HSPICE® Simulation and Analysis User Guide

Version Y-2006.03, March 2006

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, Hypermodel, iN-Phase, in-Sync, Leda, MAST, Meta, Meta-Software, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, RapidScript, Saber, SiVL, SNUG, SolvNet, Superlog, System Compiler, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAII, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, Direct RTL, Direct Silicon Access, Discovery, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, Galaxy, Gatran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSIM^{plus}, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLlint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Protocol Compiler, PSMGen, Raphael, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

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

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

Printed in the U.S.A.

HSPICE® Simulation and Analysis User Guide, Y-2006.03

Contents

Contents

About This Manual

This manual describes how to use HSPICE to simulate and analyze your circuit designs.

Inside This Manual

This manual contains the chapters described below. For descriptions of the other manuals in the HSPICE documentation set, see the next section, The HSPICE Documentation Set.

Chapter	Description
Chapter 1, Overview	Describes HSPICE features and the simulation process.
Chapter 2, Setup and Simulation	Describes the environment variables, standard I/O files, invocation commands, and simulation modes.
Chapter 3, Input Netlist and Data Entry	Describes the input netlist file and methods of entering data.
Chapter 4, Elements	Describes the syntax for the basic elements of a circuit netlist in HSPICE or HSPICE RF.
Chapter 5, Sources and Stimuli	Describes element and model statements for independent sources, dependent sources, analog-to-digital elements, and digital-to-analog elements.
Chapter 6, Parameters and Functions	Describes how to use parameters within an HSPICE netlist.
Chapter 7, Simulation Output	Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.

Chapter	Description
Chapter 8, Initializing DC/ Operating Point Analysis	Describes DC initialization and operating point analysis.
Chapter 9, Transient Analysis	Describes how to use transient analysis to compute the circuit solution.
Chapter 10, AC Sweep and Small Signal Analysis	Describes how to perform AC sweep and small signal analysis.
Chapter 11, Linear Network Parameter Analysis	Describes how to perform an AC sweep to extract small-signal linear network parameters.
Chapter 12, Using Verilog-A	Describes how to use Verilog-A in HSPICE simulations.
Chapter 13, Simulating Variability	Introduces variability, describes how it is defined in HSPICE, and introduces the variation block.
Chapter 14, Variation Block	*Describes the use model and structure of the variation block.*
Chapter 15, Monte Carlo Analysis	*Describes Monte Carlo analysis in HSPICE.*
Chapter 16, DC Mismatch Analysis	Describes the use of DCmatch analysis.
Chapter 17, Optimization	Describes optimization in HSPICE for optimizing electrical yield.
Chapter 18, RC Reduction	Describes RC network reduction.
Chapter 19, Running Demonstration Files	Contains examples of basic file construction techniques, advanced features, and simulation tricks. Lists and describes several HSPICE and HSPICE RF input files.
Appendix A, Statistical Analysis	Describes the features available in HSPICE for statistical analysis before the Y-2006.03 release.
Appendix B, Full Simulation Examples	Contains information and sample input netlists for two full simulation examples.

Chapter	Description
Appendix C, HSPICE GUI for Windows	Describes how to use the HSPICE GUI for Windows.

The HSPICE Documentation Set

This manual is a part of the HSPICE documentation set, which includes the following manuals:

Manual	Description
HSPICE Simulation and Analysis User Guide	Describes how to use HSPICE to simulate and analyze your circuit designs. This is the main HSPICE user guide.
HSPICE Signal Integrity Guide	Describes how to use HSPICE to maintain signal integrity in your chip design.
HSPICE Applications Manual	Provides application examples and additional HSPICE user information.
HSPICE Command Reference	Provides reference information for HSPICE commands.
HPSPICE Elements and Device Models Manual	Describes standard models you can use when simulating your circuit designs in HSPICE, including passive devices, diodes, JFET and MESFET devices, and BJT devices.
HPSPICE MOSFET Models Manual	Describes standard MOSFET models you can use when simulating your circuit designs in HSPICE.
HSPICE RF Manual	Describes a special set of analysis and design capabilities added to HSPICE to support RF and high-speed circuit design.
AvanWaves User Guide	Describes the AvanWaves tool, which you can use to display waveforms generated during HSPICE circuit design simulation.

Manual	Description
HSPICE Quick Reference Guide	Provides key reference information for using HSPICE, including syntax and descriptions for commands, options, parameters, elements, and more.
HSPICE Device Models Quick Reference Guide	Provides key reference information for using HSPICE device models, including passive devices, diodes, JFET and MESFET devices, and BJT devices.

Searching Across the HSPICE Documentation Set

Synopsys includes an index with your HSPICE documentation that lets you search the entire HSPICE documentation set for a particular topic or keyword. In a single operation, you can instantly generate a list of hits that are hyperlinked to the occurrences of your search term. For information on how to perform searches across multiple PDF documents, see the *HSPICE Release Notes* (available on SolvNet at http://solvnet.synopsys.com/ReleaseNotes) or the Adobe Reader online help.

Note:

To use this feature, the HSPICE documentation files, the Index directory, and the index.pdx file must reside in the same directory. (This is the default installation for Synopsys documentation.) Also, Adobe Acrobat must be invoked as a standalone application rather than as a plug-in to your web browser.

Other Related Publications

For additional information about HSPICE, see:

- The *HSPICE Release Notes*, available on SolvNet (see Known Limitations and Resolved STARs, below)
- Documentation on the Web, which provides PDF documents and is available through SolvNet at http://solvnet.synopsys.com/DocsOnWeb
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at http://mediadocs.synopsys.com

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

- CosmosScope
- Aurora
- Raphael
- VCS

Known Limitations and Resolved STARs

You can find information about known problems and limitations and resolved Synopsys Technical Action Requests (STARs) in the *HSPICE Release Notes* in SolvNet.

To see the *HSPICE Release Notes*:

1. Go to https://solvnet.synopsys.com/ReleaseNotes. (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.)
2. Click HSPICE, then click the release you want in the list that appears at the bottom.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`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]`

Convention	Description
`...`	Indicates that parameters can be repeated as many times as necessary: *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.

Customer Support

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

Accessing SolvNet

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

To access SolvNet:

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

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

Contacting the Synopsys Technical Support Center

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

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

1

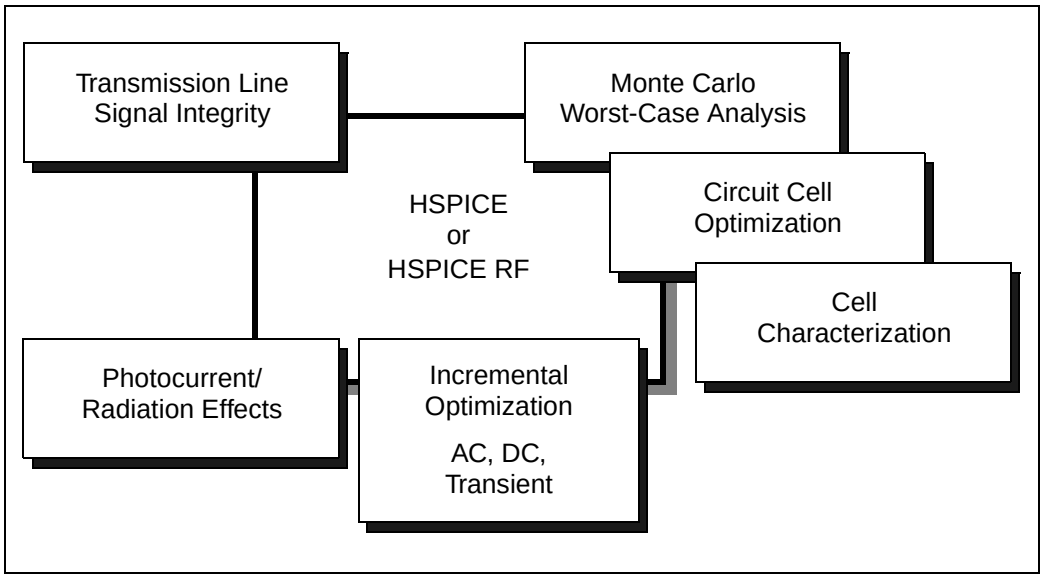
Overview

Describes HSPICE features and the simulation process.

Synopsys HSPICE is an optimizing analog circuit simulator. You can use it to simulate electrical circuits in steady-state, transient, and frequency domains.

HSPICE or HSPICE RF is unequalled for fast, accurate circuit and behavioral simulation. It facilitates circuit-level analysis of performance and yield, by using Monte Carlo, worst-case, parametric sweep, and data-table sweep analyses, and employs the most reliable automatic-convergence capability (see Figure 1).

Figure 1 Synopsys HSPICE or HSPICE RF Design Features



HSPICE or HSPICE RF forms the cornerstone of a suite of Synopsys tools and services that allows accurate calibration of logic and circuit model libraries to actual silicon performance.

The size of the circuits that HSPICE or HSPICE RF can simulate is limited only by memory. As a 32-bit application, HSPICE can address a maximum of 2Gb or 4Gb of memory, depending on your system.

For a description of commands that you can include in your HSPICE netlist, see the "Netlist Commands" chapter in the *HSPICE Command Reference*.

HSPICE Varieties

Synopsys HSPICE is available in two varieties:

- HSPICE
- HSPICE RF

Like traditional SPICE simulators, HSPICE is Fortran-based, but it is faster and has more capabilities than typical SPICE simulators. HSPICE accurately simulates, analyzes, and optimizes circuits, from DC, to microwave frequencies that are greater than 100 GHz. HSPICE is ideal for cell design and process modeling. It is also the tool of choice for signal-integrity and transmission-line analysis.

HSPICE RF is a newer, C++ version of the traditional Fortran-based HSPICE. Many (but not all) HSPICE simulation capabilities have been implemented in HSPICE RF, and HSPICE RF offers some new capabilities that are not in available in traditional HSPICE.

HSPICE RF usually produces results at the desired level of accuracy in a shorter time than HSPICE requires for the same level of accuracy. HSPICE RF can also perform HSPICE simulations of radio-frequency (RF) devices, which HSPICE does *not* support.

This guide describes all of the features that HSPICE supports. HSPICE RF supports some—but not all—of these features as well. For descriptions of HSPICE RF features and a list of the differences between HSPICE and HSPICE RF, see the "HSPICE RF Features and Functionality" chapter in the *HSPICE RF User Guide*.

Features

Synopsys HSPICE or HSPICE RF is compatible with most SPICE variations, and has the following additional features:

- Superior convergence
- Accurate modeling, including many foundry models
- Hierarchical node naming and reference
- Circuit optimization for models and cells, with incremental or simultaneous multiparameter optimizations in AC, DC, and transient simulations
- Interpreted Monte Carlo and worst-case design support
- Input, output, and behavioral algebraics for cells with parameters
- Cell characterization tools, to characterize standard cell libraries
- Geometric lossy-coupled transmission lines for PCB, multi-chip, package, and IC technologies
- Discrete component, pin, package, and vendor IC libraries
- Interactive graphing and analysis of multiple simulation waveforms by using with AvanWaves and CosmosScope
- Flexible license manager that allocates licenses intelligently based on run status and user-specified job priorities you specify.

  If you suspend a simulation job, the load sharing facility (LSF) license manager signals HSPICE to release that job's license. This frees the license for another simulation job, or so the stopped job can reclaim the license and resume. You can also prioritize simulation jobs you submit; LSF automatically suspends low-priority simulation jobs to run high-priority jobs. When the high-priority job completes, LSF releases the license back to the lower-priority job, which resumes from where it was suspended.
- A number of circuit analysis types (see Figure 2) and device modeling technologies (see Figure 3).

Figure 2 Synopsys HSPICE or HSPICE RF Circuit Analysis Types

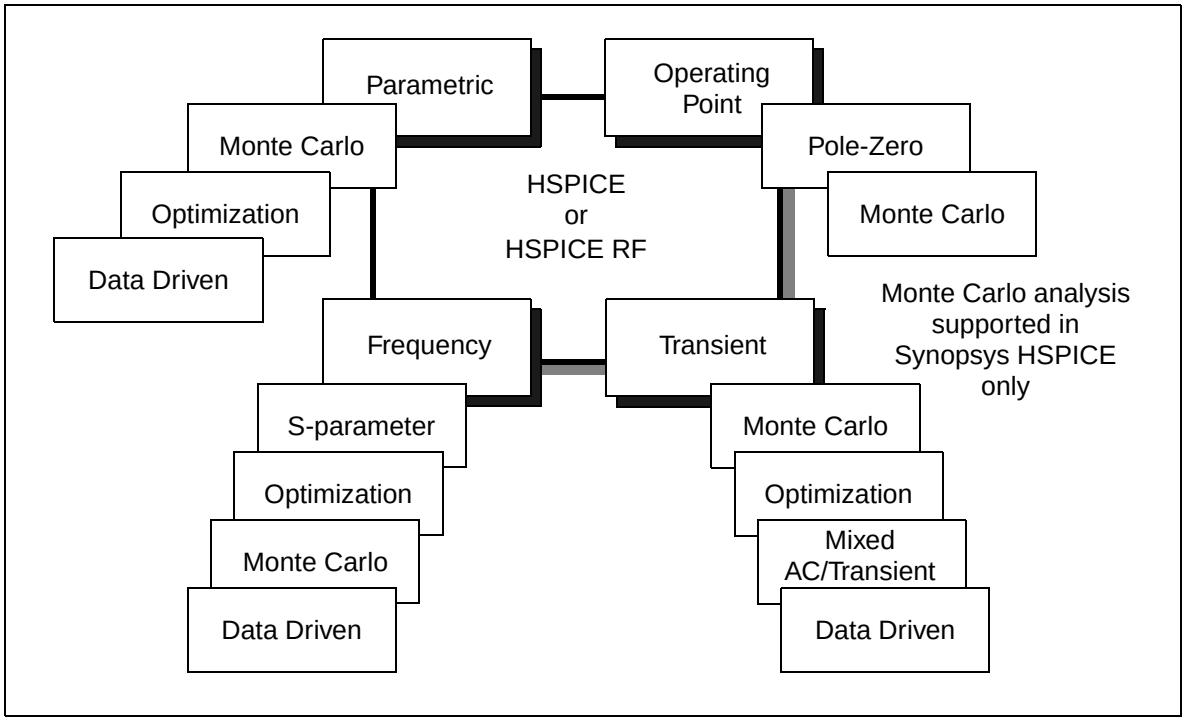
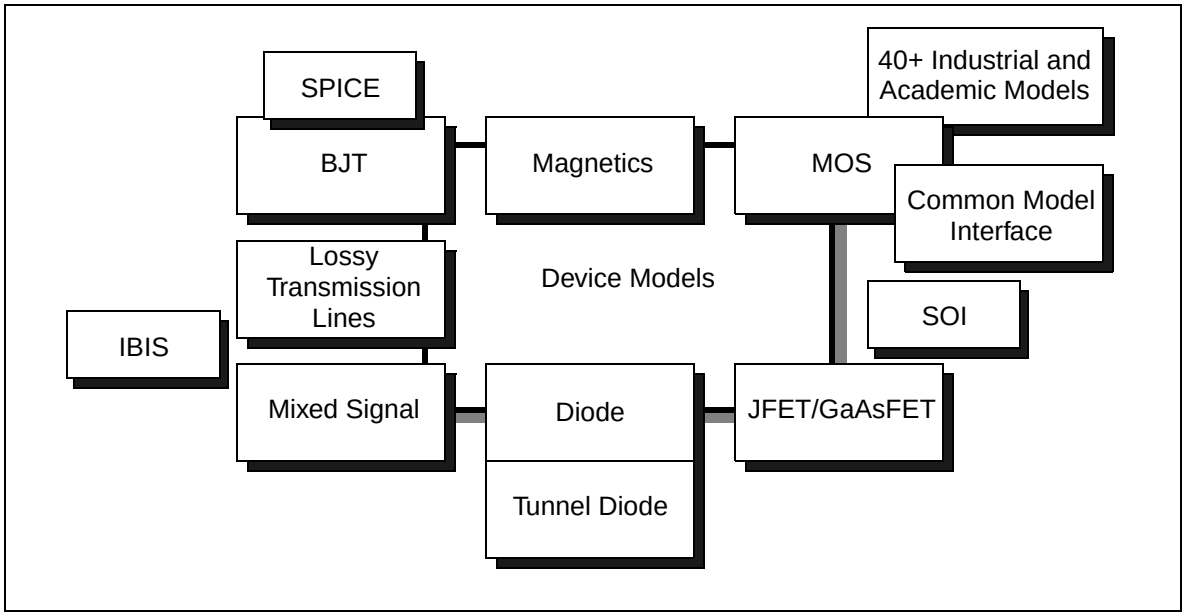


Figure 3 Synopsys HSPICE Modeling Technologies



HSPICE Features for Running Higher-Level Simulations

Simulations at the integrated circuit level and at the system level require careful planning of the organization and interaction between transistor models and subcircuits. Methods that worked for small circuits might have too many limitations when applied to higher-level simulations.

You can use the following HSPICE or HSPICE RF features to organize how simulation circuits and models run:

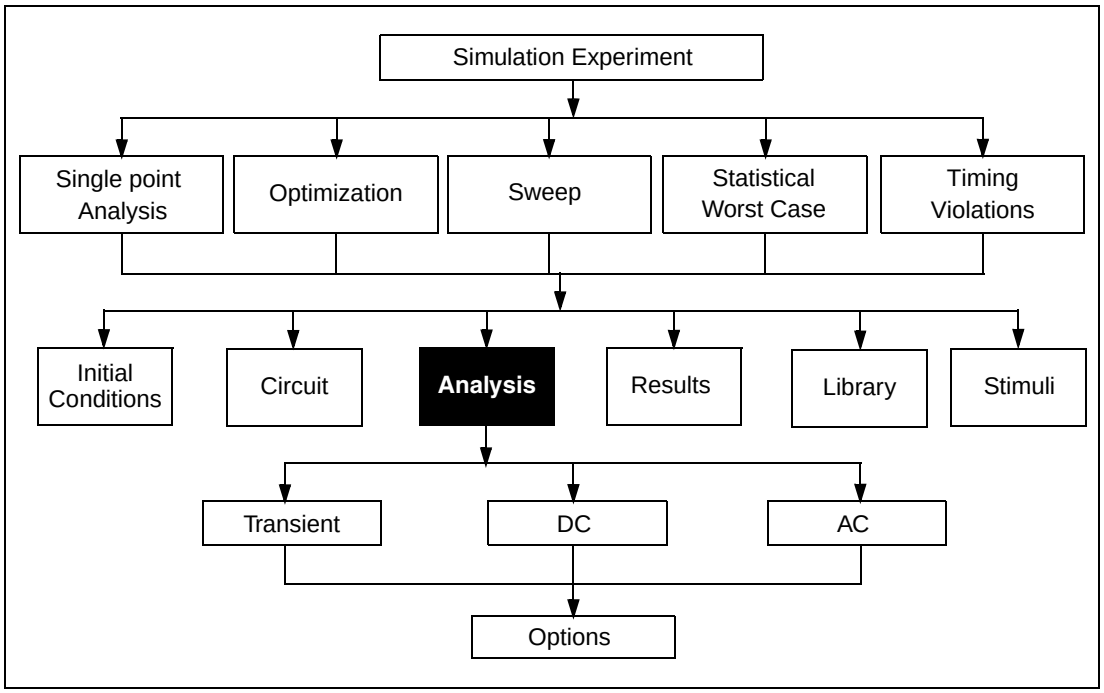
- Explicit include files – `.INCLUDE` statement.
- Implicit include files – `.OPTION SEARCH='lib_directory'` (HSPICE only).
- Algebraics and parameters for devices and models – `.PARAM` statement.
- Parameter library files – `.LIB` statement.
- Automatic model selector – `LMIN`, `LMAX`, `WMIN`, and `WMAX` model parameters.
- Parameter sweep – sweep analysis statements.
- Statistical analysis – sweep monte analysis statements (HSPICE only).
- Multiple alternative – `.ALTER` statement (HSPICE only).
- Automatic measurements – `.MEASURE` statement.
- Condition-controlled netlists (`IF-ELSEIF-ELSE-ENDIF` statements).

Simulation Structure

Experimental Methods Supported by HSPICE

Typically, you use experiments to analyze and verify complex designs. These experiments can be simple sweeps, more complex Monte Carlo and optimization analyses (HSPICE only), or setup and hold violation analyses of DC, AC, and transient conditions.

Figure 4 Simulation Program Structure



For each simulation experiment, you must specify tolerances and limits to achieve the desired goals, such as optimizing or centering a design. Common factors for each experiment are:

- process
- voltage
- temperature
- parasitics

HSPICE or HSPICE RF supports two experimental methods:

- Single point – a simple procedure that produces a single result, or a single set of output data.
- Multipoint – performs an analysis (single point) sweep for each value in an outer loop (multipoint) sweep.

The following are examples of multipoint experiments:

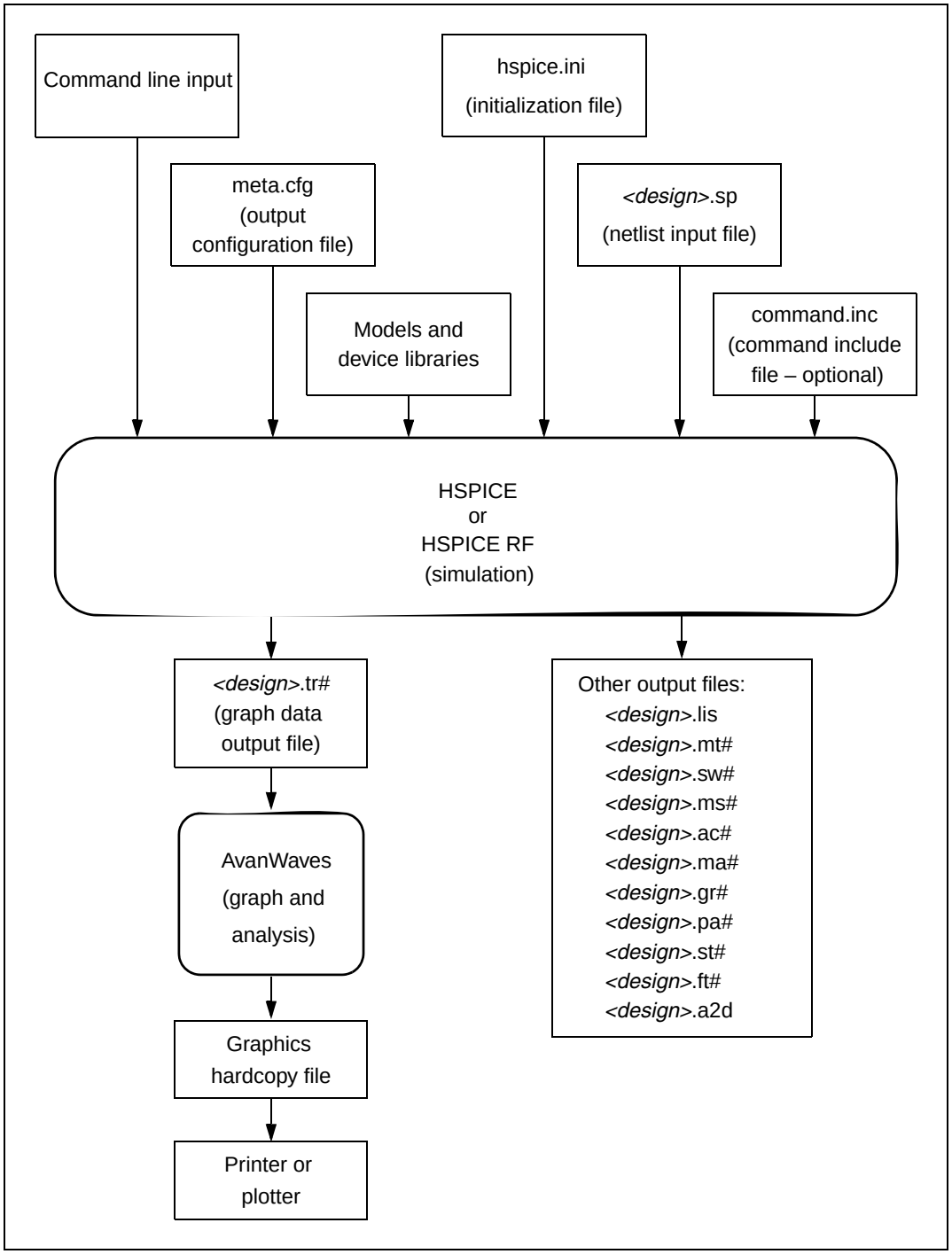
- Process variation – Monte Carlo or worst-case model parameter variation (HSPICE only).
- Element variation – Monte Carlo (HSPICE only) or element parameter sweeps.

- Voltage variation – VCC, VDD, or substrate supply variation.
- Temperature variation – design temperature sensitivity.
- Timing analysis – basic timing, jitter, and signal integrity analysis.
- Parameter optimization – balancing complex constraints, such as speed versus power, or frequency versus slew rate versus offset (analog circuits).

HSPICE Data Flow

HSPICE or HSPICE RF accepts input and simulation control information from several different sources. They can output results in a number of convenient forms for review and analysis. Figure 5 shows the overall data flow.

Figure 5 Overview of Data Flow



To simulate a design in HSPICE, you do the following:

1. To begin design entry and simulation, create an input netlist file.

   Most schematic editors and netlisters support the SPICE or HSPICE hierarchical format.

2. HSPICE or HSPICE RF executes the analyses specified in the input file.

3. HSPICE or HSPICE RF stores the simulation results requested in either an output listing file or (if you specified `.OPTION POST`) a graph data file.

   If you specified POST, HSPICE or HSPICE RF stores the circuit solution (in either steady state, time, or frequency domain).

4. To view or plot the results for any nodal voltage or branch current, use a high-resolution graphic output terminal or laser printer.

   HSPICE provides a complete set of print and plot variables for viewing analysis results. HSPICE RF supports some, but not all, HSPICE print variables.

The HSPICE or HSPICE RF programs include a textual command line interface. For example, to execute the program, enter the `hspice` or `hspicext` command, the input file name, and the desired options. You can use the command line at the prompt in a Unix shell, or a Windows command prompt, or (for HSPICE only) click on an icon in a Windows environment.

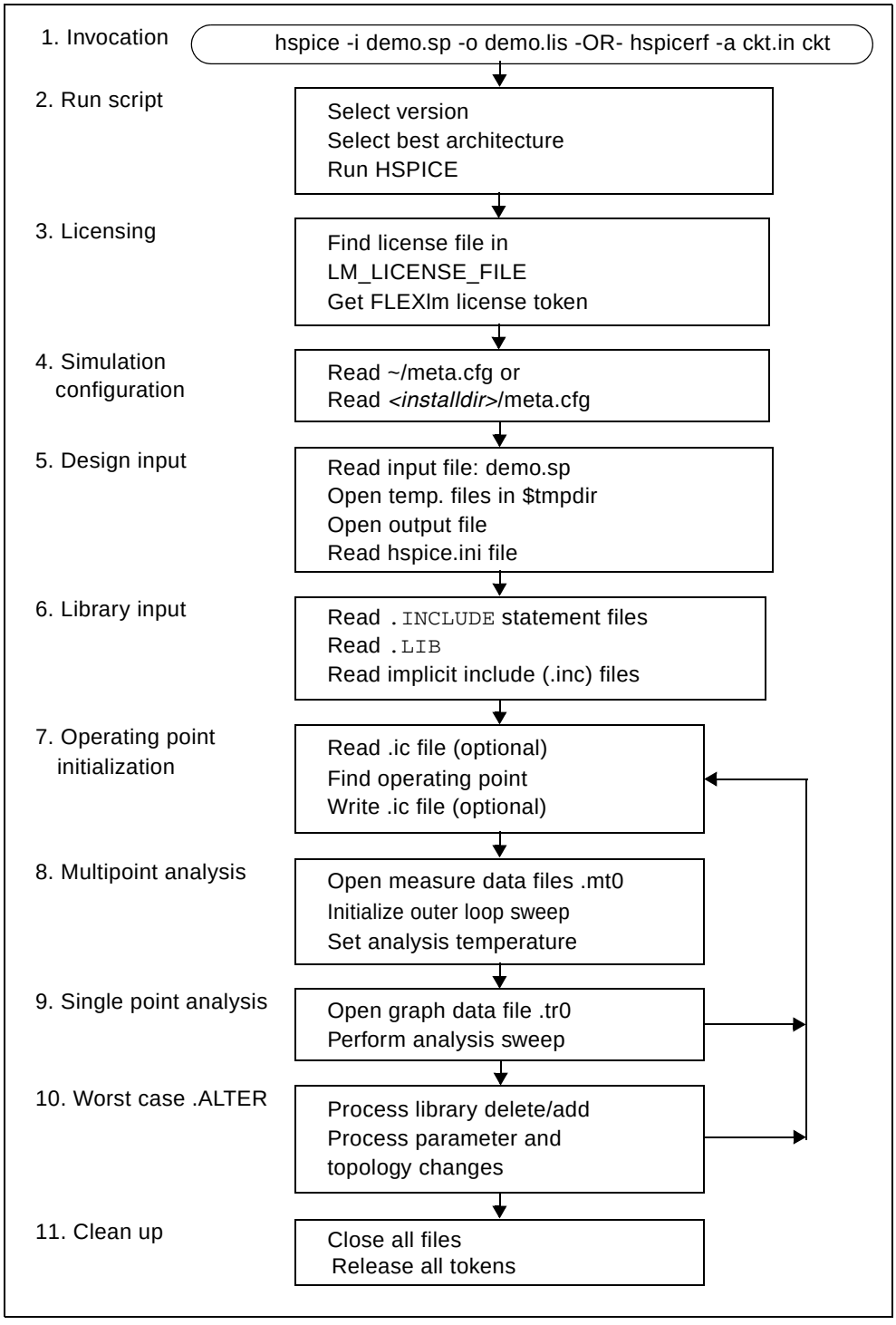
You can specify whether the HSPICE or HSPICE RF program simulation output appears in an output listing file, or in a graph data file (HSPICE only). HSPICE or HSPICE RF creates standard output files to describe initial conditions (.ic extension) and output status (.st0 extension). In addition, HSPICE or HSPICE RF creates various output files, in response to user-defined input options—for example, HSPICE creates a *<design>*.tr0 file, in response to a `.TRAN` transient analysis statement.

The default waveform display tool CosmosScope. See the *CosmosScope User Guide* for instructions about how to use CosmosScope.

Simulation Process Overview

Figure 6 shows the HSPICE or HSPICE RF simulation process.

Figure 6 Simulation Process



2

Setup and Simulation

Describes the environment variables, standard I/O files, invocation commands, and simulation modes.

For descriptions of individual HSPICE commands mentioned in this chapter, see the *HSPICE Command Reference*.

Setting Environment Variables

Before using HSPICE, you need to set these environment variables

- `LM_LICENSE_FILE`—Specifies the path to the license file (required)
- `META_QUEUE`—Enables HSPICE licenses to be queued
- `tmpdir` (UNIX), `TEMP` or `TMP` (Windows)—Allows you to control the location of the temporary files

Setting License Variables

HSPICE or HSPICE RF requires you to set the `LM_LICENSE_FILE` environment variable. This variable specifies the full path to the license.dat license file. Set the `LM_LICENSE_FILE` environment variable to point to the HSPICE and HSPICE RF license file.

For example, if your HSPICE RF license file is in:

```
/usr/cad/hspicext/license.dat
```

And your HSPICE license file is in:

```
/usr/cad/hspice/license.dat
```

Then you would enter:

```
setenv LM_LICENSE_FILE /usr/cad/hspicext/license.dat:\
/usr/cad/hspice/license.dat
```

You can also set the variable `port@hostname` to point to a license file on a server.

- If you are using the C shell, add the following line to the .cshrc file:

  ```
  setenv LM_LICENSE_FILE port@hostname
  ```

- If you are using the Bash or Bourne shell, add these lines to the .bashrc or .profile file:

  ```
  LM_LICENSE_FILE=port@hostname
  export LM_LICENSE_FILE
  ```

The port and host name variables correspond to the TCP port and license server host name specified in the SERVER line of the Synopsys license file.

Note:

> To ensure better performance, it is recommended that you use `port@hostname` rather than using the path to the license file.

Each license file can contain licenses for many packages from multiple vendors. You can specify multiple license files by separating each entry. For UNIX, use a colon (:) and for Windows, use a semicolon (;).

For details about setting license file environment variable, see “Setting Up HSPICE for Each User” in the *Installation Guide*.

License Queuing

Setting the optional `META_QUEUE` environment variable to 1 enables HSPICE licenses to be queued:

```
setenv META_QUEUE 1
```

The licensing queuing works as follows: If you have five HSPICE floating licenses and all five licenses are checked out with the `META_QUEUE` environment variable enabled, then the next job submitted waits in the queue until a license is available (when one of the previous five jobs finishes). When `META_QUEUE` is enabled and all available licenses are in use, an error message is issued that says no licenses are available.

If you have more than one HSPICE token (`INCREMENT` line) and the version dates are different, only the first token in your license file is queued. FLEXlm

queues the first increment line that satisfies the request. If you have two increment lines with different versions, two license pools are created on the server. When you issue the queuing request, the server attempts to satisfy the request, but if it is not possible, the server queues the first increment line that satisfies the request. Once that particular increment line is queued, it waits for that increment line to become free. The server does not continually look for any other line that satisfies this request. This is normal operation for FLEXlm.

Standard Input Files

This section describes the standard input files to HSPICE or HSPICE RF.

Design and File Naming Conventions

The design name identifies the circuit and any related files, including:

- Schematic and netlist files.
- Simulator input and output files.
- Design configuration files.
- Hardcopy files.

HSPICE, HSPICE RF, and AvanWaves extract the design name from their input files, and perform actions based on that name. For example, AvanWaves reads the <*design*>.cfg configuration file to restore node setups used in previous AvanWaves runs.

HSPICE, HSPICE RF, and AvanWaves read and write files related to the current circuit design. Files related to a design usually reside in one directory. The output file is stdout on Unix platforms, which you can redirect.

Table 1 lists input file types, and their standard names. The sections that follow describe these files.

Table 1 Input Files

Input File Type	File Name
Output configuration file	meta.cfg
Initialization file	hspice.ini
DC operating point initial conditions file	<*design*>.ic#

Table 1 Input Files (Continued)

Input File Type	File Name
Input netlist file	<*design*>.sp
Library input file	<*library_name*>
Analog transition data file	<*design*>.d2a

Output Configuration File

You use the output configuration file to set up the printer, plotter, and terminal. It includes a line (default_include=<*filename*>) to set up a path to the default .ini file (for example, hspice.ini).

The default include filename is case-sensitive (except for the PC and Windows versions of HSPICE).

Initialization File

You use the initialization file to specify user defaults. If the run directory contains an hspice.ini file, HSPICE or HSPICE RF includes its contents at the top of the input file.

To include initialization files, you can define default_include=<*filename*> in a command.inc or meta.cfg file.

You can use an initialization file to set options (for `.OPTION` statements) and to access libraries.

DC Operating Point Initial Conditions File

The DC operating point initial conditions file, <*design*>.ic#, is an optional input file that contains initial DC conditions for particular nodes. You can use this file to initialize DC conditions, by using either a `.NODESET` or an `.IC` statement.

A `.SAVE` statement can also create a <*design*>.ic# file. A subsequent `.LOAD` statement initializes the circuit to the DC operating point values specified in this file.

Input Netlist File

The input netlist file, <*design*>.sp, contains the design netlist. Optionally, it can also contain statements specifying the type of analysis to run, type of output desired, and what library to use.

Library Input File

You use <*library_name*> files to identify libraries and macros that need to be included for simulating <*design*>.sp.

Analog Transition Data File

When you run HSPICE in standalone mode, a <*design*>.d2a file contains state information for a U Element mixed-mode simulation.

Standard Output Files

This section describes the standard output files from HSPICE. For informaton about the standard output file from HSPICE RF, see section HSPICE RF Output File Types in the *HSPICE RF Manual*. The various types of output files produced are listed in Table 2.

Table 2 HSPICE Output Files and Suffixes

Output File Type	Extension
AC analysis measurement results	.ma#[a]
AC analysis results (from `.POST` statement)	.ac#
DC analysis measurement results	.ms#
DC analysis results (from `.POST` statement)	.sw#
Digital output	.a2d
FFT analysis graph data (from `FFT` statement)	.ft#

Table 2 HSPICE Output Files and Suffixes (Continued)

Output File Type	Extension
Hardcopy graph data (from meta.cfg PRTDEFAULT)	.gr#[b] †
Operating point information (from .OPTION OPFILE statement)	.dp#
Operating point node voltages (initial conditions)	.ic#
Output listing	.lis, or user-specified
Output status	.st#
Output tables (from .DCMATCH OUTVAR statement)	.dm#
Subcircuit cross-listing (HSPICE only; not supported in HSPICE RF)	.pa#
Transient analysis measurement results	.mt#
Transient analysis results (from .POST statement)	.tr#

a. # can be either a sweep number or a hardcopy file number. For .ac#, .dp#, .dm#, .ic#, .st#, .sw#, and .tr# files, # is from 0 through 9999.

b. Requires a .GRAPH statement, or a pointer to a file in the meta.cfg file. The PC version of HSPICE does not generate this file.

AC Analysis Results File

HSPICE writes AC analysis results to file `<output_file>.ac#`, where `#` is `0-9999`, according to your specifications following the `.AC` statement. These results list the output variables as a function of frequency.

AC Analysis Measurement Reults File

HSPICE writes AC analysis measurement results to file `<output_file>.ma#` when the input file includes a `.MEASURE AC` statement.

DC Analysis Results File

HSPICE writes DC analysis results to file `<output_file>.sw#`, where `#` is `0-9999`, when the input file includes a `.DC` statement. This file contains the results of the applied stepped or swept DC parameters defined in that statement. The results can include noise, distortion, or network analysis.

DC Analysis Measurement Results File

HSPICE writes DC analysis measurement results to file `<output_file>.ms#` when the input file includes a `.MEASURE DC` statement.

Digital Output File

The digital output file, <*design*>.a2d, contains data that the A2D conversion option of the U element converted to digital form.

FFT Analysis Graph Data File

The FFT analysis graph data file, `<output_file>.ft#`, contains the graphical data needed to display the FFT analysis waveforms.

Hardcopy Graph Data File

HSPICE writes hardcopy graph data to file `<output_file>.gr#` when the input file includes a `.GRAPH` statement. The file produced is in the form of a printer file, typically in Adobe PostScript or HP PCL format. This facility is not available in the PC version of HSPICE.

Operating Point Information File

HSPICE writes operating point information to file <*design*>.dp*#* when the input file includes an `.OPTION OPFILE=1` statement.

Operating Point Node Voltages File

HSPICE writes operati.96 T67350 0 TD0.0024 Tc-0.0151 Tw[ingpo(intnNodev)2li.96 27.650

Output Status File

The output status file, <output_file>.st#, where # is 0-9999, contains the following runtime reports:

- Start and end times for each CPU phase.
- Options settings, with warnings for obsolete options.
- Status of preprocessing checks for licensing, input syntax, models, and circuit topology.
- Convergence strategies that HSPICE uses on difficult circuits.

You can use the information in this file to diagnose problems, particularly when communicating with Synopsys Customer Support.

Output Tables

The .DCMATCH output tables file, <output_file>.dm#, contains the variability data from analysis.

Subcircuit Cross-Listing File

If the input netlist includes subcircuits, HSPICE automatically generates a subcircuit cross-listing file, <output_file>.pa#, where # is 0-9999. This file relates the subcircuit node names, in the subcircuit call, to the node names used in the corresponding subcircuit definitions. In HSPICE RF, you cannot replicate output commands within subcircuit (subckt) definitions.

Transient Analysis Measurement Results File

HSPICE writes transient analysis measurement results to file <output_file>.mt# when the input file includes an .MEASURE TRAN statement.

Transient Analysis Results File

Both HSPICE and HSPICE RF place the results of transient analysis in file <output_file>.tr#, where # is 0-9999, as set forth in the -n command-line argument. This file lists the numerical results of transient analysis.

A `.TRAN` statement in the input file, together with an `.OPTION POST` statement, creates this post-analysis file.

If the input file includes an `.OPTION POST` statement, then the output file contains simulation output suitable for a waveform display tool.

Running HSPICE Simulations

Use the following syntax to start HSPICE:

```
hspice <-i <path/input_file>> <-o <path/output_file>>
<-n number> <-html <path/html_file>> <-b> <-d>
<-C <path/input_file>> <-I> <-K> <-L command_file>
<-S> <-mt number> <-meas measurefile> <-hdl filename>
<-hdlpath pathname> <<name> -vamodel <name2>...>
```

For a description of the `hspice` command syntax and arguments, see "HSPICE Command Syntax" in the *HSPICE Command Reference*.

When your invoke an HSPICE simulation, the following sequence of events occurs:

1. Invocation.

   For example, at the shell prompt, enter:

   ```
   hspice demo.sp > demo.out &
   ```

   This command invokes the UNIX `hspice` shell command on input netlist file demo.sp and directs the output listing to file demo.out. The "&" character at the end of the command invokes HSPICE in the background, so that you can continue to use the window and keyboard while HSPICE runs.

2. Script execution.

   The `hspice` shell command starts the HSPICE executable from the appropriate architecture (machine type) directory. The UNIX run script launches a HSPICE simulation. This procedure establishes the environment for the HSPICE executable. The script prompts for information, such as the platform that you are using, and the version of HSPICE to run. (Available versions are determined when you install HSPICE.)

3. Licensing.

   HSPICE supports the FLEXlm licensing management system. When you use FLEXlm licensing, HSPICE reads the `LM_LICENSE_FILE` environment variable to find the location of the license.dat file.

If HSPICE cannot authorize access, the job terminates at this point, and prints an error message in the output listing file.

4. Simulation configuration.

   HSPICE reads the appropriate meta.cfg file. The search order for the configuration file is the user login directory, and then the product installation directory.

5. Design input.

   HSPICE opens the input netlist file demo.sp. If this file does not exist, a no input data error appears in the output listing file.

   (UNIX) HSPICE opens three scratch files in the /tmp directory. To change this directory, reset the `tmpdir` environment variable in the HSPICE command script.

   (Windows) HSPICE opens three scratch files in the c:\<path>\TEMP (or \TMP) directory. To change this directory, reset the `TEMP` or `TMP` environment variable in the HSPICE command script.

   HSPICE opens the output listing file demo.out for writing. If you do not own the current directory, HSPICE terminates with a file open error.

   Here's an example of a simple HSPICE input netlist:

```
Inverter Circuit
.OPTION LIST NODE POST
.TRAN 200P 20N SWEEP TEMP -55 75 10
.PRINT TRAN V(IN) V(OUT)
M1 VCC IN OUT VCC PCH L=1U W=20U
M2 OUT IN 0 0 NCH L=1U W=20U
VCC VCC 0 5
VIN IN 0 0 PULSE .2 4.8 2N 1N 1N 5N 20N
CLOAD OUT 0 .75P
.MODEL PCH PMOS
.MODEL NCH NMOS
.ALTER
CLOAD OUT 0 1.5P
.END
```

6. Library input.

   HSPICE reads any files that you specified in `.INCLUDE` and `.LIB` statements.

7. Operating point initialization.

   HSPICE reads any initial conditions that you specified in `.IC` and `.NODESET` statements, finds an operating point (that you can save with a `.SAVE` statement), and writes any operating point information that you requested.

8. Multipoint analysis.

   HSPICE performs the experiments specified in analysis statements. In the above example, the `.TRAN` statement causes HSPICE to perform a multipoint transient analysis for 20 ns for temperatures ranging from -55°C to 75°C, in steps of 10°C.

9. Single-point analysis.

   HSPICE performs a single or double sweep of the designated quantity, and produces one set of output files.

10. Worst-case `.ALTER`.

    You can vary simulation conditions, and repeat the specified single or multipoint analysis. The above example changes `CLOAD` from 0.75 pF to 1.5 pF, and repeats the multipoint transient analysis.

11. Normal termination.

    After you complete the simulation, HSPICE closes all files it opened and releases all license tokens.

Running HSPICE RF Simulations

Use the following syntax to invoke HSPICE RF:

```
hspicerf [-a] inputfile [outputfile] [-h] [-v]
```

For a description of the `hspicerf` command syntax and arguments, see “HSPICE RF Command Syntax” in the *HSPICE Command Reference*.

Running HSPICE Interactively

When HSPICE is in the interactive mode, you can then use these HSPICE commands at the HSPICE prompt to help you simulate circuits interactively:

ac [...statement]	cd
dc [...statement]	edit
help	info outflag
input	list [lineno]
load filename	ls [directory]
measure [statement]	op
print <tran/ac/dc>,v/vm/vr/vi/vp/vdb>	pwd
quit	run
save <netlist/command> filename	set outflag <true/false>
tran [...statement]	

To Start Interactive Mode

Starting HSPICE in the interactive mode lets you use a subset of commands to simulate your circuits interactively. To invoke the interactive mode, enter:

```
hspice -I
```

You can also use the `help` command at the HSPICE prompt for an annotated list of the commands supported in the interactive mode.

The interactive mode also supports saving commands into a script file. To save the commands that you use, and replay them later, enter:

```
HSPICE > save command <filename>
```

To Run a Command File in Interactive Mode

To run the command you have saved in a command file, enter:

```
hspice -I -L <filename>
```

To Quit Interactive Mode

To exit the interactive mode and return to the system prompt, enter:

```
HSPICE > quit
```

Running Multithreading HSPICE Simulations

HSPICE simulations include device model evaluations and matrix solutions. You can run model evaluations concurrently on multiple CPUs, by using multithreading to significantly improve simulation performance. Model evaluation dominates most of the time. To determine how much time HSPICE spends evaluating models and solving matrices, specify `.OPTION ACCT=2` in the netlist.

By using multithreading, you can speed-up simulations with no loss of accuracy. Multithreading gives the best results for circuit designs that contain many MOSFET, JFET, diode, or BJT models in the netlist.

To Run Multithreading

To run multithreading on UNIX platforms, enter:

```
hspice -mt number -i <input_file> -o <output_file>
```

To run multithreading on Windows platforms, enter:

```
hspice_mt.exe -mt number -i <input_file> -o <output_file>
```

- If you omit the *number* parameter, an error message results. You must include this parameter.
- If you specify a *number* parameter that is larger than the number of available CPUs, then HSPICE sets the number of threads equal to the number of available CPUs.

For additional information about command-line options, see “HSPICE Command Syntax” in the *HSPICE Command Reference*.

In Windows NT Explorer:

1. Double-click the hsp_mt application icon.
2. Select the File/Simulate button, to select the input netlist file.

In Windows, the program automatically detects the number of available processors.

Under the Synopsys HSPICE User Interface (HSPUI):

1. Select the correct hsp_mt.exe version in the Version combo box.
2. Select the correct number of CPUs in the MT option box.
3. Click the Open button to select the input netlist file.
4. Click the Simulate button to start the simulation.

Performance Improvement Estimations

For HSPICE-MT, the CPU time is:

```
Tmt=Tserial + Tparallel/Ncpu + Toverhead
```

Where:

`Tserial` represents HSPICE calculations that are not threaded.

`Tparallel` represents threaded HSPICE calculations.

`Ncpu` is the number of CPUs used.

`Toverhead` is the overhead from multithreading. Typically, this represents a small fraction of the total runtime.

For example, for a 151-stage NAND ring oscillator using LEVEL 49, `Tparallel` is about 80% of `T1cpu` (the CPU time associated with a single CPU) if you run with two threads on a multi-CPU machine. Ideally, assuming `Toverhead=0`, you can achieve a speedup of:

Using HSPICE in Client/Server Mode

When you run many small simulation cases, you can use the client/server mode to improve performance. This performance improvement occurs because you check out and check in an HSPICE license only once. This is an effective measure when you characterize cells.

To Start Client/Server Mode

Starting the client/server mode creates an HSPICE server and checks out an HSPICE license. To start the client/server mode, enter:

```
hspice -C
```

Server

The server name is a specific name connected with the machine on which HSPICE runs. When you create the server, HSPICE also generates a hidden .hspicecc directory in your home directory. HSPICE places some related files in this directory, and removes them when the server exits.

HSPICE Client/Server mode does not let one user create several servers on the same machine.

When you create a server, the output on the screen is:

```
**************************************
*Starting HSPICE Client/Server Mode...*
**************************************
Checking out HSPICE license...
HSPICE license has been checked out.
******************************************
*Welcome to HSPICE Client/Server Mode!*
******************************************
```

After you create the server, it automatically runs in the background.

If the server does not receive any request from a client for 12 hours, the server releases the license, and exits automatically.

Client

The client can send a request to the server to ask whether an HSPICE license has been checked out, or to kill the server.

- If the request is to check the license status, the server checks whether an HSPICE license has been checked out, and replies to the client. The syntax of this request is:

```
hspice -C casename.sp
```

Where *casename* is the name of the circuit design to simulate.

- If the client receives `ok`, it begins to simulate the circuit design.
- If the client receives `no`, it exits.
- If the server receives several requests at the same time, it queues these requests, and process them in the order that the server received them.
- If HSPICE does not find a server, it creates a server first. Then the server checks out an HSPICE license, and simulates the circuit.
- If the request is to kill the server, the server releases the HSPICE license and other sources, and exits.

  When you kill the server, any simulation cases that are queued on that server do not run, and the server's name disappears from the hidden .hspicecc directory in your home directory.

If you do not specify an output file, HSPICE directs output to the client terminal. Use the following syntax to redirect the output to a file, instead of to the terminal:

```
hspice -C casename.sp > <output_file>
```

Note:

HSPICE RF does not support PKG and EBD simulation.

To Simulate a Netlist in Client/Server Mode

Once you have started the client/server mode, which automatically checks out an HSPICE license, you can run simulations. To simulate a netlist in client/server mode, enter:

```
hspice -C <path/input_file>
```

Note:

This mode also supports other HSPICE command line options. For a description of the options shown, see "HSPICE Command Syntax" in the *HSPICE Command Reference*.

To Quit Client/Server Mode

Quitting the client/server mode releases the HSPICE license and exits HSPICE. To exit the client/server mode, enter:

```
hspice -C -K
```

Running HSPICE to Calculate New Measurements

When you want to calculate new measurements from previous simulation results produced by HSPICE, you can rerun HSPICE.

To Calculate New Measurements

To get new measurements from a previous simulation, enter:

```
hspice -meas measurefile -i <wavefile> <-o <outputfile>>
```

For a description of the options shown, see "HSPICE Command Syntax" in the *HSPICE Command Reference*.

3

Input Netlist and Data Entry

Describes the input netlist file and methods of entering data.

For descriptions of individual HSPICE commands referenced in this chapter, see the "Netlist Commands" chapter in the *HSPICE Command Reference*.

Input Netlist File Guidelines

HSPICE and HSPICE RF operate on an input netlist file, and store results in either an output listing file or a graph data file. An input file, with the name *<design>.sp,* contains the following:

- Design netlist (subcircuits, macros, power supplies, and so on).
- Statement naming the library to use (optional).
- Specifies the type of analysis to run (optional).
- Specifies the type of output desired (optional).

An input filename can be up to 1024 characters long. The input netlist file cannot be in a packed or compressed format.

To generate input netlist and library input files, HSPICE or HSPICE RF uses either a schematic netlister or a text editor.

Statements in the input netlist file can be in any order, except that the first line is a title line, and the last `.ALTER` submodule must appear at the end of the file and before the `.END` statement.

Note:

If you do not place an `.END` statement and a `[Return]` at the end of the input netlist file, HSPICE or HSPICE RF issues an error message.

Netlist input processing is case insensitive, except for file names and their paths. HSPICE and HSPICE RF do not limit the identifier length, line length, or file size.

Input Line Format

- The input reader can accept an input token, such as:
  - a statement name.
  - a node name.
  - a parameter name or value.

    Any valid string of characters between two token delimiters is a token. You can use a character string as a parameter value in HSPICE, but not in HSPICE RF. See Delimiters on page 32.
- An input statement, or equation can be up to 1024 characters long.
- HSPICE or HSPICE RF ignores differences between upper and lower case in input lines, except in quoted filenames.
- To continue a statement on the next line, enter a plus (+) sign as the first non-numeric, non-blank character in the next line.
- To indicate “to the power of” in your netlist, use two asterisks (**). For example, `2**5` represents two to the fifth power (2^5)
- To continue all HSPICE

- Names are input tokens. Token delimiters must precede and follow names. See “Delimiters” below.
- Names can be up to 1024 characters long and are not case-sensitive.
- Do not use any of the time keywords as a parameter name or node name in your netlist.
- The following symbols are reserved operator keywords: , () = “ ‘

  Do not use these symbols as part of any parameter or node name that you define. Using any of these reserved operator keywords as names causes a syntax error, and HSPICE or HSPICE RF stops immediately.

First Character

The first character in every line specifies how HSPICE and HSPICE RF interprets the remaining line. Table 3lists and describes the valid characters.

Table 3 First Character Descriptions

Line	If the First Character is...	Indicates
First line of a netlist	Any character	Title or comment line. The first line of an included file is a normal line and not a comment.
Subsequent lines of netlist, and all lines of included files	. (period)	Netlist keyword. For example, .TRAN 0.5ns 20ns
	c, C, d, D, e, E, f, F, g, G, h, H, i, I, j, J, k, K, l, L, m, M, q, Q, r, R, s, S, v, V, w, W	Element instantiation
	* (asterisk) # (number)	Comment line (HSPICE) Comment line (HSPICE RF)
	+ (plus)	Continues previous line

Delimiters

- An input token is any item in the input file that HSPICE or HSPICE RF recognizes. Input token delimiters are: tab, blank, comma (,), equal sign (=), and parentheses ( ).
- Single (‘) or double quotes (“) delimit expressions and filenames.
- Colons (:) delimit element attributes (for example, `M1:VGS`).
- Periods (.) indicate hierarchy. For example, `X1.X2.n1` is the n1 node on the X2 subcircuit of the X1 circuit.

Node Identifiers

Node identifiers can be up to 1024 characters long, including periods and extensions. Node identifiers are used for node numbers and node names.

- Node numbers are valid in the range of 0 through 9999999999999999 (1-1e16).
- Leading zeros in node numbers are ignored.
- Trailing characters in node numbers are ignored. For example, `node 1A` is the same as `node 1`.
- A node name can begin with any of these characters:

  `! # % * / < > _ ? | . &`

  For additional information, see Node Naming Conventions on page 44.
- To make node names global across all subcircuits, use a `.GLOBAL` statement.
- The `0`, `GND`, `GND!`, and `GROUND` node names all refer to the global HSPICE or HSPICE RF ground. Simulation treats nodes with any of these names as a ground node, and produces v(0) into the output files.

Instance Names

The names of element instances begin with the element key letter (see Table 4), except in subcircuits where instance names begin with X. (Subcircuits are sometimes called macros or modules.) Instance names can be up to 1024

characters long. The `.OPTION LENNAM` defines the length of names in printouts (default=8).

Table 4 Element Identifiers

Letter (First Char)	Element	Example Line
B	IBIS buffer	`b_io_0 nd_pu0 nd_pd0 nd_out nd_in0 nd_en0 nd_outofin0 nd_pc0 nd_gc0`
C	Capacitor	`Cbypass 1 0 10pf`
D	Diode	`D7 3 9 D1`
E	Voltage-controlled voltage source	`Ea 1 2 3 4 K`
F	Current-controlled current source	`Fsub n1 n2 vin 2.0`
G	Voltage-controlled current source	`G12 4 0 3 0 10`
H	Current-controlled voltage source	`H3 4 5 Vout 2.0`
I	Current source	`I A 2 6 1e-6`
J	JFET or MESFET	`J1 7 2 3 GAASFET`
K	Linear mutual inductor (general form)	`K1 L1 L2 1`
L	Linear inductor	`LX a b 1e-9`
M	MOS transistor	`M834 1 2 3 4 N1`
P	Port	`P1 in gnd port=1 z0=50`
Q	Bipolar transistor	`Q5 3 6 7 8 pnp1`
R	Resistor	`R10 21 10 1000`
S, T, U, W	Transmission line	`S1 nd1 nd2 s_model2`
V	Voltage source	`V1 8 0 5`

Table 4 Element Identifiers (Continued)

Letter (First Char)	Element	Example Line
W	Transmission Line	`W1 in1 0 out1 0 N=1 L=1`
X	Subcircuit call	`X1 2 4 17 31 MULTI WN=100 LN=5`

Hierarchy Paths

- A period (.) indicates path hierarchy.
- Paths can be up to 1024 characters long.
- Path numbers compress the hierarchy for post-processing and listing files.
- You can find path number cross references in the listing and in the *<design>*.pa0 file.
- The `.OPTION PATHNUM` controls whether the list files show full path names or path numbers.

Numbers

You can enter numbers as integer, floating point, floating point with an integer exponent, or integer or floating point with one of the scale factors listed in Table 5.

Table 5 Scale Factors

Scale Factor	Prefix	Symbol	Multiplying Factor
T	tera	T	1e+12
G	giga	G	1e+9
MEG or X	mega	M	1e+6
K	kilo	k	1e+3
M	milli	m	1e-3

Table 5 Scale Factors (Continued)

Scale Factor	Prefix	Symbol	Multiplying Factor
U	micro	μ	1e-6
N	nano	n	1e-9
P	pico	p	1e-12
F	femto	f	1e-15
A	atto	a	1e-18

Note:

Scale factor A is not a scale factor in a character string that contains amps. For example, HSPICE interprets the 20amps string as 20e-18mps (20^{-18}mps), but it correctly interprets 20amps as 20 amperes of current, not as 20e-18mps (20^{-18}mps).

- Numbers can use exponential format or engineering key letter format, but not both (1e-12 or 1p, but not 1e-6u).
- To designate exponents, use D or E.
- The `.OPTION EXPMAX` limits the exponent size.
- Trailing alphabetic characters are interpreted as units comments.
- Units comments are not checked.
- The `.OPTION INGOLD` controls the format of numbers in printouts.
- The `.OPTION NUMDGT=x` controls the listing printout accuracy.
- The `.OPTION MEASDGT=x` controls the measure file printout accuracy.
- The `.OPTION VFLOOR=x` specifies the smallest voltage for which HSPICE or HSPICE RF prints the value. Smaller voltages print as 0.

Parameters and Expressions

- Parameter names in HSPICE or HSPICE RF use HSPICE name syntax rules, except that names must begin with an alphabetic character. The other characters must be either a number, or one of these characters:

  ```
  ! # $ % [ ] _
  ```

- To define parameter hierarchy overrides and defaults, use the `.OPTION PARHIER=global | local` statement.
- If you create multiple definitions for the same parameter or option, HSPICE or HSPICE RF uses the last parameter definition or `.OPTION` statement, even if that definition occurs later in the input than a reference to the parameter or option. HSPICE or HSPICE RF does not warn you when you redefine a parameter.
- You must define a parameter before you use that parameter to define another parameter.
- When you select design parameter names, be careful to avoid conflicts with parameterized libraries.
- To delimit expressions, use single or double quotes.
- Expressions cannot exceed 1024 characters.
- For improved readability, use a double slash (`\\`) at end of a line, to continue the line.
- You can nest functions up to three levels.
- Any function that you define can contain up to two arguments.
- Use the `PAR` (expression or parameter) function to evaluate expressions in output statements.

Input Netlist File Structure

An input netlist file should consist of one main program, and one or more optional submodules. Use a submodule (preceded by an `.ALTER` statement) to automatically change an input netlist file; then rerun the simulation with different options, netlist, analysis statements, and test vectors.

You can use several high-level call statements (`.INCLUDE`, `.LIB` and `.DEL LIB`) to restructure the input netlist file modules. These statements can call netlists, model parameters, test vectors, analysis, and option macros into a file,

from library files or other files. The input netlist file also can call an external data file, which contains parameterized data for element sources and models. You must enclose the names of included or internally-specified files in single or double quotation when they begin with a number (0-9).

Schematic Netlists

HSPICE or HSPICE RF typically use netlisters to generate circuits from schematics, and accept either hierarchical or flat netlists.

The process of creating a schematic involves:

- Symbol creation with a symbol editor.
- Circuit encapsulation.
- Property creation.
- Symbol placement.
- Symbol property definition.
- Wire routing and definition

Table 6 Input Netlist File Sections

Sections	Examples	Definition
Title	.TITLE	The first line in the netlist is the title of the input netlist file.
Set-up	.OPTION .IC or .NODESET, .PARAM, .GLOBAL	Sets conditions for simulation. Initial values in circuit and subcircuit. Set parameter values in the netlist. Set node name globally in netlist.
Sources	Sources and digital inputs	Sets input stimuli (I or V element).
Netlist	Circuit elements .SUBKCT, .ENDS, or .MACRO, .EOM	Circuit for simulation. Subcircuit definitions.
Analysis	.DC, .TRAN, .AC, and so on. .SAVE and .LOAD .DATA, .TEMP	Statements to perform analyses. Save and load operating point information. Create table for data-driven analysis. Set temperature analysis.

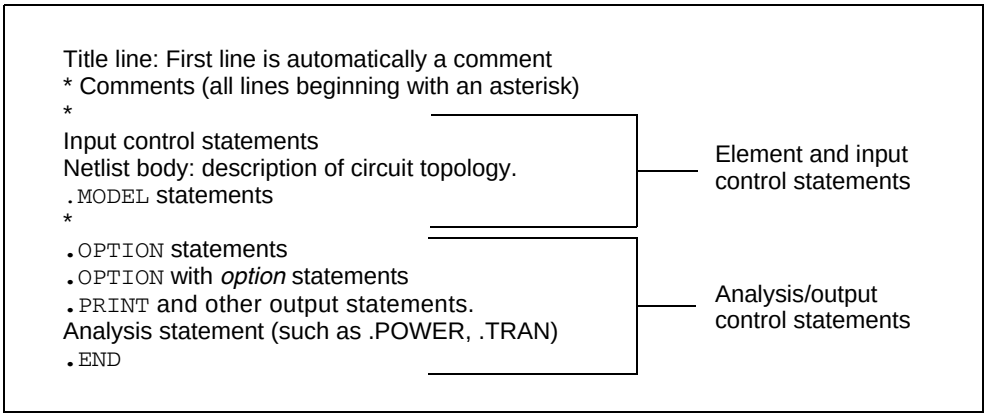
Table 6 Input Netlist File Sections (Continued)

Sections	Examples	Definition
Output	.PRINT, .PLOT, .GRAPH, .PROBE, .MEASURE	Statements to output variables.
		Statement to evaluate and report user-defined functions of a circuit.
Library, Model and File Inclusion	.INCLUDE	General include files.
	.MALIAS	Assigns an alias to a diode, BJT, JFET, or MOSFET.
	.MODEL	Element model descriptions.
	.LIB	Library.
	.OPTION SEARCH	Search path for libraries and included files.
	.PROTECT and .UNPROTECT	Control printback to output listing.
Alter blocks	.ALIAS, .ALTER, .DEL LIB	Renames a previous model. Sequence for in-line case analysis. Removes previous library selection.
End of netlist	.END	Required statement; end of netlist.

Input Netlist File Composition

The HSPICE and HSPICE RF circuit description syntax is compatible with the SPICE input netlist format. Figure 7 shows the basic structure of an input netlist.

Figure 7 Basic Netlist Structure



The following is an example of a simple netlist file, called inv_ckt.in. It shows a small inverter test case that measures the timing behavior of the inverter.

To create the circuit:

1. Define the MOSFET models for the PMOS and NMOS transistors of the inverter.
2. Insert the power supplies for both VDD and GND power rails.

Insert the pulse source to the inverter input.

This circuit uses transient analysis and produces output graphical waveform data for the input and output ports of the inverter circuit.

```
* Sample inverter circuit
* **** MOS models *****
.MODEL n1 NMOS LEVEL=3 THETA=0.4 ...
.MODEL p1 PMOS LEVEL=3 ...
* ***** Define power supplies and sources *****
VDD VDD 0 5
VPULSE VIN 0 PULSE 0 5 2N 2N 2N 98N 200N
VGND GND 0 0
* ***** Actual circuit topology *****
M1 VOUT VIN VDD VDD p1
M2 VOUT VIN GND GND n1
* ***** Analysis statement *****
```

```
.TRAN 1n 300n
* ***** Output control statements *****
.OPTION POST PROBE
.PROBE V(VIN) V(VOUT)
.END
```

For a description of individual commands used in HSPICE or HSPICE RF netlists, see the "Netlist Commands" chapter in the *HSPICE Command Reference*.

Title of Simulation

You set the simulation title in the first line of the input file. HSPICE or HSPICE RF always reads this line, and uses it as the title of the simulation, regardless of the line's contents. The simulation prints the title verbatim, in each section heading of the output listing file.

To set the title, you can place a `.TITLE` statement on the first line of the netlist. However, HSPICE or HSPICE RF does not require the `.TITLE` syntax.

The first line of the input file is always the implicit title. If any statement appears as the first line in a file, simulation interprets it as a title, and does not execute it.

An `.ALTER` statement does not support use the `.TITLE` statement. To change a title for a `.ALTER` statement, place the title content in the `.ALTER` statement itself.

Comments and Line Continuation

The first line of a netlist is always a comment, regardless of its first character; comments that are not the first line of the netlist require either an asterisk (`*`) in HSPICE or a number sign (`#`) in HSPICE RF as the first character of the line, or a dollar sign (`$`) directly in front of the comment anywhere on the line. For example,

```
* <comment_on_a_line_by_itself>
-or-
<HSPICE_statement> $ <comment_following_HSPICE_input>
```

You can place comment statements anywhere in the circuit description.

The dollar sign must be used for comments that do *not* begin in the first character position on a line (for example, for comments that follow simulator input on the same line). If it is not the first nonblank character, then the dollar sign must be preceded by either:

- Whitespace
- Comma (,)
- Valid numeric expression.

You can also place the dollar sign within node or element names. For example,

```
* RF=1K GAIN SHOULD BE 100
$ MAY THE FORCE BE WITH MY CIRCUIT
VIN 1 0 PL 0 0 5V 5NS $ 10v 50ns
R12 1 0 1MEG $ FEED BACK
.PARAM a=1w$comment a=1, w treated as a space and ignored
.PARAM a=1k$comment a=1e3, k is a scale factor
```

A dollar sign is the preferred way to indicate comments, because of the flexibility of its placement within the code.

Line continuations require a plus sign (+) as the first character in the line that follows. Here is an example of comments and line continuation in a netlist file:

```
.ABC Title Line (HSPICE or HSPICE RF ignores the netlist keyword
* on this line, because the first line is always a comment)

* This is a comment line
.MODEL n1 NMOS $ this is an example of an inline comment
* This is a comment line and the following line is a continuation
+ LEVEL=3
```

Element and Source Statements

Element statements describe the netlists of devices and sources. Use nodes to connect elements to one another. Nodes can be either numbers or names. Element statements specify:

- Type of device.
- Nodes to which the device is connected.
- Operating electrical characteristics of the device.

Element statements can also reference model statements that define the electrical parameters of the element.

Table 7 lists the parameters of an element statements.

Table 7 Element Parameters

Parameter	Description
elname	Element name that cannot exceed 1023 characters, and must begin with a specific letter for each element type: B IBIS buffer C Capacitor D Diode E,F,G,H Dependent current and voltage sources I Current (inductance) source J JFET or MESFET K Mutual inductor L Inductor model or magnetic core mutual inductor model M MOSFET Q BJT P Port R Resistor S, T, U, WTransmission line V Voltage source X Subcircuit call
node1 ...	Node names identify the nodes that connect to the element. The node name begins with a letter and can contain a maximum of 1023 characters. You cannot use the following characters in node names:=( ),' <space>
mname	HSPICE or HSPICE RF requires a model reference name for all elements, except passive devices.
pname1 ...	An element parameter name identifies the parameter value that follows this name.
expression	Any mathematical expression containing values or parameters, such as param1 * val2
val1 ...	Value of the *pname1* parameter, or of the corresponding model node. The value can be a number or an algebraic expression.
M=val	Element multiplier. Replicates *val* element times, in parallel. Do not assign a negative value or zero as the M value.

For descriptions of element statements for the various types of supported elements, see the chapters about individual types of elements in this user guide.

Example 1

```
Q1234567  4000  5000 6000 SUBSTRATE BJTMODEL AREA=1.0
```

The preceding example specifies a bipolar junction transistor, with its collector connected to node 4000, its base connected to node 5000, its emitter connected to node 6000, and its substrate connected to the `SUBSTRATE` node. The `BJTMODEL` name references the model statement, which describes the transistor parameters.

```
M1 ADDR SIG1 GND SBS N1 10U 100U
```

The preceding example specifies a MOSFET named `M1`, where:

- drain node=`ADDR`
- gate node=`SIG1`
- source node=`GND`
- substrate nodes=`SBS`

The preceding element statement calls an associated model statement, `N1`. The MOSFET dimensions are width=`100` microns and length=`10` microns.

Example 2

```
M1 ADDR SIG1 GND SBS N1 w1+w l1+l
```

The preceding example specifies a MOSFET named `M1`, where:

- drain node=`ADDR`
- gate node=`SIG1`
- source node=`GND`
- substrate nodes=`SBS`

The preceding element statement calls an associated model statement, `N1`. MOSFET dimensions are algebraic expressions (width=`w1+w`, and length=`l1+l`).

Defining Subcircuits

You can create a subcircuit description for a commonly-used circuit, and include one or more references to the subcircuit in your netlist.

- Use `.SUBCKT` and `.MACRO` statements to define subcircuits within your HSPICE netlist or HSPICE RF.
- Use the `.ENDS` statement to terminate a `.SUBCKT` statement.
- Use the `.EOM` statement to terminate a `.MACRO` statement.
- Use `X<subcircuit_name>` (the subcircuit call statement) to call a subcircuit that you previously defined in a `.MACRO` or `.SUBCKT` command in your netlist, where <*subcircuit_name*> is the element name of the subcircuit that you are calling. This subcircuit element name can be up to 15 characters long.
- Use the `.INCLUDE` statement to include another netlist as a subcircuit in the current netlist.

Node Naming Conventions

Nodes are the points of connection between elements in the input netlist. You can use either names or numbers to designate nodes. Node numbers can be from 1 to 999999999999999; node number 0 is always ground. HSPICE or HSPICE RF ignores letters that follow numbers in node names. When the node name begins with a letter or a valid special character, the node name can contain a maximum of 1024 characters.

In addition to letters and digits, node names can include the following characters:

`+, -, *, /, $, #, [], !, <>, _, %`

Node names that begin with one or more numerical digits cannot contain brackets; for example, 123[r55]. Whereas, node names that begin with alphabetic character may contain brackets; for example, n123[r55].

If you use braces { } in node names, HSPICE or HSPICE RF changes them to brackets [ ].

You cannot use the following characters in node names: `() ,=' <blank>`

You should avoid using the dollar sign ($) after a numerical digit in a node name, because HSPICE assumes whatever follows the "$" symbol is an in-line comment (see Comments and Line Continuation on page 40 for additional

information). It can cause error and warning messages depending on where the node containing the "$" is located. For example, HSPICE generates an error indicating that a resistor node is missing:

```
R1 1$ 2 1k
```

Also, in this example, HSPICE issues a warning indicating that the value of resistor R1 is limited to 1e-5 and interprets the line as "`R1 2 1`" without a defined value:

```
R1 2 1$ 1k
```

The period (.) is reserved for use as a separator between a subcircuit name and a node name: *subcircuitName.nodeName*. If a node name contains a period, the node will be considered a top level node unless there is a valid match to a subcircuit name and node name in the hierarchy.

The sorting order for operating point nodes is:

```
a-z, !, #, $, %, *, +, -, /
```

Using Wildcards on Node Names

You can use wildcards to match node names.

- `?` wildcard matches any single character. For example, `9?` matches `92`, `9a`, `9A`, and `9%`.
- `*` wildcard matches any string of zero or more characters. For example:
  - If your netlist includes a resistor named `r1` and a voltage source named `vin`, then `.PRINT i(*)` prints the current for both of these elements: `i(r1)` and i(vin).
  - And `.PRINT v(o*)` prints the voltages for all nodes whose names start with `o`; if your netlist contains nodes named `in` and `out`, this example prints only the `v(out)` voltage.
- `[ ]` matches any character tht appears within the brackets. For example, `[123]` matches `1`, `2`, or `3`. A hyphen inside the brackets indicates a character range. For example, `[0-9]` is the same as `[0123456789]`, and matches any digit.

For example, the following prints the results of a transient analysis for the voltage at the matched node name.

```
.PRINT TRAN V(9?t*u)
```

Wildcards must begin with a letter or a number; for example,

```
.PROBE v(*)                $ correct format
.PROBE *                   $ incorrect format
.PROBE x*                  $ correct format
```

Here are some practical applications for these wildcards:

- If your netlist includes a resistor named `r1` and a voltage source named `vin`, then `.PRINT i(*)` prints the current for both elements `i(r1)` and `i(vin)`.
- The statement `.PRINT v(o*)` prints the voltages for all nodes whose names start with o; if your netlist contains nodes named `in` and `out`, this example prints only the `v(out)` voltage.
- If your netlist contains nodes named `0`, `1`, 2, and `3`, then `.PRINT v(0,*)` or `.PRINT v(0 *)` prints the voltage between node `0` and each of the other nodes: `v(0,1)`, `v(0,2)`, and `v(0,3)`.

Examples

The following examples use wildcards with `.PRINT`, `.PROBE`, and `.LPRINT` statements.

- Probe node voltages for nodes at all levels.

  ```
  .PROBE v(*)
  ```

- Probe all nodes whose names start with “a”. For example: `a1`, `a2`, `a3`, `a00`, `ayz`.

  ```
  .PROBE v(a*)
  ```

- Print node voltages for nodes at the first level and all levels below the first level, where zero-level are top-level nodes. For example: `X1.A`, `X4.554`, `Xab.abc123`.

  ```
  .PRINT v(*.*)
  ```

- Probe node voltages for all nodes whose name start with “x” at the first level and all levels below the first level, where zero-level are top-level nodes. For example: `x1.A`, `x4.554`, `xab.abc123`.

  ```
  .PROBE v(x*.*)
  ```

- Print node voltages for nodes whose names start with “x” at the second-level and all levels below the second level. For example: `x1.x2.a`, `xab.xdff.in`.

  ```
  .PRINT v(x*.*.*)
  ```

- Match all first-level nodes with names that are exactly two characters long. For example: `x1.in`, `x4.12`.

  ```
  .PRINT v(x*.*.*)
  ```

- In HSPICE RF, print the logic state of all top-level nodes, whose names start with `b`. For example: `b1`, `b2`, `b3`, `b56`, `bac`.

  ```
  .LPRINT (1,4) b*
  ```

Element, Instance, and Subcircuit Naming Conventions

Instances and subcircuits are elements and as such, follow the naming conventions for elements.

Element names in HSPICE or HSPICE RF begin with a letter designating the element type, followed by up to 1023 alphanumeric characters. Element type letters are R for resistor, C for capacitor, M for a MOSFET device, and so on (see Element and Source Statements on page 41).

Subcircuit Node Names

HSPICE assigns two subcircuit node names.

- To assign the first name, HSPICE or HSPICE RF uses the (`.`) extension to concatenate the circuit path name with the node name—for example, `X1.XBIAS.M5`.

  Node designations that start with the same number, followed by any letter, are the same node. For example, `1c` and `1d` are the same node.

- The second subcircuit node name is a unique number that HSPICE automatically assigns to an input netlist subcircuit. The (`:`) extension concatenates this number with the internal node name, to form the entire subcircuit's node name (for example, `10:M5`). The output listing file cross-references the node name.

  Note:

  > HSPICE RF does not support short names for internal subcircuits, such as `10:M5`.

To indicate the ground node, use either the number `0`, the name `GND`, or `!GND`. Every node should have at least two connections, except for transmission line nodes (unterminated transmission lines are permitted) and MOSFET substrate

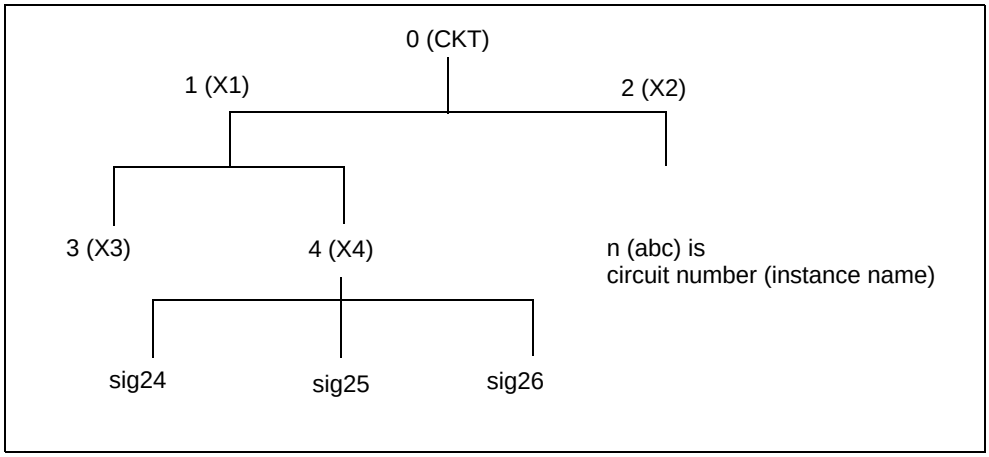
nodes (which have two internal connections). Floating power supply nodes are terminated with a 1Megohm resistor and a warning message.

Path Names of Subcircuit Nodes

A path name consists of a sequence of subcircuit names, starting at the highest-level subcircuit call, and ending at an element or bottom-level node. Periods separate the subcircuit names in the path name. The maximum length of the path name, including the node name, is 1024 characters.

You can use path names in `.PRINT`, `.PLOT`, `.NODESET`, and `.IC` statements, as another way to reference internal nodes (nodes not appearing on the parameter list). You can use the path name to reference any node, including any internal node. Subcircuit node and element names follow the rules shown in Figure 8 on page 48.

Figure 8 Subcircuit Calling Tree, with Circuit Numbers and Instance Names



In Figure 8, the path name of the `sig25` node in the `X4` subcircuit is `X1.X4.sig25`. You can use this path in HSPICE or HSPICE RF statements, such as:

```
.PRINT v(X1.X4.sig25)
```

Abbreviated Subcircuit Node Names

In HSPICE, you can use circuit numbers as an alternative to path names, to reference nodes or elements in `.PRINT`, `.PLOT`, `.NODESET`, or `.IC`

statements. Compiling the circuit assigns a circuit number to all subcircuits, creating an abbreviated path name:

<*subckt-num*>:<*name*>

Note:

HSPICE RF does not recognize this type of abbreviated subcircuit name.

The subcircuit name and a colon precede every occurrence of a node or element in the output listing file. For example, `4:INTNODE1` is a node named `INTNODE1`, in a subcircuit assigned the number `4`.

Any node not in a subcircuit has a 0: prefix (0 references the main circuit). To identify nodes and subcircuits in the output listing file, HSPICE uses a circuit number that references the subcircuit where the node or element appears.

Abbreviated path names let you use DC operating point node voltage output, as input in a `.NODESET` statement for a later run.

You can copy the part of the output listing titled *Operating Point Information* or you can type it directly into the input file, preceded by a `.NODESET` statement. This eliminates recomputing the DC operating point in the second simulation.

Automatic Node Name Generation

HSPICE or HSPICE RF can automatically assign internal node names. To check both nodal voltages and branch currents, you can use the assigned node name when you print or plot. HSPICE or HSPICE RF supports several special cases for node assignment—for example, simulation automatically assigns node 0 as a ground node.

For CSOS (CMOS Silicon on Sapphire), if you assign a value of -1 to the bulk node, the name of the bulk node is B#. Use this name to print the voltage at the bulk node. When printing or plotting current—for example `.PLOT I(R1)`—HSPICE inserts a zero-valued voltage source. This source inserts an extra node in the circuit named V*nn*, where *nn* is a number that HSPICE (or HSPICE RF) automatically generates; this number appears in the output listing file.

Global Node Names

The `.GLOBAL` statement globally assigns a node name, in HSPICE or HSPICE RF. This means that all references to a global node name, used at any level of the hierarchy in the circuit, connect to the same node.

The most common use of a .GLOBAL statement is if your netlist file includes subcircuits. This statement assigns a common node name to subcircuit nodes. Another common use of .GLOBAL statements is to assign power supply connections of all subcircuits. For example, .GLOBAL VCC connects all subcircuits with the internal node name VCC.

Ordinarily, in a subcircuit, the node name consists of the circuit number, concatenated to the node name. When you use a .GLOBAL statement, HSPICE or HSPICE RF does not concatenate the node name with the circuit number, and assigns only the global name. You can then exclude the power node name in the subcircuit or macro call.

Circuit Temperature

To specify the circuit temperature for a HSPICE or HSPICE RF simulation, use the .TEMP statement, or the TEMP parameter in the .DC, .AC, and .TRAN statements. HSPICE compares the circuit simulation temperature against the reference temperature in the TNOM control option. HSPICE or HSPICE RF uses the difference between the circuit simulation temperature and the TNOM reference temperature to define derating factors for component values.

In HSPICE RF, you can use multiple .TEMP statements to specify multiple temperatures for different portions of the circuit. HSPICE permits only one temperature for the entire circuit. Multiple .TEMP statements in a circuit behave as a sweep function.

Data-Driven Analysis

In data-driven analysis, you can modify any number of parameters, then use the new parameter values to perform an operating point, DC, AC, or transient analysis. An array of parameter values can be either inline (in the simulation input file) or stored as an external ASCII file. The .DATA statement associates a list of parameter names with corresponding values in the array.

HSPICE supports the entire functionality of the .DATA statement. However, HSPICE RF supports .DATA only for*:*

- Data-driven analysis.
- Inline or external data files.

For more details about using the .DATA statement in different types of analysis, see Chapter 8, Initializing DC/Operating Point Analysis and Chapter 9, Transient Analysis.

Library Calls and Definitions

To create and read from libraries of commonly-used commands, device models, subcircuit analysis, and statements in library files, use the .LIB call statement. As HSPICE or HSPICE RF encounters each .LIB call name in the main data file, it reads the corresponding entry from the designated library file, until it finds an .ENDL statement.

You can also place a .LIB call statement in an .ALTER block.

Library Building Rules

- A library cannot contain .ALTER statements.
- A library can contain nested .LIB calls to itself or to other libraries. If you use a relative path in a nested .LIB call, the path starts from the directory of the parent library, not from the work directory. If the path starts from the work directory, HSPICE can also find the library, but it prints a warning. The depth of nested calls is limited only by the constraints of your system configuration.
- A library cannot contain a call to a library of its own entry name, within the same library file.
- A HSPICE or HSPICE RF library cannot contain the .END statement.
- .ALTER processing cannot change .LIB statements, within a file that an .INCLUDE statement calls.

Automatic Library Selection

Automatic library selection searches a sequence of up to 40 directories. The hspice.ini file sets the default search paths.

Note:

HSPICE RF does not read the hspice.ini file.

Use this file for directories that you want HSPICE to always search. HSPICE searches for libraries in the order specified in .OPTION SEARCH statements.

When HSPICE encounters a subcircuit call, the search order is:

1. Read the input file for a .SUBCKT or .MACRO with the specified call name.
2. Read any .INC files or .LIB files for a .SUBCKT or .MACRO with the specified call name.

3. Search the directory containing the input file for the call_name.inc file.
4. Search the directories in the `.OPTION SEARCH` list.

You can use the HSPICE library search and selection features to simulate process corner cases, using `.OPTION SEARCH =‘<libdir>’` to target different process directories. For example, if you store an input or output buffer subcircuit in a file named iobuf.inc, you can create three copies of the file, to simulate fast, slow and typical corner cases. Each file contains different HSPICE transistor models, representing the different process corners. Store these files (all named iobuf.inc) in separate directories.

Defining Parameters

The `.PARAM` statement defines parameters. Parameters in HSPICE or HSPICE RF are names that have associated numeric values. You can also use either of the following specialized methods to define parameters:

- Predefined Analysis
- Measurement Parameters

Predefined Analysis

HSPICE or HSPICE RF provides several specialized analysis types, which require a way to control the analysis. For the syntax used in these `.PARAM` commands, see the description of the .PARAM command in the *HSPICE Command Reference*.

HSPICE or HSPICE RF supports the following predefined analysis parameters:

- Temperature functions (*fn*)
- Optimization guess/range

HSPICE also supports the following predefined parameter types, that HSPICE RF does not support:

- frequency
- time
- Monte Carlo functions

Measurement Parameters

A .MEASURE statement produces a measurement parameter. In general, the rules for measurement parameters are the same as those for standard parameters. However, measurement parameters are not defined in a .PARAM statement, but directly in the .MEASURE statement. For more information, see .MEASURE Parameter Types on page 269.

Altering Design Variables and Subcircuits

The following rules apply when you use an .ALTER block to alter design variables and subcircuits in HSPICE. This section does not apply to HSPICE RF.

- If the name of a new element, .MODEL statement, or subcircuit definition is identical to the name of an original statement of the same type, then the new statement replaces the old. Add new statements in the input netlist file.
- You can alter element and .MODEL statements within a subcircuit definition. You can also add a new element or .MODEL statement to a subcircuit definition. To modify the topology in subcircuit definitions, put the element into libraries. To add a library, use .LIB; to delete, use .DEL LIB.
- If a parameter name in a new .PARAM statement in the .ALTER module is identical to a previous parameter name, then the new assigned value replaces the old value.
- If you used parameter (variable) values for elements (or model parameter values) when you used .ALTER, use the .PARAM statement to change these parameter values. Do not use numerical values to redescribe elements or model parameters.
- If you used an .OPTION statement (in an original input file or a .ALTER block) to turn on an option, you can turn that option off.
- Each .ALTER simulation run prints only the actual altered input. A special .ALTER title identifies the run.
- .ALTER processing cannot revise .LIB statements within a file that an .INCLUDE statement calls. However, .ALTER processing can accept .INCLUDE statements, within a file that a .LIB statement calls.

Using Multiple .ALTER Blocks

This section does not apply to HSPICE RF.

- For the first simulation run, HSPICE reads the input file, up to the first .ALTER statement, and performs the analyses up to that .ALTER statement.
- After it completes the first simulation, HSPICE reads the input between the first .ALTER statement, and either the next .ALTER statement or the .END statement.
- HSPICE then uses these statements to modify the input netlist file.
- HSPICE then resimulates the circuit.
- For each additional .ALTER statement, HSPICE performs the simulation that precedes the first .ALTER statement.
- HSPICE then performs another simulation, using the input between the current .ALTER statement, and either the next .ALTER statement or the .END statement.

If you do not want to rerun the simulation that precedes the first .ALTER statement, every time you run an .ALTER simulation, then do the following:

1. Put the statements that precede the first .ALTER statement, into a library.
2. Use the .LIB statement in the main input file.
3. Put a .DEL LIB statement in the .ALTER section, to delete that library for the .ALTER simulation run.

Connecting Nodes

Use a .CONNECT statement to connect two nodes in your HSPICE netlist, so that simulation evaluates two nodes as only one node. Both nodes must be at the same level in the circuit design that you are simulating: you cannot connect nodes that belong to different subcircuits. You also cannot use this statement in HSPICE RF.

Deleting a Library

Use a .DEL LIB statement to remove library data from memory. The next time you run a simulation, the .DEL LIB statement removes the .LIB call statement, with the same library number and entry name, from memory. You can then use a .LIB statement to replace the deleted library.

You can use a .DEL LIB statement with a .ALTER statement. HSPICE RF does not support the .ALTER statement.

Ending a Netlist

An .END statement must be the last statement in the input netlist file. Text that follows the .END statement is a comment, and has no effect on the simulation.

An input file that contains more than one simulation run must include an .END statement for each simulation run. You can concatenate several simulations into a single file.

Condition-Controlled Netlists (IF-ELSE)

You can use the IF-ELSE structure to change the circuit topology, expand the circuit, set parameter values for each device instance, select different model cards, reference subcircuits, or define subcircuits in each 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
```

- 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)).
- In an IF, ELSEIF, or ELSE statement block, you can include most valid HSPICE or HSPICE RF analysis and output statements. The exceptions are:
  - .END, .ALTER, .GLOBAL, .DEL LIB, .MALIAS, .ALIAS, .LIST, .NOLIST, and .CONNECT statements.
  - search, d_ibis, d_imic, d_lv56, biasfi, modsrh, cmiflag, nxx, and brief options.
- You can include IF-ELSEIF-ELSE statements in subcircuits and subcircuits in IF-ELSEIF-ELSE statements.
- You can use IF-ELSEIF-ELSE blocks to select different submodules to structure the netlist (using .INC, .LIB, and .VEC statements).
- 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.
- Parameters in an IF-ELSE block do not affect the parameter value within the condition expression. HSPICE or HSPICE RF updates the parameter value only after it selects the IF-ELSE block.
- You can nest IF-ELSE blocks.
- You can include .SUBCKT and .MACRO statements within an IF-ELSE block.
- You can include an unlimited number of ELSEIF statements within an IF-ELSE block.
- You cannot include sweep parameters or simulation results within an IF-ELSE block.
- You cannot use an IF-ELSE block within another statement. In the following example, HSPICE or HSPICE RF 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
```

Using Subcircuits

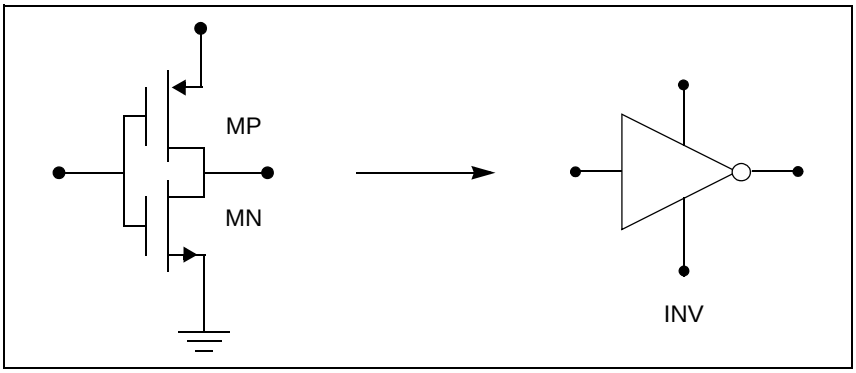
Reusable cells are the key to saving labor in any CAD system. This also applies to circuit simulation, in HSPICE or HSPICE RF.

- To create and simulate a reusable circuit, construct it as a subcircuit.
- Use parameters to expand the utility of a subcircuit.

Traditional SPICE includes the basic subcircuit, but does not provide a way to consistently name nodes. However, HSPICE or HSPICE RF provides a simple method for naming subcircuit nodes and elements: use the subcircuit call name as a prefix to the node or element name.

In HSPICE RF, you cannot replicate output commands within subcircuit (subckt) definitions.

Figure 9 Subcircuit Representation



The following input creates an instance named X1 of the INV cell macro, which consists of two MOSFETs, named MN and MP:

```
X1 IN OUT VD_LOCAL VS_LOCAL inv W=20
.MACRO INV IN OUT VDD VSS W=10 L=1 DJUNC=0
MP OUT IN VDD VDD PCH W=W L=L DTEMP=DJUNC
MN OUT IN VSS VSS NCH W='W/2' L=L DTEMP=DJUNC
.EOM
```

Note:

To access the name of the MOSFET, inside of the `INV` subcircuit that `X1` calls, the names are `X1.MP` and `X1.MN`. So to print the current that flows through the MOSFETs, use `.PRINT I (X1.MP)`.

Hierarchical Parameters

You can use two hierarchical parameters, the `M` (multiply) parameter and the `S` (scale) parameter.

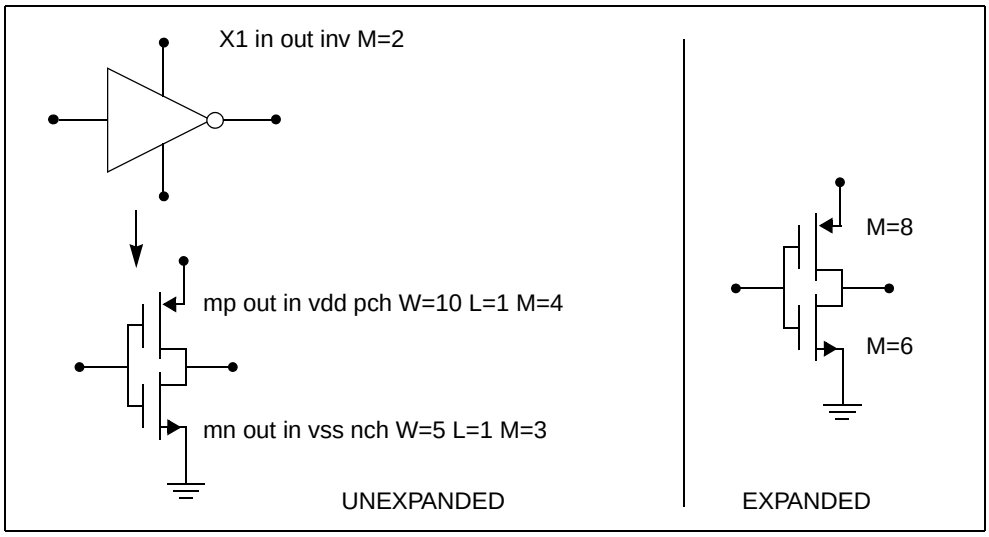
M (Multiply) Parameter

The most basic HSPICE subcircuit parameter or HSPICE RF is the `M` (multiply) parameter. This keyword is common to all elements, including subcircuits, except for voltage sources. The `M` parameter multiplies the internal component values, which in effect creates parallel copies of the element or subcircuit. To simulate 32 output buffers switching simultaneously, you need to place only one subcircuit; for example,

```
X1 in out buffer M=32
```

Multiply works hierarchically. For a subcircuit within a subcircuit, HSPICE or HSPICE RF multiplies the product of both levels. Do not assign a negative value or zero as the `M` value.

Figure 10 How Hierarchical Multiply Works



S (Scale) Parameter

To scale a subcircuit, use the S (local scale) parameter. This parameter behaves in much the same way as the M parameter in the preceding section.

```
.OPTION hier_scale=value
.OPTION scale=value
X1 node1 node2 subname S=valueM parameter
```

The .OPTION HIER_SCALE statement defines how HSPICE or HSPICE RF interprets the S parameter, where value is either:

- 0 (the default), indicating a user-defined parameter, or
- 1, indicating a scale parameter.

The .OPTION SCALE statement defines the original (default) scale of the subcircuit. The specified S scale is relative to this default scale of the subcircuit.

The scale in the subname subcircuit is value*scale. Subcircuits can originate from multiple sources, so scaling is multiplicative (cumulative) throughout your design hierarchy.

```
x1 a y inv S=1u
subckt inv in out
x2 a b kk S=1m
.ends
```

In this example:

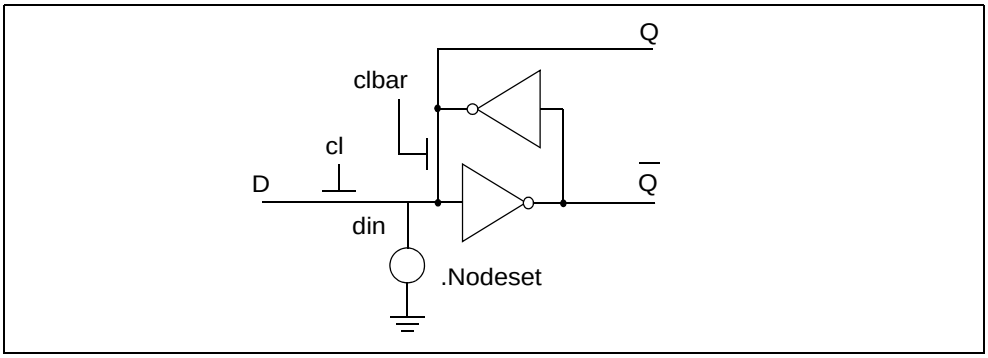
- HSPICE or HSPICE RF scales the X1 subcircuit by the first S scaling value, 1u*scale.
- Because scaling is cumulative, X2 (a subcircuit of X1) is then scaled, in effect, by the S scaling values of both X1 and X2: 1m*1u*scale.

Using Hierarchical Parameters to Simplify Simulation

You can use the hierarchical parameter to simplify simulations. An example is shown in the following listing and Figure 11 on page 60.

```
X1 D Q Qbar CL CLBAR dlatch flip=0
.macro dlatch
+ D Q Qbar CL CLBAR flip=vcc
.nodeset v(din)=flip
xinv1 din qbar inv
xinv2 Qbar Q inv
m1 q CLBAR din nch w=5 l=1
m2 D CL din nch w=5 l=1
.eom
```

Figure 11 D Latch with Nodeset



HSPICE does not limit the size or complexity of subcircuits; they can contain subcircuit references, and any model or element statement. However, in HSPICE RF, you cannot replicate output commands within subcircuit definitions. To specify subcircuit nodes in `.PRINT` or `.PLOT` statements, specify the full subcircuit path and node name.

Undefined Subcircuit Search

(HSPICE) If a subcircuit call is in a data file that does not describe the subcircuit, HSPICE automatically searches directories in the following order:

1. Present directory for the file.
2. Directories specified in `.OPTION SEARCH = "directory_path_name"` statements.
3. Directory where the Discrete Device Library is located.

HSPICE searches for the model reference name file that has an .inc suffix. For example, if the data file includes an undefined subcircuit, such as `X 1 1 2 INV`, HSPICE searches the system directories for the inv.inc file and when found, places that file in the calling data file.

Subcircuit Call Statement Discrete Device Libraries

The Synopsys Discrete Device Library (DDL) is a collection of HSPICE device models of discrete components, which you can use with HSPICE or HSPICE RF. The $<installdir>/parts directory contains the various subdirectories that form the DDL. Synopsys used its own ATEM discrete device characterization system to derive the BJT, MESFET, JFET, MOSFET, and diode models from

laboratory measurements. The behavior of op-amp, timer, comparator, SCR, and converter models closely resembles that described in manufacturers' data sheets. HSPICE and HSPICE RF have a built-in op-amp model generator.

Note:

The value of the `$INSTALLDIR` environment variable is the pathname to the directory where you installed HSPICE or HSPICE RF. This installation directory contains subdirectories, such as /parts and /bin. It also contains certain files, such as a prototype meta.cfg file, and the license files.

DDL Library Access

To include a DDL library component in a data file, use the `X` subcircuit call statement with the `DDL` element call. The `DDL` element statement includes the model name, which the actual DDL library file uses.

For example, the following element statement creates an instance of the 1N4004 diode model:

```
X1 2 1 D1N4004
```

Where `D1N4004` is the model name.

See Element and Source Statements on page 41 and the *HSPICE Elements and Device Models Manual* for descriptions of element statements.

Optional parameter fields in the element statement can override the internal specification of the model. For example, for op-amp devices, you can override the offset voltage, and the gain and offset current. Because the DDL library devices are based on HSPICE circuit-level models, simulation automatically compensates for the effects of supply voltage, loading, and temperature.

HSPICE or HSPICE RF accesses DDL models in several ways:

- The installation script creates an hspice.ini initialization file.
- HSPICE or HSPICE RF writes the search path for the DDL and vendor libraries into a `.OPTION SEARCH='<lib_path>'` statement.

  This provides immediate access to all libraries for all users. It also automatically includes the models in the input netlist. If the input netlist references a model or subcircuit, HSPICE or HSPICE RF searches the directory to which the `DDLPATH` environment variable points for a file with the same name as the reference name. This file is an include file so its filename suffix is .inc. HSPICE installation sets the `DDLPATH` variable in the meta.cfg configuration file.

- Set `.OPTION SEARCH='<lib_path>'` in the input netlist.

  Use this method to list the personal libraries to search. HSPICE or HSPICE RF first searches the default libraries referenced in the hspice.ini file, then searches libraries in the order listed in the input file.

- Directly include a specific model, using the `.INCLUDE` statement. For example, to use a model named T2N2211, store the model in a file named T2N2211.inc, and put the following statement in the input file:

  ```
  .INCLUDE <path>/T2N2211.inc
  ```

  This method requires you to store each model in its own .inc file, so it is not generally useful. However, you can use it to debug new models, when you test only a small number of models.

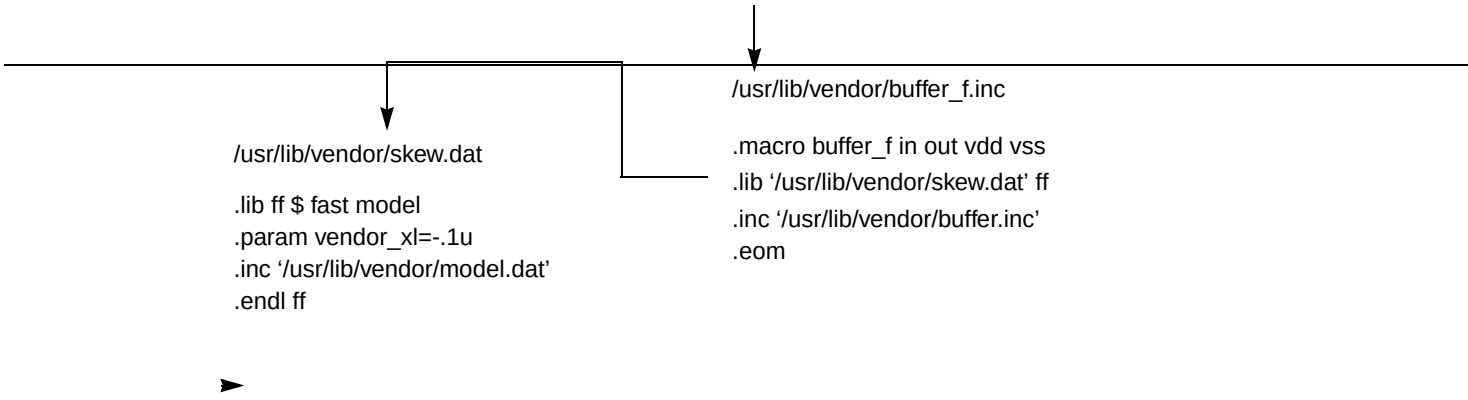
Vendor Libraries

The vendor library is the interface between commercial parts and circuit or system simulation.

- ASIC vendors provide comprehensive cells, corresponding to inverters, gates, latches, and output buffers.
- Memory and microprocessor vendors supply input and output buffers.
- Interface vendors supply complete cells for simple functions and output buffers, to use in generic family output.
- Analog vendors supply behavioral models.

To avoid name and parameter conflicts, models in vendor cell libraries should be within the subcircuit definitions.

Figure 12 Vendor Library Usage



Subcircuit Library Structure

Your library structure must adhere to the .INCLUDE statement specification in the implicit subcircuit. You can use this statement to specify the directory that contains the <subname>.inc subcircuit file, and then reference the *<subname>* in each subcircuit call.

The component naming conventions for each subcircuit is:

<*subname*>.inc

Store the subcircuit in a directory that is accessible by a .OPTION SEARCH='<lib_path>' statement.

Create subcircuit libraries in a hierarchy. Typically, the top-level subcircuit fully describes the input/output buffer; any hierarchy is buried inside. The buried hierarchy can include model statements, lower-level components, and parameter assignments. Your library cannot use .LIB or .INCLUDE statements anywhere in the hierarchy.

4

Elements

Describes the syntax for the basic elements of a circuit netlist in HSPICE or HSPICE RF.

Elements are local and sometimes customized instances of a device model specified in your design netlist.

For descriptions of the standard device models on which elements (instances) are based, see the *HSPICE Elements and Device Models Manual* and the*HSPICE MOSFET Models Manual*.

Passive Elements

This section describes the passive elements: resistors, capacitors, and inductors.

Values for Elements

HSPICE RF accepts equation-based resistors and capacitors. You can specify the value of a resistor or capacitor as an arbitrary equation, involving node voltages or variable parameters. Unlike HSPICE, you cannot use parameters to indirectly reference node voltages in HSPICE RF.

Resistor Elements in a HSPICE or HSPICE RF Netlist

```
Rxxx n1 n2 <mname> Rval <TC1 <TC2><TC>> <SCALE=val> <M=val>
+ <AC=val> <DTEMP=val> <L=val> <W=val> <C=val>
+ <NOISE=val>

Rxxx n1 n2 <mname> <R=>resistance <<TC1=>val>
+ <<TC2=>val> <<TC=>val> <SCALE=val> <M=val>
+ <AC=val> <DTEMP=val> <L=val> <W=val>
+ <C=val> <NOISE=val>
Rxxx n1 n2 R='equation' ...
```

Parameter	Description
Rxxx	Resistor element name. Must begin with R, followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name.
n2	Negative terminal node name.
mname	Resistor model name. Use this name in elements, to reference a resistor model.
TC	TC1 alias. The current definition overrides the previous definition.
TC1	First-order temperature coefficient for the resistor. See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for temperature-dependent relations.
TC2	Second-order temperature coefficient for the resistor.
SCALE	Element scale factor; scales resistance and capacitance by its value. Default=1.0.
R= resistance	Resistance value at room temperature. This can be: ▪ a numeric value in ohms ▪ a parameter in ohms ▪ a function of any node voltages ▪ a function of branch currents ▪ any independent variables such as `time`, `hertz`, and `temper`

Parameter	Description
M	Multiplier to simulate parallel resistors. For example, for two parallel instances of a resistor, set M=2, to multiply the number of resistors by 2. Default=1.0.
AC	Resistance for AC analysis. Default=Reff.
DTEMP	Temperature difference between the element and the circuit, in degrees Celsius. Default=0.0.
L	Resistor length in meters. Default=0.0, if you did not specify L in a resistor model.
W	Resistor width. Default=0.0, if you did not specify W in the model.
C	Capacitance connected from node n2 to bulk. Default=0.0, if you did not specify C in a resistor model.
user-defined equation	Can be a function of any node voltages, element currents, temperature, frequency, or time
NOISE	▪ NOISE=0, do not evaluate resistor noise. ▪ NOISE=1, evaluate resistor noise (default).

Resistance can be a value (in units of ohms) or an equation. Required parameters are the two nodes, and the resistance or model name. If you specify other parameters, the node and model name must precede those parameters. Other parameters can follow in any order. If you specify a resistor model (see the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual*), the resistance value is optional.

HSPICE Examples

In the following example, the R1 resistor connects from the Rnode1 node to the Rnode2 node, with a resistance of 100 ohms.

```
R1 Rnode1 Rnode2 100
```

The RC1 resistor connects from node 12 to node 17, with a resistance of 1 kilohm, and temperature coefficients of 0.001 and 0.

```
RC1 12 17 R=1k TC1=0.001 TC2=0
```

The Rterm resistor connects from the input node to ground, with a resistance determined by the square root of the analysis frequency (non-zero for AC analysis only).

```
Rterm input gnd R='sqrt(HERTZ)'
```

The Rxxx resistor, from node 98999999 to node 87654321, with a resistance of 1 ohm for DC and time-domain analyses, and 10 gigohms for AC analyses.

```
Rxxx 98999999 87654321 1 AC=1e10
```

HSPICE RF Examples

Some basic examples for HSPICE RF include:

- `R1` is a resistor whose resistance follows the voltage at node `c`.

  ```
  R1 1 0 'v(c)'
  ```

- `R2` is a resistor whose resistance is the sum of the absolute values of nodes `c` and `d`.

  ```
  R2 1 0 'abs(v(c)) + abs(v(d))'
  ```

- `R3` is a resistor whose resistance is the sum of the `rconst` parameter, and 100 times `tx1` for a total of 1100 ohms.

  ```
  .PARAM rconst=100 tx1=10
  R3 4 5 'rconst + tx1 * 100'
  ```

Linear Resistors

R*xxx node1 node2* < *modelname* > < R = > *value* < TC1=*val* >
+ < TC2=*val* > < W=*val* > < L=*val* > < M=*val* >
+ < C=*val* > < DTEMP=*val* > < SCALE=*val* >

Parameter	Description
Rxxx	Name of a resistor.
node1 and node2	Names or numbers of the connecting nodes.
modelname	Name of the resistor model.
value	Nominal resistance value, in ohms.
R	Resistance, in ohms, at room temperature.
TC1, TC2	Temperature coefficient.
W	Resistor width.

Parameter	Description
L	Resistor length.
M	Parallel multiplier.
C	Parasitic capacitance between node2 and the substrate.
DTEMP	Temperature difference between element and circuit.
SCALE	Scaling factor.

Example

```
R1 1 2 10.0
Rload 1 GND RVAL

.param rx=100
R3 2 3 RX TC1=0.001 TC2=0
RP X1.A X2.X5.B .5
.MODEL RVAL R
```

In the example above, `R1` is a simple 10Ω linear resistor and `Rload` calls a resistor model named `RVAL`, which is defined later in the netlist.

Note:

If a resistor calls a model, then you do not need to specify a constant resistance, as you do with `R1`.

- `R3` takes its value from the `RX` parameter, and uses the `TC1` and `TC2` temperature coefficients, which become 0.001 and 0, respectively.
- `RP` spans across different circuit hierarchies, and is 0.5Ω.

Behavioral Resistors in HSPICE or HSPICE RF

`R`*`xxx n1 n2`* `. . . <R=> '`*`equation`*`' . . .`

Note:

The equation can be a function of any node voltage or branch current, and any independent variables such as `time`, `hertz`, or `temper`.

Example

```
R1 A B R='V(A) + I(VDD)'
```

Frequency-Dependent Resistors

```
Rxxx n1 n2 R=equation <CONVOLUTION=[0|1|2] <FBASE=value>
+ <FMAX=value>>
```

Parameter	Description
CONVOLUTION	Indicates which method is used. ■ 0 : Acts the same as the conventional method. This is the default. ■ 1 : Applies recursive convolution, and if the rational function is not accurate enough, it switches to linear convolution. ■ 2 : Applies linear convolution.
FBASE	Specifies the lower bound of the transient analysis frequency. For CONVOLUTION=1 mode, HSPICE starts sampling at this frequency. For CONVOLUTION=2 mode, HSPICE uses this value as the base frequency point for Inverse Fourier Transformation. For recursive convolution, the default value is 0Hz, and for linear convolution, HSPICE uses the reciprocal of the transient period.
FMAX	Specifies the possible maximum frequency of interest. The default value is the frequency point where the function reaches close enough to infinity value, assuming that the monotonous function is approaching the infinity value and that it is taken at 10THz. The equation should be a function of HERTZ. If CONVOLUTION is turned on when a HERTZ keyword is not used in the equation, it is automatically be turned off to let the resistor behave as conventional.The equation can be a function of temperature, but it cannot be node voltage or branch current and time.

The equation can only be a function of time-independent variables such as hertz, and temperature.

Example

```
R1 1 2 r='1.0 + 1e-5*sqrt(HERTZ)' CONVOLUTION=1
```

Skin Effect Resistors

```
Rxxx n1 n2 R=value Rs=value
```

The `Rs` indicates the skin effect coefficient of the resistor.

The complex impedance of the resistor can be expressed as the following equation:

```
R(f)=Ro + (1+j)*Rs*sqrt(f)
```

The `Ro`, `j`, and `f` are DC resistance, imaginably unit (j^2=-1) and frequency, respectively.

Capacitors

```
Cxxx n1 n2 <mname> <C=>capacitance <<TC1=>val>
+ <<TC2=>val> <SCALE=val> <IC=val> <M=val>
+ <W=val> <L=val> <DTEMP=val>
Cxxx n1 n2 <C=>'equation' <CTYPE=0|1>
+ <above_options...>
```

Polynomial form:

```
Cxxx n1 n2 POLY c0 c1... <above_options...>
```

Parameter	Description
Cxxx	Capacitor element name. Must begin with C, followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name.
n2	Negative terminal node name.
mname	Capacitance model name. Elements use this name to reference a capacitor.
C=capacitance	Capacitance at room temperature—a numeric value or a parameter in farads.
TC1	First-order temperature coefficient for the capacitor. See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for temperature-dependent relations.
TC2	Second-order temperature coefficient for the capacitor.

Parameter	Description
SCALE	Element scale parameter, scales capacitance by its value. Default=1.0.
IC	Initial voltage across the capacitor, in volts. If you specify UIC in the .TRAN statement, HSPICE or HSPICE RF uses this value as the DC operating point voltage. The .IC statement overrides it.
M	Multiplier, used to simulate multiple parallel capacitors. Default=1.0
W	Capacitor width, in meters. Default=0.0, if you did not specify W in a capacitor model.
L	Capacitor length, in meters. Default=0.0, if you did not specify L in a capacitor model.
DTEMP	Element temperature difference from the circuit temperature, in degrees Celsius. Default=0.0.
C='equation'	Capacitance at room temperature, specified as a function of: ▪ any node voltages ▪ any branch currents ▪ any independent variables such as `time`, `hertz`, and `temper`
CTYPE	Determines capacitance charge calculation for elements with capacitance equations. If the C capacitance is a function of V(n1<,n2>), set CTYPE=0. Use this setting correctly, to ensure proper capacitance calculations, and correct simulation results. Default=0.
POLY	Keyword, to specify capacitance as a non-linear polynomial.
c0 c1...	Coefficients of a polynomial, described as a function of the voltage across the capacitor. c0 represents the magnitude of the 0th order term, `c1` represents the magnitude of the 1st order term, and so on. You cannot use parameters as coefficient values.

You can specify capacitance as a numeric value, in units of farads, as an equation, or as a polynomial of the voltage. The only required fields are the two nodes, and the capacitance or model name.

- If you use the parameter labels, the nodes and model name must precede the labels. Other arguments can follow in any order.
- If you specify a capacitor model (see the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual*), the capacitance value is optional.

If you use an equation to specify capacitance, the CTYPE parameter determines how HSPICE calculates the capacitance charge. The calculation is different, depending on whether the equation uses a self-referential voltage (that is, the voltage across the capacitor, whose capacitance is determined by the equation).

To avoid syntax conflicts, if a capacitor model has the same name as a capacitance parameter, HSPICE or HSPICE RF uses the model name.

Example 1

In the following example, C1 assumes its capacitance value from the model, not the parameter.

```
.PARAMETER CAPXX=1
C1 1 2 CAPXX
.MODEL CAPXX C CAP=1
```

Example 2

In the following example, the C1 capacitors connect from node 1 to node 2, with a capacitance of 20 picofarads:

```
C1 1 2 20p
```

In this next example, Cshunt refers to three capacitors in parallel, connected from the node output to ground, each with a capacitance of 100 femtofarads.

```
Cshunt output gnd C=100f M=3
```

The Cload capacitor connects from the driver node to the output node. The capacitance is determined by the voltage on the capcontrol node, times 1E-6. The initial voltage across the capacitor is 0 volts.

```
Cload driver output C='1u*v(capcontrol)' CTYPE=1 IC=0v
```

The C99 capacitor connects from the in node to the out node. The capacitance is determined by the polynomial C=c0 + c1*v + c2*v*v, where v is the voltage across the capacitor.

```
C99 in out POLY 2.0 0.5 0.01
```

Linear Capacitors

```
Cxxx node1 node2 < modelname > < C=> value < TC1=val >
+ < TC2=val > <W=val > < L=val > < DTEMP=val >
+ < M=val > < SCALE=val > < IC=val >
```

Parameter	Description
Cxxx	Name of a capacitor. Must begin with C, followed by up to 1023 alphanumeric characters.
node1 and node2	Names or numbers of connecting nodes.
value	Nominal capacitance value, in Farads.
modelname	Name of the capacitor model.
C	Capacitance, in Farads, at room temperature.
TC1, TC2	Specifies the temperature coefficient.
W	Capacitor width.
L	Capacitor length.
M	Multiplier to simulate multiple parallel capacitors.
DTEMP	Temperature difference between element and circuit.
SCALE	Scaling factor.
IC	Initial capacitor voltage.

Example

```
Cbypass 1 0 10PF
C1 2 3 CBX
.MODEL CBX C
CB B 0 10P IC=4V
CP X1.XA.1 0 0.1P
```

In this example:

- Cbypass is a straightforward, 10-picofarad (PF) capacitor.
- C1, which calls the CBX model, does not have a constant capacitance.

- CB is a 10 PF capacitor, with an initial voltage of 4V across it.
- CP is a 0.1 PF capacitor.

Frequency-Dependent Capacitors

You can specify frequency-dependent capacitors using the `C='equation'` with the `HERTZ` keyword. The `HERTZ` keyword represents the operating frequency. In time domain analyses, an expression with the `HERTZ` keyword behaves differently according to the value assigned to the `CONVOLUTION` keyword.

Syntax

```
Cxxx n1 n2 C='equation' <CONVOLUTION=[0|1|2]
+ <FBASE=val> <FMAX=val>>
```

Parameter	Description
n1 n2	Names or numbers of connecting nodes.
equation	Expressed as a function of HERTZ. If CONVOLUTION=1 or 2 and HERTZ is not used in the equation, CONVOLUTION is turned off and the capacitor behaves conventionally. The equation can be a function of temperature, but it does not support variables of node voltage, branch current, or time. If these variables exist in the expression and CONVOLUTION=1 or 2, then only their values at the operating point are considered in calculation.
CONVOLUTION	Specifies the method used. ■ 0 (default): HERTZ=0 in time domain analysis. ■ 1 or 2: performs Inverse Fast Fourier Transformation (IFFT) linear convolution.
FBASE	Base frequency to use for transient analysis. This value becomes the base frequency point for Inverse Fast Fourier Transformation (IFFT) when CONVOLUTION=1 or 2. If you do not set this value, the base frequency is a reciprocal value of the transient period.
FMAX	Maximum frequency to use for transient analysis. Used as the maximum frequency point for Inverse Fourier Transformation. If you do not set this value, the reciprocal value of RISETIME is taken.

Example

```
C1 1 2 C='1e-6 - HERTZ/1e16' CONVOLUTION=1 fbase=10
+ fmax=30meg
```

Behavioral Capacitors in HSPICE or HSPICE RF

Cxxx n1 n2 . . . C=‘*equation*’ CTYPE=0 or 1

Parameter	Description
CTYPE	Determines the calculation mode for elements that use capacitance equations. Set this parameter carefully, to ensure correct simulation results. HSPICE RF extends the definition and values of CTYPE, relative to HSPICE: ▪ CTYPE=0, if C depends only on its own terminal voltages—that is, a function of V(n1<, n2>). ▪ CTYPE=1, if C depends only on outside voltages or currents. ▪ CTYPE=2, if C depends on both its own terminal and outside voltages. This is the default for HSPICE RF. HSPICE does not support C=2.

You can specify the capacitor value as a function of any node voltage or branch current, and any independent variables such as `time`, `hertz`, and `temper`.

Example

```
C1 1 0 C=’1e-9*V(10)’ CTYPE=1
V10 10 0 PWL(0,1v t1,1v t2,4v)
```

DC Block Capacitors

```
Cxxx node1 node2 <C=> INFINITY  <IC=val>
```

When the capacitance of a capacitor is infinity, this element is called a “DC block.” In HSPICE, you specify an `INFINITY` value for such capacitors.

HPSICE does not support any other capacitor parameters for DC block elements, because HSPICE assumes that an infinite capacitor value is independent of any scaling factors.

The DC block acts as an open circuit for all DC analyses. HSPICE calculates the DC voltage across the nodes of the circuit. In all other (non-DC) analyses, a DC voltage source of this value represents the DC block—HSPICE does not allow `dv`/`dt` variations.

Charge-Conserved Capacitors

`Cxxx node1 node2 q='expression'`

HSPICE supports AC, DC, TRAN, and PZ analyses for charge-conserved capacitors.

The `expression` supports the following parameters and variables:

- Parameters
  - node voltages
  - branch currents
- Variables
  - `time`
  - `temper`
  - `hertz`

  Note:

  The `hertz` variable is not supported in transient analyses.

Parameters must be used directly in an equation. HSPICE does not support parameters that represent an equation containing variables.

Error Handling If you use an unsupported parameter in an expression, HSPICE issues an error message and aborts the simulation. HSPICE ignores unsupported analysis types and then issues warning a message.

Limitations The following syntax does not support charge-conserving capacitors:

```
Cxx node1 node2 C='expression'
```

Capacitor equations are not implicitly converted to charge equations.

Example 1: Capacitance-based Capacitor

```
C1 a b C='Co*(1+alpha*V(a,b)' ctype=0
```

You can obtain Q by integrating ‘C’ w.r.t V(a,b)

Example 2: Charge-based Capacitor

```
C1 a b Q='Co*V(a,b)(1+0.5*alpha*V(a,b))
```

Example 3: Capacitance-based Capactor

```
.option list node post
r1 1 2 100
r2 3 0 200
Vin 1 0 pulse(0 5v 1ns 2ns 2ns 10ns 20ns)
C1 2 3 c='cos(v(2,3)) + v(1,2)' ctype=2
.tran 1ns 100ns
.print tran  i(c1)
.end
```

Example 4: Charge-based Capacitor

```
.option list node post
r1 1 2 100
r2 3 0 200
Vin 1 0 pulse(0 5v 1ns 2ns 2ns 10ns 20ns)
C1 2 3 q='sin(v(2,3)) + v(2,3)*v(1,2)'
.tran 1ns 100ns
.print tran  i(c1)
.end
```

Inductors

General form:

L*xxx n1 n2* <L=>*inductance* <*mname*> <<TC1=>*val*>
+ <<TC2=>*val*> <SCALE=*val*> <IC=*val*> <M=*val*>
+ <DTEMP=*val*> <R=*val*>
L*xxx n1 n2* L='*equation*' <LTYPE=*val*> <*above_options...*>

Polynomial form:

L*xxx n1 n2 POLY c0 c1... <above_options...>*

Magnetic winding form:

L*xxx n1 n2* NT=*turns* <*above_options...*>

Parameter	Description
Lxxx	Inductor element name. Must begin with L, followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name.
n2	Negative terminal node name.

