=2
```

This syntax checks every n-MOSFET to see if its length is within [1.1 microns, and 5.0 microns]. If the length is within these constraints, HSIM continues to check vgs and vgd to see if vgs <-1 or vgs > 1 or vgd <-1 or vgd > 2.

Example 43

```
.tcheck 7 mosv model=p cond='(l <=2e-6 && vds > 7) || (l > 2e-6
&& vds > 10)'
```

This syntax checks whether vds is greater than 7 V when MOSFET length is less than or equal to 2 microns, or if vds is greater than 10 V when MOSFET length is greater than 2 microns.

Example 44

```
.tcheck tag mosv model=p minL=0 maxL=2e-6 uvds=7
.tcheck tag mosv model=p minL=2e-6 maxL=3e-6 uvds=8
.tcheck tag mosv model=p minL=3e-6 uvds=10
```

The voltage difference depends on the transistor length. For example, if a transistor length is smaller than or equal to 2um, the voltage difference between drain and source can not be larger than 7V. Otherwise, a Warning is issued. If the length is between 2um and 3um, the max Vds is 8V. If the length is larger than 3um, the max Vds is 10V. minL and maxL specify the range of transistor length.

Voltage and transistor length constraints are specified in one conditional expression, as shown in the example below.

Example 45

```
.tcheck 7 mosv model=p cond='(l <=2e-6 && vds > 7) ||
(l > 2e-6 && l <=3e-6 && vds > 8) ||
(l > 3e-6 && vds > 10)'
```

The above expression requires that when the length is no larger than 2um, and VDS is greater than 7V, a Warning is issued. If the length is greater than 3um and VDS is larger than 10V, report a Warning.

subckt

Specifies the sub-circuit to be checked.

Example

```
.tcheck tag1 mosv subckt=inv lvgs=1
```

This command checks all the MOSFETs inside subckt inv. If there are two inv instances in the design, this check will examine every MOSFET inside those two inv instances. lvgs is defined as the lower bound of the voltage difference between a gate and a source.

mos

Specifies the instance to be checked.

Example

```
.tcheck tag1 mosv mos=x1.* lvgs=1
```

Checks all the MOSFETs inside instance x1.

model

Specifies the model to be checked.

Example

```
.tcheck tag1 mosv model=nch lvgs=1
```

Check all the MOSFETs using nch as a model.

```
.tcheck tag1 mosv mos=x1.* model=nch lvgs=1
```

Checks all the MOSFETs inside instance x1 using nch as a model.

```
.tcheck tag1 mosv subckt=inv model=nch lvgs=1
```

Checks all the MOSFETs inside subckt inv using nch as a model. Table 2 describes the checking conditions.

Table 4 Checking Condition Primitive Definitions

Primitive	Description
lvgd	When MOSFET vgd is less than the value specified in lvgd, report an error to the output file. vgd is the voltage value between gate and drain.
uvgd	When MOSFET vgd is greater than the value specified in uvgd, report an error to the output file. vgd is the voltage value between gate and drain.

Table 4 Checking Condition Primitive Definitions (Continued)

Primitive	Description
lvds	When MOSFET vds is less than the value specified in lvds, report an error to the output file. vds is the voltage value between drain and source.
uvds	When MOSFET vds is greater than the value specified in uvds, report an error to the output file. vds is the voltage value between drain and source.
lvdb	When MOSFET vdb is less than the value specified in lvdb, report an error to the output file. vdb is the voltage value between drain and bulk.
uvdb	When MOSFET vdb is greater than the value specified in uvdb, report an error to the output file. vdb is the voltage value between drain and bulk.
lvgs	When MOSFET vgs is less than the value specified in lvgs, report an error to the output file. vgs is the voltage value between gate and source.
uvgs	When MOSFET vgs is greater than the value specified in uvgs, report an error to the output file. vgs is the voltage value between gate and source.
lvgb	When MOSFET vgb is less than the value specified in lvgb, report an error to the output file. vgb is the voltage value between gate and bulk.
uvgb	When MOSFET is greater than the value specified in uvgb, report an error to the output file. vgb is the voltage value between gate and bulk.
lvsb	When MOSFET vsb is less than the value specified in lvsb, report an error to the output file. vsb is the voltage value between source and bulk.
uvsb	When MOSFET vsb is greater than the value specified in uvsb, report an error to the output file. vsb is the voltage value between source and bulk.
minL	Define the minimum MOSFET length that mosv will check. default value is 0.
maxL	Define the maximum MOSFET length that mosv will check. default value is infinite.

Table 4 Checking Condition Primitive Definitions (Continued)

Primitive	Description
cond	A conditional expression may be specified as the checking criterion, and this user-defined condition can not be used with all the lower and upper bounds described above. This means that if the cond option is used, then the lvgd, ... parameters can not appear in the same command. Another restriction is when cond is used, the error report will go to a separate file whose name is hsim.tag_name or out_file.tag_name ---> hsim.mosv_tag.chk or out_file.mosv_tag.chk. Conditional expressions accept numbers and the following operators: <, >, <=, >=, ==, ||, and &&. Conditional expressions also accept the following predefined variables vgs, vgd, vbs, vbd, vds, vgb, and I.

report

Specify how many times elements with errors will be reported for the associated tag. If not limited, CCK reports all violations by default.

Example 46

```
.tcheck tag1 mosv mos=x1.* lvgs=1 report=2
```

Checks all the MOSFETs inside x1. If vgs is < 1V an error is reported to either the hsim.mosv or out_file.mosv file. Only the first two errors are reported for tag1, even if there are additional errors.

time

Specify the time duration that the checking sustains before triggering an error report.

Example 47

```
.tcheck tag1 mosv mos=x1.* lvgs=1 time=5n
```

Checks all the MOSFETs inside x1. If vgs is < 1V for more than 5ns, an error is reported to either the hsim.mosv or out_file.mosv files.

parallel

Used to reduce the number of errors reported for the parallel devices.

Example 48

```
.tcheck tag1 mosv mos=x1.* lvgs=1 parallel=1
```

Checks all the MOSFETs inside x1. If vgs is < 1V, an error is reported to either the hsim.mosv or out_file.mosv file. Only one error is reported if there are parallel devices having the same error. An asterisk (*) is added after the element name to indicate there are parallel MOSFETs.

separate_file

When separate_file=1, errors are reported with a tag to a separate file.

Example 49

```
.tcheck tag2 mosv mos=x1.* lvgs=1 separate_file=1
```

Error of this check goes to hsim.tag2 or out_file.tag2.

Output Sorting

It may be determined whether and by what methodology to sort the output. Table on page 273, lists the sorting keys and their tag names:

Output Sorting Keys

Table 5

Parameter	**Description**
tag	**Tag Name**
t1	The time an error occurs
el_name	Element name
node1	First node name
node2	Second node name
err_v	Error voltage value

Example 50

```
.tcheck tag2 mosv mos=x1.* lvgs=1 sort=node1|tag|err_v
```

The error output will be sorted in the following order:

1. Node name
2. Tag name
3. Error voltage

start/stop

The time span allowed for HSIM to perform checking is specified.

```
.tcheck tag2 mosv mos=x1.* lvgs=1 start=10n stop=20n start=100n
```

In the above example, checking is accomplished between 10 ns and 20 ns. It is also repeated after 100 ns.

step

Controls which time step to check.

Note:

This command must be used with either Start or Stop

```
.tcheck tag2 mosv mos=x1.* lvgs=1 start=10n stop=120n step=30n
```

Checking starts at 10 ns and is repeated every 30 ns thereafter until 120 ns.

Note:

Time duration is not tracked when using Step if the condition is true. Only a particular time point is reported. Therefore, time parameter will be ignored.

Similar checking is applied to bipolar junction bjtv) devices as shown in the following example.

tcheck bjtv

BJT device voltage check.

Syntax

```
.tcheck tag_name bjtv <model=name> <subckt=subckt_name>
   <bjt=inst_name> <lvbe=val> <uvbe=val> <lvbc=val>
   <uvbc=val> <lvbs=val> <uvbs=val> <lvce=val>
   <uvce=val> <lvcs=val> <uvcs=val> <lves=val>
   <uves=val> <cond='expression'> <report=#>
```

```
<time=val> <parallel=0|1> <separate_file=1>
<sort=tag|t1|el_name|node1|node2|err_v>
<start=time> <stop=time> <step=time>
```

Parameters

The parameters for bjtv are similar to mosv except for the following:

- bjtv: The command keyword is bjtv, instead of mosv.
- bjt: The instance keyword is bjt, instead of mos.

Keywords for voltage-pair conditions are shown in Table on page 275.

Voltage-Pair Condition Keywords - Primitive Definitions

Table 6

Parameter	Description
lvbc	When bjt vbc is less than the value specified in lvbc, report an error to the output file. vbc is the voltage value between base and collector.
uvbc	When bjt vbc is greater than the value specified in uvbc, report an error to the output file. vbc is the voltage value between base and collector.
lvce	When bjt vce is less than the value specified in lvce, report an error to the output file. vce is the voltage value between collector and emitter.
uvce	When bjt vce is greater than the value specified in uvce, report an error to the output file. vce is the voltage value between collector and emitter.
lvcs	When bjt vcs is less than the value specified in lvcs, report an error to the output file. vcs is the voltage value between collector and substrate.
uvcs	When bjt vcs is greater than the value specified in uvcs, report an error to the output file. vcs is the voltage value between collector and substrate.
lvbe	When bjt vbe is less than the value specified in lvbe, report an error to the output file. vbe is the voltage value between base and emitter.
uvbe	When bjt vbe is greater than the value specified in uvbe, report an error to the output file. vbe is the voltage value between base and emitter.

Table 6

Parameter	Description
lvbs	When bjt vbs is less than the value specified in lvbs, report an error to the output file. vbs is the voltage value between base and substrate.
uvbs	When bjt vbs is greater than the value specified in uvbs, report an error to the output file. vbs is the voltage value between base and substrate.
lves	When bjt ves is less than the value specified in lves, report an error to the output file. ves is the voltage value between emitter and substrate.
uves	When bjt ves is greater than the value specified in uves, report an error to the output file. ves is the voltage value between emitter and substrate.

Similar checking is applied to diodev and capv as shown below:

tcheck diodev

Syntax

```
.tcheck tag_name diodev <model=name> <subckt=subckt_name>
   <diode=inst_name> <lvac=val> <uvac=val> <report=#>
   <time=val> <parallel=0|1> <separate_file=1>
   <sort=tag|t1|el_name|node1|node2|err_v> <start=time>
   <stop=time> <step=time>
```

Parameters

Diode device voltage check is very similar to mosv except for the following:

- diodev: The command keyword is diodev, instead of mosv.
- diode: The keyword for instance is diode, instead of mos.

Keyword for voltage-pair conditions are as follows:

lvac

When diode vac (voltage value between anode and cathode) is less than the value specified in lvac, report an error to the output file.

uvac

When diode vac (voltage value between anode and cathode) is greater than the value specified in uvac, report an error to the output file.

Note:

Conditional expressions are not supported in diodev.

tcheck capv

Capacitor device voltage check.

Syntax

```
.tcheck tag_name capv <subckt=subckt_name>
   <cap=inst_name> <lvpn=val> <uvpn=val>
   <report=#> <time=val> <separate_file=1>
   <sort=tag|t1|el_name|node1|node2|err_v>
   <start=time> <stop=time> <step=time>
```

Parameters

Capacitor device voltage check is very similar to mosv except for the following:

The following voltage-pair key words apply:

- capv: The command keyword is capv, instead of mosv.
- cap: The keyword for instance is cap, instead of mos.

lvpn

When cap, the absolute voltage value between positive and negative nodes is less than the value specified in lvpn, reports an error to the output file.

uvpn

When cap, the absolute voltage value between positive and negative nodes is greater than the value specified in uvpn, reports an error to the output file.

Note:

Conditional expressions are not supported in capv.

Post-Process Device Voltage Check

The post-process device voltage check is accomplished with two methods.

Method 1

The first method for post-process device voltage check uses the following steps:

1. Add a device voltage check command in any of input file using the following syntax:

   ```
   .param hsimDeviceV=cmdFile
   ```

2. Add the following syntax to the cmdFile:

   ```
   .tcheck post=1
   ```

   The following is a command file example for post-process voltage check:

   ```
   .tcheck post=1
   .tcheck jtl1 mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1
   .tcheck jtl2 mosv model=pch mos=xram* lvgs=-0.6 uvds=1
   .tcheck jtl3 mosv model=nch mos=* cond='(vds < -1.0 || vgs <
   -0.75)'
   ```

3. Run HSIM. An FSDB file will be created such as hsim.fsdb. The file contains all the print nodes and all the nodes required in comparing device voltage.
4. Run HSIM with a new option:

   ```
   -post_devv fsdbFile
   ```

   HSIM will read in netlist and device voltage commands. It then reads the nodes needed for device voltage comparison from the FSDB and outputs the results.

   Note:

   Partial results are written throughout the process.

Method 1 Example:

Run hsim -o pp. A pp.fsdb file is created.

Run hsim -o pp1 -post_devv pp.fsdb. Signals are read from the fadb file and compared.

Method 2

The second method for post-process device voltage check uses the following steps:

1. Add a device voltage check command in any of input file using the following syntax:

```
.param hsimDeviceV=cmdFile
```

2. Add the following syntax to the cmdFile:

   ```
   .tcheck post=2
   ```

   When post=2 appears, the voltage will continue to be read in the fsb file using the same process. The following is a command file example for post-process voltage check:

   ```
   .tcheck post=2
   .tcheck jtl1 mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1
   .tcheck jtl2 mosv model=pch mos=xram* lvgs=-0.6 uvds=1
   ```

3. Run HSIM – an FSBD or WDF waveform file like hsim.fsdb is created. When simulation is finished, it automatically uses the created waveform file to perform the post-process voltage check.

   Note:

   When running post=2, HSIM automatically performs the post-process device voltage check after the waveform file is created.

Signal Integrity Checks

These CircuitCheck commands are designed to help check for static and dynamic crosstalk and noise-sensitivity estimation. These commands are presented in alphabetic order to make it easier to find them.

Dynamic Crosstalk Analysis

Dynamic crosstalk refers to the parasitic effects which lead to distortion of digital signals in the post-layout netlist. The effect is measured according to user-specified thresholds for change of characteristics of the signals. Violations are reported in the CircuitCheck result file.

To illustrate the usage concept, consider a digital signal at a node of interest in Figure 27. There are two overlapped waveforms of the signal. The leading one is the waveform in the pre-layout netlist. The second is the waveform found in the post-layout netlist. It somewhat differs from the first due to parasitic effect. Refer to Figure 27 on page 280.

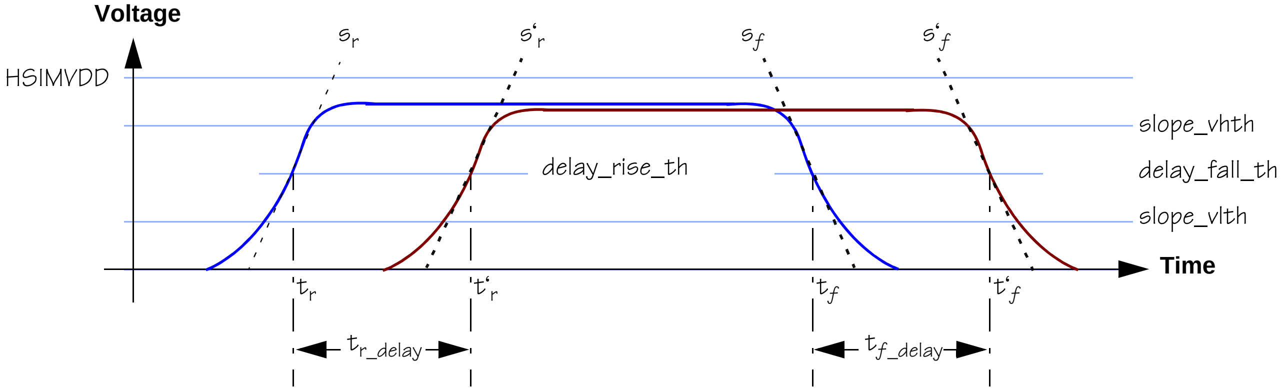


Figure 29 Pre-/Post-layout Waveform Node Differences

cckDXtalk

Syntax

Note:

The values or numbers specified in the usage syntax are the default values, if corresponding parameters are not specified.

```
cckDXtalk ccFile=<ccfile_name> noccFile=<noccfile_name>
   [nodeListFile=<nodelist_name>] [node=<node_name>]*
   [skipnode=<nodeName>]* [slope_rel=10] [slope_abs=1e-9]
   [slope_vhth=80] [slope_vlth=20] [delay_rise_th=50]
   [delay_rise_abs=1e-9] [delay_fall_th=50]
   [delay_fall_abs=1e-9] [separate_file=[0|1]]
   [extract_signals=[0|1]] [bc_wc=[0|1]]
```

Parameters

ccFile

Optional. ccFile specifies the post-layout fsdb file name. Refer to Usage Flow Methods on page 283.

noccFile

Optional. noccFile specifies the pre-layout fsdb file name. Refer to Usage Flow Methods on page 283.

nodeListFile

Optional.

spfFile

Optional. spfFile specifies the \SPF file name. Refer to Usage Flow Methods on page 283

node

Optional.

skipnode

Optional.

slope_rel

Relative slope in percentage. Unit is %. The default is 10.

slope_abs

Absolute slope. Unit is V/ns. The default is 1.

slope_vhth or slope_high

High threshold voltage of output. Unit is %.The default is 80, meaning that 80% of HSIMVDD.

slope_vlth or slope_low

Low threshold voltage of output. Unit is %. The default is 20, meaning that 20% of HSIMVDD.

delay_rise_th or delay_rise_thr

Rising threshold in percentage. Unit is %. The default is 50, which means 50% of HSIMVDD.

delay_rise_abs

Absolute value of rise time. Unit is second(s). The default is 1ps.

delay_fall_th or delay_fall_thr

Falling threshold in percentage. Unit is %. The default is 50, which means 50% of HSIMVDD.

delay_fall_abs

Absolute value of fall time. Unit is second(s). The default is 1ps.

separate_file

If set to 1, output to a separate file. The default is 0.

extract_signals

If set to 1, extract signals which violate cckDXtalk rules from ccFile and noccFile into a separate fsdb file. The default is 0.

bc_wc or wc_bc

If set to 1 best/worst violations of a signal in the simulations are reported. The default is 0.

Signal Edge Characteristics

The signal edge characteristics for the waveform shown in Figure 27 on page 280 are described below.

Thresholds

delay_rise_th and delay_fall_th

Thresholds for the rising and falling edges. Normally these two values are 50% of VDD. Their simulation time at the point where the signal crosses the thresholds are tr and tf.

slope_vhth and slope_vlth

Defines the signal edges. The default value of slope_vhth is 80% of VDD and 20% of VDD for slope_vlth.

Rising Slope

srising

The positive average slope of a signal crossing from slope_vlth to slope_vhth as shown by the following syntax: srising=(slope_ vhth-slope_ vlth)/(tr_ end-tr_ start).

Falling Slope

sfalling

the negative average slope of a signal crossing from slope_vhth to slope_vlth as shown by the following syntax:

```
sfalling=(slope_ vlth-slope_ vhth) (tf_ end-tf_ start)
```

Validation Validation is measured based on four criteria:

Rising Edge Delay

Rising edge delay measures the time delay of a rising edge. It measures the difference of tr, i.e., tr_delay between pre-layout and post-layout waveforms. If the difference, in second, is greater than the value specified by delay_rise_abs, a Warning is issued into the result file.

Falling Edge Delay

Falling edge delay measures the time delay of a falling edge. It measures the difference of tf, i.e., tf_delay between pre-layout and post-layout waveforms. If the difference, in second, is greater than the value specified by delay_fall_abs, a Warning is issued into the result file.

Slope Relative Error

Slope relative error measures the relative error of rising/falling slopes between pre-layout and post-layout waveforms. The error is calculated using the formula (slopepost_ layout-slopepre_ layout)*100%. If the absolute value of the error is greater than the value specified by slope_rel, a Warning is issued into the result file.

Slope Absolute Error

Slope absolute error measures the difference of rising/falling slopes between pre-layout and post-layout waveforms. If the difference, in V/ns, is greater than the value specified by slope_abs, a Warning is issued into the result file.

Usage Flow Methods

Various methods of executing dynamic crosstalk checking:

Method 1

The first method of executing dynamic crosstalk checking is to execute HSIM separately. Using this approach, simulation results must be prepared for both pre-layout and post-layout netlists in advance. The result files are denoted as noccFile.fsdb and ccFile.fsdb respectively. The nodes of interest have to be output using .print command for both cases.

The analysis needs an optional node list file such as nodelist.txt. The node list file is a text file containing a list of analysis nodes itemized one node name per line. If the file is not given, the node specifier in cckDXtalk will specify which nodes are to be analyzed.

Dynamic crosstalk analysis is eventually driven by adding cckDXtalk into the CircuitCheck command file of the netlist. This is either a pre- or post-layout netlist file and run HSIM with this revised netlist. Figure 28, shows the detailed analysis flow.

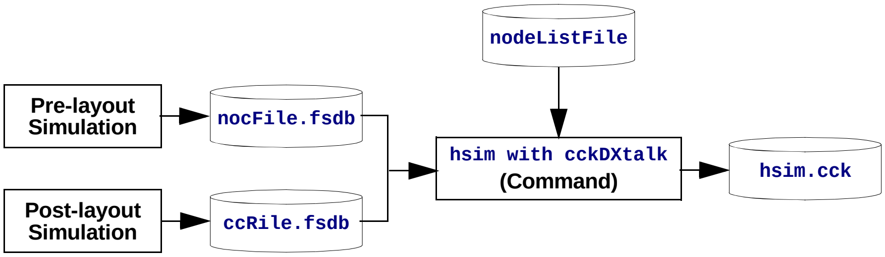


Figure 30 Dynamic Cross Talk Detailed Analysis Flow

Method 2

The second method of executing dynamic crosstalk checking is to run cckDXtalk in batch mode. Using this flow, only spfFile must be specified while not specifying ccFile and noccFile. The command will automatically execute HSIM twice.

- The first run simulates the pre-layout netlist.
- The second run simulates the post-layout netlist with the specified SPF file and carries out the analysis.

Nodes found that are to be analyzed are automatically added to output files as shown in the following examples:

Examples

When ccFile and noccFile are both given, add cckDXtalk to the CircuitCheck command file using the following syntax:

```
cckDXtalk ccFile=post.fsdb noccFile=pre.fsdb
nodeListFile=nodelist.txt\
delay_rise_th=45 delay_rise_abs=1.12n \
slope_vlth=14 slope_vhth=80 slope_abs=1 slope_rel=10 \
delay_fall_th=60 delay_fall_abs =0.001n \
separate_file=1
```

Following is typical report sample from cckDXtalk:

```
* --------------------------------------------------------
* Dynamic Cross Talk Check
* noccFile                                      = pre.fsdb
* ccFile                                        = post.fsdb
* nodeListFile                                  = nodelist.txt
* referenced vdd                                = 1.5 V
* slope_rel                                     = 10%
* slope_abs                                     = 1 V/ns
* slope_vlth                                    = 14%
* slope_vhth                                    = 80%
* delay_rise_th                                 = 55%
* delay_rise_abs                                = 1 ns
* delay_fall_th                                 = 60%
* delay_fall_abs                                = 0.001 ns
* separate_file                                 = 1
* --------------------------------------------------------
Signalsim time(ns) delay time(ns) slope rel(%) slope \
abs(v/ns)
v(oup) 0.375-       -28.3   3.69
v(oup) 3.96        0.232(F) -47.7 8.73
v(oup) 7.36     --34 4.49
v(oup)    11    0.232(F)-47.7 8.72
v(out_b) 0.28     - -34   4.49
v(out_b)    2.71    0.232(R) -47.7 8.72
.......
.......
     find 24 rise delay violation(s).
     find 89 fall delay violation(s).
     find 174 absolute slope violation(s).
     find 174 relative slope violation(s).
Total of 437 dynamic cross talk violation(s).
Total of 6 node checked.
```

Using cckDXtalk when the spfFile is given using the batch mode method. When spfFile is given, add cckDXtalk into the CircuitCheck command file using the following syntax:

```
cckDXtalk spfFile=cc.spf nodeListFile=nodelist.txt\
delay_rise_th=45 delay_rise_abs=1.12n \
slope_vlth=15 slope_vhth=75 slope_abs=4 slope_rel=19 \
delay_fall_th=35 delay_fall_abs =3.12n \
separate_file=1
```

cckParasiticRC

cckParasiticRC performs statistical analysis of the parasitic RC during netlist loading to aid in post layout circuit debugging. See cckParasiticRC on page 170 for detailed description.

Static Crosstalk Noise Analysis: Estimating Noise Glitches

Due to the ever increasing density in a chip design, the coupling capacitors between signal nets is an important topic to study. Aggressor nets can induce noise on victim nets. If the noise glitch is too large, it will trigger the circuit to change state unexpectedly. Furthermore, with the decrease of power supply voltage (VDD) such as from 3V to 2.5V to 1.8V, the noise glitch requires more attention and needs to be addressed.

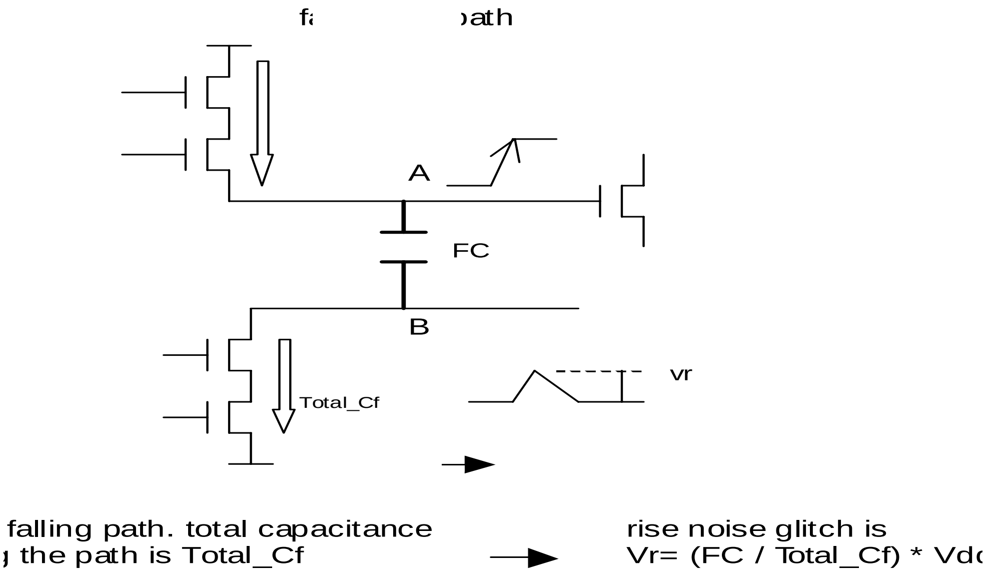


Figure 31 Fast Rising vs. Slow Falling Path]

If FC is a floating capacitor with a value exceeding a specific threshold; and from node A, the fast rising paths are located; from node B, slow falling paths are located. The fast rising at A will create a rising noise glitch at node B. The noise bump is approximated as (FC / Total_Cf) * VDD, where FC is the coupling capacitance; Total_Cf is the total capacitance on the slow falling path.

Similarly, fast falling paths and slow rising paths may exist at the two ends of a floating capacitor. The noise glitch on the slow path needs to be computed.

Note:

> When cckStaticXtalk is issued, CircuitCheck automatically disables HSIM transition simulation. This means that there is no transition simulation even if users have .tarn in the HSIM control file or netlist file.

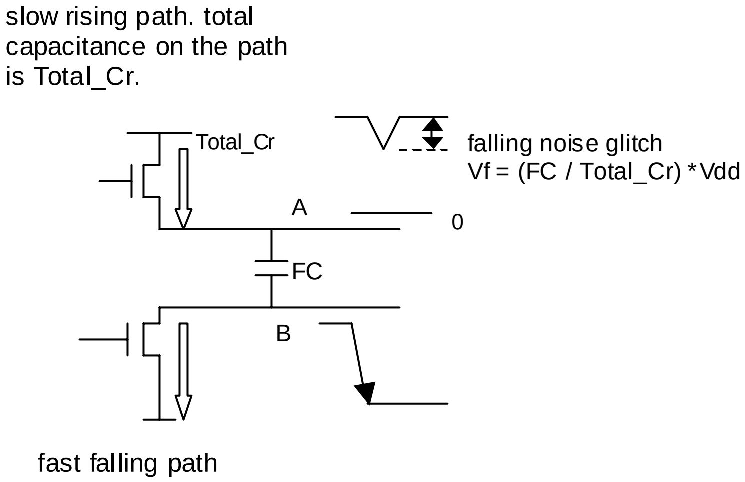

slow rising path. total capacitance on the path is Total_Cr.
Total_Cr
falling noise glitch
Vf = (FC / Total_Cr) *Vdd
A
0
FC
B
fast falling path

Figure 32 Fast Falling vs. Slow Rising Path

The following specifications are needed to analyze crosstalk.

Running Crosstalk Glitch Analysis

The cckStaticXtalk command runs crosstalk glitch analysis. It locates the significant floating capacitors and identifies the victim and potential aggressor signals, then estimates the static glitch noise caused by the aggressor signals through the coupling capacitors. Glitches over the user-specified threshold values are reported.

Syntax

```
cckStaticXtalk fCapTh=fcap_threshold
   vrth=rise_noise_glitch_threshold
   vfth=fall_noise_glitch_threshold
   [method=1|2] [modelWLRatio=model_name min
   max>][RCTime=riseMin riseMax fallMin
   fallMax][fCapRptRatio value] [rptAggrNode node_name]
```

```
[subckt=subcircuit_name] [node=node_name]
[skipSub=subcircuit_name] [skipNode=node=name]
[skpElem=element_name] [debug 0|1]
```

Parameters

fCapTh

Floating capacitors with values larger than the specified threshold value are considered for crosstalk analysis. The value is in farads.

vrth

Reports nodes if they have rise glitch larger than the specified threshold value. The value is in volts.

vfth

Reports nodes if they have fall glitch larger than the specified threshold value. The value is in volts.

method

A value of 1 (default) identifies the fast/slow path by the transistor W/L ratio. Specify a value of 2 to identify the fast/slow path by the path RC delay. The method values are mutually exclusive in the same static crosstalk command set.

modelWLRatio

This argument applies only to method 1. For the specified model name, the path is a fast path if the smallest W/L ratio is greater than the specified maximum value. The path is a slow path if the greatest W/L ratio is less than the specified minimum value.

RCTime

This argument applies only to method 2. The path RC delay calculation is based on the approach used by the cckRCDlyPath command. If the path delay is less than the specified minimum value, the path is treated as a fast path. If the path delay is greater than the specified maximum value, the path is treated as a slow path.

fCapRptRatio

This argument works only with the rptAggrNote argument. If specified, the cckStaticXtalk command uses the fCapRpt ratio value (0<=value<=1) to launch a quick check of coupling capacitors only. The other end of coupling capacitors is reported when its (CC/C_total) is greater that thefCapRpt ratio value.

If you do not use this argument, the coupling capacitors check does not run. There is no default.

rptAggrNode

This argument works only with the fCapRptRatio argument. It reports the qualified aggressor signals across the related coupling capacitors through the specified nodes. You can use a wildcard character in the node name.

subckt

Specifies to conduct a static crosstalk analysis within the specified subcircuit domain.

node

Specifies a static crosstalk analysis on the specified nodes. You can use this argument with the subckt argument.

skipSub

Runs static crosstalk analysis without considering the nodes in the specified subcircuit domain.

skipNode

Runs static crosstalk analysis without considering the specified nodes.

skipElem

Runs static crosstalk analysis without considering the specified elements.

debug

Specify a value of 1 to output all valid paths to a gziped file with a `.xtkdebug` extension. The default is 0 (off).

Examples

```
cckStaticXtalk fCapTh=1p vrth=0.01 vfth=0.02 method=1
modelWLRatio=(nx, 12, 18) modelWLRatio=(px, 16, 24)
```

In this example the transistor W/L ratio determines the fast/slow paths in the design. The analysis only considers the capacitors larger than 1pF. A path with an nx type transistor with a minimum W/L ratio greater than 18 is most likely a fast path. A path with a px type transistor with a minimum W/L ratio greater than 24 is most likely a fast path. The cckStaticXtalk command collects and reports the violations with rise glitch greater that 0.01v and fall glitch greater that 0.02v.

```
ckStaticXtalk fCapTh=1p vrth=0.01 vfth=0.01 method=2 RCTime=(0.5n
3n 0.4n 4n)
```

In this example the path RC delay (RC time constant) determines the fast/slow paths in the design. The analysis only considers the capacitors larger than 1 pF. The cckStaticXtalk command collects and reports the violations with rise glitch greater that 0.01v and fall glitch greater that 0.02v. A rising path with a delay less than 0.5n is considered a fast path. If the delay is larger than 3n, it is considered a slow path. A falling path with a delay less that 0.4n is considered a fast path. If the delay is larger than 4n, it is considered a slow path.

The following three files are created:

- hsim.cckxtk or out_file.cckxtk: These files show path-pairs on the interested floating capacitors. In each pair, one path is fast and the other is slow.
- hsim.cckvr or out_file.cckvr: Path-pairs that cause a large rising noise glitch will be reported in these files.
- hsim.cckvf or out_file.cckvf: These files contain path-pairs that cause large falling glitch, hence smaller vf.

Note:

The cckStaticXtalk command replaces the cckStaticXtalk_GroupCmd command. Table 7 shows how the cckStaticXtalk command arguments map to the cckStaticXtalk_GroupCmp commands.

Table 7

cckStaticXtalk argument	Equivalent cckStaticXtalk_GroupCmd command
fCapTh	cckXtalkFloatingCap
vrth	cckXtalkRiseVolt
vfth	cckXtalkFallVolt
method	cckXtalkByWL and cckXtalkByRC
modeWLRatio	cckXtalkModelWLratio
RCTime	cckXtalkRiseTimeConst and cckXtalkFallTimeConst
fCapRptRatio	cckXtalkReportCouplingCapRatio
rptAggrNode	cckXtalkReportAggressorNode
subckt	cckXtalkAtNode

Table 7

cckStaticXtalk argument	Equivalent cckStaticXtalk_GroupCmd command
node	cckXtalkAtNode
skipSub	cckXtalkSkipNode
skipElem	cckXtalkSkipElem

hsim.cckxtk Output Sample

The hsim.cckxtk output sample contains pairs of slow and fast paths as shown in the following:

```
; Report format:
;  Serial_num Capacitor_name Cap_1st_node_name Cap_2nd_node_name
Cap_value \
;   minWL_mos_name_on_low_imp_path
minWL_mos_name_on_high_imp_path
;     Low_imp_path:
;       Vsrc_node_name
;       Elem_name Elem_type L/W or Resistance
;     High_imp_path:
;       Vsrc_node_name
;       Elem_name Elem_type L/W or Resistance
Total number of low and high impedance path pairs for
floating capacitors=2912
User defined values:
Floating capacitance threshold=1e-012(F)
Use MOSFET WL ratio to find low/high impedance paths.
nx   min=12, max=18
px   min=16, max=24
px2  min=10, max=20
1     c6x gx1 wl8 4.51e-012 mx7|mn1 mx55|mi30
   low_imp_path:
          gnd
          mx7|mn1nmos0.4/5
        * mx7|xi38|mn1@3 nmos0.4/5
        * mx7|xi38|mn1@2 nmos0.4/5
        * mx7|xi38|mn1@5 nmos0.4/5
        * mx7|xi38|mn1@4 nmos0.4/5
   high_imp_path:
          gnd
          mx55|mi30 nmos 0.4/3
```

Note:

> MOSFET mx7|mn1 has parallel transistors. The sum of their W/L ratios is documented. Hence, the effective W/L is (5 * 5/0.4)=62.5 > (max=18). It is a fast path.

Floating capacitor c6x has two end nodes: gx1 and wl8. A fast falling path, whose MOSFET with smallest W/L is mx7|mn1, consists of parallel transistors. For the high impedance path, its smallest MOSFET is mx55|mi30, whose W/L is 3/0.4 < (min=12).

hsim.cckvr Output Sample

The hsim.cckvr output sample is shown in the following:

```
Total number of Vr errors=2
User defined values:
Floating capacitance threshold:1e-012(F)
Vr threshold:0.01(V)
Use mos WL ratio to find low/high impedance paths.
nx:  min=12, max=18
px:  min=16, max=24
px2: min=10, max=20
S.N. Cap_name  Node1 Node2  Ctotal(F) Cfloat(F) Vr(V) Low_imp_mos
High_imp_mos
1 c101 x5|io6   x4|i6b 3.7e-012 1.3e-012 0.35 mx4|mi79 mx9|mi54
 low_imp_path:
        gnd
        mx4| mi79 nmos 0.4/25
 high_imp_path:
        vdd
        mx9|mi54 pmos 0.5/5
```

Note:

> The floating capacitor is c101. It has two nodes: x5|io6, x4|i6b. The total capacitance on the slow path is 3.7 pF (including the floating capacitance=1.3 pf). The rising noise glitch is 0.35V. The smallest MOSFET on the low impedance path is mx4|mi79. The smallest MOSFET on the high impedance path is mx9|mi54.

hsim.cckvf Output Sample

The hsim.cckvf output sample is shown in the following:

```
Total number of Vf errors=115
User defined values:
Floating capacitance threshold:1e-012(F)
Vf threshold:0.3(V)
Use mos WL ratio to find low/high impedance paths.
nx:  min=12, max=18
px:  min=16, max=24
px2: min=10, max=20
S.N. Cap_name Node1 Node2 Ctotal(F) Cfloat(F) Vf(V) Low_imp_mos
High_imp_mos
1 c201 xi5|io4 xi4|io4b 6.7e-012 2.3e-012 0.66 mx54|mi79 mx59|mi54
     low_imp_path:
          gnd
          mx54| mi79 nmos 0.4/25
     high_imp_path:
          vdd
          mx59|mi54 pmos 0.5/5
```

Note:

> The floating capacitor is c201. The total capacitance on the slow path is 6.7 pF. The falling glitch will result in vf=0.66V. The smallest MOSFET on the low impedance path is mx54|mi79. The smallest MOSFET on the high impedance path is mx59|mi54.

Example 51

```
cckXtalkFloatingCap     1.e-12
cckXtalkRiseVolt        0.01
cckXtalkFallVolt        0.3
cckXtalkByRC            1
cckXtalkRiseTimeConst   min=0.5n max=3n
cckXtalkFallTimeConst   min=0.4n max=4n
```

Use RC-delay to decide the fast and slow paths. The floating capacitors threshold is 1 pF. Any coupling capacitors value exceeding 1 pF will be considered. From those capacitors' two ends, trace to VDD and GND to find rising and falling paths. For a rising path, if its rc-delay is smaller than 0.5 ns, it is a fast path. If its delay is larger than 3 ns, it is a slow path. Similarly for the falling paths. Fast and slow path-pairs are located at each interested floating capacitor. Then, compute the coupling impact. If the rising noise glitch is larger than 0.01V, this path-pair will be reported in hsim.cckvr. If the falling glitch is large enough to make Vf smaller than 1.79V, then this path is in hsim.cckvf.

cckxtk Output Sample

The hsim.cckxtk output sample is shown in the following:

```
;    Report format:
;         Serial_num Capacitor_name Cap_1st_node_name
Cap_2nd_node_name Cap_value \
;    Time_constant_on_low_imp_path
Time_constant_on_high_imp_path
;    Low_imp_path:
;         Vsrc_node_name
;         Elem_name Elem_type L/W or Resistance
;    High_imp_path:
;         Vsrc_node_name
;         Elem_name Elem_type L/W or Resistance
Total number of low and high impedance path pairs for floating
capacitors=2647
User defined values:
Floating capacitance threshold1e-012(F)
Use RC delay to find low/high impedance paths.
Rise time-constant:min=5e-010, max=3e-009
Fall time-constant:min=4e-010, max=4e-009
S.N. Cap_name  Node1   Node2    Cap_value(F)  Min_time_constant
Max_time_constant
1 c249 xite<0 rd<9 7.6e-012 3.9933e-010             1.43892e-008
low_imp_path:
     gnd
          mxer|xi8|mn1 nmos 0.4/10
high_imp_path:
     vdd
          mxer|xi9|mi54 pmos 0.5/20
          mxer|xi9|mi56 pmos 0.5/20
```

hsim.cckvr Output Sample

The hsim.cckvr output sample is shown in the following:

```
Total number of Vr errors=4
User defined values:
Floating capacitance threshold:1e-014(F)
Vr threshold:0.01(V)
Use RC delay to find low/high impedance paths.
Rise time-constant: min=5e-010, max=3e-009
Fall time-constant: min=4e-010, max=4e-009S.N. Cap_name Node1
Node2 Ctotal(F) Cfloat(F) Vr(V) Min_time_constant
Max_time_constant
1 c81 xik2|i5 xik4|io2b 5.87e-012 2.4e-012 0.2.42 3.68e-010 5.0e-
009
     low_imp_path:
          gnd
          xi43|mi9 nmos 0.4/25
     high_imp_path:
          vdd
          xi13|mi5 pmos 0.5/5
```

hsim.cckvf Output Sample

The hsim.cckvf output sample is shown in the following:

```
Total number of Vf errors=1501 (1501)
User defined values:
Floating capacitance threshold:1e-012(F)
Vf threshold:0.2(V)
Use RC delay to find low/high impedance paths.
Rise time-constant: min=5e-010, max=3e-009
Fall time-constant: min=4e-010, max=4e-009
S.N. Cap_name Node1 Node2 Ctotal(F) Cfloat(F) Vf(V)
Min_time_constant Max_time_constant
1 c1081 xik1|io2 xik1|io4b 5.487e-012 2.04e-012 0.73042 3.68e-
010 5.0e-009
     low_imp_path:
          gnd
          xi313|mi79                           nmos0.4/25
     high_imp_path:
          vdd
          xi313|mi54                           pmos0.5/5
```

Leakage Current Detection

These CircuitCheck commands are designed to detect leakage-current paths, and to estimate the contribution of the nodes to the leakage current.

Detect Leakage Paths Between Voltage Supply Nodes

cckMaxStaticLeak

Syntax

```
cckMaxStaticLeak <num=n>
```

Description

CircuitCheck reports any path leaking current from one voltage source to another source statically. For example, there is a conducting path in Figure 31 on page 297 from VDD to GND.

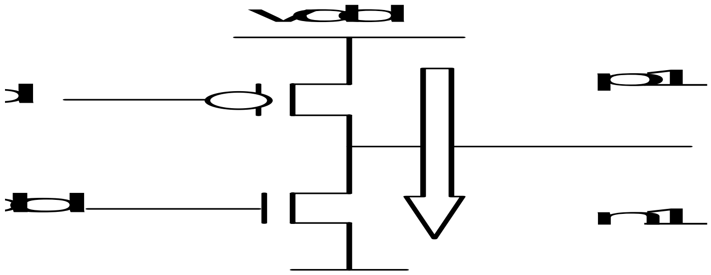


Figure 33 Conducting Path

CircuitCheck limits the number of Warnings by num=n. This type of Warning will be in .cckleak file.

Examples

```
cckMaxStaticLeaknum=10
```

The following is the Static leakage path between voltage sources output sample resulting from the command example above:

```
**************************************
* Static leakage path between voltage sources
**************************************
A static leakage path to
   node (vdd, v=1.65)
   from node (rmaddr<63>) thru (xm.xyy2.mp)
   from node (0, v=0) thru (xm.rz)
```

Leakage Current Detection in Non-Conducting Transistors

This function detects the OFF transistors at the standby time and computes the leakage current. Since the leakage current of an OFF device depends on its length, width and state of its drain and source nodes. For example, if a node is in high impedance mode; each group will be detailed in a summary report.

cckOffLeakI

Syntax

```
cckOffLeakI <tag=tagName> <subckt=subName> <inst=instName>
   <vsrc=vsrcName> <vt=threshold> <time=standByTime>
   <ioff=ioffCmd> <hzStage=hzLimit> <detail=[0|1]>
   <separate_file=[0|1]>
```

Parameters

tag

A label in the report.

subckt

Examines the instances in this subckt.

inst

Instance name(s) to be checked.

vsrc

Voltage source name from which OFF devices are traced.

vt

Voltage threshold used to decide if a device has a leakage current. The default is 0.1V.

time

Standby time when checking is performed.

ioff

File name that contains the current ratios per unit length for each model.

hzStage

A series of OFF devices will make some nodes to be in high-impedance mode. In that case, each OFF device will have a high-z stage number, based on the distance from the non-high-z nodes. The default is 2.

detail

If set to 1, all the OFF transistors and the high-z stage for each instance will be printed. Otherwise, it is 0.

separate_file

If set to 1, the summary will be printed to a separate file.

Description

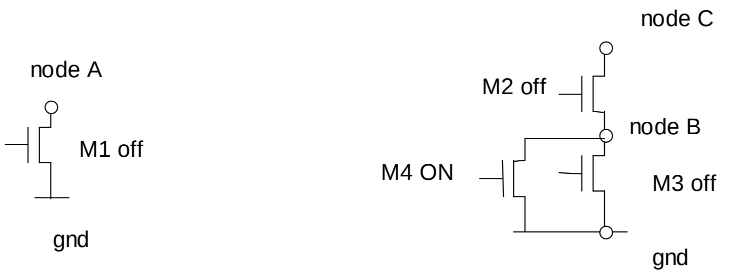
The cckOffLeakI command is added to a CircuitCheck command file and invoked through:

```
.param hsimCktCheck=cmdFile
```

Leakage Current in OFF Transistor

The leakage current of an OFF device depends on the voltage of its drain (D) and source (S) node. If a trace is run from a given voltage source node to an OFF device and reach the S node first, then if V(D) > V(S) +vt, this OFF device has a leakage current.

Checking is performed at time 10n, 20n, 41n and 66n and traced from node 0. Multiple commands are allowed at different times and from different voltage sources.



if node A's voltage
Va > Vgnd + vt, M1 is
considered to have leakage
current.

M2 and M3 are off. Since M4 is ON, node B is not a high-impedance node. If the voltage of node B is greather than (gnd + vt), M3 has a leak. For M2, we compare the voltage difference between nodes C and B. if it is larger than vt, then M2 has a leakage.

Figure 34 OFF Device Leakage Current

Tracing starts from the given voltage source specified in the command. For example, Figure 32 on page 299, a trace to node A is begun from a GND node with zero voltage, and where M1 is OFF. Then the voltage at A is compared. If Va > gnd +vt, M1 has a leakage current.

From the GND node, OFF device M3 can be traced continuously until it reaches M2. To consider the leakage current in M3, the voltage at node B and GND are compared. If Vb > gnd+vt, M3 has a leakage. At this point, Vb=0 and Vgnd=0 so M3 has no leakage. For M2, which is also OFF, the voltages of nodes C and B are compared. If Vc > Vb + vt, M2 has a leakage.

If some nodes are in a high-impedance (high-Z) state, the effects on adjacent nodes must be considered. Since the voltage of a high-z node is unknown, tracing must be continued to find the node voltage of the next stage. Before proceeding, however, the high-z stage must be defined.

High-Z Stage Definition A High-z stage is the distance from an OFF device with high-z node to an non-high-z node. Figure 33 on page 300 shows an example of a definition of high-z stage.

In Figure 33, tracing starts at a given voltage source such as GND. For one branch, OFF devices such as M3, M2 and M1 are tested and stop at node A, which is VDD or non-high-z node. Another tracing starts from GND where two branches of OFF devices are found. One branch is M8 -> M7 -> M6 and the other is M8 -> M7 -> M4 -> M5. Nodes D and E are either power supply or non-high-z nodes.

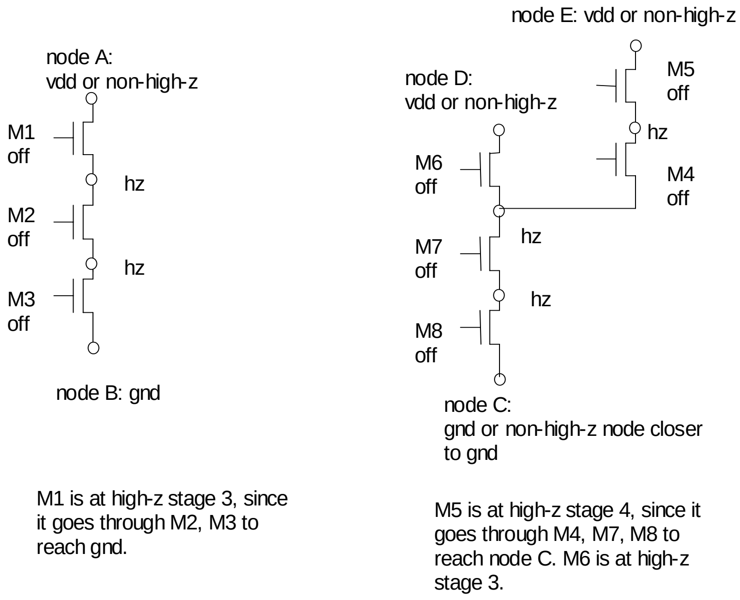

node E: vdd or non-high-z
node A:
vdd or non-high-z
M1
off
hz
M2
off
hz
M3
off
node B: gnd
node D:
vdd or non-high-z
M5
off
hz
M4
off
M6
off
M7
off
hz
hz
M8
off
node C:
gnd or non-high-z node closer
to gnd
M1 is at high-z stage 3, since
it goes through M2, M3 to
reach gnd.
M5 is at high-z stage 4, since it
goes through M4, M7, M8 to
reach node C. M6 is at high-z
stage 3.

Figure 35 OFF Device Leakage Current with a High Impedance Node

When deciding the leakage current, only the top device in each branch is considered. In Figure 33 on page 300, only M1, M6 and M5 are considered since they are the last element in each branch starting from GND node.

Figure 33, illustrates the following:

- M1 is at high-z stage 3
- M6 is at high-z stage 3
- M5 is at stage 4.

The leakage current at a large high-z stage is smaller than that at the small high-z stage as illustrated by the measurement. Therefore, the current ratios at different high-z stages need to be specified. Refer to Leakage Current Ratio on page 301 and Example 51 on page 302.

Leakage Current Ratio

The ratio in the current IoffCmd file is the leakage current per unit width for each model and length.

For example, in the following command:

```
nch 0.18u 2e-4 z2 1.5e-4 z3 1e-4
```

`nch` is the model name. `0.18u` is the length of the transistor. The leakage current per unit width is `2e-4` (Amp/meter) for an OFF device with high-z stage 0 or 1. If an OFF device is at high-z stage 2, the unit width leakage current is `1.5e-4` (Amp/meter). If a device is at high-z stage 3, then its leakage current is `1e-4` (Amp/meter). The higher the high-z stage, the less its leakage current.

- Determine which OFF devices in Figure 33 on page 300 have leaks:
  - In Figure 33, for M1, nodes A and B are considered. The nodes between A and B are not considered since they are at high-z mode. Tracing begins from node B (i.e., node B is closer to GND node). If Va > Vb + Vt, where Vt is the threshold specified in the command, M1 has a leak. If M1 is a nch device of L=0.18u, its leakage current is the current ratio of z3 (1e-4 Amp/meter) multiplied by the width of M1.
  - For M6, nodes D and C are considered. If Vd > Vc +vt, M6 has a leakage current. Otherwise, it does not have a leak. If M6 is a nch device with L=0.25u, then its leakage current is the current ratio z3 (0.5e-4 Amp/meter) multiplied by the width of M6.
  - For M5 at high-z stage 4, nodes E and C are considered. If Ve > Vc + vt, M5 has a leak. If only the OFF device with maximum high-z stage 3 is to be considered, M5 will not be considered. Now for the remaining OFF devices, for example, M4, since it is not the last element in the branch, it is considered to have no leakage.
- The total leakage current is I (leak in M1) + I (leak in M6).
- If the current ratio is not given, the total width of the OFF devices with the same model and length is added together.
- The length in the current ratio file may have a range. For example 0.18u to 4u means the devices with length from 0.18u to 4u, inclusively.

Add the cckOffLeakI syntax in Example 51 to the CircuitCheck command file:

Example 52

```
cckOffLeakI tag=t1 inst=* vsrc=0 vt=0.4 time=10n time=20n \
time=41n time=66n detail=1 separate_file=1 ioff=ioff.cmd \
hzStage=3
cckOffLeakI tag=t2 inst=* vsrc=vbb vt=0.4 time=6n time=8n \
time=15n time=56n detail=1 separate_file=1 ioff=ioff.cmd \
hzStage=3
```

Add the ioff.cmd syntax to the current ratio file as shown below:

```
/* leakage current for every size L
 * syntax:
 *
 * model_name L(meter) off_current(Amp/meter) highZCnt
 * off_current
 *
 * length unit=meter
 * current unit=Amp
 *
 * unit must be specified. for instance,
 * L=2u (2e-6 memter)
 * the unit for off_current ratio is Amp/meter
 * if length has range, use “to”
 */
     nch 0.18u 2e-4 z2 1.5e-4 z3 1e-4
     nch 0.19u to 4u 1e-4 z2 0.8e-4 z3 0.5e-4
     pch 0.18u 1.5e-4 z2 1.1e-4 z3 0.7e-4
```

Command Output

Producing the command output is accomplished as follows:

1. The options specified are listed in the command as follows:

```
****************************************************
* tag=t1 vt=0.400 hzStage=3 separate_file=1 detail=1
* vsrc=0 time=10n time=20n time=41n time=66n
* inst=*
* ioff=ioff.cmd
* leakage current spec is:
*         model length offI(A/meter) highZ_stage offI(A/m)
*         pch 0.18u       0.00015 z2 0.00011 z3 7e-005
*         nch 0.19u to 4u 0.0001 z2 8e-005 z3 5e-005
*         nch 0.18u       0.0002 z2 0.00015 z3 0.0001
****************************************************
```

2. All the OFF devices of model nch with length 0.18u are detected. These nch devices will be further separated into different high-z stages.

   - For devices at high-z stage 0 or 1, its unit width leakage current is 2e-4 Amp/meter. Therefore the total width of nch devices of L=0.18u are computed.
   - For nch device at high-z stage 2, the unit width ratio 1.5e-4 is used.
   - For nch device at high-z stage 3, the unit width ratio 1e-4 is used.

- For nch device at high-z stage 4 and above, their leakage current are not computed since they are too small.
- Similarly, a nch device with L=0.19u to L=4u is considered. They will use different current ratio.
- For a pch device, there are different ratios.
- When an OFF device is found and its current ratio is not specified in the current spec file, only the total width for each given length is computed.

A sample output summary is shown in Example 52:

Example 53 :

```
Summary @ time 10n from Vsrc (0) tag (t1):
     model length totWidth OFF_I(Amp)
pch L=0.48u
     W=2u (pch device with L=0.48u is not specified in the spec
     file. we simply compute the total width.)
     pch L=0.18u
                W=4u Ioff=6e-010(@0.00015)
                W=301.7u Ioff=3.3187e-008 (@0.00011) HiZ=2
     nch L=0.19u to 4u
                W=0u Ioff=0 (@0.0001)
                W=0u Ioff=0 (@8e-005) HiZ=2
                W=0u Ioff=0 (@5e-005) HiZ=3
     nch L=0.18u
                W=183.2u Ioff=3.664e-008 (@0.0002)
                W=130u Ioff=1.95e-008 (@0.00015) HiZ=2
                W=0u Ioff=0 (@0.0001) HiZ=3
```

In the detailed mode, every OFF device is printed and the high-z stage is indicated as shown in Example 53.

Example 54 :

```
xram.xr2.xr0.xu8.mn : nch L=1.8e-007 W=8e-007
xram.xr2.xr18.xu10.xu8.mn: nch L=1.8e-007 W=8e-007
xram.xr2.xr6.xu10.xu8.mn: nch L=1.8e-007 W=8e-007
xcam.xh_30.xi7.mn   : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_22.xi11.mn  : nch L=1.8e-007 W=1e-006
xcam.xh_54.xi7.mn   : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_31.xi7.mn   : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_54.xi9.mn   : nch L=1.8e-007 W=1e-006
xcam.xh_59.xi9.mn   : nch L=1.8e-007 W=1e-006
xcam.xh_16.xi9.mn   : nch L=1.8e-007 W=1e-006
```

Power-Down Floating-Gate Checking

Floating gates are detected by extending the existing Hi-Z (high-impedance) check to detect the leakage path during the power down mode. Hi-Z node voltage may be about one-half of the VDD voltage causing its fanout elements to partially conduct leakage current.

This check detects Hi-Z nodes, and marks their associated fanout transistors to be ON. If the operation forms any conducting path with device(s) driven by Hi-Z node(s), or a DC path with the leakage current larger than the threshold value, then these paths will be reported. The list can be checked to isolate the problematic Hi-Z nodes, so they can be clamped to either VDD or GND; which should avoid the leakage problem.

cckAnalogPDown

Syntax

```
cckAnalogPDown <tag=name> <time=check_time_1
   <time=check_time_2 ..>>> <vsrc=source
   destination_vsrc_name 1 <vsrc=source
   destination_vsrc_name 2 … >>>>
```

Parameters

time

The time interval when cckAnalogPDown is being performed. Multiple check times can be specified.

vsrc

vsrc name(s) that form the potential leakage path. cckAnalogPDown only looks for the conducting path(s) associated with the specified vsrc names.

cckAnalogPDownIth

Syntax Definitions

```
cckAnalogPDownIth <current_threshold_value>
```

Parameters

current_threshold_value

If the leakage current on the dcpath is detected by cckAnalogPDown is larger than this specified threshold value, the path will be listed in the cckAnalogPDown report.

Example

```
cckAnalogPDown tag=chk_1 time=5n time=7n vsrc=vdd vsrc=gnd
```

cckAnalogPDown conducts a power-down leakage-path check at both 5.0 ns and 7.0 ns. It then reports DC paths with leakage currents larger than the default 1uA current threshold flowing from VDD to GND, or the conducting paths with device(s) driven by a Hi-Z node. See the following example.

```
cckAnalogPDown tag=chk_2 time=5n time=7n vsrc=vdd vsrc=gnd
cckAnalogPDownIth 10u
```

Using the example above, except changing the default 1uA to 10uA, the report will appear as follows:

```
* ------------------------------------------------------
* DC Path Check (analog power-down check)
* pdownIth = 10e-006 Amp
* vsrc = vdd
* vsrc = gnd
* title = chk_2
* at = 0.000000005
* at = 0.000000007
* sort = 0
* separate_file = 1
* ------------------------------------------------------
path 1 from vdd to 0 @ 0.000000005s
     (gateIsHiZ)         xls.x1i146.mp1 (PMOS: Ids=0)
          drain          xlsda1.n1n152 5.25
          source         vdd 5.25
          gate           xlsda1.rstd -0.0155812
     (gateIsHiZ)         xlsda1.x1i146.mn1
     (NMOS: Ids=1.50143e-012)
          drain          xlsda1.n1n152 5.25
          source         0 0
          gate           xlsda1.rstd -0.0155812
```

Note:

> If the value for HSIMSTEADYCURRENT is greater than the value for cckAnalogPDownIth incorrect results may occur. Ensure that HSIMSTEADYCURRENT is less than cckAnalogPDownIth

cckElemI

cckElemI is used in transient simulation to monitor the current through each element. If the absolute value of the current exceeds the threshold ith, CircuitCheck reports the element name, current, and time. See cckElemI on page 191 for detailed description.

cckExiPath

cckExiPath is used to detect excessive current paths from a power supply to ground. See cckExiPath on page 192 for detailed description.

Static Analysis

The concept of static analysis is the ability to propagate characteristics of the design, in this case static logic 0 and static logic 1, throughout the whole design without the need of vectors or transient simulation. To propagate static logic 0 and static logic 1 throughout the design, it is necessary to define rules associated with the propagation. These propagation rules are used to determine if a static logic state (0 or 1) can be seen by a node through a network of design elements like MOSFETs, resistors, capacitors, and diodes.

HSIM^{plus}, with the Circuit Check Option, can analyze the design's state to identify static high impedance (HiZ) nodes and static DC paths, with a minimum of a design netlist and supply voltages defined.

Static 0 and Static 1 Notes

A node is considered static 0 if it has a static conducting path to a qualified GND node and no possibility of a static conducting path to a qualified VDD node, see below for definitions of qualified GND and VDD nodes relative to this static 0 and static 1 node concept. Similarly a node is considered static 1 if it has a static conducting path to a qualified VDD node and no possibility of a static conducting path to qualified GND node.

A qualified VDD node is defined as a node directly connected to a constant voltage source with a value greater than 0 volts.

A qualified GND node is defined as a node directly connected to a constant voltage source with a value equal to 0 volts.

The definition of a static conducting path is a series path consisting of resistors, inductors, 'on' n-MOSFET or p-MOSFET transistor drain/source channels , or a combination of these.

Static HiZ node and static DC path analysis requires MOSFETs to be either considered conducting (On), non-conducting (Off), or 'unknown'. The definitions/analysis used to determine the state of a MOSFET are described below. To help facilitate more representative results the concept of MOSFET threshold is necessary. MOSFET threshold is defined using the command parameters nmosOn and pmosOn, for n-MOSFET and p-MOSFET transistors, respectively. For example:

Example 55 Mosfet Conducting (On)/Non-Conducting (Off) Rule

```
nmosOn and pmosOn :user defined command parameters used in the
On/Off mosfet analysis with units of volts
     (example: nmosOn=0.0 pmosOn=3.0).

If (nmos gate node is static 1 && its' gate voltage > 'nmosOn')
     the mosfet is considered 'on'
else if nmos gate node is static 0 the mosfet is considered 'off'
else the mosfet is considered 'unknown'

If (pmos gate node is static 0 && its' gate voltage < 'pmosOn')
     the mosfet is considered 'on'.
else if pmos gate node is static 1 the mosfet is considered 'off'.
else the mosfet is considered 'unknown'
```

By performing the static logic state propagation it is possible to check for static HiZ nodes and static DC paths in order to improve design functionality and to improve power consumption. Because the propagation is performed statically, the coverage is greater than what can be achieved via dynamic simulation, ensuring more design errors are reported and ultimately more corrections can be implemented to ensure design correctness.

Static High Impedance Node

A node is said to be a Static High Impedance Node if it has no HiZ qualified static conducting path to either a qualified VDD or GND node.

A HiZ qualified static conducting path is defined as a series path consisting of resistors, inductors, ‘on’ or ‘unknown’ n-MOSFET transistor drain/source channels , ‘on’ or ‘unknown’ p-MOSFET transistor drain/source channels, or a combination of these.

cckStaticHZNode

To execute the CircuitCheck’s ‘Static High Impedance Node Check’ specify the parameter cckstaticHZNode in the circuit check command file.

Syntax

```
cckstaticHZNode <pmosOn=val0> <nmosOn=val1>
   <fanout=0|1|2|3> <pCap=0|1> <separate_file=0|1>
```

Parameters

pmosOn=<val0> (required)

pmosOn defines the voltage threshold <val0> used to determine 'on' p-MOSFET transistors. See Example 54 on page 308.

nmosOn=<val1> (required)

nmosOn defines the voltage threshold <val1> used to determine 'on' n-MOSFET transistors. See Example 54 on page 308.

fanout=[0|1|2|3]

fanout defines the nodes to report. The possible values are:

- 0 - reports all qualified HiZ nodes.
- 1 - reports only nodes connected to MOSFET gates.
- 2 - reports only nodes connected to MOSFET bulks
- 3 - reports only nodes connected to either MOSFET gates or bipolar transistor (BJT) bases. This is the default value.
- 4 - reports only nodes connected to either MOSFET drain/source or BJT emitter/collector

pCap=[0|1]

pCap defines the nodes to report when pseudo capacitors are encountered. A pseudo capacitor is defined as a MOSFET with drain and source shorted (or a bi-polar transistor with emitter and collector shorted). The possible values are:

- 0 - pseudo capacitors have no affect on how HiZ nodes are reported. This is the default value.
- 1 - if a node is considered HiZ and "only" pseudo capacitors are connected via their MOSFET gates (or bi-polar transistor bases) the node is not reported.

Note:

The pCap parameter has higher priority than the fanout parameter.

separate_file=[0|1]

Circuit Check places Static High Impedance Node warnings into the <hsim output prefix>.cck output file when separate_file is set to 0. If separate_file is set to 1, the results are placed into the <hsim output prefix>.SHZNd output file.

Note:

All of these parameters can be specified with global settings. Refer to Global Parameter Settings on page 321.

Example

```
cckstaticHZNode pmosOn=3.0 nmosOn=0.0
```

Static DC Path

A static DC Path is said to exist if a “DC qualified static conducting path” exists between a qualified VDD node and another qualified VDD or GND node.

A DC qualified static conducting path is a series path consisting of resistors, inductors, ‘on’ NMOS transistor drain/source channels, ‘on’ PMOS transistor drain/source channels.

To view static DC paths through the active circuit debugger (ACD), use the -acd option when calling HSIM as shown below:

```
hsim input_deck.sp -acd -o output_prefix
```

The ACD browser opens, shows static DC paths grouped by qualified power node name, and traces them by element or node.

To execute static DC path analysis, the parameter cckStaticDCPath must be specified in the circuit check command file.

