HSIM® Simulation Reference

Version C-2009.06, June 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Contents

Part II: HSIM Advanced Analysis

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 xxviii)
- Documentation on the Web, which provides HTML and PDF documents and is available on SolvNet (see Accessing SolvNet on page xxviii)

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.

Part: 1 HSIM Core Basics

1

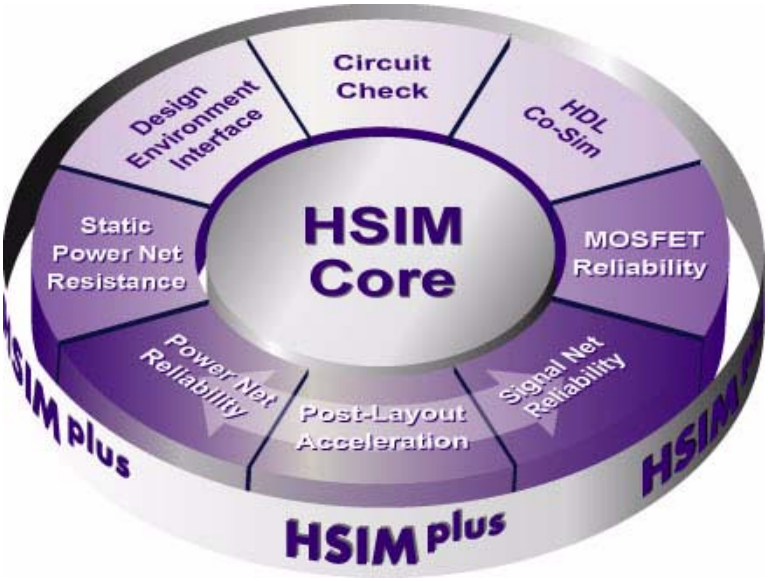
Introduction

Describes tool features and provides recommendations for use.

HSIM with HSIM^{plus} Features

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 3 shows the HSIM Core with 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 Features

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 Chapter 12, AC Small-Signal Analysis for details.

Interactive Circuit Analysis

HSIM 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 Chapter 15, Timing and Power Analysis for detailed information about check commands.

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

Hierarchical Simulation Technology

HSIM 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 isomorphic matching techniques eliminates redundant computation for identical subcircuits 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

Simulation Flow

Describes the HSIM simulation flow including parsing a SPICE-compatible ASCII input netlist, optional parasitic reduction, building and optimizing the hierarchical simulation database, DC initialization, transient simulation, and outputting the results in ASCII or a binary format. Additionally, a Quick Start section provides the tools to aid in determining which commands should be used to achieve optimum performance and precision trade-offs by quickly and easily controlling some of the HSIM algorithms.

Simulation Flow Overview

The HSIM simulation flow begins by parsing a SPICE-compatible ASCII input netlist as described in Chapter 7, Input Netlist. Optional parasitic RC reduction can be executed on a netlist that contains significant parasitic data. The RC reduction operation reduces each interconnect network into a significantly smaller network with equivalent timing behavior, which increases the efficiency of the HSIM simulation.

Layout parasitics can be back-annotated through a Detailed Standard Parasitic Format (DSPF) or Standard Parasitic Exchange Format (SPEF) route in which one or more files containing the extracted interconnect parasitic RC data are parsed, reduced under your control, and applied to the circuit being simulated. Both signal and power net DSPF/SPEF files pass through similar, but unique, back-annotation sub-flows.

The device parameter format back-annotation capability supports this flow. In this sub-flow, extracted device geometries (W, L, AS, AD, PS, PD, etc.) are taken from the LPE or ideal netlist, and back-annotated to the active devices contained in the original pre-layout schematic netlist. Refer to Chapter 8, Post-Layout Back-Annotation.

After the netlist is parsed in, HSIM builds and optimizes the hierarchical simulation database. In most cases, memory usage peaks at this point. After

the simulation database has been built, storage requirements typically decrease prior to transient simulation.

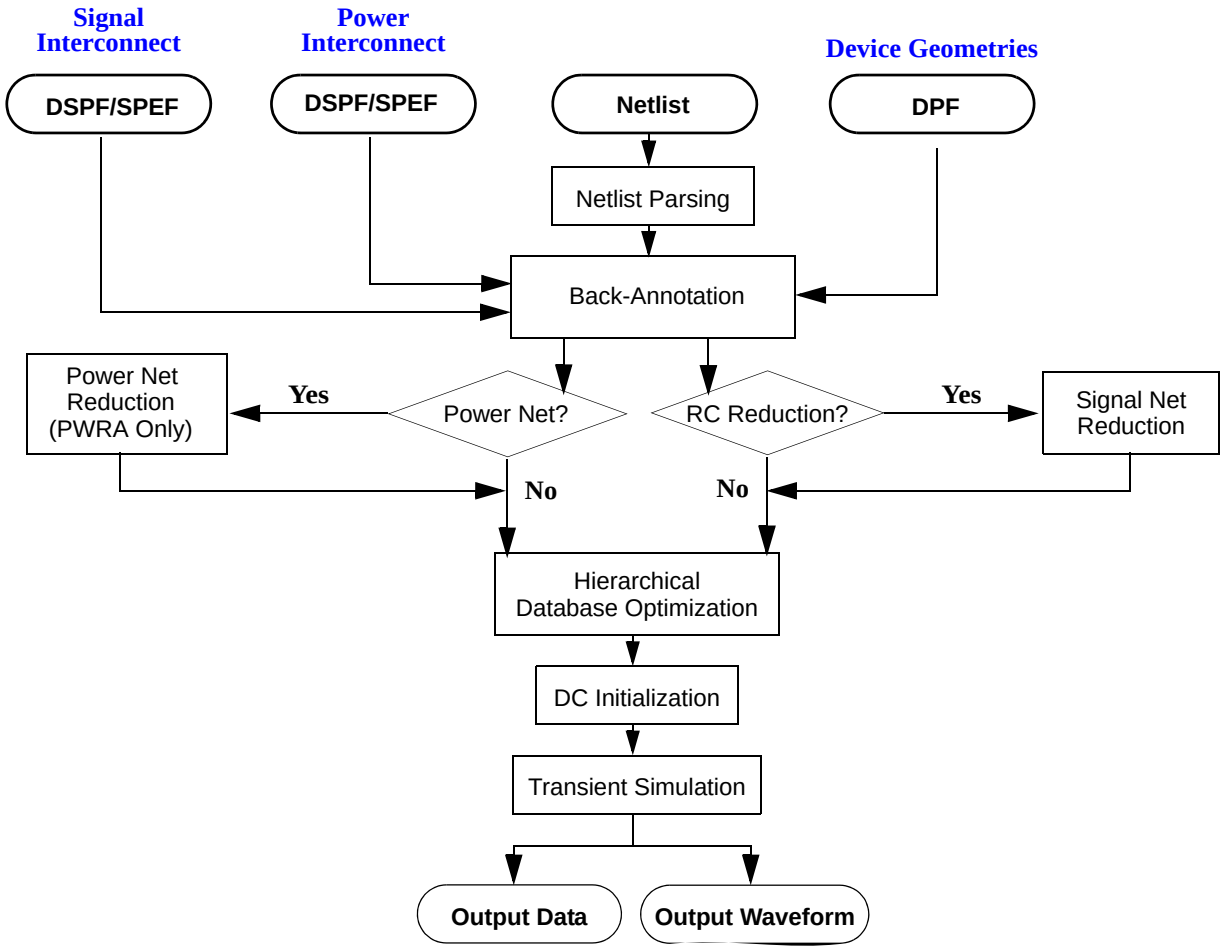
DC initialization begins after the hierarchical simulation database is constructed. This initialization process is completed when steady state convergence is achieved or when the iteration count reaches the user-defined limit.

The transient simulation follows DC initialization, and it proceeds until the final simulation time is reached. The default settings for simulation control are provided by HSIM. Simulation precision can be modified using the simulation control settings. Refer to, Chapter 7, Simulation Parameters for details about simulation control.

Simulation results are available in either an ASCII or a binary format. The default binary output format is FSDB, which is readily viewable by the nWave waveform viewer.

An overview of the HSIM Simulation Flow is shown in Figure 2 on page 11.

Figure 2 HSIM Simulation Flow



Supported Output Formats

HSIM also supports two ASCII formats:

- Plain ASCII
- HSIM Output Format

Note:

For more information, see Chapter 10, Simulation Output, HSIM Output Formats on page 331 or contact Synopsys for information on other available output formats.

HSIM format provides a single output format that can be translated into other formats as required by various waveform display tools. The hs2tbl utility included with the software can convert this output format into tabular data that can be displayed by either of the following:

- Xgraph in an X windows environment
- Dplot in a Windows NT/Windows 2000 environment

HSIM^{plus} Quick Start

The HSIM^{plus} Quick Start provides a means to quickly and easily get a design running in HSIM while applying a set of parameter values for HSIM internal algorithms that are appropriate for the simulation being performed. This document does not detail the specifics of the algorithms. Training is available on request through a sales or application engineering representative. Figure 3 on page 14 will aid in applying the correct commands to achieve the performance/precision trade-off required.

HSIM Setup Commands

Use commands from the following sections to set up HSIM. (Detailed command information is presented in Chapter 4, HSIM Commands in Alphabetical Order.)

High-Sensitivity Analog Circuit Simulation Parameters

HSIMANALOG on page 48

DC Control Parameters

HSIMDCITER on page 64

HSIMENHANCEDC on page 75

HSIMSPEED on page 125

Precision Control Parameters

HSIMPARPRECISION on page 108

HSIMRCPRECISION on page 116

HSIMMOSPRECISION on page 98

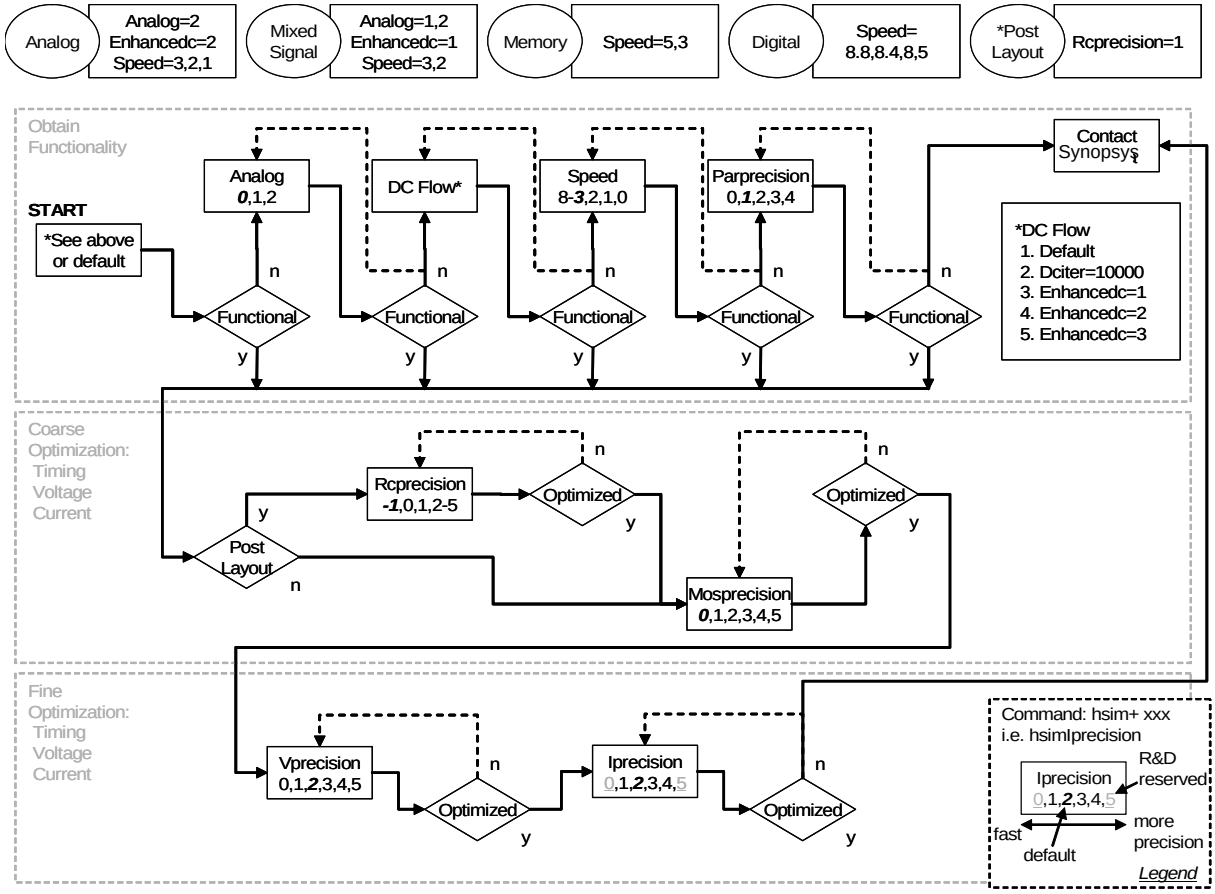
HSIMVPRECISION on page 169

HSIMIPRECISION on page 88

Setup Flow

Figure 3 on page 14 shows the process flow to achieve three distinct goals for setting up an HSIM simulation.

Figure 3 HSIM Setup Flow Chart



HSIM Simulation Setup Goals

The HSIM simulation setup goals are:

1. Functionality
2. Coarse optimization of timing, voltage, and current
3. Fine optimization of timing, voltage, and current

Successful Simulation Criteria

Accuracy is subjective, and can depend on your criteria.

To determine successful simulation criteria, perform the following steps.

1. Find the type of design that best describes the project from the following:

- Analog
- Mixed Signal
- Memory
- Digital

If the design also includes post-layout data, the options identified by the Post Layout indicator should be appended. The HSIM commands shown in the flow chart are based on engineering results and where more than one value is specified for a command, you should start with the *first* indicated for performance or the *last* indicated for precision, values in-between, if more than two are provided, are a trade off between performance and precision.

Note:

These commands are available during setup to both global and subcircuit-based implementations when the algorithm has localization applicability. If a design includes more than one type of design style, use .subckt to apply the appropriate commands to the respective design styled subcircuits. Refer to .subckt on page 299.

2. In most cases, the examples shown here provide good trade-offs between precision and performance. However, if this is not the case in a simulation, you should follow the flow chart illustrated in Figure 3 on page 14 by adjusting the parameters indicated.

Note:

The command boxes in the flow chart in Figure 3 on page 14 show the options available to the commands. Command values indicate the default in *bold italic* and read from *left* to *right* increases precision. For ease of reading the chart, the HSIM prefix has been omitted; thus the HSIM prefix must be applied to all commands identified in the flow chart. For example, PARPRECISION must be inserted into the simulation deck as HSIMPARPRECISION.

Example of Command Box Interpretation The MOSPRECISION default is 0; 0 should provide the highest performance and the available values to set are 0,1, 2, 3, 4, and 5, with 5 providing the highest precision.

Example of Customer Flow for a Mixed Signal Design with NO Post-Layout Insert the following parameters into the simulation deck to obtain functionally correct results.

HSIMANALOG=2

`HSIMANALOG` provides Mixed Signal recommendations modified by the Analog functional decision box to obtain a desired functionality.

HSIMENHANCEDC=1

`HSIMENHANCEDC` provides Mixed Signal recommendations modified by the DC Flow functional decision box to obtain a desired functionality.

HSIMSPEED=3

`HSIMSPEED` provides Mixed Signal recommendations.

Using HSIM in a Nanometer VLSI Design Flow

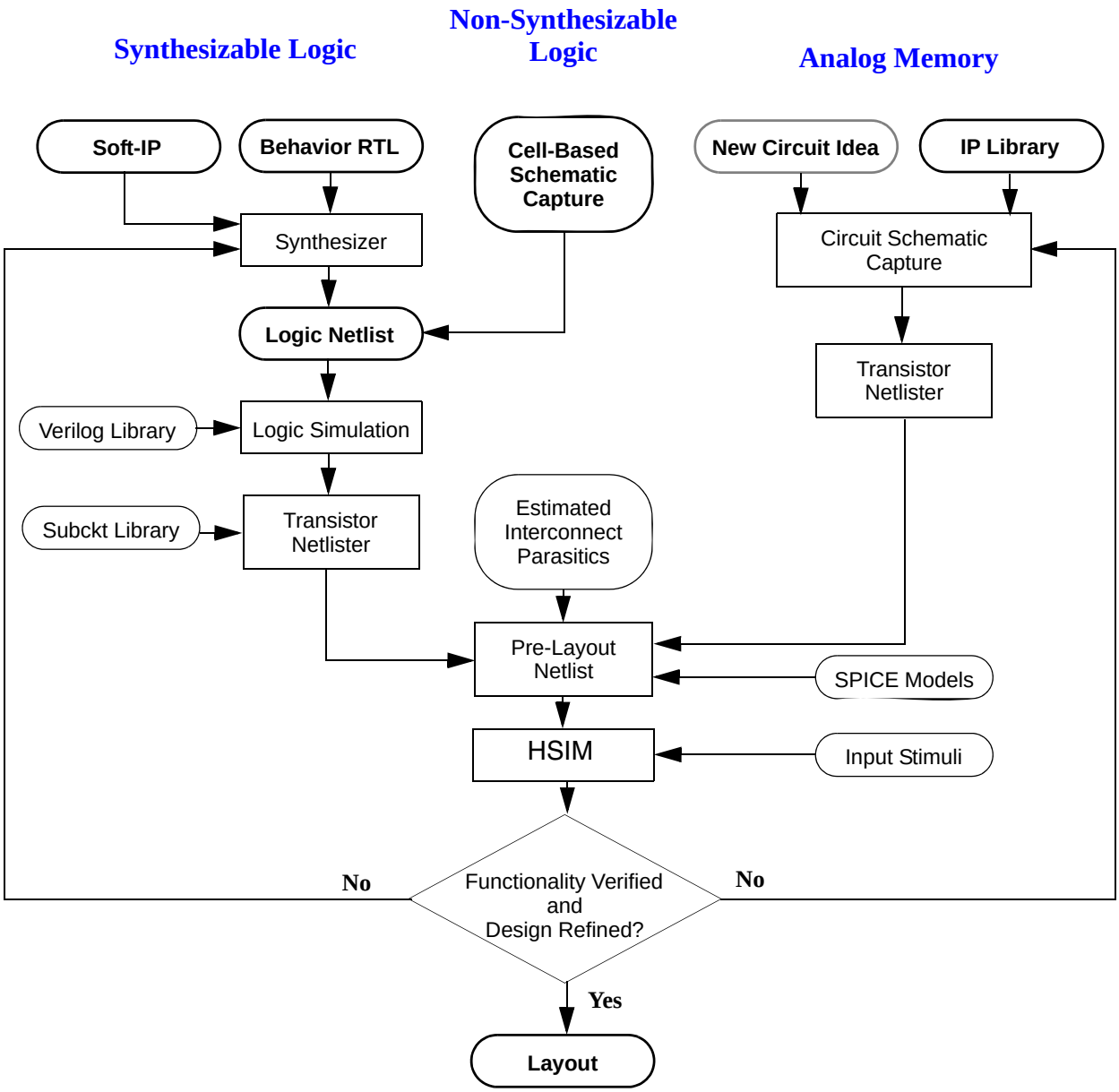
A typical nanometer very large system integration (VLSI) design effort can be divided into pre-layout and post-layout design flows. These design flows are discussed in the following sections.

Pre-Layout Design Flow

Pre-layout design flow is designed to create functionally correct designs for layout implementation. Estimated parasitics for the inter-block interconnects are often used for exploring the timing behavior at this early design stage.

At the block level illustrated in Figure 4 on page 17, HSIM shows the use of three design types in a pre-layout flow. A full-chip functionality and timing simulation flow is an extension from the block-level flow. By assembling the netlists of all the individually verified blocks and providing the top-level input stimuli, the full-chip can be simulated without difficulties or complications.

Figure 4 Pre-Layout Design Flow



HSIM provides design flows for the following three design styles:

- Synthesizable Logic on page 18
- Non-Synthesizable Logic on page 18
- Analog/Memory on page 18

Synthesizable Logic

Synthesizable logic uses high-speed standard cells for digital design. Simulations can be run to explore cross-talk issues along certain critical paths. Then, the information from this step is used to refine layout strategies that prevent cross-talk problems. Block current obtained through the simulation can be used for power bus sizing.

To maximize HSIM capabilities in the synthesizable logic flow, the subcircuit library and the estimated interconnect parasitics must be provided. The estimated parasitics are placed inside the circuit netlist with the other functional elements of the design, and then simulated by HSIM.

The subcircuit library transforms the design from a gate-level representation to a transistor-level representation. The estimated parasitics inject an early physical dimension into a pure logical design in order to study the potential physical effects within the design. Estimated parasitics may include any of the following:

- Long interconnects for a timing behavior assessment, particularly for clock net analysis (jitter and skew).
- Estimated cross-coupling capacitors for a cross-talk assessment
- All possible wire capacitors for a block current assessment

Note:

Estimated parasitics are not required for the functional verification of a given circuit.

Non-Synthesizable Logic

Non-synthesizable logic is often used for high performance application-specific integrated circuit (ASIC) and semi-custom designs. These types of designs typically utilize complex design techniques such as dynamic logic, and are subject to the circuit effects of noise and ground bounce. As logic simulators do not provide an accurate description of the behavior of these circuits, HSIM simulation is strongly recommended.

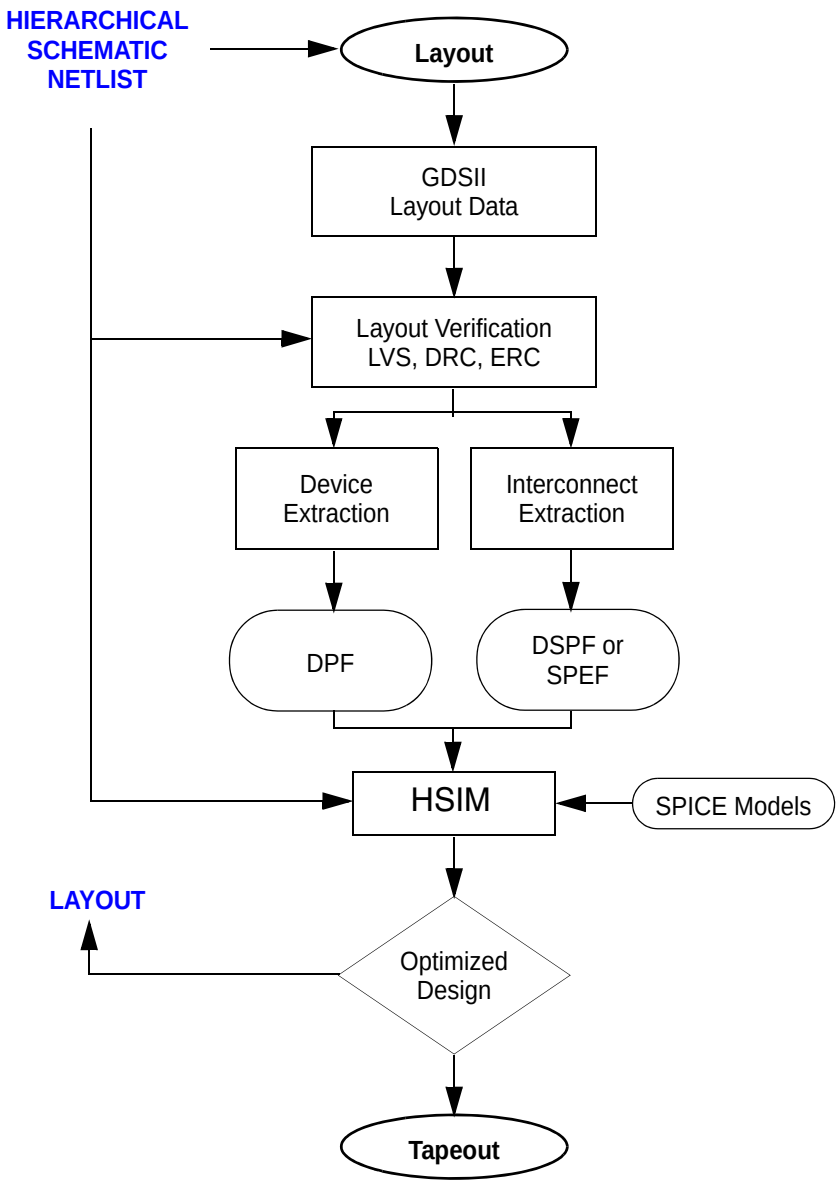
Analog/Memory

For analog/memory design styles, HSIM provides speed and accuracy to successfully simulate these circuit types.

Post-Layout Design Flow

The main purpose of the post-layout design flow is to optimize the design by layout refinements and to verify circuit performance in the presence of layout parasitics. In particular, to determine the influence of IR drop and coupling capacitance on design characteristics. An overview of the post-layout design flow is shown in Figure 5 on page 20.

Figure 5 Post-Layout Design Flow



Note:

This flow is the recommend post-layout flow.

HSIM also supports simulating an ideal netlist and other post-layout flows.

GDSII Layout Database

The post-layout flow begins with a GDSII layout database. The next stage is physical layout verification to ensure there are no major flaws in the layout, such as:

- DRC
- ERC
- LVS

Layout Functional Verification

After physical layout verification, the basic functionality of the layout is verified. This is best accomplished by doing the following:

- Extracting the active components of the design.
- Back-annotating DPF to the pre-layout hierarchical netlist

Once this is completed, a top-level full-chip simulation can be performed. Top-level stimuli from the pre-layout flow can be used to ensure that the basic functionality of the extracted netlist has not changed.

Design Optimization

Once design functionality has been confirmed, layout parasitics can be incorporated into the simulator and the results of the simulation can be used to optimize the design. Optimization goals include:

- Enhancing circuit speed
- Maintaining power consumption within specification
- Increasing the design margin
- Improving circuit reliability and robustness
- Decreasing sensitivity to manufacturing variability

The circuit characteristics to address in this phase include:

- Speed
- Power
- Reliability
- Manufacturability

Circuit Extraction and Analysis

Enhanced design margins provides greater tolerance against manufacturing fluctuations. Greater circuit reliability ensures a long product life in the field. To refine a layout, a detailed circuit must be extracted and analyzed to determine if and where the circuit needs improvement.

A detailed extracted circuit netlist consists of the following:

- Active circuit devices
- The device's topological connectivity
- Interconnect parasitics for both signal and power nets

Many extraction tools are configured to extract both active devices and interconnect parasitics into a single, flat netlist file. This straightforward approach eliminates any potential problems with issues such as name matching between netlist and DSPF/SPEF files. The one proviso is that HSIM must have sufficient memory available to read-in the entire flat netlist.

After the design netlist is parsed, HSIM partitions the design, looking for hierarchy with the available partitioned subcircuits. This ensures that HSIM utilizes the hierarchy in the design for memory efficiency even if the input netlist is flat. Since HSIM peak memory usage occurs while parsing the netlist and building the hierarchical simulation structures, if the circuit netlist can be parsed, simulation can usually proceed. This process works well for circuits of small to medium size but full advantage of the hierarchical simulation engine cannot be realized in the same way that it can with a hierarchical circuit of many levels, containing a large number of related small isomorphic instances.

Most extraction tools also offer separate netlisting for active devices and interconnect parasitics after extraction is complete. HSIM is designed and implemented to take maximum advantage of this flow and data. Extracting the active devices to the Device Parameter Format (DPF) and interconnect parasitic to DSPF/SPEF provides for efficient back-annotation into the hierarchical simulation database created after parsing the hierarchical pre-layout netlist.

Interconnect Segmentation Resolution

Another issue is the resolution of the interconnect segmentation. When a fine interconnect segmentation resolution is requested, a flat extractor can create a very large number of extracted parasitic elements. Although HSIM uses efficient hierarchical data structures, it is not completely immune from the large data volume problems associated with parsing in these large flat netlists. The

amount of extracted data must be controlled by selecting the appropriate segmentation resolution in the parasitic extraction process.

Note:

For a large parasitic database, it is recommended to enable RC reduction features in HSIM.

This substantially reduces the memory required to store the interconnect parasitics and ensures that the transient simulation proceeds with maximum efficiency.

A memory circuit contains the same core cell that composes the majority of the layout. If extracted hierarchically, the netlist size can be reduced to one-fifth the size of the same netlist from a flat extraction. System-on-Chips (SoC) also have memory and other regular structures, although the storage saving might not be as great as that of memory circuits.

If the extracted interconnect parasitics can be generated in DSPF/SPEF, then the DSPF back-annotation capability allows a net-by-net RC reduction and instantiation. This feature removes the need to store all layout parasitics because each net is reduced and back-annotated one net at a time. This approach effectively eliminates the typical bottleneck associated with handling large flat parasitic netlists and provides greater circuit simulation capacity.

HSIM share signal net reduction technologies and the parameter settings operate identically in each simulator. For signal nets, HSIM^{plus} offers hierarchical back-annotation capability where the reduced signal net can be partitioned and distributed into the pre-layout netlist; preserving the benefits of hierarchical simulation as much as possible. Compared to HSIM flattening back-annotation, hierarchical back-annotation offers the following benefits for signal nets:

- Small, but not insignificant, throughput improvement.
- Large increase in memory efficiency.

HSIM^{plus} power net capabilities offer a significant enhancement over HSIM. HSIM offers small power net reduction technologies such as RMIN elimination. However HSIM flattening back-annotation of the globally coupled power nets defeats the hierarchical simulation engine and forces flat simulation.

HSIM^{plus} adds two key pieces of innovative technology that enable efficient full-chip post-layout simulation including all power nets:

- Power Net reduction
- Hierarchical power net back-annotation

Power Net reduction has two sub-flows that are applied in sequence:

a. Straight-forward timing-based sub-flow largely preserves the power net topology and applies resistor reduction based on a specified threshold, series merging, and parallel elimination to reduce the number of power net resistors by an order of magnitude.

b. The second sub-flow alters the power net topology and applies user controlled heuristics of varying aggression levels that transform the power net structures into significantly smaller networks.

These reduced power nets can be saved at intermediate steps in the reduction process facilitating quick turnaround times for repeated simulations. The final reduced power networks are then back-annotated to the hierarchical pre-layout database using HSIM^{plus} hierarchical back-annotation capability. Circuits that most lend themselves to a hierarchical simulator solution typically have, but are not exclusively limited to, small isomorphic instances that are greatly replicated, post-layout, with all signal and power net interconnect parasitics incorporated into the simulator.

HSIM can precisely verify circuits of tens or even hundreds of millions of transistors at tape out providing the following benefits:

- Increased confidence in achieving first time simulation success.
- Increased probability of ramping the first design to high volume.
- High and consistent yields.

Design Optimization

Design optimization is accomplished through an iterative layout improvement process based on HSIM simulation and analysis results generated in each iteration. Obtaining a highly optimized design may require several iterations; longer backend verification and optimization schedules may need to be allocated. However, additional analysis time can result in fewer silicon re-spins, more competitive products, and higher product quality.

3

HSIM Command Groups

The following tables list the HSIM commands by their functional categories.

Global Macro Set-Up Commands

For details about these commands, see HSIM Setup Flow Chart on page 14.

Global Macro Set-up Command	Default Setting
HSIMIPRECISION on page 88	2
HSIMMOSPRECISION on page 98	0
HSIMPARPRECISION on page 108	1
HSIMRCPRECISION on page 116	-1
HSIMVPRECISION on page 169	2

Netlist Commands

Netlist Command	Default Setting
HSIMNTLFMT on page 102	hspice
HSIMOPTSEARCHEXT on page 105	
HSIMSUBBUSDELIM on page 148	
HSIMTOP on page 152	null

Netlist Command	Default Setting
HSIMWRAPPERSUB on page 173	

Digital Vector File Commands

Vector File Command	Default Setting
HSIMVCD2VEC on page 164	
HSIMVECTORFILE on page 166	
HSIMVHI, HSIMVHL on page 167	
HSIMVLO on page 168	

Ungrounded/Grounded Capacitor Commands

For information about how the HSIMFCAP, HSIMFCAPR, HSIMFCM, HSIMGCAP, and HSIMGCAPR commands work together, see Control Commands for Floating Capacitors on page 309.

Ugrounded/Grounded Capacitor Commands	Default Setting
HSIMFCAP	`1E-13`
HSIMFCAPR	`0.5`
HSIMFCM	`3`
HSIMGCAP	`1E-15`
HSIMGCAPR	`0.02`
HSIMKEEPALLGNDCAPS on page 89	
HSIMLUMPCAP on page 91	`1`

Isomorphic Matching Commands

Isomorphic Matching Command	Default Setting
HSIMPORTCR on page 109	`0.01` or 1%
HSIMPORTI on page 109	`0.001` pV
HSIMPORTV on page 110	`1` mV

DC Initialization Commands

DC Command	Default Setting
HSIMANALOG on page 48	`1`
HSIMANLASCIPRSN on page 50	
HSIMDCI on page 63	`5E-7` Amperes
HSIMDCINIT on page 63	`1`
HSIMDCITER on page 64	`250`
HSIMDCSTEP on page 66	`1000` picoseconds
HSIMDCSTOPAT on page 66	
HSIMDCV on page 66	`0.001` Volts
HSIMENHANCEDC on page 75	`0`
HSIMIDDQ on page 86	
HSIMKEEPNODESET on page 89	`0`
HSIMMULTIDC on page 99	`1`
HSIMNODCNODES on page 100	`0`
HSIMSTOPIFNODC on page 146	`0`

DC Command	Default Setting
HSIMWARNIFNODC on page 172	0

Peformance/Accuracy Control Commands

Note:

For information about how the HSIMFCAP, HSIMFCAPR, HSIMFCM, HSIMGCAP, and HSIMGCAPR commands work together, see Control Commands for Floating Capacitors on page 309.

Performance/Accuracy Control Command	Default Setting
HSIMALLOWEDDV on page 47	0.3 Volts
HSIMANALOG on page 48	1
HSIMASIM on page 50	0
HSIMAUTOVDD on page 51	0
HSIMAUTOVDDSUP on page 51	0
HSIMENHANCEDIDDQ on page 76	0
HSIMFCAP	1E-13
HSIMFCAPR	0.5
HSIMFCM	3
HSIMFLAT on page 77	0
HSIMGCAP	1E-15
HSIMGCAPR	0.02
HSIMGCSC on page 79	1
HSIMIDDQ on page 86	
HSIMIPRECISION on page 88	

Performance/Accuracy Control Command	Default Setting
HSIMKEEPALLGNDCAPS on page 89	
HSIMLUMPCAP on page 91	`1`
HSIMMQS on page 99	`1`
HSIMPORTCR on page 109	`0.01` or 1%
HSIMPORTI on page 109	`0.001` pA
HSIMPORTV on page 110	`0.001`
HSIMPREFERVERILOGA on page 112	`0`
HSIMSAMPLERATE on page 123	`160`
HSIMSNCS on page 124	`1`
HSIMSPEED on page 125	`3`
HSIMSPICE on page 143	`0`
HSIMSTEADYCURRENT on page 144	`10` nA
HSIMSTEP2TAUMAX on page 145	`0`
HSIMTAUMAX on page 149	`2` ns
HSIMTIMESCALE on page 151	`1` picosecond
HSIMTLINEDV on page 152	`1`
HSIMTRAPEZOIDAL on page 153	`0`
HSIMVDD on page 165	`3.0` Volts
HSIMVSRCDV on page 170	
HSIMVSRCSYNC on page 171	`0`

Device Model Commands

Device Model Command	Default Setting
HSIMABMOS on page 41	0
HSIMACCURATERDS on page 41	0
HSIMAUTOVDD on page 51	
HSIMB4REMOVE on page 52	0
HSIMBSIM3ISUB on page 57	
HSIMBJTCC on page 56	0
HSIMBJTCV on page 57	0
HSIMBMOS on page 57	0
HSIMCAPCC on page 58	0
HSIMCC on page 60	0
HSIMCHECKMOSBULK on page 61	0
HSIMDCSOIACC on page 66	0
HSIMDETAILBSIM4 on page 68	0
HSIMDIOCC on page 68	0
HSIMDIOCV on page 68	0
HSIMDIODECURRENT on page 68	0
HSIMDIODEVT on page 69	0.5
HSIMEFFICIENTTBL on page 74	0
HSIMENHANCECAP on page 74	0
HSIMENHANCEDIDDQ on page 76	0
HSIMENHANCEMOSIV on page 77	0

Device Model Command	Default Setting
HSIMFORWARDBIASDIODE on page 78	0
HSIMGATEMOD on page 79	0
HSIMIDDQ on page 86	
HSIMIGISUB on page 86	0
HSIMMODELSTAT on page 94	0
HSIMUSEHM on page 154	
HSIMUSEHMINST on page 156	
HSIMUSERLIB on page 158	
HSIMNVBS on page 103	200
HSIMNVDS on page 103	60
HSIMNVGS on page 104	60
HSIMVBS3END on page 163	
HSIMVBS3START on page 164	
HSIMVDD on page 165	3.0 Volts
HSIMVDSEND on page 165	HSIMVDD+1

Connectivity Checking Commands

Connectivity Checking Command	Default Setting
HSIMBUSDELIMITER on page 58	
HSIMCONNCHECK on page 62	1
HSIMSUBBUSDELIM on page 148	

Activity Checking Commands

Activity Checking Command	Default Setting
HSIMACHECKFANOUT on page 42	0
HSIMACHECKRFEDGE on page 44	0
HSIMINACTOPT on page 87	2

High-Impedance Node Checking Commands

High-Impedance Node Checking Command	Default Setting
HSIMHZ on page 82	0
HSIMHZALL on page 83	0
HSIMHZFANOUT on page 83	0
HSIMHZNDNAME on page 84	
HSIMHZSTART on page 84	
HSIMHZSTOP on page 85	
HSIMHZTIME on page 85	5.0
HSIMHZXSUBCKT on page 85	

Signal Net Post-Layout Commands

Signal Net Post-Layout Command	Default Setting
HSIMCMIN on page 61	
HSIMDELAYNTLRMIN on page 67	0
HSIMLMIN on page 90	1E-11

Signal Net Post-Layout Command	Default Setting
HSIMNTLRMIN on page 103	Current value of `HSIMRMIN`.
HSIMPOSTL on page 110	`0`
HSIMPOSTLMANY2M on page 112	`0`
HSIMPOSTLONE2M on page 112	`0`
HSIMRCNPO on page 115	`0`
HSIMRCRIO on page 117	
HSIMRCRKEEPELEM on page 118	
HSIMRCRKEEPMODEL on page 118	
HSIMRCRKEEPNODE on page 119	
HSIMRCRTAU on page 119	`1.0e-10` seconds
HSIMRMIN on page 122	`0.1`
HSIMVSRCLMIN on page 170	`1.0e-10`
HSIMVSRCRMIN on page 171	`0.1`

DPF/SPEF Back-Annotation Commands

DPF/SPEF Back-Annotation Commands	Default Setting
HSIMCAPFILE on page 59	
HSIMDPF on page 69	
HSIMDPFHIERID on page 70	Period character (".")
HSIMDPFMERGEFDEV on page 70	`1`
HSIMDPFMULTISUB on page 70	`0`
HSIMDPFPFX on page 71	`m_`

DPF/SPEF Back-Annotation Commands	Default Setting
HSIMDPFPREPORTNOBA on page 71	0
HSIMDPFSCALE on page 72	1
HSIMDPFSFX on page 72	Asterisk character ("@")
HSIMDPFSPLITFD on page 73	0
HSIMDTNAME on page 73	
HSIMSPFEFTRIPLET on page 142	2
HSIMSPF, HSIMSPEF on page 129	
HSIMSPFADDNETPINXY on page 129	
HSIMSPFCC on page 130	0
HSIMSPFCCSCALE on page 130	1
HSIMSPFCHLEVEL on page 131	
HSIMSPFCMIN on page 131	
HSIMSPFCNET on page 131	
HSIMSPFDSJNET on page 132	0
HSIMSPFFDELIM on page 132	0
HSIMSPFFEEDTHRU on page 133	0
HSIMSPFHLEVEL on page 134	
HSIMSPFKEEPNODECAP on page 134	
HSIMSPFKS on page 135	0
HSIMSPFMERGEFDEV on page 135	1
HSIMSPFMSGLEVEL on page 135	1
HSIMSPFMULTISUB on page 136	0

DPF/SPEF Back-Annotation Commands	Default Setting
HSIMSPFNETPINDEL on page 136	
HSIMSPFNETWARNFILTER on page 137	
HSIMSPFNOWARNONCAP on page 137	0
HSIMSPFPOS on page 137	
HSIMSPFPRINTPIN on page 138	0
HSIMSPFRCNET on page 139	
HSIMSPFREPORTNOBA on page 140	0
HSIMSPFSKIPNET on page 140	
HSIMSPFSKIPPWNET on page 141	0
HSIMSPFSKIPSIGNET on page 141	0
HSIMSPFSPLITCC on page 141	0
HSIMSPFWARNFILE on page 142	0
HSIMSTNAME on page 146	
HSIMSTSWAP on page 147	

Selected Net Back-Annotation Command

Selected Net Back-Annotation Command	Default Setting
HSIMACHECKFILENAME on page 43	
HSIMACHECKOUTFMT on page 43	

Output Control Commands

Output Control Command	Default Setting
HSIMCOILIB on page 61	
HSIMMEASOUT on page 92	`0`
HSIMOUTPUT on page 105	`fsdb` format.
HSIMOUTPUTFLUSH on page 105	`-1`
HSIMOUTPUTFSDBSIZE on page 106	`-1`
HSIMOUTPUTIRES on page 106	`1E-9`, or `1` nA
HSIMOUTPUTMEAS on page 106	`1`
HSIMOUTPUTTBL on page 107	
HSIMOUTPUTTRES on page 107	`1E-12` or `1` ps
HSIMOUTPUTTSTEP on page 107	Not set.
HSIMOUTPUTVRES on page 108	`1E-3`, or `1` mV
HSIMOUTPUTWDFSIZE on page 108	`-1`
HSIMTIMEFORMAT on page 150	`1`
HSIMUTFCOMPRESSLEVEL on page 159	`3`

AC and DC Analysis Commands

AC Analysis Command	Default Setting
HSIMACOUT on page 45	`0`
HSIMACOUTFMT on page 47	`0`
HSIMACFREQSCALE on page 42	

AC Analysis Command	Default Setting
HSIMANLASCIPRSN on page 50	
HSIMDCI on page 63	`5E-7` A
HSIMDCINIT on page 63	`1`
HSIMDCITER on page 64	`250`
HSIMDCOUTFMT on page 65	`0`
HSIMDCSOIACC on page 66	`0`
HSIMDCSTEP on page 66	`1000` picoseconds
HSIMDCSTOPAT on page 66	
HSIMDCV on page 66	`0.001` Volts
HSIMUSEPREVIOUSDC on page 158	`0`

Monte Carlo Analysis Commands

Monte Carlo Analysis Command	Default Setting
HSIMMONTECARLO on page 95	`1`
HSIMMONTECARLOINST on page 95	`1`
HSIMMONTECARLOSAVEOUT on page 96	`1`

Verilog-A Commands

Verilog-A Command	Default Setting
HSIMPREFERVERILOGA on page 112	
HSIMUSEVA on page 158	

Verilog-A Command	Default Setting
HSIMUSEVATABLE on page 159	0
HSIMVABRANCHPART on page 161	
HSIMVACROSSTTOL on page 162	100 picseconds
HSIMVACROSSVTOL on page 162	0.1 Volt
HSIMVAPARTITION on page 162	0
HSIMVAPRINTVAR on page 163	0
HSIMVATABLERANGE on page 163	2*Vdd
HSIMVATABLESIZE on page 163	100
HSIMVERILOGA on page 166	

Miscellaneous Commands

Miscellaneous Command	Default Setting
HSIMACHECKFANOUT on page 42	0
HSIMACHECKRFEDGE on page 44	
HSIMALLOWEDDV on page 47	0.3 Volts
HSIMDELVTO on page 67	
HSIMHIERID on page 81	Period character (“.”)
HSIMHSPICEVEC on page 82	0
HSIMLIS on page 90	0
HSIMLOGDCPROGRATE on page 90	0.1
HSIMLOGPROGRATE on page 91	0.1
HSIMINACTOPT on page 87	2

Miscellaneous Command	Default Setting
HSIMNODECAP on page 100	
HSIMNOSIMTIME on page 101	0
HSIMOPCOMPRESS on page 104	0
HSIMPRINTSIMSTATUS on page 114	1
HSIMREDEFSUB on page 120	
HSIMSCALE on page 124	1E-6
HSIMSPISCIUNITERR on page 144	
HSIMTIMESCALE on page 151	1 picosecond
HSIMTRAPEZOIDAL on page 153	0
HSIMVCD2VEC on page 164	
HSIMVECTORFILE on page 166	
HSIMVHI, HSIMVHL on page 167	
HSIMVLO on page 168	
HSIMWARNSTOP on page 172	0

4

HSIM Commands in Alphabetical Order

This chapter lists the HSIM commands in alphabetical order.

HSIMABMOS

Sets the Precise Model on the coupling effect between MOSFET drain, source, gate, and bulk terminals.

Syntax

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

Arguments

Argument	Description
`0`	Turns `HSIMABMOS` off (the default).
`1`	Turns `HSIMABMOS` on and is recommended for flash memory circuits.

Description

Simulation slows down significantly when `HSIMABMOS` is set to 1. You should only use it locally.

HSIMACCURATERDS

Controls accuracy levels for the NRS or NRD instance parameters used in BSIM4 models.

Syntax

```
.param HSIMACCURATERDS = <0|1|2>
```

Arguments

Argument	Description
`0`	HSIM applies a simple scaling method to calculate drain current to achieve good accuracy and a reasonable run time. This is the default setting.
`1`	Applies a more accurate scaling method to calculate drain current.
`2`	Applies a SPICE-like method to calculate current and capacitance.

Description

When you set `HSIMACCURATERDS =1`, there is a penalty in simulation time compared to the default setting of 0. When you set `HSIMACCURATERDS =2`, HSIM applies a SPICE-like method to calculate current and capacitance. Because this value has a long run time penalty, it is recommended only when you need the highest possible accuracy.

HSIMACFREQSCALE

Sets the frequency units for AC analysis results.

Syntax

`.param HSIMACFREQSCALE=`*`freq_units`*

Description

`HSIMACFREQSCALE` sets the frequency units when HSIM outputs the results of AC analysis. In the following example, frequency units are set to MHz.

Example

```
.param HSIMACFREQSCALE=1e6
```

HSIMACHECKFANOUT

Controls node checking.

Syntax

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

Arguments

Argument	Description
`0`	Checks all of the nodes. This is the default setting.
`1`	Checks only nodes that are connected to MOSFET gates.

Description

`HSIMACHECKFANOUT` checks all nodes or only the nodes that connect to MOSFET gates.This command is useful when you only want to check driver nodes.

Example

```
.param HSIMACHECKFANOUT=1
.acheck '*' dv=0.5 exclude='in,1'
```

Check all driver nodes except the in and 1 that have voltage changes greater than 0.5 volts during the simulation.

HSIMACHECKFILENAME

Writes active nets to the specified file.

Syntax

```
.param HSIMACHECKFILENAME=<filename>
```

Description

This command writes all of the active nets to the specified file name. It is optional.

If `HSIMACHECKFILENAME` is not specified, and `HSIMACHECKOUTFMT` is set to 1, HSIM writes the active nets to the <prefix>.rcxt file. If `HSIMACHECKFILENAME` is not specified, and `HSIMACHECKOUTFMT` is set to 2, HSIM writes the active nets to the `<prefix>.hsimba` file. In both cases, <prefix> is the -o parameter specified in the HSIM command line.

HSIMACHECKOUTFMT

Specifies the format to report active nets.

Syntax

```
.param HSIMACHECKOUTFMT=<1|2>
```

Arguments

Argument	Description
1	Reports active nets in STAR-RCXT (*.rcxt) format for the Selective Net Extraction Flow.
2	Reports active nets in HSIM Back-Annotation (*.hsimba) format for the Selective Net Back-Annotation Flow.

Description

This command is specifies the format for the active nets. It is mandatory during the Initial Activity Run. It is not included in the Post-Layout Simulation Run.

HSIMACHECKRFEDGE

Checks if signals have rising or falling edge activity for the nodes that belong to the node pattern.

Syntax

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

Arguments

Argument	Description
0	No edge checking (default).
1	Checks all nodes defined by node pattern.

Description

The rising threshold is set as 0.75*(global voltage), and the falling threshold is set as 0.25*(global voltage). If a signal starts off low and rises above the rising threshold, a rising-edge activity is reported in the file_name.rfedge file. Conversely, if a signal starts off high and falls below the falling threshold, a falling-edge activity is reported in the same file.

Example

```
.param HSIMACHECKRFEDGE=1
.acheck V(x3m.x0.x1.xsap.x1.*)
```

The output file format of the `HSIMACHECKRFEDGE` parameter is:

hierarchy_node node_voltage activity_time(ns) activity_type

The following is an output file example. Table 1shows the output file format.

```
Output file: hsim.rfedge
*HSIM Win32 Debug Version 6.0 - 155206302004
*Tracking No - HSIM 2005.26.7
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Rising(R) and Falling(F) edge detection
```

Table 1 HSIMACHECKRFEDGE Output File Format

hierarchy_node	node_voltage	activity_time (ns)	activity_type
x3m.x0.x1.xsap.x1.pc	0.77	4.90	F
x3m.x0.x1.xsap.x1.x2.gdx	2.49	30.40	R
x3m.x0.x1.xsap.x1.da1	2.55	30.74	R
x3m.x0.x1.xsap.x1.dan	0.71	31.09	F
x3m.x0.x1.xsap.x1.x2.ctrb	0.63	120.86	F
x3m.x0.x1.xsap.x1.s1	2.53	126.20	R
x3m.x0.x1.xsap.x1.s2	2.53	147.12	R
x3m.x0.x1.xsap.x1.out	0.64	147.12	F

HSIMACOUT

Controls the definitions used for magnitude (m), phase (p) and decibel (db) modifiers for AC analysis.

Syntax

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

Arguments

Argument	Description
`0`	Applies the SPICE definition.
`1`	Applies the default definition.

Description

The following formulae apply to the SPICE definition:

`0`

```
vm (nd1,nd2) = abs(v(nd1)-v(nd2))
vp (nd1,nd2) = phase(v(nd1)-v(nd2))
vdb (nd1,nd2) = 20 log10(abs(v(nd1)-v(nd2)))
```

`1` (default)

```
vm (nd1,nd2) = vm(nd1) - vm(nd2)
vp (nd1,nd2) = vp(nd1) - vp(nd2)
vdb (nd1,nd2) = 20 log10(vm(nd1)/vm (nd2))
```

Note:

abs and phase are the magnitude and the phase of the complex phasor.

The following formulae apply to the default definition:

```
vm (nd1,nd2)=vm(nd1) - vm(nd2)
vp (nd1,nd2)=vp(nd1) - vp(nd2)
vdb (nd1,nd2)=20 log10(vm(nd1)/vm (nd2))
```

Example

```
.print ac vm(n1,n2) vp(n2,n3) vdb(n3,n4) vr(n4,n5) vi(n5,n1)
```

The following formulae are applied to this definition:

- `vm(nd1,nd2)=(abs(v(nd1)-v(nd2)))`
- `vp (nd1,nd2)=phase(v(nd1)-v(nd2))`
- `vdb (nd1,nd2)=20 log10(abs(v(nd1)-v(nd2)))`
- `vrvr (n4,n5)=real(v(n4)-v(n5))`
- `vivi (n5,n1)=imag(v(n5)-v(n1))`

HSIMACOUTFMT

Sets the AC analysis output format.

Syntax

```
.param HSIMACOUTFMT = <0|2|4|7>
```

Arguments

Argument	Description
0	Default ASCII table format.
2	ASCII raw file format.
4	Similar to HSIMACOUTFMT=2, but if there is an external sweep in AC analysis, the results for each value of the external sweep go to a separate external file.
7	Output formats of third party vendors.

Description

`HSIMACOUTFMT` can set the following file output file formats for AC analysis:

- 0 specifies that results are written to the <project>.ac file.
- 2 specifies that results are written to the <project>.ac.raw file.
- 4 specifies that if there is an external sweep in AC analysis, the results for each value of the external sweep go to a separate external file such as <project>.ac.raw.s1, <project>.ac.raw.s2, and so on.
- 7 is used for output formats of third party vendors. Use this option with `HSIMOUTPUT`. Combining `HSIMACOUTFMT` and HSIMOUTPUT on page 105 allows the AC result to be printed in third party vendor output formats.

HSIMALLOWEDDV

Dynamically adjusts the time step size so that each node voltage change over the time step is limited by the specified value.

Syntax

```
.param HSIMALLOWEDDV =voltage_value
```

Description

The default value for HSIMALLOWEDDV is 0.3V. The larger the value is, the faster the simulation speed. Better precision is achieved by using a smaller HSIMALLOWEDDV value. The default of 0.3V might give the optimal trade-off between precision and speed in most applications.

A reduced HSIMALLOWEDDV setting is recommended for small analog circuits and circuits with low power-supply voltage. HSIMALLOWEDDV is recommended to be set to 10% of the power-supply voltage when the power-supply voltage is below 2V. By specifying HSIMVDD on page 165, HSIMALLOWEDDV is reset to 10% of the HSIMVDD on page 165 value.

HSIMALLOWEDDV can be set in any local subcircuit, either subcircuit definition or subcircuit instance, for a local effect on the associated subcircuit. If a design has multiple power supplies, HSIMAUTOVDD on page 51 can be used to perform a path search and set HSIMALLOWEDDV as required.

HSIMAMOS

When set to 1, HSIMAMOS turns on the Precise Model for the coupling effect of MOSFETs in a subcircuit definition or a subcircuit instance.

Syntax

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

Description

The $AMOS setting is limited to an individual MOSFET, either at the top level or within a subcircuit definition. If $AMOS is set to 1 for a MOSFET in the subcircuit definition, then each corresponding MOSFET in the subcircuit instances have the same setting. The default is 0.

HSIMANALOG

Controls the complexity of analog simulation algorithm.

Syntax

```
.param HSIMANALOG = <-1|0|1|2|3>
```

Arguments

Argument	Description
`-1`	In simulating digital circuits where no analog circuit behavior is expected, set `HSIMANALOG=-1`. This setting assumes no sensitive coupling exists between neighboring subcircuits.
`0`	Might cause some feedback couplings across the hierarchy boundary to not be simulated accurately.
`1`	Extends precision to cover feedback couplings across the hierarchy boundary. Unless the memory usage is crucial, `HSIMANALOG=1` is recommended because it increases memory usage in order to flatten the subcircuits containing feedback couplings.
`2`	Further increases the complexity of the analog simulation algorithm by extending the feedback couplings to a wider scope, such as including voltage control oscillators (VCOs) found in most PLL circuits. `HSIMANALOG=2` is recommended for use with analog circuits containing highly sensitive topology such as the following: ▪ Bandgap reference voltage generator ▪ High-gain amplifier ▪ VCO ▪ Switched-capacitor filter ▪ PLL ▪ A/D and D/A converters
`3`	Automatically applies HHSIMSPICE on page 143=3 to those MOSFETs involved in feedback coupling. This setting is recommended for simulating A/D and D/A converters.

Description

To achieve optimal HSIM performance in analog and mixed-signal circuit simulation, `HSIMANALOG` controls the complexity of analog simulation algorithm. The higher the value you specify, the more time-consuming and precise the analog simulation algorithm is.

The `HSIMANALOG=0` and `HSIMANALOG=1` settings are used to simulate full-custom circuits or memory circuits. Compared with H`SIMANALOG=-1, t`hese

commands provide for increased complexity from more precise simulation in local feedback coupling between neighboring subcircuits. The difference between `HSIMANALOG=0` and `HSIMANALOG=1` is related to the circuit hierarchy.

The cross-coupled inverter pair in an SRAM cell is simulated more accurately compared with the `HSIMANALOG=-1` setting.

Setting `HSIMANALOG=2` or `HSIMANALOG=3` causes simulation speed to slowdown due to the more conservative analog simulation algorithms. There are two methods to speed up analog simulation.

1. Set `HSIMANALOG=2` or `HSIMANALOG=3` in selected subcircuits that contain sensitive analog subcircuits.
2. Set `HSIMANALOG=1.5`, which invokes the fast mixed-signal simulation mode. `HSIMANALOG=1.5` applies the same analog simulation algorithm as the setting `HSIMANALOG=2`.

   Note:

   The difference between `HSIMANALOG=1.5` and `HSIMANALOG=2` is that HSIMGCSC is automatically set to 0 for those MOSFETs not involved in feedback coupling. The speedup in setting `HSIMANALOG=1.5` is achieved by sacrificing the timing delay precision in the digital portion of the circuit.

HSIMANLASCIPRSN

Sets the precision as the number of printed digits in output ASCII files.

Syntax

```
.param HSIMANLASCIPRSN =value
```

Description

The minimum value is 2 and maximum value is 15. This command sets HSIM output of DC analysis to 12-digit representation. Any number larger than 15 is reset to 15, and any number smaller than 2 is reset to 2.

HSIMASIM

Enables a more conservative analog simulation algorithm.

Syntax

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

Description

Set `HSIMASIM=1` to enable a more conservative analog simulation algorithm for smaller analog circuits containing less than 50K elements. It is not recommended to apply this setting for large full-chip simulation, especially for large memory circuits. The default is 0.

HSIMAUTOVDD

Traces a design and automatically identifies multiple power supplies.

Syntax

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

Description

When simulating designs containing multiple power supply voltages, HSIM can trace the design and automatically identify multiple power supplies. `HSIMAUTOVDD` controls time-steps, logical print voltage thresholds, and MOS model table look-up ranges.

Set `HSIMAUTOVDD =1` for designs containing multiple power supplies. This setting performs a sophisticated path search to all feasible power supplies associated with every element and node. If a design contains only a single power supply, use `HSIMVDD` instead and set it to the value of this single voltage supply.

HSIMAUTOVDDSUP

Determines the logical print voltage thresholds.

Syntax

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

Description

In cases when a single node can be driven by multiple power supplies, use the `HSIMAUTOVDDSUP=1` command to determine the logical print voltage thresholds. Specify `HSIMAUTOVDDSUP = 0` to use the lowest supply voltage for the .lprint threshold calculation. This is the default.

In cases when logical print statements (.lprint) are used, and `vlth`/`vhth` or `HSIMVLTH`/`HSIMVHTH` are not specified, HSIM issues a warning message and uses the 30% and 70% of `HSIMVDD` as the lower and upper thresholds, respectively. If a design contains multiple power supplies, the setting `HSIMAUTOVDD=1` automatically sets `HSIMVLTH`/`HSIMVHTH` to 30%/70% of the multiple voltage supplies for associated nodes.

Note:

`HSIMAUTOVDDSUP` is only triggered if `HSIMAUTOVDD = 1`.

HSIMB4REMOVE

Use this command with BSIM4 models to remove the unused tables in the simulation for better memory utilization.

Syntax

```
.param HSIMB4REMOVE = <0|1>
```

Arguments

Argument	Description
`0`	Unused tables are left in the simulation.
`1`	Removes unused tables from the simulation.

Description

When `HSIMB4REMOVE=1` is set, the following usages stop simulations:

- HSIMSPICE=2 is specified at the top and the lower levels of the subcircuits use HSIMSPICE=0/1.
- HSIMSPICE=2 is specified at the top and the lower levels of the subcircuits use HSIMSPICE=0/1.

HSIMBISECTION

Supports concurrent bisection optimization on multiple pins without re-reading the netlist, greatly reducing the data I/O overhead.

Syntax

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

Description

Concurrent bisection optimization only applies to PWL voltage source elements. Additional parameters must be specified in the following statements:

- PWL voltage source statement (bisectparam). Add the following syntax to the PWL statement:

  ```
  bisectparam=1
  ```

- .param statement. Add the following syntax to the .param statement:

  ```
  .param HSIMBISECTION=1
  ```

- .tran statement

Example

```
.param HSIMBISECTION=1

Vin ck 0 pwl
+0 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven d 0 pwl
+'0' 0
+'delaytime' 0
+'t_time2+delaytime' 5
+ bisectparam=1
Vin2 ck2 0 pwl
+'0' 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven2 d2 0 pwl
+'0' 0
+'delaytime2' 0
+'t_time2+delaytime2' 5
+ bisectparam=1
.param delaytime=opt(0.0n, 0.0n, 20.0n)
.param delaytime2=opt2(0.0n, 0.0n, 20.0n)
.param t_time=alter1(0.1n, 0.1n, 0.1n, 0.5n, 0.5n, 0.5n, 1n, 1n,
1n)
.param t_time2=alter2(0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n, 0.1n, 0.5n,
1n)
.tran 0.1n 40.0n
+ sweep
+ optimize=opt
+ result=maxvout
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.tran 0.1n 40.0n
+ sweep
+ optimize=opt2
+ result=maxvout2
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.model optmod opt
+ method=bisection
+ relin=1.0e-3
.measure tran ck_slope param=t_time
.measure tran d_slope param=t_time2
```

```
.measure tran mt_delaytime param=delaytime
.measure tran maxvout max v(q) goal=5
.measure tran setuptime trig v(d) val=2.5 rise=1
+ targ v(ck) val=2.5 rise=1
.measure tran ck_slope2 param=t_time
.measure tran d_slope2 param=t_time2
.measure tran mt_delaytime2 param=delaytime2
.measure tran maxvout2 max v(q2) goal=5
.measure tran setuptime2 trig v(d2) val=2.5 rise=1
+ targ v(ck2) val=2.5 rise=1
```

In this example, HSIM extracts the setup time for two different flip-flops at the same time. The benefit of concurrent bisection optimization that is the total iteration time is similar to the iteration time required to extract one dff setup time, so this greatly reduces the total run time. Setup time can be extracted for a memory from data bus to clock without concurrent bisection by separately extracting the setup time for each data bus bit. The entire data bus-to-clock setup time can be extracted at the same time. The result of this example contains nine .mt files including the following:

```
2dff.mt.a0 :
Measurement results:
ck_slope=1.000000000000e-010
d_slope=1.000000000000e-010
mt_delaytime=1.876953000000e-008
maxvout=5.000000000000e+000   at=4.000000000000e-008
setuptime=2.300000000000e-010 targ=1.905000000000e-008
trig=1.882000000000e-008
ck_slope2=1.000000000000e-010
d_slope2=1.000000000000e-010
mt_delaytime2=1.873047000000e-008
maxvout2=5.000000000000e+000   at=4.000000000000e-008
setuptime2=2.700000000000e-010 targ=1.905000000000e-008
trig=1.878000000000e-008
2dff.mt.a1 :
Measurement results:
ck_slope=1.000000000000e-010
d_slope=5.000000000000e-010
mt_delaytime=1.849609000000e-008
maxvout=5.000000000000e+000   at=4.000000000000e-008
setuptime=3.040000000000e-010 targ=1.905000000000e-008
trig=1.874600000000e-008
ck_slope2=1.000000000000e-010
d_slope2=5.000000000000e-010
mt_delaytime2=1.848632000000e-008
maxvout2=5.000000000000e+000   at=4.000000000000e-008
setuptime2=3.140000000000e-010 targ=1.905000000000e-008
trig=1.873600000000e-008
...
2dff.mt.a8 :
Measurement results:
ck_slope=1.000000000000e-009
d_slope=1.000000000000e-009
mt_delaytime=1.873047000000e-008
maxvout=5.000000000000e+000   at=4.000000000000e-008
setuptime=2.700000000000e-010 targ=1.950000000000e-008
trig=1.923000000000e-008
ck_slope2=1.000000000000e-009
d_slope2=1.000000000000e-009
mt_delaytime2=1.865235000000e-008
maxvout2=5.000000000000e+000   at=4.000000000000e-008
setuptime2=3.480000000000e-010 targ=1.950000000000e-008
trig=1.915200000000e-008
```

HSIMBJTCC

Invokes the exact charge conservation model for BJT capacitors.

Syntax

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

Description

`HSIMBJTCC` invokes the exact charge conservation model for the charge stored by the BJT capacitors during the simulation. The default is 0.

HSIMBJTCV

Invokes the approximate charge conservation model for BJT capacitors.

Syntax

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

Description

`HSIMBJTCV` invokes the approximate charge conservation model for the charge stored by the BJT capacitors during the simulation. The default is 0.

HSIMBMOS

Enables the Precise Model for MOSFET terminals.

Syntax

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

Description

When set to 1, `HSIMBMOS` enable the Precise Model on coupling effect between MOSFET drain, source, and bulk terminals. The default value is 0. `HSIMBMOS=1` is recommended for flash memory circuits.

HSIMBSIM3ISUB

`HSIMBSIM3ISUB` is used to simulate accurate substrate current with the BSIM3 model. HSIM does not account for this current component without setting the parameter.

Syntax

```
.param HSIMBBSIM3ISUB =model_name
```

Description

The syntax accepts the model name or a wildcard. For multiple models, the parameter can be used multiple times. For example:

```
HSIMDBSIM3ISUB = nch*
HSIMDBSIM3ISUB = pch*
```

HSIMBUSDELIMITER

Specifies the bus notation used in a SPICE netlist.

Syntax

```
.param HSIMBUSDELIMITER =bus_notation
```

Description

`HSIMBUSDELIMITER` is a global parameter with a default of null. `HSIMBUSDELIMITER` specifies the bus notation used in SPICE netlist and is referenced in 2 locations:

- hdl testbench interface: Bus signal name in a testbench module is broken down into bit signal names with this bus notation so that HSIM can find the signal connection in the SPICE netlist.
- SPEF: If SPEF is specified, the parser converts all bus names in SPEF input so that they can be matched in HSIM database for back annotation.

Example

If a bus is named A<0:3>, then applying .param `HSIMBUSDELIMITER=<>` becomes:

```
A<0>, A<1>, A<2> & A<3>
```

HSIMCAPCC

Invokes the exact charge conservation model for the charge stored by a capacitor.

Syntax

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

Description

`HSIMCAPCC` invokeds the exact charge conservation model for the charge stored by a capacitor during the simulation. The default is 0.

HSIMCAPFILE

Controls adding capacitance to previously back-annotated nodes.

Syntax

`.param HSIMCAPFILE = node_name cap_value <otc|occ>`

Arguments

Argument	Description
node_name	Either a node name or a node pattern containing the asterisk (*) wildcard character.
cap_value	Capacitance value.
otc	Keyword to overwrite total capacitance.
occ	Keyword to overwrite constant capacitance.

Description

If otc or occ is not specified, the capacitance value *cap_value* is added to any capacitance previously back-annotated to the *node_name*.

Example

```
A1 10fF
B1 1.5f occ
x1.n* 0.6f otc
```

In this example:

- The first syntax statement above adds 10fF to node A1.
- The second syntax statement above overwrites the constant node capacitance of node B1 to 1.5fF.
- The third syntax statement above overwrites the total node capacitance of all nodes that match with x1.n* to 0.6fF.

HSIMCC

Enables the charge-conservation model for MOSFETs.

Syntax

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

Description

When set to 1, `HSIMCC`, enables the charge-conservation model for MOSFETs in the circuit. Similar to $AMOS, $CC is set to an individual MOSFET and enables the charge-conservation model for the MOSFET if $CC=1 is appended at the end of a MOSFET statement. The charge-conservation model is recommended for a charge-pump circuit or A/D and D/A converters.

Other than HHSIMAMOS on page 48 and HSIMCC on page 60, control parameters HSIMSPICE on page 143 and HSIMGCSC on page 79 also determine the MOSFET model complexity. All these model control parameters can be locally set such that the setting only affects those MOSFET elements within the subcircuit blocks specified with the setting including:

- HSIMABMOS on page 41
- HSIMAMOS on page 48
- HSIMBMOS on page 57
- HSIMCC on page 60
- HSIMGCSC on page 79
- HSIMSPICE on page 143

Device model control parameters are shown in Table 2

Table 2 Device Model Control Parameters

Parameter	Type	Default	Location(s)
$AMOS	Boolean	0	MOSFET Element
$CC	Boolean	0	MOSFET Element
$SPICE	Int	0	MOSFET Element

Example

```
M1 drain gate source bulk mos_model w=0.3u l=0.13u $AMOS=1
M2 dr    ga   sor    sub model2    w=0.6u l=0.18u $CC=1
```

HSIMCHECKMOSBULK

Performs a sanity check to identify potential forward bias conditions in MOSFET parasitic diodes.

Syntax

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

Description

HSIM reports warnings when the bulk of a NMOSFET transistor can potentially be greater than 0.5v or the bulk of a PMOSFET transistor can potentially be smaller than 0.5v. The bulk node of the MOSFET transistor has to be connected to a node to ground voltage source element. HSIM uses the DC voltage value of a DC voltage source or the maximum and minimum voltage of a variable voltage source to determine whether forward bias condition can occur.

HSIMCMIN

Sets the minimum capacitance value.

Syntax

`.param HSIMCMIN =`*`cap_value`*

Description

`HSIMCMIN` sets the minimum capacitor value allowed in simulation. All capacitors with values smaller than `HSIMCMIN` are ignored. This command is used to eliminate small capacitors. If the netlist file or DSPF/SPEF file contains negative capacitors, `HSIMCMIN` needs to be set to a negative value.

HSIMCOILIB

Specifies the full path of the waveform file generator library.

Syntax

`.param HSIMCOILIB=`*`path`*

Description

The full path should include the library name. `HSIMCOILIB` is optional. If there is no `HSIMCOILIB` specified, HSIM searches for the lib<out_format>.so in the following directories, in the order shown:

1. Run directory
2. $HOME directory
3. $HSIM_HOME/platform/<port>/bin
4. LD_LIBRARY_PATH on Solaris and on Linux, and SHLIB_PATH on HP

where <output_format> is the value of HSIMOUTPUT on page 105. All the letters in <output_format> have to be in upper case.

Example

To generate a MYFORMAT format, use the following two HSIM commands:

```
.param HSIMOUTPUT=MYFORMAT
.param HSIMCOILIB=/usr/local/lib/libMYFORMAT.so
```

HSIMCONNCHECK

Controls connectivity checking during a simulation.

Syntax

```
.param HSIMCCONNCHECK = <0|1|1.1|2>
```

Arguments

Argument	Description
`0`	When `HSIMCONNCHECK=0`, no connectivity check is performed.
`1`	When `HSIMCONNCHECK=1`, the default value, a connectivity check prints a warning message if connectivity errors are detected. In this setting, the simulation continues if there are errors.
`1.1`	When `HSIMCONNCHECK=1.1`, a more thorough check is performed and the simulation takes longer to finish.
`2`	When `HSIMCONNCHECK=2`, any error messages are printed and simulation stops if connectivity errors are detected.

Description

`HSIMCONNCHECK` controls connectivity checking.

HSIMDCI

Controls DC convergence checking.

Syntax

```
.param HSIMDCI = value
```

Description

During DC convergence, a convergent state is achieved if, for each DC iteration, the current flowing into any net changes by less than `HSIMDCI`. By default, HSIM only checks against `HSIMDCV`. `HSIMDCI` is only checked if `HSIMIDDQ` is enabled. The default value is 5E-7. Units are in Amperes.

HSIMDCINIT

Controls the initialization of DC analysis.

Syntax

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

Arguments

Argument	Description
`0`	Use this argument for situations when there is no need for DC initialization. When `HSIMDCINIT` is set to 0, DC initialization is suppressed and the transient simulation starts immediately following the preprocessing. Under this condition, the DC initialization proceeds as if running the transient simulation using zero initial state for each node voltage, unless the node has an initial condition setting through the use of the .ic command.
`1`	When `HSIMDCINIT` is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the number of iteration steps exceeds the limit specified by HSIMDCITER on page 64.

Description

The `HSIMDCINIT` control parameter is used to activate DC initialization. DC initialization is activated after the netlist processing and hierarchical database building phase is completed and before the beginning of a transient simulation.

A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by HSIMDCV on page 66, and each net current is less than the current specified by HSIMDCI on page 63. These commands can be configured to create levels of precision for DC operation needed in their specific circuits.

HSIMDCITER

Limits the number of DC interation steps.

Syntax

```
.param HSIMDCITER = value
```

Description

When HSIMDCINIT on page 63 is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the

number of iteration steps exceeds the limit specified by `HSIMDCITER`. A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by HSIMDCV on page 66, and each net current is less than the currentspecified by HSIMDCI on page 63.

HSIMDCOUTFMT

Sets the DC output format.

Syntax

```
.param HSIMDCOUTFMT = <0|2|4|7>
```

Arguments

Argument	Description
0	The default output format is ASCII table. Results are written to the <project>.dc file.
2	This output format is ASCII raw file. Results are written to the <project>.dc.raw file.
4	This format is similar to `HSIMDCOUTFMT=2`, however if there is an external sweep in DC analysis, the results for each value of the external sweep go to a separate external file such as the following: <project>.dc.raw.s1, <project>.dc.raw.s2, and so on.
7	This format is used for output formats of third party vendors. Use this option with HSIMOUTPUT on page 105. Combining `HSIMDCOUTFMT` and HSIMOUTPUT on page 105 allow the DC result to be printed into third party vendor output formats

Description

Currently, HSIM only supports the WDF format as shown in the following example.

Example

```
.param HSIMDCOUTFMT=7
.param HSIMOUTPUT=wdf
```

HSIMDCSOIACC

Controls the use of the BSIM3SOI model.

Syntax

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

Description

`HSIMDCSOIACC` controls whether the BSIM3SOI model is used with the self-heating turned on or off during DC initialization. To turn self-heating off, set `HSIMDCSOIACC` to 0. To turn it on, set the value to 1.

HSIMDCSTEP

Sets the DC initialization time step.

Syntax

```
.param HSIMDCSTEP = value
```

Description

Determines the smallest time step used in DC initialization. The default is 1000. Units are in picoseconds.

HSIMDCSTOPAT

Stops DC analysis.

Syntax

```
.param HSIMDCSTOPAT = value
```

Description

Specifies the interation number at which to stop DC analysis.

HSIMDCV

Controls the DC convergent state theshold for each DC iteration.

Syntax

```
.param HSIMDCV = value
```

Description

During DC convergence, a convergent state is achieved if for each DC iteration a node voltage change is less than `HSIMDCV`. The default value is 0.001. Units are in Volts.

HSIMDELAYNTLRMIN

Removes resistors with values smaller than the HSIMNTLRMIN value.

Syntax

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

Description

`HSIMDELAYNTLRMIN` removes resistors with values smaller than HSIMNTLRMIN on page 103. If a DSPF/SPEF file is specified, back-annotation is performed.

When `HSIMDELAYNTLRMIN=1`, shorting the resistors according to HSIMNTLRMIN on page 103 is delayed until after back-annotation.

HSIMDELVTO

Specifies flash core transistor usage.

Syntax

`.param HSIMDELVTO = devto_valuesubckt = subcircuit_name`

Description

`HSIMDELVTO` sets DELVTO for individual transistors or those in subcircuits so the transistor can be used on flash core cells as shown in the following example.

Note:

Previously, `HSIMDELVTO` was allowed on instances only but is now extended to subcircuits.

Example

```
.param HSIMDELVTO=0.1 subckt=core_sub
```

This example sets DELVTO to 0.1 for all transistors in the core_sub subcircuit.

HSIMDETAILBSIM4

Provides the RGATEMOD model parameter with good accuracy.

Syntax

```
.param HSIMDETAILBSIM4 = <0|1>
```

Description

Specify `HSIMDETAILBSIM4 = 1` only for the BSIM4 model to provide the RGATEMOD model parameter with good accuracy. The default is `HSIMDETAILBSIM4`=0.

HSIMDIOCC

Invokes the exact charge conservation model for the charge stored by the diode capacitors.

Syntax

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

Description

`HSIMDIOCC` invokes the exact charge conservation model for the charge stored by the diode capacitors during the simulation. The default is 0.

HSIMDIOCV

Invokes the approximate charge conservation model for the charge stored by the diode capacitors.

Syntax

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

Description

`HSIMDIOCV` invokes the approximate charge conservation model for the charge stored by the diode capacitors during the simulation. The default is 0.

HSIMDIODECURRENT

Calculates the dc current of MOSFET diodes.

Syntax

```
.param HSIMDDIODECURRENT = <0|1|2>
```

Description

The default of 0 does not calculate the DC current of MOSFET parasitic diodes.

When set to 1, this command calculates the DC current of MOSFET parasitic diodes. In addition, set the `HSIMBMOS` parameter to ensure a correct partition to calculate current calculation effect.

When set to 2, this command also calculates the DC current of MOSFET parasitic diodes. If the`HSIMABMOS` parameter s also set, there is much greater performance penalty when `HSIMDIODECURRENT` is set to 2.

HSIMDIODEVT

Calculates the junction diode current.

Syntax

```
.param HSIMDIODEVT = current
```

Description

In CMOS circuitry, MOSFET junction diodes are usually in the reverse bias region and diode current can be ignored. If a MOSFET junction diode becomes forward biased during simulation, and its voltage drop is bigger than`HSIMDIODEVT` ,and MOSFET drain, SOURCE, and BULK are in the same partition, HSIM calculates the junction diode current. The default current is 0.5

HSIMDPF

Defines the DPF file for device parameter back-annotation.

Syntax

```
.param HSIMDPF = DPF_file
```

Description

The advantage of DPF back-annotation is that the pre-layout hierarchy is maintained for simulation.

For MOSFET devices, the supported DPF parameters are: L, W, AD, AS, PD, PS, NRD, NRS, SA, SB, SD, NF, DELVTO, MULU0, RGEOMOD, RDC, RSC, SCA, SCB, SCC, SA1, SA2, SA3, SA4, SA5, SA6, SA7, SA8, SA9, SA10, SB1,

SB2, SB3, SB4, SB5, SB6, SB7, SB8, SB9, SB10, SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8, SW9, SW10.

HSIMDPFHIERID

Specifies the hierarchical separator used in DPF file.

Syntax

```
.param HSIMDPFHIERID = separator_character
```

Description

If the hierarchical separator used in DPF file is different from `HSIMDPFHIERID`, the hierarchical separator must be specified with `HSIMDPFHIERID`. The default character is a period (".").

HSIMDPFMERGEFDEV

Controls how fingered devices in DPF file are processed.

Syntax

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

Arguments

Argument	Description
`0`	Splits devices in the pre-layout netlist to match the number of fingered devices in the SPF/SPEF file before back-annotation.
`1`	Shorts all fingered devices during back-annotation (default).

Description

If .param `HSIMDPFMERGEFDEV` is specified without a value, 0 is assumed. If `HSIMDPFMERGEFDEV` is not specified, 1 is used.

HSIMDPFMULTISUB

Controls how to read subcircuits in a DPF file.

Syntax

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

Arguments

Argument	Description
`0`	Does not read multiple subcircuits in a DPF file (default).
`1`	Allows reading multiple subcircuits in a single DPF file.

Description

Allows reading multiple subcircuits in a DPF file.

HSIMDPFPFX

Specifies the DPF prefix string.

Syntax

```
.param HSIMDPFPFX = prefix_character
```

Description

The device name in the DPF file usually comes in two types. The extractor may reverse the hierarchy or prepend a string with leading M to the device name. For example, in pre-layout netlist, A device named x1.x2.mp1 can become M_x1.x2.mp1 or mp1.x2.x1 in the DPF file.

HSIM automatically checks for reverse hierarchy in device name. But if the name is prepended with a prefix string, you need to specify that prefix string using `HSIMDPFPFX` so HSIM can correctly extract the hierarchical name. The default prefix string is m_.

HSIMDPFPREPORTNOBA

Controls whether to generate a report for input netlist devices that are not back-annotated.

Syntax

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

Arguments

Argument	Description
`0`	Does not generate a report for input netlist devices not back-annotated (default).
`1`	Generates a report for input netlist devices not back-annotated.

Description

Checks and reports any input netlist devices that are not DPF back-annotated. The report is stored in an `output_filename.dpfnoba` file.

HSIMDPFSCALE

Specifies the scale factor for device parameters in the DPF file.

Syntax

```
.param HSIMSCALE = unit
```

Description

Specifies the scale factor for device parameters in the DPF file. For example, if the unit used in the DPF file is micron, then you need to specify `HSIMDPFSCALE=1u`.

HSIMDPFSFX

Specifies the delimiter character used on fingered devices.

Syntax

```
.param HSIMDPFSFX = suffix_character
```

Description

Specifies delimiter the delimiter character used on fingered devices in the DPF file.The default character is an asterisk ("@").

HSIMDPFSPLITFD

Back-annotate the parameter values for finger devices.

Syntax

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

Arguments

Argument	Description
`0`	Does not back-annotate the parameter values for each individual finger device (default).
`1`	Back-annotate the parameter values for each individual finger device.

Description

When HSIM encounters finger devices during DPF annotation, the device parameter values for each finger device group are averaged during the back-annotation process. In order to back-annotate the parameter values for each individual finger device, `HSIMDPFSPLITFD` must be set to 1 (default is 0).

HSIMDTNAME

Forces HSIM to recognize the specified terminal names for a particular device type.

Syntax

```
.param HSIMDTNAME = "device_type:term1, term2, term3,... "
```

Description

This parameter is used when uncommon device terminal names are specified in the post-layout netlists. It provides a mechanism to force HSIM to recognize the specified terminal names for a particular device type.

Example

In this example, device_type can be i,v,m,q,r,c,d, and l. The number of terminals specified must correspond to the number of terminals associated with the particular device type.

To use tm1, tm2, tm3, and tm4 to denote terminals of a MOS device in the post-layout file (for example, m1:tm2), specify:

```
.param HSIMDTNAME="m:tm1,tm2,tm3,tm4"
```

HSIMDUMPOPI

Dumps element currents to an operating point.

Syntax

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

Description

Set `HSIMDUMPOPI` to dump element currents to an operating point. The default is 0.

HSIMEFFICIENTTBL

Accounts for STILOD/WPE effects of pre-layout or post-layout simulation.

Syntax

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

Description

Set `HSIMEFFICIENTTBL = 1` with the instance parameters SA, SB, SD, SCA, SCB, SCC, SC in the netlist to account for STILOD/WPE effects of pre-layout or post-layout simulation to minimize simulation memory with BSIM4 table models. This command dynamically updates the table values by considering these parameters during simulation while maintaining accuracy.

This command applies only to table models. However, if you set `HSIMSPICE=1` or higher, that the table models are turned off and direct modeling is used.

HSIMENHANCECAP

Applies and internal table to capacitance values from BSIM equations.

Syntax

```
.param HSIMENHANCEDCAP = <0|1|2>
```

Arguments

Argument	Description
`0`	Calculates the capacitance values of a MOSFET transistor with an efficient formula (default).
`1`	Applies a compact internal table to calculate capacitance values from the associated BSIM equations.
`2`	Uses a more extensive internal table to calculate capacitance values from detailed BSIM equations. For the BSIM4 model, the default setting is equivalent to `HSIMENHANCECAP`=1.

Description

For the BSIM4 model, the default setting is equivalent to `HSIMENHANCECAP`=1.

HSIMENHANCEDC

Controls DC initialization.

Syntax

```
.param HSIMENHANCEDC = <0|1|2|3>
```

Arguments

Argument	Description
`0`	Does not manually specify a DC initialization algorithm (the default).
`1`	Uses a more conservative DC initialization algorithm and settings and increases the DC iteration limit to 1000.
`2`	Utilizes the voltage-dependent MOSFET capacitance model in DC initialization.
`3`	Opens the potential for performance trade-offs to provide a more conservative approach targeted at achieving better DC convergence results.

Description

Specifies the DC initialization algorithm.

HSIMENHANCEDIDDQ

Speeds up the simulation by setting a pre-defined time slot using a PWC function.

Syntax

`.param HSIMENHANCEDIDDQ =`*`pwc_function_syntax`*

Description

`HSIMENHANCEDIDDQ` causes HSIM to run very slowly during transient simulation. To speed up the simulation, set a pre-defined time slot using a PWC function in IDDQ current evaluation. The following example turns on the IDDQ current evaluation during a window defined by the `20n, 30n` parameters.

Example

```
.param HSIMENHANCEDIDDQ=pwc(On, 0, 20n, 2, 30n, 0)
```

In addition to setting `HSIMENHANCEDIDDQ` on user-selected subcircuits, this command can also be activated during some simulation periods using PWC syntax. This flexibility of spatial and temporal specifications can be used to analyze the subcircuit and the time period where accurate device model and

leakage current computations are needed and help reduce CPU time for quiescent or leaking current measurements.

```
.hsimparam subckt=acu_blk / HSIMENHANCEDIDDQ=pwc(0.0n,0,100n,1)
```

The syntax in this example sets enhanced IDDQ on the acu_blk block to 1 after time=100n, causing the simulation to proceed as usual before 100n. After 100n, the most accurate model and evaluation is employed for leakage current, device model computation, and so on.

HSIMENHANCEMOSIV

Calculates static drain current.

Syntax

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

Description

When `HSIMENHANCEMOSIV = 0`, HSIM applies a compact internal table to calculate the static drain current value of a MOSFET transistor. When `HSIMENHANCEMOSIV = 1`, it uses a more extensive internal table to calculate the drain current value in order to capture the fine detail of the device characteristics.

Note:

The default grid number in the Vbs dimension is 6.

HSIMFLAT

Selects a flat or a hierarchical simulation.

Syntax

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

Description

In simulating subcircuit cells, the efficiency gained by saving identical subcircuit computations is significant enough to compensate for the overhead. If each subcircuit cell has a different behavior during the simulation, then the flat simulation might have an efficiency advantage over hierarchical simulation. Hierarchical simulation still has an advantage in saving the storage space. To

allow a trade-off choice in memory usage and simulation speed, the `HSIMFLAT` command enables either a flat or a hierarchical simulation.

When H`SIMFLAT=1`, HSIM does not flatten the entire circuit; it only selectively flattens those subcircuits with no sufficient subcircuit instances, or the subcircuit that has a different behavior from other subcircuit instances. The default is `0`.

HSIMFMODLIB

Specifies the library path to the netlist.

Syntax

```
.param HSIMFMODLIB = lib_file_name
```

Description

Lets you specify the path to the library file to link to the circuit netlist.

HSIMFORWARDBIASDIODE

Calculates static drain.

Syntax

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

Arguments

Argument	Description
`0`	Applies a compact internal table to calculate the static drain current value of a MOSFET transistor (the default).
`1`	Uses a more extensive internal table to calculate the drain current value in order to capture the fine detail of the device characteristics. The default grid number in the Vbs dimension is 6.

Description

`HSIMENHANCEMOSIV` applies internal tables to calculate static drain.

HSIMFSDBDOUBLE

Specifies how to store signal values.

Syntax

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

Description

Set `HSIMFSDBDOUBLE = 1` to store signal values in double precision for the FSDB output format. The default is 0, which stores signal values as floating point.

HSIMGATEMOD

Models constant gate load resistance.

Syntax

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

Description

Set `HSIMGATEMOD` to model the constant gate electrode resistance. The default is 0.

HSIMGCSC

Controls the calculation of MOSFET gate and junction diode capacitances.

Syntax

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

Arguments

Argument	Description
`0`	Simplifies the voltage-dependent MOSFET capacitances to average capacitances and ignores the Miller Effect between gate-to-drain and gate-to-source terminals. This simplification typically achieves a 2X speedup but may lose some precision, especially for circuits operating at high frequencies in which the timing delay is sensitive to how the MOSFET capacitances are handled. This value is recommended for low-frequency applications or high-speed functionality simulation where 5-10% precision loss can be tolerated.
`1`	This default value specifies the most precise mode, which you can set at various levels of the circuit hierarchy.When`HSIMSPEED = 8` and `HSIMGCSC` are set, the simulation time steps are controlled conservatively. If `HSIMGCSC` is set at local circuits, only these circuit parts are affected by the conservative time step selection.

Description

The MOSFET gate and junction diode capacitances are voltage-dependent. This requires calculation for each capacitance value within the MOSFET at every time step. In addition, the Miller Effect due to the coupling capacitors between gate-to-drain and gate-to-source terminals requires more computations and slows down the simulation speed.

HSIMGMIWELM

supports the HSPICE -compatible W-element of the RLGC format.

Syntax

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

Description

HSIM supports the HSPICE compatible W-element of the RLGC format in model or file formats with the same syntax. To invoke this option, use `HSIMGMIWELM=1`. The default value (0) uses the HSIM implementation.

HSIMHASCOMPLEXBJT

Controls simulation when using a complex BJT model, such as VBIC or Mextram.

Syntax

```
.param HSIMHASCOMPLEXBJT = <0|1|2>
```

Arguments

Argument	Description
`0`	Sets `HSIMTIMESCALE` and `HSIMALLOWEDDV` to their default values.
`1`	Sets `HSIMTIMESCALE` to 0.001 and `HSIMALLOWEDDV` to 0.1, unless smaller values are explicitly specified for subcircuits containing VBIC or Mextram devices.
`2`	Tightens simulation to be more synchronous, in addition to tightening up HSIMTIMESCALE on page 151 and HSIMALLOWEDDV on page 47.

Description

When a complex BJT model such as VBIC or Mextram is used in a design, it is recommended to set `HSIMHASCOMPLEXBJT = 1` or `=2`. When a Complex BJT model such as VBIC or Mextram is used in a design, it is recommended to set `HSIMHASCOMPLEXBJT=1` or `=2`.

HSIMHIERID

Specifies the hierarchical separator character.

Syntax

```
.param HSIMHIERID = separation_character
```

Description

`HSIMHIERID` specifies the hierarchical separator character. The following example illustrates the use of colon (:) as the hierarchical separator.

```
.param   HSIMHIERID=':'
```

The default is the period character (.).

HSIMHSPICEVEC

Creates a compatible format between HSPICE® and HSIM to handle the slight difference in formats for the bus notation of vector files.

Syntax

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

Description

Suppose the bus notation in a vector file is:

```
vname ck[1:3]
```

When `HSIMHSPICEVEC = 0` (the default), this bus is recognized as `ck[1] ck[2] ck[3]`. When `HSIMHSPICEVEC = 1`, this bus is recognized as `ck1 ck2 ck3`. It is compatible with HSPICE.

HSIMHZ

Performs Hi-Z node checking.

Syntax

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

Description

When `HSIMHZ = 1`, HSIM performs Hi-Z node checking. If the Hi-Z check detects a Hi-Z node on one subcircuit instance, it is possible that there are more Hi-Z nodes on other similar subcircuit instances.

If the expected state (from vector file) is "x", there is no "DOUT error" triggered regardless of the measured state (from simulation).

If the expected state is not "x", only the following three conditions trigger a "DOUT error" :

- expect = 1 state = 0
- expect = 0 state = 1
- expect= (any thing other than “x”) state= x

Note:

`HSIMHZ = 1`reports only one Hi-Z node from all possible Hi-Z nodes on other similar subcircuit instances in order to streamline the Hi-Z node report and improve performance. The default is `HSIMHZ = 0`.

HSIMHZALL

Overwrites the `HSIMHZ` default behavior.

Syntax

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

Description

When `HSIMHZALL = 1` and `HSIMHZ = 1` are set, all existing Hi-Z nodes found on similar subcircuit instances are reported. All Hi-Z results reported using `HSIMHZALL = 1` should match the results achieved when pcheck zstate is run. The default is `HSIMHZALL = 0`.

HSIMHZFANOUT

Limits high impedance state checks to those driver nodes with fan-outs.

Syntax

```
.param HSIMHZFANOUT = <0|1|2|3>
```

Arguments

Argument	Description
`0`	Checks all nodes (the default).
`1`	Checks the nodes that have a direct connection to a gate of transistors.
`2`	Checks nodes having a direct connection to a transistor bulk.
`3`	Checks nodes having a direct connection to a transistor gate, or to a BJT base.

Description

To avoid unnecessary internal node checks, `HSIMHZFANOUT` limits high impedance state checks to those driver nodes with fan-outs.

HSIMHZNDNAME

Defines the node name to be checked.

