HSPICE™ Simulation and Analysis User Guide

Release U-2003.03-PA, March 2003

Comments?
E-mail your comments about Synopsys documentation to doc@synopsys.com

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, Trademarks, and Service Marks of Synopsys, Inc.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, CoCentric, COSSAP, CSim, DelayMill, DesignPower, DesignSource, DesignWare, Eagle*i*, EPIC, Formality, in-Sync, LEDA, ModelAccess, ModelTools, PathBlazer, PathMill, PowerArc, PowerMill, PrimeTime, RailMill, RapidScript, SmartLogic, SNUG, Solv-It, SolvNet, Stream Driven Simulator, Superlog, System Compiler, TestBench Manager, TetraMAX, TimeMill, and VERA are registered trademarks of Synopsys, Inc.

Trademarks (™)

BCView, Behavioral Compiler, BOA, BRT, Cedar, ClockTree Compiler, DC Expert, DC Expert *Plus*, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Compiler, DesignTime, DFT Compiler SoCBIST, Direct RTL, Direct Silicon Access, DW8051, DWPCI, ECL Compiler, ECO Compiler, ExpressModel, Floorplan Manager, FormalVera, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, HDL Advisor, HDL Compiler, Integrator, Interactive Waveform Viewer, JVXtreme, Liberty, Library Compiler, ModelSource, Module Compiler, MS-3200, MS-3400, NanoSim, OpenVera, Physical Compiler, Power Compiler, PowerCODE, PowerGate, ProFPGA, Protocol Compiler, RoadRunner, Route Compiler, RTL Analyzer, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, SmartLicense, SmartModel Library, Source-Level Design, SWIFT, Synopsys EagleV, Test Compiler, TestGen, TetraMAX TenX, TimeTracker, Timing Annotator, Trace-On-Demand, TymeWare, VCS, VCS Express, VCSi, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

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

HSPICE Simulation and Analysis Manual, Release U-2003.03-PA, March 2003

Comments?
E-mail your comments about Synopsys
documentation to doc@synopsys.com

SYNOPSYS®

Table of Contents

Preface

This preface includes the following sections:

- What's New in This Release
- About This Manual
- Audience
- Conventions
- Customer Support

What's New in This Release

This section describes the new features, enhancements, and changes included in *Simulation and Analysis Manual* version 2003.03. Unless otherwise noted, you can find additional information about these changes later in this book.

New Features

See the *Release Notes* for information about new features and last-minute changes.

Known Limitations and Resolved D/Es

Information about known problems and limitations, as well as about resolved Defects and Enhancements (D/Es), is available in the *HSPICE Release Notes* in SolvNet. 2003.03 is the last release of HSPICE that uses the D/E terminology and numbering system; in future releases, these defects and enhancements will be referred to as Synopsys Technical Action Requests (STARs).

To see the *HSPICE Release Notes:*

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNet.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, click New Synopsys User Registration.)
3. Click Release Notes in the Main Navigation section, find the 2003.03 Release Notes, then open the *HSPICE Release Notes.*

Improved Documentation

To improve its usefulness, HSPICE documentation has been divided from the two thick manuals used in previous releases, into five smaller manuals in the 2003.03 release. A table in the *HSPICE Release Notes* maps the location of information from the old manuals, to the equivalent location in the new manual set*.*

About This Manual

The *Simulation and Analysis Manual* describes how to use HSPICE to simulate and analyze your circuit designs.

Audience

This manual is for logic designers and engineers who use Synopsys HSPICE for circuit simulation and analysis.

Conventions

This manual uses the following conventions.

Commands

SYNTAX:

command_name [*argument(s)*]

argument types: keyword **|** *value* **|** tag=*value* **|** tag=keyword

Command Argument	Definition
keyword	Keywords are identifiers that must be used as they appear. They are shown in base font.
value	Values are user-determined. They are shown in italic text to distinguish them from commands and keywords.
tag= *value* keyword	Tags can be followed by either a value or a keyword. Tags and keywords are in the base font. Argument values are in italics to distinguish them from commands, keywords, and tags.

Symbol	Definition
|	A pipe symbol (|) represents the word “or” and separates choices between two or more arguments.
...	An ellipsis (...) indicates that more than one argument can be specified. Ellipses are used only for multiple arguments with tags.
[ ]	Open and closed square brackets indicate that the enclosed argument is optional.
( )	Open and closed parenthesis indicate that there is a choice between the enclosed arguments (two or more). These are used only when a command has several groups of argument choices; multiple pipe symbols (|), in this case, would result in an ambiguous syntax.

SYNTAX:

report_node_i a[vg] **|** r[ms] **|** p[eak] **|** h[ist] *node_name(s)*

For this command, a[vg], r[ms], p[eak], and h[ist] are keywords–choose one of these. The *node_name(s)* are user-determined.

EXAMPLE:

```
report_node_i a VDD GND
```

In this example, a is a keyword, and *VDD* and *GND* are values.

SYNTAX:

report_node_ic [a[ll]] [q[uoted]] [for=epic **|** spice] [*time(s)*]

For this command, a[ll] and q[uoted] are optional keywords. The tag for= is part of an optional argument for which you can choose the epic or spice keyword. The *time(s)* are user-determined and, in this case, optional.

EXAMPLE:

```
report_node_ic all for=spice 1u
```

In this example, `all` is a keyword, `for=` is a tag, `spice` is a keyword, and `1u` is a value.

Menu Text, File Names, and Examples

Menu text appears in bold, as shown in the following example.

EXAMPLE 1:

To start setting up a run, select **File > Design Data Setup**.

File names are shown in the same font as the surrounding text, but in italics.

EXAMPLE 2:

This is an example of a *file_name.out* being shown in text.

Examples are shown in a courier font as they might appear on your screen. Each example is followed by an explanation. Anything shown in the example that appears in the explanation is shown in the same courier font used in the example.

EXAMPLE 3:

```
add_node_cap RS100 275
```

In the example, the capacitance value of node `RS100` is increased by `275` femtofarads.

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 including software downloads, documentation on the Web, and “Enter a Call With 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, click New Synopsys User Registration.)

If you need help using SolvNet, click SolvNet Help in the column on the left side of the SolvNet Web page.

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 (Synopsys user name and password required), then clicking “Enter a Call With the Support Center.”
- Send an e-mail message to support_center@synopsys.com.
- 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

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

This chapter explains the following topics:

- Applications
- Features
- Supported Platforms
- Simulation Structure

Applications

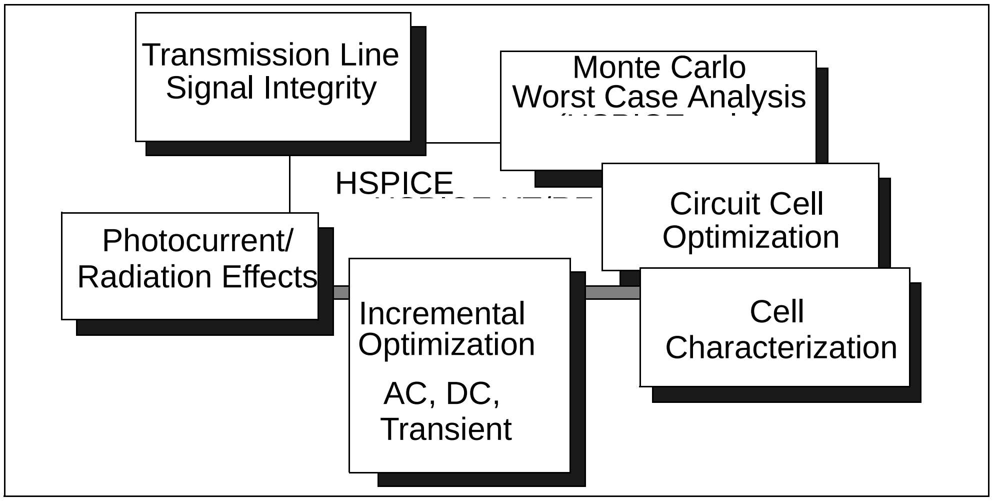
HSPICE is unequalled for fast, accurate circuit and behavioral simulation. They facilitate circuit-level analysis of performance and yield, using Monte Carlo, worst case, parametric sweep, and data-table sweep analysis, and employ the most reliable automatic-convergence capability.

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

The size of the circuits that HSPICE 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.

Features

Figure 1-1 Synopsys HSPICE Design Features



Synopsys HSPICE 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 calibrate models for high-level logic simulators.
- Geometric lossy-coupled transmission lines for PCB, multi-chip, package, and IC technologies.
- Discrete component, pin, package, and vendor IC libraries.
- AvanWaves interactively graphs and analyzes multiple simulation waveforms.
- If you suspend a simulation job, LSF license manager sends a signal to that job, and HSPICE releases the occupied license. Another simulation job can use that license, or the stopped job can reclaim the license and continue from where you stopped it. You can also submit simulation jobs with priority into the LSG queue; LSF automatically suspends low-priority simulation jobs, to run high-priority simulation jobs. When a high-priority job completes, LSF automatically releases the license back to the lower-priority job, which resumes from the point where LSF suspended it.

Figure 1-2 Synopsys HSPICE Circuit Analysis Types

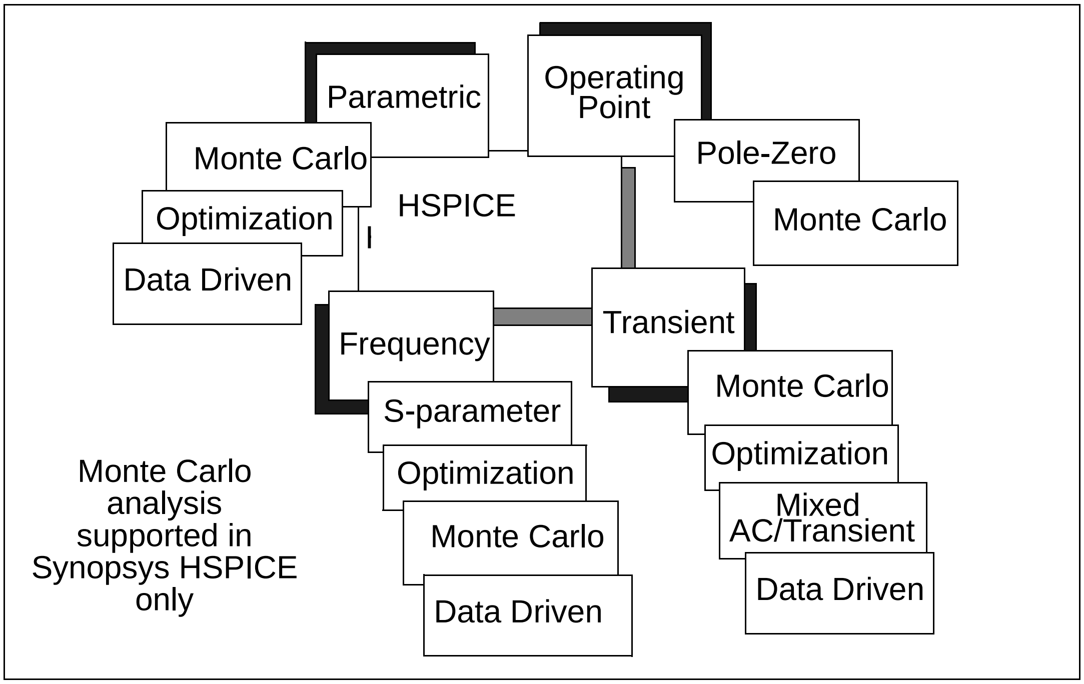
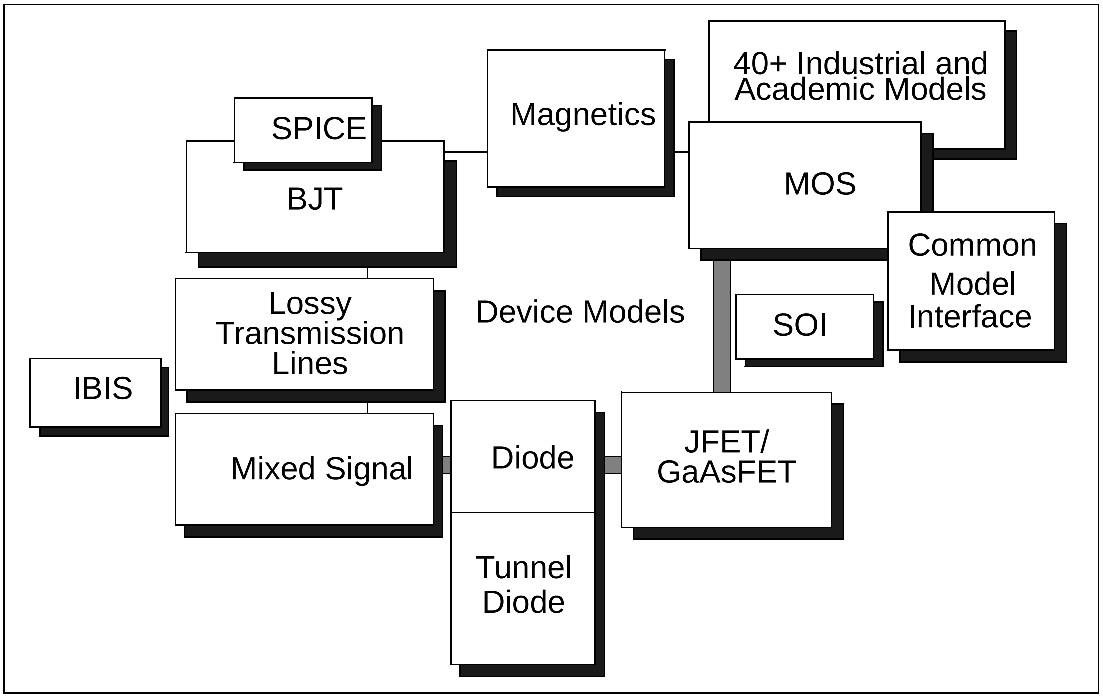


Figure 1-3 Synopsys HSPICE Modeling Technologies



Simulation at the integrated circuit level, and at the system level, requires 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.

Use the following HSPICE features to organize how simulation circuits and models run:

- Explicit include files – .INC statement.
- Implicit include files – .OPTION SEARCH = ‘lib_directory’.
- 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 statement.
- Statistical analysis – SWEEP MONTE analysis statement.
- Multiple alternative – .ALTER statement.
- Automatic measurements – .MEASURE statement.
- Condition-controlled netlists (IF-ELSEIF-ELSE-ENDIF statements).

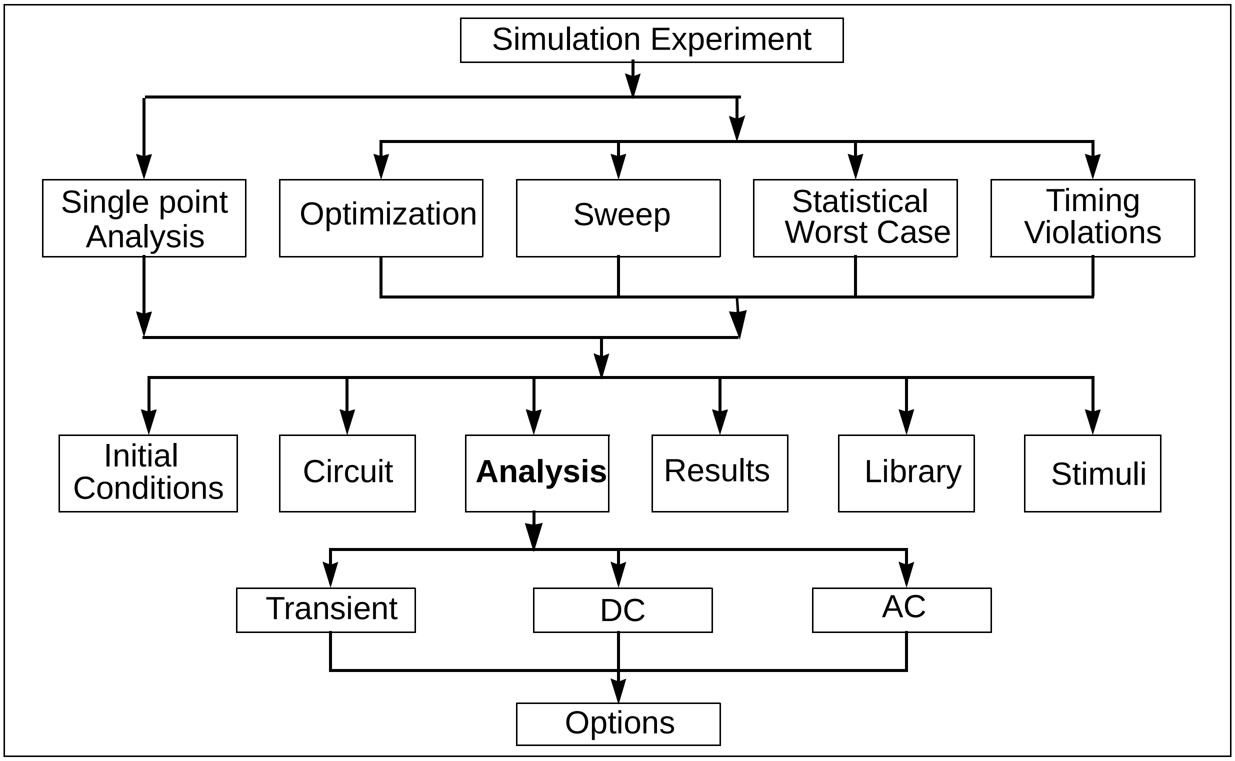
Supported Platforms

Table 1-1 HSPICE Supported Platforms

Platform	Operating System
Sun Ultra	Solaris 2.5.1, 2.7, and 2.8
Sun Blade	Solaris 2.8
HP PA	UX 10.20, UX 11.00
IBM RS6000	AIX 4.3 (4.3.3)
DEC Alpha	OSF 4.0
PC	Windows ME, 2000, NT 4.0, XP-Home, XP-Professional.
Linux	RedHat 6.2, 7.0, 7.1, and 7.2 (Does not support MOSFET level 29 or 45).
Note: HSPICE supports a single AMD CPU for WinNT4.0, and RedHat 7.2. HSPICE is a 32-bit executable.	

Simulation Structure

Figure 1-4 Simulation Program Structure



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

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 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.
- Element variation – Monte Carlo 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).

Data Flow

HSPICE 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 1-5 on page 1-9 shows the overall data flow.

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 executes the analyses specified in the input file.

3. HSPICE 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 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.

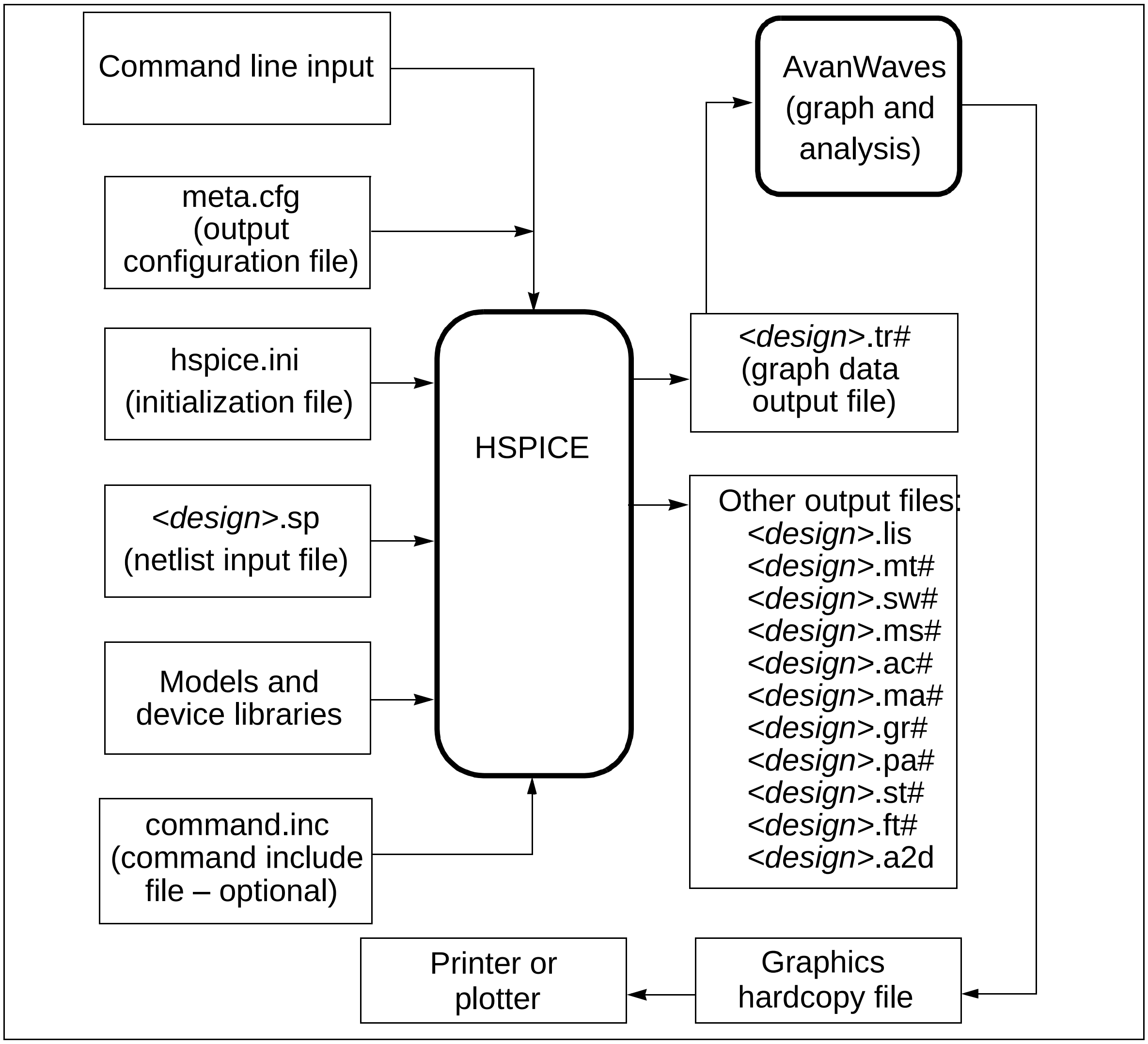
   HSPICE provides a complete set of print and plot variables for viewing analysis results.

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

You can specify whether the HSPICE program simulation output appears in an output listing file, or in a graph data file. HSPICE creates standard output files to describe initial conditions (*.ic* extension) and output status (*.st0* extension). In addition, HSPICE 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.

AvanWaves output display and analysis includes a graphical user interface. Use the mouse to select options, and to execute commands, in the AvanWaves windows. Refer to the *AvanWaves User Guide* for instructions about how to use AvanWaves.

Figure 1-5 Overview of Data Flow

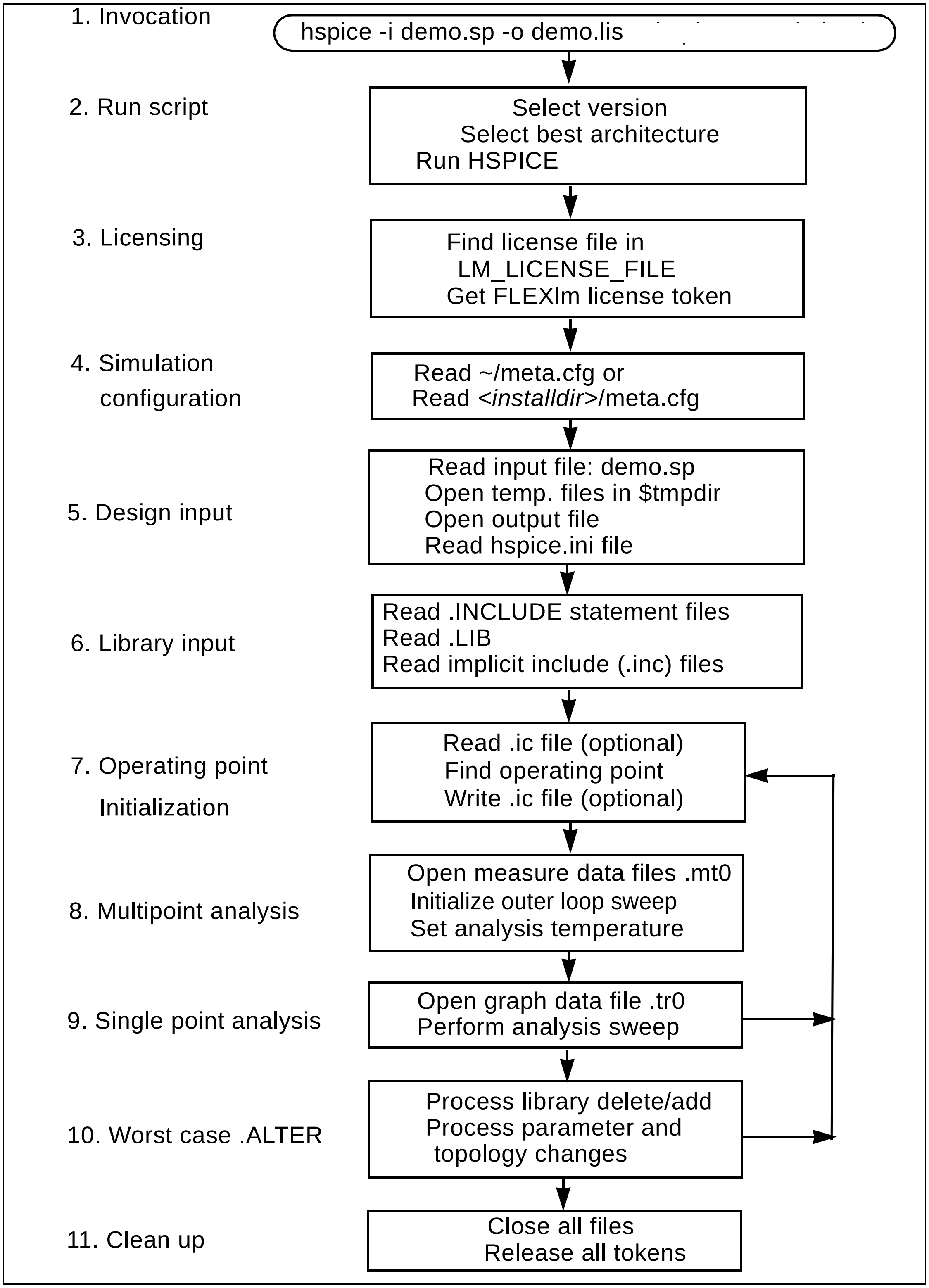


Command line input
meta.cfg (output configuration file)
hspice.ini (initialization file)
<design>.sp (netlist input file)
Models and device libraries
command.inc (command include file – optional)
HSPICE
AvanWaves (graph and analysis)
<design>.tr# (graph data output file)
Other output files:
<design>.lis
<design>.mt#
<design>.sw#
<design>.ms#
<design>.ac#
<design>.ma#
<design>.gr#
<design>.pa#
<design>.st#
<design>.ft#
<design>.a2d
Printer or plotter
Graphics hardcopy file

Simulation Process Overview

Figure 1-6 on page 1-10 shows the HSPICE simulation process. The following section summarizes the steps in a typical simulation.

Figure 1-6   Simulation Process



2

Setup for Simulation

This chapter describes the required setup steps, and background information that you should understand, before you run Synopsys HSPICE to perform IC circuit analyses.

This chapter includes the following examples:

- Setting Environment Variables
- Using Wildcards
- Netlist Overview

Setting Environment Variables

HSPICE 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 license file.

EXAMPLE:

If your HSPICE license file is in /usr/cad/hspice/license.dat path, then enter:

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

Using Wildcards

You can use wildcards to match node names. For more information about using wildcards in a configuration file, see Using Wildcards on page 2-2.

The following statements support wildcards:

- .PRINT
- .PROBE

EXAMPLE:

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

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

- The ? wildcard matches any single character. For example, 9? matches 92, 9a, 9A, and 9%.

- The * 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).
  - .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).

SYNTAX:

```
.PROBE wildcard_expression
```

The characters that formulate the *wildcard_expression* are:

Table 2-2 .PROBE Wildcard Syntax

Wildcard	**Description**
*	Matches any string of characters.
?	Matches any single character.
[]	Matches any character that 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 character.
any other character	Matches itself.

Wildcards must begin with a letter or a number. For example:

Table 2-3 Using Wildcards in .PROBE Statements

.probe v(*)	<--- correct format
.probe *	<--- incorrect format
.probe x*	<--- correct format

Examples

The following examples use wildcards with .PRINT and .PROBE statements. You must create an *.admrc* file to use these wildcards.

- Probe all top-level nodes.

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

- Probe all top-level nodes whose names start with a. For example: a1, a2, a3, a00, ayz.

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

- Print all first-level nodes, where zero-level are top-level nodes. For example: X1.A, X4.554, Xab.abc123.

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

- Probe all first-level nodes, where zero-level are top-level nodes. For example: x1.A, x4.554, xab.abc123. This example creates only waveform data, without an ASCII table of results.

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

- Print all second-level nodes. 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.

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

- Probe nodes that combine variables, with a specific range of possible digits. This example outputs the x*.00 through x*.99 nodes. The node name must be two characters long, and must be integers from 00 to 99, inclusive.

```
.PROBE v(x*.[0-9][0-9])
```

- Print all first-level nodes, where zero-level are top-level nodes, that are only two characters long. However, the first character must be either 1, 2, or 3, and the second character must be either a, b, or c. For example: xdd.1b, xdd.2c, xy.3a.

```
.PRINT v(x*.[123][abc])
```

- Print all second-level nodes that start with a, b, c, d, or e.

```
.PRINT v(*.*.[a-e]*)
```

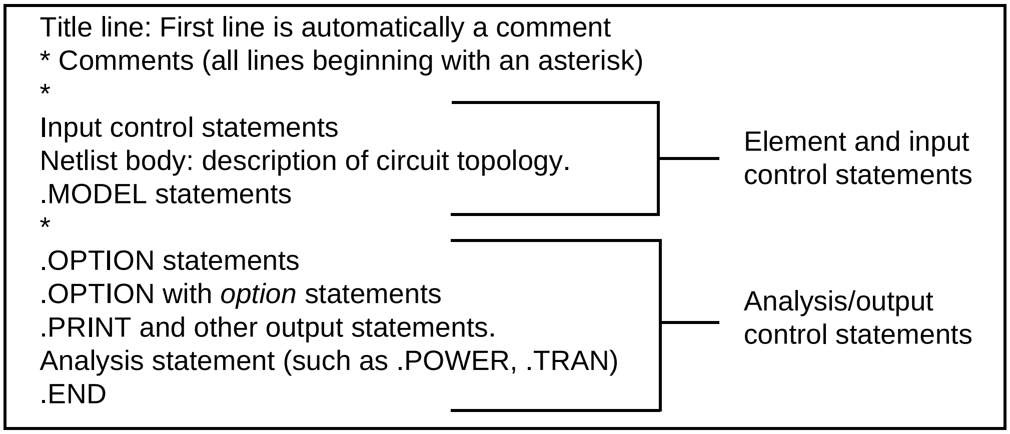
Netlist Overview

The circuit description syntax, for HSPICE, is compatible with the SPICE input netlist format.

Basic Structure

Figure 2-1 on page 2-5 shows the basic structure of an input netlist.

Figure 2-1 Basic Netlist Structure



Title line: First line is automatically a comment
* Comments (all lines beginning with an asterisk)
*
Input control statements
Netlist body: description of circuit topology.
.MODEL statements
*
.OPTION statements
.OPTION with *option* statements
.PRINT and other output statements.
Analysis statement (such as .POWER, .TRAN)
.END

Element and input control statements
Analysis/output control statements

EXAMPLE:

1. This example of a simple netlist file, called *inv_ckt.in*, shows a small inverter test case, which measures the timing behavior of the inverter. To create the circuit:
2. Define the MOSFET models for the PMOS and NMOS transistors of the inverter.
3. Insert the power supplies for both VDD and GND power rails.
4. 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
```

First Character

The first character in every line specifies how HSPICE interprets the remaining line. Table 2-4 lists and describes the valid characters.

Table 2-4 First Character Descriptions

Line	If the First Character is...	Indicates
First line of a netlist	Any character	Comment line
Subsequent lines of netlist, and all lines of included files	.	Netlist keyword
	c, C, d, D, e, E, f, F, g, G, h, H, i, I, k, K, l, L, m, M, q, Q, r, R, v, V	Element instantiation
	*	Comment line
	+	Continues previous line

The first line of a netlist is a comment, no matter what letter starts the line. The first line of an included file is a normal line, not a comment.

EXAMPLE:

The . character at the start of the line below, indicates that .TRAN is a keyword:

```
.TRAN 0.5ns 20ns
```

Adding Elements

Lines that add an instance of an element begin with a specific letter, as shown in Table 2-5 .

Table 2-5 Element Identifiers

Letter (First Character)	**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
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
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

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

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 (*) as the first character of the line, or a dollar sign ($) directly in front of the comment anywhere on the line.

- You can insert all comment text, after and including the `$`, anywhere in a line of code, not just at the beginning of a line (as required when you use *).
- $ is the preferred way to indicate comments, because of the flexibility of its placement within the code.
- Line continuations require + as the first character in the line that follows.

EXAMPLE:

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

* This line is a comment
.MODEL n1 NMOS $this is an example of an inline comment

* The following line is a continuation
+ LEVEL = 3
```

Subcircuit Node Names

To assign subcircuit node names, HSPICE traces the node, from the top level of the circuit, to the final subcircuit level. It then concatenates the different level names, using . as a delimiter.

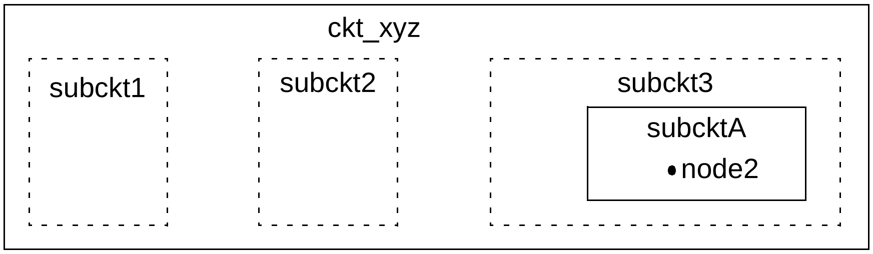
The last name must be a node name or an element.

EXAMPLE:

In Figure 2-2, the top-level (main) circuit is named *ckt_xyz*. This circuit contains three subcircuits: *subckt1*, *subckt2*, and *subckt3*. The *subckt3* subcircuit contains another subcircuit, named *subcktA*. To reference the *node2* node, specify the path as follows:

```
subckt3.subcktA.node2
```

Figure 2-2   Example Top-Level Circuit



Reserved Keywords

Do not use any of these keywords as parameter names or node names in your netlist.

EXAMPLE:

```
TIME
```

Reserved Operator Keywords

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 stops immediately.

Scale Factors for Numbers

Numbers can use any of the following formats:

- Integers.
- Floating-point values.
- A floating-point number, followed by an integer exponent (scientific notation).
- An integer, or a floating-point number, followed by one of the scale factors listed in Table 2-6 on page 2-11.

Table 2-6 Scale Factors

Scale Factor	Prefix	Multiplying Factor
T	tera	1e+12
G	giga	1e+9
MEG or X	mega	1e+6
K	kilo	1e+3
M	milli	1e-3
U	micro	1e-6
N	nano	1e-9
P	pico	1e-12
F	femto	1e-15
A	atto	1e-18

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

3

Simulation Input and Controls

This chapter describes the input requirements, methods of entering data, and Synopsys HSPICE statements used to enter input. This chapter explains the following topics:

- Using Netlist Input Files
- Input Netlist File Composition
- Using Subcircuits
- Discrete Device Libraries
- Using Standard Input Files
- Starting HSPICE
- Improving Simulation Performance with Multithreading
- HSPICE Output Files

Using Netlist Input Files

This section describes how to use standard netlist input files.

Input Netlist File Guidelines

HSPICE operates on an input netlist file, and stores results in either an output listing file or a graph data file. An input file, with the name *<design>.sp* (use this form for clarity), 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).

To generate input netlist and library input files, HSPICE 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, 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 issues an error message.

Input Line Format

- The input netlist file cannot be in a packed or compressed 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. See Delimiters on page 3-4.

- An input filename, statement, or equation can be up to 1024 characters long.
- HSPICE ignores differences between upper and lower case in input lines, except in quoted filenames.
- To continue a statement on the next line, in HSPICE, enter a plus (+) sign as the first non-numeric, non-blank character in the next line.
- To continue all HSPICE statements, including quoted strings (such as paths and algebraics), use a backslash (\) or a double backslash (\\) at the end of the line that you want to continue.

  - A single backslash preserves white space.
  - A double backslash squeezes out any white space between the continued lines. The double backslash guarantees that path names are joined without interruption.

  Input lines can be 1024 characters long, so you generally need to fold and continue a line only to improve readability.
- You can add comments anywhere in an HSPICE file. Lines that begin with an asterisk (*) are comments. To place a comment on the same line as input text, enter a dollar sign ($), preceded by one or more blanks, after the input text.
- If your input netlist file includes any special control characters, HSPICE reports an error. Most systems cannot print control characters, so the error message is ambiguous, because the error message cannot show the erroneous character. Use the .OPTION BADCHAR statement to locate such errors. The default for BADCHAR is off.

Names

- Names must begin with an alphabetic character, but thereafter can contain numbers and the following characters:

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

- Names are input tokens. Token delimiters must precede and follow these names. See Delimiters on page 3-4.
- Names can be 1024 characters long.
- Names are not case sensitive.

Delimiters

- An input token is any item in the input file that HSPICE 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.

Nodes

- Node identifiers can be up to 1024 characters long, including periods and extensions.
- Numerical node names are valid in the range of 0 through 9999999999999999 (1-1E16).
- HSPICE ignores leading zeros in node numbers.
- HSPICE ignores trailing characters in node numbers. For example, node 1A is the same as node 1. Exception: HSPICE recognizes the following special alphabetic trailing characters (a, d, e, f, g, i, k, m, n, o, p, t, u, x).
- A node name can begin with any of these characters: # _ ! %.
- To make nodes global across all subcircuits, use a .GLOBAL statement.
- The 0, GND, GND!, and GROUND node names all refer to the global HSPICE 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 (for example, M for a MOSFET element, D for a diode, R for a resistor, and so on), except in subcircuits.
- Subcircuit instance names begin with X. (Subcircuits are sometimes called macros or modules.)
- Instance names are limited to 1024 characters.
- .OPTION LENNAM defines the length of names in printouts (default = 8).

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.
- .OPTION PATHNUM controls whether the list files show full path names or path numbers.

Numbers

- You can enter numbers as integer or real.
- 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.
- .OPTION EXPMAX limits the exponent size.
- HSPICE interprets trailing alphabetic characters as units comments.
- HSPICE does not check units comments.

- .OPTION INGOLD controls the format of numbers in printouts.
- .OPTION NUMDGT = x controls the listing printout accuracy.
- .OPTION MEASDGT = x controls the measure file printout accuracy.
- .OPTION VFLOOR = x specifies the smallest voltage for which HSPICE prints the value. Smaller voltages print as 0.

Parameters and Expressions

- Parameter names in HSPICE 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 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 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 256 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.

Schematic Netlists

HSPICE typically uses netlisters to generate circuits from schematics, and accept either hierarchical or flat netlists. The normal SPICE netlisters flatten all subcircuits, and rename all nodes to numbers. Avoid flat netlisters if possible.

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 3-1 Input Netlist File Sections and Chapter References

Sections	Examples	Chapter	Definition
Title	.TITLE	3	The first line in the netlist is the title of the input netlist file.
Set-up	.OPTION	9	Sets conditions for simulation.
	.IC or .NODESET	10	Initial values in circuit and subcircuit.
	.PARAM	7	Set parameter values in the netlist.
	.GLOBAL	7	Set node name globally in netlist.
Sources	Sources and digital inputs	5	Sets input stimuli (I or V).
Netlist	Circuit elements	3-4	Circuit for simulation.
	.SUBKCT, .ENDS	3	Subcircuit definitions.
Analysis	.DC, .TRAN, .AC, etc.	10-12	Statements to perform analyses.
	.SAVE and .LOAD	10	Save and load operating point info.
	.DATA	3	Create table for data-driven analysis.
	.TEMP	3	Set analysis temperature.
Output	.PRINT, .PLOT, .GRAPH, .PROBE	8	Statements to output variables.
	.MEASURE	8	Statement to evaluate and report user-defined functions of a circuit.
Library, Model and File Inclusion	.INCLUDE	3	General include files.
	.MALIAS	3	Assigns an alias to a diode, BJT, JFET, or MOSFET.
	.MODEL	3, 8	Element model descriptions.
	.LIB	3	Library.
	.<UN>PROTECT	3	Control printback to output listing.

Table 3-1 Input Netlist File Sections and Chapter References (Continued)

Sections	**Examples**	**Chapter**	**Definition**
Alter blocks	.ALIAS	3	Renames a previous model.
	.ALTER	3	Sequence for in-line case analysis.
	.DELETE LIB	3	Removes previous library selection.
End of netlist	.END	3	Required statement; end of netlist.

Input Netlist File Composition

Title of Simulation and .TITLE Statement

You set the simulation title in the first line of the input file. HSPICE 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.

As shown in the first syntax below, to set the title, you can place a .TITLE statement on the first line of the netlist. However, HSPICE does not *require* the .TITLE syntax.

In the second form shown, the string is the first line of the input file. 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.

SYNTAX:

```
.TITLE <string_of_up_to_72_characters>
```

or

```
<string_of_up_to_72_characters>
```

Comments

An asterisk (`*`) as the first non-blank character, or an inline dollar sign (`$`) that is not the first character on the line, indicates a comment statement.

SYNTAX:

```
* <comment_on_a_line_by_itself>
```

or

```
<HSPICE_statement> $ <comment_following_HSPICE_input>
```

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
```

You can place comment statements anywhere in the circuit description. The * must be in the first space on the line.

The $ must be used for comments that do not begin at the first space on a line (for example, for comments that follow simulator input on the same line). The $ must be preceded by a space or comma, if it is not the first nonblank character. You can place the $ within node or element names.

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.

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

SYNTAX:

```
elname <node1 node2 ... nodeN> <mname>
+ <pname1 = val1> <pname2 = val2> <M = val>
```

or

```
elname <node1 node2 ... nodeN> <mname>
+ <pname = ’expression’> <M = val>
```

or

```
elname <node1 node2 ... nodeN> <mname>
+ <val1 val2 ... valn>
```

Table 3-2 Element Name Syntax

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 R Resistor S, T,U,W Transmission line V Voltage source X Subcircuit call
node1 ...	Node names identify the nodes that connect to the element. Node names must begin with a letter, followed by up to 1023 additional alphanumeric characters. You cannot use the following characters in node names: = ( ),’ <*space*>
mname	HSPICE 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 to the corresponding model node. The value can be a number or an algebraic expression.
M = val	Element multiplier. Replicates the *val* element times, in parallel. Do not assign a negative value or zero as the M value.

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).

.SUBCKT or .MACRO Statement

You can use .subckt and .macro statements in HSPICE.

SYNTAX:

```
.SUBCKT subnam n1 < n2 n3 …> < parnam = val …>
.ENDS
```

or

```
.MACRO subnam n1 < n2 n3 … > < parnam = val …>
.EOM
```

Table 3-3 .SUBCKT Syntax

Parameter	Description
subnam	Specifies a reference name for the subcircuit model call.
n1 …	Node numbers for external reference; cannot be the ground node (zero). Any element nodes that are in the subcircuit, but are not in this list, are strictly local, with three exceptions: 1. Ground node (zero). 2. Nodes assigned using BULK = node in MOSFET or BJT models. 3. Nodes assigned using the .GLOBAL statement.
parnam	A parameter name set to a value. Use only in the subcircuit. To override this value, assign it in the subcircuit call, or set a value in a .PARAM statement.

EXAMPLE:

```
*FILE SUB2.SP TEST OF SUBCIRCUITS
.OPTION LIST ACCT
   V1 1 0 1
.PARAM P5 = 5 P2 = 10
.SUBCKT SUB1 1 2 P4 = 4
   R1 1 0 P4
   R2 2 0 P5
   X1 1 2 SUB2 P6 = 7
   X2 1 2 SUB2
.ENDS
```

```
*
.MACRO SUB2 1 2 P6 = 11
   R1 1 2 P6
   R2 2 0 P2
.EOM
   X1 1 2 SUB1 P4 = 6
   X2 3 4 SUB1 P6 = 15
   X3 3 4 SUB2

*
.MODEL DA D CJA = CAJA CJP = CAJP VRB = -20 IS = 7.62E-18
+   PHI = .5 EXA = .5 EXP = .33
.PARAM CAJA = 2.535E-16 CAJP = 2.53E-16
.END
```

The preceding example defines two subcircuits: `SUB1` and `SUB2`. These are resistor divider networks, whose resistance values are parameters (variables). The X1, X2, and X3 statements call these subcircuits. Because the resistor values are different in each call, these three calls produce different subcircuits.

.ENDS or .EOM Statement

SYNTAX:

```
.ENDS <SUBNAM>
```

```
.EOM <SUBNAM>
```

EXAMPLE:

```
.ENDS OPAMP
.EOM MAC3
```

This statement terminates a subcircuit named *subname*. If you omit *subname*, this statement terminates all subcircuit definitions.

This statement must be the last for any subcircuit definition.

You can nest subcircuit references (calls) within subcircuits, in HSPICE.

Subcircuit Call Statement

SYNTAX:

```
Xyyy n1 <n2 n3 …> subnam <parnam = val …> <M = val>
```

Table 3-4 Subcircuit Call Syntax

Parameter	Description
Xyyy	Subcircuit element name. Must begin with an *X*, followed by up to 15 alphanumeric characters.
n1 …	Node names, for external reference.
subnam	Subcircuit model reference name.
parnam	A parameter name set to a value (val), for use only in the subcircuit. It overrides a parameter value in the subcircuit definition, but is overridden by a value set in a .PARAM statement.
M	Multiplier. Makes the subcircuit appear as *M* subcircuits in parallel. You can use this multiplier to characterize circuit loading. HSPICE does not need additional calculation time, to evaluate multiple subcircuits. Do not assign a negative value or zero as the M value.

EXAMPLE 1:

```
X1 2 4 17 31 MULTI WN = 100 LN = 5
```

The above example calls a subcircuit model named MULTI. It assigns the WN = 100 and LN = 5 parameters in the .SUBCKT statement (not shown). The subcircuit name is X1. All subcircuit names must begin with X.

EXAMPLE 2:

```
.SUBCKT YYY NODE1 NODE2 VCC = 5V
.IC NODEX = VCC
   R1 NODE1 NODEX 1
   R2 NODEX NODE2 1
.EOM
XYYY 5 6 YYY VCC = 3V
```

The preceding example defines a subcircuit named YYY. The subcircuit consists of two 1-ohm resistors in series. The .IC statement uses the VCC passed parameter to initialize the NODEX subcircuit node.

Note: If you initialize a non-existent subcircuit node, HSPICE generates a warning message. This can occur if you use an existing *.ic* file (initial conditions) to initialize a circuit that you modified since you created the *.ic* file.

Element and Node Naming Conventions

Node Names

Nodes are the points of connection between elements in the input netlist file. 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 ignores letters that follow numbers in node names. Node names must begin with a letter, followed by up to 1023 characters.

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

Table 3-5 Node Name Legal Characters

Symbol	**Definition**
+	plus sign
-	minus sign or hyphen
*	asterisk
/	slash
$	dollar sign
#	pound sign
[]	left and right square brackets
!	exclamation mark

Table 3-5   Node Name Legal Characters

Symbol	Definition
<>	left and right angle brackets
_	underscore
%	percent sign

If you use braces { } in node names, HSPICE changes them to brackets [ ]. You cannot use the following characters in node names:

Table 3-6   Node Name Illegal Characters

Symbol	Definition
(	left and right parentheses
,	comma
=	equal sign
‘	apostrophe
	blank space

Also, period (.) is reserved for use as a separator between the subcircuit name and the node name:

```
<subcircuitName>.<nodeName>
```

The sorting order for operating point nodes is:

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

Instance and Element Names

Element names in HSPICE 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 3-11).

Subcircuit Node Names

HSPICE assigns two subcircuit node names.

- To assign the first name, HSPICE 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.

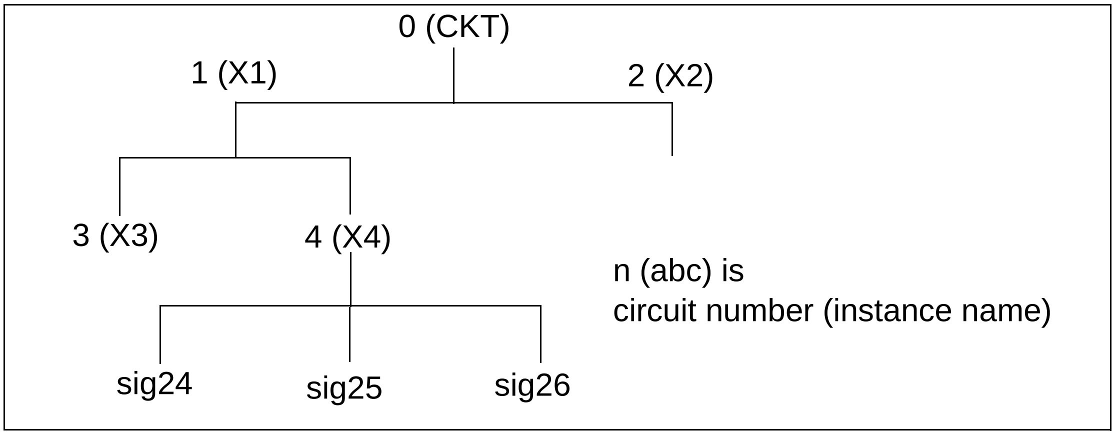
To indicate the ground node, use either the number `0`, the name `GND`, or !GND. Every node should have at least two connections, except for transmission line nodes (unterminated transmission lines are permitted) and MOSFET substrate nodes (which have two internal connections). Floating power supply nodes are terminated with a 1-megohm 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 3-1 on page 3-20.

Figure 3-1   Subcircuit Calling Tree, with Circuit Numbers and Instance Names



In Figure 3-1 on page 3-20, the path name of the *sig25* node in the X4 subcircuit is *X1.X4.sig25.* You can use this path in HSPICE 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`*`>`

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 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 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 automatically generates; this number appears in the output listing file.

.GLOBAL Statement

The .GLOBAL statement globally assigns a node name, in HSPICE. 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 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.

SYNTAX:

```
.GLOBAL node1 node2 node3 ...
```

In this syntax, node1 specifies global nodes, such as supply and clock names; overrides local subcircuit definitions.

EXAMPLE:

This example shows global definitions for VDD and input_sig nodes.

```
.GLOBAL VDD input_sig
```

.TEMP Statement

To specify the circuit temperature for a HSPICE 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 uses the difference between the circuit simulation temperature and the TNOM reference temperature to define derating factors for component values. For information about temperature analysis, see Temperature Analysis on page 12-6.

HSPICE permits only one temperature for the entire circuit.

SYNTAX:

```
.TEMP t1 <t2 <t3 ...>>
```

In this syntax, *t1 t2* are the temperatures, in °C, at which HSPICE simulates the circuit.

EXAMPLE 1:

```
.TEMP -55.0 25.0 125.0
```

The .TEMP statement sets the circuit temperatures for the entire circuit simulation. To simulate the circuit, using individual elements or model temperatures, HSPICE uses:

- Temperature, as set in the .TEMP statement.
- TNOM option setting (or the TREF model parameter).
- DTEMP element temperature.

EXAMPLE 2:

```
.TEMP 100
D1 N1 N2 DMOD DTEMP=30
D2 NA NC DMOD
R1 NP NN 100 TC1=1 DTEMP=-30

.MODEL DMOD D IS=1E-15 VJ=0.6 CJA=1.2E-13 CJP=1.3E-14
+ TREF=60.0
```

In this example:

- The .TEMP statement sets the circuit simulation temperature to 100°C.
- You do not specify TNOM, so it defaults to 25°C.
- The temperature of the diode is 30°C above the circuit temperature, as set in the DTEMP parameter.

That is:

- D1temp = 100°C + 30°C = 130°C.
- HSPICE simulates the D2 diode at 100°C.
- R1 simulates at 70°C.

Because the diode model statement specifies TREF at 60°C, HSPICE derates the specified model parameters by:

- 70×C (130×C - 60°C) for the D1 diode.
- 40×C (100×C - 60°C) for the D2 diode.
- 45×C (70°C - TNOM) for the R1 resistor.

.DATA Statement

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 full .DATA functionality.

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

Data-driven analysis syntax requires a .DATA statement, and an analysis statement that contains a DATA = dataname keyword.

You can use the .DATA statement to concatenate or column-laminate data sets, to optimize measured I-V, C-V, transient, or s-parameter data.

You can also use the .DATA statement for a first or second sweep variable, when you characterize cells, and test worst-case corners. Simulation reads data measured in a lab, such as transistor I-V data, one transistor at a time, in an outer analysis loop. Within the outer loop, the analysis reads data for each transistor (IDS curve, GDS curve, and so on), one curve at a time, in an inner analysis loop.

The .DATA statement specifies parameters that change values, and the sets of values to assign during each simulation. The required simulations run as an internal loop. This bypasses reading-in the netlist and setting-up the simulation, which saves computing time. In internal loop simulation, you can also plot simulation results against each other, and print them in a single output.

You can enter any number of parameters in a .DATA block.
The .AC, .DC, and .TRAN statements can use external and inline data provided in .DATA statements. The number of data values per line does not need to correspond to the number of parameters. For example, you do not need to enter 20 values on each line in the .DATA block, if each simulation pass requires 20 parameters: the program reads 20 values on each pass, no matter how you format the values.

Each .DATA statement can contain up to 50 parameters. If you need more than 50 parameters in a single .DATA statement, place 50 or fewer parameters in the .DATA statement, and use .ALTER statements for the remaining parameters.

HSPICE refers to .DATA statements by their datanames, so each dataname must be unique. HSPICE support three .DATA statement formats:

- Inline data
- Data concatenated from external files
- Data column laminated from external files

These formats are described below.

- The MER and LAM keywords tell HSPICE to expect external file data, rather than inline data.
- The FILE keyword denotes the external filename.

- You can use simple file names, such as out.dat, without the single or double quotes (‘ ’ or “ ”), but use the quotes when file names start with numbers, such as 1234.dat.
- File names are case sensitive on Unix systems.

For more details about using the .DATA statement in different types of analysis, see Chapter 8, “Simulation Options”, Chapter 9, “Initializing DC/Operating Point Analysis”, and Chapter 10, “Transient Analysis”.

SYNTAX:

Operating point:

```
.DC DATA = dataname
```

DC sweep:

```
.DC vin 1 5 .25 SWEEP DATA = dataname
```

AC sweep:

```
.AC dec 10 100 10meg SWEEP DATA = dataname
```

TRAN sweep:

```
.TRAN 1n 10n SWEEP DATA = dataname
```

For data-driven analysis, specify the start time (time 0) in the analysis statement, so analysis correctly calculates the stop time.

Inline .DATA Statement

Inline data is parameter data, listed in a .DATA statement block. The *datanm* parameter, in a .DC, .AC, or .TRAN analysis statement, calls this statement.

SYNTAX:

```
.DATA datanm pnam1 <pnam2 pnam3 ... pnamxxx >
+   pval1<pval2  pval3 ... pvalxxx>
+   pval1’ <pval2’ pval3’ ... pvalxxx’>
.ENDDATA
```

Table 3-7 .DATA Syntax

Parameter	Description
datanm	Specifies the data name, referenced in the .TRAN, .DC, or .AC statement.
pnami	Specifies the parameter names used for source value, element value, device size, model parameter value, and so on. You must declare these names in a .PARAM statement.
pvali	Specifies the parameter value.

The number of parameters that HSPICE reads, determines the number of columns of data. The physical number of data numbers per line does not need to correspond to the number of parameters. For example, if the simulation needs 20 parameters, you do not need 20 numbers per line.

EXAMPLE:

```
 .TRAN 1n  100n        SWEEP DATA= devinf
 .AC DEC 101hz  10khz  SWEEP DATA= devinf
 .DC TEMP-55125 10     SWEEP DATA= devinf

 .DATAdevinfwidth length  thresh   cap
  +50u   30u      1.2v    1.2pf
  + 25u  15u      1.0v    0.8pf
  + 5u   2u       0.7v    0.6pf
  + .... ...      ...     ...
.ENDDATA
```

HSPICE performs the above analyses for each set of parameter values defined in the .DATA statement. For example, the program first uses the width = 50u, length = 30u, thresh = 1.2v, and cap = 1.2pf parameters to perform .TRAN, .AC, and .DC analyses.

HSPICE then repeats the analyses for width = 25u, length = 15u, thresh = 1.0v, and cap = 0.8pf, and again for the values on each subsequent line in the .DATA block.

EXAMPLE:

The following is an example of .DATA as the inner sweep:

```
M1 1 2 3 0 N W = 50u L = LN
  VGS 2 0 0.0v
  VBS 3 0 VBS
  VDS 1 0 VDS
  .PARAM VDS = 0 VBS = 0 L = 1.0u
  .DC DATA = vdot
  .DATA vdot

   VBS   VDS      L
    0    0.1      1.5u
    0    0.1      1.0u
    0    0.1      0.8u
   -1    0.1      1.0u
   -2    0.1      1.0u
   -3    0.1      1.0u
    0    1.0      1.0u
    0    5.0      1.0u
.ENDDATA
```

The preceding example performs a DC sweep analysis for each set of VBS, VDS, and L parameters in the *.DATA vdot* block. That is, HSPICE runs eight DC analyses, one for each line of parameter values in the .DATA block.

EXAMPLE:

The following is an example of .DATA as an outer sweep:

```
 .PARAM W1 = 50u W2 = 50u L = 1u CAP = 0
 .TRAN 1n 100n SWEEP DATA = d1
 .DATA d1
   W1     W2    L      CAP
   50u    40u   1.0u   1.2pf
   25u    20u   0.8u   0.9pf
.ENDDATA
```

In the previous example:

- The default start time for the .TRAN analysis is 0.
- The time increment is 1 ns.
- The stop time is 100 ns.

This results in transient analyses at every time value from 0 to 100 ns, in steps of 1 ns, using the first set of parameter values in the *.DATA d1* block. Then HSPICE reads the next set of parameter values, and performs another 100 transient analyses. It sweeps time from 0 to 100 ns, in 1 ns steps. The outer sweep is time, and the inner sweep varies the parameter values. HSPICE performs two hundred analyses: 100 time increments, times 2 sets of parameter values.

External File .DATA Statement

SYNTAX:

The following is the syntax for concatenated data files:

```
.DATA datanm MER
  FILE = 'filename1' pname1 = colnum
+ <pname2 = colnum ...>
  <FILE = 'filename2' pname1 = colnum
+ <pname2 = colnum ...>>
...
<OUT = 'fileout'>
.ENDDATA
```

Table 3-8 .DATA Syntax in External File

Parameter	**Description**
datanm	Data name, referred to in the .TRAN, .DC or .AC statement.
MER	Specifies concatenated (series merging) data files to use.
filenamei	Data file to read. HSPICE concatenates files in the order they appear in the .DATA statement. You can specify up to 10 files.

Table 3-8 .DATA Syntax in External File (Continued)

Parameter	Description
pnami	Parameter names, used for source value, element value, device size, model parameter value, and so on. Declare these names in a .PARAM statement.
colnum	Specifies the column number in the data file, for the parameter value. The column does not need to be the same between files.
fileouti	Data file name, where simulation writes concatenated data. This file contains the full syntax for an inline .DATA statement, and can replace the .DATA statement that created it in the netlist. You can output the file, and use it to generate one data file from many.

Concatenated data files are files with the same number of columns, placed one after another. For example, if you concatenate the three files (A, B, and C).

```
File AFile BFile C
```

```
a a ab b bc c c
a a ab b bc c c
a a a
```

The data appears as follows:

```
a a a
a a a
a a a
b b b
b b b
c c c
c c c
```

Note: The number of lines (rows) of data in each file does not need to be the same. The simulator assumes that the associated parameter of each column of the A file is the same as each column of the other files.

EXAMPLE:

```
.DATA inputdata MER
   FILE = ‘file1’ p1 = 1 p2 = 3 p3 = 4
   FILE = ‘file2’ p1 = 1
   FILE = ‘file3’
.ENDDATA
```

The above listing concatenates *file1*, *file2*, and *file3*, to form the *inputdata* dataset. The data in *file1* is at the top of the file, followed by the data in *file2*, and *file3*. The *inputdata* in the .DATA statement references the dataname specified in either the .DC, .AC, or .TRAN analysis statements. The parameter fields specify the column that contains the parameters (you must already have defined the parameter names in .PARAM statements). For example, the values for the p1 parameter are in column 1 of *file1* and *file2*. The values for the p2 parameter are in column 3 of *file1*.

For data files with fewer columns than others, HSPICE assigns values of zero to the missing parameters.

Column Laminated .DATA Statement

SYNTAX:

```
.DATA datanm LAM
   FILE = ’filename1’ pname1 = colnum
+  <panme2 = colnum ...>
   <FILE = ’filename2’ pname1 = colnum
+  <pname2 = colnum ...>>
...
   <OUT = ’fileout’>
.ENDDATA
```

Table 3-9 Column-Laminated .DATA Syntax

Parameter	**Description**
datanm	Data name, referred to in the .TRAN, .DC or .AC statement.
LAM	Column-laminated (parallel merging) data files to use.
filenamei	Name of a data file to read. HSPICE concatenates files in the order listed in the .DATA statement. Specify up to 10 files.
pnami	Parameter names used for source value, element value, device size, model parameter value, and so on. Declare these names in a .PARAM statement.
colnum	Column number in the data file, that contains the parameter value. The column does not need to be the same between files.

Table 3-9 Column-Laminated .DATA Syntax (Continued)

Parameter	Description
fileouti	Name of the data file, where HSPICE writes concatenated data. This file contains the complete syntax for an inline .DATA statement, and can replace the .DATA statement that created it. You can output the file, and generate one data file from many.

Column lamination means that the columns of files with the same number of rows, are arranged side-by-side.

EXAMPLE:

Three files (*D*, *E*, and *F*) contain the following columns of data:

```
File DFile E File F

d1 d2 d3e4 e5f6
d1 d2 d3e4 e5f6
d1 d2 d3e4 e5f6
```

The laminated data appears as follows:

```
d1 d2 d3e4 e5f6
d1 d2 d3e4 e5f6
d1 d2 d3e4 e5f6
```

The number of columns of data does not need to be the same in the three files.

Note: The number of lines (rows) of data in each file does not need to be the same. HSPICE interprets missing data points as zero.

EXAMPLE:

```
.DATA dataname LAM
   FILE = ‘file1’ p1 = 1 p2 = 2 p3 = 3
   FILE = ‘file2’ p4 = 1 p5 = 2
   OUT = ‘fileout’
.ENDDATA
```

This listing laminates columns from *file1*, and *file2*, into the *fileout* output file. Columns one, two, and three of *file1*, and columns one and two of *file2*, are designated as the columns to place in the output file. You can specify up to 10 files per .DATA statement.

Note: If you run HSPICE on a different machine than the one on which the input data files reside (such as when you work over a network), use full path names instead of aliases. Aliases might have different definitions on different machines.

.INCLUDE Statement

SYNTAX:

```
.INCLUDE ‘<filepath> filename’
```

Table 3-10 .INCLUDE Syntax

Parameter	**Description**
filepath	Path name of a file, for computer operating systems that support tree-structured directories. A .INC file can contain nested .INC calls to itself, or to another .INC file. If you use a relative path in a nested .INC call, the path starts from the directory of the parent .INC file, not from the work directory. If the path starts from the work directory, HSPICE can also find the .INC file, but prints a warning.
filename	Name of a file to include in the data file. The file path, plus the file name, can be up to 1024 characters long. You can use any valid file name for the computer’s operating system. You *must* enclose the file path and name in single or double quotation marks.

.MODEL Statement

SYNTAX:

```
.MODEL mname type <VERSION = version_number>
+ <pname1 = val1 pname2 = val2 ...>
```

Table 3-11 .MODEL Syntax

mname	Model name reference. Elements must use this name to refer to the model. If model names contain periods (.), the automatic model selector might fail.
type	Selects a model type. Must be one of the following. AMP operational amplifier model C capacitor model CORE magnetic core model D diode model L inductor model or magnetic core mutual inductor model NJF n-channel JFET model NMOS n-channel MOSFET model NPN npn BJT model OPT optimization model PJF p-channel JFET model PLOT plot model for the .GRAPH statement PMOS p-channel MOSFET model PNP pnp BJT model R resistor model U lossy transmission line model (lumped) W lossy transmission line model SP S parameter
pname1 ...	Parameter name. Assign a model parameter name (*pname1*) from the parameter names for the appropriate model type. Each model section provides default values. For legibility, enclose the parameter assignment list in parentheses, and use either blanks or commas to separate each assignment. Use a plus sign (+) to start a continuation line.
VERSION	HSPICE version number. Allows portability of the BSIM (LEVEL=13) and BSIM2 (LEVEL = 39) models, between HSPICE releases. HSPICE release numbers, and the corresponding version numbers, are: HSPICE *release* *Version number* 9007B 9007.02 9007D 9007.04 92A 92.01 92B 92.02 93A 93.01 93A.02 93.02 95.3 95.3 96.1 96.1

Table 3-11 .MODEL Syntax (Continued)

	The VERSION parameter is valid only for LEVEL 13 and LEVEL 39 models. Use it with HSPICE Release H93A.02 and higher. If you use the parameter with any other model, or with a release before H93A.02, HSPICE issues a warning, but the simulation continues.You can also use VERSION to denote the BSIM3v3 version number only, in model LEVELs 49 and 53. For LEVELs 49 and 53, the HSPVER parameter denotes the HSPICE release number.

EXAMPLE:

```
.MODEL MOD1 NPN BF=50 IS=1E-13 VBF=50 AREA=2 PJ=3,
+ N=1.05
```

.LIB Call and Definition Statements

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 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.

.LIB Library Call Statement

SYNTAX:

```
.LIB ‘<filepath> filename’ entryname
```

Table 3-12 .LIB Syntax

Parameter	Description
filepath	Path to a file. Used where a computer supports tree-structured directories. When the `LIB` file (or alias) is in the same directory where you run HSPICE, you do not need to specify a directory path; the netlist runs on any machine. Use the “../” syntax in the filepath, to designate the parent directory of the current directory.
entryname	Entry name, for the section of the library file to include. The first character of an entryname cannot be an integer.

Table 3-12 .LIB Syntax (Continued)

Parameter	Description
filename	Name of a file to include in the data file. The combination of filepath plus filename can be up to 256 characters long, structured as any filename that is valid for the computer’s operating system. Enclose the file path and file name in single or double quotation marks. Use the “../” syntax in the filename, to designate the parent directory of the current directory.

EXAMPLE:

```
.LIB ’MODELS’ cmos1
```

.LIB Library File Definition Statement

To build libraries, use the .LIB statement in a library file. For each macro in a library, use a library definition statement (.LIB entryname) and an .ENDL statement. The .LIB statement begins the library macro, and the .ENDL statement ends the library macro.

SYNTAX:

```
.LIB entryname1
. $ ANY VALID SET OF HSPICE STATEMENTS
.ENDL entryname1
.LIB entryname2
.
. $ ANY VALID SET OF HSPICE STATEMENTS
.ENDL entryname2
.LIB entryname3
.
. $ ANY VALID SET OF HSPICE STATEMENTS
.ENDL entryname3
```

The text after a library file entry name must consist of HSPICE statements.

.LIB Nested Library Calls

Library calls can call other libraries, if they are different files.

EXAMPLE:

Below are an illegal example and a legal example for the *file3* library.

Illegal:

```
.LIB MOS7
...
.LIB ’file3’ MOS7 $ This call is illegal in MOS7 library
...
...
.ENDL
```

Legal:

```
.LIB MOS7
...
.LIB ’file1’ MOS8
.LIB ’file2’ MOS9
.LIB CTT $ file2 is already open for the CTT entry point
.ENDL
```

You can nest library calls to any depth. Use nesting with the .ALTER statement, to create a sequence of model runs. Each run can consist of similar components, using different model parameters, without duplicating the entire input file.

Library Building Rules

1. 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.

2. A library *cannot* contain a call to a library of its own entry name, within the same library file.
3. A HSPICE library cannot contain the .END statement.
4. .ALTER processing cannot change .LIB statements, within a file that an .INCLUDE statement calls.

The simulator uses the .LIB statement and the .INCLUDE statement, to access the models and skew parameters. The library contains parameters that modify .MODEL statements. The example below is a .LIB of model skew parameters, and features both worst-case and statistical distribution data. The statistical distribution median value is the default, for all non-Monte Carlo analysis.

EXAMPLE:

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY
$ PROCESS: 1.0U CMOS, FAB7
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN
.PARAM TOX = AGAUSS(200,20,3)$ 200 angstrom +/- 20a
+ XL = AGAUSS(0.1u,0.13u,3)$ polysilicon CD
+ DELVTON = AGAUSS(0.0,.2V,3)$ n-ch threshold change
+ DELVTOP = AGAUSS(0.0,.15V,3)$ p-ch threshold change
.INC ‘/usr/meta/lib/cmos1_mod.dat’$ model include file
.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES
.PARAM TOX = 220 XL = -0.03 DELVTON = -.2V
+ DELVTOP = -0.15V
.INC ‘/usr/meta/lib/cmos1_mod.dat’$ model include file
.ENDL FF
```

The model is in the /usr/meta/lib/cmos1_mod.dat include file.

```
.MODEL NCH NMOS LEVEL = 2 XL = XL TOX = TOX
+ DELVTO = DELVTON .....

.MODEL PCH PMOS LEVEL = 2 XL = XL TOX = TOX
+ DELVTO = DELVTOP .....
```

The model keyword (left side) equates to the skew parameter (right side). A model keyword can be the same as a skew parameter.

.OPTION SEARCH Statement

Use this statement to automatically access a library.

SYNTAX:

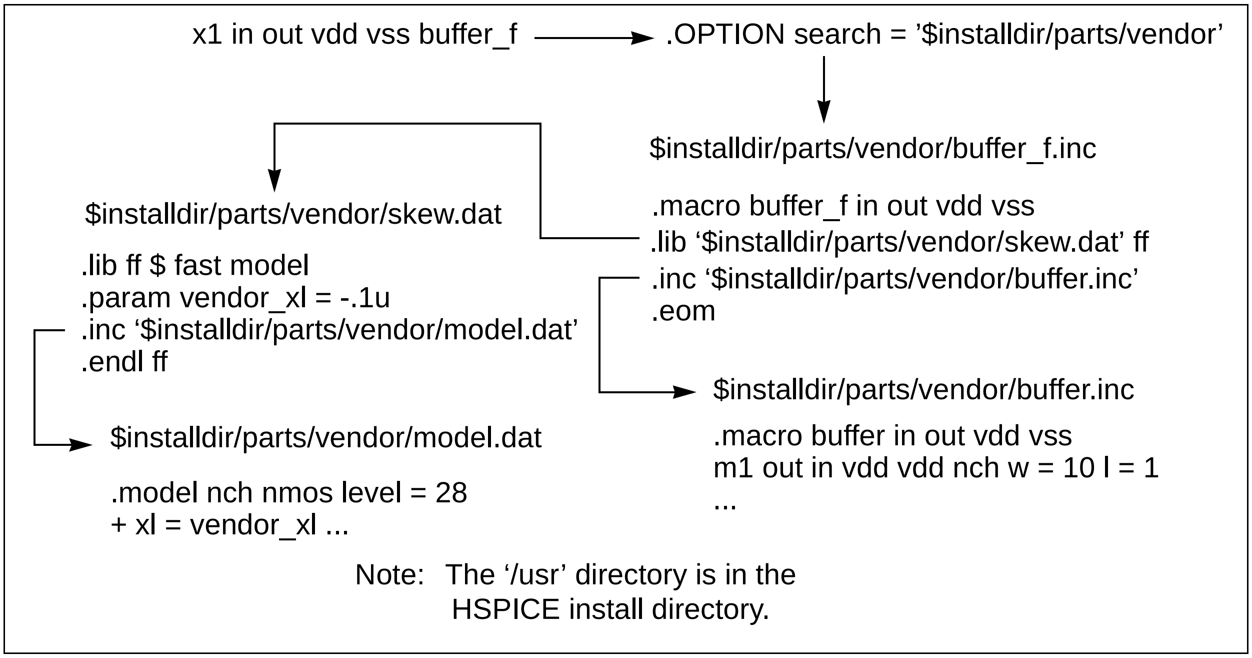
```
.OPTION SEARCH = ‘directory_path’
```

EXAMPLE:

```
.OPTION SEARCH = ‘$installdir/parts/vendor’
```

The preceding example searches for models in the *vendor* subdirectory, under the *$installdir/parts* installation directory (see Figure 3-2). The *parts/* directory contains the DDL subdirectories.

Figure 3-2   Vendor Library Usage



```
x1 in out vdd vss buffer_f
.OPTION search = ’$installdir/parts/vendor’

$installdir/parts/vendor/buffer_f.inc
.macro buffer_f in out vdd vss
.lib ‘$installdir/parts/vendor/skew.dat’ ff
.inc ‘$installdir/parts/vendor/buffer.inc’
.eom

$installdir/parts/vendor/skew.dat
.lib ff $ fast model
.param vendor_xl = -.1u
.inc ‘$installdir/parts/vendor/model.dat’
.endl ff

$installdir/parts/vendor/buffer.inc
.macro buffer in out vdd vss
m1 out in vdd vdd nch w = 10 l = 1
...

$installdir/parts/vendor/model.dat
.model nch nmos level = 28
+ xl = vendor_xl ...
```

Note: The ‘/usr’ directory is in the HSPICE install directory.

Automatic Library Selection

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

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/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.

.PARAM Statement

The .PARAM statement defines parameters. Parameters in HSPICE are names that have associated numeric values. You can use any of the following methods to define parameters:

- Simple Parameter Assignments
- Algebraic Parameter (Equation)
- User-Defined Function
- Subcircuit Default Definition
- Predefined Analysis
- Measurement Parameters

Simple Parameter Assignments

A simple parameter assignment is a constant real number. The parameter keeps this 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.

SYNTAX:

```
.PARAM <ParamName>=<RealNumber>
```

Algebraic Parameter (Equation)

To assign algebraic parameters, use an algebraic expression of real values, a predefined or user-defined function, or circuit or model values. Enclose a complex expression in single quotes to invoke the algebraic processor, *unless* the expression begins with an alphabetic character and contains no spaces. A simple expression consists of a single parameter name.

SYNTAX:

```
.PARAM <ParamName>='<AlgebraicExpression>'
.PARAM <ParamName1>=<ParamName2>
```

To use an algebraic expression as an output variable in a .PRINT, .PLOT, or .PROBE statement, use the PAR keyword.

EXAMPLE:

```
.PRINT DC v(3) gain=PAR(‘v(3)/v(2)’)
+ PAR(‘V(4)/V(2)’)
```

EXAMPLE:

```
.para x=cos(2)+sin(2)
```

User-Defined Function

A user-defined function assignment is similar to the definition of an algebraic parameter. HSPICE extends the algebraic parameter definition to include function parameters, used in the algebraic that defines the function. You can nest user-defined functions up to three deep.

SYNTAX:

```
.PARAM <ParamName>(<pv1>[<pv2>])=’<Expression>’
```

Subcircuit Default Definition

When you use hierarchical subcircuits, you can pick default values for circuit elements. You can use this feature in cell definitions, to simulate the circuit with typical values.

SYNTAX:

```
.SUBCKT <SubName><PinList>[<SubDefaultsList>]
```

The SubDefaultsList is:

```
<SubParam1>=<Expression>[<SubParam1>=<Expression>...]
```

Predefined Analysis

HSPICE provides several specialized analysis types, that require a way to control the analysis. For the syntax of these uses of .PARAM, see .PARAM Distribution Function on page 12-16.

HSPICE supports the following predefined analysis parameters:

- Temperature functions (*fn*)
- Optimization guess/range
- frequency
- time
- Monte Carlo functions

Measurement Parameters

.MEASURE statements produce 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 7-42.

.PROTECT Statement

The .PROTECT statement keeps models and cell libraries private.

- The .PROTECT statement suppresses printing text from the list file, such as when you use the BRIEF option.
- The .UNPROTECT command restores normal output functions.
- Any elements and models located between a .PROTECT and an .UNPROTECT statement, inhibit the element and model listing from the LIST option.

- The .OPTION NODE nodal cross reference, and the .OP operating point printout, do not list any nodes that are contained within the .PROTECT and .UNPROTECT statements.

SYNTAX:

```
.PROTECT
```

.UNPROTECT Statement

In HSPICE, the .UNPROTECT statement restores normal output functions that a .PROTECT statement restricted.

- Any elements and models located between .PROTECT and .UNPROTECT statements, inhibit the element and model listing from the LIST option.
- Neither the .OPTION NODE cross reference, nor the .OP operating point printout, list any nodes within the .PROTECT and .UNPROTECT statements.

SYNTAX:

```
.UNPROTECT
```

.ALTER Statement

You can use the .ALTER statement to rerun a HSPICE simulation, using different parameters and data.

Use parameter (variable) values for print and plot statements, before you alter them. The .ALTER block cannot include .PRINT, .PLOT, .GRAPH or any other input/output statements. You can include analysis statements (.DC, .AC, .TRAN, .FOUR, .DISTO, .PZ, and so on) in a .ALTER block in an input netlist file.

However, if you change only the analysis type, and you do not change the circuit itself, then simulation runs faster if you specify all analysis types in one block, instead of using separate .ALTER blocks for each analysis type.

The .ALTER sequence or block can contain:

- Element statements (except source elements)
- .DATA statements
- .DEL LIB statements
- .INCLUDE statements
- .IC (initial condition) and .NODESET statements
- .LIB statements
- .MODEL statements
- .OP statements
- .OPTION statements
- .PARAM statements
- .TEMP statements
- .TF statements
- .TRAN, .DC, and .AC statements
- .ALIAS statements

Altering Design Variables and Subcircuits

The following rules apply when you alter design variables and subcircuits in HSPICE.

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Each .ALTER simulation run prints only the actual altered input. A special .ALTER title identifies the run.
7. .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 Statements

1. 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.

2. 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.
3. HSPICE then uses these statements to modify the input netlist file.
4. HSPICE then resimulates the circuit.
5. For each additional .ALTER statement, HSPICE performs the simulation that precedes the first .ALTER statement.
6. 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 a .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.

SYNTAX:

```
.ALTER <title_string>
```

The *title_string* is any string up to 72 characters. HSPICE prints the appropriate title string for each .ALTER run, in each section heading of the output listing, and in the graph data (.tr#) files.

.ALIAS Statement

As listed in the previous section, you can use .alter statements to rename a model, to rename a library containing a model, or to delete an entire library of models in HSPICE. If your netlist references the old model name, then after you use one of these types of .alter statements, HSPICE no longer finds this model.

For example, if you use .DEL LIB in the .ALTER block to delete a library, the .ALTER command deletes all models in this library. If your netlist references one or more models in the deleted library, then HSPICE no longer finds the models.

To resolve this issue, HSPICE provides a .ALIAS command, to let you alias the old model name to another model name that HSPICE can find in the existing model libraries.

For example, you might delete a library named poweramp, that contains a model named pa1. Another library might contain an equivalent model named par1. You can then alias the pa1 model name to the par1 model name:

```
.alias pa1 par1
```

During simulation, when HSPICE encounters a model named pa1 in your netlist, it initially cannot find this model, because you used a .ALTER statement to delete the library that contained this model. However, the .ALIAS statement indicates to use the par1 model, in place of the old pa1 model. HSPICE *does* find this new model in another library, so simulation continues.

You must specify an old model name and a new model name to use in its place. You cannot use .ALIAS without any model names:

```
.ALIAS
```

or with only *one* model name:

```
.ALIAS pa1
```

You also cannot alias a model name to *more than one* model name, because then the simulator would not know which of these new models to use in place of the deleted or renamed model:

```
.ALIAS pa1 par1 par2
```

For the same reason, you cannot alias a model name to a second model name, and then alias the second model name to a third model name:

```
.ALIAS pa1 par1
.ALIAS par1 par2
```

If your netlist does not contain a .ALTER command, and if the .ALIAS does not report a usage error, then the .ALIAS does not affect the simulation results.

EXAMPLE:

Your netlist might contain the statement:

```
.ALIAS myfet nfet
```

Without a .ALTER statement, HSPICE does not use *nfet* to replace *myfet* during simulation.

If your netlist contains one or more .ALTER commands, the first simulation uses the original *myfet* model. After the first simulation, if the netlist references *myfet* from a deleted library, .ALIAS substitutes *nfet* in place of the missing model.

- If HSPICE finds model definitions for both *myfet* and *nfet*, it reports an error and aborts.
- If HSPICE finds a model definition for *myfet*, but not for *nfet*, it reports a warning, and simulation continues, using the original *myfet* model.
- If HSPICE finds a model definition for *nfet*, but not for *myfet*, it reports a *replacement successful* message.

.MALIAS Statement

You can use the .MALIAS statement to assign an alias (another name) to a diode, BJT, JFET, or MOSFET model that you defined in a .MODEL statement.

The syntax of the .MALIAS statement is:

```
.MALIAS model_name=alias_name1 <alias_name2 ...>
```

- *model_name* is the model name defined in the .model card.
- *alias_name1*... is the alias that an instance (element) of the model references.

EXAMPLE:

```
*file: test malias statement
.OPTION acct tnom=50 list gmin=1e-14 post
.temp 0.0 25
.tran .1 2
vdd 2 0 pwl 0 -1 1 1
d1 2 1 zend dtemp=25
d2 1 0 zen dtemp=25
* malias statements
.malias zendef = zen zend
* model definition
.model zendef d (vj=.8 is=1e-16 ibv=1e-9 bv=6.0 rs=10
+ tt=0.11n n=1.0 eg=1.11 m=.5 cjo=1pf tref=50)
.end
```

- *zendef* is a diode model
- *zen* and *zend* are its aliases.
- The *zendef* model points to both the *zen* and *zend* aliases.

.malias differs from .alias in two ways:

- The alias in an .alias statement is defined in a .model card, but the model card does not define the alias in a .malias statement.
- The .alias command works only if you include .alter in the netlist. You can use .malias without .alter.

.CONNECT Statement

This statement connects 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.

SYNTAX:

```
.connect node1 node2
```

Table 3-13 .CONNECT Syntax

Parameter	**Description**
node1	Name of the first of two nodes to connect to each other.
node2	Name of the second of two nodes to connect to each other. The first node replaces this node in the simulation.

If you connect node2 to node1, HSPICE does not recognize *node2* at all. To apply any HSPICE statement to *node2*, apply it to *node1* instead. Then, to change the netlist construction to recognize *node2*, use a .alter statement.

EXAMPLE:

```
*example for  .connect
vcc 0 cc 5v
r1  0 1  5k
r2  1 cc 5k
.tran 1n 10n
.print i(vcc) v(1)
.alter
.connect cc 1
.end
```

The first .tran simulation includes two resistors. Later simulations have only one resistor, because r2 is shorted by connecting cc with 1. v(1) does not print out, but v(cc) prints out instead.

Use multiple .connect statements to connect several nodes together.

EXAMPLE:

```
.connect node1 node2
.connect node2 node3
```

connects both *node2* and *node3* to *node1*. All connected nodes must be in the same subcircuit, or all in the main circuit. The first HSPICE simulation evaluates only *node1*; *node2*, and *node3* are the same node as *node1*. Use .alter statements to simulate *node2* and *node3*.

If you set .option node, HSPICE prints out a node connection table.

.DEL LIB Statement

Use the .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 the .DEL LIB statement with the .ALTER statement.

SYNTAX:

```
.DEL LIB ‘<filepath>filename’ entryname
.DEL LIB libnumber entryname
```

Table 3-14 .DEL LIB Syntax

Parameter	**Description**
entryname	Entry name, used in the library call statement to delete.
filename	Name of a file to delete from the data file. The file path, plus the file name, can be up to 64 characters long. You can use any file name that is valid for the operating system that you use. Enclose the file path and file name in single or double quote marks.
filepath	Path name of a file, if the operating system supports tree-structured directories.
libnumber	Library number, used in the library call statement to delete.

EXAMPLE 1:

This example uses an .ALTER block.

```
FILE1: ALTER1 TEST CMOS INVERTER
 .OPTION ACCT LIST
 .TEMP 125
 .PARAM WVAL = 15U VDD = 5
 *
 .OP
 .DC VIN 0 5 0.1
 .PLOT DC V(3) V(2)
 *
 VDD 1 0 VDD
 VIN 2 0
 *
 M1 3 2 1 1 P 6U 15U
 M2 3 2 0 0 N 6U W = WVAL
 *

.LIB ’MOS.LIB’ NORMAL
.ALTER
 .DEL LIB ’MOS.LIB’ NORMAL     $removes LIB from memory

$PROTECTION
.PROT                          $protect statements below .PROT
 .LIB ’MOS.LIB’ FAST           $get fast model library

.UNPROT
 .ALTER
 .OPTION NOMOD OPTS            $suppress printing model parameters
 *                             and print the option summary
 .TEMP -50 0 50                $run with different temperatures
 .PARAM WVAL = 100U VDD = 5.5  $change the parameters
 VDD 1 0 5.5                   $using VDD 1 0 5.5 to change the
                               $power supply VDD value doesn’t
                               $work
 VIN 2 0 PWL 0NS 0 2NS 5 4NS 0 5NS 5
                               $change the input source
 .OP VOL                       $node voltage table of operating
                               $points
 .TRAN 1NS 5NS                 $run with transient also
 M2 3 2 0 0 N 6U WVAL          $change channel width
 .MEAS SW2 TRIG V(3) VAL = 2.5 RISE = 1 TARG V(3)
 + VAL = VDD CROSS = 2         $measure output
 *
.END
```

Example 1 calculates a DC transfer function for a CMOS inverter.

1. First, HSPICE simulates the device, using the NORMAL inverter model from the MOS.LIB library.

2. Using the .ALTER block and the .LIB command, HSPICE substitutes a faster CMOS inverter, FAST, for NORMAL.
3. HSPICE then resimulates the circuit.
4. Using the second .ALTER block, HSPICE executes DC transfer analysis simulations at three different temperatures, and with an n-channel width of 100 mm, instead of 15 mm.
5. HSPICE also runs a transient analysis, in the second .ALTER block. Use the .MEASURE statement to measure the rise time of the inverter.