Parameter	Description
TC1	First-order temperature coefficient for the inductor. See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for temperature-dependent relations.
TC2	Second-order temperature coefficient for the inductor.
SCALE	Element scale parameter; scales inductance by its value. Default=1.0.
IC	Initial current through the inductor, in amperes. HSPICE or HSPICE RF uses this value as the DC operating point voltage, when you specify UIC in the .TRAN statement. The .IC statement overrides it.
L=inductance	Inductance value. This can be: ▪ a numeric value, in henries ▪ a parameter in henries ▪ a function of any node voltages ▪ a function of branch currents ▪ any independent variables such as `time`, `hertz`, and `temper`
M	Multiplier, used to simulate parallel inductors. Default=1.0.
DTEMP	Temperature difference between the element and the circuit, in degrees Celsius. Default=0.0.
R	Resistance of the inductor, in ohms. Default=0.0.
L='equation'	Inductance at room temperature, specified as: ▪ a function of any node voltages ▪ a function of branch currents ▪ any independent variables such as `time`, `hertz`, and `temper`
LTYPE	Calculates inductance flux for elements, using inductance equations. If the L inductance is a function of I(Lxxx), then set LTYPE=0. Otherwise, set LTYPE=1. Use this setting correctly, to ensure proper inductance calculations, and correct simulation results. Default=0.

Parameter	Description
POLY	Keyword that specifies the inductance, calculated by a polynomial.
c0 c1...	Coefficients of a polynomial in the current, describing the inductor value. c0 is the magnitude of the 0th order term, c1 is the magnitude of the 1st order term, and so on.
NT=turns	Number of turns of an inductive magnetic winding.
mname	Saturable core model name. See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for model information.

In this syntax, the inductance can be either a value (in units of henries), an equation, a polynomial of the current, or a magnetic winding. Required fields are the two nodes, and the inductance or model name.

- If you specify parameters, the nodes and model name must be first. Other parameters can be in any order.
- If you specify an inductor model (see the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual*), the inductance value is optional.

Example 1

In the following example, the L1 inductor connects from the coilin node to the coilout node, with an inductance of 100 nanohenries.

```
L1 coilin coilout 100n
```

Example 2

The Lloop inductor connects from node 12 to node 17. Its inductance is 1 microhenry, and its temperature coefficients are 0.001 and 0.

```
Lloop 12 17 L=1u TC1=0.001 TC2=0
```

Example 3

The Lcoil inductor connects from the input node to ground. Its inductance is determined by the product of the current through the inductor, and 1E-6.

```
Lcoil input gnd L='1u*i(input)' LTYPE=0
```

Example 4

The L99 inductor connects from the in node to the out node. Its inductance is determined by the polynomial L=c0 + c1*i + c2*i*i, where i is the current through the inductor. The inductor also has a specified DC resistance of 10 ohms.

```
L99 in out POLY 4.0 0.35 0.01 R=10
```

Example 5

The `L` inductor connects from node 1 to node, as a magnetic winding element, with 10 turns of wire.

```
L 1 2 NT=10
```

Mutual Inductors

General form:

`Kxxx Lyyy Lzzz <K=coupling | coupling>`

Mutual core form:

`Kaaa Lbbb <Lccc ... <Lddd>> mname <MAG=magnetization>`

Parameter	Description
Kxxx	Mutual inductor element name. Must begin with K, followed by up to 1023 alphanumeric characters.
Lyyy	Name of the first of two coupled inductors.
Lzzz	Name of the second of two coupled inductors.
K=coupling	Coefficient of mutual coupling. K is a unitless number, with magnitude > 0 and < 1. If K is negative, the direction of coupling reverses. This is equivalent to reversing the polarity of either of the coupled inductors. Use the K=coupling syntax when using a parameter value or an equation, and the keyword “k=” can be omitted.
Kaaa	Saturable core element name. Must begin with K, followed by up to 1023 alphanumeric characters.

Parameter	Description
Lbbb, Lccc, Lddd	Names of the windings about the Kaaa core. One winding element is required, and each winding element must use the magnetic winding syntax. All winding elements with the same magnetic core model should be written in one mutual inductor statement in the netlist.
mname	Saturable core model name. (See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for more information.)
MAG= magnetization	Initial magnetization of the saturable core. You can set this to +1, 0, or -1, where +/- 1 refer to positive and negative values of the BS model parameter. (See the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual* for more information.)

In this syntax, *coupling* is a unitless value, from zero to one, representing the coupling strength. If you use parameter labels, the nodes and model name must be first. Other arguments can be in any order. If you specify an inductor model (see the "Passive Device Models" chapter in the *HSPICE Elements and Device Models Manual*), the inductance value is optional.

You can determine the coupling coefficient, based on geometric and spatial information. To determine the final coupling inductance, HSPICE or HSPICE RF divides the coupling coefficient by the square-root of the product of the self-inductances.

When using the mutual inductor element to calculate the coupling between more than two inductors, HSPICE or HSPICE RF can automatically calculate an approximate second-order coupling. See the third example below for a specific situation.

Note:

> The automatic inductance calculation is an estimation, and is accurate for a subset of geometries. The second-order coupling coefficient is the product of the two first-order coefficients, which is not correct for many geometries.

Example 1

The Lin and Lout inductors are coupled, with a coefficient of 0.9.

```
K1 Lin Lout 0.9
```

Example 2

The Lhigh and Llow inductors are coupled, with a coefficient equal to the value of the COUPLE parameter.

```
Kxfmr Lhigh Llow K=COUPLE
```

- The K1 mutual inductor couples L1 and L2.
- The K2 mutual inductor couples L2 and L3.

Example 3

The coupling coefficients are 0.98 and 0.87. HSPICE or HSPICE RF automatically calculates the mutual inductance between L1 and L3, with a coefficient of 0.98*0.87=0.853.

```
K1 L1 L2 0.98
K2 L2 L3 0.87
```

Ideal Transformer

```
Kxxx Li Lj <k=IDEAL | IDEAL>
```

Ideal transformers use the IDEAL keyword with the K element to designate ideal K transformer coupling.

This keyword activates the following equation set for non-DC values, which is presented here with multiple coupled inductors. Ij is the current into the first terminal of Lj.

```
V1/sqrt(L1)=V2/sqrt(L2)=V3/sqrt(L3)=V4/sqrt(L4)=...
(I1*sqrt(L1) + (I2*sqrt(L2) + (I3*sqrt(L3) + (I4*sqrt(L4) +
     ...=0
```

HSPICE can solve any I or V in terms of L ratios. DC is treated as expected—inductors are treated as short circuits. Mutual coupling is ignored for DC.

Inductors that use the INFINITY keyword can be coupled with IDEAL K elements. In this situation, all inductors involved must have the INFINITY value, and for K=IDEAL, the ratio of all L values is unity. Then, for two L values:

```
v2= v1
i2 + i1=0
```

Example 1

This example is a standard 5-pin ideal balun transformer subcircuit. Two pins are grounded for standard operation. With all `K` values being `IDEAL`, the absolute `L` values are not crucial—only their ratios are important.

```
**
**   all K's ideal  -----o out1
**                  Lo1=.25
**   o----in-       -----o 0
**         Lin=1    Lo2=.25
** 0 o-------       -----o out2
**
.subckt BALUN1  in  out1  out2
Lin    in    gnd   L=1
Lo1    out1  gnd   L=0.25
Lo2    gnd   out2  L=0.25
K12    Lin  Lo1    IDEAL
K13    Lin  Lo2    IDEAL
K23    Lo1  Lo2    IDEAL
.ends
```

Example 2

This example is a 2-pin ideal 4:1 step-up balun transformer subcircuit with shared DC path (no DC isolation). Input and output have a common pin, and both inductors have the same value. Note that Rload=4*Rin.

```
**
**   all K's ideal
**in o-------------------o out=in
**                  L1=1
**                  -----o 0
**                  L2=1
**                  -----o out2
**
** With all K's ideal, the actual L's values are
** not important -- only their ratio to each other.
.subckt BALUN2 in  out2
L1     in   gnd   L=1
L2     gnd  out2  L=1
K12    L1   L2   IDEAL
.ends
```

Example 3

This example is a 3-pin ideal balun transformer with shared DC path (no DC isolation). All inductors have the same value (here set to unity).

```
**
**   all K's ideal  -----o out1
**                  Lo2=1
**                  -----o 0
**                  Lo1=1
**                  -----o out2
**    in            Lin=1
**   o-------------------o in
**
.subckt BALUN3 in  out1  out2
Lo2    gnd  out1  L=1
Lo1    out2 gnd   L=1
Lin    in   out2  L=1
K12    Lin  Lo1  IDEAL
K13    Lin  Lo2  IDEAL
K23    Lo1  Lo2  IDEAL
.ends
```

Linear Inductors

L*xxx node1 node2* <L => *inductance* <TC1=*val*> <TC2=*val*>
+ <M=*val*> <DTEMP=*val*> <IC=*val*>

Parameter	Description
Lxxx	Name of an inductor.
node1 and node2	Names or numbers of the connecting nodes.
inductance	Nominal inductance value, in Henries.
L	Inductance, in Henries, at room temperature.
TC1, TC2	Temperature coefficient.
M	Multiplier for parallel inductors.
DTEMP	Temperature difference between the element and the circuit.
IC	Initial inductor current.

Example:

```
LX A B 1E-9
LR 1 0 1u IC=10mA
```

- LX is a 1 nH inductor.
- LR is a 1 uH inductor, with an initial current of 10 mA.

Frequency-Dependent Inductors

You can specify frequency-dependent inductors using the L='equation' with the HERTZ keyword. The HERTZ keyword represents the operating frequency. In time domain analyses, an expression with the HERTZ keyword behaves differently according to the value assigned to the CONVOLUTION keyword.

Syntax

Lxxx *n1 n2* L='*equation*' <CONVOLUTION=[0|1|2] <FBASE=*value*> + <FMAX=*value*>>

Parameter	Description
Lxxx	Inductor element name. Must begin with L, followed by up to 1023 alphanumeric characters
n1 n2	Positive and negative terminal node names.
equation	The equation should be a function of HERTZ. If CONVOLUTION is turned on when a HERTZ keyword is not used in the equation, CONVOLUTION is automatically be turned off and the inductor behaves conventionally.The equation can be a function of temperature, but it does not support variables of node voltage, branch current, or time. If these variables exist in the equation with CONVOLUTION turned on, only their values at the operating point are considered in the calculation.
CONVOLUTION	Indicates which method is used. ■ 0 (default): Acts the same as the conventional method. ■ 1 : Applies recursive convolution, and if the rational function is not accurate enough, it switches to linear convolution. ■ 2 : Applies linear convolution.

Parameter	Description
FBASE	Specifies the lower bound of the transient analysis frequency. ▪ For CONVOLUTION=1 mode, HSPICE starts sampling at this frequency. ▪ For CONVOLUTION=2 mode, HSPICE uses this value as the base frequency point for Inverse Fourier Transformation. ▪ For recursive convolution, the default value is 0Hz. ▪ For linear convolution, HSPICE uses the reciprocal of the transient period.
FMAX	Specifies the possible maximum frequency of interest. The default value is the frequency point where the function reaches close enough to infinity value, assuming that the monotonous function is approaching the infinity value and that it is taken at 10THz.

Example

```
L1 1 2 L='0.5n + 0.5n/(1 + HERTZ/1e8)' CONVOLUTION=1 fbase=10
+ fmax=30meg
```

AC Choke Inductors

Syntax

```
Lxxx node1 node2  <L=> INFINITY  <IC=val>
```

When the inductance of an inductor is infinity, this element is called an “AC choke.” In HSPICE, you specify an `INFINITY` value for inductors.

HSPICE does not support any other inductor parameters, because it assumes that the infinite inductance value is independent of temperature and scaling factors. The AC choke acts as a short circuit for all DC analyses and HSPICE calculates the DC current through the inductor. In all other (non-DC) analyses, a DC current source of this value represents the choke—HSPICE does not allow `di/dt` variations.

To properly simulate power-line inductors with HSPICE RF, either set them to analog mode or invoke the `SIM_RAIL` option:

```
.OPTION SIM_ANALOG=“L1”
```

-or-

```
.OPTION SIM_RAIL=ON
```

Reluctors

Syntax

Reluctance Inline Form

```
Lxxx n1p n1n ... nNp nNn
+ RELUCTANCE=(r1, c1, val1, r2, c2, val2, ... , rm, cm, valm)
+ <SHORTALL=yes | no> <IGNORE_COUPLING=yes | no>
```

Reluctance External File Form

```
Lxxx n1p n1n ... nNp nNn RELUCTANCE
+ FILE="<filename1>"  [FILE="<filename2>" [...]]
+ <SHORTALL=yes | no> <IGNORE_COUPLING=yes | no>
```

Parameter	Description
Lxxx	Name of a reluctor. Must begin with L, followed by up to 1023 alphanumeric characters
n1p n1n ... nNp nNn	Names of the connecting terminal nodes. The number of terminals must be even. Each pair of ports represnets the location of an inductor.
RELUCTANCE	Keyword to specify reluctance (inverse inductance).
r1, c1, val1, r2, c2, val2, ... rm, cm, valm	Reluctance matrix data. In general, K will be sparse and only non-zero values in the matrix need be given. Each matrix entry is represented by a triplet (r,c,val). The value r and c are integers referring to a pair of inductors from the list of terminal nodes. If there are 2*N terminal nodes, there will be N inductors, and the r and c values must be in the range [1,N]. The val value is a reluctance value for the (r,c) matrix location, and the unit for reluctance is the inverse Henry (H^{-1}). Only terms along and above the diagonal are specified for the reluctance_matrix. The simulator fills in the lower triangle to ensure symmetry. If you specify lower diagonal terms, the simulator converts that entry to the appropriate upper diagonal term. If multiple entries are supplied for the same (r,c) location, then only the first one is used, and a warning will be issued indicating that some entries are ignored. All diagonal entries of the reluctance matrix must be assigned a positive value. The reluctance matrix should be positive definite.

Parameter	Description
FILE="<filename1>"	For the external file format, the data files should contain three columns of data. Each row should contain an (r,c,val) triplet separated by white space. The r, c, and val values may be expressions surrounded by single quotes. Multiple files may be specified to allow the reluctance data to be spread over several files if necessary.
SHORTALL	▪ SHORTALL=yes, all inductors in this model are converted to short circuits, and all reluctance matrix values are ignored. ▪ SHORTALL=no (default), inductors are not converted to short circuits, and reluctance matrix values are not ignored.
IGNORE_COUPLING	▪ IGNORE_COUPLING=yes, all off-diagonal terms are ignored (that is, set to zero). ▪ IGNORE_COUPLING=no (default), off-diagonal terms are not ignored.

Example

This example has 9 segments (or ports) with 12 nodes, and can potentially generate a 9x9 reluctance matrix with 81 elements.

```
L_ThreeNets a 1 1 2 2 a_1 b 4 4 5 5 b_1 c 7 7 8 8 c_1
+ RELUCTANCE=(
+ 1  1    103e9
+ 1  4    -34.7e9
+ 1  7    -9.95e9
+ 4  4    114e9
+ 4  7    -34.7e9
+ 7  7    103e9
+ 2  2    103e9
+ 2  5    -34.7e9
+ 2  8    -9.95e9
+ 5  5    114e9
+ 5  8    -34.7e9
+ 8  8    103e9
+ 3  3    103e9
+ 3  6    -34.7e9
+ 3  9    -9.95e9
+ 6  6    114e9
+ 6  9    -34.7e9
+ 9  9    103e9 )
+ SHORTALL = no IGNORE_COUPLING = no
```

Alternatively, the same element could be specified by using:

```
L_ThreeNets a 1 1 2 2 a_1 b 4 4 5 5 b_1 c 7 7 8 8 c_1 RELUCTANCE
+ FILE="reluctance.dat" SHORTALL = no IGNORE_COUPLING = no
```

Where reluctance.dat contains:

```
+ 1  1    103e9
+ 1  4    -34.7e9
+ 1  7    -9.95e9
+ 4  4    114e9
+ 4  7    -34.7e9
+ 7  7    103e9
+ 2  2    103e9
+ 2  5    -34.7e9
+ 2  8    -9.95e9
+ 5  5    114e9
+ 5  8    -34.7e9
+ 8  8    103e9
+ 3  3    103e9
+ 3  6    -34.7e9
+ 3  9    -9.95e9
+ 6  6    114e9
+ 6  9    -34.7e9
+ 9  9    103e9
```

The following shows the mapping between the port numbers and node pairs:

Ports	1	2	3	4	5	6	7	8	9
Node pairs	(a,1)	(1,2)	(2,a_1)	(b,4)	(4,5)	(5,b_1)	(c,7)	(7,8)	(8,c_1)

Active Elements

This section describes the passive elements: diodes and transistors.

Diode Element

Geometric (LEVEL=1) or Non-Geometric (LEVEL=3) form:

```
Dxxx nplus nminus mname <<AREA=>area> <<PJ=>val>
+ <WP=val> <LP=val> <WM=val> <LM=val> <OFF>
+ <IC=vd> <M=val> <DTEMP=val>

Dxxx nplus nminus mname <W=width> <L=length> <WP=val>
+ <LP=val> <WM=val> <LM=val> <OFF> <IC=vd> <M=val>
+ <DTEMP=val>
```

Fowler-Nordheim (LEVEL=2) form:

```
Dxxx nplus nminus mname <W=val <L=val>> <WP=val>
+ <OFF> <IC=vd> <M=val>
```

Parameter	Description
Dxxx	Diode element name. Must begin with D, followed by up to 1023 alphanumeric characters.
nplus	Positive terminal (anode) node name. The series resistor for the equivalent circuit is attached to this terminal.
nminus	Negative terminal (cathode) node name.
mname	Diode model name reference.
AREA	Area of the diode (unitless for LEVEL=1 diode, and square meters for LEVEL=3 diode). This affects saturation currents, capacitances, and resistances (diode model parameters are IK, IKR, JS, CJO, and RS). The SCALE option does not affect the area factor for the LEVEL=1 diode. Default=1.0. Overrides AREA from the diode model. If you do not specify the AREA, HSPICE or HSPICE RF calculates it from the width and length.

Parameter	Description
PJ	Periphery of junction (unitless for LEVEL=1 diode, and meters for LEVEL=3 diode). Overrides PJ from the diode model. If you do not specify PJ, HSPICE or HSPICE RF calculates it from the width and length specifications.
WP	Width of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides WP in the diode model. Default=0.0.
LP	Length of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides LP in the diode model. Default=0.0.
WM	Width of metal capacitor, in meters (for LEVEL=3 diode only). Overrides WM in the diode model. Default=0.0.
LM	Length of metal capacitor, in meters (for LEVEL=3 diode only). Overrides LM in the diode model. Default=0.0.
OFF	Sets the initial condition for this element to OFF, in DC analysis. Default=ON.
IC=vd	Initial voltage, across the diode element. Use this value when you specify the UIC option in the .TRAN statement. The .IC statement overrides this value.
M	Multiplier, to simulate multiple diodes in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
W	Width of the diode, in meters (LEVEL=3 diode model only)
L	Length of the diode, in meters (LEVEL=3 diode model only)

You must specify two nodes and a model name. If you specify other parameters, the nodes and model name must be first and the other parameters can appear in any order.

Example 1

The D1 diode, with anode and cathode, connects to nodes 1 and 2. Diode1 specifies the diode model.

```
D1 1 2 diode1
```

Example 2

The Dprot diode, with anode and cathode, connects to both the output node and ground, references the *firstd* diode model, and specifies an area of 10 (unitless for `LEVEL=1` model). The initial condition has the diode OFF.

```
Dprot output gnd firstd 10 OFF
```

Example 3

The Ddrive diode, with anode and cathode, connects to the driver and output nodes. The width and length are 500 microns. This diode references the model_d diode model.

```
Ddrive driver output model_d W=5e-4 L=5e-4 IC=0.2
```

Bipolar Junction Transistor (BJT) Element

```
Qxxx nc nb ne <ns> mname <area> <OFF>
+ <IC=vbeval,vceval> <M=val> <DTEMP=val>

Qxxx nc nb ne <ns> mname <AREA=area> <AREAB=val>
+ <AREAC=val> <OFF> <VBE=vbeval> <VCE=vceval>
+ <M=val> <DTEMP=val>
```

Parameter	Description
Qxxx	BJT element name. Must begin with Q, then up to 1023 alphanumeric characters.
nc	Collector terminal node name.
nb	Base terminal node name.
ne	Emitter terminal node name.
ns	Substrate terminal node name, which is optional. You can also use the BULK parameter to set this name in the BJT model.
mname	BJT model name reference.
area, AREA=area	Emitter area multiplying factor, which affects currents, resistances, and capacitances. Default=1.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON.

Parameter	Description
IC=vbeval, vceval, VBE, VCE	Initial internal base-emitter voltage (vbeval) and collector-emitter voltage (vceval). HSPICE or HSPICE RF uses this value when the .TRAN statement includes UIC. The .IC statement overrides it.
M	Multiplier, to simulate multiple BJTs in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
AREAB	Base area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.
AREAC	Collector area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.

The only required fields are the collector, base, and emitter nodes, and the model name. The nodes and model name must precede other fields in the netlist.

Example 1

In the Q1 BJT element below:

```
Q1 1 2 3 model_1
```

- The collector connects to node 1.
- The base connects to node 2.
- The emitter connects to node 3.
- model_1 references the BJT model.

Example 2

In the following Qopamp1 BJT element:

```
Qopamp1 c1 b3 e2 s 1stagepnp AREA=1.5 AREAB=2.5
AREAC=3.0
```

- The collector connects to the c1 node.
- The base connects to the b3 node.
- The emitter connects to the e2 node.
- The substrate connects to the s node.
- 1stagepnp references the BJT model.

- The AREA area factor is 1.5.
- The AREAB area factor is 2.5.
- The AREAC area factor is 3.0.

Example 3

In the Qdrive BJT element below:

```
Qdrive driver in output model_npn 0.1
```

- The collector connects to the driver node.
- The base connects to the in node.
- The emitter connects to the output node.
- model_npn references the BJT model.
- The area factor is 0.1.

JFETs and MESFETs

```
Jxxx nd ng ns <nb> mname <<<AREA>=area | <W=val>
+ <L=val>> <OFF> <IC=vdsval,vgsval> <M=val>
+ <DTEMP=val>

Jxxx nd ng ns <nb> mname <<<AREA>=area> | <W=val>
+ <L=val>> <OFF> <VDS=vdsval> <VGS=vgsval>
+ <M=val> <DTEMP=val>
```

Parameter	Description
Jxxx	JFET or MESFET element name. Must begin with J, followed by up to 1023 alphanumeric characters.
nd	Drain terminal node name
ng	Gate terminal node name
ns	Source terminal node name
nb	Bulk terminal node name, which is optional.
mname	JFET or MESFET model name reference

Parameter	Description
area, AREA=area	Area multiplying factor that affects the BETA, RD, RS, IS, CGS, and CGD model parameters. Default=1.0, in units of square meters.
W	FET gate width in meters
L	FET gate length in meters
OFF	Sets initial condition to OFF for this element, in DC analysis. Default=ON.
IC=vdsval, vgsval, VDS, VGS	Initial internal drain-source voltage (vdsval) and gate-source voltage (vgsval). Use this argument when the .TRAN statement contains UIC. The .IC statement overrides it.
M	Multiplier to simulate multiple JFETs or MESFETs in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.

Only drain, gate, and source nodes, and model name fields are required. Node and model names must precede other fields.

Example 1

In the J1 JFET element below:

```
J1 1 2 3 model_1
```

- The drain connects to node 1.
- The source connects to node 2.
- The gate connects to node 3.
- model_1 references the JFET model.

Example 2

In the following Jopamp1 JFET element:

```
Jopamp1 d1 g3 s2 b 1stage AREA=100u
```

- The drain connects to the d1 node.
- The source connects to the g3 node.
- The gate connects to the s2 node.

- 1stage references the JFET model.
- The area is 100 microns.

Example 3

In the Jdrive JFET element below:

```
Jdrive driver in output model_jfet W=10u L=10u
```

- The drain connects to the driver node.
- The source connects to the in node.
- The gate connects to the output node.
- model_jfet references the JFET model.
- The width is 10 microns.
- The length is 10 microns.

MOSFETs

```
Mxxx nd ng ns <nb> mname <<L=>length> <<W=>width>
+ <AD=val> AS=val> <PD=val> <PS=val>
+ <NRD=val> <NRS=val> <RDC=val> <RSC=val> <OFF>
+ <IC=vds,vgs,vbs> <M=val> <DTEMP=val>
+ <GEO=val> <DELVTO=val>
.OPTION WL
Mxxx nd ng ns <nb> mname <width> <length> <other_options...>
```

Parameter	Description
Mxxx	MOSFET element name. Must begin with M, followed by up to 1023 alphanumeric characters.
nd	Drain terminal node name.
ng	Gate terminal node name.
ns	Source terminal node name.
nb	Bulk terminal node name, which is optional. To set this argument in the MOSFET model, use the BULK parameter.
mname	MOSFET model name reference

Parameter	Description
L	MOSFET channel length, in meters. This parameter overrides .OPTION DEFL, with a maximum value of 0.1m. Default=DEFL.
W	MOSFET channel width, in meters. This parameter overrides .OPTION DEFW. Default=DEFW.
AD	Drain diffusion area. Overrides .OPTION DEFAD. Default=DEFAD, if you set the ACM=0 model parameter.
AS	Source diffusion area. Overrides .OPTION DEFAS. Default=DEFAS, if you set the ACM=0 model parameter.
PD	Perimeter of drain junction, including channel edge. Overrides .OPTION DEFPD. Default=DEFAD, if you set the ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
PS	Perimeter of source junction, including channel edge. Overrides .OPTION DEFPS. Default=DEFAS, if you set the ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
NRD	Number of squares of drain diffusion for resistance calculations. Overrides .OPTION DEFNRD. Default=DEFNRD, if you set ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
NRS	Number of squares of source diffusion for resistance calculations. Overrides .OPTION DEFNRS. Default=DEFNRS when you set the MOSFET model parameter ACM=0 or 1. Default=0.0, when you set ACM=2 or 3.
RDC	Additional drain resistance due to contact resistance, in units of ohms. This value overrides the RDC setting in the MOSFET model specification. Default=0.0.
RSC	Additional source resistance due to contact resistance, in units of ohms. This value overrides the RSC setting in the MOSFET model specification. Default=0.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON. This command does not work for depletion devices.
IC=vds, vgs, vbs	Initial voltage across external drain and source (vds), gate and source (vgs), and bulk and source terminals (vbs). Use these arguments with `.TRAN UIC.` `.IC` statements override these values.

Parameter	Description
M	Multiplier, to simulate multiple MOSFETs in parallel. Affects all channel widths, diode leakages, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
GEO	Source/drain sharing selector for a MOSFET model parameter value of ACM=3. Default=0.0.
DELVTO	Zero-bias threshold voltage shift. Default=0.0.

The only required fields are the drain, gate and source nodes, and the model name. The nodes and model name must precede other fields in the netlist. If you did not specify a label, use the second syntax with the `.OPTION WL` statement, to exchange the width and length options.

Example

In the following M1 MOSFET element:

```
M1 1 2 3 model_1
```

- The drain connects to node 1.
- The gate connects to node 2.
- The source connects to node 3.
- model_1 references the MOSFET model.

In the following Mopamp1 MOSFET element:

```
Mopamp1 d1 g3 s2 b 1stage L=2u W=10u
```

- The drain connects to the d1 node.
- The gate connects to the g3 node.
- The source connects to the s2 node.
- 1stage references the MOSFET model.
- The length of the gate is 2 microns.
- The width of the gate is 10 microns.

In the following Mdrive MOSFET element:

```
Mdrive driver in output bsim3v3 W=3u L=0.25u DTEMP=4.0
```

- The drain connects to the driver node.
- The gate connects to the in node.
- The source connects to the output node.
- bsim3v3 references the MOSFET model.
- The length of the gate is 3 microns.
- The width of the gate is 0.25 microns.
- The device temperature is 4 degrees Celsius higher than the circuit temperature.

Transmission Lines

A transmission line is a passive element that connects any two conductors, at any distance apart. One conductor sends the input signal through the transmission line, and the other conductor receives the output signal from the transmission line. The signal that is transmitted from one end of the pair to the other end, is voltage between the conductors.

Examples of transmission lines include:

- Power transmission lines
- Telephone lines
- Waveguides
- Traces on printed circuit boards and multi-chip modules (MCMs)
- Bonding wires in semiconductor IC packages
- On-chip interconnections

W Element

The W element supports five different formats to specify the transmission line properties:

- Model 1: RLGC-Model specification.
  - Internally specified in a .model statement.
  - Externally specified in a different file.

- Model 2: U-Model specification.
  - RLGC input for up to five coupled conductors.
  - Geometric input (planer, coax, twin-lead).
  - Measured-parameter input.
  - Skin effect.
- Model 3: Built-in field solver model.
- Model 4: Frequency-dependent tabular model.
- Model 5: S Parameter Model

W Element Statement

The general syntax for a lossy (W Element) transmission line element is:

RLGC file form:

```
Wxxx in1 <in2 <...inx>> refin out1 <out2 <...outx>>
+ refout <RLGCfile=filename> N=val L=val
```

U Model form:

```
Wxxx in1 <in2 <...inx>> refin out1 <out2 <...outx>>
+ refout <Umodel=modelname> N=val L=val
```

Field solver form:

```
Wxxx in1 <in2 <...inx>> refin out1 <out2 <...outx>>
+ refout <FSmodel=modelname> N=val L=val
```

The number of ports on a single transmission line are not limited. You must provide one input and output port, the ground references, a model or file reference, a number of conductors, and a length. HSPICE RF does not support the Field Solver form of the W element.

S Model form:

```
Wxxx in1 <in2 <...inx>> refin out1 <out2 <...outx>>
+ refout <Smodel=modelname> <NODEMAP=XiYj...> N=val L=val
```

Table Model form:

```
Wxxx in1 in2 <...inx>> refin out1 <out2 <...outx>>
+ refout N=val L=val TABLEMODEL=name
```

Parameter	Description
Wxxx	Lossy (W Element) transmission line element name. Must start with W, followed by up to 1023 alphanumeric characters.
inx	Signal input node for x^{th} transmission line (in1 is required).
refin	Ground reference for input signal
outx	Signal output node for the x^{th} transmission line (each input port must have a corresponding output port).
refout	Ground reference for output signal.
N	Number of conductors (excluding the reference conductor).
L	Physical length of the transmission line, in units of meters.
RLGCfile=filename	File name reference for the file containing the RLGC information for the transmission lines (for syntax, see "Using the W Element" in the *HSPICE Signal Integrity Guide*).
Umodel=modelname	U-model lossy transmission-line model reference name. A lossy transmission line model, used to represent the characteristics of the W-element transmission line.
FSmodel= modelname	Internal field solver model name. References the PETL internal field solver as the source of the transmission-line characteristics (for syntax, see "Using the Field Solver Model"chapter in the *HSPICE Signal Integrity Guide*).

Parameter	Description
NODEMAP	String that assigns each index of the S parameter matrix to one of the W Element terminals. This string must be an array of pairs that consists of a letter and a number, (for example, Xn), where ▪ X= I, i, N, or n to indicate near end (input side) terminal of the W element ▪ X= O, i, F, or f to indicate far end (output side) terminal of the W element. The default value for NODEMAP is "I1I2I3...InO1O2O3...On"
Smodel	S Model name reference, which contains the S parameters of the transmission lines (for the S Model syntax, see the *HSPICE Signal Integrity Guide*).
TABLEMODEL	Name of the frequency-dependent tabular model.

Example 1

The W1 lossy transmission line connects the in node to the out node:

```
W1 in gnd out gnd RLGCfile=cable.rlgc N=1 L=5
```

Where,

- Both signal references are grounded
- The RLGC file is named cable.rlgc
- The transmission line is 5 meters long.

Example 2

The Wcable element is a two-conductor lossy transmission line:

```
Wcable in1 in2 gnd out1 out2 gnd Umodel=umod_1 N=2
+ L=10
```

Where,

- in1 and in2 input nodes connect to the out1 and out2 output node
- Both signal references are grounded.
- umod_1 references the U-model.
- The transmission line is 10 meters long.

Example 3

The Wnet1 element is a five-conductor lossy transmission line:

```
Wnet1 i1 i2 i3 i4 i5 gnd o1 gnd o3 gnd o5 gnd
+ FSmodel=board1 N=5 L=1m
```

Where,

- The i1, i2, i3, i4 and i5 input nodes connect to the o1, o3, and o5 output nodes.
- The i5 input and three outputs (o1, o3, and o5) are all grounded.
- board1 references the Field Solver model.
- The transmission line is 1 millimeter long.

Example 4: S Model Example

```
Wnet1 i1 i2 gnd o1 o2 gnd
+ Smodel=smod_1 nodemap=i1i2o1o2
+ N=2 L=10m
```

Where,

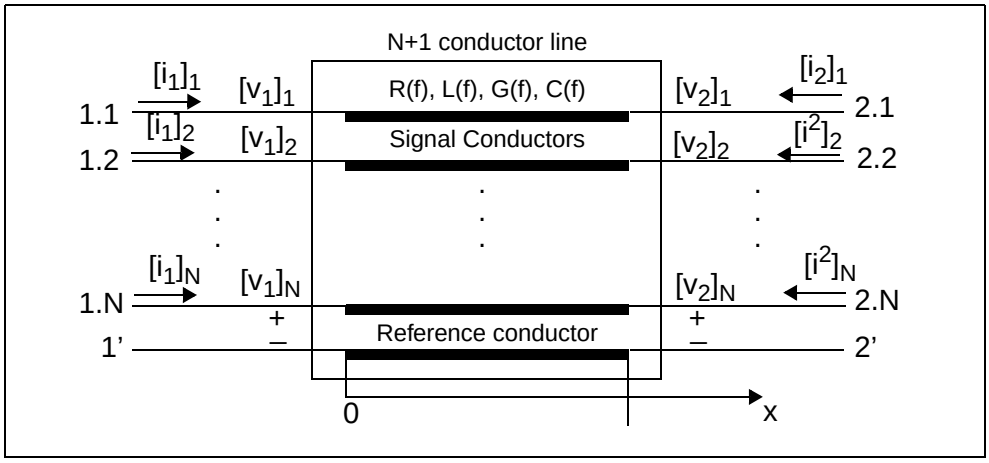
- `in1` and `in2` input nodes connect to the `out1` and `out2` output node.
- Both signal references are grounded.
- `smod_1` references the S Model.
- The transmission line is `10` meters long.

You can specify parameters in the W Element card in any order. You can specify the number of signal conductors, `N`, after the node list. You can also mix nodes and parameters in the W Element card.

You can specify only one of the `RLGCfile`, `FSmodel`, `Umodel`, or `Smodel` models, in a single W Element card.

Figure 13 shows node numbering for the element syntax.

Figure 13 Terminal Node Numbering for the W Element



For additional information about the W element, see the "Modeling Coupled Transmission Lines Using the W Element" chapter in the *HSPICE Signal Integrity User Guide*.

Lossless (T Element)

General form:

```
Txxx in refin out refout Z0=val TD=val <L=val>
+ <IC=v1,i1,v2,i2>

Txxx in refin out refout Z0=val F=val <NL=val>
+ <IC=v1,i1,v2,i2>
```

U Model form:

```
Txxx in refin out refout mname L=val
```

Parameter	Description
Txxx	Lossless transmission line element name. Must begin with T, followed by up to 1023 alphanumeric characters.
in	Signal input node.
refin	Ground reference for the input signal.
out	Signal output node.

refout	Ground reference for the output signal.
Z0	Characteristic impedance of the transmission line.
TD	Signal delay from a transmission line, in seconds per meter.
L	Physical length of the transmission line, in units of meters. Default=1.
IC=v1,i1,v2,i2	Initial conditions of the transmission line. Specify the voltage on the input port (v1), current into the input port (i1), voltage on the output port (v2), and the current into the output port (i2).
F	Frequency at which the transmission line has the electrical length specified in NL.
NL	Normalized electrical length of the transmission line (at the frequency specified in the F parameter), in units of wavelengths per line length. Default=0.25, which is a quarter-wavelength.
mname	U-model reference name. A lossy transmission line model, representing the characteristics of the lossless transmission line.

Only one input and output port is allowed.

Example 1

The T1 transmission line connects the in node to the out node:

```
T1 in gnd out gnd Z0=50 TD=5n L=5
```

- Both signal references are grounded.
- Impedance is 50 ohms.
- The transmission delay is 5 nanoseconds per meter.
- The transmission line is 5 meters long.

Example 2

The Tcable transmission line connects the in1 node to the out1 node:

```
Tcable in1 gnd out1 gnd Z0=100 F=100k NL=1
```

- Both signal references are grounded.
- Impedance is 100 ohms.
- The normalized electrical length is 1 wavelength at 100 kHz.

Example 3

The Tnet1 transmission line connects the driver node to the output node:

```
Tnet1 driver gnd output gnd Umodel1 L=1m
```

- Both signal references are grounded.
- Umodel1 references the U-model.
- The transmission line is 1 millimeter long.

Ideal Transmission Line

For the ideal transmission line, voltage and current will propagate without loss along the length of the line (±x direction) with spatial and time-dependence given according to the following equation:

$$v(x,t) = Re[Ae^{j(\omega t - \beta x)} + Be^{j(\varpi t + \beta x)}]$$

$$v(x,t) = Re\left[\frac{A}{Z_0}e^{j(\omega t - \beta x)} - \frac{B}{Z_0}e^{j(\omega t + \beta x)}\right]$$

The A represents the incident voltage, B represents the reflected voltage, Z_0 is the characteristic impeadance, and β is the propagation constant. The latter are related to the transmission line inductance (L) and capacitance (C) by the following equation:

$$Z_0 = \sqrt{\frac{L}{C}}$$

$$\beta = \omega\sqrt{LC}$$

The L and C terms are in per-unit-length units (Henries/meter, Farads/meter). The following equation gives the phase velocity:

$$\upsilon_\rho = \frac{\omega}{\beta} = \frac{1}{\sqrt{LC}}$$

At the end of the transmission line ($x = l$), the propagation term βl becomes the following equation:

$$\beta l = \omega\sqrt{LC}\cdot l = \omega\frac{l}{v_p}$$

This is equivalent to an ideal delay with the following value:

$$T = \frac{l}{V_P} = \sqrt{LC} \cdot l$$

Where,

T : absolute time delay (sec)

l : physical length (L) (meters)

V_P : phase velocity (meters/sec)

Using standard distance=velocity*time relationships, the HSPICE T element parameter values are related to these terms according to:

$$V_P = f \cdot \lambda = \frac{1}{t_d}$$

Where,

f : frequency

λ : wavelength

t_d : relative time delay (TD) (sec/meter)

$$T = \frac{l}{V_p} = t_d \cdot l = \frac{l}{f \cdot \lambda} = \frac{l/\lambda}{f} = \sqrt{LC} \cdot l$$

Where,

l : physical length (L) (meters)

l/λ : normalized length (NL)

f : frequency at NL (F) (Hz)

$$T = TD \cdot L = \frac{NL}{L} = \sqrt{LC} \cdot L$$

HSPICE therefore allows you to specify a transmission line in three different ways:

- Z_0, TD, L
- Z_0, NL, F
- L, with $\sqrt{\frac{L}{C}}$ and $\sqrt{LC}$ values taken from a U model.

Lossy (U Element)

```
Uxxx in1 <in2 <...in5>> refin out1 <out2 <...out5>>
+ refout mname L=val <LUMPS=val>
```

Parameter	Description
Uxxx	Lossy (U Element) transmission line element name. Must begin with U, followed by up to 1023 alphanumeric characters.
inx	Signal input node for the x^{th} transmission line (in1 is required).
refin	Ground reference for the input signal.
outx	Signal output node for the x^{th} transmission line (each input port must have a corresponding output port).
refout	Ground reference for the output signal.
mname	Model reference name for the U-model lossy transmission-line.
L	Physical length of the transmission line, in units of meters.
LUMPS	Number of lumped-parameter sections used to simulate the element.

In this syntax, the number of ports on a single transmission line is limited to five in and five out. One input and output port, the ground references, a model reference, and a length are all required.

Example 1

The U1 transmission line connects the in node to the out node:

```
U1 in gnd out gnd umodel_RG58 L=5
```

- Both signal references are grounded.
- umodel_RG58 references the U-model.
- The transmission line is 5 meters long.

Example 2

The Ucable transmission line connects the in1 and in2 input nodes to the out1 and out2 output nodes:

```
Ucable in1 in2 gnd out1 out2 gnd twistpr L=10
```

- Both signal references are grounded.
- twistpr references the U-model.
- The transmission line is 10 meters long.

Example 3

The Unet1 element is a five-conductor lossy transmission line:

```
Unet1 i1 i2 i3 i4 i5 gnd o1 gnd o3 gnd o5 gnd Umodel1 L=1m
```

- The i1, i2, i3, i4, and i5 input nodes connect to the o1, o3, and o5 output nodes.
- The i5 input, and the three outputs (o1, o3, and o5) are all grounded.
- Umodel1 references the U-model.
- The transmission line is 1 millimeter long.

Frequency-Dependent Multi-Terminal S Element

The S element uses the following parameters to define a frequency-dependent, multi-terminal network:

- S (scattering)
- Y (admittance)
- Z (impedance)

You can use an S element in the following types of analyses:

- DC
- AC
- Transient
- Small Signal

For a description of the S parameter and SP model analysis, see the "S Parameter Modeling Using the S Element" chapter in the *HSPICE Signal Integrity Guide*.

S Element Syntax (HSPICE):

```
Sxxx nd1 nd2 ... ndN ndRef
+ <MNAME=Smodel_name> <FQMODEL=sp_model_name>
+ <TYPE=[s|y]> <Zo=[value|vector_value]>
+ <FBASE=base_frequency> <FMAX=maximum_frequency>
+ <PRECFAC=val> <DELAYHANDLE=[1|0|ON|OFF]>
```

```
+ <DELAYFREQ=val>
+ <INTERPOLATION=STEP|LINEAR|SPLINE>
+ <INTDATTYP =[RI|MA|DBA]> <HIGHPASS=value>
+ <LOWPASS=value> <MIXEDMODE=[0|1]>
+ <DATATYPE=data_string>
+ <DTEMP=val> <NOISE=[1|0]>
```

S Element Syntax (HSPICE RF):

```
Sxxx nd1 nd2 ... ndN [ndR] s_model_name
```

S model Syntax (HSPICE):

```
.MODEL S_model_name S
+ N=dimension
+ [FQMODEL=sp_model_name | TSTONEFILE=filename |
+ CITIFILE=filename] <TYPE=[s | y]>
+ <Zo=[value | vector_value]>
+ <FBASE=base_frequency> <FMAX=maximum_frequency>
+ <PRECFAC=val> <DELAYHANDLE=ON | OFF> <DELAYFREQ=val>
```

S Model Syntax (HSPICE RF):

```
.model S_model_name S
+ [FQMODEL=sp_model_name | TSTONEFILE=filename |
+ CITIFILE=filename] <TYPE=[S | Y | Z]>
+ <FBASE=base_frequency> <FMAX=max_frequency>
+ <Zo=[50 | vector_value ] | Zof=ref_model>
+ <HIGHPASS=[0 | 1 | 2]> <LOWPASS=[0 | 1 | 2]>
+ <DELAYHANDLE=[0 | 1]> <DELAYFREQ=val>
```

Parameter	Description
nd1 nd2 ... ndN	Nodes of an S element (see Figure 14 on page 115). Three kinds of definitions are present: ▪ With no reference node `ndRef`, the default reference node in this situation is GND. Each node `ndi` (i=1~N) and GND construct one of the N ports of the S element. ▪ With one reference node, `ndRef` is defined. Each node `ndi` (i=1~N) and the `ndRef` construct one of the N ports of the S element. With an N reference node, each port has its own reference node. You can write the node definition in a clearer way as: `nd1+ nd1- nd2+ nd2- ... ndN+ ndN-` Each pair of the nodes (`ndi+` and `ndi-`, i=1~N) constructs one of the N ports of the S element.

Parameter	Description
nd_ref or NdR	Reference node.
MNAME	Name of the S model.
FQMODEL	Frequency behavior of the S,Y, or Z parameters. .MODEL statement of sp type, which defines the frequency-dependent matrices array.
TSTONEFILE	Name of a Touchstone file. Data 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).
CITIFILE	Name of the CITIfile, which is a data file that contains frequency-dependent data. For details, see *Using Instruments with ADS* by Agilent Technologies (http://www.agilent.com).
TYPE	Parameter type: ▪ S (scattering), the default ▪ Y (admittance) ▪ Z (impedance)
Zo	Characteristic impedance value of the reference line (frequency-independent). For multi-terminal lines (N>1), HSPICE assumes that the characteristic impedance matrix of the reference lines are diagonal, and their diagonal values are set to `Zo`. You can also set a vector value for non-uniform diagonal values. Use Zof to specify more general types of a reference-line system. The default is `50`.

Parameter	Description
FBASE	Base frequency used for transient analysis. HSPICE uses this value as the base frequency point for Inverse Fast Fourier Transformation (IFFT). ■ If FBASE is not set, HSPICE uses a reciprocal of the transient period as the base frequency. ■ If FBASE is set smaller than the reciprocal value of transient period, transient analysis performs circular convolution by using the reciprocal value of FBASE as a base period.
FMAX	Maximum frequency for transient analysis. Used as the maximum frequency point for Inverse Fast Fourier Transform (IFFT).
PRECFAC	Preconditioning factor to avoid a singularity (infinite admittance matrix). See Preconditioning S Parameters on page 117. Default=0.75.
DELAYHANDLE	Delay frequency for transmission line type parameters. Default=OFF. ■ 1 of ON activates the delay handler. See Group Delay Handler in Time Domain Analysis on page 116 ■ 0 of OFF (default) deactivates the delay handler. You must set the delay handler, if the delay of the model is longer than the base period specified in the FBASE parameter. If you set DELAYHANDLE=OFF but DELAYFQ is not zero, HSPICE simulates the S element in delay mode.
DELAYFREQ	Delay frequency for transmission-line type parameters. The default is FMAX. If the DELAYHANDLE is set to OFF, but DELAYFREQ is nonzero, HSPICE still simulates the S element in delay mode.
INTERPOLATION	The interpolation method: ■ `STEP`: piecewise step ■ `SPLINE`: b-spline curve fit ■ `LINEAR`: piecewise linear (default)

Parameter	Description
INTDATTYP	Data type for the linear interpolation of the complex data. ▪ `RI`: real-imaginary based interpolation ▪ `DBA`: dB-angle based interpolation ▪ `MA`: magnitude-angle based interpolation (default)
HIGHPASS	Specifies high-frequency extrapolation: 0: Use zero in Y dimension (open circuit). 1: Use highest frequency. 2: Use linear extrapolation, with the highest two points. 3: Apply window function (default). This option overrides EXTRAPOLATION in ,model SP.
LOWPASS	Specifies low-frequency extrapolation: 0: Use zero in Y dimension (open circuit). 1: Use lowest frequency (default). 2: Use linear extrapolation, with the lowest two points. This option overrides EXTRAPOLATION in .model SP.
MIXEDMODE	Set to 1 if the parameters are represented in the mixed mode.
DATATYPE	A string used to determine the order of the indices of the mixed-signal incident or reflected vector. The string must be an array of a letter and a number (Xn) where: ▪ X=D to indicate a differential term =C to indicate a common term =S to indicate a single (grounded) term ▪ n=the port number

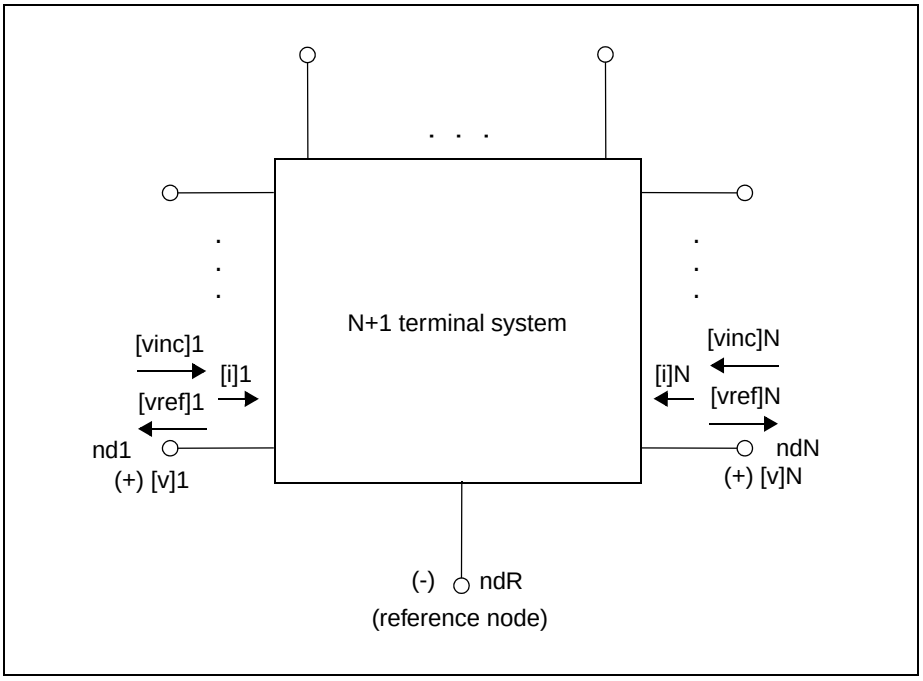
Parameter	Description
DTEMP	Temperature difference between the element and the circuit.[a] Expressed in °C. The default is 0.0.
NOISE	Activates thermal noise. ■ 1 (default): element generates thermal noise ■ 0: element is considered noiseless

a. Circuit temperature is specified by using the .TEMP statement or by sweeping the global TEMP variable in .DC, .AC, or .TRAN statements. When neither .TEMP or TEMP is used, circuit temperature is set by using .OPTION TNOM. The default for TNOM is 25 °C, unless you use .OPTION SPICE, which has a default of 27 °C. You can use the DTEMP parameter to specify the temperature of the element.

You can set all optional parameters, except `MNAME`, in both the S element and the S model statement. Parameters in element statements have higher priorities. You must specify either the `FQMODEL`, `TSTONEFILE`, or `CITIFILE` parameter in either the S model or the S element statement.

When used with the generic frequency-domain model (`.MODEL SP`), an S (scattering) element is a convenient way to describe a multi-terminal network.

Figure 14 Terminal Node Notation



Frequency Table Model

The frequency table model (SP model) is a generic model that you can use to describe frequency-varying behavior. Currently, the S element and `.LIN` command use this model. For a description of this model, see "Small-Signal Parameter Data Frequency Table Model" in the *HSPICE Signal Integrity User Guide*.

Group Delay Handler in Time Domain Analysis

The S element accepts a constant group delay matrix in time-domain analysis. You can also express a weak dependence of the delay matrix on the frequency, as a combination of the constant delay matrix and the phase shift value at each frequency point.

To activate or deactivate this delay handler, specify the `DELAYHANDLE` keyword in the S model statement.

The delay matrix is a constant matrix, which HSPICE RF extracts using finite difference calculation at selected target frequency points. HSPICE RF obtains the $\Upsilon_{\omega(i,j)}$ delay matrix component as:

$$\Upsilon_{\omega(i,j)} = \frac{d\theta_{Sij}}{d\omega} = \frac{1}{2\pi} \cdot \frac{d\theta_{Sij}}{df}$$

- *f* is the target frequency, which you can set using `DELAYFREQ=val`. The default target frequency is the maximum frequency point.
- θ_{Sij} is the phase of *S*ij.

After time domain analysis obtains the group delay matrix, the following equation eliminates the delay amount from the frequency domain system-transfer function:

$$y'_{mn(\omega)} = y_{mn(\omega)} \times e^{j\omega T_{mn}}$$

The convolution process then uses the following equation to calculate the delay:

$$i_{k(t)} = (y'_{k1(t)}, y'_{k2(t)}, \ldots, y'_{kN(t)}) \times \left(v_{1(t-T_{K1})}, v_{2(t-T_{K2})}, \ldots, v_{Nt-T_{KN}}\right)^T$$

Preconditioning S Parameters

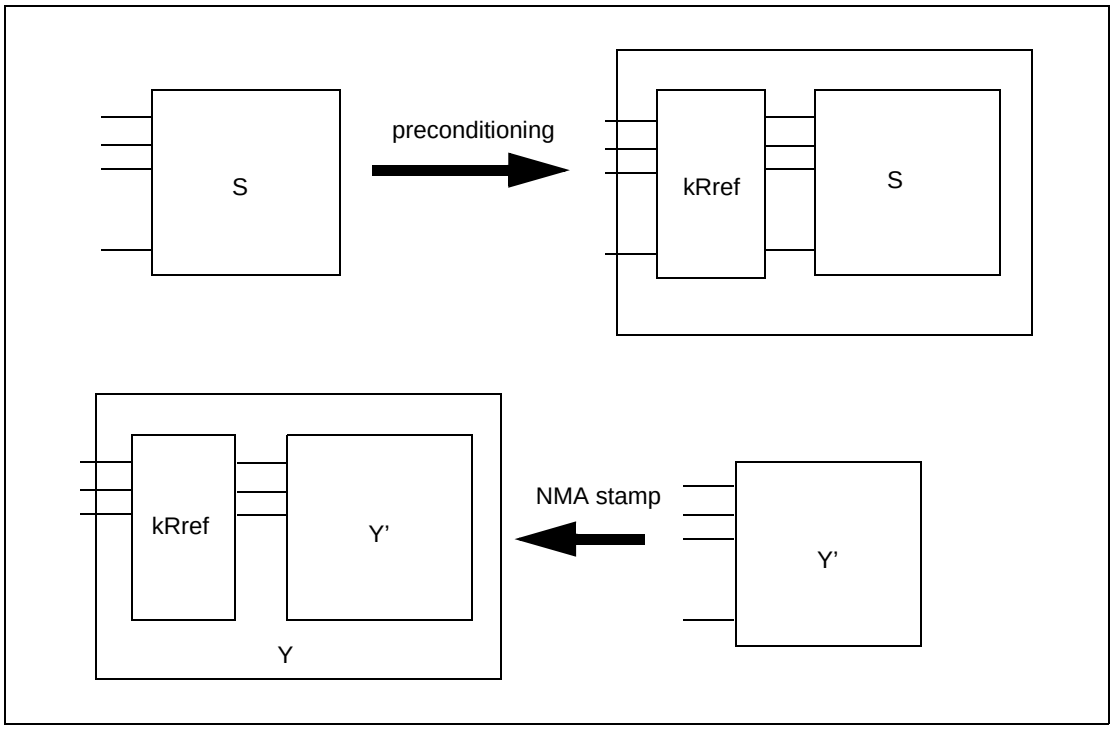
Certain S parameters, such as series inductor (2-port), show a singularity when converting S to Y parameters. To avoid this singularity, the S element preconditions S matrices by adding kR_{ref} series resistance:

$$S' = [kI + (2 - k)S][(2 + k)I - kS]^{-1}$$

- R_{ref} is the reference impedance vector.
- *k* is the preconditioning factor.

To compensate for this modification, the S element adds a negative resistor ($-kR_{ref}$) to the modified nodal analysis (NMA) matrix, in actual circuit compensation. To specify this preconditioning factor, use the `<PREFAC=`*val*`>` keyword in the S model statement. The default preconditioning factor is 0.75.

Figure 15 Preconditioning S Parameters



IBIS Buffers

The general syntax of a B element card for IBIS I/O buffers is:

```
bxxx node_1 node_2 ... node_N
+ file='filename' model='model_name'
+ keyword_1=value_1 ... [keyword_M=value_M]
```

Parameter	Description
bname	Buffer name, and starts with the letter B, which can be followed by up to 1023 alphanumeric characters.
node_1 node_2 ... node_N	List of I/O buffer external nodes. The number of nodes and their meaning are specific to different buffer types.
file='*filename*'	Name of the IBIS file.
model='*model_name*'	Name of the model.
keyword_i=value_i	Assigns a value of value_i to the keyword_i keyword. Specify optional keywords in brackets ([ ]). For more information about keywords, see "Specifying Common Keywords" in the *HSPICE Signal Integrity User Guide*.

Example

```
B1 nd_pc nd_gc nd_in nd_out_of_in
+ buffer=1
+ file='test.ibs'
+ model='IBIS_IN'
```

- This example represents an input buffer named B1.
- The four terminals are named nd_pc, nd_gc, nd_in and nd_out_of_in.
- The IBIS model named IBIS_IN is located in the IBIS file named test.ibs.

  Note:

  HSPICE or HSPICE RF connects the nd_pc and nd_gc nodes to the voltage sources. Do not manually connect these nodes to voltage sources.

For more examples, see the "Modeling Input/Output Buffers Using IBIS" chapter in the *HSPICE Signal Integrity User Guide*.

5

Sources and Stimuli

Describes element and model statements for independent sources, dependent sources, analog-to-digital elements, and digital-to-analog elements.

This chapter also explains each type of element and model statement and provides explicit formulas and examples to show how various combinations of parameters affect the simulation.

Independent Source Elements

Use independent source element statements to specify DC, AC, transient, and mixed independent voltage and current sources. Depending on the analysis performed, the associated analysis sources are used. The value of the DC source is overriden by the zero time value of the transient source when a transient operating point is calculated.

Source Element Conventions

You do not need to ground voltage sources. HSPICE or HSPICE RF assumes that positive current flows from the positive node, through the source, to the negative node. A positive current source forces current to flow out of the n+ node, through the source, and into the n- node.

You can use parameters as values in independent sources. Do not use any of the following reserved keywords to identify these parameters:

AC, ACI, AM, DC, EXP, PAT, PE, PL, PU, PULSE, PWL, R, RD, SFFM, or SIN

Independent Source Element

Syntax

```
Vxxx n+ n- <<DC=> dcval> <tranfun> <AC=acmag> <acphase>>
```

```
Ixxx n+ n- <<DC=> dcval> <tranfun> <AC=acmag> <acphase>>
+ <M=val>
```

Parameter	Description
Vxxx	Independent voltage source element name. Must begin with V, followed by up to 1023 alphanumeric characters.
Ixxx	Independent current source element name. Must begin with I, followed by up to 1023 alphanumeric characters.
n+	Positive node.
n-	Negative node.
DC=dcval	DC source keyword and value, in volts. The tranfun value at time zero overrides the DC value. Default=0.0.
tranfun	Transient source function (one or more of: AM, DC, EXP, PAT, PE, PL, PU, PULSE, PWL, SFFM, SIN). The functions specify the characteristics of a time-varying source. See the individual functions for syntax.
AC	AC source keyword for use in AC small-signal analysis.
acmag	Magnitude (RMS) of the AC source, in volts.
acphase	Phase of the AC source, in degrees. Default=0.0.
M	Multiplier, to simulate multiple parallel current sources. HSPICE or HSPICE RF multiplies source current by M. Default=1.0.

Example 1

```
VX 1 0 5V
```

Where,

- The *VX* voltage source has a 5-volt DC bias.
- The positive terminal connects to node 1.
- The negative terminal is grounded.

Example 2

```
VB 2 0 DC=VCC
```

Where,

- The `VCC` parameter specifies the DC bias for the `VB` voltage source.
- The positive terminal connects to node 2.
- The negative terminal is grounded.

Example 3

```
VH 3 6 DC=2 AC=1,90
```

Where,

- The *VH* voltage source has a 2-volt DC bias, and a 1-volt RMS AC bias, with 90 degree phase offset.
- The positive terminal connects to node 3.
- The negative terminal connects to node 6.

Example 4

```
IG 8 7 PL(1MA 0S 5MA 25MS)
```

Where,

- The piecewise-linear relationship defines the time-varying response for the `IG` current source, which is 1 milliamp at time=0, and 5 milliamps at 25 milliseconds.
- The positive terminal connects to node 8.
- The negative terminal connects to node 7.

Example 5

```
VCC in out VCC PWL 0 0 10NS VCC 15NS VCC 20NS 0
```

Where,

- The VCC parameter specifies the DC bias for the VCC voltage source.
- The piecewise-linear relationship defines the time-varying response for the VCC voltage source, which is 0 volts at time=0, VCC from 10 to 15 nanoseconds, and back to 0 volts at 20 nanoseconds.
- The positive terminal connects to the in node.
- The negative terminal connects to the out node.
- HSPICE or HSPICE RF determines the operating point for this source, without the DC value (the result is 0 volts).

Example 6

```
VIN 13 2 0.001 AC 1 SIN (0 1 1MEG)
```

Where,

- The VIN voltage source has a 0.001-volt DC bias, and a 1-volt RMS AC bias.
- The sinusoidal time-varying response ranges from 0 to 1 volts, with a frequency of 1 megahertz.
- The positive terminal connects to node 13.
- The negative terminal connects to node 2.

Example 7

```
ISRC 23 21 AC 0.333 45.0 SFFM (0 1 10K 5 1K)
```

Where,

- The ISRC current source has a 1/3-amp RMS AC response, with a 45-degree phase offset.
- The frequency-modulated, time-varying response ranges from 0 to 1 volts, with a carrier frequency of 10 kHz, a signal frequency of 1 kHz, and a modulation index of 5.
- The positive terminal connects to node 23.
- The negative terminal connects to node 21.

Example 8

```
VMEAS 12 9
```

Where,

- The VMEAS voltage source has a 0-volt DC bias.
- The positive terminal connects to node 12.
- The negative terminal connects to node 9.

DC Sources

For a DC source, you can specify the DC current or voltage in different ways:

```
V1 1 0 DC=5V
V1 1 0 5V
I1 1 0 DC=5mA
I1 1 0 5mA
```

- The first two examples specify a DC voltage source of 5 V, connected between node 1 and ground.
- The third and fourth examples specify a 5 mA DC current source, between node 1 and ground.

The direction of current in both sources is from node 1 to ground.

AC Sources

AC current and voltage sources are impulse functions, used for an AC analysis. To specify the magnitude and phase of the impulse, use the AC keyword.

```
V1 1 0 AC=10V,90
VIN 1 0 AC 10V 90
```

The preceding two examples specify an AC voltage source, with a magnitude of 10 V and a phase of 90 degrees. To specify the frequency sweep range of the AC analysis, use the `.AC` analysis statement. The AC or frequency domain analysis provides the impulse response of the circuit.

Transient Sources

For transient analysis, you can specify the source as a function of time. The following functions are available:

- Trapezoidal pulse (`PULSE` function)
- Sinusoidal (`SIN` function)
- Exponential (`EXP` function)
- Piecewise linear (`PWL` function)
- Single-frequency FM (`SFFM` function)
- Single-frequency AM (`AM` function)
- Pattern (`PAT` function)

Pseudo Random-Bit Generator Source (`PRBS` function)

Mixed Sources

Mixed sources specify source values for more than one type of analysis. For example, you can specify a DC source, an AC source, and a transient source, all of which connect to the same nodes. In this case, when you run specific analyses, HSPICE or HSPICE RF selects the appropriate DC, AC, or transient source. The exception is the zero-time value of a transient source, which overrides the DC value; it is selected for operating-point calculation for all analyses.

Example

```
VIN 13 2 0.5 AC 1 SIN (0 1 1MEG)
```

Where,

- DC source of 0.5 V
- AC source of 1 V
- Transient damped sinusoidal source

Each source connects between nodes 13 and 2.

For DC analysis, the program uses zero source value, because the sinusoidal source is zero at time zero.

Port Element

The port element identifies the ports used in `.LIN` analysis. Each port element requires a unique port number. If your design uses *N* port elements, your netlist must contain the sequential set of port numbers, 1 through *N* (for example, in a design containing 512 ports, you must number each port sequentially, 1 to 512).

Each port has an associated system impedance, `zo`. If you do not explicitly specify the system impedance, the default is 50 ohms.

The port element behaves as either a noiseless impedance or a voltage source in series with the port impedance for all other analyses (DC, AC, or TRAN).

- You can use this element as a pure terminating resistance or as a voltage or power source.
- You can use the `RDC`, `RAC`, `RHB`, `RHBAC`, and rtran values to override the port impedance value for a particular analysis.

Syntax

```
Pxxx p n port=portnumber
+ $ **** Voltage or Power Information ********
+ <DC mag> <AC <mag <phase>>> <HBAC <mag <phase>>>
+ <HB <mag <phase <harm <tone <modharm <modtone>>>>>>>
+ <transient_waveform> <TRANFORHB=[0|1]>
+ <DCOPEN=[0|1]>
+ $ **** Source Impedance Information ********
+ <Z0=val> <RDC=val> <RAC=val>
+ <RHBAC=val> <RHB=val> <RTRAN=val>
+ $ **** Power Switch ********
+ <power=[0|1|2|W|dbm]>
```

Parameter	Description
port=portnumber	The port number. Numbered sequentially beginning with 1 with no shared port numbers.
<DC mag>	DC voltage or power source value.
<AC <mag <phase>>>	AC voltage or power source value.
<HBAC <mag <phase>>>	(HSPICE RF) HBAC voltage or power source value.

Parameter	Description
<HB <mag <phase <harm <tone <modharm <modtone>>>>>>>	(HSPICE RF) HB voltage, current, or power source value. Multiple HB specifications with different harm, tone, modharm, and modtone values are allowed. ▪ phase is in degrees ▪ harm and tone are indices corresponding to the tones specified in the .HB statement. Indexing starts at 1 (corresponding to the first harmonic of a tone). ▪ modtone and modharm specify sources for multi-tone simulation. A source specifies a tone and a harmonic, and up to 1 offset tone and harmonic (modtone for tones and modharm for harmonics). The signal is then described as: V(or I)=mag*cos(2*pi* (harm*tone+modharm*modtone)*t + phase)
<transient_waveform>	(Transient analysis) Voltage or power source waveform. Any one of waveforms: AM, EXP, PULSE, PWL, SFFM, SIN, or PRBS. Multiple transient descriptions are not allowed.

Parameter	Description
<TRANFORHB=[0|1]>	■ 0 (default): The transient description is ignored if an HB value is given or a DC value is given. If no DC or HB value is given and TRANFORHB=0, then HB analysis treats the source as a DC source, and the DC source value is the time=0 value. ■ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, then the time=infinity value is used as a DC analysis source value. For example, the following statement is treated as a DC source with value=1 for HB analysis: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=1 In contrast, the following statement is a 0V DC source: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=0 The following statement is treated as a periodic source with a 1us period that uses PWL values: v1 1 0 PWL (0 0 1n 1 0.999u 1 1u 0) R + TRANFORHB=1 To override the global TRANFORHB option, explicitly set TRANFORHB for a voltage or current source.
DCOPEN	Switch for open DC connection when DC mag is not set. ■ 0 (default): P element behaves as an impedance termination. ■ 1 : P element is considered an open circuit in DC operating point analysis. DCOPEN=1 is mainly used in .LIN analysis so the P element will not affect the self-biasing device under test by opening the termination at the operating point.
<z0=val>	(LIN analysis) System impedance used when converting to a power source, inserted in series with the voltage source. Currently, this only supports real impedance. ■ When power=0, z0 defaults to 0. ■ When power=1, z0 defaults to 50 ohms. You can also enter zo=val.

Parameter	Description
<RDC=val>	(DC analysis) Series resistance (overrides `z0`).
<RAC=val>	(AC analysis) Series resistance (overrides `z0`).
<RHBAC=val>	(HSPICE RF HBAC analysis) Series resistance (overrides `z0`).
<RHB=val>	(HSPICE RF HB analysis) Series resistance (overrides `z0`).
<RTRAN=val>	(Transient analysis) Series resistance (overrides `z0`).
<power=[0 | 1 | 2 | W | dbm]>	(HSPICE RF) power switch ▪ When 0 (default), element treated as a voltage or current source. ▪ When 1 or W, element treated as a power source, realized as a voltage source with a series impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ▪ When 2 or dbm, element treated as a power source in series with the port imedance. Values are in dbms. You can use this parameter for transient analysis if the power source is either DC or SIN.

Example

For example, the following port element specifications identify a 2-port network with 50-Ohm reference impedances between the "in" and "out" nodes.

```
P1 in gnd port=1 z0=50
P2 out gnd port=2 z0=50
```

Computing scattering parameters requires `z0` reference impedance values. The order of the *port* parameters (in the P Element) determines the order of the S, Y, and Z parameters. Unlike the `.NET` command, the `.LIN` command does not require you to insert additional sources into the circuit. To calculate the requested transfer parameters, HSPICE automatically inserts these sources as needed at the port terminals. You can define an unlimited number of ports.

Independent Source Functions

HSPICE or HSPICE RF uses the following types of independent source functions:

- Trapezoidal pulse (`PULSE` function)
- Sinusoidal (`SIN` function)
- Exponential (`EXP` function)
- Piecewise linear (`PWL` function)
- Single-frequency FM (`SFFM` function)
- Single-frequency AM (`AM` function)
- Pattern (`PAT` function)

Pseudo Random-Bit Generator Source (`PRBS` function)

HSPICE also provides a data-driven version of `PWL` (not supported in HSPICE RF). If you use the data-driven `PWL`, you can reuse the results of an experiment or of a previous simulation, as one or more input sources for a transient simulation.

If you use the independent sources supplied with HSPICE or HSPICE RF, you can specify several useful analog and digital test vectors for steady state, time domain, or frequency domain analysis. For example, in the time domain, you can specify both current and voltage transient waveforms, as exponential, sinusoidal, piecewise linear, AM, or single-sided FM functions.

Trapezoidal Pulse Source

HSPICE or HSPICE RF provides a trapezoidal pulse source function, which starts with an initial delay from the beginning of the transient simulation interval, to an onset ramp. During the onset ramp, the voltage or current changes linearly, from its initial value, to the pulse plateau value. After the pulse plateau, the voltage or current moves linearly, along a recovery ramp, back to its initial value. The entire pulse repeats, with a period named *per*, from onset to onset.

Syntax

```
Vxxx n+ n- PU<LSE> <(>v1 v2 <td <tr <tf <pw <per>>>>> <)>

Ixxx n+ n- PU<LSE> <(>v1 v2 <td <tr <tf <pw + <per>>>>> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the pulse response.
PULSE	Keyword for a pulsed time-varying source. The short form is PU.
v1	Initial value of the voltage or current, before the pulse onset (units of volts or amps).
v2	Pulse plateau value (units of volts or amps).
td	Delay (propagation) time in seconds, from the beginning of the transient interval, to the first onset ramp. Default=0.0; HSPICE or HSPICE RF sets negative values to zero.
tr	Duration of the onset ramp (in seconds), from the initial value, to the pulse plateau value (reverse transit time). Default=TSTEP.
tf	Duration of the recovery ramp (in seconds), from the pulse plateau, back to the initial value (forward transit time). Default=TSTEP.
pw	Pulse width (the width of the plateau portion of the pulse), in seconds. Default=TSTOP.
per	Pulse repetition period, in seconds. Default=TSTOP.

Table 8 Time-Value Relationship for a PULSE Source

Time	Value
0	v1
td	v1
td + tr	v2
td + tr + pw	v2
td + tr + pw + tf	v1

Table 8 Time-Value Relationship for a PULSE Source (Continued)

Time	Value
tstop	v1

Linear interpolation determines the intermediate points.

Note:

TSTEP is the printing increment, and TSTOP is the final time.

Example 1

The following example shows the pulse source, connected between node 3 and node 0. In the pulse:

- The output high voltage is 1 V.
- The output low voltage is -1 V.
- The delay is 2 ns.
- The rise and fall time are each 2 ns.
- The high pulse width is 50 ns.
- The period is 100 ns.

```
VIN 3 0 PULSE (-1 1 2NS 2NS 2NS 50NS 100NS)
```

Example 2

The following example is a pulse source, which connects between node 99 and node 0. The syntax shows parameter values for all specifications.

```
V1 99 0 PU lv hv tdlay tris tfall tpw tper
```

Example 3

The following example shows an entire netlist, which contains a `PULSE` voltage source. In the source:

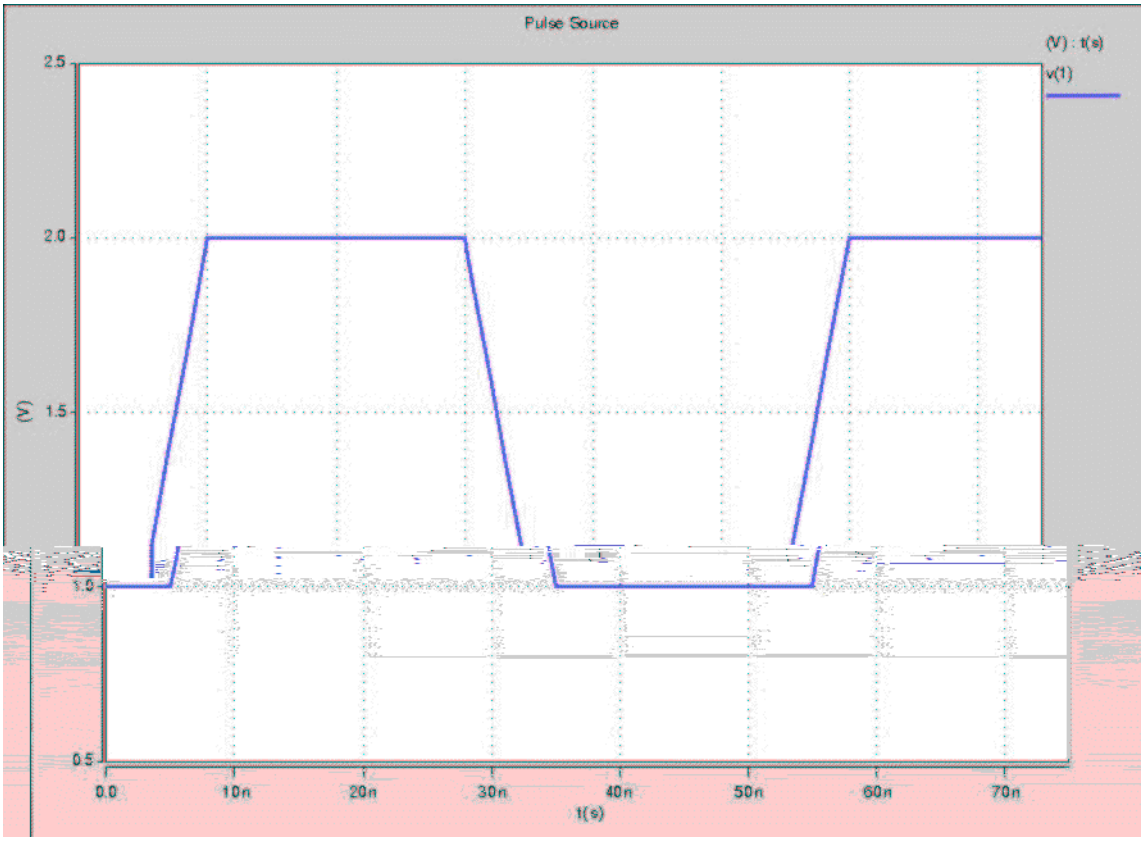
- The initial voltage is 1 volt.
- The pulse voltage is 2 volts.
- The delay time, rise time, and fall time are each 5 nanoseconds.
- The pulse width is 20 nanoseconds.
- The pulse period is 50 nanoseconds.

This example is based on demonstration netlist pulse.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
file pulse.sp test of pulse
.option post
.tran .5ns 75ns
vpulse 1 0 pulse( v1 v2 td tr tf pw per )
r1 1 0 1
.param v1=1v v2=2v td=5ns tr=5ns tf=5ns pw=20ns per=50ns
.end
```

Figure 16 shows the result of simulating this netlist, in HSPICE or HSPICE RF.

Figure 16 Pulse Source Function



Sinusoidal Source Function

HSPICE or HSPICE RF provides a damped sinusoidal source funtion, which is the product of a dying exponential with a sine wave. To apply this waveform, you must specify:

- Sine wave frequency
- Exponential decay constant
- Beginning phase
- Beginning time of the waveform

Syntax

```
Vxxx n+ n- SIN <(> vo va <freq <td <q <j>>>> <)>

Ixxx n+ n- SIN <(> vo va <freq <td <q <j>>>> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits the sinusoidal response.
SIN	Keyword for a sinusoidal time-varying source.
vo	Voltage or current offset, in volts or amps.
va	Voltage or current peak value (vpeak), in volts or amps.
freq	Source frequency in Hz. Default=1/TSTOP.
td	Time (propagation) delay before beginning the sinusoidal variation, in seconds. Default=0.0. Response is 0 volts or amps, until HSPICE or HSPICE RF reaches the delay value, even with a non-zero DC voltage.
q	Damping factor, in units of 1/seconds. Default=0.0.
j	Phase delay, in units of degrees. Default=0.0.

The following table of expressions defines the waveform shape:

Table 9 Waveform Shape Expressions

Time	Value
0 to td	$vo + va \cdot SIN\left(\frac{2 \cdot \Pi \cdot \varphi}{360}\right)$
td to tstop	$vo + va \cdot Exp[-(Time - td) \cdot \theta]$ $SIN\left\{2 \cdot \Pi \cdot \left[freq \cdot (time - td) + \frac{\varphi}{360}\right]\right\}$

In these expressions, TSTOP is the final time.

Example

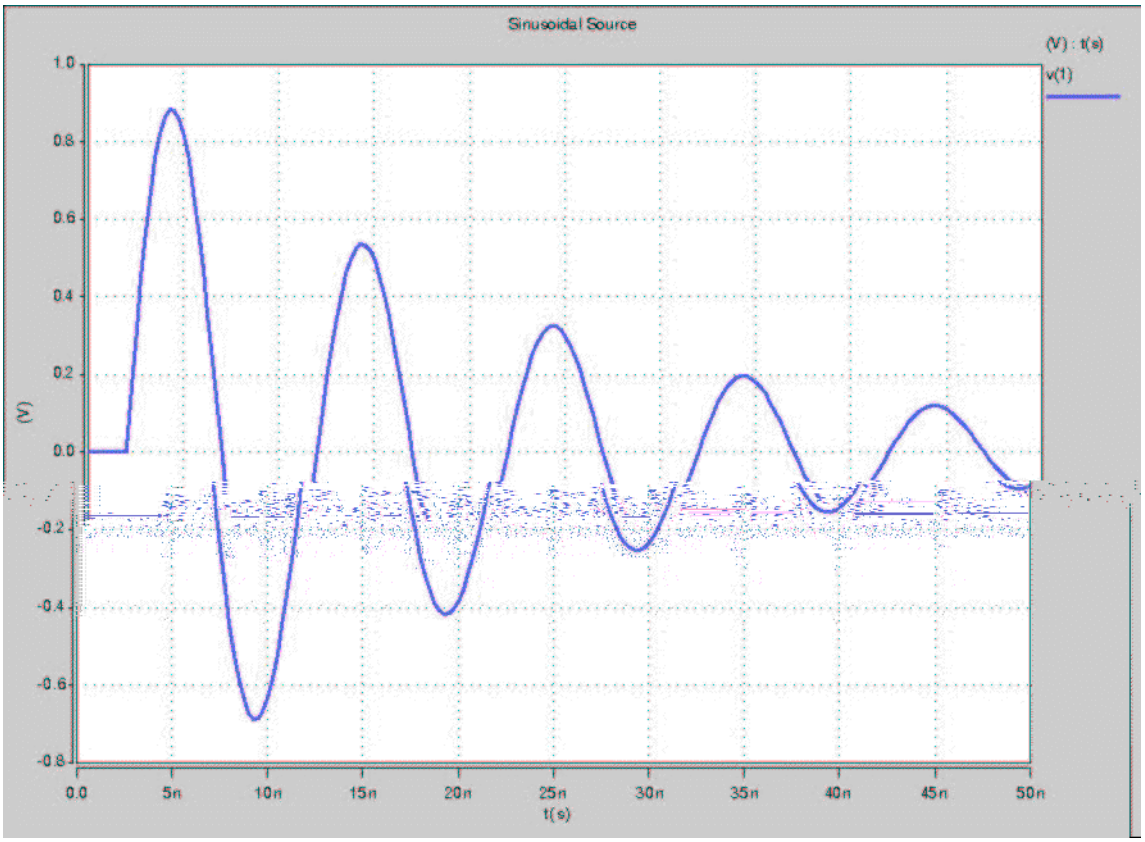
```
VIN 3 0 SIN (0 1 100MEG 1NS 1e10)
```

This damped sinusoidal source connects between nodes 3 and 0. In this waveform:

- Peak value is 1 V.
- Offset is 0 V.
- Frequency is 100 MHz.
- Time delay is 1 ns.
- Damping factor is 1e10.
- Phase delay is zero degrees.

See Figure 17 on page 135 for a plot of the source output.

Figure 17 Sinusoidal Source Function



This example is based on demonstration netlist sin.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
*file: sin.spsinusoidal source
.options post

.param v0=0 va=1 freq=100meg delay=2n theta=5e7 phase=0
v 1 0 sin(v0 va freq delay theta phase)
r 1 0 1
.tran .05n 50n
.end
```

Table 10 SIN Voltage Source

Parameter	Value
initial voltage	0 volts
pulse voltage	1 volt
delay time	2 nanoseconds
frequency	100 MHz
damping factor	50 MHz

Exponential Source Function

HSPICE or HSPICE RF provides a exponential source function, in an independent voltage or current source.

Syntax

```
Vxxx n+ n- EXP <(> v1 v2 <td1 <t1 <td2 <t2>>>> <)>

Ixxx n+ n- EXP <(> v1 v2 <td1 <t1 <td2 <t2>>>> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, exhibiting an exponential response.
EXP	Keyword for an exponential time-varying source.
v1	Initial value of voltage or current, in volts or amps.
v2	Pulsed value of voltage or current, in volts or amps.
td1	Rise delay time, in seconds. Default=0.0.
td2	Fall delay time, in seconds. Default=td1+TSTEP.
t1	Rise time constant, in seconds. Default=TSTEP.
t2	Fall time constant, in seconds. Default=TSTEP.

TSTEP is the printing increment, and TSTOP is the final time.

The following table of expressions defines the waveform shape:

Table 11 Waveform Shape Definitions

Time	Value
0 to td1	v1
td1 to td2	$v1 + (v2 - v1) \cdot \left[1 - exp\left(-\frac{Time - td1}{\tau_1}\right)\right]$
td2 to tstop	$v1 + (v2 - v1) \cdot \left[1 - exp\left(-\frac{(Time - td1)}{\tau_1}\right)\right] + (v1 - v2) \cdot \left[1 - \exp\left(-\frac{(Time - td2)}{\tau_2}\right)\right]$

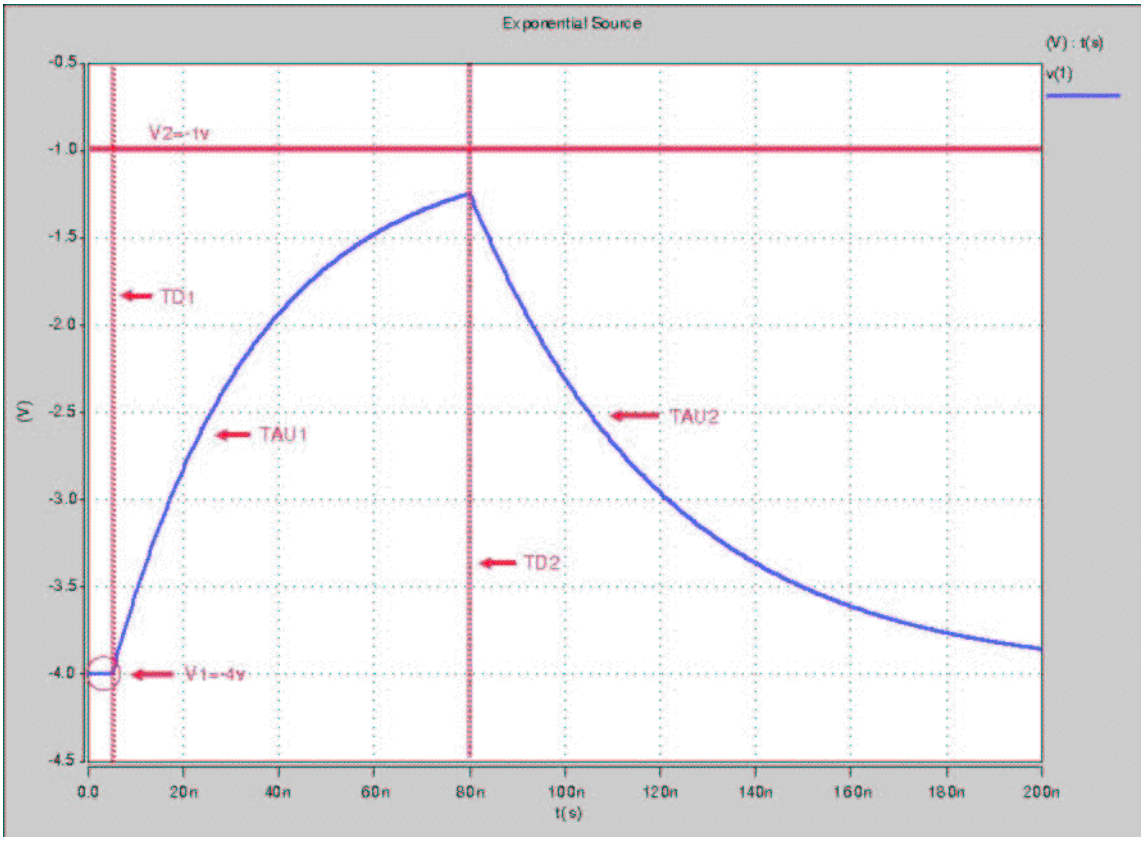
Example

```
VIN 3 0 EXP (-4 -1 2NS 30NS 60NS 40NS)
```

The above example describes an exponential transient source, which connects between nodes 3 and 0. In this source:

- Initial t=0 voltage is -4 V.
- Final voltage is -1 V.
- Waveform rises exponentially, from -4 V to -1 V, with a time constant of 30 ns.
- At 60 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

Figure 18 Exponential Source Function



This example is based on demonstration netlist exp.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
*file: exp.spexponential independant source
.options post
.param v0=-4 va=-1 td1=5n tau1=30n tau2=40n td2=80n
v 1 0 exp(v0 va td1 tau1 td2 tau2)
r 1 0 1
.tran .05n 200n
.end
```

This example shows an entire netlist, which contains an `EXP` voltage source. In this source:

- Initial t=0 voltage is -4 V.
- Final voltage is -1 V.

- Waveform rises exponentially, from -4 V to -1 V, with a time constant of 30 ns.
- At 80 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

Piecewise Linear Source

HSPICE or HSPICE RF provides a piecewise linear source function, in an independent voltage or current source.

General Form

```
Vxxx n+ n- PWL <(> t1 v1 <t2 v2 t3 v3…> <R <=repeat>>
+ <TD=delay> <)>

Ixxx n+ n- PWL <(> t1 v1 <t2 v2 t3 v3…> <R <=repeat>>
+ <TD=delay> <)>
```

MSINC and ASPEC Form

```
Vxxx n+ n- PL <(> v1 t1 <v2 t2 v3 t3…> <R <=repeat>>
+ <TD=delay> <)>

Ixxx n+ n- PL <(> v1 t1 <v2 t2 v3 t3…> <R <=repeat>>
+ <TD=delay> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source; uses a piecewise linear response.
PWL	Keyword for a piecewise linear time-varying source.
v1 v2 … vn	Current or voltage values at the corresponding timepoint.
t1 t2 … tn	Timepoint values, where the corresponding current or voltage value is valid.
R=repeat	Keyword and time value to specify a repeating function. With no argument, the source repeats from the beginning of the function. *repeat* is the time, in units of seconds, which specifies the start point of the waveform to repeat. This time needs to be less than the greatest time point, *tn*.

Parameter	Description
TD=delay	Time, in units of seconds, which specifies the length of time to delay (propagation delay) the piecewise linear function.

- Each pair of values (*t1*, *v1*) specifies that the value of the source is *v1* (in volts or amps), at time *t1*.
- Linear interpolation between the time points determines the value of the source, at intermediate values of time.
- The PL form of the function accommodates ASPEC style formats, and reverses the order of the time-voltage pairs to voltage-time pairs.
- If you do not specify a time-zero point, HSPICE or HSPICE RF uses the DC value of the source, as the time-zero source value.

HSPICE or HSPICE RF does not force the source to terminate at the `TSTOP` value, specified in the `.TRAN` statement.

If the slope of the piecewise linear function changes below a specified tolerance, the timestep algorithm might not choose the specified time points as simulation time points. To obtain a value for the source voltage or current, HSPICE or HSPICE RF extrapolates neighboring values. As a result, the simulated voltage might deviate slightly from the voltage specified in the `PWL` list. To force HSPICE or HSPICE RF to use the specified values, use `.OPTION SLOPETOL`, which reduces the slope change tolerance.

R causes the function to repeat. You can specify a value after this *R*, to indicate the beginning of the function to repeat. The repeat time must equal a breakpoint in the function. For example, if *t1*=1, *t2*=2, *t3*=3, and *t4*=4, then the repeat value can be 1, 2, or 3.

Specify TD=*val* to cause a delay at the beginning of the function. You can use `TD` with or without the repeat function.

Example

This example is based on demonstration netlist pwl.sp, which is available in directory $<installdir>/demo/hspice/sources:

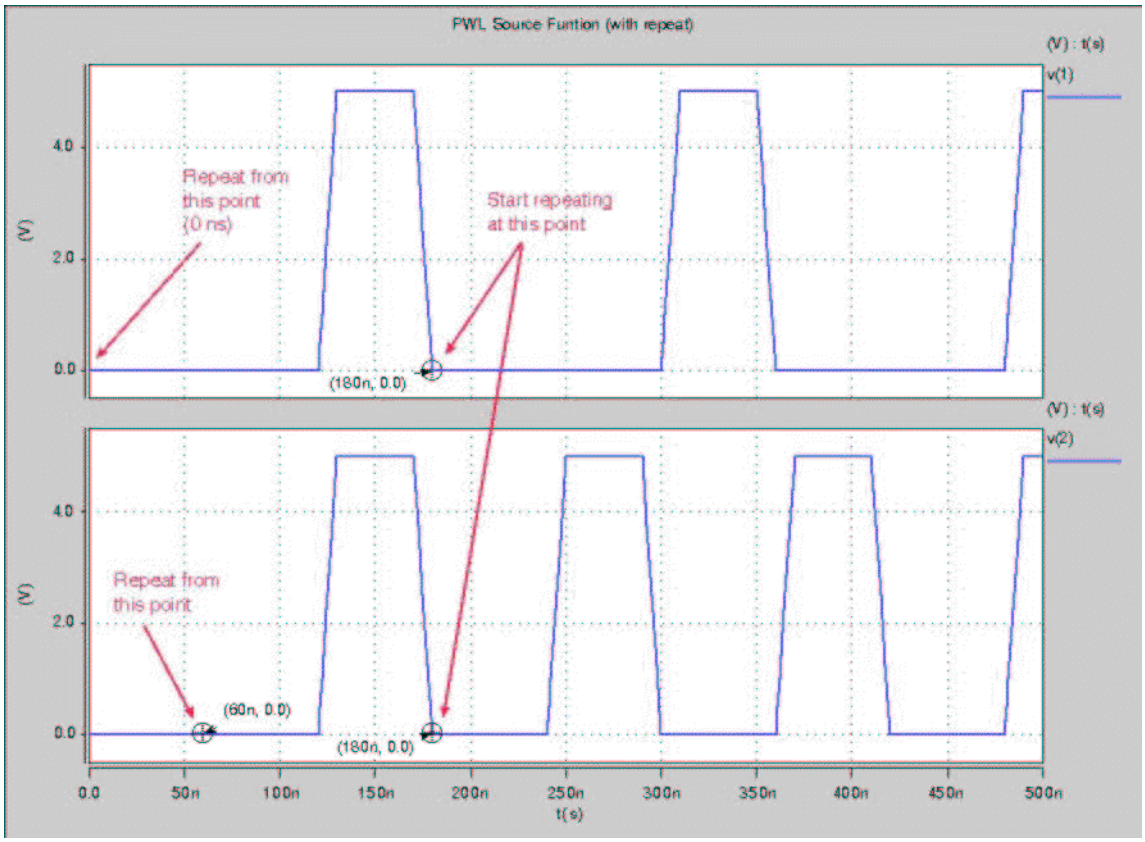
```
file pwl.sp repeated piecewise linear source
.option post
.tran 5n 500n
v1 1 0 pwl 60n 0v, 120n 0v, 130n 5v, 170n 5v, 180n 0v, r
r1 1 0 1

v2 2 0 pl 0v 60n, 0v 120n, 5v 130n, 5v 170n, 0v 180n, r 60n
r2 2 0 1
.end
```

This example shows an entire netlist, which contains two piecewise linear voltage sources. The two sources have the same function:

- First is in normal format. The repeat starts at the beginning of the function.
- Second is in ASPEC format. The repeat starts at the first timepoint.

See Figure 19 for the difference in responses.

Figure 19 Results of Using the Repeat Function



Data-Driven Piecewise Linear Source

HSPICE provides a data-driven piecewise linear source function, in an independent voltage or current source.

Syntax

```
Vxxx n+ n- PWL (TIME, PV)
Ixxx n+ n- PWL (TIME, PV)
.DATA dataname
TIME PV
t1 v1
t2 v2
t3 v3
t4 v4
. . . .
.ENDDATA
.TRAN DATA=datanam
```

Parameter	Description
TIME	Parameter name for time value, provided in a .DATA statement.
PV	Parameter name for amplitude value, provided in a .DATA statement.

You must use this source with a `.DATA` statement that contains time-value pairs. For each t*n*-v*n* (time-value) pair that you specify in the `.DATA` block, the data-driven `PWL` function outputs a current or voltage of the specified t*n* duration and with the specified v*n* amplitude.

When you use this source, you can reuse the results of one simulation, as an input source in another simulation. The transient analysis must be data-driven.

Example

This example is based on demonstration netlist datadriven_pwl.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
*DATA DRIVEN PIECEWISE LINEAR SOURCE
.options list node post
V1 1 0 PWL(TIME, pv1)
R1 1 0 1
V2 2 0 PWL(TIME, pv2)
R2 2 0 1
.DATA dsrc
TIME pv1 pv2
0n 5v 0v
5n 0v 5v
10n 0v 5v
.ENDDATA
.TRAN 1p 10n sweep DATA=dsrc
.END
```

This example is an entire netlist, containing two data-driven, piecewise linear voltage sources. The `.DATA` statement contains the two sets of values referenced in the `pv1` and `pv2` sources. The `.TRAN` statement references the data name.

Single-Frequency FM Source

HSPICE or HSPICE RF provides a single-frequency FM source function, in an independent voltage or current source.

Syntax

```
Vxxx n+ n- SFFM <(> vo va <fc <mdi <fs>>> <)>

Ixxx n+ n- SFFM <(> vo va <fc <mdi <fs>>> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the frequency-modulated response.
SFFM	Keyword for a single-frequency, frequency-modulated, time-varying source.
vo	Output voltage or current offset, in volts or amps.
va	Output voltage or current amplitude, in volts or amps.
fc	Carrier frequency, in Hz. Default=1/TSTOP.
mdi	Modulation index, which determines the magnitude of deviation from the carrier frequency. Values normally lie between 1 and 10. Default=0.0.
fs	Signal frequency, in Hz. Default=1/TSTOP.

The following expression defines the waveform shape:

$$sourcevalue = vo + va \cdot SIN[2 \cdot \pi \cdot fc \cdot Time + mdi \cdot SIN(2 \cdot \pi \cdot fs \cdot Time)]$$

Example

This example is based on demonstration netlist sffm.sp, which is available in directory $<installdir>/demo/hspice/sources:

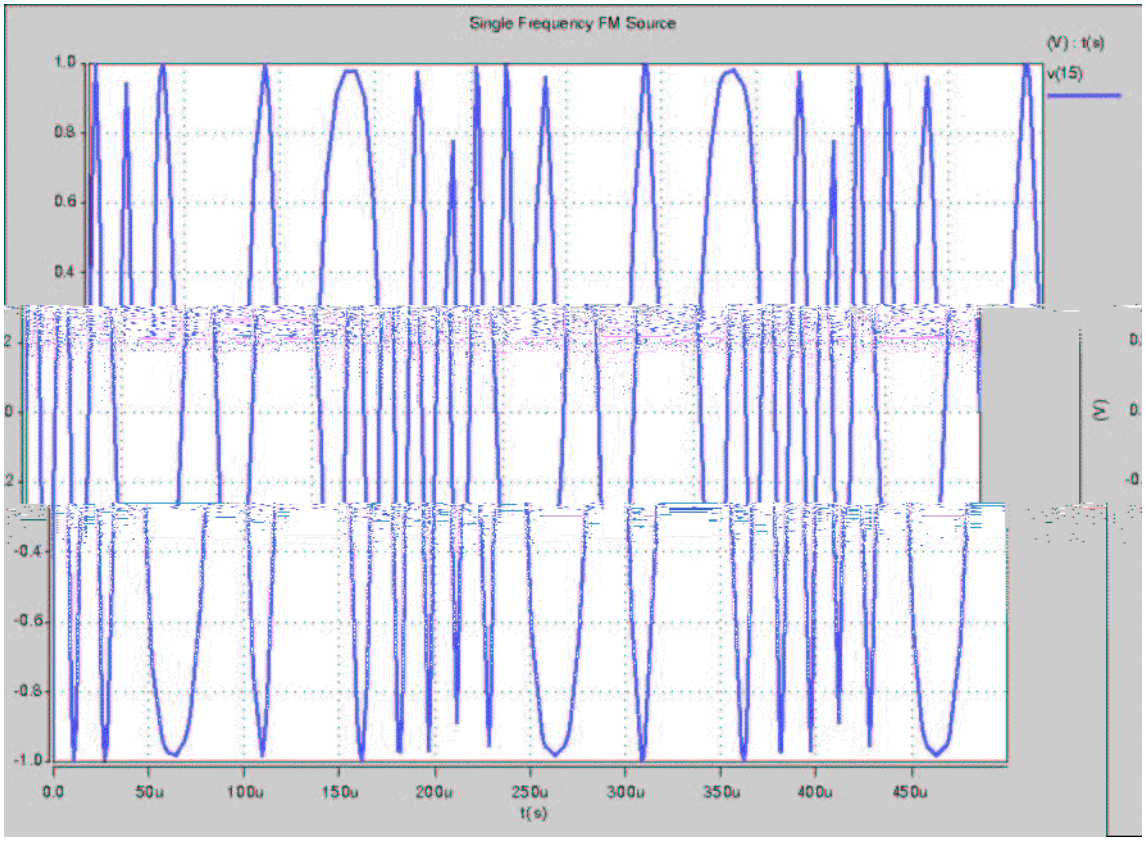
```
*file: sffm.spfrequency modulation source
.options post
vsff1 15 0 dc 3v sffm(0v 1v 20k 10 5k)
rssf1 15 0 1
.tran .001ms .5ms
.probe tran v(15)
.end
```

This example shows an entire netlist, which contains a single-frequency, frequency-modulated voltage source. In this source.

- The offset voltage is 0 volts.
- The maximum voltage is 1 millivolt.

- The carrier frequency is 20 kHz.
- The signal is 5 kHz, with a modulation index of 10 (the maximum wavelength is roughly 10 times as long as the minimum).

Figure 20 Single Frequency FM Source



Single-Frequency AM Source

HSPICE or HSPICE RF provides a single-frequency AM source function in an independent voltage or current source.

Syntax

```
Vxxx n+ n- AM < (> sa oc fm fc <td> <)>
```

```
Ixxx n+ n- AM < (> sa oc fm fc <td> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the amplitude-modulated response.
AM	Keyword for an amplitude-modulated, time-varying source.
sa	Signal amplitude, in volts or amps. Default=0.0.
fc	Carrier frequency, in hertz. Default=0.0.
fm	Modulation frequency, in hertz. Default=1/TSTOP.
oc	Offset constant, a unitless constant that determines the absolute magnitude of the modulation. Default=0.0.
td	Delay time (propagation delay) before the start of the signal, in seconds. Default=0.0.

The following expression defines the waveform shape:

$$sourcevalue = sa \cdot \{oc + SIN[2 \cdot \pi \cdot fm \cdot (Time - td)]\} \cdot SIN[2 \cdot \pi \cdot fc \cdot (Time - td)]$$

Example

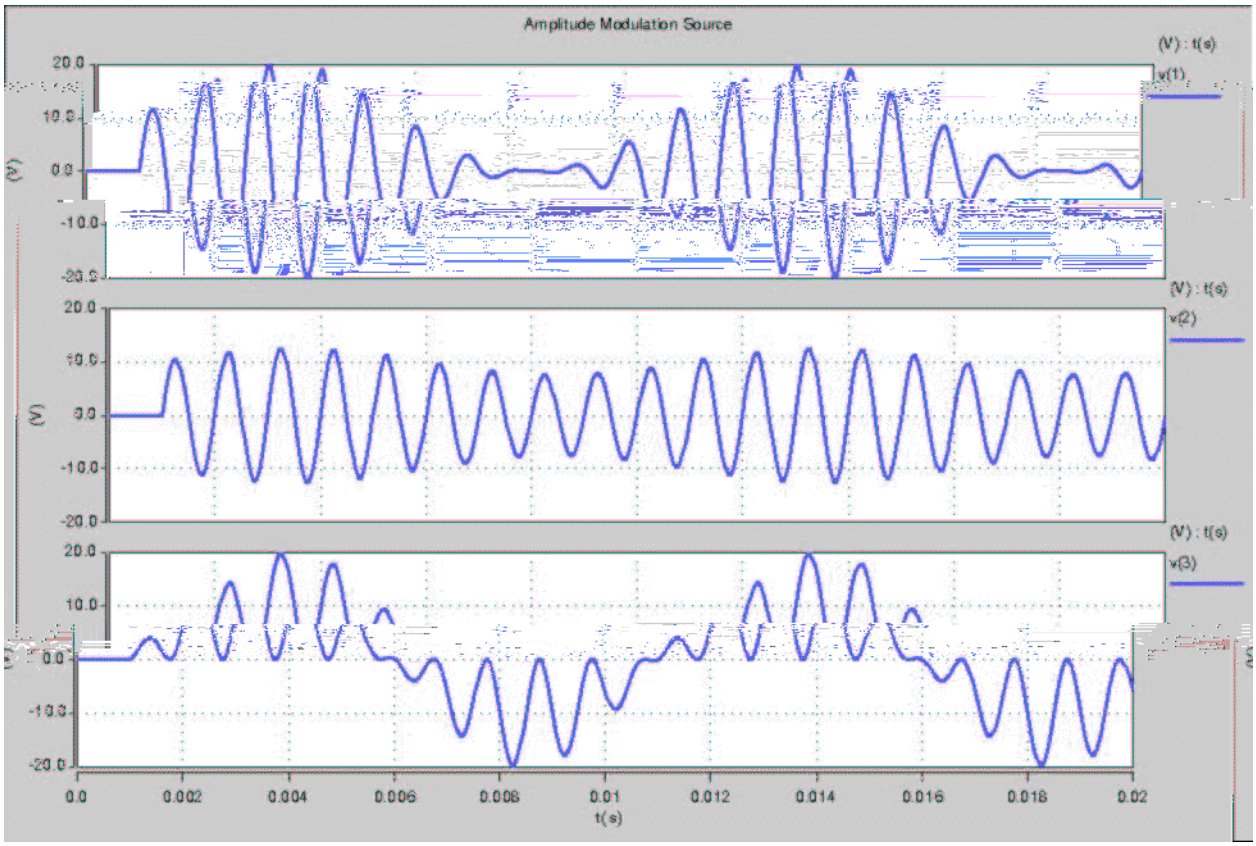
This example is based on demonstration netlist amsrc.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
*file amsrc.sp amplitude modulation
.option post
.tran .01m 20m

v1 1 0 am(10 1 100 1k 1m)
r1 1 0 1

v2 2 0 am(2.5 4 100 1k 1m)
r2 2 0 1

v3 3 0 am(10 1 1k 100 1m)
r3 3 0 1
.end
```

This example shows an entire netlist, which contains three amplitude-modulated voltage sources.

- In the first source:
  - Amplitude is 10.
  - Offset constant is 1.
  - Carrier frequency is 1 kHz.
  - Modulation frequency of 100 Hz.
  - Delay is 1 millisecond.
- In the second source, only the amplitude and offset constant differ from the first source:
  - Amplitude is 2.5.
  - Offset constant is 4.
  - Carrier frequency is 1 kHz.
  - Modulation frequency of 100 Hz.
  - Delay is 1 millisecond.
- The third source exchanges the carrier and modulation frequencies, compared to the first source:
  - Amplitude is 10.
  - Offset constant is 1.
  - Carrier frequency is 100 Hz.
  - Modulation frequency of 1 kHz.
  - Delay is 1 millisecond.

Figure 21 Amplitude Modulation Plot



Pattern Source

HSPICE or HSPICE RF provides a pattern source function, in an independent voltage or current source. The pattern source function uses four states, '1','0','m', and 'z', which represent the high, low, middle voltage, or current and high impedance state respectively. The series of these four states is called a “b-string.”

Syntax

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf  tsample data <RB=val>
+ <R=repeat> <)>

Ixxx n+ n- PAT <(> vhi vlo td tr tf  tsample data <RB=val>
+ <R=repeat> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits a pattern response.
PAT	Keyword for a pattern time-varying source.
vhi	High voltage or current value for pattern sources (units of volts or amps).
vlo	Low voltage or current value for pattern sources (units of volts or amps).
td	Delay (propagation) time in seconds from the beginning of the transient interval to the first onset ramp. It can be negative. The state in the delay time is the same as the first state specified in data.
tr	Duration of the onset ramp (in seconds) from the low value to the high value (reverse transit time).
tf	Duration of the recovery ramp (in seconds) from the high value back to the low value (forward transit time).
tsample	Time spent at '0' or '1' or 'M' or 'Z' pattern value (in seconds).
data	String of '1' ,'0','M', 'Z' representing a pattern source. The first alphabet must be 'B', which represents it is a binary bit stream. This series is called b-string. '1' represents the high voltage or current value, '0' is the low voltage or current value, 'M' represents the value which is equal to 0.5*(vhi+vlo).'Z' represents the high impedance state (only for voltage source).
RB	Keyword to specify the starting bit when repeating. The repeat data starts from the bit indicated by RB. RB must be an integer. If the value is larger than the length of the b-string, an error is reported. If the value is less than 1, it is set to 1 automatically.
R=repeat	Keyword to specify how many times to execute the repeating operation be executed. With no argument, the source repeats from the beginning of the b-string. If R=-1, it means the repeating operation will continue forever. R must be an integer and if it is less than -1, it will be set to 0 automatically.

The time from 0 to the first transition is:

```
tdelay+N*tsample-tr(tf)/2
```

- `N` is the number of the same bit, from the beginning.
- If the first transition is rising, this equation uses `tr`.
- If the first transition is falling, it uses `tf`.

Example

The following example shows a pattern source with two b-strings:

```
*FILE: pattern source gereral form
v1 1 0 pat (5 0 0n 1n 1n 5n b1011 r=1 rb=2 b0m1z)
r1 1 0 1
```

In this pattern:

- High voltage is 5 v
- Low voltage is 0 v
- Time delay is 0 n
- Rise time is 1 n
- Fall time is 1 n
- Sample time is 5 n

The first b-string is `1011`, which repeats once and then repeats from the second bit, which is `0`. The second b-string is `0m1z`. Since neither `R` and `RB` is specified here, they are set to the default value, which is `R=0`, `RB=1`.

Example

The following b-string and its repeat time R and repeating start bit `RB` cannot use a parameter—it is considered as a undivided unit in HSPICE and can only be defined in a `.PAT` command.

```
*FILE:pattern source using parameter
.param td=40ps tr=20ps tf=80ps tsample=400ps
VIN 1 0 PAT (2 0 td tr tf tsample b1010110 r=2)
r1 1 0 1
```

In this pattern:

- High voltage is 2 V.
- Low voltage is 0 V.
- Time delay is 40 ps.

- Rise time is 20 ps.
- Fall time is 80 ps.
- Sample time is 400 ps.
- Data is 1010110.

Nested-Structure Pattern Source HSPICE provides Nested Structure (NS) for the pattern source function to construct complex waveforms. NS is a combination of a b-string and other nested structures defined in a `.PAT` command, which is explained later in this section.

The following general syntax is for an NS pattern source.

Syntax

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf  tsample
+ [component 1 ... component n] <RB=val> <R=repeat> <)>
Ixxx n+ n- PAT <(> vhi vlo td tr tf  tsample
+ [component 1 ... component n] <RB=val> <R=repeat> <) >
```

Parameter	Description
component	Component is the element that makes up NS, which can be a b-string or a patname defined in other `PAT` commands. Brackets ([ ]) must be used.
RB=val	Keyword to specify the starting component when repeating. The repeat data starts from the component indicated by RB. RB must be an integer. If RB is larger than the length of the NS, an error is reported. If RB is less than 1, it is automatically set to 1.
R=repeat	Keyword to specify how many times the repeating operation is executed. With no argument, the source repeats from the beginning of the NS. If R=-1, the repeating operation continuse forever. R must be an integer, and if it is less than -1, it is automatically set to 0.

If the component is a b-string, it can also be followed by `R=repeat` and `RB=val` to specify the repeat time and repeating start bit.

Example

```
*FILE: Pattern source using nested structure
v1 1 0 pat (5 0 0n 1n 1n 5n [b1011 r=1 rb=2 b0m1z] r=2 rb=2)
r1 1 0 1
```

When expanding the nested structure, you get the pattern source like this:

```
'b1011 r=1 rb=2 b0m1z b0m1z b0m1z'
```

The whole NS repeats twice, and each time it repeats from the second b0m1z component.

Pattern-Command Driven Pattern Source The following general syntax is for including a pattern-command driven pattern source in an independent voltage or current source. The RB and R of a b-string or NS can be reset in an independent source. With no argument, the R and RB are the same when defined in the pattern command.

Syntax

Vxxx n+ n- PAT <(> *vhi vlo td tr tf tsample PatName* <RB=*val*> + <R=*repeat*> <)>
Ixxx n+ n- PAT <(> *vhi vlo td tr tf tsample Patname* <RB=*val*> + <R=*repeat*> <)>

Additional syntax applies to the .PAT-command driven pattern source:

```
.PAT <PatName>=data <RB=val> <R=repeat>
.PAT <patName>=[component 1 ... component n] <RB=val>
   <R=repeat>
```

The PatName is the pattern name that has an associated b-string or nested structure.

Example 1

```
v1 1 0 pat (5 0 0n 1n 1n 5n a1 a2 r=2 rb=2)
.PAT a1=b1010 r=1 rb=1
.PAT a2=b0101 r=1 rb=1
```

The final pattern source is:

```
b1010 r=1 rb=1 b0101 r=2 rb=2
```

When the independent source uses the pattern command to specify its pattern source, r and rb can be reset.

Example 2

```
*FILE 2: Pattern source driven by pattern command
v1 1 0 pat (5 0 0n 1n 1n 5n [a1 b0011] r=1 rb=1)
.PAT a1=[b1010 b0101] r=0 rb=1
```

The final pattern source is:

```
b1010 b0101 b0011 b1010 b0101 b0011
```

The `a1` is a predefined NS, and it can be referenced by pattern source.

Pseudo Random-Bit Generator Source

HSPICE or HSPICE RF Pseudo Random Bit Generator Source (PRBS) function, in an independent voltage or current source. This function can be used in several applications from cryptography and bit-error-rate measurement, to wireless communication systems employing spread spectrum or CDMA techniques. In general, PRBS uses a Linear Feedback Shift Register (LFSR) to generate a pseudo random bit sequence.

Syntax

```
Vxxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate seed <[>
+ taps <]> <rout=val> <)>

Ixxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate seed <[>
+ taps <]> <rout=val> <)>
```

Parameter	Description
LFSR	Specifies the voltage or current source as PRBS.
vlow	The minimum voltage or current level.
vhigh	The maximum voltage or current level.
tdelay	Specifies the initial time delay to the first transition.
trise	Specifies the duration of the onset ramp (in seconds), from the initial value to the pulse plateau value (reverse transit time).
tfall	Specifies the duration of the recovery ramp (in seconds), from the pulse plateau, back to the initial value (forward transit time).
rate	The bit rate.
seed	The initial value loaded into the shift register.

Parameter	Description
taps	The bits used to generate feedback.
rout	The output resistance.

Example 1

The following example shows the pattern source that is connected between node in and node gnd:

```
vin in gnd LFSR (0 1 1m 1n 1n 10meg 1 [5, 2] rout=10)
```

Where,

- The output low voltage is `0` , and the output high voltage is `1` v.
- The delay time is `1` ms.
- The rise and fall times are each `1` ns.
- The bit rate is `10meg` bits/s.
- The seed is `1`.
- The taps are [`5, 2]`.
- The output resistance is `10` ohm.
- The output from the LFSR is: 1000010101110110001111100110100...

Example 2

The following example shows the pattern source connected between node 1 and node 0:

```
.PARAM td1=2.5m tr1=2n
vin 1 0 LFSR (2 4 td1 tr1 1n 6meg 2 [10, 5, 3, 2])
```

Where,

- The output low voltage is `2` v, and the output high voltage is `4` v.
- The delay is `2.5` ms.
- The rise time is `2` ns, and the fall time is `1` ns.
- The bit rate is `6meg` bits/s.
- The seed is `2`.
- The taps are `[10, 5, 3, 2]`.
- The output resistance is `0` ohm.

Example 3

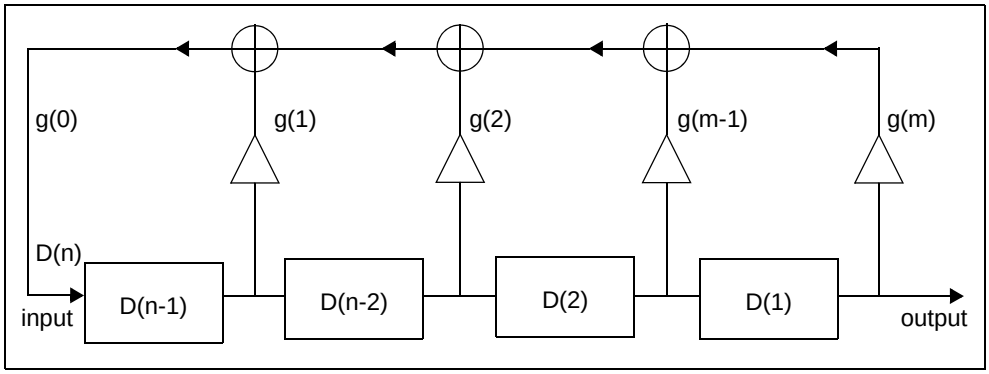
This example is based on demonstration netlist prbs.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
* prbs.sp
.OPTION POST
.TRAN 0.5n 50u
V1 1 0 LFSR (0 1 1u 1n 1n 10meg 1 [5, 2] rout=10)
R1 1 0 1
.END
```

Linear Feedback Shift Register

A LFSR consists of several simple-shift registers in which a binary-weighted modulo-2 sum of the taps is fed back to the input. The modulo-2 sum of two1-bit binary numbers yields 0 if the two numbers are identical and 1 if the differ is 0+0=0, 0+1=1, or 1+1=0.

Figure 22 LFSR Diagram



For any given tap, the weight “gi” is either 0, (meaning "no connection"), or 1, (meaning it is fed back). Two exceptions are g0 and gm, which are always 1 and therefore always connected. The gm is not really a feedback connection, but rather an input of the shift register that is assigned a feedback weight for mathematical purposes.

The maximum number of bits is defined by the first number in your TAPS definition. For example [23, 22, 21, 20, 19, 7] denotes a 23 stage LFSR. The TAPS definition is a specific feedback tap sequence that generates an M-Sequence PRB. The LFSR stages limit is between 2 and 30. The seed cannot be set to zero; HSPICE reports an error and exits the simulation if you set the seed to zero.

Conventions for Feedback Tap Specification

A given set of feedback connections can be expressed in a convenient and easy-to-use shorthand form with the connection numbers listed within a pair of brackets. The g0 connection is implied and not listed since it is always connected. Although gm is also always connected, it is listed in order to convey the shift register size (number of registers).

The following line is a set of feedback taps where j is the total number of feedback taps (not including g0), f(1)=m is the highest-order feedback tap (and the size of the LFSR), and f(j) are the remaining feedback taps:

```
[f(1), f(2), f(3), ..., f(j)]
```

Example

The following line shows that the number of registers is 7 and the total number of feedback taps is 4:

```
[7, 3, 2, 1]
```

The following feedback input applies for this specification:

```
D(n)=[D(n-7)+D(n-3)+D(n-2)+D(n-1)] mod 2
```

Voltage and Current Controlled Elements

HSPICE or HSPICE RF provides two voltage-controlled and two current-controlled elements, known as E, G, H, and F Elements. You can use these controlled elements to model:

- MOS transistors
- bipolar transistors
- tunnel diodes
- SCRs
- analog functions, such as:
  - operational amplifiers
  - summers
  - comparators
  - voltage-controlled oscillators

- modulators
- switched capacitor circuits

Depending on whether you used the polynomial or piecewise linear functions, the controlled elements can be:

- Linear functions of controlling-node voltages.
- Non-linear functions of controlling-node voltages.
- Linear functions of branch currents.
- Non-linear functions of branch currents.

The functions of the E, F, G, and H controlled elements are different.

- The E element can be:
  - A voltage-controlled voltage source
  - A behavioral voltage source
  - An ideal op-amp.
  - An ideal transformer.
  - An ideal delay element.
  - A piecewise linear, voltage-controlled, multi-input AND, NAND, OR, or NOR gate.
- The F element can be:
  - A current-controlled current source.
  - An ideal delay element.
  - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.
- The G element can be:
  - A voltage-controlled current source.
  - A behavioral current source.
  - A voltage-controlled resistor.
  - A piecewise linear, voltage-controlled capacitor.
  - An ideal delay element.
  - A piecewise linear, multi-input AND, NAND, OR, or NOR gate.

- The H element can be:
  - A current-controlled voltage source.
  - An ideal delay element.
  - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.

The next section describes polynomial and piecewise linear functions. Later sections describe element statements for linear or nonlinear functions. For detailed PWL examples, see section "PWL/DATA/VEC Converter" in the *HSPICE Applications Manual*.

Polynomial Functions

You can use the controlled element statement to define the controlled output variable (current, resistance, or voltage), as a polynomial function of one or more voltages or branch currents. You can select three polynomial equations, using the `POLY(NDIM)` parameter in the E, F, G, or H element statement.

Value	Description
POLY(1)	One-dimensional equation (function of one controlling variable).
POLY(2)	Two-dimensional equation (function of two controlling variables).
POLY(3)	Three-dimensional equation (function of three controlling variables).

Each polynomial equation includes polynomial coefficient parameters (P0, P1 … Pn), which you can set to explicitly define the equation.

One-Dimensional Function

If the function is one-dimensional (a function of one branch current or node voltage), the following expression determines the `FV` function value:

$$FV = P0 + (P1 \cdot FA) + (P2 \cdot FA^2) + (P3 \cdot FA^3) + (P4 \cdot FA^4) + (P5 \cdot FA^5) + \ldots$$

Parameter	Description
FV	Controlled voltage or current, from the controlled source.

P0. . .PN	Coefficients of a polynomial equation.
FA	Controlling branch current, or nodal voltage.

Note:

If you specify one coefficient in a one-dimensional polynomial, HSPICE or HSPICE RF assumes that the coefficient is P1 (P0=0.0). Use this as input for linear controlled sources.

The following controlled source statement is a one-dimensional function. This voltage-controlled voltage source connects to nodes 5 and 0.