Syntax

```
.param HSIMHZNDNAME = 'node_name'
```

Description

`HSIMHZNDNAME` defines the node name to be checked. Each node name can be the name of a single node or a node name containing an wildcard asterisk (*) character which represents a group of node names.

HSIMHZSTART

Starts the check at the specified start time value.

Syntax

```
.param HSIMHZSTART = start_time
```

Description

Checks are performed within the time window defined by `HSIMHZSTART` and `HSIMHZSTOP`. You can use multiple time windows with multiple `HSIMHZSTART` and `HSIMHZSTOP` values.

HSIMHZSTOP

Stops the check at the specified stop time value.

Syntax

`.param HSIMHZSTOP = stop_time`

Description

Checks are performed within the time window defined by`HSIMHZSTART` and `HSIMHZSTOP`. You can use multiple time windows with multiple `HSIMHZSTART` and `HSIMHZSTOP` values.

HSIMHZTIME

Threshold time for a specified node staying in the high-impedance state to be reported.

Syntax

`.param HSIMHZTIME = time`

Description

Any specified node staying in the high-impedance state longer than `HSIMHZTIME` is reported in the `title_name.hz` file.

`HSIMHZTIME` is specified in nanoseconds. The default is 5.

HSIMHZXSUBCKT

Excludes the specified subcircuit from checking.

Syntax

`.param HSIMHZXSUBCKT = subckt_name`

Description

`HSIMHZXSUBCKT` excludes checking the specified subcircuit. To exclude multiple subcircuits, use one `HSIMHZXSUBCKT` for each subcircuit.

HSIMIDDQ

Tightens the simulation accuracy criteria.

Syntax

```
.param HSIMIDDQ = <1|2>
```

Arguments

Argument	Description
`1`	Increases the DC iteration number for higher quality DC convergence. It also tightens up the precision criterion during transient simulation. Leakage current between is measured in the V_{DD} and G_{ND} nodes. The elements that conduct the current can be either in the strong-inversion region or in the subthreshold region.
`2`	Further tightens the precision criterion for more enhanced accuracy. Leakage is measured current through the regular or parasitic diode junction.

Description

The steady current state is commonly referred to as the IDDQ current. IDDQ consists of two components: `HSIMIDDQ=1` and HSIMIDDQ=2. `THSIMIDDQ=1` and `HSIMIDDQ=2` provide control commands to tighten up the precision criterion for the IDDQ measurement. To obtain accurate IDDQ measurement results, you must tighten up the simulation accuracy criterion.

HSIMIGISUB

Calculates the static gate leakage currents and the substrate current.

Syntax

```
.param HSIMIGISUB = <0|1|2|3>
```

Arguments

Argument	Description
`0`	Default.
`1`	Includes both the static gate leakage currents and the substrate current in the calculation.
`2`	Includes the substrate current in the calculation but not the gate leakage currents.
`3`	Includes the gate leakage currents in the calculation but not the substrate current.

Description

In order to minimize the memory usage and the computation time, by default, the static gate leakage currents and the substrate current are not included in the calculation of I-V result for a MOSFET transistor with the BSIM4 model. You can use the `HSIMIGISUB` command to specify the calculation.

HSIMINACTOPT

Controls how to generate inactive and active list files.

Syntax

```
.param HSIMINACTOPT = <1|2|3>
```

Arguments

Argument	Description
1	Generates the entire inactive list file .ina_all.
2	Generates the surrounding inactive list file .ina (the default).
3	Generates both the .ina_all and .ina files.

Description

The active .ach list file is always created with .acheck command. The inactive list file .ina or .ina_all can be selectively created. Use the `HSIMINACTOPT` command to control how to generate the files.

HSIMIPRECISION

Controls current precision. For details about the `HSIMIPRECISION` settings, see HSIM[plus] Quick Start on page 12.

Syntax

```
.param HSIMIPRECISION = <0|1|2|3|4|5>
```

Description

`HSIMIPRECISION` controls current precision. Similar to `VPRECISION` for voltage, `HSIMIPRECISION` defines current thresholds considered to be negligible during simulation.

HSIMITERMODE

Enables the Newton Raphson iteration scheme.

Syntax

```
.param HSIMITERMODE = <0|1|2>
```

Arguments

Argument	Description
`0`	Does not enable this mode (the default).
`1`	Uses the HSIM proprietary iteration scheme, which balances accuracy and performance.
`2`	Uses SPICE-like Newton Raphson iteration.

Description

The Newton Raphson iteration scheme is available in HSIM to provide a high level of accuracy using the iteration algorithm. The trade off is that using this algorithm might cause impact performance.

HSIMKEEPALLGNDCAPS

Specifies whether capacitors connected to a zero constant voltage source are lumped to the signal node.

Syntax

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

Description

Specifies whether or not to keep capacitors connected to a zero constant voltage source. Set `HSIMKEEPALLGNDCAPS=`1 to keep all ground capacitors independent of the HSIMLUMPCAP on page 91 setting. You can also specify `HSIMLUMPCAP=0` with `HSIMKEEPALLGNDCAPS` to specify whether (`HSIMKEEPALLGNDCAPS=0`) or not (`HSIMKEEPALLGNDCAPS=1`) capacitors connected to a zero constant voltage source are lumped to the signal node. See HSIMLUMPCAP on page 91 for more details.

HSIMKEEPNODESET

Keeps nodeset values.

Syntax

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

Description

Ensures that all nodeset values are kept during the first 1/10 of the total iterations. HSIM automatically identifies cross-coupled nodes in circuits and applies a logic 1 nodeset value to one of the nodes. This avoids a meta stable condition after DC initialization. Since the nodeset value is kept only at the first iteration, it can be overridden. The default is 0.

HSIMLMIN

Sets the minimum inductor value.

Syntax

```
.param HSIMLMIN = value
```

Description

Sets the minimum inductor value allowed in simulation. All inductors with a value smaller than `HSIMLMIN` are short-circuited. This command is used at the netlist parser of HSIM to eliminate small inductors. It can only be set globally. The default is 1E-11.

HSIMLIS

Concatenates input files in a single file.

Syntax

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

Description

When `HSIMLIS = 1`, HSIM concatenates all input files into a single file.

HSIMLOGDCPROGRATE

Controls the DC simulation progress status reporting.

Syntax

```
.param HSIMLOGDCPROGRATE = progress_rate_value
```

Description

`HSIMLOGDCPROGRATE` controls the DC simulation status report. DC simulation status is reported when 10% of the total iteration is completed. The default is 0.1.

HSIMLOGPROGRATE

Controls the DC simulation progress status reporting.

Syntax

```
.param HSIMLOGPROGRATE = progress_rate_value
```

Description

`HSIMLOGPROGRATE` controls the transient simulation status report. Transient simulation status is reported when 10% of the total iteration is completed. The default is 0.1.

HSIMLUMPCAP

Keeps all the capacitors as regular elements.

Syntax

```
.param HSIMLUMPCAP = <0|1|2>
```

Arguments

Argument	Description
0	If HSIMLUMPCAP=0 and HSIMKEEPALLGNDCAPS=0, only the capacitors connected to the zero constant voltage source are lumped to the signal node. The capacitors connected to the non-zero constant voltage source are not lumped to the signal node. If HSIMLUMPCAP=0 and HSIMKEEPALLGNDCAPS=1, capacitors connected to either zero or non-zero voltage sources are not lumped to the signal nodes. The default value of HSIMKEEPALLGNDCAPS is 0. Use HSIMLUMPCAP=0 only if the simulation requires very high precision on peak current value.
1	HSIMLUMPCAP=1 is the default. All capacitors from signal nodes to constant voltage source nodes are lumped to the signal nodes. The default value is suitable for most applications.
2	HSIMLUMPCAP=2 keeps the gnd capacitors for back-annotation and removes them after back-annotation.

Description

Use the HSIMLUMPCAPcommand to keep all the capacitors as regular elements.

Note:

This command can be expensive in terms of CPU time and memory usage.

HSIMMEASOUT

Generates an output file from a .measure command.

Syntax

```
.param HSIMMEASOUT = <0|1|2>
```

Arguments

Argument	Description
0	Generates <prefix>.mt<.a#> as the output file, where <prefix> is the output prefix, <.a#> is the alter number if alter is used (default).
1	Generates <prefix>.mt<.#> as the output file, where <prefix> is the output prefix, <.#> is the alter number if alter is used.
2	Generates <prefix>.mt.all as the output file, where <prefix> is the output prefix.

Description

If you specify 0, the format in the file (for example, hsim.mt.a0) is a user-readable format. If you specify 1, the format in the file (for example, hsim.mt.0) is in HSPICE-like (tabular) format. If you specify 2, if alter is used, all data is outputted into the file (for example, hsim.mt.all) in HSPICE tabular format similar to HSIMMEASOUT=1.

HSIMMONITORMRES

Monitors and reports the initial state conditions and/or state transition information for MRAM bit cells.

Syntax

```
.param HSIMONITORMRES = <0|1|2|3>
```

Arguments

Argument	Description
`0`	Turns off `HSIMMONITORMRES` (the default).
`1`	Reports MRES state transitions to the `outputprefix.monitormres` file.
`2`	Same as `1`, plus print s a state change message to screen and `.log` file.
`3`	Same as `2`, plus prints ALL MRES initial states to the `outputprefix.icmres` file.

Description

HSIM has the ability to monitor and report the initial state conditions and/or state transition information of any MRAM bit cells when using the MRES built-in proprietary MRAM models.

HSIMMODELSTAT

Lists model statistics in the log file.

Syntax

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

Description

You can view the number of instances created for each device model by setting `HSIMMODELSTAT = 1` If set, a table listing the model statistics in the log file is displayed. `HSIMMODELSTAT` is turned off by default. A sample statistics table is shown in the following example.

Example

```
Model Statistics
MOS - n: 10
MOS - p: 25
JFET - njf1: 5
BJT - bjt1: 6
```

HSIMMONTECARLO

Sets the Monte Carlos analysis mode.

Syntax

```
.param HSIMONTECARLO = <0|1|2>
```

Arguments

Argument	Description
`0`	Sets all model parameters to be supported.
`1`	The default of 1 is mainly for an analog circuit that does not have high isomorphic matching.
`2`	Use this value for memory circuits with better isomorphic matching.

Description

If `HSIMMONTECARLO = 1` or `= 2`, only a limited parameter set is supported:

LW

AD

AS

PD

PS

NRD

NRS

M

HSIMMONTECARLOINST

Prints the device instance parameter variations.

Syntax

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

Description

Set `HSIMMONTECARLOINST = 1` to print the device instance parameter variations for each iteration. Instance parameter variations are saved in files ending with the .mip suffix.

HSIMMONTECARLOSAVEOUT

Keeps all Monte Carlo iteration files.

Syntax

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

Description

By default, Monte Carlo analysis only keeps the of the last iteration of following files:

- Waveform
- Measure
- Model Parameter Variation
- Device Instance Parameter Variation

Set `HSIMMONTECARLOSAVEOUT = 1`to keep all of the files listed above for each iteration with the .m# suffix.

The final statistical information results of all .measure commands are saved into the .mc file for transient analysis or .dc_mc file for DC analysis. The output contains the following information:

Nominal Result

The measured result with the original value defined in parameter distribution function.

Mean Result

The average of all the measured results of a measure statement after iteration is completed.

Variance

The following formula:

$$\frac{\sum_{i=1} (x_i - x_{ave})^2}{n}$$

where the following apply:

where the following apply:

- xi is the measured result of a measure statement at the `i`th HSIM iteration.
- Xave is the Mean Result.
- i=1,...n, and n number of HSIM iterations.

Standard Deviation

The square root of the variance.

Minimum Result

The smallest measured result of a measure statement.

Maximum Result

The largest measured result of a measure statement.

Average Deviation

The average deviation is calculated as follows:

$$\frac{\sum_{i=1} |x_i - x_{ave}|}{n}$$

nb

The MCA histogram is depicted in 10 bins, evenly divided between the minimum and maximum result. The number per bin is the number of measured results that fall into that value range.

run_max and run_min

The index for the measured files where the maximum and minimum results are stored.

Example

Example 1 HSIM Monte Carlo Example

```
input:
.tran 1e-011 10u sweep monte=45
.param HSIMMONTECARLO=1
.param mlen1=gauss(2e-005, .5, 3)
.param mlen2=agauss(1.47e-006, 1e-012, 3)
.param mwidth1=unif(2e-005, .5)
.param mwidth2=aunif(1.47e-006, 1e-012)
.meas vx1vcocontrol find v(x1.vcoctrl) at=9.5u
.
output:
meas_variable=vx1vcocontrol
nominal=2.480476e-001
mean=2.481729e-001varian=4.461293e-005
stddev=6.679291e-003avgdev=5.256493e-003
min=2.336792e-001max=2.669093e-001
run_min=1      run_max=14
2.370022e-001, nb=1, freq=2.222222e-002 |*
2.403252e-001, nb=5, freq=1.111111e-001 |********
2.436482e-001, nb=5, freq=1.111111e-001 |********
2.469712e-001, nb=8, freq=1.777778e-001 |*************
2.502943e-001, nb=12, freq=2.666667e-001 |**************
2.536173e-001, nb=6, freq=1.333333e-001 |**********
2.569403e-001, nb=5, freq=1.111111e-001 |********
2.602633e-001, nb=1, freq=2.222222e-002 |*
2.635863e-001, nb=1, freq=2.222222e-002 |*
2.669093e-001, nb=1, freq=2.222222e-002 |*
```

HSIMMOSPRECISION

Controls MOSFET small signal approximations. For details about the `HSIMMOSPRECISION` settings, see HSIM[plus] Quick Start on page 12.

Syntax

```
.param HSIMMOSPRECISION = <0|1|2|3|4|5>
```

Description

`HSIMMOSPRECISION` is a macro command. It controls MOSFET small signal approximations that are available to HSIM. Simulators apply differing degrees of approximations to increase performance. For example, some digital designs can be simulated with enough precision using average capacitance models, whereas sensitive analog designs might require voltage dependent capacitance modeling and charge conservation.

HSIMMQS

Controls dynamic time step adjustment during transient simulation.

Syntax

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

Arguments

Argument	Description
0	Specifies the most conservative selection. This option requires a uniform time step for every node at any time.
1	Specifies the stepsize at each node, depending on its time constant, and can be multiple times of minimal stepsize at that time (default).

Description

During the transient simulation, the time step is adjusted dynamically such that the voltage change over the time step is limited to a predefined voltage value, defined by HSIMALLOWEDDV on page 47.The time constant of the transient behavior at each node of the circuit could be very different at the same time; namely, a sharp transition could occur to a node at the same time when other nodes have very slow transition. If a uniform time step is chosen for every node in the circuit, then the smallest time step among all the nodes is needed in order to meet the requirement that the voltage increment over each time step is no more than the value defined by HSIMALLOWEDDV on page 47. Such uniform stepsize selection apparently is too conservative because it is unnecessarily small for those nodes with slow transition.

HSIMMULTIDC

Enables the multi-rate time-step algorithm during DC convergence to improve performance.

Syntax

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

Description

To disable multi-rate time-step selection during DC convergence, set `HSIMMULTIDC=1` (the default). To enable the multi-rate time-step selection during DC convergence, set `HSIMMULTIDC=0`.

HSIMMSGFILTER

Filters messages.

Syntax

`.param HSIMMSGFILTER = [`*`max_messages`*`:] "`*`message_text`*`"`

Arguments

Argument	Description
max_messages:	Limits the number of messages to suppress.
message_text	Specifies the message text to suppress from printing.

Description

Add `HSIMMSGFILTER` at the beginning of the top-level netlist in order to have an effect on the messages displayed.

HSIMNODCNODES

Lists the nodes that do not converge at the completion of DC initialization.

Syntax

`.param HSIMNODCNODES = <0|1>`

Description

Specify `HSIMNODCNODES = 1` to generate a *prefix*`.nodcnodes` file that lists the nodes that do no converge at the completion of DC initialization. The default is 0.

HSIMNODECAP

Controls how to report average capacitance.

Syntax

```
.param HSIMNODECAP="<subckt_name> node_pattern <level>"
```

Description

When HSIM is invoked, if -o file_name is specified, the average capacitance of the specified `node_pattern` is reported and stored in the capacitance report file hsim.cap or file_name.cap. If the string `node_pattern` contains asterisk (*) wildcard character(s), or if a subcircuit name or level is specified, that string must be enclosed with double (")or single (') quotation characters as shown in the following example.

Example

```
.param HSIMNODECAP="x1.x2.*"
.param HSIMNODECAP='x1.*.x2.*'
```

- subckt_name

  If subckt_name is specified, only nodes in instances of the given subcircuit are printed.

- level

  If level is specified and the node pattern contains wild card characters, matching is only applied up to the number of hierarchical levels specified.

HSIMNOMULTIEND

Errors out during parsing when multiple .end statements exist.

Syntax

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

Description

HSIM accepts multiple .end statements. If you want HSIM to error out during parsing when multiple .end statements exist, set `HSIMNOMULTIEND=1`.

HSIMNOSIMTIME

Controls simulation time reporting.

Syntax

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

Arguments

Argument	Description
`0`	Displays a stop watch on screen to report the current simulation time. The time display updates at every 0.01% of total transient simulation time is completed. This is the default value.
`1`	Turns the stop watch feature off.

Description

Set `HSIMNOSIMTIME` to control the simulation time reported on the screen.

HSIMNTLFMT

Specifies the netlist format of the input file.

Syntax

`.param HSIMNTLFMT = netlist_format`

Description

`HSIMNTLFMT` specifies the netlist format of the input file. It is an alternative to the -<netlist_format> command line option. It is a global parameter and its default is hspice. When using this command, the following restriction applies: `HSIMTOP` and `HSIMNTLFMT` must be the first commands in the input.

Example

The following two methods have the equivalent effect.

Method 1

Place HSIMNTLFMT=spectre in the input file, then enter the following at the command line:

```
A> HSIM input_spectre_file
```

Method 2

Run HSIM at the command line as shown below:

```
A> HSIM -spectre input_spectre_file
```

HSIMNTLRMIN

Shorts resistors with values smaller than a specified threshold value.

Syntax

```
.param HSIMNTLRMIN = threshold_value
```

Description

`HSIMNTLRMIN`Is a subcircuit-level parameter that shorts resistors with values smaller than a specified threshold immediately after parsing a netlist. If `HSIMNTLRMIN` is not set, the default value is the current value of `HSIMRMIN`.

HSIMNVBS

Specifies the grid resolution.

Syntax

```
.param HSIMNVBS = value
```

Description

The grid resolution for V_{gs} voltage is equal to the V_{gs} voltage range divided by the value specified by HSIMNVGS on page 104, which is the number of grids in V_{gs} dimension. Similarly, HSIMNVDS on page 103 and HSIMNVBS on page 103 specify the grid numbers in V_{ds} and V_{bs} dimension, respectively. The default is 200.

Note:

HSIMVGSEND on page 166, HSIMVDSEND on page 165, and HSIMVBSEND on page 164 can also be included in each individual MOSFET model to achieve flexible control.

HSIMNVDS

Specifies the grid numbers in the VDS dimension.

Syntax

```
.param HSIMNVDS = value
```

Description

You can use `HSIMNVDS` to specify the grid numbers in VDS dimension. The default is 60.

HSIMNVGS

Specifies grid numbers in the VGS dimension.

Syntax

```
.param HSIMNVGS = value
```

Description

You can use `HSIMNVGS` to specify grid numbers in VGS dimension. The default is 60.

HSIMOPCOMPRESS

Specifies the operating point output file format.

Syntax

```
.param HSIMOPCOMPRESS = <0|1|2>
```

Arguments

Argument	Description
`0`	Specifies regular text format (.ic extension, which is the default setting).
`1`	Specifies .gz format.
`2`	Specifies .Z format.

Description

When .op statement is specified in the netlist, HSIM prints each node voltage at the specified time into a file. This command specifies the format of the output file.

The prefix of the output file is `hsim` by default, but can also be specified on the command line by entering -o file_name. The default value of time is 0 ns.

Note:

To dump the element currents at an operating point, set `HSIMDUMPOPI = 1`. The default is 0.

HSIMOPTSEARCHEXT

Specifies the file extension used in the .OPTIONS SEARCH (5-47) statement.

Syntax

`.param HSIMOPTSEARCHEXT = extension_string`

Description

`HSIMOPTSEARCHEXT` specifies the file extension used in the .OPTIONS SEARCH (5-47) statement. During the subckt/macro file search, it looks for the <cell>.<ext> file in the specified search paths. The default extensions are inc and sp. For example, param `HSIMOPTSEARCHEXT= cir`. If the cell name it looks for is AND2, it looks for file AND2.cir in the search paths.

HSIMOUTPUT

Specifies the waveform output format.

Syntax

`.param HSIMOUTPUT = format`

Description

Valid values are: fsdb (the default), nassda, and out.

HSIMOUTPUTFLUSH

Forces simulation data from memory to the disk file according to the specified time period.

Syntax

`.param HSIMOUTPUTFLUSH = time_period_value`

Description

While running the simulation, HSIM keeps the simulation result in a memory buffer. When the memory buffer is full, HSIM transfers the simulation result

data from the memory buffer to the disk file and frees up the memory buffer for the up-coming simulation result data. `HSIMOUTPUTFLUSH` changes this behavior by forcing HSIM to transfer the simulation result data to the disk file time for every time period specified by `HSIMOUTPUTFLUSH`, even before the memory is filled. It ensures that HSIM flushes the result based on the memory buffer capacity. The default is -1, which ensures that HSIM flushes the result based on the memory buffer capacity. For example:`HSIMOUTPUTFLUSH=200n`

HSIMOUTPUTFSDBSIZE

Controls the output size of the FSDB output file.

Syntax

`.param HSIMOUTPUTFSDBSIZE = size_value`

Description

`HSIMOUTPUTFSDBSIZE` controls the output size of the FSDB output file. The default value provides no splitting.`HSIMOUTPUTFSDBSIZE = 1` means split the FSDB file every 1 MB. `HSIMOUTPUTFSDBSIZE = 200` means split the FSDB file every 200 MB.

HSIMOUTPUTIRES

Controls the waveform output resolution.

Syntax

`.param HSIMOUTPUTIRES = output_resolution_value`

Description

To control the waveform output file size, HSIM does not write a new waveform data point for a given signal unless the value of the signal has changed from the last written data point by more than the output resolution. The output resolution is controlled by `HSIMOUTPUVRES` for voltage signals and `HSIMOUTPUTIRES` for current signals. The default is 1E-9 (1 nA).

HSIMOUTPUTMEAS

Writes the signals referenced in .meas to the output file.

Syntax

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

Description

Specifies whether (1, the default) or not (0) the signals referenced in .meas are output to the fsdb output file.

HSIMOUTPUTTBL

Creates waveform output files in csdf or raw formats.

Syntax

`.param HSIMOUTPUTTBL = format`

Description

`HSIMOUTPUTTBL` allows the creation of output waveform file in the SPICE raw and csdf formats. These files are created in post-processing after the simulation completes. Set *format* to `csdf` for csdf format or `rawfile` for raw format.

HSIMOUTPUTTRES

Controls the output time resolution.

Syntax

`.param HSIMOUTPUTRES =value`

Description

The `HSIMOUTPUTRES` value cannot exceed 1000 times the `HSIMTIMESCALE` value. Note that when you use `HSIMOUTPUTRES` with a value that is aggressively greater than the simulation time step, which is confined by `HSIMOUTAUMAX`, the resulting waveforms might be distorted. The default value is 1E-12 (1ps).

HSIMOUTPUTTSTEP

Sets the output result format to be a fixed time step.

Syntax

`.param HSIMOUTPUTSTEP =value`

Description

The default for `HSIMOUTPUTSTEP` is not set. The result format is in value changed form. When `HSIMOUTPUTSTEP` is set to a certain time value, such as 1ns or 100ps, the result format is in fixed time steps by every 1ns or 100ps repetitively.

HSIMOUTPUTVRES

Controls output voltage resolution.

Syntax

`.param HSIMOUTPUTVRES =`*`output_resolution_value`*

Description

To control the waveform output file size, HSIM does not write a new waveform data point for a given signal unless the value of the signal has changed from the last written data point by more than the output resolution. The output resolution is controlled by `HSIMOUTPUVRES` for voltage signals and `HSIMOUTPUTIRES` for current signals. The default is 1E-3 (1 mV).

HSIMOUTPUTWDFSIZE

Controls the output size of the WDF output file.

Syntax

`.param HSIMOUTPUTWDFSIZE =`*`value`*

Description

The default value (-1) for `HSIMOUTPUTWDFSIZE` specifies no splitting. `HSIMOUTPUTWDFSIZE = 1` means splits the WDF file every 1 MB.`HSIMOUTPUTWDFSIZE = 200` means splits the WDF file every 200 MB and so on.

HSIMPARPRECISION

Controls the coarseness or aggressiveness of partitioning. For details about the `HSIMPARPRECISION` settings, see HSIM^{plus} Quick Start on page 12.

Syntax

```
.param HSIMPARPRECISION = <0|1|2|3|4>
```

Description

`HSIMPARPRECISION` is a macro command. It controls the coarseness or aggressiveness of partitioning. Simulators use partitioning to handle large designs and by breaking a large mathematical problem into smaller, more manageable matrices. Without this approach, all simulators would have the same size limitations as traditional SPICE. If this algorithmic approach is done appropriately, performance will increase with minimal impact on precision.

HSIMPORTCR

Controls the subcircuit capacitance ratio for isomorphic matching.

Syntax

```
.param HSIMPORTCR = tolerance_value
```

Description

HSIM matches any two identical subcircuits (or partitions) that have similar port voltages and capacitive loading (within specified tolerances). If the port loading difference among partitions are within the tolerance value specified by `HSIMPORTCR` (default 0.01 or 1%), all such partitions are matched and share simulation results.

Example

```
.param HSIMPORTCR=0.05
```

To determine isomorphic matching, use a 5% tolerance for port capacitance ratio.

HSIMPORTI

Defines the port current tolerance.

Syntax

```
.param HSIMPORTI =value
```

Description

The port current tolerance is defined such that two corresponding port currents are considered matched if their difference is less than the value specified by

`HSIMPORTI`. By default, HSIM only considers port voltage tolerance. However, to obtain accurate currents through small resistors and zero volt floating voltage sources, you should specify a `HSIMPORTI` value. The unit is in Ampere.

HSIMPORTV

Specifies the port voltage tolerance value.

Syntax

```
.param HSIMPORTV =value
```

Description

Voltage drops across parasitic resistors generally result in different port voltages at identical subcircuits connected in parallel. For example, the voltage at the memory core cell port connecting to the bit-line is different for each memory cell connecting to the same bit-line when a parasitic resistor exists between any two neighboring memory core cells. The port voltages are otherwise identical in the absence of resistors.

If the port voltage difference exceeds the tolerance value specified by `HSIMPORTV`, each subcircuit requires separate calculations. If it does not exceed the tolerance voltage, calculated results can be shared among subcircuits with port voltage difference being less than the tolerance value specified by `HSIMPORTV`. The default is 1 mV.

To provide additional speed-up for post-layout simulation, `HSIMPOSL = 3` relaxes the port voltage tolerance from 1 mV to 10 mV.

HSIMPOSTL

Controls the RC reduction method.

Syntax

```
.param HSIMPOSTL = <0|1|2|3>
```

Arguments

Argument	Description
`0`	Turns off RC reductions (the default).
`1`	Activates the RC reduction method.
`2`	This setting is recommended when: ▪ The circuit contains nets that connect to a large number of MOSFET drain or source terminals. ▪ Current flow direction in these transistors is primarily bidirectional.
`3`	To speed up post-layout simulation, this setting relaxes the port voltage tolerance from 1 mV to 10 mV.

Description

There is no RC reduction when the default setting `HSIMPOSTL=0` is used. If you specify this value, HSIM checks every resistor in RC network, including the SPF RC network against the rmin value. If all the resistors are greater than rmin, no processing is performed. If at least one resistor is smaller than rmin, the whole RC network is processed for parallel reduction and rmin: rmin=max(HSIMRMIN, HSIMVSRCRMIN). Note that this calculation is processed no matter which RC network is connected to VSRC nodes.

When `HSIMPOSTL = 1`is set, the RC reduction method is activated. HSIM automatically reduces the RC network in each signal net to a smaller equivalent RC network. The reduced network contains the following elements:

- Resistors
- Capacitors
- Multiterminal control sources

The reduced network has nearly identical time-domain circuit behavior as the unreduced network. You can also set `HSIMPOSTL`on a subcircuit level.

The RC reduction algorithm set by `HSIMPOSTL = 1` handles parasitic RCs in the signal nets where the loading gates have coupled capacitance to the net through the transistor gate terminals such as a clock tree. In cases where most transistors are connected to the net through their drain or source terminals, the traditional RC reduction algorithm is not effective because the transistor current

direction could flow either way. Such is the case of bit-line in a memory core. The pass transistor connected between a bit-line and a core cell capacitor has a different current direction in the read or write cycle. A proprietary algorithm is implemented to reduce parasitic RCs in nets connecting to MOSFET drain or source terminals.

HSIMPOSTLMANY2M

Enables a more conservative reduction algorithm.

Syntax

```
.param HSIMPOSTLMANY2M = <0|1>
```

Description

Works in conjunction with `HSIMPOSTL` on nets with many-to-many connections and enables fine tuning for application specific reduction. Specify `HSIMPOSTLMANY2M = 1` to enable a more conservative reduction algorithm, applied for RC networks with many-to-many connections. Examples of nets with many-to-many connections are memory bitline and clock tree mesh nets (with many drivers to many receivers). The default is 0.

HSIMPOSTLONE2M

Enables a more conservative RC reduction algorithm.

Syntax

```
.param HSIMPOSTLONE2M = <0|1>
```

Description

Works in conjunction with `HSIMPOSTL` on nets with one-to-many connections and enables fine tuning for application specific reduction. Set `HSIMPOSTONE2M = 1` to enable a more conservative reduction algorithm, applied for RC networks with one-to-many connections. Examples of nets with one-to-many connections are memory wordline and clock tree nets (with one driver to many receivers). The default is 0.

HSIMPREFERVERILOGA

Specifies the priority order to search names

Syntax

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

Description

Whenever there is a clash between the names of subcircuits, C-modes, and Verilog-A model names in an HSIM netlist, HSIM resolves the names in the following order:

1. Subcircuit name.
2. C-model name.
3. Verilog-A model name.

For each X instance in the HSIM netlist, HSIM first attempts to match the name with the subcircuit name. If the subcircuit name is not matched, HSIM tries to find a C-model with the same name. Verilog-A models with that name are searched last.

To increase the Verilog-A priority with respect to other model types in HSIM, use the following syntax for `HSIMPREFERVERILOGA`:

```
.param HSIMPREFERVERILOGA=1
```

When `HSIMPREFERVERILOGA` is specified, the Verilog-A model name option supersedes all other options and the order changes as follows:

1. Verilog-A model name.
2. Subcircuit name.
3. C-model name.

HSIMPREFLAT

Flattens the netlist before partitioning.

Syntax

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

Description

`HSIMPREFLAT=1` is equivalent to reading in a flat netlist. This command is valuable in some cases of post-layout simulation where the parasitic resistors and capacitors (RC)s are defined within the subcircuits.

Note:

This command is not recommended for large circuit simulation due to high memory usage considerations.

HSIMPRINTSIMSTATUS

Controls the simulation time refresh rate.

Syntax

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

Description

`HSIMPRINTSIMSTATUS` controls the rate at which simulation time is refreshed. The default (1) is to refresh simulation status on-screen every nanosecond.

With long simulation times, especially when a simulation is running in the background, this refresh rate is typically too short. To refresh the screen every 10% of the total simulation time, specify a value of 0.

HSIMRASPFMODXY

Calculates bounding box data.

Syntax

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

Description

`HSIMRASPMODXY` is used when the extractor generates bounding box data instead of true node coordinates. If it is set to 1, the coordinates of the bounding box centers are calculated and used for visualization of resistor connectivity.

HSIMRASTART

Controls the starting time for the second phase RA simulation.

Syntax

```
.param HSIMRASTART = time
```

Description

This command is equivalent to the `TSTART` RA command, where `TSTART` is specified in the RA Tcl command file. The difference is that `HSIMRATSTART` is parsed and processed in the first phase simulation.

HSIMRASTOP

Controls the stop time for the second phase RA simulation.

Syntax

```
.param HSIMRASTOP = time
```

Description

This command is equivalent to the `TSTOP` RA command, where `TSTOP` is specified in the RA Tcl command file. The difference is that `HSIMRATSTOP` is parsed and processed in the first phase simulation.

HSIMRCNPO

Controls the hierarchical RC reduction process.

Syntax

```
.param HSIMRCNPO = <0|1|2>
```

Arguments

Argument	Description
`0`	Performs RC reduction independently in each subcircuit (the default). Even if there is a connection between two RC networks located in different subcircuit instances, they are still treated as two separate RC networks.
`1`	Performs RC reduction hierarchically, causing all of the connections between RC elements to be taken into account, including the connections between elements located in different subcircuit instances. This method is similar to selective flattening of RC networks without flattening the containing subcircuits. When necessary, RC networks located at lower levels of hierarchy are pulled out of their subcircuits and moved to higher levels of hierarchy.
`2`	Only the resistor elements (R-networks) are pulled out before reduction. After the R-networks are processed, capacitor reduction is performed as it is when `HSIMRCNPO = 0`.

Description

For RC reduction, `HSIMRCNPO` has the same effect as `HSIMPREFLAT` however, `HSIMPREFLAT` flattens entire subcircuits, which can be extremely inefficient in terms of speed and memory consumption compared to the selective RC network pull-out. `HSIMRCNPO` is a global parameter.

HSIMRCPRECISION

Controls the degree of reduction applied to designs. For details about the `HSIMRCPRECISION` settings, see HSIM^{plus} Quick Start on page 12.

Syntax

```
.param HSIMRCPRECISION = <-1|0|1|2|3|4|5>
```

Description

`HSIMRCPRECISION` is a macro that controls the degree of reduction applied. Reduction is required for large designs with back- annotated layout parasitics. `HSIMRCPRECISION` determines the precision and performance trade-off associated with the different methods of reduction available to HSIM. Some of these methods include serial, parallel, and moment based reduction.

HSIMRCRIO

Controls how to preserve nodes specified in .ic/.nodeset/.print statements.

Syntax

```
.param HSIMRCRIO = <0|1|2>
```

Arguments

Argument	Description
0	Keeps track of where a node is merged to when it is reduced during RC reduction. This ensures that input and output (I/O) statements such as .ic/.nodeset/.print can still apply.
1	If an improvement of precision for internal RC nodes is desired, specify this value to preserve all nodes specified in .ic/.nodeset/.print statements.
2	Set this values to preserve the nodes with .ic/.nodeset/ .print and keep the adjacent RLC elements.

Description

The commands and parameters that preserve subcircuit nodes and elements during RC reduction are shown in Table 3.

Table 3 Commands that Preserve Subcircuit Node/Elements During RC Reduction

Parameter	Description
.measure	All nodes and RLC elements specified in .measure statements.

Table 3 Commands that Preserve Subcircuit Node/Elements During RC Reduction (Continued)

Parameter	Description
VEC CHECK	All nodes specified in VEC CHECK.
$MODEL=model_name	All RLC elements with $MODEL=model_name specified and model_name is specified in parameter HSIMRCRKEEPMODEL.

HSIMRCRKEEPELEM

Preserves elements in RC reduction if they match the specified pattern.

Syntax

`.param HSIMRCRKEEPELEM = element_pattern`

Description

`HSIMRCRKEEPELEM` can be used to specify an element pattern. All the elements that match the pattern are preserved in RC reduction.

HSIMRCRKEEPMODEL

Preserves the specified model in RC reduction.

Syntax

`.param HSIMRCRKEEPMODEL = model_name`

Description

`HSIMRCRKEEPMODEL` preserves the specified model name in the `$MODEL` statement. See the following example.

Example

In the following example, R1 is preserved during the RC reduction because the R1 description includes rrb in the $MODEL assignment and rrb is specified in an `HSIMRCRKEEPMODEL` statement. A similar situation applies to C2.

```
.param hsimrcrkeepmodel=rrb
.param hsimrcrkeepmodel=rrc

R1   1   2   100   $MODEL=rrb
C2   2   3   15f   $MODEL=rrc
...
```

HSIMRCRKEEPNODE

Preserves nodes in RC reduction if they match the specified pattern.

Syntax

`HSIMRCRKEEPMODE = node_pattern`

Description

`HSIMRCRKEEPNODE` can specify a node pattern such that all the nodes that match the pattern and the RLC elements directly connected to the nodeare preserved in RC reduction.

HSIMRCRTAU

Controls the accuracy of the RC reduction solver.

Syntax

`HSIMRCRTAU = value`

Description

Controls the accuracy of the RC reduction solver and defines a cutoff frequency for RC networks during reduction. The default value is 1.0e-10s. It is not recommended to set it higher than default value.

The only time to use `HSIMRCRTAU` for a more accurate RC reduction is when `HSIMPOSTL = 1`. If you need better accuracy, set `HSIMRCRTAU` to a smaller value. While you might not see a big difference in every design, this is the correct way to use `HSIMRCRTAU` and `HSIMPOSTL`. Use either the default `HSIMRCRTAU` value or set it to a smaller value such as 5ps.

When `HSIMPOSTL = 0`(the default), there is no RC reduction, so `HSIMRCRTAU` HSIMRCRTAU is irrelevant. When `HSIMPOSTL = 2`, RC reduction is relatively aggressive, so it does not make sense to use `HSIMPOSTL = 2` with `HSIMRCRTAU`.

HSIMREDEFSUB

Allows a simulation to continue when the are duplicate subcircuit names.

Syntax

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

Description

When you specify a value of 1 for this command in a simulation with duplicate subcircuit names, the last subcircuit definition is selected and the previous definition is ignored.

Example

The following netlist has two subcircuit definitions on `inv`:

```
.subckt inv a b
...
.ends
.subckt inv p q r
...
.ends
```

HSIM selects the second subcircuit definition for subcircuit `inv` if:

- The syntax statement shown below is present in the netlist
- It is located at the top-level and outside of any subcircuit definition.

```
.param HSIMREDEFSUB=1
```

HSIMREGNODE

Improves performance and reduces memory usage when simulating designs with voltage-regulated power nodes.

Syntax

```
.param HSIMREGNODE = "node=node_type...
   input=regnode_drivers...
   [subckt=subcircuit_name...]|[inst=instance_name...]"
```

Arguments

Argument	Description
`node=`*`node_type`*	Specify one of the following node types: ▪ Full hiearchical node name. ▪ Subcircuit port name, which uses the `subckt=`*`subcircuit_name`* argument.
`input=`*`regnode_drivers`*	Specifies one of the following instance type. ▪ Full hiearchical instances (transistors or instances) that drive the regnode when you specify a full hiearchical node name. ▪ Subcircuit instances (transistors or instances) that drive the regnode when you use the `subckt=`*`subcircuit_name`* argument. Separate the list of regnodes with a space.
`subckt=`*`subcircuit_name`*	Specifies the subcircuit name when `HSIMREGNODE` is applied to a subcircuit.
`inst=`*`instance_name`*	Specifies the instance name when `HSIMREGNODE` is applied to an instance.

Description

In pre-layout simulations, `HSIMREGNODE` applies the similar decoupled nature of designs with constant voltage sources to designs with voltage-regulated power nodes. The result is better performance and improved memory efficiency simulating this type of design.

Note:

You cannot use both the `inst` and `subckt` arguments in the same `HSIMREGNODE` command.

Example

```
.param HSIMREGNODE="node=xtop.vddinternal
input=xtop.xio.x1.xhd_switch"
```

Applies `HSIMREGNODE` to the `xtop.xio.x1.xhd_switch`driver and its `vddinternal` port.

```
.param HSIMREGNODE=" node=vddinternal input=xhd_switch
inst=xtop.xi0.x1"
```

HSIM searches the `xtop.xio.x1` instance and looks for its `xhd_switch` internal instance and the related `vddinternal` port to apply `HSIMREGNODE`.

```
.param HSIMREGNODE=" node=vddinternal input=xi0 xi1
subckt=sram_head_switch_10"
```

HSIM searches the `sram_head_switch_10` subcircuit and looks for its `xio` and `xi1` internal instances and the related `vddinternal` port to apply `HSIMREGNODE`.

HSIMREPORTAREA

Reports the total transistor gate area.

Syntax

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

Description

When `HSIMREPORTAREA=1` a reported is printed that shows the length, width, and gate area. The default is 0.

Example

```
Total MOS length : 3.6e-06
Total MOS width : 0.000704
Total MOS gate area : 1.2672e-10
************************************
```

HSIMRGATEMOD

Models gate load resistance.

Syntax

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

Description

Set `HSIMRGATEMOD = 1` to model constant gate load resistance. The default is 0.

HSIMRMIN

Sets the minimum resistor value.

Syntax

```
.param HSIMRMIN = value
```

Description

To eliminate small resistors, this command sets the minimum resistor value allowed in simulation. All resistors with a value smaller than `HSIMRMIN` are short-circuited. Since it is performed before RC reduction, eliminating small resistors also helps give a more effective RC reduction with a very minor loss of precision. `HSIMRMIN` can only be set globally. The default value is 0.1

HSIMRMVDIO

Removes voltage clipping diodes.

Syntax

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

Description

`HSIMRMVDIO` removes voltage clipping diodes connected to voltage sources. The default is 0 to not remove them. However, in some cases you can use 1 to improve performance.

HSIMSAMPLERATE

Determines how long the sine wave should trigger the input node.

Syntax

```
.param HSIMRSAMPLERATE = value
```

Description

For sine voltage sources, HSIM needs to determine how long the sine wave should trigger the input node. The default value of `HSIMSAMPLE = 160` means that for each cycle there are 160 triggering points.

Example

In the following example every 3ps the input sine wave triggers the input node. For a sine wave input with a 2 GHz frequency:

```
one cycle=1/2GHz=500ps
500ps/160 points=~3ps/point
```

HSIMSCALE

Sets the scale factor for the specified subcircuit.

Syntax

```
.param HSIMRSSCALE = value
```

Description

The HSIMSCALE command only provides scaling only for the specified subcircuit. Its uses the same rules as the .option scale command described in .option on page 296.

Example

```
.hsimparam subckt=need_to_scale HSIMSCALE=1.e-6
```

HSIMSNCS

Bypasses idle subcircuit elements to speed up simulation.

Syntax

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

Description

Using circuit latency may cause the simulation to run faster. Instead of simulating every circuit element at every time step, setting `HSIMSNCS = 1` (the default) bypasses those subcircuit elements at the time when those elements are idle. A subcircuit is treated as being idle if every node in the subcircuit is idle.

Table 4 contains the `HSIMSNCS` parameter descriptions.

Table 4 HSIMSNCS Node Voltage Parameters

Parameter	Description
vn+1, vn	Node voltages at two consecutive time points
M	Tolerance defined by M=h*Itolerance/C
h	Time stepsize
C	Node capacitance
Itolerance	Steady current tolerance defined by HSIMSTEADYCURRENT.

The latency setting using `HSIMSNCS = 1` should not cause a precision problem if the circuit functionality is not affected by neglecting the small current defined by `HSIMSTEADYCURRENT`.

The default value of `HSIMSNCS = 1` can only be set at the top level, so it has a global effect on the entire circuit.

Example

The criterion for determining if a node is idle is by checking the node voltage change:

```
|Vn+1 - Vn| < M
```

HSIMSPEED

Sets the simulation speed in relation to precision.

Syntax

```
.param HSIMRSPEED = <0|1|2|3|4|5|6|7|8>
```

Arguments

Argument	Description
`0`	Set this value when the simulation result is expected to closely match with the SPICE result. This value is recommended only for simulating small analog circuits with a circuit size less than 1,000 elements.
`1`	Significantly speeds up simulation compared to the value of 0 is achieved while the precision loss is minimal, especially for large circuits. Significant precision loss may occur when setting HSIMSPEED on page 125 `= 1` in two applications: ▪ Simulating leakage currents ▪ Simulating circuits with very low power supply voltage Accurate simulation result can be achieved by using HSIMSPEED on page 125 in the case of leakage current simulation, setting HSIMSTEADYCURRENT on page 144 to about 1% of the expected leakage current value is recommended when the HSIMSPEED on page 125 value is greater than 0. Note that when simulating circuits with low power supply voltage, setting HSIMALLOWEDDV on page 47 to about 10% of the power supply voltage is recommended.
`2`	With this value a further speedup is achieved by flattening selected circuit hierarchies, which reduces some hierarchical simulation overhead. The selection of a subcircuit for flattening is determined by evaluating the probability of its subcircuit instances to share the same simulation result. There should be little precision loss in setting a value of 2 versus the setting of 1, but memory usage could increase.

Argument	Description
3	The default value of 3 provides additional speed-up through the multirate time stepsize selection. When HSIMSPEED on page 125 is set to less than 3, the time stepsize selection is uniform such that a common stepsize is selected for every subcircuit. When the HSIMSPEED on page 125 value is greater than or equal to 3, different time stepsizes are selected in different subcircuits, such that a smaller time step is selected for subcircuits with faster transient activities, while a larger time step is selected for subcircuits with slow transient activities. This speed-up in multirate step selection is achieved by reducing the total number of time steps. The precision loss in setting in HSIMSPEED on page 125 compared with lower values is significant only in very few applications such as simulating high-sensitivity analog circuits. However, it is generally not necessary to lower theHSIMSPEED on page 125 value: the HSIMANALOG on page 48control parameter achieves analog simulation speed without sacrificing precision.
4	This value achieves an increased speedup compared to HSIMSPEED on page 125 by relaxing the precision resolution. A value of 4 is recommended for simulating digital circuits, memory circuits, or analog circuits with lower sensitivity. In such applications, the precision difference between the values of 3 and 4 is minimal.
5	This value is recommended for functional simulation, including memory and mixed-signal circuits. In this case, simulation speed-up is achieved by sacrificing timing delay precision. In general, 5-10% of timing delay error is expected when using this high-speed simulation mode.
6	This value was formerly used for functional simulation of digital circuits and low sensitivity analog circuits. However, it is now recommended that HSIMSPEED on page 125 . Note: HSIMSPEED on page 125=6 is now obsolete.

Argument	Description
7	This value is recommended for functional simulation of digital circuits or low sensitivity analog circuits. It is optimized for circuits such as ROM and CAM.
8	This value is recommended for functional simulation optimized for Flash memory and mixed-signal circuits. There are three submodes: HSIMSPEED on page 125=8.4, HSIMSPEED on page 125=8.7 and HSIMSPEED on page 125=8.8 to take advantage of circuit latency in large circuit blocks to speed up simulation. These submodes are especially useful in circuits that contain voltage regulators or charge pumps. Note: The interactive tree command only works in HSIMSPEED on page 125=8.7.

Description

Higher HSIMSPEED on page 125 values cause faster simulation speed at reduced simulation precision. HSIMSPEED on page 125 can be selectively set in local subcircuits such that:

- Lower HSIMSPEED on page 125 values are set for either of the following:
  - Analog subcircuits
  - Timing critical subcircuits
- Higher HSIMSPEED on page 125 values can be set for digital subcircuits.

  Caution!

  The performance of local HSIMSPEED on page 125setting may still rely on the global HSIMSPEED on page 125 value set at the top level.

Some parameters that are automatically controlled by HSIMSPEED on page 125 value cannot be locally set in subcircuits. They follow the global HSIMSPEED on page 125 value.

Example

When setting HSIMSPEED on page 125 in a local subcircuit while setting HSIMSPEED on page 125=5 in the top-level, the value of HSIMIDDQ on page 86 remains at the value of 1 in this subcircuit: HSIMSPEED on page 125 HSIMIDDQ on page 86 can only be globally set. This global setting is determined by the top-level setting of HSIMSPEED on page 125 =5.

HSIMSPF, HSIMSPEF

Specifies the DSPF and SPEF file names.

Syntax

`.param HSIMRSPF = file_name` and `.param HSIMRSPF = file_name`

Description

`HSIMSPF` defines the defines the DSPF file for back-annotation. `HSIMSPEF` defines the SPEF file for back-annotation. In the following example, file1 contains parasitic RC in DSPF format, and file2 contains parasitic RC in SPEF format, and they are both back-annotated onto the transistor netlist.

HSIMSPFADDNETPINXY

Lets you insert a "*|P" net pin by specifying XY coord inates.

Syntax

```
.param HSIMSPFADDNETPINXY = "net X=xcoord Y=ycoord
   layer=malyer pinxydist=xydist"
```

Arguments

Argument	Description
net	Specifies the net name.
X=*xcoord*	X coordinate (in microns).
Y=*ycoord*	Y coordinate (in microns).

Description

This command finds the closest location to the specified XY coordinates on defined metal layer. It uses the following formula 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

HSIMSPF

Specifies the back-annotation file formats.

Syntax

```
.param HSIMSPF=file1 HSIMSPEF=file2
```

Description

`HSIMSPF` defines the DSPF file for back-annotation. `HSIMSPEF` defines the SPEF file for back-annotation. In the following example, file1 contains parasitic RC in DSPF format, file2 contains parasitic RC in SPEF format, and they are both back-annotated onto the transistor netlist.

HSIMSPFCC

Turns on the processing of coupling capacitors in DSPF/SPEF files.

Syntax

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

Description

This command turns on the processing of coupling capacitors from DSPF/SPEF file. The default is 1 unless `HSIMPWRA` is set to 1. Then `HSIMSPFCC` defaults to 0.

HSIMSPFCCSCALE

Specifies the scale factor for coupling capacitors.

Syntax

```
.param HSIMSPFCCSCALE = scale_value
```

Description

Specifies the scale factor value by which to multiply the value of each coupling capacitor. When coupling capacitors are processed from DSPF/SPEF, the value of each coupling capacitor is multiplied by the `HSIMSPFCCSCALE` value. The default is 1.

HSIMSPFCHLEVEL

Specifies the hierarchical level to back annotate capacitance.

Syntax

```
.param HSIMSPFCHLEVEL = level
```

Description

For all nets at a level in the hierarchy below the `HSIMSPFCHLEVEL` value only capacitance is back-annotated. For nets that are at a level above or equal to the specified level, RC back-annotation is performed.

HSIMSPFCMIN

Controls whether to perform RC or capacitance only back annotation.

Syntax

```
.param HSIMSPFCMIN = value
```

Description

For all nets with capacitance larger than or equal to the `HSIMSPFCMIN` value, RC back-annotation is performed. For all nets with capacitance smaller than the specified value, only the capacitance is back-annotated.

HSIMSPFCNET

Back annotates capacitance only by net name.

Syntax

```
.param HSIMSPFCNET = net_name
```

Description

For all net names matching the net name specified in `HSIMSPFCNET`, only capacitance is back-annotated. You can specify a wildcard character in the net name. The name must be quoted if wildcard characters are used. This command can be specified multiple times.

Example

```
.param HSIMSPFCNET="x1/*"
```

All the nets matching “x1/*” have capacitance-only back-annotation and all the other nets have full RC back-annotation.

```
.param HSIMSPFCNET=”x1/*” HSIMSPFRCNET=” “
```

All the nets matching “x1/*” have capacitance-only back-annotation. All other nets have no annotation.

HSIMSPFDSJNET

Controls the reporting detail for disjointed components.

Syntax

```
.param HSIMSPFDSJNET = <0|1|2>
```

Arguments

Argument	Description
0	Generates no report (the default).
1	Reports pins only.
2	Compiles a complete report including elements and pins on each disjointed component.

Description

This command controls the reporting of elements and pins for disjointed components according to the value you specify.

HSIMSPFFDELIM

Specifies SPF delimiter characters.

Syntax

```
.param HSIMSPFFDELIM = delimiter_character
```

Description

In SPEF, `HSIMSPFFDELIM` specifies delimiter character used on fingered devices.

HSIMSPFFEEDTHRU

Suppresses the merging of net ports.

Syntax

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

Arguments

Argument	Description
0	Merges net ports (the default).
1	Does not merge net ports, and the DSPF file for a low-level subcircuit remains the same.

Description

Merging the net ports into one node may introduce some small inaccuracies. To separate the net ports set `HSIMSPFFEEDTHRU` to 1. The top-level subcircuits have the syntax shown in the following example.

Example

```
*|NET VDD 10ff
*|P (VDD X 0.0)
*|I (X1:VDD X1 VDD X 0ff)
*|I (X1:VDD_1 X1 VDD_1 X 0ff)
*|I (X1:VDD_2 X1 VDD_2 X 0ff)
*|I (X2:VDD X2 VDD X 0ff)
*|I (X2:VDD_1 X2 VDD_1 X 0ff)
*|I (X2:VDD_2 X2 VDD_2 X 0ff)
R1 VDD X1:VDD 10.0
R2 VDD X2:VDD 10.0
R3 VDD X1:VDD_1 10.0
R4 VDD X2:VDD_1 10.0
R5 VDD X1:VDD_2 10.0
R6 VDD X2:VDD_2 10.0
```

After back-annotating the low-level subcircuit cell, the following two additional ports are added to the subcircuit.

- VDD_1
- VDD_2

When HSIM back-annotates the top-level subcircuit without port merging, all the instance pins are correctly matched with those of the top-level DSPF file as follows:

- X1:VDD, X1:VDD_1, X1:VDD_2
- X2:VDD, X2:VDD_1, X2:VDD_2

It is important that pin names, such as VDD_2 in the previous example, are defined to match the instance pin names one level up in the hierarchy. Otherwise, connections are not be made by the simulator. HSIM uses the following parameters to control back-annotation for hierarchical DSPF for feed-through nets:

```
.param HSIMSPFFEEDTHRU=1
```

HSIMSPFHLEVEL

Controls RC back-annotation according to the hierarchy level.

Syntax

`.param HSIMSPFHLEVEL = hierarchy_level_value`

Description

For all nets at a level in the hierarchy above `HSIMSPFHLEVEL`, RC back-annotation is performed. For nets lower than `HSIMSPFHLEVEL`, only the capacitance is back-annotated.

HSIMSPFKEEPNODECAP

Merges pre-layout node capacitance with the value from the SPF file.

Syntax

`.param HSIMSPFFEEDTHRU = <0|1>`

Description

Specify `HSIMSPFKEEPNODECAP = 1` to merge pre-layout node capacitance with the value from the SPF file. By default, HSIM overwrites the pre-layout net capacitance with what is contained in the DSPF netlist.

HSIMSPFKS

Keeps backslash characters (\) in names in the DSPF/SPEF file.

Syntax

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

Description

This command controls whether to keep backslashes (\) in names in DSPF/SPEF file. The default is 0, which removes all of these characters.

HSIMSPFMERGEFDEV

Controls how fingered devices in DPF file are processed.

Syntax

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

Arguments

Argument	Description
0	Splits a device in the pre-layout netlist to match the number of fingered devices in the SPF/SPEF file before back-annotation.
1	Shorts all fingered devices during back-annotation.

Description

This command controls how fingered devices in DPF file are processed. The default is 1.

HSIMSPFMSGLEVEL

Controls the printing of error and warning messages.

Syntax

```
.param HSIMSPFSGLEVEL = <0|1|2|3>
```

Arguments

Argument	Description
`0`	No net error or warning messages are printed.
`1`	Error messages are printed for discarded nets.
`2`	Error and warning messages are printed for discarded nets.
`3`	Error and warning messages are printed for all nets.

Description

An error is defined as a connectivity problem that causes the net to be discarded. A warning is a connectivity problem that can be corrected. A net can have both types of problems.

HSIMSPFMULTISUB

Allows reading multiple subcircuits in a single SPF file.

Syntax

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

Description

Specify `HSIMSPFMULTISUB = 1` to permit multiple .subckt statements in the SPF file. The default is 0.

HSIMSPFNETPINDEL

Converts DSPF net pins to internal nodes.

Syntax

`.param HSIMSPFNETPINDEL = net_name net_pin1 net_pin2 ...`

Description

Converts one or more selected DSPF net pins into the internal node(s) of the net.

HSIMSPFNETWARNFILTER

Suppresses printing floating nets in the log file.

Syntax

`.param HSIMSPFNETWARNFILTER = net_name_pattern`

Description

In the DSPF and SPEF files, there can be many floating nets which only exist in the layout. These nets are typically of the name ln_# where # is a number. It is desirable for these nets not to be listed in the .log file in order to reduce the number of warnings.

This command allows you to specify the name pattern of the floating nets, with wildcard character support, such that the nets are not listed. There are no warning message for the floating nets which match the name pattern(s). There are warning messages for all DSPF/SPEF nets which match the floating net pattern, but do not exist in the schematic netlist and are attached to the real instances.

HSIMSPFNOWARNONCAP

Controls whether or not to print a warning message when a SPF net cannot be found in the netlist.

Syntax

`.param HSIMSPFNOWARNONCAP = value`

Description

If a capacitance value is specified for `HSIMSPFWARNOCAP`, only those SPF nets with net capacitance greater than the specified value report a warning message when the nets do not exist. If you specify `HSIMSPFWARNOCAP = 0`, HSIM gives a warning for every net when the net does not exist. If you specify `HSIMSPFWARNOCAP = 1.e-13`, for example, HSIM only gives warning for the net with a capacitance value greater than 100 fF.

HSIMSPFPOS

Reads placement information from the specified file.

Syntax

```
.param HSIMSPFPOS = file_name
```

Description

This command reads in the position information from Star-RCXT- generated placement file. This information is required for the hierarchical instances in the hierarchically extracted netlists.

HSIMSPFPOSTL

Controls parasitic reduction.

Syntax

```
.param HSIMSPFPOSTL = <0|1|2|3>
```

Description

`HSIMSPFPOSTL` controls parasitic reduction just like HSIMPOSTL on page 110. The only difference is that `HSIMPOSTL` applies reduction only to back-annotated RCs.

HSIMSPFPRINTPIN

Allows printing both pre- and post-layout results.

Syntax

```
.param HSIMSPFPRINTPIN = <0|1|2|3>
```

Arguments

Argument	Description
0	Prints only the net node (the default).
1	Prints the top-level instance pin.
2	Prints hierarchical names.
3	Prints both the net node and top-level instance pins.

Description

HSIM permits printing of both pre- and post-layout results. In pre-layout, the node name is aliased to the net name. This is accomplished in the following syntax:

```
.param HSIMSPFPRINTPIN = 1
.print v(a)
```

In nWave, the format is:

```
v(a::xi7:i)
```

as shown in the following syntax:

```
<netname>::<inst>:<pin>
```

Example

In the following example, all instance pins are printed.

```
HXI7:I
.XI6-MM2:D
```

HSIMSPFRCNET

Runs back-annotation for the specified nets.

Syntax

```
.param HSIMSPFPFRCNET = net_name
```

Description

Runs back-annotation for all nets names that match the specific net name. You can use a wildcard character in the net name. The name must be quoted if wildcard characters are used. You can specify this command multiple times.

Example

```
.param HSIMSPFRCNET="x1/x2/*"
```

All the nets matching “x1/x2/*” run full RC back-annotation. All the remaining nets have capacitance-only back-annotation.

```
.param HSIMSPFRCNET="x1/*" HSIMSPFCNET=" "
```

All nets matching “x1/*” run full RC back-annotation. All other nets have no annotation.

```
.param HSIMSPFRCNET="x1/*" HSIMSPFCNET="x2/*"
```

All nets matching “x1/*” run full RC back-annotation. All nets matching “x2/*” will do capacitance-only back-annotation. All other nets have no annotation.

HSIMSPFREPORTNOBA

Controls back-annotation data reporting.

Syntax

```
.param HSIMSPFREPORTNOBA = <0|1|2|3>
```

Arguments

Argument	Description
0	No data reporting (the default).
1	Reports instances containing no back-annotation data. No nets inside the instances have SPF data.
2	Reports all nodes containing no SPF data.
3	Reports all nodes and ports containing no SPF data.

Description

Controls instance and node back-annotation data reporting.

HSIMSPFSKIPNET

Does not run back-annotation for specified nets.

Syntax

```
.param HSIMSPFSKIPNET = net_name
```

Description

Does not run back-annotation for the net names that match the specified net name You can use a wildcard character in the net name. The name must be quoted if wildcard characters are used. You can use this command multiple times.

HSIMSPFSKIPPWNET

Controls back-annotation of power nets.

Syntax

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

Description

Specify `HSIMSPFSKIPPWNET = 1` for power nets that are connected to constant voltage source not to be back-annotated. The default is 0.

HSIMSPFSKIPSIGNET

Controls back-annotation of signal nets.

Syntax

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

Description

Specify `HSIMSPFSKIPPWNET = 1` for signal nets that are connected to constant voltage source not to be back-annotated. The default is 0.

HSIMSPFSPLITCC

Splits coupling capacitors.

Syntax

```
.param HSIMSPFSPLITCC = <0|1>net_name
```

Description

Specify `HSIMSPITICC = 1` to split all coupling capacitors into two grounded capacitors with the same capacitance. The default is 0.

HSIMSPFEFTRIPLET

Sets the value for triplet analysis.

Syntax

```
.param HSIMSPEFTRIPLET = <1|2|3>
```

Arguments

Argument	Description
`1`	Best case.
`2`	Typical case.
`3`	Worst case.

Description

In SPEF, parasitic RC values can be specified as a triplet for best (1), typical (2), and worst case (3). HSIM uses one value in a triplet for the analysis. This option specifies which value in a triplet to use. The default is 2, which uses the 2nd value (typical) in the triplet.

HSIMSPFWARNFILE

Writes SPF-related warnings to a file.

Syntax

```
.param HSIMSPFWARNFILE = <0|1>net_name
```

Description

Set `HSIMSPFWARNFILE = 1` to redirect all SPF-related warnings to an <output_prefix>.spfwarn file. The default is 0.

HSIMSPICE

Controls simulation precision for MOSFETs.

Syntax

```
.param HSIMSPSPICE = <0|1|2|3>
```

Arguments

Argument	Description
`0`	This default value generates the Table Model during the HSIM preprocessing such that the efficient Table Model reduces the MOSFET model evaluation time during circuit simulation.
`1`	This value suppresses the usage of Table Model and directly evaluates the exact MOSFET DC Model equations to calculate the Ids current from the terminal node voltages.
`2`	This value further improves the simulation precision by employing the exact MOSFET Gate Capacitance Model equations, instead of the efficient model equations used when `HSIMSPICE` is set to either 0 or 1.
`3`	This setting precisely models and simulates the coupling effects between each pair of gate/drain/source/bulk terminals for each MOSFET affected by the `HSIMSPICE = 3` setting.

Description

When `HSIMSPICE = 0`, (the default) the Table Model in general causes little precision loss unless the circuit functionality is sensitive to the small voltage variation that has similar magnitude of the voltage grid used in the Table Model.

However, the Table Model only contains the DC data of MOSFET Ids current with respect to the terminal node voltages. The MOSFET gate capacitances are separately calculated by a set of efficient equations.

When `HSIMSPICE = 1`, the MOSFET gate capacitances are calculated in the same way as the setting of 0. In case of simulating high-sensitivity analog circuits, where the simulation precision is sensitive to the small difference in Ids current caused by linear interpolation between grid points of table data, it may require choosing a very small voltage grid resolution to improve the precision.

For information about voltage grid information, see MOSFET Table Model Control Parameters on page 320.

This value may be impractical due to a very large number of table data, which also requires a long time to generate in the preprocessing. Although the setting of 1 improves the precision, the simulation speed will inevitably slow down due to the more time-consuming analytical MOSFET Model evaluation. So this setting is suggested in simulating small and highly sensitive analog circuits, or in local subcircuit blocks where the simulation precision is sensitive to the Table Model grid resolution. When set locally, this value only covers the MOSFETs affected by the setting.

When `HSIMSPICE = 2`, the model equations for MOSFET Ids current are the same when `HSIMSPICE` is set to 1 or 2. For simulating circuits such as a charge-pump voltage generator, analog-to-digital (A/D), or digital-to-analog (D/A) converters, it is recommended that `HSIMSPICE = 2` be used.

When `HSIMSPICE = 2`, it includes the most precise setting includes all the models specified by `HSIMSPICE = 2`.

HSIMSPISCIUNITERR

Allows mixed syntax values in a simulation.

Syntax

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

Description

This command handles values that are mixed scientific notation and units. Mixed notation such as 1.0E-6F is recognized as 1.0E-21 however, this is an illegal SPICE syntax format. Set `HSIMSPISCIUNITERR =1` to cause HSIM to error out when parsing mixed syntax.

If mixed syntax is read by a DSPF file, it is converted as is; even if the `HSIMSPISCIUNITERR =1` is set.

HSIMSTEADYCURRENT

Skips the simulation of idle subcircuits.

Syntax

```
.param HSIMSTEADYCURRENT = value
```

Description

If `HSIMSNCS` is set to 1, the simulation of idle subcircuits is skipped to boost the simulation speed. A subcircuit is treated as idle if every node in the subcircuit is idle. The criterion to determine whether a node is idle is by checking the node voltage change shown in HSIMSNCS on page 124.

The higher the value of `HSIMSTEADYCRRENT`, the more likely a subcircuit becomes idle, which enables the simulation to run faster.

In HSIM, `HSIMSTEADYCRRENT`can be set in any local subcircuit, either subcircuit definition or subcircuit instance, such that it has a local effect on the associated subcircuit.The default value is 10nA.

HSIMSTEP2TAUMAX

Sets the the `tstep` value in the `.tran` command as specified by the `HSIMSTEPTAUMAX` value.

Syntax

```
.param HSIMSTEP2TAUMAX = <0|1>
```

Description

If `HSIMSTEP2TAUMAX=1`, the `tstep` value in the `.tran` command is used as the `HSIMTAUMAX` value. If multiple `tstep`/`tstop` pairs are specified in the `.tran` command, HSIM automatically sets the `HSIMTAUMAX` command as a PWC function of the step/stop time pairs specified in `.tran`. The default is 0.

Example

```
.tran 0.01ns 10ns 0.05ns 100ns
.param HSIMTSTEP2TAUMAX=1
=> .param HSIMTAUMAX=pwc(0ns, 0.01ns, 10ns, 0.05ns)
```

HSIMSTITOL

Sets a tolerance value to minimize memory usage.

Syntax

```
.param HSIMSTEADYCURRENT = value
```

Description

This command is used to set a tolerance to minimize the memory usage in STILOD stress so that devices with SA, SB and SD can share a common table

of tolerance values if other instance parameters are the same. The default value is 5.0e-9 meter.

HSIMSTOPIFNODC

Stops a simulation at the end of DC initialization.

Syntax

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

Description

Specify `HSIMSTOPIFNODC = 1` to stop the simulation at the end of DC initialization if no DC solution is found. If DC does not resolve all nodes, an error is reported. The default is 0.

HSIMSTNAME

Provides a mechanism to match terminal names in pre- and post-layout netlists.

Syntax

```
.param HSIMSTNAME  = "sub=sub_name pre_term1=post_term1
   pre_term2=post_term2 …..."
```

Description

This command is used when terminal names of a pre-layout subcircuit do not match the terminal names in the post-layout instance pin definition. It provides a mechanism for one-to-one mapping of terminal names for a particular subcircuit.

Example

```
Pre-layout netlist :

.subckt nmos drn gate src body
…….
.ends nmos

xi1 n1 n2 n3 n4 nmos

DSPF file :

*|NET n1
*|I xi1:a

*|NET n2
*|I xi1:b

*|NET n3
*|I xi1:c

*|NET n4
*|I xi1:d
```

In the above situation, subcircuit terminal names drn, gate, src, and body correspond to a, b, c, and d in the DSPF file, respectively. The following `HSIMSTNAME` command allows for terminal name matching.

```
.param HSIMSTNAME = "sub=nmos drn=a gate=b src=c body=d"
```

HSIMSTSWAP

Provides a mechanism to match terminals in pre- and post-layout netlists.

Syntax

```
.param HSIMSTSWAP="sub=sub_name port# port#"   (where the
   port# starts at 1)
```

Description

This command is used when terminals in a pre-layout subcircuit definition are swapped in the post-layout instance pin definition. It provides a mechanism to indicate that two particular terminals of a given subcircuit are swapped. An example is given below:

Example

```
Pre-layout netlist:

.subckt nmos drn gate src body
…….
.ends nmos

xi1 n1 n2 n3 n4 nmos

DSPF file:

*|NET n1
*|I xi1:src

*|NET n2
*|I xi1:gate

*|NET n3
*|I xi1:drn

*|NET n4
*|I xi1:body
```

In the example above, the drn and src terminals are swapped in the DSPF file, causing a mismatch during back-annotation. In order to avoid this problem, use the following `HSIMSTSWAP` command:

```
.param HSIMSTSWAP="sub=nmos 1 3"
```

HSIMSUBBUSDELIM

`HSIMSUBBUSDELIM` controls how HSIM expands buses in a .subckt line in a netlist.

Syntax

`.param HSIMSUBBUSDELIM =` *`delim_character`*

Description

This command specifies the bus notation used in a SPICE netlist and is referenced in two locations:

- hdl test interface: A bus signal name in a testbench module is broken down into bit signal names with this bus notation so that HSIM can find the signal connection in the SPICE netlist.
- SPEF: If SPEF is specified, the parser converts all bus names in SPEF input so that they can be matched in the HSIM database for back annotation.

Example

If a bus is named A<0:3>, applying .param `HSIMSUBBUSDELIM=<>` becomes:

```
A<0>, A<1>, A<3>
```

HSIMSUBMODELBIN

`HSIMSUBMODELBIN` allows model sharing in the macro model.

Syntax

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

Arguments

Argument	Description
`0`	Expands all the models in each instance hierarchy. For a binning model, each bin has a unique name for that model.
`1`	Expands but reuses the model if the model parameters are identical (the default).

Description

Note that this command does not benefit simulations that use macro-style models.

HSIMTAUMAX

Defines an upper limit for the simulation time step size.

Syntax

```
.param HSIMTAUMAX = value
```

Description

This command defines an upper limit for the time step size that safeguards the unusual situation when a sudden fast transition is self-triggered within an almost idle subcircuit. The `HSIMTAUMAX` setting in general has little effect on simulation speed, except in the case of low-frequency circuit simulation. Due to the large time constant of low-frequency circuits, which can allow larger time step size, the 2 ns default value for `HSIMTAUMAX`could be too small. It is recommended to increase the value to a similar order of the time constant of such circuit.

In HSIM, you can set `HSIMTAUMAX` at the top-level such that it has a global effect on the entire circuit. You can also specify it locally, however, the locally applied value must be more conservative. If `HSIMTAUMAX` is more aggressively applied, no local effects are derived.

HSIMTIMEFORMAT

Controls the print out format of the CPU time.

Syntax

```
.param HSIMTIMEFORMAT = <1|2|3|4|7|8|15>
```

Arguments

Argument	Description
`1`	Seconds only (the default)
`2`	Minutes only
`3`	Minutes/seconds
`4`	Hours only
`7`	Hours/minutes/seconds
`8`	Days only
`15`	Days/hours/minutes/seconds

Description

This command controls the print out format of CPU time in seconds, minutes, hours, or days.

HSIMTIMESCALE

Changes the default time unit.

Syntax

```
.param HSIMTIMESCALE = value
```

Description

The time unit in HSIM is 1 picosecond (1E-12 second) by default. You can change this default time unit by setting the `HSIMTIMESCALE` global parameter `HSIMTIMESCALE`. Refer to Table 5.

Table 5 Simulation Time-Scale Control Parameters

Command	Description
HSIMTIMESCALE=10	Sets the time unit to 10 picoseconds.
HSIMTIMESCALE=0.1	Sets the time unit to 0.1 picosecond.

The default value of 1 should have adequate time precision resolution for practical applications in which the operating frequency is likely to be less than 100 GHz. The time unit value limits the lower bound of time stepsize. Each time stepsize must be an integer multiple of the time unit.

Example

Example 2 Setting HSIMTIMESCALE=100 limits each time stepsize to be:

`100 ps, 200 ps, … 1 ns, 1.1 ns, …`

It is recommended to adjust the value of `HSIMTIMESCALE` to the same order of the required time resolution.

HSIMTLINEDV

Controls the simulation time step for transmission lines.

Syntax

`.param HSIMTLINEDV = value`

Description

This command affects selecting timestep when simulating transmission lines. It provides direct influence on precision control of the signal propagation through transmission lines.

`HSIMTLINEDV` is similar to `HSIMTALLOWEDDV`, but `HSIMTLINEDV` only affects the lossy transmission line. The separate command is necessary because the transmission line behavior is determined both by voltage and current wave propagation in the transmission line. The current quantities have different dimensions than voltage quantities, so `HSIMTALLOWEDDV` is not a proper parameter to set current waves tolerances. `HSIMTLINEDV` sets relative tolerance for current waves.

A smaller `HSIMTLINEDV` value leads to a smaller time step, which causes higher precision and slower simulation speed and vice versa. The allowed parameter range is from 1.0e-3 to 1.0e+3. The default value is 1.0.

HSIMTOP

Specifies the top-level subcircuit name.

Syntax

`.param HSIMTOP = top_subcircuit_name`

Description

This command specifies the top-level subcircuit name. It is an alternation to the -top command line option and applies globally. The default is null.

When using this command, the following restriction applies: `HSIMTOP` and `HSIMNTLFMT` must be the first commands in the input.

HSIMTRAPEZOIDAL

Enables the algorithm to simulate oscillators with inductors.

Syntax

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

Description

To simulate oscillators containing inductors, you must set `HSIMTRAPEZOIDAL = 1`. This command invokes the second-order numerical integration algorithm: the Trapezoidal Algorithm. Though more suitable for simulating the RLC oscillator circuit, the Trapezoidal Algorithm may cause artificial oscillation in high-impedance nodes.This command can be set globally or locally in selected subcircuits. The default is 0.

HSIMUFILIB

Complied library location.

Syntax

`.param HSIMUFILIB=`*`compiled_library_path`*

Description

The UFI code typically consists in the following files:

- `UFI.c:` support code, do not edit
- `b3ufimain.c:` support code, do not edit
- `UFI.h:` header file, do not edit
- `b3ufiread.c:` edit to define custom parameters
- `b3ufiset.c:` edit to define custom parameters
- `b3ufi.h:` edit to define custom parameters
- `b3ufild.c:` edit to define custom program/erase equations

Example

```
.param HSIMUFILIB=./ufi_code/ufi.so
```

HSIMUSEHM

Specifies a list of C model names.

Syntax

```
.param HSIMUSEHM = model_names
```

Description

Instantiation of a C functional model is defined as a regular subckt instantiation. If both a subckt definition and a C model exist, HSIM uses the subckt definition by default. You can use the `HSIMUSEHM` command to specify a list of C model names so that HSIM uses he C model instead of subckt definition if both exist.

Example

The following example is a SPICE netlist file for an entire circuit.

Example 3

```
** SPICE netlist file of the entire circuit
* C-functional model inv_D, instead of electrical subcircuit *
inv_D,
* is used for instance X3 if the following line is *
** uncommented.
*.param HSIMUSEHM=inv_D
.global vdd
vdd 0 3.0
Vin in 0 pulse(0 3.0 1n 0.1n 0.1n 10n 20n)
X1 in n1 inv_E
* C functional model inv_C is used for X2
X2 n1 n2 inv_C
* C-functional model inv_D is used for X3 if HSIMUSEHM=inv_D is
defined
X3 n2 n3 inv_D
.tran 1n 100n

.subckt inv_E in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.subckt inv_D in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.end
** end of SPICE netlist file

/* .c file with C functional model */
hsimC_model ()
{
    pHMMODEL pInv1, pInv2;

    pInv1=hmCreateModel ("inv_C", inv1);
    pInv2=hmCreateModel ("inv_D", inv2);
    ……————————————
}

void inv1 ()
{
————————————————
}
```

```
void inv2 ()
{
...............
}
/* end of .c file */
```

HSIMUSEHMINST

Specifies a C model instantiation.

Syntax

`.param HSIMUSEHMINST = C_model_instantiation`

Description

When both a subckt definition and a C model exist, device patterns can be specified to use C model instantiation.

When subckt and C model definitions exist for inv_D, the commands in the example can be used to force C model instantiation on matching devices.

Example

The following example is a SPICE netlist file for an entire circuit.

```
** SPICE netlist file of the entire circuit
* C-functional model inv_D, instead of electrical subcircuit *
inv_D,
* is used for instance X3 if the following line is *
** uncommented.
*.param HSIMUSEHM=inv_D
.global vdd
vdd 0 3.0
Vin in 0 pulse(0 3.0 1n 0.1n 0.1n 10n 20n)
X1 in n1 inv_E
* C functional model inv_C is used for X2
X2 n1 n2 inv_C
* C-functional model inv_D is used for X3 if HSIMUSEHM=inv_D is
defined
X3 n2 n3 inv_D
.tran 1n 100n

.subckt inv_E in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.subckt inv_D in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.end
** end of SPICE netlist file

/* .c file with C functional model */
hsimC_model ()
{
    pHMMODEL pInv1, pInv2;

    pInv1=hmCreateModel ("inv_C", inv1);
    pInv2=hmCreateModel ("inv_D", inv2);
    ……................
}

void inv1 ()
{
…................
}

void inv2 ()
{
```

```
...............
}
/* end of .c file */

.param HSIMUSEHMINST=x1.xa
.param HSIMUSEHMINST=x1.xb.*
```

HSIMUSEPREVIOUSDC

Reuses the previous DC solution.

Syntax

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

Description

When performing a large number of DC steps over a voltage source range it is often practical to reuse the previous solution to seed the subsequent solution. To have HSIM reuse the previous solution as a seed for the next incremental DC solution, set `HSIMUSEPREVIOUS = 1`. The default value is 0, which results in a full independent DC analysis being performed for every increment.

HSIMUSERLIB

Specifies a UMI model library path.

Syntax

```
.param HSIMUSEHMINST = library_name
```

Description

Runs a simulation with a UMI model.

Example

In the following example, user.so is the name holder for the dynamic library.

```
param HSIMUSERLIB=”./user.so”
```

HSIMUSEVA

Specifies the Verilog-A module name.

Syntax

```
param HSIMUSEVA = module_name
```

Description

In a netlist, a Verilog-A module can have the same name as a SPICE subcircuit. By default, HSIM uses the SPICE subcircuit name in the simulation. If you specify `HSIMUSEVA=module_name`, HSIM uses the Verilog-A module instead of its SPICE counterpart.

HSIMUSEVATABLE

Actives the table model for Verilog-A models.

Syntax

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

Description

Set `HSIMUSEVATABLE = 1`to activate the table model for Verilog-A models. The default is 0.

HSIMUTFCOMPRESSLEVEL

Controls the amount of compression in `UTF` output.

Syntax

```
.param HSIMUTFCOMPRESSLEVEL=compression_level
```

Description

If simulations runs take longer in the `utf` format than they do in other output formats, you can use the `HSIMUTFCOMPRESSLEVEL` command to control the amount of compression in the output. For example:

```
.param HSIMUTFCOMPRESSLEVEL=compression_level_number
```

where *compression_level_number* can be an integer 1 through 9, with 9 specifying the greatest amount of compression. The default is 3.

You can also increase the run time for `utf` output by directing the output to a local disk, such as /tmp.

Note:

Contact Synopsys to obtain the latest libUTF.so library.

HSIMV2S

Invokes the v2s utility during netlist parsing.

Syntax

```
.param HSIMV2S="<options> filename"
```

Description

This command invokes the v2s utility during netlist parsing. It is an alternative to running v2s manually in order to obtain the converted output and including the file in the SPICE input. Instead, the entire procedure is done inside HSIM.

`HSIMV2S` specifies the v2s arguments and tells HSIM that certain netlist information is read from Verilog inputs. It inserts default arguments shown in the following table if they are not explicitly specified.

Table 6 HSIMV2S Option Defaults

Option	Default
-s	Provided for compatibility with v2s. Will be ignored.
-const0	HSIMLOGICHV
-const1	HSIMLOGICLV
-bn	
-top	HSIMTOP
-o	Provided for compatibility with v2s. Will be ignored. v2s.spi.

Example

```
.param HSIMV2S="top.v -o top.spi"
```

This example converts modules defined in top.v to subckt definitions and writes the output to top.spi.

Note:

Refer to the demo at hsim/tutorial/v2s.

HSIMV2SD

Reads the Verilog netlist directly instead of invoking v2s.

Syntax

```
.param HSIMV2S="<options> filename"
```

Description

This utility works in a similar manner to `HSIMV2S` however, `HSIMV2SD` reads the Verilog netlist directly instead of invoking v2s. Additionally, the entire process is done within HSIM. All `HSIMV2S` options can be used with `HSIMV2SD` except the -o option as described in the following table.

Table 7 HSIMV2S Option Defaults

option	Default
-s	Provided for compatibility with v2s. Will be ignored.
-const0	HSIMLOGICHV
-const1	HSIMLOGICLV
-bn	HSIMBUSDELIMITER
-top	HSIMTOP
-o	Provided for compatibility with v2s. Will be ignored. v2s.spi.

Example

```
.param HSIMV2SD='verilog_netlist' -top top_sub_circuit_name.
```

HSIMVABRANCHPART

Partitions current nodes.

Syntax

```
.param HSIMVAPARTITION="module_name"
```

Description

This command partitions current nodes in addition to the HSIM default nodes. By default, HSIM considers current branch nodes as inout regardless of their

declaration in a module. The `HSIMVABRANCHPART` command forces HSIM to consider all nodes.

HSIMVACROSSTTOL

Add the proper time point for the Verilog-A `cross` function.

Syntax

```
.param HSIMVACROSSTTOL= value
```

Description

`HSIMVACROSSTTOL` is for adding the proper time point in HSIM for the Verilog-A `cross` function by defining the tolerance of the time specification. The default value is 100p. For a tight simulation with a high accuracy demand, the default value of 100p is might be too large and should be set to a smaller value.

HSIMVACROSSVTOL

Add the voltage tolerance for the Verilog-A `cross` function.

Syntax

```
.param HSIMVACROSSVTOL = value
```

Description

Use `HSIMVACROSSVTOL` to add voltage tolerance in HSIM for Verilog-A the `cross` function. The default is +0.1V. For a tight simulation with a high accuracy demand, the default value of 0.1V may be too large and should be set to a smaller value.

HSIMVAPARTITION

Partitions a Verilog-A module.

Syntax

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

Description

Set `HSIMVAPARTITION=1` to re-order the ports and internal nodes of the module so that inout and output nodes are channel-connected and separated from input nodes. In order for partitioning to work efficiently, the input and

inout,output ports must be defined correctly. There should not be any current branches from input ports.The default is 0.

HSIMVAPRINTVAR

Prints variables in Verilog-A instances.

Syntax

```
.param HSIMVAPRINTVAR= <0|1)
```

Description

When `HSIMVAPRINTVAR=0` (the default), no Verilog-A instance variables are printed. Set `HSIMVAPRINTVAR=1` to print all variables in Verilog-A instances.

HSIMVATABLERANGE

Defines the range of the Verilog-A table.

Syntax

```
.param HSIMVATABLERANGE = value
```

Description

This command defines the range of the Verilog-A table. The default value is 2*Vdd.

HSIMVATABLESIZE

Defines the number of elements in the Verilog-A table.

Syntax

```
.param HSIMVATABLERANGE = value
```

Description

This command defines the number of elements in the Verilog-A table. The default is 100.

HSIMVBS3END

Specifies the ending Vbs voltage.

Syntax

```
.param HSIMVBS3END = value
```

Description

`HSIMVBS3END` specifies the ending Vbs voltage with a positive value.

HSIMVBS3START

Specifies the starting Vbs voltage.

Syntax

```
.param HSIMVBS3START = value
```

Description

`HSIMVBS3START` specifies the starting Vbs voltage. The specified value must be non-positive.

HSIMVBSEND

Specifies the `HSIMVDD` value for the BSIM3 model.

Syntax

```
.param HSIMVBSEND = value
```

Description

The default value is `HSIMVDD`+1. For the BSIM3 model, the default value for `HSIMVBSEND` is 20.

HSIMVCD2VEC

Specifies the VCD and signal information file names.

Syntax

```
.param HSIMVCD2VEC = "-nvcd vcd_file1-nsig sig_file1
.param HSIMVCD2VEC = "-nvcd vcd_file2-nsig sig_file
...
.param HSIMVCD2VEC = "-nvcd vcd_file_n-nsig sig_file_n
```

Description

vcd_file is the VCD file name. *sig_file*is the signal information file name. This file maps the digital signals in the VCD file to the analog signals in the design netlist.You specify the VCD file and signal information files in the HSIM netlist. You can specify multiple files.

For details on signal information file, see Using the Signal Information File on page 269.

See the HSIM log file if there are errors processing the VCD or signal information files. Simulation stops if there is an error processing these files.

HSIMVDD

Influences supply voltage control.

Syntax

```
.param HSIMVDD = value
```

Description

Voltages are influenced by the supply voltage under the control of `HSIMVDD`. The default for `HSIMVDD` is 3V.

In cases where HSIM cannot easily find DC convergence, set HSIMAUTOVDD on page 51 `= 1` to aid HSIM. These cases include situations where there are POWER nets present in the netlist, and where there are multiple power supplies used for different parts of transistors.

HSIMVDSEND

Specifies the VDS ending value.

Syntax

```
.param HSIMVDSEND = value
```

Description

`HSIMVDSEND` IS A MOSFET table control command that specifies the VDS voltage ending value. The default value is `HSIMVDD`+1.

HSIMVGSEND

Specifies the VGS ending value.

Syntax

```
.param HSIMVGSEND = value
```

Description

`HSIMVDSEND` IS A MOSFET table control command that specifies the VGS voltage ending value. The default value is `HSIMVDD`+1.

HSIMVECTORFILE

Specifies the digital vector input and output files.

Syntax

```
HSIMVECTORFILE = file_name
```

Description

Specifies the digital vector input and output files for HSIM. The default unit of time for handling vector files is nanoseconds. Any number of vector files can be specified by repeatedly specifying `HSIMVECTORFILE`. To change the unit of time, such as from nanosecond to picoseconds, use the tunit statement. Refer to tunit on page 265. The maximum number of vector file characters per line is 2048.

HSIMVERILOGA

Specifies a Verilog-A modules file.

Syntax

```
.param HSIMVERILOGA = v-a_module_file_path
```

Description

To include Verilog-A modules in a SPICE netlist, use the `HSIMVERILOGA` command to specify the source file name. The path to the file name can be relative or absolute.

The Verilog-A source file is compiled and the model is instantiated in the circuit in the same format as a SPICE subcircuit.

HSIMVHI, HSIMVHL

Specifies the dynamic threshold voltage values.

Syntax

```
.print HSIMVHI =
   "vhi_1*(logic_exp1)+vhi_2*(logic_exp2)<+vhi_3*
   (logic_exp3) …>"
```

Arguments

Argument	Description
vhi_1, vhi_2, vhi_3	Voltage values.
logic_exp1, logic_exp2, logic_exp3	Can be expressions such as b0*b1==1 when both b0 and b1 are 1s, or b0*b1==0, when both b0 and b1 are 0s. b0 and b1 are the netlist nodes.

Description

The vector check process uses the fixed threshold voltage values throughout the entire simulation. However, these fixed threshold voltage values cannot capture the circuit dynamic behavior in all cases. Use .print HSIMVHI and .print HSIMVHL to specify dynamic threshold voltage values for logic HIGH and logic LOW states to resolve this problem.

HSIMVHTH

Specifies the voltage threshold for the HIGH logic state.

Syntax

```
.param HSIMVHTH = value
```

Description

Specifies default voltage threshold for the HIGH logic state. The default value is 9.7*VDD.

HSIMVHTHRATIO

Specifies the ratio for the voltage threshold for the HIGH logic state.

Syntax

```
.param HSIMVHTHRATIO = value
```

Description

Specifies the ratio for the voltage threshold for the HIGH logic state at a ratio other than $0.7*V_{DD}$. The default is 0.7. This parameter is specified globally. For example:

```
.param hsimautovdd=1 hsimvlthratio=0.4 hsimvhthratio=0.6
```

The 3.0 supply domain is 1.2 vlth / 1.8 vhth and the 5.0 supply domain is 2.0 vlth / 3.0 vhth threshold levels.

HSIMVLO

Specifies the dynamic threshold voltage vare for the LOW logic state.

Syntax

```
.print HSIMVLO =
   "vhi_1*(logic_exp1)+vhi_2*(logic_exp2)<+vhi_3*
   (logic_exp3) …>"
```

Arguments

Argument	Description
vhi_1, vhi_2, vhi_3	Voltage values.
logic_exp1, logic_exp2, logic_exp3	Can be expressions such as b0*b1==1 when both b0 and b1 are 1s, or b0*b1==0, when both b0 and b1 are 0s. b0 and b1 are the netlist nodes.

Description

The dynamic threshold voltage values for the logic HIGH and logic LOW states can be limited to the selected signal by using vector file masks.

Example

```
signal in1_1 in1_2 in2_1 in2_2 out1 out2 outx invout
radix 11111111
io iiiioooo
mask m3 invout
vhth HSIMVHI m3
vlth HSIMVLO m3
```

In this example, the .print `HSIMVHI=`... and `.print HSIMVLO` commands are applied to signal invout only. outx, out1, and out2 continue using the default threshold voltage values.

HSIMVLTH

Specifies the ratio for the voltage threshold for the LOW logic state

Syntax

`.param HSIMVLTH = value`

Description

Specifies default voltage threshold for the LOW logic state.The default is 0.7*VDD.

HSIMVLTHRATIO

Specifies the ratio for the voltage threshold for the LOW logic state at a ratio other than 0.3*V_{DD}.

Syntax

`.param HSIMVLTHRATIO = value`

Description

Specifies the ratio for the voltage threshold for the LOW logic state at a ratio other than 0.3*V_{DD}. This parameter is specified globally. The default is 0.3.

HSIMVPRECISION

Controls voltage precision. For details about the `HSIMVPRECISION` settings, see HSIM^{plus} Quick Start on page 12.

Syntax

```
.param HSIMVPRECISION = <0|1|2|3|4|5>
```

Description

`HSIMVPRECISION` is a macro command that controls voltage precision. The concept of voltage precision is based on identifying the voltage level/ contribution that a simulator considers negligible. Simulators, including traditional SPICE, need to set these type of thresholds to ensure adequate throughput is maintained during the millions of mathematical calculations resolving Kirkoff's voltage laws.

HSIMVSRCDV

Adjusts the simulation time step on the primary input voltage source nodes.

Syntax

```
.param HSIMVSRCDV = value
```

Description

Similar to `HSIMALLOWEDDV`, you can use the `HSIMVSRCDV` command to adjust the simulation time step on the primary input voltage source nodes by setting the voltage change limit. the internally-calculated `HSIMVSRCDV` value for a particular input voltage source node is one tenth of abs(Vmax-Vmin). If a circuit has multiple voltage sources, there could be different internally-determined `HSIMVSRCDV` values.

A global voltage change limit for all voltage source nodes can be set by setting the `HSIMVSRCDV` value. The smaller the value, the slower the simulation speed.

HSIMVSRCLMIN

Shorts any inductor that connects to a voltage source and is smaller than the specified value.

Syntax

```
.param HSIMVSRCLMIN = value
```

Description

The `HSIMVSRCLMIN` global command is similar to `HSIMLMIN`, except it applies only to inductor networks that connect to voltage sources. `HSIMVSRCLMIN` can

only be set globally. Any inductor that connects to a voltage source and is smaller than the `HSIMVSRCLMIN` value is shorted. The default value is 1e-10.

HSIMVSRCRMIN

Shorts any resistor that connects to a voltage source and is smaller than the specified value.

Syntax

```
.param HSIMVSRCRMIN = value
```

Description

The `HSIMVSRCLMIN` global command is similar to `HSIMRMIN`, except it applies only to resistor networks that connect to voltage sources. `HSIMVSRCRMIN` can only be set globally. Any resistor that connects to a voltage source and is smaller than the `HSIMVSRCRMIN` value is shorted. The default value is 0.1.