EXAMPLE 2:

This example uses an .ALTER block.

```
FILE2: ALTER2.SP CMOS INVERTER USING SUBCIRCUIT
.OPTION LIST ACCT
.MACRO INV 1 2 3
M1 3 2 1 1 P 6U 15U
M2 3 2 0 0 N 6U 8U
.LIB ’MOS.LIB’ NORMAL
.EOM INV
XINV 1 2 3 INV
VDD 1 0 5
VIN 2 0
.DC VIN 0 5 0. 1
.PLOT V(3) V(2)
.ALTER
.DEL LIB ’MOS.LIB’ NORMAL
.TF V(3) VIN                    $DC small-signal transfer function
*
.MACRO INV 1 2 3                $change data within subcircuit def
M1 4 2 1 1 P 100U 100U          $change channel length,width,also
                                $topology
M2 4 2 0 0 N 6U   8U            $change topology
R4 4 3 100                      $add the new element
C3 3 0 10P                      $add the new element
.LIB ’MOS.LIB’ SLOW             $set slow model library
$.INC ’MOS2.DAT’                $not allowed to be used inside
                                $subcircuit allowed outside
                                $subcircuit
.EOM INV
.END
```

In this example, the .ALTER block adds a resistor and capacitor network to the circuit. The network connects to the output of the inverter, and HSPICE simulates a DC small-signal transfer function.

.END Statement

An .END statement must be the last statement in the input netlist file. The period preceding END is a required part of the statement.

Any text that follows the .END statement is a comment, and has no effect on that 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.

SYNTAX:

```
.END <comment>
```

EXAMPLE:

```
MOS OUTPUT
 .OPTION NODE NOPAGE
 VDS 3 0
 VGS 2 0
 M1 1 2 0 0 MOD1 L = 4U W = 6U AD = 10P AS = 10P
 .MODEL MOD1 NMOS VTO = -2 NSUB = 1.0E15 TOX = 1000 UO = 550
 VIDS 3 1
 .DC   VDS 0 10 0.5    VGS 0 5 1
 .PRINT DC I(M1) V(2)
.END MOS OUTPUT

MOS CAPS
 .OPTION SCALE = 1U SCALM = 1U WL ACCT
 .OP
 .TRAN .1 6
 V1 1 0 PWL 0 -1.5V 6 4.5V
 V2 2 0 1.5VOLTS
 MODN1 2 1 0 0 M 10 3

 .MODEL M NMOS VTO = 1 NSUB = 1E15 TOX = 1000 UO = 800
LEVEL = 1
 + CAPOP = 2

 .PLOT TRAN V(1) (0,5) LX18(M1) LX19(M1) LX20(M1) (0,6E-13)
.END MOS CAPS
```

Condition-Controlled Netlists (IF-ELSE)

```
.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 (condition3)
 <statement_block3>
]
.endif
```

You can use this IF-ELSE structure to change the circuit topology, expand the circuit, set parameter values for each device instance, or select different model cards in each IF-ELSE block.

- 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 analysis and output statements. The exceptions are:

  .end, .alter, .subckt, .ends, .macro, .eom, .global, .del, .mailias, . 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, but you cannot include subcircuits in IF-ELSEIF-ELSE statements.
- However, 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 updates the parameter value only after it selects the IF-ELSE block.
- You can nest IF-ELSE blocks.
- 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 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
```

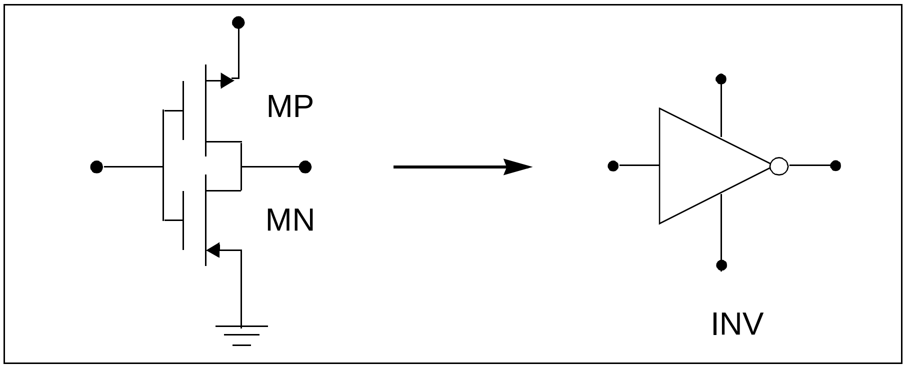
Using Subcircuits

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

- 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 provides a simple method for naming subcircuit nodes and elements: use the subcircuit call name as a prefix to the node or element name.

Figure 3-3   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

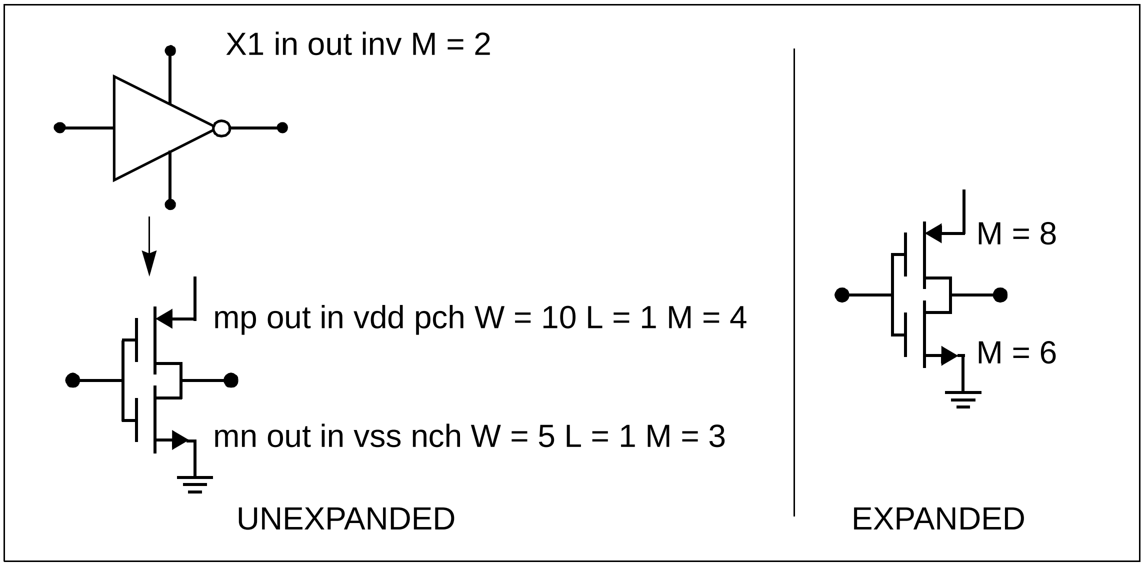
M (Multiply) Parameter

The most basic HSPICE subcircuit parameter 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:

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

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

Figure 3-4 Hierarchical Parameters Simplify Flip-flop Initialization


X1 in out inv M = 2
mp out in vdd pch W = 10 L = 1 M = 4
mn out in vss nch W = 5 L = 1 M = 3
UNEXPANDED
M = 8
M = 6
EXPANDED

EXAMPLE:

```
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
```

S (Scale) Parameter

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

SYNTAX:

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

The option hier_scale statement defines how HSPICE 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 sub-circuit. The specified S scale is relative to this default scale of the sub-circuit.

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

EXAMPLE:

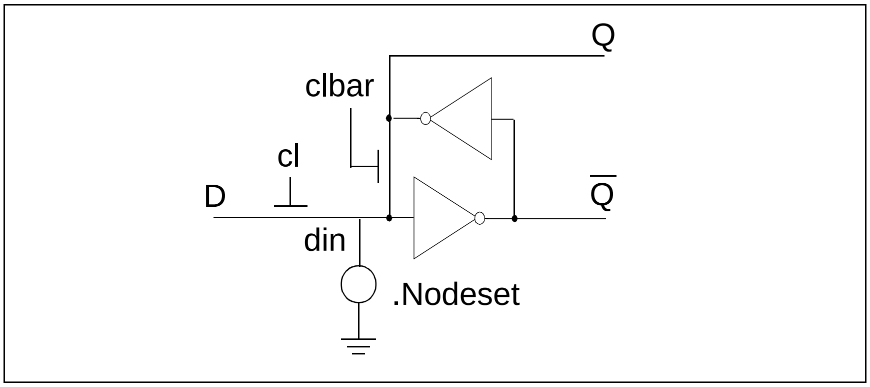
```
x1 a y inv S=1u
subckt inv in out
x2 a b kk S=1m
.ends
```

In this example:

1. HSPICE scales the X1 sub-circuit by the first S scaling value, 1u*(SCALE).
2. Because scaling is cumulative, X2 (a sub-circuit of X1) is then scaled, in effect, by the S scaling values of *both* X1 and X2:

   ```
   1m*1u*(SCALE)
   ```

Figure 3-5   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. To specify subcircuit nodes in .PRINT or .PLOT statements, specify the full subcircuit path and node name.

Undefined Subcircuit Search

If a subcircuit call is in a data file that does not describe the subcircuit, HSPICE automatically searches the:

1. Present directory for the file.
2. Directories specified in any .OPTION SEARCH = “*directory_path_name*” statement.
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.

Discrete Device Libraries

The Synopsys Discrete Device Library (DDL) is a collection of True-Hspice device models of discrete components, which you can use with HSPICE. 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 has a built-in op-amp model generator.

Note: The value of the *$installdir* environment variable is the path name to the directory where you installed HSPICE. This installation directory contains subdirectories, such as */parts* and */bin*. It also contains certain files, such as a prototype *meta.cfg* file, and the HSPICE 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.

EXAMPLE:

The following element statement creates an instance of the 1N4004 diode model:

```
X1 2 1 D1N4004
```

D1N4004 is the model name. See Element and Source Statements on page 3-11 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 True-Hspice circuit-level models, simulation automatically compensates for the effects of supply voltage, loading, and temperature.

HSPICE accesses DDL models in several ways:

1. The installation script creates an hspice.ini initialization file.
2. HSPICE 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 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.
3. Set .OPTION SEARCH = ‘<*library_path*>’ in the input netlist.

   Use this method to list the personal libraries to search. HSPICE first searches the default libraries referenced in the *hspice.ini* file, then searches libraries in the order listed in the input file.
4. 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.

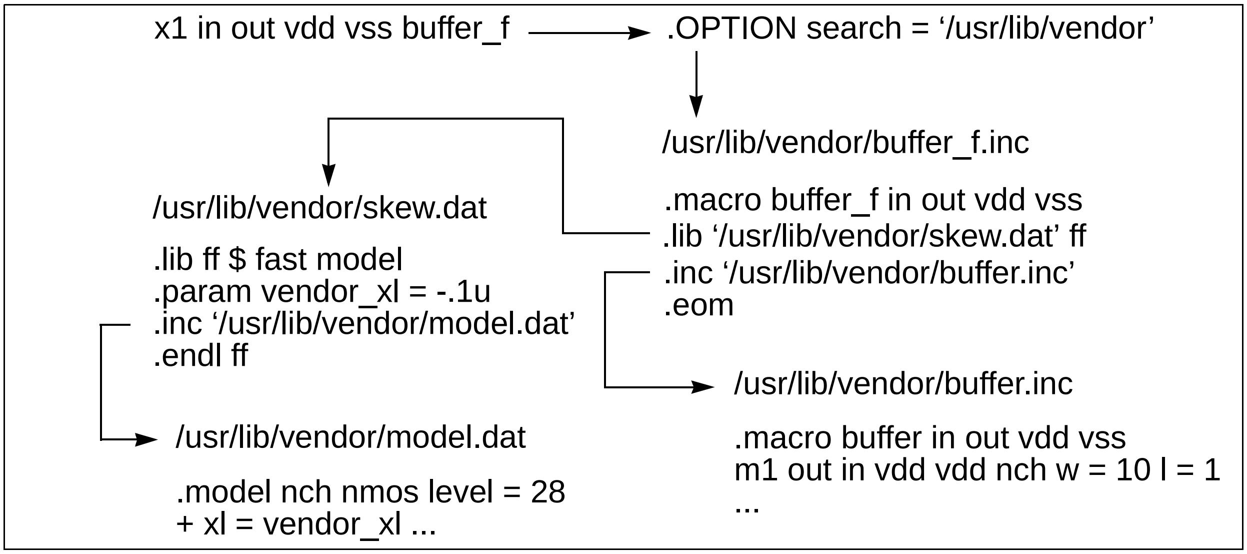
Vendor Libraries

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

- 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 3-6   Vendor Library Usage



```
x1 in out vdd vss buffer_f
```

```
.OPTION search = ‘/usr/lib/vendor’
```

/usr/lib/vendor/buffer_f.inc

```
.macro buffer_f in out vdd vss
.lib ‘/usr/lib/vendor/skew.dat’ ff
.inc ‘/usr/lib/vendor/buffer.inc’
.eom
```

/usr/lib/vendor/skew.dat

```
.lib ff $ fast model
.param vendor_xl = -.1u
.inc ‘/usr/lib/vendor/model.dat’
.endl ff
```

/usr/lib/vendor/buffer.inc

```
.macro buffer in out vdd vss
m1 out in vdd vdd nch w = 10 l = 1
...
```

/usr/lib/vendor/model.dat

```
.model nch nmos level = 28
+ xl = vendor_xl ...
```

Subcircuit Library Structure

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

The HSPICE component naming conventions for each subcircuit is:

```
<subname>.inc
```

Store the subcircuit in a directory that is accessible through the .OPTION SEARCH = ‘<*libdir*>’ 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.

Using Standard Input Files

This section describes how to use standard input files in HSPICE.

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 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 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 standard output on Unix platforms, and you can redirect it.

Table 3-15 on page 3-66 lists input file types, and their standard names. The following sections describe these files.

Table 3-15 Input Files

Input File Type	File Name
Output configuration file	meta.cfg
Initialization file	hspice.ini
DC operating point initial conditions file	<*design*>.ic#
Input netlist file	<*design*>.sp
Library input file	<*library_name*>
Analog transition data file	<*design*>.d2a

Configuration File (*meta.cfg*)

This file sets up the printer, plotter, and terminal. It includes a line, default_include = file name, which sets up a path to the default .ini file (for example, hspice.ini).

The default_include file name is case-sensitive (except for the PC and Windows versions of HSPICE).

Initialization File (*hspice.ini*)

Specify user defaults in an hspice.ini initialization file. If the run directory contains an hspice.ini file, HSPICE includes its contents at the top of the input file.

To include initialization files, you can define DEFAULT_INCLUDE = <*filename*> in the system, or in a meta.cfg file.

You can use an initialization file to set options (in an .OPTION statement) and to access libraries, as the Synopsys installation procedure does.

DC Operating Point Initial Conditions File (<design>.ic#)

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

The .SAVE statement creates a <*design*>.ic# file. A subsequent .LOAD statement initializes the circuit to the DC operating point values, specified in the <*design*>.ic# file.

Starting HSPICE

Use the following syntax to start HSPICE:

```
hspice <-i> <path/>input_file <-o path/output_file>
+  <-n number> <-html<path/html_file>> <-b>
```

Table 3-16 HSPICE Startup Syntax

Parameter	Description
input_file	Specifies the input netlist file name, for which an extension `<.ext>` is optional. If you do not specify an input filename extension in the command, HSPICE searches for the `<input_file>.sp` file. Precede the input file with -i. HSPICE uses the input filename as the root for the output files. Star- Hspice also checks for an initial conditions file (`.ic`) that has the input file root name. The following is an example of an input file name: `/usr/sim/work/rb_design.sp` In this file name: • /usr/sim/work/ is the directory path to the design. • rb_design is the design root name. • .sp is the filename suffix.
-n	Specifies the starting number for numbering output data file revisions (output_file.tr#, output_file.ac#, output_file.sw#, where # is the revision number).

Table 3-17 lists available HSPICE command arguments..

Table 3-17 HSPICE Command Options

Option	Description
-b	Batch processing switch, for PC platforms only.
-html <*path/*>html_file	Specifies an HTML output file. If you do not specify a path, HSPICE saves the HTML output file in the same directory that you specified in the `-o` option. If you do not specify the `-o` option, HSPICE saves the HTML output in the running directory.
-i <*input_file*>	Name of the input netlist file. If you do not enter an extension, HSPICE assumes *.sp*.
-n <*number*>	Revision number at which to start numbering .gr#, .tr#, and other output files. By default, file numbers start at zero: .gr0, .tr0, and so on. Use this option to specify the number (-n 7 for .gr7, .tr7, for example).
-o <*output_file*>	Name of the output file. If you do not specify an extension, HSPICE assigns .lis.

You do not need to include a filename extension in the output file specification. HSPICE names it *output_file.lis*. In output file names, everything up to the final period is the root filename, and everything after the last period is the filename extension.

- If you either do not use the -o option, or you use the -o option without pointing to a filename, then HSPICE uses the output root file name specified in the -html option.
- If you do not specify an output file name in either the -o or -html option, then HSPICE uses the input root filename as the output file root filename.
- If you include the .lis extension in the filename that you enter using -o, then HSPICE does not append another .lis extension to the root filename of the output file.
- If you do not specify an output file, HSPICE directs output to the terminal.

Use the following syntax to redirect the output to a file, instead of to the terminal:

```
hspice input_file <-n number> > output_file
```

EXAMPLE:

```
hspice demo.sp -n 7 > demo.out
```

Table 3-18 HSPICE Syntax for Output Files

Parameter	**Description**
demo.sp	Input netlist file; the .sp extension to the filename is optional.
-n 7	Starts the output data file revision numbers at 7: demo.tr7, demo.ac7, and demo.sw7
>	Redirects the program output listing to demo.out

Executing a Simulation

Perform these steps to execute a HSPICE simulation.

1. Invocation.

   To invoke HSPICE, use a Unix command such as:

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

   This command invokes the HSPICE shell, and uses an input netlist file named demo.sp, and an output listing file named demo.out. The & at the end of the command invokes HSPICE in the background, so that you can still use the window and keyboard while HSPICE runs.

2. Script execution.

   The HSPICE shell 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. If the input netlist file does not exist, a no input data error appears in the output listing file.

   HSPICE opens three scratch files in the /tmp directory. To change this directory, reset the TMPDIR environment variable in the HSPICE command script.

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

   An example of a simple HSPICE input netlist is:

```
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 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 that it opened, and releases all license tokens.

Interactive Simulation

To invoke HSPICE in *interactive* mode, enter:

```
hspice -I
```

You can then use other HSPICE commands to help you simulate circuits interactively. You can also use the `help` command for detailed information about a command.

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

```
hspice -I -L scriptifile.cmd
```

Sample HSPICE Commands

The following are some additional examples of HSPICE commands.

- hspice -i demo.sp

  demo is the root filename. Output files are named demo.lis, demo.tr0, demo.st0, and demo.ic.

- hspice -i demo.sp -o demo

  demo is the output file root name (designated with the -o option). Output files are named demo.lis, demo.tr0, demo.st0, and demo.ic.

- hspice -i rbdir/demo.sp

  demo is the root name. HSPICE writes the demo.lis, demo.tr0, and demo.st0 output files into the directory where you executed the HSPICE command. It also writes the demo.ic output file into the same directory as the input source—that is, rbdir.

- hspice -i a.b.sp

  a.b is the root name. The output files are ./a.b.lis, ./a.b.tr0, ./a.b.st0, and ./a.b.ic.

- hspice -i a.b -o d.e

  a.b is the root name for the input file.

  d.e is the root for output file names, except for the .ic file, to which HSPICE assigns the a.b input file root name. The output files are d.e.lis, d.e.tr0, d.e.st0, and a.b.ic.

- hspice -i a.b.sp -o outdir/d.e

  a.b is the root for the .ic file. HSPICE writes the .ic file into a file named a.b.ic.

  d.e is the root for other output files. Output files are outdir/d.e.lis, outdir/d.e.tr0, and outdir/d.e.st0.

- hspice -i indir/a.b.sp -o outdir/d.e.lis

  a.b is the root for the .ic file. HSPICE writes the .ic file into a file named indir/a.b.ic.

  d.e is the root for the output files.

- hspice test.sp -o test.lis -html test.html

  This command creates output file in both .lis and .html format, after simulating the test.sp input netlist.

- hspice test.sp -html test.html

  This command creates only a .html output file, after simulating the test.sp input netlist.

- hspice test.sp -o test.lis

  This command creates only a .lis output file, after simulating the test.sp input netlist.

- hspice -i test.sp -o -html outdir/a.html

  This command creates output files in both .lis and .html format. Both files are in the outdir directory, and their root filename is a.

- hspice -i test.sp -o out1/a.lis -html out2/b.html

  This command creates output files in both .lis and .html format. The .lis file is in the out1 directory, and its root filename is a. The .html file is in the out2 directory, and its root filename is b.

Improving Simulation Performance with Multithreading

HSPICE simulations include device model evaluations and matrix solutions. You can run model evaluations concurrently on multiple CPUs, 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. Using multithreading speeds-up simulations, with no loss of accuracy.

Multithreaded (MT) HSPICE is supported on Sun Solaris 2.5.1 (SunOS 5.5.1), Sun Solaris 2.7 (SunOS 5.7), Sun Solaris 2.8 (SunOS 5.8), HP-UX 11.0, PC/RedHat Linux 7.0, PC/RedHat Linux 7.1, Windows NT, Windows 2000, and Windows XP.

Multithreading improves simulation speed, especially for circuit designs that contain many MOSFET, JFET, diode, or BJT models in the netlist.

Running HSPICE-MT

To run HSPICE-MT, use the following syntax:

```
hspice_mt -mt #num -i input_filename -o output_filename
```

- If you omit the #num, or if the #num that you specify is larger than the number of online CPUs, then HSPICE sets the number of threads to the number of online CPUs.
- If you omit the -o output_file option, HSPICE prints the result to the standard output.

Under 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 online CPUs. Under the Synopsys HSPUI (HSPICE User Interface):

1. Select the correct hsp_mt.exe version in the Version Combo Box.
2. Select the correct number of processors 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 multithreaded HSPICE, the CPU time is:

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

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:

```
T1cpu/(0.2T1cpu + 0.8T1cpu/2cpus) = 1.67
```

The typical Tparallel value is 0.6 to 0.7, for moderate to large circuits.

HSPICE Output Files

HSPICE produces various types of output files, listed in Table 3-19.

Table 3-19 HSPICE Output Files and Suffixes

Output File Type	**Extension**
Output listing	*.lis*, or user-specified
Transient analysis results	.tr# †
Transient analysis measurement results	.mt#
DC analysis results	.sw# †
DC analysis measurement results	.ms#
AC analysis results	.ac# †
AC analysis measurement results	.ma#
Hardcopy graph data (from *meta.cfg* PRTDEFAULT)	.gr# ††
Digital output	.a2d
FFT analysis graph data	.ft#†††
Subcircuit cross-listing	.pa#

Table 3-19 HSPICE Output Files and Suffixes (Continued)

Output File Type	Extension
Output status	.st#
Operating point node voltages (initial conditions)	.ic#

# Either a sweep number, or a hardcopy file number.

† Created only if you use .POST to generate graphical data.

†† 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.

††† Created only if you use a .FFT statement.

The files are listed in Table 3-19 on page 3-76 and described below.

Output listing can appear as output_file (no file extension), output_file.lis, or with a file extension that you specify, depending on which format you use to start the simulation. Output_file is the output file specification, not including any extension. This file includes the following information:

- Name of the simulator used.
- Version of the HSPICE simulator used.
- Synopsys message block.
- Input file name.
- User name.
- License details.
- Copy of the input netlist file.
- Node count.
- Operating point parameters.

- Details of the volt drop, current, and power for each source and subcircuit.
- Low-resolution plots, originating from the .PLOT statement.
- Results of .PRINT statement.
- Results of .OPTION statements.

HSPICE places *transient analysis results* in output_file. tr#, where # is 0-9 or a-z, and follows the -n 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.

The output file is in proprietary binary format if POST = 0 or 1, or in ASCII format if POST = 2. You can also use the explicit expressions POST = BINARY, or POST=ASCII.

HSPICE writes *transient analysis measurement results* to output_file.mt#. The .MEASURE TRAN statement creates this output file.

DC analysis results appear in output_file.sw#, which a .DC statement produces. 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.

If the input file includes a .MEASURE DC statement, the output_file. ms# file specifies the *DC analysis measurement results*.

HSPICE places *AC analysis results* in output_file.ac#. These results list the output variables as a function of frequency, according to your specifications following the .AC statement.

If the input file contains a .MEASURE AC statement, then output_file.ma# contains *AC analysis measurement results*.

HSPICE places *hardcopy graph data* in output_file.gr#, which a .GRAPH statement produces. It 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.

Digital output contains data that the A2D conversion option of the U element converted to digital form.

FFT analysis graph data contains the graphical data needed to display the FFT analysis waveforms.

If the input netlist includes subcircuits, HSPICE automatically generates the *subcircuit cross-listing*, and writes it into output_file.pa#. This file relates the subcircuit node names, in the subcircuit call, to the node names used in the corresponding subcircuit definitions.

Use the output file specification, with a .st# extension, to name the output status. The output status contains the following runtime reports:

- Start and end times for each CPU phase.
- Options settings, with warnings for obsolete options.
- Status of pre-processing 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.

Operating point node voltages are DC operating point initial conditions, which the .SAVE statement stores.

4

Elements

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*.

This chapter describes the syntax for the basic elements of a circuit netlist in HSPICE. Refer to the *HSPICE Elements and Device Models Manual* for detailed syntax descriptions and model descriptions.

This chapter explains the following topics:

- Passive Elements
- Active Elements
- Transmission Lines

- Buffers

Passive Elements

Resistors

The general syntax for a resistor element in a HSPICE netlist is:

SYNTAX:

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

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

```
Rxxx n1 n2 R=‘user-defined_equation’
```

Resistance can be a value (in units of ohms) or an equation. Required fields are the two nodes, and the resistance or the model name. If you use the parameter labels, the node and model name must precede the labels. Other arguments can follow in any order. If you specify a resistor model (see Chapter 2 in the *HSPICE Elements and Device Models Manual*), the resistance value is optional.:

Table 4-1 Resistor Syntax

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.
TC1	First-order temperature coefficient for the resistor. Refer to Chapter 2, “Using Passive Device Models”, in the *HSPICE Elements and Device Models Manual*, for temperature-dependent relations.
TC2	Second-order temperature coefficient for the resistor.

Table 4-1 Resistor Syntax (Continued)

Parameter	**Description**
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 • frequency (HERTZ) • temperature
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, and time

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
```

Linear Resistors

SYNTAX:

The input syntax of a resistor is:

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

Table 4-2 Resistor Syntax

Parameter	Description
R*xxx*	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.
L	Resistor length.
M	Parallel multiplier.
C	Parasitic capacitance between node2 and the substrate.
DTEMP	Temperature difference between element and circuit.

Table 4-2 Resistor Syntax (Continued)

Parameter	Description
SCALE	Scaling factor.

EXAMPLE:

The first resistor, R1, is a simple 10-ohm linear resistor. The second resistor, Rload, calls a resistor model named RVAL, 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 0.5 ohms.

```
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
```

Behavioral Resistors

HSPICE supports resistors with the following equation type:

```
Rxxx n1 n2 . . . <R=> ‘equation’ . . .
```

Note: The equation can be a function of any node voltage, and any branch current, but not a function of time, frequency, or temperature.

EXAMPLE:

```
R1 A B R = ‘V(A) + I(VDD)’
```

Capacitors

The general syntax for a capacitor element is:

```
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...>
```

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 Chapter 2, 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 uses the model name.

EXAMPLE:

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
```

Table 4-3 Example Capacitance Syntax

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. Refer to Chapter 2, “Using Passive Device Models”, in the *HSPICE Elements and Device Models Manual*, for temperature-dependant relations.
TC2	Second-order temperature coefficient, for the capacitor.
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 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 • frequency (HERTZ) • temperature

Table 4-3 Example Capacitance Syntax (Continued)

Parameter	**Description**
CTYPE	Determines capacitance charge calculation, for elements with capacitance equations. If the capacitance equation is a function of v(n1,n2), set CTYPE = 1. 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.

EXAMPLE:

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

```
C1 1 2 20p
```

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

SYNTAX:

The input syntax of a capacitor is:

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

Table 4-4 Capacitor Syntax

Parameter	**Description**
C*xxx*	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.

Behavioral Capacitors

HSPICE supports capacitors with the following equation type:

```
Cxxx n1 n2 . . . C='equation' CTYPE=0, 1 or 2
```

Note: You can specify the capacitor value as a function of any node voltage, and any branch current, but not as a function of time, frequency, or temperature.

CTYPE Parameter

CTYPE determines the calculation mode, for elements that use capacitance equations. Set this parameter carefully, to ensure correct simulation results.

- *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.

EXAMPLE 1:CTYPE

```
V1 1 0 pwl(0n 0v 100n 10v)
V2 2 0 pwl(0n 0v 100n 10v)
C1 1 0 CTYPE = '(V(1) + V(2))*1e-12'
```

Charge-Conserving Capacitors

C*xxx* n1 n2 . . . Q = ‘equation’

$$C = \frac{dQ}{dV}, \ V = V(n1,n2)$$

C*xxx* a b Q=’f(V(a,b))’

The above equation is equivalent to:

C*xxx* a b Q=’f(V(a,b))’ where $d(x) = \frac{df(x)}{dx}$

EXAMPLE 1:

```
C1 a b Q = ’sin(V(a,b)) + V(c,d)*V(a,b)’
```

The above equation is equivalent to:

```
C1 a b C = ’cos (V(a,b)) + V(c,d)’
```

Note: Charge-conserving capacitors are a more-accurate solution.

Inductors

General Form

```
Lxxx n1 n2 <L = >inductance <mname> <<TC1 = >val>
+ <<TC2 = >val> <SCALE = val> <IC = val> <M = val>
+ <DTEMP = val> <R = val>
```

```
Lxxx n1 n2 L = ‘equation’ <LTYPE = val> <above_options...>
```

Polynomial Form

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

Magnetic Winding Form

```
Lxxx n1 n2 NT = turns <above_options...>
```

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 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 Chapter 2 in the *HSPICE Elements and Device Models Manual*), the inductance value is optional.

Table 4-5 Inductor Syntax (Sheet 1 of 2)

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.
TC1	First-order temperature coefficient for the inductor. Refer to Chapter 2, “Using Passive Device Models”, 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 uses this value as the DC operating point voltage, when you specify UIC in the .TRAN statement. The .IC statement overrides it.

Table 4-5 Inductor Syntax (Sheet 2 of 2)

Parameter	**Description**
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 • frequency (HERTZ) • temperature
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 • frequency (HERTZ) • temperature
LTYPE	Calculates inductance flux for elements, using inductance equations. If the inductance equation is a function of `i(L`*`xxx`*`)`, then set `LTYPE = 1`. Use this setting correctly, to ensure proper inductance calculations, and correct simulation results. Default = 0.
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 Chapter 2, "Using Passive Device Models", in the *HSPICE Elements and Device Models Manual* for model information.

EXAMPLE:

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
```

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
```

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
```

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
```

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

The general syntax for a mutual inductor element is:

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

Mutual Core Form

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

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 Chapter 2, “Using Passive Device Models”, in the *HSPICE Elements and Device Models Manual*), the inductance value is optional.

Table 4-6 Mutual Inductor Syntax

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.
Kaaa	Saturable core element name. Must begin with K, followed by up to 1023 alphanumeric characters.
Lbbb, Lccc, Lddd	Names of the windings about the K*aaa* core. One winding element is required, and each winding element must use the magnetic winding syntax.
mname	Saturable core model name. See Chapter 2, “Using Passive Device Models”, in the *HSPICE Elements and Device Models Manual* for model 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 Chapter 2, “Using Passive Device Models”, in the *HSPICE Elements and Device Models Manual*).

You can determine the coupling coefficient, based on geometric and spatial information. To determine the final coupling inductance, HSPICE 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 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:

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

```
K1 Lin Lout 0.9
```

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.

The coupling coefficients are 0.98 and 0.87. HSPICE 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
```

Linear Inductors

SYNTAX:

```
Lxxx node1 node2 <L => inductance <TC1 = val> <TC2 = val>
+ <M = val> <DTEMP = val> <IC = val>
```

Table 4-7 Linear Inductor Syntax

Parameter	**Description**
L*xxx*	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.

Active Elements

Diode Element

The general syntax for a diode element is:

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>
```

The only required fields are the two nodes, and the model name. If you use the parameter labels, the nodes and model name must be first, and the other optional arguments can be in any order.

Table 4-8 Diode Element Syntax

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 calculates it from the width and length.
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 calculates it from the width and length specifications.

Table 4-8 Diode Element Syntax (Continued)

Parameter	Description
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)

Examples

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

```
D1 1 2 diode1
```

The Dprot diode, with anode and cathode, connects to the output node. 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
```

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

The general syntax for a BJT element is:

SYNTAX:

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

or

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

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.

Table 4-9 BJT Element Syntax

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.
IC = vbeval, vceval, VBE, VCE	Initial internal base-emitter voltage (vbeval) and collector-emitter voltage (vceval). HSPICE 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.

EXAMPLE:

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.

In the Qopamp1 BJT element below:

```
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.

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

SYNTAX:

```
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>
```

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

Table 4-10 JFET/MESFET Syntax

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
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.

EXAMPLE:

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.

In the Jopamp1 JFET element below:

```
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.

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

SYNTAX:

```
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...>
```

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.

Table 4-11 MOSFET Element Syntax

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
L	MOSFET channel length, in meters. This parameter overrides .OPTION DEFL. Default = DEFL, with a maximum value of 0.1m.
W	MOSFET channel width, in meters. This parameter overrides DEFW in an .OPTION statement. Default = DEFW.
AD	Drain diffusion area. Overrides DEFAD in the .OPTION statement. Default = DEFAD, if you set the ACM = 0 model parameter.

Table 4-11 MOSFET Element Syntax (Continued)

Parameter	Description
AS	Source diffusion area. Overrides DEFAS in the .OPTION statement. 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 DEFNRS in the .OPTION statement. 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.
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.

EXAMPLE:

In the M1 MOSFET element below:

```
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 Mopamp1 MOSFET element below:

```
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 Mdrive MOSFET element below:

```
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.

Input Syntax for the W Element

The W element supports four 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.

The input syntax for the W element card is:

```
Wxxx i1 i2 ... iN iR o1 o2 ... oN oR N=val L=val
+ TABLEMODEL=name
```

Table 4-12 W Element Input Syntax

Parameter	Description
N	Number of signal conductors (excluding the reference conductor).
i1...iN	Node names for the near-end signal-conductor terminal.
iR	Node name for the near-end reference-conductor terminal.
o1...oN	Node names for the far-end signal-conductor terminal.
oR	Node name for the far-end reference-conductor terminal.
L	Length of the transmission line.
TABLEMODEL	Name of the frequency-dependent tabular 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 = fname> <COORD=0|DESCART|1|POLAR>
+ N = val L = val
```

U-Model Form

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

Field Solver Form

```
Wxxx in1 <in2 <...inx>> refin out1 <out2 <...outx>>
+ refout <FSmodel = mname> 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.

Table 4-13 W Element Syntax

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 = fname	File name reference, for the file containing the RLGC information for the transmission lines (for syntax, see Chapter 6, “Using Transmission Lines”, in the *HSPICE Elements and Device Models Manual*).
COORD	Invokes the polar field solver only if COORD=1 or COORD=POLAR.
ORIGIN	Should be (radius, degree) for the polar field solver.
Umodel = mname	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 = mname	Internal field solver model name. References the PETL internal field solver, as the source of the transmission-line characteristics (for syntax, see Chapter 6, “Using Transmission Lines”, in the *HSPICE Elements and Device Models Manual*).

EXAMPLE:

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
```

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

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

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

- 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.

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
```

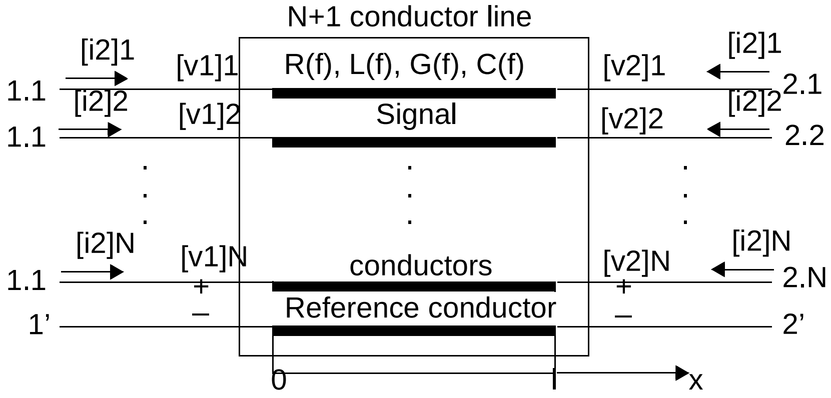
- The i1, i2, i3, i4 and i5 input nodes connect to the o1, o3, and o5 output nodes.
- The i5 input and three outputs (o1, o3, and o5) are all grounded.
- board1 references the Field Solver model.
- The transmission line is 1 millimeter long.

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, or Umodel models, in a single W-element card.

Figure 4-1 shows node numbering for the element syntax.

Figure 4-1 Terminal Node Numbering for W Element



Example 2: Coaxial Line

```
*PETL Example: Coaxial line
.OPTION PROBE POST
VIMPULSE in1 gnd AC=1v PULSE 4.82v 0v 5ns V+0.5ns 0.5ns 25ns
*W element VW1 in1 gnd out1 gnd FSMODEL=coax N=1, L=1
R1 out1 gnd 50

* [[Material List]]
.MATERIAL diel_1 DIELECTRIC ER=4
.MATERIAL copper METAL CONDUCTIVITY=57.6meg

* [[Shape List]]
.SHAPE circle_1 CIRCLE RADIUS=0.5m

* [[Layer Stack]]
.LAYERSTACK coaxial LAYER=(diel_1 11m) $ only one

* [[Field solver option]]
.FSOPTIONS myOpt printdata=yes computers=yes computegd=yes
computego=yes

* [[Field solver Model]]
.MODEL coax W MODELTYPE=FIELDSOLVER FSOPTIONS=myOpt
COORD=polar
+ LAYERSTACK=coaxial, RLGCFILE=coax.rlgc
+ CONDUCTOR = (SHAPE=circle_1, MATERIAL=copper, ORIGIN=(0,
0))
```

```
.TRAN 0.5n 100n
.PROBE v(in1) v(out1)
.END
```

Example: Shield Twin-Lead Lines

```
*PETL Example: Shield twin-lead lines
.OPTION PROBE POST
VIMPULSE in1 gnd AC=1v PULSE 4.82v 0v 5ns
+0.5ns 0.5ns 25ns

*W element
W1 in1 in2 0 out1 out2 0 FSMODEL=twin, N=2, L=1
R1 out1 gnd 50
R2 out2 gnd 50
R3 in2 gnd 50

* [[Material List]]
.MATERIAL diel_1 DIELECTRIC ER=4
.MATERIAL copper METAL CONDUCTIVITY=57.6meg

* [[Shape List]]
.SHAPE circle_1 CIRCLE RADIUS=0.5m

* [[Layer Stack]]
.LAYERSTACK coaxial LAYER=(diel_1 11m)) $ only one

* [[Field solver option]]
.FSOPTIONS myOpt printdata=yes computers=yes computegd=yes
+ computego=yes

* [[Field solver Model]]
.MODEL twin W MODELTYPE=FIELDSOLVER FSOPTIONS=myOpt
+ COORD=polar LAYERSTACK=coaxial, RLGCFILE=twin.rlgc
+ CONDUCTOR = (SHAPE=circle_1, MATERIAL=copper,
+ ORIGIN=(4.5m, 0)) CONDUCTOR = (SHAPE=circle_1,
+ MATERIAL=copper, ORIGIN=(4.5m, 180))

.TRAN 0.5n 100n
.PROBE v(in1) v(out1) v(out2)
.END
```

T Element Statement

The general syntax for a lossless (T Element) transmission line element is:

```
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
```

Only one input and output port is allowed.

Table 4-14 T Element Syntax

T*xxx*	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.

Table 4-14   T Element Syntax (Continued)

mname	U-model reference name. A lossy transmission line model, representing the characteristics of the lossless transmission line.

EXAMPLE:

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.

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.

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.

U Element Statement

General syntax for a lossy (U Element) transmission line element is:

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

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.

Table 4-15 U Element Syntax

Parameter	**Description**
U*xxx*	Lossy (U Element) transmission line element name. Must begin with U, followed by up to 1023 alphanumeric characters.
in*x*	Signal input node for the x^{th} transmission line (in1 is required).
refin	Ground reference for the input signal.
out*x*	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.

EXAMPLE:

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.

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.

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

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

This 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 analysis, see Chapter 6 of the *HSPICE Elements and Device Models Manual*.

SYNTAX:

In HSPICE, the syntax of the S Element is:

```
Sxxx nd1 nd2 ... ndN nd_ref <MNAME=Smodel_name>
+ [FQMODEL=sp_model_name | TSTONEFILE=filename |
+ CITIFILE=filename]<TYPE=[s | y]> <Zo=val | vector_value>
+ <Zof=ref_model> <FBASE=base_fequency>
+ <FMAX=maximum_frequency>
+ <PRECFAC=val> <DELAYHANDLE=ON|OFF> <DELAYFQ=val>
```

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.

Table 4-16 S Element Syntax

Parameter	**Description**
`nd1 nd2 ... ndN`	*N* signal nodes (see Figure 4-2 on page 4-42). Required fields; you must specify these parameters first. The other parameters are optional, and you can specify them in any order.
nd_ref or NdR	Reference node.
MNAME	S model name.
FQMODEL	Frequency behavior of S,Y, or Z parameters. .MODEL statement of sp type defines the frequency-dependent matrices array.

Table 4-16 S Element Syntax (Continued)

Parameter	**Description**
Zo	Characteristic impedance value, for the reference line (frequency-independent). For multiple terminals (*N*>1), HSPICE assumes that the characteristic impedance matrix of the reference lines is diagonal, and that you set diagonal values to Zo. To specify general types of reference lines, use Zof. The default value is 50Ω.
Zof	Name of the frequency-varying model, which defines the frequency behavior of the reference system. If you define both `Zo` and `Zof`, then `Zof` has precedence.
TYPE	Parameter type: • S (scattering), the default • Y (admittance) • Z (impedance)
FBASE	Base frequency to use for transient analysis. This value becomes the base frequency point for Inverse Fourier Transformation. • If you do not set this value, the base frequency is a reciprocal value of the transient period. • If you set a frequency that is smaller than the reciprocal value of the transient, then transient analysis performs circular convolution, and uses the reciprocal value of FBASE as its base period.
FMAX	Maximum frequency to use for transient analysis. Used as the maximum frequency point for Inverse Fourier Transformation.
IDC	Terminal bias current at DC. If you set this value, then the element acts as a current source at DC, instead of using the network parameter matrix.
VDC	Terminal bias voltage at DC. If you set this value, then the element acts as a voltage source at DC, instead of using the network parameter matrix.
s_model_name	Name of .MODEL with S keyword.

The S model syntax is:

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

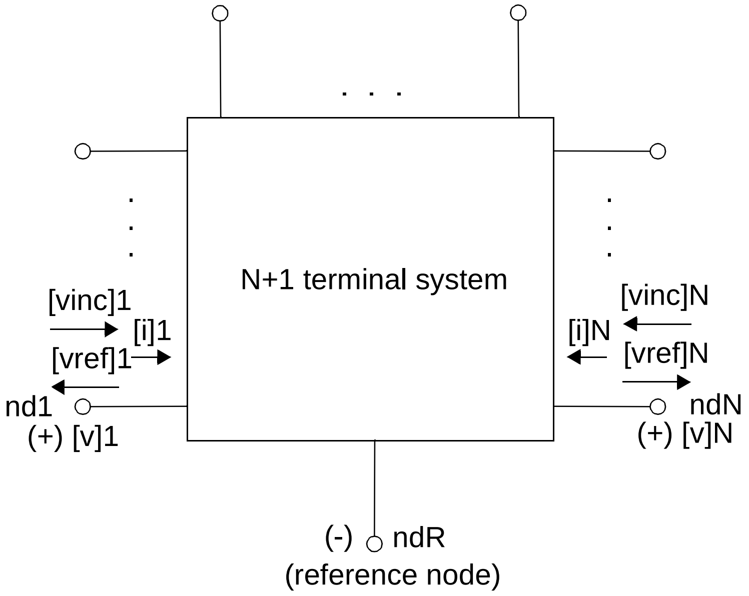
Table 4-17 S Model Syntax

Parameter	**Description**
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 the touch stone file. This file name must use the following file extension syntax: *filename*.[s|y|z]#
CITIFILE	Name of the citi file.
TYPE	One of the following parameter types: • S (scattering), the default • Y (admittance) • Z (impedance)
Zo	Characteristic impedance value for the reference line (frequency-independent). For multiple terminals (*N*>1), HSPICE that the characteristic impedance matrix of the reference lines is diagonal, and that you set diagonal values to Zo. To specify more general types of reference lines, use Zof. Default=50Ω.
Zof	Name of the frequency-varying model, which defines the frequency behavior of the reference system. If you define both Zo and Zof, then Zof has precedence.
FBASE	Base frequency to use for transient analysis. This value becomes the base frequency point for Inverse Fourier Transformation. • If you do not set this value, the base frequency is a reciprocal value of the transient period. • If you set a frequency that is smaller than the reciprocal value of the transient, then transient analysis performs circular convolution, and uses the reciprocal value of FBASE as its base period.

Table 4-17 S Model Syntax (Continued)

Parameter	Description
FMAX	Maximum frequency for transient analysis. Used as the maximum frequency point for Inverse Fourier Transform.
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.
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.
DELAYHANDLE	Delay frequency for transmission line type parameters. Default=OFF. 1 (default) activates the delay handler. See Group Delay Handler in Time Domain Analysis on page 4-42 0 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.
DELAYFQ	Delay frequency for transmission line type parameters. For best performance, set this to the inverse of the delay time. Default=FMAX.
PRECFAC	Preconditioning factor, to avoid a singularity (infinite admittance matrix). See Pre-Conditioning S Parameters on page 4-43. Default=0.75.

Figure 4-2   Terminal Node Notation



Frequency Table Model

The Frequency Table Model is a generic model that you can use to describe frequency-varying behavior. Currently, the *S* element and .sp use this model. For a description of this model, see “Frequency Table Model” in Chapter 6, Linear N-Ports/Transmission Lines, in the *HSPICE Elements and Device Models Manual*.

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=0|1> keyword in the S model statement.

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

$$y'_{mn(\omega)} = y_{mn(\omega)} \times e^{j\omega\Upsilon_{mn}} \qquad (2)$$

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

$$i_{k(t)} = (y'_{k1(t)}, y'_{k2(t)}, \ldots, y'_{kN(t)}) \times (v_{1(t-\Upsilon k1)}, v_{2(t-\Upsilon k2)}, \ldots, v_{N(t-\Upsilon kN)})^{\Upsilon} \qquad (3)$$

Pre-Conditioning S Parameters

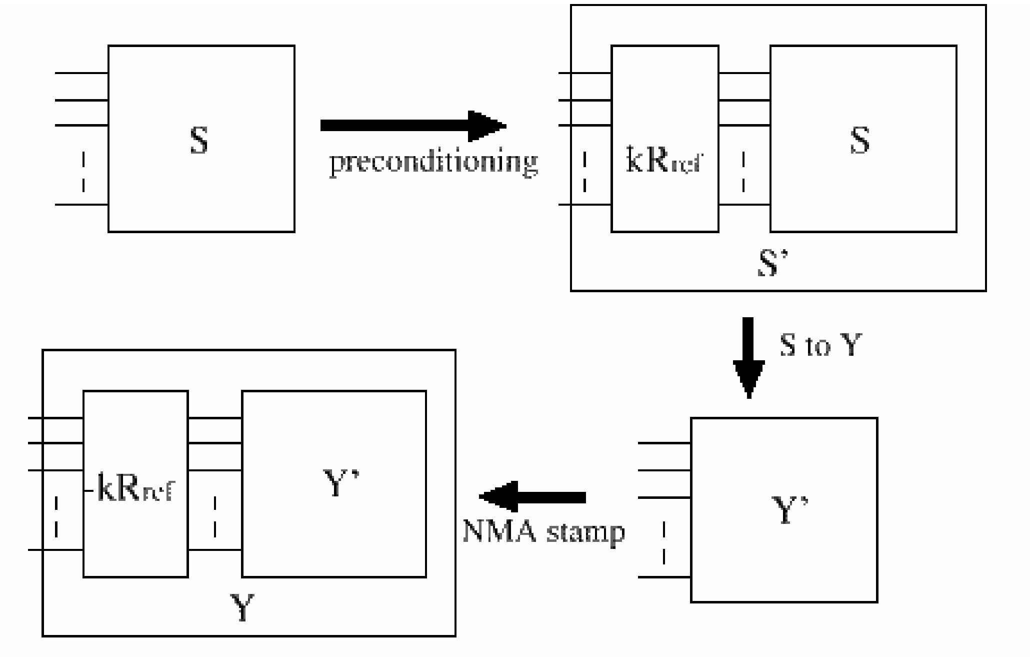
Certain S parameters, such as series inductor (2-port), show a singularity when converting S to Y parameters. To avoid this singularity, the S element adds *k*R$_{ref}$ series resistance, to pre-condition S matrices:

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

- R$_{ref}$ is the reference impedance vector.
- *k* is the pre-conditioning factor.

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

Figure 4-3 Pre-Conditioning S Parameters



Buffers

The general syntax of an element card for input/output buffers is:

```
bname node_1 node_2 ... node_N keyword_1 = value_1 ...
+ [keyword_M = value_M]
```

Table 4-18 B Element Syntax

Parameter	Description
bname	Buffer name, and starts with the letter *B.*
node_1 node_2 ... node_N	List of input/output buffer external nodes. The number of nodes and their meaning are specific to different buffer types.
keyword_i = value_i	Assigns a value of *value_i* to the *keyword_i* keyword*.* Specify optional keywords in [brackets].

For information about the keywords, see Chapter 7, "Using IBIS Models", in the *HSPICE Elements and Device Models Manual*.

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 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 Chapter 7, “Using IBIS Models”, in the *HSPICE Elements and Device Models Manual*.

5

Sources and Stimuli

This chapter describes element and model statements for independent sources, dependent sources, analog-to-digital elements, and digital-to-analog elements. It also explains each type of element and model statement. Explicit formulas and examples show how various combinations of parameters affect the simulation.

The chapter explains the following topics:

Independent Source Elements

Use independent source element statements to specify DC, AC, transient, and mixed independent voltage and current sources. Some types of analysis use the associated analysis sources. For example, in a DC analysis, if you specify both DC and AC sources in one independent source element statement, HSPICE removes the AC source from the circuit, for the DC analysis. If you specify an independent source for an AC, transient, and DC analysis, HSPICE removes transient sources, calculates the operating point, and removes DC sources, for the AC analysis. Initial transient values always override the DC value.

Source Element Conventions

You do not need to ground voltage sources. HSPICE 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 reserved keywords to identify these parameters:

```
AC    ACI      AM     DC     EXP     PE     PL
PU    PULSE    PWL    R      RD      SFFM   SIN
```

Independent Source Element

The general syntax for an independent source is:

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

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

Table 5-1 Independent Source Element Syntax

Parameter	**Description**
Vxxx	Independent voltage source element name. Must begin with *V*, followed by up to 1023 alphanumeric characters.
Iyyy	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, 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 multiplies source current by M. Default=1.0.

EXAMPLE 1:

```
VX 1 0 5V
```

- 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
```

- 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
```

- 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)
```

- 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
```

- 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 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)
```

- 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)
```

- 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 9The 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:

- pulse
- exponential
- damped sinusoidal
- single-frequency FM
- piecewise linear

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 selects the appropriate DC, AC, or transient source. The exception is the zero-time value of a transient source, which over-rides 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)
```

The preceding example specifies:

- 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.

Independent Source Functions

HSPICE uses the following types of independent source functions:

- Pulse (PULSE function)
- Sinusoidal (SIN function)
- Exponential (EXP function)
- Piecewise linear (PWL function)
- Single-frequency FM (SFFM function)
- Single-frequency AM (AM function)

HSPICE also provides a data-driven version of PWL. 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, 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.

Pulse Source Function

HSPICE 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.

The syntax for a pulse source, in an independent voltage or current source, is:

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

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

Table 5-2 Pulse Source Syntax

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 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 5-3 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
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:

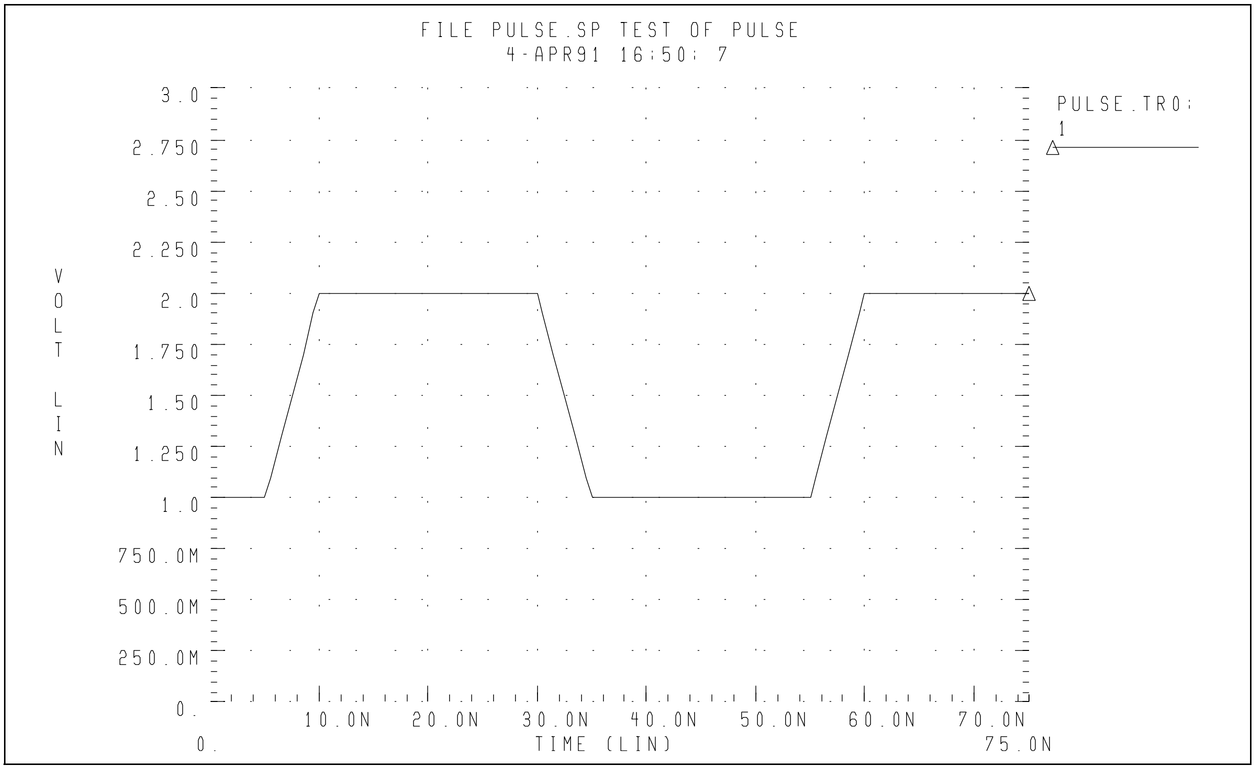
The following example shows an entire netlist, which contains a PULSE voltage source. In the source:

- 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.

```
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 5-1 shows the result of simulating this netlist, in HSPICE.

Figure 5-1   Pulse Source Function



Sinusoidal Source Function

HSPICE provides a damped sinusoidal source, 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:

The syntax for a sinusoidal source in an independent voltage or current source is:

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

Table 5-4   Sinusoidal Source Syntax

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 RMS amplitude, 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 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:

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

In these expressions, TSTOP is the final time.

EXAMPLE:

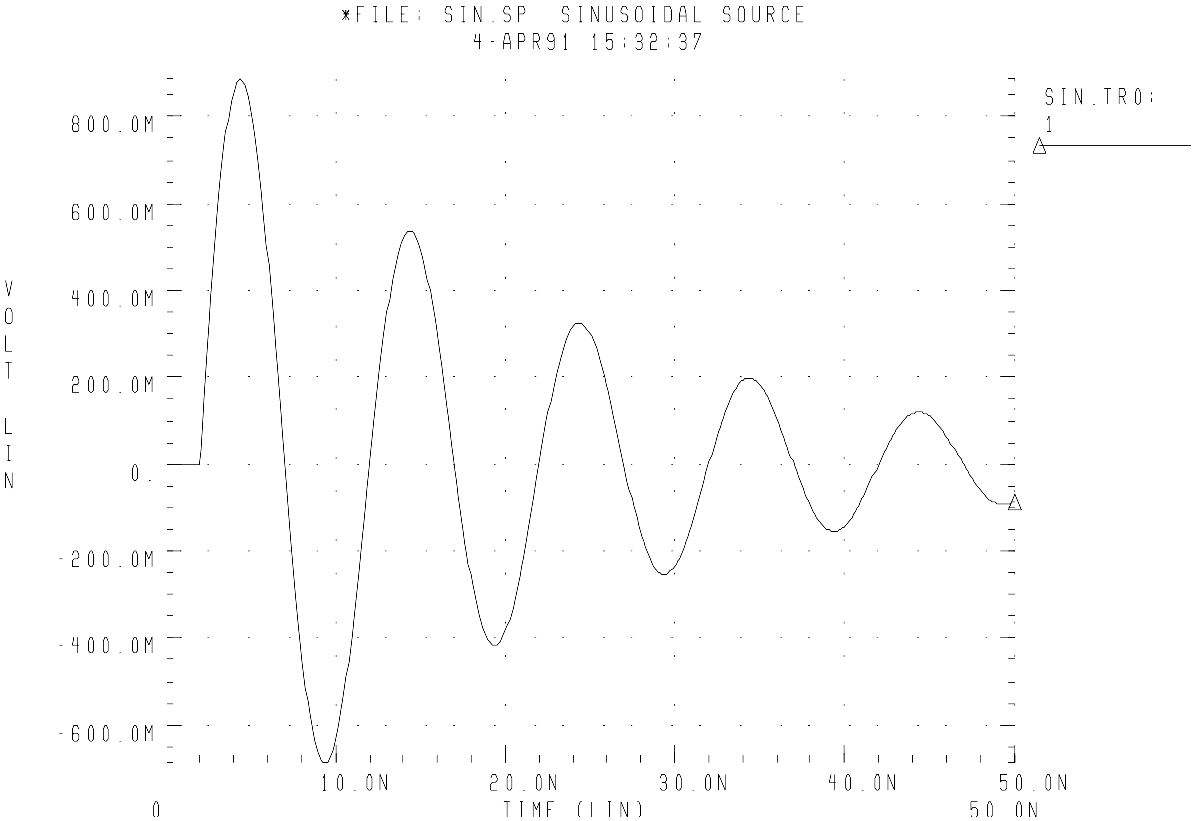
```
VIN 3 0 SIN (0 1 100MEG 1NS 1e10)
```

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 5-2 on page 5-14 for a plot of the source output.

Figure 5-2 Sinusoidal Source Function



```
*File: SIN.SP THE SINUSOIDAL WAVEFORM
*<decay envelope>
.OPTION 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 5-5 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

The general syntax for an exponential source, in an independent voltage or current source, is:

```
Vxxx n+ n- EXP <(> v1 v2 <td1 <t 1 <td2 <t 2>>>> <)>

Ixxx n+ n- EXP <(> v1 v2 <td1 <t 1 <td2 <t 2>>>> <)>
```

Table 5-6 Exponential Source Syntax

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.
t 1	Rise time constant, in seconds. Default=TSTEP.
t 2	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:

Time Value

0 to td1 v1

$$\text{td1 to td2} \quad v1 + (v2 - v1) \cdot \left[1 - \mathrm{Exp}\left(-\frac{\mathrm{Time} - td1}{\tau_1}\right)\right]$$

$$\text{td2 to tstop} \quad v1 + (v2 - v1) \cdot \left[1 - \mathrm{Exp}\left(-\frac{(\mathrm{Time} - td1)}{\tau_1}\right)\right] + (v1 - v2) \cdot \left[1 - \exp\left(-\frac{(\mathrm{Time} - td2)}{\tau_2}\right)\right]$$

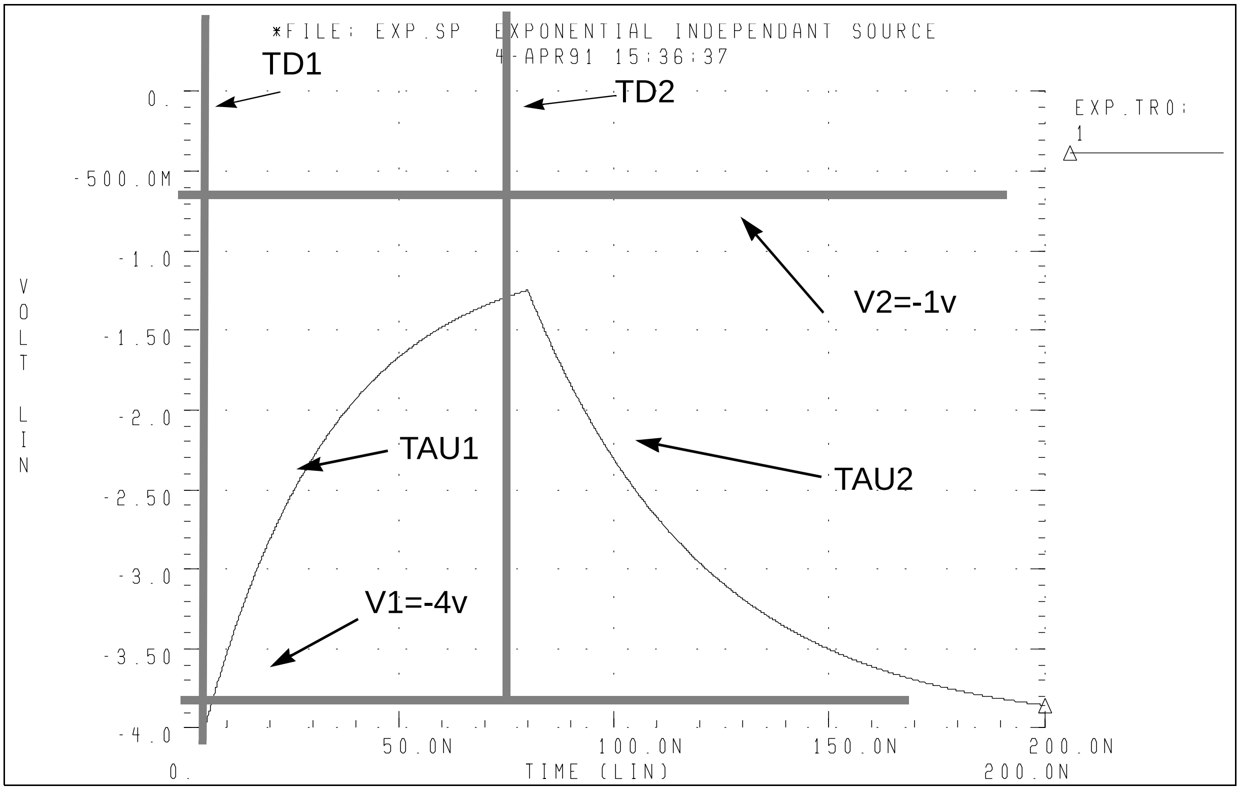
EXAMPLE:

```
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 5-3 Exponential Source Function



```
*FILE: EXP.SP THE EXPONENTIAL WAVEFORM
.OPTION POST
.PARAM V1=-4 V2=-1 TD1=5N TAU1=30N TAU2=40N TD2=80N
V 1 0 EXP (V1 V2 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 (PWL) Source Function

The general syntax for a piecewise linear source, in an independent voltage or current source, is:

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> <)>
```

Table 5-7 Piecewise Linear Source Syntax

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*.
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 uses the DC value of the source, as the time-zero source value.
- HSPICE 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 extrapolates neighboring values. As a result, the simulated voltage might deviate slightly from the voltage specified in the PWL list. To force HSPICE to use the specified values, use the SLOPETOL option, 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:

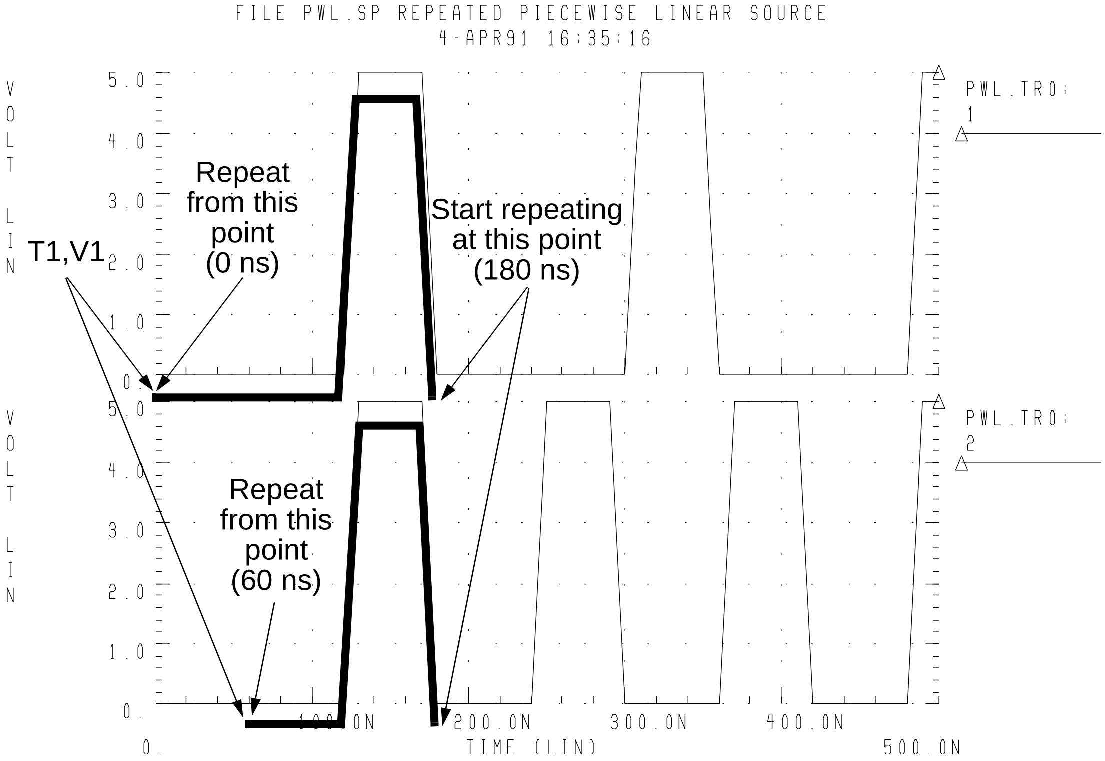
```
*FILE: PWL.SP THE REPEATED PIECEWISE LINEAR SOURCE
*ILLUSTRATION OF THE USE OF THE REPEAT FUNCTION “R”
*file pwl.sp REPEATED PIECEWISE LINEAR SOURCE
.OPTION POST
.TRAN 5N 500N
V1 1 0 PWL 60N 0V, 120N 0V, 130N 5V, 170N 5V, 180N 0V,
+ R 0N
R1 1 0 1
V2 2 0 PL 0V 60N, 0V 120N, 5V 130N, 5V 170N, 0V 180N,
+ 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 5-4 on page 5-20 for the difference in responses.

Figure 5-4 Results of Using the Repeat Function



Data-Driven Piecewise Linear Source

The general syntax for a data-driven piecewise linear source, in an independent voltage or current source, is:

```
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
```

Table 5-8 Data-Driven Piecewise Linear Source Syntax

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:

```
*DATA DRIVEN PIECEWISE LINEAR SOURCE
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 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 Function

The general syntax for including a single-frequency, frequency-modulated source in an independent voltage or current source is:

```
Vxxx n+ n- SFFM <(> vo va <fc <mdi <fs>>> <)>
```

```
Ixxx n+ n- SFFM <(> vo va <fc <mdi <fs>>> <)>
```

Table 5-9 Single-Frequency FM Source Syntax

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.
*md*i	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:

$$\text{sourcevalue} = \text{vo} + \text{va} \cdot \text{SIN}[2 \cdot \pi \cdot \text{fc} \cdot \text{Time} + \text{mdi} \cdot \text{SIN}(2 \cdot \pi \cdot \text{fc} \cdot \text{Time})]$$

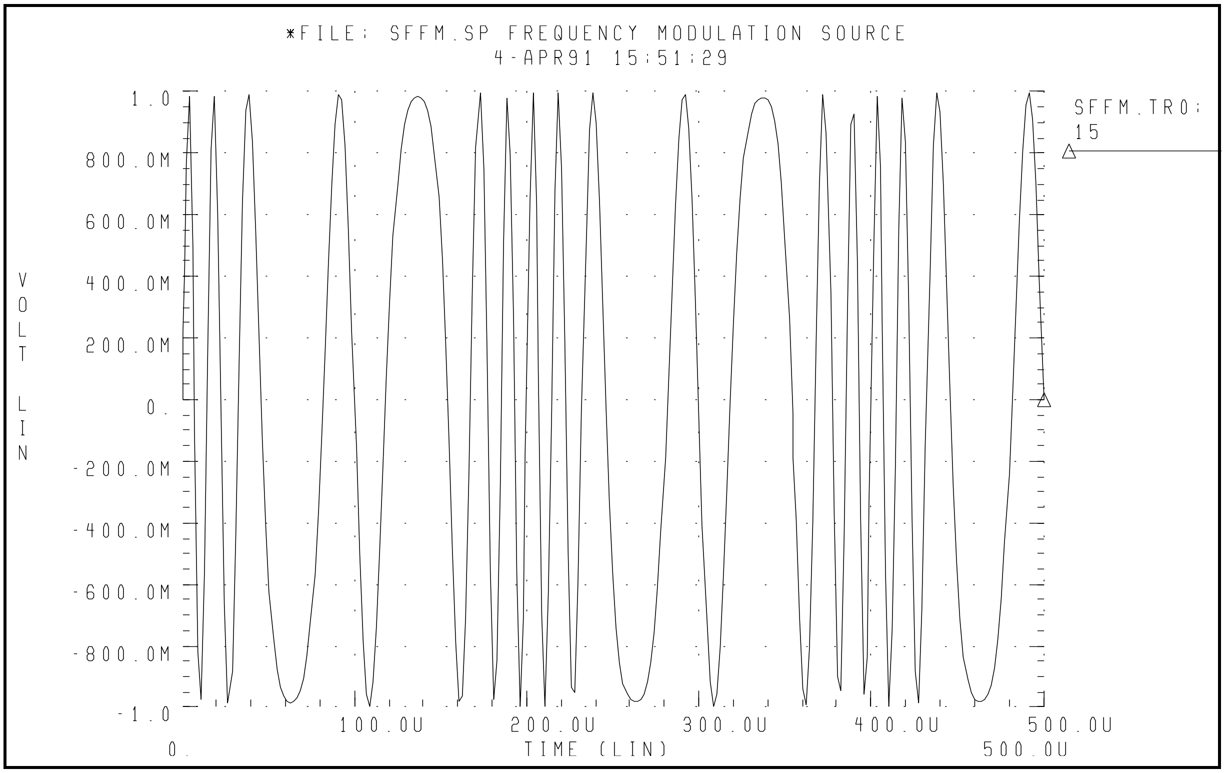
EXAMPLE:

```
*FILE: SFFM.SP THE SINGLE FREQUENCY FM SOURCE
.OPTION POST
V 1 0 SFFM (0, 1M, 20K. 10, 5K)
R 1 0 1
.TRAN .0005M .5MS
.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 5-5 Single Frequency FM Source



Amplitude Modulation Source Function

The general syntax for including a single-frequency, frequency-modulated source in an independent voltage or current source is:

V*xxx* *n+* *n-* AM < (> *sa* *oc* *fm* *fc* <*td*> <)>

I*xxx* *n+* *n-* AM < (> *sa* *oc* *fm* *fc* <*td*> <)>

Table 5-10 AM Source Syntax

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:

$$\text{sourcevalue} = \text{sa} \cdot \{\text{oc} + \text{SIN}[2 \cdot \pi \cdot \text{fm} \cdot (\text{Time} - \text{td})]\} \cdot \text{SIN}[2 \cdot \pi \cdot \text{fc} \cdot (\text{Time} - \text{td})]$$

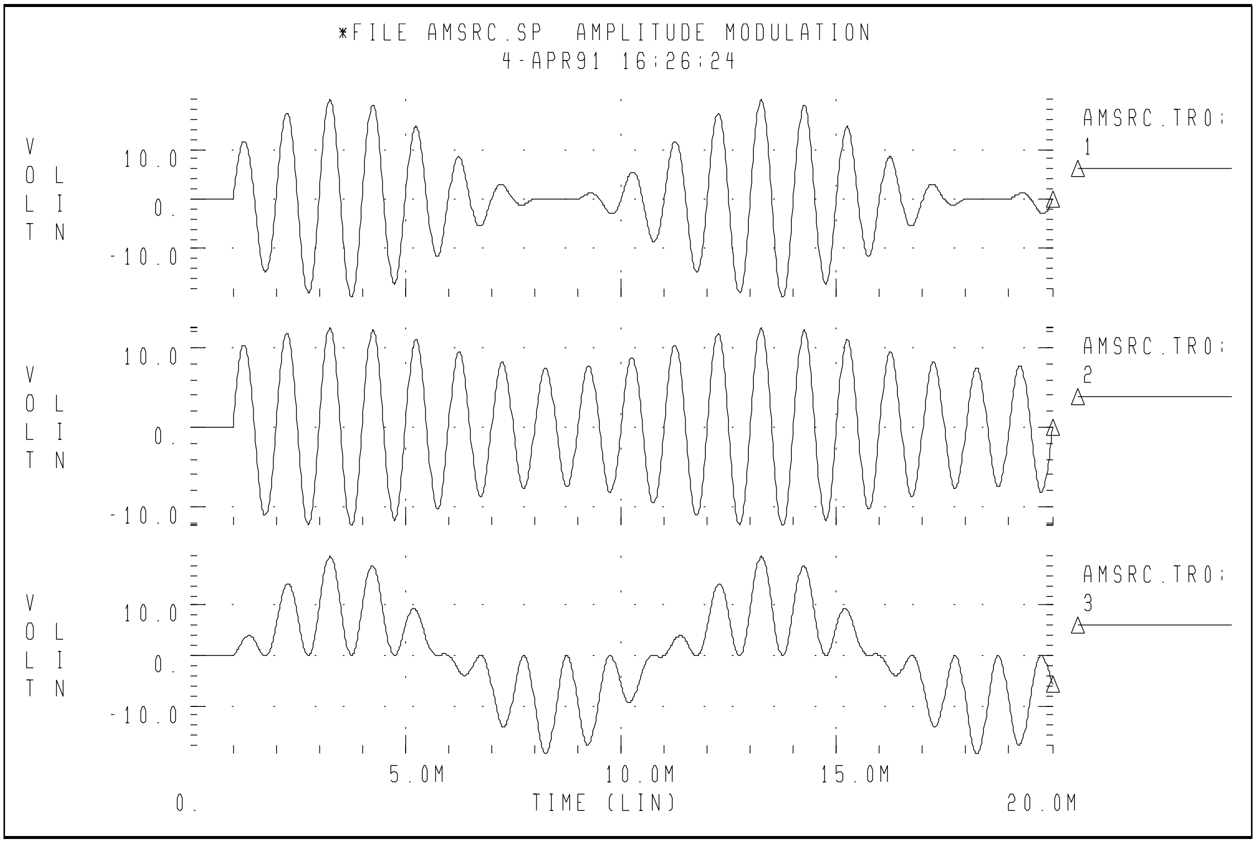
EXAMPLE:

```
.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 5-6 Amplitude Modulation Plot



Voltage and Current Controlled Elements

HSPICE 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 current-controlled 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 current-controlled 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 non-linear functions.

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.

Table 5-11 Polynomial Syntax

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:

$$FV = P0 + (P1 \cdot FA) + (P2 \cdot FA^2) + (P3 \cdot FA^3) + (P4 \cdot FA^4) + (P5 \cdot FA^5) + \ldots$$

Table 5-12 One-Dimensional Syntax

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 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
```

1. The single-dimension polynomial function parameter, POLY(1), informs HSPICE 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).

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:

$$FV = P0 + (P1 \cdot FA) + (P2 \cdot FB) + (P3 \cdot FA^2) + (P4 \cdot FA \cdot FB) + (P5 \cdot FB^2) \\ + (P6 \cdot FA^3) + (P7 \cdot FA^2 \cdot FB) + (P8 \cdot FA \cdot FB^2) + (P9 \cdot FB^3) + \ldots$$

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:

V(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 0 5

or

```
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 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>>(...)
```

Table 5-13 Power Source Parameters

Parameter	**Description**
powerVal	A constant power source supplies the available power. If you specify option=POWER_DB, then the value is in *decibels*; otherwise it is in *Watts*POWER_SCAL*. In this equation, *POWER_SCAL* is a scaling factor that you specify in the .OPTION statement.
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 Hertz, the power value is 2 decibel/unit.
- At 10K Hertz, 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.

Table 5-14 Independent Source Options

Option	**Description**
POWER_SCAL	Sets the scaling factor, for power values. • If you specify this value, the power unit is in *Watts*POWER_SCAL*. • Default is 1.
POWER_DB	Specifies that the power value is in decibels.

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.

- Level=1 is an OPAMP.
- Level=2 is a TRANSFORMER.

Voltage-Controlled Voltage Source (VCVS)

The syntax is:

Linear

```
Exxx n+ n- <VCVS> in+ in- gain <MAX=val> <MIN=val>
+ <SCALE=val> <TC1=val> <TC2=val><ABS=1> <IC=val>
```

You must specify the MAX, MIN, SCALE, TC1, TC2, ABS, and IC parameters.

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.

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>
```

You must specify the DELTA, SCALE, TC1, TC2, and IC parameters.

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.

Delay Element

```
Exxx n+ n- <VCVS> DELAY in+ in- TD=val <SCALE=val>
+ <TC1=val> <TC2=val> <NPDELAY=val>
```

You must specify the NPDELAY, SCALE, TC1, and TC2 parameters.

Behavioral Voltage Source

SYNTAX:

```
Exxx n+ n- VOL=’equation’ <MAX=val> <MIN=val>
```

In this syntax, E*name n1 n2* VOL=’*equation*’ [MAX=*max*][MIN=*min*.

Ideal Op-Amp

SYNTAX:

```
Exxx n+ n- OPAMP in+ in-
```

You can also substitute Level=1 in place of OPAMP.

Ideal Transformer

SYNTAX:

```
Exxx n+ n- TRANSFORMER in+ in- k
```

You can also substitute Level=2 in place of TRANSFORMER.

Table 5-15 E Element Syntax (Sheet 1 of 3)

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.
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.

Table 5-15 E Element Syntax (Sheet 2 of 3)

Parameter	Description
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
k	Ideal transformer turn ratio: V(in+,in-) = k Þ 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 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.
P0, P1 …	The polynomial coefficients. If you specify one coefficient, HSPICE 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 5-29).
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.

Table 5-15 E Element Syntax (Sheet 3 of 3)

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.
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 sets E1 to -5V, and not -8V as the equation suggests.

```
Eopamp 2 3 14 1 MAX=+5 MIN=-5 2.0
```

You can also substitute Level=1 in place of OPAMP.

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 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.

Voltage-Controlled Oscillator (VCO)

The VOL keyword defines a single-ended input, which controls output of a VCO.

In the following 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.

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>
```

You must specify the MAX, MIN, SCALE, TC1, TC2, M, ABS, and IC parameters.

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>
```

You must specify the DELTA, SCALE, TC1, TC2, M, and IC parameters.

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

```
Fxxx n+ n- <CCCS> DELAY vn1 TD=val <SCALE=val>
+ <TC1=val><TC2=val> NPDELAY=val
```

You must specify the NPDELAY, SCALE, TC1, TC2, and M parameters.

Note: G Elements with algebraics make *CCCS* elements obsolete. You can still use *CCCS* elements for backward-compatibility with existing designs.

F Element Parameters

Table 5-16 F Element Syntax

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.

Table 5-16 F Element Syntax (Continued)

Parameter	Description
MAX	Maximum output current. Default=undefined; sets no maximum.
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 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 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 assumes that the polynomials are P0, P1, P2 … See Polynomial Functions on page 5-29.
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: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
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.
y1,...	Corresponding output current values of *x*.

EXAMPLE 1:

```
F1 13 5 VSENS MAX=+3 MIN=-3 5
```

Example 1 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.

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:

```
F2 12 10 POLY VCC 1MA 1.3M
```

Example 2 is a current-controlled current source, with the value:

$$I(F2)=1e-3 + 1.3e-3 \cdot I(VCC)$$

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).

EXAMPLE 3:

```
Fd 1 0 DELAY vin TD=7ns SCALE=5
```

Example 3 is a delayed, current-controlled current source.

EXAMPLE 4:

```
Filim 0 out PWL(1) vsrc -1a,-1a 1a,1a
```

Example 4 is a piecewise-linear, current-controlled current source.

Voltage-Dependent Current Sources — G Elements

This section explains G Element syntax statements, and their parameters.

- Level=0 is a Voltage-Controlled Voltage Source (VCVS).
- 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).

Voltage-Controlled Current Source (VCCS)

The Level=0 syntax is:

Linear

```
Gxxx n+ n- <VCCS> in+ in- transconductance <MAX=val>
+ <MIN=val> <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ <ABS=1> <IC=val>
```

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>
```

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>
```

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.

Delay Element

```
Gxxx n+ n- <VCCS> DELAY in+ in- TD=val <SCALE=val>
+ <TC1=val> <TC2=val> NPDELAY=val
```

Behavioral Current Source Syntax

```
Gxxx n+ n- CUR=’equation’ <MAX>=val> <MIN=val> <M=val>
+ <SCALE=val>
```

Voltage-Controlled Resistor (VCR)

The Level=1 syntax is:

Linear

```
Gxxx n+ n- VCR in+ in- transfactor <MAX=val> <MIN=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val> <IC=val>
```

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>
```

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>
```

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>
```

Voltage-Controlled Capacitor (VCCAP)

The Level=2 (NPWL) and Level=3 (PPWL) piecewise linear syntax is:

```
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 determines whether to use Level=2 (NPWL) or Level=3 (PPWL), based on the relationship of the (n+, n-) and (in+, in-) nodes.

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*+:

- If *v*(*n*+,*n*-) < 0, then the controlling voltage is *v*(*in*+,*in*-).
- Otherwise, the controlling voltage is *v*(*in*+,*n*+).

PPWL Function

For the *in*- node, connected to *n*-:

- If *v*(*n*+,*n*-) < 0, then the controlling voltage is *v*(*in*+,*in1*-).
- Otherwise, the controlling voltage is *v*(*in*+,*n*+).

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*+).

If the *in*- node does not connect to either *n*+ or *n*-, then HSPICE changes NPWL and PPWL to PWL.

G Element Parameters

Table 5-17 G Element Syntax (Sheet 1 of 3)

Parameter	**Description**
ABS	Output is an absolute value, if ABS=1.
CUR, VALUE	Current output that flows from n+ to n-. The equation that you define can be a function of: • node voltages • branch currents • TIME • temperature (TEMPER) • frequency (HERTZ)
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.

Table 5-17 G Element Syntax (Sheet 2 of 3)

Parameter	Description
NDIM	Number of polynomial dimensions. If you do not specify `POLY(NDIM)`, HSPICE 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, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$. 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 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 5-29).
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.
TD	Keyword for the time (propagation) delay.
transconductance	Voltage-to-current conversion factor.
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 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.

Table 5-17 G Element Syntax (Sheet 3 of 3)

Parameter	Description
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$.
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 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)'
```

Current-Dependent Voltage Sources — H Elements

This section explains H Element syntax statements, and defines their parameters.

Current-Controlled Voltage Source (CCVS)

The syntax is:

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

```
Hxxx n+ n- <CCVS> DELAY vn1 TD=val <SCALE=val> <TC1=val>
+ <TC2=val> <NPDELAY=val>
```

Note: E Elements with algebraics make *CCVS* elements obsolete. You can still use *CCVS* elements for backward-compatibility with existing designs.

H Element Parameters

Table 5-18 H Element Syntax

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 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.

Table 5-18 H Element Syntax (Continued)

Parameter	**Description**
P0, P1 . . .	Polynomial coefficients. • If you specify one polynomial coefficient, the source is linear, and HSPICE 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 5-29.
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: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
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*.

H Element Examples

```
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 1:

The defining equation of the CCVS is:

$$HX = 1000 \cdot VCUR$$

The defining equation specifies that the voltage output of HX is 1000 times the value of the current flowing through CUR.

- 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.

CUR 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 2:

```
.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.
- Vector input files.

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 *HSPICE Signal Integrity Guide*.

In HSPICE, the U Element can reference digital input and digital output models, for mixed-mode simulation. Viewlogic's Viewsim mixed mode simulator uses HSPICE, with digital input from Viewsim. 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

The general syntax for a U Element digital source is:

U*xxx interface nlo nhi mname* SIGNAME = *sname* IS = *val*

Table 5-19 U Element Syntax

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= sname	Signal name, as referenced in the digital output file header. Can be a string of up to eight alphanumeric characters.
IS=val	Initial state of the input element. Must be a state that the model defines.

Model Syntax

.MODEL *mname* U LEVEL=5 <*parameters*...>

Digital input.

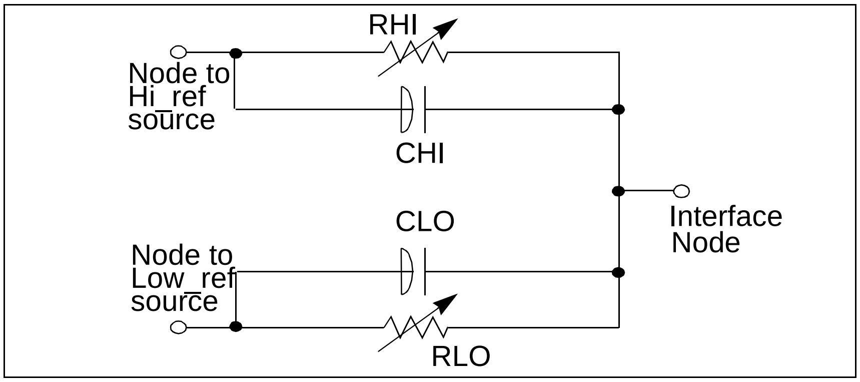
Digital-to-Analog Input Model Parameters

Table 5-20 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.
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 5-7 shows the circuit representation of the element.

Figure 5-7   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.