```
E1 5 0 POLY(1) 3 2 1 2.5
```

In the above source statement, the single-dimension polynomial function parameter, `POLY(1)`, informs HSPICE or HSPICE RF that E1 is a function of the difference of one nodal voltage pair. In this example, the voltage difference is between nodes 3 and 2, so `FA=V(3,2)`.

The dependent source statement then specifies that P0=1 and P1=2.5. From the one-dimensional polynomial equation above, the defining equation for V(5,0) is:

$$V(5, 0) = 1 + 2.5 \cdot V(3,2)$$

You can also express V(5,0) as *E1*:

$$E1 = 1 + 2.5 \cdot V(3,2)$$

Two-Dimensional Function

If the function is two-dimensional (that is, a function of two node voltages or two branch currents), the following expression determines `FV`:

$$FV = P0 + (P1 \cdot FA) + (P2 \cdot FB) + (P3 \cdot FA^2) + (P4 \cdot FA \cdot FB) + (P5 \cdot FB^2) + (P6 \cdot FA^3) + (P7 \cdot FA^2 \cdot FB) + (P8 \cdot FA \cdot FB^2) + (P9 \cdot FB^3) + ...$$

For a two-dimensional polynomial, the controlled source is a function of two nodal voltages or currents. To specify a two-dimensional polynomial, set `POLY(2)` in the controlled source statement.

For example, generate a voltage-controlled source that specifies the controlled voltage, V(1,0), as:

$$V(1,0) = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2$$

or

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2$$

To implement this function, use this controlled-source element statement:

```
E1 1 0 POLY(2) 3 2 7 6 0 3 0 0 0 4
```

This example specifies a controlled voltage source, which connects between nodes 1 and 0. Two differential voltages control this voltage source:

- Voltage difference between nodes 3 and 2.
- Voltage difference between nodes 7 and 6.

That is, `FA=V(3,2)`, and `FB=V(7,6)`. The polynomial coefficients are:

- P0=0
- P1=3
- P2=0
- P3=0
- P4=0
- P5=4

Three-Dimensional Function

For a three-dimensional polynomial function, with `FA`, FB, and FC as its arguments, the following expression determines the `FV` function value:

$$\begin{aligned} FV = {} & P0 + (P1 \cdot FA) + (P2 \cdot FB) + (P3 \cdot FC) + (P4 \cdot FA^2) \\ & + (P5 \cdot FA \cdot FB) + (P6 \cdot FA \cdot FC) + (P7 \cdot FB^2) + (P8 \cdot FB \cdot FC) \\ & + (P9 \cdot FC^2) + (P10 \cdot FA^3) + (P11 \cdot FA^2 \cdot FB) + (P12 \cdot FA^2 \cdot FC) \\ & + (P13 \cdot FA \cdot FB^2) + (P14 \cdot FA \cdot FB \cdot FC) + (P15 \cdot FA \cdot FC^2) \\ & + (P16 \cdot FB^3) + (P17 \cdot FB^2 \cdot FC) + (P18 \cdot FB \cdot FC^2) \\ & + (P19 \cdot FC^3) + (P20 \cdot FA^4) + \ldots \end{aligned}$$

For example, generate a voltage-controlled source that specifies the voltage as:

$$V(1, 0) = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2 + 5 \cdot V(9,8)^3$$

or

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2 + 5 \cdot V(9,8)^3$$

The resulting three-dimensional polynomial equation is:

$$FA = V(3,2)$$

$$FB = V(7,6)$$

$$FC = V(9,8)$$

$$P1 = 3$$

$$P7 = 4$$

$$P19 = 5$$

Substitute these values into the voltage controlled voltage source statement:

```
E1 1 0 POLY(3) 3 2 7 6 9 8 0 3 0 0 0 0 0 4 0 0 0 0 0 0
+ 0 0 0 0 0 5
```

The preceding example specifies a controlled voltage source, which connects between nodes 1 and 0. Three differential voltages control this voltage source:

- Voltage difference between nodes 3 and 2.
- Voltage difference between nodes 7 and 6.
- Voltage difference between nodes 9 and 8.

That is:

- `FA=V(3,2)`
- `FB=V(7,6)`
- `FC=V(9,8)`

The statement defines the polynomial coefficients as:

- P1=3
- P7=4
- P19=5
- Other coefficients are zero.

Piecewise Linear Function

You can use the one-dimensional piecewise linear (PWL) function to model special element characteristics, such as those of:

- tunnel diodes
- silicon-controlled rectifiers
- diode breakdown regions

To describe the piecewise linear function, specify measured data points. Although data points describe the device characteristic, HSPICE or HSPICE RF automatically smooths the corners, to ensure derivative continuity. This, in turn, results in better convergence.

The DELTA parameter controls the curvature of the characteristic at the corners. The smaller the DELTA, the sharper the corners are. The maximum DELTA is limited to half of the smallest breakpoint distance. If the breakpoints are sufficiently separated, specify the DELTA to a proper value.

- You can specify up to 100 point pairs.
- You must specify at least two point pairs (each point consists of an *x* and a *y* coefficient).

To model bidirectional switch or transfer gates, G elements use the NPWL and PPWL functions, which behave the same way as NMOS and PMOS transistors.

You can also use the piecewise linear function to model multi-input AND, NAND,OR, and NOR gates. In this usage, only one input determines the state of the output.

- In AND and NAND gates, the input with the smallest value determines the corresponding output of the gates.
- In OR and NOR gates, the input with the largest value determines the corresponding output of the gates.

Power Sources

This section describes independent sources and controlled sources.

Independent Sources

A power source is a special kind of voltage or current source, which supplies the network with a pre-defined power that varies by time or frequency. The source produces a specific input impedance.

To apply a power source to a network, you can use either:

- A Norton-equivalent circuit (if you specify this circuit and a current source)—the I (current source) element, or
- A Thevenin-equivalent circuit (if you specify this circuit and a voltage source)—the V (voltage source) element.

As with other independent sources, simulation assumes that positive current flows from the positive node, through the source, to the negative node. A power source is a time-variant or frequency-dependent utility source; therefore, the value/phase can be a function of either time or frequency.

A power source is a sub-class of the independent voltage/current source, with some additional keywords or parameters:

- You can use I and V elements in DC, AC, and transient analysis.

The I and V elements can be data-driven.

Supported formats include:

- `PULSE`, a trapezoidal pulse function.
- `PWL`, a piecewise linear function, with repeat function.
- `PL`, a piecewise linear function. `PWL` and `PL` are the same piecewise linear function, except `PL` uses the v1 t1 pair instead of the t1 v1 pair.
- `SIN`, a damped sinusoidal function.
- `EXP`, an exponential function.
- `SFFM`, a single-frequency FM function.

`AM`, an amplitude-modulation function.

Syntax

If you use the *power* keyword in the netlist, then simulation recognizes a current/voltage source as a power source:

```
Vxxx node+ node- power=<powerVal <powerFun>> imp=value1
+ imp_ac=value2,value3 powerFun=<FREQ <TIME>>(...)
Ixxx node+ node- power=<powerVal <powerFun>> imp=value1
+ imp_ac=value2,value3 powerFun=<FREQ <TIME>>(...)
```

Parameter	Description
powerVal	A constant power source supplies the available power. If you specify POWER_DB, then the value is in decibels; otherwise, it is in Watts*POWER_SCAL, where POWER_SCAL is a scaling factor that you specify in a SCALE option (default=1).
powerFun	This function name indicates the time-variant or frequency-variant power source. In this equation, *powerFun* defines the functional dependence on time or frequency. ▪ If the function name for *powerFun* is FREQ, then it is a frequency power source: FREQ(freq1, val1, freq2, val2,...) ▪ If the function name for *powerFun* is TIME, then it is a piece-wise time variant function: TIME(t1, val1, t2, val2...)
imp=	DC impedance value.
imp_ac=	Magnitude and phase offset (in degrees) of AC impedance.

Example 1

```
V11 10 20 power=5 imp=5K
```

This example applies a 5-decibel/unit power source to node 10 and node 20, in a Thevenin-equivalent manner. The impedance of this power source is 5k Ohms.

Example 2

```
Iname 1 0 power=20 imp=9MEG
```

This example applies a 20-decibel/unit power source to node 1 and to ground, in a Norton-equivalent manner. The source impedance is 9 mega-ohms.

Example 3

```
V5 6 0 power=FREQ(10HZ, 2, 10KHZ, 0.01) imp=2MEG imp_ac=(100K, 60)
V5 6 0 power=func1 imp=2MEG imp_ac=(100K, 60DEC)
+ func1=FREQ(10HZ, 2, 10KHZ, 0.01)
```

In the two preceding examples, a power source operates at two different frequencies, with two different values:

- At 10 Hz, the power value is 2 decibel/unit.
- At 10 kHz, the power value is 0.01 decibel/unit.

Also in these examples:

- The DC impedance is 2 mega-ohms.
- The AC impedance is 100 kilo-ohms.
- The phase offset is 60 degrees.

Outputs

None.

Controlled Sources

In addition to independent power sources, you can also create four types of controlled sources:

- Voltage-controlled voltage source (VCVS), or E element
- Current-controlled current source (CCCS), or F element
- Voltage-controlled current source (VCCS), or G element
- Current-controlled voltage source (CCVS), or H element

Voltage-dependent Voltage Sources — E Elements

This section explains E Element syntax statements, and defines their parameters. See also "Using G and E Elements" in the *HSPICE Applications Manual*.

- `LEVEL=1` is an OpAmp.
- `LEVEL=2` is a Transformer.

Voltage-Controlled Voltage Source (VCVS)

Linear

```
Exxx n+ n- <VCVS> in+ in- gain <MAX=val> <MIN=val>
+ <SCALE=val> <TC1=val> <TC2=val><ABS=1> <IC=val>
```

For a description of these parameters, see E Element Parameters on page 173.

Polynomial (POLY)

```
Exxx n+ n- <VCVS> POLY(NDIM) in1+ in1- ...
+ inndim+ inndim-<TC1=val> <TC2=val> <SCALE=val>
+ <MAX=val> <MIN=val> <ABS=1> p0 <p1…> <IC=val>
```

In this syntax, *dim* (*dimensions*) ≤ 3. For a description of these parameters, see E Element Parameters on page 173.

Piecewise Linear (PWL)

```
Exxx n+ n- <VCVS> PWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> x1,y1 x2,y2  ...
+ x100,y100 <IC=val>
```

For a description of these parameters, see E Element Parameters on page 173.

Multi-Input Gates

```
Exxx n+ n- <VCVS> gatetype(k) in1+ in1- ... inj+ inj-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val>
+ x1,y1  ...   x100,y100 <IC=val>
```

In this syntax, *gatetype(k)* can be AND, NAND, OR, or NOR gates. For a description of these parameters, see E Element Parameters on page 173.

Delay Element

```
Exxx n+ n- <VCVS> DELAY in+ in- TD=val <SCALE=val>
+ <TC1=val> <TC2=val> <NPDELAY=val>
```

For a description of these parameters, see E Element Parameters on page 173.

Laplace Transform

Voltage Gain H(s):

```
Exxx n+ n- LAPLACE in+ in-  k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val>
```

For a description of these parameters, see E Element Parameters on page 173.

Transconductance H(s):

```
Gxxx n+ n- LAPLACE in+ in-  k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val> <M=val>
```

H(s) is a rational function, in the following form:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

You can use parameters to define the values of all coefficients (k_0, k_1, ..., d_0, d_1, ...).

For a description of the G Element parameters, see G Element Parameters on page 189.

Example

```
Glowpass 0 out LAPLACE in 0   1.0 / 1.0  2.0  2.0  1.0
Ehipass out 0 LAPLACE in 0  0.0,0.0,0.0,1.0 / 1.0,2.0,2.0,1.0
```

The `Glowpass` element statement describes a third-order low-pass filter, with the transfer function:

$$H(s) = \frac{1}{1 + 2s + 2s^2 + s^3}$$

The `Ehipass` element statement describes a third-order high-pass filter, with the transfer function:

$$H(s) = \frac{s^3}{1 + 2s + 2s^2 + s^3}$$

Pole-Zero Function

Voltage Gain H(s):

```
Exxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm <SCALE=val> <TC1=val>
+ <TC2=val>
```

For a description of these parameters, see E Element Parameters on page 173.

Transconductance H(s):

```
Gxxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm <SCALE=val> <TC1=val>
+ <TC2=val> <M=val>
```

The following equation defines H(s) in terms of poles and zeros:

$$H(s) = \frac{a \cdot (s + \alpha_{z1} - j2\pi f_{z1}) \ldots (s + \alpha_{zn} - j2\pi f_{zn})(s + \alpha_{zn} + j2\pi f_{zn})}{b \cdot (s + \alpha_{p1} - j2\pi f_{p1}) \ldots (s + \alpha_{pm} - j2\pi f_{pm})(s + \alpha_{pm} + j2\pi f_{pm})}$$

The complex poles or zeros are in conjugate pairs. The element description specifies only one of them, and the program includes the conjugate. You can use parameters to specify the a, b, α, and f values.

For a description of the G Element parameters, see G Element Parameters on page 189.

Example

```
Ghigh_pass 0 out POLE in 0 1.0 0.0,0.0 / 1.0 0.001,0.0
Elow_pass out 0 POLE in 0 1.0 / 1.0, 1.0,0.0 0.5,0.1379
```

The `Ghigh_pass` statement describes a high-pass filter, with the transfer function:

$$H(s) = \frac{1.0 \cdot (s + 0.0 + j \cdot 0.0)}{1.0 \cdot (s + 0.001 + j \cdot 0.0)}$$

The `Elow_pass` statement describes a low-pass filter, with the transfer function:

$$H(s) = \frac{1.0}{1.0 \cdot (s + 1)(s + 0.5 + j2\pi \cdot 0.1379)(s + 0.5 - (j2\pi \cdot 0.1379))}$$

Frequency Response Table

Voltage Gain H(s):

```
Exxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val> <LEVEL=val> <ACCURACY=val>
```

For a description of these parameters, see E Element Parameters on page 173

Transconductance H(s):

```
Gxxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val> <M=val> <LEVEL=val> <ACCURACY=val>
```

Where,

- Each `fi` is a frequency point, in hertz.
- `ai` is the magnitude, in dB.
- `f1` is the phase, in degrees.

At each frequency, HSPICE or HSPICE RF uses interpolation to calculate the network response, magnitude, and phase. HSPICE or HSPICE RF interpolates the magnitude (in dB) logarithmically, as a function of frequency. It also interpolates the phase (in degrees) linearly, as a function of frequency.

$$|H(j2\pi f)| = \left(\frac{a_i - a_k}{\log f_i - \log f_k}\right)(\log f - \log f_i) + a_i$$

$$\angle H(j2\pi f) = \left(\frac{\phi_i - \phi_k}{f_i - f_k}\right)(f - f_i) + \phi_i$$

For a description of the G Element parameters, see G Element Parameters on page 189.

Example

```
Eftable output   0 FREQ input   0
+ 1.0k   -3.97m   293.7
+ 2.0k   -2.00m   211.0
+ 3.0k   17.80m   82.45
+  ...... ...
+ 10.0k -53.20    -1125.5
```

- The first column is frequency, in hertz.
- The second column is magnitude, in dB.
- The third column is phase, in degrees.

Set the `LEVEL` to `1` for a high-pass filter.

Set the last frequency point to the highest frequency response value that is a real number, with zero phase.

You can use parameters to set the frequency, magnitude, and phase, in the table.

Foster Pole-Residue Form

Gain E(s) form

```
Exxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1})/ (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2})/ (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3})/ (Re{p3}, Im{p3})
+ ...
```

For a description of these parameters, see E Element Parameters on page 173.

Tranconductance G(s) form

```
Gxxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1})/ (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2})/ (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3})/ (Re{p3}, Im{p3})
+ ...
```

In the above syntax, paranthesis , commas, and slashes are separators—they have the same meaning as a space. A pole-residue pair is represented by four numbers (real and imaginary part of the residue, then real and imaginary part of the pole).

You must make sure that Re[pi]<0; otherwise, the simulations will certainly diverge. Also, it is a good idea to assure passivity of the model (for an N-port admittance matrix Y, Re{Y} should be positive-definite), or the simulation is likely to diverge).

For a description of the G Element parameters, see G Element Parameters on page 189.

```
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is V^2/Hz.

Ideal Op-Amp

`Exxx` *`n+ n-`* `OPAMP` *`in+ in-`*

You can also substitute `LEVEL=1` in place of `OPAMP`:

`Exxx` *`n+ n- in+ in-`* `level=1`

For a description of these parameters, see E Element Parameters.

Ideal Transformer

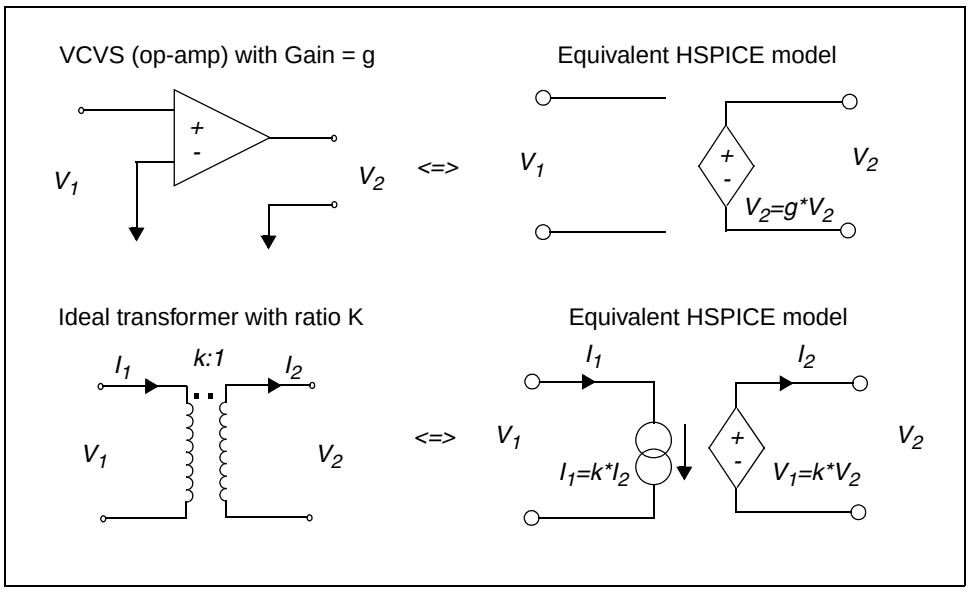
`Exxx` *`n+ n-`* `TRANSFORMER` *`in+ in-`* `k`

You can also substitute `LEVEL=2` in place of `TRANSFORMER`:

`Exxx` *`n+ n- in+ in-`* `level=2 k`

For a description of these parameters, see E Element Parameters.

Figure 23 Equivalent VCVS and Ideal Transformer HSPICE Models



E Element Parameters

The E element parameters described in the following list.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
DELAY	Keyword for the delay element. Same as for the voltage-controlled voltage source, except it has an associated propagation delay, TD. This element adjusts propagation delay in macro (subcircuit) modeling. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of the piecewise linear corners. This parameter defaults to one-fourth of the smallest distance between breakpoints. The maximum is one-half of the smallest distance between breakpoints.
Exxx	Voltage-controlled element name. Must begin with E, followed by up to 1023 alphanumeric characters.
gain	Voltage gain.

Parameter	Description
gatetype(k)	Can be AND, NAND, OR, or NOR. *k* represents the number of inputs of the gate. *x* and *y* represent the piecewise linear variation of output, as a function of input. In multi-input gates, only one input determines the state of the output.
IC	Initial condition: initial estimate of controlling voltage value(s). If you do not specify IC, default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
k	Ideal transformer turn ratio: $V(in+,in-) = k \cdot V(n+,n-)$ or, number of gates input.
MAX	Maximum output voltage value. The default is undefined, and sets no maximum value.
MIN	Minimum output voltage value. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of a controlled element.
NDIM	Number of polynomial dimensions. If you do not set POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$ The .TRAN statement specifies tstep and tstop values.
OPAMP or Level=1	The keyword for an ideal op-amp element. OPAMP is a HSPICE reserved word; do not use it as a node name.

Parameter	Description
P0, P1 …	The polynomial coefficients. If you specify one coefficient, HSPICE or HSPICE RF assumes that it is P1 (P0=0.0), and that the element is linear. If you specify more than one polynomial coefficient, the element is nonlinear, and P0, P1, P2 ... represent them (see Polynomial Functions on page 158).
POLY	Keyword for the polynomial function. If you do not specify `POLY(ndim)`, *HSPICE assumes a one-dimensional polynomial.* *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
TRANSFORMER or LEVEL=2	Keyword for an ideal transformer. TRANSFORMER is a reserved word; do not use it as a node name.
VCVS	Keyword for a voltage-controlled voltage source. VCVS is a reserved word; do not use it as a node name.
x1,...	Controlling voltage across the *in*+ and *in*- nodes. The *x* values must be in increasing order.
y1,...	Corresponding element values of *x*.

E Element Examples

Ideal OpAmp

You can use the voltage-controlled voltage source to build a voltage amplifier, with supply limits.

- The output voltage across nodes 2,3 is v(14,1) * 2.
- The value of the voltage gain parameter is 2.
- The `MAX` parameter sets a maximum E1 voltage of 5 V.
- The `MIN` parameter sets a minimum E1 voltage output of -5 V.

Example

If V(14,1)=-4V, then HSPICE or HSPICE RF sets E1 to -5V, and not -8V as the equation suggests.

```
Eopamp 2 3 14 1 MAX=+5 MIN=-5 2.0
```

To specify a value for polynomial coefficient parameters, use the following format:

```
.PARAM CU=2.0
E1 2 3 14 1 MAX=+5 MIN=-5 CU
```

Voltage Summer

An ideal voltage summer specifies the source voltage, as a function of three controlling voltage(s):

- V(13,0)
- V(15,0)
- V(17,0)

To describe a voltage source, the voltage summer uses this value:

$$V(13,0) + V(15,0) + V(17,0)$$

This example represents an ideal voltage summer. It initializes the three controlling voltages for a DC operating point analysis, to 1.5, 2.0, and 17.25 V.

```
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.25
```

Polynomial Function

A voltage-controlled source can also output a non-linear function, using a one-dimensional polynomial. This example does not specify the POLY parameter, so HSPICE or HSPICE RF assumes it is a one-dimensional polynomial—that is, a function of one controlling voltage. The equation corresponds to the element syntax. Behavioral equations replace this older method.

```
V (3,4)=10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)^2"
E2 3 4 POLY 21 17 10.5 2.1 1.75
E2 3 4 VOLT="10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)^2"
E2 3 4 POLY 21 17 10.5 2.1 1.75
```

Zero-Delay Inverter Gate

Use a piecewise linear transfer function to build a simple inverter, with no delay.

```
Einv out 0 PWL(1) in 0 .7v,5v 1v,0v
```

Ideal Transformer

If the turn ratio is 10 to 1, the voltage relationship is V(out)=V(in)/10.

```
Etrans out 0 TRANSFORMER in 0 10
```

You can also substitute LEVEL=2 in place of TRANSFORMER:

```
Etrans out 0 in 0 level=2 10
```

Voltage-Controlled Oscillator (VCO)

The VOL keyword defines a single-ended input, which controls output of a VCO.

Example 1

In this example, the voltage at the control node controls the frequency of the sinusoidal output voltage at the out node. v0 is the DC offset voltage, and gain is the amplitude. The output is a sinusoidal voltage, whose frequency is specified in *freq · control*.

```
Evco out 0 VOL='v0+gain*SIN(6.28 freq*v(control)*TIME)'
```

Note:

> This equation is valid only for a steady-state VCO (fixed voltage). If you sweep the control voltage, this equation does not apply.

Example 2

In this example, a Verilog-A module is used to control VCO output by tracking phase to ensure continuity.

```
`include "disciplines.vams"

module vco(vin, vout);
 inout vin, vout;
 electrical vin, vout;
 parameter real amp = 1.0;
 parameter real offset = 1.0;
 parameter real center_freq = 1G;
 parameter real vco_gain = 1G;
 real phase;

 analog begin
  phase = idt(center_freq + vco_gain*V(vin), 0.0);
  V(vout) <+ offset+amp*sin(6.2831853*phase);
 end
endmodule
```

Example 3

This example is a controlled-source equivalent of the Verilog-A module shown in the previous example. Like the previous example, it establishes a continuous phase and therefore, a continuous output voltage.

```
.subckt vco in out amp=1 offset=1 center_freq=1 vco_gain=1
.ic v(phase)=0
cphase phase 0 1e-10
g1 0 phase cur='1e-10*(center_freq+vco_gain*v(in))'
eout out 0 vol='offset+amp*sin(6.2831853*v(phase))'
.ends
```

Example 4

In this example, controlled-sources are used to control VCO output.

```
.param pi=3.1415926
.param twopi='2*pi'
.subckt vco in inb out outb f0=100k kf=50k out_off=0.0 out_amp=1.0
gs 0 s poly(2) c 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
gc c 0 poly(2) s 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
cs s 0 1e-9 ic=0
cc c 0 1e-9 ic=1
eout out 0 vol='out_off+(out_amp*v(s))'
eoutb outb 0 vol='out_off+(out_amp*v(c))'
.ic v(c)=1 v(s)=0
.ends
```

Using the E Element for AC Analysis

The following equation describes an E element:

```
E1 ee 0 vol=f(v(1), v(2))
```

In an AC analysis, voltage is computed as follows:

v(ee)=A * delta_v1 + B * delta_v2

Where,

- A is the derivative of f(v(1), v(2)) to v(1) at the operating point
- B is the derivative of f(v(1), v(2)) to v(2) at the operating point
- delta_v1 is the AC voltage variation of v(1)
- delta_v2 is the AC voltage variation of v(2)

Example

This example is based on demonstration netlist eelm.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
*****************************************************
****** E element for AC analysis
.option post
.op
*CASE1-Mixed and zero time unit has zero value(tran)
v_n1 n1 gnd dc=6.0 pwl 0.0 6.0 1.0n 6.0 ac 5.0
v_n2 n2 gnd dc=4.0 pwl 0.0 4.0 1.0n 6.0 ac 2.0
e1 n3 gnd vol='v(n1)+v(n2)'
e2 n4 gnd vol='v(n1)*v(n2)'
r1 n1 gnd 1
r2 n2 gnd 1
r3 n3 gnd 1
r4 n4 gnd 1
.tran 10p 3n
.ac dec 1 1 100meg
.print ac v(n?)
.end
*****************************************************
```

The AC voltage of node n3 is:

```
v(n3)=1.0 *v(n1)(ac)  +  1.0 * v(n2)(ac)
   =  1.0 * 5.0    +  1.0  *  2.0
   =  7.0  (v)
```

The AC voltage of node n4 is:

```
v(n4)=v(n2)(op) * v(n1)(ac) + v(n1)(op) * v(n2)(ac)
   =  4.0   *   5.0  +  6.0   *  2.0
   =  32.0 (v)
```

Current-Dependent Current Sources — F Elements

This section explains the F element syntax and parameters.

Current-Controlled Current Source (CCCS) Syntax

Linear

```
Fxxx n+ n- <CCCS> vn1 gain <MAX=val> <MIN=val> <SCALE=val>
+ <TC1=val> <TC2=val> <M=val> <ABS=1> <IC=val>
```

Polynomial (POLY)

```
Fxxx n+ n- <CCCS> POLY(ndim) vn1 <... vnndim> <MAX=val>
+ <MIN=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ <ABS=1> p0 <p1…> <IC=val>
```

In this syntax, *dim* (*dimensions*) ≤ 3.

Piecewise Linear (PWL)

```
Fxxx n+ n- <CCCS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> <M=val> x1,y1  ... x100,y100 <IC=val>
```

Multi-Input Gates

```
Fxxx n+ n- <CCCS> gatetype(k) vn1, ... vnk <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <ABS=1>
+ x1,y1  ...   x100,y100 <IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

Note:

G elements with algebraics make F elements obsolete. You can still use F elements for backward-compatibility with existing designs.

```
Fxxx n+ n- <CCCS> DELAY vn1 TD=val <SCALE=val>
+ <TC1=val><TC2=val> NPDELAY=val
```

F Element Parameters

The F Element parameters are described in the following list.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCCS	Keyword for current-controlled current source. CCCS is a HSPICE reserved keyword; do not use it as a node name.
DELAY	Keyword for the delay element. Same as for a current-controlled current source, but has an associated propagation delay, TD. Adjusts the propagation delay in the macro model (subcircuit) process. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. The maximum is 1/2 of the smallest distance between breakpoints.
Fxxx	Element name of the current-controlled current source. Must begin with F, followed by up to 1023 alphanumeric characters.
gain	Current gain.
gatetype(k)	AND, NAND, OR, or NOR. *k* is the number of inputs for the gate. *x* and *y* are the piecewise linear variation of the output, as a function of input. In multi-input gates, only one input determines the output state. Do not use the above keywords as node names.
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.
M	Number of replications of the element, in parallel.
MAX	Maximum output current. Default=undefined; sets no maximum.

Parameter	Description
MIN	Minimum output current. Default=undefined; sets no minimum.
n+/-	Connecting nodes for a positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$ The `.TRAN` statement specifies the tstep and tstop values.
P0, P1 …	The polynomial coefficients. If you specify one coefficient, HSPICE or HSPICE RF assumes it is P1 (P0=0.0), and the source element is linear. If you specify more than one polynomial coefficient, then the source is non-linear, and HSPICE or HSPICE RF assumes that the polynomials are P0, P1, P2 … See Polynomial Functions on page 158.
POLY	Keyword for the polynomial function. If you do not specify POLY(*ndim*), HSPICE assumes that this is a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
vn1 …	Names of voltage sources, through which the controlling current flows. Specify one name for each dimension.
x1,...	Controlling current, through the *vn1* source. Specify the *x* values in increasing order.

Parameter	Description
y1,...	Corresponding output current values of *x*.

F Element Examples

Example 1

This example describes a current-controlled current source, connected between nodes 13 and 5. The current, which controls the value of the controlled source, flows through the voltage source named `VSENS`.

```
F1 13 5 VSENS MAX=+3 MIN=-3 5
```

Note:

> To use a current-controlled current source, you can place a dummy independent voltage source into the path of the controlling current.

The defining equation is:

$$I(F1) = 5 \cdot I(VSENS)$$

- Current gain is 5.
- Maximum current flow through F1 is 3 A.
- Minimum current flow is -3 A.

If `I(VSENS)=2 A`, then this examples sets `I(F1)` to 3 amps, not 10 amps (as the equation suggests). You can define a parameter for the polynomial coefficient(s):

```
.PARAM VU=5
F1 13 5 VSENS MAX=+3 MIN=-3 VU
```

Example 2

This example is a current-controlled current source, with the value:

I(F2)=1e-3 + 1.3e-3 · I(VCC)

Current flows from the positive node, through the source, to the negative node. The positive controlling-current flows from the positive node, through the source, to the negative node of `vnam` (linear), or to the negative node of each voltage source (nonlinear).

```
F2 12 10 POLY VCC 1MA 1.3M
```

Example 3

This example is a delayed, current-controlled current source.

```
Fd 1 0 DELAY vin TD=7ns SCALE=5
```

Example 4

This example is a piecewise-linear, current-controlled current source.

```
Filim 0 out PWL(1) vsrc -1a,-1a 1a,1a
```

Voltage-Dependent Current Sources — G Elements

This section explains G element syntax statements, and their parameters.

- `LEVEL=0` is a Voltage-Controlled Current Source (VCCS).
- `LEVEL=1` is a Voltage-Controlled Resistor (VCR).
- `LEVEL=2` is a Voltage-Controlled Capacitor (VCCAP), Negative Piece-Wise Linear (NPWL).
- `LEVEL=3` is a VCCAP, Positive Piece-Wise Linear (PPWL).

See also "Using G and E Elements" in the *HSPICE Applications Manual*.

Voltage-Controlled Current Source (VCCS)

Linear

```
Gxxx n+ n- <VCCS> in+ in- transconductance <MAX=val>
+ <MIN=val> <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ <ABS=1> <IC=val>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Polynomial (POLY)

```
Gxxx n+ n- <VCCS> POLY(NDIM) in1+ in1- ... <inndim+ inndim->
+ <MAX=val> <MIN=val> <SCALE=val> <M=val> <TC1=val>
+ <TC2=val> <ABS=1> P0<P1…> <IC=vals>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Piecewise Linear (PWL)

```
Gxxx n+ n- <VCCS> PWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
Gxxx n+ n- <VCCS> NPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val><TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
Gxxx n+ n- <VCCS> PPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Multi-Input Gate

```
Gxxx n+ n- <VCCS> gatetype(k) in1+ in1- ...
+ ink+ ink- <DELTA=val> <TC1=val> <TC2=val> <SCALE=val>
+ <M=val> x1,y1 ... x100,y100<IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates. For a description of the G Element parameters, see G Element Parameters on page 189.

Delay Element

```
Gxxx n+ n- <VCCS> DELAY in+ in- TD=val <SCALE=val>
+ <TC1=val> <TC2=val> NPDELAY=val
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Laplace Transform

For details, see Laplace Transform on page 167.

Pole-Zero Function

For details, see Pole-Zero Function on page 168.

Frequency Response Table

For details, see Frequency Response Table on page 169.

Foster Pole-Residue Form

For details, see Foster Pole-Residue Form on page 170.

Behavioral Current Source (Noise Model)

Expression form

```
gxxx node1 node2 noise='noise_expression'
```

The *xxx* parameter can be set with a value up to 1024 characters. The *node1* and *node2* are the positive and negative nodes that connect to the noise source. The noise expression can contain the bias, frequency, or other parameters, and involve node voltages and currents through voltage sources.

For a description of the G Element parameters, see G Element Parameters on page 189.

This syntax creates a simple two-terminal current noise source, whose value is described in A^2/Hz. The output noise generated from this noise source is:

noise_expression*H

H is the transfer function from the terminal pair (node1,node2) to the circuit output, where the output noise is measured.

You can also implement a behavioral noise source with an E Element. As noise elements, they are two-terminal elements that represent a noise source connected between two specified nodes.

```
gname node1 node2 node3 node4 noise='expression'
```

This syntax produces a noise source correlation between the terminal pairs (`node1 node2`) and (`node3 node4`). The resulting output noise is computed as:

noise_expression*sqrt(H1*H2*)

- H1 is the transfer function from (node1,node2) to the output.
- H2 is the transfer function from (node3,node4) to the output.

The noise expression can involve node voltages and currents through voltage sources.

Data form

```
Gxxx node1 node2 noise data=dataname
.DATA dataname
+ pname1 pname2
```

```
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is A^2/Hz.

For a description of the G Element parameters, see G Element Parameters on page 189.

Example

The following netlist shows a 1000 ohm resistor (`g1`) using a G element. The `g1noise` element, placed in parallel with the `g1` resistor, delivers the thermal noise expected from a resistor. The `r1` resistor is included for comparison: The noise due to `r1` should be the same as the noise due to `g1noise`.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2
+ noise='4*1.3806266e-23*(TEMPER+273.15)*0.001'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Voltage-Controlled Resistor (VCR)

Linear

```
Gxxx n+ n- VCR in+ in- transfactor <MAX=val> <MIN=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val> <IC=val>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Polynomial (POLY)

```
Gxxx n+ n- VCR POLY(NDIM) in1+ in1- ...
+ <inndim+ inndim-> <MAX=val> <MIN=val><SCALE=val>
+ <M=val> <TC1=val> <TC2=val>    P0 <P1…> <IC=vals>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Piecewise Linear (PWL)

```
Gxxx n+ n- VCR PWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR NPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR PPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Multi-Input Gates

```
Gxxx n+ n- VCR gatetype(k) in1+ in1- ... ink+ ink-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ x1,y1 ... x100,y100 <IC=val>
```

For a description of the G Element parameters, see G Element Parameters on page 189.

Voltage-Controlled Capacitor (VCCAP)

```
Gxxx n+ n- VCCAP PWL(1) in+ in-   <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

HSPICE or HSPICE RF uses either `LEVEL=2` (`NPWL`) or `LEVEL=3` (`PPWL`), based on the relationship of the (n+, n-) and (in+, in-) nodes. For a description of the G Element parameters, see G Element Parameters on page 189.

Use the `NPWL` and `PPWL` functions to interchange the n+ and n- nodes, but use the same transfer function. The following summarizes this action:

NPWL Function

For the *in*- node connected to *n*+:

- If *v*(*n*+,*n*-) < 0, then the controlling voltage is *v*(*in*+,*in*-).
- Otherwise, the controlling voltage is v(*in*+,*n*-).

For the *in*- node connected to *n*-(+:)Tj/F4 1 Tf

Parameter	Description
DELAY	Keyword for the delay element. Same as in the voltage-controlled current source, but has an associated propagation delay, TD. Adjusts propagation delay in macro (subcircuit) modeling. DELAY is a keyword; do not use it as a node name.
DELTA	Controls curvature of piecewise linear corners. Defaults to 1/4 of the smallest distance between breakpoints. Maximum is 1/2 of the smallest distance between breakpoints.
Gxxx	Name of the voltage-controlled element. Must begin with G, followed by up to 1023 alphanumeric characters.
gatetype(k)	AND, NAND, OR, or NOR. The *k* parameter is the number of inputs of the gate. *x* and *y* represent the piecewise linear variation of the output, as a function of the input. In multi-input gates, only one input determines the state of the output.
IC	Initial condition. Initial estimate of the value(s) of controlling voltage(s). If you do not specify IC, the default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
M	Number of replications of the elements in parallel.
MAX	Maximum value of the current or resistance. The default is undefined, and sets no maximum value.
MIN	Minimum value of the current or resistance. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of the controlled element.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.

Parameter	Description
NPDELAY	Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$. The .TRAN statement specifies the tstep and tstop values.
NPWL	Models symmetrical bidirectional switch/transfer gate, NMOS.
P0, P1 …	The polynomial coefficients. ■ If you specify one coefficient, HSPICE or HSPICE RF assumes that it is P1 (P0=0.0), and the element is linear. ■ If you specify more than one polynomial coefficient, the element is non-linear, and the coefficients are P0, P1, P2 ... (see Polynomial Functions on page 158).
POLY	Keyword for the polynomial dimension function. If you do not specify *POLY*(*ndim*), HSPICE assumes that it is a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
PPWL	Models symmetrical bidirectional switch/transfer gate, PMOS.
SCALE	Multiplier for the element value.
SMOOTH	For piecewise-linear, dependent-source elements, SMOOTH selects the curve-smoothing method. A curve-smoothing method simulates exact data points that you provide. You can use this method to make HSPICE or HSPICE RF simulate specific data points, which correspond to either measured data or data sheets. Choices for SMOOTH are 1 or 2: ■ Selects the smoothing method used in Hspice versions before release H93A. Use this method to maintain compatibility with simulations that you ran, using releases older than H93A. ■ Selects the smoothing method, which uses data points that you provide. This is the default for Hspice versions starting with release H93A.

Parameter	Description
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$.
TD	Keyword for the time (propagation) delay.
transconductance	Voltage-to-current conversion factor.
transfactor	Voltage-to-resistance conversion factor.
VCCAP	Keyword for voltage-controlled capacitance element. VCCAP is a reserved HSPICE keyword; do not use it as a node name.
VCCS	Keyword for the voltage-controlled current source. VCCS is a reserved HSPICE keyword; do not use it as a node name.
VCR	Keyword for the voltage controlled resistor element. VCR is a reserved HSPICE keyword; do not use it as a node name.
x1,...	Controlling voltage, across the *in*+ and *in*- nodes. Specify the *x* values in increasing order.
y1,...	Corresponding element values of *x*.

G Element Examples

Switch

A voltage-controlled resistor represents a basic switch characteristic. The resistance between nodes 2 and 0 varies linearly, from 10 meg to 1 m ohms, when voltage across nodes 1 and 0 varies between 0 and 1 volt. The resistance remains at 10 meg when below the lower voltage limit, and at 1 m ohms when above the upper voltage limit.

```
Gswitch 2 0 VCR PWL(1) 1 0 0v,10meg 1v,1m
```

Switch-Level MOSFET

To model a switch level n-channel MOSFET, use the N-piecewise linear resistance switch. The resistance value does not change when you switch the *d* and *s* node positions.

```
Gnmos d s VCR NPWL(1) g s LEVEL=1 0.4v,150g
+ 1v,10meg 2v,50k 3v,4k 5v,2k
```

Voltage-Controlled Capacitor

The capacitance value across the (*out,0*) nodes varies linearly (from 1 p to 5 p), when voltage across the ctrl,0 nodes varies between 2 v and 2.5 v. The capacitance value remains constant at 1 picofarad when below the lower voltage limit, and at 5 picofarads when above the upper voltage limit.

```
Gcap out 0 VCCAP PWL(1) ctrl 0 2v,1p 2.5v,5p
```

Zero-Delay Gate

To implement a two-input AND gate, use an expression and a piecewise linear table.

- The inputs are voltages at the a and b nodes.
- The output is the current flow from the out to 0 node.
- HSPICE or HSPICE RF multiplies the current by the `SCALE` value—which in this example, is the inverse of the load resistance, connected across the out,0 nodes.

  ```
  Gand out 0 AND(2) a 0 b 0 SCALE='1/rload' 0v,0a 1v,.5a
  + 4v,4.5a 5v,5a
  ```

Delay Element

A delay is a low-pass filter type delay, similar to that of an opamp. In contrast, a transmission line has an infinite frequency response. A glitch input to a G delay attenuates in a way that is similar to a buffer circuit. In this example, the output of the delay element is the current flow, from the *out* node to the *1* node, with a value equal to the voltage across the (*in*, *0*) nodes, multiplied by the `SCALE` value, and delayed by the `TD` value.

```
Gdel out 0 DELAY in 0 TD=5ns SCALE=2 NPDELAY=25
```

Diode Equation

To model forward-bias diode characteristics, from node 5 to ground, use a runtime expression. The saturation current is 1e-14 amp, and the thermal voltage is 0.025 v.

```
Gdio 5 0 CUR='1e-14*(EXP(V(5)/0.025)-1.0)'
```

Diode Breakdown

You can model the diode breakdown region to a forward region. When voltage across a diode is above or below the piecewise linear limit values (-2.2v, 2v), the diode current remains at the corresponding limit values (-1a, 1.2a).

```
Gdiode 1 0 PWL(1) 1 0 -2.2v,-1a -2v,-1pa .3v,.15pa
+.6v,10ua 1v,1a 2v,1.2a
```

Triodes

Both of the following voltage-controlled current sources implement a basic triode.

- The first example uses the poly(2) operator, to multiply the anode and grid voltages together, and to scale by .02.
- The second example uses the explicit behavioral algebraic description.

```
gt i_anode cathode poly(2) anode,cathode
+ grid,cathode 0 0 0 0 .02

gt i_anode cathode
+ cur='20m*v(anode,cathode)*v(grid,cathode)'
```

Behavioral Noise Model

The following netlist shows a 1000 Ohm resistor (g1) implemented using a G element. The g1noise element, placed in parallel with the g1 resistor, delivers the thermal noise expected from a resistor. The r1 resistor is included for comparison: the noise due to r1 should be the same as the noise due to g1noise.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2 noise='sqrt(4*1.3806266e-23*(TEMPER+273.15)*0.001)'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Current-Dependent Voltage Sources — H Elements

This section explains H element syntax statements, and defines their parameters.

Current-Controlled Voltage Source (CCVS)

Linear

```
Hxxx n+ n- <CCVS> vn1 transresistance <MAX=val> <MIN=val>
+ <SCALE=val> <TC1=val><TC2=val> <ABS=1> <IC=val>
```

Polynomial (POLY)

```
Hxxx n+ n- <CCVS> POLY(NDIM) vn1 <... vnndim>
+ <MAX=val><MIN=val> <TC1=val> <TC2=val> <SCALE=val>
+ <ABS=1> P0  <P1…> <IC=val>
```

Piecewise Linear (PWL)

```
Hxxx n+ n- <CCVS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1  ...   x100,y100 <IC=val>
```

Multi-Input Gate

```
Hxxx n+ n- gatetype(k) vn1, ...vnk <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1 ...   x100,y100 <IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

Note:

> E elements with algebraics make CCVS elements obsolete. You can still use CCVS elements for backward-compatibility with existing designs.

```
Hxxx n+ n- <CCVS> DELAY vn1 TD=val <SCALE=val> <TC1=val>
+ <TC2=val> <NPDELAY=val>
```

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCVS	Keyword for the current-controlled voltage source. CCVS is a HSPICE reserved keyword; do not use it as a node name.
DELAY	Keyword for the delay element. Same as for a current-controlled voltage source, but has an associated propagation delay, TD. Use this element to adjust the propagation delay in the macro (subcircuit) model process. DELAY is a HSPICE reserved keyword; do not use it as a node name.
DELTA	Controls curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. Maximum is 1/2 of the smallest distance between breakpoints.
gatetype(k)	Can be AND, NAND, OR, or NOR. The *k* value is the number of inputs of the gate. The *x* and *y* terms are the piecewise linear variation of the output, as a function of the input. In multi-input gates, one input determines the output state.
Hxxx	Element name of current-controlled voltage source. Must start with H, followed by up to 1023 alphanumeric characters.
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.
MAX	Maximum voltage. Default is undefined; sets no maximum.
MIN	Minimum voltage. Default is undefined; sets no minimum.
n+/-	Connecting nodes for positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.

Parameter	Description
NPDELAY	Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is: $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$. The .TRAN statement specifies the tstep and tstop values.
P0, P1 . . .	Polynomial coefficients. ■ If you specify one polynomial coefficient, the source is linear, and HSPICE or HSPICE RF assumes that the polynomial is P1 (P0=0.0). ■ If you specify more than one polynomial coefficient, the source is non-linear. HSPICE assumes the polynomials are P0, P1, P2 … See Polynomial Functions on page 158.
POLY	Keyword for polynomial dimension function. If you do not specify *POLY*(*ndim*), HSPICE assumes a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for a piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
transresistance	Current-to-voltage conversion factor.
vn1 …	Names of voltage sources, through which controlling current flows. You must specify one name for each dimension.
x1,...	Controlling current, through the *vn1* source. Specify the *x* values in increasing order.
y1,...	Corresponding output voltage values of *x*.

Example 1

```
HX 20 10 VCUR MAX=+10 MIN=-10 1000
```

The example above selects a linear current-controlled voltage source. The controlling current flows through the dependent voltage source, called VCUR.

Example 2

The defining equation of the CCVS is:

$$HX = 1000 \cdot I(VCUR)$$

The defining equation specifies that the voltage output of HX is 1000 times the value of the current flowing through VCUR.

- If the equation produces a value of HX greater than +10 V, then the `MAX` parameter sets HX to 10 V.
- If the equation produces a value of HX less than -10 V, then the `MIN` parameter sets HX to -10 V.

VCUR is the name of the independent voltage source, through which the controlling current flows. If the controlling current does not flow through an independent voltage source, you must insert a dummy independent voltage source.

Example 3

```
.PARAM CT=1000
HX 20 10 VCUR MAX=+10 MIN=-10 CT
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5, 1.3
```

The example above describes a dependent voltage source, with the value:

$$V = I(VIN1) \cdot I(VIN2)$$

This two-dimensional polynomial equation specifies:

- FA1=VIN1
- FA2=VIN2
- P0=0
- P1=0
- P2=0
- P3=0
- P4=1

The initial controlling current is .5 mA through VIN1, and 1.3 mA for VIN2.

Positive controlling current flows from the positive node, through the source, to the negative node of vnam (linear). The (non-linear) polynomial specifies the source voltage, as a function of the controlling current(s).

Digital and Mixed Mode Stimuli

HSPICE input netlists support two types of digital stimuli:

- U element digital input files (HSPICE only).
- Vector input files (HSPICE or HSPICE RF).

This section describes both types.

U Element Digital Input Elements and Models

This section describes the input file format for a U Element. For a description of the U Element, see the "Modeling Ideal and Lumped Transmission Lines" chapter in the *HSPICE Signal Integrity Guide*.

In HSPICE (but not in HSPICE RF), the U Element can reference digital input and digital output models for mixed-mode simulation. If you run HSPICE in standalone mode, the state information originates from a digital file. Digital outputs are handled in a similar fashion. In digital input file mode, the input file is named <design>.d2a, and the output file is named <design>.a2d.

A2D and D2A functions accept the terminal "\" backslash character as a line-continuation character, to allow more than 255 characters in a line. Use line continuation if the first line of a digital file, which contains the signal name list, is longer than the maximum line length that your text editor accepts.

Do not put a blank first line in a digital D2A file. If the first line of a digital file is blank, HSPICE issues an error message.

Example

The following example demonstrates how to use the "\" line continuation character, to format an input file for text editing. The example file contains a signal list for a 64-bit bus.

```
...
a00 a01 a02 a03 a04 a05 a06 a07 \
a08 a09 a10 a11 a12 a13 a14 a15 \
... * Continuation of signal names
```

```
a56 a57 a58 a59 a60 a61 a62 a63 End of signal names
... Remainder of file
```

General Form

```
Uxxx interface nlo nhi mname SIGNAME=sname IS=val
```

Parameter	Description
Uxxx	Digital input element name. Must begin with U, followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit, to which the digital input attaches.
nlo	Node connected to the low-level reference.
nhi	Node connected to the high-level reference.
mname	Digital input model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. Can be a string of up to eight alphanumeric characters.
IS	Initial state of the input element. Must be a state that the model defines.

Model Syntax

```
.MODEL mname U LEVEL=5 <parameters...>
```

Digital input (not supported in HSPICE RF).

Digital-to-Analog Input Model Parameters

Table 12 Digital-to-Analog Parameters

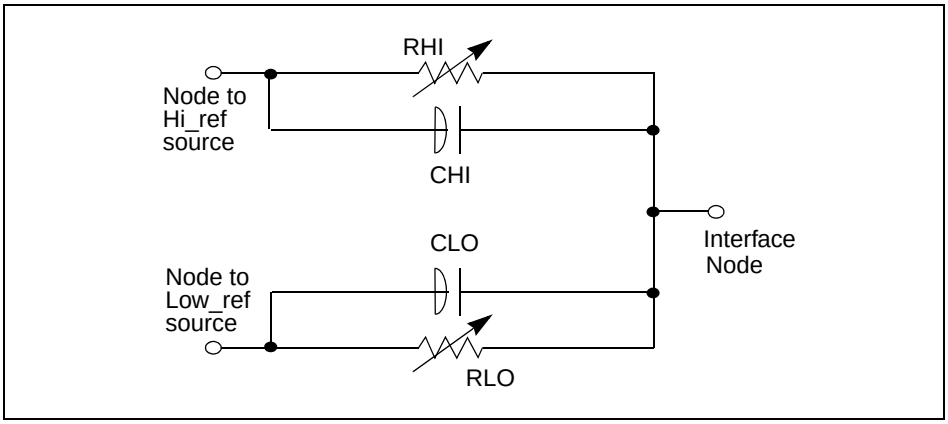
Names (Alias)	Units	Default	Description
CLO	farad	0	Capacitance, to low-level node.
CHI	farad	0	Capacitance, to high-level node.
S0NAME			State 0 character abbreviation. A string of up to four alphanumerical characters.

Table 12 Digital-to-Analog Parameters (Continued)

Names (Alias)	Units	Default	Description
S0TSW	sec		State 0 switching time.
S0RLO	ohm		State 0 resistance, to low-level node.
S0RHI	ohm		State 0 resistance, to high-level node.
S1NAME			State 1 character abbreviation. A string of up to four alphanumerical characters.
S1TSW	sec		State 1 switching time.
S1RLO	ohm		State 1 resistance, to low-level node.
S1RHI	ohm		State 1 resistance, to high-level node.
S19NAME			State 19 character abbreviation. A string of up to four alphanumerical characters.
S19TSW	sec		State 19 switching time.
S19RLO	ohm		State 19 resistance, to low-level node.
S19RHI	ohm		State 19 resistance, to high-level node.
TIMESTEP	sec		Step size for digital input files only.

To define up to 20 different states in the model definition, use the S*n*NAME, S*n*TSW, S*n*RLO and S*n*RHI parameters, where *n* ranges from 0 to 19. Figure 24 is the circuit representation of the element.

Figure 24 Digital-to-Analog Converter Element



Example

The following example shows how to use the U element and model, as a digital input for a HSPICE netlist (you cannot use the U element in a HSPICE RF netlist).

This example is based on demonstration netlist uelm.sp, which is available in directory $<installdir>/demo/hspice/sources:

```
* EXAMPLE OF U-ELEMENT DIGITAL INPUT
.option post
UC carry-in VLD2A VHD2A D2A SIGNAME=1 IS=0
VLO VLD2A GND DC 0
VHI VHD2A GND DC 1
.MODEL D2A U LEVEL=5 TIMESTEP=1NS,
+ S0NAME=0 S0TSW=1NS S0RLO = 15, S0RHI = 10K,
+ S2NAME=x S2TSW=3NS S2RLO = 1K, S2RHI = 1K
+ S3NAME=z S3TSW=5NS S3RLO = 1MEG,S3RHI = 1MEG
+ S4NAME=1 S4TSW=1NS S4RLO = 10K, S4RHI = 60
.PRINT V(carry-in)
.TRAN 1N 100N
.END
```

The associated digital input file is:

```
1
00 1:1
09 z:1
10 0:1
11 z:1
20 1:1
30 0:1
39 x:1
40 1:1
41 x:1
```

```
50 0:1
60 1:1
70 0:1
80 1:1
```

U Element Digital Outputs

Digital output (not supported in HSPICE RF).

Syntax

```
Uxxx interface reference mname SIGNAME=sname
```

Parameter	Description
Uxxx	Digital output element name. Must begin with U, followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit, at which HSPICE measures the digital output.
reference	Node to use as a reference for the output.
mname	Digital output model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. A string of up to eight alphanumeric characters.

Model Syntax

```
.MODEL mname U LEVEL=4 <parameters...>
```

Analog-to-Digital Output Model Parameters

Table 13 Analog-to-Digital Parameters

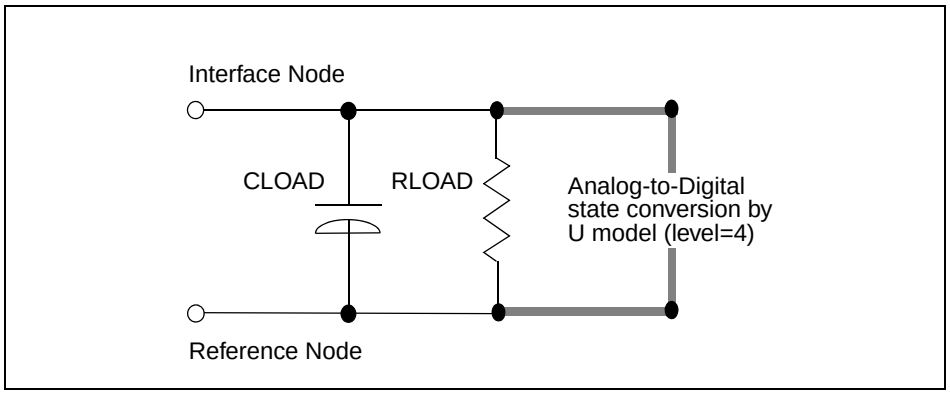
Name (Alias)	Units	Default	Description
RLOAD	ohm	1/gmin	Output resistance.
CLOAD	farad	0	Output capacitance.

Table 13 Analog-to-Digital Parameters (Continued)

Name (Alias)	Units	Default	Description
S0NAME			State 0 character abbreviation. A string of up to four alphanumerical characters.
S0VLO	volt		State 0 low-level voltage.
S0VHI	volt		State 0 high-level voltage.
S1NAME			State 1 character abbreviation. A string of up to four alphanumerical characters.
S1VLO	volt		State 1 low-level voltage.
S1VHI	volt		State 1 high-level voltage.
S19NAME			State 19 character abbreviation. A string of up to four alphanumerical characters.
S19VLO	volt		State 19 low-level voltage.
S19VHI	volt		State 19 high-level voltage.
TIMESTEP	sec	1E-9	Step size for digital input file.
TIMESCALE			Scale factor for time.

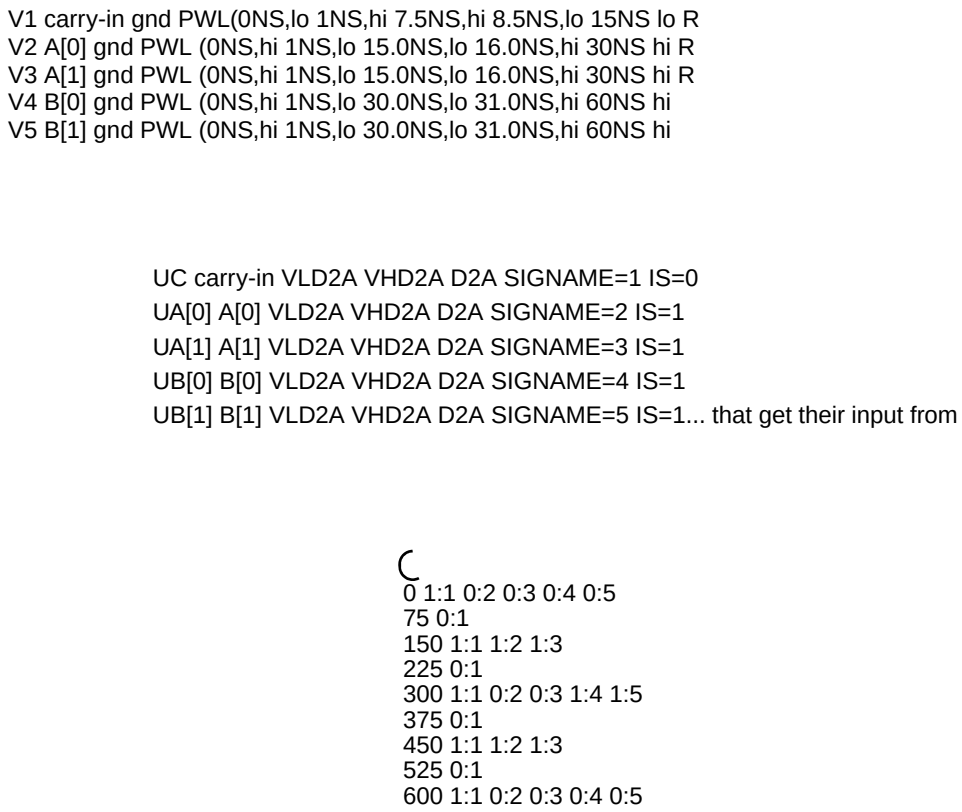
To define up to 20 different states in the model definition, use the S*n*NAME, S*n*VLO and S*n*VHI parameters, where *n* ranges from 0 to 19. Figure 25 shows the circuit representation of the element.

Figure 25 Analog-to-Digital Converter Element



Replacing Sources With Digital Inputs

Figure 26 Digital File Signal Correspondence



Example

The following is an example of replacing sources with digital inputs. This example is based on demonstration netlist digin.sp, which is available in directory $<installdir>/demo/hspice/cchar:

```
* EXAMPLE OF U-ELEMENT DIGITAL OUTPUT
.OPTION POST
VOUT carry_out GND PWL 0N 0V 10N 0V 11N 5V 19N 5V 20N 0V
+ 30N 0V 31N 5V 39N 5V 40N 0V
VREF REF GND DC 0.0V
UCO carry_out REF A2D SIGNAME=12
R1 REF 0 1k

* DEFAULT DIGITAL OUTPUT MODEL (no "X" value)
```

```
.MODEL A2D U LEVEL=4 TIMESTEP=0.1NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI= 2.7
+ S4NAME=1 S4VLO= 1.4 S4VHI=9.0
+ CLOAD=0.05pf
.TRAN 1N 500N
.END
```

The digital output file should look something like this:

```
12
0       0:1
105     1:1
197     0:1
305     1:1
397     0:1
```

- *12* represents the signal name
- The first column is the time, in units of 0.1 nanoseconds.
- The second column has the signal *value:name* pairs.
- This file uses more columns to represent subsequent outputs.

Also, this example based on demonstration netlist tdgt1.sp, which is available in directory $<installdir>/demo/hspice/cchar:

```
*file: mos2bit.sp - adder - 2 bit all-nand-gate binary adder
*
.options post nomod fast scale=1u gmindc=100n
+
.param lmin=1.25 hi=2.8v lo=.4v vdd=4.5
.global vdd

.tran .5ns 60ns

.meas prop-delay trig v(carry-in) td=10ns val='vdd*.5' rise=1
+ targ v(c[1]) td=10ns val='vdd*.5' rise=3
*
.meas pulse-width trig v(carry-out_1) val='vdd*.5' rise=1
+ targ v(carry-out_1) val='vdd*.5' fall=1
*
.meas fall-time trig v(c[1]) td=32ns val='vdd*.9' fall=1
+ targ v(c[1]) td=32ns val='vdd*.1' fall=1

vdd vdd gnd dc vdd
x1 a[0] b[0] carry-in c[0] carry-out_1 onebit
x2 a[1] b[1] carry-out_1 c[1] carry-out_2 onebit
```

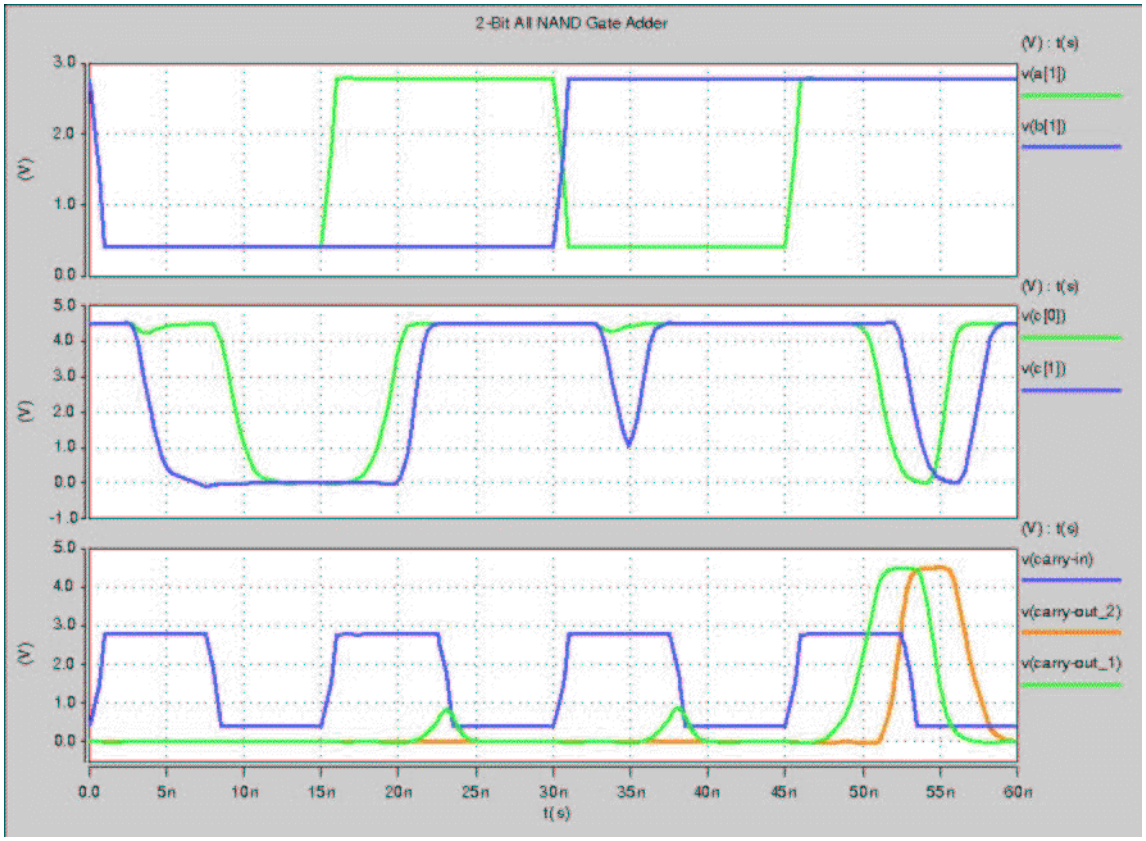
```
*** subcircuit definitions

.subckt nand in1 in2 out wp=10 wn=5
     m1 out in1 vdd vdd p w=wp l=lmin ad=0
     m2 out in2 vdd vdd p w=wp l=lmin ad=0
     m3 out in1 mid gnd n w=wn l=lmin as=0
     m4 mid in2 gnd gnd n w=wn l=lmin ad=0
     cload out gnd 'wp*5.7f'
.ends

.subckt onebit in1 in2 carry-in out carry-out
     x1 in1 in2 #1_nand nand
     x2 in1 #1_nand 8 nand
     x3 in2 #1_nand 9 nand
     x4 8 9 10 nand
     x5 carry-in 10 half1 nand
     x6 carry-in half1 half2 nand
     x7 10 half1 13 nand
     x8 half2 13 out nand
     x9 half1 #1_nand carry-out nand
.ends onebit

* stimulus

     * carryin:
     * ___ ___ ___ ___
     *\___/ \___/ \___/ \___/

     * a register inputs:
     * _______ _______
     *\_______/ \_______/

     * b register inputs:
     * _______________
     *\_______________/

*v1 carry-in gnd pwl(0ns,lo 1ns,hi 7.5ns,hi 8.5ns,lo 15ns lo r
*v2 a[0] gnd pwl (0ns,hi 1ns,lo 15.0ns,lo 16.0ns,hi 30ns hi r
*v3 a[1] gnd pwl (0ns,hi 1ns,lo 15.0ns,lo 16.0ns,hi 30ns hi r
*v4 b[0] gnd pwl (0ns,hi 1ns,lo 30.0ns,lo 31.0ns,hi 60ns hi
*v5 b[1] gnd pwl (0ns,hi 1ns,lo 30.0ns,lo 31.0ns,hi 60ns hi

*1 2 3 4 5
*0 1:1 0:2 0:3 0:4 0:5
*75 0:1
*150 1:1 1:2 1:3
*225 0:1
```

```
*300 1:1 0:2 0:3 1:4 1:5
*375 0:1
*450 1:1 1:2 1:3
*525 0:1
*600 1:1 0:2 0:3 0:4 0:5

uc carry-in vld2a vhd2a d2a signame=1 is=0
ua[0] a[0] vld2a vhd2a d2a signame=2 is=1
ua[1] a[1] vld2a vhd2a d2a signame=3 is=1
ub[0] b[0] vld2a vhd2a d2a signame=4 is=1
ub[1] b[1] vld2a vhd2a d2a signame=5 is=1

uc0 c[0] vrefa2d a2d signame=10
uc1 c[1] vrefa2d a2d signame=11
uco carry-out_2 vrefa2d a2d signame=12
uci carry-in vrefa2d a2d signame=13

* models

.model n nmos level=3 vto=0.7 uo=500 kappa=.25 kp=30u
 + eta=.01 theta=.04 vmax=2e5 nsub=9e16 tox=400 gamma=1.5
 + pb=0.6 js=.1m xj=0.5u ld=0.1u nfs=1e11 nss=2e10
 + rsh=80 cj=.3m mj=0.5 cjsw=.1n mjsw=0.3
 + acm=2 capop=4
*
.model p pmos level=3 vto=-0.8 uo=150 kappa=.25 kp=15u
 + eta=.015 theta=.04 vmax=5e4 nsub=1.8e16 tox=400 gamma=.672
 + pb=0.6 js=.1m xj=0.5u ld=0.15u nfs=1e11 nss=2e10
 + rsh=80 cj=.3m mj=0.5 cjsw=.1n mjsw=0.3
 + acm=2 capop=4
*
* default digital input interface model
.model d2a u level=5 timestep=0.1ns,
+ s0name=0 s0tsw=1ns s0rlo = 15, s0rhi = 10k,
+ s2name=x s2tsw=5ns s2rlo = 1k, s2rhi = 1k
+ s3name=z s3tsw=5ns s3rlo = 1meg,s3rhi = 1meg
+ s4name=1 s4tsw=1ns s4rlo = 10k, s4rhi = 60
vld2a vld2a 0 dc lo
vhd2a vhd2a 0 dc hi

* default digital output model (no "x" value)
.model a2d u level=4 timestep=0.1ns timescale=1
+ s0name=0 s0vlo=-1 s0vhi= 2.7
+ s4name=1 s4vlo= 1.4 s4vhi=6.0
+ cload=0.05pf
vrefa2d vrefa2d 0 dc 0.0v

.end
```

See the plot in Figure 27 on page 210.

In this example, a 2-bit MOS adder uses a digital input file. In the plot, the `a[0]`, `a[1]`, `b[0]`, `b[1]`, and `carry-in` nodes all originate from a digital file input similar to Figure 26 on page 206. HSPICE or HSPICE RF outputs a digital file.

Figure 27 Digital Stimulus File Input



Specifying a Digital Vector File

You can call a digital vector (VEC) file from an HSPICE netlist or from HSPICE RF. A VEC file consists of three parts:

- Vector Pattern Definition section
- Waveform Characteristics section
- Tabular Data section

To incorporate this information into your simulation, include the `.VEC` command in your netlist.

Commands in a Digital Vector File

For descriptions of all commands that you can use in a VEC file, see the "Commands in Digital Vector Files" chapter in the *HSPICE Command Reference*.

Vector Patterns

The *Vector Pattern Definition* section defines the vectors, their names, sizes, signal direction, sequence or order for each vector stimulus, and so on. A RADIX line must occur first and the other lines can appear in any order in this section. All keywords are case-insensitive.

Here is an example Vector Pattern Definition section:

```
; start of Vector Pattern Definition section
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
```

These four lines are required and appear in the first lines of a VEC file:

- `RADIX` defines eight single-bit vectors.
- `VNAME` gives each vector a name.
- `IO` determines which vectors are inputs, outputs, or bidirectional signals. In this example, all eight are input signals.
- `TUNIT` indicates that the time unit for the tabular data to follow is in units of nanoseconds.

For additional information about these keywords, see Defining Tabular Data on page 211.

Defining Tabular Data

Although the *Tabular Data* section generally appears last in a VEC file (after the *Vector Pattern* and *Waveform Characteristics* definitions), this chapter describes it first to introduce the definitions of a vector.

The *Tabular Data* section defines (in tabular format) the values of the signals at specified times. Rows in the Tabular Data section must appear in chronological order, because row placement carries sequential timing information. Its general format is:

```
time1 signal1_value1 signal2_value1 signal3_value1...
time2 signal1_value2 signal2_value2 signal3_value2...
time3 signal1_value3 signal2_value3 signal3_value3...
.
.
```

Where *timex* is the specified time, and *signaln_valuen* is the values of specific signals at specific points in time. The set of values for a particular signal (over all times) is a vector, which appears as a vertical column in the tabular data and vector table. The set of all *signal1_valuen* constitutes one vector.

For example,

```
11.0 1000 1000
20.0 1100 1100
33.0 1010 1001
```

This example shows that:

- At 11.0 time units, the value for the first and fifth vectors is 1.
- At 20.0 time units, the first, second, fifth, and sixth vectors are 1.
- At 33.0 time units, the first, third, fifth, and eighth vectors are 1.

Input Stimuli

HSPICE or HSPICE RF converts each input signal into a PWL (piecewise linear) voltage source, and a series resistance. Table 14 shows the legal states for an input signal. Signal values can have any of these legal states.

Table 14 Legal States for an Input Signal

State	Description
0	Drive to ZERO (gnd). Resistance set to 0.
1	Drive to ONE (vdd). Resistance set to 0.
Z, z	Floating to HIGH IMPEDANCE. A `TRIZ` statement defines resistance value.
X, x	Drive to ZERO (gnd). Resistance set to 0.

Table 14 Legal States for an Input Signal (Continued)

L	Resistive drive to ZERO (gnd). An OUT or OUTZ statement defines resistance value.
H	Resistive drive to ONE (vdd). An OUT or OUTZ statement defines resistance value.
U, u	Drive to ZERO (gnd). Resistance set to 0.

Expected Output

HSPICE or HSPICE RF converts each output signal into a .DOUT statement in the netlist. During simulation, HSPICE or HSPICE RF compares the actual results with the expected output vector(s). If the states are different, an error message appears. The legal states for expected outputs include the values listed in Table 15.

Table 15 Legal States for an Output Signal

State	Description
0	Expect ZERO.
1	Expect ONE.
X, x	Don't care.
U, u	Don't care.
Z, z	Expect HIGH IMPEDANCE (don't care). Simulation evaluates Z, z as "don't care", because HSPICE or HSPICE RF cannot detect a high impedance state.

For example,

```
...
IO OOOO
; start of tabular section data
11.0 1001
20.0 1100
30.0 1000
35.0 xx00
```

Where,

- The first line is a comment line, because of the semicolon character.
- The second line expects the output to be 1 for the first and fourth vectors, while all others are expected to be low.
- At 20 time units, HSPICE or HSPICE RF expects the first and second vectors to be high, and the third and fourth to be low.
- At 30 time units, HSPICE or HSPICE RF expects only the first vector to be high, and all others low.
- At 35 time units, HSPICE or HSPICE RF expects the output of the first two vectors to be “don’t care”; it expects vectors 3 and 4 to be low.

Verilog Value Format

HSPICE or HSPICE RF accepts Verilog-sized format to specify numbers; for example,

```
<size> '<base format> <number>
```

Where:

- <*size*> specifies the number of bits, in decimal format.
- *<base format>* indicates:
  - binary (’b or ’B)
  - octal (’o or ’O)
  - hexadecimal (’h or ’H).
- <*number*> values are combinations of the *0*, *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *A*, *B*, *C*, *D*, *E*, and *F* characters. Depending on what base format you choose, only a subset of these characters might be legal.

  You can also use unknown values (X) and high-impedance (Z) in the <*number*> field. An X or Z sets four bits in the hexadecimal base, three bits in the octal base, or one bit in the binary base.

  If the most significant bit of a number is 0, X, or Z, HSPICE or HSPICE RF automatically extends the number (if necessary), to fill the remaining bits with 0, X, or Z, respectively. If the most significant bit is 1, HSPICE or HSPICE RF uses 0 to extend it.

For example,

```
4’b1111
12’hABx
32’bZ
8’h1
```

This example specifies values for:

- 4-bit signal in binary
- 12-bit signal in hexadecimal
- 32-bit signal in binary
- 8-bit signal in hexadecimal

  Equivalents of these lines in non-Verilog format, are:

```
1111
AB xxxx
ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ
1000 0000
```

Periodic Tabular Data

Tabular data is often periodic, so you do not need to specify the absolute time at every time point. When you specify the `PERIOD` statement, the Tabular Data section omits the absolute times. For more information, see Defining Tabular Data on page 211.

For example, the `PERIOD` statement in the following sets the time interval to 10ns between successive lines in the tabular data. This is a shortcut when you use vectors in regular intervals throughout the entire simulation.

```
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
PERIOD 10
; start of vector data section
1000 1000
1100 1100
1010 1001
```

Waveform Characteristics

The *Waveform Characteristics* section defines various attributes for signals, such as the rise or fall time, the thresholds for logic high or low, and so on. For example,

```
TRISE 0.3 137F 0000
TFALL 0.5 137F 0000
VIH 5.0 137F 0000
VIL 0.0 137F 0000
```

The waveform characteristics are based on a bit-mask. Where:

- The TRISE (signal rise time) setting of 0.3ns applies to the first four vectors, but not to the last four.
- The example does not show how many bits are in each of the first four vectors, although the first vector is at least one bit.
- The fourth vector is four bits, because F is hexadecimal for binary 1111.
- All bits of the fourth vector have a rise time of 0.3ns for the constant you defined in TUNIT. This also applies to TFALL (fall time), VIH (voltage for logic-high inputs), and VIL (voltage for logic-low inputs).

Modifying Waveform Characteristics

The TDELAY, IDELAY, and ODELAY statements define the delay time of the signal, relative to the absolute time of each row in the Tabular Data section.

- TDELAY applies to the input and output delay time of input, output, and bidirectional signals.
- IDELAY applies to the input delay time of bidirectional signals.
- ODELAY applies to the output delay time of bidirectional signals.

The SLOPE statement specifies the rise and fall times for the input signal. To specify the signals to which the *slope* applies, use a mask.

The TFALL statement sets an input fall time for specific vectors.

The TRISE statement sets an input rise time for specific vectors.

The TUNIT statement defines the time unit.

The OUT and OUTZ keywords are equivalent, and specify output resistance for each signal (for which the mask applies); OUT (or OUTZ) applies only to input signals.

The TRIZ statement specifies the output impedance, when the signal (for which the mask applies) is in tristate; TRIZ applies only to the input signals.

The VIH statement specifies the logic-high voltage for each input signal to which the mask applies.

The VIL statement specifies the logic-low voltage for each input signal to which the mask applies.

Similar to the TDELAY statement, the VREF statement specifies the name of the reference voltage for each input vector to which the mask applies. VREF applies only to input signals.

Similar to the TDELAY statement, the VTH statement 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.

The VOH statement specifies the logic-high voltage for each output signal to which the mask applies.

The VOL statement specifies the logic-low voltage for each output signal to which the mask applies.

Using the Context-Based Control Option

The OPTION CBC (Context-Based Control) specifies the direction of bidirectional signals. A bidirectional signal is an input if its value is 0, 1, or Z; conversely, a bidirectional signal is an output if its value is H, L, U, or X.

For example,

```
RADIX 1 1 1
IO I O B
VNAME A Z B
OPTION CBC
10.0 0 X L
20.0 1 1 H
30.0 1 0 Z
```

This example sets up three vectors, named A, Z, and B. Vector A is an input, vector Z is an output, and vector B is a bidirectional signal (defined in the IO statement).

The OPTION CBC line turns on context-based control. The next line sets vector A to a logic-low at 10.0 ns, and vector Z is "do not care." Because the L value is under vector B, HSPICE expects a logic-low output.

At 20 ns, vector A transitions high, and the expected outputs at vectors Z and B are high. Finally, at 30 ns, HSPICE expects vector Z to be low, vector B changes from an output to a high-impedance input, and vector the A signal does not change.

Comment Lines and Line Continuations

Any line in a VEC file that begins with a semicolon (;) is a comment line. Comments can also start at any point along a line. HSPICE or HSPICE RF ignores characters after a semicolon. For example,

```
; This is a comment line
radix 1 1 4 1234 ; This is a radix line
```

As in netlists, any line in a VEC file that starts with a plus sign (+) is a continuation from the previous line.

Parameter Usage

You can use .PARAM statements with some VEC statements when you run HSPICE. These VEC statements fall into the three groups, which are described in the following sections. No other VEC statements but those identified here support .PARAM statements.

First Group

- PERIOD
- TDELAY
- IDELAY
- ODELAY
- SLOPE
- TRISE
- TFALL

For these statements, the TUNIT statement defines the time unit. If you do not include a TUNIT statement, the default time unit value is ns.

Do not specify absolute unit values in a .PARAM statement. For example, if in your netlist:

```
.param myperiod=10ns            $ 'ns' makes this incorrect
```

And in your VEC file:

```
tunit ns
period myperiod
```

What you wanted for the time period is 10ns; however, because you specified absolute units, 1e-8ns is the value used. In this example, the correct form is:

```
.param myperiod=10
```

Second Group

- OUT or OUTZ
- TRIZ

In these statements, the unit is ohms.

- If you do not include an OUT (or OUTZ) statement, the default is 0.
- If you do not include a TRIZ statement, the default is 1000M.

The .PARAM definition for this group follows the HSPICE syntax.

For example, if in your netlist:

```
.param myout=10                 $ means 10 ohm
.param mytriz= 10Meg            $ means 10,000,000 ohm, don't
$ confuse Meg with M, M means 0.001
```

And in your VEC file:

```
out myout
triz mytriz
```

Then, HSPICE returns 10 ohm for OUT and 10,000,000 ohm for TRIZ.

Third Group

- VIH
- VIL
- VOH
- VOL
- VTH

In these statements, the unit is volts.

- If you do not include an VIH statement, the default is 3.3.
- If you do not include a VIL statement, the default is 0.0.
- If you do not include a VOH statement, the default is 2.64.
- If you do not include an VOL statement, the default is 0.66.
- If you do not include an VTH statement, the default is 1.65.

Digital Vector File Example

```
; specifies # of bits associated with each vector
radix 1 2 444
;*****************************************************
; defines name for each vector. For multi-bit vectors,
; innermost [] provide the bit index range, MSB:LSB
vname v1 va[[1:0]] vb[12:1]
;actual signal names: v1, va[0], va[1], vb1, vb2, ... vb12
;*****************************************************
; defines vector as input, output, or bi-directional
io i o bbb
; defines time unit
tunit ns
;*****************************************************
; vb12-vb5 are output when 'v1' is 'high'
enable v1 0 0 FF0
; vb4-vb1 are output when 'v1' is 'low'
enable ~v1 0 0 00F
;*****************************************************
; all signals have a delay of 1 ns
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
tdelay 1.0
; va1 and va0 signals have 1.5ns delays
tdelay 1.5 0 3 000
;*****************************************************
; specify input rise/fall times (if you want different
; rise/fall times, use the trise/tfall statement.)
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
slope 1.2
;*****************************************************
; specify the logic 'high' voltage for input signals
vih 3.3 1 0 000
vih 5.0 0 0 FFF
; to specify logic low, use 'vil'
;*****************************************************
```

```
; va & vb switch from 'lo' to 'hi' at 1.75 volts
vth 1.75 0 1 FFF

;*************************************************
; tabular data section
10.0 1 3 FFF
20.0 0 2 AFF
30.0 1 0 888
```

6

Parameters and Functions

Describes how to use parameters within an HSPICE netlist.

Parameters are similar to the variables used in most programming languages. Parameters hold a value that you assign when you create your circuit design or that the simulation calculates based on circuit solution values. Parameters can store static values for a variety of quantities (resistance, source voltage, rise time, and so on). You can also use them in sweep or statistical analysis.

For descriptions of individual HSPICE commands referenced in this chapter, see the "Netlist Commands" chapter in the *HSPICE Command Reference*.

Using Parameters in Simulation (.PARAM)

Defining Parameters

Parameters in HSPICE are names that you associate with numeric values. (See Assigning Parameters on page 225.) You can use any of the methods described in Table 16 to define parameters.

Table 16 .PARAM Statement Syntax

Parameter	Description
Simple assignment	.PARAM <SimpleParam>=1e-12
Algebraic definition	.PARAM <AlgebraicParam>='SimpleParam*8.2' SimpleParam excludes the output variable. You can also use algebraic parameters in .PRINT and .PROBE statements (HSPICE or HSPICE RF), and in .PLOT, and .GRAPH statements (HSPICE only). For example: .PRINT AlgebraicParam=par('algebraic expression') You can use the same syntax for .PROBE, .PLOT, and .GRAPH statements. See Using Algebraic Expressions on page 228.
User-defined function	.PARAM <*MyFunc*(*x, y*)>='Sqrt((*x**x)+(*y**y))'
Character string definition	.PARAM <*paramname*>=str('*string*')
Subcircuit default	.SUBCKT <SubName> <ParamDefName>=<Value> str('string') .MACRO <SubName> <ParamDefName>=<Value> str('string')
Predefined analysis function	.PARAM <mcVar>=Agauss(1.0,0.1)
.MEASURE statement	.MEASURE <DC | AC | TRAN> result TRIG ... + TARG ... <GOAL=val> <MINVAL=val> + <WEIGHT=val> <MeasType> <MeasParam> (See Specifying User-Defined Analysis (.MEASURE) on page 267.)
.PRINT | .PROBE | .PLOT | .GRAPH	.PRINT | .PROBE | .PLOT | .GRAPH <DC|AC|TRAN> + outParam=Par_Expression

A parameter definition in HSPICE always uses the last value found in the input netlist (subject to local versus global parameter rules). The definitions below assign a value of 3 to the *DupParam* parameter.

```
.PARAM DupParam=1
...
.PARAM DupParam=3
```

HSPICE assigns `3` as the value for all instances of `DupParam`, including instances that are earlier in the input than the `.PARAM DupParam=3` statement.

All parameter values in HSPICE are IEEE double floating point numbers. The parameter resolution order is:

1. Resolve all literal assignments.
2. Resolve all expressions.
3. Resolve all function calls.

Table 17 shows the parameter passing order.

Table 17 Parameter Passing Order

.OPTION PARHIER=GLOBAL	.OPTION PARHIER=LOCAL
Analysis sweep parameters	Analysis sweep parameters
.PARAM statement (library)	.SUBCKT call (instance)
.SUBCKT call (instance)	.SUBCKT definition (symbol)
.SUBCKT definition (symbol)	.PARAM statement (library)

Assigning Parameters

You can assign the following types of values to parameters:

- Constant real number
- Algebraic expression of real values
- Predefined function
- Function that you define
- Circuit value
- Model value

To invoke the algebraic processor, enclose a complex expression in single quotes. A simple expression consists of one parameter name.

The parameter keeps the assigned value, unless:

- A later definition changes its value, or
- An algebraic expression assigns a new value during simulation.

HSPICE does not warn you, if it reassigns a parameter.

Inline Parameter Assignments

To define circuit values, using a direct algebraic evaluation:

```
r1 n1 0 R='1k/sqrt(HERTZ)' $ Resistance for frequency
```

Parameters in Output

To use an algebraic expression as an output variable in a `.PRINT`, `.PLOT`, `.PROBE` `.GRAPH`, or `.MEASURE` statement, use the PAR keyword. (See Chapter 7, Simulation Output, for more information.)

Example

```
.PRINT DC v(3) gain=PAR('v(3)/v(2)') PAR('v(4)/v(2)')
```

User-Defined Function Parameters

You can define a function that is similar to the parameter assignment, but you cannot nest the functions more than two deep.

- An expression can contain parameters that you did not define.
- A function must have at least one argument, and can have up to 20 (and in many cases, more than 20) arguments.
- You can redefine functions.

The format of a function is:

```
funcname1(arg1[,arg2...])=expression1
+ [funcname2(arg1[,arg2...])=expression2] off
```

Parameter	Description
funcname	Specifies the function name. This parameter must be distinct from array names and built-in functions. In subsequently defined functions, all embedded functions must be previously defined.
arg1, arg2	Specifies variables used in the expression.

Parameter	Description
off	Voids all user-defined functions.

Example

```
.PARAM f(a,b)=POW(a,2)+a*b g(d)=SQRT(d)
+ h(e)=e*f(1,2)-g(3)
```

Predefined Analysis Function

HSPICE includes specialized analysis types, such as Optimization and Monte Carlo, that require a way to control the analysis.

Measurement Parameters

.MEASURE statements produce a *measurement* parameter. The rules for measurement parameters are the same as for standard parameters, except that measurement parameters are defined in a .MEASURE statement, not in a .PARAM statement. For a description of the .MEASURE statement, see Specifying User-Defined Analysis (.MEASURE) on page 267.

.PRINT, .PROBE, .PLOT, and .GRAPH Parameters

.PRINT, .PROBE, .PLOT, and .GRAPH statements in HSPICE produce a *print* parameter. The rules for print parameters are the same as the rules for standard parameters, except that you define the parameter directly in a .PRINT, .PROBE, .PLOT, or .GRAPH statement, not in a .PARAM statement. HSPICE RF does not support .PLOT or .GRAPH statements.

For more information about the .PRINT, .PROBE, .PLOT, or .GRAPH statements, see Displaying Simulation Results on page 243.

Multiply Parameter

The most basic subcircuit parameter in HSPICE is the M (multiply) parameter. For a description of this parameter, see M (Multiply) Parameter on page 58.

Using Algebraic Expressions

Note:

> Synopsys HSPICE uses double-precision numbers (15 digits) for expressions, user-defined parameters, and sweep variables. For better precision, use parameters (instead of constants) in algebraic expressions, because constants are only single-precision numbers (7 digits).

In HSPICE, an algebraic expression, with quoted strings, can replace any parameter in the netlist.

In HSPICE, you can then use these expressions as output variables in `.PLOT`, `.PRINT`, and `.GRAPH` statements. Algebraic expressions can expand your options in an input netlist file.

Some uses of algebraic expressions are:

- Parameters:

```
.PARAM x='y+3'
```

- Functions:

```
.PARAM rho(leff,weff)='2+*leff*weff-2u'
```

- Algebra in elements:

```
R1 1 0 r='ABS(v(1)/i(m1))+10'
```

- Algebra in `.MEASURE` statements:

```
.MEAS vmax MAX V(1)
.MEAS imax MAX I(q2)
.MEAS ivmax PARAM='vmax*imax'
```

- Algebra in output statements:

```
.PRINT conductance=PAR('i(m1)/v(22)')
```

The basic syntax for using algebraic expressions for output is:

`PAR('`*`algebraic expression`*`')`

In addition to using quotations, you must define the expression inside the `PAR( )` statement for output.The continuation character for quoted parameter strings, in HSPICE, is a double backslash (`\\`). (Outside of quoted strings, the single backslash (`\`) is the continuation character.)

Built-In Functions and Variables

In addition to simple arithmetic operations (+, –, *, /), you can use the built-in functions listed in Table 18 and the variables listed in Table 17 on page 225 in HSPICE expressions.

Table 18 Synopsys HSPICE Built-in Functions

HSPICE Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x (radians)
cos(x)	cosine	trig	Returns the cosine of x (radians)
tan(x)	tangent	trig	Returns the tangent of x (radians)
asin(x)	arc sine	trig	Returns the inverse sine of x (radians)
acos(x)	arc cosine	trig	Returns the inverse cosine of x (radians)
atan(x)	arc tangent	trig	Returns the inverse tangent of x (radians)
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x (radians)
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic cosine of x (radians)
tanh(x)	hyperbolic tangent	trig	Returns the hyperbolic tangent of x (radians)
abs(x)	absolute value	math	Returns the absolute value of x: |x|
sqrt(x)	square root	math	Returns the square root of the absolute value of x: sqrt(-x)=-sqrt(|x|)
pow(x,y)	absolute power	math	Returns the value of x raised to the integer part of y: $x^{(\text{integer part of y})}$
pwr(x,y)	signed power	math	Returns the absolute value of x, raised to the y power, with the sign of x: (sign of x)$\|x\|^y$

Table 18 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
x**y	power		If x<0, returns the value of x raised to the integer part of y. If x=0, returns 0. If x>0, returns the value of x raised to the y power.
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(|x|)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)$\log_{10}(\|x\|)$
exp(x)	exponential	math	Returns e, raised to the power x: e^x
db(x)	decibels	math	Returns the base 10 logarithm of the absolute value of x, multiplied by 20, with the sign of x: (sign of x)$20\log_{10}(\|x\|)$
int(x)	integer	math	Returns the integer portion of x. The fractional portion of the number is lost.
nint(x)	integer	math	Rounds x up or down, to the nearest integer.
sgn(x)	return sign	math	Returns -1 if x is less than 0. Returns 0 if x is equal to 0. Returns 1 if x is greater than 0
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y: (sign of y)|x|
min(x,y)	smaller of two args	control	Returns the numeric minimum of x and y
max(x,y)	larger of two args	control	Returns the numeric maximum of x and y
val(element)	get value	various	Returns a parameter value for a specified element. For example, val(r1) returns the resistance value of the *r1* resistor.

Table 18 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
val(element. parameter)	get value	various	Returns a value for a specified parameter of a specified element. For example, val(rload.temp) returns the value of the *temp* (temperature) parameter for the *rload* element.
val(model_type: model_name. model_param)	get value	various	Returns a value for a specified parameter of a specified model of a specific type. For example, val(nmos:mos1.rs) returns the value of the *rs* parameter for the *mos1* model, which is an nmos model type.
lv(<Element>) or lx(<Element>)	element templates	various	Returns various element values during simulation. See Element Template Output on page 266 for more information.
v(<Node>), i(<Element>)...	circuit output variables	various	Returns various circuit values during simulation. See DC and Transient Output Variables on page 251 for more information.
[cond] ?x : y	ternary operator		Returns *x* if *cond* is not zero. Otherwise, returns *y*. .param z='condition ? x:y'
<	relational operator (less than)		Returns 1 if the left operand is less than the right operand. Otherwise, returns 0. .para x=y<z (y less than z)
<=	relational operator (less than or equal)		Returns 1 if the left operand is less than or equal to the right operand. Otherwise, returns 0. .para x=y<=z (y less than or equal to z)
>	relational operator (greater than)		Returns 1 if the left operand is greater than the right operand. Otherwise, returns 0. .para x=y>z (y greater than z)

Table 18 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
>=	relational operator (greater than or equal)		Returns 1 if the left operand is greater than or equal to the right operand. Otherwise, returns 0. .para x=y>=z (y greater than or equal to z)
==	equality		Returns 1 if the operands are equal. Otherwise, returns 0. .para x=y==z (y equal to z)
!=	inequality		Returns 1 if the operands are not equal. Otherwise, returns 0. .para x=y!=z (y not equal to z)
&&	Logical AND		Returns 1 if neither operand is zero. Otherwise, returns 0. .para x=y&&z (y AND z)
||	Logical OR		Returns 1 if either or both operands are not zero. Returns 0 only if both operands are zero. .para x=y||z (y OR z)

Example

```
.parameters p1=4 p2=5 p3=6
r1 1 0 value='p1 ? p2+1 : p3'
```

HSPICE reserves the variable names listed in Table 19 on page 232 for use in elements, such as E, G, R, C, and L. You can use them in expressions, but you cannot redefine them; for example, this statement would be illegal:

```
.param temper=100
```

Table 19 Synopsys HSPICE Special Variables

HSPICE Form	Function	Class	Description
time	current simulation time	control	Uses parameters to define the current simulation time, during transient analysis.

Table 19 Synopsys HSPICE Special Variables (Continued)

HSPICE Form	Function	Class	Description
temper	current circuit temperature	control	Uses parameters to define the current simulation temperature, during transient/temperature analysis.
hertz	current simulation frequency	control	Uses parameters to define the frequency, during AC analysis.

Parameter Scoping and Passing

If you use parameters to define values in sub-circuits, you need to create fewer similar cells, to provide enough functionality in your library. You can pass circuit parameters into hierarchical designs, and assign different values to the same parameter within individual cells, when you run simulation.

For example, if you use parameters to set the initial state of a latch in its subcircuit definition, then you can override this initial default in the instance call. You need to create only one cell, to handle both initial state versions of the latch.

You can also use parameters to define the cell layout. For example, you can use parameters in a MOS inverter, to simulate a range of inverter sizes, with only one cell definition. Local instances of the cell can assign different values to the size parameter for the inverter.

In HSPICE, you can also perform Monte Carlo analysis or optimization on a cell that uses parameters.

How you handle hierarchical parameters depends on how you construct and analyze your cells. You can construct a design in which information flows from the top of the design, down into the lowest hierarchical levels.

- To centralize the control at the top of the design hierarchy, set *global* parameters.
- To construct a library of small cells that are individually controlled from within, set *local* parameters and build up to the block level.

This section describes the scope of parameter names, and how HSPICE resolves naming conflicts between levels of hierarchy.

Library Integrity

Integrity is a fundamental requirement for any symbol library. Library integrity can be as simple as a consistent, intuitive name scheme, or as complex as libraries with built-in range checking.

Library integrity might be poor if you use libraries from different vendors in a circuit design. Because names of circuit parameters are not standardized between vendors, two components can include the same parameter name for different functions. For example, one vendor might build a library that uses the name `Tau` as a parameter to control one or more subcircuits in their library. Another vendor might use `Tau` to control a different aspect of their library. If you set a global parameter named `Tau` to control one library, you also modify the behavior of the second library, which might not be the intent.

If the scope of a higher-level parameter is global to all subcircuits at lower levels of the design hierarchy, higher-level definitions override lower-level parameter values with the same names. The scope of a lower-level parameter is local to the subcircuit where you define the parameter (but global to all subcircuits that are even lower in the design hierarchy). Local scoping rules in HSPICE prevent higher-level parameters from overriding lower-level parameters of the same name, when that is not desired.

Reusing Cells

Parameter name problems also occur if different groups collaborate on a design. Global parameters prevail over local parameters, so all circuit designers must know the names of all parameters, even those used in sections of the design for which they are not responsible. This can lead to a large investment in standard libraries. To avoid this situation, use local parameter scoping, to encapsulate all information about a section of a design, within that section.

Creating Parameters in a Library

To ensure that the input netlist includes critical, user-supplied parameters when you run simulation, you can use “illegal defaults”—that is, defaults that cause the simulator to abort if you do not supply overrides for the defaults.

If a library cell includes illegal defaults, you must provide a value for each instance of those cells. If you do not, the simulation aborts.

For example, you might define a default MOSFET width of 0.0. HSPICE aborts, because MOSFET models require this parameter.

Example 1

```
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0 $ Inherit illegal values by default
mp1 <NodeList> <Model> L=1u W='Wid*2'
mn1 <NodeList> <Model> L=1u W=Wid
.ENDS

* Invoke symbols in a design
x1 A Y1 Inv        $ Bad! No widths specified
x2 A Y2 Inv Wid=1u $ Overrides illegal value for Width
```

This simulation aborts on the `x1` subcircuit instance, because you never set the required `Wid` parameter on the subcircuit instance line. The `x2` subcircuit simulates correctly. Additionally, the instances of the `Inv` cell are subject to accidental interference, because the `Wid` global parameter is exposed outside the domain of the library. Anyone can specify an alternative value for the parameter, in another section of the library or the circuit design. This might prevent the simulation from catching the condition on `x1`.

Example 2

In this example, the name of a global parameter conflicts with the internal library parameter named `Wid`. Another user might specify such a global parameter, in a different library. In this example, the user of the library has specified a different meaning for the `Wid` parameter, to define an independent source.

```
.Param Wid=5u         $ Default Pulse Width for source
v1 Pulsed 0 Pulse ( 0v 5v 0u 0.1u 0.1u Wid 10u )
...
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0      $ Inherit illegals by default
mp1 <NodeList> <Model> L=1u W='Wid*2'
mn1 <NodeList> <Model> L=1u W=Wid
.Ends
* Invoke symbols in a design
x1 A Y1 Inv        $ Incorrect width!
x2 A Y2 Inv Wid=1u   $ Incorrect! Both x1 and x2
$ simulate with mp1=10u and
$ mn1=5u instead of 2u and 1u.
```

Under global parameter scoping rules, simulation succeeds, but incorrectly. HSPICE does not warn you that the x1 inverter has no assigned width, because the global parameter definition for `Wid` overrides the subcircuit default.

Note:

Similarly, sweeping with different values of *Wid* dynamically changes both the *Wid* library internal parameter value, and the pulse width value to the *Wid* value of the current sweep.

In global scoping, the highest-level name prevails, when resolving name conflicts. Local scoping uses the lowest-level name.

When you use the parameter inheritance method, you can specify to use local scoping rules. This feature can cause different results than you obtained using HSPICE versions before release 95.1, on existing circuits.

When you use local scoping rules, the Example 2 netlist correctly aborts in x1 for W=0 (default Wid=0, in the `.SUBCKT` definition, has higher precedence, than the `.PARAM` statement). This results in the correct device sizes for x2. This change can affect your simulation results, if you intentionally or accidentally create a circuit such as the second one shown above.

As an alternative to width testing in the Example 2 netlist, you can use `.OPTION DEFW` to achieve a limited version of library integrity. This option sets the default width for all MOS devices during a simulation. Part of the definition is still in the top-level circuit, so this method can still make unwanted changes to library values, without notification from the HSPICE simulator.

Table 20 compares the three primary methods for configuring libraries, to achieve required parameter checking for default MOS transistor widths.

Table 20 Methods for Configuring Libraries

Method	Parameter Location	Pros	Cons
Local	On a .SUBCKT definition line	Protects library from global circuit parameter definitions, unless you override it. Single location for default values.	You cannot use it with versions of HSPICE before Release 95.1.
Global	At the global level and on .SUBCKT definition lines	Works with older HSPICE versions.	An indiscreet user, another vendor assignment, or the intervening hierarchy can change the library. Cannot override a global value at a lower level.

Table 20 Methods for Configuring Libraries (Continued)

Method	Parameter Location	Pros	Cons
Special	.OPTION DEFW statement	Simple to do.	Third-party libraries, or other sections of the design, might depend on .OPTION DEFW.

String Parameter

HSPICE uses a special delimiter to identify string and double parameter types. The single quotes (‘), double quotes (“), or curly brackets ({}) do not work for these kinds of delimiters. Instead, use the sp1=str('string') keyword for an sp1 parameter definition and use the str(sp1) keyword for a string parameter instance.

Example

The following sample netlist shows an example of how you can use these definitions for various commands, keywords, parameters, and elements:

```
xibis1 vccq vss out in IBIS
+ IBIS_FILE=str('file1.ibs') IBIS_MODEL=str('model1')
xibis2 vccq vss out in IBIS
+ IBIS_FILE=str('file2.ibs') IBIS_MODEL=str('model2')

.subckt IBIS vccq vss out in
+ IBIS_FILE=str('file.ibs')
+ IBIS_MODEL=str('ibis_model')
ven en 0 vcc
BMCH vccq vss out in en v0dq0 vccq vss buffer=3
+ file= str(IBIS_FILE) model=str(IBIS_MODEL)
+ typ=typ ramp_rwf=2 ramp_fwf=2 power=on
.ends
```

HSPICE can now support these kinds of definitions and instances with the following netlist components:

- .PARAM statements
- .SUBCKT statements
- FQMODEL keywords
- S Parameters
- FILE and MODEL keywords

- B Elements
- RLGCFILE, UMODEL, FSMODEL, RLGCMODEL, TABLEMODEL, and SMODEL keywords in the W Element

Parameter Defaults and Inheritance

Use the .OPTION PARHIER parameter to specify scoping rules.

Syntax:

```
.OPTION PARHIER=< GLOBAL | LOCAL >
```

The default setting is GLOBAL.

Example

This example explicitly shows the difference between local and global scoping for using parameters in subcircuits.

The input netlist includes the following:

```
.OPTION parhier=<global | local>
.PARAM DefPwid=1u
.SUBCKT Inv a y DefPwid=2u DefNwid=1u
Mp1 <MosPinList> pMosMod L=1.2u W=DefPwid
Mn1 <MosPinList> nMosMod L=1.2u W=DefNwid
.ENDS
```

Set the .OPTION PARHIER=parameter scoping option to GLOBAL. The netlist also includes the following input statements:

```
xInv0 a y0 Inv            $ override DefPwid default,
$ xInv0.Mp1 width=1u
xInv1 a y1 Inv DefPwid=5u $ override DefPwid=5u,
$ xInv1.Mp1 width=1u

.measure tran Wid0 param='lv2(xInv0.Mp1)' $ lv2 is the
               $ template for
.measure tran Wid1 param='lv2(xInv1.Mp1)' $ the channel
               $ width
           $ 'lv2(xInv1.Mp1)'
.ENDS
```

Simulating this netlist produces the following results in the listing file:

```
wid0=1.0000E-06
wid1=1.0000E-06
```

If you change the `.OPTION PARHIER=parameter scoping` option to `LOCAL`:

```
xInv0 a y0 Inv            $ not override .param
  $ DefPwid=2u,
  $ xInv0.Mp1 width=2u
xInv1 a y1 Inv DefPwid=5u      $ override .param
         $ DefPwid=2u,
         $ xInv1.Mp1 width=5u:
.measure tran Wid0 param='lv2(xInv0.Mp1)'$ override the
.measure tran Wid1 param='lv2(xInv1.Mp1)'$ global .PARAM
...
```

Simulation produces the following results in the listing file:

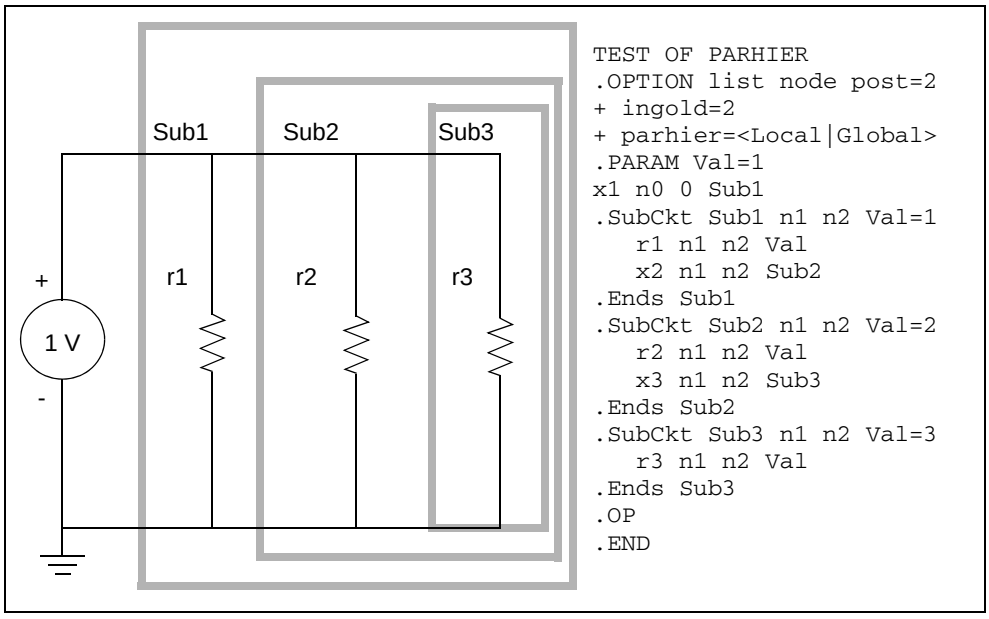
```
wid0=2.0000E-06
wid1=5.0000E-06
```

Parameter Passing

Figure 28 on page 239 shows a flat representation of a hierarchical circuit, which contains three resistors.

Each of the three resistors obtains its simulation time resistance from the *Val* parameter. The netlist defines the *Val* parameter in four places, with three different values.

Figure 28 Hierarchical Parameter Passing Problem



The total resistance of the chain has two possible solutions: 0.3333Ω and 0.5455Ω.

You can use `.OPTION PARHIER` to specify which parameter value prevails, when you define parameters with the same name at different levels of the design hierarchy.

Under global scoping rules, if names conflict, the top-level assignment `.PARAM Val=1` overrides the subcircuit defaults, and the total is 0.3333Ω. Under local scoping rules, the lower level assignments prevail, and the total is 0.5455Ω (one, two, and three ohms in parallel).

The example in Figure 28 produces the results in Table 21, based on how you set `.OPTION PARHIER` to local/global:

Table 21 PARHIER=LOCAL vs. PARHIER=GLOBAL Results

Element	PARHIER=Local	PARHIER=Global
r1	1.0	1.0
r2	2.0	1.0
r3	3.0	1.0

Parameter Passing Solutions

Changes in scoping rules can cause different simulation results for circuit designs created before HSPICE Release 95.1, than for designs created after that release. The checklist below determines whether you will see simulation differences when you use the new default scoping rules. These checks are especially important if your netlists contain devices from multiple vendor libraries.

- Check your sub-circuits for parameter defaults, on the `.SUBCKT` or `.MACRO` line.
- Check your sub-circuits for a `.PARAM` statement, within a `.SUBCKT` definition.
- To check your circuits for global parameter definitions, use the `.PARAM` statement.
- If any of the names from the first three checks are identical, set up two HSPICE simulation jobs: one with `.OPTION PARHIER=GLOBAL`, and one with `.OPTION PARHIER=LOCAL`. Then look for differences in the output.

7

Simulation Output

Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.

You can also use output variables in behavioral circuit analysis, modeling, and simulation techniques. To display electrical specifications such as rise time, slew rate, amplifier gain, and current density, use the output format features.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Command Reference*.

Overview of Output Statements

Output Commands

The input netlist file contains output statements, including `.PRINT`, `.PLOT`, `.GRAPH`, `.PROBE`, `.MEASURE`, `.DOUT`, and `.STIM`. Each statement specifies the output variables, and the type of simulation result, to display—such as `.DC`, `.AC`, or `.TRAN`. When you specify `.OPTION POST`, Synopsys HSPICE puts all output variables, referenced in `.PRINT`, `.PLOT`, `.GRAPH`, `.PROBE`, `.MEASURE`, `.DOUT`, and `.STIM` statements into HSPICE output files.

HSPICE RF supports only `.OPTION POST`, `.OPTION PROBE`, `.PRINT`, `.PROBE`, and `.MEASURE` statements. It does not support `.DOUT`, `.PLOT`, `.GRAPH`, or `.STIM` statements. CosmosScope provides high-resolution, post-simulation, and interactive display of waveforms.

Table 22 Output Statements

Output Statement	Description
.PRINT	Prints numeric analysis results in the output listing file (and post-processor data, if you specify .OPTION POST).
.PLOT (HSPICE only)	Obsolete option. Use .PRINT or ..PROBE to generate necessary plot in the output listing file. Generates low-resolution (ASCII) plots in the output listing file (and post-processor data, if you specify .OPTION POST), in HSPICE only (not supported in HSPICE RF).
.GRAPH (HSPICE only)	Obsolete option. Use .PRINT or ..PROBE to generate necessary plot in the output listing file. Generates high-resolution plots for specific printing devices (such as HP LaserJet), or in PostScript format (intended for hard-copy outputs, without using a post-processor).
.PROBE	Outputs data to post-processor output files, but not to the output listing (used with .OPTION PROBE, to limit output).
.MEASURE	Prints the results of specific user-defined analyses (and post-processor data, if you specify .OPTION POST), to the output listing file. or HSPICE RF
.DOUT	Specifies the expected final state of an output signal (HSPICE only; not supported in HSPICE RF).
.STIM (HSPICE only)	Specifies simulation results to transform to PWL, Data Card, or Digital Vector File format.

Output Variables

The output format statements require special output variables, to print or plot analysis results for nodal voltages and branch currents. HSPICE or HSPICE RF uses the following output variables:

- DC and transient analysis
- AC analysis

- element template (HSPICE)
- `.MEASURE` statement
- parametric analysis

For HSPICE or HSPICE RF, DC and transient analysis displays:

- individual nodal voltages: *V*(*n1* [,*n2*])
- branch currents: *I*(*Vxx*)
- element power dissipation: *In*(*element*)

AC analysis displays imaginary and real components of a nodal voltage or branch current, and the magnitude and phase of a nodal voltage or branch current. AC analysis results also print impedance parameters, and input and output noise.

Element template analysis displays element-specific nodal voltages, branch currents, element parameters, and the derivatives of the element's node voltage, current, or charge.

The `.MEASURE` statement variables define the electrical characteristics to measure in a `.MEASURE` statement analysis or HSPICE RF.

Parametric analysis variables are mathematical expressions, which operate on nodal voltages, branch currents, element template variables (HSPICE only; not supported in HSPICE RF), or other parameters that you specify. Use these variables when you run behavioral analysis of simulation results. See Using Algebraic Expressions on page 228 or HSPICE RF.

Displaying Simulation Results

The following section describes the statements that you can use to display simulation results for your specific requirements.

.PRINT Statement

The `.PRINT` statement specifies output variables for which HSPICE or HSPICE RF prints values.

- The maximum number of variables in a single `.PRINT` statement, was 32 before Release 2002.2, but has been extended. For example, you can enter:

```
.PRINT v(1) v(2) ... v(32) v(33) v(34)
```

This function previously required two .PRINT statements:

```
.PRINT v(1) v(2) ... v(32)
   .PRINT v(33) v(34)
```

- To simplify parsing of the output listings, HSPICE or HSPICE RF prints a single x in the first column, to indicate the beginning of the .PRINT output data. A single y in the first column indicates the end of the .PRINT output data.

You can include wildcards in .PRINT statements.

You can also use the iall keyword in a .PRINT statement, to print all branch currents of all diode, BJT, JFET, or MOSFET elements in your circuit design.

Example

If your circuit contains four MOSFET elements (named m1, m2, m3, m4), then .PRINT iall (m*) is equivalent to .PRINT i(m1) i(m2) i(m3) i(m4). It prints the output currents of all four MOSFET elements.

Statement Order

HSPICE or HSPICE RF creates different .sw0 and .tr0 files, based on the order of the .PRINT and .DC statements. If you do not specify an analysis type for a .PRINT command, the type matches the last analysis command in the netlist, before the .PRINT statement.

.PLOT Statement

Note:

This is an obsolete statement. You can gain the sa
th o .PRINT statemenl sehe

To make HSPICE find plot limits for each plot individually, use .OPTION PLIM to create a different axis for each plot variable. The PLIM option is similar to the plot limit algorithm in SPICE2G.6, where each plot can have limits different from any other plot. A number from 2 through 9 indicates the overlap of two or more traces on a plot.

If more than one output variable appears on the same plot, HSPICE prints *and* plots the first variable specified. To print out more than one variable, include another .PLOT statement.

You can specify an unlimited number of .PLOT statements for each type of analysis. To set the plot width, use .OPTION CO (columns out). If you set CO=80, the plot has 50 columns. If CO=132, the plot has 100 columns.

You can include wildcards in .PLOT statements (HSPICE only).

.PROBE Statement

HSPICE or HSPICE RF usually saves all voltages, supply currents, and output variables. Set .OPTION PROBE, to save output variables only. Use the .PROBE statement to specify the quantities to print in the output listing.

If you are interested only in the output data file, and you do not want tabular or plot data in your listing file, set .OPTION PROBE and use .PROBE to select the values to save in the output listing.

You can include wildcards in .PROBE statements.

.GRAPH Statement

Note:

> This is an obsolete statement. You can gain the same functionality by using the .PROBE statement (see .PROBE Statement on page 245).

Use the .GRAPH statement when you need high-resolution plots of HSPICE simulation results.

Note:

> You cannot use .GRAPH statements in the PC version of HSPICE, or in any versions of HSPICE RF.

The .GRAPH statement is similar to the .PLOT statement, with the addition of an optional model. When you specify a model, you can add or change graphing properties for the graph. The .GRAPH statement generates a .gr# graph data

file and sends this file directly to the default high resolution graphical device (to specify this device, set `PRTDEFAULT` in the meta.cfg configuration file).

.MODEL Statement for .GRAPH

For a description of how to use the `.MODEL` statement with `.GRAPH`, see the .MODEL command in the *HSPICE Command Reference*. HSPICE RF does not support the `.GRAPH` statement.

Table 23 Model Parameters

Name (Alias)	Default	Description
MONO	0.0	Monotonic option. MONO=1 automatically resets the x-axis, if any change occurs in the x direction.
TIC	0.0	Shows tick marks.
FREQ	0.0	Plots symbol frequency. ▪ A value of 0 does not generate plot symbols. ▪ A value of *n* generates a plot symbol every *n* points. This is not the same as the FREQ keyword in element statements (see the "Modeling Filters and Networks" chapter in the *HSPICE Applications Manual*).
XGRID, YGRID	0.0	Set these values to 1.0, to turn on the axis grid lines.
XMIN, *XMA*X	0.0	▪ If XMIN is not equal to XMAX, then XMIN and XMAX determine the x-axis plot limits. ▪ If XMIN equals XMAX, or if you do not set XMIN and XMAX, then HSPICE automatically sets the plot limits. These limits apply to the actual x-axis variable value, regardless of the XSCAL type.
XSCAL	1.0	Scale for the x-axis. Two common axis scales are: Linear(LIN) (XSCAL=1) Logarithm(LOG) (XSCAL=2)

Table 23 Model Parameters (Continued)

Name (Alias)	Default	Description
YMIN, *YMAX*	0.0	▪ If YMIN is not equal to YMAX, then YMIN and YMAX determine the y-axis plot limits. The y-axis limits in the .GRAPH statement overrides YMIN and YMAX in the model. ▪ If you do not specify plot limits, HSPICE sets the plot limits. These limits apply to the actual y-axis variable value, regardless of the YSCAL type.
YSCAL	1.0	Scale for the y-axis. Two common axis scales are: Linear(LIN) (XSCAL=1) Logarithm(LOG) (XSCAL=2)

Using Wildcards in PRINT, PROBE, PLOT, and GRAPH Statements

You can include wildcards in `.PRINT` and `.PROBE` statements (HSPICE and HSPICE RF), and in `.PLOT` and `.GRAPH` statements (HSPICE only). Refer to this example netlist in the discussion that follows:

```
* test wildcard
.option post
v1 1 0 10
r1 1 n20 10
r20 n20 n21 10
r21 n21 0 10
.dc v1 1 10 1
***Wildcard equivalent for:
*.print i(r1) i(r20) i(r21) i(v1)
.print i(*)
***Wildcard equivalent for:
*.probe v(0) v(1)
.probe v(?)
***Wildcard equivalent for:
*.print v(n20) v(n21)
.print v(n2?)
***Wildcard equivalent for:
*.probe v(n20, 1) v(n21, 1)
.probe v(n2*, 1)
.end
```

Supported Wildcard Templates

```
v vm vr vi vp vdb vt
i im ir ii ip idb it
p pm pr pi pp pdb pt
lxn<n> lvn<n> (n is a number 0~9)
i1 im1 ir1 ii1 ip1 idb1 it1
i2 im2 ir2 ii2 ip2 idb2 it2
i3 im3 ir3 ii3 ip3 idb3 it3
i4 im4 ir4 ii4 ip4 idb4 it4
iall
```

For detailed information about the templates, see `.PRINT` statement (see Selecting Simulation Output Parameters on page 251).

Using wildcards in statements such as `v(n2?)` and `v(n2*,1)` in the preceding test case (named test wildcard), you can also use the following in statements (they are not equivalent if you use an `.AC` statement instead of a `.DC` statement):

```
vm(n2?) vr(n2?) vi(n2?) vp(n2?) vdb(n2?) vt(n2?)
vm(n2*,1) vr(n2*,1) vi(n2*,1) vp(n2*,1) vdb(n2*,1) vt(n2*,1)
```

Using wildcards in statements such as `i(*)` in this test wildcard case. You can also use the following in statements (they are not equivalent if you use an `.AC` statement instead of a `.DC` statement):

```
im(*) ir(*) ip(*) idb(*) it(*)
```

`iall` is an output template for all branch currents of diode, BJT, JFET, or MOSFET output. For example, `iall(m*)` is equivalent to:

```
i1(m*) i2(m*) i3(m*) i4(m*).
```

Print Control Options

The codes that you can use to specify the element templates for output in HSPICE or HSPICE RF are:

- `.OPTION CO` to set column widths in printouts.
- `.WIDTH` statement to set the width of a printout.
- `.OPTION INGOLD` for output in exponential form.

- .OPTION POST to display high-resolution AvanWaves plots of simulation results, on either a graphics terminal or a high-resolution laser printer.
- .OPTION ACCT to generate a detailed accounting report. HSPICE RF does not support this statement.

Changing the File Descriptor Limit

A simulation that uses a large number of .ALTER statements might fail, because of the limit on the number of file descriptors. For example, for a Sun workstation, the default number of file descriptors is 64, so a design with more than 50 .ALTER statements probably fails, with the following error message:

```
error could not open output spool file /tmp/tmp.nnn
a critical system resource is inaccessible or exhausted
```

To prevent this error on a Sun workstation, enter the following operating system command, before you start the simulation:

```
limit descriptors 128
```

For platforms other than Sun workstations, ask your system administrator to help you increase the number of files that you can open concurrently.

Printing the Subcircuit Output

The following examples demonstrate how to print or plot voltages of nodes that are in subcircuit definitions, using .PRINT, .PLOT, .PROBE, or .GRAPH.

Note:

> In the following example, you can substitute .PROBE, .PLOT, or .GRAPH instead of .PRINT. HSPICE RF does not support .PLOT or .GRAPH.

Example 1

```
.GLOBAL vdd vss
X1 1 2 3 nor2
X2 3 4 5 nor2
.SUBCKT nor2 A B Y
.PRINT v(B) v(N1) $ Print statement 1
M1 N1 A vdd vdd pch w=6u l=0.8u
M2 Y B N1 vdd pch w=6u l=0.8u
M3 Y A vss vss vss nch w=3u l=0.8u
M4 Y B vss vss nch w=3u l=0.8u
.ENDS
```

Print statement 1 prints out the voltage on the B input node, and on the N1 internal node for every instance of the nor2 subcircuit.

```
.PRINT v(1) v(X1.A) $ Print statement 2
```

The preceding .PRINT statement specifies two ways to print the voltage on the A input of the X1 instance.

```
.PRINT v(3) v(X1.Y)   v(X2.A) $ Print statement 3
```

The preceding .PRINT statement specifies three different ways to print the voltage at the Y output of the X1 instance (or the A input of the X2 instance).

```
.PRINT v(X2.N1) $ Print statement 4
```

The preceding .PRINT statement prints the voltage on the N1 internal node of the X2 instance.