HSIMVSRCSYNC

Synchronizes sampling points for different sinusoidal voltage sources.

Syntax

```
param HSIMVSRSYNC = <0|1>
```

Arguments

Argument	Description
0	Specifies that the sampling points are generated independently for different voltage sources (the default).
1	Specifies that the sampling points are generated only for voltage sources with highest frequency and used for voltage sources at other frequencies.

Description

When `HSIMVSRCSYNC=0` (the default), HSIM generates the sampling points for different sinusoidal voltage sources independently, which might result in oversampling and reduce performance. If you set `HSIMVSRCSYNC=1`, HSIM generates sampling points for the voltage source with highest frequency only, and applies them to voltage sources at other frequencies.

HSIMWARNFILTER

Filters warnings.

Syntax

```
.param HSIMWARNFILTER = [max_warnings:] "warning_text"
```

Arguments

Argument	Description
max_warnings:	Limits the number of warnings to suppress.
warning_text	Specifies the warning text to suppress from printing.

Description

Add `HSIMWARNFILTER` at the beginning of the top-level netlist in order to have an effect on the warnings displayed.

HSIMWARNIFNODC

Warns if DC initialization does not resolve all nodes.

Syntax

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

Description

Set `HSIMWARNIFNODC = 1` warns if DC initialization does not resolve all nodes. The process reports the warning in the log file. The default is 0.

HSIMWARNSTOP

Terminates simulation upon reaching a user-specified number of warnings.

Syntax

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

Description

Set `HSIMWARNSTOP = 1` to terminate simulation upon reaching a user-specified number of warnings. The maximum number of warnings can be set by setting:

```
.option warnlimit=#
```

You must set `HSIMWARNSTOP = 1` in order for .option warnlimit to take any effect. The default is 0.

HSIMWPETOL

Sets a tolerance to minimize the memory usage.

Syntax

```
.param HSIMWPETOL = value
```

Description

Sets a tolerance to minimize the memory usage and computation time BSIM4 WPE takes so that devices with SCA, SCB and SCC can share a common table of tolerance values if other instance parameters are the same. The default value is 1.0e-2.

HSIMWRAPPERSUB

Specifies a subcircuit name/pattern as a device model name.

Syntax

```
.param HSIMWRAPPERSUB = subckt_name
```

Description

`HSIMWRAPPERSUB` specifies a subcircuit name/pattern that can be used as device model name.

Example

```
.param HSIMWRAPPERSUB=nfet
.subckt nfet d g s b
...
.ends
M1 d g s b nfet w=.. l=..
```

With `HSIMWRAPPERSUB` specified, HSIM treats M1 as a subckt instance of subckt nfet instead of warning about model “nfet not found ...”

5

Running HSIM

Provides instructions for setting up a netlist and invoking HSIM.

Using HSIM

To use HSIM, complete the following tasks:

- Netlist setup
- Parse Error Limit setup
- HSIM invocation

Netlist Setup

Data Inputs

HSIM accepts most SPICE-compatible netlists with following components.

- Circuit description
- Device models for the following:
  - Diode
  - BJT
  - JFET
  - MOSFET
- Input stimuli
- Output specification

Only one netlist file can be specified as the HSIM input file. If a netlist contains multiple files, the additional files can be included in the primary netlist file. To include additional netlist files, use one of the following:

- .include
- .lib: Used for device models

Note:

Refer to Chapter 7, Input Netlist, for additional information.

Initialization File

Before reading the input netlist file, HSIM searches for the hsim.ini initialization file in the following directories, in the order shown:

- local dir
- $HSIM_PROJECT
- $HOME
- $HSIM_HOME/etc

Only the first initialization file that is found is read. The typical usage of the initialization file is setting the optional HSIM Simulation Parameters described in Chapter 7, Simulation Parameters. An initialization file in the $HSIM_HOME/etc directory provides the default settings for all users. Each user can have a default setup by creating an hsim.ini file in each user's home directory.

To keep the original netlist file intact, add HSIM commands to hsim.ini.

```
.param HSIMOUTPUT=wdf /* generates WaveView wdf file */
.print v(*) level=1 /* prints all top level nodes voltage waveforms
*/
```

Parse Error Limit Setup

The environment variable HSIM_PARSE_ERROR_LIMIT controls the maximum number of parse errors before HSIM aborts netlist compilation. The default value is 20. The number of parse errors can be changed, as shown in the following example:

```
% setenv HSIM_PARSE_ERROR_LIMIT 30
```

Invoking HSIM

To invoke HSIM, use the following command syntax.

```
hsim <[-i]|netlist_format> netlist_name <[-o] out_file>
   <-fsdb out_file.fsdb> <-wdf out_file.wdf> <-time
   final_time> <-top subckt_name> <-critic arguments>
   <-memlimit memory> <-wait_lic> <-cktsizeonly> <-case 0|1>
   <-post_devv> <-dp> <-parseonly> <-h>
```

Table 1 HSIM Command Line Arguments

Argument	Description
`-i`	Specifies the input netlist and provides compatibility with SPICE.
netlist_format	Specifies one of the following input netlist formats: ▪ `-spectre` for a Spectre® input file ▪ `-eldo` for an Eldo input file ▪ `-mcspice` for an MCSPICE input file ▪ `-tispice` for a TISPICE input file ▪ `-stver` for an ST-Eldo input file
netlist_name	Specifies the input netlist file name.
`-o` *out_file*	Specifies the prefix for output and log files. The default prefix name is `hsim`. HSIM generates log files that contain simulation statistics and the output files that contain simulation results.
`-fsdb` *out_file*.fsdb	Specify `-fsdb` to perform .fft and .measure analyses in the post-processing step. After regular simulation is completed, the simulation output is stored in a .fsdb file, such as out_file.fsdb. You can add .measure and/or .fft statements to the netlist file and invoke HSIM to perform .fft and measure analyses without repeating the simulation.

Table 1 HSIM Command Line Arguments (Continued)

Argument	Description
`-wdf` *out_file*.wdf	Specify `-wdf` to perform .fft and .measure analyses in post-processing step. After regular simulation is completed, the simulation output is stored in a .wdf file, such as out_file.wdf. You can add .measure and/or .fft statements to the netlist file and invoke HSIM to perform .fft and measure analyses without repeating the simulation.
`-time` *final_time*	Specifies the stop time in transient analysis.
`-top` *subckt_name*	Specifies the top level subcircuit name.
`-critic` *arguments*	Specifies the Critic command arguments that perform analysis on post-layout synthesis-driven designs. Find Critic documentation on SolvNet.
`-memlimit` *memory*	Checks the amount of memory available.

`-wait_lic`	`-wait_lic` queues the job if the requested license keys are not available, just as defined by setting the HSIM_WAIT_LICENSE environment variable to 1:
	`% setenv HSIM_WAIT_LICENSE=1`
	To invoke the HSIM 64 Bit executable, set the following environment variable:
	`% setenv HSIM_64 1`

`-cktsizeonly`	Reports which size-based license is necessary.
`-case 0|1`	Enables case-sensitive simulation.
`-dp`	Set the CPU to double precision mode.
`-parseonly`	Tells HSIM to only parse input netlists, search for .alter, alterparam, .data (transient data sweep), and Monte Carlo as well as the number of iterations and report (write a *rpt file) the type of analysis and number of iterations.
`-h`	Lists all of the command line options.

In addition, HSIMplus can use the following command line arguments.

Table 2 HSIMplus Command Line Arguments

Argument	Description
`-ra`	Invokes Phase II of Reliability Analysis. See Chapter 3, Power Net Reliability Analysis (PWRA), in the HSIMplus Reference Manual.
`-ralayout`	Generates the violation map for reliability analysis. See Chapter 3, Power Net Reliability Analysis (PWRA), in the HSIMplus Reference Manual.
`-sp2dspf`	Invokes the SP2DSPF utility. See Chapter 2, Post-Layout Acceleration (PLX) and SP2DSPF Utility, in the HSIMplus Reference Manual.
`-r`	Invokes static power net resistance analysis. See Chapter 4, Static Power Net Resistance (SPRES), in the HSIMplus Reference Manual.
`-rout`	Specifies report generation for static power net resistance analysis. See Chapter 4, Static Power Net Resistance (SPRES), in the HSIMplus Reference Manual.
`-primerail`	Generates the IVEC file for PrimeRail from reliability analysis data. See Chapter 7, HSIMplus-PrimeRail Interface, in the HSIMplus Reference Manual.
`-post_devv`	Enables post-processing for the hsimDeviceV circuit check command. See Chapter 8, CircuitCheck, in the HSIMplus Reference Manual.

6

HSIM Circuit Simulation Examples

Provides circuit simulation examples such as: an example of a circuit with a resistor ladder containing 2 billion resistors and an SRAM circuit containing up to 51 million transistors.

This chapter provides two examples of circuit simulation:

- Resistor ladder circuits containing 2 billion resistors
- SRAM circuits containing up to 51 million transistors

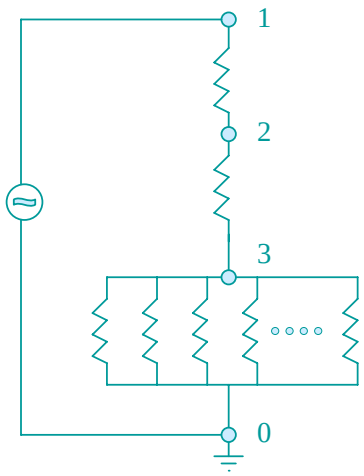
The example directory is located at $HSIM_HOME/tutorial.

Resistor Ladder Test Case

Figure 6 on page 182 illustrates a resistor ladder that is constructed hierarchically with the following features:

- The lowest level subcircuit is called r10 and contains 20 resistors.
- Each resistor has a resistance of $1.0e^{12}$ ohms.
- The equivalent resistance between subcircuit port nodes A and B is $2.0e^{11}$ ohm.
- The highest level subcircuit is called r1b. It contains 2 billion resistors with equivalent resistance of 2000 ohms.

Figure 6 The Resistor Ladder Test Case Circuit

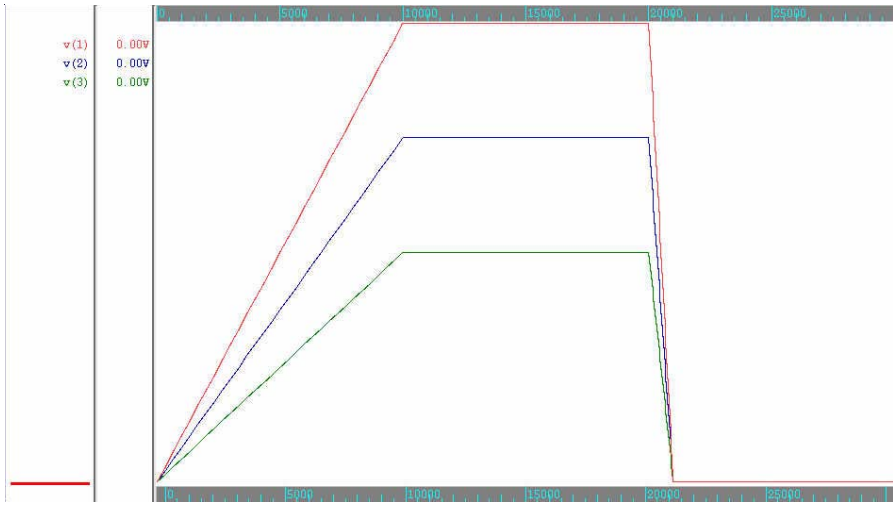


The circuit in Figure 6 on page 182 shows the top-level containing:

- Two 1000 ohm resistors
- One instance of subcircuit r1b forming a resistor ladder
- 2,000,000,002 total resistors

The input is a PWL voltage source that ramps up then down from 0V to 3V to 0V as illustrated in Figure 7 on page 182.

Figure 7 Piecewise Linear Driving Voltage and Simulation Results



Note:

The example given above has an artificially high number of resistor branches in the voltage divider to demonstrate HSIM capacity and speed in handling a large hierarchical resistive ladder.

Resistor Ladder Simulation Example

The following instructions provide step-by-step guidance for the resistor ladder test case.

1. Run the Resistor Ladder Simulation by changing the directory to tutorial/2br. Use run script to run the tutorial.
2. After the resistor ladder simulation is complete, use a waveform display tool to inspect the simulation results.

   The results are shown in Figure 7 on page 182. This test case circuit shown Figure 6 on page 182 in functions as a voltage divider where the voltages of node 2 and node 3 follow the voltage of input node 1.
3. Review the resistor ladder simulation statistics by inspecting the simulation statistics collected in the hsim.log file. The simulation takes only a few seconds. Most of the simulation time is used for netlist processing and hierarchical storage of circuit elements.

   Transient simulation has the following requirements:

   - Time < 0.1 second
   - Peak memory usage is 29M
   - Average memory usage is 0.0145 byte per element

Figure 7 on page 182 shows the results of the Resistor Ladder test.

Input Netlist for the Resistor Ladder

Following is the input netlist for the Resistor Ladder test case.

```
* 2 billion resistor circuits
.subckt r10 a b
ra1 a i1 1.0E12
rb1 i1 b 1.0E12
ra2 a i2 1.0E12
rb2 i2 b 1.0E12
ra3 a i3 1.0E12
rb3 i3 b 1.0E12
ra4 a i4 1.0E12
rb4 i4 b 1.0E12
ra5 a i5 1.0E12
rb5 i5 b 1.0E12
ra6 a i6 1.0E12
rb6 i6 b 1.0E12
ra7 a i7 1.0E12
rb7 i7 b 1.0E12
ra8 a i8 1.0E12
rb8 i8 b 1.0E12
ra9 a i9 1.0E12
rb9 i9 b 1.0E12
ra0 a i0 1.0E12
rb0 i0 b 1.0E12
.ends
.subckt r100 a b
xa1 a b r10
xa2 a b r10
xa3 a b r10
xa4 a b r10
xa5 a b r10
xa6 a b r10
xa7 a b r10
xa8 a b r10
xa9 a b r10
xa0 a b r10
.ends
.subckt r1000 a b
xa1 a b r100
xa2 a b r100
xa3 a b r100
xa4 a b r100
xa5 a b r100
xa6 a b r100
xa7 a b r100
xa8 a b r100
xa9 a b r100
xa0 a b r100
.ends
.subckt r10K a b
```

```
xa1 a b r1000
xa2 a b r1000
xa3 a b r1000
xa4 a b r1000
xa5 a b r1000
xa6 a b r1000
xa7 a b r1000
xa8 a b r1000
xa9 a b r1000
xa0 a b r1000
.ends
.subckt r100K a b
xa1 a b r10K
xa2 a b r10K
xa3 a b r10K
xa4 a b r10K
xa5 a b r10K
xa6 a b r10K
xa7 a b r10K
xa8 a b r10K
xa9 a b r10K
xa0 a b r10K
.ends
.subckt r1M a b
xa1 a b r100K
xa2 a b r100K
xa3 a b r100K
xa4 a b r100K
xa5 a b r100K
xa6 a b r100K
xa7 a b r100K
xa8 a b r100K
xa9 a b r100K
xa0 a b r100K
.ends
.subckt r10M a b
xa1 a b r1M
xa2 a b r1M
xa3 a b r1M
xa4 a b r1M
xa5 a b r1M
xa6 a b r1M
xa7 a b r1M
xa8 a b r1M
xa9 a b r1M
xa0 a b r1M
.ends
.subckt r100M a b
```

```
xa1 a b r10M
xa2 a b r10M
xa3 a b r10M
xa4 a b r10M
xa5 a b r10M
xa6 a b r10M
xa7 a b r10M
xa8 a b r10M
xa9 a b r10M
xa0 a b r10M
.ends
.subckt r1B a b
xa1 a b r100M
xa2 a b r100M
xa3 a b r100M
xa4 a b r100M
xa5 a b r100M
xa6 a b r100M
xa7 a b r100M
xa8 a b r100M
xa9 a b r100M
xa0 a b r100M
.ends
in 1 0 pwl 0 0 10n 3 20n 3 21n 0
r1 1 2 1000
r2 2 3 1000
x1 3 0 r1B

.tran 1n 30n
.print tran v(1) v(2) v(3)
.end
```

SRAM Test Case

The SRAM test case computation is focused on memory cell and sense amplifier operations.

The SRAM circuit is constructed hierarchically and contains the following elements:

- Memory cells
- Sense amplifiers
- Read/Write (R/W) control logic

Note:

The address decoder is not included in the SRAM test case.

This test case has the following characteristics:

- Memory cell: Typical 6-transistor SRAM cell.
- Sense amplifier: Differential amplifier commonly used in SRAM circuits.
- Memory bank: 1024 rows, 32 columns

The word lines used in this case have the following properties:

- Only four rows are used in this simulation: w10, w11, w12, and w13.
- Remaining word lines are connected to w14 and then to GND.

Each pair of bit-lines is connected to the sense amplifier. Subcircuit 32K contains 32K memory cells and 32 sense amplifiers. This memory bank is used to construct the following:

- 128K memory bank (subcircuit 128K).
- 512K memory bank (subcircuit 512K).

The circuits constructed have the following characteristics:

- Circuit 1 sram1.spi contains the following:
  - 1 active 512 K memory bank
  - 3.1 million transistors
- Circuit 2 sram2.spi contains the following:
  - 1 active 512 K memory bank
  - 3 dummy 512K memory banks
  - 12.6 million transistors
- Circuit 3 sram3.spi contains the following:
  - 1 active 512K memory bank
  - 15 dummy 512K memory banks
  - 51 million transistors

SRAM Example

The following instructions provide step-by-step guidance for the SRAM test case.

1. Run the SRAM Simulation by changing the directory to tutorial/sram and running the following test scripts, located in the tutorial directory:
   - run1: Performs 512K memory circuit simulations.
   - run2: Performs 2Mg memory circuit simulations.
   - run3: Performs 8Mg memory circuit simulations.
2. After simulation is complete, inspect the log files and check for the following:
   - CPU time
   - Memory usage

The three simulation results are shown in Figure 8 on page 189.

- da0: A bidirectional input/output (I/O) port driven by a PWLZ voltage source. This is an HSIM^{plus} circuit simulation-only feature that supports bidirectional operation.
- bl0: Column 0 bit line
- bl0n: Column 0 bit line bar

The CPU time to simulate the 51 million transistor SRAM is only 5 times longer than the time to simulate the 3.1 million transistor SRAM.

Note:

The 5-times increase in CPU time and memory utilization is significantly less than the 16-times increase in circuit size.

During the write cycle, the input vector writes the following data into the rows and columns shown in Table 3 on page 188.

Table 3 Input Vector Data Write Distribution

DATA	COLUMN	ROW
1 (one)	0	0
0 (zero)	0	1
1 (one)	0	2

Table 3 Input Vector Data Write Distribution

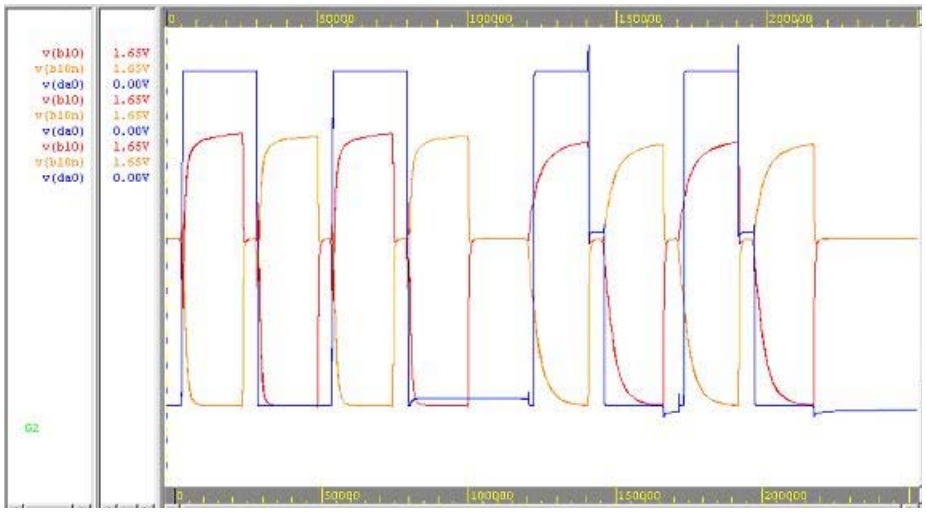
DATA	COLUMN	ROW
0 (zero)	0	3

HSIM reads back the data bits from the core cells during the read cycle.

The results of the three simulations are very close. The waveform comparison for the three simulation results is shown in Figure 8 on page 189. These results provide the following data:

- Confirm a correct data R/W operation
- Verifies there is no precision degradation in the HSIM simulation as the circuit size increases substantially.

Figure 8 Three Overlapping SRAM Test Case Simulation Results



Reviewing the SRAM Simulation Statistics

This example demonstrates HSIM efficiency. In a desktop PC with Pentium II 550 MHz processor and Microsoft Windows 2000, the simulation provides the following results shown in Table 4 on page 190.

Table 4 Desktop PC Run Results

RUN	RESULTS
run1	▪ 3.1 million SRAM transistors ▪ 8.81 seconds ▪ 11 Mbytes (of memory utilization)
run2	▪ 12.6 million SRAM transistors ▪ 16.6 seconds ▪ 14.6 M bytes
run3	▪ 51 million SRAM transistors ▪ 42.5 seconds ▪ 25.6 Mbytes

Note:

Simulation results vary, depending on the hardware and software versions used.

7

Input Netlist

Describes the procedure for developing an HSIM netlist using one or multiple files. The netlist file includes information such as: circuit topology description consisting of circuit elements and their connectivity, element models and their parameter values, simulation control parameters, stimulus input sources, and output specifications.

HSIM input netlists contain some or all of the following information:

- Circuit topology description consisting of circuit elements and their connectivity.
- Element models.
- Simulation control parameters.
- Stimulus input sources and output specifications.

These data can be described in either single file or multiple files. When multiple files are involved, an .include statement must be used to include main netlist files. Refer to .include on page 293.

Netlist Syntax Summary

HSIM netlist syntax can be summarized in the table below.

Table 5 Netlist Syntax Summary

Syntax	Description
Title line	The first line in the main netlist file is treated only as the title line. It has no effect on the circuit description.

Table 5 Netlist Syntax Summary (Continued)

Syntax	Description
Characters	Except for filename and the directory pathname used in the .include and .lib statements, each character in the netlist is case-insensitive.
Buffer size	The buffer size to store the token or identifier can be set up to 1,024 characters.
Asterisk symbol	A line that starts with an asterisk (*) is treated as a comment line. It is ignored.
Dollar symbol	To add comments after the input text on the same line, precede that comment with the dollar ($) character.
Line continuation symbols	A line may continue with a backslash '\' or double back slash (\\) character placed at the end of the line. A plus (+) sign can also be placed at the beginning of the next line.
Key identification character	Each element description starts with a specific key identification character, such as M for MOSFET and R for resistor.
Period symbol	Other than the element description, there are lines that start with the period character '.'. Examples include:.param - defines the parameter value.model - defines the device model
Element and control lines	The element and control lines placed inside the subcircuit scope are mostly effective within the subcircuit.
Line sequence	The line sequence has no effect on the circuit description.
Line placement	Except for elements defined within the subcircuit definition which cannot be placed outside the scope between .subckt and .ends, any element or control option line can be placed anywhere between the first or title line and the last line. The last line is .end.
HSIMNOMULTIEND on page 101	HSIM accepts multiple .end statements. If you want HSIM to error out during parsing when multiple .end statements exist, issue the following command: `.param HSIMNOMULTIEND=1`

Netlist Differences Between HSIM and SPICE

The input netlist format for HSIM is almost completely compatible with that of SPICE or HSPICE® simulators. There are some differences which are described as follows:

- HSIM reads input netlist and performs the circuit simulation, even when the netlist contains elements that are supported by SPICE but are not supported by HSIM. Elements that are not supported are ignored and warning messages are displayed. To view which elements HSIM does not support, refer to Table 6 on page 193.
- HSIM also supports elements that are not supported in SPICE. For detailed information refer to Table 6 on page 193.

SPICE and HSIM Element and Capability Support

The table below shows the relationship between SPICE and HSIM elements that are supported or not supported by each other.

Table 6 Element Support Matrix

Element List	SPICE Elements/ HSIM Supported	HSIM Element/ Not SPICE Supported
PASSIVE		
Resistor	Yes	
Capacitor	Yes	
Self and Mutual Inductors	Yes	
Lossless Transmission Line (LLTL)	Yes	
Lossy Transmission Line (LYTL)	Yes	
ACTIVE		

Table 6 Element Support Matrix (Continued)

Element List	SPICE Elements/ HSIM Supported	HSIM Element/ Not SPICE Supported
Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)	Yes	
Diode	Yes	
Bipolar Junction Transistor (BJT)	Yes	
Junction Field-Effect Transistor (JFET)	Yes	
Metal-Semiconductor Field-Effect Transistor (MESFET)	Yes	
SOURCE		
Current-Controlled Current Source (CCCS)	Yes	
Current-Controlled Voltage Source (CCVS)	Yes	
Voltage-Controlled Current Source (VCCS)	Yes	
Voltage-Controlled Voltage Source (VCVS)	Yes	
Independent Voltage Sources: ▪ AM ▪ DC ▪ EXP ▪ PULSE ▪ PWL ▪ SFFM ▪ SIN	Yes	
OTHER		
Voltage-Controlled Resistor (VCR)	Yes	
Voltage-Controlled Capacitor (VCC)	Yes	
PWLZ Element		Yes

Table 6 Element Support Matrix (Continued)

Element List	SPICE Elements/ HSIM Supported	HSIM Element/ Not SPICE Supported
VECTOR Input Element		Yes
VECTOR Output Automatic Comparison		Yes
Functional element modeled by C behavioral description. Refer to Chapter 21, C Language Functional Model.		Yes

The table below shows the relationship between SPICE and HSIM capabilities that are supported or not supported by each other.

Table 7 Capabilities Support Matrix

Capabilities List	SPICE Capabilities/ HSIM Supported	SPICE Capabilities/ Not HSIM Supported	HSIM Capabilities/ Not SPICE Supported
Transient Analysis	Yes		
DC Operating Point	Yes		
Measure	Yes		
Fourier Transform, and Fast Fourier Transform	Yes		
Bisection Optimization	Yes		
AC Analysis	Yes		
DC Analysis	Yes		
Sensitivity Analysis		Yes	
Noise Analysis		Yes	
Pole-Zero Analysis		Yes	

Table 7 Capabilities Support Matrix (Continued)

Capabilities List	SPICE Capabilities/ HSIM Supported	SPICE Capabilities/ Not HSIM Supported	HSIM Capabilities/ Not SPICE Supported
Timing Analysis			Yes
Power Analysis			Yes
Interactive Mode			Yes

Encrypted HSPICE Netlists

HSIM is able to parse encrypted HSPICE transistor level netlists, including Triple DES (3DES) and Verilog-A encryption, as long as the netlist was encrypted using the `metaencrypt` encryption utility with the `-r synopsys_tool` option set to either `hsim` or `synopsys_all`.

Note:

When encrypting a Verilog-A file, the entire unencrypted model must reside in one file before using the `metaencrypt` utility. HSIM currently does not support partial encryption for Verilog-A modules.

For details on `metaencrypt`, refer to the “Library and Data Encryption” chapter in the *HSPICE® User Guide: Simulation and Analysis*.

Device Models

HSIM supported device models include the following:

- BSIM1 MOSFET Model
- BSIM3 MOSFET Model v3.0, v3.1, v3.2, v3.2.3, v3.2.4, and v3.3.0

  Note:

  The stress model formula originally included in BSIM4.3 is also available in the BSIM3 Model by setting stimod=1.
- BSIM3SOI Model v2.0, v2.2.1, v2.2.2, v2.2.3, v3.0, v3.1, and v3.2

- BSIM4SOI Model v4.0. HSIM supports the following instance parameters supported in HSPICE: MULID0, MULVSAT, DELK1, DELNFCT and DTEMP for BSIM SOI4.
- BSIM4 MOSFET Model, v4.1, v4.2, v4.2.1, v4.3.0, v4.4.0, v4.5.0, v4.6.0, and v4.6.3
- HiSIM MOSFET Model v2.3.0
- Philips MOSFET Models: MOS9, MOS11, MOS30, MOS31, and MOS40
- EKV MOSFET Model
- SPICE Level-1/Level-2/Level-3 MOSFET Models
- BJT Models: G.-P. Model, VBIC Model, Mextram Model, HICUM Model
- Diode Model: JUNCAP2, v200.1
- PSP Model v102.0, v103
- MOS varactor model in Spectre® format (model mname mosvar)
- JFET Model
- Ferroelectric Capacitor (FeCAP) Model. Refer to Chapter 18, Ferroelectric Capacitor (FeCap) Model.) for detailed information.

Gate Capacitance Model

HSIM uses its own gate capacitance model to closely match corresponding SPICE results. The gate capacitance model automatically adjusts its parameter values according to the given SPICE model.

The optional charge conservation feature can be enabled to rigorously conserve charge at a slight expense of the simulation speed. Control parameters for MOSFET model complexity are described in the following sections of Chapter 7, Simulation Parameters:

- HSIMSPICE on page 310
- Device Model Parameters on page 315
- MOSFET Table Model Control Parameters on page 320

Stress Model

The stress model formula originally included in BSIM4.3 Model is also available in BSIM3 Model by setting stimod=1.

Element Statements

Netlist elements are described as follows:

```
NAME node1 node2 <… nodeN> <model_name> evalue <optional
   parameters>
```

Element statements are described in the table below.

Table 8 Element Syntax Conventions

Parameter	Description
NAME	Specifies the type and the name of an element. The first letter in the name field identifies the element type. For example: ▪ M represents a MOSFET ▪ C represents a capacitor ▪ R represents a resistor The remaining alphanumeric characters in the name field define the unique element name.
node1 node2 <… nodeN>	Specifies the names of the circuit nodes to which the element is connected.
model_name	Refers to a model. Detailed information of that model is provided in a separate model definition.
evalue	Specifies the value of the element. For example, the statement R100 a12 b55 1000 indicates that a resistor named R100 is connected to nodes a12 and b55, and the resistance value is 1000 ohms.
Line continuation	Statement continues to the next line. A plus sign (+) sign must be used at the beginning of the continuation line.
Commas	Provided only for ease of reading. Values are recognized as separate with blank spaces. For example, in the following line of code, `x1`, `y1`, `x2`, and `y2`, with or without commas, are recognized as four separate variables: `Gaa pos_n neg_n <vccs> pwl(1) nc1+ nc1- <delta=val2> x1, y1, x2, y2`

Note:

In this document, the syntax for some elements and models can appear to be multiple lines. However, the plus sign (+) is not included as those lines are intended to be used as single lines.

Resistor

Syntax

```
Raa n1 n2 <model_name> <r=>rval   <tc1=val2 <tc2=val3>
   <scale=val4> <m=val5> <dtemp=val6> <l=val7> <w=val8>
   <c=val9>
Raa n1 n2 <r=>"expr"
```

Parameters

Resistor parameters are described in the table below.

Table 9 Resistor Parameters

Parameter	Default	Description
Raa		Resistor element name which must begin with the character R.
n1		First node name.
n2		Second node name.
model_name		Resistor model name. It is used to refer to a resistor model.
r		Keyword to identify resistance value, rval, in ohms at room temperature.
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter which scales the resistance by the specified value.
expr		A mathematical expression defining the resistance. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.
m	1	Multiplier for parallel instances of a resistor.

Table 9 Resistor Parameters (Continued)

Parameter	Default	Description
dtemp	0	Difference between the resistor temperature and the global circuit temperature in degrees Celsius.
l	0	Length of resistor. When unspecified, l is assigned the default value.
w	0	Width of resistor. When unspecified, w is assigned the default value.
c	0	Capacitance connected from n2 to ground. When unspecified, c is assigned the default value.

Examples

```
RK15   node1 node5   100

R1990 X12    X90     1.5K

rbx    n1     n2      "200+0.02*(v(n3) - v(n4))"
```

Capacitor

Syntax

```
Caa n1 n2 <model_name> <c=>cval <tc1=val2 <tc2=val3>> <scale=val4>
<m=val5> <dtemp=val6> <ic=val7> <l=val7> <w=val8>
```

Caa n1 n2 <c=>”expr”

Caa n1 n2 Q=“expr”

Caa n1 n2 POLY C0 C1 ...

Parameters

Capacitor parameters are described in the table below.

Table 10 Capacitor Parameters

Parameter	Default	Description
Caa		Capacitor element name, which must begin with the character C.

Table 10 Capacitor Parameters (Continued)

Parameter	Default	Description
n1		Positive node name.
n2		Negative node name.
model_name		Capacitor model name. It is used to refer to a capacitor model.
c		Keyword to identify capacitance value, cval, in farads at room temperature.
Q		Q identifies the capacitor charge in Coulombs given by expression expr, which can depend on node voltages. In this case value of the capacitance is determined by differentiation of expression over node voltages. This element can be charge conserving if parameter hsimcapcc=1 is set (see section Simulation parameters, subsection Device model parameters).
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter.
expr		A mathematical expression defining the capacitance. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.
m	1	Multiplier for parallel instances of a capacitor.
dtemp	0	Difference between the capacitor temperature and the global circuit temperature in degrees Celsius.
ic		Initial voltage across the capacitor. The .ic on page 292 overrides this value.
l	0	Length of capacitor. When unspecified l is assigned the default value.
w	0	Width of capacitor. When unspecified, w is assigned the default value.

Table 10 Capacitor Parameters (Continued)

Parameter	Default	Description
POLY		Specifies capacitance using a polynomial form.
C0 C1 ...		Coefficient of a polynomial in voltage describing the capacitor value. An example is: C=C0 + C1 * V + C2 * V * V +...

Note:

The Ferroelectric Capacitor Model from Ramtron is supported as a Level-6 Model. The MOSCAP Model from Motorola is supported.

Examples

```
C12A5   HB12   0       1.2pf
```

cpump aref ngnd "1e-13 * v(bias1) * v(state3)"

Self-Inductor

Syntax

```
Laa n1 n2 <l=>lval <tc1=val2 <tc2=val3>> <scale=val4>
   <m=val5> <dtemp=val6> <ic=val7>
```

Parameters

Self-inductor parameters are described in the table below.

Table 11 Self-Inductor Parameters

Parameter	Default	Description
Laa		Self-inductor element name which begin with the character L.
n1		Positive node name.
n2		Negative node name.
l		Key word to identify self inductance value, lval.
tc1		First-order coefficient for temperature effect.

Table 11 Self-Inductor Parameters (Continued)

Parameter	Default	Description
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter.
m	1	Multiplier for parallel instances of self-inductor.
dtemp	0	Difference between the self-inductor temperature and the global circuit temperature in degrees Celsius.
ic		Initial current flowing through the inductor.

Example

```
LT170   42   69   1uh
```

Mutual Inductor

Syntax

```
K11 L22 L33   <k=>kvalue
```

Parameters

Mutual inductor parameters are described in the table below.

Table 12 Mutual Inductor Parameters

Parameter	Description
K11	Mutual inductor element name, which must begin with the character K.
L22	First coupled inductor name.
L33	Second coupled inductor name.
k	Keyword to identify mutual coupling coefficient value, kvalue. The value is greater than 0, and less or equal to 1.

Example

```
K200   L100   L32   0.6
```

Lossless Transmission Line

Syntax

```
Taa in_n  refin_n  out_n  refout_n z0=val2 td=val3 <l=val4>
Taa in_n  refin_n  out_n  refout_n z0=val2 f=val3 <nl=val4>
```

Parameters

Lossless transmission line parameters are described in the table below.

Table 13 Lossless Transmission Line Parameters

Parameter	Default	Description
Taa		Lossless transmission line element name, which must begin with the character T.
in_n		Input node name.
out_n		Output node name.
refin_n		Reference node name for input signal.
out_n		Reference node name for output signal.
z0		Characteristic impedance.
td		Transmission delay time in the unit of second per meter.
l	1	Transmission line length in the unit of meter.
f		Characteristic frequency to determine the normalized electrical length which can be specified by the nl parameter.
nl	0.25	Normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency specified by f parameter.

Example

```
T100 in1   gnd out1 gnd   z0=50   td=1n l=2
```

In this example, a 2-meter transmission line T100 is connected from node `in1` to node `out1` with the reference nodes for input signal and output signal grounded. The characteristic impedance is 50 ohms and the transmission delay is 1 ns per meter.

Lossy Transmission Line

HSIM supports lossy multiconductor transmission lines. The N-Wire line (N=1,2,3,..) has (2*n+2) external terminals which must be specified in the element description. The line is assumed to be uniform in the direction of its length.

Syntax

```
Waa n1 <n2 ... nN > nR m1 <m2 ... mN > mR n=N l=val3
   rlgcmodel=model_name|rlgcfile=file_name|
   fsmodel=fs_model_name
```

This syntax indicates the value of N is at least 1 (one), and that at least one terminal must be specified.

Parameters

Lossy transmission line parameters are described in the table below.

Table 14 Lossy Transmission Line Parameters

Parameter	Description
Waa	Lossy transmission line element name, which must begin with the character W.
nk	Input node name for the kth wire of the transmission line.
nR	Reference node name for input signal.
mk	Output node name for the kth wire of the transmission line.
mR	Reference node name for output signal.
n	Number of wires (signal conductors).
l	Transmission line length in the unit of meter.

Table 14 Lossy Transmission Line Parameters

Parameter	Description
rlgcmodel	Model name for the transmission line.
rlgcfile	Specifies a file name that contains RLGC information.
fsmodel	Specifies a model name for the field solver which calculates RLGC information.

Example

```
W100   n1 n2 n3 0 m1 m2 m3 0 n=3 l=0.3 rlgcmodel=tline1
```

Lossy Transmission Line Model

The n-wire transmission line behavior can be described by the following equations:

```
- ∂ V / ∂ x=R I + L ∂ I / ∂ t
- ∂ I / ∂ x=G V + C ∂ V / ∂ t
```

V=V(x,t) and I=I(x,t)

are n-dimensional voltage and current vectors. L, C, R, and g are inductance, capacitance, resistance, and conductance matrices per unit length. In the case of temporal dispersion, these matrices can be frequency dependent.

The current implementation of the multiwire lossy transmission line model assumes the following functional form of the matrices in the frequency domain: L=Lo (frequency independent), C=Co (frequency independent), R=Ro + (1+j) f½ Rs, G=Go + f Gd.

Here f is the frequency, j is imaginary unit, matrices Ro, Go, Rs, and Gd are frequency independent. Matrices Rs and Gd are known as the skin effect matrix and the dielectric loss matrix.

Matrices Lo, Co, Ro, Go, Rs, and Gd are symmetrical and positive definite. All entries of the matrices Lo, Ro, Rs should be nonnegative. All diagonal terms of the matrices Co, Go, and Gd should be non-negative, while all off-diagonal entries should be non-positive. Diagonal terms of matrices Lo and Co should be positive. If some of the matrices are omitted, their default values are assumed to be zeros. Matrices Lo and Co should always be defined. If any of the other four matrices is defined, Ro should also be defined.

Syntax

```
.model model_name w modeltype=rlgc n=val_n lo=L_matrix
   co=C_matrix <ro=Ro_matrix <go=Go_matrix> < rs=Rs_matrix>
   <gd=Gd_matrix>>
```

Parameters

Lossy transmission line model parameters are described in the table below.

Table 15 Lossy Transmission Line Model

Parameter	Description
model_name	The model name for the transmission line.
w	Keyword to identify the lossy transmission line model.
modeltype	Model format selector. At present, only rlgc format is available. This format is used to indicate that L, C, R, and G matrices are used. This is the only format that is supported by the HSIM at present.
n	Number of signal conductors. This number should be the same as that used on the element line.
lo	Inductance matrix in henries per meter .
co	Capacitance matrix in farads per meter.
ro	Resistance matrix in ohms per meter.
go	Shunt conductance matrix in ohms per meter.
rs	Skin-effect resistance matrix in ohms per meter.
gd	Dielectric-loss conductance matrix in ohms per meter.

Because of the matrix symmetry, only part of the matrices must be specified. HSIM requires specification of the lower diagonal part of matrices in the row major order. Each matrix keyword is followed by N*(N+1)/2 matrix coefficients. For instance, in the case of n=3, the matrix lo should be specified in the following format: lo=Lo11 Lo21 Lo22 Lo31 Lo32 Lo33.

Example

```
.model tline1 w modeltype=rlgc n=3
+ lo=
+ 1.9e-07
+ 2.0e-08  2.0e-07
+ 1.0e-08  2.0e-08  2.1e-07
+ co=
+ 2.4e-10
+ 2.0e-11  2.3e-10
+ 1.0e-11  2.0e-11  2.2e-10
+ ro=
+ 3.0
+ 0.0    3.0
+ 0.0    0.0    3.0
+ rs=
+ 4.2e-4
+ 0  4.2e-4
+ 0  0  4.2e-4
+ gd=
+ 3.1e-11
+ 3.0e-12  3.1e-11
+ 3.0e-12  3.0e-12  3.1e-11
```

Lossy Transmission Line File

Lossy transmission line data can be specified in a separate RLGC file. This method does not offer any advantage over RLGC model which can also be placed in a separate file and included through .include statement, however it is supported for compatibility.

To use RLGC file, the element statement should use keyword RLGCFILE=<file.name>. The file <file.name> specifies transmission line data in the following format:

```
<N>
<L-matrix>
<C-matrix>
<R-matrix>
<G-matrix>
<Rs-matrix>
<Gd-matrix>
```

where <N> is the number of lines. The following the L, C, R, G, Rs, and Gd matrices should be given in lower triangular format similar to how these matrices are specified in RLGC model. This is a positional format and hence has no keywords. Star in the first position can be used to include comments.

Example

```
*
* N=
     3
*
* Lo=
     2.2e-6
     4.0e-7          3.0e-6
     8.0e-8          5.0e-7          2.8e-6
*
* Co=
     2.4e-11
     -5.0e-12        2.1e-11
     -1.0e-12        -6.0e-12        2.5e-11
*
* Ro=
     20.0
     0               25.0
     0               0               30.0
*
* Go=
     6.0e-4
     -1.0e-4         5.0-4
     -2.0e-5         -9.e-5          5.5e-4
*
* Rs=
     1.0e-3
     0               1.2e-3
     0               0               1.4e-3
*
* Gd=
     5.0e-13
     -1.0e-13        5.2e-13
     -1.5e-14        -8.0e-14        4.8e-13
*
```

Starting from the 2007.03 HSIM release, HSIM also supports the HSPICE-compatible W-element of the RLGC format in model or file formats with the same syntax. To invoke this option, use .param HSIMGMIWELM=1. The default value (0) uses the HSIM implementation.

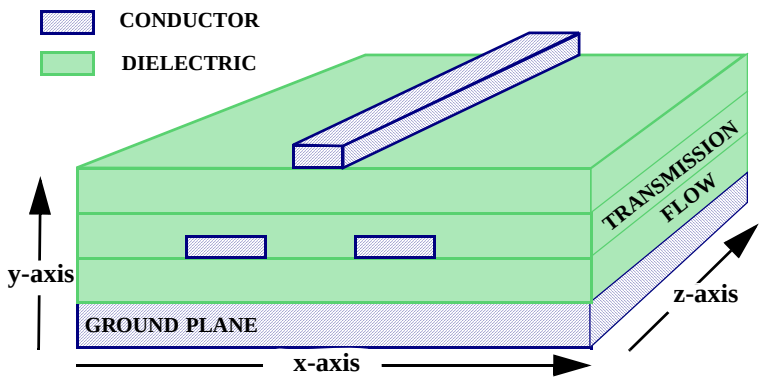
Field Solver Model

HSIM supports field solver syntax for finding the L, C, Ro, Go, Rs, and Gd matrices required for simulation of a lossy transmission line. Field solver syntax provides the specification for a two-dimensional geometry consisting of

conductors, ground planes, and material parameters such as dielectric permittivity, conductivity, loss tangent, etc.

The syntax assumes that a transmission line flows in z-direction and is situated in layered medium referred to as stack. The y-axis is assumed to be orthogonal to layers. The stack has at least one ground plane with xz-coordinates at the bottom with an optional top ground plane in some cases. The medium above the bottom plane consists of one of more layers of dielectric containing one or more conductors. Refer to Figure 9 on page 210.

Figure 9 Typical Field Solver IC Model



The element referring to the field-solver model must use the following parameter to indicate that field-solver will be used:

```
fsmodel=fs_model_name
```

The field solver model card uses the following syntax:

```
.model fs_model_name W ModelType=FieldSolver +
   layerstack=stack_name <FSOptions=FS_Option> +
   conductor=(material=material1, shape=shape1,
   origin=(x1,y1)) /
...
+ conductor=(material=materialN, shape=shapeN,
   origin=(xN,yN)) /
```

The `layerstack` keyword gives a name of the stack which should be specified somewhere else. Keywords conductor in number N (which must be the same as N in the transmission line element specification) assigns material, geometrical shape, and reference origin to each conductor. Details of material and shape are given with separate cards.

Material specification uses the following syntax:

```
.material material_name DIELECTRIC|METAL <er=val1 mu=val2
   losstangent=val3 conductivity=val4 >
```

There are two predefined material names: PEC for perfect conductor and AIR for free space. Required parameters have self-explanatory names, relative permittivity, er, and relative permeability, mu, must be positive and not less than one.

There are two shapes supported, rectangular box and a strip. The following syntax is used:

```
.shape name RECTANGLE width=val1 height=val2
.shape name STRIP width=val1
```

Width referred to x-direction, height referred to y-direction. The stack has the following syntax:

```
.layerstack stack_name
+ layer=(material1, thickness1)
...
+ layer=(materialK, thickness)
< + background=materialB>
```

The first layer must be a conductor and it is bottom ground plane. The following layers 2, 3, etc. are listed in the order of increasing coordinate y. Thickness of each layer should also be given. Each conductor except ground plane(s) must be completely situated in one dielectric layer. Touching interface of layers is allowed. Optional background is a layer of infinite thickness above layer number K. It is used only if there is no top ground plane. default background is AIR.

Additional options to field solver can be passed through the following command:

```
.fsoptions FS_Option < printdata=YES|NO computero=YES|NO
   computego=YES|NO computers=YES|NO computegd=YES|NO
   accuracy=HIGH|MEDIUM|LOW >
```

Matrices L and C are always computed. Computing matrices Ro, Go, Rs, and Gd should be requested with the above commands. The computed matrices can be written down to external file fs_model_name.rlgc if printdata=YES. Command accuracy affects the two-dimensional meshes which are built to solve Laplace equation in 2-dimensional domain. Higher accuracy results in finer mesh and larger matrices for electromagnetic fields. Refer to the following example.

Example

```
* Half Space, er=1
* XXXXXX XXXXXX
* ------------------------------------ Z=0.000302
* er=3.2 thickness=0.0001
*
* XXXXX
* ------------------------------------ Z=0.000202
* er=4.3 thickness=0.0002
* ------------------------------------ Z=2e-06
* //// Bottom Ground Plane ///////////
* ------------------------------------ Z=0
* Materials
.material copper METAL conductivity=5e+07
.material diel_1 DIELECTRIC er=4.3 losstangent=1e-3
conductivity=8e-4
.material diel_2 DIELECTRIC er=3.2 losstangent=1e-3
conductivity=8e-4
* conductor crossection shapes
.shape rect RECTANGLE width=350e-6 height=70e-6
* Dielectric stack
.layerstack Stack
+ layer=(copper, 1u)
+ layer=(diel_1, 200u)
+ layer=(diel_2, 100u)
+ background=air
* Field-solver options
.fsoptions myOption accuracy=HIGH
+ ComputeRo=yes ComputeRs=yes ComputeGo=Yes ComputeGd=yes
+ printdata=YES
.model demo W ModelType=FieldSolver
+ layerstack=Stack FSOptions=myOption
+ conductor=(material=copper, shape=rect, origin=(0u, 202u))
+ conductor=(material=copper, shape=rect, origin=(500u, 302u))
+ conductor=(material=copper, shape=rect, origin=(1000u, 302u))
```

N-Port

N-port is an element with 2*N terminals (N ports) for which the behavior is specified in the table form in the frequency domain. For HSIM, these tables should contain S-parameters of n-port. The following syntax is used for an N-port element:

Syntax

```
Sname port1 ref1 port2 ref2 ... portN refN file=filename
   <format=touchstone> <scale=scalevalue>
```

Parameters

In this syntax example, the following rules apply:

Sname

The element name which should start with S.

port1 ref1 ... portN refN

The 2*N terminals that define N ports.

port1, ref1

The first port; followed by additional ports as required.

file

Contains a table of S-parameters in one of acceptable formats.

format

Specifies the format of the table. The allowed format is `touchstone`.

scale

A multiplier for frequency in the table. For example: if scale=1e9, then the frequency is given in Gigahertz.

Alternate Syntax

The following alternate syntax can be used:

```
Sname port1 port2 ... portN ref mname=<s_model_name>
```

Where `mname=<s_model_name>` is reflected in:

```
.MODEL <s_model_name> s tstonefile=filename
```

Alternate Parameters

The Sname, port1... portN ref parameters are the same as the ones defined above. The additional parameters are:

mname

The name of the S model.

tstonefile

The name of a Touchstone file. The data in this file contains frequency-dependent array of matrixes. Touchstone files must follow the .s#p file extension rule, where # represents the dimension of the network.

For details, see *Touchstone® File Format Specification* by the EIA/IBIS Open Forum (http://www.eda.org).

Refer to the *HSPICE Simulation and Analysis User Guide* (Y-2006.03 or later) for more information on S model syntax.

MOSFET

Syntax

```
maa ndr nga nso <nbu> model_name <l=val2> <w=val3> <ad=val4>
   <as=val5> <pd=val6> <ps=val7>   <nrd=val8> <nrs=val9>
   <m=val10> <geo=val11> <rdc=val12> <rsc=val13>
   <delvto=val14> <off> <dtemp=val15>
```

Parameters

MOSFET parameters are described in the table below.

Table 16 MOSFET Parameters

Parameter	Default	Description
maa		MOSFET element name which must begin with the character M.
ndr		Drain node name.
nga		Gate node name.
nso		Source node name.
nbu		Bulk node name.
model_name		MOSFET model name. It is used to refer to a MOSFET model.
l		Gate length of MOSFET.
w		Gate width of MOSFET.
ad		Drain diffusion area.
as		Source diffusion area.
pd		Perimeter of drain-bulk junction.
ps		Perimeter of source-bulk junction.

Table 16 MOSFET Parameters (Continued)

Parameter	Default	Description
nrd		Number of squares of drain diffusion in drain resistance calculation.
nrs		Number of squares of source diffusion in drain resistance calculation.
m	1	Multiplier for parallel instances of MOSFET.
geo	0	Drain/source sharing selector for MOSFET with model parameter ACM=3.
rdc	0	Additional drain resistance.
rsc	0	Additional source resistance.
delvto	0	Zero-bias threshold voltage shift.
off	ON	Set initial condition to OFF for this element in DC operating point calculation.
dtemp	0	Difference between the MOSFET temperature and the global circuit temperature in degrees Celsius.

Example

```
MP1X 29   27   0    20   NMOS1 L=0.13U W=0.2U

M355 20   17   76   10   PMOS2 L=0.18U W=0.5U
```

MOSFET Model

Syntax

```
.model model_name nmos|pmos <encmode=0|1> <level=val2>
   <parameter1=val3> <parameter2=val4 ...>
```

Parameters

MOSFET model parameters are described in the table below:

Table 17 MOSFET Model Parameters

Parameter	Description
model_name	MOSFET model name. MOSFET element refers to the model through this name.
nmos|pmos	Keyword to identify n-MOSFET or p-MOSFET model: select one name. (See the examples below)
encmode	Applicable to BSIM4 model libraries. Use 1 (on) to suppress the printing of warning/error messages originating in BSIM4.* model cards. Default is 0 (off).
level	Selector for different levels of MOSFET model.
parameter	Model parameter.
ako	pSPICE AKO model support. (See the example below)

Examples

```
.model p18_L18_W120_ ako: p18 PMOS
```

This means that a new model with the name "p18_L18_W120_" will be created based on the reference model "p18".

```
.model mos2 nmos   level=49 tox=8e-09 tnom=25

.model mos1 pmos   level=54 vto=-0.55
```

MOSFET models are described in the table below.

Table 18 MOSFET Models

Model Level	Version	MOSFET Model
49/53	3.0	BSIM3v3.0
49/53	3.1	BSIM3v3.1
49/53	3.24 [default]	BSIM3v3.24

Table 18 MOSFET Models (Continued)

Model Level	Version	MOSFET Model
49/53	3.3	BSIM3v3.3
47	N/A	BSIM3v2
54	4.0/4.1	BSIM4.0.0/4.1.0
54	4.21	BSIM4.2.0 and BSIM4.2.1
54	4.3	BSIM4.3
54	4.4	BSIM4.4
54	4.5	BSIM4.5
54	4.6.0 [default]	BSIM 4.6
57	2.0	B3SOIPD2.0
57	2.1	B3SOIPD2.2.1
57	2.2	B3SOIPD2.2.2
57	3.0	B3SOIPD3.0
57	3.1	B3SOIPD3.1
57	3.2 [default]	B3SOIPD3.2
64/111	1.1.2 [default]	HiSIM1.1.1 and 1.2
68	2.3.0 [default]	HiSIM2.3.0
55	2.6	EKV Model
50	N/A	Philips MOS9 Model
62	N/A	RPI TFT Model
63	1100	Philips MOS11 Model Level 1100
63	1101	Philips MOS11 Model Level 1101

Table 18 MOSFET Models (Continued)

Model Level	Version	MOSFET Model
63	1102.3 [default]	Philips MOS11 Model Level 1102.3
66	1.3	HSPICE HVMOS
69	200	PSP Model
70	4.0	BSIM4SOI
6	N/A	Motorola SSIM Model
10	N/A	Motorola SSIM SOI Model
13	N/A	BSIM1
3	N/A	SPICE Level-3 Model
2	N/A	SPICE Level-2 Model
1	N/A	SPICE Level-1 Model

Note:

1. The RPI TFT Model was originally developed at Rensselaer Polytechnic Institute.
2. HSIM supports the non-quasi-static (NQS) feature for the BSIM3 and BSIM4 models. The NQS feature is turned on if NQSMOD=1 appears at BSIM3 model file or TRNQSMOD=1 appears at BSIM4 model file.
3. Juncap is the capacitance supported in the MOS9 and MOS11 models.
4. Both the built-in model and UMI approach are supported for the HSIM model by STARC/Hiroshima University in Japan.
5. Consent from Motorola is required to activate the Level-6 SSIM model.
6. HSIM supports SOIMOD=0 (PD mode) SOIMOD=1 (DD mode) and SOIMOD=2 (FD mode) for BSIM3SOIv3.2.

Diode

Syntax

```
Daa pos_n neg_n model_name <area=val2> <pj=val3> <m=val4>
   <off> <ic=val5> <pgate=val6>
Daa pos_n neg_n model_name <w=val2> <l=val3> <pj=val3>
   <m=val4> <off> <ic=val4>
```

Parameters

Diode parameters are described in the table below.

Table 19 Diode Parameters

Parameter	Default	Description
Daa		Diode element name, which must begin with the character D.
pos_n		Positive node name.
neg_n		Negative node name.
model_name		Diode model name. It is used to refer to a diode model.
area	1	Area of diode. If area is un-specified, it is calculated from w and l.
w		Width of diode.
l		Length of diode.
pj		Periphery of diode junction. If pj is unspecified, the default value is used and it is calculated from w and l.
m	1	Multiplier for parallel instances of diode.
OFF	ON	Set initial condition to OFF for this element in DC operating point calculation.
ic		Initial voltage across the diode, when uic is specified in .tran statement. The .ic statement overrides this value. Refer to .ic on page 292.
pgate		Length of the side-wall in the AB diffusion area, which is under the gate. In the model card, LG uses this value.

Example

```
DF127 n12 mm99 diode1
```

Diode Model

Syntax

```
.model model_name d <level=val2> <parameter1=val3>
   <parameter2=val4 ...>
```

Parameters

Diode model parameters are described in the table below.

Table 20 Diode Model Parameters

Parameter	Default	Description
model_name		Diode model name. Diode element refers to the model through this name.
d		Keyword to identify diode model.
level	1	Selector for different levels of diode model: ▪ level=1 for junction diode ▪ level=2 for Fowler-Nordheim diode ▪ level=3 for junction diode with geometric processing. ▪ level=6 for JUNCAP2 version 200, 200.1
parameter		Model parameter.

Example

```
.model diode1 d level=1 bv=4.0 cj=2e-9 pb=0.6
```

Bipolar Junction Transistor (BJT)

Syntax

```
Qaa ncol nbase nemit <nsub> model_name <area=><val2>
   <m=val3> <areab=val4> <areac=val5>
```

Parameters

BJT parameters are described in the table below.

Table 21 Bipolar Junction Transistor (BJT) Parameters

Parameter	Default	Description
Qaa		BJT element name, which must begin with the character Q.
ncol		Collector node name.
nbase		Base node name.
nemit		Emitter node name.
nsub		Substrate node name.
model_name		BJT model name. It is used to refer to a BJT model.
area	1	Area multiplying factor.
m	1	Multiplier for parallel instances of BJT.
areab	Area	Base area multiplying factor.
areac	Area	Collector area multiplying factor.

Example

```
Q100 CX BX EX      bjt1      area=1.5

Q50A 11 265 4   20 bipolar1 1.5
```

Bipolar Junction Transistor (BJT) Model

Syntax

```
.model model_name npn|pnp <level=val1> <parameter1=val2>
   <parameter2=val3 ...>
```

Parameters

BJT Model parameters are described in the table below.

Table 22 Bipolar Junction Transistor (BJT) Model Parameters

Parameter	Default	Description
model_name		BJT model name. BJT element refers to the model through this name.
npn|pnp		Keyword to identify either npn or pnp transistor model: select one.
level	1	Selector for different levels of BJT model. ▪ level=1 for Gummel-Poon model ▪ level=4 for VBIC model ▪ level=6 for Mextram Model ▪ level=8 for HICUM Model ▪ level=13 for HICUM0 Model
Parameter		Model parameter.

Example

```
.model bjt1 npn   bf=120 br=30   is=1e-15
```

Table 23 BJT Models

Model Level	Version	BJT Model
1	N/A	Gummel-Poon BJT Model
4	1.15	VBIC-1.15
4	1.2 [default]	VBIC-1.2
6	503	Mextram-503
6	504 [default]	Mextram-504
8	2.1	HICUM version 2.1
8	2.2	HICUM version 2.2
8	2.2.1	HICUM version 2.2.1
13	1.1	HICUM0

Junction Field Effect Transistor (JFET) and Metal Semiconductor Field Effect Transistor (MESFET)

Syntax

```
Jaa   ndr nga nso model_name <<area=>area1 |<l=val2>
   <w=val3>> <m=val4> <dtemp=val5>
```

Parameters

JFET and MESFET parameters are described in the table below.

Table 24 Junction Field Effect Transistor and Metal Semiconductor Field Effect Transistor (JFET and MESFET)

Parameter	Default	Description
Jaa		JFET or MESFET element name, which must begin with the character J.

Table 24 Junction Field Effect Transistor and Metal Semiconductor Field Effect Transistor (JFET and MESFET) (Continued)

Parameter	Default	Description
ndr		Drain node name.
nga		Gate node name.
nso		Source node name.
model_name		JFET or MESFET model name. It is used to refer to a model.
area	1	Area multiplying factor.
l		Gate length of JFET or MESFET.
w		Gate width of JFET or MESFET.
m	1	Multiplier for parallel instances of JFET or MESFET.
dtemp	0	Difference between the transistor temperature and the global circuit temperature in degrees Celsius.

Example

```
J2   4   3   1   jfet1 area=2
```

Junction Field Effect Transistor (JFET) Model

Syntax

```
.model model_name njf|pjf   <level=val2> <capop=val3>
   <parameter1=val4> <parameter2=val5 ...>
```

Parameters

JFET Model parameters are described in the table below.

Table 25 Junction Field Effect Transistor (JFET) Model

Parameter	Default	Description
model_name		JFET model name. JFET element refers to the model through this name.

Table 25 Junction Field Effect Transistor (JFET) Model (Continued)

Parameter	Default	Description
njf|pjf		Keyword to identify either n-JFET or p-JFET model: select one.
level	1	Selector for different levels of JFET model. Level value can be 1, 2, or 3. level=3 model is for MESFETs where the following parameters apply: ▪ SAT=0 selects standard Curtice model. ▪ SAT=1 selects the model that includes vgst in the argument of hyperbolic tangent function. ▪ SAT=2 selects Statz model with constant exponent and denominator. ▪ SAT=3 selects Statz model with parameter values for exponent and denominator.
capop		Capacitor model selector. capop=0 selects default capacitance equation. capop=1 selects Statz capacitance equation for MESFETs.
parameter		Model parameter.

Example

```
.model jfet1 njf    is=2e-14 acm=0   capop=0
```

IF-ELSE Syntax in Netlists

IF-ELSE syntax can be added to condition-controlled netlists to change the circuit topology, expand the circuit, set parameter values for each device instance, or select different model cards in each IF-ELSE block.

The following is a typical example of an IF-ELSE block.

```
.if (condition1)
<statement_block1>
# The following statement block in {braces} is
# optional, and you can repeat it multiple times:
{ .elseif (condition2)
<statement_block2>
}
# The following statement block in [brackets]
# is optional, and you cannot repeat it:
[ .else
<statement_block3>
]
.endif
```

IF-ELSE Block Rules

There are rules that must be observed when using IF-ELSE blocks for condition-controlled netlists.

1. In an `.IF`, `.ELSEIF`, or `.ELSE` condition statement, complex Boolean expressions must not be ambiguous. For example, change `(a==b && c>=d)` to `((a==b) && (c>=d))`.
2. In an `.IF`, `.ELSEIF`, or `.ELSE` statement block you can include most valid HSIM analysis and output statements. The exceptions are:

   `.END`, `.ALTER`, `.MACRO`, `.EOM`, `.GLOBAL`, `.DEL`, `.LIB`, `.MALIAS`, `.ALIAS`, `.LIST`, `.NOLIST`, and `.CONNECT` statements.
3. You can use IF-ELSEIF-ELSE blocks to select different sub-modules or to structure the netlist by using `.INC`, `.LIB`, and `.VEC` statements.
4. If two or more models in an IF-ELSE block have the same model name and model type, they must also be the same revision level.
5. Parameters in an IF-ELSE block do not affect the parameter value within the condition expression. HSIM updates the parameter value only after it selects the IF-ELSE block.
6. You can nest IF-ELSE blocks.
7. You can include an unlimited number of `.ELSEIF` statements within an IF-ELSE block.
8. You cannot include sweep parameters or simulation results within an IF-ELSE block.

9. You can NOT use an IF-ELSE block within another statement. In the following example, HSIM does not recognize the IF-ELSE block as part of the resistor definition:

```
r 1 0
.if (r_val>10k)
+ 10k
.else
+ r_val
.endif
```

10. You can NOT include IF-ELSE block within a `.model` statement.
11. You can use IF-ELSE blocks inside `.subckt` statement; however, you can use only element statements and `.MODEL` inside these IF-ELSE blocks. `.LIB, .MODEL,` or similar commands cannot be place within these IF-ELSE blocks. Please refer to Example 8 on page 234 for the proper usage of IF-ELSE blocks for model binning.
12. No sweep variables or `.ALTER` variables are allowed inside an IF-ELSE evaluation expression. The example below is not allowed.

```
*This is NOT allowed
.dc vg 0 3 0.1 sweep swp_var poi 7 -40 -25 -10 5 20 35 50
.if (swp_var > 0)
…
.else
…
.endif
```

Likewise, the variable name inside the `.ALTER` variable can not be `swp_var`.

13. When `.subckt` is used inside an IF-ELSE block, the same `.subckt` name can have multiple definitions as shown in the example below.

```
.IF (select==1)
.subckt sub1 n1 n2
...
.ends
.ELSE
.subckt sub1 n1 n2 n3
...
.ends
.ENDIF
```

However, the example below can not be supported because the `.subckt` is re-defined.

```
.IF (select==1)
.subckt sub1 n1 n2
...
.ends
.subckt sub1 n1 n2 n3
...
.ends
.ENDIF
```

IF-ELSE Block Syntax Descriptions

Syntax

```
.IF (condition1)
...
<.ELSEIF (condition2)
... >
<.ELSE
... >
.ENDIF
```

Arguments

condition1

The condition must be true before HSIM executes the commands that follow the `.IF` statement.

condition2

The condition must be true before HSIM executes the commands that follow the `.ELSEIF` statement. HSIM executes the commands that follow *condition2*, only if *condition1* is false and *condition2* is true.

Example

```
.IF (a==b)
.INCLUDE /myhome/subcircuits/diode_circuit1
...
.ELSEIF (a==c)
.INCLUDE /myhome/subcircuits/diode_circuit2
...
.ELSE
.INCLUDE /myhome/subcircuits/diode_circuit3
...
.ENDIF
```

Description

HSIM executes the commands that follow the first `.ELSEIF` statement, only if *condition1* in the preceding `.IF` statement is false, and *condition2* in the first `.ELSEIF` statement is true.

If *condition1* in the `.IF` statement and *condition2* in the first `.ELSEIF` statements are both false, then HSIM moves on to the next `.ELSEIF` statement, if there is one. If this second `.ELSEIF` condition is true, HSIM executes the commands that follow the second `.ELSEIF` statement, instead of the commands after the first `.ELSEIF` statement.

HSIM ignores the commands in all false `.IF` and `.ELSEIF` statements, until it reaches the first `.ELSEIF` condition that is true. If no `.IF` or `.ELSEIF` condition is true, HSIM continues to the `.ELSE` statement.

The `.ELSE` statement precedes one or more commands in a conditional block, after the last `.ELSEIF` statement, but before the `.ENDIF` statement. HSIM executes these commands by default, if the conditions in the preceding .IF statement, and in all of the preceding `.ELSEIF` statements in the same conditional block, are all false.

The `.ENDIF` statement ends a conditional block of commands that begin with an `.IF` statement.

IF-ELSE Netlist Examples

The following examples demonstrate some of the ways in which IF-ELSE blocks can be used in condition-controlled netlists.

Example 4 IF-ELSE Block in Subcircuit Definition

```
**********************************************
* This Case is for If...else
* The subckt call in if...else block
.GLOBAL VNODE
** Options
.OPTION POST=2 $PROBE NODE LIST
** Parameter
.PARAM VDD=3v
.PARAM VHI=1.0V  VLO='-VHI'
.PARAM TDELAY=0 TRISE=0.1n TFALL=0.1n TPLAT=5n TPERIOD=10n
** Analysis
.TRAN 10p  20n
** Probe
.PROBE TRAN
+   VIN  =  V(IN)
+   VOUT =  V(OUT)
** Circuit(Source and Load)
VCC    VNODE    0   VDD
VIN  IN    0  PULSE ( VHI VLO TDELAY TRISE TFALL TPLAT TPERIOD )
RIN    IN       0   50
** Subckt Definition
* vth=0.7v
.SUBCKT INV1 IN OUT L=0.1u W=1u PD=4u PS=4u AD=10p AS=10p
MN1   OUT    IN  GND   GND   NCH1  L=L W=W PD=PD PS=PS AD=AD AS=AS
MP1   OUT    IN  VNODE  VNODE  PCH1  L=L W=W PD=PD PS=PS AD=AD AS=AS
.ENDS
.MODEL  NCH1  NMOS LEVEL=49 VERSION=3.2 VTH0=0.8
.MODEL  PCH1  PMOS LEVEL=49 VERSION=3.2 VTH0=-0.8
*vth=1.48v
.SUBCKT INV2 IN OUT L=1u W=20u PD=40u PS=40u AD=20p AS=20p
MN2   OUT    IN   0     0     NCH2 L=L W=W PD=PD PS=PS AD=AD AS=AS
MP2   OUT    IN   VNODE VNODE PCH2 L=L W=W PD=PD PS=PS AD=AD AS=AS
.ENDS
.MODEL  NCH2  NMOS  LEVEL= 2
+   vto = 1.48   kp = 8.0e-05   gamma = 0.6
.MODEL  PCH2  PMOS  LEVEL=2
+   vto = -1.48  kp = 7.5e-05  gamma = 0.8
** If...Else... Block
.IF ( VHI > 1.5 )
    X1 IN OUT INV2 L=2u w=20u
.ELSEIF ( 0.6 < VHI < 1.5 )
    X1 IN OUT INV1
    XAMP OUT 0 AMPOUT AMP
   RL     AMPOUT  0  2K
    .PROBE V(AMPOUT)
.ELSE
    R1 IN OUT 1Meg
```

```
.ENDIF
** Load
RLOAD  OUT  0  50

** AMP
.SUBCKT  AMP  V+  V-  VO
RI  V+  V-  2MEG
RO  VD  VO  75
E1  VD  0  V+  V-  20
.ENDS  AMP

.END
*******************************************************
```

Example 5 IF-ELSE Block to Define Parameter Values

```
*******************************************************
.OPTION  LIST  POST  OPTS
.TEMP  -40  -25  -10  5  20  35  50
.DC  VG  0  3  0.1
.IF (TEMPER>25)
        .PARAM  PVTO = 100m
.ELSE
        .PARAM  PVTO = 200m
.ENDIF
**** MOSFET *****
M1  D  G  S  B  NMOS  L=1u  W=1u
**** Bias Condition ***
VD  D  0  3
VG  G  0  1
VS   S  0  0
VB  B  0  0
.MODEL NMOS  NMOS  LEVEL = 3
+ VTO =  PVTO
.END
*********************************************************
```

Example 6 IF-ELSE Block to Define Model Cards

```
*************************************************************
*  if-else with model cards selected
.option post
vd d 0 3v
vg g 0 1v
vs s 0 0v
vb b 0 0v
.param length=0.25e-6
m1 d g s b nch l='length' w=0.5e-6
*** select model based on channel length
.if ((length>=2.4e-007) && (length<3.5e-007))
     .model nch nmos level=54 version=4.2
     + vth0 = 0.4
.elseif ((length>=3.5e-007) && (length<5e-007))
     .model nch nmos level=54 version=4.2
     + vth0 = 0.42
.endif
.op
.dc vg 0 3 0.1
.print i(vd)
.end
*************************************************************
```

Example 7 Nested IF-ELSE Blocks

```
*********************************************************
.option post list node probe
.probe v(*)

.param v_sw1=1
+      v_sw2=1
.param test1=0 test2=0 test3=0
+      test4=0

.if (v_sw1 == 1)
.param test4=1
    .if (v_sw2 == 1)
      .tran 0.05n 50n
      Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 4.5n 10n)
      .param test1=1
    .elseif (v_sw1 == 2)
    .else
      .tran 0.05n 100n
      Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 9.5n 20n)
      .param test2=1
    .endif
.else
   .tran 0.05n 200n
   Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 19.5n 40n)
   .param test3=1
.endif

Xbuf1 in_in1 out_out1 drv1

Rload1 out_out1 gnd R=10k

.subckt drv1 in out
Rout1 in out0 R=5
Cout1 in gnd C=10p
Rdamp1 out0 out R=1
.ends

.print tran par('test1') par('test2') par('test3')
+           par('test4')
+           par('v_sw1') par('v_sw2')
.end
*********************************************************
```

Example 8 IF-ELSE Blocks for Model Binning

```
*********************************************************
* Test circuit with model binning
.subckt nchi_ll 1 2 3 4
.param length=0.25e-6
*** select model based on channel length
.if ((length>=2.4e-007) && (length<3.5e-007))
m1 1 2 3 4 nch1 l='length' w=0.5e-6
.elseif ((length>=3.5e-007) && (length<5e-007))
m1 1 2 3 4 nch2 l='length' w=0.5e-6
.endif
.ends nchi_ll
.model nch1 nmos level=54 version=4.2
+ vth0 = 0.4
.model nch2 nmos level=54 version=4.2
+ vth0 = 0.42
vdd   vdd gnd 1.0v
vsub  vsub gnd 0.0v
vin   1 gnd dc 1.0v
xt1 2 1 gnd gnd nchi_ll length=2.5e-7
xt2 2 1 gnd gnd nchi_ll length=4.5e-7
.option post
.op
.end
*********************************************************
```

Driving Sources and Input Stimuli

Element statements for independent sources and dependent sources that drive an electronic circuit are summarized in this section.

Independent Voltage Source

Syntax

```
Vaa pos_n   neg_n <<dc=>dcval> <tranfun>
```

Parameters

Independent voltage source parameters are described in the table below.

Table 26 Independent Voltage Source Parameters

Parameter	Default	Description
vaa		Independent voltage source name, which must begin with the character V.
pos_n		Positive node name.
neg_n		Negative node name.
dc		Keyword for DC source value in volt.
dcval	0	The DC voltage source value. If not specified, the default value is used.
transfun		Transient source function.

Example

```
VDD 1       0      3.3V

VBB A10     Gnd    1.5
```

Independent Current Source

Syntax

```
Iaa pos_n   neg_n <<dc=>dcval> <tranfun> <m=val2>
```

Parameters

Independent current source parameters are described in the table below.

Table 27 Independent Current Source Parameters

Parameter	Default	Description
Iaa		Independent current source name. It must begin with the character I.
pos_n		Positive node name.

Table 27 Independent Current Source Parameters (Continued)

Parameter	Default	Description
neg_n		Negative node name.
dc		Keyword for DC source value in ampere.
dcval	0	The DC current source value. If not specified, the default value used.
transfun		Transient source function.
m	1	Multiplier for parallel instances of independent current source.

Example

```
IPX N1 N2   dc 0.005
```

Pulse Source Function (PULSE)

Syntax

```
pulse <(> val1 val2 <t_delay <t_rise <t_fall <pulse_width
   <pulse_period >>>>> <)>
```

Parameters

Pulse source function parameters are described in the table below.

Table 28 Pulse Source Function Parameters

Parameter	Default	Description
pulse		Keyword to identify pulse source function.
val1		Initial value of the pulse source.
val2		Pulse peak value.
t_delay	0.0	Delay time before the first onset ramp. The unit of measure is seconds.
t_rise		Rise time of the pulse. The unit of measure is second.

Table 28 Pulse Source Function Parameters (Continued)

Parameter	Default	Description
t_fall		Fall time of the pulse. The unit of measure is second.
pulse_width		Pulse width. The unit of measure is second.
pulse_period		The pulse period. The unit of measure is second.

Example

```
VIN 1 0 pulse (0 2.5    2N   0.5N 0.5N 5N 10N)

I2   2 0 pulse (0 1e-3   1N   0.5N 0.5N 3N 6N)
```

Sinusoidal Source Function (SIN)

Syntax

```
sin <(> v_dc v_amplitude <freq <t_delay < ATHETA <APHI >>>>
   <)>
```

Parameters

Sinusoidal source function parameters are described in the table below.

Table 29 Sinusoidal Source Function Parameters

Parameter	Default	Description
sin		Keyword to identify sinusoidal source function.
v_dc		Offset value of the source.
V_amplitude		Amplitude value of the source.
freq		Frequency of the source.
t_delay		Delay time before the onset of sinusoidal value. The unit of measure is second.
A_{THETA}	0	Damping factor in unit of 1/second.
A_{PHI}	0	Phase delay in unit of degree.

Example

```
VDA      ax 0 sin (0 3.3   100meg 1N    2e8)

Isource n1 0 sin (0 1e-3 5e6     0N    0    30)
```

Single-Frequency Frequency Modulation (SFFM) Source Function

Syntax

```
sffm <(> v_offset v_amplitude <freq_ca <md_ind <freq_sig>>>
   <)>
```

Parameters

SFFM source function parameters are described in the table below.

Table 30 SFFM Source Function Parameters

Parameter	Default	Description
sffm		Identifies single-frequency FM source function.
v_offset		Offset value of the source.
v_amplitude		Amplitude value of the source.
freq_ca		Carrier frequency in Hz.
md_ind	0	Modulation index.
freq_sig		Signal frequency in Hz.

Example

```
V2    bx 0 sffm (0 1.8   1.0e6   4   10K)
```

Amplitude Modulation (AM) Source Function

Syntax

```
am <(> amplitude offset freq_mod freq_ca <td> <)>
```

Parameters

AM source function parameters are described in the table below.

Table 31 Amplitude Modulation (AM) Source Function Parameters

Parameter	Description
amplitude	Signal amplitude.
offset	Offset value.
freq_mod	Modulation frequency in Hz.
freq_ca	Carrier frequency in Hz.
td	Delay time.

Example

```
VDA    ax 0 am (1.8 0   1k 100k   0.5m)
```

Exponential Source Function (EXP)

Syntax

```
exp <(> val1 val2 <td_rise <tr_const <td_fall <tf_const >>>>
   <)>
```

Exponential source function parameters are described in the table below.

Table 32 Exponential Source Function Parameters

Parameter	Default	Description
exp		Keyword to identify exponential source function.
val1		Initial value of the exponential source. The unit of measure is volt.
val2		Pulse peak value. The unit of measure is volt.
td_rise	0	Delay time for rising edge. The unit of measure is second.
td_fall		Delay time for falling edge. The unit of measure is second.
t_fall		Fall time of the pulse. The unit of measure is second.

Table 32 Exponential Source Function Parameters (Continued)

Parameter	Default	Description
tr_const		Time constant for rising edge. The unit of measure is second.
tf_const		Time constant for falling edge. The unit of measure is second.

Example

```
V77       77   0 exp (0   2.5    2N   30N   2N   40N)

Isource   n1   0 exp (0 1e-3    2N   15N   2N   30N)
```

Piecewise Linear Source Function (PWL)

Syntax

```
pwl <(> t1 val1 <t2 val2 ...tN valN> <r <=t_REPEAT>> <td=valm
   > <)>
```

Parameters

Piecewise linear source function parameters are described in the table below.

Table 33 Piecewise Linear Source Function Parameters

Parameter	Default	Description
pwl		Keyword to identify piecewise linear source function.
tk		kth time point.
valk		Source value at kth time point.
r		Keyword to identify repeat function.
t_REPEAT	0	Start time for repeat function which must be less than the greatest time point tN. The unit of measure is second.
td		Delay time.

Example

```
VXD 5 0 pwl (0 0 2N 3.3   10N 3.3 12N 0 20N 0 22N 3.3)
```

Piecewise Linear Source Function with High Impedance State (PWLZ)

Syntax

```
pwlz <(> t1 val1   <t2 val2   t3 z t4 val4 ...> <r <=t_REPEAT
   >> <td=val5 > <)>
```

z can be used in place of source value. The voltage source will be disconnected for time periods marked with the keyword z.

Example

```
VXD 5 0 pwlz (0   0 2N 0.75 10N 1.5 50N z 60N 0.75)
```

In this example, node 5 is connected to a 0V source at time 0, and rises from 0V to 0.75V in 2 ns. Between 2 ns and 10 ns, the voltage source value rises from 0.75V to 1.5V. The voltage source value stays at 1.5V between 10 ns and 50 ns. Starting from 50 ns, node 5 is disconnected from the voltage source until 60 ns. It is connected to a 0.75V voltage source after 60 ns.

Voltage-Controlled Current Source (VCCS)

VCCS syntax statements are described below:

Linear Syntax

```
Gaa pos_n   neg_n   <vccs> nc+   nc- transconductance
   <max=val2> <min=val3> <m=val4>
```

Polynomial Syntax

```
Gaa i <vccs > poly(N)   nc1+   nc1- ..... <ncN +   ncN->
   <min=val2> <max=val3> <m=val4> p0 <p1 ....>
```

Piecewise Linear Syntax

```
Gaa pos_n neg_n <vccs > pwl(1) nc1+ nc1- <delta=val2>
   <m=val3> x1, y1, x2, y2 ......
```

Multi-Input Gates Syntax

```
Gaa pos_n neg_n <vccs > logic_gate(m) nc1+ nc1- .... ncm+
   ncm- <scale=val5> <m=val6> <delta=val2>
```

Behavioral Current Source Syntax

```
Gaa   pos_n   neg_n   <vccs >   cur="expr"
```

Parameters

VCCS parameters are described in the table below.

Table 34 Voltage-Controlled Current Source (VCCS) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled current source name which must begin with the character G.
pos_n		Positive node name for controlled source.
neg_n		Negative node name for controlled source.
vccs		Identify voltage-controlled current source.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
transconductance		Transfer factor from voltage-to-current.
max		Maximum current value.
min		Minimum current value.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1, ...		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.
m		Multiplier for parallel instances.

Table 34 Voltage-Controlled Current Source (VCCS) Parameters (Continued)

Parameter	Default	Description
logic_gate		Choose one from reserved words: ▪ AND ▪ NAND ▪ OR ▪ NOR
cur		Keyword to identify current output.
expr		A mathematical expression defining the current from node pos_n to node neg_n. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.

Example

```
GXY 5   0   1   0    0.002
```

Voltage-Controlled Voltage Source (VCVS)

VCVS syntax statements are described below:

Linear Syntax

```
Eaa pos_n neg_n <vcvs> nc+ nc- vgain <min=val2> <max=val3>
```

Polynomial Syntax

```
Eaa pos_n  neg_n <vcvs > poly(N) nc1+ nc1- ... <ncN+  ncN->
   <min=val2> <max=val3> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Eaa pos_n neg_n <vcvs > pwl(1) nc1+ nc1- <delta=val2> x1,
   y1, x2, y2 ...
```

Multi-Input Gates Syntax

```
Eaa pos_n   neg_n <vcvs > logic_gate(m) nc1+   nc1- ....
   ncm+   ncm- <delta=val2>
```

Delay Element Syntax

```
Eaa pos_n neg_n <vcvs > delay nc+ nc- td=td1
```

Behavior Voltage Source Syntax

```
Eaa pos_n neg_n <vcvs > <scale=val5> <m=val6> vol="expr"
```

VCVS Parameters

VCVS parameters are described in the table below.

Table 35 Voltage-Controlled Voltage Source (VCVS) Parameters

Parameter	Default	Description
Eaa		Voltage-controlled voltage source name, which must begin with the character E.
pos_n		Positive node name for controlled source.
neg_n		Negative node name for controlled source.
vcvs		Keyword to identify voltage-controlled voltage source.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
vgain		Voltage gain factor.
max		Maximum voltage value.
min		Minimum voltage value.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1, ...		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.

Table 35 Voltage-Controlled Voltage Source (VCVS) Parameters (Continued)

Parameter	Default	Description
logic_gate		Choose one from reserved words ▪ AND ▪ NAND ▪ OR ▪ NOR
delay		Delay element.
td		Delay time.
vol		Keyword to identify voltage output.
expr		A mathematical expression defining the current from node pos_n to node neg_n. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.

Example

```
EXY 5   0   1   0    0.5
```

Ideal Transformer

Syntax

```
Exxx n+ n- <transformer> in+ in- k
```

Parameters

Ideal transformer parameters are described in the table below.

Table 36 Ideal Transformer Parameters

Parameter	Description
Exxx	**Ideal transformer name. The name must begin with E.**
n +/-	Positive or negative control element nodes.
in +/-	Positive or negative controlling nodes.

Table 36 Ideal Transformer Parameters

Parameter	Description
Exxx	**Ideal transformer name. The name must begin with E.**
k	Ideal transformer turn ratio: V(in+, in-)=k * V(n+, n-)

Example

```
E1 t1   t2   transformer   b1    b2    5
```

Current-Controlled Current Source (CCCS)

CCCS syntax statements are described below:

Linear Syntax

```
Faa pos_n neg_n <cccs > vc igain <min=val2> <max=val3>
   <m=val4>)
```

Polynomial Syntax

```
Faa pos_n neg_n <cccs > poly(N) vc1 ... vcN <min=val2>
   <max=val3> <m=val4> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Faa pos_n neg_n <cccs > pwl(1) vc <delta=val2> <m=val3> x1,
   y1, x2, y2 ...
```

Multi-Input Gates Syntax

```
Faa pos_n neg_n <cccs > logic_gate(m) vc1 ... vcm
   <delta=val2> <m=val3>
```

Parameters

CCCS parameters are described in the table below.

Table 37 Current-Controlled Current Source (CCCS) Parameters

Parameter	Default	Description
Faa		Current-controlled current source name which must begin with the character F.
pos_n		Positive node name for controlled source.

Table 37 Current-Controlled Current Source (CCCS) Parameters (Continued)

Parameter	Default	Description
neg_n		Negative node name for controlled source.
cccs		Keyword to identify current-controlled current source.
vc		Voltage source name for the controlling current to flow. Use one for each dimension.
igain		Current gain factor.
max		Maximum current value.
min		Minimum current value.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.
m		Multiplier for parallel instances.
logic_gate		Choose one from reserved words: ▪ AND ▪ NAND ▪ OR ▪ NOR
xk		Controlling current through vc source.
yk		Corresponding output current of xk.

Example

```
F1 5 0 VIN   0.2
```

Current-Controlled Voltage Source (CCVS)

CCVS syntax statements are described below:

Linear Syntax

```
Haa pos_n neg_n <ccvs > vc transresistance <min=val2>
   <max=val3>
```

Polynomial Syntax

```
Haa pos_n neg_n <ccvs > poly(N) vc1 ..... vcN <min=val2>
   <max=val3> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Haa pos_n neg_n <ccvs > pwl(1) vc <delta=val2> x1, y1, x2,
   y2 ...
```

Multi-Input Gates Syntax

```
Haa pos_n neg_n vcr logic_gate(m) vc1+ vc1- ... vcm+ vcm-
   <delta=val2> x1, y1, x2, y2 ...
```

Parameters

CCVS parameters are described in the table below.

Table 38 Current-Controlled Voltage Source (CCVS) Parameters

Parameter	Default	Description
Haa		Voltage-controlled resistor name which must begin with the character H.
pos_n		Positive node name.
neg_n		Negative node name.
ccvs		Keyword to identify voltage-controlled resistor.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.

Table 38 Current-Controlled Voltage Source (CCVS) Parameters (Continued)

Parameter	Default	Description
p0, p1, ...		Polynomial coefficients.
poly		Polynomial controlling function.
pwl		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
logic_gate		Choose one from reserved words: ▪ AND ▪ NAND ▪ OR ▪ NOR
xk		Controlling voltage through vc source.
yk		Corresponding output element value.

Example

```
H23 5 0 VIKIN   121.0
```

Voltage-Controlled Resistor (VCR)

VCR syntax statements are described below:

Linear Syntax

```
Gaa pos_n neg_n vcr vc+ vc- transfactor
```

Polynomial Syntax

```
Gaa pos_n neg_n vcr poly(N) vc1+ vc1- ... <vcN+   vcN-> P0
   <P1 ...>
```

Piecewise Linear Syntax

```
Gaa pos_n neg_n vcr pwl(1) vc+ vc- <delta=val2> x1, y1, x2,
   y2 ...
```

Multi-Input Gates Syntax

```
Gaa pos_n neg_n vcr logic_gate(m) vc1+ vc1- ... vcm+ vcm-
   <delta=val2>
```

Parameters

VCR parameters are described in the table below.

Table 39 Voltage-Controlled Resistor (VCR) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled resistor name which must begin with the character G.
pos_n		Positive node name.
neg_n		Negative node name.
vcr		Keyword to identify voltage-controlled resistor.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
p0, p1, ...		Polynomial coefficients.
poly		Polynomial controlling function.
pwl		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
logic_gate		Choose one from reserved words: ▪ AND ▪ NAND ▪ OR ▪ NOR
xk		Controlling voltage through vc source.

Table 39 Voltage-Controlled Resistor (VCR) Parameters (Continued)

Parameter	Default	Description
yk		Corresponding output element value.

Example

```
Gsw3   3    0 vcr    PWL(1) 2   0   0V,   1K,   1V, 10K
```

Voltage-Controlled Capacitor (VCC)

Syntax

```
Gaa pos_n neg_n vccap pwl(1) nc+ nc- <delta=val2> x1, y1,
   x2, y2 ... <tc1=val3 <tc2=val4>> <scale=val5> <m=val6>
   <ic=val7>
```

Parameters

VCC parameters are described in the table below.

Table 40 Voltage-Controlled Capacitor (VCC) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled capacitor name, which must begin with the character G.
pos_n		Positive node name.
neg_n		Negative node name.
vccap		Keyword to identify voltage-controlled capacitor.
nc+/-		Positive or negative controlling node.
pwl		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
xk		Controlling current through vc source.
yk		Corresponding output current of xk.

Table 40 Voltage-Controlled Capacitor (VCC) Parameters (Continued)

Parameter	Default	Description
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1.0	Element scale parameter.
m	1	Multiplier for parallel instances of capacitor.
ic		Initial voltage across the capacitor. The .ic Statement will override this value.

Laplace Element

A laplace element performs transformation of an input signal to output signal given by an s-domain rational function R(s) as shown in this Equation :

$$R(s) = \frac{Pn(s)}{Qm(s)}$$

where numerator Pn(s) and denominator Qm(s) are polynomials in complex frequency s with real coefficients of power n and m respectively as shown in Equation 9:

Example 9

```
Pn(s)=a0 + a1*s + a2*s^2 + ... + an*s^n,
Qm(s)=b0 + b1*s + b2*s^2 + ... + bm*s^m.
```

These polynomials can be specified either through their coefficients or through their roots. The power of the denominator should be larger then the power of the numerator.

The input and output signals can have the following meanings:

Input signal

Node voltage

Output signal

Node voltage or branch current. Depending on which type of output signal is used (voltage or current) either of the following laplace elements are possible:

- VCVS
- VCCS

If rational function R(s) is specified through polynomial coefficients, then the following syntax is used for Laplace element as shown in the following syntax:

Syntax

```
Gname out1 out2 LAPLACE in1 in2 a0 a1 a2...an /
   b0 b1...bm <scale=value>
Ename out1 out2 LAPLACE in1 in2 a0 a1 a2...an /
   b0 b1...bm <scale=value>
```

Parameters

In this syntax, the following definitions apply:

Line 1

Specifies VCCS

Line 2

Specifies VCVS

out1 and out2

Output nodes

in1 and in2

Input nodes

LAPLACE

A keyword to distinguish the element

a0 ... an

Coefficients of polynomial of power n in numerator

b0 ... bm

Coefficients of polynomial of power m in denominator.

scale

Optional parameter used as a multiplier for output.

/

Separator between numerator and denominator coefficients.

Example

```
g1 0 m1 LAPLACE n1 0 1 / 1 0.2e-9 scale=2
e2 m2 0 LAPLACE n1 0 1 / 1 0.2e-9 scale=2
```

Rational function R(s) can be specified through the polynomial's root. Because polynomials have real coefficients, their roots should be either real or complex conjugate. In case of multiple roots they must be specified as many times as the multiplicity of the roots.

Syntax

```
Gname out1 out2 POLE in1 in2 a az1,fz1 ... azn,fzn /
   b bp1,fp1 ... bpm,fpm <scale=value>
Ename out1 out2 POLE in1 in2 a az1,fz1 ... azn,fzn /
   b bp1,fp1 ... bpm,fpm <scale=value>
```

Parameters

In this syntax, the following definitions apply:

Pn(s)=a*(s+az1-2*pi*j*fz1)*...*(s+azn-2*pi*j*fzn),

Numerator polynomial

Qm(s)=b*(s+bp1-2*pi*j*fp1)*...*(s+bpn-2*pi*j*fpn),

Denominator polynomial

pi=3.1415926535

j

An imaginary unit

Comma

Separates real and imaginary parts of roots. Commas are optional.

scale

A multiplier for output variable. If polynomial is constant has only zero root, it can be omitted.

/

Separator between numerator and denominator coefficients.