```
* EXAMPLE OF U-ELEMENT DIGITAL INPUT
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

The general syntax for a digital output in a HSPICE output is:

```
U<name> interface reference mname SIGNAME = sname
```

Table 5-21 U Element Syntax

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= sname	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...>
```

Digital output.

Analog-to-Digital Output Model Parameters

Table 5-22 Analog-to-Digital Parameters

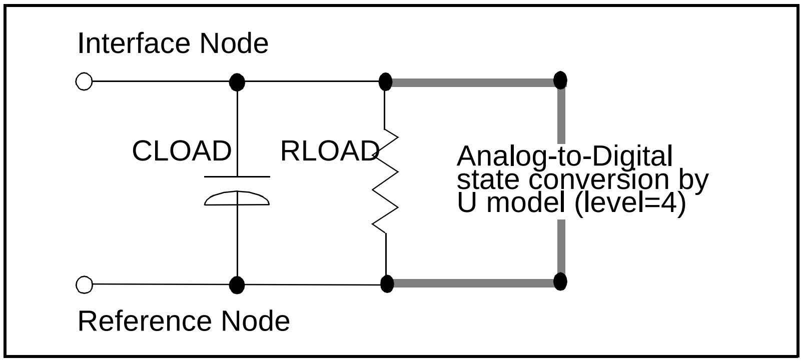
Name (Alias)	Units	Default	Description
RLOAD	ohm	1/gmin	Output resistance.
CLOAD	farad	0	Output capacitance.
S0NAME			State 0 character abbreviation. A string of up to four alphanumerical characters.
S0VLO	volt		State 0 low-level voltage.
S0VHI	volt		State 0 high-level voltage.
S1NAME			State 1 character abbreviation. A string of up to four alphanumerical characters.
S1VLO	volt		State 1 low-level voltage.

Table 5-22 Analog-to-Digital Parameters (Continued)

Name (Alias)	Units	Default	Description
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.

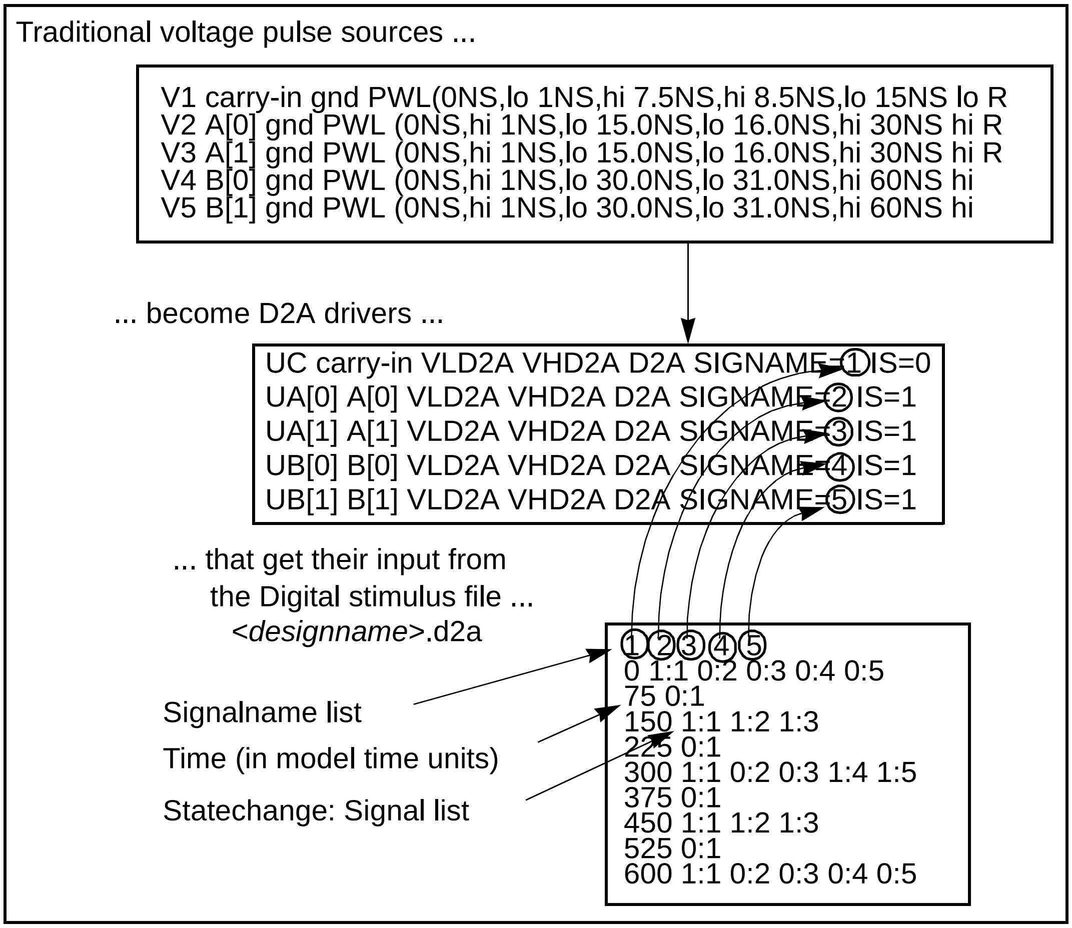
To define up to 20 different states in the model definition, use the S*n*NAME, S*n*VLO and S*n*VHI parameters, where *n* ranges from 0 to 19. Figure 5-8 shows the circuit representation of the element.

Figure 5-8 Analog-to-Digital Converter Element



Replacing Sources With Digital Inputs

Figure 5-9 Digital File Signal Correspondence


Traditional voltage pulse sources ...
... become D2A drivers ...
... that get their input from
the Digital stimulus file ...
<designname>.d2a
Signalname list
Time (in model time units)
Statechange: Signal list

```
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
```

```
UC carry-in VLD2A VHD2A D2A SIGNAME=①IS=0
UA[0] A[0] VLD2A VHD2A D2A SIGNAME=②IS=1
UA[1] A[1] VLD2A VHD2A D2A SIGNAME=③IS=1
UB[0] B[0] VLD2A VHD2A D2A SIGNAME=④IS=1
UB[1] B[1] VLD2A VHD2A D2A SIGNAME=⑤IS=1
```

```
①②③④⑤
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
```

EXAMPLE:

The following is an example of replacing sources with digital inputs.

```
* EXAMPLE OF U-ELEMENT DIGITAL OUTPUT
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
* 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 50N
.END
```

The digital output file should look like:

```
12
0  0:1
1051:1
1970:1
3051:1
3970:1
```

- *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.

The following two-bit MOS adder uses the digital input file. In the plot below, the ‘A[0], A[1], B[0], B[1], and CARRY-IN’ nodes all originate from a digital file input (see Figure 5-9 on page 5-68). HSPICE outputs a digital file.

```
FILE: MOS2BIT.SP - ADDER - 2 BIT ALL-NAND-GATE
+ BINARY ADDER
*
.OPTION ACCT NOMOD FAST scale=1u gmindc=100n post
.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
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
*
```

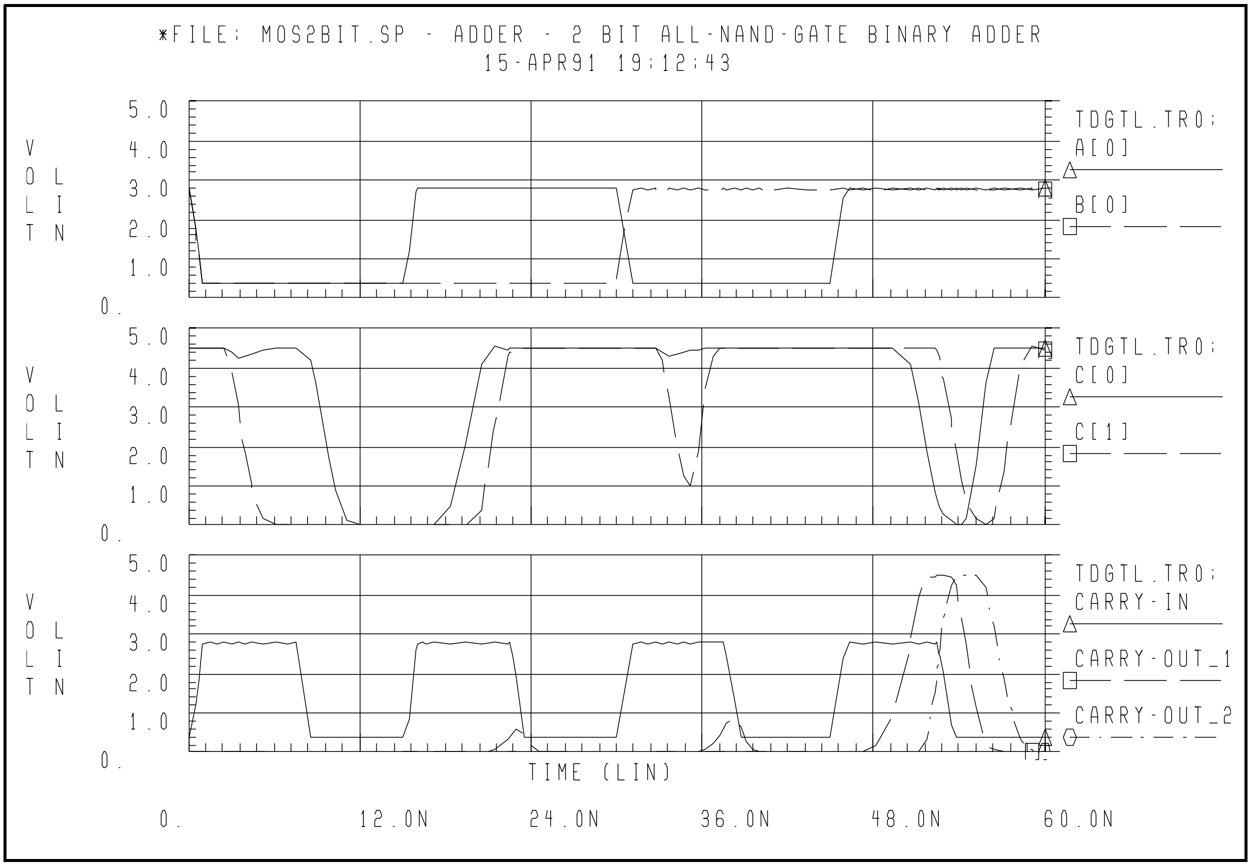
```
* Models
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500 KAPPA=.25 KP=30U
+ ETA=.01 THETA=.04 VMAX=2E5 NSUB=9E16 TOX=400
+ GAMMA=1.5 PB=0.6 JS=.1M XJ=0.5U LD=0.1U NFS=1E11
+ NSS=2E10 RSH=80 CJ=.3M MJ=0.5 CJSW=.1N MJSW=0.3
+ acm=2 capop=4
*
.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 KP=15U
+ ETA=.015 THETA=.04 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
```

Figure 5-10 Digital Stimulus File Input



Specifying a Digital Vector File

The digital vector file consists of three parts:

- *Vector Pattern Definition* section
- *Waveform Characteristics* section
- *Tabular Data* section.

You can use a digital vector file in HSPICE.

To incorporate this information into your simulation, include this line in your netlist:

```
.VEC 'digital_vector_file'
```

The .VEC file must be a text file. If you transfer it between Unix and Windows, use *text* mode.

A single .VEC statement should not reference more than four signal names. Alphabetic characters in hexadecimal numbers must be lower-case (for example, use f instead of F).

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. It must occur first in the digital vector file. The statements within this section (except the *radix* statement) can appear in any order, and all keywords are case-insensitive.

A sample Vector Pattern Definition section follows:

```
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
```

These lines are required, and should be the first lines in a vector file:

- The RADIX line 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, are in units of nanoseconds.

For an explanation of keywords, such as *radix* and *vname*, see Defining Tabular Data on page 5-81.

Radix Statement

The *radix* statement specifies the number of bits associated with each vector. Valid values for the number of bits range from 1 to 4.

Table 5-23 Radix Syntax

# bits	Radix	Number System	Valid Digits
1	2	Binary	0, 1
2	4	–	0 – 3
3	8	Octal	0 – 7
4	16	Hexadecimal	0 – F

The file contains only one *radix* statement. It must be the first non-comment line.

SYNTAX:

```
RADIX 1 1 1 1 1 1 1 1 1 1 1 1
```

EXAMPLE:

This example illustrates two 1-bit signals, followed by a 4-bit signal, followed by one each 1-bit, 2-bit, 3-bit, and 4-bit signals, and finally eight 1-bit signals.

```
; start of Vector Pattern Definition section
RADIX 1 1 4 1234 1111 1111
VNAME A B C[3:0] I9 I[8:7] I[6:4] I[3:0] O7 O6 O5 O4
+ O3 O2 O1 O0
IO I I I IIII OOOO OOOO
```

Vname Statement

The *vname* statement defines the name of each vector. If you do not specify *vname*, HSPICE assigns a default name to each signal: V1, V2, V3, and so on. If you define more than one VNAME statement, the last statement overrules the previous statement.

SYNTAX:

```
RADIX 1 1 1 1 1 1 1 1 1 1 1 1
VNAME V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
```

To concisely specify a range, or a bus, use square brackets and a colon. Define a range as follows:

```
VNAME vector_name[starting_index : ending_index]
```

In this range, *vector_name* is the name of the range. You can associate a single name with multiple bits (such as bus notation).

The opening and closing brackets and the colon are required; they indicate that this is a range. The vector name must correlate with the number of bits available.

You can nest the bus definition inside other grouping symbols, such as {}, (), [], and so on. The bus indices expand in the specified order.

EXAMPLE:

```
a[0:3]
```

This example represents a0, a1, a2, and a3, in that order. HSPICE does not reverse the order to make a3 the first bit.

The bit order is MSB:LSB, which means most significant bit to least significant bit. For example, you can represent a 5-bit bus such as: {a4 a3 a2 a1 a0}, using this notation: a[4:0]. The high bit is a4, which represents 2^4. It is the largest value, and therefore is the MSB.

EXAMPLE 1:

If you specify:

```
 RADIX 2 4
 VNAME VA[0:1] VB[4:1]
```

HSPICE generates voltage sources with the following names:

```
VA0 VA1 VB4 VB3 VB2 VB1
```

- *VA0* and *VB4* are the MSBs.
- *VA1* and *VB1* are the LSBs.

EXAMPLE 2:

If you specify:

```
VNAME VA[[0:1]] VB<[4:1]>
```

HSPICE generates voltage sources with the following names:

```
VA[0] VA[1] VB<4> VB<3> VB<2> VB<1>
```

EXAMPLE 3:

To specify a single bit of a bus:

```
VNAME VA[[2:2]]
```

This range creates a voltage source named:

```
VA[2]
```

EXAMPLE 4:

This example generates signals named A0, A1, A2, ... A23:

```
RADIX 444444
VNAME A[0:23]
```

IO Statement

The *io* statement defines the type, for each vector. The line starts with the *io* keyword, followed by a string of i, b, o, or u definitions. These definitions indicate whether each corresponding vector is an input (i), bidirectional (b), output (o), or unused (u) vector.

Table 5-24 IO Syntax

Parameter	**Description**
i	Input, which HSPICE uses to stimulate the circuit.
o	Expected output, which HSPICE compares with the simulated outputs.
b	Bidirectional vector.
u	Unused vector, which HSPICE ignores.

SYNTAX:

```
IO I O B U
```

EXAMPLE:

- If you do not specify the *io* statement, HSPICE assumes that all signals are input signals.
- If you define more than one *io* statement, the last statement overrules previous statements.

```
io i i i bbbb iiiioouu
```

Tunit Statement

The TUNIT statement defines the time unit in the digital vector file, for PERIOD, TDELAY, SLOPE, TRISE, TFALL, and absolute time.

SYNTAX:

```
TUNIT {fs|ps|ns|us|ms}
Default value = ns below
```

Table 5-25 TUNIT Syntax

Unit	**Description**
fs	femtosecond
ps	picosecond
ns	nanosecond
us	microsecond
ms	millisecond

- If you do not specify the *tunit* statement, the default time unit value is ns.
- If you define more than one *tunit* statement, the last statement overrules the previous statement.

EXAMPLE:

The TUNIT statement in this example specifies that the absolute times in the Tabular Data section are 11.0ns, 20.0ns, and 33.0ns.

```
TUNIT ns
11.0 1000 1000
20.0 1100 1100
33.0 1010 1001 Period and Tskip Statements
```

The PERIOD statement defines the time interval for the Tabular Data section. You do not need to specify the absolute time at every time point. If you use a PERIOD statement, without the TSKIP statement, the Tabular Data section contains only signal values, not absolute times. The TUNIT statement defines the time unit of the PERIOD.

SYNTAX:

```
PERIOD x
```

In this syntax, *x* represents the time interval.

EXAMPLE 1:

In this example:

- The first row of the tabular data (1000 1000) is at time 0ns.
- The second row (1100 1100) is at 10ns.
- The third row (1010 1001) is at 20ns.

```
radix 1111 1111
period 10
1000 1000
1100 1100
1010 1001
```

The *tskip* statement specifies to ignore the absolute time field in the tabular data. You can then keep, but ignore, the absolute time field of each row in the tabular data, when you use the *period* statement.

EXAMPLE 2:

If your netlist contains:

```
radix 1111 1111
period 10
tskip
11.0 1000 1000
20.0 1100 1100
33.0 1010 1001
```

HSPICE ignores the absolute times 11.0, 20.0 and 33.0.

You might do this, for example, if the absolute times are not perfectly periodic for testing reasons. Another reason might be that a path in the circuit does not meet timing, but you might still use it as part of a test bench. Initially, HSPICE writes to the vector file, using absolute time. After you fix the circuit, you might want to use periodic data.

Enable Statement

The ENABLE statement specifies the controlling signal(s) for bidirectional signals. All bidirectional signals require an ENABLE statement. If you specify more than one ENABLE statement, the last statement overrules the previous statement, and HSPICE issues a warning message:

```
[Warning]:[line 6] resetting enable signal to WENB for
bit ’XYZ’
```

SYNTAX:

```
ENABLE controlling_signalname mask
```

In this syntax, *controlling_signalname* and *mask* define the bidirectional signals to which ENABLE applies.

The controlling signal, for bidirectional signals, must be an input signal, with a radix of 1. The bidirectional signals become output when the controlling signal is at state 1 (or high). To reverse this default control logic, start the control signal name with a tilde (~).

EXAMPLE:

In this example, the *x* and *y* signals are bidirectional, as defined by the *b* in the *io* line.

- The first enable statement indicates that *x* (as defined by the position of *F*) becomes output, when the *a* signal is 1.
- The second enable specifies that the *y* bidirectional bus becomes output, when the *a* signal is 0.

```
radix 144
io ibb
vname a x[3:0] y[3:0]
enable a 0 F 0
enable ~a 0 0 F
```

Defining Tabular Data

Although the Tabular Data section generally appears last in a digital vector file (after the *Vector Pattern* and *Waveform Characteristics* definitions), this chapter describes it first, to introduce the definitions of a *vector*.

SYNTAX:

The Tabular Data section defines (in *tabular* format) the values of the signals, at specified times. 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...
.
.
```

In this syntax, *timex* is the specified time, and *signalx_valuex* 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_valuex* constitutes one vector. Signal values can have any of the legal states, described in the next section.

Rows in the Tabular Data section must appear in chronological order, because row placement carries sequential timing information.

EXAMPLE:

```
10.0 1000 0000
15.0 1100 1100
20.0 1010 1001
30.0 1001 1111
```

This example feature eight signals, so it has eight vectors. The first signal (starting from the left) vector is [1 1 1 1]; the second vector is [0 1 0 0]; and so on.

Input Stimuli

HSPICE converts each input signal into a PWL (piecewise linear) voltage source, and a series resistance. Table 5-26 shows the legal states for an input signal.

Table 5-26 Input Stimuli States

Value	Description
0	Drive to ZERO (gnd).
1	Drive to ONE (vdd).
Z, z	Floating to HIGH IMPEDANCE.
X, x	Drive to ZERO (gnd).
L	Resistive drive to ZERO (gnd).
H	Resistive drive to ONE (vdd).
U, u	Drive to ZERO (gnd).

- For the 0, 1, X, x, U, and u states, HSPICE sets resistance to zero.
- For L or H states, the Out /Outz Statements on page 5-92 define the resistance value.
- For Z or z states, the Triz Statement on page 5-93 defines the resistance.

Expected Output

HSPICE converts each output signal into a .DOUT statement in the netlist. During simulation, HSPICE 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:

Table 5-27 Expected Output Values

Value	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 cannot detect a high impedance state.

EXAMPLE:

- The first line of the example below is a comment line, because of the semicolon character.
- The next 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 expects the first and second vectors to be high, and the third and fourth to be low.
- At 30 time units, HSPICE expects only the first vector to be high, and all others low.
- At 35 time units, HSPICE expects the output of the first two vectors to be “don’t care”; it expects vectors 3 and 4 to be low.

```
...
IO OOOO; start of tabular section data
11.0 1 0 0 1
20.0 1 1 0 0
30.0 1 0 0 0
35.0 x x 0 0
```

Verilog Value Format

HSPICE accepts Verilog *sized* format, to specify numbers:

```
<size> '<base format> <number>
```

- `<size>` specifies the number of bits, in decimal format.
- `<base format>` indicates:
- binary (*’b* or *’B*)
- octal (*’o* or *’O*)
- hexadecimal (*’h* or *’H*).
- Valid `<number>` fields 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 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 uses 0 to extend it.

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 (see Tunit Statement on page 5-77), the Tabular Data section omits the absolute times. For more information, see Tabular Data on page 5-86.

EXAMPLE:

The PERIOD statement in this example sets the time interval to 10ns, between successive lines in the tabular data. This is a shortcut, when you use the 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
```

Tabular Data

The *Tabular Data* section defines the values of the input signals, at specified times. The first column lists the time, followed by signal values in the subsequent columns, in the order specified in the *vname* statement.

EXAMPLE:

```
11.0 1000 1000
20.0 1100 1100
33.0 1010 1001
```

This small table 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.

For more information about this section of the digital vector file, see Defining Tabular Data on page 5-81.

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. A sample Waveform Characteristics section follows:

```
TRISE 0.3 137F 0000
TFALL 0.5 137F 0000
VIH 5.0 137F 0000
VIL 0.0 137F 0000
```

The waveform characteristics options are based on a bit-mask. For example:

- 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

This section describes how to modify waveform characteristics of your circuit.

Tdelay, Idelay, and Odelay Statements

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.

- IDELAY applies to the input signals.
- ODELAY applies to the output signals.
- TDELAY applies to both input and output signals.

SYNTAX:

`TDELAY` *x*

The statement starts with a keyword (*tdelay*, *idelay*, or *odelay*), followed by a delay value (*x*), and then a *mask*. The mask defines the signals to which the delay applies. If you do not provide a mask, the delay value applies to all signals.

The *tunit* statement defines the time unit of *tdelay*, *idelay* and *odelay*. Normally, you need to use only the *tdelay* statement; use the *idelay* and *odelay* statements only to specify different input and output delay times, for bidirectional signals. HSPICE ignores *idelay* settings on output signals (or *odelay* settings on input signals), and issues a warning message.

You can specify more than one *tdelay*, *idelay*, or *odelay* statement.

- If you apply more than one *tdelay* (*idelay*, *odelay*) statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.
- If you do not specify the signal delays in a *tdelay*, *idelay*, or *odelay* statement, HSPICE defaults to zero.

EXAMPLE:

This example does not specify the `TUNIT` statement, so HSPICE uses the default, ns, as the time unit for this example. The first `TDELAY` statement indicates that all signals have the same delay time of 1.0ns. Subsequent TDELAY statements overrule the delay time of some signals.

- The delay time for the V2 and Vx signals is -1.2.
- The delay time for the V4, V5[0:1], and V6[0:2] signals is 1.5.
- The input delay time for the V7[0:3] signals is 2.0, and the output delay time is 3.0.

```
RADIX 1 1 4 1234 11111111
IO i i o iiib iiiiiiii
VNAME V1 V2 VX[3:0] V4 V5[1:0] V6[0:2] V7[0:3]
+ V8 V9 V10 V11 V12 V13 V14 V15
TDELAY 1.0
TDELAY -1.2 0 1 F 0000 00000000
TDELAY 1.5 0 0 0 1370 00000000
IDELAY 2.0 0 0 0 000F 00000000
ODELAY 3.0 0 0 0 000F 00000000
```

Slope Statement

- The *slope* statement specifies the fall times for the input signal.
- The *tunit* statement defines the time unit.

To specify the signals to which the *slope* applies, use a mask.

SYNTAX:

SLOPE *x*
Default value = 0.1ns

In this syntax, *x* is the input signal rise/fall time.

- If you do not specify the *slope* statement, the default slope value is 0.1 ns.
- If you specify more than one *slope* statement, the last statement overrules the previous statements, and HSPICE issues a warning message.

The *slope* statement has no effect on the expected output signals. You can specify the optional *trise* and *tfall* statements, to overrule the rise time and fall time of a signal.

EXAMPLE:

- In the first example, the rising and falling times of all signals are 1.2 ns.
- The second example specifies a rising/falling time of 1.1 ns for the first, second, sixth, and seventh signals.

```
SLOPE 1.2
SLOPE 1.1 1100 0110
```

Trise Statement

The TRISE statement specifies the rise time of each input signal for which the mask applies. The TUNIT statement defines the time unit of TRISE.

SYNTAX:

```
TRISE x
```

In this syntax, *x* is the input signal rise time.

- If you do not use any *trise* statement to specify the rising time of the signals, HSPICE uses the value defined in the *slope* statement.
- If you apply more than one *trise* statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning message.

TRISE statements have no effect on the expected output signals.

EXAMPLE:

In the example below, the first TRISE statement assigns a rise time of 0.3 time units to all vectors. The second statement assigns a rise time of 0.5 time units, overriding the older setting of 0.3 in at least some of the bits in vectors 2, 3, and 4 through 7. Vectors 8 through 11 have a rise time of 0.8 time units, based on the final TRISE statement.

```
TRISE 0.3
TRISE 0.5 0 1 1 137F 00000000
TRISE 0.8 0 0 0 0000 11110000
```

Tfall Statement

The TFALL statement specifies the falling time of each input signal for which the mask applies.The TUNIT statement defines the time unit of TFALL.

SYNTAX:

`TFALL` *x*

In this syntax, *x* is the input signal fall time.

- If you do not specify the falling time of the signals in a *tfall* statement, HSPICE uses the value defined in the *slope* statement.
- If you use more than one *tfall* statement for a signal, the last statement over-rules previous statements. HSPICE issues a warning.

TFALL statements have no effect on the expected output signals.

EXAMPLE:

As with the TRISE example:

- The first TFALL statement applies a 0.5 time unit fall time globally.
- The second TFALL statement applies a fall time of 0.3 time units to vectors 2, 3, and 4 through 7.
- The third TFALL statement applies a fall time of 0.9 time units to vectors 8 to 11.

```
TFALL 0.5
TFALL 0.3 0 1 1 137F 00000000
TFALL 0.9 0 0 0 0000 11110000
```

Out /Outz Statements

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.

- If you do not specify the output resistance of a signal, in an *out* (or *outz*) statement, HSPICE uses the default (zero).
- If you specify more than one *out* (or *outz*) statement for a signal, the last statement overrules the previous statements, and HSPICE issues a warning message.

The *out* (or *outz*) statements have no effect on the expected output signals.

SYNTAX:

```
OUT x (or OUTZ x)
```
Default value = 0

In this syntax, *x* is the output resistance for an input signal.

EXAMPLE:

The first OUT statement in this example creates a 15.1 ohm resistor, to place in series with all vector inputs. The next OUT statement sets the resistance to 150 ohms for vectors 1 to 3. The OUTZ statement changes the resistance to 50.5 ohms for vectors 4 through 7.

```
OUT 15.1
OUT 150 1 1 1 0000 00000000
OUTZ 50.5 0 0 0 137F 00000000
```

Triz Statement

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.

- If you do not specify the *tristate* impedance of a signal, in a *triz* statement, HSPICE assumes 1000M.
- If you apply more than one *triz* statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.

TRIZ statements have no effect on the expected output signals.

SYNTAX:

```
TRIZ x
```
Default value = 1000Meg

In this syntax, *x* is the output impedance for an input signal.

EXAMPLE:

- The first TRIZ statement sets the high impedance resistance globally, at 15.1 Mohms.
- The second TRIZ statement increases the value to 150 Mohms, for vectors 1 to 3.

- The last TRIZ statement increases the value to 50.5 Mohms, for vectors 4 through 7.

```
TRIZ 15.1Meg
TRIZ 150Meg 1 1 1 0000 00000000
TRIZ 50.5Meg 0 0 0 137F 00000000
```

VIH Statement

The VIH statement specifies the logic-high voltage, for each input signal to which the mask applies.

- If you do not specify the logic high voltage of the signals, in a *vih* statement, HSPICE assumes 3.3.
- If you use more than one *vih* statement for a signal, the last statement over-rules previous statements. HSPICE issues a warning.

VIH statements have no effect on the expected output signals.

SYNTAX:

```
VIH x
```

Default value = 3.3

In this syntax, *x* is the logic-high voltage for an input signal.

EXAMPLE:

- The first VIH statement sets all input vectors to 5V, when they are high.
- The last VIH statement changes the logic-high voltage from 5V to 3.5V, for the last eight vectors.

```
VIH 5.0
VIH 3.5 0 0 0 0000 11111111
```

VIL Statement

The VIL statement specifies the logic-low voltage, for each input signal to which the mask applies.

- If you do not specify the logic-low voltage of the signals, in a *vil* statement, HSPICE assumes 0.0.
- If you apply more than one *vil* statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.

VIL statements have no effect on the expected output signals.

SYNTAX:

```
VIL x
```

Default value = 0.0

where *x* is the logic-low voltage for an input signal.

EXAMPLE:

- The first VIL statement sets the logic-low voltage to 0V, for all vectors.
- The second VIL statement changes the voltage to 0.5, for the last eight vectors.

```
VIL 0.0
VIL 0.5 0 0 0 0000 11111111
```

VREF Statement

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.

- If you do not specify the reference voltage name of the signals, in a *vref* statement, HSPICE assumes 0.
- If you apply more than one *vref* statement, the last statement overrules the previous statements, and HSPICE issues a warning.

VREF statements have no effect on the output signals.

SYNTAX:

```
VREF vx
```

Default value = 0

where *x* is the reference voltage for each input vector.

EXAMPLE:

```
VNAME v1 v2 v3 v4 v5[1:0] v6[2:0] v7[0:3] v8 v9 v10
VREF 0
VREF 0 111 137F 000
VREF vss 0 0 0 0000 111
```

When HSPICE implements these statements into the netlist, the voltage source realizes *v1*:

```
v1 V1 0 pwl(......)
```

as well as *v2*, *v3*, *v4*, *v5*, *v6*, and *v7*.

However, *v8* is realized by

```
V8 V8 vss pwl(......)
```

v9 and *v10* use a syntax similar to v8.

VTH Statement

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.

- If you do not specify the threshold voltage of the signals, in a *vth* statement, HSPICE assumes 1.65.
- If you apply more than one *vth* statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.

VTH statements have no effect on the input signals.

SYNTAX:

```
VTH x
```

Default value = 1.65

In this syntax, *x* is the logic threshold voltage for an output signal.

EXAMPLE:

- The first VTH statement sets the logic threshold voltage at 1.75V.
- The next line changes that threshold to 2.5V, for the first 7 vectors.
- The last line changes that threshold to 1.75V, for the last 8 vectors.

```
VTH 1.75
VTH 2.5 1 1 1 137F 00000000
VTH 1.75 0 0 0 0000 11111111
```

Note: All of these examples apply the same vector pattern, and both output and input control statements, so the vectors are all bidirectional.

VOH Statement

The VOH statement specifies the logic-high voltage, for each output signal to which the mask applies.

- If you do not specify the logic-high voltage, in a *voh* statement, HSPICE assumes 3.3.
- If you apply more than one *voh* statement to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.

VOH statements have no effect on input signals.

SYNTAX:

```
VOH x
```

Default value = 3.3

x is the logic-high voltage for an output signal.

EXAMPLE:

- The first line tries to set a logic-high output voltage of 4.75V, but it is redundant.
- The second line changes the voltage level to 4.5V, for the first seven vectors.
- The last line changes the last eight vectors to a 3.5V logic-high output.

These second and third lines completely override the first VOH statement.

```
VOH 4.75
VOH 4.5 1 1 1 137F 00000000
VOH 3.5 0 0 0 0000 11111111
```

Note: If you do not define either *voh* or *vol*, HSPICE uses *vth* (default or defined).

VOL Statement

The VOL statement specifies the logic-low voltage, for each output signal to which the mask applies.

- If you do not specify the logic-low voltage, in a *vol* statement, HSPICE assumes 0.0.
- If you apply more than one *vol* statements to a signal, the last statement overrules the previous statements, and HSPICE issues a warning.

VOL statements have no effect on input signals.

Note: If you do not define either VOH or VOL, then HSPICE uses VTH (default or defined).

SYNTAX:

```
VOL x
```

Default value = 0.0

x is the logic-low voltage for an output signal.

EXAMPLE:

- The first VOL statement below sets the logic-low output to 0V.
- The second VOL statement sets the output voltage to 0.2V, for the fourth through seventh vectors.
- The last statement increases the voltage further to 0.5V, for the first three vectors.

```
VOL 0.0
VOL 0.2 0 0 0 137F 00000000
VOL 0.5 1 1 1 0000 00000000
```

Comment Lines

Any line in a vector file that begins with a semi-colon (;) is a comment line. Comments can also start at any point along a line. HSPICE ignores characters after a semi-colon.

EXAMPLE:

```
; This is a comment line
radix 1 1 4 1234 ; This is a radix line
```

Continuing a Line

As in netlists, any line in a vector file that starts with a plus sign (+) is a continuation from the previous line.

Digital Vector File Example

An example of a vector pattern definition follows:

```
; 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 ... 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,
; because HSPICE multiplies this value by the unit
; specified in the ‘tunit’ line.
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,
; because HSPICE multiplies this value by the unit
; specified in the ‘tunit’ line.
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

Parameters are similar to variables, which most programming languages use. They hold a value that you either assign when you create your circuit design, or 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.

This chapter describes how to use parameters within a HSPICE netlist:

- Using Parameters in Simulation (.PARAM)
- Using Algebraic Expressions
- Built-In Functions
- Parameter Scoping and Passing

Using Parameters in Simulation (.PARAM)

Defining Parameters

Parameters in HSPICE are names that you associate with numeric values (see .PARAM Statement on page 3-41). You can use any of these methods to define parameters:

Table 6-1 .PARAM 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, and in .PLOT, and .GRAPH statements. 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 6-8.
User-defined function	.PARAM <*MyFunc*(x, y)> = ‘Sqrt((x*x)+(y*y))’
Subcircuit default	.SUBCKT <*SubName*> <*ParamDefName*> = <*Value*> .MACRO <*SubName*> <*ParamDefName*> = <*Value*>
Predefined analysis function	.PARAM <*mcVar*> = Agauss(1.0,0.1) (see Statistical Analysis and Optimization on page 12-1).
.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 7-39).
.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. Parameter resolution order is:

1. Resolve all literal assignments.
2. Resolve all expressions.
3. Resolve all function calls.

Table 6-2 shows the parameter passing order.

Table 6-2 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:

- Later definition changes its value, or
- Algebraic expression assigns a new value during simulation.

HSPICE does not warn you, if they reassign a parameter.

SYNTAX:

```
.PARAM <ParamName> = <RealNumber>
.PARAM <ParamName> = '<Expression>' $ Quotes are mandatory
.PARAM <ParamName1> = <ParamName2> $ Cannot be recursive!
```

EXAMPLE 1:(Numerical)

```
.PARAM TermValue = 1g
   rTerm Bit0 0 TermValue
   rTerm Bit1 0 TermValue
...
```

EXAMPLE 2:(Expression)

```
.PARAM Pi             = '355/113'
.PARAM Pi2            = '2*Pi'
.PARAM npRatio        = 2.1
.PARAM nWidth         = 3u
.PARAM pWidth         = 'nWidth * npRatio'
Mp1                  ... <pModelName> W = pWidth
Mn1                  ... <nModelName> W = nWidth
...
```

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 Simulation Output on page 7-1 for more information about simulation output).

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.

The format of a function is:

```
funcname1(arg1[,arg2...]) = expression1
+ [funcname2(arg1[,arg2...]) = expression2] off
```

- 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.

Table 6-3 funcname Syntax

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.
off	Voids all user-defined functions.

EXAMPLE:

```
f(a,b) = POW(a,2)+a*b g(d) = SQRT(d)
+ h(e) = e*f(1,2)-g(3)
```

SYNTAX:

```
.PARAM <ParamName>(<pv1>[, <pv2>...]) = '<Expression>'
```

EXAMPLE:

```
.PARAM CentToFar (c)  = '(((c*9)/5)+32)'
.PARAM F(p1,p2) = 'Log(Cos(p1)*Sin(p2))'
.PARAM SqrdProd (a,b) = '(a*a)*(b*b)'
```

Subcircuit Default Parameter Definitions

When you use hierarchical sub-circuits, you can pick default values for circuit elements. You typically use defaults in cell definitions, to simulate the circuit using typical values (see Using Subcircuits on page 3-57).

SYNTAX:

```
.SUBCKT <SubName> <PinList> [<SubDefaultsList>]
```

In this syntax, <*SubDefaultsList*> is <*SubParam1*> = <*Expression*> [<*SubParam2*> = <*Expression*> ...]

EXAMPLE:

This example implements an inverter that uses a *Strength* parameter. By default, the inverter can drive three devices. Enter a new value for the *Strength* parameter in the element line, to select larger or smaller inverters for the application.

```
.SUBCKT Inv a y Strength = 3
   Mp1 <MosPinList> pMosMod L = 1.2u W = 'Strength * 2u'
   Mn1 <MosPinList> nMosMod L = 1.2u W = 'Strength * 1u'
.ENDS
```

```
...
xInv0 a y0 Inv              $ Default devices: p
device = 6u,
                            $ n device = 3u
xInv1 a y1 Inv Strength = 5 $ p device = 10u, n device = 5u
xInv2 a y2 Inv Strength = 1 $ p device =  2u, n device = 1u
...
```

Predefined Analysis Function

HSPICE includes specialized analysis types, such as Optimization and Monte Carlo, that require a way to control the analysis. For definitions of the parameters for these analysis types, see Statistical Analysis and Optimization on page 12-1.

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 7-39.

.PRINT|.PROBE|.PLOT|.GRAPH Parameters

.PRINT|.PROBE|.PLOT|.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|.GRAPH statement, not in a .PARAM statement.

EXAMPLE:

```
.print p1 = 3
.print p2 = par("p1*5")
```

You can use p1 and p2 as parameters in netlist. The p1 value is 3; the p2 value is 15.

For more information about the .PRINT|.PROBE|.PLOT|.GRAPH statements, see Displaying Simulation Results on page 7-4.

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 3-58.

Using Algebraic Expressions[1]

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’
```

[1] *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).*

- 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

In addition to simple arithmetic operations (+, -, *, /), HSPICE provides several built-in functions, listed in Table 6-4, that you can use in expressions:

Table 6-4 Synopsys HSPICE Built-in Functions (Sheet 1 of 3)

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)$\|x\|^y$
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|)

Table 6-4 Synopsys HSPICE Built-in Functions (Sheet 2 of 3)

HSPICE Form	Function	Class	Description
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)$20\log_{10}(\lvert x\rvert)$
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)$\lvert x\rvert$
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.
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 7-38 for more information.

Table 6-4 Synopsys HSPICE Built-in Functions (Sheet 3 of 3)

HSPICE Form	Function	Class	Description
v(<*Node*>), i(<*Element*>)...	circuit output variables	various	Returns various circuit values during simulation. See DC and Transient Output Variables on page 7-23 for more information.
[*cond*] ?x : y	ternary operator		Returns *x* if *cond* is not zero. Otherwise, returns *y*. .para x=[condition] ?y:z
<	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)
>=	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 6-5 on page 6-13, for use in elements such as E, G, R, C, and L. You cannot use them for any other purpose in your netlist (for example, in .PARAM statements).

Table 6-5 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.
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 sub-circuits 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 6-6 compares the three primary methods for configuring libraries, to achieve required parameter checking for default MOS transistor widths.

Table 6-6 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.
Special	.OPTION DEFW statement	Simple to do.	Third-party libraries, or other sections of the design, might depend on the DEFW option.

Parameter Defaults and Inheritance

Use the .OPTION PARHIER parameter to specify scoping rules.

```
.OPTION PARHIER = < GLOBAL | LOCAL >
```

The default setting is GLOBAL, which uses the same scoping rules that HSPICE used before Release 95.1.

EXAMPLE:

The following example explicitly shows the difference between local and global scoping, for using parameters in sub-circuits.

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 the
.measure tran Wid1 param = ’lv2(xInv1.Mp1)’$ 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:

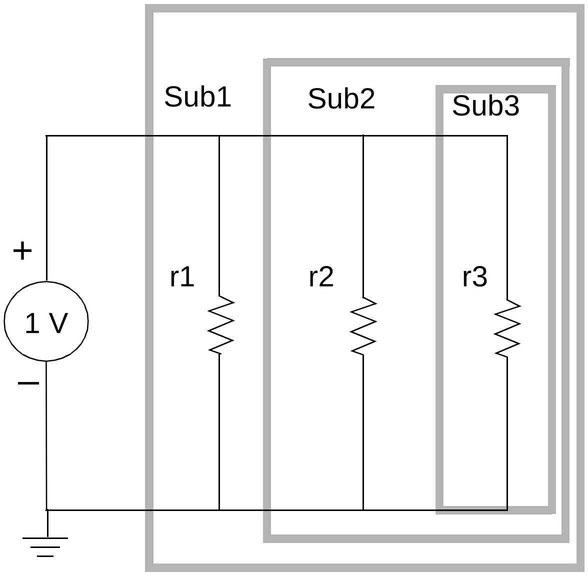
```
wid0   =    2.0000E-06
wid1   =    5.0000E-06
```

Parameter Passing

Figure 6-1 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 6-1   Hierarchical Parameter Passing Problem



```
TEST OF PARHIER
.OPTION list node post = 2
+ ingold = 2
+ parhier = <Local|Global>
.PARAM Val = 1
x1 n0 0 Sub1
.SubCkt Sub1 n1 n2 Val = 1
    r1 n1 n2 Val
    x2 n1 n2 Sub2
.Ends Sub1
.SubCkt Sub2 n1 n2 Val = 2
    r2 n1 n2 Val
    x3 n1 n2 Sub3
.Ends Sub2
.SubCkt Sub3 n1 n2 Val = 3
    r3 n1 n2 Val
.Ends Sub3
.OP
.END
```

The total resistance of the chain has two possible solutions: 0.3333 Ω and 0.5455Ω .

You can use the PARHIER option 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 0.3333 Ω Under local scoping rules, the lower level assignments prevail, and the total is 0.5455 Ω (one, two, and three ohms in parallel).

The example in Figure 6-1 on page 6-20 produces the results in Table 6-7, based on how you set the local/global PARHIER option:

Table 6-7 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

Use output format statements and variables to display steady state, frequency, and time domain simulation results. You can also use these 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.

This chapter explains the following topics:

Overview of Output Statements

Output Commands

The input netlist file contains output statements, including .PRINT, .PLOT, .GRAPH, .PROBE, .MEASURE, and .DOUT. 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 AvanWaves interface files.

AvanWaves provides high-resolution, post-simulation, and interactive display of waveforms.

Table 7-1 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)	Generates low-resolution (ASCII) plots in the output listing file (and post-processor data, if you specify .OPTION POST), in HSPICE.
.GRAPH (HSPICE only)	Generates high-resolution plots, for specific printing devices (such as HP LaserJet), or in PostScript format (intended for hard-copy outputs, without a 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. You can use the .MEASURE statement in HSPICE.
.DOUT	Specifies the expected final state of an output signal.
.STIM	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 uses the following output variables:

- DC and transient analysis
- AC analysis
- element template
- .MEASURE statement
- parametric analysis

For HSPICE, *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.

Parametric analysis variables are mathematical expressions, which operate on nodal voltages, branch currents, element template variables, or other parameters that you specify. Use these variables when you run behavioral analysis of simulation results. See Using Algebraic Expressions on page 6-8.

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 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 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.

SYNTAX:

```
.PRINT antype ov1 <ov2 … >
```

Table 7-2 .PRINT Syntax

Parameter	Description
antype	Type of analysis for outputs. Antype is one of the following types: DC, AC, TRAN, NOISE, or DISTO.
ov1 …	Output variables to print. These are voltage, current, or element template variables, from a DC, AC, TRAN, NOISE, or DISTO analysis.

You can include wildcards in .PRINT statements. See Using Wildcards on page 2-2.

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, and m4), then .print iall (m*) is equivalent to .print i(m1) i(m2) i(m3) i(m4), and prints the output currents of all four MOSFET elements.

Statement Order

HSPICE 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.

EXAMPLE 1:

```
CASE 1
.print v(din) i(mxn18)
.dc vdin 0 5.0 0.05
.tran 1ns 60ns
```

```
CASE 2
.dc vdin 0 5.0 0.05
.tran 1ns 60ns
.print v(din) i(mxn18)
```

```
CASE 3
.dc vdin 0 5.0 0.05
.print v(din) i(mxn18)
.tran 1ns 60ns
```

- If you replace the .print statement with:

  ```
  .print TRAN v(din) i(mnx)
  ```

  then all three cases have identical .sw0 and .tr0 files.

- If you replace the .print statement with:

  ```
  .print DC v(din) i(mnx)
  ```

  then the .sw0 and .tr0 files are different.

EXAMPLE 2:

```
.PRINT TRAN V(4) I(VIN) PAR('V(OUT)/V(IN)')
```

This example prints the results of a transient analysis, for the nodal voltage named 4. It also prints the current, through the voltage source named VIN. It also prints the ratio of the nodal voltage at the OUT and IN nodes.

EXAMPLE 3:

```
.PRINT AC VM(4,2) VR(7) VP(8,3) II(R1)
```

- Depending on the value of the ACOUT option, VM(4,2) prints the AC magnitude of the voltage difference, or the difference of the voltage magnitudes, between nodes 4 and 2.
- VR(7) prints the real part of the AC voltage, between node 7 and ground.
- Depending on the ACOUT value, VP(8,3) prints the phase of the voltage difference between nodes 8 and 3, or the difference of the phase of voltage at node 8 and voltage at node 3.
- II(R1) prints the imaginary part of the current, through R1.

EXAMPLE 4:

```
.PRINT AC ZIN YOUT(P) S11(DB) S12(M) Z11(R)
```

This example prints:

- The magnitude of the input impedance.
- The phase of the output admittance.
- Several *S* and *Z* parameters.

This statement accompanies a network analysis, using the .AC and .NET analysis statements.

EXAMPLE 5:

```
.PRINT DC V(2) I(VSRC) V(23,17) I1(R1) I1(M1)
```

This example prints the DC analysis results for several different nodal voltages and currents, through:

- The resistor named R1.
- The voltage source named VSRC.
- The drain-to-source current of the MOSFET named M1.

EXAMPLE 6:

```
.PRINT NOISE INOISE
```

This example prints the equivalent input noise.

EXAMPLE 7:

```
.PRINT DISTO HD3 SIM2(DB)
```

This example prints the magnitude of third-order harmonic distortion, and the decibel value of the intermodulation distortion sum, through the load resistor that you specify in the .DISTO statement.

EXAMPLE 8:

```
.PRINT AC INOISE ONOISE VM(OUT) HD3
```

This statement includes NOISE, DISTO, and AC output variables in the same .PRINT statement in HSPICE.

EXAMPLE 9:

```
.PRINT pj1 = par(‘p(rd) +p(rs)‘)
```

This statement prints the value of pj1, with the specified function.

Note: HSPICE ignores .PRINT statement references to nonexistent netlist part names, and prints those names in a warning.

EXAMPLE 10:

Derivative function:

```
   .PRINT der=deriv(’v(NodeX)’)
```

Integrate function:

```
   .PRINT int = integ(’v(NodeX)’)
```

The parameter can be a node voltage, or a reasonable expression.

.PLOT Statement

The .PLOT statement plots the output values of one or more variables, in a selected HSPICE analysis. Each .PLOT statement defines the contents of one plot, which can contain more than one output variable.

If you do not specify plot limits, HSPICE determines the minimum and maximum values of each output variable that it plots, and scales each plot to fit common limits. To force HSPICE to set limits for certain variables, set the plot limits to (0,0) for the variables.

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 the CO (columns out) option. If you set CO to 80, the plot has 50 columns. If CO is 132, the plot has 100 columns.

You can include wildcards in .PLOT statements. See Using Wildcards on page 2-2.

SYNTAX:

```
.PLOT antype ov1 <(plo1,phi1)> <ov2> <(plo2,phi2)> ...>
```

Table 7-3 .PLOT Syntax

Parameter	Description
antype	Type of analysis for the specified plots. Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
ov1 …	Output variables to plot: voltage, current, or element template, from a DC, AC, TRAN, NOISE, or DISTO analysis. See the next sections for syntax.
plo1, phi1 …	Lower and upper plot limits. The plot for each output variable uses the first set of plot limits, after the output variable name. Set a new plot limit for each output variable, after the first plot limit. For example, to plot all output variables that use the same scale, specify one set of plot limits at the end of the .PLOT statement. If you set the plot limits to (0,0) HSPICE automatically sets the plot limits.

EXAMPLE:

In the following example, PAR plots the ratio of the collector current and the base current, for the Q1 transistor.

```
.PLOT DC V(4) V(5) V(1) PAR(‘I1(Q1)/I2(Q1)’)
.PLOT TRAN V(17,5) (2,5) I(VIN) V(17) (1,9)
.PLOT AC VM(5) VM(31,24) VDB(5) VP(5) INOISE
```

The second of the two above examples uses the VDB output variable to plot AC analysis results (in decibels), for node 5. The AC plot can include NOISE results and other variables that you specify.

```
.PLOT AC ZIN YOUT(P) S11(DB) S12(M) Z11(R)
.PLOT DISTO HD2 HD3(R) SIM2
.PLOT TRAN V(5,3) V(4) (0,5) V(7) (0,10)
.PLOT DC V(1) V(2) (0,0) V(3) V(4) (0,5)
```

In the last example above, HSPICE sets the plot limits for V(1) and V(2), but you specify 0 and 5 volts as the plot limits for V(3) and V(4).

.PROBE Statement

The .PROBE statement saves output variables into interface and graph data files. HSPICE 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. See Using Wildcards on page 2-2.

SYNTAX:

```
.PROBE antype ov1 <ov2 ...>
```

Table 7-4 .PROBE Syntax

Parameter	Description
antype	Type of analysis for the specified plots. Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
ov1 …	Output variables to plot. These are voltage, current, or element template variables from a DC, AC, TRAN, NOISE, or DISTO analysis. A .PROBE statement can include more than one output variable.

EXAMPLE:

```
.PROBE DC V(4) V(5) V(1) beta = PAR(‘I1(Q1)/I2(Q1)’)
```

EXAMPLE 2:

Derivative function:

```
   .PROBE der=deriv(’v(NodeX)’)
```

Integrate function:

```
   .PROBE int = integ('v(NodeX)')
```

The parameter can be a node voltage, or a reasonable expression.

.GRAPH Statement

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.

This 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).

Each .GRAPH statement creates a new .gr# file, where # ranges first from 0 to 9, and then from a to z. You can create up to 36 graph files. If you specify more than 36 .GRAPH statements, HSPICE overwrites the graph files, starting with the .gr0 file.

To overcome this limitation, use ALT999 or ALT9999. These options extend the number of digits allowed in the file name extension, to either .gr### (ALT999) or .gr#### (ALT9999), where # ranges from 0 to 9.

You can include wildcards in .GRAPH statements. See Using Wildcards on page 2-2.

SYNTAX:

```
.GRAPH antype <MODEL = mname> <unam1 = > ov1,
+ <unam2 = >ov2 ... <unamn = >ovn (plo,phi)
```

Table 7-5 .GRAPH Syntax

Parameter	Description
antype	Type of analysis for the specified plots (outputs). Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
mname	Plot model name, referenced in the .GRAPH statement. Use .GRAPH and its plot name to create high-resolution plots directly from HSPICE.
unam1…	You can define output names, which correspond to the ov1 ov2 … output variables (*unam1 unam2 ...*), and use them as labels, instead of output variables, for a high resolution graphic output.
ov1 …	Output variables to print. Can be voltage, current, or element template variables, from a different type of analysis. You can also use algebraic expressions as output variables, but you must define them inside the PAR( ) statement.
plo, phi	Lower and upper plot limits. Set the plot limits only at the end of the .GRAPH statement.

.MODEL Statement for .GRAPH

This section describes the model statement for .GRAPH in HSPICE.

SYNTAX:

```
.MODEL mname PLOT (pnam1 = val1 pnam2 = val2….)
```

Table 7-6 .MODEL Syntax for .GRAPH

Parameter	Description
mname	Plot model name, referenced in .GRAPH statements
PLOT	Keyword for a .GRAPH statement model
pnam1 = val1…	Each .GRAPH statement model includes several model parameters. If you do not specify model parameters, HSPICE uses the default values of the model parameters, described in the following table. Pnamn is one of the model parameters of a .GRAPH statement, and valn is the value of pnamn. Valn can be more than one parameter.

EXAMPLE:

```
.GRAPH DC cgb = lx18(m1) cgd = lx19(m1) cgs = lx20(m1)
.GRAPH DC MODEL = plotbjt
+ model_ib = i2(q1)    meas_ib = par(ib)
+ model_ic = i1(q1)    meas_ic = par(ic)
+ model_beta = par(’i1(q1)/i2(q1)’)
+ meas_beta = par(’par(ic)/par(ib)’)(1e-10,1e-1)
.MODEL plotbjt PLOT MONO = 1 YSCAL = 2 XSCAL = 2
+ XMIN = 1e-8 XMAX = 1e-1
```

Table 7-7 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 “Modeling Filters and Networks” in the *HSPICE Applications Manual*).
XGRID, YGRID	0.0	Set these values to 1.0, to turn on the axis grid lines.
XMIN, XMAX	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)

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.

Table 7-7 Model Parameters (Continued)

Name (Alias)	**Default**	**Description**

YSCAL	1.0	Scale for the y-axis. Two common axis scales are:
		Linear(LIN)(YSCAL = 1)
		Logarithm(LOG)(YSCAL = 2)

Using Wildcards in PRINT, PROBE, PLOT, and GRAPH Statements

You can include wildcards in .PRINT and .PROBE statements, and in .PLOT and .GRAPH statements.

EXAMPLE:

```
 * test wildcard
.option post=2
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:
*.plot v(n20) v(n21)
.plot v(n2?)

***Wildcard equivalent for:
*.graph v(n20, 1) v(n21, 1)
.graph v(n2*, 1)
.end
```

Supported wildcard characters are:

? Matches any single character that HSPICE supports.
* Matches zero or more characters that HSPICE supports.

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 Selecting Simulation Output Parameters on page 7-23.

EXAMPLE:

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

.OPTION CO for Printout Width

The number of output variables that print on a single line of output, is a function of the number of columns, which you use the CO option to set in HSPICE.

Typical values are CO = 80 (the default) for narrow printouts, and CO = 132 for wide printouts. You can set up to five output variables per 80-column output, and up to eight output variables per 132-column output, with twelve characters per column. HSPICE automatically creates additional print statements and tables, for all output variables beyond the number that the CO option specifies.

.WIDTH Statement

You can use the .WIDTH statement to define the print-out width in HSPICE.

SYNTAX:

```
.WIDTH OUT = {80 |132}
```

where *OUT* is the output print width

EXAMPLE:

```
.WIDTH OUT = 132 $ SPICE compatible style
.OPTION CO = 132  $ preferred style
```

Permissible values for OUT are 80 and 132. You can also use the CO option to set the OUT value.

.OPTION ALT999 or ALT9999, to Extend Output File Name

The output files for a postprocessor (from .OPTION POST in HSPICE) or .GRAPH statements have unique extensions .xx#:

- *xx* is a two-character text string, to denote the output type (see Simulation Output on page 7-1 for more information).
- # is an alphanumeric character, that denotes the `.ALTER` number of the current simulation. This limits the total number of `.ALTER` statements in a netlist to 36, before the outputs begin overwriting the current files.

The ALT999 and ALT9999 options extend the output file name suffix to .xx### and .xx####, respectively, where # represents a numerical character (0 to 9) only. Use this syntax to include up to 1000 or 10,000 .ALTER statements in the input netlist, which creates a unique file name for each output file.

.OPTION INGOLD for Printout Numerical Format

By default, HSPICE prints variable values in engineering notation:

```
F = 1e-15    M = 1e-3
P = 1e-12    K = 1e3
N = 1e-9     X = 1e6
U = 1e-6     G = 1e9
```

In contrast to exponential form, engineering notation provides two to three extra significant digits, and aligns columns to facilitate comparison. To obtain output in exponential form, specify INGOLD = 1 or 2, with an .OPTION statement.

Table 7-8 .OPTION INGOLD Syntax

Value	Description	Defaults
INGOLD = 0 (default)	Engineering Format	1.234K 123M
INGOLD = 1	G Format (fixed and exponential)	1.234e+03 .123
INGOLD = 2	E Format (exponential SPICE)	1.234e+03 .123e-1

.OPTION POST for High Resolution Graphics

Use an .OPTION POST statement to display high-resolution AvanWaves plots of simulation results, on either a graphics terminal or a high-resolution laser printer. Use .OPTION POST to provide output, without specifying other parameters. POST has defaults, which supply usable data to most parameters.

Table 7-9 .OPTION POST Syntax

Value	Description
POST = 0,1,BINARY	Output format is binary.
POST = 2,ASCII	Output format is ASCII.
POST = 3	Output format is New Wave binary.

.OPTION ACCT Summary of Job Statistics

The ACCT option in HSPICE generates a detailed accounting report:

Table 7-10 .OPTION ACCT Syntax

Value	Description
.OPTION ACCT	Enables reporting.
.OPTION ACCT = 1 (default)	Is the same as ACCT, without arguments.
.OPTION ACCT = 2	Enables reporting, and matrix statistic reporting.

EXAMPLE:

The following output example appears at the end of an output listing.

```
**** job statistics summary tnom = 25.000 temp = 25.000
# nodes = 15 # elements = 29 # real*8
mem avail/used = 333333/13454
# diodes =  0 # bjts = 0 # jfets = 0 # mosfets = 24
```

```
   analysis   time   # points  tot. iter   conv.iter
   op point   0.24   1         11
   transient  5.45   161       265         103 rev = 1
   pass1      0.08
   readin     0.12
   errchk     0.05
   setup      0.04
   output     0.00
```

The analysis time includes the following time statistics:

```
   load      5.22
   solver    0.16
# external nodes = 15 # internal nodes = 0
# branch currents = 5 total matrix size = 20
pivot based and non pivoting solution times
non pivoting: ---- decompose 0.08 solve 0.08
matrix size(109) = initial size(105) + fill(4)
words copied =  111124
total cpu time 6.02 seconds
job started at 11:54:11 21-sep92
job ended   at 11:54:36 21-sep92
```

The definitions for the items in the previous listing follow:

Table 7-11   Output Example Syntax

Parameter	**Description**
# BJTS	Number of bipolar transistors in the circuit.
# ELEMENTS	Total number of elements.
# JFETS	Number of JFETs in the circuit.
# MOSFETS	Number of MOSFETs in the circuit.
# NODES	Total number of nodes.
# POINTS	Number of transient points set in the .TRAN statement. JTRFLG is usually at least 50, unless you set the DELMAX option.

Table 7-11   Output Example Syntax (Continued)

Parameter	Description
CONV.ITER	Number of points that the simulator needs, to preserve the accuracy that the tolerances specify.
DC	DC operating point analysis time, and number of iterations required. ITL1 sets the maximum number of iterations.
ERRCHK	Part of the input processing.
MEM +	Amount of workspace available, and amount used in simulation.
AVAILUSED	Measured in 64-bit (8-byte) words.
OUTPUT	Time required, to process all prints and plots.
LOAD	Constructs the matrix equation.
SOLVER	Solves equations.
PASS1	Part of the input processing.
READIN	Input reader reads the user data file and any additional library files, and generates an internal representation of the information.
REV	Number of times that the simulator had to cut time (reversals). This measures how difficult the design is to simulate.
SETUP	Constructs a sparse matrix pointer system.
TOTAL JOB TIME	Total CPU time required, to process the simulation. This is not the amount of actual (clock) time used to simulate, and can differ slightly from run to run, even if the runs are identical.

The ratio of TOT.ITER to CONV.ITER is the best measure of simulator efficiency. The theoretical ratio is 2:1. In this example the ratio was 2.57:1. SPICE generally has a ratio from 3:1 to 7:1.

In transient analysis, the ratio of CONV.ITER to # POINTS is the measure of the number of points evaluated, to the number of points printed. If this ratio is greater than about 4:1, the convergence and time step control tolerances might be too tight for the simulation.

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 for .PRINT.

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 .PRINT statement above 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
```

This 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 statement prints the voltage on the N1 internal node of the X2 instance.

```
.PRINT i(X1.M1) $ Print statement 5
```

The print statement above 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)
```

Table 7-12 .PRINT Voltages

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.
- Element templates. 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 7-16 summarizes the codes that you can use, to specify the element templates for output in HSPICE.

Nodal Voltage Syntax

```
V (n1<,n2>)
```

Table 7-13 Nodal Voltage Syntax

Parameter	Description
n1, n2	HSPICE prints or plots the voltage difference (*n1*-*n2*) between the specified nodes. If you omit *n2*, HSPICE prints or plots the voltage difference between *n1* and ground (node 0).

Current: Voltage Sources

SYNTAX:

```
I (Vxxx)
```

Table 7-14 Current Source Syntax

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(X1.Vxxx).

EXAMPLE:

```
.PLOT TRAN I(VIN)
.PRINT DC I(X1.VSRC)
.PLOT DC I(XSUB.XSUBSUB.VY)
```

Current: Element Branches

SYNTAX:

```
In (Wwww)
Iall (Wwww)
```

Table 7-15 Element Branch Syntax

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 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 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 7-1 through Figure 7-6 is defined in terms of arrow notation (current direction), and node position number (terminal type).

Figure 7-1 Resistor (node1, node2)

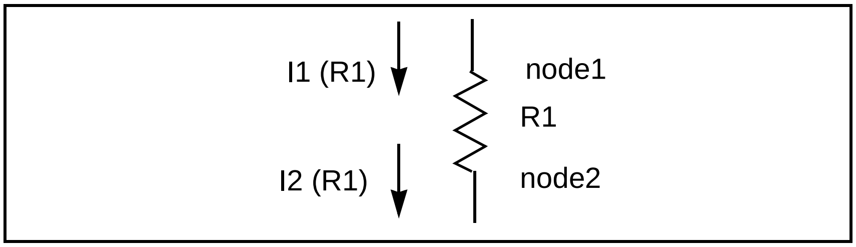


Figure 7-2 Capacitor (node1, node2); Inductor (node 1, node2)

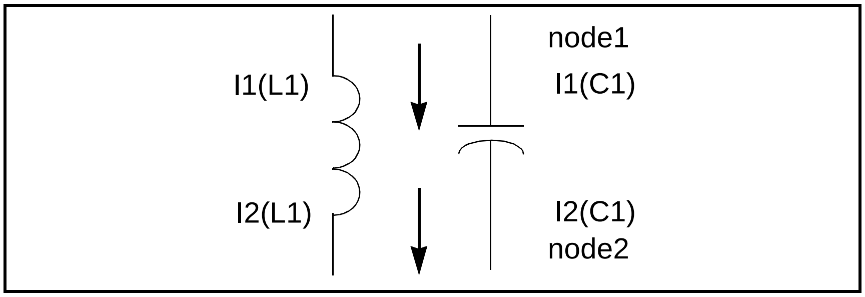


Figure 7-3 Diode (node1, node2)

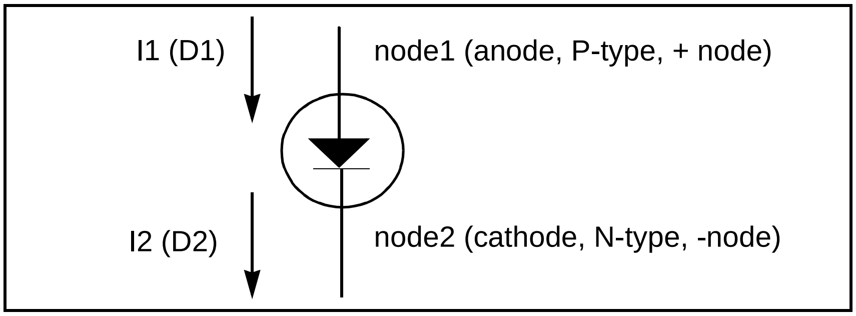


Figure 7-4 JFET (node1, node2, node3) - n-channel

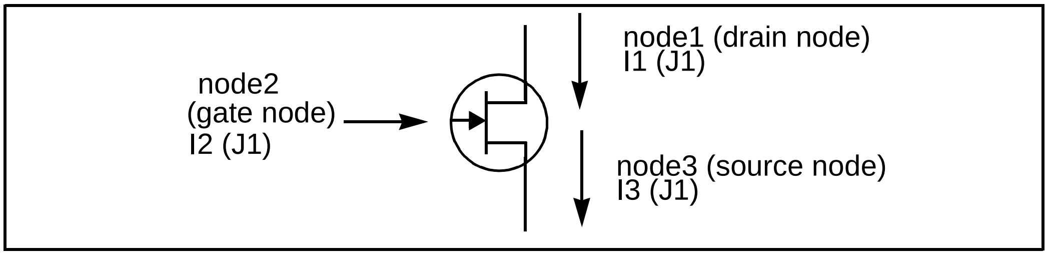


Figure 7-5   BJT (node1, node2, node3, node4) - npn

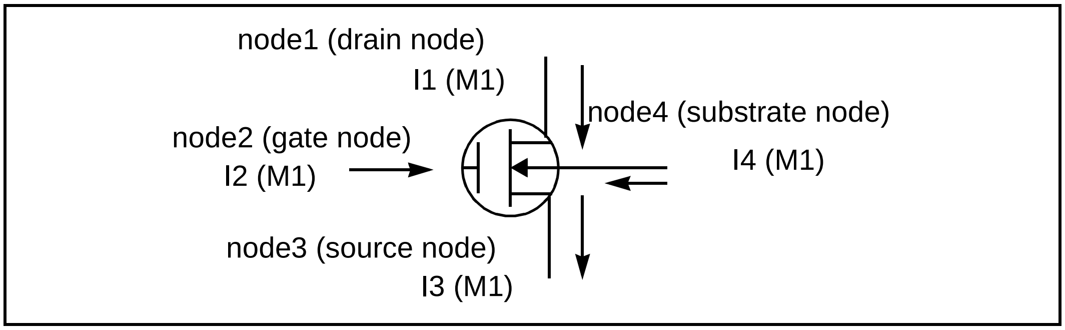
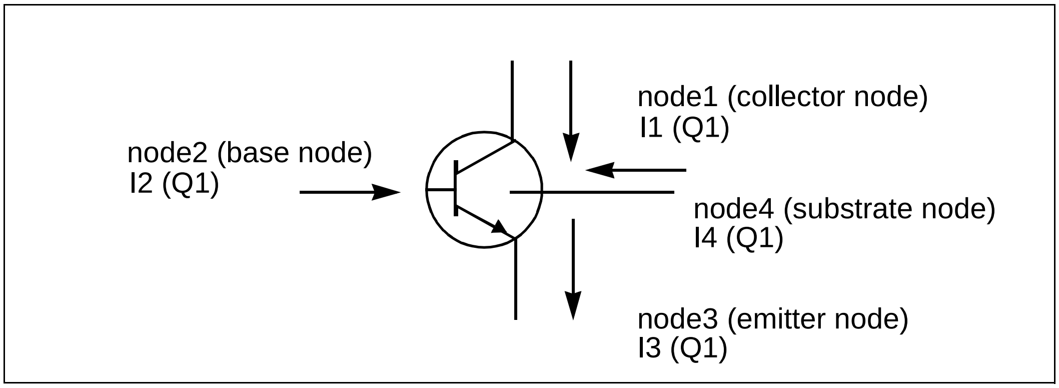


Figure 7-6   MOSFET (node1, node2, node3, node4) - n-channel



Power Output

For power calculations, HSPICE 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 multiplies the voltage across an element, and its corresponding branch current.

However, for semiconductor devices, HSPICE 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 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 also computes the power dissipation for each subcircuit.

Note: For the total power (dissipated power + stored power), HSPICE does not add the power of each independent source (voltage and current sources).

Print or Plot Power

To output the instantaneous element power, and the total power dissipation, use a .PRINT or .PLOT statement in HSPICE.

SYNTAX:

```
.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.

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

$$Pd = Vpp' \cdot (Ido + Icap) + Vp'n \cdot Ido$$

Table 7-16 Diode Power Dissipation Syntax

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

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vbb' \cdot Ibtot + Vee' \cdot Ietot + \\ Vsb' \cdot Iso - Vbb' \cdot Istot$$

Table 7-17 BJT Power Dissipation Syntax

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.

Table 7-17 BJT Power Dissipation Syntax (Continued)

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

$$\begin{aligned} Pd = {} & Vd's' \cdot Ido + Vgd' \cdot Igdo + Vgs' \cdot Igso + \\ & Vs's \cdot (Ido + Igso + Icgs) + Vdd' \cdot (Ido - Igdo - Icgd) \end{aligned}$$

Table 7-18 JFET Power Dissipation Syntax

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.

$$\begin{aligned} Pd = {} & Vd's' \cdot Ido + Vbd' \text{Þ} Ibdo + Vbs' \cdot Ibso + \\ & Vs's \cdot (Ido + Ibso + Icbs + Icgs) + Vdd' \cdot (Ido - Ibdo - Icbd - Icgd) \end{aligned}$$

Table 7-19 MOSFET Power Dissipation Syntax

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.
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.
- Element branch current.

- Impedance (Z), admittance (Y), hybrid (H), and scattering (S) parameters.
- Input and output impedance, and admittance.

Table 7-20 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 7-20 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 Voltage

SYNTAX:

```
Vx (n1,<,n2>)
```

Table 7-21 Nodal Voltage Syntax

Parameter	**Description**
x	Specifies the voltage output type (see Table 7-20 on page 7-32])
n1, n2	Specifies node names. If you omit *n2*, HSPICE assumes ground (node 0).

EXAMPLE 1:

```
.PLOT AC VM(5) VDB(5) VP(5)
```

The preceding example 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.

To produce complex results, an AC analysis uses either the SPICE or HSPICE method, and the 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 example defines the AC analysis output variables for the HSPICE method, and then for the SPICE method.

EXAMPLE 2:: HSPICE Method (ACOUT = 1, Default)

- 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,N2) =  20 · LOG10(VM(N1,0)/VM(N2,0))
```

EXAMPLE 3: SPICE Method (ACOUT = 0)

- Real and imaginary:

```
VR(N1,N2)  =  REAL [V(N1,0) - V(N2,0)]
VI(N1,N2)  =  IMAG [V(N1,0) - V(N2,0)]
```

- Magnitude:

$$VM(N1,N2) = [VR(N1,N2)^{2}+VI(N1,N2)^{2}]^{0.5}$$

- Phase:

```
VP(N1,N2)  =  ARCTAN[VI(N1,N2)/VR(N1,N2)]
```

- Decibel:

```
VDB(N1,N2) =  20 · LOG10[VM(N1,N2)]
```

Current: Independent Voltage Sources

SYNTAX:

I*z* (*Vxxx*)

Table 7-22 Independent Voltage Source Syntax

Parameter	Description
z	Current output type (see Table 7-20 on page 7-32]).
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, IM(X1.Vxxx).

EXAMPLE:

```
.PLOT AC IR(V1) IM(VN2B) IP(X1.X2.VSRC)
```

Current: Element Branches

SYNTAX:

```
Izn (Wwww)
```

Table 7-23   Element Branch Syntax

Parameter	**Description**
z	Current output type (see Table 7-20 on page 7-32]).
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 prints the magnitude value, IMn(Xxxx).

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 uses the difference method to compute TD:

$$\mathrm{TD} = -\frac{1}{360} \cdot \frac{(\mathrm{phase2} - \mathrm{phase1})}{(\mathrm{f2} - \mathrm{f1})}$$

where 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.

EXAMPLE:

```
INTEG.SP ACTIVE INTEGRATOR
****** INPUT LISTING
******
V1    1     0      .5      AC1
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*)

Table 7-24   Network Syntax

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 7-20 on page 7-32]). If you omit z, HSPICE 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)
```

Noise and Distortion

This section describes the variables used for noise and distortion analysis.

SYNTAX:

ovar <(*z*)>

Table 7-25   Noise and Distortion Syntax

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 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).

```
.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.

Element Template Output

.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.

SYNTAX:

```
Elname:Property
```

Table 7-26 Element Template Syntax

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)
```

Table 7-27 Element Template Alias Syntax

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
```

Specifying User-Defined Analysis (.MEASURE)

Use the .MEASURE statement to modify information, and to define the results of successive HSPICE simulations. The .MEASURE statement prints user-defined electrical specifications of a circuit. Optimization uses .MEASURE statements extensively. The specifications include:

- propagation
- delay
- rise time
- fall time
- peak-to-peak voltage
- minimum and maximum voltage over a specified period
- other user-defined variables

You can also use .MEASURE with either the error function or GOAL parameter, to optimize circuit component values, and to curve-fit measured data to model parameters.

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. See Input and Output Options on page 8-34 for more information about the INTERP option.

The .MEASURE statement can use several different formats, depending on the application. You can use it for either DC sweep, AC, or transient analysis.

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 writes 0.0e0 in the .mt# file as the .MEASURE result, and writes FAILED in the output listing file. Use the MEASFAIL option to write results to the .mt#, .ms#, or .ma# files. See Input and Output Options on page 8-34 for information about the MEASFAIL option.

To control the output variables, listed in .MEASURE statements, use the .PUTMEAS option. See Input and Output Options on page 8-6 for more information.

.MEASURE Performance

If you specify a large number of .measure statements, HSPICE might not complete for several minutes, or several hours. Overall simulation run time depends on the number of .measure statements to process for each iteration, and the number of iterations required to achieve convergence.

To reduce simulation run time, place similar variables together, when you list them in the .measure statement.

EXAMPLE 1:

Original Case (Slower, due to repeated switching between the v1 and v2 variables):

```
.meas tran val1 AVG v(1) FROM=0ms TO=50ms
.meas tran val2 AVG v(2) FROM=0ms TO=50ms
.meas tran val3 AVG v(1) FROM=50ms TO=100ms
.meas tran val4 AVG v(2) FROM=50ms TO=100ms
```

EXAMPLE 2:

Improved Case (Faster):

```
.meas tran val1 AVG v(1) FROM=0ms TO=50ms
.meas tran val3 AVG v(1) FROM=50ms TO=100ms
.meas tran val2 AVG v(2) FROM=0ms TO=50ms
.meas tran val4 AVG v(2) FROM=50ms TO=100ms
```

The second example lists all V(1) variables consecutively, followed by all v(2) variables. In this second case, HSPICE applies all measurements to a single variable (v1) at the same time. This reduces overall simulation run time, compared to switching back to the same variable repeatedly, when you do not sort the .measure list by variable name.

To automatically sort large numbers of .measure statements in this way, use the .option meassort statement.

SYNTAX:

```
.option meassort=0 (default; does not sort .measure statements)
.option meassort=1 (internally sorts .measure statements)
```

Set this option to 1 only if you use a large number of .measure statements, where you need to list similar variables together, to reduce simulation run time. For a small number of .measure statements, turning on internal sorting might slow-down the simulation while sorting, compared to not sorting first.

.MEASURE Parameter Types

Measurement parameter results produced by .PARAM statements in .SUBCKT blocks produce measurement results, but you cannot use those results 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 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 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:

The following example illustrates how HSPICE 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
```

not a value derived from a measured value in transient analysis.

.MEASURE Statement: Rise, Fall, and Delay

Use this format to measure independent-variable (time, frequency, or any parameter or temperature) differential measurements such as rise time, fall time, slew rate, or any measurement that requires determining independent variable values. The format specifies TRIG and TARG substatements. These two statements specify the beginning and end of a voltage or current amplitude measurement.

The rise, fall, and delay measurement mode computes the time, voltage, or frequency between a trigger value and a target value. Examples for transient analysis include rise/fall time, propagation delay, and slew rate measurement. Applications for AC analysis are the measurement of the bandwidth of an amplifier or the frequency at which a certain gain is achieved.

SYNTAX:

```
.MEASURE <DC|AC|TRAN> result TRIG … TARG …
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

Table 7-28 TRIG and TARG Measurement Syntax

Parameter	Description
MEASURE	Specifies measurements. You can abbreviate to MEAS.
result	Name associated with the measured value, in the HSPICE output. This example measures the independent variable, beginning at the trigger, and ending at the target: • Transient analysis measures time. • AC analysis measures frequency. • DC analysis measures the DC sweep variable. If simulation reaches the target before the trigger activates, the resulting value is negative. Do not use DC, TRAN, or AC as the *result* name.
TRIG…	Identifies the beginning of trigger specifications.
TARG …	Identifies the beginning of target specifications.
<DC|AC|TRAN>	Specifies the analysis type of the measurement. If you omit this parameter, HSPICE uses the last analysis mode that you requested.
GOAL	Specifies the desired measure value in ERR calculation for optimization. To calculate the error, the simulation uses the equation: $\text{ERRfun} = (\text{GOAL} - \text{result}) / \text{GOAL}$.
MINVAL	If the absolute value of GOAL is less than MINVAL, the MINVAL replaces the GOAL value, in the denominator of the ERRfun expression. Used only in ERR calculation for optimization. Default = 1.0e-12.
WEIGHT	Multiplies the calculated error by the weight value. Used only in ERR calculation for optimization. Default = 1.0.

You can use the LAST keyword in *TARG_SPEC* to indicate the last event. *TRIG_SPEC* and *TARG_SPEC* can also use the syntax:

```
TRIG AT = time
```

Trigger

```
TRIG trig_var VAL = trig_val <TD = time_delay>
+ <CROSS = c> <RISE = r> <FALL = f>
```

```
TRIG AT = val
```

Target

```
TARG targ_var VAL = targ_val <TD = time_delay>
+ <CROSS = c | LAST> <RISE = r | LAST> <FALL = f | LAST>
```

Table 7-29 TRIG and TARG Syntax

Parameter	Description
TRIG	Indicates the beginning of the trigger specification.
trig_val	Value of *trig_var*, which increments the counter for crossings, rises, or falls, by one.
trig_var	Specifies the name of the output variable, that determines the logical beginning of a measurement. If HSPICE reaches the target before the trigger activates, .MEASURE reports a negative value.
TARG	Indicates the beginning of the target signal specification.
targ_val	Specifies the value of the *targ_var*, which increments the counter for crossings, rises, or falls, by one.
targ_var	Name of the output variable, at which HSPICE determines the propagation delay with respect to the *trig_var*.
time_delay	Amount of simulation time that must elapse, before HSPICE enables the measurement. Simulation counts the number of crossings, rises, or falls, only after the *time_delay* value. Default trigger delay is zero.

CROSS = c
RISE = r
FALL = f
	Numbers indicate which CROSS, FALL, or RISE event to measure. For example:
	.meas tran tdlay trig v(1) val=1.5 td=10n
	+ rise=2 targ v(2) val=1.5 fall=2

	In the above example, rise=2 specifies to measure the v(1) voltage, only on the first two rising edges of the waveform. The value of these first two rising edges is 1. However, trig v(1) val=1.5 indicates to trigger when the voltage on the rising edge voltage is 1.5, which never occurs on these first two rising edges. So the v(1) voltage measurement never finds a trigger.
	• RISE = r, the WHEN condition is met, and measurement occurs after the designated signal has risen r rise times. • FALL = f, measurement occurs when the designated signal has fallen f fall times. A crossing is either a rise or a fall, so for CROSS = c, measurement occurs when the designated signal has achieved a total of c crossing times, as a result of either rising or falling. For TARG, the LAST keyword specifies the last event.

Table 7-29 TRIG and TARG Syntax (Continued)

Parameter	Description
LAST	HSPICE measures when the last CROSS, FALL, or RISE event occurs. • CROSS = LAST, measurement occurs the last time the WHEN condition is true, for a rising or falling signal. • FALL = LAST, measurement occurs the last time the WHEN condition is true, for a falling signal. • RISE = LAST, measurement occurs the last time the WHEN condition is true, for a rising signal. LAST is a reserved word; you cannot use it as a parameter name in the above .MEASURE statements.
AT = val	Special case for trigger specification. *val* is: • Time for TRAN analysis. • Frequency for AC analysis. • Parameter for DC analysis. The trigger determines where measurement starts.

HSPICE Example

```
.MEASURE TRAN tdlay TRIG V(1) VAL = 2.5 TD = 10n
+ RISE = 2 TARG V(2) VAL = 2.5 FALL = 2
```

This example measures propagation delay between nodes 1 and 2, for a transient analysis. HSPICE measures the delay from the second rising edge of the voltage at node 1, to the second falling edge of node 2. The measurement begins when the second rising voltage at node 1 is 2.5 V, and ends when the second falling voltage at node 2 is 2.5 V. The TD = 10n parameter counts the crossings, after 10 ns has elapsed. HSPICE prints results as tdlay = <*value*>.

```
.MEASURE TRAN riset TRIG I(Q1) VAL = 0.5m RISE = 3
+ TARG I(Q1) VAL = 4.5m RISE = 3
```

```
.MEASURE pwidth TRIG AT = 10n TARG V(IN) VAL = 2.5 CROSS = 3
```

In the last example, TRIG. AT = 10n starts measuring time at t = 10 ns, in the transient analysis. The TARG parameters end time measurement, when V(IN) = 2.5 V, on the third crossing. pwidth is the printed output variable.

Note: If you use the .TRAN statement with a .MEASURE statement, do not use a non-zero START time in .TRAN statement, or the .MEASURE results might be incorrect.

Average, RMS, and Peak Measurements

This .MEASURE statement reports the average, RMS, or peak value of the specified output variable.

SYNTAX:

```
.MEASURE < TRAN > out_var func var FROM = start TO = end
```

Table 7-30 Average, RMS, and Peak Syntax

Parameter	**Description**
varname	User-defined variable name for the measurement.
func	One of the following keywords: • AVG: Average area under *var*, divided by the period of interest. • MAX: Maximum value of *var* over the specified interval. • MIN: Minimum value of *var* over the specified interval. • PP: Peak-to-peak: reports the maximum value, minus the minimum of *var* over the specified interval.
	• RMS: Root mean squared: calculates the square root of the area under the *var*^{2} curve, divided by the period of interest. • INTEG: Integral of *var* over the specified period.
out_var var	Name of the output variable, which can be either the node voltage or the branch current of the circuit. You can also use an expression, consisting of the node voltages or the branch current.
start	Starting time of the measurement period.
end	Ending time of the measurement period.

EXAMPLE:

1. In the example below, the .MEASURE statement calculates the RMS voltage of the OUT node, from 0ns to 10ns. It then labels the result RMSVAL:

   ```
   .MEAS TRAN RMSVAL RMS V(OUT) FROM = 0NS TO = 10NS
   ```

2. In the example below, the .MEASURE statement finds the maximum current of the VDD voltage supply, between 10ns and 200ns in the simulation. The result is called MAXCUR.

   ```
   .MEAS MAXCUR MAX I(VDD) FROM = 10NS TO = 200NS
   ```

3. In the example below, the .MEASURE statement uses the ratio of V(OUT) and V(IN) to find the peak-to-peak value, in the interval of 0ns to 200ns.

   ```
   .MEAS P2P PP PAR(‘V(OUT)/V(IN)’) FROM = 0NS TO = 200NS
   ```

FIND and WHEN Functions

The FIND and WHEN functions 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 as 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 7-4.

SYNTAX:

```
.MEASURE <DC|TRAN| AC> result WHEN out_var = val <TD = val>
+ < RISE = r | LAST > < FALL = f | LAST > < CROSS = c | LAST >
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

```
.MEASURE <DC|TRAN|AC> result WHEN out_var1 = out_var2
+ < TD = val > < RISE = r | LAST > < FALL = f | LAST >
+ < CROSS = c| LAST > <GOAL = val> <MINVAL = val>
+ <WEIGHT = val>
```

```
.MEASURE <DC|TRAN|AC> result FIND out_var1
+ WHEN out_var2 = val < TD = val > < RISE = r | LAST >
+ < FALL = f | LAST > < CROSS = c| LAST > <GOAL = val>
+ <MINVAL = val> <WEIGHT = val>
```

```
.MEASURE <DC|TRAN|AC> result FIND out_var1
+ WHEN out_var2 = out_var3 <TD = val > < RISE = r | LAST >
+ < FALL = f | LAST > <CROSS = c | LAST> <GOAL = val>
+ <MINVAL = val> <WEIGHT = val>
```

```
.MEASURE <DC|TRAN|AC> result FIND out_var1 AT = val
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

Parameter Definitions

Table 7-31 FIND and WHEN Syntax

Parameter	**Description**

CROSS = c
RISE = r
FALL = f
	The numbers indicate which occurrence of a CROSS, FALL, or RISE event starts measuring.
	• For RISE = r, after the designated signal rises r rise times, the WHEN condition is met, and measurement begins.
	• For FALL = f, measurement starts when the designated signal has fallen f fall times.
	A crossing is a rise or a fall. For CROSS = c, measurement starts when the designated signal has achieved a total of c crossing times, as a result of either rising or falling.

<DC|AC|TRAN>	Analysis type for the measurement. If you omit this parameter, HSPICE assumes the last analysis type that you requested.