```
 .param hsimcktcheck=<circuit check command file>
```

cckStaticDCPath

Syntax

```
cckStaticDCPath pmosOn=val0 nmosOn=val1 <separate_file=0|1>
   <diodeon vt_value>
```

Parameters

pmosOn=<val0> (required)

pmosOn defines the voltage threshold <val0> used to determine ‘on’ p-MOSFET transistors.

nmosOn=<val1> (required)

nmosOn defines the voltage threshold <val1> used to determine ‘on’ n-MOSFET transistors.

separate_file=[0|1]

Circuit Check places Static High Impedance Node warnings into the <hsim output prefix>.cck output file when separate_file is set to 0. If separate_file is set to 1, the results are placed into the <hsim output prefix>.SDCPath output file.

diodevalue=vt_value

If Vanode - Vcathode >= vt value you specify, the conducting path is formed between P and N. Otherwise no conducting path is formed between P and N. If you do not use this argument, by default all diode devices are considered not to be conducting.

Note:

All of these parameter can be specified with global settings. Refer to Global Parameter Settings on page 321.

Example

```
cckStaticDCPath pmosOn=3.0 nmosOn=0.0
```

CircuitCheck Utilities

These CircuitCheck commands are designed to check various circuit functions that are not covered in one of the other categories. These commands include:

- cckBasic
- cckCompareOp
- cckPatternMatch on page 316
- cckPatternConstraint on page 317
- cckSetMosDir on page 319
- cckTgPair on page 319

Basic Checking

cckBasic

cckBasic performs a basic check for valid parameters, substrate connections, node paths to voltage sources and floating gates. Warnings are reported when errors or inconsistencies are found.

Syntax

cckBasic [0|1] num=<n>

Parameters

cckBasic [0|1]

> If 0 or 1 is not specified in cckBasic, CircuitCheck issues a warning message in the log file. cckBasic 0 is the default. Set to 1 to turn basic checking on.

num=<n>

> This value limits the total number of Warnings reported. The default is 300.

Description

To use this CircuitCheck option, add cckBasic 1 in the CircuitCheck command file. No other commands must be put into this file. CircuitCheck then checks for the following:

- Parameters
- Substrate connection before DC
- Whether nodes have path to voltage sources
- Floating gate

Example

```
cckBasic 1 num=2000
```

This enables cckBasic and sets the maximum warnings to 2000.

Comparing DC Results Between HSIM and Other Simulators

cckCompareOp

cckCompareOp provides an automatic DC comparison of HSIM's results with results in a given reference file. The comparison automatically finds the nodes within the given file, and then compares HSIM's results with the reference data at those specified nodes.

The function of cckCompareOp is to automatically compare the DC OP, which is computed in the current simulation with a given DC OP reference file using the following syntax:

Syntax

```
cckCompareOp <refFile=fileName> <format=[eldo|hsim]>
   <subckt=subckt_name> <node=node_name>
   <skipsub=skip_sub_name> <skipnode=skipnode_node_name>
   <time=doComparisionAtThisTime >
   <outFile=output_file_name>
```

Syntax Definitions

refFile

Selected reference file.

format

Currently, only eldo and hsim formats are supported.

subckt, node

subckt/node to be compared, cckCompareOp will use this setting as a filter to filter out nodes listed in the reference file. Only nodes specified in the scope of subckt/node will be compared.

skipsub, skipnode

Contrary to subckt/node, skipsub/skipnode will skip nodes in the given scope when comparing with reference nodes.

time

DC time at which comparison will be conducted, default is 0.

outFile

Compared results file. If not specified, the output will be written to the <prefix>.cck file, where <prefix> is given via the -o option of HSIM.

Output Fields

The output is an ASCII report file containing the following fields:

Node_Name

Compares ELDO and HSIM hierarchical names.

HSIM_DC_OP(V)

HSIM-provided DC op condition.

REFERENCE_DC_OP(V)

Eldo-provided DC op condition.

Relative difference in %

Relative to HSIMVDD

The output file is sorted by relative difference with the highest value on top. The reference DC OP format is ELDO.

Examples

```
cckCompareOp refFile=inv1_0.iic time=0n format=eldo \
outFile=hsim_cmpop.log skipsub=inv10 skipnode=*
cckCompareOp refFile=inv1_100n.iic time=100n format=eldo \
outFile=hsim100_cmpop.log node=*
```

The following is a typical cckCompareOp output file.

```
***************************************************
* Comparison of operating points.
*
* Spectre OP file: inv1_100n.iic
* transient time: 0
***************************************************
***-------------Start Compare Op----------------***
***---------------------------------------------***
Node_NameHSIM_DC_OP(V)Eldo_DC_OP(V)(HSIM-Eldo)_DC_OP/
               Hsim_Vdd(%)
x3.5 +5.0000000000000000-0.0000017426983195+166.667
x5.9 +5.0000000000000000-0.0000042658173509+166.667
x5.7 +5.0000000000000000-0.0000006782416429+166.667
x5.5 +5.0000000000000000-0.0000027092119370+166.667
x1.3 +5.0000000000000000-0.0000007398899505+166.667
x5.3 +5.0000000000000000-0.0000001779232473+166.667
x1.5 +5.0000000000000000-0.0000001967053253+166.667
x5.1 +5.0000000000000000-0.0000024511264310+166.667
x1.7 +5.0000000000000000-0.0000024611588271+166.667
x4.10+5.0000000000000000-0.0000000221863606+166.667
x1.9 +5.0000000000000000-0.0000001658784807+166.667
x4.8 +5.0000000000000000-0.0000029060007400+166.667
x4.6 +5.0000000000000000-0.0000022913118397+166.667
x2.2 +5.0000000000000000-0.0000027995864791+166.667
x4.4 +5.0000000000000000-0.0000017390747854+166.667
***------------------------------------------------------***
***-----------------Compare Op End-----------------------***
```

Find Subcircuit Instances

cckMatchSub

Reports all the instances of the specified subcircuits to the output file <prefix>.ccksub by traversing the hierarchical netlist.

Syntax

```
cckMatchSub <subckt=subckt_name1,subckt_name2, ...etc.>
   <ReptHierNode=0|1>
```

Parameters

subckt

The subckts specified are the ones to be matched.

ReptHierNode

Setting ReptHierNode=1 reports all the hierarchical node names in the output file <prefix>.ccksub. ReptHierNode=0 is the default.

Examples

```
cckMatchSub subckt=buf1 ReptHierNode=1
```

The output is shown in the following:

```
Instances of Sub buf1:
Instance 1: x1
* Port *
in
-- x1.in
-- tx1
out
-- x1.out
-- tx2
vdd
-- x1.vdd
-- v3
gnd
-- x1.gnd
-- v5
Instance 2: x2
* Port *
in
-- x2.in
-- tx2
out
-- x2.out
-- tx3
vdd
-- x2.vdd
-- v2
gnd
-- x2.gnd
-- v4
2 Instances of Sub buf1 in Total
```

Pattern Matching Capability

cckPatternMatch

Syntax

```
cckPatternMatch <file=pattern_file_name>
   <subckt=subckt_name> <ReptHierNode=0|1>
   <IgnorePattern=pattern_name(s)> <noOverlap=1|0>
   <MatchModel=0|1> <ReptParallel=0|1>
```

Description

cckPaternMatch enables pattern matching to designated devices while conducting selected CircuitCheck commands. Compatible commands include:

- cckTgPair
- cckDlySkipElem
- cckXtalkSkipElem

Parameters

file

Specifies the pattern file name containing the patterns to be selected for matching to the circuit.

subckt

Defines the specified subckt as the scope to apply pattern matching.

ReptHierNode

Setting ReptHierNode=0 default causes CircuitCheck to only report the pattern matched node. When ReptHierNode=1, the pattern matched node and the up-stream hierarchical nodes are reported as well.

IgnorePattern

Specifies pattern name(s) that are excluded during pattern matching. Multiple pattern names are specified and wildcards are supported.

noOverlap

noOverlap is used only if there is an overlap between pattern matching results.

noOverlap=1

default specifies that the MOSFET devices inside the circuit are matched only once during pattern matching.

noOverlap=0

indicates that the MOSFET devices inside the circuit are matched multiple times during pattern matching. This option is used only there is overlap between mappings of the pattern matching result.

MatchModel

MatchModel=1 requires that the models of the mappings of the pattern matching result must also match the models of the patterns specified in the pattern file. MatchModel=0 is the default.

ReptParallel

ReptParallel=1 reports all parallel devices in the output file <prefix>.cckpat_match. ReptParallel=0 is the default.

Example

The following example shows multiple pattern names in the IgnorePattern extension:

```
cckPatternMatch file=pattern IgnorePattern=p1 p2 p12*
```

cckPatternConstraint

Syntax

```
cckPatternMatch file=aa subckt=s32_d1lat_0
cckDlyskipElem pattern=1 inst=m3 inst=m4 subckt=s32_d1lat_0
cckPatternConstraint pattern=<pattern_name>
Wlratio=<logical_expression>
cckNoSimu 1
```

where file=aa within cckPatternMatch points to the file that contains the pattern to be matched as shown in Example 55.

Examples

Example 56

```
** 1st line must be comment or empty (same as spice file) **
.subckt 1 n3 n5
m3 n5 n3 vdd p
m4 n5 n3 gnd n
m5 n3 n5 vdd p
m6 n3 n5 gnd n
.ends
```

Note:

> Each pattern in the file must be described in SPICE format terms except, there size information is not required. All or the patterns are grouped in the same file specified in the file option.

Pattern matching will be enabled and applied onto the cckDlySkipElem command, to look for the designated pattern named 1 within the scope of subckt s32_d1lat_0. If there is a match, the corresponding m3 and m4 transistors will be classified as qualified candidates to be skipped. Since the command file also contains cckPatternConstraint, which is used to specify additional matching criteria, the pattern 1 in this example must meet the matching criteria of the WLratio='m3>m5'.

If cckNoSimu=1, no simulation is performed. Example 56 shows an example.

Example 57

```
***-----------------------------------***
In sub (s32_d1lat_0) match 1 (1) patterns
Mapping 0:
* Mos *
m3 -- xia2.mp1
m4 -- xia2.mn1
m5 -- xia4.mp1
m6 -- xia4.mn1
* Net *
vdd -- vdd
gnd -- gnd
n3 -- zo
n5 -- o
***-----------------------------------***
```

Note:

Nodes connected to voltage sources are treated differently. Example 57 illustrates two patterns that are not the same.

Example 58

```
<pattern 1>
.subckt clampnmos11 vdd nb
MMa vdd nc nb nb NY
RRa nb nc $[RF]
.ends
<pattern 2>
.subckt clampnmos11 na nb
MMa na nc nb nb NY
RRa nb nc $[RF]
.ends
```

In pattern 1 has nodes connected to the global VDD signal which is a voltage source node. In pattern 2, the global VDD signal is a non-voltage source node na. Since these two patterns are not the same, pattern matching will not treat them the same.

cckSetMosDir

cckSetMosDir is used to set the signal or current flow direction through MOSFETs in order to perform other static and dynamic tests. See cckSetMosDir on page 259 for detailed description.

Setting Transistor Directions

There are two methods for setting transistor direction at either the gate or circuit level by using cckTgPair.

cckTgPair

```
cckTgPair <dirN=[D2S|S2D]> <subckt=subcktname>
   <inst=nmosName> <pattern=patternname>
```

Parameters

cckTgPair

Works with pattern matching technology to identify designated transistors based on patterns pre-defined using cckPatternMatch. Pattern names selected from those predefined are matched. The direction of the transistors to be matched will be limited to those with the direction specified in the dirN option. Refer to the cckPatternMatch section for details.

Method 1: Setting a User-Specified Transistor Gate Transistor Direction
It is not easy to determine the correct direction for a transfer gate. Therefore, transistor direction in a transfer gate may be specified using the following syntax:

```
tgPair <dirN=[D2S|S2D]> <subckt=subcktName> <inst=nmosName>
```

Parameters

tgPair

Transfer gate pair (tgPair) includes an n-MOSFET and a p-MOSFET forming a transfer gate.

dirN=D2S|S2D

The parameters are defined as drain-to-source (D2S) and source-to-drain (S2D). The direction of an n-MOSFET device is set using this syntax.

Note:

Only the n-MOSFET direction needs to be set since other p-MOSFETs are found in this pair and assigned the same direction as in n-MOSFET.

Example

```
tgPair dirN=S2D subckt=aa inst=mn1
```

subckt=aa, finds a n-MOSFET transistor mn1 and assigns its direction as S2D. All the instantiations of aa will have this direction for mn1.

The node name may also be used as the starting point to define the direction. For example, a MOSFET device has a drain node called dd and source node called ss. If this transistor is from drain to source, then node dd sets the direction in mn1.

```
tgPair <fromNode=nd1> <subckt=aa> <inst=mn1>
```

where `fromNode=nd1`. The `mn1` instance in `subckt aa` is first found. Node `nd1` must be either a drain or source node of `mn1`. If `nd1` is `mn1`'s drain node, this syntax command sets the `mn1` direction to be drain-to-source. If `nd1` is `mn1`'s source node, then the `mn1` direction is source-to-drain.

Method 2: Defining Transistor Direction in a Circuit. This method defines the direction for an individual transistor in a subckt. It is added into an input SPICE file.

Syntax

```
.param <hsimMosDir=[1|2|3]> <subckt=aa> <inst=bb>
```

Parameters

hsimMosDir=1

Source-to-drain (S2D)

hsimMosDir=2

Drain-to-source (D2S)

hsimMosDir=3

Bi-directional (BID)

subckt=aa inst=bb

In the subckt aa, the direction selected by the option is assigned to transistor bb. The transistors bb in all the instantiations of aa will have the same direction.

Global Parameter Settings

Global parameter settings allow you to specify, un-specify, specify_global, and restore_global CircuitCheck commands when setting parameters for the following CircuitCheck commands:

- cckMosV on page 173
- cckCapV on page 180
- cckDioV on page 180
- cckResV on page 180
- cckFloatGateIsrc on page 194
- cckDiode on page 188
- cckNmosB_gt_DS on page 196
- cckNmosG_gt_DS on page 200
- cckPmosB_lt_DS on page 213
- cckPmosG_lt_DS on page 217
- cckStaticHZNode on page 308
- cckStaticDCPath on page 310

The parameters covered by these commands are: vlth, vhth, vpth, vnth, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, num, rptTrace, vt, subinfo, listall, limitmos, risePmosfallNmos, separate_file, filterAlert, pmoson, nmoson, trace, pcap, fanout, lvp, lvn, uvp, uvn, extTrace, and mode.

Each parameter has a current value and a global value. Current values and global values can be different.

For each CircuitCheck command, the value of a given command parameter is assigned based on the following priority order:

1. Local command value if defined
2. Current value if defined
3. Global value if defined
4. Parameter remains undefined (in some instances, the default value will be used)

specify Syntax:

```
specify param1=val1 … paramN=valN
```

Sets the current values for the parameters used in all of the commands defined after the specify command.

unspecify Syntax:

```
unspecify param1 … paramN
```

Removes the current values of param1 … paramN used in all of the commands defined after the unspecify command.

```
unspecify *
```

Removes the current values from all parameters.

specify_global Syntax:

```
specify_global param1=val1 … paramN=valN
```

Sets the global values of parameters used in all commands that are defined in the CircuitCheck command file.

restore_global Syntax:

```
restore_global param1 … paramN
```

Restores the current value of param1 … paramN to the global values.

```
restore_global *
```

For all parameters with global values defined, the current value of each parameter is restored to the global value.

Example:

CircuitCheck command file:

```
-----------------------------------------------------------
cckPmosG_lt_DS vhth=1.0 inst=* vt=0.3 vnth=0 vpth=0 rptTrace=1
cckPmosB_lt_DS vhth=1.0 inst=* vt=0.3 vnth=0 vpth=0 rptTrace=1

specify vhth=1 uvg=4.0 lvd=1.0
cckMosv NA_report model=NA inst=* vhth=0.1 uvs=2.0 uvb=2.0 vnth=0
vpth=0 rptTrace=1

specify_global vpth=-0.5 rptTrace=0 uvg=3.0 uvs=3.0 lvs=1.0 lvd=0
cckMosv PA_report model=PA inst=*  lvd=2.0  lvs=2.0 uvb=2.0 vnth=0

restore_global uvg lvd
cckMosv tag1 model=NA inst=*  uvs=2.0 uvb=2.0 vnth=0 vpth=-0.5
rptTrace=0

cckMosv tag2 model=PA inst=* vhth=0.1 uvg=5.0 lvs=0 uvb=2.0 vnth=0
vpth=-0.5 rptTrace=0
```

Resulting CircuitCheck commands:

```
--------------------------------------------------------------
cckPmosG_lt_DS vhth=1.0 inst=* vt=0.3 vnth=0 vpth=0 rptTrace=1
uvg=3.0 uvs=3.0 lvs=1.0 lvd=0

cckPmosB_lt_DS vhth=1.0 inst=* vt=0.3 vnth=0 vpth=0 rptTrace=1
uvg=3.0 uvs=3.0 lvs=1.0 lvd=0

cckMosv NA_report model=NA inst=* vhth=0.1 uvg=4.0 uvs=2.0 uvb=2.0
lvd=1.0 vnth=0 vpth=0 rptTrace=1 lvs=1 rptTrace=1

cckMosv PA_report model=PA inst=* lvd=2.0 lvs=2.0 uvb=2.0 vnth=0
uvg=4.0 vpth=-0.5 rptTrace=0 uvs=3.0

cckMosv tag1 model=NA inst=* vhth=1 uvg=3.0 lvd=0  lvs=1 uvs=2.0
uvb=2.0 vnth=0 vpth=-0.5 rptTrace=0

cckMosv tag2 model=PA inst=* vhth=0.1  uvg=5.0  uvs=3 lvs=0 lvd=0
uvb=2.0 vnth=0 vpth=-0.5 rptTrace=0
```

CircuitCheck Tutorial

This tutorial provides information and examples for using CircuitCheck commands in HSIM. The commands are specified in an input netlist file as follows:

Invoking CircuitCheck

In the main input file - (top.sp), add the following lines so that HSIM will read in the two cck and device files.

```
.param hsimCktCheck=cck_cmd_file
```

The tcheck mosv commands with its parameters (see tcheck mosv on page 266) is used to monitor the node voltages. The set can be invoked by adding the following syntax to an HSIM input file.

```
.param hsimDeviceV=dev_v_file
```

The ntrig and intrig commands (see ntrig on page 242) are used in the interactive mode to find the first state change for the node after a specified time in nanoseconds. ntrig specifies the node name while intrig specifies the node ID. The report lists the elements causing the node voltage to change: i.e.,

```
ntrig node_name <-t time> <-f file1> <-mt time1>
intrig node_id <-t time> <-f file1> <-mt time1>
```

Test Case

Typical commands used in CircuitCheck include the following from the cck file.

Note:

For comments: /* (start comment) and */ (end comment) are used. /* must always be placed at the beginning of a line before simulation; for example, immediately following netlist parsing.

Note:

For syntax: “\” denotes a line continuation.

```
cckParam erMaxCap=0.001 waMaxCap=1.e-8
cckParam erMaxMosW=0.01 waMaxMosW=1000u
cckParam erMaxMosL=0.01 waMaxMosL=1000u
cckParam erMaxMosAD=0.0001 waMaxMosAD=1.e-6
cckParam erMaxMosPD=0.01 waMaxMosPD=0.001
cckParam erMaxMosTox=5.e-8 waMaxMosTox=3e-8
cckParam num=100
/* To check a particular model, add one of the following:
 * cckParam model=name, erMaxMosW=..., and waMaxMosW=...
 * These parameters contain the specific values shown in
 * this test case.*/
 cckParam erMaxMosW=2
 cckParam model=m1 erMaxMosW=4
/* In this test case, template A0 is created to keep the
 * default parameter values.
 * Line 1.Does not have any model, will overwrite the
 * maxMosW to be 2 in template A0, which is used for general
 * purpose checking.
 * Line 2.Creates a new template A1 that contains the
 * default parameter values and the new maxMosW=4.
 * Therefore, complete the following steps:
 * Define all the parameters' values for general purpose
 * checking.
 * Start to define the special values for different models;
 * one model per line where each line contains the new
 * values for this model.
 */
/* Substrate check. Forward bias threshold is 0.2 */
cckSubstrate num=500 vt=0.2 mode=2 start=10n stop=50n /
start=200n stop=300n
/* Diode check. Forward bias threshold is 0.4 */
cckDiode num=500 vt=0.4 start=10n stop=50n
/* Floating gate check */
cckFloatGateIsrc 1
/* Static leakage path check */
cckMaxStaticLeak num=100
/* Check whether a node is stuck at 0 or 1. Limit the node
checking to some area of the design indicated by an asterisk (*) */
cckMaxStuckAt0 num=5000 node=xcam* node=xram*
cckMaxStuckAt1 num=5000 node=xcam*
/* Check each pmos to determine which logic-high power
 * supply its nodes can reach.
 * a.) The (min, max) voltage sources reached from gate,
 *                  drain, src.
 * b.) If the gate node is less than drain/src by 0.3 volts,
 *              it is reported.
 * c.) Since voltage drop is considered through nmos and
 *     pmos, specify vnth=0.6 for nmos and vpth=-0.5 for pmos
 */
```

```
cckPmosG_lt_DS vhth=0.9 vnth=0.6 vpth=-0.5 inst=* vt=0.3
/* Check each pmos. If a gate has a higher voltage than D/S,
 * it is reported. */
cckNmosG_gt_DS vlth=0.7 vnth=0.6 vpth=-0.5 inst=* vt=0.2
/* Pmos substrate check. Trace from D/S and Bulk nodes. If a
 * bulk node reaches a lower power supply, it is reported. */
cckPmosB_lt_DS vhth=0.9 vnth=0.6 vpth=-0.5 inst=* vt=0.1 \
num=200
/* similar for nmos's bulk checking */
cckNmosB_gt_DS vlth=0.7 vnth=0.6 vpth=-0.5 inst=* vt=0.1 \ num=200
/* Path to vdd/gnd check*/
cckPathToVsrc num=900 node=* fanout=1
/* Add the following syntax immediately following the DC
 * operating point: */
/* To check for un-initialized latches. Focus on the
 * instances in subckt=q1latchsd1 */
cckLatchUnInit 1
cckLatchInElem subckt=q1latchsd1 inst=*
cckLatchSkipElem subckt=a920traddrbuf2 inst=xu5*
cckLatchSkipElem inst=xram.xpipe_5*
/* To get a static path delay analysis, specify: */
cckRCDlyPath 1
cckRCRiseDelay min=0.05n max=3e-9 inside=0
cckRCFallDelay min=0.04n max=3.2e-9 inside=0
cckLimitRisePmosFallNmos 1
cckSetMosDir 0
cckDlySkipElem subckt=p2sticky inst=mpwk
inst=mnwk
/* To do static crosstalk analysis, specify: */
cckXtalkFloatingCap 1.e-14
cckXtalkRiseVolt 0.04
cckXtalkFallVolt 1.5
cckXtalkByWL 0
cckXtalkmodelWLratio model=pch min=12 max=20
cckXtalkmodelWLratio model=nch min=12 max=15
cckXtalkByRC 1
cckXtalkRiseTimeConst min=0.5n max=3n
cckXtalkFallTimeConst min=0.4n max=4n
/* For a description of cckRCDlyPath see page 257,
 * for cckXtalkByRC and cckXtalkByWL, see *
 cckStaticXtalk_GroupCmd on page 287 */
/* To skip some node or look at some node, then specify: */
cckXtalkAtNode node=xcam* --> examine these nodes
cckXtalkAtNode node=xram* --> examine these nodes
cckXtalkSkipNode node=xram* --> don't go thru these nodes
cckXtalkSkipElem inst=xram.x* --> don't go thru these \ elements
/* NOTE: At this time Skip and AtNode are mutually exclusive.
 * If Skip is used, then AtNode cannot be specified.
 * Conversely, if the AtNode is chosen,
 * then Skip can't be used. */
/* during simulation, fast check node voltage, elem current
 * and leakage current */
/* in simulation, check node min and max voltage */
cckNodeVoltage num=100 vmax=2.0 vmin=-0.3
cckNodeVoltage num=100 vmax=1.64 vmin=-0.3 model=nch
cckNodeVoltage num=100 lvgs=-1.66 lvgd=-1.66 model=* \ start=60n
cckNodeVoltage num=100 lvgs=-1.64 lvgd=-1.64 model=pch \
start=20n stop=50n
/* (Note: in this comment an asterisk (*) is used to
 * represent a number.) To check the node voltage of
 * transistors connecting to constant * power supply,
 * use ...*. The following command will check the tp018
 * transistors
 * connecting through * resistors to constant power supply
 * 1.8V. If the D/S node voltage * is less than 1.79, it is
 * reported. This command can do a quick check * on
 * IR-drop */
cckNodeVoltage model=tp018 constV=1.8 vmin=1.79
/* Check the current through element in simulation */
cckElemI ith=1.e-6 model=pch constV=1.65
```

Test Case Example for tcheck mosv

Typical commands used in CircuitCheck for device voltage checking include the following from the tcheck mosv file.

```
.tcheck jtlv mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1 \ lvsb=-
2.6 uvsb=-1.5 stop=30n start=60n
.tcheck jtl2 mosv model=pch \ mos=xram* lvgs=-0.6 uvds=1 .tcheck
jtl1 mosv model=nch mos=* cond=' \
(vds < -1.0 || vgs < -0.75)'
/* to do post-process device voltage check, add */
.tcheck post=1
.tcheck post=2
```

Run HSIM

When post=2 appears, voltage will continue to be read in the fsdb file using the same process.

Note:

If post=1, then 2 processes are run:

- Standard HSIM
- HSIM -post_devv

9

HSIM-ADMS Integration

Provides information on the single-kernel integration of HSIM into Mentor's multilanguage ADvanced Mixed-signal Simulator (ADMS).

Introduction to HSIM-ADMS

HSIM-ADMS is the single-kernel integration of HSIM into Mentor Graphics'® multilanguage ADvanced Mixed-signal Simulator™ (ADMS). This integration adds capability for co-simulation of circuit blocks represented in SPICE format in HSIM, along with VHDL, Verilog, VHDL-AMS, and Verilog-AMS in ADMS. HSIM-ADMS allows designers to verify large mixed-signal designs with the flexibility of simulating various blocks at different levels of abstraction.

ADMS Overview

ADMS is a language-neutral, mixed-signal simulator that enables top-down design and bottom-up verification of multimillion gate analog/mixed-signal System-on-Chip (SoC) designs. The integration with HSIM provides a choice of four high-performance simulation engines: HSIM, Eldo, ModelSim, and Eldo-RF for modulated steady state simulation. ADMS is integrated into the Mentor Graphics AMS SoC Design flow and the Virtuoso® Analog Design Environment. For more detailed information on the features, tool flow, and usage of ADMS please refer to Mentor's ADMS User's Manual.

HSIM-ADMSTool Setup

Licensing

Mentor ADMS License

In order to add HSIM-ADMS integration to Mentor Graphics' ADMS simulator, a license must be obtained for the following Mentor part number:

Part Number	Product Name	Category	Class
222514	ADMS	Foreign SPICE I/F	Op SW

Synopsys HSIM License

To enable HSIM simulation in ADMS, the "hsim-adms" license feature is required.

Installing and Configuring ADMS

Follow these steps to install and configure ADMS:

1. Install the ADMS software tree into the desired location, as described in the Mentor Graphics' *ADMS Installation Manual*.
2. Install the HSIM software tree into a separate location as described in the *<Product Name> User's Manual*, Chapter 3: Installation and Operation.
3. Set up the ADMS environment using the following syntax:

```
setenv anacad <tree where you installed ADMS>
source $anacad/com/init_anacad
```

4. Set up the HSIM environment using the following syntax:

```
setenv HSIM_LIBRARY_PATH <path inside the HSIM software tree
to libadmshsim.so>
setenv LD_LIBRARY_PATH $HSIM_LIBRARY_PATH:<path to
libtcl8.4.so>:$LD_LIBRARY_PATH
```

5. Set up the Licenses using the following syntax:

```
setenv LM_LICENSE_FILE <path to ADMS license>:<path to HSIM
license>
```

HSIM-ADMS High-level Architecture and Data Flow

HSIM-ADMS employs black box (BB) type communications for design elaboration using the architecture shown in Figure 36 on page 359.

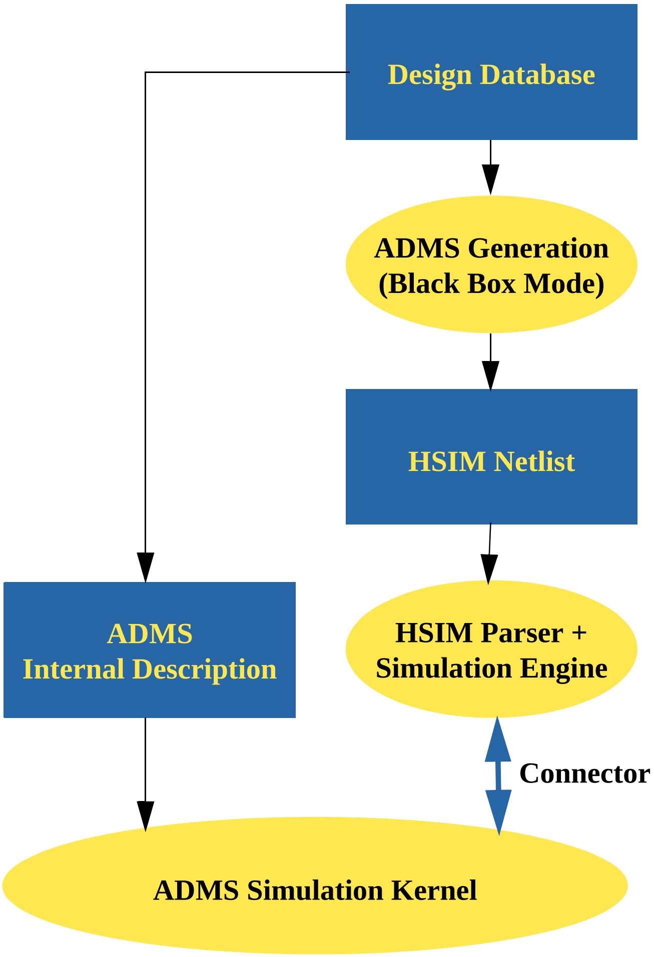

Design Database
ADMS Generation (Black Box Mode)
HSIM Netlist
ADMS Internal Description
HSIM Parser + Simulation Engine
Connector
ADMS Simulation Kernel

Figure 36 HSIM-ADMS Architecture

ADMS starts elaborating the design in the Generation phase where it accomplishes the following:

- Loads all compiled modules
- Establishes block-level connectivity

Note:

HSIM establishes connectivity inside the blocks.

During parsing and elaboration, the ADMS kernel analyzes the SPICE portion of the design looking for all subcircuit instances whose definitions are defined as HSIM black box in the ADMS configuration commands located in the command file.

ADMS then creates a SPICE netlist containing all of the elements that you specified to be sent to HSIM. By virtue of this type of black box implementation SPICE block parsing accomplished by the ADMS kernel is minimal. It is accomplished only to the extent that is required for ADMS to send these blocks to HSIM. ADMS only looks inside the SPICE blocks to see the X instances of subcircuits that are sent to HSIM.

The SPICE intermediate netlist containing all the HSIM elements is then sent to HSIM. HSIM parses the netlist and reports warnings or errors as they would be in a stand-alone simulation. When parsing is successfully completed, simulation begins with ADMS as the master and HSIM as the slave. ADMS coordinates all of the synchronization between ADMS and HSIM engines.

HSIM-ADMS Simulation Flow

To simulate a design in HSIM-ADMS, perform the following steps:

1. Compile language modules and entities.

   All ADMS language modules, including the following types, must be compiled into ADMS libraries:

   - Verilog
   - Verilog-AMS
   - VHDL
   - VHDL-AMS

   ADMS provides various compilation commands to meet the requirements for different hierarchical structures such as SPICE instantiating Verilog, VHDL instantiating SPICE, etc.

   Note:

   Refer to Mentor Graphics' *ADMS User's Manual* for complete information on creating ADMS libraries and how to correctly compile language-based models.

2. Create a command file.

   Create a command file containing the appropriate analysis, post processing commands, etc. The ADMS command file is then inserted into the simulator as follows:

   - SPICE-on-Top: If SPICE is at the top level of the design hierarchy, the command file is simply the top level SPICE netlist.
   - Verilog-, VHDL-, Verilog-AMS-, Verilog-AMS-on-Top: If any of these language modules is at the top level of the design hierarchy, the command file should contain all of the following:

     Post Processing Commands: .meas, .plot, .probe, etc.

     Design Parameter List: .param definitions, etc.

     Analysis Commands: .tran , .AC, etc.
   - Language-on-Top Restrictions: The only restriction pertaining to Language-On-Top is that the command file must not contain any X instances.

     Note:

     > Refer to the *ADMS User's Manual* to find complete information on how to create these command files.

3. Insert HSIM-ADMS partitioning commands.

   ADMS enables different partitioning commands to specify the blocks to be sent to the HSIM simulator.

   - Partitioning Command Placement: Partitioning commands can be handled in either of the following ways:

     Inserted directly into the command file

     Inserted using .include from another text file
   - Specifying Blocks: Specify the blocks to be sent to HSIM using ADMS partitioning commands described in HSIM-ADMS Configuration Commands on page 366.

     Note:

     > Refer to the ADMS User's Manual for complete information on writing partitioning commands for HSIM-ADMS.

4. Add HSIM-ADMS command file options.

   HSIM and/or ADMS options are added as follows:

- ADMS options: ADMS .cmd file
- HSIM options: hsim.ini file

  Note:

  > Refer to the *ADMS User's Manual* for complete information on adding ADMS options to the ADMS .cmd file. Also refer to Chapter 3 of the *<Product Name> User's Manual* for a description of hsim.ini files.

5. Invoke HSIM-ADMS using the `vasim` command.

   After the design is compiled into an ADMS library, and the appropriate command file is created, HSIM-ADMS is invoked using the command `vasim`.

HSIM-ADMS Examples

Figure 37 on page 362 illustrates a five element inverter chain with Verilog at the top level. The inverter chain has both Verilog- and SPICE-based inverters.

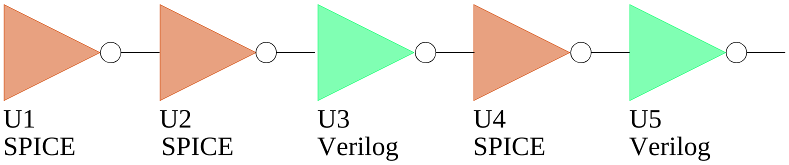


Figure 37 HSIM-ADMS Inverter Chain

Based on the HSIM-ADMS inverter chain shown in Figure 37, Example 59 is an example of the ADMS command file test_admshsim_verilogontop.sp.

Example 59 ADMS Command File Example: test_admshsim_verilogontop.sp

```
* HSIM-ADMS partitioning commands
.HSIMBB
.part bb subckt=(inv_spice, supply)
.option compat
.lib 'c018lv.l' TT
.temp 27
*subckt definitions for the inverters and power supply block
.subckt supply vdd vin
vvdd vdd 0 pwl(0 0 10n 0 12n 1.8 30n 1.8 32n 0.0 60n 0.0 62n +
1.8 150n 1.8)
vvin vin 0 pwl(0 0 10n 0 12n 1.8 30n 1.8 32n 0.0 60n 0.0 62n +
1.8 150n 1.8)
.ends
.subckt inv_spice vdd in out
m1 out in vdd vdd pch w=10e-6 l=0.18e-6
m2 out in 0 0 nch w=5e-6 l=0.18e-6
c0 out 0 1fF
.ends
```

The boundary elements shown in Example 60 are required to allow correct domain conversion from Analog to Digital and vice-versa.

Example 60 ADMS A2D and D2A Boundary Elements

```
.defhook a2d_def d2a_def
.model d2a_def d2a mode=std_logic vhi=1.8 vlo=0.0
.model a2d_def a2d mode=std_logic vth=0.9
* Post Processing and Analysis Commands
.measure tran freq1 TRIG V(ring_out) VAL=0.9 TD=20e-9 RISE=1 +
TARG V(ring_out) VAL=0.9 TD=20e-9 RISE=2
.measure tran freq2 TRIG V(ring_out) VAL=0.9 TD=90e-9 RISE=1 +
TARG V(ring_out) VAL=0.9 TD=90e-9 RISE=2
.plot tran v(ring_out) v(s1) v(s2) v(s3) v(s4) v(s5) v(s6) + v(vdd)
.tran 1n 30n
.end
```

Example 61 Verilog Top-Level Module: top.v

```
module top;
wire ring_out,vin,vdd;
inv_spice I1(vdd,vin ,s1);
inv_spice I2(vdd,s1,s2);
inv_verilog I3(vdd,s2,s3);
inv_spice I4(vdd,s3,s4);
inv2_verilog I5(vdd,s4,ring_out);
supply m2 (.vdd(vdd), .vin(vin));
endmodule
```

Example 62 ADMS do File

```
run -all
quit -f
```

Use the following steps to compile and run simulation in Verilog-on-top configurations:

1. Create the ADMS Library using the following syntax:

   ```
   valib amslib
   ```

2. Compile the Verilog inverter into the ADMS Library using the following syntax:

   ```
   valog inv_verilog.v -ms
   ```

3. Compile the Verilog Top Level module into the ADMS library using the following syntax:

   ```
   valog top.v -ms
   ```

4. Invoke HSIM-ADMS using the command file test_verilogontop.sp and do file ams.do using the following syntax:

   ```
   vasim -cmd test_admshsim_verilogontop.sp top -do ams.do
   ```

5. Invoke the EZwave Viewer to see the waveforms using the following syntax:

   ```
   ezwave test_admshsim_verilogontop.wdb &
   ```

The same inverter chain illustrated in Figure 37 can be run using a SPICE-on-TOP configuration as shown in Example 63 through Example 67:

Example 63 SPICE-on-Top

```
Command file: test_admshsim_spiceontop.sp.
.lib 'c018lv.l' TT
.temp 27
```

Example 64 HSIM-ADMS Partitioning Commands

```
.HSIMBB
.part HSIMBB subckt=(inv_spice)
* Top level Netlist connectivity
X1 vdd vin s1 inv_spice
X2 vdd s1 s2 inv_spice
Y3 inv_verilog PORT: vdd s2 s3
X4 vdd s3 s4 inv_spice
Y5 inv_verilog PORT: vdd s4 ring_ out
vvdd vdd 0 1.8
vin in 0 pwl(0 0 10n 0 12n 1.8 30n 1.8 32n 0.0 60n 0.0 62n
+ 1.8 150n 1.8)
.subckt inv vdd in out
m1 out in vdd vdd pch w=10e-6 l=0.18e-6
m2 out in 0 0 nch w=5e-6 l=0.18e-6
c0 out 0 1fF
.ends
```

Example 65 HSIM Model Card to Define The Verilog Inverter Model to be Instantiated from SPICE

```
.model inv_verilog macro lang=verilog mod=inv
```

Example 66 ADMS A2D and D2A boundary elements are required to allow correct domain conversion from A2D and D2A.

```
.defhook a2d_def d2a_def
.model d2a_def d2a mode=std_logic vhi=1.8 vlo=0.0
.model a2d_def a2d mode=std_logic vth=0.9
```

Example 67 Post-processing and Analysis Commands

```
.measure tran freq1 TRIG V(ring_out) VAL=0.9 TD=20e-9 RISE=1 +
TARG V(ring_out) VAL=0.9 TD=20e-9 RISE=2
.measure tran freq2 TRIG V(ring_out) VAL=0.9 TD=90e-9 RISE=1 +
TARG V(ring_out) VAL=0.9 TD=90e-9 RISE=2
.plot tran v(ring_out) v(s1) v(s2) v(s3) v(s4) v(s5) v(s6)
+ v(vdd)
.tran 1n 30n
.end
```

Follow these steps to compile and run the design in a SPICE-on-TOP configuration using HSIM-ADMS commands.

1. Create the ADMS Library using the following syntax:

   ```
   valib amslib
   ```

2. Compile the Verilog inverter into the ADMS library using the following syntax:

```
valog inv_verilog.v -ms
```

3. Invoke HSIM-ADMS using the command file test_spiceontop.sp and ams.do do file using the following syntax:

   ```
   vasim -cmd test_admshsim_spiceontop.sp top -do ams.do
   ```

4. Invoke the EZwave Viewer to see the waveforms using the following syntax:

   ```
   ezwave test_admshsim_spiceontop.wdb &
   ```

HSIM-ADMS Configuration Commands

Partitioning Your Design

HSIM-ADMS provides three different ways to partition your design between ADMS and HSIM:

- .HSIMBB and .part: Partitioning commands
- #HSIMBB and #ENDHSIMBB: Black-box delimiters
- .bbinclude: Black-box include function

Since HSIM only supports SPICE and Verilog-A models, only blocks containing SPICE or Verilog-A elements are sent to HSIM. The current partitioning model is subcircuit-based so that all subcircuit instances specified in the partitioning commands will be sent to HSIM.

.HSIMBB and .part

Multiple .part commands can be used in the netlist. However, if two .part commands refer to the same subckt name, one of them will be ignored. These commands should appear at the netlist header.

Syntax

```
.HSIMBB
.part HSIMBB subckt=<comma separated subckt name list>
```

.HSIMBB

Tells ADMS to activate HSIM.

.part

Lists the subckt-names to send to HSIM.

#HSIMBB and #ENDHSIMBB

The delimiters `#HSIMBB` and `#ENDHSIMBB` can be used to enclose sections of a netlist containing only subckt definitions.

#HSIMBB

Indicates the beginning of an enclosed list of subcircuit definitions.

#ENDSHIMBB

Indicates the end of an enclosed list of subcircuit definitions.

.bbinclude

Adds the subcircuit definition file to the main netlist.

Syntax

```
.bbinclude <netlist containing subckt definitions>
```

HSIM-ADMS Hierarchy

Only the top subckt in a given hierarchy tree must be specified for HSIM in black box mode and ADMS sends the entire hierarchy tree to HSIM. Figure 38 provides an example.

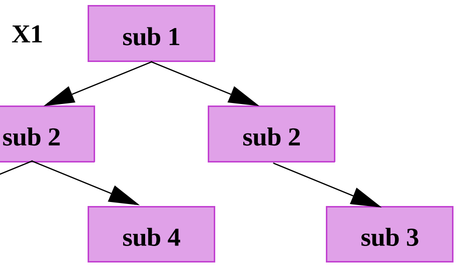


Figure 38 HSIM-ADMS Hierarchy Tree

Specify only the top level sub1 as HSIM black box mode to send the entire hierarchy tree depicted in Figure 38 to HSIM. Example 68 and Example 69 illustrate two methods to accomplish this procedure.

Example 68 Using #HSIMBB and #ENDHSIMBB

```
#HSIMBB
.subckt sub1 n1 n2 n3
     M1 …
.ends
.subckt sub2 n1 n2 n3
     M2 …
.ends
.subckt sub3 n1 n2 n3
     M3 ….
.ends
.subckt sub4 n1 n2 n3
     M4 ….
.ends
#ENDHSIMBB
X1 n1 n2 n2 sub1
```

Example 69 Using .part

```
.HSIMBB
.part BB subckt = (sub1)
X1 n1 n2 n2 sub1
```

HSIM-ADMS Control Options

Passing HSIM Options in HSIM-ADMS

HSIM accepts various simulation control options for the following functions:

- Accuracy
- Simulation Speed
- Netlist Format
- Vector Files

In HSIM-ADMS, these options can be passed to HSIM through the following methods:

- ADMS Netlist file
- ADMS Command line

Passing HSIM Options in the ADMS Netlist File

HSIM options are specified by .param statements specified in the ADMS netlist file. Use the following syntax:

```
.param HSIM<optionname>=<val>
```

In the ADMS netlist file, .param statements should be enclosed between #HSIMBB and #ENDHSIMBB. For example:

```
#HSIMBB
.param HSIMANALOG=2.0
.param HSIMTAUMAX=1us
#ENDHSIMBB
```

Passing HSIM Options on the ADMS Command Line

HSIM options can also be passed to ADMS using the ADMS command line with the following syntax:

```
vasim -cmd <command file> <other ADMS arguments> -eldoopt
   "-hsimopt <HSIM options separated by spaces>
   -endhsimopt>”
```

The following example illustrates this feature.

```
vasim -cmd test.sp -eldoopt "-hsimopt -eldo -endhsimopt"
```

HSIM-ADMS DC Iterations

The maximum number of DC iterations can be controlled using the HSIMDCITER option in the ADMS command file using the following syntax:

```
.option HSIMDCITER=<val>
```

HSIM-ADMS Boundary Elements

Since HSIM-ADMS is a mixed-signal simulator, a provision to define boundary elements that convert signals from analog-to-digital and digital-to-analog domains is required. ADMS provides several types of built-in boundary elements to support this function. User-designed custom boundary elements can also be added in VHDL-ADMS to facilitate specialized modeling requirements. Refer to the *ADMS User’s Manual* for detailed information on HSIM-ADMS boundary elements.

Special Supply Converters

A special type of D-to-A boundary element is employed when a SPICE subcircuit is powered by a digital power supply. To control digital supply converters, the following syntax is used:

```
.model <user supplied model_name> D2A mode=std_supply
   vhi=<val> vlo=<val>
```

In the syntax example above, a D2A element is defined with a user-supplied name. This converter is a special supply converter that allows ADMS to partition SPICE when the power supplies are coming from a digital domain.

You must also use `.hook` syntax to insert this boundary element on the specified supply net, as in the example below.

The following example illustrates a code sample to be placed into the ADMS command file that defines and uses the specified digital supply converters.

Commands to insert the boundary elements with special supply converters on the nets vdd_pll, vss_pll:

```
.defhook a2d_def d2a_def
.hook vdd_pll mod=d2a_POWER
.hook vss_pll mod=d2a_POWER
```

Commands to define the default boundary elements:

```
.model d2a_def d2a mode=std_logic vhi=1.8 vlo=0.0
.model a2d_def a2d mode=std_logic vth=0.9
```

Commands to define the special digital supply boundary element:

```
.model d2a_POWER d2a mode=STD_SUPPLY vhi=1.8 vlo=0.0
```

Interactive Debugging

It is possible to do interactive debugging during ADMS simulation. In order to pass HSIM commands to ADMS during simulation, add following code into your `admshsim.tcl` file, located in your home directory:

```
proc hsim { command } {
  set tcp [ms_ipc::getClient hdlav3]
    dp_RPC $tcp starteldotcl "hsim $command"
}
```

Next, add the following line into your .vams_setup file, which is also located in your home directory:

```
source ~/admshsim.tcl
```

During an ADMS simulation, you can pass HSIM commands at the prompt with the following syntax:

```
hsim "nc out"
```

HSIM-ADMS Outputs

Logfile Outputs

ADMS generates a logfile such as `<command_file>.chi` file. By default, HSIM generates `hsim.log` logfile

Waveform Outputs

By default ADMS generates a `.wdb` waveform output format file to EZwave.

In order to generate to `.cou` format file for Xelga, add following at the ADMS command line:

```
vasim -eldoopt "-gwl cou"
```

To generate to `.fsdb` waveform output format to turboWave (or nWave), set `HsimOutput=fsdb` in the `hsim.ini` file or enclose it between #HSIMBB and #ENDHSIMBB in the ADMS netlist file. For example:

```
#HSIMBB
HsimOutput=fsdb
#ENDHSIMBB
```

Black Box Mode Limitations[1]

HSIM-ADMS uses a black box mode implementation in which ADMS cannot see the design being sent to HSIM. Users should be aware of the following limitations in the ADMS usage model.

- Limitations when Using the Standard ADMS GUI

  The features of the standard ADMS GUI cannot be used on design blocks partitioned to HSIM including:

  - Descending the hierarchy in the structure window

  - Viewing the nets inside a subckt instance partitioned to HSIM.
  - Viewing the nets inside a verilog-A model instance partitioned to HSIM.
  - The HSIMSPEED is limited to 5.

  Therefore, net plots inside HSIM from the ADMS GUI (using add wave or add log) can not be created.

  Net plots can be developed from either of the following files:

  - ADMS command file
  - Do file

- Limitations on Changing SPICE Blocks Previously Sent to HSIM

  SPICE blocks previously sent to HSIM can be changed using the ADMS .BIND command. However, it is not possible to swap SPICE blocks internal to HSIM; such as subcircuits at levels lower than the top level of SPICE; with ADMS behavioral models.

- Limitations on Using SPICE Net Spy with HSIM

  It is not possible to use Net Spy on SPICE signals internal to HSIM. Also, add wave and add log do not work from the ADMS Do file.

References

[1] The limitations discussed in this section apply to Mentor Graphics's HSIM integration into ADMS.

10

HSIM-Virtuoso Analog Design Environment Interface

This chapter describes how to integrate HSIM into the Cadence® Virtuoso® Analog Design Environment using the HSIM-Virtuoso Interface. The approaches described in this chapter can be tightly coupled to a Cadence database that requires intensive data preparation or they can be loosely integrated while maintaining the signal cross probing functionality.

Note:

> Cadence's Analog Artist product is now known as Virtuoso Analog Design Environment, so the HSIM interface is now referred to as the HSIM-Virtuoso Interface, or for simplification purposes, the Interface. In previous releases, the interface was referred to as the Analog Artist Interface (AAI).

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

Synopsys provides an All-In-One Package that permits installation of one or multiple installation approaches listed below:

- AANNI (Native Netlist Integration)
- AAI HSIM (HSIM Socket Interface)
- AAI HSIMD (HSIM Direct Interface)
- AACoSim (CoSim Integration)

The HSIM Virtuoso Analog Design Environment Interface has been developed on Cadence Design Framework II 4.4.5. and has been tested on the following versions:

4.4.5

4.4.6

5.0.0

5.1.41

The Interface is supported on the following platforms:

- Solaris
- HPUX
- Linux

All-In-One Package (AAIM)

AAIM Installation & Setup

Synopsys provides an installation tar file. The tar file can be used to install any or all of the Interface options: AANNI, AAI HSIM Socket, AAI HSIMD, and/or AACoSim.

To install the All-In-One Package, use the following steps:

1. Obtain the tar file.

   Obtain the following installation tar file from the Synopsys ftp site:

   ```
   AAIM-<version>-mmddyear.tar.gz
   ```

2. Set up HSIM_ARTISTIF.

   Unzip and untar the tar file then, set the following environment variable:

   ```
   setenv HSIM_ARTISTIF <AAI_installation_dir>/AAIM-<version>-
   mmddyear
   ```

3. Add the nsdaAAIMPkgList and load statements.

   Add the following two statements in the .cdsinit file:

   ```
   nsdaAAIMPkgList='(“hsimD” “hsim” “AACoSim” “AANNI”)
   load(“<AAI_installation_dir>/AAIM-<version>-mmddyear/
   install/nsdaAAIMInvoker.il”)
   ```

   Important:

   nsdaAAIMPkgList must be entered as shown and placed prior to the load statement.

   All of the package(s) specified in nsdaAAIMPkgList are installed. The syntax shown above installs the following:

- AANNI
- AAI HSIM
- AAI HSIMD
- AACoSim

  To add the Sandwork's WaveView Analyzer link to the Interface, add `WV` in the nsdaAAIMPkgList statement, as shown in the following syntax:

```
nsdaAAIMPkgList='("hsimD" "hsim" "AACoSim" "AANNI" "WV")
```

  AANNI is the default package if nsdaAAIMPkgList is an empty list, as shown in the following syntax:

```
nsdaAAIMPkgList='()
```

  The menu files are based on the specific integration approaches as shown in the following:

- AACoSim: spectreVerilog.menus, spectreSVerilog.menus
- AANNI: simui.menus
- AAI HSIM: hsim.menus
- AAI HSIMD: hsimD.menus

  All the menu files are located under the `AAIM-<version>-`*`mmddyear`*`/menus` subdirectory.

4. Locate the menu file subdirectory.

   Locate the menu file in user's working environment. It can be either of the following directories:

   - `<user_working_dir>/menus`
   - `<user_home_dir>/menus`

5. Merge the menu files.

   The provided menu file can be merged with the menu file in the Cadence tree, as shown in the following example:

```
<CDS_install_dir>/tools/dfII/etc/tools/menus/simui.menus
```

6. Set up the executable.

   Set the Interface to point to the appropriate HSIM executable as shown in the following command syntax:

```
setenv HSIM_HOME <your_HSIM_installation_dir>
```

7. Obtain an Interface license.

   Obtaining the hmartld license daemon is required before conducting any interface flows. The license daemon resides in the HSIM release tree such as $HSIM_HOME/bin/hmartld. Without the Interface license daemon properly located, the following error message will be displayed in the Cadence central information window and the Interface flow will not proceed.

```
.......
Checking out Synopsys license ...
*Error* ipcWriteChild: Attempt to write to expired process -
ipc:-1
.....
```

AAIM Uninstallation

To uninstall any part of the AAIM software, delete any unwanted software package name in the nsdaAAIMPkgList statement as shown in the following syntax example:

```
nsdaAAIMPkgList='("hsim" "AANNI")
```

Note:

Using the syntax above, only AAI HSIM socket and AANNI are installed. AAI HSIMD and AAICoSim will not be installed.

Native Netlist Integration (AANNI)

Native Netlist Integration Installation & Setup

This section describes how to use the HSIM-Virtuoso Interface to integrate HSIM's simulation engine into the Virtuoso Analog Design Environment. The Native Netlist Integration software has a different integration approach than traditional approaches such as HSIM socket and HSIMD direct. The key features are described below.

Native Netlist Integration Features

Native Netlist Integration has the following features:

- HSIM parameters and command line options specification.
- Top-level netlist creation.
- HSIM simulation.
- Browse and view ASCII output files.
- Signal cross-probing and waveform display.
- Save/load state.
- Does not conduct design host netlisting.
- Does not require a special view and simInfo from device libraries.
- Does not require an additional Cadence license for operation.

Basic Native Netlist Integration Flow

Figure 39 displays the typical Native Netlist Integration process flow.

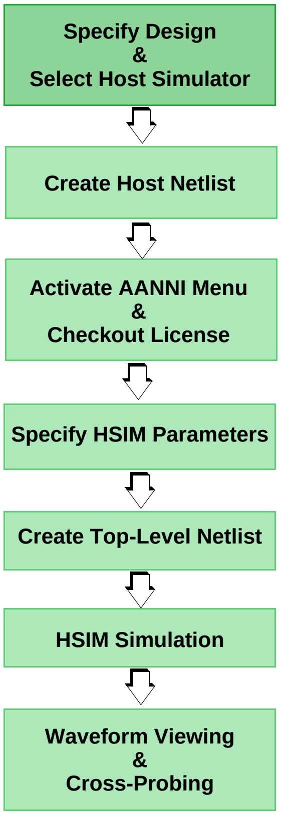


Figure 39 Process Flow

Porting the Existing Design

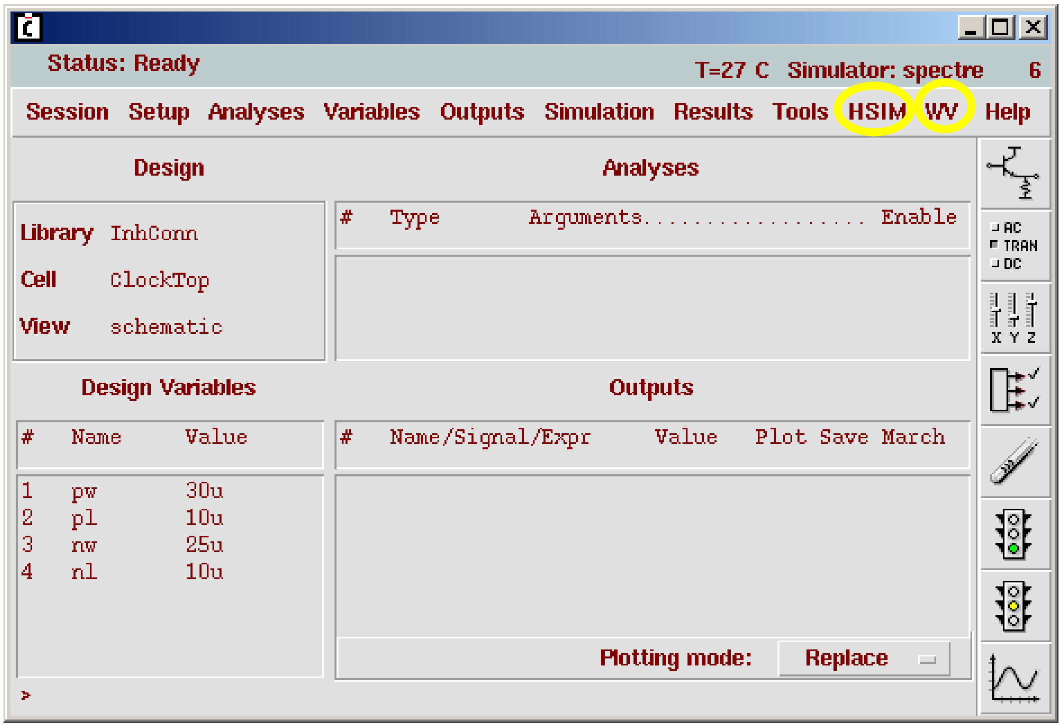
Since Native Netlist Integration does not conduct host netlist creation nor require special view and simInfo from the device libraries, there are no major porting issues for existing designs when migrating from traditional socket and direct integration approaches.

Native Netlist Integration Window and Pull-Down Menus

With Native Netlist Integration properly setup and installed in the Virtuoso Analog Design Environment window shown in Figure 40, the pull-down menu appears. It is located on the RIGHT hand side of the form. Also, if WV is also specified in the nsdaAAIMPkgList statement, there will also be a WV menu next to the menu header.

Caution!

> The window may need to be enlarged to view the menu if there are a large number of existing pull-down menus displayed from the local environment setup.



Design

Library InhConn
Cell ClockTop
View schematic

Analyses

Type Arguments.................. Enable

Design Variables

#	Name	Value
1	pw	30u
2	pl	10u
3	nw	25u
4	nl	10u

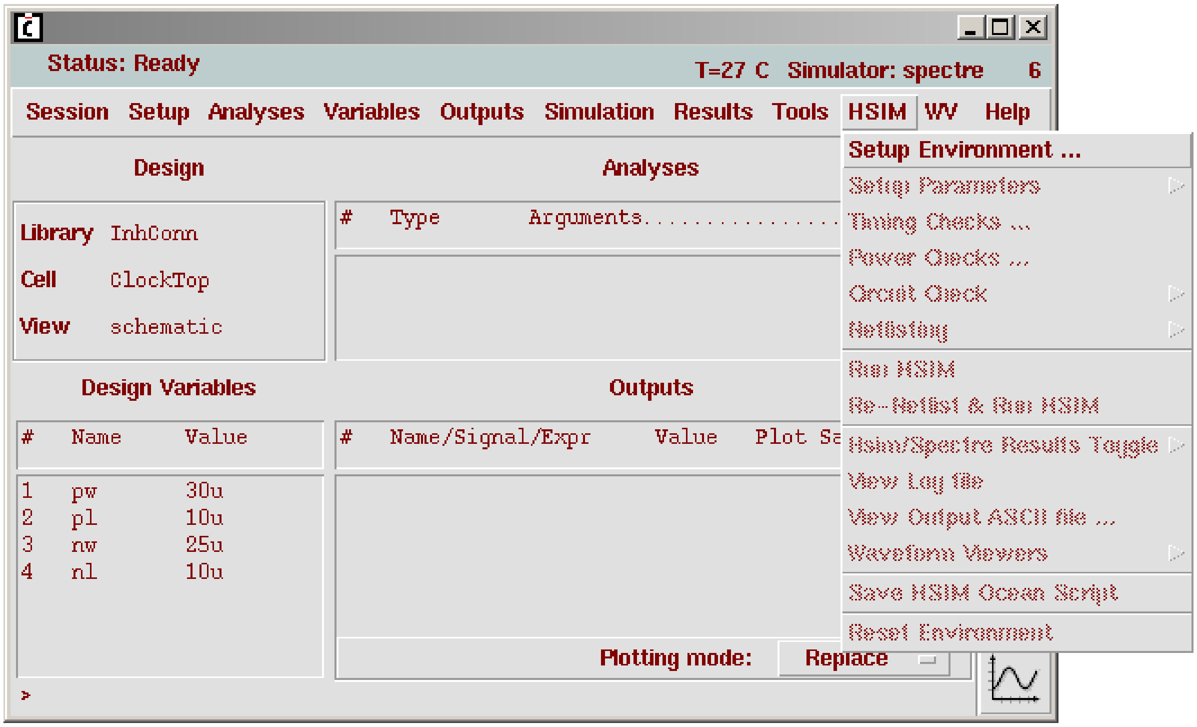
Outputs

Name/Signal/Expr Value Plot Save March

Plotting mode: Replace

Figure 40 HSIM-Virtuoso Design Environment Interface Window

Initially, all items in the menu are disabled and appear as grayed out text as shown in Figure 41.



Library InhConn
Cell ClockTop
View schematic

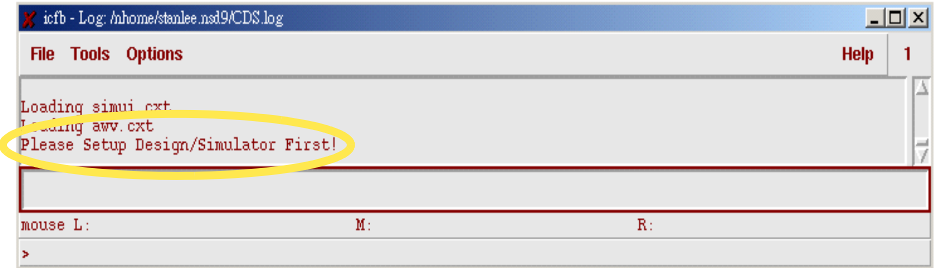
#	Type	Arguments. . . . .

#	Name	Value
1	pw	30u
2	pl	10u
3	nw	25u
4	nl	10u

#	Name/Signal/Expr	Value	Plot	Sa

Figure 41 HSIM-Virtuoso Design Environment Interface Window & Pull-down Menu

A design and designated host simulator such as spectre or hspiceS must be specified before the pull-down menu commands become active. If they are not specified, a warning message is displayed as shown in Figure 42.

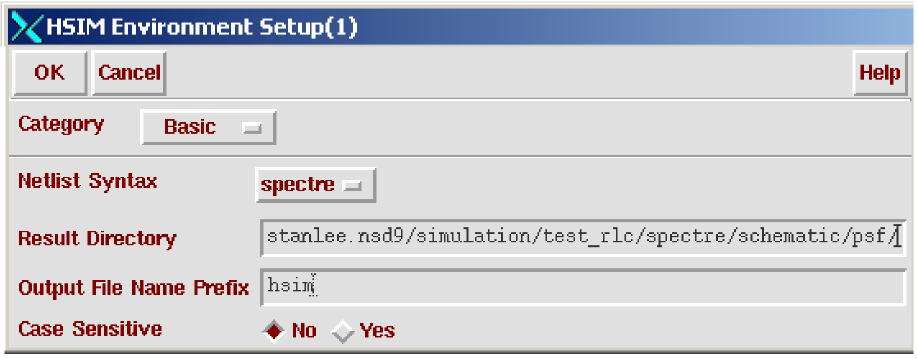


```
Loading simui.cxt
Loading awv.cxt
Please Setup Design/Simulator First!
```

Figure 42 Warning Message Window

Environment Setup

After specifying a design and simulator, select Setup Environment in the pull down menu. The Environment Setup(1) window containing the Environment Options form shown in Figure 43 will appear.



Category: Basic

Netlist Syntax: spectre

Result Directory: stanlee.nsd9/simulation/test_rlc/spectre/schematic/psf/

Output File Name Prefix: hsim

Case Sensitive: ● No ○ Yes

Figure 43 Setup Options Form Window

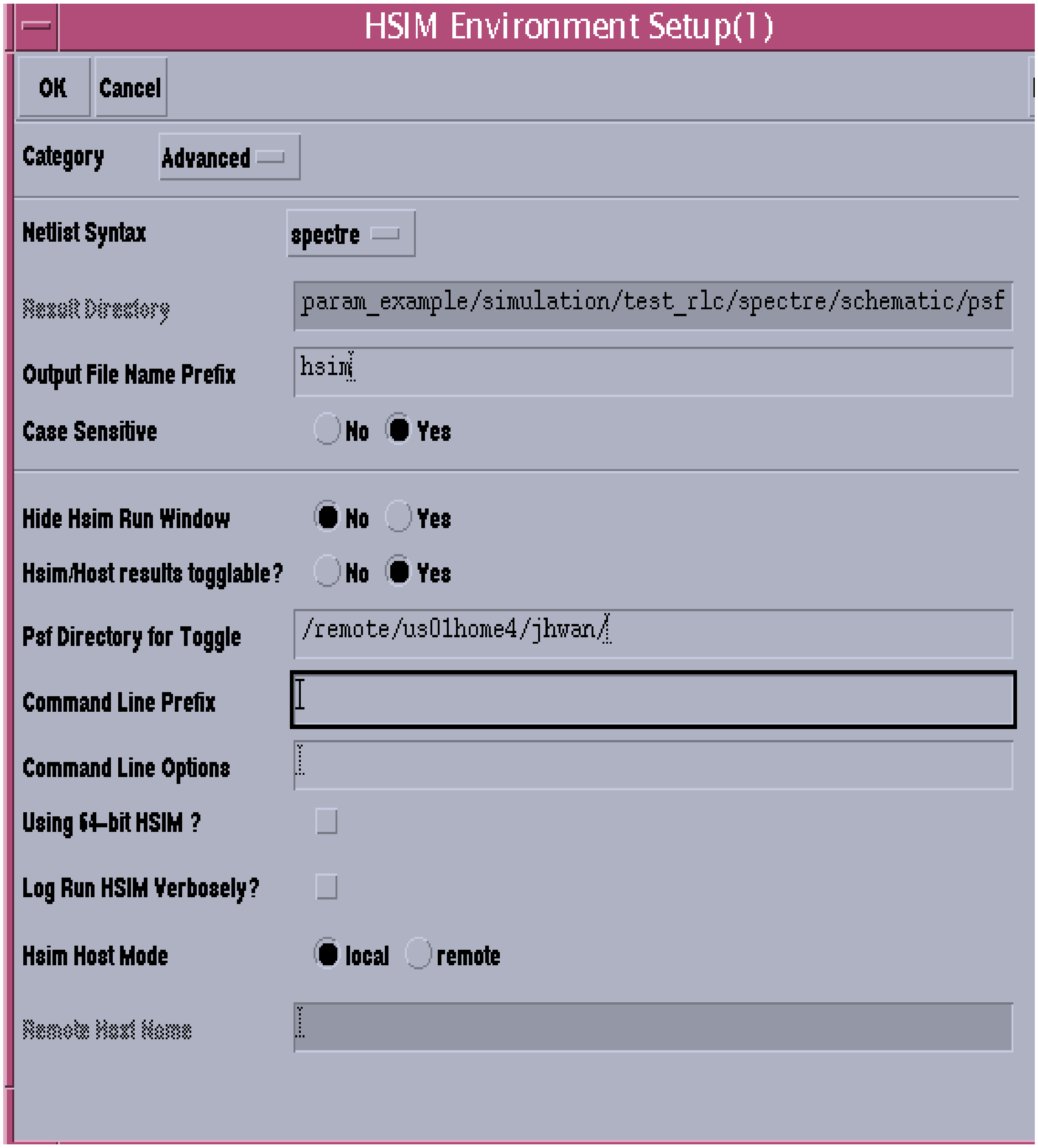
Basic Setup

The options displayed on the basic form include the following:

- [Basic]: If the [Basic] screen button is displayed, the setup form only displays the most frequently used buttons and entry fields.
- Netlist Syntax: Indicates what kind of netlist syntax the host netlister is going to generate.
- Result Directory: Allows user to specify directory for storing output data.
- Output File Name Prefix: Is equivalent to the -o option of HSIM,
- Case Sensitive: Is equivalent to the -case option of HSIM

Advanced Setup

To perform more advanced environment setups, click on the [Basic] screen button and select Advanced from the options displayed. The form enlarges and provides additional setup options as shown in Figure 44.



Category: Advanced

Netlist Syntax: spectre

Result Directory: param_example/simulation/test_rlc/spectre/schematic/psf

Output File Name Prefix: hsim

Case Sensitive: ○ No ● Yes

Hide Hsim Run Window: ● No ○ Yes

Hsim/Host results togglable? ○ No ● Yes

Psf Directory for Toggle: /remote/us01home4/jhwan/

Command Line Prefix:

Command Line Options:

Using 64-bit HSIM ? □

Log Run HSIM Verbosely? □

Hsim Host Mode: ● local ○ remote

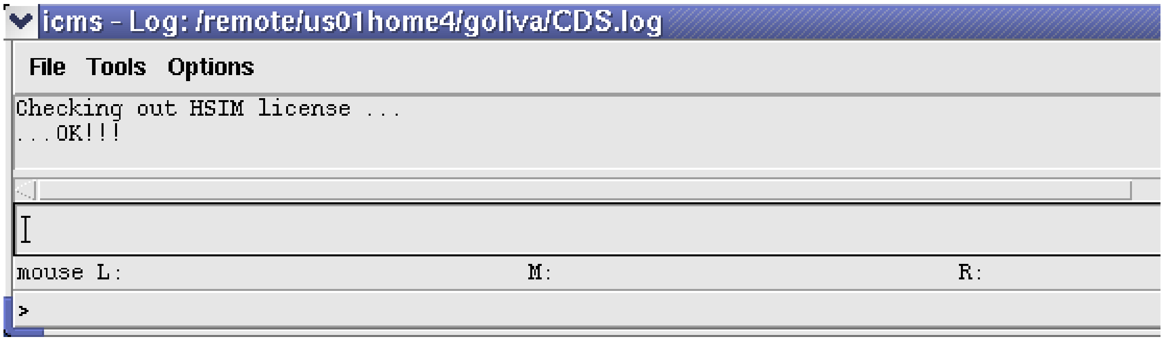
Remote Host Name:

Figure 44 Advanced Setup Options Form Window

The advanced options displayed on this form include the following:

- HSIM/Host results togglable?: The default is Yes, which allows AANNI to preserve a copy of the HSIM simulation result in case you would like to conduct HSIM-based or host simulator-based waveform probing/viewing. The default location for the preserved HSIM simulation copy is your home directory. You can change the location through the "Psf Directory for Toggle" entry field. Refer to Toggle between Spectre and HSIM Simulation Database for Waveform Probing on page 391 for more details.
- Command Line Prefix: Whatever specified in this field will be attached in the beginning of the HSIM command line.
- Command Line Options: Allows user to specify additional HSIM command line options if applicable.
- Using 64-bit HSIM?: If turned on, it is equivalent to set the following in your working environment: setenv HSIMPLUS_64 1.
- HSIM HOST MODE with local and remote radio buttons: The remote radio button permits AANNI users to run HSIM installed on a remote machine. To run HSIM through remote mode, select the remote radio button from HSIM HOST MODE and fill in the remote host name. Since the operation is done via the rsh Unix command, make sure that the .rhosts file is set up and the HSIM executable is in the search path on the remote machine.

When the [OK] screen button is selected, the Synopsys artistIF license is automatically checked out. If the license checkout is successful, all of the menu commands will be activated and a licence checkout confirmation will be displayed as shown in Figure 45.