```
.PRINT i(X1.M1) $ Print statement 5
```

The preceding .PRINT statement prints out the drain-to-source current, through the M1 MOSFET in the X1 instance.

Example 2

```
X1 5 6 YYY
 .SUBCKT YYY 15 16
  X2 16 36 ZZZ
  R1 15 25 1
  R2 25 16 1
 .ENDS
 .SUBCKT ZZZ 16 36
  C1 16 0 10P
  R3 36 56 10K
  C2 56 0 1P
 .ENDS
 .PRINT V(X1.25) V(X1.X2.56) V(6)
```

Value	Description
V(X1.25)	Local node to the YYY subcircuit definition, which the X1 subcircuit calls.
V(X1.X2.56)	Local node to the ZZZ subcircuit. The X2 subcircuit calls this node; X1 calls X2.
V(6)	Voltage of node 16, in the X1 instance of the YYY subcircuit.

This example prints voltage analysis results at node 56, within the X2 and X1 subcircuits. The full path, X1.X2.56, specifies that node 56 is within the X2 subcircuit, which in turn is within the X1 subcircuit.

Selecting Simulation Output Parameters

Parameters provide the appropriate simulation output. To define simulation parameters, use the `.OPTION` and `.MEASURE` statements, and define specific variable elements.

DC and Transient Output Variables

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for any element.
- Current output for a subcircuit pin.
- Element templates (HSPICE only). For each device type, the templates contain:
  - values of variables that you set
  - state variables
  - element charges
  - capacitance currents
  - capacitances
  - derivatives

Print Control Options on page 248 summarizes the codes that you can use, to specify the element templates for output in HSPICE or HSPICE RF.

Nodal Capacitance Output

Syntax

`Cap(`*`nxxx`*`)`

For nodal capacitance output, HSPICE prints or plots the capacitance of the specified node nxxxx.

Example

```
.print dc Cap(5) Cap(6)
```

Nodal Voltage

Syntax

V(*n1*<,*n2*>)

Parameter	Description
n1, n2	HSPICE or HSPICE RF prints or plots the voltage difference (*n1-n2*) between the specified nodes. If you omit *n2*, HSPICE or HSPICE RF prints or plots the voltage difference between *n1* and ground (node 0).

Current: Independent Voltage Sources

Syntax

I(V*xxx*)

Parameter	Description
Vxxx	Voltage source element name. If an independent power supply is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, I(X*1*.V*xxx*).

Example

```
.PLOT TRAN I(VIN)
.PRINT DC I(X1.VSRC)
.PLOT DC I(XSUB.XSUBSUB.VY)
```

Current: Element Branches

Syntax

I*n*(W*www*)
I*all*(W*www*)

Parameter	Description
n	Node position number, in the element statement. For example, if the element contains four nodes, I3 is the branch current output for the third node. If you do not specify *n*, HSPICE or HSPICE RF assumes the first node.
Wwww	Element name. To access current output for an element in a subcircuit, append a dot and the subcircuit name to the element name. For example, I3(X1.Wwww).
Iall (Wwww)	An alias just for diode, BJT, JFET, and MOSFET devices. ▪ If *Wwww* is a diode, it is equivalent to: I1(Wwww) I2(Wwww). ▪ If *Wwww* is one of the other device types, it is equivalent to: I1(Wwww) I2(Wwww) I3(Wwww) I4(Wwww)

Example 1

```
I1(R1)
```

This example specifies the current through the first R1 resistor node.

Example 2

```
I4(X1.M1)
```

This example specifies the current, through the fourth node (the substrate node) of the M1 MOSFET, defined in the X1 subcircuit.

Example 3

```
I2(Q1)
```

The last example specifies the current, through the second node (the base node) of the Q1 bipolar transistor.

To define each branch circuit, use a single element statement. When HSPICE or HSPICE RF evaluates branch currents, it inserts a zero-volt power supply, in series with branch elements.

If HSPICE cannot interpret a `.PRINT` or `.PLOT` statement that contains a branch current, it generates a warning.

Branch current direction for the elements in Figure 29 through Figure 34 is defined in terms of arrow notation (current direction), and node position number (terminal type).

Figure 29 Resistor (node1, node2)

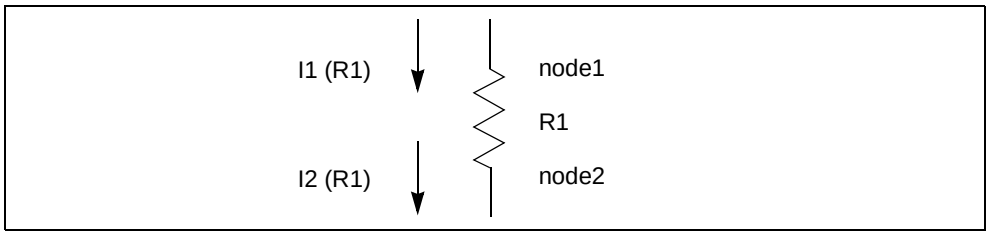


Figure 30 Inductor (node1, node2); capacitor (node 1, node2)

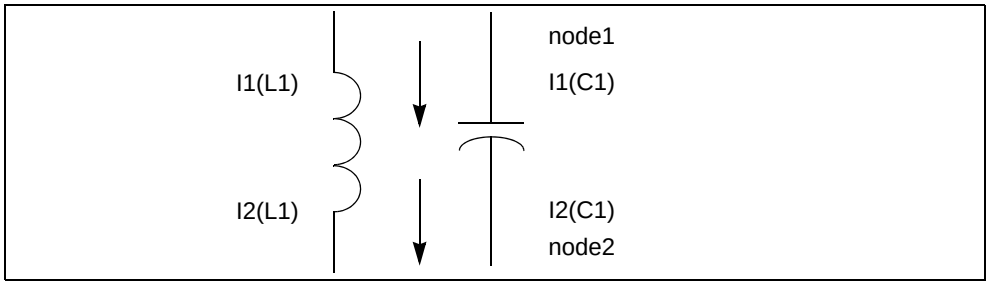


Figure 31 Diode (node1, node2)

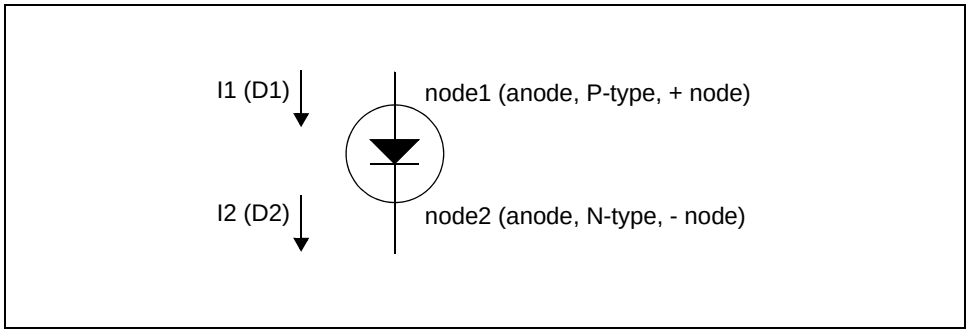


Figure 32 JFET (node1, node2, node3) - n-channel

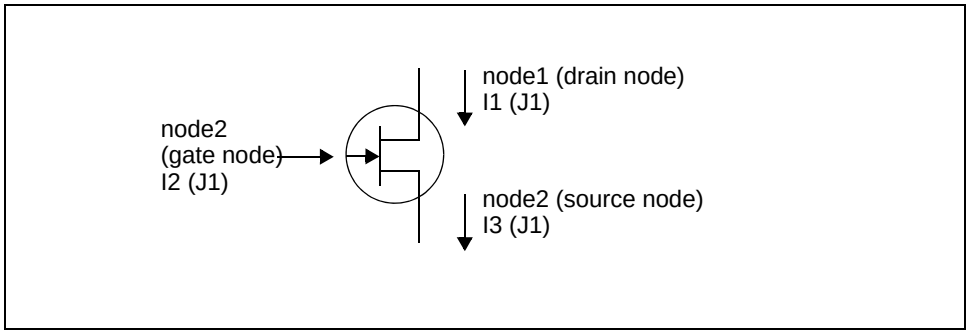


Figure 33 MOSFET (node1, node2, node3, node4) - n-channel

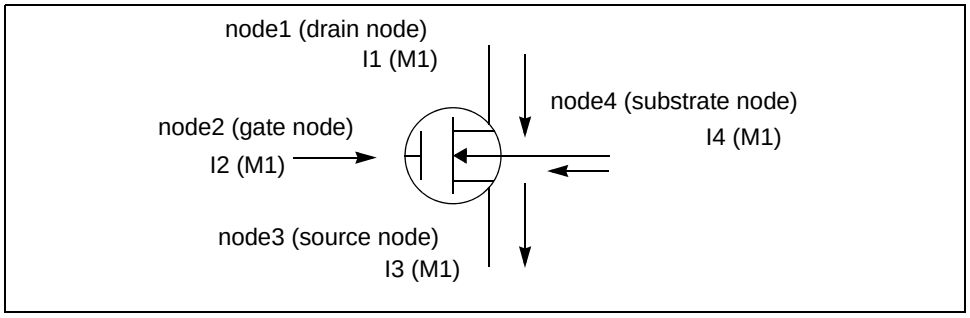
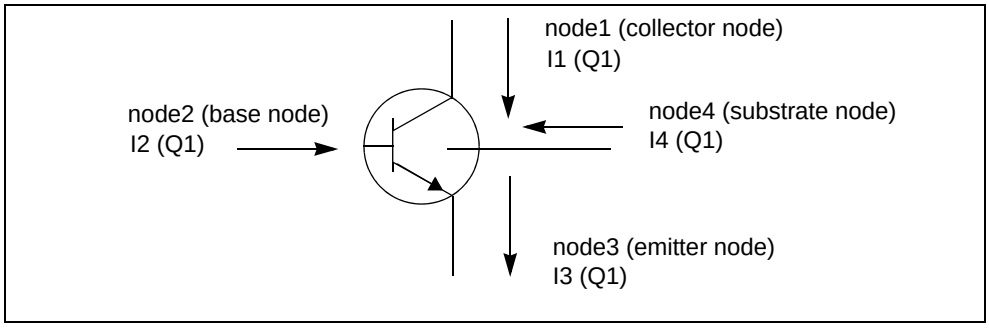


Figure 34 BJT (node1, node2, node3, node4) - npn



Current: Subcircuit Pin

Syntax

```
ISUB(X****.****)
```

Example

```
.PROBE ISUB(X1.PIN1)
```

Power Output

For power calculations, HSPICE or HSPICE RF computes dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H). To compute this power, HSPICE or HSPICE RF multiplies the voltage across an element, and its corresponding branch current.

However, for semiconductor devices, HSPICE or HSPICE RF calculates only the dissipated power. It excludes the power stored in the device junction or parasitic capacitances, from the device power computation. The following sections show equations for calculating the power that different types of devices dissipate.

HSPICE or HSPICE RF also computes the total power dissipated in the circuit, which is the sum of the power dissipated in:

- Devices
- Resistors
- Independent current sources
- All dependent sources

For hierarchical designs, HSPICE or HSPICE RF also computes the power dissipation for each subcircuit.

Note:

> For the total power (dissipated power + stored power), HSPICE or HSPICE RF does not add the power of each independent source (voltage and current sources).

Print or Plot Power

Note:

To output the instantaneous element power, and the total power dissipation, use a .PRINT or .PLOT statement in HSPICE.HSPICE RF does not support .PLOT statements or power variables in DC/transient analysis.

```
.PRINT <DC | TRAN> P(element_or_subcircuit_name)POWER
```

HSPICE calculates power only for transient and DC sweep analyses. Use the .MEASURE statement to compute the average, RMS, minimum, maximum, and peak-to-peak value of the power. The POWER keyword invokes the total power dissipation output.

HSPICE RF supports p(instance) but not the POWER variable in DC/transient analysis.

Example

```
.PRINT TRAN   P(M1)   P(VIN)   P(CLOAD)   POWER
.PRINT TRAN   P(Q1)   P(DIO)   P(J10)   POWER
.PRINT TRAN   POWER   $ Total transient analysis
* power dissipation
.PLOT DC POWER   P(IIN)   P(RLOAD)   P(R1)
.PLOT DC POWER   P(V1)   P(RLOAD)   P(VS)
.PRINT TRAN P(Xf1) P(Xf1.Xh1)
```

Diode Power Dissipation

$$Pd = Vpp' \cdot (Ido + Icap) + Vp'n \cdot Ido$$

Parameter	Description
Pd	Power dissipated in the diode.
Ido	DC component of the diode current.
Icap	Capacitive component of the diode current.
Vp'n	Voltage across the junction.
Vpp'	Voltage across the series resistance, RS.

BJT Power Dissipation

- Vertical

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vee' \cdot Ietot + Vsc' \cdot Iso - Vcc' \cdot Istot$$

- Lateral

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vbb' \cdot Ibtot + Vee' \cdot Ietot$$
$$Vsb' \cdot Iso - Vbb' \cdot Istot$$

Parameter	Description
Ibo	DC component of the base current.
Ico	DC component of the collector current.
Iso	DC component of the substrate current.
Pd	Power dissipated in a BJT.
Ibtot	Total base current (excluding the substrate current).
Ictot	Total collector current (excluding the substrate current).
Ietot	Total emitter current.
Istot	Total substrate current.
Vb'e'	Voltage across the base-emitter junction.
Vbb'	Voltage across the series base resistance, RB.
Vc'e'	Voltage across the collector-emitter terminals.
Vcc'	Voltage across the series collector resistance, RC.
Vee'	Voltage across the series emitter resistance, RE.
Vsb'	Voltage across the substrate-base junction.
Vsc'	Voltage across the substrate-collector junction.

JFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vgd' \cdot Igdo + Vgs' \cdot Igso + Vs's \cdot (Ido + Igso + Icgs) + Vdd' \cdot (Ido - Igdo - Icgd)$$

Parameter	Description
Icgd	Capacitive component of the gate-drain junction current.
Icgs	Capacitive component of the gate-source junction current.
Ido	DC component of the drain current.
Igdo	DC component of the gate-drain junction current.
Igso	DC component of the gate-source junction current.
Pd	Power dissipated in a JFET.
Vd's'	Voltage across the internal drain-source terminals.
Vdd'	Voltage across the series drain resistance, RD.
Vgd'	Voltage across the gate-drain junction.
Vgs'	Voltage across the gate-source junction.
Vs's	Voltage across the series source resistance, RS.

MOSFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vbd' \cdot Ibdo + Vbs' \cdot Ibso + Vs's \cdot (Ido + Ibso + Icbs + Icgs) + Vdd' \cdot (Ido - Ibdo - Icbd - Icgd)$$

Parameter	Description
Ibdo	DC component of the bulk-drain junction current.
Ibso	DC component of the bulk-source junction current.
Icbd	Capacitive component of the bulk-drain junction current.
Icbs	Capacitive component of the bulk-source junction current.

Parameter	Description
Icgd	Capacitive component of the gate-drain current.
Icgs	Capacitive component of the gate-source current.
Ido	DC component of the drain current.
Pd	Power dissipated in the MOSFET.
Vbd'	Voltage across the bulk-drain junction.
Vbs'	Voltage across the bulk-source junction.
Vd's'	Voltage across the internal drain-source terminals.
Vdd'	Voltage across the series drain resistance, RD.
Vs's	Voltage across the series source resistance, RS.

AC Analysis Output Variables

Output variables for AC analysis include:

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for a subcircuit pin.
- Element branch current.
- Impedance (Z), admittance (Y), hybrid (H), and scattering (S) parameters.
- Input and output impedance, and admittance.

Table 24 lists AC output variable types. In this table, the type symbol is appended to the variable symbol, to form the output variable name. For example, VI is the imaginary part of the voltage, or IM is the magnitude of the current.

Table 24 AC Output Variable Types

Type Symbol	Variable Type
DB	decibel
I	imaginary part
M	magnitude
P	phase
R	real part
T	group delay

Specify real or imaginary parts, magnitude, phase, decibels, and group delay for voltages and currents.

Nodal Capacitance Output

Syntax

`Cap(`*`nxxx`*`)`

For nodal capacitance output, HSPICE prints or plots the capacitance of the specified node nxxxx.

Example

```
.print ac Cap(5) Cap(6)
```

Nodal Voltage

Syntax

`Vz(`*`n1`*`<,`*`n2`*`>)`

Parameter	Description
z	Specifies the voltage output type (see Table 24 on page 261)
n1, n2	Specifies node names. If you omit *n2*, HSPICE or HSPICE RF assumes ground (node 0).

Example

This example applies to HSPICE, but not HSPICE RF. It plots the magnitude of the AC voltage of node 5, using the VM output variable. HSPICE uses the VDB output variable to plot the voltage at node 5, and uses the VP output variable to plot the phase of the nodal voltage at node 5.

```
.PLOT AC VM(5) VDB(5) VP(5)
```

HSPICE and SPICE Methods for Producing Complex Results

To produce complex results, an AC analysis uses either the SPICE or HSPICE method, and the `.OPTION ACOUT` control option, to calculate the values of real or imaginary parts for complex voltages of AC analysis, and their magnitude, phase, decibel, and group delay values. The default for HSPICE is `ACOUT=1`. To use the SPICE method, set `ACOUT=0`.

A typical use of the SPICE method is to calculate the nodal vector difference, when comparing adjacent nodes in a circuit. You can use this method to find the phase or magnitude across a capacitor, inductor, or semiconductor device.

Use the HSPICE method to calculate an inter-stage gain in a circuit (such as an amplifier circuit), and to compare its gain, phase, and magnitude.

The following examples define the AC analysis output variables for the HSPICE method, and then for the SPICE method.

HSPICE Method Example:

Real and imaginary:

```
VR(N1,N2)= REAL [V(N1,0)] - REAL [V(N2,0)]
VI(N1,N2)= IMAG [V(N1,0)] - IMAG [V(N2,0)]
```

Magnitude:

$$VM(N1,0)= [VR(N1,0)^2 + VI(N1,0)^2]^{0.5}$$
$$VM(N2,0)= [VR(N2,0)^2 + VI(N2,0)^2]^{0.5}$$

```
VM(N1,N2)= VM(N1,0) - VM(N2,0)
```

Phase:

```
VP(N1,0)= ARCTAN[VI(N1,0)/VR(N1,0)]
VP(N2,0)= ARCTAN[VI(N2,0)/VR(N2,0)]
VP(N1,N2)= VP(N1,0) - VP(N2,0)
```

Decibel:

```
VDB(N1,0)=20·LOG10[VM(N1,0)]
VDB(N1,N2)= 20·LOG10(VM(N1,0)/VM(N2,0))
```

Parameter	Description
n	Node position number, in the element statement. For example, if the element contains four nodes, IM3 denotes the magnitude of the branch current output for the third node.
Wwww	Element name. If the element is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, IM3(X1.Wwww).

```
.PRINT AC IP1(Q5) IM1(Q5) IDB4(X1.M1)
```

If you use the form In(Xxxx) for AC analysis output, then HSPICE or HSPICE RF prints the magnitude value, IMn(Xxxx).

Current: Subcircuit Pin

Syntax

```
ISUB(X****.****)
```

Example

```
.PROBE ISUB(X1.PIN1)
```

Group Time Delay

The TD group time delay is associated with AC analysis. TD is the negative derivative of the phase in radians, with respect to radian frequency. HSPICE or HSPICE RF uses the difference method to compute TD:

$$TD = -\frac{1}{360} \cdot \frac{(phase2 - phase1)}{(f2 - f1)}$$

phase1 and *phase2* are the phases (in degrees) of the specified signal, at the f1 and f2 frequencies (in hertz).

Syntax

```
.PRINT AC VT(10) VT(2,25) IT(RL)
.PLOT AC IT1(Q1) IT3(M15) IT(D1)
```

Note:

> Because the phase has a discontinuity every 360×, TD shows the same discontinuity, even though TD is continuous. The `.PRINT` example applies to both HSPICE and HSPICE RF, but the `.PLOT` example applies only to HSPICE.

Example

```
INTEG.SP ACTIVE INTEGRATOR
****** INPUT LISTING
******
V1    1    0    .5    AC    1
R1    1    2       2K
C1    2    3       5NF
E3    3    0       2 0 -1000.0
.AC DEC       15    1K    100K
.PLOT AC       VT(3)    (0,4U)    VP(3)
.END
```

Network

Syntax

Xij (z), ZIN(*z*), ZOUT(*z*), YIN(*z*), YOUT(*z*)

Parameter	Description
X	Specifies Z (impedance), Y (admittance), H (hybrid), or S (scattering).
ij	i and j can be 1 or 2. They identify the matrix parameter to print.
z	Output type (see Table 24 on page 261). If you omit z, HSPICE or HSPICE RF prints the magnitude of the output variable.
ZIN	Input impedance. For a one-port network, ZIN, Z11, and H11 are the same.
ZOUT	Output impedance.
YIN	Input admittance. For a one-port network, YIN and Y11 are the same.
YOUT	Output admittance.

Example

```
.PRINT    AC    Z11(R)    Z12(R)    Y21(I)    Y22    S11    S11(DB)
.PRINT    AC    ZIN(R)    ZIN(I)    YOUT(M)    YOUT(P)    H11(M)
.PLOT    AC    S22(M)    S22(P)    S21(R)    H21(P)    H12(R)
```

The .PRINT examples apply to both HSPICE and HSPICE RF. The .PLOT example applies only to HSPICE.

Noise and Distortion

This section describes the variables used for noise and distortion analysis.

Syntax

```
ovar <(z)>
```

Parameter	Description
ovar	Noise and distortion analysis parameter. It can be ONOISE (output noise), INOISE (equivalent input noise), or any of the distortion analysis parameters (HD2, HD3, SIM2, DIM2, DIM3).
z	Output type (only for distortion). If you omit z, HSPICE or HSPICE RF outputs the magnitude of the output variable.

Example

```
.PRINT DISTO HD2(M) HD2(DB)
```

Prints the magnitude and decibel values of the second harmonic distortion component, through the load resistor that you specified in the `.DISTO` statement (not shown). You cannot use the `.DISTO` statement in HSPICE RF.

```
.PLOT NOISE INOISE ONOISE
```

Note:

> You can specify the noise and distortion output variable, and other AC output variables, in the `.PRINT AC` or `.PLOT AC` statements. The `.PRINT` example applies to both HSPICE and HSPICE RF. The `.PLOT` example applies only to HSPICE.

Element Template Output

(HSPICE) The `.PRINT`, `.PROBE`, `.PLOT`, and `.GRAPH` statements use element templates to output user-input parameters, state variables, stored charges, capacitor currents, capacitances, and derivatives of variables. Element templates are listed at the end of this chapter.t

Syntax

Elname:*Property*

Parameter	Description
Elname	Name of the element.
Property	Property name of an element, such as a user-input parameter, state variable, stored charge, capacitance current, capacitance, or derivative of a variable.

The alias is:

LVnn(*Elname*)
LXnn(*Elname*)

Parameter	Description
LV	Form to obtain output of user-input parameters, and state variables.
LX	Form to obtain output of stored charges, capacitor currents, capacitances, and derivatives of variables.
nn	Code number for the desired parameter (listed in tables in this section).
Elname	Name of the element.

Example

```
.PLOT TRAN V(1,12) I(X2.VSIN) I2(Q3) DI01:GD
.PRINT TRAN X2.M1:CGGBO M1:CGDBO X2.M1:CGSBO
```

The .PRINT example applies to both HSPICE and HSPICE RF; the .PLOT example applies to HSPICE only.

Specifying User-Defined Analysis (.MEASURE)

Use the .MEASURE statement to modify information, and to define the results of successive HSPICE or HSPICE RF simulations.

Computing the measurement results is based on postprocessing output. If you use the INTERP option to reduce the size of the postprocessing output, then the measurement results can contain interpolation errors. For more

information, see the .OPTION INTERP option in the *HSPICE Command Reference*.

Fundamental measurement modes in HSPICE are:

- Rise, fall, and delay
- Find-when
- Equation evaluation
- Average, RMS, min, max, and peak-to-peak
- Integral evaluation
- Derivative evaluation
- Relative error

If a .MEASURE statement does not execute, then HSPICE or HSPICE RF writes 0.0e0 in the *.mt#* file as the .MEASURE result, and writes FAILED in the output listing file. Use .OPTION MEASFAIL to write results to the *.mt#*, *.ms#*, or *.ma#* files. For more information, see the .OPTION MEASFAIL option in the *HSPICE Command Reference*.

Note:

> Beginning with the 2004.03 release, the *.mt#* format consists of 72 characters in a line and fields that contain 16 characters each. The extra line that existed in previous releases has been removed.

To control the output variables, listed in .MEASURE statements, use the .PUTMEAS option. For more information, see the .OPTION PUTMEAS option in the *HSPICE Command Reference*

In versions of HSPICE before 2003.09, to automatically sort large numbers of .MEASURE statements, you could use the MEASSORT option. Starting in version 2003.09, this option is obsolete. Now the measure performance is order-independent, and HSPICE ignores this option.

.MEASURE Statement Order

The .MEASURE statement matches the last analysis command in the netlist before the .MEASURE statement.

Example

```
.tran 20p 1.0n sweep sigma -3 3 0.5
.tran 20p 1.0n sweep monte=20
.meas mover max v(2,1)
```

In this example, .meas matches the second .tran statement and generates only one measure output file.

.MEASURE Parameter Types

You cannot use measurement parameter results that the .PARAM statements in .SUBCKT blocks produce, outside of the subcircuit. That is, you cannot pass any measurement parameters defined in .SUBCKT statements, as bottom-up parameters in hierarchical designs.

Measurement parameter names must not conflict with standard parameter names. HSPICE or HSPICE RF issues an error message, if it encounters a measurement parameter with the same name as a standard parameter definition.

To prevent .MEASURE statement parameters from overwriting parameter values in other statements, HSPICE or HSPICE RF keeps track of parameter types. If you use the same parameter name in both a .MEASURE statement and a .PARAM statement at the same hierarchical level, simulation terminates and reports an error.

No error occurs if parameter assignments are at different hierarchical levels. .PRINT statements that occur at different levels, do not print hierarchical information for parameter name headings.

Example

In HSPICE RF simulation output, you cannot apply .MEASURE to waveforms generated from another .MEASURE statement in a parameter sweep.

The following example illustrates how HSPICE or HSPICE RF handles .MEASURE statement parameters.

```
...
.MEASURE tran length TRIG v(clk) VAL=1.4
+ TD=11ns RISE=1 TARGv(neq) VAL=1.4 TD=11ns
+ RISE=1
.SUBCKT path out in width=0.9u length=600u
+ rm1 in m1 m2mg w='width' l='length/6'
...
.ENDS
```

In the above listing, the *length* in the resistor statement:

```
rm1 in m1 m2mg w='width' l='length/6'
```

does not inherit its value from *length* in the .MEASURE statement:

```
.MEASURE tran length ...
```

because they are of different types.

The correct value of l in rm1 should be:

```
l=length/6=100u
```

The value should not be derived from a measured value in transient analysis.

FIND and WHEN Functions

The `FIND` and `WHEN` functions of the `.MEASURE` statement specify to measure:

- Any independent variables (time, frequency, parameter).
- Any dependent variables (voltage or current for example).
- Derivative of a dependent variable, if a specific event occurs.

You can use these measure statements in unity gain frequency or phase measurements. You can also use these statements to measure the time, frequency, or any parameter value:

- When two signals cross each other.
- When a signal crosses a constant value.

The measurement starts after a specified time delay, `TD`. To find a specific event, set `RISE`, `FALL`, or `CROSS` to a value (or parameter), or specify `LAST` for the last event.

`LAST` is a reserved word; you cannot use it as a parameter name in the above measure statements. For definitions of parameters of the measure statement, see Displaying Simulation Results on page 243.

Equation Evaluation

Use the Equation Evaluation form of the `.MEASURE` statement to evaluate an equation, that is a function of the results of previous `.MEASURE` statements. The equation must not be a function of node voltages or branch currents.

The `expression` option is an arithmetic expression that uses results from other prior `.MEASURE` statements. If `equation` or `expression` includes node voltages or branch currents, Unexpected results may incur.

Average, RMS, MIN, MAX, INTEG, and PP

Average (AVG), RMS, MIN, MAX, and peak-to-peak (PP) measurement modes report statistical functions of the output variable, rather than analysis values.

- AVG calculates the area under an output variable, divided by the periods of interest.
- RMS divides the square root of the area under the output variable square, by the period of interest.
  - MIN reports the minimum value of the output function, over the specified interval.
  - MAX reports the maximum value of the output function, over the specified interval.
  - PP (peak-to-peak) reports the maximum value, minus the minimum value, over the specified interval.

    AVG, RMS, and INTEG have no meaning in a DC data sweep, so if you use them, HSPICE or HSPICE RF issues a warning message.

INTEGRAL Function

The INTEGRAL function reports the integral of an output variable, over a specified period.

DERIVATIVE Function

The DERIVATIVE function provides the derivative of:

- An output variable, at a specified time or frequency.
- Any sweep variable, depending on the type of analysis.
- A specified output variable, when some specific event occurs.

In the HSPICE RF example below, the SLEW measurement provides the slope of V(OUT) during the first time, when V(1) is 90% of VDD.

```
.MEAS TRAN SLEW DERIV V(OUT) WHEN V(1)='0.90*VDD'
```

ERROR Function

The relative error function reports the relative difference between two output variables. You can use this format in optimization and curve-fitting of measured data. The relative error format specifies the variable to measure and calculate, from the `.PARAM` variable. To calculate the relative error between the two, HSPICE or HSPICE RF uses the `ERR`, `ERR1`, `ERR2`, or `ERR3` function. With this format, you can specify a group of parameters to vary, to match the calculated value and the measured data.

Error Equations

ERR

1. *ERR* sums the squares of (M-C)/max (`M`, `MINVAL`) for each point.
2. It then divides by the number of points.
3. Finally, it calculates the square root of the result.
   - M (meas_var) is the measured value of the device or circuit response.
   - C (calc_var) is the calculated value of the device or circuit response.
   - NPTS is the number of data points.

$$ERR = \left[\frac{1}{NPTS}\cdot\sum_{i=1}^{NPTS}\left(\frac{M_i - C_i}{\max(MINVAL, M_i)}\right)^2\right]^{1/2}$$

ERR1

`ERR1` computes the relative error at each point. For `NPTS` points, HSPICE or HSPICE RF calculates `NPTS ERR1` error functions. For device characterization, the `ERR1` approach is more efficient than the other error functions (`ERR`, `ERR2`, `ERR3`).

$$ERR1_i = \frac{M_i - C_i}{\max(MINVAL, M_i)}, \text{ i=1,NPTS}$$

HSPICE or HSPICE RF does not print out each calculated ERR1 value. When you set the ERR1 option, HSPICE or HSPICE RF calculates an ERR value, as follows:

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR1_i^2\right]^{1/2}$$

ERR2

This option computes the absolute relative error, at each point. For NPTS points, HSPICE or HSPICE RF calls NPTS error functions.

$$ERR2_i = \left|\frac{M_i - C_i}{\max(MINVAL, M_i)}\right|, \text{ i=1,NPTS}$$

The returned value printed for ERR2 is:

$$ERR = \frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR2_i$$

ERR3

$$ERR3_i = \frac{\pm\log\left|\frac{M_i}{C_i}\right|}{\left|\log[\max(MINVAL, |M_i|)]\right|}, \text{ i=1,NPTS}$$

The + and - signs correspond to a positive and negative M/C ratio.

Note:

If the M measured value is less than MINVAL, HSPICE or HSPICE RF uses MINVAL instead. If the absolute value of M is less than the IGNOR or YMIN value, or greater than the YMAX value, the error calculation does not consider this point.

Reusing Simulation Output as Input Stimuli

You can use the .STIM statement to reuse the results (output) of one simulation, as input stimuli in a new simulation.

Note:

.STIM is an abbreviation of .STIMULI. You can use either form to specify this statement in HSPICE. HSPICE RF does not support this statement.

The .STIM statement specifies:

- Expected stimulus (PWL source, data card, or VEC file).
- Signals to transform.
- Independent variables.

One .STIM statement produces one corresponding output file.

For the syntax and description of the .STIM statement, see the .STIM command in the *HSPICE Command Reference.*

Output Files

The .STIM statement generates the following output files:

Output File Type	Extension
PWL Source	.pwl$_tr#The .STIM statement writes PWL source results to output_file.pwl$_tr#. This output file results from a .STIM <tran> pwl statement in the input file.
Data Card	.dat$_tr#, .dat$_ac#, or .dat$_sw#The .STIM statement writes DATA Card results to output_file.dat$_sw#, output_file.dat$_ac#, or output_file.dat$_tr#. This output file is the result of a .stim <tran| ac|dc> data statement in the input file.
Digital Vector File	.vec$_tr#The .STIM statement writes Digital Vector File results to output_file.vec$_tr#. This output file is the result of a .stim <tran> vec statement in the input file.

Symbol	Description
tr | ac | sw	▪ tr=transient analysis. ▪ ac=AC analysis. ▪ sw=DC sweep analysis.
#	Either a sweep number, or a hard-copy file number. For a single sweep run, the default number is `0`.
$	Serial number of the current .STIM statement, within statements of the same stimulus type (pwl, data, or vec). $=0 ~ n-1 (n is the number of the .STIM statement of that type). The initial $ value is 0. For example, if you specify three .STIM pwl statements, HSPICE generates three PWL output files, with the suffix names pwl0_tr#, pwl1_tr#, and pwl2_tr#.

Element Template Listings

This section applies only to HSPICE. HSPICE RF does not support element template output.

Table 25 Resistor (R Element)

Name	Alias	Description
G	LV1	Conductance at analysis temperature.
R	LV2	Resistance at analysis temperature.
TC1	LV3	First temperature coefficient.
TC2	LV4	Second temperature coefficient.

Table 26 Capacitor (C Element)

Name	Alias	Description
CEFF	LV1	Computed effective capacitance.
IC	LV2	Initial condition.

Table 26 Capacitor (C Element) (Continued)

Name	Alias	Description
Q	LX0	Charge, stored in capacitor.
CURR	LX1	Current, flowing through capacitor.
VOLT	LX2	Voltage, across capacitor.
–	LX3	Capacitance (not used after HSPICE releases after 95.3).

Table 27 Inductor (L Element)

Name	Alias	Description
LEFF	LV1	Computed effective inductance.
IC	LV2	Initial condition.
FLUX	LX0	Flux, in the inductor.
VOLT	LX1	Voltage, across inductor.
CURR	LX2	Current, flowing through inductor.
–	LX4	Inductance (not used after HSPICE releases after 95.3).

Table 28 Mutual Inductor (K Element)

Name	Alias	Description
K	LV1	Mutual inductance.

Table 29 Voltage-Controlled Current Source (G Element)

Name	Alias	Description
CURR	LX0	Current, through the source, if VCCS.
R	LX0	Resistance value, if VCR.

Table 29 Voltage-Controlled Current Source (G Element) (Continued)

Name	Alias	Description
C	LX0	Capacitance value, if VCCAP.
CV	LX1	Controlling voltage.
CQ	LX1	Capacitance charge, if VCCAP.
DI	LX2	Derivative of the source current, relative to the control voltage.
ICAP	LX2	Capacitance current, if VCCAP.
VCAP	LX3	Voltage, across capacitance, if VCCAP.

Table 30 Voltage-Controlled Voltage Source (E Element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Current, through source.
CV	LX2	Controlling voltage.
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 31 Current-Controlled Current Source (F Element)

Name	Alias	Description
CURR	LX0	Current, through source.
CI	LX1	Controlling current.
DI	LX2	Derivative of the source current, relative to the control current.

Table 32 Current-Controlled Voltage Source (H Element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Source current.
CI	LX2	Controlling current.
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 33 Independent Voltage Source (V Element)

Name	Alias	Description
VOLT	LV1	DC/transient voltage.
VOLTM	LV2	AC voltage magnitude.
VOLTP	LV3	AC voltage phase.

Table 34 Independent Current Source (I Element)

Name	Alias	Description
CURR	LV1	DC/transient current.
CURRM	LV2	AC current magnitude.
CURRP	LV3	AC current phase.

Table 35 Diode (D Element)

Name	Alias	Description
AREA	LV1	Diode area factor.
AREAX	LV23	Area, after scaling.
IC	LV2	Initial voltage, across diode.

Table 35 Diode (D Element) (Continued)

Name	Alias	Description
VD	LX0	Voltage, across diode (VD), excluding RS (series resistance).
IDC	LX1	DC current, through diode (ID), excluding RS. Total diode current is the sum of IDC and ICAP.
GD	LX2	Equivalent conductance (GD).
QD	LX3	Charge of diode capacitor (QD).
ICAP	LX4	Current, through the diode capacitor. Total diode current is the sum of IDC and ICAP.
C	LX5	Total diode capacitance.
PID	LX7	Photo current, in diode.

Table 36 BJT (Q Element)

Name	Alias	Description
AREA	LV1	Area factor.
ICVBE	LV2	Initial condition for base-emitter voltage (VBE).
ICVCE	LV3	Initial condition for collector-emitter voltage (VCE).
MULT	LV4	Number of multiple BJTs.
FT	LV5	FT (Unity gain bandwidth).
ISUB	LV6	Substrate current.
GSUB	LV7	Substrate conductance.
LOGIC	LV8	LOG 10 (IC).
LOGIB	LV9	LOG 10 (IB).
BETA	LV10	BETA.

Table 36 BJT (Q Element) (Continued)

Name	Alias	Description
LOGBETAI	LV11	LOG 10 (BETA) current.
ICTOL	LV12	Collector current tolerance.
IBTOL	LV13	Base current tolerance.
RB	LV14	Base resistance.
GRE	LV15	Emitter conductance, 1/RE.
GRC	LV16	Collector conductance, 1/RC.
PIBC	LV18	Photo current, base-collector.
PIBE	LV19	Photo current, base-emitter.
VBE	LX0	VBE.
VBC	LX1	Base-collector voltage (VBC).
CCO	LX2	Collector current (CCO).
CBO	LX3	Base current (CBO).
GPI	LX4	$g\pi = {}^1ib / {}^1vbe$, constant vbc.
GU	LX5	$g\mu = {}^1ib / {}^1vbc$, constant vbe.
GM	LX6	$g_m = {}^1ic / {}^1vbe + {}^1ic / {}^1vbe$, constant vce.
G0	LX7	$g_0 = {}^1ic / {}^1vce$, constant vbe.
QBE	LX8	Base-emitter charge (QBE).
CQBE	LX9	Base-emitter charge current (CQBE).
QBC	LX10	Base-collector charge (QBC).
CQBC	LX11	Base-collector charge current (CQBC).
QCS	LX12	Current-substrate charge (QCS).

Table 36 BJT (Q Element) (Continued)

Name	Alias	Description
CQCS	LX13	Current-substrate charge current (CQCS).
QBX	LX14	Base-internal base charge (QBX).
CQBX	LX15	Base-internal base charge current (CQBX).
GXO	LX16	1/Rbeff Internal conductance (GXO).
CEXBC	LX17	Base-collector equivalent current (CEXBC).
–	LX18	Base-collector conductance (GEQCBO), (not used in HSPICE releases after 95.3).
CAP_BE	LX19	cbe capacitance (Cπ).
CAP_IBC	LX20	cbc internal base-collector capacitance (Cμ).
CAP_SCB	LX21	csc substrate-collector capacitance for vertical transistors. csb substrate-base capacitance for lateral transistors.
CAP_XBC	LX22	cbcx external base-collector capacitance.
CMCMO	LX23	^{1}(TF*IBE) /^{1}vbc.
VSUB	LX24	Substrate voltage.

Table 37 JFET (J Element)

Name	Alias	Description
AREA	LV1	JFET area factor.
VDS	LV2	Initial condition for drain-source voltage.
VGS	LV3	Initial condition for gate-source voltage.
PIGD	LV16	Photo current, gate-drain in JFET.
PIGS	LV17	Photo current, gate-source in JFET.

Table 37 JFET (J Element) (Continued)

Name	Alias	Description
VGS	LX0	VGS.
VGD	LX1	Gate-drain voltage (VGD).
CGSO	LX2	Gate-to-source (CGSO).
CDO	LX3	Drain current (CDO).
CGDO	LX4	Gate-to-drain current (CGDO).
GMO	LX5	Transconductance (GMO).
GDSO	LX6	Drain-source transconductance (GDSO).
GGSO	LX7	Gate-source transconductance (GGSO).
GGDO	LX8	Gate-drain transconductance (GGDO).
QGS	LX9	Gate-source charge (QGS).
CQGS	LX10	Gate-source charge current (CQGS).
QGD	LX11	Gate-drain charge (QGD).
CQGD	LX12	Gate-drain charge current (CQGD).
CAP_GS	LX13	Gate-source capacitance.
CAP_GD	LX14	Gate-drain capacitance.
–	LX15	Body-source voltage (not used after HSPICE release 95.3).
QDS	LX16	Drain-source charge (QDS).
CQDS	LX17	Drain-source charge current (CQDS).
GMBS	LX18	Drain-body (backgate) transconductance (GMBS).

Table 38 MOSFET

Name	Alias	Description
L	LV1	Channel length (L).
W	LV2	Channel width (W).
AD	LV3	Area of the drain diode (AD).
AS	LV4	Area of the source diode (AS).
ICVDS	LV5	Initial condition for drain-source voltage (VDS).
ICVGS	LV6	Initial condition for gate-source voltage (VGS).
ICVBS	LV7	Initial condition for bulk-source voltage (VBS).
–	LV8	Device polarity: ■ 1=forward ■ -1=reverse (not used after HSPICE releases after 95.3).
VTH	LV9	Threshold voltage (bias dependent).
VDSAT	LV10	Saturation voltage (VDSAT).
PD	LV11	Drain diode periphery (PD).
PS	LV12	Source diode periphery (PS).
RDS	LV13	Drain resistance (squares), (RDS).
RSS	LV14	Source resistance (squares), (RSS).
XQC	LV15	Charge-sharing coefficient (XQC).
GDEFF	LV16	Effective drain conductance (1/RDeff).
GSEFF	LV17	Effective source conductance (1/RSeff).
CDSAT	LV18	Drain-bulk saturation current, at -1 volt bias.
CSSAT	LV19	Source-bulk saturation current, at -1 volt bias.
VDBEFF	LV20	Effective drain bulk voltage.

Table 38 MOSFET (Continued)

Name	Alias	Description
BETAEFF	LV21	BETA, effective.
GAMMAEFF	LV22	GAMMA, effective.
DELTAL	LV23	ΔL (MOS6 amount of channel length modulation), (valid only for LEVELs 1, 2, 3 and 6).
UBEFF	LV24	UB effective (valid only for LEVELs 1, 2, 3 and 6).
VG	LV25	VG drive (valid only for LEVELs 1, 2, 3 and 6).
VFBEFF	LV26	VFB effective.
–	LV31	Drain current tolerance (not used in HSPICE releases after 95.3).
IDSTOL	LV32	Source-diode current tolerance.
IDDTOL	LV33	Drain-diode current tolerance.
COVLGS	LV36	Gate-source overlap capacitance.
COVLGD	LV37	Gate-drain overlap capacitance.
COVLGB	LV38	Gate-bulk overlap capacitance.
VBS	LX1	Bulk-source voltage (VBS).
VGS	LX2	Gate-source voltage (VGS).
VDS	LX3	Drain-source voltage (VDS).
CDO	LX4	DC-drain current (CDO).
CBSO	LX5	DC source-bulk diode current (CBSO).
CBDO	LX6	DC drain-bulk diode current (CBDO).
GMO	LX7	DC-gate transconductance (GMO).
GDSO	LX8	DC drain-source conductance (GDSO).

Table 38 MOSFET (Continued)

Name	Alias	Description
GMBSO	LX9	DC-substrate transconductance (GMBSO).
GBDO	LX10	Conductance of the drain diode (GBDO).
GBSO	LX11	Conductance of the source diode (GBSO).
Meyer and Charge Conservation Model Parameters		
QB	LX12	Bulk charge (QB).
CQB	LX13	Bulk-charge current (CQB).
QG	LX14	Gate charge (QG).
CQG	LX15	Gate-charge current (CQG).
QD	LX16	Channel charge (QD).
CQD	LX17	Channel-charge current (CQD).
CGGBO	LX18	CGGBO = ∂Qg/∂Vgb =CGS + CGD + CGB
CGDBO	LX19	CGDBO = ∂Qg/∂Vdb , (for Meyer CGD=-CGDBO)
CGSBO	LX20	CGSBO = ∂Qg/∂Vsb , (for Meyer CGS=-CGSBO)
CBGBO	LX21	CBGBO = ∂Qb/∂Vgb , (for Meyer CGB=-CBGBO)
CBDBO	LX22	CBDBO = ∂Qb/∂Vdb
CBSBO	LX23	CBSBO = ∂Qb/∂Vsb
QBD	LX24	Drain-bulk charge (QBD).
–	LX25	Drain-bulk charge current (CQBD), (not used in HSPICE releases after 95.3).
QBS	LX26	Source-bulk charge (QBS).

Table 38 MOSFET (Continued)

Name	Alias	Description
–	LX27	Source-bulk charge current (CQBS), (not used after HSPICE releases after 95.3).
CAP_BS	LX28	Bulk-source capacitance.
CAP_BD	LX29	Bulk-drain capacitance.
CQS	LX31	Channel-charge current (CQS).
CDGBO	LX32	CDGBO = ∂Qd/∂Vgb
CDDBO	LX33	CDDBO = ∂Qd/∂Vdb
CDSBO	LX34	CDSBO = ∂Qd/∂Vsb

Table 39 Saturable Core Element (K Element)

Name	Alias	Description
MU	LX0	Dynamic permeability (mu), Weber/(amp-turn-meter).
H	LX1	Magnetizing force (H), Ampere-turns/meter.
B	LX2	Magnetic flux density (B), Webers/meter^2.

Table 40 Saturable Core Winding

Name	Alias	Description
LEFF	LV1	Effective winding inductance (Henry).
IC	LV2	Initial condition.
FLUX	LX0	Flux, through winding (Weber-turn).
VOLT	LX1	Voltage, across winding (Volt).

8

Initializing DC/Operating Point Analysis

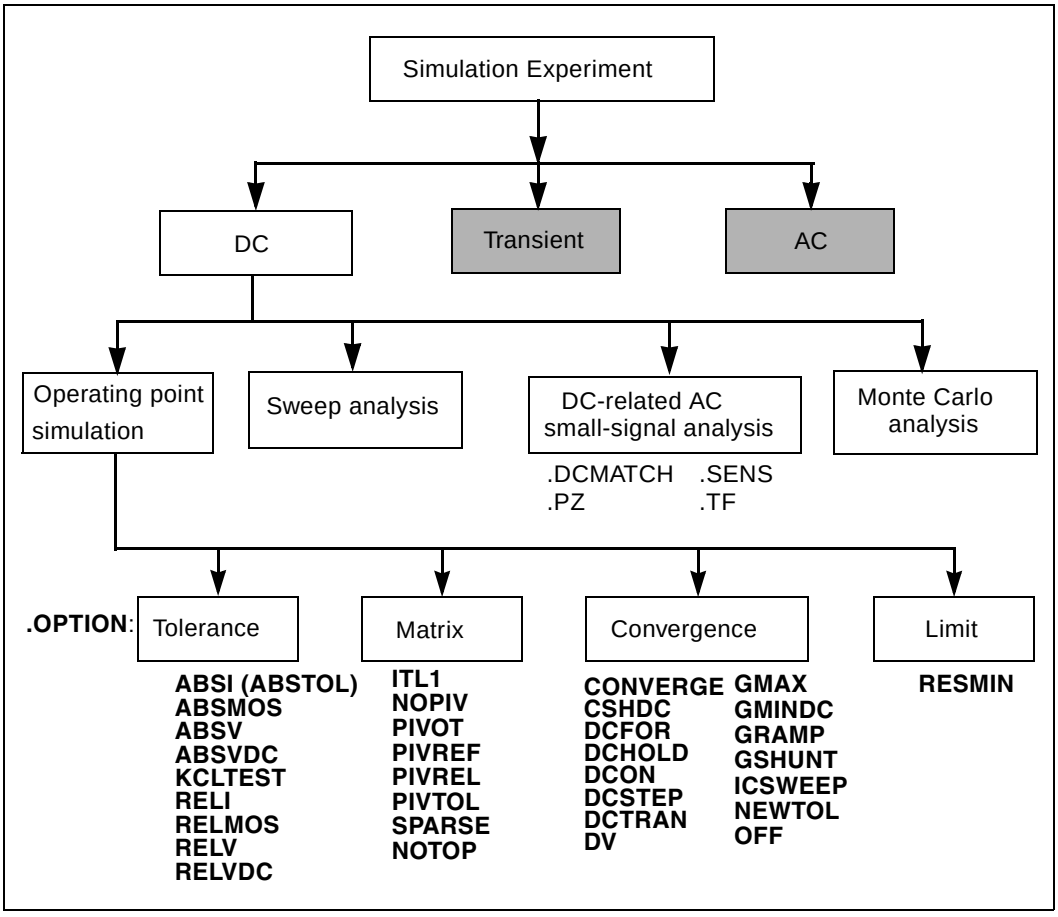
Describes DC initialization and operating point analysis.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Command Reference*.

Simulation Flow

Figure 35 shows the simulation flow for DC analysis in Synopsys HSPICE and HSPICE RF.

Figure 35 DC Initialization and Operating Point Analysis Simulation Flow



Initialization and Analysis

Before it performs `.OP`, `.DC` sweep, `.AC`, or `.TRAN` analyses, HSPICE or HSPICE RF first sets the DC operating point values for all nodes and sources. To do this, HSPICE or HSPICE RF does one of the following:

- Calculates all values
- Applies values specified in `.NODESET` and `.IC` statements
- Applies values stored in an initial conditions file.

The `.OPTION OFF` statement, and the `OFF` and `IC=val` element parameters, also control initialization.

Initialization is fundamental to simulation. HSPICE or HSPICE RF starts any analysis with known nodal voltages (or initial estimates for unknown voltages) and some branch currents. It then iteratively finds the exact solution. Initial estimates that are close to the exact solution increase the likelihood of a convergent solution and a lower simulation time.

A transient analysis first calculates a DC operating point using the DC equivalent model of the circuit (unless you specify the UIC parameter in the .TRAN statement). HSPICE or HSPICE RF then uses the resulting DC operating point as an initial estimate to solve the next timepoint in the transient analysis.

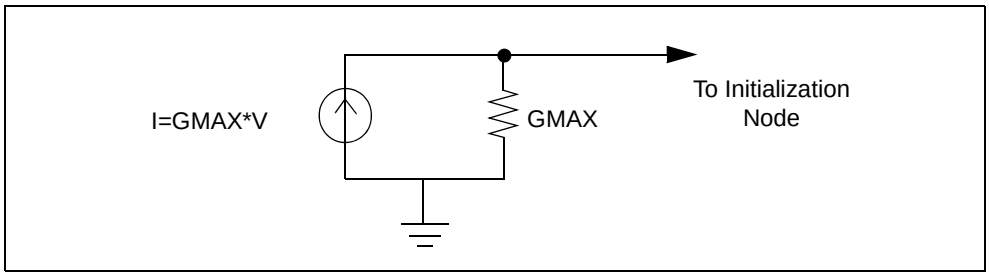
Here's how this is done:

1. If you do not provide an initial guess or if you provide only partial information, HSPICE or HSPICE RF provides a default estimate for each node in the circuit.
2. HSPICE or HSPICE RF then uses this estimate to iteratively find the exact solution.

   The .NODESET and statements supply an initial guess for the exact DC solution of nodes within a circuit.
3. To set any circuit node to any value, use the .NODESET statement.
4. HSPICE or HSPICE RF then connects a voltage source equivalent, to each initialized node (a current source, with a GMAX parallel conductance, set with a .OPTION statement).
5. HSPICE or HSPICE RF next calculates a DC operating point, with the .NODESET voltage source equivalent connected.
6. HSPICE or HSPICE RF disconnects the equivalent voltage sources, which you set in the .NODESET statement, and recalculates the DC operating point.

   This is the DC operating point solution.

Figure 36 Equivalent Voltage Source: NODESET and .IC



The .IC statement provides both an initial guess and a solution for selected nodes within the circuit. Nodes that you initialize with the .IC statement become part of the solution of the DC operating point.

You can also use the OFF option to initialize active devices. The OFF option works with .IC and .NODESET voltages as follows:

1. If the netlist includes any .IC or .NODESET statements, HSPICE or HSPICE RF sets node voltages, according to those statements.
2. If you set the OFF option, then HSPICE or HSPICE RF sets values to zero for the terminal voltages of all active devices (BJTs, diodes, MOSFETs, JFETs, MESFETs) that are not set in .IC or .NODESET statements, or by sources.
3. If element statements specify any IC parameters, HSPICE or HSPICE RF sets those initial conditions.
4. HSPICE or HSPICE RF uses the resulting voltage settings, as the initial guess at the operating point.

   Use OFF to find an exact solution, during an operating point analysis, in a large circuit. The majority of device terminals are at zero volts for the operating point solution. To initialize the terminal voltages to zero for selected active devices, set the OFF parameter in the element statements for those devices.

   After HSPICE finds a DC operating point, use .SAVE to store operating-point node voltages in a *<design>*.ic file. Then use the .LOAD statement to restore operating-point values, from the ic file for later analyses.

   Note:

   HSPICE RF does not support the .SAVE and .LOAD (save and restart) statements.

When you set initial conditions for Transient Analysis:

- If you include UIC in a .TRAN statement, HSPICE or HSPICE RF starts a transient analysis, using node voltages specified in an .IC statement.
- Use the .OP statement, to store an estimate of the DC operating point, during a transient analysis.
- HSPICE RF does not output node voltage from operating point (.OP), if time (t) < 0.
- An internal timestep too small error message indicates that the circuit failed to converge. The cause of the failure can be that HSPICE or HSPICE RF cannot use stated initial conditions to calculate the actual DC operating point.

DC Initialization and Operating Point Calculation

You use a .OP statement in HSPICE or HSPICE RF to:

- Calculate the DC operating point of a circuit
- Produce an operating point during a transient analysis

A simulation can only have one .OP statement.

.OP Statement — Operating Point

When you include an .OP statement in an input file, HSPICE or HSPICE RF calculates the DC operating point of the circuit. You can also use the .OP statement to produce an operating point, during a transient analysis. You can include only one .OP statement in a simulation.

If an analysis requires calculating an operating point, you do not need to specify the .OP statement; HSPICE or HSPICE RF calculates an operating point. If you use a .OP statement, and if you include the UIC keyword in a .TRAN analysis statement, then simulation omits the time=0 operating point analysis, and issues a warning in the output listing.

Output

```
***** OPERATING POINT INFORMATION  TNOM=25.000 TEMP=25.000
***** OPERATING POINT STATUS IS ALL  SIMULATION TIME IS 0.
NODE     VOLTAGE  NODE     VOLTAGE  NODE      VOLTAGE
+ 0:2=0  0:3=437.3258M  0:4=455.1343M
```

```
+ 0:5=478.6763M  0:6=496.4858M  0:7=537.8452M
+ 0:8=555.6659M  0:10=5.0000  0:11=234.3306M
 **** VOLTAGE SOURCES
SUBCKT
ELEMENT  0:VNCE  0:VN7  0:VPCE  0:VP7
VOLTS  0  5.00000  0  -5.00000
AMPS  -2.07407U  -405.41294P  2.07407U  405.41294P
POWER  0.  2.02706N  0.   2.02706N
 TOTAL VOLTAGE SOURCE POWER DISSIPATION=4.0541 N WATTS
**** BIPOLAR JUNCTION TRANSISTORS
SUBCKT
  ELEMENT   0:QN1  0:QN2   0:QN3  0:QN4
* Note: HSPICE RF does not support qn(element)
* charge output.
  MODEL      0:N1     0:N1    0:N1    0:N1
  IB  999.99912N  2.00000U  5.00000U  10.00000U
  IC  -987.65345N  -1.97530U  -4.93827U  -9.87654U
  VBE  437.32588M  455.13437M  478.67632M  496.48580M
  VCE  437.32588M  17.80849M  23.54195M  17.80948M
  VBC  437.32588M  455.13437M  478.67632M  496.48580M
  VS  0.  0.  0.  0.
  POWER  5.39908N  875.09107N  2.27712U  4.78896U
  BETAD -987.65432M -987.65432M -987.65432M -987.65432M
  GM  0.  0.  0.  0.
  RPI  2.0810E+06  1.0405E+06  416.20796K  208.10396K
  RX  250.00000M  250.00000M  250.00000M  250.00000M
  RO  2.0810E+06  1.0405E+06  416.20796K  208.10396K
  CPI  1.43092N  1.44033N  1.45279N  1.46225N
  CMU  954.16927P  960.66843P  969.64689P  977.06866P
  CCS  800.00000P  800.00000P  800.00000P  800.00000P
  BETAAC  0.  0.  0.  0.
  FT   0.  0.  0.  0.
```

Element Statement IC Parameter

Use the element statement parameter, `IC=<val>`, to set DC terminal voltages for selected active devices.

HSPICE uses the value, set in `IC=<val>`, as the DC operating point value, in the DC solution.

- HSPICE RF does not support this option, so IC is always set to `IC=OFF`.

Example

This example describes an H element dependent-voltage source:

```
HXCC 13 20 VIN1 VIN2 IC=0.5, 1.3
```

The current, through VIN1, initializes to 0.5 mA. The current, through VIN2, initializes to 1.3 mA.

Initial Conditions

Use the .IC statement, or the .DCVOLT statement, to set transient initial conditions in HSPICE, but not in HSPICE RF. How it initializes depends on whether the .TRAN analysis statement includes the UIC parameter.

Note:

In HSPICE RF, .IC is always set to OFF.

If you specify the UIC parameter in the .TRAN statement, HSPICE does not calculate the initial DC operating point, but directly enters transient analysis. Transient analysis uses the .IC initialization values as part of the solution for timepoint zero (calculating the zero timepoint applies a fixed equivalent voltage source). The .IC statement is equivalent to specifying the IC parameter on each element statement, but is more convenient. You can still specify the IC parameter, but it does not have precedence over values set in the .IC statement.

If you do *not* specify the UIC parameter in the .TRAN statement, HSPICE computes the DC operating point solution before the transient analysis. The node voltages that you specify in the .IC statement are fixed to determine the DC operating point. HSPICE RF does not output node voltage from operating point (.OP) if time (t) < 0. Transient analysis releases the initialized nodes to calculate the second and later time points.

.NODESET initializes all specified nodal voltages for DC operating point analysis. Use the .NODESET statement to correct convergence problems in DC analysis. If you set the node values in the circuit, close to the actual DC operating point solution, you enhance convergence of the simulation. The HSPICE or HSPICE RF simulator uses the NODESET voltages only in the first iteration.

SAVE and LOAD Statements

HSPICE saves the operating point, unless you use the .SAVE LEVEL=NONE statement. HSPICE restores the saved operating-point file, only if the input file contains a .LOAD statement.

The .SAVE statement in HSPICE stores the operating point of a circuit, in a file that you specify. HSPICE RF does not support the .SAVE statement. For quick DC convergence in subsequent simulations, use the .LOAD statement to input the contents of this file. HSPICE saves the operating point by default, even if the HSPICE input file does not contain a .SAVE statement. To not save the operating point, specify .SAVE LEVEL=NONE.

A parameter or temperature sweep saves only the first operating point.

Note:

HSPICE RF does not support .SAVE and .LOAD statements.

If any node initialization commands, such as .NODESET and .IC, appear in the netlist after the .LOAD command, then they overwrite the .LOAD initialization. If you use this feature to set particular states for multistate circuits (such as flip-flops), you can still use the .SAVE command to speed up the DC convergence.

.SAVE and .LOAD work even on changed circuit topologies. Adding or deleting nodes results in a new circuit topology. HSPICE initializes the new nodes, as if you did not save an operating point. HSPICE ignores references to deleted nodes, but initializes coincidental nodes to the values that you saved from the previous run.

When you initialize nodes to voltages, HSPICE inserts Norton-equivalent circuits at each initialized node. The conductance value of a Norton-equivalent circuit is GMAX=100, which might be too large for some circuits.

If using .SAVE and .LOAD does not speed up simulation, or causes simulation problems, use .OPTION GMAX=1e-12 to minimize the effect of Norton-equivalent circuits on matrix conductances.

HSPICE still uses the initialized node voltages to initialize devices. HSPICE RF does not output node voltage from operating point (.OP), if time (t) < 0.

.SAVE Statement

The .SAVE statement in HSPICE stores the operating point of a circuit, in a file that you specify. HSPICE RF does not support the .SAVE statement. For quick DC convergence in subsequent simulations, use the .LOAD statement to input

the contents of this file. HSPICE saves the operating point by default, even if the HSPICE input file does not contain a .SAVE statement. To not save the operating point, specify .SAVE LEVEL=NONE.

You can save the operating point data as either an .IC or a .NODESET statement.

.LOAD Statement

Use the .LOAD statement to input the contents of a file, that you stored using the .SAVE statement in HSPICE.

Note:

HSPICE RF does not support the .SAVE and .LOAD (save and restart) statements.

Files stored with the .SAVE statement contain operating point data for the point in the analysis at which you executed .SAVE. Do not use the .LOAD command for concatenated netlist files.

.DC Statement—DC Sweeps

You can use the .DC statement in DC analysis, to:

- Sweep any parameter value (HSPICE and HSPICE RF).
- Sweep any source value (HSPICE and HSPICE RF).
- Sweep temperature range (HSPICE and HSPICE RF).
- Perform a DC Monte Carlo (random sweep) analysis (HSPICE only; not supported in HSPICE RF).
- Perform a data-driven sweep (HSPICE and HSPICE RF).
- Perform a DC circuit optimization for a data-driven sweep (HSPICE and HSPICE RF).
- Perform a DC circuit optimization, using start and stop (HSPICE only; not supported in HSPICE RF).
- Perform a DC model characterization (HSPICE only; not supported in HSPICE RF).

The .DC statement format depends on the application that uses it.

Other DC Analysis Statements

HSPICE or HSPICE RF also provides the following DC analysis statements. Each statement uses the DC-equivalent model of the circuit in its analysis. For `.PZ`, the equivalent circuit includes capacitors and inductors.

Statement	Description
.DCMATCH	(HSPICE) A technique for computing the effects of local variations in device characteristics on the DC solution of a circuit.
.PZ	Performs pole/zero analysis.
.SENS	(HSPICE) Obtains DC small-signal sensitivities of output variables for circuit parameters.
.TF	Calculates DC small-signal values for transfer functions (ratio of output variable, to input source).

HSPICE or HSPICE RF includes DC control options, and DC initialization statements, to model resistive parasitics and initialize nodes. These statements enhance convergence properties and accuracy of simulation. This section describes how to perform DC-related, small-signal analysis.

DC Initialization Control Options

Use control options in a DC operating-point analysis, to control DC convergence properties and simulation algorithms. Many of these options also affect transient analysis, because DC convergence is an integral part of transient convergence. Include the following options for *both* DC and transient convergence:

- Absolute and relative voltages.
- Current tolerances.
- Matrix options.

Use `.OPTION` statements to specify the following options, which control DC analysis:

ABSTOL	DV	ITL2	PIVREL
CAPTAB	GDCPATH	KCLTEST	PIVTOL
CSHDC	GRAMP	MAXAMP	RESMIN
DCCAP	GSHDC	NEWTOL	SPARSE
DCFOR	GSHUNT	NOPIV	SYMB
DCHOLD	ICSWEEP	OFF	
DCIC	ITLPTRAN	PIVOT	
DCSTEP	ITL1	PIVREF	

DC and AC analysis also use some of these options. Many of these options also affect the transient analysis, because DC convergence is an integral part of transient convergence. For a description of transient analysis, see Chapter 9, Transient Analysis.

Accuracy and Convergence

Convergence is the ability to solve a set of circuit equations, within specified tolerances, and within a specified number of iterations. In numerical circuit simulation, a designer specifies a relative and absolute accuracy for the circuit solution. The simulator iteration algorithm then attempts to converge to a solution that is within these set tolerances. That is, if consecutive simulations achieve results within the specified accuracy tolerances, circuit simulation has converged. How quickly the simulator converges, is often a primary concern to a designer—especially for preliminary design trials. So designers willingly sacrifice some accuracy for simulations that converge quickly.

Accuracy Tolerances

HSPICE or HSPICE RF uses accuracy tolerances that you specify, to assure convergence. These tolerances determine when, and whether, to exit the convergence loop. For each iteration of the convergence loop, HSPICE or

HSPICE RF subtracts previously-calculated values from the new solution, and compares the result with the accuracy tolerances.

If the difference between two consecutive iterations is within the specified accuracy tolerances, the circuit simulation has converged.

$| Vn^{k} - Vn^{k-1} | <= \text{accuracy tolerance}$

- Vn^{k} is the solution at the *n* timepoint for iteration *k*.
- Vn^{k-1} is the solution at the *n* timepoint for iteration *k* - 1.

As Table 41 shows, HSPICE or HSPICE RF defaults to specific absolute and relative values. You can change these tolerances, so that simulation time is not excessive, but accuracy is not compromised. Accuracy Control Options on page 299 describes the options in Table 41.

Table 41 Absolute and Relative Accuracy Tolerances

Type	.OPTION	Default
Nodal Voltage Tolerances	ABSVDC	50 μv
	RELVDC	.001
Current Element Tolerances	ABSI	1 nA
	RELI	.01
	ABSMOS	1 uA
	RELMOS	.05

HSPICE or HSPICE RF compares nodal voltages and element currents, to the values from the previous iteration.

- If the absolute value of the difference is less than ABSVDC or ABSI, then the node or element has converged.

  ABSV and ABSI set the floor value, below which HSPICE or HSPICE RF ignores values. Values above the floor use RELVDC and RELI as relative tolerances. If the iteration-to-iteration absolute difference is less than these tolerances, then it is convergent.

  Note:

  ABSMOS and RELMOS are the tolerances for MOSFET drain currents.

Accuracy settings directly affect the number of iterations before convergence.

- If accuracy tolerances are tight, the circuit requires more time to converge.
- If the accuracy setting is too loose, the resulting solution can be inaccurate and unstable.

Table 42 shows an example of the relationship between the `RELVDC` value, and the number of iterations.

Table 42 RELV vs. Accuracy and Simulation Time for 2 Bit Adder

RELVDC	Iteration	Delay (ns)	Period (ns)	Fall time (ns)
.001	540	31.746	14.336	1.2797
.005	434	31.202	14.366	1.2743
.01	426	31.202	14.366	1.2724
.02	413	31.202	14.365	1.3433
.05	386	31.203	14.365	1.3315
.1	365	31.203	14.363	1.3805
.2	354	31.203	14.363	1.3908
.3	354	31.203	14.363	1.3909
.4	341	31.202	14.363	1.3916
.4	344	31.202	14.362	1.3904

Accuracy Control Options

The default control option settings are designed to maximize accuracy, without significantly degrading performance. For a description of these options and their settings, see Simulation Speed and Accuracy on page 327.

ABSH	DCON	RELH
ABSI	DCTRAN	RELI
ABSMOS	DI	RELMOS

ABSVDC	GMAX	RELV
CONVERGE	GMINDC	RELVDC

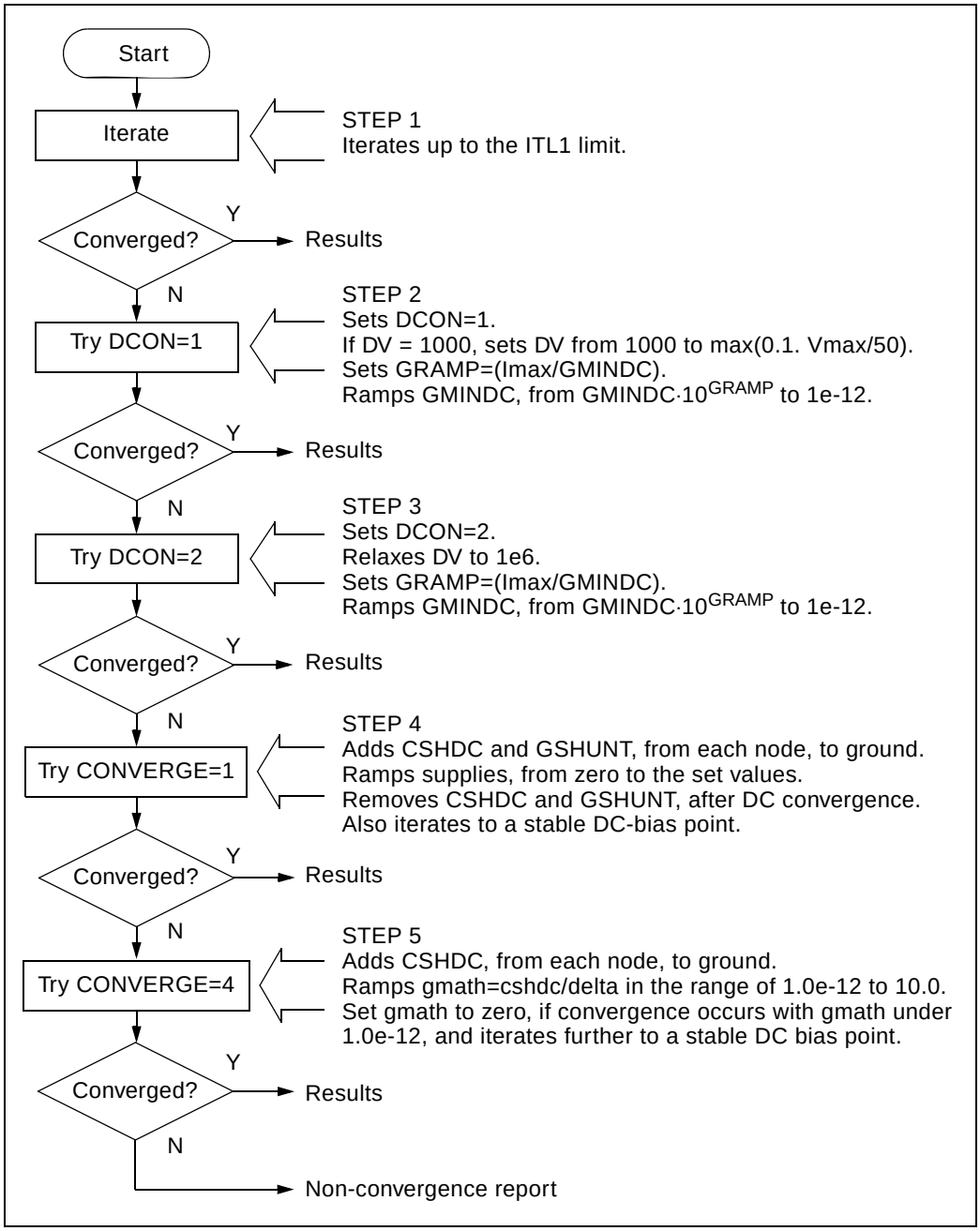
Autoconverge Process

If a circuit does not converge in the number of iterations that `ITL1` specifies, HSPICE or HSPICE RF initiates an auto-convergence process. This process manipulates `DCON`, `GRAMP`, and `GMINDC`, and even `CONVERGE` in some cases. Figure 37 on page 301 shows the autoconverge process.

Note:

HSPICE uses autoconvergence in transient analysis, but it also uses autoconvergence in DC analysis if the Newton-Raphson (N-R) method fails to converge.

Figure 37 Autoconvergence Process Flow Diagram



In Figure 37 above:

- Setting `.OPTION DCON=-1` disables steps 2 and 3.
- Setting `.OPTION CONVERGE=-1` disables steps 4 and 5.

- Setting `.OPTION DCON=-1 CONVERGE=-1` disables steps 2, 3, 4, and 5.
- If you set the `DV` option to a value other than the default, step 2 uses the value you set for `DV`, but step 3 changes `DV` to 1e6.
- Setting `.OPTION GRAMP` has no effect on autoconverge. Autoconverge sets `GRAMP` independently.
- If you set `.OPTION GMINDC`, then `GMINDC` ramps to the value you set, instead of to 1e-12, in steps 2 and 3.

DCON and GMINDC

The `GMINDC` option helps stabilize the circuit, during DC operating-point analysis. For MOSFETs, `GMINDC` helps stabilize the device in the vicinity of the threshold region. HSPICE or HSPICE RF inserts `GMINDC` between:

- Drain and bulk.
- Source and bulk.
- Drain and source.

The drain-to-source `GMINDC` helps to:

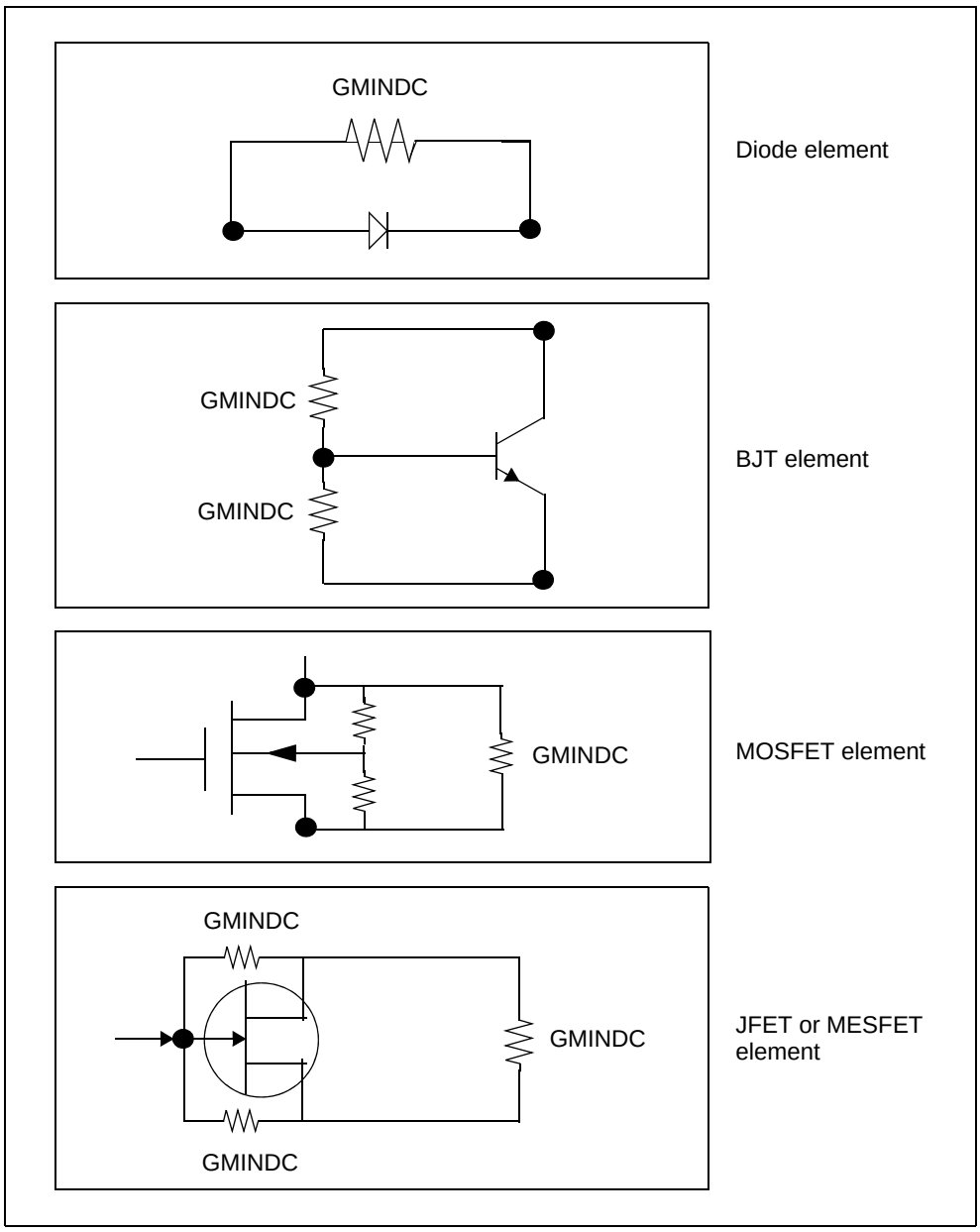
- Linearize the transition from cutoff to weakly-on.
- Smooth-out model discontinuities.
- Compensate for the effects of negative conductances.

The pn junction insertion of `GMINDC` in junction diodes linearizes the low conductance region. As a result, the device behaves like a resistor in the low-conductance region. This prevents the occurrence of zero conductance, and improves the convergence of the circuit.

If a circuit does not converge, HSPICE or HSPICE RF automatically sets the `DCON` option. This option invokes `GMINDC` ramping, in steps 2 and 3 of Figure 37.

`GMINDC` for various elements is shown in Figure 38 on page 303.

Figure 38 GMINDC Insertion



Reducing DC Errors

To reduce DC errors, perform the following steps:

1. To check topology, set `.OPTION NODE`, to list nodal cross-references.
   - Do all MOS p-channel substrates connect to either VCC or positive supplies?
   - Do all MOS n-channel substrates connect to either GND or negative supplies?
   - Do all vertical NPN substrates connect to either GND or negative supplies?
   - Do all lateral PNP substrates connect to negative supplies?
   - Do all latches have either an OFF transistor, a `.NODESET`, or an `.IC`, on one side?
   - Do all series capacitors have a parallel resistance, or is `.OPTION DCSTEP` set?
2. Check your `.MODEL` statements.
   - Check all model parameter units. Use model printouts to verify actual values and units, because HSPICE multiplies some model parameters by scaling options.
   - Are sub-threshold parameters of MOS models, set with reasonable value (such as `NFS=1e11` for SPICE 1, 2, and 3 models, or `N0=1.0` for HSPICE BSIM1, BSIM2, and Level 28 device models)?
   - Do not set UTRA in MOS Level 2 models.
   - Are JS and JSW set in the MOS model for the DC portion of a diode model? A typical JS value is 1e-4A/M^2.
   - Are CJ and CJSW set, in MOS diode models?
   - Is weak-inversion NG and ND set in JFET/MESFET models?
   - If you use the MOS Level 6 LGAMMA equation, is UPDATE=1?
   - Make sure that DIODE models have non-zero values for saturation current, junction capacitance, and series resistance.
   - Use MOS ACM=1, ACM=2, or ACM=3 source and drain diode calculations, to automatically generate parasitics.

3. General remarks:
   - Ideal current sources require large values of `.OPTION GRAMP`, especially for BJT and MESFET circuits. Such circuits do not ramp up with the supply voltages, and can force reverse-bias conditions, leading to excessive nodal voltages.
   - Schmitt triggers are unpredictable for DC sweep analysis, and sometimes for operating points for the same reasons that oscillators and flip-flops are unpredictable. Use slow transient.
   - Large circuits tend to have more convergence problems, because they have a higher probability of uncovering a modeling problem.
   - Circuits that converge individually, but fail when combined, are almost guaranteed to have a modeling problem.
   - Open-loop op-amps have high gain, which can lead to difficulties in converging. Start op-amps in unity-gain configuration, and open them up in transient analysis, using a voltage-variable resistor, or a resistor with a large AC value (for AC analysis).
4. Check your options:
   - Remove all convergence-related options, and try first with no special `.OPTION` settings.
   - Check non-convergence diagnostic tables for non-convergent nodes. Look up non-convergent nodes in the circuit schematic. They are usually latches, Schmitt triggers, or oscillating nodes.
   - For stubborn convergence failures, bypass DC all together, and use `.TRAN` with UIC set. Continue transient analysis until transients settle out, then specify the `.OP` time, to obtain an operating point during the transient analysis. To specify an AC analysis during the transient analysis, add an `.AC` statement to the `.OP` time statement.
   - `SCALE` and `SCALM` scaling options have a significant effect on parameter values in both elements and models. Be careful with units.

Shorted Element Nodes

HSPICE or HSPICE RF disregards any capacitor, resistor, inductor, diode, BJT, or MOSFET, if all of its leads connect together. Simulation does not count the component in its component tally, and issues a warning:

```
** warning **
all nodes of element x:<name> are connected together
```

Inserting Conductance, Using DCSTEP

In a DC operating-point analysis, failure to include conductances in a capacitor model results in broken circuit loops (because a DC analysis opens all capacitors). This might not be solvable. If you include a small conductance in the capacitor model, the circuit loops are complete, and HSPICE or HSPICE RF can solve them.

Modeling capacitors as complete opens, can result in this error:

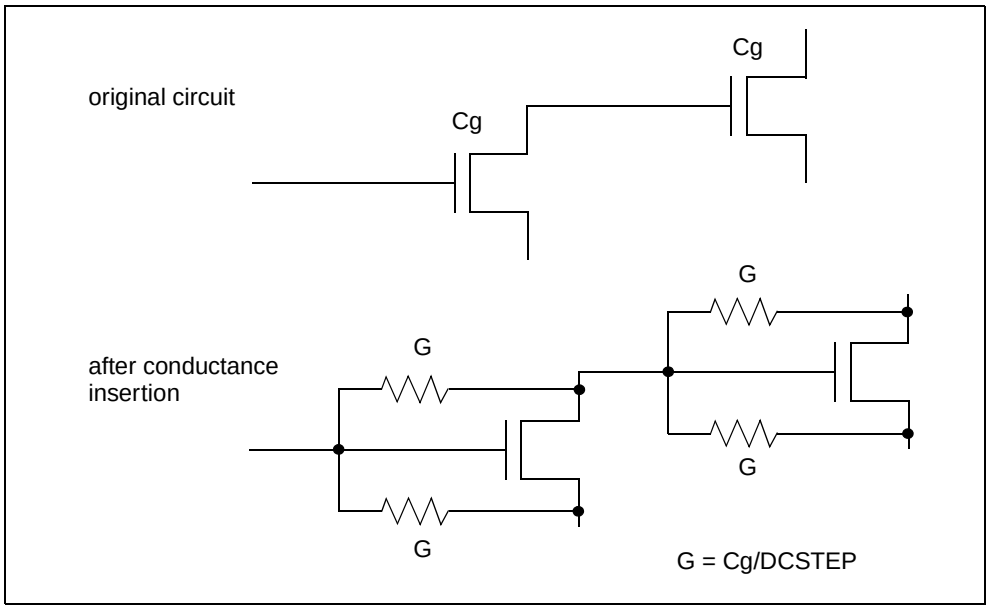
```
"No DC Path to Ground"
```

For a DC analysis, use `.OPTION DCSTEP`, to assign a conductance value to all capacitors in the circuit. `DCSTEP` calculates the value as:

```
conductance=capacitance/DCSTEP
```

In Figure 39 on page 307, HSPICE or HSPICE RF inserts conductance (G), in parallel with capacitance (C_g). This provides current paths around capacitances, in DC analysis.

Figure 39 Conductance Insertion



Floating-Point Overflow

If MOS conductance is negative or zero, HSPICE or HSPICE RF might have difficulty converging. An indication of this type of problem is a floating-point overflow, during matrix solutions. HSPICE or HSPICE RF detects floating-point overflow, and invokes the Damped Pseudo Transient algorithm (`CONVERGE=1`), to try to achieve DC convergence without requiring you to intervene. If `GMINDC` is 1.0e-12 or less when a floating-point overflows, HSPICE or HSPICE RF sets it to 1.0e-11.

Diagnosing Convergence Problems

Before simulation, HSPICE or HSPICE RF diagnoses potential convergence problems in the input circuit, and provides an early warning, to help you in debugging your circuit. If HSPICE or HSPICE RF detects a circuit condition that might cause convergence problems, it prints the following message into the output file:

```
"Warning: Zero diagonal value detected at node ( ) in
   equation solver, which might cause convergence problems.
   If your simulation fails, try adding a large resistor
   between node ( ) and ground."
```

Non-Convergence Diagnostic Table

If a circuit cannot converge, HSPICE or HSPICE RF automatically generates two printouts, called the diagnostic tables:

- Nodal voltage printout: Prints the names of all no-convergent node voltages, and the associated voltage error tolerances (tol).
- Element printout: Lists all non-convergent elements, and their associated element currents, element voltages, model parameters, and current error tolerances (tol).

To locate the branch current or nodal voltage that causes non-convergence, use the following steps:

1. Analyze the diagnostic tables. Look for unusually large values of branch currents, nodal voltages or tolerances.
2. After you locate the cause, use the `.NODESET` or `.IC` statements, to initialize the node or branch.

   If circuit simulation does not converge, HSPICE or HSPICE RF automatically generates a non-convergence diagnostic table, indicating:

   - The quantity of recorded voltage failures.
   - The quantity of recorded branch element failures.

     Any node in a circuit can create voltage failures, including *hidden* nodes (such as extra nodes that parasitic resistors create).
3. Check the element printout for the subcircuit, model, and element name for all parts of the circuit where node voltages or currents do not converge.

For example, Table 43 on page 309 identifies the xinv21, xinv22, xinv23, and xinv24 inverters, as problem sub-circuits in a ring oscillator. It also indicates that the p-channel transistors, in the xinv21, xinv22, xinv24 sub-circuits, are nonconvergent elements. The n-channel transistor of xinv23 is also a nonconvergent element.

The table lists voltages and currents for the transistors, so you can check whether they have reasonable values. The tolds, tolbd, and tolbs error tolerances indicate how close the element currents (drain to source, bulk to drain, and bulk to source) are, to a convergent solution. For tol variables, a value close to or below 1.0 is a convergent solution. In Table 43, the *tol* values that are around 100, indicate that the currents were far from convergence. The element current and voltage values are also shown (id, ibs, ibd, vgs, vds, and

vbs). Examine whether these values are realistic, and determine the transistor regions of operation.

Table 43 Subcircuit Voltage, Current, and Tolerance

subckt element model	**xinv21 21:mphc1 0:p1**	**xinv22 22:mphc1 0:p1**	**xinv23 23:mphc1 0:p1**	**xinv23 23:mnch1 0:n1**	**xinv24 24: mphc1 0:p1**
id	27.5809f	140.5646u	1.8123p	1.7017m	5.5132u
ibs	205.9804f	3.1881f	31.2989f	0.	200.0000f
ibd	0.	0.	0.	-168.7011f	0.
vgs	4.9994	-4.9992	69.9223	4.9998	-67.8955
vds	4.9994	206.6633u	69.9225	-64.9225	2.0269
vbs	4.9994	206.6633u	69.9225	0.	2.0269
vth	-653.8030m	-745.5860m	-732.8632m	549.4114m	-656.5097m
tolds	114.8609	82.5624	155.9508	104.5004	5.3653
tolbd	0.	0.	0.	0.	0.
tolbs	3.534e-19	107.1528m	0.	0.	0.

Traceback of Non-Convergence Source

To locate a non-convergence source, trace the circuit path for error tolerance. For example, in an inverter chain, the last inverter can have a very high error tolerance. If this is the case, examine the error tolerance of the elements that drive the inverter. If the driving tolerance is high, the driving element could be the source of non-convergence. However, if the tolerance is low, check the driven element as the source of non-convergence.

Examine the voltages and current levels of a non-convergent MOSFET to discover the operating region of the MOSFET. This information can flow to the location of the discontinuity in the model—for example, subthreshold-to-linear, or linear-to-saturation.

When considering error tolerances, check the current and nodal voltage values. If these values are extremely low, a relatively large number is divided by a very

small number. This produces a large calculation result, which can cause the non-convergence errors. To solve this, increase the value of the absolute-accuracy options.

Use the diagnostic table, with the DC iteration limit (`ITL1` option), to find the sources of non-convergence. When you increase or decrease `ITL1`, HSPICE or HSPICE RF prints output for the problem nodes and elements for a new iteration—that is, the last iteration of the analysis that you set in `ITL1`.

Solutions for Non-Convergent Circuits

Non-convergent circuits generally result from:

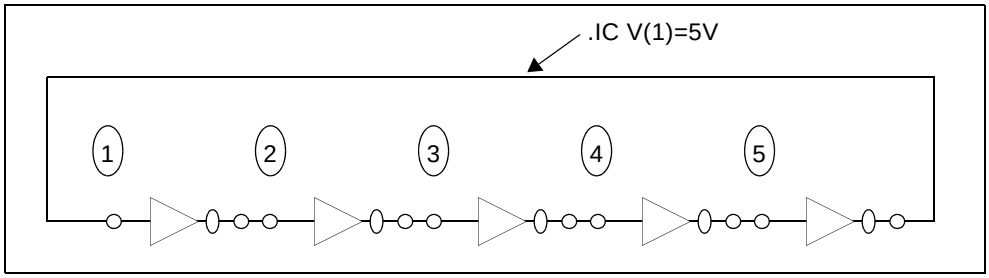
- Poor Initial Conditions
- Inappropriate Model Parameters
- PN Junctions (Diodes, MOSFETs, BJTs)

The following sections explain these conditions.

Poor Initial Conditions

Multi-stable circuits need state information, to guide the DC solution. You must initialize ring oscillators and flip-flops. These multi-stable circuits can either produce an intermediate forbidden state, or cause a DC convergence problem. To initialize a circuit, use the `.IC` statement, which forces a node to the requested voltage. Ring oscillators usually require you to set only one stage.

Figure 40 Ring Oscillator



The best way to set up the flip-flop is to use an `.IC` statement in the subcircuit definition.

Example

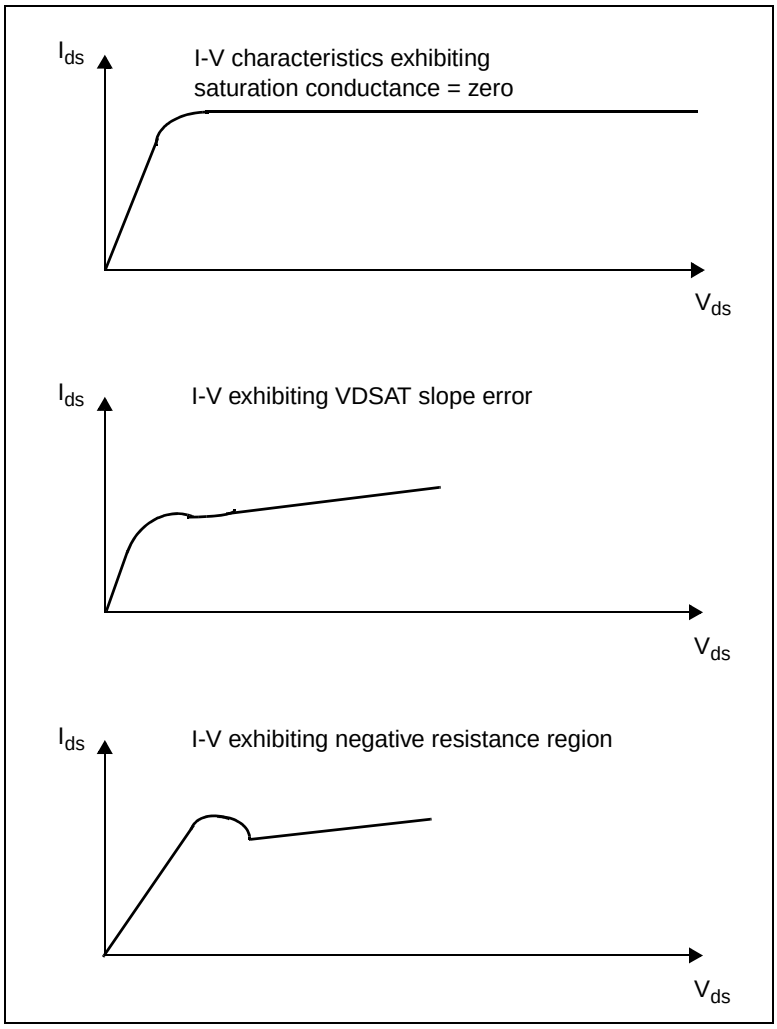
The following example sets the local `Qset` parameter to 0, and uses this value for the `.IC` statement, to initialize the Q latch output node. As a result, all latches have a default state of Q low. Setting `Qset` to vdd calls a latch, which overrides this state.

```
.subckt latch in Q Q/ d Qset=0
.ic Q=Qset
...
.ends
Xff data_in[1] out[1] out[1]/ strobe LATCH Qset=vdd
```

Inappropriate Model Parameters

If you impose non-physical model parameters, you might create a discontinuous IDS or capacitance model. This can cause an *internal timestep too small* error, during the transient simulation. The mosivcv.sp demonstration file shows IDS, VGS, GM, GDS, GMB, and CV plots for MOS devices. A sweep near threshold, from *Vth-0.5* V to *Vth+0.5* V (using a delta of 0.01 V), sometimes discloses a possible discontinuity in the curves.

Figure 41 Discontinuous I-V Characteristics



If simulation does not converge when you add a component or change a component value, then the model parameters are not appropriate or do not correspond to physical values they represent.

To locate the problem, follow these steps:

1. Check the input netlist file for non-convergent elements.

   Devices with a *TOL* value greater than 1, are non-convergent.

2. Find the devices at the beginning of the combined-logic string of gates that seem to start the non-convergent string.

3. Check the operating point of these devices very closely, to see what region they operate in.

   Model parameters associated with this region are probably inappropriate.

Circuit simulation uses single-transistor characterization, to simulate a large collection of devices. If a circuit fails to converge, the cause can be a single transistor, anywhere in the circuit.

PN Junctions (Diodes, MOSFETs, BJTs)

PN junctions found in diode, BJT, and MOSFET models, might exhibit non-convergent behavior, in both DC and transient analysis.

Example

PN junctions often have a high *off* resistance, resulting in an ill-conditioned matrix. To overcome this, use `.OPTION GMINDC` and `.OPTION GMIN` to automatically parallel every PN junction in a design, with a conductance.

Non-convergence can occur if you overdrive the PN junction. This happens if you omit a current-limiting resistor, or if the resistor has a very small value. In transient analysis, protection diodes are often temporarily forward-biased (due to the inductive switching effect). This overdrives the diode, and can result in non-convergence, if you omit a current-limiting resistor.

9

Transient Analysis

Describes how to use transient analysis to compute the circuit solution.

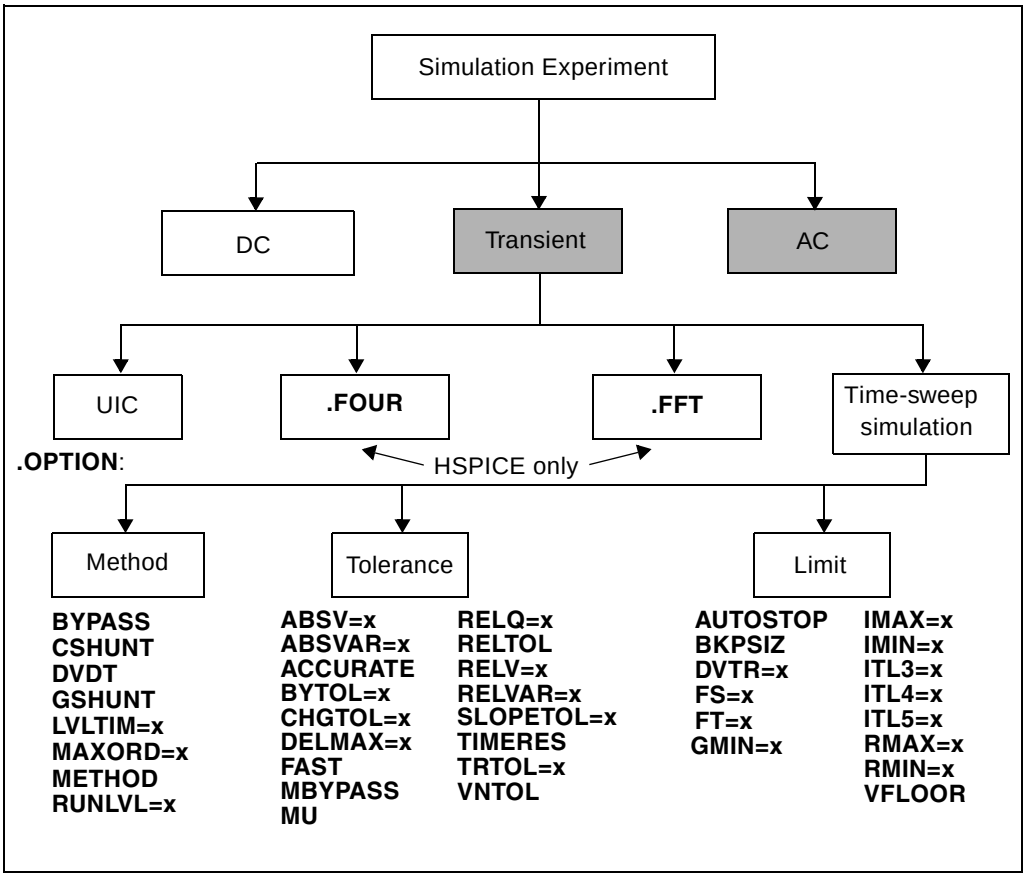
Transient analysis computes the circuit solution, as a function of time, over a time range specified in the `.TRAN` statement.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Command Reference*.

Simulation Flow

Figure 42 illustrates the simulation flow for transient analysis in Synopsys HSPICE and HSPICE RF.

Figure 42 Transient Analysis Simulation Flow



Overview of Transient Analysis

Transient analysis simulates a circuit at a specific time. *Some* of its algorithms, control options, convergence-related issues, and initialization parameters are different than those used in DC analysis. However, a transient analysis first performs a DC operating point analysis, unless you specify the UIC option in the `.TRAN` statement. Therefore, *most* DC analysis algorithms, control options, initialization issues, and convergence issues, also apply to transient analysis.

Unless you set the initial circuit operating conditions, some circuits (such as oscillators, or circuits with feedback) do not have stable operating point solutions. For these circuits, either:

- Break the feedback loop, to calculate a stable DC operating point, or
- Specify the initial conditions, in the simulation input.

If you include the UIC parameter in the .TRAN statement, HSPICE or HSPICE RF bypasses the DC operating point analysis. Instead, it uses node voltages, specified in an .IC statement, to start a transient analysis. For example, if a .IC statement sets a node to 5 V in, the value at that node for the first time point (time 0) is 5 V.

You can use the .OP statement to store an estimate of the DC operating point, during a transient analysis.

Example

In the following example, the UIC parameter (in the .TRAN statement) bypasses the initial DC operating point analysis. The .OP statement calculates the transient operating point (at t=20 ns), during the transient analysis.

```
.TRAN 1ns 100ns UIC
.OP 20ns
```

Although a transient analysis might provide a convergent DC solution, the transient analysis can still fail to converge. In a transient analysis, the internal timestep too small error message indicates that the circuit failed to converge. The cause of this convergence failure might be that stated initial conditions are not close enough to the actual DC operating point values. Use the commands in this chapter to help achieve convergence in a transient analysis.

Transient Analysis Output

```
.print tran ov1 [ov2 ... ovN]
.probe tran ov1 [ov2 ... ovN]
.measure tran measspec
.plot tran ov1 [ov2 ... ovN]
.graph tran ov1 [ov2 ... ovN]
```

HSPICE RF does not support .PLOT or .GRAPH.

The *ov1, ... ovN* output variables can include the following:

- V(n): voltage at node *n*.
- V(n1<,n2>): voltage between the *n1* and *n2* nodes.
- Vn(d1): voltage at *n*th terminal of the *d1* device.
- In(d1): current into *n*th terminal of the *d1* device.
- '*expression*': expression, involving the plot variables above

You can use wildcards (*), or as specified in the .hspicerf configuration file) to specify multiple output variables in a single command. Output is affected by .OPTION POST or .OPTION PROBE, SIM_DELTAI, and SIM_DELTAV.

Parameter	Description
*.print	Writes the output from the .PRINT statement to a *.print file. HSPICE does not generate a *.print# file. ■ The header line contains column labels. ■ The first column is time. ■ The remaining columns represent the output variables specified with .PRINT. ■ Rows that follow the header contain the data values for simulated time points.
*.tr#	Writes output from the .PROBE, .PRINT, .PLOT, .GRAPH, or .MEASURE statement to a *.tr# file.

Transient Analysis of an RC Network

Follow these steps to run a transient analysis of a RC network with a pulse source, a DC source, and an AC source:

1. Type the following netlist into a file named quickTRAN.sp.

```
A SIMPLE TRANSIENT RUN
.OPTION LIST NODE POST
.OP
.TRAN 10N 2U
.PRINT TRAN V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1 PULSE 0 5 10N 20N 20N 500N 2U
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

   This example is based on demonstration netlist quickTRAN.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
A SIMPLE TRANSIENT RUN
.OPTION LIST NODE POST
.OP
.TRAN 10N 2U
.PRINT TRAN V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1 PULSE 0 5 10N 20N 20N 500N 2U
```

```
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

Note:

The V1 source specification includes a pulse source. For the syntax of pulse sources and other types of sources, see Chapter 5, Sources and Stimuli.

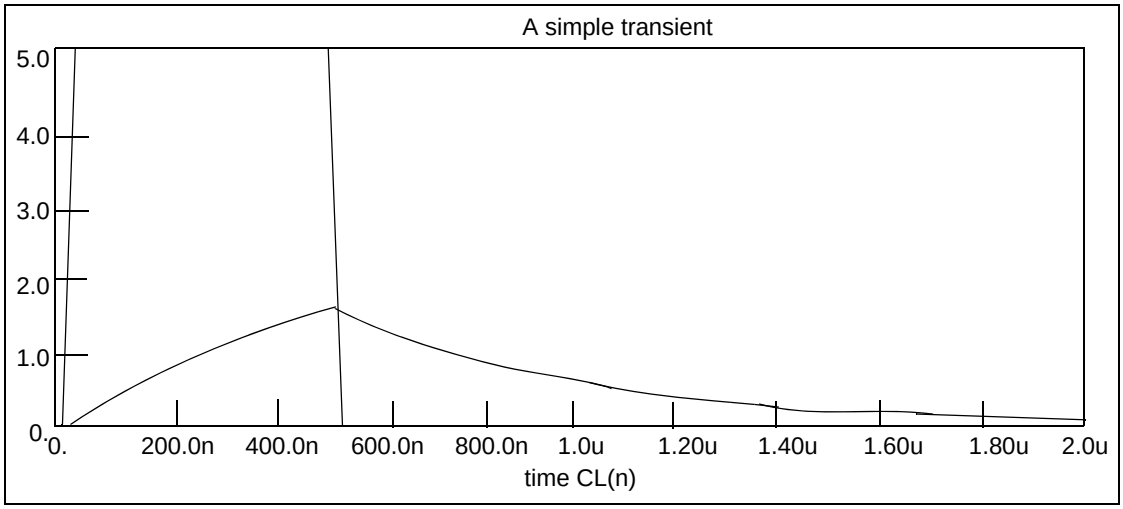
2. To run HSPICE, type the following:

```
hspice quickTRAN.sp > quickTRAN.lis
```

3. To examine the simulation results and status, use an editor and view the .lis and .st0 files.
4. Run AvanWaves and open the .sp file.
5. To view the waveform, select the *quickTRAN.tr0* file from the Results Browser window.
6. Display the voltage at nodes 1 and 2 on the x-axis.

Figure 43 shows the waveforms.

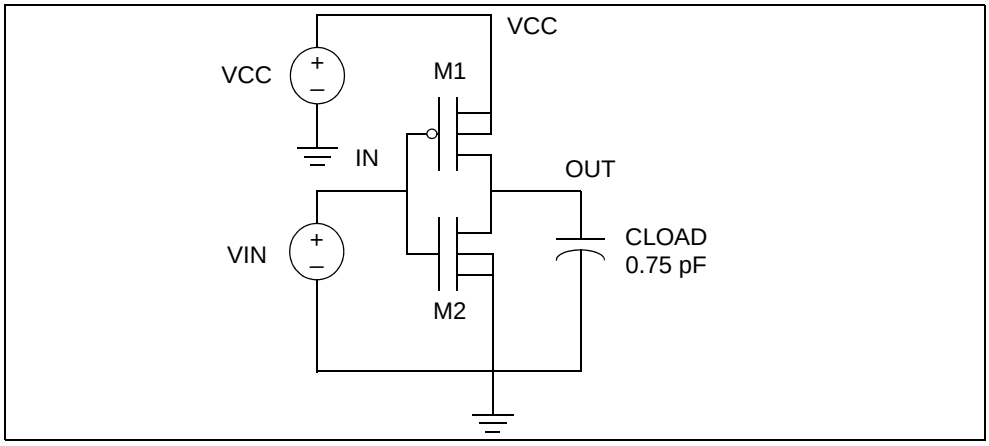
Figure 43 Voltages at RC Network Circuit Node 1 and Node 2



Transient Analysis of an Inverter

As a final example, you can analyze the behavior of the simple MOS inverter shown in Figure 44.

Figure 44 MOS Inverter Circuit



Follow these steps to analyze this behavior:

1. Type the following netlist data into a file named quickINV.sp.

```
Inverter Circuit
.OPTION LIST NODE POST
.TRAN 200P 20N
.PRINT TRAN V(IN) V(OUT)
M1 OUT IN VCC VCC PCH L=1U W=20U
M2 OUT IN 0 0 NCH L=1U W=20U
VCC VCC 0 5
VIN IN 0 0 PULSE .2 4.8 2N 1N 1N 5N 20N
CLOAD OUT 0 .75P
.MODEL PCH PMOS LEVEL=1
.MODEL NCH NMOS LEVEL=1
.END
```

   You can find the complete netlist for this example in directory $<installdir>/demo/hspice/apps/quickINV.sp.

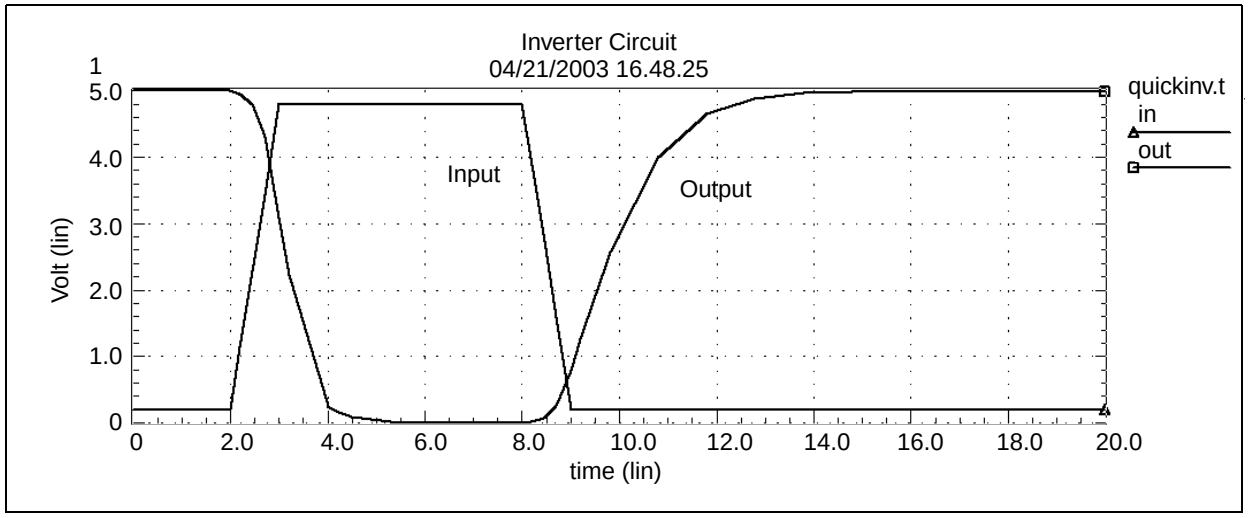
2. To run HSPICE, type the following:

   `hspice quickINV.sp > quickINV.lis`

3. Use AvanWaves to examine the voltage waveforms, at the inverter IN and OUT nodes.

Figure 45 shows the waveforms.

Figure 45 Voltage at MOS Inverter Node 1 and Node 2



Using the .BIASCHK Statement

The `.BIASCHK` statement can monitor the voltage bias, current, device-size, expression and region during transient analysis, and reports:

- Element name
- Time
- Terminals
- Bias that exceeds the limit
- Number of times the bias exceeds the limit for an element

For the syntax and description of this statement, see the .BIASCHK command in the *HSPICE Command Reference*.

HSPICE or HSPICE RF saves the information as both a warning and a BIASCHK summary in the **.lis* file. You can use this command only for active elements and capacitors.

You can also use `.OPTION BIASFILE` and `.OPTION BIAWARN` with a `.BIASCHK` statement.

The following limitations apply to the .BIASCHK statement:

- .BIASCHK is only supported for diode, jfet, nmos, pmos, bjt, and c models, as well as subcircuits.
- For a device-size check, only W and L MOSFET models are supported.
- Wildcards in element and model names, and except definitions are supported but not in expressions.

Data Checking Methods

Four methods are available to check the data with the .BIASCHK command:

- Limit and noise method
- Maximum method
- Minimum method
- Region method

  Note:

  The region method of data checking is only supported in MOSFET models.

Limit and Noise Method

For a transient simulation using the limit and noise method to check the data, use the following syntax:

For local_max

`v(tn-1) > limit_value`

The bias corresponds anyone of the following two conditions:

- `v(tn-1) > v(tn) && v(tn-1) >= v(tn-2)`
- `v(tn-1) >= v(tn) && v(tn-1) > v(tn-2)`

local_min: The minimum bias after the time last local max occurs.

During a transient analysis, the local_max is recorded if it is greater than the limit. In the summary reported after transient analysis, the local_max(current) is replaced with the local_max(next) when the following comparison is true:

local_max(current) - local_min < noise && local_max(next) - local_min < noise && local_max(current) < local_max(next)

At the end of the simulation, all `local_max` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is greater than the `limit_value` you specify.

Maximum Method

For a transient simulation using the maximum method to check the data, use the following syntax:

For `local_max`:

`v(tn-1) > max_value`

The bias corresponds any one of the following two conditions:

- `v(tn-1) > v(tn) && v(tn-1) >= v(tn-2)`
- `v(tn-1) >= v(tn) && v(tn-1) > v(tn-2)`

During a transient analysis, all `local_max` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is greater than `max_value` you specify.

Minimum Method

For a transient simulation using the minimum method to check the data, use the following syntax:

For `local_min`:

`v(tn) < min_value`

The bias corresponds any one of the following two conditions:

- `v(tn-1) < v(tn) && v(tn-1) <= v(tn-2)`
- `v(tn-1) <= v(tn) && v(tn-1) < v(tn-2)`

During a transient analysis, all `local_min` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is smaller than `min_value` you specify.

Region Method

This method is only for MOSFET models. Three regions exist:

- cutoff
- linear
- saturation

When the specified transistor enters and exits during transient analysis, the specified region is reported.

Example

The following example is a netlist that uses the .BIASCHK command for a transient simulation. This example is based on demonstration netlist biaschk.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
* Test Case
* Transient simulation
.Tran 1n 8n
.Options Post NoMod biasfile = 'result'
.biaschk nmos terminal1=nd terminal2=nb max=.006
.biaschk nmos terminal1=nb terminal2=ns limit=.006
+ noise=.005
.biaschk mos region=saturation region=linear mname=nmos
+ mname=pmos
.biaschk c terminal1=n1 max=.1
+ name='c10','c14','c15','c8','c7'
.biaschk 'v(net27)-v(net25)' min=.1e-10 max=1 simu=all
+ $monitor=op monitor=dc monitor=tr
.biaschk 'v(net25)-v(0)' min=.1e-5 max='0.1*v(net31)'
+ simu=tran
.biaschk mos terminal1=nb monitor = l mname=nmos
+ simulation = tran min=1u
.biaschk mos terminal1=nb monitor = w mname=nmos
+ simulation = tran min =10u
.biaschk mos terminal1=nb monitor = i mname=nmos simu = op
+ simu=tran max = 1m
.biaschk 'v(net25)' min=2.5 tstart=2n tstop=6n $autostop
v28 data gnd PWL 0s 5v 1n 5v 2n 0v
v27 clock gnd PWL 0s 0v 3n 0v 4n 5v
.model nmos nmos level=2
.model pmos pmos level=2
.Global vdd gnd
.subckt XGATE control in n_control out
m0 in n_control out vdd pmos l=1.2u w=3.4u
m1 in control out gnd nmos l=1.2u w=3.4u
.ends
.subckt INV in out wp=9.6u wn=4u l=1.2u
```

```
mb2 out in gnd gnd nmos l=l w=wn
mb1 out in vdd vdd pmos l=l w=wp
.ends
.subckt DFF c d nc nq
Xi64 nc net46 c net36 XGATE
Xi66 nc net38 c net39 XGATE
Xi65 c nq nc net36 XGATE
Xi62 c d nc net39 XGATE
Xi60 net722 nq INV
Xi61 net46 net38 INV
Xi59 net36 net722 INV
Xi58 net39 net46 INV
c20 net36 gnd c=17f
c15 net39 gnd c=15f
c12 net46 gnd c=25f
c4 nq gnd c=25f
c3 net722 gnd c=19f
c16 net38 gnd c=16f
.ends
*-----------------------------------------------------
* Main Circuit Netlist:
*-----------------------------------------------------------
v14 vdd gnd dc=5
c10 vdd gnd c=35f
c15 d_output gnd c=21f
c12 dff_nq gnd c=11f
c11 net31 gnd c=42f
c14 net27 gnd c=34f
c13 net25 gnd c=41f
c8 clock gnd c=5f
c7 data gnd c=7f
Xi3 net25 net31 net27 dff_nq DFF l=1u wn=3.8u wp=10u
Xi6 data net31 INV
Xi5 net25 net27 INV
Xi4 clock net25 INV
Xi2 dff_nq d_output INV wp=26.4u wn=10.6u
.print v(clock) v(net25)
.op
.end
```

Transient Control Options

Method, tolerance, and limit options in this section modify the behavior of transient analysis integration routines. Delta is the internal timestep. TSTEP and TSTOP are the step and stop values in the .TRAN statement. Asterisk denotes an option only available in HSPICE RF (not supported in HSPICE).

Table 44 Transient Control Options, Arranged by Category

Method	Tolerance		Limit
BYPASS	ABSH	RELVAR	AUTOSTOP
CSHUNT	ABSV	SIM_ACCURACY*	BKPSIZ
DVDT	ABSVAR	SIM_DELTAI*	DELMAX
GSHUNT	ACCURATE	SIM_DELTAV*	DVTR
INTERP	BYTOL	SLOPETOL	FS
ITRPRT	CHGTOL	TIMERES	FT
LVLTIM	DI	TRTOL	GMIN
MAXORD	FAST	VNTOL	ITL3
METHOD	MBYPASS		ITL4
POST*	MAXAMP		ITL5
PROBE*	MU		RMAX
PURETP	RELH		RMIN
SIM_ORDER*	RELI		VFLOOR
RUNLVL	RELQ		
SIM_TRAP*	RELTOL		
TRCON	RELV		

Matrix Manipulation Options

After HSPICE generates individual linear elements in an input netlist, it constructs linear equations for the matrix. You can set variables that affect how HSPICE constructs and solves the matrix equation, including `.OPTION PIVOT` and `.OPTION GMIN` (HSPICE RF does not support these options). `GMIN` places a variable in the matrix, so the matrix does not become *ill-conditioned*.

`.OPTION PIVOT` selects a pivoting method, which reduces simulation time, and assists in DC and transient convergence. Pivoting reduces errors, resulting from elements in the matrix that are widely different in magnitude. `PIVOT` searches the matrix, to find the largest element value, and uses this value as the pivot.

Simulation Speed and Accuracy

Convergence is the ability to solve a set of circuit equations within specified tolerances and within a specified number of iterations. In numerical circuit simulation, you can specify relative and absolute accuracy for the circuit solution. The simulator iteration algorithm attempts to converge to a solution that is within these set tolerances. If consecutive simulations achieve results within the specified accuracy tolerances, circuit simulation has converged. How quickly the simulator converges, is often a primary concern to a designer—especially for preliminary design trials. So designers willingly sacrifice some accuracy for simulations that converge quickly.

Simulation Speed

HSPICE or HSPICE RF can substantially reduce the computer time needed to solve complex problems. Use the following options to alter the internal algorithms to increase simulation efficiency.

- `.OPTION FAST` – sets additional options, which increase simulation speed, with minimal loss of accuracy
- `.OPTION AUTOSTOP` – terminates the simulation, after completing all `.MEASURE` statements. This is of special interest, when testing corners.

Simulation Accuracy

In HSPICE or HSPICE RF, the default control option values aim for superior accuracy, within an acceptable amount of simulation time. The control options and their default settings (to maximize accuracy) are:

DVDT=4 LVLTIM=1 RMAX=5 SLOPETOL=0.75

FT=FS=0.25 BYPASS=1 BYTOL=MBYPASS x VNTOL=0.100m

Note:

`BYPASS` is on (set to 1), only when `DVDT=4`. For other `DVDT` settings, `BYPASS` is off (0). The `SLOPETOL` value is 0.75, only if `DVDT=4` and `LVLTIM=1`. For all other values of `DVDT` or `LVLTIM`, `SLOPETOL` defaults to 0.5.

Timestep Control for Accuracy

The DVDT control option selects the timestep control algorithm. For a description of the relationships between DVDT and other control options, see Selecting Timestep Control Algorithms on page 333.

The DELMAX control option also affects simulation accuracy. DELMAX specifies the maximum timestep size. If you do not set .OPTION DELMAX, HSPICE or HSPICE RF computes a DELMAX value. Factors that determine the computed DELMAX value are:

- .OPTION RMAX and .OPTION FS.
- Breakpoint locations for a PWL source.
- Breakpoint locations for a PULSE source.
- Smallest period for a SIN source.
- Smallest delay for a transmission line component.
- Smallest ideal delay for a transmission line component.
- TSTEP value, in a .TRAN analysis.
- Number of points, in an FFT analysis (HSPICE only).

Use the FS and RMAX control options, to control the DELMAX value.

- .OPTION FS, which defaults to 0.25, scales the breakpoint interval in the DELMAX calculation.
- .OPTION RMAX defaults to 5 (if DVDT=4 and LVLTIM=1), and scales the TSTEP (timestep) size in the DELMAX calculation.

For circuits that contain oscillators or ideal delay elements, use .OPTION DELMAX, to set DELMAX to one-hundredth of the period or less.

.OPTION ACCURATE tightens the simulation options to output the most accurate set of simulation algorithms and tolerances. If you set ACCURATE to 1, HSPICE or HSPICE RF uses these control options:

Table 45 Control Option Settings When ACCURATE=1

DVDT=2	RELVAR=0.2LVL	BYPASS=0	FT=FS=0.2	RELMOS=0.01
BYTOL=0	TIM=3	ABSVAR=0.2	RMAX=2	SLOPETOL=0.5

Models and Accuracy

Simulation accuracy depends on the sophistication and accuracy of the models you use. Advanced MOS, BJT, and GaAs models provide superior results for critical applications.

The following model types increase simulation accuracy:

- Algebraic models, which describe parasitic interconnect capacitances as a function of the width of the transistor. The wire model extension of the resistor can model the metal, diffusion, or poly interconnects, to preserve the relationship between the physical layout and the electrical property.
- The `ACM` parameter in MOS models, which calculates source and drain junction parasitic defaults. `ACM` equations calculate:
  - size of the bottom wall
  - length of the sidewall diodes
  - length of a lightly doped structure.

  SPICE defaults do not calculate the junction diode. Specify `AD`, `AS`, `PD`, `PS`, `NRD`, and `NRS` to override the default calculations.
- `CAPOP=4` models the most advanced charge conservation, non-reciprocal gate capacitances. HSPICE or HSPICE RF calculates the gate capacitors and overlaps, from the IDS model for LEVEL 49 or 53. Simulation ignores the `CAPOP` parameter; instead, use the `CAPMOD` model parameter, with a reasonable value.

Guidelines for Choosing Accuracy Options

Use `.OPTION ACCURATE` for:

- Analog or mixed signal circuits.
- Circuits with long time constants, such as RC networks.
- Circuits with ground bounce.

Use the default options (`DVDT=4`) for:

- Digital CMOS.
- CMOS cell characterization.
- Circuits with fast moving edges (short rise and fall times).

For ideal delay elements, use one of the following:

- ACCURATE.
- DVDT=3.
- DVDT=4. If the minimum pulse width of a signal is less than the minimum ideal delay, set DELMAX to a value smaller than the minimum pulse width.

Numerical Integration Algorithm Controls

In HSPICE transient analysis, you can select one of three options to convert differential terms into algebraic terms:

- Gear
- Backward-Euler
- Trapezoidal

Gear algorithm:

```
.OPTION METHOD=GEAR
```

Backward-Euler:

```
.OPTION METHOD=GEAR MU=0
```

Trapezoidal algorithm (default):

```
.OPTION METHOD=TRAP
```

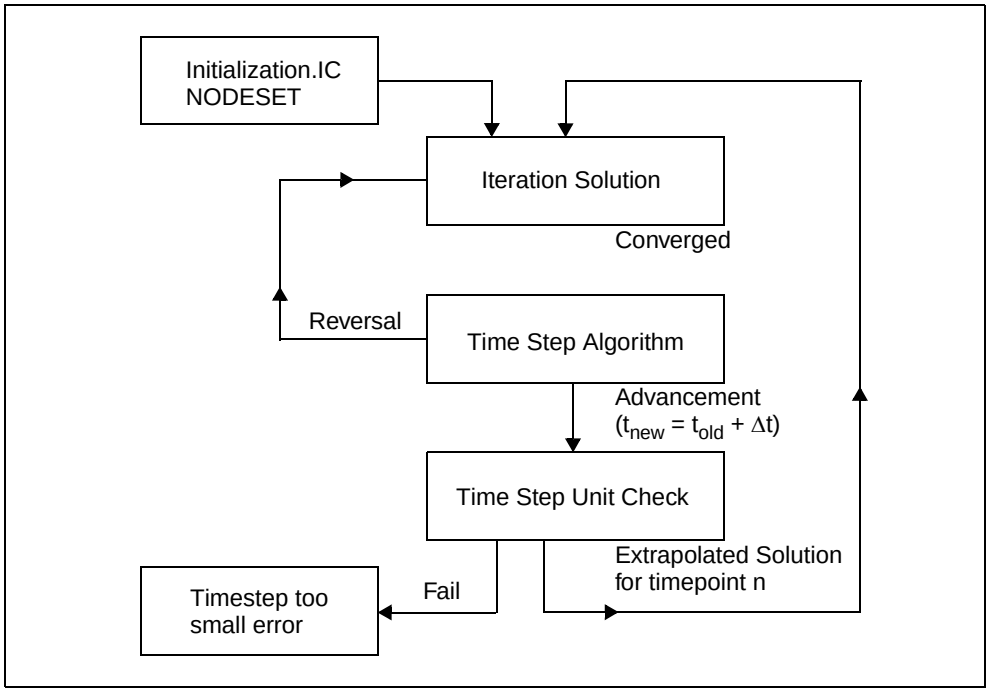
Each algorithm has advantages and disadvantages. Ideally, the trapezoidal is the preferred algorithm overall, because of its highest accuracy level and lowest simulation time.

However, selecting the appropriate algorithm for convergence is not always that easy or ideal. Which algorithm you select, largely depends on the type of circuit, and its associated behavior when you use different input stimuli.

Gear and Trapezoidal Algorithms

The algorithm that you select, automatically sets the timestep control algorithm. In HSPICE, if you select the GEAR algorithm (including Backward-Euler), the timestep control algorithm defaults to the truncation timestep algorithm. However, if you select the trapezoidal algorithm, the DVDT algorithm is the default. To change these HSPICE defaults, use the timestep control options.

Figure 46 Time Domain Algorithm



The trapezoidal algorithm can cause computational oscillation—that is, oscillation that the algorithm itself causes, not oscillation from the circuit design. This also produces an unusually long simulation time. If this occurs in inductive circuits (such as switching regulators), use the `GEAR` algorithm.

If transient analysis fails to converge using `.OPTION METHOD=TRAP` and `DVDT` timesteps (for example, due to trapezoidal oscillation), and HSPICE reports an *internal timestep too small* error, HSPICE then starts the *autoconvergence* process by default. This process sets `.OPTION METHOD=GEAR` and `LVLTIM=2`, and uses the Local Truncation Error (LTE) timestep algorithm. HSPICE then runs another transient analysis, to automatically obtain convergent results.

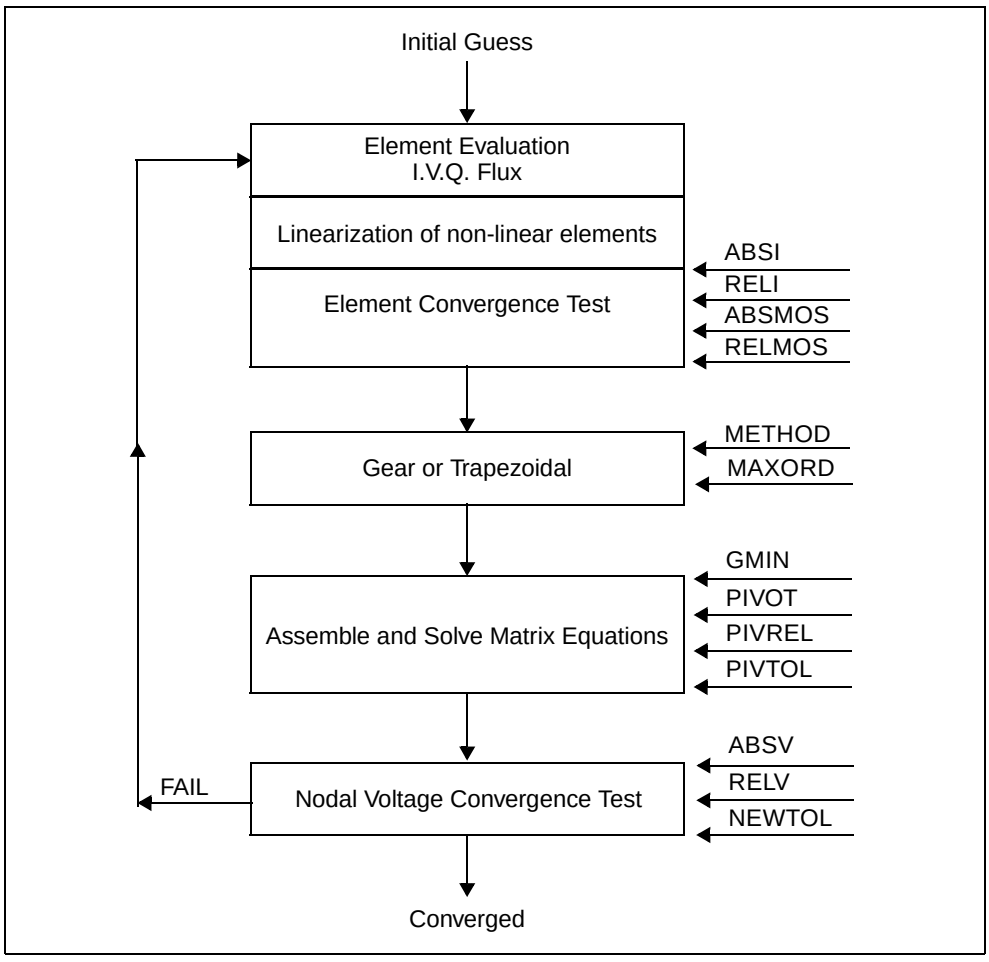
To manually improve on autoconvergence results, or if autoconvergence fails to converge, you can do either of the following:

- Set `.OPTION METHOD=GEAR` in the netlist, and try to obtain convergent results directly.

  To improve accuracy or speed, you can adjust `TSTEP` in a `.TRAN` statement, or in transient control options (such as `RMAX`, `RELQ`, `CHGTOL`, or `TRTOL`).

- Set `.OPTON METHOD=TRAP` in the netlist, then manually adjust `TSTEP` and the relevant control options (such as `CSHUNT` or `GSHUNT`).

Figure 47 Iteration Algorithm



Numerical Integration Algorithm Controls (HSPICE RF)

The numerical integration algorithm control in HSPICE RF is described in "RF Numerical Integration Algorithm Control" in the *HSPICE RF Manual*.

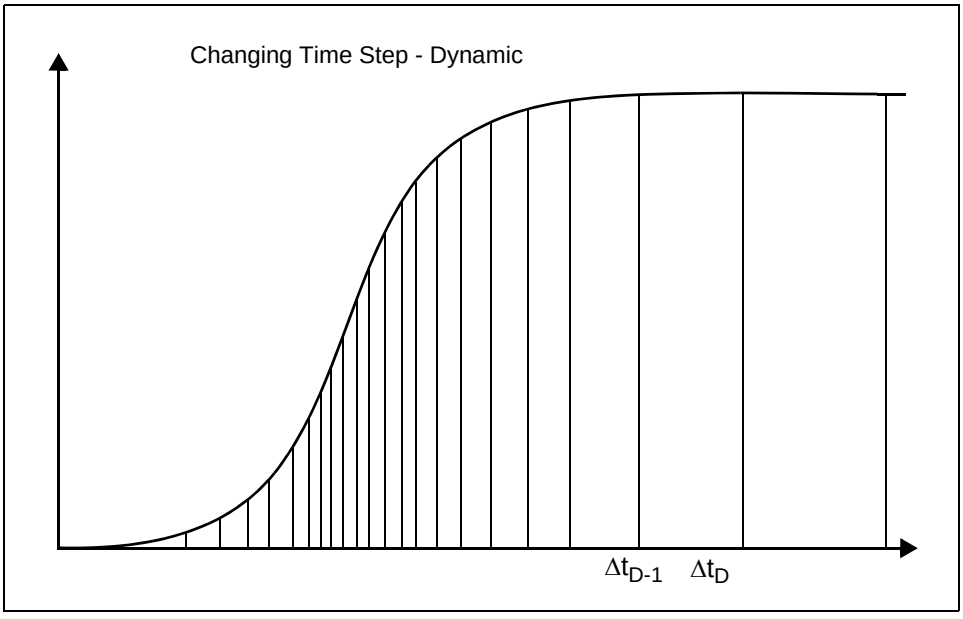
Selecting Timestep Control Algorithms

In HSPICE or HSPICE RF, you can select one of three dynamic timestep-control algorithms:

- Iteration Count Dynamic Timestep
- Local Truncation Error Dynamic Timestep
- DVDT Dynamic Timestep

Each algorithm uses a dynamically-changing timestep, which increases the accuracy of simulation, and reduces the simulation time. To do this, simulation varies the value of the timestep, over the transient analysis sweep, depending on the stability of the output. Dynamic timestep algorithms increase the timestep value when internal nodal voltages are stable, and decrease the timestep value when nodal voltages change quickly.

Figure 48 Internal Variable Timestep



The LVLTIM option selects the timestep algorithm:

- LVLTIM=0 selects the iteration count algorithm.
- LVLTIM=1 selects the DVDT timestep algorithm, and the iteration count algorithm. To control operation of the timestep control algorithm, set the DVDT control option. For LVLTIM=1 and DVDT=0, 1, 2, or 3, the algorithm does not use timestep reversal. For DVDT=4, the algorithm uses timestep reversal.

  For more information about the DVDT algorithm, see DVDT Dynamic Timestep on page 335.
- LVLTIM=2 selects the truncation timestep algorithm, and the iteration count algorithm (with reversal).
- LVLTIM=3 selects the DVDT timestep algorithm (with timestep reversal), and the iteration count algorithm. For LVLTIM=3 and DVDT=0, 1, 2, 3, or 4, the algorithm uses timestep reversal.

If HSPICE or HSPICE RF starts the autoconvergence process, it sets LVLTIM=2.

Iteration Count Dynamic Timestep

The simplest dynamic timestep algorithm is the iteration count algorithm. The control options that control this algorithm are .OPTION IMAX and .OPTION IMIN.

Local Truncation Error Dynamic Timestep

The local truncation error (LTE) timestep algorithm uses a Taylor-series approximation to calculate the next timestep for a transient analysis. This algorithm uses the allowed LTE to calculate an internal timestep.

- If the calculated timestep is smaller than the current timestep, HSPICE or HSPICE RF sets back the timepoint (timestep reversal), and uses the calculated timestep to increment the time.
- If the calculated timestep is larger than the current timestep, then HSPICE or HSPICE RF does not reverse the timestep. The next timepoint uses a new timestep.

To select the LTE timestep algorithm, set LVLTIM=2 or METHOD=GEAR. The control options available with the algorithm for local truncation error, are:

```
TRTOL (default=7)
CHGTOL (default=1e-15)
RELQ (default=0.01)
```

For some circuits (such as magnetic core circuits), GEAR and LTE provide more accurate result than TRAP. You can use this method with circuits containing inductors, diodes, BJTs (even Level 4 and above), MOSFETs, or JFETs.

DVDT Dynamic Timestep

To select DVDT dynamic timestep algorithm, set the LVLTIM option to 1 or 3.

- If you set LVLTIM=1, the DVDT algorithm does not use timestep reversal. HSPICE or HSPICE RF saves the results for the current timepoint, and uses a new timestep for the next timepoint.
- If you set LVLTIM=3, the algorithm uses timestep reversal. If the results do not converge at a specified iteration, HSPICE or HSPICE RF ignores the results of the current timepoint, sets back the time by the old timestep, and then uses a new timestep. Therefore, LVLTIM=3 is more accurate, and more time-consuming, than LVLTIM=1.

  This algorithm uses different tests, to decide whether to reverse the timestep, depending on how you set the DVDT control option.
- For DVDT=0, 1, 2, or 3, the decision is based on the SLOPETOL control option.
- For DVDT=4, the decision is based on how you set the SLOPETOL, RELVAR, and ABSVAR control options.

The DVDT algorithm calculates the internal timestep, based on the rate of nodal voltage changes.

- For circuits with rapidly-changing nodal voltages, the DVDT algorithm uses a small timestep.
- For circuits with slowly-changing nodal voltages, the DVDT algorithm uses larger timesteps.

The DVDT=4 setting selects a timestep control algorithm for non-linear node voltages. If you set the LVLTIM option to either 1 or 3, then DVDT=4 also uses timestep reversals. To measure non-linear node voltages, HSPICE or HSPICE

RF measures changes in slopes of the voltages. If the change in slope is larger than the SLOPETOL control setting, simulation reduces the timestep by the factor set in the FT control option. The FT option defaults to 0.25.

HSPICE or HSPICE RF sets the SLOPETOL value to 0.75 for LVLTIM=1, and to 0.50 for LVLTIM=3. Reducing the value of SLOPETOL increases simulation accuracy, but also increases simulation time.

- For LVLTIM=1, SLOPETOL and FT control simulation accuracy.
- For LVLTIM=3, the RELVAR and ABSVAR control options also affect the timestep, and therefore affect the simulation accuracy.

Use .OPTION RELVAR and .OPTION ABSVAR with the DVDT option to improve simulation time or accuracy. For faster simulation time, increase RELVAR and ABSVAR (but this might decrease accuracy).

Note:

> If you need backward compatibility, use these options. Setting .OPTION DVDT=3 automatically sets all of these values.