Table 7-31 FIND and WHEN Syntax (Continued)

Parameter	**Description**
FIND	Selects the FIND function.
GOAL	Desired .MEASURE value. Optimization uses this value in ERR calculation. The following equation calculates the error: $\mathrm{ERRfun} = (\mathrm{GOAL} - \mathrm{result}) / \mathrm{GOAL}$.
LAST	Starts measurement at the last CROSS, FALL, or RISE event. • For CROSS = LAST, measurement starts the last time the WHEN condition is true, for either a rising or falling signal. • For FALL = LAST, measurement starts the last time the WHEN condition is true, for a falling signal. • For RISE = LAST, measurement starts the last time the WHEN condition is true for a rising signal. LAST is a reserved word. Do not use it as a parameter name in these .MEASURE statements.
MINVAL	If the absolute value of GOAL is less than MINVAL, then MINVAL replaces the GOAL value in the denominator of the ERRfun expression. Used only in ERR calculation for optimization. Default = 1.0e-12.
out_var(1,2,3)	These variables establish conditions that start a measurement.
result	Name of a measured value, in the HSPICE output.
TD	Time at which measurement starts.
WEIGHT	Multiplies the calculated error by the weight value. Used only in ERR calculation for optimization. Default = 1.0.
WHEN	Selects the WHEN function.

EXAMPLE:

In the following example, the first measurement, TRT, calculates the difference between V(3) and V(4), when V(1) is half the voltage of V(2) at the last rise event.

The second measurement, STIME, finds the time when V(4) is 2.5V at the third rise-fall event. A CROSS event is a rising or falling edge.

```
.MEAS TRAN TRT FIND PAR(‘V(3)-V(4)’)
WHEN V(1)=PAR(‘V(2)/2’) + RISE = LAST
.MEAS STIME WHEN V(4) = 2.5 CROSS = 3
```

Equation Evaluation

Use this 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 syntax is:

```
.MEASURE <DC|TRAN|AC> result PARAM = 'equation'
+ <GOAL = val> <MINVAL = val>
```

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.

Note: AVG, RMS, and INTEG have no meaning in a DC data sweep, so if you use them, HSPICE issues a warning message.

SYNTAX:

```
.MEASURE <DC|AC|TRAN> result func out_var <FROM = val>
+ <TO = val> <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

Table 7-32 AVG, RMS, MIN, MAX, and PP Syntax

Parameter	Description
<DC|AC|TRAN>	Specifies the analysis type for the measurement. If you omit this parameter, HSPICE assumes the last analysis mode that you requested.
FROM	Specifies the initial value for the *func* calculation. For transient analysis, this value is in units of time.
TO	Specifies the end of the *func* calculation.
GOAL	Specifies the .MEASURE value. Optimization uses this value for ERR calculation. This equation calculates the error: $\mathrm{ERRfun} = (\mathrm{GOAL} - \mathrm{result}) / \mathrm{GOAL}$
MINVAL	If the absolute value of GOAL is less than MINVAL, MINVAL replaces the GOAL value in the denominator of the ERRfun expression. Used only in ERR calculation for optimization. Default = 1.0e-12.
func	Indicates one of the measure statement types: • AVG (average): Calculates the area under the out_var, divided by the periods of interest. • MAX (maximum): Reports the maximum value of the out_var, over the specified interval. • MIN (minimum): Reports the minimum value of the out_var, over the specified interval. • PP (peak-to-peak): Reports the maximum value, minus the minimum value, of the out_var, over the specified interval. • RMS (root mean squared): Calculates the square root of the area under the out_var2 curve, divided by the period of interest.
result	Name of the measured value, in the output. The value is a function of the variable (*out_var*) and *func*.
out_var	Name of any output variable whose function (*func*) the simulation measures.
WEIGHT	Multiplies the calculated error, by the weight value. Used only in ERR calculation for optimization. Default = 1.0.

EXAMPLE 1:

```
.MEAS TRAN avgval AVG V(10) FROM = 10ns TO = 55ns
```

The example above calculates the average nodal voltage value for node 10, during the transient sweep, from the time 10 ns to 55 ns. It prints out the result as *avgval*.

EXAMPLE 2:

```
.MEAS TRAN MAXVAL MAX V(1,2) FROM = 15ns  TO = 100ns
```

The preceding example finds the maximum voltage difference between nodes 1 and 2, for the time period from 15 ns to 100 ns.

EXAMPLE 3:

```
.MEAS TRAN MINVAL MIN V(1,2) FROM = 15ns TO = 100ns
.MEAS TRAN P2PVAL PP I(M1)  FROM = 10ns TO = 100ns
```

INTEGRAL Function

The INTEGRAL function reports the integral of an output variable, over a specified period.

SYNTAX:

```
.MEASURE <DC|AC|TRAN> result INTEGRAL out_var
+ <FROM = val> <TO = val> <GOAL = val> <MINVAL = val>
+ <WEIGHT = val>
```

The INTEGRAL function (with func), uses the same syntax as the average (AVG), RMS, MIN, MAX, and peak-to-peak (PP) measurement mode, to defined the INTEGRAL (INTEG).

EXAMPLE:

The following example calculates the integral of *I(cload)*, from 10 ns to 100 ns.

```
.MEAS TRAN charge INTEG I(cload) FROM = 10ns TO = 100ns
```

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.

SYNTAX:

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ AT = val <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ WHEN var2 = val <RISE = r | LAST> <FALL = f | LAST>
+ <CROSS = c | LAST> <TD = tdval> <GOAL = goalval>
+ <MINVAL = minval> <WEIGHT = weightval>
```

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ WHEN var2 = var3 <RISE = r | LAST> <FALL = f | LAST>
+ <CROSS = c | LAST> <TD = tdval> <GOAL = goalval>
+ <MINVAL = minval> <WEIGHT = weightval>
```

Table 7-33 Derivative Function Syntax

Parameter	**Description**
AT = val	Value of *out_var*, at which the derivative is found.

CROSS = c
RISE = r
FALL = f
	The numbers indicate which occurrence of a CROSS, FALL, or RISE event starts a measurement.
	For RISE = r, when the designated signal has risen r rise times, the WHEN condition is met, and measurement starts.
	For FALL = f, measurement starts when the designated signal has fallen f fall times.
	A crossing is either a rise or a fall, so for CROSS = c, measurement starts when the designated signal has achieved a total of c crossing times, as a result of either rising or falling.

<DC|AC|TRAN>	Specifies the analysis type to measure. If you omit this parameter, HSPICE assumes the last analysis mode that you requested.
DERIVATIVE	Selects the derivative function. You can abbreviate to DERIV.

Table 7-33 Derivative Function Syntax (Continued)

Parameter	Description
GOAL	Specifies the desired .MEASURE value. Optimization uses this value for ERR calculation. This equation calculates the error: $\mathrm{ERRfun} = (\mathrm{GOAL} - \mathrm{result}) / \mathrm{GOAL}$
LAST	Measures when the last CROSS, FALL, or RISE event occurs. CROSS = LAST, measures the last time the WHEN condition is true, for a rising or falling signal. FALL = LAST, measures the last time WHEN is true, for a falling signal. RISE = LAST, measures the last time WHEN is true, for a rising signal. LAST is a reserved word; do not use it as a parameter name in the above .MEASURE statements.
MINVAL	If the absolute value of GOAL is less than MINVAL, MINVAL replaces the GOAL value in the denominator of the ERRfun expression. Used only in ERR calculation for optimization. Default = 1.0e-12.
out_var	Variable for which HSPICE finds the derivative.
result	Name of the measured value, in the output.
TD	Identifies the time when measurement starts.
var(2,3)	These variables establish conditions that start a measurement.
WEIGHT	Multiplies the calculated error, between result and GOAL, by the weight value. Used only in ERR calculation for optimization. Default = 1.0.
WHEN	Selects the WHEN function.

EXAMPLE 1:

The following example calculates the derivative of V(out), at 25 ns:

```
.MEAS TRAN slew rate DERIV V(out) AT = 25ns
```

EXAMPLE 2:

The following example calculates the derivative of v(1), when v(1) is equal to 0.9*vdd:

```
.MEAS TRAN slew DERIV v(1) WHEN v(1) = ’0.90*vdd’
```

EXAMPLE 3:

The following example calculates the derivative of VP(output)/360.0, when the frequency is 10 kHz:.

```
.MEAS AC delay DERIV ’VP(output)/360.0’ AT = 10khz
```

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 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.

SYNTAX:

```
.MEASURE <DC|AC|TRAN> result ERRfun meas_var calc_var
+ <MINVAL = val> < IGNORE | YMIN = val> <YMAX = val>
+ <WEIGHT = val> <FROM = val> <TO = val>
```

Table 7-34   ERROR Function Syntax

Parameter	**Description**
<DC|AC|TRAN>	Specifies the analysis type, for the measurement. If you omit this parameter, HSPICE assumes the last analysis mode that you requested.
result	Name of the measured result, in the output.
ERRfun	ERRfun indicates which error function to use: ERR, ERR1, ERR2, or ERR3.
meas_var	Name of any output variable or parameter, in the data statement. *M* denotes the *meas_var*, in the error equation.
calc_var	Name of the simulated output variable or parameter, in the .MEASURE statement, to compare with *meas_var*. *C* is the *calc_var* in the error equation.
IGNOR|YMIN	If the absolute value of meas_var is less than the IGNOR value, then the ERRfun calculation does not consider this point. Default = 1.0e-15.

Table 7-34 ERROR Function Syntax (Continued)

Parameter	Description
FROM	Specifies the beginning of the ERRfun calculation. For transient analysis, the *from* value is in units of time. Defaults to the first value of the sweep variable.
WEIGHT	Multiplies the calculated error, by the weight value. Used only in ERR calculation for optimization. Default = 1.0.
YMAX	If the absolute value of meas_var is greater than the YMAX value, then the ERRfun calculation does not consider this point. Default = 1.0e+15.
TO	End of the ERRfun calculation. Default is last value of the sweep variable.
MINVAL	If the absolute value of meas_var is less than MINVAL, MINVAL replaces the meas_var value in the denominator of the ERRfun expression. Used only in ERR calculation for optimization. Default = 1.0e-12.

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 calculates NPTS ERR1 error functions. For device characterization, the ERR1 approach is more efficient than the other error functions (ERR, ERR2, ERR3).

$$\mathrm{ERR1}_i = \frac{M_i - C_i}{\max(\mathrm{MINVAL}, M_i)} \quad i = 1,\mathrm{NPTS}$$

HSPICE does not print out each calculated ERR1 value. When you set the ERR1 option, HSPICE calculates an ERR value, as follows:

$$\mathrm{ERR} = \left[\frac{1}{\mathrm{NPTS}} \cdot \sum_{i=1}^{\mathrm{NPTS}} \mathrm{ERR1}_i^2\right]^{1/2}$$

ERR2

This option computes the absolute relative error, at each point. For NPTS points, HSPICE calls NPTS error functions.

$$\mathrm{ERR2}_i = \left|\frac{M_i - C_i}{\max(\mathrm{MINVAL}, M_i)}\right| , = 1,\mathrm{NPTS}$$

The returned value printed for ERR2 is:

$$\mathrm{ERR} = \frac{1}{\mathrm{NPTS}} \cdot \sum_{i=1}^{\mathrm{NPTS}} \mathrm{ERR2}_i$$

ERR3

$$ERR3_i = \frac{\pm\log\left|\frac{M_i}{C_i}\right|}{\left|\log\left[\max(MINVAL, \left|M_i\right|)\right]\right|} = 1,NPTS$$

The + and - signs correspond to a positive and negative M/C ratio.

Note: If the M measured value is less than MINVAL, HSPICE uses MINVAL instead. If the absolute value of M is less than the IGNOR | YMIN value, or greater than the YMAX value, the error calculation does not consider this point.

Arithmetic Expression Measurements

The *expression* option is an arithmetic expression, that uses results from other prior .MEASURE statements.

SYNTAX:

```
.MEASURE < TRAN > varname PARAM = “expression”
```

EXAMPLE:

In the example below, the first two measurements, V3MAX and V2MIN, set up the variables for the third measurement statement:

```
.MEAS TRAN V3MAX MAX V(3) FROM 0NS TO 100NS
.MEAS TRAN V2MIN MIN V(2) FROM 0NS TO 100NS
.MEAS VARG PARAM = “(V2MIN + V3MAX)/2”
```

- V3MAX is the maximum voltage of V(3) between 0ns and 100ns of the simulation.
- V2MIN is the minimum voltage of V(2) during that same interval.
- VARG is the mathematical average of the V3MAX and V2MIN measurements.

Note: Expressions used in arithmetic expression must not be a function of node voltages or branch currents. Expressions used in all other .MEASURE statements can contain either node voltages or branch currents, but must not use results from other .MEASURE statements.

.DOUT Statement: Expected Digital Output Signal

The digital output (.DOUT) statement specifies the expected final state of an output signal, in HSPICE.

During simulation, HSPICE compares simulation results with the expected output. If the states are different, HSPICE reports an error.

SYNTAX:

The .DOUT statement can use either of two syntaxes. In both syntaxes, the time and state parameters describe the expected output of the nd node.

The first syntax specifies a single threshold voltage, VTH. A voltage level above VTH is high; any level below VTH is low.

```
.DOUT nd VTH ( time state < time state > )
```

Table 7-35 .DOUT Syntax

Parameter	**Description**
nd	is the node name.
time	is an absolute timepoint.
state	is the expected condition of the *nd* node at the specified *time:* • 0 expect ZERO,LOW. • 1 expect ONE,HIGH. • else Don’t care.
VTH	is the single voltage threshold.

The second syntax defines a threshold for both a logic high (*VHI*) and low (*VLO*).

```
.DOUT nd VLO VHI ( time state < time state > )
```

Table 7-36 .DOUT Syntax with Thresholds

Parameter	**Description**
VLO	is the voltage of the logic-low state
VHI	is the voltage of the logic-high state
nd	is the node name
time	is an absolute timepoint
state	is the expected condition of the *nd* node at the specified *time*: • 0 expect ZERO,LOW. • 1 expect ONE,HIGH. • else Don’t care.

Note: If you specify VTH, VLO, and VHI, then HSPICE processes only VTH, and ignores VLO and VHI.

For both cases, the *time*, *state* pair describes the expected output. During simulation, HSPICE compares the simulated results against the expected output vector. If the states are different, HSPICE reports an error message. The legal values for *state* are:

Table 7-37 State Values

Value	**Description**
0	expect ZERO
1	expect ONE
X, x	do not care
U, u	do not care
Z, z	expect HIGH IMPEDANCE (don’t care)

EXAMPLE:

The .PARAM statement in the example below sets the VTH variable value to 3. The .DOUT statement, operating on the node1 node, uses VTH as its threshold voltage.

```
.PARAM VTH = 3.0
.DOUT node1 VTH(0.0n 0 1.0n 1
+ 2.0n X 3.0n U 4.0n Z 5.0n 0)
```

When *node1* is above 3V, it is a logic 1; otherwise, it is a logic 0.

- At 0ns, the expected state of *node1* is logic-low.
- At 1ns, the expected state is logic-high.
- At 2ns, 3ns, and 4ns, the expected state is “do not care”.
- At 5ns, the expected state is again logic-low.

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.

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.

SYNTAX:

Brackets [ ] enclose comments, which are optional.

```
.stim <tran|ac|dc> PWL|DATA|VEC
+ <filename=output_filename> ...
```

PWL Source

You can use this syntax only in transient analysis.

```
.stim[tran] PWL [filename=output_filename]
+  [name1=] ovar1 [node1=n+] [node2=n-]
+  [[name2=]ovar2 [node1=n+] [node2=n-] ...]
+  [from=val] [to=val] [npoints=val]
```

```
.stim[tran] PWL [filename=output_filename]
+  [name1=] ovar1 [node1=n+] [node2=n-]
+  [[name2=]ovar2 [node1=n+] [node2=n-] ...]
+  indepvar=[(]t1 [t2 ...[)]]
```

Table 7-38 .STIM PWL Source Syntax

Parameter	**Description**
tran	Transient simulation.
filename	Output file name. If you do not specify a file, HSPICE uses the input filename.
namei	PWL Source Name that you specify. The name must start with V (for a voltage source) or `I` (for a current source).
ovari	Output variable that you specify. *ovar* can be: • Node voltage. • Element current. • Parameter string. If you use a parameter string, you must specify *name1*. For example: v(1), i(r1), v(2,1), par('v(1)+v(2)')
node1	Positive terminal node name.
node2	Negative terminal node name.
from	Keyword; specifies the time to start output of simulation results. For transient analysis, uses the time units that you specified.
npoints	Number of output time points.
to	Keyword; specifies the time to end output of simulation results. For transient analysis, uses the time units that you specified. The *from* value can be greater than the *to* value.
indepvar	Keyword; specifies dispersed (independent-variable) time points. You must specify dispersed time points in *increasing* order.

Data Card

```
.stim[tran |ac |dc] DATA [filename=output_filename]
+  dataname [name1=] ovar1
+  [[name2=]ovar2 ...] [from= val] [to=val]
+  [npoints=val] [indepout=val]
```

```
.stim[tran |ac |dc] DATA [filename=output_filename]
+  dataname [name1=] ovar1
+  [[name2=]ovar2 ...] indepvar=[(]t1 [t2 ...[)]]
+  [indepout=val]
```

Table 7-39 .STIM Data Card Syntax

Parameter	**Description**
tran | ac | dc	Selects the simulation type: transient, AC, or DC.
filename	Output file name. If you do not specify a file, HSPICE uses the input filename.
dataname	Name of the data card to generate.
from	Keyword; specifies the time to start output of simulation results. For transient analysis, uses the time units that you specified.
to	Keyword; specifies the time to end output of simulation results. For transient analysis, uses the time units that you specified.
namei	Name of a parameter of the data card to generate.
npoints	Number of output independent-variable points.
indepvar	Keyword; specifies dispersed independent-variable points.
indepout	Indicates whether to generate the independent variable column. • indepout, indepout = 1, or on, produces the independent variable column. You can specify the independent-variables in any order. • indepout= 0 or off (default) does not create an independent variable column. You can place the indepout field anywhere, after the ovari field.

Digital Vector File

You can use this syntax only in transient analysis.

```
.stim [tran] VEC [filename=output_filename]
+  vth=val vtl=val [voh=val] [vol=val] [name1=] ovar1
+  [[name2=]ovar2 ...] [from=val] [to=val]
+  [npoints=val]
```

```
.stim [tran] VEC [filename=output_filename]
+  vth=val vtl=val[voh=val] [vol=val] [name1=] ovar1
+  [[name2=]ovar2 ...] indepvar=[(]t1 [t2 ...[)]]
```

Table 7-40 .STIM Vector File Syntax

Parameter	Description
namei	Signal name that you specify.
filename	Output file name. If you do not specify a file, HSPICE uses the input filename.
ovari	Output variable that you specify. *ovar* must be a node voltage.
from	Keyword; specifies the time to start output of simulation results. For transient analysis, uses the time units that you specified.
to	Keyword; time to the end output of simulation results. For transient analysis, uses the specified time units.The *from* value can be greater than the *to* value.
npoints	Number of output time points.
indepvar	Keyword; specifies dispersed independent-variable points. You must specify dispersed time points in *increasing* order.
vth	High voltage threshold.
vtl	Low voltage threshold.
voh	Logic-high voltage for each output signal.
vol	Logic-low voltage for each output signal.
ovari	Output variable that you specify. *ovar* can be: • Node voltage. • Element current. • Element templates. • Parameter string. For example: v(1), i(r1), v(2,1), par('v(1)+v(2)'), LX1(m1), LX2(m1)

Output Files

The .STIM statement generates the following output files:

Table 7-41 .STIM 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.

Table 7-42 .STIM Symbols

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

Table 7-43 Resistor

Name	Alias	Description
G	LV1	Conductance, at analysis temperature.
R	LV2	Resistance, at reference temperature.
TC1	LV3	First temperature coefficient.
TC2	LV4	Second temperature coefficient.

Table 7-44 Capacitor

Name	Alias	Description
CEFF	LV1	Computed effective capacitance.
IC	LV2	Initial condition.
Q	LX0	Charge, stored in capacitor.
CURR	LX1	Current, flowing through capacitor.
VOLT	LX2	Voltage, across capacitor.
–	LX3	Capacitance (not used after HSPICE release 95.3).

Table 7-45 Inductor

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 release 95.3).

Table 7-46 Mutual Inductor

Name	**Alias**	**Description**
K	LV1	Mutual inductance.

Table 7-47 Voltage-Controlled Current Source

Name	**Alias**	**Description**
CURR	LX0	Current, through the source, if VCCS.
R	LX0	Resistance value, if VCR.
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 7-48 Voltage-Controlled Voltage Source

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 7-49 Current-Controlled Current Source

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 7-50 Current-Controlled Voltage Source

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 7-51 Independent Voltage Source

Name	**Alias**	**Description**
VOLT	LV1	DC/transient voltage.
VOLTM	LV2	AC voltage magnitude.
VOLTP	LV3	AC voltage phase.

Table 7-52 Independent Current Source

Name	**Alias**	**Description**
CURR	LV1	DC/transient current.
CURRM	LV2	AC current magnitude.
CURRP	LV3	AC current phase.

Table 7-53 Diode

Name	**Alias**	**Description**
AREA	LV1	Diode area factor.
AREAX	LV23	Area, after scaling.
IC	LV2	Initial voltage, across diode.
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).

Table 7-53 Diode (Continued)

Name	Alias	Description
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 7-54 BJT (Sheet 1 of 3)

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.
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.

Table 7-54 BJT (Sheet 2 of 3)

Name	**Alias**	**Description**
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_{π} = ^{1}ib /^{1}vbe, constant vbc.
GU	LX5	g_{μ} = ^{1}ib /^{1}vbc, constant vbe.
GM	LX6	g_m = ^{1}ic /^{1}vbe+ ^{1}ic /^{1}vbe, constant vce.
G0	LX7	g_0 = ^{1}ic /^{1}vce, 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).
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 ($C\Pi$).
CAP_IBC	LX20	cbc internal base-collector capacitance ($C\mu$).

Table 7-54 BJT (Sheet 3 of 3)

Name	Alias	Description
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 7-55 JFET

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.
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.

Table 7-55 JFET (Continued)

Name	**Alias**	**Description**
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 7-56 MOSFET (Sheet 1 of 3)

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 release 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).

Table 7-56 MOSFET (Sheet 2 of 3)

Name	Alias	Description
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.
BETAEFF	LV21	BETA, effective.
GAMMAEFF	LV22	GAMMA, effective.
DELTAL	LV23	∆L (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).
GMBSO	LX9	DC-substrate transconductance (GMBSO).
GBDO	LX10	Conductance of the drain diode (GBDO).
GBSO	LX11	Conductance of the source diode (GBSO).

Table 7-56 MOSFET (Sheet 3 of 3)

Name	Alias	Description
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	$CGGBO = \partial Qg / \partial Vgb$ = CGS + CGD + CGB
CGDBO	LX19	$CGDBO = \partial Qg / \partial Vdb$, (for Meyer CGD = -CGDBO)
CGSBO	LX20	$CGSBO = \partial Qg / \partial Vsb$, (for Meyer CGS = -CGSBO)
CBGBO	LX21	$CBGBO = \partial Qb / \partial Vgb$, (for Meyer CGB = -CBGBO)
CBDBO	LX22	$CBDBO = \partial Qb / \partial Vdb$
CBSBO	LX23	$CBSBO = \partial Qb / \partial Vsb$
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).
–	LX27	Source-bulk charge current (CQBS), (not used after HSPICE release 95.3).
CAP_BS	LX28	Bulk-source capacitance.
CAP_BD	LX29	Bulk-drain capacitance.
CQS	LX31	Channel-charge current (CQS).
CDGBO	LX32	$CDGBO = \partial Qd / \partial Vgb$
CDDBO	LX33	$CDDBO = \partial Qd / \partial Vdb$
CDSBO	LX34	$CDSBO = \partial Qd / \partial Vsb$

Table 7-57 Saturable Core 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 7-58 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

Simulation Options

This chapter describes the options that you can use to modify various aspects of a Synopsys HSPICE simulation, including:

- output types
- accuracy
- speed
- convergence

This chapter explains all options available in the .OPTION statement in HSPICE, including the following topics:

- Setting Control Options
- General Control Options
- Model Analysis Options\
- DC Operating Point, DC Sweep, and Pole/Zero Options
- Transient and AC Small Signal Analysis Options

Setting Control Options

This section describes how to set control options.

.OPTION Statement

To set control options, use .OPTION statements. You can set any number of options in one .OPTION statement, and you can include any number of .OPTION statements in an input netlist file. Table 8-2 on page 8-3 lists all control options. Descriptions of the options follow the table. For descriptions of options that are relevant to a specific simulation type, see the appropriate DC, transient, and AC analysis chapters.

Most options default to 0 (OFF) when you do not assign a value, using either .OPTION <*opt*> = <*val*> or the option with no assignment: .OPTION <*opt*>. Each option description in this section also shows the default value.

SYNTAX:

```
.OPTION opt1 <opt2 opt3 ...>
```

Table 8-1 .OPTION OPT Syntax

Parameter	Description
opt1 ...	Specifies any input control options. Many options are in the form <*opt*> = *x*, where <*opt*> is the option name and *x* is the value assigned to that option. This section describes all options.

EXAMPLE:

To reset options, set them to zero (.OPTION *<opt>* = 0). To redefine an option, enter a new .OPTION statement; HSPICE uses the last definition. For example, set the BRIEF option to 1, to suppress the printout. Then reset BRIEF to 0 later in the input file, to resume the printout.

```
.OPTION BRIEF $ Sets BRIEF to 1 (turns it on)
* Netlist, models,
...
.OPTION BRIEF = 0 $ Turns BRIEF off
Options Keyword Summary
```

Table 8-2 lists the keywords for the .OPTION statement, grouped by their typical application. The sections that follow the table, describe the options listed under each type of analysis.

Table 8-2 .OPTION Keyword Application Table (Sheet 1 of 3)

GENERAL CONTROL OPTIONS		**MODEL ANALYSIS**	**DC OPERATING POINT, DC SWEEP, and POLE/ZERO**		**TRANSIENT and AC SMALL SIGNAL ANALYSIS**	
Input, Output	Interfaces	General	Accuracy	Convergence	Accuracy	Timestep
ACCT	ARTIST	DCAP	ABSH	CONVERGE	ABSH	ABSVAR
ACOUT	CDS	SCALE	ABSI ABSTOL	CSHDC	ABSV, VNTOL	DELMAX
ALT999	CSDF	TNOM	ABSMOS	DCFOR	ACCURATE	DVDT
ALT9999	MEASOUT		ABSV VNTOL	DCHOLD	ACOUT	FS
ALTER	DLENCSDF					
BEEP						
BINPRNT	MENTOR	MOSFETs	ABSVDC	DCON	CHGTOL	FT

Table 8-2 .OPTION Keyword Application Table (Sheet 2 of 3)

GENERAL CONTROL OPTIONS		**MODEL ANALYSIS**	**DC OPERATING POINT, DC SWEEP, and POLE/ZERO**		**TRANSIENT and AC SMALL SIGNAL ANALYSIS**	
BRIEF	POST	CVTOL	DI	DCSTEP	CSHUNT	IMIN, ITL3
CO	PROBE	DEFAD	KCLTEST	DCTRAN	GSHUNT	IMAX, ITL4
INGOLD	PSF	DEFAS	MAXAMP	DV	DI	ITL5
LENNAM	SDA	DEFL	RELH	GMAX	GMIN	RELVAR
LIST	ZUKEN	DEFNRD	RELI	GMINDC	GSHUNT	RMAX
MEASDGT						
MEASFAIL						
MEASSORT		DEFNRS	RELMOS	GRAMP	CSHUNT	RMIN
NODE	Analysis	DEFPD	RELV RLTOL	GSHUNT	MAXAMP	SLOPETOL
NOELCK	ASPEC	DEFPS	RELVDC	ICSWEEP	RELH	TIMERES
NOMOD	LIMPTS	DEFW		ITLPTRAN	RELI	
NOPAGE	PARHIER	SCALM	Matrix	NEWTOL	RELQ	Algorithm
NOTOP	SPICE	WL	ITL1	OFF	RELTOL RELV	DVTR
NUMDGT	SEED		ITL2	RESMIN	RISETIME	IMAX
NXX		Inductors	NOPIV		TRTOL	IMIN
OPTLST	Error	GENK	PIVOT, SPARSE	Pole/Zero	VNTOL, ABSV	LVLTIM
OPTS	BADCHR	KLIM		CSCAL	Speed	MAXORD
PATHNUM	DIAGNOSTIC		PIVREF	FMAX	AUTOSTOP	METHOD PURETP
PLIM	NOWARN	BJTs	PIVREL	FSCAL	BKPSIZ	MU

Table 8-2 .OPTION Keyword Application Table (Sheet 3 of 3)

GENERAL CONTROL OPTIONS		**MODEL ANALYSIS**	**DC OPERATING POINT, DC SWEEP, and POLE/ZERO**		**TRANSIENT and AC SMALL SIGNAL ANALYSIS**	
				GSCAL		
POST_ VERSION	WARNLIMIT	EXPLI	PIVTOL	LSCAL	BYPASS	
PUTMEAS						
SEARCH			SPARSE, PIVOT	PZABS	BYTOL	Input, Output
STATFL	Version	Diodes		PZTOL	FAST	INTERP
VERIFY	H9007	EXPLI		RITOL	ITLPZ	ITRPRT
CPU			Input, Output	X*n*R, X*n*I	MBYPASS	UNWRAP
CPTIME			CAPTAB	NEWTOL	TRCON	
EPSMIN			DCCAP			
EXPMAX			VFLOOR			
LIMTIM						

General Control Options

Descriptions of the general control options follow. Descriptions are alphabetical by keyword, under the sections presented in the table.\

Table 8-3 Input and Output Options (Sheet 1 of 5)

Parameter	Description
ACCT	Reports job accounting and runtime statistics, at the end of the output listing. The ratio of output points to total iterations, determines simulation efficiency. Reporting is automatic, unless you disable it. Choices for ACCT are: • disables reporting • enables reporting • enables reporting of matrix statistics
ACOUT	AC output calculation method, for the difference in values of magnitude, phase, and decibels. Use these values for prints and plots. Default is 1. The default (ACOUT = 1) selects the HSPICE method, which calculates the difference of the magnitudes of the values. The SPICE method, ACOUT = 0, calculates the magnitude of the differences in HSPICE.
ALT999, ALT9999	This option generates up to 1000 (ALT999) or 10,000 (ALT9999) unique output files, from .ALTER simulation runs. HSPICE appends a number from 0-999 (ALT999) or 0-9999 (ALT9999) to the output file extension. For example, if a .TRAN analysis has 50 .ALTER statements, the filenames are filename.tr0, filename.tr1, ..., filename.tr50. Without this option, HSPICE overwrites files after the 36th .ALTER statement.
altchk	By default, HSPICE automatically reports topology errors in the latest elements, in your top-level netlist. It also reports errors in elements that you redefine, using the .ALTER statement (altered netlist). To disable topology checking in redefined elements (that is, to check topology only in the top-level netlist, but not in the altered netlist), set: .option altchk=0
	By default, .OPTION ALTCHK is set to 1: .option altchk=1 .option altchk This enables topology checking, in elements that you redefine using the .ALTER statement.
BEEP	BEEP=1 sounds an audible tone when simulation returns a message, such as *info: hspice job completed*. BEEP=0 turns off the audible tone.

Table 8-3 Input and Output Options (Sheet 2 of 5)

Parameter	Description
BINPRINT	Outputs the binning parameters of the CMI MOSFET model. Currently available only for Level 57.
BRIEF, NXX	Stops printback of the data file, until HSPICE finds an .OPTION BRIEF = 0, or the .END statement. It also resets the LIST, NODE, and OPTS options, and sets NOMOD. BRIEF = 0 enables printback. NXX is the same as BRIEF.
CO = x	Number of columns for printout: *x* can be either `80` (for narrow printout) or `132` (for wide carriage printouts). You also can use the `.WIDTH` statement to set the output width. Default=`80`.
INGOLD = x	Printout data format. Use INGOLD = 2 for SPICE compatibility in HSPICE. Default is 0. You can print numeric output from HSPICE, in one of three ways: INGOLD = 0 Specifies engineering format, which expresses exponents as one character: 1G = 1e9 1X = 1e6 1K = 1e3 1M = 1e-3 1U = 1e-6 1N = 1e-9 1P = 1e-12 1F = 1e-15
	INGOLD = 1 Combines fixed and exponential format (G Format). Uses fixed format for numbers 0.1 to 999. Uses exponential format for numbers greater than 999, or less than 0.1.
	INGOLD = 2 Uses exponential format exclusively (SPICE2G style). Exponential format generates constant number sizes, suitable for post-analysis tools.
	Use .OPTION MEASDGT, with INGOLD, to control the output data format for .MEASURE results.
LENNAM = x	Maximum length of names, in the printout of operating point analysis results. Default is 8. Maximum x value=16.
LIST, VERIFY	Produces an element summary of the input data to print. Calculates effective sizes of elements, and the key values. • BRIEF suppresses the LIST option. • VERIFY is an alias for LIST.
MEASDGT = x	Formats the .MEASURE statement output, in both the listing file and the .MEASURE output files (.ma0, .mt0, .ms0, and so on). The value of x is typically between 1 and 7, although you can set it as high as 10. Default is 4.0.

Table 8-3 Input and Output Options (Sheet 3 of 5)

Parameter	**Description**
	For example, if MEASDGT = 5, then .MEASURE displays numbers as: • Five decimal digits, for numbers in scientific notation. • Five digits to the right of the decimal, for numbers between 0.1 and 999. In the listing (.lis), file, all .MEASURE output values are in scientific notation, so .OPTION MEASDGT = 5 results in five decimal digits. Use MEASDGT with .OPTION INGOLD = x to control the output data format.
NODE	Prints a node cross reference table. BRIEF suppresses NODE. The table lists each node and all elements connected to it. A code indicates the terminal of each element. A colon (`:`) separates the code from the element name. The codes are: + Diode anode - Diode cathode B BJT base B MOSFET or JFET bulk C BJT collector D MOSFET or JFET drain E BJT emitter G MOSFET or JFET gate S BJT substrate S MOSFET or JFET source For example, part of a cross reference might look like: 1 M1:B D2:+ Q4:B This line indicates that the bulk of M1, the anode of D2, and the base of Q4, all connect to node 1.
NOELCK	No element check; bypasses element checking, to reduce to reduce pre-processing time for very large files.
NOMOD	Suppresses the printout of model parameters.
NOPAGE	Suppresses page ejects for title headings.
NOTOP	Suppresses topology checks, to increase speed for pre-processing very large files.
NUMDGT = x	Number of significant digits to print, for output variable values. The value of *x* is typically between 1 and 7, although you can set it as high as 10. Default is `4.0`. This option does not affect the accuracy of the simulation.
NXX	Stops printback of the data file, until HSPICE finds an .OPTION BRIEF = 0, or the .END statement. It also resets the LIST, NODE, and OPTS options, and sets NOMOD. BRIEF = 0 enables printback. NXX is the same as BRIEF.

Table 8-3 Input and Output Options (Sheet 4 of 5)

Parameter	**Description**
OPTLST = x	Outputs additional optimization information: • No information (default). • Prints parameter, Broyden update, and bisection results information. • Prints gradient, error, Hessian, and iteration information. • Prints all of the above, and Jacobian.
OPTS	Prints the current settings, for all control options. If you change any of the default values of the options, the OPTS option prints the values that the simulation actually uses. The BRIEF option suppresses OPTS.
PATHNUM	Prints subcircuit path numbers, instead of path names.
PLIM = x	Specifies plot size limits, for current and voltage plots: • Finds a common plot limit, and plots all variables on one graph, at the same scale • Enables SPICE-type plots in HSPICE, which create a separate scale and axis for each plot variable. This option does not affect post- processing of graph data.
POST_ VERSION = x	Sets the post-processing output version: • x = 9007 truncates the node name in the post-processor output file, to a maximum of 16 characters. • x = 9601 sets the node name length for the output file, consistent with the input restrictions (1024 characters).
POST_ VERSION= 2001	Sets the post-processing output version to 2001. This option shows you the new output file header, which includes the right number of output variables, rather than `****` when the number exceeds 9999. If you set `.OPTION post_version=2001 post=2` in the netlist, then HSPICE returns more-accurate ASCII results. .option post_version=2001 To use binary values (with double precision) in the output file, include the following in the input file: *** .option post (or post=1) post_version=2001 ** For more accurate simulation results, comment this format.
STATFL	Controls whether HSPICE creates a `.st0` file. • statfl=0 (default) outputs a .st0 file. • statfl=1 suppresses the .st0 file.

Table 8-3 Input and Output Options (Sheet 5 of 5)

Parameter	**Description**
SEARCH	Search path for libraries and included files. HSPICE searches the directory specified in .OPTION SEARCH, for libraries (file with a .INC suffix) that the simulation references. Includes these referenced files in the netlist.
VERIFY	VERIFY is an alias for LIST.

Table 8-4 CPU Options

Parameter	**Description**
CPTIME = x	Sets the maximum CPU time, in seconds, allotted for this simulation job. When the time allowed for the job exceeds CPTIME, HSPICE prints or plots the results up to that point, and concludes the job. Use this option if you are uncertain how long the simulation will take, especially when you debug new data files. Also see LIMTIM. Default is 1e7 (400 days).
EPSMIN = x	Specifies the smallest number that a computer can add or subtract, a constant value. Default is 1e-28.
EXPMAX = x	Specifies the largest exponent that you can use for an exponential, before overflow occurs. Typical value for an IBM platform is 350.
LIMTIM = x	Amount of CPU time reserved to generate prints and plots, if a CPU time limit (CPTIME = x) terminates simulation. Default=2 (seconds), normally sufficient for short printouts and plots.

Table 8-5 Interface Options (Sheet 1 of 3)

Parameter	**Description**
ARTIST = x	ARTIST = 2 enables the Cadence Analog Artist interface. This option requires a specific license. Supported on Sun Solaris 2.5/2.7/2.8, HPUX 10.20 and 11.20, and IBM AIX 4.3 platforms only. Not available on Linux platforms.
CDS, SDA	CDS = 2 produces a Cadence WSF (ASCII format) post-analysis file, for Opus™. This option requires a specific license. SDA is the same as CDS.
CSDF	Selects Common Simulation Data Format (Viewlogic-compatible graph data file format).

Table 8-5 Interface Options (Sheet 2 of 3)

Parameter	**Description**
DLENCSDF	If you use the Common Simulation Data Format (Viewlogic graph data file format) as the output format, this *digit length* option specifies how many digits to include, in scientific notation (exponents), or to the right of the decimal point. Valid values are any integer from 1 to 10. Default is 5. If you assign a *floating* decimal point, or if you specify less than 1 or more than 10 digits, HSPICE uses the default. For example, it places 5 digits to the right of a decimal point.
MEASOUT	Outputs .MEASURE statement values and sweep parameters into an ASCII file. Post-analysis processing (AvanWaves or other analysis tools) uses this <*design*>.mt# file, where # increments for each .TEMP or .ALTER block. For example, for a parameter sweep of an output load, which measures the delay, the .mt# file contains data for a delay-versus-fanout plot. Default is 1. You can set this option to 0 (off) in the hspice.ini file.
MENTOR = x	MENTOR = 2 enables the Mentor MSPICE-compatible (ASCII) interface. Requires a specific license.
MONTECON	Continues a Monte Carlo analysis in HSPICE. Retrieves the next random value, even if non-convergence occurs. A random value can be too large, or too small, to cause convergence to fail. Other types of analysis can use this Monte Carlo random value.
POST = x	Stores simulation results for analysis, using the AvanWaves graphical interface or other methods. • POST = 1 (default) saves the results in binary format. • POST = 2 saves the results in ASCII format. • POST = 3 saves the results in New Wave binary format.
	Set the POST option, and use the .PROBE statement to specify the data to save. To use binary values (with double precision) in the output file, include the following in the input file: ** .option post (or post=1) post_version=2001 ** For more accurate simulation results, comment this format.
post_version =2001	Sets the post-processing output version with a value of 2001. If you use this option, a new output file header includes the right number of output variables, rather than **** when the number exceeds 9999.

Table 8-5 Interface Options (Sheet 3 of 3)

Parameter	**Description**
PROBE	Limits post-analysis output to only variables specified in .PROBE, .PRINT, .PLOT, and .GRAPH statements. By default, HSPICE outputs all voltages and power supply currents, in addition to variables listed in .PROBE/.PRINT/.PLOT/.GRAPH statements. PROBE significantly decreases the size of simulation output files.
PSF = x	Specifies whether HSPICE outputs binary or ASCII data, when you run an HSPICE simulation from Cadence Analog Artist. Supported on Sun Solaris 2.5/2.7/2.8, HPUX 10.20 and 11.20, and IBM AIX 4.3 platforms only. Not available on Linux platforms. The value of *x* can be 1 or 2. • If x is 2, HSPICE produces ASCII output. • If .OPTION ARTIST PSF = 1, HSPICE produces binary output.
SDA	CDS = 2 produces a Cadence WSF (ASCII) format, post-analysis file, for Opus. This option requires a specific license. SDA is the same as CDS.
ZUKEN = x	• If x is 2, enables the Zuken interactive interface. • If x is 1 (default), disables this interface.

Table 8-6 Analysis Options

Parameter	**Description**
FFTOUT	Prints 30 harmonic fundamentals, sorted by size, THD, SNR, and SFDR, but only if you specify a .OPTION fftout statement and a .fft freq=xxx statement.
LIMPTS = x	Number of points to print or plot in AC analysis. You do not need to set LIMPTS for DC or transient analysis. HSPICE spools the output file to disk. Default=2001.
PARHIER	Selects parameter-passing rules that control the evaluation order of subcircuit parameters. Applies only to parameters with the same name, at different levels of subcircuit hierarchy. LOCAL A parameter name in a subcircuit, prevails over the same parameter name at a higher level of hierarchy. GLOBAL A parameter name at a higher level of hierarchy. Overrides the same parameter name at a lower level.
SEED	Starting seed for random-number generator in HSPICE Monte Carlo analysis. The minimum value is 1; the maximum value is 259200.

Table 8-6 Analysis Options (Continued)

Parameter	**Description**
ASPEC	Sets HSPICE to ASPEC-compatibility mode. When you set this option, the simulator reads ASPEC models and netlists, and the results are compatible. Default is 0 (HSPICE mode). If you set ASPEC, the following model parameters default to ASPEC values: *ACM = 1:* Changes the default values for CJ, IS, NSUB, TOX, U0, and UTRA. *Diode Model:* TLEV = 1 affects temperature compensation for PB. *MOSFET Model:* TLEV = 1 affects PB, PHB, VTO, and PHI. *SCALM, SCALE:* Sets the model scale factor to microns, for length dimensions. *WL:* Reverses implicit order for stating width and length in a MOSFET statement. Default (WL = 0) assigns the length first, then the width.

SPICE	Makes HSPICE compatible with Berkeley SPICE. If you set this option, HSPICE uses these options and model parameters:
	Example of general parameters, used with .OPTION SPICE:
	TNOM = 27 DEFNRD = 1 DEFNRS = 1 INGOLD = 2
	ACOUT = 0 DC
	PIVOT PIVTOL = IE-13 PIVREL = 1E-3 RELTOL = 1E-3
	ITL1 = 100
	ABSMOS = 1E-6 RELMOS = 1E-3 ABSTOL = 1E-12
	VNTOL = 1E-6
	ABSVDC = 1E-6 RELVDC = 1E-3 RELI = 1E-3
	Example of transient parameters, used with .OPTION SPICE:
	DCAP = 1 RELQ = 1E-3 CHGTOL-1E-14 ITL3 = 4 ITL4 = 10
	ITL5 = 5000 FS = 0.125 FT = 0.125
	Example of model parameters, used with .OPTION SPICE:
	For BJT: MJS = 0
	For MOSFET, CAPOP = 0
	LD = 0 if not user-specified
	UTRA = 0 not used by SPICE for LEVEL = 2
	NSUB must be specified
	NLEV = 0 for SPICE noise equation

Error Options

You can use the following error options in HSPICE:

Table 8-7 Error Options

Parameter	Description
BADCHR	Generates a warning, when it finds a non-printable character in an input file.
DIAGNOSTIC	Logs the occurrence of negative model conductances.
NOWARN	Suppresses all warning messages, except those generated from statements in .ALTER blocks.
WARNLIMIT = x	Limits how many times certain warnings appear in the output listing. This reduces the output listing file size. *x* is the maximum number of warnings for each warning type. This limit applies to these warning messages: • MOSFET has negative conductance. • Node conductance is zero. • Saturation current is too small. • Inductance or capacitance is too large. Default is 1.

Version Options

You can use version options in HSPICE:

Table 8-8 Version Options

Version	Description
H9007	Sets default values for general-control options, to correspond to values for HSPICE H9007D. If you set this option, HSPICE does not use the EXPLI model parameter.

Model Analysis Options\

Table 8-9 General Options

Parameter	Description
DCAP	Selects equations, which HSPICE uses to calculate depletion capacitance for Level 1 and 3 diodes, and BJTs. The *HSPICE Elements and Device Models Manual* describes these equations.
MODSRH	If MODSRH=1, HSPICE does not load or reference a model described in a .MODEL statement, if the netlist does not use that model. This option shortens simulation run time, when the netlist references many models, but no element in the netlist calls those models. Default is MODSRH=0. If MODSRH=1, then the read-in time increases slightly. example.sp: * modsrh used incorrectly .option post modsrh=1 xi1 net8 b c t6 xi0 a b net8 t6 v1 a 0 pulse 3.3 0.0 10E-6 1E-9 1E-9 + 25E-6 50E-6 v2 b 0 2 v3 c 0 3 .model nch nmos level=49 version=3.2 .end This input file automatically searches for t6.inc. If t6.inc includes the nch model, and you set MODSRH to 1, HSPICE does not load nch. Do not set MODSRH=1 in this type of file call. Use this option in front of the .MODEL card definition.
SCALE	Element scaling factor, in HSPICE. Scales parameters in element cards, by their value. Default=1.
HIER_SCALE	If you set the HIER_SCALE option, you can use the S parameter to scale sub-circuits. • 0 interprets S as a user-defined parameter. • 1 interprets S as a scale parameter. For more information about the *S* parameter, see S (Scale) Parameter on page 3-59.
TNOM	Reference temperature for HSPICE simulation. At this temperature, component derating is zero. Default is 25 degrees Celsius; if you enable .OPTION SPICE, Default is 27 degrees Celsius.

Table 8-9 General Options (Continued)

Parameter	Description
MODMONTE	If MODMONTE=1, then within a single simulation run, each device that shares the same model card and is in the same Monte Carlo index receives a different random value for parameters that have a Monte Carlo definition. If MODMONTE=0 (default), then within a single simulation run, each device that shares the same model card and is in the same Monte Carlo index, receives the same random value for its parameters that have a Monte Carlo definition.

Table 8-10 MOSFET Control Options

Parameter	Description
CVTOL	Changes the number of numerical integration steps, when calculating the gate capacitor charge for a MOSFET, using CAPOP = 3. See the discussion of CAPOP = 3 in the "Overview of MOSFETS" chapter of the *HSPICE Elements and Device Models Manual*, for explicit equations and discussion.
DEFAD	Default MOSFET drain diode area in HSPICE. Default=0.
DEFAS	Default MOSFET source diode area in HSPICE. Default=0.
DEFL	Default MOSFET channel length in HSPICE. Default=1e^{-4}m.
DEFNRD	Default number of squares for the drain resistor, on a MOSFET. Default is 0.
DEFNRS	Default number of squares for the source resistor, on a MOSFET. Fault is `0`.
DEFPD	Default MOSFET drain diode perimeter, in HSPICE. Default is 0.
DEFPS	Default MOSFET source diode perimeter, in HSPICE. Default is 0.
DEFW	Default MOSFET channel width, in HSPICE. Default=1e^{-4}m.
SCALM	Model scaling factor, in HSPICE. Scales model parameters by their value. Default is 1. See the *HSPICE Elements and Device Models Manual*, for parameters this option scales.
WL	Reverses the specified order, in the VSIZE MOS element. Default order is length-width; changes the order to width-length. Default is 0.

\

Table 8-11 Inductor Options

Parameter	Description
GENK	Automatically computes second-order mutual inductance, for several coupled inductors. 1 (default) enables the calculation.
KLIM	Minimum mutual inductance, below which automatic second-order mutual inductance calculation no longer proceeds. KLIM is unitless (analogous to coupling strength, specified in the K Element). Typical KLIM values are between .5 and 0.0. Default is 0.01.

Table 8-12 BJT and Diode Options

Parameter	Description
EXPLI	Current-explosion model parameter. PN junction characteristics, above the explosion current, are linear. HSPICE determines the slope at the explosion point. This improves simulation speed and convergence. Default is 0.0 amp/ AREAeff.

DC Operating Point, DC Sweep, and Pole/Zero Options

Table 8-13 Accuracy Options (Sheet 1 of 3)

Parameter	Description
ABSH = x	Sets the absolute current change, through voltage-defined branches (voltage sources and inductors). Use ABSH with DI and RELH, to check for current convergence. Default is 0.0.
ABSI = x	Sets the absolute error tolerance for branch currents, in diodes, BJTs, and JFETs, during DC and transient analysis. Decrease ABSI, if accuracy is more important than convergence time. To analyze currents less than 1 nanoamp, change ABSI to a value at least two orders of magnitude smaller than the minimum expected current. Default is 1e-9 for KCLTEST = 0, or 1e-6 for KCLTEST = 1.
ABSTOL = x	Sets the absolute error tolerance for branch currents, for DC and transient analysis. Decrease ABSTOL, if accuracy is more important than convergence time. ABSTOL is the same as ABSI.

Table 8-13 Accuracy Options (Sheet 2 of 3)

Parameter	**Description**
ABSMOS = x	Current error tolerance (for MOSFET devices), in DC or transient analysis. The ABSMOS setting determines whether the drain-to-source current solution has converged. The drain-to-source current converged if: • The difference between the drain-to-source current in the last iteration, versus the present iteration, is less than ABSMOS, or • This difference is greater than ABSMOS, but the percent change is less than RELMOS. If other accuracy tolerances also indicate convergence, HSPICE solves the circuit at that timepoint, and calculates the next timepoint solution. For low-power circuits, optimization, and single transistor simulations, set ABSMOS = 1e-12. Default is 1e-6 (amperes).
ABSVDC = x	Sets the minimum voltage, for DC and transient analysis. If accuracy is more critical than convergence, decrease ABSVDC. If you need voltages less than 50 micro-volts, reduce ABSVDC, to two orders of magnitude less than the smallest voltage. This ensures at least two digits of significance. Typically, you do not need to change ABSVDC, unless you simulate a high-voltage circuit. For 1000-volt circuits, a reasonable value is 5 to 50 millivolts. Default=VNTOL (VNTOL default = 50 mV).
DI = x	Sets the maximum iteration-to-iteration current change, through voltage-defined branches (voltage sources and inductors). Use this option only if the value of the ABSH control option is greater than 0. Default is 0.0.
KCLTEST	Activates KCL (Kirchhoff’s Current Law) test. increases simulation time, especially for large circuits, but very accurately checks the solution. Default=`0`.
	If you set this value to 1, HSPICE sets these options: • Sets RELMOS and ABSMOS options to 0 (off). • Sets ABSI to 1e-6 A. • Sets RELI to 1e-6. To satisfy the KCL test, each node must satisfy this condition: $\left\|\Sigma i_b\right\| < \mathrm{RELI} \cdot \Sigma\left\|i_b\right\| + \mathrm{ABSI}$ In this equation, the ibs are the node currents.
MAXAMP = x	Sets the maximum current, through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, HSPICE reports an error. Default is 0.0.
RELH = x	Relative current tolerance, through voltage-defined branches (voltage sources and inductors). Use RELH to check current convergence, but only if the value of the ABSH control option is greater than zero. Default is 0.05.

Table 8-13 Accuracy Options (Sheet 3 of 3)

Parameter	Description
RELI = x	Sets the relative error/tolerance change, from iteration to iteration. This parameter determines convergence for all currents, in diode, BJT, and JFET devices. (RELMOS sets tolerance for MOSFETs). This is the change in current, from the value calculated at the previous timepoint. • Default = 0.01 for KCLTEST = 0. • Default = 1e-6 for KCLTEST = 1.
RELMOS = x	Sets the relative error tolerance (percent) for drain-to-source current, from iteration-to-iteration. This parameter determines convergence for currents in MOSFET devices. (RELI sets the tolerance for other active devices.) Sets the change in current, from the value calculated at the previous timepoint. HSPICE uses the RELMOS value, only if the current is greater than the ABSMOS floor value. Default is 0.05.
RELV = x	Sets the relative error tolerance for voltages. If voltage or current exceeds the absolute tolerances, a RELV test determines convergence. Increasing RELV increases the relative error. You should generally maintain RELV at its default value. RELV conserves simulator charge. For voltages, RELV is the same as RELTOL. Default is 1e-3.
RELVDC = x	Sets the relative error tolerance for voltages. If voltages or currents exceed their absolute tolerances, the RELVDC test determines convergence. Increasing RELVDC increases the relative error. You should generally maintain RELVDC at its default value. RELVDC conserves simulator charge. Default is RELTOL (RELTOL default = 1e-3).

Table 8-14 Matrix Options

Parameter	Description
ITL1 = x	Maximum DC iteration limit. Increasing this value rarely improves convergence in small circuits. Values as high as 400 have resulted in convergence for some large circuits with feedback (such as operational amplifiers and sense amplifiers). However, to converge, most models do not require more than 100 iterations. Set .OPTION ACCT to list how many iterations an operating point requires. Default is 200.
ITL2 = x	Iteration limit for the DC transfer curve. Increasing this limit improves convergence, only for very large circuits. Default is 50.
NOPIV	Prevents HSPICE from automatically switching to pivoting matrix factors, if a nodal conductance is less than PIVTOL. NOPIV inhibits pivoting (see PIVOT).

Table 8-14 Matrix Options (Continued)

Parameter	Description
PIVOT = x	Selects a pivot algorithms. Use these algorithms to reduce simulation time, and to achieve convergence in circuits that produce hard-to-solve matrix equations. To select the pivot algorithm, set PIVOT to one of these values: 0: Original non-pivoting algorithm. 1: Original pivoting algorithm. 2: Picks the largest pivot in the row. 3: Picks the best pivot in a row. 10 (default): Fast, non-pivoting algorithm; requires more memory. 11: Fast, pivoting algorithm; requires more memory than PIVOT values less than 11. 12: Picks the largest pivot in the row; requires more memory than PIVOT values less than 12. 13: Fast, best pivot: faster; requires more memory than PIVOT values less than 13. The fastest algorithm is PIVOT = 13, which can improve simulation time up to ten times, on very large circuits. However, PIVOT = 13 requires substantially more memory for simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might require pivoting. If PIVTOL = 0, HSPICE automatically changes from non-pivoting, to a row-pivot strategy, if it detects any diagonal-matrix entry less than PIVTOL. This strategy provides the time and memory advantages of non-pivoting inversion, and avoids unstable simulations and incorrect results. Use .OPTION NOPIV, to prevent HSPICE from pivoting. For very large circuits, PIVOT = 10, 11, 12, or 13, can require excessive memory. If HSPICE switches to pivoting during a simulation, it prints the message: pivot change on the fly followed by the node numbers that cause the problem. Use .OPTION NODE to cross-reference a node to an element. SPARSE is the same as PIVOT.
PIVREF	Pivot reference. Use PIVREF in PIVOT = 11, 12, or 13, to limit the size of the matrix. Default is 1e+8.
PIVREL = x	Sets the maximum and minimum ratio of a row or matrix. Use only if PIVOT = 1. Large values for PIVREL can result in very long matrix pivot times. If the value is too small, however, no pivoting occurs. Start with small values of PIVREL, using an adequate (but not excessive) value, for convergence and accuracy. Default is 1E-20 (max = 1e-20, min = 1).

Table 8-14 Matrix Options (Continued)

Parameter	Description
PIVTOL = x	Absolute minimum value for which HSPICE accepts a matrix entry as a pivot. If PIVOT=0, PIVTOL is the minimum conductance in the matrix. Default=1.0e-15. PIVTOL must be less than GMIN or GMINDC. Values that approach 1 increase the pivot.
SPARSE = x	SPARSE is the same as PIVOT.

Table 8-15 Pole/Zero I/O Options

Parameter	Description
CAPTAB	Prints table of single-plate node capacitances, for diodes, BJTs, MOSFETs, JFETs, and passive capacitors, at each operating point.
DCCAP	Generates C-V plots. Prints capacitance values of a circuit (both model and element), during a DC analysis. You can use a DC sweep of the capacitor, to generate C-V plots. Default = 0 (off).
VFLOOR = x	Minimum voltage to print in output listing. All voltages lower than VFLOOR, print as 0. Affects only the output listing: VNTOL (ABSV) sets minimum voltage to use in a simulation.

Table 8-16 Convergence Options (Sheet 1 of 4)

Parameter	Description
CONVERGE or DCTRAN	Invokes different methods to solve non-convergence problems. CONVERGE = -1 Use with DCON = -1, to disable autoconvergence. CONVERGE = 0 Autoconvergence (default). CONVERGE = 1 Uses the Damped Pseudo Transient algorithm. If simulation does not converge within the set CPU time (in the CPTIME control option), then simulation halts. CONVERGE = 2 Uses a combination of DCSTEP and GMINDC ramping. Not used in the autoconvergence flow. CONVERGE = 3 Invokes the source-stepping method. Not used in the autoconvergence flow.

Table 8-16 Convergence Options (Sheet 2 of 4)

Parameter	Description
	CONVERGE = 4 Uses the *gmath* ramping method. Even you did not set it in an .OPTION statement, the CONVERGE option activates if a matrix floating-point overflows, or if HSPICE reports a *timestep too small* error. Default = `0`. If a matrix floating-point overflows, then CONVERGE = 1.
CSHDC	The same option as CSHUNT; use only with the CONVERGE option.
DCFOR = x	Use with DCHOLD and the .NODESET statement, to enhance DC convergence.
	DCFOR sets the number of iterations to calculate, after a circuit converges in the steady state. The number of iterations after convergence is usually zero, so DCFOR adds iterations (and computation time) to the DC circuit solution. DCFOR ensures that a circuit actually, not falsely, converges. Default is 0.
DCHOLD = x	Use DCFOR and DCHOLD together, to initialize DC analysis. DCFOR and DCHOLD enhance the convergence properties of a DC simulation. DCFOR and DCHOLD work with the .NODESET statement. Default is 1. DCHOLD specifies how many iterations to hold a node, at the .NODESET voltage values. The effects of DCHOLD on convergence differ, according to the DCHOLD value, and the number of iterations before DC convergence. If a circuit converges in the steady state, in fewer than DCHOLD iterations, the DC solution includes the values set in .NODESET. If a circuit requires more than DCHOLD iterations to converge, HSPICE ignores the values set in the .NODESET statement, and calculates the DC solution, using the .NODESET fixed-source voltages open circuited.
DCSTEP = x	Converts DC model and element capacitors to a conductance, to enhance DC convergence properties. HSPICE divides the value of the element capacitors by DCSTEP, to model DC conductance. Default is 0 (seconds).
DCON = X	If a circuit cannot converge, HSPICE automatically sets DCON = 1, and calculates the following: $DV = \max\left(0.1, \frac{V_{max}}{50}\right)$, if DV = 1000 $GRAMP = \max\left(6, \log_{10}\left(\frac{I_{max}}{GMINDC}\right)\right)$ $ITL1 = ITL1 + 20 \cdot GRAMP$ V_{max} is the maximum voltage, and I_{max} is the maximum current.

Table 8-16 Convergence Options (Sheet 3 of 4)

Parameter	Description
	• If the circuit still cannot converge, HSPICE sets DCON = 2, which sets DV = 1e6. • If the circuit uses discontinuous models or uninitialized flip-flops, simulation might not converge. Set DCON = -1 and CONVERGE = -1, to disable autoconvergence. HSPICE lists all non-convergent nodes and devices.
DCTRAN	Invokes different methods to solve non-convergence problems. DCTRAN is an alias for CONVERGE.
DV = x	Maximum iteration-to-iteration voltage change, for all circuit nodes, in both DC and transient analysis. High-gain bipolar amplifiers can require values of 0.5 to 5.0, to achieve a stable DC operating point. Large CMOS digital circuits frequently require about 1 volt. Default is 1000 (or 1e6 if DCON = 2).
GMAX = x	Conductance, in parallel with a current source, for .IC and .NODESET initialization circuitry. Some large bipolar circuits require you to set GMAX=1, for convergence. Default=100 (mho).
GMINDC = x	Conductance in parallel to all pn junctions and MOSFET nodes except *gate* (see Figure 4-2 on page 4-42), for DC analysis. GMINDC helps overcome DC convergence problems, caused by low values of off-conductance, for pn junctions and MOSFETs. You can use GRAMP to reduce GMINDC, by one order of magnitude, for each step. Set GMINDC between 1e-4 and the PIVTOL value. Default is 1e-12. Large values of GMINDC can cause unreasonable circuit response. If your circuit requires large values to converge, suspect a bad model or circuit. If a matrix floating-point overflows, and if GMINDC is 1.0e-12 or less, HSPICE sets it to 1.0e-11. HSPICE manipulates GMINDC in auto-converge mode (see Autoconverge Process on page 9-31).
GSHUNT	Conductance, added from each node to ground. Default is zero. Add a small GSHUNT to each node, to help solve *Timestep too small* problems, caused by either high-frequency oscillations or numerical noise.
GRAMP = x	HSPICE sets this value during auto-convergence (default=0). Use GRAMP, with the GMINDC option, to find the smallest GMINDC value that results in DC convergence. For a description of GMINDC, see on page 9-28. GRAMP specifies a conductance range, over which DC operating point analysis sweeps GMINDC. HSPICE replaces GMINDC values over this range, simulates each value, and uses the lowest GMINDC value where the circuit converges in a steady state.

Table 8-16   Convergence Options (Sheet 4 of 4)

Parameter	Description
	If you sweep GMINDC between 1e-12 mhos (default) and 1e-6 mhos, GRAMP is 6 (value of the exponent difference, between the default and the maximum conductance limit). In this example: • HSPICE first sets GMINDC to 1e-6 mhos, and simulates the circuit. • If circuit simulation converges, HSPICE sets GMINDC to 1e-7 mhos, and simulates the circuit. • The sweep continues until HSPICE simulates all values of the GRAMP ramp. • If the combined GMINDC and GRAMP conductance is greater than 1e-3 mho, false convergence can occur.
ICSWEEP	Saves the current analysis result of a parameter or temperature sweep, as the starting point in the next analysis in the sweep. • If ICSWEEP = 1 (default), the next analysis uses the current results. • If ICSWEEP = 0, the next analysis does not use the results of the current analysis.
ITLPTRAN	Controls the iteration limit used in the final try of the pseudo-transient method, in OP or DC analysis. If simulation fails in the final try of the pseudo-transient method, enlarge this option. Default is 30.
NEWTOL	Calculates one or more iterations past convergence, for every calculated DC solution and timepoint circuit solution. If you do not set NEWTOL, after HSPICE determines convergence, the convergence routine ends, and the next program step begins. Default is 0.
OFF	For all active devices, initializes terminal voltages to zero, if you did not initialize them to other values. For example, if you did not initialize both drain and source nodes of a transistor (using .NODESET or .IC statements, or connecting them to sources), then OFF initializes all nodes of the transistor to zero. HSPICE checks the OFF option, before element IC parameters. If you assigned an element IC parameter to a node, simulation initializes the node to the element IC parameter value, even if the OFF option previously set it o zero.
	You can use the OFF element parameter to initialize terminal voltages to zero, for specific active devices. Use the OFF option to help find exact DC operating-point solutions, for large circuits.
RESMIN = x	Minimum resistance for all resistors, including parasitic and inductive resistances. Default is 1e-5 (ohm). Range: 1e-15 to 10 ohm.

Table 8-17 Pole/Zero Control Options

Parameter	**Description**
CSCAL	Sets the capacitance scale. HSPICE multiplies capacitances by CSCAL. Default is 1e+12 (capacitances in pF).
FMAX	Maximum frequency of angular velocity for poles/zeros. Default=1.0e+12 rad/sec.
FSCAL	Sets the frequency scale. HSPICE multiplies the frequency by FSCAL. Default is 1.0e-9 (that is, all frequencies are in units of GHz).
GSCAL	Sets the conductance scale. HSPICE multiplies conductances, and divides resistances, by GSCAL. Default is 1e+3 (that is, by default, you enter all resistances in units of $k\Omega$).
ITLPZ	Sets the iteration limit for pole/zero analysis. Default = 100.
LSCAL	Sets the inductance scale. HSPICE multiplies inductances by LSCAL. Default is 1e+6 (that is, all inductances are in units of mH).
	Scale factors must satisfy the following relations: $GSCAL = CSCAL \cdot FSCAL$ $\quad GSCAL = \frac{1}{LSCAL} \cdot FSCAL$ If you change scale factors, modify initial Muller points (X0R, X0I), (X1R, X1I), and (X2R, X2I). HSPICE multiplies initial values by (1.0e-9/GSCAL).
PZABS	Absolute tolerances, for poles and zeros. Affects only low-frequency poles or zeros. Use it as follows: If $(\lvert X_{real}\rvert + \lvert X_{imag}\rvert < PZABS)$, then $X_{real} = 0$ and $X_{imag} = 0$. You can also use this option for convergence tests. Default is 1.0e-2.
PZTOL	Relative error tolerance, for poles or zeros. Default=1.0e-6.
RITOL	Minimum ratio for (real/imaginary), or (imaginary/real) parts of poles or zeros.Default is 1.0e-6. If $\lvert X_{imag}\rvert \le RITOL \cdot \lvert X_{real}\rvert$, then $X_{imag} = 0$. If $\lvert X_{real}\rvert \le RITOL \cdot \lvert X_{imag}\rvert$, then $X_{real} = 0$.

(X0R,X0I),
(X1R,X1I),
(X2R,X2I)
	Three complex starting points, in the Muller pole/zero analysis algorithm, are:
	X0R = -1.23456e6 X0I = 0.0
	X1R = -1.23456e5 X1I = 0.0
	X2R = +.23456e6 X2I = 0.0
	HSPICE multiplies these initial points, and FMAX, by FSCAL.

Transient and AC Small Signal Analysis Options

Table 8-18 Accuracy Options (Sheet 1 of 3)

Parameter	Description
ABSH = x	Absolute current change, through voltage-defined branches (voltage sources and inductors). Use ABSH with DI and RELH to check for current convergence. Default is 0.0.
ABSV = x	Sets absolute minimum voltage for DC and transient analysis. ABSV is the same as VNTOL. If accuracy is more critical than convergence, decrease VNTOL. If you need voltages less than 50 microvolts, reduce VNTOL to two orders of magnitude less than the smallest desired voltage. This ensures at least two significant digits. Typically, you do not need to change VNTOL, except to simulate a high-voltage circuit. A reasonable value for 1000-volt circuits is 5 to 50 millivolts. Default is 50 (microvolts).
ACCURATE	Selects a time algorithm that uses LVLTIM = 3 and DVDT = 2, for circuits such as high-gain comparators. Use this option with circuits that combine high gain and large dynamic range, to guarantee accurate solutions in HSPICE. When set to 1, ACCURATE sets these control options: • LVLTIM = 3 • DVDT = 2 • RELVAR = 0.2 • ABSVAR = 0.2 • FT = 0.2 • RELMOS = 0.01 Default is 0.
ACOUT	AC output calculation method, for the difference in values of magnitude, phase, and decibels. Use this option for prints and plots. Default is 1. Default value, ACOUT = 1, selects HSPICE method, which calculates the difference of the magnitudes of the values. SPICE method, ACOUT = 0, calculates the magnitude of the differences.
CHGTOL = x	Sets a charge error tolerance, if you set LVLTIM = 2. Use CHGTOL with RELQ to set the absolute and relative charge tolerance for all HSPICE capacitances. Default=1e-15 (coulomb).
CSHUNT	Capacitance added from each node to ground, in HSPICE. Add a small CSHUNT to each node, to solve internal timestep too small problems, caused by high-frequency oscillations or numerical noise. Default=0.
GMIN = x	Minimum conductance added to all PN junctions, for a time sweep in transient analysis. Default is 1e-12.

Table 8-18 Accuracy Options (Sheet 2 of 3)

Parameter	Description
DI = x	Maximum iteration-to-iteration current change, through voltage-defined branches (voltage sources and inductors). Use this option only if the value of the ABSH control option is greater than 0. Default is 0.0.
GSHUNT	Conductance, added from each node to ground. Default is zero. Add a small GSHUNT to each node, to help solve some *internal timestep too small* problems, caused by high-frequency oscillations or numerical noise.
MAXAMP = x	Maximum current, through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, HSPICE issues an error. Default is 0.0.
RELH = x	Relative current tolerance, through voltage-defined branches (voltage sources and inductors). Use RELH to check current convergence, but only if the value of the ABSH control option is greater than zero. Default is 0.05.
RELI = x	Relative error/tolerance change, from iteration to iteration. This parameter determines convergence for all currents, in diode, BJT, and JFET devices. (RELMOS sets tolerance for MOSFETs). This is the change in current, from the value calculated at the previous timepoint. • Default = 0.01 for KCLTEST = 0. • Default = 1e-6 for KCLTEST = 1.
RELQ = x	Used in the timestep algorithm for local truncation error (LVLTIM = 2). RELQ changes the timestep size. If the capacitor charge calculation (in the present iteration) exceeds that of the past iteration by a percentage greater than the RELQ value, then HSPICE reduces the internal timestep (Delta). Default=0.01.
RELTOL, RELV	Relative error tolerance for voltages. Use RELV, with the ABSV control option, to determine voltage convergence. Increasing RELV increases the relative error. RELV is the same as RELTOL. RELI and RELVDC options default to the RELTOL value. Default is 1e-3.
RISETIME	Smallest risetime of a signal, .OPTION RISETIME=x. Use it only in transmission line models, in HSPICE. In the U Element, this equation determines the number of lumps: $\mathrm{MIN}\left[20, 1+\left(\frac{\mathrm{TDeff}}{\mathrm{RISETIME}}\right)\cdot 20\right]$ TDeff is the end-to-end delay in a transmission line. The W Element uses RISETIME, only if Rs or Gd is non-zero. In such cases, RISETIME determines the maximum signal frequency.

Table 8-18 Accuracy Options (Sheet 3 of 3)

Parameter	**Description**
TRTOL = x	Used in the timestep algorithm for local truncation error (LVLTIM = 2). HSPICE multiplies TRTOL by the internal timestep, which the timestep algorithm for the local truncation error generates. TRTOL reduces simulation time, and maintains accuracy. It estimates the amount of error introduced, when the algorithm truncates the Taylor series expansion. This error reflects the minimum time-step, to reduce simulation time and maintain accuracy. The range of TRTOL is 0.01 to 100; typical values are 1 to 10. If you set TRTOL to 1 (the minimum value), HSPICE uses a very small timestep. As you increase the TRTOL setting, the timestep size increases. Default is 7.0.
VNTOL = x, ABSV	Absolute minimum voltage, for DC and transient analysis. ABSV is the same as VNTOL. Decrease VNTOL, if accuracy is more critical than convergence. If you need voltages less than 50 microvolts, reduce VNTOL to two orders of magnitude less than the smallest desired voltage. This ensures at least two significant digits. Typically, you change VNTOL only if you simulate a high-voltage circuit. For 1000-volt circuits, a reasonable value is 5 to 50 millivolts. Default is 50 (microvolts).

Table 8-19 Speed Options (Sheet 1 of 3)

Parameter	**Description**
AUTOSTOP	Stops a transient analysis in HSPICE, after calculating all TRIG-TARG and FIND-WHEN measure functions. This option can substantially reduce CPU time. By default, if the data file contains measure functions (such as AVG, RMS, MIN, MAX, PP, ERR, ERR1,2,3, or PARAM), then AUTOSTOP is disabled (that is, .OPTION autostop or .OPTION autostop from_to=0 is set). To use AUTOSTOP with these measure functions, set .OPTION autostop from_to or .OPTION autostsop from_to=1. For trig-targ and find-when measure functions, if you set autostop, do not use the preceding measure result as the measured parameter. Otherwise, the measured result is probably inaccurate.
BKPSIZ = x	Sets the size of the breakpoint table. Default is 5000. This is an old option, provided only for backward-compatibility.
BYPASS	Bypasses model evaluations, if the terminal voltages do not change. Can be 0 (off) or 1 (on). To speed-up simulation, this option does not update the status of latent devices. To enable bypassing, set .OPTION BYPASS = 1, for MOSFETs, MESFETs, JFETs, BJTs, or diodes. Default = 1. Use the BYPASS algorithm cautiously. Some circuit types might not converge, and might lose accuracy in transient analysis and operating-point calculations.

Table 8-19 Speed Options (Sheet 2 of 3)

Parameter	Description
BYTOL = x	Specifies a voltage tolerance, at which a MOSFET, MESFET, JFET, BJT, or diode becomes latent. HSPICE does not update status of latent devices. Default = MBYPASS x VNTOL.
FAST	To speed-up simulation, this option does not update the status of latent devices. Use this option for MOSFETs, MESFETs, JFETs, BJTs, and diodes. Default is 0. A device is latent, if its node voltage variation (from one iteration to the next) is less than the value of either the BYTOL control option, or the BYPASSTOL element parameter. (If FAST is on, HSPICE sets BYTOL to different values, for different types of device models.) Besides the FAST option, you can also use the NOTOP and NOELCK options, to reduce input pre-processing time. Increasing the value of the MBYPASS or BYTOL option, also helps simulations to run faster, but can reduce accuracy.
ITLPZ	Sets the iteration limit for pole/zero analysis. Default is 100.
MBYPASS = x	Computes the default value of the BYTOL control option: BYTOL = MBYPASSxVNTOL Also multiplies the RELV voltage tolerance. Set MBYPASS to about 0.1, for precision analog circuits. • Default is 1, for DVDT = 0, 1, 2, or 3. • Default is 2, for DVDT = 4.
TRCON	Controls the speed of some special circuits. For some large non-linear circuits with large TSTOP/TSTEP values, analysis might run for an excessively long time. In this case, HSPICE might automatically set a new and bigger RMAX value, to speed up the analysis for primary reference. In most cases, however, HSPICE does not activate this type of autospeedup process. For autospeedup to occur, all three of the following conditions must occur: • N1 (Number of Nodes) > 1,000 • N2 (TSTOP/TSTEP) >= 10,000 • N3 (Total Number of Diode, BJTs, JFETs and MOSFETs) > 300 Autospeedup is most likely to occur if the circuit also meets either of the following conditions: • N2 >= 1e+8, and N3 > 500, or • N2 >= 2e+5, and N3 > 1e+4

Table 8-19 Speed Options (Sheet 3 of 3)

Parameter	Description
	If HSPICE does activate autospeedup, you might need to disable it. To do this, set TRCON=-1, and increase TSTEP or RMAX (or both), to balance accuracy and speed. • TRCON = 0 or TRCON=1 enables autospeedup for circuits that meet necessary conditions. • TRCON = -1 disables autospeedup. The default value of TRCON is 1. TRCON also controls the automatic convergence process. See on page 8-32.

Table 8-20 Timestep Options (Sheet 1 of 3)

Parameter	Description
ABSVAR = x	Sets the absolute limit for the maximum voltage change, from one time point to the next. Use this option with the DVDT algorithm. If the simulator produces a convergent solution that is greater than ABSVAR, then HSPICE discards the solution, sets the timestep to a smaller value, and recalculates the solution. This is called a timestep reversal. Default=0.5 (volts).
DELMAX = x	Sets the maximum Delta of the internal timestep. HSPICE automatically sets the DELMAX value, based on the factors listed in Timestep Control for Accuracy on page 10-25. The initial DELMAX value, shown in the HSPICE output listing, is generally not the value used for simulation.
DVDT	Adjusts the timestep, based on rates of change for node voltage. Default is 4. • 0 - original algorithm • 1 - fast • 2 - accurate 3,4 - balance speed and accuracy
FS = x	Decreases Delta (internal timestep) by the specified fraction of a timestep (TSTEP), for the first time point of a transient. Decrease the FS value to help circuits that have timestep convergence difficulties. DVDT = 3 uses FS to control the timestep. $\mathrm{Delta} = \mathrm{FS} \times [\mathrm{MIN}(\mathrm{TSTEP}, \mathrm{DELMAX}, \mathrm{BKPT})]$ • You specify DELMAX. • BKPT is related to the breakpoint of the source. • The .TRAN statement sets TSTEP. Default = 0.25.

Table 8-20 Timestep Options (Sheet 2 of 3)

Parameter	**Description**
FT = x	Decreases Delta (the internal timestep), by a specified fraction of a timestep (TSTEP), for an iteration set that does not converge. If DVDT = 2 or DVDT = 4, FT controls the timestep. Default = 0.25.

IMIN = x,
ITL3 = x
	Minimum timestep, in timestep algorithms for transient analysis. IMIN is the minimum number of iterations required, to obtain convergence. If the number of iterations is less than IMIN, the internal timestep (Delta) doubles.

	Use this option to decrease simulation times, in circuits where the nodes are stable most of the time (such as digital circuits). If the number of iterations is greater than IMIN, the timestep stays the same, unless the timestep exceeds the IMAX option. ITL3 is the same as IMIN. Default is 3.0.

IMAX = x,
ITL4 = x
	Maximum timestep, in timestep algorithms for transient analysis. IMAX sets the maximum iterations, to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep (Delta) decreases, by a factor equal to the FT transient control option. HSPICE uses the new timestep to calculate a new solution. IMAX also works with the IMIN transient control option. ITL4 is the same as IMAX. Default is 8.0.

ITL3 = x	ITL3 is the same as IMIN. Default is 3.0.
ITL4 = x	ITL4 is the same as IMAX. Default is 8.0.
ITL5 = x	Sets an iteration limit for transient analysis. If a circuit uses more than ITL5 iterations, the program prints all results, up to that point. The default (0.0) allows an infinite number of iterations.
RELVAR = x	Use this option with ABSVAR, and the DVDT timestep algorithm. RELVAR sets the relative voltage change, for LVLTIM = 1 or 3. If the node voltage at the current time point exceeds the node voltage at the previous time point by RELVAR, then HSPICE reduces the timestep, and calculates a new solution at a new time point. Default is 0.30 (30%).
RMAX = x	Sets the TSTEP multiplier, which controls the maximum value (DELMAX) for the Delta of the internal timestep: DELMAX = TSTEP x RMAX • Default = 5, if dvdt = 4 and lvltim = 1. • Otherwise, the default = 2. The maximum value is 1e+9, the minimum value is 1e-9. The recommended maximum value is 1e+5.

Table 8-20 Timestep Options (Sheet 3 of 3)

Parameter	Description
RMIN = x	Sets the minimum value of Delta (internal timestep). An internal timestep smaller than RMINxTSTEP, terminates the transient analysis, and reports an internal timestep too small error. If the circuit does not converge in IMAX iterations, Delta decreases by the amount you set in the FT option. Default = 1.0e-9.
SLOPETOL = x	Minimum value, for breakpoint table entries in a piecewise linear (PWL) analysis. If the difference in the slopes of two consecutive PWL segments is less than the SLOPETOL value, HSPICE ignores the breakpoint, for the point between the segments. Default is 0.5.
TIMERES = x	Minimum separation between breakpoint values, for the breakpoint table. If two breakpoints are closer together (in time) than the TIMERES value, HSPICE enters only one of them in the breakpoint table. Default is 1 ps.

Table 8-21 Algorithm Options (Sheet 1 of 3)

Option	Description
DVTR	Limits voltage in transient analysis. Default is 1000.

IMAX = x,
ITL4 = x
	Maximum timestep, in timestep algorithms for transient analysis. IMAX sets the maximum iterations, to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep (Delta) decreases, by a factor equal to the FT transient control option. HSPICE uses the new timestep to calculate a new solution. IMAX also works with the IMIN transient control option. ITL4 is the same as IMAX. Default is 8.0.

IMIN = x,
ITL3 = x
	Minimum timestep, in timestep algorithms for transient analysis. IMIN is the minimum number of iterations required, to obtain convergence. If the number of iterations is less than IMIN, the internal timestep (Delta) doubles. Use this option to decrease simulation times, in circuits where the nodes are stable most of the time (such as digital circuits). If the number of iterations is greater than IMIN, the timestep stays the same, unless the timestep exceeds the IMAX option. ITL3 is the same as IMIN. Default is 3.0.

LVLTIM = x	Selects the timestep algorithm, for transient analysis. • LVLTIM = 1 (default) uses the DVDT timestep algorithm. • LVLTIM = 2 uses the timestep algorithm for local truncation error. • LVLTIM = 3 uses the DVDT timestep algorithm with timestep reversal.

Table 8-21 Algorithm Options (Sheet 2 of 3)

Option	Description
	• To use the GEAR method of numerical integration and linearization, select LVLTIM = 2. • To use the TRAP linearization algorithm, select LVLTIM = 1 or 3. Using LVLTIM = 1 (DVDT option) is the default, and helps avoid internal timestep too small non-convergence. The local truncation algorithm (LVLTIM = 2) provides a higher degree of accuracy than the TRAP method. If you use this option, errors do not propagate from time point to time point, which can result in an unstable solution.
MAXORD = x	Maximum order of integration, for the GEAR method in HSPICE (see METHOD). The x value can be either 1 or 2. • MAXORD = 1 uses the backward Euler integration method. • MAXORD = 2 (default) is more stable, accurate, and practical.
METHOD = *name*	Sets the numerical integration method, for a transient analysis, to either GEAR or TRAP. • To use GEAR, set METHOD = GEAR, which sets LVLTIM = 2. • To change LVLTIM from 2 to 1 or 3, set LVLTIM = 1 or 3, after the METHOD = GEAR option. This overrides METHOD=GEAR, which sets LVLTIM = 2. TRAP (trapezoidal) integration usually reduces program execution time, with more accurate results. However, this method can introduce an apparent oscillation on printed or plotted nodes, which might not result from circuit behavior. To test this, run a transient analysis, using a small timestep. If oscillation disappears, the cause was the trapezoidal method. The GEAR method is a filter, removing oscillations that occur in the trapezoidal method. Highly non-linear circuits (such as operational amplifiers) can require very long execution times, when you use the GEAR method.
	Circuits that do not converge in trapezoidal integration, often converge if you use GEAR. Default is TRAP (trapezoidal).
PURETP	Integration method to use, for reversal time point. Default is 0. If you set puretp=1, then if HSPICE finds non-convergence, it uses TRAP (instead of B.E) for the reversed time point. Use this option, with the method=TRAP statement, to help some oscillating circuits to oscillate, if the default simulation process cannot satisfy the result.
MU = x	Coefficient for trapezoidal integration. Range is 0.0 to 0.5. Default is 0.5.

Table 8-21 Algorithm Options (Sheet 3 of 3)

Option	Description
TRCON	Controls the automatic convergence (*autoconvergence*) and automatic speedup (*autospeedup*) processes in HSPICE. HSPICE also uses autoconvergence in DC analysis, if the Newton-Raphson (N-R) method fails to converge. • TRCON=1 (the default) enables both autoconvergence and autospeedup. • TRCON= 0 enables *autospeedup* only. • TRCON =-1 disables both *autoconvergence* and *autospeedup*. If the circuit fails to converge using the trapezoidal (TRAP) numerical integration method (for example, because of trapezoidal oscillation), HSPICE uses the GEAR method and LTE timestep algorithm, to run the transient analysis again from time=0. This process is called autoconvergence. Autoconvergence sets options to their default values before the second try: • METHOD=GEAR, LVLTIM=2, MBYPASS=1.0, BYPASS=0.0, SLOPETOL=0.5, BYTOL= min{mbypas*vntol and reltol} • RMAX=2.0 if it was 5.0 in the first run. Otherwise RMAX does not change.

Input and Output Options

You can use the following input and output options in HSPICE:

Table 8-22 Input/Output Options

Parameter	Description
INTERP	Limits output for post-analysis tools, such as Cadence or Zuken, to only the .TRAN timestep intervals. By default, HSPICE outputs all convergent iterations. INTERP typically produces a much smaller design.tr# file. If the netlist includes .MEASURE statements, use INTERP = 1 cautiously. To compute measure statements, HSPICE uses the postprocessing output. Reduced postprocessing output can incorrectly interpolate measure results. If you run data-driven transient analysis (.TRAN DATA statement) within optimization, HSPICE forces INTERP to 1. All measurement results are at time points set in the data-driven sweep. To measure only at converged internal timesteps (such as to calculate AVG or RMS), set ITRPRT = 1.
ITRPRT	Prints output variables, at their internal time points. This option might generate a long output list. Use the ITRPRT option if you use .PRINT statements that include functions such as ABS, AVG, RMS, INT, NINT, and so on, to avoid interpolation errors.

Table 8-22 Input/Output Options (Continued)

Parameter	Description
MEASFAIL	• If measfail=0, outputs 0 into the .mt#, .ms#, or .ma# file, and prints failed to the listing file. • If measfail=1 (default), prints failed into the .mt#, .ms#, or .ma# file, and into the listing file: .option measfail=1 | 0
MEASSORT	To automatically sort large numbers of .MEASURE statements, use the .OPTION MEASSORT statement. • .OPTION MEASSORT=0 (default; do not sort .MEASURE statements). • .OPTION MEASSORT=1 (internally sort .MEASURE statements). Set this option to 1 only if you use a large number of .MEASURE statements, where you need to list similar variables together (to reduce simulation time). For a small number of .MEASURE statements, turning on internal sorting can slow-down simulation while sorting, compared to not sorting first.
PUTMEAS	Controls the output variables, listed in the .MEASURE statement. .option putmeas=0 or (1) Does not save variable values, which are listed in the .MEASURE statement, into the corresponding output file (such as .tr#, .ac# or .sw#). This option decreases the size of the output file. Default. Saves variable values, which are listed in the .MEASURE statement, into the corresponding output file (such as .tr#, .ac# or .sw#). This option is similar to the output of Hspice 2000.4.
UNWRAP	Displays phase results from AC analysis, in unwrapped form (with a continuous phase plot). HSPICE uses these results to accurately calculate group delay. It also uses unwrapped phase results to compute group delay, even if you do not set the UNWRAP option.

9

Initializing DC/Operating Point Analysis

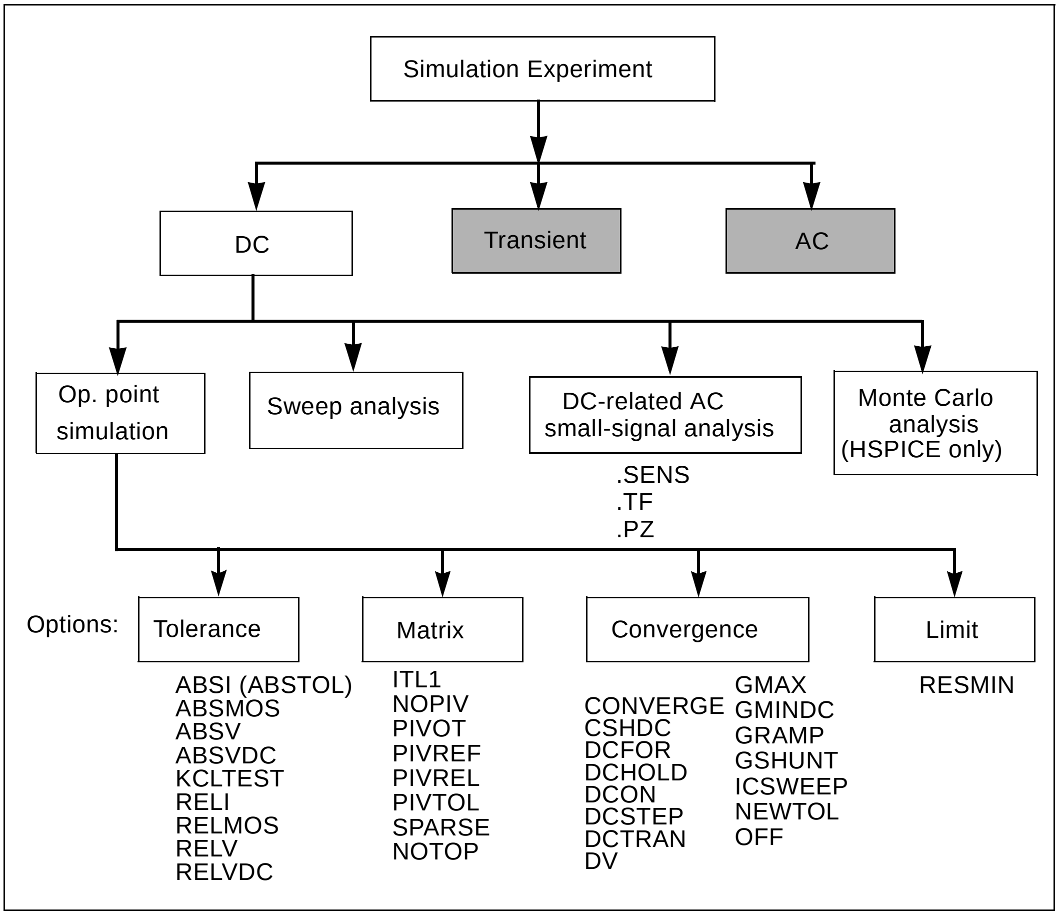
This chapter describes DC initialization and operating point analysis. It explains the following topics:

- Simulation Flow
- Initialization and Analysis
- DC Initialization and Operating Point Statements
- .DC Statement—DC Sweeps
- Other DC Analysis Statements
- DC Initialization Control Options
- Accuracy and Convergence
- Reducing DC Errors
- Diagnosing Convergence Problems

Simulation Flow

Figure 9-1 shows the simulation flow, for DC analysis in Synopsys HSPICE.

Figure 9-1 DC Initialization and Operating Point Analysis Simulation Flow



Initialization and Analysis

Before it performs .OP, .DC sweep, .AC, or .TRAN analyses, HSPICE first sets the DC operating point values, for all nodes and sources. To do this, HSPICE 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 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 then uses the resulting DC operating point as an initial estimate, to solve the next timepoint in the transient analysis.

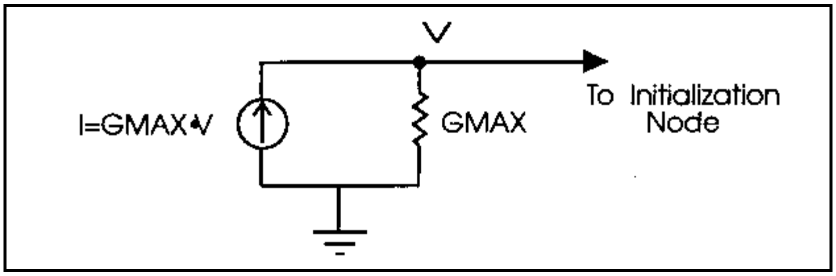
1. If you do not provide an initial guess, or if you provide only partial information, HSPICE provides a default estimate, for each node in the circuit.
2. HSPICE then uses this estimate to iteratively find the exact solution.

   The .NODESET and .IC 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 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 next calculates a DC operating point, with the .NODESET voltage source equivalent connected.
6. HSPICE 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 9-2 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 sets node voltages, according to those statements.
2. If you set the OFF option, then HSPICE 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 sets those initial conditions.
4. HSPICE 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.

When you set initial conditions for Transient Analysis:

- If you include UIC in a .TRAN statement, HSPICE 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.
- An internal timestep too small error message indicates that the circuit failed to converge. The cause of the failure can be that HSPICE cannot use stated initial conditions to calculate the actual DC operating point.

DC Initialization and Operating Point Statements

.OP Statement — Operating Point

When you include an .OP statement in an input file, HSPICE 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 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.

SYNTAX:

```
.OP <format> <time> <format> <time>
```

Table 9-1 .OP Syntax

Parameter	Description
format	Any of the following keywords. Only the first letter is required. Default = ALL • ALL: Full operating point, including voltage, currents, conductances, and capacitances. This parameter outputs voltage/current for the specified time. • BRIEF: Produces a one-line summary of each element’s voltage, current, and power. Current is stated in milliamperes, and power is in milliwatts. • CURRENT: Voltage table, with a brief summary of element currents and power. • DEBUG: Usually invoked only if a simulation does not converge. Debug prints back the non-convergent nodes, with the new voltage, old voltage, and the tolerance (degree of non-convergence). It also prints back the non-convergent elements, with their tolerance values. • NONE: Inhibits node and element print-outs, but performs additional analysis that you specify. • VOLTAGE: Voltage table only. The preceding keywords are mutually- exclusive; use only one at a time.
time	Place this parameter directly after ALL, VOLTAGE, CURRENT, or DEBUG. It specifies the time at which HSPICE prints the report.

EXAMPLE 1:

The following example calculates:

- Operating point voltages and currents, for the DC solution.
- Currents at 10 ns, for the transient analysis.
- Voltages at 17.5 ns, 20 ns and 25 ns, for the transient analysis.