Example

```
e3 m3 0 POLE n3 0 5e9 / 1 +5e9, 0
g4 0 m4 POLE n4 0 25e18 / 1 +5e9, -1e8 +5e9, +1e8 scale=2
```

Digital Vector File

Digital vector input and output files for HSIM are specified with the HSIMVECTORFILE on page 166 command. A digital vector file consists of a vector definition section and a vector data section. The vector definition section describes the signals for each of the following:

- Vector stimulus
- Vector type (input stimulus / expected digital output)
- Rise/fall time
- Driving strength
- Cycle period (optional)

The vector data section describes the signal states at specified times in tabular format. The vector definition section must be placed before the vector data section. Long lines in vector file can be split by putting a back-slash (\) at the end of a line or a plus sign (+) at the beginning of next line. A line beginning with a semi-colon character ; is treated as a comment line.

The global parameters defined in the simulation file can be referenced inside the vector file. This feature is available to the value filed of the following commands:

- delay
- rise
- fall
- slope
- period
- check_window (start/stop)
- tunit
- logichv
- logiclv
- vhth
- vlth
- resistance

Example

```
/* in simulation file */
.param LOGICHV_PARAM=2.7
.param LOGICLV_PARAM=0.3
……
/* in vector file */
logichv LOGICHV_PARAM
; define logichv to be 2.7 V
logiclv   LOGICLV_PARAM
; define logiclv to be 0.3 V
……
```

Vector Statements

Following are descriptions of the vector statements.

check_window

The check_window statement is to define a time window around the vector strobe time or user-defined first_time such that the output comparison, for signals specified as output in .io statement, is checked over this time window.

Syntax

```
check_window start_offset stop_offset steady <mask_name>
```

or

```
check_window start_offset stop_offset steady period_time
   first_time <mask_name>
```

Description

In the first syntax statement above, the values specified by start_offset and stop_offset define the time window as [t-start_offset, t+stop_offset] in which t is the vector stop time. The unit of time for start_offset and stop offset is nanosecond. When steady is specified as 1, the comparison check passes if the output state matches with the expected state throughout the time window period. When steady is specified as 0, the output comparison passes as long as the output state ever reaches the expected state at any time within the window. mask_name is optional. When mask_name is specified, check_window applies to the signals defined under mask_name only.

In the second syntax statement above, the values specified by start_offset and stop_offset define the time window as [t-start_offset, t+stop_offset] in which t is the first time. The checking will be repeated every period_time. The unit of time for start_offset, stop_offset, period_time, and first_time is nanosecond. When

steady is specified as 3, the comparison check passes if the output state matches with the expected state throughout the time window period. When steady is specified as 2, the output comparison passes as long as the output state ever reaches the expected state at any time within the time window. mask_name is optional. When mask_name is specified, check_window applies to the signals defined under mask_name only.

Examples

```
signal         a    b    c    d
radix          1    1    1    1
io             i    o    o    i
check_window   1.5  2.0  1
               1    1    0    1
     ....................


signal         clk  D2   D3
radix          1    1    1
io             i    o    o
mask           m1   D2
check_window   0.2  0.2  3 10 4    m1

0    0    X    X
5    1    0    1
6.5  1    1    1
8    1    0    0
10   0    0    1
10.2 0    1    0
15   1    1    0
15.2 1    0    1
17.3 1    1    1
18.6 1    0    0
20   0    0    0
. . . .
```

In this example, the output comparison on signal D2 passes if the logic state of D2 is 0 between 3.8ns to 4.2ns, and the logic state of D2 is 1 between 13.8ns to 14.2ns, and so on.

delay|tdelay

Description

The delay statement defines the timing shift between the actual vector time and the vector time defined in the vector file. The statement begins with a keyword delay or tdelay followed by the delay time. The unit of time is nanosecond. The

delay value can be either positive or negative. If the delay value is negative, then the actual vector time is sooner than the specified vector time.

Syntax

```
delay|tdelay   time1
```

Example

```
delay    2
```

This shifts each vector time by 2 ns later.

enable

Description

The enable statement specifies the controlling signal of a bidirectional signal as shown in the example below.

Example

```
radix 1 1 1 1 1 1
vname SCL SDA TEST1 TEST2 VEP_PAD A
io b b i i b i
enable A 0 1 0 0 0 0
```

This enable statement indicates that whenever signal A becomes 1, the SDA will be in the output state. To reverse the control logic, add a tilde character (~) in front of control signal: ~A.

io

Description

The I/O statement defines the type of each column of signal node(s). The statement starts with the keyword io followed by the type for each column. A signal type can either be one of the following:

- i: As input stimulus to the circuit.
- o: As expected output to compare with simulated output.
- u: Unused.
- b: Bidirectional nodes

Spaces can be inserted between columns to improve readability. If the io statement is not specified, all the signals are assumed to be input signals.

Syntax

```
io dddd dddd ...
```

Example

```
io iiii oooo uuuu bb
```

The io statement indicates the signal nodes in the first 4 columns are input signals and the nodes in the next columns are outputs that require output comparison. The last two nodes are bi-directional. The data assigned with a z-state will mark a high impedance bidirectional state when the output signal emerges from the bidirectional terminal.

logichv or vih|logiclv or vil|logicxv

Description

Note:

For compatibility with vector file statements of other simulators, the following are interchangeable:

- logichv and vih
- logiclv and vil
- logicxv

The logichv (or vih), logiclv (or vil), and logicxv statements have the following characteristics:

- Specify the voltage values of logic 1, logic 0, and logic x for all input stimuli.
- Begin with the keyword logichv (or vih) or logiclv (or vil) followed by a voltage value.
- If the unit is not specified, volt is used as the default unit.
- logichv (or vih), logiclv (or vil) can also be specified from parameters HSIMLOGICHV and HSIMLOGICLV.
- If the logichv and logiclv values are not specified in the vector file or by using HSIM parameters, the default values will be used:
  - 3V for logichv
  - 0V for logiclv
  - logiclv for logicxv

A specific voltage can be specified when x is encountered in the vector file.

Syntax

```
logichv|vih|logiclv|vil val2
```

Examples

```
logichv 2.1V
logiclv 0.2V
```

mask

The values specified by slope/rise/fall/logichv/logiclv/vhth/vlth/delay define the analog waveform shape of each signal in the vector file. This is used when signals require values other than the globally defined ones. The mask pattern can be specified to define those selected signals.

The mask statement defines the mask name to represent the mask pattern. The statement starts with a keyword mask, followed by a mask name and then the mask pattern. The mask pattern can be specified by either the values or the signal nodes. If the signal nodes are used to describe the mask pattern, a logic 1 is set to each signal node at the corresponding column location. Any unspecified column is default to logic 0.

Syntax

```
mask  mask_name  dddd dddd ...
mask  mask_name  node1 <node2 ... >
```

Examples

```
signal    a    b    c    d
mask      m1   01   0    1
mask      m2   b    d
```

The above statements specify that both m1 and m2 have a mask pattern of 0101.

For those statements which take an optional mask pattern, the pattern can be specified by either the values or the predefined mask name.

```
signal    a b[0-3] c d e[0-15]
radix     14114444
logichv   3.0
logichv   2.5 0f000000
logichv   2.8 m3
```

This example defines the logic 1 voltage for signals a, d, e[0-7] and e[12-15] as 3.0V. The logic 1 voltage for b[0-3] is 2.5V. The logic 1 voltage for signals c and e[8-11] is 2.8V. The mask pattern for m3 is 001000f0.

period

Description

The period statement is used to define the cycle time between two consecutive vector patterns in the vector data section. When the period time is defined, the first column data of each vector pattern should start with digital logic pattern such as 1 and 0. In such case, the i-th vector pattern is implicitly defined to be at time i-1 times the cycle time defined in the period statement. If an extra vector needs to be inserted in a cycle, use the pound # character to define a delay time from its predecessor.

Syntax

```
period time1
```

Example

```
period    25
period    25ns
period    0.025us
tskip
```

Each period statement above defines the cycle time to be 25 ns. tskip is described below.

```
==================================
period 10
0101 <-- 0ns
#5 0101 <-- vector changed at 5n, which is relative to previous line
1110 <-- 10ns
0101 <-- 20ns
==================================
```

radix

Description

The radix statement specifies the number of signal nodes associated with the column location in the vector file. Each column can specify up to 4 signal nodes, so the valid radix values are from 1 to 4. The statement starts with a keyword radix, followed by a list of numbers from 1 to 4, which represents the radix of the corresponding column. Spaces can be inserted between the numbers. This does not affect functionality, however, it improves readability. This statement must be specified in the vector file.

Syntax

```
radix   dddd dddd …
```

Example

```
radix   1111 4444 1234
```

The above radix statement indicates there are 12 columns with a total of 30 signal nodes in the vector file. Each digit represents one column and the number of nodes in that column.

resistance|out|outz

Description

Output resistance can be specified for all the input signals in a vector file by the resistance statement. The statement starts with the keyword resistance (out/outz) followed by a value for output resistance. If no unit is specified in the resistance value, the default unit is ohms. The default output resistance is 0 ohm.

Syntax

```
resistance|out|outz rval2
```

Example

```
resistance 10
```

signal|vname

Description

Note:

For compatibility with vector files of other simulators, signal and vname are interchangeable.

The signal|vname statement describes all the nodes that this vector file is driving or checking. The statement starts with the keyword signal|vname followed by a list of node names. The number of node names specified in the signal|vname statement must be the same as the total number of signal nodes defined in the radix statement. Bus notation is accepted in the node name specification. The followings are the bus notation accepted by HSIM.

```
[N-M], [N:M], <N-M>, <N:M>
```

The signal|vname statement must be specified in the vector file.

Syntax

```
signal|vname   node1 <node2 ...>
```

Examples

```
signal n1 n2 n3 data[0-7]

vname n1 n2 n3 data[0-7]
```

The above signal|vname statement lists 11 nodes in the vector file as shown below.

```
n1 n2 n3 data[0] data[1] data[2] data[3] data[4] data[5] data[6]
data[7]
```

To interpret the bus signal without brackets ‘[‘and ‘]’, the bus is represented using the notation [N~M].

```
signal n1 n2 n3 data[0~7]
vname n1 n2 n3 data[0~7]
```

The above signal statement lists 11 nodes in the vector file in the following order:

```
n1 n2 n3 data0 data1 data2 data3 data4 data5 data6 data7
```

The bus notation of vector file format is slightly different between HSPICE and HSIM. In order to make it compatible the HSIMHSPICEVEC on page 82 command is introduced. The default value is 0.

```
vname ck[1:3]
.param HSIMHSPICEVEC=0
```

This bus is recognized as "ck[1]" "ck[2]" "ck[3]".

```
vname ck[1:3]
.param HSIMHSPICEVEC=1
```

This bus is recognized as "ck1" "ck2" "ck3" to be compatible to HSPICE notation.

slope|rise|trise|fall|tfall

Description

Rise and fall times can be specified for input signals by the following statements:

- slope: The slope statement starts with a keyword slope followed by a value for the rise and fall time.
- rise: The rise statement starts with the keyword rise (or trise) followed by a value for the rise time.
- fall: The fall statement starts with the keyword fall (or tfall) followed by a value for the fall time.

The slope, rise, and fall times have the following characteristics:

- If no unit is specified in the time, the default tunit is 1 nanosecond.
- The rise/fall statements overwrite the slope statement.
- If no rise/fall time is specified, the default is 1 ps.

Syntax

```
slope|rise|trise|fall|tfall time1
```

Examples

```
slope 10ps
```

The rise and fall times for the input signals are both 10 ps.

Note:

A space between 10 and ps is not permissible.

```
rise 0.1
```

The rise time for the input signals is 0.1 ns, and the default fall time is 1 ps.

stop_at_error

stop_at_error is used to stop circuit simulation whenever output comparisons are performed and mismatched outputs occur.

tskip

tskip is only used when the period statement is defined in the vector file. tskip skips the vector time for each vector pattern defined in the first vertical column. The vector time for the i-th vector row is i-1 times the cycle time defined in the period statement.

Syntax

```
tskip
```

Examples

Refer to the example for period on page 261.

triz

triz specifies the output impedance, when the signal (for which the mask applies) is in tristate; it applies only to the input signals. triz has no effect on the output signals

Note:

If you do not specify the tristate impedance of a signal in a triz statement, 1000M is assumed.

If you apply triz more than once to a signal the last statement overrules the previous statements and a warning is issued.

Syntax

```
triz <output_impedance>
```

Examples

```
triz 15.1Meg
triz 150Meg 1 1 1 0000 00000000
triz 50.5Meg 0 0 0 137F 00000000
```

The first triz statement sets the high impedance resistance globally, at 15.1 Mohms. The second triz statement increases the value to 150 Mohms, for vectors1 to 3. The last triz statement increases the value to 50.5 Mohms, for vectors 4 through 7.

tunit

The tunit statement defines the unit of time in the vector file. The default value is nanosecond; written as 1.0e-9 second.

Syntax

```
tunit     time_unit1
```

Examples

```
tunit     1.0e-8
tunit     10ns
tunit     0.01u
```

Each tunit statement above defines the unit of time in the vector file as 10 ns.

Note:

The tunit statement will affect the default time units of all parameters within the same vector file.

```
tunit 1ps
delay 1.0
```

The amount of delay time is 1 ps because the default time unit has been changed from 1 ns to 1 ps by the tunit statement.

vchk_ignore

The vchk_ignore command causes checking to be ignored for all nodes specified when using the optional mask between times specified by t1 and t2.

Syntax

```
vchk_ignore t1 t2 <mask>
```

Parameters

t1

t1 is the start time.

t2

t2 is the end time.

mask

<mask> is an option.

If <mask> is not specified, all signals between t1 and t2 will be ignored. To ignore selected signals over the entire time period, specify the t1 start and t2 end times. This command can be repeated for cumulative effect.

Example

```
vchk_ignore 0 2 0101
```

In this example, the apply t2 to the specified mask from 0 to 2 ns.

vhth|voh|vlth|vol

Description

Note:

For compatibility with vector file statements of other simulators, the following are interchangeable:

- vhth and voh
- vlth and vol

The vhth (or voh) and vlth (or vol) statements have the following characteristics:

- Specify the threshold voltages of logic HIGH and logic LOW states for the expected output check signals.
- The statement begins with the keyword vhth or vlth, followed by a voltage value.
- If the unit is not specified, volt is used as the default unit.
- The vhth (or voh) and vlth (or vol) can also be specified from HSIM parameters HSIMVHTH on page 167 and HSIMVLTH on page 169.
- If vhth and vlth are not specified in the vector file or set by the HSIMVHTH and HSIMVLTH parameters in the netlist, the following will be set:
  - vhth will be set to 70% of voltage between logichv and logiclv
  - vlth will be set to 30% of voltage between logichv and logiclv

Syntax

```
vhth|voh|vlth|vol val2
```

Example

```
vhth 1.8
vlth 0.3
```

vref

Similar to delay, vref specifies the name of the reference voltage for each input vector to which the mask applies. It applies only to input signals. vref has no effect on output signals.

Note:

If you do not specify the reference voltage name of the signals in a vref statement, HSIM assumes 0.

If more than one vref statement is applied, the last statement overrules the previous statements, and a warning is issued.

Syntax

```
vref <reference_voltage>
```

Examples

```
VNAME v1 v2 v3 v4 v5[[1:0]] v6[[2:0]] v7[[0:3]] v8 v9 v10
VREF 0
VREF 0 111 137F 000
VREF vss 0 0 0 0000 111
```

When HSPICE or HSPICE RF implements these statements into the netlist, the voltage source realizes v1:

```
v1 V1 0 pwl(......)
```

as well as v2, v3, v4, v5, v6, and v7.

However, v8 is realized by

```
V8 V8 vss pwl(......)
```

v9 and v10 use a syntax similar to v8.

vth

Similar to tdelay, vth specifies the logic threshold voltage for each output signal to which the mask applies. The threshold voltage determines the logic state of output signals for comparison with the expected output signals. vth has no effect on the input signals.

Note:

If you do not specify the threshold voltage of the signals in a vth statement, HSIM assumes 1.65.

If you apply more than one VTH statement to a signal, the last statement overrules the previous statements, and a warning is issued.

Syntax

```
VTH <logic-threshold_voltage>
```

Example

```
VTH 1.75
VTH 2.5 1 1 1 137F 00000000
VTH 1.75 0 0 0 0000 11111111
```

In the above example, the first VTH statement sets the logic threshold voltage at 1.75V. The next line changes that threshold to 2.5V, for the first 7 vectors. The last line changes that threshold to 1.75V, for the last 8 vectors.

All of these examples apply the same vector pattern and both output and input control statements. Therefore, the vectors are all bidirectional.

Dynamic vhi and vlo for Logic HIGH and Logic LOW States

The vhi and vlo vector file statements define threshold voltages for the logic HIGH and logic LOW states of the expected output check signals. This HIGH and LOW logic values are static or fixed state voltages.

VCD Direct-Read Feature

A VCD file is an open industry standard ASCII format file, generated by a test pattern generator, RTL simulator, or other high-level simulator. A VCD file does not contain signal direction or waveform characteristics. Therefore, you are required to create a signal information file to map the signal names in the VCD file to the node names in the design netlist, and also to provide this signal information for each signal to HSIM.

A VCD file contains three sections:

- The *header information* section describes the date, the version number of the simulator, and the time-scale used.
- The *variable definitions* section contains the scope of the hierarchy, and type of variables. A variable can be a scalar or a bus. Each variable is represented by a unique identifier character.
- The *value changes* section contains the actual value changes for all variables specified at each simulation time increment. Only the variables, which change during a time increment, are listed. Each variable with a set of values forms a vector over time.

To read-in a VCD file to HSIM, use the HSIMVCD2VEC on page 164 command.

Using the Signal Information File

The signal information file provides information to map the variable names in the VCD file to the node names in the design netlist. All signal names in the signal information file are case-sensitive because Verilog variable names are case-sensitive. If multiple expressions match the same signal, the last match takes priority. To avoid mapping problems, always use unique matches.

The following procedures enable you to create the signal information file:

1. Define bus syntax in the design netlist.

See the Defining Bus Syntax with the #format Command section.

2. Define signal directions, such as input, output, or bi-directional.

   See the Defining Signal Directions section.

3. Map the variable names in the VCD file to the node names in the design netlist.

   See the Defining Mapping Information with the #alias Command section.

4. Define analog waveform characteristics, such as *rise time* and *fall time*.

   See the Defining Attributes for Signals section.

Defining Bus Syntax with the #format Command

The `#format` command specifies the bus naming format. The default name format, if the `#format` command is not specified, is `%[#]`. For a character descriptions, see Table 41 on page 270.

Table 41 % and # character descriptions

Character	Description
%	Represents the variable names
#	Represents the bus indexes

See the following syntax:

`#format|#fo` *`bus_naming_format`*

Only one `#format` command is allowed in a signal information file. Create another signal information file, if multiple formats are needed.

Example 10 displays the bus line format in a VCD file.

Example 10 Bus line format command example

```
$var reg 3 k tA [2:0] $end
```

The `#format` command causes `tA` to expand to the following names in the VCD file.

For a description of the commands and corresponding mapping names, see Table 42.

Table 42 #format command mapping names

Command	Mapping name
`#format %<#>`	`tA<2>  tA<1>  tA<0>`
`#format %.#`	`tA.2  tA.1  tA.0`
`#format %[#]`	`TA[2]  tA[1]  tA[0]`

Defining Signal Directions

You must define signal directions to the variable names in the VCD and EVCD files. Signal directions can be input signals, output signals, or bi-directional signals. HSIM uses all input, output, and bi-directional signals specified in the VCD file for the simulation. HSIM uses input signals as stimulus for the simulation. Simulation output signals are checked against simulation output signals for any mismatch. Bidirectional signals require `enable` signal(s) to control the direction, either as input signals or output signals. All other signals in the VCD file, which are not specified in the signal information file, are ignored.

NOTE: Bi-directional signals cannot be included within an `enable_signal` logic expression.

The `#bi-directional enable_signal` option (see Example 11) can be specified as a logic expression, as shown in Example 12.

Example 11 Bi-directional enable_signal

```
#input | #in signal_name(s)
#output | #out signal_name(s)
#bidirectional | #bi signal_name(s) [in | out enable_signal(s)]
```

Example 12 Logic expression of bi-directional enable_signal

```
#format %[#]
#in cen oen
#in a[[3:0]]
#out c b*
#bi io[[7:0]] (out cen & oen)
```

Example 12 declares the following (description of each line):

- bus `format`
- `in`put signals `cen` and `oen`
- `in`put bus signal `a[[3:0]]`

  The outer bracket signifies bus notification, while the inner bracket signifies a wild card range.
- `out`put signal `c` and bus `b`

  The asterisk character (`*`) is a wild card for any group of characters.
- `bi`-directional signal bus `io[[7:0]]` is controlled by signals `cen` and `oen`
- bus `io[[7:0]]` is output when both `oen` and `cen` are logic-high; otherwise bus `io` is input

Logic expression operators are supported to define `enable` signals. See Table 43 for definitions of the logic expression operators.

Table 43 Logic expression operators

Operator	Definition
~	NOT
!	NOT
^	exclusive OR
&	AND
&&	AND
|	inclusive OR
||	inclusive OR

Important:

You must insert a space between a logical operator and the signal name. If there is no space, the signal name will not be recognized. For example:

```
#bi bi_driver (in ~ ctrl)
```

Note the space between the NOT operator (`~`) and the signal name (`ctrl`).

You can use wild cards to match node names, as described in Table 44.

Table 44 Wild card support

Wildcard	Description
*	Matches any group of characters
?	Matches any single character
[:]	Matches any range expression

You need to provide the full hierarchical name for the specified signal in the signal information file, based on the VCD file hierarchy. The period (.) character is used as the hierarchical delimiter in the signal information file. If you choose not to specify the full hierarchical name, see The #scope Command section. See the Signal Information File Sample section for a detailed sample.

HSIM uses input signals as stimulus for the simulation. Each input signal is converted to a PWL voltage source, in series with a resistor. The rise and fall times are used to smoothly transition from one state to another.

Each output signal is converted to an expected output checking statement. During simulation, the actual voltages are converted to a digital value and compared to the expected output checking statement at the specified times. A warning is generated, if the states differ. HSIM checks the output signals to the expected outputs at every time point specified in the VCD file.

See Example 13 for further details.

Example 13 Signal direction example

```
...
$var reg 4 " A [3:0] $end
$var reg 4 % B [3:0] $end
...
$dumpvars
b0000 "
b0000 %
$end
#10
b0010 "
#12
b0001 %
```

Bus `A` is defined as input and bus `B` is defined as output in the signal information file. At time `0`, bus `A` is initialized to `0000`; at time `10`, it changes state from `0000` to `0010`; at time `12`, there is no change in the state. Since bus `B` is specified as output, the simulation output of bus `B` is checked against the expected result specified in the VCD file. HSIM performs vector-checking at every time point (0, 10, 12), even if bus `B` does not change state at time `10`.

A delay usually occurs between the logic simulation and the transistor-level simulation. Therefore, HSIM issues warning messages if the state in the actual simulation does not match the VCD expected output. You can use the `#odelay` command to delay the expected outputs in the VCD file, so accurate vector-checking is implemented during simulation without generating warning messages. Refer to the `#odelay` command in the section The #idelay and #odelay Commands for further details.

Defining Mapping Information with the #alias Command

The `#alias` command maps the signal names in the VCD file into different names to match with the design netlist. You can use multiple aliases and regular expressions, but use only one specific alias for each signal name. If multiple patterns match the same name, the last pattern takes priority.

The following are situations in which you must use the `#alias` command:

- Different names are detected in the VCD file and the design netlist.
  For example, you want to map signal name *A1* in the VCD file to node name *B1* in the design netlist.
- Different hierarchy in the VCD file and the design netlist.
  For example, you want to map the signal from top module *top.A1* to the node name of the top hierarchy of the design netlist *A1*.

See the following syntax:

`#alias | #al` *`vcd_signal_name`* *`netlist_node_name`*

In Example 14, the `#alias` commands convert the following signal names in the VCD file to map with the signal names in the design netlist. The percent sign (`%`) represents the signal name, and the pound sign (`#`) represents the bus index. See Table 41.

Example 14 #alias command example

```
#alias tX64 x_64     ! alias #1
#alias t% %          ! alias #2
#alias tAP% Ap_%     ! alias #3
#alias tAPE% APE%    ! alias #4
```

See Table 45 for the mapping description.

Table 45 #alias command converting signal names

From (VCD file)	To (design netlist)	#alias matched
TAPEND	APEND	4
tL2CIB	L2CIB	2
tX64	x_64	1
TAPEB	APEB	4
TGBLB	GBLB	2
TAP0	Ap_0	3
TADD	ADD	2
TAPDD1	Ap_DD1	3

Defining Attributes for Signals

The signal information file supports commands that define various attributes for signals, such as *rise time* or *fall time*. Multiple specifications of each individual command is allowed in the signal information file. If more than one individual command is specified to a signal, the last command overrides the previous one. If the signal name is not specified, the value applies to all input or output signals.

See the following supported commands:

- The #edge_shift Command
- The #idelay and #odelay Commands
- The #outz Command
- The #scale Command

- The #trise, #tfall, and #slope Commands
- The #triz Command
- The #vih and #vil Commands
- The #voh and #vol Commands
- The #scope Command

The #edge_shift Command

The `#edge_shift` command specifies the timing shift on the transitions for specified signals. The default values for all `#edge_shift` arguments are 0.0. See the following syntax, and descriptions of the command arguments:

```
#edge_shift  time1  time2  time3   signal_name(s)
```

Table 46 #edge_shift command argument descriptions

Argument	Description
time1	Specifies the time shift to be applied to positive transitions (to 1or H)
time2	Specifies the time shift to be applied to negative transitions (to 0 or L)
time3	Specifies the time shift of all other transitions

The #idelay and #odelay Commands

The `#idelay` and `#odelay` commands specify the delay time of the specified input and output signals. The default values for `#idelay` and `#odelay` are 0.

See the following syntax:

```
#idelay  value  signal_name(s)
#odelay  value  signal_name(s)
```

The #outz Command

The `#outz` command specifies the drive resistance for specified input signals—for instance, the “H” and “L” state characters. State characters 1and 0 are considered to be strong logic and have infinite strength. The default value for `#outz` is 0.0.

See the following syntax:

```
#outz  value  signal_name(s)
```

The #scale Command

The `#scale` command specifies the time multiplier—a global setting that applies to all signals. This command does not specify individual signals. The default value for `#scale` is 1.0. The `#scale` command applies to all time variables, such as `#rise` and `#fall`.

See the following syntax:

```
#scale option
```

The #trise, #tfall, and #slope Commands

The `#trise` and `#tfall` commands specify the *rise time* and *fall time* for specified signals, respectively. The `#slope` command applies to both *rise time* and *fall time*. The default values for `#trise`, `#tfall`, and `#slope` are 0.0.

See the following syntax:

```
#trise value  signal_name(s)
#tfall value  signal_name(s)
#slope value  signal_name(s)
```

The first example in Example 15 assigns `10ps` as the fall time for all the signals.

If multiple patterns match the same signal name, the last pattern takes priority. The second example in Example 15 assigns `10ps` as the fall time for all signals, except `c` and bus `io`.

Example 15 #tfall command example

```
#tfall 10ps
#tfall 10ps *
#tfall 1ps c io*
```

The #triz Command

The `#triz` command specifies the output impedance for specified tri-state input signals. The default value for `#triz` is 1000Meg.

See the following syntax:

```
#triz  value  signal_name(s)
```

The #vih and #vil Commands

The `#vih` and `#vil` commands specify the logic-high and logic-low input voltages of specified signals. By default, HSIM detects the rail-to-rail voltage supply of the design and sets `#vih` to the maximum rail voltage, and `#vil` to the minimum rail voltage.

See the following syntax:

```
#vih  value  signal_name(s)
#vil  value  signal_name(s)
```

Example 16 assigns `1.1`V as logic-high, and `0.2`V as logic-low for all input signals.

Example 16 #vih and #vil command examples

```
#vih 1.1
#vil 0.2
```

Example 17 overrides the logic-high of bus `add` as `3.3`V, and logic-low of reset as `0.0`V. You could override the global values of logic-high and logic-low for specific input signals.

Example 17 #vih and #vil command examples

```
#vih 3.3 add*
#vil 0.0 reset
```

The #voh and #vol Commands

The `#voh` and `#vol` commands specify the output voltage logic-high and logic-low of specified output signals. The default value of `#voh` is set to `vih`; `#vol` is set to `vil`.

See the following syntax:

```
#voh  value  signal_name(s)
#vol  value  signal_name(s)
```

Example 18 assigns `3.3`V and `0.0`V as the global logic-high and logic-low, respectively, for all output signals except bus `out`. The logic-high of bus `out` is overridden as `1.1`V.

Example 18 #voh and #vol command examples

```
#voh 3.3
#vol 0.0
#voh 1.1 out*
```

The #scope Command

The `#scope` command specifies the name of the scope from which signals are used for matching. By default, if `#scope` is not specified, HSIM processes the signals starting from the top level in the VCD file. Note that the signals must be selected using `#in`, `#bi` or `#out` to be used as stimuli. You must specify the corresponding scope, so that the correct signals are used for simulation. If `#scope` is not specified, HSIM searches the signals starting from the top-level

of the VCD file. This eliminates the use of the `#alias` command to map the signals in the specified scope to the top-level hierarchy in the design netlist.

Only one `#scope` command is allowed in a signal information file. Create another signal information file if multiple scopes are needed.

Use the following syntax:

`#scope` *`scope_name`*

The scope definition in the VCD file is shown in Example 19.

Example 19 Scope definition in a VCD file

```
$var wire 1 A data_in  $end
$var wire 32 B data_out [31:0] $end

$scope module st $end
$var wire 1 ` data_in  $end
$var wire 32 ^ data_out [31:0] $end
$upscope $end

$scope module tpu $end
$var wire 1 _ data_in  $end
$var wire 32 Y data_out [31:0] $end
$upscope $end
```

Example 20 uses `st` as the top-level hierarchy in the VCD file, and searches the signals only in scope `st`.

Example 20 #scope st command example

```
#scope st
```

VCD File Sample

See Example 21 for a sample VCD file:

Example 21 VCD file sample

```
$date
     Sun Oct 10 18:20:30
$end
$version
     Synopsys VCS Version 7.0
$end
$timescale
     1ns
$end

$scope module adder $end
$var reg       1 !  CIN $end
$var reg       4 "  A [3:0] $end
$var reg       4 #  B [3:0] $end
$var wire      1 $  COUT $end
$var wire      4 %  S [3:0] $end
$upscope $end

$enddefinitions $end
$dumpvars
0!
b0000 "
b0000 #
0$
b0000 %
$end
#10
b0000 #
b0000 "
0!
#12
b0000 %
0$
#20
b0001 #
#22
b0001 %
#30
b0010 #
#32
b0010 %
#40
b0011 #
#42
b0011 %
#50
b0100 #
```

```
#52
b0100 %
#60
b0101 #
#62
b0101 %
#70
b0110 #
#72
b0110 %
#80
b0111 #
#82
b0111 %
#90
b1000 #
#92
b1000 %
#100
b1001 #
```

Signal Information File Sample

See Example 22 for a sample signal information file:

Example 22 Signal information file sample

```
; Define bus syntax
#format     %[#]

; Define signal directions
#in         adder.CIN
#in         adder.A[[3:0]]
#in         adder.B[[3:0]]
#out        adder.COUT
#out        adder.S[[3:0]]

; Define mapping information
#alias     adder.CIN     CIN
#alias     adder.A[3]     A[3]
#alias     adder.A[2]     A[2]
#alias     adder.A[1]     A[1]
#alias     adder.A[0]     A[0]
#alias     adder.B[3]     B[3]
#alias     adder.B[2]     B[2]
#alias     adder.B[1]     B[1]
#alias     adder.B[0]     B[0]
#alias     adder.COUT     COUT
#alias     adder.S[3]     S[3]
#alias     adder.S[2]     S[2]
#alias     adder.S[1]     S[1]
#alias     adder.S[0]     S[0]

; Define analog waveform characteristics
#vih        3.3
#vil        0.0
#voh        3.0
#vol        0.3
#slope     10ps
```

See Example 23 for a sample signal information file generated with the `#scope` command:

Example 23 Signal information file sample generated with the #scope command

```
; Define bus syntax
#format        %[#]

; Define scope
#scope         adder


; Define signal directions
#input         CIN
#input         A*
#input         B*
#output        COUT
#output        S*


; Define analog waveform characteristics
#vih           3.3
#vil           0.0
#voh           3.0
#vol           0.3
#slope         10ps
```

Vector Data

Vector data are defined in tabular format. The first vertical column is time. The values in the time column must be in increasing order. If no unit is specified in the value, the default unit of nanosecond will be used. If the period is specified in the vector definition section, it is not necessary to specify the time at each row of vector data section.

The time at the i-th row is implicitly defined to be i-1 times the period. Each subsequent vertical column defines the logic pattern associated with the corresponding signal node(s) defined in signal statement. The pound sign (#) at the first vertical column indicates that relative time is used.

Each input data bit can be one of the following valid states.

- 0: Drive to ZERO
- 1: Drive to ONE
- z: High Impedance
- x: Drive to ZERO

The expected output bit can be one of the following states:

- 0: Expect ZERO
- 1: Expect ONE
- x: Don't Care
- u: Don't Care
- h: Expect HiZ-H state
- l: Expect HiZ-L state
- z : Expect either HiZ-H or HiZ-L state

The HSIMHZ on page 82 command controls HSIM treatment of output bits. When HSIMHZ=0 (default), the following bits are treated identically: h, l, x, and u.

When HSIMHZ=1, each output bit has a unique definition. Do not use h with 1, or l with 0. Do not use x or u with z.

When HSIMHZ= -1, HSIM treats h and 1 identically, and l and 0 identically.

Example

```
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>

radix 1111 4444
io iiii iiii
0   1001 fffe
202 1001 0000
246 0001 0000
250 0000 00a0
300 1000 0000
346 0000 z300
396 0001 z000
400 1001 z000
402 1001 z002
446 0001 z002
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>
period 50
radix 1111 4444
io iiii iiii
1001 fffe
1001 0000
0001 0000
0000 00a0
1000 0000
0000 z300
0001 z000
1001 z100
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>

radix 1111 4444
io iiii iiii
0   1001 fffe
#10 1001 0000
#4 0001 0000
#20 0000 00a0
50 1000 0000
#5 0000 z300
```

The patterns are applied at the times indicated below:

- The first pattern is applied during 0 – 10 ns
- The second pattern during 10 ns – 14 ns

- Third pattern during 14 ns – 34 ns
- Fourth pattern during 34 ns – 50 ns
- Fifth pattern during 50 ns – 55 ns
- Sixth pattern starts at 55 ns

Built-in Functions

The HSIM built-in functions and their descriptions are described in the table below.

Table 47 Built-in Functions

Function	Description
sin(x)	Returns sine of x.
cos(x)	Returns cosine of x.
tan(x)	Returns tangent of x.
asin(x)	Returns arc sine of x.
acos(x)	Returns arc cosine of x.
atan(x)	Returns arc tangent of x.
atan2(x,y)	Returns the arc tangent of x/y using the signs of x and y to determine the quadrant of the return value.
sinh(x)	Returns hyperbolic sine of x.
cosh(x)	Returns hyperbolic cosine of x.
tanh(x)	Returns hyperbolic tangent of x.
abs(x)	Returns the absolute value of x.
sqrt(x)	Returns square root of x.
pow(x,y)	Returns the value of x raised to the integer part of y: x(integer part of y).
pwr(x,y)	Returns pow(abs(x),y) with the sign of x.

Table 47 Built-in Functions (Continued)

Function	Description
log(x)	Returns nature log of abs(x) with the sign of x.
ln(x)	Same as log(x).
log10(x)	Returns base 10 log of abs(x) with the sign of x.
exp(x)	Returns the e to power of x.
db(x)	Returns 20.0 * log10(abs(x)) with the sign of x.
floor(x)	Returns the largest integral value which is not grater than x.
ceil(x)	Returns the smallest integral value which is not smaller than x.
int(x)	Returns ceil(x) if x is less than 0, otherwise returns floor(x).
trunc(x)	Same as int(x).
sgn(x)	Returns 0 if x is 0, otherwise returns the sign of x.
sign(x)	Same as sgn(x).
sign(x,y)	Returns sign(y) * abs(x).
min(x,y)	Returns the minimum of x and y.
dmin(x,y)	Same as min(x,y).
max(x,y)	Returns the maximum of x and y.
dmax(x,y)	Same as max(x,y).

Simulation and Control Statements

This sections lists the simulation and control statements.

.alter

Description

The .alter statement repeats a simulation using alternative parameter values and modified netlist. In each simulation run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .a#, where # is the run number.

Syntax

```
.alter
```

Example

```
* first simulation run
.temp 25
......
.alter
.temp 100
.end
```

The circuit temperatures for the two runs are: Run 1; : 25° C and Run 2; 100° C.

.data

Description

The .data statement allows modifying parameter values in transient simulation. This command is for cell or block characterization and optimization. Refer to Chapter 15, Timing and Power Analysis, Bisection Optimization on page 444. The group of parameter values is included in the input file.

Syntax

```
.data dlink + param1 <param2 param3 ... paramN> + val1a
   <val2a val3a ... valNa> + val1b <val2b val3b ... valNb>
   + ...
.enddata
```

Parameters

dlink

dlink is also used in the .tran command

param

Parameter names

val

Parameter values.

Examples

```
.tran 0.1n   100n sweep data=dlink1
.data dlink1
+ L     W   CLOAD
+ 0.18u    0.36u     10f
+ 0.13u    0.26u     5f
.enddata
```

HSIM accepts the following parameter values and performs the first transient simulation:

```
L=0.18u, W=0.36u, CLOAD=10f
```

Afterwards, the simulation is repeated for the following:

```
L=0.13u, W=0.26u, CLOAD=5f
```

.del param

Description

The .del param statement removes selective parameter setting which is especially useful in the .alter section.

Syntax

```
.del param <param1> <param2> ... <subckt=<subckt_name>>
```

For instance, a vector file can be removed from the simulation in the .alter on page 288.

Example

```
. param hsimvectorfile=vec1
......
......
.alter
.del param hsimvectorfile
.param hsimvectorfile=vec2
      ......
```

In this case, the vector file vec1 is removed in the .alter section and another file, vec2, is added to the simulation. If the .del param statement is not used, then both vec1 and vec2 files are included in the simulation in the .alter on page 288.

.end

Description

The .end statement defines the end of an HSIM run.

Syntax

```
.end <comments>
```

If .end is not specified in an input netlist, the default is end of file. An input netlist can contain multiple HSIM runs by having an .end at the end of each run. In each HSIM run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .e#, where # is the run number.

Example

See the example for .alter on page 288.

.endl

The .endl statement ends the library macro definition.

Syntax

```
.endl <lib_entry_name>
```

Example

See the example for .lib on page 293.

.ends

The .ends statement specifies the end of a subckt definition.

Syntax

```
.ends <subckt_name>
```

Example

See the example for .subckt on page 299.

.force

Syntax

```
.force node_name voltage_value <subckt=subckt_name>
   <time=force_time>
```

Parameters

node_name

node_name can be a specific node name or a pattern.

subckt_name

subckt_name is the subcircuit name. When the node_name subcircuit parameter is used, the node is inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.

force_time

force_time is the starting time that forces the specified node to stay at a specified constant voltage_value. Time unit is seconds.

Description

.force forces node_name to stay at the same voltage_value as force_time until one of the following occurs:

- Simulation ends
- When the constant node voltage status is released by either of the following:
  - .release
  - rv.

.force does not work on voltage source or vector file input nodes. Refer to the following sections for additional information:

- Forces constant voltage to nodes. on page 406
- Releases constant voltage from nodes. on page 422
- .force on page 290

Example

```
.force pump 6.5 time=100u
```

.global

The .global statement defines global nodes.

Syntax

```
.global node1 <node2 ...>
```

A global node can be directly referenced from any level of hierarchy. Any node name appearing at the top-level or any low level hierarchy is connected to the same node. Declaring a global node allows a direct reference to an internal

subcircuit node without defining the node in the subcircuit port list. HSIM recognizes any of the following as the ground node:

- 0 (zero)
- gnd
- gnd!
- ground

.ic

The .ic statement defines the initial condition.

Syntax

```
.ic v(node1)=val2 <v(node2)=val3 ...> <subckt=sub_name>
   <level=val4>
```

The specified node can be the node name of a single node or a pattern containing asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of level is specified to control the scope of wildcard match.

Parameters

The .IC statement parameters are described in the table below.

Table 48 .IC Statement Parameters

Parameter	Default	Description
subckt=sub_name		Initial condition is set to the specified node name(s) within all instances of the specified subcircuit name. This subckt setting is equivalent to placing the .ic statement within the subcircuit definition.

Table 48 .IC Statement Parameters

Parameter	Default	Description
level=val4	-1	This setting is effective only when the asterisk (*) wildcard character is specified in the output variable. The level value val4 specifies the number of hierarchical depth levels when matching the wildcard node/element name. ▪ When val4 is set to 1, the wildcard match applies to the same depth level where the .ic statement is located. ▪ When val4 is set to 2, it applies to same level and one level below the current level where .ic is located. ▪ When val4 is set to 0, no wildcard name is matched so that any output variable containing wildcard character is ignored. ▪ When val4 is set to –1, the wildcard match applies to all the depth levels below and including the current level of .ic statement.

Example

```
.ic v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2
```

.include

The .include statement includes another data file.

Syntax

```
.include   file_name
```

or

```
.inc file_name
```

Example

```
.inc netlist.net
```

.lib

The .lib statement includes the model library.

Syntax

```
.lib file_name library_name
```

This is a library call statement which indicates the specified library entry in the library file should be read in.

You can also use:

```
.lib library_name
```

This is a library definition statement that begins the library entry in the library file. The .endl indicates the end of the library entry. Refer to .endl on page 290.

Parameters

The .lib statement parameters are described in the table below.

Table 49 .LIB Statement Parameters

Parameter	Description
file_name	The filename of the library.
library_name	The library entry name.

Examples

```
.lib models.lib TT
```

The TT portion of models.lib is read in.

```
.lib TT
.model ck1 nmos level=49
... ... ...
.endl TT
```

A library is defined.

.malias

The .malias statement provides an alias for a model name.

Syntax

```
.malias model_name alias_name1 <alias_name2 ...>
```

The word alias_name1 is aliased to model_name.

Example

```
.malias tn013   ma   mb
```

Both model names ma and mb are aliased to tn013.

.nodeset

This statement sets the starting voltage at DC initialization.

Syntax

```
.nodeset v(node1)=val2 <v(node2)=val3 ...>
   <subckt=sub_name> <level=val4>
```

Description

The specified node can be the node name of a single node or a pattern containing a asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of level controls the scope of wildcard match. Refer to .ic on page 292 for details.

The optional setting of subckt is for the nodes within all instances of the specified subcircuit name; this is equivalent to placing the .nodeset statement inside the subcircuit definition.

Example

```
.nodeset v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2
```

.op

The .op statement dumps the operating condition at the specified time(s).

Syntax

```
.op <time1> <time2 ...>
```

Description

The DC operating point output is partially supported in HSIM. HSIM only creates the node voltages. The DC operating analysis should generate node voltages and element currents at the resulting DC operating point.

When .op statement is specified in the netlist, HSIM prints each node voltage at the specified time into a file. The format of the output file can be changed by setting HSIMOPCOMPRESS on page 104.

.option

Defines optional values.

Syntax

```
.option name1=val1 <name2=val2> . . . <nameN=valN>
```

Description

HSIM recognizes popular simulation options described in the table below..

Table 50 SPICE Simulation Options Recognized by HSIM

aspec	defnrs	spice
badchr	defpd	warnlimit
defad	defps	wl
defas	genk	scale
defl	klim	scalm
defw	nowarn	search
defnrd	parhier	tnom

All other SPICE options are ignored.

Note:

Chapter 10, Simulation Output describes output statements that are supported in HSIM.

Parameters

The .option parameters are described in the table below.

Table 51 .option Parameters

Parameter	Default	Description
aspec		Makes the simulation compatible with aspec setting, such as when the wl option (see below) is activated. scale and scalm are set to scale geometry dimension to microns.

Table 51 .option Parameters (Continued)

Parameter	Default	Description
badchr		Produces a warning if n unprintable characters are detected in the input file.
defad	0	The default value for drain-bulk junction diode area in a MOSFET.
defas	0	The default value for source-bulk junction diode area in a MOSFET.
defl	0	The default value for channel length in a MOSFET.
defnrd	0	The default value for the number of squares for MOSFET drain resistor.
defnrs	0	The default value for the number of squares for MOSFET source resistor.
defpd	0	The default value for drain bulk junction diode perimeter in a MOSFET.
defps	0	The default value for source bulk junction diode perimeter in a MOSFET.
defw	0	default value for channel width of a MOSFET.
genk	1	When assigned the value of 1 (one), genk is the control parameter for the automatic calculation of second-order mutual inductance for multiple coupled inductors.
klim	0.01	Minimum mutual inductance threshold for calculation of second-order mutual inductance.
nowarn		Suppress warning messages.
Parhier=global		A parameter name specified at a higher hierarchical level which prevails over the same parameter name specified at a lower level.
Parhier=local	global	A parameter name specified inside a subcircuit which prevails over the same parameter name specified at a higher hierarchical level.

Table 51 .option Parameters (Continued)

Parameter	Default	Description
scale	1.0	Element scale parameter. It scales the geometry of the transistor instance.
scalm	1.0	Model scaling factor. Relevant model parameters are scaled by this value.
search=dir_path		Sets the search path for model libraries and included files. The HSIM searches the specified directory for libraries used in the simulation.
spice		Makes the simulation compatible with SPICE setting, such as: tnom=27 defnrd=1 defnrs=1
tnom	25 or 27	Simulation reference temperature. The default value is 25 degrees C unless .option SPICE is specified; then the default is 27° C.
warnlimit=y		Limits the number of warnings to appear in the .log file. Value y is the total number of warnings allowed for each warning type.
wl		MOSFET element geometry order is changed from (defaulted) length-width to width-length.

Example

```
.option scale=1e-6
```

.param

The .param statement defines parameters.

Syntax

```
.param name1=val1 <name2=val2 ...>
```

Example

```
.param a=2 b=4
.param c='b+3*a'
```

.release

Syntax

```
.release node_name <subckt=subckt_name> <time=release_time>
```

Parameters

node_name

node_name can be a specific node name or a pattern.

subckt_name

subckt_name is the subcircuit name. When the node_name subcircuit parameter is used, it is the node inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.

release_time

release_time is the starting time that releases the specified node. Time unit is seconds.

Description

.release releases the node voltage from the value fixed by the .force command or by the interactive rv command. The node voltage are then determined by the regular simulation result. Refer to the following sections for additional information:

- Forces constant voltage to nodes. on page 406
- Releases constant voltage from nodes. on page 422
- .force on page 290

.subckt

The .subckt statement defines a subcircuit.

Syntax

```
.subckt sub_name   term1 <term2 ...> <parameter1=val1>
   <parameter2=val2 ... >
```

Example

```
.subckt inverter 1 2
M1 2 1   0    0    nmos1 w=0.36u l=0.18u
M2 2 1   vdd vdd pmos1 w=0.36u l=0.18u
.ends
```

Subcircuit Instance

The subcircuit instance is defined below:

Syntax

```
Xaa term1 <term2 ...> sub_name <parameter1=val2>
   <parameter2=val3>
```

Example

```
Xinv1 1    2   inverter
```

.temp

The .temp statement defines the temperature.

Syntax

```
.temp temp1 <temp2 ...>
```

Description

The temperature value is in degrees Celsius. If multiple temperatures are specified, HSIM will perform one simulation for each temperature with the output files (i.e. hsim.ic, hsim.fsdb, etc.) having a .t# suffix. The default temperature is 25 degrees Celsius.

Example

```
.temp 85
```

.tran

The .tran statement defines transient analysis.

Syntax

Three versions of sweep syntax are supported.

```
.tran steptime stoptime <uic> <sweep data=data_name>
.tran steptime stoptime <uic> sweep var pstart pstop incr
.tran steptime stoptime <uic> sweep var poi np p1 p2 ...
```

Description

Except for uic and sweep data, all optional parameters specified after stoptime are ignored.

For more information about sweep data, refer to the example in .data on page 288.

Parameters

The .tran statement parameters are described in the table below.

Table 52 .TRAN Parameters

Parameter	Description
steptime	A placeholder for compatibility with the SPICE syntax–this parameter does not effect HSIM simulation.
uic	Disables DC initialization and uses the IC conditions as DC solution. If a node does not have an I condition, its DC solution will be 0.
var	Parameter name or keyword temp (for temperature sweep)
pstart	Starting value.
pstop	Final value.
incr	Increment value.
np	Number of points.
p1, p2	Values for the parameter sweep.
poi	Indicates that type of sweep is a list of points.
sweep	Indicates sweep operation according to the parameter values specified in data_name. Refer to the example in .data on page 288.

Example

```
.tran   1N   100N
```

8

Simulation Commands

This chapter describes how to set the HSIM simulation commands. It also describes how these commands affect performance and accuracy.

Specifying HSIM Commands

There are two methods for specifying HSIM control commands:

- In the top-level netlist file.
- In an external file included in the main netlist.

When commands are specified inside the netlist file, some HSIM commands can be flexibly placed within either of the following:

- Subcircuit
- Top-level netlist

Specifying HSIM Commands in the Top-Level Netlist

Specifying HSIM commands within the netlist file has the advantage that their influence can be localized to the module in which the command is defined. Also, the netlist can still be simulated by other SPICE simulators. The HSIM commands are ignored.

An HSIM command specified at the top-level netlist is a global command which affects the entire circuit. When specified with a different value within a subcircuit, the HSIM command is a local command and the value only applies at the local level within the subcircuit. Commands defined in a subcircuit alone only affect values within the range of the subcircuit.

Start each HSIM control command with the keyword HSIM, shown in the syntax sample below, to identify the commands and avoid interference with other commands supported in SPICE.

```
hsimparameter_name=value1
```

The modified netlist containing the HSIM command setting with a real value operates correctly when performing a standard SPICE simulation using the same netlist. However, using the HSIM command setting with string value or PWC function can not be simulated in a standard SPICE simulation. These commands can be entered in `hsim.ini` to avoid this problem. There are the following different syntax types used to set the HSIM command:

1. A single value HSIM command is input as follows:

   ```
   .param hsimparameter_name=value1
   <hsimparameter_name2=value2>
   ```

2. A PWC function controlled by simulation time command is input as follows:

   ```
   .param hsimparameter_name3=pwc (t1, val1, t2, val2, ...,   tN,
   valN)
   ```

Syntax Definitions

t1, t2, ... tN

Simulation time.

val1, ... valN

Setting different parameter values corresponding to t1, t2, ... tN represent different simulation times.

A PWC function controlled by a scheduled event is input as follows:

```
.param hsimparameter_name4=pwc (label1, val1, label2, val2,
   ..., labelN, valN)
.schev labelN var=variable val=threshold_val edge=<r|f|c>
   <from=time1 to=time2>
```

labelN

Label name of scheduled event command.

variable

Node name (v(node_name))or branch current (i(elem_name)).

r

Rise

f

Fall

c

Cross

Examples

```
.param hsimalloweddv=pwc(schev1, 0.3, schev2, 0.1)
.schev schev1 var=v(node1) val=1.5 edge=r
.schev schev2 var=v(node2) val=1.5 edge=f
```

The setting can be placed anywhere at the top-level netlist outside each subcircuit definition. Some commands can also be specified as local commands, which are either defined within the subcircuit definition or attached to a subcircuit instance.

To attach the command setting to the end of port list in the subcircuit definition, add the command within the subcircuit definition that affects all instances of the subcircuit, except for those instances with the same setting but a different command value. Use the following syntax.

```
.subckt subcircuit_name port_1 <port_2 ...>
   hsimparameter_name1=value1 <hsimparameter_name2=value2
   ... >
```

To set the command in the subcircuit instance to affect only the instance, append the setting to the end of the subcircuit instance call. Use the following syntax.

```
Xname port_1 <port_2 ...> subcircuit_name
   hsimparameter_name1=value1 <hsimparameter_name2=value2
   ...>
```

Specify exactly where the global commands are to be placed using one of the following syntax statements:

```
.param hsimparameter_name1=val1 <hsimparameter_name2=val2>
.param hsimparameter_name3=pwc(t1, val1, t2, val2, ..., tN,
   valN)
```

The following syntax applies to the subcircuit definition associated with the subcircuit instance.

```
.hsimparam <subckt=sub_name>
<inst=subcircuit_instance_name>
hsimparameter_name1=val1 <hsimparameter_name2=val2 ...>
```

- If the subcircuit definition is specified, the particular HSIM command is effective for each instance of the specified subcircuit definition `sub_name`.
- If the subcircuit instance is specified, the HSIM command is effective for the specified subcircuit instance `subcircuit_instance_name` at the top-level when subckt is not specified or at the specified `subckt` level.

Equivalent to adding the `.param HSIMANALOG=2` statement into the subcircuit definition of `samp`:

```
.hsimparam subckt=samp HSIMANALOG=2
```

Equivalent to appending HSIMANALOG=2 to the end of instance X1 within the subcircuit definition `samp`:

```
.hsimparam subckt=samp inst=X1 HSIMANALOG=2
```

Equivalent to appending HSIMANALOG=2 to the end of top-level instance X1:

```
.hsimparam inst=X1 HSIMANALOG=2
```

In addition to setting the command at the global and subcircuit levels, some transistor-specific commands can be specified at the individual transistor level. Unlike other commands, transistor-specific commands do not start with the HSIM keyword HSIM.

Transistor-specific commands are distinguished from the commands supported in SPICE by starting with the dollar sign character ($). These commands must start with the dollar sign character ($) so the commands are treated in the same way as a comment in a standard SPICE parser. Use the following syntax:

```
Mname drain gate source bulk model_name w=width l=length
   <ad as pd ps nrd nrs ...> $parameter_name1=value1
```

Specifying Commands in an External File

HSIM commands can be specified in a separate file outside the netlist. When specified at the top-level netlist, they can be directly relocated to another file. This is done by including the file in the main netlist using the .include statement. Refer to Chapter 7, Input Netlist; .include on page 293.

The advantage of specifying control commands in an external file is that the fewest modifications are made to the main netlist. Specifying the commands in a separate file, then including the file in HSIM using the .include option, does not change the original circuit netlist.

Note:

HSIM supports the TISPICE .SECTION statement conventions for including netlist commands.

Some HSIM commands can possess values as a piecewise constant (PWC) function of time in order to have different values for different time windows. Refer to the examples in Specifying HSIM Commands in the Top-Level Netlist on page 303.

This feature provides higher flexibility in controlling the simulation speed and precision trade off. More conservative precision settings can be used in time windows when simulation precision is most critical, while aggressive speed settings can be used in modules or time windows when precision is not so critical. For example, in phase-locked loop (PLL) circuit simulation, higher precision is necessary when the PLL circuit is approaching the lock-in stage.

Note:

This new pwc function is not supported by other SPICE simulators. It must be commented out if the netlist is simulated by other SPICE simulators.

Aliasing Commands to User-Defined Names

The .palias command is used to alias HSIM commands to a user-defined name:

```
.palias hsim_parameter_name alias1 alias2 ...
```

.palias can alias multiple alias names to the same HSIM command. The alias name must begin with hsim. The .palias command must be defined before using the alias name in the netlist.

Example

```
.palias HSIMMQS hsim_multi_rate
```

Control Commands for Ungrounded Capacitors

Capacitors have a complicated impact on simulation efficiency. If a capacitor has one of its terminals connected to a constant voltage source, such as V_{DD} or GND, this grounded capacitor does not require extra computations because the capacitance value only affects the diagonal entry value in the circuit admittance matrix.

However, if both terminals of a capacitor are connected to signal nodes, such an ungrounded capacitor occupies not only the diagonal entries associated with its two terminals but also the off-diagonal entries associated with the coupling between the two terminals. This causes a major impact on the computational complexity in solving the matrix equations. The existence of off-diagonal entries affect the matrix solution and might also create additional non-zero couplings in the off-diagonal entries, called fill-ins in the matrix solution.

Coupling Effect

The coupling effect between both terminals of an ungrounded capacitor, is proportional to the ungrounded capacitance value as well as the ratio of the capacitance to the total node capacitance of each terminal. The total node capacitance is defined to be the sum of all capacitances, both grounded and ungrounded capacitances, connected to the node. The default setting of modeling an ungrounded capacitor depends on the absolute capacitance value and its ratio to the total node capacitance. The ratio is defined as the maximum of the ratios to either terminal node capacitance.

Control Commands for Grounded Capacitors

Each linear capacitor from a signal node to a constant voltage source node is considered a grounded capacitor in order to:

- Improve simulation speed
- Reduce memory usage

HSIM does not keep these types of capacitor as a regular element. They become the property of the node and each node only keeps track of the sum of all such capacitors to the constant voltage source nodes. This does not affect simulation precision because constant voltage source nodes are AC grounded during transient simulation.

The limitation to this methodology is that when voltage source node currents are reported, HSIM can no longer keep track of where the lumped capacitors are connected. Therefore, the simulator cannot account for the corresponding capacitive current.

Missing these capacitive components mostly affects the peak current values of voltage source nodes and has less effect on the average current values. If the

circuit does not have many grounded capacitors, then the effect on current values is minimal.

Control Commands for Floating Capacitors

HSIM can process floating capacitors with three different models, as shown in Figure 10.

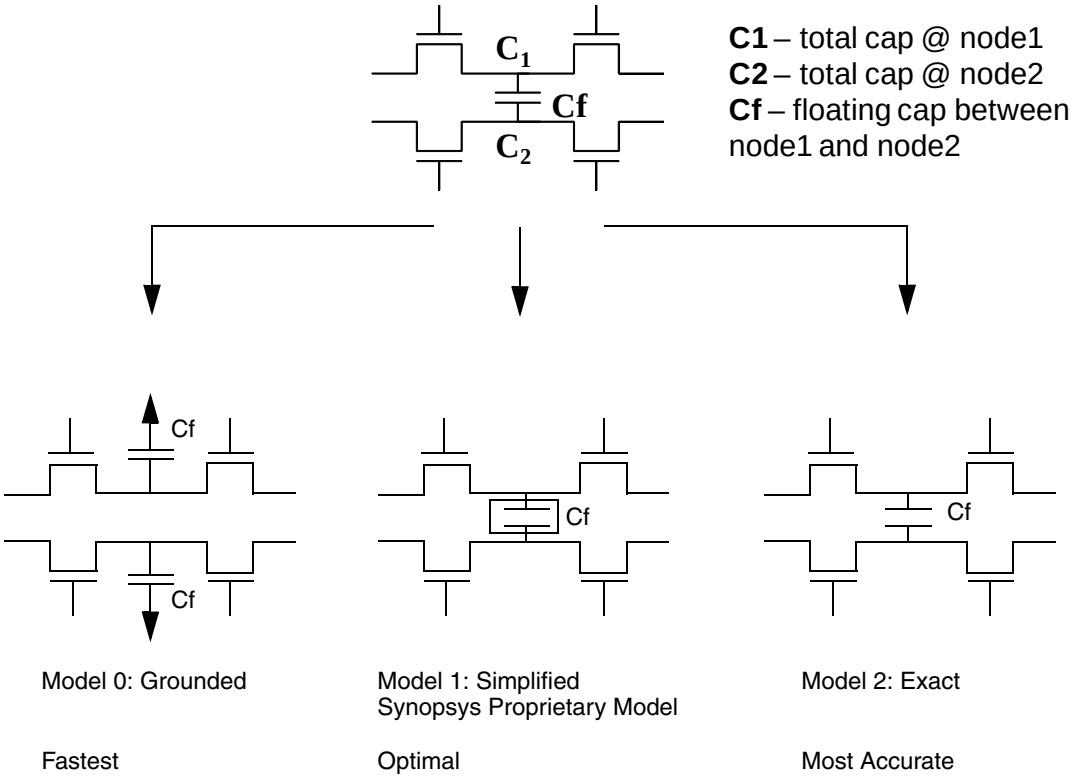


Figure 10 Floating Capacitor Models

You use the HSIMFCM command to specify the floating capacitor model:

HSIMFCM Argument	Description
0	The grounded model ignores the coupling effect between capacitor terminals. In this case, the capacitance coupling between terminals is modeled as two grounded capacitors: one connected between node A and the ground, and the other connected between node B and the ground. Moreover, each capacitance value is the same as the original ungrounded capacitance value. This model provides the best speed performance but may lose some precision if the coupling effect is important.
1	The simplfied mode handles the coupling effect approximately. This model provides speed and precision performance in between those of grounded and exact models.
2	The exact model handles the coupling effect precisely by loading the capacitance value into the diagonal and off-diagonal entries of the admittance matrix. Tis model is precise, but if the circuit contains many ungrounded capacitors it might slow down the simulation considerably.
3	This is the default value for HSIMFCM. You can further control the floating capacitor model selection with the following commands: HSIMANALOG, HSIMFCAP, HSIMFCAPR, HSIMGCAP, and HSIMGCAPR. See the following figure. The other command defaults are: HSIMANALOG=1 HSIMFCAP=1E-13 HSIMFCAPR=0.5 HSIMGCAP=1E-15 HSIMGCAPR=0.02

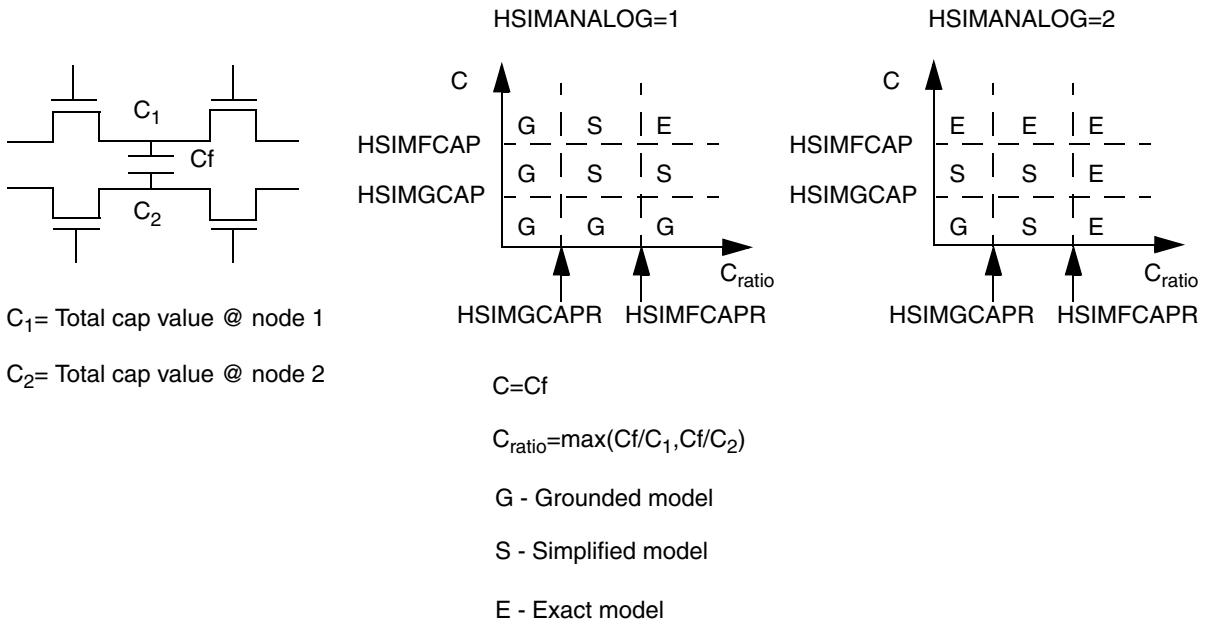


Figure 11 Controlling the Floating Capacitor Model Selection with HSIMANALOG, HSIMFCAP, HSIMFCAPR, HSIMGCAP, and HSIMGCAPR

In Figure 11, if the value on the C axis is a large portion of the overall node capacitance (the C_{ratio} value), a more accurate model is used to preserve the coupling effect. If the C and C_{ratio} values are smaller, the coupling effect has less impact, so a faster, less accurate model can be used. The HSIMFCAP, HSIMFCAR, HSIMGCAP, and HSIMGCAPR commands control the these values.

Note:

When HSIMANALOG=2, the matrix values are more conservative.

Control Commands for Isomorphic Matching

If identical subcircuit instances have the same behavior, they can share:

- Storage
- Simulation results

These instances of the same subcircuit definition may behave differently if the driving port voltages or fanout port capacitances are different. Control commands are provided to define the tolerances in determining whether subcircuit instances can share the same computation result. The control commands are shown in Table 8.

Table 8 Port Control Command Descriptions

Command	Description
Port Voltage Tolerance	Port voltage tolerance is defined by HSIMPORTV on page 110 such that two corresponding port voltages are considered matched if their difference is less than the value specified by HSIMPORTV on page 110.
Port Capacitance	Port capacitance is defined as the total lumped capacitance contributed by elements. This can be either inside or outside the subcircuit connected to the port. The port capacitance tolerance is defined by HSIMPORTCR on page 109.
Port Current Tolerance	Port current tolerance is defined by HSIMPORTI on page 109 such that two corresponding port currents are considered matched if their difference is less than the value specified by HSIMPORTI on page 109. By default, only port voltage tolerance is considered. For small resistors and zero volt floating voltage sources, in order to obtain accurate currents through those elements, HSIMPORTI on page 109 may need to be specified. Units are in Amperes.

Two port capacitances are considered matched if the ratio of the capacitance difference to the average of two corresponding port capacitances is less than the value specified by HSIMPORTCR on page 109.

DC Initialization

DC initialization is activated after the netlist processing and hierarchical database building phase is completed and before the beginning of a transient simulation. However, under certain special situations, there is no need to perform DC initialization to achieve the desired results. The control command HSIMDCINIT on page 63 is used to activate the DC initialization.

Note:

When HSIMDCINIT on page 63 is set to zero, DC initialization is suppressed and the transient simulation starts immediately following the preprocessing. Under this condition, the DC initialization proceeds as if running the transient simulation using zero initial state for each node voltage, unless the node has an initial condition setting through the use of the .ic command.

When HSIMDCINIT on page 63 is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the number of iteration steps exceeds the limit specified by HSIMDCITER on page 64. A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by HSIMDCV on page 66, and each net current is less than the current specified by HSIMDCI on page 63. These commands can be configured to create levels of precision for DC operation needed in their specific circuits. DC initialization commands are described in Table 9.

Table 9 DC Initialization Commands

Command	Type	Default Value	Location(s)
HSIMDCINIT on page 63	Boolean	1	Top-level
HSIMDCITER on page 64	Int	250	Top-level
HSIMDCV on page 66	Double	0.001	Top-levelsubckt Def.subckt Inst
HSIMDCI on page 63	Double	5E-7	Top-levelsubckt Def.subckt Inst
HSIMDCSTEP on page 66	Int	1000	Top-level
HSIMMULTIDC on page 99	Boolean	1	Top-level
HSIMENHANCEDC on page 75	Boolean	0	Top-level
HSIMKEEPNODESET on page 89	Boolean	0	Top-level
HSIMNODCNODES on page 100	Boolean	0	Top level
HSIMSTOPIFNODC on page 146	Boolean	0	Top level

Table 9 DC Initialization Commands

Command	Type	Default Value	Location(s)
HSIMWARNIFNODC on page 172	Boolean	0	Top level

Implicit Backward Euler Algorithm and Trapezoidal Integration Algorithm

Interrogation algorithms used by HSIM to solve equations are described in Table 10.

Table 10 Integration Algorithms

Algorithm	Description
Implicit Backward Euler Algorithm	HSIM applies the Implicit Backward Euler Algorithm by default. It is a first-order numerical integration algorithm that solves differential equations. It has been well researched in the literature that the Implicit Backward Euler Algorithm suppresses the oscillation in the RLC circuit.
Trapezoidal Algorithm	To simulate oscillators containing inductors, set HSIMTRAPEZOIDAL on page 153 to 1. This command invokes the second-order numerical integration algorithm: the Trapezoidal Algorithm. Though more suitable for simulating the RLC oscillator circuit, the Trapezoidal Algorithm may cause artificial oscillation in high-impedance nodes.

Simulation Control Commands

Circuit simulations always contain a trade-off between:

- Speed
- Precision

Accurate simulation requires sophisticated models and conservative algorithms that demand more computations and result in longer simulation time. However, simulation speed can be pushed to go faster through the use of simplified models and more aggressive algorithms. This produces less accurate results. Selecting the optimal trade-off between speed and accuracy performance is a

complicated task because the same model and simulation algorithm may have dramatically different speed and accuracy performance in various circuit types. For example, the simplified MOSFET table model may provide accurate performance for digital circuits, but provide insufficient accuracy for high-sensitivity analog circuits.

Simulation Speed Control Summary

HSIM precision and speed performance are affected by many control commands which control the selection of various models and computational algorithms. HSIMSPEED on page 125 is a macro command containing a few HSIM commands controlling HSIM's simulation speed and precision. It is also possible to directly set these individual commands and overwrite the default command value defined by HSIMSPEED.

HSIMSPEED on page 125 control commands are summarized in the Table 11 on page 315 and Table 12 on page 316.

Table 11 Simulation Control Commands

Command	Type	Default Value	Location(s)
HSIMSNCS on page 124	Boolean	1	Top-level
HSIMMQS on page 99	Int	1	Top-level
HSIMSTEADYCURRENT on page 144	Double	1E-8	Top-levelsubckt Def.subckt Inst
HSIMALLOWEDDV on page 47	Double	0.3	Top-levelsubckt Def.subckt Inst
HSIMTAUMAX on page 149	Double	2E-9	Top-level
HSIMSPICE on page 143	Int	0	Top-levelsubckt Def.subckt Inst
HSIMGCSC on page 79	Boolean	1	Top-levelsubckt Def.subckt Inst
HSIMFLAT on page 77	Boolean	1	Top-levelsubckt Def.subckt Inst

Table 12 on page 316 shows the relationship between HSIMSPEED and other control commands.

Table 12 Speed Command Relationships

HSIMSPEED on page 125	HSIMSNCS on page 124	HSIMFLAT on page 77	HSIMGCSC on page 79	HSIMALLOWEDDV on page 47	HSIMSTEADY CURRENT on page 144
0	0	1	1	0.1	1E-8
1	1	1	1	0.3	1E-8
2	1	1	1	0.3	1E-8
3	1	1	1	0.3	1E-8
4	1	1	1	0.3	1E-7
5	1	1	0	0.5	1E-7
7	1	1	0	0.5	1E-7
8	1	1	0	0.5	1E-7

Piecewise Constant Setting

The control commands for adjusting HSIM simulation precision and speed, have a continuous effect from the beginning to the end of simulation. In practice, there may be different precision requirements in different simulation cycles such that the higher precision performance is crucial only in certain time windows. For example: high precision is necessary in PLL circuit simulation when it approaches the lock-in state.

To optimize the trade-off in simulation speed and precision, it is preferred to have control commands adjustable in different time windows. A more conservative setting is applied to time windows when the precision is crucial and a more relaxed setting is applied otherwise in order to speed up simulation.

In HSIM, the following commands can have different values in different time windows in order to achieve better precision-speed trade-off:

- HSIMALLOWEDDV on page 47
- HSIMTAUMAX on page 149

- HSIMSPICE on page 143
- HSIMSTEADYCURRENT on page 144

Except for HSIMTAUMAX on page 149, each command can be set within a subcircuit such that the effect covers selected time windows in specified subcircuit.

Example

```
.param HSIMALLOWEDDV=pwc(0, 0.5, 100n, 0.1, 200n, 0.01, 350n, 0.3)
```

This setting assigns piecewise constant values to command HSIMALLOWEDDV on page 47, which is 0.5 from time 0 to 100 ns, then 0.1 from 100 ns to 200 ns, 0.01 from 200 ns till 350 ns, and finally 0.3 after 350 ns.

Note:

Unlike pwl, pwc requires the use of commas. If commas are not used to separate the variables, a syntax error will occur.

Examples of HSIM Simulation Control Commands

The following examples illustrate how and when to set HSIM simulation control commands, and the results of setting the following commands:

- HSIMSPEED on page 125
- HSIMPORTCR on page 109
- HSIMGCAP, HSIMGCAPR, HSIMFCAP, and HSIMFCAPR

HSIMSPEED Example

This example is stored in the directory $HSIM_HOME/tutorial/fadd8. It demonstrates how to control the speed of a simulation.

The steps for running this demonstration follow.

1. Run all the scripts. There are six run scripts for running six simulations with different settings on command HSIMSPEED on page 125.

Table 13 Run Scripts

Script	Runs
run0	Sets HSIMSPEED on page 125 to 0, which is the most precise mode
run1	Sets HSIMSPEED on page 125 to 1
run2, run3, run4, run5	Sets HSIMSPEED on page 125 to 2, 3, 4 and 5, respectively.

2. Observe the CPU time used for each simulation. The simulation speed accelerates with the larger HSIMSPEED value setting, although the speedup is quite insignificant due to the small circuit size in this example.
3. Compare the simulation results of each simulation, using the results of run1 as reference. The simulation results shown in Figure 12 on page 319 are the overlay of HSIMSPEED on page 125=1, 3 and 5. In this example, the waveforms of HSIMSPEED on page 125=1 and HSIMSPEED on page 125=3 are almost identical, but HSIMSPEED on page 125=5 has significant delay difference compared with the other two simulation results.

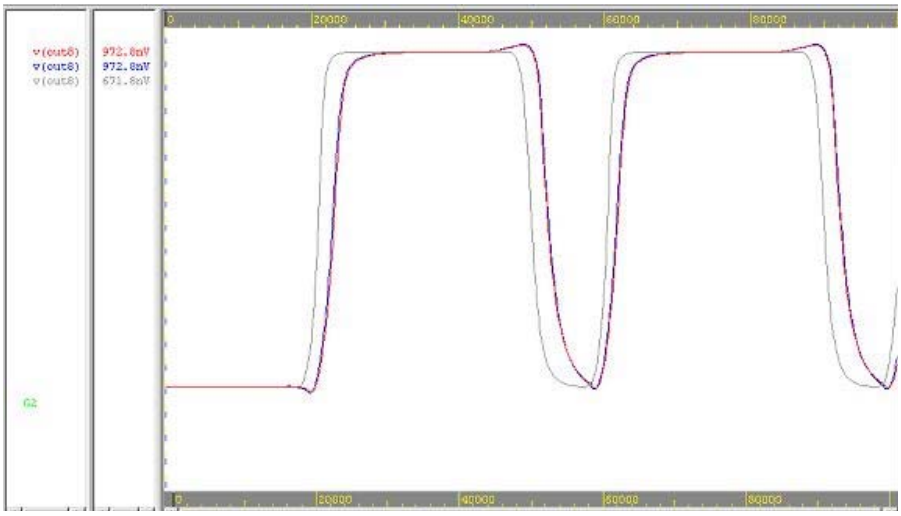


Figure 12 Results of Three Different HSIMSPEED Settings

HSIMPORTCR Example

The HSIMPORTCR on page 109 example is stored in directory $HSIM_HOME/tutorial/isocap. It demonstrates the role of fanout load capacitance in isomorphic matching.

There are two identical circuits that have different control command settings. The circuit contains two similar inverter chains with slightly different port capacitance values.

In the first circuit, each inverter chain has slightly different capacitance values on some nodes.

```
Node 2: 10 fF
Node12: 10.05 fF
```

HSIM will treat the two nodes in the same way because the difference of capacitance is less than the default setting of 1%. Node 3 has 100 fF and node 13 has 101 fF. HSIM will also treat the two nodes in the same way because the percentage difference is the 1 fF divided by the average node capacitance at nodes 3 and 13. The average node capacitance is higher than 100.5 fF due to an inclusion of the average transistor gate and drain/source capacitances connected to node 3 or 13. Node 4 has 100 fF and node 14 has 105 fF, HSIM

will treat them differently because the difference exceeds 1%. Node 5 has 10 fF and node 15 has 10 fF, HSIM will also treat them the same.

First Demonstration

Following are the instructions for this demonstration.

1. Invoke run script run1.
2. View the simulation results.

Observe that the results of node 2 and node12 are the same, and those of node 3 and node 13 are the same. The results of node 4 and node 14 are different and those of node 5 and node15 are different because their inputs are different. Since the differences between the two inverter-chains are relatively small, HSIM can be forced to treat the two inverter-chains in the same way.

Second Demonstration

For the second circuit, the command HSIMPORTCR on page 109=0.05 is used which forces HSIM to match two similar nodes with a capacitance tolerance of 5%. This effectively matches the two inverter chains.

The instructions for this demonstration follow.

1. Execute run script run2.
2. Observe the corresponding nodes. Each inverter chain has exactly the same waveform. When compared to the results of the first simulation, the maximum error at node 15 is approximately 2.5%.

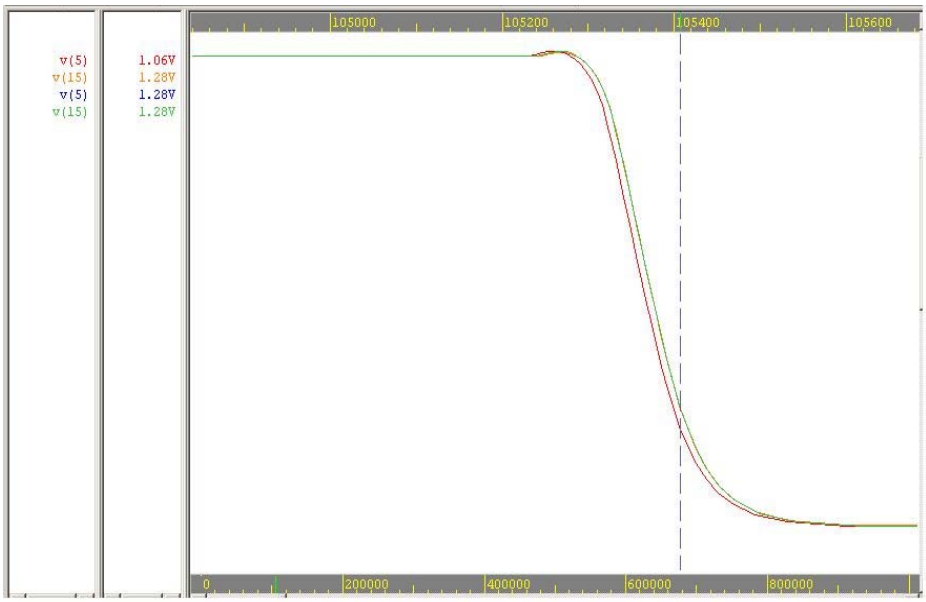


Figure 13 Results of Different HSIMPORTCR Settings

HSIMGCAP, HSIMGCAPR, HSIMFCAP, HSIMFCAPR Example

This example is stored in directory $HSIM_HOME/tutorial/gscap. The example demonstrates how to control the splitting of ungrounded capacitors in a netlist.

This circuit has two identical inverter chains with 0.1 fF, 1 fF, 10 fF, 100 fF or 1 pF cross-coupling capacitors added between them at certain regularly spaced nodes. Depending on the settings of HSIMGCAP, HSIMGCAPR, HSIMFCAP, and HSIMFCAPR, some small ungrounded capacitors might be split to the ground; some large ungrounded capacitors might be kept as the regular capacitors; and the medium ungrounded capacitors will be approximated.

The instructions for this demonstration follow.

1. Execute script run1. This script uses default settings (HSIMGCAP=1fF, HSIMGCAPR=0.02, HSIMFCAP=100fF, HSIMFCAPR=0.5) and results in one split capacitor, three approximated capacitors, and one regular capacitor.
2. Execute script run2. This script sets HSIMGCAP to 1.0E-12 and HSIMFCAP to 1.0E-11, which split all the ungrounded capacitors.

3. Execute script run3. This script sets HSIMGCAP=0.01fF, HSIMGCAPR=0.00001, HSIMFCAP=0.02fF, and HSIMFCAPR=0.00002 and keeps all of the ungrounded capacitors as regular capacitors.

The simulation results are shown in Figure 14 on page 322. The simulation results of run1 and run3 are fairly close, whereas results of run2 are poor. This is to be expected due to a total missing of the coupling effect between the two inverter-chains in run2. The agreement in results between run1 and run3 is excellent. This indicates that the default setting of run1 can result in a very accurate result. The default settings intelligently approximated the coupling effect: splitting the small ungrounded capacitance, modeling the intermediate ungrounded capacitance with approximation, and precisely handling only the large ungrounded capacitance.

These results demonstrate that computing time can be greatly reduced, without losing much precision, when very large circuits are simulated by using this intelligent cross-coupling capacitance handling method.

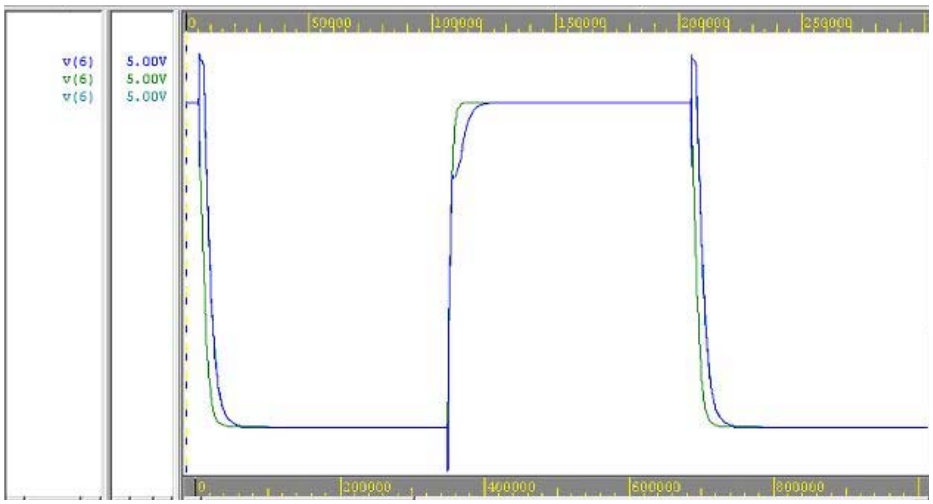


Figure 14 Results of Different HSIMFCAP and HSIMGCAP Settings

9

Post-Layout Back-Annotation

Describes the post-layout back-annotation features of HSIM.

Signal Net Post-Layout Control

The presence of parasitic RCs increases memory usage and slows down the simulation performance. To improve the performance in post-layout simulation containing a large number of parasitic RCs, HSIMPOSTL on page 110 controls the reduction of signal net parasitic RCs to trade off minor precision loss for significant speedup.

Back-Annotation

This section contains information on the following types of back-annotation:

- Device back-annotation
- RC back-annotation

Post-Layout Devices and RC Back-Annotation

A post-layout netlist typically contains:

- Transistors
- Parasitic capacitors
- Parasitic resistors

There are two types of post-layout netlists:

- A self-contained netlist includes all the transistor, resistor, and capacitor elements in the same netlist. The netlist can be flat or hierarchical and can be readily simulated by HSIM.
- A back-annotated netlist has two separate netlists:
  - A netlist of parasitic RCs only, represented in DSPF/SPEF.
  - A transistor netlist without parasitic RCs. A transistor netlist can be either a pre-layout hierarchical netlist or extracted netlist from the layout.

Device Parameter Format (DPF)

If the transistor netlist is a pre-layout hierarchical netlist, the geometric device data can be back-annotated to the hierarchical netlist from a device parameter format (DPF) netlist file. The DPF is a flat SPICE netlist containing only MOSFETs with extracted geometric data. Alternatively, the DSPF file instance section, if present, can also be used to back-annotate device geometry data. For a list of the PF annotation-related commands, see DPF/SPEF Back-Annotation Commands on page 33.

RC Back-Annotation (DSPF/SPEF)

The transistor netlist is ready for simulation but no parasitic RC element is simulated unless the parasitic resistors and capacitors in the DSPF/SPEF netlist are annotated into the transistor terminals through the setting of parameter HSIMSPF, HSIMSPEF on page 129 .

Node Capacitance Back-Annotation

Other than the SPF file, HSIM can also annotate capacitances represented in the following format by the pair of node name and the corresponding node capacitance. The filename of the capacitor file is specified byHSIMCAPFILE on page 59.

HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets

There are two issues related to RC back-annotation for feed-through nets using HSIM:

- Difference between logical and physical pins
- Capacity limits in the RC extractor

In a physical design, multiple physical pins can map to one logical pin which is called the feed-through pin. HSIM reads an original logical netlist. When back-annotating parasitic RC through DSPF files, multiple physical pins merge into one, which creates an inaccurate RC back-annotation. Examples of feed-through nets include:

- Physical Designs: Power Nets
- Memory Designs: Word lines, bit lines, and power nets

Due to the capacity limitations of RC extractors, power net extraction sometimes becomes impossible. One solution aimed at conquering these capacity limitations is to divide the workload by extracting the blocks and top-level separately.

To solve feed-through net and extractor capacity problems, HSIM takes advantage of extended DSPF files to model physical connections. HSIM reads the logical netlist, RC back-annotated by the extended DSPF's, and correctly connects the additional physical pins. Figure 15 on page 326 provides an example of this process.

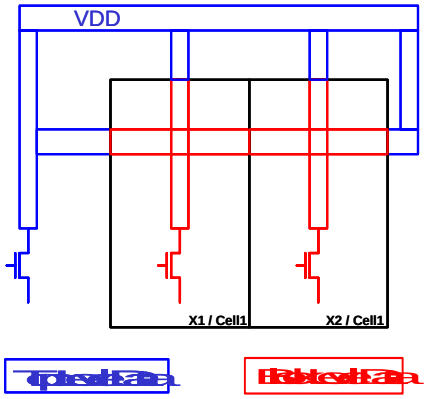


Figure 15 Extended DSPF Example

Figure 15 on page 326 shows the layout of instances X1 and X2. These instances are referenced by the same Cell1 master cell. Geometries shown in blue are top-level data and geometries in red are block-level data, also referred to as cell level data. Two DSPF's are generated by the following extraction flow:

- Extract Cell1 once
- Extract the top-level

Cell1 contains 3 externally connected VDD physical connection points. Logically, one VDD pin is represented in the logical netlist in this example. X1 and X2 are abutted cells. One X1 VDD pin is connected to a X2 VDD pin.

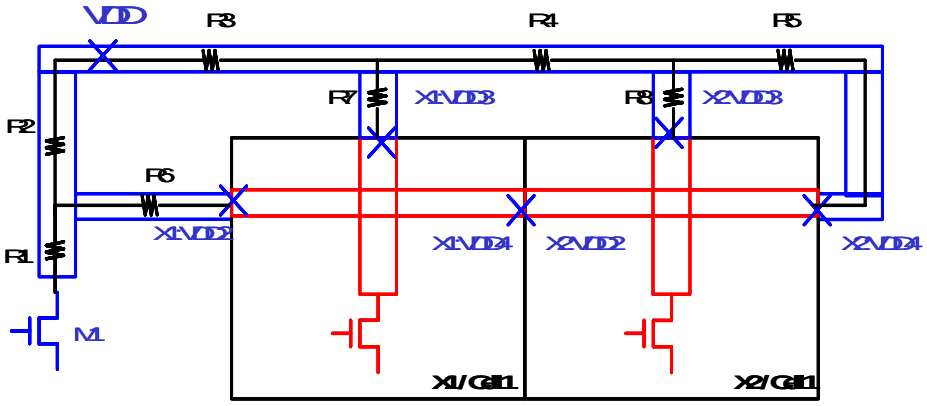


Figure 16 Top-Level Geometry Extraction

In Figure 16 on page 327, the geometry extractions are shown as follows:

- blue-to-blue connections: Top-level geometry extraction to top-level device connections.
- blue-to-red connections: The top-level geometry extraction to hierarchical instance(X1 and X2) pin connections.
- red-to-red connections: The hierarchical blocks recognize X1 and X2 pin connections.

Top-Level DSPF File

An instance name and pin name must be defined for all instance pins connected to the top-level VDD net as shown in the following top-level DSPF file syntax example:

Note:

In the following example, comments following a $ sign are not present in the actual file. These comments are for documentation purposes only.

```
*|NET VDD xxPF                      $Net to be back-annotated
                                    $beginning of the net
                                    $description
*|P (VDD I xxPF X Y)                $Defining pin for the net
*|I (X1:VDD2 X1 VDD2 B xxPF X Y)
*|I (X1:VDD3 X1 VDD3 B xxPF X Y)
                                    $Defining Instance pin
                                    $connection to the net at
                                    $the level top
*|I (X1:VDD4 X1 VDD4 B xxPF X Y)
*|I (X2:VDD2 X2 VDD2 B xxPF X Y)
*|I (X2:VDD3 X2 VDD3 B xxPF X Y)
*|I (X2:VDD4 X2 VDD4 B xxPF X Y)
*|I (M1:SRC M1 SRC B xxPF X Y)
R1 M1:SRC VDD:2 0.1                 $Resistor connects to a top
                                    $level transistor and a top
                                    $level vdd subnet
R2 VDD:2 VDD 0.1                    $Resistor connects to top
                                    $level vdd subnets
R3 VDD VDD:3 0.1
R4 VDD:3 VDD:4 0.1
R5 VDD:4 X2:VDD4 0.1                $Resistor connects to a
                                    $top level
                                    $subnet and an inst. pin
R6 VDD:2 X1:VDD2 0.1
R7 VDD:3 X1:VDD3 0.1
R8 VDD:4 X2:VDD3 0.1
R9 X1:VDD4 X2:VDD2 0.1              $Resistor connects to two
                                    $inst. pins
C1 VDD:2 0 …                        $Cap definition
```

Block-Level RC Extraction

RC extraction of block-level data with pin definition which is based on pin assignment consistent to the top-level as shown in Figure 17 on page 329.

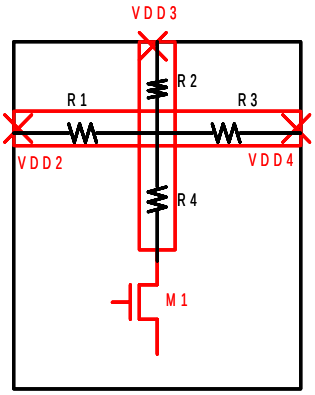


Figure 17 Block-Level Data Pin Definition

Block-Level DSPF File

The following is an example of the block-level DSPF file:

```
.subckt Cell1                        $Needs .subckt definition if
                                     $not top level
*|NET VDD xxPF                       $Net definition
*|P (VDD2 I xxPF X Y)
                                     $Pin definition which makes
                                     $external
*|P (VDD3 I xxPF X Y)                $connection at one level up
                                     $in the hierarchy
*|P (VDD4 I xxPF X Y)
*|I M1:SRC M1 SRC
R1 VDD2 VDD:2 0.1
R2 VDD3 VDD:2 0.1
R3 VDD4 VDD:2 0.1
R4 VDD:2 M1:SRC 0.1
…
```

Enhanced HSIM Subcircuit Instance Parameters

Ths subcircuit instance description is used to support hierarchical DSPF flows. The location and orientation of hierarchically extracted subcircuits are with respect to the instance's local axes, which need to be translated into the global axes.

Star-RCXT provides HSIM^{plus} with a Hierarchical Instance position file containing the location information for each hierarchical instance.

Hierarchical Instance Position File

To generate a placement file generated by Star-RCXT, add the following commands in the Star-RCXT cmd_file when running extraction:

SKIP_CELL_PLACEMENT_INFO_FILE: YES
SKIP_CELL_PLACEMENT_INFO_FILE: <filename> .

If no filename is specified, Star-RCXT will dump out a file named blockname.placement_info. The placement file will contain the following information: Angle, Reflection, Location of Cell, Cell Name, and Cross-referenced Instance Name. This placement file is then read into HSIM using the HSIMSPFPOS parameter.

The following example shows sample output syntax.