```
LVLTIM=1  RMAX=2  SLOPETOL=0.5
FT=FS=0.25  BYPASS=0  BYTOL=0.050
```

Timestep Controls in HSPICE

The RMIN, RMAX, FS, FT, and DELMAX control options define the minimum and maximum internal timestep for the DVDT dynamic timestep algorithm. If the timestep is below the minimum, program execution stops. For example, if the timestep becomes less than the minimum internal timestep (defined as TSTEP*RMIN), HSPICE reports an *internal timestep too small* error.

Note:

> TSTEP is the time increment set in the .TRAN statement. RMIN is the minimum timestep coefficient. Default is 1e-9.

If you set .OPTION DELMAX, HSPICE uses DVDT=0. If you do not specify .OPTION DELMAX, then HSPICE computes a DELMAX value. For DVDT=0, 1, or 2, the maximum internal timestep is:

```
min[(TSTOP/50), DELMAX, (TSTEP*RMAX)]
```

The TSTOP time is the transient sweep range, as set in the .TRAN statement.

One exception is in the autospeedup process. When dealing with large non-linear circuit with very big TSTOP or TSTEP values (for example, .TRAN 1n 1),

HSPICE might activate autospeedup. This process automatically sets RMAX to a bigger value, and sets the maximum internal timestep to:

```
min[(TSTOP/20),(TSTEP*RMAX)]
```

Set TRCON=-1 to disable autospeedup. You can then adjust TSTEP and RMAX, to balance accuracy and speed.

In circuits with piecewise linear (PWL) transient sources, then .OPTION SLOPETOL also affects the internal timestep. A PWL source, with a large number of voltage or current segments, contributes a correspondingly-large number of entries to the internal breakpoint table. The number of breakpoint table entries contributes to the internal timestep control.

If the difference in the slope for consecutive segments of a PWL source, is less than the SLOPETOL value, then HSPICE ignores the breakpoint table entry for the point between the segments. For a PWL source, with a signal that changes value slowly, ignoring its breakpoint table entries can help reduce the simulation time. Data in the breakpoint table is a factor in the internal timestep control, so setting a high SLOPETOL reduces the number of usable breakpoint table entries, which reduces the simulation time.

Effect of TSTEP on Timestep Size Selection

HSPICE's timestep size selection is affected by:

- voltage, current, and charge tolerances
- value of the .TRAN statement TSTEP argument
- value of the RMAX option
- settings of the timestep control method options such as RUNLVL, LVLTIM, and DVDT.

The affect of TSTEP and RMAX depend on the timestep control method in use.

- If RUNLVL=0, HSPICE never takes timesteps larger than TSTEP*RMAX. The size of the timestep is controlled by voltage, current, and charge tolerances, and LVLTIM/DVDT, but in any case, the step size is never allowed to exceed TSTEP*RMAX.
- If RUNLVL>0, HSPICE is allowed to take timesteps larger than TSTEP*RMAX. The size of the timestep is controlled by voltage, current, and charge tolerances. However, these tolerance values are affected by TSTEP*RMAX, so that smaller TSTEP values do result in tighter tolerances and therefore, smaller timesteps.

  Compared with RUNLV=0, this tends to result in larger timesteps and faster simulation speeds, especially in regions with flat or slowly varying waveforms.

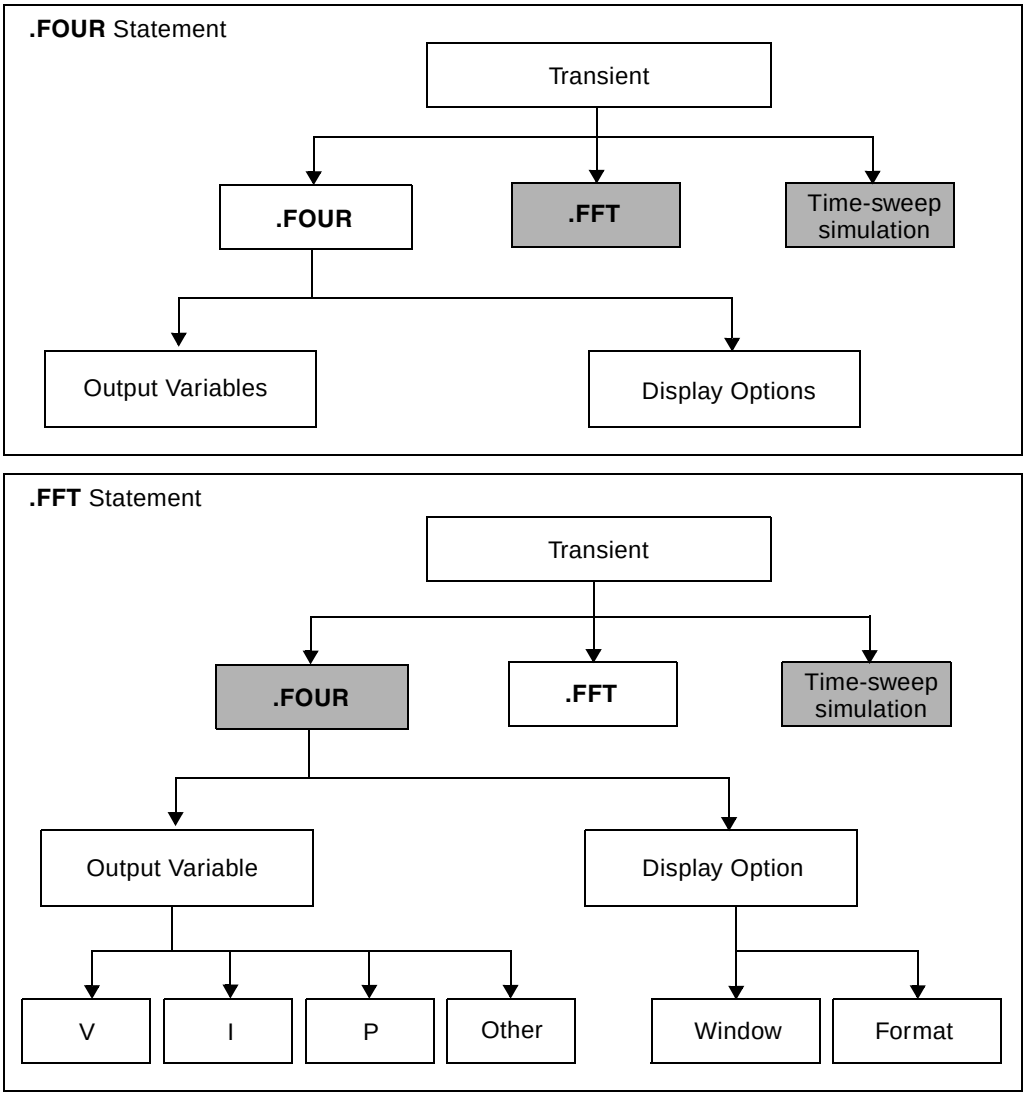
Timestep Controls in HSPICE RF

The timestemp controls in HSPICE RF are described in "RF Transient Analysis Accuracy Control" in the *HSPICE RF Manual*.

Fourier Analysis

This section describes the Fourier and FFT Analysis flow for HSPICE.

Figure 49 Fourier and FFT Analysis



HSPICE provides two different Fourier analyses, but HSPICE RF does not support either type of Fourier analysis:

- `.FOUR` is the same as is available in SPICE 2G6: a standard, fixed-window analysis tool. The `.FOUR` statement performs a Fourier analysis, as part of the transient analysis.
- `.FFT` is a much more flexible Fourier analysis tool. Use it for analysis tasks that require more detail and precision.

Accuracy and DELMAX

For better accuracy, set small values for `.OPTION RMAX` or `.OPTION DELMAX`. For maximum accuracy, set `.OPTION DELMAX` to (period/500). For circuits with very high resonance factors (high-Q circuits, such as crystal oscillators, tank circuits, and active filters), set `DELMAX` to less than (period/500).

Fourier Equation

The total harmonic distortion is the square root of the sum of the squares, of the second through ninth normalized harmonic, times 100, expressed as a percent:

$$THD = \frac{1}{R1} \cdot \left(\sum_{m=2}^{9} R_m^2 \right)^{1/2} \cdot 100\%$$

This interpolation can result in various inaccuracies.

If the transient analysis runs at intervals longer than 1/(501*f), then the frequency response of the interpolation dominates the power spectrum. Furthermore, this interpolation does not derive an error range for the output.

The following equation calculates the Fourier coefficients:

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(mt) + \sum_{m=0}^{9} D_m \cdot \sin(mt)$$

The following equations calculate values for the preceding equation:

$$C_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \cos(m \cdot t) \cdot dt$$

$$D_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \sin(m \cdot t) \cdot dt$$

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(m \cdot t) + \sum_{m=0}^{9} D_m \cdot \sin(m \cdot t)$$

The following equations approximate the C and D values:

$$C_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \cos\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

$$D_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \sin\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

The following equations calculate the magnitude and phase:

$$R_m = (C_m^2 + D_m^2)^{1/2}$$

$$\Phi_m = \arctan\left(\frac{C_m}{D_m}\right)$$

Example 1

The following is input for an `.OP`, `.TRAN`, or `.FOUR` analysis. This example is based on demonstration netlist four.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
CMOS INVERTER
*
M1 2 1 0 0 NMOS W=20U L=5U
M2 2 1 3 3 PMOS W=40U L=5U
VDD 3 0 5
VIN 1 0 SIN 2.5 2.5 20MEG
*
.MODEL NMOS NMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.MODEL PMOS PMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.OP
.TRAN 1N 500N
.FOUR 20MEG V(2)
.PRINT TRAN V(2) V(1)
.END
```

Example 2

```
******
cmos inverter
**** fourier analysis   tnom = 25.000 temp = 25.000 ****
fourier components of transient response v(2)
dc component=2.430D+00
harmonic   frequency  fourier   normalized  phase    normalized
   no        (hz)    component   component  (deg)    phase (deg)
 1       20.0000x    3.0462     1.0000   176.5386     0.
```

```
 2       40.0000x  115.7006m   37.9817m -106.2672  -282.8057
 3       60.0000x  753.0446m  247.2061m  170.7288    -5.8098
 4       80.0000x   77.8910m   25.5697m -125.9511  -302.4897
 5      100.0000x  296.5549m   97.3517m  164.5430   -11.9956
 6      120.0000x   50.0994m   16.4464m -148.1115   -324.6501
 7      140.0000x  125.2127m   41.1043m  157.7399   -18.7987
 8      160.0000x   25.6916m    8.4339m  172.9579    -3.5807
 9      180.0000x   47.7347m   15.6701m  154.1858   -22.3528
    total harmonic distortion=   27.3791   percent
```

Spectrum analysis represents a time-domain signal, within the frequency domain. It most commonly uses the Fourier transform. A Discrete Fourier Transform (DFT) uses sequences of time values to determine the frequency content of analog signals, in circuit simulation.

The Fast Fourier Transform (FFT) calculates the DFT, which Synopsys HSPICE uses for spectrum analysis. The `.FFT` statement uses the internal time point values.

By default, `.FFT` uses a second-order interpolation to obtain waveform samples, based on the number of points that you specify.

You can use windowing functions to reduce the effects of waveform truncation on the spectral content. You can also use the `.FFT` command to specify:

- output format
- frequency
- number of harmonics
- total harmonic distortion (THD)

10

AC Sweep and Small Signal Analysis

Describes how to perform AC sweep and small signal analysis.

This chapter covers AC small signal analysis, AC analysis of an RC network, and other AC analysis statements. For information on output variables, see AC Analysis Output Variables on page 260.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Command Reference*.

Using the .AC Statement

You can use the .AC statement for the following applications:

- Single/double sweeps
- Sweeps using parameters
- .AC analysis optimization
- Random/Monte Carlo anlayses

For .AC command syntax and examples, see the .AC command in the *HSPICE Command Reference*.

.AC Control Options

You can use the following .AC control options when performing an AC analysis:

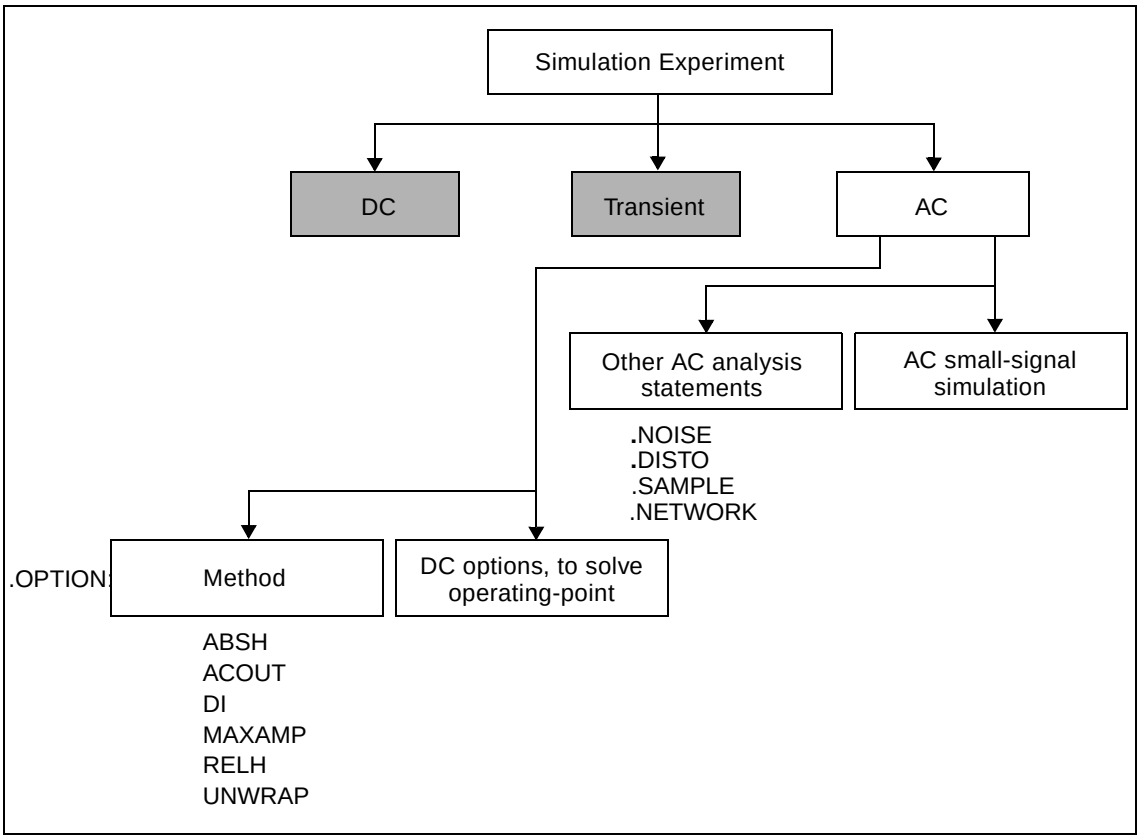
ABSH	ACOUT	DI
MAXAMP	RELH	UNWRAP

For syntax descriptions for these options, see the "Netlist Control Options" chapter in the *HSPICE Command Reference*.

AC Small Signal Analysis

AC small signal analysis in HSPICE or HSPICE RF computes AC output variables as a function of frequency (see Figure 50 on page 344). HSPICE or HSPICE RF first solves for the DC operating point conditions. It then uses these conditions to develop linear, small-signal models for all non-linear devices in the circuit.

Figure 50 AC Small Signal Analysis Flow



In HSPICE or HSPICE RF, the output of AC Analysis includes voltages and currents.

HSPICE or HSPICE RF converts capacitor and inductor values to their corresponding admittances:

$y_C = j\omega C$ for capacitors

$y_L = \frac{1}{j\omega L}$ for inductors

Resistors can have different DC and AC values. If you specify `AC=<value>` in a resistor statement, HSPICE or HSPICE RF uses the DC value of resistance to calculate the operating point, but uses the AC resistance value in the AC analysis. When you analyze operational amplifiers, HSPICE or HSPICE RF uses a low value for the feedback resistance to compute the operating point for the unity gain configuration. You can then use a very large value for the AC resistance in AC analysis of the open loop configuration.

AC analysis of bipolar transistors is based on the small-signal equivalent circuit, as described in the *HSPICE Elements and Device Models Manual*. MOSFET AC-equivalent circuit models are described in the *HSPICE Elements and Device Models Manual*.

The AC analysis statement can sweep values for:

- Frequency.
- Element.
- Temperature.
- Model parameter (HSPICE and HSPICE RF).
- Randomized (Monte Carlo) distribution (HSPICE only; not supported in HSPICE RF).
- Optimization and AC analysis (HSPICE or HSPICE RF).

Additionally, as part of the small-signal analysis tools, HSPICE or HSPICE RF provides:

- Noise analysis.
- Distortion analysis.
- Network analysis.
- Sampling noise.

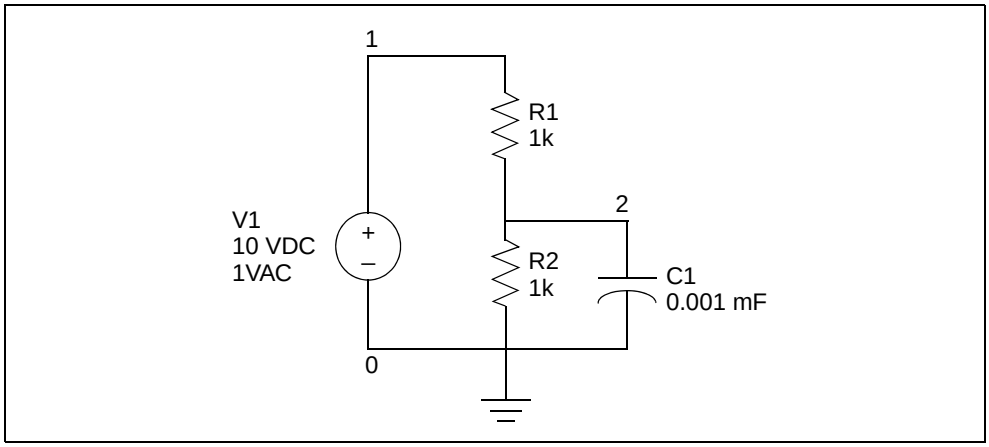
You can use the `.AC` statement in several different formats, depending on the application. You can also use the `.AC` statement to perform data-driven analysis in HSPICE, but not in HSPICE RF.

AC Analysis of an RC Network

Figure 51 on page 346 shows a simple RC network with a DC and AC source applied. The circuit consists of:

- Two resistors, R1 and R2.
- Capacitor C1.
- Voltage source V1.
- Node 1 is the connection between the source positive terminal and R1.
- Node 2 is where R1, R2, and C1 are connected.
- HSPICE ground is always node 0.

Figure 51 RC Network Circuit



The netlist for this RC network is based on demonstration netlist quickAC.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
A SIMPLE AC RUN
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 1MEG
.PRINT AC V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

Follow the procedure below to perform AC analysis for an RC network circuit.

1. Type the above netlist into a file named *quickAC.sp*.
2. To run a HSPICE analysis, type:

   ```
   hspice quickAC.sp > quickAC.lis
   ```

   For HSPICE RF, type:

   ```
   hspicext quickAC.sp > quickAC.lis
   ```

   When the run finishes, HSPICE displays:

   ```
   >info:      ***** hspice job concluded
   ```

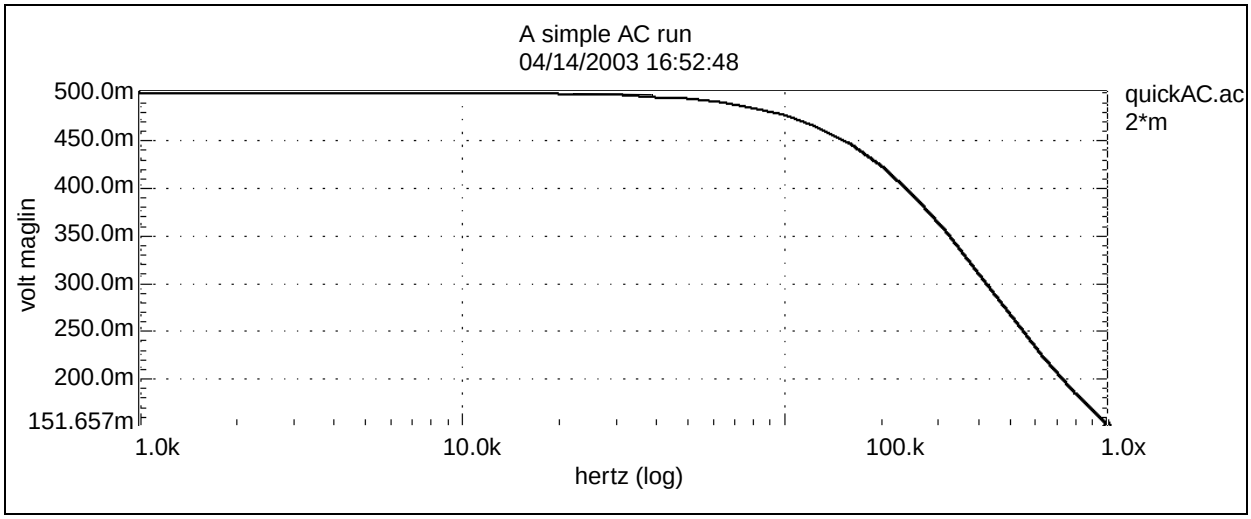
   This is followed by a line that shows the amount of real time, user time, and system time needed for the analysis.

   Your run directory includes the following new files:

   - quickAC.ac0
   - quickAC.ic0
   - quickAC.lis
   - quickAC.st0
3. Use an editor to view the *.lis* and *.st0* files to examine the simulation results and status.
4. Run AvanWaves and open the *.sp* file.
5. To view the waveform, select the *quickAC.ac0* file from the Results Browser window.
6. Display the voltage at node 2 by using a log scale on the x-axis.

Figure 52 on page 348 shows the waveform that HSPICE or HSPICE RF produces if you sweep the response of node 2, as you vary the frequency of the input from 1 kHz to 1 MHz.

Figure 52 RC Network Node 2 Frequency Response



As you sweep the input from 1 kHz to 1 MHz, the quickAC.lis file displays:

- Input netlist.
- Details about the elements and topology.
- Operating point information.
- Table of requested data.

The *quickAC.ic0* file contains information about DC operating point conditions. The quickAC.st0 file contains information about the simulation run status.

To use the operating point conditions for subsequent simulation runs, execute the `.LOAD` statement (HSPICE only; HSPICE RF does not support the `.LOAD` statement).

Other AC Analysis Statements

The following sections describe the commands you can use to perform other types of AC analyses:

- Using .DISTO for Small-Signal Distortion Analysis on page 349
- Using .NOISE for Small-Signal Noise Analysis on page 349
- Using .SAMPLE for Noise Folding Analysis on page 350

Use the `.NOISE` and `.AC` statements to control the noise analysis of the circuit.

Using .DISTO for Small-Signal Distortion Analysis

The `.DISTO` statement computes the distortion characteristics of the circuit in an AC small-signal, sinusoidal, steady-state analysis. HSPICE computes and reports five distortion measures at the specified load resistor. The analysis is performed assuming that one or two signal frequencies are imposed at the input. The first frequency, F1 (used to calculate harmonic distortion), is the nominal analysis frequency set by the `.AC` statement frequency sweep. The optional second input frequency, F2 (used to calculate intermodulation distortion), is set implicitly by specifying the `skw2` parameter, which is the ratio F2/F1.

For command syntax and examples, see the .DISTO command in the *HSPICE Command Reference*.

Using .NOISE for Small-Signal Noise Analysis

Noise calculations in HSPICE or HSPICE RF are based on complex AC nodal voltages, which in turn are based on the DC operating point. For descriptions of noise models for each device type, see the *HSPICE Elements and Device Models Manual*. Each noise source does not statistically correlate to other noise sources in the circuit; the HSPICE or HSPICE RF simulator calculates each noise source independently. The total output noise voltage is the RMS sum of the individual noise contributions:

$$onoise = \sum_{k=0}^{N} |Z_k|^2 \overline{|i_{nk}|^2}$$

Where,

onoise is the total output noise (HSPICE or HSPICE RF).

i_{nk} is the normal current source due to thermal, shot, or other noise.

Z_k is the equivalent transimpedance between each noise current source and output.

N is the number of noise sources associated with all circuit elements.

The input noise (*inoise*) voltage is the total output noise divided by the gain or transfer function of the circuit. HSPICE or HSPICE RF prints the contribution of each noise generator in the circuit for each inter frequency point. The simulator

also normalizes the output and input noise levels relative to the square root of the noise bandwidth. The units are volts/$Hz^{1/2}$ or amps/$Hz^{1/2}$.

To simulate flicker noise sources in the noise analysis, include values for the KF and AF parameters on the appropriate device model statements. Use the `.PRINT` or `.PLOT` statement to print or plot output noise, and the equivalent input noise.

If you specify more than one `.NOISE` statement in a single simulation, HSPICE or HSPICE RF runs only the last statement.

Table 46 .NOISE Measurements Available for MOSFETs

.ac	.lis	Unit	Description
nd	rd	$\frac{V^2}{Hz}$	Output thermal noise due to drain resistor
ns	rs	$\frac{V^2}{Hz}$	Output thermal noise due to source resistor
ni	id	$\frac{V^2}{Hz}$	Output channel thermal noise
nf	fn	$\frac{V^2}{Hz}$	Output flicker noise
ntg	total	$\frac{V^2}{Hz}$	Total output noise: TOT=RD + RS + ID + FN

Using .SAMPLE for Noise Folding Analysis

For data acquisition of analog signals, data sampling noise often needs to be analyzed. This is accomplished with the `.SAMPLE` statement used in conjunction with the `.NOISE` and `.AC` statements. The SAMPLE analysis performs a simple noise folding analysis at the output node.

For the syntax and description of the `.SAMPLE` statement, see the .SAMPLE command in the *HSPICE Command Reference*.

11

Linear Network Parameter Analysis

Describes how to perform an AC sweep to extract small-signal linear network parameters.

The chapter covers .LIN analysis, RF measurements from .LIN, extracting mixed-mode S (scattering) parameters, and .NET parameter analysis.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Command Reference*.

.LIN Analysis

The .LIN command extracts noise and linear transfer parameters for a general multi-port network.

When used with the .AC command, .LIN makes available a broad set of linear port-wise measurements:

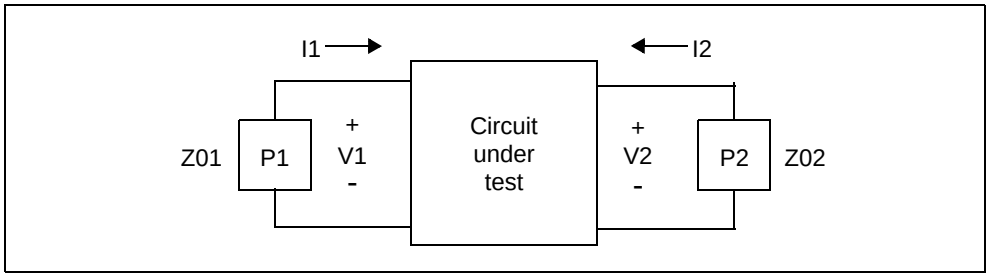
- Multi-port scattering [S] parameters
- Noise parameters
- Stability factors
- Gain factors
- Matching coefficients

The .LIN analysis is similar to basic small-signal, swept-frequency .AC analysis, but it also automatically calculates a series of noise and small-signal transfer parameters between the terminals identified using port (P) elements.

HSPICE can output the result of group delay extraction and two-port noise analysis to either a *.sc** file, a Touchstone file, or a CITIfile.

The .PRINT/.PROBE/.MEAS output syntax for .LIN supports H (hybrid) parameters and S/Y/Z/H group delay.

Figure 53 Basic Circuit in .LIN Analysis



Identifying Ports with the Port Element

The .LIN command computes the S (scattering), Y (admittance), and Z (impedance) parameters directly based on the location of the port (P) elements in your circuit, and the specified values for their reference impedances.

The port element identifies the ports used in .LIN analysis. Each port element requires a unique port number. If your design uses N port elements, your netlist must contain the sequential set of port numbers 1 through *N* (for example, in a design containing 512 ports, you must number each port sequentially 1 to 512).

Each port has an associated system impedance, z0. If you do not explicitly specify the system impedance, the default is 50 ohms.

The port element behaves as either a noiseless impedance or a voltage source in series with the port impedance for all other analyses (DC, AC, or TRAN).

- You can use this element as a pure terminating resistance or as a voltage or power source.
- You can use the RDC, RAC, RHB, RHBAC, and rtran values to override the port impedance value for a particular analysis.

Syntax

```
Pxxx p n port=portnumber
+ $ **** Voltage or Power Information ********
+ <DC mag> <AC <mag <phase>>> <HBAC <mag <phase>>>
+ <HB <mag <phase <harm <tone <modharm <modtone>>>>>>>
+ <transient_waveform> <TRANFORHB=[0|1]>
+ <DCOPEN=[0|1]>
+ $ **** Source Impedance Information ********
```

```
+ <Z0=val> <RDC=val> <RAC=val>
+ <RHBAC=val> <RHB=val> <RTRAN=val>
+ $ **** Power Switch ********
+ <power=[0|1|2|W|dbm]>
```

Parameter	Description
port=portnumber	The port number. Numbered sequentially beginning with 1 with no shared port numbers.
<DC mag>	DC voltage or power source value.
<AC <mag <phase>>>	AC voltage or power source value.
<HBAC <mag <phase>>>	(HSPICE RF) HBAC voltage or power source value.
<HB <mag <phase <harm <tone <modharm <modtone>>>>>>>	(HSPICE RF) HB voltage, current, or power source value. Multiple HB specifications with different *harm*, *tone*, *modharm*, and *modtone* values are allowed. ▪ *phase* is in degrees ▪ *harm* and *tone* are indices corresponding to the tones specified in the .HB statement. Indexing sta4 r(m)-2.1(0 Tw(to(

Parameter	Description
<TRANFORHB=[0|1]>	■ 0 (default): The transient description is ignored if an HB value is given or a DC value is given. If no DC or HB value is given and TRANFORHB=0, then HB analysis treats the source as a DC source, and the DC source value is the time=0 value. ■ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, then the time=infinity value is used as a DC analysis source value. For example, the following statement is treated as a DC source with value=1 for HB analysis: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=1 In contrast, the following statement is a 0V DC source: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=0 The following statement is treated as a periodic source with a 1us period that uses PWL values: v1 1 0 PWL (0 0 1n 1 0.999u 1 1u 0) R + TRANFORHB=1 To override the global TRANFORHB option, explicitly set TRANFORHB for a voltage or current source.
DCOPEN	Switch for open DC connection when DC *mag* is not set. ■ 0 (default): P element behaves as an impedance termination. ■ 1 : P element is considered an open circuit in DC operating point analysis. DCOPEN=1 is mainly used in .LIN analysis so the P element will not affect the self-biasing device under test by opening the termination at the operating point.
<z0=val>	(LIN analysis) System impedance used when converting to a power source, inserted in series with the voltage source. Currently, this only supports real impedance. ■ When power=0, z0 defaults to 0. ■ When power=1, z0 defaults to 50 ohms. You can also enter z0=*val*.

Parameter	Description
<RDC=val>	(DC analysis) Series resistance (overrides `z0`).
<RAC=val>	(AC analysis) Series resistance (overrides z0).
<RHBAC=val>	(HSPICE RF HBAC analysis) Series resistance (overrides z0).
<RHB=val>	(HSPICE RF HB analysis) Series resistance (overrides `z0`).
<RTRAN=val>	(Transient analysis) Series resistance (overrides `z0`).
<power=[0 | 1 | 2 | W | dbm]>	(HSPICE RF) power switch ▪ When `0` (default), element treated as a voltage or current source. ▪ When 1 or W, element treated as a power source, realized as a voltage source with a series impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ▪ When 2 or dbm, element treated as a power source in series with the port imedance. Values are in dbms. You can use this parameter for Transient analysis if the power source is either DC or SIN.

Example

For example, the following port element specifications identify a 2-port network with 50-Ohm reference impedances between the "in" and "out" nodes.

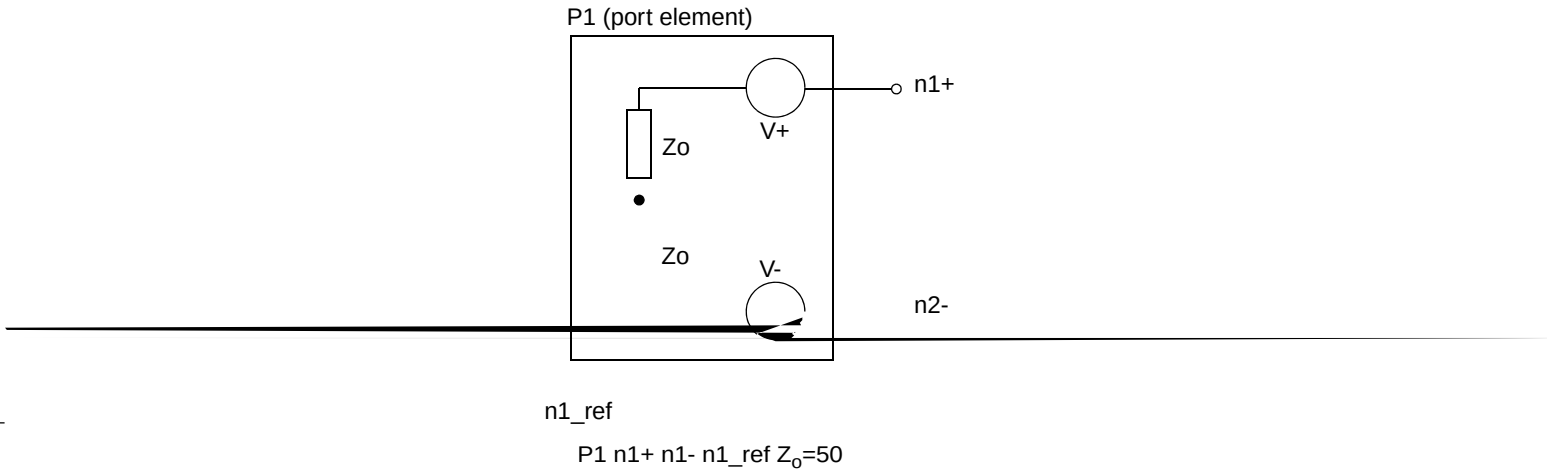
```
P1 in gnd port=1 z0=50
P2 out gnd port=2 z0=50
```

Computing scattering parameters requires `z0` reference impedance values. The order of the *port* parameters (in the P Element) determines the order of the S, Y, and Z parameters. Unlike the `.NET` command, `.LIN` does not require you to insert additional sources into the circuit. To calculate the requested transfer parameters, HSPICE automatically inserts these sources as needed at the port terminals. You can define an unlimited number of ports.

Using the P (Port) Element for Mixed-Mode Measurement

You can use a port element with three terminals as the port element for measuring the mixed mode S parameters. Except for the number of external terminals, the syntax of the port element remains the same. The `.LIN` analysis function internally sets the necessary drive mode (common/differential) of these mixed mode port elements. For analyses other than the `.LIN` analysis (such as DC, AC, TRAN, and so on), the mixed-mode P Element acts as a differential driver that drives positive nodes with half of their specified voltage and the negative nodes with a negated half of the specified voltage. Figure 54 shows the block diagram of the mixed mode port element.

Figure 54 Mixed Mode Port Element



The port element can also be used as a signal source with a built in reference impedance. For further information on its use as a signal source, see Chapter 5, Sources and Stimuli.

.LIN Input Syntax

```
.LIN <sparcalc=[1|0] <modelname = ...>>
+ <filename = ...> <format=[selem|citi|touchstone]>
+ <noisecalc=[1|0] <gdcalc=[1|0]>
+ <mixedmode2port=[dd|dc|ds|cd|cc|cs|sd|sc|ss]>
+ <dataformat=[ri|ma|db]>
```

For argument descriptions, see the .LIN command in the *HSPICE Command Reference*.

.LIN Output Syntax

This section describes the syntax for the `.PRINT` and `.PROBE` statements used for LIN analysis.

.PRINT and .PROBE Statements

```
.PRINT AC <Xmn | Xmn LINPARAM(TYPE) | X(m,n) |
+ X(m,n) LINPARAM(TYPE)> <Hmn | Hmn(TYPE) |
+ H(m,n) | H(m,n)(TYPE)>
.PROBE AC <Xmn | Xmn LINPARAM(TYPE) | X(m,n) |
+ X(m,n) LINPARAM(TYPE)> <Hmn | Hmn(TYPE) |
+ H(m,n) | H(m,n)(TYPE)>
```

Argument	Description
Xmn, X(m,n)	One of these parameter types: ▪ S (scattering parameters) ▪ Y (admittance parameters) ▪ Z (impedance parameters) ▪ H (hybrid parameters) mn refers to a pair of port numbers, where m can be 1 or 2, and n can be 1 or 2.
Hmn, H(m,n)	Complex hybrid (H-) parameters. mn refers to a pair of port numbers, where m can be 1 or 2, and n can be 1 or 2. If m,n=0 or m,n>2, HSPICE issues a warning and ignores the output request. ▪ To calculate a one-port H parameter, you must specify at least one port (P) element. ▪ To calculate a two-port H parameter, you must specify two or more port (P) elements. For additional information, see Hybrid Parameter Calculations on page 359.

Argument	Description
LINPARAM	Two-port noise parameters: ■ NFMIN (noise figure minimum) ■ NF (Noise figure) ■ VN2 (Equivalent input noise voltage squared ■ IN2 (Equivalent input noise current squared) ■ RHON (Correlation coefficient between input noise voltage and input noise current) ■ RN (Noise equivalent resistance) ■ GN (Noise equivalent conductance) ■ ZCOR (Noise correlation impedance) ■ YCOR (Noise correlation admittance) ■ ZOPT (Optimum source impedance for noise) ■ YOPT (Optimum source admittance for noise) ■ GAMMA_OPT (source reflection coefficient that achieves the minimum noise figure) ■ ZOPT (source impedance that achieves minimum noise) ■ RN (noise equivalent resistance) ■ K_STABILITY_FACTOR (Rollett stability factor) ■ MU_STABILITY_FACTOR (Edwards & Sinsky stability factor) ■ G_MAX (maximum available/operating power gain) ■ G_MSG (Maximum stable gain) ■ G_TUMAX (Maximum unilateral transducer power gain) ■ G_U (Unilateral power gain) ■ G_MAX_GAMMA1 (source reflection coefficient that achieves maximum available power gain) ■ G_MAX_GAMMA2 (load reflection coefficient that achieves maximum operating power gain) ■ G_MAX_Z1=Source impedance needed to realize G_MAX (complex, Ohms) ■ G_MAX_Z2=Load impedance needed to realize G_MAX (complex, Ohms) ■ G_MAX_Y1=Source admittance needed to realize G_MAX (complex, Siemens) ■ G_MAX_Y2=Load admittance needed to realize G_MAX (complex, Siemens) ■ G_AS (associate gain—maximum gain at the minimum noise figure) ■ VSWR(n) (voltage standing-wave ratio at the n port) ■ GD (group delay from port=1 to port=2) ■ G_MSG (maximum stable gain) ■ G_TUMAX (maximum unilateral transducer power gain) ■ G_U (unilateral power gain)

Argument	Description
TYPE	Data type definitions: ■ R=Real ■ I=Imaginary ■ M=Magnitude ■ P=PD=Phase in degrees ■ PR=Phase in radians ■ DB=decibels

Examples

```
.print AC S11 S21(DB) S(2,3)(D) S(2,1)(I)
.print AC NFMIN GAMMA_OPT G_AS
.probe AC RN G_MAX ZOPT Y(3,1)(M) Y31(P)
```

Hybrid Parameter Calculations

The hybrid parameters are transformed from S-parameters:

- For a one-port circuit, the calculation is:

$$H_{11} = Z_{01}\frac{(1+S_{11})}{(1-S_{11})}$$

- For a two-port circuit, the calculation is:

$$H_{11} = Z_{01}\frac{(1+S_{11})(1+S_n)-S_{12}S_{21}}{(1-S_{11})(1+S_n)+S_{12}S_{21}}$$

$$H_{12} = \sqrt{\frac{Z_{02}}{Z_{01}}}\frac{2S_{12}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

$$H_{21} = \sqrt{\frac{Z_{02}}{Z_{01}}}\frac{-S_{21}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

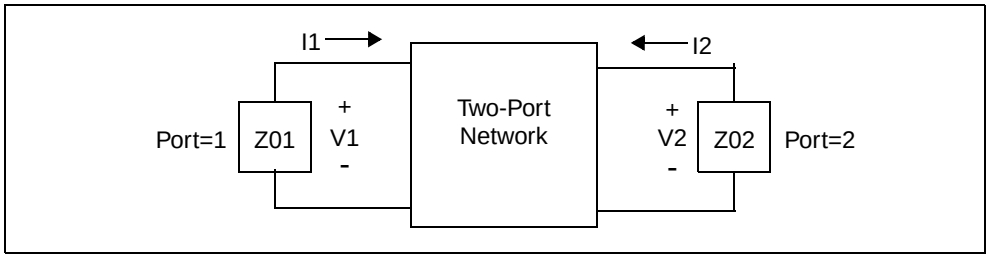
$$H_{22} = \frac{1}{Z_{02}}\frac{(1-S_{11})(1-S_{22})-S_{12}S_{21}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

For networks with more than two ports when computing the 1,2 H index parameters, HSPICE assumes that ports numbered 3 and above terminate in their port reference impedance (z0). The above two-port calculations therefore remain appropriate, because S11, S12, S21, and S22 remain valid, and simulation can ignore higher order S-parameters.

Multi-Port Scattering (S) Parameters

S-parameters represent the ratio of incident and scattered (or forward and reflected) normalized voltage waves. Figure 55 shows a two-port network.

Figure 55 Two-Port Network



The following equations define the incident (forward) waves for this two-port network:

$$a_1 = \frac{v_1 + Z_{01} I_1}{2 \cdot \sqrt{Z_{01}}} \qquad a_2 = \frac{v_2 + Z_{02} I_2}{2 \cdot \sqrt{Z_{02}}}$$

The following equations define the scattered (reflected) waves for this two-port network:

$$b_1 = \frac{v_1 - Z_{01} I_1}{2 \cdot \sqrt{Z_{01}}} \qquad b_2 = \frac{v_2 - Z_{02} I_2}{2 \cdot \sqrt{Z_{02}}}$$

The following equations define the S parameters:

$$S_{11} = \frac{b_1}{a_1}\bigg|_{a_2 = 0} \qquad S_{12} = \frac{b_1}{a_2}\bigg|_{a_1 = 0}$$

$$S_{21} = \frac{b_2}{a_1}\bigg|_{a_2 = 0} \qquad S_{22} = \frac{b_2}{a_2}\bigg|_{a_1 = 0}$$

Each S-parameter is a complex number, which can represent gain, isolation, or a reflection coefficient.

Example

The following examples show how you can represent a gain, isolation, or reflection coefficient:

```
.PRINT AC S11(DB) $ Input return loss
.PRINT AC S21(DB) $ Gain
```

```
.PRINT AC S12(DB) $ Isolation
.PRINT AC S22(DB) $ Output return loss
```

Two-Port Transfer and Noise Calculations

Two-port noise analysis is a linear AC noise analysis method that determines the noise figure of a linear two-port for an arbitrary source impedance.

Several output parameter measurements are specific to two-port networks. `.LIN` analysis supports two-port calculations for 3 or more ports if port=1 is the input and port=2 the output. All other ports terminate in their characteristic impedance. This is equivalent to operating on the two-port [S] sub-matrix extracted from the multi-port [S] matrix. This occurs for both signal and noise calculations. A warning appears if N>2 and you specified two-port quantities.

Noise and signal port-wise calculations do not require that port elements use a ground reference. You can therefore measure fully-differential circuits.

`.LIN` generates a set of noise parameters. The analysis assumes a noise model consisting of:

- A shunt current noise source, called In, at the input of a noiseless two-port linear network.
- A series voltage noise source, called Vn, at the input of a noiseless two-port linear network.
- A source with impedance, called Zs, that drives this two-port network.
- The two-port network drives a noiseless load, called Zl.

Equivalent Input Noise Voltage and Current

For each analysis frequency, HSPICE computes a noise equivalent circuit for a linear two-port. The noise equivalent circuit calculation results in an equivalent noise voltage and current, and their correlation coefficient.

- `VN2`: Equivalent input noise voltage squared (Real, V^2).
- `IN2`: Equivalent input noise current squared (Real, A^2).
- `RHON`: Correlation coefficient between the input noise voltage and the input noise current (complex, unitless).

Equivalent Noise Resistance and Conductance

These measurements are the equivalent resistance and conductance, which generate the equivalent noise voltage and current values at a temperature of T=290K in a 1Hz bandwidth.

- RN: Noise equivalent resistance (Real, Ohms)
- GN: Noise equivalent conductance (Real, Siemens)

Noise Correlation Impedance and Admittance

These measurements represent the equivalent impedance and admittance that you can insert at the input of the noise equivalent circuit to account for the correlation between the equivalent noise voltage and the current values.

- ZCOR: Noise correlation impedance (Complex, Ohms)
- YCOR: Noise correlation admittance (Complex, Siemens)

Optimum Matching for Noise

These measurements represent the optimum impedance, admittance, and reflection coefficient value that result in the best noise performance (minimum noise figure).

- ZOPT: Optimum source impedance for noise (Complex, Ohms)
- YOPT: Optimum source admittance for noise (Complex, Siemens)
- GAMMA_OPT: Optimum source reflection coefficient (Complex, unitless)

Because ZOPT and YOPT can commonly take on infinite values when computing optimum noise conditions, calculations for optimum noise loading reflect the GAMMA_OPT coefficient.

Noise Figure and Minimum Noise Figure

Noise figure represents the ratio of the SNR (signal to noise ratio) at the input to the SNR at the output. You can set the input source impedance to ZOPT to obtain the minimum noise figure.

- NFMIN: Minimum noise figure (source at ZOPT) (real, unitless, power ratio)
- NF: Noise figure (value obtained with source impedance at Zc[1]) (real, unitless, power ratio)

Associated Gain

This measurement assumes that the input impedance matches the minimum noise figure, and the output matches the maximum gain.

G_AS is the associated gain—maximum power gain at NFMIN (real, power ratio)

Output Format for Group Delay in .sc* Files

All of the S/Y/Z/H parameters support a group delay calculation. The output syntax of .PRINT and .PROBE statement for group delay is:

```
Xmn(T) | Xmn(TD) | X(m,n)(T) | X(m,n)(TD)
```

- X=S, Y, Z, or H
- m, n=port number (1 or 2 for H parameter)

The output of group delay matrices in .sc* files lets HSPICE directly read back the group delay information, the tabulated data uses the regular HSPICE model syntax with the SP keyword:

```
*| group delay parameters
.MODEL SMODEL_GD SP N=2 SPACING=POI INTERPOLATION=LINEAR
+ MATRIX=NONSYMMETRIC VALTYPE=REAL
+ DATA=3
+          1e+08
+              0         5e-09
+          5e-09             0
+    {...data...}
```

model name is the model name of the S parameters, plus _GD.

GROUPDELAY=[0|1] in the top line indicates group delay data:

```
*| N=2 DATA=3 NOISE=0 GROUPDELAY=1
*| NumOfBlock=1 NumOfParam=0
```

Output Format for Two-Port Noise Parameters in .sc* Files

Output of two-port noise parameter data in .sc* files shows the tabulated data with the following quantities in the following order:

```
*| 2-port noise parameters
*|   frequency Fmin[dB] GammaOpt(M) GammaOpt(P) RN/Z0
*|     {...data...}
```

In this syntax:

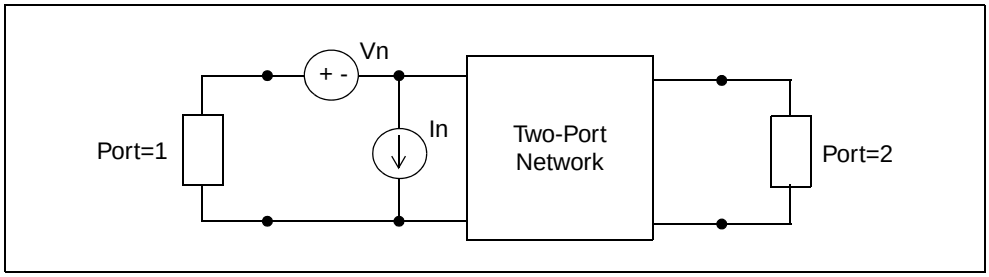
- `Fmin[dB]`=minimum noise figure (dB).
- `GammaOpt(M)`=magnitude of the reflection coefficient needed to realize Fmin.
- `GammaOpt(P)`=phase (degrees) of the reflection coefficient needed to realize Fmin.
- `RN/Z0`=normalized noise resistance.

Both GammaOpt and RN/Z0 values are normalized with respect to the characteristic impedance of the port=1 element (that is, Z01).

Noise Parameters

You can use the `.LIN` analysis to compute the equivalent two-port noise parameters for a network. The `noisecalc=1` option automatically calculates the following equivalent circuit values.

Figure 56 Noise Equivalent Circuit



- V_n is the equivalent input-referred noise voltage source.
- I_n is the equivalent input-referred noise current source.
- I_nV_n is their correlation.

HSPICE can output the result of `.LIN` noise analysis to a .sc*, Touchstone, or CITIfile.

HSPICE noise analysis also makes the following measurements available:

$$R_n = \frac{\overline{|V_n|^2}}{4kT} \qquad G_n = \frac{\overline{|I_n|^2}}{4kT}$$

$$Z_{cor} = \frac{\overline{I_n * V_n}}{\overline{|I_n|^2}} = R_{cor} + jX_{cor} \qquad Z_{opt} = \sqrt{\frac{R_n}{G_n} - (R_{cor})^2} - jX_{cor}$$

$$F_{min} = 1 + 2G_n\left(R_{cor} + \sqrt{\frac{R_n}{G_n} - (X_{cor})^2}\right) \qquad \gamma I_{opt} = \frac{Z_{opt} - Z_0}{Z_{opt} + Z_0}$$

Hybrid (H) Parameters

.LIN analysis can calculate the complex two-port H (hybrid) parameter of a multi-terminal network.

The H parameters of a two-port network relate the voltages and currents at input and output ports:

$$V1 = h11 \cdot I1 + h12 \cdot V2$$

$$I2 = h21 \cdot I1 + h22 \cdot V2$$

In the preceding equations:

- $H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$ Hybrid matrix
- V1=Voltage at input port
- I2=Current at output port
- V2=Voltage at output port
- I1=Current at input port

You can add the hybrid H parameter matrixes of two networks to describe networks that are in series at their input and in parallel at their output.

.LIN can calculate H parameters based on the scattering parameters of the networks. .LIN analysis can extract one-port and two-port network H parameters. For networks with more than two ports, .LIN assumes that the ports numbered 3 and above terminate in their port characteristic impedance (Zc[i], i>2).

Group Delay

Group delay measures the transit time of a signal through a network versus frequency. It reduces the linear portion of the phase response to a constant value, and transforms the deviations from linear phase into deviations from constant group delay (which causes phase distortion in communications systems). The average delay represents the average signal transit time through a network system.

HSPICE can output the result of `.LIN` group delay measurement to a .sc*, Touchstone, or CITIfile.

Group delay is a function of frequency:

$$gd(w) = \frac{-d(phase)}{d(w)}$$

Where,

- gd=Group delay at the *f* frequency, $2\pi f = w$
- phase=phase response at the *f* frequency
- w=radians frequency

All complex S, Y, Z, and H parameters support a group delay calculation.

Syntax

```
Xmn(T) | Xmn(TD) | X(m,n)(T) | X(m,n)(TD)
X=S, Y, Z, or H (parameters)
m,n=port number (1 or 2 for H-parameters)
```

The results of the group delay calculation are scalar real numbers in units of seconds. For `.LIN`, group delay values are a function of frequency. The calculation is:

$$r_{ij} = |r_{ij}|e^{jf_{ij}(w)}$$

Differentiating the complex logarithm with respect to omega results in:

$$\frac{1}{r_{ij}}\frac{dr_{ij}}{dw} = \frac{1}{|r_{ij}|}\frac{d|r_{ij}|}{dw} + j\frac{df}{dw}$$

The group delay is the negative derivative of the phase. Simulation can compute it from the imaginary component of the derivative w.r.t. frequency of the measurement:

$$\tau(w) = -\frac{df}{dw} = -Im\left[\frac{1}{r_{ij}}\frac{dr_{ij}}{dw}\right]$$

RF Measurements From .LIN

In addition to S, Y, Z, and H parameters, a LIN analysis can include the output measurements in the following sections.

Impedance Characterizations

- `VSWR(i)`=Voltage standing wave ratio at port i (real, unit-less)
- `ZIN(i)`=Input impedance at port i (complex, Ohms)
- `YIN(i)`=Input admittance at port i (complex, Siemens)

Stability Measurements

- `K_STABILITY_FACTOR`=Rollett stability factor (real, unit-less)
- `MU_STABILITY_FACTOR`=Edwards & Sinsky stability factor (real, unit-less)

Gain Measurements

- `G_MAX`=Maximum available/operating power gain (real, power ratio)
- `G_MSG`=Maximum stable gain (real, power ratio)
- `G_TUMAX`=Maximum unilateral transducer power gain (real, power ratio)
- `G_U`=Unilateral power gain (real, power ratio)

Matching for Optimal Gain

- G_MAX_GAMMA1=Source reflection coefficient needed to realize G_MAX (complex, unit-less)
- G_MAX_GAMMA2=Load reflection coefficient needed to realize G_MAX (complex, unit-less)
- G_MAX_Z1=Source impedance needed to realize G_MAX (complex, Ohms)
- G_MAX_Z2=Load impedance needed to realize G_MAX (complex, Ohms)
- G_MAX_Y1=Source admittance needed to realize G_MAX (complex, Siemens)
- G_MAX_Y2=Load admittance needed to realize G_MAX (complex, Siemens)

Noise Measurements

- VN2=Equivalent input noise voltage squared (real, V2)
- IN2=Equivalent input noise current squared (real, A2)
- RHON=Correlation coefficient between input noise voltage and input noise current (complex, unit-less)
- RN=Noise equivalent resistance (real, Ohms)
- GN=Noise equivalent conductance (real, Siemens)
- ZCOR=Noise correlation impedance (complex, Ohms)
- YCOR=Noise correlation admittance (complex, Siemens)
- ZOPT=Optimum source impedance for noise (complex, Ohms)
- YOPT=Optimum source admittance for noise (complex, Siemens)
- GAMMA_OPT=Optimum source reflection coefficient (complex, unit-less)
- NFMIN=Noise figure minimum (source at Zopt) (real, unit-less power ratio)
- NF=Noise figure (value obtained with source impedance at Z01) (real, unit-less power ratio)
- G_AS=Associated gain -- maximum power gain at NFMIN (real, power ratio)

Two-Port Transfer and Noise Measurements

Several of the output parameter measurements are assumed to be for two-port networks. When the network has 3 or more ports, the measurements are still carried out by assuming that `port=1` is the input and `port=2` is the output. All other ports are assumed terminated in their (noiseless) characteristic (z0) impedances. Note that this assumption is equivalent to operating on the two-port [S] sub-matrix extracted from the multi-port [S] matrix. This is true for both signal and noise calculations. A warning message is issued in cases where N>2 when two-port quantities are requested.

Signal and noise port-wise calculations do not require that port elements use a ground reference. Measurements are therefore possible; for example, for fully differential circuits.

Since Zopt and Yopt can commonly take on infinite values when computing optimum noise conditions, calculations for optimum noise loading is performed in terms of the reflection coefficient GammaOpt, and is made as robust as possible.

Output Format for Two-Port Noise Parameters in .sc* Files

The output of two-port noise parameter data in *.sc** files are slightly modified. The tabulated data appears with the following quantities in the following order:

```
*| 2-port noise parameters
*|  frequency  Fmin[dB]  GammaOpt(M) GammaOpt(P)  RN/Z0
*|    {...data...}
```

Where

- `Fmin[dB]` is the minimum noise figure (dB)
- `GammaOpt(M)` is the magnitude of reflection coefficient needed to realize `Fmin`
- `GammaOpt(P)` is the phase (degrees) of reflection coefficient needed to realize `Fmin`
- `RN/Z0` is the normalized noise resistance

Both `GammaOpt` and `RN/Z0` values are normalized with respect to the characteristic impedance of the port=1 element; for example, Z01.

VSWR

The *Voltage Standing Wave Ratio* represents the ratio of maximum to minimum voltages along a standing wave pattern due to a port's impedance mismatch. All ports other than the port of interest terminate in their characteristic impedances. VSWR is a real number related to that port's scattering parameter:

$$VSWR[i] = \frac{1 + |s_{ii}|}{1 - |s_{ii}|}$$

ZIN(i)

The Input Impedance at the *i* port is the complex impedance into a port with all other ports terminated in their appropriate characteristic impedance. It is related to that port's scattering parameter:

$$ZIN[i] = Z_{0i}\frac{1 + S_{ii}}{1 - S_{ii}}$$

YIN(i)

The Input Admittance at the *i* port is the complex admittance into a port with all other ports terminated in their appropriate characteristic impedance. It is related to that port's scattering parameter:

$$YIN[i] = \frac{1}{Z_{0i}}\frac{1 - S_{ii}}{1 + S_{ii}}$$

K_STABILITY_FACTOR (Rollett Stability Factor)

The *Rollett* stability factor is:

$$K = \frac{1 - |s_{11}|^2 - |s_{22}|^2 + |\Delta|^2}{2|s_{12}||s_{21}|}$$

Δ determines the two-port *S* matrix calculated from this equation:

$$\Delta = s_{11}s_{22} - s_{12}s_{21}$$

An amplifier where K>1 is unconditionally stable at the selected frequency.

MU_STABILITY_FACTOR (Edwards-Sinsky Stability Factor)

The following equation defines the Edwards-Sinsky stability factor.

$$\mu = \frac{1 - |S_{11}|^2}{|S_{22} - \Delta S_{11}^*| + |S_{21}S_{12}|}$$

$$\Delta = S_{11}S_{22} - S_{12}S_{21}$$

An amplifier with μ>1 is considered unconditionally stable at the specified frequency.

Maximum Available Power Gain—G_MAX

This is the gain value that can be realized if the two-port is simultaneously conjugate-matched at both input and output (with no additional feedback):

$$G_{max} = \frac{|s_{21}|}{|s_{12}|}\left(K - \sqrt{K^2 - 1}\right)$$

K is the Rollett stability factor. Special cases of `G_MAX` are handled in the following manner:

- If $|S_{12}|=0$ and ($|S_{11}|=1$ or $|S_{22}|=1$), G_MAX=$|S_{21}|^2$
- If $|S_{12}|=0$ and $|S_{11}|\neq1$ and $|S_{22}|\neq1$, G_MAX=G_TUMAX
- If $|S_{12}|\neq0$ and K$\leq$ 1, G_MAX=G_MSG

When values for K$\leq$ 1, the Maximum Available Power Gain is undefined, and HSPICE RF returns the Maximum Stable Gain.

Maximum Stable Gain - G_MSG

For a two-port that is conditionally stable (K<1), the following equation calculates the maximum stable gain:

$$G_{MSG} = \frac{|s_{21}|}{|s_{12}|}$$

To achieve this gain, resistively load the unstable two-port so that K=1, and then simultaneously conjugately match the input and output ports. G_MSG is therefore equivalent to G_MAX with K=1. In terms of admittance parameters:

$$G_{MSG} = \frac{|y_{21}|}{|y_{12}|}$$

MSG is equivalent to the Maximum Available Power Gain if K=1.

Maximum Unilateral Transducer Power Gain —G_TUMAX

This is the highest possible gain that a two-port with no feedback (that is, S_{12}=0) can achieve.

$$G_{tumax} = \frac{|s_{21}|^2}{(1-|s_{11}|^2)(1-|s_{22}|^2)}$$

Unilateral Power Gain—GU

This is the highest gain that the active two-port can ever achieve by embedding in a matching network that includes feedback. The frequency where the unilateral gain becomes unity defines the boundary between an active and a passive circuit. The frequency is usually referred to as f_{max}, the maximum frequency of oscillation.

To realize this gain, HSPICE RF neutralizes the feedback of the two-port, and simultaneously conjugate-matches both input and output:

$$G_U = \frac{\left|\frac{s_{21}}{s_{12}} - 1\right|^2}{2K\left|\frac{s_{21}}{s_{12}}\right| - 2Re\left\{\frac{s_{21}}{s_{12}}\right\}}$$

Simultaneous Conjugate Match for G_MAX

A simultaneous conjugate match is required at the source and load to realize the G_{max} gain value. The source reflection coefficient at the input must be:

$$\Gamma_1 = \frac{C_1^*}{|C_1|}\left[\frac{B_1}{2|C_1|} - \sqrt{\frac{B_1^2}{|2C_1|^2} - 1}\right]$$

$$B_1 = 1 - |s_{22}|^2 + |s_{11}|^2 - |\Delta|^2$$

$$C_1 = s_{11} - \Delta s_{22}^* \qquad \Delta = s_{11}s_{22} - s_{12}s_{21}$$

The load reflection coefficient (`G_MAX_GAMMA_2`) is:

$$\Gamma_2 = \frac{C_2^*}{|C_2|}\left[\frac{B_2}{2|C_2|} - \sqrt{\frac{B_2^2}{|2C_2|^2} - 1}\right]$$

In the preceding equation:

$$B_2 = 1 - |s_{11}|^2 + |s_{22}|^2 - |\Delta|^2$$

$$C_2 = s_{22} - \Delta s_{11}^* \qquad \Delta = s_{11}s_{22} - s_{12}s_{21}$$

You can obtain useful solutions only when:

$$\frac{B_1}{|2C_1|} > 1 \qquad \frac{B_2}{|2C_2|} > 1$$

These equations also imply that K>1.

HSPICE RF derives calculations for the related impedances and admittances from the preceding values.

For G_MAX_Z1:

$$Z_1 = Z_{01}\frac{1+\Gamma_1}{1-\Gamma_1}$$

For G_MAX_Z2:

$$Z_2 = Z_{02}\frac{1+\Gamma_2}{1-\Gamma_2}$$

For G_MAX_Y1

$$Y_1 = \frac{1}{Z_{01}}\frac{1-\Gamma_1}{1+\Gamma_1}$$

For G_MAX_Y2

$$Y_2 = \frac{1}{Z_{02}}\frac{1-\Gamma_2}{1+\Gamma_2}$$

Equivalent Input Noise Voltage and Current—IN2, VN2, RHON

For each analysis frequency, HSPICE RF computes a noise-equivalent circuit for a linear two-port. The noise analysis assumes that all ports terminate in noise-less resistances. For circuits with more than two ports, ports identified as 3 and above terminate, and the analysis considers only ports 1 and 2. The noise-equivalent circuit calculation results in an equivalent noise voltage and current, and their correlation coefficient. These values are:

$$VN2 = \overline{|v_n|^2} \qquad IN2 = \overline{|i_n|^2}$$

$$RHON = \rho_n = \frac{\overline{i_n v_n^*}}{\sqrt{\overline{|i_n|^2}\,\overline{|v_n|^2}}}$$

Equivalent Noise Resistance and Conductance—RN, GN

These measurements are the equivalent resistance and conductance that would generate the equivalent noise voltage and current values at a temperature of $T_0 = 290k$ in a 1 Hz bandwidth (that is, $\Delta f = 1Hz$).

$$RN = R_n = \frac{\overline{|v_n|^2}}{4kT_0\Delta f} \qquad GN = G_n = \frac{\overline{|i_n|^2}}{4kT_0\Delta f}$$

Noise Correlation Impedance and Admittance—ZCOR, YCOR

These measurements represent the equivalent impedance and admittance that you can insert at the input of the noise-equivalent circuit to account for the correlation between the equivalent noise voltage and the current values.

$$ZCOR = \rho_n \sqrt{\frac{|v_n|^2}{|i_n|^2}} = \frac{\overline{i_n^* v_n}}{|i_n|^2} = R_{cor} + jX_{cor}$$

$$YCOR = \rho_n \sqrt{\frac{|i_n|^2}{|v_n|^2}} = \frac{\overline{i_n v_n^*}}{|v_n|^2} = G_{cor} + jB_{cor}$$

ZOPT, YOPT, GAMMA_OPT – Optimum Matching for Noise

The equivalent input noise sources and their correlation make it possible to compute the impedance, admittance, and reflection coefficient values that, if presented at the input of the noisy two-port, result in the best noise performance. These values are:

$$Z_{opt} = \sqrt{\frac{R_n}{G_n} - X_{cor}^2} - jX_{cor} \qquad Y_{opt} = \frac{1}{Z_{opt}} = \sqrt{\frac{G_n}{R_n} - B_{cor}^2} - jB_{cor}$$

$$GAMMA\_OPT = \Gamma_{opt} = \frac{Z_{opt} - Z_{01}}{Z_{opt} + Z_{01}}$$

Noise Figure and Noise Figure Minimum—NF, NFMIN

If you set the input source impedance to ZOPT, the two-port operates with the minimum Noise Figure. The definition of Noise Figure (F) is unusual, because it involves the available gain of the two-port and not its transducer gain. You can express it in the following form:

$$F = 1 + \frac{N_a}{G_a k T_0 \Delta f}$$

- G_a is the available power gain.
- N_a is the available noise power at the output of the two-port (due solely to the two-port's noise and not to the input impedance).

- k is Boltzmann's constant.
- T_0 is the 290 Kelvin reference temperature.

The NMIN minimum noise figure value is computed as:

$$NFMIN = F_{min} = 1 + 2G_n\left(R_{cor} + \sqrt{\frac{R_n}{G_n} - X_{\text{cor}}^2}\right)$$

where NFMIN≥1. For input source impedance values other than `ZOPT`, the Noise Figure varies as a function of the input source reflection coefficient, according to:

$$F = F_{min} + \frac{R_n|\Gamma_S - \Gamma_{opt}|^2}{2Z_{01}|1+\Gamma_{opt}|^2}$$

The HSPICE RF Noise Figure measurement (NF) returns the noise figure value if the input terminates in the port characteristic impedance (that is, $\Gamma_s = 0$). This value is:

$$NF = F_{min} + \frac{R_n|\Gamma_{opt}|}{2Z_{01}|1+\Gamma_{opt}|^2} = F_{min} + \frac{G_n}{Z_{01}}|Z_{01} - Z_{opt}|^2$$

Associated Gain—G_As

HSPICE RF also includes a measurement named Associated Gain, which assumes that the Γ_S inout impedance is matched for the minimum noise figure (that is, $\Gamma_s = \Gamma_{opt}$), while the output is matched for the maximum gain.

$$G_{AS} = \frac{\left|s_{21}\right|^2(1 - \left|\Gamma_{Opt}\right|^2)}{\left|1 - s_{11}\Gamma_{Opt}\right|^2(1 - \left|s'_{22}\right|^2)}$$

In the preceding equation: $s'_{22} = s_{22} + \frac{s_{12}s_{21}\Gamma_{Opt}}{1 - s_{11}\Gamma_{Opt}}$

Extracting Mixed-Mode Scattering (S) Parameters

In HSPICE RF, the .LIN analysis includes a keyword for extracting mixed-mode scattering (S) parameters.

Syntax

```
.LIN … [ mixedmode2port= dd|dc|ds|cd|cc|cs|sd|sc|ss ]
```

The following keywords in a .PRINT and .PROBE statements specifies the elements in the mixed mode S parameter matrices:

```
S|Y|Z<xy>nm<(t)>
```

Argument	Description
x, y	One of the following: ▪ D (differential) ▪ C (common) ▪ S (single-ended) For example: ▪ SCC=common mode S parameters ▪ SDC or SCD=cross mode S parameters If you omit x,y, then HSPICE uses the value set for the mixedmode2port.
S_{cc}	Common-mode S parameters
S_{cd} and S_{dc}	Mode-conversion or cross-mode S parameters
m, n	port number
type	One of the following: ▪ DB: magnitude in decibels ▪ I: imaginary part ▪ M: magnitude (default) ▪ P: phase in degree ▪ R: real part

Defaults

Availability and default value for the `mixedmode2port` keyword depends on the port configuration.

Example 1

```
p1=p2=single
```

Where,

- Available: ss
- Default: ss

Example 2

```
p1=p2=balanced
```

Where,

- Available: dd,cd,dc,cc
- Default: dd

Example 3

```
p1=balanced p2=single
```

Where,

- Available: ds,cs
- Default: ds

Example 4

```
p1=single p2=balanced
```

Where,

- Available: sd,sc
- Default: sd

Output File Formats

An *sc#* file format for the mixed mode:

- The S element model has additional keywords, such as `mixedmodei` and `idatatype`, if the netlist includes one or more balanced ports.
- The `mixedmode2port` keyword prints in the header line.
- The other S Element keywords also appear in the header lines.

Touchstone format for the mixed mode:

The following lines for data mapping are added to the head of the Touchstone output file if the netlist includes one or more balanced ports.

```
! S11=SDD11
! S12=SDD12
! S13=SDC11
! S14=SDC12
```

Two-Port Parameter Measurement

Two-port parameter measurement function takes the first 2 ports, then reads the corresponding parameter with the drive condition specified by the `mixedmode2port` keyword.

Output Format and Description

File Type	Description
*.ac#	Output from both the .PROBE and .PRINT statements.
*.printac#	Output from the .PRINT statement. This is available in HSPICE RF only.
*.sc#	The extracted S parameters/2-port noise parameters are written to a *.sc# file by using the S-element format. If you want to simulate the S element, you can reference the *.sc# file in your netlist.

```
* N=numOfPorts DATA=numOfFreq NOISE=[0|1] GROUPDELAY=[0|1]
* NumOfBlock=numOfSweepBlocks NumOfParam=numOfSweptParameters
* MIXEDMODE=[0|1] DATATYPE=mixedModeDataTypeString
.MODEL mname S
```

```
+ N=numOfPorts FQMODEL=SFQMODEL TYPE=S Zo=*** *** ...
.MODEL SFQMODEL SP N=numberOfPorts SPACING=POI
   INTERPOLATION=LINEAR MATRIX=NONSYMMETRIC
+ DATA= numberOfData
+ freq1
+ s11real s11imag s12real s12imag ... s1Nreal s1Nimag
...
+ sN1real sN1imag ... sNNreal sNNimag
...
...
+ freqNumberOfData
+ s11real s11imag s12real s12imag ...s1Nreal s1Nimag
...
+ sN1real sN1imag ... sNNreal sNNimag

* 2-port noise parameter
* frequency   Fmin [dB]   GammaOpt(M)   GammaOpt(P)   RN/Zo
* 0.10000E+09   0.   1.0000    0.    1.0281
* ...
```

The 2-port noise section starts with “*” so that you can include this file in your HSPICE or HSPICE RF input netlists.

Features Supported

.LIN analysis in HSPICE and HSPICE RF supports the following features:

- Automatic calculation of bias-dependent S, Y, and Z parameters. No additional sources required.
- Automatic calculation of noise parameters.
- Automatic calculation of group delay matrices.

In addition, HSPICE RF supports all existing HSPICE RF models. For noise analysis, HSPICE and HSPICE RF view port 1 as the input and port 2 as the output.

Prerequisites and Limitations

The following prerequisites and limitations apply to `.LIN` analysis in HSPICE RF:

- Requires one `.LIN` statement to specify calculation options.
- Requires one `.AC` statement to specify frequency sweep and parameter sweep.
- Requires at least one P element, numbered from port 1 to N.
- For noise analysis, HSPICE RF views port 1 as the input and port 2 as the output.

Reported Statistics for the Performance Log (HSPICE RF Only)

- Simulation time
  - DC op time
  - Total simulation time
- Memory used
  - Total memory

Errors and Warnings

- If the circuit contains fewer than two P Elements and noisecalc=1, then the 2-port noise calculation is skipped.
- If the circuit contains fewer than two P Elements, does not let you cannot use the `.PRINT`, `.PROBE`, or `.MEAS` command with any two-port noise or gain parameters.
- If the circuit contains more than two P Elements, all two-port parameters are computed. By default, `port=1` is the input and `port=2` is the output. All other ports terminate in their reference impedances.

Example

```
.OPTION POST=2
.AC DEC 1 20MEG 20G
.LIN noisecalc=1
Pout outs gnd port=2 RDC=50 RAC=50 DC=0 AC=1 0
Pin ins gnd port=1 RDC=50 RAC=50 DC=0.5 AC=0.5 0
xlna_2_ ins outs lna
```

```
.subckt lna in out
rhspr5 in _n481 50
rhspr6 _n523 out 100
vvdd _n523 gnd dc=1.8
qhspnpn3 out _n481 gnd gnd bjtm1 area=3
.ends lna
.global gnd
.END
```

.NET Parameter Analysis

HSPICE or HSPICE RF uses the AC analysis results to perform network analysis. The .NET statement defines Z, Y, H, and S parameters to calculate. The following list shows various combinations of the .NET statement for network matrices that HSPICE or HSPICE RF calculates:

```
.NET Vout Isrc   V  =  [Z]    [I]
.NET Iout Vsrc   I  =  [Y]    [V]
.NET Iout Isrc   [V1 I2]^T  =  [H]   [I1 V2]^T
.NET Vout Vsrc   [I1 V2]^T  =  [S]   [V1 I2]^T
([M]^T represents the transpose of the M matrix).
```

Note:

> The preceding list does not mean that you must use combination (1) to calculate Z parameters. However, if you specify .NET Vout Isrc, HSPICE or HSPICE RF initially evaluates the Z matrix parameters. It then uses standard conversion equations to determine S parameters or any other requested parameters.

Figure 57 shows the importance of variables in the .NET statement. Here, Isrc and Vce are the DC biases, applied to the BJT.

Figure 57 Parameters with .NET V(2) Isrc

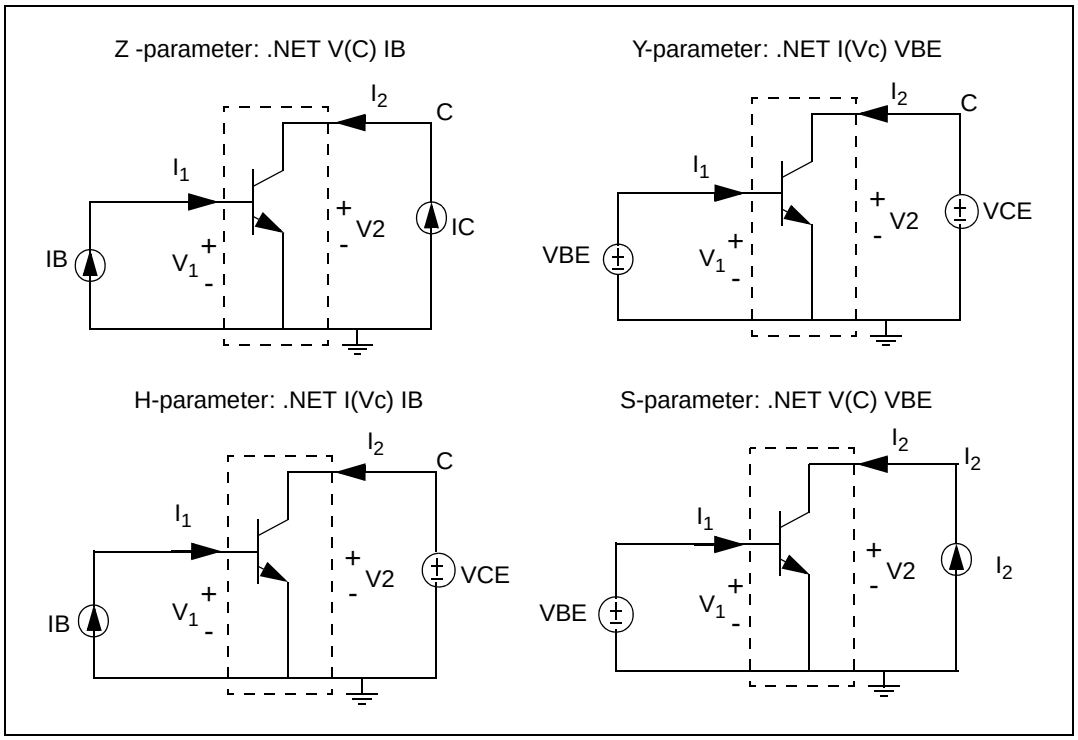
This `.NET` statement provides an incorrect result for the Z parameter calculation:

```
.NET V(2) Isrc
```

When HSPICE or HSPICE RF runs AC analysis, it shorts all DC voltage sources; all DC current sources are open-circuited. As a result, V(2) shorts to ground and its value is zero in AC analysis. This affects the results of the network analysis.

In this example, HSPICE or

Figure 58 Network Parameter Configurations



Example

To calculate the H parameters, HSPICE or HSPICE RF uses the `.NET` statement.

```
.NET I(VC) IB
```

VC denotes the voltage at the C node, which is the collector of the BJT. With this statement, HSPICE or HSPICE RF uses the following equations to calculate H parameters immediately after AC analysis:

$$V1 = H11 \cdot I1 + H12 \cdot V2$$

$$I2 = H21 \cdot I1 + H22 \cdot V2$$

To calculate Hybrid parameters (H11 and H21), the DC voltage source (V_{CE}) sets V2 to zero, and the DC current source (IB) sets I1 to zero. Setting I1 and V2 to zero, precisely meets the conditions of the circuit under examination: the input current source is open-circuited, and the output voltage source shorts to ground.

A data file containing measured results can drive external DC biases applied to a BJT. Not all DC currents and voltages (at input and output ports) might be available. When you run a network analysis, examine the circuit and select suitable input and output variables. This helps you to obtain correctly-calculated results. The following example demonstrates HSPICE network analysis of a BJT or HSPICE RF.

Network Analysis Example: Bipolar Transistor

This example is based on demonstration netlist net_ana.sp, which is available in directory $<installdir>/demo/hspice/bjt:

```
BJT network analysis
.option post nopage list
+ newtol reli=1e-5 absi=1e4 468e/b3*[j.5(li=list)]TJT*r[(+ n)lvdc absi=1e4
```

```
+ trc1=-4.020700e-03 trm1=0.000000e+00
.print ac par('ib') par('ic')
+ h11(m) h12(m) h21(m) h22(m)
+ z11(m) z12(m) z21(m) z22(m)
+ s11(m) s21(m) s12(m) s22(m)
+ y11(m) y21(m) y12(m) y22(m)
.ac Dec 10 1e6 5g sweep data=bias
.data bias
vbe vce ib ic
771.5648m 292.5047m 1.2330u 126.9400u
797.2571m 323.9037m 2.6525u 265.0100u
821.3907m 848.7848m 5.0275u 486.9900u
843.5569m 1.6596 8.4783u 789.9700u
864.2217m 2.4031 13.0750u 1.1616m
884.3707m 2.0850 19.0950u 1.5675m
.enddata
.end
```

Other possible biasing configurat

.NET Parameter Analysis

Xij(*z*), ZIN(*z*), ZOUT(*z*), YIN(*z*), YOUT(*z*)

Parameter	Description
X	In HSPICE or HSPICE RF, can be Z (impedance), Y (admittance), H (hybrid), or S (scattering).
ij	i and j identify the matrix parameter to print in HSPICE or HSPICE RF. Value can be 1 or 2. Use with the *X* value above (for example, S*ij*, Z*ij*, Y*ij*, or H*ij*).
ZIN	Input impedance. For the one-port network, ZIN, Z11, and H11 are the same. (HSPICE or HSPICE RF).
ZOUT	Output impedance. (HSPICE or HSPICE RF).
z	Output type (HSPICE or HSPICE RF): ■ R: real part. ■ I: imaginary part. ■ M: magnitude. ■ P: phase. ■ DB: decibel. ■ T: group time delay (HSPICE RF does not support group time delays in AC analysis output).
YIN	Input admittance. For a one-port network, YIN is the same as Y11. (HSPICE or HSPICE RF).
YOUT	Output admittance. (HSPICE or HSPICE RF).

If you omit *z*, output includes the magnitude of the output variable. The output of AC Analysis includes voltages and currents.

Example

```
.PRINT AC Z11(R) Z12(R) Y21(I) Y22 S11 S11(DB) Z11(T)
.PRINT AC ZIN(R) ZIN(I) YOUT(M) YOUT(P) H11(M) H11(T)
.PLOT   AC S22(M) S22(P) S21(R) H21(P) H12(R) S22(T)
```

Bandpass Netlist: Network Analysis Results

This example is based on demonstration netlist fbpnet.sp, which is available in directory $<installdir>/demo/hspice/filters:

```
file fbpnet.sp network analysis
*
* scattering parameters computations.
* input and output impedance computations.
* computation of frequecy where zin cross 50 ohm.
* computation of phase of zin when zin cross 50 ohm.
*
.options dcstep=1 post
*band pass filter
c1 in 2 3.166pf
l1 2 3 203nh
c2 3 0 3.76pf
c3 3 4 1.75pf
c4 4 0 9.1pf
l2 4 0 36.81nh
c5 4 5 1.07pf
c6 5 0 3.13pf
l3 5 6 233.17nh
c7 6 7 5.92pf
c8 7 0 4.51pf
c9 7 8 1.568pf
c10 8 0 8.866pf
l4 8 0 35.71nh
c11 8 9 2.06pf
c12 9 0 4.3pf
l5 9 10 200.97nh
c13 10 out 2.97pf
rx out 0 1e14
vin in 0 ac 1
.ac lin 250 200meg 300meg
.net v(out) vin rout=50 rin=50
.probe ac s11(db) s11(p) s21(db) s21(p)
.probe ac zin(m) zin(p)
.meas ac cross50 when zin(m)=50 td=230meg
.meas ac phase50 find zin(p) when zin(m)=50 td=230meg
.end
```

Figure 59 S11 Magnitude and Phase Plots

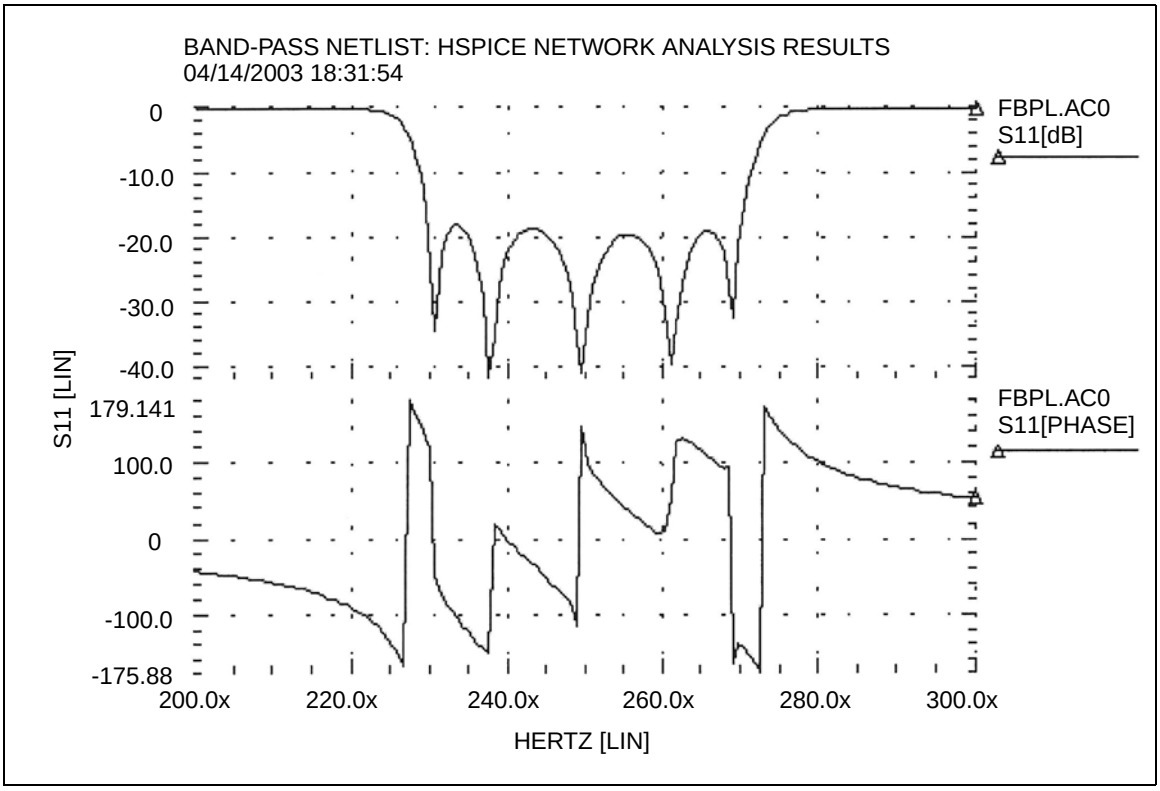
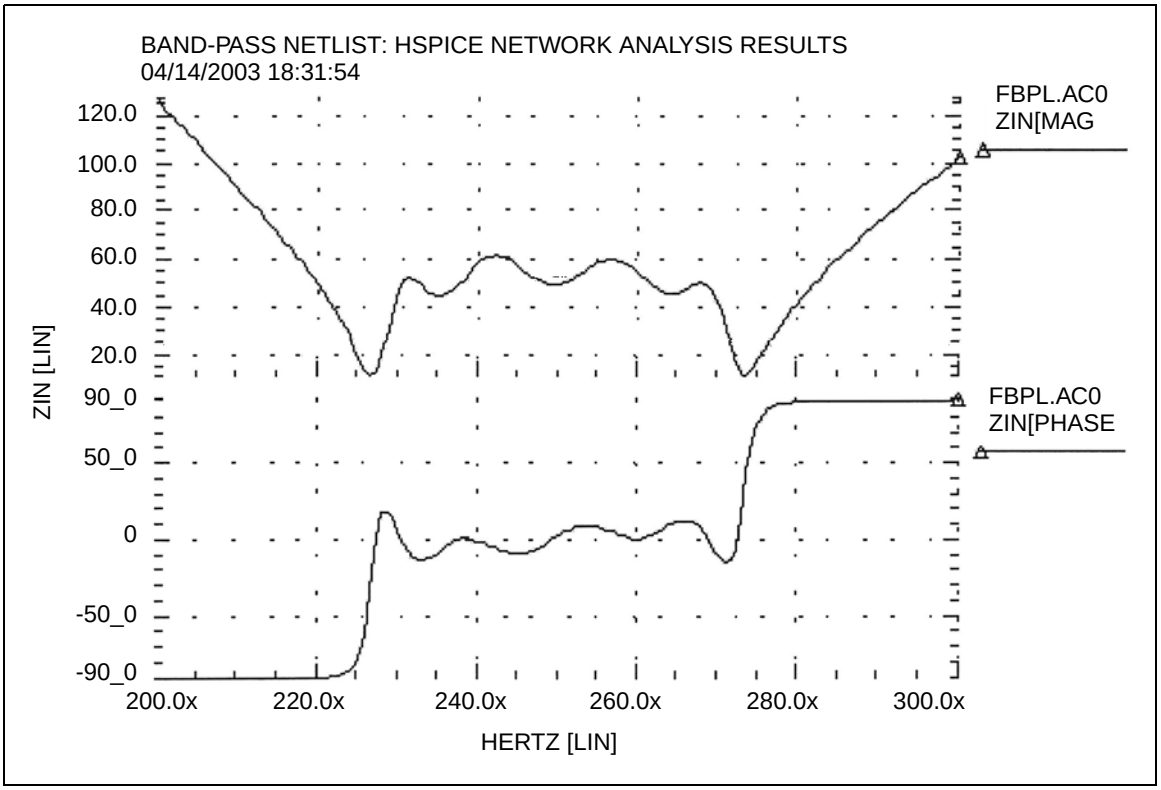


Figure 60 ZIN Magnitude and Phase Plots



References

[1] Goyal, Ravender. “S-Parameter Output From SPICE Program”, *MSN & CT,* February 1988, pp. 63 and 66.

[2] Robert J. Weber "Introduction to Microwave Circuits", IEEE Press.

[3] Behzad Razavi, "Design of Analog CMOS Integrated Circuits", McGraw Hill.

[4] Reinhold Ludwig, Pavel Bretchko, "RF Circuit Design Theory and Applications".

[5] G.D. Vendelin, Design of Amplifiers and Oscillators by the S-Parameter Method, John Wiley & Sons, 1982.

[6] R.S. Carson, High-Frequency Amplifiers, 2nd Edition, John Wiley & Sons, 1982.

[7] G. Gonzalez, Microwave Transistor Amplifiers: Analysis and Design, 2nd Edition, Prentice-Hall, 1997.

[8] M.L. Edwards and J.H. Sinsky, "A single stability parameter for linear 2-port networks," IEEE 1992 MTT-S Symposium Digest, pages 885-888.

[9] H. Rothe and W. Dahlke, "Theory of noisy fourpoles", Proc. IRE, volume 44, pages 811-818, June 1956.

[10] David E. Bockeman, “Combined Differential and Common-Mode Scattering Parameters: Theory and Simulation,” IEEE trans. on MTT Volume 43, Number 7, Jul. 1995.

[11] “Understanding Mixed Mode S parameters,” http://www.si-list.org/files/tech_files/Understandmm.pdf

[12] Robert J. Weber "Introduction to Microwave Circuits", IEEE Press.

[13] Behzad Razavi, "Design of Analog CMOS Integrated Circuits", McGraw Hill.

[14] Reinhold Ludwig, Pavel Bretchko, "RF Circuit Design Theory and Applications."

12

Using Verilog-A

Describes how to use Verilog-A in HSPICE simulations.

Note:

> You can use Verilog-A in both HSPICE and HSPICE RF simulations; therefore, in the context of this chapter, “HSPICE” refers to both HSPICE and HSPICE RF unless noted otherwise.

Verilog-A is used to create and use analog behavioral descriptions that encapsulate high-level behavioral and structural descriptions of systems and components.

The language allows the behavior of each model, or module, to be described mathematically in terms of its ports and parameters applied to an instance of the module. A module can be defined at a level of abstraction appropriate for the model and analysis, including architectural design, and verification. Verilog-A supports both a top-down design as well as a bottom-up verification methodology.

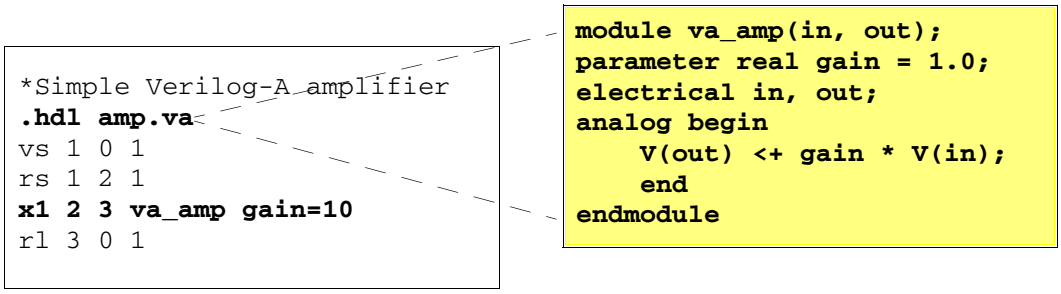
Verilog-A was derived from the IEEE 1364 Verilog Hardware Description Language (HDL) specification and is intended for describing behavior in analog systems. The Verilog-A language that HSPICE supports is compliant with *Verilog-AMS Language Reference Manual, Version 2.2,* with limitations listed in Unsupported Language Features on page 424.

The Verilog-A implementation in HSPICE supports a mixed design of Verilog-A descriptions and transistor-level SPICE netlists with a simple use model. Most analysis features available in HSPICE are supported for Verilog-A based devices, including AC, DC, transient analysis, statistical analysis, and optimization. The HSPICE RF supported analysis types are HB, HBOSC, HBAC, HBNOISE, HBXF, PHASENOISE, and ENV.

Getting Started

This section explains how to get started using a compact device model written in Verilog-A in HSPICE.

Figure 61 HSPICE and Verilog-A



Verilog-A devices use the following conventions:

- modules are loaded into the simulator with either the `.HDL` netlist command or the `-hdl` command-line option (not supported in HSPICE RF).
- modules are instantiated in the same manner as HSPICE subcircuits. The first character for the name of instance should be “X”.
- instance and model parameters can be modified in the same way as other HSPICE instances.
- module names should not conflict with any HSPICE built-in device keyword (see Using Model Cards with Verilog-A Modules on page 410). If this happens, HSPICE issues a warning message and ignores the Verilog-A module definition.
- node voltages and branch currents can be output using conventional output commands.

To run an HSPICE Verilog-A simulation, you need to run the "hspice" script, which is located in the $<installdir>/hspice_2006.03/bin/hspice, regardless of the platform. For example,

```
/installed_hspice/hspice_2006.03/bin/hspice
```

The following example illustrates how a compact device model written in Verilog-A can be analyzed with HSPICE.

Example: JFET Compact Device Model

HSPICE contains a large number of compact device models coded natively in the simulator. Verilog-A provides a convenient method to introduce new

compact models. The JFET device model uses a simple expression to relate the source-drain current to the gate voltage.

The simplified Verilog-A description of this model is shown in below.

```
`include "constants.vams"
`include "disciplines.vams"
module jfet(d, g, s);
parameter real Vto = -2.0 from (-inf:inf);  // Threshold voltage
parameter real Beta = 1.0e-4 from [0:inf);// Transconductance
parameter real Lambda = 0.0 from [0:inf); // Channel modulation
  electrical d, g, s;
  real Id, Vgs, Vds;
  analog begin
      Vgs = V(g,s);
      Vds = V(d,s);
      if (Vds <= Vgs-Vto)
          Id = Beta*(1+Lambda*Vds)*Vds*(2*(Vgs-Vto)- Vds);
      else if (Vgs-Vto < Vds)
          Id = Beta*(1+Lambda*Vds)*(Vgs-Vto)*(Vgs-Vto);
      I(d,s) <+ Id;
  end
endmodule
```

In this example the module name is jfet and the module has three ports, named `d`, `g`, and `s`. Three parameters, `Vto`, `Beta`, and `Lambda`, can be passed in from the netlist. The electrical behavior is defined between the analog begin and end statements. The node voltages across the gate to source and drain to source is accessed and assigned to the variables `Vgs` and `Vgd`. These values are used to determine the drain-source current, `Id`. The calculated current is contributed to the branch from `d` to `s` in the final statement using the contribution operator, `<+`.

This Verilog-A module is loaded into HSPICE with an `.HDL` command in the netlist. The device is then instantiated using the X prefix for the device name. The connectivity, module name, and parameter assignments follow the format of a subcircuit device. The following instantiation line in the netlist is for this device:

```
x1 drain gate source jfet Beta=1.1e-4 lambda=0.01
```

The nodes drain, gate, and source are mapped to the ports d, g, s in the same order as defined in the module definition. Any parameters in the instantiation line are passed to the module; otherwise, the default value defined on the parameter declaration line is used. The parameter declaration allows ranges and exclusions to be easily defined.

```
    @ ( initial_step ) begin
    /*    Code inside an initial_step block is executed
      at the first step of each analysis */
     end

     real_var = I(port1); // Current port1 to ground
     V(bus[0], bus[1]) <+ real_var * real_param * int_param;

     @ ( final_step ) begin
     /* Code inside an final_step block is executed
      at the last step of each analysis */
      end
   end
endmodule
```

Data Types

Four Verilog-A data types are available. The parameter type is used to pass values from the netlist to the module.

Table 47 Verilog-A Data Types

Data Type	Description
attribute	A mechanism for specifying properties about objects, statements, and groups of statements that may be used to control the operation or behavior of the tool. `(* attr_spec {, attr_spec } *)`
genvar	Special integer-valued variable for behavioral looping constructs `genvar genvar_name {, genvar_name};`
integer	Discrete numerical type `integer integer_name {, integer_name};`
local parameters	Identified by the localparam keyword, local parameters are identical to parameters except that they cannot directly be modified with the defparam statement or by the ordered or named parameter value assignment. Local parameters can be assigned to a constant expression containing a parameter that can be modified with the defparam statement or by the ordered or named parameter value assignment.

Table 47 Verilog-A Data Types (Continued)

Data Type	Description
parameter	Attribute that indicates data type is determined at module instantiation. `parameter [{integer | real }] param_name = default_value [ from[ range_begin:range_end ] [ exclude exclude_value_or_range ] ] ;`

parameter aliases	Aliases can be defined for parameters. This allows an alternate name to be used when overriding module parameter values; for example,
	`parameter real dtemp = 0 from [-‘P_CELSIUS0:inf);`
	`aliasparam trise = dtemp;`
	Then the following two instantiations of the module are valid:
	`nmos #(.trise(5)) m1(.d(d), .g(g), .s(s), .b(b));`
	`nmos #(.dtemp(5)) m2(.d(d), .g(g), .s(s), .b(b));`
	And the value of the parameter dtemp, as used in the module equations for both instances, is 5.

real	Continuous numerical type `real real_name {, real_name};`
string parameters	String parameters can be declared. Strings are useful for parameters that act as flags, where the correspondence between numerical values and the flag values may not be obvious. The set of allowed values for the string can be specified as a comma-separated list of strings inside curly braces.

Analog Operators and Filters

Analog operators and filters maintain memory states of past behavior. They can not be used in an analog function.

Table 48 Verilog-A Analog Operators and Filters

Operator	Description
Time derivative	The ddt operator computes the time derivative of its argument. `ddt(expr)`
Time integral	The idt operator computes the time-integral of its argument. `idt(expr, [ic [ , assert [ , abstol ] ] ] )`

Table 48 Verilog-A Analog Operators and Filters (Continued)

Operator	Description
Derivative	The ddx operator provides access to symbolically-computed partial derivatives of expressions in the analog block. `ddx(expr, V(a))`
Linear time delay	absdelay() implements the absolute transport delay for continuous waveform. `absdelay(input, time_delay [, maxdelay ])`

Discrete waveform filters	The transition filter smooths out piecewise linear waveforms.
	`transition(expr[,td[,rise_time[,fall_time [,time_tol]]]])`
	The slew analog operator bounds the rate of change (slope) of the waveform.
	`slew(expr[,max_pos_slew_rate [,max_neg_slew_rate]])`
	The last_crossing() function returns a real value representing the simulation time when a signal expression last crossed zero.
	`last_crossing(expr, direction)`

Laplace transform filters	laplace_zd() implements the zero-denominator form of the Laplace transform filter. The laplace_np() implements the numerator-pole form of the Laplace transform filter. laplace_nd() implements the numerator- denominator form of the Laplace transform filter. laplace_zp() implements the zero-pole form of the Laplace transform filter. `laplace_*(expr, u, v)`
Z-transform filters	The Z-transform filters implement linear discrete-time filters. All Z-transform filters share three common arguments: T, t, and t0. T specifies the period of the filter, is mandatory, and must be positive. t specifies the transition time, is optional, and must be nonnegative. ▪ zi_zd() implements the zero-denominator form of the Z-transform filter. ▪ zi_np() implements the numerator-pole form of the Z-transform filter. ▪ zi_nd() implements the numerator-denominator form of the Z-transform filter. ▪ zi_zp() implements the zero-pole form of the Z-transform filter. `zi_*( expr , u , v , T [ , t [ , t0 ] ] )`

Table 48 Verilog-A Analog Operators and Filters (Continued)

Operator	Description
Limited Exponential	Limits exponential argument change from one iteration to the next. `limexp(arg)`

Mathematical Functions

The following mathematical functions are available when using HSPICE with Verilog-A.

Table 49 Verilog-A Mathematical Functions

Function	Description	Domain	Return Value
ln()	natural log	x>0	real
log(x)	log base 10	x>0	real
exp(x)	exponential	x<80	real
sqrt(x)	square root	x>=0	real
min(x,y)	Minimum of x and y	all x, y	If either is real, returns real, otherwise returns the type of x,y.
max(x,y)	Maximum of x and y	all x, y	If either is real, returns real, otherwise returns the type of x,y.
abs(x)	absolute value	all x	same as x
pow(x,y)	x**y	if x>=0, all y; if x<0, int(y)	real
floor(x)	Floor	all x	real
ceil(x)	Ceiling	all x	real

Transcendental Functions

The following mathematical functions are available when using HSPICE with Verilog-A.

Table 50 Verilog-A Transcendental Function

Function	Description	Domain
sin(x)	sine	all x
cos(x)	cosine	all x
tan(x)	tangent	x != n (pi/2), n is odd
asin(x)	arc-sine	-1<= x <= 1
acos(x)	arc-cosine	-1<= x <= 1
atan(x)	arc-tangent	all x
atan2(x,y)	arc-tangent of x/y	all x, all y
hypot(x,y)	sqrt(x**2 + y**2)	all x, all y
sinh(x)	hyperbolic sine	x < 80
cosh(x)	hyperbolic cosine	x < 80
tanh(x)	hyperbolic tangent	all x
asinh(x)	arc-hyperbolic sine	all x
acosh(x)	arc-hyperbolic cosine	x >= 1
atanh(x)	arch-hyperbolic tangent	-1 <= x <= 1

AC Analysis Stimuli

The AC stimulus function produces a sinusoidal stimulus for use during a small-signal analysis:

```
ac_stim( [ analysis_name [ , mag [ , phase ]]])
```

Noise Functions

The noise functions contribute noise during small-signal analyses. The functions have an optional name, which the simulator uses to tabularize the contributions.

Table 51 Verilog-A Noise Functions

Function	Description
White Noise	Generates a frequency-independent noise of power pwr. `white_noise(pwr[,name])`
Flicker Noise	Generates a frequency-dependent noise of power pwr at 1 Hz which varies in proportion to the expression 1/fexp. `flicker_noise(pwr,exp[,name])`
Noise Table	Defines noise via a piecewise-linear function of frequency. Vector is frequency, power pairs in ascending frequencies. `Noise_table(vector[,name])`

Analog Events

The following analog events are available when using HSPICE with Verilog-A.

Table 52 Verilog-A Analog Event-Controlled Statements

Function	Description
Initial Step	Event trigger at first step of an analysis. `@(initial_step[(list_of_analyses)])`
Final Step	Event trigger at last step of an analysis. `@(final_step[(list_of_analyses)])`

Table 52 Verilog-A Analog Event-Controlled Statements (Continued)

Function	Description
Cross	Zero crossing threshold detection. `cross(expr[,dir[,time_tol[,expr_tol]]]);`
Timer	Generates analog event at specific time. `timer(start_time[,period[,time_tol]]);`
Above	Generates an event when a specified expression becomes greater than or equal to zero. `above(expr[,time_tol[,expr_tol]]);`

Timestep and Simulator Control

These functions provide a mechanism to alert the simulator to discontinuities or to limit the time step.

Table 53 Verilog-A Discontinuity and Time Step Limit Functions

Function	Description
Bound time step	Controls the maximum time step the simulator takes during a transient simulation. `$bound_step( expression );`
Announce discontinuity	Provides the simulator information about known discontinuities to provide help for simulator convergence algorithms. `$discontinuity [ ( constant_expression ) ] ;`

System Tasks and I/O Functions

System functions provide access to system-level tasks as well as access to simulator information.

Table 54 Verilog-A System Tasks and I/O Functions

Function	Description
$param_given	Returns 1 if the parameter was overridden by a module instance parameter value assignment and 0 otherwise.

Table 54 Verilog-A System Tasks and I/O Functions (Continued)

Function	Description
$table_model	Models behavior of a system by interpolating between data points that are samples of that system's behavior.
$strobe $display $write	Displays simulation data when the simulator has converged on a solution for all nodes using a printf() style format.
$strobe(args); $fopen	Opens a file for writing and assigns it to an associated channel. `multi-channel_desc = $fopen("file");`
$fclose	Closes a file from a previously-opened channel(s). `$fclose(multi-channel_descriptor);`
$fstrobe $fdisplay $fwrite	Writes simulation data to an opened channel(s) when the simulator has converged. Follows format for $strobe. `$fstrobe(multi-channel_descriptor, "information to be written");`
$dist_functions	Probabilistic distribution functions
$debug	Provides the capability to display simulation data while the analog simulator is solving the equations.
$random	Provides a mechanism for generating random numbers. `random_function ::= $random [ ( seed [, type_string] ) ] ;`

Simulator Environment Functions

The environment parameter functions return simulator environment information to the module. Return circuit ambient temperature in Kelvin.

Table 55 Verilog-A Environment Parameter Functions

Function	Description
Circuit temperature	Returns circuit ambient temperature in Kelvin. `$temperature`

Table 55 Verilog-A Environment Parameter Functions (Continued)

Function	Description
Time	Returns absolute time in seconds. `$abstime`
Thermal voltage	$vt can optionally have Temperature (in Kelvin) as an input argument and returns the thermal voltage (kT/q) at the given temperature. $vt without the optional input temperature argument returns the thermal voltage using $temperature. `$vt [ (Temperature) ]`
Analysis flag	Returns true (1) if current analysis matches any one of the passed arguments. `$analysis(str {, str } )`
Simulation parameter	Returns the value of the named specified simulation parameter. `gmin = $simparam("gmin", 1.0);`

Module Hierarchy

Modules can instantiate other modules so that networks of modules can be constructed. Structural statements are used inside the module block but cannot be used inside the analog block.

```
module_name #({.param1(expr){, .param2(expr})}
instance_name ({node {, node});
```

Example

```
my_src #(.fstart(100), .ramp(z)) u1 (plus, minus);
```

Parameter Sets

Parameter sets (paramsets) bring the concept of model cards directly into Verilog-A. Paramsets allow sharing of a set of parameters among several modules. They may also be chained allowing a common parameter set to be used.

Example

```
paramset nch my_nmos; // default paramset
parameter real l=1u from [0.25u:inf);
parameter real w=1u from [0.2u:inf);
```

```
.l=l; .w=w; .ad=w*0.5u; .as=w*0.5u;
.level=3; .kp=5e-5; .tox=3e-8; .u0=650; .nsub=1.3e17;
.vmax=0; .tpg=1; .nfs=0.8e12;
endparamset
```

Simulation with Verilog-A Modules

When simulating with Verilog-A in HSPICE, you need to have the following basic input files:

- HSPICE netlist/model card (Mandatory)
- Verilog-A model file (for example, **.va* or **.vams* file) or Compiled Model Library file (**.cml*) (Mandatory)
- HSPICE Verilog-A feature setup options (Optional, but mandatory under certain conditions)

Basic output files:

- HSPICE standard output files
- The **.val* file, Verilog-A log file, which contains Verilog-A specific message from compiling and simulating phase. The contents of **.val* file is also echoed to the **.lis* file.
- Compiled Verilog-A code (*.cml* file) (when Verilog-A modules are compiled manually).

Loading Verilog-A Devices

This section describes loading Verilog-A modules into HSPICE and specifying cell names for Verilog-A definitions. A module must be loaded before it can be instantiated.

Verilog modules are loaded into HSPICE in one of two ways:

- by including an `.HDL` statement in an HSPICE netlist
- by using the `-hdl` command-line option (not supported in HSPICE RF).

Files can be in the current directory or specified via an absolute or relative path. The Verilog-A file is assumed to have a **.va* extension when only a prefix is provided. For example, `.hdl "model"` looks for a model.va file and *not* a file named “model”.

Use the `-vamodel` command-line option to specify cell names for Verilog-A definitions (not supported in HSPICE RF).

For a description of the `.hdl` statement, see the .HDL command in the *HSPICE Command Reference*. For a description of the `-hdl` and `-vamodel` command-line options, see "HSPICE Command Syntax" in the *HSPICE Command Reference*.

Verilog-A File Search Path

During a simulation, HSPICE searches in the current directory for Verilog-A files. You can also provide a search path via either the `-hdlpath` command-line option (not supported in HSPICE RF) or the `HSP_HDL_PATH` environment variable to have HSPICE search other directories for the files. The `-hdlpath` HSPICE command-line option is provided for HSPICE Verilog-A use only, which defines the search path specifically for Verilog-A files.

For a description of the `-hdlpath` command-line option, see "HSPICE Command Syntax" in the *HSPICE Command Reference*.

When a Verilog-A file cannot be found in the current working directory or the directory defined by `-hdlpath`, or there is no `-hdlpath` defined, HSPICE searches directory defined by `HSP_HDL_PATH` for the Verilog-A file.

The directory search order for Verilog-A files is:

1. Current working directory
2. Path defined by `-hdlpath`
3. Path defined by `HSP_HDL_PATH`

The path defined by either `-hdlpath` or `HSP_HDL_PATH` can consist a set of directory names. The path separator must follow HSPICE conventions or platform conventions (";" on UNIX). Path entries that do not exist are ignored and no error or warning messages are issued.

Example

This example assumes the c-shell is used.

```
setenv HSP_HDL_PATH  /lib_path/veriloga
```

Verilog-A File Loading Considerations

Several restrictions and issues must be considered when loading Verilog-A modules:

- You can place an `.HDL` statement anywhere in the top-level circuit. All Verilog-A modules are loaded into the system prior to any device instantiation.
- An `.HDL` statement is not allowed inside a `.subckt` or `IF-ELSEIF-ELSE` block; otherwise, the simulation will exit with an error message.
- When a module to be loaded has the same name as a previously-loaded module, or the names differ in case only, the latter one is ignored and the simulator issues a warning message.
- If a Verilog-A module file is not found or the Compiled Model Library (CML) file has an incompatible version, the simulation exits and an error message is issued.