```
.OP .5NS CUR 10NS VOL 17.5NS 20NS 25NS
```

EXAMPLE 2:

The following example calculates the complete DC operating point solution. The next section shows a printout of the solution.

```
.OP
```

Output

```
***** OPERATING POINT INFORMATIONTNOM = 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
   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.

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.

.IC and .DCVOLT Initial Condition Statements

Use the .IC statement, or the .DCVOLT statement, to set transient initial conditions in HSPICE How it initializes depends on whether the .TRAN analysis statement includes the UIC parameter.

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. Transient analysis releases the initialized nodes, to calculate the second and later time points.

SYNTAX:

```
.IC V(node1) = val1 V(node2) = val2 ...
```

```
.DCVOLT V(node1) = val1 V(node2) = val2 ...
```

```
.DCVOLT V node1 val1 <node2 val2 ...>
```

Table 9-2 .IC Syntax

Parameter	**Description**
val1 ...	Specifies voltages. The significance of these voltages depends on whether you specify the UIC parameter in the .TRAN statement.
node1 ...	Node numbers or names can include full paths, or circuit numbers.

EXAMPLE:

```
.IC V(11) = 5 V(4) = -5 V(2) = 2.2
.DCVOLT 11 5 4 -5 2 2.2
```

.NODESET Statement

.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 simulator uses the NODESET voltages, only in the first iteration.

SYNTAX:

```
.NODESET V(node1) = val1 <V(node2) = val2 ...>
```

or

```
.NODESET node1 val1 <node2 val2>
```

Table 9-3 .NODESET Syntax

Parameter	**Description**
node1 ...	Node numbers or names can include full paths or circuit numbers.
val1	Specifies voltages.

EXAMPLE:

```
.NODESET V(5:SETX) = 3.5V V(X1.X2.VINT) = 1V
.NODESET V(12) = 4.5 V(4) = 2.23
.NODESET 12 4.5 4 2.23 1 1
```

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.

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.

.SAVE Statement

The .SAVE statement in HSPICE stores the operating point of a circuit, in a file that you specify. 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.

SYNTAX:

```
.SAVE <TYPE = type_keyword> <FILE = save_file>
+ <LEVEL = level_keyword> <TIME = save_time>
```

Table 9-4 .SAVE Syntax

Parameter	Description
type_keyword	Storage method, for saving the operating point. The type can be one of the following. Default is NODESET. • .NODESET: Stores the operating point as a .NODESET statement. Later simulations initialize all node voltages to these values, if you use the .LOAD statement. If circuit conditions change incrementally, DC converges within a few iterations. • .IC: Stores the operating point as a .IC statement. Later simulations initialize node voltages to these values if the netlist includes the .LOAD statements.
save_file	Name of the file that stores `DC` operating point data. The file name format is <*design*>.ic#. Default is <*design*>.ic0.
level_keyword	Circuit level, at which you save the operating point. The level can be one of the following. • ALL (default): Saves all nodes, from the top to the lowest circuit level. This option offers the greatest improvement in simulation time. • TOP: Saves only nodes in the top-level design. Does not save subcircuit nodes. • NONE: Does not save the operating point.
save_time	Time during transient analysis, when HSPICE saves the operating point. HSPICE requires a valid transient analysis statement, to save a DC operating point. Default = 0.

A parameter or temperature sweep saves only the first operating point.

EXAMPLE:

If the input netlist file contains the statement:

```
.TEMP -25 0 25
```

then HSPICE saves the operating point corresponding to .TEMP -25.

.LOAD Statement

Use the .LOAD statement to input the contents of a file, that you stored using the .SAVE statement in HSPICE.

Files stored with the .SAVE statement contain operating point information, for the point in the analysis at which you executed .SAVE.

Do not use the .LOAD command for concatenated netlist files.

SYNTAX:

```
.LOAD <FILE = load_file>
```

load_file is the name of the file, in which .SAVE saved an operating point, for the circuit under simulation.The format of the file name is <*design*>.ic#. Default is <*design*>.ic0, where design is the root name of the design.

.DC Statement—DC Sweeps

You can use the .DC statement in DC analysis, to:

- Sweep any parameter value.
- Sweep any source value.
- Sweep temperature range.
- Perform a DC Monte Carlo (random sweep) analysis.
- Perform a data-driven sweep.
- Perform a DC circuit optimization, for a data-driven sweep.
- Perform a DC circuit optimization, using start and stop.
- Perform a DC model characterization.

The format for the .DC statement depends on the application that uses it, as shown in the examples that follow.

SYNTAX:

Sweep or Parameterized Sweep:

```
.DC var1 START = start1 STOP = stop1 STEP = incr1
```

```
.DC var1 START = <param_expr1> STOP = <param_expr2>
+   STEP = <param_expr3>
```

```
.DC var1 start1 stop1 incr1
+ <SWEEP var2 type np start2 stop2>
```

```
.DC var1 start1 stop1 incr1 <var2 start2 stop2 incr2>
```

Data-Driven Sweep:

```
.DC var1 type np start1 stop1 <SWEEP DATA = datanm>
```

```
.DC DATA = datanm<SWEEP var2 start2 stop2 incr2>
```

```
.DC DATA = datanm
```

Monte Carlo:

```
.DC var1 type np start1 stop1 <SWEEP MONTE = val>
```

```
.DC MONTE = val
```

Optimization:

```
.DC DATA = datanm OPTIMIZE = opt_par_fun
+   RESULTS = measnames MODEL = optmod
```

```
.DC var1 start1 stop1 SWEEP OPTIMIZE = OPTxxx
+   RESULTS = measname MODEL = optmod
```

Keywords and Parameters

Table 9-5 .DC Syntax

Parameter	**Description**
DATA = *datanm*	*Datanm* is the reference name of a .DATA statement.
incr1 …	Voltage, current, element, or model parameters; or temperature increments.
MODEL	Specifies the optimization reference name. The .MODEL OPT statement uses this name in an optimization analysis
MONTE = *val*	*val* is the number of randomly-generated values, which you can use to select parameters from a distribution. The distribution can be *Gaussian*, *Uniform*, or *Random Limit.*
np	Number of points per decade or per octave, or just number of points, based on which keyword precedes it.
OPTIMIZE	Specifies the parameter reference name, used for optimization in the .PARAM statement
RESULTS	Measure name used for optimization in the .MEASURE statement
start1 …	Starting voltage, current, element, or model parameters; or temperature values. If you use the POI (list of points) variation type, specify a list of parameter values, instead of *start stop*.
stop1 …	Final voltage, current, any element, model parameter, or temperature values.
SWEEP	Keyword, to indicate that a second sweep has a different type of variation (DEC, OCT, LIN, POI, or DATA statement; or MONTE = *val*)
TEMP	Keyword, to indicate a temperature sweep.
type	Can be any of the following keywords: • DEC — decade variation • OCT — octave variation • LIN — linear variation • POI — list of points
var1 …	• Name of an independent voltage or current source, or • Name of any element or model parameter, or • TEMP keyword (indicating a temperature sweep). HSPICE supports a source value sweep, which refers to the source name (SPICE style). However, if you select a parameter sweep, a .DATA statement, and a temperature sweep, then you must select a parameter name for the source value. A later .DC statement must refer to this name. The parameter name must not start with V, I, or TEMP.

EXAMPLE 1:

The following example sweeps the value of the VIN voltage source, from 0.25 volts to 5.0 volts, in increments of 0.25 volts.

```
.DC VIN 0.25 5.0 0.25
```

EXAMPLE 2:

The following example sweeps the drain-to-source voltage, from 0 to 10 V, in 0.5 V increments, at VGS values of 0, 1, 2, 3, 4, and 5 V.

```
.DC VDS 0 10 0.5 VGS 0 5 1
```

EXAMPLE 3:

The following example starts a DC analysis of the circuit, from -55°C to 125°C, in 10°C increments.

```
.DC TEMP -55 125 10
```

EXAMPLE 4:

The following script runs a DC analysis, at five temperatures: 0, 30, 50, 100, and 125°C.

```
.DC TEMP POI 5 0 30 50 100 125
```

EXAMPLE 5:

The following example runs a DC analysis on the circuit, at each temperature value. The temperatures result from a linear temperature sweep, from 25°C to 125°C (five points), which sweeps a resistor value named *xval*, from 1 k to 10 k, in 0.5 k increments.

```
.DC xval 1k 10k .5k SWEEP TEMP LIN 5 25 125
```

EXAMPLE 6:

The example below specifies a sweep of the *par1* value, from 1 k to 100 k, in increments of 10 points per decade.

```
.DC DATA = datanm SWEEP par1 DEC 10 1k 100k
```

EXAMPLE 7:

The next example also requests a DC analysis, at specified parameters in the .DATA *datanm* statement. It also sweeps the par1 parameter, from 1k to 100k, in increments of 10 points per decade.

```
.DC par1 DEC 10 1k 100k SWEEP DATA = datanm
```

EXAMPLE 8:

The final example invokes a DC sweep of the par1 parameter from 1k to 100k by 10 points per decade, using 30 randomly generated (Monte Carlo) values.

```
   .DC par1 DEC 10 1k 100k SWEEP MONTE = 30
```

Schmitt Trigger Example

```
*file: bjtschmt.spbipolar schmitt trigger
.OPTION post = 2
vcc 6 0 dc 12
vin 1 0 dc 0 pwl(0,0 2.5u,12 5u,0)
cb1 2 4 .1pf
rc1 6 2 1k
rc2 6 5 1k
rb1 2 4 5.6k
rb2 4 0 4.7k
re 3 0 .47k
*
diode 0 1 dmod
q1 2 1 3 bmod 1 ic = 0,8
q2 5 4 3 bmod 1 ic = .5,0.2
*
```

```
.dc vin 0,12,.1
.model dmod d is = 1e-15 rs = 10
.model bmod npn is = 1e-15 bf = 80 tf = 1n
+ cjc = 2pf cje = 1pf rc = 50 rb = 100 vaf = 200
.plot v(1) v(5)
.graph dc model = schmittplot input = v(1)
+ output = v(5) 4.0 5.0
.model schmittplot plot xscal = 1 yscal = 1 xmin = .5u
+ xmax = 1.2u
.end
```

Other DC Analysis Statements

HSPICE 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.

Table 9-6 DC Analysis Statements

Statement	Description
.PZ	Performs pole/zero analysis (you do not need to specify .OP)
.SENS	Obtains DC small-signal sensitivities of output variables, for circuit parameters (you do not need to specify `.OP`)
.TF	Calculates DC small-signal values for transfer functions (ratio of output variable, to input source). You do not need to specify .OP.

HSPICE provides DC control options, and DC initialization statements, which 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.

.SENS Statement — DC Sensitivity Analysis

If the input file includes a .SENS statement, HSPICE determines DC small-signal sensitivities for each specified output variable, relative to every circuit parameter. The sensitivity measurement is the partial derivative of each output variable, for a specified circuit element, measured at the operating point, and normalized to the total change in output magnitude. Therefore, the sum of the sensitivities of all elements is 100%. HSPICE calculates sensitivities for:

- resistors
- voltage sources
- current sources
- diodes
- BJTs (including Level 4, the VBIC95 model)
- MOSFETs (Level49 and Level53, Version=3.22).

You can perform only one .SENS analysis per simulation. If you specify more than one .SENS statement, HSPICE runs only the last .SENS statement.

SYNTAX:

```
.SENS ov1 <ov2 ...>
```

Table 9-7 .SENS Syntax

Parameter	Description
ov1 ov2 …	Branch currents, or nodal voltage, for DC component-sensitivity analysis.

EXAMPLE:

```
.SENS V(9) V(4,3) V(17) I(VCC)
```

Note: The .SENS statement can generate very large amounts of output for large circuits.

.TF Statement — DC Small-Signal Transfer Function Analysis

The transfer function statement (.TF) defines small-signal output and input, for DC small-signal analysis. When you use the .TF statement, HSPICE computes:

- DC small-signal value of the transfer function (output/input),.
- Input resistance.
- Output resistance.

SYNTAX:

```
.TF ov srcnam
```

Table 9-8 .TF Syntax

Parameter	**Description**
ov	Small-signal output variable.
srcnam	Small-signal input source.

EXAMPLE:

```
.TF V(5,3) VIN
.TF I(VLOAD) VIN
```

For the first example, HSPICE computes the ratio of V(5,3) to VIN. This is the ratio of small-signal input resistance at VIN, to the small-signal output resistance (measured across nodes 5 and 3). If you specify more than one .TF statement in a single simulation, HSPICE runs only the last .TF statement.

.PZ Statement— Pole/Zero Analysis

SYNTAX:

```
.PZ output input
```

```
.PZ ov srcname
```

Table 9-9 .PZ Syntax

Parameter	Description
PZ	Invokes the pole/zero analysis.
input	Input source. Can be the name of any independent voltage or current source.
output	Output variables, which can be: • Any node voltage, V(*n*). • Any branch current, I(*branch_name*).
ov	Output variable: a node voltage V(n), or branch current I(element)
srcnam	Input source: an independent voltage or current source name

EXAMPLE:

```
.PZ   V(10)   VIN
.PZ   I(RL)   ISORC
```

See “Pole/Zero Analysis” in the *HSPICE Applications Manual*, for complete information about pole/zero 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 (see Chapter 8, “Simulation Options”):

- ABSTOL
- CAPTAB
- CSHDC
- DCCAP
- DCFOR
- DCHOLD
- DCSTEP
- DV
- GRAMP
- GSHUNT
- ICSWEEP
- ITLPTRAN
- ITL1
- ITL2
- KCLTEST
- MAXAMP
- NEWTOL
- NOPIV
- OFF
- PIVOT
- PIVREF
- PIVREL
- PIVTOL
- RESMIN
- SPARSE

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 10, “Transient Analysis”.

Table 9-10 DC Initialization Control Options (Sheet 1 of 5)

Parameter	**Description**
ABSTOL = *x*	Sets the absolute node voltage error tolerance, for DC and transient analysis. Decrease ABSTOL, if accuracy is more important than convergence time. ABSTOL is the same as ABSI.
CAPTAB	Prints single-plate node capacitances, for diodes, BJTs, JFETs, MOSFETs, and passive capacitors, at each operating point.
CSHDC	Same option as CSHUNT, but used only with the CONVERGE option.
DCCAP	Generates C-V plots. Prints capacitance values of a circuit (both model and element) during a DC analysis. You can use a DC sweep of the capacitor, to generate C-V plots. Default = 0 (off).
DCFOR = *x*	Use with DCHOLD and .NODESET to enhance DC convergence. DCFOR sets how many iterations to calculate, after a circuit converges in a steady state. The number of iterations after convergence is usually zero. DCFOR adds iterations (and computing time) when calculating a DC circuit solution, to ensure that a circuit did not falsely converge. Default = 0.
DCHOLD = *x*	Use DCFOR and DCHOLD together, to initialize DC analysis. DCFOR and DCHOLD enhance convergence properties in DC simulation. DCFOR and DCHOLD work with .NODESET. DCHOLD specifies how many iterations to hold a node, at the .NODESET voltage values. The effects of DCHOLD on convergence differ, according to the DCHOLD value, and the number of iterations before DC converges.

Table 9-10 DC Initialization Control Options (Sheet 2 of 5)

Parameter	Description
	If a circuit converges in a steady state, in fewer than DCHOLD iterations, the DC solution includes the values set in .NODESET. If the circuit requires more than DCHOLD iterations to converge, HSPICE ignores the .NODESET values, and calculates the DC solution, using the .NODESET fixed-source voltages, open-circuited. Default = 1.
DCSTEP = *x*	Converts DC model and element capacitors to a conductance, to enhance DC convergence. HSPICE divides the value of the element capacitors by DCSTEP, to model DC conductance. Default = 0 (seconds). \
DV = *x*	Maximum iteration-to-iteration voltage change, for all circuit nodes, in both DC and transient analysis. High-gain bipolar amplifiers can require values of 0.5 to 5.0, to achieve a stable DC operating point. Large CMOS digital circuits frequently require about 1 volt. Default = 1000 (or 1e6 if DCON = 2).
GRAMP = *x*	HSPICE sets the value during auto-convergence (default=0). Use GRAMP, with the GMINDC convergence-control option, to find the smallest GMINDC value that results in DC convergence. For a description of GMINDC, see on page 9-28.
	GRAMP specifies the conductance range, over which DC operating point analysis sweeps GMINDC. HSPICE replaces GMINDC values over this range, simulates each value, and uses the lowest GMINDC value where the circuit converges in a steady state. • If you sweep GMINDC between 1e-12 mhos (default) and 1e-6 mhos, GRAMP is 6 (value of the exponent difference, between the default and the maximum conductance limit). In this example: • HSPICE first sets GMINDC to 1e-6 mhos, and simulates the circuit. • If circuit simulation converges, HSPICE sets GMINDC to 1e-7 mhos, and simulates the circuit. The sweep continues until HSPICE simulates all values on the GRAMP ramp. If the combined GMINDC and GRAMP conductance is greater than 1e-3 mho, false convergence can occur.
GSHUNT	Conductance added from each node, to ground. Default is zero. Add a small GSHUNT value to each node, to help solve internal timestep too small problems, caused by high-frequency oscillations or numerical noise.
ICSWEEP	Saves the current analysis result of a parameter or temperature sweep, as the starting point in the next analysis in the sweep. • If ICSWEEP = 1 (default), the next analysis uses the current results. • If ICSWEEP = 0, the next analysis does not use current analysis results.

Table 9-10 DC Initialization Control Options (Sheet 3 of 5)

Parameter	Description
ITLPTRAN	Controls the iteration limit used in the final try of the pseudo-transient method, in OP or DC analysis. If simulation fails in the final try of the pseudo-transient method, enlarge this option. Default is 30.
ITL1 = x	Maximum DC iterations. Increasing this value rarely improves convergence in small circuits. Values as high as 400 can result in convergence for some large circuits with feedback (such as operational or sense amplifiers). However, to converge, most models do not require more than 100 iterations. Set .OPTION ACCT to list how many iterations an operating point requires. Default is 200.
ITL2 = val	Iteration limit for the DC transfer curve. Increasing this limit improves convergence, *only* for very large circuits. Default is 50.
KCLTEST	Activates a KCL (Kirchhoff's Current Law) test. This test increases simulation time, especially for large circuits, but accurately checks the solution. Default = 0 If you set this value to 1, HSPICE sets these options: • Sets RELMOS and ABSMOS options to 0 (off). • Sets ABSI to 1e-6 A. • Sets RELI to 1e-6. To satisfy the KCL test, each node must satisfy this condition: $\left\|\Sigma i_b\right\| < \mathrm{RELI} \cdot \Sigma\left\|i_b\right\| + \mathrm{ABSI}$ ibs are the node currents.
MAXAMP = x	Sets the maximum current, through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, HSPICE reports an error. Default = 0.0.
NEWTOL	Calculates one or more iterations past convergence, for every calculated DC solution and timepoint circuit solution. If you do not set NEWTOL, after HSPICE determines convergence, the convergence routine ends, and the next program step begins. Default is 0.
NOPIV	Prevents HSPICE from automatically switching to pivoting-matrix factoring, if a nodal conductance is less than PIVTOL. NOPIV inhibits pivoting (see PIVOT).
OFF	For all active devices, initializes terminal voltages to zero, if you did not initialize them to other values. For example, if you did not initialize the drain and source nodes of a transistor (using .NODESET or .IC statements, or connecting them to sources), then OFF initializes all nodes of the transistor to zero. HSPICE checks the OFF option before element IC parameters. If you assign an element IC parameter to a node, simulation initializes the node to the element IC parameter value, even if the OFF option has set it o zero. Use the OFF element parameter to initialize terminal voltages to zero (for specific active devices), or for exact DC operating-point solutions for large circuits.

Table 9-10 DC Initialization Control Options (Sheet 4 of 5)

Parameter	Description
PIVOT = x (same as SPARSE = x)	Selects a pivot algorithms. Use these algorithms to reduce simulation time, and to achieve convergence in circuits that produce hard-to-solve matrix equations. To select the pivot algorithm, set PIVOT to one of these values: 0: Original non-pivoting algorithm. 1: Original pivoting algorithm. 2: Picks the largest pivot in the row. 3: Picks the best pivot in a row. 10 (default): Fast, non-pivoting algorithm; requires more memory. 11: Fast, pivoting algorithm; requires more memory than PIVOT values less than 11. 12: Picks the largest pivot in the row; requires more memory than PIVOT values less than 12. 13: Fast, best pivot: faster; uses more memory than PIVOT values less than 13. The fastest algorithm is PIVOT = 13, which can improve simulation time up to ten times, on very large circuits. However, PIVOT = 13 requires substantially more memory for simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might require pivoting. If PIVTOL = 0, HSPICE automatically changes from non-pivoting, to a row-pivot strategy, if it detects any diagonal-matrix entry less than PIVTOL. This strategy provides the time and memory advantages of non-pivoting inversion, and avoids unstable simulations and incorrect results. Use .OPTION NOPIV, to prevent HSPICE from pivoting. For very large circuits, PIVOT = 10, 11, 12, or 13, can require excessive memory. If HSPICE switches to pivoting during a simulation, it prints the message: pivot change on the fly followed by the node numbers that cause the problem. Use .OPTION NODE to cross-reference a node to an element. SPARSE is the same as PIVOT.
PIVREL = x	Sets the maximum and minimum ratio of a row or matrix. Use only if PIVOT = 1. Large values for PIVREL can result in very long matrix-pivot times. If the value is too small, however, no pivoting occurs. Start with small values of PIVREL, using an adequate (but not excessive) value, for convergence and accuracy. Default = 1E-20 (max = 1e-20, min = 1).
PIVREF	Pivot reference. Use PIVREF in PIVOT = 11, 12, or 13, to limit the size of the matrix. Default = 1e+8.

Table 9-10 DC Initialization Control Options (Sheet 5 of 5)

Parameter	**Description**
PIVTOL = x	Absolute minimum value for which HSPICE accepts a matrix entry as a pivot. If PIVOT=0, PIVTOL is the minimum conductance in the matrix. Default=1.0e-15. PIVTOL must be less than GMIN or GMINDC. Values that approach 1 increase the pivot.
RESMIN = x	Minimum resistance for all resistors, including parasitic and inductive resistances. Default = 1e-5 (ohm). Range: 1e-15 to 10 ohm.
SPARSE = x (same as PIVOT = x)	Same as PIVOT.

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 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 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 9-11 shows, HSPICE 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 9-28 describes the options in Table 9-11.

Table 9-11 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 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 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 9-12 shows an example of the relationship between the RELVDC value, and the number of iterations.

Table 9-12 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 10-24.

Table 9-13 Convergence Control Options (Sheet 1 of 3)

Parameter	**Description**
ABSH = x	Sets the absolute current change, through voltage-defined branches (voltage sources and inductors). Use ABSH with DI and RELH, to check for current convergence. Default is 0.0.
ABSI = x	Sets the absolute error tolerance for branch currents, in diodes, BJTs, and JFETs, during DC and transient analysis. Decrease ABSI, if accuracy is more important than convergence time. To analyze currents less than 1 nanoamp, change ABSI to a value at least two orders of magnitude smaller than the minimum expected current. Default is 1e-9 for KCLTEST = 0, or 1e-6 for KCLTEST = 1.
ABSMOS = x	Current error tolerance (for MOSFET devices), in DC or transient analysis. The ABSMOS setting determines whether the drain-to-source current solution has converged. The drain-to-source current converged if: • The difference between the drain-to-source current in the last iteration, versus the present iteration, is less than ABSMOS, or • This difference is greater than ABSMOS, but the percent change is less than RELMOS. If other accuracy tolerances also indicate convergence, HSPICE solves the circuit at that timepoint, and calculates the next timepoint solution. For low-power circuits, optimization, and single transistor simulations, set ABSMOS = 1e-12. Default is 1e-6 (amperes).
ABSVDC = x	Sets the minimum voltage for DC and transient analysis. If accuracy is more critical than convergence, decrease ABSVDC. If you need voltages less than 50 micro-volts, reduce ABSVDC to two orders of magnitude less than the smallest voltage. This ensures at least two significant digits. Typically, you do not need to change ABSVDC, unless you simulate a high-voltage circuit. For 1000-volt circuits, a reasonable value is 5 to 50 millivolts. Default=VNTOL (VNTOL default = 50 mV).
CONVERGE	Invokes different methods to solve non-convergence problems • CONVERGE = -1: Use with DCON = -1, to disable autoconvergence. • CONVERGE = 0” Autoconvergence (default). • CONVERGE = 1: Uses the Damped Pseudo Transient algorithm. If simulation does not converge within the amount of CPU time (set in the CPTIME control option), then simulation halts. • CONVERGE = 2: Uses a combination of DCSTEP and GMINDC ramping. Not used in the autoconvergence flow. • CONVERGE = 3: Invokes the source-stepping method. Not used in the autoconvergence flow.

Table 9-13 Convergence Control Options (Sheet 2 of 3)

Parameter	Description
	• CONVERGE = 4: Uses the gmath ramping method. Even you did not set it in an .OPTION statement, the CONVERGE option activates if a matrix floating-point overflows, or if HSPICE reports a timestep too small error. Default = 0. If a matrix floating-point overflows, CONVERGE = 1.
DCON = x	If a circuit cannot converge, HSPICE sets DCON = 1, and calculates: $DV = \max\left(0.1, \frac{V_{max}}{50}\right)$, if DV = 1000 $GRAMP = \max\left(6, \log_{10}\left(\frac{I_{max}}{GMINDC}\right)\right) \qquad ITL1 = ITL1 + 20 \cdot GRAMP$ Vmax is maximum voltage, and Imax is maximum current. If the circuit cannot converge, HSPICE sets DCON = 2, which sets DV = 1e6. If a circuit contains discontinuous models or uninitialized flip-flops, simulation might not converge. Set DCON = -1 and CONVERGE = -1, to disable auto-convergence. HSPICE then lists non-convergent nodes and devices.
DCTRAN	DCTRAN is an alias for CONVERGE. See CONVERGE.
DI = x	Sets the maximum iteration-to-iteration change in current, through voltage-defined branches (voltage sources and inductors). Use his option only if the value of the ABSH control option is greater than 0. Default = 0.0.
GMAX = x	Conductance, in parallel with a current source, for .IC and .NODESET initialization circuitry. Some large bipolar circuits require you to set GMAX=1, for convergence. Default=100 (mho).
RELH = x	Sets relative tolerance for currents, through voltage-defined branches (voltage sources and inductors). Use RELH to check current convergence, but only if the value of the ABSH control option is greater than zero. Default = 0.05.
GMINDC = x	Conductance in parallel to all pn junctions, and all MOSFET nodes except *gate* (see Figure 4-2 on page 4-42), for DC analysis. GMINDC helps overcome DC convergence problems, caused by low values of off-conductance, for pn junctions and MOSFETs. You can use GRAMP to reduce GMINDC, by one order of magnitude, for each step. Set GMINDC between 1e-4 and the PIVTOL value. Default = 1e-12. Large values of GMINDC can cause unreasonable circuit response. If your circuit requires large values to converge, suspect a bad model or circuit. If a matrix floating-point overflows, and if GMINDC is 1.0e-12 or less, HSPICE sets it to 1.0e-11. HSPICE manipulates GMINDC in auto-converge mode (see Autoconverge Process on page 9-31).

Table 9-13 Convergence Control Options (Sheet 3 of 3)

Parameter	Description
RELI = x	Sets the relative error/tolerance change, from iteration to iteration. This value determines convergence for all currents, in diode, BJT, and JFET devices. (RELMOS sets the tolerance for MOSFETs). This is the percent change in current, from the value calculated at the previous timepoint. • Default = 0.01 for KCLTEST = 0. • Default = 1e-6 for KCLTEST = 1.
RELMOS = x	Sets the relative error tolerance (percent) for drain-to-source current, from iteration-to-iteration. This parameter determines convergence, for currents in MOSFET devices. (RELI sets the tolerance for other active devices.) Sets the change in current, from the value calculated at the previous timepoint. HSPICE uses the RELMOS value, only if the current is greater than the ABSMOS floor value. Default = 0.05.
RELV = x	Sets the relative error tolerance, for voltages. If voltage or current exceeds the absolute tolerance, a RELV test determines convergence. Increasing RELV increases the relative error. You should generally maintain RELV at its default value. RELV conserves simulator charge. For voltages, RELV is the same as RELTOL. Default = 1e-3.
RELVDC = x	Sets the relative error tolerance, for voltages. If voltage or current exceeds their absolute tolerances, the RELVDC test determines convergence. Increasing RELVDC increases the relative error. You should generally maintain RELVDC at its default value. RELVDC conserves simulator charge. Default = RELTOL (RELTOL default = 1e-3).

Autoconverge Process

If a circuit does not converge in the number of iterations that *ITL1* specifies, HSPICE initiates an auto-convergence process. This process manipulates DCON, GRAMP, and GMINDC, and even CONVERGE in some cases. Figure 9-3 on page 9-33 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.

Notes:

1. Setting .OPTION DCON = -1 disables steps 2 and 3.
2. Setting .OPTION CONVERGE = -1 disables steps 4 and 5.
3. Setting .OPTION DCON = -1 CONVERGE = -1 disables steps 2, 3, 4, and 5.
4. 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.
5. Setting GRAMP in an .OPTION statement has no effect on autoconverge. Autoconverge sets GRAMP independently.
6. If you set a GMINDC value in an .OPTION statement, GMINDC ramps to the value you set, instead of to 1e-12, in steps 2 and 3.

DCON and GMINDC

GMINDC helps stabilize the circuit, during DC operating-point analysis. For MOSFETs, GMINDC helps stabilize the device, in the vicinity of the threshold region. HSPICE inserts GMINDC between:

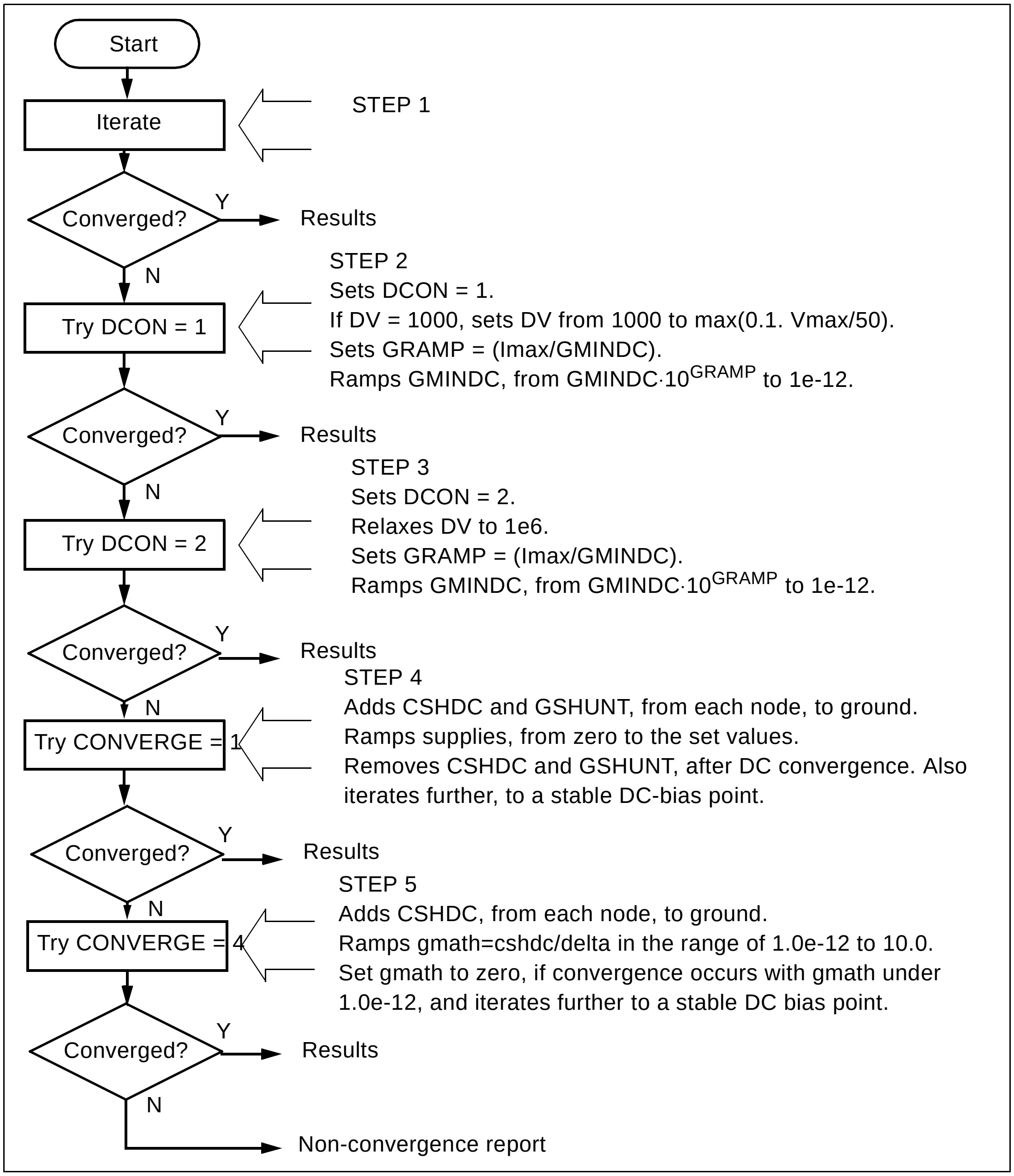
- Drain and bulk.
- Source and bulk.
- Drain and source.

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.

Figure 9-3 Autoconvergence Process Flow Diagram



STEP 1

STEP 2
Sets DCON = 1.
If DV = 1000, sets DV from 1000 to max(0.1. Vmax/50).
Sets GRAMP = (Imax/GMINDC).
Ramps GMINDC, from GMINDC·10^{GRAMP} to 1e-12.

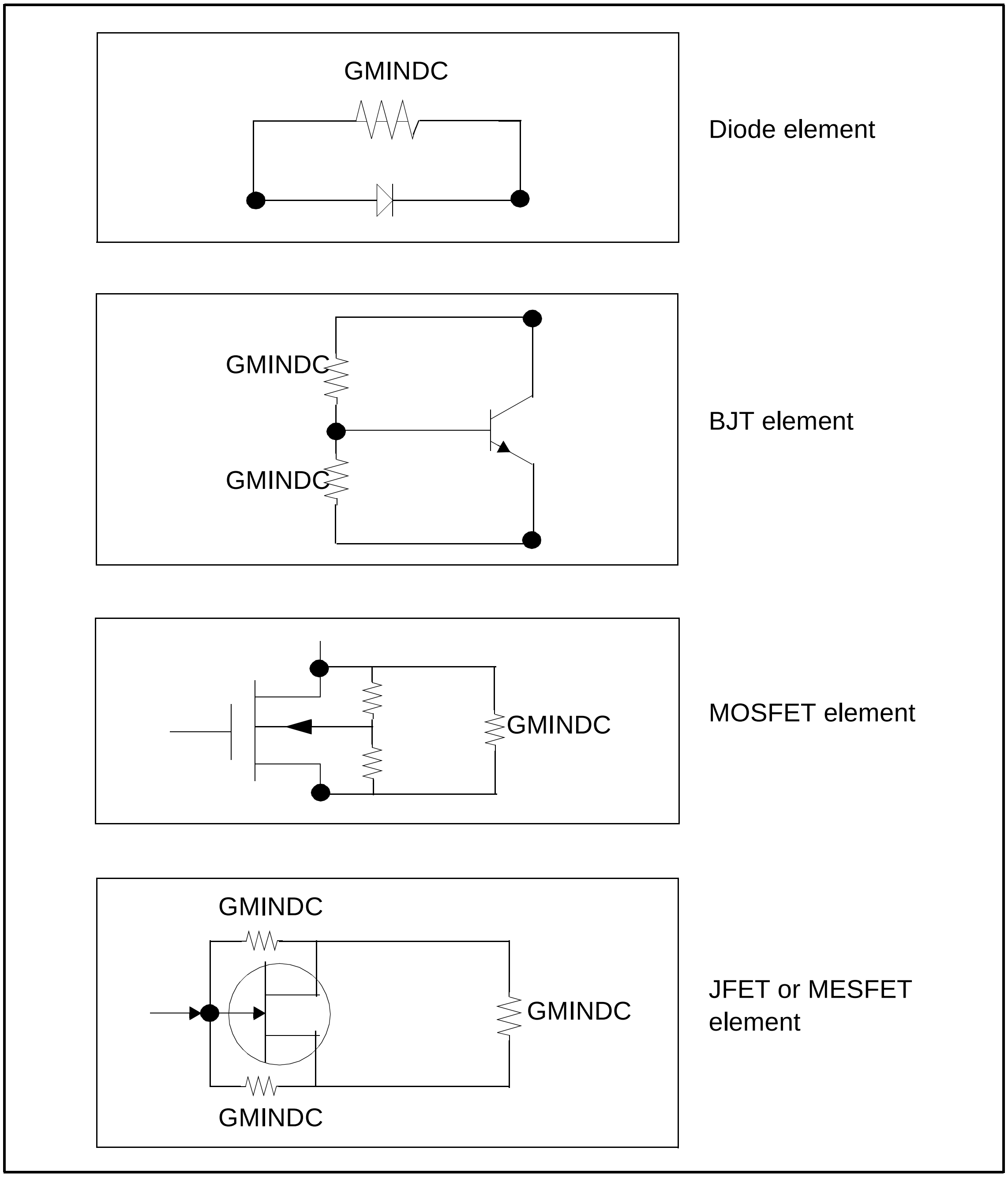
STEP 3
Sets DCON = 2.
Relaxes DV to 1e6.
Sets GRAMP = (Imax/GMINDC).
Ramps GMINDC, from GMINDC·10^{GRAMP} to 1e-12.

STEP 4
Adds CSHDC and GSHUNT, from each node, to ground.
Ramps supplies, from zero to the set values.
Removes CSHDC and GSHUNT, after DC convergence. Also iterates further, to a stable DC-bias point.

STEP 5
Adds CSHDC, from each node, to ground.
Ramps gmath=cshdc/delta in the range of 1.0e-12 to 10.0.
Set gmath to zero, if convergence occurs with gmath under 1.0e-12, and iterates further to a stable DC bias point.

If a circuit does not converge, HSPICE automatically sets the DCON option. This option invokes GMINDC ramping, in steps 2 and 3 of Figure 9-3 on page 9-33. Figure 9-4 shows GMINDC, for various elements.

Figure 9-4 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 True-Hspice BSIM1, BSIM2, and Level 28 device models)?
   - Avoid setting 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/M2.
   - 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, 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 options 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 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 can solve them.

Modeling capacitors as complete opens, can result in this error:

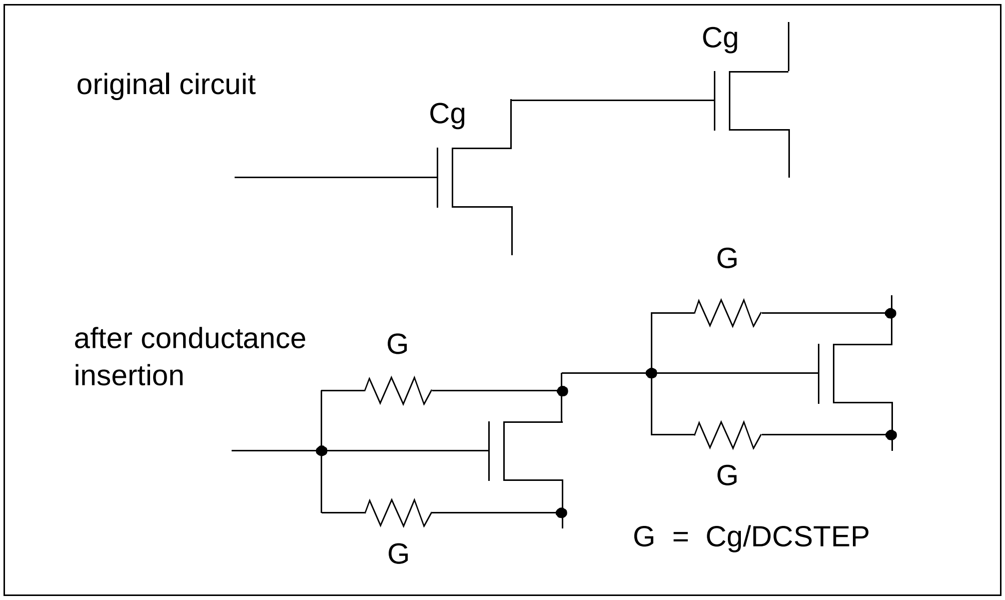
```
“No DC Path to Ground”
```

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 9-5 on page 9-38, HSPICE inserts conductance (G), in parallel with capacitance (C_g). This provides current paths around capacitances, in DC analysis.

Figure 9-5 Conductance Insertion



Floating-Point Overflow

If MOS conductance is negative or zero, HSPICE might have difficulty converging. An indication of this type of problem is a floating-point overflow, during matrix solutions. HSPICE 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 sets it to 1.0e-11.

Diagnosing Convergence Problems

Before simulation, HSPICE diagnoses potential convergence problems in the input circuit, and provides an early warning, to help you in debugging your circuit. If HSPICE 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 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, which lists all non-convergent elements, and their associated element currents, element voltages, model parameters, and current error tolerances (*tol*).

1. To locate the branch current or nodal voltage that causes non-convergence, 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 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 sub-circuit, model, and element name, for all parts of the circuit where node voltages or currents do not converge.

For example, Table 9-14 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 9-14, 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 9-14 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

Table 9-14 Subcircuit Voltage, Current, and Tolerance (Continued)

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**
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 statement), to find the sources of non-convergence. When you increase or decrease ITL1, HSPICE 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:

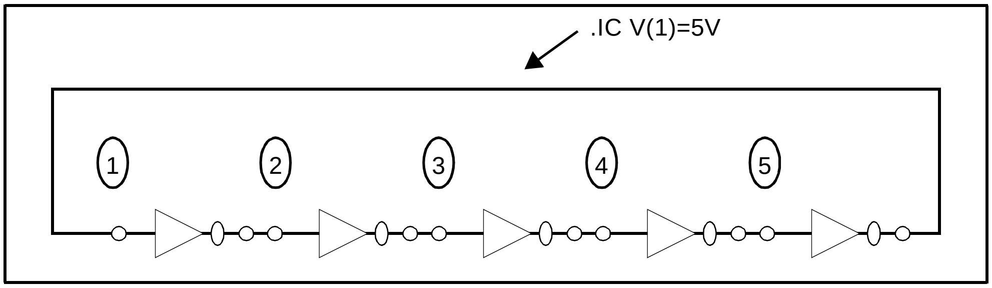
- Poor Initial Conditions
- Inappropriate Model Parameters
- PN Junctions (Diodes, MOSFETs, BJTs)

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 9-6 Ring Oscillator



The best way to set up the flip-flop is to use an .IC statement in the subcircuit definition.

EXAMPLE:

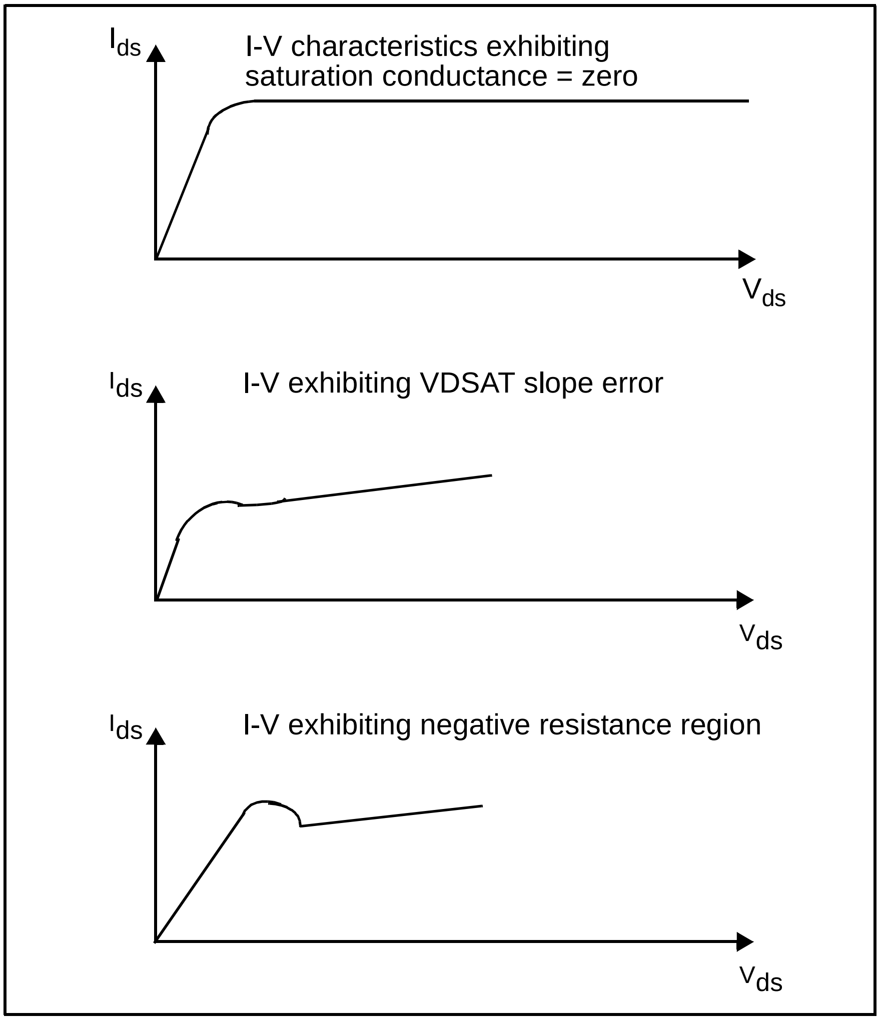
The following example sets the local Qset parameter to 0, and uses this value for the .IC statement, to initialize the Q latch output node. As a result, all latches have a default state of Q low. Setting Qset to vdd calls a latch, which overrides this state.

```
.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 9-7 Discontinuous I-V Characteristics



If the simulation no longer converges, when you add a component or change a component value, then the model parameters are inappropriate, or do not correspond to the physical values that they represent.

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, the GMINDC and GMIN options 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.

10

Transient Analysis

Transient analysis computes the circuit solution, as a function of time, over a time range specified in the .TRAN statement.

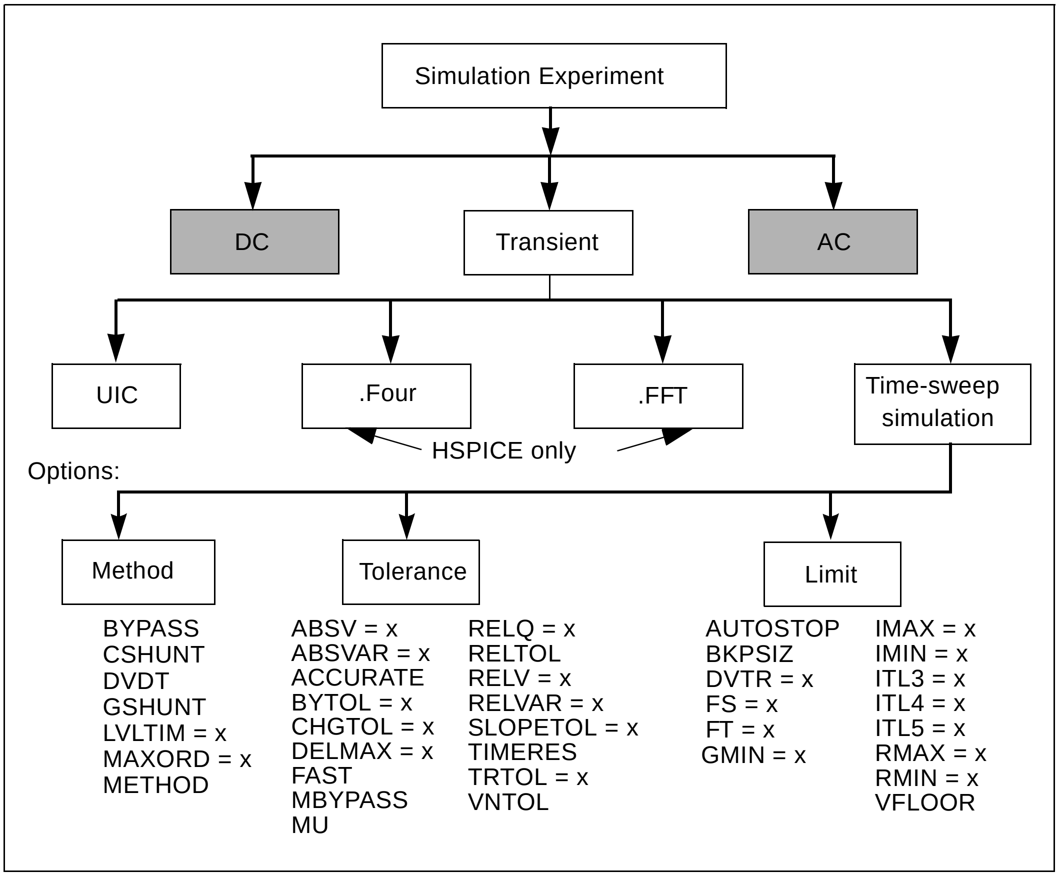
This chapter explains the following topics:

- Simulation Flow
- Overview of Transient Analysis
- Using the .TRAN Statement
- Transient Analysis of an RC Network
- Transient Analysis of an Inverter
- Using the .BIASCHK Statement
- Transient Control Options
- Simulation Speed and Accuracy
- Numerical Integration Algorithm Controls
- Selecting Timestep Control Algorithms
- Fourier Analysis

Simulation Flow

Figure 10-1 illustrates the simulation flow, for transient analysis in Synopsys HSPICE.

Figure 10-1 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 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.

Using the .TRAN Statement

SYNTAX:

Single-Point Analysis

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC>
```

Double-Point Analysis

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC>
+ <SWEEP var type np pstart pstop>
```

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC> <SWEEP var START="param_expr1"
+ STOP="param_expr2"
+ STEP="param_expr3">
```

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ... tincrN tstopN>
+ <START=val> <UIC> <SWEEP var start_expr stop_expr
+ step_expr>
```

Data-Driven Sweep

HSPICE supports the following types of data-driven sweep syntax:

```
.TRAN DATA = datanm
```

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC> <SWEEP DATA = datanm>
```

```
.TRAN DATA = datanm<SWEEP var type np pstart pstop>
```

```
.TRAN DATA=datanm <SWEEP var START="param_expr1"
+STOP="param_expr2" STEP="param_expr3">
```

```
.TRAN DATA=datanm <SWEEP var start_expr stop_expr step_expr>
```

Monte Carlo Analysis

HSPICE supports this Monte Carlo syntax for transient analysis.

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC><SWEEP MONTE = val>
```

Optimization

HSPICE supports this Optimization syntax for transient analysis.

```
.TRAN DATA = datanm OPTIMIZE = opt_par_fun
+ RESULTS = measnames MODEL = optmod
```

```
.TRAN  <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

.TRAN Keywords and Parameters

Transient sweep specifications can include these keywords and parameters:

Table 10-1   Keywords and Parameters in a Transient Sweep

Parameter	**Description**
DATA = datanm	Data name, referenced in the .TRAN statement.
MONTE = val	Produces a specified number (*val*) of randomly-generated values. HSPICE uses them to select parameters from a *Gaussian*, *Uniform*, or *Random Limit.*
np	Number of points, or number of points per decade or octave, depending on what keyword precedes it.
param_expr...	Expressions you specify: *param_expr1...param_exprN*.
pincr	Voltage, current, element, or model parameter; or any temperature increment value. If you set the *type* variation, use *np* (number of points), not *pincr*.

Table 10-1 Keywords and Parameters in a Transient Sweep (Continued)

Parameter	**Description**
pstart	Starting voltage, current, or temperature; or any element or model parameter value. If you set the *type* variation to POI (list of points), use a list of parameter values, instead of *pstart pstop*.
pstop	Final voltage, current, or temperature; or element or model parameter value.
START	Time when printing or plotting begins. The START keyword is optional: you can specify a start time without the keyword. If you use .TRAN with .MEASURE, a non-zero START time can cause incorrect .MEASURE results. Do not use non-zero START times in .TRAN statements, when you also use .MEASURE.
SWEEP	Keyword. Indicates that .TRAN specifies a second sweep.
tincr1…	Specifies the printing or plotting increment for printer output, and the suggested computing increment for post-processing.
tstop1…	Time when a transient analysis stops incrementing by the first specified time increment (*tincr1*). If another tincr-tstop pair follows, analysis continues with a new increment.
UIC	Uses the nodal voltages specified in the .IC statement (or in the *IC* = parameters of the various element statements) to calculate initial transient conditions, rather than solving for the quiescent operating point.
type	Specifies any of the following keywords: • DEC – decade variation. • OCT – octave variation (the value of the designated variable is eight times its previous value). • LIN – linear variation. • POI – list of points.
var	Name of an independent voltage or current source, any element or model parameter, or the TEMP keyword (indicating a temperature sweep). You can use a source value sweep, referring to the source name (SPICE style). However, if you specify a parameter sweep, a .DATA statement, and a temperature sweep, you must choose a parameter name for the source value, and subsequently refer to it in the .TRAN statement. The parameter name must not start with V or I.

.TRAN Examples

1. The following example performs and prints the transient analysis, every 1 ns, for 100 ns.

   ```
   .TRAN 1NS 100NS
   ```

2. The following example performs the calculation every 0.1 ns, for the first 25 ns; and then every 1 ns, until 40 ns. Printing and plotting begin at 10 ns.

   ```
   .TRAN .1NS 25NS 1NS 40NS START = 10NS
   ```

3. The following example performs the calculation every 10 ns, for 1 µs. This example bypasses the initial DC operating point calculation. It uses the nodal voltages, specified in the .IC statement (or by IC parameters in element statements), to calculate the initial conditions.

   ```
   .TRAN 10NS 1US UIC
   ```

4. The following example increases the temperature by 10 °C, through the range -55 °C to 75 °C. It also performs transient analysis, for each temperature.

   ```
   .TRAN 10NS 1US UIC SWEEP TEMP -55 75 10
   ```

5. The following example analyzes each load parameter value, at 1 pF, 5 pF, and 10 pF.

   ```
   .TRAN 10NS 1US SWEEP load POI 3 1pf 5pf 10pf
   ```

6. The following example is a data-driven time sweep. It uses a data file as the sweep input. If the parameters in the data statement are controlling sources, then a piecewise linear specification must reference them.

   ```
   .TRAN data = dataname
   ```

Table 10-2 .TRAN Options

Option	Description
BYPASS	Bypasses model evaluations for MOSFETs, if terminal voltages do not change. Can be `0` (off) or `1` (on, default).
CSHUNT	Capacitance, added from each node to ground. Add a small CSHUNT to each node, to solve some *internal timestep too small* problems, caused by high-frequency oscillations or numerical noise. Default is 0.
AUTOSTOP	If on, .TRAN simulation stops when it finds all .MEASURE results. Can be 0 (off) or 1 (on). Default is 0.
GMIN	Minimum conductance added to all PN junctions, for a time sweep in transient analysis. Default = 1e-12.

.TRAN Output Syntax

```
.print tran ov1 [ov2 ... ovN]
.probe tran ov1 [ov2 ... ovN]
.measure tran measspec
.plot tran ov1 [ov2 ... ovN]
.graph tran ov1 [ov2 ... ovN]
```

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.
- V*n*(*d1*): voltage at *n*th terminal of the *d1* device.
- I*n*(*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 .admrc) to specify multiple output variables in a single command. Output is affected by .OPTION post, .OPTION probe

Table 10-3 .TRAN Output Format/Description

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

You can run a transient analysis, using an 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
```

Note: The V1 source specification includes a pulse source. For the syntax of pulse sources and other types of sources, see Sources and Stimuli on page 5-1.

2. To run HSPICE, type the following:

```
hspice quickTRAN.sp > quickTRAN.lis
```

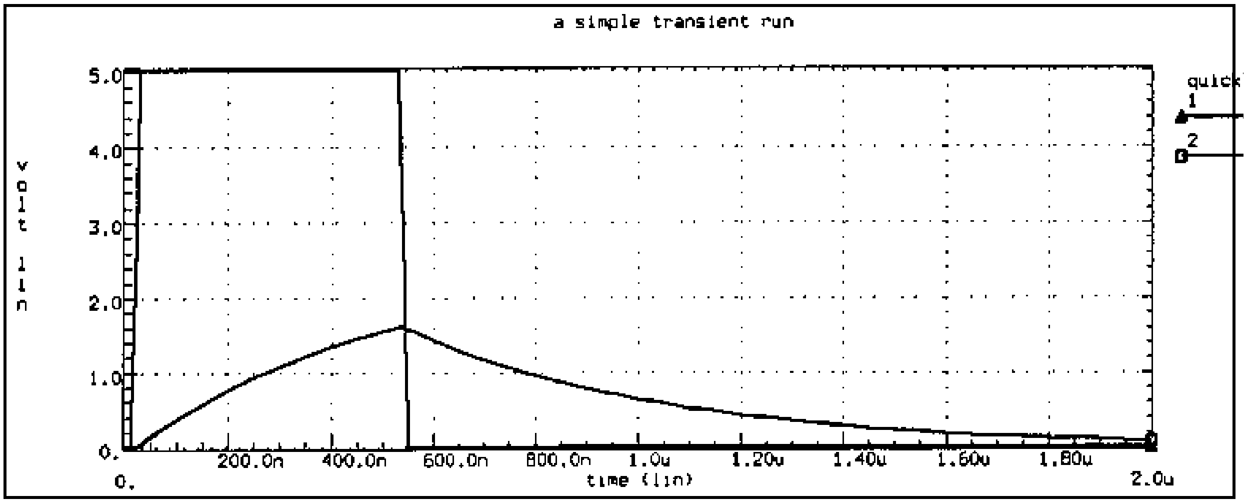
3. To examine the simulation results and status, use an editor and view the .lis and .st0 files.

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 10-2 on page 10-10 shows the waveforms.

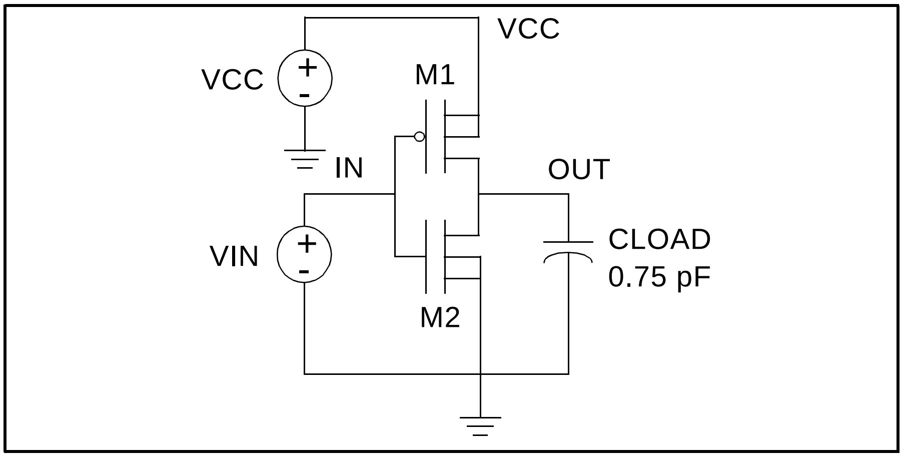
Figure 10-2 Voltages at RC Network Circuit Node 1 and Node 2



Transient Analysis of an Inverter

As a final example, analyze the behavior of the simple MOS inverter shown in Figure 10-3.

Figure 10-3 MOS Inverter Circuit



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
   ```

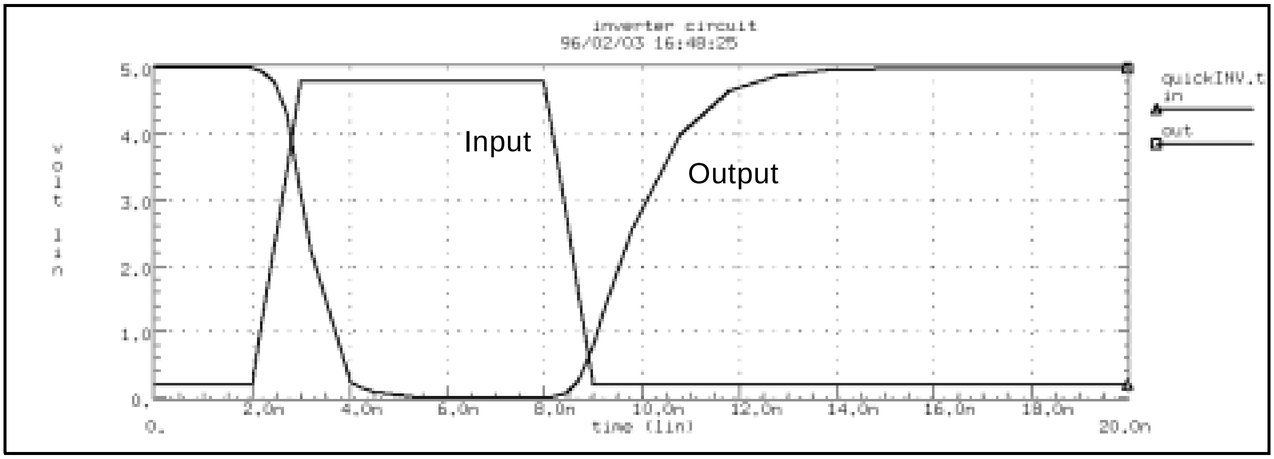
2. To run HSPICE, type the following:

   ```
   hspice quickINV.sp > quickINV.lis
   ```

3. Use AvanWaves to examine the voltage waveforms, at the inverter IN and OUT nodes.

   Figure 10-4 shows the waveforms.

Figure 10-4 Voltage at MOS Inverter Node 1 and Node 2



Using the .BIASCHK Statement

Breakdown can occur if a voltage bias between some terminals of an element is too large. The .BIASCHK statement monitors the voltage bias, using the limits and noise that you define. Bias monitoring checks the specified bias, 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

HSPICE saves the information as both a warning and a BIASCHK summary, in the *.lis file.

You can use this command *only* for MOS and capacitors.

EXAMPLE:

A .BIASCHK statement might check for voltages that exceed a specified limit, for MOS dielectric breakdown. BIASCHK can check voltages from the gate, to the source, drain, or bulk.

BIASCHK cannot detect the bias that exceeds the limit, if the bias is always the same value during transient analysis.

If a model name, referenced in an active element statement, contains a period (.), then .BIASCHK reports an error. This occurs because it is unclear whether a reference such as x.123 is a model name or a sub-circuit name (123 model in the x sub-circuit).

Instance (element) and model names can contain wildcards, either ? (stands for one character) or * (stands for 0 or more characters).

SYNTAX:

```
.biaschk type terminal1=t1 terminal2=t2 limit=lim
+ <noise=ns><name=devname1><name=devname2>...
+ <mname=modelname1><mname=modelname2> ...
```

Table 10-4 .BIASCHK Syntax

Parameter	Description
type	Element type to check MOS (R, C ...) The *type* can be NMOS, PMOS, or C.
terminal 1, 2	Terminals, between which HSPICE checks (that is, checks between *terminal1* and *terminal2*): • For MOS level 57: *nd, ng, ns, ne, np, n6* • For MOS level 58: *nd, ngf, ns, ngb* • For MOS level 59: *nd, ng, ns, ne, np* • For other MOS level: *nd, ng, ns, nb* • For Capacitor: *n1, n2*
limit	Biaschk limit that you define. Reports an error, if the bias voltage (between appointed terminals, of appointed elements and models), is larger than the limit.
noise	Biaschk noise that you define. The default is 0.1v. Noise-filter some of the results (the local maximum bias voltage, that is larger than the limit). The next local max replaces the local max, if all of the following conditions are satisfied: local_max-local_min<noise>. next local_max-local_min<noise>. This local max is smaller than the next local max.
name	Element name to check.
mname	Model name. HSPICE checks elements of the model for bias.

If you do not set name and mname, HSPICE checks all elements of this type for bias voltage (you must include type in the .biaschk card).

You can use a wild card, to describe name and mname, in the biaschk card.

- ? stands for one character.
- * stands for 0 or more characters.

EXAMPLE:

```
.biaschk NMOS terminal1=ng terminal2=nb limit=2v
+ noise=0.01v name=x1.x3.m1 mname=nch.1 name=m3
```

Options for the .biaschk Command

biasfile Option

- If you use this option, HSPICE outputs the results of all .biaschk commands to a file that you specify.
- If you do not set this option, HSPICE outputs the results to the *.lis file.

EXAMPLE:

```
.option biasfile='biaschk/mos.bias'
```

biawarn Option

- If you set this option to 1, HSPICE immediately outputs a warning message, when any local max bias voltage exceeds the limit *during* transient analysis. *After* this transient analysis, HSPICE outputs the results summary, as filtered by noise.
- If you set this option to 0 (the default), HSPICE does not output a warning message *during* transient analysis. HSPICE outputs the results, *after* this transient analysis.

EXAMPLE:

```
.option biawarn=1
```

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.:

Table 10-5 Transient Control Options, Arranged by Category

Method	Tolerance		Limit	
BYPASS	ABSH	RELQ	AUTOSTOP	ITL3
CSHUNT	ABSV	RELTOL	BKPSIZ	ITL4
DVDT	ABSVAR	RELV	DELMAX	ITL5
GSHUNT	ACCURATE	RELVAR	DVTR	RMAX
INTERP	BYTOL	SLOPETOL	FS	RMIN
ITRPRT	CHGTOL	TIMERES	FT	VFLOOR
LVLTIM	DI	TRTOL	GMIN	
MAXORD	FAST	VNTOL		
METHOD	MBYPASS			
PURETP	MAXAMP			
	MU			
TRCON	RELH			
	RELI			

Table 10-6 Method Options (Sheet 1 of 4)

Option	Description
BYPASS	Bypasses model evaluations, if the terminal voltages do not change. Can be 0 (off) or 1 (on). To speed-up simulation, this option does not update the status of latent devices. To enable bypassing, set .OPTION BYPASS = 1, for MOSFETs, MESFETs, JFETs, BJTs, or diodes. Default = 1. Use the BYPASS algorithm cautiously. Some circuits might not converge and might lose accuracy in transient analysis and operating-point calculations.
CSHUNT	Capacitance added from a node to ground in HSPICE. Add a small CSHUNT to each node, to solve *internal timestep too small* problems caused by high-frequency oscillations or numerical noise. Default=0.
GSHUNT	Conductance added from each node, to ground. Default is zero. Add a small GSHUNT value to each node, to help solve *internal timestep too small* problems, caused by high-frequency oscillations or numerical noise.

Table 10-6 Method Options (Sheet 2 of 4)

Option	**Description**
DVDT	Adjusts the timestep, based on rates of change for node voltages.: 0 - original algorithm. 1 - fast. 2 - accurate. 3,4 - balance speed and accuracy. Default is 4.
INTERP	Limits output for post-analysis tools, such as Cadence or Zuken, to only the .TRAN timestep intervals. By default, HSPICE outputs all convergent iterations. INTERP typically produces a much smaller design.tr# file. Use INTERP = 1 with caution, when the netlist includes .MEASURE statements. To compute measure statements, HSPICE uses the post-processing output. Reducing post-processing output can lead to interpolation errors, in measure results. When you run data-driven transient analysis (.TRAN DATA) in an optimization routine, HSPICE forces INTERP to 1. All measurement results are at the time points specified in the data-driven sweep. To measure only at converged internal timesteps (for example, to calculate the AVG or RMS), set ITRPRT = 1.
ITRPRT	Prints output variables. at their internal time points. This option might generate a long output list.
LVLTIM = x	Selects the timestep algorithm, for transient analysis. • LVLTIM = 1 DVDT timestep algorithm. • LVLTIM = 2 timestep algorithm for local truncation error. • LVLTIM = 3 DVDT timestep algorithm, with timestep reversal. To use the GEAR method of numerical integration and linearization, select LVLTIM = 2. To use the TRAP linearization algorithm, select LVLTIM 1 or 3. LVLTIM = 1 (DVDT option) is the default, and helps avoid internal timestep too small non-convergence. The local truncation algorithm (LVLTIM = 2) is more accurate than the TRAP method. If you use this option, errors do not propagate from time point to time point, which can result in an unstable solution.
MAXORD = x	Maximum order of integration, for the `GEAR` method (see `METHOD`). The value of *x* can be `1` or 2. • MAXORD = 1 uses the backward Euler integration method. • MAXORD = 2 (default) is more stable, accurate, and practical.

Table 10-6 Method Options (Sheet 3 of 4)

Option	Description
METHOD = name	Sets the numerical integration method, for a transient analysis, to either GEAR or TRAP. • To use GEAR, set METHOD = GEAR, which sets LVLTIM = 2. • To change LVLTIM from 2, to either 1 or 3, set LVLTIM = 1 or 3, after the METHOD = GEAR option. This overrides METHOD=GEAR, which sets LVLTIM = 2. TRAP (trapezoidal) integration usually reduces program execution time, with more accurate results. However, this method can introduce an apparent oscillation on printed or plotted nodes, which might not be caused by circuit behavior. To test this, run a transient analysis, using a small timestep. If the oscillation disappears, the cause was the trapezoidal method. The GEAR method is a filter, removing oscillations that occur in the trapezoidal method. Highly non-linear circuits (such as operational amplifiers) can require long execution times if you use the GEAR method. Circuits that do not converge in trapezoidal integration, often converge if you use GEAR. Default = TRAP (trapezoidal).
PURETP	Sets the integration method to use, for the reversal time point. Default is 0. If you set puretp=1, then if HSPICE encounters non-convergence, it uses TRAP (instead of B.E) for the reversed time point. Use this option to help some oscillating circuits to oscillate, if the default simulation process cannot satisfy the result. Use this option with the method=TRAP statement.
TRCON	Controls the automatic convergence (*autoconvergence*) and automatic speedup (*autospeedup*) processes in HSPICE. HSPICE also uses autoconvergence in DC analysis, if the Newton-Raphson (N-R) method fails to converge. • TRCON=1 (the default) enables both *autoconvergence* and *autospeedup*. • TRCON= 0 enables *autospeedup* only. • TRCON =-1 disables both *autoconvergence* and *autospeedup*. *Aoutoconvergence* If the circuit fails to converge using the trapezoidal (TRAP) numerical integration method (for example, because of trapezoidal oscillation), HSPICE uses the GEAR method and LTE timestep algorithm, to run the transient analysis again from time=0. This process is called autoconvergence.

Table 10-6 Method Options (Sheet 4 of 4)

Option	Description
	Autoconvergence sets the following options to their default values before the second try: METHOD=GEAR, LVLTIM=2, MBYPASS=1.0, BYPASS=0.0, SLOPETOL=0.5 BYTOL= min{mbypas*vntol and reltol} RMAX=2.0 if it was 5.0 in the first run. Otherwise, RMAX does not change. For some large non-linear circuits with large TSTOP/TSTEP values, analysis might run for an excessively long time. In this case, HSPICE might automatically set a new and bigger RMAX value, to speed up the analysis for primary reference. In most cases, however, HSPICE does not activate this type of autospeedup process. For autospeedup to occur, all three of the following conditions must occur: • N1 (Number of Nodes) > 1,000 • N2 (TSTOP/TSTEP) >= 10,000 • N3 (Total Diodes, BJTs, JFETs and MOSFETs) > 300
	Autospeedup is most likely to occur if the circuit also meets either of the following conditions: • N2 >= 1e+8, and N3 > 500, or • N2 >= 2e+5, and N3 > 1e+4 If HSPICE does activate autospeedup, you might need to disable it. To do this, set TRCON=-1, and increase TSTEP or RMAX (or both), to balance accuracy and speed.

Table 10-7 Tolerance Options (Sheet 1 of 4)

Parameter	Description
ABSH = x	Sets the absolute current change, through voltage- defined branches (voltage sources and inductors). Use ABSH with DI and RELH, to check for current convergence. Default is 0.0.
ABSV = *x*	Absolute minimum voltage, for DC and transient analysis. ABSV is the same as VNTOL. If accuracy is more critical than convergence, decrease VNTOL. If you need voltages less than 50 microvolts, reduce VNTOL to two orders of magnitude less than the smallest desired voltage. This ensures at least two digits of significance. Typically, you do not need to change VNTOL, except to simulate a high-voltage circuit. A reasonable value for 1000-volt circuits is 5 to 50 millivolts. Default is 50 (microvolts).

Table 10-7 Tolerance Options (Sheet 2 of 4)

Parameter	**Description**
ABSVAR = x	Maximum voltage change, from one time point to the next. Use this option with the DVDT algorithm. If the simulator produces a convergent solution that is greater than ABSVAR, it: • Discards the solution. • Sets the timestep to a smaller value. • Recalculates the solution. This is a *timestep reversal*. Default is 0.5 (volts).
BYTOL = x	Voltage tolerance, at which a MOSFET, MESFET, JFET, BJT, or diode becomes latent. HSPICE does not update status of latent devices. Default = MBYPASS x VNTOL.
ACCURATE	Selects a time algorithm that uses LVLTIM=3 and DVDT=2, for circuits such as high-gain comparators. Use this option with circuits that combine high gain and large dynamic range, to guarantee solution accuracy. If you set ACCURATE to 1, HSPICE uses the following control options: • LVLTIM = 3 • DVDT = 2 • RELVAR = 0.2 • ABSVAR = 0.2 • FT = 0.2 • RELMOS = 0.01 Default = 0.
CHGTOL = x	Charge error tolerance, if LVLTIM = 2. Use CHGTOL with RELQ, to set the absolute and relative charge tolerance, for all HSPICE capacitances. Default = 1e-15 (coulomb).
DI = x	Sets the maximum iteration-to-iteration current change, through voltage-defined branches (voltage sources and inductors). Use this option only if the value of the ABSH control option is greater than 0. Default = 0.0.
FAST	To speed-up simulation, this option does not update the status of latent devices. Use this option for MOSFETs, MESFETs, JFETs, BJTs, and diodes. Default = 0. A device is latent if its node voltage variation (from one iteration to the next) is less than the value of the BYTOL control option, or the BYPASSTOL element parameter. (If FAST is on, HSPICE sets BYTOL to different values, for different types of device models.) Besides the FAST option, you can also use NOTOP and NOELCK to reduce input pre-processing time. Increasing the MBYPASS or BYTOL value also helps simulations to run faster, but can reduce accuracy.

Table 10-7 Tolerance Options (Sheet 3 of 4)

Parameter	Description
MAXAMP = x	Maximum current through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, HSPICE reports an error. Default = 0.0.
MBYPASS = x	Computes the default value for the BYTOL option: BYTOL = MBYPASS x VNTOL Also multiplies the RELV voltage tolerance. Set MBYPASS to about 0.1, for precision analog circuits. • Default = 1, for DVDT = 0, 1, 2, or 3. • Default = 2 for DVDT = 4.
MU = x	Coefficient for trapezoidal integration. Range is 0.0 to 0.5. Default=0.5.
RELH = x	Sets relative current tolerance, through voltage-defined branches (voltage sources and inductors). Use RELH to check current convergence, but only if the value of the ABSH control option is greater than zero. Default = 0.05.
RELI = x	Sets the relative error/tolerance change, from iteration to iteration. This value determines convergence for all currents, in diode, BJT, and JFET devices. (RELMOS sets the tolerance for MOSFETs). This is the percent change in current, from the value calculated at the previous timepoint. • Default = 0.01 for KCLTEST = 0. • Default = 1e-6 for KCLTEST = 1.
RELQ = x	Used in the timestep algorithm for local truncation error (LVLTIM = 2). RELQ changes the size of the timestep. If the capacitor charge calculation (in the present iteration) exceeds that of the past iteration, by a percentage greater than the value of RELQ, then HSPICE reduces the internal timestep (Delta). Default = 0.01.
RELTOL, RELV	Sets the relative error tolerance, for voltages. Use RELV, with the ABSV control option, to determine voltage convergence. Increasing RELV increases the relative error. RELV is the same as RELTOL. The RELI and RELVDC options default to the RELTOL value. Default = 1e-3.
RELVAR = x	Use this option with ABSVAR and the DVDT timestep algorithm, to set the relative voltage change for LVLTIM = 1 or 3. If the node voltage at the current time point exceeds the node voltage at the previous time point by RELVAR, then HSPICE reduces the timestep, and calculates a new solution at a new time point. Default = 0.30 (30%).

Table 10-7 Tolerance Options (Sheet 4 of 4)

Parameter	Description
SLOPETOL = x	Minimum value for breakpoint table entries in a piecewise linear (PWL) analysis. If the difference in the slopes of two consecutive PWL segments is less than the SLOPETOL value, HSPICE ignores the breakpoint, for the point between the segments. Default=0.5.
TIMERES = x	Minimum separation between breakpoint values, for the breakpoint table. If two breakpoints are closer together (in time) than the TIMERES value, HSPICE enters only one of them in the breakpoint table. Default = 1 ps.
TRTOL = x	Used in the timestep algorithm for local truncation error (LVLTIM = 2). After this algorithm generates TRTOL, HSPICE multiplies the internal timestep by TRTOL. Although TRTOL reduces simulation time, it also maintains accuracy. This factor estimates the amount of error introduced, if you truncate Taylor series expansion, which the algorithm uses. This error reflects the minimum timestep required, to reduce simulation time and maintain accuracy. The range of TRTOL is 0.01 to 100; typical values range from 1 to 10. If you set TRTOL to 1 (the minimum value), HSPICE uses a very small timestep. As you increase the TRTOL setting, the timestep size increases. Default = 7.0.
VNTOL = x,	Absolute minimum voltage, for DC and transient analysis. ABSV is the same as VNTOL. Decrease VNTOL, if accuracy is more critical than convergence. If you need voltages less than 50 microvolts, reduce VNTOL to two orders of magnitude less than the smallest desired voltage. This ensures at least two significant digits. Typically, you do not need to change VNTOL, unless you are simulating a high-voltage circuit. For 1000-volt circuits, a reasonable value can be 5 to 50 millivolts. Default = 50 (microvolts).

Table 10-8 Limit Options (Sheet 1 of 3)

Parameter	Description
AUTOSTOP	Stops the transient analysis, after calculating all TRIG-TARG and FIND-WHEN measure functions. This option can substantially reduce CPU time. If the data file contains measure functions (such as AVG, RMS, MIN, MAX, PP, ERR, ERR1,2,3, and PARAM), then HSPICE disables AUTOSTOP. If on, .TRAN simulation stops when it finds all .MEASURE results. Can be 0 (off, the default) or 1 (on).
BKPSIZ = x	Size of the breakpoint table. Default = 5000.

Table 10-8 Limit Options (Sheet 2 of 3)

DELMAX = x	Maximum Delta of the internal timestep.HSPICE automatically sets the DELMAX value, based on factors listed in Timestep Control for Accuracy on page 10-25. The initial DELMAX value, in the output listing, is generally not the value used for simulation
DVTR	Limits voltage in transient analysis. Default = 1000.
FS = x	Decreases Delta (internal timestep) by the specified fraction of a timestep (TSTEP), for the first time point of a transient. Decrease the FS value to help circuits that have timestep convergence difficulties. DVDT = 3 uses FS to control the timestep. $\mathrm{Delta} = \mathrm{FS} \times [\mathrm{MIN}(\mathrm{TSTEP}, \mathrm{DELMAX}, \mathrm{BKPT})]$ You specify DELMAX. BKPT is related to the breakpoint of the source. The .TRAN statement sets TSTEP. Default = 0.25.
FT = x	Decreases Delta (the internal timestep), by a specified fraction of a timestep (TSTEP), for an iteration set that does not converge. If DVDT = 2 or DVDT = 4, FT controls the timestep. Default = 0.25.
GMIN = x	Minimum conductance added to all PN junctions for a time sweep in transient analysis. Default = 1e-12.