Example 24 Placement File Generated by Star-RCXT

```
* SKIP CELL PLACEMENT FILE
* VENDOR "Synopsys, Inc."
* PROGRAM "Star-RCXT X-2005.06" - Program name and version
* DATE "Mon Dec 5 11:32:56 2005" - Date when file was generated
* UNIT "MICRONS"
TOP_CELL=<cell_name> <instance_name> <cell_name> <X-coord> <Y-
coord> <Angle> <reflection>
XSI_0 INV1 49 132 0 NO   - a cell not rotated or flipped
XSI_50 XOR2 484 132 180 NO  - a cell rotated 180 counter-clockwise
(ccw)
XSI_61 XOR2 124 312 180 YES  - a cell reflected about the X-axis
and then rotated 180 degrees ccw
```

Note:

If both rotation and reflection are necessary, the instance will first be reflected about the x-axis and then rotated counter-clockwise by the angle provided. These operations follow the GDSII standard syntax.

10

Simulation Output

Describes HSIM output formats including: FSDB, Nassda/Synopsys, WSF, Other ASCII. The chapter discusses output control statements, digital vector file parameters, and vector data formats.

HSIM Output Formats

This chapter describes HSIM output formats. The HSIMOUTPUT on page 105 command sets the desired output format. For a list of the HSIM output commands, see Output Control Commands on page 36.

The available formats are listed in Table 53. The full descriptions and limitations of the formats are defined in the following sections. Click on the Format Type to view the descriptions.

Table 53 HSIM Output Formats

Format Type	Supported Viewer
FSDB Output Format	Binary format supported by the nWave waveform display tool. This is the HSIMOUTPUT default.
Nassda Output Format	NOF is a unique text output format in HSIM.
WDF Output Format	Binary format supported by the Sandwork SPICE Explorer waveform display tool.
WSF Output Format	A Cadence® format that stores output waveform data.
.out ASCII Output Format	ASCII format similar to a combined control and data file.
PSF Output Format	psfbin is a binary file for Cadence waveform viewer.

Table 53 HSIM Output Formats (Continued)

Format Type	Supported Viewer
PSF Float Output Format	psfbinf is a binary-float file for Cadence waveform viewer.
UTF Output Format	Binary file for Veritools waveform viewer.
rawfile Output Format	ASCII file for the Berkeley SPICE output format.
Simultaneous Multiple Output Files	Multiple output files can be produced with the use of ampersand (&) symbol between the format specifiers.

FSDB Output Format

FSDB is a binary format supported by the nWave™ waveform display tool. Since the fsdb is a binary file, the file size is typically half the size of an ASCII output file. Additionally, a binary file is processed more efficiently by the nWave waveform viewer. Signal values are stored as floats by default. In order to store the values as double, set HSIMFSDBDOUBLE on page 79.

Nassda Output Format

The Nassda Output Format (NOF) is a unique text output format in HSIM. This output format is set when the control parameter HSIMOUTPUT is set to nassda.

Example 25

```
.param HSIMOUTPUT=nassda
```

This output format consists of two files: a control file (.ctr) and a data file (.out). Following is the Backus Naur Form (BNF) of the two files.

```
<nassda_ctr_file>        ::=<ctr_settings><signal_list>
<ctr_settings>           ::=(<comments>|<setting>)*
<signal_list>            :=(<comments>|<signal>)*
<comments>               :=# <anything> <EOL>
<setting>                :=<keyword> <value> <EOL>
<keyword>                :=timescale|vscale|iscale
<signal>                 :=<id> <name> <EOL>
<nassda_out_file>        :=(<time_slot>)*
<time_slot>              :=<time_line> (<data_line>)+
<time_line>              :=<time> <EOL>
<data_line>              :=<id> <value> <EOL>
<time>                   :=integer
<id>                     :=integer
<value>                  :=double
<name>                   :=string
<EOL>                    :=end-of-line
```

A utility program hs2tbl is provided with HSIM software. The program converts NOF to tabular data output that can be displayed with either of the following:

- Xgraph In the Xwindows environment
- Dplot In the Windows NT environment.

Example 26 Nassda Control File

```
#hsim plot control file
#Version 6.0
timescale 1
vscale 0.1
iscale 0.001
0 v(bl0n)
1 v(en)
2 v(bl0)
3 v(vddx)
4 v(bl5n)
5 v(enb)
6 v(gndx)
7 v(io0)
8 v(rd)
9 v(wr)
10 v(pch)
11 v(inio0)
12 i(vsu)
```

Example 27 Nassda Data File

```
#hsim plot file
#Version 6.0
0 15
1 0
2 15
3 17.6646
4 15
5 30
6 13.2823
7 2.81737e-007
8 0
9 0
10 0
11 0
12 -0.121817
2000
12 -0.124846
4000
12 -0.00316119
4800
10 0
11 0
12 -0.00106606
4806
0 14.9968
2 14.9968
4 14.9968
7 2.81737e-007
10 0.99
11 0.15
```

WDF Output Format

WDF is a binary format supported by the Sandwork SPICE Explorer waveform display tool. HSIM generates an .wdf output file in WDF format if HSIMOUTPUT is set to wdf.

```
.param HSIMOUTPUT=wdf
```

Simultaneous Multiple Output Files

Multiple output files can be produced with the use of ampersand (&) symbol between the format specifiers.

To produce FSDB and NOF formatted output files, set the following parameter:

```
.param HSIMOUTPUT="fsdb&nassda"
```

The FSDB format can be combined with any other format or formats. Certain combinations of the output formats are not allowed such as:

- out&nassda: Because both formats produce .out file.
- nassda&wsf: Because both formats use the COI interface.

WSF Output Format

WSF is a Cadence format that stores output waveform data. HSIM generates a .wsf output file when HSIMOUTPUT is set to wsf.

```
.param HSIMOUTPUT=wsf
```

.out ASCII Output Format

The .out format is an ASCII format similar to a combined nassda control and data file. HSIM generates a .out file if HSIMOUTPUT is set to out.

```
.param HSIMOUTPUT=out
```

PSF Output Format

psfbin is a binary file for Cadence Waveform viewer. HSIM generates PSF binary format when HSIMOUTPUT set to psfbin.

```
.param HSIMOUTPUT=psfbin
```

Note:

If the PSF binary output file size exceeds 2GB, the output file size increases even when you also use the HSIMOUTPUTTRES command. To maintain the effect of HSIMOUTPUTTRES in conjuction with psf binary output format, specify HSIMOUTPUT=psfbin2 (instead of psfbin). However, doing so imposes a 2GB file size limit.

PSF Output Format

psfbin is a binary file for the Cadence Waveform viewer.

```
.param HSIMOUTPUT=psfbin
```

Note:

If the PSF binary output file size exceeds 2GB, the output file size increases even when you also use the HSIMOUTPUTTRES command. To maintain the effect of HSIMOUTPUTTRES in conjuction with psf binary output format, specify HSIMOUTPUT=psfbin2 (instead of psfbin). However, doing so imposes a 2GB file size limit.

You can generate multiple output formats. For example:

```
.param HSIMOUTPUT=psfbin51&&psfbin
```

Generates both PSF V5.1.4 and V6.1 output formats.

```
.param HSIMOUTPUT=psfbin51&wdf
```

Generates both PSF V5.1.4 and wdf output formats.

PSF Float Output Format

psfbinf is a binary file for Cadence Waveform viewer. HSIM generates PSF binary format when HSIMOUTPUT set to psfbinf.

```
.param HSIMOUTPUT=psfbinf
```

UTF Output Format

utf is a binary file for Veritools' Waveform Viewer. HSIM generates `utf` binary format when HSIMOUTPUT set to `utf`. It uses version 2007.1.2 of the

Veritools UTF API.

```
.param HSIMOUTPUT=UTF
```

HSIM searches for the lib<out_format>.so file in the following directories, in the order shown:

- Run directory
- $HOME directory
- $HSIM_HOME/platform/<port>/bin
- LD_LIBRARY_PATH on Solaris and on Linux, and SHLIB_PATH on HP

<output_format> is the value of HSIMOUTPUT and all the letters in <output_format> are in upper case.

Note:

Please contact Synopsys to obtain the latest libUTF.so library.

rawfile Output Format

rawfile is an ASCII file for the Berkeley SPICE output format. HSIM generates raw format when both HSIMOUTPUT is set to fsdb and HSIMOUTPUTTBL is set to rawfile.

Example

```
.param HSIMOUTPUT=fsdb
.param HSIMOUTPUTTBL=rawfile
```

Output Control Statements

The output control statements include the following:

.print/.probe/.plot/.graph

Prints the output. Each of the statements is indistinguishable from each other in HSIM.

.lprint

Prints the logic waveforms of output data.

.print window

Prints output data only within the specified time window.

.store/.restore

Saves the intermediate simulation data and restores the saved simulation data.

.measure

Prints the results of user-specified defined analysis.

.four

Prints the results of Fourier analysis.

.fft

Prints the results of Fast Fourier analysis.

Syntax and descriptions are shown in the following sections.

PRINT/.PROBE/.PLOT/.GRAPH Statements

HSIM produces output waveforms from the following statements:

- .print
- .probe
- .plot
- .graph

All these statements are treated in the same way in HSIM.

The .print statement can be placed either at the top-level netlist or within any subcircuit definition. If it is located within the subcircuit definition, the scope of printout only covers node voltages or element currents within that particular subcircuit.

HSIM does not support the HSPICE command .option post. This command dumps all node voltage waveforms, all current waveforms of independent voltage sources and waveforms from .print and .probe statements. HSIM does however provide similar output commands as described below.

Note:

Unless an ASCII output mode is specified, HSIM only produces a waveform output. HSIM does not produce SPICE .print type tabular format.

Note:

The syntax for .print is also valid for .probe, .plot, and .graph.

Note:

The syntax for .print <file=*name*> and <format=*name*> must be specified together. Using one or the other alone will be ignored. However, multiple formats can be specified by repeatedly specifying format=*name*.

Syntax

```
.print <tran> <name1=>ov1 <<name2=>ov2 ... >
   <subckt=sub_name> <level=val2> <matchport=val3>
   <outputres=val4> <adonly=1> <branch=name> <file=name>
   <format=name>
```

The output variables `ov1` and `ov2` specify the type of outputs. Supported output types are shown in Table 54 on page 339.

Table 54 Output Variable Types

Types	Description
v(node_name)	Prints the node voltage waveform of the specified node_name. In case the node_name contains asterisk (*) wildcard character, HSIM will print all the voltages of matched nodes. In addition to the asterisk (*) wildcard character, the simulator also supports the single question mark (?) wildcard character so that the waveforms of all the matched nodes are printed.
i(element_name)	Prints the branch current waveform through the element element_name. The element name may contain the asterisk (*) wildcard character and/or question mark (?) wildcard character.
v(node1, node2)	Prints the difference v(node1)-v(node2). No wildcard character is allowed in either node1 or node2.
par(‘expr’)	Prints the functional value of the expression expr. No wildcard character is allowed in the expression expr.
X(subckt_node_path)	Prints the current flowing into the port of the specified subcircuit. The name specified by subckt_node_path contains combined information of hierarchical_path_name.vname; the full hierarchical path name of the subcircuit is specified by hierarchical_path_name; vname denotes the node name of a port of the subcircuit or the name of a global node which connects to elements within the subcircuit. A wildcard can not be used in the vname but can be used in a hierarchical path name. This feature allows printing the subcircuit block current of the specified subcircuit instance. When HSIMXSW is set, the total switching current of the subcircuit that is connected to the port is reported.

Table 54 Output Variable Types (Continued)

Types	Description
in(node_name)	Prints the node current waveform flowing into the specified node_name. The node_name may contain the wildcard characters * or ? to match nodes by pattern. Note: This node current printout may be very CPU-intensive, especially for high connectivity nodes like VDD, therefore is should be used discreetly. For high connectivity nodes it is better to use an element current printout such as i(VVDD). Wildcard characters such as an asterisk (*) should not be used for all nodes in a netlist. Refer to the interactive command ni on page 418 for a node current description.
isw(element_name)	Prints the total switching current connected to the element. The element must be a voltage source connected to ground. Switching current is defined as the positive current that is charging either a node to grounded capacitor or a coupling capacitance, the discharging current is not considered.

Optional Settings

The optional settings of the matchport and level parameters are specified to control the scope of wildcard match. The subckt setting allows printouts for all instances of the specified subcircuit name. Optional settings are shown in Table 55 on page 340

.

Table 55 Optional Settings for .Print/.Probe/.Plot/.Graph Statements

Setting	Description
subckt=sub_name	Prints the node voltage(s) and/or element current(s) of the output variable(s) specified in the same .print statement. The printout applies to the specified node name(s) and/or element name(s) within all instances of the specified subcircuit name. This subckt setting is equivalent to placing the .print statement within the subcircuit definition.

Table 55 Optional Settings for .Print/.Probe/.Plot/.Graph Statements

Setting	Description
level=val2	This setting is effective only when the wildcard character is specified in the output variable. The level value val2 specifies the number of hierarchical depth levels when the wildcard node/element name matches. ▪ When val2 is set to 1, the wildcard match applies to the same depth level where the .print statement is located. ▪ When val2 is set to 2, it applies to the same level and to one level below the current level where .print is located. ▪ When val2 is set to -1, the wildcard match applies to all the depth levels below and including the current level of .print statement. ▪ The default value of val2 is -1.
matchport=val3	This setting is effective only when the wildcard character is specified in the output variable and the output variable type is voltage, i.e. v(node_name). ▪ When val3 is set to 1, the wildcard matching extends to match the port nodes of the subcircuit instance name. ▪ When val3 is set to 0, the port nodes are excluded from the wildcard match. ▪ The default value of val3 is 0 (zero).
outputres=val4	outputres=val4 allows specifying the output resolution for all the print items listed in the .print statement. If outputres is not specified, values from the global parameters HSIMOUTPUTVRES on page 108 and HSIMOUTPUTIRES on page 106 will be used.
name1, name2	The optional names name1 and name2 define the output variables ov1 and ov2 as name1 and name2, respectively. A wildcard character is not allowed in the output variable expression when it is redefined for another name.
adonly=1	If adonly is specified as 1, then only the nodes that connect to at least one active device will be printed. The setting is effective only when the wildcard character is specified in the output variable and the output variable type in voltage, i.e. v(node_name).

Table 55 Optional Settings for .Print/.Probe/.Plot/.Graph Statements

Setting	Description
filter=pattern	When printing node voltage(s) and/or element current(s) that are specified by wildcard patterns such as: .print v(x1.x2.*), nodes/elements that match the pattern specified in the filter clause will not be printed. Each filter applies to all wildcard voltages/currents being printed on the .print statement. For example: `.print v(x1.x2.*) i(x1.x2.*) filter='x1.x2.n*' filter='x1.x2.a*'` This syntax example will print the voltages of all nodes in subckt x1.x2 that do not start with n or a, and the current of all elements in subckt x1.x2 that do not start with either n or a.
branch=name	To print an entire branch in the circuit, simply write .print branch='branch_name'. No output variable is required when using the branch option. Wildcard characters are not supported in the branchname: .print branch='x1.x2.x3' is equivalent to the following: .print v(x1.x2.x3.*) level=1 .print v(x1.x2.*) level=1 .print v(x1.*) level=1
file=name format=name	The file=and format=options allows users to direct results of the current print statement to a specified file. In order for this feature to be activated, you must specify both the filename with file=and the format name with format=. Also, note that the formats specified with format= does not necessarily have to be specified with the HSIMOUTPUT parameter. For example: .param HSIMOUTPUT=fsdb .print v(n1) file=foo format=out .print v(n2) file=bar format=wdf .print v(n3) In this example, v(n1) will be written to foo.out, v(n2) to bar.wdf, and v(n3) to hsim.fsdb (since that is the default output file according to the HSIMOUTPUT parameter). Note that now for this feature to be activated, you must specify BOTH file=and format=. If either one is missing, the results will be printed to the default file.

Examples

The following are optional setting examples.

```
.subckt samp ......
.print v(x1.*) v(q*)
..................
.ends
```

Prints node voltages for all nodes which match x1.* and q* in all instances of subcircuit samp.

```
.print v(x1.*) v(q*) subckt=samp
```

Same as the first example which places the `.print` statement within the subcircuit definition of samp.

```
.print v(*)
```

Prints all node voltages of the circuit.

```
.print v(*) level=1
```

Prints all top-level node voltages.

```
.print v(x1.*) level=3
```

Prints node voltages within subcircuit instance x1. The printout includes nodes within x1, and the nodes in two levels below x1. The nodes located more than two levels deeper than the level of instance x1 are excluded.

```
.print v(x1.*) level=3 matchport=1
```

Prints all the node voltages as in the previous example and each port node voltage of instance x1.

```
.print diff=v(x1.a, x2.b)
```

Prints the node voltage difference v(x1.a)-v(x2.b). The variable diff is defined as the node voltage difference.

```
.print i(mp1) subckt=latch
```

Prints the branch currents for element name mp1 in all instances of subcircuit latch.

```
.print tran I_cur=par('I(x1.mp1) + I(x2.mp2) - I(x1.x2.mp1)')
```

Prints I_cur which is defined as the functional value of the following:

```
I(x1.mp1) + I(x2.mp2) - I(x1.x2.mp1).
.print wrong=v(x1.*)
```

This print statement will be ignored. The wildcard is not allowed in case the output variable is redefined for another name.

```
.print x(x1.x2.vcc)
```

Prints the subcircuit block current entering the vcc port of the subcircuit instance x1.x2.

```
.print V(x1.*) adonly=1
```

Prints all the nodes, inside the instance x1, that are connected to at least one active device.

```
.print in(n*)
```

It will print current waveform of all nodes that match n*.

Print Average and RMS Currents

For the printout of currents or block-level currents, HSIM also supports the printout of their average and RMS values. The average results will be written to a file that uses the output prefix plus the avg suffix. For instance, if the output prefix is output, the average current values will be written to the file named output_avg. RMS results will be written to a file using the output prefix plus the rms suffix. For example, if the output file name is output, the RMS current values will be written to the file named output_rms.

```
.print <tran> <name=>ov window=time_win start=time1
   stop=time2 step=time_step
```

ov

Specifies the output variable. The eligible types are i(element_name) and x(subckt_node_path) only. See the descriptions stated in Table 54 on page 339 for the usage on i(element_name) and x(subckt_node_path).

Note:

Only one output variable is allowed in each statement.

name

Defines the output variable ov as name. If name is used, wildcard characters are not allowed for the output variable, ov.

window=time_win start=time1 stop=time2 step=time_step

The current waveform is averaged and RMSed over the time interval specified by the parameter window. The first calculation begins at the time specified by start and covers the time interval between start-window/2 and start+window/2. The calculations repeat at the increment specified by step and ends at the time specified by stop.

Example

```
.print i(m1) window=6n start=30n stop=100n step=10n
```

Prints the average and RMS branch current for element name m1. The time window for first calculation is from 27 ns to 33 ns (the centered start time is 30 ns) and repeats at 10 ns. The last calculation centered at 100 ns.

```
.print x(xadd.vdd) window=10n start=10n stop=70n step=20n
```

Prints the average and RMS subcircuit block current entering the V_{DD} port for the instances xadd. The time window for first calculation is from 5 ns to 15 ns (the centered start time is 30 ns) and repeats at 20 ns. The last calculation centered at 70ns.

```
.print in(m2) window=10n start=5ns stop=100n step=20n
```

Prints the average and RMS current for node name m2. The time window for first calculation is from 20ns to 30ns (the centered time is 25ns) and repeats every 20ns. The last calculation time centered at 85ns.

.lprint Statement

HSIM generates logic waveforms from .lprint statements. The signal can be 1/0/U. If the signal is used to cover Hi-Z(H) and Lo-Z(L), HSIMHZ=1 must be set. The default value is 0. The .lprint statement syntax is similar to .print.

```
.lprint <tran> <vlth> <vhth> <name1=>ov1 <<name2=>ov2 ….>
   <subckt=sub_name> <level=val2> <matchport=val3>
   <adonly=1>
```

vlth

Threshold voltage for LOW logic state. The 0 (zero) state is printed if the node voltage is less than or equal to vlth. If the node voltage is between vlth and vhth, the X state is printed. Its default value is the value given by the HSIM global parameter HSIMVLTH on page 169.

vhth

Threshold voltage for HIGH logic state. The 1 (one) state is printed if the node voltage is higher than or equal to vhth. If the node voltage is between vlth and vhth, the X state is printed. Its default value is the value given by the HSIM global parameter HSIMVHTH on page 167.

ov1, ov2

Specifies the output variables in the format of v(node_name). In case the wildcard contains wildcard characters, '*' or '?', HSIM will print the logic states of all of the matched nodes.

<name1>, <subckt>, <level>, <matchport>, <adonly>

Optional settings; see .PRINT/.PROBE/.PLOT/.GRAPH statements.

Note:

If vlth and vhth are not specified in the .lprint statement and HSIMVLTH on page 169 and HSIMVHTH on page 167 are not explicitly specified in the netlist, HSIM will issue a warning message and the 30% and 70% of supply voltage thresholds, defined by HSIMVDD on page 165, will be used. If a design involves multiple power supplies, use HSIMAUTOVDD on page 51 to automatically set HSIMVLTH and HSIMVHTH in accordance with the circuit's supply voltage.

Example

```
.subckt samp ...........................
.lprint 1 3 v(*)
.ends
.lprint v(x1.n1)   v(x1.x2.*)
```

This example prints all the node logic waveforms for all nodes in each instance of subcircuit samp with LOW threshold voltage of 1V and HIGH threshold voltage of 3V, the logic waveform at node x1.n1, all node logic waveforms within the subcircuit instance x1.x2.

.STORE/.RESTORE Statement

The .store statement saves the intermediate simulation data. The .restore statement restores the saved data file.

.STORE

```
.store <file=simul_file> <time=time1 <time=time2 ...> >
   <repeat=time_interval> <split=1>
```

If no simul_file name is specified, then the default file name is <output_prefix>.iic. If the repeat parameter value is specified, then only the first time value, time1, is effective, and the store operation is performed in every time_interval. The default time is 0. split=1 will split the intermediate simulation data into multiple files with time1 and time2 suffixes. time1 and time2 will be converted into pico seconds to become the suffix of the file name.

For information about <output_prefix>, refer to Chapter 5, Running HSIM.

Example

```
.store file=simul_file1 time=123n repeat=1000n
```

The simulation data are stored at time being 123 ns, 1123 ns, 2123 ns, and so on.

.RESTORE

.restore <outformat=file_format> <file=simulation_file> <time=time1> <outfile=wave_file>

The filename information is required for the .restore statement. The keyword file= is optional. If the time parameter information is not specified, then the most recently saved simulation data in the saved file will be restored. If the outfile parameter is specified, the waveform for each output variable from the wave_file will be added to the output. The fsdb and wdf output formats are currently supported. The outformat parameter specifies the waveform format to be added. The default for outformat is fsdb. The simulation file can be a full path file name or a file prefix.

Caution!

When using output to restore, do not use the same -o <prefix> used in the previous simulation. Using the same prefix WILL corrupt the save file as well as the previous .fsdb file.

Examples

```
.restore file=simul_file1 time=1000n
```

The simulation data are restored at time 1000n from the previously stored simulation data file, simul_file1.

```
.restore outformat=wdf  file=simul_file1 time=1000n
outfile=test.wdf
```

The simulation data are restored at time 1000n from the previously stored simulation data file, simul_file1. Also, the waveform for each output variable from test.wdf will be added to the output waveform file in .WDF format.

.measure Statement

Measurement output format can be controlled by the following option parameters:

```
.option ingold=x
.option measdgt=x
```

Table 56 shows the command syntax and output descriptions for ingold and measdgt.

Table 56 Command Descriptions

Syntax	Output Description
.OPTION AUTOSTOP	Terminates the simulation after completing all .measure statements. In HSIM, the support of this feature is limited. Measure types supported: Find and When Measurements; Trig and Targ Measurements; Equation Evaluation for temporary output variables only. Limitation: there should not be any dependencies among measurement statements. Also, waveform comparison is not supported.
.OPTION INGOLD=0	HSIM print values in engineering format as follows: 1g=1e9, 1x=1e6, 1k=1e3, 1m=1e-3, 1u=1e-6, 1n=1e-9, 1p=1e-12, 1f=1e-15
.OPTION INGOLD=1	HSIM print values in engineering format for values between 0.1 and 999. Any other values will be printed in exponential format.
.OPTION INGOLD=2	[default] HSIM print values in exponential format
.OPTION MEASDGT=6	HSIM will print values with 6 decimal digits.
.OPTION MEASDGT=12	[default]

HSIM supports the .measure statement. The output variables which can be measured by the .measure statement are node voltage v(n1), voltage difference v(n1,n2), branch current i(m1), and any .measure result variable. The following measurement types are supported. All .measure results are saved in the file out_file.mt.

For information about out_file, refer to Chapter 5, Running HSIM.

Rise, Fall, and Delay Time Measurements

```
.measure <tran> result trig ... targ ...
```

trig

Trigger. The syntax for trig is:

```
trig trig_var val=val2 <td=td1> <cross=val3> <rise=val4>
<fall=val5>
trig at=val6
```

targ

Target. The syntax for targ is:

```
targ targ_var val=val2 <td=td1> <cross=val3|last>
<rise=val4|last> <fall=val5|last>
```

Arguments for trig and targ

cross

The syntax for cross is:

```
cross=val3, rise=val4, and fall=val5
```

In the syntax for cross=val3, rise=val4, and fall=val5, the val3, val4, and val5 values can be either numbers or the keyword last. A number indicates which occurrence of a cross, rise, or fall event is used. If the keyword last is used, the last occurrence of a cross, rise or fall event is used.

```
fall/rise/cross=-1, -2 ... (in addition to 'last')
```

td

The syntax for td is:

```
td=td1
```

td=td1 is the amount of simulation time that must elapse before a measurement is performed. The number of cross, rise, or fall occurrences is counted after the required td1 simulation time.

val

The syntax for val is:

```
val=val2
```

val=val2 is the value of a target or trigger variable at which the cross, rise, or fall counter is incremented by one.

```
trig_var and tart_var
```

The syntax for trig_var and tart_var is:

trig_var and tart_var are the output variable names for trig and targ.

Example

```
.measure tran tw0 trig v(dep) val=1.5 fall=4 targ v(ffarb) val=0.5
td=10n rise=1
.measure <name> when v(<name2>)=<val> rise=<-1/-2/last>
from=<val> to=<val>
```

Derivative Measurements

Derivative measurements use one of the following three syntax statements:

```
.measure <tran> result derivative var at=val
.measure <tran> result derivative var1 when var2=val1
   <td=td1> <rise=val2|last> <fall=val3|last>
   <cross=val4|last>
.measure <tran> result derivative var1 when var2=var3
   <td=td1> <rise=val1|last> <fall=val2|last>
   <cross=val3|last>
```

Example

```
.measure tran slope derivative v(n) at=10n
```

This calculates the derivative of v(n) when transient analysis is at 10ns.

.measure tran rise_tr derivative v(n) when v(n)='VDD/2' rise=2

This calculates the derivative of v(n) when v(n) reaches half of the VDD voltage level at 2nd rising edge.

.measure tran cross_tr derivative v(n) when v(n1)=v(n2)

This calculates the derivative of v(n) when both node voltages of n1 and n2 are equal.

Average, RMS, MIN, MAX, Peak-To-Peak Measurements

```
.measure <dc> result func out_var1 from=time1 to=time2
```

func can be one of the following reserved words.

avg

Average

max

Maximum

min

Minimum

pp

Peak-To-Peak

rms

Root Mean Square

integ

Integral

Example

```
.measure avg_ivdd avg i(vdd) from=10n to=250n
```

Find and When Measurements

```
.measure <tran> result when out_var1=val2 <td=td1>
   <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure <tran> result when out_var1=out_var2 <td=td1>
   <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure <tran> result find out_var1 when out_var2=val2
   <td=td1> <rise=val3|last> <fall=val4|last>
   <cross=val5|last>
.measure <tran> result find out_var1 when out_var2=out_var3
   <td=td1> <rise=val3|last> <fall=val4|last>
   <cross=val5|last>
.measure <tran> result find out_var1 at=val2
```

Refer to the examples in Equation Evaluation on page 352.

Equation Evaluation

```
.measure <tran> result param='expr'
```

Example

```
.measure teq when v(rb)=v(rt)
```

.measure veq find v(eq) when v(rb)=0.5 fall=3

.measure eq_sum param='teq + veq'

.measure statement 3 above evaluates the expression that is a function of results from .measure statement(s) 1 and 2. The expression for param can not be a function of branch current(s) or node voltage(s).

Continuous Measurement

The continuous measurement feature of HSIM simulator is used by specifying the tran_cont option in the .measure statement. This type of measure will perform the specified measurement continuously until the end of simulation or whenever the measurement condition is fulfilled.

```
.measure tran_cont result trig ... targ ...
.measure tran_cont result when out_var1=val2 <td=td1>
   <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure tran_cont result when out_var1=out_var2 <td=td1>
   <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure tran_cont result find out_var1 when out_var2=val2
   <td=td1> <rise=val3|last> <fall=val4|last>
   <cross=val5|last>
.measure tran_cont result find out_var1 when
   out_var2=out_var3 <td=td1> <rise=val3|last>
   <fall=val4|last> <cross=val5|last>
```

A specific measure output file will be created, <output_prefix>_<result>.mt, to store the continuous measurement results.

Example

```
.measure tran_cont cont_vout1 find v(out1) when v(a1)=2.5 fall=1
```

.measure statements will continuously find the voltage out1 when the voltage value of node a1 reaches to 2.5 starting from the first falling edge. In this example, the additional output filename is hsim_cont_vout1.mt if the output file specifier is not used. hsim_cont is the output_prefix.

```
.measure tran_cont cont_vout1 when v(a1)=2.5 fall=2
```

The .meas statement will continuously report the time when the voltage value of node a1 reaches 2.5V, starting from the second falling edge.

```
.measure tran_cont cont_va1_pulse trig v(a1) val=2.5 rise=1 targ
v(a1) val=2.5 fall=1
```

The .measure statement will continuously measure the pulse width of node a1. The output filename will be hsim_cont_va1_pulse.mt. hsim_cont is the output prefix.

Jitter and Histogram Report

Jitter specifications are of interest in PLL design. By using continuous measurement, each cycle period after lock-in can easily be measured. With the jitter parameter in the continuous measurement command, HSIM reports the jitter value and a histogram to show whether the design is within the specifications. In jitter=nb, nb is the number of bins.

A jitter histogram is evenly divided into nb between the minimum and maximum results. Both the histogram and jitter will be stored in the <output_prefix>_<result.mt.hist> file. Example 28 illustrates the example syntax used to produce the associated histogram.

Example 28

```
.measure tran_cont td trig v(pllclk) val=0 td=4500ns rise=1 targ
v(pllclk) val=0 td=4500ns rise=2 jitter=10
/* Add this parameter into continuous measurement command */
```

In the hsim_td.mt.hist file:

```
meas_variable=td
jitter=4.180651e-012
min=9.996815e-009 max=1.000100e-008
run_min=1 run_max=81
9.997233e-009, nb=1, freq=1.010101e-002 |
9.997651e-009, nb=2, freq=2.020202e-002 |*
9.998069e-009, nb=4, freq=4.040404e-002 |***
9.998487e-009, nb=11, freq=1.111111e-001 |********
9.998906e-009, nb=11, freq=1.111111e-001 |********
9.999324e-009, nb=26, freq=2.626263e-001 |*************
9.999742e-009, nb=20, freq=2.020202e-001 |********
1.000016e-008, nb=18, freq=1.818181e-001 |********
1.000058e-008, nb=4, freq=4.040404e-002 |***
1.000100e-008, nb=2, freq=2.020202e-002 |*
```

.FOUR Statement

HSIM supports Fourier transformation. Any valid voltage or current output variable can be used in the .four analysis. The results are stored in a tabular format in the .log file. In addition, one .ft file is generated for each .four output variable.

```
.four   freq1 out_var1 <out_var2 …>
```

The variable value freq1 specifies the fundamental frequency for the analysis, while the output variable(s) are indicated as out_var1, out_var2, and so on.

Example

```
.four 20Meg v(2) v(3)
```

.FFT Statement

HSIM supports Fast Fourier Transform (FFT) analysis. Any valid voltage or current output variable can be used in the FFT analysis. The results of FFT analysis are stored in a tabular format in the .log file. In addition, one .ft file is generated for each FFT output variable.

```
.fft out_var1 <start=time1> <stop=time2> <np=fft_np>
   <format=norm|unorm > <window=win1> <alfa=a1>
   <freq=freq1> <fmin=freq2> <fmax=freq3>
```

out_var1 can be any valid voltage or current output variable. Parameters start and stop specify the start time and end time of the analysis, respectively. The default value for start is 0, and default value for stop is stoptime in the .tran statement. Refer to Chapter 7, Input Netlist, .tran on page 300 for details. FFT statements are described in Table 57 on page 354.

Table 57 FFT Statements

Parameter	Description
alfa	Used with Gauss or Kaiser window for controlling the highest sidelobe level, bandwidth, and so on. A valid value is 1 <=alfa <=20. The default value is 3.
freq	Specifies the frequency of interest. If freq value is nonzero, the output listing is limited to the harmonics of this frequency. The default value is 0 Hz. The Total Harmonic Distortion (THD) for the harmonics is also printed.

Table 57 FFT Statements (Continued)

Parameter	Description
fmin	Specifies the lower frequency boundary for the FFT output to be printed to the logfile. The default value is 1/T (Hz) with T=(stop – start).
fmax	Specifies the upper frequency boundary for the FFT output to be printed to the logfile. The default value is 0.5*np *fmin (Hz).
format	Indicates the output format. A valid value is either norm for normalized magnitude or unorm for un-normalized magnitude. The default value is norm.
np	Specifies the numbers of points used in the FFT analysis, and must be a power of 2. If the specified value is not a power of 2, HSIM will adjust it to the closest higher number of the power of 2. The default value is 1024.
window	Indicates the window type. The values of this parameter are as follows: ▪ rec - Simple rectangular truncation window, default value ▪ bart - Bartlett triangular window ▪ hann - Hanning window ▪ hamm - Hamming window ▪ black - Blackman window ▪ harris - Blackman-Harris window ▪ gauss - Gaussian window ▪ kaiser - Kaiser-Bessel window

Example

```
.fft v(3) fmin=0.5g fmax=2.2g window=gauss
```

Print Windows

Typically, the HSIM simulation output file size is smaller than the output file size of SPICE. Since the circuit size that can be handled by HSIM is much larger than that for SPICE simulators, it is expected that the HSIM output file size may be large for large circuit case.

In addition to limiting the number of printout nodes or elements, specifying the print window(s) can also reduce the output file size. The output data is printed only when the print time is within the specified print time window(s).

Syntax

```
.print window start_time1 <stop_time1 <start_time2
   <stop_time2 …>>>
```

Example

```
.print window 100n 1u 4u 5u
```

The printout is limited to the time windows between 100 ns and 1 us, and between 4 us and 5 us.

11

Conversion Utilities

Provides details on two HSIM utility commands that convert FSDB files. `fsdb2tbl` *converts HSIM generated, fsdb simulation output format to tabular format and FSDB2PWL extracts given signals in the signal_file from FSDB output to the piecewise linear (PWL) voltage source elements.*

In HSIM, four utility commands are available to convert the FSDB and WDF files.

- `fsdb2tbl`: Converts HSIM generated, fsdb simulation output format to tabular format.
- `fsdb2pwl`: Extracts given signals in the signal_file from FSDB output to the piecewise linear (PWL) voltage source elements.
- `wdf2tbl`: Converts HSIM generated, wdf simulation output format to tabular format.
- `wdfb2pwl`: Extracts given signals in the signal_file from WDF output to the piecewise linear (PWL) voltage source elements.

The `fsdb2tbl` Utility

```
fsdb2tbl -i fsdb_file <-hn> <-stdout> <-s sig_file>
   <-o out_file> <-csdf> <-start time1> <-stop time2>
   <-step steptime>
```

or

```
fsdb2tbl -i fsdb_file <-hn> <-stdout> <-s sig_file>
   <-o out_file> <-csdf> <-at time1 <time2 ...>>
```

The `fsdb2tbl` parameters are described in Table 58 on page 358.

Table 58 `fsdb2tbl` Parameters

Parameter	Description
–i	Indicates the FSDB file generated by HSIM.
-s	Specifies a text file to contain the signal names whose values will be converted to the tabular data format.
–o	Specifies a text file to contain the converted tabular data result. The default out_file name is composed of the fsdb filename with the suffix .tbl.
–hn	Prints signal horizontally in the output table file.
–stdout	Prints output table on the screen.
-csdf	Indicates Common Simulation Data Format (CSDF) is the selected tabular output format.
-start	The first conversion begins at the time specified by -start. The unit of time is nanosecond.
-step	After the conversion begins, it is repeated at the increments specified by -step. User may specify multiple steps after the -step flag. The unit of time is nanosecond.
-stop	The conversion ends at the time specified by -stop. The unit of time is nanosecond.
-at	The conversion is performed for the specific time point. The unit of time is nanosecond.

Example 29 signal_file example with the variables noted

```
analog_unit n
analog_to_digital 0.7 2.5
time_step 20 30
precision_digit 6
hex a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa
oct a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa a8 a7 a6 a5 a4 a3 a2 a1
spa
v(a1)
v(xful1.o1)
```

The parameters used in the above example are shown in Table Note: on page 359.

Note:

These parameters only apply if –csdf is not specified.

Table 59 fsdb2tbl Example Parameter Descriptions

Parameter	Description
analog_unit	femtom=milli u =micro n =nano p =pico f = femto
analog_to_digital	Specifies the low and high thresholds to convert analog values to digital representation. In the above example, 0.7 specifies the low threshold, and 2.5 specifies the high threshold.
time_step	Specifies the time steps. The conversion takes place at the specified time step values and their multiples. The selected unit of measure is nanosecond.
precision_digit	Specifies the number of precision digits of the printed values.
hex	Groups the digital signals into hex format. The syntax follows: hex group_name signal1_in_fsdb signal2_in_fsdb signal3_in_fsdb ...

Table 59 fsdb2tbl Example Parameter Descriptions (Continued)

Parameter	Description
oct	Groups the digital signals into octal format. The syntax follows: oct group_name signal1_in_fsdb signal2_in_fsdb singal3_in_fsdb ...
spa	The keyword to insert space between signals in the tabular format. In default, there is no space between the digital signals in tabular format.

Note:

The `fsdb2pwl` Utility

```
fsdb2pwl -i fsdb_file <-s sig_file> <-o out_file> <-start
   time1> <-stop time2>
```

The fsdb2pwl parameters are described in Table on page 363.

Table 60 FSDB2PWL Parameters

Parameter	Description
–i	Indicates the FSDB file generated by HSIM.
-s	Specifies that a text file must contain the signal names whose values will be converted to PWL voltage and current source elements. Additionally, specify the node name and signal type along with the signal name if the signal name is not the preferred node name used in voltage and current source elements.
-start	Specifies the time for the first conversion to start. The unit of time is nanosecond.
-stop	Specifies the time for the conversion to stop. The unit of time is nanosecond.

Table 60 FSDB2PWL Parameters

Parameter	Description
–o	Specifies a text file to contain the converted piecewise linear (PWL) voltage source elements. The default out_file name is composed of the fsdb filename with the suffix .pwl.

Example 30 signal_file syntax example

```
v(a1)
i(b1)
current_i1 vdd1 i
voltage_v1 vdd2 v
```

v(a1), i(b1), current_i1, and voltage_v1 are the existing signal names in the FSDB file. No additional node name and signal type are associated with v(a1) and i(b1). a1 and b1 from the signal name will be used as the node name in both voltage and current source elements. The source element for a1 will be the voltage source because of v(a1) and the source element for b1 will be the current source because of i(b1).

curent_i1 is tagged with the following:

- Node name: vdd1
- Type: i

The source element generated for the current_i1 signal is the current source with the vdd1 node name.

voltage_v1 is tagged with the following:

- Node name: vdd2
- Type: v

The source element generated for voltage_v1 signal is voltage source with vdd2 node name in it.

Example 31 out_file syntax example

```
Va1 a1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Ib1 b1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Icurent_i1 vdd1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Ivoltage_v1 vdd2 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
```

The wdf2tbl Utility

```
wdf2tbl -i wdf_file <-hn> <-stdout> <-s sig_file>
   <-o out_file> <-csdf> <-start time1> <-stop time2> <-step
   steptime>
```

or

```
wdf2tbl -i wdf_file <-hn> <-stdout> <-s sig_file>
   <-o out_file> <-csdf> <-at time1 <time2 ...>>
```

The wdf2tbl parameters are described in Table 62 on page 364.

Table 61 WDF2TBL Parameters

Parameter	Description
–i	Indicates the WDF file generated by HSIM.
-s	Specifies a text file to contain the signal names whose values will be converted to the tabular data format.
–o	Specifies a text file to contain the converted tabular data result. The default out_file name is composed of the wdf filename with the suffix .tbl.
–hn	Prints signal horizontally in the output table file.
–stdout	Prints output table on the screen.
-csdf	Indicates Common Simulation Data Format (CSDF) is the selected tabular output format.
-start	The first conversion begins at the time specified by -start. The unit of time is nanosecond.
-step	After the conversion begins, it is repeated at the increments specified by -step. User may specify multiple steps after the -step flag. The unit of time is nanosecond.
-stop	The conversion ends at the time specified by -stop. The unit of time is nanosecond.
-at	The conversion is performed for the specific time point. The unit of time is nanosecond.

Example 32 signal_file example with variables.

```
analog_unit n
analog_to_digital 0.7 2.5
time_step 20 30
precision_digit 6
hex a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa
oct a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa a8 a7 a6 a5 a4 a3 a2 a1
spa
v(a1)
v(xful1.o1)
```

The parameters used in the above example are shown in Table 62 on page 364.

Note:

These parameters only apply if –csdf is not specified.

Table 62 .WDF2TBL Example Parameter Descriptions

Parameter	Description
analog_unit	Specifies the unit. The available units are as follows:
m	milli
u	micro
n	nano
p	pico
f	femto
analog_to_digital	Specifies the low and high thresholds to convert analog values to digital representation. In the above example, 0.7 specifies the low threshold, and 2.5 specifies the high threshold.
time_step	Specifies the time steps. The conversion takes place at the specified time step values and their multiples. The selected unit of measure is nanosecond.
precision_digit	Specifies the number of precision digits of the printed values.

Table 62 .WDF2TBL Example Parameter Descriptions

Parameter	Description
hex	Groups the digital signals into hex format. The syntax follows: hex group_name signal1_in_wdf signal2_in_wdf signal3_in_wdf ...
oct	Groups the digital signals into octal format. The syntax follows: oct group_name signal1_in_wdf signal2_in_wdf singal3_in_wdf ...
spa	The keyword to insert space between signals in the tabular format. In default, there is no space between the digital signals in tabular format.

The wdf2pwl Utility

```
wdf2pwl -i wdf_file <-s sig_file> <-o out_file> <-start
   time1> <-stop time2>
```

The wdf2pwl parameters are described in Table 63 on page 365.

Table 63 WDF2PWL Parameters

Parameter	Description
–i	Indicates the WDF file generated by HSIM.
-s	Specifies that a text file must contain the signal names whose values will be converted to PWL voltage and current source elements. Additionally, specify the node name and signal type along with the signal name if the signal name is not the preferred node name used in voltage and current source elements.
-start	Specifies the time for the first conversion to start. The unit of time is nanosecond.

Table 63 WDF2PWL Parameters

Parameter	Description
-stop	Specifies the time for the conversion to stop. The unit of time is nanosecond.
–o	Specifies a text file to contain the converted piecewise linear (PWL) voltage source elements. The default out_file name is composed of the wdf filename with the suffix .pwl.

Example 33 signal_file syntax example.

```
v(a1)
i(b1)
current_i1 vdd1 i
voltage_v1 vdd2 v
```

v(a1), i(b1), current_i1, and voltage_v1 are the existing signal names in the WDF file. No additional node name and signal type are associated with v(a1) and i(b1). a1 and b1 from the signal name will be used as the node name in both voltage and current source elements. The source element for a1 will be the voltage source because of v(a1) and the source element for b1 will be the current source because of i(b1).

curent_i1 is tagged with the following:

Node name

vdd1

Type

The source element generated for the current_i1 signal is the current source with the vdd1 node name.

voltage_v1 is tagged with the following:

Node name

vdd2

Type

v

The source element generated for voltage_v1 signal is voltage source with vdd2 node name in it.

Example 34 out_file syntax example.

```
Va1 a1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Ib1 b1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Icurent_i1 vdd1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Ivoltage_v1 vdd2 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
```

The hsencrypt Utility

hsencrypt is an encryption program to encrypt netlist files. Netlist encryption allows proprietary models, parameters, and circuits to be distributed without revealing any intellectual property. Recipients of an encrypted netlist can run simulations using HSIM but cannot print or query the encrypted circuit elements or nodes. HSIM can only read encrypted netlist files that are encrypted using hsencrypt.

To encrypted portion of the netlist file, insert .PROTECT and .UNPROTECT statements around text to be encrypted and then run hsencrypt. hsencrypt

produces an ASCII text file in which all text that follows .PROTECT and precedes .UNPROTECT is encrypted. The syntax follows:

```
% hsencrypt <key> <netlist_file> [<output_file>]
```

`hsencrypt` requires command arguments:

Netlist file name

The netlist file is a SPICE netlist that contains .PROTECT/.UNPROTECT statements and the key is used for encryption or decryption.

Encryption key

xxxyyyzzz is the key used for encryption and decryption as shown in the Example .

Output file name

Optional. The encrypted text will be printed to [<output_file>]. If you do not specify a file name, it will be printed to standard output.

Example

```
% hsencrypt xxxyyyzzz models models.enc
```

Using the xxxyyyzzz key encrypts the portion of the models file specified with .PROTECT/.UNPROTECT and writes the file to models.enc.

Included files and library files specified using .INCLUDE and .LIB statements will not be processed by `hsencrypt` directly as shown in the following example:

```
.PROTECT
.lib prop.mod TT
.UNPROTECT
```

`hsencrypt` will encrypt the .lib prop.mod TT text but not the context of the library file prop.mod. If the prop.mod file content needs to be encrypted, insert .PROTECT/.UNPROTECT statements inside prop.mod and run `hsencrypt` on that file.

The encrypted file can be used in .INCLUDE and .LIB statements just like any other file.

The v2s Utility

v2s is a program to convert the netlist from Verilog to SPICE.

```
v2s <options> filename
```

where filename is one or more Verilog input files that describe the design netlist. The options are shown in Table 64 on page 369.

Table 64 V2S Parameter Descriptions

Option Parameter	Description
<-h[elp]>	Prints a short-form command usage description.
<-bn bus_id>	Specifies the bus notation. Output names are converted with the following bus symbols according to the bus_id value: 0 - [default] Expands buses like: A[0] A[1] A[2] ... 1 - Expands buses like: A<0> A<1> A<3> ... 2 - Expands buses like: A0 A1 A3 ...
<-const0 val>	Specifies voltage level for CONST0.
<-const1 val>	Specifies voltage level for CONST1.
<-pe>	Prints empty subckt definition. By default, an empty module was detected and not outputted.
<-top name>	Specifies the top-level subckt name.
<-map file>	Specifies a name mapping table used when the subckt port mismatches.
<-mapfile file>	Automatically renames the module, node, port, and instance names to ensure syntax agreement as v2s translates a Verilog netlist to SPICE. By default, v2s outputs a mapfile to track the name changes however. If desired, the file name can be changed by specifying -mapfile filename.
<-ks>	Keeps the backslash character in names.
<-v>	Prints a program version.
<-o file>	Specifies the output file name. The default is v2s.spi.
<-s file>	Specifies the SPICE subckt definition files. It is required to output the correct port ordering if instance's port connections are referenced by name.

Table 64 V2S Parameter Descriptions (Continued)

Option Parameter	Description
<-eldo>	The subckt definition file in Eldo format.
<-case 0|1>	Specifies case sensitivity. The default is 0.
<-mapstr>	Refers to characters used as instance or net names in a module that are used as comments in SPICE; i.e., the asterisk, dollar, and ampersand characters (*, $, &), will be replaced using V2S during Verilog netlist translation with an underscore (_) character. If V2S discovers a pre-existing identifier with a similar name, it will modify the new string to create a unique identifier. These changes will be logged in the map file. V2S -mapstr option provides a user option to specify a different map string as shown in the following example: In the Verilog file. wire *logic0*; nand U1(.a(in), .b(*logic0*), .out(out)); Then V2S by default will map *logic0* to _logic0_ in the xu1 in _logic0_ out nand SPICE netlist. Users may also select a different replacement string using the -mapstr option as follows: V2S … -mapstr _abc_ Then the SPICE file will appear as follows: Xu1 in _abc_logic0_abc_ out nand.

Example

```
v2s top.v -s model -o top.spi -top top
```

This example will convert modules defined in top.v to subckt definitions and write output to top.spi with top-level subckt name top.

Note:

Refer to the demo tutorial/v2s.

12

AC Small-Signal Analysis

Provides details of AC Small-signal Analysis (AC), a frequency response analysis that calculates the small-signal response of a circuit to a combination of inputs. This Chapter provides details on AC analysis support for: MOSFET, BJT, JFET, Diode, Resistor, Capacitor, Inductor, Voltage and Current sources (VCVS, VCCS, CCCS, and CCVS), and Mutual Inductor.

AC Small-Signal Analysis (AC) is a frequency domain analysis that calculates the small-signal response of a circuit to a combination of inputs. This is accomplished by creating a linear solution for the circuit at its DC operating point. AC analysis has the following features:

- Nonlinear devices are transformed to linear devices around their bias point value before running an AC analysis. Examples include:
  - Voltage-controlled sources
  - Current-controlled sources
- AC analysis only considers gain and phase responses of a circuit because it is a linear analysis
- AC analysis does not limit voltages or currents.

During AC analysis the simulator seeks a linear solution for the circuit in the frequency domain at the appropriate operating point. Hence, as a first step in conducting AC analysis, the operating point information is determined.

Each voltage or current source can have any or all of the following components:

- AC component
- DC component
- Transient component

For AC analysis, the transient component of the source is ignored and the DC component is used to determine the operating point. If the DC component is unspecified, the default value zero is used.

It is recommended that source magnitude be set to 1.This will ensure that the measured output equals the gain, relative to the input source, at that output.

The following elements are supported in AC analysis:

- MOSFET
- BJT
- JFET
- Diode
- Resistor
- Capacitor
- Inductor
- Voltage and Current sources
  - Voltage-Controlled Voltage Source (VCVS)
  - Voltage-Controlled Current Source (VCCS)
  - Current-Controlled Current Source (CCCS)
  - Current-Controlled Voltage Source (CCVS)
- Mutual inductor

The following sections provide an overview of the AC analysis commands.

AC Analysis Features

Invoking AC analysis

After the operating point is determined, frequency domain analysis can begin. The following statement invokes AC analysis and specifies the required frequency range.

.ac

```
.ac sweep_type nf start stop
```

AC analysis is conducted for the frequency range between start and stop.

sweep_type

One of the following keywords:

dec: for decade increment

oct: for octave increment

lin: for linear increment

poi: for list of points

nf: number of frequencies for lin and poi types; or number of frequencies per decade for dec type, per octave for oct type.

start

Starting frequency for dec, oct, lin.

stop

Final frequency for dec, oct, lin.

Example

```
.ac poi 4 1e8 5e8 8e8 1e9
```

Note:

POI (Points of Interest)

AC analysis is conducted at four different frequencies:

- 100 MHz
- 500 MHz
- 800 MHz
- 1 GHz

AC frequency analysis can be performed when sweeping some external parameter(s) or value(s) of an independent source. An external sweep is specified by augmenting the .ac statement described above with the keyword sweep followed by one of the following specifications.

```
sweep variable_name start stop incr
sweep variable_name sweep_type np start stop
sweep variable_name poi np p_1 p_2 ... p_n
sweep data=data1
```

Here variable_name is one of the following:

- Independent voltage source name
- Independent current source name
- Parameter name
- Keyword: temp for temperature

The first format specifies variation of variable_name in the interval bounded by start and stop on the linear scale, in the increment of incr. The second and the third formats are similar to the format of the internal frequency sweep. In the second format the keyword sweep_type can be one of the following:

- lin
- dec
- oct

Note:

np is number of points for lin and poi types. It can also be the number of points per decade for dec type or per octave for oct type.

When a list of points is specified, poi applies. If several variables are to be changed in AC analysis, the data syntax can be used.

Examples

```
.ac dec 10 1.e6 1.e9 sweep vcc 2 4 0.25
```

The above code example sweeps frequencies between 1 MHz and 1 GHz by placing 10 frequencies per decade for each value of voltage source vcc changing between 2V and 4V in increment of 0.25V.

```
.ac oct 4 1.e6 1.e9 sweep r2val dec 10 1e4 1e5
```

The above code sample sweeps frequencies between 1 MHz and 1 GHz by placing 4 frequencies per octave for each value of parameter r2val changing between 10 K and 100 K uniformly on logarithmic scale with 10 points per decade.

```
.ac lin 10 0.1e9 1.e9 sweep width 1e-7 2e-7 1e-8
```

The above code example sweeps frequencies between 100 MHz and 1 GHz by uniformly placing 10 frequencies between them for each value of parameter width changing between 0.1 µm and 0.2 µm in increment of 10 nm.

```
.ac poi 6 10x 20x 100x 200x 1g 2g sweep temp 0 100 10
```

The above code example sweeps frequencies through a given set of six values for each value of temperature changing between 0° C and 100° C in increment of 10° C.

```
.ac lin 11 1e8 1e9 sweep data=data4ac
.data data4ac
p1   p2   p3
1.0  2.0  3.0
1.1  2.1  3.1
1.2  2.2  3.2
1.3  2.3  3.3
.enddata
```

In AC analysis, four outer sweeps are performed. At the first outer sweep, parameters p1, p2, p3 take the values of 1, 2, and 3, respectively. At the second outer sweep, they take the values of 1.1, 2.1, and 3.1, respectively, and so on.

AC Sources

To perform AC analysis, AC source values must be defined. By default, DC voltage and current sources have AC values equal to zero. Non-zero AC values can be specified on the independent voltage/current sources statement after DC value and before transient function. The syntax is:

```
ac mag <ph>
```

where

- ac is a keyword
- mag is the magnitude of the source
- ph (an optional parameter) is the phase of the source in unit of degrees

When specifying the phase, the following convention is used, all AC signals are proportional to:

```
exp(j (W*t + q ) )
```

where

- j is imaginary unit
- t is time

- Ω is cyclic frequency (2 *π* frequency)
- Phase θ is measured in radians

Example

```
v1 node1 0 3.3 ac 1 45 pulse(3.3 0 1n 0.1n 0.1n 0.4n 2n)
```

Voltage source v1 has the following properties:

- dc value of 3.3V
- `ac` magnitude of 1V
- Phase of 45 degrees
- Followed by the pulse function.

Example

```
i2 node2 node3 1m ac 1
```

Current source i2 is connected between node2 and node3, with DC value of 1mA, `ac` magnitude of 1A, and zero phase.

```
v4 node4 node5 ac 1 30 pwl (0n 0.0 1n 1.0)
```

Voltage source v4 is connected between node4 and node5, with default DC value of zero, `ac` magnitude of 1V and phase of 30 degrees, followed by a piecewise linear function (`pwl`).

AC Analysis Output

The output of the .print ac statement goes to the outfile.ac if –o outfile. This file is specified when invoking HSIM simulation.

Note:

The default file name is hsim.ac.

.alter

If the .alter statement is included in the netlist, the output goes to the output file:

```
.ac.a<n>
```

where

<n> is the alter statement number.

The output file contains names of output variables and their values for each frequency of the sweep in the tabular form. The number of output files corresponds to the number of outer sweeps.

Output variables in AC analysis are complex numbers. Thus, the output syntax in AC analysis provides additional support than that for transient and DC analyses.

.print

```
.print ac var1 <var2 ...>
```

Each output variable such as var1 could be:

- Node voltage
- Voltage between two nodes
- Branch current
- A function of the above

The output variable should contain a modifier such as suffix, to specify the part of the complex number to be printed. Allowable modifiers include:

- r: Real part
- i: Imaginary part
- m: Magnitude
- p: Phase

Voltages between two nodes in addition to modifiers described above can also have the following modifier:

- db: Decibel

Examples

```
.print ac vr(node1) vi(node1) vm(node1) vp(node1)
```

Prints real part of node voltage as well as its imaginary part, magnitude, and phase.

```
.print ac im(v1) ir(r2) ii(r2) im3(m3) ip3(m3)
```

Prints magnitude of current through voltage source v1, real and imaginary part of current through resistor r2, and magnitude and phase of the current through source terminal of MOSFET m3.

In the case of the voltages between two nodes, say nd1 and nd2, the exact meaning of the modifier is defined through the complex phasors of voltages, v(nd1) and v(nd2). The real and imaginary parts are defined as follows:

```
vr(nd1,nd2)=real(v(nd1)-v(nd2)),
vi(nd1,nd2)=imag(v(nd1)-v(nd2)).
```

There are two different definitions for the following modifiers:

- Magnitude
- Phase
- Decibel

AC Analysis Measurement

.measure

Measurement in AC analysis is similar to measurement in transient analysis.

Example

```
.measure ac max vr(node1)
```

The maximum value of `vr(node1)` is determined and the result is stored in the file outfile.ac_mt if –o outfile is specified when invoking HSIM simulation. The default file name is hsim.ac_mt.

13

DC Analysis

Provides details for using DC Analysis to determine the quiescent state or operating point information of the circuit, or a DC sweep of a source value. DC Analysis features described in this chapter include: sweeping a parameter value, sweeping a temperature value, and sweeping any combination of parameter and temperature values.

DC Analysis is used to determine the quiescent state or operating point information of the circuit. Different types of DC analysis are available in the HSIM simulator as described below:

- Sweep a source value
- Sweep the values of two voltage or current sources
- Sweep a parameter value
- Sweep temperature value
- Any combination of the above

Voltage or current sources must be specified as a netlist item if they are included in the .dc statement.

Sweep One or Two Source Value(s)

Either of the following syntax statements can be used.

```
.dc var start stop incr <var2 start2 stop2 incr2>
.dc var start stop incr <sweep var2 type np start2 stop2>
```

Sweeps the voltage or current source var. If the second source var2 is specified, then the first source is swept over its range for each value of the second source.

Table 65 Sweep One or Two Source Value Keyword Descriptions

Keyword	Description
sweep	Indicates that the second sweep has a different type of variation, such as the following examples: ▪ dec ▪ oct ▪ poi
start	Starting value.
stop	Final value.
incr	Linearly increment value.
start2	Starting value of the second source.
stop2	Final value of the second source.
incr2	Linearly increment value of the second source.
sweep-type	One of the following keywords: dec - for decade increment oct - for octave increment lin - for linear increment poi - for list of points np - number of points for lin and poi types; or number of points per decade for dec type, per octave for oct type.

Example

```
.dc i2   0   10m 0.5m
```

The current source `i2` is swept from 0 A to 10 mA in increment of 0.5 mA.

```
.dc vds 0   2   0.1 vgs 0 2 1
```

Voltage source vds is swept from 0Vto 2V in increment of 0.1V at `vds` values of: 0.0V, 1.0V, 2.0V.

```
.dc v1 poi 4   0 0.3 0.5 1
```

The voltage source `v1` is swept at values of: 0.0V, 0.3V, 0.5V, 1.0V.

```
.dc vds 1   3   0.2 sweep r1 dec 3 5k 500k
```

Voltage source vds is swept from 1V to 3V in increment of 0.2V with the resistor `r1` value being swept from 5 Kohm to 500 Kohm by 3 values per decade (dec.).

Sweep Parameter Value

```
.dc param_name start stop incr
```

Sweeps `param_name`. `.dc` has the following parameters:

start

Starting value

stop

Final value

incr

Increment value

Example

```
.dc param3 1   5   1
```

Parameter `param3` is swept from 1 to 5 in increments of 1.

Sweep Simulation Temperature

```
.dc temp start stop incr
```

Sweeps the simulation temperature. `.dc temp` has the following parameters:

start

Starting temperature

stop

Final temperature

incr

Increment value

Example

```
.dc temp poi 4   0   25 50 75
```

HSIM is conducted at the following temperatures: 0 °C, 25 °C, 50 °C, 75 °C.

Multiple Sweeps and Data Sweeps

Double sweeps on parameters or a combination of a source value and a parameter can be performed on HSIM.

```
.dc param1 start1 stop1 incr1 sweep param2 start2 stop2 incr2
```

Sweeps parameter param1 from start1 to stop1 in increments of incr1 for each value of parameter param2 that takes a value from start2 to stop2 in increments of incr2.

Examples

The following example sweeps voltage source v1 from 0V to 5V in increment of 0.5V for each value of parameter `par2` being 1, 2, 3, ... ,10.

```
.dc v1 0   5   0.5 sweep par2 lin 10   1    10
```

The following example sweeps parameter `par1` from 0 to 5 in increment of 1 for each value of parameter `par2` : 1, 2, 3, ... ,10.

```
.dc par1 0 5 1 sweep par2 lin   10   1   10
```

Note:

If one `sweep` variable is a source value and the other is a parameter or temperature, then the source should always be specified as the first `sweep` variable. If this rule is not followed, a Warning message is generated and the two `sweep` variables are interchanged.

If several variables are to be changed for each point of HSIM, the `data` syntax can be used.

In the following example, the `sweep` information is taken from the data1 block.

```
.dc data=data1
```

In the following example, DC analysis is performed for four points. At the first point, parameters p1, p2, p3 take value of 1, 2, and 3, respectively. At the second point, they will take value of 1.1, 2.1, and 3.1, respectively, and so on.

```
.dc data=data1
.data data1
p1   p2   p3
1.0  2.0  3.0
1.1  2.1  3.1
1.2  2.2  3.2
1.3  2.3  3.3
.enddata
```

Output and Control Commands

print

To print output variables/parameters in HSIM use the following statement:

```
.print dc var1 <var2 …>
```

Example

```
.print dc v(node1) v(node2,node3) i(R4)
+ fun5=par('v(node4)*2.345')
```

Unlike output in transient analysis, the output of .print dc statement goes to the file outfile.dc if –o outfile is specified when invoking HSIM simulation. The default file name is hsim.dc. If the .alter statement is included in the netlist, the output goes to files outfile.dc.a<n>, where <n> is the alter number.

The HSIM output contains names of the variables and the associated values in ASCII format. In the case of two multiple sweeps or double sweeps, a separate table is printed for each value of the external loop variable.

Measurement In DC Analysis

.measure

Measurement in HSIM is similar to measurement in transient analysis. The syntax follows:

```
.measure <dc> result func out_var1
```

Example

```
.measure dc vn0 max v(node1)
```

The maximum value of `v(node1)` is determined and the result is stored in the file outfile.dc_mt if –o outfile is specified when invoking HSIM simulation. The default file name is hsim.dc_mt.

DC Interactive Mode Debugging

There are two methods to invoke the interactive mode.

1. Specify the desired DC iteration stop point by including the following HSIMDCSTOPAT on page 66 command in the top-level netlist, as shown:

   ```
   .param HSIMDCSTOPAT=n_iter
   ```

   where the value of *n_iter* is the interruption number.

2. Press [Ctrl] [C] simultaneously at any time during DC iteration.

DC Interactive Mode Commands

The interactive mode commands that are only used for DC include the following:

dcount, quietdc, quiet, pt

```
dccont <diteration>
```

Continues DC iteration. If the optional iteration increment diteration, is followed by dccont, DC will continue for another diteration iterations, then enter the DC interactive mode. If the optional iteration increment is not specified, DC will

continue without entering the interactive mode. It will stop interactively if [Ctrl] [C] are pressed simultaneously.

quitdc

Terminates DC iteration.

quit

Terminates both DC iteration and transient simulation

pt

Obtains the current DC iteration number.

The other transient interactive mode commands are supported in DC mode.

Part: 2 HSIM Advanced Analysis

14

Interactive Mode Debugging

Describes the HSIM interactive circuit debugging environment that probes dynamic circuit data (circuit elements and their parameters, circuit connectivity, and instantaneous signal values) and circuit behavior (voltage and current waveforms simulated up to the present time instance) using interactive mode commands. It also describes the Tool Command Language's (TCL) interactive control features.

Overview of Interactive Mode Debugging

The interactive circuit debugging environment probes the dynamic circuit data and circuit behavior with interactive mode commands. The circuit data includes circuit elements and the parameters, circuit connectivity, and instantaneous signal values. The dynamic circuit behavior includes voltage and current waveforms simulated up to the present time step.

The interactive debugging environment can be used in DC simulation and transient analysis simulation.

DC init. Interactive Mode Debugging

There are two methods to invoke the interactive mode.

Method 1

Specify the desired DC iteration stop point by including the following syntax line at the top-level netlist.

```
.param HSIMDCSTOPAT=n_iter
```

where *n_iter* is the number of the iteration.

Method 2

Press Ctrl-C at any time during DC iteration.

DC Interactive Mode Commands

The interactive mode commands for DC analysis are as follows:

dccont <diteration>

Continues DC iteration as follows:

- If the optional iteration increment `diteration` is followed by dccont, DC will continue for another `diteration` iterations, then enter the DC interactive mode.
- If the optional iteration increment is not specified, DC will continue without entering the interactive mode. It will stop interactively by pressing [Ctrl] [C].

quitdc

Terminates DC iteration.

quit

Terminates both DC iteration and transient simulation

pt

Obtains the current DC iteration number of DC iterations.

Other transient interactive mode commands are supported in DC mode.

Transient Interactive Mode Debugging

The transient interactive mode permits you to perform either of the following operations at any time.

- Change simulation control parameters
- Store simulation database

Refer to inc on page 409 and savesim on page 423.

When activated, the interactive mode prompt HSIM> appears on the screen at the interruption time. It allows the interactive mode commands to be invoked. The commands are described in this chapter.

During each interactive session, Tool Command Language (TCL) uses a built-in command history that lists all the previous commands. The commands

described in Table 66 on page 391, can be used to retrieve previous commands.

Table 66 TCL Command Descriptions

Command	Description
!!	Most recently used command
!#	The #th command on the command history list

Time is measured in nanoseconds in the interactive mode.

There are three methods to invoke the interactive mode.

1. Specify the desired interruption time by including the following syntax in the top-level netlist:

   ```
   .param HSIMSTOPAT=time1
   ```

   The value of `time1` is the interruption time measured in seconds.

   An equivalent method is to include the following syntax in the input netlist:

   ```
   .stop at=time1 <cmf=interactive_command_file>
   ```

   When `interactive_command_file` is specified, the commands included in the `interactive_command_file` are executed when the simulation is switched into the interactive mode.

2. Press [Ctrl] and [C] simultaneously at any time after DC initialization.

3. Specify the conditional interruption in the netlist.

   ```
   .stop v(node_name) val=voltage_value <edge=r|f|c>
   <from=time1> <to=time2> <cmf=interactive_command_file>
   ```

   The syntax is described in Table 67 on page 391.

Table 67 Conditional Interruption Syntax Descriptions

Parameter	Description
node_name	Specifies the voltage node, such as n1 or out2.
voltage_value	The target voltage value of the node at which the simulation stops. The simulation will enter the interactive mode if this is set in the previous batch mode.

Table 67 Conditional Interruption Syntax Descriptions (Continued)

Parameter	Description
edge	The format of edge is similar to that of .measure. ▪ r indicates action at the rising edge when the node voltage reaches the target value. ▪ f indicates action at the falling edge. ▪ c indicates action at either the rising or the falling edge. ▪ If no edge parameter is specified, c is the default.
from, to	This parameter specifies the time window in which to check the condition.

Following is an example of a command that is placed in the input netlist.

```
.stop v(out2) val=1.5 edge=r from=0 to=100n
```

If the `out2` node reaches 1.5V at the rising edge of a time window between 0 ns and 100 ns, HSIM enters the interactive simulation mode.

List of Interactive Mode Commands

Table 68 on page 392 lists the commands used by HSIM.

Note:

Table 68 lists the commands in Key Word alphabetical order. Command descriptions are documented in alphabetical order by command name in the sections that follow.

Table 68 Interactive Mode Debugging Commands

Command	Function
alias	Create Alias
ap	Add Printout Nodes
closelog	Close Interactive Mode Logfile
cont	Continue Simulation
dcon	Print Max. Connectivity Node Name

Table 68 Interactive Mode Debugging Commands (Continued)

Command	Function
dcpath	Find DC Current Path
dn	Dump User-Specified Nodes
dp	Delete Printout Nodes
eid	Get Element Index From Element Name
ename	Get Element Name From Element Index
ev	Get Element Information
exi	Report Elements with Excessive Current
fcc	Report Flash Core Cell Elements
fmeta	Find Meta Condition
fv	Force Constant Voltage to Nodes
help	Get Interactive Mode Command Syntax and Description
HSIMDELVDTO	Flash Core Transistor Usage Syntax
HSIMALLOWEDDV, HSIMSTEADYCURRENT, HSIMSPICE,HSIMTAUMAX	Change Simulation Control Parameters
iap	Add Printout Node by Node Index
idp	Delete Printout Node by Node Index
iev	Get Element Information by Element Index
inc	Get Node Connectivity Information by Node Index
inv	Get Node Voltage and Capacitance by Node Index
lx	List subckt_name Instances
matche	Find Element Names of a Specified Pattern

Table 68 Interactive Mode Debugging Commands (Continued)

Command	Function
matchn	Find Nodes with Logic Value Changes
nc	Get Node Connectivity Information by Node Name
nctr	Print connected Transistors Through R/L/C Network
ni	Print Node Current
nid	Get Node Index from Node Name
nm	Print Nodes Merged to a Specified Node Name
nname	Get Node Name from Node Index
nv	Get Node Voltage and Capacitance by Node Name
op	Dump Operating Point
openlog	Open Interactive Mode Logfile
pt	Get Current Time
quit	Terminate Simulation
rcf	Read Commands from the File
restart	Restart Simulation from Saved Database
rv	Release Constant Voltage From Nodes
savesim	Save Simulation Database
stop <condition>	Conditional Interruption
stop -at	Interrupt Simulation
stop -delete stop -list	List and Delete Interruption Points
vni	Print Voltage Source Node

Table 68 Interactive Mode Debugging Commands (Continued)

Command	Function
trace_thru_on nodeName	Traces the Circuit through Conducting Elements and Identifies High Impedance
tree	Tree Command

Interactive Mode Commands

alias

Creates alias command.

Syntax

```
alias alias_n cmd_name
```

Example

```
HSIM> alias continue cont
```

Description

This command creates the alias name `alias_n` for the interactive command `cmd_name`.

Note:

continue has the same effect as the command cont.

ap

Adds printout nodes.

Syntax

```
ap <-logic> <-vlth value> <-vhth value> node_name
```

Description

If -logic is specified, this command prints the logic waveform. If -vlth/-vhth are specified, that value will be the threshold.

`ap` adds printout nodes in the waveform node list and starts printing the waveform data for the specified nodes from the current time. The `node_name`

may contain the asterisk (*) wildcard character to specify all the nodes matching the specified pattern. The ap command can be used with dp to control the time windows for the waveform data of selected nodes. Refer to dp on page 402 for additional details.

To debug circuit behavior, it may be necessary to print the waveform data of every node in the circuit. This data can produce a large output file which slows down the simulation if the circuit size is large. In most cases it is only necessary to debug the waveform of selected time windows.

The interactive mode commands ap and dp permit restricting the printout data to the specified time window(s). This reduces the waveform file size. This command only works when the FSDB and WDF output formats are used.

Examples

In the following example, the `pt` command obtains the present time information. The example uses `ap` to add nodes `xalu3.xlatch2.mi5` and all nodes matching the pattern `xpll.*` into the waveform list. These nodes will be printed starting from the current time at 125.3 ns. Refer to pt on page 421 for details about `pt`.

If the dp command is not entered, then the printout of these nodes continues until the end of the simulation. If dp is entered, the printout of those specified nodes is terminated at the time `dp` is entered.

```
HSIM> pt
Tnow=125.3ns    tmax=1000ns
HSIM> ap   xalu3.xlatch2.mi5 xpll.*
```

In the following example, from 251.4 ns the nodes matching the pattern `xpll.x2.*` will not be printed.

```
HSIM> pt
tnow=251.4ns   tmax=1000ns
HSIM> dp xpll.x2.*
```

---

closelog

Closes interactive mode logfile.

Syntax

```
closelog
```

Description

The closelog command closes the interactive mode log file that was opened by the openlog command. The log file contains all records of interactive mode

commands and the results reported by the commands entered between `openlog` and closelog. For details about `openlog`, refer to openlog on page 420.

---

cont

Continues simulation.

Syntax

```
cont <dtime>
```

Description

This command continues the transient simulation. If the optional time increment `dtime`, measured in nanoseconds, is followed by the cont command, the simulation stops and again enters the interactive mode at time t+dtime. t is the current time. If t+dtime is less than the simulation stop time, the simulation will continue until t+dtime, then enter the interactive mode at t+dtime. If the optional time increment is not specified, the simulation continues until it is completed, or until the interrupt [Ctrl] and [C] is entered.

Examples

```
HSIM> cont 25.3
```

Continues the simulation and enters the interactive mode 25.3 ns later.

```
HSIM> cont
```

Leaves the interactive mode and continues the simulation.

---

dcon

Prints max. connectivity node name.

Syntax

```
dcon [val]
```

Description

The dcon command finds the circuit node that has the most connections based on the netlist. When `val` is specified, nodes with more connections than `val` are reported. Without `val`, the command reports the most connected node in the netlist. Each node will contain subckt name, node name, number of connection, and a flag Vsrc showing if it is voltage source.

Example

```
HSIM> dcon 20
```

Reports the nodes that have more than 20 connections in the netlist.

dcpath

Finds the DC current path.

Syntax

```
dcpath <-ith current1> <ith2 current2> <-time time1> <-file
   file1> <node1 < node2 ...>>
```

Description

`dcpath` finds the DC current path between specified nodes at present time or a given time later than present time. The options are as described in Table 69 on page 398.

Table 69 Find DC Current Path: `dcpath` Syntax Descriptions

Parameter	Description
-ith current1	Specifies the threshold current of DC path. The default value is 5e-5.
-ith2 current2	Specifies the threshold current for additional remark on the DC path generated by "-ith". When this optional parameter is set, MOSFET devices on the DC path with an Ids current that exceeds the ith2 value are marked with wildcard ("*"). The default ith2 value is 1e-3A.
-time time1	Specifies the time, which is measured in the unit of nanosecond 1.0e-9 second, for HSIM to enter interactive mode and perform DC path search at that time. If -time parameter is not specified, then the DC current path is searched at the current time.
-file file1	Redirects the DC path search result to the specified file. If the file is not specified, then the result is printed on the screen.

Table 69 Find DC Current Path: `dcpath` Syntax Descriptions (Continued)

Parameter	Description
node1 node2 ...	Specifies the terminal node names of DC current path. The DC path search starts from any node specified in this node list and ends when it reaches either another node in the list or a DC voltage source node. If no node is specified, then HSIM reports DC current path(s) between any pair of voltage source nodes.

Example

```
* ------------------------------------------------------------
* DC Path Check
* title         = t1
* ith           = 1e-09
* ith2          = 8e-06
* period        =
* delay         =
* at            = 3e-08
* sort          = 0
* separate_file = 1
* ------------------------------------------------------------
path 1 from vdd! to 0 @ 3e-08s
     i5.m1.m0.lvpd (PMOS: Ids=3.12947e-06)
          source  vdd! 3.6
          drain  o 3.5618
          gate   cm2 0.597389
     i5.m0.m0.lvn (NMOS: Ids=3.02668e-06)
          source  0 0
          drain  o 3.5618
          gate   cm2 0.597389
path 2 from vdd! to 0 @ 3e-08s
     * m1.m0.lvpd (PMOS: Ids=8.05206e-06)
          source  vdd! 3.6
          drain  cm2 0.597389
          gate   cm1 3.08191
     * m3.m0.lvn (NMOS: Ids=8.63069e-06)
          source  0 0
          drain  cm2 0.597389
          gate   cm2 0.597389
* t1: Total number of paths = 2
```

The report is generated based on the threshold current "ith" with a value of 1e-9Amp. Based on this report an "ith2" threshold current can be specified with a value of 8e-6Amp. As a result a wildcard ("*") placed in front of the MOSFET

devices that have an Ids current exceeding this ith2 value. In this example the candidates are "m1.m0.lvpd" and "m3.m0.lvn".

Note:

Do not specify a Z-state within PWLZ when you perform a dcpath check. If a PWLZ source and Z-state are specified within PWLZ, HSIM does not consider PWLZ as a voltage source because there might be a period of time when PWLZ is not driven by voltage source elements. The dcpath check has no indicators of the starting VSRC node locations, and because of this situation, the reported path might not be a legitimate path.

Using dcpath

Keep in mind the following guidelines when using the dcpath interactive command.

1. Two or more node names (pattern) need to be specified to form the pair of nodes for the DC path search.
2. If no node names (pattern) are specified, HSIM automatically forms the vsrc pairs among all of the recognized vsrc nodes for the DC path search.
3. Among the specified node names (pattern), if one (or more than one) is marked with a wildcard ("*") then HSIM treats it as (2) above.
4. The node name pattern can support partial wild cards, such as *abc*, *abc, or abc*.
5. If the interactive dcpath command is used to form "selective" vsrc pairs, we recommended you do the following:
   a. Apply the interactive command "vni" to list all of the vsrc nodes in the design.
   b. Reference this vsrc node list and apply the dcpath command with the above guidelines from items (1) to (4).

For details about `dcpath` power check, refer to Chapter 15, Timing and Power Analysis, DC Path Check on page 448.

---

de

Finds larger circuit capacitor element.

Syntax

```
de [[-ncap [val]]] | [[-pcap [val]]]
```

Description

This command finds the circuit capacitor element with the large capacitance as described in the following bullets.

-ncap [val]

If val is specified, -ncap [val] finds capacitors with capacitance smaller than val. If val is not specified, it finds the minimum negative capacitance. val is in farads.

-pcap [val]

If val is specified, -pcap [val] finds capacitors with capacitance greater than val. If val is not specified, it finds the maximum positive capacitance. val is in farads.

Example

```
HSIM > de -pcap
Maximum CAP=1pF at element c5, time_step=1e+006ps
HSIM > de -pcap 1e-14
     CAP elem c4 cap=1e-013
     CAP elem c5 cap=1e-012

HSIM > de -pcap
Maximum CAP=1pF at element c5, time_step=1e+006ps
```

---

dn

Dumps user-specified nodes.

Syntax

```
dn [[-dv [val]] | [-v [val]] | [-ncap [val]]] [-print [file]]
```

Description

This command finds the circuit node that has the large voltage change, voltage, or node capacitance. The options are as described in Table 69 on page 398

-dv [val]

Finds the node with change voltage greater than val (if val is specified) or the maximum delta voltage (if no val is specified). The unit of val is volt.

-v [val]

Finds the node with voltage greater than val (if val is specified) or the maximum voltage (if no val is specified). The unit of val is volt.

-ncap [val]

Finds the node with capacitance greater than val (if val is specified) or the maximum capacitance (if no val is specified). The unit of val is farad.

-print [file]

Adds the matching nodes to the printout node list and create an nWave .rc file for signal restore. If no file is specified, the .rc file is named hsim.rc (if no output prefix is specified when running HSIM) or {prefix}.rc (if -o prefix is specified when running HSIM).

Examples

```
HSIM> dn -ncap 1e-15
```

Reports nodes that have capacitance value greater than 1e-15f.

```
HSIM> dn -dv
```

Reports the node that has maximum voltage change.

```
HSIM> dn -v 0.5 -print
```

Reports nodes with voltage greater than 0.5v. Also add those nodes into a printout list and generate a .rc file that can be debugged using nWave.

An example of the generate .rc is shown below:

```
; rc file created by command dn
;
addGroup "Gxxx.rc"
addSignal -w analog -c ID_ORANGE5 /xbuff/v(n4)
addSignal -w analog -c ID_ORANGE5 /v(n2)
addSignal -w analog -c ID_ORANGE5 /v(n4)
addSignal -w analog -c ID_ORANGE5 /v(n5)
addSignal -w analog -c ID_ORANGE5 /xbuff/v(n3)
```

---

dp

Deletes printout nodes.

Syntax

```
dp node_name1 <node_name2 ...>
```

Description

`dp` deletes printout node(s) from the waveform node list and stops printing the waveform data for the specified node(s) from the current time. The `node_name` may contain the asterisk (*) wildcard character to specify all the node(s) that

match the entered pattern. Together with the ap command, the time window(s) for the waveform data of selected nodes can be controlled. Refer to ap on page 395 for details about ap.

To debug the circuit behavior, it may be necessary to print the waveform data for every node in the circuit. However, this generates a large output file which can slow down HSIM simulation when the circuit size is very large. In most cases, only the selected time window(s) requires debugging. The interactive mode commands ap and dp restrict the printout data to the specified time window(s) which reduces the waveform file size. Refer to the example in ap on page 395.

---

eid

Gets element index from element name.

Syntax

```
eid elem_name
```

Description

`eid` obtains an index number to a specific element within the HSIM simulation database. To access the element information through the query commands such as ev and iev, either the element name or the corresponding element index number is required. If the element name is very long, it may be convenient to use the index number instead. The eid command accesses the element index number. In the following example, the index number is 168.

Refer to the following for additional information on ev and iev: ev on page 404 and iev on page 408.

Example

```
HSIM> eid xalu3.xlatch2.mi5
```

---

ename

Gets element name from element index.

Syntax

```
ename elem_index
```

Description

`ename` reports the element name corresponding to the specified index number.

Example

```
HSIM> ename 168
xalu3.xlatch2.mi5
```

---

ev

Gets element information.

For models that include drain/source resistors (RDSMOD=1 or NRD/NRS instance parameters) the reported source/drain voltage is representative of the internal drain/source voltage, which may differ from the external source/drain node voltage.

Syntax

```
ev element_name
```

Description

`ev` prints the detailed element information for the given `element_name` at the specific time it is activated including:

- Element terminals
- Element type
- Terminal voltages
- Element currents
- Element conductance
- Element capacitances

The element_name can contain the asterisk (*) wildcard character.

Example

```
HSIM> ev xalu3.xlatch2.mi5
xalu3.xlatch2.mi5 (168) - DRN: xalu3.xlatch2.n24 (755), GATE:
xalu3.xlatch2.q (888), SRC: Gnd (0), BULK: Gnd (0)
NMOS VG=2.23, VD=0.87, VS=0, VB=0
Ids=2.352E-5, gds=1.13E-4, gm=2.115E-4
Voltage-dependent diode cap: cbd=2.17E-15, cbs=1.23E-15
Voltage-dependent gate cap: cgs=3.17E-15, cgd=1.21E-16,
cgb=2.22E-15
```

---

exi

Reports elements with excessive current.

Syntax

```
exi   <-ith current_val>   elem_name1 <elem_name2 ... >
```

Description

`exi` identifies and reports the element(s) from the specified element name(s) that has excessive current greater than the specified threshold current value. The unit of the threshold current value is ampere. The default value is 5e-5. The element name may contain asterisk (*) wildcard character to represent all elements matching the pattern.

Example

```
HSIM> exi   -ith 1e-5   x1.*
```

Each element in instance x1 is reported if the element current value exceeds 10 uA at the present time.

---

fcc

Reports flash core cell elements.

Syntax

```
fcc <-dvth value> <-file filename> <-save filename>
   elem_name1 <elem_name2... >
```

Description

fcc identifies and reports the flash core cell elements from specified element names having a threshold voltage shift with either of the following parameters:

- Greater than or equal to the specified value if it is positive or zero
- Smaller than or equal to the specified value if it is negative or zero

fcc The element name may contain an asterisk (*) wildcard character to represent all elements matching the pattern.

The fcc command supports the following options: BSIM3, BSIM4, EKV, MOS9, MOS11, and MOS Level-1.

Examples

In the example below, each element in instance x1 is reported if the element threshold voltage shift is greater than or equal to 0.2 V at the present time:

```
HSIM> fcc -dvth 0.2 x1.*
```

The example below shows a printout:

```
m1, Vth=-0.870335, dVth=-3.37534, delvto=0
m3, Vth=-0.500000, dVth=-2.12673, delvto=0.2
```

The -file option saves the printout to a file instead of showing it on a monitor. This is convenient for test cases where there are many flash core cells and dumping to a file would be much faster.

The -save option saves a printout in a file that can be used to restore a simulation with flash core cells. The syntax is as follows:

```
.hsimparam HSIMDDVTO=dvth_value inst= elem_name
```

Using the syntax in the previous syntax, the following is saved.

```
.hsimparam HSIMDDVTO=-3.37534 inst=m1
.hsimparam HSIMDDVTO=-2.12673 inst=m3
```

---

fmeta

Finds meta condition.

Syntax

```
fmeta
```

Description

`fmeta` finds the meta-stable conditions in the circuit. Meta-stable conditions can induce large currents through power supply, it can also cause oscillation that slows down the simulation. If meta-stable conditions are found after DC initialization, either node logic 0 or 1 should be forced and then released using the fv and rv command. A logic 1 or 0 ic condition can also be applied on one of the nodes and then, rerun the simulation.

Example

```
HSIM > fmeta
Find meta stable nodes
potential meta stable node nod1(1192) nod2 (1193)
element mn2(6698)
```

---

fv

Forces constant voltage to nodes.

Syntax

```
fv node1 <node2...> val1
```

Description

`fv` forces the specified nodes to stay at the specified constant voltage. The node voltage stays at the same value from the current time until either the end of simulation or when the constant node voltage status is released by the command rv. In the interactive mode, the `fv` command does not work on voltage source nodes of or vector files. Refer to rv on page 422.

Example

```
HSIM> fv   pump 6.5
```

The voltage at node pump stays at 6.5V from the current time on.

---

help

Gets interactive mode command syntax and description.

Syntax

```
help <command_name>
```

Description

`help` displays a brief description of the specified command. If a command is not specified, all interactive mode commands are displayed. To get the command syntax for a specific command, enter the following:

```
<cmd_name> -help
```

Example

```
HSIM> help ename
ename - get element name by element id.
```

---

iap

Adds printout node by node index.

Syntax

```
iap <-logic> <-vlth value> <-vhth value> node_index
```

Description

If -logic is specified, this command prints the logic waveform. If -vlth/-vhth are specified, that value will be the threshold.

`iap` adds printout nodes that are specified by the node index to the waveform data list. This command works the same as ap except the node index number follows the iap command while the node name follows the ap command. This

command works only in FSDB and WDF output formats. For details about ap, refer to ap on page 395.

Example

```
HSIM> iap 168
```

---

idp

Deletes printout node by node index.

Syntax

```
idp node_index1 <node_index2 ...>
```

Description

`idp` deletes printout node, specified by the node index, from the waveform data list. This command functions as does dp, except the node index number follows the idp command where the node name follows the dp command. This command operates only in FSDB and WDF output formats. For details about dp, refer to dp on page 402.

Example

```
HSIM> idp 168
```

---

iev

Gets element information by element index.

For models that include drain/source resistors (RDSMOD=1 or NRD/NRS instance parameters) the reported source/drain voltage is representative of the internal drain/source voltage, which may differ from the external source/drain node voltage.

Syntax

```
iev elem_index
```

Description

`iev` prints the detailed element information for the specified element index, including:

- Element terminals
- Element type
- Terminal voltage

- Element current
- Element conductance
- Element capacitance

Example

```
HSIM> iev 168
xalu3.xlatch2.mi5 (168) - DRN: xalu3.xlatch2.n24 (755), GATE:
xalu3.xlatch2.q (888), SRC: Gnd (0), BULK: Gnd (0)
NMOS VG=2.23, VD=0.87, VS=0, VB=0
Ids=2.352E-5, gds=1.13E-4, gm=2.115E-4
V-dependent diode cap: cbd=2.17E-15, cbs=1.23E-15
V-dependent gate cap: cgs=3.17E-15, cgd=1.21E-16, cgb=2.22E-15
```

inc

Gets node connectivity information by node index.

Syntax

```
inc node_index <-level num> <-inst name> <-etype
   (c|d|f|h|i|l|m|q|r|v)> <-iin val2> <-iout val3> <-term
   i<,j ...> >   <-on val4> <-total>
```

Description

`inc` prints the detailed node connectivity information for the given node index. If no option is specified, then all the elements connected to the node are printed. Each option selectively prints a subset of elements described in Table 70 on page 409.

Table 70 inc Command Element Descriptions

Parameter	Description
-level num	Only prints the elements that are located in the hierarchies between the current level where the specified node index is located, to num-1 levels below the current level.
-inst name	Only prints the elements that are located within the given instance name.

Table 70 inc Command Element Descriptions (Continued)

Parameter	Description
-etype (...)	Only prints the elements that match the specified type(s). More than one character can be specified. Each character defines the element type. For example: ▪ c specifies the capacitor ▪ r specifies the resistor ▪ m specifies the MOSFET Space is not allowed between the element characters when more than one element is specified.
-iin val2	Only prints the element which, at the current time, has conducting current higher than the specified value val2 flowing into the specified node index.
-iout val3	Only prints the element which, at the current time, has conducting current higher than the specified value val3 flowing out of the specified node index.
-term i,j ...	Only prints the elements connected to the specified terminal number(s) of the specified node index. Multiple terminal numbers, separated by a comma (,), can be specified. Space is not allowed between the comma (,) and the terminal number.
-on val4	Only prints the conducting element which, at the current time, has conducting current higher than the specified current value val4, and is connected to the specified node index.
-total	The conducting current also includes the capacitor current.

Examples

A description follows each example.

```
HSIM> inc 888
```

Prints all the elements connected to the `888`th node.

```
HSIM> inc 888 -level 2
```

Prints elements connected to the `888`th node and are also located in either the same hierarchical level as the `888`th node or one level below.

```
HSIM> inc 888 -inst xalu3.xlatch2
```

Prints the elements connected to the `888`th node and are also located within the subcircuit instance `xalu3.xlatch2`.

```
HSIM> inc 888 -etype rm
```

Prints resistors and MOSFET transistors connected to the `888`th node.

```
HSIM> inc 888 -iin 0.001
```

Prints elements which conduct more than 1 mA of current flowing into the `888`th node.

```
HSIM> inc 888 -iout 0.00012
```

Prints elements which conduct more than 120 uA of current flowing out of the `888`th node.

```
HSIM> inc 888 -term 1,3 -etype m
```

Prints the MOSFET transistors which have either drain or source terminal connecting to the `888`th node.

```
HSIM> inc 888 -on 1E-4
```

Prints elements which conduct more than 100 uA of current and are also connected to the 888th node.

---

inv

Gets node voltage and capacitance by node index.

Syntax

```
inv node_index
```

Description

`inv` prints node voltage, slope of node voltage waveform, and node capacitance for the given node index. Each value is evaluated at the most recent time, which is when the node voltage is last updated. The data are then printed.

Example

If the current time is 100 ns and the printed voltage is 2.5V with the printed time 95 ns, it implies the node voltage remains unchanged at 2.5V between 95 ns

and 100 ns. The capacitance report contains both constant and voltage-dependent capacitances. The constant capacitance is the sum of constant capacitances connected to the node and the voltage-dependent capacitance is the sum of all MOSFET capacitances connected to the node.

```
HSIM> inv 888
V=5, dV/dt=3528.7, time=5.278E-8
Constant capacitance=2.38fF, V-dependent capacitance=5.73fF
Total capacitance=8.11fF
```

---

lx

Lists subckt_name instance.

Syntax

`lx subckt_name`

Description

`lx` lists all instances of specified subckt_name.

Example

```
HSIM> lx add8
x1.x1
x1.x3
x1.x2
x1.x10
x1.x4
x1.x5
x1.x6
x1.x7
x1.x8
x1.x9
```

---

matche

Finds element names of a specified pattern.