```
Checking out HSIM license ...
...OK!!!
```

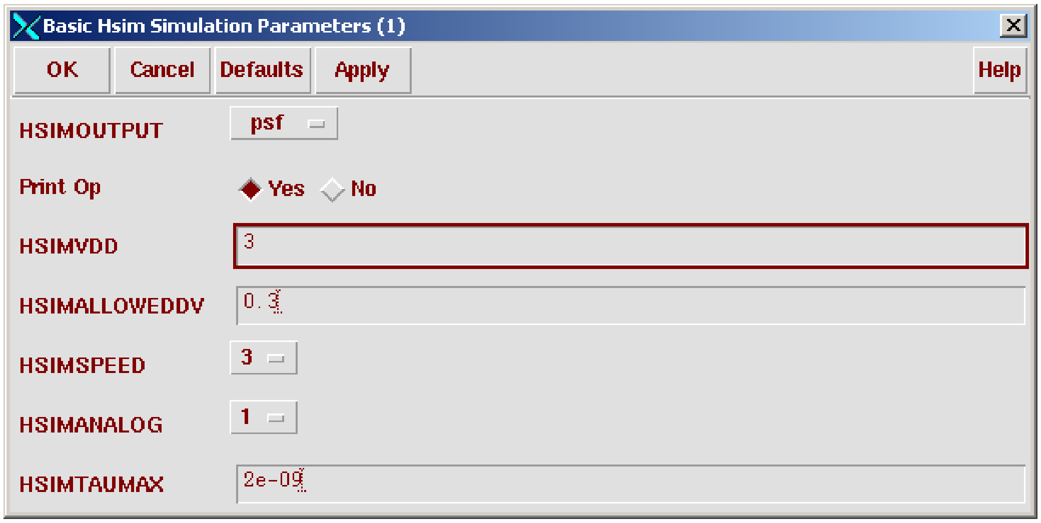
Figure 45 License Checkout Confirmation Window

Setup Parameters

Native Netlist Integration provides several forms to allow specification of HSIM parameter and command options. This works similarly with HSIM socket and HSIMD direct integrations.

- HSIM -> Setup Parameters -> Basic: Basic specifies parameters for HSIM commands including HSIMVDD, HSIMSPEED, HSIMOUTPUT, etc. Refer to the *HSIM Simulation Reference Manual: Chapter 6, Simulation Parameters* for details on HSIMVDD and HSIMSPEED. Refer to the *HSIM Simulation Reference Manual: Chapter 8, Simulation Output* for information on HSIMOUTPUT and other output formats.
- HSIM -> Setup Parameters -> Advanced: Advanced specifies less frequently used parameters. These parameters are classified into different categories based on their specific functionality.

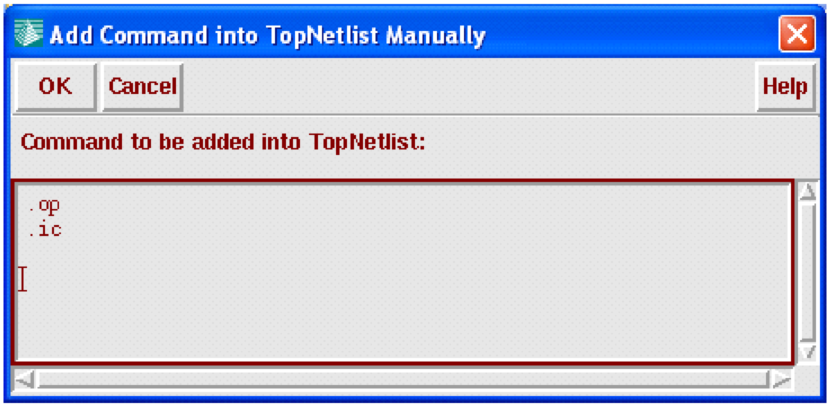
Basic and Advanced HSIM parameters in the form have the same default values as HSIM.



HSIMOUTPUT psf
Print Op ● Yes ○ No
HSIMVDD 3
HSIMALLOWEDDV 0.3
HSIMSPEED 3
HSIMANALOG 1
HSIMTAUMAX 2e-09

Figure 46 HSIM Basic Parameters Form

- HSIM -> Setup Parameters -> Manual
  Netlist options or statements can be manually keyed-in and then included in the top (final) netlist. All content specified in this form must follow the legal netlist syntax shown in Figure 47.


Add Command into TopNetlist Manually
OK
Cancel
Help
Command to be added into TopNetlist:

```
.op
.ic
```

Figure 47 Manual Command Entry Window

Netlisting

Native Netlist Integration netlisting provides the following functionality:

- Create Top Netlist
- Edit Top Netlist
- Create Top & Host Netlist
- View Host Netlist

Create Top Netlist

HSIM -> Netlisting -> Create Top Netlist

By default, Create Top Netlist only creates the top-level netlist content. If there is no existing host netlist, Native Netlist Integration automatically creates the host netlist before making the top-level netlist. The top-level netlist can then be created which includes the following:

- Host netlist: Automatically generated by the host netlister.
- HSIM parameters and options: HSIM Basic and Advanced parameters.
- Manually generated netlist statements or options: HSIM Manual parameters.

Edit Top Netlist

HSIM -> Netlisting -> Edit Top Netlist

Edit Top Netlist allows manual editing of the top-level netlist generated by Create Top Netlist with any text editor. To set up the editor for opening the top netlist, set and export the HSIM_FAVORITE_EDITOR shell environment variable as shown in the following syntax example:

```
export HSIM_FAVORITE_EDITOR=xedit
```

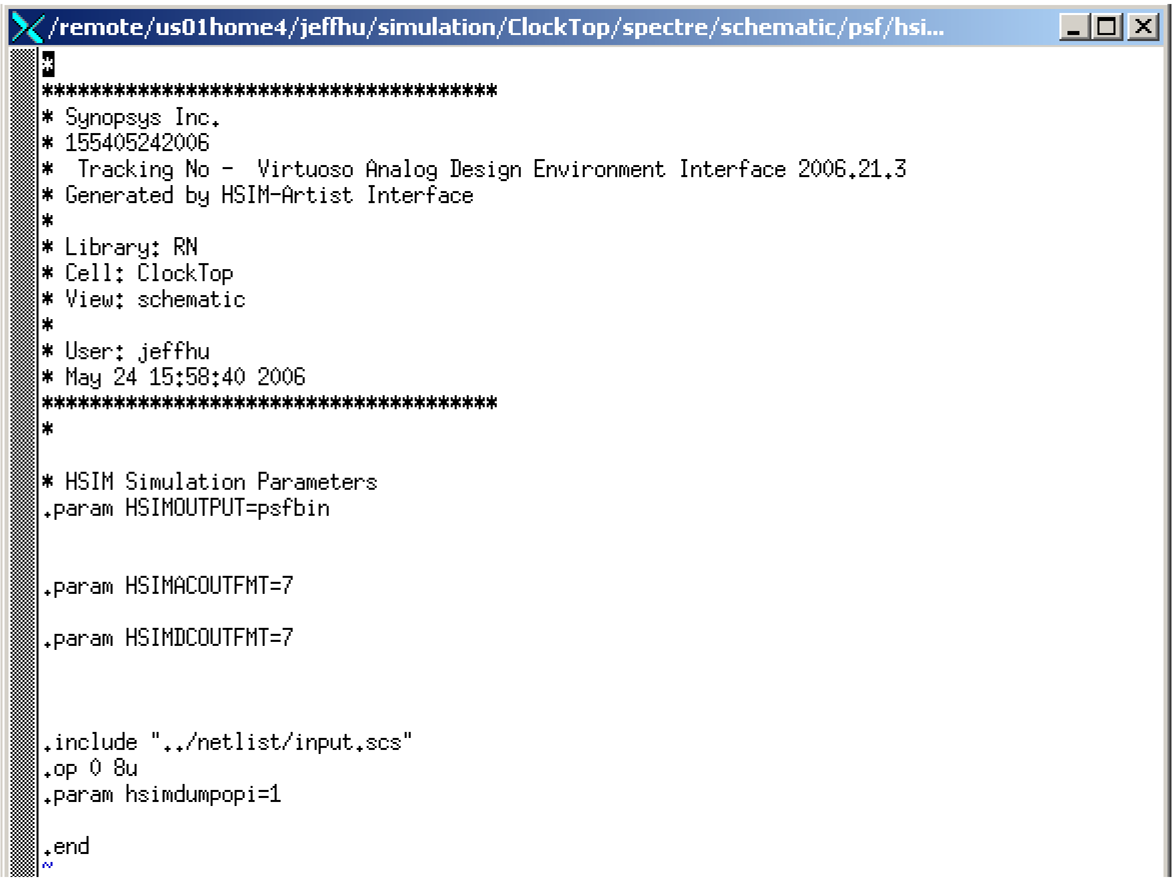
In this syntax example, xEdit is used as the is set as the default editor so that Native Netlist Integration reads in this environment variable and automatically set the editor accordingly in icfb or icms. Users can temporarily switch to and from various editors at any time entering the following command directly in the Command Interpreter Window (CIW).

```
editor=”commandToInvokeEditor”
```

To set the editor to vi, enter the following in the CIW window:

```
editor=”vi”
```

Figure 48 shows a typical top-level netlist.



/remote/us01home4/jeffhu/simulation/ClockTop/spectre/schematic/psf/hsi...

```
*
**************************************
* Synopsys Inc.
* 155405242006
*  Tracking No -  Virtuoso Analog Design Environment Interface 2006.21.3
* Generated by HSIM-Artist Interface
*
* Library: RN
* Cell: ClockTop
* View: schematic
*
* User: jeffhu
* May 24 15:58:40 2006
**************************************
*

* HSIM Simulation Parameters
.param HSIMOUTPUT=psfbin


.param HSIMACOUTFMT=7

.param HSIMDCOUTFMT=7



.include "../netlist/input.scs"
.op 0 8u
.param hsimdumpopi=1

.end
```

Figure 48   Typical Top-Level Netlist

Create Top & Host Netlist

HSIM -> Netlisting -> Create Top & Host Netlist

Create Top & Host Netlist creates the top-level netlist and forces the host simulator to update or regenerate the host netlist.

View Host Netlist

HSIM -> Netlisting -> View Host Netlist

View Host Netlist opens the host netlist file for viewing.

Run HSIM

HSIM -> Run HSIM

- HSIM -> Run HSIM runs HSIM simulation on the top-level netlist with command line options specified using the Setup Environment option form, when applicable.
- After running HSIM, all output files are stored in the directory specified under the Result Directory in the Setup Environment form.
- A Waveform window similar to the one shown in Figure 49 will automatically display for pre-selected signals.

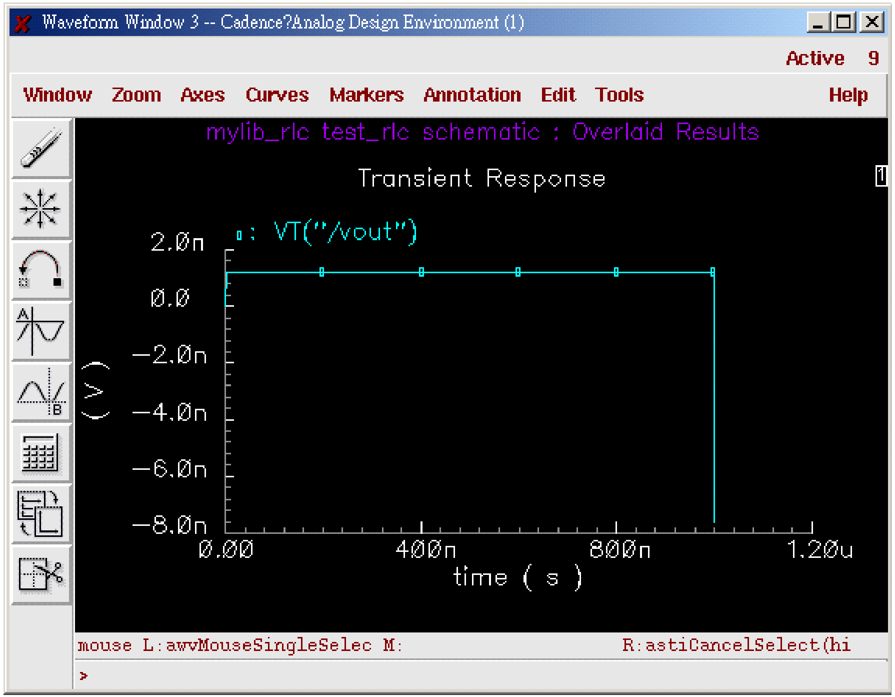
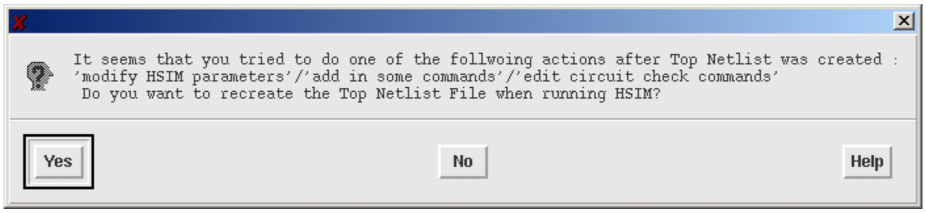


Figure 49 Waveform Window

- Cross probing of results is also supported.
- If parameter settings are modified and HSIM -> Run HSIM is run directly without updating the top netlist, Native Netlist Integration displays a confirmation dialog box asking whether the Top Netlist File should be recreated when running HSIM as shown in Figure 50. Use the [Yes] or [No] screen buttons to tell the system whether to update the Top Netlist File or not.



```
It seems that you tried to do one of the follwoing actions after Top Netlist was created :
'modify HSIM parameters'/'add in some commands'/'edit circuit check commands'
 Do you want to recreate the Top Netlist File when running HSIM?
```

Figure 50 Top Netlist File Regeneration Confirmation Window

Regenerate the Netlist and Run HSIM

HSIM -> Re-Netlist & Run HSIM

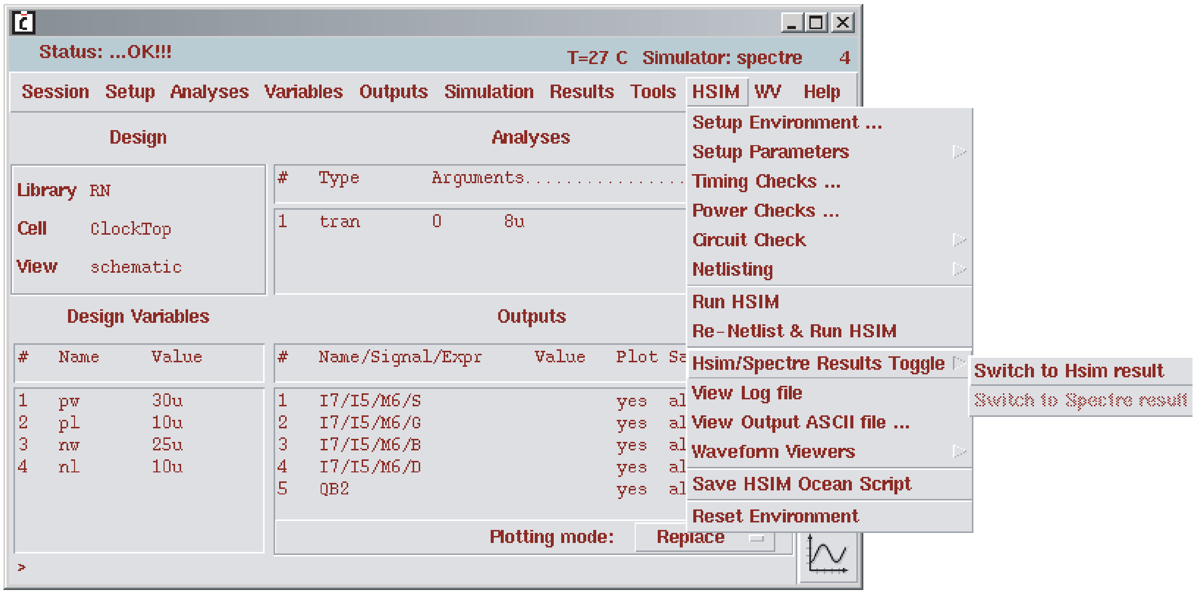
Re-Netlist & Run HSIM forces the host netlist or top netlist to be updated and/or regenerated and run HSIM.

Toggle between Spectre and HSIM Simulation Database for Waveform Probing

HSIM -> Hsim/Spectre Results Toggle

If you want to perform Spectre® simulation and HSIM simulation within the same Virtuoso Analog Design Environment session you can display the associated waveform through the standard plotting and cross-probing features by switching between the Spectre simulation results and the HSIM simulation results. After setting up the HSIM environment, the program points to the HSIM simulation database for waveform display by default. If you are interested in HSIM simulation only through the whole Virtuoso Analog Design Environment session you can ignore this feature.

The default setting for case sensitivity and database toggle can be specified in either a .cdsinit file or .aannienv file. The .cdsinit file can reside in your current working directory or your home directory. If you specify the setting in the .aannienv file, the interface program searches for it first in the current working directory and then in the home directory. If the setting resides in both of these files, the .aannienv setting has the higher priority (and in this case AAI also issues a warning message).



Library RN
Cell ClockTop
View schematic

#	Type	Arguments................
1	tran	0 8u

#	Name	Value
1	pw	30u
2	pl	10u
3	nw	25u
4	nl	10u

#	Name/Signal/Expr	Value	Plot	Sa
1	I7/I5/M6/S		yes	al
2	I7/I5/M6/G		yes	al
3	I7/I5/M6/B		yes	al
4	I7/I5/M6/D		yes	al
5	QB2		yes	al

>

Figure 51 HSIM-Spectre Results Toggle

There are two database toggle flows that are recommended to if you need to perform Spectre simulation as well as HSIM simulation:

Toggle Flow I:

Start a Virtuoso Analog Design Environment session and complete the following steps:

1. Setup -> Design
2. Setup -> Simulator -> Spectre
3. Setup -> Model Libraries…
4. Analyses -> Choose…
5. Outputs -> To Be Plotted…
6. Simulation->Netlist and Run (conduct Spectre simulation)
7. Results -> Direct Plot (plot Spectre simulation result)
8. HSIM -> Setup Environment
9. HSIM -> Re-Netlist & Run HSIM

10. Results -> Direct Plot (plot HSIM simulation result)
11. HSIM -> Hsim/Spectre Results Toggle -> Switch to Spectre result
12. Results -> Direct Plot (plot Spectre simulation result)
13. HSIM -> Hsim/Spectre Results Toggle -> Switch to Hsim result
14. Results -> Direct Plot (plot HSIM simulation result)

Toggle Flow II:

Start a Virtuoso Analog Design Environment session and complete the following steps:

1. Setup -> Design
2. Setup -> Simulator -> Spectre
3. HSIM -> Setup Environment
4. Setup -> Model Libraries…
5. Analyses -> Choose…
6. Outputs -> To Be Plotted…
7. HSIM -> Re-Netlist & Run HSIM
8. Results -> Direct Plot (plot HSIM simulation result)
9. HSIM -> Reset Environment
10. Simulation -> Netlist and Run (conduct Spectre simulation)
11. Results -> Direct Plot (plot Spectre simulation result)
12. HSIM -> Setup Environment
13. HSIM -> Re-Netlist & Run HSIM
14. HSIM -> Hsim/Spectre Results Toggle -> Switch to Spectre result
15. Results -> Direct Plot (plot Spectre simulation result)
16. HSIM -> Hsim/Spectre Results Toggle -> Switch to Hsim result
17. Results -> Direct Plot (plot HSIM simulation result)

Note:

> The above recommended flows assume that Spectre simulation and HSIM simulation are each conducted once respectively. If you intend to conduct more than one Spectre run within the same HSIM-Virtuoso session you must to first select HSIM -> Reset Environment immediately before running

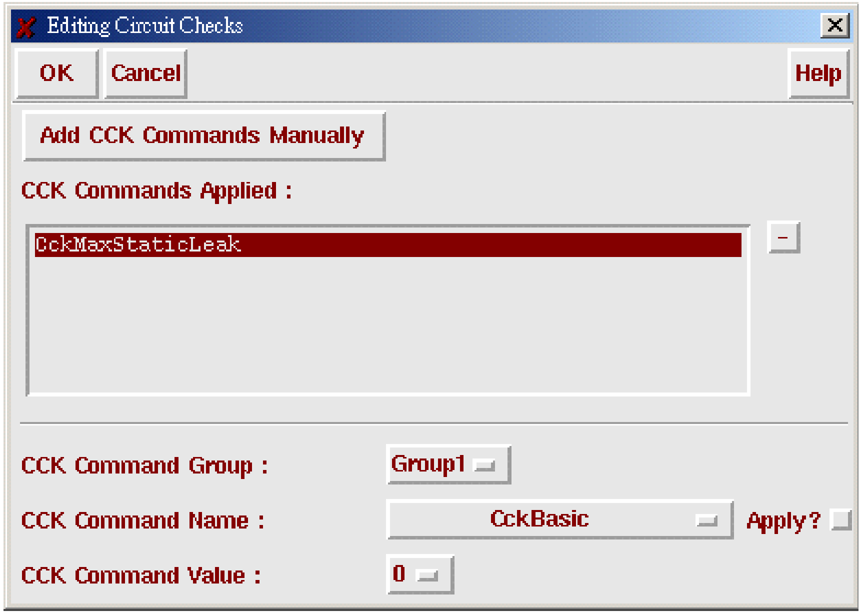
the Spectre simulation. This ensures that the data access function points to the correct simulator. Otherwise, the plotting may not be able to map to the appropriate waveform database.

CircuitCheck in the HSIM-Virtuoso Interface Environment

HSIM -> CircuitCheck

The HSIM-Virtuoso Interface integrates the CircuitCheck (CCK) features into the Virtuoso Analog Design Environment. Specific CCK commands or the entire CCK command file can be specified using the Native Netlist Integration GUI as shown in Figure 52. The interface combines CCK-related information into the top-level netlist for HSIM to analyze.

CCK usage details are described in HSIM-Virtuoso CircuitCheck Integration on page 405. Detailed CCK command syntax and usage is contained in Chapter 8, CircuitCheck.



CCK Commands Applied :

CckMaxStaticLeak

CCK Command Group : Group1

CCK Command Name : CckBasic Apply? □

CCK Command Value : 0

Figure 52 Editing CircuitChecks Window

View Log File

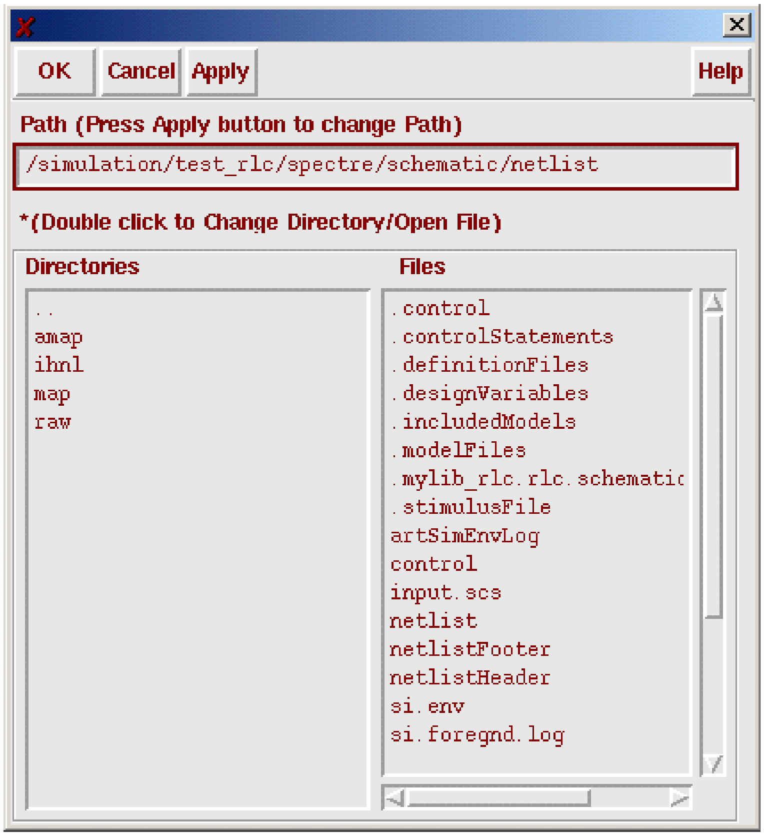
HSIM -> View Log File

View Log File displays the HSIM log file.

View Output ASCII Files

HSIM -> View Output ASCII File

View Output ASCII File brings up the file browser containing all the ASCII results stored in the result directory as shown in Figure 53.



Path (Press Apply button to change Path)

/simulation/test_rlc/spectre/schematic/netlist

*(Double click to Change Directory/Open File)

Directories

```
..
amap
ihnl
map
raw
```

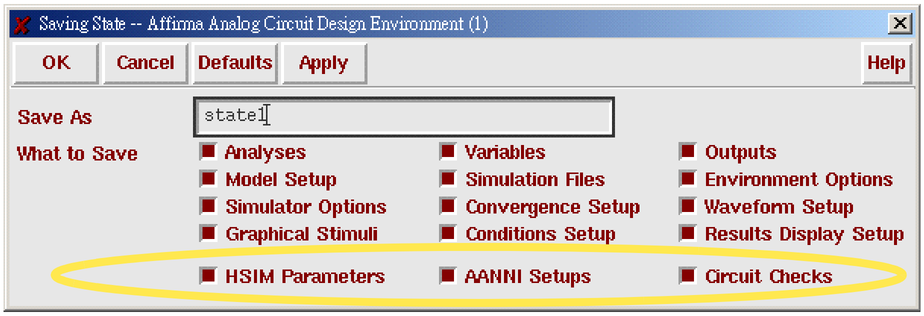
Files

```
.control
.controlStatements
.definitionFiles
.designVariables
.includedModels
.modelFiles
.mylib_rlc.rlc.schematic
.stimulusFile
artSimEnvLog
control
input.scs
netlist
netlistFooter
netlistHeader
si.env
si.foregnd.log
```

Figure 53 File Browser Window

Save/Load States

Native Netlist Integration supports the Virtuoso Analog Design Environment saving and loading states feature that contains HSIM Parameters, AANNI Setups, and CircuitChecks options. These options are added to the Virtuoso Analog Design Environment Saving State window as shown in Figure 54.


Saving State -- Affirma Analog Circuit Design Environment (1)
OK
Cancel
Defaults
Apply
Help

Save As: state1

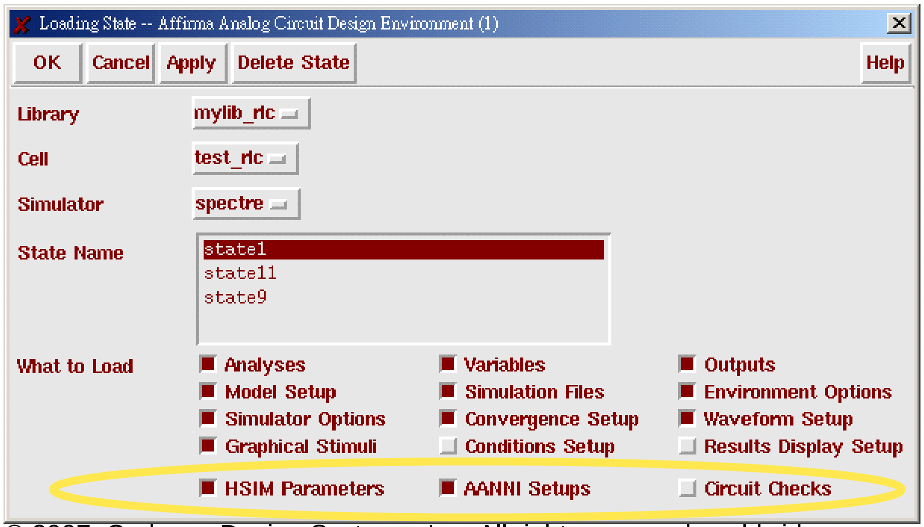
What to Save:

- ☑ Analyses
- ☑ Model Setup
- ☑ Simulator Options
- ☑ Graphical Stimuli
- ☑ Variables
- ☑ Simulation Files
- ☑ Convergence Setup
- ☑ Conditions Setup
- ☑ Outputs
- ☑ Environment Options
- ☑ Waveform Setup
- ☑ Results Display Setup
- ☑ HSIM Parameters
- ☑ AANNI Setups
- ☑ Circuit Checks

Figure 54 Saving State Form Window

These options have the following functions:

- HSIM Parameters: HSIM Parameters is used to save or load states specified through Setup Parameters->Basic.
- AANNI Setups: This option is used for miscellaneous related parameters to the Native Netlist Integration flow.
- CircuitChecks: CircuitChecks is primarily used for commands specified through the Editing CircuitCheck form.
- Saved states will be stored in files under the directory specified in the Session-> Options…->State Save Directory field.
- Once states for Native Netlist Integration are saved, they can be loaded using the Session->Load States… form as shown in Figure 55.


Loading State -- Affirma Analog Circuit Design Environment (1)
OK
Cancel
Apply
Delete State
Help

Library: mylib_rlc

Cell: test_rlc

Simulator: spectre

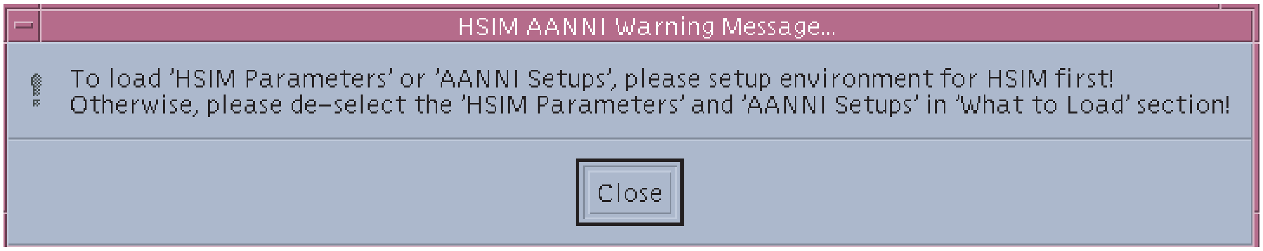
State Name: state1, state11, state9

What to Load:
- ☒ Analyses
- ☒ Model Setup
- ☒ Simulator Options
- ☒ Graphical Stimuli
- ☒ Variables
- ☒ Simulation Files
- ☒ Convergence Setup
- □ Conditions Setup
- ☒ Outputs
- ☒ Environment Options
- ☒ Waveform Setup
- □ Results Display Setup
- ☒ HSIM Parameters
- ☒ AANNI Setups
- □ Circuit Checks

Figure 55 Loading State Form Window

Note:

Before loading Native Netlist Integration states, the HSIM -> Setup Environment must first be activated or a warning message is displayed as shown in Figure 56. Loading save state without first activating Native Netlist Integration results in missing parameters settings.

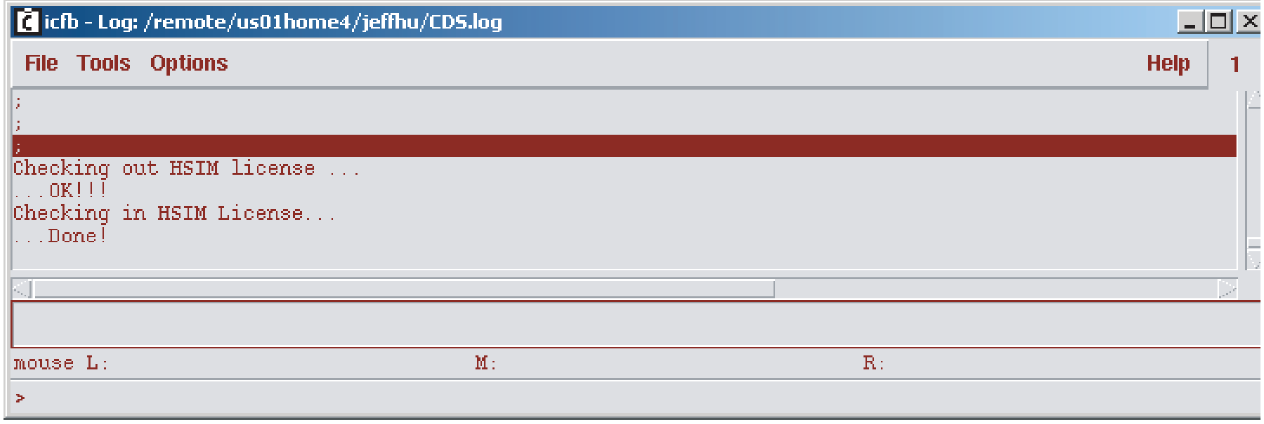


To load 'HSIM Parameters' or 'AANNI Setups', please setup environment for HSIM first!
Otherwise, please de-select the 'HSIM Parameters' and 'AANNI Setups' in 'What to Load' section!

Figure 56 HSIM Native Netlist Integration Warning Message Window

Check in Synopsys License

When the Virtuoso Analog Design Environment session is terminated, the artistIF license is automatically checked in and a message is generated Command Interpreter Window as shown in Figure 57.



```
;
;
;
Checking out HSIM license ...
...OK!!!
Checking in HSIM License...
...Done!
```

Figure 57 Log Window with Synopsys Licence Check Confirmation

Note:

The HSIM -> Reset Environment command is also located under the HSIM pull-down menu for the following reasons:

- The artistIF license is forced to be checked in so that even a current session is active, the commands under the HSIM pull down menu will be inactivated and the released artistIF license can be used by other users.
- Native Netlist Integration releases ownership of the data access function back to the host simulator. This is required to perform special features such as signal cross probing. It is possible for users to switch back-and-forth between the host simulator and HSIM within the same Virtuoso Analog Design Environment session to conduct simulation and then cross probing.

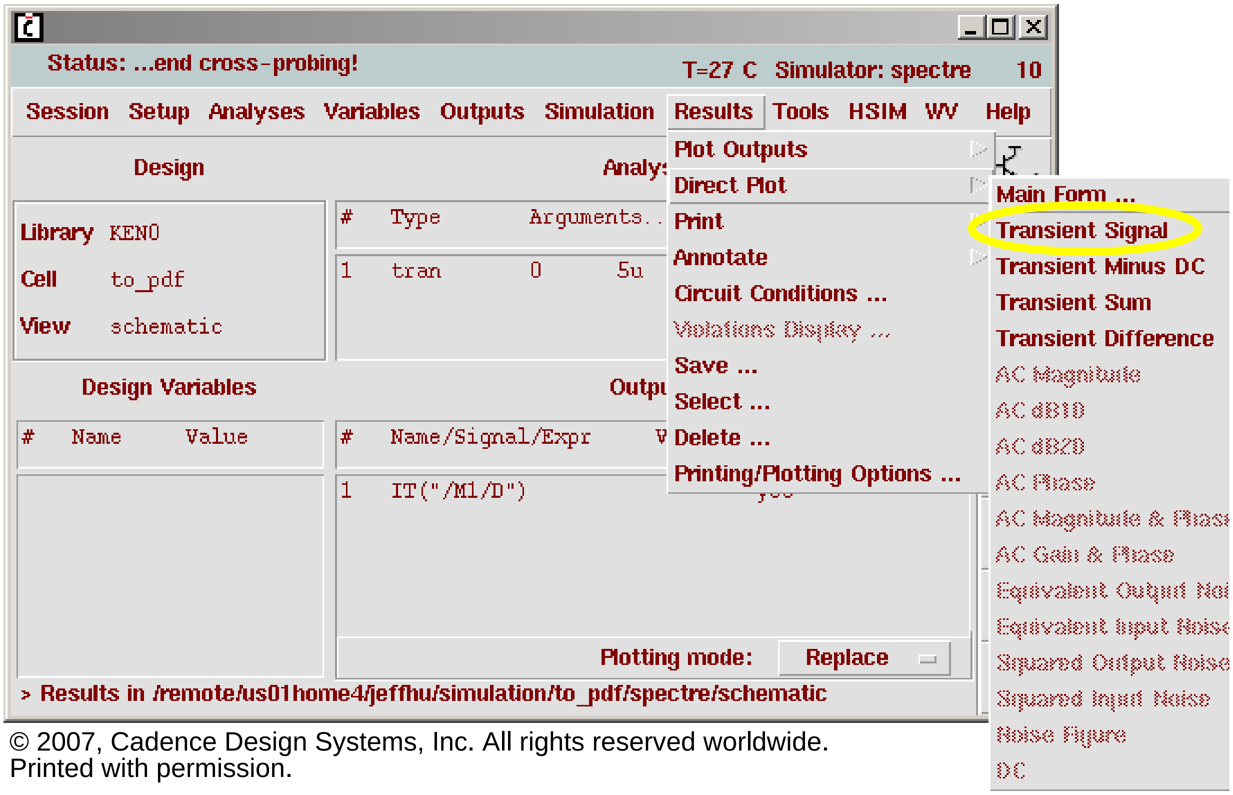
Caution!

It is strongly recommended that users issue the HSIM -> Reset Environment command before switching back to the host simulator to eliminate associated host simulator signal cross-probing failures.

Cadence Cross-probing

Native Netlist Integration works with Cadence® cross-probing functionality with the following steps:

1. After simulation has completed, select Results > Direct Plot > Transient Signal, as shown in Figure 58.



Status: ...end cross-probing!

T=27 C Simulator: spectre 10

Library KEN0

Cell to_pdf

View schematic

#	Type	Arguments..
1	tran	0 5u

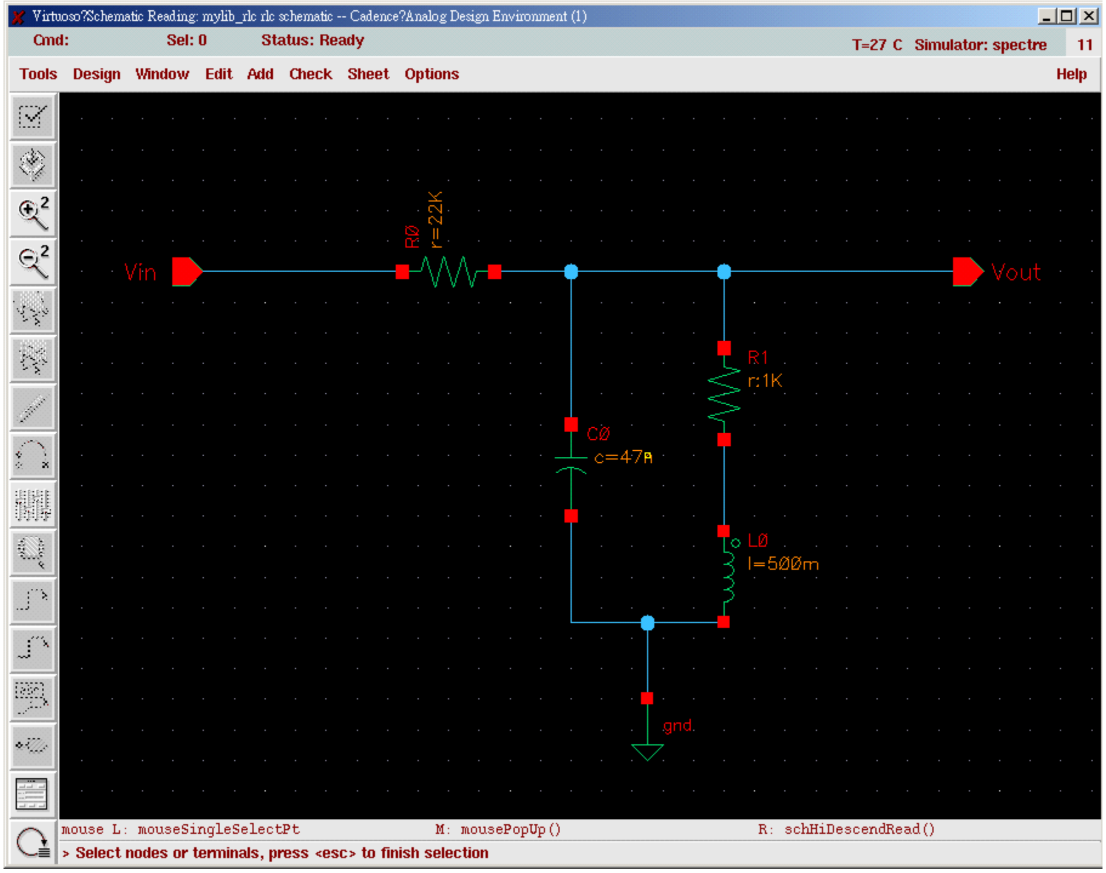
#	Name/Signal/Expr
1	IT("/M1/D")

Plotting mode: Replace

> Results in /remote/us01home4/jeffhu/simulation/to_pdf/spectre/schematic

Figure 58 Virtuoso Analog Design Environment Window with the Results->Direct Plot->Transient Signal Pull-down Windows

When Transient Signal is selected, the Virtuoso Schematic Editor window shown in Figure 59 is automatically displayed.

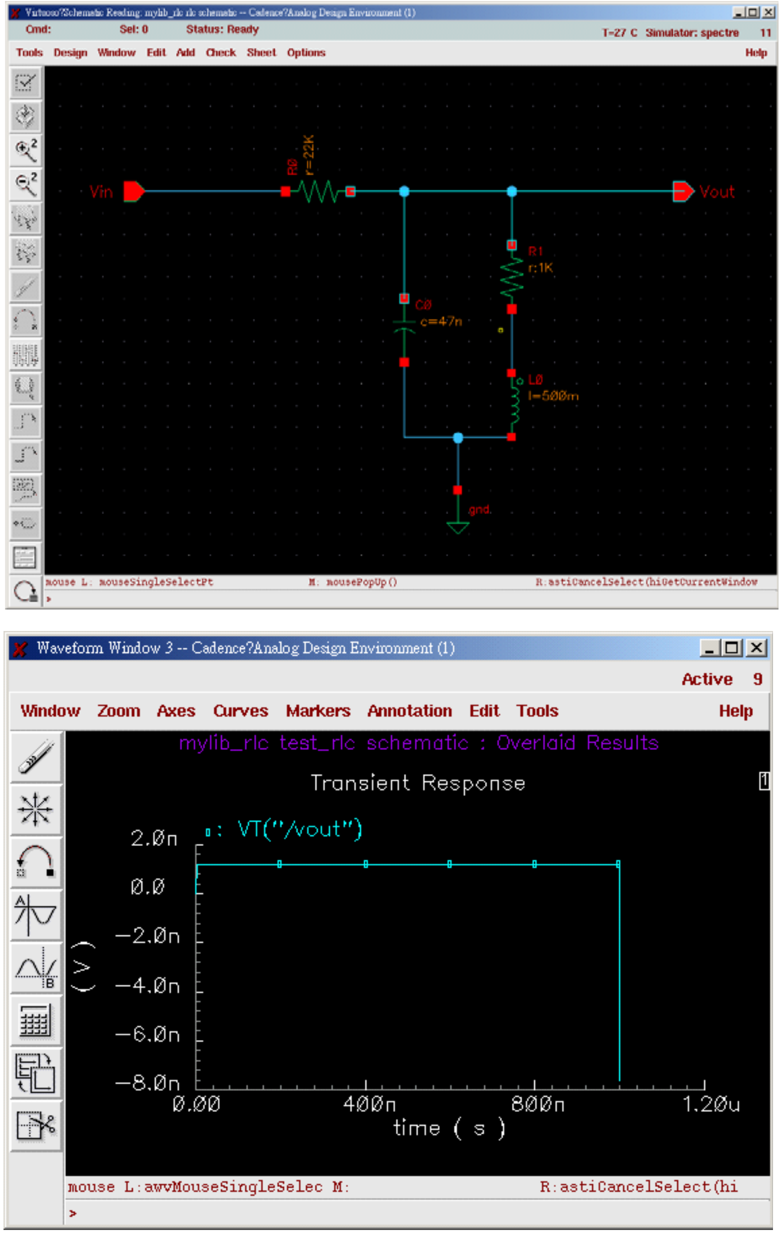


```
mouse L: mouseSingleSelectPt          M: mousePopUp()          R: schHiDescendRead()
> Select nodes or terminals, press <esc> to finish selection
```

Figure 59 Virtuoso Schematic Editor Window

2. Go to the desired level of design by using Design->Hierarchy->Descend Edit (or Read) in the Virtuoso Schematic Editor window as shown in Figure 59.

3. Select the signals to probe by LEFT clicking with the mouse. Signals can be either node voltage or instance current. Press [Esc] to terminate the signal selection process. The corresponding signal wave form will be displayed on the Waveform window as shown in Figure 60.



```
mouse L: mouseSingleSelectPt          M: mousePopUp()          R:astiCancelSelect(hiGetCurrentWindow
>
```

```
mouse L:awvMouseSingleSelec M:                 R:astiCancelSelect(hi
>
```

Figure 60 Virtuoso Schematic Editor Window & Corresponding Waveform Window

WaveView Analyzer Cross-probing

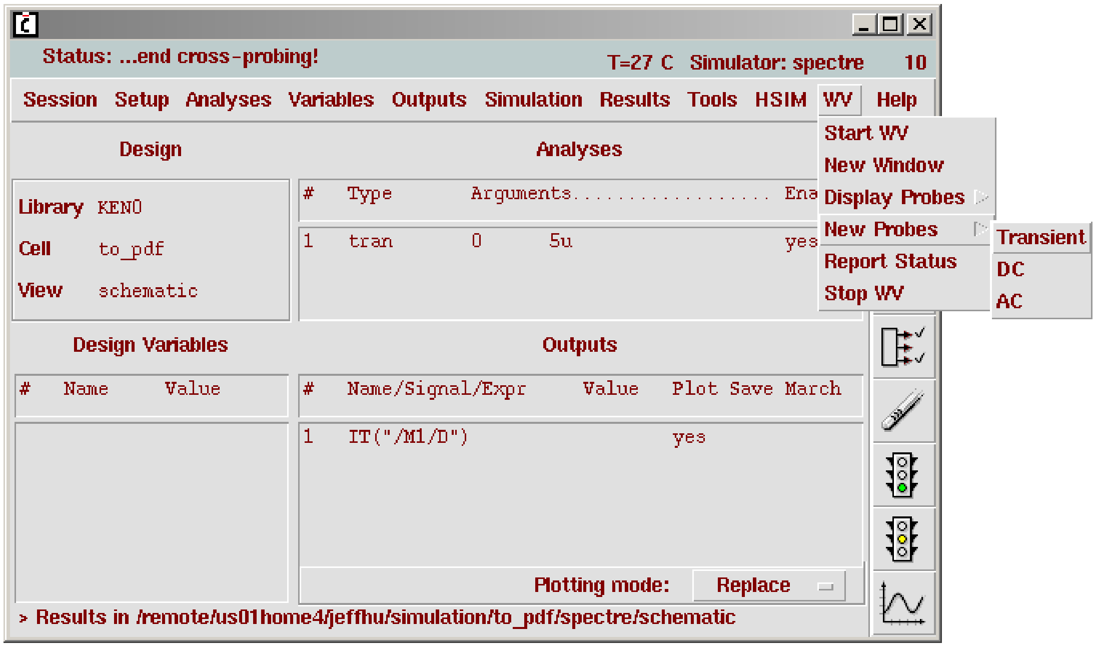
In addition to supporting the Cadence Waveform viewer, Native Netlist Integration also supports cross-probing functions with Sandwork's WaveView Analyzer (WV). Currently, WaveView Analyzer only partly supports hierarchical cross-probing.

Note:

Cross-hierarchy signal probing is not supported.

To use WaveView Analyzer for cross probing, use the following steps.

1. Select WV->start WV.
2. In the Virtuoso Schematic Editor window, select the desired circuit hierarchy level.
3. Select WV->New probe->Transient, as shown in Figure 61.



Status: ...end cross-probing! T=27 C Simulator: spectre 10

Design

Library KENO
Cell to_pdf
View schematic

Analyses

#	Type	Arguments.................	Ena
1	tran	0 5u	yes

Design Variables

#	Name	Value

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	IT("/M1/D")		yes		

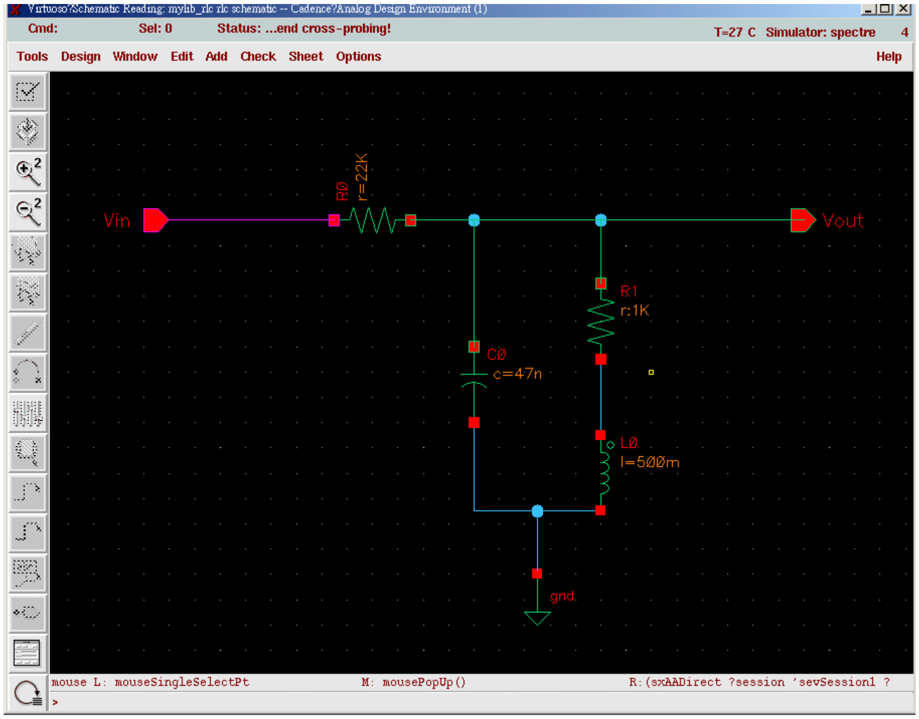
Plotting mode: Replace

> Results in /remote/us01home4/jeffhu/simulation/to_pdf/spectre/schematic

© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 61 Cross-probing with WaveView Analyzer

4. Select the signal to be plotted by clicking VIN and then press Escape to display the waveform as shown in Figure 62.


Virtuoso?Schematic Reading: mylib_rlc rlc schematic -- Cadence?Analog Design Environment (1)
Cmd:
Sel: 0
Status: ...end cross-probing!
T=27 C
Simulator: spectre
4
Tools
Design
Window
Edit
Add
Check
Sheet
Options
Help
Vin
R0
r=22K
Vout
R1
r:1K
C0
c=47n
L0
l=500m
gnd
mouse L: mouseSingleSelectPt
M: mousePopUp()
R:(sxAADirect ?session 'sevSession1 ?
>

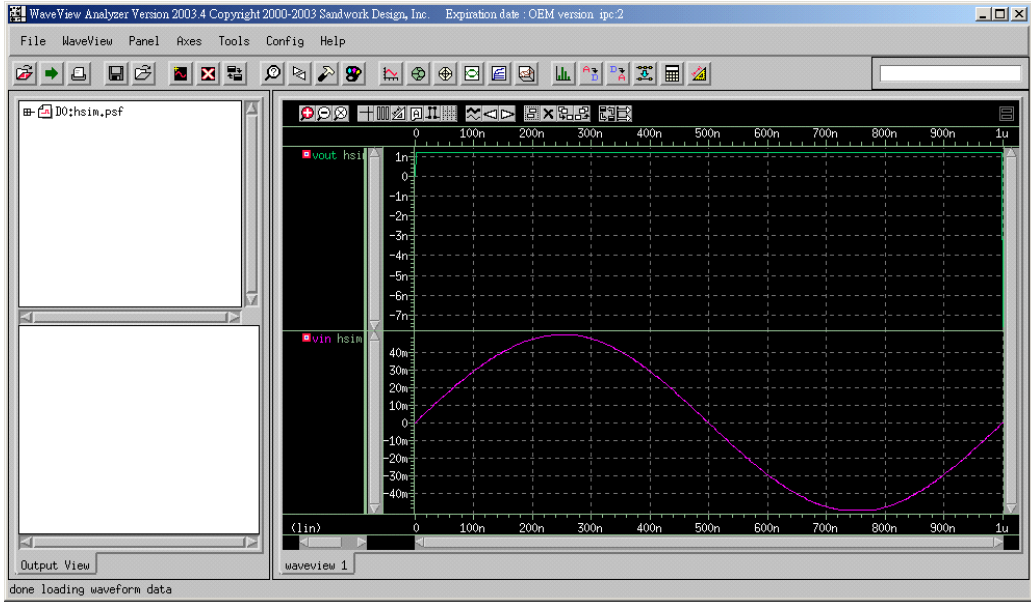


Figure 62 Virtuoso Schematic Editor Window Showing a Selected Signal Plotted in WaveView Analyzer

5. Repeat Step 1 through Step 4 in this section for additional signals.

CoSim (AACoSim) Integration

The purpose of CoSim integration is to integrate the Synopsys co-simulation technology into Cadence Virtuoso Analog Design Environment, the integration approach is very similar to Native Netlist Integration (AANNI), which utilizes the host simulator (such as spectreVerilog or hspiceSVerilog) to generate the host netlist (digital as well as analog parts), then make the top-level netlist with the cosim configuration file to conduct HSIM Verilog co-simulation. CoSim shares the same installation procedures with Native Netlist Integration with additional UNIX environment setup.

Note:

During the simulation cycle for CoSim (AACoSim) the hsim-cosim license is required, in addition to the artistIF license.

UNIX Setup

To set up CoSim for the Unix environment, perform the following tasks.

1. Set up the search path for Verilog simulator executable.

```
set path=(/usr/local/vendors/cadence/ldv40/tools/bin $path)
```

2. Set up $LD_LIBRARY_PATH

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:
<vpi_library_location>
```

Note:

The vpi library is normally located in the HSIM release tree; e.g., “/version/06-23-2004/hsimplus/platform/sunos_57/bin”) The vpi library is required to perform the co-simulation

CoSim Installation

To install CoSim, you must first receive the AAIM package, then set the HSIM_HOME and HSIM_ARTISTIF environment variables to where HSIM and AAIM packages are located, respectively.

Next, perform the following steps:

1. Copy $HSIM_ARTISTIF/menus/spectreVerilog.menus to either user's $HOME/menus/ or <working_directory>/menus directory
2. Use the following two lines in user's .cdsinit file to invoke CoSim :

```
nsdaAAIMPkgList='("AACoSim")
load("<AAI_installation_directory>/install/
nsdaAAIMInvoker.il")
```

Basic CoSim Flow

Here is the basic CoSim flow:

1. User starts icms or icfb session, brings up the Virtuoso Analog Design Environment window.
2. Select spectreVerilog as the simulator, specify library and design.
3. HSIM pull-down menu shows on the RIGHT hand side of the Virtuoso window.
4. Conduct spectreVerilog netlisting, generate digital and analog host netlist.
5. Select HSIM -> Setup Environment, [OK] to check out HSIM artistIF license and activate all the CoSim commands under HSIM pull-down menu.
6. Similar to Native Netlist Integration, use HSIM -> Setup Parameters to specify desired HSIM parameters.
7. Select HSIM -> Netlisting -> Create Top Netlist to generate the top-level netlist (cosim.scs).
8. Select HSIM -> Run HSIM CoSim to start the co-simulation.

HSIM-Virtuoso CircuitCheck Integration

Native Netlist CircuitCheck

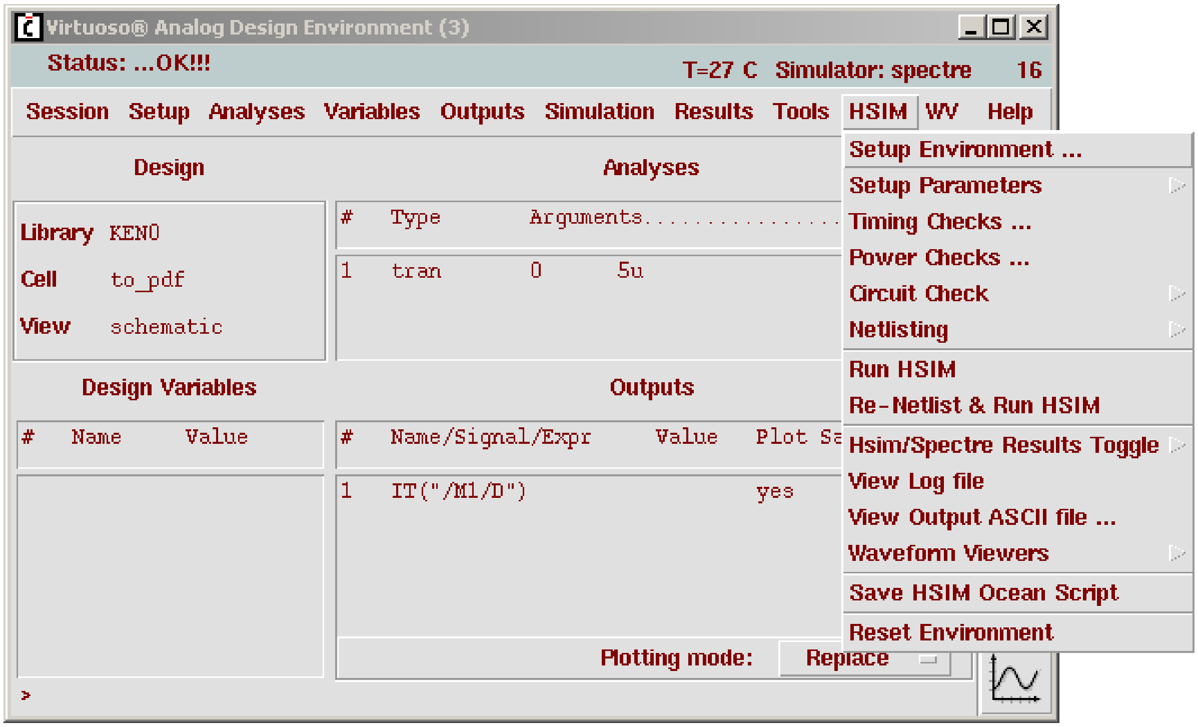
Native Netlist CircuitCheck is the integration interface between Virtuoso and the CCK commands built into HSIM. This approach and the GUI interface provide an easy to use mechanism for setting up all kinds of CCK commands to perform various circuit checking. For detailed usage of each CCK command and their option settings, please refer to Chapter 8, CircuitCheck.

Note:

> Although Native Netlist Integration does not require a CCK license, a separate CCK license is required in addition to the HSIM license during the simulation cycle. This allows the CCK commands to be set and edited however, these commands will not take effect when HSIM is run unless a CCK license is available.

Perform the following steps to open the CCK GUI form.

1. Check out an artistIF license from the Virtuoso Analog Design Environment window shown in Figure 63 as follows:
2. Choose HSIM->Setup Environment.