IMIN = x,
ITL3 = x
	Minimum timestep, in timestep algorithms for transient analysis. IMIN is the minimum number of iterations required, to obtain convergence. If the number of iterations is less than IMIN, the internal timestep (Delta) doubles. This option decreases simulation times, in circuits where nodes are stable most of the time (such as digital circuits). If the number of iterations is greater than IMIN, the timestep stays the same, unless the number of iterations exceeds IMAX (see IMAX). ITL3 is the same as IMIN. Default = 3.0.

IMAX = x,
ITL4 = x
	Maximum timestep, in timestep algorithms in transient analysis. IMAX sets the maximum iterations to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep (Delta) decreases by a factor equal to the FT transient control option. HSPICE uses the new timestep to calculate a new solution. IMAX also works with the IMIN transient control option. ITL4 is the same as IMAX. Default = 8.0.

ITL5 = x	Iteration limit for transient analysis. If a circuit uses more than ITL5 iterations, the program prints all results, up to that point. The default (0.0) allows an infinite number of iterations.

Table 10-8 Limit Options (Sheet 3 of 3)

RMAX = x	Sets the TSTEP multiplier, which controls the maximum value (DELMAX) for the Delta of the internal timestep: DELMAX = TSTEPxRMAX • Default = 5, if dvdt = 4 and lvltim = 1. • Otherwise, the default = 2. The maximum value is 1e+9, the minimum value is 1e-9. The recommended maximum value is 1e+5.
RMIN = x	Sets the minimum value of Delta (internal timestep). An internal timestep smaller than RMINxTSTEP, terminates the transient analysis, and reports an *internal timestep too small* error. If the circuit does not converge in IMAX iterations, Delta decreases by the amount you set in the FT option. Default = 1.0e-9.
VFLOOR = x	Minimum voltage to print in output listing. All voltages lower than VFLOOR, print as 0. Affects only the output listing: VNTOL (ABSV) sets minimum voltage to use in a simulation.

Matrix Manipulation Options

After HSPICE generates individual linear elements in an input netlist, it constructs the linear equations for the matrix. You can set variables that affect how HSPICE constructs and solves the matrix equation, including the PIVOT and GMIN options. GMIN places a variable into the matrix, to prevent the matrix becoming *ill-conditioned*.

The PIVOT option 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 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.

For descriptions of the FAST and AUTOSTOP options, see Transient Control Options on page 10-15.

Simulation Accuracy

In HSPICE, 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 = MBYPASSxVNTOL = 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 10-31.

The DELMAX control option also affects simulation accuracy. DELMAX specifies the maximum timestep size. If you do not set DELMAX in an .OPTION statement, HSPICE computes a DELMAX value. Factors that determine the computed DELMAX value are:

- .OPTION RMAX and 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.

Use the FS and RMAX control options, to control the DELMAX value.

- The FS option, which defaults to 0.25, scales the breakpoint interval in the DELMAX calculation.
- The RMAX option 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 an .OPTION statement, to set DELMAX to one-hundredth of the period or less.

The ACCURATE control option tightens the simulation options, to output the most accurate set of simulation algorithms and tolerances. If you set ACCURATE to 1, HSPICE uses these control options:

DVDT = 2
BYTOL = 0

RELVAR = 0.2
LVLTIM = 3

BYPASS = 0
ABSVAR = 0.2

FT = FS = 0.2
RMAX = 2

RELMOS = 0.01
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 defaults for source and drain junction parasitics. 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, NRS, to override the default calculations.
- CAPOP = 4 models the most advanced charge conservation, non-reciprocal gate capacitances. HSPICE 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 the ACCURATE option 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

SYNTAX:

Gear algorithm:

```
.OPTION METHOD = GEAR
```

Backward-Euler:

```
.OPTION METHOD = GEAR MU = 0
```

Trapezoidal algorithm (default):

```
.OPTION METHOD = TRAP
```

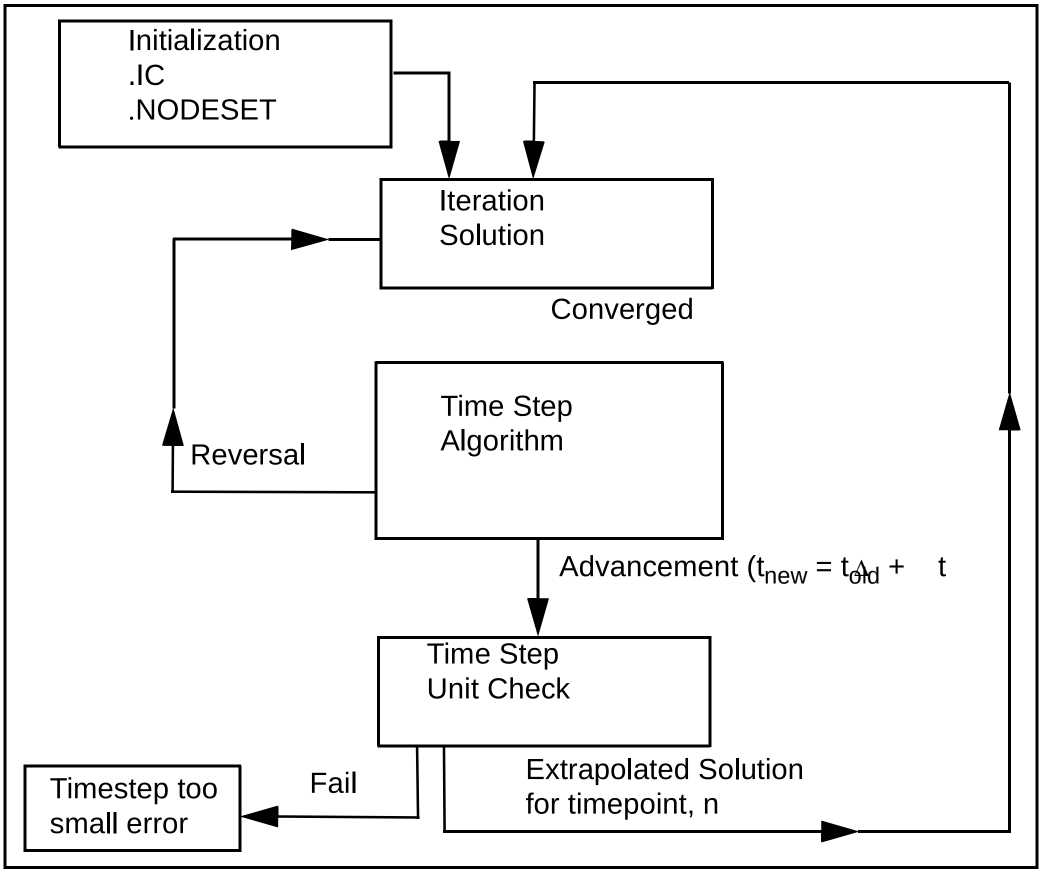
Each algorithm has advantages and disadvantages. Ideally, the trapezoidal is the preferred algorithm overall, because of its highest accuracy level and lowest simulation time.

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 10-5 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 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 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.

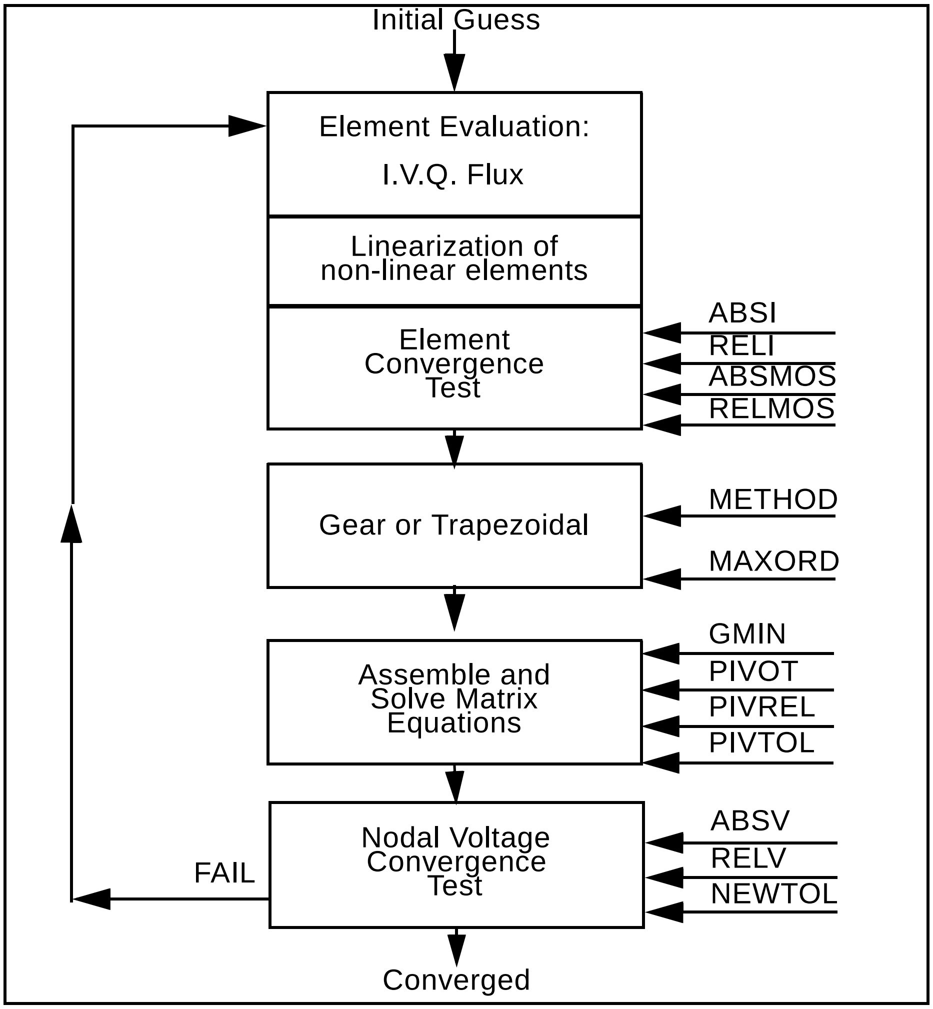
To manually improve on autoconvergence results, or if autoconvergence fails to converge, you can do either of the following:

- Set 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 METHOD=TRAP in the netlist, then manually adjust tstep and the relevant control options (such as CSHUNT or GSHUNT).

Figure 10-6 Iteration Algorithm



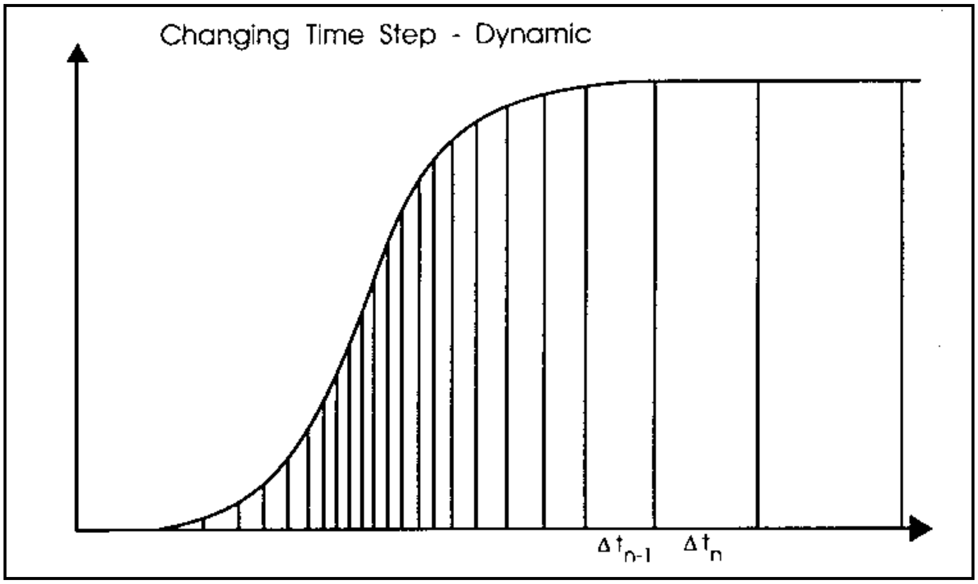
Selecting Timestep Control Algorithms

In HSPICE, you can select one of three dynamic timestep-control algorithms:

- Iteration Count Dynamic Timestep Algorithm .
- Local Truncation Error (LTE) Dynamic Timestep .
- DVDT Dynamic Timestep Algorithm .

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 10-7 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 Algorithm on page 10-33.
- 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 starts the autoconvergence process, it sets LVLTIM = 2.

Iteration Count Dynamic Timestep Algorithm

The simplest dynamic timestep algorithm is the iteration count algorithm. The following options control this algorithm:

Table 10-9 Dynamic Timestep Options

Option	Description
IMAX	Controls the internal timestep size, based on the number of iterations required for a timepoint solution. If the number of iterations per timepoint exceeds the IMAX value, the internal timestep decreases. Default = 8.
IMIN	Controls the internal timestep size, based on the number of iterations required for the previous timepoint solution. If the last timepoint solution completes in fewer than IMIN iterations, the internal timestep increases. Default = 3.

Local Truncation Error (LTE) Dynamic Timestep

The local truncation error timestep method uses a Taylor-series approximation, to calculate the next timestep for a transient analysis. This method uses the allowed local truncation error, to calculate an internal timestep. If the calculated timestep is smaller than the current timestep, HSPICE 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 does not reverse the timestep. The next timepoint uses a new timestep.

To select the timestep algorithm for local truncation error, 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 Algorithm

To select this algorithm, set the LVLTIM option to 1 or 3.

- If you set LVLTIM = 1, the DVDT algorithm does not use timestep reversal. HSPICE 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 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 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 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 the RELVAR and ABSVAR options 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 with HSPICE Release 95.3, 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

The RMIN, RMAX, FS, FT, and DELMAX control options define the minimum and maximum internal timestep, for the DVDT algorithm. If the timestep is below the minimum, program execution stops.

EXAMPLE:

If the timestep becomes less than the minimum internal timestep (defined as TSTEPxRMIN), HSPICE reports an *internal timestep too small* error.

Note: RMIN is the minimum timestep coefficient. Default is 1e-9. TSTEP is the time increment, as set in the .TRAN statement.

If you set DELMAX in an .OPTION statement, HSPICE uses DVDT = 0. If you do not specify DELMAX in an .OPTION statement, then HSPICE computes a DELMAX value. For DVDT = 0, 1, or 2, the maximum internal timestep is:

```
   min[(TSTOP/50), DELMAX, (TSTEPxRMAX)]
```

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/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),(TSTEPxRMAX)]
```

Set TRCON=-1 to disable autospeedup. You can then adjust TSTEP and RMAX, to balance accuracy and speed.

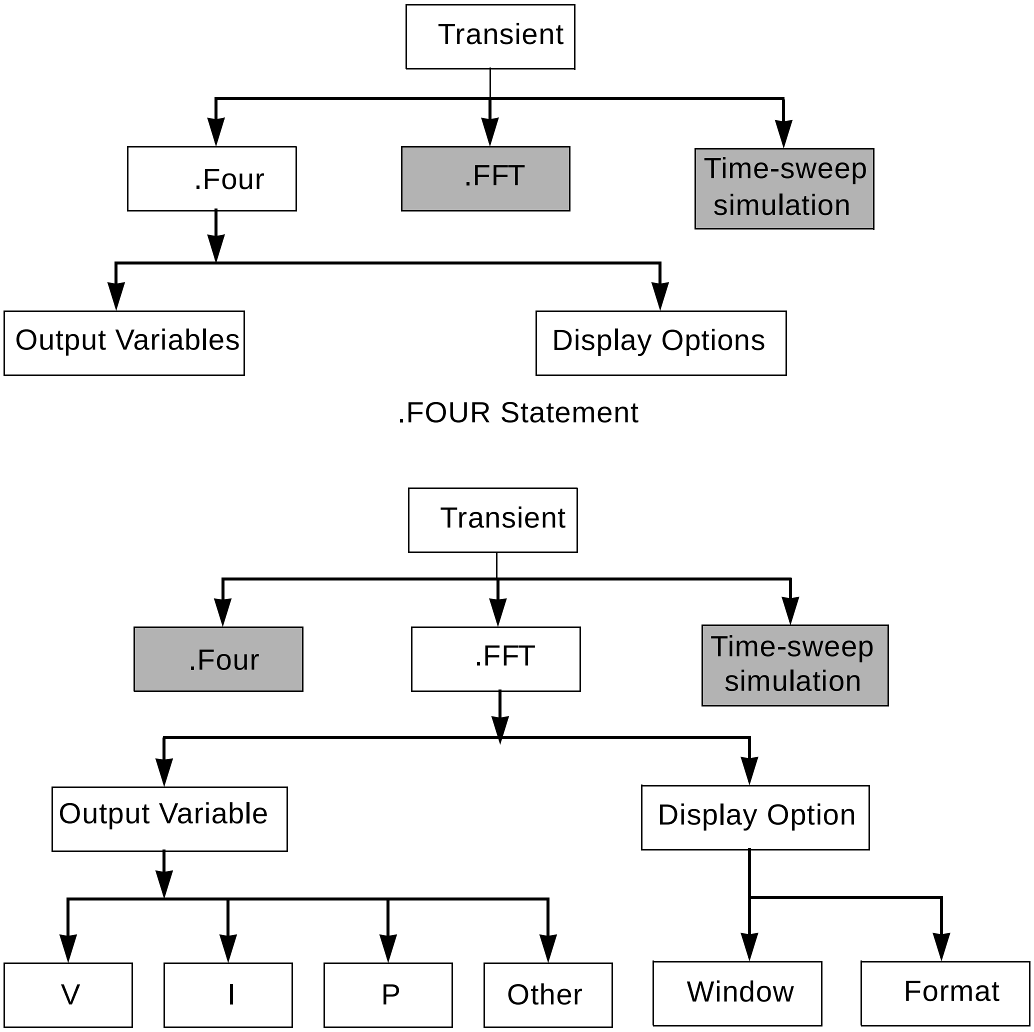
In circuits with piecewise linear (PWL) transient sources, the SLOPETOL option also affects the internal timestep. A PWL source, with a large number of voltage or current segments, contributes a correspondingly-large number of entries to the internal breakpoint table. The number of breakpoint table entries 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.

Fourier Analysis

This section describes the Fourier and FFT Analysis flow.

Figure 10-8 Fourier and FFT Analysis



.FOUR Statement

.FFT Statement

HSPICE provides two different Fourier analyses:

- .FOUR is the same as is available in SPICE 2G6: a standard, fixed-window analysis tool.
- .FFT is a much more flexible Fourier analysis tool. Use it for analysis tasks that require more detail and precision.

.FOUR Statement

This statement performs a Fourier analysis, as part of the transient analysis. You can use the .FOUR statement in HSPICE perform the Fourier analysis over the interval (tstop-fperiod, tstop), where:

- tstop is the final time, specified for the transient analysis (see Using the .TRAN Statement on page 10-4).
- fperiod is a fundamental frequency period (*freq* parameter).

HSPICE performs Fourier analysis on 501 points of transient analysis data on the last 1/f time period, where f is the fundamental Fourier frequency. HSPICE interpolates transient data, to fit on 501 points, running from (tstop-1/f) to tstop.

To calculate the phase, the normalized component, and the Fourier component, HSPICE uses 10 frequency bins. The Fourier analysis determines the DC component, and the first nine AC components. For improved accuracy, the .FOUR statement can use non-linear, instead of linear, interpolation.

SYNTAX:

```
.FOUR freq ov1 <ov2 ov3 ...>
```

freq	Fundamental frequency.
ov1 …	Output variables to analyze.

EXAMPLE:

```
.FOUR 100K V(5)
```

Accuracy and DELMAX

For better accuracy, set small values for the RMAX or DELMAX options. 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.

EXAMPLE:

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: (Input)

The following is input for an .OP, .TRAN, or .FOUR analysis.

```
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 = .2N CGSO = .2N
+ CGBO = 2N
.MODEL PMOS PMOS LEVEL = 3 CGDO = .2N CGSO = .2N
+CGBO = 2N
.OP
.TRAN 1N 100N
.FOUR 20MEG V(2)
.PRINT TRAN V(2) V(1)
.END
```

EXAMPLE: (Output)

```
******
cmos inverter
**** fourier analysistnom = 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
```

For information about Fourier analysis, see Fourier Analysis on page 10-37.

.FFT Statement

For information about the .FFT statement, see “FFT Spectrum Analysis” in the *HSPICE Applications Manual*.

11

AC Sweep and Small Signal Analysis

This chapter describes how to perform an AC sweep, and a small signal analysis. It explains the following topics:

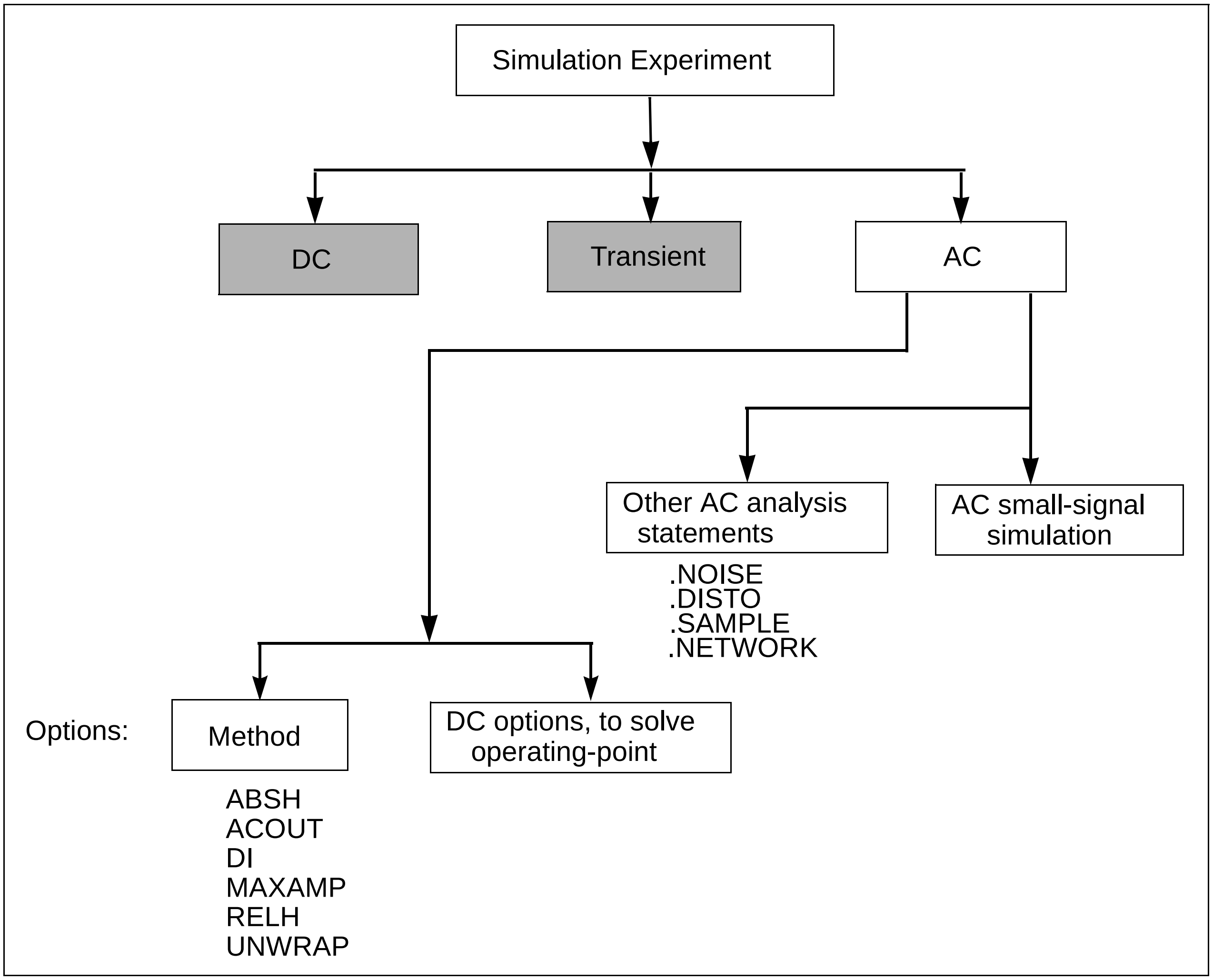
- AC Small Signal Analysis
- .AC Statement
- AC Control Options
- AC Analysis of an RC Network
- Other AC Analysis Statements

Also see AC Analysis Output Variables on page 7-31.

AC Small Signal Analysis

AC small signal analysis in HSPICE computes AC output variables as a function of frequency (see Figure 11-1). HSPICE 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 11-1 AC Small Signal Analysis Flow



In HSPICE, the output of AC Analysis includes voltages and currents.

HSPICE converts capacitor and inductor values to their corresponding admittances:

$$YC = j\omega C \quad \text{for capacitors}$$

$$YL = 1/j\omega L \quad \text{for inductors}$$

Resistors can have different DC and AC values. If you specify AC = <*value*> in a resistor statement, HSPICE 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 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 Chapter 4, “Using BJT Models”, in the *HSPICE Elements and Device Models Manual*. MOSFET AC-equivalent circuit models are described in Chapter 8, “Introducing MOSFETs”, in the *HSPICE Elements and Device Models Manual*.

The AC analysis statement can sweep values for:

- Frequency.
- Element.
- Temperature.
- Model parameter.
- Randomized (Monte Carlo) distribution.
- Optimization and AC analysis.

Additionally, as part of the small-signal analysis tools, HSPICE provides:

- Noise analysis.
- Distortion analysis.
- Network analysis.
- Sampling noise.

.AC Statement

You can use the .AC statement in several different formats, depending on the application, as shown in the examples below. You can also use the .AC statement to perform data-driven analysis in HSPICE.

SYNTAX:

Single/Double Sweep

```
.AC type np fstart fstop
```

```
.AC type np fstart fstop <SWEEP var <START=>start
+ <STOP=>stop <STEP=>incr>
```

```
.AC type np fstart fstop <SWEEP var type np start stop>
```

```
.AC type np fstart fstop <SWEEP var START="param_expr1"
+ STOP="param_expr2" STEP="param_expr3">
```

```
.AC type np fstart fstop <SWEEP var start_expr
+ stop_expr step_expr>
```

Sweep Using Parameters

SYNTAX:

```
.AC type np fstart fstop <SWEEP DATA = datanm>
```

```
.AC DATA = datanm
```

```
.AC DATA = datanm <SWEEP var <START=>start <STOP=>stop
+ <STEP=>incr>
```

```
.AC DATA = datanm <SWEEP var type np start stop>
```

```
.AC DATA = datanm <SWEEP var START="param_expr1"
+ STOP="param_expr2" STEP="param_expr3">
```

```
.AC DATA = datanm <SWEEP var start_expr stop_expr
+ step_expr>
```

Optimization

SYNTAX:

```
.AC DATA = datanm OPTIMIZE = opt_par_fun
+ RESULTS = measnames MODEL = optmod
```

See also Analysis Statement (.DC, .TRAN, .AC) on page 12-41.

Random/Monte Carlo

You can use the following syntax in HSPICE:

```
.AC type np fstart fstop <SWEEP MONTE = val>
```

The .AC statement keywords and parameters are:

Table 11-1 .AC Monte Carlo Syntax

Parameter	Description
DATA = datanm	Data name, referenced in the .AC statement
incr	Increment value of the voltage, current, element, or model parameter. If you use *type* variation, specify the *np* (number of points) instead of *incr*.
fstart	Starting frequency. If you use POI (list of points) type variation, use a list of frequency values, not fstart fstop.
fstop	Final frequency.
MONTE = val	Produces a number (*val*) of randomly-generated values. HSPICE uses these values to select parameters from a distribution, either *Gaussian*, *Uniform*, or *Random Limit* (see Monte Carlo Analysis on page 12-13).
np	Number of points, or points per decade or octave, depending on which keyword precedes it.
start	Starting voltage or current, or any parameter value for an element or model.
stop	Final voltage or current, or any parameter value for an element or a model.
SWEEP	This keyword indicates that the .AC statement specifies a second sweep.
TEMP	This keyword indicates a temperature sweep
type	Can be any of the following keywords: • DEC – decade variation. • OCT – octave variation. • LIN – linear variation. • POI – list of points.
var	Name of an independent voltage or current source, element or model parameter, or the TEMP (temperature sweep) keyword. HSPICE supports source value sweep, referring to the source name (SPICE style). If you select a parameter sweep, a .DATA statement, and a temperature sweep, then you must choose a parameter name for the source value. You must also later refer to it in the .AC statement. The parameter name cannot start with V or I.

EXAMPLE 1:

The following example performs a frequency sweep, by 10 points per decade, from 1 kHz to 100 MHz.

```
.AC DEC 10 1K 100MEG
```

EXAMPLE 2:

The next line runs a 100-point frequency sweep from 1 Hz to 100 Hz.

```
.AC LIN 100 1 100HZ
```

EXAMPLE 3:

The following example performs an AC analysis, for each value of cload. This results from a linear sweep of cload between 1 pF and 10 pF (20 points), sweeping the frequency by 10 points per decade, from 1 Hz to 10 kHz.

```
.AC DEC 10 1 10K SWEEP cload LIN 20 1pf 10pf
```

EXAMPLE 4:

The following example performs an AC analysis, for each value of rx, 5 k and 15 k, sweeping the frequency by 10 points per decade, from 1 Hz to 10 kHz.

```
.AC DEC 10 1 10K SWEEP rx POI 2 5k 15k
```

EXAMPLE 5:

The next example uses the .DATA statement to perform a series of AC analyses, modifying more than one parameter. The datanm file contains the parameters.

```
.AC DEC 10 1 10K SWEEP DATA = datanm
```

EXAMPLE 6:

The following example illustrates a frequency sweep, and a Monte Carlo analysis, with 30 trials.

```
.AC DEC 10 1 10K SWEEP MONTE = 30
```

EXAMPLE 7:

If the input file includes an .AC statement, HSPICE runs AC analysis for the circuit, over a selected frequency range, for each parameter in the second sweep.

For AC analysis, the data file must include at least one independent AC source element statement (for example, VI INPUT GND AC 1V). HSPICE checks for this condition, and reports a fatal error if you did not specify such AC sources (see Sources and Stimuli on page 5-1).

AC Control Options

Table 11-2 AC Control Options

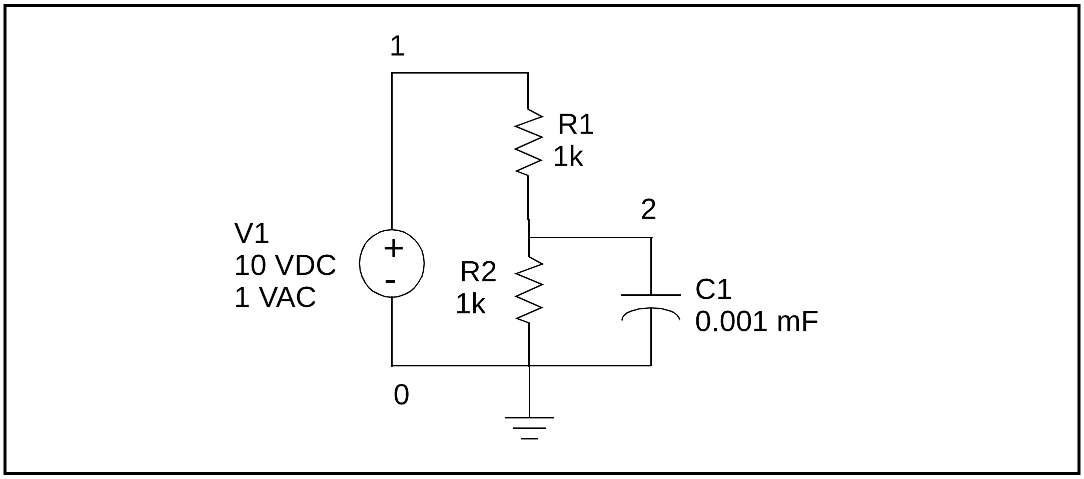
Parameter	Description
ABSH = x	Absolute current change, through voltage-defined branches (voltage sources and inductors). Use ABSH with DI and RELH, to check for current convergence. Default is 0.0.
ACOUT	AC output calculation method, for the difference in values of magnitude, phase, and decibels. Use this option for prints and plots. Default = 1. Default value, ACOUT = 1, selects HSPICE method, which calculates the difference of the magnitudes of the values. SPICE method, ACOUT = 0, calculates magnitude of differences.
DI = x	Maximum iteration-to-iteration current change, through voltage-defined branches (voltage sources and inductors). Use this option only if the ABSH control value is greater than 0. Default = 0.0.
MAXAMP = x	Maximum current, through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, HSPICE reports an error. Default = 0.0.
RELH = x	Relative tolerance for currents, through voltage-defined branches (voltage sources and inductors). Use RELH to check current convergence, but only if the value of the ABSH control option is greater than zero. Default = 0.05.
UNWRAP	Displays phase results for AC analysis, in unwrapped form (with a continuous phase plot).HSPICE uses these results to accurately calculate group delay. It also uses unwrapped phase results to compute group delay, even if you do not set UNWRAP.

AC Analysis of an RC Network

Figure 11-2 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 11-2   RC Network Circuit



The input netlist for the RC network circuit is:

```
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
   ```

   When the run finishes, HSPICE displays:

   ```
   >info:     ***** hspice job concluded
   ```

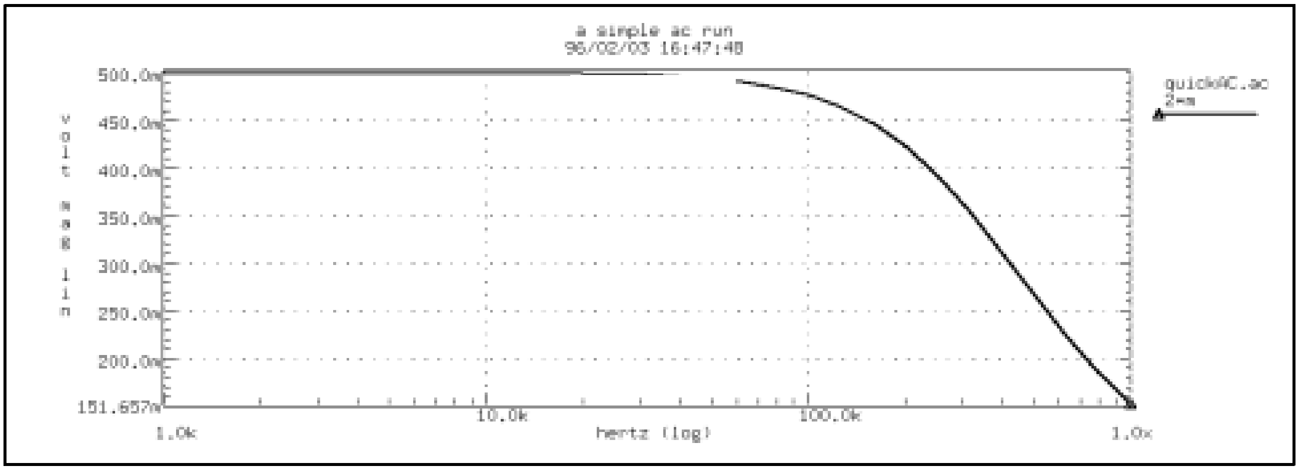
   This is followed by a line that shows the amount of real time, user time, and system time needed for the analysis.

   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, using a log scale on the x-axis.

Figure 11-3 shows the waveform that HSPICE produces if you sweep the response of node 2, as you vary the frequency of the input from 1 kHz to 1 MHz.

Figure 11-3 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.

Other AC Analysis Statements

This section describes how to use other AC analysis statements.

.DISTO — AC Small-Signal Distortion Analysis

The .DISTO statement computes the distortion characteristics of the circuit in an AC small-signal, sinusoidal, steady-state analysis.

The program computes and reports five distortion measures at the specified load resistor. The analysis assumes that the input uses one or two signal frequencies.

- HSPICE uses the first frequency (F1, the nominal analysis frequency) to calculate harmonic distortion. The .AC statement frequency-sweep sets it.
- HSPICE uses the optional second input frequency (F2) to calculate intermodulation distortion. To set it implicitly, specify the skw2 parameter, which is the F2/F1 ratio.

Table 11-3 .DISTO Syntax

Parameter	**Description**
DIM2	Intermodulation distortion, first difference. Relative magnitude and phase of the frequency component (F1 - F2).
DIM3	Intermodulation distortion, second difference. The relative magnitude and phase of the frequency component (2 · F1 - F2).
HD2	Second-order harmonic distortion. Relative magnitude and phase of the frequency component 2 · F1 (ignores F2).
HD3	Third-order harmonic distortion. Relative magnitude and phase of the frequency component 3 · F1 (ignores F2).
SIM2	Intermodulation distortion, sum. Relative magnitude and phase of the frequency component (F1 + F2).

The .DISTO summary report includes:

- A set of distortion measures, for each component in each element.
- A summary of distortion measures for each element.
- A summary of distortion measures for the entire circuit.

SYNTAX:

```
.DISTO Rload <inter <skw2 <refpwr <spwf>>>>
```

Table 11-4 .DISTO Summary Syntax

Parameter	**Description**
Rload	The resistor element name of the output load resistor, into which the output power feeds.
refpwr	Reference power level, used to compute the distortion products. If you omit *refpwr*, the default value is 1mW, measured in decibels magnitude (dbM). The value must be ≥ *1e-10*.
skw2	Ratio of the second frequency (F2) to the nominal analysis frequency (F1), in the range *1e-3* < *skw2* < *0.999*. If you omit *skw2*, the default value is 0.9.
spwf	Amplitude of the second frequency (F2). The value must be ≥ *1e-3*. Default = 1.0.
inter	Interval at which HSPICE prints a distortion-measure summary. Specifies a number of frequency points in the AC sweep (see the *np* parameter, in .AC Statement on page 11-4). • If you omit *inter*, or set it to zero, HSPICE does not print a summary. To print or plot the distortion measures, use the .PRINT or .PLOT statement. • If you set inter to 1 or higher, HSPICE prints a summary of the first frequency, and of each subsequent inter-frequency increment. To obtain a summary printout for only the first and last frequencies, set inter equal to the total number of increments needed, to reach fstop in the .AC statement. For a summary printout of only the first frequency, set inter to greater than the total number of increments required, to reach *fstop*.

EXAMPLE:

```
.DISTO RL 2 0.95 1.0E-3 0.75
```

HSPICE performs only one distortion analysis per simulation. If your design contains more than one .DISTO statement, HSPICE runs only the last statement. The .DISTO statement calculates distortions for diodes, BJTs (levels 1, 2, 3, and 4), and MOSFETs (Level49 and Level53, Version 3.22).

Note: HSPICE prints an extensive summary from the distortion analysis, for each frequency listed. Use the *inter* parameter in the .DISTO statement to limit the amount of output generated.

.NOISE Statement — AC Noise Analysis

SYNTAX:

```
.NOISE ovv srcnam inter
```

Table 11-5 .NOISE Syntax

Parameter	**Description**
ovv	Nodal voltage output variable. Defines the node at which HSPICE sums the noise.
srcnam	Name of the independent voltage or current source, to use as the noise input reference
inter	Interval at which HSPICE prints a noise analysis summary. *inter* specifies how many frequency points to summarize in the AC sweep. If you omit *inter*, or set it to zero, HSPICE does not print a summary. If inter is equal to or greater than one, HSPICE prints summary for the first frequency, and once for each subsequent increment of the inter frequency. The noise report is sorted according to the contribution of each node to the overall noise level.

EXAMPLE:

```
.NOISE V(5) VIN 10
```

Use the .NOISE and .AC statements, to control the noise analysis of the circuit.

Noise Calculations

Noise calculations in HSPICE 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 simulator calculates each noise source independently. The total output noise voltage is the RMS sum of the individual noise contributions:

$$\text{onoise} = \sum_{n=1}^{n} \left| Z_n \cdot I_n \right|^2$$

Table 11-6 Noise Calculations

Parameter	**Description**
onoise	Total output noise.
I	Equivalent current due to thermal, shot, or flicker noise.
Z	Equivalent transimpedance, between noise source and output.
n	Number of noise sources, associated with all resistors, MOSFETs, diodes, JFETs, and BJTs.

The input noise (*inoise*) voltage is the total output noise, divided by the gain or transfer function of the circuit. HSPICE 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 runs only the last statement.

.SAMPLE Statement — Noise Folding Analysis

To acquire data from analog signals, use the .SAMPLE statement, with the .NOISE and .AC statements, to analyze data sampling noise in HSPICE. The SAMPLE analysis performs a noise-folding analysis, at the output node.

SYNTAX:

```
.SAMPLE FS = freq <TOL = val> <NUMF = val>
+ <MAXFLD = val> <BETA = val>
```

Table 11-7 .SAMPLE Syntax

Parameter	**Description**
FS = freq	Sample frequency, in Hertz.
TOL	Sampling-error tolerance: the ratio of the noise power (in the highest folding interval) to the noise power (in baseband). Default = `1.0e-3`.
NUMF	Maximum number of frequencies that you can specify. The algorithm requires about ten times this number of internally-generated frequencies, so keep this value small. Default = `100`.

Table 11-7 .SAMPLE Syntax

MAXFLD	Maximum number of folding intervals (default = `10.0`). The highest frequency (in Hertz) that you can specify is: FMAX = MAXFLD · FS

BETA	Optional noise integrator (duty cycle), at the sampling node:
	BETA = 0 no integrator
	BETA = 1 simple integrator (default)
	If you clock the integrator (integrates during a fraction of the 1/FS sampling interval), then set BETA to the duty cycle of the integrator.

.NET Statement - AC Network Analysis

You can use the .NET statement to compute parameters for:

- Z impedance matrix.
- Y admittance matrix.
- H hybrid matrix
- S scattering matrix.

HSPICE also computes:

- Input impedance.
- Output impedance.
- Admittance.

This analysis is part of AC small-signal analysis. To run network analysis, specify the frequency sweep for the AC statement.

SYNTAX:

<u>One-Port Network</u>

```
.NET input <RIN = val>
.NET input <val>
```

<u>Two-Port Network</u>

```
.NET output input <ROUT = val> <RIN = val>
```

Table 11-8 .NET Syntax

Parameter	Description
input	Name of the voltage or current source for AC input.
output	Output port. It can be: • An output voltage, V(n1,n2). • An output current, I(source), or I(element).
RIN	Keyword, for input or source resistance. RIN calculates output impedance, output admittance, and scattering parameters. The default RIN value is 1 ohm.
ROUT	Keyword, for output or load resistance. ROUT calculates input impedance, admittance, and scattering parameters. Default=1 ohm.

EXAMPLE:

One-Port Network

```
.NET    VINAC     RIN = 50
.NET    IIN       RIN = 50
```

Two-Port Network

```
.NET V(10,30)   VINAC     ROUT = 75RIN = 50
.NET I(RX)      VINAC     ROUT = 75RIN = 50
```

AC Network Analysis - Output Specification

`Xij(z), ZIN(z), ZOUT(z), YIN(z), YOUT(z)`

Table 11-9 AC Network Analysis Output

Parameter	Description
X	In HSPICE, can be Z (impedance), Y (admittance), H (hybrid), or S (scattering).
ij	i and j identify the matrix parameter to print in HSPICE. 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.
ZOUT	Output impedance.

Table 11-9 AC Network Analysis Output (Continued)

Parameter	Description
z	Output type (HSPICE): • R: real part. • I: imaginary part. • M: magnitude. • P: phase. • DB: decibel. • T: group time delay.
YIN	Input admittance. For a one-port network, YIN is the same as Y11.
YOUT	Output admittance.

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:[1] Network Analysis Results

```
*FILE: FBP_1.SP
.OPTION 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 41 200MEG 300MEG
.NET V(OUT) VIN ROUT = 50 RIN = 50
.PLOT AC S11(DB) (-50,10) S11(P) (-180,180)
.PLOT AC ZIN(M) (5,130) ZIN(P) (-90,90)
.END
```

Figure 11-4   S11 Magnitude and Phase Plots

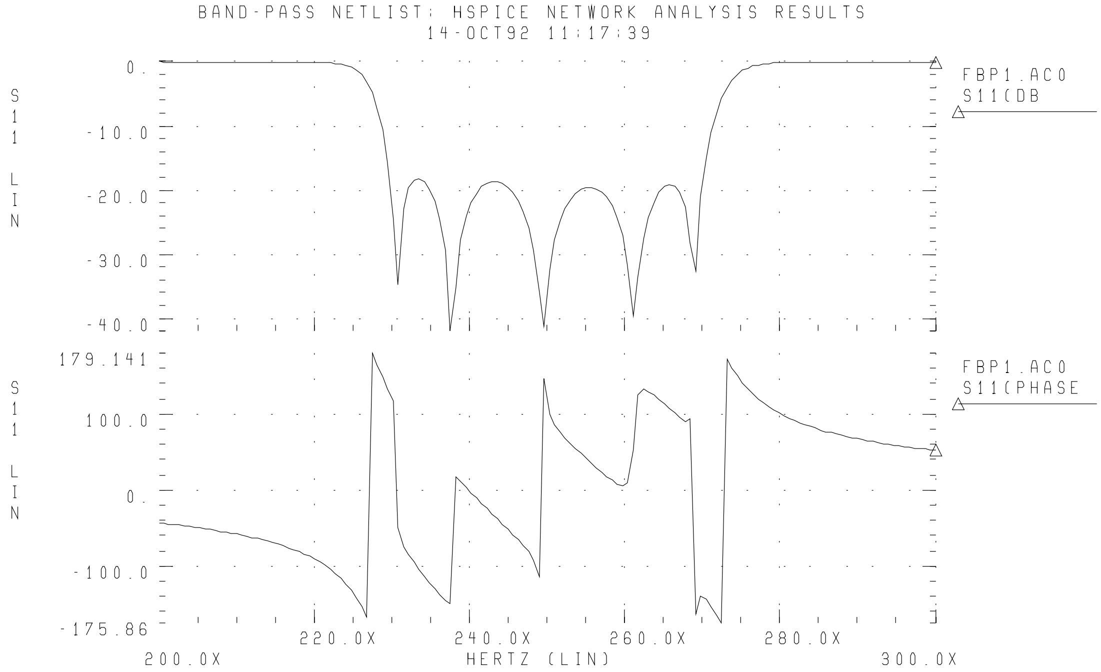
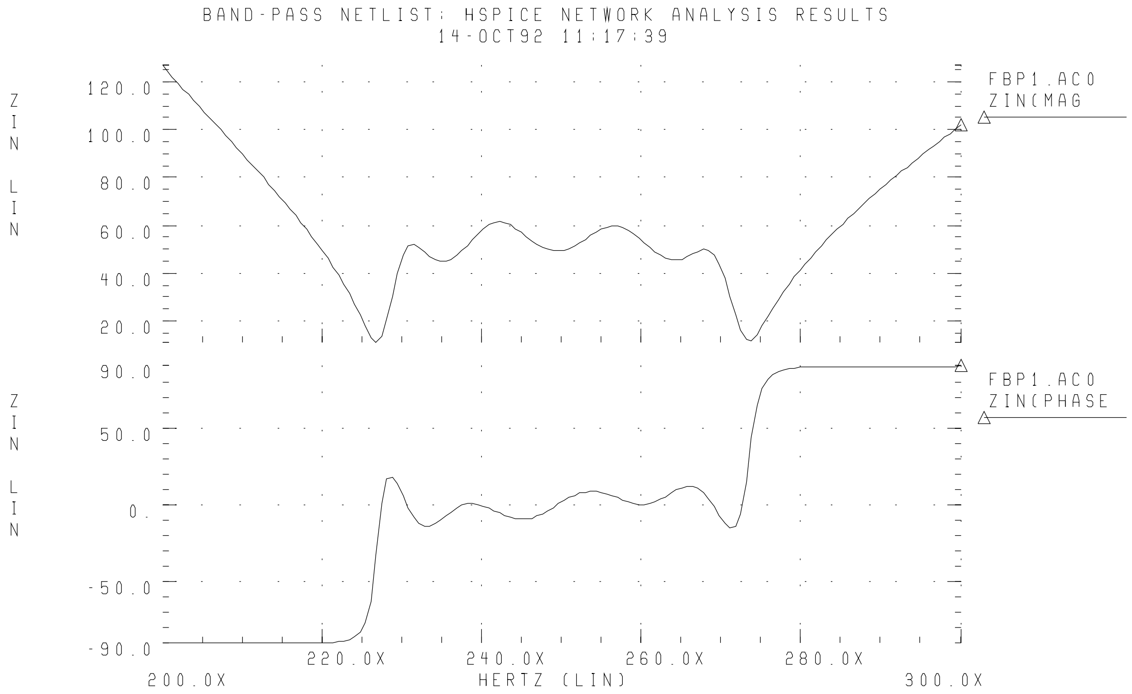


Figure 11-5 ZIN Magnitude and Phase Plots



NETWORK Variable Specification

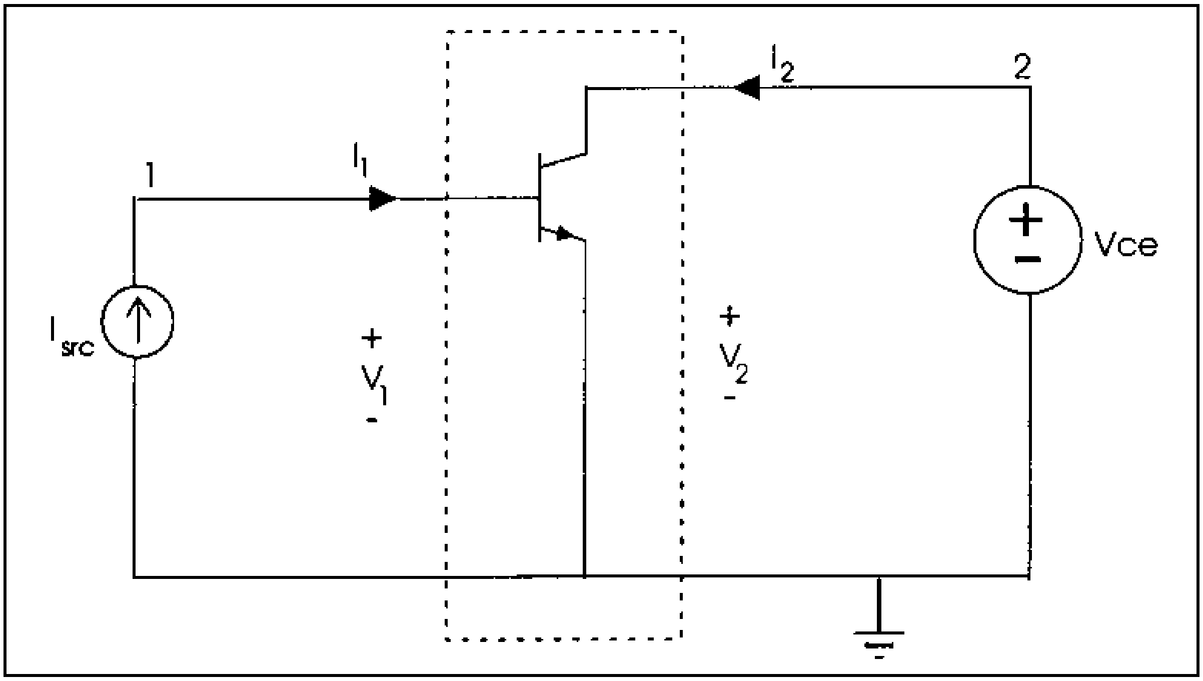
HSPICE uses the AC analysis results to perform network analysis. The .NET statement defines Z, Y, H, and S parameters to calculate. The following list shows various combinations of the .NET statement, for network matrices that HSPICE calculates:

`.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 i evaluates the Z matrix parameters. It then uses standard conversion equations, to determine S parameters or any other requested parameters.

Figure 11-6 shows the importance of variables in the .NET statement. Here, *Isrc* and *Vce* are the DC biases, applied to the BJT.

Figure 11-6 Parameters with .NET V(2) Isrc



This .NET statement provides an incorrect result for the Z parameter calculation:

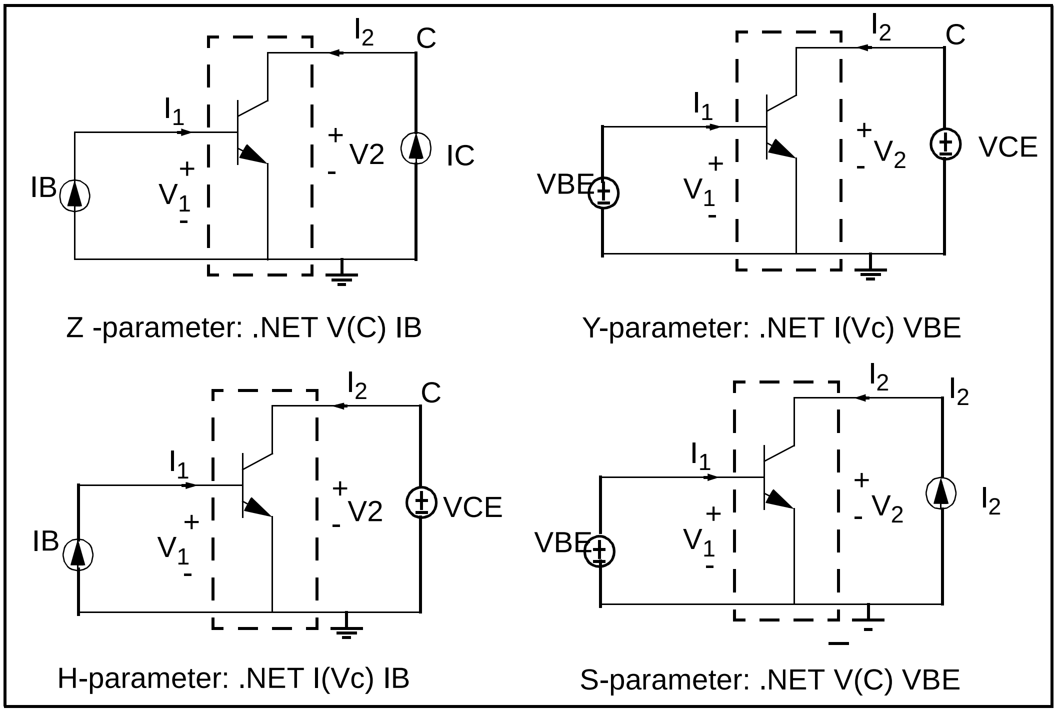
```
.NET V(2) Isrc
```

When HSPICE 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 attempts to calculate the Z parameters (Z11 and Z21), defined as Z11 = V1/I1 and Z21 = V2/I1 with I2=0. The above example does not satisfy the requirement that I2 must be zero. Instead, V2 is zero, which results in incorrect values for Z11 and Z21.

Figure 11-7 shows the correct biasing configurations, for performing network analysis for the Z, Y, H, and S parameters.

Figure 11-7 Network Parameter Configurations



Z -parameter: .NET V(C) IB

Y-parameter: .NET I(Vc) VBE

H-parameter: .NET I(Vc) IB

S-parameter: .NET V(C) VBE

EXAMPLE:

To calculate the H parameters, HSPICE uses the .NET statement.

.NET I(V_C) I_B

VC denotes the voltage at the C node, which is the collector of the BJT. With this statement, HSPICE 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.

Network Analysis Example: Bipolar Transistor

```
BJT network analysis
.option nopage list
+ newtol reli = 1e-5 absi = 1e-10 relv = 1e-5
+ relvdc = 1e-7 nomod post gmindc = 1e-12
.op
.param vbe = 0 ib = 0 ic = 0 vce = 0
  $ H-parameter
  .NET i(vc) ibb rin = 50 rout = 50
  ve      e      0               0
  ibb     0      b               dc = ’ib’ ac = 0.1
  vc      c      0               ’vce’
  q1      c      b      e 0      bjt

  .model bjt npn subs = 1
  + bf = 1.292755e+02 br = 8.379600e+00
  + is = 8.753000e-18 nf = 9.710631e-01
  + nr = 9.643484e-01 ise = 3.428000e-16
  + isc = 1.855000e-17 iss = 0.000000e+00
  + ne = 2.000000e+00 nc = 9.460594e-01
  + ns = 1.000000e+00 vaf = 4.942130e+01
  + var = 4.589800e+00 ikf = 5.763400e-03
  + ikr = 5.000000e-03 irb = 8.002451e-07
  + rc = 1.216835e+02 rb = 1.786930e+04
  + rbm = 8.123460e+01 re = 2.136400e+00
  + cje = 9.894950e-14 mje = 4.567345e-01
  + vje = 1.090217e+00 cjc = 5.248670e-14
  + mjc = 1.318637e-01 vjc = 5.184017e-01
  + xcjc = 6.720303e-01 cjs = 9.671580e-14
  + mjs = 2.395731e-01 vjs = 5.000000e-01
  + tf = 3.319200e-11 itf = 1.457110e-02
  + xtf = 2.778660e+01 vtf = 1.157900e+00
  + ptf = 6.000000e-05 xti = 4.460500e+00
  + xtb = 1.456600e+00 eg = 1.153300e+00
  + tikf1 = -5.397800e-03 tirb1 = -1.071400e-03
  + tre1 = -1.121900e-02 trb1 = 3.039900e-03
  + 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 configurations, for the network analysis, are:

$S-parameter

```
.NET v(c) vbb rin = 50 rout = 50
ve          e       0                0
vbb         b       0                dc = 'vbe' ac = 0.1
icc         0       c                'ic'
q1          c       b      e 0       bjt
```

$Z-parameter

```
.NET v(c) ibb rin = 50 rout = 50
ve           e      0               0
ibb          0      b               dc = ’ib’ ac = 0.1
icc          0      c               ’ic’
q1           c      b      e 0      bjt

$Y-parameter
.NET i(vc) vbb rin = 50 rout = 50
ve           e      0               0
vbb          b      0               ’vbe’ ac = 0.1
vc           c      0               ’vce’
q1           c      b      e 0      bjt
```

References

1. Goyal, Ravender. “S-Parameter Output From SPICE Program”, *MSN & CT,* February 1988, pp. 63 and 66.

12

Statistical Analysis and Optimization

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 uses 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:

- The .PARAM statement investigates the performance of a circuit as you change circuit parameters. See Simulation Input and Controls on page 3-1, for details about the .PARAM statement.
- Temperature Variation Analysis varies the circuit and component temperatures, and compares the circuit responses. You can study the temperature-dependent effects of the circuit, in detail.
- Monte Carlo Analysis. If you know the statistical standard deviations of component values, use this analysis to center a design. This provides maximum process yield, and determines component tolerances.
- Worst Case Corners Analysis. If you know the component value limit, use this analysis 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. Use for cell characterization, response surface, or Taguchi analysis. See “Characterizing Cells” 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 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.

These analyses can generate scatter plots, for operating point analysis. They can also generate a family of curve plots for DC, AC, and transient analysis.

Use the .MEASURE statement, with yield analyses, to view distributions of delay times, power, or any other characteristic described in a .MEASURE statement. Often, this is more useful than viewing a family of curves, that a Monte Carlo analysis generates.

When you use the .MEASURE statement, HSPICE generates a table of results, in an .mt# file. You can read this file in ASCII format, and you can use AvanWaves to display it. Also, if you use .MEASURE statements in a Monte Carlo or data-driven analysis, then the HSPICE output file includes calculations for standard statistical descriptors:

$$\text{Mean} \quad = \frac{x_1 + x_2 + \ldots + x_n}{N}$$

$$\text{Variance} \quad = \frac{(x_1 - \text{Mean})^2 + \ldots(x_n - \text{Mean})^2}{N - 1}$$

$$\text{Sigma} \quad = \sqrt{\text{Variance}}$$

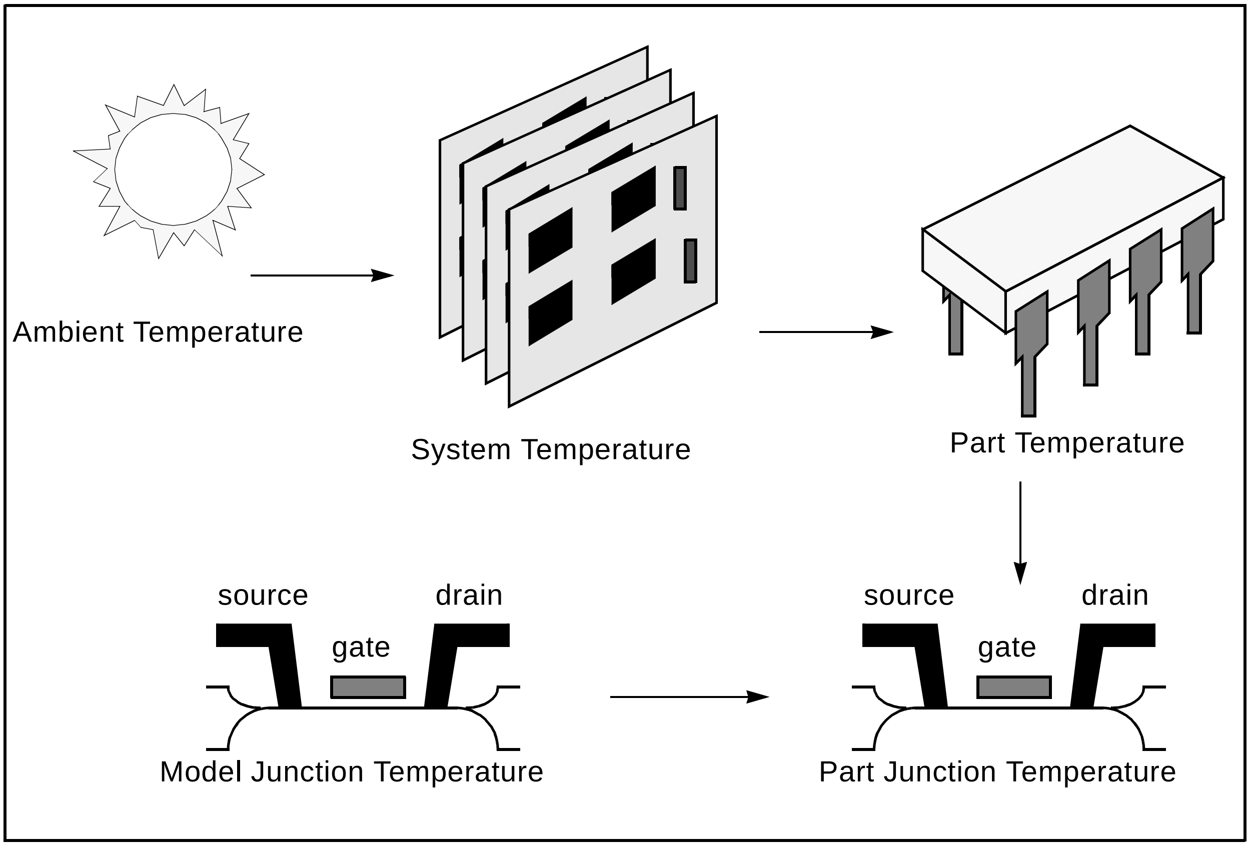
$$\text{Average Deviation} \quad = \frac{|x_1 - \text{Mean}| + \ldots + |x_n - \text{Mean}|}{N - 1}$$

Simulating Circuit and Model Temperatures

Temperature affects *all* electrical circuits. Figure 12-1 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 12-1   Part Junction Temperature Sets System Performance



HSPICE calculates temperatures as differences from the ambient temperature:

$$\mathrm{Tambient} + \Delta \mathrm{system} + \Delta \mathrm{part} + \Delta \mathrm{junction} = \mathrm{Tjunction}$$

$$\mathrm{Ids} = f(\mathrm{Tjunction}, \mathrm{Tmodel})$$

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 measures and extracts the model parameters. Set the value of TNOM in a .OPTION statement. Its default value is 25 °C.
- Circuit temperature, which you specify using a .TEMP statement or the TEMP parameter. This is the temperature, in °C, at which HSPICE 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 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 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). 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 simulation, use the .TEMP statement. See .TEMP Statement on page 3-22.

Worst Case Analysis

You can use *Worst Case* analysis (.wcase statement) when you design and analyze MOS and BJT IC circuits in HSPICE. To simulate the worst case, HSPICE sets all variables to their 2-sigma 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 True-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, polysilicon, 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.

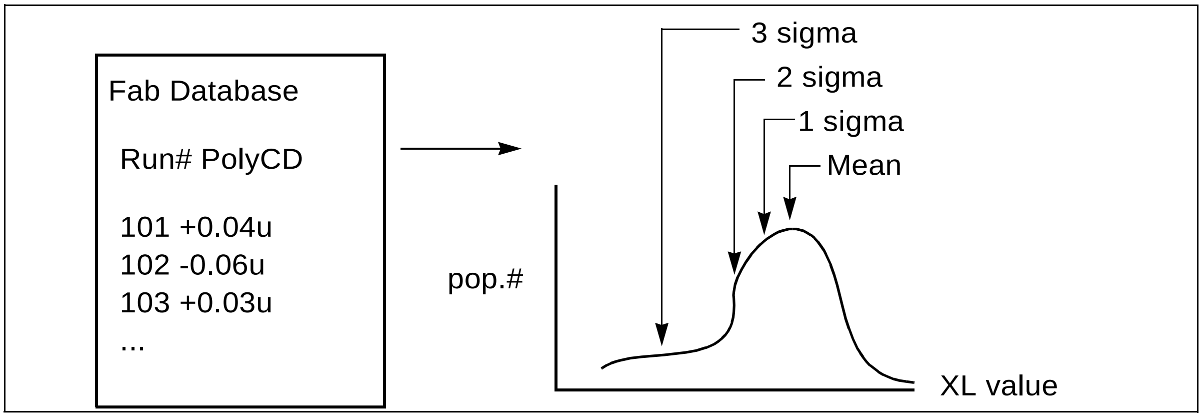
You can use these parameters in any level of MOS model, within the True-Hspice device models. The DELVTO parameter shifts the threshold value. HSPICE adds this value to VTO for the Level 3 model, and adds or subtracts it from VFB0 for the BSIM model. Table 12-1 on page 12-9 shows whether HSPICE adds or subtracts deviations from the average.

Table 12-1 Sigma Deviations

Type	**Param**	**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 12-2 is an example of data that historical records produce.

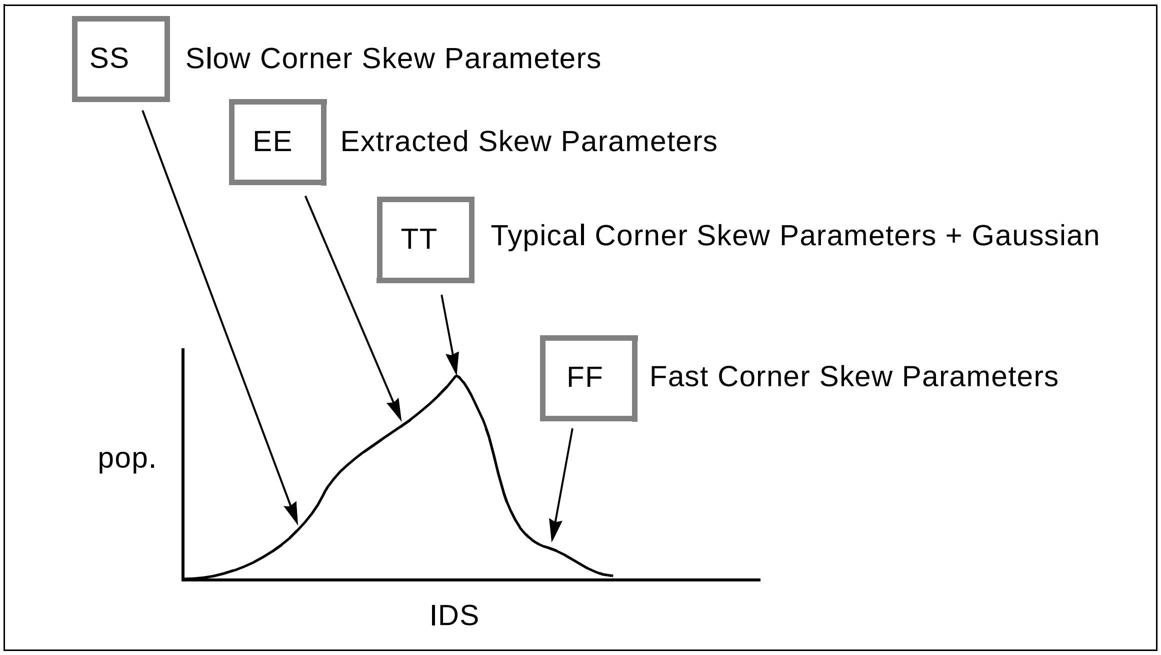
Figure 12-2 Historical Records for Skew Parameters in a MOS Process



Using Skew Parameters in HSPICE

Figure 12-3 shows how to create a worst-case, corners library file, for a CMOS process model in HSPICE. 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 12-3 Worst Case Corners Library File for a CMOS Process Model


SS
Slow Corner Skew Parameters
EE
Extracted Skew Parameters
TT
Typical Corner Skew Parameters + Gaussian
FF
Fast Corner Skew Parameters
pop.
IDS

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, using model skew parameters, features both worst-case and statistical-distribution data. In statistical distribution, the median value is the default for all non-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 side) equates to the skew parameter (right side). Model keynames and skew parameters can use the same names.

Skew File Interface to Device Models

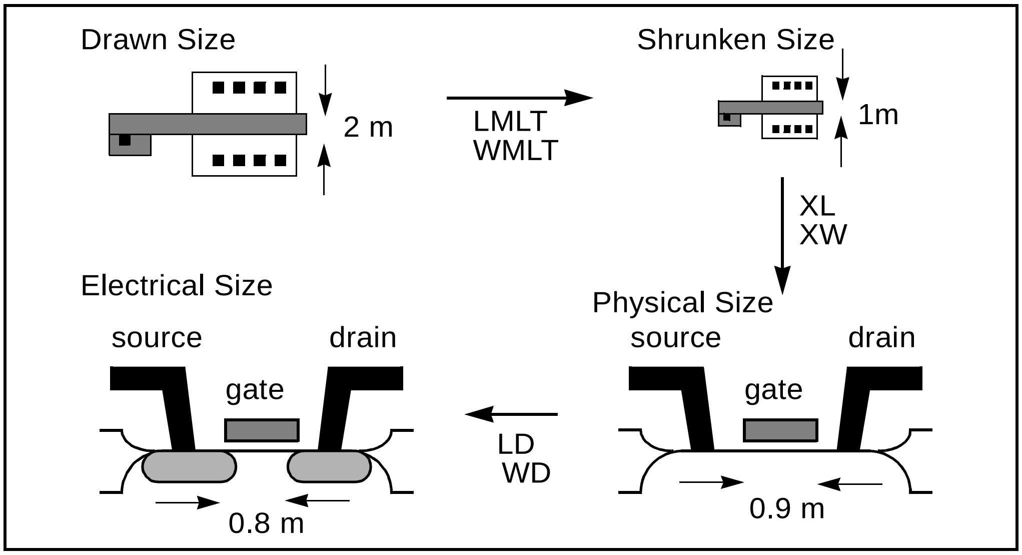
Skew parameters are model parameters, for transistor models or passive components. A typical device model set includes:

- MOSFET models, for all device sizes, 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 2 μ dimension, and shrink it to 1 μ, the physical size is 0.9 μ. The effective electrical size is 0.8 μ. Account for the four dimension levels:

  drawn size
  shrunken size
  physical size
  electrical size

Most simulator models scale directly from *drawn* to *electrical* size. True-Hspice MOS models support all four size levels, as explained in Figure 12-4 on page 12-13.

Figure 12-4 Device Model from Drawn to Electrical Size



Monte Carlo Analysis

Monte Carlo analysis uses a random number generator, to create the following types of functions.

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.

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.

SYNTAX:

Select the type of analysis to run, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating Point

```
.DC MONTE=val
```

DC Sweep

```
.DC vin 1 5 .25 SWEEP MONTE=val
```

AC Sweep

```
.AC dec 10 100 10meg SWEEP MONTE=val
```

TRAN Sweep

```
.TRAN 1n 10n SWEEP MONTE=val
```

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 $val^{-1/2}$.

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.

  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

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)
```

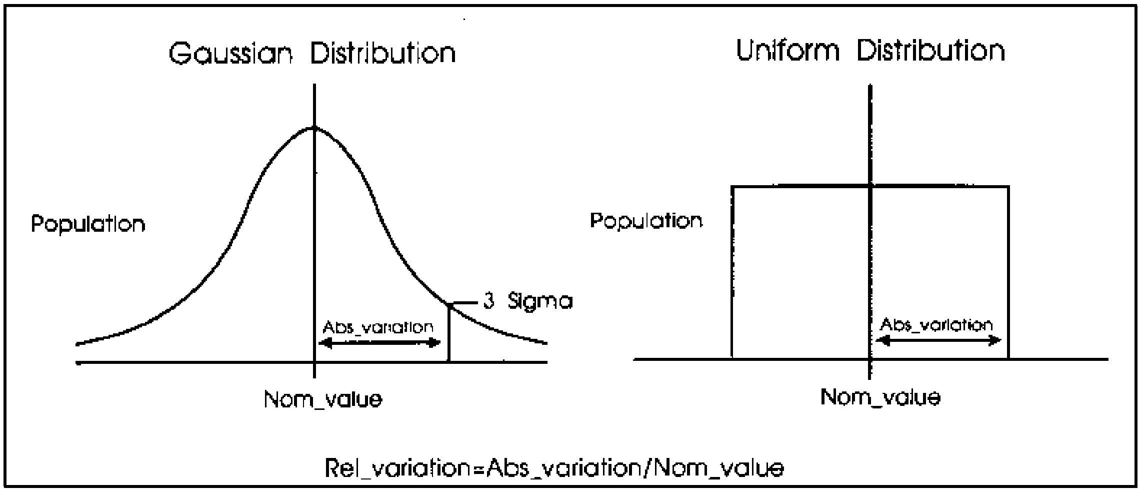
Table 12-2 .PARAM Syntax

Parameter	**Description**
xx	Distribution function calculates the value of this parameter.
UNIF	Uniform distribution function, using relative variation.
AUNIF	Uniform distribution function, using absolute variation.
GAUSS	Gaussian distribution function, using relative variation.
AGAUSS	Gaussian distribution function, using absolute variation
LIMIT	Random-limit distribution function, 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.

Table 12-2 .PARAM Syntax (Continued)

Parameter	**Description**
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.
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.

Figure 12-5 Monte Carlo Distribution



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

Gaussian, Uniform, and Limit Functions

```
Test of monte carlo gaussian, uniform, and limit functions
.OPTION post
.dc monte=60
* setup plots
.model histo plot ymin=80 ymax=120 freq=1
.graph model=HISTO aunif_1=v(au1)
.graph model=HISTO aunif_10=v(au10)
.graph model=HISTO agauss_1=v(ag1)
.graph model=HISTO agauss_10=v(ag10)
.graph model=HISTO 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 12-6   Gaussian Functions

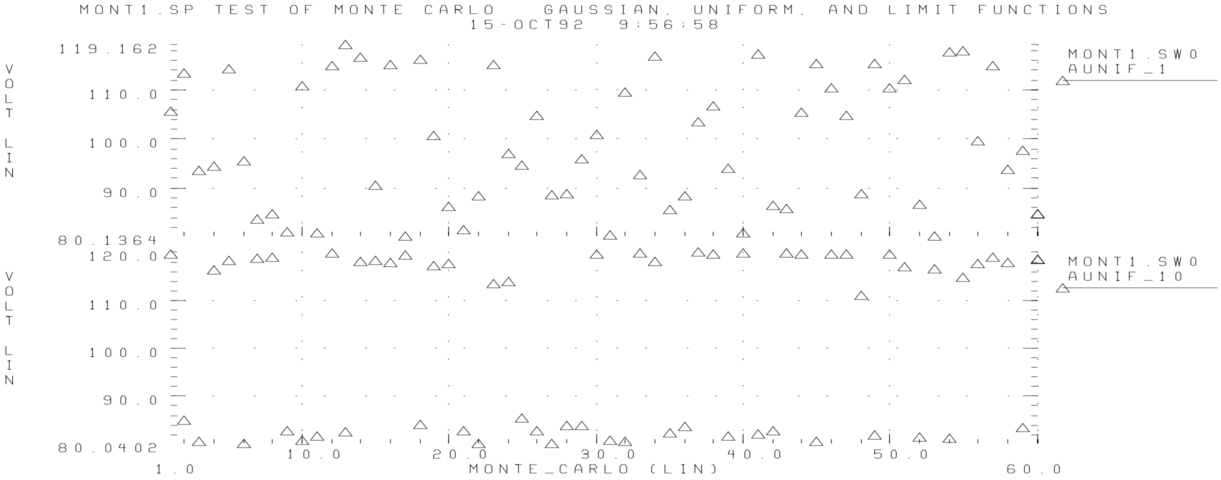


Figure 12-7 Uniform Functions

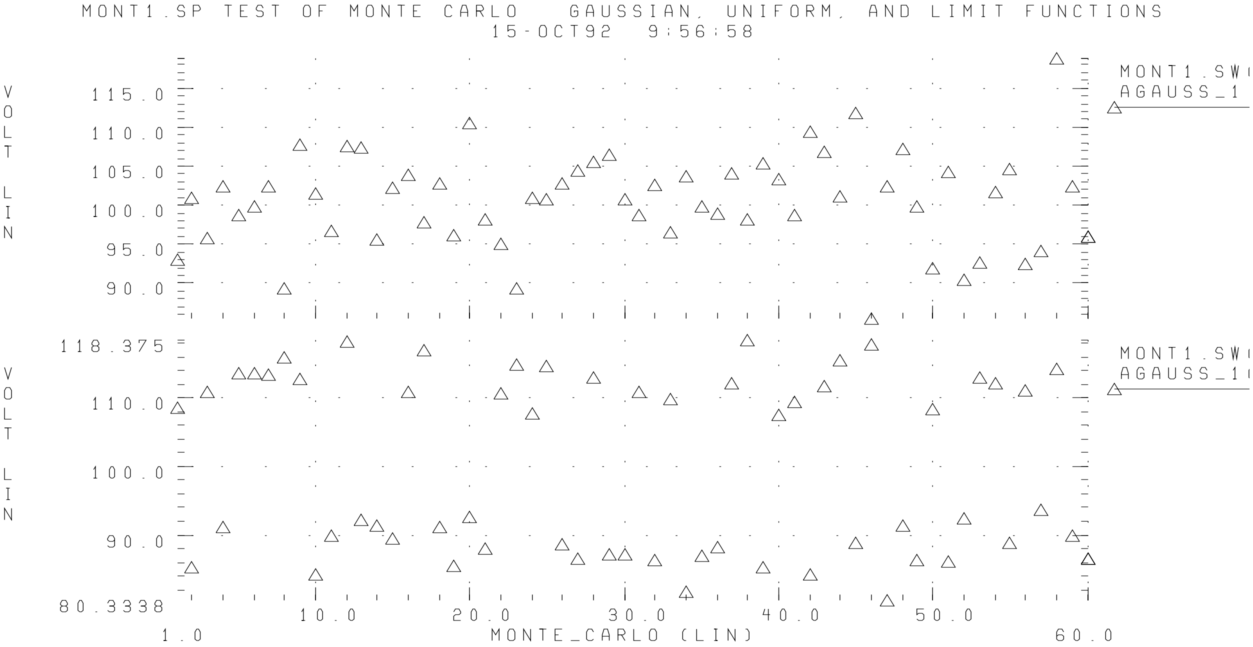
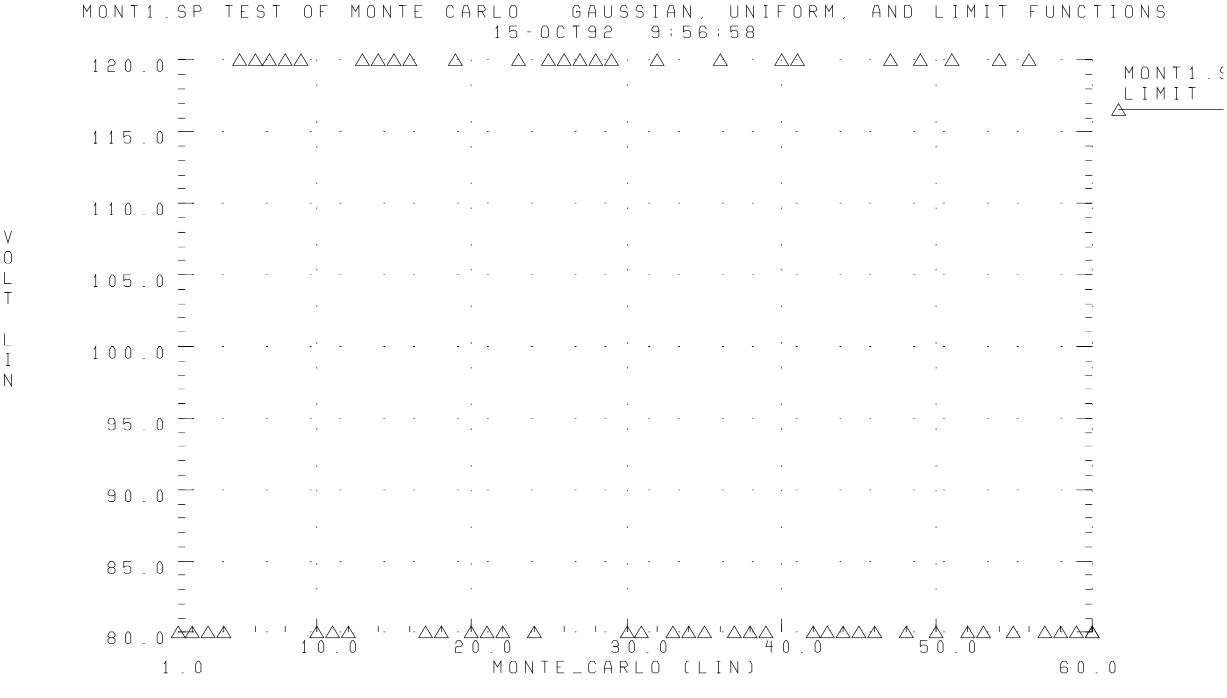


Figure 12-8 Limit Functions

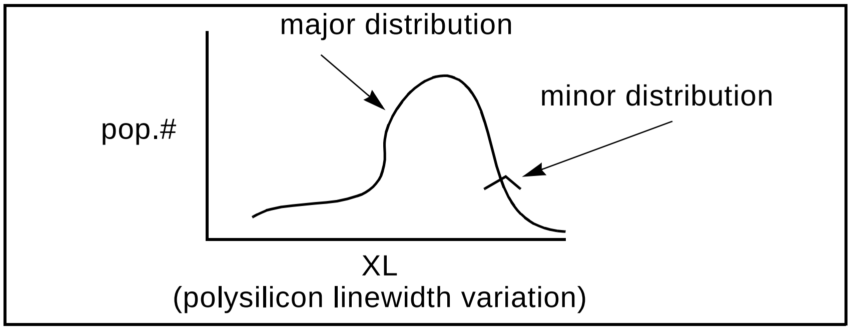


Major and Minor Distribution

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 12-9 Major and Minor Distribution of Manufacturing Variations



The example below is a Monte Carlo analysis of a DC sweep, in HSPICE. Monte Carlo sweeps the VDD supply voltage, from 4.5 volts to 5.5 volts.

```
File: MONDC_A.SP
.DC VDD 4.5 5.5  .1   SWEEP MONTE=30
.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
.INC Model.dat
.END
```

- 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.

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.

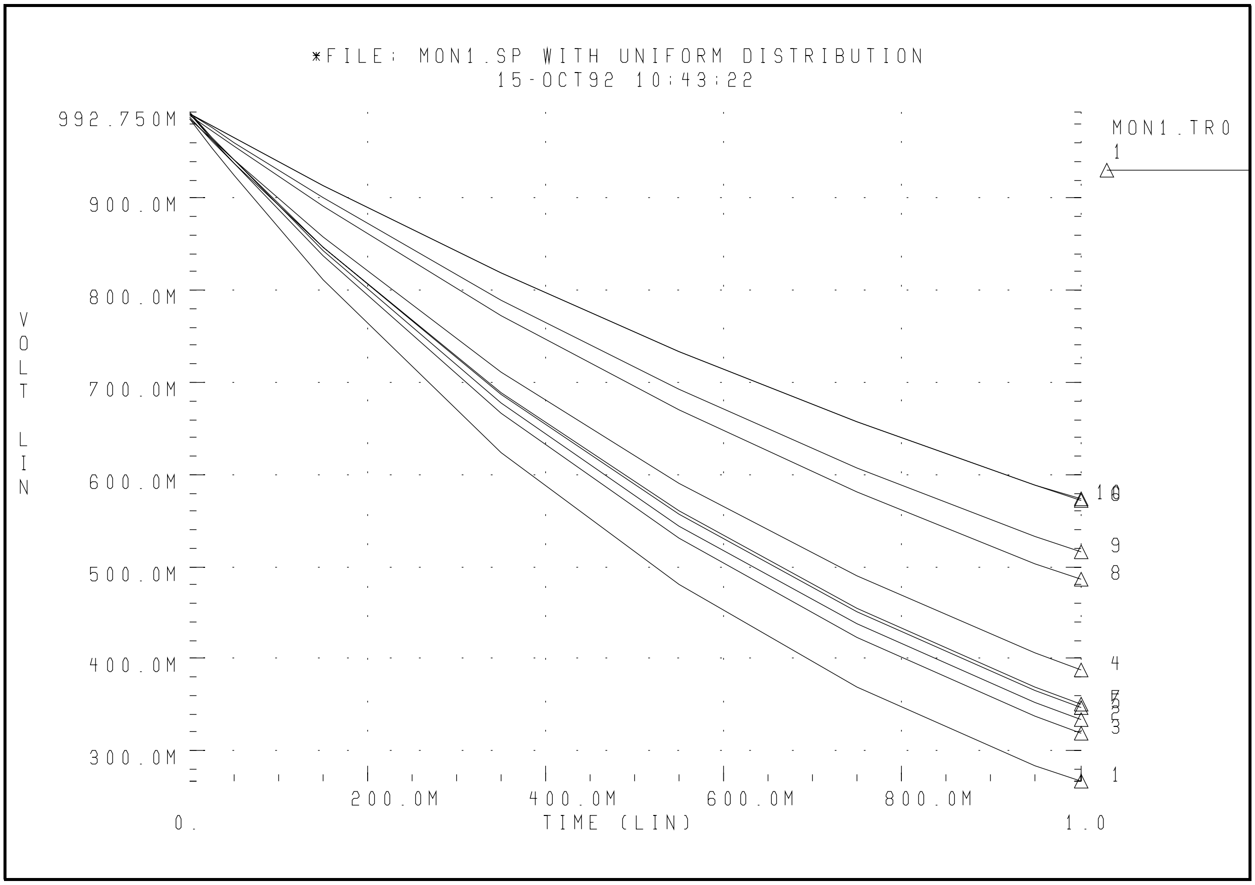
XPHOTO controls PHOTO lithography, for both NMOS and PMOS devices, which is consistent with the physics of manufacturing.