Syntax

`matche pattern`

Description

`matche` finds all element names that match the specified pattern.

matchn

Finds nodes with logic value changes.

Syntax

```
matchn [val]
```

Description

In order to trace the logic signal value changes, matchn has been enhanced with the following new options. The logic values for a signal are one of the following: 0, 1, X, and Z.

Those options require that the matched nodes have .lprint specified for the netlist so the waveform histories of those nodes are kept. The matched nodes without waveform history will be ignored. Currently the only way to specify the matching condition for those options is using -ntl.

- -change val: Define the change of a node. val can be one of rise, fall, or all.
- rise: A node changes from 0 to 1.
- fall: A node changes from 1 to 0.
- all: A node has any changes in values (0,1,Z,X)
- -print [file]: Add the matching nodes to the printout node list and create an nWave .rc file for signal restore. If no file is specified, the .rc file is named hsim.rc (if no output prefix is specified when running HSIM) or {prefix}.rc (if -o prefix is specified when running HSIM).

By default, waveform histories for those nodes will be analyzed from time 0 to the current simulation time. The waveform range can also be controlled by the following two options:

- -start time: Specify the start time of the waveform range. The default is 0.
- -stop time: Specify the stop time of the waveform range. The default is the current time.

Example

```
HSIM> matchn -ntl xp11.x2.* -change rise -print xxx.rc
```

Reports the nodes that have logic value changes from 0 to 1 in the current simulation run. Create an nWave .rc file xxx.rc.

Note:

This is assuming .lprint xp11.x2.* is required to be in the netlist.

nc

Gets node connectivity information by node name.

Syntax

```
nc node_name <-level num> <-inst name> <-etype
   (c|d|f|h|i|l|m|q|r|v)> <-iin val2> <-iout val3> <-term
   i<,j ...> > <-on val4> <-total>
```

Description

`nc` prints the detailed node connectivity information for the given node name. If there is no option specified, then all the elements connected to the node are printed. Each option selectively prints a subset of elements.

Arguments

Table 71 on page 414 shows the parameters.

Table 71 Node Name Connectivity Information Command Parameters

Parameter	Description
-level num	Only prints elements that are located in the hierarchies between the current level where the specified node is located, to num-1 levels below the current level.
-inst name	Only prints elements that are located within the given instance name.
-etype ...	Only prints elements that match the specified type(s). More than one character can be specified such that each character defines the element type. For example: ▪ c specifies the capacitor ▪ m specifies the MOSFET ▪ Space is not allowed between element characters when more than one element is specified.
-iin val2	Only prints the element which, at the current time, is conducting current higher than the specified value val2. The current is flowing into the specified node name.
-iout val3	Only prints the element which, at the current time, is conducting current higher than the specified value val3. The current is flowing out of the specified node name.

Table 71 Node Name Connectivity Information Command Parameters

Parameter	Description
-term i,j ...	Only prints the elements that connect to the specified terminal number(s) of the specified node name. Multiple terminal numbers, separated by a comma (,), can be specified. No space is allowed between the comma (,) and the terminal number.
-on val4	Only prints the element that is connected to the specified node name and, at the current time, is conducting current higher than the specified current value val4.
-total	The conducting current also includes the capacitor current.

Examples

A description follows each example of code.

```
HSIM> nc xalu3.xlatch2.q
```

Prints all the elements connected to node `xalu3.xlatch2.q`.

```
HSIM> nc xalu3.xlatch2.q -level 2
```

Prints elements connected to the node `xalu3.xlatch2.q` and are also located in either the same hierarchical level as `xalu3.xlatch2.q` or one level below.

```
HSIM> nc xalu3.xlatch2.q -inst xalu3.xlatch2
```

Prints the elements connected to node `xalu3.xlatch2.q` and are also located within the subcircuit instance `xalu3.xlatch2`.

```
HSIM> nc xalu3.xlatch2.q -etype rm
```

Prints resistors and MOSFET transistors connected to the node `xalu3.xlatch2.q`.

```
HSIM> nc xalu3.xlatch2.q -iin 0.001
```

Prints elements which conduct more than 1mA of current flowing into the node `xalu3.xlatch2.q`.

```
HSIM> nc xalu3.xlatch2.q -iout 0.00012
```

Prints elements which conduct more than 120uA of current flowing out of the node `xalu3.xlatch2.q`.

```
HSIM> nc xalu3.xlatch2.q -term 1,3 -etype m
```

Prints the MOSFET transistors which have either drain or source terminal connecting to the node `xalu3.xlatch2.q`.

```
HSIM> nc xalu3.xlatch2.q -on 1E-4
```

Prints elements which conduct more than 100 uA of current and are also connected to the node xalu3.xlatch2.q.

---

nctr

Prints connected transistors through R/L/C network.

For a given node name, nctr uses it as starting point to traverse R/L/C network to find all the connected transistor device(s).

Syntax

```
nctr <node_name> <-term (terminal index)>
   <-on (conducting threshold current)>
   <-iin (incoming threshold current)>
   <-iout (outgoing threshold current)>
   <-etype (device type)> <-total>
```

Description

This command prints connected transistors through R/L/C network. For a given node name, nctr uses it as starting point to traverse R/L/C network to find all the connected transistor device(s).

Note:

nctr usage is the same as for the nc command.

Examples

Example 35

```
HSIM > nctr out1
Prints transistor connection through R/L/C network:

Node: out1
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS - DRN:out1 (5), SRC:out2 (6), GATE:in1 (3),
BULK:gnd (2)
xmos1.mp1 (6) PMOS - DRN:out1 (5), SRC:vdd (9), GATE:in1 (3),
BULK:vdd (9)

Node: in2
mn3 (0) NMOS - DRN:in2 (4), SRC:dummys (1), GATE:dummyin (0),
BULK:gnd (2)
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos2.mn1 (7) NMOS - DRN:out3 (7), SRC:out4 (8), GATE:in2 (4),
BULK:gnd (2)
xmos2.mn2 (8) NMOS - DRN:out4 (8), SRC:gnd (2), GATE:in2 (4),
BULK:gnd (2)
xmos2.mp1 (9) PMOS - DRN:out3 (7), SRC:vdd (9), GATE:in2 (4),
BULK:vdd (9)
```

Example 36

```
HSIM > nctr out1 -term 1
Prints transistor connection through R/L/C network:

Node: out1
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS - DRN:out1 (5), SRC:out2 (6), GATE:in1 (3),
BULK:gnd (2)
xmos1.mp1 (6) PMOS - DRN:out1 (5), SRC:vdd (9), GATE:in1 (3),
BULK:vdd (9)

Node: in2
mn3 (0) NMOS - DRN:in2 (4), SRC:dummys (1), GATE:dummyin (0),
BULK:gnd (2)
```

Example 37

```
HSIM > nctr out1 -term 1 -on 1e-10
Prints transistor connection through R/L/C network:

Node: out1
r1 (1) 8.50951e-009 A - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS -8.79645e-009 A - DRN:out1 (5), SRC:out2 (6),
GATE:in1 (3), BULK:gnd (2)
xmos1.mp1 (6) PMOS 2.81614e-010 A - DRN:out1 (5), SRC:vdd (9),
GATE:in1 (3), BULK:vdd (9)

Node: in2
```

ni

Prints node current.

Syntax

`ni node_name`

Description

`ni` prints the node current for the given node name. For nodes not connected to voltage sources, the node current is the sum of all the branch currents flowing away from the node. In this definition the branch currents entering the node are neglected. In the case of nodes connected to voltage sources, the node current is defined as the sum of all the branch currents.

Example

```
HSIM > ni xiwire11.n0_4
Node xiwire11.n0_4 (18):

Node current=6.95149e-006.
```

nid

Gets node index from node name.

Syntax

`nid node_name`

Description

`nid` prints the node index corresponding to the given node name.

Example

```
HSIM> nid xalu3.xlatch2.q
888
```

---

nm

Prints nodes merged to a specified node.

Syntax

`nm node`

Description

`nm` prints nodes merged to the specified node.

---

nname

Gets node name from node index.

Syntax

`nname node_index`

Description

`nname` prints the node name corresponding to the given node index.

Example

```
HSIM> nname 888
xalu3.xlatch2.q
```

---

nv

Gets node voltage and capacitance by node name.

Syntax

`nv node_name`

Description

`nv` prints the node voltage, slope of node voltage waveform, and node capacitance for the given node name. Each value is evaluated at the most recent time when the node voltage is last updated. The data are printed.

Example

If the current time is 100 ns and the printed voltage is 2.5V with printed time 95 ns, this implies that the node voltage stays unchanged at 2.5V between 95 ns and 100 ns. The capacitance report contains both constant and voltage dependent capacitances. The constant capacitance is the sum of constant capacitances connected to the node; and the voltage dependent capacitance is the sum of all MOSFET capacitances connected to the node. The `node_name` can contain the asterisk (*) wildcard character.

```
HSIM> nv xalu3.xlatch2.q
V=5, dV/dt=3528.7, time=5.278E-8
Constant capacitance=2.38fF, V-dependent capacitance=5.73fF
Total capacitance=8.11fF
```

---

op

Dumps operating point.

Syntax

```
op file_name <node_pattern>
```

Description

`op` prints the operating point state, such as the node voltage value, at current time into the specified file `file_name`. The optional `node_pattern` limits the printout nodes to those nodes matching the pattern, which may contain asterisk (*) wildcard character. The node voltage in the printout file has the following format:

```
.ic v(node_name)=node_voltage_value
```

The file can be included as an initial condition file in a later simulation which starts from the same operating condition listed in this file.

Example

```
HSIM> op file1
```

---

openlog

Opens interactive mode logfile.

Syntax

```
openlog logfile_name <-a>
```

Description

`openlog` opens the interactive mode `logfile_name` which contains the record of interactive mode commands and the results reported by these commands until the logfile is closed by the closelog command. If -a is specified, the contents will be appended to any preexisting file of the same name. For detailed information about closelog, refer to closelog on page 396.

Example

```
openlog file2
```

---

pt

Gets current time.

Syntax

```
pt
```

Description

pt prints the current time and the final simulation time.

Example

```
HSIM> pt
tnow=25.6ns, tmax=100ns
```

---

quit

Terminates simulation.

Syntax

```
quit
```

Description

`quit` terminates the simulation.

Example

```
HSIM> quit
```

---

rcf

Reads commands from the file.

Syntax

```
rcf cmd_file
```

Description

`rcf` reads and executes each interactive mode command listed in the specified `cmd_file` file.

Example

```
HSIM> rcf command_file2
```

---

restart

Restarts simulation from saved database.

Syntax

```
restart time1 <file_name>
```

Description

`restart` restarts simulation at the earlier time specified by `time1`. The time is measured in nanoseconds. If the `file_name`is specified, it indicates the saved file from either the interactive savesim command or from the .store command in the netlist.

When restarting simulation inside the interactive mode at an earlier time, the output file will be created with .rs# suffix with # representing the number of restarts. For detailed information about saveism, refer to savesim on page 423 For .store, refer to Chapter 10, Simulation Output, .STORE/.RESTORE Statement on page 346.

Example

```
HSIM> restart 35   save_file
```

---

rv

Releases constant voltage from nodes.

Syntax

```
rv node1 <node2 ...>
```

Description

`rv` releases the node voltages from the value(s) fixed by the fv command. The node voltages are then determined by the regular simulation result. Refer to fv on page 406 for details about fv.

Example

```
HSIM> rv   n2   n5
```

---

savesim

Saves simulation database.

Syntax

```
savesim file_name
```

Description

`savesim` saves the simulation database at the current time to the specified file. The `file_name` must be specified when this command is first used. If `file_name` is not specified, the previously entered filename will be used.

Example

```
HSIM> savesim save_file
```

For related information, refer to Chapter 10, Simulation Output, .STORE/ .RESTORE Statement on page 346.

---

stop <conditional>

Interrupts simulation under specified condition.

Syntax

```
stop node_name -val voltage_value <-edge r|f|c> <-from time1>
   <-to time2> <-cmf interactive_command_file>
stop node_name val=voltage_value <-edge=[r|f|c>
   <from=time1> <to=time2> <cmf=interactive_command_file>
```

Description

This command pauses the simulation at the specified condition. `node_name` is the voltage node such as n1 or out2, and `voltage_value` is the target voltage value of the node at which stop action occurs. `-edge` is in a format similar to the one in .measure.

Arguments

r

Indicates stop at the rising edge when the node voltage reaches the target value.

f

Indicates stop at the falling edge.

c

Indicates stop at either the rising edge or the falling edge. If the edge parameter is not specified, the default is c.

-from

Defines the start of the time window in which to check the condition.

-to

Defines the end of the time window in which to check the condition.

Example

```
HSIM> stop n1 -val 1.5 –edge r -from 0 -to 100
```

The simulation will be paused if node n1 reaches 1.5V at a rising edge within the time window of 0 ns to 100 ns.

---

stop -at

Interrupts simulation.

Syntax

```
stop –at time1 <-cmf interactive_command_file>
stop at=time1 <cmf=interactive_command_file>
```

Description

If the `-cmf` option is specified, this command pauses the simulation at the specified time `time1` and executes the commands included in the `interactive_command_file`. Time is measured in nanoseconds.

Example

```
HSIM> stop –at 1050
```

---

stop -list and stop -delete

Lists and deletes interruption points.

Syntax

```
stop -list
stop -delete st_point1
```

Description

The `stop -list` statement lists all the existing stop points and the associated numbers. The `stop -delete` will remove the stop point.

Example

```
HSIM> stop -list
HSIM> stop –delete
```

---

vni

Prints voltage source node.

Syntax

```
vni [-threshold value]
```

Description

`vni` prints the voltage source node with its current greater than or equal to the threshold value only.

---

trace_thru_on

Traces the circuit through conducting elements and identifies high impedance.

Syntax

```
trace_thru_on nodeName
```

Description

trace_thru_on `nodeName` determines whether a node is in the high-z mode then debugs leakage currents through OFF-elements in the node. It traces from the specified node through conducting MOSFETs and resistors until reaching either vsrc or OFF-elements.

Example

For the xib.i609_d node, trace_thru_on xib.i609_d produces the following printout.

```
at time (150000.0), from node xib.i609_d (id=173564) trace thru
ON elems
     thru xia.xi5.rpi608 to node xia.sa_la<33>
     thru xib.rpi619 to node xib.i40_d
     thru xib.rpi603 to node xib.i605_d
     thru xib.rpi608 to node xib.i609_d
     thru xib.rpi613 to node xib.i613_plus
     thru xib.rpi618 to node xib.i617_d
     thru xib.rpi606 to node xib.i606_plus
     thru xib.rpi624 to node xib.i624_minus
     thru xib.mi627 to node xib.i627_d
     thru xib.rpi630 to node xib.i705_d
     thru xib.mi719 to node xib.i17_minus
     thru xib.rpi17 to node xib.i633_d
     thru xib.mi633 to node vbl
```

tree

Prints hierarchical tree structure for specified subckt instances.

Syntax

```
tree [<subinst_pattern> ...]<-level num> <-a> <-def>
   <-count> <file filename> <subinst_pattern> -subckt
   instances
```

Arguments

-level num

Prints the subckt instances located in the hierarchies between the current level where the specified instance is located, to num-1 levels below the current level.

-a

Prints the full hierarchical instance name.

-def

Prints the subckt name of the instance.

-count

Prints the number of elements inside the specified instance.

-file *filename*

Prints the results in a file named *filename*. Without this option, the results are displayed on the screen.

Examples

The following example prints the complete hierarchical tree.

```
> xtop
> +---x1
> +---xa
> +---xb
> +---x2
> +---x1
> +---xa
> +---xb
> +---xc
```

The following example prints hierarchical trees of xtop.x1 & xtop.x2 down to next hierarchy tree xtop.x* -level 2.

```
> x1
> +---xa
> +---xb
> x2
> +---x1
```

In the following example, tree -a -def prints the complete tree with full hierarchical instance name and subckt definition name.

```
> xtop (cir)
> xtop.x1 (cir1)
> xtop.x1.xa (and2)
> xtop.x1.xb (or2)
> xtop.x2 (cir2)
> xtop.x2.x1 (cir3)
> xtop.x2.x1.xa (nand2)
> xtop.x2.x1.xb (nand2)
> xtop.x2.x1.xc (nand2)
```

15

Timing and Power Analysis

Describes HSIM timing checking (setup, hold, pulse width, edge, checking windows, bisection optimization) and power analyses (DC path, excessive current, excessive rise/fall, high impedance node) tools that work within the netlist file or in a separate file included in the netlist file.

HSIM provides a set of commands that perform timing and power analyses. Each command must be specified either within the netlist file or in a separate file included in the netlist file using the .include option. (See Chapter 7, Input Netlist, .include on page 293 for information about the .include statement.)

Checks use the statements shown below:

- .tcheck: Timing check command
- .pcheck: Power check command

If there is either a timing or power check error, it is reported in one of the following files:

- hsim.chk
- out_file.chk: If -o out_file is specified when invoking HSIM.

If separate_file is set in the command, the report file name will include the check name and the title name. For example:

```
hsim.edge_t1.chk
```

Timing Checking

This section provides descriptions of the HSIM timing checks.

Setup Time Check

Checks whether the reference node transition time minus signal node transition time is less than the setup time.

Syntax

```
.tcheck title_name setup sig_name edge_type ref_name
   ref_edge_type setup_time <subckt=sub_name>
   <window=window_limit> <vlth=logic_low_voltage>
   <vhth=logic_high_voltage>
   <refvlth=ref_logic_low_voltage>
   <refvhth=ref_logic_high_voltage> <separate_file=0|1>
```

Parameters

Table 72 Setup Time Check Parameters

Parameter	Description
title_name	Defines the title name of this setup time check. Setup time errors during this timing check are listed in the timing/power check error file. The name of the setup time error starts with this title name.
sig_name	Defines signal node name(s) that can be the node name of a single node or a node name containing an asterisk (*) wildcard character that represents a group of node names. The node name with a wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax all characters after wildcards are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then sig_name should be the full path name.
ref_name	Defines the reference node name. ref_name is the node name of a single node; it must not contain an asterisk (*) wildcard character. When using the subckt parameter, the ref_name is the node name inside the subcircuit which must not be the full path name. If the subckt parameter is not used, then the ref_name must represent the full path name.
edge_type	Defines the permissible state transition type for signal node with r, f, and x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.
ref_edge_type	Defines the permissible state transition type for reference node with r, f, and x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.

Table 72 Setup Time Check Parameters (Continued)

Parameter	Description
setup_time	Defines the setup time. A setup timing error occurs when the time difference between the permissible state transition time of the reference node and the permissible state transition time of the signal node is less than setup_time. If setup_time is negative, window_limit must be specified.
sub_name	When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.
window_limit	When window_limit is specified, the setup timing check error occurs when the signal node has a permissible transition at the time interval (t_ref - setup_time, t_ref + window_limit). If setup_time is negative, window_limit must be specified. If setup_time is positive and window_limit is not specified, the setup timing check error occurs when the signal node has a permissible transition at the time interval (t_ref - setup_time, t_ref).
logic_low_voltage	Represents the threshold voltage of logic state 0 (zero). The signal node is in logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Represents the threshold voltage of logic state 1 (one). The signal node is in logic 1 state if the signal node voltage is higher than logic_high_voltage.
ref_logic_low_voltage	Represents the low logic threshold voltages for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage is defined to the same value if it is not separately defined by refvlth.
ref_logic_high_voltage	Represents the high logic threshold voltages for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage follows the same value if it is not separately defined by refvhth.

Table 72 Setup Time Check Parameters (Continued)

Parameter	Description
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: This separate file is used to keep the output: hsim.setup_<title_name>.chk

Note:

If vhth=logic_high_voltage and vlth=logic_low_voltage are not specified, HSIM uses 0.7*HSIMVDD for vhth and 0.3*HSIMVDD for vlth.

Also, the setup timing check only checks the transition near the reference node edge.

Examples

```
.tcheck check1 setup data1 x clk f 2n
.tcheck check2 setup v(x1.*.data1) x clk f 2n
.tcheck check2 setup 'x1.*.data1' x clk f 2n
```

A setup time error occurs when any state transition at node data1 occurs less than 2 ns before the fall transition of the node clk. The error is reported following the title name of check1. Similar checks are performed for all nodes that match the pattern x1.*.data1. The setup timing error, if any, is reported in the timing/power check file following the title name of check2. The last two examples clearly illustrate different ways to specify the sig_name information.

```
.tcheck     check33 setup d x clk r 3n subckt=dff
```

subckt is specified in which d and clk are the node names inside the subckt. The setup time check reports error for every dff subcircuit instance if the state transition at data pin d occurs less than 3 ns before the rise transition at the clk pin within the same dff subcircuit instance.

```
.tcheck check3 setup qn f clk r -0.5n window=10n
```

A setup time error, following the title name of check3, is reported when a fall state transition at node qn occurs 0.5 ns after the node clk has a rise transition, and this fall state transition at qn occurs no later than 10 ns after the rise transition at node clk.

Hold Time Check

Checks whether signal node transition time minus reference node transition time is less than hold_time.

Syntax

```
.tcheck title_name hold sig_name edge_type ref_name
   ref_edge_type hold_time <subckt=sub_name>
   <window=window_limit> <vlth=logic_low_voltage>
   <vhth=logic_high_voltage>
   <refvlth=ref_logic_low_voltage>
   <refvhth=ref_logic_high_voltage> <separate_file=0|1>
```

Parameters

Table 73 Hold Time Check Parameters

Parameter	Description
title_name	Defines the title name of this hold time check. Hold time errors during this timing check are listed in the timing/power check error file and starts with this title name.
sig_name	Defines signal node name(s) that can be the node name of a single node, or a node name containing the asterisk (*) wildcard character that represents a group of node names. The node name with wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax all characters after '*' are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then sig_name should be the full path name.
ref_name	Defines the reference node name. ref_name is the node name of a single node, and it can not contain an asterisk (*) wildcard character. When using the subckt parameter, the ref_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then the ref_name must be the full path name.
edge_type	Defines the permissible state transition type for the signal node with r, f, and x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.
ref_edge_type	Defines the permissible state transition type for the reference node with r, f, and x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.

Table 73 Hold Time Check Parameters (Continued)

Parameter	Description
hold_time	Defines the hold time. A hold timing error occurs when the time difference between the permissible state transition time of signal node and the permissible state transition time of reference node is less than hold_time. If hold_time is negative, window_limit must be specified.
sub_name	When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.
window_limit	When window_limit is specified, the hold timing check error occurs when the signal node has a permissible transition at the time interval (t_ref – window_limit, t_ref + hold_time). If hold_time is negative, window_limit must be specified. If hold_time is positive and window_limit is not specified, the hold timing check error occurs when the signal node has a permissible transition at the time interval (t_ref, t_ref+hold_time).
logic_low_voltage	Defines the threshold voltage of logic 0 (zero) state of signal node. The signal node is in logic 0 state if its node voltage is less than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of logic 1 state of the signal node. The signal node is in logic 1 state if the signal node voltage is greater than logic_high_voltage.
ref_logic_low_voltage	Defines the low logic threshold voltage for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage will be defined to the same value if it is not separately defined by refvhth.
ref_logic_high_voltage	Defines the high logic threshold voltage for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage will be defined to the same value if it is not separately defined by refvhth.

Table 73 Hold Time Check Parameters (Continued)

Parameter	Description
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.hold_<title_name>.chk

Note:

The hold timing check only checks the transition near the reference node edge.

Examples

```
.tcheck check3 hold data x clk f 2.1n
```

A hold time error occurs when any state transition at node data occurs at the time interval (t, t+2.1 ns), where t is the time when node clk has a fall state transition The error is reported following the title name of check3.

```
.tcheck check4 hold qn f clk r -0.5n window=10n
```

A hold time error, following the title name of check4, is reported when a fall state transition at node qn occurs at the time interval (t –10 ns, t –0.5 ns) in which where t is the time when the node clk has a rise transition.

Pulse Width Check

Checks whether the pulse width (time difference between rise and fall state transitions) of the specified node meets the required range.

Syntax

```
.tcheck title_name pulsew sig_name low_min_time low_max_time
   high_min_time high_max_time <subckt=sub_name>
   <vlth=logic_low_voltage> <vhth=logic_high_voltage>
   <vlow=low_min_voltage> <vhigh=high_max_voltage>
   <separate_file=0|1>
```

Parameters

Table 74 Pulse Width Check Parameters

Parameter	Description
title_name	Defines the title name of this pulse width check. Any violation against the required pulse width range resulted from this check will be listed in the timing/power check error file, and starts with this title name.
sig_name	Defines signal node name(s) which can be the node name of a single node or a node name containing an asterisk (*) wildcard character representing a group of node names. The node name with a wildcard character must be specified as v(node_name) or node_name, because in the HSIM netlist parser and in SPICE syntax all characters after '*' are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit -- it must not be the full path name. If the subckt parameter is not used, then sig_name must be the full path name.
logic_low_voltage	Defines the threshold voltage of logic 0 stat. The node has a logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of logic 1 state of the node. The node has a logic 1 state if the node voltage is higher than logic_high_voltage.
low_min_time	Defines the minimum value of the low-state pulse. Each low-state pulse (the time period the node stays at the logic 0 state) is required to be greater than low_min_time and less than low_max_time.
low_max_time	Defines the maximum value of the low-state pulse. Each low-state pulse (the time period the node stays at the logic 0 state) is required to be greater than low_min_time and less than low_max_time.
sub_name	When this parameter is specified, the operation is performed to all instances of the particular subcircuit. When using the subckt sub_name parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.

Table 74 Pulse Width Check Parameters (Continued)

Parameter	Description
high_min_time	Defines the minimum value of the high-state pulse. Each high-state pulse (the time period the node stays at the logic 1 state) is required to be greater than high_min_time and less than high_max_time.
high_max_time	Defines the maximum value of the high-state pulse. Each high-state pulse (the time period the node stays at the logic 1 state) is required to be greater than high_min_time and less than high_max_time.
low_min_voltage	Defines the threshold voltage of logic 0 of the node. If defined, the node has a logic 0 state only when its minimum voltage during the pulse is lower than low_min_voltage. The default is the value of logic_low_voltage.
high_max_voltage	Defines the threshold voltage of logic 1 of the node. If defined, the node has a logic 1 state only when its maximum voltage during the pulse is higher than high_max_voltage. The default is the value of logic_high_voltage.
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.pulsew_<title_name>.chk

Examples

```
.tcheck check5 pulsew data[3] 8n 11n 7n 9n
```

A pulse width violation error, following the title name of check5, is reported when the low pulse width at node data[3] is less than 8 ns or greater than 11 ns; or when the high pulse width is less than 7 ns or greater than 9 ns.

The default value is used in this example because vlth is not defined. The default values are:

- vhth: 3V*0.7=2.1 volts
- vlth: 3V*0.3=0.9 volts

```
.tcheck chk_pwidth pulsew v(*) 0.3n 1000n 0.3n 1000n vlth=0.4
vhth=0.6 vlow=0.15 vhigh=1.0
```

This command checks the pulse width of all the nodes in the circuit. A violation will be reported for a node if the low pulse width is less than 0.3 ns or greater than 1000 ns; or when the high pulse width is less than 0.3 ns or greater than 1000 ns. A pulse is low when the node voltage is less than 0.4V, and the lowest voltage in a pulse must be lower than 0.15V. A pulse is high when the node voltage is higher than 0.6V, and the highest voltage in a pulse must be higher than 1.0V.

Note:

Using tcheck chk_pwidth pulsew v(*) 0.3n 1000n 0.3n 1000n vlth=0.4 vhth=0.6 results in small voltage dips to be reported. For example, a node moving from 0.5 volts dips to 0.35 volts and comes up still reports a pulsew error.

Timing Edge Check

Checks the time delay between two specified nodes and reports error when the delay doesn't meet the specified range.

Syntax

```
.tcheck title_name edge sig_name edge_type ref_name
   ref_edge_type min_time max_time <subckt=sub_name>
   <window=window_limit> <vlth=logic_low_voltage>
<vhth=logic_high_voltage> <refvlth=ref_logic_low_voltage>
<refvhth=ref_logic_high_voltage> <trigger=0|1|2>
   <separate_file=0|1>
```

Parameters

Table 75 Timing Edge Check Parameters

Parameter	Description
title_name	Defines the title name of this timing edge check. Every violation resulting from this timing check is listed in the timing/power check error file and starts with this title name.
sig_name	Defines signal node name(s) which can be the node name of a single node or a node name containing an asterisk (*) wildcard character representing a group of node names. The node name with wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit, and must not be the full path name. If the subckt parameter is not used, then the sig_name should be the full path name
ref_name	Defines the reference node name. ref_name is the node name of a single node, and it can not contain the asterisk (*) wildcard character. When using the subckt parameter, the ref_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then the ref_name should be the full path name.
edge_type	Defines the permissible state transition type for the signal node with r, f, or x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.
ref_edge_type	Defines the permissible state transition type for the reference node with r, f, or x. Notice that: ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition–from 0 to 1 or from 1 to 0.

Table 75 Timing Edge Check Parameters (Continued)

Parameter	Description
min_time	Defines the lower boundary of the timing edge difference between the signal and reference nodes. A timing edge error is reported when the timing edge difference is less than min_time. The timing edge difference is calculated only for the pair of permissible state transitions at the signal node and the reference node.
max_time	Define the upper boundary of the timing edge difference between the signal and reference nodes. A timing edge error is reported when the timing edge difference is greater than max_time. The timing edge difference is calculated only for the pair of permissible state transitions at the signal node and the reference node.
window_limit	Eliminates a timing edge error report if the timing edge difference exceeds window_limit. window_limit is optional.
logic_low_voltage	Defines the threshold voltage of the logic 0 (zero) state of the signal node. The signal node has a logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of the logic 1 (one) state of the signal node. The signal node has a logic 1 state if the signal node voltage is higher than logic_high_voltage.
ref_logic_low_voltage	Defines the low logic threshold voltage for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage will be defined to the same value if not separately defined by refvhth.
ref_logic_high_voltage	Defines the high logic threshold voltage for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage will be defined to the same value if not separately defined by refvhth.
sub_name	When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.

Table 75 Timing Edge Check Parameters (Continued)

Parameter	Description
trigger value=0|1|2	Defines the condition to trigger the timing edge check. ▪ 0 [default] If the value is 0, any permissible state transition at either signal node or reference node triggers the timing edge check. ▪ 1 Only the permissible state transition at the signal node triggers the check. ▪ 2 Only the permissible state transition at the reference node triggers the check. Trigger value is optional.
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0 [default] No separate output file. ▪ 1 The following separate file is used to keep the output the following: hsim.edge_<title_name>.chk

Examples

```
.tcheck check6 edge data x ctrl r 2n 5n
```

When node data has a state transition, such as at time t2, the time edge difference t2-t1 must be within the range of 2 ns and 5 ns. Otherwise, a timing edge error is reported following the title name of check6. Time t1 is the most recent rise state transition time at node ctrl before time t2. When node ctrl has a rise state transition at time t4, the time edge difference t4-t3 must be within the range of 2 ns and 5 ns. Otherwise a timing edge error is also reported. The time t3 is the most recent state transition time at node data before time t4.

```
.tcheck check7 edge data x ctrl r 2n 5n trigger=2
```

It is similar to the example above, except that the edge error check is triggered only by the rise state transition at node ctrl. Any edge error is reported following the title name of check7.

```
.tcheck check8 edge data x ctrl r 2n 5n trigger=2 window=10n
```

When node ctrl has a rise state transition at time t1, an edge error is reported (following the title name of check8) only when t1-t2 is less than 2 ns or is less than 10 ns but greater than 5 ns. The time t2 is the most recent state transition time at node data before time t1.

Timing Check Windows

.tcheck window

Defines the timing windows for the timing check commands.

Syntax

```
.tcheck window start_time1 <stop_time1 <start_time2
   <stop_time2 …>>>
```

Description

`.tcheck window` defines the timing windows for the timing check commands. All timing check commands use the same set of windows specified by this command. If this command is not specified, the default window for all timing check commands will be from the beginning of simulation to the end of simulation.

Multiple timing check windows can be specified by providing multiple pairs of start and stop time values. If the stop time is not provided in the last window, it will be extended to the end of simulation.

Example

```
.tcheck window 10n 20n 110n 120n 210n
```

In this example, the command specifies three timing check windows. The first window is from 10 ns to 20 ns, the second window is from 110 ns to 120 ns, and the third window is from 210 ns until the end of simulation.

Bisection Optimization

HSIM supports bisection optimization (see HSIMBISECTION on page 52). The bisection results are printed in the .optz file. The results of the .measure and .print statements of the last bisection iteration are stored in the .mt and .fsdb/.out files, respectively. Several statements are required to perform bisection optimization.

- .model statement
- .param statement
- .tran statement

.model

Syntax

```
.model opt_model_name opt method=bisection|passfail
   relin=tolerance
```

The optimization model reference name is opt_model_name. The keyword opt indicates that this particular .model statement is for bisection optimization usage. The same name is used in the corresponding .tran statement. method specifies the method to use in the bisection optimization–a valid value is either bisection or passfail with bisection being the default.

relin specifies the error tolerance. Its default value is 10e-3. Bisection optimization continues halving the target region until the interval between successive test values meets the criteria shown in the following equation:

$$\frac{|Xn-Xn-1|}{Xupper - Xlower|} \leq relin$$

Then, bisection optimization reports the value of Xn associated with the measured value that passed when method=passfail or the measure value minus the goal that is less than zero when method=bisection.

.param

Syntax

```
.param param_name=optxxx(init_value, low_value,
   upper_value)
```

Description

The name of the parameter used in bisection optimization is param_name. optxxx is the selected optimization parameter reference name–xxx can be replaced with a suitable choice. The same optxxx name is referenced in the corresponding .tran statement. The initial value, the lower boundary, and the upper boundary of the parameter are specified as init_value, low_value, and upper_value, respectively.

Example

```
.param param_name=alterxxx(value1, value2, .,valuen)
```

In the previous syntax, alterxxx is used with alterparam in .tran command. Please see the following .tran syntax.

.tran

Syntax

```
.tran steptime stoptime sweep optimize=optxxx
   result=measure_var model=opt_model_name
   <alterparam=alterxxx>
```

Description

The steptime and stoptime are specified as *steptime* and *stoptime*, respectively. optxxx is the same optimization parameter reference name in the corresponding .param statement. The result parameter specifies the measure variable defined in .measure statement. The model parameter is specified with the same model optimization reference name in the corresponding .model statement.

.alterparam

Description

The alterparam will accomplish the following:

- Alter the parameter value which is defined in alterxxx.
- Prevent HSIM from re-reading the netlist making the simulation faster.

Example

```
Vin ck 0 pwl
+0 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven d 0 pwl
+'0' 0
+'delaytime' 0
+'t_time2+delaytime' 5
+ bisectparam=1
.param delaytime=opt(0.0n, 0.0n, 20.0n)
.param t_time=alter1(0.1n, 0.1n, 0.1n, 0.5n, 0.5n, 0.5n, 1n, 1n,
1n)
.param t_time2=alter2(0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n, 0.1n, 0.5n,
1n)
.tran 0.1n 40.0n
+ sweep
+ optimize=opt
+ result=maxvout
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.model optmod opt
+ method=bisection
+ relin=1.0e-3
.measure tran ck_slope param=t_time
.measure tran d_slope param=t_time2
.measure tran mt_delaytime param=delaytime
.measure tran maxvout max v(q) goal=5
.measure tran setuptime trig v(d) val=2.5 rise=1
+ targ v(ck) val=2.5 rise=1
```

This test case extracts the setup time between data pin d and clock pin ck. In the .tran command there are two alterparams, alter1 and alter2. Both alterparams have nine values allowing HSIM to extract the setup time under nine different conditions. HSIM generates nine .mt files.

```
dff.mt.a0:
Measurement results:
ck_slope=1.000000000000e-010
d_slope=1.000000000000e-010
mt_delaytime=1.876953000000e-008
maxvout=5.000000000000e+000 at=4.000000000000e-008
setuptime=2.300000000000e-010 targ=1.905000000000e-008
trig=1.882000000000e-008
dff.mt.a1:
Measurement results:
ck_slope=1.000000000000e-010
d_slope=5.000000000000e-010
mt_delaytime=1.849609000000e-008
maxvout=5.000000000000e+000 at=4.000000000000e-008
setuptime=3.040000000000e-010 targ=1.905000000000e-008
trig=1.874600000000e-008
...
dff.mt.a8:
Measurement results:
ck_slope=1.000000000000e-009
d_slope=1.000000000000e-009
mt_delaytime=1.873047000000e-008
maxvout=5.000000000000e+000 at=4.000000000000e-008
setuptime=2.700000000000e-010 targ=1.950000000000e-008
trig=1.923000000000e-008
```

bisectparam=1 must also be added to the PWL voltage source to prevent HSIM from re-reading the netlist. HSIM only supports this feature in PWL voltage source as described in the following section.

Power Checking

This section describes the commands that perform power checks.

DC Path Check

This command checks the DC current path(s) among voltage sources in the circuit. All reported DC current paths are written into a separate file. This prevents mixing data with other violations reported by other timing and power checks.

Syntax

```
.pcheck title_name dcpath <ith=threshold_current> <node1
   <node2 …>> <period=period_time | delay=delay_time>
   <start=start_time1 <stop=stop_tim1 <start=start_time2
   <stop=stop_time2 …>>>>
.pcheck title_name dcpath <ith=threshold_current> <node1
   <node2 …>> at=t1 <at=t2 …>
```

Note:

Do not specify a Z-state within PWLZ when you perform a depth check. If a PWLZ source and Z-state are specified within PWLZ, HSIM does not consider PWLZ as a voltage source because there might be a period of time when PWLZ is not driven by voltage source elements. The dcpath check has no indicators of the starting VSRC node locations, and because of this situation, the reported path might not be a legitimate path.

Parameters

Table 76 DC Path Check Parameters

Parameter	Description
title_name	Each DC path is reported in the file title_name.
ith	Defines threshold_current, the value of the threshold current, which has a default value of 50 uA.
node1, node2 ... nodeN	The specified names define the nodes that DC current path checking will be performed between these nodes. The DC path search starts from any node specified in this node list and ends when it reaches another node in the list. If no node is specified, then HSIM reports dc current path(s) between any pair of voltage source nodes.
period	When specified, the DC path is checked for every period_time starting from the start time of each specified window defined by start_time and stop_time. If the period is not specified, the checking point will depend on the change points of PWL input sources. For better control of pcheck dcpath, always specify period= when start= is issued.
delay	When specified, the DC current path is checked at each time defined as t+delay_time. t is the time an input voltage source changes to a new voltage level. period and delay cannot be used simultaneously.
at	When specified, the DC current path is checked at the time defined by at=t1.
start	Specifies the start_time of the window. When specified, the checking point begins at this time. For better control of pcheck dcpath, always specify period= when start= is issued.
stop	Specifies the stop_time of the window.

Output File Name

If you specify -o out_file when invoking HSIM, the output file name is out_file.dcpath_title_name.chk. Otherwise, the default name is hsim_dcpath_title_name.chk.

Examples

```
.pcheck result1 dcpath ith=1u start=5n stop=110n
```

If the input source is defined as follows:

```
vin in 0 pwl (0 0 10n 0 11n 6.5 12n 6.5 12.5n 0)
```

and the pcheck command is issued without using -period then the following occurs:

The first checking point will be at time=5ns specified in start=

It will also check 11n and 12.5n point, not 12n point for it has the same value of point 11n.

Based on the analyses of the previous bullets, the checking points in this example are at time 5n, 11n, and 12.5n respectively.

```
.pcheck dc1 dcpath ith=1e-6 vdd gnd delay=5n start=10n stop=210n
```

.pcheck checks each DC current path between vdd and gnd every 5 ns after a voltage level change at any input voltage source. The dc current path is checked only during the time window between 10 ns and 210 ns. The dc current path is reported in the dc1 file hsim.dcpath_dcl.chk if the dc path current exceeds 1 uA when checked.

```
.pcheck dc2 dcpath ith=1e-6 vcc vss period=10n start=10n stop=210n
```

.pcheck checks each DC current path between vcc and vss in every 10 ns starting from simulation time at 10 ns and simulation time at 210 ns (included). The dc current path is reported in the dc2 file hsim.dcpath_dc2.chk if the dc path current exceeds 1 uA when checked.

```
.pcheck dc3 dcpath vcc vss at=130n at=150n
```

`.pcheck` checks each DC current path between vcc and vss at 130 ns time and 150 ns time. The DC current path is reported in the file dc3 if the dc path current exceeds the default value of 50 uA at 130 ns or at 150 ns time.

Excessive Current Check

The excessive element-current check exi checks when the current of specified element(s) exceeds a threshold value.

Syntax

```
.pcheck title_name exi elem1 <elem2…>
   <ith=threshold_current> <tth=exi_time>
   <start=start_time1 <stop=stop_time1 <start=start_time2
   <stop=stop_time2 …>>>> <level=val1> <factor=val2>
   <separate_file=0|1>
```

Parameters

Table 77 Excessive Current Check Parameters

Parameter	Description
title_name	Defines the title name of this excessive current check. Each excessive current report result from this check is listed in the timing/power check error file, and starts with this title name.
elem1, elem2 . . . elemN	The excessive current check is applied to each specified element elem1, elem2 . . . elemN. The element name can include the asterisk (*) wildcard character – the excessive current check applies to each element with matching pattern. The element name with a wildcard character must be specified as i(elem_name) or 'elem_name' because in the HSIM netlist parser and in SPICE syntax, all characters after (*)are treated as comments and are ignored.
ith	Defines threshold_current, the value of the threshold current. An element is reported to have excessive current if its element current exceeds threshold_current for a time duration longer than exi_time.
tth	Defines exi_time, the time duration. An element is reported to have excessive current if its element current exceeds threshold_current for a time duration longer than exi_time
start, stop	Specifies time window(s). The excessive element current is checked at the time within the specified time window(s). If no time window is specified, the check is performed from the time 0 ns to the end of simulation.
level	level is effective when the wildcard character is specified in elem1 <elem2..>. For additional information, refer to the .print command in Chapter 10, Simulation Output, Output Control Statements on page 337. The default value of val1 is -1.
factor	If factor is specified, pcheck will choose the larger one of threshold_current and val2*(w/l) as the new threshold_current. w is the transistor width and l is the transistor length.

Table 77 Excessive Current Check Parameters (Continued)

Parameter	Description
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: This separate file is used to keep the output: hsim.exi_<title_name>.chk

Example

```
.pcheck largei exi m1 m2 i(x1.*) ith=1e-3 tth=3n start=100n
```

`.pcheck` checks if the elements m1, m2, and all elements within instance x1 have current greater than 1 mA for longer than 3 ns. Any element detected with excessive current after 100 ns is reported in the timing/power check file out_file.chk following the title name of largei. The name out_file is the output file prefix with the default name being HSIM.

Excessive Rise/Fall Time Check

The excessive rise/fall time check exrf checks if the specified node(s) have excessive rise and/or fall times.

Syntax

```
.pcheck title_name exrf node1 <node2 …> <fanout=val2>
   <rise=rtime> <fall=ftime> <utime=utime1> <xsubckt=subx>
   <subinfo=0|1> <vlth=logic_low_voltage>
   <vhth=logic_high_voltage> <start=start_time1
   <stop=stop_time1 <start=start_time2 <stop=stop_time2
   …>>>> <separate_file=0|1>
```

Parameters

Table 78 Excessive Rise/Fall Time Check Parameters

Parameter	Description
title_name	Any excessive rise/fall time violation is reported in the timing/power check file xxxx.chk following the title name of title_name.
node1, node2 …nodeN	Defines signal node name which can be the node name of a single node or a node name with the asterisk (*) wildcard character that represents a group of node names. The node name with wildcard character must be specified as v(node_name) or node_name because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored.
fanout	Defines the fanout of driver nodes. If fanout is set to 1 (one), only the driver nodes with fanouts are checked to avoid unnecessary check on internal nodes within logic gates. If fanout is set to 0 (zero), both the internal node and the driver node are checked. The default value is 0. This parameter is optional. ▪ fanout=0: all nodes. ▪ fanout=1: nodes that have direct connection to transistor's gate. ▪ fanout=2: nodes that have direct connection to transistor's bulk.
rise	Defines the rise time of the signal as time duration t2-t1. t1 is the time when the rising signal voltage crosses the voltage level logic_low_voltage. t2 is the time when the same continuously rising signal voltage crosses the voltage level logic_high_voltage. The default value of rtime is 5 ns.
fall	The fall time (ftime) of the signal is defined as t4-t3. t3 is the time when the falling signal voltage crosses the voltage logic_high_voltage. t4 is the time when the same continuously falling signal voltage crosses the voltage logic_low_voltage. The default value is 5 ns.

Table 78 Excessive Rise/Fall Time Check Parameters (Continued)

Parameter	Description
utime	The U-state time (utime1)of the signal is defined as t6-t5. t5 is the time signal voltage enters the U state which is when the signal voltage is between logic_low_voltage and logic_high_voltage. t6 is the time the signal voltage leaves the U-state and the signal voltage is the same as the earlier voltage at t5 -- U-state represents an incomplete rise or fall transition. The default value is 5 ns.
xsubckt	Excludes the specified subcircuit from the check. Specify this option only when node1 is v(*). To exclude multiple subcircuits, use multiple xsuckt arguments: xsubckt=sub1 xsubckt=sub2 ... The maximum number of subcircuits to exclude is 1024.
subinfo	Specifies whether to print the subcircuit information in the report. The default is not to print the subcircuit informatin (0).
vlth	Defines the logic low state (logic_low_voltage).
vhth	Defines the logic high state (logic_high_voltage).
start, stop	The exrf check is performed within the time window defined by start_time1, stop_time1 . . . start_timeN, stop_timeN. An excessive rise/fall time violation is reported in the following conditions. ▪ The signal rise time exceeds rtime if it is specified following rtime, or utime1. ▪ The signal fall time exceeds ftime if it is specified following ftime, or utime1. ▪ The U-state duration exceeds utime1.
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.exrf_<title_name>.chk

Example

```
.pcheck longrf exrf a1 a2 v(x1.x2.*) fanout=1
+ rise=3n fall=4n vlth=0.3 vhth=2.7 start=100n stop=1000n
```

This command checks if the signal voltages at nodes a1, a2, and the driver nodes that match the pattern of x1.x2.* have excessive rise and fall times. An exrf is checked between 100 ns and 1,000 ns. A violation is reported in the timing/power check file following the title name of longrf if the signal rise time exceeds 3 ns, or the signal fall time exceeds 4 ns, or the U-state time exceeds the default of 5 ns. The U-state is defined as the voltage between 0.3V and 2.7V.

```
.pcheck check1 exrf v(*) rise=0.05n fall=0.05n vlth=0.5 vhth=4.5
+ start=0 stop=100n subinfo=1
```

Because subinfo is set to 1, the subcircuit information is printed as shown in the following report file:

…

```
check1: node v(xnand.in1a) in 'nand' excessive fall time
from time 1.0095e-08 to 1.016e-08 (6.5e-11)

check1: node v(xnand.x2.n1) in 'nand.nor' excessive rise
time from time 0 to 1.017e-08 (1.017e-08)
```

…

High Impedance Node Check

This command checks for the high impedance state in the circuit.

Syntax

```
.pcheck title_name zstate node1 <node2 …> <fanout=val2>
   <xsubckt=subx> <psubckt=subp> <ztime=ztime>
   <start=start_time1 <stop=stop_time1 <start=start_time2
   <stop=stop_time2 …>>>> <separate_file=0|1>
```

Parameters

Table 79 High Impedance Node Check

Parameter	Description
title_name	Every high-impedance is reported in the high-impedance check file .chk following the title name of title_name.
node1 .. nodeN	Each name, specified by node1 … nodeN defines signal node name. Each name can be the node name of a single node or a node name containing an asterisk (*) wildcard character which represents a group of node names. The node name with wildcard character must be specified as v(node_name) or ‘node_name’, because in the HSIM netlist parser and in SPICE syntax, all characters after (*) are treated as comments and are ignored.
fanout	The optional setting fanout=val2 limits the high impedance state checking to those driver nodes with fanouts. This setting will avoid unnecessary check on internal nodes within logic gates. The default fanout=0 will check both internal node and the driver node. ▪ fanout=0: all nodes. ▪ fanout=1: nodes that have direct connection to transistor's gate. ▪ fanout=2: nodes that have direct connection to transistor's bulk.
psubckt	Only checks the ports of the subcircuit subp. This option should be used only when node1 is v(*). To exclude multiple subcircuits, use one psubckt for each subp. For example: psubckt=sub1 psubckt=sub2 ... The maximum total number of psubckt is 1024.
xsubckt	Exclude the specified subcircuit subx for the check. This option should be used only when node1 is v(*). To exclude multiple subcircuits, use one xsubckt for each subx. For example: xsubckt=sub1 xsubckt=sub2 ... The maximum total number of xsubckt is 1024.

Table 79 High Impedance Node Check (Continued)

ztime	Any specified node staying in the high-impedance state longer than ztime will be reported in the timing/power check file following the title name of title_name. The default value of ztime is 5 ns.
start, stop	This check is performed within the time window defined by start_time1, stop_time1 .. start_timeN, stop_timeN.
separate_file=0|1	Specifies whether a separate output file is needed: ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.zstate_<title_name>.chk

Examples

```
.pcheck highz zstate a1 a2 v(x1.x2.*) ztime=50n start=100n
stop=1000n
```

This command checks the high-impedance state of the nodes a1, a2 and those nodes matching the pattern of x1.x2.*. The high-impedance state is checked between 100 ns and 1,000 ns. If any specified node remains in the high impedance state longer than 50 ns, it will be reported in the timing/power check file following the title name of highz.

```
.pcheck highz zstate v(*) xsubckt=wctrl xsubckt=rctrl
```

This command checks the high-impedance state of all nodes except those in subcircuits wctrl and rctrl.

Power Check Windows

Syntax

```
.pcheck window start_time1 <stop_time1 <start_time2
   <stop_time2 …>>>
```

Description

`.pcheck window` defines the time window(s) for each power check command that doesn't have its own time window specification. Each power check with its own time window specification will not be affected by this window setting.

Examples

```
.pcheck window 10n 20n 110n 120n 210n
```

This sets three time windows for those power checks without time window specification. The time windows are from 10 ns to 20 ns, from 110 ns to 120 ns, and from 210 ns to the end of simulation.

Activity Checking

This section describes the commands that perform activity checks.

Active Node Check

.acheck finds the active nodes in a design. A node is active if it has a voltage change during the simulation runtime.

Syntax

```
.acheck node_pattern <level=val2> <dv=val> \
   <start=start_time>
<stop=stop_time> <exclude=exc_pattern>
```

Parameters

Table 80 Activity Checking Parameters

Parameter	Description
node_pattern	Defines the signal node name(s) which can be the node name of a single node or a node name containing wildcard character '*' representing a group of node names. The node name with wildcard character must be specified as v(node_name) or node_name, because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored.
dv=val	Defines the threshold of voltage difference. A node is active when the voltage change, compared to the initial value of the node, is larger than val. DEFAULT of val is 0.1 volt.
exclude=exc_pattern	Defines the signal node name(s) which are excluded from the list of nodes that need to be checked. Wildcard characters can be used and need to be quoted such as: ‘a*’
level=val2	This setting is effective only when the wildcard character is specified in the output variable. The level value val2 specifies the number of hierarchical depth levels when the wildcard node/element name matches. The level option acts on both active and inactive files. ▪ When val2 is set to 1, the wildcard match applies to the same depth level where the `.acheck` statement is located. ▪ When val2 is set to 2, it applies to the same level and to one level below the current level where `.acheck` is located. ▪ When val2 is set to 0, no wildcard name is matched so that any output variable containing wildcard character is ignored. ▪ When val2 is set to –1, the wildcard match applies to all the depth levels below and including the current level of `.acheck` statement. ▪ The default value of val2 is –1.
start=start_time, stop=stop_time	Specifies the start_time and stop_time in the time window. The activity is checked at the time within the specified time. If no time window is specified, the check is performed from the time 0 ns to the end of simulation.

After simulation, the active node list is saved in the .ach file, The inactive node list is saved in .ina and .ina_all files. The .ina file keeps all the inactive nodes not affected by their surrounding active nodes. The .ina_all file keeps all the inactive nodes in the entire circuit. .ina_all file can be huge. The formats of the activity files are in ASCII and line-by-line. Example shows the .ach file.

Note:

In the case of a 2-input NAND gate with in1 and in2 inputs and an out output, if in1 is connected to GND and in2 is given an input pulse, the out node will be reported in the .ina file because the out node is an inactive node that is not affected by the in2 active node.

Active Files

Example

```
*HSIM Win32
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Active driver node list
ina
```

Inactive Files

The format of the inactive files is:

Syntax

```
<index> <node_name> 0 <voltage>
```

Parameters

Table 81 Inactive File Parameters

Parameter	Description
index	A generated index name
node_name	The inactive node name
voltage	The constant voltage kept by the node

Example

```
*HSIM Win32
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Inactive driver node list (complete)
V_inact_0 0 0 0
V_inact_1 xinvseta.inv 0 1.52086
V_inact_2 xinvsetb.inv 0 3
```

16

Monte Carlo Analysis

Provides information on how Monte Carlo Analysis (MCA) calculates the circuit response to the statistical variation of design and model parameters by applying randomly selected values from user-specified distributions.

Monte Carlo Analysis (MCA) calculates the circuit response to changes component values by randomly varying all of the model parameters for which a tolerance is specified. This provides statistical data on the impact of a device parameter's variation on circuit operation. Model parameters are given tolerances. Multiple simulations are run using these tolerances for:

- DC analysis
- Transient analysis

One frequent use of MCA is to predict the yields of circuit production runs. MCA obtains statistical information by applying random variation to parameters in the following:

- Device models
- Circuit element instances

Parameters are specified with nominal values and tolerances along with the selected random number distribution function. The analysis involves:

- Several simulation runs
- Collecting all simulation results
- Producing a statistic summary report

The MCA commands are described in the following sections.

Monte Carlo Analysis Features

Selecting the Monte Carlo Analysis Type

DC Analysis

```
.dc var start stop incr sweep monte=iter_num
```

Transient Analysis

```
.tran steptime stoptime sweep monte=iter_num
```

The *iter_num* represents the number of HSIM iterations (for example: simulation runs) to be performed.

Selecting the Distribution Function Parameter Type

Three distribution functions can be used in the analysis:

- Gaussian Distribution
- Uniform Distribution
- Limited Distribution

The syntax of the three distribution functions are shown below:

Gaussian Distribution

Either of the following syntax statements can be used.

```
.param param_name=gauss(nominal_val, rel_variation,
   + sigma)
.param param_name=agauss(nominal_val, abs_variation,
   + sigma)
```

Uniform Distribution

Either of the following syntax statements can be used.

```
.param param_name=unif(nominal_val, rel_variation)
.param param_name=aunif(nominal_val, abs_variation)
```

Limit Distribution

```
.param param_name=limit(nominal_val, abs_variation)
```

param_name

The parameter whose value is calculated by applying the distribution function.

gauss

Gaussian distribution using relative variation.

agauss

Gaussian distribution using absolute variation.

unif

Uniform distribution using relative variation.

aunif

Uniform distribution using absolute variation.

limit

Limit distribution using absolute variation.

Note:

abs_variation=rel_variation * nominal_value.

To take the effects of both element and model parameter variation into account, the distribution function is recalculated for each circuit element and each device model.

Specifying Starting Seed

```
.OPTIONS SEED=<seed_number>
```

Use .OPTIONS SEED to specify the starting seed for the random number generator.

Selecting the HSIM Monte Carlo Analysis Mode

Use the following commands for Monte Carlo analysis:

- HSIMMONTECARLO on page 95
- HSIMMONTECARLOINST on page 95

- HSIMMONTECARLOINST on page 95
- HSIMMONTECARLOSAVEOUT on page 96

The results of simulation and .measure statements are saved into files with an <prefix>_mc.mt suffix. The model parameter variation values of each iteration are saved in the files using the .mmp suffix.

Part: 3 HSIM Advanced Modeling

17

Using Verilog-A with HSIM

Provides information on how the Verilog-A compiler adds behavioral modelling capability to HSIM. This implementation of Verilog-A is based on the analog subset of Open Verilog International's Verilog-AMS Language where, the Verilog-A parser reads source code written in the Verilog-A language and provides HSIM with the necessary simulation information. With the Verilog-A feature, designs including Verilog-A modules can be included in the HSIM netlist.

Verilog-A Compiler

The Verilog-A compiler adds analog behavioral modeling capability to HSIM. The Verilog-A parser reads source code written in the Verilog-A language and provides HSIM with the necessary simulation information. With the Verilog-A feature, designs including Verilog-A modules can be included in the HSIM netlist.

The HSIM implementation of Verilog-A is based on the analog subset of *OVI, Verilog-AMS Language Reference Manual, v2.0*. Limitations and extensions are listed in Verilog-A Language and HSIM Implementation-Specific Features on page 482.

Note:

Additional information is contained in the following documentation: *OVI, Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL, Version 2.0. Open Verilog International*.

Verilog-A Options

Compiler Option

In addition to interpreted evaluation, HSIM provides a Compile option for Verilog-A. The Verilog-A parser currently provides two types of compilation for a Verilog-A module:

- Batch: Batch compilation is enabled by setting the environment variable HSIM_COMPILE_VERILOGA to 1, y, or Y. This option creates a compiled library for all parsed Verilog-A modules
- Module-based: Module-based compilation is specified by including the hsim_compile_module in a module definition command line. This option creates a compiled library for a specific module that can be evaluated at a later time as shown in Example 38.

Example 38 Module-Based Compilation Syntax

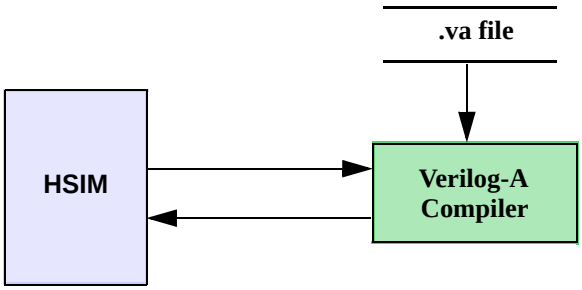
```
module aaa(a,b,..)
// Put the specifier in the module definition,
// such as the following line,
// 'hsim_compile_module
inout a, b;
...

endmodule
```

Verilog-A Architecture

The HSIM Verilog-A compiler interface is shown in Figure 18 on page 472.

Figure 18 Verilog-A Compiler Architecture



Getting Started with the Verilog-A Compiler

The Verilog-A compiler is invoked within HSIM during the HSIM simulation run.

The following code sample is provided to familiarize users with Verilog-A.

1. Create an example Verilog-A source file, res.va.

```
resistor
`include "discipline.h"
module resistor(a,b);
inout a,b;
electrical a,b;
branch(a,b) res;
parameter real R=1;
analog begin
I(res) <+ V(res)/R;
end
endmodule
```

2. Create the example HSIM netlist file, res.sp.

```
* res.sp
.param HSIMVERILOGA="res.va"
vsource 1 0 DC 5
r1 1 2 10K
x1 2 0 resistor R=10K
c1 2 0 1uF
.tran 0.1n 100n
.end
```

3. Start HSIM with the command.

```
HSIM.exe res.sp
```

This process will successfully run a Verilog-A simulation using HSIM.

Including Verilog-A Modules in HSIM Simulations

The HSIM parser invokes the Verilog-A parser when it executes on a netlist containing specific Verilog-A include statements. After parsing the module, the Verilog-A parser registers the module with the HSIM simulator. All the information, warning and error messages go into the simulation log file.

Verilog-A files are written in text file(s) that are usually named as <filename>.va. Although, it is a good practice to name your Verilog-A files with

.va extension, it is not required by HSIM to do so. A sample Verilog-A file is shown below.

```
File "res.va":

`include "discipline.h"
`include "constants.h"
// res
//
// - resistor
//
// vp,vn:       terminals (V,A)
//
// INSTANCE parameters
//    r=resistance (Ohms)
//
// MODEL parameters
//    {none}
//

module res(vp, vn);
inout vp, vn;
electrical vp, vn;
parameter real r=1K;
     analog
          I(vp, vn) <+ V(vp, vn)/r;

endmodule
```

Important:

"discipline.h" and "constants.h" are header files that define the disciplines/natures and constants respectively. These files are searched according to the path given in the Verilog-A file. If they are not found, they will be read from $HSIM_path/etc/include. The files included in this directory include:

"discipline.h"

"disciplines.h"

"constants.h"

"disciplines.vams"

"constants.vams"

Including Verilog-A Modules in a SPICE Netlist

Verilog-A modules are included in a SPICE netlist by specifying the source file name in HSIMVERILOGA on page 166 or an HSPICE .hdl statement.

Syntax

```
.param HSIMVERILOGA=<"file_name">
.hdl "file_name"
```

Arguments

file_name <file_name>

file_name may include a relative or absolute path name and is the name of the Verilog-A source file to be compiled. The Verilog-A model is then instantiated in the netlist by "X<text>" like a SPICE subcircuit.

Example

A sample SPICE file with Verilog-A statement is below.

```
* sample RC circuit
.param HSIMVERILOGA="./res1.va"

vpulse n0 gnd pulse 0 3000.0m 0 30n 30n 300n 600n
x1 n0 n1 res1
c1 n1 gnd 100p
rr1 n0 nn1 1K
cc1 nn1 gnd 100p

.meas n0 rms v(n0)
.meas n1 rms v(n1)
.meas n2 rms v(nn1)

.tran 1n 2000n
.print v(*) i(cc1) i(rr1) level=1
.param HSIMOUTPUT=wdf
.end
```

Including Verilog-A Modules in a SPECTRE Netlist

To incorporate Verilog-A modules (contained in a Verilog-A source file) in a SPECTRE® netlist, specify the source file name in the ahdl_include statement in the netlist.

Syntax

```
ahdl_include <file_name>
```

Arguments

file_name

file_name may include a relative or absolute pathname as the name of the Verilog-A source file to be compiled. Since, HSIM can run SPECTRE netlists directly. The Verilog-A model is then instantiated in the netlist by any SPECTRE element.

Example

A sample SPECTRE netlist file is given below.

```
simulator lang=spectre

ahdl_include "res.va"


v1 n0 gnd vsource type=dc dc=1
a95 n0 n1 RES r=1k
r0 n1 gnd resistor r=1k

.end
```

Including Verilog-A Modules in an ELDO Netlist

To incorporate Verilog-A modules (contained in a Verilog-A source file) in an ELDO netlist, specify the source file name in a .verilog statement in the netlist.

Syntax

```
.verilog <file_name>
```

Arguments

file_name

file_name may include a relative or absolute pathname as the name of the Verilog-A source file to be compiled. The instantiation is done similar to SPICE/SPECTRE netlists.

Verilog-A Model Explanation

The following examples are included to provide a better understanding of Verilog-A models.

The following code resistor.va shows how a resistor can be implemented and coded in Verilog-A. The numbers shown on the margins of the Verilog-A file are

only for illustration purposes, please do not include them into the actual Verilog-A source file. Refer to examples for the disciplines.h file.

Example 39

```
0 // resistor.va
1 `include "discipline.h"
2 module resistor(a,b);
3 inout a,b;
4 electrical a,b;
5 branch(a,b) res;
6 parameter real R=1;
7 analog begin
8 I(res) <+ V(res)/R;
9 end
10 endmodule
```

The following is a line-by-line explanation of the code:

Line no.	Code	Description
0		This line is a comment. All comments start with //.
1	`include discipline.h`	Directs the compiler to insert the contents of discipline.h before compiling.
2	`module resistor(a,b)`	Defines the interface between the resistor module and HSIM. It tells HSIM that there is a module called resistor that has two ports, a and b.
3	`inout a,b`	Specifies the direction of the ports. The Verilog-A language standards define three port directions, input, output and inout. Each port must have a direction associated with it.
4	`electrical a,b`	Defines the disciplines for each of the ports. Users can define and name a discipline. Typical disciplines may be electrical, thermal, kinematic, etc. Refer to disciplines.h for defining disciplines.
5	`branch(a,b) res`	Defines a branch res. A branch is simply defined as a path between two nodes.

Line no.	Code	Description
6	`parameter real R=1`	Defines the R parameter that has a default value of 1 (one). Parameters provide another interface between the HSIM netlist and Verilog-A module. If you decide to specify another value for R, this may be done at the HSIM netlist level (without the need to change and recompile the Verilog-A module in the source file).
7	`analog begin`	Specifies the beginning of an analog block. An analog block is where the behavioral description of the component being modeled is specified. There can only be one analog block for each module.
8	`I(res) <+ V(res)/R`	Specifies the behavioral description of the resistor. The <+ in this line is called a contribution statement. The line is read as: The current flowing through branch res is contributed to by the voltage across branch res divided by the parameter P.
9	`end`	Specifies the end of the analog block.
10	`endmodule`	Specifies the end of the module.

An example HSIM netlist, res.sp, which contains a reference to the Verilog- A module, is shown in Example 40.

Example 40

```
0 ****** res.sp
1 .param HSIMVERILOGA “resistor.va”
2 vsource 1 0 dc 5
3 r1 1 2 10K
4 xr2 2 0 resistor R=1OK
5 c1 2 0 1uF
6 .tran 1n 1u
```

7 .end

The following is a line-by-line explanation of the code

Line no.	Code	Description
0		HSIM comment line.
1	`verilog resistor.va`	A reference to the Verilog-A source file, resistor.va. It tells HSIM that there is a Verilog-A file to be used within the netlist.
2	`vsource 1 0 dc 5`	Creates a 5V DC voltage source.
3	`r1 1 2 10K`	Creates a 10k resistor.
4	`xr2 2 0 resistor R=1OK`	Defines an instance, xr2, of the module, resistor. Note that x defines a Verilog-A component (much the same as an r defines a SPICE resistor). Once the instance has been named (in this case xr2), the nodes that it connects to are specified. These nodes will correspond to the terminals specified in the Verilog-A module contained within the res.va file. In this example, node 2 corresponds to terminal a and node 0 corresponds to terminal b in the Verilog-A module resistor. Once the nodes have been specified, assignment of the parameters is done. In this case, the parameter R is assigned a value of 10k, overriding the default value of 1.
5	`c1 2 0 1uF`	Creates a 1uF capacitor.
6	`.tran`	Specifies transient analysis.
7	`.end`	Marks the end of the SPICE netlist.

Executing the HSIM command on this netlist causes HSIM to invoke the Verilog-A compiler. The Verilog-A compiler would do the following:

- Compile the res.va input file.
- Register the module resistor with HSIM, given that res.va is syntactically and semantically correct.
- The Verilog-A compiler will send all the Info, Warning, and Error messages to the simulation log file.

- If res.va contains syntactic or semantic errors, all error messages would be directed to the simulation log file.
- The xr2 instance can then be used in an HSIM simulation, just like any other instances, within the supported analysis types.

HSIM .log file Information

During simulation, HSIM prints several messages related to the Verilog-A models. When HSIM invokes the Verilog-A parser, parser issues and information messages are seen on line 13. For each model, read into this module file, the parser issues a message when registering these models on line 14.

.log file

When the circuit statistics are generated at a later time, the number of times Verilog-A models are instantiated are shown. The Verilog-A models are listed as VAMOD elements on line 29. A model-by-model list of instantiations are also given in the .log file on line 37. Example 41 illustrates the .log file output.

Example 41

```
12 Reading (pass #1) 'res.sp'
13 Reading Verilog-A model file: res.va
14 Register Verilog-A Module:resistor
15 Reading (pass #2) 'res.sp'
          :
          :
24 Circuit Statistics
25
26 RES    Elements:1
27 GCAP   Elements:1
28 VSRC_DCElements:1
29 VAMOD  Elements:1
30
31 Total # of Elements:3
32 Total # of Nodes:4
33
34
35 Verilog-A Model Instances
36
37 resistor    :1
38             :
```

Case Sensitivity Issue In SPICE Netlist

Since Verilog-A Language is case sensitive and SPICE is case insensitive, there might be problems with the module names. In order to overcome this issue, HSIM looks for the lowercase model name in default mode. If the -case 1 option is used during HSIM invocation, then HSIM will honor the case of the Verilog-A module name.

Verilog-A Module Name Clash with Sub-circuit Names

Whenever there is a clash between the names of subcircuits in an HSIM netlist, C-model names and Verilog-A model names, HSIM resolves the name in the following order.

1. Sub-circuit Name
2. C-model Name
3. Verilog-A Model Name

For each X instance in the HSIM netlist, HSIM first attempts to match the name with the subcircuit name. If the subcircuit name is not matched, then HSIM tries to find a C-model with the same name. Verilog-A models with that name are searched last.

To increase the Verilog-A priority vs. other model types in HSIM, use HSIMVERILOGA on page 166.

When HSIMPREFERVERILOGA on page 112 is specified, the Verilog-A Model Name option supersedes all other options and the order changes as follows:

1. Verilog-A Model Name
2. Subcircuit Name
3. C-model Name

Supported Features

Features supported in HSIM Verilog-A include:

- AC Analysis
- ac_stim Functions
- z-Transform

Verilog-A Language and HSIM Implementation-Specific Features

This section provides an overview of the Verilog-A language and specific HSIM implementations. Complete Verilog-A details are provided in the OVI, Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL, Version 2.0. Open Verilog International and other related textbooks.

HSIM Implementation-Specific Features

HSIM implementation-specific features documented in this manual include the following:

- Data types
- Parameters

Integer or real type parameter declarations are supported by HSIM Verilog-A. Since they are converted to real double precision numbers during simulation, the default type is real if not otherwise specified. Only scalar parameters are supported.

Parameters can also be declared with an optional permissible range. The parameter value is verified with the specified range during instantiation stage. Simulation aborts if verification fails.

Parameters represent constants. Their values are resolved at instantiation stage and cannot be modified during simulation. Following are examples:

Example 42

```
parameter integer rate1=12 from [0:inf] exclude (30:50) exclude
100;
parameter rate2=12; //real is the default type.
parameter real rate3=12;
parameter real poles[0:2]={1.0, 2.0, 3.0}; // Not allowed.
```

In these cases, rate1 is declared as an integer with default value of 12 and, rate2 and rate3 are declared as real parameters.

Integer and Real Number Variables

Variables can be declared as integer or real. Arrays are supported. Values of variables are preset to zero during instantiation and the values can be updated during simulation. Examples are as following:

```
parameter integer ran=5
integer a1[1:ran];
real a2[1-:5];
```

Note:

HSIMs .store/.restore feature does not support Verilog-A variables.

Natures and Disciplines

Natures define a set of attributes for physical and simulation quantities, such as units, access function names or tolerances. Sample excerpts from standard definitions are shown below:

```
nature Current
     units="A";
     access=I;
     abstol=1e-12
endnature
nature Voltage
     units="V";
     access=V;
     abstol=1e-6;
endnature
```

The three attributes, units, access and abstol, are mandatory for declaration of natures. HSIM aborts when access is not defined but defaults units and abstol to NA and 1e-6 if they are not defined.

Disciplines bind natures to potential and flow as defined in Kirchoff's laws.

Kirchoff's Potential Law

Potential binding defines the access function of nature.

Kirchoff's Flow Law

Flow binding defines that of the nature which obeys Kirchoff's Flow Law.

An example defining analog circuit signal electrical is as follows:

```
discipline electrical
     potential    Voltage;
     flow         Current;
enddiscipline
```

Multiple declarations do not stop HSIM parsing. However, the new definition is discarded with a Warning message.

Nodes

Nodes in Verilog-A modules are associated with disciplines. A node declared without a range is a scalar node. A node declared with a range is a vector node or analog bus. HSIM only allows constant node widths overriding the width during instantiation is not permitted.

```
electrical out, in; // sacalar nodes
electrical [5:10] node1; // vector nodes
```

Branches

Branches can be explicitly declared by the branch keyword and given terminal pairs. HSIM only supports declaration of scalar branches. A branch defines a path between two terminals. Potential difference, i.e. voltage, when the terminals are declared as electrical, can be accessed using the access function V(<branch name>) defined by voltage nature. Flow, i.e. current for electrical terminals, is accessed by I(<branch name>) defined by current nature. Examples of declarations are as shown below:

```
electrical a, b
electrical [-3:0] a1;
electrical [17:20] b1;
branch (a, b) br1;
branch (a1, b1) x; // not supported, bus to bus
branch(a1[-2], b1[17]) y;
```

To access potential difference and flow of branch br1, we use V(br1) and I(br1).

Implicit branch declaration is also supported. When an access function, for example V(b), is referred to in the behavioral description and b is a node, an

implicit branch is constructed between node b and ground. Similarly V(a,b) where a and b are nodes, constructs a branch from a to b.

There can be only one implicit branch between the same two terminals. That is, multiple references of V(a) are all referring to the same branch. This also applies to V(a,b) and V(b,a), for which the access functions return the voltage of V(a)-V(b) and V(b)-V(a) respectively.

Operators

The supported operators for expressions are tabulated in this section.

Unary Operators

Table lists the Verilog-A ternary operators.

Unary Operators

Table 82

Operator	Description	Allowable Operands	Example
+	positive	Integer, real	I=+10; //I=10
-	negative	Integer, real	R=-14.1; // R=-14.1

Binary Operators

Table lists the Verilog-A ternary operators.

Binary Operators

Table 83

Operator	Description	Allowable Operands	Example
+	add	Integer, real	I=10+2.1; // I=12.1
-	subtract	Integer, real	I=10-3.7; // I=6.3
*	multiply	Integer, real	I=2*4.2; // I=8.4
/	divide	Integer, real	I=4/2.0; // I=2.0

Table 83

Operator	Description	Allowable Operands	Example
<	less than	Integer, real	I=5<7; // I=1
>	greater than	Integer, real	I=5>7; // I=0
<=	less than or equal to	Integer, real	I=5<=7; // I=1
>=	greater than or equal to	integer, real	I=5>=7; // I=0
==	equal to	Integer, real	I=5==7; // I=0
!=	not equal to	Integer, real	I=5!=7; // I=0
&&	logical and	Integer, real	I=(1==2) &&(3==3); // I=0
||	logical or	Integer, real	I=(1==2) ||(3==3); // I=1
or	event or	Event expression	@(initial_step or cross(V(in), +1))

Ternary Operators

Table lists the Verilog-A ternary operators.

Ternary Operators

Table 84

Operator	Allowable Operands	Example
Cond? expr1:expr2	Integer, real	I=(3>2)? 1: 0;

Cond, expr1 and expr2 are all expressions. The operator evaluates expr1 if the condition is true, and otherwise expr2.

Statements

Supported statement types include the following:

- Procedural Assignment
- Branch Contribution
- Block
- If-Else
- Case
- Repeat Loop
- While Loop
- For Loop
- Generate

Procedural Assignment Statements

Procedural assignment statements evaluates the expression on the RIGHT hand side and assigns the value to a variable on the LEFT hand side. The variable has to be modifiable during simulation, i.e. it is declared as integer, real or an integer element or real array. Procedural assignment takes effect immediately when the statement is executed. The RIGHT-hand operand can be any arbitrary scalar expression.

```
real a[0:2], sum;
.
.
.
r[0]=5.0;
r[1]=3.0;
r[2]=2.0;
sum=r[0]+r[1]+r[2];
```

Branch Contribution Statements

The LEFT-hand operand of the branch contribution specifies a source branch signal. It is in the form of access function of the branch, to which the RIGHT hand side expression contributes. During simulation, HSIM evaluates the value of the RIGHT-hand side expression and accumulates it to the source branch. After all the statements in the module having been executed, thus the algebraic expressions for the model have been evaluated, the system is solved by the HSIM kernel for the time point. Therefore, the contribution statement is also referred to as a simultaneous assignment.

It is important to note that this operation is cumulative. Voltage contribution branches are summed up as if they are in series, while current contribution branches are summed up in parallel. In the following two contribution statements:

```
V(br) <+ 3;
V(br) <+ 4;
```

the module is equivalent to the following single statement:

```
V(br) <+7;
```

Block Statements

A block statement consists of a list of statements. The statements are executed sequentially as in the following syntax example, which shows a begin and end statements:

```
begin
     integer k;
     k=k+1;
end
```

An if-else statement describes conditional execution behavior as shown in the following example:

```
if(value<0) $strobe("value < 0");
else if((value>0) && (value<=1)) $strobe("value in (0,1]");
else if((value >1) && (value<=2)) $strobe("value in (1,2]");
else $strobe("value > 2");
```

Case Statements

A case statement defines a multiway decision behavior structure. The testing condition, which is the argument expression to the case, is evaluated and compared against the condition labels in the case items. When found, the execution branches to the corresponding statements. When no matching condition is found and default statements are defined after default label, the statements are executed. Unlike C language, the condition label can also be an expression, which is evaluated during condition matching. The following syntax example illustrates a case statement:

```
case(1)
     (value<0): $strobe("value < 0");
     ((value>0) && (value<=1)): $strobe("value in (0,1]");
     ((value>1) && (value<=2)): $strobe("value in 1,2]");
     default $strobe("value > 2");
endcase
```

Repeat Loop Statements

A repeat loop statement is used to run the enclosed statements repeatedly for a fixed number of times. A repeat statement repeats the associated statements a fixed number of times given by the argument to the repeat keyword as shown in the following syntax example:

```
repeat(5) begin
     i=i+1;
     total=total + i;
end
```

While Loop Statements

The while statement executes the associated statements repetitively when the condition expression to the while keyword is evaluated as true as shown in the following syntax example:

```
while(a<10) begin
     i=i+1;
     sum=sum+i
     a=a+1;
end
```

For Loop Statements

The for loop statement executes the associated statements repetitively when the condition expression evaluates true. The statement extends the previous while statement with additional index manipulating facilities as shown in the following syntax example:

```
for(i=1; i<10; i=i+1) begin
     i=i+1;
     sum=sum+i
end
```

Generate Statements

The generate statement unrolls its statement during compilation as shown in the following syntax:

```
generate i (3,1) begin
     V(out[i]) <+ 3;
end
```

The generate statement is compiled as follows:

```
V(out[3]) <+ 3
V(out[2]) <+ 3
V(out[1]) <+ 3
```

Analog Events

initial_step returns true when the simulation is at the first step in simulation. It has the following syntax:

```
@(initial_step) begin
    v1=0;
    v2=1;
end
```

Limitation:

HSIM does not support initial_step with any argument.

Cross Statements

cross

`cross` returns true when the signal in its argument crosses the given threshold along a specified direction. `cross` also adjusts the simulation timestep in order to precisely catch the processing point.

```
@(cross(V(in)-2.5, +1)) begin
     V(out) <+ transition(x, 1n, 3n);
end
```

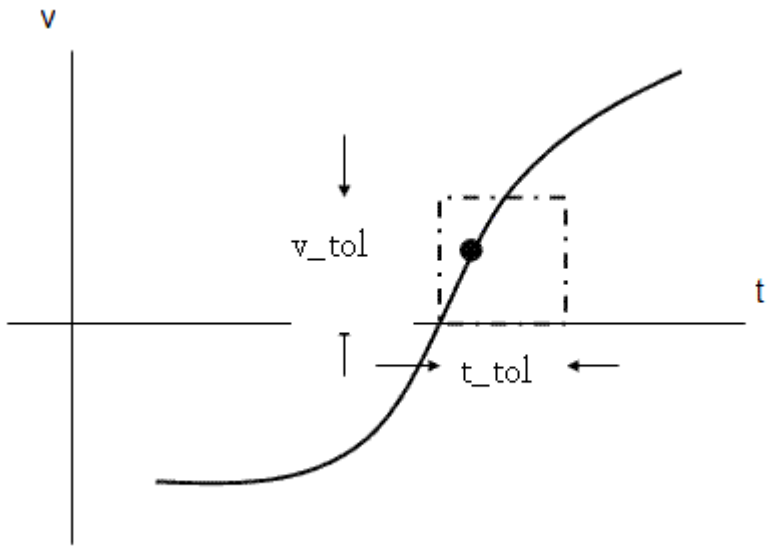
In this example, when "V(in)-2,5" becomes 0 "and" is going in the positive direction, the action below the `cross` statement will be evaluated and executed.

HSIM places the simulation time step close to the crossing point during the VA crossing event. By default, this should be determined by the simulator engine dynamically, however, you can apply the third optional parameter t_tol and fourth optional parameter s_tol to the `cross` function as guidance for HSIM to place the time step properly.

```
@(cross(V(in)-2.5, +1, 10p, 0.01v) begin
     V(out) <+ transition(x, 1n, 3n);
end
```

This example indicates a similar execution procedure, except that the simulation time step (the event) will most likely fall into the box formed by t_tol(10ps) and s_tol(0.01v). The t_tol and s_tol parameters have a default value of 0. See Figure 19 on page 491 for an illustration of the example.

Figure 19 Simulation Time Step Placement



HSIM has its own time step selection scheme during the simulation, therefore it may not fully honor the setting of "t_tol" and "s_tol" in the `cross` function statement to accurately locate the time step as expected. To enforce the simulation time step falling inside the t_tol and v_tol box, use HSIMVACROSSTTOL on page 162 and HSIMVACROSSVTOL on page 162 to control the time step selection for the `cross` function.

Timer Statements

`timer` returns true when the simulation time reaches the time point or periodic time points specified in its argument as shown in the following example:

```
@(timer(0, period)) x=-x;
V(out) <+ transition(x, 1n, 10n);
```

In the syntax example above, x changes sign at t=0 and 0+n*period, where n is a positive integer. The `period` cannot be negative and it must be greater than zero. An error message is displayed when an illegal period is assigned to the `timer` function.

Analog Operators

ddt Operator

The `ddt` operator computes the time derivative of its argument. In current implementation, tolerance specification is ignored and the value of HSIMALLOWEDDV on page 47 parameter is used. Following isis the syntax for ddt:

```
V(out) <+ ddt(V(in));
```

idt Operator

The `idt` operator computes the time integral of its argument. In the current implementation, the tolerance specification is ignored and the value of HSIMALLOWEDDV on page 47 is used. Following is the syntax for `idt`:

```
V(out) <+ idt(V(in),1);
```

delay Operator

The `delay` operator delays a continuous signal by a given delayed time. The delay time and maximum delay have to be positive. The following error message is given when either of them evaluates negative:

```
DelayDelay/maximum delay time has to be positive.
```

The delay function syntax follows:

```
V(out) <+ delay(V(in), 5n);
```

Transition Operator

The `transition` operator smooths out and delays a piece-wise signal. The delay time has to be nonnegative. The rise/fall time has to be positive. The current implementation uses a default rise/fall time of 1ps if they are not user- or model-specified. The following error messages are displayed for delay, rise, and fall times that are evaluated and found to be negative:

```
DelayTransition delay cannot be smaller than zero.
Rise Transition rise time cannot be smaller than zero
Fall Transition fall time cannot be smaller than zero
```

Following is the syntax for `transition`:

```
V(out) <+ transition(V(in), 1n, 5n, 10n);
```

slew Operator

The `slew` contribution filter bounds the rate of change of the signal in its argument. The maximum positive and negative slew rate has to be positive and negative constant number during evaluation.

The following error messages are given respectively when either maximum positive or negative slew rate is evaluated negative.

- Maximum positive slew rate must be greater than zero
- Maximum negative slew rate must be smaller than zero

The slew function syntax follows:

```
V(out) <+ slew(V(in), slewrate);
```

bound_step Operator

The `bound_step` statement limits the maximum time-step of transient analysis. The step size has to be positive. The following error message is issued when a negative step is given:

```
Time step of bound_step has to be positive
```

The bound function syntax follows:

```
bound_step(1n);
```

discontinuity Operator

HSIM will ignore large voltage deviations for Verilog-A modules when discontinuity is used as shown in the following syntax:

```
discontinuity(0);
```

$STOP Operator

The `$stop` operator stops simulation from a Verilog-A module. HSIM enters into the interactive mode when $stop is activated. Following is the syntax for $stop:

```
$stop
```

$finish Operator

The `$finish` operator ends simulation from within Verilog-A modules. Following is the syntax for $finish:

```
$finish
```

Interpolation Function

$table_model

The `$table_model` function models the behavior of a system by interpolating between data points that are samples of that system's behavior. The $table_model function supports 1D, 2D and 3D tables. The syntax for the $table_model function is as follows.

```
$table_model(table_inputs, table_data_source,
   table_control_string)
```

An example of a 1D table follows:

```
$table_model(V(node1),"./voltage_samples", "L")
```

Laplace Transform Filters

Use the laplace transform function to filter the input waveform as shown in the following example:

```
V(out) <+ laplace_zp(V(in), [0,1], [1, -0.4, 0.2]);
```

There are four types of laplace transform functions:

- laplace_zp(expr,z,p): zero-pole
- laplace_zd(expr,z,d): zero-denominator
- laplace_np(expr,n,p): numerator-pole
- laplace_nd(expr,n,d): numerator-denominator

The following error message is issued when non-constant expressions are specified for zeros and poles:

```
Illegal zeros/poles assigned to laplace functions
```

Analysis

Currently, only the following three analysis functions are supported:

- analysis(“static”): Applies to DC initialization
- analysis(“ic”): Applies to DC initialization
- analysis(“tran”): Applies to Transient Analysis

I/O and Messages

$strobe, $monitor, $display, $fstrobe

The `$strobe`, `$monitor`, `$display`, and `$fstrobe` statements are used in debugging module variables during simulation. Their usage is similar to (f)printf in C language. The format string supports C language specifiers as well as the %m Verilog-A specific format converter specifier. $strobe, $monitor, and $display are treated the same. $fstrobe dumps messages to a designated file channel.

If the format string is not given, it is assumed to be “”. If there are fewer items in the output than are specified, HSIM produces and error and stops. It issues the following error message:

```
Too few items to print according to the output format
```

If there are more items than are listed in the format specifiers, a warning is issued as shown in the following example:

```
Too many items to print, omitting the extras
```

$error

`$error` prints out a message in its argument, triggers the HSIM error routine, and aborts simulation with the following error message:

```
User requested $warn() task
```

$warn

`$warn` prints out a message in its argument, triggers the HSIM Warning routine, and logs the Warning message in both the simulation log file and the console. The Warning message is prefixed with “#WARNING#”.

Multiple Inclusions of Module Files and Duplicate Declarations

A Verilog-A file is parsed only once in a given netlist. HSIM ignores multiple references to the same module.

If different modules are declared using the same name, the new definition is discarded by the parser. HSIM issues a Warning in the simulation log. The same scheme is applied to both disciplines and natures.

Macro Definitions

Macros are defined using ‘define compiler directives. The macros are expanded in the netlist according to their definition. The only restriction is that after the macro reference is replaced with a textual expansion, the description must form a valid statement or statements.

Hierarchical Instantiation

HSIM supports hierarchical instantiation of Verilog-A models and SPICE primitives inside Verilog-A models. During simulation, these instances are flattened in the netlist up to the level of the parent instance.

Example 43 shows a simple netlist that instantiates two inverter models.

Example 43

```
.param HSIMVERILOGA="inv.va"

vin ref1 gnd PWL(0 0 10n 0 11n 4 20n 4 21n 0)
x1 ref1 out inverter
x2 out out2 inverter

.tran 0.1n 30n
.printf v(*)
.ends
```

The inv.va file is shown in the following syntax sample. In the sample, the inverter module instantiates two resistors (SPICE Primitives) and one fgopamp module (Verilog-A).

```
`include "../discipline.h"
`include "../constants.h"

module inverter(in,out);
inout in,out;
electrical in, out;
electrical gnd,minus;
parameter real gain=1e5;
parameter real R=1e5;
resistor # (.r(2k)) r1(in,minus);
resistor # (.r(1K)) r2(minus,out);
fgopamp # (.gain(1e6)) op(gnd,minus,out);
endmodule


module fgopamp(,minus,out);
inout minus,out;
electrical minus,out;
parameter real gain=1e5;

analog
     V(out) <+ gain*V(,minus);
Endmodule
```

Note:

If the Verilog-A model has no behavior, but only hierarchy inside, that module will be removed during simulation. The instances of this module will be instantiated regularly. For the syntax sample above, there will be only one Verilog-A model in the circuit inventory and that will be fgopamp. The inverter model will be removed from the circuit after the devices inside are instantiated.

Using Tables for Verilog-A models

For simple Verilog-A models, HSIM has an option to use table models instead of evaluation during simulation. Using tables improves simulation speed. However, only simple models are read from the table: larger models are still evaluated.

The following parameters are used for activating table models:

- HSIMUSEVATABLE on page 159
- HSIMVATABLERANGE on page 163
- HSIMVATABLESIZE on page 163

Simulation Speed Limitations

HSIM simulation speed will be affected adversely if the design employs a large number of Verilog-A models. Designs that include a large number of transient models written in Verilog-A will run slower than simulations with built-in models.

Partitioning Verilog-A Modules

Use the following commands for partitioning:

- HSIMVAPARTITION on page 162
- HSIMVABRANCHPART on page 161

Verilog-A Interactive Commands

ev, iev

The ev and iev commands are enhanced for Verilog-A elements. Ports, port connections, and branch equation information can be accessed using these commands. An example of the ev command follows:

```
HSIM > ev xi23.xi11.xi1<0>.xi1<0>.xi1<14>.xi1
xi23.xi11.xi1<0>.xi1<0>.xi1<14>.xi1 (25145) - TERM#0:r2bl<3>
(1262), TERM#1:r2nb
l<3> (1266), TERM#2:vddm (1438), TERM#3:vddm (1438), TERM#4:vss
(1440), TERM#5:x
i23.xi11.xi1<0>.xi1<0>.xi1<14>.xi1.mem (12850),
TERM#6:xi23.xi11.xi1<0>.xi1<0>.x
i1<14>.xi1.memx (12851), TERM#7:xi23.wlr<28> (10650)
TYPE VAMOD
module name: MSCS125B0X6ZA00LL
term[0]=bl
term[1]=nbl
term[2]=vddb
term[3]=vdd
term[4]=vss
term[5]=mem
term[6]=memx
term[7]=wl
I(bl_mem)                                       val=0
               Arg 0 : V(wl_mem)                     jacob=0
               Arg 1 : V(bl_mem)                     jacob=0
I(mem_vss)                                      val=0
               Arg 0 : V(memx_vss)                   jacob=0
               Arg 1 : V(mem_vss)                    jacob=0
               Arg 2 : ddtV(mem_vss)                 jacob=0
I(mem_vdd)                                      val=-8.51593e-006
               Arg 0 : V(memx_vdd)                   jacob=1.5089e-
005
               Arg 1 : V(mem_vdd)                    jacob=1.2209e-
005
I(nbl_memx)                                     val=0
               Arg 0 : V(wl_memx)                    jacob=0
               Arg 1 : V(nbl_memx)                   jacob=0
I(memx_vss)                                     val=2.16207e-009
               Arg 0 : V(memx_vss)
jacob=3.44087e-005
               Arg 1 : V(mem_vss)
jacob=8.04181e-009
               Arg 2 : ddtV(memx_vss)                jacob=2.6e-016
I(memx_vdd)                                     val=-2.16182e-009
               Arg 0 : V(memx_vdd)                   jacob=0
               Arg 1 : V(mem_vdd)
jacob=3.88833e-007
```

nc, inc

The nc and inc commands are new etypes used with Verilog-A elements. To get a list of Verilog-A elements connected to a node, enter the following command:

```
nc vdd -etype y
```

Limitations and Unsupported Features

Unsupported features and limitations include the following:

- Only transient simulation is supported. AC analysis is not supported.
- Bus width declarations must be constant expressions and cannot be overridden from instantiating circuits.
- Numerical scheme tolerances set by reltol/abstol or specified in natures are ignored. The error controlling parameter HSIMALLOWEDDV on page 47 of the HSIM kernel overrides the setting.
- Nature statements:
  - ldt_nature
  - ddt_nature
- Discrete disciplines
- Indirect branch assignment statement
- Noise functions such as the following are ignored:
  - White noise
  - Flicker noise

18

Ferroelectric Capacitor (FeCap) Model

Describes the ferroelectric capacitor (FeCap) model, FeCap elements, and FeCap model parameter extraction.