Status: ...OK!!! T=27 C Simulator: spectre 16

Library KEN0
Cell to_pdf
View schematic

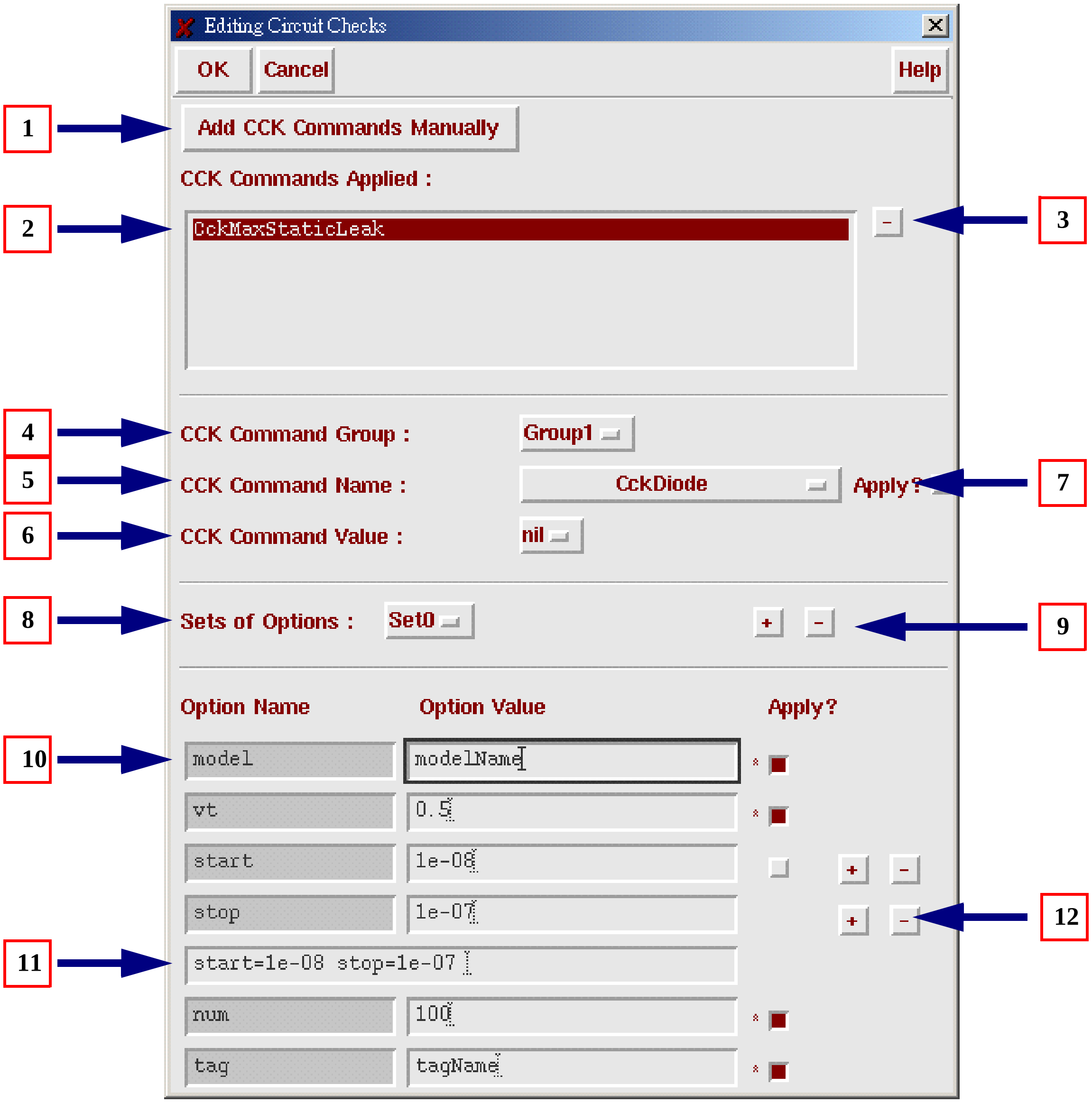
#	Type	Arguments...............
1	tran	0 5u

#	Name	Value

#	Name/Signal/Expr	Value	Plot Sa
1	IT("/M1/D")		yes

Figure 63 Virtuoso Analog Design Environment Window

3. Choose HSIM->CircuitCheck to open the Editing CircuitChecks window shown in Figure 64.



CCK Commands Applied : CckMaxStaticLeak

CCK Command Group : Group1

CCK Command Name : CckDiode Apply?

CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	■
vt	0.5	■
start	1e-08	□
stop	1e-07	
start=1e-08 stop=1e-07		
num	100	■
tag	tagName	■

Figure 64 Editing CircuitChecks

The following are descriptions of the Editing CircuitChecks window features shown in Figure 64.

1 [Add{{CCK{{Commands{{Manually]

Most CCK commands can be covered and selected through the cyclic selection on the form. New commands or those with complex parameter configurations can also be entered manually. Figure 65 displays the window used to manually enter commands.

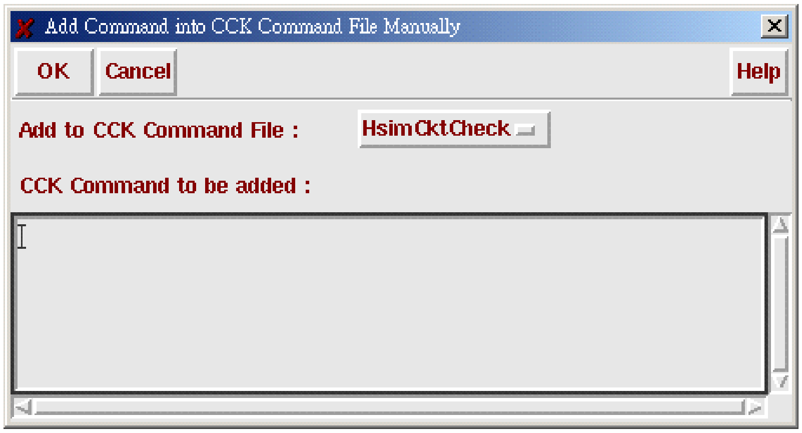

Add Command into CCK Command File Manually
OK
Cancel
Help
Add to CCK Command File :
HsimCktCheck
CCK Command to be added :

Figure 65 Add Command into CCK Command File Manually Window

Commands added using this window are included in the file generated via the Edit CircuitChecks form as described in the following sections.

2 CCK Command Applied

CCK Command Applied shows the list of CCK commands selected for application. To apply a command, press any associated Apply? button on the form.

The name of the selected command is listed in the CCK Command Applied: list box of the Editing CircuitChecks window shown in Figure 64 on page 407. Only the commands listed in the list box will be output into a CCK command file. These will then be included in the HSIM simulation netlist.

3 [{-{]

The [{-{] screen button to the RIGHT of the CCK Command Applied list box is used to deselect an applied command. To deselect a command, select and highlight the command from the list of commands in the CCK Command Applied: list box and press the [{-{] screen button.

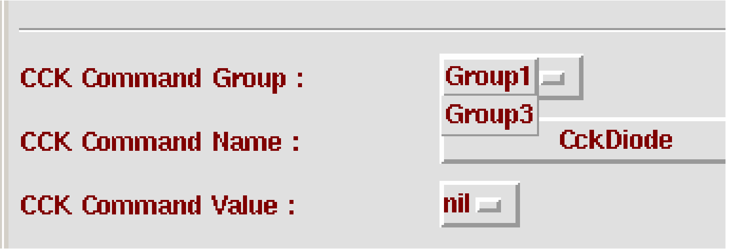
4 CCK Command Groups

Chapter 8, CircuitCheck, describes all CCK commands, which are classified into 3 groups having the following classifications:

- Group 1
- Group 2
- Group 3

Note:

> There is no Group 2 displayed in the drop down list shown in Figure 66 since Group 2 commands are mainly used in the interactive mode.



CCK Command Group : Group1

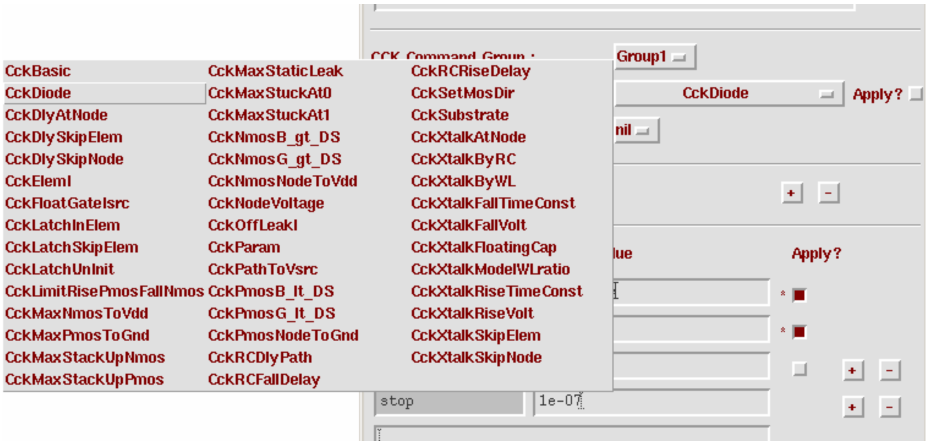
CCK Command Name : CckDiode

CCK Command Value : nil

Figure 66 CCK Command Group Drop-Down List

5 CCK Command Name

This drop down menu accommodates all CCK commands within the selected command group; for example, Group 1 commands are shown in Figure 67.



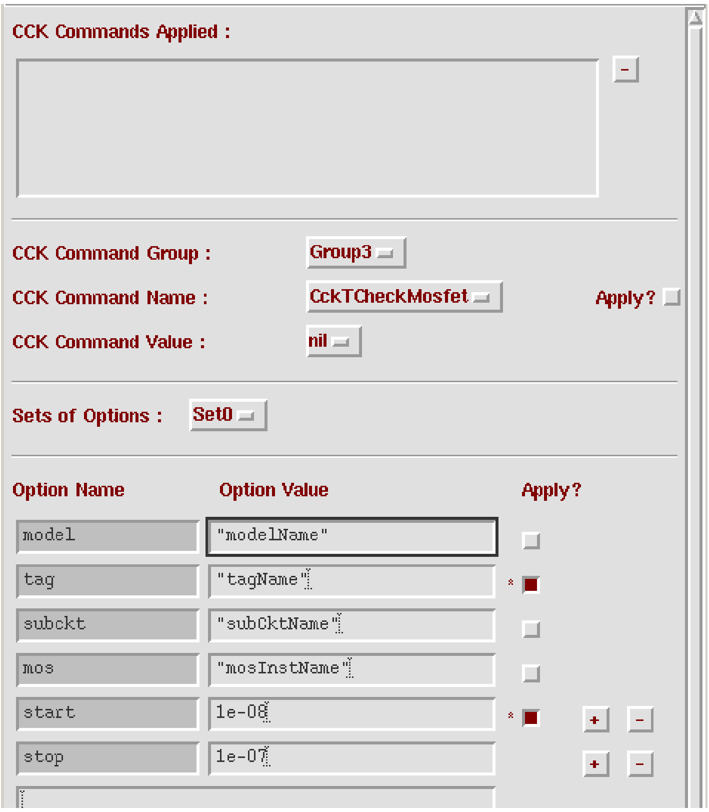
CCK Command Group : Group1

CckDiode

nil

Figure 67 CCK Command List

In the list shown in Figure 68, there is only one Group 3 command; however, it has numerous options listed.



CCK Commands Applied :

CCK Command Group : Group3

CCK Command Name : CckTCheckMosfet Apply? □

CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	"modelName"	□
tag	"tagName"	* ■
subckt	"subCktName"	□
mos	"mosInstName"	□
start	1e-08	* ■
stop	1e-07	

Figure 68 CCK Commands Applied: Window

6 CCK Command Value

CCK Command Value contains a list of commands with assigned values selected from a pre-determined list. A cyclic value button permits selecting from the command list.

Some commands themselves have values but without options.

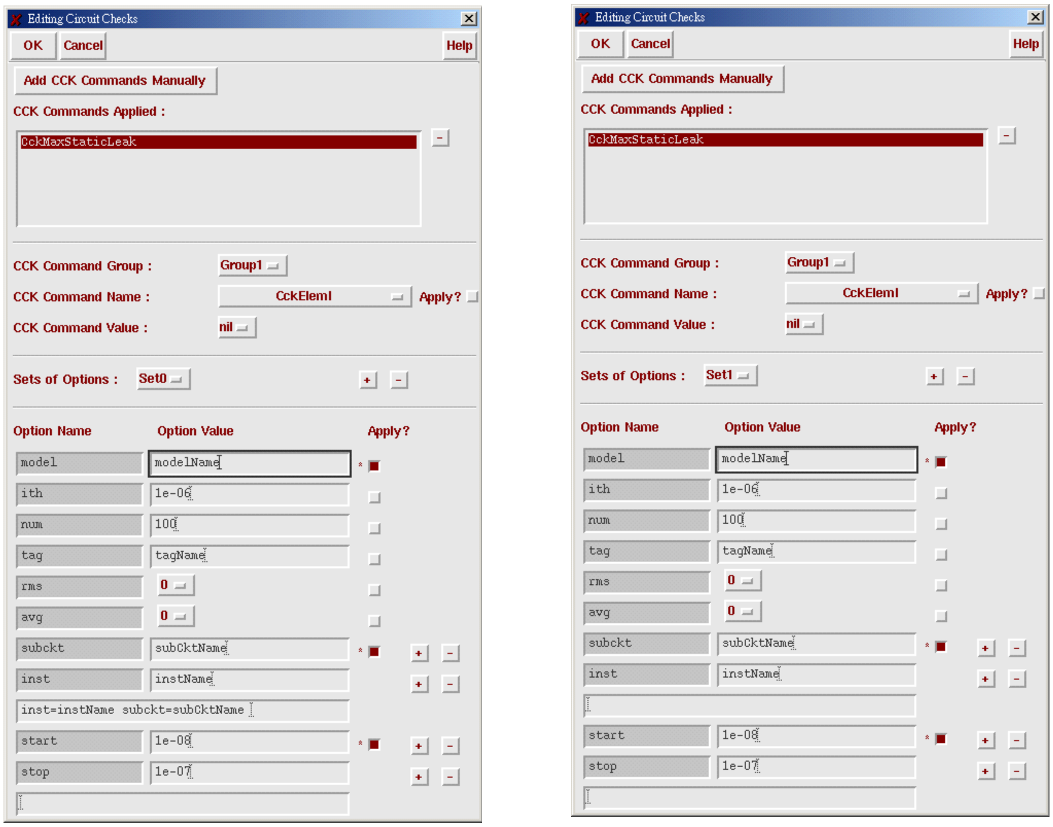
7 Apply?

Apply? applies the command selected from the CCK Command Name: drop down menu. The name of selected command is listed in the CCK Command Name: list box when activated. If Apply? is deselected, then the corresponding command name will be removed from the applied command list box.

8 Sets of options

Certain commands allow multiple sets of options, e.g. different CCK command sets may be specified with different models. To setup multiple options, perform the following, Select [+]. The [+] screen button is located on the RIGHT of the window.

Select [+] refreshes the bottom portion of the Editing CircuitChecks form to reflect a new option set with the new set number. The default option set values are set as shown in Figure 69:



CCK Commands Applied : CckMaxStaticLeak

CCK Command Group : Group1

CCK Command Name : CckEleml Apply? □

CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	* ■
ith	1e-06	□
num	100	□
tag	tagName	□
rms	0	□
avg	0	□
subckt	subCktName	* ■
inst	instName	
inst=instName subckt=subCktName		
start	1e-08	* ■
stop	1e-07	

CCK Commands Applied : CckMaxStaticLeak

CCK Command Group : Group1

CCK Command Name : CckEleml Apply? □

CCK Command Value : nil

Sets of Options : Set1

Option Name	Option Value	Apply?
model	modelName	* ■
ith	1e-06	□
num	100	□
tag	tagName	□
rms	0	□
avg	0	□
subckt	subCktName	* ■
inst	instName	
start	1e-08	* ■
stop	1e-07	

Figure 69   Editing CircuitChecks Window

If any command has multiple option sets, each set is presented on an individual command line in the output file.

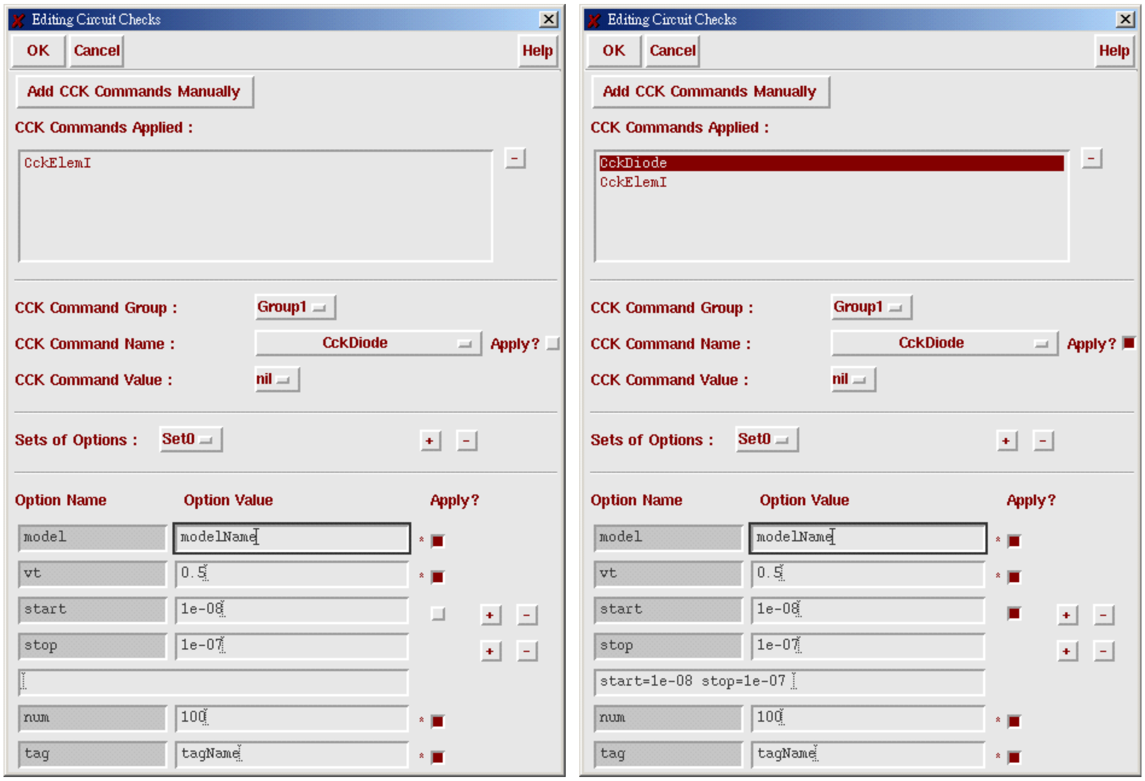
9 Sets of options

Use the [{+{] and the [{-{] screen buttons to add or delete an option set from an option set. The [{+{] and the [{-{] screen buttons function as follows:

- [{+{] adds a new set of options.
- [{-{] deletes the selected set of options from the current selection shown on the menu.

10 Command option

The Command option specifies the command option name, especially when multiple option sets are applied in the CCK command.



Left window:

CCK Commands Applied : CckElemI

CCK Command Group : Group1

CCK Command Name : CckDiode Apply? □

CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	■
vt	0.5	■
start	1e-08	□
stop	1e-07	
num	100	■
tag	tagName	■

Right window:

CCK Commands Applied : CckDiode
CckElemI

CCK Command Group : Group1

CCK Command Name : CckDiode Apply? ■

CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	■
vt	0.5	■
start	1e-08	■
stop	1e-07	
start=1e-08 stop=1e-07		
num	100	■
tag	tagName	■

Figure 70 Editing CircuitChecks Window with Commands

11 Combinations of related options

Some related CCK command options are used in combination such as start/ stop, subckt/inst, subckt/node, skipsub/skipinst, etc. These combinations can be specified in multiple sets having different option values. The string field displayed at item [11]. displays the current user specified setting. The controls described in [12] to are used to modify this field.

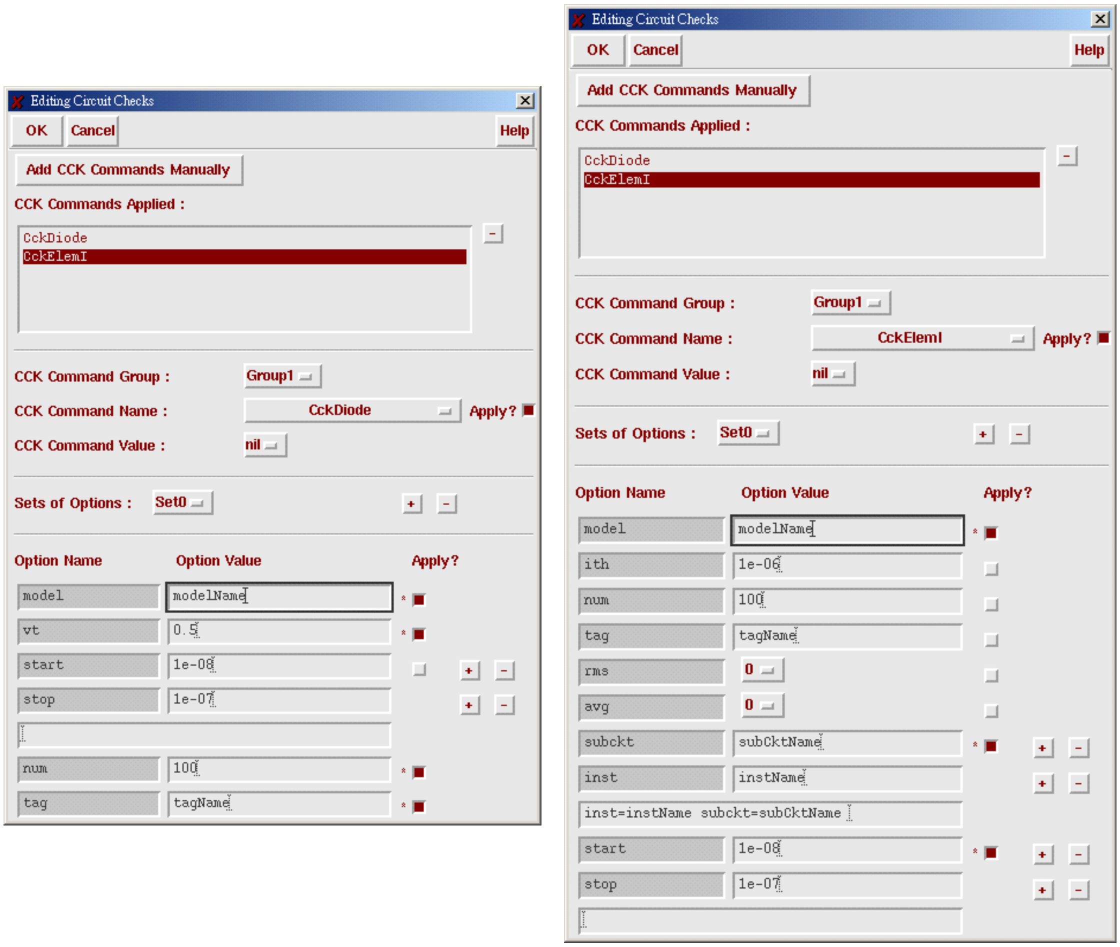
12 [{+{] and [{-{] for combined options

The [{+{] and [{-{] screen buttons are used to control the option as follows:

- [{+{] adds the corresponding option=value pair into the string field described in [11].
- [{-{] deletes the last added option=value pair.

Viewing Commands When Multiple Commands Are Applied

When more than one command is applied, it is possible to navigate between each of the commands by double clicking on a command name listed in the applied command list box as shown in Figure 71.



CCK Commands Applied :
CckDiode
CckElemI

CCK Command Group : Group1
CCK Command Name : CckDiode Apply? ■
CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	* ■
vt	0.5	* ■
start	1e-08	□
stop	1e-07	
num	100	* ■
tag	tagName	* ■

CCK Commands Applied :
CckDiode
CckElemI

CCK Command Group : Group1
CCK Command Name : CckElemI Apply? ■
CCK Command Value : nil

Sets of Options : Set0

Option Name	Option Value	Apply?
model	modelName	* ■
ith	1e-06	□
num	100	□
tag	tagName	□
rms	0	□
avg	0	□
subckt	subCktName	* ■
inst	instName	
inst=instName subckt=subCktName		
start	1e-08	* ■
stop	1e-07	

Figure 71 Editing CircuitChecks Window

cckCommandFile, cckDeviceVFile

Selecting the [OK] screen button on the CircuitCheck form generates two files in the same directory as the top netlist file.

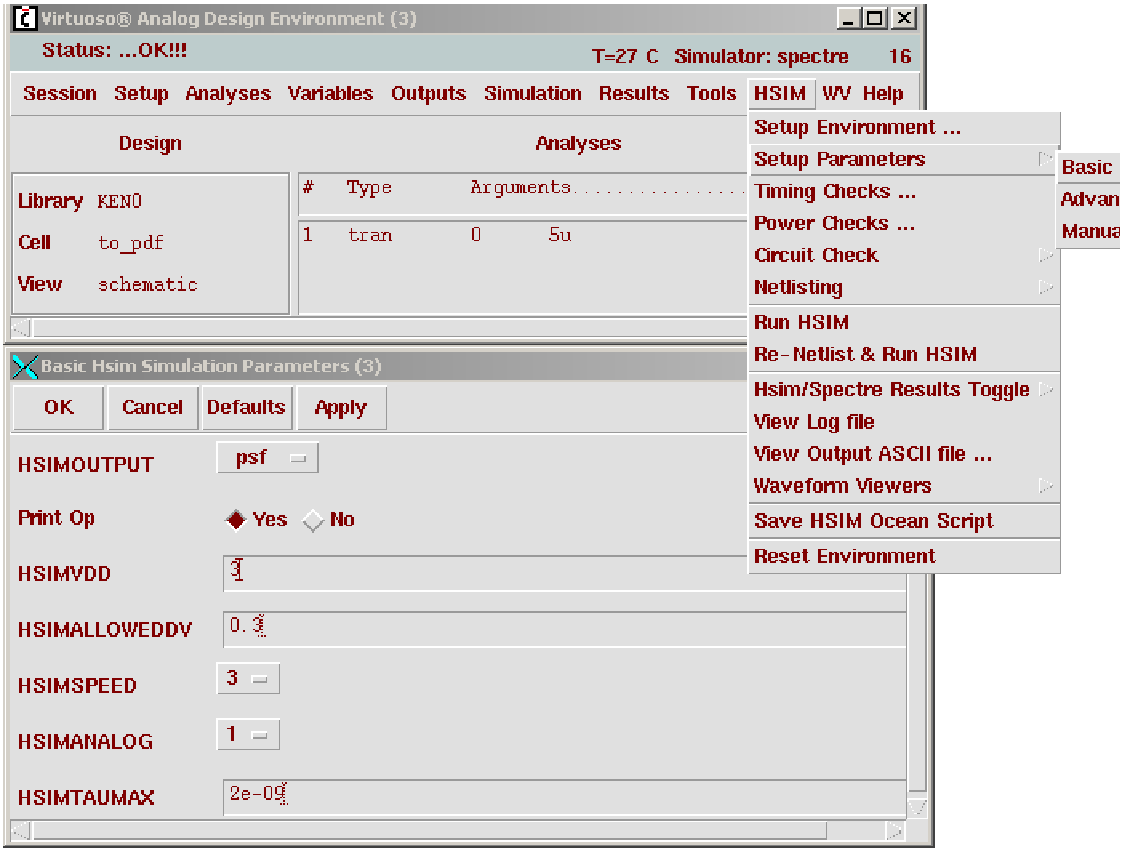
- cckCommandFile lists all applied Group 1 commands.
- cckDeviceVFile lists all applied Group 3 commands.

The top netlist file should always be recreated to include these CCK command files.

WaveView Analyzer Integration

To use WaveView, do the following (use Native Netlist Integration as an example):

1. Set HSIMOUTPUT to wdf as shown in Figure 72.



Library KENO

Cell to_pdf

View schematic

#	Type	Arguments.	
1	tran	0	5u

HSIMOUTPUT psf

Print Op ● Yes ○ No

HSIMVDD 3

HSIMALLOWEDDV 0.3

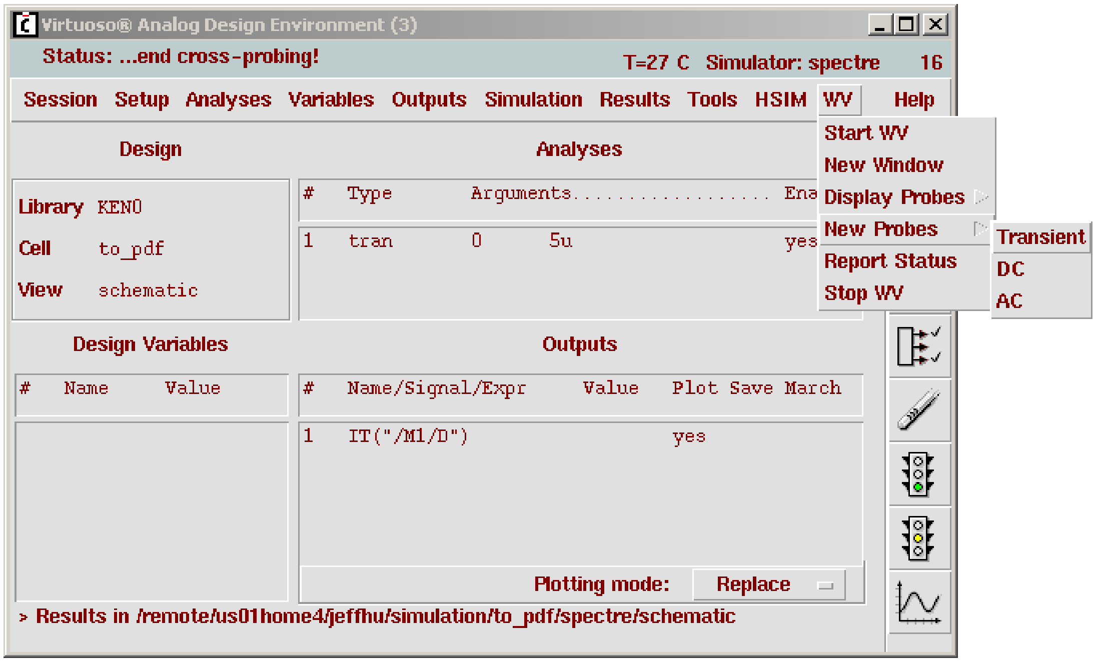
HSIMSPEED 3

HSIMANALOG 1

HSIMTAUMAX 2e-09

Figure 72 Basic HSIM Simulator Parameters Form

2. Generate the netlist.
3. Run the simulation.
4. Choose WV -> New Probes -> Transient to cross-probe the waveform using WaveView Analyzer, as shown in Figure 73.


Virtuoso® Analog Design Environment (3)
Status: ...end cross-probing!
T=27 C Simulator: spectre 16
Session Setup Analyses Variables Outputs Simulation Results Tools HSIM WV Help
Start WV
New Window
Display Probes
New Probes
Report Status
Stop WV
Transient
DC
AC
Design
Analyses
Design Variables
Outputs
Plotting mode: Replace

Library KENO
Cell to_pdf
View schematic

#	Type	Arguments.................	Ena
1	tran	0 5u	yes

#	Name	Value

#	Name/Signal/Expr	Value	Plot	Save	March
1	IT("/M1/D")		yes		

> Results in /remote/us01home4/jeffhu/simulation/to_pdf/spectre/schematic

Figure 73 Cross-probing Window

5. Select the desired signals from the schematic.
6. Press [Esc] to display the waveforms to WaveView Analyzer.

See the *WaveView v2004.5 User's Guide* for details of the WaveView Analyzer features.

11

Verilog/VHDL/HSIM Co-Simulation

Provides information on the Verilog/VHDL/HSIM co-simulation works.

This chapter presents a solution to Verilog/VHDL/HSIM co-simulation. The chapter is structured in the following order:

- System environment variable setup
- Analog/digital partitioning flows and examples
- Configuration commands
- Setup and partitioning guidelines

Verilog/VHDL/HSIM co-simulation allows a design to be partitioned into digital and analog blocks and simulated as one. The digital partition is in Verilog/VHDL and simulated by a Verilog/VHDL simulator. HSIM simulates the analog partitions with SPICE netlist format. The co-simulation interfaces synchronize both the Verilog/VHDL simulator and HSIM as well as passing and translating signal values back and forth between these two simulators.

The Synopsys Verilog/VHDL/HSIM co-simulation offers three analog/digital partitioning flows to fit into different design and verification methodologies:

- Verilog/VHDL netlist on top with leaf instances assigned to SPICE.
- SPICE netlist on top with leaf instances assigned to Verilog or VHDL.
- Integration with Virtuoso® Analog Design Environment with analog and digital netlists generated by SPECTRE/Verilog co-simulation.

Cadence® NC-Verilog®/VHDL[1] and Mentor Graphics ModelSim® are supported. Cadence Verilog-XL[2] also works with co-simulation, but it is not fully tested. Other PLI 2.0 compliant Verilog/VHDL simulators may also work with HSIM co-simulation but have not been fully tested.

Co-simulation uses Verilog Procedural Interface (VPI) or Programming Language Interface (PLI) 2.0, to interact with ncsim, NC-Verilog/VHDL simulator. The co-simulation executable is a shared library including VPI code and the HSIM engine. Co-simulation starts with ncsim as the master simulator which dynamically links with the co-simulation library and invokes the HSIM engine. The single process combines the ncsim, co-simulation interface, and HSIM engine. The interactions between ncsim and HSIM go through VPI function calls. This approach does not need any communication backplane and interprocess communication (IPC). Thus, best performance can be achieved.

Setting Up System Environment Variables for Co-Simulation

Before running co-simulation, the system must be set up using the following steps.

1. Set the NC-Verilog/VHDL executables path as shown in the following example:

   ```
   set path=($path /usr/local/vendors/cadence/ldv40/tools/bin)
   ```

2. Add the NC-Verilog/VHDL library path to the LD_LIBRARY_PATH environment variable as shown in the following syntax example:

   ```
   setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}/bin:/usr/local/
   vendors/cadence/ldv40/tools/inca/lib:/usr/local/vendors/
   cadence/ldv40/tools/lib
   ```

3. Add the directory containing libvpihsim.so to LD_LIBRARY_PATH. libvpihsim.so is the VPI shared library shipped to Synopsys customers.
4. Add a directory containing the Tool Command Language (TCL) shared library and output the shared library, such as libFSDB.so, to LD_LIBRARY_PATH. Shared libraries are provided in the tool installation.

   Note:

   For the HP-UX platform, the library path environment variable is SHLIB_PATH and the VPI shared library is libvpihsim.sl.

   Note:

   To run co-simulation with VCS and Verilog-XL, set the TCL_LIBRARY environment variable to the HSIM tool installation directory containing the init.tcl file.

Co-Simulation with Verilog as the Top Instance

High-Level Co-Simulation Instructions

High-level co-simulation provides the steps necessary to run Verilog/HSIM Co-simulation without using the cell view approach.

Note:

> Syntax Convention: A backslash character (\) in syntax examples marks a line continuation. Where there is no space before the \, the line continues unbroken. If a there is a space prior to the \, a space exists in the syntax.

To perform high-level co-simulation without using the cell view approach, do the following:

1. For those Verilog modules to be simulated by HSIM, replace their module body with only one line as shown in the following syntax example:

   ```
   initial $nsda_module();
   ```

2. Provide a SPICE netlist for Verilog modules that have the same module/ subcircuit and port names.
3. Recompile the Verilog source code using ncvlog. Proper hdl.var and cds.lib are required for ncvlog.
4. Insert ncelab with an additional command line option where libvpihsim.so is the VPI share library shipped to Synopsys customers as shown in the following syntax example:

   ```
   -loadvpi libvpihsim.so:nsda_vpi_startup
   ```

5. Specify HSIM parameters such as the netlist file name in the cosim.cfg file.
6. Run ncsim with additional command line option as shown in the following syntax example:

   ```
   -loadvpi libvpihsim.so:nsda_vpi_startup +nsda+”cosim.cfg”
   ```

Detailed Co-Simulation Instructions

The information contained in this section provides details on running Verilog/ HSIM co-simulation using the cell view approach.

To perform detailed co-simulation using the cell view approach, do the following:

1. Create new Verilog source files for the modules to be simulated in HSIM. It is not necessary to modify the original Verilog source code. The modules should have the same module name and port name as the original Verilog modules. The module body contains only the following syntax line:

   ```
   initial $nsda_module();
   ```

   Use a new file name extension, such as .cs, to compile the new files into a new view such as cosim view.
2. Modify hdl.var to define a new view as shown in the following syntax example:

   ```
   DEFINE VIEW_MAP (.cs => cosim)
   ```

3. The selected view described in the previous syntax is cell based. For an instance based view selection, insert the following compilation directive before the instance in the original Verilog source code as shown in the following syntax:

   ```
   `uselib lib=cosim_lib view=cosim
   ```

4. Create a SPICE netlist for the Verilog modules. Make sure to have the same subcircuit name as the Verilog module name and port names as well.

   Note:

   > If a subcircuit name is different from the module name, use map_subckt_name to associate them.
5. Compile the new Verilog files into a new view as specified in hdl.var.
6. Prepare co-simulation configuration file, e.g. cosim.cfg, with HSIM parameters and co-simulation parameters.
7. Run the NC-Verilog command with additional `-loadvpi` command option. NC-Verilog is a Cadence product that requires three steps to run a simulation: compilation, elaboration, and simulation. The related commands are:
   - Compilation—Use the `ncvlog` command. The syntax is :

     ```
     % ncvlog top.v gate.cs
     ```

   - Elaboration—Use the `ncelab` command. The syntax is:

```
% ncelab -loadvpi libvpihsim.so:nsda_vpi_startup -access \
   +rwc -LIBNAME cosim_lib cosim_lib.top -snapshot \
   cosim_lib.top:cosim
```

- Simulation—Use the `ncsim` command. The syntax is:

```
% ncsim -loadvpi libvpihsim.so:nsda_vpi_startup \
   +nsda+"cosim.cfg" cosim_lib.top:cosim
```

where `libvpihsim.so` is the VPI share library are shipped with the product.

Note:

> The `ncsim` command line option `+nsda+` is used to pass the cosim.cfg configuration file name to co-simulation. If the `+nsda+` option is not specified, the default configuration file is cosim.cfg.

Example 70 on page 423 illustrates a simple inverter chain with five inverters of which two are in analog and three in digital. The inv module is shown in both the top.v and gate.cs files. Since hdl.var defines the .cs file as cosim view with a higher precedence over the default module view, the inv module is simulated in HSIM. Its equivalent subcircuit is defined in the inv.spi file.

Sample files for the example include the following:

- top.v

  Verilog source code that contains the default inv module. This file is compiled into the default module view.

- gate.cs

  Verilog source code containing an inv module to be simulated by HSIM. This file is compiled into the cosim view.

- hdl.var

  Verilog configuration file that specifies a cosim view, asks the compiler to compile *.cs source files into cosim view, and asks elaborator to pick cells with cosim view whenever available.

- cds.lib

  Verilog configuration file that defines design libraries. The physical directory for a design library must pre-exist. Refer to the Cadence *NC-Verilog User Manual* for details.

- inv.spi and test.spi

SPICE netlist with inv subcircuit simulated by HSIM.

- cosim.cfg

  Co-simulation configuration file that specifies both HSIM and co-simulation parameters such as the SPICE netlist file name.

Example 70 Verilog Netlist on Top Flow Co-Simulation

```
   ======== top.v ========
// Top cell with 5 chain inverters, but pushing thru
// one more level of hierarchy by my_buf
`timescale 1ns / 10ps

module top;
     wire z1, z2, z3;
     testbench tb(z1, z2, z3, a);
     chain main(a, z1, z2, z3);
endmodule

module testbench(z1, z2, z3, a0);
     input z1, z2, z3;
     output a0;
     reg a0;
     always #25 a0=~a0;
     initial begin
          a0=1'b1;
          $monitor($time,, a0,, z1,, z2,, z3);
          #200;
          $finish;
     end
endmodule

module chain (a, z1, z2, z3);
     input a;
     output z1, z2, z3;
     my_buf x1 (a, z1);
     my_inv x2 (z1, z2);
     my_buf x3 (z2, z3);
endmodule
module inv (a, z);
     input                   a;
     output                  z;
     assign z=~a;
endmodule

module my_inv (a, z);
     input     a;
     output    z;
     assign z=~a;
endmodule

module my_buf (a, z);
     input     a;
     output    z;
     wire      t;
     my_inv IV1(a, t);
     inv IV2(t, z);
endmodule

======= gate.cs =======
module inv (a, z);
     input     a;
     output    z;
     initial $nsda_module();
endmodule

====== inv.spi ======
.subckt inv a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0
+ pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

====== test.spi =====
.param VDDVAL=3v
* global nodes
.global vdd vss gnd
* supplies
vvdd vdd 0 dc VDDVAL
vgnd gnd 0 dc 0v
.inc models
.inc inv.spi
.print v(*)
.end

======== cosim.cfg ========
set_args        test.spi

======== hdl.var =========
DEFINE WORK     cosim_lib
DEFINE VIEW_MAP (.cs => cosim)

======== cds.lib =========
DEFINE cosim_lib ./cosim_lib
```

Instance Based Instantiation with Verilog Configuration

NC-Verilog 5.1 supports instance based instantiation by using Verilog configurations in accordance with the IEEE standard, IEEE 1364-2001. A library map file contains the binding rules. This feature is invoked using ncvlog and ncelab with the -libmap command line option to specify the library map file.

Multiple implementations of the same module can be compiled into different design libraries. Using Verilog configurations, ncelab searches design libraries to bind instances as shown in Example 71.

Example 71 Instance Based Instantiation

```
====== File: top.v ======
module top();
     chain a1(...);
     chain a2(...);
endmodule

module chain(...);
     inv i1(...);
     inv i2(...);
endmodule

module inv(in, out);
     input in;
     output out;
     assign out = ~in;
endmodule

====== File: inv.cs ======
module inv(in, out);
     input in;
     output out;
     initial $nsda_module();
endmodule

====== Library map file: lib.map ======
library rtlLib top.v;
library cosimLib inv.cs;

config cfg;
     design rtlLib.top;
     default liblist rtlLib cosimLib;
     instance top.a2.i1 liblist cosimLib;
endconfig
```

To compile the design units, invoke ncvlog using the following syntax:

```
% ncvlog -libmap lib.map top.v inv.cs
```

This compiles the design units into the appropriate libraries as follows:

Design Units	Library
top	rtlLib

Design Units	Library
chain	rtlLib
inv (from inv.cs)	cosimLib
inv (from top.v)	rtlLib

In the lib.map (library map) file, the Verilog configuration cfg specifies an instance based instantiation for instance top.a2.i1. To elaborate the top design, use the following command:

```
% ncelab -libmap lib.map cfg -loadvpi \
    libvpihsim.so:nsda_vpi_startup -access +rwc
```

The instance top.a2.i1 is bound to the design unit inv in cosimLib while the remaining three inv instances are bound to the design unit inv in rtlLib. During co-simulation, the instance top.a2.i1 is partitioned to the analog simulator and the others are simulated by Verilog simulator. With the Verilog configuration, analog/digital partitioning for co-simulation can be accomplished in an instance based fashion.

Refer to the *NC-Verilog User Manual* and *IEEE 1364-2001 standard* for further details on instance based instantiation.

Co-Simulation with VHDL as the Top Instance

The Cadence ncsim is able to simulate pure Verilog, pure VHDL, and mixed Verilog/VHDL designs. This Synopsys co-simulation uses VPI to interact with ncsim. However, VPI can only access Verilog objects. In order to co-simulate with VHDL, HSIM needs Verilog as the media to interact with VHDL indirectly. Therefore, a Verilog wrapper is required.

Use the following steps to run co-simulation with VHDL designs.

1. Create a Verilog module with the same port definition as the VHDL entity to be partitioned to SPICE. This Verilog module contains only one statement as the module body and functions as a wrapper around the SPICE block as shown below:

   ```
   initial $nsda_module();
   ```

2. Modify the original VHDL code to instantiate the new Verilog module.

3. Compile the VHDL code with ncvhdl as shown in the following syntax:

   ```
   % ncvhdl top.vhd
   ```

4. Compile the Verilog code with ncvlog as shown in the following syntax:

   ```
   % ncvlog gate.cs
   ```

5. Elaborate the design with ncelab as shown in the following syntax:

   ```
   % ncelab -loadvpi libvpihsim.so:nsda_vpi_startup -access +rwc
   top:a
   ```

6. Prepare SPICE netlist for the SPICE block.
7. Setup co-simulation configuration file.
8. Run co-simulation with ncsim as shown in the following syntax:

   ```
   % ncsim -loadvpi libvpihsim.so:nsda_vpi_startup
   +nsda+cosim.cfg top
   ```

   Note:

   Bi-directional port in VHDL/HSIM co-simulation is not supported.

Example 72 on page 428 shows a VHDL on top design of a inverter chain with two leaf inverters assigned to SPICE. The VHDL entity inv is replaced by a SPICE subcircuit inv for co-simulation. A Verilog module inv is created as the wrapper of the SPICE subcircuit.

The following sample files are used in Example 72:

- top.vhd: The VHDL design
- gate.cs: The Verilog wrapper for SPICE subcircuit inv.
- test.spi: SPICE netlist
- inv_sub.spi: SPICE netlist
- cosim.cfg: Co-simulation configuration file
- hdl.var: NC-Verilog configuration file
- cds.lib: NC-Verilog configuration file

Example 72 VHDL on Top Flow Co-Simulation Files

```
========== File: top.vhd ===========
library ieee;
use ieee.std_logic_1164.all;
library std;
use std.textio.all;
entity top is
end top;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
library std;
use std.textio.all;
architecture A of top is
    component
     testbench port (z1, z2, z3: in std_logic;
          a: out std_logic);
    end component;
    component
     cut port (a: in std_logic;
          z1, z2, z3: out std_logic);
    end component;
    signal a, z1, z2, z3: std_logic;
begin
    tb: testbench PORT MAP (z1, z2, z3, a);
    main: cut PORT MAP (a, z1, z2, z3);
    process (a, z1, z2, z3)
     VARIABLE I: LINE;
    begin
         write( I, now, left, 15);
         write( I, a , right, 3);
         write( I, z1, right, 3 );
         write( I, z2 , right, 3);
         write( I, z3 , right, 3);
         writeline(output, I);
    end process;
end;
library ieee;
use ieee.std_logic_1164.all;
entity testbench is
    port (z1, z2, z3: in std_logic;
       a: out std_logic);
end testbench;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
library std;
use std.textio.all;
```

```
architecture behav of testbench is
    signal tick: std_logic;
begin
    process
        variable i: std_logic := '0';
        variable initial: integer := 0;
    begin
        a <= i;
        tick <= i after 3 ns;
     if (now >= 250 ns) then
         wait;
     end if;
        wait for 25 ns;
        i := NOT i;
    end process;
    process(tick)
        variable error: STRING (1 to 7) := "ERROR: ";
     VARIABLE I: LINE;
    begin
      if (now > 0 ns) then
        if (tick /= z1) or
           (tick = z2 ) or
           (tick = z3 ) then
            write( I, error, left, 7);
            write( I, now, left, 15);
            write( I, tick, right, 3);
            write( I, z1, right, 3);
            write( I, z2, right, 3);
            write( I, z3, right, 3);
            writeline(output, I);
        end if;
      end if;
    end process;
end behav;
library ieee;
use ieee.std_logic_1164.all;
entity cut is
    port (a: in std_logic;
       z1, z2, z3: out std_logic);
end cut;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
library std;
use std.textio.all;
architecture gate of cut is component
     my_buf port (a: in std_logic;
          z: out std_logic);
    end component;
    component
     my_inv port(a: in std_logic;
         z: out std_logic);
    end component;
    signal m1, m2, m3: std_logic;
begin
    x1: my_buf PORT MAP (a, m1);
    x2: my_inv PORT MAP (m1, m2);
    x3: my_buf PORT MAP (m2, m3);
    z1 <= m1;
    z2 <= m2;
    z3 <= m3;
--    process (a, m1, m2, m3)
--   VARIABLE I: LINE;
--    begin
--       write( I, now, left, 15);
--       write( I, a , right, 3);
--       write( I, m1, right, 3 );
--       write( I, m2 , right, 3);
--       write( I, m3 , right, 3);
--       writeline(output, I);
--    end process;
end;
library ieee;
use ieee.std_logic_1164.all;
entity my_inv is
    port(a: in std_logic;
      z: out std_logic);
end my_inv;
library ieee;
use ieee.std_logic_1164.all;
architecture behav of my_inv is
begin
--    z <= NOT a after 1 ns;
    z <= NOT a;
end;
--library ieee;
--use ieee.std_logic_1164.all;
--entity inv is
--    port(a: in std_logic;
--    z: out std_logic);
--end inv;
--library ieee;
--use ieee.std_logic_1164.all;
--architecture behav of inv is
--begin
--    z <= NOT a after 1 ns;
--    z <= NOT a;
--end;
library ieee;
use ieee.std_logic_1164.all;
entity my_buf is
    port (a: in std_logic;
       z: out std_logic);
end my_buf;
library work;
use work.all;
architecture gate of my_buf is
    component
     my_inv port(a: in std_logic;
         z: out std_logic);
    end component;
    component
     inv port(a: in std_logic;
         z: out std_logic);
    end component;
    signal t: std_logic;
begin
    IV1: my_inv PORT MAP (a, t);
    IV2: inv PORT MAP (t, z);
end gate;

========== File: gate.cs ===========
module inv (a, z);
    input   a;
    output  z;
    initial $nsda_module();
endmodule
========== File: test.spi ===========
.param VDDVAL=3v
* global nodes
.global vdd vss gnd
* supplies
vvdd vdd 0 dc VDDVAL
vgnd gnd 0 dc 0v
.inc models
.inc inv_sub.spi
.print v(*)
.end
========== File: inv_sub.spi ===========
.subckt inv a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends
.subckt invs a z vcc
```

```
m1 z A vdd vdd p l=0.35u w=400u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.35u w=200u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends
========== File: cosim.cfg ===========
set_argsspice/test.spi
========== File: hdl.var ===========
DEFINE WORKcosim_lib
DEFINE VIEW_MAP( .cs => cosim, .vhd => vhd, .v => module )
========== File: cds.lib ===========
INCLUDE /rmnt/tools/cadence/LDV51QSR1/tools/inca/files/
cds.lib
DEFINE cosim_lib ./cosim_lib
```

Co-Simulation with SPICE as the Top Instance

In this design flow, the SPICE netlist is the design top instance. Verilog instances are instantiated from the SPICE netlist. In general, circuit designers have the whole SPICE netlist and would like to replace certain digital blocks with Verilog instances.

To run co-simulation, the cosim.v Verilog interface file is automatically generated by executing HSIM command against the original SPICE netlist. The cosim.v file contains the Verilog top module instaitiating Verilog instances to replace SPICE subcircuits. Then co-simulation is conducted against cosim.v, other Verilog source files for digital instances, and the original SPICE netlist.

The procedure to run co-simulation for this design flow is described as follows:

1. Create a configuration file such as cosim.cfg for both HSIM and co-simulation containing the following commands:
   - set_args: Used with HSIM command line options including the SPICE netlist to run HSIM.
   - digital_cell or digital_cell_inst: Specifies the Verilog instances in the SPICE netlist.
   - verilog_file: Specifies the Verilog file containing the Verilog module definitions.

     Here is an example of a cosim.cfg file:

   ```
   set_args spice/test.spi
   digital_cell invd
   verilog_file verilog/invd.v
   ```

In this example, `invd.v` defines the Verilog inverter, and `digital_cell` defines the digital partitions instantiated in cosim.v which is generated by HSIM.

2. Run HSIM with the configuration file. HSIM stops simulation after generating cosim.v as shown in the following example:

   ```
   % hsim -cscfg cosim.cfg
   ```

   Once cosim.v is generated, Step 2 can be skipped in future co-simulation runs if analog/digital partitioning and the Verilog port analog/digital interface definitions remain unchanged.

3. Use the Verilog compiler to compile cosim.v together with other Verilog source files.

4. Start co-simulation from the top Verilog module defined in cosim.v and the SPICE netlist. HSIM will skip simulating the SPICE subcircuits specified in digital_cell or digital_cell_inst commands.

Example 73 on page 434 presents a simple inverter chain with a SPICE netlist on top and two leaf inverters partitioned to Verilog. Sample files for Example 73 include the following:

- cosim.v: Verilog top module that instantiates two Verilog inverters. This digital interface file is automatically generated by HSIM.
- test.spi: SPICE top netlist of an inverter chain.
- inv.spi: SPICE netlist for an inverter.
- invd.spi: SPICE netlist of the inverter to be partitioned to Verilog.
- buf.spi: A SPICE inverter chain.
- invd.v: A Verilog inverter module.

Example 73 SPICE Netlist-On-Top Flow Co-Simulation

```
====================== File: cosim.v ======================
'timescale 1ns / 10ps
module top;
     wire w1; // x1.n1
     wire \x1.n1 = w1;
     wire w2; // out1
     wire \out1 = w2;
     wire w3; // x3.n1
     wire \x3.n1 = w3;
     wire w4; // out
     wire \out = w4;
     // Instance section
     invd \x1.x2 (
     w1, w2);
     invd \x3.x2 (
     w3, w4);
     // interface nodes
     initial begin
          $nsda_a2d_node(w1, "x1.n1");
          $nsda_d2a_node(w2, "out1");
          $nsda_a2d_node(w3, "x3.n1");
          $nsda_d2a_node(w4, "out");
     end
     initial $nsda_module(1);
     //        By default, spiceflow co-simulation will use
.tran time
     //        for the simulation time
     //        To specify simulation time from verilog, please
add
     //        command "spice_finish 0"
     //        in the cosim config file
     //        initial begin
     //        #100 $finish;
     //        end
endmodule
====================== File: test.spi ======================
*
.param VDDVAL=3v
* global nodes
.global vdd vss gnd
* supplies
vvdd vdd 0 dc VDDVAL
vgnd gnd 0 dc 0v
* top level netlist
x1 in out1 buf
x2 out1 out2 inv
x3 out2 out buf
```

```
x4 out dummy inv
.inc models
.inc inv.spi
.inc invd.spi
.inc buf.spi
vin in 0 pwl 0n 0v 1n 0v 1.1n 3v 6n 3v 6.2n 0v r
.print v(*)
.tran 0.1n 100n
.end
=========== File: inv.spi =============
.subckt inv a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends
=========== File: invd.spi ============
.subckt invd a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends
=========== File buf.spi ==============
.subckt buf in out
x1 in n1 inv
x2 n1 out invd
.ends buf
============ File: invd.v ============
`timescale 1ns/10ps
module invd (a, z);
     input   a;
     output z;
     assign z =~a;
endmodule
```

Note:

To co-simulate a VHDL block in SPICE on the top flow, create a Verilog wrapper for the VHDL entity with the same port definition. In the new Verilog module, instantiate the VHDL entity.

Spectre/Verilog Co-Simulation Running Under the Virtuoso Analog Design Environment

The Virtuoso Analog Design Environment provides the co-simulation capability for Spectre® and Verilog®. The HSIM and NC-Verilog co-simulation are integrated into the Virtuoso Analog Design Environment GUI based on the

Spectre and Verilog netlists generated by Virtuoso. Refer Chapter 10, HSIM-Virtuoso Analog Design Environment Interface for specific details.

The procedure to run batch mode in HSIM and NC-Verilog co-simulation with netlists generated by Virtuoso Analog Design Environment is as follows:

1. Generate Spectre and Verilog netlists: In the Analog Artist window, select spectreVerilog as the simulator. Select design and then generate netlists.
2. Run spectreVerilog co-simulation to generate the vmx run script, runSimulation, in the project directory. Stop spectreVerilog co-simulation after runSimulation is created.
3. From the project directory, run HSIM co-simulation by using runnsdavmx command with the vmx run script as an input argument as shown in the following syntax example:

   ```
   % runnsdavmx runSimulation [options]
   ```

   `runnsdavmx` options include:

   - `-include <hsim_netlist_file>`
   - `-config <cosim_config_file>`
   - `-prefix` <hsim_prefix>: (default) hsim.
   - `-outdir <output_directory>`: (default) spectre -raw option.
   - `-vsrcd2a <0|1>`: (default is 0) This option is used to set the D2A input as a voltage source.
   - `-xl`: This option is used to invoke Verilog-XL for co-simulation.
   - `-help`

Donut Partitioning with Verilog as the Top Instance (V-S-V)

Using Verilog-on-Top Partitioning

In the V-S-V partitioning flow, the top instance is in Verilog and some modules will be simulated in HSIM. Within the modules to be simulated in HSIM, submodules can be partitioned to Verilog.

To use donut partition with Verilog on top, perform the following steps.

1. Use the `analog_cell` statements specify analog/digital partitioning. The syntax for `analog_cell` is:

```
analog_cell <cell name> -vmod <verilog module name>...
```

where <*cell name*> specifies the module to be simulated in HSIM and -vmod <*verilog module name*> specifies the module under <*cell name*> that remains in Verilog.

Any number of module names can be specified by adding additional -vmod <*verilog module name*> parameters. Refer to analog_cell on page 448 for additional information.

2. Run the simulation as normal Verilog netlist-on-top flow. The first run will exit before simulation time 0, generating a .cs and file for each donut cell.
3. Include the .cs files in Verilog compilation and run the simulation. The modules will be partitioned and simulated as specified in the analog_cell command.

   Note:

   The cosim view file extension (.cs) is optional and can be specified in the analog_cell command.

Figure 74 and Example 74 show donut partitioning with a Verilog top. The co-simulation configuration file should contain the analog_cell command.

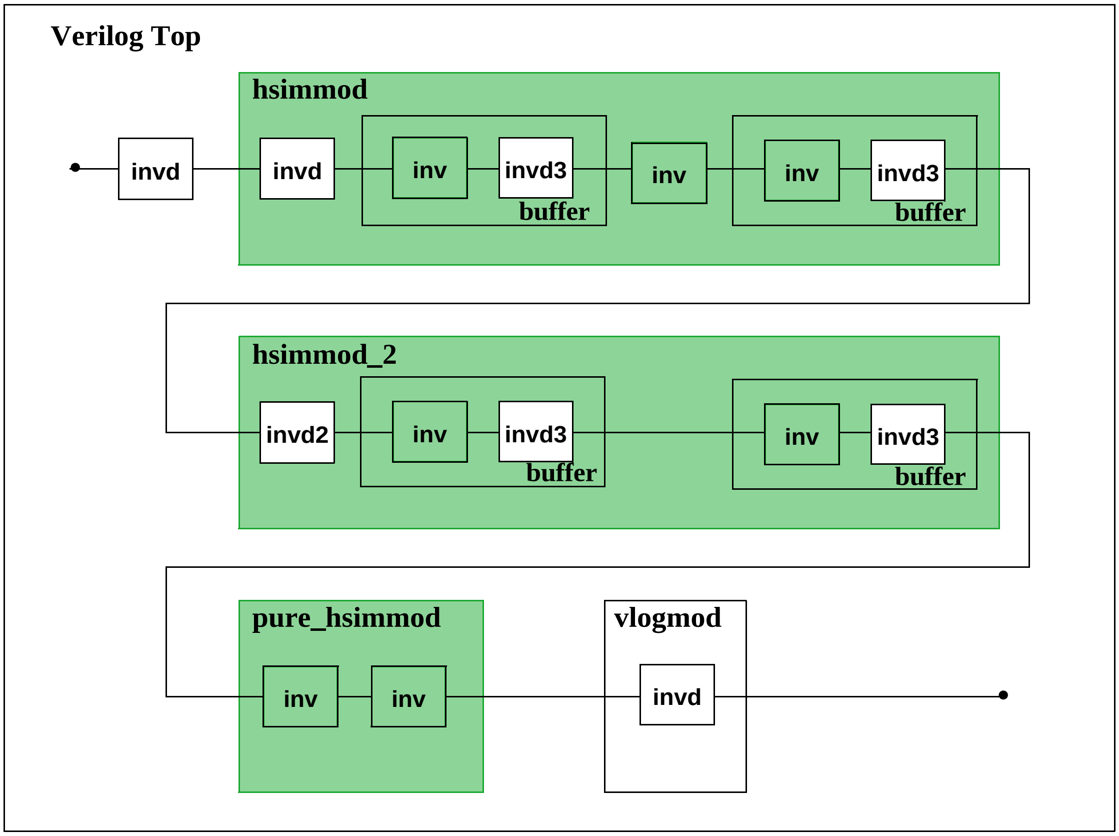


Figure 74 Verilog Top Co-Simulation Working in Donut Partitioning

Example 74 Verilog Top Co-Simulation Working in Donut Partitioning

```
============= cosim.cfg =============
set_args spice/test.spi
analog_cell -ext cs -dir . hsimmod   -vmod invd -vmod invd3
analog_cell -ext cs -dir . hsimmod_2 -vmod invd2 -vmod invd3
analog_cell -ext cs -dir . pure_hsimmod
```

In the configuration file shown in Figure 74 and Example 74, the syntax is as follows:

- `hsimmod`, `hsimmod_2`, and `pure_hsimmod`: Modules simulated in HSIM.
- `invd`, `invd2`, and `invd3`: Submodules simulated in Verilog.
- `-vmod`: A global option. For example, if `-vmod invd3` is present only in the `hsimmod` command line, all instances of `invd3` will be in Verilog, including those in `hsimmod_2`.

First Run Example

```
% ncvlog verilog/top.v verilog/invd.v
% ncelab -loadvpi libvpihsim.so:nsda_vpi_startup -access \ +rwc
-LIBNAME cosim_lib cosim_lib.top -snapshot \ cosim_lib.top:cosim
% ncsim -loadvpi libvpihsim.so:nsda_vpi_startup \
+nsda+”cosim.cfg” cosim_lib.top:cosim
```

After first run, the following files are generated in the specified “.” output directory. In this example, it is the current directory:

- hsimmod.cs
- hsimmod_2.cs
- pure_hsimmod.cs

Second Run Example

```
% ncvlog verilog/top.v verilog/invd.v hsimmod.cs \ hsimmod_2.cs
pure_hsimmod.cs
% ncelab -loadvpi libvpihsim.so:nsda_vpi_startup -access \ +rwc
-LIBNAME cosim_lib cosim_lib.top -SNAPSHOT \ cosim_lib.top:cosim
% ncsim -loadvpi libvpihsim.so:nsda_vpi_startup \
+nsda+”cosim.cfg” cosim_lib.top:cosim
```

Verilog and SPICE Files:

```
============= top.v =============
`timescale 1ns/10ps
module top;
     reg            vlog_drv;
     wire           h_in_1;
     wire           h_out_1;
     wire           h_out_2;
     wire           h_out_3;
     wire           v_out;

     invd           inv_vtop(vlog_drv, h_in_1);
     hsimmod        h_mod_1(h_in_1, h_out_1);
     hsimmod_2      h_mod_2(h_out_1, h_out_2);
     pure_hsimmod   h_mod_3(h_out_2, h_out_3);
     vlogmod        v_mod(h_out_3, v_out);

     initial begin
     #0             vlog_drv = 1'bz;
     #10            vlog_drv = 1'b1;
     #10            vlog_drv = 1'b0;
     #10            vlog_drv = 1'b1;
     #10            vlog_drv = 1'b0;
     #10            $finish;
     end
endmodule

module vlogmod (vmod_in, vmod_out);
     input          vmod_in;
     output         vmod_out;
     invd           inverter(vmod_in, vmod_out);
endmodule

module hsimmod (in, out);
     input          in;
     output         out;
     wire           out0, out1, out2;
     invd           inst_invd(in, out0);
     buffer         inst_buf1(out0, out1);
     inv            nst_inv(out1, out2);
     buffer         inst_buf2(out2, out);
endmodule

module hsimmod_2 (in, out);
     input          in;
     output         out;
     wire           out0, out1;
     invd2          inst_invd2(in, out0);
     buffer         inst_buf1(out0, out1);
     buffer         inst_buf2(out1, out);
endmodule

module pure_hsimmod (in, out);
     input          in;
     output         out;
endmodule

============= invd.v =============
`timescale 1ns/10ps

module invd (a, z);
     input          a;
     output         z;

     assign         z =~a;
     always @(a)$display(“%t invd : z = %v”, $time, z);
endmodule

module invd2 (a, z);
     input          a;
     output         z;

     assign         z =~a;
     always @(a)$display”%t invd2: z = %v”, $time, z);
endmodule

module invd3 (a, z);
     input          a;
     output         z;

     assign         z =~a;
     always @(a)$display(“%t invd3: z = %v”, $time, z);
endmodule

module inv (a, z);
     input          a;
     output         z;

     assign         z =~a;
     always @(a)v$display(“%t inv : z = %v”, $time, z);
endmodule

module buffer (in, out);
     input          in;
     output         out;
     wire           n1;
     inv           inst_inv(in, n1);
     invd3         inst_invd(n1, out);
endmodule

============= test.spi =============
.param VDDVAL=3v

* global nodes
.global vdd vss gnd

* supplies
vvdd vdd 0 dc VDDVAL
vgnd gnd 0 dc 0v

.inc models
.inc inv.spi
.inc invd.spi
.inc buf.spi

.subckt hsimmod in out
x0 in out0 invd
x1 out0 out1 buffer
x2 out1 out2 inv
x3 out2 out buffer
.ends

.subckt hsimmod_2 in out
x0 in out0 invd2
x1 out0 out1 buffer
x3 out1 out buffer
.ends

.subckt pure_hsimmod in out
x4 in out0 inv
x5 out0 out inv
.ends

vin in 0 pwl 0n 0v 1n 0v 1.1n 3v 6n 3v 6.2n 0v
.print v(*)
.tran 0.1n 10n

.end

============= inv.spi =============
.subckt inv a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
```

```
.ends

============= invd.spi =============
.subckt invd a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

.subckt invd2 a z
m1 z a vdd vdd p l=1.0u w=10u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=1.0u w=6u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

.subckt invd3 a z
m1 z a vdd vdd p l=1.0u w=10u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=1.0u w=6u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

============= buf.spi =============
.subckt buffer in out
x1 in n1 inv
x2 n1 out invd3
.ends buffer
```

Donut Partitioning with SPICE as the Top Instance (S-V-S)

Using SPICE-on-Top Partitioning

In the S-V-S partitioning flow, the top instance is in SPICE and Verilog instances are instantiated from the SPICE netlist. This is similar to the normal SPICE netlist-on-top flow. Within the Verilog module, sub blocks can be simulated in HSIM. The Verilog module containing analog cells is similar to the top Verilog module in the normal standalone Verilog netlist-on-top flow.