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.

```
*FILE: MON1.SP WITH UNIFORM DISTRIBUTION
.OPTION LIST POST=2
.PARAM RX=UNIF(1,  .5) CX=UNIF(1,  .5)
.TRAN  .1 1 SWEEP MONTE=10
.IC 1 1
R1 1 0   RX
C1 1 0   CX
.END
```

Figure 12-10   Monte Carlo Analysis of RC Time Constant

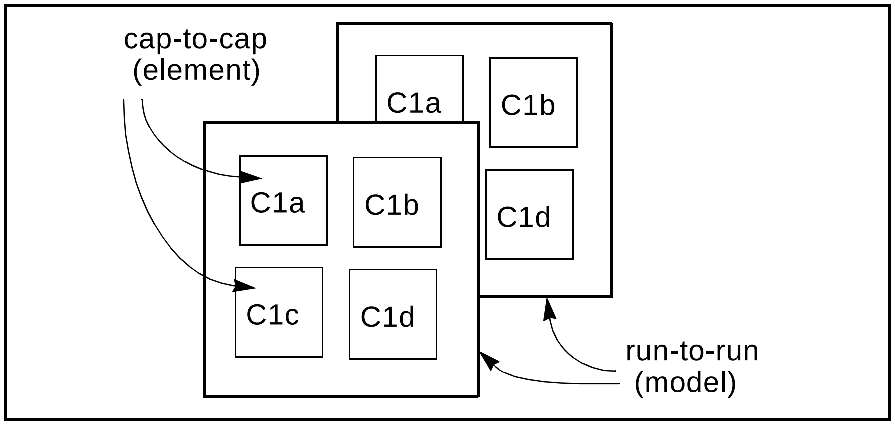


Switched Capacitor Filter Design

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 12-11 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. Oxide thickness control is excellent for small areas on a single wafer. Therefore, 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 linewidth control for a line 10 μ wide, manufactured with process A, is ±0.02 μ, for a 1-sigma distribution.
   - The global (model level) polysilicon line-width control is much wider; use 0.1 μ 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 · 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 of a pair of inverters.

- An inverter buffers the input.
- Another inverter loads the output.

The model is prepared according to the scheme described in the previous sections:

- The first .TRAN analysis statement sweeps from the worst-case 3-sigma slow, to 3-sigma fast.
- The second .TRAN performs 100 Monte Carlo sweeps.

HSPICE Input File

The HSPICE input file can contain the following sections.

Analysis Setup Section

To accelerate the simulation, the AUTOSTOP option automatically stops the simulation, when the .MEASURE statements achieve their target values.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma,
$ then Monte Carlo
.option nopage nomod acct
+ autostop post=2
.tran 20p 1.0n sweep sigma -3 3 .5
.tran 20p 1.0n sweep monte=20
.option post co=132
.param vref=2.5
.meas m_delay trig v(2) val=vref fall=1
+ targ v(out) val=vref fall=1
.meas m_power rms power to=m_delay
.param sigma=0
```

Circuit Netlist Section

```
.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 5 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
```

Skew Parameter Overlay for Model Section

```
* overlay of gaussian and algebraic for best case worst case
+ and + monte carlo
* +/- 3 sigma is the maximum value for parameter sweep

.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=’vtopcd+sigma*0.05’
+ rshncd=agauss(50,8,1) rshn=’rshncd-sigma*8’
+ rshpcd=agauss(150,20,1) rshp=’rshpcd-sigma*20’
```

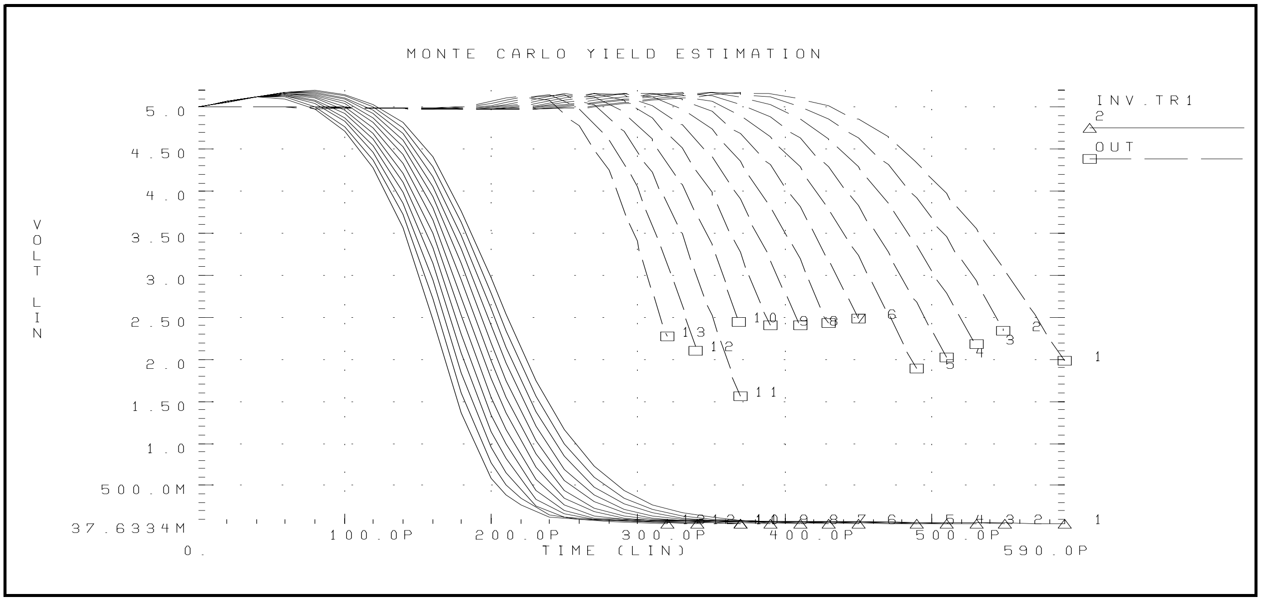
MOS Model for N-Channel and P-Channel Transistors

```
* LEVEL=28 example model for high accuracy 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
+ ld=0.06u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0
+ js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=1.4e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=0
+ vfb0=-.5 phi0=0.65 k1=.9 k2=.1 eta0=0
+ muz=500 u00=.075
+ x3ms=15 u1=.02 x3u1=0
+ b1=.28 b2=.22 x33m=0.000000e+00
+ alpha=1.5 vcr=20
+ n0=1.6 wfac=15 wfacu=0.25
+ lvfb=0 lk1=.025 lk2=.05 lalpha=5
.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
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=-1.7e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=5
+ vfb0=-.1 phi0=0.65 k1=.35 k2=0 eta0=0
+ muz=200 u00=.175
+ x3ms=8 u1=0 x3u1=0.0
+ b1=.25 b2=.25 x33m=0.0
+ alpha=0 vcr=20
+ n0=1.3 wfac=12.5 wfacu=.2
+ lvfb=0 lk1=-.05
.end
```

Transient Sigma Sweep Results

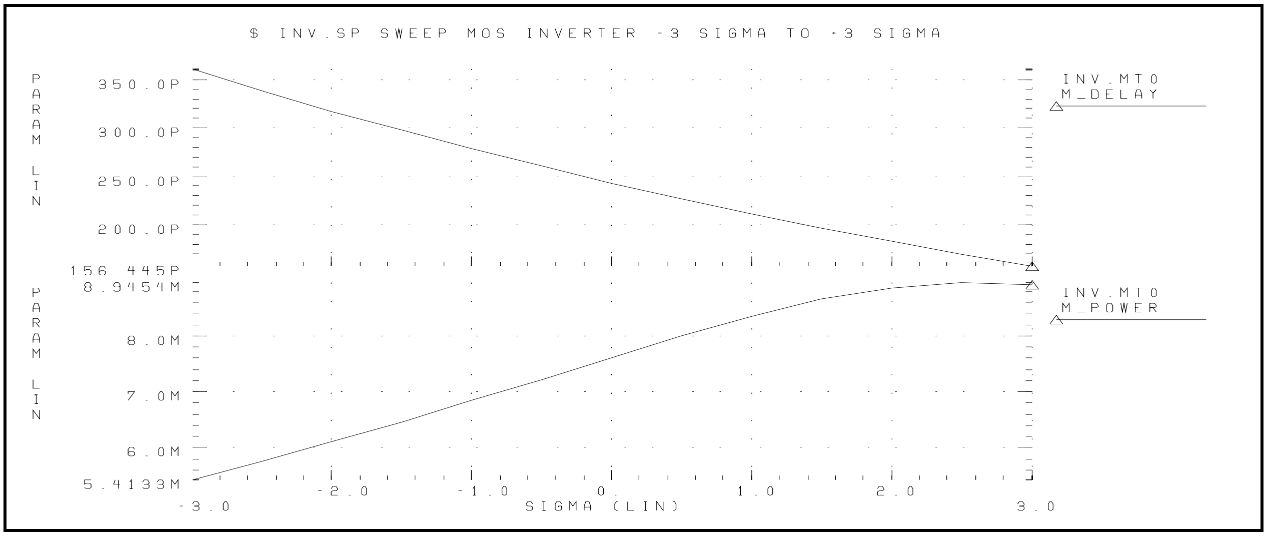
The plot in Figure 12-12 on page 12-29 shows the family of transient analysis curves, for the transient sweep of the sigma parameter, from -3 to +3. HSPICE then algebraically couples sigma into the skew parameters. The resulting parameters modify the actual NMOS and PMOS models.

Figure 12-12 Sweep of Skew Parameters from -3 Sigma to +3 Sigma



To view the transient curves, plot the .MEASURE output file. The plot in Figure 12-13 on page 12-29 shows the measured pair delay, and the total dissipative power, against the SIGMA parameter.

Figure 12-13 Sweep MOS Inverter, Pair Delay and Power: -3 Sigma to 3 Sigma



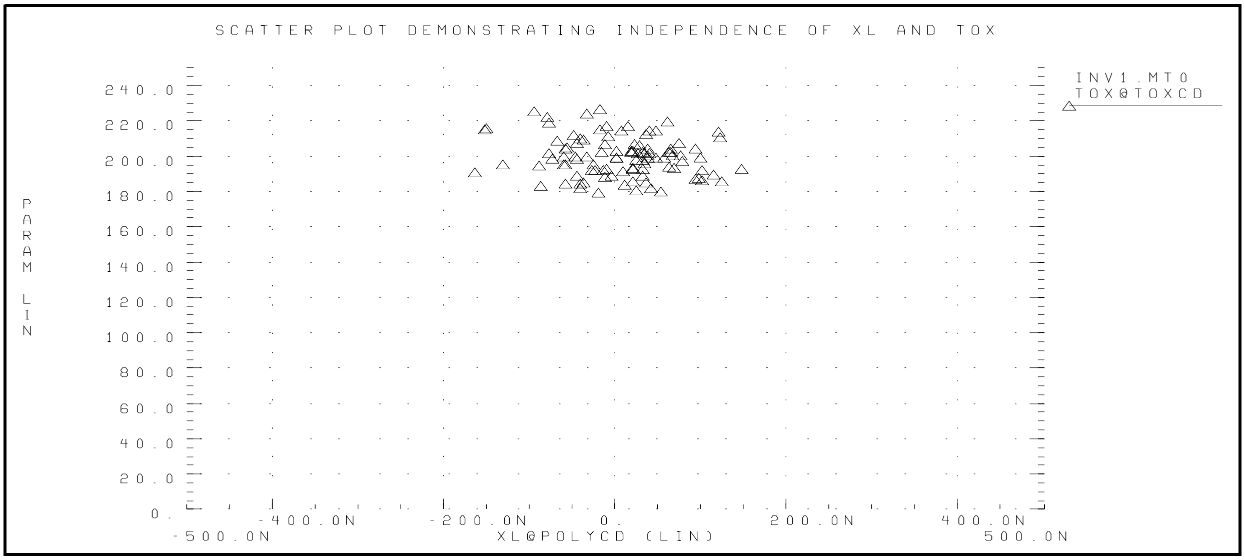
Monte Carlo Results

This section evaluates the output of the Monte Carlo analysis in HSPICE. The plot in Figure 12-14 on page 12-30 is a quality-control step, which plots *TOX* against *XL* (polysilicon critical dimension). Synopsys graphing software returned the cloud of points, based on:

- Setting *XL* as the X-axis independent variable.
- Plotting *TOX*, with a symbol frequency of 1.

These settings plot points, without connecting lines. The resulting graph demonstrates that the *TOX* model parameter is randomly independent of *XL*.

Figure 12-14 Scatter Plot, XL and TOX



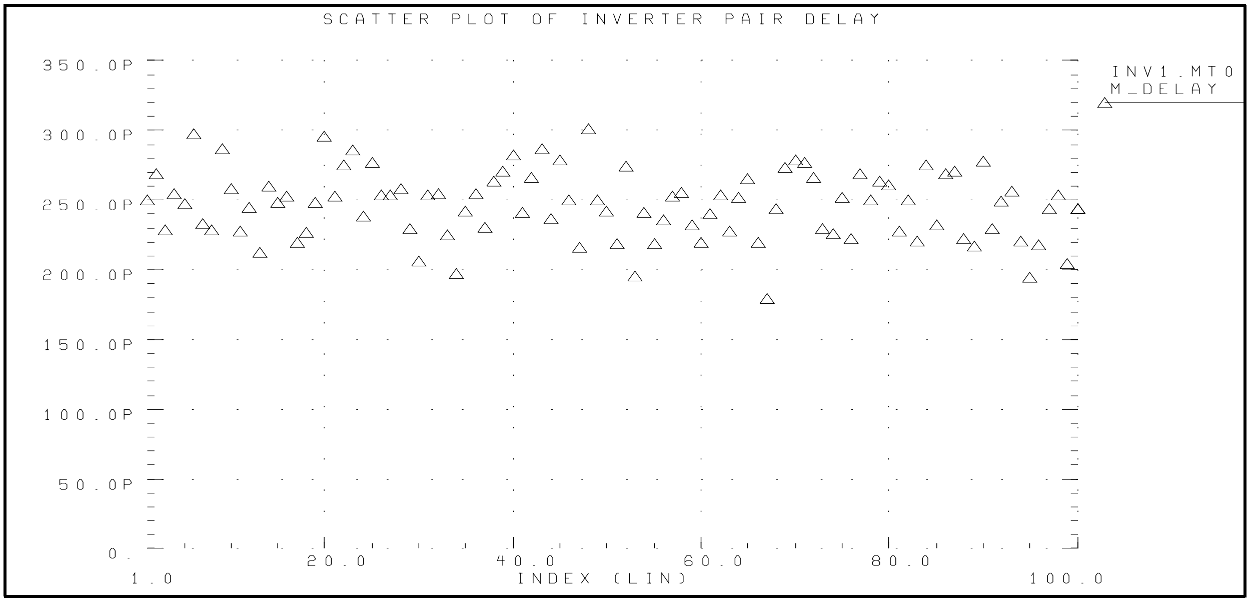
The next graph (see Figure 12-15 on page 12-31) is a standard scatter plot, showing the measured delay for the inverter pair, against the Monte Carlo index number. If a particular result looks interesting—for example, if the simulation 68 (*monte carlo index = 68*) produces the smallest delay—then you can read the output listing file, and 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
vtopcd: delvtop = 4.1776E-02
rshncd: rshn = 45.16
rshpcd: rshp = 166.2

m_delay = 1.7946E-10 targ= 3.4746E-10 trig= 1.6799E-10
m_power = 7.7781E-03 from= 0.0000E+00 to= 1.7946E-10
```

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.

Figure 12-15 Scatter Plot of Inverter Pair Delay



Plotting against the Monte Carlo index number does not help to center the design. To center the design:

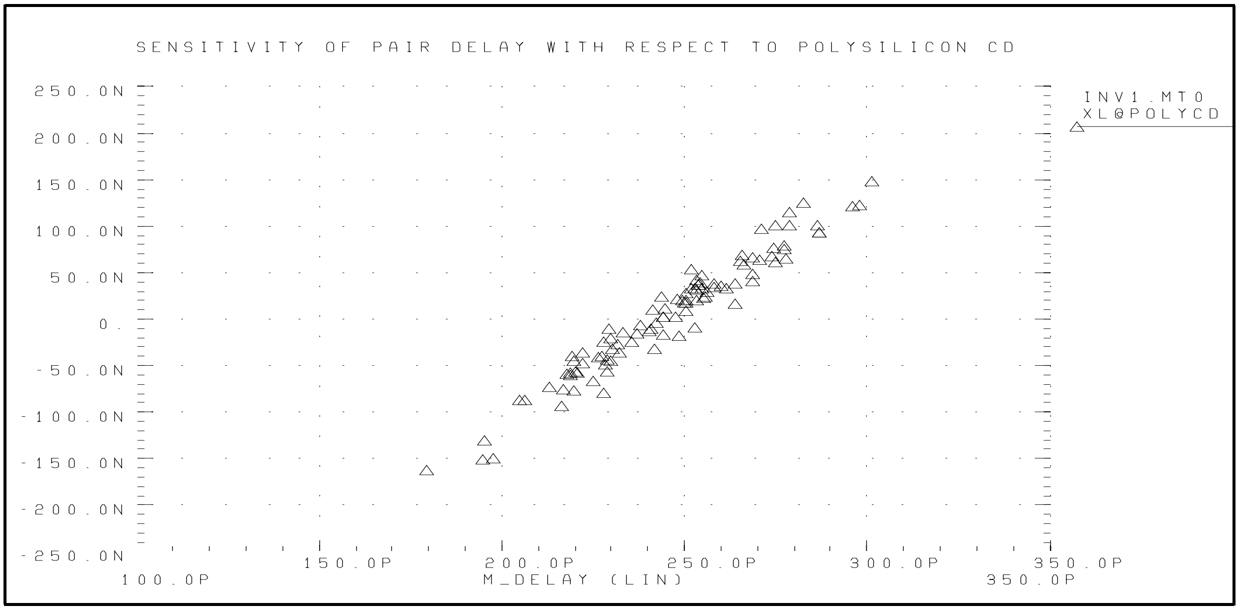
1. Graph the various process parameters against the pair delay.

   This graph determines the most sensitive process variables.

2. Select the pair delay, as the X-axis independent variable.
3. Set the symbol frequency to 1, to obtain the scatter plot.

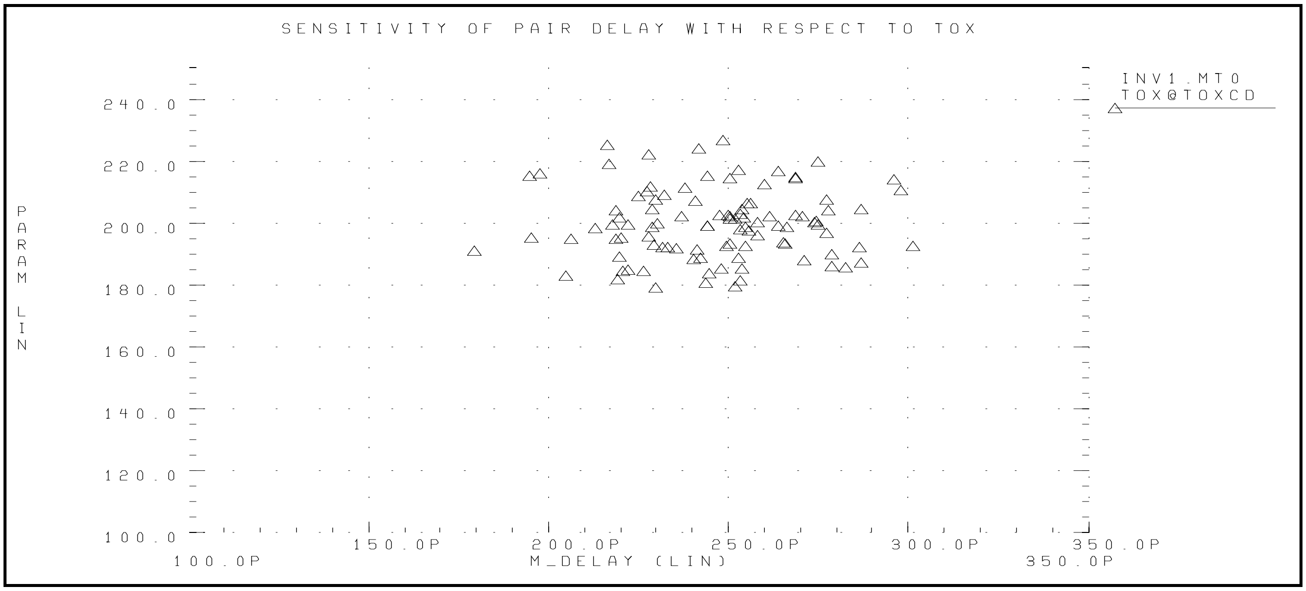
Figure 12-16 plots the expected sensitivity of the output pair delay, to channel length variation (polysilicon variation).

Figure 12-16 Sensitivity of Delay with Poly CD (XL)



The next plot shows the *TOX* parameter, against the pair delay (Figure 12-17 on page 12-33). The scatter plot does not have a clear tilt, because *TOX* is a secondary process parameter, compared to *XL*. To explore this in more detail, set the *XL* skew parameter to a constant, and run a simulation.

Figure 12-17 Sensitivity of Delay with TOX



The plot in Figure 12-18 on page 12-34 overlays a 3-sigma, worst-case corners response, on a 100-point Monte Carlo analysis. The actual (Monte Carlo) distribution for power/delay is very different from the +3 sigma to -3 sigma plot.

- This example simulated the worst case in 0.5 sigma steps.
- The actual response is closer to ± 1.5 sigma, instead of ± 3 sigma.
- This produces a predicted delay variation of 100 ps, instead of 200 ps.

Therefore, the advantage of using Monte Carlo over traditional 3-sigma, worst-case corners, is a 100% improvement in accuracy of simulated-to-actual distribution. This shows how the worst-case procedure is overly pessimistic.

Figure 12-18 Superimposing Sigma Sweep Over Monte Carlo

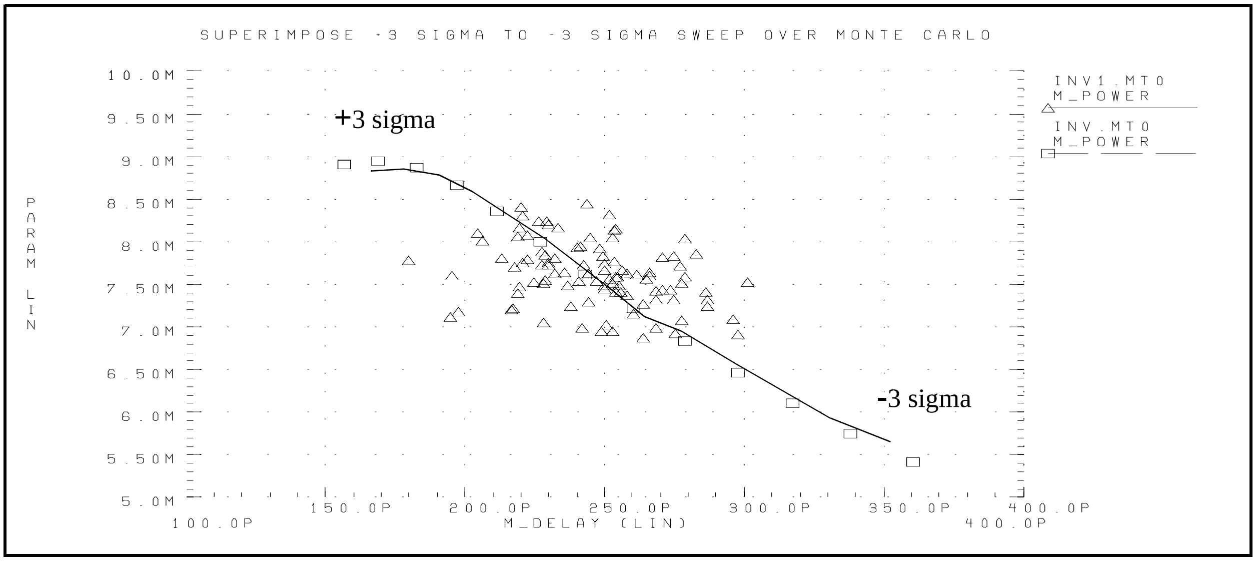
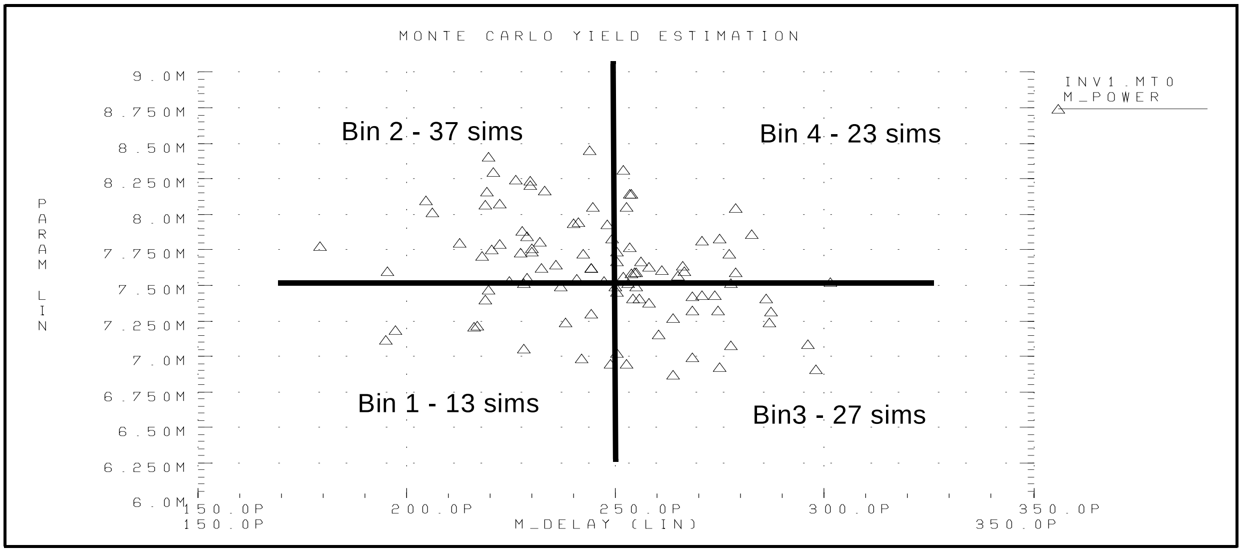


Figure 12-19 on page 12-35 superimposes the assumed part grades from marketing studies, onto the Monte Carlo plot. This example uses a 250 ps delay, and 7.5 mW power dissipation, to determine the four binning grades. A manual count shows:

- Bin1 - 13%
- Bin2 - 37%
- Bin3 - 27%
- Bin4 - 23%

If this circuit is representative of the entire chip, then the present yield should be 13% for the premium *Bin 1* parts, assuming variations in design and process.

Figure 12-19 Speed/Power Yield Estimation



Optimization

Optimization automatically generates model parameters and component values, from a set of electrical specifications or measured data. With 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.

HSPICE optimization is the result of more than ten years of research, in both optimizing algorithms and user interface.

- The optimizing function is integrated into the core of HSPICE, for optimum efficiency.
- The circuit-result targets are part of the .MEASURE command structure.
- HSPICE optimize its own internally-defined parameter functions.

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 Input and Output Options on page 8-34 for more information about these options.

The most powerful feature of HSPICE optimization is its incremental optimization technique. You can use this technique to solve 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.

1. To specify goal optimizations, 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. You can use various types of input and output variables, and a transfer function to relate them.

EXAMPLE:

- voltage
- current
- delay time
- gain

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 Syntax

Optimization requires several HSPICE statements:

- .MODEL *modname* OPT ...
- .PARAM *parameter*=OPT*xxx* (*init*, *min*, *max*)

  Use the .PARAM statement to specify initial, lower, and upper bounds.

- A .DC, .AC, or .TRAN analysis statement, with:

  - MODEL=*modname*
  - OPTIMIZE=OPT*xxx*
  - RESULTS=*measurename*

  Use the .PRINT, .PLOT, and .GRAPH output statements, with the .DC, .AC, or .TRAN analysis statements.

  Use an analysis statement, with the OPTIMIZE keyword, only 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 Simulation Output on page 7-1.

The proper specification order is:

1. Analysis statement, with OPTIMIZE.
2. .MEASURE statements, specifying optimization goals or error functions.
3. Ordinary analysis statement.
4. Output statements.

Analysis Statement (.DC, .TRAN, .AC)

SYNTAX:

```
.DC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

```
.AC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

See also Optimization on page 11-5.

```
.TRAN  <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

Table 12-3 DC, TRAN, and AC Analysis Syntax

Parameter	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.
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.

.PARAM Statement

SYNTAX:

```
.PARAM parameter=OPTxxx (initial_guess, low_limit,
+ upper_limit)
```

or

```
.PARAM parameter=OPTxxx (initial_guess, low_limit,
+ upper_limit, delta)
```

Table 12-4 .PARAM Syntax

Parameter	**Description**
OPTxxx	Optimization parameter reference name. The associated optimization analysis references this name. Must agree with the OPTxxx name in the analysis command associated with an OPTIMIZE keyname.
parameter	Parameter to vary. • Initial value estimate • Lower limit. • Upper limit. If the optimizer does not find the best solution within these constraints, it attempts to find the best solution without constraints.
delta	The final parameter value is the initial guess ± (*n*·delta). If you do not specify *delta*, the final parameter value is between *low_limit* and *upper_limit*. For example, you can use this parameter to optimize transistor drawn widths and lengths, which must be quantized.

In the following example, *uox* and *vtx* are the variable model parameters, which optimize a model for a selected set of electrical specifications.

```
.PARAM vtx=OPT1(.7,.3,1.0) uox=OPT1(650,400,900)
```

The estimated initial value for the *vtx* parameter is 0.7 volts. You can vary this value within the limits of 0.3 and 1.0 volts, for the optimization procedure. The optimization parameter reference name (OPT1) references the associated optimization analysis statement (not shown).

.MODEL Statement

For each optimization within a data file, specify a .MODEL statement. HSPICE can then execute more than one optimization per simulation run. The .MODEL optimization statement defines:

- Convergence criteria.
- Number of iterations.
- Derivative methods.

SYNTAX:

```
.MODEL mname OPT <parameter=val ...>
```

You can specify the following OPT parameters in the .MODEL statement:

Table 12-5 .MODEL Syntax

Parameter	Description
mname	Model name. Elements use this name to refer to the model.
CENDIF	Selects different derivative methods. Default=1.0e-9. The following calculates the gradient of the RESULTS functions: ||Transpose(Jacobi(F(X))) * F(X)||, where F(X) is the RESULT function If the resulting gradient is less than CENDIF, HSPICE uses more accurate but more time-consuming derivative methods. By default, HSPICE uses faster but less-accurate derivative methods. To use the more-accurate methods, set CENDIF to a larger value than GRAD. If the gradient of the RESULTS function is less than GRAD, optimization finishes before CENDIF takes effect. • If the value is too large, the optimizer requires more CPU time. • If the value is too small, the optimizer might not find as accurate an answer.
CLOSE	Initial estimate of how close parameter initial value estimates are, to the solution. CLOSE multiplies changes in new parameter estimates. If you use a large CLOSE value, the optimizer takes large steps toward the solution. For a small value, the optimizer takes smaller steps toward the solution. You can use a smaller value for close parameter estimates, and a larger value for rough initial guesses. Default=1.0. • If CLOSE is greater than 100, the steepest descent in the Levenburg-Marquardt algorithm dominates. • If CLOSE is less than 1, the Gauss-Newton method dominates. For more details, see L. Spruiell, “Optimization Error Surfaces,” *Meta-Software Journal,* Vol. 1, No. 4, December 1994.
CUT	Modifies CLOSE, depending on how successful iterations are, toward the solution. If the last iteration succeeds, descent toward the `CLOSE` solution decreases by the `CUT` value. That is, CLOSE = CLOSE / CUT If the last iteration was not a successful descent to the solution, `CLOSE` increases by `CUT` squared. That is, CLOSE = CLOSE * CUT * CUT CUT drives CLOSE up or down, depending on the relative success in finding the solution. The CUT value must be > 1. Default = 2.0.

Table 12-5 .MODEL Syntax (Continued)

Parameter	Description
DIFSIZ	Increment change in a parameter value, for gradient calculations ($\Delta x = DIFSIZ \cdot max(x, 0.1)$). If you specify delta in a .PARAM statement, then Δx = delta. Default = 1e-3.
GRAD	Represents possible convergence, if the gradient of the RESULTS function is less than GRAD. Most applications use values of 1e-6 to 1e-5. Too large a value can stop the optimizer before finding the best solution. Too small a value requires more iterations. Default=1.0e-6.
ITROPT	Maximum number of iterations. Typically, you need no more than 20-40 iterations, to find a solution. Too many iterations can imply that the RELIN, GRAD, or RELOUT values are too small. Default=20.
LEVEL	Selects an optimizing algorithm. Currently, the only option is LEVEL=1, a modified Levenburg-Marquardt algorithm.
MAX	Sets the upper limit on CLOSE. Use values > 100. Default=6.0e+5.
PARMIN	Allows better control of incremental parameter changes, during error calculations. Default=0.1. This produces more control over the trade-off between simulation time and optimization result accuracy. To calculate parameter increments, HSPICE uses the relationship: Δπαρ_ϖαλ = DIFSIZ · MAX(παρ_ϖαλ, PARMIN)
RELIN	Sets the relative input parameter (delta_par_val / MAX(par_val,1e-6)), for convergence. If all optimizing input parameters vary by no more than RELIN between iterations, the solution converges. RELIN is a relative variance test, so a value of 0.001 implies that optimizing parameters vary by less than 0.1%, from one iteration to the next. Default=0.001.
RELOUT	Sets the relative tolerance to finish optimization. For RELOUT=0.001, if the relative difference in the RESULTS functions, from one iteration to the next, is less than 0.001, then optimization is finished. Default=0.001.

Optimization Examples

This section contains examples of HSPICE optimizations:

- 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. The POST option generates AvanWaves files, for comparing the model to the data.

Input Netlist File, for Level 3 Model DC Optimization

```
$LEVEL 3 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=3
+ acm=2 ldif=0 hdif=4u tlev=1 n=2
+ capop=4 meto=0.08u xqc=0.4
$...note capop=4 is ok for H8907 and later, otherwise
$...use Capop=2
$...fixed parameters
+ wd=0.15u ld=0.07u
+ js=1.5e-04 jsw=1.8e-09
+ cj=1.7e-04 cjsw=3.8e-10
+ nfs=2e11 xj=0.1u delta=0 eta=0

$...process skew parameters
+ tox=toxn rsh=rshn
+ xw=xwn xl=xln
```

Optimized Parameters

```
+ vto=vto gamma=gamma
+ uo=uo vmax=vmax theta=theta kappa=kappa

.PARAM
+ vto    = opt1(1,0.5,2)
+ gamma  = opt1(0.8,0.1,2)
+ uo     = opt1(480,400,1000)
+ vmax   = opt1(2e5,5e4,5e7)
+ theta  = opt1(0.05,1e-3,1)
+ kappa  = opt1(2,1e-2,5)
```

Optimization Sweeps

```
.DC DATA=all optimize=opt1 results=comp1 model=optmod
.MEAS DC comp1 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05
```

DC 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 all vds vgs vbs ids
1.000000e-01 1.000000e+00 0.000000e+00 1.655500e-05
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDATA

.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 2.302000e-04
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDDATA
.END
```

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.

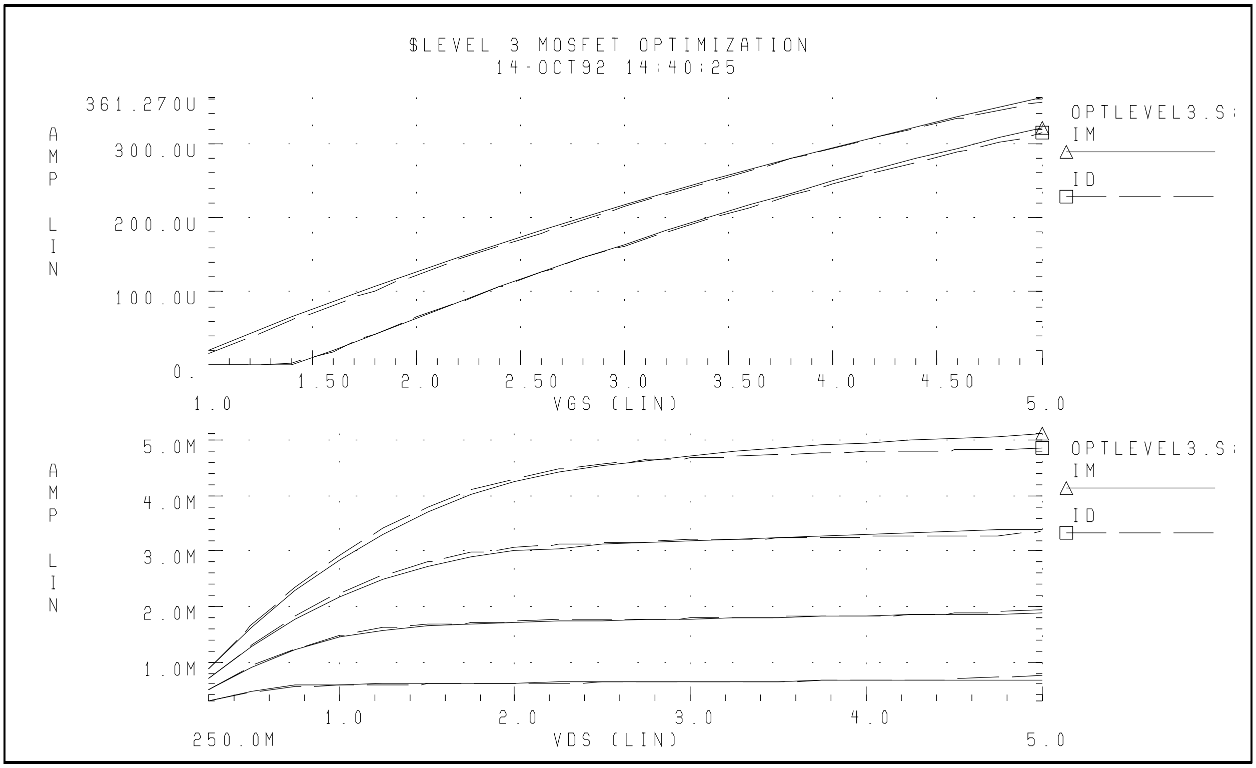
EXAMPLE:

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 12-20 shows the results.

Figure 12-20 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 12-21 on page 12-53 shows the results.

DC Optimization Input Netlist File, for Level 13 Model

```
$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

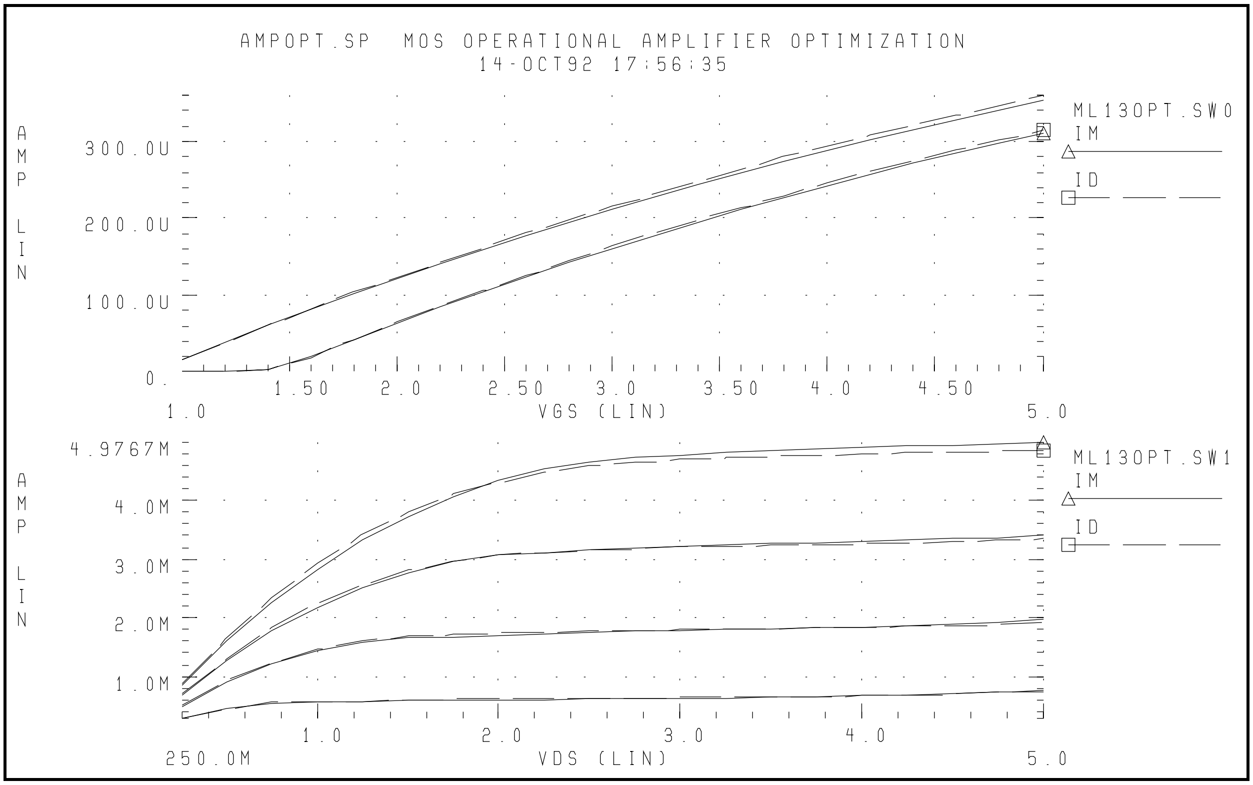
```
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
```

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 12-21   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.
- OPTrc 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.

RC Network Optimization Input Netlist File

```
.title RCOPT.sp
.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

.ic 1 1
R1 1 0 RX
c1 1 0 CX
```

Optimization Results

```
RESIDUAL SUM OF SQUARES     = 1.323651E-06
NORM OF THE GRADIENT        = 6.343728E-03
MARQUARDT SCALING PARAMETER = 2.799235E-06
NO. OF FUNCTION EVALUATIONS =       24
NO. OF ITERATIONS           =         12
```

Residual Sum of Squares

The residual sum of squares is a measure of the total error. The smaller this value is, the more accurate the optimization results are.

$$\text{residual sum of squares} = \sum_{i = 1}^{ne} E_i^2$$

In this equation, *E* is the error function, and *ne* is the number of error functions.

Norm of the Gradient

The norm of the gradient is another measure of the total error. The smaller this value is, the more accurate the optimization results are. The following equations calculates the *G* gradient:

$$G_j = \sum_{i = 1}^{ne} E_i \cdot (DE_i / DP_j)$$

$$\text{norm of the gradient} = 2 \cdot \sqrt{\sum_{i = 1}^{np} G_j^2}$$

In this equation, *P* is the parameter, and *np* is the number of parameters to optimize.

Marquardt Scaling Parameter

The Levenburg-Marquardt algorithm uses this parameter to find the actual solution for the optimizing parameters. The search direction is a combination of the Steepest Descent method, and the Gauss-Newton method.

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

```
*                                  %NORM-SEN   %CHANGE
.PARAM RX           =   6.7937   $  54.5260    50.2976M
.PARAM CX           = 147.3697M $  45.4740    33.7653M
```

Figure 12-22 Power Dissipation and Time Constant (VOLT) RCOPT.TR0 = Before Optimization, RCOPT.TR1 = Optimized Result

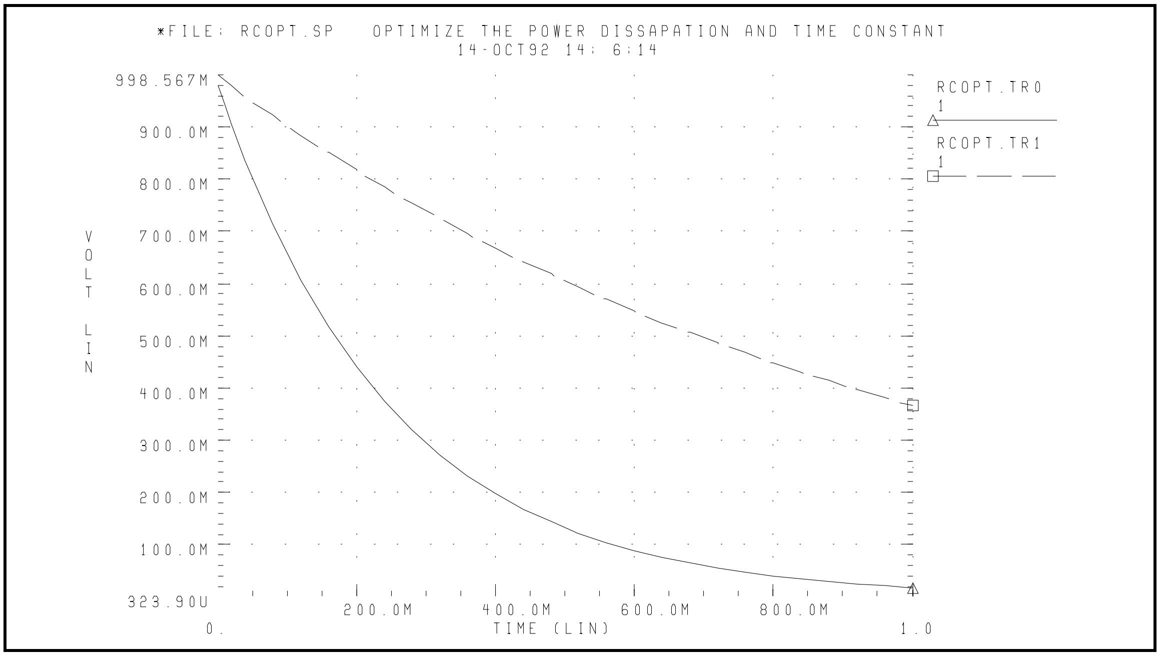
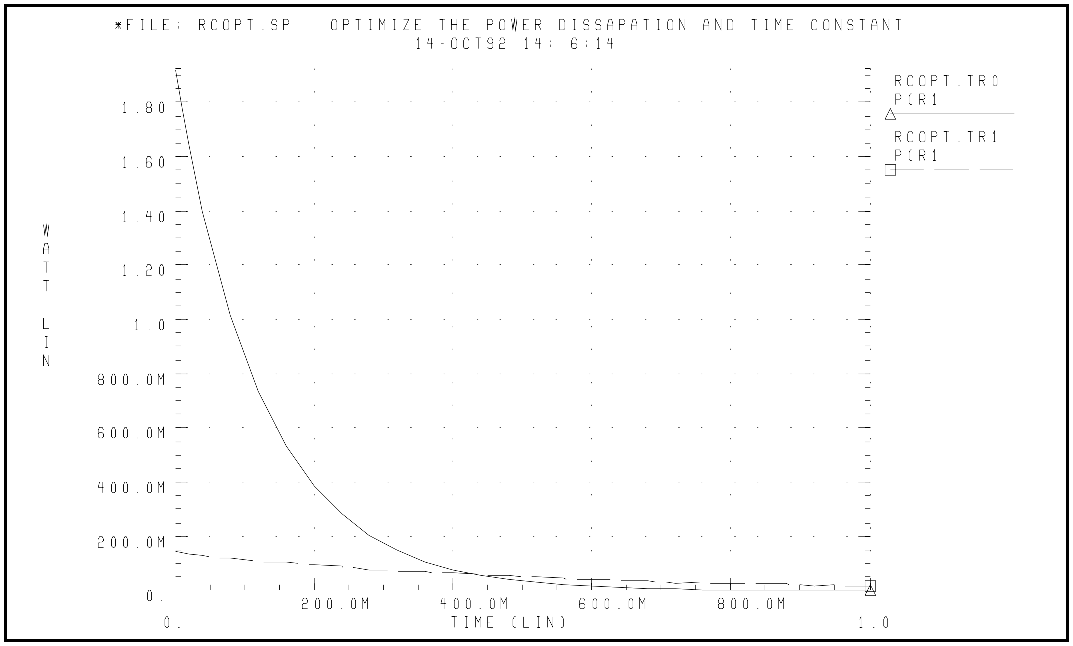


Figure 12-23 Power Dissipation and Time Constant (WATT) RCOPT.TR0 = Before Optimization, RCOPT.TR1 = Optimized Result



Optimizing CMOS Tristate Buffer

The example circuit is an inverting CMOS tristate buffer. The design targets are:

1. Rising edge delay of 5 ns (input 50% voltage, to output 50% voltage).
2. Falling edge delay of 5 ns (input 50% voltage. to output 50% voltage).
3. RMS power dissipation should be as low as possible.
4. 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

```
*Tri-State input/output Optimization
.OPTION defl=1.2u nomod post=2
+ 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 15ns
.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
.mea tran tfopt trig v(data) val=2.5 rise=1 targ v(out)
+ val=2.5 fall=1 goal 5.0n

.mea 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=30 max=1e+5

.tran 1ns 15ns
* output section *
*.plot tran v(data) v(out)
.plot 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
```

Optimization Results

```
residual sum of squares     = 2.388803E-02
norm of the gradient        = 0.769765
marquardt scaling parameter =   12624.2
no. of function evaluations =      175
no. of iterations          =       23
```

Optimization Completed

```
Parameters relin= 1.0000E-03 on last iterations
```

Optimized Parameters OPT1

```
*                                %norm-sen   %change
.param wp2          =  84.4981u  $   22.5877  -989.3733u
.param wn2          =  34.1401u  $    7.6568  -659.2874u
.param wp3          = 161.7354u  $  730.7865m -351.7833u
.param wn3          = 248.6829u  $    8.1362    -2.2416m
.param wp4          = 238.9825u  $    1.2798    -1.5774m
.param wn4          =  61.3509u  $  315.4656m   43.5213m
.param wp5          =   1.7753m  $    4.1713     2.1652m
.param wn5          =   1.0238m  $    5.8506   413.9667u
.param wp10         = 268.3125u  $    8.1917    -2.0266m
.param wn10         = 115.6907u  $   40.597  -422.8411u
.param wp11         = 153.1344u  $  482.0655m   -30.6813m

*** optimize results measure names and values
* tfopt          =   5.2056n
* tropt          =   5.5513n
* rmspowo        = 200.1808m
```

Figure 12-24 Tristate Buffer Optimization Circuit

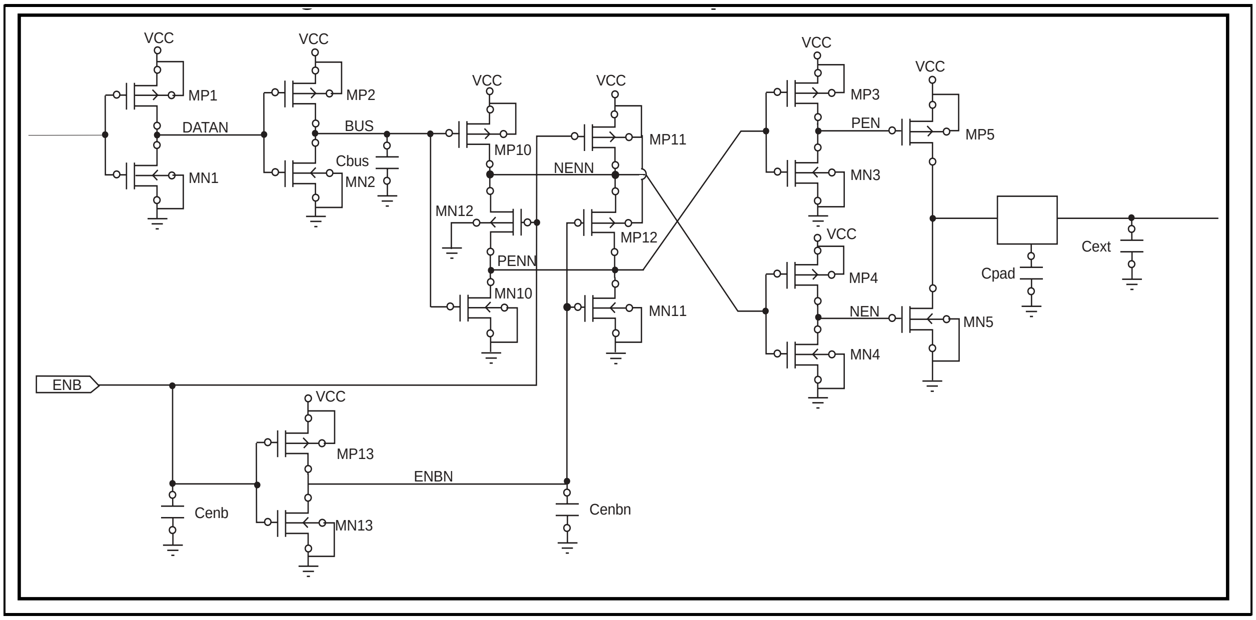
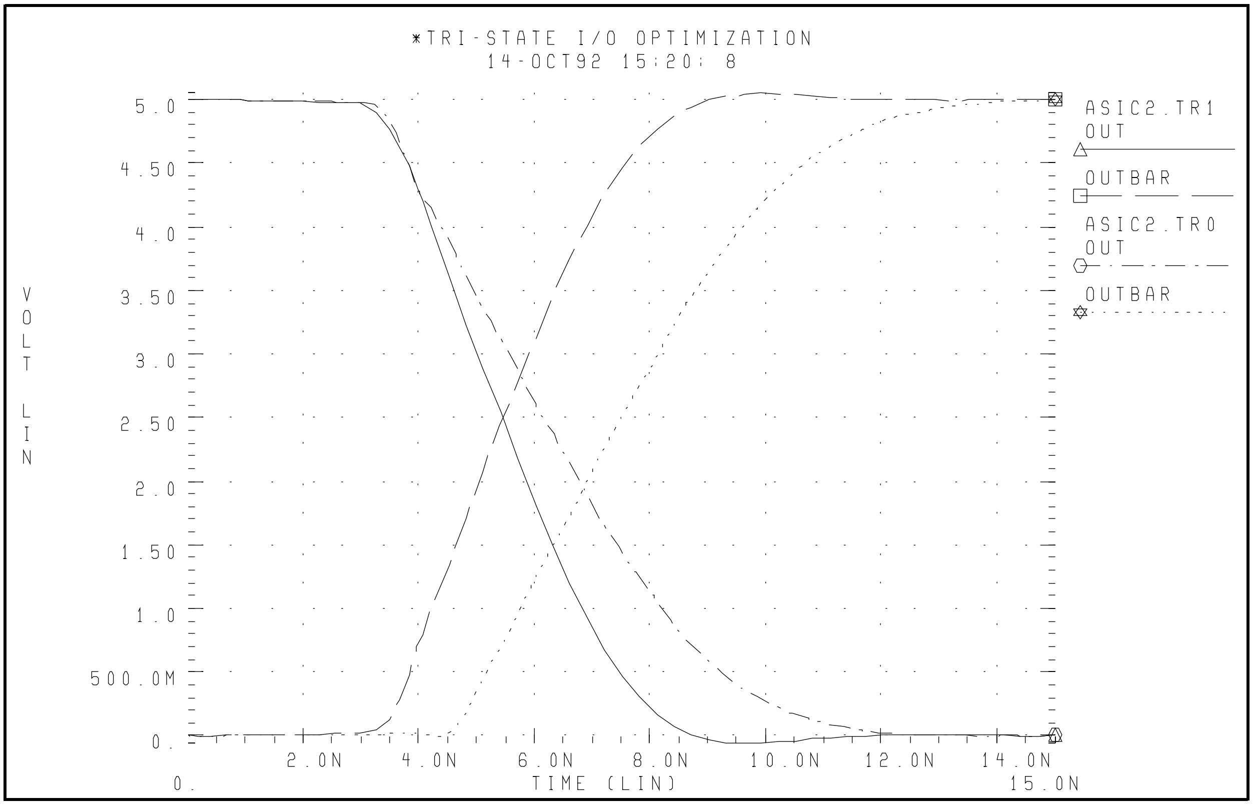


Figure 12-25 Tristate Input/Output Optimization ACIC2B.TR0 = Before Optimization, ACIC2B.TR1 = Optimized Result



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

```
* NET COMMAND AUTOMATICALLY REVERSES THE SIGN OF THE POWER
* SUPPLY CURRENT, FOR NETWORK CALCULATIONS
.NET I(VCE) IBASE ROUT=50 RIN=50
VCE VCE 0 10V
IBASE 0 IIN   AC=1 DC=.1MA
LBB IIN BASE LBB
LCC VCE COLLECT LCC
LEE EMIT 0 LEE
Q1 COLLECT BASE EMIT T2N6604
.MODEL T2N6604 NPN RB=RB BF=100 TF=TF CJE=CBE CJC=CBC
+ RE=RE RC=RC IS=IS
.PARAM
+ LBB= OPT1(100P, 1P, 10N)
+ LCC= OPT1(100P, 1P, 10N)
+ LEE= OPT1(100P, 1P, 10N)
+ TF = OPT1(1N, 5P, 5N)
+ CBE= OPT1(.5P, .1P, 5P)
+ CBC= OPT1(.4P, .1P, 5P)
+ RB= OPT1(10, 1, 300)
+ RE= OPT1(.4, .01, 5)
+ RC= OPT1(10, .1, 100)
+ IS= OPT1(1E-15, 1E-16, 1E-10)
.AC DATA=MEASURED OPTIMIZE=OPT1
+ RESULTS=COMP1,COMP3,COMP5,COMP6,COMP7
+ MODEL=CONVERGE

.MODEL CONVERGE OPT RELIN=1E-4 RELOUT=1E-4 CLOSE=100
+ ITROPT=25
.MEASURE AC COMP1 ERR1 PAR(S11M) S11(M)
.MEASURE AC COMP2 ERR1 PAR(S11P) S11(P) MINVAL=10
.MEASURE AC COMP3 ERR1 PAR(S12M) S12(M)
.MEASURE AC COMP4 ERR1 PAR(S12P) S12(P) MINVAL=10
.MEASURE AC COMP5 ERR1 PAR(S21M) S21(M)
.MEASURE AC COMP6 ERR1 PAR(S21P) S21(P) MINVAL=10
.MEASURE AC COMP7 ERR1 PAR(S22M) S22(M)

.AC DATA=MEASURED
.PRINT PAR(S11M) S11(M) PAR(S11P) S11(P)
.PRINT PAR(S12M) S12(M) PAR(S12P) S12(P)
.PRINT PAR(S21M) S21(M) PAR(S21P) S21(P)
.PRINT PAR(S22M) S22(M) PAR(S22P) S22(P)
```

```
.DATA MEASURED
FREQ   S11M   S11P   S21M    S21P   S12M   S12P S22M   S22P
100ME  .6    -52     19.75   148    .02     65    .87   - 21
200ME  .56   -95     15.30   127    .032    49    .69   - 33
500ME  .56   -149     7.69    97    .044    41    .45   - 41
1000ME .58   -174     4.07    77    .061    42    .39   - 47
2000ME .61    159     2.03    50    .095    40    .39   - 70
.ENDDATA

.PARAM FREQ=100ME S11M=0, S11P=0, S21M=0, S21P=0, S12M=0,
+ S12P=0, S22M=0, S22P=0
.END
```

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
```

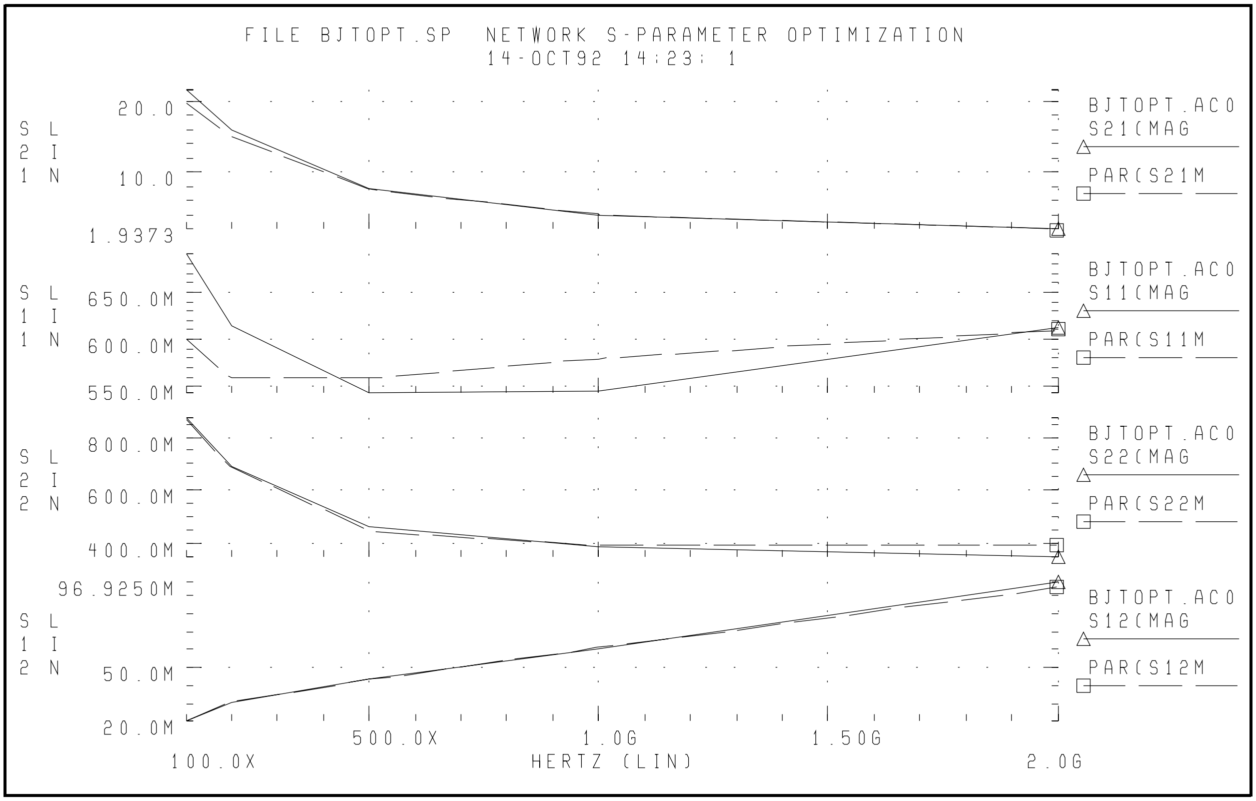
The maximum number of iterations (25) was exceeded. However, the results probably are accurate. Increase ITROPT accordingly.

```
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 12-26   BJT-S Parameter Optimization



BJT Model DC Optimization

The goal is to match forward and reverse Gummel plots, obtained from a HP4145 semiconductor analyzer, using the True-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

```
* FILE OPT_BJT.SP BJT OPTIMIZATION T2N3947
* OPTIMIZE THE DC FORWARD AND REVERSE CHARACTERISTICS
+ FROM A GUMMEL PLOT
* ALL DC GUMMEL-POON DC PARAMETERS EXCEPT BASE
+ RESISTANCE AND EARLY VOLTAGES OPTIMIZED
*

$. .TIGHTEN THE SIMULATOR CONVERGENCE PROPERTIES
.OPTION NOMOD INGOLD=2 NOPAGE VNTOL=1E-10 POST
+ 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
```

Model and Initial Estimates

```
.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)
```

Optimization Results OPT1

```
RESIDUAL SUM OF SQUARES = 2.196240E-02
```

Optimized Parameters OPT1

```
+ %NORM-SEN %CHANGE
.PARAM BF = 1.4603E+02 $ 2.7540E+00 -7.3185E-07
.PARAM IS = 2.8814E-15 $ 3.7307E+00 -5.0101E-07
.PARAM NF = 9.9490E-01 $ 9.1532E+01 -1.0130E-08
.PARAM IKF = 8.4949E-02 $ 1.3782E-02 -8.8082E-08
.PARAM RE = 6.2358E-01 $ 8.6337E-02 -3.7665E-08
.PARAM ISE = 5.0569E-16 $ 1.0221E-01 -3.1041E-05
.PARAM NE = 1.3489E+00 $ 1.7806E+00 2.1942E-07
```

Optimization Results OPT2

```
RESIDUAL SUM OF SQUARES = 1.82776
```

Optimized Parameters OPT2

```
* %NORM-SEN %CHANGE
.PARAM BR = 1.0000E+01 $ 1.1939E-01 1.7678E+00
.PARAM NR = 9.8185E-01 $ 1.4880E+01 -1.1685E-03
.PARAM IKR = 7.3896E-01 $ 1.2111E-03 -3.5325E+01
.PARAM ISC = 1.8639E-12 $ 6.6144E+00 -5.2159E-03
.PARAM NC = 1.2800E+00 $ 7.8385E+01 1.6202E-03
```

Figure 12-27 BJT Optimization Forward Gummel Plots

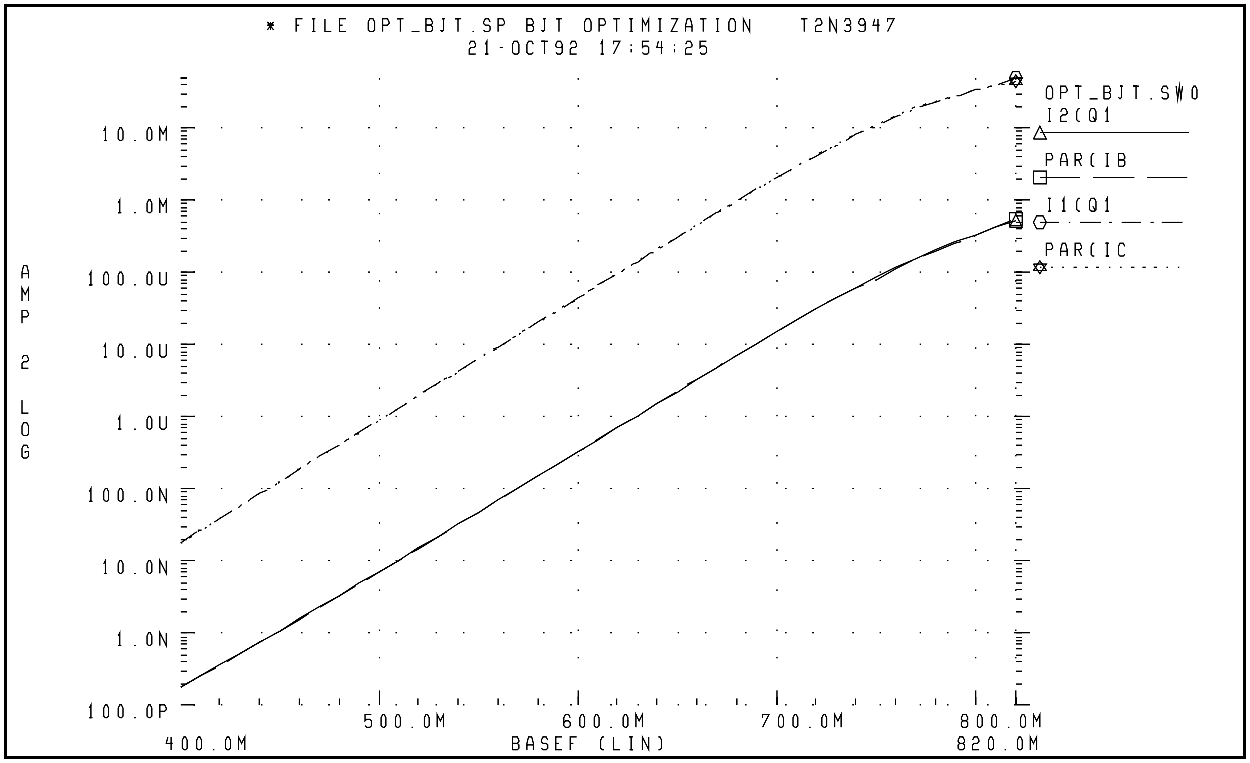
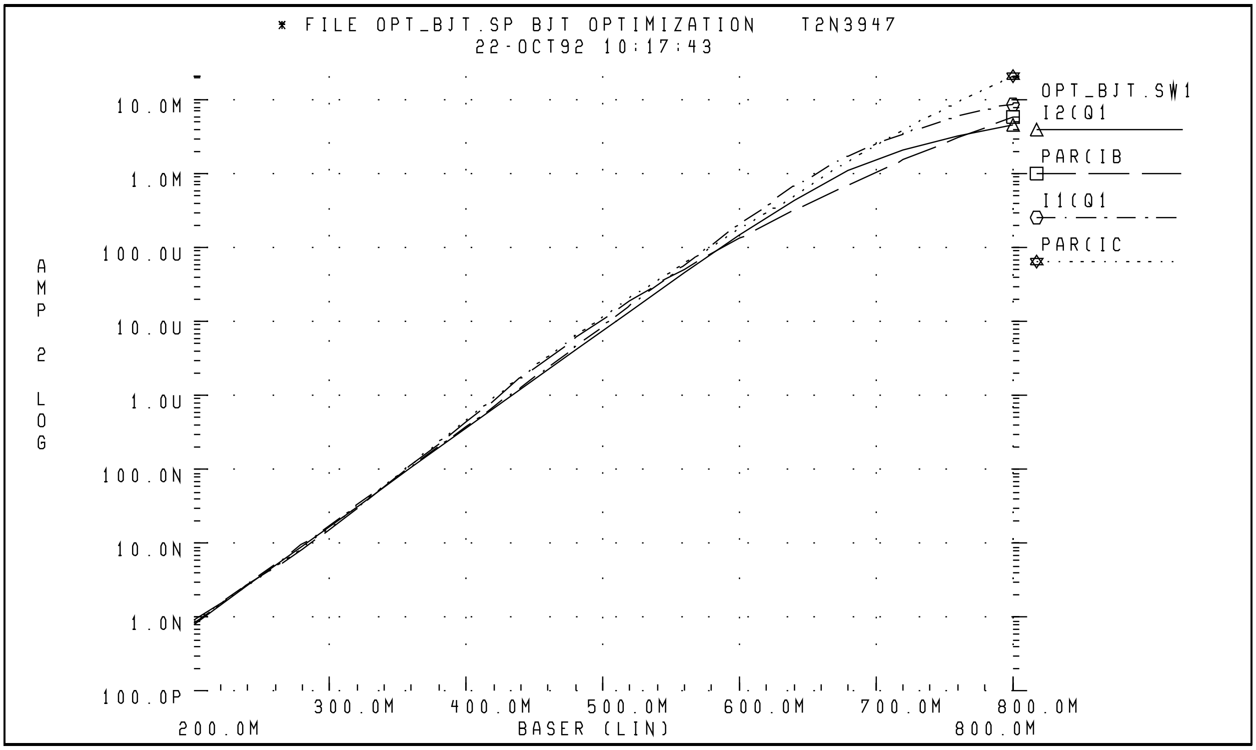


Figure 12-28   BJT Optimization Reverse Gummel Plots



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 True-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

```
*FILE JOPT.SP JFET OPTIMIZATION
.OPTION ACCT NOMOD POST
+ RELI=2E-4 RELV=2E-4
VG GATE 0 XVGS
VD DRAIN 0 XVDS
J1 DRAIN GATE 0 JFETN1

.MODEL JFETN1 NJF LEVEL=3 CAPOP=1 SAT=3
+ NG=1
+ CGS=1P CGD=1P RG=1
+ VTO=VTO BETA=BETA LAMBDA=LAMBDA
+ RS=RDS RD=RDS IS=1E-15 ALPHA=ALPHA
+ UCRIT=UCRIT SATEXP=SATEXP
+ GAMDS=GAMDS VGEXP=VGEXP

.PARAM
+ VTO=OPT1(-.8,-4,0)
+ VGEXP=OPT1(2,1,3.5)
+ GAMDS=OPT1(0,-.5,0)
+ BETA= OPT1(6E-3, 1E-5,9E-2)
+ LAMBDA=OPT1(30M,1E-7,5E-1)
+ RDS=OPT1(1,.001,40)
+ ALPHA=OPT1(2,1,3)
+ UCRIT=OPT1(.1,.001,1)
+ SATEXP=OPT1(1,.5,3)

.DC DATA=DESIRED OPTIMIZE=OPT1 RESULTS=IDSERR MODEL=CONV
.MODEL CONV OPT RELIN=1E-4 RELOUT=1E-4 CLOSE=100 ITROPT=25
.MEASURE DC IDSERR ERR1 PAR(XIDS) I(J1) MINVAL=20M
+ IGNORE=1M
.DC DATA=DESIRED
.GRAPH PAR(XIDS) I(J1)
.MODEL GRAPH PLOT MONO=1
.PRINT PAR(XVGS) PAR(XIDS) I(J1)

.DATA DESIRED MERGE
+ FILE=JDC.DAT XVDS=1 XVGS=2 XIDS=3
.ENDDATA
.END
```

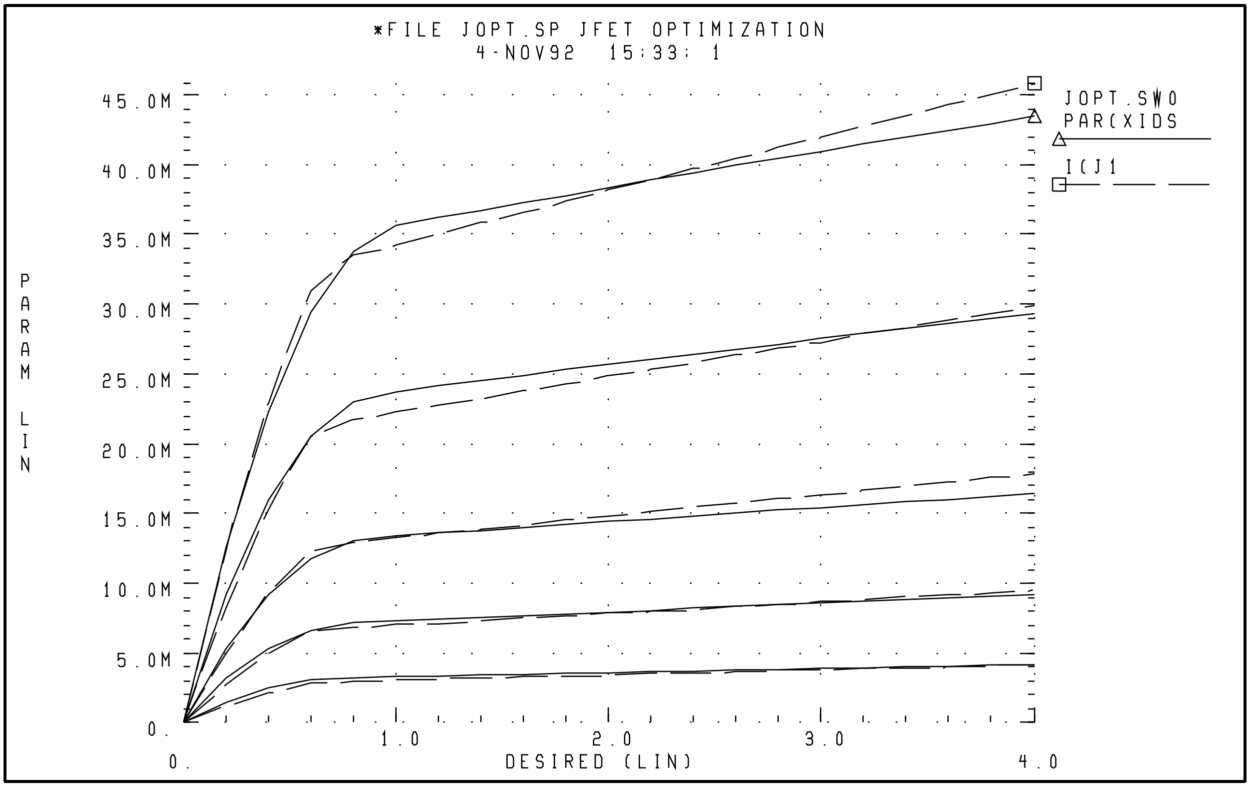
Optimization Results

```
RESIDUAL SUM OF SQUARES = 7.582202E-02
```

Optimized Parameters Opt1

```
*%NORM-SEN%CHANGE
.PARAM VTO= -1.1067   $   64.6110     43.9224M
.PARAM VGEXP= 2.9475    $   13.2024    219.4709M
.PARAM GAMDS= 0.        $    0.          0.
.PARAM BETA = 11.8701M  $   17.2347    136.8216M
.PARAM LAMBDA= 138.9821M $    2.2766      -1.5754
.PARAM RDS= 928.3216M $  704.3204M   464.0863M
.PARAM ALPHA= 2.2914    $  728.7492M    168.4004M
.PARAM UCRIT= 1.0000M   $   18.2438M   -125.0856
.PARAM SATEXP= 1.4211    $    1.2241       2.2218
```

Figure 12-29 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 algebraically, use a voltage value, and minimize the resulting voltage.
- The transistor area measure statement uses MINVAL, which assigns less weight to the area minimization.
- Optimizes the bias voltage.

MOS Op-amp Optimization Input Netlist File

```
AMPOPT.SP MOS OPERATIONAL AMPLIFIER OPTIMIZATION
.OPTION RELV=1E-3 RELVAR=.01 NOMOD ACCT POST
.PARAM VDD=5   VREF='VDD/2'
VDD VSUPPLY 0 VDD
VIN+    VIN+     0 PWL(0 ,'VREF-10M'   10NS 'VREF+10M' )
VINBAR+ VINBAR+  0 PWL(0 ,'VREF+10M'   10NS 'VREF-10M' )
VIN- VIN-   0 VREF
VBIAS VBIAS 0 BIAS
.GLOBAL VSUPPLY VBIAS
XRISE VIN+ VIN- VOUTR AMP
```

```
CLOADR VOUTR 0 .4P
XFALL VINBAR+ VIN- VOUTF AMP
CLOADF VOUTF 0 .4P

.MACRO AMP VIN+ VIN- VOUT
M1 2     VIN-   3       3       MOSN W=WM1 L=LM
M2 4     VIN+   3       3       MOSN W=WM1 L=LM
M3 2     2      VSUPPLY VSUPPLY MOSP W=WM1 L=LM
M4 4     2      VSUPPLY VSUPPLY MOSP W=WM1 L=LM
M5 VOUT  VBIAS  0       0       MOSN W=WM5 L=LM
M6 VOUT  4      VSUPPLY VSUPPLY MOSP W=WM6 L=LM
M7 3     VBIAS    0        0    MOSN W=WM7 L=LM
.ENDS

.PARAM AREA=’4*WM1*LM + WM5*LM + WM6*LM + WM7*LM’
VX 1000 0 AREA
RX 1000 0 1K

.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 MODEL=OPT
.MODEL OPT OPT CLOSE=100

.TRAN 2N 150N
.MEASURE DELAYR TRIG AT=0 TARG V(VOUTR) VAL=2.5 RISE=1
+ GOAL=100NS

.MEASURE DELAYF TRIG AT=0 TARG V(VOUTF) VAL=2.5 FALL=1
+ GOAL=100NS

.MEASURE TOT_POWER AVG POWER GOAL=10MW
.MEASURE AREA MIN PAR(AREA) GOAL=1E-9 MINVAL=100N
.PRINT V(VIN+) V(VOUTR) V(VOUTF)
.END
```

Optimization Results

```
RESIDUAL SUM OF SQUARES  = 4.654377E-04
NORM OF THE GRADIENT    = 6.782920E-02
```

Optimized Parameters Opt1

```
* %NORM-SEN%CHANGE

.PARAM WM1          =  47.9629U  $    1.6524  -762.3661M
.PARAM WM5          =  66.8831U  $   10.1048    23.4480M
.PARAM WM6          = 127.1928U  $   12.7991    22.7612M
.PARAM WM7          = 115.8941U  $    9.6104  -246.4540M
.PARAM LM           =   6.2588U  $   20.3279  -101.4044M
.PARAM BIAS        =   2.7180   $   45.5053    5.6001M

*** OPTIMIZE RESULTS MEASURE NAMES AND VALUES
* DELAYR         = 100.4231N
* DELAYF         =  99.5059N
* TOT_POWER      =  10.0131M
* AREA          =   3.1408N
```

Figure 12-30 CMOS Op-amp

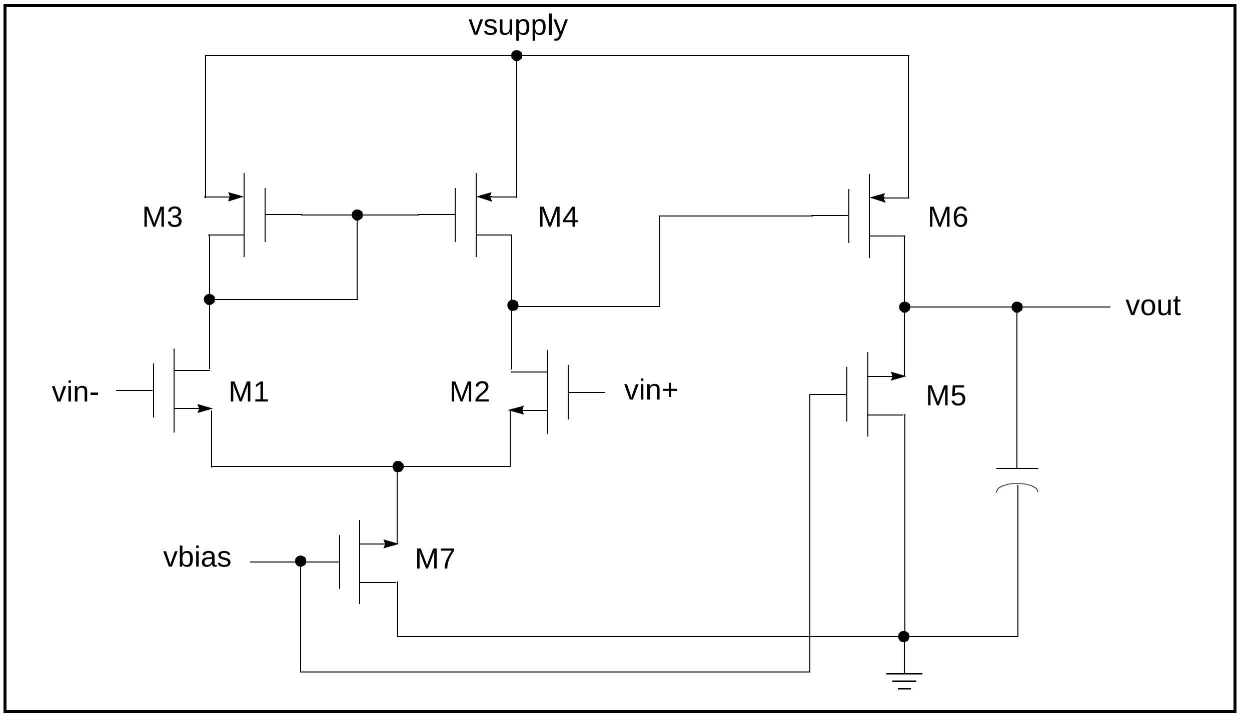
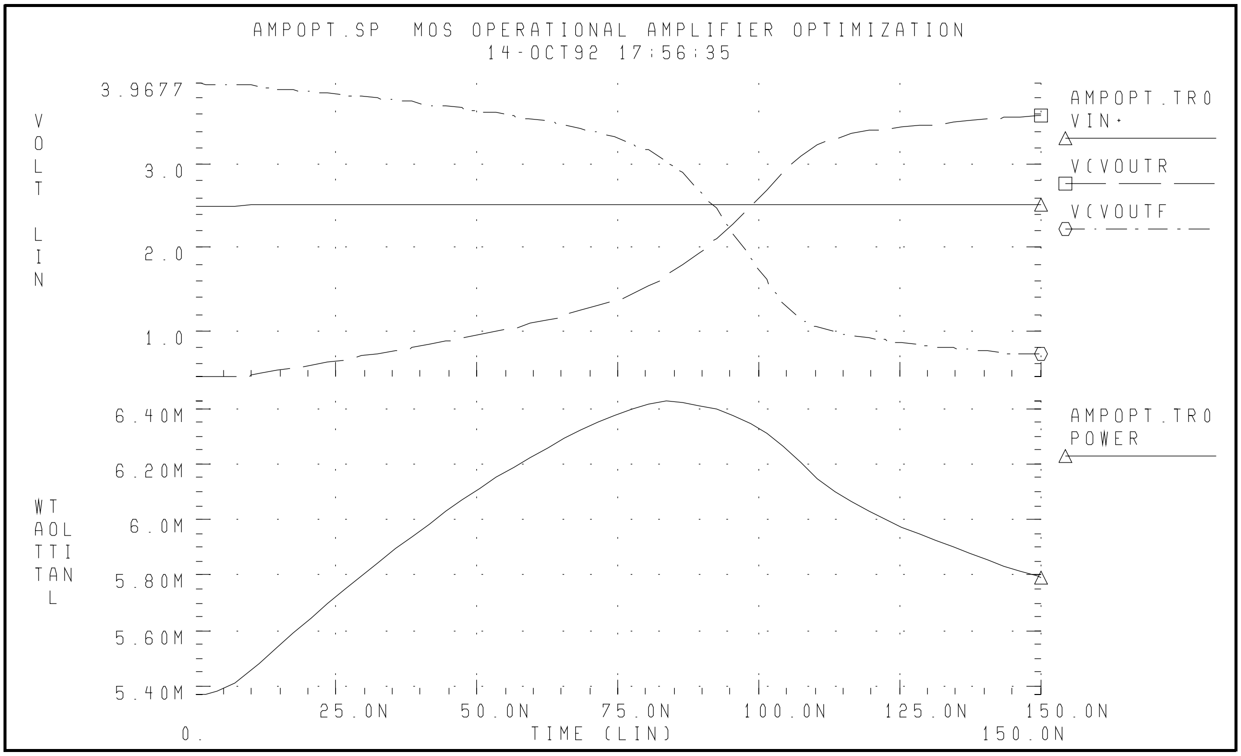


Figure 12-31   Operational Amplifier Optimization



13

Running Demonstration Files

This chapter contains examples of basic file construction techniques, advanced features, and simulation tricks. It also lists and describes several Synopsys HSPICE input files.

This chapter explains the following topics:

Using the Demo Directory Tree

Demonstration Input Files on page 13-24 lists demonstration files, which are designed as good training examples. Most HSPICE distributions include these examples in the *demo* directory tree, where *$installdir* is the installation directory environment variable:

Table 13-1 Demo Directory Tree

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
	/fft	Fourier analysis
	/filters	filters
	/mag	transformers, magnetic core components
	/mos	MOS components
	/rad	radiation effects (photocurrent)
	/sources	dependent and independent sources
	/tline	filters and transmission lines

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 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 uses two calls to ONEBIT. Independent piecewise linear voltage sources provide the input stimuli. The *R* repeat function creates complex waveforms.

Figure 13-1   One-bit Adder sub-circuit

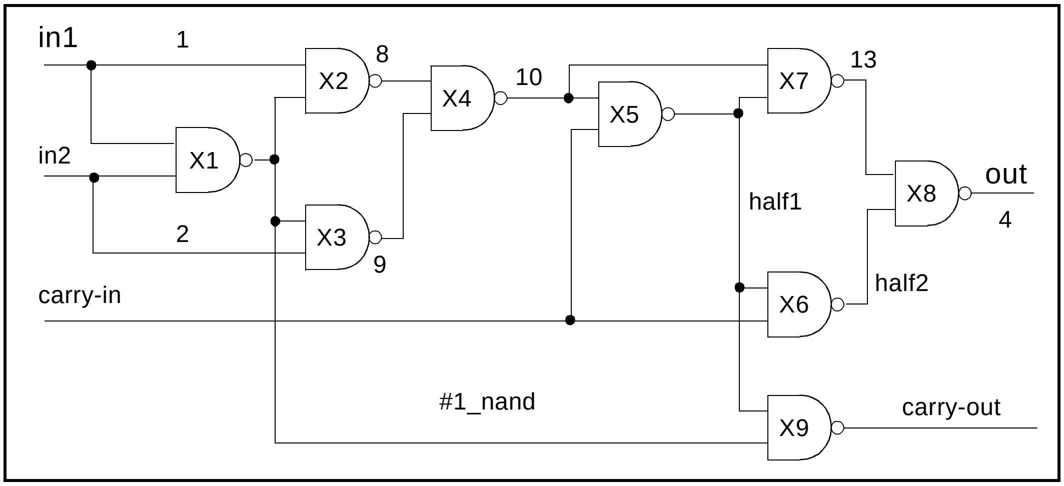


Figure 13-2   Two-bit Adder Circuit

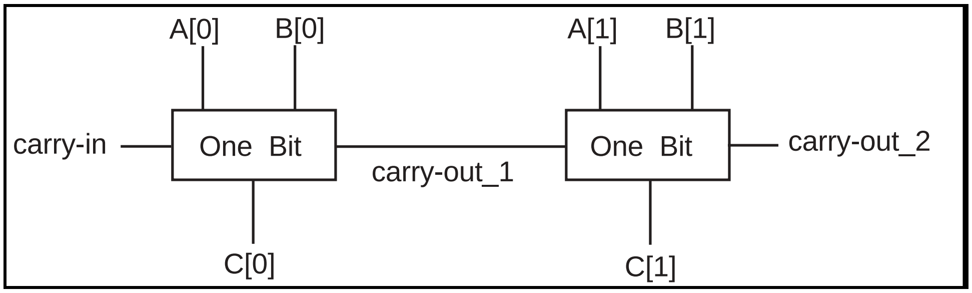
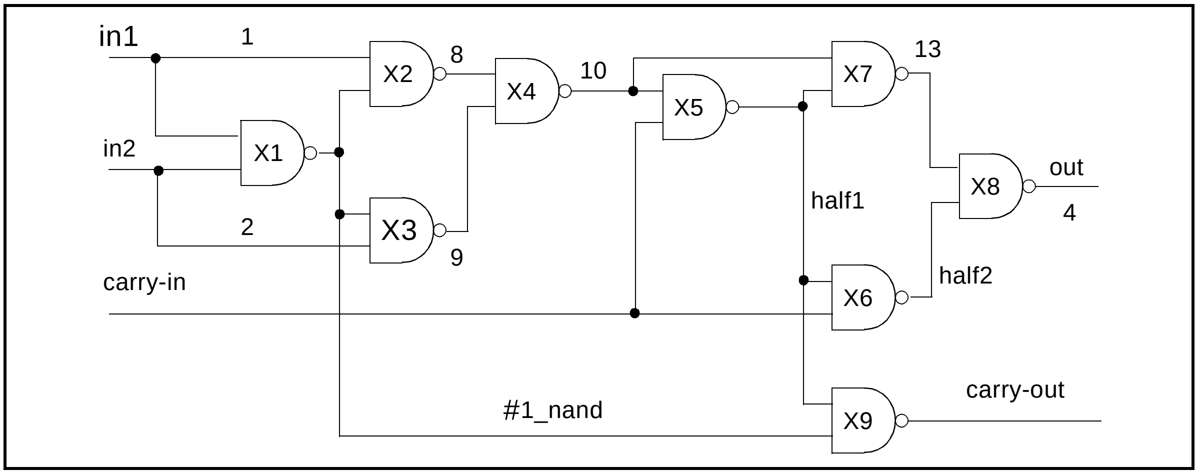


Figure 13-3   1-bit NAND Gate Binary Adder



MOS Two-Bit Adder Input File