Instantiating Verilog-A Devices

Verilog-A devices are X elements. A Verilog-A device can have zero or more nodes and can accept zero or more parameter assignments. Verilog-A devices also support the concept of a model card. In either instance statements or model card statements, invalid parameters that are not predefined in the Verilog-A module file are ignored. HSPICE issues a warning message on those invalid parameters.

Syntax

```
X<inst> <nodes> moduleName|ModelName param=value
```

Verilog-A module definitions are unique in each HSPICE simulation. A Verilog-A module that matches the name, or differs only in case of a previously loaded module is ignored. A Verilog-A module definition is ignored if its name conflicts with HSPICE built-in models.

For any X element, the default search order to find the cell definition is:

1. HSPICE subcircuit definition
2. Verilog-A model card
3. Verilog-A module definition

Example

Suppose you have the following HSPICE netlist fragment:

```
.hdl "mydiode"
X1 a b mydiode
.model mydiode D …
```

In this example, the simulation fails even though the Verilog-A module `mydiode` is loaded. The reason is that the simulator finds the model card mydiode first, which is an HSPICE built-in 'D' model—not the Verilog-A model the X1 statement is trying to locate.

Using Model Cards with Verilog-A Modules

The HSPICE Verilog-A device supports the concept of model cards, with similar usage to HSPICE standard built-in devices. The Verilog-A module name should not conflict with the following built-in device keywords. In the event of a conflict, HSPICE issues a warning message and ignores the module definition.

AMP, C, CORE, D, L, NJF, NMOS, NPN, OPT, PJF, PLOT, PMOS, PNP, R, U, W, SP

The model card type should be the same as the Verilog-A module name. Every Verilog-A module can have one or more associated model cards.

Unlike built-in device model cards and instances, you can specify any module parameter in Verilog-A model cards, instance statements, or inherited parameter values from module definitions. Instance parameters always override model parameters.

If the model card includes parameters that are not predefined in its associated module file, HSPICE issues a warning message, ignores the definition, and continues with the simulation.

Syntax

```
.model mname type <pname1= > <pname2= > <pname3= > …
```

Argument	Description
mname	User defined model name reference. Elements must use this name to refer to this model card.
type	Model type, it must be the same as Verilog-A module name.

Argument	Description
pname#	Parameter name. Every parameter must be predefined in its associated Verilog-A module with default parameter value set. For legibility, use either blanks or commas to separate each assignment. Use a plus sign (+) to start a continuation line.

Example

For the following examples, assume the following Verilog-A module is used:

```
module va_amp(in, out);
electrical in, out;
input in;
output out;
parameter real gain=1.0;
parameter real fc=100e6;
...
analog begin
...
```

Its associated model cards can then be:

```
.model myamp va_amp gain=2 fc=200e6
.model myamp2 va_amp gain=10
```

The instantiations of Verilog-A module va_amp are:

```
x1 n1 n2 myamp
x2 n3 n4 myamp gain=3.0
x3 n5 n6 myamp gain=2.0 fc=150e6
x4 n7 n8 myamp2 fc=300e6
x5 n9 n10 va_amp
```

- Instance x1 inherits model myamp parameters (that is, `gain=2`, `fc=200e6`).
- Instance x2 inherits "`fc=200e6`" from model myamp, but overrides "`gain`" with the value 3.0.
- Instance x3 overrides all model myamp parameters.
- Instance x4 inherits parameter "`gain=10`" from model myamp2, and overrides parameter "`fc`", which is an implicit parameter in myamp2.
- Instance x5 does not use a model card and directly instantiates the Verilog-A module va_amp and inherits all module va_amp default parameters, which are "`gain=1`" and "`fc=100e6`".

Restrictions on Verilog-A Module Names

Verilog-A module name cannot conflict with certain HSPICE built-in device keywords. If a conflict occurs, HSPICE issues a warning message and the Verilog-A module definition is ignored.

The following built-in device keywords cannot be used as Verilog-A module names: AMP, C, CORE, D, L, NJF, NMOS, NPN, OPT, PJF, PLOT, PMOS, PNP, R, U, W, SP

Overriding Subcircuits with Verilog-A Modules

If both a subcircuit and a Verilog-A module have the same case-insensitive name, by default, HSPICE uses the subcircuit definition. This behavior can be changed by setting `vamodel` options, either at the command line or in a `.OPTION` statement. The `vamodel` options are not supported in HSPICE RF.

The `VAMODEL` option works on cell-based definitions only. Instance-based overriding is not supported.

Netlist Option

Syntax

```
.OPTION vamodel[=name]
```

This option is not supported in HSPICE RF. The `name` is the cell name that uses a Verilog-A definition rather than the subcircuit when both exist. Each `vamodel` option can take no more than one name. Multiple names need multiple `vamodel` options.

If no name is provided for the `vamodel` option, HSPICE uses the Verilog-A definition whenever it is available.

Example 1

```
.option vamodel=vco
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco`.

Example 2

```
.option vamodel=vco vamodel=chargepump
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco` and cell `chargepump`.

Example 3

```
.option vamodel
```

This example instructs HSPICE to always use the Verilog-A definition whenever it is available.

Command-line Option

Syntax

```
-vamodel <name> -vamodel <name2> …
```

This command-line option is not supported in HSPICE RF. The `name` is the cell name that uses a Verilog-A definition rather than subcircuit when both are exist. Each command-line `-vamodel` option can take no more than one name. Repeat `-vamodel` if multiple Verilog-A modules are defined.

If no `name` after `-vamodel` is supplied, then in any case the Verilog-A definition, whenever it is available, overrides the subcircuit.

The following examples show various ways to set the option and the resulting HSPICE behavior.

Example 1

```
hspice pll.sp –vamodel vco
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell vco.

Example 2

```
hspice pll.sp –vamodel vco –vamodel chargepump
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco` and cell `chargepump`.

Example 3

```
hspice pll.sp –vamodel
```

This example instructs HSPICE to always use a Verilog-A definition whenever it is available.

Disabling .OPTION vamodel with .OPTION spmodel

These options are not supported in HSPICE RF. Use the `.OPTION spmodel` netlist option to switch back to the HSPICE definition. For example, if you override the HSPICE definition with the Verilog-A definition using `.OPTION`

`vamodel`, use `.OPTION spmodel` during `.ALTER` analysis to revert to the HSPICE definition, which is the same as the `VAMODEL` option. The `SPMODEL` option works on cell-based definitions only. Instance-based overriding is not supported.

Syntax

```
.OPTION spmodel[=name]
```

The name is the cell name that will use spice definition. Each `spmodel` option can take no more than one name; multiple names need multiple `spmodel` options.

Example 1

```
.option spmodel
```

This example disables the previous `.OPTION vamodel`, but has no effect on the other `vamodel` options if they are specified for the individual cells. For example, if `.option vamodel=vco` is set, the cell of vco uses the Verilog-A definition whenever it is available.

Example 2

```
.option spmodel=chargepump
```

This example disables the previous `.option vamodel=chargepump`, which causes all instantiations of `chargepump` to now use the subcircuit definition again.

Using Vector Buses or "Ports"

The Verilog-A language supports the concept of buses (vector ports), whereas HSPICE does not. If you instantiate a module that has a vector port, the connections to individual bus signals in the HSPICE netlist must be specified. The Verilog-A module internally expands the vector port and connects them to the signals inside the Verilog-A module.

Example

Given a Verilog-A module with a vector port defined:

```
module d2a(in, out);
   electrical [1:4] in;
   electrical out;
      analog ...
```

Its instantiation in HSPICE could be:

```
x1 in1 in2 in3 in4 o1 d2a
```

In this case, the nodes in1 through in4 are mapped to ports in[1] -> in[4], respectively. If the bus in Verilog-A module is specified as electrical [4:1], then the signals would be connected as in1 -> in4 to in[4] -> in[1], respectively.

Using Integer Parameters

HSPICE netlist parameters are all of type real. When an integer Verilog-A parameter is assigned a real value, it is coerced to an integer value.

Implicit Parameter M Support

Verilog-A supports the multiplicity factor. A Verilog-A device can have parameter that is not device specific:

```
M          Multiplicity factor
```

If a loaded Verilog-A module has parameter with the name of either “M” or “m”, then that module parameter cannot be set in the instance line. The “M” or “m” parameter in the instance line always means the "Multiplicity factor" parameter and the appropriate multiplicity factor is applied to the Verilog-A device during the simulation. The implicit device parameter scaling factor S and the temperature difference between the element and circuit, `DTEMP`, are not supported.

Module and Parameter Name Case Sensitivity

Verilog-A is case-sensitive, whereas HSPICE is case-insensitive. This places certain restrictions on use in terms of module and parameter names and output control.

Module Names

When an attempt to load a second module into the system with a module name that differs from a previously loaded module by case only, then the second module is ignored and a warning message is issued.

Module Parameters

Parameters in the same module with names that only differ by case cannot be redefined in either Verilog-A instance line or Verilog-A .MODEL cards. HSPICE issues an error message and exits the simulaton.

Example

In this example a simple amplifier accepts two parameters, gain and Gain, as input to the module.

```
module my_amp(in, out);
   electrical in, out;
   parameter real gain = 1.0;
   parameter real Gain = 1.0;
   analog V(out) <+ (Gain+gain)*V(in);
endmodule
```

If you instantiate this module as:

```
x1 n1 n2 my_amp Gain=1
```

HSPICE cannot uniquely define the Gain parameter, so a warning message is issued and the definition of Gain is ignored. This module can be instantiated as is, provided neither the Gain nor gain parameter is assigned in the netlist.

Output Simulation Data

Verilog-A devices support the same output capabilities as built-in devices. You can access the following Verilog-A device quantities via any of these HSPICE output statements: .PRINT, .PLOT, .PROBE, .GRAPH, .DOUT, and so forth.

- Port current
- Port voltage
- Internal node voltage (HSPICE only)
- Internal named branch current (HSPICE only)
- Internal module variables (HSPICE only)
- Module parameters (HSPICE only)

V() and I() Access Functions

You can access port voltage and internal node voltage of Verilog-A devices via the V() function. Port current and internal branch currents can be accessed via the I() function.

The internal nodes of Verilog-A devices are accessible via the V() function when the full hierarchical name is provided. The port current and named branches (on the instance base only) can be accessible via the I() function.

Examples:

For the following examples, assume the Verilog-A module definition fragment is:

```
module va_fnc(plus, minus);
inout plus, minus;
electrical plus, minus;
electrical int1, int2;
branch (int1, int2) br1;
  //creates an internal branch br1 between internal
  //nodes int1 and int2;

analog begin
…
```

And the Verilog-A module may be instantiated in the netlist as:

```
x1 1 2 va_fnc
```

To print the current on Verilog-A device port name plus for the instance x1:

```
.print I(x1.plus)
```

The `plus` is the port name defined in Verilog-A module, not the netlist node name.

To print the Verilog-A module internal node named int1 for the instance x1:

```
.print V(x1.int1)
```

If the `va_fnc` module is hierarchical and has a child instance called c1 with an internal node int1 then the node int1 can be output as

```
.print V(x1.c1.int1)
```

That is, the full HSPICE instance name is concatenated with the full internal Verilog-A instance name to form the complete name.

During compilation of Verilog-A modules, the compiler optimizes some internal branches out of the system such that these branches are not available for output. HSPICE Verilog-A provides a compilation environment variable, `HSP_VACOMP_OPTIONS`, with `-B` option being set, all internal named branches in Verilog-A modules become accessible. However, making all internal branches accessible may have negative impact on simulation performance; turn on the option only when necessary.

Refer to section “Setting Environment option for HSPICE Verilog-A Compiler” for examples of setting `HSP_VACOMP_OPTIONS`.

After `HSP_VACOMP_OPTIONS -B` is set, you can probe branch current with HSPICE output commands. In the previous Verilog-A module, there is an internal branch name br1 declared. To probe the branch current

```
.print I(x1.br1)
```

Output Bus Signals

Verilog-A bus signals can be accessed with HSPICE output commands using the Verilog-A naming and accessing conventions.

Example

Given an example Verilog-A module:

```
module my_bus(in, out);
electrical in;
electrical [1:4] out;
…
```

And instantiated in the netlist as

```
x1 1 2 3 4 5 my_bus
```

then the values of the vector port out can be output by explicitly listing each position.

```
.print v(x1.out[1]), v(x1.out[2]), v(x1.out[3]), v(x1.out[4])
```

Bus elements can also be specified using wildcards, as described in the section Using Wildcards in Verilog-A (HSPICE only) on page 420.

Output Internal Module Variables (HSPICE only)

Verilog-A internal variables, by default, are hidden from output. However, module variables with a description or units attribute, or both, are known as output variables, and HSPICE provides access to their values; for example, suppose a module for a MOS transistor with the following declaration at module scope provides the output variable `cgs`:

```
(* desc="gate-source capacitance", units="F" *) real cgs;
```

The cgs module variable can be printed just like a normal parameter variable. In addition, HSPICE Verilog-A provides a compilation environment variable `HSP_VACOMP_OPTION`, with `-G` option being set, you can use HSPICE output command to access internal module variables of Verilog-A instances.

Syntax

```
Instance:internal_variable
```

Example

```
.print xva_vco:freq
```

This example outputs internal variable frequency value of Verilog-A instance xva_vco.

Output Module Parameters (HSPICE only)

You can use HSPICE output commands to output parameter values for Verilog-A instances.

Syntax

```
Instance:parameter
```

Example

```
.print xva_1:gain
```

This example outputs the `gain` parameter value for the `xva_1` Verilog-A instance.

Case Sensitivity in Simulation Data Output

When Verilog-A information is output via the HSPICE output commands, the case of the node names associated with the quantities to be output is ignored.

Contributions from the Verilog-A noise sources that have the same name when case is ignored are combined.

Example

```
I(d,s) <+ white_noise(4*k*T/R1, "thermalnoise");
I(d2,s2) <+ white_noise(4*k*T/R2, "ThermalNoise");
```

The two noise contributions are combined into one contribution called `thermalnoise` in the output files.

Using Wildcards in Verilog-A (HSPICE only)

Verilog-A names support the use of wildcards to simplify using the output commands.

Examples:

Given the Verilog-A module,

```
module test(p,n);
electrical p,n;
electrical int1, int2;
…
```

instantiated as

```
x1 1 2 test
```

then all of the internal nodes (in this case `int1` and `int2`) can be printed using the command:

```
.print v(x1.*)
```

All indices of a bus in the module:

```
module my_bus(in, out);
electrical in;
electrical [1:4] out;
…
```

Can be specified as:

```
x1 1 2 3 4 5 my_bus
.print v(x1.out[*])
.print v(x1.*)
```

Both of the internal nodes, `int1` and `int2` for the child `ch1` in the instance `x_par1` can be specified using

```
.print v(x_par1.ch1.int*)
```

The HSPICE `.OPTION POST` command does not output internal nodes from Verilog-A modules. Use the wildcard feature to specify a Verilog-A instance if you need to output all internal nodes.

Port Probing and Branch Current Reporting Conventions

When printing and reporting currents for Verilog-A devices, HSPICE follows the same conventions when specifying the direction of current flow as in built-in devices. A positive branch current implies that current is flowing into the device terminal or internal branch.

Unsupported Output Function Features

The following output functions are not supported in this release:

- Port probing: `In( )`, where `n` is the node number). Instead, you can use `I(instance.port_name_in_module)`.
- `Iall()`: Instead, you can output all the terminal currents using a wild card.
- `Isub()`: This is not applicable to Verilog-A components.
- `P()` and `Power()`: Instead, you can use the `$strobe` Verilog-A function .
- Nodal capacitance
- Group delay

Using the Stand-alone Compiler

Verilog-A modules used in HSPICE simulations are automatically compiled and cached by the simulator. You can compile files manually if you wish (to check syntax for example). The Verilog-A compiler takes a Verilog-A file as an input and produces a Compiled Model Library (CML) file, which is a platform-specific shared library.

Example 1

```
hsp-vacomp resistor.va
```

The Verilog-A compiler, `hsp-vacomp`, compiles the Verilog-A module file resistor.va, and produces a CML file resistor.cml in the same directory.

You can include the CML file in the same manner as the Verilog-A file in an HSPICE netlist.

Example 2

```
.hdl "resistor.cml"
```

Note:

When a CML file is specified in the load command the compiler is never invoked, even if the source file is modified.

Setting Environment Option for HSPICE Verilog-A Compiler

While Verilog-A modules are automatically compiled in HSPICE simulation, you can set environment variable `HSP_VACOMP_OPTIONS` to control compiler options from default setting.

Example 1

```
setenv HSP_VACOMP_OPTIONS –G
```

When `–G` is set, all internal variables are accessible for output.

Example 2

```
setenv HSP_VACOMP_OPTIONS –B
```

When `–B` is set, all internal named branches are accessible for output.

The Compiled Model Library Cache

The HSPICE Verilog-A solution provides the performance of a compiled solution without the need for user intervention. The first time a Verilog-A source file is loaded, or after a Verilog-A source file is modified, the system automatically invokes the compiler. The Compiled Model Library (CML) is automatically cached and subsequent simulations use this cached file and bypass the compilation process. Although for the most part you do not need to be concerned with the cache mechanism, you can control some aspects. You can change the cache location, prevent caching, or delete the cache.

Cache Location

By default the cache directory is located in your $HOME directory under the hidden directory .hsp-model-cache

This directory holds a directory structure that indicates compiler version, platform, and model directory.

Example

Given the load command

```
.hdl "/users/finn/modules/amp.va"
```

the compiler generates a CML file in

```
/users/finn/.hsp-model-
cache/1.30/users/finn/modules/lib.<arch>/amp.cml
```

Where `<arch>` is one of hpux, sun, linux, and so on.

The location of the cache can be changed from the default value by setting the environment variable `HSP_CML_CACHE` to an appropriate location.

Example

The following sets the environment variable `HSP_CML_CACHE` so that the model cache is created under the my_local_cache directory.

```
setenv HSP_CML_CACHE /users/finn/my_local_cache
```

If the previous example were now simulated the CML file would be

```
/users/finn/my_local_cache/1.30/users/finn/modules/lib.<arch>
/amp.cml
```

Deleting the Cache

The cache structure is maintained unless you choose to delete it manually. You can do this any time; HSPICE automatically recreates the cache when needed. One reason to delete the cache is if a newer version of the HSPICE Verilog-A compiler is used and the previous cache is no longer necessary. The cache can be deleted using conventional operating system commands.

Example

To delete the default cache, from the operating system command prompt, execute

```
rm –r ~/.hsp-model-cache
```

Unsupported Language Features

The following Verilog-A LRM 2.1 Language Features are not supported.

- Escaped identifiers

```
real \bus+index; // Not supported
```

- Derived natures described in LRM 2.1 section 3.4.1.1

  For example, the following deriving the nature New_curr from Ttl_curr is not supported.

```
nature Ttl_curr
   units = "A" ;
   access = I ;
abstol = 1u ;
endnature
// The derived nature is not supported:
nature New_curr : Ttl_curr
   abstol = 1m ;
   maxval = 12.3 ;
endnature
```

- Input, output, and inout enforcement described in LRM 2.1, section 7.1.

```
module test(in,out);
   electrical in,out;
   input in;
   output out;
   real out_value;
   analog begin
      out_value = 1.0;
      V(in) <+ out_value;   // Input node used as output
         // is not prevented, V(in) will be
         // assigned to out_value
   end
endmodule
```

- The `defparam` statement as described in LRM 2.1, section 7.2.1

  For example:

```
module rc(n1, n2);
   electrical n1, n2;
   my_res r1 (n1, n2);
   my_cap c1 (n1, n2);
endmodule
module  my_res(n1, n2);
   electrical n1, n2;
```

```
   parameter dev_temp = 27;
   parameter res = 50;
   parameter tcr = 1;
   analog
      V(n1,n2) <+ I(n1,n2)*res*tcr*($temperature-dev_temp);
endmodule
module  my_cap(n1, n2);
   electrical n1, n2;
   parameter dev_temp = 25;
   parameter cap = 1;
   parameter tcc = 1;
   analog
      I(n1,n2) <+ cap*ddt(V(n1,n2))*tcc*($temperature
      dev_temp);
endmodule
// defparam statement not supported
module annotate;
defparam
   rc.r1.dev_temp = 30;
   rc.c1.dev_temp = 25;
endmodule
```

- Ordered parameter lists in hierarchical instantiation as described in LRM 2.1, section 7.2.2.

  For example:

```
module module_a(out,out2);
   electrical out,out2;
   parameter real value1 = -10.0;
   parameter real value2 = -20.0;
   analog begin
      V(out) <+ value1;
      V(out2) <+ value2;
   end
endmodule
module test_param_by_order(out,out2);
   electrical out,out2;
   parameter real value1 = -1.0;
   parameter real value2 = -2.0;
   // Ordered parameter lists are not supported:
   module_a #(1,2) A1(out,out2);
   // instead use:
   module_a #(.value1(1),.value2(2)) A1(out,out2);
endmodule
```

- Hierarchical and out-of-module-references as described in the LRM 2.1, section 7.

  In this example, the reference to `example2.net` inside the `example1` module is not supported.

```
module example1;
   electrical example2.net; // Feature not supported
endmodule
module example2;
   electrical net;
endmodule
```

- Vector ports, where the port expression defining the size of a port is a parameter expression, as described in LRM 2.1 section 7.3.1.

  In this example, the vector port range size must be a constant—not a parameter value.

```
module test(out);
   parameter integer size = 7 from [1:16];
   electrical [0:size] out; // Feature not supported
   analog begin
      V(out[0]) <+ 0.0;
   end
endmodule
```

- The `` `timescale `` directive, as described in LRM 2.2, section 3.2.3.

```
`timescale 1ns/10ps  // Feature not supported
```

- The `$monitor` function, as described in LRM 2.1, section 10.6.

```
$monitor("\nEvent occurred."); // Feature not supported
```

- Parameters used to specify ranges for the `generate` statement, as described in LRM 2.1, section C.19.3.

```
generate indexr_identifier (start_expr,end_expr[,incr_expr ])
```

- Time tolerances on timer() and transition() functions, as described in LRM 2.1, section 6.5.7.3 and 4.4.9.1, respectively.

```
timer (start_time [, period [, time_tol ] ] ) ;
transition(expr[,td [,rise_time [,fall_time [,time_tol ] ] ] ])
```

- The `` `default_discipline `` directive, as described in LRM 2.1, section 11.1.

In the following example, the ports in and out must have their discipline explicitly declared.

```
`include "disciplines.vams"
`default_discipline electrical  // Feature not supported
module test(in,out);
// in,out default to electrical
   analog
      V(out) <+ I(in);
endmodule
```

- Access to HSPICE primitives from a Verilog-A module, as described in LRM 2.1, section E.2.

```
module spice_rc (p1,p2);
   electrical p1, p2;
   capacitor #(.c(3p)) C3 (p1, p2); // Feature not
         // supported
   resistor #(.r(1k)) R1 (p1, p2);  // Feature not
         // supported
endmodule
```

- "random type-string" feature (10.2 - 10.3).
- reg-strings as described in Section 2.6 of LRM.
- String parameters are supported only from other Verilog-A modules, but not from the HSPICE netlist level.
- `$mfactor`, `$xposition`, `$yposition`, `$angle`, `$hflip`, `$vflip` functions.
- “paramset” instantiation from HSPICE netlists is supported, but instantiation from hierarchical Verilog-A modules is not supported.
- Output variables and string parameters on paramsets.
- `$port_connected` function.
- `$limit` function.
- The following are limitations in HSPICE RF Verilog-A only:
  - `$strobe` in DC analysis
  - `$simparam` simulation parameter
  - @(final_step)
  - 0 port module

- Delays (`absdelay()`), event-controlled constructs, memory states (variables that hold their value between timesteps), and explicit time-dependent functions are not supported in RF analyses.

Known Limitations

This section describes the known limitiations when using Verilog-A with HSPICE.

analysis() Function Behavior

The `analysis()` function definition assumes that the operating point (OP) analysis associated with any user-specified analysis is unique to that user-specified analysis.

For example, when you specify the following function, it must return 1 for AC analysis and 1 for its underlying operation point (OP) analysis.

```
analysis("ac")
```

Similarly, `analysis("tran")` must return 1 for transient analysis and 1 for its underlying OP analysis.

In HSPICE, a single "common" OP analysis is performed in the setup that is outside the context of AC, transient, or other analyses. Since that OP is outside the context of the user-specified analysis, the analysis() function does not know the parent analysis type (during the OP analysis). The analysis("ac"), analysis("tran"), and so on, returns 0 during this “common” OP analysis. You can ensure that the analysis function returns true (1) during these analyses by adding “static” to the list of functions.

Example

```
if ( analysis("ac") )
begin
   // do something
end
```

Should be written as:

```
if( analysis("ac", "static") )
begin
   // do something
end
```

The same is true for the “tran” and “noise” analysis names.

13

Simulating Variability

Introduces variability, describes how it can be defined in HSPICE, and introduces the variation block.

Introduction

As semiconductor technologies migrate to ever smaller geometries and they exhibit larger variations in device characteristics, it becomes more important to simulate (or predict) the effects of these variations on circuit response. Variations in device characteristics are expressed through variations on parameter values of the underlying device models.

- Monte Carlo analysis is typically used to find the variation in circuit response as a result of the parameter variations.
- DC mismatch analysis is an efficient method for simulating the effects of local variations on the DC response.

To get satisfactory answers from these analyses, the target technology must have been characterized for variability and the characterization data properly converted to variation definitions on device model parameters.

How To Define Variability in HSPICE

Three approaches are available to define variability in HSPICE:

- Defining variations on parameters; for example,

  ```
  .param var=agauss(20,1.2,3)
  ```

  For a discussion of this topic, see see Appendix A, Statistical Analysis.

- Defining variations on models using `lot` and `dev` parameters in the model file; for example,

  ```
  vth0=0.6 lot/0.1 dev/0.02
  ```

  For a discussion of this topic, see Appendix A, Statistical Analysis.

- Defining a variation block; for example,

  ```
  .variation
     global and local variation definitions
  .end_variation
  ```

  For additional information, see Chapter 14, Variation Block.

Variation Blocks Replace Previous Approaches

The variation block approach is expected to replace the approaches of defining variations on parameters and models, because it best fulfills the requirements for simulating Nanometer technology devices.

The advantages of the variation block over previous solutions are:

- The variation block consolidate variation definitions in single records
- A clear distinction exists between global and local variations
- Only global or only local variations can be selected
- The syntax allows for defining a local variation as a function of device geometry
- A new type of Monte Carlo output file allows for data mining
- Monte Carlo and DC mismatch analyses give consistent results.

14

Variation Block

Describes the use model and structure of the variation block.

Overview

The characteristics of circuits produced in semiconductor processing are subject to variability, as is the case for any other manufactured product. For a given target technology, the nominal device characteristics are described with a set of parameters, which applies to a certain device model (for example, BSIM3). In HSPICE, the variability of the model parameters is described in a so-called “variation block”. A variation block is a container for specifying variations introduced by the effects in manufacturing on geometry and model parameters.

For the purpose of dealing with variations in HSPICE, they are separated into global variations, defined as variations from lot to lot, wafer to wafer and chip to chip, and into local variations, which are defined for devices in proximity on the same integrated circuit. Both classes can be described in the variation block in a very flexible way by user-defined expressions. Since there are currently no industry-wide standards for specifying process variability, this feature allows each company to implement their own proprietary model for variability. The variation block is generally provided by a modeling group, very similar to device models (for example, BSIM), because it must be created specifically for each technology from test circuits.

The structure of the variation block allows for building expressions to model interdependence and hierarchy of the variations. For example, one random variable can control the variation in oxide thickness of both PMOS and NMOS devices, as it is generally the same for both types of devices.

Note that the earlier methods of specifying variation are not compatible with the variation block. For controlling the behavior of variation blocks, see section on options for controlling the behavior. The variation block is currently used for

Monte Carlo and DCmatch analyses; for a description of these analyses, see Chapter 15, Monte Carlo Analysis and Chapter 16, DC Mismatch Analysis, respectively.

For the functions available to build expressions as presented in the next sections, see Built-In Functions and Variables on page 229.

Variation Block Structure

The structure of a variation block is:

```
.variation
   Define options
   Define common parameters that apply to all sub-blocks
   .global_variation
      Define the univariate independent random variables
      Define additional random variables through
   transformation
      Define variations of model parameters
   .end_global_variation
   .local_variation
      Define the univariate independent random variables
      Define additional random variables through
   transformation
      Define variations of model parameters
      .element_variation
         Define variations of element parameters
      .end_element_variation
   .end_local_variation
.end_variation
```

This structure contains three parts:

- general section
- sub-block for global variations
- sub-block for local variations

General Section

In the general section, options can be defined that control how the information in the variation block is used. Also, parameters can be defined that apply to

both sub-blocks; however, these cannot contain any distribution related functions.

Control Options

At the beginning of the variation block, options can be specified, one per line.

- `Ignore_variation_block=yes`

  Previous methods of specifying variations on parameters and models are not compatible with the variation block. By default, the contents of the variation block are used and any other specification is ignored, thus no changes are required in existing netlists other than adding the variation block. If it is still desirable to run the previous style variations, then the option `ignore_variation_block` can be used.

- `Ignore_local_variations=yes` and `Ignore_global_variations=yes`

  For investigating the effects of global or local variations only.

- Monte Carlo-specific options (see Chapter 15, Monte Carlo Analysis).

Some of these options are useful if a variation block is part of a model file that you cannot edit. One option can be specified per line. For example,

```
option Ignore_local_variations=yes
```

Global and Local Variations Sub-Blocks

Within the global or local variations sub-blocks, univariate independent random variables can be defined. These are random variables with specific distributions over a certain sample space. Additional random variables can be generated through transformations. These random variables form the basis for correlations and complex distributions.

In both sub-blocks, variations on model parameters can be defined. This is where global or local variations on the parameters of semiconductor devices are specified.

A special section within the sub-block for local variations allows for defining local variations on elements. This is either for specifying local temperature variations or variations on generic elements that do not have a model, as used early in the pre-layout design phase; for example, resistors and capacitors.

Independent Random Variables

When describing variations, a basic normal (Gaussian) distribution is assumed, unless otherwise specified explicitly. This default behavior is explained in later sections. Other types of distributions or correlations must be modeled by using independent random variables. These come from three basic distributions:

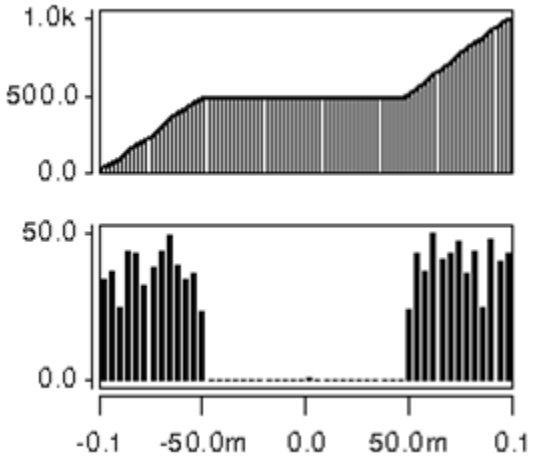
- Uniform distribution: defined over the range from -0.5 to 0.5: `U()`
- Normal distribution: with mean=0 and variance=1, default range +/-4: `N()`
- User-defined cumulative distribution function: CDF(xyPairs)

  If f(x) is the probability density of a random variable x, then the cumulative distribution function is the integral of f(x). A cumulative distribution function can be approximated by a piecewise linear function, which can be described as a sequence of pairs of points [xi, yi]. The following rules apply:

  a. at least two pairs are required
  b. white space or a comma is required between each number
  c. the CDF starts at zero: y1=0
  d. CDF ends at one: yn=1
  e. xi values must be monotonically increasing xi+1 > xi
  f. yi values must be monotonically non-decreasing yi+1 >= yi
  g. the CDF must have zero mean: x1 = - xn

Example

```
parameter var=CDF(-0.1 0 -0.05 0.5 0.05 0.5 0.1 1.0)
```



The distributions N() and U() do not accept any arguments.

The syntax for defining independent random variables is:

```
parameter  a=U()  b=N()   c=CDF(x1,y1,.....xn,yn)
```

These distributions cannot be referenced within expressions; variables must be assigned and the variables can be used within expressions.

Dependent Random Variables

To model distributions which are more complex than the ones which are available through the predefined independent random variables, transformations can be applied by using expressions on independent random variables. A dependent variable can also be created as a function of more than one independent random variable to express correlation.

Example 1

This example creates a random variable with normal distribution, with mean A and standard deviation B.

```
parameter var=N()  Y='A + B * var '
```

Example 2

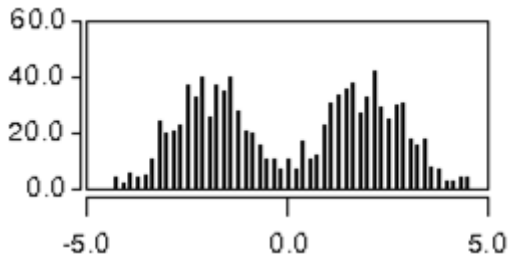
This example creates a random variable with a uniform distribution from D to E, where D and E are arbitrary constants.

```
parameter var=U()  Y='0.5*(D+E) + (E-D) * var '
```

Example 3

This example creates a random variable with two peaks.

```
parameter a=N() b=N() c='a+2*sgn(b)'
```



Variations of Model Parameters

Variations on model parameters can be defined in both global and local sub-blocks. In the course of the simulation, these variations are then applied to the specified device model parameters.

In the simplest case, a variation with normal distribution is described with the following syntax:

```
model_type model_name model_parameter='Expression for
   Sigma'
```

If the expression references only constants and parameters that evaluate to constants, then a Gaussian variation with zero mean and a sigma equal to the expression is automatically implied. A shorthand notation is used; for example,

```
parameter='expression'
```

The meaning of this is actually:

```
variation_in_parameter='expression'
```

For example, the following defines a normal distribution with sigma of 10 on the parameter `rsh` of the resistor with model Rpoly.

```
.global_variation
   R Rpoly rsh=10
.end_global_variation
```

In a more complex case, the variation is modeled as a dependency on one or several previously defined random variables by using the following syntax:

```
model_type model_name model_parameter=
   Perturb('Expression')
```

For example, the variable Toxvar in the following is used to model global variations on oxide thickness and by definition has a normal distribution with mean=0 and sigma=1. The pmos and the nmos models receive the same random variable and apply the variation to the model parameter `tox` in different amounts; the oxide thickness for the two models is correlated.

```
.global_variation
   Parameter Toxvar =N()
   Nmos nch tox=Perturb('7e-10*Toxvar')
   Pmos pch tox=Perturb('8e-10*Toxvar')
.end_global_variation
```

These model types are currently supported: NMOS, PMOS, R, and C.

Variations can only defined on parameters that are explicitly specified in the device model, and are included in the following list:

Model	Parameters							
BSIM3 (level 49)	lint	wint	vth0	Vfb	tox	u0	nsub	
BSIM4 (level 54)	lint	wint	vth0	vfb	toxm	toxe	u0	nsub
R	dlr	dw	rsh					
C	cox	del	capsw	thick				

For binned models, variations can be defined separately by specifying the model name with the bin extension; for example, devices from bins 1 and 2 receive different variation on the parameter `lint`, which models length variation:

```
nmos snps20N.1 lint=10n
nmos snps20N.2 lint=12n
```

Variations of Element Parameters

Devices are not only affected by variations in the underlying model parameters, but also through variations of properties specified at instantiation of an element, or variations on implied properties, such as local temperature. Also, early in the design phase, passive devices sometimes have only a nominal value, but no model yet, because no decision has been made on the particular implementation. For these elements, variations can be specified on the implicit value parameter; for example: R1 1 0 **1k** .

In the simplest case, when there are no dependencies or correlations, a variation with normal distribution is described with the following syntax:

```
element_type element_parameter='Expression for Sigma'
```

If the expression references only constants and parameters that evaluate to constants, then a Gaussian variation with zero mean and a sigma equal to the expression is automatically implied. Note that a shorthand notation is used:

```
parameter='expression'
```

The meaning of this is actually:

```
variation_in_parameter='expression'
```

For example, the following defines a normal distribution with sigma of 10 on the resistors without model:

```
.element_variation
   R R=10
.end_element_variation
```

The currently supported element types and their parameters are:

Element	Parameter
M	DTEMP[2]
R	Rval*[1] DTEMP[2]
C	Cval*[1] DTEMP[2]
Q	AREA DTEMP[2]
D	DTEMP[2]
L	Lval*[1] DTEMP[2]
I	DCval
V	DCval

1. Asterisk "" denotes implicit value parameter.*

2. The DTEMP parameter is implicit; it does not have to be specified on the element instantiation line.

Because different classes of devices might be affected differently, a selection mechanism based on element name and model name is provided by using a condition clause:

```
element_type(condition_clause) element_parameter=
   'Expression for Sigma'
```

The condition clause allows for specifying variations on selected elements, according to their name or associated model. Wildcard substitutions can be indicated as “?” for single character and “*” for multiple characters. Examples for condition clause syntax are:

```
element_type(model_name~='modelNameA')
element_type(element_name~='elNameB')
element_type(model_name~='modelNameC' OPERATOR element_name~='elNameD') par='exp'
element_type(model_name~='modelNameE' OPERATOR model_name~='modelNameF') par='exp'
element_type(element_name~='elNameG' OPERATOR element_name~='elNameH')  par='exp'
```

Where `OPERATOR` can be `&& (AND)`, `|| (OR)`. The operator "~=" stands for "matches".

All pattern matching operations are case-insensitive. A leading subcircuit prefix is ignored when matching the element name.

Example

In this example, only resistor ra1 varies.

```
ra1 1 0 1k
rb1 2 0 1k
.variation
   .local_variation
   .element_variation
   R(element_name~='ra*') R=20
   .end_element_variation
   .end_local_variation
.end_variation
```

In a more complex case, the variation is modeled as a dependency on one or several previously defined random variables by using the following syntax:

```
element_type(condition_clause) element_parameter=
   Perturb('Expression')
```

Example

In this example, only resistor ra2 is affected by the temperature variation specified with a uniform distribution from 0 to 10 degrees (the resistor is located next to a power device).

```
ra1 1 0 1k
rb1 2 0 1k
ra2 3 0 rpoly l=10u w=1u
rb2 4 0 rpoly l=10u w=1u
.model rpoly r rsh=100 tc1=0.01
.variation
   .local_variation
   .element_variation
      parameter tempvar=U()
      R(element_name~='ra*' && model_name~='rpoly')
      +   dtemp=perturb('10*tempvar+5')
   .end_element_variation
   .end_local_variation
.end_variation
```

Absolute Versus Relative Variation

By default, the specified variation is absolute, which means additive to the original model or element parameter; however, sometimes it is more appropriate to specify relative variations that are defined by appending a space and a “%” sign to the expression. The simulator divides the result of the expression by 100, and multiplies by the original parameter value and the random number from the appropriate generator to calculate the change.

Example

In this example, the variation on the threshold parameter `vth0` is specified as absolute (sigma of 80 or 70mV), the variation on the mobility `u0` as relative (15 or 13 percent).

```
global_variation
   nmos snps20N vth0=0.08 u0=15 %
   pmos snps20P vth0=0.07 u0=13 %
.end_global_variation
```

Access Functions

Certain local variations depend on element geometry, as defined with parameters at instantiation. The access function `get_E` allows for using these parameters in expressions by using the following syntax:

```
get_E(element_parameter)
```

Where `element_parameter` is the name of an element parameter, which must be defined on the instantiation line (except for the `DTEMP` parameter). This access function is only supported for local variations, because it does not make sense to define global variations as a function of an element value in the context of semiconductor technology.

For example, the variation on the threshold is specified as inversely proportional to the square root of the total area of the device, as calculated from the product of the element parameters `W`, `L`, and `M`.

```
nmos nch vth0='1.234e-9/sqrt(get_E(W)*get_E(L)*get_E(M))'
```

Another function allows for accessing the values of global parameters by using the following syntax:

```
get_P(global_parameter)
```

For example, the resistor variation is proportional to absolute temperature.

```
.temp 100
ra1 1 0 1k
.variation
   .local_variation
   .element_variation
      R R='(273+get_P(temper))*0.25'
   .end_element_variation
   .end_local_variation
.end_variation
```

Variation Block Example

This variation block is used in the example netlists opampdcm.sp and opampmc.sp.

Most pieces of this example are explained below:

Global variations on vth0 (absolute)
Global variations on u0 (relative)
Local variations on vth0 (absolute), as a function of device area
Local variations on u0 (relative), as a function of device area
Local variation on the implicit value of resistors (relative)

```
.variation
  .global_variation
     NMOS SNPS20N vth0=0.07 u0=10 %
     PMOS SNPS20P vth0=0.08 u0=8 %
  .end_global_variation
  .local_variation
     nmos snps20N vth0='1.234e-9/sqrt(get_E(W)*get_E(L)*get_E(M))'
     + u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
     pmos snps20P vth0='1.234e-9/sqrt(get_E(W)*get_E(L)*get_E(M))'
     + u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
  .element_variation
      R r=10 %
  .end_element_variation
  .end_local_variation
.end_variation
```

15

Monte Carlo Analysis

Describes Monte Carlo analysis in HSPICE.

Overview

Monte Carlo analysis is the generic tool for simulating the effects of variations in device characteristics on circuit performance. The variations in device characteristics are expressed as distributions on the underlying model parameters. For each sample of the Monte Carlo analysis, random values are assigned to these parameters and a complete simulation is executed, producing one or more measurement results. The series of results from a particular measurement represent a distribution, which can be characterized by statistical terms; for example, mean value and standard deviation (σ). With increasing number of samples, the shape of the distribution gets better defined with the effect that the two quantities converge to their final values.

A standard way of analyzing the results is by arranging them in bins. Each bin represents how many results fall into a certain range (slice) of the overall distribution. A plot of these bins is a histogram, which shows the shape of the distribution as the number of results versus slice. As the number of samples increases, the shape of the histogram gets smoother.

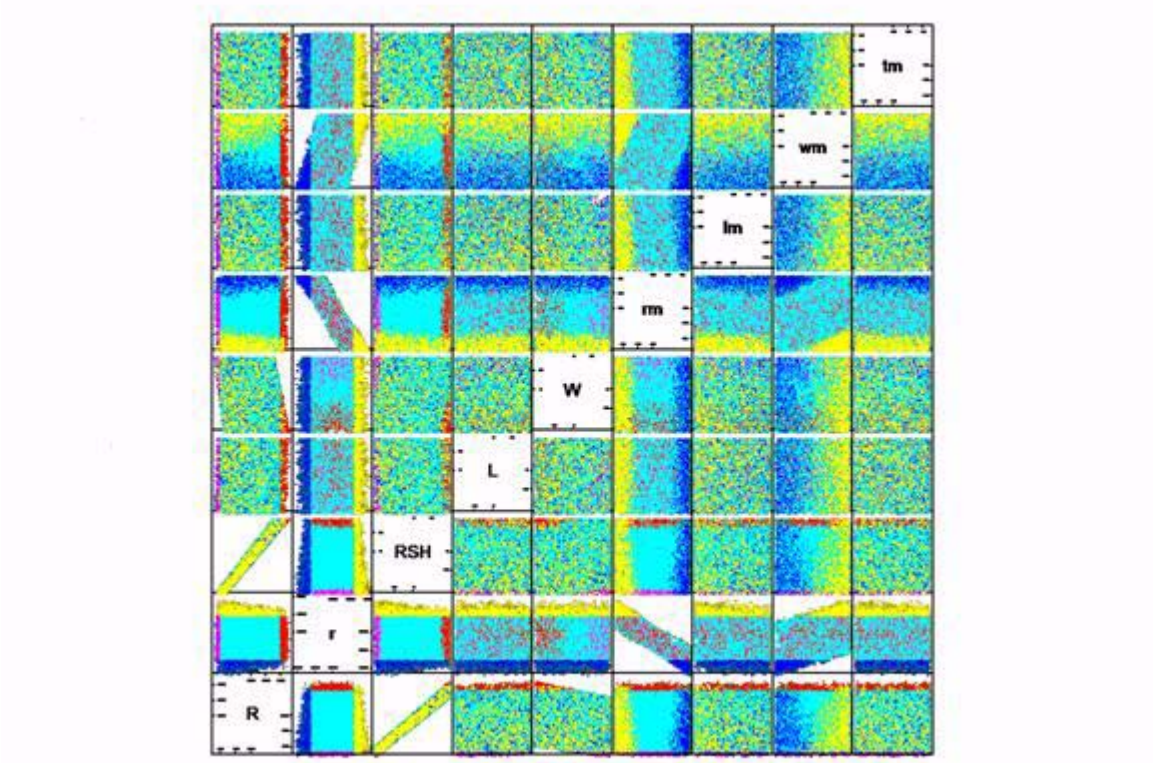
The ultimate interest of Monte Carlo simulation is to find out how the distribution in circuit response relates to the specification. The aspect of yield is considered here:

- What is the percentage of devices which meet the specification?
- Is the design centered with respect to the specification?

Closely related is the aspect of over-design. This is when the circuit characteristics are within specification with a wide margin, which could be at the expense of area or power and ultimately cost.

A typical design process is iterative, first for finding a solution which meets the nominal specification, and then moving on to a solution that meets yield and economic constraints, including the effects of variations in device characteristics. In this optimization process, it helps to understand the relationship of the design parameters to the circuit response, and the relationships of the different types of circuit response. This information is available after running Monte Carlo analysis and can best be presented by Pairs Plots. This is a matrix of two-dimensional plots for investigating pair-wise relationships and exploring the data interactively. HSPICE does not produce such plots, but makes the necessary data available from Monte Carlo simulation. Figure 63 shows an example of a Pairs Plot from a simple resistive divider:

Figure 63 Pairs Plot example



Monte Carlo analysis is computationally expensive; therefore, other types of analysis have been created that produce certain results more efficiently. For cases where only the effects of local variations on the DC response of a circuit is of interest, a method called DC mismatch (DCmatch) analysis can be used; for a description of DCmatch, see Chapter 16, DC Mismatch Analysis.

Monte Carlo Analysis in HSPICE

Monte Carlo analysis has been available in HSPICE for some time and is based on two approaches:

- defining distributions on global parameters (using `AGAUSS`, `GAUSS`, `UNIF`, and `AUNIF`) in a netlist; for example,

  ```
  .param var=agauss(20,1.2,3)
  ```

- defining distributions on model parameters using `DEV` and `LOT` constructs in a model file; for example,

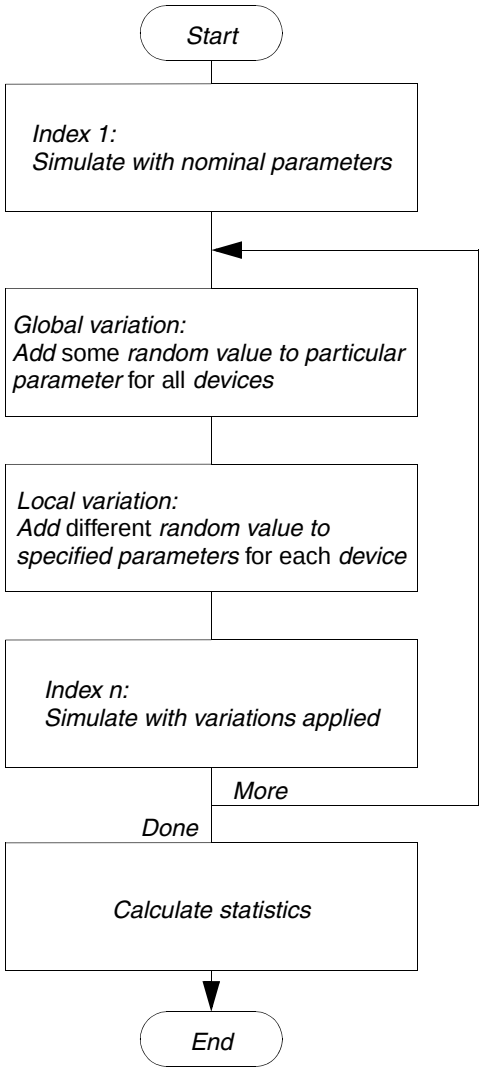
  ```
  vth0=0.6 lot/0.1 dev/0.02
  ```

The above two methods are documented in Appendix A, Statistical Analysis.

To satisfy some key requirements for modern semiconductor technologies, a new approach is available based on the variation block, which is described in detail in chapter 14. This new approach is not compatible with the earlier ones; see the section on options for ways to select one or the other method.

For the following discussion, refer to diagram in Figure 64. Sample number 1 of a Monte Carlo analysis is always executed with nominal values and no variation. For subsequent samples, HSPICE updates the parameters specified for variation in the variation block with random values. For global variations, a specified parameter is changed by the same random value for all elements that share a common model. For local variation, the specified parameter is changed by a different random value for each element. The changes due to global and local variations are additive and are saved in a file for post-processing. When all the elements have been updated, the simulation is executed and the measurement results are saved. When all the requested samples have been simulated, HSPICE calculates the statistics of the measurement results and includes them in the run listing.

Figure 64 Monte Carlo analysis flow in HSPICE



Input Syntax

Monte Carlo analysis is always executed in conjunction with some other analysis:

```
.DC MCcommand
.DC sweepVar start stop step sweep MCcommand
.AC type step start stop sweep MCcommand
.TRAN step stop MCcommand
```

Syntax for *MCcommand*:

```
MONTE =
+ <val |
+ list num |
+ val firstrun=num |
+ list(<num1:num2><num3><num4:num5>)>
```

Parameter	Description
val	Specifies the number of random samples to produce.
val firstrun=num	Specifies the sample number on which the simulation starts.
list num	Specifies the sample number to execute.
list(<num1: num2> <num3> <num4: num5>)	Samples from num1 to num2, sample num3, and samples from num4 to num5 are executed (parentheses are optional).

The parameter values and results are always the same for a particular sample, whether generated in one pass or using `firstrun` or the `list` syntax. Therefore, Monte Carlo analyses can be split or distributed and the results spliced together.

Examples

In these examples a DC sweep is applied to a parameter k. In the first case, 10 samples are produced. In the second case, five samples are produced, starting with sample number 6. In the last two examples, samples 5, 6, 7 and 10 are simulated.

```
.dc k start=2 stop=4 step=0.5 monte=10
.dc k start=2 stop=4 step=0.5 monte=5 firstrun=6
.dc k start=2 stop=4 step=0.5 monte=list 5:7 10
.dc k start=2 stop=4 step=0.5 monte=list(5:7 10)
```

Note that options for the previous Monte Carlo style are ignored when simulations based on the variation block are executed.

Simulation Output

The output listing file contains a summary of the names of the models and model parameters, as well as the elements and element parameters, that are subject to global or local variations. For each sample, the measured results are printed. Towards the end, the statistics for the measured data are shown.

Partial printout of an output listing:

```
MONTE CARLO DEFINITIONS
Global variations:    model              parameter
                      snps20n            vth0
                      snps20n            u0

Local variations:     model              parameter
                      snps20n            vth0
                      snps20n            u0

Element variations:   element             parameter
                       r1                 r

*** monte carlo  index =     1 ***
      systoffset=  1.0857E-03
*** monte carlo  index =     2 ***
      systoffset= -2.6826E-04
          .
          .
MONTE CARLO STATISTICS
  meas_variable = systoffset
  mean  =   1.0124m      varian = 504.8187n
  sigma = 710.5059u      avgdev = 523.4913u
  max   =   2.2942m      min    =-268.2590u
```

Measure commands cause simulation results to be saved for each sample, along with its index number. Depending on the analysis type, the name of the result file has an extension of ms#, ma#, or mt#, where the # denotes the regular sequence number for HSPICE output files. In addition, the changes in all parameter values subject to variation are saved in a file with an extension of mcs#, mca#, or mct#, depending on the analysis type.

The structure of this file is the same as for regular measure files. In the header section, the names of the parameters are presented as follows:

- for global variation on model parameter:

  ```
  Model_name@parameter_name(ID)
  ```

- for local variation on model parameter:

  ```
  Element_name@model_name@parameter_name(ID)
  ```

- for local variation on element parameter:

  ```
  Element_name@parameter_name(ID)
  ```

Where ID is a string for identifying the type of the parameter as follows:

First character		Second character		Third character	
M	Model	G	Global	R	Relative
E	Element	L	Local	A	Absolute

Results for parameters that have absolute variation specified in the variation block are reported as absolute deviation from the nominal value. Results for parameters that have relative variation specified are reported as a relative deviation in percent. The printed value for parameter “status” is “1” for a successful simulation, and “0” for a failed simulation.

For example, a mcs# file:

```
index        snps20n@vth0@MGA   snps20n@u0@MGR
             xi82.mn1@snps20n@vth0@MLA  xi82.mn1@snps20n@u0@MLR
             xi82.mn6@snps20n@vth0@MLA  xi82.mn6@snps20n@u0@MLR
             xi82.r0@r@ELR  status  alter#
 1.0000      0.               0.             0.
             0.               0.             0.
             0.               1.0            1.0000
 2.0000      4.299e-02         6.285e-02     2.113e-04
             2.354e-04        -4.926e-05     4.965e-04
             0.2185            1.0           1.0000
    .
    .
```

In this example, the changes due to the global variations on parameters vth0 (absolute) and u0 (relative) are reported first, then the changes on each device due to local variations on the same parameters are reported next, and finally, the local variation on the parameter r of the element r0 are reported. Note that

the parameter value applied to the device for a particular sample is the nominal value, plus the reported change due to global variations, plus the reported change due to local variations.

The contents of this parameter file are useful for data mining. In connection with the measured data in the regular output file, the relationship of circuit response variation to parameter variation can be investigated by using, for example, a Pairs plot as shown in Figure 63 on page 446.

If a simulation fails without creating a result, HSPICE currently substitutes the results of the previous sample, which can be misleading when analyzing the relationship between results and underlying parameters. The information in the “status” column of the result file can be used to skip the incorrect data.

Application Considerations

If too large variations are applied, caused by the combinations of variation specified in the variation block and the value of `Normal_limit`, some circuits show abnormal behavior under these conditions and the simulation result can be completely off or missing. This can distort the result statistics reported by HSPICE at the end of the Monte Carlo simulation. Therefore, you should to review the individual measurement results for outliers and analyze them properly. For example, when plotting measurement results as a function of index in CosmosScope, the outliers are readily apparent.

16

DC Mismatch Analysis

Describes the use of DCmatch analysis.

Mismatch

Variations in materials and procedures are the source of differences in characteristics of identically designed devices on the same integrated circuit. These are random time-independent variations by nature and are collectively called *mismatch*.

Mismatch is one of the limiting factors in analog signal processing. It affects more and more circuit types as device dimensions and signal swings are reduced. Mismatch is a function of the geometry of the devices involved, their spatial relationship (distance and orientation) and their environment. Mismatch does not include the effects of global variations, such as batch-to-batch or wafer-to-wafer variations, nor the effects or device degradation. For cases where the effects of local variation on DC response of a circuit are of interest, a method called DC mismatch (DCmatch) analysis can be used.

DCmatch analysis is related to sensitivity and noise analyses, and requires significantly less runtime than Monte Carlo analysis. Thus, DCmatch analysis provides an efficient technique for the approximate computation of the effects of variability on circuit DC solutions.

DCmatch Analysis

In DCmatch analysis, the combined effects of variations of all devices on a specified node voltage or branch current are determined. The primary purpose is to consider the effects of local variations (that is, for devices in close proximity). DCmatch analysis also allows for identifying groups of matched devices (that is, devices that should be implemented on the layout according to

special rules). A secondary result is calculated as the influence of global variations, which is useful for investigating whether their effects on circuit response are much smaller then the effects of local variations, when optimizing a design.

DCmatch analysis is based on the following dependencies and assumptions:

- variations in device characteristics are modeled through variations in the underlying model parameters.
- statistics of the model parameters exhibit Normal distributions.
- no correlation exists between the variations of different parameters of a single device, or between the same parameter for different devices.
- effects on a circuit's DC solution are small; therefore, these variations also exhibit Normal distributions.

In HSPICE, the variations in model parameters are defined in the variation block (see Chapter 14, Variation Block). Those definitions are used to calculate the variation in DC response. DCmatch analysis runs either from a default operating point or for each value of the independent variable in a single DC sweep. The default output is in the form of tables containing the sorted contributions of the relevant devices to the total variation, as well as information on matched devices. In the current implementation, a heuristic algorithm makes a best guess effort to identify matched devices. This means that the results are suggestions only. In addition to the table, the total variation and contributions of selected devices can be output using .PROBE and .MEASURE commands.

Input Syntax

```
.DCMATCH OUTVAR <THRESHOLD=T> <FILE=string>
+ <PERTURBATION=P> <INTERVAL=Int>
```

Parameter	Description
OUTVAR	Valid node voltages, the difference between node pairs, or branch currents.

Parameter	Description
THRESHOLD	Report devices with a relative variance contribution above Threshold in the summary table. ▪ T=0: reports results for all devices ▪ T<0: suppresses table output; however, individual results are still available through .PROBE or .MEASURE statements. The upper limit for `T` is 1, but at least 10 devices are reported, or all if there are less than 10. Default value is 0.01.
FILE	Valid file name for the output tables. Default is `basename.dm#` where "`#`" is the usual sequence number for HSPICE output files.
PERTURBATION	Indicates that perturbations of `P` standard deviation will be used in calculating the finite difference approximations to device derivatives. The valid range for `P` is 0.01 to 6, with a default value of 2.
INTERVAL	Applies only if a DC sweep is specified. *Int* is a positive integer. A summary is printed at the first sweep point, then for each subsequent increment of *Int*, and then, if not already printed, at the final sweep point. Only single sweeps are supported.

Note:

If more than one DCmatch analysis is specified per simulation, only the last statement is used.

Example 1

In this example, HSPICE reports DCmatch variations on the voltage of node 9, the voltage difference between nodes 4 and 2, and on the current through the source VCC.

```
.DCmatch V(9) V(4,2) I(VCC)
```

Example 2:

In this example, the variable `XVal` is being swept in the DC command, from 1k to 9k in increments of 1k. DCmatch variations are calculated for the voltage on node `out`. Tables with DCmatch results are generated for the set XVal={1K, 4K, 7K, 9K}.

```
.DC XVal Start=1K Stop=9K Step=1K
.DCmatch V(out) Interval=3
```

DCmatch Table Output

For each output variable and sweep point, HSPICE generates a result record, that includes setup information, total variations, and a table with the sorted contributions of the relevant devices. The individual entries are:

- sweep or operating points for which the table is generated
- name of the output variable
- DC value of this output variable
- values used for DCmatch options
- output sigma due to combined global and local variations

  $$\sqrt{\sigma_{\text{global}^2} + \sigma_{\text{local}^2}}$$

- results for global variations
  - number of devices that had no global variability specified
  - output sigma due to global variations
  - table with parameter contributions
  - contribution sigma (volts or amperes)
  - contribution variance for ith parameter (in percent)

    $$\frac{sigma(i)^2}{\sum_1^n sigma(k)^2} \times 100$$

  - cumulative variance through ith parameter (in percent)

    $$\frac{\sum_1^i sigma(k)^2}{\sum_1^n sigma(k)^2} \times 100$$

- results for local variations
  - number of devices that had no local variability specified
  - output sigma due to local variations
  - number of devices that had local variance contributions below the threshold value and were not included in the table

- table with sorted device contributions

  Contribution sigma (in volts or amperes). Values below 100nV or 1PA are rounded to zero to avoid reporting numerical noise.

- contribution variance for the ith device (in percent)

$$\frac{sigma(i)^2}{\sum_1^n sigma(k)^2} \times 100$$

  The parameter “Threshold” applies to this column.

- cumulative variance through ith device (in percent)

$$\frac{\sum_1^i sigma(k)^2}{\sum_1^n sigma(k)^2} \times 100$$

The table also includes a suggestion on matched devices that should be verified independently. Devices with the same number in the column “Matched pair” are likely to be matched. Their layout should be reviewed for conformity to established matching rules.

Example

```
sweep point = operating point
============================================================
output = v(out) node voltage =      1.25V  threshold = 1.000E-2
 perturbation =  2.00     interval = 1
 Output sigma due to global and local variations = 619.62uV
 DCMATCH GLOBAL VARIATION
  10 Devices had no Global Variability specified.
 Output sigma due to global variations = 289.66uV
 ------------------------------------------------------------
 Contribution    Contribution    Cumulative      Independent
 Sigma(V)        Variance (%)    Variance (%)    Variable
 227.94u          61.92           61.92           snps20p@u0
 139.48u          23.19           85.11           snps20p@vth0
 109.93u          14.40           99.51           snps20n@u0
  20.19u          485.62m         100.00          snps20n@vth0
 DCMATCH LOCAL VARIATION
   10 Devices had no Local Variability specified
 Output sigma due to local variations =   547.74uV
    4 Devices with Contribution Variance larger than Threshold
------------------------------------------------------------
Contribution  Contribution  Cumulative   Matched   Device
Sigma(V)      Variance (%)  Variance (%)  pair     Name
```

```
295.88u      29.18          29.18        1        xi82.mn1
295.65u      29.13          58.31        1        xi82.mn2
252.09u      21.18          79.50        2        xi82.mp4
247.94u      20.49          99.98        2        xi82.mp3
  6.49u      14.03m        100.00        0        xi82.mp5
  1.72u     984.38u        100.00        0        xi82.mn7
658.15n     144.37u        100.00        0        xi82.mn6
  0.          0.           100.00        0        xi82.mn8
```

Output From .PROBE and .MEASURE Commands

The different results produced by DCmatch analysis can be saved by using `.PROBE` and `.MEASURE` commands, for the output variable specified on the `.DCMATCH` command. If multiple output variables are specified, a result is produced for the last one only. A DC sweep needs to be specified to produce these kinds of outputs; a single point is enough.

The keywords available for saving specific results from DCmatch analysis are:

- `DCm_total`

  Output sigma due to global and local variations.

- `DCm_global`

  Output sigma due to global variations.

- `DCm_global(par)`

  Contribution of parameter “par” to output sigma due to global variations.

- `DCm_local`

  Output sigma due to local variations.

- `DCm_local(dev)`

  Contribution of device “dev” to output sigma due to local variations.

Syntax for .PROBE Command

A `.PROBE` statement in conjunction with `.OPTION POST` creates a data file with waveforms that can be displayed in CosmosScope.

```
.PROBE DC DCm_total
.PROBE DC DCm_global
.PROBE DC DCm_local
.PROBE DC DCm_global(ModelType,ModelName,ParameterName)
.PROBE DC DCm_local(InstanceName)
```

This type of output is useful for plotting the effects of mismatch as a function of bias current, temperature, or a circuit parameter.

Examples

In the first example, the contribution of the variations on vth0 (threshold) of the nmos devices with model SNPS20N is saved. In the second example, the contribution of device mn1 in subcircuit X8 is saved.

```
.probe dcm_global(nmos,SNPS20N,vth0)
.probe dcm_local(X8.mn1)
```

Syntax for .MEASURE Command

With `.MEASURE` statements, HSPICE performs measurements on the simulation results and saves them in a file with a .ms# extension.

```
.MEAS DC res1 max DCm_total
.MEAS DC res2 max DCm_global
.MEAS DC res3 max DCm_local
.MEAS DC res4 max DCm_global(ModelType,ModelName,ParameterName)
.MEAS DC res5 max DCm_local(InstanceName)
.MEAS DC res6 find DCm_local at=SweepValue
.MEAS DC res7 find DCm_local(InstanceName) at=SweepValue
```

Example

In this example, the result systoffset reports the systematic offset of the amplifier; the result matchoffset reports the variation due to mismatch; and the result maxoffset reports the maximum (3-sigma) offset of the amplifier.

```
.MEAS DC systoffset avg V(inp,inn)
.MEAS DC matchoffset avg DCm_local
.MEAS DC maxoffset param='abs(systoffset)+3.0*matchoffset'
```

Practical Considerations

This section discusses practical considerations when using DCmatch analysis.

DCmatch Variability as a Function of Device Geometry

Various parameter relationships for device variability have been used in the industry. Two approaches are shown below with their expressions for HSPICE. The basic construct to calculate mismatch as a function of device size is `get_E` (for details, see Access Functions on page 442).

Example 1

This example assumes a standard transistor size and scales the variation with the number of devices in parallel. This covers the practice of interdigitating matched devices of a characterized standard size:

```
pmos pch vth0 ='dmvp0/sqrt(E(M))'  u0='dmup0/sqrt(E(M))' %
```

Example 2

This example shows an approach that calculates the variation as a function of device size. Two of the three terms implement the well known dependence on the inverse of the square root of the device area:

```
pmos pch vth0 =
+ 'dmpvtwl/sqrt(get_E(W)*get_E(L)*get_E(M)) +
+ dmpvtwll/(get_E(L)*sqrt(get_E(W)*get_E(M)))'
+ u0 ='dmpu0wl/sqrt(get_E(W)*get_E(L)*get_E(M)) %
```

Note that the HSPICE approach with user-defined expressions for sigma allows for much more flexibility than the relationships shown in the above two examples. For example, the variations due to body effect can easily be included.

Parameter Traceability

The parameters and expressions are derived from characterizing dedicated test structures for a given semiconductor technology. To use this information successfully, it must be understood how it relates to the results from the DCmatch analysis.

In the simple example considered in this discussion, variability is modeled as threshold dependence only in the DCmatch definition block, `VTH0 =dmvp0`. It is assumed that this `VTH0` change maps directly to a VGS change. In the characterization of the test structures, the differences in VGS for a transistor pair at a certain current are collected and then, for example, the (one) sigma of a large number of pairs is calculated as 1mV. The value for `dmvp0` has to be defined as 1mV / sqrt (2) = 0.707mV, because two devices are involved. After simulating such a pair under the same conditions as for the characterization, HSPICE reports a contribution of 0.707mV from each device, and a total variation of sqrt ($0.707^2 + 0.707^2$) mV = 1mV between the two devices. This is the same value as the original sigma from the test structure. For this flow to work properly, it is crucial to know that

- transistor pairs were measured
- characterization results represent one sigma

- DCmatch parameter value was adjusted for a single transistor
- simulation results represent one sigma.

Of course, other scenarios are possible—it is just important to connect these pieces properly to achieve correct results.

Example

An example netlist for running DCmatch analysis using a classic 7-transistor CMOS operational amplifier. This example is available in the HSPICE demo directory as $<installdir>/demo/hspice/apps/opampdcm.sp

In this netlist, device sizes are set up as a function of a parameter k, which allows for investigating the effects of the global and local variations as a function of device size. The following lines relate to DCmatch analysis:

```
...
.param k=2
...
mn1 net031 inn net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mn2 net18 inp net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mp3 net031 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
mp4 net18 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
...
.variation
  .global_variation
     nmos snps20N vth0=0.07 u0=10 %
     pmos snps20P vth0=0.08 u0=8 %
  .end_global_variation
.local_variation
   nmos snps20N vth0='1.234e-9/sqrt(get_E(W)*get_E(L)*get_E(M))'
+                u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
   pmos snps20P vth0='1.234e-9/sqrt(get_E(W)*get_E(L)*get_E(M))'
+               u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
.element_variation
   R r=10 %
.end_element_variation
.end_local_variation
.end_variation
...
.dcmatch v(out)
.dc k start=1 stop=4 step=0.5
...
.meas DC systoffset find V(in_pos,in_neg)  at=2
```

```
.meas DC dcmoffset find DCm_local at=2
.meas DC maxoffset param='abs(systoffset)+3.0*dcmoffset'
.meas DC dcm_mn2 find DCm_local(xi82.mn2) at=2
.meas DC gloffset find DCm_global at=2
.option post
...
```

The DCmatch analysis produces four types of output from this netlist:

- table from operating point with k=2 in the output listing
- table from DC sweep for k=1 to 4 in file opampdcm.dm0
- waveform for output variation as a function of `k` in file opampdcm.sw0
- in file opampdcm.sw0 for `k=2`:
  - values for systematic offset
  - output sigma due to local variation
  - 3-sigma amplifier offset
  - contribution of device mn2 to output sigma due to local variation
  - output sigma due to global variation.

References

[1] M.Pelgrom, A.Duinmaijer, and A.Welbers, “Matching Properties of MOS Transistors,” IEEE J. Solid-State Circuits, vol. 24, no. 5, pp. 1433-1439, May 1989

[2] P.R.Kinget, “Device Mismatch and Tradeoffs in the Design of Analog Circuits,” IEEE J. Solid-State Circuits, vol. 40, no. 6, pp. 1212-1224, June 2005

17

Optimization

Describes optimization in HSPICE for optimizing electrical yield.

Overview

Optimization automatically generates model parameters and component values from a set of electrical specifications or measured data. When you define an optimization program and a circuit topology, HSPICE automatically selects the design components and model parameters to meet your DC, AC, and transient electrical specifications.

The circuit-result targets are part of the `.MEASURE` command structure and you use a `.MODEL` statement to set up the optimization.

Note:

> HSPICE uses post-processing output to compute the `.MEASURE` statements. If you set `INTERP=1` to reduce the post-processing output, the measurement results might contain interpolation errors. See the *HSPICE Command Reference* for more information about these options.

HSPICE employs an incremental optimization technique. This technique solves the DC parameters first, then the AC parameters, and finally the transient parameters. A set of optimizer measurement functions not only makes transistor optimization easy, but significantly improves cell and circuit optimization.

To perform optimization, create an input netlist file that specifies:

- Minimum and maximum parameter and component limits.
- Variable parameters and components.
- An initial estimate of the selected parameter and component values.
- Circuit performance goals or a model-versus-data error function.

If you provide the input netlist file, optimization specifications, component limits, and initial guess, then the optimizer reiterates the circuit simulation until it either meets the target electrical specification, or finds an optimized solution.

For improved optimization, reduced simulation time, and increased likelihood of a convergent solution, the initial estimate of component values should produce a circuit whose specifications are near those of the original target. This reduces the number of times the optimizer reselects component values and resimulates the circuit.

Optimization Control

How much time an optimization requires before it completes depends on:

- Number of iterations allowed.
- Relative input tolerance.
- Output tolerance.
- Gradient tolerance.

The default values are satisfactory for most applications. Generally, 10 to 30 iterations are sufficient to obtain accurate optimizations.

Simulation Accuracy

For optimization, set the simulator with tighter convergence options than normal. The following are suggested options:

For DC MOS model optimizations:

```
absmos=1e-8
relmos=1e-5
relv=1e-4
```

For DC JFET, BJT, and diode model optimizations:

```
absi=1e-10
reli=1e-5
relv=1e-4
```

For transient optimizations:

```
relv=1e-4
relvar=1e-2
```

Curve Fit Optimization

Use optimization to curve-fit DC, AC, or transient data:

1. Use the .DATA statement to store the numeric data for curves in the data file as in-line data.
2. Use the .PARAM xxx=OPTxxx statement to specify the variable circuit components and the parameter values for the netlist.

   The optimization analysis statements use the DATA keyword to call the in-line data.
3. Use the .MEASURE statement to compare the simulation result to the values in the data file

   In this statement, use the ERR1 keyword to control the comparison.

If the calculated value is not within the error tolerances specified in the optimization model, HSPICE selects a new set of component values. HSPICE then simulates the circuit again and repeats this process until it obtains the closest fit to the curve or until the set of error tolerances is satisfied.

Goal Optimization

Goal optimization differs from curve-fit optimization, because it usually optimizes only a particular electrical specification, such as rise time or power dissipation.

To specify goal optimizations, do the following:

1. Use the GOAL keyword.
2. In the .MEASURE statement, select a relational operator where GOAL is the target electrical specification to measure.

For example, you can choose a relational operator in multiple-constraint optimizations when the absolute accuracy of some criteria is less important than for others.

Timing Analysis

To analyze circuit timing violation, HSPICE uses a binary search algorithm. This algorithm generate a set of operational parameters, which produce a failure in the required behavior of the circuit. When a circuit timing failure

occurs, you can identify a timing constraint, which can lead to a design guideline. Typical types of timing constraint violations include:

- Data setup time before a clock.
- Data hold time after a clock.
- Minimum pulse width required to allow a signal to propagate to the output.
- Maximum toggle frequency of the component(s).

Bisection Optimization finds the value of an input variable (target value) associated with a goal value for an output variable. To relate them, you can use various types of input and output variables, such as voltage, current, delay time, or gain, and a transfer function.

You can use the bisection feature in either a pass-fail mode or a bisection mode. In each case, the process is largely the same.

Optimization Statements

Optimization requires several statements:

- `.MODEL modname OPT ...`
- `.PARAM parameter=OPTxxx (init, min, max)`

  Use `.PARAM` statements to define initial, lower, and upper bounds.
- A `.DC`, `.AC`, or `.TRAN` analysis statement, with:

  `MODEL=modname`

  `OPTIMIZE=OPTxxx`

  `RESULTS=measurename`

  Use the `.PRINT`, `.PLOT`, and `.GRAPH` output statements, with the `.DC`, `.AC`, or `.TRAN` analysis statements.

Only use an analysis statement with the OPTIMIZE keyword for optimization. To generate output for the optimized circuit, specify another analysis statement (.DC, .AC, or .TRAN), and the output statements.

- .MEASURE measurename ... <GOAL=| < | > val>

  Include a space on either side of the relational operator:

  =

  <

  >

  For a description of the types of .MEASURE statements that you can use in optimization, see Chapter 7, Simulation Output

  The proper specification order is:

  a. Analysis statement with OPTIMIZE.
  b. .MEASURE statements specifying optimization goals or error functions.
  c. Ordinary analysis statement.
  d. Output statements.

Optimizing Analysis (.DC, .TRAN, .AC)

The following syntax optimizes HSPICE simulation for a DC, AC, and Transient analysis.

```
.DC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.AC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.TRAN <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

Argument	Description
DATA	Specifies an in-line file of parameter data to use in optimization.
MODEL	The optimization reference name, which you also specify in the .MODEL optimization statement.

Argument	Description
OPTIMIZE	Indicates that the analysis is for optimization. Specifies the parameter reference name used in the .PARAM optimization statement. In a .PARAM optimization statements, if OPTIMIZE selects the parameter reference name, then the associated parameters vary during an optimization analysis.
RESULTS	The measurement reference name. You also specify this name in the .MEASURE optimization statement. RESULTS passes the analysis data to the .MEASURE optimization statement.

Optimization Examples

This section contains examples of HSPICE optimizations (for HSPICE RF optimization, see “Optimization” in the *HSPICE RF Manual*):

- MOS Level 3 Model DC Optimization
- MOS Level 13 Model DC Optimization
- RC Network Optimization
- Optimizing CMOS Tristate Buffer
- BJT S Parameters Optimization
- BJT Model DC Optimization
- Optimizing GaAsFET Model DC
- Optimizing MOS Op-amp

MOS Level 3 Model DC Optimization

This example shows an optimization of I-V data to a Level 3 MOS model. The data consists of gate curves (ids versus vgs) and drain curves (ids versus vds).

This example optimizes the Level 3 parameters:

- `VTO`
- `GAMMA`
- `UO`
- `VMAX`

- THETA
- KAPPA

After optimization, HSPICE compares the model to the data for the gate, and then to the drain curves. .OPTION POST generates AvanWaves files for comparing the model to the data.

Input Netlist File for Level 3 Model DC Optimization You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/devopt/ml3opt.sp
```

The HSPICE input netlist shows:

- Using .OPTION to tighten tolerances, which increases the accuracy of the simulation. Use this method for I-V optimization.
- .MODEL optmod OPT itropt=30 limits the number of iterations to 30.
- The circuit is one transistor. The VDS, VGS, and VBS parameter names, match names used in the data statements.
- .PARAM statements specify XL, XW, TOX, and RSH process variation parameters, as constants. The device characterizes these measured parameters.
- The model references parameters. In GAMMA= GAMMA, the left side is a Level 3 model parameter name; the right side is a .PARAM parameter name.
- The long .PARAM statement specifies initial, min and max values for the optimized parameters. Optimization initializes UO at 480, and maintains it within the range 400 to 1000.
- The first .DC statement indicates that:
  - Data is in the in-line .DATA all block, which contains merged gate and drain curve data.
  - Parameters that you declared as OPT1 (in this example, all optimized parameters) are optimized.
  - The COMP1 error function matches the name of a .MEASURE statement.
  - The OPTMOD model sets the iteration limit.

- The `.MEASURE` statement specifies least-squares relative error. HSPICE divides the difference between data par(ids) and model i(m1) by the larger of:
  - the absolute value of `par(ids)`, or
  - `minval=10e-6`

    If you use `minval`, low current data does not dominate the error.
- Use the remaining `.DC` and `.PRINT` statements for print-back after optimization. You can place them anywhere in the netlist input file, because parsing the file correctly assigns them.
- The `.PARAM VDS=0 VGS=0 VBS=0 IDS=0` statements declare these data column names as parameters.

  The `.DATA` statements contain data for `IDS` versus `VDS`, `VGS`, and `VBS`. Select data that matches the model parameters to optimize.

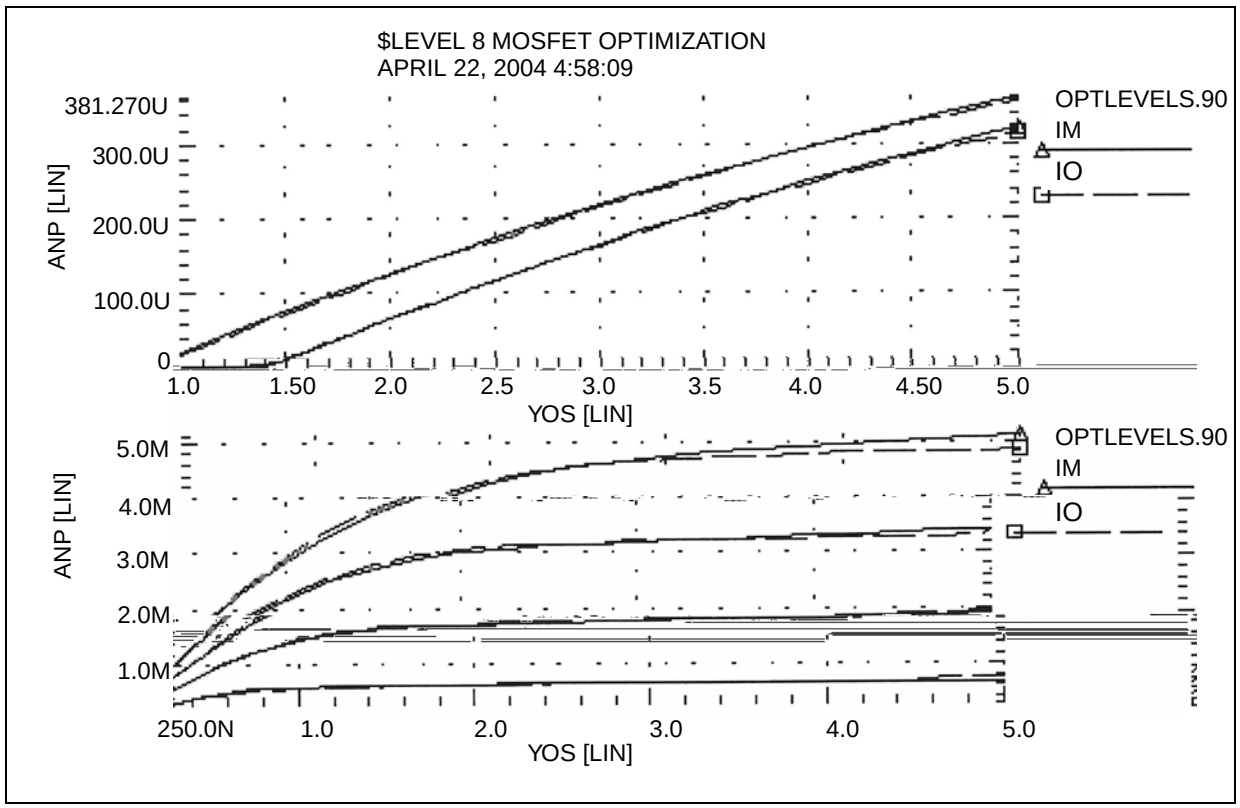
Example

To optimize GAMMA, use data with back bias (VBS= -2 in this case). To optimize KAPPA, the saturation region must contain data. In this example, the all data set contains:

- Gate curves: vds=0.1 vbs=0,-2 vgs=1 to 5 in steps of 0.25.
- Drain curves: vbs=0 vgs=2,3,4,5 vds=0.25 to 5 in steps of 0.25.

Figure 65 shows the results.

Figure 65 Level 3 MOSFET Optimization



MOS Level 13 Model DC Optimization

This example shows I-V data optimization to a Level 13 MOS model. The data consists of gate curves (ids versus vgs) and drain curves (ids versus vds). This example demonstrates two-stage optimization.

1. HSPICE optimizes the `vfb0`, `k1`, `muz`, `x2m`, and `u00` Level 13 parameters to the gate data.
2. HSPICE optimizes the `MUS`, `X3MS`, and `U1` Level 13 parameters, and the `ALPHA` impact ionization parameter to the drain data.

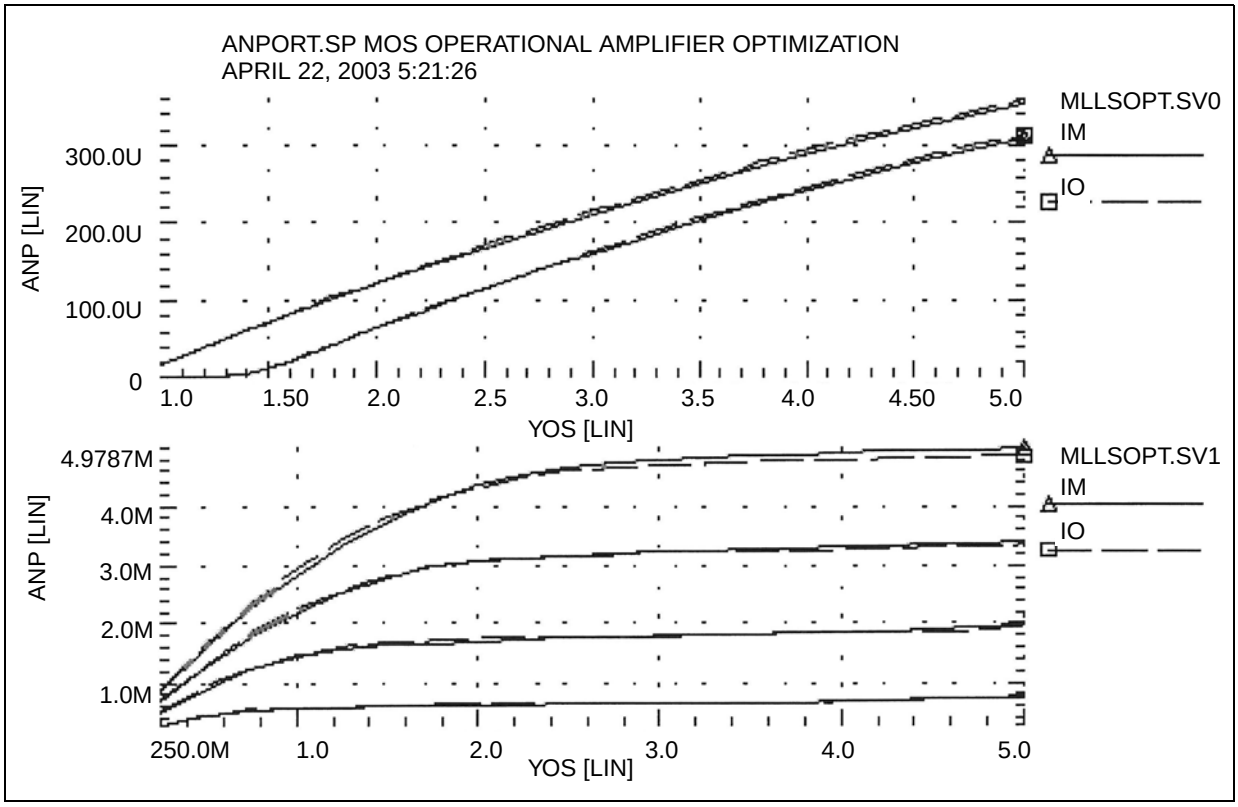
After optimization, HSPICE compares the model to the data. The `POST` option generates AvanWaves files to compare the model to the data. Figure 66 on page 476 shows the results.

DC Optimization Input Netlist File for Level 13 Model This example is based on demonstration netlist ml13opt.sp, which is available in directory $<installdir>/demo/hspice/mos:

```
$LEVEL 13 mosfet optimization
$..tighten the simulator convergence properties
.OPTION nomod post=2 newtol relmos=1e-5 absmos=1e-8
.MODEL optmod OPT itropt=30

*Circuit Input

vds 30 0 vds
vgs 20 0 vgs
vbs 40 0 vbs
m1 30 20 0 40 nch w=50u l=4u
$..
$..process skew parameters for this data
.PARAM xwn=-0.3u xln=-0.1u toxn=196.6 rshn=67
$..the model and initial guess
.MODEL nch NMOS LEVEL=13
+ acm=2 ldif=0 hdif=4u tlev=1 n=2 capop=4 meto=0.08u
+ xqc=0.4
$...parameters obtained from measurements
+ wd=0.15u ld=0.07u js=1.5e-04 jsw=1.8e-09
+ cj=1.7e-04 cjsw=3.8e-10
$...parameters not used for this data
+ k2=0 eta0=0 x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x3u1=0
$...process skew parameters
+ toxm=toxn rsh=rshn
+ xw=xwn xl=xln
$...optimized parameters
+ vfb0=vfb0 k1=k1 x2m=x2m muz=muz u00=u00
+ mus=mus x3ms=x3ms u1=u1
$...impact ionization parameters
+ alpha=alpha vcr=15
.PARAM
+ vfb0 = opt1(-0.5, -2, 1)
+ k1 = opt1(0.6,0.3,1)
+ muz = opt1(600,300,1500)
+ x2m = opt1(0,-10,10)
+ u00 = opt1(0.1,0,0.5)
+ mus = opt2(700,300,1500)
+ x3ms = opt2(5,0,50)
+ u1 = opt2(0.1,0,1)
+ alpha = opt2(1,1e-3,10)
```

```
*Optimization Sweeps

.DC DATA=gate optimize=opt1 results=comp1 model=optmod
.MEAS DC comp1 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05
.DC DATA=drain optimize=opt2 results=comp2 model=optmod
.MEAS DC comp2 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05

*DC Data Sweeps

.DC DATA=gate
.DC DATA=drain

*Print Sweeps

.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)

*DC Sweep Data

$..data
.PARAM vds=0 vgs=0 vbs=0 ids=0
.DATA gate vds vgs vbs ids
1.000000e-01 1.000000e+00 0.000000e+00 1.655500e-05
1.000000e-01 5.000000e+00 -2.000000e+00 3.149500e-04
.ENDDATA
.DATA drain vds vgs vbs ids
2.500000e-01 2.000000e+00 0.000000e+00 2.809000e-04
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDDATA
.END
```

Figure 66 Level 13 MOSFET Optimization



RC Network Optimization

The following example optimizes the power dissipation and time constant for an RC network. The circuit is a parallel resistor and capacitor. Design targets are:

- 1 s time constant.
- 50 mW rms power dissipation through the resistor.

The HSPICE strategy is:

- RC1 `.MEASURE` calculates the RC time constant, where the GOAL of .3679 V corresponds to 1 s time constant e-rc.
- RC2 `.MEASURE` calculates the rms power, where the GOAL is 50 mW.
- OPT_{rc} identifies `RX` and `CX` as optimization parameters, and sets their starting, minimum, and maximum values.

Network optimization uses these HSPICE features:

- Measure voltages and report times that are subject to a goal.
- Measure device power dissipation subject to a goal.
- Measure statements replace the tabular or plot output.
- Parameters used as element values.
- Parameter optimizing function.
- Transient analysis with SWEEP optimizing.

Example

This example is based on demonstration netlist rcopt.sp, which is available in directory $<installdir>/demo/hspice/ciropt:

```
*file: rcopt.sp optimize the power dissapation and time constant
* of an rc network
*
* optimize to the goals of 1sec time constant and 50mwatts rms
power.
* optrc identifies rx and cx as optimization parameters and sets
* their starting, minimum, and maximum values. measure statement
rc1
* calculates the rc time constant. ( .3679=e**-rc ) where the
goal is
* rc=1sec. measure statement rc2 calculates the rms power where the
* goal is 50 milliwatts.
*
* hspice features used:
*    - measure voltages and report times subject to goal
*    - measure device power dissapation subject to goal
*    - element value parameterization
*    - parameter optimization function
*    - transient with sweep optimize
*
.option post

.param rx=optrc(.5, 1e-2, 1e+2)
.param cx=optrc(.5, 1e-2, 1e+2)

.measure tran rc1 trig at=0 targ v(1) val=.3679 fall=1 goal=1sec
.measure tran rc2 rms p(r1) goal=50mwatts

.model opt1 opt

.tran .1 2$ initial values
.tran .1 2 sweep optimize=optrc results=rc1,rc2 model=opt1
.tran .1 2$ analysis using final optimized values
```

The optimizer initially uses the Steepest Descent method as the fastest approach to the solution. It then uses the Gauss-Newton method to find the solution. During this process, the Marquardt Scaling Parameter becomes very small, but starts to increase again if the solution starts to deviate. If this happens, the optimizer chooses between the two methods to work toward the solution again.

If the optimizer does not attain the optimal solution, it prints both an error message, and a large Marquardt Scaling Parameter value.

Number of Function Evaluations:

This is the number of analyses (for example, finite difference or central difference) needed to find a minimum of the function.

Number of Iterations:

This is the number of iterations needed to find the optimized or actual solution.

Optimized Parameters OPTRC

```
.param rx=  7.4823   $   55.6965  5.7945m
.param cx=133.9934m  $   44.3035  5.1872m
```

Figure 67 Power Dissipation and Time Constant (VOLT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result

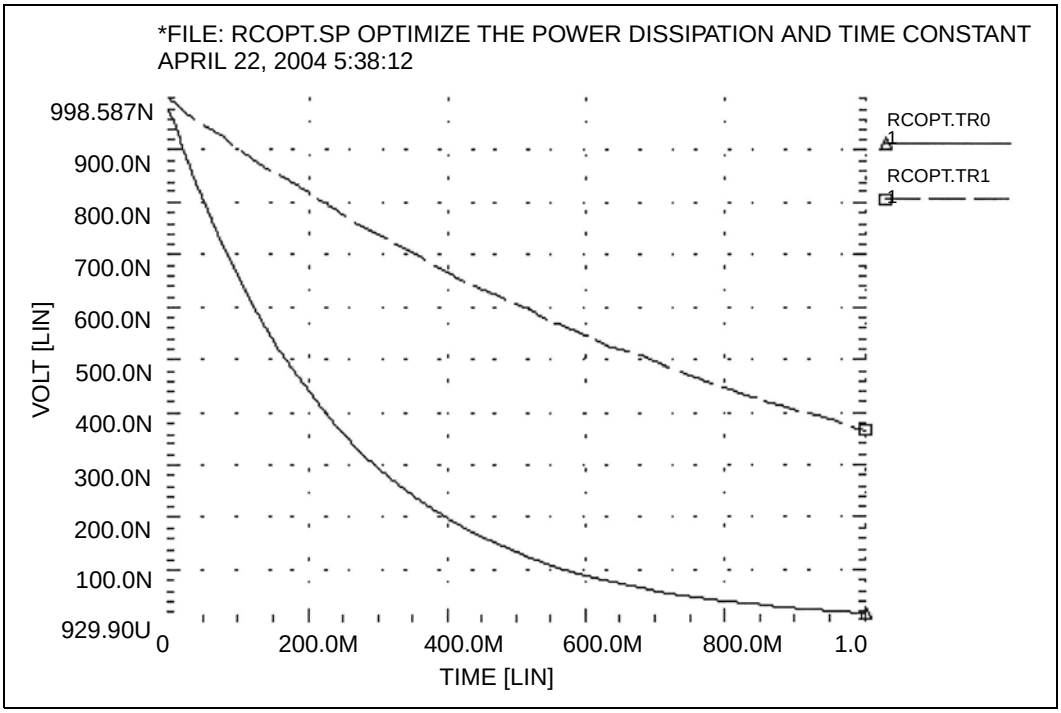
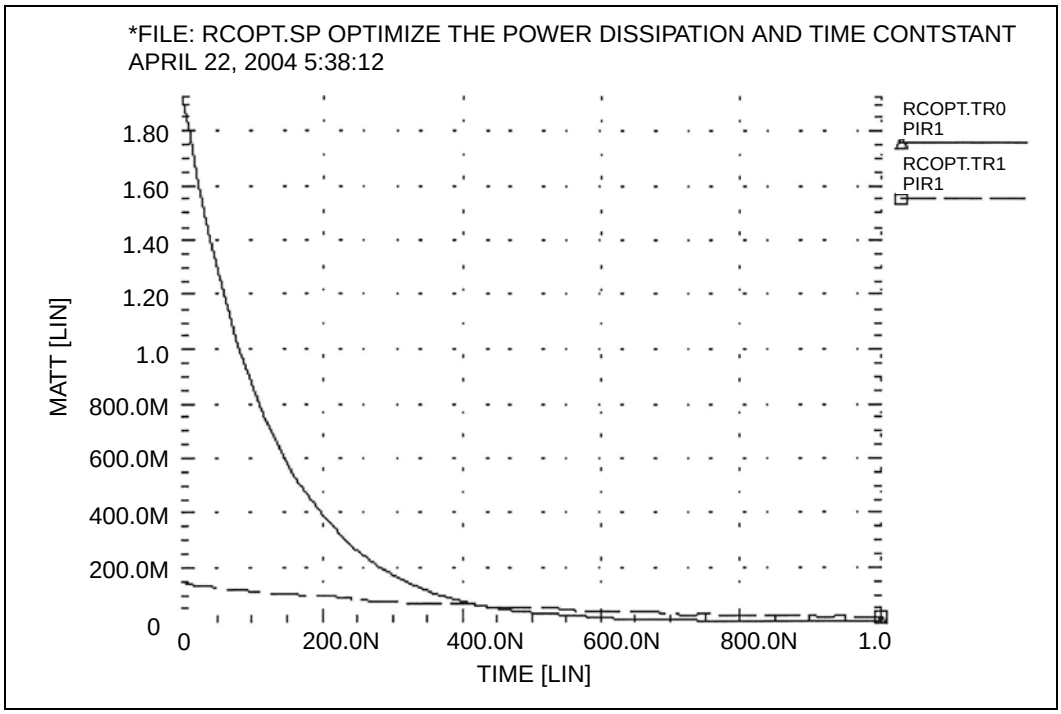


Figure 68 Power Dissipation and Time Constant (WATT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result



Optimizing CMOS Tristate Buffer

The example circuit is an inverting CMOS tristate buffer. The design targets are:

- Rising edge delay of 5 ns (input 50% voltage to output 50% voltage).
- Falling edge delay of 5 ns (input 50% voltage to output 50% voltage).
- RMS power dissipation should be as low as possible.
- Output load consists of:
  - pad capacitance
  - leadframe inductance
  - 50 pF capacitive load

The HSPICE strategy is:

- Simultaneously optimize both the rising and falling delay buffer.
- Set up the internal power supplies, and the tristate enable as global nodes.

- Optimize all device widths except:
  - Initial inverter (assumed to be standard size).
  - Tristate inverter and part of the tristate control (optimizing is not sensitive to this path).
- Perform an initial transient analysis for plotting purposes. Then optimize and perform a final transient analysis for plotting.
- To use a weighted RMS power measure, specify unrealistically-low power goals. Then use MINVAL to attenuate the error.

Input Netlist File to Optimize a CMOS Tristate Buffer This example is based on demonstration netlist trist_buf_opt.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
*Tri-State input/output Optimization
.OPTION nomod post
+ defl=1.2u relv=1e-3 absvar=.5 relvar=.01

*Circuit Input

.global lgnd lvcc enb
.macro buff data out
mp1 DATAN DATA LVCC LVCC p w=35u
mn1 DATAN DATA LGND LGND n w=17u
mp2 BUS DATAN LVCC LVCC p w=wp2
mn2 BUS DATAN LGND LGND n w=wn2
mp3 PEN PENN LVCC LVCC p w=wp3
mn3 PEN PENN LGND LGND n w=wn3
mp4 NEN NENN LVCC LVCC p w=wp4
mn4 NEN NENN LGND LGND n w=wn4
mp5 OUT PEN LVCC LVCC p w=wp5 l=1.8u
mn5 OUT NEN LGND LGND n w= wn5 l=1.8u
mp10 NENN BUS LVCC LVCC p w=wp10
mn12 PENN ENB NENN LGND n w=wn10
mn10 PENN BUS LGND LGND n w=wn10
mp11 NENN ENB LVCC LVCC p w=wp11
mp12 NENN ENBN PENN LVCC p w=wp11
mn11 PENN ENBN LGND LGND n w=80u
mp13 ENBN ENB LVCC LVCC p w=35u
mn13 ENBN ENB LGND LGND n w=17u
cbus BUS LGND 1.5pf
cpad OUT LGND 5.0pf
.ends

* * input signals *
```

```
vcc VCC GND 5V
lvcc vcc lvcc 6nh
lgnd lgnd gnd 6nh
vin DATA LGND pl (0v 0n, 5v 0.7n)
vinb DATAbar LGND pl (5v 0n, 0v 0.7n)
ven ENB GND 5V
** circuit **
x1 data out buff
cext1 out GND 50pf
x2 databar outbar buff
cext2 outbar GND 50pf

*Optimization Parameters

.param
+ wp2=opt1(70u,30u,330u)
+ wn2=opt1(22u,15u,400u)
+ wp3=opt1(400u,100u,500u)
+ wn3=opt1(190u,80u,580u)
+ wp4=opt1(670u,150u,800u)
+ wn4=opt1(370u,50u,500u)
+ wp5=opt1(1200u,1000u,5000u)
+ wn5=opt1(600u,400u,2500u)
+ wp10=opt1(240u,150u,450u)
+ wn10=opt1(140u,30u,280u)
+ wp11=opt1(240u,150u,450u)

*Control Section

.tran 1ns 16ns
.tran .5ns 15ns sweep optimize=opt1
+ results=tfopt,tropt,rmspowo model=optmod
** put soft limit for power with minval setting (i.e. values
** less than 1000mw are less important)
.measure rmspowo rms power goal=100mw minval=1000mw
.meas tran tfopt trig v(data) val=2.5 rise=1 targ v(out)
+ val=2.5 fall=1 goal=5.0n
.meas tran tropt trig v(databar) val=2.5 fall=1 targ
+ v(outbar) val=2.5 rise=1 goal=5.0n
.model optmod opt itropt=40 max=1e5 difsiz=1e-5
*.tran 1ns 16ns
* output section *
.probe tran v(data) v(out)
.probe tran v(databar) v(outbar)

*Model Section
```

```
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500 KAPPA=.25 KP=30U
+ ETA=.03 THETA=.04 VMAX=2E5 NSUB=9E16 TOX=500E-10
+ GAMMA=1.5 PB=0.6 JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10
+ CGSO=200P CGDO=200P CGBO=300P
.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 KP=15U
+ ETA=.03 THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=500E-10
+ NFS=1E11 GAMMA=.672 PB=0.6 JS=.1M XJ=0.5U LD=0.0
+ NSS=2E10 CGSO=200P CGDO=200P CGBO=300P
.end
```

Figure 69 Tristate Buffer Optimization Circuit

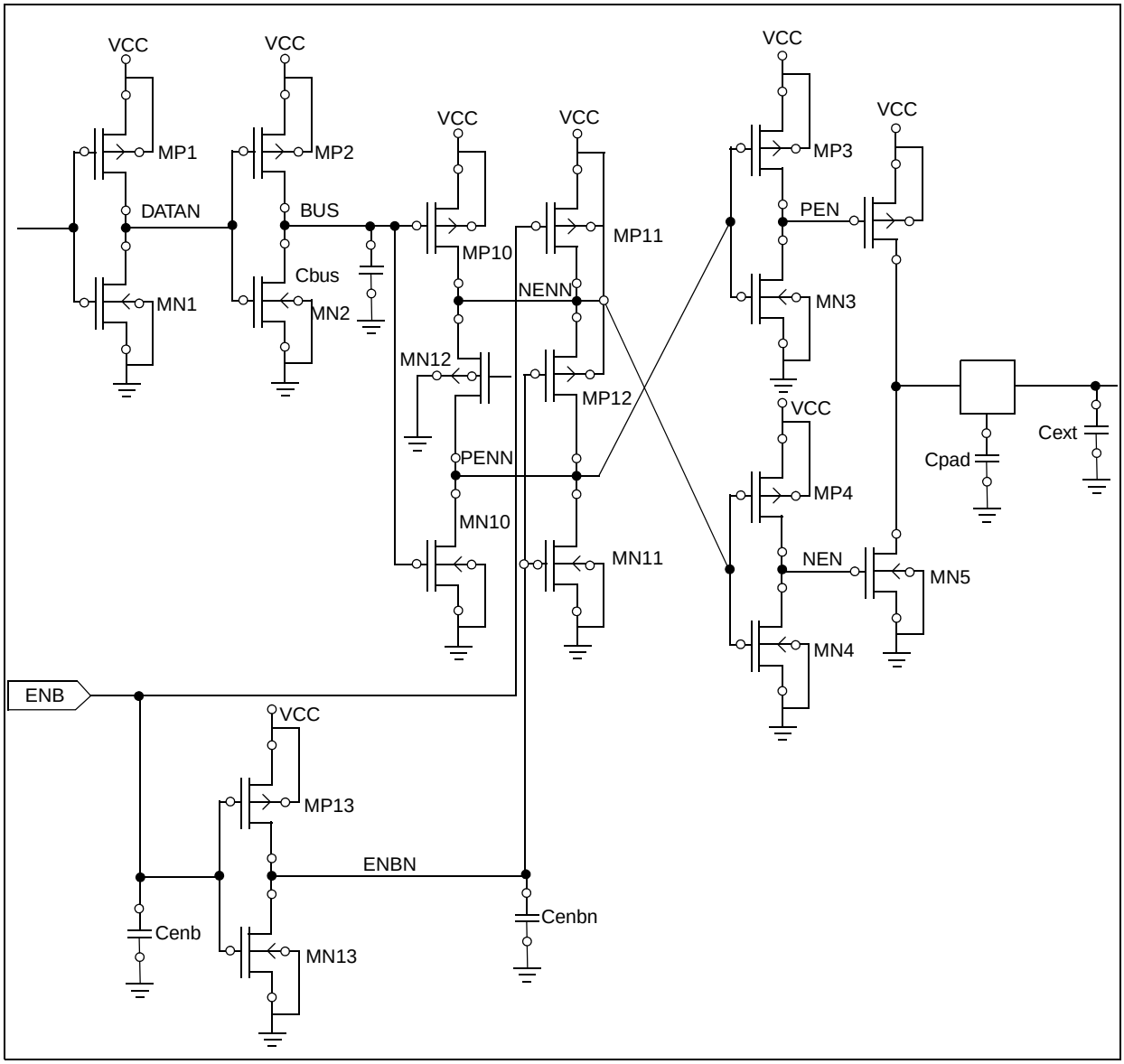
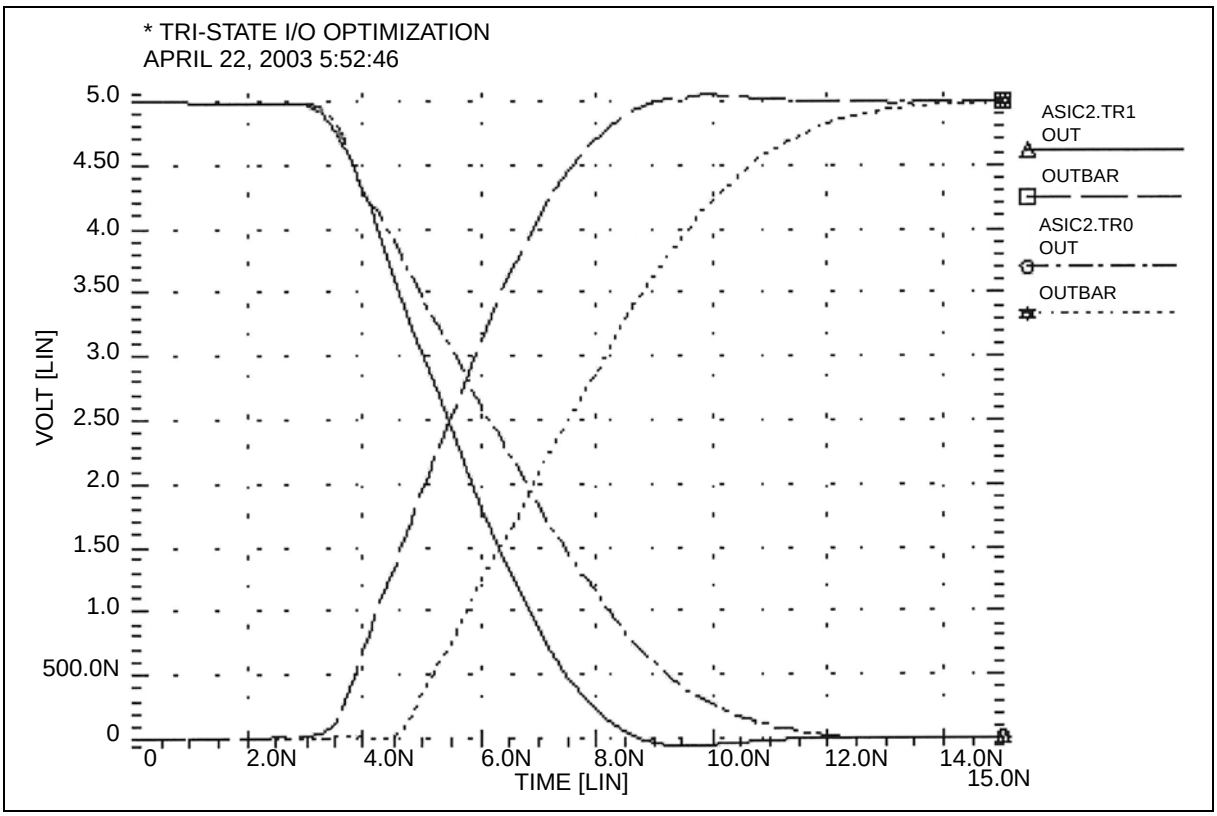


Figure 70 Tristate Input/Output Optimization ACIC2B.TR0 = Before Optimization, ACIC2B.TR1=Optimized Result



BJT S Parameters Optimization

The following example optimizes the S parameters to match those specified for a set of measurements. The `.DATA` measured in-line data statement contains these measured S parameters as a function of frequency. The model parameters of the microwave transistor (`LBB`, `LCC`, `LEE`, `TF`, `CBE`, `CBC`, `RB`, `RE`, `RC`, and `IS`) vary. As a result, the measured S parameters (in the `.DATA` statement) match the calculated S parameters from the simulation results.

This optimization uses a 2n6604 microwave transistor, and an equivalent circuit that consists of a BJT, with parasitic resistances and inductances. The BJT is biased at a 10 mA collector current (0.1 mA base current at DC bias and bf=100).

Key HSPICE Features Used

- .NET command to simulate network analyzer action.
- .AC optimization.
- Optimized element and model parameters.
- Optimizing, compares measured S Parameters to calculated parameters.
- S Parameters used in magnitude and phase (real and imaginary available).
- Weighting of data-driven frequency versus S Parameter table. Used for the phase domain.

Input Netlist File for Optimizing BJT S Parameters

```
* BJTOPT.SP BJT S PARAMETER OPTIMIZATION
.OPTION ACCT NOMOD POST=2
```

BJT Equivalent Circuit Input

Use the bjtopt.sp netlist file located in your $<installdir>/demo/hspice/devopt directory for optimizing BJT S Parameters.

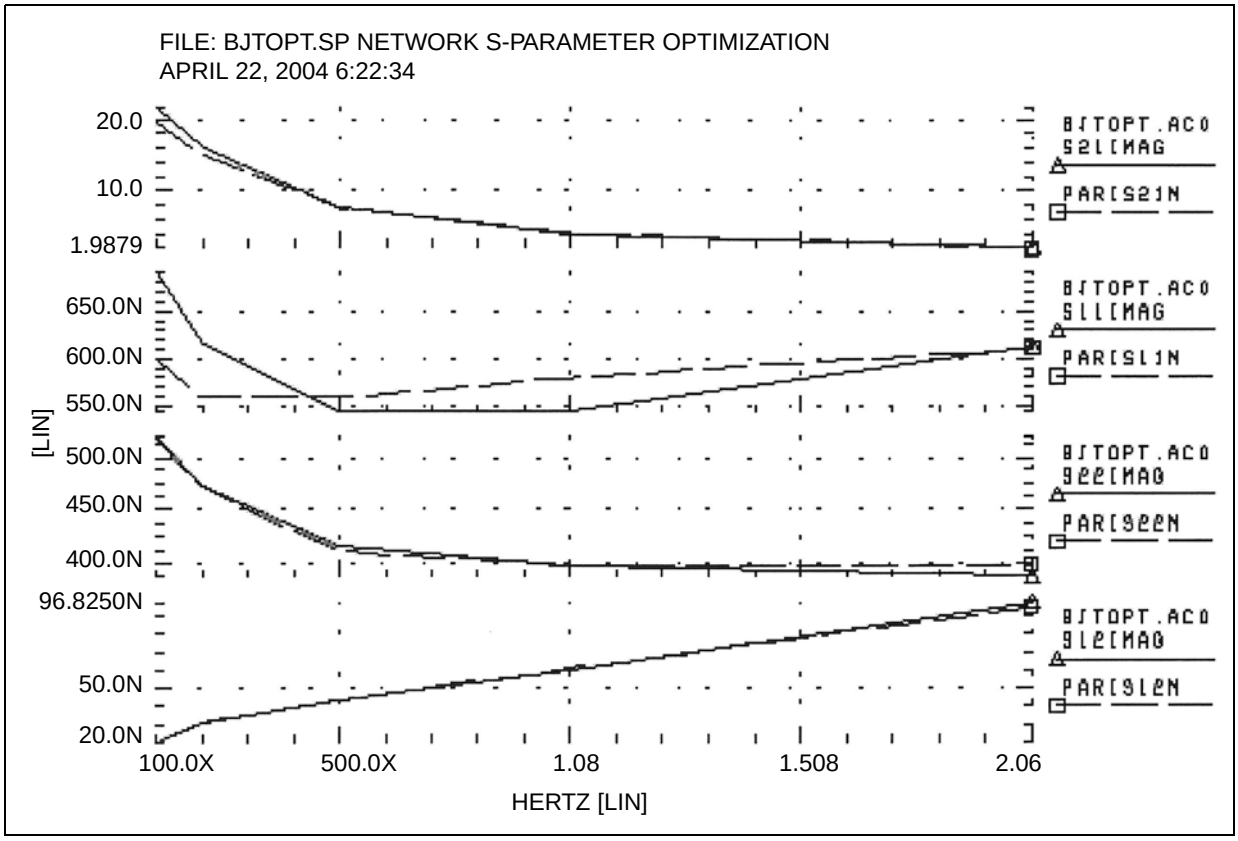
Optimization Results

```
RESIDUAL SUM OF SQUARES    =5.142639e-02
NORM OF THE GRADIENT       =6.068882e-02
MARQUARDT SCALING PARAMETER=0.340303
CO. OF FUNCTION EVALUATIONS=170
NO. OF ITERATIONS          =35
```

The maximum number of iterations (25) was exceeded. However, the results probably are accurate. Increase ITROPT accordingly.

```
Optimized Parameters OPT1- Final Values
***OPTIMIZED PARAMETERS OPT1 SENS  %NORM-SEN
.PARAM  LBB = 1.5834N  $ 27.3566X  2.4368
.PARAM  LCC = 2.1334N  $ 12.5835X  1.5138
.PARAM  LEE  =723.0995P  $254.2312X 12.3262
.PARAM  TF  =12.7611P  $  7.4344G 10.0532
.PARAM  CBE  =620.5195F  $ 23.0855G  1.5300
.PARAM  CBC = 1.0263P  $346.0167G 44.5016
.PARAM  RB   =  2.0582   $ 12.8257M  2.3084
.PARAM  RE   =869.8714M  $ 66.8123M  4.5597
.PARAM  RC  =54.2262   $  3.1427M 20.7359
.PARAM  IS  =99.9900P  $  3.6533X 34.4463M
```

Figure 71 BJT-S Parameter Optimization



BJT Model DC Optimization

The goal is to match forward and reverse Gummel plots obtained from a HP4145 semiconductor analyzer by using the HSPICE `LEVEL=1` Gummel-Poon BJT model. Because Gummel plots are at low base currents, HSPICE does not optimize the base resistance. HSPICE also does not optimize forward and reverse Early voltages (VAF and VAR), because simulation did not measure VCE data.

The key feature in this optimization is incremental optimization:

1. HSPICE first optimizes the forward-Gummel data points.
2. HSPICE updates forward-optimized parameters into the model.

   After updating, you cannot change these parameters.
3. HSPICE next optimizes the reverse-Gummel data points.

BJT Model DC Optimization Input Netlist File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/devopt/opt_bjt.sp
```

Figure 72 BJT Optimization Forward Gummel Plots

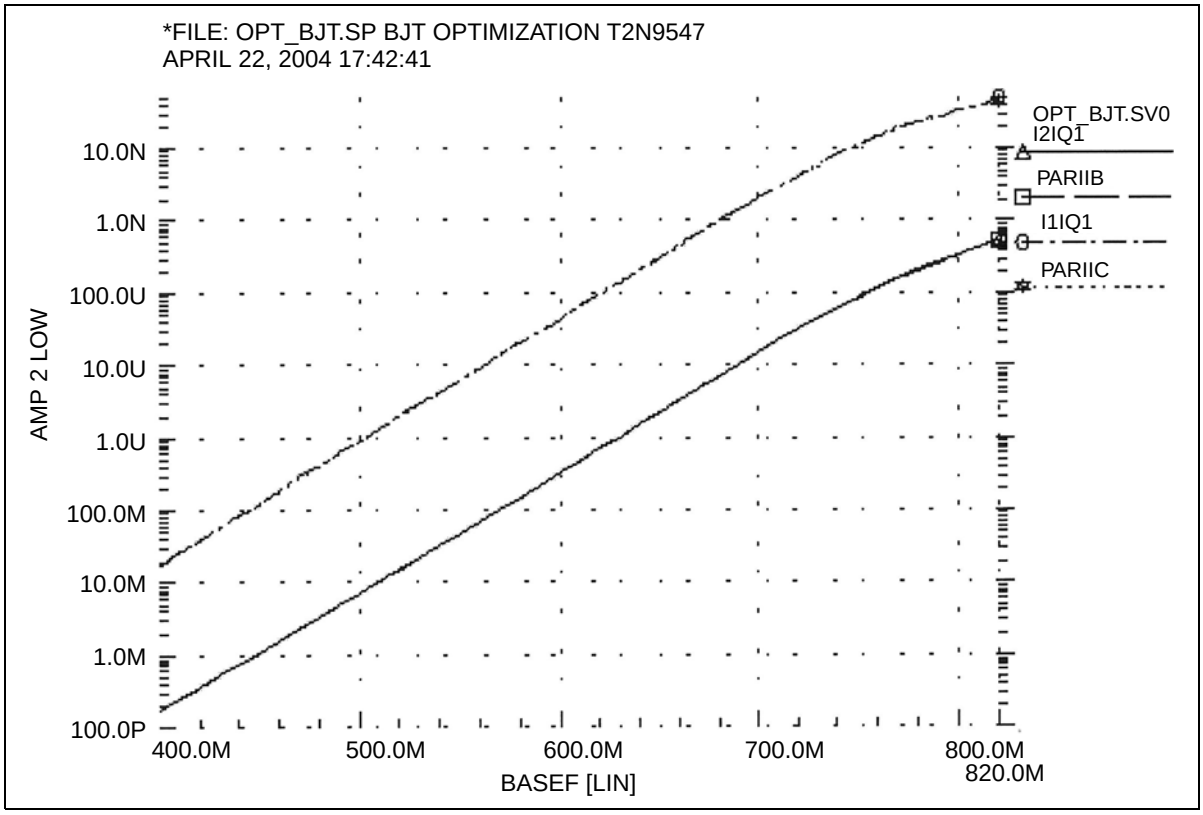
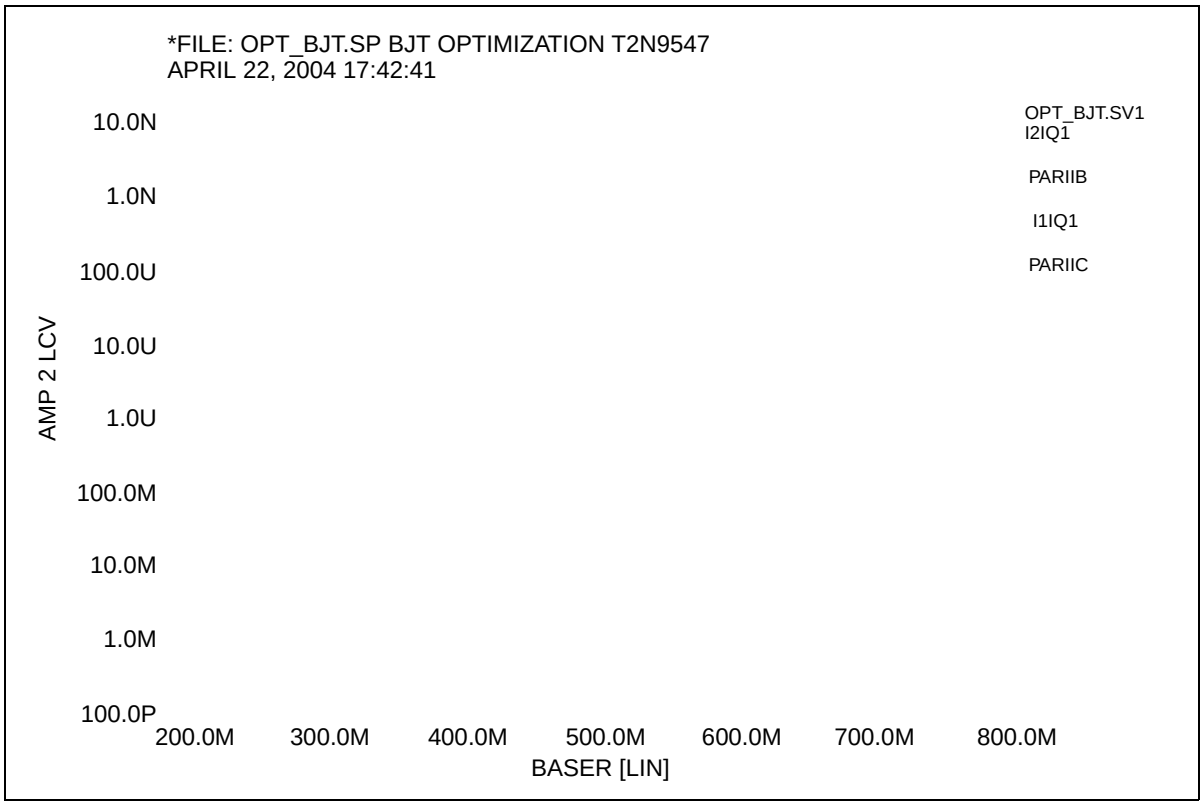


Figure 73 BJT Optimization Reverse Gummel Plots



Optimizing GaAsFET Model DC

This example circuit is a high-performance, GaAsFET transistor. The design target is to match HP4145 DC measured data to the HSPICE LEVEL=3 JFET model.

The HSPICE strategy is:

- `.MEASURE IDSERR` is an `ERR1` type function. It provides linear attenuation of the error results starting at 20 mA. This function ignores all currents below 1 mA. The high-current fit is the most important for this model.
- The `OPT1` function simultaneously optimizes all DC parameters.
- The `.DATA` statement merges TD1.dat and TD2.dat data files.
- The graph plot model sets the `MONO=1` parameter to remove the retrace lines from the family of curves.