To use SPICE-on-top donut partitioning, perform the following steps:

1. Specify the cosim view of the sub block in the Verilog module to be simulated in HSIM. This is similar to the Verilog netlist-on-top flow where the hdl.var file should contain the following syntax specifying cosim view:

   ```
   DEFINE VIEW_MAP (.cs => cosim )
   ```

2. Replace the module body of the sub block in the Verilog module to be simulated in HSIM with the following syntax line:

```
initial $nsda_module();
```

3. Specify the following in the co-simulation config file:

```
set_args spice/test.spi
digital_cell buffer
verilog_file verilog/buf.v
```

   where `buffer` is the Verilog module name, and `verilog/buf.v` is the Verilog file name.

4. Run HSIM with the configuration file. HSIM stops simulation after generating cosim.v as shown in the following example:

```
% hsim -cscfg cosim.cfg
```

   Note:

   Once cosim.v is generated, Step 3 can be skipped in future co-simulation runs if analog/digital partitioning and the Verilog port analog/digital interface definitions remain unchanged.

5. The Verilog compiler is used to compile cosim.v together with other Verilog source files.

6. Start co-simulation from the top Verilog module defined in cosim.v and the SPICE netlist. HSIM will skip simulating the SPICE subcircuits specified in digital_cell commands while partitioning the sub-block specified in Step 2 into HSIM.

Figure 75 is an example of donut partitioning with a SPICE top. Within the SPICE top, the buffer is partitioned to Verilog while one of its sub blocks, inva, is partitioned to be simulated in HSIM.

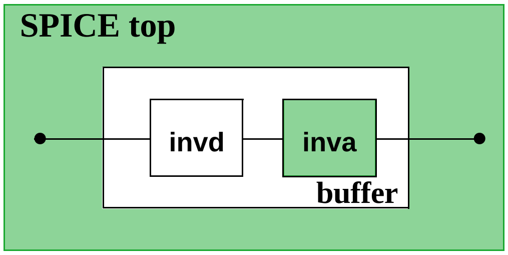


Figure 75 SPICE Top Co-Simulation Working in Donut Partitioning

First Run Example

```
% hsim -cscfg cosim.cfg
```

Second Run Example

```
% ncvlog verilog/cosim.v verilog/buf.v verilog/gate.cs

% ncelab -loadvpi libvpihsim.so:nsda_vpi_startup -access \ +rwc
-LIBNAME cosim_lib cosim_lib.top -SNAPSHOT \ cosim_lib.top:cosim

% ncsim -loadvpi libvpihsim.so:nsda_vpi_startup \
+nsda+"cosim.cfg" cosim_lib.top:cosim
```

Verilog and SPICE Files

```
============= test.spi =============
.param VDDVAL=3v

* global nodes
.global vdd vss gnd

* supplies
vvdd vdd 0 dc VDDVAL
vgnd gnd 0 dc 0v

* top level netlist
x1 in out1 buffer

.inc models
.inc inva.spi
.inc invd.spi
.inc buf.spi

vin in 0 pwl 0n 0v 1n 0v 1.1n 3v 6n 3v 6.2n 0v
.print v(*)
.tran 0.1n 10n

.end

============= inva.spi =============
.subckt inva a z
m1 z a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 z a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

============= invd.spi =============
.subckt invd d_a d_z
m1 d_z d_a vdd vdd p l=0.5u w=5u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
m2 d_z d_a 0 0 n l=0.5u w=3u as=1.0e-10 ad=1.0e-10 ps=0 pd=0
.ends

============= buf.spi =============
.subckt buffer in out
x1 in n invd
x2 n out inva
.ends buffer

============= buf.v =============
`timescale 1ns/10ps

module buffer (in, out);
     input     in;
     output    out;
     wire      n;
invd inst_invd (in, n);
inva inst_inva(n, out);
endmodule

module invd (d_a, d_z);
     input     d_a;
     output    d_z;

     assign    d_z = ~d_a;
     always @ (d_a) $display("%t invd : d_z = %v", $time, d_z);
endmodule

============= gate.cs =============
`timescale 1ns/10ps

module inva (a, z);
     input     a;
     output    z;

endmodule
```

Save-Restart in Co-Simulation

Co-simulation allows you to save the complete simulation state. Simulation can be restarted at a later time by loading the simulation state and continued from where it was saved. The simulation state is saved to a Verilog snapshot and an HSIM save file, hsim.iic.<time>. You can restart the simulation by invoking the Verilog snapshot.

To save a simulation state, you can get into ncsim interactive mode and apply the save command as shown in the following example:

```
ncsim> run -clean
ncsim> save <snapshot name>
```

To restart the simulation, use ncsim command with the saved snapshot as shown in the following example:

```
% ncsim <snapshot name>
```

Restart from within ncsim interactive mode is not supported.

Appending a Waveform in Co-Simulation

Co-simulation allows waveforms to be appended to fsdb or wdf files using the appropriate syntax as follows:

```
ncsim +restart+"<-fsdb fsdb_filename>" <snapshot name>
ncsim +restart+"<-wdf wdf_filename>" <snapshot name>
```

In the following command example, a new waveform will be appended to hsim_save.fsdb:

```
ncsim +restart+"-fsdb hsim_save.fsdb" <snapshot name>
```

Configuration File Commands

This section lists the configuration file commands used in Verilog/VHDL/HSIM co-simulation.

The value sets of some common configuration command arguments are as follows:

`<bool>`

0, 1

`<positive number>`

1, 2, 3, etc.

`<double>`

Floating point number

`<time>`

Floating point number plus time unit. For example, 100p and 1n stand for 100 pico seconds and 1 nano second, respectively.

`<file>`

File name

analog_cell

Generates Verilog module templates containing the $nsda_module() statement for analog partitions in the Verilog as the top instance flow.

Syntax

```
analog_cell [-ext <file name extension>] [-dir <directory>]
   <cell 1> <cell 2> -vmod <verilog sub-module name>...
```

Arguments

`-ext`

Specifies file name extension for the generated Verilog module templates. The default file name extension is cs.

`-dir`

Specifies the directory to put the generated Verilog module template. The default directory is the current working directory.

`cell` name

Can be a wildcard.

`-vmod`

Specifies the submodule that remains in Verilog. Do not use wild cards.

Note:

When using the -vmod option, only one cell within an analog_cell command may be used.

Description

The `analog_cell` command generates Verilog module templates containing the $nsda_module() statement for analog partitions in the Verilog as the top instance flow. If the design module of an analog partition does not exist in the design library, co-simulation stops after the template is generated. Then this new file must be compiled in order to start co-simulation. If the design module of an analog partition already exists in the design library, analog_cell will not generate the module template.

auto_vsrc_warning

Issues warning message if conflict exists between automatically detected voltage level and voltage level set by `set_port_prop` command.

Syntax

```
auto_vrsc_warning <bool>
```

Description

Default for `<bool>` is 0. When set to 1, a warning message is issued if a conflict between an automatically detected voltage level and a voltage level set by the set_port_prop configuration command occurs. Refer to Automatic Voltage Level Detection on page 463 for information regarding the rules for setting automatic voltage detection levels.

correct_netlist

Syntax

```
correct_netlist <bool>
```

Description

Default for `<bool>` is 1. If this option is on, and:

- Verilog module has more ports than subcircuit ports, it will drop port a connection which is found as a global node, i.e. vdd, vss.
- Subcircuit has more ports than inst module ports, it will create dummy node to let simulation go on.

define_print_variable

Defines a print variable used as a reference voltage in the `set_port_prop` command.

Syntax

```
define_print_variable <print variable name> = <expression>
```

Description

This command defines a print variable used as a reference voltage in the `set_port_prop` command. The print variable will be added to nsda_cosim.sp netlist file with SPICE .print statement.

Note:

> The syntax for the print variable in define_print_variable is identical to the .print statement syntax.

define_strength

Defines a strength table with resistances mapped to Verilog seven strength levels.

Syntax

```
define_strength <strength table name> [<double>] [-<strength
   option> <double>] [-<strength option> <double>] ...
```

Description

This command defines a strength table with resistances mapped to Verilog seven strength levels.

Each `-<strength option>` is used to map to the corresponding Verilog strength level and can be any of the following:

- -supply
- -strong
- -pull
- -weak
- -large
- -medium
- -small

The value inserted after `-<strength option>` is a strength resistor's resistance. If a value does not have an associated `-<strength option>`, it will be set as the default value for the remaining strength levels not specified using the `-<strength option>`.

`<strength table name>` is used in the `-strength` port property of the `set_port_prop` command for strength resolution at inout ports. Verilog inputs will be applied through the resistor with respect to the Verilog strength level and HSIM resolves contributions of both the Verilog- and SPICE-sides in order to obtain the final bi-directional net value.

digital_cell

Specifies the SPICE subcircuit to be partitioned to Verilog.

Syntax

```
digital_cell <sub-circuit name>
```

Description

In SPICE flow, specifies the SPICE subcircuit to be partitioned to Verilog.

digital_cell_inst

Specifies the SPICE instance to be partitioned to Verilog.

Syntax

```
digital_cell_inst <SPICE instance name>
```

Description

In SPICE flow, specifies the SPICE instance to be partitioned to Verilog.

dump_interface

Produces a report file showing the mapping result between analog and digital ports.

Syntax

```
dump_interface [0|1|2]
```

Arguments

0

Do not dump the .csintf file.

1

Generates the .csintf that lists all interface nodes and properties.

2

(Default) Generates the .csintf at the end of co-simulation that lists all interface nodes, properties, and the number of interface events for each interface node.

Description

This command produces a report file showing the mapping result between analog and digital ports.

Example

Here is a .csintf file example.

```
----------------------------------------
a2d main.out2 xmain.out2 node=out2 vhi=2.1 vlo=0.9
d2a main.out1 xmain.out1 node=out1 logichv=3 logiclv=0 rise=1000
fall=1000 rm_glitch=1000
----------------------------------------
```

- Column two lists the verilog ports.
- Column three lists the HSIM ports.

dump_port_prop

Dumps port properties associated with matching ports.

Syntax

```
dump_port_prop <file>
```

Description

Dumps out what port properties have been associated with the matching ports.

dump_setting

Dumps configuration command settings to HSIM log file.

Syntax

```
dump_setting <bool>
```

Description

Dumps configuration command settings to the HSIM log file. Default for `<bool>` is 0.

keep_iface_file

Specifies whether to deletes nsda_cosim.sp interface file automatically after completing simulation.

Syntax

```
keep_iface_file <bool>
```

Description

Co-simulation engine generates the nsda_cosim.sp interface file for the analog blocks in analog view, and it serves as the interface media between Verilog and HSIM simulators. Turning off this flag deletes this file automatically after simulation is complete. Default for `<bool>` is 1.

map_subckt_name

Maps module name to correct subcircuit definition in SPICE instantiation.

Syntax

```
map_subckt_name <module_name> <subckt_name>
```

Description

If module name is different than the subcircuit name, this command will map it to the correct subcircuit definition in SPICE instantiation.

map_unfound_port

Maps unfound port to the specified SPICE node name.

Syntax

```
map_unfound_port [-cell <pattern>] <map_node>
   <unfound_port> …
```

Description

When writing the interface netlist file, if a subcircuit has more ports than inst module ports, this command will map the unfound port to the specified SPICE node name.

The search priority is in a top-down order as follow:

- Exact cell name.
- Match cell pattern.
- Match unfound port list for rules without -cell argument.

report_logic_delay

Reports delayed logic output when the signal voltage crosses logic threshold voltage.

Syntax

```
report_logic_delay <time>
```

Description

[default] 2 ns

This command reports delayed logic output when the signal voltage crosses the logic threshold voltage, if the delayed logic output is greater than a specified time.

report_port_resistance

Generates a report of path resistances in the hsim.csres file.

Syntax

```
report_port_resistance {0|1|2}
```

Arguments

0

no report (default)

1

Report for inout ports only.

2

Report for all interface ports.

Description

This command generates a report of path resistances in the hsim.csres file. The report contains statistics of resistances of paths from interface nodes to voltage sources. The resistance values can be used as a reference to set up strength tables to map Verilog seven strength levels to resistors.

set_args

Passes the regular HSIM command line argument to HSIM.

Syntax

```
set_args <nsda_args> …
```

Description

This command passes the regular HSIM command line argument to HSIM. For example: set_args test.spi asks HSIM to accept test.spi as an input netlist.

set_intr_mode

Sets the interactive mode.

Syntax

```
set_intr_mode <bool>
```

Description

By default, Ctrl-C stops simulation in the Verilog simulator's interactive mode. To move between the interactive modes of the Verilog simulator and HSIM, use the following commands:

- `call nsda_intr_mode`: Leaves the Verilog simulator's interactive mode and enters the HSIM interactive mode.
- `quit`: Leaves the HSIM interactive mode and returns to the Verilog simulator's interactive mode.

If `set_intr_mode` is set to 1, Ctrl-C stops the simulation in HSIM's interactive mode instead of Verilog simulator's interactive mode. HSIM interactive commands can be applied to debug the simulation. In this case, Verilog's interactive mode can not be entered by users. Default for `<bool>` is 0.

set_fall_step

Specifies the number of stop times to update signal voltages when a rising/ falling slope occurs.

Syntax

```
set_fall_step <positive number>
```

Description

This command specifies the number of stop times to update signal voltages when a rising/falling slope occurs. Default for `<positive number>` is 10.

set_port_prop

Applies specified properties to matched cells or instances and their ports.

Syntax

```
set_port_prop [-cell <pattern>|-inst <pattern>] [-port
   <pattern>] -<port property1> <value1> -<port property2>
   <value2> … [-no_a2d <bool>] [-no_d2a <bool>]
```

Description

The port properties apply to the matched cells or instances and their ports.

- -cell is used for cell based port properties.
- -inst is used for instance based port properties.
- Port names match Verilog port definitions that are case sensitive.
- This specified value overrides any default value.
- If more than one rule is found for a particular property, the last rule is used.
- Without any cell or port pattern specified, the default value is used.

The options for port properties are listed below:

-alloweddv <double>

[default] HSIMALLOWEDDV

Set HSIMALLOWEDDV at the interface node.

-logichv <double> | <output variable>

[default] HSIMLOGICHV

Set port logic1 voltage.

Its value can be a double number or an output variable which is a string identifier starting with an alphabetic letter.

The output variable is defined with a .print statement to represent a voltage expression. For example:

```
.print logichv=par('0.7 * v(vdd)')
```

where logichv is the output variable and '0.7 * v(vdd)' is the voltage expression.

-logiclv <double>|<output variable>

[default] HSIMLOGICLV

Set port logic0 voltage.

Its value can be a double number or an output variable which is a string identifier starting with an alphabetic letter.

-logicxv <double>|<output variable>

[default] HSIMLOGIGLV

Set port logic X voltage.

Its value can be a double number or an output variable which is a string identifier starting with an alphabetic letter.

-vhi <double>|<output variable>

[default] HSIMVHTH if specified, otherwise use the following:
(logichv - logiclv) * 0.7

Set port logic1 threshold voltage.

Its value can be a double number or an output variable which is a string identifier starting with an alphabetic letter.

-vlo <double> <output variable>

[default] HSIMVLTH if specified, otherwise use the following:
(logichv - logiclv) * 0.3

Set port logic0 threshold voltage.

Its value can be a double number or an output variable which is a string identifier starting with an alphabetic letter.

-timex <time>

[default] No state X report.

Report X when the output port voltage stays between -vlo and -vhi longer than the timex time.

-slope <time>

HSIMSLOPE

Set port rising & falling time.

-rise <time>

HSIMRISE if specified, otherwise use HSIMSLOPE

Set port rising time.

-fall <time>

HSIMFALL if specified, otherwise use HSIMSLOPE.

Set port falling time.

-delay <time>

[default] 0

Set port delay. This delays the signal output to Verilog.

Only allow positive delay.

-delay1 <time>

[default] 0

Apply port delay to the rising edge only.

-delay0 <time>

[default] 0

Apply port delay to the falling edge only.

-delay_hz2st <time>

[default] 0

Apply port delay to signal changes from a Hi-Z state to a strong state.

-rm_glitch <time>

[default] -slope value

Remove glitches within <time> after Verilog input changes.

Apply to inout port.

-strength <strength name>

[default] no strength

<strength name> is a string identifier defined in define_strength.

Apply to inout port for strength resolution with the resistor specified in <strength name>.

-vsrc <bool>

[default] 0

Model input as a voltage source. It will be partitioned into a smaller block and results in a faster simulation runtime.

Only inputs without Hi-Z can use this option, otherwise the simulation may be incorrect.

-vprint <bool>

[default] 0

Insert .print statement to print voltage value.

-lprint <bool>

[default] 0

Insert .lprint statement to print voltage logic.

-no_a2d <bool>

[default] 0

Skips the a2d interface element insertion on the specified interface node.

-no_d2a< bool>

[default] 0

Skips the d2a interface element insertion on the specified interface node.

set_port_prop_warning

Specifies the number of warning messages allowed before simulation stops.

Syntax

```
set_port_prop_warning <number> [-stop]
```

Description

Warning messages are issued when `set_prop_prop` command specifies mismatched ports. This command specifies the number of warning messages allowed before simulation stops. If `-stop` option is specified, simulation stops whenever there is any mismatched port. The default is 250 warning messages with -stop option.

set_print_progress

Specifies the time interval to output co-simulation progress.

Syntax

```
set_print_progress <time>
```

Description

This command specifies the time interval to output co-simulation progress.

set_rise_step

Specifies the number of stop times to update signal voltages when a rising/ falling slope occurs.

Syntax

```
set_rise_step <positive number>
```

Description

This command specifies the number of stop times to update signal voltages when a rising/falling slope occurs. Default for `<positive number>` is 10.

set_slope_step

Specifies the number of stop times to update signal voltages when a rising/falling slope occurs.

Syntax

```
set_slope_step <positive number>
```

Description

This command specifies the number of stop times to update signal voltages when a rising/falling slope occurs. Default for `<positive number>` is 10.

set_verbose

Sets the level of detail for output messages.

Syntax

```
set_verbose <level>
```

Description

This command sets the level of detail for output messages. `<level>` can be none, low, high, or detail; default is high.

none

> Suppresses any messages generated by the co-simulation interface except error message.

low

> Writes information messages generated by the co-simulation interface.

high

> Writes warning messages generated by the co-simulation interface.

detail

> Writes suggestion on which D-to-A input should be defined as VSRC to speedup the simulation time and other messages.

set_verilog_supply1

Defines voltage level for Verilog supply1.

Syntax

```
set_verilog_supply1 <double>
```

Description

This command defines voltage level for Verilog supply1.

set_verilog_supply0

Defines voltage level for Verilog supply0.

Syntax

```
set_verilog_supply0 <double>
```

Description

This command defines voltage level for Verilog supply0.

verilog_file

Specifies Verilog source file containing Verilog module definitions for digital_cell or digital_cell_inst.

Syntax

```
verilog_file <Verilog source file name>
```

Description

In SPICE flow, this command specifies Verilog source file containing Verilog module definitions for digital_cell or digital_cell_inst. `verilog_file` can be applied multiple time for different verilog sources.

Example 75 Configuration File Example

```
set_args  hsim_top.sp
set_rise_step10
set_fall_step6
set_port_prop-cell top -port outport1 outport2 \
         outport3 -vhi 2.64 -vlo 0.66
set_port_prop-cell top -port inport* \
         -logichv 3.3 -logiclv 0 -slope 100ps
```

Automatic Voltage Level Detection

Co-simulation is able to automatically identify voltage levels at interface nodes thereby reducing the need for user intervention. The rules described in the Voltage Setting Rules section have the following precedence: Rule 1, Rule 2, and Rule 3.

Voltage Setting Rules

Rule 1

The set_port_prop configuration command provides the flexibility to over write both default and automatically detected voltages.

Rule 2

Search through channel connected voltage sources. The voltage levels of the voltage sources will be applied to the interface nodes. Warning messages are given if there is any conflict between detected voltage sources and configuration commands. By default, Warning messages are suppressed. They can be turned on using the auto_vsrc_warning configuration command. A Warning message is given if the interface node is not connected to any voltage source and HSIMVDD is applied.

Rule 3

HSIMVDD is used as the default voltage if Rule 1 and Rule 2 do not apply. If any channel connected voltage source is detected with a different voltage than HSIMVDD, a Warning message is issued and the detected voltage is applied.

Co-Simulation Interactive Mode

The co-simulation interactive commands add to the command set described in the *HSIM Simulation Reference Manual: Chapter 12, Interactive Mode Debugging*. The co-simulation interactive mode allows information to be obtained on both interface elements and interface activity history. It also permits watch points to be set on interface node activities to catch a specific event in the interactive debugging mode.

Co-simulation interactive commands are used with HSIM>, the HSIM interactive mode prompt. The *HSIM Simulation Reference Manual: Chapter 12, Interactive Mode Debugging* provides information on how to get into the HSIM interactive mode.

To get into HSIM interactive mode from the ncsim>, the NC-Verilog interactive prompt, use the following command:

```
call nsda_intr_mode
```

To continue simulation in NC-Verilog, issue the command cont from the HSIM interactive mode prompt HSIM> and simulation will continue. If you are prompted with ncsim> after issuing the cont command at HSIM>, type run and NC-Verilog will continue.

Note:

> Currently only NC-Verilog is supported in co-simulation interactive debugging.

Table 8 on page 464 lists the commands used in co-simulation interactive debugging.

Table 8 Co-Simulation Interactive Mode Commands

Command	Function
csli	List Interface Nodes
csh	Print Global Interface History in Time
csnh	Print Interface Node History by Node Name
csinh	Print Interface Node History by Node Index
csnph	Set the Number of Entries Printed by csnh and csinh
csnw	Set Watchpoint to Interface Node by Node Name
csinw	Set Watchpoint to Interface Node by Node Index
csdnw	Delete Watchpoint by Node Name
csdinw	Delete Watchpoint by Node Index

List Interface Nodes

csli

```
csli <pattern> <-a2d|-d2a|-biput>
```

csli lists all co-simulation interface nodes if no option is specified. A pattern can be used to search for certain names. You can choose to list a certain type of interface node by specifying -a2d, -d2a or -biput.

Table 9 List Interface Nodes: csli Syntax Descriptions

Parameter	Description
pattern	Pattern used to search for certain interface node names. Pattern matching is based on the Tool Command Language (TCL) API.
-a2d	Lists only a2d (HSIM to Verilog) interface nodes.
-d2a	Lists only d2a (Verilog to HSIM) interface nodes.
-biput	Lists only bi-directional interface nodes.

Table 10 Example

HSIM > csli	Prints all interface nodes.
HSIM > csli *addr* -d2a	Prints d2a interface nodes with names matching the pattern *addr*.

A typical result of csli command is shown in the following example. Note that <=> denotes bi-directional ports:

```
HSIM > csli
cosim interface nodes:
---------------------------------
id        type          node
---------------------------------
7         <=>a2d        b[3]
6         <=>a2d        db[2]
5         <=>a2d        db[1]
4         <=>a2d        db[0]
10        d2a           pch2
12        d2a           rd2
15        d2a           wr2
7         <=>d2a        db[3]
3         d2a           addr[2]
6         <=>d2a        db[2]
2         d2a           addr[1]
5         <=>d2a        db[1]
1         d2a           addr[0]
4         <=>d2a        db[0]
8         d2a           en2
---------------------------------
```

Note:

In this example, the bi-directional ports have both a2d and d2a interface nodes: 7 <=>a2d db[3] and 7 <=>d2a db[3].

Print Global Interface History in Time

csh

```
csh <number of entries (default is 10)>
```

csh prints the global interface activity history in chronological order. If no argument is specified, csh prints the maximum number of entries available up to a maximum of 10 entries. If the number of entries is specified, csh prints up to the specified number of entires. The maximum number of global history entries is set to 10000 by default and can be changed by the max_history command in co-simulation configuration file as follows:

```
max_history <max # of global history entires>
```

Table 11 Example

HSIM > csh	Prints 10 global interface activity history entries.

Table 11 Example

HSIM > csh 20	Prints 20 global interface activity history entries.

Print Interface Node History

csnh, csinh

```
csnh <name>
csinh <id>
```

csnh and csinh print the activity of a specified interface node if available. Entries for the specified node stored in the history buffer are printed in chronological order. Both a2d and d2a history will be printed if available. The maximum number of entries printed each time by csnh and csinh can be set by the command csnph. The default is 10.

The id corresponds to the id field in the output of the csli command. This id can also be used in other HSIM interactive commands.

Table 12 Example

HSIM > csnh db[3]	Prints activity history of interface node on db[3].
HSIM > csinh 10	Prints activity history of interface node with index 10.

Set the Number of Entries Printed By csnh and csinh

csnph

```
csnph <number of entries>
```

csnph reports the current setting if no argument is given. If an argument is specified, the number of entries to be printed by csnh and csinh commands are set. The number is limited between max_history and 0.

Table 13 Example

HSIM > csnph	Prints current csnph setting.

Table 13 Example

HSIM > csnph 20	Sets the max number of entries to be printed in each csnh and csinh call to 20.

Set Watchpoint to Interface Node

csnw, csinw

```
csnw <name> <-a2d|-d2a|-hz>
csinw <id> <-a2d|-d2a|-hz>
```

csnw and csinw set a watch point to the specified interface node. If no additional option is given, any activity on the interface node will trigger the watch point and you will enter the HSIM> prompt. Use -a2d|-d2a|-hz to catch a specific type of interface activity. If no argument is given to csnw and csinw, a list of current watch points is printed. Previous watch point settings are overridden by the new setting.

Table 14 Set watch point to interface node: csnw csinw Syntax Descriptions

Parameter	Description
name	Interface node name to which the watchpoint is set.
id	Interface node id to which the watchpoint is set.
-a2d	Watch for a2d (HSIM to Verilog) activity only.
-d2a	Watch for d2a (Verilog to HSIM) activity only.
-hz	Watch for Hi-Z event only.

Table 15 Example

HSIM > csnw	Prints the list of currently set watchpoints.
HSIM > csnw addr[2]	Sets watchpoint on interface node addr[2].
HSIM > csinw 5 -hz	Sets watchpoint on interface node with id 5 to watch for high-z events.

Table 15 Example

HSIM > csnw db[0] -d2a	Sets watchpoint on d2a part of interface node db[0].

Delete Watchpoint

csdnw and csdinw delete the watchpoint specified by name or id, or delete all watchpoints if -a or -all option is used. If no argument is given, csdnw and csdinw print the list of currently set watchpoints.

csdnw, csdinw

```
csdnw <name|-a|-all>
csdinw <id|-a|-all>
```

Table 16 Example

HSIM > csdinw	Prints the list of currently set watchpoints.
HSIM > csdnw db[1]	Deletes the watchpoint on db[1].
HSIM > csdinw 4	Deletes the watchpoint on node with id 4.

Verilog System Tasks for Co-Simulation

The following system tasks are available for interactions between Verilog and SPICE. They are incorporated into Verilog source code to pass data to or retrieve data from analog blocks. System tasks should be put into the initial block of a Verilog module.

```
$nsda_a2d_node (net, “SPICE node name”)
```

Creates an A-to-D interface element between the Verilog net and SPICE node. This system task behaves like a continuous assignment from a SPICE internal node to the Verilog net.

```
$nsda_add_cap (net, variable)
```

This system task adds capacitance to the SPICE node connecting to the interface net. It requires two arguments, one Verilog net and one Verilog variable, constant, or parameter of real type. The Verilog net has to be an interface connecting to a SPICE node. The second argument specifies capacitance in Farad and represents the lumped sum capacitance of Verilog side components connecting to the SPICE node.

`$nsda_d2a_node(net, "SPICE node name")`

Creates a D-to-A interface element between the Verilog net and SPICE node. This allows Verilog to connect the net directly to a SPICE internal node, instead of going through port mapping. For example,

- initial $nsda_d2a_node (test, “xi1.xi2.sync”);
- where test is a Verilog net in the module containing this system task and xi1.xi2.sync is the hierarchical path name to identify a SPICE node.

`$nsda_get_volt (net, variable)`

Requires two arguments, one Verilog net and one Verilog variable of real type. The Verilog net has to be an interface connecting to a SPICE node. This system task retrieves the analog voltage of the SPICE node at current time and assigns the voltage to the variable.

`$nsda_inout_node (net, "SPICE node name")`

Creates one D-to-A interface element and one A-to-D interface element between the Verilog net and SPICE node. This is equivalent to one $nsda_d2a_node() and one $nsda_a2d_node() combined.

`$nsda_module()`

Designates the current module to be partitioned into an SPICE subcircuit. The module body should contain nothing but only one initial block of this system task.

`$nsda_save_waveform(obj1 [, level1], obj2 [, level2], ...)`

Allows Verilog object waveforms to be saved to the HSIM waveform file. The Verilog objects can be net, register, net bit, register bit, and module instance. The optional level argument is valid only for module instance objects and specifies all nets under the design hierarchy level. Its default value is 1. A 0 level means full hierarchy of the given instance.

`$nsda_set_volt (net, variable)`

Requires two arguments, one Verilog net and one Verilog variable, constant, or parameter of real type. The Verilog net has to be an interface connecting to a SPICE node. This system task assigns the value of the second argument to the SPICE node as an analog voltage at current time.

Co-Simulation Setup Guidelines

There are several areas that require special attention during co-simulation setup in order to help simulation performance and avoid common mistakes.

Map Correct Port Voltages

This is especially true for logicxv: Verilog assigns logic X value to the nets which are not initialized. For input ports from Verilog to HSIM, HSIM takes port voltages for DC initialization and simulation. It is important to map a correct analog voltage for logic X value at input ports. Some circuits require logic X to have the same analog voltage as logic0, while some circuits require it to be the middle voltage between logic1 and 0.

Define Clear Port Direction

If a port direction is known to be unidirectional for the SPICE block, its corresponding co-simulation view Verilog module should clearly define an input or output port rather than an inout port. This will reduce the number of interface elements and improve simulation performance.

Set Input Ports As Voltage Sources If Possible

If the input from Verilog to HSIM will never become HiZ, this input can be treated as a voltage source to the SPICE block. This will improve HSIM simulation. Use the -vsrc option of set_port_prop configuration command to set ports as voltage sources.

Define SPICE Netlist Bus Notation

Usually, Verilog defines vector nets at ports. The SPICE netlist only has bit level port definition. A bus notation is required to map each individual bit level port back to Verilog vector ports. The default bus notation for SPICE netlist is square brackets []. Other bus notations can be set by using HSIMBUSDELIMITER command in the SPICE netlist as shown in the following example:

```
.param HSIMBUSDELIMITER=<>
.param HSIMBUSDELIMITER=_
```

Handle Bi-Directional Ports

If the analog partition presents Hi-Z states to interface ports, use .param HSIMHZ=1 in the SPICE deck.

If strength fighting occurs at bi-directional interface ports, use the -strength option for set_port_prop and define_strength to map verilog strengths to the proper resistance values.

Partitioning Guidelines

Partition Boundary with Clear Digital Behavior

In co-simulation, digital and analog signals are presented on two sides of a partition boundary. In order to reduce the loss of accuracy and to maintain correct functionality, the boundary signals should have clear digital behavior and should not be voltage sensitive.

Avoid Partitioning at Timing Sensitive Signals

The signal conversion from analog to digital depends on high and low threshold voltages. If the circuit design is timing sensitive at the interface signals, functionality errors may occur due to timing shift by a slight change in threshold voltages. There should be certain timing error margin for the interface signals. Also, the timing representation in Verilog may not match the exact timing in SPICE. It is recommended not to partition at timing sensitive signals.

Avoid Reach-in Signals in Analog Partitions

Verilog elaboration will fail when the Verilog netlist contains a reach-in signal partitioned into an analog partition whose internal objects are not visible to Verilog elaborator.

Reach-in signals can be replaced with a new Verilog net using either of the following system tasks; depending on the direction of the original reach-in signal.

```
$nsda_a2d_node()
$nsda_d2a_node(),
```

The system task associates the new Verilog net to the SPICE node that is equivalent to the original reach-in signal.

Avoid Partitioning at Bi-directional Signals Involved Strength Fighting and Pass Switches

Bi-directional interface signals are supported. However, the signal value set by VPI at one terminal of a pass switch (the primitive gate tranif0 and tranif1) cannot be propagated to the other end. A Verilog n-MOSFET gate is added in between the pass switch and the interface signal to allow signal value passing through the pass switch. If the bi-directional interface signals involve strength fighting, the final signal value is resolved by HSIM. A resistor is added to incorporate the contribution of the digital signal in resolving the final value. Special attention is required to map the resistance value, specified by set_port_prop configuration command, to its corresponding digital strength.

Avoid Fine Grain Partitioning

Fine granularity partitioning creates many small analog and digital blocks and introduces many interface signals which decrease co-simulation performance. Frequent and unnecessary analog/digital signal conversion may also introduce functionality errors.

Strength Table Setup Guidelines

Multiple drivers can drive the same net using different values. The final value of the net depends on the strengths of the drivers. Strength fighting may occur at bi-directional nets or inout ports of digital/analog partitions. Verilog defines seven strength levels and rules to resolve strength fights. HSIM models Verilog strength as a resistor. The Verilog signal input is applied through the resistor and HSIM resolves both Verilog and SPICE contributions to obtain the final values of the bi-directional nets.

A strength table defines a set of resistance strength values that are mapped to Verilog seven strength levels for use in strength resolution at inout ports. If Verilog-side signals always win during strength fighting or there is no strength fighting at inout ports, it is not necessary to introduce strength resistors.

The define_strength configuration command specifies the resistances of strength resistors to map to the Verilog seven strength levels. The syntax is shown in the following example.

```
define_strength strength_tbl -supply 10 -strong 100 \
     -pull 1000 -large 10000 -weak 100000 \
     -medium 1000000 -small 10000000
```

The resistance presents a Verilog strength relative to a lumped sum SPICE impedance at the bi-directional net. The SPICE impedance depends on the transistor model, technology, and process used in the design. Therefore, a default strength table will not satisfy the requirement because the relative resistances are both design-dependent and port-dependent.

Data flow direction must be available in order to select proper resistances. If the Verilog-side signal wins the strength fight, the strength resistor's resistance must be significantly smaller than SPICE-side impedance. Conversely, if the SPICE-side signal wins the strength fight, the resistance of the strength resistor must be significantly larger than the SPICE-side impedance. The following two examples show how this works:

Example 1

Assume the following for this example:

- Strength fighting occurs at port Y
- Simulation time is 10 ns
- The Verilog-side presents a weak logic0
- The SPICE-side has 3V and logic1 before strength resolution
- Data flows from the SPICE- to the Verilog-side at the 10 ns mark indicating that the SPICE-side driving strength is stronger.

In this example, the final value at port Y should be logic1.

If the impedance at the SPICE-side is 1000 Ohms, then the proper resistance of the strength resistor can be 10000 Ohms; in which case HSIM does the following:

- Resolves that the voltage at port Y to be 2.8V
- Sets port Y to logic1

In this case, the weak Verilog strength is mapped to a 1000 Ohm strength resistor.

Example 2

- Later, at simulation time 20 ns
- The Verilog-side presents a strong logic0 at port Y
- SPICE-side voltage is 3V with a driving impedance of 1000 Ohms before strength resolution
- Data flows from the Verilog- to SPICE-side
- The final value at port Y should be logic0

The proper resistance of the strength resistor may be 100 Ohms for HSIM to resolve the strength fight and produce a final value of 0.5V and a logic0. Therefore, strong Verilog strength is mapped to a 100 Ohm strength resistor.

The above two examples show why it is important to know the data flow direction in order to select proper resistance values. Designers specify strength resolution as follows:

- Full Verilog netlist: Data flow direction is specified by setting different verilog strength levels that drive the same net.
- Full SPICE netlist: Data flow direction is determined by different size transistors connecting to the same net.
- Co-Simulation netlist: Data flow information is not available. Signal strength tables are constructed from information provided by designers to determine data flow directions.

The report_port_resistance configuration command creates a report that details the SPICE-side resistance or impedance from interface nodes to voltage sources. Strength tables can be constructed from the data flow directions in a circuit design and the SPICE-side path resistance.

Note:

> The resistance difference between two consecutive Verilog strength levels can be one (1) order of magnitude such that if 100 Ohms is described a strong level then the pull level can be 1,000 Ohms.

Co-simulation with VCS

HSIM-VCS co-simulation provides a simulation solution in which HSIM uses the Verilog Procedural Interface (VPI) or Programming Language Interface (PLI) 2.0 to interact between VCS and HSIM. During the simulation, VCS functions as the digital simulator and HSIM functions as the analog simulator.

The signal interactions between the two simulators pass through the analog-to-digital and digital-to-analog interface elements. HSIM inserts the interface elements after you manually partition the design.

The co-simulation flow supports two verification methodologies::

- Verilog on top with some lower-level instances in SPICE
- SPICE on top with some lower-level instances in Verilog

For accurate simulation results and performance, the following basic guidelines are recommended for co-simulation:

- Avoid partitioning the design into numerous small analog and digital blocks.
- If possible, specify SPICE ports as input or output to reduce the number of interface elements for better performance.
- Avoid partitioning the digital and analog boundary at timing or voltage sensitive ports.

HSIM-VCS Co-simulation Usage Flow

For a graphical representation of the HSIM-VCS co-simulation usage flow, see Figure 1.

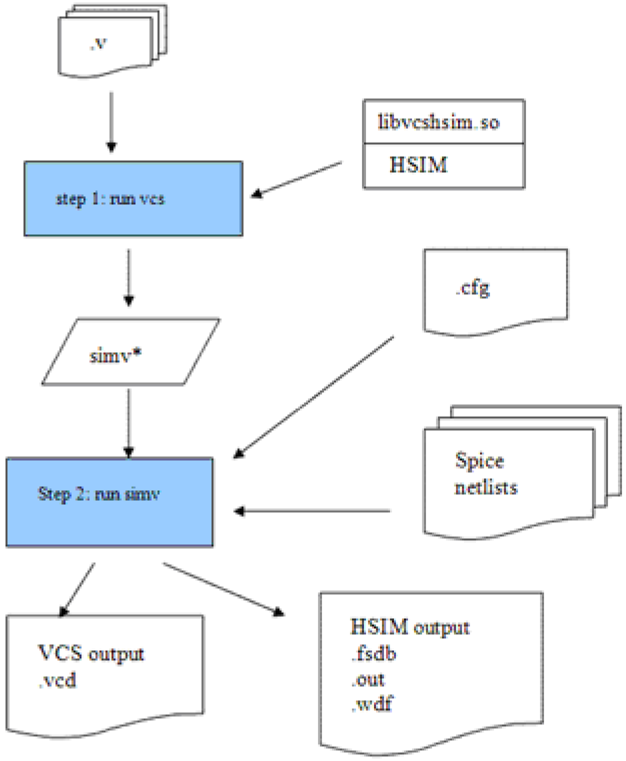


Figure 76 Figure 1: HSIM-VCS Co-simulation

Setting up System Environment Variables for Co-simulation

Before running the co-simulation, the system must first be set up by following these steps:

1. Follow standard HSIM installation and setup described in the *Synopsys Installation Guide*.
2. Set the VCS executable path:

```
setenv VCS_HOME /path/to/vcs_<version>
set path = ($VCS_HOME/bin $path)
```

`<version>` is the co-simulation qualified version of VCS. If you are uncertain as to the qualified version of VCS, contact your Application Consultant for assistance.

3. Add the directory containing the libvcshsim.so file, which is the HSIMplus installation directory, to LD_LIBRARY_PATH. This is the VPI shared library and is required for the co-simulation. This library is platform-dependent. See Platform Support for HSIM/VCS Co-Simulation on page 484.

Running the Designs with Co-simulation

The VPI-based HSIM-VCS co-simulation supports two verification methodologies:

- Verilog as the top instance
- SPICE as the top instance

The steps (and examples) for these two flows are described in the following sections.

Co-Simulation with Verilog as the Top Instance

To run co-simulation with Verilog as the top-level instance:

1. Create a co-simulation configuration file to specify the SPICE netlist with the model files, power supply, and print nodes used by the HSIM simulation. Use the `analog_cell` command to specify the block(s) to be replaced by the sub-circuit(s) and simulated by HSIM.

```
set_args     <netlist file> [other HSIM options]
analog_cell  <Verilog cell replaced by sub-circuit>
```

2. Use VCS to compile the Verilog netlists. The co-simulation configuration file is specified by adding the option “`-ad_hsim=<cosim_config_file>`”:

```
% vcs +vpi -ad_hsim=<cosim_config_file> -load
libvcshsim.so:cs_vpi_startup +cli+3 [verilog files and other
vcs options]
```

Note:

The `vcs` compile option `+cli+3` is required for the co-simulation.

Use the `-full64` option of vcs to compile and simulate designs on a 64-bit machine. Make sure your LD_LIBRARY_PATH is pointing to the path of 64-bit `libvcshsim.so` library, the `HSIM_64` environment variable for standalone HSIM does not have effect on HSIM shared library.

The HSIM-VCS link does not require you to manually replace the original Verilog module content with the `$nsda_module()` system task, and the compilation list of the Verilog files remains unchanged, hence, there is less manual effort for the setup.

The HSIM configuration file `hsim.ini` is also read during analog netlist parsing. HSIM options can be specified in the same way as in standalone HSIM simulation.

3. Start co-simulation by running the `simv` executable generated by step 2:

```
% simv +nsda+<cosim configuration file> [other options]
```

Example 1: This Verilog-top example is located in the HSIM installation under the tutorial directory:

```
tutorial/cosim/vcs_vpi_simple/verilog_top
```

The two (shaded) inverters, inv, are replaced by SPICE for the co-simulation. See the following schematic:

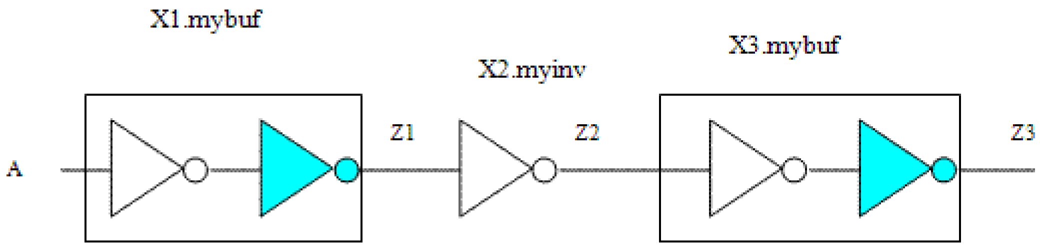


Figure 77 Schematics for Verilog Instance at Top

1. Prepare a co-simulation configuration file, `cosim.cfg`, to specify the HSIM netlist file and other configuration requirements. In this example, the HSIM netlist file is `test.spi` which includes `inv_sub.spi`. In `cosim.cfg`, use `analog_cell` to specify the Verilog instance inv to be replaced by sub-circuit.

   ```
   # cosim.cfg
   set_args      spice/test.spi
   analog_cell   inv
   ```

2. Use VCS to compile the Verilog source files, using the `-ad_hsim=<cosim_config_file>` option to specify the configuration file:

   ```
   % vcs +vpi -ad_hsim=cosim.cfg -load
   libvcshsim.so:cs_vpi_startup +cli+3 verilog/top.v verilog/
   inv.v
   ```

   Note:

   No change is needed on the compilation list of the Verilog files.

   Use the `-full64` option to compile and simulate designs on a 64-bit machine.

   The default executable file name generated by VCS compilation is `simv`, to use other names, use `-o <name>` option during compilation.

3. Run the `simv` executable to start the co-simulation:

   ```
   % simv +nsda+cosim.cfg
   ```

   Note:

   By default, HSIM generates output files in the current directory with the prefix “`hsim`”. To specify other paths/names, use the HSIM command-line option “`-o <path/name>`” on the `set_args` line in the co-simulation configuration file.

   Besides the HSIM log file, co-simulation also generates a `hsim.csintf` file that contains interface node information.

   The default analog waveform file is `hsim.fsdb`. To use other output formats, use the HSIM parameter HSIMOUTPUT to override the default.

   The digital waveform file is generated by VCS.

Co-Simulation with SPICE as the Top Instance

The original SPICE as top instance flow can still be used, see Co-Simulation with SPICE as the Top Instance on page 432. Unlike with third-party Verilog simulator, the HSIM-VCS link has a simplified flow for the SPICE-top flow that is described below:

1. In the co-simulation configuration file, use the `digital_cell` or `digital_cell_inst` command to specify the Verilog cell(s) or instance(s) used to replace SPICE sub-circuit blocks. Use the `verilog_file` command to specify the Verilog source file(s). The top-level Verilog `cosim.v` file is automatically generated during compilation by VCS and it is no longer necessary to run `hsim -cscfg <cosim_config_file>` to create it. However, this last method can still be used if you need to manually generate the `cosim.v` file and modify it before compiling it using VCS.

```
set_args              <netlist file> [other HSIM options]
digital_cell          <sub-circuit replaced by Verilog cell>
```

   or:

```
digital_cell_inst     <instance replaced by Verilog cell>
verilog_file          <Verilog source file>
```

2. Use VCS with the`-ad_hsim=<cosim_config_file>` option to compile the Verilog source files:

```
% vcs +vpi -ad_hsim=<cosim_config_file> -load
libvcshsim.so:cs_vpi_startup +cli+3 [verilog files and other
vcs options]
```

3. Start co-simulation by running simv generated by step 2:

```
% simv +nsda+<cosim_config_file>
```

Note:

After running the simplified flow of SPICE as top instance, a `sp2cs.log` file is generated, which represents the log file generated by the `hsim -cscfg <cosim_config_file>` command called by VCS to create the top-level `cosim.v` file on the fly.

It is recommended to use the simplified flow if you do not need to modify the `cosim.v` file.

In the simplified flow, do not include the `cosim.v` file in the compilation list for VCS. VCS automatically includes `cosim.v` after it is generated.

For SPICE as top instance flow, the only difference between the simplified flow and the standard one is the automatic generation and inclusion of the `cosim.v` file.

Example 2: This SPICE-top example can be found in your HSIM installation under the tutorial directory:

```
tutorial/cosim/vcs_vpi_simple/spice_top
```

The two shaded inverters are replaced by Verilog modules for the co-simulation, as shown in the figure below:

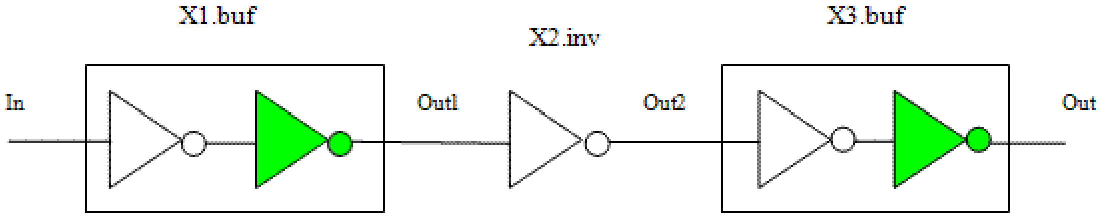


Figure 78 Figure 3: Schematics for SPICE Instance at Top

1. Create a `cosim.cfg` file, the top-level SPICE netlist is specified on the `set_args` line as `spice/test.spi`. The sub-circuit `invd` is replaced by the Verilog module defined in Verilog source file `verilog/invd.v`.

```
# cosim.cfg
set_args            spice/test.spi
digital_cell        invd
verilog_file        verilog/invd.v
```

2. Compile the Verilog files with `–ad_hsim=<cosim_config_file>` option:

```
% vcs -ad_hsim=cosim.cfg +vpi -load
libvcshsim.so:cs_vpi_startup +cli+3 verilog/invd.v
```

3. Run co-simulation with `simv`:

```
% simv +nsda+cosim.cfg
```

Summary of Commands

For both of the co-simulation flows, Verilog as top instance and SPICE as top instance, you use similar steps, the difference is in the co-simulation configuration file, which is specified with the VCS option during compilation:

```
-ad_hsim=<cosim_config_file>
```

Additional VCS options required for HSIM-VCS VPI co-simulation are:

```
+vpi -load libvcshsim.so:cs_vpi_startup +cli+3
```

The following commands are used in co-simulation configuration file:

- set_args on page 455
- analog_cell on page 448
- digital_cell on page 451
- digital_cell_inst on page 452
- verilog_file on page 462

Interactive Mode

All HSIM interactive mode commands are supported. See Co-Simulation Interactive Mode on page 463 for a complete list of commands.

To invoke HSIM in interactive mode, you have three methods to choose from:

- Specify the HSIM parameter to let HSIM stop at a certain time and show HSIM interactive mode prompt:

  ```
  .param hsimstopat=<time>
  ```

  OR

- Press CRTL-C during transient simulation. VCS prompt `cli_0>` appears:

  ```
  cli_0> $nsda_intr_mode
  HSIM>
  ```

  OR

- Add set_intr_mode 1 to the co-simulation configuration file. Press CRTL-C to enter HSIM interactive mode.

To continue simulation in interactive mode:

- At the HSIM prompt, enter `cont [time]`:

  ```
  HSIM> cont
  ```

- At the VCS prompt, enter the period character (.)

  ```
  cli_0> .
  ```

To exit co-simulation, type `quit` at the command-line prompt.

Limitations

Existing limitations of HSIM VPI co-simulation still apply. Currently the following features are not supported in HSIM VCS VPI co-simulation:

- Donut partitioning flow
- VCS GUI
- Save/restore
- VHDL

Platform Support for HSIM/VCS Co-Simulation

HSIM/VCS co-simulation supports the following platforms shared by the both of the releases of HSIM and VCS. The table below shows the platform-dependant path of `libvcshsim` to use. `LD_LIBRARY_PATH` (or `SHLIB_PATH` on HP-UX) must be set accordingly:

Platform	**O/S**	**Path**
Linux	Red Hat Enterprise Linux WS release 3, 32-bit	`$HSIM_HOME/platform/linux`
	Red Hat Enterprise Linux WS release 3, 64-bit	`$HSIM_HOME/platform/amd64`
SuSe	SUSE LINUX Enterprise Server 9 (x86_64), 32-bit	`$HSIM_HOME/platform/suse32`
	SUSE LINUX Enterprise Server 9 (x86_64), 64-bit	`$HSIM_HOME/platform/suse64`
Solaris	SunOS59, 32-bit	`$HSIM_HOME/platform/sparcOS5`

Co-Simulation with ModelSim

Verilog/VHDL is used with HSIM to co-simulate mixed Verilog/VHDL and SPICE-based designs that contain both digital and analog partitions. This is accomplished by using the Verilog/VHDL simulator to simulate the digital netlist; while HSIM simulates the analog SPICE netlist. When complete, analog and/or digital simulation results are available for designers to verify their designs.

In addition to Cadence NCSIM, co-simulation with Mentor Graphics ModelSim is now supported.

ModelSim/HSIM Integration

The libvpihsim.so co-simulation library supports ModelSim integration with either of the following:

- Stand-alone ModelSim
- In the ADMS environment

In both Verilog and SPICE design partitioning flows, most co-simulation features and limitations for NC-Verilog/VHDL are applicable to ModelSim. Since the cell view and Verilog configurations for instance based instantiation are not available in ModelSim, users need to modify the original Verilog source files to add the $nsda_module() system task in order to designate analog partitions.

Note:

save-restart is not supported in ModelSim/HSIM co-simulation.

Running ModelSim/HSIM Co-simulation with Stand-alone ModelSim

Use the ModelSim commands shown in the following steps to run ModelSim in a stand-alone ModelSim environment.

1. Create a design library using the following syntax. In this example, the design library will be named `work`.

   ```
   % vlib work
   ```

2. Compile the Verilog source code using the following syntax:

```
% vlog top.v
```

3. Use the -pli command line option to link libvpihsim.so into ModelSim for simulation using the following syntax. In this example, top is the name of the top design module.

   ```
   % vsim -c -pli libvpihsim.so +nsda+cosim.cfg top
   ```

   To enter HSIM interactive mode from ModelSim, press "Ctrl-C" during simulation to pause at ModelSim interactive prompt, and use the command:

   ```
   % vsim(paused) nsda_intr_mode
   ```

Running ModelSim/HSIM Co-simulation Under the ADMS Environment

Use the ADMS commands shown the following steps to run ModelSim/HSIM co-simulation under the ADMS environment.

1. Create a design library using the following syntax. In this example, the design library will be named `work`.

   ```
   % valib work
   ```

2. Compile the Verilog source code using the following ModelSim syntax. In this example, `-ms` invokes the ModelSim compiler:

   ```
   % valog top.v -ms
   ```

3. Invoke ModelSim to run the simulation using the following syntax. In this example, top is the name of the top design module.

   ```
   % vasim top -ms -pli libvpihsim.so
   ```

   Note:

   > The -c command line option does not work for co-simulation in batch mode.

HSIM Features Not Supported by Co-simulation

Co-simulation is designed for transient analysis. The following HSIM features and parameters are not supported:

- DC Analysis
- AC Analysis

- Monte Carlo Analysis
- Parameter Sweeping Analysis

References

[1] NC-Verilog/VHDL is a functional verification tool from Cadence Design Systems, Inc.

[2] Verilog-XL is functional verification tool from Cadence Design Systems, Inc.

12

Physical Visualization Manager (PVM)

Describes the Physical Visualization Manager (PVM), a graphical user interface for generating and visualizing violation maps.

The Physical Visualization Manager (PVM) provides a graphical user interface (GUI) for generating and visualizing violation maps from HSIM^{plus} PWRA and SIGRA.

PVM Installation

Installation of PVM files is completed during the HSIM installation process. For PVM to work properly with the Cadence® Virtuoso® Layout Editor, perform the following tasks:

1. Verify that the following directory is included in the PATH variable:

   ```
   <install_path>/hsimplus<version>/bin
   ```

2. Customize the Cadence system files as necessary using the files that are installed with PVM. These files have the following meanings:

   - pvmProcEnc.il: Contains procedures in SKILL[1], responsible for communication, and layout navigation.
   - pvmBindKeys.il: Example of bind keys mapping to functions from pvmProcEnc.il.
   - hsim.drf: Contains resources for drawing violation maps.

Installing PVM: No Existing Customized Cadence Tools

If users have not previously developed customized Cadence design tools, the following steps are required.

1. Copy the .cdsinit file from the <install_path>/hsimplus<version>/etc/pvm directory to the HOME directory. This will automatically load customization files.
2. Create a library directory to be used for storing all libraries (cells, views, etc.) for PVM.
3. Create a cds.lib file in the HOME directory. The file must contain the following instruction:

   ```
   INCLUDE <libraries dir>/cds.lib,
   ```

   where <libraries dir> is the directory for the libraries described in Step 2 above. Make sure you specify full library name.

   Note:

   An example file can be copied from the following directory: <install_path>/hsimplus<version>/etc/pvm.

Installing PVM: Existing Custom Cadence Tools

If an account has already been customized for Cadence tools, perform the following steps.

1. Load pvmProcEnc.il and pvm.drf to the .cdsinit file from <install_path>/hsimplus<version>/etc/pvm.
2. Perform one of the following operations:
   - Copy pvmProcEnc.il from <install_path>/hsimplus<version>/etc/pvm to the directory, which is within the Skill Search Path.
   - Add <install_path>/hsimplus<version>/etc/pvm directory to the Skill Search Path.
3. If required, change the Bind Key file for Virtuoso Layout Editor and Cadence Command Interpreter Window (CIW) to make quick calls of PVM functions.
4. Verify that the cds.lib file in the HOME directory contains reference to the library path to be used with PVM.

Using PVM

To use PVM, perform the following steps:

1. Start Virtuoso using the following command:

```
unix% layout &
```

2. Start PVM using the following command:

```
unix% pvm &
```

The PVM main window appears as shown in Figure 79.

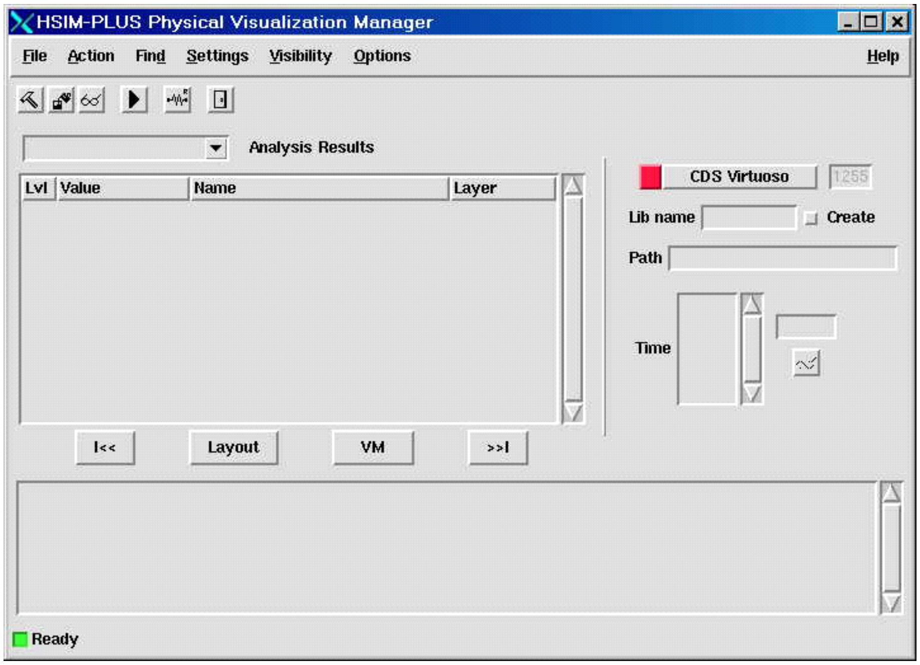


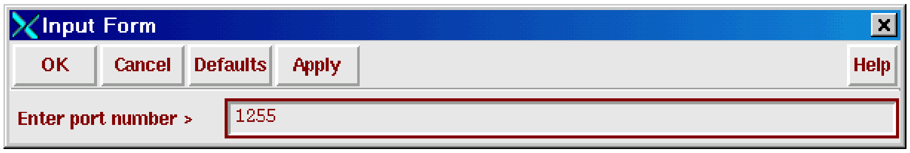
Figure 79 PVM Main Window

3. Establish a connection between PVM and Virtuoso by either of the following methods:

   - Method 1: Enter pvmStartVrtcli() in the CIW command line.
   - Method 2: Press Ctrl-D if the pvmBindKeys.il file was loaded as described in Installing PVM: No Existing Customized Cadence Tools on page 490. Virtuoso will open the dialog window shown in Figure 80. Next, enter the port number in the Enter port number > window.

Note:

The port number must be taken from the text box adjacent to the CDS Virtuoso screen button located in the PVM main window. Refer to Figure 79.



Enter port number > 1255

Figure 80 Port Setup Dialog Window

If a connection is successfully established the ready indicator at the bottom of the window shown in will change to green. Refer to Figure 79.

4. Open the reliability analysis (RA) database or the PVM data file using one of the following menu items:

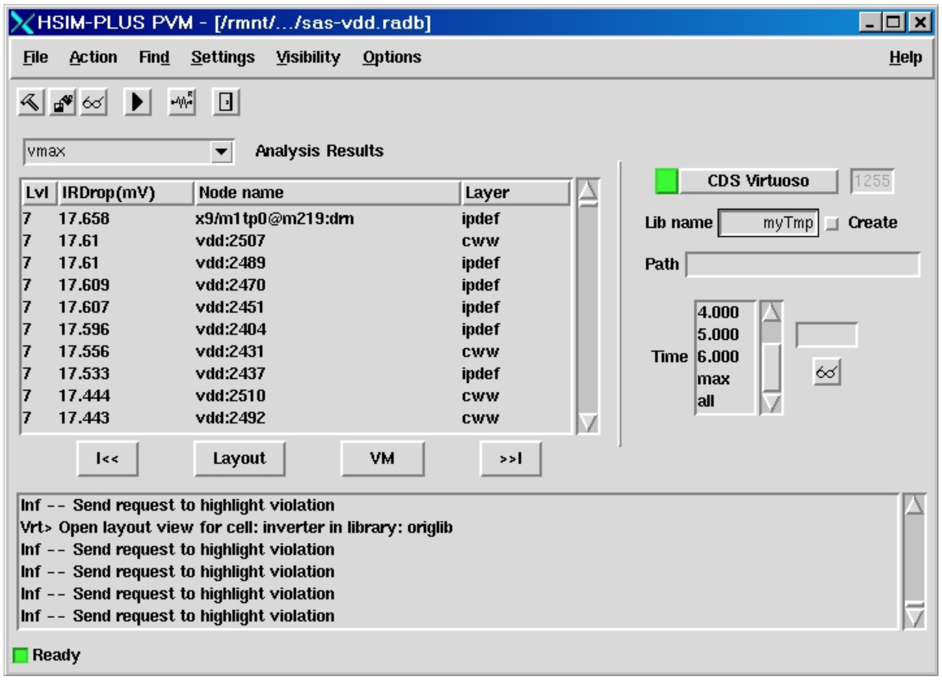
   - File/Open RA database: PVM opens a dialog window to select a file with a *.radb file extension.
   - File/Open: If data from a previous session was stored in the PVM's data file, open the file using the menu item. PVM opens the RA database and fills in all fields.

5. Set the working parameters by using the appropriate input fields, list box, and dialogs as follows:

   - Analysis: Analysis selects one of the analyses from the drop-down list. This list contains only the analyses completed during the second phase of simulation. If the list is empty, the *.radb file was incorrect or corrupted.
   - Lib name: Lib name specifies a library name (without a path) where the program should store the generated layout.
   - Path: the Path field is only used to create a new library. To create a new library, click in the Create check box and a.) select an option from the dialog box or b.) manually enter the path to the new library.
   - Time: The Time list automatically populates after actions such as Open File or Open RA database. The list contains the time points specified in the RATCL file before reliability analysis was performed. For these time

points, <Product Name> RA either: a) Generates additional information used to describe the violation map at a selected time point, or; b) dynamically displays changes in a violation map.

After completing the previous steps, the main window shown in Figure 81 appears.



vmax ▾ Analysis Results

Lvl	IRDrop(mV)	Node name	Layer
7	17.658	x9/m1tp0@m219:drn	ipdef
7	17.61	vdd:2507	cww
7	17.61	vdd:2489	ipdef
7	17.609	vdd:2470	ipdef
7	17.607	vdd:2451	ipdef
7	17.596	vdd:2404	ipdef
7	17.556	vdd:2431	cww
7	17.533	vdd:2437	ipdef
7	17.444	vdd:2510	cww
7	17.443	vdd:2492	cww

Lib name myTmp □ Create

Path

Time 4.000 5.000 6.000 max all

```
Inf -- Send request to highlight violation
Vrt> Open layout view for cell: inverter in library: origlib
Inf -- Send request to highlight violation
Inf -- Send request to highlight violation
Inf -- Send request to highlight violation
Inf -- Send request to highlight violation
```

🟩 Ready

Figure 81 *HSIM*plus *PVM Window*

The following actions can now be accomplished:

- Generate violation map layouts in GDSII format
- Load GDSII files into Virtuoso database
- Open loaded layout views
- Analyze violation maps

Generating a GDSII File

To generate GDSII files, perform the following tasks:

- Select an analysis type
- Use the Settings menu items specify the following:
  - Violation map parameters
  - Processing mode
  - Analysis value thresholds
  - Violation map output filters

    Action/Generate GDSII Button

- Select Action/Generate GDSII or push the Action/Generate GDSII tool bar button. PVM will prepare the control commands and run <Product Name> RA.
- All diagnostics are placed into the Notes window. Current actions are displayed in the status bar. The status indicator at the bottom of the window blinks red. When the indicator stops blinking and turns green, a Ready message appears and the next action can be performed.

Left & Right Buttons

|<< >>|

When the violation map generation process is complete, PVM will load the ASCII violation map into the analysis results table. The table contains the first page of the violation map. To navigate through pages, use the Left and Right buttons. If a violation map page is requested that is outside of the file boundaries, PVM displays a notification.

Loading a GDSII File

Action/Load GDSII Button

To load a GDSII file into the Virtuoso database, select Action/Load GDSII or push the Action/Load GDSII toolbar button.

- PVM prepares control information and runs pipe-in pipe-out (PIPO).
- All diagnostics are placed in the Notes window.
- The status indicator at the bottom of the CDS Virtuoso window blinks red. When the indicator stops blinking and turns green, a Ready message appears and the process is complete.

To load a violation map into a new library, click on the Create screen button and type in the full path to new library in the Path field. PVM then sends the Virtuoso request to create a new library prior to running PIPO.

Opening a View

VM Button

👓

To open the layout view corresponding to the generated violation map, click on the Action/Open View toolbar button. PVM sends a request to Virtuoso and opens a new window with the correct view.

Violation Map Analysis: Visualization

Localization and Highlighting

PVM can locate simulation results within a violation map layout.