```
*FILE: MOS2BIT.SP - ADDER - 2 BIT ALL-NAND-GATE BINARY ADDER
.OPTION ACCT NOMOD FAST autostop scale=1u gmindc=100n
.param lmin=1.25 hi=2.8v lo=.4v vdd=4.5
.global vdd
.TRAN .5NS 60NS
.graph TRAN V(c[0]) V(carry-out_1) V(c[1]) V(carry-out_2)
+ par('V(carry-in)/6 + 1.5')
+ par('V(a[0])/6     + 2.0')
+ par('V(b[0])/6     + 2.5') (0,5)
.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
sub-circuit 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

* switch model equivalent of the NAND. Gives a 10 times
* speedup over the MOS version.
.subckt NANDx in1 in2 out   wp=10 wn=5
   G1 out vdd vdd in1 LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
   G2 out vdd vdd in2 LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
   G3 out mid in1 gnd LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
   G4 mid gnd in2 gnd LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
   cout out gnd 300f
.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
```

Stimuli

```
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
```

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
.END
```

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, 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 13-2   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 to evaluate the voltage variable capacitors, during a DC sweep.
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 13-3 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 13-4 MOS IDS Plot

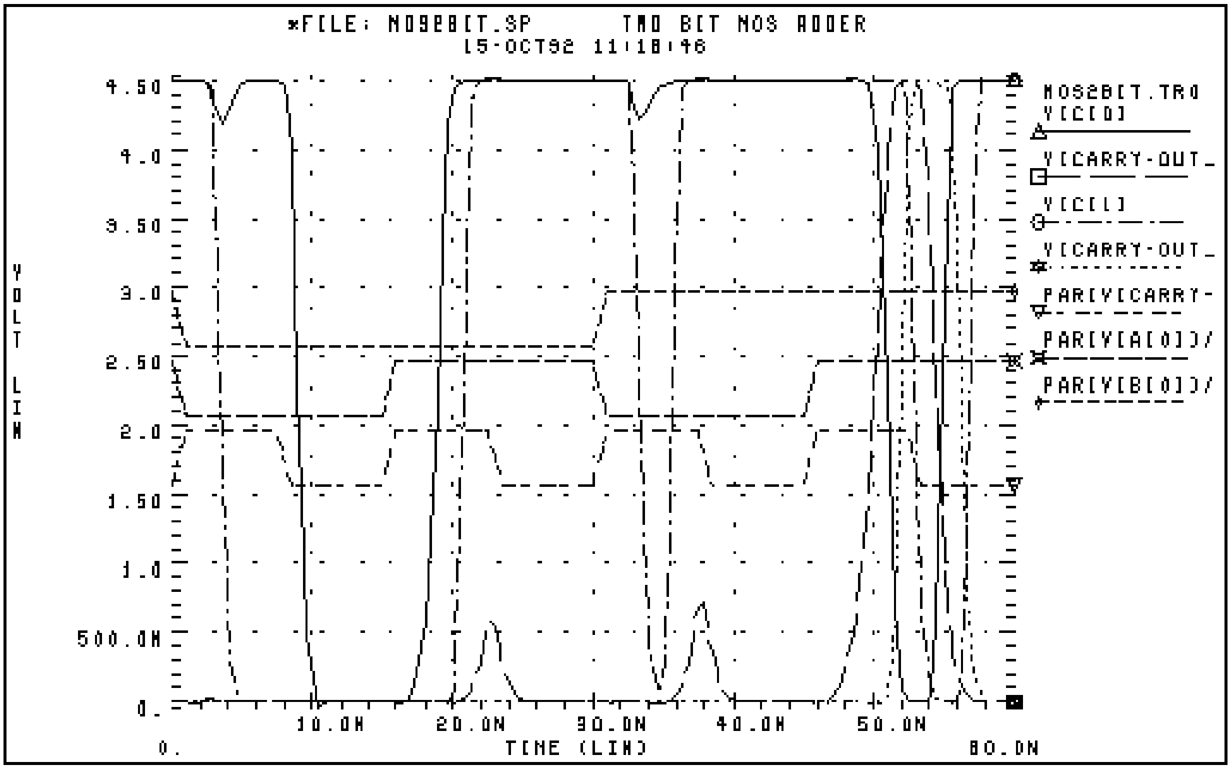


Figure 13-5 MOS VGS Plot

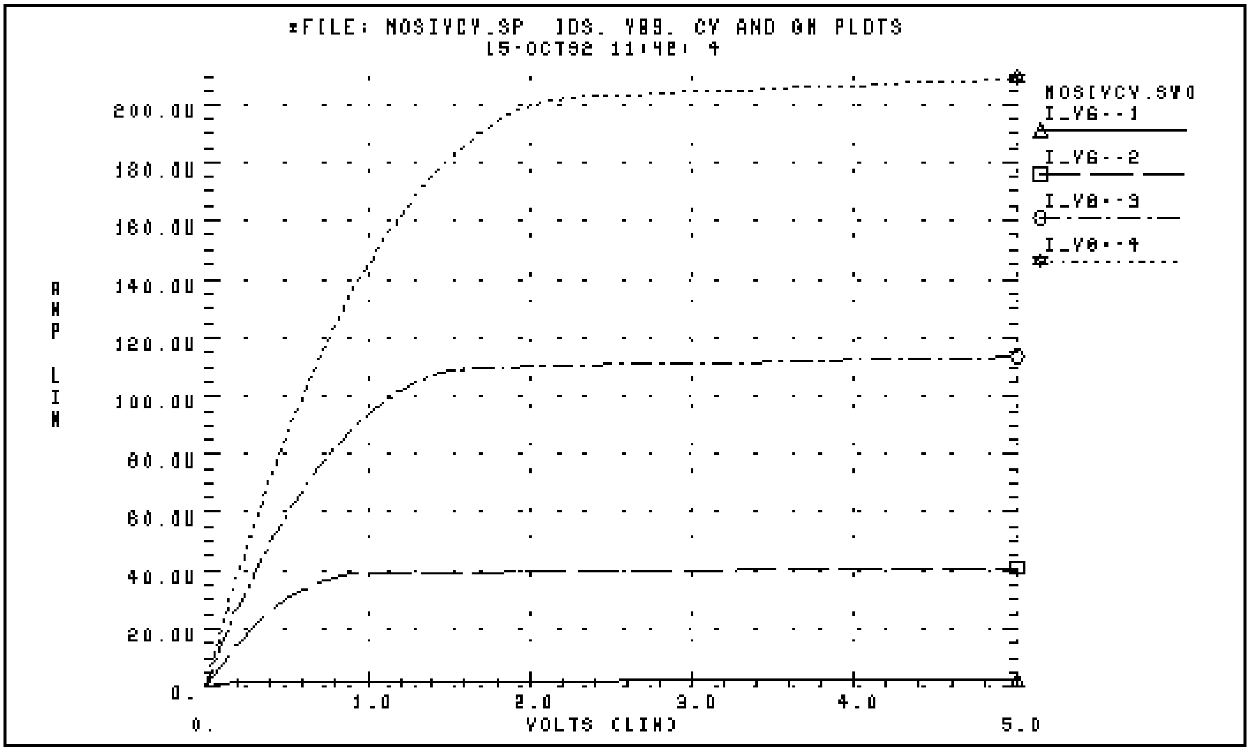


Figure 13-6 MOS GM Plot

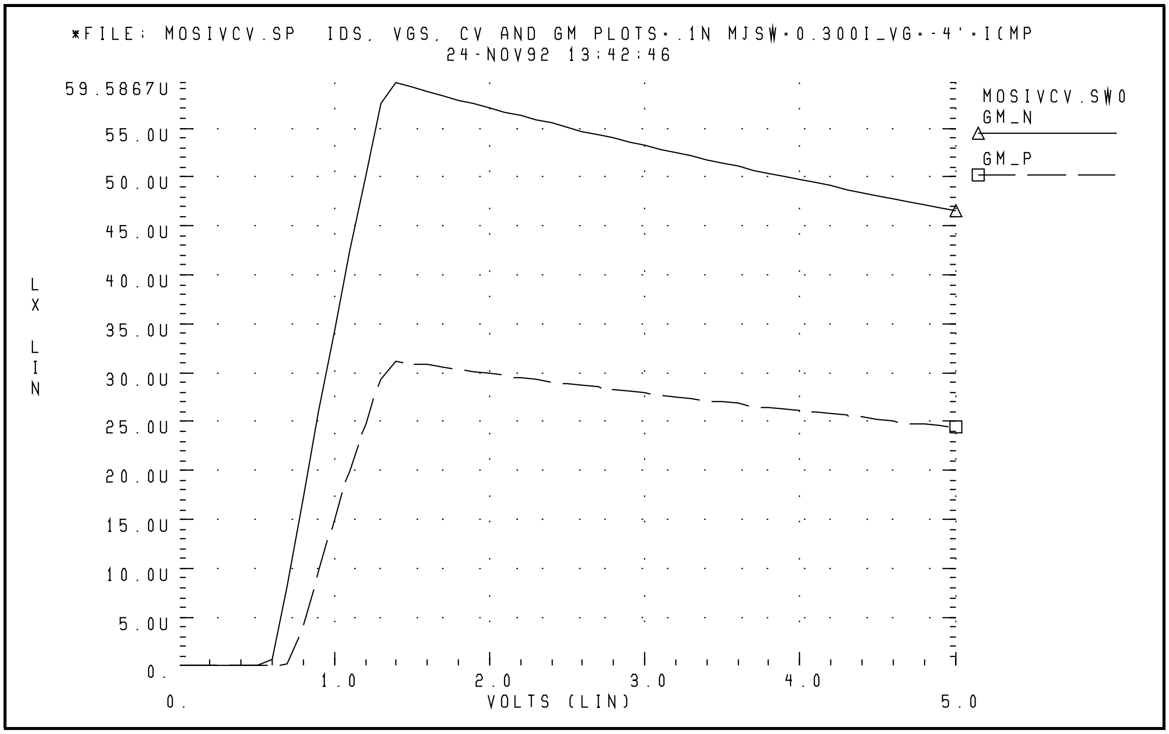
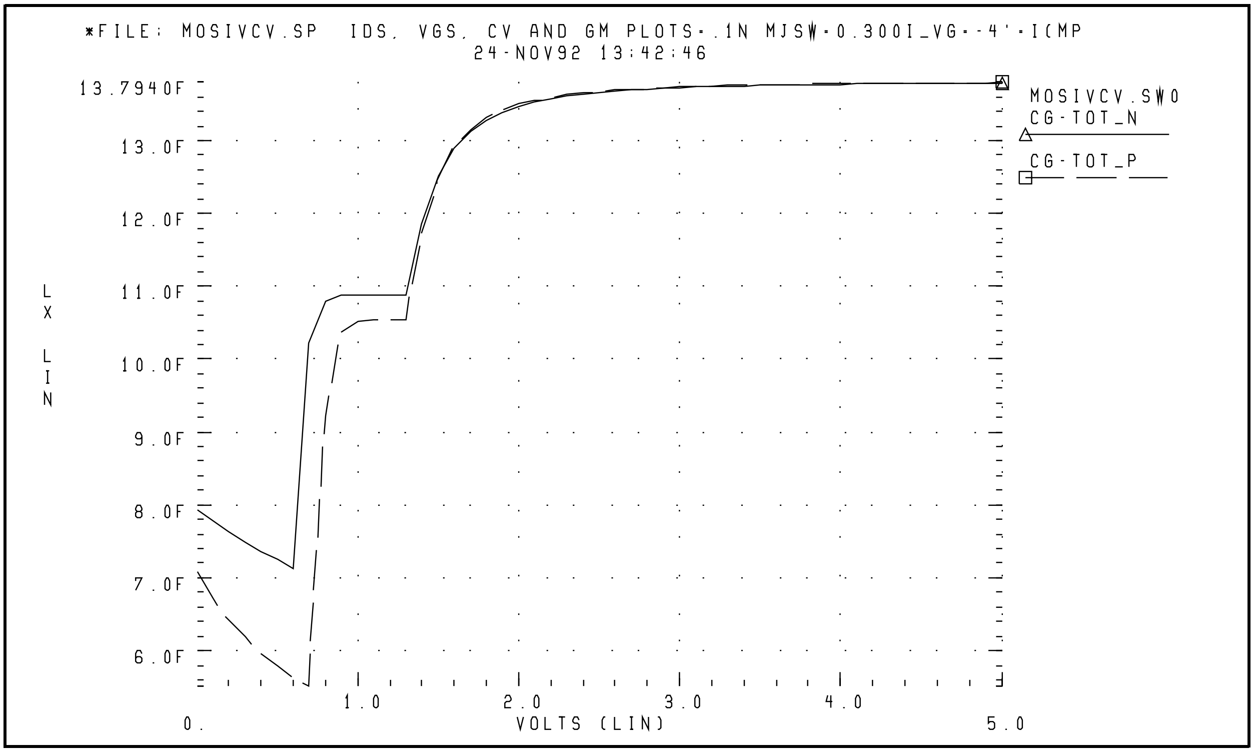


Figure 13-7   MOS C-V Plot



MOS I-V and C-V Plot Example Input File

```
*FILE: MOSIVCV.SP IDS, VGS, CV AND GM PLOTS
.OPTION SCALE=1U DCCAP
.DC VDDN 0 5.0 .1 $VBBN 0 -3 -3      sweep supplies
.PARAM ww=8 LL=2
$ ids-vds curves
.GRAPH ’I_VG=1’ =I(MN1) ’I_VG=2’ =I(MN2) ’I_VG=3’ =I(MN3)
+ ’I_VG=4’ =I(MN4)
.GRAPH ’I_VG=-1’=I(MP1) ’I_VG=-2’=I(MP2) ’I_VG=-3’=I(MP3)
+ ’I_VG=-4’=I(MP4)
$ ids-VGs curves
.GRAPH ’I_VD=.5’=I(MN6) ’I_VD=-.5’=I(MP6)
$ gate caps (cgs+cgd+cgb)
.GRAPH ’CG-TOT_N’=LX18(MN6) ’CG-TOT_P’= LX18(MP6)
$ gm
.GRAPH ’GM_N’=LX7(MN6) ’GM_P’=LX7(MP6)
VDDN vdd_n gnd 5.0
VBBN vbb_n gnd 0.0
```

```
EPD vdd_p gnd vdd_n gnd -1   $ reflect vdd for P devices
EPB vbb_p gnd vbb_n gnd -1   $ reflect vbb for P devices
V1 vg1 gnd 1
V2 vg2 gnd 2
V3 vg3 gnd 3
V4 vg4 gnd 4
V5 vddlow_n gnd .5
V-1 vg-1 gnd -1
V-2 vg-2 gnd -2
V-3 vg-3 gnd -3
V-4 vg-4 gnd -4
V-5 vddlow_p gnd -.5
MN1 vdd_n vg1 gnd vbb_n N W=ww L=LL
MN2 vdd_n vg2 gnd vbb_n N W=ww L=LL
MN3 vdd_n vg3 gnd vbb_n N W=ww L=LL
MN4 vdd_n vg4 gnd vbb_n N W=ww L=LL
MP1 gnd vg-1 vdd_p vbb_p P W=ww L=LL
MP2 gnd vg-2 vdd_p vbb_p P W=ww L=LL
MP3 gnd vg-3 vdd_p vbb_p P W=ww L=LL
MP4 gnd vg-4 vdd_p vbb_p P W=ww L=LL
MN6 vddlow_n vdd_n gnd vbb_n N W=ww L=LL
MP6 gnd vdd_p vddlow_p vbb_p P W=ww L=LL
.MODEL  N NMOS LEVEL=3  VTO=0.7 UO=500  KAPPA=.25  ETA=.01
+ THETA=.04 VMAX=2E5 NSUB=9E16 TOX=400  KP=30U  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=.67
+ 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
.END
```

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 the performance of the system. The demonstration file, $*installdir*/demo/hspice/apps/asic1.sp 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 strategy is to:

- Create a five-lump transmission line model, for the copper wire.
- Create single lumped models, for leadframe loads.

Figure 13-8   Noise Bounce

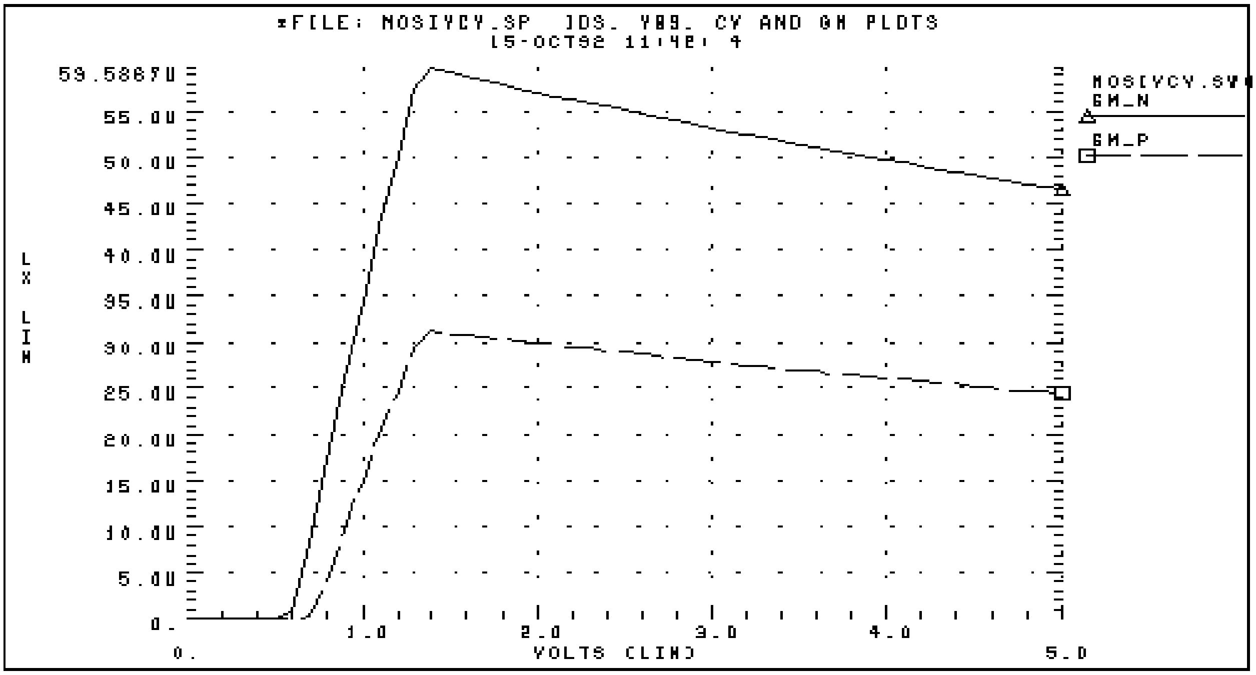


Figure 13-9   Asic1.sp Demo Local Supply Voltage

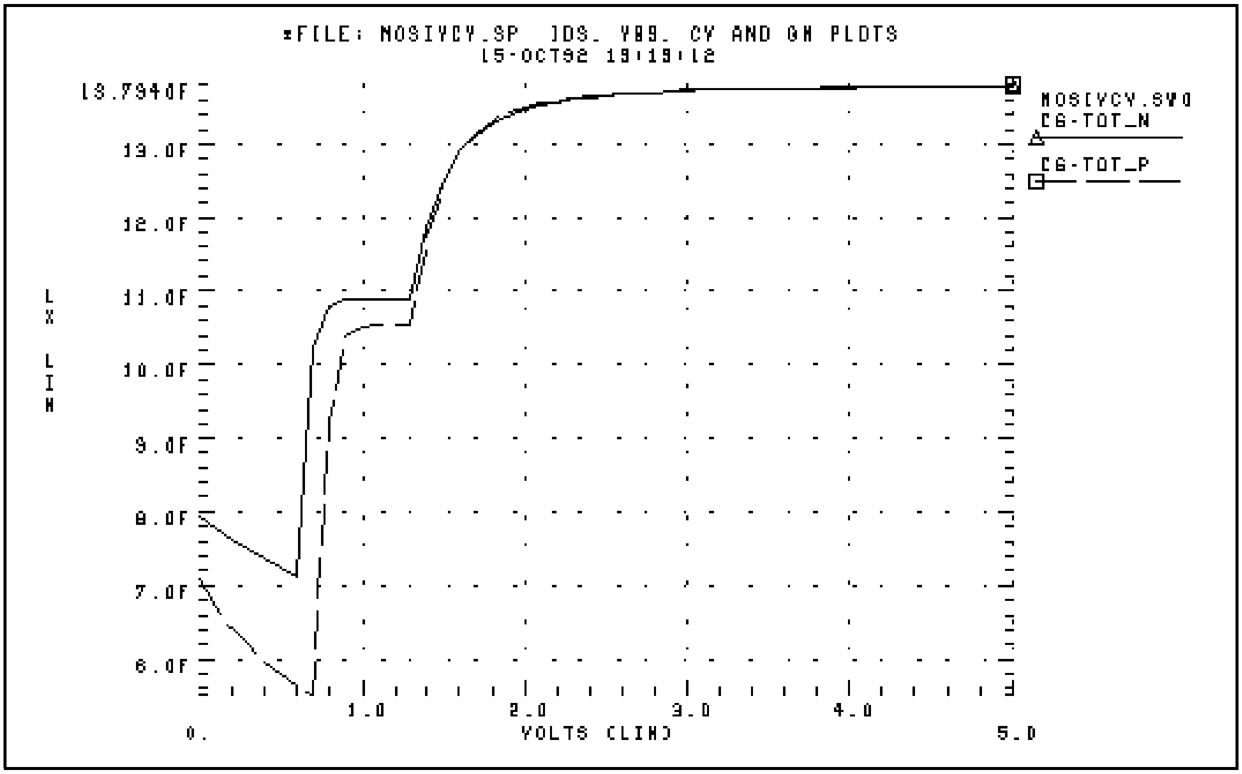


Figure 13-10 Asic1.sp Demo Local Supply Current

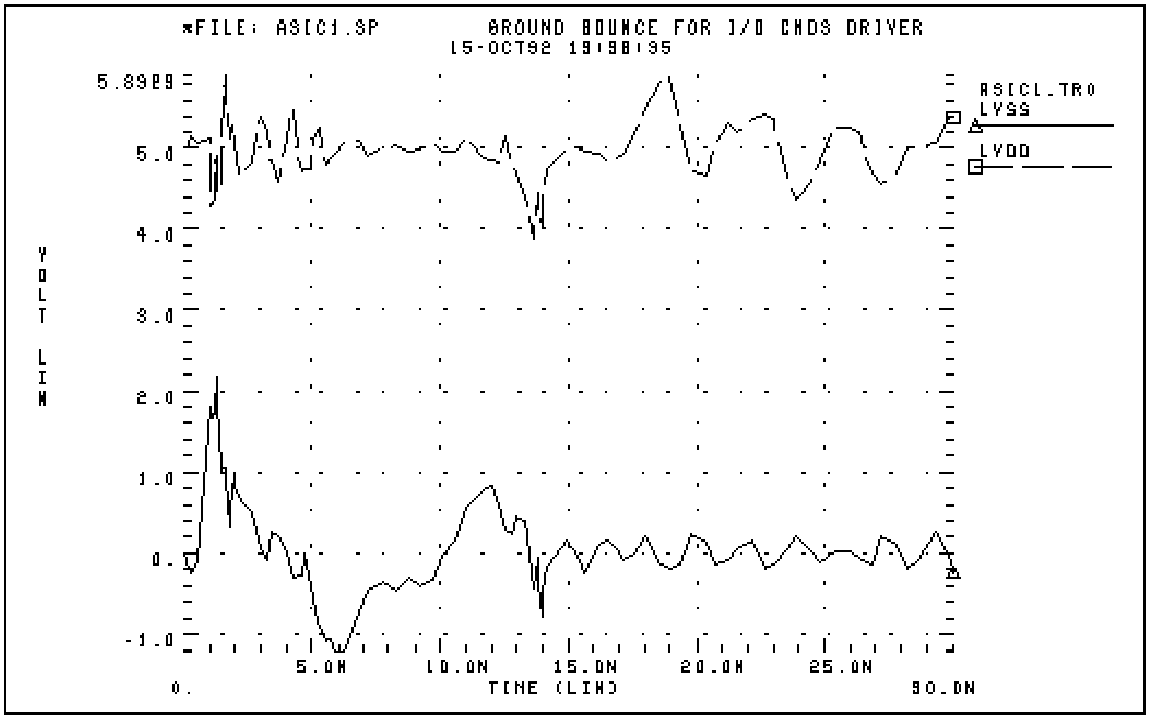
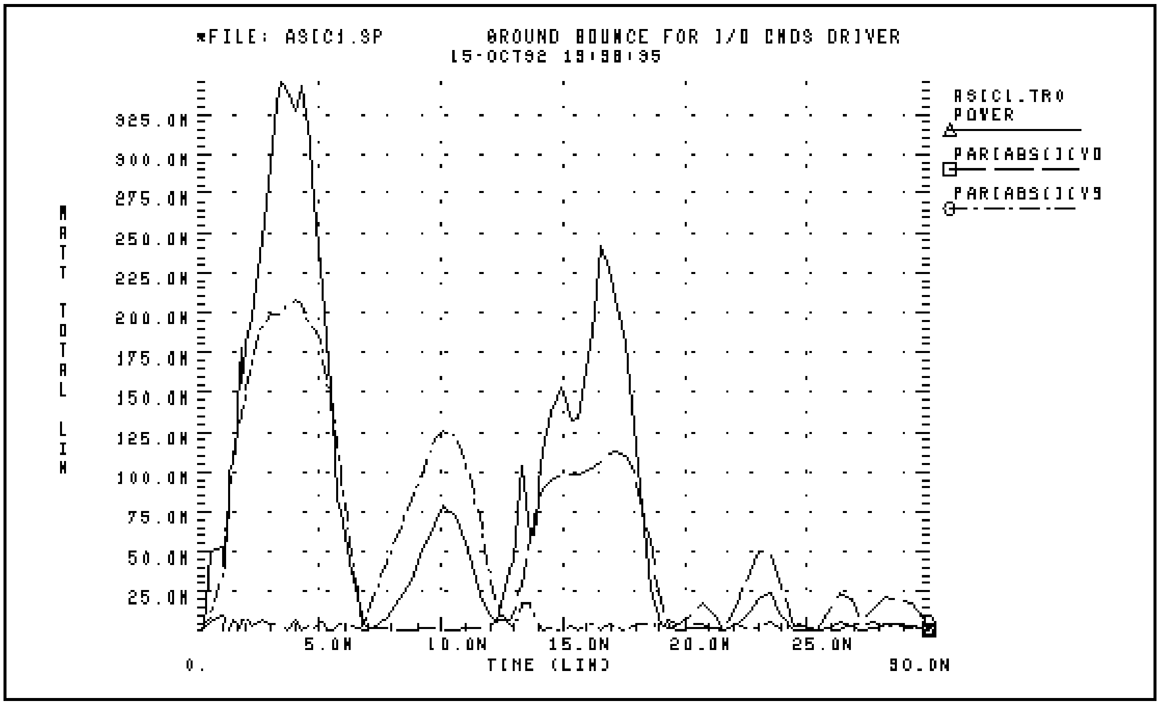


Figure 13-11 Asic1.sp Demo Input and Output Signals



CMOS Output Driver Example Input File

```
* FILE: ASIC1.SP
* SIMULATE AN OUTPUT DRIVER DRIVING 6 INCHES OF 6MIL PRINTED
*    CIRCUIT BOARD COPPER WITH 25PF OF LOAD CAPACITANCE
* MEASURE PEAK TO PEAK GROUND VOLTAGE
* MEASURE MAXIMUM GROUND CURRENT
* MEASURE MAXIMUM SUPPLY CURRENT
GROUND BOUNCE FOR I/O CMOS DRIVER 1200/1.2 & 800/1.2 MICRONS
.OPTION POST=2 RELVAR=.05
.TRAN .25N 30N
.MEASURE IVDD_MAX MAX PAR(’ABS(I(VD))’)
.MEASURE IVSS_MAX MAX PAR(’ABS(I(VS))’)
.MEASURE PEAK_GNDV PP V(LVSS)
.MEASURE PEAK_IVD   PP PAR(’ ABS(I(VD)*V(VDD,OUT)) ’)
.MEASURE PEAK_IVS   PP PAR(’ ABS(I(VS)*V(VSS,OUT)) ’)
.MEASURE RMS_POWER RMS POWER
.MEASURE FALL_TIME TRIG V(IN) RISE=1 VAL=2.5V
+                   TARG V(OUT) FALL=1 VAL=2.5V
.MEASURE RISE_TIME TRIG V(IN) FALL=1 VAL=2.5V
+                   TARG V(OUT) RISE=1 VAL=2.5V
.MEASURE TLINE_DLY TRIG V(OUT) RISE=1 VAL=2.5V
+                   TARG V(OUT2) RISE=1 VAL=2.5V
```

Input Signals

```
VIN IN LGND PWL(0N 0V, 2N 5V, 12N 5, 14N 0)
* OUTPUT DRIVER
MP1 LOUT IN LVDD LVDD P W=1400U L=1.2U
MN1 LOUT IN LVSS LVSS N W=800U L=1.2U
xout LOUT OUT LEADFRAME
*POWER AND GROUND LINE PARASITICS
Vd VDD GND 5V
xdd vdd lvdd leadframe
Vs VSS gnd 0v
xss vss lvss leadframe
*OUTPUT LOADING – 3 INCH FR-4 PC BOARD + 5PF LOAD +
*3 INCH FR-4 + 5PF LOAD
XLOAD1 OUT OUT1 GND LUMP5 LEN=3 WID=.006
CLOAD1 OUT1 GND 5PF
XLOAD2 OUT1 OUT2 GND LUMP5 LEN=3 WID=.006
CLOAD2 OUT2 GND 5PF
```

```
.macro leadframe in out
rframe in mid .01
lframe   mid out 10n
cframe   mid gnd .5p
.ends

*Transmission Line Parameter Definitions
.param rho=.6mho/sq cap=.55nf/in**2 ind=60ph/sq
*The 5-lump macro defines a parameterized transmission line
.macro lump5 in out ref len_lump5=1 wid_lump5=.1
.prot
.param reseff='len_lump5*rho/wid_lump5*5'
+       capeff='len_lump5*wid_lump5*cap/5'
+       indeff='len_lump5*ind/wid_lump5*5'
r1 in 1   reseff
c1   1 ref capeff
l1   1 2   indeff
r2   2 3   reseff
c2   3 ref capeff
l2   3 4   indeff
r3   4 5   reseff
c3   5 ref capeff
l3   5 6   indeff
r4   6 7   reseff
c4   7 ref capeff
l4   7 8   indeff
r5   8 9   reseff
c5   9 ref capeff
l5   9 out indeff
.unprot
.ends
```

Model Section

```
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500 KAPPA=.25 ETA=.03
+ THETA=.04 VMAX=2E5 NSUB=9E16 TOX=200E-10 GAMMA=1.5 PB=0.6
+ JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10 capop=4
.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 ETA=.03
+ THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=200E-10 GAMMA=.672
+ PB=0.6 JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10 capop=4
.end
IVDD_MAX        = 0.1141        AT= 1.7226E-08
            FROM= 0.0000E+00    TO= 3.0000E-08
IVSS_MAX        = 0.2086        AT= 3.7743E-09
            FROM= 0.0000E+00    TO= 3.0000E-08
```

```
PEAK_GNDV = 3.221       FROM= 0.0000E+00 TO= 3.0000E-08
PEAK_IVD  = 0.2929      FROM= 0.0000E+00 TO= 3.0000E-08
PEAK_IVS  = 0.3968      FROM= 0.0000E+00 TO= 3.0000E-08
RMS_POWER = 0.1233      FROM= 0.0000E+00 TO= 3.0000E-08
FALL_TIME = 1.2366E-09  TARG= 1.9478E-09 TRIG= 7.1121E-10
RISE_TIME = 9.4211E-10  TARG= 1.4116E-08 TRIG= 1.3173E-08
TLINE_DLY = 1.6718E-09 TARG= 1.5787E-08 TRIG= 1.4116E-08
```

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 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 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

```
*FILE OPTTEMP.SP   OPTIMIZE RESISTOR TC1 AND TC2
v-25 1 0 1v
v0   2 0 1v
v+25 3 0 1v
r-25 1 0 rmod dtemp=-25
r0   2 0 rmod dtemp=0
r+25 3 0 rmod dtemp=25
.model rmod R res=1k tc1r=tc1r_opt tc2r=tc2r_opt
```

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
```

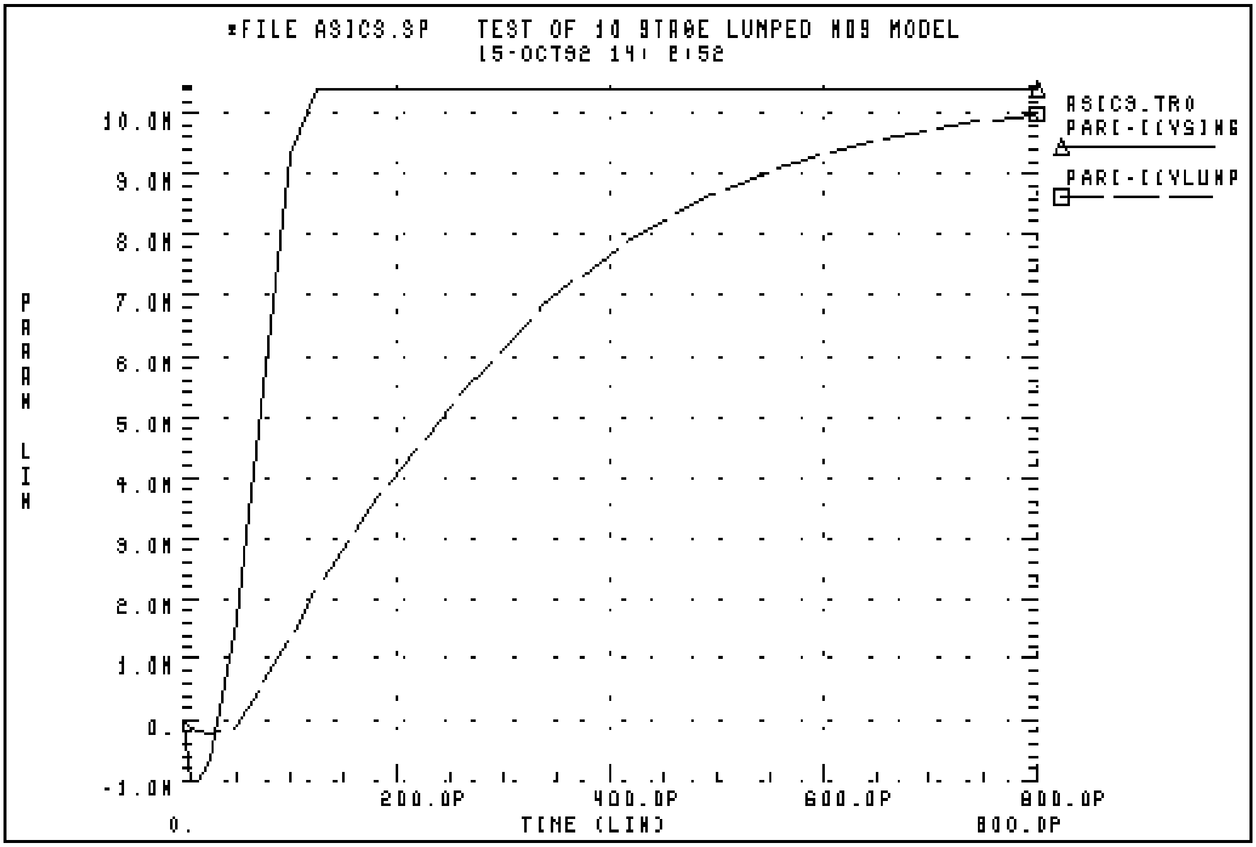
Simulating Electrical Measurements

In this example, HSPICE simulates electrical measurements, which return device characteristics for data sheets. The demonstration file for this example is $*installdir*/demo/hspice/ddl/t2n2222.sp. 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 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 13-12   T2N2222 Optimization



T2N2222 Optimization Example Input File

```
* FILE: T2N2222.SP
** assume beta=200 ft250meg at ic=20ma and vce=20v for 2n2222
.OPTION   nopage autostop search=’ ’
*** ft measurement
* the net command automatically reverses the sign of
* the power supply current, for the network calculations
.NET I(vce) IBASE ROUT=50 RIN=50
VCE C 0 vce
IBASE 0 b   AC=1 DC=ibase
xqft c b 0 t2n2222
.ac dec 10 1 1000meg
.graph s21(m) h21(m)
.measure ’phase @h21=0db’ WHEN h21(db)=0
.measure ’h21_max’ max h21(m)
.measure ’phase @h21=0deg’ FIND h21(p) WHEN h21(db)=0
.param ibase=1e-4 vce=20 tauf=5.5e-10
```

Transient Measurements

```
** vccf power supply for forward reverse step recovery time
** vccr power supply for inverse reverse step recovery time
**   VPLUSF positive voltage for forward pulse generator
**   VPLUSr positive voltage for reverse pulse generator
**   Vminusf positive voltage for forward pulse generator
**   Vminusr positive voltage for reverse pulse generator
**   rloadf load resistor for forward
**   rloadr load resistor for reverse
.param vccf=30v
.param VPLUSF=9.9v
.param VMINUSF=-0.5v
.param rloadf=200
.TRAN 1N 75N 25N 200N 1N 300N 25N 1200N
.measure ’turn-on time ’trig par(’v(inf)-0.9*vplusf’) val=0
+ rise=1 targ par(’v(outf)-0.9*vccf’) val=0 fall=1
.measure ’fall time ’trig par(’v(outf)-0.9*vccf’) val=0
+ fall=1 targ par(’v(outf)-0.1*vccf’) val=0 fall=1
.measure ’turn-off time’ trig par(’v(inf)-0.1*vplusf’)
+ val=0 fall=1 targ par(’v(outf)-0.1*vccf’) val=0 rise=1
.measure ’rise time ’trig par(’v(outf)-0.1*vccf’) val=0
+ rise=1 targ par(’v(outf)-0.9*vccf’) val=0 rise=1

.graph V(INF) V(OUTF)
VCCF    VCCF   0      vccf
RLOADF  VCCF   OUTF   RLOADF
RINF    INF    VBASEF 1000
RPARF   INF    0      58
XSCOPf  OUTF   0      SCOPE

VINF    INF    0      PL    VMINUSF 0S   VMINUSF 5NS
+ VPLUSF 7NS VPLUSF 207NS VMINUSF 209NS
* CCX0F    VBASEF OUTF   CCX0F
* CEX0F    VBASEF 0      CEX0F
XQF      OUTF VBASEF   0    t2n2222
.MACRO SCOPE VLOAD VREF
RIN    VLOAD VREF 100K
CIN    VLOAD VREF   12P
.EOM
.END
```

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:

$$\text{Rpoly} = \frac{W}{L} \cdot 40 \qquad \text{poly} = \frac{\text{Esio} \cdot \text{nsi}}{\text{tox}} \cdot L \cdot W$$

$$\text{Rpoly} = \frac{100}{5} \cdot 40 = 800, \quad \text{Co} = \frac{3.9 \cdot 8.86}{800} \cdot 100 \cdot 5 = 215\,\text{fF} \quad \text{RC} = 138\ \text{ps}$$

For a transistor with:

- Channel width = 100 microns.
- Channel length = 1.2 microns.
- Gate oxide thickness = 250 Angstroms.

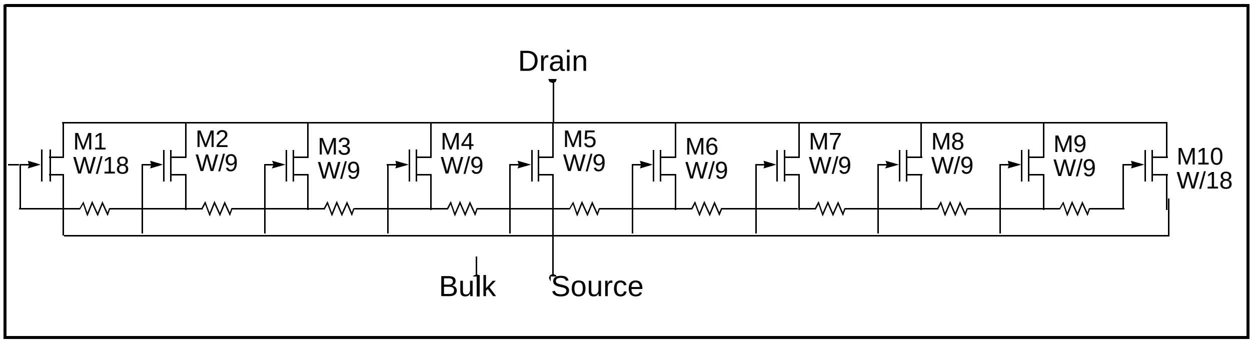
The resulting RC product for the polysilicon gate is:

$$\text{Rpoly} = \frac{\text{channel width}}{\text{channel length}} \cdot 40$$

$$\text{Co} = \frac{3.9 \cdot 8.86}{\text{Tox}} \cdot \text{channel width} \cdot \text{channel length} \quad \text{RC} = 546\ \text{ps}$$

You can use a nine-stage ladder model, to model the RC delay in CMOS devices.

Figure 13-13 Nine-stage Ladder Model



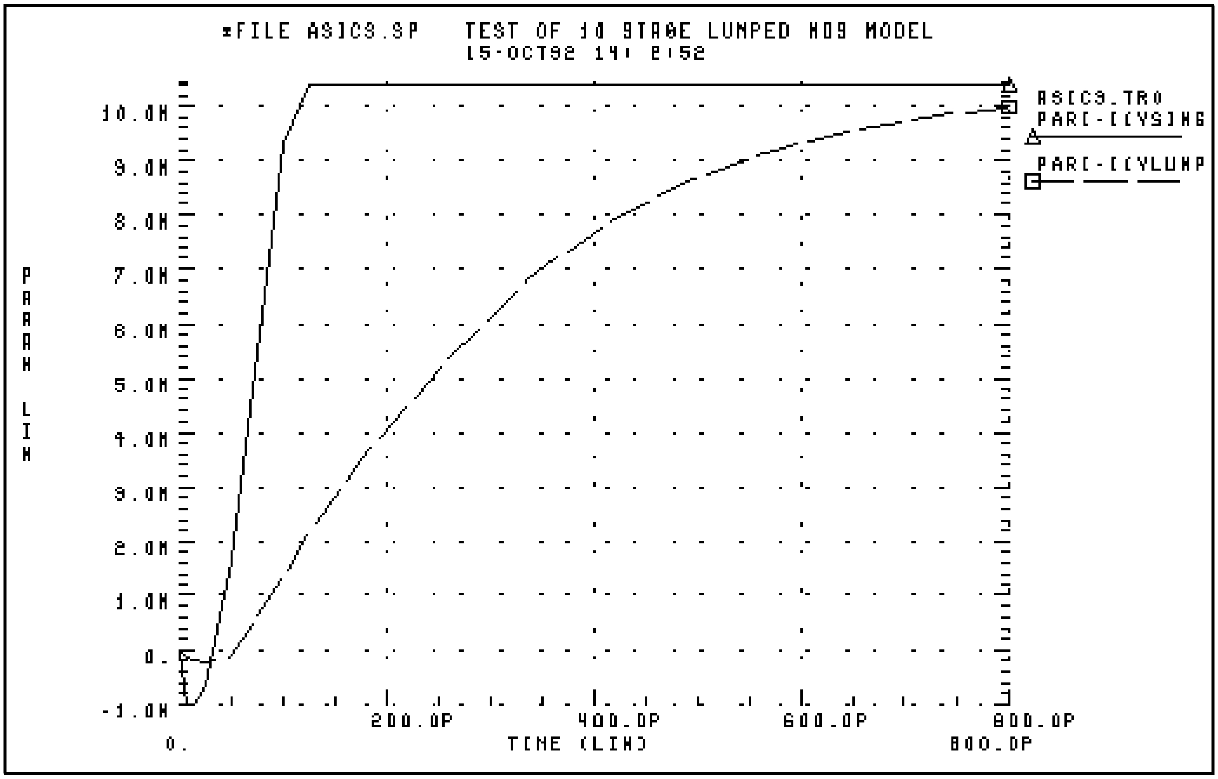
In this example, the nine-stage ladder model is in a data file, $*installdir*/demo/hspice/apps /asic3.sp. To optimize this model, HSPICE uses measured data from a wide channel transistor, as the target data\. Optimization produces a nine-stage ladder model, which matches the timing characteristics of the physical data. HSPICE compares the simulation results for the nine-stage ladder model, and the one-stage model, using the nine-stage ladder model as the reference. The one-stage model results are about 10% faster than actual physical data indicates.

EXAMPLE:

The following is an example of a Nine-Stage Ladder model:

```
* FILE: ASIC3.SP Test of 9 Stage Ladder Model
.subckt lrgtp drain gate source bulk
m1 drain gate source bulk p w=’wt/18’ l=lt
m2 drain g1 source bulk p w=’wt/9’ l=lt
m3 drain g2 source bulk p w=’wt/9’ l=lt
m4 drain g3 source bulk p w=’wt/9’ l=lt
m5 drain g4 source bulk p w=’wt/9’ l=lt
m6 drain g5 source bulk p w=’wt/9’ l=lt
m7 drain g6 source bulk p w=’wt/9’ l=lt
m8 drain g7 source bulk p w=’wt/9’ l=lt
m9 drain g8 source bulk p w=’wt/9’ l=lt
m10 drain g9 source bulk p w=’wt/18’ l=lt
r1 gate g1 ’wt/lt*rpoly/9’
r2 g1 g2 ’wt/lt*rpoly/9’
r3 g2 g3 ’wt/lt*rpoly/9’
r4 g3 g4 ’wt/lt*rpoly/9’
r5 g4 g5 ’wt/lt*rpoly/9’
r6 g5 g6 ’wt/lt*rpoly/9’
r7 g6 g7 ’wt/lt*rpoly/9’
r8 g7 g8 ’wt/lt*rpoly/9’
r9 g8 g9 ’wt/lt*rpoly/9’
.ends lrgtp
.end pro
.end
```

Figure 13-14 Asic3 Single vs. Lumped Model



Demonstration Input Files

Table 13-4 Demonstration Input Files (Sheet 1 of 15)

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
xalg2.sp	QA of parameters

Table 13-4   Demonstration Input Files (Sheet 2 of 15)

File Name	**Description**
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
mondc_b.sp	Monte Carlo DC analysis
mont1.sp	Monte Carlo Gaussian, uniform, and limit function
mos2bit.sp	two-bit MOS adder
pll.sp	phase-locked loop
sclopass.sp	switched-capacitor low-pass filter
worst.sp	worst case skew models, using .ALTER
xbjt2bit.sp	BJT NAND gate two-bit binary adder

Table 13-4 Demonstration Input Files (Sheet 3 of 15)

File Name	**Description**
Behavioral Applications$installdir/demo/hspice/behave	
acl.sp	acl gate
amp_mod.sp	amplitude modulator, with pulse waveform carrier
behave.sp	AND/NAND gates, 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, using a PWL VCCS
dlatch.sp	CMOS D-latch, 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
pdb.sp	phase detector, 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, using a PWL VCVS
pwl8.sp	smoothing the triangle waveform, using the PWL CCCS
ring5bm.sp	five-stage ring oscillator – macromodel CMOS inverter
ringb.sp	ring oscillator, using behavioral model
sampling.sp	sampling a sine wave

Table 13-4 Demonstration Input Files (Sheet 4 of 15)

File Name	Description
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, using behavioral elements
triodex.sp	triode model family of curves, using behavioral elements
tunnel.sp	modeling tunnel diode characteristic, using PWL VCCS
vcob.sp	voltage-controlled oscillator, using PWL functions
Benchmarks$installdir/demo/hspice/bench	
bigmos1.sp	large MOS simulation
demo.sp	quick demo file, to test installation
m2bit.sp	72-transistor two-bit adder – typical cell simulation
m2bitf.sp	fast simulation example
m2bitsw.sp	Fast simulation example. Same as m2bitf.sp, but uses behavioral elements
senseamp.sp	bipolar analog test case
Timing Analysis$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, 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

Table 13-4 Demonstration Input Files (Sheet 5 of 15)

File Name	**Description**
Cell Characterization$installdir/demo/hspice/cchar	
dff.sp	DFF bisection search, for setup time
inv3.sp	characteristics of an inverter
inva.sp	characteristics of an inverter
invb.sp	characteristics of an inverter
load1.sp	inverter sweep, delay versus fanout
setupbsc.sp	setup characteristics
setupold.sp	setup characteristics
setuppas.sp	setup characteristics
sigma.sp	sweep MOSFET -3 sigma to +3 sigma, using measure output
tdgtl.a2d	Viewsim A2D HSPICE input file
tdgtl.d2a	Viewsim D2A HSPICE input file
tdgtl.sp	two-bit adder, using D2A Elements
Circuit Optimization$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

Table 13-4   Demonstration Input Files (Sheet 6 of 15)

File Name	Description
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
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

Table 13-4 Demonstration Input Files (Sheet 7 of 15)

File Name	Description
alm258.sp	characteristics of National op-amp
alm258a.sp	characteristics of National op-amp
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
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

Table 13-4 Demonstration Input Files (Sheet 8 of 15)

File Name	Description
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
t2n2222.sp	characteristics of BJT 2n2222
Device Optimization$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

Table 13-4   Demonstration Input Files (Sheet 9 of 15)

File Name	**Description**
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$installdir/demo/hspice/fft	
am.sp	FFT analysis, AM source
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

Table 13-4 Demonstration Input Files (Sheet 10 of 15)

File Name	**Description**
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
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

Table 13-4 Demonstration Input Files (Sheet 11 of 15)

File Name	Description
Filters$installdir/demo/hspice/filters	
fbp_1.sp	bandpass LCR filter, measurement
fbp_2.sp	bandpass LCR filter, pole/zero
fbpnet.sp	bandpass LCR filter, s-parameters
fbprlc.sp	LCR AC analysis, for resonance
fhp4th.sp	high-pass LCR, fourth-order Butterworth filter
fkerwin.sp	pole/zero analysis of Kerwin’s circuit
flp5th.sp	low-pass, fifth-order filter
flp9th.sp	low-pass, ninth-order `FNDR`, with ideal op-amps
micro1.sp	test of microstrip
micro2.sp	test of microstrip
tcoax.sp	test of RG58/AU coax
trans1m.sp	FR-4, printed-circuit, lumped transmission line
Magnetics$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

Table 13-4 Demonstration Input Files (Sheet 12 of 15)

File Name	**Description**
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
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

Table 13-4   Demonstration Input Files (Sheet 13 of 15)

File Name	**Description**
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
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

Table 13-4   Demonstration Input Files (Sheet 14 of 15)

File Name	Description
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 voltage-divider
rad13.sp	three-input NMOS NAND gate, with non-EPI, EPI, and SOS examples
rad14.sp	GaAs differential-amplifier circuit
rad14dc.sp	n-channel JFET, DC I-V curves
Sources$installdir/demo/hspice/sources	
amsrc.sp	amplitude modulation
exp.sp	exponential independent source
pulse.sp	test of pulse
pwl.sp	repeated piecewise-linear source
pwl10.sp	op-amp, voltage follower
rtest.sp	voltage-controlled resistor, inverter chain
sffm.sp	single-frequency, FM modulation source
sin.sp	sinusoidal source, waveform
vcr1.sp	switched-capacitor network, using G-switch
Transmission Lines$installdir/demo/hspice/tline	
fr4.sp	microstrip test, FR-4 PC board material
fr4o.sp	optimizing model, for microstrip FR-4 PC board material
fr4x.sp	FR4 microstrip test

Table 13-4 Demonstration Input Files (Sheet 15 of 15)

File Name	**Description**
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

A

Full Simulation Examples

- The examples in this chapter show the basic text and post-processor output, for a sample input netlist.The first example uses AvanWaves to view results.
- The second example uses Cosmos-Scope.

This chapter includes the following sections:

Simulation Example Using AvanWaves

Input Netlist and Circuit

The following example is an input netlist for a linear CMOS amplifier. Comment lines indicate the individual sections of the netlist. Refer to Simulation Input and Controls on page 3-1, for information about the individual commands.

```
* 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
```

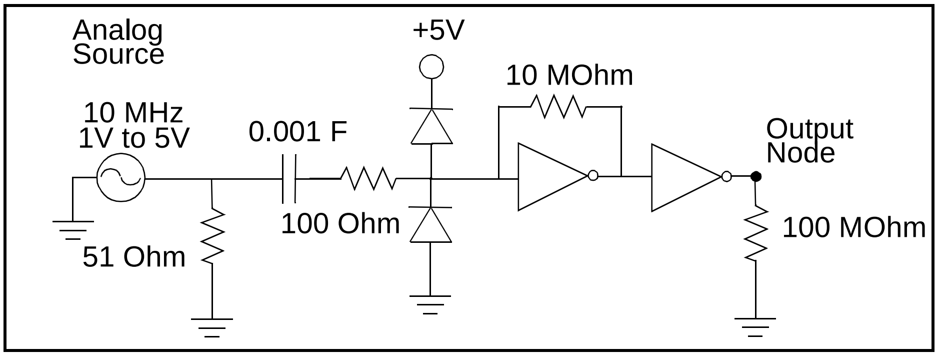
```
.model diode_mod d
* analysis specifications
.TRAN 10n 1u sweep analog_voltage lin 5 1.0 5.0

* 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 A-1 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 A-1 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 A-1   HSPICE Output Files

File Name	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, 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 on the following pages show the six different post-processor outputs from the simulation of the example netlist. The format of these plots is for AvanWaves, the Synopsys graphical waveform viewer. These plots are postscript output, from the actual data.

Figure A-2 Plot of Voltage on Node in

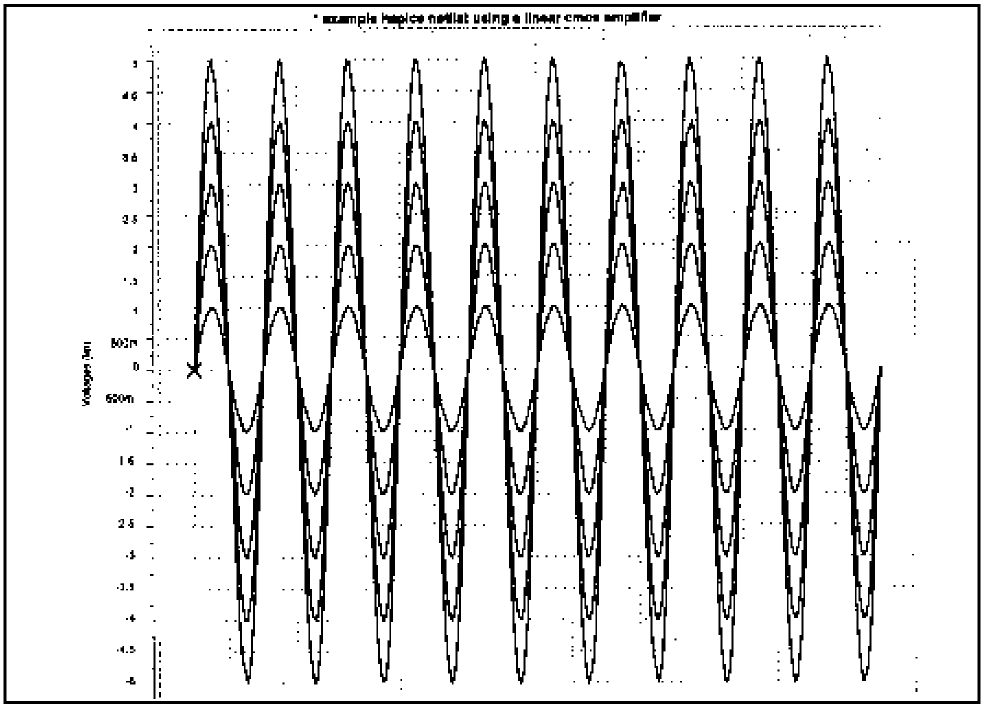


Figure A-3 Plot of Voltage on Node clamp vs. Time

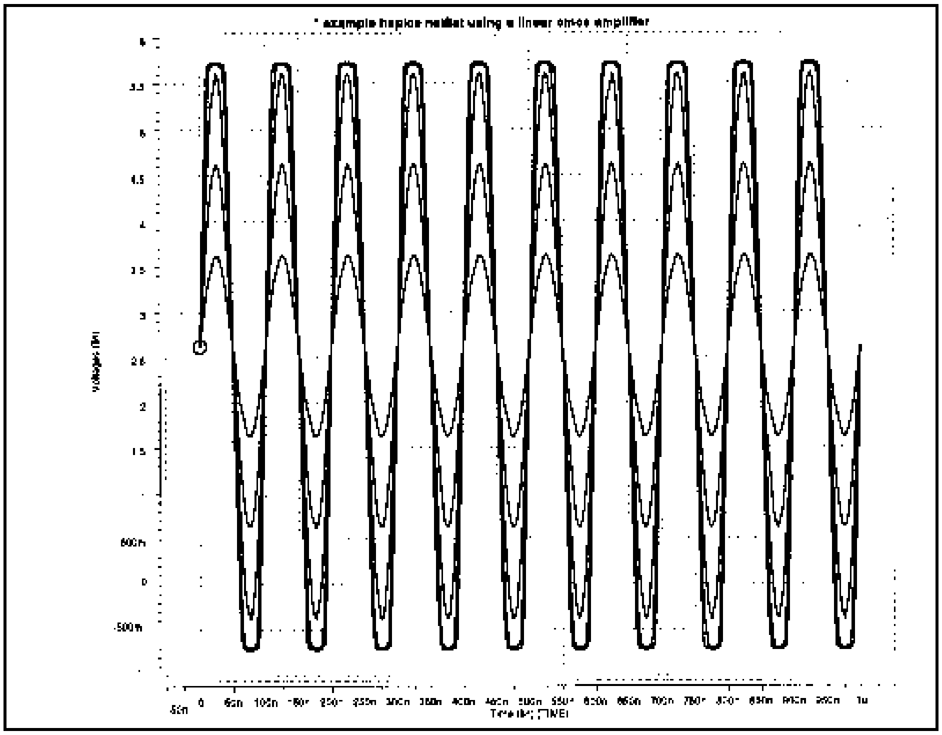


Figure A-4   Plot of Voltage on Node inv1out vs.Time

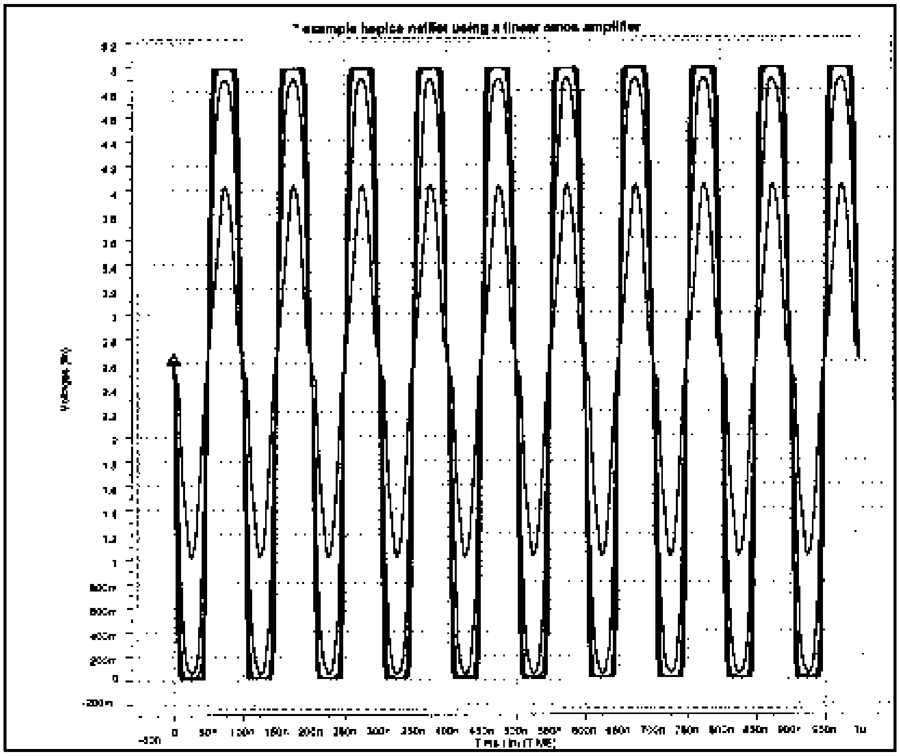


Figure A-5   Plot of Voltage on Node inv2out vs. Time

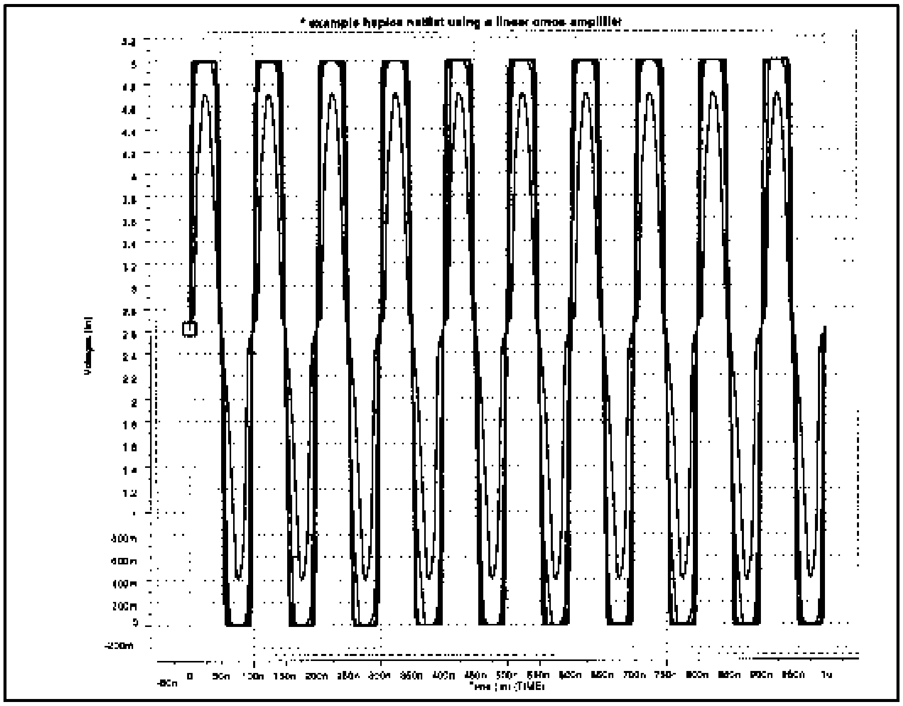


Figure A-6   Plot of Current through Diode dlow vs. Time

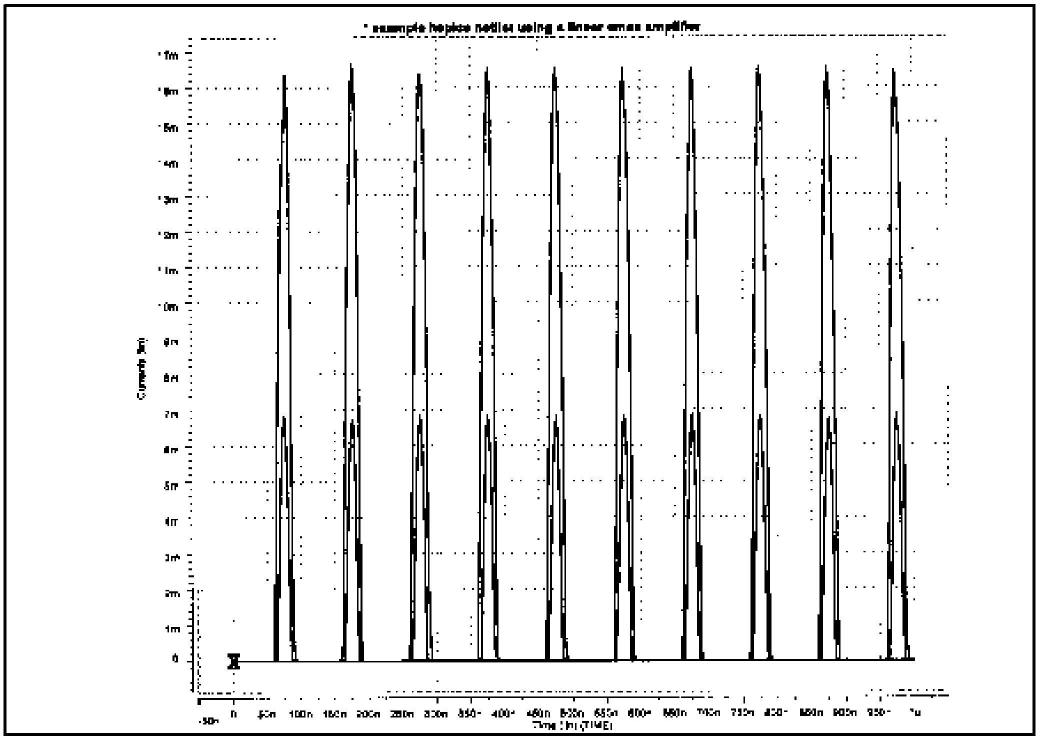
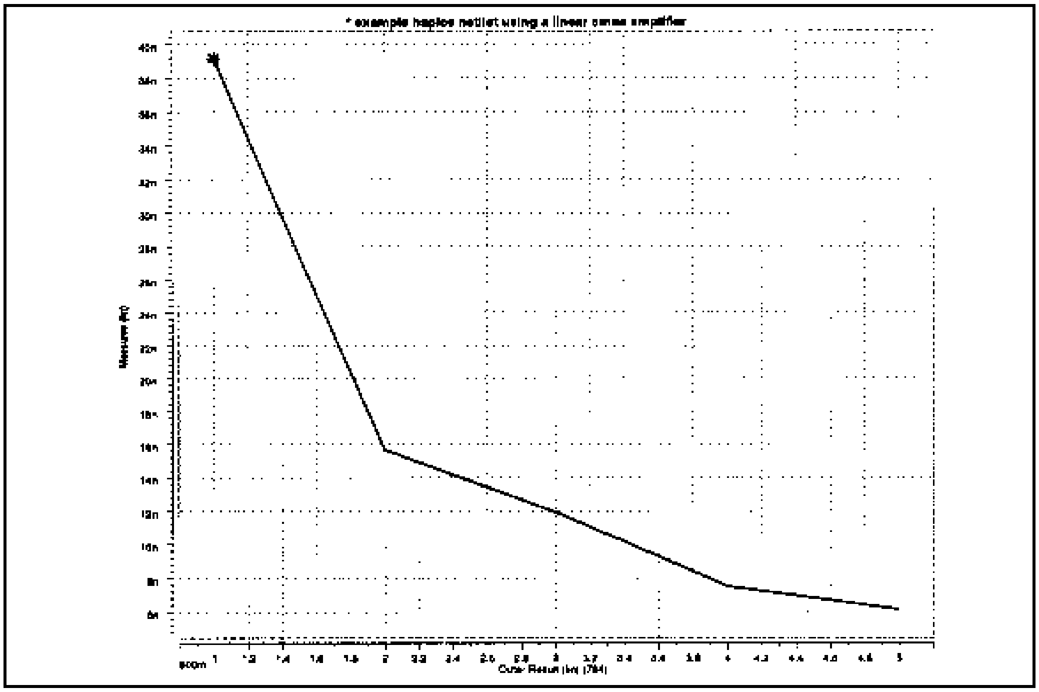


Figure A-7   Plot of Measured Variable falltime vs. Amplifier Input Voltage



Simulation Example Using Cosmos-Scope

This example demonstrates the basic steps to perform simulation output, and to view the waveform results, using the Synopsys Cosmos-Scope Waveform Viewer.

Input Netlist and Circuit

The Syntax section below shows the input netlist for a BJT diff amplifier. Comment lines indicate the individual sections of the netlists. See Simulation Input and Controls on page 3-1, for information about the individual commands.

SYNTAX:

```
*file: bjtdiff.sp bjt diff amp with every analysis type
*# ANALYSIS: ac dc tran tf noise new four sens pz disto temp
*# OPTIONS: list node ingold=2 measdgt=5 numdgt=8
+ probe post
*# TEMPERATURE: 25
* netlist options
.OPTION list node ingold=2 measdgt=6 numdgt=8 probe post
* defined parameters
.PARAM rb1x=aunif(20k,1k,30k) rb2x=aunif(20k,1k,30k)
* analysis specifications
.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
.DISTRO rc1 20 .9 1m 1.0
.SENS v(4)
.TRAN 5ns 200ns
.FOUR 5meg v(5) v(15)
.TEMP -55 150
* output specifications
.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
.PROBE dc v(4) v(5) v(14) v(15)
.PROBE ac vm(5) vp(5) vm(15) vp(15)
.PROBE ac vt(5) vt(15)
.PROBE noise onoise(m) inoise(m)
.PROBE ac z11(m) z12(m) z22(m) zin(m)
.PROBE disto hd2 hd3 sim2 dim2 dim3
.PROBE tran v(4) v(5) v(14) v(15)
.PROBE tran p(vcc) p(vee) p(vin) power
* source statements
VIN 1 0 sin(0 0.1 5meg) ac 1
VCC 8 0 12
VEE 9 0 -12
* circuit statements
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 definitions
.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 A-2 Output Files

File Name	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 Cosmos-Scope

The steps below show how to use the Synopsys Cosmos-Scope 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

1. Invoke Cosmos-Scope.

   From a Unix command line:

   ```
   % cscope
   ```

   On a Windows-NT system:

   ```
   Programs > (user_install_location)> Cosmos-Scope
   ```

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.

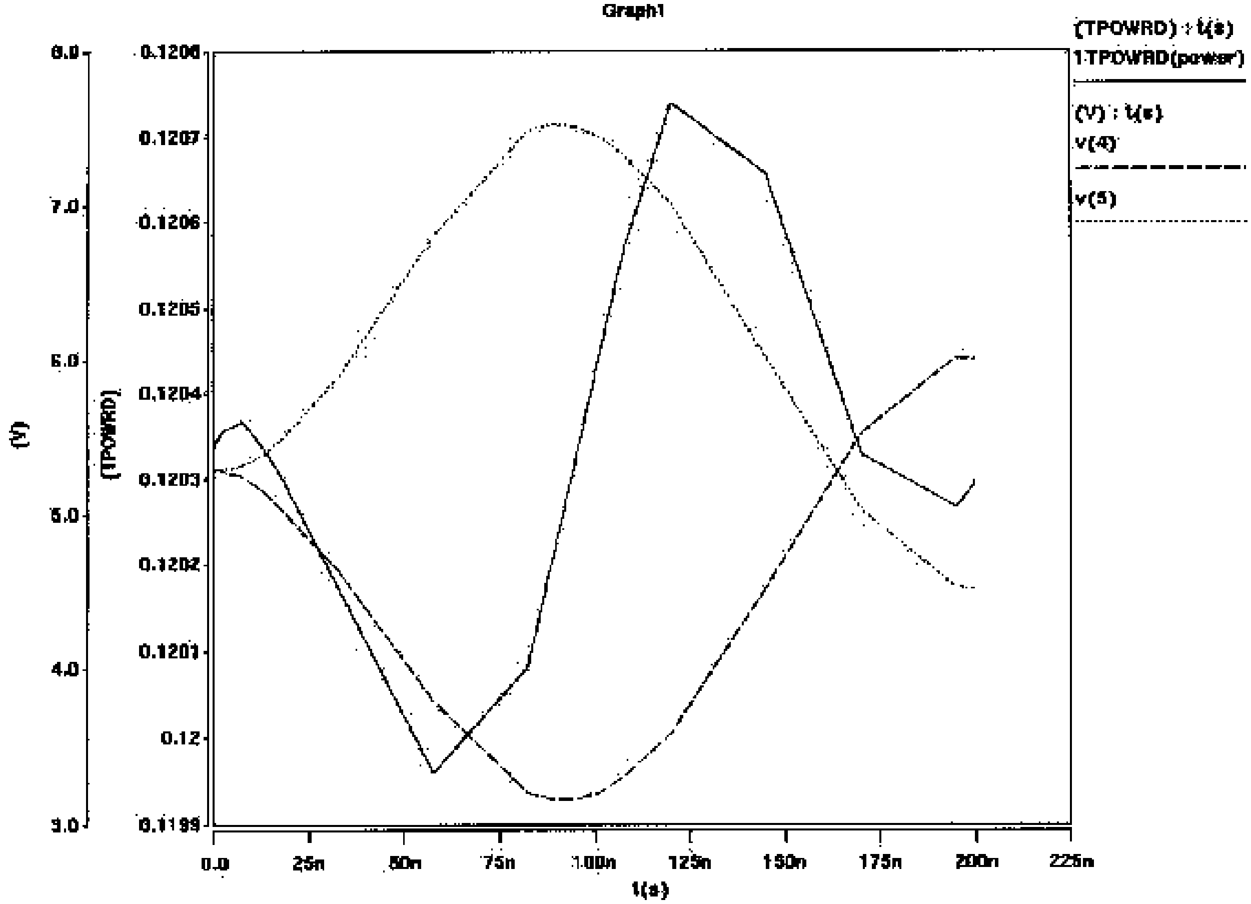
   The Signal Menu opens.

8. From the Signal Menu, select Stack Region > Analog 0.

9. Repeat Step 7 for the next top-most signal.

   A plot opens, similar to Figure A-8 on page A-19.

Figure A-8 Transient Analysis: Plot of v(4), v(5), and ITPOWERD (power)



Viewing HSPICE AC Analysis Waveforms

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.

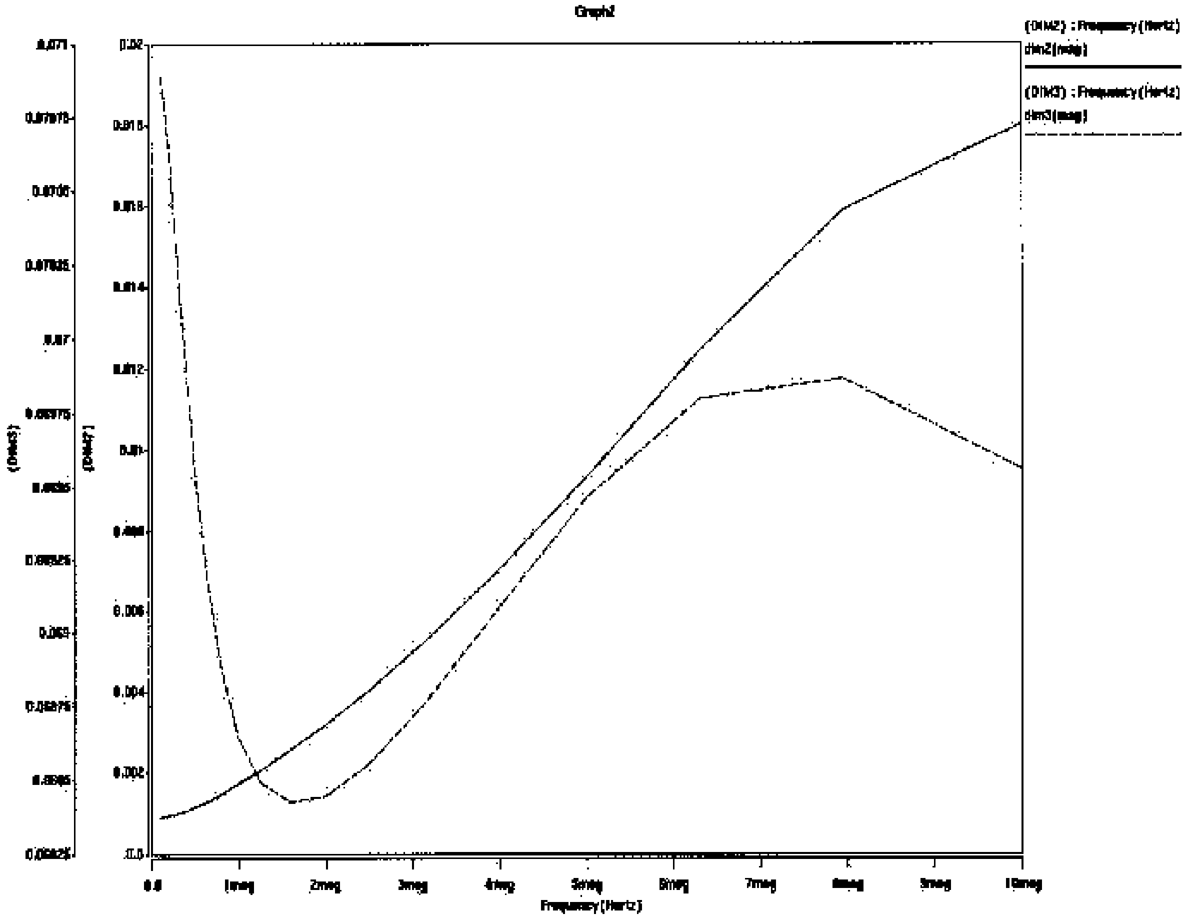
7. For two signals in a plot, right-click on dim2(mag).

   A Signal Menu opens.

8. From the Signal Menu, select Stack Region > Analog 0.

   A plot opens, similar to Figure A-9 on page A-20.

Figure A-9 AC Analysis Result: Plot of dim2(mag) and dim3(mag) from bjtdiff.ac0



Viewing HSPICE DC Analysis Waveforms

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.

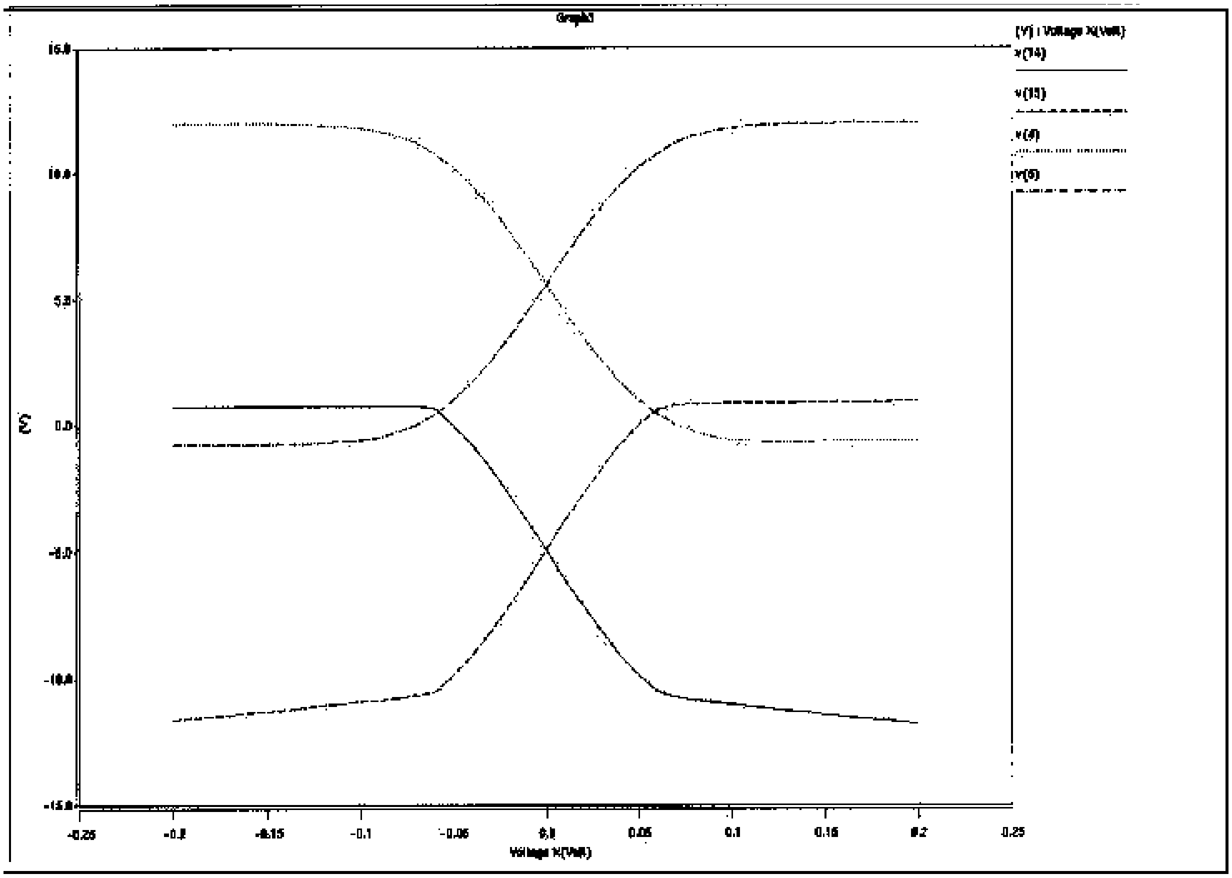
7. To see four signals in one plot, right-click on the name of the top-most signal.

   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 Figure A-10.

Figure A-10   DC Analysis Result: Plot of v(14), v(15), v(4), and v(5) from bjtdiff.sw0



The *Cosmos-Scope 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`

Index