HSIM supports the ferroelectric capacitor (FeCap) model. Ferroelectric random access memory (FRAM) is nonvolatile memory featuring fast read/write operation as low power battery-backed SRAM. FRAM eliminates the need for a battery.

Ferroelectric material is distinguished from other materials by the P-V hysteresis loop. When voltage is externally applied to a ferroelectric capacitor, sweeping from a large negative to a large positive voltage and back to the original negative voltage, the corresponding polarization (which is directly related to the charge quantify for circuit design purpose) changes from a negative value to a positive value along the lower characteristic curve and then back to the original negative value along an upper characteristic curve.

The hysteresis loop of a ferroelectric film is not fixed. It depends on several factors including:

- Film history
- Peak and frequency of the applied voltage
- Operating temperature

The hysteresis loop corresponding to the maximum peak voltage in one application is called the major hysteresis loop, while the other loops are called the sub-loops (or minor loops).

In the FRAM operation, the changes in the operating point switch among the loops during the transient period according to the switching conditions determined by the following factors:

- Ferroelectric capacitor history
- Current state
- Voltage applied

FeCap Elements

The FeCap element can be handled in HSIM as a capacitor with model parameter level=6.

Syntax

```
Caa n1 n2 model_name area=val1 p0=val2 v0=val3
```

Parameters

FeCap parameters are described in Table 85 on page 504.

Table 85 FeCap Element Parameters

Parameter	Description
Caa	Capacitor element name, which must begin with the character C.
n1	Positive node name.
n2	Negative node name.
model_name	Capacitor model name.
area	Area of the ferroelectric capacitor (units in square meter). The default value is 1.0e-12.
p0	Polarization (units in uC/cm2). The default value is 0.
v0	Initial voltage (units in volt). The default value is 0.

Example

```
c1 1 2 frmc area=3p p0=0 v0=0
```

FeCap Model

The hysteresis loops of the FeCap model from Ramtron are symmetric with respect to the origin point in a P-V plot.

Syntax

```
.model model_name c level=6 <parameter=val1>
```

Parameters

FeCap model parameters are shown in Table 86.

Table 86 FeCap Model Parameters

Parameter	Default Value	Description
vmax	5.0	Maximum voltage used in measured data
pmax	30.27	Maximum polarization at vmax in uC/cm2
kpmax	-3.47e-2	Temperature coefficient of pmax
vsat	2.0	Saturation voltage
vcr	0.5	Minimum voltage for domain switching on unpolarized cap only
as1	2.5000e-1	Curve fitting parameter for a saturated loop
bs1	0.12390e1	Curve fitting parameter for a saturated loop
cs1	0.13483e1	Curve fitting parameter for a saturated loop
as2	5.6726e-2	Curve fitting parameter for a saturated loop
bs2	2.4269e-1	Curve fitting parameter for a saturated loop
cs2	-3.8183e-2	Curve fitting parameter for a saturated loop
ds0	5.0536e-1	Curve fitting parameter for a saturated loop
au1	5.0152e-1	Curve fitting parameter for a unsaturated loop

Table 86 FeCap Model Parameters

Parameter	Default Value	Description
bu1	1.4908	Curve fitting parameter for a unsaturated loop
cu1	1.0950	Curve fitting parameter for a unsaturated loop
au2	7.4715e-2	Curve fitting parameter for a unsaturated loop
bu2	1.3517e-7	Curve fitting parameter for a unsaturated loop
cu2	-8.2528	Curve fitting parameter for a unsaturated loop
du0	2.1247e-3	Curve fitting parameter for a unsaturated loop
kas1	-1.0107e-3	Temperature coefficient for a saturated loop
kbs1	1.4579e-3	Temperature coefficient for a saturated loop
kcs1	-3.0895e-3	Temperature coefficient for a saturated loop
kas2	4.8937e-4	Temperature coefficient for a saturated loop
kbs2	6.2892e-4	Temperature coefficient for a saturated loop
kcs2	7.2631e-3	Temperature coefficient for a saturated loop
kds0	9.7153e-5	Temperature coefficient for a saturated loop
kau1	-9.5819e-4	Temperature coefficient for an unsaturated loop
kbu1	2.6618e-3	Temperature coefficient for an unsaturated loop
kcu1	-6.1257e-3	Temperature coefficient for an unsaturated loop
kau2	3.7970e-4	Temperature coefficient for an unsaturated loop
kbu2	2.7597e-4	Temperature coefficient for an unsaturated loop
kcu2	1.0525	Temperature coefficient for an unsaturated loop
kdu0	-3.4559e-2	Temperature coefficient for an unsaturated loop

Example

```
.model frmc c level=6
```

FeCap for Model Parameter Extraction

There are 18 parameters in the ferroelectric capacitor model, FeCap obtained from curve fitting of a saturated loop or an unsaturated loop as shown in Table on page 507.

FeCap Model Parameters from Curve-fitting Saturated & Unsaturated Loops

Table 87

Saturated Loop	Unsaturated Loop
as1	au1
bs1	bu1
cs1	cu1
as2	au2
bs2	bu2
cs2	cu2
ds0	du0

Each of these parameters has temperature coefficients to estimate its variation with temperature. Vmax is the maximum voltage in the parameter extraction measurement. Usually, Vmax is the maximum V_{DD} in the application of the film.

Pmax is the polarization when Vmax is applied on the ferroelectric capacitor.

Vsat is the criteria voltage to distinguish saturated and unsaturated loops. When a peak voltage applied on the capacitor is higher than Vsat, the parameters for saturated loops will be used; otherwise, the parameters for unsaturated loops will be used. Vsat should be larger than the coercive voltage of the film. The coercive voltage is defined as the voltage corresponding to the zero polarization point when the applied voltage sweeps from -Vmax to Vmax.

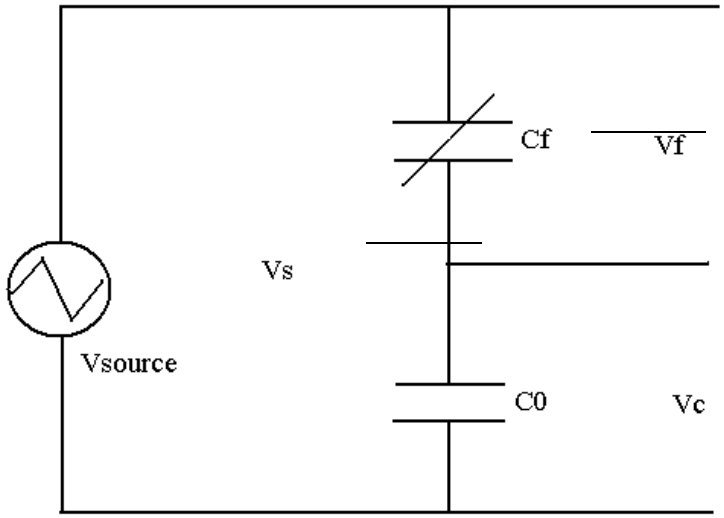
Vcr is the criteria voltage for domain switching. When a voltage applied on an unpolarized ferroelectric capacitor is lower than Vcr, we assume no domain

switching and the capacitor is a linear capacitor. When the voltage is higher than Vcr, domain switching starts and the P-V curve exhibits hysteresis loops. Usually, Vcr is about 0.5V for 5V films. Vcr should be less than the coercive voltage of the film.

Measuring Ferroelectric Hysteresis Loops

All model parameters are extracted from hysteresis loops that are measured with a Sawyer-tower circuit, as shown in Figure 20 on page 508.

Figure 20 Sawyer-Tower Circuit for Hysteresis Loop Measurement



In Figure 20 on page 508:

- Vsource is a function generator that generates either a sinusoidal wave or a triangular wave. Typically, a lower frequency such as 2KHz is used.
- C0 is a high precision linear capacitor.
- Cf is a ferroelectric capacitor.

C0 is usually chosen to be much larger than Cf so that most of Vs will be dropped on Cf and the hysteresis loops can be conveniently monitored with an oscilloscope.

In a ferroelectric film with an area of 2500um^2, the maximum value of Cf is about 500pF. We choose C0 to be about 5000pF. In the measurement, Vs and Vc are sampled and stored. Vf is obtained from Vs - Vc.

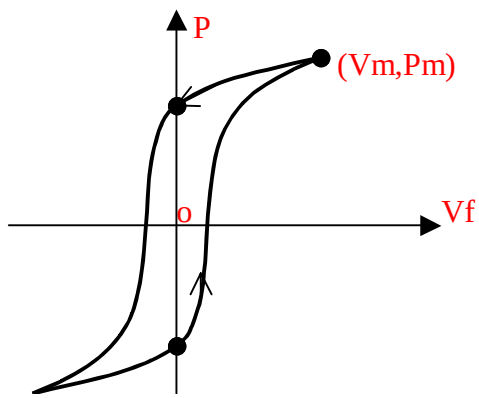
The polarization in the ferroelectric capacitor is calculated using the following equation:

```
P=C0 × Vc/Af
```

where `Af` is the area of the ferroelectric capacitor.

If a plot of P vs. Vf, or Vc vs. Vf is conducted, a loop is observed as Vf goes from -Vmax to Vmax and then returns to -Vmax. A typical ferroelectric hysteresis loop is shown in Figure 21 on page 509.

Figure 21 Ferroelectric Hysteresis Loop Example



Two hysteresis loops are used for parameter extraction. One is a saturated loop which corresponds to Vmax=VDDmax, where VDDmax is the maximum source voltage in the application. The other one is an unsaturated loop, which corresponds to Vmax @ $V_{DD}/2$. Actually, Vmax should be slightly larger than the coercive voltage of the ferroelectric film in this case. An unsaturated loop is used to obtain parameters, au1, bu1, cu1, au2, bu2, cu2, and du0.

For a 5V film, Vmax=5.5V. A saturated loop is used to obtain parameters, `as1`, `bs1`, `cs1`, `as2`, `bs2`, `cs2` and `ds0`.

FeCap Model Parameter Extraction

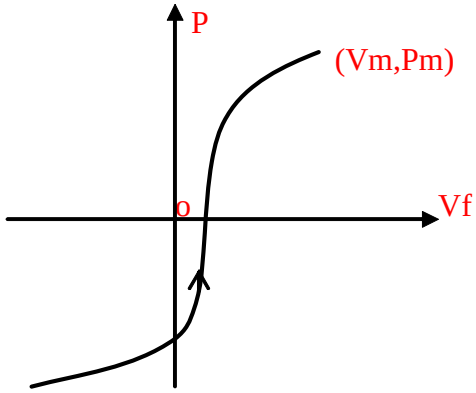
The saturated parameters, `as1`, `bs1`, `cs1`, `as2`, `bs2`, `cs2` and `ds0`, are extracted from a saturated hysteresis loop.

The steps are as follows:

1. Obtain a saturated hysteresis loop from measurement data.

2. Remove the upper-branch of the hysteresis loop. Since the hysteresis loop is assumed to be symmetric about the origin, only the lower-branch is needed, as shown in Figure 22 on page 510.

Figure 22 Lower Branch of a Hysteresis Loop

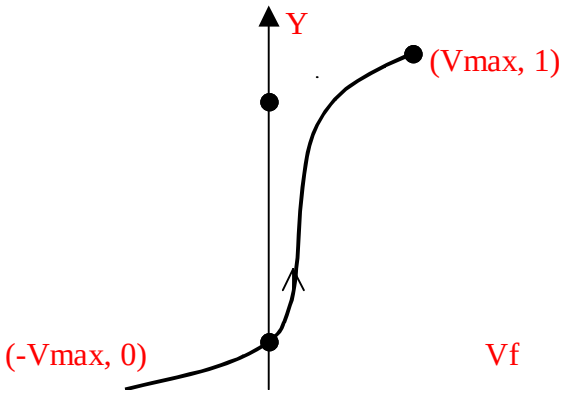


3. Normalize the lower branch of the hysteresis loop, as shown in Figure 23 on page 510. For each data point (V, P), it is normalized by:

```
Y=(P - Pmin)/(Pmax - Pmin)
```

where Pmax and Pmin are the maximum and the minimum values of P between -Vmax and Vmax.

Figure 23 Normalized Lower Branch of a Hysteresis Loop



4. Curve-fit the normalized lower-branch to the following equation to obtain the parameters, a1, b1, c1, a2, b2, c2 and d0, which are the values of `as1`, `bs1`, `cs1`, `as2`, `bs2`, `cs2` and `ds0`, respectively.

```
Y=d0 + (a1/b1)× arctan(b1(x-c1)) + (a2/b2)× arctan(b2(x-c2))
```

The initial values for curve fitting are: c1=Vsat/2, and c2=2*Vsat.

The steps for extracting the unsaturated parameters au1, bu1, cu1, au2, bu2, cu2, and du0 are the same, except when an unsaturated hysteresis loop is used.

FeCap Model Temperature Coefficients

Saturated and unsaturated hysteresis loops are measured at different temperatures. The parameters, Pmax, `as1`, `bs1`, `cs1`, `as2`, `bs2`, `cs2`, `ds0`, au1, bu1, cu1, au2, bu2, cu2, and du0, are extracted for each temperature point. Then, temperature coefficients for each of these parameters are obtained by linear fitting.

The temperature coefficient `kas1` of `as1` is obtained by curve-fitting the following equation:

```
as1@T=as1@25C + kas1 × (T - 25)
```

The temperature coefficient for Pmax is obtained from the saturated Pmax vs. Temperature measurements.

19

User Model Interface (UMI)

Provides information on HSIM proprietary MOSFET models. It describes how to implement user-defined models in a dynamic library file using the user model interface (UMI).

HSIM allows the use of proprietary MOSFET models. While built-in device models are included with HSIM, a user-defined model must be described in a dynamic library file.

UMI Models

The user model interface (UMI) models are specified with the model parameter level. The levels used in the examples are as follows:

- Level 102 MOSFET Level 2 Model for Meyer capacitances illustration purpose
- Level 108 BSIM3v3 model for charge-based capacitances illustration purpose

  Note:

  The level of the user-defined MOSFET model must be between 100 and 200.

In the example, model b3 will be compiled. The files below are required.

- UMI.h
- UMI.c
- b3defs.h
- b3assigngeometry.c
- b3load.c

- b3main.c
- b3readmodel.c
- b3set.c
- b3temp.c

The corresponding files are described in the following sections.

Building a Dynamic Library

This section describes how to build the appropriate dynamic library for one or multiple models.

The file buildfmod, located in $HSIM_HOME/bin, can be used to compile the user-defined model.

The one-line command to compile the model m115 follows:

```
%buildfmod user.so UMI.c   m115assigngeometry.c   m115load.c
m115main.c   m115readmodel.c   m115set.c   m115temp.c
```

This command compiles the C files into the dynamic library file user.so.

If two models b3 and m2 are compiled together, then the one-line command is as follows:

```
%buildfmod user.so UMI.c  b3assigngeometry.c  b3load.c  b3main.c
b3readmodel.c  b3set.c  b3temp.c m2assigngeometry.c  m2load.c
m2main.c   m2readmodel.c   m2set.c   m2temp.c
```

User Files

Header File UMI.h

Caution!

Do not alter the file UMI.h.

The header file UMI.h defines the communication protocol between HSIM and the dynamic library user.so. The initialize() function is first called by HSIM. The TEMPS structure includes two elements: tnom for the nominal temperature and temp for the simulation temperature. The UMI_VAR struct contains elements to pass information between HSIM and the dynamic library user.so. The biasing

voltages, vds, vgs, vbs, and device mode are sent from HSIM to the dynamic library.

The calculated results are returned from the dynamic library user.so to HSIM. The calculated results can include threshold voltage von, saturation voltage vdsat, drain current ids, output conductance gds, transconductance gm, gate-source overlap capacitance cgso, gate-drain overlap capacitance cgdo, gate-bulk overlap capacitance cgbo, substrate-source junction diode current ibs, substrate-drain junction diode current ibd, substrate-source junction diode capacitance capbs, and substrate-drain junction diode capacitance capbd.

If charge-based capacitance model equations are used, the returned capop value shall be 13 and the following can also be returned: gate charge qg, drain charge qd, source charge qs, and capacitance values cggb, cgdb, cgsb, cbgb, cbdb, cbsb, cdgb, cddb, and cdsb.

If the Meyer capacitance model equations are used, the returned capop value shall be 0, and positive gate-source capacitance capgs, gate-drain capacitance capgd, and gate-bulk capacitance capgb are returned.

Note:

The cgso and cgdo capacitances are multiplied by the channel width. The cgbo capacitance is multiplied by the channel length.

The USER_MOSDEF struct contains information to assist HSIM to allocate adequate amount of memory space. It also contains function names for HSIM to call. The pModel address helps link the model struct specified in the dynamic library user.so back to HSIM. The value modelsize notifies HSIM about the required memory size for the particular model struct. The value instsize notifies HSIM the required memory size for the transistor instance struct.

In addition to initialize() function, HSIM calls other functions such as:

- initial_model()
- initial_geometry()
- read_model()
- assign_geometry()
- set_model()
- temp_geometry()
- model_load()

Two functions are optional:

- start()
- conclude()

The UMI.h file shall be kept intact without any modification because a copy exists inside HSIM to facilitate data communication between HSIM and the dynamic library user.so. Content of the UMI.h file is listed below.

Note:

Contact a Synopsys Application Engineer for updates to the content of the UMI.h file.

Example 44 UMI.h File Example

```
UMI.h
/*********************************************************/
/*    Copyright 1998 - 2004*/
/*    Synopsys Corporation    */
/*                        */
/* Module      : UMI.h   */
/* Last Update : Jan.2004*/
/* Description : USER Model Interface struct*/
/*                        */
/* DO NOT CHANGE THIS HEADER FILE*/
#ifndef UMI_H
#define UMI_H
#if defined(WIN32)
__declspec(dllexport) char * initialize();
#else
char * Initialize();
#endif

/* USER device type */
#ifndef NMOS
#define NMOS 1
#define PMOS -1
#endif

typedef struct TEMPS {
double tnom;/* nominal temperature    */
double temp;/* simulation temperature */
} TEMPS;

/* USER MODEL interface variables */
typedef struct UMI_VAR {
/* device input formation */
int    mode;/* device mode */
double vds;/* vds bias */
double vgs;/* vgs bias */
double vbs;/* vbs bias */

double von;/* threshold voltage (i.e., turn-on
            voltage) */
double vdsat;/* saturation voltage */
double ids;/* drain dc current   */
double gds;/* output conductance (dIds/dVds) */
double gm;/* transconductance (dIds/dVgs)   */

double cgso; /* gate-source overlap capacitance
                        (already multiplied by width) */
double cgdo;/* gate-drain overlap capacitance
```

```
                          (already multiplied by width) */
double cgbo;/* gate-bulk overlap capacitance
                          (already multiplied by length) */

int    capop;/* capacitor selector */
          /* capop can have following values
             13: charge-based model; 0: Meyer
             model */

/* intrinsic terminal charges and trans-capacitances,
   used for capop=13 */
/* NOTE: these are intrinsic capacitance ONLY,
   not including overlap value     */
double qg;/* gate charge   */
double qd;/* drain charge */
double qs;/* source charge */
double cggb;
double cgdb;
double cgsb;
double cbgb;
double cbdb;
double cbsb;
double cdgb;
double cddb;
double cdsb;
/* next: Meyer capacitances: intrinsic capacitance + overlap
capacitance */
double capgs;/* Meyer model gate-source capacitance
             (dQg/dVgs + cgso) */
double capgd;/* Meyer model gate-drain capacitance
             (dQg/dVds + cgdo) */
double capgb;/* Meyer model gate-bulk capacitance
             (dQg/dVbs + cgbo) */ /* substrate-
             junction information */
double ibs;/* substrate-source junction diode
             current */
double ibd;/* substrate-drain junction diode
             current */
double gbs;/* substrate source
             junction-conductance */
double gbd;/* substrate drain junction
             conductance */
double capbs;/* substrate-source junction diode
             capacitance */
double capbd;/* substrate-drain junction diode
             capacitance*/
/* additional*/
double ibd;/* drain-to-bulk static current through
             CHANNEL REGION not counting substrate
```

```
            junction. Please add isub
            (substrate current)to this term */
double isb;/* source-to-bulk static current through
            CHANNEL REGION, not counting
            substrate junction */
double igd;/* gate-to-drain static current */
double igs;/* gate-to-source static current */
double igb;/* gate-to-bulk static current */
} UMI_VAR;

/* USER interface functions */
typedef struct USER_MOSDEF {
#ifdef __STDC__
char ModelName[80];
char *pModel;
int  modelsize;
int  instsize;
void (*initial_model)(char*,int);
void (*initial_geometry)(char*);
void (*read_model)(char*,char*,double);
void (*assign_geometry)(char*,char*,double);
void (*set_model)(TEMPS*,char*);
void (*temp_geometry)(TEMPS*,char*,char*);
void (*model_load)(UMI_VAR *,char*,char*);
void (*start)();/* Optional */
void   (*conclude)();/* Optional */
#else
char ModelName[80];
char *pModel;
int  modelsize;
int  instsize;
void (*initial_model)();
void (*initial_geometry)();
void (*read_model)();
void (*assign_geometry)();
void (*set_model)();
void (*temp_geometry)();
void (*model_load)();
void (*start)();/* Optional */
void (*conclude)();/* Optional */
#endif
} USER_MOSDEF;

#ifndef NULL
#define NULL 0
#endif
#endif /* end of UMI.h file */
/*********************************************************/
```

Interface File UMI.c

The interface file UMI.c contains one interface function initialize(). It is the first function inside the dynamic library user.so that is called by HSIM. HSIM sends two pieces of information to this function: type to be 1 for n-MOSFET and –1 for p-MOSFET, and level between 100 and 200. UMI.c returns the address of the device model if it finds a corresponding one with the same model level number. It can support multiple user-defined models. In addition, the initialize() function also calls the initial_model() function to grab the memory space for the matched model and also to store the device type value. An example of the UMI.c file is listed below.

Note:

Boldface text identifies code that must not to be changed.

Example 45 UMI.c File Example

```
/*********************************************************/
/* Copyright 1998 - 2004*/
/* Synopsys Corporation*/
/* Module      : UMI.c*/
/* Last Update : Jan.2004*/
/* Description : General Interface function of user-defined
model*/
/*
** Sign Convention for MOSFET Current:
** -------------------------------------------------------
** positive current direction:
** channel current : from drain to source terminal
** junction-diode current : from bulk to source/drain
**
** At the bias conditions for characterization
** NMOS : vds >=0,then ids >=0
**        vbd, vbs < 0,then ibs <=0 & ibd <=0

** PMOS : vds <=0,then ids <=0
**        vbd, vbs > 0,then ibs >=0 & ibd >=0
**
** -------------------------------------------------------
** derivatives are defined as:
**         gm   is derivative of ids w.r.t. vgs
**         gds is derivative of ids w.r.t. vds
**
** Meyer model capacitances (for capop=0) and p-n junction
** capacitances always have positive values for both NMOS
** and PMOS:
** -------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"

extern USER_MOSDEF *p_m2;
extern USER_MOSDEF *p_b3;

char *
#ifdef __STDC__
initialize(
int   type,
int   level,
char *name)
#else
initialize(type,level,name)
```

```
int   type;
int   level;
char *name;
#endif
{
int length;
char *pDevice=NULL;

/* select model level */

switch(level) {
/* you can add multiple model levels here   */
case 102:    /* "MOS Level-2 Model" */
pDevice=(char *)p_m2;
break;
case 108:    /* "BSIM3 Model" */
pDevice=(char *)p_b3;
          break;
default:
    (void)fprintf(stderr,"\n[Error]: level %d model not
    supported\n",    level);
return NULL;
}

length=strlen(name);

if(length < 79)
strcpy(((USER_MOSDEF*)pDevice)->ModelName, name);
else {
strncpy(((USER_MOSDEF*)pDevice)->ModelName, name, 79);
((USER_MOSDEF*)pDevice)->ModelName[79]=
(char)NULL;
}
((USER_MOSDEF*)pDevice)->initial_model(((USER_MOSDEF*)
               pDevice)->pModel, type);

return pDevice;
}         /* end of UMI.c file */
/*************************************************************/
```

Header File b3defs.h

The header file b3defs.h contains the struct definitions for the particular user-defined model and the corresponding instance. If a different model name is to be used, then "b3" shall be replaced by the new name. A small example is listed below.

Example 46 b3defs.h Header File Example

```
/*********************************************************/
#ifndef B3defs_h
#define B3defs_h

/* declaration for MOSFETs */
/* information needed for each instance */

typedef struct B3instance {
/* add the actual content at this location */

} B3instance;

typedef struct B3model {
/* add the actual content at this location */
} B3model;

#endif /* end of b3defs.h file */
/*********************************************************/
```

Primary File for Model b3main.c

This is the primary source file for a particular user-defined model. It contains the actual implementation of the b3_initialmodel() function (which corresponds to initial_model() function described in UMI.h header file), the b3_initialgeometry() function (which corresponds to initial_geometry() function), the optional b3_start() function (which corresponds to start() function) and the optional b3_conclude() function (which corresponds to conclude() function). Content of the b3main.c file is listed below. Boldface is used to identify words that are not to be changed. Comments are printed at small font size.

Example 47 b3main.c File Example

```
/*********************************************************/
/*                              */
/*    Copyright 1998 - 2004     */
/*    Synopsys Corporation         */
/*                              */
/* Module      : b3main.c      */
/* Last Update : Jan.2004     */
/* Description : Unique Interface function for BSIM3 model*/

#include <stdio.h>
#include <string.h>
#include <memory.h>
#include "UMI.h"
#include "m2defs.h"

#ifdef __STDC__
void b3_initialmodel(char*,int,int);
void b3_initialgeometry (char*);
void b3_read_model(char*, char*, double);
void b3_assign_geometry(char*, char*, double);
void b3_set_model(TEMPS*, char*);
void b3_temp_geometry(TEMPS*, char*, char*);
void b3_model_load(UMI_VAR*, char*, char*);
void b3_start();
void b3_conclude();
extern void b3_readmodel(B3model*, char*, double);
extern void b3_assigngeometry(B3instance*, char*, double);
extern void b3_setmodel(TEMPS*,B3model*);
extern void b3_tempgeometry(TEMPS*,B3model*,B3instance*);
extern void b3_load(UMI_VAR*,B3model*,B3instance*);
#else
void b3_initialmodel();
void b3_initialgeometry();
void b3_read_model();
void b3_assign_geometry();
void b3_set_model();
void b3_temp_geometry();
void b3_model_load();
void b3_start();
void b3_conclude();
extern void b3_readmodel();
extern void b3_assigngeometry();
extern void b3_setmodel();
extern void b3_tempgeometry();
extern void b3_load();
#endif
```

```
/* local */
static B3model    _B3Model;

static USER_MOSDEF bsim3def={
 (char*)&_B3Model,
 sizeof(B3model),
 sizeof(B3instance),
 b3_initialmodel,
 b3_initialgeometry,
 b3_read_model,
 b3_assign_geometry,
 b3_set_model,
 b3_temp_geometry,
 b3_model_load,
 b3_start,       /* Optional, replace with NULL if not used */
 b3_conclude,    /* Optional, replace with NULL if not used */
 };

USER_MOSDEF *p_b3=&bsim3def;
/*** PARTICULAR MODEL INTERFACE SUBROUTINE ***/

void
#ifdef __STDC__
b3_initialmodel(
char *model,
int   type)
#else
b3_initialmodel(model, type)
char *model;
int   type;
#endif
{
/* initialization */
(void)memset(model, 0, sizeof(B3model));
if(type==-1) {
((B3model*)model)->B3type=-1;
((B3model*)model)->B3typeGiven=1;
}
return;
}

void
#ifdef __STDC__
b3_initialgeometry(
char *here)
#else
b3_initialgeometry(here)
char *here;
```

```
#endif
{
(void)memset(here, 0, sizeof(B3instance));
return;
}

void
#ifdef __STDC__
b3_read_model(
char   *model,
char   *name,
double value)
#else
b3_read_model(model,name,value)
char   *model;
char   *name;
double value;
#endif
{
b3_readmodel((B3model*)model,name,value);
return;
}

void
#ifdef __STDC__
b3_assign_geometry(
char   *here,
char   *name,
double value)
#else
b3_assign_geometry(here,name,value)
char   *here;
char   *name;
double value;
#endif
{
b3_assigngeometry((B3instance*)here,name,value);
return;
}

void
#ifdef __STDC__
b3_set_model(
TEMPS *ckt_temp,
char   *model)
#else
b3_set_model(ckt_temp,model)
TEMPS *ckt_temp;
```

```
char   *model;
#endif
{
      b3_setmodel(ckt_temp,(B3model*)model);
      return;
}

void
#ifdef __STDC__
b3_temp_geometry(
TEMPS   *ckt_temp,
char    *model,
char    *here)
#else
b3_temp_geometry(ckt_temp,model,here)
TEMPS   *ckt_temp;
char    *model;
char    *here;
#endif
{
/* temperature-updated parameters */

b3_tempgeometry(ckt_temp,(B3model*)model,(B3instance*)here);
return;
}
void
#ifdef __STDC__
b3_model_load(
UMI_VAR *custom,
char    *model,
char    *here)
#else
b3_model_load(custom,model,here)
UMI_VAR *custom;
char    *model;
char    *here;
#endif
{
b3_load(custom,(B3model*)model,(B3instance*)here);
return;
}

void
#ifdef __STDC__
b3_start()
#else
b3_start()
#endif
```

```
{
/* Use of this function is optional */
printf("Start: Use of this function is optional.\n");
return;
}
void
#ifdef __STDC__
b3_conclude()
#else
b3_conclude()
#endif
{
/* Use of this function is optional */
printf("Conclusion for UMI: Use of this function is optional.\n");
return;
} /* end of b3main.c file */
/*************************************************************/
```

Model Parameter Processing File b3readmodel.c

This file contains the b3_readmodel() function which corresponds to the read_model() function. For each pair of the model parameter and its associated value, HSIM calls this function once. If the model parameter name is recognized by this function, the value is stored. Otherwise, this function prints a Warning message. A small example is listed below.

Example 48 b3_readmodel() File Example

```
/*****************************************************/
/*                        */
/*    Copyright 1998 - 2004*/
/*    Synopsys Corporation    */
/*                        */
/* Module      : b3readmodel.c*/
/* Last Update : Jan.2004*/
/* Description : get BSIM3 Model parameters*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_readmodel(
B3model *model,
char    *pname,
double value)
#else
b3_readmodel(model,pname,value)
B3model *model;
char    *pname;
double value;
#endif
{
/* process every model parameter value specified by the user */
   return;
} /* end of b3readmodel.c file */
/*****************************************************/
```

Model Default Value Setting File b3setmodel.c

This file contains the b3_setmodel() function which corresponds to the set_model() function in the UMI.h header file. HSIM calls this function to assign default value to a model parameter if you do not provide input value for that particular model parameter. A small example is listed below.

Example 49 b3_setmodel() File Example

```
/*********************************************************/
/*    Copyright 1998 - 2004     */
/*    Synopsys Corporation        */
/*                            */
/* Module      : b3set.c      */
/* Last Update : Jan.2004     */
/* Description : assign default BSIM3 model     */
/*               parameter values     */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_setmodel(
TEMPS   *ckt_temp,
B3model *model)
#else
b3_setmodel(ckt_temp,model)
TEMPS   *ckt_temp;
B3model *model;
#endif
{
/* assign default values to those model parameters
   that do not have input values      */

return;
} /* end of b3setmodel        */
/*********************************************************/
```

Geometry Assignment File b3assigngeometry.c

This file contains the b3_assigngeometry() function which corresponds to the assign_geometry() function in the UMI.h header file. The geometric information about a transistor instance includes width w, length l, drain-diode area ad, source-diode area as, drain-diode periphery pd., and source-diode periphery ps. To pass each piece of geometric information into the dynamic library user.so, HSIM will call this function once. A small example is listed below.

Example 50 b3_assigngeometry() File Example

```
/*********************************************************/
/*    Copyright 1998 - 2004     */
/*    Synopsys Corporation        */
/*                            */
/* Module      : b3assigngeometry.c     */
/* Last Update : Jan.2004     */
/* Description : assign BSIM3 geometry parameters     */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_assigngeometry(
B3instance *here,
char *name,
double value)
#else
b3_assigngeometry(here,name,value)
B3instance *here;
char *name;
double value;
#endif
{
/* geometric information to be used in model equation
   calculation is processed at this function             */
return;
} /* end of b3_assigngeometry.c file      */
/*********************************************************/
```

Temperature Updating File b3temp.c

This file contains the b3_tempgeometry() function which corresponds to the temp_geometry() function. The information about nominal temperature tnom and simulation temperature temp are passed from HSIM. Model parameter values and transistor instance parameter values are updated for the given simulation temperature. A small example is listed below.

Example 51 b3_tempgeometry() File Example

```
/*****************************************************/
/*    Copyright 1998 - 2004*/
/*    Synopsys Corporation    */
/*                        */
/* Module      : b3temp.c*/
/* Last Update : Jan.2004*/
/* Description : temperature-updated model parameters*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_tempgeometry(
TEMPS      *ckt_temp,
B3model    *model,
B3instance *here)
#else
b3_tempgeometry(ckt_temp,model,here)
TEMPS      *ckt_temp;
B3model    *model;
B3instance *here;
#endif
{
/* temperature-related parameter values are updated at this
function for the given simulation temperature */
return;
} /* end of b3temp.c file */
/*********************************************************/
```

Model Evaluation and Load File b3load.c

The file b3load.c contains the b3_load() function which corresponds to the model_load() function in the Header File UMI.h on page 514.

Biasing voltages vds, vgs, vbs, and device mode are passed from HSIM. The calculated results are returned from the dynamic library user.so to HSIM. The results can include the following: threshold voltage von, drain current ids, output conductance gds, transconductance gm, gate-source overlap capacitance cgso, gate-drain overlap capacitance cgdo, gate-bulk overlap capacitance cgbo, substrate-source junction diode current ibs, substrate-drain

junction diode current ibd, substrate-source junction diode capacitance capbs, substrate-drain junction diode capacitance capbd.

Charge-Based and Meyer Capacitance Models

If Charge-Based Capacitance Model equations are used as described in Charge-Based Capacitance Model Example on page 533, the following can also be returned: gate charge qg, drain charge qd, source charge qs, and capacitance values cggb, cgdb, cgsb, cbgb, cbdb, cbsb, cdgb, cddb, and cdsb.

If the Meyer Capacitance Model equations are used as described in Meyer Capacitance Model Example on page 536, the returned capop value is 0, and positive gate-source capacitance capgs, gate-drain capacitance capgd, and gate-bulk capacitance capgb are also returned. The capacitances cgso and cgdo have been multiplied by the channel width, and capacitance cgbo has already been multiplied by channel length.

Charge-Based Capacitance Model Example

Content of the b3load.c follows.

Example 52 b3load.c File Example

```
/*********************************************************/
/*    Copyright 1998 - 2004*/
/*    Synopsys Corporation    */
/*                        */
/* Module      : b3load.c*/
/* Last Update : Jan.2004*/
/* Description : evaluate BSIM3 current, the derivatives and
   capacitances          */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_load(
UMI_VAR    *custom,
B3model    *model,
B3instance *here)
#else
b3_load(custom,model,here)
UMI_VAR    *custom;
B3model    *model;
B3instance *here;
#endif
{
VgsExt=custom->vgs;
VdsExt=custom->vds;
VbsExt=custom->vbs;
DevMode=custom->mode;

/* NOTE: If the model code is SPICE-like,
/* keep the following three lines in order
   to handle PMOS properly*/

vgs=model->M2type*VgsExt;
vds=model->M2type*VdsExt;
vbs=model->M2type*VbsExt;

vbd=vbs - vds;
vgd=vgs - vds;
vgb=vgs - vbs;

if (vds >=0.0) {/* normal mode */
           here->B3mode=1;
```

```
            Vds=vds;
            Vgs=vgs;
            Vbs=vbs;
}
else {/* inverse mode */
            here->B3mode=-1;
            Vds=-vds;
            Vgs=vgd;
            Vbs=vbd;
}
here->B3cbs=/* RETURN RESULT */
here->B3cbd=/* RETURN RESULT */
here->B3gds=/* RETURN RESULT */
here->B3gm=/* RETURN RESULT */
here->B3von=/* RETURN RESULT */
here->B3cd=/* RETURN RESULT */

here->B3capbs=/* RETURN RESULT */
here->B3capbd=/* RETURN RESULT */

here->B3cgso=/* RETURN RESULT */
here->B3cgdo=/* RETURN RESULT */
here->B3cgbo=/* RETURN RESULT */
here->B3qgate=/* RETURN RESULT */
here->B3gdrn=/* RETURN RESULT */
here->B3gbulk=/* RETURN RESULT */
here->B3cggb=/* RETURN RESULT */
here->B3cgdb=/* RETURN RESULT */
here->B3cgsb=/* RETURN RESULT */
here->B3cbgb=/* RETURN RESULT */
here->B3cbdb=/* RETURN RESULT */
here->B3cbsb=/* RETURN RESULT */
here->B3cdgb=/* RETURN RESULT */
here->B3cddb=/* RETURN RESULT */
here->B3cdsb=/* RETURN RESULT */

/*********************************************/
/* return result back to interface function */
/*********************************************/

custom->von=here->B3von;/* threshold voltage */
custom->vdsat=here->B3vdsat;/* saturation voltage */
custom->ids=here->B3cd;/* drain current */
custom->gds=here->B3gds;/* output conductance */
custom->gm=here->B3gm;/* transconductance */

/* overlap capacitances */
custom->cgso=here->B3cgso;/* multiplied by width already */
```

```
custom->cgdo=here->B3cgdo;/* multiplied by width already */
custom->cgbo=here->B3cgbo;/* multiplied by length already */

/* capop: 0 for Meyer model;   13 for charge-based model */
custom->capop=13;
custom->qg=here->B3qgate;/* gate charge */
custom->qd=here->B3qdrn;/* drain charge */
custom->qs=-(here->B3qgate+here->qdrn+here->B3qbulk);         /
* source charge */
custom->cggb=here->B3cggb;
custom->cgdb=here->B3cgdb;
custom->cgsb=here->B3cgsb;
custom->cbgb=here->B3cbgb;
custom->cbdb=here->B3cbdb;
custom->cbsb=here->B3cbsb;
custom->cdgb=here->B3cdgb;
custom->cddb=here->B3cddb;
custom->cdsb=here->B3cdsb;

/* substrate diode */
custom->ibs=here->B3cbs;/* source-substrate diode current*/
custom->ibd=here->B3cbd;/* drain-substrate diode current */
custom->capbs=here->B3capbs;/* source-substrate diode
                                    capacitance*/
custom->capbd=here->B3capbd;/* drain-substrate diode
                                    capacitance */
custom->idb=
custom->isb=
custom->igd=
custom->igs=
custom->igb=
return;
} /* end of b3load.c file */
/*************************************************************/
```

Meyer Capacitance Model Example

The last portion of the file m2_load() is listed below and includes the HSIM return values.

Example 53 m2_load() File Example

```
/*******************************************/
/* return result back to interface function */
/*******************************************/

custom->von=here->M2von;/* threshold voltage */
custom->vdsat=here->M2vdsat;/* saturation voltage */
custom->ids=here->M2cd;/* drain current */
custom->gds=here->M2gds;/* output conductance */
custom->gm=here->M2gm;/* transconductance */

/* overlap capacitances */
custom->cgso=here->M2cgso;/* multiplied by width already */
custom->cgdo=here->M2cgdo;/* multiplied by width
                  already */
custom->cgbo=here->M2cgbo;/* multiplied by length
                  already*/

/* capop: 0 for Meyer model; 13 for charge-based model */
custom->capop=0;
custom->capgs=here->M2capgs;/* Meyer model gate-source
capacitance (dQg/dVgs + cgso)*/

custom->capgd=here->M2capgd;/* Meyer model gate-drain
capacitance (dQg/dVds + cgdo)*/
custom->capgb=here->M2capgb;/* Meyer model gate-bulk capacitance
(dQg/dVbs + cgbo)*/
/* substrate diode */
custom->ibs=here->M2cbs;/* source-substrate diode
current */
custom->ibd=here->M2cbd;/* drain-substrate diode
current */
custom->capbs=here->M2capbs;/* source-substrate diode
capacitance*/
custom->capbd=here->M2capbd;/* drain-substrate diode
capacitance */
return;
} /* end of m2load.c file */
/*********************************************************/
```

20

Flash Core Cell Model

Provides data about support of flash and MRAM core cells. Flash is a non-volatile memory technology based on a floating-gate device. The HSIM flash core cell model permits simulation of most flash design styles including NOR, NAND structures and multi-level cells (MLC). MRAM architectures integrate magnetic devices within standard CMOS microelectronics.

Flash Cell Core Models

The HSIM flash core cell model provides an efficient way to model NMOS-based flash memory cells. This compact modeling capability lets you verify the peripheral circuitry and its routing to the array simultaneously. The verification methodology can be applied at the pre-layout and post-layout stage. The initial Vth can be set on a cell by cell basis so that the simulation can start with the memory array in a desired state (see HSIMDELVTO on page 67 and fcc on page 405 interactive mode command). An optional fifth terminal is available to model the Nwell in twin-well processes.

Flash memory designers typically have access to a MOS model for the floating-gate device. The model used could be BSIM3 or BSIM4, for example. This device accurately models loading on word and bit lines as well as Ids current, but it usually does not model the floating gate and therefore does not provide a way to simulate programming or erasing events.

The HSIM flash core cell model is an extension of the MOS model described above. It allows some level of modeling of the floating gate by continuously adjusting the threshold voltage of the device based on the conditions applied to its terminals. The programming and erasing conditions can be specified in terms of minimum or maximum terminal voltages. A built-in model is provided for the Vth change as a function of time, terminal voltages and built-in parameters. See the *Using the HSIM User Flash Interface (UFI)* application note on Solvnet for custom Vth change equations.

Floating Gate Modeling

The floating gate modeling is basic. It provides enough flexibility to model the functionality of NOR and NAND bit-cells, as well as Multi-level cells (MLC). It can be used for floating gate devices as long as the effects used for programming and erasing can be described in terms of minimum or maximum terminal voltages, which is the case for various technologies, including Channel Hot-Electron (CHE) programming and Fowler-Nordheim (F-N) tunneling. Since the model is not physics-based, there are limitations in terms of the effects that can be modeled. For example, the charge flowing from and to the floating gate is not seen on the device terminals.

Defining and Instantiating a Flash Model

The flash cell model is an extension of a base NMOS transistor model. BSIM3, BSIM4, MOS1, MOS9, MOS11 and EKV models are supported as base model for flash cell modeling extension. The base NMOS model card is defined using the .model statement (see MOSFET on page 214).

The flash extension consists of an additional set of model parameters. It is enabled when the flashlevel parameter is set to 1 or 2. See Flashlevel = 1 Parameters and Operation on page 542 and Flashlevel=2 Parameters and Operation on page 545 sections for detailed description. The additional set of parameters of the flash extension can be added directly to the base NMOS model card or defined in a separate model card using the .model statement and then added to the base NMOS model card using the .appendmodel statement using the following syntax:

```
.appendmodel source_model destination_model
```

where `source_model` is the model card to be appended to the `destination_model`.

The following example shows a flash model definition and instantiation.

```
*Base nmos model card
.model mos1 nmos level=1 vto=2.5 kp=25u gamma=0 lambda=0.1 phi=0

*Flash model extension model card
.model flash flashcell flashlevel=1

*Adding the flash extension to the base nmos model
.appendmodel flash mos1

*Device instantiation
m1 nd ng ns nb mos1 L=1u W=1u
```

Optional Nwell Terminal

For twin-well processes, you can specify the Nwell terminal using the optional B2 parameter. If the Nwell is specified using the B2 instance parameter, a simple level=1 diode is inserted between this terminal and the pwell bulk (fourth terminal). Note that this diode is only taken into account by HSIM only when HSIMBMOS or HSIMABMOS is set. See HSIMBMOS on page 57 or HSIMABMOS on page 41 .

The following example uses the optional Nwell terminal:

```
* Device instantiation

m1 nd ng ns nb mos1 L=1u W=1n B2=nb2
```

Conventions for HSIM Flash Core Cell Models

In this chapter, programming normally refers to an increase of the threshold voltage of the NMOS transistor, while erasing normally refers to a decrease of the threshold voltage. A flash device enters a program or an erase event when its terminals meet the program or erase conditions respectively. The event ends when the conditions are no longer met. Also, V(S) always refers to the lowest voltage between terminal 1 and terminal 3. Conversely, V(D) always refers to the highest voltage between terminal 1 and terminal 3. Effectively, source and drain are swapped during simulation such that these definitions remain true.

Initializing and Probing Flash Core Cell Models

You can use the delvto instance parameter available for the base model to initialize flash core cells. Alternately, you can use HSIMDELVTO on page 67.

You can plot transient waveforms for the threshold voltage and the threshold voltage change due to program and erase events using LV9 and LV0 element templates, respectively. For example:

```
print lv9(x1.m*)
* Plots Vth for mosfets in x1

print 1v0(x1.m*)
* Plots Vth shift due to program/erase for mosfets in x1
```

Flashlevel = 1 Parameters and Operation

Flashlevel =1 is a simple model that targets NOR flash technology. The programming and erasing conditions are fixed in terms of maximum or minimum voltage on the device terminals. All Vth programming and erasing behavior is controlled by built-in equations.

Program Event

A flash device enters a program event when all of the following conditions are met:

Drain: V(D) > VDPGMMIN

Gate: V(G) > VGPGMMIN

Source: V(S) < VSPGMMAX

Bulk: V(B) < VPWPGMMAX

Nwell (if B2 is defined):

- If no VNWPGM* parameter is set or VNPGMMAX is set, V(B2) < VNWPGMMAX.
- If VNWPGMMIN only is set, V(B2) > VNWPGMMIN.
- If VNWPGMMAX and VNWPGMMIN are both set, V(B2) > VNWPGMMIN.

Erase Event

An erase event occurs when all of the following conditions are met:

Drain: V(D) > VDERSMIN

Gate: V(G) < VGERSMAX

Source: V(S) > VSERSMIN

Bulk: V(B) > VPWERSMIN

Nwell (if B2 is defined):

- If no VNWERS* parameter is set or VNWERSMIN is set, V(B2) >VNWERSMIN.
- If VNWERSMAX only is set, V(B2) < VNWERSMAX.
- If both VNWERSMIN and VNWERSMAX are set, V(B2) < VNWERSMAX.

Flashlevel=1 Voltage Threshold Change

Let VTH_init be the VTH at the time a program or erase event begins. During a program event VTH increase for TPGMSTEP period of time is given by:

KPGM*(V(G)-VTPGMSAT-VTH)*(V(D)-VDPGMMIN)

This VTH increase is prorated to the time step taken by the simulator. The maximum for VTH is the minimum of:

V(G)

VTH_init + abs(VTPGMMAXSHIFT)

VTHIGH - VTPGMSAT

During an erase event the VTH decrease for TERSSTEP period of time is given by:

KERS*(VTH-V(G)-VTERSSAT)*(V(S)-VSERSMIN)

This VTH decrease is prorated to the time step taken by the simulator. The minimum for VTH is the maximum of:

V(G)

VTH_init - abs(VTERSMAXSHIFT)

VTLOW + VTERSSAT

Flashlevel=1 Parameter List and Default Values

Parameters	Default Setting
FLASHLEVEL	Must be set to 1.

Parameters	Default Setting
VDPGMMIN	3.0
VGPGMMIN	8.0
VSPGMMAX	1.0
VPWPGMMAX	1.0
VNWPGMMAX	2.0
VNWPGMMIN	No default, active when set.
VDERSMIN	7.0
VGERSMAX	-8.0
VSERSMIN	-8.0
VPWERSMIN	7.0
VNWERSMIN	7.0
VNWERSMAX	No default, active when set.
TPGMSTEP	1n
KPGM	1m
VTPGMSAT	0.0
VTPGMMAXSHIFT	5.0
VTHIGH	7.0
TERSSTEP	1n
KERS	1m
VTERSSAT	0.0
VTERSMAXSHIFT	5.0
VTLOW	0.0

Flashlevel=1 Single Cell Simulation Example

The following example shows the flashlevel=1 default behavior.

```
*signal printing section
.print lv9(m*) lv0(m*) v(*) i(*)

*nmos model definition section
.model mos1 nmos level=1 vto=2.5 kp=25u gamma=0 lambda=0.1 phi=0

*flash model definition section
.model flash flashcell flashlevel=1
.appendmodel flash mos1

*device instantiation section
m1 nd ng ns nb mos1 L=1u W=1u B2=nb2

*stimuli section
Vng ng 0
+pwl(0.5u 5.5 0.6u 9 1u 9 1.1u 5.5 1.5u 5.5 1.6u -9 2u -9 2.1u 5.5)
Vnd nd 0
+pwl(0.5u 0.9 0.6u 4.5 1u 4.5 1.1u 0.9 1.5u 0.9 1.6u -6 2u -62.1u
0.9)
Vns ns 0 pwl(1u 0 1.1u 0 1.5u 0 1.6u 8 2u 8 2.1u 0)
Vnb nb 0 pwl(0.5u -8 0.6u 1 1u 1 1.1u -8 1.5u -8 1.6u 8 2u 8 2.1u -8)
Vnb2 nb2 0 10

*simulation setup section
.tran 10n 2.5u
.param hsimsncs=0
.end
```

Flashlevel=2 Parameters and Operation

Based on the same concept as flashlevel = 1, flashlevel = 2 offers more flexibility, which makes it suitable to model NAND technology for example. The flexibility is increased by:

- Allowing the programming and erasing condition on each terminal to be a minimum or a maximum.
- Allowing the use of custom equations for the Vth change in programming and erasing conditions through the user flash interface (UFI).

This section documents the default flashlevel=2 operation, which uses a built-in equation for Vth change. See the *Using the Flash User Interface (UFI)*

application note on SolvNet for information about using custom equations for Vth change.

Flashlevel=2 Program and Erase Events

A flash device enters a program or an erase event when its terminals meet the program or erase conditions respectively. The event ends when the conditions are no longer met. In flashlevel=2, a minimum or a maximum can be defined for each terminal, but not both. If both a minimum and maximum are defined for a given terminal, HSIM errors out.

Program Event

A program event occurs when the following program conditions are met simultaneously:

Drain:

If VDPGMMIN is set

V(D) > VDPGMMIN

else

V(D) < VDPGMMAX

Gate:

If VGPGMMAX is set

V(G) < VGPGMMAX

else

V(G) > VGPGMMIN

Source:

If VSPGMMIN is set

V(S) > VSPGMMIN

else

V(S) < VSPGMMAX

Bulk:

If VPWPGMMIN is set

V(B) > VPWPGMMIN

else

V(B) < VPWPGMMAX

N-well (if B2 is defined):

If VNWPGMMIN is set

V(B2) > VNWPGMMIN

else

V(B2) < VNWPGMMAX

Erase Event

An erase event occurs when the following erase conditions are met simultaneously:

Drain:

If VDERSMAX is set

V(D) < VDERSMAX

else

V(D) > VDERSMIN

Gate:

If VGERSMIN is set

V(G) > VGERSMIN

else

V(G) < VGERSMAX

Source:

If VSERSMAX is set

V(S) < VSERSMAX

else

V(S) > VSERSMIN

Bulk:

If VPWERSMAX is set

V(B) < VPWERSMAX

else

V(B) > VPWERSMIN

Nwell (if B2 is defined):

If VNWPGMMAX is set

V(B2) < VNWPGMMAX

else

V(B2) > VNWPGMMIN

Flashlevel=2 Threshold Voltage Change

This section describes the default behavior for flashlevel=2. If the HSIMUFI parameter is defined, the Vth change equation are defined by the user. See the *Using the Flash User Interface (UFI)* application note on Solvnet for information abut using custom equations for Vth change.

Let VTH_init be the VTH at the time program or erase event begins. During a program event:

VTH increase for TPGMSTEP period of time is given by

If VDPGMMIN is set

KPGM*(V(G)-VTPGMSAT-VTH)*(V(D)-VDPGMMIN)

else

KPGM*(V(G)-VTPGMSAT-VTH)*(VDPGMMAX-V(D))

This VTH increase is prorated to the time step taken by the simulator. The maximum for VTH is the minimum of:

V(G)

VTH_init + abs(VTPGMMAXSHIFT)

VTHIGH - VTPGMSAT

During an erase event:

VTH decrease for TERSSTEP period of time is given by

If VPWERSMAX is set

KERS*(VTH-V(G)-VTERSSAT)*(VPWERSMAX-V(B))

else

KERS*(VTH-V(G)-VTERSSAT)*(V(B)-VPWERSMIN)

This VTH decrease is prorated to the time step taken by the simulator. The minimum for VTH is the maximum of:

V(G)

VTH_init - abs(VTERSMAXSHIFT)

VTLOW + VTERSSAT

Flashlevel=2 Parameter List and Default Values

Parameters	Default Setting
FLASHLEVEL	Must be set to 2.
VDPGMMIN	No default, active when set.
VDPGMMAX	3.0
VGPGMMIN	15.0
VGPGMMAX	No default, active when set.
VSPGMMIN	No default, active when set.
VSPGMMAX	1.0
VPWPGMMIN	No default, active when set.
VPWPGMMAX	1.0
VNWPGMMIN	No default, active when set.
VNWPGMMAX	1.0
VDPGMMIN	15.0
VDPGMMAX	No default, active when set.
VGPGMMIN	No default, active when set.
VGPGMMAX	1.0
VSPGMMIN	15.0
VSPGMMAX	No default, active when set.
VPWPGMMIN	15.0
VPWPGMMAX	No default, active when set.
VNWPGMMIN	15.0

Parameters	Default Setting
VNWPGMMAX	No default, active when set.
TPGMSTEP	1n
KPGM	1m
VTPGMSAT	0.0
VTPGMMAXSHIFT	5.0
VTHIGH	7.0
TERSSTEP	1n
KERS	1m
VTERSSAT	0.0
VTERSMAXSHIFT	5.0
VTLOW	0.0

MRAM Core Cell Models

MRAM architectures integrate magnetic devices within standard CMOS microelectronics. Unlike other memory technologies, MRAM data is stored as a magnetic state, rather than electrical charge. The elements are formed from two ferromagnetic plates, each of which can hold a magnetic field, separated by a thin insulating layer.

The read (or “sensing”) of the magnetic state is accomplished by measuring the electrical resistance of the cell. Due to the magnetic tunnel effect, the electrical resistance of the cell changes due to the orientation of the fields in the two magnetic plates. By measuring the current flowing through the cell, the resistance inside a cell can be determined. Typically, if the two magnetic plates have the same polarity (parallel) this is considered to mean a "0" state (or R_{MIN}), while if the two plates are of opposite polarity (anti-parallel) the resistance will be higher (R_{MAX}) meaning a "1" state.

Due to the various different MRAM architectures, MRAM core cells differ in structure, functionality and write/read cycles. There are several types of MRAM

core cells supported within HSIM. Users may choose to use one of the supported MRAM core cell models to properly define the correct functionality of their MRAM designs.

Currently, HSIM supports three MRAM core cell architectures with 0, 1, and 2 word lines:

- Spin-Torque-Transfer MRAM - MRES0

  Bidirectional symmetrical write

  Write driver has bidirectional current flow (behaves as a current source AND current sink)

  Parallelizing-direction read

- Dual 'Active' Layer MRAM - MRES1

  Contains two 'active' magnetic layers whose polarity can be switched

  Symmetrical writing current and time between "0" and "1"

  Two-cycle read

- Toggle MRAM - MRES2

  Same pulse sequence used to write "0" to "1" or "1" to "0"

  Toggle current magnetic state to the opposite state with each execution

Spin-Torque-Transfer (STT) MRAM Core Cell Model (MRES0)

Spin-torque-transfer (STT) MRAM uses spin-aligned ("polarized") electrons to directly torque the magnetic domains. Specifically, if the electrons flowing into a layer have to change their spin, this will develop a torque that will be transferred to the nearby layer.

A stream of conducting electrons moving through the fixed magnetic layer are spin polarized (that is, most of the electrons' spins become aligned to that of the fixed layer). When these spin-polarized electrons pass through the free layer, they become re-polarized. In re-polarizing, the free layer magnet experiences a torque associated with the change in angular momentum resulting from the rotation of the spins. This torque pumps enough energy to reverse the orientation of the free layer magnet.

This spin-alignment of the electrons is ultimately achieved by changing the direction of the write current requiring a bidirectional and symmetrical write current.

The magnetic tunnel junction (MTJ) or magnetic device within the STT MRAM core cell consists of two ferromagnetic layers, one free to change polarity (sense layer) and the other fixed to some predetermined polarity (reference layer).

Graphically, the STT MRAM core cell can be represented as a 2-terminal device, consisting of a bidirectional bit line (BL) shown in Figure 24.

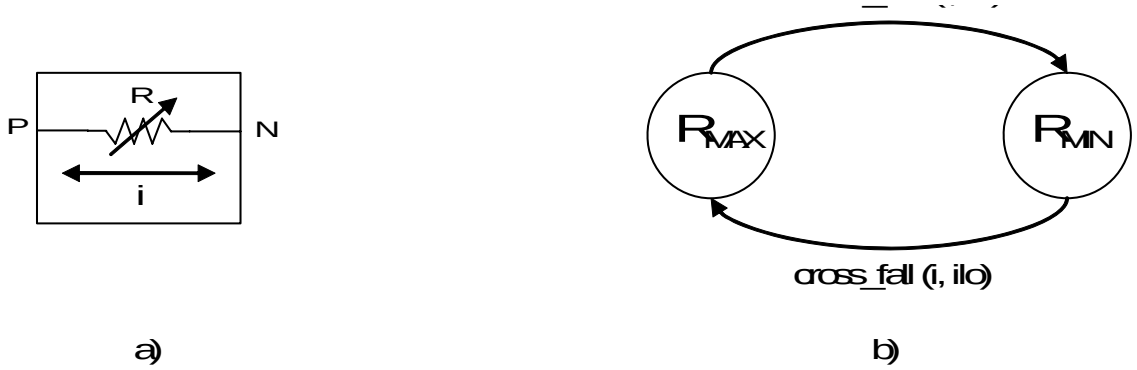


Figure 24 STT MRAM 2-terminal device (MRES0) and b) STT MRAM state sequence and current requirements

MRES0 - STT MRAM Core Cell Definition

Gxx p n MRES0

+ rmin=<val>

+ rmax=<val>

+ ihi=<val>

+ ilo=<val>

+ [ic=<val>]

+ [iwin=<val>]

MRES0 - Supported Parameter Description

rmin

minimal resistance of 'parallel' state of the MRAM cell dipoles

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipole

ihi

i upper threshold

ilo

i lower threshold

ic

initial state of MRAM cell; default=0 <Boolean> (0 ~ rmin; 1 ~ rmax

iwin

time duration window which current must be above threshold (ihi | ilo); default=0

Note:

This current threshold condition must always be followed: ihi > ilo

iwin - Check when current crosses threshold, the current remains above (or below) that threshold for a certain period of time (specified by iwin) in order to cause a switch. If the current falls before meeting the iwin criteria, the Examtime data is reset.

MRES0 Instantiation Example

```
G0 p n MRES0 rmin=1k rmax=1.2k ihi=1m ilo=-1m
```

MRES0 - STT MRAM Core Cell Functionality

```
if (cross_rise (i, ihi)) {
state = rmin ;
} elsif (cross_fall (i, ilo)) {
state = rmax ;
}
```

Limitations and Assumptions for the MRES0 Core Model

Current thresholds (ihi and ilo)

- i is bi-directional current, which can flow into or out of the MRAM cell
- The bi-directional functionality determines the state to which the cell resolves
- The current flowing may either be 'positive' (flowing in one direction) or 'negative' (flowing in the opposite direction)

  Therefore, ihi and ilo are typically close in magnitude but opposite in sign: ihi >> ilo.

Dual-Active Layer (DAL) MRAM (MRES1)

Similar to the Toggle and STT MRAM architectures, the MTJ within the Dual-Active Layer (DAL) MRAM core cell also consists of two ferromagnetic layers. However, within the DAL MRAM both magnetic layers are free to change polarity. For naming differentiation from the other architectures, they are referred to here as the soft (S) and hard (H) layers.

The word line (WL) is bidirectional, therefore the direction and magnitude of the WL current will determine the resulting polarity of the two dipoles. The S layer dipole is controlled via the WL current i1 > i1shi || i1 < i1slo. At this current value (i1shi || i1slo), the H layer dipole is not affected, as it is thicker. The H layer dipole is controlled via i1 > i1hhi || i1 < i1hlo. I1shi < i1hhi && i1slo > i1hlo, therefore changing the H layer dipole may also cause a change in the S layer dipole.

In order to determine the state of the cell or orientation of the H layer, a two-cycle read operation is required. The H layer dipole direction is determined by flipping the S layer only (i1shi < i1 < i1hhi) first in one direction (shaping the S layer dipole in a fixed orientation) and then reversing the direction if the i1 (i1hlo < i1 < i1slo) current (shaping the S layer dipole in the opposite orientation). The difference in resistance measured between the two read cycles determines the orientation of the H layer and thus the state of the cell. If the S and H layer dipoles are parallel (or aligned), the measured resistance will be lower (R_{MIN}) then if the dipoles of the S and H layers were in an anti-parallel (or unaligned) position (R_{MAX})

The model also requires a sense line (SL) running in between the two magnetic layers used to help change the polarity of the magnetic layers as well as help determine resistance shifts.

Graphically, the DAL MRAM core cell can be represented as a 4-terminal device, consisting of a bidirectional word line (WL) and a unidirectional sense line (SL) and a two-state structure (one for each magnetic layer) shown in Figure 25.

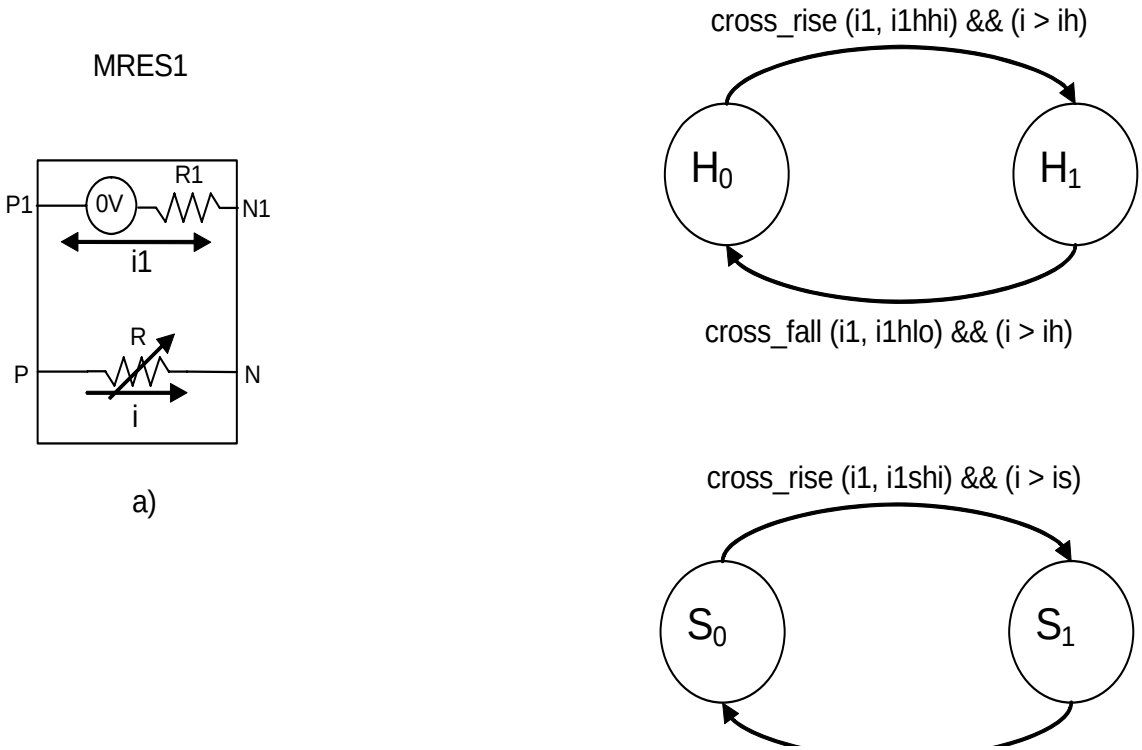


Figure 25 DAL MRAM 4-terminal device (MRES1) and DAL MRAM state sequence and current requirements

MRES1 - DAL MRAM Core Cell Definition

Gxx p n MRES1 p1 n1

+ rmin=<val>

+ rmax=<val>

+ ih=<val>

+ is=<val>

+ i1hhi=<val>

+ i1hlo=<val>

+ i1shi=<val>
+ i1slo=<val>
+ [r1=<val>]
+ [ich=<val>]
+ [ics=<val>]

MRES1 - Supported Parameter Description

min

minimal resistance of 'parallel' state of the MRAM cell dipoles

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipoles

ih

i threshold for magnetic hard (H) layer

is

i threshold for magnetic soft (S) layer

i1hhi

i1 upper threshold for magnetic hard (H) layer

i1hlo

i1 lower threshold for magnetic hard (H) layer

i1shi

i1 upper threshold for magnetic soft (S) layer

i1slo

i1 lower threshold for magnetic soft (S) layer

r1

additional internal line resistance (optional and defaults to 10mΩ ('10m'))

ich

initial state of MRAM hard layer; default=0 <Boolean>

ics

initial state of MRAM soft layer; default=0 <Boolean

Note:

These current threshold conditions must always be followed:

ih > is

i1hhi > i1shi > i1slo > i1hlo

MRES1 Instantiation Example

```
G1 p n MRES1 p1 n1  rmin=1k rmax=1.2k
+ ih=3m is=2m i1hhi=15m i1hlo=-15m i1shi=10m i1slo=-10
```

MRES1 - DAL MRAM Core Cell Functionality

```
if (cross_rise (i1, i1hhi)) {
     if (i > ih) {
          stateH = 1 ;
     }
} elsif (cross_fall (i1, i1hlo)) {
     if (i > ih) {
          stateH = 0 ;
}

if (cross_rise (i1, i1shi)) {
     if (i > is) {
          stateS = 1 ;
     }
} elsif (cross_fall (i1, i1slo)) {
     if (i > is) {
          stateS = 0 ;
}

if (stateH == stateS) {
     R = rmin ;
} elsif (stateH != stateS) {
R = rmax ;
}
```

Limitations and Assumptions for the MRES1 Core Model

Current thresholds for i1:

- i1 is bi-direction current which flow into or out of the MRAM cell
- The bi-directional functionality determines the state to which the cell resolves
- The current flowing may either be 'positive' (flowing in one direction) or 'negative' (flowing in the opposite direction)

  Therefore, ihi and ilo are typically close in magnitude but opposite in sign:

  i1hhi >> i1hlo && i1shi >> i1slo

  ih > is

  i1hhi > i1shi > i1slo > i1hlo

Toggle MRAM Core Cell Model (MRES2)

The Toggle MRAM architecture obtains its name from its use of the same pulse sequence when writing a "0" to "1" or "1" to "0" state. Each time the write current sequence is executed, the magnetic device changes from its current magnetic state to the opposite magnetic state.

Similar to the STT MRAM architecture, the MTJ within the Toggle MRAM core cell also consists of two ferromagnetic layers, one free to change polarity and the other fixed to some predetermined polarity.

By applying a current pulse sequence through two independent write lines, a rotating magnetic field is generated which moves the free, or sense magnetic layer, from one state to the other. Since the write sequence toggles the bit to its opposite state regardless of its existing state, a pre-read must be performed to determine if a write is required.

Graphically, the toggle MRAM core cell can be represented as a 6-terminal device, consisting of unidirectional write-through bit line (BL) and word lines (WL1 and WL2) shown in Figure 26.

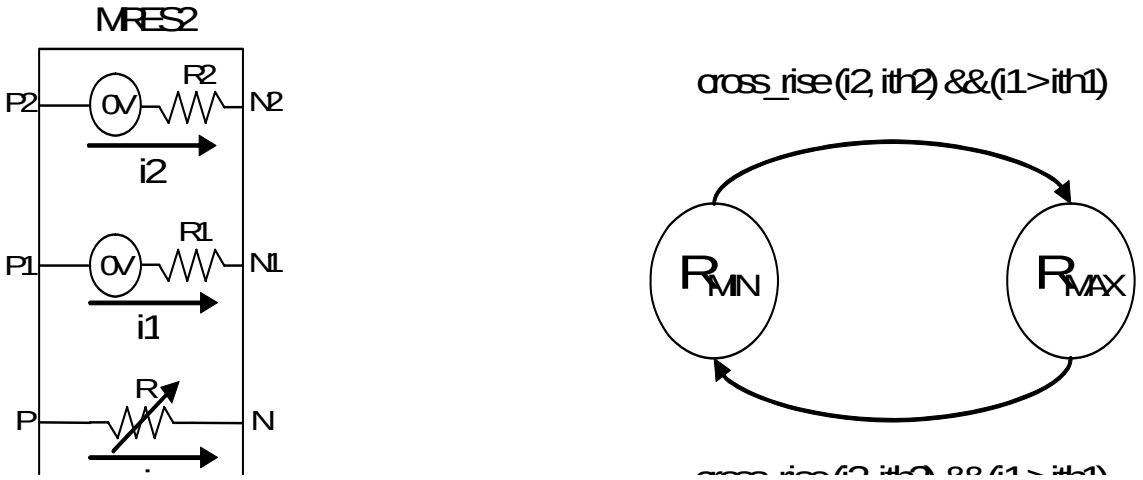


Figure 26 Toggle MRAM 6-terminal device (MRES2) and Toggle MRAM state sequence and current requirements

Based on the same symmetrical current write sequence, the state of the Toggle MRAM core cell will toggle from one state to the other. The memory cell has two states of resistance values corresponding to the direction of the magnetization in the free layer with respect to that of the fixed layer. When the two magnetic layers are parallel, the cell is in the low resistance state (R_{MIN}) and when the two magnetic layers are anti-parallel, the cell is in the high resistance state (R_{MAX}). See Figure 26.

MRES2 - Toggle MRAM Core Cell Definition

Gxx p n MRES2 p1 n1 p2 n2

+ rmin=<val>

+ rmax=<val>

+ ith1=<val>

+ ith2<val>

+ [r1=<val> r2=<val>]

+ [ic=<val>]

MRES2 - Supported Parameter Description

min

minimal resistance of 'parallel' state of the MRAM cell dipoles

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipole

ith1

i1 threshold value of current flowing through word line 1 (WL1)

ith2

i2 threshold value of current flowing through word line 2 (WL2)

r1

additional internal word line (WL1) resistance (optional and defaults to 10mΩ ('10m'))

r2

additional internal word line (WL2) resistance (optional and defaults to 10mΩ ('10m'))

ic

initial state of MRAM cell; default=0 <Boolean> (0 ~ rmin; 1 ~rmax)

MRES2 Instantiation Example

```
G2 p n MRES2 p1 n1 p2 n2 rmin=1k rmax=1.2k ith1=3m ith2=2m
```

MRES2 - Toggle MRAM Core Cell Functionality

```
if (cross_rise (i2, ith2) && (i1 > ith1)) {
     state = ~state;
}
```

Limitation of MRES Elements

To improve simulation performance, HSIM takes advantage of latent partitions and does not simulate such circuits considered latent or 'asleep.' Latency within HSIM is globally controlled using the HSIMSNCS on page 124 command. HSIM by default does, however, automatically consider any partitions to be

latent if the current of its entire boundary and internal nodes are less than HSIMSTEADYCURRENT on page 144.

MRES core cell elements are current-based voltage controlled resistors where the current across the device is sensed in order to determine the cell's state or functionality.

Therefore, if the current thresholds for the MRES cells are set to a number less than HSIMSTEADYCURRENT, HSIM will consider them latent and not simulate them.

Monitoring MRAM Array State Conditions

HSIM has the ability to monitor and report the initial state conditions and/or state transition information of any MRAM bit cells when using the MRES built-in proprietary MRAM models. See HSIMMONITORMRES on page 93.

21

C Language Functional Model

Describes HSIM functional element support written in C language and the electrical elements defined in the SPICE netlist. This chapter provides information about C language functional models. The instantiation of each C language functional model is listed in the SPICE netlist file and is defined as a regular electrical subcircuit instantiation. The chapter includes a list of API functions that perform various tasks, reference information for compiling and linking the C language functional models, and example code for A/D and D/A converters and RAM.

Model Flow Overview

A functional model written in standard C language can be compiled and linked with HSIM. This requires a C compiler in the system. The file buildfmod, in directory $HSIM_HOME/bin, is provided to compile the C functional model.

```
% buildfmod abc.so a1.c a2.c a3.c
```

`buildfmod` compiles C files a1.c, a2.c and a3.c into the library file abc.so.

HSIM supports the functional elements described in C language as well as the electrical elements defined in the SPICE netlist. While all electrical elements are defined in the SPICE netlist file, the C functional model must be described in a separate source code .c file. This chapter provides information about functional models in the C language.

- The instantiation of each C functional model is listed in SPICE netlist file and is defined as regular electrical subcircuit instantiation. Defining the model is described in Model Definition on page 564.
- A list of API functions that perform various tasks is provided in Model Interfaces on page 573.

- Compiling and linking the C functional models are described in Model Flow Overview on page 563.
- Example code for A/D and D/A converters and RAM are provided in Examples on page 607.

Windows NT/2000/XP Requirements

If using the Windows NT or Windows 2000 platform, Visual C++ must be installed and used to run the file VCVARS32.BAT, which is provided by Visual C++. This bat file will set up the correct path to use nmake as shown in the following example:

```
% nmake /f c:/hsim2.0/etc/include/buildfmod.mak SOURCES="a1.c
a2.c a3.c" LIBNAME=abc.dll
```

This one line command compiles C files a1.c, a2.c, and a3.c into the library file abc.dll. This example assumes that the name of the directory for HSIM installation in your machine is hsim2.0 in the C drive. If necessary, replace with the suitable directory name.

HP-UX Requirements

If using HP-UX, a compiler option is required to generate a model library with position-independent code.

Dynamic Library Link

The library file can be linked to the circuit netlist by adding HSIMFMODLIB on page 78 code in the netlist, as follows:

```
.param HSIMFMODLIB="./abc.so"
```

Model Definition

Each C functional model you write must be defined within an HSIM functional model interface function hsimC_model. The following model definition APIs are provided to define the model name, interface ports, and internal states.

Signal direction at the interface ports can be defined as one of the following:.

hmInput

Defines the port to be an input port such that the functional model element is driven by this input port.

hmOutput

Defines the port to be an output port such that the functional model element drives this output port.

hmBiput

Defines the port to be a bidirectional (biput) port such that the functional model element can drive the port at some time period, and can be driven by the outside driver connecting to the port at some other time period.

Each port defined by the functions listed in Model Port APIs on page 567 connects a physical node in the circuit. Each port definition is similar to the argument in a C function.

Local variables can be defined within the model function. This allows each local variable, although not present in the circuit database, to retain value after exiting the mode. Also, each local variable can be retrieved when the same model is re-evaluated later.

Local variables are defined as the state of the model function. Functionally, this is similar to the static local variable within the C function–the content is not lost after exiting from the function.

There is a difference between the state of model function and the static variable with the C function. There is only one static variable storage such that the same storage is used in each C function evaluation. In case of model state, the state storage is instance based such that each instance of the model function has its own storage of the state.

Model Creation APIs

hmCreateModel

```
pHMMODEL hmCreateModel (model_name, function_name)
```

model__name

char

function_name

pFUNC

`hmCreateModel` creates a pointer link, which has the data type of pHMMODEL, between a functional model in the netlist and a C functional model you write. The pointer serves as the ID number for any later reference to this model. To create the model, the model name defined in the netlist is given in the string model_name. The function name is specified by function_name which has a data type of pointer to a function.

Example 54

```
pRam=hmCreateModel ("ram_model", ram_function);
```

In Example 54, the pointer pRam is created to link the reference between a functional model ram_model and the C function ram_function().

hmSetModelAttr

```
void hmSetModelAttr(hdl, attr, val)
pHMMODEL hdl;
HMModelAttr attr;
int val;
```

HMNOTVOUT hmSetModelAttr defines a value val for the attribute attr on the hdl functional model. Only the HMNOTVOUT attribute is currently used. If its value is set to a non-zero number, the hdl functional model is partitioned into a block during simulation. This helps to speed up the simulation however, only functional models with a large number of I/O ports should set this attribute. It is not advisable to define this attribute for all functional models. Furthermore, hmSetModelAttr should be used immediately after a functional model is created; before port creation. In Example 55, only the pRam functional model needs be specified.

Example 55

```
hsimC_Model()
{
pHMMODEL pAdder, pRam;
pAdder=hmCreateModel ("8-bit_adder", adder8);
hmDefAnalogPort (pAdder, "C_IN", hmInput);
hmDefAnalogPortBus (pAdder, "Ain", 0, 7, hmInput);
hmDefAnalogPortBus (pAdder, "Bin", 0, 7, hmInput);
hmDefDigitalPortBus (pAdder, "Sum", 0, 7, hmOutput);
hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
hmDefAnalogPort (pAdder, "dummy", hmOutput);
pRam=hmCreateModel ("sram_512K", ram_func);
hmSetModelAttr(pRam, HMNOTVOUT,1);
hmDefAnalogPortBus (pRam, "ADDR", 256, 0, hmInput);
hmDefAnalogPortBus (pRam, "DATA", 0, 1024, hmBiput);
hmDefAnalogStateVec (pRam, "CORE", 0, 4095);
hmDefAnalogPort (pRam, "dummy", hmOutput);
}
```

Note:

HMNOTVOUT can only be applied to models with ports consisting of non-time varying output, such as voltage input ports, disabled ports, or constant ports.

Model Port APIs

hmDefAnalogPort

```
int hmDefAnalogPort (model_pointer, port_name, port_dir)
pHMMODEL  model_pointer;
char      *port_name;
HMPortDir port_dir;
```

`hmDefAnalogPort` defines an analog port terminal in a functional model referenced by the pointer model_pointer which was created earlier by hmCreateModel. The analog port implies the signal value at the port is represented by analog voltage. The port name is given in the string port_name.

The signal direction at the port is specified by port_dir which can be one of the following: hmInput, hmOutput, and hmBiput.

Example 56

```
hmDefAnalogPort (pRam, “READ”, hmInput);
hmDefAnalogPort (pRam, “WRITE”, hmInput);
hmDefAnalogPort (pRam, “DATA[0]”, hmBiput);
```

In Example 58, the analog ports `READ` and `WRITE` are input ports driving the functional model element referenced by pRam. The port `DATA[0]` can be either input or output port of the functional model element.

hmDefAnalogPortBus

```
int hmDefAnalogPortBus (model_pointer, port_bus, start, end,
port_dir)
pHMMODEL  model_pointer;
char      *port_bus;
int       start, end;
HMPortDir port_dir;
```

`hmDefAnalogPortBus` defines a group of analog port terminals in a functional model referenced by the pointer model_pointer that was created earlier by hmCreateModel. The signal value at each port is represented by analog voltage. The ports are specified by a bus representation port_bus[start:end]. The name of the port bus is specified by the string port_bus, start and end are integers representing the starting bit number and the ending bit number of the bus, respectively.

The signal direction at the port is specified by port_dir which can be one of the following: hmInput, hmOutput, and hmBiput as shown in Example 57.

Example 57

```
hmDefAnalogPortBus (pRam, “DATA”, 0, 15, hmBiput);
Each analog port DATA[i], i=0, 1, …, 15, in the analog port bus
DATA[0:15] is defined as bidirectional.
```

hmDefDigitalPort

```
int hmDefDigitalPort (model_pointer, port_name, port_dir)
pHMMODEL  model_pointer;
char      *port_name;
HMPortDir port_dir;
```

`hmDefDigitalPort` defines a digital port terminal in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The digital port implies the signal value at the port is represented by digital logic state. The port name is given in the string port_name. The signal direction at the port is specified by port_dir, which can

be one of the following: hmInput, hmOutput, and hmBiput as shown in Example 58.

Example 58

```
hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
```

The digital port C_OUT is defined as the output port of an adder referenced by the pointer pAdder.

hmDefDigitalPortBus

```
int hmDefDigitalPortBus (model_pointer, port_bus, start, end,
port_dir)
pHMMODEL  model_pointer;
char      *port_bus;
int       start, end;
HMPortDir port_dir;
```

`hmDefDigitalPortBus` defines a group of digital port terminals in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The signal value at each port is represented by digital logic state. The ports are specified by a bus representation port_bus[start:end] where the name of the port is specified by the string port_bus, start and end are integers representing the starting bit number and the ending bit number of the bus.

The signal direction at the port is specified by port_dir, which can be one of the following: hmInput, hmOutput, and hmBiput as shown in Example 59.

Example 59

```
hmDefDigitalPortBus (pAdder, "SUM", 0, 7, hmOutput);
```

Each digital port SUM[i], i=0, 1, …, 7, in the digital bus port SUM[0:7] of an adder, referenced by pAdder, is defined to be an output port.

hmDefVarAnalogPortBus

```
int hmDefVarAnalogPortBus (model_pointer, port_bus, port_dir,
size_param_name)
pHMMODEL  model_pointer;
char      *port_bus, *size_param_name;
HMPortDir port_dir;
```

`hmDefVarAnalogPortBus` is similar to `hmDefAnalogPortBus`, except the port bus size defined by `hmDefVarAnalogPortBus` is not specified when the model function is written. Instead, the size is determined after the circuit netlist

is compiled: the bus size is set to the value specified in the instantiation call to the functional model. This allows the same functional model to be instantiated by modules with different bus sizes.

For example, eneric RAM model can be referenced by 16-bit address modules and 32-bit address modules. The address bus size is not specified. If the address port addr is defined as a variable bus using `hmDefVarAnalogPortBus`, the string size_param_name defines the port bus size and the parameter name. The parameter name must be specified in the instantiation call.

```
hmDefVarAnalogPortBus (pRam, "ADDRESS", hmInput, "addr_size");
```

Each analog port ADDRESS[i], i=0, 1, …, m is defined to be an input port. The bus size m+1 is determined by the instantiation call. An example follows:

```
Xram1 …… ADDR0 ADDR1 ADDR2 ADDR3 ADDR4 ADDR5 ADDR6 ADDR7 ….
+ …….. addr_size=8
```

The analog port bus ADDRESS is defined to have a size of 8 through addr_size value in the instantiation call.

hmDefVarDigitalPortBus

```
int hmDefVarDigitalPortBus (model_pointer, port_bus, port_dir,
size_param_name)
pHMMODEL  model_pointer;
char      *port_bus, *size_param_name;
HMPortDir port_dir;
```

`hmDefVarDigitalPortBus` is similar to `hmDefDigitalPortBus` except the port bus size defined by `hmDefVarDigitalPortBus` is not specified when the model function is written. The size is determined after the circuit netlist is compiled. The port bus size is set to the value specified in the instantiation call to the functional model. This allows the same functional model to be instantiated by modules with different bus sizes as shown in the following two examples.

Examples

In a generic RAM model the address bus size is not specified. If the address port “addr” is defined as a variable bus using `hmDefVarDigitalPortBus`, the RAM model can be referenced by 16-bit address modules and also by 32-bit address modules. The string size_param_name defines the parameter name which defines the port bus size. The parameter name must be specified in the instantiation call.

```
hmDefVarDigitalPortBus (pRam, "ADDRESS", hmInput, "addr_size");
```

Each digital port ADDRESS[i], i=0, 1, …, m, is defined to be an input port. The bus size m+1 is determined by the instantiation call. An example follows.

```
Xram1 …… ADDR0 ADDR1 ADDR2 ADDR3 ADDR4 ADDR5 ADDR6 ADDR7 …
+ …… addr_size=8
```

This defines the digital port bus ADDRESS to have a size of 8 through addr_size value in the instantiation call.

hmDefCurrentPort

```
int hmDefCurrentPort (port_pointer, port_name, port_dir)
pHMMODEL  port_pointer;
char      *port _name;
HMPortDir port_dir;
```

`hmDefCurrentPort` defines a current port in a functional model referenced by the pointer port_pointer, which was created earlier by hmCreateModel. The port name is given in the string port_name. The signal direction at the port is specified by port_dir, which should be hmOutput that defines the port to be an output port for the functional model. To disable the Current Port, set the current value to 0 (zero). Current is measured in amperes. Refer to Example 60.

Example 60

```
hmDefCurrentPort (pPort, "OUT", hmOutput);
```

The current port OUT is driven by the functional model element referenced by `pPort`.

Internal State APIs

hmDefAnalogState

```
int hmDefAnalogState (model_pointer, state_name)
pHMMODEL model_pointer;
char      *state_name;
```

`hmDefAnalogState` defines an analog state variable in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The name of the state variable is specified by the string state_name.

hmDefAnalogStateVec

```
int hmDefAnalogStateVec (model_pointer, state_vec, start, end)
pHMMODEL  model_pointer;
char      *state_vec;
int       start, end;
```

`hmDefAnalogStateVec` defines a vector of analog state variables in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The state vector is represented by state_vec[start:end], where start is the starting bit of the vector and end is the ending bit of the vector.

hmDefDigitalState

```
int hmDefDigitalState (model_pointer, state_name)
pHMMODEL model_pointer;
char *state_name;
```

`hmDefDigitalState` defines a digital state variable in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The name of the state variable is specified by the string state_name.

hmDefDigitalStateVec

```
int hmDefDigitalStateVec (model_pointer, state_vec, start, end)
pHMMODEL model_pointer;
char *state_vec;
int  start, end;
```

`hmDefDigitalStateVec` defines a vector of digital state variables in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The state vector is represented by state_vec[start:end] in which start is the starting bit of the vector and end is the ending bit of the vector. Refer to Example 61.

Example 61 Model Definition Example.

```
hsimC_Model()
{
    pHMMODEL           pAdder, pRam;
    pAdder=hmCreateModel ("8-bit_adder", adder8);
    hmDefAnalogPort (pAdder, "C_IN", hmInput);
    hmDefAnalogPortBus (pAdder, "Ain", 0, 7, hmInput);
    hmDefAnalogPortBus (pAdder, "Bin", 0, 7, hmInput);
    hmDefDigitalPortBus (pAdder, "Sum", 0, 7, hmOutput);
    hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
    hmDefAnalogPort (pAdder, "dummy", hmOutput);

    pRam=hmCreateModel ("sram_512K", ram_func);
    hmDefAnalogPortBus (pRam, "ADDR", 9, 0, hmInput);
    hmDefAnalogPortBus (pRam, "DATA", 0, 15, hmBiput);
    hmDefAnalogStateVec (pRam, "CORE", 0, 4095);
    hmDefAnalogPort (pRam, "dummy", hmOutput);
}

void adder8 ()
{
..................
}
void ram_func ()
{
.......................
}
```

Model Interfaces

Simulation Interface APIs

hmSimStage

`HMPHASE hmSimStage ()`

`hmSimStage` returns a value with data type of HMPHASE which, depending on the HSIM simulation phase at which `hmSimStage` is called, can be one of the following:

- HMSTART: The HSIM simulation is in the pre-processing stage which covers the phases from parsing the netlist until the beginning of DC initialization.
- HMDCINIT: The HSIM simulation is in DC initialization stage.
- HMSIM: The HSIM simulation is in transient simulation stage.
- HMEND: HSIM will wake up every model instance at the end of simulation with an HMEND simulation phase.

It is recommended to call those API functions during the HSIMSTART stage if the API functions are needed either only once or at the beginning of simulation. Examples include:

- Memory allocation
- Static variable initialization

hmPresentTime

```
double hmPresentTime ()
```

`hmPresentTime` returns the present time of simulation. The time is represented by a double precision value in the unit of second.

hmWakeUpModel

```
int hmWakeUpModel (wake_up_time)
double    wake_up_time;
```

`hmWakeUpModel` forces an evaluation of the functional model at the specified time wake_up_time, which has the unit of second.

hmQuitSim

```
void hmQuitSim()
```

hmQuitSim interprets the simulation and enters the interactive mode after the current simulation step.

hmIntrSim

```
void hmIntrSim()
```

hmIntrSim interprets the simulation and enters the interactive mode after the current simulation step.

Name Interface APIs

hmPortName2PortId

```
int hmPortName2PortId (port_name)
char *port_name;
```

`hmPortName2PortId` returns an integer number which represents the ID number of the port specified by the string port_name. Using the ID number instead of the port name is recommended. This eliminates the time to search for the port index which is required when the port name is specified in an API function.

hmPortId2PortName

```
char *hmPortId2PortName (port_id)
int  port_id;
```

`hmPortId2PortName` returns a pointer to the string that represents the port name with the port ID number port_id.

Note:

The content of the pointer must not be modified, and the pointer must not be freed.

hmStateName2StateId

```
int hmStateName2StateId (state_name)
char *state_name;
```

`hmStateName2StateId` returns an integer number that represents the ID number of the internal state variable name specified by the string state_name. Using the ID number instead of the port name is recommended. This eliminates the time needed to search for the state index that is required when the state name is specified in the API function.

hmStateId2StateName

```
char *hmStateId2StateName (state_id)
int  state_id;
```

`hmStateId2StateName` returns a pointer to the string which represents the state variable name with the state ID number state_id.

Note:

The content of the pointer must not be modified, and the pointer must not be freed.

hmPortName2CktNodeName

```
char *hmPortName2CktNodeName (port_name)
char *port_name;
```

`hmPortName2CktNodeName` returns the full hierarchy path name of the circuit node that connects to the port specified by the string port_name.

Note:

The string returned by `hmPortName2CktNodeName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls.

hmPortId2CktNodeName

```
char *hmPortId2CktNodeName (port_id)
int  *port_id;
```

`hmPortId2CktNodeName` returns the full hierarchy path name of the circuit node that connects to the port specified by the ID number port_id.

Note:

The string returned by `hmPortId2CktNodeName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls.

hmModelInstName

```
char *hmModelInstName ()
```

`hmModelInstName` returns a pointer to the string which represents the full hierarchy path name of the functional model instance.

Note:

The string returned by `hmModelInstName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls. Refer to Example 62.

Example 62

```
hsimC_Model()
{
    pHMMODEL           pAdder, pRam;
    pAdder=hmCreateModel ("8-bit_adder", adder8);
    ..............................
}

void adder8 ()
{
    char        *model_name, *inst_name, *node1, *node2;
    ................................
    model_name=hmModelFuncName ();
                /* model_name is "8-bit_adder"     */
    inst_name=hmModelInstName ();
                /* inst_name is Xalu.Xadd3         */
    node1=hmPortName2CktNodeName ("SUM");
                /* the node which connects to SUM */
    node2=hmPortName2CktNodeName ("C_OUT");
               /* the node which connects to C_OUT */
    ...............................
}
```

Note:

The model_name will be 8-bit_adder when hmModelName is called from the model function adder8(). The instance name, however, could change in different occasions depending on which instance of 8-bit_adder invokes the function call of adder8().

hmModelFuncName

```
char *hmModelFuncName ()
```

`hmModelFuncName` returns a pointer to the string that represents the functional model name.

Note:

The content of the pointer must not be modified, and the pointer must not be freed.