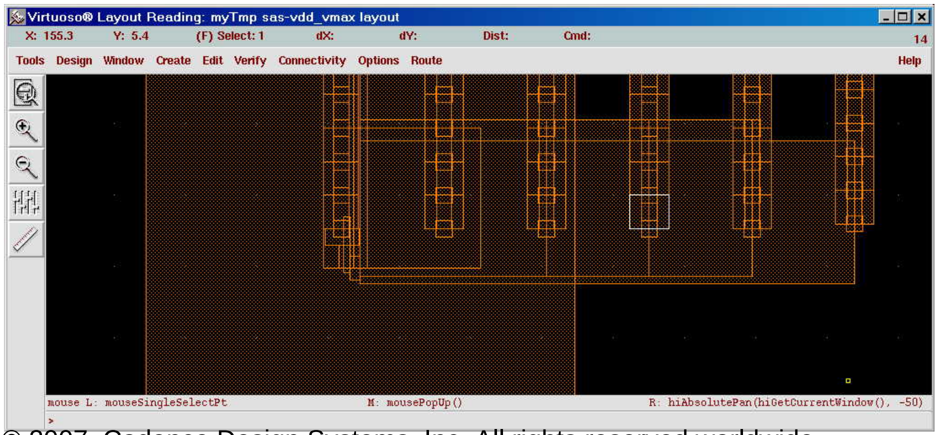
Action/Open View Button

VM

To find a shape corresponding to a particular IR drop result or EM analysis, perform the following tasks:

1. Select the desired result from the list.
2. Click on the VM screen button.

   PVM sends a request to Virtuoso. In the appropriate view, the requested shape will be highlighted as shown in Figure 82.



```
mouse L: mouseSingleSelectPt          M: mousePopUp()          R: hiAbsolutePan(hiGetCurrentWindow(), -50)
>
```

Figure 82   PVM Highlighted View

Zoom Settings

To zoom in or out, click the Right Mouse Button and select the applicable zoom setting from the menu shown in Figure 83.

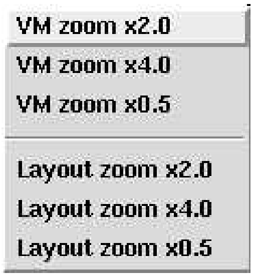


Figure 83   Zoom Pop Up Box

If the zoom setting is for a violation map, use the upper part of the menu shown in Figure 83. Each of these entries begin with VM.

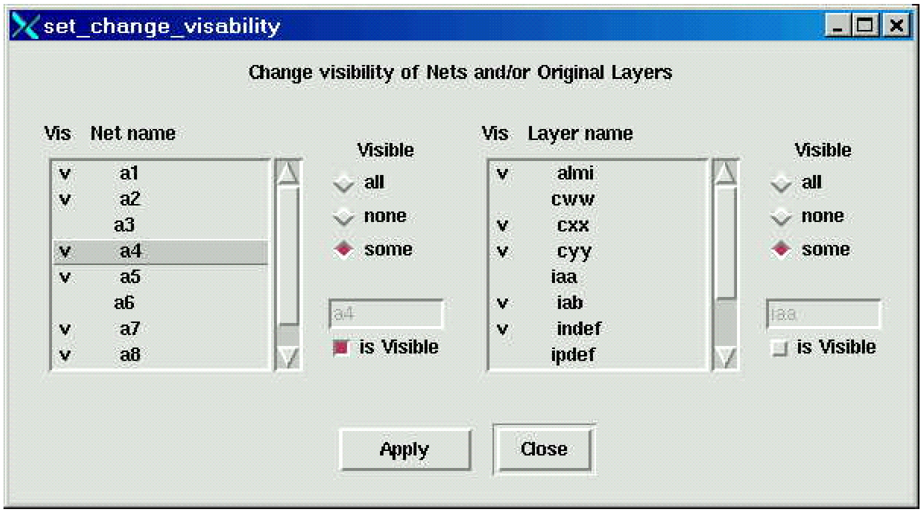
Visibility

Violation map layout visibility can be controlled using the Visibility/Nets&Layers options shown in Figure 84. PVM opens a dialog window consisting of two lists located side-by-side in the window. Each section contains information relating to the:

- Net name
- Layer name

Each of these net name lists consist of Net name list on the Left side of the screen and a Layer name list on the Right side of the screen. The two sections of the screen contain the following elements:

- Net Name List (Left): A scrolling list of net names and their visibility status.
- Layer Name List (Right): A scrolling list of layer names and their visibility status.
- Both Lists: Radio buttons for selecting whether to select all, none, or some elements from the list and an is Visible check box to select whether a specific net or layer is visible; shown in the text box immediately above the radio button.



set_change_visability

Change visibility of Nets and/or Original Layers

Vis	Net name
v	a1
v	a2
	a3
v	a4
v	a5
	a6
v	a7
v	a8

Visible
○ all
○ none
● some

a4
☒ is Visible

Vis	Layer name
v	almi
	cww
v	cxx
v	cyy
	iaa
v	iab
v	indef
	ipdef

Visible
○ all
○ none
● some

iaa
□ is Visible

Figure 84 Nets and/or Layers Visibility Window

To change the visibility status of all nets or layers, click on the all or none radio buttons. To change visibility status of a single object, do the following:

1. Click on the some radio button.
2. Perform either of the following operations:

- Select an object from either the Net name or Layer name lists. The selected object will appear in the is Visible window.
- Double click the Left mouse button within the desired list.

3. Apply the new visibility settings to the layout by clicking on the [Apply] screen button.

   PVM then processes the layout visibility changes specified in the set_change_visibility window. After the changes are applied, PVM displays the changes in the analysis.

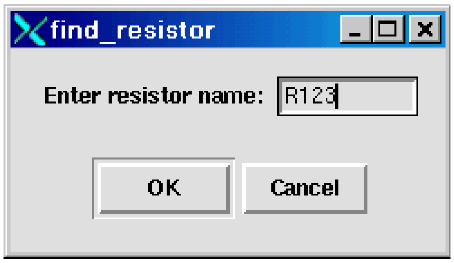
   Note:

   > The violation levels of invisible results from invisible tables in the GDSII cellView window will be displayed in Left and Right parenthesis characters ().

Find Resistor

Find/Resistor Button

To find a resistor by name, select Find/Resistor in the menu or click on the Find/Resistor tool bar button. The dialog window shown in Figure 85 appears.



find_resistor

Enter resistor name: R123

Figure 85 find_resistor Pop Up Box

Type the resistor name in the Enter resistor name: text box and click on the OK screen button. PVM sends a request to Virtuoso to find a shape with a name property equivalent to the specified resistor.

Dynamic Visualization

HSIM^{plus} RA generates a violation map for the maximum values calculated during the entire simulation cycle. To view the violation map for a specific time point, the gdstiming command must be specified in the RATCL file before performing reliability analysis. HSIM^{plus} RA generates time-specific layout information for PVM's use in performing dynamic visualization.

To view the visualization map for a specific time point, perform the following steps:

1. Select the desired time point from the time list shown in Figure 86.

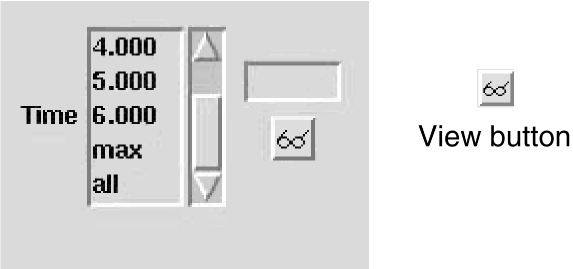


Figure 86 HSIMplus PVM Window: Time Window and View Button

2. Click on the View button. PVM sends Virtuoso a request to redraw the visualization map for the selected time point.
3. Select max from the Time list to view a visualization map of maximum values.
4. Press the View button.

To view the dynamic changes to VM, select all from the time list and press the View button. Virtuoso sequentially displays violation maps for all time steps.

Original Layout: Localization & Highlighting

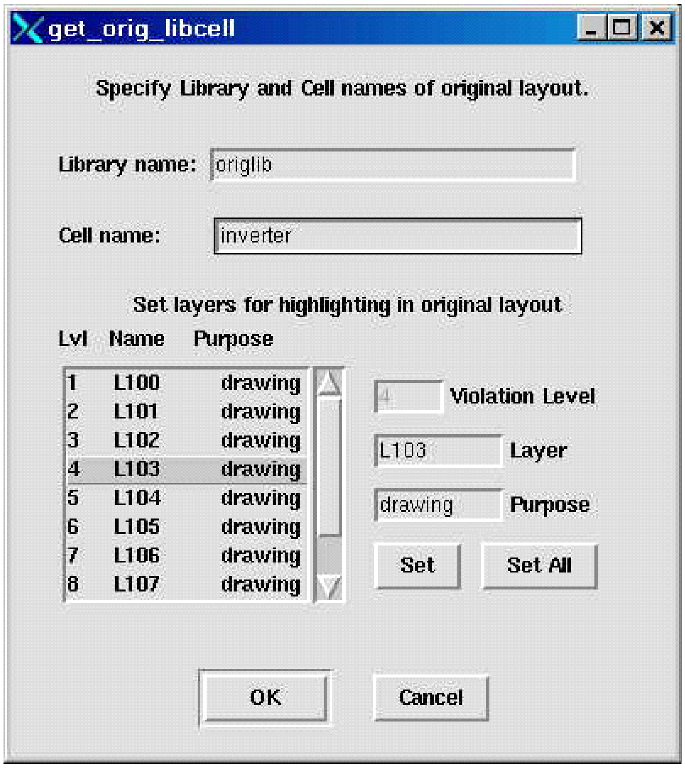
PVM will show the position of simulation results within the original layout by specifying the same information as the original layout including the following:

- Library name
- Top cell name
- Layers used for highlighting

You use the get_orig_libcell window, shown in Figure 87, to specify this information.

To specify the information, follow these steps:

1. To open the get_orig_libcell window, select Options/Original Layout.



get_orig_libcell

Specify Library and Cell names of original layout.

Library name: origlib

Cell name: inverter

Set layers for highlighting in original layout

Lvl	Name	Purpose
1	L100	drawing
2	L101	drawing
3	L102	drawing
4	L103	drawing
5	L104	drawing
6	L105	drawing
7	L106	drawing
8	L107	drawing

Violation Level: 4

Layer: L103

Purpose: drawing

Figure 87 get_orig_libcell Pop Up Box

2. To specify library and cell names, enter the appropriate information in the Library name and Cell name fields.
3. PVM uses the layers specified in Settings/Violation Map dialog as the default highlighting specification. To change the default layer and purpose names:

   a. Select the desired violation level from the list

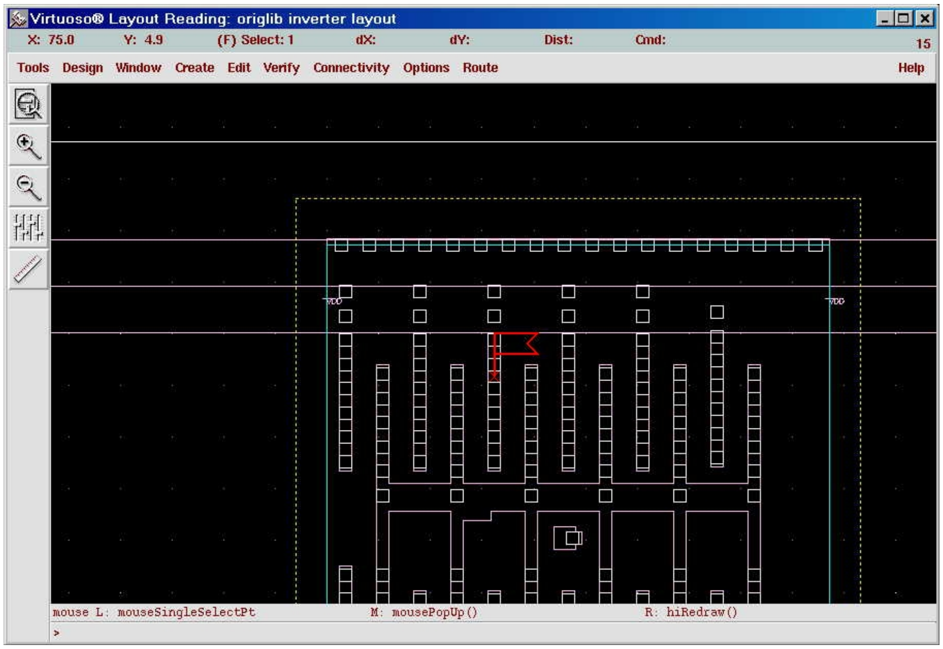
   b. Enter the appropriate data in the Layer and Purpose fields.

   c. Click on the [Set] screen button. To set the same layer for all levels of violation click [Set{All] screen button.
4. To highlight the result in the original layout, perform the following:

a. Select the result in the Analysis Results table shown in Figure 88.

b. Click on the [Layout] screen button.

PVM will automatically open the original cellView layout if it is not already opened. A PVM request is sent to Virtuoso causing a flag to appear in the location corresponding to the selected violation.


Virtuoso® Layout Reading: origlib inverter layout

X: 75.0 Y: 4.9 (F) Select: 1 dX: dY: Dist: Cmd: 15

mouse L: mouseSingleSelectPt M: mousePopUp() R: hiRedraw()

© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 88 Analysis Results Window

Refer to Zoom Settings on page 496 for information on zooming in or out of a layout during the localization process.

PVM Graphical User Interface (GUI)

File Menu

The File menu is used to work with PVM files (*.ldm). containing user specified parameters and options described in Table 17

Table 17 File Menu List

Action	Description
Open RA Database	Opens the RA database and loads the simulation results.
Open	Select File Open from dialog and click on the OK screen button. All parameters from the file will be used to populate the appropriate fields. If a parameter is not in the file, the corresponding field will be empty.
Save	Use File Save to save files that are already open. If no files were opened, PVM uses Save As...
Save As	Use File Save As to insert working parameters into a new PVM file. A dialog box opens to select a directory and specify a file name. If a file extension is not specified, PVM automatically adds the appropriate extension.
Exit	Use File Exit to exit PVM

Action Menu

The Action menu is used to perform one or all of the following actions:

- Generate GDSII
- Load GDSII into Virtuoso database
- Open the layout view

Table 18 describes the operations contained in the Action menu.

Table 18 Action Menu List

Action	Description
Generate GDSII	Use Generate GDSII to generate a violation map layout for a specified: RA database, analysis type, threshold value, or other parameters.
Load GDSII	Use Load GDSII to load a violation map layout into the Virtuoso database. The violation map layout will be stored as a layout view of the cell named <radb name>_<analysis>.
Open View	Use Open View to open a pre-loaded layout view.
Make All	Use Make All to perform the previous three actions. PVM will control the sequence of operations as necessary.

Find Menu

The Find menu performs the search described in Table 19.

Table 19 Find Menu List

Action	Description
Find resistor	Use Find resistor to search for a resistor's geometry by its name. PVM sends a request to Virtuoso. If the geometry of the required resistor is found, Virtuoso displays it as a selected rectangle in the center of a magnified window.

Settings Menu

The Settings menu options let you set GDSII generation parameters. The Settings menu contains the following options:

- Layers
- Modes
- Properties
- Reference

- Violation Map
- Thresholds...
- Filtering...

Layers

The Layers menu option displays the set_layers_info dialog box, shown in Figure 89 on page 505. You use this dialog box to specify additional information from the original layers, including:

- Layer height for interconnect layers
- Effective area for vias

You can specify independent values for each layer, and set default values for layers that have no settings.

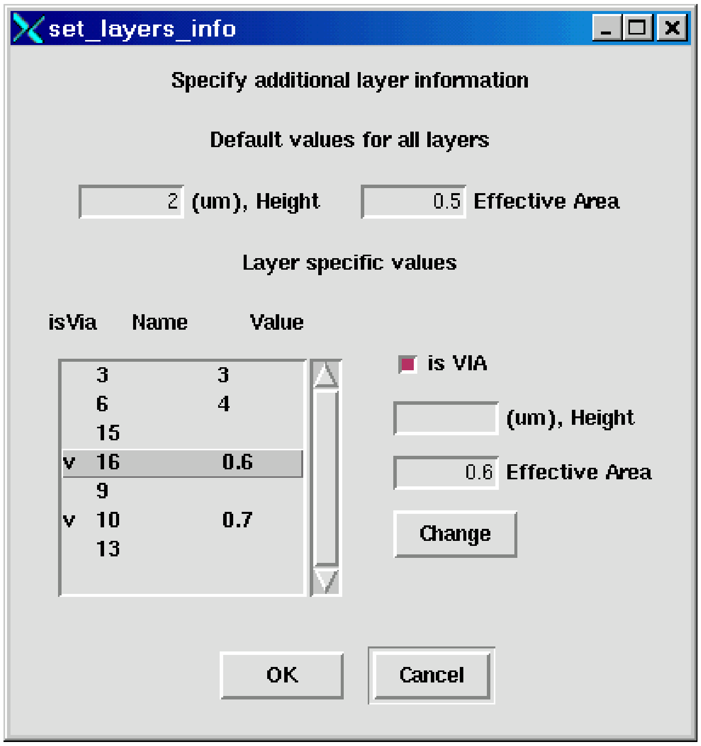
To set values for a layer, perform the following:

1. Choose Settings > Layers to display the set_layers_info dialog box.
2. Select a layer from the layers list. The layers list contains the following scrolling menus:
   - isVia
   - Name
   - Value
3. Choose a layer type.

   Note:

   For a VIA, select the is VIA check box.
4. Fill in the appropriate field(s) and click the Change button.

All changes are immediately reflected in the layers list as shown in Figure 89.



set_layers_info

Specify additional layer information

Default values for all layers

2 (um), Height   0.5 Effective Area

Layer specific values

isVia	Name	Value
	3	3
	6	4
	15	
v	16	0.6
	9	
v	10	0.7
	13	

■ is VIA

(um), Height

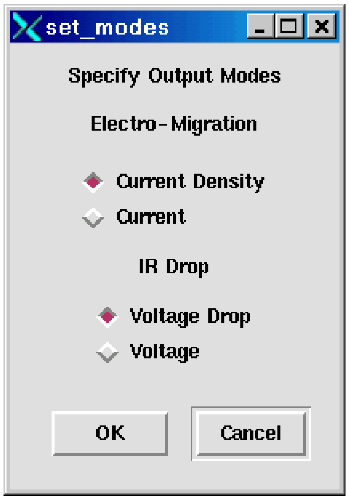
0.6 Effective Area

Figure 89 set_layers_info Pop Up Box

Modes

The Modes menu option displays the set_modes dialog box, shown in Figure 90. You use this dialog box to specify the output mode:

- Electro-Migration Analysis: Select either Current Density or Currents
- IR Drop Analysis: Select Voltage Drop or Voltage



set_modes

Specify Output Modes

Electro-Migration

● Current Density
○ Current

IR Drop

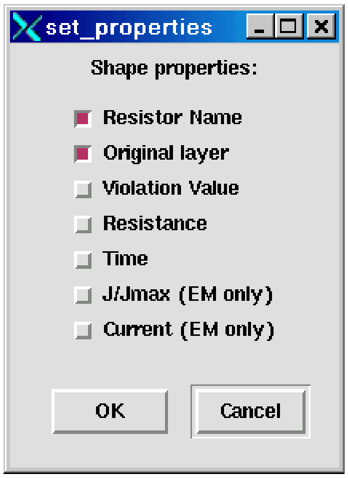
● Voltage Drop
○ Voltage

Figure 90 set_modes Pop Up Box

Properties

The Properties menu option displays the set_properties dialog box, shown in Figure 91. You use this dialog box to specify the information to be dumped to GDSII as shape properties:

- Resistor Name: The unique resistor identification name.
- Original Layer: The original layer in which the register resided.
- Violation Value: A value obtained from appropriate analysis in accordance with output mode. For example it may be actual voltage on some node for IR drop analysis or current density through resistor for EM analysis.
- Resistance: Total resistance.
- Time: The time point when the Violation Value occurred.
- J/Jmax: The ratio of current density to maximum current density for a specific resistor and original layer.
- Current: The actual current through a resistor.



Shape properties:

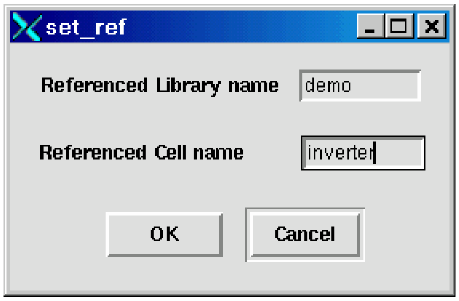
- ☒ Resistor Name
- ☒ Original layer
- □ Violation Value
- □ Resistance
- □ Time
- □ J/Jmax (EM only)
- □ Current (EM only)

Figure 91 set_properties Pop Up Box

Reference

The Reference menu option displays the set_ref dialog box, shown in Figure 92. You use this dialog box to specify the referenced library and cell names. These names are needed for overlapping a violation map over the original layout as follows:

- Referenced Library name: Library name in the original layout.
- Referenced Cell name: Top cell name in the original layout.



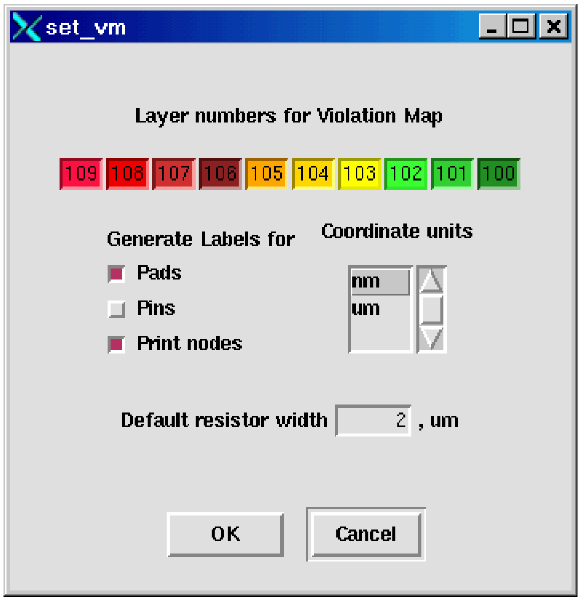
Referenced Library name: demo

Referenced Cell name: inverter

Figure 92 set_ref Pop Up Box

Violation Map

This Violation Map menu option displays the set_vm dialog box, shown in Figure 93. You use this dialog box to specify the general parameters used to generate a violation map.



set_vm

Layer numbers for Violation Map

109 108 107 106 105 104 103 102 101 100

Generate Labels for

☑ Pads

□ Pins

☑ Print nodes

Coordinate units

nm
um

Default resistor width 2 , um

Figure 93 set_vm Pop Up Box

The layer numbers specified in the violation map menu are used to generate a violation map layout. Since PVM merges layer information for different cell views within the same library, using overlapping layers may cause the redefinition of a layers display parameters and result in confusing situations.

Caution!

> Using only one layer setting for one library is recommended to avoid misleading results.

To specify the general parameters used to generate a violation map, follow these steps:

1. Specify whether text labels are to be used by selecting the appropriate options by clicking on one or more of the three Generate Labels for check boxes. The labeling criteria are described in the following bullets.
   - Pads: Terminals specified in SPF file as *|P
   - Pins: Terminals specified in SPF file as *|I

- Printed nodes: Node names specified in the PRINTV, PRINTI or PRINTIPAD commands

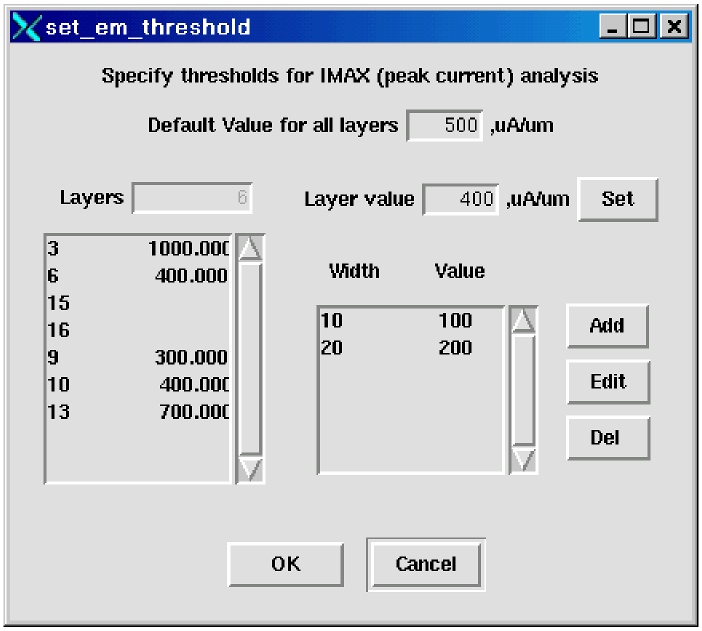
2. Coordinate units may be stated in nanometers (nm - default) or micrometers (Tm) and tells PVM what coordinates units are used in SPF files.
3. The default resistor width is used for resistors having no width information in the SPF files.

   Note:

   For EM analysis, default resistor width is not used; it is only used for visualization purposes.

Threshold

The Threshold menu option lets you specify thresholds for various types of analysis. All EM analyses use the set_em_threshold dialog box, shown in Figure 94.



set_em_threshold

Specify thresholds for IMAX (peak current) analysis

Default Value for all layers 500 ,uA/um

Layers 6

Layer value 400 ,uA/um

3	1000.00
6	400.000
15	
16	
9	300.000
10	400.00
13	700.00

Width	Value
10	100
20	200

Figure 94 set_em_threshold Dialog Box

The EM analysis dialog box contains the following elements:

- The Default threshold value used for all layers is used for layers having no specific value. If this value is not set, PVM uses maximum current density as a threshold.
- The original layers list is presented in the left-most scrolling list. The data are extracted from resistor geometry information taken from the SFP files.
- The Layers text box is the disabled entry field for displaying of selected layer.
- Layer value shows the specific threshold value.
- The Set screen button is used to change the layer specific threshold value.
- The Width_Value scrolling list shows the width and corresponding thresholds for the selected layer.
- The Add, Edit, and Del screen buttons create or modify specific threshold widths.

  Note:

  Width-specific thresholds may be used only for non-VIA layers.

Setting Width-Specific Thresholds To set width-specific thresholds, values for the layer’s upper threshold current and maximum width must be input. This is accomplished using the following steps:

1. Click on the Add screen button shown in Figure 94. The set_width_threshold dialog window shown in Figure 95 opens containing the following fields:
   - Width: Width is specified in micrometers (Tm)
   - Threshold: Threshold is specified in microamperes (TA) divided by micrometers (Tm)

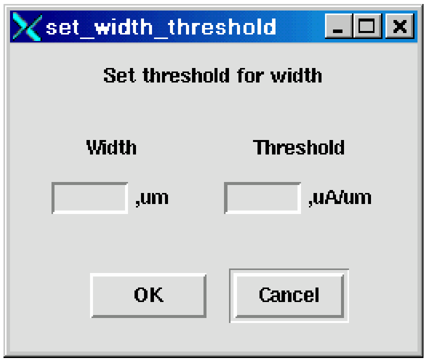


Figure 95 set_width_threshold Pop Up Box

2. Insert the required values in the Width and Threshold fields
3. Click on the OK screen button.

The new interval is added to the thresholds.

Example of setting width-specific thresholds

In Figure 94, the layer threshold is 400 and there are two widths with the following thresholds: 10 - 100 and 20 - 200. In this example, layer 6 will have 3 threshold intervals as follows:

- Widths up to 10 (um) have a 100 (uA/um) threshold
- Widths between 10 - 20 (um) have a 200 (uA/um) threshold
- Widths of 20 - max (um) have a 400 (uA/um) threshold

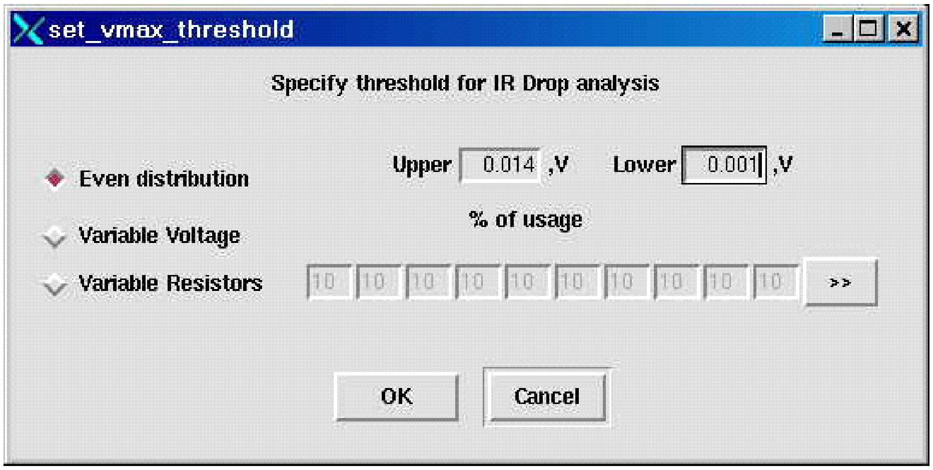
Note:

The width threshold value must be lower than the layer-specific threshold.

IR Drop Threshold The IR Drop threshold may be specified in one of three methods:

- Uniform distribution among 10 violation levels
- Variable voltage distribution
- Variable resistors distribution

To select the voltage distribution type, click on one the appropriate radio button on the left side of the window as shown in Figure 96.



set_vmax_threshold

Specify threshold for IR Drop analysis

● Even distribution

○ Variable Voltage

○ Variable Resistors

Upper 0.014 ,V Lower 0.001 ,V

% of usage

10	10	10	10	10	10	10	10	10	10

Figure 96 set_vmax_thershold Pop Up Box

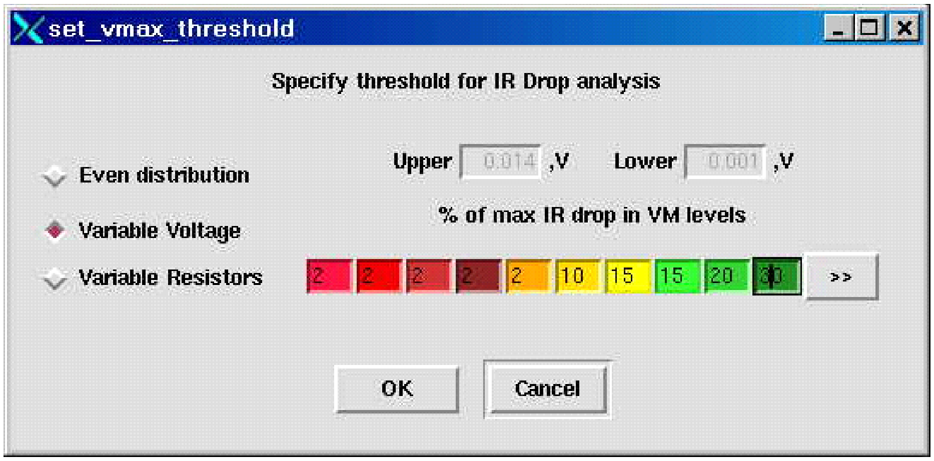
Note:

> In the following paragraphs on uniform and variable voltage distribution, the violation levels being discussed are represented in the ten information windows to the right of the Variable Resistors radio button in Figure 96 (inactivated) and Figure 97 (activated). When discussing level numbers, the information windows are counted from left to right.

Uniform Voltage Distribution Figure 96 provides an example of uniform voltage distribution where the Upper and Lower boundaries can be specified when generating a violation map. The result of specifying the upper and lower boundaries in this case is:

- All nodes with a voltage drop lower than 0.001 V will be placed into the least violated level.
- All nodes with voltage drop more than 0.014 V will be in the most violated level and displayed in blinking red color.
- All others are uniformly distributed among the other 8 violation levels.

Variable Voltage Distribution Figure 97 shows an example where the Variable Voltage radio button is activated.



Specify threshold for IR Drop analysis

◇ Even distribution

◆ Variable Voltage

◇ Variable Resistors

Upper 0.014 ,V Lower 0.001 ,V

% of max IR drop in VM levels

2 2 2 2 2 10 15 15 20 30

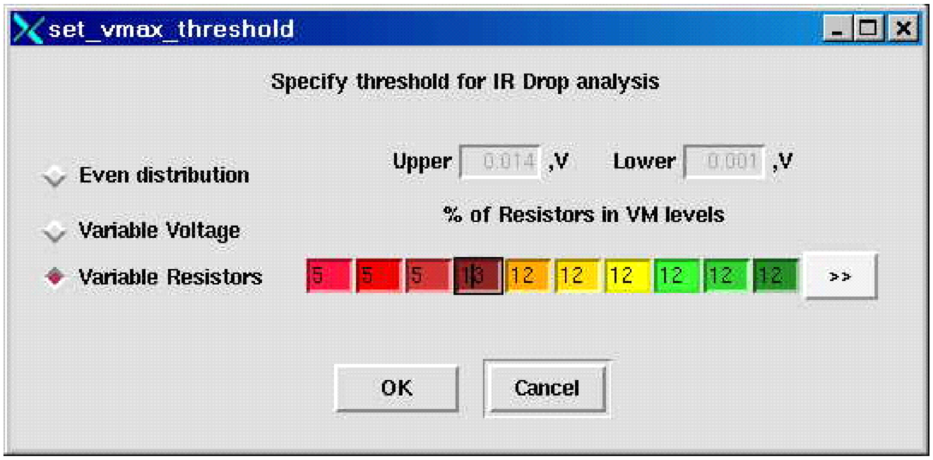
Figure 97 set_vmax_thershold Pop Up Box with Variable Voltage Selected

The voltage percentage within each level of violation may be set assuming that the maximum voltage drop is 100%.

Table 20 Voltage Drop Scale

Level	Voltage Drop
10th	98mV - 100mV (Most Violated)
9th	96mV - 98mV
8th	94mV - 96mV
7th	92mV - 94mV
6th	90mV - 92mV;
5th	80mV - 90mV
4th	65mV - 80mV
3rd	50mV - 65mV
2nd	30mV - 50mV
1st	0mV - 30mV (Least Violated)

Variable Resistors Distribution Figure 98 provides an example where the Variable Resistors radio button is selected. This method uses a variable distribution of resistors to create a violation map showing a clear picture of the IR drop gradient. In this example, the percentage of resistors within each violation level are set.



set_vmax_threshold

Specify threshold for IR Drop analysis

○ Even distribution   Upper 0.014 ,V   Lower 0.001 ,V

○ Variable Voltage   % of Resistors in VM levels

● Variable Resistors   5 5 5 13 12 12 12 12 12 12 >>

OK   Cancel

Figure 98 set_vmax_thershold Pop Up Box with Variable Resistors Selected

The Right arrow button (located at the right end of the levels line) is used to quickly set values. Values for some levels can be set while requesting recalculation of all other values.

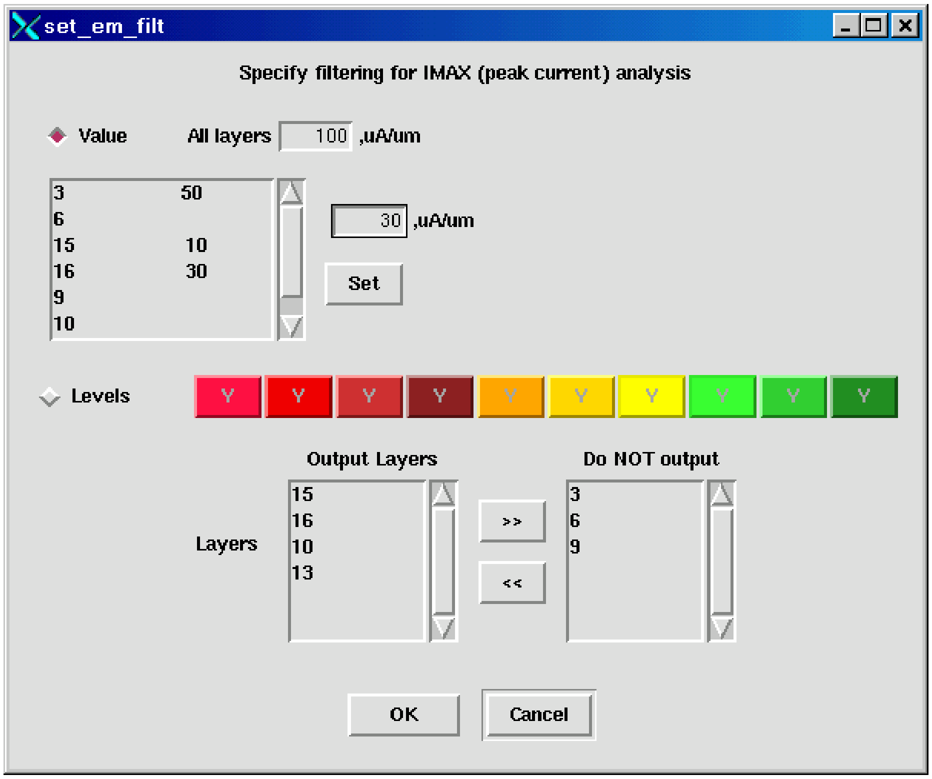
If 5% of all resistors are placed in the 3 most violated levels, all other resistors may be uniformly distributed among other 7 levels. To accomplish this, perform the following steps:

1. Enter 5 in the three left-most fields
2. Enter 0 into 4th field.
3. Click on the Right screen button.

   The Program will calculate and fill all of the fields as shown in Figure 98.

EM Analysis Filtering EM analyses uses the dialog window shown in Figure 99in to perform any of the following filter types:

- Filter by violation value
- Filter by violation level
- Filter by layer name



set_em_filt

Specify filtering for IMAX (peak current) analysis

● Value All layers 100 ,uA/um

3	50
6	
15	10
16	30
9	
10	

30 ,uA/um

○ Levels

Layers

Output Layers: 15, 16, 10, 13

Do NOT output: 3, 6, 9

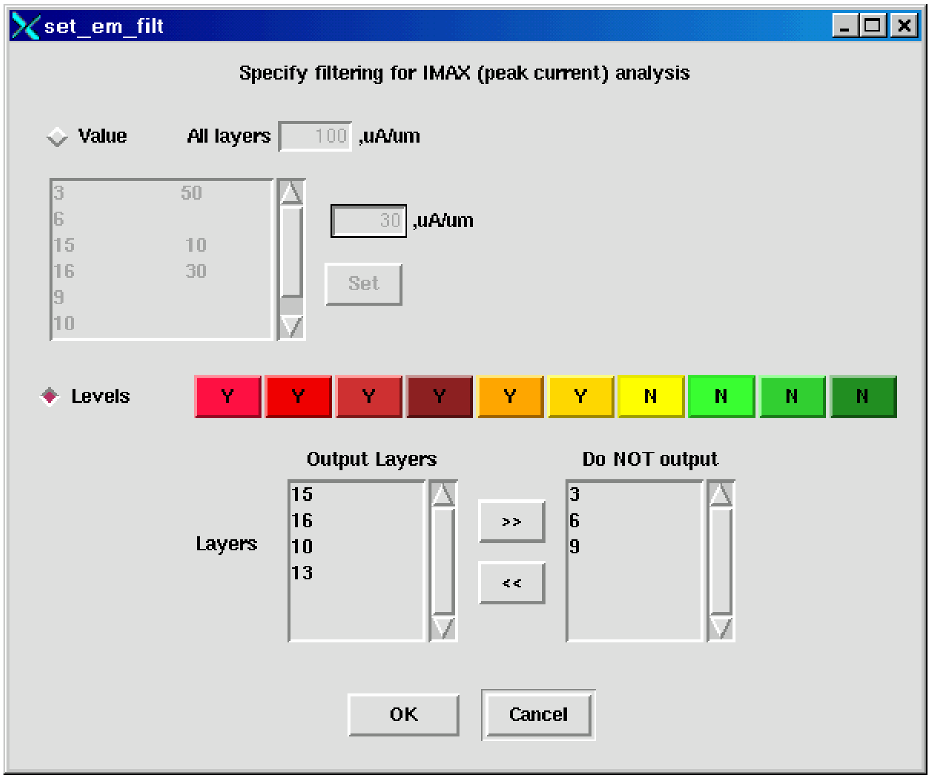
Figure 99 set_em_filt Pop Up Box

Violation Value Filtering Figure 99 illustrates filtering a violation value by performing the following steps:

1. Select the Value radio button.
2. Specify the default value for All layers.
3. Specify values for particular layers.

PVM will put only those resistors in the GDSII file that have violation values greater than the filtered value. If a resistor belongs to a layer that has no filtered value, PVM uses the default value for All layers. If neither a layer value nor a default value are set, the violation map will not be filtered.

Violation Level Filtering Figure 100 illustrates violation level filtering.



Specify filtering for IMAX (peak current) analysis

○ Value   All layers 100 ,uA/um

3	50
6	
15	10
16	30
9	
10	

30 ,uA/um

● Levels   Y Y Y Y Y Y N N N N

Layers

Output Layers	Do NOT output
15	3
16	6
10	9
13	

Figure 100 set_em_filt Pop Up Box with Levels Selected

Clicking on the Y/N screen buttons switches between Y and N to determine how levels are to be treated. When the Levels radio button is selected, the levels to be inserted in the GDSII file can be specified. Levels marked with a Y or N are treated as follows:

- Y: Placed into the GDSII file.
- N : Filtered out.

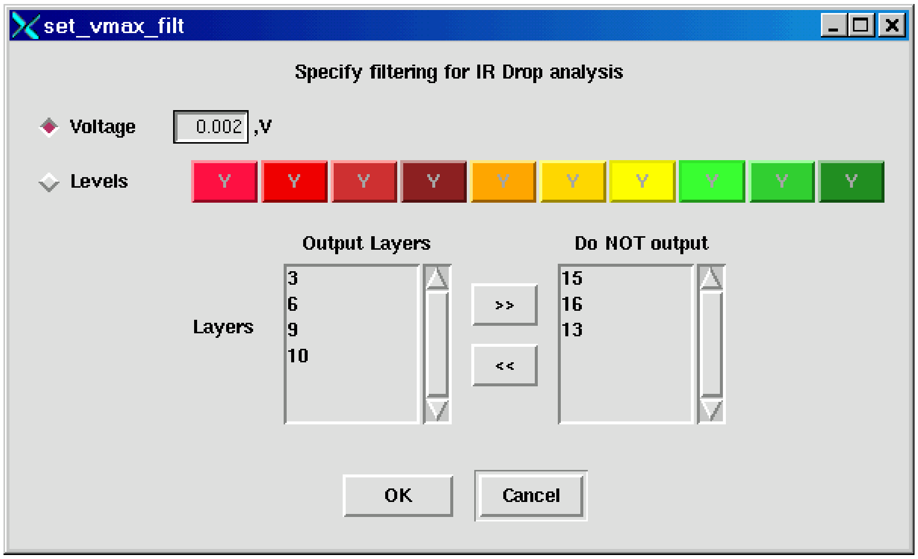
Note:

In Figure 100, only the 6 most violated levels will be output into the GDSII file.

Layer Name Filtering For both violation value and violation level filtering, the original layers may also be filtered. This is accomplished by selecting the desired layer from either the Output Layers or Do NOT output scrolling lists and click on the Right or Left screen button.

To forbid outputting layer 15, click on 15 in Output Layers and click on the Right screen button. Layer 15 will move to the Do NOT output list.

IR Drop Analysis Filtering IR Drop filtering is accomplished using a dialog widow similar to EM filtering as shown in Figure 101 on page 517. The only difference is that there is no layer specific filtering.


set_vmax_filt

Specify filtering for IR Drop analysis

◆ Voltage 0.002 ,V

◇ Levels

Layers

Output Layers	Do NOT output
3	15
6	16
9	13
10	

Figure 101 set_vmax Pop Up Window with Voltage Selected

Visibility Menu

The Visibility menu list descriptions are shown in Table 21.

Table 21 Visibility Menu List

Action	Description
Visibility/Nets&Layers	Visibility/Nets&Layers controls the visibility of a violation map layout. Selecting this menu item opens a dialog window consisting of two parts: net visibility and layer visibility. Either of both net(s) and layers may be selected and should be visible within the violation map layout. Refer to Violation Map Analysis: Visualization on page 495 for a usage details.

Options Menu

The Options menu list descriptions are shown in Table 22.

Continuation Button

Table 22 Options Menu List

Item	Description
Lib&Cell	Lib&Cell specifies the library and top cell names of the original layout. PVM uses these names to open a layout for localization of violations.
Tools	Tools opens a dialog window to specify the tools needed to generate a violation map and load the map layout into a Virtuoso database. PVM provides data fields with default names. To use this feature, select the Continuation screen button and use the standard dialog to find the location of tools. NOTE: The violation map generation tool must be called through a wrapper. Otherwise there may be problems with TCL/TK shared libraries.
Results	Results specifies the number of violation records to be shown in the violations list. The number must be between 10 and 10000 records.

Toolbar

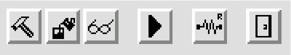


Figure 102 Toolbar

The tool bar shown in Figure 102 contains the following screen buttons:

Button Icon	Description
🔨	Action/Generate GDSII
	Action/Load GDSII
👓	Action/Open View
▶	Action/Make All
	Find/Resistor
🚪	File/Exit

Log Notes

This Log Notes area of application window displays notes generated by PVM and other programs used by PVM, including:

- PIPO
- Virtuoso Layout Editor

PVM notes have the following legend:

- Inf: Information
- Wrn: Warning
- Err: Error

Notes from all other applications, errors, Warnings, and other essential information from PVM are stored in the pvm.log file.

Status bar

The status bar displays the current PVM status. A status indicator in the top left corner changes color to indicate the following status conditions:

- Green: PVM is ready for any operation.
- Blinking Red: PVM is busy and you should wait for the completion of operation(s).
- Red: Program stopped. It should be killed.

The status bar also contains textual information about the current operation.

References

[1] SKILL is a Cadence extension language.

Part: 13 HSIM^{plus} Appendices

14

HSIM-Virtuoso Interface Netlist Properties

The appendix provides a list of the HSIM-Virtuoso Interface netlist properties and provides syntax examples and explanations.

HSIM Netlist Properties

The table below describes the netlist properties associated with HSIM.

Table 23 HSIM Netlist Properties

Primitive	Function	Default Parameter
res	netlistProcedure:	HSIMCompPrim
	instParameters:	r tc1 tc2 scale m dtemp l w
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	res
	namePrefix:	R
presistor	netlistProcedure:	HSIMCompPrim
	instParameters:	r tc1 tc2 scale m dtemp l w
	termOrder	PLUS MINUS
	propMapping:	
	componentName:	presistor

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	R
cap	netlistProcedure:	HSIMCompPrim
	instParameters:	c tc1 tc2 scale m dtemp ic l w
	termOrder:	PLUS MINUS
	propMapping:	cap
	componentName:	
	namePrefix:	C
pcapacitor	netlistProcedure:	HSIMCompPrim
	instParameters:	c tc1 tc2 scale m dtemp ic l w
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	pcapacitor
	namePrefix:	C
ind	netlistProcedure:	HSIMCompPrim
	instParameters:	l tc1 tc2 scale m dtemp ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	ind
	namePrefix:	L
mind	netlistProcedure:	HSIMCompPrim
	instParameters:	k

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder:	
	propMapping:	
	componentName:	mind
	namePrefix:	K
vdc	netlistProcedure:	HSIMSrcPrim_vdc
	instParameters:	vdc
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
idc	netlistProcedure:	HSIMSrcPrim_idc
	instParameters:	idc m
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vpulse	netlistProcedure:	HSIMSrcPrim_vpulse
	instParameters:	v1 v2 td tr tf pw per
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	V
ipulse	netlistProcedure:	HSIMSrcPrim_ipulse
	instParameters:	i1 i2 td tr tf pw per
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vpwl	netlistProcedure:	HSIMSrcPrim_vpwl
	instParameters:	rpt td tvpairs t1 v1 t2 v2 t3 v3
	termOrder:	PLUS MINUS
		t4 v4 ... t50 v50
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
ipwl	netlistProcedure:	HSIMSrcPrim_ipwl
	instParameters:	rpt td tvpairs t1 i1 t2 i2 t3 i3
		t4 i4 ... t50 v50
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
vsin	netlistProcedure:	HSIMSrcPrim_vsin
	instParameters:	vo va freq td theta
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
isin	netlistProcedure:	HSIMSrcPrim_isin
	instParameters:	io ia freq td theta
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vexp	netlistProcedure:	HSIMSrcPrim_vexp
	instParameters:	v1 v2 td1 td2 tau1 tau2
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
iexp	netlistProcedure:	HSIMSrcPrim_iexp
	instParameters:	i1 i2 td1 td2 tau1 tau2

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vcvs	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hegain maxv minv
		delta htd xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	refTermOrder:	NC+ NC-
	propMapping:	nil max maxv min minv td htd
	componentName:	vcvs
	namePrefix:	E
vccs	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hggain maxi mini delta xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	refTermOrder:	NC+ NC-
	propMapping:	nil max maxi min mini
	componentName:	vccs
	namePrefix:	G
ccvs	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hhgain vref maxv minv
		delta xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	propMapping:	nil max maxv min minv
	componentName:	ccvs
	namePrefix:	H
cccs	netlistProcedure:	
	instParameters:	HSIMCompPrimRef
		delta xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	propMapping:	nil max maxi min mini
	componentName:	cccs
	namePrefix:	F
tline	netlistProcedure:	HSIMCompPrim
	instParameters:	z0 td len f nl
	termOrder:	IN+ IN- OUT+ OUT-
	propMapping:	nil l len
	componentName:	tline
	namePrefix:	T
diode	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
pdiode	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
zener	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
schottky	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	nil vd Vd
	componentName:	diode
	namePrefix:	D
npn	netlistProcedure:	HSIMDevPrim
	instParameters:	area areab areac m
	termOrder:	C B E
	propMapping:	
	componentName:	npn

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	Q
pnp	netlistProcedure:	HSIMDevPrim
	instParameters:	area areab areac m
	termOrder:	C B E
	propMapping:	
	componentName:	npn
	namePrefix:	Q
njfet	netlistProcedure:	HSIMDevPrim
	instParameters:	area l w m dtemp
	termOrder:	D G S progn(bn)
	propMapping:	nil vds Vds vgs Vgs vgbs Vgbs
	componentName:	njfet
	namePrefix:	J
pjfet	netlistProcedure:	HSIMDevPrim
	instParameters:	area l w m dtemp
	termOrder:	D G S progn(bn)
	propMapping:	nil vds Vds vgs Vgs vgbs Vgbs
	componentName:	njfet
	namePrefix:	J
nmos	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
pmos	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
nmos4	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
pmos4	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	componentName:	mosfet
	namePrefix:	M
nbsim	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
pbsim	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
nbsim4	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
pbsim4	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
bcs	netlistProcedure:	HSIMCompPrim
	instParameters:	cur
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	bcs
	namePrefix:	G
bvs	netlistProcedure:	HSIMCompPrim
	instParameters:	vol
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	bvs
	namePrefix:	E
vcres	netlistProcedure:	HSIMCompPrimRef

Table 23 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	instParameters:	csType transfactor delta xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	n+ n-
	refTermOrder:	in+ in-
	propMapping:	
	componentName:	vcres
	namePrefix:	G
vccap	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType tc1 tc2 scale delta xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	n+ n-
	refTermOrder:	in+ in-
	propMapping:	
	componentName:	vccap
	namePrefix:	G

HSIMD Netlist Properties

The table below describes the netlist properties associated with HSIMD.

Table 24 HSIMD Netlist Properties

Primitive	Function	Default Parameter
res	netlistProcedure:	
	instParameters:	r l w m scale dtemp tc1 tc2
	otherParameters:	model
	componentName:	resistor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	R
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
presistor	netlistProcedure:	
	instParameters:	r l w m scale dtemp tc1 tc2
	otherParameters:	model
	componentName:	resistor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	R
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
cap	netlistProcedure:	
	instParameters:	c l w m scale dtemp ic tc1 tc2

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	otherParameters:	model
	componentName:	capacitor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	C
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
pcapacitor	netlistProcedure:	
	instParameters:	c l w m scale dtemp ic tc1 tc2
	otherParameters:	model
	componentName:	capacitor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	C
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
ind	netlistProcedure:	
	instParameters:	l m dtemp ic
	otherParameters:	model
	componentName:	inductor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	L
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
mind	netlistProcedure:	hsimDMindPrim
	instParameters:	coupling
	otherParameters:	ind1 ind2
	componentName:	mutual_inductor
	termOrder	
	proMapping	nil coupling k
	namePrefix:	K
	termMapping	
vdc	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type
	otherParameters:	noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acptype srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
idc	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	otherParameters:	noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc idc mag acm phase acp type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS "(FUNCTION minus(root(\"PLUS\")))")
vpulse	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type va10 val1 period delay rise fall width
	otherParameters:	fundname noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
	termOrder	PLUS MINUS
	proMappin	nil dc vdc mag acm phase acp va10 v1 val1 v2 period per delay td rise tr fall tf width pw type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS "(FUNCTION minus(root(\"PLUS\")))")
ipulse	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type va10 val1 period delay rise fall width

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	otherParameters:	
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	(nil dc vdc mag acm phase acp va10 v1 val1 v2 period per delay td rise tr fall tf width pw type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
vpwl	netlistProcedure:	hsimDPwlsrcPrim
	instParameters:	dc type mag phase delay offset scale stretch pwlperiod twidth
	otherParameters:	tvpairs t1 v1 t2 v2 t3 v3 t4 v4 t5 v5 t6 v6 t7 v7 t8 v8 t9 v9 t10 v10 t11 v11 t12 v12 t13 v13 t14 v14 t15 v15 t16 v16 t17 v17 t18 v18 t19 v19 t20 v20 t21 v21 t22 v22 t23 v23 t24 v24 t25 v25 t26 v26 t27 v27 t28 v28 t29 v29 t30 v30 t31 v31 t32 v32 t33 v33 t34 v34 t35 v35 t36 v36 t37 v37 t38 v38 t39 v39 t40 v40 t41 v41 t42 v42 t43 v43 t44 v44 t45 435 t46 v46 t47 v47 t48 v48 t49 v49 t50 v50 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td offset vo type srcType
	namePrefix:	V

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
ipwl	netlistProcedure:	hsimDPwlsrcPrim
	instParameters:	dc type mag phase delay offset scale stretch pwlperiod twidth
	otherParameters:	tipairs t1 i1 t2 i2 t3 i3 t4 i4 t5 i5 t6 i6 t7 i7 t8 i8 t9 i9 t10 i10 t11 i11 t12 i12 t13 i13 t14 i14 t15 i15 t16 i16 t17 i17 t18 i18 t19 i19 t20 i20 t21 i21 t22 i22 t23 i23 t24 i24 t25 i25 t26 i26 t27 i27 t28 i28 t29 i29 t30 i30 t31 i31 t32 i32 t33 i33 t34 i34 t35 i35 t36 i36 t37 i37 t38 i38 t39 i39 t40 i40 t41 i41 t42 i42 t43 i43 t44 i44 t45 435 t46 i46 t47 i47 t48 i48 t49 i49 t50 i50 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc idc mag acm phase acp delay td offset io type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
vsin	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay sinedc ampl sinephase freq damp
	otherParameters:	
	componentName:	vsource
	termOrder	PLUS MINUS

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	proMapping	nil dc vdc mag acm phase acp delay td sinedc vo amp1 damp theta type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
isin	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay sinedc amp1 sinephase freq damp
	otherParameters:	fundname fundname2 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td sinedc vo ampl damp theta type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
vexp	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay val0 val1 td1 tau1 td2 tau2
	otherParameters:	
	componentName:	vsource
	termOrder	PLUS MINUS

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	proMapping	nil dc vdc mag acm phase acp delay td val0 v1 val1 v2 type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
ixep	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay val0 val1 td1 tau1 td2 tau2
	otherParameters:	
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td val0 v1 val1 v2 type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
vcvs	netlistProcedure:	hsimDVCPrim
	instParameters:	gain
	otherParameters:	
	componentName:	vcvs
	termOrder	PLUS MINUS NC\+ NC\-
	proMapping	nil gain egain
	namePrefix:	E

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))” NC\+ “(FUNCTION zero(root(\”PLUS\”)))” NC\- “(FUNCTION zero(root(\”PLUS\”)))”)
vccs	netlistProcedure:	hsimDVCPrim
	instParameters:	gm m
	otherParameters:	
	componentName:	vccs
	termOrder	PLUS MINUS NC\+ NC\-
	proMapping	nil gm ggain
	namePrefix:	G
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))” NC\+ “(FUNCTION zero(root(\”PLUS\”)))” NC\- “(FUNCTION zero(root(\”PLUS\”)))”)
ccvs	netlistProcedure:	hsimDCCPrim
	instParameters:	rm
	otherParameters:	vref
	componentName:	ccvs
	termOrder	PLUS MINUS
	proMapping	nil rm hgain probe vref
	namePrefix:	H
	termMapping	(nil PLUS \:p MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
cccs	netlistProcedure:	hsimDCCPrim

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	instParameters:	gain m
	otherParameters:	vref
	componentName:	cccs
	termOrder	PLUS MINUS
	proMapping	nil gain fgain probe vref
	namePrefix:	F
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
tline	netlistProcedure:	
	instParameters:	z0 td f nl vel len m
	otherParameters:	model
	componentName:	tline
	termOrder	IN\+ IN\- OUT\+ OUT\-
	proMapping	ni1 z0 zo f freq
	namePrefix:	T
	termMapping	(nil IN\+ \:t1 IN\- \:b1 OUT\+ \:t2 OUT\- \:b2)
diode	netlistProcedure:	
	instParameters:	area perim 1 w m scale trise region
	otherParameters:	model
	componentName:	
	termOrder	PLUS MINUS
	proMapping	

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
pdiode	netlistProcedure:	
	instParameters:	area perim 1 w m scale trise region
	otherParameters:	model
	componentName:	
	termOrder	PLUS MINUS
	proMapping	
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
zener	netlistProcedure:	
	instParameters:	area perim 1 w m scale trise region
	otherParameters:	model
	componentName:	
	termOrder	PLUS MINUS
	proMapping	
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
schottky	netlistProcedure:	
	instParameters:	area perim 1 w m scale trise region

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	otherParameters:	model
	componentName:	
	termOrder	PLUS MINUS
	proMapping	
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(\”PLUS\”)))”)
npn	netlistProcedure:	
	instParameters:	area m trise region
	otherParameters:	model
	componentName:	
	termOrder	C B E S
	proMapping	
	namePrefix:	Q
	termMapping	(nil C \:c B \:b E \:e S \:s)
pnp	netlistProcedure:	
	instParameters:	area m trise region
	otherParameters:	model
	componentName:	
	termOrder	C B E S
	proMapping	
	namePrefix:	Q

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termMapping	(nil C \:c B \:b E \:e S \:s)
njfet	netlistProcedure:	
	instParameters:	area m region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	J
	termMapping	(nil D \:d G \:g S \:s B \:b)
pjfet	netlistProcedure:	
	instParameters:	area m region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	J
	termMapping	(nil D \:d G \:g S \:s B \:b)
nmos	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
pmos	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
nmos4	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
pmos4	netlistProcedure:	

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
nbsim	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)
pbsim	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	

Table 24 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)
nbsim4	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)
pbsim4	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)

15

HSIM-Virtuoso Interface Advanced Topics

Provides information on creating an hsim/hsimD view as a stop view for the Cadence® database traversing scheme. It details the hsim/hsimD SimInfo requirements for providing information for the netlister from primitive cells required by Cadence software.

Generating hsim/hsimD View and SimInfo

An hsim/hsimD view can be created as a stop view for Cadence's database traversing scheme. hsim/hsimD view is required to indicate the stop point of database traversing for the hsim/hsimD netlister. Cadence software requires this name to ensure proper database traversing during netlisting. hsim/hsimD SimInfo is required for primitive cells to provide cell information for the hsim/hsimD netlister.

Three functions can be used to create an hsim/hsimD view and hsim/hsimD SimInfo:

```
nsdCreateView(l_libnames @optional(target "hsim")(sources nil))
nsdCreateSimInfo(l_libnames @optional(target "hsim")(sources
nil))
nsdCreateSimInfoAndView(l_libnames @optional(target
"hsim")(sources nil))
```

Note:

> An optional argument in Skill function provides users the ability to either ignore the argument or explicitly give the value of the argument to override its default value.

The following example creates hsim view for the analogLib where the value of the original target is hsim:

```
nsdCreateView("analogLib")
```

The following example creates hsimD view for the libraries lib1 and lib2 that contain the value of target that are overridden to hsimD.

```
nsdCreateView('("lib1" "lib2") "hsimD"))
```

Note:

The value of target can be either hsim or hsimD as described in the following:

- If the value is hsim, it copies from the hspiceS view if sources is not specified.
- If the value is hsimD, it copies from the Spectre® view if sources is not specified.

Examples

```
nsdCreateView("liba")
```

Creates hsim view for liba from the HSPICE view.

```
nsdCreateView('("liba" "libb"))
```

Creates hsim view for liba and libb from HSPICES view.

```
nsdCreateView("liba" "hsimD" ("Spectre" "SpectreS" "hspiceS"))
```

Creates hsimD view for liba from Spectre, SpectreS, and HSPICES, with priority in the same order in which they are listed.

```
nsdCreateSimInfo("liba" "hsimD")
```

Creates hsimD simInfo for liba from Spectre siminfo.

```
nsdCreateSimInfoAndView("mylib")
```

Creates hsim view and simInfo for mylib.

Modifying hsim/hsimD SimInfo

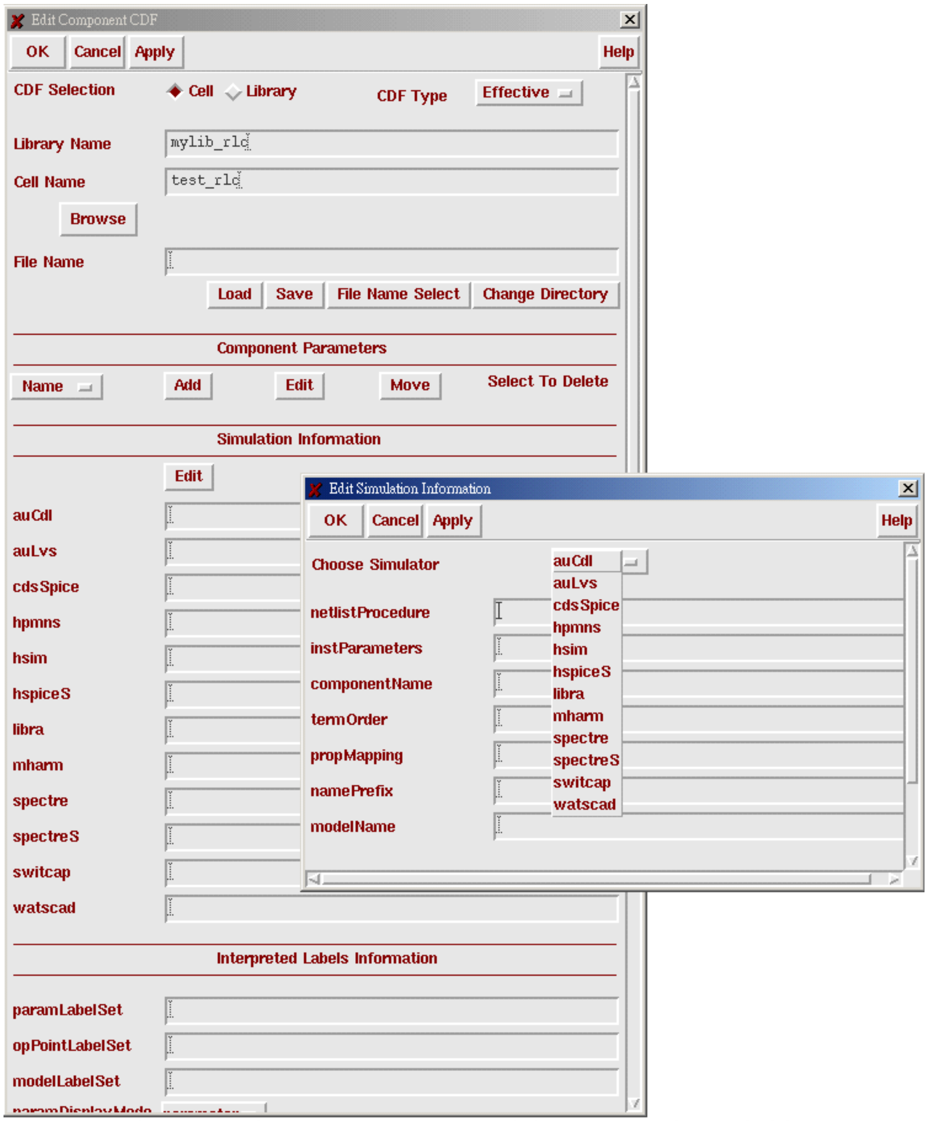
The Interface relies on the Component Description Format hsim/hsimD SimInfo fields for netlisting. The Component Description Format editor is used to modify the fields. To use this editor, launch icms from a shell prompt. In the graphic environment, select the following in order:

Tools->CDF->Edit

Cells and libraries can be browsed and edited using the editor. User's can select from four alternatives for working with an analog component primitive:

- Component Parameters
- Simulation Information
- Interpreted Labels Information
- Other Information

After successfully installing the HSIM-Virtuoso Interface, an [hsim/hsimD] screen button appears in the Choose Simulator field as shown in Figure 103 on page 558.