GaAsFET Model DC Optimization Input Netlist File

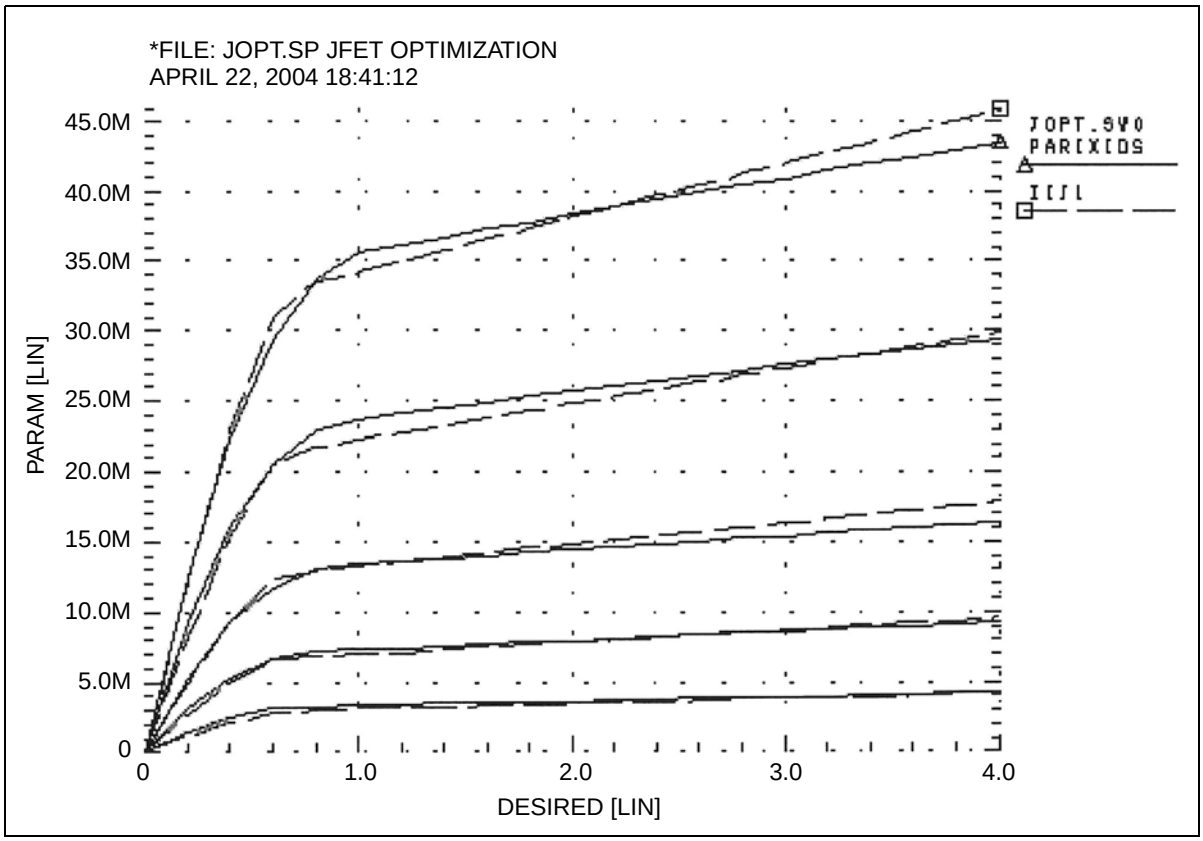
This example is based on demonstration netlist jopt.sp, which is available in directory $<installdir>/demo/hspice/devopt:

```
 * file opt_bjt.sp bjt optimization t2n3947
 *
 * optimize the dc forward and reverse characteristics from a
gummel probe
 * all dc gummel-poon dc parameters except base resistance and early
 * voltages optimized
 *

 $..tighten the simulator convergence properties
 .option post nomod ingold=2 nopage vntol=1e-10
 + numdgt=5 reli=1e-4 relv=1e-4

 $..optimization convergence controls
 .model optmod opt relin=1e-4 itropt=30 grad=1e-5 close=10 cut=2
 + cendif=1e-6 relout=1e-4 max=1e6

 *****room temp device*******
 vber base 0 vbe
 vbcr base col vbc
 q1 col base 0 bjtmod

 $..the model and inital guess
 .model bjtmod npn
 + iss = 0. xtf = 1. ns = 1.
 + cjs = 0. vjs = 0.50000 ptf = 0.
 + mjs = 0. eg = 1.10000 af = 1.
 + itf = 0.50000 vtf = 1.00000
 + fc = 0.95000 xcjc = 0.94836
 + subs = 1
 + tf=0.0 tr=0.0 cje=0.0 cjc=0.0 mje=0.5 mjc=0.5 vje=0.6 vjc=0.6
 + rb=0.3 rc=10 vaf=550 var=300
 $..these are the optimized parameters
 + bf=bf is=is ikf=ikf ise=ise re=re
 + nf=nf ne=ne
 $..these are for reverse base opt
 + br=br ikr=ikr isc=isc
 + nr=nr nc=nc

 .param vbe=0 ib=0 ic=0 vce_emit=0 vbc=0 ib_emit=0 ic_emit=0
 + bf= opt1( 100, 50, 350)
 + is= opt1( 5e-15, 5e-16, 1e-13)
 + nf= opt1( 1.0, 0.9, 1.1)
 + ikf=opt1( 50e-3, 1e-3, 1)
 + re= opt1( 10, 0.1, 50)
```

```
 + ise=opt1( 1e-16, 1e-18, 1e-11)
 + ne= opt1( 1.5, 1.2, 2.0)

 + br= opt2( 2, 1, 10)
 + nr= opt2( 1.0, 0.9, 1.1)
 + ikr=opt2( 50e-3, 1e-3, 1)
 + isc=opt2( 1e-12, 1e-15, 1e-10)
 + nc= opt2( 1.5, 1.2, 2.0)

 .dc data=basef sweep optimize=opt1 results=ibvbe,icvbe
model=optmod
 .meas dc ibvbe err1 par(ib) i2(q1) minval=1e-14 ignore=1e-16
 .meas dc icvbe err1 par(ic) i1(q1) minval=1e-14 ignore=1e-16

 .dc data=baser sweep optimize=opt2 results=ibvber,icvber
model=optmod
 .meas dc ibvber err1 par(ib) i2(q1) minval=1e-14 ignore=1e-16
 .meas dc icvber err1 par(ic) i1(q1) minval=1e-14 ignore=1e-16

 .dc data=basef
 .print dc par(ic) i1(q1) par(ib) i2(q1)
 .dc data=baser
 .print dc par(ic) i1(q1) par(ib) i2(q1)


 .option brief=1
 .data basef
 vbe vbc ic ib
 + 0.40 0. 1.809e-08 1.793e-10
 + 0.410 0. 2.667e-08 2.546e-10
 + 0.420 0. 3.952e-08 3.640e-10
 + 0.430 0. 5.840e-08 5.198e-10
 + 0.440 0. 8.627e-08 7.487e-10
 + 0.450 0. 1.276e-07 1.082e-09
 + 0.460 0. 1.884e-07 1.564e-09
 + 0.470 0. 2.793e-07 2.278e-09
 + 0.480 0. 4.130e-07 3.318e-09
 + 0.490 0. 6.102e-07 4.836e-09
 + 0.50 0. 9.040e-07 7.083e-09
 + 0.510 0. 1.331e-06 1.033e-08
 + 0.520 0. 1.967e-06 1.514e-08
 + 0.530 0. 2.899e-06 2.219e-08
 + 0.540 0. 4.298e-06 3.261e-08
 + 0.550 0. 6.346e-06 4.786e-08
 + 0.560 0. 9.379e-06 7.036e-08
 + 0.570 0. 1.382e-05 1.034e-07
 + 0.580 0. 2.048e-05 1.522e-07
 + 0.590 0. 3.022e-05 2.236e-07
```

```
+ 0.60 0. 4.463e-05 3.288e-07
+ 0.610 0. 6.586e-05 4.834e-07
+ 0.620 0. 9.735e-05 7.119e-07
+ 0.630 0. 1.430e-04 1.043e-06
+ 0.640 0. 2.105e-04 1.529e-06
+ 0.650 0. 3.104e-04 2.250e-06
+ 0.660 0. 4.564e-04 3.298e-06
+ 0.670 0. 6.681e-04 4.819e-06
+ 0.680 0. 9.806e-04 7.058e-06
+ 0.690 0. 1.429e-03 1.028e-05
+ 0.70 0. 2.075e-03 1.492e-05
+ 0.710 0. 2.984e-03 2.151e-05
+ 0.720 0. 4.250e-03 3.070e-05
+ 0.730 0. 5.971e-03 4.353e-05
+ 0.740 0. 8.297e-03 6.089e-05
+ 0.750 0. 1.127e-02 8.364e-05
+ 0.760 0. 1.493e-02 1.126e-04
+ 0.770 0. 1.918e-02 1.504e-04
+ 0.780 0. 2.378e-02 1.984e-04
+ 0.790 0. 2.864e-02 2.587e-04
+ 0.80 0. 3.383e-02 3.345e-04
+ 0.810 0. 3.929e-02 4.270e-04
+ 0.820 0. 4.504e-02 5.386e-04
.enddata
.data baser
vbc vbe ic ib
+ 0.20 0. -9.170e-10 9.058e-10
+ 0.240 0. -2.700e-09 2.660e-09
+ 0.280 0. -8.681e-09 8.483e-09
+ 0.320 0. -3.072e-08 2.992e-08
+ 0.360 0. -1.177e-07 1.137e-07
+ 0.40 0. -4.708e-07 4.517e-07
+ 0.440 0. -1.848e-06 1.752e-06
+ 0.480 0. -6.574e-06 6.130e-06
+ 0.520 0. -2.088e-05 1.876e-05
+ 0.560 0. -6.245e-05 5.265e-05
+ 0.60 0. -1.823e-04 1.377e-04
+ 0.640 0. -5.194e-04 3.276e-04
+ 0.680 0. -1.467e-03 7.302e-04
+ 0.720 0. -3.969e-03 1.552e-03
+ 0.760 0. -9.658e-03 3.180e-03
+ 0.80 0. -2.050e-02 6.329e-03
.enddata
.end
```

Figure 74 JFET Optimization



Optimizing MOS Op-amp

The design goals for the MOS operational amplifier are:

- Minimize the gate area (and therefore the total cell area).
- Minimize the power dissipation.
- Open-loop transient step response of 100 ns for rising and falling edges.

The HSPICE strategy is:

- Simultaneously optimize two amplifier cells for rising and falling edges.
- Total power is power for two cells.
- The optimization transient analysis must be longer to allow for a range of values in intermediate results.
- All transistor widths and lengths are optimized.

- Calculate the transistor area algebraica

```
.model mosp pmos (vto=-1 kp=2.4e-5 lambda=.004
+ gamma =.37 tox=3e-8 level=3)
.model mosn nmos (vto=1.2 kp=6.0e-5 lambda=.0004
+ gamma =.37 tox=3e-8 level=3)

.param wm1=opt1(60u,20u,100u)
+ wm5=opt1(40u,20u,100u)
+ wm6=opt1(300u,20u,500u)
+ wm7=opt1(70u,40u,200u)
+ lm=opt1(10u,2u,100u)
+ bias=opt1(2.2,1.2,3.0)

.tran 2.5n 300n sweep optimize=opt1
+ results=delayr,delayf,tot_power,area_min model=opt
.model opt opt itropt=40 close=10 relin=1e-5 relout=1e-5
.tran 2n 150n
.measure delayr trig at=0 targ v(voutr) val=2.5 rise=1 goal=100ns
weight=10
.measure delayf trig at=0 targ v(voutf) val=2.5 fall=1 goal=100ns
weight=10
.measure tot_power avg power goal=10mw weight=5
.measure area_min min par(area) goal=1e-9 minval=100n
.print v(vin+) v(voutr) v(voutf)
.end
```

Figure 75 CMOS Op-amp

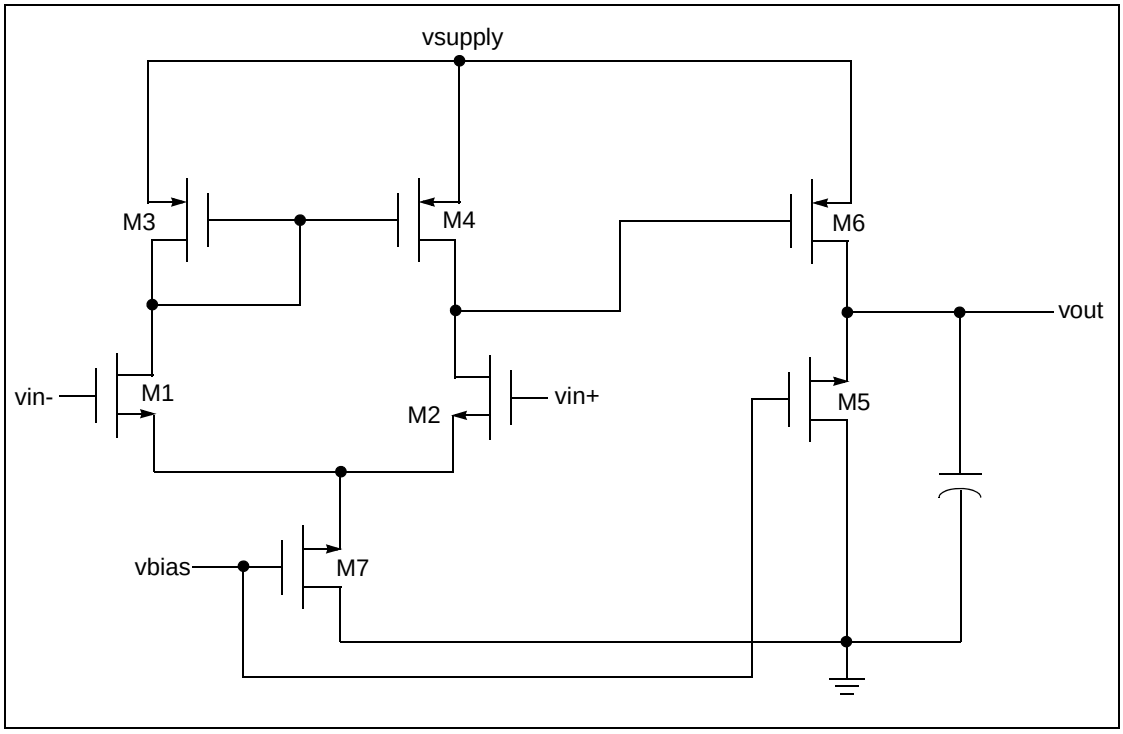
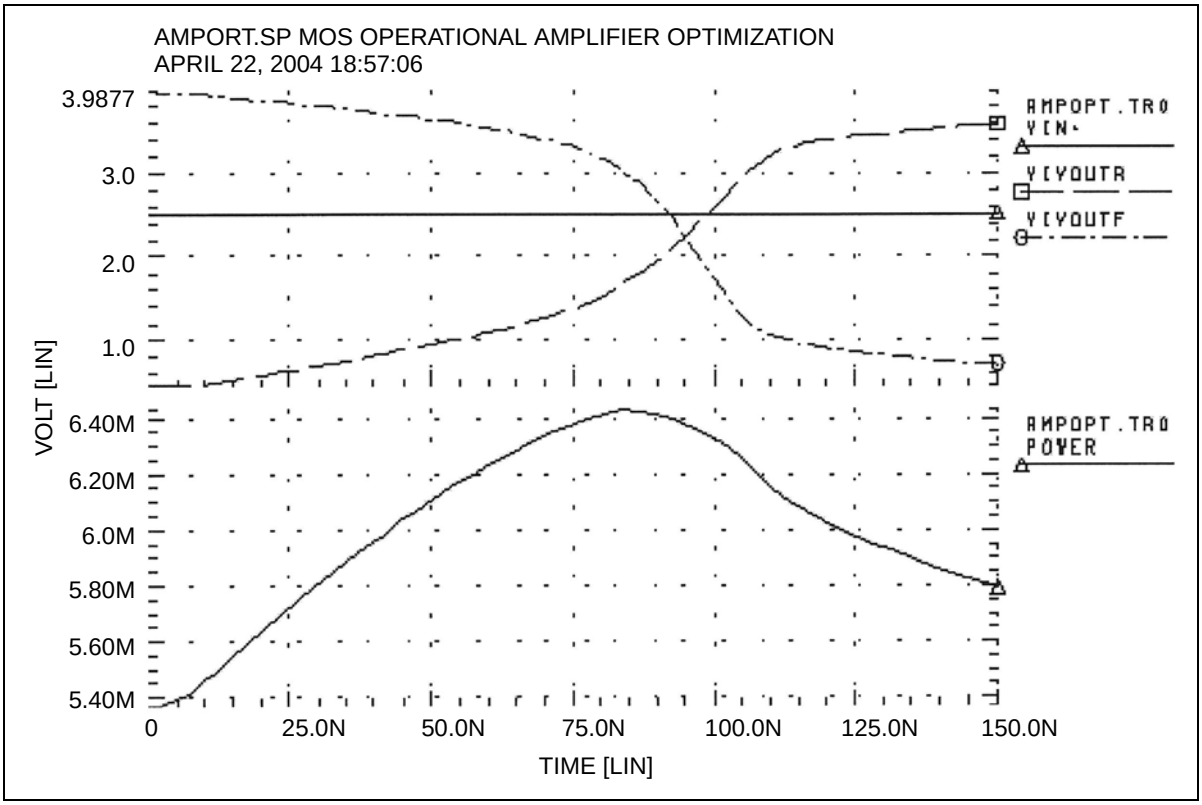


Figure 76 Operational Amplifier Optimization



18

RC Reduction

Describes RC network reduction.

Linear Acceleration

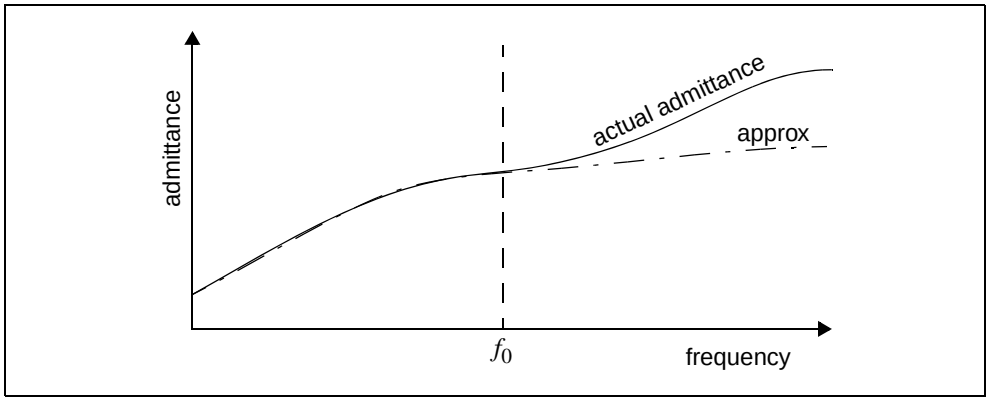
Linear acceleration, by using the SIM_LA option, accelerates the simulation of circuits that include large linear RC networks. To achieve this acceleration, HSPICE linearly reduces all matrices that represent RC networks. The result is a smaller matrix that maintains the original port behavior, yet achieves significant savings in memory and computation. Thus, the SIM_LA option is ideal for circuits with large numbers of resistors and capacitors, such as clock trees, power lines, or substrate networks.

In general, the RC elements are separated into their own network. The nodes shared by both main circuit elements (including .PRINT, .PROBE, and .MEASURE statements), and RC elements are the port nodes of the RC network,. All other RC nodes are internal nodes. The currents flowing into the port nodes are a frequency-dependent function of the voltages at those nodes.

The multiport admittance of a network represents this relationship.

- The SIM_LA option formulates matrices to represent multiport admittance.
- Then, to eliminate as many internal nodes as possible, it reduces the size of these matrices, while preserving the admittance, otherwise known as port node behavior.
- The amount of reduction depends on the *f*0 upper frequency, the threshold frequency where SIM_LA preserves the admittance. This is shown graphically in Figure 77.

Figure 77 Multiport Admittance vs. Frequency



The SIM_LA option is very effective for post-layout simulation, because of the volume of parasitics. For frequencies below *f*0, the *approx* signal matches that of the original admittance. Above f_0, the two waveforms diverge, but presumably the higher frequencies are not of interest. The lower the *f*0 frequency, the greater the amount of reduction.

For the syntax and description of this control option, see .OPTION SIM_LA in the HSPICE Command Reference.

You can choose one of two algorithms, explained in the following sections:

- PACT Algorithm
- PI Algorithm

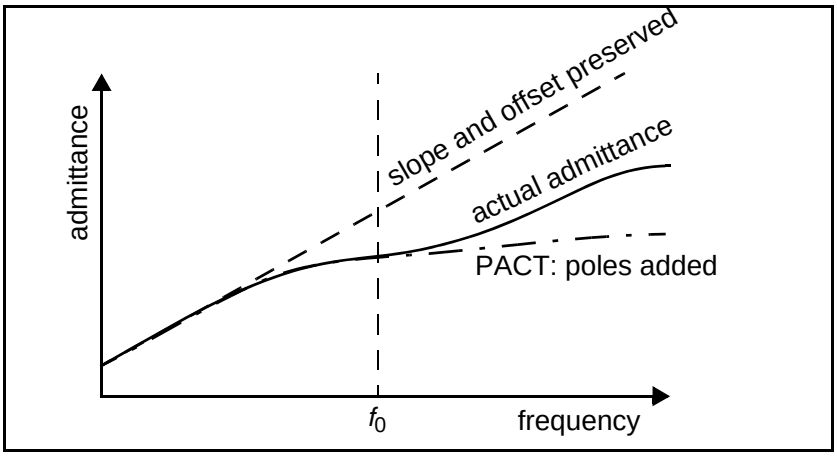
PACT Algorithm

The PACT (Pole Analysis via Congruence Transforms) algorithm reduces the RC networks in a well-conditioned manner, while preserving network stability.

- The transform preserves the first two moments of admittance at DC (slope and offset), so that DC behavior is correct (see Figure 78).
- The algorithm preserves enough low-frequency poles from the original network to maintain the circuit behavior up to a specified maximum frequency *f*0, within the specified tolerance.

This approach is more accurate between these two algorithms, and is the default.

Figure 78 PACT Algorithm



PI Algorithm

This algorithm creates a *pi* model of the RC network.

- For a two-port, the *pi* model reduced network consists of:
  - a resistor connecting the two ports, and
  - a capacitor connecting each port to ground

  The result resembles the Greek letter pi.
- For a general multiport, `SIM_LA` preserves the DC admittance between the ports, and the total capacitance that connects the ports to ground. All floating capacitances are lumped to ground.

Linear Acceleration Control Options Summary

In addition to `SIM_LA`, other options are available to control the maximum resistance and minimum capacitance values to preserve, and to limit the operating parameters of the PACT algorithm. Table 56 contains a summary of

these control options. For their syntax and descriptions, see the respective section in the HSPICE Command Reference.

Table 56 PACT Options

Syntax	Description
.OPTION SIM_LA=PACT | PI	Activates linear matrix reduction and selects between two methods. For HSPICE RF, if you set the entire netlist to ANALOG mode, linear matrix reduction does not occur.
.OPTION LA_FREQ=<value>	Upper frequency where you need accuracy preserved. *value* is the upper frequency for which the PACT algorithm preserves accuracy. If *value* is 0, PACT drops all capacitors, because only DC is of interest. The maximum frequency required for accurate reduction depends on both the technology of the circuit and the time scale of interest. In general, the faster the circuit, the higher the maximum frequency. The default is 1GHz.
.OPTION LA_MAXR=<value>	Maximum resistance for linear matrix reduction. *value* is the maximum resistance preserved in the reduction. SIM_LA assumes that any resistor greater than *value* has an infinite resistance, and drops the resistor after reduction finishes. The default is 1e15 ohms.
.OPTION LA_MINC=<value>	Minimum capacitance for linear matrix reduction. *value* is the minimum capacitance preserved in the reduction. After reduction completes, SIM_LA lumps any capacitor smaller than *value* to ground. The default is 1e-16 farads.
.OPTION LA_TIME=<value>	Minimum time for which accuracy must be preserved. *value* is the minimum switching time for which the PACT algorithm preserves accuracy. HSPICE does not accurately represent waveforms that occur more rapidly than this time. LA_TIME is simply the dual of LA_FREQ. The default is 1ns, equivalent to setting LA_FREQ=1GHz.
.OPTION LA_TOL=<value>	Error tolerance for the PACT algorithm. *value* is the error tolerance for the PACT algorithm, is between 0.0 and 1.0. The default is 0.05.

Example

In this example, the circuit has a typical risetime of 1ns. Set the maximum frequency to 1 GHz, or set the minimum switching time to 1ns.

```
.OPTION LA_FREQ = 1GHz
-or-
.OPTION LA_TIME = 1ns
```

However, if spikes occur in 0.1ns, HSPICE will not accurately simulate them. To capture the behavior of the spikes, use:

```
.OPTION LA_FREQ = 10GHz
-or-
.OPTION LA_TIME = 0.1ns
```

Note:

Higher frequencies (smaller times) increase accuracy, but only up to the minimum time step used in HSPICE.

19

Running Demonstration Files

Contains examples of basic file construction techniques, advanced features, and simulation tricks. Lists and describes several HSPICE and HSPICE RF input files.

Using the Demo Directory Tree

Demonstration Input Files on page 524 lists demonstration files, which are designed as good training examples. Most HSPICE or HSPICE RF distributions include these examples in the demo directory tree, where `$installdir` is the installation directory environment variable:

Table 57

Directory	File	Description
$*installdir*/demo/hspice	/alge	algebraic output
	/apps	general applications
	/behave	analog behavioral components
	/bench	standard benchmarks
	/bjt	bipolar components
	/cchar	characteristics of cell prototypes
	/ciropt	circuit level optimization
	/ddl	Discrete Device Library
	/devopt	device level optimization

Table 57

Directory	File	Description
	/fft	Fourier analysis (HSPICE only)
	/filters	filters
	/mag	transformers, magnetic core components
	/mos	MOS components
	/rad	radiation effects (photocurrent)
	/sources	dependent and independent sources
	/tline	filters and transmission lines
	/veriloga	Verilog-A examples

Two-Bit Adder Demo

This two-bit adder shows how to improve efficiency, accuracy, and productivity in circuit simulation. The adder is in the $*installdir*/demo/hspice/apps/mos2bit.sp (or $*installdir*/demo/hspicext/apps/mos2bit.sp for HSPICE RF) demonstration file. It consists of two-input NAND gates, defined using the NAND sub-circuit. CMOS devices include length, width, and output loading parameters. Descriptive names enhance the readability of this circuit.

One-Bit Subcircuit

The ONEBIT subcircuit defines the two half adders, with carry in and carry out. To create the two-bit adder, HSPICE or HSPICE RF uses two calls to ONEBIT. Independent piecewise linear voltage sources provide the input stimuli. The *R* repeat function creates complex waveforms.

Figure 79 One-bit Adder sub-circuit

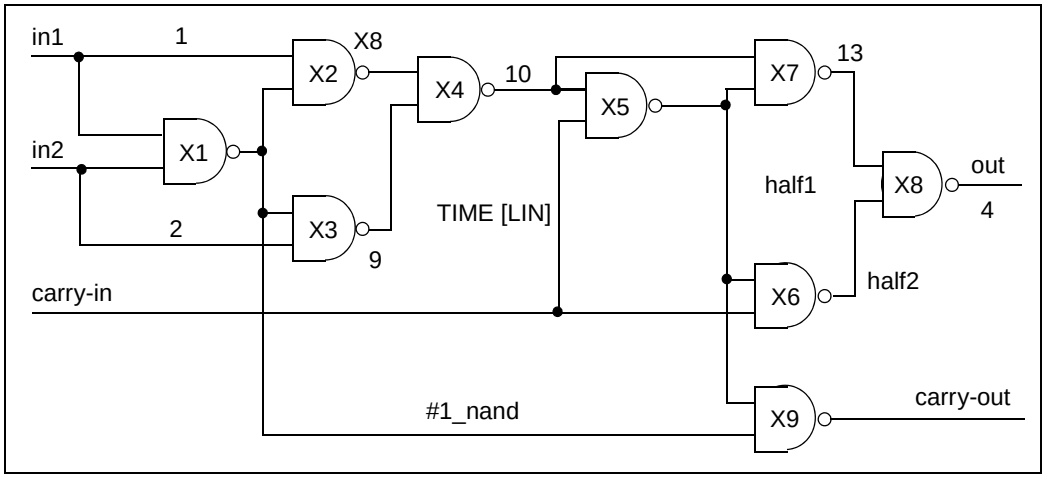


Figure 80 Two-bit Adder Circuit

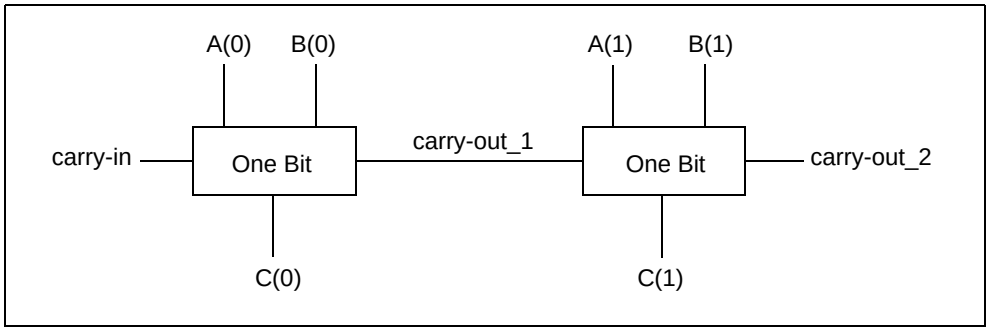
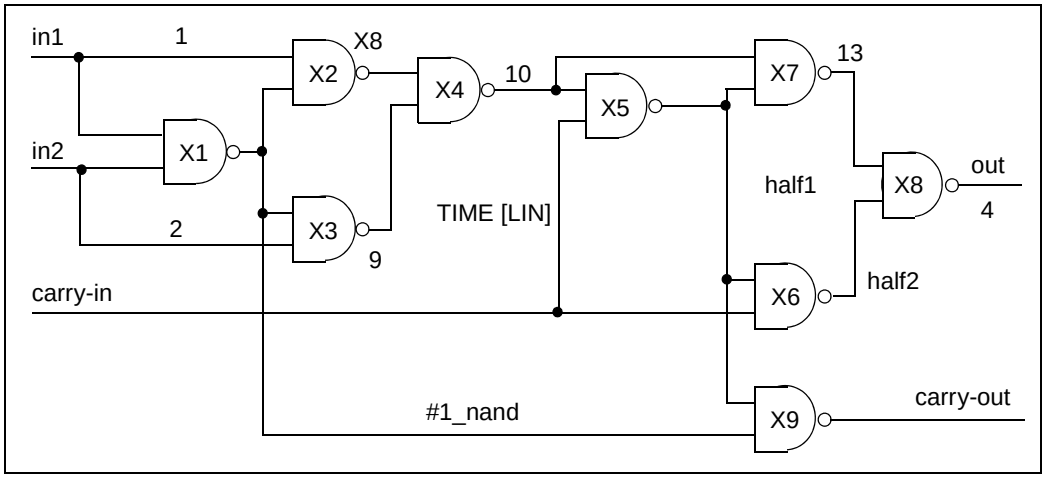


Figure 81 1-bit NAND Gate Binary Adder



MOS Two-Bit Adder Input File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/mos2bit.sp
```

MOS I-V and C-V Plotting Demo

To diagnose a simulation or modeling problem, you usually need to review the basic characteristics of the transistors. You can use this demonstration template file, $*installdir*/demo/hspice/mos/mosivcv.sp (or $*installdir*/demo/hspicext/mos/mosivcv.sp for HSPICE RF), with any MOS model. The example shows how to easily create input files, and how to display the complete graphical results. The following features aid model evaluations:

Table 58 MOS I-V and C-V Plotting Demo

Value	Description
SCALE=1u	Sets the element units to microns (not meters). Most circuit designs use microns.
DCCAP	Forces HSPICE or HSPICE RF to evaluate the voltage variable capacitors, during a DC sweep.

Table 58 MOS I-V and C-V Plotting Demo

Value	Description
node names	Makes a circuit easy to understand. Symbolic name contains up to 16 characters.
.GRAPH	.GRAPH statements create high-resolution plots. To set additional characteristics, add a graph model.

Plotting Variables

Use this template to plot internal variables, such as:

Table 59 Demo Plotting Variables

Variable	Description
i(mn1)	i1, i2, i3, or i4 can specify the true branch currents for each transistor node.
LV18(mn6)	Total gate capacitance (C-V plot).
LX7(mn1)	GM gate transconductance. (LX8 specifies GDS, and LX9 specifies GMB).

Figure 82 MOS IDS Plot

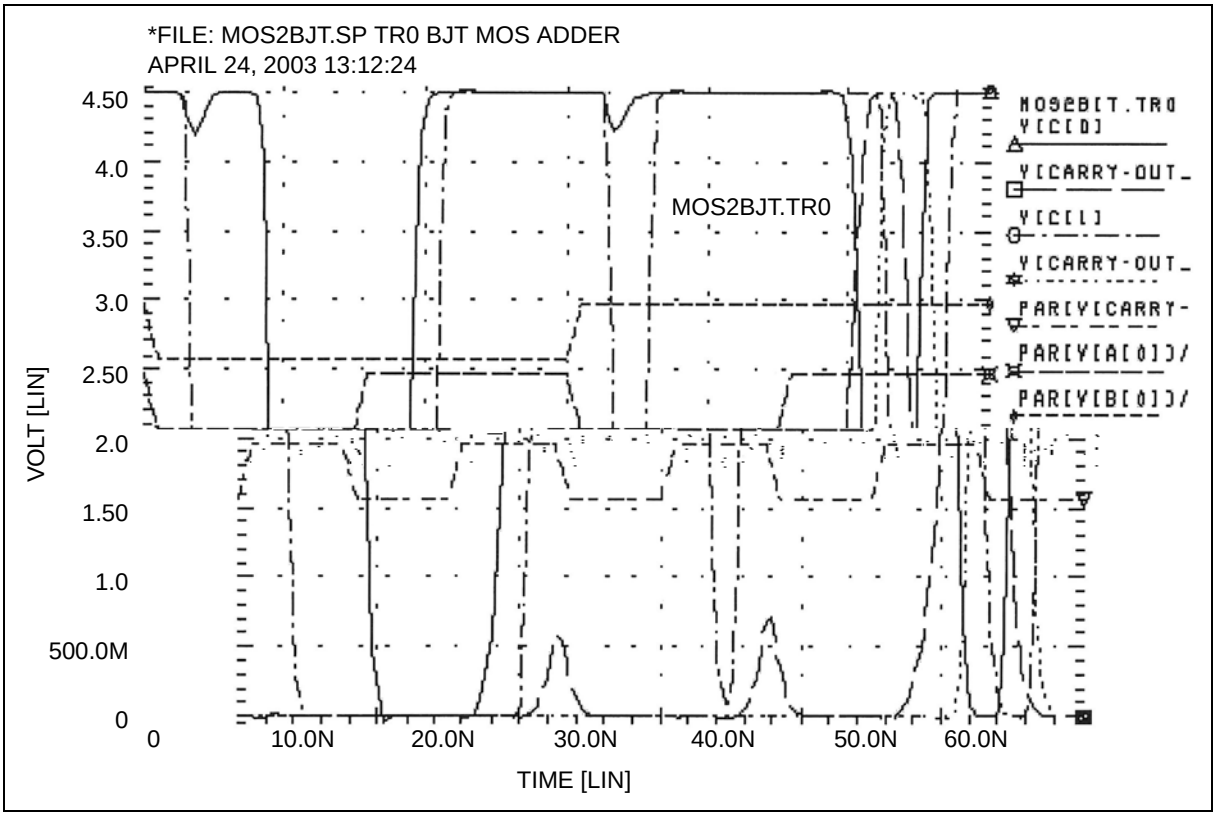


Figure 83 MOS VGS Plot

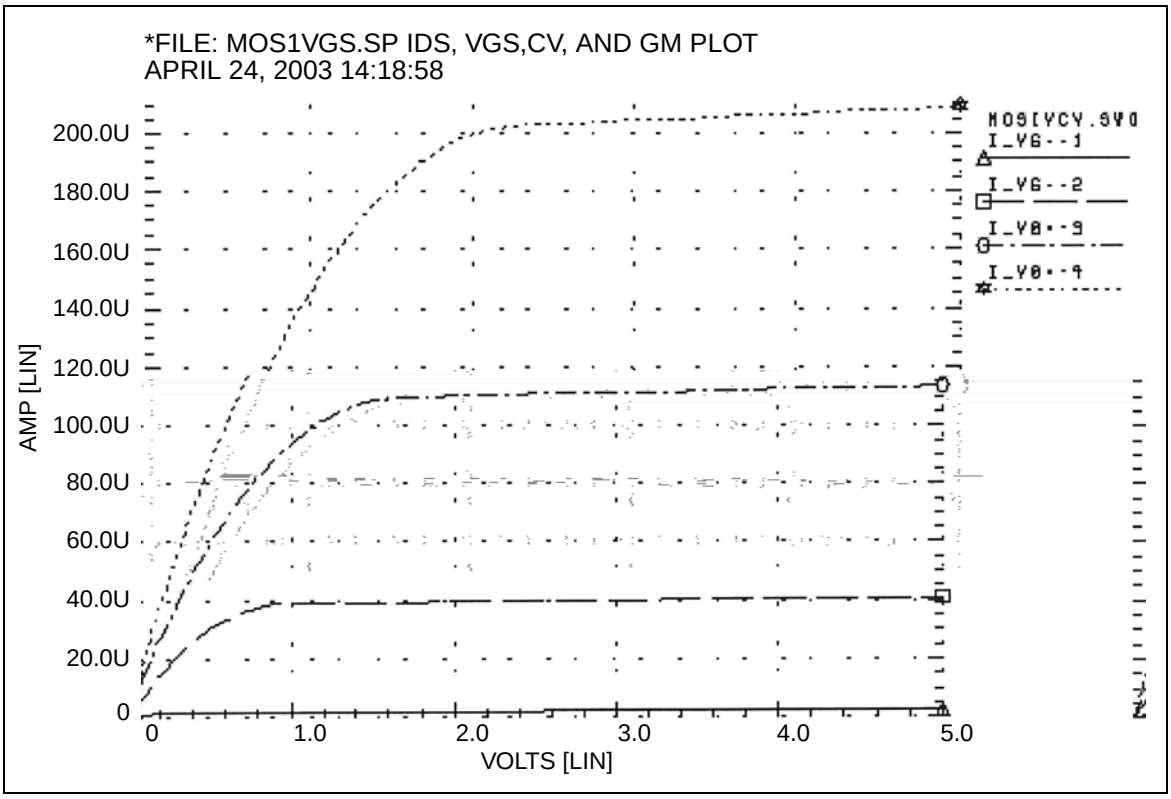


Figure 84 MOS GM Plot

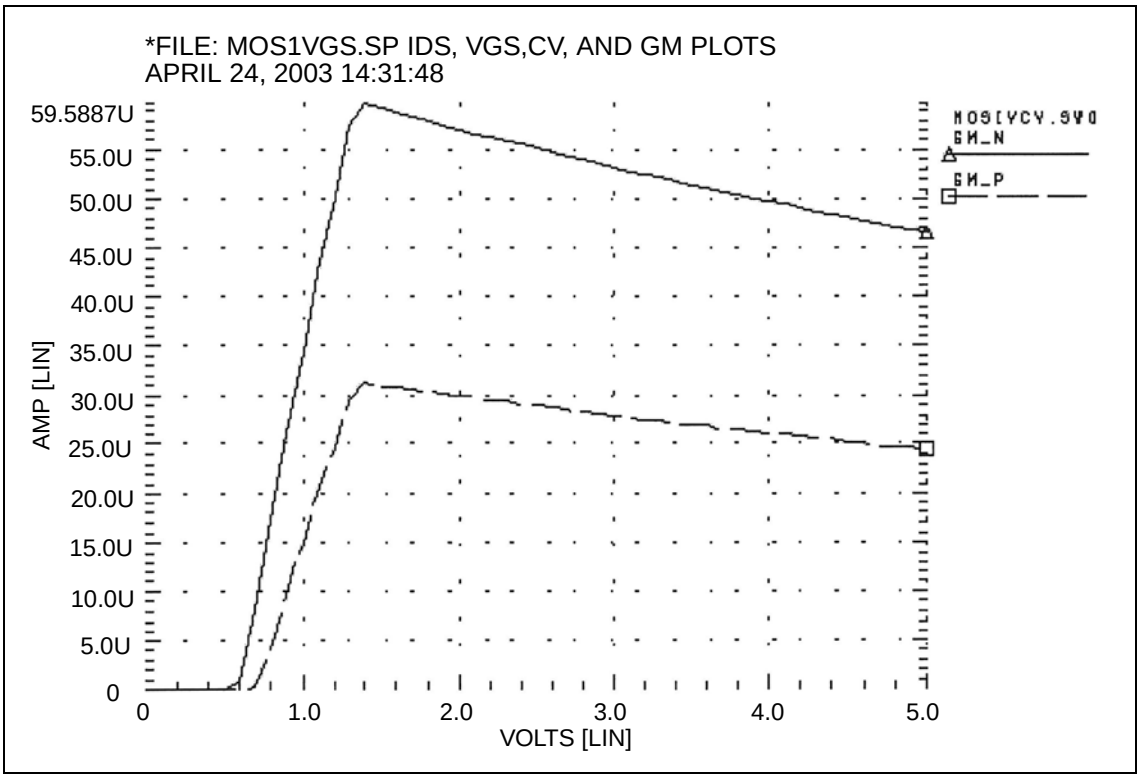
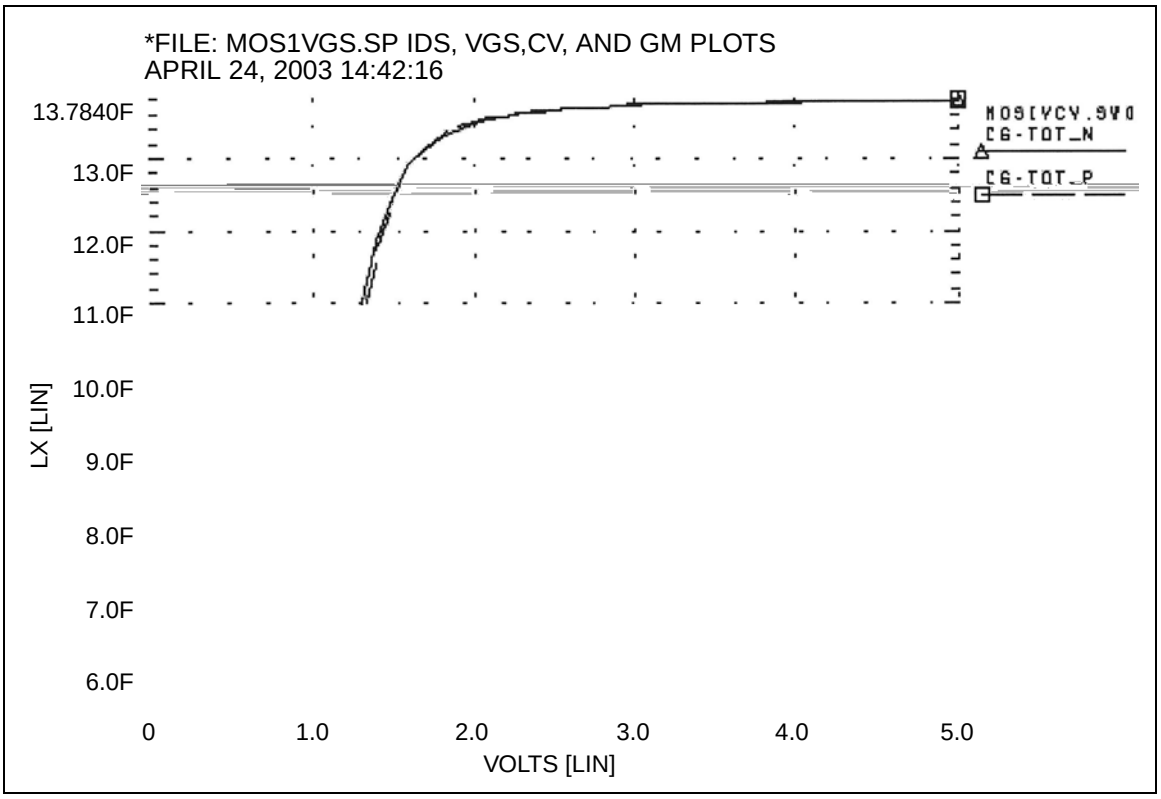


Figure 85 MOS C-V Plot



MOS I-V and C-V Plot Example Input File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/mos/mosivcv.sp`

CMOS Output Driver Demo

ASIC designers need to integrate high-performance IC parts onto a printed circuit board (PCB). The output driver circuit is critical to system performance. The $*installdir*/demo/hspice/apps/asic1.sp (or $installdir/demo/hspicext/apps/asic1.sp for HSPICE RF) demonstration file shows models for an output driver, the bond wire and leadframe, and a six-inch length of copper transmission line.

This simulation demonstrates how to:

- Define parameters, and measure test outputs.
- Use the LUMP5 macro to input geometric units, and convert them to electrical units.
- Use `.MEASURE` statements to calculate the peak local supply current, voltage drop, and power.
- Measure RMS power, delay, rise times, and fall times.
- Simulate and measure an output driver under load. The load consists of:
  - Bondwire and leadframe inductance.
  - Bondwire and leadframe resistance.
  - Leadframe capacitance.
  - Six inches of 6-mil copper, on an FR-4 printed circuit board.
  - Capacitive load, at the end of the copper wire.

Strategy

The HSPICE or HSPICE RF strategy is to:

- Create a five-lump transmission line model for the copper wire.
- Create single lumped models for leadframe loads.

Figure 86 Noise Bounce

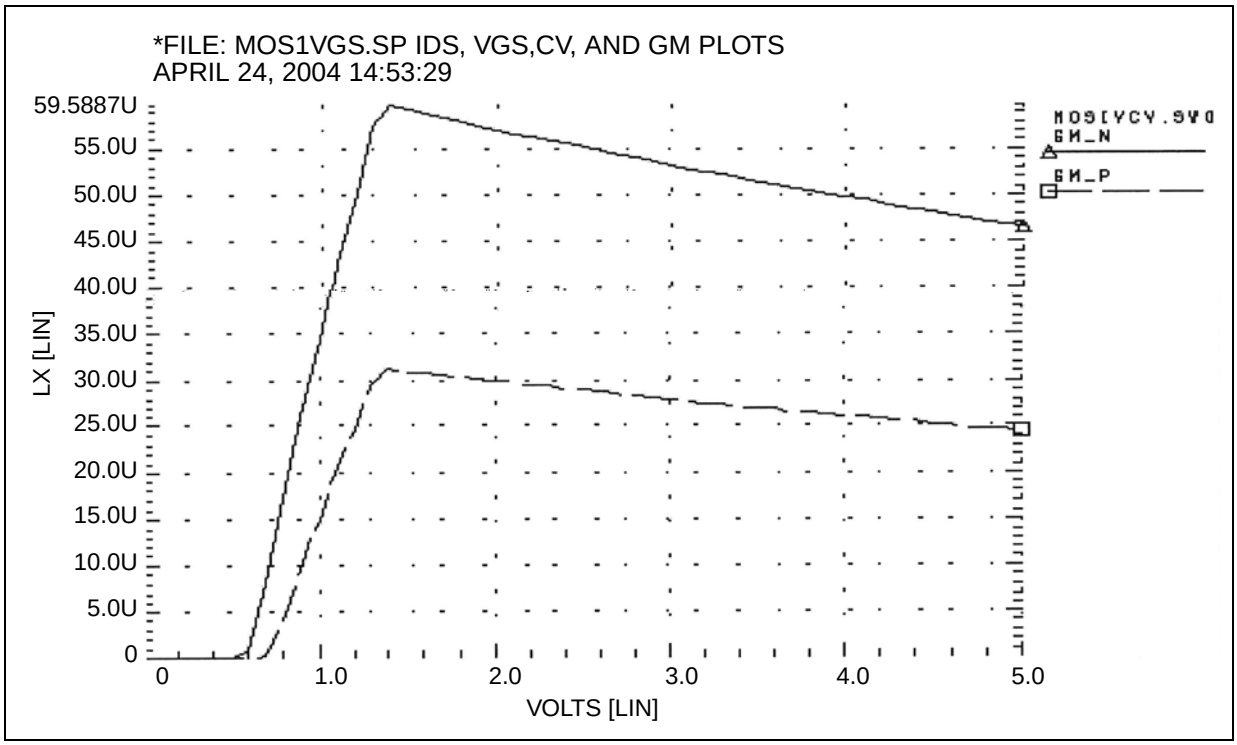


Figure 87 Asic1.sp Demo Local Supply Voltage

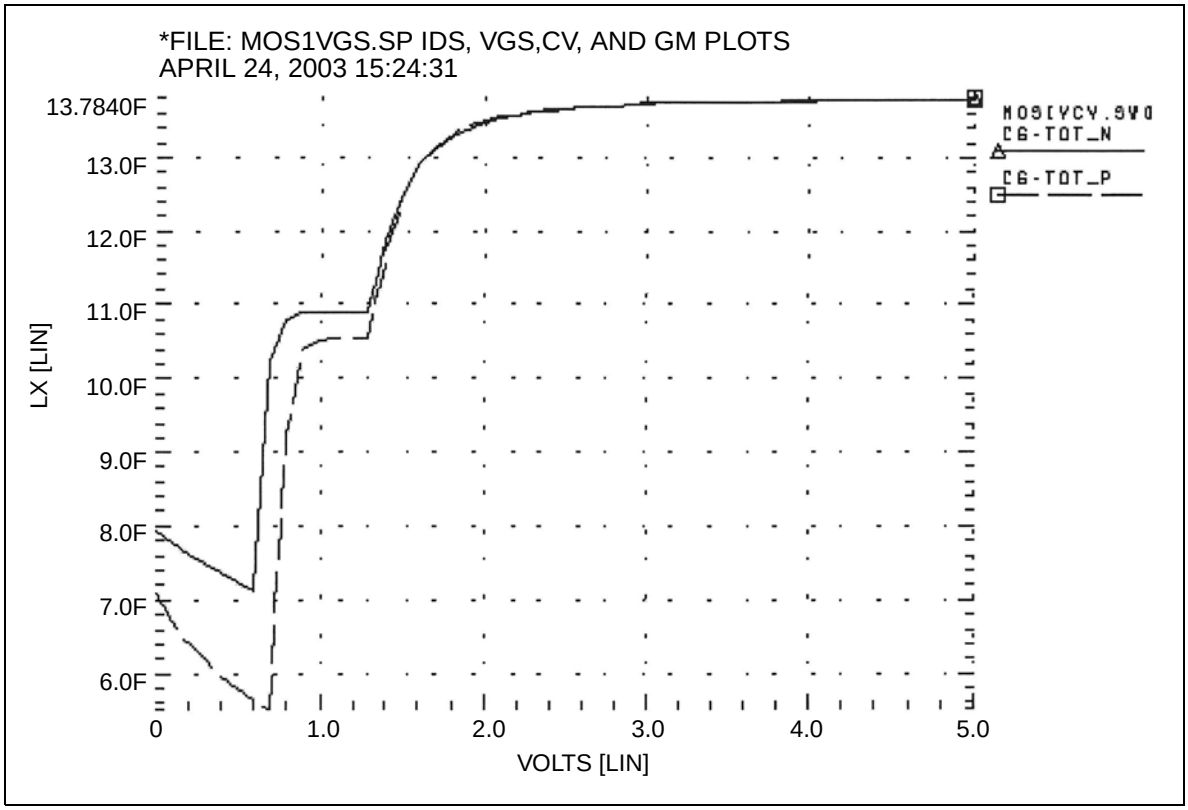


Figure 88 Asic1.sp Demo Local Supply Current

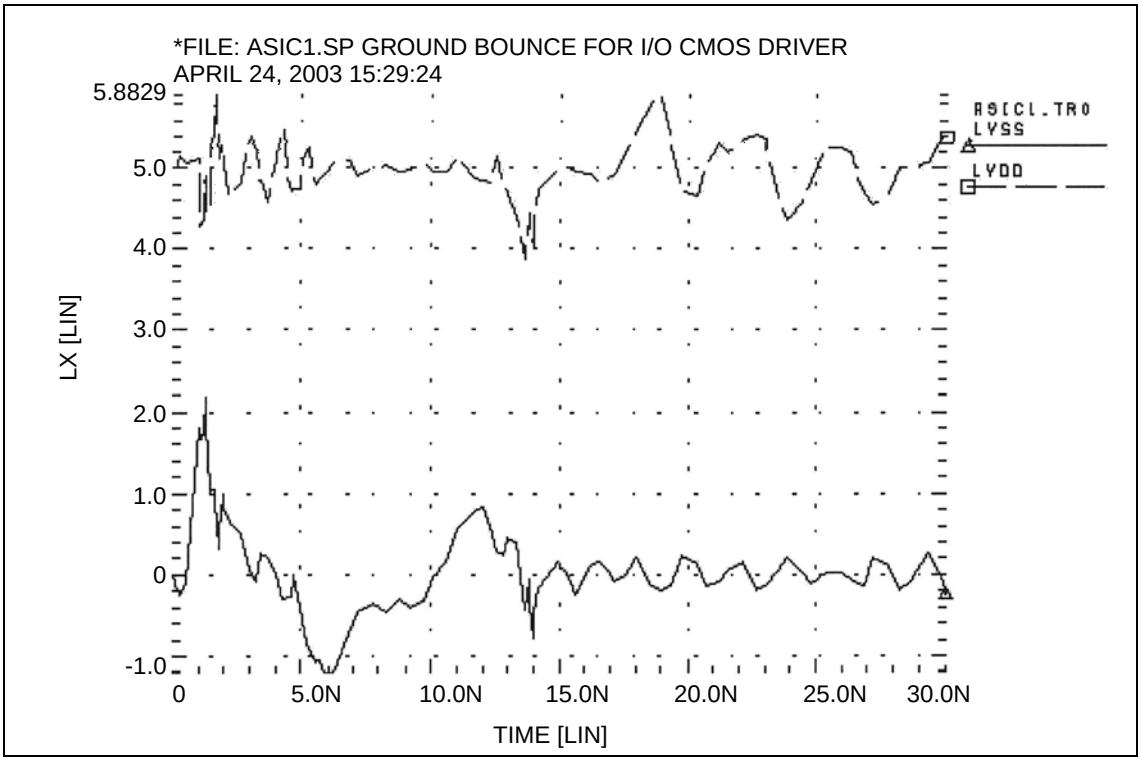
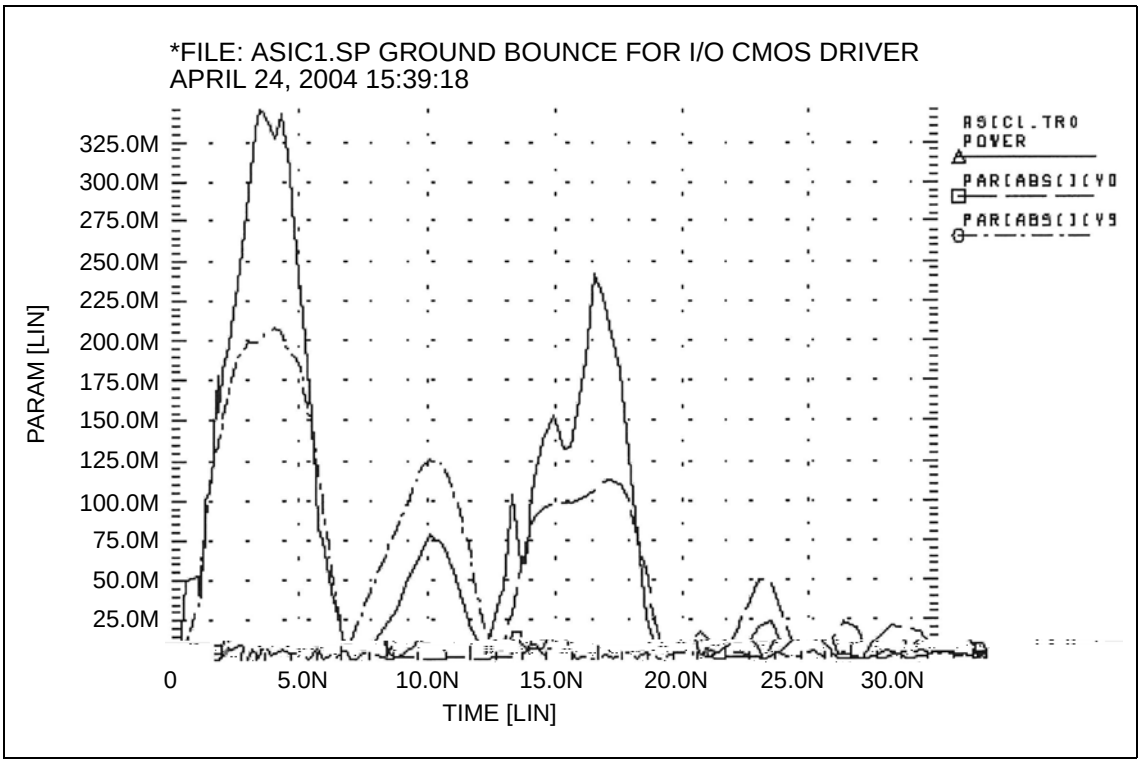


Figure 89 Asic1.sp Demo Input and Output Signals



CMOS Output Driver Example Input File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/asic1.sp
```

Temperature Coefficients Demo

SPICE-type simulators do not always automatically compensate for variations in temperature. The simulators make many assumptions that are not valid for all technologies. Many of the critical model parameters in HSPICE or HSPICE RF provide first-order and second-order temperature coefficients, to ensure accurate simulations.

You can optimize these temperature coefficients in either of two ways.

- The first method uses the TEMP DC sweep variable.

  All analysis sweeps allow two sweep variables. To optimize the temperature coefficients, one of these must be the optimize variable. Sweeping TEMP limits the component to a linear element, such as a resistor, inductor, or capacitor.

- The second method uses multiple components at different temperatures.

Example

The following example, the $*installdir*/demo/hspice/ciropt/opttemp.sp (or $*installdir*/demo/hspicext/ciropt/opttemp.sp for HSPICE RF) demo file, simulates three circuits of a voltage source. It also simulates a resistor at -25, 0, and +25°C from nominal, using the DTEMP parameter for element delta temperatures. The resistors share a common model.

You need three temperatures to solve a second-order equation. You can extend this simulation template to a transient simulation of non-linear components (such as bipolar transistors, diodes, and FETs).

This example uses some simulation shortcuts. In the internal output templates for resistors, LV1 (resistor) is the conductance (reciprocal resistance) at the desired temperature.

- You can run optimization in the resistance domain.
- To optimize more complex elements, use the current or voltage domain, with measured sweep data.

The error function expects a sweep on at least two points, so the data statement must include two duplicate points.

Input File for Optimized Temperature Coefficients

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/ciropt/opttemp.sp
```

Optimization Section

```
.model optmod opt
.dc data=RES_TEMP optimize=opt1
+           results=r@temp1,r@temp2,r@temp3
+           model=optmod
.param tc1r_opt=opt1(.001,-.1,.1)
.param tc2r_opt=opt1(1u,-1m,1m)
.meas r@temp1 err2 par(R_meas_t1) par('1.0 / lv1(r-25)')
.meas r@temp2 err2 par(R_meas_t2) par('1.0 / lv1(r0) ')
.meas r@temp3 err2 par(R_meas_t3) par('1.0 / lv1(r+25) ')
* * Output section *
.dc data=RES_TEMP
.print 'r1_diff'=par('1.0/lv1(r-25)')
+      'r2_diff'=par('1.0/lv1(r0) ')
+      'r3_diff'=par('1.0/lv1(r+25)')
.data RES_TEMP R_meas_t1 R_meas_t2 R_meas_t3
950 1000 1010
950 1000 1010
.enddata
.end
```

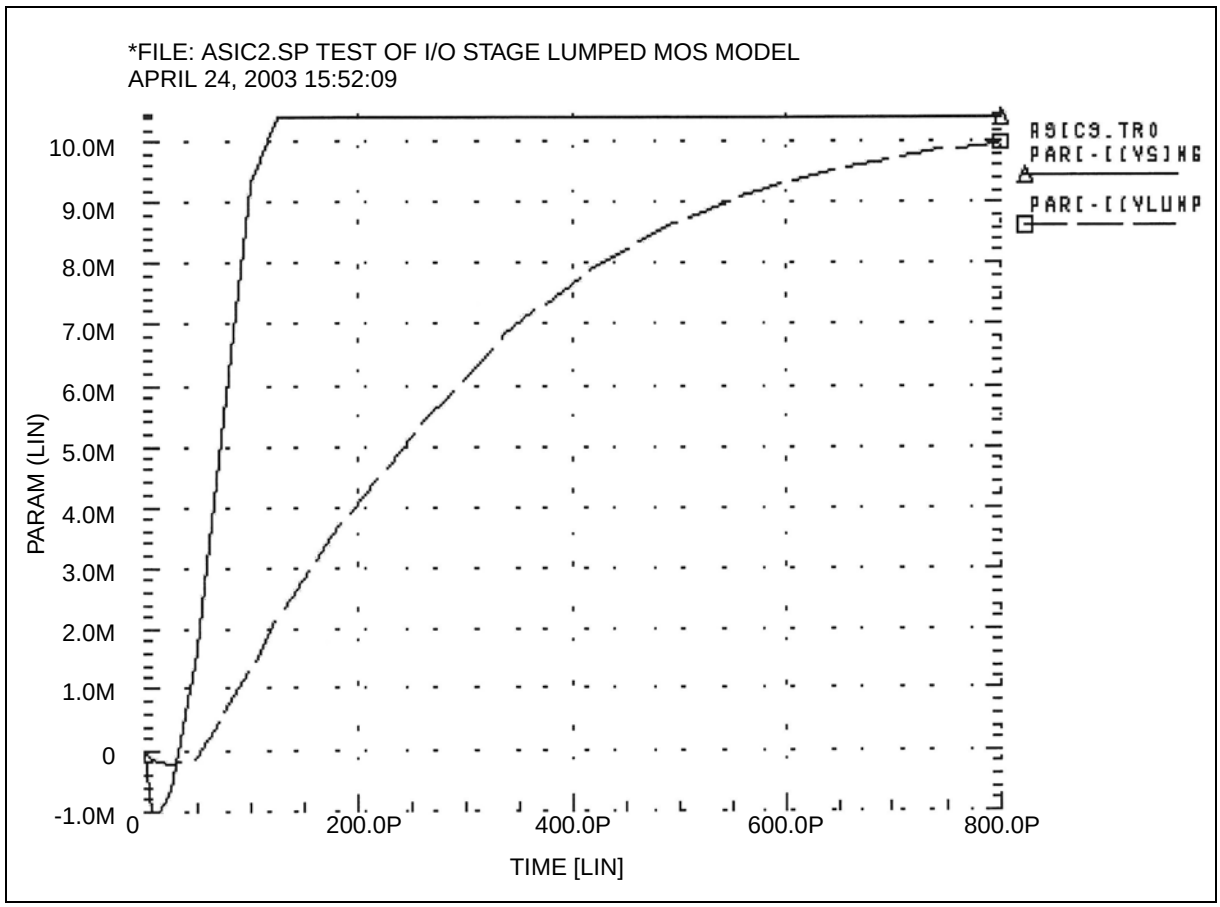
Simulating Electrical Measurements

In this example, HSPICE or HSPICE RF simulates electrical measurements, which return device characteristics for data sheets. The demonstration file for this example is $*installdir*/demo/hspice/ddl/t2n2222.sp (or $*installdir*/demo/hspicext/ddl/t2n2222.sp for HSPICE RF). This example automatically includes DDL models by reference, using either the DDLPATH environment variable, or the `.OPTION SEARCH=path` statement. It also combines an AC circuit and measurement, with a transient circuit and measurement.

The AC circuit measures the maximum Hfe, which is the small-signal common emitter gain. HSPICE or HSPICE RF uses the `.MEASURE WHEN` statement to calculate the unity gain frequency, and the phase at the specified frequency. In the *Transient Measurements* section of the input file, a segmented transient statement speeds-up simulation, and compresses the output graph. Measurements include:

- TURN ON from 90% of input rising, to 90% of output falling.
- OUTPUT FALL from 90% to 10% of output falling.
- TURN OFF from 10% of input falling, to 10% of output rising.
- OUTPUT RISE from 10% to 90% of output rising.

Figure 90 T2N2222 Optimization



T2N2222 Optimization Example Input File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/ddl/t2n2222.sp`

Modeling Wide-Channel MOS Transistors

If you select an appropriate model for I/O cell transistors, simulation accuracy improves. For wide-channel devices, model the transistor as a *group* of transistors, connected in parallel, with appropriate RC delay networks. If you model the device as only *one* transistor, the polysilicon gate introduces delay.

When you scale to higher-speed technologies, the area of the polysilicon gate decreases, reducing the gate capacitance. However, if you scale the gate oxide thickness, the capacitance per unit area increases, which also increases the RC product.

Example

The following example illustrates how scaling affects the delay. For example, for a device with:

- Channel width=100 microns.
- Channel length=5 microns.
- Gate oxide thickness=800 Angstroms.

The resulting RC product for the polysilicon gate is:

$$\text{Rpoly} = \frac{W}{L} \cdot 40 \qquad \text{poly} = \frac{Esio \cdot nsi}{tox} \cdot L \cdot W$$

$$\text{Rpoly} = \frac{100}{5} \cdot 40 = 800, \quad \text{Co} = \frac{3.9 \cdot 8.86}{800} \cdot 100 \cdot 5 = 215fF \quad \text{RC=138 ps}$$

For a transistor with:

- Channel width=100 microns.
- Channel length=1.2 microns.
- Gate oxide thickness=250 Angstroms.

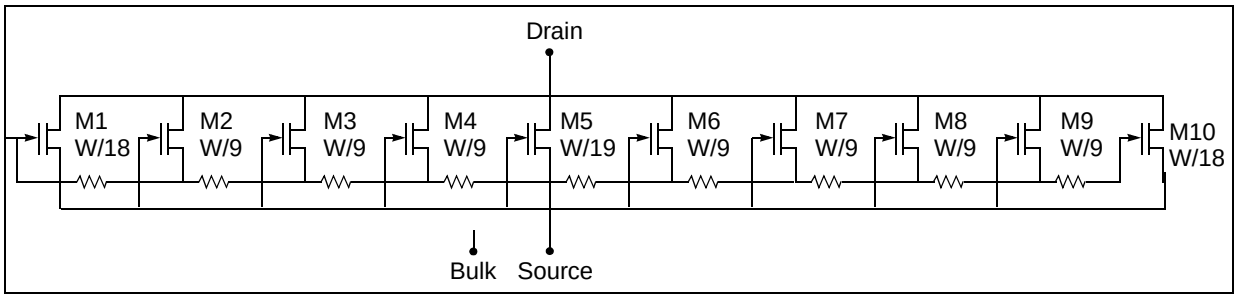
The resulting RC product for the polysilicon gate is:

$$\text{Rpoly} = \frac{channel\ \ width}{channel\ \ length} \cdot 40$$

$$\text{Co} = \frac{3.9 \cdot 8.86}{Tox} \cdot channel\ \ width \cdot channel\ \ length \quad \text{RC=546 ps}$$

You can use a nine-stage ladder model to model the RC delay in CMOS devices.

Figure 91 Nine-stage Ladder Model



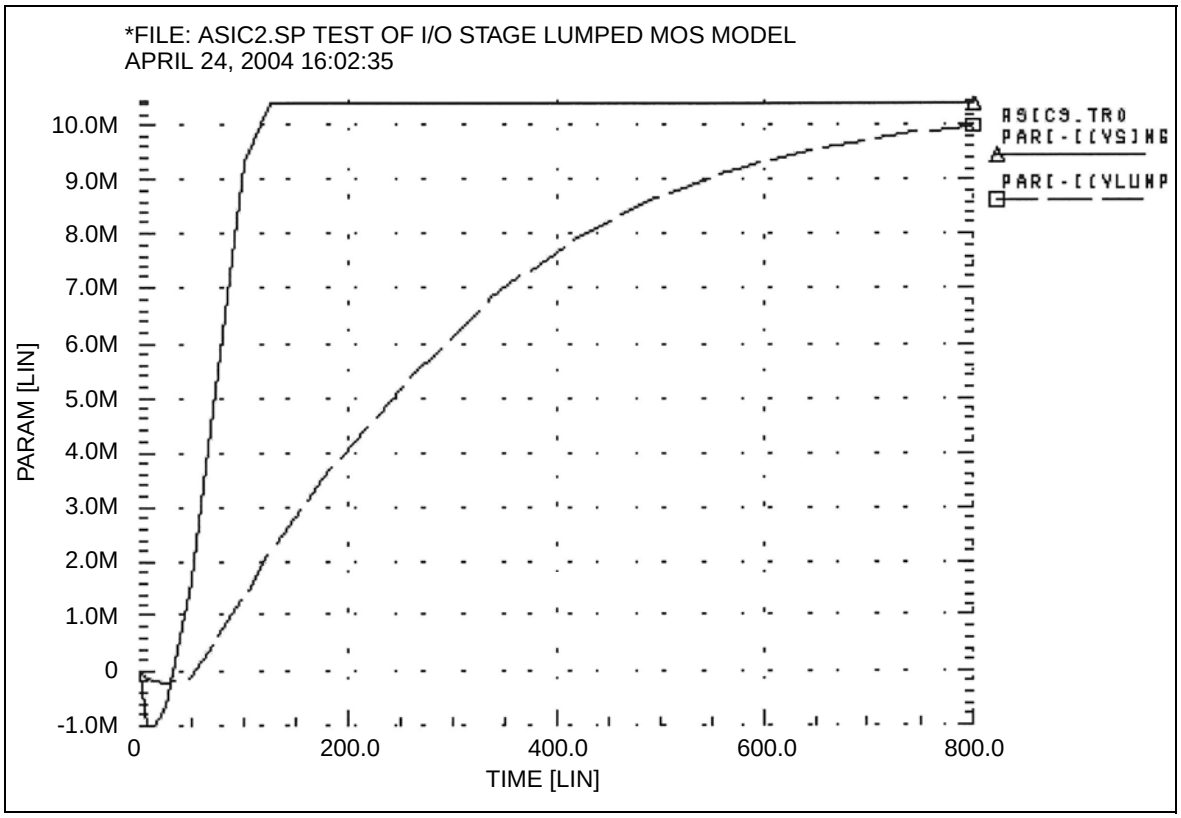
In this example, the nine-stage ladder model is in data file $*installdir*/demo/hspice/apps /asic3.sp. To optimize this model, HSPICE uses measured data from a wide channel transistor as the target data\. Optimization produces a nine-stage ladder model, which matches the timing characteristics of the physical data (HSPICE RF does not support optimization). HSPICE compares the simulation results for the nine-stage ladder model, and the one-stage model by using the nine-stage ladder model as the reference. The one-stage model results are about 10% faster than actual physical data indicates.

Example

You can find the sample Nine-Stage Ladder model netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/asic3.sp
```

Figure 92 Asic3 Single vs. Lumped Model



Demonstration Input Files

File Name	Description
Algebraic Output Variable Examples $installdir/demo/hspice/alge	
alg.sp	demonstrates algebraic parameters
alg_fil.sp	magnitude response of the behavioral filter model
alg_vco.sp	voltage-controlled oscillator
alg_vf.sp	voltage-to-frequency converter behavioral model
xalg1.sp	QA of parameters

File Name	Description
xalg2.sp	QA of parameters
Applications of General Interest $installdir/demo/hspice/apps	
alm124.sp	AC, noise, and transient op-amp analysis
alter2.sp	`.ALTER` examples
ampg.sp	pole/zero analysis of a G source amplifier
asic1.sp	ground bounce for I/O CMOS driver
asic3.sp	ten-stage lumped MOS model
bjt2bit.sp	BJT two-bit adder
bjt4bit.sp	four-bit all NAND gate, binary adder
bjtdiff.sp	BJT diff amp with every analysis type
bjtschmt.sp	bipolar Schmidt trigger
bjtsense.sp	bipolar sense amplifier
cellchar.sp	characteristics of ASIC inverter cell
crystal.sp	crystal oscillator circuit
gaasamp.sp	simple GaAsFET amplifier
grouptim.sp	group time-delay example
inv.sp	sweep MOSFET -3 sigma to +3 sigma use .MEASURE output
mcdiff.sp	CMOS differential amplifier
mondc_a.sp	Monte Carlo of MOS diffusion and photolithographic effects (HSPICE only)
mondc_b.sp	Monte Carlo DC analysis (HSPICE only)
mont1.sp	Monte Carlo Gaussian, uniform, and limit function (HSPICE only)

File Name	Description
mos2bit.sp	two-bit MOS adder
opampdcm.sp	DCmatch analysis
pll.sp	phase-locked loop
sclopass.sp	switched-capacitor low-pass filter
worst.sp	worst case skew models by using .ALTER
xbjt2bit.sp	BJT NAND gate two-bit binary adder
Behavioral Applications $installdir/demo/hspice/behave	
acl.sp	acl gate
amp_mod.sp	amplitude modulator with pulse waveform carrier
behave.sp	AND/NAND gates by using G, E Elements
calg2.sp	voltage variable capacitance
det_dff.sp	double edge-triggered flip-flop
diff.sp	differentiator circuit
diode.sp	behavioral diode by using a PWL VCCS
dlatch.sp	CMOS D-latch by using behaviorals
galg1.sp	sampling a sine wave
idealop.sp	ninth-order low-pass filter
integ.sp	integrator circuit
invb_op.sp	optimizes the CMOS macromodel inverter
ivx.sp	characteristics of the PMOS and NMOS as a switch
op_amp.sp	op-amp from Chua and Lin
pd.sp	phase detector modeled as switches

File Name	Description
pdb.sp	phase detector by using behavioral NAND gates
pwl10.sp	operational amplifier used as a voltage follower
pwl2.sp	PPW-VCCS with a gain of 1 amp/volt
pwl4.sp	eight-input NAND gate
pwl7.sp	modeling inverter by using a PWL VCVS
pwl8.sp	smoothing the triangle waveform by using the PWL CCCS
ring5bm.sp	five-stage ring oscillator – macromodel CMOS inverter
ringb.sp	ring oscillator by using behavioral model
sampling.sp	sampling a sine wave
scr.sp	silicon-controlled rectifier, modeled using the PWL CCVS
swcap5.sp	fifth-order elliptic switched capacitor filter
switch.sp	test for PWL switch element
swrc.sp	switched capacitor RC circuit
triode.sp	triode model family of curves by using behavioral elements
triodex.sp	triode model family of curves by using behavioral elements
tunnel.sp	modeling tunnel diode characteristic by using PWL VCCS
vcob.sp	voltage-controlled oscillator by using PWL functions
Benchmarks $installdir/demo/hspice/bench	
bigmos1.sp	large MOS simulation
demo.sp	quick demo file to test installation
m2bit.sp	72-transistor two-bit adder – typical cell simulation
m2bitf.sp	fast simulation example

File Name	Description
m2bitsw.sp	Fast simulation example. Same as m2bitf.sp, but uses behavioral elements
senseamp.sp	bipolar analog test case
Timing Analysis $installdir/demo/hspice/bisect	
fig3a.sp	DFF bisection search for setup time
fig3b.sp	DFF early, optimum, and late setup times
inv_a.sp	inverter bisection (pass-fail)
BJT and Diode Devices $installdir/demo/hspice/bjt	
bjtbeta.sp	plot BJT beta
bjtft.sp	plot BJT FT by using s-parameters
bjtgm.sp	plot BJT Gm, Gpi
dpntun.sp	junction tunnel diode
snaphsp.sp	convert SNAP to HSPICE
tun.sp	tunnel oxide diode
Cell Characterization $installdir/demo/hspice/cchar	
dff.sp	DFF bisection search for setup time
inv3.sp	characteristics of an inverter
inva.sp	characteristics of an inverter
invb.sp	characteristics of an inverter
load1.sp	inverter sweep, delay versus fanout
setupbsc.sp	setup characteristics
setupold.sp	setup characteristics

File Name	Description
setuppas.sp	setup characteristics
sigma.sp	sweep MOSFET -3 sigma to +3 sigma by using measure output
tdgtl.a2d	Viewsim A2D HSPICE or HSPICE RF input file
tdgtl.d2a	Viewsim D2A HSPICE or HSPICE RF input file
tdgtl.sp	two-bit adder by using D2A Elements
Circuit Optimization $installdir/demo/hspice/ciropt	
ampgain.sp	set unity gain frequency of a BJT diff pair
ampopt.sp	optimize area, power, speed of a MOS amp
asic2.sp	optimize speed, power of a CMOS output buffer
asic6.sp	find best width of a CMOS input buffer
delayopt.sp	optimize group delay of an LCR circuit
lpopt.sp	match lossy filter to ideal filter
opttemp.sp	find first and second temperature coefficients of a resistor
rcopt.sp	optimize speed or power for an RC circuit
DDL $installdir/demo/hspice/ddl	
ad8bit.sp	eight-bit A/D flash converter
alf155.sp	characteristics of National JFET op-amp
alf156.sp	characteristics of National JFET op-amp
alf157.sp	characteristics of National JFET op-amp
alf255.sp	characteristics of National JFET op-amp
alf347.sp	characteristics of National JFET op-amp
alf351.sp	characteristics of National wide-bandwidth, JFET input, op-amp

File Name	Description
alf353.sp	characteristics of National wide-bandwidth, dual JFET input, op-amp
alf355.sp	characteristics of Motorola JFET, op-amp
alf356.sp	characteristics of Motorola JFET, op-amp
alf357.sp	characteristics of Motorola JFET, op-amp
alf3741.sp	
alm101a.sp	
alm107.sp	characteristics of National op-amp
alm108.sp	characteristics of National op-amp
alm108a.sp	characteristics of National op-amp
alm118.sp	characteristics of National op-amp
alm124.sp	characteristics of National low-power, quad op-amp
alm124a.sp	characteristics of National low-power, quad op-amp
alm158.sp	characteristics of National op-amp
alm158a.sp	characteristics of National op-amp
alm201.sp	characteristics of LM201 op-amp
alm201a.sp	characteristics of LM201 op-amp
alm207.sp	characteristics of National op-amp
alm208.sp	characteristics of National op-amp
alm208a.sp	characteristics of National op-amp
alm224.sp	characteristics of National op-amp
alm258.sp	characteristics of National op-amp
alm258a.sp	characteristics of National op-amp

File Name	Description
alm301a.sp	characteristics of National op-amp
alm307.sp	characteristics of National op-amp
alm308.sp	characteristics of National op-amp
alm308a.sp	characteristics of National op-amp
alm318.sp	characteristics of National op-amp
alm324.sp	characteristics of National op-amp
alm358.sp	characteristics of National op-amp
alm358a.sp	characteristics of National op-amp
alm725.sp	characteristics of National op-amp
alm741.sp	characteristics of National op-amp
alm747.sp	characteristics of National op-amp
alm747c.sp	characteristics of National op-amp
alm1458.sp	characteristics of National dual op-amp
alm1558.sp	characteristics of National dual op-amp
alm2902.sp	characteristics of National op-amp
alm2904.sp	characteristics of National op-amp
amc1458.sp	characteristics of Motorola internally-compensated, high-performance op-amp
amc1536.sp	characteristics of Motorola internally-compensated, high-voltage op-amp
amc1741.sp	characteristics of Motorola internally-compensated, high-performance op-amp
amc1747.sp	characteristics of Motorola internally-compensated, high-performance op-amp

File Name	Description
ane5534.sp	characteristics of TI low-noise, high-speed op-amp
anjm4558.sp	characteristics of TI dual op-amp
anjm4559.sp	characteristics of TI dual op-amp
anjm4560.sp	characteristics of TI dual op-amp
aop04.sp	characteristics of PMI op-amp
aop07.sp	characteristics of PMI ultra-low offset voltage, op-amp
aop14.sp	characteristics of PMI op-amp
aop15b.sp	characteristics of PMI precision JFET input, op-amp
aop16b.sp	characteristics of PMI precision JFET input, op-amp
at094cns.sp	characteristics of TI op-amp
atl071c.sp	characteristics of TI low-noise, op-amp
atl072c.sp	characteristics of TI low-noise, op-amp
atl074c.sp	characteristics of TI low-noise, op-amp
atl081c.sp	characteristics of TI JFET op-amp
atl082c.sp	characteristics of TI JFET op-amp
atl084c.sp	characteristics of TI JFET op-amp
atl092cp.sp	characteristics of TI op-amp
atl094cn.sp	characteristics of TI op-amp
aupc358.sp	characteristics of NEC general, dual op-amp
aupc1251.sp	characteristics of NEC general, dual op-amp
j2n3330.sp	characteristics of JFET 2n3330 I-V
mirf340.sp	characteristics of IRF340 I-V

File Name	Description
t2n2222.sp	characteristics of BJT 2n2222
Device Optimization (HSPICE only) $installdir/demo/hspice/devopt	
beta.sp	LEVEL=2 beta optimization
bjtopt.sp	s-parameter optimization of a 2n6604 BJT
bjtopt1.sp	2n2222 DC optimization
bjtopt2.sp	2n2222 Hfe optimization
d.sp	diode, multiple temperatures
dcopt1.sp	1n3019 diode, I-V and C-V optimization
gaas.sp	JFET optimization
jopt.sp	300u/1u GaAs FET, DC optimization
jopt2.sp	JFET optimization
joptac.sp	300u/1u GaAs FET, 40 MHz–20 GHz, s-parameter optimization
l3.sp	MOS LEVEL 3 optimization
l3a.sp	MOS LEVEL 3 optimization
l28.sp	LEVEL=28 optimization
ml2opt.sp	MOS LEVEL=2 I-V optimization
ml3opt.sp	MOS LEVEL=3 I-V optimization
ml6opt.sp	MOS LEVEL=6 I-V optimization
ml13opt.sp	MOS LEVEL=13 I-V optimization
opt_bjt.sp	2n3947 forward and reverse Gummel optimization
Fourier Analysis (HSPICE only) $installdir/demo/hspice/fft	
am.sp	FFT analysis, AM source

File Name	Description
bart.sp	FFT analysis, Bartlett window
black.sp	FFT analysis, Blackman window
dist.sp	FFT analysis, second harmonic distortion
exam1.sp	FFT analysis, AM source
exam3.sp	FFT analysis, high-frequency signal detection test
exam4.sp	FFT analysis, small-signal harmonic distortion test
exp.sp	FFT analysis, exponential source
fft.sp	FFT analysis, transient, sweeping a resistor
fft1.sp	FFT analysis, transient
fft2.sp	FFT analysis on the product of three waveforms
fft3.sp	FFT analysis, transient, sweeping frequency
fft4.sp	FFT analysis, transient, Monte Carlo Gaussian distribution
fft5.sp	FFT analysis, data-driven transient analysis
fft6.sp	FFT analysis, sinusoidal source
gauss.sp	FFT analysis, Gaussian window
hamm.sp	FFT analysis, Hamming window
hann.sp	FFT analysis, Hanning window
harris.sp	FFT analysis, Blackman-Harris window
intermod.sp	FFT analysis, intermodulation distortion
kaiser.sp	FFT analysis, Kaiser window
mod.sp	FFT analysis, modulated pulse
pulse.sp	FFT analysis, pulse source

File Name	Description
pwl.sp	FFT analysis, piecewise linear source
rect.sp	FFT analysis, rectangular window
rectan.sp	FFT analysis, rectangular window
sffm.sp	FFT analysis, single-frequency FM source
sine.sp	FFT analysis, sinusoidal source
swcap5.sp	FFT analysis, fifth-order elliptic, switched-capacitor filter
tri.sp	FFT analysis, rectangular window
win.sp	FFT analysis, window test
window.sp	FFT analysis, window test
winreal.sp	FFT analysis, window test
Filters $installdir/demo/hspice/filters	
fbp_1.sp	bandpass LCR filter, measurement
fbp_2.sp	bandpass LCR filter, pole/zero
fbpnet.sp	bandpass LCR filter, s-parameters
fbprlc.sp	LCR AC analysis for resonance
fhp4th.sp	high-pass LCR, fourth-order Butterworth filter
fkerwin.sp	pole/zero analysis of Kerwin's circuit
flp5th.sp	low-pass, fifth-order filter
flp9th.sp	low-pass, ninth-order FNDR, with ideal op-amps
micro1.sp	test of microstrip
micro2.sp	test of microstrip
tcoax.sp	test of RG58/AU coax

File Name	Description
trans1m.sp	FR-4, printed-circuit, lumped transmission line
Magnetics $installdir/demo/hspice/mag	
aircore.sp	air-core transformer circuit
bhloop.sp	b-h loop, non-linear, magnetic-core transformer
mag2.sp	three primary, two secondary, magnetic-core transformer
magcore.sp	magnetic-core transformer circuit
royerosc.sp	Royer magnetic-core oscillator
MOSFET Devices $installdir/demo/hspice/mos	
bsim3.sp	LEVEL=47 BSIM3 model
cap13.sp	plot MOS capacitances, LEVEL=13 model
cap_b.sp	capacitances for LEVEL=13 model
cap_m.sp	capacitance for LEVEL=13 model
capop0.sp	plot MOS capacitances, LEVEL=2
capop1.sp	plot MOS capacitances, LEVEL=2
capop2.sp	plot MOS capacitances, LEVEL=2
capop4.sp	plot MOS capacitances, LEVEL=6
chrgpump.sp	charge-conservation test, LEVEL=3
iiplot.sp	plot of impact ionization current
ml6fex.sp	plot temperature effects, LEVEL=6
ml13fex.sp	plot temperature effects, LEVEL=13
ml13ft.sp	s-parameters for LEVEL=13
ml13iv.sp	plot I-V for LEVEL=13

File Name	Description
ml27iv.sp	plot I-V for LEVEL=27 SOSFET
mosiv.sp	plot I-V for files that you include
mosivcv.sp	plot I-V and C-V for LEVEL=3
qpulse.sp	charge-conservation test, LEVEL=6
qswitch.sp	charge-conservation test, LEVEL=6
selector.sp	automatic model selector for width and length
tgam2.sp	LEVEL=6, gamma model
tmos34.sp	MOS LEVEL=34 EPFL, test DC
Radiation Effects $installdir/demo/hspice/rad	
brad1.sp	example of bipolar radiation effects
brad2.sp	example of bipolar radiation effects
brad3.sp	example of bipolar radiation effects
brad4.sp	example of bipolar radiation effects
brad5.sp	example of bipolar radiation effects
brad6.sp	example of bipolar radiation effects
drad1.sp	example of diode radiation effects
drad2.sp	example of diode radiation effects
drad4.sp	example of diode radiation effects
drad5.sp	example of diode radiation effects
drad6.sp	example of diode radiation effects
dradarb2.sp	example of diode radiation effects
jex1.sp	example of JFET radiation effects

File Name	Description
jex2.sp	example of JFET radiation effects
jprad1.sp	example of JFET radiation effects
jprad2.sp	example of JFET radiation effects
jprad4.sp	example of JFET radiation effects
jrad1.sp	example of JFET radiation effects
jrad2.sp	example of JFET radiation effects
jrad3.sp	example of JFET radiation effects
jrad4.sp	example of JFET radiation effects
jrad5.sp	example of JFET radiation effects
jrad6.sp	example of JFET radiation effects
mrad1.sp	example of MOSFET radiation effects
mrad2.sp	example of MOSFET radiation effects
mrad3.sp	example of MOSFET radiation effects
mrad3p.sp	example of MOSFET radiation effects
mrad3px.sp	example of MOSFET radiation effects
rad1.sp	example of total MOSFET dose
rad2.sp	diode photo-current test circuit
rad3.sp	diode photo-current test circuit, RLEV=3
rad4.sp	diode photo-current test circuit
rad5.sp	BJT photo-current test circuit, with an NPN transistor
rad6.sp	BJT secondary photo-current effect, which varies with R1
rad7.sp	BJT RLEV=6 example (semi-empirical model)

rad8.sp	JFET RLEV=1 example with Wirth-Rogers square pulse
rad9.sp	JFET stepwise-increasing radiation source
rad10.sp	GaAs RLEV=5 example (semi-empirical model)
rad11.sp	NMOS E-model, LEVEL=8 with Wirth-Rogers square pulse
rad12.sp	NMOS 0.5x resistive vl(Ru)htage-d-eivGuirse

File Name	Description
fr4x.sp	FR4 microstrip test
hd.sp	ground bounce for I/O CMOS driver
rcsnubts.sp	ground bounce for I/O CMOS driver, at snubber output
rcsnubtt.sp	ground bounce for I/O CMOS driver
strip1.sp	two microstrips, in series (8 mil and 16 mil wide)
strip2.sp	two microstrips, coupled together
t14p.sp	1400 mil by 140 mil, 50-ohm tline, on FR-4, 50 MHz to 10.05 GHz
t14xx.sp	1400 mil by 140 mil, 50-ohm tline, on FR-4 optimization
t1400.sp	1400 mil by 140 mil, 50-ohm tline, on FR-4 optimization
tcoax.sp	RG58/AU coax, with 50-ohm termination
tfr4.sp	microstrip test
tfr4o.sp	microstrip test
tl.sp	series source, coupled and shunt-terminated transmission lines
transmis.sp	algebraics, and lumped transmission lines
twin2.sp	twin-lead model
xfr4.sp	microstrip test sub-circuit, expanded
xfr4a.sp	microstrip test sub-circuit, expanded, larger ground-resistance
xfr4b.sp	microstrip test
xulump.sp	test 5-, 20-, and 100-lump, U models
Verilog-A $installdir/demo/hspice/veriloga	
resistor.sp	a very simple Verilog-A resistor model
sinev.sp	simple voltage source

File Name	Description
deadband.sp	deadband amplifier
pll.sp	behavioral model of PLL
psfet.sp	Parker Skellern FET model
colpitts.va	Colpitts BJT oscillator
ecl.sp	ECL inverter
opamp.sp	opamp
sample_hold.sp	sample and hold
biterrorrate.sp	bit error rate counter
dac.sp	DAC and ADC

A

Statistical Analysis

Describes the features available in HSPICE for statistical analysis before the Y-2006.03 release.

Overview

Described in this appendix are the features available in HSPICE for statistical analysis before the Y-2006.03 release. These features are still supported; however, the new features described in Chapter 13, Simulating Variability, Chapter 14, Variation Block, and Chapter 15, Monte Carlo Analysis represent a significant enhancement over prior approaches.

The previously available documentation on statistical analysis has been reviewed and enhanced for the benefit of those users who are not yet ready to migrate to the new approach. In particular, the last section was added to explain the setup for simulating the effects of global and local variations on silicon with Monte Carlo.

The following subjects are described in this appendix:

- Application of Statistical Analysis
- Analytical Model Types
- Simulating Circuit and Model Temperatures
- Worst Case Analysis
- Monte Carlo Analysis
- Worst Case and Monte Carlo Sweep Example
- Simulating the Effects of Global and Local Variations with Monte Carlo

Application of Statistical Analysis

When you design an electrical circuit, it must meet tolerances for the specific manufacturing process. The electrical yield is the number of parts that meet the electrical test specifications. Overall process efficiency requires maximum yield. To analyze and optimize the yield, Synopsys HSPICE supports statistical techniques and observes the effects of variations in element and model parameters.

Analytical Model Types

To model parametric and statistical variation in circuit behavior, use:

- `.PARAM` statement to investigate the performance of a circuit as you change circuit parameters. For details about the `.PARAM` statement, see the .PARAM statement in the *HSPICE Command Reference*.
- Temperature variation analysis to vary the circuit and component temperatures, and compare the circuit responses. You can study the temperature-dependent effects of the circuit, in detail.
- Monte Carlo analysis when you know the statistical standard deviations of component values to center a design. This provides maximum process yield, and determines component tolerances.
- Worst-case corner analysis when you know the component value limit to automate quality assurance for:
  - basic circuit function
  - process extremes
  - quick estimation of speed and power trade-offs
  - best-case and worst-case model selection
  - parameter corners
  - library files
- Data-driven analysis for cell characterization, response surface, or Taguchi analysis. See “Performing Digital Cell Characterization” in the *HSPICE Applications Manual*. Automates characterization of cells and calculates the coefficient of polynomial delay for timing simulation. You can simultaneously vary any number of parameters and perform an unlimited number of

analyses. This analysis uses an ASCII file format so HSPICE can automatically generate parameter values. This analysis can replace hundreds or thousands of HSPICE simulation runs.

Use yield analyses to modify:

- DC operating points
- DC sweeps
- AC sweeps
- Transient analysis.

CosmosScope can generate scatter plots from the operating point analysis or a family of curve plots for DC, AC, and transient analysis.

Use `.MEASURE` statements to save results for delay times, power, or any other characteristic extracted in a `.MEASURE` statement. HSPICE generates a table of results in an .mt# file in ASCII format. You can analyze the numbers directly or read this file into CosmosScope to view the distributions. Also, if you use `.MEASURE` statements in a Monte Carlo or data-driven analysis, then the HSPICE output file includes the following statistical results in the listing:

Mean $\frac{x_1 + x_2 + \ldots + x_n}{N}$

Variance $\frac{(x_1 - Mean)^2 + \ldots (x_n - Mean)^2}{N - 1}$

Sigma $\sqrt{Variance}$

Average Deviation $\frac{|x_1 - Mean| + \ldots + |x_n - Mean|}{N - 1}$

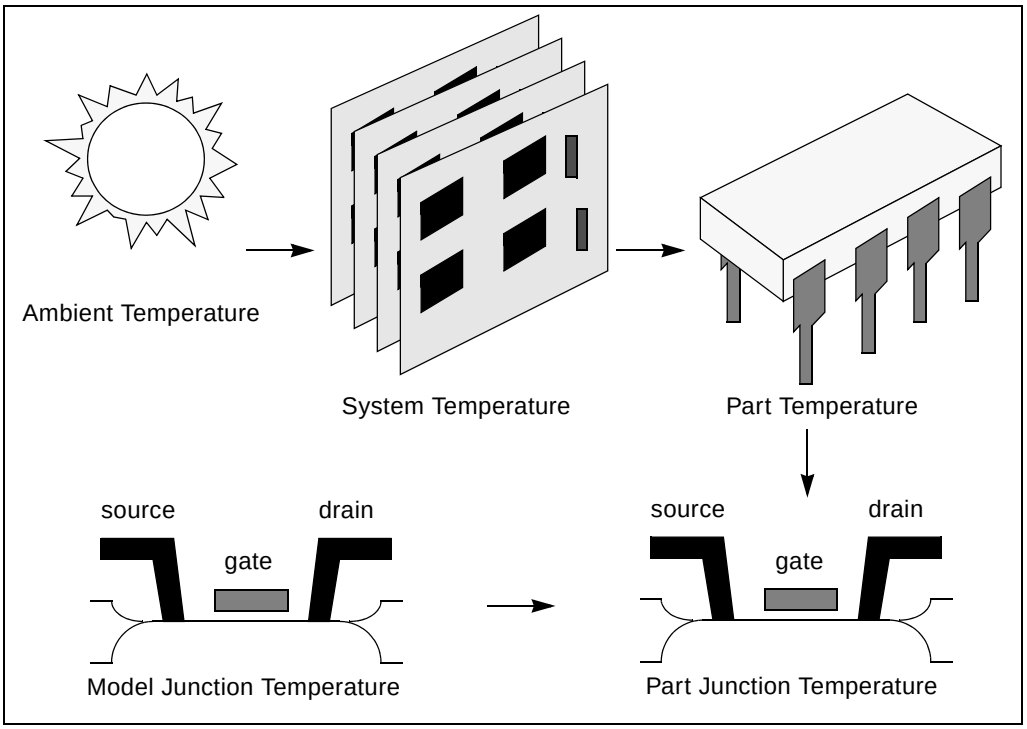
Simulating Circuit and Model Temperatures

Temperature affects *all* electrical circuits. Figure 93 shows the key temperature parameters associated with circuit simulation:

- Model reference temperature – you can model different models at different temperatures. Each model has a `TREF` (temperature reference) parameter.
- Element junction temperature – each resistor, transistor, or other element generates heat so an element is hotter than the ambient temperature.

- Part temperature – at the system level each part has its own temperature.
- System temperature – a collection of parts form a system, which has a local temperature.
- Ambient temperature – the ambient temperature is the air temperature of the system.

Figure 93 Part Junction Temperature Sets System Performance



HSPICE or HSPICE RF calculates temperatures as differences from the ambient temperature:

$$Tambient + \Delta system + \Delta part + \Delta junction = Tjunction$$

$$Ids = f(Tjunction, Tmodel)$$

Every element includes a DTEMP keyword, which defines the difference between junction and ambient temperature.

Example

The following example uses DTEMP in a MOSFET element statement:

```
M1 drain gate source bulk Model_name W=10u L=1u DTEMP=+20
```

Temperature Analysis

You can specify three temperatures:

- Model reference temperature specified in a .MODEL statement. The temperature parameter is usually TREF, but can be TEMP or TNOM in some models. This parameter specifies the temperature, in °C, at which HSPICE or HSPICE RF measures and extracts the model parameters. Set the value of TNOM in an .OPTION statement. Its default value is 25°C.
- Circuit temperature that you specify using a .TEMP statement or the TEMP parameter. This is the temperature, in °C, at which HSPICE or HSPICE RF simulates all elements. To modify the temperature for a particular element, use the DTEMP parameter. The default circuit temperature is the value of TNOM.
- Individual element temperature, which is the circuit temperature, plus an optional amount that you specify in the DTEMP parameter.

To specify the temperature of a circuit in a simulation run, use either the .TEMP statement, or the TEMP parameter in the .DC, .AC, or .TRAN statements. HSPICE or HSPICE RF compares the circuit simulation temperature that one of these statements sets against the reference temperature that the TNOM option sets. TNOM defaults to 25°C, unless you use the SPICE option, which defaults to 27°C. To calculate the derating of component values and model parameters, HSPICE or HSPICE RF uses the difference between the circuit simulation temperature, and the TNOM reference temperature.

Elements and models within a circuit can operate at different temperatures. For example, a high-speed input/output buffer that switches at 50 MHz is much hotter than a low-drive NAND gate that switches at 1 MHz). To simulate this temperature difference, specify both an element temperature parameter (DTEMP), and a model reference parameter (TREF). If you specify DTEMP in an element statement, the element temperature for the simulation is:

```
element temperature=circuit temperature + DTEMP
```

Specify the DTEMP value in the element statement (resistor, capacitor, inductor, diode, BJT, JFET, or MOSFET statement), or in a subcircuit element. Assign a parameter to DTEMP, then use the .DC statement to sweep the parameter. The DTEMP value defaults to zero.

If you specify TREF in the model statement, the model reference temperature changes (TREF overrides TNOM). Derating the model parameters is based on the difference between circuit simulator temperature and TREF (instead of TNOM).

.TEMP Statement

To specify the temperature of a circuit for a HSPICE or HSPICE RF simulation, use the .TEMP statement.

Worst Case Analysis

Circuit designers often use worst-case analysis when designing and analyzing MOS and BJT IC circuits. To simulate the worst case, set all variables to their 2- or 3-sigma worst-case values. Because several independent variables rarely attain their worst-case values simultaneously, this technique tends to be overly pessimistic and can lead to over-designing the circuit. However, this analysis is useful as a fast check.

Model Skew Parameters

The Synopsys HSPICE device models include physically-measurable model parameters. The circuit simulator uses parameter variations to predict how an actual circuit responds to extremes in the manufacturing process. Physically-measurable model parameters are called *skew* parameters, because they skew from a statistical mean to obtain predicted performance variations.

Examples of skew parameters are the difference between the drawn and physical dimension of metal, postillion, or active layers, on an integrated circuit.

Generally, you specify skew parameters independently of each other, so you can use combinations of skew parameters to represent worst cases. Typical skew parameters for CMOS technology include:

- XL – polysilicon CD (critical dimension of the poly layer, representing the difference between drawn and actual size).
- XW_n, XW_p – active CD (critical dimension of the active layer, representing the difference between drawn and actual size).
- TOX – thickness of the gate oxide.

- RSH_n, RSH_p – resistivity of the active layer.
- $DELVTO_n$, $DELVTO_p$– variation in threshold voltage.

You can use these parameters in any level of MOS model, within the HSPICE device models. The DELVTO parameter shifts the threshold value. HSPICE adds this value to VTO for the Level 3 model, and adds or subtracts it from VFB0 for the BSIM model. Table 60 shows whether HSPICE adds or subtracts deviations from the average.

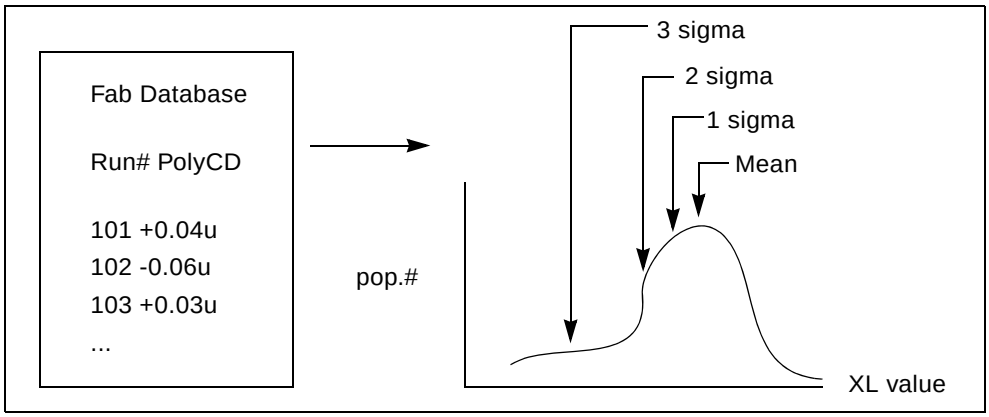
Table 60 Sigma Deviations

Type	Parameter	Slow	Fast
NMOS	XL	+	-
	RSH	+	-
	DELVTO	+	-
	TOX	+	-
	XW	-	+
PMOS	XL	+	-
	RSH	+	-
	DELVTO	-	+
	TOX	+	-
	XW	-	+

HSPICE selects skew parameters based on the available historical data that it collects either during fabrication or electrical test. For example, HSPICE collects the *XL skew* parameter for poly CD during fabrication. This parameter is usually the most important skew parameter for a MOS process.

Figure 94 is an example of data that historical records produce.

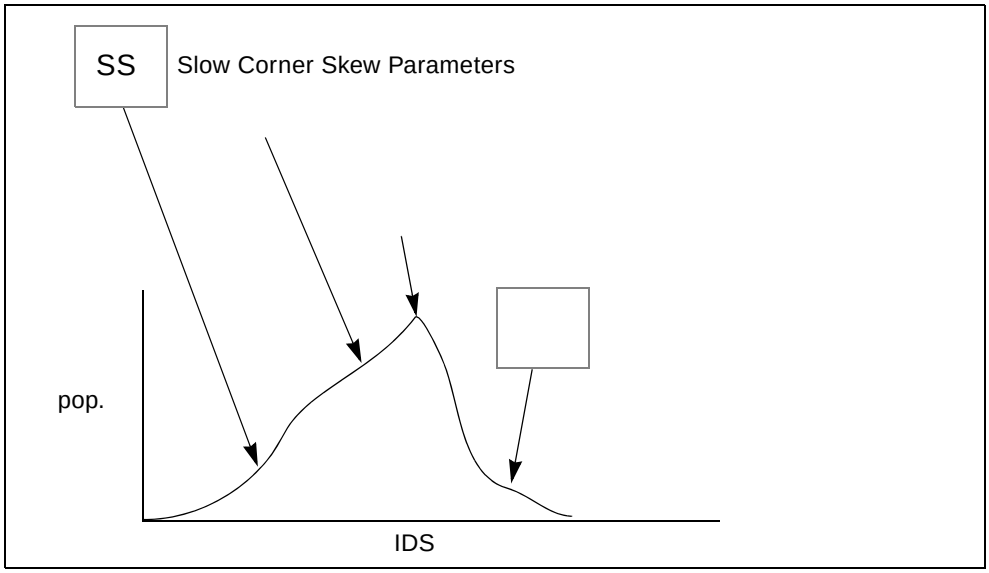
Figure 94 Historical Records for Skew Parameters in a MOS Process



Using Skew Parameters in HSPICE

Figure 95 on page 550 shows how to create a worst-case corners library file for a CMOS process model in HSPICE (HSPICE RF does not support worst-case analysis). Specify the physically-measured parameter variations so that their proper minimum and maximum values are consistent with measured current (IDS) variations. For example, HSPICE can generate a 3-sigma variation in IDS from a 2-sigma variation in physically-measured parameters.

Figure 95 Worst Case Corners Library File for a CMOS Process Model



The `.LIB` (library) statement, and the `.INCLUDE` (include file) statement, access the models and skew. The library contains parameters that

modify .MODEL statements. The following example of .LIB features both worst-case and statistical-distribution data by using model skew parameters. In statistical distribution, the median value is the default for all non-Monte Carlo analysis (HSPICE RF does not support Monte Carlo analysis).

Example

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY DATE:3/4/91
$ PROCESS: 1.0U CMOS, FAB22, STATISTICS COLLECTED 3/90-2/91
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN

.PARAM
$ polysilicon Critical Dimensions
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
$ Active layer Critical Dimensions
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
$ Gate Oxide Critical Dimensions (200 angstrom +/- 10a at 1
$ sigma)
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'

$ Threshold voltage variation
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'


.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file
.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES

.PARAM TOX=230 XL=-0.18u DELVTON=-.15V DELVTOP= 0.15V
.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file

.ENDL FF
```

The /usr/meta/lib/cmos1_mod.dat include file contains the model.

```
.MODEL NCH NMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTON . .
.MODEL PCH PMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTOP . .
```

Note:

The model keyname (left) equals the skew parameter (right). Model keys and skew parameters can use the same names.

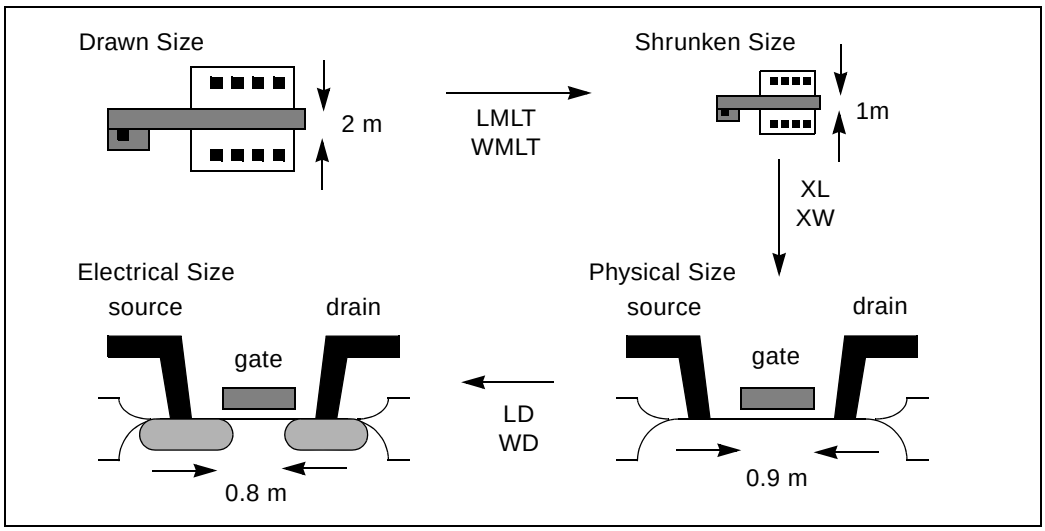
Skew File Interface to Device Models

Skew parameters are model parameters for transistor models or passive components. A typical device model set includes:

- MOSFET models for all device sizes by using an automatic model selector.
- RC wire models for polysilicon, metal1, and metal2 layers in the drawn dimension. Models include temperature coefficients and fringe capacitance.
- Single-diode and distributed-diode models for N+, P+, and well (includes temperature, leakage, and capacitance based on the drawn dimension).
- BJT models for parasitic bipolar transistors. You can also use these for any special BJTs, such as a BiCMOS for ECL BJT process (includes current and capacitance as a function of temperature).
- Metal1 and metal2 transmission line models for long metal lines.
- Models must accept elements. Sizes are based on a drawn dimension. If you draw a cell at 2μ dimension and shrink it to 1μ, the physical size is 0.9μ. The effective electrical size is 0.8μ. Account for the four dimension levels:
  - drawn size
  - shrunken size
  - physical size
  - electrical size

Most simulator models scale directly from *drawn* to *electrical* size. HSPICE MOS models support all four size levels as in Figure 96.

Figure 96 Device Model from Drawn to Electrical Size



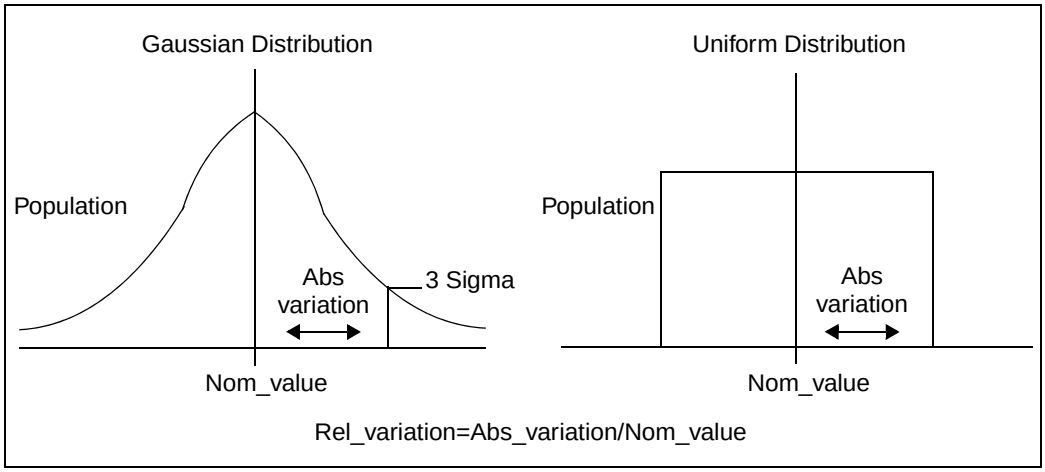
Monte Carlo Analysis

Monte Carlo analysis (HSPICE only; HSPICE RF does not support Monte Carlo analysis) uses a random number generator to create the following types of functions.

- Gaussian parameter distribution
  - Relative variation—variation is a ratio of the average.
  - Absolute variation—adds variation to the average.
  - Bimodal–multiplies distribution to statistically reduce nominal parameters.
- Uniform parameter distribution
  - Relative variation—variation is a ratio of the average.
  - Absolute variation—adds variation to the average.
  - Bimodal–multiplies distribution to statistically reduce nominal parameters.
- Random limit parameter distribution
  - Absolute variation—adds variation to the average.
  - Monte Carlo analysis randomly selects the *min* or *max* variation.

The value of the MONTE analysis keyword determines how many times to perform operating point, DC sweep, AC sweep, or transient analysis.

Figure 97 Monte Carlo Distribution



Monte Carlo Setup

To set up a Monte Carlo analysis, use the following HSPICE statements:

- .PARAM statement—sets a model or element parameter to a Gaussian, Uniform, or Limit function distribution.
- .DC, .AC, or .TRAN analysis—enables MONTE.
- .MEASURE statement—calculates the output mean, variance, sigma, and standard deviation.
- .MODEL statement—sets model parameters to a Gaussian, Uniform, or Limit function distribution.

Select the type of analysis to run, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating Point

```
.DC MONTE=<firstrun=num1>
```

-or-

```
.DC MONTE=list <(> <num1:num2> <num3> <num5:num6> <num7> <)>
```

DC Sweep

```
.DC vin 1 5 0.25 sweep MONTE=val <firstrun=num1>
```

-or-

```
.DC vin 1 5 0.25 sweep MONTE=list<(> <num1:num2> <num3>
+ <num5:num6> <num7> <)>
```

AC Sweep

```
.AC dec 10 100 1meg sweep MONTE=val <firstrun=num1>
```

-or-

```
.AC dec 10 100 1meg sweep MONTE=list<(> <num1:num2>
+ <num3> <num5:num6> <num7> <)>
```

TRAN Sweep

```
.TRAN 1n 10n sweep MONTE=val <firstrun=num1>
```

-or-

```
.TRAN 1n 10n sweep MONTE=list<(> <num1:num2> <num3>
+ <num5:num6> <num7> <)>
```

The *val* value specifies the number of Monte Carlo iterations to perform. A reasonable number is 30. The statistical significance of 30 iterations is quite high. If the circuit operates correctly for all 30 iterations, there is a 99% probability that over 80% of all possible component values operate correctly. The relative error of a quantity, determined through Monte Carlo analysis, is proportional to $val^{-1/2}$.

The *firstrun* values specify the desired number of iterations. HSPICE runs from num1 to num1+val-1. The number after *firstrun* can be a parameter. You can write only one number after *list*. The colon represents "from ... to ...". Specifying only one number makes HSPICE runs only a the one specified point.

Example 1

In this example, HSPICE runs from the 90th to 99th Monte Carlo iterations:

```
.tran 1n 10 sweep monte=10 firstrun=90
```

You can write more than one number after *list*. The colon represents "from ... to ...". Specifying only one number makes HSPICE run only at that single point.

Example 1

In this example, HSPICE begins running at the 10th iteration, then continues from the 20th to the 30th, at the 40th, and finally from the 46th to 72nd Monte Carlo iteration. The numbers after list can not be parameter.

```
.tran 1n 10n sweep monte=list(10 20:30 40 46:72)
```

Monte Carlo Output

- `.MEASURE` statements are the most convenient way to summarize the results.
- `.PRINT` statements generate tabular results, and print the values of all Monte Carlo parameters.
- `.MCBRIEF` determines the output types of the random parameters during Monte Carlo analysis to improve output performance.
- If one iteration is out of specification, you can obtain the component values from the tabular listing. A detailed resimulation of that iteration might help identify the problem.
- `.GRAPH` generates a high-resolution plot for each iteration.
- By contrast, AvanWaves superimposes all iterations as a single plot so you can analyze each iteration individually.

.PARAM Distribution Function

This section describes how to use assign a `.PARAM` parameter in Monte Carlo analysis. For a general description of the `.PARAM` statement, see the .PARAM command in the *HSPICE Command Reference*.

You can assign a `.PARAM` parameter to the keywords of elements and models, and assign a distribution function to each `.PARAM` parameter. HSPICE recalculates the distribution function each time that and element or model keyword uses a parameter. When you use this feature, Monte Carlo analysis can use a parameterized schematic netlist without additional modifications.

Syntax

```
.PARAM xx=UNIF(nominal_val, rel_variation
+ <, multiplier>)

.PARAM xx=AUNIF(nominal_val, abs_variation <,
+ multiplier>)

.PARAM xx=GAUSS(nominal_val, rel_variation, sigma <,
+ multiplier>)

.PARAM xx=AGAUSS(nominal_val, abs_variation, sigma <,
+ multiplier>)

.PARAM xx=LIMIT(nominal_val, abs_variation)
```

Argument	Description
xx	Distribution function calculates the value of this parameter.
UNIF	Uniform distribution function by using relative variation.
AUNIF	Uniform distribution function by using absolute variation.
GAUSS	Gaussian distribution function by using relative variation.
AGAUSS	Gaussian distribution function by using absolute variation
LIMIT	Random-limit distribution function by using absolute variation. Adds +/- *abs_variation* to *nominal_val* based on whether the random outcome of a -1 to 1 distribution is greater than or less than 0.
nominal_val	Nominal value in Monte Carlo analysis and default value in all other analyses.
abs_variation	AUNIF and AGAUSS vary the nominal_val by +/- *abs_variation*.
rel_variation	UNIF and GAUSS vary the *nominal_val* by +/- (*nominal_val* · *rel_variation*).
sigma	Specifies *abs_variation* or *rel_variation* at the *sigma* level. For example, if *sigma*=3, then the standard deviation is *abs_variation* divided by 3.

Argument	Description
multiplier	If you do not specify a multiplier, the default is 1. HSPICE recalculates many times and saves the largest deviation. The resulting parameter value might be greater than or less than *nominal_val*. The resulting distribution is bimodal.

Example 1

In this example, each R has an unique variation.

```
.param mc_var=agauss(0,1,3)   $ +/- 20% swing
.param val='1000*(1+mc_var)'
v_vin vin 0 dc=1 ac=.1
r1 vin 0  '1000*(1+mc_var)'
r2 vin 0  '1000*(1+mc_var)'
```

Example 2

In this example, each R has an identical variation.

```
.param mc_var=agauss(0,1,3)   $ +/- 20% swing
.param val='1+mc_var'
v_vin vin 0 dc=1 ac=.1
r1 vin 0  '1000*val'
r2 vin 0  '1000*val'
```

Example 3

In this example, local variations to an instance parameter are applied by assigning randomly-generated variations directly to each instance parameter. Each resistor r1 through r3 receives randomly different resistance values during each Monte Carlo run.

```
.param r_local=agauss(...)
r1 1 2 r=r_local
r2 3 4 r=r_local
r3 5 6 r=r_local
```

Example 4

In this example, global variations to an instance parameter are applied by assigning the variation to an intermediate parameter before assigning it to each instance parameter. Each resistor r1 through r3 receives the same random resistance value during each Monte Carlo run.