Analog Port Interface APIs

hmAnalogPortValue

```
double hmAnalogPortValue (port_name)
char *port_name;
```

`hmAnalogPortValue` returns the voltage value of the analog port specified by the string port_name. The result is represented by a double precision value and has the unit of volt.

hmAnalogPortValue(“dummy”)returns the port voltage of dummy in the model adder8 if it is called within the function adder8(). hmAnalogPortValue(“dummy”) returns the port voltage of dummy in the model ram_func if it is called from the function ram_func().

hmAnalogPortValueById

```
double hmAnalogPortValueById (port_id)
int  port_id;
```

`hmAnalogPortValueById` returns the voltage value of the analog port specified by the ID number port_id. The result is represented by a double precision value and has the unit of volt.

hmAnalogPortBusValue

```
double *hmAnalogPortBusValue (port_bus, start, end)
char *port_bus;
int  start, end;
```

`hmAnalogPortBusValue` returns a pointer to an array of port voltages for a group of analog ports represented by the bus port_bus[start:end], where start is the starting bit of the bus, and end is the ending bit of the bus. Each voltage value is represented by a double precision value and has the unit of volt. Since the array space is allocated within the function hmAnalogPortBusValue, it is required to free the array within the calling function when the array is no longer needed or before exiting the calling function. Refer to Example 63.

Example 63

```
test_func ()
{
    double       *a, *b;

    a=hmAnalogPortBusValue ("addr", 4, 7);
    /* a[0]=addr[4], a[1]=addr[5], a[2]=addr[6],
       a[3]=addr[7]                */

    b=hmAnalogPortBusValue ("data", 7, 0);
    /* b[0]=data[7], b[1]=data[6], .....,
       b[6]=data[1], b[7]=data[0]   */
    ..............
    free (a);
    free (b);
}
```

hmAnalogPortBusValueById

```
double *hmAnalogPortBusValueById (port_bus_id, start, end)
int  port_bus_id;
int  start, end;
```

`hmAnalogPortBusValueById` returns a pointer to an array of port voltages for a group of analog ports represented by the bus port_bus[start:end], where the port bus name is specified by the ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. Each voltage value is represented by a double precision value and has the unit of volt. Since the array space is allocated within the function hmAnalogPortBusValueById, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmSetAnalogPortValue

```
int hmSetAnalogPortValue (port_name, voltage)
char      *port_name;
double    voltage;
```

`hmSetAnalogPortValue` sets the voltage of analog port, specified by the string port_name, to the value specified by voltage. voltage is a double precision number and has the unit of Volt.

hmSetAnalogPortValueById

```
int hmSetAnalogPortValueById (port_id, voltage)
int       port_id
double    voltage;
```

`hmSetAnalogPortValueById` sets the voltage of analog port, specified by the ID number port_id, to be the value specified by voltage, which is a double precision number and has the unit of Volt.

hmSetAnalogPortBusValue

```
int hmSetAnalogPortBusValue (port_bus, start, end, voltage_array)
char      *port_bus;
int       start, end;
double    *voltage_array;
```

`hmSetAnalogPortBusValue` sets the voltages of analog port bus, specified by port_bus[start:end], to be the array values voltage_array[0:abs(start-end)], where abs(x) represents the absolute value of x, and voltage_array is a pointer to a double array. Refer to Example 64.

Example 64

```
write_state ()
{
    double         x[16];
    .......................
    hmSetAnalogPortBusValue ("DATA", 15, 0. x);
    /* DATA[15]=x[0], DATA[14]=x[1], ........ DATA[0]=x[15] */
    .......................
}
```

hmSetAnalogPortBusValueById

```
int hmSetAnalogPortBusValueById (port_bus_id, start, end,
voltage_array)
int       port_bus_id;
int       start, end;
double    *voltage_array;
```

`hmSetAnalogPortBusValueById` sets the voltages of analog port bus, specified by port_bus[start:end], to be the array values voltage_array[0:abs(start-end)], where abs(x) represents the absolute value of x, the port bus name is specified by its ID number port_bus_id, and voltage_array is a pointer to a double array.

Digital Port Interface APIs

hmDigitalPortValue

```
HMLogic hmDigitalPortValue (port_name)
char *port_name;
```

`hmDigitalPortValue` returns the digital state of the digital port specified by the string port_name. The state value has the data type of hmLogic.

hmDigitalPortValueById

```
HMLogic hmDigitalPortValueById (port_id)
Int  port_id;
```

`hmDigitalPortValueById` returns the digital state of the digital port specified by the ID number port_id. The state value has the data type of hmLogic.

hmSetDigitalPortBusDelay

```
hmSetDigitalPortBusDelay (port_bus, start, end,
rising_delay_time_array, falling_delay_time_array)
     char *port_bus;
     int start, end;
     double *rising_delay_time_array;
     double *falling_delay_time_array;
```

hmSetDigitalPortBusDelay sets the intrinsic rising and falling delay times on the digital port bus specified by port_bus[start:end]. Delay times should be double precision arrays corresponding to port_bus[start:end] and in the unit of second.

hmSetDigitalPortBusDelayById

```
hmSetDigitalPortBusDelayById(port_bus_id, start, end,
rising_delay_time_array, falling_delay_time_array)
int port_bus_id;
int start, end;
double *rising_delay_time_array;
double *falling_delay_time_array;
```

hmSetDigitalPortBusDelayById sets the intrinsic rising and falling delay times on the digital port bus specified by port_bus_id[start:end]. Delay times should

be double precision arrays corresponding to port_bus_id[start:end] and in the unit of second.

hmDigitalPortBusValue

```
HMLogic *hmDigitalPortBusValue (port_bus, start, end)
char *port_bus;
int  start, end;
```

hmDigitalPortBusValue returns a pointer to an array of port states for a group of digital ports represented by the bus port_bus[start:end], where start is the starting bit of the bus, and end is the ending bit of the bus. Each state value has the data type of hmLogic.

Since the array space is allocated within the function hmDigitalPortBusValue, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmDigitalPortBusValueById

```
HMLogic *hmDigitalPortBusValueById (port_bus_id, start, end)
int  port_bus_id;
int  start, end;
```

hmDigitalPortBusValueById returns a pointer to an array of port states for a group of digital ports represented by the bus port_bus[start:end], where the port bus is specified by the ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. Each state value has the data type of hmLogic.

Since the array space is allocated within the function hmDigitalPortBusValue, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmSetDigitalPortDelay

```
hmSetDigitalPortDelay(port_name, rising_dalay_time,
falling_delay_time)
     char *port_name;
     double rising_dalay_time;
     double falling_delay_time;
```

hmSetDigitalPortDelay sets the intrinsic rising and falling delay times on the digital port specified by port_name. Both delay times should be double

precision numbers and in the unit of second. It should be noted that these delay times are in addition to the RC delays on the port.

hmSetDigitalPortDelayById

```
SetDigitalPortDelayById(port_id, rising_dalay_time,
falling_delay_time)
int port_id;
double rising_dalay_time;
double falling_delay_time;
```

SetDigitalPortDelayById sets the intrinsic rising and falling delay times on the digital port specified by port_id. Both delay times should be double precision numbers and in the unit of second.

hmSetDigitalPortValue

```
int hmSetDigitalPortValue (port_name, state)
char      *port_name;
HMLogic   state;
```

hmSetDigitalPortValue sets the logic state of digital port, specified by the string port_name, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalPortValueById

```
int hmSetDigitalPortValueById (port_id, state)
int       port_id;
HMLogic   state;
```

hmSetDigitalPortValueById sets the logic state of digital port, specified by the ID number port_id, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalPortBusValue

```
int hmSetDigitalPortBusValue (port_bus, start, end, state_array)
cha       *port_bus;
int       start, end;
HMLogic   *state_array;
```

hmSetDigitalPortBusValue sets the logic states of digital port bus, specified by port_bus[start:end], to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x

and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

hmSetDigitalPortBusValueById

```
int hmSetDigitalPortBusValueById (port_bus_id, start, end,
state_array)
int       port_bus_id;
int       start, end;
HMLogic   *state_array;
```

hmSetDigitalPortBusValueById sets the logic states of digital port bus, specified by port_bus[start:end], to be the array values specified by the array state_array[0:abs(start-end)], where the port bus name is specified by its ID number port_bus_id, abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

Current Port APIs

hmCurrentPortValue

```
double hmCurrentPortValue (port_name)
char *port_name;
```

hmCurrentPortValue returns the current value of the current port specified by the string port_name. It is intended to work with a port that is defined as an Analog Port. If the port direction of the Analog Port is Input or Disabled, the returned value is 0. If the port is defined as Current Port and the function is used to obtain the current value, the most recent value of the current set by the function hmSetCurrentPortValue is returned. The current value is represented by a double precision value and has the unit of ampere.

hmCurrentPortValueById

```
double hmCurrentPortValueById (port_id)
int  port_id;
```

hmCurrentPortValueById returns the current value of the current port specified by the ID number port_id. The usage is similar to hmCurrentPortValue on page 584 except for port_id.

hmSetCurrentPortValue

```
int hmSetCurrentPortValue (port_name, current)
char      *port_name;
double    current;
```

hmSetCurrentPortValue sets the electrical current value of current port, specified by the string port_name, to be the value specified by current. Current flow is a double precision number measured in Amperes as shown in Table on page 585.

Current Flow Direction

Table 88

Current Flow	Flow Direction
POSITIVE value	OUT of the functional model port
NEGATIVE value	INTO the functional model port

hmSetCurrentPortValueById

```
int hmSetCurrentPortValueById (port_id, current)
int       port_id
double    current;
```

hmSetCurrentPortValueById sets the current of current port, specified by the ID number port_id, to be the value specified by current. Current direction is shown in Table on page 585.

Port Capacitance APIs

hmPortCap

```
double hmPortCap (port_name)
char *port_name;
```

hmPortCap returns node capacitance value of the port specified by the string port_name. The capacitance value is in double precision and has the unit of Farad.

hmPortCapById

```
double hmPortCapById (port_id)
int       port_id;
```

hmPortCapById returns node capacitance value of the port specified by the ID number port_id. The capacitance value is in double precision and has the unit of Farad.

hmPortBusCap

```
double *hmPortBusCap (port_bus, start, end)
char      *port_bus;
int       start, end;
```

hmPortBusCap returns a pointer to a double array which stores the node capacitances of port_bus[start:end], where the name of the port bus is specified by the string port_bus, start is the starting bit of the bus, and end is the ending bit of the bus. The capacitance value is in double precision and has the unit of Farad.

hmPortBusCapById

```
double   *hmPortBusCapById (port_bus_id, start, end)
int       port_bus_id;
int       start, end;
```

hmPortBusCapById returns a pointer to a double array which stores the node capacitances of port_bus[start:end], where the name of the port bus is specified by its ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. The capacitance value is in double precision and has the unit of Farad.

hmSetPortCap

```
int hmSetPortCap (port_name, cap)
char      *port_name;
double    cap;
```

hmSetPortCap sets the node capacitance of the port, specified by the string port_name, to be the value cap. The capacitance value is in double precision and has the unit of Farad.

hmSetPortCapById

```
int hmSetPortCapById (port_id, cap)
int       port_id;
double    cap;
```

hmSetPortCapById sets the node capacitance of the port, specified by the ID number port_id, to be the value cap. The capacitance value is in double precision and has the unit of Farad.

hmSetPortBusCap

```
int hmSetPortBusCap (port_bus, start, end, cap_array)
char      *port_bus;
int       start, end;
double    *cap_array;
```

hmSetPortBusCap sets the node capacitance of each port in the bus port_bus[start:end] to be the value corresponding to the capacitor array element, where the name of the bus port is specified by the string port_bus, start is the starting bit of the bus and end is the ending bit of the bus. The pointer to the capacitor array is given by cap_array. The capacitance value is in double precision and has the unit of Farad.

hmSetPortBusCapById

```
int hmSetPortBusCapById (port_bus_id, start, end, cap_array)
int       port_bus_id;
int       start, end;
double    *cap_array;
```

hmSetPortBusCapById sets the node capacitance of each port in the bus port_bus[start:end] to be the value corresponding to the capacitor array element, where the name of the bus port is specified by its ID number port_bus_id, start is the starting bit of the bus and end is the ending bit of the bus. The pointer to the capacitor array is given by cap_array. The capacitance value is in double precision and has the unit of Farad. Refer to Example 65.

Example 65 Port Capacitance Usage Example

```
define_capacitance ()
{
    double     *a, b[16];

    .......................
    a=hmPortBusCap ("data", 7, 0);
    /* a[0]=capacitance at data[7]
       a[1]=capacitance at data[6]
       ............................
       a[7]=capacitance at data[0] */

    hmSetPortBusCap ("addr", 11, 8, b);
    /* capacitance at addr[11]=b[0]
       capacitance at addr[10]=b[1]
       capacitance at addr[9]=b[2]
       capacitance at addr[8]=b[3] */

    .......................
    free(a);
}
```

Enable/Disable Port APIs

hmDisablePortDrive

```
int hmDisablePortDrive (port_name)
char *port_name;
```

hmDisablePortDrive disables the output driver of a bidirectional or output port in a functional model. The port name is specified by the string port_name. After the output driver is disabled, the port direction becomes hmHiZ if the port is an output port; the port direction becomes hmInput if the port is a bidirectional port.

hmDisablePortDriveById

```
int hmDisablePortDriveById (port_id)
int  port_id;
```

hmDisablePortDriveById disables the output driver of a bidirectional or output port in a functional model. The port ID number is specified by the integer port_id. After the output driver is disabled, the port direction becomes hmHiZ if

the port is an output port; the port direction becomes hmInput if the port is a bidirectional port.

hmEnablePortDrive

```
int hmEnablePortDrive (port_name)
char *port_name;
```

hmEnablePortDrive enables the output driver to a bidirectional or output port in a functional model. The port name is specified by the string port_name.

hmEnablePortDriveById

```
int hmEnablePortDriveById (port_id)
int  port_id;
```

hmEnablePortDriveById enables the output driver to a bidirectional or output port in a functional model. The port ID number is specified by the integer port_id.

hmDisablePortBusDrive

```
int hmDisablePortBusDrive (port_bus, start, end)
char *port_bus;
int  start, end;
```

hmDisablePortBusDrive disables the output driver to each output or bidirectional port in the bus port_bus[start:end].

hmDisablePortBusDriveById

```
int hmDisablePortBusDriveById (port_bus_id, start, end)
int  port_bus_id;
int  start, end;
```

hmDisablePortBusDriveById disables the output driver to each output or bidirectional port in the bus port_name[start:end], where the ID number for port_name is port_bus_id.

hmEnablePortBusDrive

```
int hmEnablePortBusDrive (port_bus, start, end)
char *port_bus;
int  start, end;
```

hmEnablePortBusDrive enables the output driver to each output or bidirectional port in the bus port_bus[start:end].

hmEnablePortBusDriveById

```
int hmEnablePortBusDriveById (port_bus_id, start, end)
int  port_bus_id;
int  start, end;
```

hmEnablePortBusDriveById enables the output driver to each output or bidirectional port in the bus port_name[start:end], where the ID number for port_name is port_bus_id.

hmDisableAllPorts

```
int hmDisableAllPorts (port_dir)
HMPortDirport_dir;
```

hmDisableAllPorts disables the output driver to each model port which has the port direction port_dir. The direction must be either hmOutput or hmBiput.

hmEnableAllPorts

```
int hmEnableAllPorts (port_dir)
HMPortDir port_dir;
```

hmEnableAllPorts enables the output driver to each model port which has the port direction port_dir. The direction must be either hmOutput or hmBiput.

Set Port APIs

The following APIs enable model ports to be set to active or idle. These APIs include:

- hmSetPortActive
- hmSetPortActiveByID
- hmSetPortBusActive
- hmSetPortBusActiveByID
- hmSetPortIdle
- hmSetPortIdleById

- hmSetPortBusIdle
- hmSetPortBusIdleById

hmSetPortActive

```
hmSetPortActive (port_name)
char *port_name;
```

hmSetPortActive enables the model port so that the model will be evaluated when the port changes.

hmSetPortActiveByID

```
hmSetPortActiveByID (port_id)
char *port_id;
```

hmSetPortActiveByID enables a model port specified with an ID so that the model will be evaluated when the port changes.

hmSetPortBusActive

```
hmSetPortBusActive (port_name, start, end)
char *port_name;
int  start, end;
```

hmSetPortBusActive enables the model port bus so that the model will be evaluated when the port changes.

hmSetPortBusActiveByID

```
hmSetPortBusActiveByID (port_id, start, end)
char *port_id;
int  start, end;
```

hmSetPortBusActiveByID enables a model port bus specified with an ID so that the model will be evaluated when the port changes.

hmSetPortIdle

```
hmSetPortIdle (port_name)
int  port_name;
```

hmSetPortIdle disables the model port so that the model will not be evaluated when the port changes.

hmSetPortIdleByID

```
hmSetPortIdleByID (port_id)
int  port_id;
```

hmSetPortIdleByID disables a model port specified with an ID so that the model will not be evaluated when the port changes.

hmSetPortBusIdle

```
hmSetPortBusIdle (port_bus_name, start, end)
int  port_bus_name;
int  start, end;
```

hmSetPortBusIdle disables the model port bus so that the model will not be evaluated when the port changes.

hmSetPortBusIdleByID

```
hmSetPortBusIdleByID (port_bus_id, start, end)
int  port_bus_id;
int  start, end;
```

hmSetPortBusIdleByID disables a model port bus specified with an ID so that the model will not be evaluated when the port changes.

Port Delay/Strength APIs

The following four functions specify the delay time (tau) for a signal outputted by a port or ports given by one of the following:

- Name
- ID
- Bus name
- Bus ID

Implementing the delay is accomplished using an RC circuit in which R*C=tau. In addition to tau, the value of R (default is 1 Ohm) can also be specified so that the value of capacitance will be tau/R. For this implementation to be accurate, R and C should be chosen as follows:

- R should be much smaller than the loading impedance of the port.
- C should be much lager than the loading capacitance of the port.

hmSetPortDelayRes

```
int hmSetPortDelayRes (port_name, delay, res)
char      *port_name;
double    delay, res;
```

hmSetPortDelayRes sets the delay time delay and driver impedance res to the output or bidirectional port specified by the string port_name. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision value and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortDelayResByID

```
int hmSetPortDelayResByID (port_name, delay, res)
char      *port_name;
double    delay, res;
```

hmSetPortDelayResByID sets the delay time delay and driver impedance res to the output or bidirectional port specified by the string port_name. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision value and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortBusDelayRes

```
int hmSetPortBusDelayRes (port_bus, start, end, delay, res)
char      *port_bus;
int       start, end;
double    delay, res;
```

hmSetPortBusDelayRes sets the delay time delay and driver impedance res to each output or bidirectional port of the bus port_bus[start:end]. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision number and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortBusDelayResById

```
int hmSetPortBusDelayResById (port_bus_id, start, end, delay,
res)
int       port_bus_id, start, end;
double    delay, res;
```

hmSetPortBusDelayResById sets the delay time delay and driver impedance res to each output or bidirectional port of the bus port_bus[start:end], where port_bus is the name of the bus with ID number port_bus_id. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision number and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

Port Sensitivity APIs

hmSetPortEventDv

```
int hmSetPortEventDv (port_name, event_dv)
char      *port_name;
double    event_dv;
```

hmSetPortEventDv sets the event threshold voltage event_dv to the port specified by the string port_name. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage event_dv.

The value of event_dv adjusts the trade-off between performance and precision: Larger event_dv value reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortEventDvById

```
int hmSetPortEventDvById (port_id, event_dv)
int       port_id;
double    event_dv;
```

hmSetPortEventDvById sets the event threshold voltage event_dv to the port specified by the ID number of port_id. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage event_dv.

The value of event_dv adjusts the trade-off between performance and precision: Larger event_dv value reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortBusEventDv

```
int hmSetPortBusEventDv (port_bus, start, end, event_dv_array)
char      *port_bus;
int       start, end;
double    *event_dv_array;
```

hmSetPortBusEventDv sets the event threshold voltage of each port in the bus port_bus[start:end] to be the corresponding value in the array element, where event_dv_array is the pointer to the array, start is the starting bit of the bus and end is the ending bit of the bus. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage.

The value of event threshold voltage adjusts the trade-off between performance and precision: Larger event threshold voltage reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortBusEventDvById

```
int hmSetPortBusEventDvById (port_bus_id, start, end,
event_dv_array)
int       port_bus_id, start, end;
double    *event_dv_array;
```

hmSetPortBusEventDvById sets the event threshold voltage of each port in the bus port_bus[start:end] to be the corresponding value in the array element, where the bus is specified by its ID number port_bus_id, event_dv_array is the

pointer to the array, start is the starting bit of the bus and end is the ending bit of the bus. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage.

The value of event threshold voltage adjusts the trade-off between performance and precision: Larger event threshold voltage reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements. Refer to Example 66.

Example 66

```
set_event_voltage ()
{
    int        i;
    double ev[16];mfs

    for (i=0; i<16; i++)
        ev[i]=(i > 7)? 0.1: 0.05;
    hmSetPortBusEventDv ("addr", 15, 0, ev);
    /* event threshold voltage at addr[15]=ev[0]
        event threshold voltage at addr[14]=ev[1]
        ..................................................
        event threshold voltage at addr[0]=ev[15] */
    ..................
}
```

Port Change APIs

hmPortChange, hmPortChangeById

```
int hmPortChange(port_name)
char *port_name;
int hmPortChangeById(port_id)
int port_id;
```

`hmPortChange and hmPortChangeById return 1 if the digital port specified by port_name or port_id has value change since the last time this model is evaluated. This function should be used for` digital ports only. For analog ports, the functions will always return 1.

hmPortBusChange, hmPortBusChangeById

```
int hmPortBusChange(port_bus, start, end)
char *port_bus;
int start;
int end;
int hmPortBusChangeById(port_bus_id, start, end)
int port_bus_id;
int start;
int end;
```

hmPortBusChange and hmPortBusChangeById returns 1 if any of the specified group of digital ports represented by the bus port_bus[start:end] has value change since the last time this model is evaluated. For analog ports, the functions will always return 1.

hmAnyPortChange

```
int hmAnyPortChange(dir)
HMPortDir dir;
```

hmAnyPortChange returns 1 if any digital port matches the port direction specified by dir has a value change since the last time this model is evaluated. dir can be hmInput or hmBiput.

Port Bus Size APIs

hmPortBusSize

```
int hmPortBusSize (port_bus)
char *port_bus;
```

`hmPortBusSize` returns the bus size of the port specified by the string port_size.

If the bus is defined by hmDefAnalogPortBus or hmDefDigitalPortBus, the size is equal to abs(start-end+1) in which start and end are the starting and ending bit numbers of the port bus, respectively.

If the bus is defined by hmDefVarAnalogPortBus or hmDefVarDigitalPortBus, the size is equal to the parameter value specified in the instance call. The parameter name is defined by the string specified in hmDefVarAnalogPortBus or hmDefVarDigitalPortBus.

hmPortBusSizeById

```
int hmPortBusSizeById (port_id)
int  port_id;
```

`hmPortBusSizeById` returns the bus size of the port specified by its ID number port_id.

If the bus is defined by hmDefAnalogPortBus or hmDefDigitalPortBus, the size is equal to abs(start-end+1) in which start and end are the starting and ending bit numbers of the port bus, respectively.

If the bus is defined by hmDefVarAnalogPortBus or hmDefVarDigitalPortBus, then the size is equal to the parameter value specified in the instance call. The parameter name is defined by the string specified in hmDefVarAnalogPortBus or hmDefVarDigitalPortBus.

hmPortDir. hmPortDirByID

```
HMPortDir hmPortDir (port_name)
char *port_name;


HMPortDir hmPortDirById(port_id)
int port_id;
```

`hmPortDir` checks and returns the signal flow direction at the port specified by the string port_name.

`hmPortDirById` checks and returns the signal flow direction at the port specified by the ID number port_id.

The direction has the data type of `hmPortDir` or `hmPortDirById` and can be one of the following:

hmInput There are two possibilities for hmInput:

1. The port has been defined as hmInput in the function hmDefAnalogPort or hmDefDigitalPort.
2. The port has been defined as hmBiput and has also been disabled from the output driver by hmDisablePortDrive.

hmOutput There are two possibilities for hmOutput:

1. The port has been defined as hmOutput and has also been enabled to the output driver by hmEnablePortDrive;
2. The port has been defined as hmBiput and has also been enabled to the output driver by hmEnablePortDrive.

hmHiZ hmHiZ has been defined as hmOutput and the port has also been disabled from the output driver by hmDisablePortDrive.

Internal State Interface APIs: Analog State APIs

hmAnalogStateValue

```
double hmAnalogStateValue (state_name)
char *state_name;
```

`hmAnalogStateValue` returns the numerical value of the internal state variable specified by the string state_name. Double-precision value is used.

hmAnalogStateValueById

```
double hmAnalogStateValueById (state_id)
int  state_id;
```

`hmAnalogStateValueById` returns the numerical value of the internal state variable specified by the ID number state_id. Double-precision value is used.

hmAnalogStateVecValue

```
double *hmAnalogStateVecValue (state_vec, start, end)
char *state_vec;
int  start, end;
```

`hmAnalogStateVecValue` returns a pointer to an array of internal state values for a group of analog states represented by the vector state_vec[start:end], where start is the starting bit of the vector, and end is the ending bit of the vector. A double precision value is used. Since the array space is allocated within the hmAnalogStateVecValue function, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmAnalogStateVecValueById

```
double *hmAnalogStateVecValueById (state_vec_id, start, end)
int  state_vec_id;
int  start, end;
```

`hmAnalogStateVecValueById` returns a pointer to an array of internal state values for a group of analog states represented by the vector state_vec[start:end], where the state vector name is specified by the ID number state_vec_id, start is the starting bit of the vector, and end is the ending bit of the vector. A double precision value is used. Since the array space is allocated within the function hmAnalogStateVecValueById, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmSetAnalogStateValue

```
int hmSetAnalogStateValue (state_name, value)
char      *state_name;
double    value;
```

`hmSetAnalogStateValue` sets the state value of an internal state variable, specified by the string state_name, to a double precision value.

hmSetAnalogStateValueById

```
int hmSetAnalogStateValueById (state_id, value)
int       state_id;
double    value;
```

`hmSetAnalogStateValueById` sets the state value of an internal analog state variable to a double precision value. The name of the state variable is specified by its ID number state_id.

hmSetAnalogStateVecValue

```
int hmSetAnalogStateVecValue (state_vec, start, end, value_array)
char      *state_vec;
int       start, end;
double    *value_array;
```

`hmSetAnalogStateVecValue` sets the values of internal analog state vector, specified by state_vec[start:end], to be the array values value_array[0:abs(start-end)], where abs(x) represents the absolute value of x, and value_array is a pointer to a double-type array.

hmSetAnalogStateVecValueById

```
int hmSetAnalogStateVecValueById (state_vec_id, start, end,
value_array)
int       state_vec_id;
int       start, end;
double    *value_array;
```

`hmSetAnalogStateVecValueById` sets the values of internal analog state vector, specified by state_vec[start:end], to be the array values value_array[0:abs(start-end)], where abs(x) represents the absolute value of x, and value_array is a pointer to a double-type array. The vector name is specified by its ID number state_vec_id.

Internal State Interface APIs: Digital State APIs

hmDigitalStateValue

```
HMLogic hmDigitalStateValue (state_name)
char*state_name;
```

`hmDigitalStateValue` returns the digital state value of the internal state variable specified by the string state_name. The state value has the data type of hmLogic.

hmDigitalStateValueById, hmDigitalStateVecValue, hmDigitalStateVecValueById

```
HMLogic hmDigitalStateValueById (state_id)
int  state_id;


HMLogic *hmDigitalStateVecValue (state_vec, start, end)
char *state_vec;
int  start, end;


HMLogic *hmDigitalStateVecValueById (state_vec_id, start, end)
char state_vec_id;
int  start, end;
```

hmDigitalStateValueById returns the digital state value of the internal state variable specified by the ID number state_id.

hmDigitalStateVecValue returns a pointer to an array of internal state values for a group of digital states end is the ending bit of the vector.

hmDigitalStateVecValueById returns a pointer to an array of internal state values for a group of digital states represented by the vector state_vec[start:end]

,

where the state vector name is specified by its ID number state_vec_id, start is the starting bit of the vector, and end is the ending bit of the vector.

The state value has the data type of hmLogic and can be one of the following:

- hmZero: Represents logic 0 state.
- hmOne: Represents logic 1 state.
- hmU: Represents the port is in transition between logic 0 and logic 1 states.
- hmX: Represents unknown state which could be in either logic 0 or logic 1 state.
- hmZ: Implies the port is in floating or high-impedance state.
- hmZL: Implies the port is floating and also in logic 0 state.
- hmZH: Implies the port is floating and also in logic 1 state.

Since the array space is allocated within any of the functions, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmSetDigitalStateValue

```
int hmSetDigitalStateValue (state_name, state)
char      *state_name;
HMLogic   state;
```

`hmSetDigitalStateValue` sets the logic state of an internal digital state variable, specified by the string state_name, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalStateValueById

```
int hmSetDigitalStateValueById (state_id, state)
int       state_id;
HMLogic   state;
```

hmSetDigitalStateValueById sets the logic state of an internal digital state variable to be the value specified by state. The state variable name is specified by its ID number state_id. The state value has the data type of hmLogic.

hmSetDigitalStateVecValue

```
int hmSetDigitalStateVecValue (state_vec, start, end,
state_array)
char      *state_vec;
int       start, end;
HMLogic   *state_array;
```

`hmSetDigitalStateVecValue` sets the logic states of internal digital state vector state_vec[start:end] to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

hmSetDigitalStateVecValueById

```
int hmSetDigitalStateVecValueById (state_vec_id, start, end,
state_array)
int       state_vec_id;
int       start, end;
HMLogic   *state_array;
```

hmSetDigitalStateVecValueById sets the logic states of internal digital state vector state_vec[start:end] to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state vector name is specified by its ID number state_vec_id. The state value has the data type of hmLogic.

Miscellaneous Interface APIs

hmModelInstParamValue

```
char hmModelInstParamValue (param_name, param_value)
char *param_name;
double *param_value;
```

hmModelInstParamValue returns the parameter value of the specified param_name for the current model instance. If the function returns 1,

param_name is defined in the model instance and the value is returned in param_val. If the function returns 0, the parameter cannot be found in the model instance, and no value is written to param_val.

Example 67

```
* netlist
x1 1 2 3 a2d_model start=10n


/* C model */
if (hmModelInstParamValue("start", &start_val)) {
  ...
  ...
}
else {
  printf("parameter \'start\' is not defined\n");
}
```

hmModelInstStrParamValue

```
char hmModelInstStrParamValue (param_name, str_param_value)
char *param_name;
char **str_param_value;
```

hmModelInstStrParamValue returns the string parameter value of the specified param_name for the current model instance. If the function returns 1, param_name is defined in the model instance and the value is returned in str_param_val. If the function returns 0, the parameter cannot be found in the model instance, and no value is written to str_param_val. Refer to Example 68.

Example 68

```
* netlist
x1 1 2 3 a2d_model key="abc"

/* C model */
if (hmModelInstStrParamValue(key, &sptr)) {
     if (!strcmp(sptr, "abc")) {
          ...
          ...
     }
}
else {
     printf("parameter key is not defined\n");
}
```

hmSimOptValue

```
double hmSimOptValue(opt)
HMSimOpt opt;
```

hmSimOptValue returns the value of a simulation parameter specified by opt. Simulation parameter can be one of the followings:

- HMTEMP - circuit temperature
- HMSTOP - transient time

hmFree

```
void hmFree (mem_pointer)
void *mem_pointer;
```

hmFree frees the memory allocated by functions like hmDigitalPortBusValueById, hmDigitalPortBusValue which allocate memories and return them during runtime. mem_pointer specifies the memory location.

hmMsg, hmWarn, hmError

```
void hmMsg(fmt, ...)
char *fmt;
void hmWarn(fmt, ...)
char *fmt;
void hmError(fmt, ...)
char *fmt;
```

hmMsg, hmWarn and hmError are 3 functions that print messages to both the screen and the HSIM log file. The usage is similar to C library printf(). hmMsg will print the formatted message to both the screen and logoff.

hmWarn will print Warning: and then the formatted message to both the screen and log file.

hmError will print Error: and then the formatted message to both the screen and log file and abort.

Modeling Memory Core

hmDefMemCore

```
int hmDefMemCore (model_pointer, mem_name)
pHMMODEL  model_pointer;
char      *mem_name;
```

hmDefMemCore defines a memory core within the functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The name of memory core is specified by the string mem_core. `hmDefMemCore`, as the functions listed in Chapter 15, Timing and Power Analysis, Hold Time Check on page 434, can be called only from the model interface function hsimC_Model. Each core cell state has a data type of hmLogic.

hmInitMemCore

```
hmInitMemCore (mem_name, addr_size, data_size, def_state,
data_file)
char      *mem_core, *data_file, *default_state;
int       addr_size, data_size;
```

hmInitMemCore allocates and initializes the memory core specified by the string mem_name that was declared by the function hmDefMemCore. The memory size is defined by the number of address bits, addr_size, and by the number of data bits, data_size. Refer to Example .

A total of 2^{10}=1024 words of memory core cells are defined if addr_size=10. Each word is 8-bit wide if data_size=8.

The default initial state for each memory word is specified by a hexadecimal string default_state.

Each memory word has the default initial state 0xfff0 if default_state is "fff0" and assuming data_size=16. The exception is the initial state is overwritten by the bit pattern defined in the file with the filename specified by the string data_file. The format of the initial state in the file is as follows.

```
addr1      data11 data12 data13 .........
addr2      data21 data22 data23 .........
.........................................
```

This sets the initial state at address addr1 to be data11, the initial state at address addr1+1 to be data12, the initial state at address addr1+2 to be data13, and so forth. Similarly, the initial state at address addr2 is data21, and data22 at address addr2+1, data23 at address addr2+2. Each address or data value is represented as a hexadecimal number (see the next example).

If addr_size=8 and data_size=12, then the initial data file may appear as follows.

```
00 21f 59a 3be
39 8c2 3d2 107 29f
```

This initializes the data to be 21f at address 0, 59a at address 1, 3be at address 2, 8c2 at address 39, 3d2 at address 3a, 107 at address 3b, and 29f at address 3c.

hmReadMemCore

```
hmReadMemCore (mem_name, addr, data)
char *mem_name;
HMLogic*addr, *data;
```

hmReadMemCore reads the memory core, specified by the string mem_name, and sets the bit pattern at the array data to be the memory core content at address addr.

hmWriteMemCore

```
hmWriteMemCore (mem_name, addr, data)
char      *mem_name;
hmLogic   *addr, *data;
```

hmWriteMemCore writes the bit pattern at the array data into the memory core addressed by addr.

Examples

This section provides examples for the following applications:

- A/D and D/A Converter Examples on page 608
- Port Delay Example on page 612

- RAM Example on page 614
- Event Handling Example on page 618

A/D and D/A Converter Examples

The A/D and D/A converter example contains a 16-bit A/D converter and a 16-bit D/A converter. A 200-KHz sinusoidal input voltage source vin drives the A/D converter which has the 16-bit digital output bus b0, b1, ... , b15. The digital output bus from the A/D converter is applied to the 16-bit D/A converter that has the analog output port vout. The A/D conversion occurs at positive clock edge while the D/A conversion occurs at negative clock edge. The analog output voltage at vout should follow the analog input at vin.

```
* SPICE netlist file
* 16-bit A/D and D/A converters
.param HSIMFMODLIB=”./a.so”
x1 vin vref clk b0 b1 b2 b3 b4 b5 b6 b7
+ b8 b9 b10 b11 b12 b13 b14 b15 a2d size=16
x2 vout vref clk b0 b1 b2 b3 b4 b5 b6 b7
+ b8 b9 b10 b11 b12   b13 b14 b15 d2a size=16
vclk clk gnd pulse 0 3.0 0 .2n .2n 2n 5n
vin vin gnd sin(1.5 1.5 2e5)
vref vref gnd 3.0
.tran 1n 10u
.print v(*) level=1
.end
/************ C functional model file *****************/
#include <math.h>
#include <stdio.h>
#include "hm.h"
/* digital to analog converter */
void d2a()
{
     static int data_size, d_id, vout_id, clk_id, conv_id,
vref_id;
     int                                i;
     HMLogic                            clk, conv, *dd;
     double                             vref, vout, vinc;
     double                             *cap_array;
     if (hmSimStage()==HMSTART) {
          /* convert port name to port id; using port id */
          /* is more efficient than using the name since */
          /* it eliminates the time needed to find the   */
          /* port index when the port name is used in    */
          /* each later API reference                    */
          d_id      =hmPortName2PortId(“d”);
          vout_id =hmPortName2PortId(“vout”);
          vref_id =hmPortName2PortId(“vref”);
          clk_id =hmPortName2PortId(“clk”);
          conv_id =hmPortName2PortId(“conv”);
          /* the data bus size is given by the instantiation call */
          data_size=hmPortBusSizeById(d_id);
          hmSetAnalogPortValueById(vout_id, 0.0);
          hmSetDigitalStateValue("conv", hmZero);
          /* set data bus capacitance */
          cap_array=(double *) calloc (data_size,
sizeof(double));
               for (i=0; i<data_size; i++)
               cap_array[i]=1.0E-13; /* 100fF */
          hmSetPortBusCapById (d_id, 0, data_size-1, cap_array);
     }
```

```
          else if (hmSimStage()==HMSIM) {
          clk=hmDigitalPortValueById (clk_id);
          conv=hmDigitalStateValueById (conv_id);
          /* no data conversion until the next clock toggle */
          if (clk==hmOne) {
          hmSetDigitalStateValueById (conv_id, hmZero);
          return;
          }
          /* data conversion occurs at negative clk edge */
          if (clk==hmZero && conv==hmZero) {
               vout=0.0;
               vref=hmAnalogPortValueById (vref_id);
               /* digital to analog data conversion */
               vinc=0.5* vref;
               dd=hmDigitalPortBusValueById (d_id, 0, data_size-
1);
               for (i=0; i<data_size; i++) {
               vout +=dd[data_size-i-1]*vinc;
               vinc *=0.5;
               }
          hmSetAnalogPortValueById (vout_id, vout);
          hmSetDigitalStateValueById (conv_id, hmOne);
          /* data converted */
          /* the array space of dd is allocated in
hmDigitalPortBusValue */
          /* and is required to be freed otherwise memory leakage
occurs */
          hmFree (dd);
          }
     }
}
/* analog to digital converter */
void a2d()
{
          int                                        i, dbus_id,
mt=2;
          static int                                 data_size,
dout_id, clk_id, conv_id;
          static HMLogic                             *dout;
          HMLogic                                    clk, conv;
          double                                     vref, vin,
vdiff;
          if (hmSimStage()==HMSTART) {
               data_size=hmPortBusSize("dout");
               dout_id=hmPortName2PortId("dout");
               conv_id=hmPortName2PortId("conv");
               clk_id=hmPortName2PortId("clk");
               dout=(HMLogic *) calloc(data_size,
sizeof(HMLogic));
```

```
                hmSetDigitalPortBusValueById(dout_id, 0,
data_size-1, dout);
                hmSetDigitalStateValueById(conv_id, hmZero);
                return;
                }
           if(hmSimStage()==HMSIM) {
                clk=hmDigitalPortValueById(clk_id);
                conv=hmDigitalStateValueById(conv_id);
                /* data conversion occurs at rising clock edge */
                if (clk==hmZero) {
                     hmSetDigitalStateValueById(conv_id, hmZero);
                     return;
                }
                vin=hmAnalogPortValue("vin");
                vref=hmAnalogPortValue("vref");
                if(clk==hmOne && conv==hmZero) {/* rising clock
edge */
                for (i=0; i<data_size; i++) {
                     vdiff=vin - vref/mt;
                     dout[data_size-i-1]=(vdiff > 0.0)? hmOne:
hmZero;
                     vin=(vdiff > 0.0)? vdiff : vin;
                     mt +=mt;
                }
                hmSetDigitalPortBusValueById(dout_id, 0,
data_size - 1, dout);
                hmSetDigitalStateValueById(conv_id, hmOne);
           }
      }
}
void hsimC_model()
{
      pHMMODEL pf, pg;
      pf=hmCreateModel("a2d",a2d);
      hmDefAnalogPort(pf,"vin", hmInput); /* analog input voltage
*/
      hmDefAnalogPort(pf,"vref", hmInput); /* supply voltage */
      hmDefDigitalPort(pf,"clk", hmInput); /* clock input    */
      hmDefVarDigitalPortBus(pf,"dout",hmOutput,"size");
/* digital output */
      hmDefDigitalState(pf,"conv"); /* clock edge recorder */
      pg=hmCreateModel(“d2a”, d2a);
      hmDefAnalogPort(pg, "vout", hmOutput); /* analog output
voltage */
      hmDefAnalogPort(pg, "vref", hmInput); /* supply voltage */
      hmDefDigitalPort(pg,"clk", hmInput); /* clock input */
      hmDefVarDigitalPortBus(pg,"d", hmInput, "size"); /* digital
input */
      hmDefDigitalState(pg,"conv"); /* clock edge recorder */
```

```
}
```

Port Delay Example

Port Delay Example - C Functional Model File

```
/*                                                  */
/*   Synopsys Corporation                           */
/*                                                  */
/*   Port Delay example                             */
/*   C Functional Model File                        */
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "hm.h"
void inv() /* just invert */
{
     static in_id, out_id;
          HMLogic in;
          if (hmSimStage()==HMSTART) {
               /* convert port name to port id; using port id*/
               /* is more efficient than using the name since*/
               /* it eliminates the time needed to find the*/
               /* port index when the port name is used in*/
               /* each later API reference        */
               in_id=hmPortName2PortId("in");
               out_id=hmPortName2PortId("out");
               }
               in=hmDigitalPortValueById (in_id);
               if (in==hmOne) {
                    hmSetDigitalPortValueById (out_id, hmZero);
                    return;
               }
               else if (in==hmZero) {
                    hmSetDigitalPortValueById (out_id, hmOne);
                    return;
               }
}
void dinv() /* delayed inversion */
{
          static in_id, out_id;
          HMLogic in;
               if (hmSimStage()==HMSTART) {
               /* convert port name to port id; using port id*/
               /* is more efficient than using the name since*/
               /* it eliminates the time needed to find the*/
               /* port index when the port name is used in*/
               /* each later API reference        */
               in_id=hmPortName2PortId("in");
               out_id=hmPortName2PortId("out");
               /* Set the port rising delay time 15ns and falling
delay time 25ns */
               hmSetDigitalPortDelayById(in_id, 15e-9, 25e-9);
          }
```

```
          in=hmDigitalPortValueById (in_id);
          if (in==hmOne) {
               hmSetDigitalPortValueById (out_id, hmZero);
          return;
          }
          else if (in==hmZero) {
               hmSetDigitalPortValueById (out_id, hmOne);
               return;
          }
}
void hsimC_model()
{
     pHMMODEL pf, pg;
     pf=hmCreateModel("inv",inv);
     hmDefDigitalPort(pf,"in", hmInput); /* input */
     hmDefDigitalPort(pf,"out", hmOutput); /* output */
     pg=hmCreateModel("dinv", dinv);
     hmDefDigitalPort(pg,"in", hmInput); /* input */
     hmDefDigitalPort(pg,"out", hmOutput); /* output */
}
```

Port Delay Example - HSIM Netlist File

```
*                                                  *
*    Synopsys Corporation                          *
*                                                  *
*    Port Delay Example                            *
*    hsim netlist file                             *
*
*.param HSIMFMODLIB="./fmdq_debug.dll"
.param HSIMFMODLIB="./fmdq.lib"
.param HSIMSPEED=5
.param HSIMUSEHM="inv dinv"
x1 vin vout inv
x2 vin dvout dinv
*vin vin 0 pwl 0 0 0.8u 0 0.81u 3v 1.5u 3v 1.51u 0v
vin vin 0 pulse 0v 3v 100n 10n 10n 100n 200n
.tran 1n 2000n
.print v(*) level=1
.end
```

RAM Example

The RAM example models the READ/WRITE operations of a RAM with 10-bit address and 8-bit data buses. The initial state of some memory core cells,

other than the default state setting, is defined in the file core_state. The address and data inputs come from the vector file ram. vec and the simulation result is checked by the pattern specified in ram_out.vec.

```
* SPICE netlist file
* memory with 10-bit address and 8-bit data buses
.param HSIMOUTPUT=fsdb
.param HSIMFMODLIB=”./a.so”
.param hsimvectorfile=ram.vec
.param hsimvectorfile=ram_out.vec
x1 CE READ_RAM ADDR9 ADDR8 ADDR7 ADDR6 ADDR5
+ ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
+ DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 hmram
+ addr_size=10 data_size=8
.print v(*) level=1
.tran 1n 2u
.end
; input vector file ram.vec
signal CE READ_RAM ADDR9 ADDR8 ADDR7 ADDR6 ADDR5
+ ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
+ DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
radix              11        244            44
io                 ii        iii            bb
0                  00        000            00
100                01        000            a5
200                10        137            37
300                10        2f4            2d
400                10        0ec            c6
500                11        2f4            ZZ
600                11        137            ZZ
700                11        15a            ZZ
800                11        2f4            ZZ
900                11        0ec            ZZ
1000               11        326            ZZ
; output vector comparison file ram_out.vec
signal DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
io oo
radix 44
;DATA
0                  XX
100                XX
200                XX
300                XX
400                XX
505                2d
605                37
705                f0
805                2d
905                c6
1005               cf
/*************** C functional model file ***************/
#include <stdio.h>
```

```
#include "hm.h"
void hmram()
{
     static int          DATAid, ADDRid, CEid, RWid;
     int                 ADDR_COUNT, DATA_COUNT;
     HMLogic             *DATA, *ADDR, CE, READ, DATA_BUF[100];
     ADDR_COUNT=hmPortBusSize("ADDR"); /* address bus size */
     DATA_COUNT=hmPortBusSize("DATA"); /* data bus size */
     if(hmSimStage()==HMSTART) {
          /* using id instead of name is more */
          /* efficient in each API reference */
          DATAid=hmPortName2PortId("DATA");
          ADDRid=hmPortName2PortId("ADDR");
          CEid=hmPortName2PortId("CE");
          RWid=hmPortName2PortId("RW");
          /*        allocate a core memory of ADDR_COUNT words
*/
          /*             each word has DATA_COUNT bits*/
          /*        the default state of each core cell is f0*/
          /*        except for those cells defined in file
core_state                                          */
               hmInitMemCore("core", ADDR_COUNT, DATA_COUNT,
"f0",
               "core_state");
               }
          ADDR=hmDigitalPortBusValueById(ADDRid, ADDR_COUNT-1,
0);
          CE=hmDigitalPortValueById(CEid);
          READ=hmDigitalPortValueById(RWid);
          if (CE==hmZero) {/* RAM disabled */
               hmDisablePortBusDriveById(DATAid,0,DATA_COUNT-
1);
               }
          else if (CE==hmOne) {
               if (READ==hmOne) {/* read operation */
                    hmEnablePortBusDriveById(DATAid, DATA_COUNT-
1, 0);
                    hmReadMemCore("core", ADDR, DATA_BUF);

hmSetDigitalPortBusValueById(DATAid,DATA_COUNT-1, 0,DATA_BUF);
               }
               else if (READ==hmZero) {/* write operation */

hmDisablePortBusDriveById(DATAid,0,DATA_COUNT-1);
                    DATA=hmDigitalPortBusValueById(DATAid,
DATA_COUNT-1, 0);
                    hmWriteMemCore("core", ADDR, DATA);
               /* the DATA array is allocated in
hmDigitalPortBusValue    */
```

```
                /* need to free this array otherwise memory leakage
occurs */
                     hmFree (DATA);
                     }
                }
          /* need to free this array to avoid memory leakage */
          hmFree(ADDR);
}
hsimC_model()
{
     pHMMODEL    pf;
     pf=hmCreateModel("hmram",hmram);  /* RAM model definition */
     hmDefDigitalPort(pf,"CE",hmInput); /* enable control */
     hmDefDigitalPort(pf,"RW",hmInput); /* Read and Write control
*/
     hmDefVarDigitalPortBus(pf, "ADDR", hmInput, "addr_size");
     /* address */
     /* bidirectional data bus; the bus size is determined by */
     /* parameter data_size in the instantiation call */
     hmDefVarDigitalPortBus(pf, "DATA", hmBiput, "data_size");
     /* memory core definition */
     hmDefMemCore(pf, "core");
}
```

Event Handling Example

hmEventHandle

```
HMEventHandle hmSchedulePortEvent(char *portname, double time,
void *userdata); HMEventHandle hmSchedulePortEventById(int
portid, double time, void *userdata); HMEventHandle
hmGetCurrentEvent(void);
int hmDescheduleEvent(HMEventHandle eventhandle);
int hmGetEventPortId(HMEventHandle eventhandle);
void * hmGetEventData(HMEventHandle eventhandle);
```

Note:

Allocating and freeing memory for user-defined data is your responsibility.

Note:

Do *not* de-schedule an event that has been processed.

```
---------------------------fmev.c---------------------------
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "hm.h"
void inv()
{
static int in_id, out_id, id1, id2, id3;
HMLogic in;
HMEventHandle *ev;
int id;
int d;
if (hmSimStage() == HMSTART) {
/*        convert port name to port id; using port id*/
/*        is more efficient than using the name since*/
/*        it eliminates the time needed to find the*/
/*        port index when the port name is used in*/
/*        each later API reference               */
in_id=hmPortName2PortId("in");
out_id=hmPortName2PortId("out");
id1=hmPortName2PortId("out1");
id2=hmPortName2PortId("out2");
id3=hmPortName2PortId("out3");
hmSchedulePortEventById(id1, 1e-10, (void*)0);
hmSchedulePortEventById(id2, 2e-10, (void*)0);
hmSchedulePortEventById(id3, 3e-10, (void*)0);
hmSetDigitalPortValueById(out_id, hmZero);
}
else if (hmSimStage() == HMSIM) {
in=hmDigitalPortValueById (in_id);
while(ev=hmGetCurrentEvent()) {
id=hmGetEventPortId(ev);
if(id==id1) {
d=(int)hmGetEventData(ev);
if(d==1) {
hmSetDigitalPortValueById(id1, hmZero);
hmSchedulePortEventById(id1, hmPresentTime() + 1e-10, (void*)0);
}
else {
hmSetDigitalPortValueById(id1, hmOne);
hmSchedulePortEventById(id1, hmPresentTime() + 1e-10, (void*)1);
}
}
if(id==id2) {
d=(int)hmGetEventData(ev);
if(d==1) {
hmSetDigitalPortValueById (id2, hmZero);
hmSchedulePortEventById (id2, hmPresentTime() + 2e-10, (void*)0);
```

```
}
else {
hmSetDigitalPortValueById (id2, hmOne);
hmSchedulePortEventById (id2, hmPresentTime() + 2e-10, (void*)1);
}
}
if(id==id3) {
d=(int)hmGetEventData(ev);
if(d==1) {
hmSetDigitalPortValueById (id3, hmZero);
hmSchedulePortEventById (id3, hmPresentTime() + 3e-10, (void*)0);
}
else {
hmSetDigitalPortValueById (id3, hmOne);
hmSchedulePortEventById (id3, hmPresentTime() + 3e-10, (void*)1);
}
}
}
/* no data conversion until the next clock toggle */
if (in == hmOne) {
hmSetDigitalPortValueById (out_id, hmZero);
return;
}
else if (in == hmZero) {
hmSetDigitalPortValueById (out_id, hmOne);
return;
}
}
}
void hsimC_model()
{
pHMMODEL pf;
pf=hmCreateModel("inv",inv);
hmDefDigitalPort(pf,"in", hmInput); /* clock input */
hmDefDigitalPort(pf,"out", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out1", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out2", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out3", hmOutput); /* clock input */
}
************************************************************
```

Part: 4 HSIM Appendices

A

Device Model Reference

Lists sources of reference information for device models.

Reference Sources for Device Models

The information shown in the table below is provided to assist users in locating valuable background information not covered in this manual.

Table 89 Device Model References

Models	Reference
BSIM3 & BSIM4	http://www-device.eecs.berkeley.edu/~bsim3/
BSIM3SOI & BSIM4SOI	http://www-device.eecs.berkeley.edu/~bsimsoi/
MOS9 & MOS11	http://www.semiconductors.philips.com/philips_Models
EKV	http://legwww.epfl.ch/ekv
HiSIM	http://www.starc.jp/kaihatu/pdgr/hisim
UCB SPICE	http://www-cad.eecs.berkeley.edu/Software/software.html
VBIC BJT	http://www.fht-esslingen.de/institute/iafgp/neu/VBIC/
MEXTRAM BJT	http://www.semiconductors.philips.com/philips_Models
HICUM BJT	http://www.iee.et.tu-dresden.de/iee/eb/comp_mod.html
JUNCAP2	http://www.semiconductors.philips.com/Philips_Models/additional/juncap/index.html

Table 89 Device Model References (Continued)

Models	Reference
PSP	http://pspmodel.ee.psu.edu/

B

Custom Output Interface

Provides information on how to accommodate a specific output format not compatible with existing design flows using a template library to construct a generator of a waveform that will be suitable for that format.

HSIM supports the following output formats.

- FSDB Output Format
- WSF Output Format
- EPIC ASCII Output Format
- HSIM Output Format
- WDF Sandwork Output Format

On some occasions the regular output formats may not be compatible with an existing design flow. To accommodate a specific output format, a template library can be used to construct a generator of a waveform that will be suitable for that format.

For HSIM to load the template library, a dynamic library loader, such as dlopen on the Solaris system, is required.

The examples in this appendix illustrate the use of this interface only, and include the following:

- The header file coi.h which contains a list of common data shared by HSIM and your waveform generator library. The coi.h is available in $HSIM_HOME/etc/include after the completion of HSIM installation.
- The generation library myformat.c which illustrates a simple implementation of the interface.

  Note:

  In operation, the detailed content of the library is developed by you.

Call Custom Library Mechanism

The output file format is specified by the following HSIM command.

```
.param HSIMOUTPUT=out_format
```

To specify a particular output format, the name of the corresponding format must be set with HSIMOUTPUT on page 105. An additional HSIM parameter is used to specify the full path of the waveform file generator library.

```
.param HSIMCOILIB="full path of the waveform file generator
   library"
```

The full path should include the library name. HSIMCOILIB on page 61 is optional. If there is no HSIMCOILIB specified, HSIM searches for the lib<out_format>.so in the following directories, in the order shown:

- Run directory
- $HOME directory
- $HSIM_HOME/platform/<port>/bin
- LD_LIBRARY_PATH on Solaris and on Linux, and SHLIB_PATH on HP

where <output_format> is the value of HSIMOUTPUT on page 105 and all the letters in <output_format> have to be in upper case.

Functional Specification

For flexibility, the interface specification handles the following items:

- Common include file
- Creation and initialization of a waveform file function
- Creation of a header function
- Creation of a waveform into a waveform file function
- Creation of a duplicate waveform into a waveform file function
- Addition of values to one or more waveforms into a waveform file function
- Completion of the waveform file creation function

A procedural interface is provided for you to prepare a waveform file generator library. For transient analysis, the HSIMAPI calling sequence is in the following order:

```
out_format_CreateWaveFile()
out_format_CreateHeader()
out_format_BeginCreateWave()
out_format_CreateWave()
out_format_EndCreateWave()
out_format_AddNextDigitalValueChange() [optional; see note below]
out_format_AddNextAnalogValueChange() [optional; see note below]
out_format_CloseWaveFile()
```

For DC analysis, the HSIMAPI calling sequence is in the following order:

```
out_format_CreateWaveFileDC()
out_format_CreateHeaderDC()
out_format_BeginCreateWaveDC()
out_format_CreateWaveDC()
out_format_EndCreateWvaveDC()
out_format_AddNextAnalogValueChangeDC()
out_format_CloseWaveFileDC()
```

Note:

The API out_format serves as a name holder. out_format is specified in the command .param HSIMOUTPUT=out_format. A different name can be used, such as MYFORMAT. Except for steps 6 and 7 , all other steps are called only once by HSIM. Steps 6 and 7 are called when a signal value changes.

HSIM invokes these functions only when a value change occurs to a specific signal at the current simulation time. At a particular simulation time, HSIM calls these functions exclusively for those signals with value changes. The generated waveform file is value-change based instead of tabular based.

For AC analysis, HSIM API calling sequence is in the following order:

```
out_format_CreateWaveFileAC()
out_format_CreateHeaderAC()
out_format_BeginCreateWaveAC()
out_format_CreateWaveAC()
out_format_EndCreateWaveAC()
out_format_AddNextAnalogValueChangeAC()
out_format_format_CloseWaveFileAC()
```

Common Include File

The common include (coi.h) header file describes the common information used by HSIM and the user libraries to generate waveform files. The header file must contain the following declarations.

- Declaration of structs
- Declaration of enumerations

Note:

The coi.h header file must remain intact and be used in building the waveform file generator library.

For more information, refer to Building a Waveform File Generator Library on page 641. Example 69 shows the code for the header file coi.h.

Example 69 The header file coi.h

```
/*                                  */
/* Synopsys Corporation            */
/*                                  */
#ifndef COI_H
#define COI_H
/* Structures and Enumerations
**********************************************/
enum SignalType {
   VOLTAGE=0,
   CURRENT,
   LOGIC,
   POWER,
   DEFAULT
};
/***** DON'T CHANGE the following hsimparam data structure. Use
it as it is *****/
typedef struct hsimparam
{
double   StopTime;/* stopTime as in the */
               /* .TRAN, .DC, or .AC */
double   TimeScale;/* time scale factor */
double   VoltageScale;/* voltage scale factor */
double   CurrentScale;/* current scale factor */
double   Temperature;/* temperature as in .TEMP */
int      NumOfSweep;/* total number of sweep */
               /* step in .TRAN ... SWEEP */
int      NumOfTemperature;/* total number of */
               /* temperatures in .TEMP */
int      NumOfAlter;/* total number of .ALTER */
int      CurrOfSweep;/* current number of SWEEP */
int      CurrOfTemperature;/* current number of */
               /* Temperature */
int      CurrOfAlter;/* current number of .ALTER */
int      CurrOfIteration;/* current number of */
               /* iteration generated */

double   Vres; /* voltage resolution */
double   Ires; /* current resolution */
/******************* USED BY HSIM ONLY *******************/
char     *HierId;/* used by HSIM only */
double   TopSupply;/* used by HSIM only */
double   OutFileSplit;/* used by HSIM only */
double   Cfg_Banner;/* used by HSIM only */
double   Cfg_Xout;/* used by HSIM only */
double   Cfg_OutputFsdbSize;/* used by HSIM only */
double Cfg_OutputWdfSize;/* used by HSIM only */
double Cfg_OutputWdfComperess;/* used by HSIM only */
*************************************************************/
```

```
double   StartTime;/* startTime as in the .DC */
               /* or .AC sweep */
char*    SweepVarName;/* Sweep Varialbe Name as */
               /* in the .DC or .AC */
               /* sweep */
char*    SweepType;/* For .DC and .AC sweep */
               /* type, LIN, DEC, OCT, */
               /* OTHERS */
int      NumOfMonteCarlo;/* total number of */
               /* MONTECARLO */
int      CurrOfMonteCarlo;/* current number of */
               /* MONTECARLO */
char*    SweepVarName2;/* external Sweep Varialbe */
               /* Name as in the .DC or */
               /* .AC */
} hsimparam;
#endif
```

Create and Initialize the Waveform File Function

The following section provides examples of creating and initializing waveform file functions using the CreateWaveFile function.

CreateWaveFile

Returns a handle and creates the waveform file. When necessary, multiple waveform files are created to store all the waveform information.

Syntax

For Transient Analysis

```
void * Hdl=void * out_format_CreateWaveFile(char *BaseName,
   hsimparam *hsimparam);
```

For DC Analysis

```
void *Hdl=void * out_format_CreateWaveFileDC(char
   *BaseName, hsimparam *hsimparam);
```

For AC Analysis

```
void *Hdl=void * out_format_CreateWaveFileAC(char
   *BaseName, hsimparam *hsimparam);
```

Parameters

BaseName

The string name of the base name for waveform information.

hsimparam

Contains parameters for file initialization.

Return Values

Hdl

The handle attached to the file descriptors, and so on.

-1

Indicates an error occurred.

Note:

The out_format is the name specified in .param HSIMOUTPUT=out_format.

Description

The waveform information is stored in HSIM format in the .ctrl and the .out files. The function creates the files, stores the file descriptors into the customer-specific structure, and returns the reference of that structure. The rest of the APIs use the reference as a handler to dump out the waveform information. The base name is the prefix for those waveform files.

HSIM does not need to know the customer specific structure; the handler is a pass-through token for the rest of APIs.

Example

In this example, hsimparam is defined as follows:

```
typedef struct hsimparam {
double StopTime;
double TimeScale;
double VoltageScale;
double CurrentScale;
double Temperature;
int NumOfSweep;
int NumOfTemperature;
int NumOfAlter;
int CurrOfSweep;
int CurrOfTemperature;
int CurrOfAlter;
int CurrOfIteration;
double Vres;
double Ires;
} hsimparam;
```

Create a Header Function

The following section provides an example of the header function.

CreateHeader

Adds a header to the waveform file.

Syntax

For Transient Analysis

```
int out_format_CreateHeader(void *Hdl);
```

For DC Analysis

```
int out_format_CreateHeaderDC(void *Hdl);
```

For AC Analysis

```
int out_format_CreateHeaderAC(void *Hdl);
```

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Create a Waveform Into a Waveform File

The following sections provides examples for creating a waveform into a waveform file. The functions include creating and ending a wave.

BeginCreateWave

Can be used for additional initialization when starting to add new signal(s) into the waveform file.

Syntax

For Transient Analysis

```
int out_format_BeginCreateWave(void *Hdl);
```

For DC Analysis

```
int out_format_BeginCreateWaveDC(void *Hdl);
```

For AC Analysis

```
int out_format_BeginCreateWaveAC(void *Hdl);
```

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Description

BeginCreateWave may be used for additional initialization when starting to add new signal(s) into the waveform file. If no special setup takes place before the new waveform is created, BeginCreateWave only needs to return 0 (zero).

CreateWave

Creates a new waveform handle.

Syntax

For Transient Analysis

```
void * WaveHdl=void * out_format_CreateWave(
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

For DC Analysis

```
void * WaveHdl=void * out_format_CreateWaveDC(
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

For AC Analysis

```
void * WaveHdl=void * out_format_CreateWaveAC(
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

FullPathScopeName

The full path of the hierarchy, except the signal name.

SignalName

The name of the signal.

ScopeSeparator

The separating character used in FullPatchScopenName.

Type

The waveform type.

Return Values

WaveHdl

The signal handle.

-1

An error occurred.

Description

CreateWave creates a new waveform handle. The handle is used by the following APIs to print the value when a value change occurs to this signal waveform handle.

Example

In this example, SignalType is defined as follows:

```
enum SignalType {
   VOLTAGE=0,
   CURRENT,
   LOGIC,
   POWER,
   DEFAULT
};
```

Note:

LOGIC type is not available to DC or AC analysis.

EndCreateWave

Can be used at the end of creating wave.

Syntax

For Transient Analysis

```
int out_format_EndCreateWave (void *Hdl);
```

For DC Analysis

```
int out_format_EndCreateWaveDC (void *Hdl);
```

For AC Analysis

```
int out_format_EndCreateWaveAC (void *Hdl);
```

Description

EndCreateWave may be used at the end of creating wave. If no special setup takes place before the new waveform is created, EndCreateWave only needs to return 0 (zero).

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Create a Duplicate Waveform Into a Waveform File

CreateDuplWave is used to create a duplicate waveform and insert it into a waveform file for transient analysis. The values of the duplicate wave form are copied from a master wave to which it is aliased.

CreateDuplWave

Creates a duplicate waveform.

Syntax

For Transient Analysis

```
void * WaveHdl=void * out_format_CreateDuplWave(
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type
void *MasterWave);
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

FullPathScopeName

The full path of the hierarchy, except the signal name.

SignalName

The name of the signal.

ScopeSeparator

The separating character used in FullPatchScopenName.

Type

The waveform type.

MasterWave

The MasterWave provides the values for creating a duplicate wave.

Return Values

WaveHdl

The signal handle.

-1

An error occurred.

Description

CreateDuplWave is used to create a duplicate waveform and insert it into a waveform file for transient analysis. The values of the duplicate wave form are copied from a master wave to which it is aliased.

Add Waveform Values Into a Waveform File

This section provides examples for adding digital and analog values into a waveform file for transient analysis, and for adding analog values into a wave file for DC and AC analysis.

AddNextDigitalValueChange

Stores digital data of a waveform into a waveform file.

Syntax

```
int out_format_AddNextDigitalValueChange(
 void *Hdl,
 void *WaveHdl,
 double Time,
 int Value
 );
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile()

WaveHdl

The handle returned from the CreateWave() for transient analysis, CreateWaveDC() for DC analysis, or CreateWaveAC() for AC analysis.

Time

The current simulation time

b

The waveform data at the current time

Return Values

0

No errors occurred.

-1

An error occurred.

Description

AddNextDigitalValueChange stores digital data of a waveform into a waveform file.

AddNextAnalogValueChange

Stores analog values of a waveform into a waveform file.

Syntax

For Transient Analysis

```
int out_format_AddNextAnalogValueChange(
     void *Hdl,
     void *WaveHdl,
     double Time,
     double X_Value
 );
```

For DC Analysis

```
int out_format_AddNextAnalogValueChangeDC(
     void *Hdl,
     void *WaveHdl,
     double Time,
     double X_Value
 );
```

For AC Analysis

```
int out_format_AddNextAnalogValueChangeAC(
     void *Hdl,
     void *WaveHdl,
     double Time,
     double X_Value
 );
```

Parameters

The input parameters include:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

WaveHdl

The handle returned from the CreateWave() for transient analysis, CreateWaveDC() for DC analysis, or CreateWaveAC() for AC analysis.

Time

The current simulation `X_Value`.

Value

The waveform value at the current `X_Value`.

Return Values

0

No errors occurred.

-1

An error occurred.

Description

AddNextAnalogValueChange stores analog values of a waveform into a waveform file.

End the Waveform File Process

This section provides an example for closing a waveform.

CloseWaveFile

Closes a waveform file at the end of the simulation.

Syntax

For Transient Analysis

```
int out_format_CloseWaveFile(void *Hdl);
```

For DC Analysis

```
int out_format_CloseWaveFileDC(void *Hdl);
```

Description

CloseWaveFile closes a waveform file at the end of the simulation.

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Building a Waveform File Generator Library

This section provides an example waveform file generator library. The sample code generates a MYFORMAT waveform file format.

The following files must be prepared for the MYFORMAT waveform file generator library:

- myformat.c: Source file of the waveform file generator library.
- myformat.h: Header file of the waveform file generator library.
- makefile: Makefile for building the file generator library.

To build this library, enter make as shown in Example 70.

Example 70

```
<<< myformat.h >>>

/* Copyright 1998 - 2004 */
/* Synopsys Corporation */
/* */
#ifndef MYFORMAT_H
#define MYFORMAT_H
#ifdef WIN32
#include <windows.h>
#ifdef COI_EXPORTS
#define COI_API __declspec(dllexport) extern
#define COI_API __declspec(dllexport) extern
#define COI_API __declspec(dllexport) extern
#else
#define COI_API __declspec(dllimport) extern
__declspec(dllexport) void *MYFORMAT_CreateWaveFile (char
*,HsimParam*);
__declspec(dllexport) int MYFORMAT_CreateHeader (void *);
__declspec(dllexport) int MYFORMAT_BeginCreateWave (void *);
__declspec(dllexport) int MYFORMAT_EndCreateWave (void *);
__declspec(dllexport) void *MYFORMAT_CreateWave (void *, char
*,char *, char, enum SignalType);
__declspec(dllexport) void *MYFORMAT_CreateDuplWave (void *, char
*,char *, char, enum SignalType, void *MasterWave);
__declspec(dllexport) int MYFORMAT_AddNextDigitalValueChange
(void *, void *, double time, int);
__declspec(dllexport) int MYFORMAT_AddNextAnalogValueChange
(void *, void*, double, double);
__declspec(dllexport) int MYFORMAT_CloseWaveFile (void *);
#endif
#else
#define COI_API extern
#endif
#endif

<<< myformat.c >>>
/* */
/* Copyright 1998 - 2004 */
/* Synopsys Corporation */
/* */
#include "coi.h"
#include <stdio.h>
#include "myformat.h"
static int numOfWave=0;
int *myFileId;
int *myWaveId;
void *
```

```
MYFORMAT_CreateWaveFile(char *BaseName, struct HsimParam
*hsimparam)
{
fprintf(stdout,"MYFORMAT/CreateWaveFile\n");
fprintf(stdout,"BaseName=%s\n",BaseName);
fprintf(stdout,"StopTime=%g\n",hsimparam->StopTime);
fprintf(stdout,"Temperature=%g\n",hsimparam->Temperature);
myFileId=(int *)malloc(sizeof(int));
*myFileId=1;
return(myFileId);
}
int
MYFORMAT_CreateHeader(void *Hdl)
{
int *hdl=(int *)Hdl;
fprintf(stdout,"MYFORMAT/CreateHeader\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
return(0);
}
void *
MYFORMAT_CreateWave(void *Hdl, char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type)
{
int *hdl=(int *) Hdl;
fprintf(stdout,"MYFORMAT/CreateWave\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
fprintf(stdout,"FullPathScopeName=%s\n",FullPathScopeName);
fprintf(stdout,"SignalName=%s\n",SignalName);
fprintf(stdout,"ScopeSeparator=%c\n",ScopeSeparator);
fprintf(stdout,"Type=%s\n",(Type==VOLTAGE)?VOLTAGE:((Type==CURR
ENT)?CURRENT:LOGIC));
myWaveId=(int *)malloc(sizeof(int));
*myWaveId=numOfWave++;
return(myWaveId);
}
void *
MYFORMAT_CreateDuplWave(void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type,
void *MasterWave)
{
int *hdl=(int *) Hdl;
fprintf(stdout,"MYFORMAT/CreateDuplWave\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
```

```
fprintf(stdout,"FullPathScopeName=%s\n",FullPathScopeName);
fprintf(stdout,"SignalName=%s\n",SignalName);
fprintf(stdout,"ScopeSeparator=%c\n",ScopeSeparator);
fprintf(stdout,"Type=%s\n",(Type==VOLTAGE)?VOLTAGE:((Type==CURR
ENT)?CURRENT:LOGIC));
myWaveId=(int *)malloc(sizeof(int));
*myWaveId=numOfWave++;
return(myWaveId);
}

int
MYFORMAT_BeginCreateWave(void *Hdl)
{
int *hdl;
hdl=(int *)Hdl;
fprintf(stdout,"MYFORMAT/BeginCreateWave\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
return(0);
}
int
MYFORMAT_AddNextDigitalValueChange(void *Hdl,
void *WaveHdl,
double Time,
int Value)
{
int *hdl1=(int *) Hdl;
int *hdl2=(int *) WaveHdl;
fprintf(stdout,"MYFORMAT/AddNextDigitalValueChange\n");
fprintf(stdout,"Hdl=%d\n",*hdl1);
fprintf(stdout,"WaveHdl=%d\n",*hdl2);
fprintf(stdout,"Time=%g\n",Time);
fprintf(stdout,"Value=%d\n",Value);
return(0);
}
int
MYFORMAT_AddNextAnalogValueChange(void *Hdl,
void *WaveHdl,
double Time,
double Value)
{
int *hdl1=(int *) Hdl;
int *hdl2=(int *) WaveHdl;
fprintf(stdout,"MYFORMAT/AddNextAnalogValueChange\n");
fprintf(stdout,"Hdl=%d\n",*hdl1);
fprintf(stdout,"WaveHdl=%d\n",*hdl2);
fprintf(stdout,"Time=%g\n",Time);
fprintf(stdout,"Value=%g\n",Value);
return(0);
```

```
}
int
MYFORMAT_CloseWaveFile(void *Hdl)
{
int *hdl=(int *)Hdl;
fprintf(stdout,"MYFORMAT/CloseWaveFile\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
return(0);
}
int
MYFORMAT_EndCreateWave(void *Hdl)
{
int *hdl=(int *)Hdl;
fprintf(stdout,"MYFORMAT/EndCreateWave\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
return(0);
}

<<< makefile >>>

#!/bin/sh
#==========================================================
BIN=.
#==========================================================
# --- Makefile rules
# -c C program
# -g Debug option
# -O Optimization
#==========================================================
CC=cc -fPIC -c -g -O -D_DEBUG
LINK=cc -o
SHARED=ld -G -o
#==========================================================
# --- libraries used for linking
SYSLIBS=-lc -ldl
#==========================================================
# -- Make the executable program
default:
     make libMYFORMAT.so
#==========================================================
myformat.o : myformat.c myformat.h
     $(CC) myformat.c
libMYFORMAT.so : myformat.o
     $(SHARED) $(BIN)/libMYFORMAT.so myformat.o
```

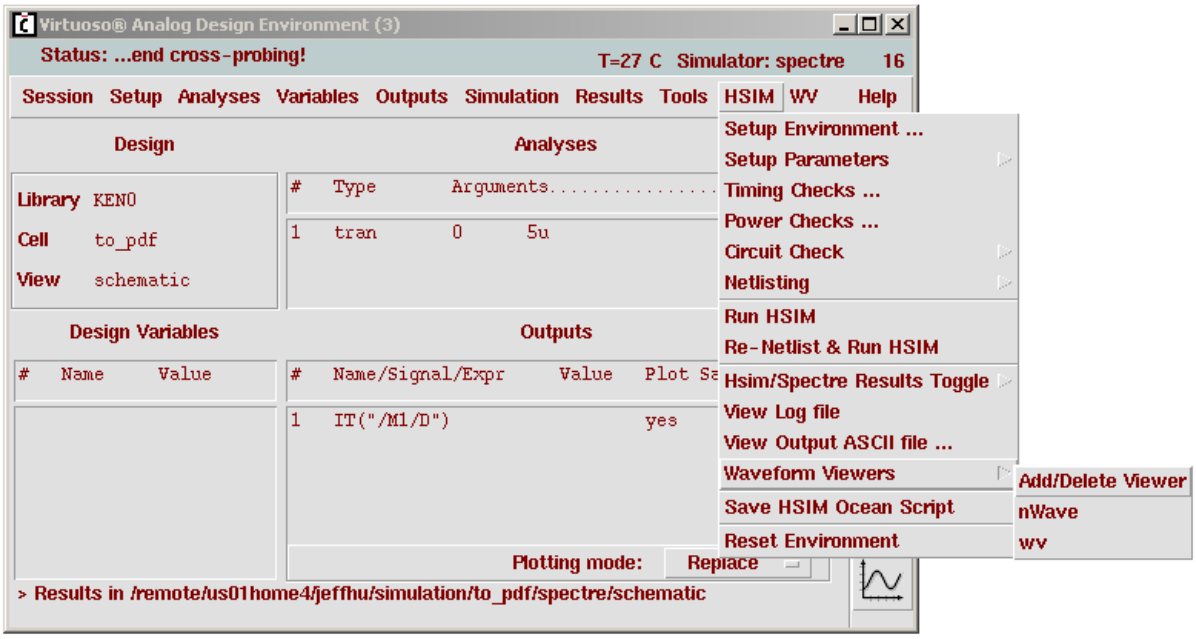
C

Waveform Viewer Customization

This appendix provides details on the customer features in nWave and WaveView Analyzer.

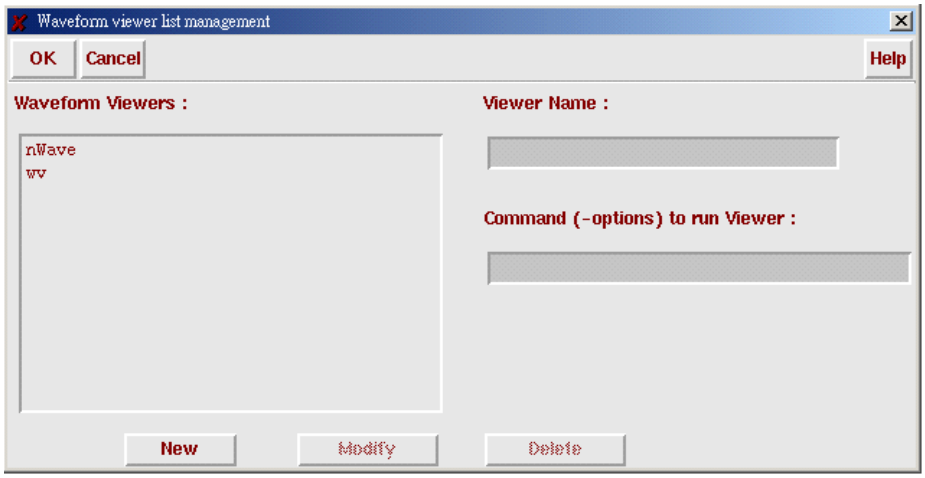
When installed, the Waveform Viewers option is displayed in the HSIM menu:

Figure 27 Waveform Viewers Menu



nWave and wv are listed in the waveform viewer list by default. The list is customizable by using the Add/Delete Viewer option. Click on Add/Delete Viewer and a customization form pops up as shown in Figure 28 on page 648.

Figure 28 Waveform Viewer List Management Screen

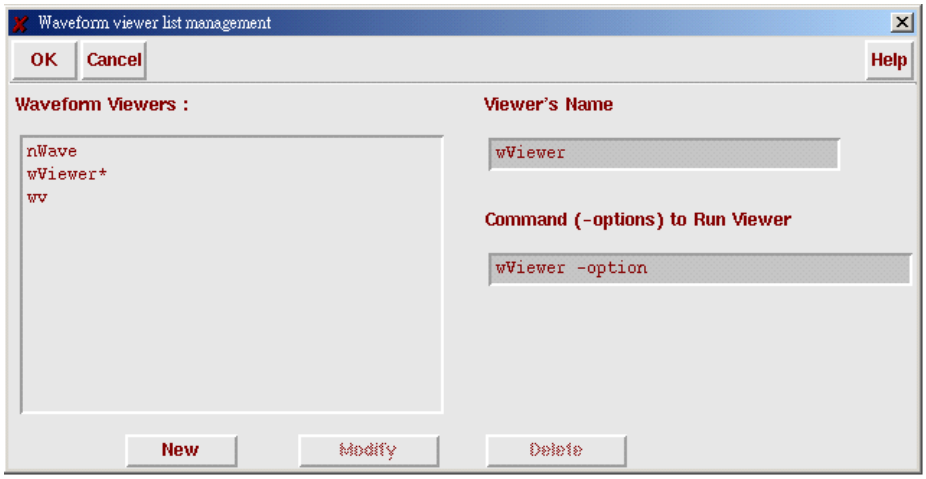


In the beginning, only the [New] button is activated and both the Viewer name and the Command (-options) to run Viewer fields are disabled. A new waveform viewer can be added to the list by pressing the [New] button or by selecting an existing waveform viewer and modifying its commands and options.

New

Clicking the [New] button makes both the Viewer Name and Command option fields editable and transforms the [New] button into the [Add] button. After entering the new viewer's name, pressing the [Add] button adds the new viewer to the list as shown in Figure 29 on page 649.

Figure 29 Waveform Viewer List



Note:

The asterisk * after the name indicates the viewer is newly added but not activated yet. Newly added viewers will not be activated and can not be invoked from the drop down menu under HSIM until the next Virtuoso session is opened.

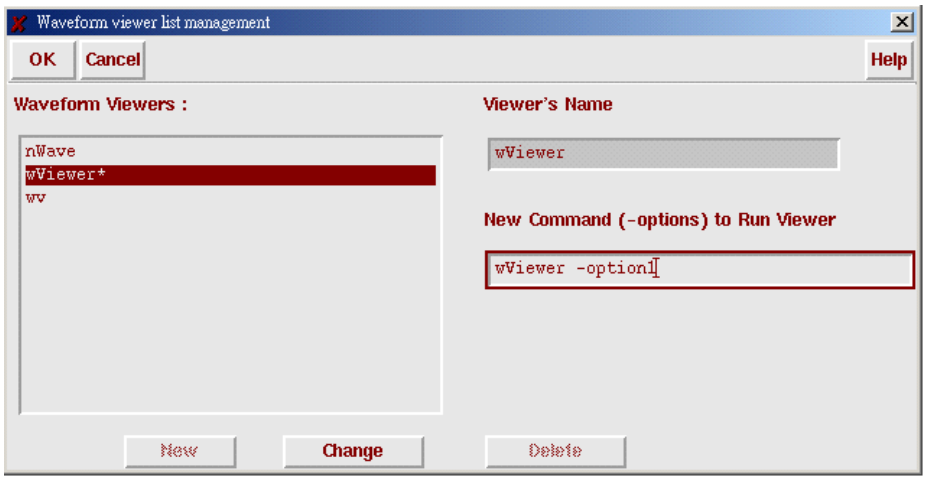
Modify, Delete

Viewer functions are affected by using the following steps:

- Select one of the existing viewers from the list to display the viewer's settings.
- Press the [Modify] and [Delete] button to disable the Viewer's Name and Command (-options) fields.
- Press the [Delete] button to remove the selected viewer from the list.
- Press the [Modify] button to make the Command (-options) field editable and change the label text. Additionally, [Modify] button transforms into the [Change] button and the [New] and [Delete] buttons are disabled.
- Once the command field is modified, press the [Change] button to complete the process.

The Viewer's Name can not be modified. To change the viewer name the current viewer must first be deleted and a new viewer added as shown in Figure 30 on page 650.

*Figure 30 Selecting wViewer**



OK

Clicking the [OK] button updates the Waveform Viewers list and saves it to the .waveViewerTable file. The modification does not immediately become effective in the HSIM->Waveform Viewers menu. The change will becomes valid in the next Virtuoso Analog Design Environment session.

.waveViewerTable

HSIM->Waveform Viewers content is stored in the .waveViewerTable file. At start up, Native Netlist Integration first looks for this file and read its content using the following rules:

Rule 1

Search the current working directory, if there is one available, and use that directory.

Rule 2

If the current working directory is not available, search your home directory, if there is one available, and use that directory.

Rule 3

If neither the current working directory or your home directory are available, Native Netlist Integration searches $HSIM_ARTISTIF/install/.waveViewerTable and use that directory.. If this file is read and a user made changes to the list, then a new .waveViewerTable file will be created under user's home directory and this file will be used in the following sessions.

The $HSIM_ARTISTIF/install/.waveViewerTable file is distributed through the Native Netlist Integration package. The system administrator normally has authority to customize this file via the GUI interface or manually and permits all users share the same file from a central location.

Two reference files are provided in the $HSIM_ARTIST_IF/install directory:

- .waveViewerTable_nWave_WaveView: This file directly accesses nWave and WaveView. It is the default for .waveViewerTable. Deleted items may also be added into the list and saved using the GUI form.
- .waveViewerTable_empty: Only allows a viewer to be added or deleted by copying the preferred viewer into .waveViewerTable.

  Note:

  It is possible to toggle between the two settings at any time.

Rule 4

If the .waveViewerTable file is not available using Rules 1 through 3, Native Netlist Integration automatically creates a .waveViewerTable in the home directory. The default .waveViewerTable contains the following menu items contained in the HSIM drop down menu:

- Add/Delete Viewer
- nWave
- wv

D

HSIM/Eldo Compatibility

Provides information on HSIM's support for Mentor Graphic's Eldo SPICE simulator.

This Appendix provides information on HSIM's support for Mentor Graphic's Eldo SPICE simulator. The information is provided as a guide to known Eldo features only. This appendix does not cover Eldo features that may exist but are not documented, or functions that will be introduced in future releases.

Starting HSIM in the Eldo Mode

To use HSIM in the Eldo mode, use the following syntax:

```
Hsim -eldo my_netlist.cir
```

Note:

HSIM always considers a GND=0. Eldo does not default to GND=0.

Note:

All Eldo syntax equations must be defined between curly brackets {}. Refer to the Mentor Graphics Eldo documentation for specifics.

Transistor Model Compatibility

Table 14 lists the HSIM/ELDO transistor model compatibility.

Table 14 HSIM/ELDO Transistor Model Compatibility

Transistor Model	HSIM	Eldo
BSIM3	Level = 53 or 49	Level = 53

Table 14 HSIM/ELDO Transistor Model Compatibility (Continued)

Transistor Model	HSIM	Eldo
BSIM4	Level = 60	Level = 60
MM9	Level = 50	Level = 50
BSIM3SO	Level = 57	Level = 55
EKV2.6	Level = 55	Level = 44

Note:

ST proprietary devices such as diodes, resistors, capacitors, MOSFETs, and Bipolar device models are integrated into HSIM to support Bipolar CMOS DMOS (BCD) technologies. These components use the same level as Eldo, but the stver option is required in the netlist.

HSIM-Supported Eldo-Specific Syntax

Table lists the HSIM-supported ELDO-specific syntax definitions.

Table 15 HSIM Supported Eldo-Specific Syntax Definitions

Command	Functional Description
!	Defines the end of a comment line(s).
TEMPER	The variable used in equations to indicate the simulation temperature specified by .temp. The HSIM equivalent TEMP.
FREQ	The variable used to indicate current frequency. When FREQ=1 for DC and TRAN simulations, it reflects the current frequency during AC analysis.
#com	Defines the beginning of a comment line(s).
#endcom	Defines the end of a comment line(s).
Yxx	Defines lossy transmission lines in Eldo as W elements in HSIM.

Table 15 HSIM Supported Eldo-Specific Syntax Definitions (Continued)

Command	Functional Description
Yxx	Verilog-A model instantiations are supported with the same syntax as Eldo.

Eldo Syntax Not Supported by HSIM

Table lists the Eldo syntax not supported by HSIM.

Table 16 Eldo syntax Not Supported by HSIM

Command	Functional Description
Jxx	MESFET
NOISE	Any NOISE sources
Yxx	No Specific Eldo macro-models (Analog, Digital, Amplifier, converters, switch capacitor models, etc.)

Eldo Commands Partially or Fully Supported by HSIM

Table lists the Eldo commands partially or fully supported by HSIM.

Table 17 Eldo Commands Partially or Fully Supported by HSIM

Command	Function
.AC	Full support
.ALTER	Generalized re-run facility. HSIM will create only one .log file and one output waveform file per run. These are named sequentially for all runs with the following extension(s): a0, a1..., aN.
.CHRENT	Piecewise linear source
.CHECKSOA	Check safe operating area limits

Table 17 Eldo Commands Partially or Fully Supported by HSIM (Continued)

Command	Function
.CONNECT	Connects two nodes. HSIM only accepts .connect between a voltage/current source and a node.
.CONSO	Current used by a circuit
.DATA	Parameter sweep
.DC	DC analysis
.DEFMAC	Macro definition
.DEFWAV	Waveform definition
.DEFMOD	Define a model alias
.DEL	Remove a library name
.DSPF_INCLUDE	Include a DSPF file
.END	End netlist
.ENDL	End a library variant description
.ENDS	End a subcircuit description
.EXTRACT	Extract waveform characteristics
.HIER	Changing the hierarchy separator
.FOUR FFT	Select waveform
.GLOBAL	Global node allocation
.INCLUDE	Include a vile in an input netlist
.IC	Initial transient analysis conditions
.LIB	Insert circuit information from a library file
.MC	Monte Carlo analysis
.MCMOD LOT & DEV	Variation specification on model parameters

Table 17 Eldo Commands Partially or Fully Supported by HSIM (Continued)

Command	Function
.MEAS	Measure waveform characteristics
.MODEL	Device model description
.NODESET DC	Analysis conditions
.OP	DC operating point calculation
.OPTION	Simulator configuration
.OPTFOUR	FFT post-processor options
.PARAM	Global declarations
.PLOT	Plotting of simulation results
.PRINT	Prints results. Same behavior as .plot command. Beware it is different than Eldo.
.PROBE	Output short form. Same behavior as .plot command.
.probe VTOP	Displays all the top level node voltages.
.probe VN	Probe only node names which are not numbers
.probe vi	HSIM-supported
.probe isub	HSIM-supported
.SETBUS	Create bus
.SETSOA	Set safe operating area
.SIGBUS	Set bus signal
.STEP	Parameter sweep (supported only with .TRAN)
.SUBCKT	Sub-circuit definition
.TEMP	Set circuit temperature
.TOPCELL	Select the TOP cell subcircuit

Table 17 Eldo Commands Partially or Fully Supported by HSIM (Continued)

Command	Function
.TRAN	Transient analysis. In HSIM, the first parameter is ignored, possibility to have a .tran sweep (see documentation)

HSIM-Supported Eldo Options

Table lists the HSIM-supported Eldo options.

Table 18 HSIM-Supported Eldo Options

Command	Function
.OPTION TNOM=val	Sets the model temperature.
.OPTION	Model for binning parameters
.OPTION SCALE=va	Multiplier for MOSFET w, l, as and ad
.OPTION SEARCH=va	Search path definition for subckt files
.OPTION HISTAUMAX=val	Refer to HSIMTAUMAX on page 312
.OPTION HMIN=val	Refer to HSIMTIMESCALE on page 298
.OPTION FREQSMP=val	Refer to HSIMOUTPUTTSTEP on page 107
.OPTION TUNING=val	Refer to HSIMSPEED on page 303
.OPTION DSCGLOB=global	Works the same as in Eldo

Eldo Commands Not Supported in HSIM

Table lists the Eldo commands not supported in HSIM.

Table 19 Eldo Commands Not Supported in HSIM

Command	Function
.A2D	Analog-to-digital converter

Table 19 Eldo Commands Not Supported in HSIM (Continued)

Command	Function
.ADDLIB	Insert a model or sub circuit file
.CALL_TCL	Call a TCL program
.CHECKBUS	Check bus values. This command can be replaced by an automatic output check in the digital vector file used by HSIM.
.CHRSIM	Input from a prior simulation. This command can be replaced by tools such as: cou2fsdb, fsdb2tbl or fsdb2pwl
.COMCHAR	Define the new comment character
.D2A	Digital to analog converter
.DISTRIB	User-defined distributions (Monte Carlo Analysis)
.DSP	Digital signal processing computation
.DSPMOD	Power spectral density computation
.FFILE S, Y, Z	Parameter output file specification
.GUESS	Initial DC analysis conditions
.INIT	Initial digital circuit conditions
.LIBFAS	Specification of a library containing FAS models
.LOAD	Use previously simulated results
.LOOP	Insert a feedback loop
.LSTB	Loop stability analysis
.LOTGROUP	Share distributions
.MODDUP	Aspire/SimPilot command
.MODLOGIC	Digital model definition
.MPRUN	Multiprocessor simulation

Table 19 Eldo Commands Not Supported in HSIM (Continued)

Command	Function
.NET	Network analysis
.NEWPAGE	Control page layout for Xelga
.NOCOM	Suppress comment lines from output file
.NOISE	Noise analysis
.NOISETRAN	Transient noise analysis
.NOTRC	Suppress netlist from an output file
.NWBLOCK	Partition netlist into Newton blocks
.OPTPWL	Accuracy by time window
.OPTWIND	Accuracy by time window
.OPTIMIZE	Circuit optimization
.OPTNOISE AC	Noise analysis
.PLOTBUS	Plotting of bus signals
.PROTECT	Netlist protection
.PZ	Pole & zero analysis
.RAMP	Automatic ramping
.RESTART	Restart simulation. The command in HSIM is .restore with the HSPICE syntax
.SAVE	Save simulation run. The command in HSIM is .store with the HSPICE syntax
.SENS	Sensitivity analysis
.SINUS	Sinusoidal voltage source
.SNF	Spot noise figure

Table 19 Eldo Commands Not Supported in HSIM (Continued)

Command	Function
.SOLVE	Sizing Facility. Identical function in HSIM, but with the HSPICE syntax.
.SUBDUP	Sub-circuit duplicate parameters
.TABLE	Value tables
.TITLE	Set title of cou file
.TF	Transfer function
.TVINCLUDE	Test vector files
.UNPROTECT	Netlist protection
.USE	Use previously simulated results
.USE_TCL	Load a TCL file
.WCASE	Worst case analysis
.WIDTH	Set printer paper width

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z