CDF Selection ◆ Cell ◇ Library CDF Type Effective

Library Name mylib_rlc

Cell Name test_rlc

File Name

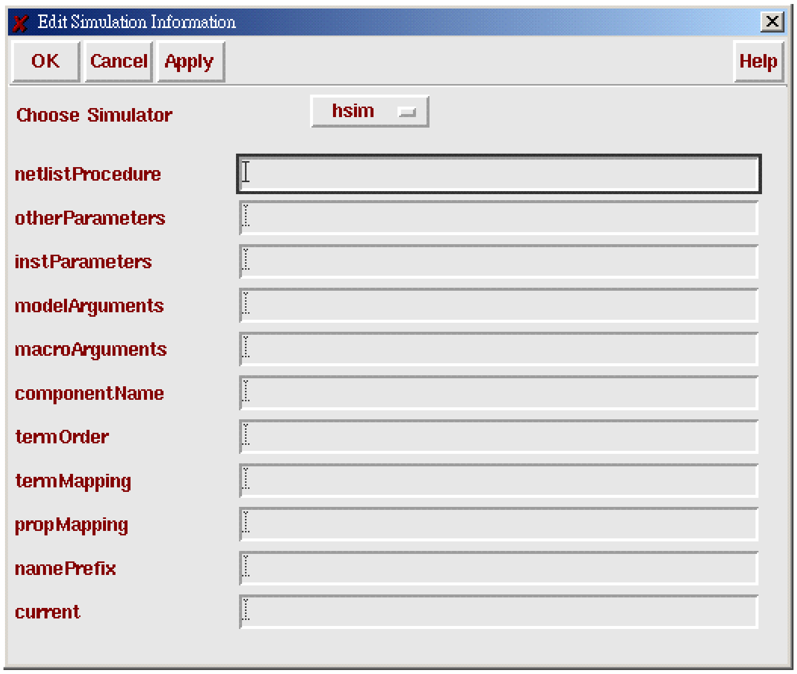
Choose Simulator auCdl

Figure 103 Edit Simulation Information Window

To edit simulation information, complete the following steps:

1. Select the [Edit] screen button under Simulation Information menu bar. The Edit Simulation Information window is displayed as shown in Figure 104 on page 559.

2. Select the [hsim/hsimD] screen button in the Edit Simulation Information Window. The hsim/hsimD simulation information will be displayed in the form as shown in Figure 104 on page 559.



Choose Simulator hsim

netlistProcedure
otherParameters
instParameters
modelArguments
macroArguments
componentName
termOrder
termMapping
propMapping
namePrefix
current

Figure 104 Edit Simulation Information Window

The form fields and their HSIM-specific descriptions are shown in Table 25 on page 560. There are several fields required for the interface, including:

- netlistProcedure
- instParameters
- componentName
- termOrder
- propMapping

- namePrefix

Table 25 Form Field Names and Descriptions

Field Name	Description
netlistProcedure	The HSIM-Virtuoso Interface provides a set of netlist subroutines for hsim/ hsimD. Refer to Table 26 on page 562 for details.
otherParameters	No specific meaning for HSIM.
instParameters	List of parameters which will be included in the netlist file associated with HSIM.
modelArguments	No specific meaning for HSIM.
macroArguments	No specific meaning for HSIM.
componentName	The type of component created.
termOrder	List of terminals that defines the net order for the component. Programmable nodes are supported.
refTermOrder	Reference term order. It is only used by netlist procedures for controlled sources, i.e. HSIMCompPrimRef.
termMapping	No specific meaning for HSIM.
propMapping	List which defines the map of users own Component Description Format parameter name and the name HSIM recognizes. The fields have to be in a format as shown below: nil <hsim parameter name 1> <user's component description format name 1> <hsim parameter name 2> <user's CDF name 2> <hsim parameter name 3> <user's CDF name 3> ...
namePrefix	The prefix for the device in the netlist file.
current	No specific meaning for HSIM.

3. Using the Simulator cyclic button, select HSIM.
4. For netlist to traverse the design database, the following must be set:

- stopViewList
- switchViewList

In icms, select the following

- Setup
- Environment

The following features are described below:

- stopViewList: A list of cellview names that specifies the leaf cells
- switchViewLis: A list of cellview names that binds the netlist procedures in the design hierarchy

Typical setting is as follows:

```
stopViewList=hsim hspiceS
switchViewList=hsim hspiceS schematic
```

Removing hsim/hsimD SimInfo

Commands for removing hsim/hsimD SimInfo fields from the database of primitives are also provided. They are:

```
nsdRemoveSimInfo(libName @ Optional (target “hsim”))
```

The argument target in the procedure is optional where hsim is the default. If another argument replaces hsim, it will be used instead.

The following command removes hsimD SimInfo from the libName library:

```
nsdRemoveSimInfo(“libName” “hsimD”)
```

The following command removes hsim SimInfo for the liba library:

```
nsdRemoveSimInfo(“liba”)
```

Netlist Procedures for Component Primitives

The netlistProcedure field defines which netlist procedure to call for the primitive. The HSIM-Virtuoso Interface supplies the five classes of netlist procedures for component primitives that use fields in the hsim/hsimD SimInfo section of each of the primitives shown in Table 26 on page 562.

Table 26 HSIM Netlist Primitives

Class	Netlist Primitive	Explanation
I.	HSIMCompPrim	For components which a device model is not necessary; such as, resistors, capacitors, and inductors.
II.	HSIMDevPrim	For components which requires a device model, such as bjt, diode, MOSFET etc. Warning messages are printed if no correspondent models found in the design.
III.	HSIMCompPrimRef	For controlled source components, that is, vccs, vcvs, cccs, ccvs, vcres and vccap.

Table 26 HSIM Netlist Primitives (Continued)

Class	Netlist Primitive	Explanation
IV.	HSIMSrcPrim_XXX	This is a set of procedures for independent sources, where XXX is the name of the source. Examples: The netlist procedure for a DC current source (idc) is HSIMSrcComp_idc. The HSIM-Virtuoso Interface supports the following independent sources: HSIMSrcPrim_idc HSIMSrcPrim_ipulse HSIMSrcPrim_isin HSIMSrcPrim_isffm HSIMSrcPrim_iam HSIMSrcPrim_iexp HSIMSrcPrim_ipwl HSIMSrcPrim_ipwlf HSIMSrcPrim_vdc HSIMSrcPrim_vpulse HSIMSrcPrim_vsin HSIMSrcPrim_vsffm HSIMSrcPrim_vam HSIMSrcPrim_vexp HSIMSrcPrim_vpwl HSIMSrcPrim_vpwlf HSIMSrcPrim_vpwlz HSIMSrcPrim (for vsource & isource)
V.	HSIMSubcktCall	For netlist macro primitives. The primitive is defined as a user-supplied netlist. The namePrefix and componentName should be set as X and subcircuit. The name of the correspondent macro is defined in the CDF parameter macro of the primitive.

Note:

> In Table 26 on page 562, refer to the supported component primitives and their default parameters listed in Appendix 14, HSIM-Virtuoso Interface Netlist Properties.

The procedure selected depends on the primitive characteristics to be netlisted as described in the table.

HSIMD Netlist Procedures for Component Primitives

Unlike the socket interface, most primitives do not need to be assigned with a netlist procedure, except primitives shown in Table 27 on page 564.

Table 27 HSIMD Netlist Primitives

Lib	Cell	HSIMD Netlist Procedure
analogLib	vcss, vccs	hsimDVCPrim
	ccvs, cccs	hsimDCCPrim
	dc, pulse, sin, exp	hsimDSrcPrim
	pwl	hsimDPwlSrcPrim

instParameters Field

The instParameters field is a list of parameters that are included in the netlist with the primitive. The parameter name in the list is the Component Description Format name used by the primitive. It is not necessarily the name recognized by HSIM. If it is not the same name, additional information must be added to map the unrecognized name to the one that HSIM recognizes in propMapping. Refer to propMapping Field below.

componentName Field

The netlister refers to the componentName field for the type of component specified.

termOrder Field

The termOrder field is a list of terminals that define the terminal order of the primitive being netlisted. Terminal names are defined in the symbol view of the primitive.

propMapping Field

The propMapping field records the map between the Component Description Format parameter name and the HSIM recognized name. It has the form of:

```
nil <hsim parameter name 1> <user's CDF name1>
<hsim parameter name 2> <user's CDF name2>
<hsim parameter name 3> <user's CDF name3>...
```

The netlister uses this map to translate Cadence parameters to HSIM parameters.

namePrefix Field

The namePrefix field defines the netlist primitive prefix. HSIM uses this to determine which component type is in the netlist file.

Setting up hsim SimInfo fields for a user defined component:
In this example, a capacitor named ucap is built. Assume that ucap has Cadence parameter Cap, TC_1, TC_2 where the following occur:

Cap

Capacitance

TC_1

Temperature coefficient 1

TC_2

Temperature coefficient 2 ucap has two terminals: plus (+) and minus (-).

Refer to the Virtuoso Analog Design Environment user documentation for additional component setup details.

Since the device model for a capacitor is optional in HSIM, HSIMCompPrim is selected as the netlistProcedure. It is assumed that only c and tc_1 must be included in the netlist therefore, instParameters are c and tc_1.

Since ucap is a capacitor so the componentName is cap. Plus is the leading terminal for this component hence, termOrder (Terminal Order) is Plus Minus. Since Cap and TC_1 are not the parameter names that HSIM recognizes, a propMapping field is required. The translation pair for the parameters are Cap<->c and TC_1<->tc1, therefore, propMapping is nil c Cap tc1 TC_1.

Finally, this component is netlisted using namePrefix C as it is a capacitor to HSIM. The hsim SimInfo fields for this component are shown in Table 28 on page 566.

Table 28 Example HSIM SimInfo Fields

netlist name	**hsim/hsimD SimInfo name**
netlistProcedure	HSIMCompPrim
instParameters	c tc_1
componentName	cap
termOrder	Plus Minus
propMapping	nil c Cap tc1 TC_1
namePrefix	C

For hsimD, keep netlistProcedure empty since hsimD uses the default netlist procedures provided by Cadence for primitive elements like capacitor, resistor, etc.

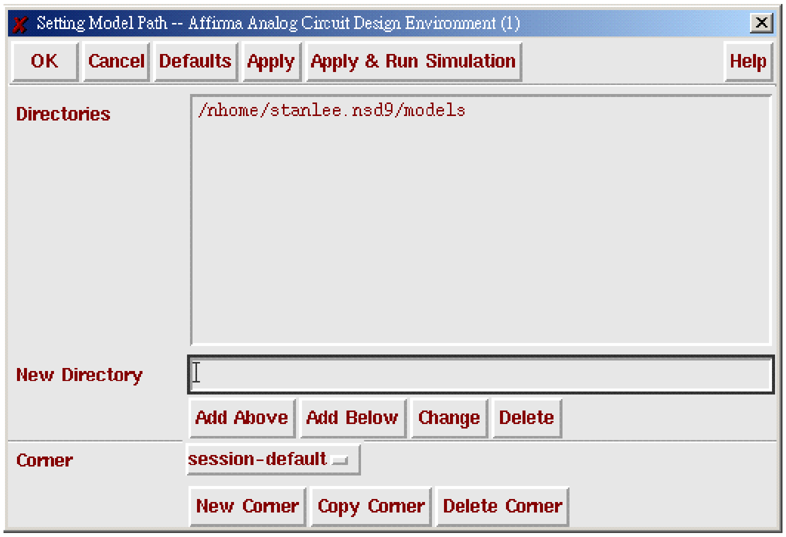
namePrefix Field

A set of component primitives is supported by the interface and installed into the default Cadence analogLib library. The installation script generates the necessary hsim/hsimD SimInfo field for them. A detailed description of the supported primitives is displayed in Appendix 14, HSIM-Virtuoso Interface Netlist Properties.

Models, Macros, and Include Files

Models

Device models in a design are specified in individual files and appended to the final netlist. For hsim, the netlist procedures will search these files accordingly in a user-defined list of model paths. For hsimD, the netlister will search the definition of models in a user-defined list of model libraries. hsim (socket) specifies the model paths by adding the paths where these models are located. To set the model path, select Setup-> Model Path from the screen shown in the HSIM-Virtuoso Interface banner menu.



Directories: /nhome/stanlee.nsd9/models

New Directory:

Corner: session-default

© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 105 Set Model Path Window

Using this approach, each model file can contain only one device model. The file name uses a fixed naming convention that adds a .m to the end of the models name.

A model n-MOSFET is named nmos.m. A model file is a plain text file that can be edited by a common editor such as UNIX vi. To edit a model file, select the following in order:

Setup -> Simulation Files -> Edit Model File

The file is written in native HSIM model syntax with minor modifications as shown in the following syntax example:

```
*
* The comments start with an *
*
.model &1 NMOS $After '$' is an in-line component
+Level=49 $which will be dropped in the final netlist
+Tnom=27.0& $An & at the end of a line is for line continuation
+Nch=2.498+E17 & $$An & at the end of a line is for line continuation
+Tox-9E-09 $A + at the beginning of a line is
+Lint=9.36e-8 $continuation from the previous.
+Vth0= .6322 K1=.756 K2=-3.83e-2 K3=-2.612
+Dvt0= 2.812 Dvt1=0.462 Dvt2=-9.17e-2
+Nlx=3.523E-08 $The rest are omitted from this example
```

In this example, comments and line continuation syntax are remarked. To avoid model name clashes in a design hierarchy during netlisting, the model name is replaced with the special tag &1.

hsimD (Direct)

Instead of specifying the paths to where model files are located, the model libraries to be used for designs should be specified. To specify a model library, complete the following steps:

1. Select Setup > Model Libraries from the Banner Menu.
2. Set Model Library Window.
3. In the Model Library Window, specify the library files and the section to be used for a design.

Macros

Macros are user-supplied sub-circuit definitions for component primitives in socket flow. Macros are not used by the direct flow of hsimD. The netlist procedure for these primitives are HSIMSubcktCall.

The macro definitions to be included in the final netlist. HSIMSubcktCall gathers the name of the macro definition from the Component Description Format parameter macro of the primitive being netlisted. The procedure searches for the definition file in the same paths specified for model files. For the details setting up the path, refer to n in Figure 105 on page 567.

Each model file can contain only one macro or sub-circuit definition. The file name uses a fixed naming convention that adds a .s to the end of the models name such that a model nmoscap is named nmoscap.s. A model file is plain text file that can be edited by common editor such as vi in Solaris.

The file is written in native HSIM netlist syntax with minor modifications as shown in Example 76. Attention is drawn to the replacement macro name with the special tag &1. This is to avoid macro name clashes in a design hierarchy during netlisting.

Example 76 Macro Definition

```
* This is an example of a macro definition
* This subcircuit is called nmoscap.s
* This macro simulates a capacitor using an nmos.
.subckt &1 PLUS MINUS B
.param w='20u' l='20u' as='5p' ad='5p' ps='15u'
+ pd='15u'
*declaration of the parameters is for parameter override.
MC0 MINUL PLUS MINUL B nch w='w' l='l' as='as'
+ ad='ad' ps='ps' pd='pd'
.ends &1
**CAUTION* The model name for devices in macro definition
*will not be manipulated
* Macro/subcircuit definition shares the same line
* continuation and comments style as model files.
```

Include File

The include file is only used by hsim. Apart from giving individual definitions for models and sub-circuits, all predefined definitions can be lumped into a single include file. Users must confirm that the name of the models and sub-circuits are cross referenced in both the include file and design schematics. A graphical file browser is provided to make inputting the include file list as described in Environment Setup on page 383.

Net Name Conversion Macro

The interface attempts to keep the following user-defined names as identical as possible in both the schematic drawing and netlist.

- Net names
- Instance names
- Model names

In some circumstances, the names MUST be changed due to ambiguities. These changes are referred to as name mapping. If the names are specified in both the macro and include files, correct connection may be lost. When this occurs, a name conversion macro is required to ensure correct name mapping. The net to be mapped in the macro and include files must be bracketed as shown in the following example:

```
[# and]
```

The macro then has a specific [#netname] format that forces the interface net netname to be filtered by the name mapping procedure.

Expansion of pPar, iPar

Expansion of pPar and iPar is accomplished as follows:

pPar(x)

> Refers to the value of the parameter from the parent instance of the current instance. To ensure that the resulting netlist is well ordered, the HSIM-Virtuoso Interface utilizes the HSIM parser parameter override mechanism so that after checking the x parameter exists in the parent instance, x replaces pPar(x).

iPar(x)

> Is evaluated as the parameter x value of for the instance being netlisted. the HSIM-Virtuoso Interface replaces the iPar(x) function with the value of x.

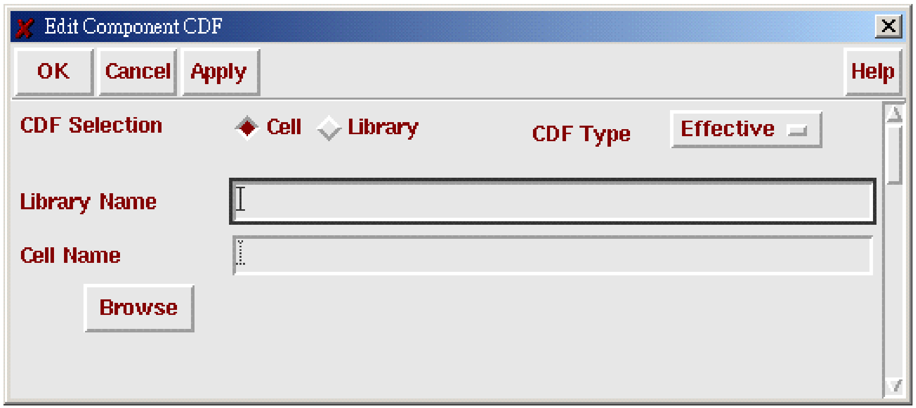
Assigning HSIM Parameters

HSIM parameters can be assigned using any of following methods:

- Sub-circuit
- Instance
- To instance whose sub-circuit has been assigned with same HSIM parameter in CDF.

Assigning HSIM Parameters for Subcircuit

Assigning HSIM parameters for a sub-circuit is accomplished through the Component Description Format as shown in the figure below.



CDF Selection ◆ Cell ◇ Library

CDF Type: Effective

Library Name

Cell Name

Figure 106 Component Description Format Window

To assign HSIM parameters for a subcircuit, follow these steps:

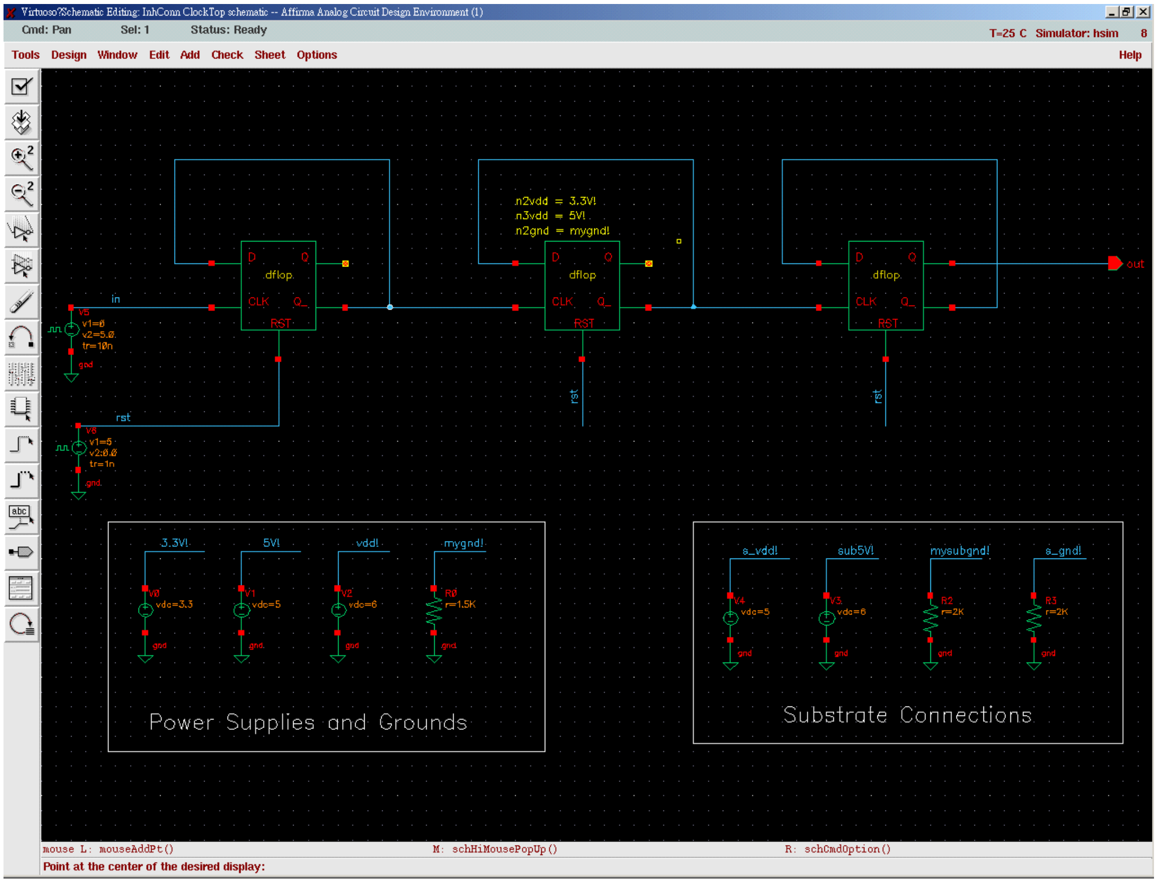
1. In CIW, select Tools -> CDF -> Edit to bring up the Edit Component CDF window.
2. Select one of the following to specify where the HSIM parameter is to be assigned in the same window for the target sub-circuit cell:
   - Library
   - Cell
3. Select the CDF Type to specify the following information:
   - Effective: The change is effective for this instance only.
   - User: The change will be effective for one user only.
   - Base: The change will be effective for all users.
4. In the same window, select Add for component parameters. An Add CDF Parameter window appears.

   For example, if HSIMSPEED=5 is assigned for this specific sub-circuit, the following changes should be made:
   - paramType: is changed to int
   - name: is changed to HSIMSPEED
   - defValue: is changed to 5
5. Click [OK] or [Apply] for both windows. The netlist is displayed as follows:

```
.subckt [Subckt Name] <port> HSIMSPEED=5
.....
.ends
```

Assigning an HSIM Instance Parameter

Assigning an HSIM parameter for an instance must be done using the Virtuoso Schematic Editor. For the example illustrated in Figure 107, the ClockTop design in the InhConn library is used.



mouse L: mouseAddPt()   M: schHiMousePopUp()   R: schCmdOption()

Point at the center of the desired display:

Figure 107 Virtuoso Schematic Editor Window

Figure 107 on page 572, shows the ClockTop design in the Virtuoso Schematic Editor. I3 of the dflop will be assigned to the parameter described later in this section. This is located on the LEFT of the ClockTop cell.

To assign an HSIM instance parameter, follow these steps:

1. Select I3 using the LEFT mouse button.
2. Either click the [Property] button in Equalizer shape cell or select the following in the order shown:

   Edit -> Properties -> Objects...

   An Edit Object Properties window appears.
3. In the Edit Object Properties window, select Add to display the Add Property window.
4. In the Add Property window, enter the HSIM parameter to be added.

   For example, if HSIMABMOS=1 is to be added, enter the following in the Add Property window:

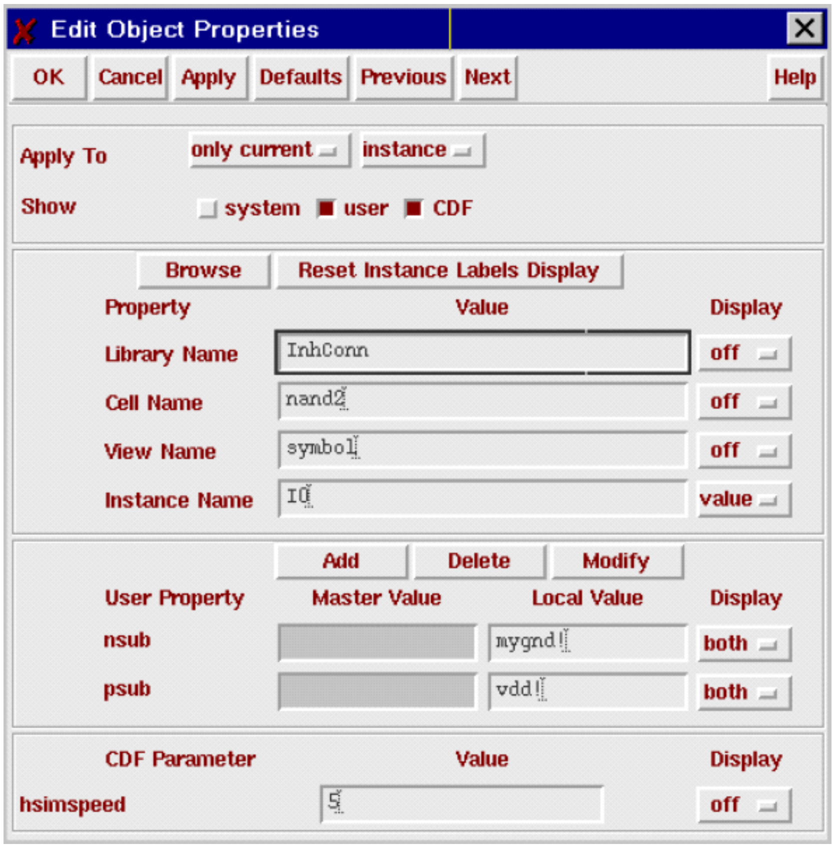
   - Name HSIMABMOS
   - Type int
   - Value 1
   - [OK] or [Apply] For both Add Property and Edit Object Property windows

     The netlist will be displayed as follows:

```
Xname <portlist> <subcktname> HSIMABMOS=1
```

Assigning an HSIM Parameter to an Instance with a Subcircuit (Cell) Assigned the Same HSIM Parameter in Component Description Format

This method is similar to the method described in Assigning an HSIM Instance Parameter on page 572 for an instance shown above however, a parameter does not need to be added. The parameter already exists when the Edit Object Properties window appears. This window is shown in Figure 108.



Apply To: only current, instance

Show: □ system ■ user ■ CDF

Property	Value	Display
Library Name	InhConn	off
Cell Name	nand2	off
View Name	symbol	off
Instance Name	I0	value

User Property	Master Value	Local Value	Display
nsub		mygnd!	both
psub		vdd!	both

CDF Parameter	Value	Display
hsimspeed	5	off

Figure 108 Edit Object Properties Window

The properties for I0 (nand2) in I7 (dflop) of the top cell. Its defValue of HSIMSPEED=5, as set in Step 4 in Assigning HSIM Parameters for Subcircuit on page 570. The value can be changed to any other value through the instance base Edit Object Properties window.

Naming Conventions

The following naming limitations apply to nets and elements:

1. The net name can not be 0 (zero). This is to avoid ambiguity with the nomenclature for a ground. The HSIM-Virtuoso Interface maps zero as follows:

```
Net_Named_0
```

It will not be treated as ground in HSIM. Specify the ground net using the following SPICE command:

```
vxx ground_net gnd 0
```

2. When the first character of a net name is unacceptable to HSIM, it is mapped to another character. For example, `'+'` is mapped to `'n'`. In Table 29 on page 575, first characters of net names are mapped to 'n'.

Table 29 First Characters of Net Names Mapped to 'n'

Character	Definition	Character	Definition
+	Plus sign	$	Dollar sign
,	Comma	.	Period
(	Open parenthesis	)	Close parenthesis
[	Open bracket	]	Close bracket

3. If the first character of an element name is one of those shown in Table 30 on page 575, it is automatically mapped to 'x':

Table 30 First Characters of Element Names Mapped to 'x'

Character	Definition	Character	Definition
+	Plus sign	$	Dollar sign
,	Comma	.	Period
(	Open parenthesis	)	Close parenthesis
[	Open bracket	]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark		

4. If the first character of a net names is an asterisk (*), it is removed.
5. If the first character of an element is an underline (_) symbol, it is automatically mapped to x_.

6. The characters shown in Table 31 on page 576 are not allowed in either a net or element name. If they appear they will be mapped to "".

Table 31 Characters Not Allowed in Net or Element Names Mapped to " "

Character	Definition	Character	Definition
,	Comma	$	Dollar sign
(	Open parenthesis	)	Close parenthesis
[	Open bracket	]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark	*	Asterisk
.	Period	:	Colon

7. If an element/net name has a pipe (\|) or number/pound (#) character, it is mapped to the underscore (_) character.
8. If the first character of a model name is one of those shown in Table 32 on page 576, it is automatically mapped to `'m'`:

Table 32 First Characters in Model Names Mapped to 'm'

Character	Definition	Character	Definition
+	Plus sign	$	Dollar sign
,	Comma	.	Period
(	Open parenthesis	)	Close parenthesis
[	Open bracket	]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark		

9. If the first character of a model name is one of those shown in Table 33 on page 577, it is automatically added as the second character of the model name with ’m’ as the first character: for example, 1model would become m1model.

Table 33 First Characters Added as the First Character of a Model Name

Character	Definition	Character	Definition
_	Underscore	1 ... 9	Numerals 1 through 9
0	Zero		

10. All uppercase letters will be mapped to lowercase letters in the model name.
11. The characters shown in Table 34 on page 577, will be mapped to " " if they appear in the model name.

Table 34 Model Name Characters Mapped to " "

Character	Definition	Character	Definition
+	Plus sign	!	Exclamation mark
,	Comma	$	Dollar sign
(	Open parenthesis	)	Close parenthesis
[	Open bracket	]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket

HSIM-Virtuoso Interface Ocean Script Command Usage

The “HSIMOcean” package can be applied to all integration approaches such as “hsim”, “hsimD”, “aanni” and “aaCosim”. To enable HISMOcean, specify the HSIM-Virtuoso Interface initialization setup in your local `.oceanrc` file as follows:

```
path = getShellEnvVar("NASSDA_ARTISTIF")
nsdaAAIMPkgList='("aanni")
load("$NASSDA_ARTISTIF/install/nsdaAAIMInvoker.il")
```

Public APIs for “HSIMOcean” package

There are seven Application Interface (API) functions available for public use:

Note:

The order of these functions in the ocean script file is important. Please issue these API functions in the same order as they are listed below.

1. `nsdOcnEnvSetup()`

   This function performs the following environment setup tasks:

   a. checks out an AANNI license,

   b. initializes the HSIM environment variables, and

   c. initialize the HSIM parameters.

2. `nsdOcnSetHsimParam(symName value)`

   This function sets user-defined values for HSIM parameters and environment variables. All of the HSIM parameters in the existing Interface GUI and the following HSIM environment variables are supported in this API:

   - netlistSyntax
   - hsimOutputFileNamePrefix
   - isHsimCaseSensitive
   - hsimCommandLinePrefix
   - hsimCommandOptions
   - hsimUsing64Bit
   - hsimCktCheck
   - hsimDeviceV

   If an HSIM parameter is set to a value different than its default value, then HSIMOcean writes it out as part of the Ocean script.

3. `nsdOcnCreateHostNetlist()`

   This function determines if host netlist exists and tries to create it if it is missing.

4. `nsdOcnCreateTopNetlist()`

   This function creates a top level netlist (`hsim.netlist`) for HSIM according to the setup of the HSIM parameters and environment variables.

Note:

The host netlist is included in this top level netlist file

5. `nsdOcnCreateNetlist()`

   This is a group function that consists of `nsdOcnCreateHostNetlist( )` and `nsdOcnCreateTopNetlist( )`. If you want to perform both of these functions at the same time, use `nsdOcnCreateNetlist()`.

6. `nsdOcnRunHsim()`

   This function creates and executes the simulation run file, `runHsim`.

7. `nsdOcnFinishing()`

   This function finishes up after simulation by checking-in the AANNI license.

Ocean Script Example

A complete example of HSIMOcean script is shown in Example 77.

Note:

The ocean script is modified from the default script generated by CDS. Some of CDS function calls have been replaced with nsdOcn- function calls.

Example 77 HSIMOcean Script

```
ocnWaveformTool( 'wavescan )
;<<<----- Choose Host Simulator ----->>>
simulator( 'spectreVerilog )

;<<<----- Setup Design/Analysis/ResultsDir/etc ----->>>
design( "/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog/netlist/analog/netlist")
resultsDir( "/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog" )
modelFile(
    '("/remote/us01home4/jeffhu/Jason/aacosim_demo/models/
spectre/pll.scs" "")
)

analysis('tran ?stop "5u"  )
desVar(  "c1" 100p)
desVar(  "c2" 85f)
desVar(  "r1" 47K)
desVar(  "r2" 4.7K)
desVar(  "wgain" 40u)
option('temp  "27.0" )

;<<<----- Save results ----->>>
save( 'v "/load" )
temp( 27.0 )

;<<<----- Setup AANNI environment ----->>>
nsdOcnEnvSetup()

;<<<----- Setting up Hsim parameters/Env. variables ----->>>
nsdOcnSetHsimParam('hsimresultdir "/remote/us01home4/jeffhu/
simulation/top/spectreVerilog/configSpectreVerilog/psf/")
nsdOcnSetHsimParam('hostsimulator "spectreVerilog")
nsdOcnSetHsimParam('plotitemtable '(("/load" net)))
nsdOcnSetHsimParam('digitalsimulator "NC-Sim")
nsdOcnSetHsimParam('ishsimcasesensitive "no")
nsdOcnSetHsimParam('hsimoutput "psfbin")

;<<<----- Create Host/Top Netlist ----->>>
nsdOcnCreateNetlist()

;<<<----- Start simulation ----->>>
nsdOcnRunHsim()

;<<<----- Open results ----->>>
openResults("/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog/psf")
```

```
;<<<----- Select results ----->>>
selectResult( 'tran )

;<<<----- Plot results ----->>>
plot(nsdGetData("/load"))

;<<<----- Finishing up ----->>>
nsdOcnFinishing()
```

Socket (HSIM) and Direct (HSIMD) Integration

This section describes the installation and usage of HSIM socket and HSIMD direct integration, which are the traditional ways to integrate HSIM with Cadence's Virtuoso Analog Design Environment. The interface is developed on Cadence Design Framework II 4.4.5. and has been tested on the following versions:

- 4.4.5
- 4.4.6
- 5.0.0
- 5.1.41

Note:

> Only the major elements of HSIM and HSIMD installation and usage are described in this section. Refer to Appendix 15, HSIM-Virtuoso Interface Advanced Topics for detailed information.

Installing Socket (HSIM) and Direct (HSIMD) Interfaces

The installation process requires that the HSIM/HSIMD Simulation Information (SimInfo) fields be added. Site administrators are advised to make backups of the Cadence Software hierarchy before proceeding. Customers must have the appropriate write permissions to modify the Cadence environment.

As with the Native Netlist Integration installation, you must first obtain the AAIM package and complete the following steps:

1. Ensure the HSIM executable and Cadence software are in the search path.
2. Set HSIM_HOME environment to the root of HSIM release tree.
3. Set the HSIM_ARTISTIF environment to the root of the AAIM release tree.

4. Add hsim and/or hsimd into the list nsdaAAIMPkgList in .cdsinit file, as in this example:

   ```
   nsdaAAIMPkgList='("hsim" "hsimd" "aanni" …)
   ```

5. Create the menus directory. It can be either of the following:

   - `<user_home_directory>/menus`
   - `<user_working_directory>/menus`

6. Copy menus from the following to the directory created in Step 4:

   For HSIM Socket:

   ```
   cp <AAIM-Install-Tree-Root>/menus/hsim.menu
   ```

   For hsimD Direct:

   ```
   cp <AAIM-Install-Tree-Root>/menus/hsimD.menu
   ```

7. Create hsim/hsimD view and Component Description Format hsim/hsimD SimInfo for the primitive libraries.

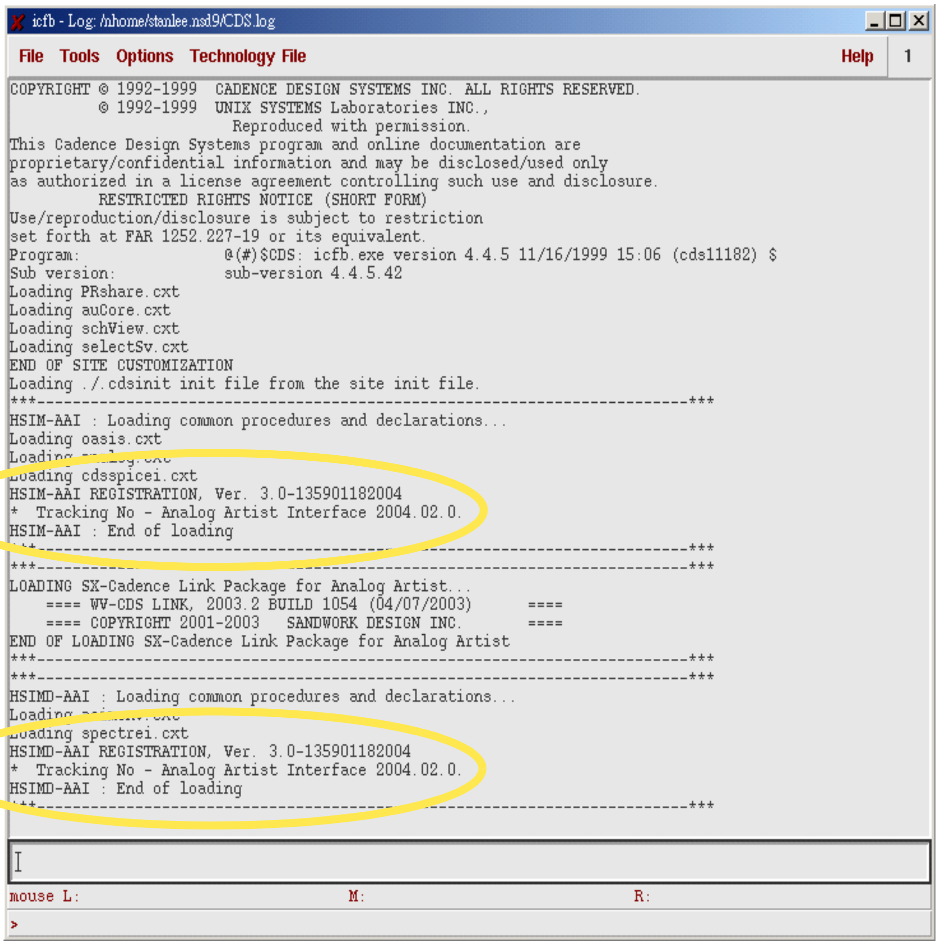
   hsim/hsimD SimInfo is required for each primitive in order to perform design netlisting in the HSIM-Virtuoso Interface. Refer to Modifying hsim/hsimD SimInfo in Appendix 15, HSIM-Virtuoso Interface Advanced Topics, for instructions on generating the information required for individual primitives.

   Note:

   It is recommended that the menu files and .cdsenv for hsim/hsimD can be placed in your CDS tree. Menu files should be placed under `<CadenceDir>/etc/tools/menus` and `.cdsenv` should be placed under either `<CadenceDir>/etc/tools/hsim` or `<CadenceDir>/etc/tools/hsimD`. Both files are included in the AAIM installation package.

8. Confirm that installation is successful.

   Successful installation can be confirmed through Cadence log. A log window is displayed upon successfully registering the Interface as shown in Figure 109.



```
COPYRIGHT © 1992-1999  CADENCE DESIGN SYSTEMS INC. ALL RIGHTS RESERVED.
          © 1992-1999  UNIX SYSTEMS Laboratories INC.,
                         Reproduced with permission.
This Cadence Design Systems program and online documentation are
proprietary/confidential information and may be disclosed/used only
as authorized in a license agreement controlling such use and disclosure.
          RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Program:                @(#)$CDS: icfb.exe version 4.4.5 11/16/1999 15:06 (cds11182) $
Sub version:            sub-version 4.4.5.42
Loading PRshare.cxt
Loading auCore.cxt
Loading schView.cxt
Loading selectSv.cxt
END OF SITE CUSTOMIZATION
Loading ./.cdsinit init file from the site init file.
***-------------------------------------------------------------------------***
HSIM-AAI : Loading common procedures and declarations...
Loading oasis.cxt
Loading analog.cxt
Loading cdsspicei.cxt
HSIM-AAI REGISTRATION, Ver. 3.0-135901182004
*  Tracking No - Analog Artist Interface 2004.02.0.
HSIM-AAI : End of loading
***-------------------------------------------------------------------------***
***-------------------------------------------------------------------------***
LOADING SX-Cadence Link Package for Analog Artist...
    ==== WV-CDS LINK, 2003.2 BUILD 1054 (04/07/2003)      ====
    ==== COPYRIGHT 2001-2003   SANDWORK DESIGN INC.       ====
END OF LOADING SX-Cadence Link Package for Analog Artist
***-------------------------------------------------------------------------***
***-------------------------------------------------------------------------***
HSIMD-AAI : Loading common procedures and declarations...
Loading asimenv.cxt
Loading spectrei.cxt
HSIMD-AAI REGISTRATION, Ver. 3.0-135901182004
*  Tracking No - Analog Artist Interface 2004.02.0.
HSIMD-AAI : End of loading
***-------------------------------------------------------------------------***
```

Figure 109 Registration Confirmation Window

Porting Existing Design

The outline for exporting existing design hierarchy to the HSIM-Virtuoso Interface is described below. Refer to Appendix 15, HSIM-Virtuoso Interface Advanced Topics, for detailed procedures.

To export an existing design, use the following steps:

1. Collect all primitives for the design including, but not limited to, the following components:
    - Resistors

- MOSFETs
- BJTs
- Voltage sources (controlled)
- Current sources (controlled)

2. Verify that hsim/hsimD view exists for primitives listed above. If not, create the view by using the nsdCreateView procedure. Refer to Generating hsim/hsimD View and SimInfo in Appendix 15, HSIM-Virtuoso Interface Advanced Topics for information using nsdCreateView.

   Note:

   Simulator views from other vendors can also be specified using the AAI switch/stop view feature.

3. Edit the hsim/hsimD SimInfo fields for these primitives. Refer to Appendix 15, HSIM-Virtuoso Interface Advanced Topics for information on editing hsim/hsimD SimInfo.
4. Verify the netlist.

Netlist and Simulation

For using HSIM socket and direct interface to run simulation, you must generate a netlist from the schematic design. Data preparation procedures need to be conducted for each primitive device to have required simInfo, such as netlistProcedure, ready for the netlisting process.

The Interface adopts the default Cadence schema as closely as possible. Refer to Cadence's Virtuoso Analog Design Environment user documentation for information on the schema.

Starting the GUI and Selecting HSIM

To start the GUI, follow these steps:

1. Select the following in order as shown in Figure 110:

   Tools -> Analog Environment -> Simulation

2. Select one of the following:
   - hsim: as the simulator for socket integration.
   - hsimD: as the simulator for direct integration.

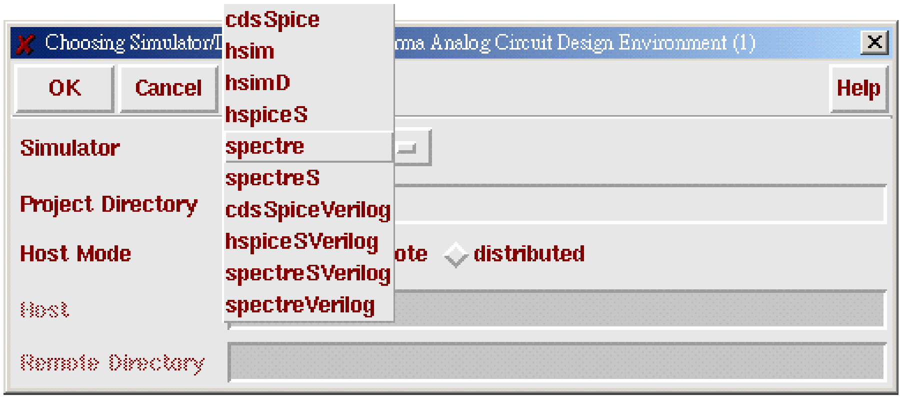


Figure 110 Choosing Simulation Directory Host Window

Specifying a Host Machine

The HSIM-Virtuoso Interface permits selection of a local or remote host to run HSIM. Special attention should be used if the remote mode is selected. The following issues must be considered when choosing to run under a remote host:

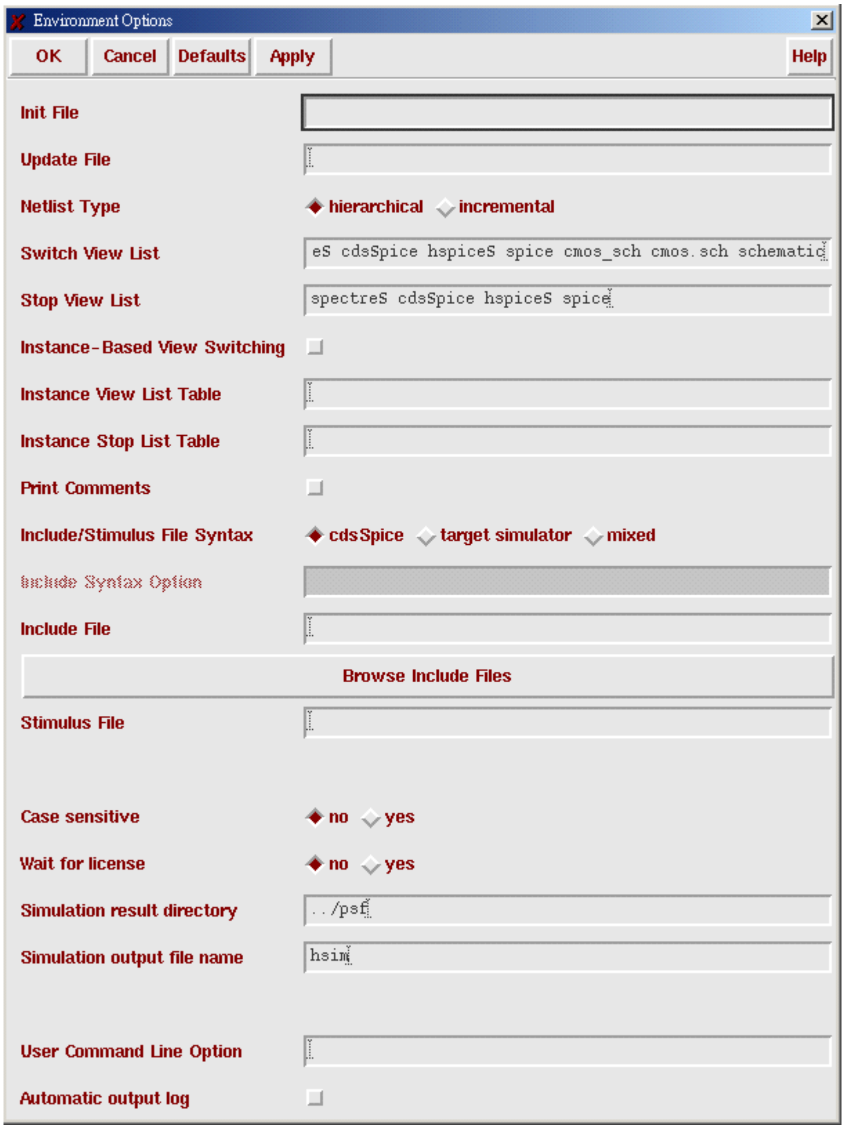
- The remote machine must be setup to grant permissions from the local machine without authentication. For example: .rhosts
- The remote machine must see an identical file hierarchy under the directory path entered in it's Remote Directory as the local machine sees under it's Project Directory.

Setup Environment

To display the Environment Options window, select the following in order:

Setup -> Environment

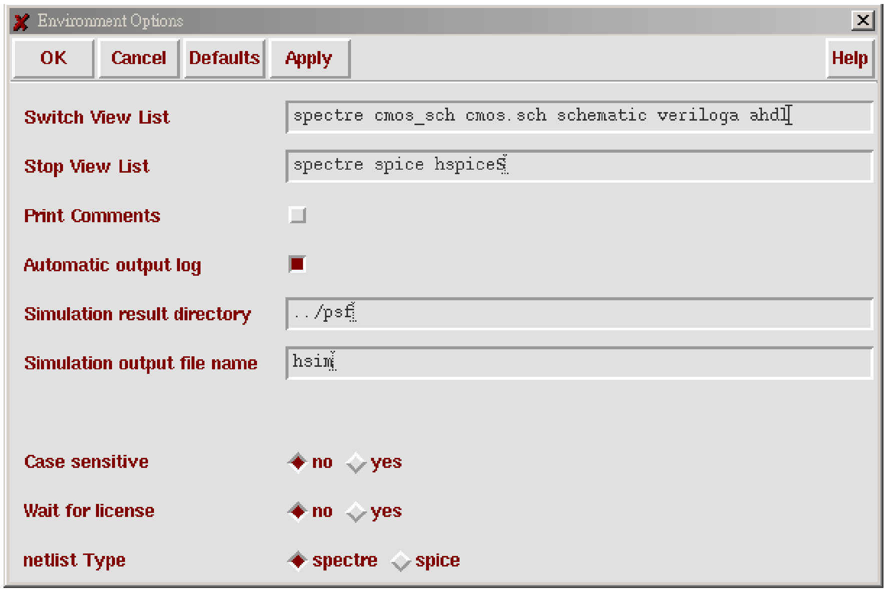
The GUIs shown in Figure 111 and Figure 112 is used to set up the Interface environment for HSIM and HSIMD respectively.



Init File	
Update File	
Netlist Type	● hierarchical ○ incremental
Switch View List	eS cdsSpice hspiceS spice cmos_sch cmos.sch schematic
Stop View List	spectreS cdsSpice hspiceS spice
Instance-Based View Switching	□
Instance View List Table	
Instance Stop List Table	
Print Comments	□
Include/Stimulus File Syntax	● cdsSpice ○ target simulator ○ mixed
Include Syntax Option	
Include File	
Stimulus File	
Case sensitive	● no ○ yes
Wait for license	● no ○ yes
Simulation result directory	../psf
Simulation output file name	hsim
User Command Line Option	
Automatic output log	□

© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 111 Environment Options Window for HSIM Socket



Switch View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop View List: spectre spice hspiceS

Print Comments □

Automatic output log ☑

Simulation result directory: ../psf

Simulation output file name: hsim

Case sensitive ● no ○ yes

Wait for license ● no ○ yes

netlist Type ● spectre ○ spice

Figure 112 Environment Options Window for HSIM Direct

Netlister Settings

For the netlister to traverse the design database correctly, it is important that the following options be correctly set:

- Stop View List: A list of cellview names that specify the leaf cells.
- Switch View List: A list of cellview names that bind the netlist procedures in the design hierarchy.

Typical HSIM socket settings include the following:

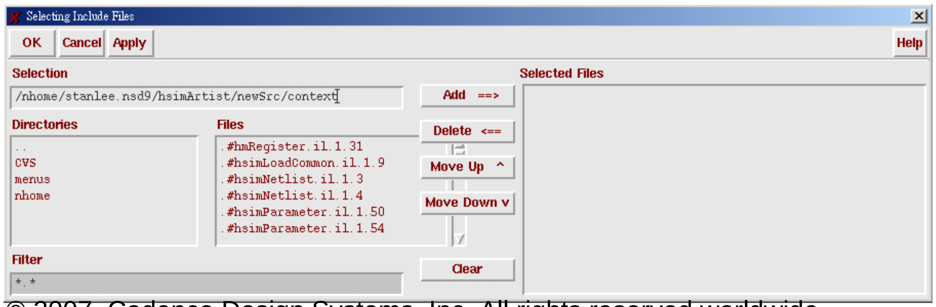
```
Stop View List: hsim hspiceS
Switch View List: hsim hspiceS schematic
```

In Figure 111, the Include/Stimulus File Syntax field setting specifies the syntax of the include stimulus, macro, and model files. The Interface’s netlister performs a syntax check if target simulator is chosen. Special attention should be paid if cdsSpice is selected because the syntax check is handled by cdsSpice and is should conform with the cdsSpice rules.

Files listed in both the Include and Stimulus files are treated equally. They are inserted line-by-line into the final netlist and the net name macro is expanded.

Refer to Appendix 15, HSIM-Virtuoso Interface Advanced Topics, Net Name Conversion Macro on page 569.

Select the [Browse Include Files] button and a graphical file browser is displayed as shown in Figure 113. The files to be included can be browsed and selected from this window.



Selection: /nhome/stanlee.nsd9/hsimArtist/newSrc/context

Directories:

```
..
CVS
menus
nhome
```

Files:

```
.#hmRegister.il.1.31
.#hsimLoadCommon.il.1.9
.#hsimNetlist.il.1.3
.#hsimNetlist.il.1.4
.#hsimParameter.il.1.50
.#hsimParameter.il.1.54
```

Filter: *.*

Figure 113 Selecting Include Files Window

Note:

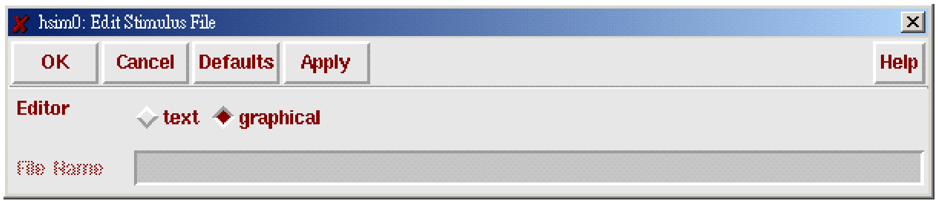
Different result directories and output file names for multiple simulations can be set up. They are specified in the following parts of the Environment Options form:

- Simulation Results directory
- Simulation output file name fields

The Results Browser can be used to inspect the results of different simulations. The associate netlist is also backed up in the Simulation Results directory.

Graphically Editing Stimulus Files

Stimulus files can be graphically edited using the GUI shown in Figure 114. Signals for Stimulus are automatically extracted when the radio button in the Edit Stimulus File window is activated.



Editor ◇ text ◆ graphical

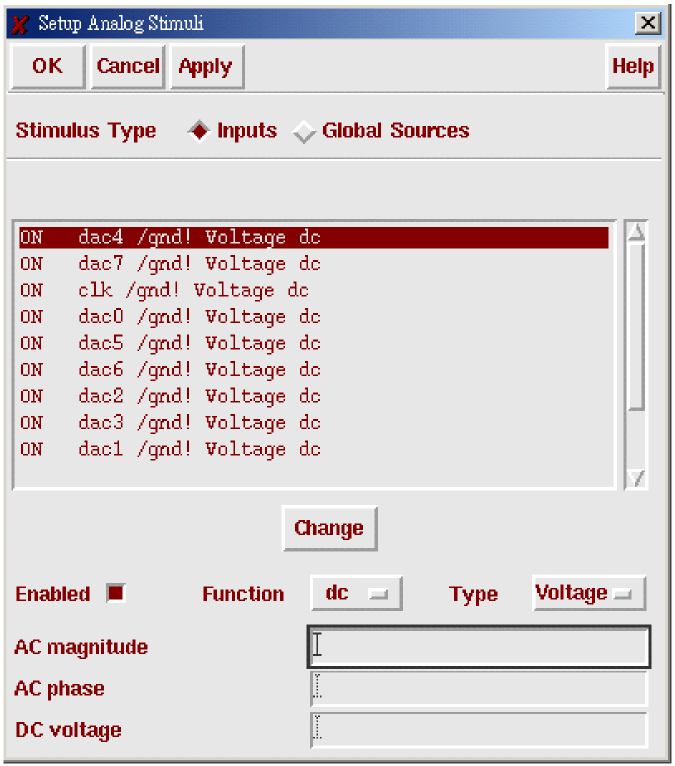
File Name

Figure 114 Edit Stimulus File Window

To display the Setup Analog Stimuli window, select the following in order:

Setup -> Stimulus -> Edit Analog

The available signals are listed in the Graphical Stimulus editor shown in the figure below.



Stimulus Type ◆ Inputs ◇ Global Sources

```
ON   dac4 /gnd! Voltage dc
ON   dac7 /gnd! Voltage dc
ON   clk /gnd! Voltage dc
ON   dac0 /gnd! Voltage dc
ON   dac5 /gnd! Voltage dc
ON   dac6 /gnd! Voltage dc
ON   dac2 /gnd! Voltage dc
ON   dac3 /gnd! Voltage dc
ON   dac1 /gnd! Voltage dc
```

Enabled ■ Function dc Type Voltage

AC magnitude

AC phase

DC voltage

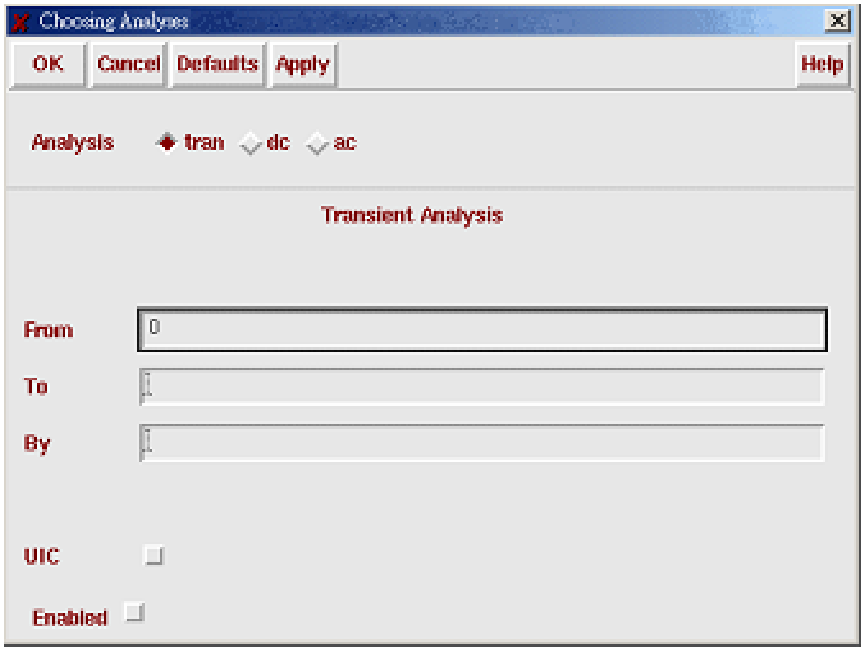
Figure 115 Setup Analog Stimuli Window

Stimulus functions can be changed using the cyclic button of Function. The following Stimulus functions are not supported:

- pwlf
- sffm

Analysis

hsim/hsimD supports AC, DC, and Transient analysis. Selecting the desired analysis process is accomplished by selecting the analysis type in the Virtuoso Analog Design Environment window shown in Figure 116.



Analysis ● tran ◇ dc ◇ ac

From 0

To

By

UIC □

Enabled □

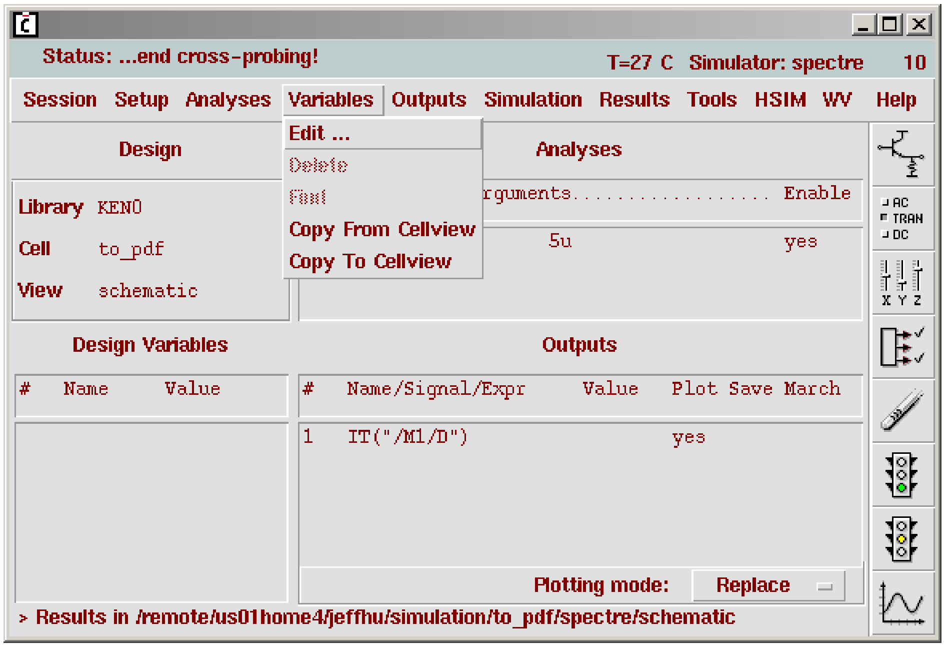
Figure 116 Choosing Analysis - Virtuoso Analog Design Environment Window

To select transient analysis, perform the following steps:

1. Navigate to the appropriate page by choosing Analysis > Choose > .
2. Click on the tran radio button.
3. Specify the simulation end time in the To field.
4. Specify the simulation time step in the By field.

Design Variables

Design variables are defined in the Component Description Format parameters through the design hierarchy. Design variables can be specified for each simulator run. Design variables are obtained using the GUI shown in Figure 117.



Status: ...end cross-probing! T=27 C Simulator: spectre 10

Library KENO
Cell to_pdf
View schematic

```
...rguments.................  Enable
   5u                         yes
```

```
#   Name      Value
```

```
#   Name/Signal/Expr      Value   Plot Save March
1   IT("/M1/D")                   yes
```

Plotting mode: Replace

> Results in /remote/us01home4/jeffhu/simulation/to_pdf/spectre/schematic

Figure 117 Virtuoso Analog Design Environment(1) Window

Perform the following steps:

1. Select Variables.
2. Copy from Cellview: Existing design variables are displayed in the Design Variables list box.
3. Select any variable: A new pop-up window will be displayed as shown in Figure 118.

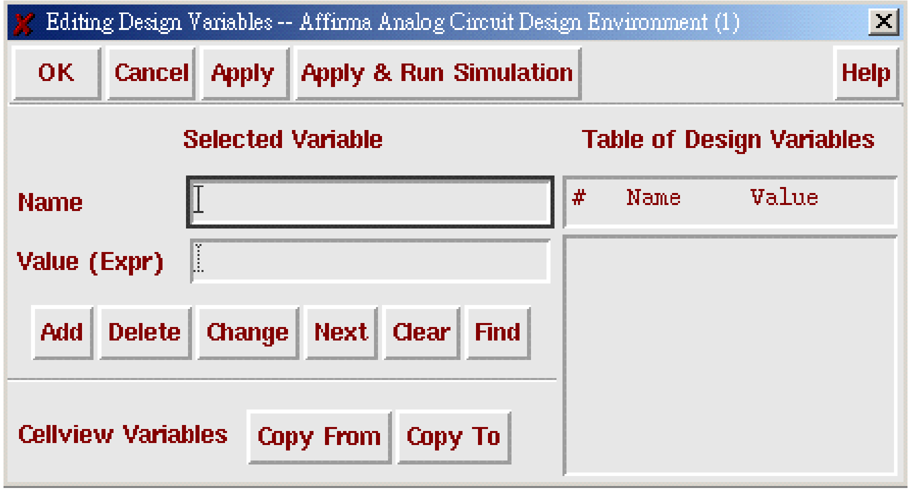
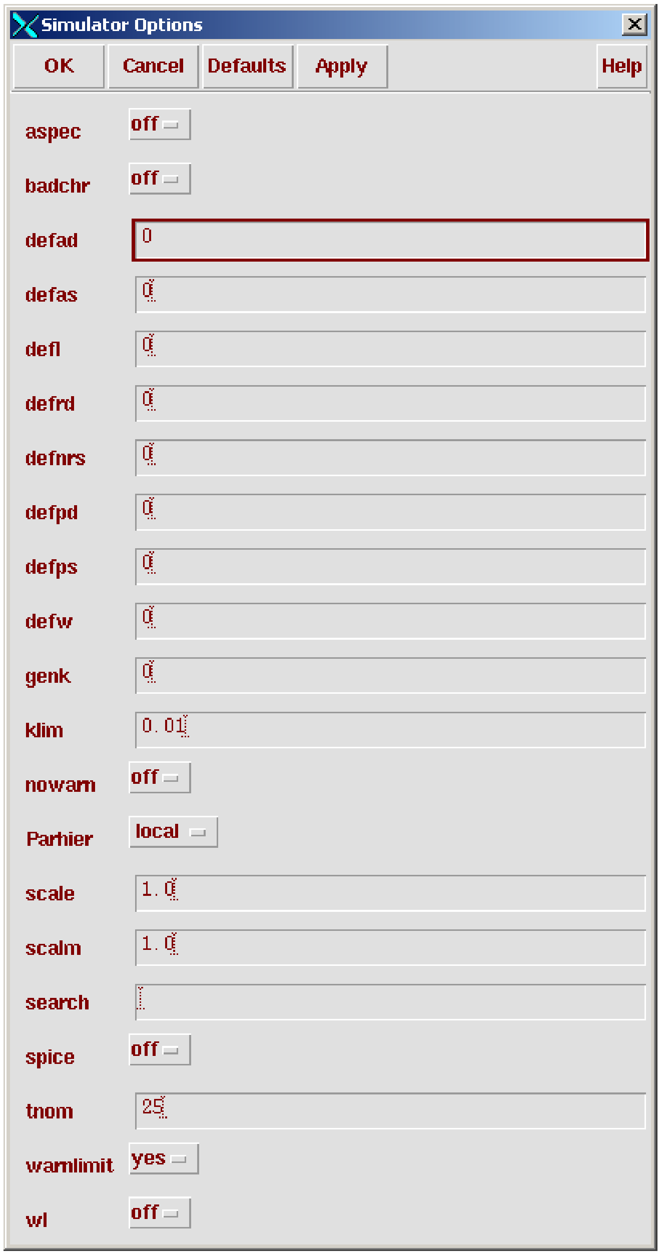


Figure 118 Editing Design Variables Window

4. Set the desired design variable values

Simulator Options

The simulator options are mostly HSPICE compatible. Setup the Interface options using the GUI shown in Figure 119.



aspec	off
badchr	off
defad	0
defas	0
defl	0
defrd	0
defnrs	0
defpd	0
defps	0
defw	0
genk	0
klim	0.01
nowarn	off
Parhier	local
scale	1.0
scalm	1.0
search	
spice	off
tnom	25
warnlimit	yes
wl	off

Figure 119 Simulator Options Window

To set up options, perform the following steps:

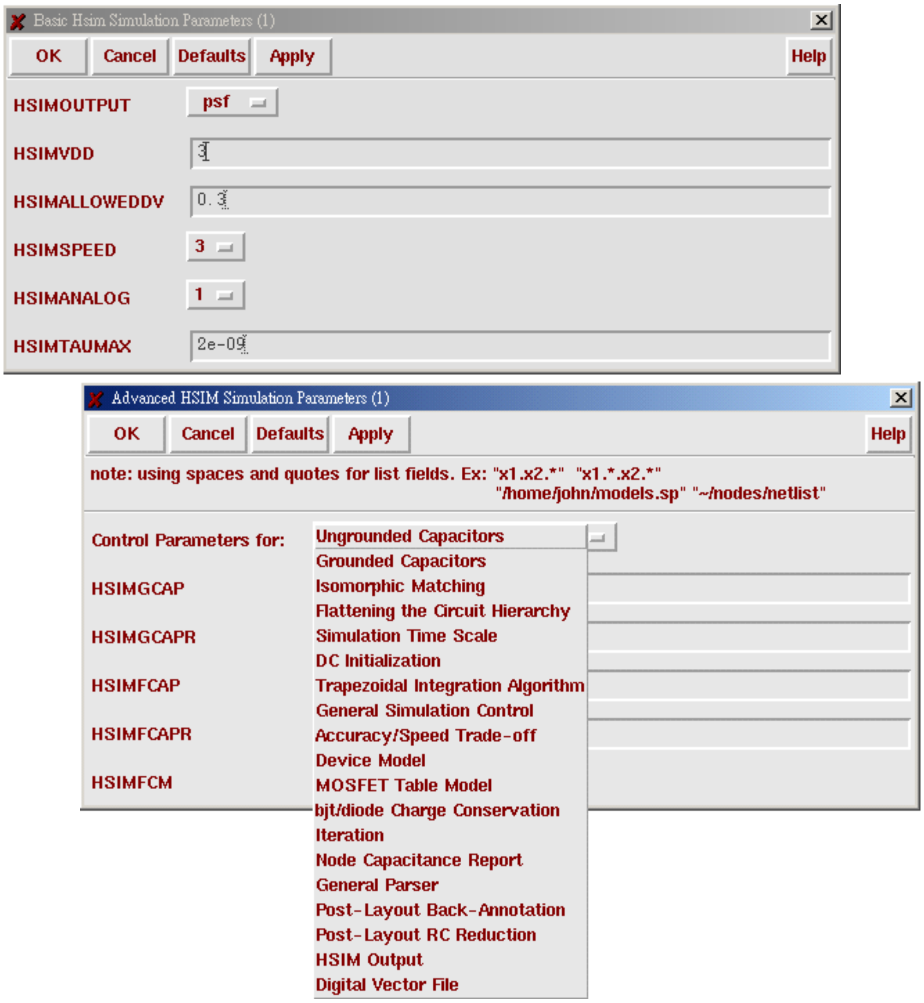
1. Navigate to: Simulation -> Options -> Analog...; a popup form is displayed.