```
.param r_random=agauss(...)
.param r_global=r_random
r1 1 2 r=r_global
r2 3 4 r=r_global
r3 5 6 r=r_global
```

Monte Carlo Parameter Distribution

Each time you use a parameter, Monte Carlo calculates a new random variable.

- If you do not specify a Monte Carlo distribution, then HSPICE assumes the nominal value.
- If you specify a Monte Carlo distribution for only one analysis, HSPICE uses the nominal value for all other analyses.

You can assign a Monte Carlo distribution to all elements that share a common model. The actual element value varies according to the element distribution. If you assign a Monte Carlo distribution to a model keyword, then all elements that share the model, use the same keyword value. You can use this feature to create double element and model distributions.

For example, the MOSFET channel length varies from transistor to transistor by a small amount that corresponds to the die distribution. The die distribution is responsible for offset voltages in operational amplifiers, and for the tendency of flip-flops to settle into random states. However, all transistors on a die site vary according to the wafer or fabrication run distribution. This value is much larger than the die distribution, but affects all transistors the same way. You can specify the wafer distribution in the MOSFET model to set the speed and power dissipation characteristics.

Monte Carlo Examples

Note:

HSPICE supports Monte Carlo analysis; HSPICE RF does not.

Gaussian, Uniform, and Limit Functions

This example is based on demonstration netlist mont1.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
mont1.sp test of monte carlo gaussian, uniform, and limit functions

.option post

.dc monte=60

* setup plots

.probe aunif_1=v(au1)
.probe aunif_10=v(au10)
.probe agauss_1=v(ag1)
.probe agauss_10=v(ag10)
.probe limit=v(l1)

* uniform distribution relative variation +/- .2
.param ru_1=unif(100,.2)

iu1 u1 0 -1
ru1 u1 0 ru_1

* absolute uniform distribution absolute variation +/- 20
* single throw and 10 throw maximum
.param rau_1=aunif(100,20)
.param rau_10=aunif(100,20,10)

iau1 au1 0 -1
rau1 au1 0 rau_1

iau10 au10 0 -1
rau10 au10 0 rau_10

* gaussian distribution relative variation +/- .2 at 3 sigma
.param rg_1=gauss(100,.2,3)

ig1 g1 0 -1
rg1 g1 0 rg_1
```

```
* absolute gaussian distribution absolute variation +/- .2 at 3
sigma
* single throw and 10 throw maximum
.param rag_1=agauss(100,20,3)
.param rag_10=agauss(100,20,3,10)

iag1 ag1 0 -1
rag1 ag1 0 rag_1

iag10 ag10 0 -1
rag10 ag10 0 rag_10

* random limit distribution absolute variation +/- 20
.param rl=limit(100,20)

il1 l1 0 -1
rl1 l1 0 rl
.end
```

Figure 98 Uniform Functions

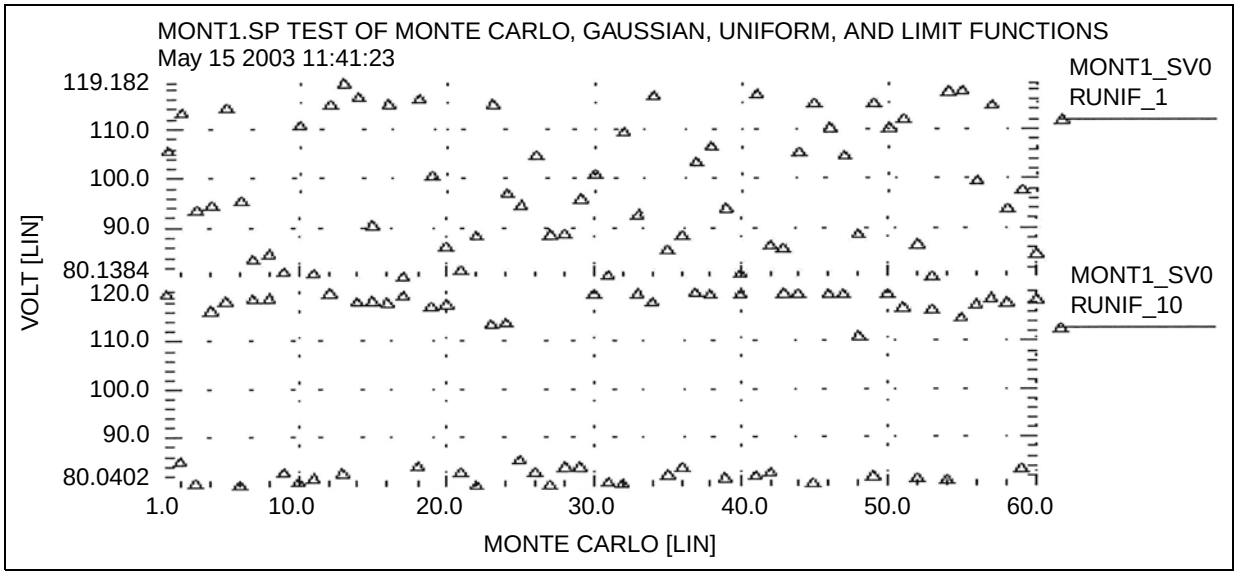


Figure 99 Gaussian Functions

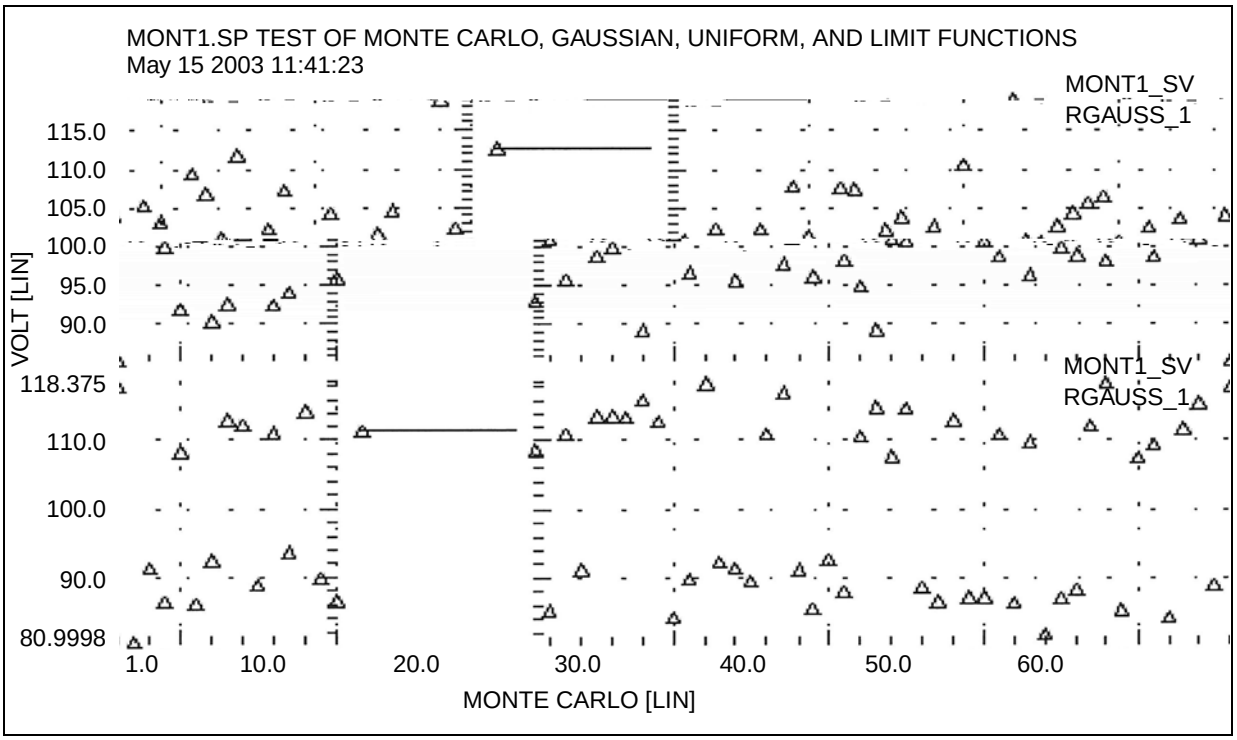
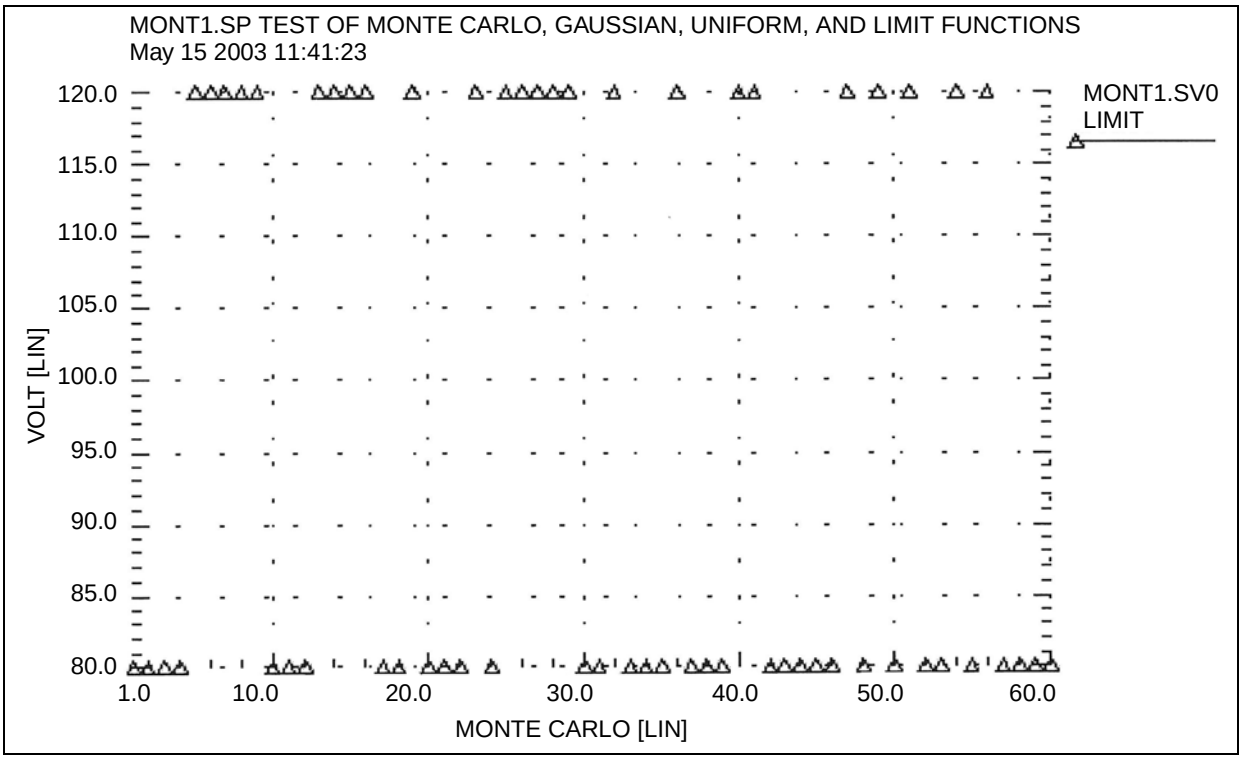


Figure 100 Limit Functions

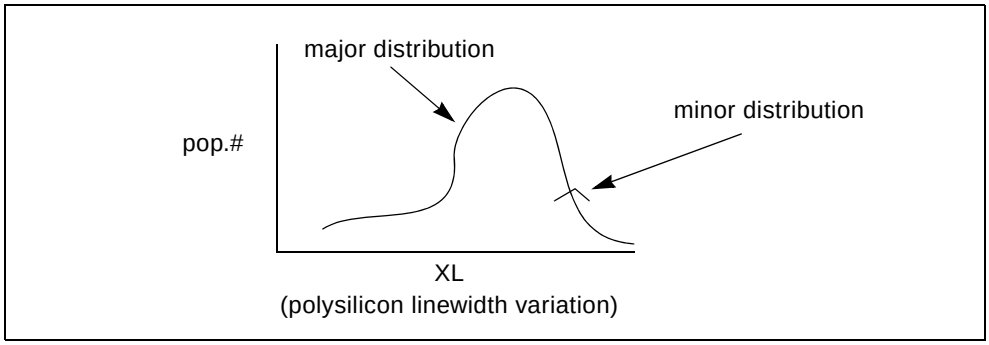


Major and Minor Distribution

In MOS IC processes, manufacturing tolerance parameters have both a major and a minor statistical distribution.

- The major distribution is the wafer-to-wafer and run-to-run variation. It determines electrical yield.
- The minor distribution is the transistor-to-transistor process variation. It is responsible for critical second-order effects, such as amplifier offset voltage and flip-flop preference.

Figure 101 Major and Minor Distribution of Manufacturing Variations



The following example is a Monte Carlo analysis of a DC sweep in HSPICE. (Note that HSPICE supports Monte Carlo analysis; HSPICE RF does not.) Monte Carlo sweeps the VDD supply voltage from 4.5 volts to 5.5 volts.

This example is based on demonstration netlist mondc_a.sp, which is available in directory $<installdir>/demo/hspice/apps:

- The M1 through M4 transistors form two inverters.
- The nominal value of the LENGTH parameter sets the channel lengths for the MOSFETs, which are set to 1u in this example.
- All transistors are on the same integrated circuit die. The LEFF parameter specifies the distribution—for example, a ±5% distribution in channel length variation at the ±3-sigma level.
- Each MOSFET has an independent random Gaussian value.

```
file: mondc_a.sp
 .options post
 .dc vdd 4.5 5.5 .1 sweep monte=30
 .probe dc i(m1)
 vdd 3 0 5v
 .param length=1u lphoto=.1u
 .param leff=gauss(length,.05,3) xphoto=gauss(lphoto,.3,3)
 .param photo=xphoto
 m1 1 2 gnd gnd nch w=10u l=leff
 m2 1 2 vdd vdd pch w=20u l=leff
 m3 2 3 gnd gnd nch w=10u l=leff
 m4 2 3 vdd vdd pch w=20u l=leff
 .model nch nmos level=2 uo=500 tox=100 gamma=.7 vto=.8 xl=photo
 .model pch pmos level=2 uo=250 tox=100 gamma=.5 vto=-.8 xl=photo
.end
```

The PHOTO parameter controls the difference between the physical gate length and the drawn gate length. Because both n-channel and p-channel transistors

use the same layer for the gates, Monte Carlo analysis sets XPHOTO distribution to the PHOTO local parameter.

XPHOTO controls PHOTO lithography for both NMOS and PMOS devices, which is consistent with the physics of manufacturing.

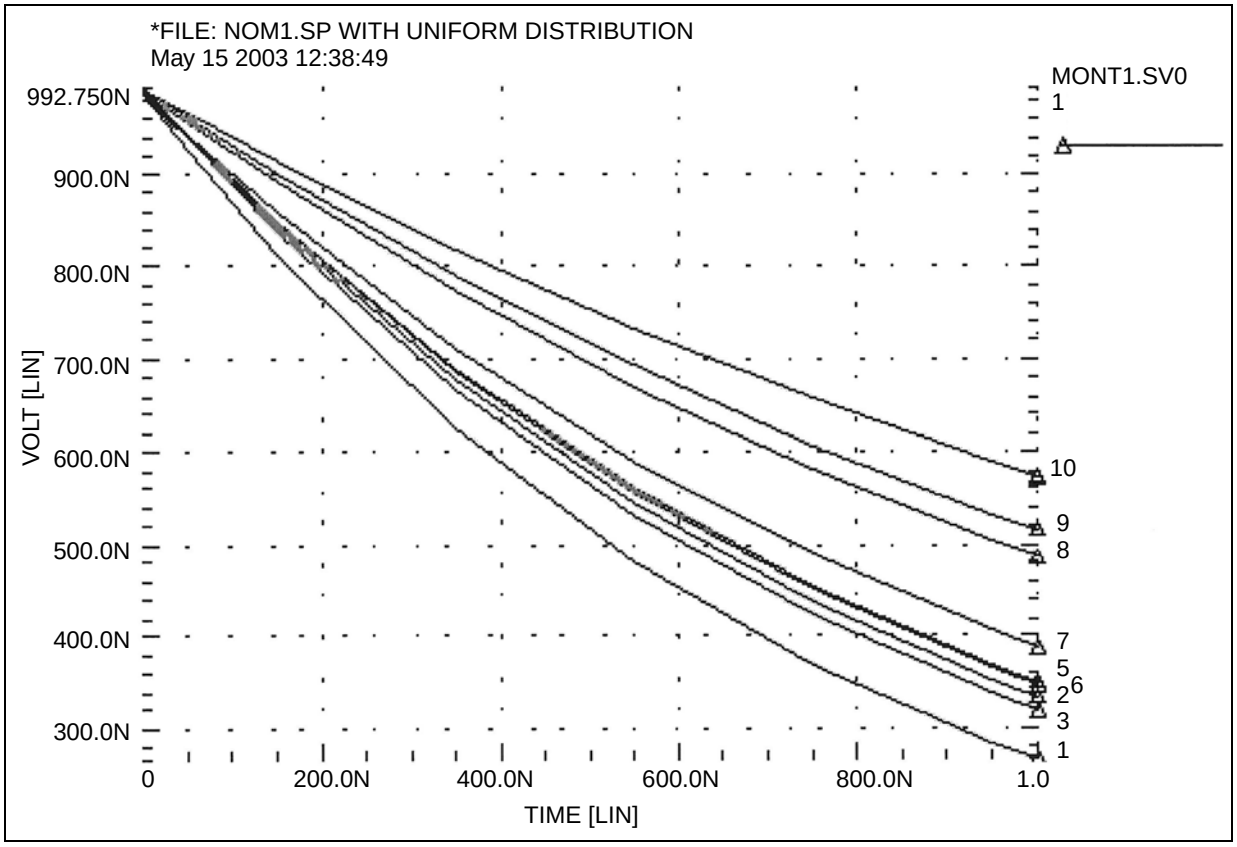
RC Time Constant

This simple example shows uniform distribution for resistance and capacitance. It also shows the resulting transient waveforms for 10 different random values.

This example is based on demonstration netlist rc_monte.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
*FILE: MON1.SP WITH UNIFORM DISTRIBUTION
.OPTION LIST POST
.PARAM RX=UNIF(1, .25) CX=UNIF(1, .25)
.TRAN .1 1 SWEEP MONTE=10
.IC 1 1
R1 1 0 RX
C1 1 0 CX
.PRINT I(R1) I(C1)
.END
```

Figure 102 Monte Carlo Analysis of RC Time Constant

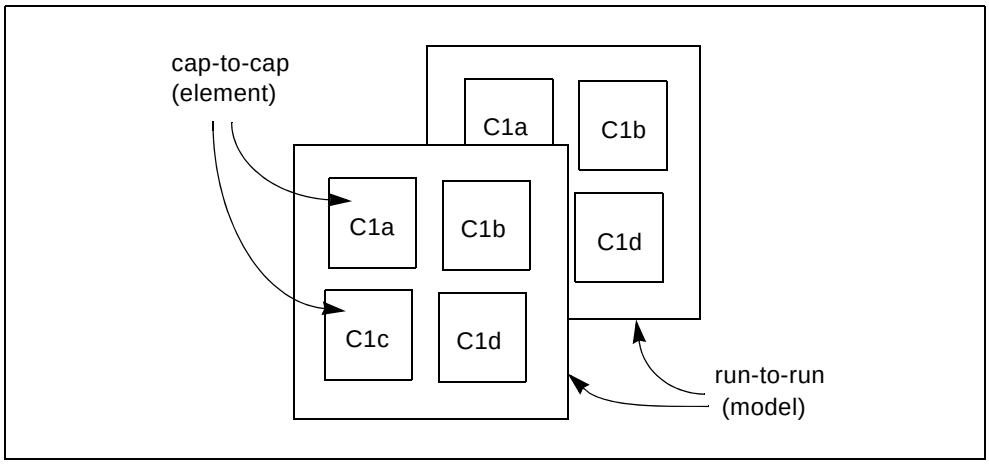


Switched Capacitor Filter Design

Capacitors used in switched-capacitor filters consist of parallel connections of a basic cell. Use Monte Carlo techniques in HSPICE to estimate the variation in total capacitance. The capacitance calculation uses two distributions:

- Minor (element) distribution of cell capacitance from cell-to-cell on a single die.
- Major (model) distribution of the capacitance from wafer-to-wafer or from manufacturing run-to-run.

Figure 103 Monte Carlo Distribution



You can approach this problem from physical or electrical levels.

- The physical level relies on physical distributions, such as oxide thickness and polysilicon line width control.
- The electrical level relies on actual capacitor measurements.

Physical Approach:

1. Since oxide thickness control is excellent for small areas on a single wafer, you can use a local variation in polysilicon to control the variation in capacitance for adjacent cells.
2. Next, define a local poly line-width variation and a global (model-level) poly line-width variation. In this example:
   - The local polysilicon line width control for a line 10 m wide, manufactured with process A, is ±0.02 m for a 1-sigma distribution.
   - The global (model level) polysilicon line-width control is much wider; use 0.1 m for this example.
3. The global oxide thickness is 200 angstroms with a ±5 angstrom variation at 1 sigma.
4. The cap element is square with local poly variation in both directions.

5. The cap model has two distributions:
   - poly line-width distribution
   - oxide thickness distribution.

   The effective length is:

   $$Leff=Ldrawn - 2 \cdot DEL$$

   The model poly distribution is half the physical per-side values:

```
C1a 1 0 CMOD W=ELPOLY L=ELPOLY
C1b 1 0 CMOD W=ELPOLY L=ELPOLY
C1C 1 0 CMOD W=ELPOLY L=ELPOLY
C1D 1 0 CMOD W=ELPOLY L=ELPOLY
$ 10U POLYWIDTH,0.05U=1SIGMA
$ CAP MODEL USES 2*MODPOLY  .05u= 1 sigma
$ 5angstrom oxide thickness AT 1SIGMA
.PARAM ELPOLY=AGAUSS(10U,0.02U,1)
+ MODPOLY=AGAUSS(0,.05U,1)
+ POLYCAP=AGAUSS(200e-10,5e-10,1)
.MODEL CMOD C THICK=POLYCAP DEL=MODPOLY
```

Electrical Approach:

The electrical approach assumes no physical interpretation, but requires a local (element) distribution and a global (model) distribution. In this example:

- You can match the capacitors to ±1% for the 2-sigma population.
- The process can maintain a ±10% variation from run to run for a 2-sigma distribution.

```
C1a 1 0 CMOD SCALE=ELCAP
C1b 1 0 CMOD SCALE=ELCAP
C1C 1 0 CMOD SCALE=ELCAP
C1D 1 0 CMOD SCALE=ELCAP
.PARAM ELCAP=Gauss(1,.01,2) $ 1% at 2 sigma
+ MODCAP=Gauss(.25p,.1,2) $10% at 2 sigma
.MODEL CMOD C CAP=MODCAP
```

Worst Case and Monte Carlo Sweep Example

The following example measures the delay and the power consumption of two inverters. Additional inverters buffer the input and load the output.

This netlist contains commands for two sets of transient analysis: parameter sweep from -3 to +3-sigma, and a Monte Carlo analysis. It creates one set of

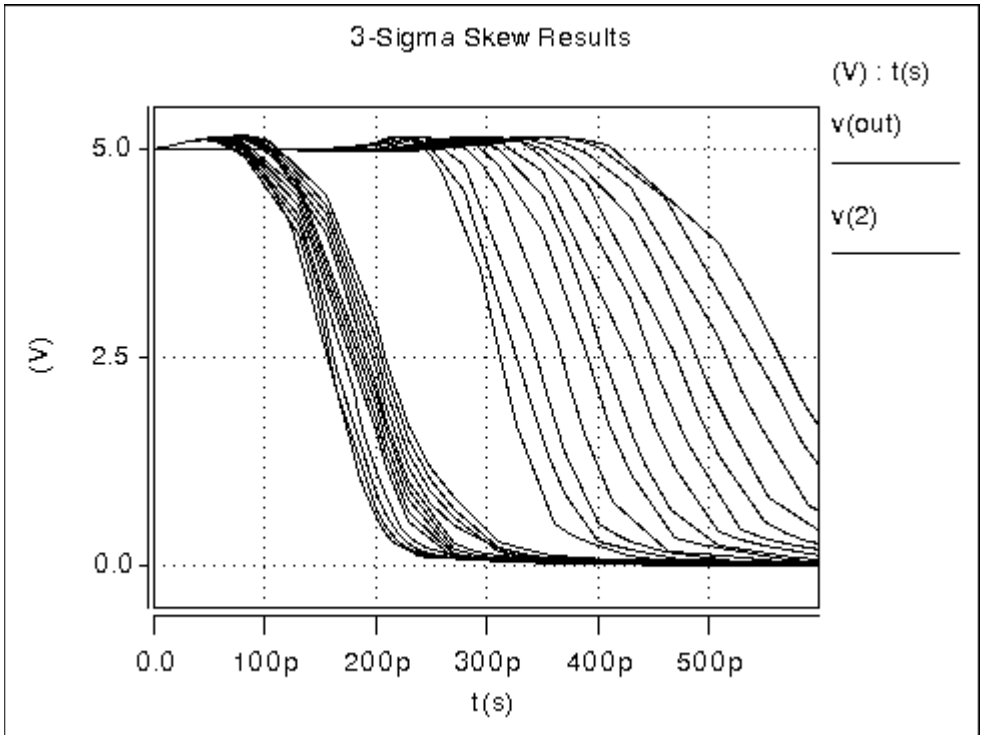
output files (mt0 and tr0) for the sigma sweep, and one set (mt1 and tr1) for Monte Carlo.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma, use measure output
.param vref=2.5 sigma=0
.global 1
vcc  1 0  5.0
vin  in 0 pwl 0,0 0.2n,5
x1 in 2 inv
x2 2 3 inv
x3 3 out inv
x4 out 4 inv
.macro inv in out
  mn out in 0 0 nch w=10u l=1u
  mp out in 1 1 pch w=10u l=1u
.eom
.param mult1=1
+ polycd=agauss(0,0.06u,1)   xl='polycd-sigma*0.06u'
+ nactcd=agauss(0,0.3u,1)  xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1)  xwp='pactcd+sigma*0.3u'
+ toxcd=agauss(200,10,1)   tox='toxcd-sigma*10'
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtoncd+sigma*0.05'
+ rshncd=agauss(50,8,1)   rshn='rshncd-sigma*8'
+ rshpcd=agauss(150,20,1)   rshp='rshpcd-sigma*20'
* level=28 example model
.model nch nmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwn tox=tox delvto=delvton rsh=rshn
...
.model pch pmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp js=3e-04 jsw=9e-10
...
* transient with sweep
.tran 20p 1.0n   sweep sigma -3 3 .5
.meas s_delay trig v(2) val=vref fall=1
+           targ v(out) val=vref fall=1
.meas s_power rms power
* transient with Monte Carlo
.tran 20p 1.0n   sweep monte=100
.meas m_delay trig v(2) val=vref fall=1
+           targ v(out) val=vref fall=1
.meas m_power rms power
.probe tran v(in) v(1) v(2) v(3) v(4)
.end
```

Transient Sigma Sweep Results

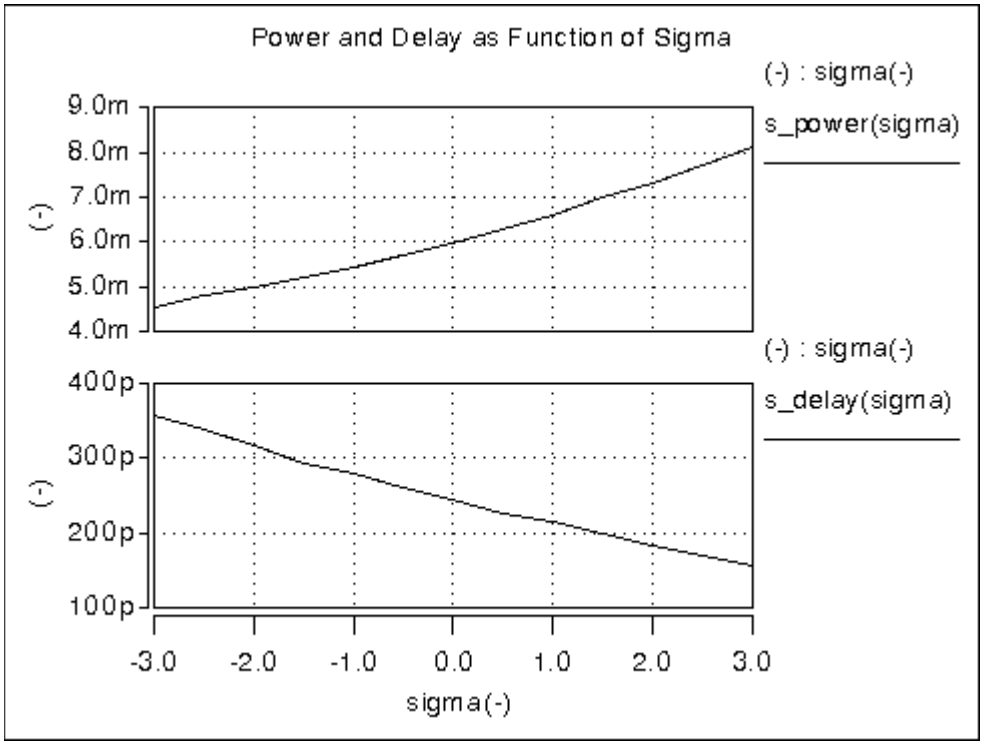
The plot in Figure 104 shows the family of transient analysis curves for the transient sweep of the sigma parameter from -3 to +3 from the file inv.tr0. In the sweep, HSPICE uses the values of sigma to update the skew parameters, which in turn modify the actual NMOS and PMOS models.

Figure 104 Sweep of Skew Parameters from -3 Sigma to +3 Sigma



To view the measured results, plot the inv.mt0 output file. The plot in Figure 105 shows the measured pair delay and the total dissipative power, as a function of the parameter sigma.

Figure 105 Sweep MOS Inverter, Pair Delay and Power: -3 Sigma to 3 Sigma



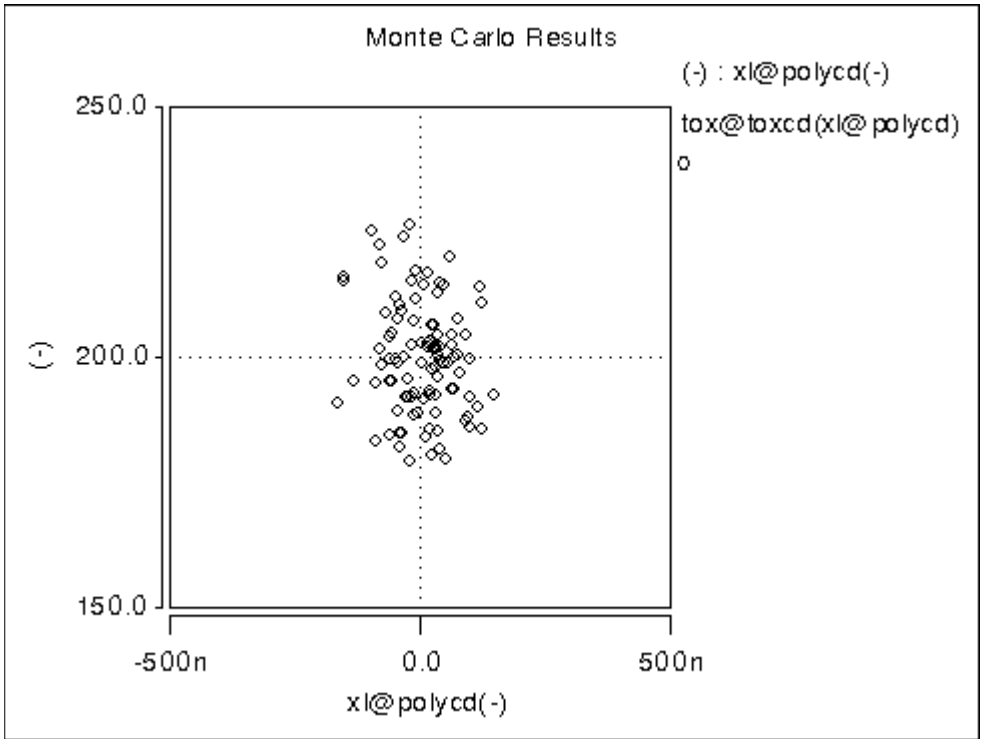
Monte Carlo Results

This section describes the output of the Monte Carlo analysis in HSPICE. The plot in Figure 106 shows that the relationship between `TOX` against `XL` (polysilicon width=transistor length)) is completely random, as set up in the input file.

To generate this plot in CosmosScope:

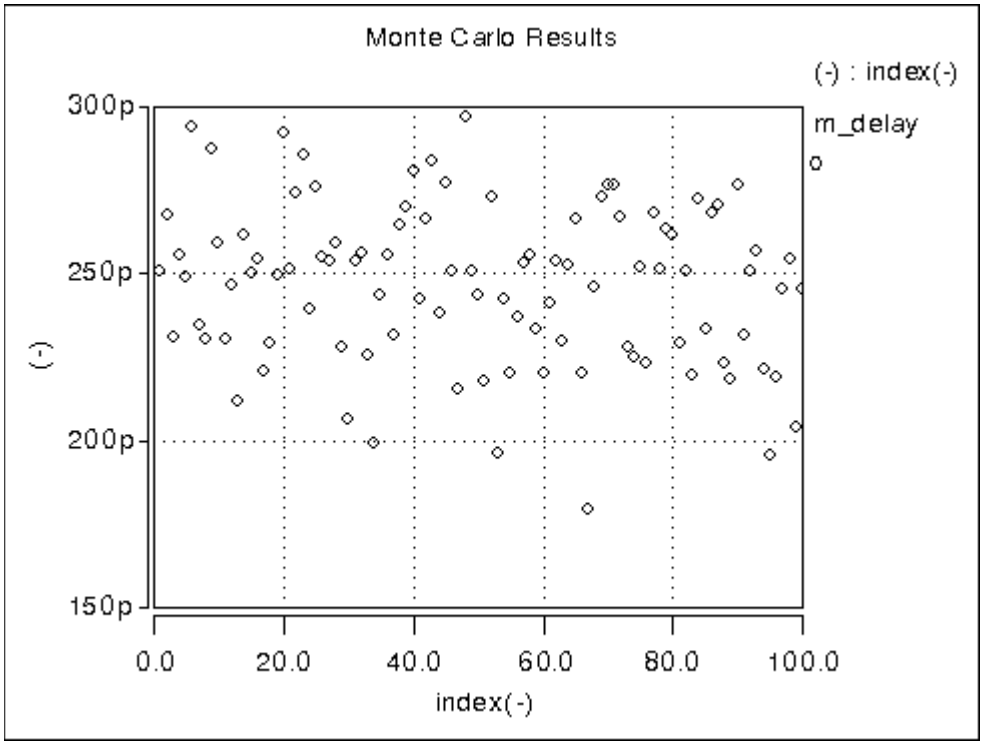
1. Read in the file inv.mt1.
2. Open the Calculator, select TOX (left mouse button), transfer to calculator (middle mouse button), and then select and transfer XL.
3. On the WAVE pulldown in the calculator, select f(x), and then click the plot icon.
4. Using the right mouse button on the plotted waveform, select Attributes to change from the line plot to symbols.

Figure 106 Scatter Plot, XL and TOX



The next graph (see Figure 107) is a standard scatter plot showing the measured delay for the inverter pair against the Monte Carlo index number.

Figure 107 Scatter Plot of Inverter Pair Delay



If a particular result looks interesting; for example, if the simulation 68 (monte carlo index=68) produces the smallest delay, then you can obtain the Monte Carlo parameters for that simulation.

```
*** monte carlo  index =    68 ***
   MONTE CARLO PARAMETER DEFINITIONS
 polycd  xl              = -1.6245E-07
 nactcd  xwn             =  3.4997E-08
 pactcd  xwp             =  3.6255E-08
 toxcd   tox             =   191.0
 vtoncd  delvton         = -2.2821E-02
         delvtop         =  4.1776E-02
 vtopcd
 rshncd  rshn            =   45.16
 rshpcd  rshp            =   166.2
 m_delay=  1.7929E-10  targ=  3.4539E-10   trig=  1.6610E-10
 m_power=  6.6384E-03  from=  0.0000E+00     to=  1.0000E-09
```

In the preceding listing, the m_delay value of 1.79e-10 seconds is the fastest pair delay. You can also examine the Monte Carlo parameters that produced this result.

The information on shortest delay and so forth is also available from the statistics section at the end of the output listing. While this information is useful to determine whether the circuit meets specification, it is often desirable to understand the relationship of the parameters to circuit performance. Plotting the results against the Monte Carlo index number does not help for this purpose. You need to generate plots that display a Monte Carlo result as a function of a parameter. For example, Figure 108 shows the inverter pair delay to channel as a function of poly width, which relates directly to device length.

Figure 108 Delay as a function of Poly width (XL)

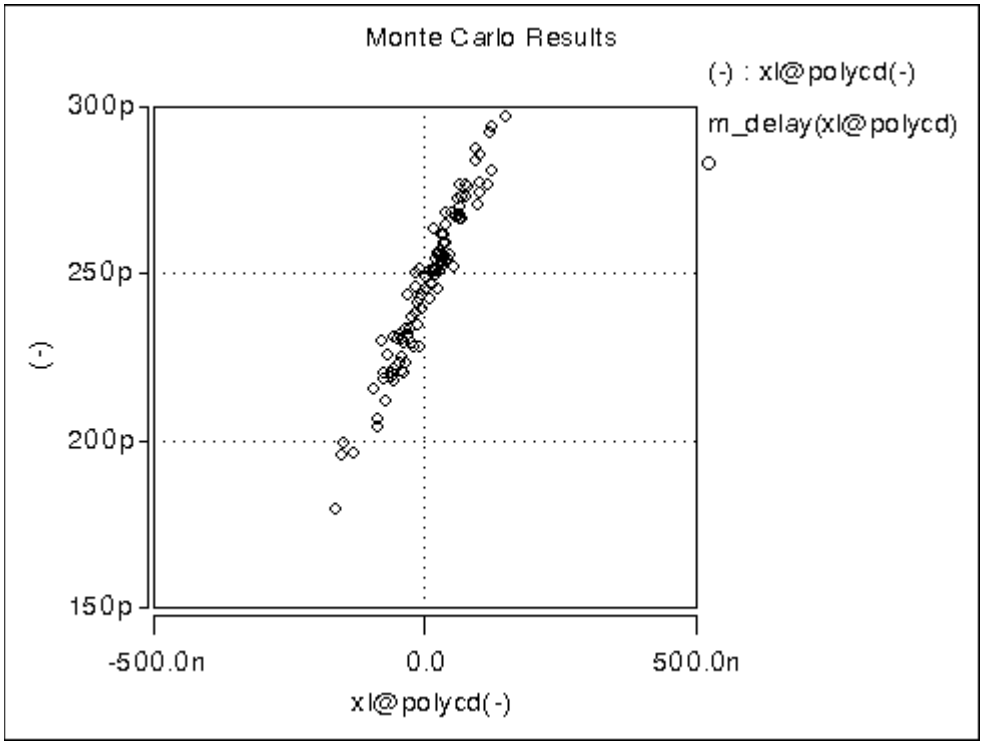
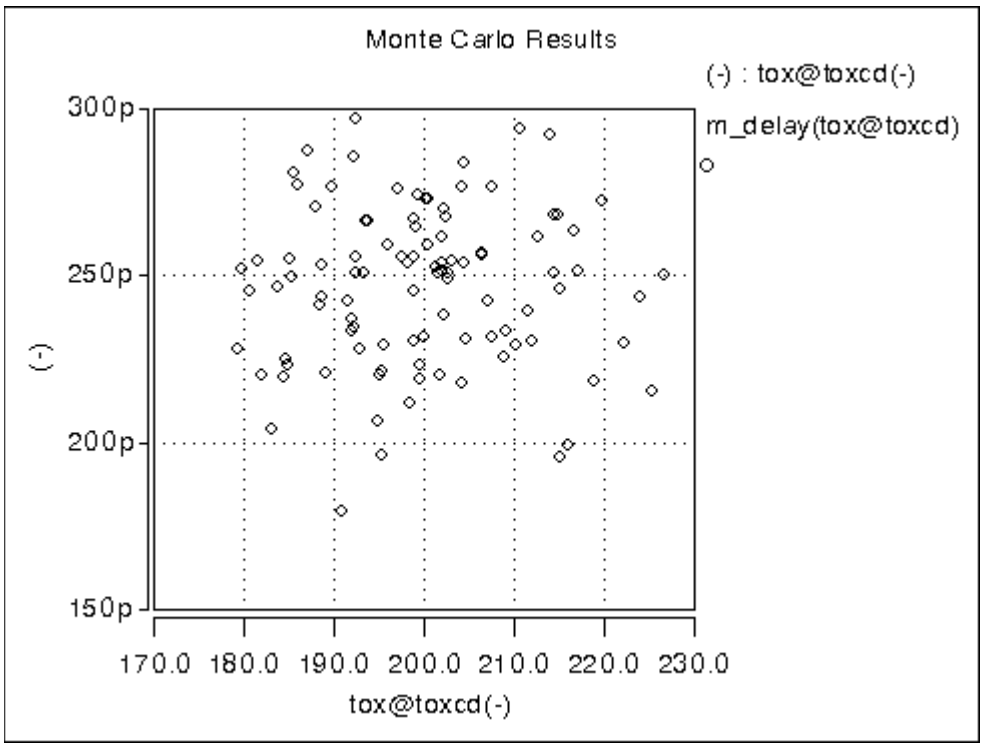


Figure 109 shows the pair delay against the TOX parameter. The scatter plot shows no obvious dependence, which means that the effect of TOX is much smaller than XL. To explore this in more detail, set the XL skew parameter to a constant and run a simulation.

Figure 109 Sensitivity of Delay with TOX



The plot in Figure 110 overlays the skew result with the ones from Monte Carlo. The skew simulation traverses the design space with all parameters changing in parallel and then produces a relationship between power and delay, which shows as a single line. Monte Carlo exercises a variety of independent parameter combinations, and shows that there is no simple relationship between the two results. Since the distributions were defined as Gaussian in the netlist, parameter values close to the nominal are more often exercised than the ones far away. With the relatively small number of samples, the chance of hitting a combination at the extremes is very small. In other words, designing for 3-sigma extreme for every parameter is probably not a good solution from the point of view of economy.

Figure 110 Superimposing Sigma Sweep Over Monte Carlo

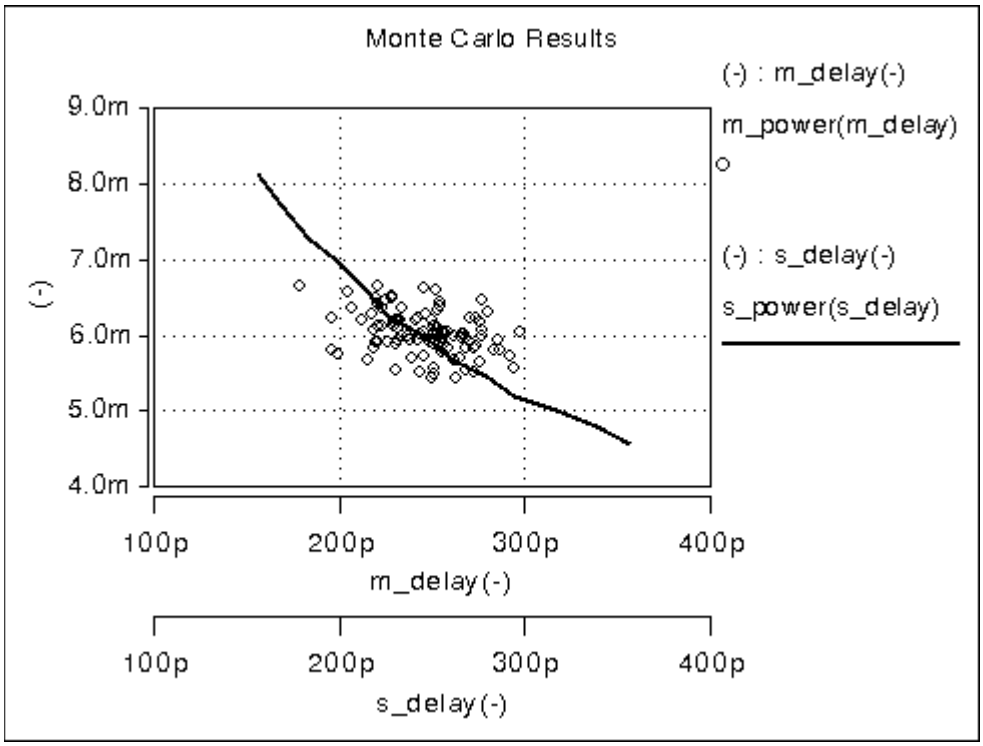
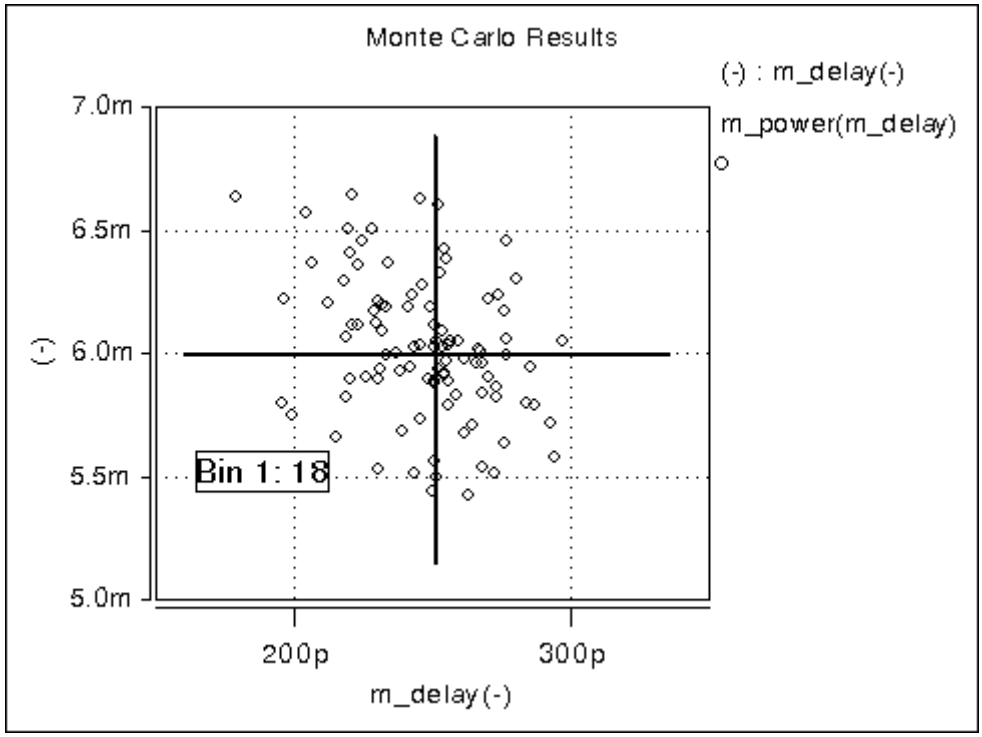


Figure 111 superimposes the required part grades for product sales onto the Monte Carlo plot. This example uses a 250 ps delay and 6.0 mW power dissipation to determine the four binning grades.

Figure 111 Speed/Power Yield Estimation



Sorting the results from inv.mt1 yields:

- Bin1 - 18%
- Bin2 - 30%
- Bin3 - 31%
- Bin4 - 21%

If this circuit is representative of the entire chip, then the present yield should be 18% for the premium Bin 1 parts, assuming variations in process parameters as specified in the netlist. Of course this example only shows the principle on how to analyze the Monte Carlo results; there is no market for a device with two of these inverters.

Simulating the Effects of Global and Local Variations with Monte Carlo

Monte Carlo analysis is dependent on a method to describe variability. Four different approaches are available in HSPICE:

- specify distributions on parameters and apply these to instance parameters
- specify distributions on parameters and apply these to model parameters
- specify distributions on model parameters using `DEV/LOT` construct
- specify distributions on model parameters in a variation block.

While the first three methods are still supported in HSPICE, the method based on the variation block emphasized here for improvements and future developments. The variation block is described in Chapter 14, Variation Block, and Monte Carlo analysis controlled by the variation block is described in Chapter 15, Monte Carlo Analysis.

In the following sections, the first three methods are described. The description relies on test cases, which can be found in the tar file monte_test.tar in directory $<installdir>/demo/hspice/apps.

Variations Specified on Geometrical Instance Parameters

This method consists of defining parameters with variation using the distribution functions `UNIF`, `AUINF`, `GAUSS`, `AGAUSS`, and `LIMIT`. These parameters are then used to generate dependent parameters or in the place of instance parameters. In a Monte Carlo simulation, at the beginning of each sample, new random values are calculated for these parameters. For each reference, a new random value is generated; however, no new value is generated for a derived parameter. Therefore, it is possible to apply independent variations to parameters of different devices, as well as the same variation to parameters of a group of devices. Parameters that describe distributions can be used in expressions, thus it is possible to create combinations of variations (correlations).

These concepts are best explained with circuit examples. In the three following examples, variation is defined on the width of a physical resistor, which has a model. If this device was a polysilicon resistor for example, then the variations describe essentially the effects of photoresist exposure and etching on the width of the poly layer.

- test1.sp has a distribution parameter defined called globw. A parameter called globwidth is assigned the value of globw. The parameter globwidth is assigned a different random value for each Monte Carlo sample. The parameter globwidth is used to define the width of the physical resistors r1, r2, r3, and r4, with model “resistor”. Since parameter globwidth does not have its own distribution defined, but rather gets its value from the parameter globw, the value for globwidth is the same wherever it is used; thus the resistors have the same width for each Monte Carlo sample, and therefore the same resistance. When plotting the simulation results v1, v2, v3, and v4 from the `.meas` file, the waveforms overlay perfectly. This type of setup is typically used to model global variations, which means variations that affect all devices the same way.
- test2.sp has a distribution parameter defined called locwidth. This parameter is used to define the width of the physical resistors r1, r2, r3, and r4, with model “resistor”. Since the parameter has its own distribution defined, its value will be different for each reference, and of course for each Monte Carlo sample. Therefore, the resistors will always have different values, and the voltages will be different. This type of setup is typically used to model local variations, which means variations that affect devices in a different way.
- test3.sp has two kinds of distributions defined: globw/globwidth as in the first example, and locwidth as in the second example. The sum of the two is used to define the width of the resistors. Therefore, the resistors will always have different widths: a common variation due to globwidth and a separate variation due to locwidth. In the example, the distribution for locwidth was chosen as narrower than for globwidth. When overlaying the measurement results, the large common variation can easily be seen; however, all voltages are different.

In summary, each reference to a parameter with a specified distribution causes a new random variable to be generated for each Monte Carlo sample. When referencing the parameter on an instance, the effect of a local variation is created. When referencing the parameter on an expression for a second parameter and using the second parameter on an instance, then the effect of a global variation is created.

Variations Specified in the Context of Subcircuits

The concept explained in the previous section applies also to subcircuits as instances, and instances within subcircuits. Here we again use the example of a physical resistor, with variation of its width.

- test4.sp uses a subcircuit for each resistor instead of the top-level resistors in test3.sp. On each subcircuit, a parameter “width” is assigned a value by an expression, which is the same for all of them. This value is then passed into the subcircuit and the resistor width gets this value. Because the expression is the same for all subcircuits, the value of parameter “width” will be the same for all subcircuits, thus it expresses a global variation. Therefore all resistors have the same width, and the terminal voltages are the same.
- In test5.sp, if a different “width” is used for the subcircuits, then the expressions are treated separately, get local variation assigned, and different values are passed into the subcircuit. In test5.sp, the differences inside of the expressions are kept numerically very small, thus the differences from the different values of “locwidth” are dominant and the results look almost identical to the ones from test3.sp.
- In test6.sp, the resistor width is assigned inside of the subcircuit. The variations get picked up from the top level. Because each subcircuit is a separate entity, the parameter “w” is treated as a separate reference, thus each resistor will have its own value, partly def

In summary, each subcircuit has its own parameter space, therefore it is possible to put groups of identical components into a subcircuit, and within each group all devices have the same parameter values, but between the groups, parameters are different. When specifying variations on these parameters, the effects of local variations between the groups are created.

Variations on a Model Parameter Using a Local Model in Subcircuit

If a model is specified within a subcircuit, then the specified parameter values apply only to the devices in the same subcircuit. Therefore, it is possible to calculate the value of a model parameter within the subcircuit; for example, as a function of geometry information.

When specifying variations on these parameters, the effects of local variations between subcircuits are created. If this method is used at the extreme with one device per subcircuit, then each device has its own model. This approach leads to a substantial overhead in the simulator and is therefore not recommended.

Indirect Variations on a Model Parameter

In sections Variations Specified on Geometrical Instance Parameters and Variations Specified in the Context of Subcircuits, variations on geometrical parameters were presented. If we want to specify variations on a model parameter; for example, the threshold of a MOS device, then the approach explained in the previous section with one model per device in a subcircuit could be used. However, this is impractical because the netlist needs to be created to call each device as a subcircuit, and because of the overhead. Since variations are of interest only on a few model parameters, an indirect method of varying model parameters can be used. Some special instance parameters are available for this purpose. For example, for MOS devices, the parameter delvt0 defines a shift in threshold.

Referencing a parameter with a distribution as value for `delvt0` creates the effect of local threshold variations. A significant number of parameters of this type are available in HSPICE for BSIM3 and BSIM4 models. The variations can be tailored for each device depending on its size for example. A disadvantage of this method is that the netlist needs to be parameterized properly to get the correct variations. The process of preparing a basic netlist for Monte Carlo simulations with this approach is tedious and error prone, therefore it is best handled with scripts.

Bsim3 supports the following instance parameters:

L, w, ad, as, pd, ps, nrd, nrs, rdc, rsc, off, ic, dtemp, delvto, geo, sa, sb, sd, nf, stimod, 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, mulu0, mulua, mulub, tnodeout, rth0, cth0, deltox, delk1, delnfct, and acnqsmod.

Bsim4 supports the following instance parameters:

L, w, ad, as, pd, ps, nrd, nrs, rdc, rsc, off, ic, dtemp, delvto, geo, rbsb, rbdb, rbpb, rbps, rbpd, trnqsmod, acnqsmod, rbodymod, rgatemod, geomod, rgeomod, nf, min, mulu0, delk1, delnfct, deltox, sa, sb, sd, stimod, 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, xgw, ngcon, sca, scb, scc, sc, delk2, delxj, mulngate, delrsh, delrshg, dellpe0, deldvt0, and mulvsat.

Variations Specified on Model Parameters

In this section, we investigate the method of specifying distributions on parameters and using these parameters to define values of model parameters. With this approach, the netlist does not have to be parameterized. The `modmonte` option can be used to distinguish between global variations (all devices of a particular model have the same parameter set) or local variations (every device has a unique random value for the specified parameters).

- test10.sp shows a simple case where the model parameter for sheet resistivity is assigned a distribution defined on the parameter `rsheet`. The results show that all resistors have the same value for each Monte Carlo sample, but a different one for different samples. This setup is useful for studying global variations.
- test11.sp has `.option modmonte=1` added. Now every resistor has a different value.

Note that `.option modmonte` has no effect on any other approach presented here.

In summary, assigning parameters with specified distributions to model parameters allows for investigating the effects of global or local variations, but not both. The possibility of selecting one or the other with a simple option is misleading in the sense that the underlying definitions for global and local variations are not the same for a realistic semiconductor technology.

Variations Specified Using DEV and LOT

The two limitations of the approach described in section Variations Specified on Model Parameters are resolved in this method by specifying global and local variations directly on a model parameter with the syntax:

```
parameterName=parameterValue LOT/distribution LotDist
+ DEV/distribution DevDist
```

Where,

`LOT` keyword for global distribution
`DEV` keyword for local distribution
distribution is as explained in section Variations Specified on Geometrical Instance Parameters
`LotDist`, `DevDist` characteristic number for the distribution. 3-sigma value for Gaussian distributions.

- test12.sp has large global and small local variation, similar to the setup in the file test3.sp The result shows four different curves, with a large common part and small separate parts. The amount of variation defined in the two files is the same. The curves look different from the test3.sp results, because different random sequences are used. However the statistical results (sigma) converge for a large number of samples.

There is no option available to select only local or only global variations. This can be an obstacle if the file is read-only or encrypted.

Combinations of Variation Specifications

Specifying distributions on parameters and applying them to model parameters can be used on some models and the `DEV`/`LOT` approach on others in the same simulation.

- test13.sp has `DEV`/`LOT` specified for model res1, and the parameter “width” for model res2. The values for the resistors with model res1 are different, and the values for resistors with model res2 are the same.
- test14.sp is similar to test7.sp and has `modmonte=1` specified. All four resistors have different values. However, note that in reality, the sigma for width would be different when simulating local or global variations.
- test15.sp has instance parameter variations specified on two resistors and `DEV`/`LOT` on two others. From the waveforms, v3 and v4 form a first pair, and v1 and v2 a second pair.

It is also possible to mix variations on instance parameters and model parameters in the same setup.

- test16.sp has small instance parameter variations specified on width and relatively large model parameter variations on the sheet resistivity, `rsh`. The results show four different waveforms, with a common behavior.
- test17.sp shows instance and model parameter variations as in the previous test case, but `.option modmonte` is set to 1, thus the model variations affect every device in a different way. The results show completely independent behavior of all four resistors.

If an instance parameter or instance parameter variations and model parameter variations are specified on the same parameter, then the instance parameter always overrides the model parameter. Because only few parameters can be used in both domains, this case is rather seldom, but it needs to be considered to avoid unexpected results.

- test18.sp has model variation specified on width with a parameter. Two resistors have width also defined on instance. The resistors with instance parameter do not vary at all. The other two resistors vary independently, as expected because `.option modmonte` is set to 1.
- test19.sp is similar to test18.sp with `.option modmonte` set to 0. The two resistors that do not have width defined on the instance line vary together.
- test20.sp has `DEV`/`LOT` specified. Instance parameters override variations on selected resistors.

Variation on Model Parameters as a Function of Device Geometry

For local variations (see DC Mismatch Analysis), it is a common requirement to specify variation on a model parameter as a function of device geometry. For example, the MOS device threshold was observed to vary with the total device area.

The approach explained in the section Indirect Variations on a Model Parameter can be used. While this allows for specifying local variations on each device, it does not include the capability of using expressions based on element parameters. Thus, variation cannot be described with an expression that includes the device's geometry. Conceptually, a netlist processor could be written that inserts the appropriate values for the parameters as a function of device size. (Synopsys does not make such a tool available).

The DEV/LOT approach has no mechanism to describe variation as a function of an element parameter.

Conclusion

The three approaches described above for specifying variations are not well suited for semiconductor technologies, because of one or more of the following issues:

- require changes to netlist
- difficult to recognize whether a variation is global or local
- no way to describe variability as a function of device size
- no way to run only global or only local variation.

To overcome these issues, a new approach was introduced in HSPICE. This approach is based on a so called *variation block*. See chapter 14 For details on the variation block, see Chapter 14, Variation Block, and for details on how Monte Carlo analysis is processed with this new approach, see Chapter 15, Monte Carlo Analysis.

B

Full Simulation Examples

Contains information and sample input netlists for two full simulation examples.

The examples in this chapter show the basic text and post-processor output for two sample input netlists.

Note:

> The examples are for Synopsys HSPICE, but with minimal modifications, you can also apply these examples to HSPICE RF.

The first example uses AvanWaves to view results. The second example uses CosmosScope.

Simulation Example Using AvanWaves

Input Netlist and Circuit

This example is based on demonstration netlist example.sp, which is available in directory $<installdir>/demo/hspice/bench. This example is an input netlist for a linear CMOS amplifier. Comment lines indicate the individual sections of the netlist.

```
* Example HSPICE netlist, using a linear CMOS amplifier
* netlist options
.option post probe brief nomod
* defined parameters
.param analog_voltage=1.0
* global definitions
.global vdd
* source statements
Vinput in gnd SIN ( 0.0v analog_voltage 10x )
Vsupply vdd gnd DC=5.0v
* circuit statements
```

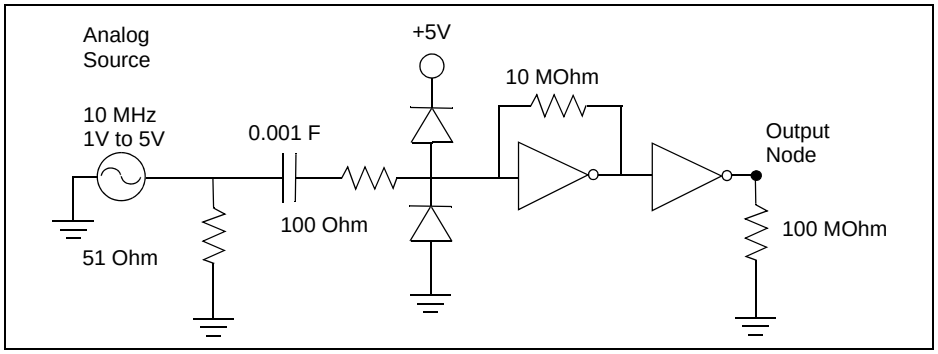
```
Rinterm in gnd 51
Cincap in infilt 0.001
Rdamp infilt clamp 100
Dlow gnd clamp diode_mod
Dhigh clamp vdd diode_mod
Xinv1 clamp inv1out inverter
Rpull clamp inv1out 1x
Xinv2 inv1out inv2out inverter
Routterm inv2out gnd 100x
* subcircuit definitions
.subckt inverter in out
Mpmos out in vdd vdd pmos_mod l=1u w=6u
Mnmos out in gnd gnd nmos_mod l=1u w=2u
.ends
* model definitions
.model pmos_mod pmos level=3
.model nmos_mod nmos level=3
.model diode_mod d
* analysis specifications
.TRAN 10n 1u sweep analog_voltage lin 5 1.0 5.0
* output specifications
.probe TRAN v(in) v(clamp) v(inv1out) v(inv2out) i(dlow)
.measure TRAN falltime TRIG v(inv2out) VAL=4.5v FALL=1
+ TARG V(inv2out) VAL=0.5v FALL=1
.end
```

Figure 112 on page 589 is a circuit diagram for the linear CMOS amplifier in the circuit portion of the netlist. The two sources in the diagram are also in the netlist.

Note:

The inverter symbols in the circuit diagram are constructed from two complementary MOSFET elements. Also, the diode and MOSFET models in the netlist do not have non-default parameter values, except to specify Level 3 MOSFET models (empirical model).

Figure 112 Circuit Diagram for Linear CMOS Inverter



Execution and Output Files

The following section displays the output files from a HSPICE simulation of the amplifier shown in the previous section. To execute the simulation, enter:

```
hspice example.sp > example.lis
```

In this syntax, the input netlist name is example.sp, and the output listing file name is example.lis. Simulation creates the following output files:

Table 61 HSPICE Output Files

Filename	Description
example.ic	Initial conditions for the circuit.
example.lis	Text simulation output listing.
example.mt0	Post-processor output for .MEASURE statements.
example.pa0	Subcircuit path table.
example.st0	Run-time statistics.
example.tr0	Post-processor output for transient analysis.

The following subsections show text files to simulate the amplifier by using HSPICE on a Sun workstation. The example does not show the two post-processor output files, which are in binary format.

Example.ic

```
* "simulator" "HSPICE"
* "version" "98.4 (981215) "
* "format" "HSP"
* "rundate" "13:58:43  01/08/1999"
* "netlist" "example.sp "
* "runtitle" "* example hspice netlist using a linear
* cmos amplifier "
* time=  0.
* temperature= 25.0000
*** BEGIN: Saved Operating Point ***
.option gmindc=  1.0000p
.nodeset
+ clamp= 2.6200
+ in= 0.
+ infilt= 2.6200
+ inv1out= 2.6200
+ inv2out= 2.6199
+ vdd= 5.0000
***   END: Saved Operating Point ***
```

Example.lis

```
Using: /net/sleepy/l0/group/hspice/98.4beta/sol4/hspice

****** HSPICE -- 98.4 (981215)  13:58:43 01/08/1999 solaris
Copyright (C) 1985-2002 by Synopsys Corporation.
Unpublished-rights reserved under US copyright laws.
This program is protected by law and is subject to the
terms and conditions of the license agreement found in:

/afs/rtp.synopsys.com/product/hspice/current/license.txt

Use of this program is your acceptance to be bound by this
license agreement. HSPICE is a trademark of Synopsys, Inc.

Input File: example.sp

lic:
lic: FLEXlm:v5.12 USER:hspiceuser HOSTNAME:hspiceserv
+ HOSTID:8086420f PID:1459

lic: Using FLEXlm license file:
lic: /afs/rtp/product/distrib/bin/license/license.dat
lic: Checkout hspice; Encryption code: AC34CE559E01F6E05809
lic: License/Maintenance for hspice will expire on 14-apr-
+ 1999/1999.200
```

```
lic: 1(in_use)/10 FLOATING license(s) on SERVER hspiceserv
lic:
******
* example hspice netlist using a linear cmos amplifier

******
* netlist options
.option post probe brief nomod

* defined parameters
Opening plot unit=15
file=./example.pa0
****** HSPICE --    98.4 (981215) 13:58:43
******  01/08/1999 solaris ******

* example hspice netlist using a linear cmos amplifier
***** transient analysis  tnom=25.000 temp=25.000 *****
*** parameter analog_voltage = 1.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp= 2.6200 0:in    =0.       0:infilt= 2.6200
+0:inv1out =2.6200 0:inv2out=2.6199   0:vdd   =5.0000

Opening plot unit=15
file=./example.tr0

**warning** negative-mos conductance=1:mnmos  iter=2
vds,vgs,vbs=     2.45          2.93       0.
gm,gds,gmbs,ids=  -3.636E-05     1.744E-04  0.  1.598E-04
******

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****
falltime=3.9149E-08 targ=7.1916E-08   trig=3.2767E-08

*** HSPICE -- 98.4 (981215) 13:58:43
*** 01/08/1999 solaris ***
* example hspice netlist using a linear cmos amplifier
****** transient analysis tnom=25.000 temp=25.000 ******
*** parameter analog_voltage = 2.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp=2.6200   0:in    =0.       0:infilt= 2.6200
+0:inv1out=2.6200  0:inv2out=2.6199 0:vdd   =5.0000
******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****
falltime=1.5645E-08 targ=5.7994E-08   trig=4.2348E-08
```

```
**** HSPICE --    98.4 (981215) 13:58:43
**** 01/08/1999 solaris ****

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****
*** parameter analog_voltage  =  3.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp= 2.6200 0:in      =  0.      0:infilt= 2.6200
+0:inv1out=2.6200 0:inv2out=2.6199 0:vdd    = 5.0000
******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****
falltime=1.1917E-08 targ=5.6075E-08   trig=4.4158E-08

****** HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******
* example hspice netlist using a linear cmos amplifier

 ***** transient analysis tnom=25.000 temp=25.000 *****
*** parameter analog_voltage  =  4.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp= 2.6200 0:in      =  0.      0:infilt= 2.6200
+0:inv1out=2.6200 0:inv2out=2.6199 0:vdd    = 5.0000
******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****

falltime=7.5424E-09 targ=5.3989E-08   trig=4.6447E-08

****** HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****
*** parameter analog_voltage  =  5.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp= 2.6200 0:in      =  0.      0:infilt= 2.6200
+0:inv1out=2.6200 0:inv2out=2.6199 0:vdd    = 5.0000
******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom=25.000 temp=25.000 *****

falltime=6.1706E-09 targ=5.3242E-08   trig=4.7072E-08
```

```
meas_variable=falltime
mean=16.0848n      varian=1.802e-16
sigma=13.4237n      avgdev= 9.2256n
max  =39.1488n      min   = 6.1706n

***** job concluded

****** HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******
* example hspice netlist using a linear cmos amplifier
*** job statistics summary tnom=25.000 temp=25.000 ***

total memory used        155 kbytes

# nodes=8 # elements=14
# diodes=2 # bjts   =   0 # jfets  =0 # mosfets=4
analysis   time   # points   tot. iter   conv.iter
op point    0.04       1        23
transient   4.71     505      9322   2624 rev=664
readin      0.03
errchk      0.01
setup       0.01
output      0.01

total cpu time          4.84 seconds
job started at 13:58:43 01/08/1999
job ended   at 13:58:50 01/08/1999

lic: Release hspice token(s)
HSPICE job example.sp completed.
Fri Jan 8 13:58:50 EST 1999

Example.pa0
1 xinv1.
2 xinv2.
```

Example.st0

```
***** HSPICE --    98.4 (981215) 13:58:43
*****  01/08/1999 solaris
Input File: example.sp
lic: FLEXlm:v5.12 USER:hspiceuser HOSTNAME:hspiceserv
+ HOSTID:8086420f PID:1459
lic: Using FLEXlm license file:
lic: /afs/rtp/product/distrib/bin/license/license.dat
lic: Checkout hspice; Encryption code: AC34CE559E01F6E05809
lic: License/Maintenance for hspice will expire on
+ 14-apr-1999/1999.200
```

```
lic: 1(in_use)/10 FLOATING license(s) on SERVER hspiceserv

lic:
init: begin read circuit files, cpu clock=2.21E+00
   option probe
   option nomod
init: end read circuit files, cpu clock=2.23E+00
+ memory=145 kb
init: begin check errors, cpu clock=2.23E+00
init: end check errors, cpu clock=2.24E+00 memory=144 kb
init: begin setup matrix, pivot=   10 cpu clock=2.24E+00
establish matrix -- done, cpu clock=2.24E+00 memory=146 kb
re-order matrix -- done, cpu clock=2.24E+00 memory=146 kb
init: end setup matrix, cpu clock=2.25E+00 memory=154 kb
sweep: parameter parameter1       begin, #sweeps=  5
parameter: analog_voltage=  1.00E+00
dcop: begin dcop, cpu clock=2.25E+00
dcop: end dcop, cpu clock=2.27E+00 memory=154 kb
tot_iter=11
output: ./example.mt0
sweep: tran tran1 begin, stop_t=1.00E-06 #sweeps=101
cpu clock=  2.28E+00
tran: time=1.03750E-07 tot_iter=78   conv_iter=24
tran: time=2.03750E-07 tot_iter=179   conv_iter=53
tran: time=3.03750E-07 tot_iter=280   conv_iter=82
tran: time=4.03750E-07 tot_iter=381   conv_iter=111
tran: time=5.03750E-07 tot_iter=482   conv_iter=140
tran: time=6.03750E-07 tot_iter=583   conv_iter=169
tran: time=7.03750E-07 tot_iter=684   conv_iter=198
tran: time=8.03750E-07 tot_iter=785   conv_iter=227
tran: time=9.03750E-07 tot_iter=886   conv_iter=256
tran: time=1.00000E-06 tot_iter=987   conv_iter=285

sweep: tran tran1 end, cpu clock=2.82E+00 memory=155 kb
parameter: analog_voltage=  2.00E+00
dcop: begin dcop, cpu clock=2.83E+00
dcop: end dcop, cpu clock=2.83E+00 memory=155 kb
+ tot_iter=14
output: ./example.mt0

sweep: tran tran2 begin, stop_t=1.00E-06 #sweeps=101
+ cpu clock=2.83E+00
tran: time=1.01016E-07 tot_iter=186   conv_iter=54
tran: time=2.02642E-07 tot_iter=338   conv_iter=98
tran: time=3.01763E-07 tot_iter=495   conv_iter=145
tran: time=4.04254E-07 tot_iter=668   conv_iter=198
tran: time=5.02594E-07 tot_iter=841   conv_iter=248
tran: time=6.10102E-07 tot_iter=983   conv_iter=289
```

```
tran: time=7.01850E-07 tot_iter=1161   conv_iter=340
tran: time=8.01776E-07 tot_iter=1306   conv_iter=383
tran: time=9.04268E-07 tot_iter=1481   conv_iter=436
tran: time=1.00000E-06 tot_iter=1654   conv_iter=486

sweep: tran tran2 end, cpu clock=3.71E+00 memory=155 kb
parameter: analog_voltage=  3.00E+00
dcop: begin dcop, cpu clock=3.71E+00
dcop: end dcop, cpu clock=3.72E+00 memory=155 kb
+ tot_iter=17
output: ./example.mt0

sweep: tran tran3 begin, stop_t=1.00E-06 #sweeps=101
+ cpu clock=3.72E+00
tran: time=1.00313E-07 tot_iter=143   conv_iter=42
tran: time=2.01211E-07 tot_iter=340   conv_iter=100
tran: time=3.01801E-07 tot_iter=539   conv_iter=156
tran: time=4.02192E-07 tot_iter=729   conv_iter=211
tran: time=5.01997E-07 tot_iter=917   conv_iter=265
tran: time=6.01801E-07 tot_iter=1088   conv_iter=314
tran: time=7.01801E-07 tot_iter=1221   conv_iter=351
tran: time=8.01801E-07 tot_iter=1362   conv_iter=392
tran: time=9.02387E-07 tot_iter=1515   conv_iter=435
tran: time=1.00000E-06 tot_iter=1674   conv_iter=479

sweep: tran tran3 end, cpu clock=4.57E+00 memory=155 kb
parameter: analog_voltage=  4.00E+00
dcop: begin dcop, cpu clock=4.57E+00
output: ./example.mt0

sweep: tran tran4 begin, stop_t=1.00E-06 #sweeps=101
+ cpu clock=4.58E+00
tran: time=1.00110E-07 tot_iter=236   conv_iter=70
tran: time=2.04376E-07 tot_iter=475   conv_iter=139
tran: time=3.07892E-07 tot_iter=767   conv_iter=221
tran: time=4.01056E-07 tot_iter=951   conv_iter=273
tran: time=5.01086E-07 tot_iter=1250   conv_iter=353
tran: time=6.00965E-07 tot_iter=1541   conv_iter=432
tran: time=7.03668E-07 tot_iter=1805   conv_iter=506
tran: time=8.01114E-07 tot_iter=2046   conv_iter=571
tran: time=9.01005E-07 tot_iter=2308   conv_iter=640
tran: time=1.00000E-06 tot_iter=2528   conv_iter=703

sweep: tran tran4 end, cpu clock=5.83E+00 memory=155 kb
parameter: analog_voltage=  5.00E+00
dcop: begin dcop, cpu clock=5.83E+00

dcop: end dcop, cpu clock=5.84E+00 memory=155 kb
```

```
+ tot_iter=23
output: ./example.mt0

sweep: tran tran5 begin, stop_t=1.00E-06 #sweeps=101
+ cpu clock=5.84E+00
tran: time=1.00195E-07 tot_iter=176   conv_iter=47
tran: time=2.00617E-07 tot_iter=431   conv_iter=115
tran: time=3.00475E-07 tot_iter=661   conv_iter=176
tran: time=4.00719E-07 tot_iter=914   conv_iter=246
tran: time=5.04084E-07 tot_iter=1157   conv_iter=311
tran: time=6.00666E-07 tot_iter=1347   conv_iter=363
tran: time=7.01830E-07 tot_iter=1623   conv_iter=435
tran: time=8.02418E-07 tot_iter=1900   conv_iter=514
tran: time=9.01178E-07 tot_iter=2161   conv_iter=585
tran: time=1.00000E-06 tot_iter=2410   conv_iter=650

sweep: tran tran5 end, cpu clock=7.03E+00 memory=155 kb
sweep: parameter     parameter       1 end
>info:         ***** hspice job concluded
lic: Release hspice token(s)
```

Simulation Graphical Output in AvanWaves

The plots in Figure 113 through Figure 118 on page 602 show the six different post-processor outputs from the simulation of the example netlist. These plots are postscript output from the actual data in AvanWaves, a Synopsys graphical waveform viewer.

Figure 113 Plot of Voltage on Node in

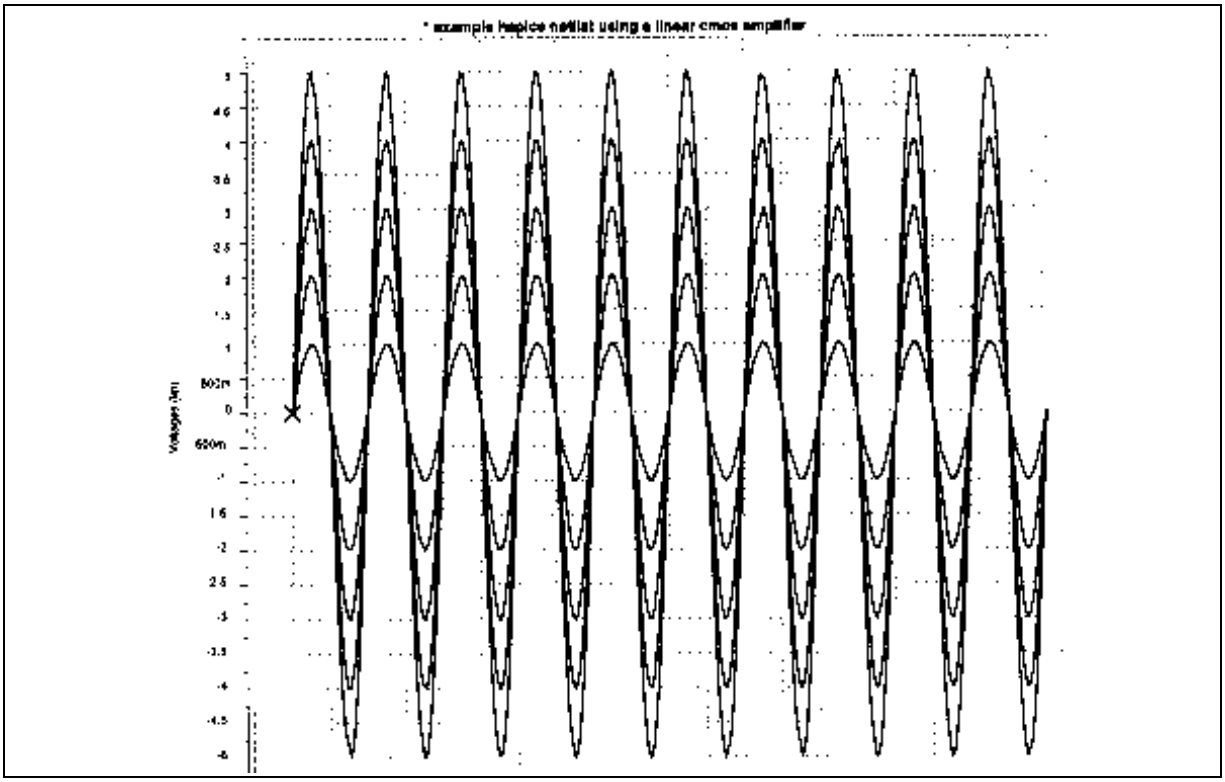


Figure 114 Plot of Voltage on Node clamp vs. Time

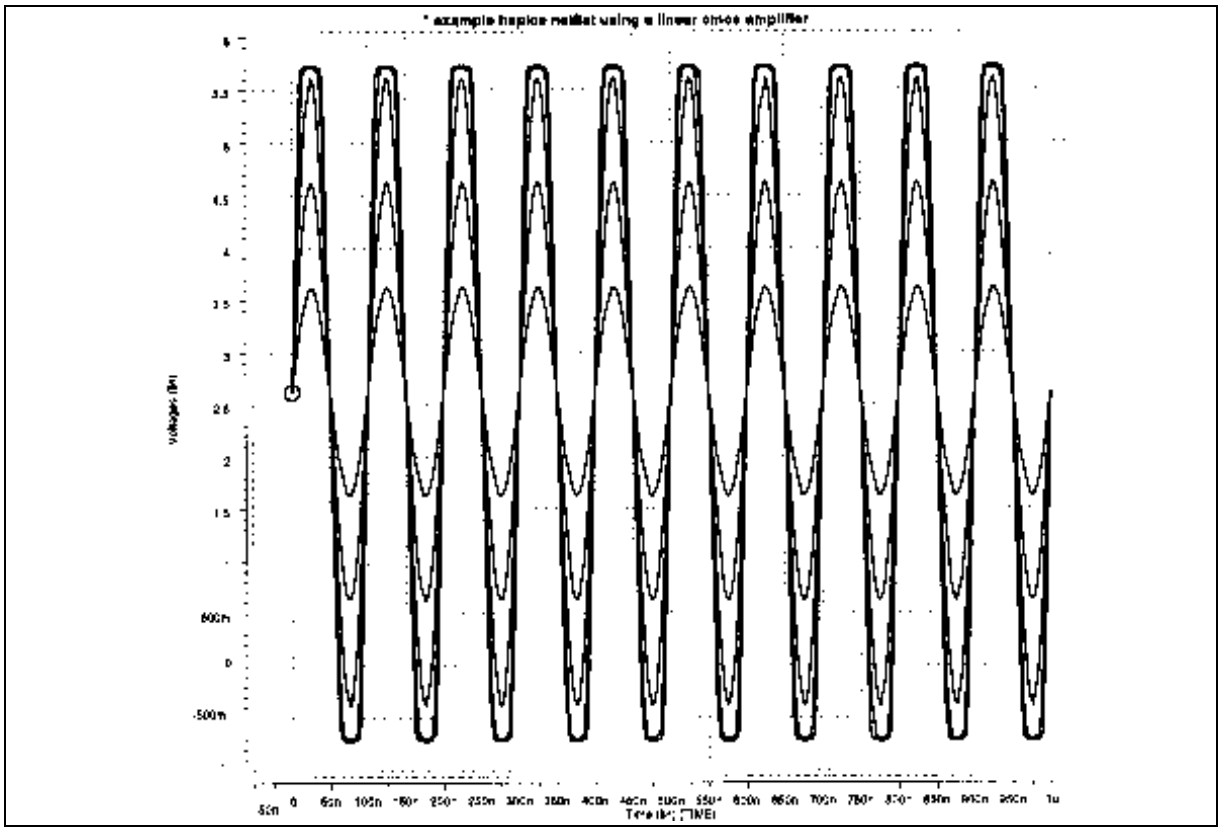


Figure 115 Plot of Voltage on Node inv1out vs.Time

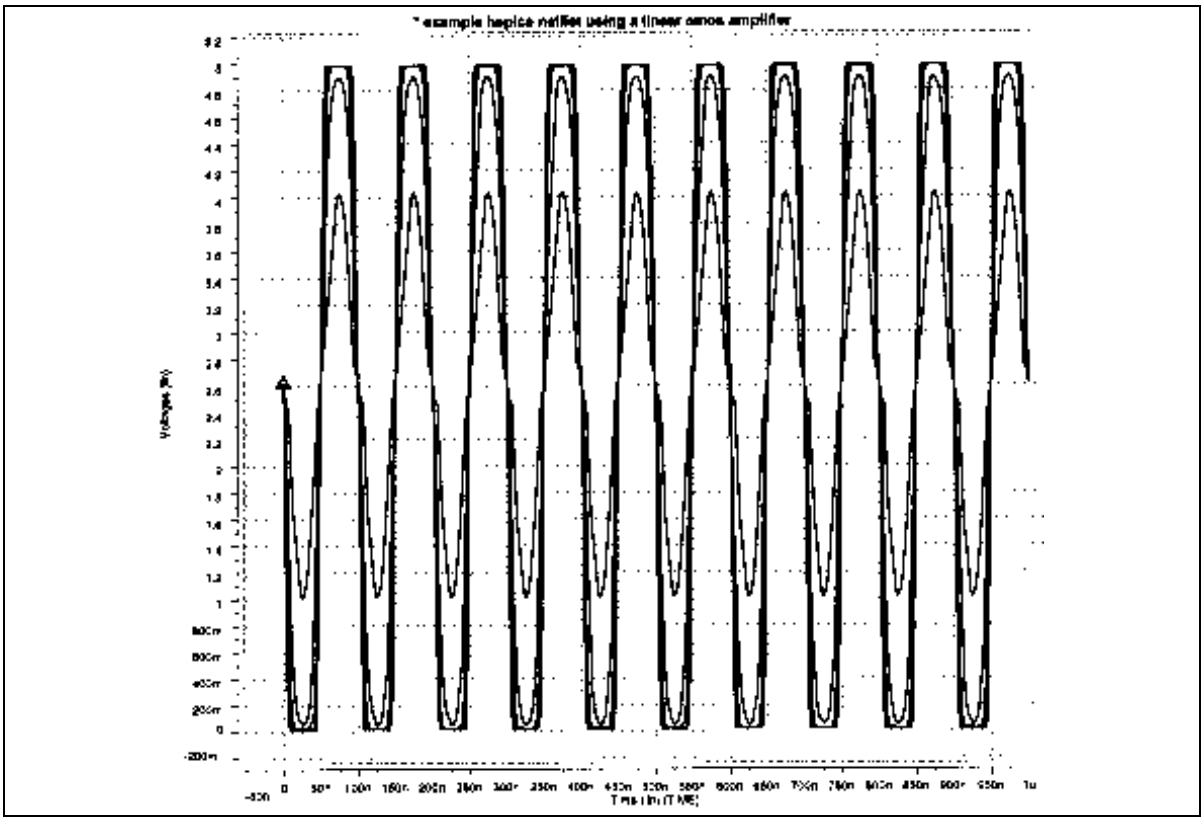


Figure 116 Plot of Voltage on Node inv2out vs. Time

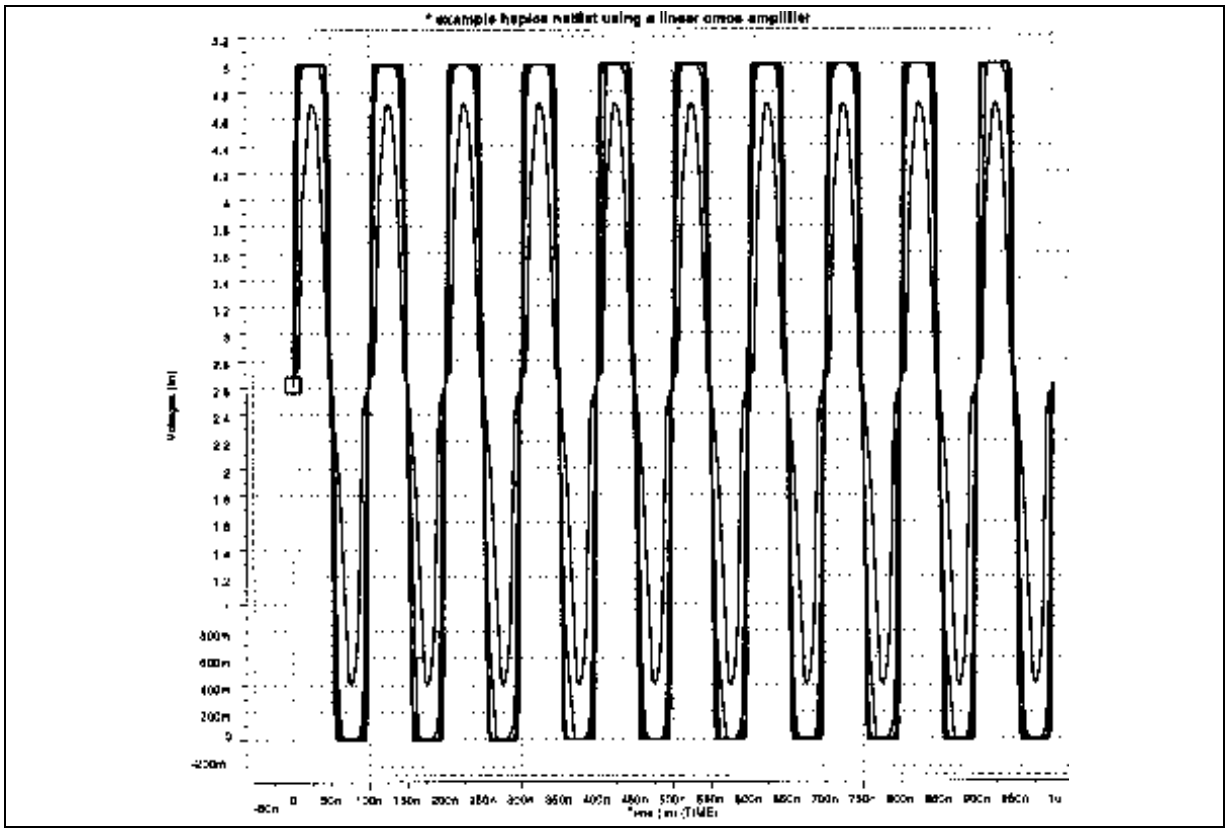


Figure 117 Plot of Current through Diode dlow vs. Time

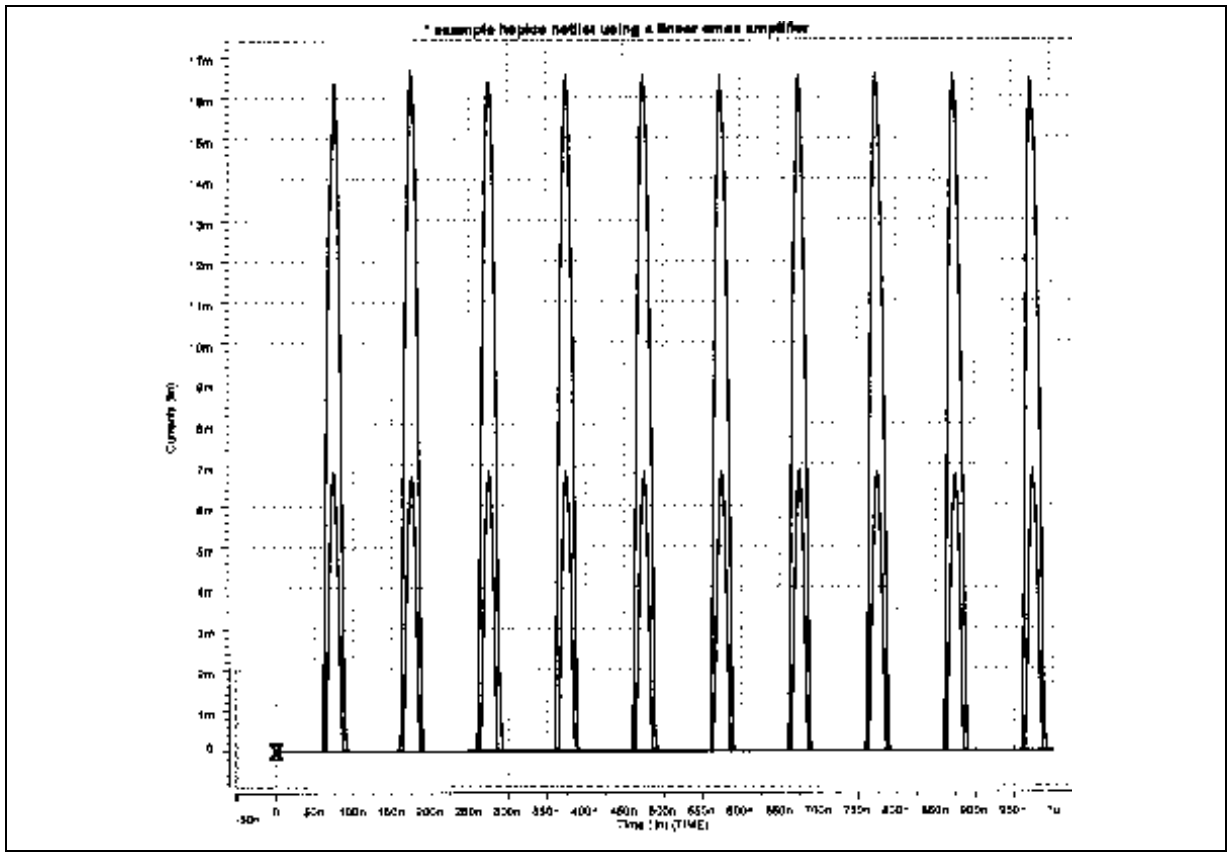
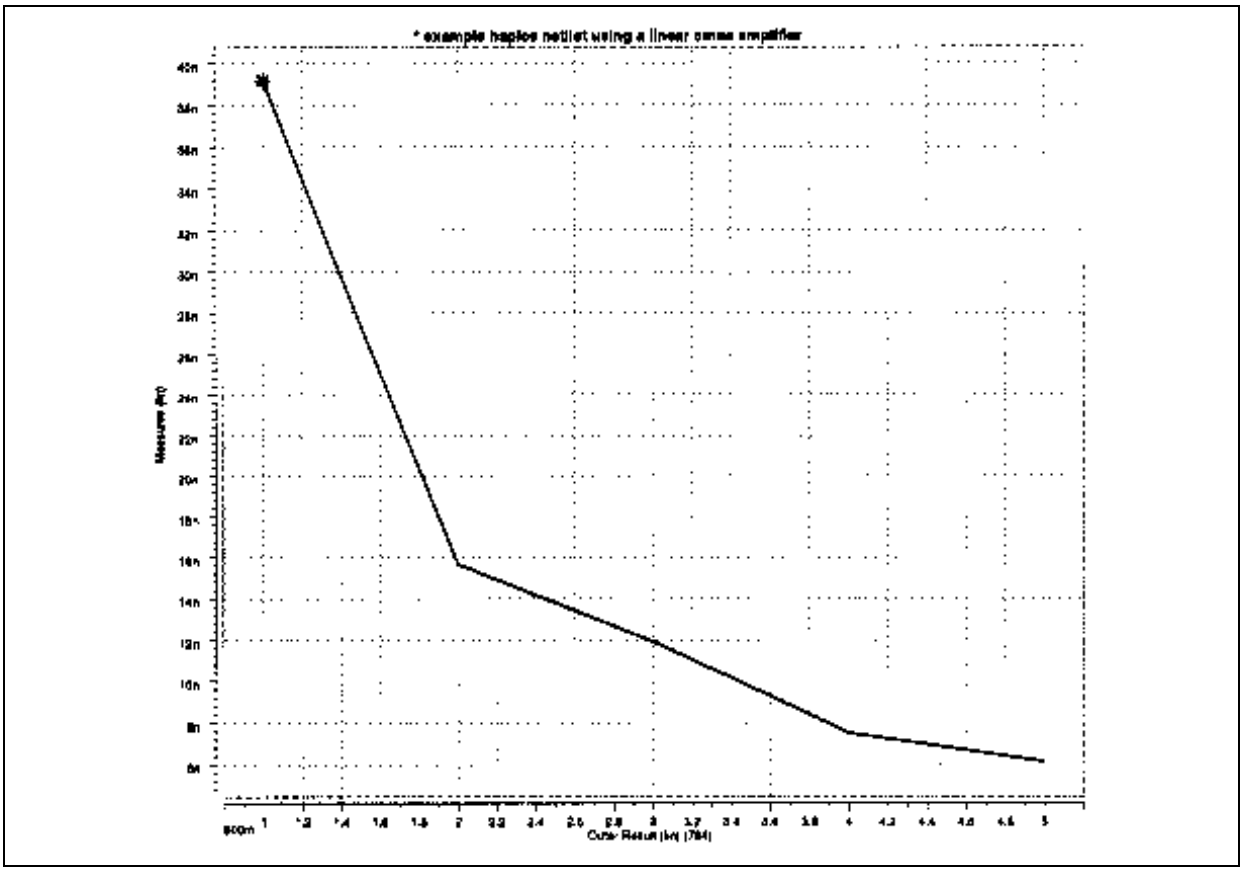


Figure 118 Plot of Measured Variable falltime vs. Amplifier Input Voltage



Simulation Example Using CosmosScope

This example demonstrates the basic steps to perform simulation output and to view the waveform results by using the Synopsys CosmosScope Waveform Viewer.

Input Netlist and Circuit

This example is based on demonstration netlist bjtdiff.sp, which is available in directory $<installdir>/demo/hspice/apps. This shows the input netlist for a BJT diff amplifier. Comment lines indicate the individual sections of the netlists. See the *HSPICE Command Reference* for information about individual commands.

```
*file: bjtdiff.spbjt diff amp with every analysis type
*
```

```
.options acct node list opts nomod post
.param rb1x=aunif(20k,1k,30k) rb2x=aunif(20k,1k,30k)
.tf v(5) vin
.dc vin -0.20 0.20 0.01 sweep monte=3
.ac dec 10 100k 10meghz
.noise v(4) vin 20
.net v(5) vin rout=10k
.pz v(5) vin
.disto rc1 20 .9 1m 1.0
.sens v(4)
.tran 5ns 200ns
.four 5meg v(5) v(15)
.temp -55 150
*
.meas qa_propdly trig v(1) val=0.09 rise=1
+ targ v(5) val=6.8 rise=1
.meas qa_magnitude max v(5)
.meas qa_rmspower rms power
.meas qa_avgv5 avg v(5)
.meas ac qa_bandwidth trig at=100k targ vdb(5) val=36 fall=1
.meas ac qa_phase find vp(5) when vm(5)=52.12
.meas ac qa_freq when vm(5)=52.12
.print dc v(4) v(5) v(14) v(15)
.probe dc v(5) v(15)
.print ac vm(5) vp(5) vm(15) vp(15)
.probe ac vm(5) vp(5) vm(15) vp(15)
.print ac vt(5) vt(15)
.probe noise onoise(m) inoise(m)
.print ac z11(m) z12(m) z22(m) zin(m)
.probe ac z11(p) z12(p) z22(p) zin(p)
.probe disto hd2 hd3 sim2 dim2 dim3
.print tran v(4) v(5) v(14) v(15)
.print tran p(vcc) p(vee) p(vin) power
.probe tran v(5) v(15)
*
vin 1 0 sin(0 0.1 5meg) ac 1
vcc 8 0 12
vee 9 0 -12
*
q1 4 2 6 qnl
q11 14 12 16 qpl
q2 5 3 6 qnl
q21 15 13 16 qpl
rs1 1 2 1k
rs11 1 12 1k
rs2 3 0 1k
rs12 13 0 1k
rc1 4 8 10k
```

```
rc11 14 9 10k
rc2 5 8 10k
rc12 15 9 10k
q3 6 7 9 qnl
q13 16 17 8 qpl
q4 7 7 9 qnl
q14 17 17 8 qpl
rb1 7 8 rb1x
rb2 17 9 rb2x
*
.model qnl npn(bf=80 rb=100 ccs=2pf tf=0.3ns tr=6ns cje=3pf
cjc=2pf
+ va=50 rc=10 trb=.005 trc=.005)
.model qpl pnp(bf=80 rb=100 ccs=2pf tf=0.3ns tr=6ns cje=3pf
cjc=2pf
+ va=50 bulk=0 rc=10)
*
.end
```

Use the previous example (linear CMOS amp) to draw a circuit diagram for this BJT diff amplifier. Also, specify parameter values.

Execution and Output Files

This section displays the various output files from a HSPICE simulation of the BJT diff amplifier example. To execute the simulation, enter:

```
hspice bjtdiff.sp > bjtdiff.lis
```

where the input file is bjtdiff.sp, and the output file is bjtdiff.lis.

Simulation creates the following output files:

Table 62 Output Files

Filename	Description
bjtdiff.ic	Initial conditions for the circuit.
bjtdiff.lis	Text simulation output listing.
bjtdiff.mt0	Post-processor output for `.MEASURE` statements.
bjtdiff.st0	Run-time statistics.
bjtdiff.tr0	Post-processor output for transient analysis.
bjtdiff.sw0	Post-processor output for DC analysis.
bjtdiff.ac0	Post-processor output for AC analysis.
bjtdiff.ma0	Post-processor output for AC analysis measurements.

View HSPICE Results in CosmosScope

The steps below show how to use the Synopsys CosmosScope Waveform Viewer to view the results of AC, DC, and transient analysis from the BJT diff amplifier simulation. Refer to previous examples of .lis, .ic, and .st0 files.

Viewing HSPICE Transient Analysis Waveforms

To view HSPICE transient analysis waveforms, do the following:

1. Invoke CosmosScope.

   From a Unix command line, type:

   ```
   % cscope
   ```

   On a Windows-NT system, choose the menu command:

   Programs > (user_install_location)> CosmosScope

2. Open the Open Plotfiles dialog box:

   File > Open > Plotfiles

3. In the Open Plotfiles dialog box, in the Files of Type fields, select the Hspice Transient (*.tr*) item.
4. In the menu, click on bjtdiff.tr0, and click Open.

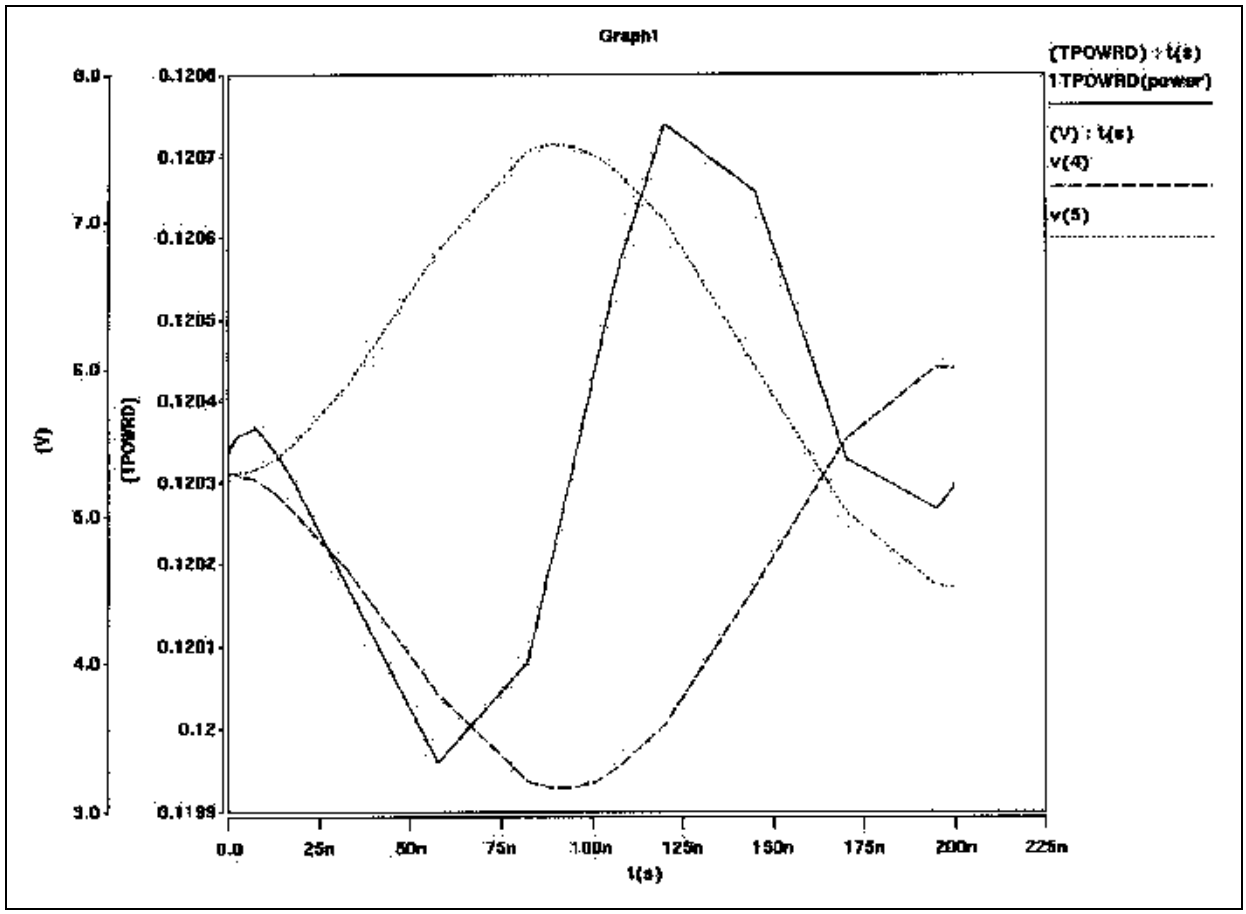
   The Signal Manager and the bjtdiff Plot File windows open.
5. Hold down the Ctrl key, and select the v(4), v(5), and ITPOWERD(power) signals.
6. Click on Plot from the bjtdiff Plot File window.

   Three cascaded plots open.
7. To see three signals in one plot, right-click on the top-most signal name.

   The Signal Menu opens.
8. From the Signal Menu, select Stack Region > Analog 0.
9. Repeat Step 7 for the next topmost signal.

   A plot opens similar the one shown in Figure 119 on page 607.

Figure 119 Transient Analysis: Plot of v(4), v(5), and ITPOWERD (power)



Viewing HSPICE AC Analysis Waveforms

To view HSPICE AC analysis waveforms, do the following:

1. From the Signal Manager dialog box, select bjtdiff(1), and click on Close Plotfiles.

   All transient plots (waveforms) close.

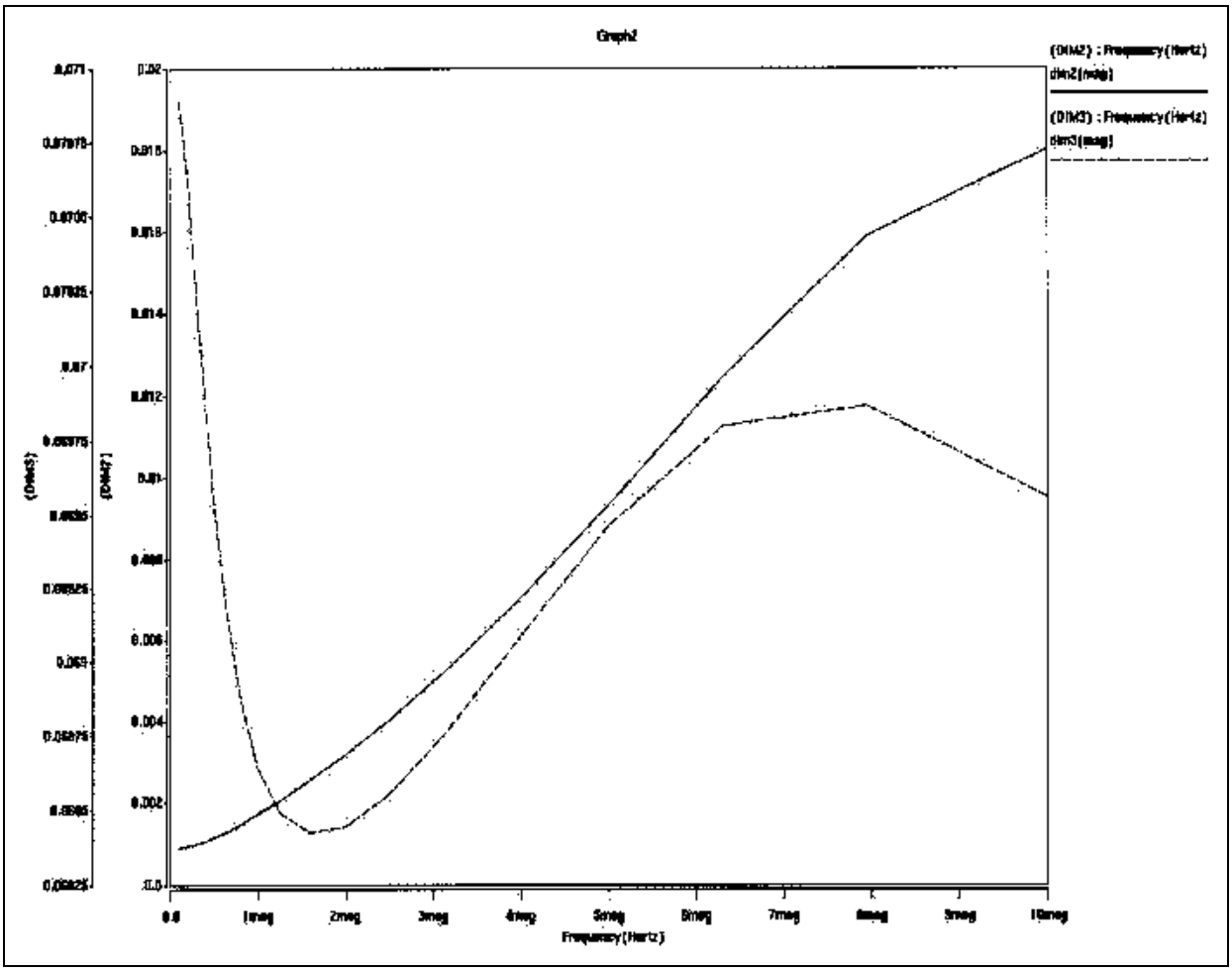
2. In the Signal Manager, click on Open Plotfiles.
3. In the Open Plotfiles dialog box, in the Files of Type fields, select the HSPICE AC (*.ac*) item.
4. Click on bjtdiff.ac0 in the menu, and click Open.

   The bjtdiff Plot File windows open.

5. Hold down the Ctrl key, and select the dim2(mag) and dim3(mag) signals.

6. Click on Plot from the bjtdiff Plot File window.
   Two cascaded plots open.
7. For two signals in a plot, right-click on dim2(mag).
   A Signal Menu opens.
8. From the Signal Menu, select Stack Region > Analog 0.
   A plot opens similar to Figure 120.

Figure 120 AC Analysis Result: Plot of dim2(mag), dim3(mag) from bjtdiff.ac0



Viewing HSPICE DC Analysis Waveforms

To view HSPICE DC analysis waveforms, do the following:

1. From the Signal Manager dialog box, select bjtdiff(1), and click on Close Plotfiles.

   All AC plots (waveforms) close.
2. In the Signal Manager, click on Open Plotfiles.
3. In the Open Plotfiles dialog, Files of Type field, select HSPICE DC (*.sw*).
4. Click on bjtdiff.sw0 and Open in the menu.

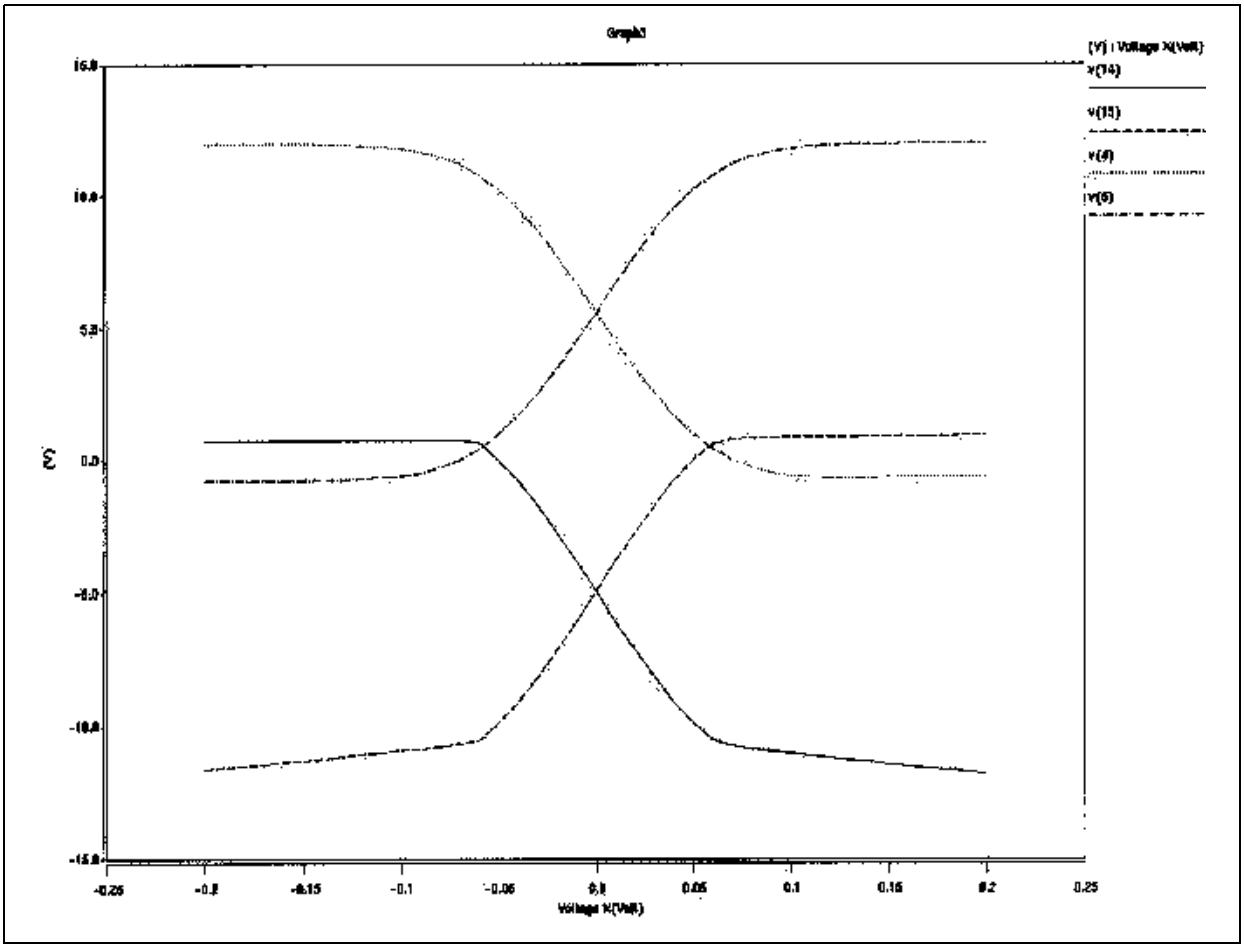
   The Plot File windows open.
5. Hold down the Ctrl key and select all signals.
6. Click on Plot from the bjtdiff Plot File window.

   Four cascaded plots open.
7. To see four signals in one plot, right-click on the name of the top-most signal.

   A Signal Menu opens.
8. From the Signal Menu, select Stack Region > Analog 0.
9. Repeat Steps 7 and 8 for the next two top-most signals.

   A plot opens similar to the one shown in Figure 121.

Figure 121 DC Analysis Result: Plot of v(14), v(15), v(4), and v(5) from bjtdiff.sw0



The *CosmosScope User's and Reference Manual* includes a full tutorial, information about the various Scope tools, and reference information about the Measure tool. You can also find more information on the Synopsys website:

http:// www.synopsys.com

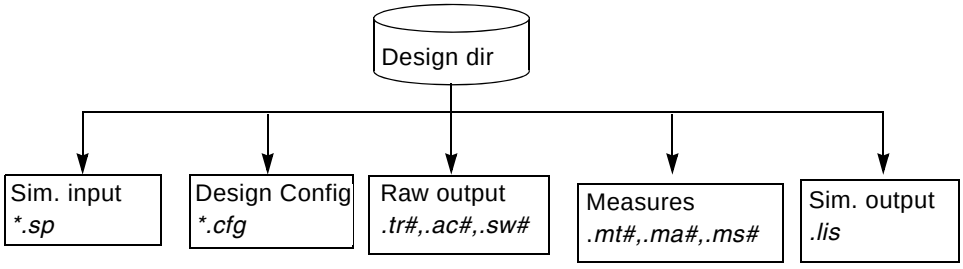
C

HSPICE GUI for Windows

Describes how to use the HSPICE GUI for Windows.

To open and install the the GUI, click on the HSPUI icon. Figure 122 shows the directory structure for the HSPICE GUI for Windows.

Figure 122 Directory Structure



Working with Designs

A new design can be created in several ways. The Launcher allows you to browse for an input file for HSPICE, which has the default file suffix *.sp*. The Launcher Browse button opens a standard file browser.

Selecting a file of the type *<design>.sp* causes the Launcher to display the main form, which contains the following items:

- input filename
- design title (the first line of the file *<design>.sp*)
- output filename
- HSPICE and AvanWaves version

New designs can be saved with the command File > Save.

Table 63 Design Commands in the Launcher

Command	Description
File > New	Clears the Launcher and opens a new design
File > Open	Opens an existing design with the file browser
File > Save	Saves the current design information
File > Save As	Not implemented in Version 1.0
File > Close	Closes the current design
<LastDesigns>	Lists the last five designs opened
File > Exit	Exits the Launcher

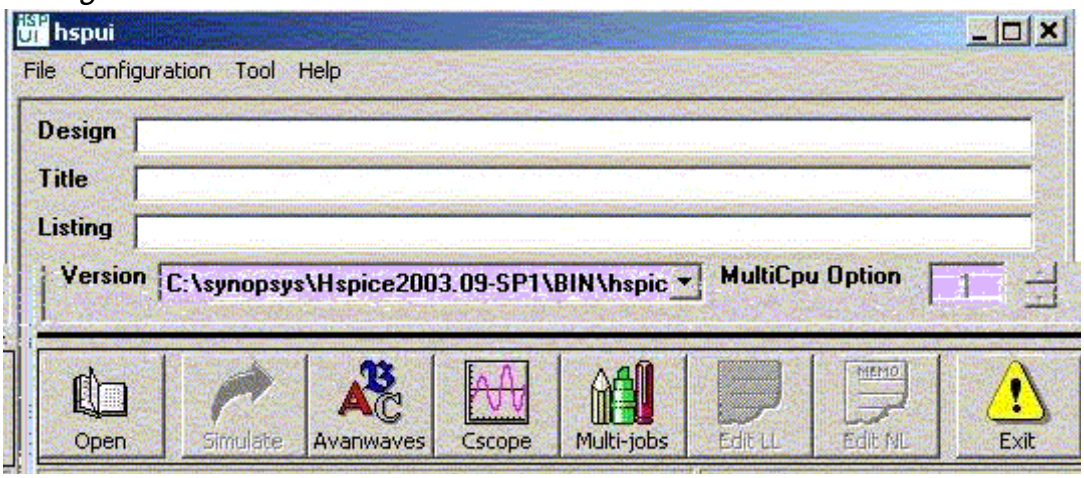
The commands File > New, Open, and Close prompt you to save the current design if changes have occurred.

The Launcher checks on the status of a given design when it is opened. If the input file exists, the Simulate button is active. If the listing file exists for the design, the Edit Listing button is active. The Edit Netlist and AvanWaves buttons are always active.

You do not need to save a design to Simulate or view the results of a simulation with AvanWaves.

Figure 123 shows the main window of the Launcher.

Figure 123 Launcher Main Window



Configuring the HSPICE GUI for Windows

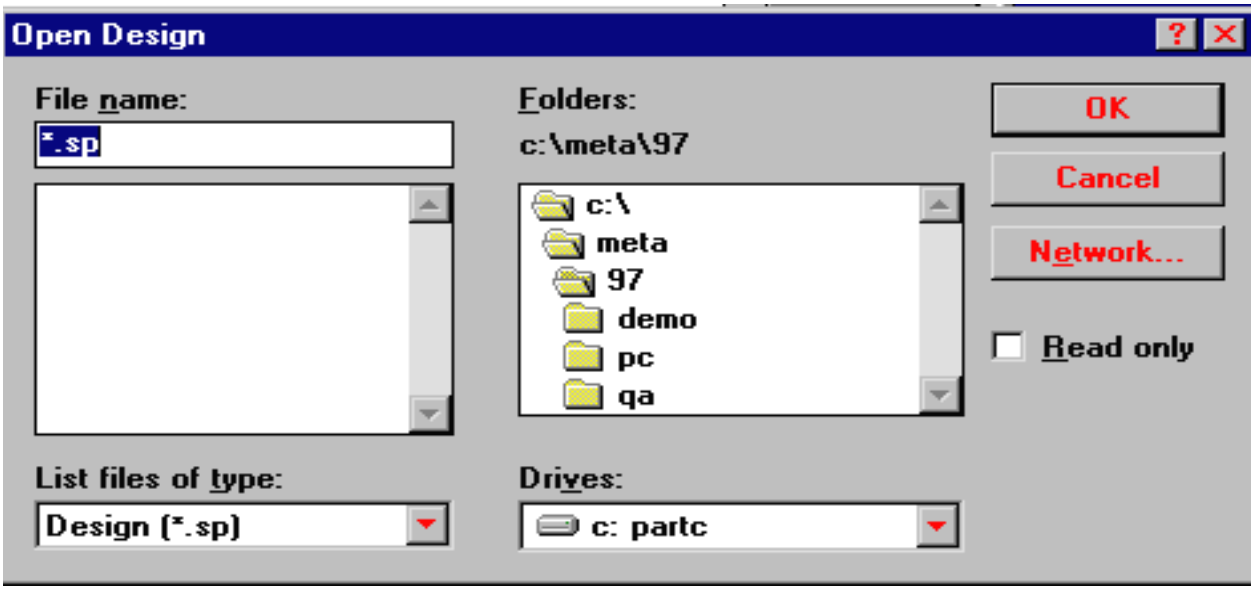
Customize configurations using the Configuration menu of the Launcher as shown in Figure 124.

The start-up directory defaults to the value of the *AVANHOME* environment variable set up during HSPICE installation.

- The input file suffix defaults to *.sp*.
- The output file suffix defaults to *.lis*.
- The editor defaults to *notepad.exe*.

If you change a value, the Launcher updates the *<AVANHOME>/hspui.cfg* file. The next Launcher run provides the new values.

Figure 124 Launcher Options Window



The Configuration > Versions item lists current executables and their paths for the Launcher (HSPUI), HSPICE, and AvanWaves.

Note:

> Standard menu items, such as File and Edit, display on the HSPICE/Win menu bar, but are not available in this release. The Configuration > Version strings change from the main window Versions combo box. You cannot change them here.

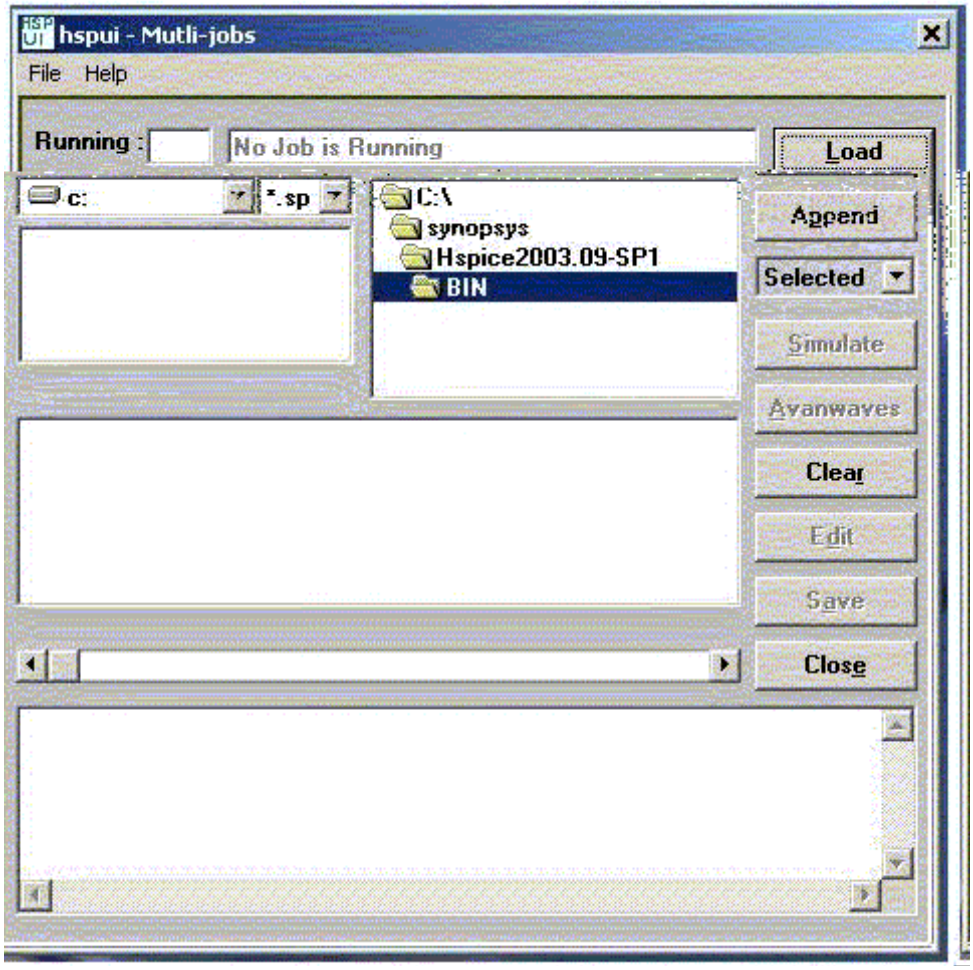
To associate your *<design>.sp* file with the Launcher, use the File >> Associate command in the Windows File Manager. You can double-click on an *.sp* file in the File Manager window to automatically invoke the HSPICE/Win Launcher. Refer to your Windows documentation for details on how to do this.

Running Multiple Simulations

Use the HSPICE/Launcher file browser to build a list of simulations from different directories for consecutive HSPICE processing.

Press Multi-Jobs in the main window to open the HSPICE Multi-Job window (Figure 125). Simulation files are chosen from the Drive/Directory list box and placed in the Files list box.

Figure 125 HSPICE Multi-Jobs Window



Building the Batch Job List

To build a batch job list:

1. Press Multi-Jobs in the main window.
2. Using the Drive/Directory boxes, locate the directory of files that you wish to simulate.
3. To copy all files in the directory, press the Copy button on the right side of the Hspbat window.

   Note that any file names already in the list will be replaced.
4. To add additional files from other directories, repeat Step 2 and use the Append button.

Simulating the Batch Job List

To simulate a batch job list:

1. To simulate all of the files in the Batch Job list, set the pulldown menu to All and press the Simulate button.
2. To run simulation on a single file or a group of files, set the pulldown menu to Selected and select those files you wish to simulate from the Batch Job list box.

   Use the left mouse button to select a single file.

   - Press and hold the Control key and select another file with the left mouse button to add to the selected list.
   - Press and hold the Shift key to select all files between the current file and the last selected file.
3. Press the Simulate button to start the consecutive simulations.

Using the Drag-and-drop Functions

The HSPICE Multi-Jobs window provides a drag-and -rop capability to remove files from the list, edit files, run simulations and view the results with AvanWaves.

Beside the icons, the user also can use the Text Editor box to view and edit the design file (*<design>.sp*). To do this, drag and drop the file from the upper list box to the bottom one. The file contents are displayed in the bottom editor for the user to view and/or edit.

To display files associated with a design, double click on the upper list box on the selected design file (*<design>.sp* file). All associated files (tr#, ac#, sw#, mt# ...) are listed in the bottom list box.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z