2. Enter the desired values. The filled (filled or (filled-in or entered)) options are netlisted as .option statements in the final netlist.
3. Save or load option settings using the Save State Simulator Options/Load State option in the Session menu.

HSIM Parameters

The HSIM-Virtuoso Interface provides a large set of HSIM-specific parameters to fine tune HSIM performance. For efficiency and user convenience, the HSIM parameters are separated into two groups:

- Basic HSIM Parameters contain only the most frequently used commands.
- Advanced HSIM Parameters include the rest of the commands and are categorized by functionality.



HSIMOUTPUT	psf
HSIMVDD	3
HSIMALLOWEDDV	0.3
HSIMSPEED	3
HSIMANALOG	1
HSIMTAUMAX	2e-09

note: using spaces and quotes for list fields. Ex: "x1.x2.*" "x1.*.x2.*"
"/home/john/models.sp" "~/nodes/netlist"

Control Parameters for: Ungrounded Capacitors

HSIMGCAP

HSIMGCAPR

HSIMFCAP

HSIMFCAPR

HSIMFCM

Figure 120 Basic and Advanced HSIM Parameters Forms

To set up set parameters, select the following in order:

1. Navigate to: Simulation -> HSIM Parameters -> Basic; for basic HSIM parameters or, Simulation -> HSIM Parameters -> Advanced; for advanced HSIM parameters.
2. Enter the desired parameters The parameters are netlisted as .param statements in the final netlist.

The GUI in Figure 120 only sets HSIM parameters for the top cell. It is possible to specify parameters in the sub-hierarchy using the netlisting Component Description Format parameters associated with sub-circuits. Refer to Appendix 15, HSIM-Virtuoso Interface Advanced Topics for details.

3. Save or load HSIM specific parameter settings using the Save State Simulator Parameters/Load State option in the Session menu.

Selecting Data to Save or Plot

Refer to the Cadence Virtuoso Analog Design Environment user documentation for details of this operation.

To select data to save or plot, perform the following steps:

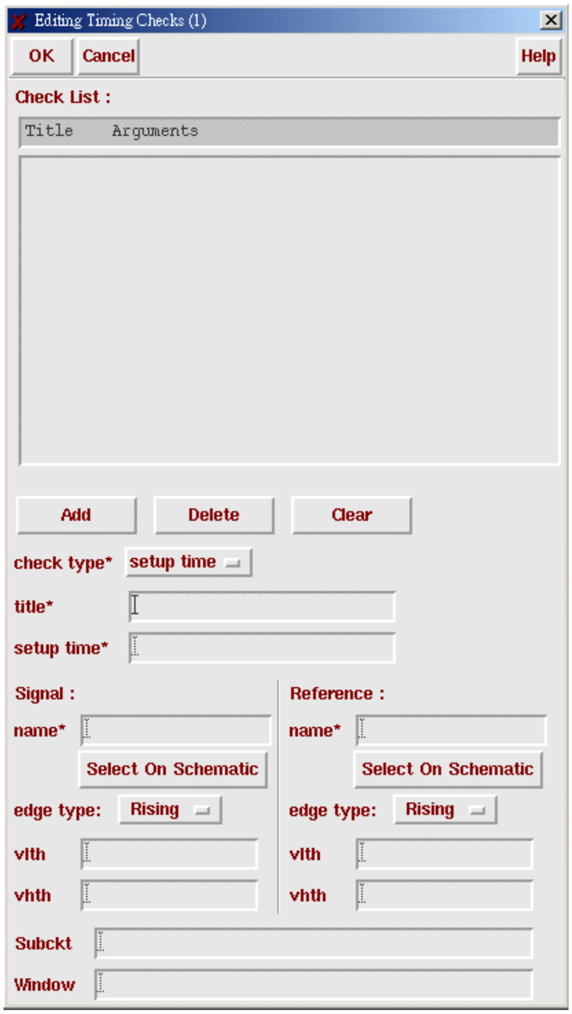
1. Navigate to either of the following to display the Virtuoso Schematic Editor window:
   - Outputs -> To Be Saved -> On schematic
   - Outputs -> To Be Plotted -> On schematic

     The Virtuoso Schematic Editor window is displayed.
2. Use the LEFT mouse button to select the desired nets or nodes The signals selected to be plotted will be displayed automatically when the simulation is completed.
3. View saved signals using the Tools menu in the Results Browser.

Timing and Power Checks

To set timing and power checks, choose Select Outputs -> Timing Checks or Power Checks. Depending on which option is selected, one of the GUI's described in the following figures is displayed:

- Figure 121
- Figure 122

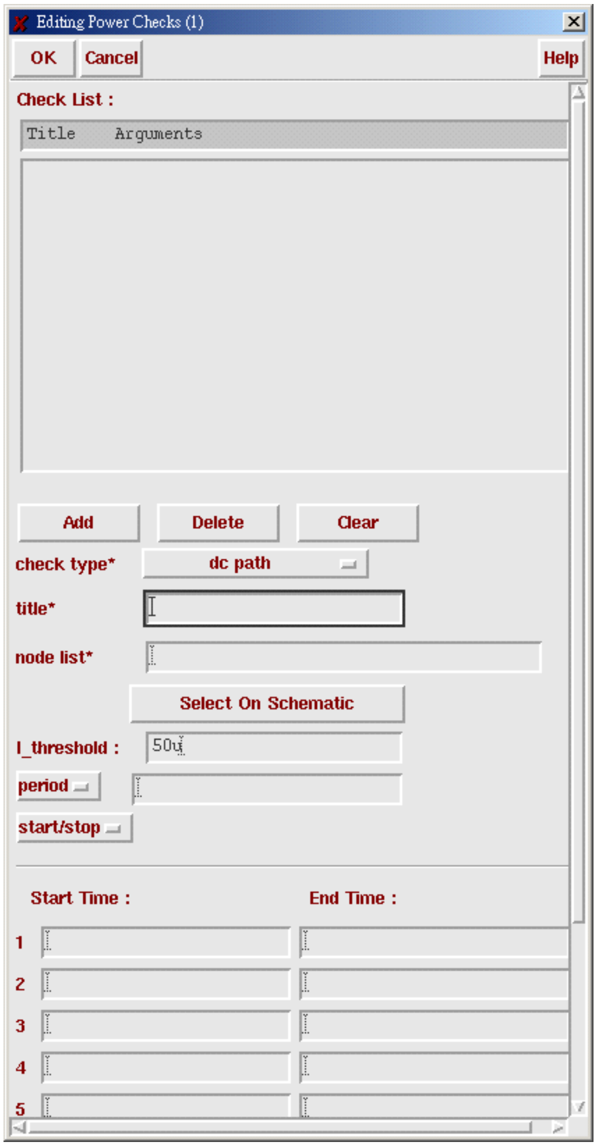


check type* setup time

Signal : edge type: Rising

Reference : edge type: Rising

Figure 121 Editing Timing Checks Window



check type* dc path

I_threshold : 50u

Figure 122 Editing Power Checks Window

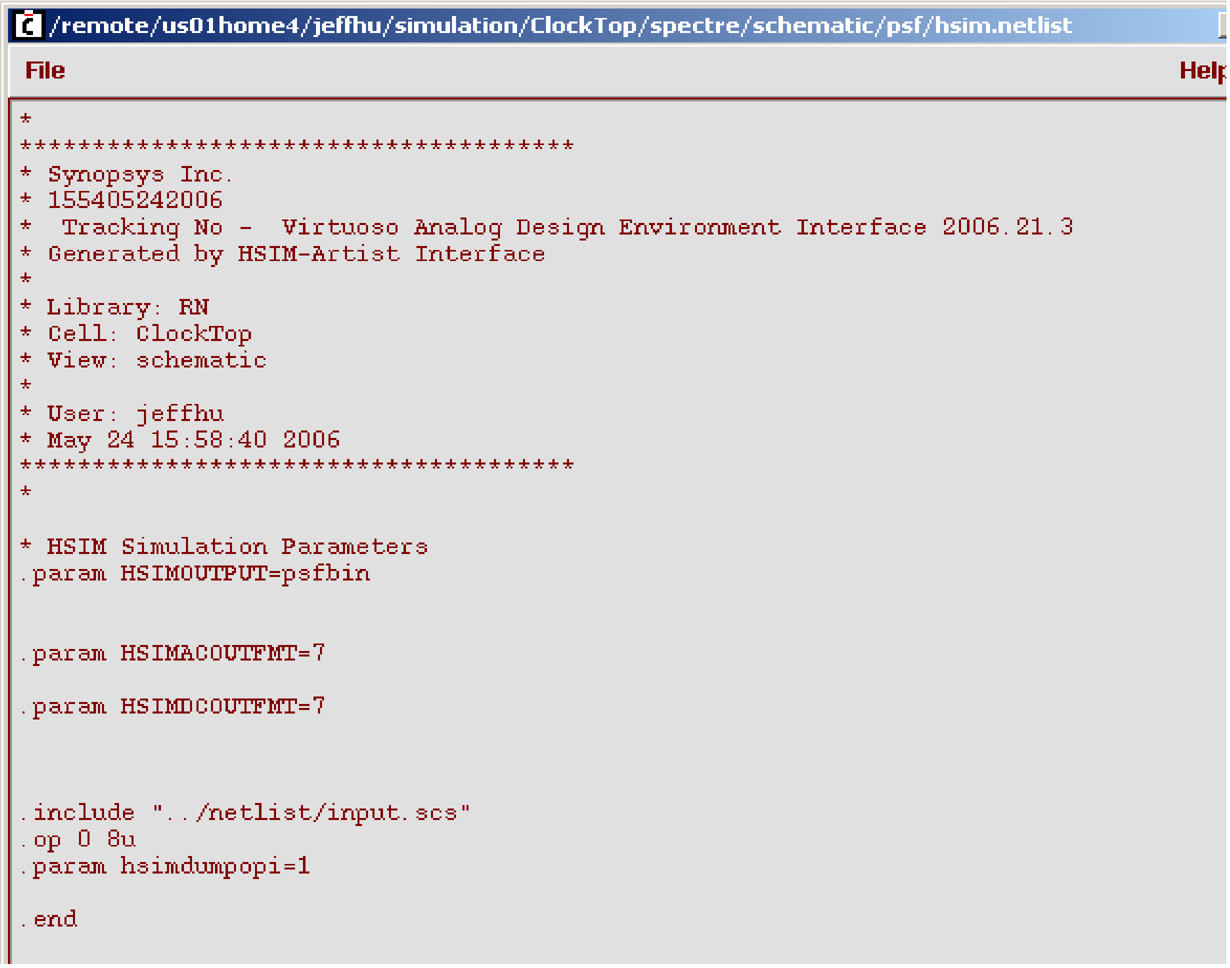
Power and timing check settings can be saved for future use or loaded from the previous session using load/save state facilities. Refer to Load and Save Sessions on page 602.

Generating Netlists

To generate a netlist, follow these steps:

1. Select Simulation -> Netlist.
2. Select one of the following:
   - Re-Netlist: Performs the entire netlisting procedure.
   - Create Netlist: Updates the netlist result incrementally based on environment or parameter changes.
   - Display Netlist: Displays the netlist.

The netlist will be displayed in the window shown in Figure 123.



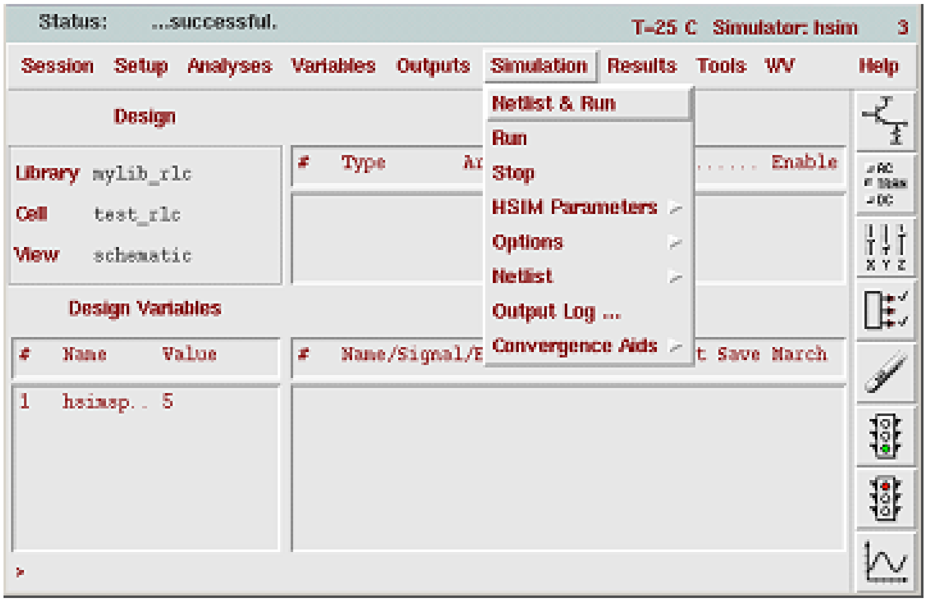
/remote/us01home4/jeffhu/simulation/ClockTop/spectre/schematic/psf/hsim.netlist
File
Help

```
*
***************************************
* Synopsys Inc.
* 155405242006
*  Tracking No -  Virtuoso Analog Design Environment Interface 2006.21.3
* Generated by HSIM-Artist Interface
*
* Library: RN
* Cell: ClockTop
* View: schematic
*
* User: jeffhu
* May 24 15:58:40 2006
***************************************
*

* HSIM Simulation Parameters
.param HSIMOUTPUT=psfbin


.param HSIMACOUTFMT=7

.param HSIMDCOUTFMT=7



.include "../netlist/input.scs"
.op 0 8u
.param hsimdumpopi=1

.end
```

Figure 123 Netlist Result Window

Running Simulations

The GUI shown in Figure 124 is used to invoke simulation.



Status: ...successful. T=25 C Simulator: hsim 3

Library mylib_rlc
Cell test_rlc
View schematic

#	Name	Value
1	hsimsp..	5

Figure 124 Running Simulations Window

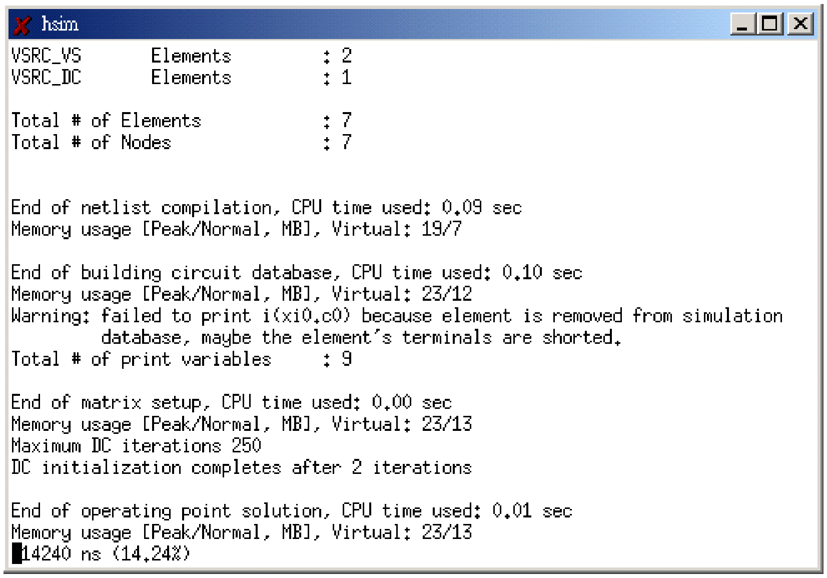
To invoke a simulation, select either of the following:

- GO

  🚦

- (Green) traffic light to run the simulation.
- Simulation -> Run

An xterm window is displayed where simulation status can be monitored as shown in Figure 125. From this window, the HSIM simulation process can be interrupted using [Ctrl] [C], which enters the interactive mode.



```
VSRC_VS       Elements         : 2
VSRC_DC       Elements         : 1

Total # of Elements            : 7
Total # of Nodes               : 7


End of netlist compilation, CPU time used: 0.09 sec
Memory usage [Peak/Normal, MB], Virtual: 19/7

End of building circuit database, CPU time used: 0.10 sec
Memory usage [Peak/Normal, MB], Virtual: 23/12
Warning: failed to print i(xi0.c0) because element is removed from simulation
         database, maybe the element's terminals are shorted.
Total # of print variables     : 9

End of matrix setup, CPU time used: 0.00 sec
Memory usage [Peak/Normal, MB], Virtual: 23/13
Maximum DC iterations 250
DC initialization completes after 2 iterations

End of operating point solution, CPU time used: 0.01 sec
Memory usage [Peak/Normal, MB], Virtual: 23/13
14240 ns (14.24%)
```

Figure 125 XTERM Status Window

Simulation can be aborted by selecting either of the following:

- STOP
- 🚦 (Red) traffic light to Stop the simulation.
- Simulation -> Stop

Viewing Results

Refer to the Cadence Virtuoso Analog Design Environment user documentation for details of this operation.

Waveforms

Waveform results are automatically displayed in a pop-up window for signals saved as plotted when a simulation finishes. Saved signals can also be viewed by the Results Browser under the Tools menu to check waveforms for signals marked as saved. The HSIM-Virtuoso Interface saves results from multiple simulations. The Results Browser can also be used to switch among the multiple results.

Annotation

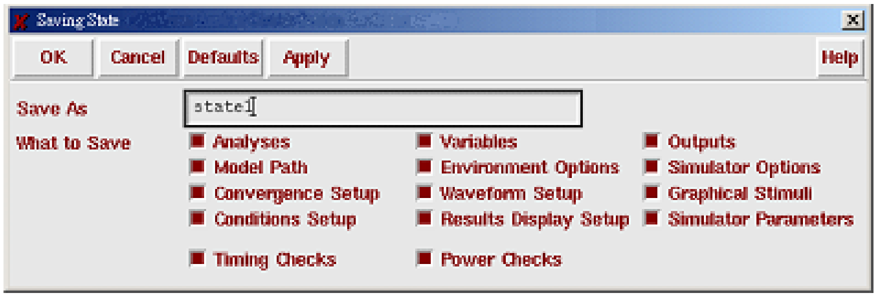
Simulation results can be annotated to schematics. Refer to Cadence user documentation for details on annotating simulation results.

Note:

> Annotation is only possible for the most recent simulation results.

Load and Save Sessions

The HSIM-Virtuoso Interface permits a session to be saved for future reference using the GUI shown in Figure 126.



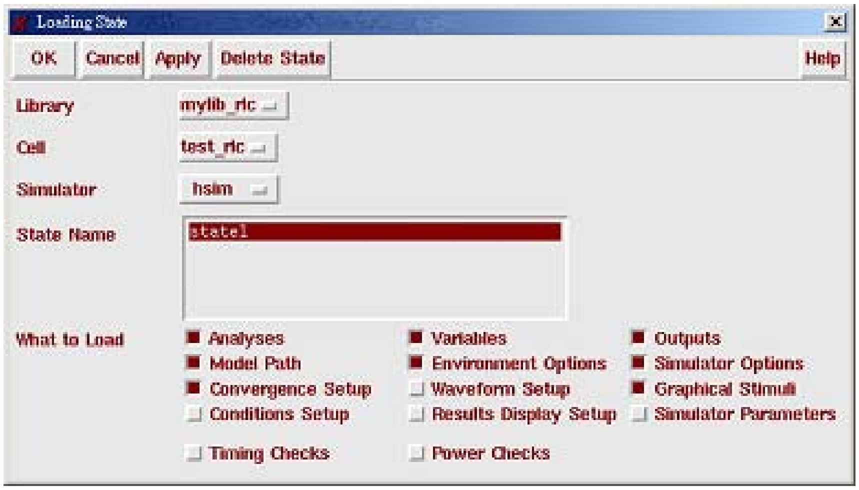
Save As: state1

What to Save:

■ Analyses
■ Model Path
■ Convergence Setup
■ Conditions Setup
■ Timing Checks
■ Variables
■ Environment Options
■ Waveform Setup
■ Results Display Setup
■ Power Checks
■ Outputs
■ Simulator Options
■ Graphical Stimuli
■ Simulator Parameters

Figure 126 Save a Session Window

To save a session, choose Session -> Save State. Sessions can be loaded using the GUI shown in Figure 127.



Library mylib_rlc

Cell test_rlc

Simulator hsim

State Name state1

What to Load

■ Analyses ■ Variables ■ Outputs
■ Model Path ■ Environment Options ■ Simulator Options
■ Convergence Setup □ Waveform Setup ■ Graphical Stimuli
□ Conditions Setup □ Results Display Setup □ Simulator Parameters
□ Timing Checks □ Power Checks

Figure 127 Load a Session Window

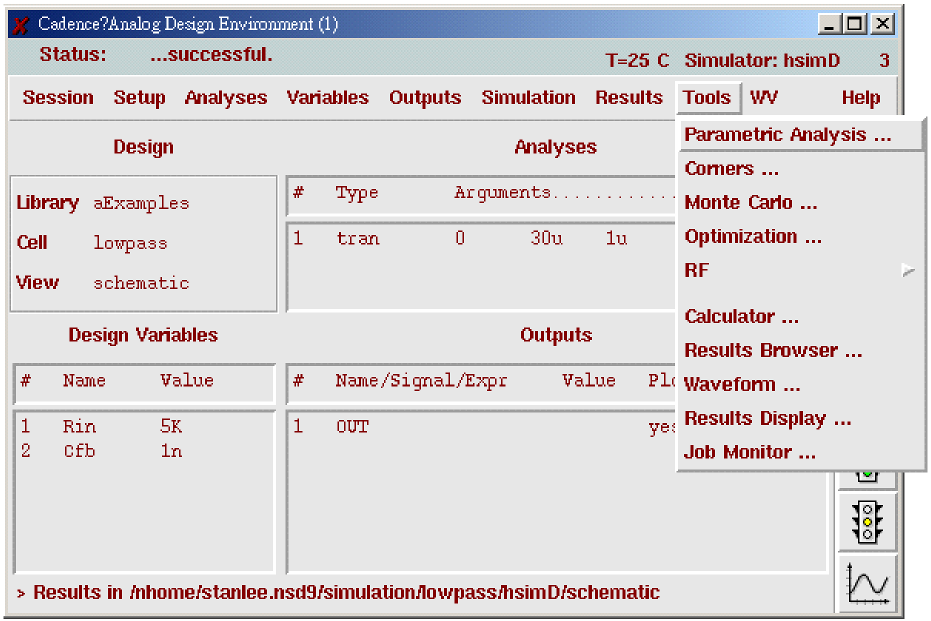
To load a session, choose Session -> Load State.

Monte Carlo Analysis

HSIM's Monte Carlo Analysis function is currently supported only using direct integration (HsimD).

To use Monte Carlo analysis, perform the following steps:

1. Select HsimD as the simulator.
2. Select Tools->Monte Carlo, as shown in Figure 128.



Status: ...successful. T=25 C Simulator: hsimD 3

Design

Library aExamples
Cell lowpass
View schematic

Analyses

#	Type	Arguments...........		
1	tran	0	30u	1u

Design Variables

#	Name	Value
1	Rin	5K
2	Cfb	1n

Outputs

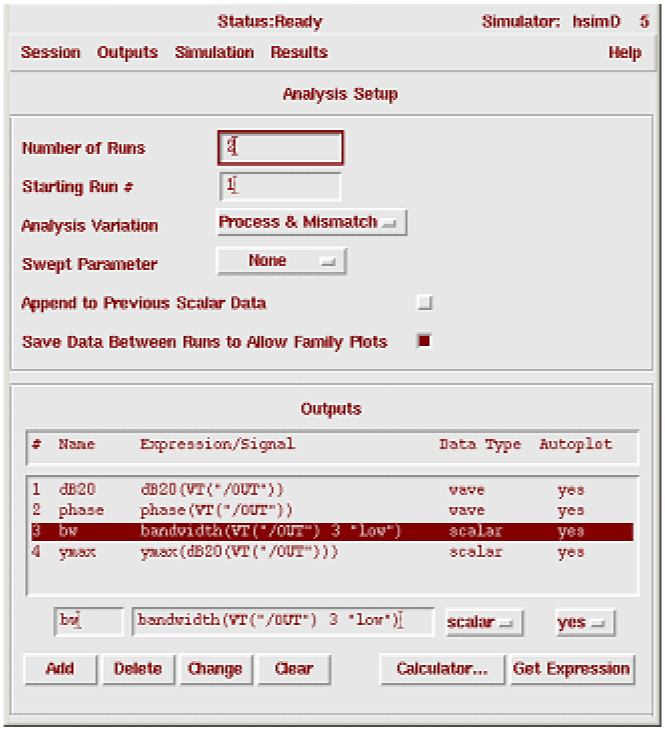
#	Name/Signal/Expr	Value	Plo
1	OUT		yes

> Results in /nhome/stanlee.nsd9/simulation/lowpass/hsimD/schematic

© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 128 Virtuoso Analog Design Environment Window

This displays the Virtuoso Statistical Analysis form. Usages and setups on this form are inherited from the generic Cadence form shown in Figure 129.



Number of Runs 3

Starting Run # 1

Analysis Variation Process & Mismatch

Swept Parameter None

Append to Previous Scalar Data □

Save Data Between Runs to Allow Family Plots ■

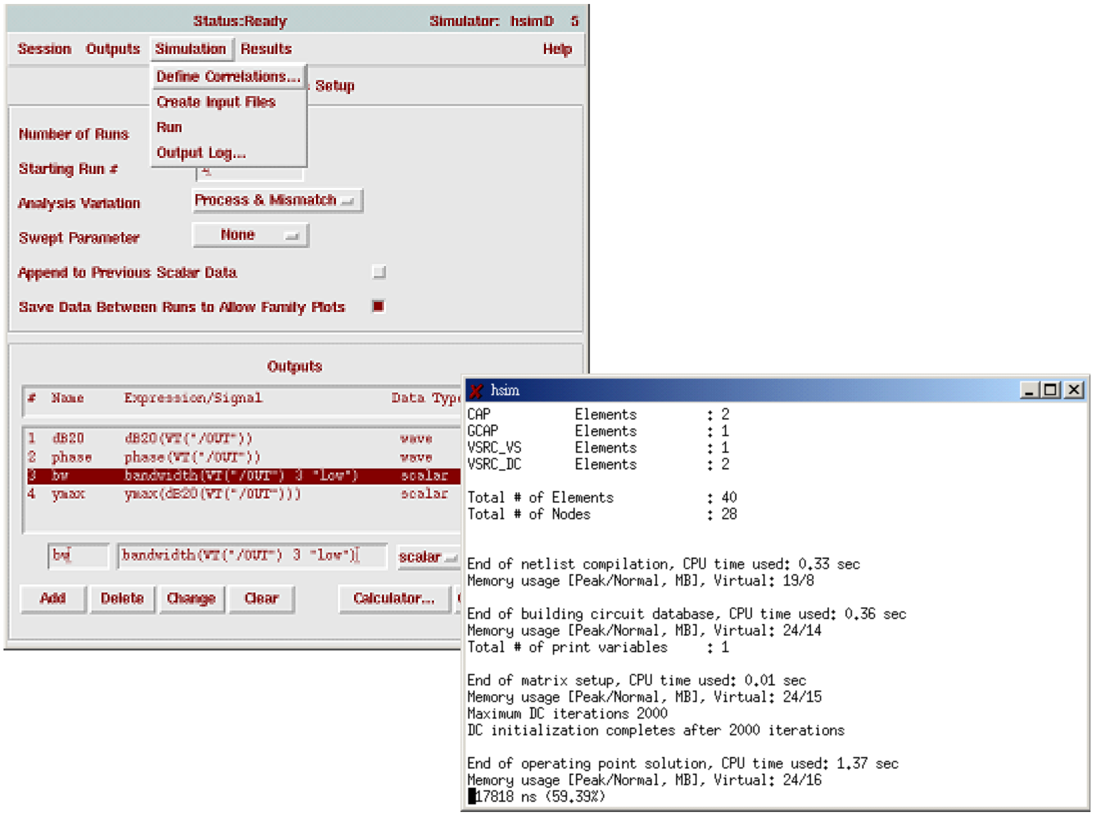
#	Name	Expression/Signal	Data Type	Autoplot
1	dB20	dB20(VT("/OUT"))	wave	yes
2	phase	phase(VT("/OUT"))	wave	yes
3	bw	bandwidth(VT("/OUT") 3 "low")	scalar	yes
4	ymax	ymax(dB20(VT("/OUT")))	scalar	yes

bw bandwidth(VT("/OUT") 3 "low") scalar yes

Figure 129 Virtuoso Analog Statistical Analysis Window

3. Invoke simulation using Statistical Analysis->Simulation->Run. A shell console will pop up as shown in Figure 130.

4. Once simulation completes, the selected output data is displayed as shown in Figure 131.



Number of Runs

Starting Run #

Analysis Variation: Process & Mismatch

Swept Parameter: None

Append to Previous Scalar Data □

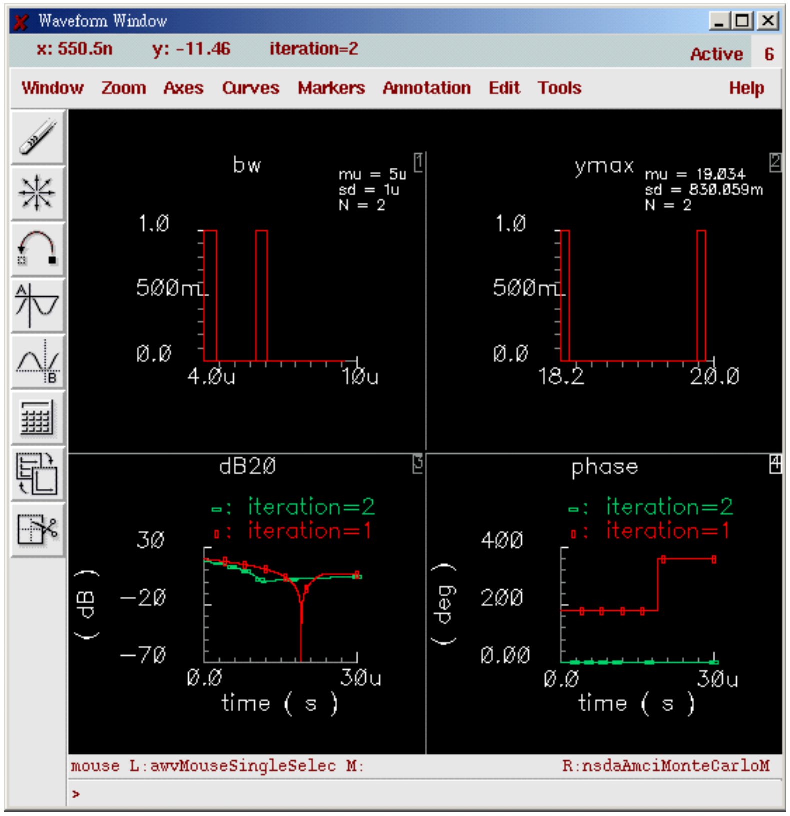
Save Data Between Runs to Allow Family Plots ■

#	Name	Expression/Signal	Data Type
1	dB20	dB20(VT("/OUT"))	wave
2	phase	phase(VT("/OUT"))	wave
3	bw	bandwidth(VT("/OUT") 3 "low")	scalar
4	ymax	ymax(dB20(VT("/OUT")))	scalar

bw | bandwidth(VT("/OUT") 3 "low") | scalar

```
CAP             Elements          : 2
GCAP            Elements          : 1
VSRC_VS         Elements          : 1
VSRC_DC         Elements          : 2

Total # of Elements               : 40
Total # of Nodes                  : 28


End of netlist compilation, CPU time used: 0.33 sec
Memory usage [Peak/Normal, MB], Virtual: 19/8

End of building circuit database, CPU time used: 0.36 sec
Memory usage [Peak/Normal, MB], Virtual: 24/14
Total # of print variables        : 1

End of matrix setup, CPU time used: 0.01 sec
Memory usage [Peak/Normal, MB], Virtual: 24/15
Maximum DC iterations 2000
DC initialization completes after 2000 iterations

End of operating point solution, CPU time used: 1.37 sec
Memory usage [Peak/Normal, MB], Virtual: 24/16
17818 ns (59.39%)
```

Figure 130 Simulation Window



```
mouse L:awvMouseSingleSelec M:                    R:nsdaAmciMonteCarloM
>
```

Figure 131 Waveform Output Window for Monte Carlo Analysis

16

MOSRA Stressed Model Application

Provides information on the MOSFET Reliability Analysis Option's StressedModel API functions.

MOSFET reliability analysis (MOSRA) uses the following stressed model APIs:

stressedModelConfig API

Syntax

```
int stressedModelConfig(char *config_name, double
   config_value);
```

Description

stressedModellConfig is the initialization API for the Stressed Model Equation Library and the API passes the values of configuration parameters to the library. This API is called only once.

Arguments

config_name

Configuration command names. The possible configuration parameters are:

- hci_eage_threshold
- nbti_eage_threshold

- hci_eage_sampling
- nbti_eage_sampling

config_value

Configuration parameter values

Returns

0

Success

1

Error

Example

```
stressedModelConfig(config_name, config_value)
char *config_name;
double config_value;
{
/* get the threshold and sampling values from hsim */
if (!strcmp(config_name,”hci_eage_threshold”))
     hci_eage_threshold=config_value;
else
if (!strcmp(config_name,”hci_eage_threshold”))
     nbti_eage_threshold=config_value;
else if (!strcmp(config_name,”hci_eage_sampling”))
     hci_eage_sampling=config_value;
else if (!strcmp(config_name,”nbti_eage_sampling”))
     nbti_eage_sampling=config_value;
return 0;
}
/* end stressedModelConfig */
```

stressedModelInit API

Syntax

```
int stressedModelInit(char *orig_model_name, double
   a_transistor_W, double a_transistor_L, double
   a_transistor_nbti_eage, double a_transistor_hci_eage,
   char **new_model_name);
```

Description

stressedModelInit will be called for each transistor. The original model name, the transistor geometries (W and L) and the transistor HCI and NBTI eage are passed to the Stressed Model Equation Library through this API. The Stressed Model Equation Library should use HCI and NBTI threshold values from stressedModelConfig to decide if a new model needs to be created. If a new model needs to be created, then the Stressed Model Equation Library uses the HCI and NBTI sampling values from stressedModelConfig to sample and to decide which new model. The new model name will be created and be stored in new_model_name.

Arguments

orig_model_name

Original model name

a_transistor_W

Current transistor width

a_transistor_L

Current transistor length

a_transistor_hci_eage

Current transistor HCI electron age

a_transistor_nbti_eage

Current transistor NBTI electron age

new_model_name

New model name

Returns

1

If a new model needs to be created and the new model name is stored in new_model_name argument.

0

Do not create a new model.

Example

```
stressedModelInit(orig_model_name, a_transistor_w,
a_transistor_l, a_transistor_nbti, a_transistor_hci,
new_model_name)
char *orig_model_name, **new_model_name;
double a_transistor_w, a_transistor_l, a_transistor_nbti,
a_transistor_hci;
{
     int count=0;
     int ret=0; /*no new model*/
     if        (a_transistor_hci>hci_eage_threshold) ||
               (a_transistor_nbti>nbti_eage_threshold)) {
               /* test if we need to create a new model*/
               ret=1; /*a new model needs to be created*/
               /* computes a unique new model number (depending
on L, W
               and the hci and nbti values */
               count=(int) fmod(a_transistor_w * 1e9 +
a_transistor_l*
               1e9 + a_transistor_nbti*3e30+ a_transistor_hci
*5e30,
               10001);
               sprintf(mycsm_new_model_name,
“%s_stressed_model_%d”,
               orig_model_name, count);
               *new_model_name=mycsm_new_model_name;
     return ret;
} /* end stressedModelInit */
```

stressedModelParamNameList API

Syntax

```
int stressedModelParamNameList(char *orig_model_name,char
   *new_model_name, char** model_param_name_list);
```

Description

The stressedModelParamNameList API makes the Stressed Model Equation Library return a list of model parameter names for the specified original model name and the new model name. These model parameter names will be used in the stressedModelParamVal API. These model parameter names are separated by comma in the model_param_name_list, e.g. vth0,xyz.

Arguments

orig_model_name

Original model name.

new_model_name

stressedModelInit model name

model_param_name_list

Comma separated model parameter name list; for example: vth0,xyz.

Returns

1

If there is a list of model parameter names in model_param_name_list.

0

NO change.

Example

```
extern int stressedModelParamNameList (orig_model_name,
new_model_name, model_param_name_list)
char *orig_model_name, *new_model_name, **model_param_name_list;
{
 int ret=1;
 /* Customer specific models modified parameters */
 sprintf(mycsm_model_name_list, "vth0");
 *model_param_name_list=mycsm_model_name_list;
 return ret;
} /* end stressedModelParamNameList */
```

stressedModelParamNameAndVal API

Syntax

```
int stressedModelParamNameAndVal(char *orig_model_name,
   char*new_model_name,char*model_param_name,
   double model_param_val);
```

Description

The stressedModelParamNamAndVal API passes model parameter values to the Stressed Model Equation Library.

Arguments

orig_model_name

Original model name

new_model_name

Model name from stressedModelInit

model_param_name

Model parameter name

model_param_value

Model parameter value

Returns

1

OK

0

Not OK

Example

```
extern int stressedModelParamNameAndVal (orig_model_name,
new_model_name, model_param_name, model_param_val)
char *orig_model_name, *new_model_name, *model_param_name;
double model_param_val;
{
     int ret=1;
     /* Customer specific models modified parameters */
     fprintf(stdout,”\norig_model_name=%s\n”, orig_model_name);
     fprintf(stdout,”new_model_name=%s\n”, new_model_name);
     fprintf(stdout,”%s=%g\n”,model_param_name,
     model_param_value);
     return ret;
} /* end stressedModelParamNameAndVal*/
```

stressedModelParamVal API

Syntax

```
int stressedModelParamVal(char *orig_model_name, char
   *new_model_name, char* model_param_name, double
   orig_model_param_value, double *new_model_param_value);
```

Description

The stressedModelParamVal API calls the Stressed Model Equation Library to obtain the new value of a specified parameter by using the following data:

- Original model name
- New model name
- Original value of the model parameter
- Transistor Geometries (W and L) from stressedModelInit
- Transistor HCI and NBTI eage from stressedModelInit

This API should be called for either of the following conditions:

- If stressedModelInit returns a 1
- For each model parameters of the given model name

The new value of the model parameter will be stored in new_model_param_value.

Arguments

orig_model_name

Original model name.

new_model_name

Model name from stressedModelInit

model_param_name

Model name parameter.

original_model_param_value

Original value of the specified model parameter

new_model_param_value

New model parameter value

Returns

1

If there is a new value for the specified model parameter

0

NO change

Example

```
extern int stressedModelParamVal(orig_model_name,
new_model_name, model_param_name,
orig_model_param_value, new_model_param_value)
char *orig_model_name, *new_model_name, *model_param_name;
double orig_model_param_value, *new_model_param_value;
{
 static int count=1;
 /* Customer specific equations */
 if (!strcmp(model_param_name,”vth0”))
*new_model_param_value=orig_model_param_value * .8;
 count++;
 return 1;
} /* end stressedModelParamVal */
```

17

User Reliability Interface

Provides information on how HSIM permits user-specified reliability interface (URI) models in addition to hot carrier injection (HCI) and negative bias temperature instability (NBTI).

URI Model

The URI model is a dynamic library implemented in the C language and the application procedure interface (API) provided by Synopsys. In the example used in this section, uri.so is built from the following files:

- URI.h
- URI.c
- b3uri.h
- b3urimain.c
- b3uriread.c
- b3uriset.c
- b3urild.c

The files are described in the following sections.

Dynamic Library

This section describes how to build the dynamic library. The buildfmod file, located in $HSIM_HOME/bin, can be used to compile the URI source code. The one line command to compile the URI follows:

```
buildmod uri.so URI.c b3urimain.c b3uriread.c b3uriset.c
b3urild.c
```

This command compiles the C files into the dynamic library file uri.so.

HSIMURILIB

To perform a simulation run with an URI model, add the following line in the input netlist:

```
.param HSIMURILIB="./uri.so"
```

In the above code, uri.so is the name holder for the dynamic library.

User Files

URI Header File (URI.h)

Caution!

Do not alter the file URI.h.

The URI.h header file shown in Example 78 on page 621 defines the communication protocol between HSIM and the dynamic library uri.so. The initialize() function is first called by HSIM.

The URI_VAR user reliability interface variable structure defines the variables for the device instance. The tnom element is the nominal temperature and temp is the simulation temperature. The variables vds, vgs, … igb are specific to the device instance that are passed from the HSIM simulator to the dynamic library.

Reliability Parameters

The reliability input parameters that are used for the standard HSIM simulator are specified in the following:

- relmoslevel
- hcih
- hcim
- na
- nhv
- nga

Reliability Variables

The following are examples of reliability effects that are calculated in this custom URI model. These reliability modes are returned from the URI routines to the HSIM simulator:

- isub: Substrate current
- hci_stress: User calculated HCI stress value
- nbti_stress: User calculated NBTI stress value

The following reliability variables should be set to 1 if they are calculated with URI. If any of the variables are set to 0, the standard HSIM equation will be used for the variable.

Note:

The following NBTI equations are NOT implemented in HSIM.

- subGiven
- hci_stressGiven
- nbti_stressGiven

Additional Stress Variables

There are five additional user-specified stress variables that can be used to perform additional calculations. These variables are declared as stress_value[5]. If any of the variables are to be calculated in this custom implementation, the corresponding variables in the stress_valueGiven array should be set to 1.

Note:

These stress variables are not needed to calculate isub, hci_stress or nbti_stress.

The UserRelDef structure contains information to assist HSIM to allocate an adequate amount of memory space. It also contains function names for HSIM to call. The pModel address helps link the model structure specified in the dynamic library uri.so back to HSIM. The value modelsize notifies HSIM about the required memory size for the particular model structure.

Other Functions

In addition to initialize() function, HSIM calls other functions such as:

- initial_mode()
- read_model()
- set_model()
- model_load()
- start() (Note: This function is optional.)
- conclude() (Note: This function is optional.)

URI.h should be kept intact without any modification because a copy exists inside HSIM to facilitate data communication between HSIM and the dynamic library uri.so. The content of the URI.h file is shown in Example 78.

Note:

Contact a Synopsys application engineer for updates to the content of the URI.h file.

Example 78 URI.h file example

```
#ifndef URI_H
#define URI_H
/*********************************************************
 **** CAUTION!   ATTENTION!   CAUTION!!
 **** Do no change this headers file.
 **** Obtain notice from Synopsys first
 **** before make any change
*********************************************************/
#if defined(WIN32)
__declspec(dllexport) char * initialize();
#else
char * Initialize();
#endif
/* user reliability interface functions */
typedef struct UserRelDef {
#ifdef __STDC__
     char                ModelName[80];
     char                *pModel;
     int                 modelsize;
     void                (*initial_model)(char*, int);
     void                (*read_model)(char*, char*, double, int
*);
     void                (*set_model)(char*, char *);
     void                (*model_load)(char*, char*);
     void                (*start)();             /*Optional */
     void   (*conclude)();                       /* Optional */
#else
     char                ModelName[80];
     char                *pModel;
     int                 modelsize;
     void                (*initial_model)();
     void                (*read_model)();
     void                (*set_model)();
     void                (*model_load)();
     void                (*start)();             /* Optional */
     void                (*conclude)();          /* Optional */
#endif
} UserRelDef;
/* User Reliability Interface (URI) variables */
typedef struct URI_VAR {
          /* Convention;
                    "HSIM:" means the value is sent from HSIM
simulator to
                         dynamic library.
                    "USER:" means the value is returned from
dynamic
                         library to HSIM
          */
 /* simulation constants */
double tnom;                                     /* HSIM: model
norm temperature, in
                                            degree-K */
double temp;                                     /* HSIM:
simulation temperature, in
                                                 degree-K */
/* variables */
double vds;                                      /*HSIM: vds
bias */
double vgs;                                      /*HSIM: vgs
bias */
double vbs;                                      /*HSIM: vbs
bias */
double w;                                        /*HSIM:
transistor width in meter unit */
double l;                                        /*HSIM:
transistor length in meter unit */
double leff;                                     /*HSIM:
effective channel length in meter
                                                 unit */
double weff;                                     /*HSIM:
effective channel width in meter
                                                 unit */
double alpha0;                                   /*HSIM: 1st
parameter for impact
                                                 ionization */
double alpha1;                                   /*HSIM: Isub
parameter for length
                                                 scaling */
double beta0;                                    /*HSIM: 2nd
parameter for impact
                                                 ionization */
double vth0;                                     /*HSIM: zero-
biased threshold voltage */
double u0;                                       /*HSIM:
transistor mobility */
double rds;                                      /*HSIM:
parasitic resistance in ohm */
double vth;                                      /*HSIM:
threshold voltage (i.e., turn-on
                                                 voltage) */
double vdsat;                                    /*HSIM:
saturation drain voltage */
double vdseff;                                   /*HSIM:
effective vds */
double va;                                       /*HSIM: total
Early voltage */
double idsa;                                     /*HSIM:
intermediate value for channel
```

```
                                                 current */
double ids;                                      /*HSIM: drain
(channel) current (not
                                                 include isub)
*/
double gds;                                      /*HSIM: output
conductance (dIds/dVds) */
double gm;                                       /*HSIM: trans-
conductance (dIds/dVgs) */
/* not available for BSIM3 */
double igd;                                      /*HSIM: gate-
to-drain static current */
double igs;                                      /*HSIM: gate-
to-source static current */
double igb;                                      /*HSIM: gate-
to-bulk static current */
/* reliability parameters */
double relmoslevel;                              /*HSIM: action
selector */
double hcih;                                     /*HSIM: HCI
pre-factor coefficient */
double hcim;                                     /*HSIM:
exponent for isub to ids ratio */
double na;                                       /*HSIM: NBTI
pre-factor coefficient */
double nhv;                                      /*HSIM: NBTI
activation energy per
                                                 electron
charge */
double nga;                                      /*HSIM: NBTI
coefficient for gate
                                                 biasing */
double simulate_time;                            /*HSIM:
present simulation time for
                                                 the particular
device */
/*        user constant flags, set by you */
unsigned int isubGiven :1;
/*        USER: this flag shall be set by you if user-
          calculated isub is to be used at HSIM */
unsigned int hci_stressGiven :1;
/*        USER: this flag shall be set by you if user-
          calculated hci_stress is to be used at HSIM */
unsigned int nbti_stressGiven :1;
/*        USER: this flag shall be set by you if user-
          calculated nbti_stree is to be used at HSIM */
/*        return data */
/*        Non-Negative, i.e. POSITIVE value or ZERO returned
          from user */
double isub;                                     /*USER: drain-
```

```
to-bulk static current for
                                                 hot-carrier
purpose only */
double hci_stress;                               /*USER:
OPTIONAL user-calculated stress
                                                 value for HCI
*/
double nbti_stress;                              /*USER:
OPTIONAL user-calculated stress
                                                 value for NBCI
*/
/* user constant flags, set by you */
int stress_valueGiven[5];
double stress_value[5];
} URI_VAR;
#ifndef NULL
#define NULL 0
#endif
#define EXP_THRESHOLD 34.0
#define MIN_EXP 1.713908431e-15
#endif /* URI_H */
```

URI Interface File (URI.c)

The interface file URI.c contains one interface function initialize(). It is the first function inside the dynamic library uri.so that is called by HSIM. HSIM sends the type of the device (1 for n_MOSFET and -1 for p_MOSFET). URI.c returns the address of the device model and also calls the initial_model() function to grab the memory space for the model. An example of the URI.c is listed Example 79.

Example 79 URI.c file example.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "URI.h"
extern UserRelDef *p_b3uri;
char *
#ifdef __STDC__
initialize(
int   type,
char *name)
#else
initialize(type, name)
int   type;
char *name;
#endif
{
int                 length;
char                *pDevice= NULL;
/*
** select model level **
*/
pDevice = (char *)p_b3uri;
length = strlen(name);
if (length < 79)
strcpy(((UserRelDef*)pDevice)->ModelName, name);
else {
strncpy(((UserRelDef*)pDevice)->ModelName, name, 79);
((UserRelDef*)pDevice)->ModelName[79] = (char)NULL;
}
((UserRelDef*)pDevice)->initial_model(((UserRelDef*)pDevice)-
>pModel,type);
return pDevice;
}
```

User-Defined Header File (b3uri.h)

The b3uri.h user-defined header file contains the `struct` definitions for your defined reliability model. The variables are additional model parameters that you specify through the model card. In the example, thirteen model parameters are defined. As each of each parameter is read from the model card defined in Example 80, variables such as simsublevel are assigned the value and the corresponding Given variable such as stmisublevelGiven is set. An example of the b3uri.h is shown in Example 80.

Example 80 b3uri.h file example.

```
#ifndef B3URI_H
#define B3URI_H
typedef struct URImodel
{
int type;
int stmisublevel;
double stmler;
double stmwer;
double stma1r;
double stmsta1;
double stmsla1;
double stmswa1;
double stma2r;
double stmsla2;
double stmswa2;
double stma3r;
double stmsla3;
double stmswa3;
unsigned int typeGiven                           :1;
unsigned int stmisublevelGiven                   :1;
unsigned int stmlerGiven                         :1;
unsigned int stmwerGiven                         :1;
unsigned int stma1rGiven                         :1;
unsigned int stmsta1Given                        :1;
unsigned int stmsla1Given                        :1;
unsigned int stmswa1Given                        :1;
unsigned int stma2rGiven                         :1;
unsigned int stmsla2Given                        :1;
unsigned int stmswa2Given                        :1;
unsigned int stma3rGiven                         :1;
unsigned int stmsla3Given                        :1;
unsigned int stmswa3Given                        :1;
} URImodel;
#define LN_MINDOUBLE                             -800
#endif /* B3URI */
```

Primary Model File (b3urimain.c)

This is the primary source file for your reliability model. It contains the actual implementation of the b3uri_initialmodel() function that corresponds to the initial_model() function described in URI.h header file. The optional b3uri_start() function that corresponds to the start() function and the optional b3uri_conclude() function that corresponds to the conclude() function. The 3urimain.c file contents are shown in Example 81.

Example 81 b3urimain.c file example.

```
/* Module                                           :b3urimain.c
*/
/* Last Update                                      :May 2003
*/
/* Description                                      : Main
interface function for BSIM3v3.2
                                                    reliability
interface */
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include "URI.h"
#include "b3uri.h"
#ifdef __STDC__
void b3uri_initialmodel(char*,int);
void b3uri_read_model(char*, char*, double, int *);
void b3uri_set_model(char*, char*);
void b3uri_model_load(char*, char*);
void b3uri_start();
void b3uri_conclude();
extern void b3uri_read(URImodel*, char*, double, int *);
extern void b3uri_set(URI_VAR *, URImodel*);
extern void b3uri_load(URI_VAR *, URImodel*);
#else
void b3uri_initialmodel();
void b3uri_read_model();
void b3uri_set_model();
void b3uri_model_load();
void b3uri_start();
void b3uri_conclude();
extern void b3uri_read();
extern void b3uri_set();
extern void b3uri_load();
#endif
/* local */
static URImodel _B3URIModel;
static UserRelDef b3uridef = {
"",
(char*)&_B3URIModel,
sizeof(URImodel),
b3uri_initialmodel,
b3uri_read_model,
b3uri_set_model,
b3uri_model_load,
b3uri_start,                                        /* Optional,
replace with NULL if
                                                    not used */
b3uri_conclude,                                  /* Optional,
replace with NULL if
                                                 not used */
};
UserRelDef *p_b3uri = &b3uridef;
/*** PARTICULAR MODEL INTERFACE SUBROUTINE ***/
void
#ifdef __STDC__
b3uri_initialmodel(
char *urimodel,
int   type)
#else
b3uri_initialmodel(urimodel, type)
char *urimodel;
int   type;
#endif
{
/* initialization */
(void)memset(urimodel, 0, sizeof(URImodel));
if(type == -1) {
((URImodel*)urimodel)->type = -1;
((URImodel*)urimodel)->typeGiven = 1;
}
else {
((URImodel*)urimodel)->type = 1;
}
return;
}
void
#ifdef __STDC__
b3uri_read_model(
char   *urimodel,
char   *name,
double value,
int *notfound)
#else
b3uri_read_model(urimodel, name, value, notfound)
char   *urimodel;
char   *name;
double value;
int    *notfound;
#endif
{
b3uri_read((URImodel*)urimodel, name, value, notfound);
return;
}
void
#ifdef __STDC__
```

```
b3uri_set_model(
char *urivar,
char *urimodel)
#else
b3uri_set_model(urivar, urimodel)
char *urivar;
char *urimodel;
#endif
{
b3uri_set((URI_VAR*)urivar, (URImodel*)urimodel);
return;
}
void
#ifdef __STDC__
b3uri_model_load(
char *urivar,
char *urimodel)
#else
b3uri_model_load(urivar, urimodel)
char *urivar;
char *urimodel;
#endif
{
b3uri_load((URI_VAR*)urivar, (URImodel*)urimodel);
return;
}
void
#ifdef __STDC__
b3uri_start()
#else
b3uri_start()
#endif
{
/* Use of this function is optional */
printf("Start: Use of this function is optional.\n");
return;
}
void
#ifdef __STDC__
b3uri_conclude()
#else
b3uri_conclude()
#endif
{
/* Use of this function is optional */
printf("Conclusion for URI: Use of this function is optional.\n");
return;
}
```

Model Parameter Processing File (b3uriread.c)

This file contains the b3uri_read_model() function which corresponds to the read_model() function. For each pair of the model parameter and its associated value, HSIM call this function once. If the model parameter is recognized by this function, the value is stored. Otherwise, this function prints a Warning message. An example is listed below.

Example 82 b3uriread.c file example.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "b3uri.h"
void
#ifdef __STDC__
b3uri_read(
URImodel *urimodel,
char *pname,
double value,
int *notfound)
#else
b3uri_read(urimodel, pname, value, notfound)
URImodel *urimodel;
char *pname;
double value;
int *notfound;
#endif
{
/* all in lower case */
if(strcmp(pname,"stmisublevel") == 0) {
urimodel->stmisublevel = (int)value;
urimodel->stmisublevelGiven = 1;
}
else if(strcmp(pname,"stma1r") == 0) {
urimodel->stma1r = (double)value;
urimodel->stma1rGiven = 1;
}
else if(strcmp(pname,"stmsta1") == 0) {
urimodel->stmsta1 = (double)value;
urimodel->stmsta1Given = 1;
}
else if(strcmp(pname,"stmsla1") == 0) {
urimodel->stmsla1 = (double)value;
urimodel->stmsla1Given = 1;
}
else if(strcmp(pname,"stmswa1") == 0) {
urimodel->stmswa1 = (double)value;
urimodel->stmswa1Given = 1;
}
else if(strcmp(pname,"stma2r") == 0) {
urimodel->stma2r = (double)value;
urimodel->stma2rGiven = 1;
}
else if(strcmp(pname,"stmsla2") == 0) {
urimodel->stmsla2 = (double)value;
urimodel->stmsla2Given = 1;
}
else if(strcmp(pname,"stmswa2") == 0) {
urimodel->stmswa2 = (double)value;
urimodel->stmswa2Given = 1;
}
else if(strcmp(pname,"stma3r") == 0) {
urimodel->stma3r = (double)value;
urimodel->stma3rGiven = 1;
}
else if(strcmp(pname,"stmsla3") == 0) {
urimodel->stmsla3 = (double)value;
urimodel->stmsla3Given = 1;
}
else if(strcmp(pname,"stmswa3") == 0) {
urimodel->stmswa3 = (double)value;
urimodel->stmswa3Given = 1;
}
else if(strcmp(pname,"stmler") == 0) {
urimodel->stmler = (double)value;
urimodel->stmlerGiven = 1;
}
else if(strcmp(pname,"stmwer") == 0) {
urimodel->stmwer = (double)value;
urimodel->stmwerGiven = 1;
}
else {
*notfound = 1;
}
return;
}
```

Model Default Setting File (b3uriset.c)

This file contains the b3uri_set_model() function which corresponds to the set_model() function in the URI.h header file. HSIM calls this function to assign the default value to a model parameter if you do not provide an input value for that particular model parameter. An example is listed below.

Example 83 b3uriset.c file example.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "URI.h"
#include "b3uri.h"
void
#ifdef __STDC__
b3uri_set(
URI_VAR *urivar,
URImodel *urimodel)
#else
b3uri_set(urivar, urimodel)
URI_VAR *urivar;
URImodel *urimodel;
#endif
{
/* set some flags */
urivar->isubGiven = 1;
/* this flag shall be set by the user if isub calculated by
user is to be used instead of the default calculation inside HSIM */
urivar->hci_stressGiven = 0;
/* this flag shall be set by the user if hci_stress calculated
by user is to be used instead of the default calculation inside
HSIM */
urivar->nbti_stressGiven = 1;
/* this flag shall be set by the user if nbti_stress calculated
by user is to be used instead of the default calculation inside
HSIM */
/* this flag shall be set by the user in order to handle any
additional stress value */
urivar->stress_valueGiven[0] = 1; /* used */
urivar->stress_valueGiven[1] = 1;
urivar->stress_valueGiven[2] = 0; /* not used */
urivar->stress_valueGiven[3] = 0;
urivar->stress_valueGiven[4] = 0;
/* example to illustrate MOS11 weak-avalanche substrate current */
if (!urimodel->stmisublevelGiven)urimodel->stmisublevel=1;
if (!urimodel->stmlerGiven)                         urimodel-
>stmler                                             =1.0e-6;
if (!urimodel->stmwerGiven)                         urimodel-
>stmwer                                             =1.0e-5;
if (!urimodel->stma1rGiven)                         urimodel-
>stma1r                                             =6.0;
if (!urimodel->stmsta1Given)                        urimodel-
>stmsta1                                            =0.0;
if (!urimodel->stmsla1Given)                        urimodel-
>stmsla1                                            =0.0;
if (!urimodel->stmswa1Given)                        urimodel-
>stmswa1                                          =0.0;
if (!urimodel->stma2rGiven)                       urimodel-
>stma2r                                           =38.0;
if (!urimodel->stmsla2Given)                      urimodel-
>stmsla2                                          =0.0;
if (!urimodel->stmswa2Given)                      urimodel-
>stmswa2                                          =0.0;
if (!urimodel->stma3rGiven)                       urimodel-
>stma3r                                           =1.0;
if (!urimodel->stmsla3Given)                      urimodel-
>stmsla3                                          =0.0;
if (!urimodel->stmswa3Given)                      urimodel-
>stmswa3                                          =0.0;
return;
}
```

Model Evaluation and Load File (b3urild.c)

This file contains the b3uri_model_load() function which corresponds to the model_load() function in the URI.h header file. Refer to URI Header File (URI.h) on page 618 for additional information. The calculation of the substrate current Isub is included in the example. The hci_stress and nbti_stress are assigned the value of 0 in Example 84. You should implement the custom calculation of hci_stress and nbti_stress in this function. The stress_value[0] and stress_value[1] are assigned value in the example since stress_valueGiven[0] and stress_valueGiven[1] were set in b3urist.c.

Example 84 b3urild.c file example.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "URI.h"
#include "b3uri.h"

#undef CHECK_SIM_TIME
void
#ifdef __STDC__
b3uri_load(
URI_VAR *urivar,
URImodel *urimodel)
#else
b3uri_load(urivar, urimodel)
URI_VAR *urivar;
URImodel *urimodel;
#endif
{
/* MOS11 example */
double scl, scw, delta_tk, a1, a2, a3;
double x, f;
/* BSIM3 example */
double tmp, diffVds;
double T0, T1, T2, T3;
/* common */
double sim_time;
double Isub;
sim_time = urivar->simulate_time;
#if defined(CHECK_SIM_TIME)
{
printf("sim_time = %g\n", sim_time);
}
#endif
/* Substrate current begins */
if (urimodel->stmisublevel == 1) {
/* temperature updating */
scl = 1.0/urivar->leff - 1.0/urimodel->stmler;
scw = 1.0/urivar->weff - 1.0/urimodel->stmwer;
delta_tk = urivar->temp - urivar->tnom;
a1        =    (urimodel->stma1r + scl * urimodel->stmsla1 + scw *
               urimodel->stmswa1)(1.0 + delta_tk * urimodel-
               >stmsta1);
a2        =    urimodel->stma2r + scl * urimodel->stmsla2 + scw *
               urimodel->stmswa2;
a3             =        urimodel->stma3r + scl * urimodel-
>stmsla3 + scw *
                        urimodel->stmswa3;
/* substrate current calculation */
x              =        urivar->vds - a3 * urivar->vdsat;
if ((x <= - a2/LN_MINDOUBLE) || (a1 == 0.0) ) {
Isub           =        0.0;
}
else {
f              =        a1 * exp (- a2/x);
Isub           =        urivar->ids * f;
}
} /* MOS11 weak avalanche current example */
else {
/* BSIM3 static substrate current example */
diffVds = urivar->vds - urivar->vdseff;
tmp = urivar->alpha0 + urivar->alpha1 * urivar->leff;
if ((tmp <= 0.0) || (urivar->beta0 <= 0.0)) {
Isub           =        0.0;
}
else {
T2             =        tmp / urivar->leff;
if (diffVds > urivar->beta0 / EXP_THRESHOLD) {
T0             =        - urivar->beta0 / diffVds;
T1             =        T2 * diffVds * exp(T0);
T3             =        T1 / diffVds * (T0 - 1.0);
}
else {
T3             =        T2 * MIN_EXP;
T1             =        T3 * diffVds;
}
Isub           =        T1 * urivar->idsa;
}
} /* BSIM3 static substrate current example */
if (Isub < 0.0)
Isub           =        0.0; /* safe-guard */
/* return non-negative value to HSIM */
urivar->isub = Isub;
/* if hci_stress is calculated by user, then return the value to
HSIM */
#if 0
urivar->hci_stress = 0.0;
#endif
/* if nbti_stress is calculated by user, then return the value
to HSIM */
#if 0
urivar->nbti_stress = 0.0;
#endif
/* if stress_value[0] and stress_value[1] are used, then return the
suitable values to HSIM */
urivar->stress_value[0] = 1e-5;
```

```
urivar->stress_value[1] = 1e-8;
return;
} /* end, b3uriload() */
```

Index