HSPICE® User Guide: Simulation and Analysis

Version B-2008.09, September 2008

SYNOPSYS®

Contents

About this User Guide

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

Inside this Guide

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

Chapter	Description
Part 1: Introduction to HSPICE	
Chapter 1, Overview	Describes HSPICE features and the simulation process.
Chapter 2, Setup	Describes the environment variables and standard I/O files.
Chapter 3, Startup and Simulation	Describes the invocation commands and types of simulation and analysis available in HSPICE.
Chapter 4, Input Netlist and Data Entry	Describes the input netlist file and methods of entering data.
Chapter 5, Using Interactive Mode	Provides details on invoking and using interactive mode.
Chapter 6, HSPICE GUI for Windows	Describes the graphical user interface available on the Windows platform only.
Chapter 7, Library and Data Encryption	Describes the three methods available to create encrypted files.
Part 2: Elements and Devices	

Chapter	Description
Chapter 8, Elements	Describes the syntax for the basic elements of a circuit netlist in HSPICE or HSPICE RF.
Chapter 9, Sources and Stimuli	Describes element and model statements for independent sources, dependent sources, analog-to-digital elements, and digital-to-analog elements.
Chapter 10, Parameters and Functions	Describes how to use parameters within an HSPICE netlist.
Chapter 11, Simulation Output	Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.
Part 3: Analyses	
Chapter 12, Initializing DC/Operating Point Analysis	Describes DC initialization and operating point analysis.
Chapter 13, Pole/Zero Analysis	Describes how to use pole/zero analysis in HSPICE or HSPICE RF to study the behavior of linear, time-invariant networks.
Chapter 14, Transient Analysis	Describes how to use transient analysis to compute the circuit solution.
Chapter 15, Spectrum Analysis	Describes how to use spectrum analysis to provide the highest FFT accuracy with minimal overhead in simulation time.
Chapter 16, AC Small-Signal and Noise Analysis	Describes how to perform AC and noise small signal analysis.
Chapter 17, Linear Network Parameter Analysis	Describes how to perform an AC sweep to extract small-signal linear network parameters.
Chapter 18, Statistical Eye Analysis	Describes how to perform statistical eye diagram analysis using .STATEYE.
Chapter 19, Timing Analysis Using Bisection	Describes how to use the bisection function in timing optimization.

Chapter	Description
Chapter 20, Analyzing Variability and Using the Variation Block	Describes the use model and structure of the Variation Block in HSPICE.
Chapter 21, Monte Carlo Analysis Using the Variation Block Flow	Describes Monte Carlo analysis in HSPICE.
Chapter 22, Mismatch Analyses	Describes the use of DCmatch analysis in HSPICE.
Chapter 23, Exploration Block	Describes the use of the Exploration Block in HSPICE.
Chapter 24, Optimization	Describes optimization in HSPICE for optimizing electrical yield.
Chapter 25, RC Reduction and Post-Layout Simulation	Describes RC network reduction in HSPICE.
Chapter 26, MOSFET Model Reliability Analysis (MOSRA)	Describes reliability analysis for MOSFET devices.
Part 4: Simulation Applications	
Chapter 27, Performing Digital Cell Characterization	Describes how to characterize cells in a data-driven analysis.
Chapter 28, Behavioral Modeling	Describes how to substitute abstract circuit models for lower-level descriptions of analog functions.
Chapter 29, Modeling Filters and Networks	Describes modeling filters and networks, including Laplace transforms.
Chapter 30, Simulation of Random Noise	Describes the characteristics of random signals, types of noise, component noise models, and noise simulation.
Chapter 31, Using Verilog-A	Describes how to use Verilog-A in HSPICE and HSPICE RF simulations.
Part 5: Demonstration Files/Errors-Warnings	

Chapter	Description
Chapter 31, Running Demonstration Files	Contains examples of basic file construction techniques, advanced features, and simulation tricks. Lists and describes several HSPICE and input files.
Chapter 32, Warning/Error Messages	Provides an overview of the type of warnings and error messages that HSPICE prints and troubleshooting measures to take when possible.
Appendix A, Statistical Analysis	Describes the features available in HSPICE for statistical analysis before the Y-2006.03 release.
Appendix B, Full Simulation Example	Contains information and sample input netlist for a full simulation example in HSPICE.
Appendix C, Obsolete HSPICE Functionality	Describes out-of-date, rarely used, or de-emphasized functionality.
Appendix D, Transient Simulation with RUNLVL=0	Describes running a transient simulation with .OPTION RUNLVL=0.

The HSPICE Documentation Set

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

The HSPICE Documentation Set

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

Manual	Description
HSPICE User Guide: Simulation and Analysis	Describes how to use HSPICE to simulate and analyze your circuit designs, and includes simulation applications. This is the main HSPICE user guide.

Manual	Description
HSPICE User Guide: Signal Integrity	Describes how to use HSPICE to maintain signal integrity in your chip design.
HSPICE User Guide: RF Analysis	Describes how to use special set of analysis and design capabilities added to HSPICE to support RF and high-speed circuit design.
HSPICE Reference Manual: Commands and Control Options	Provides reference information for HSPICE and HSPICE RF commands and options.
HSPICE Reference Manual: Elements and Device Models	Describes standard models you can use when simulating your circuit designs in HSPICE, including passive devices, diodes, JFET and MESFET devices, and BJT devices.
HSPICE Reference Manual: MOSFET Models	Describes available MOSFET models you can use when simulating your circuit designs in HSPICE.
HSPICE Integration to Cadence^{TM} Virtuoso® Analog Design Environment User Guide	Describes use of the HSPICE simulator integration to the Cadence tool.
AMS Discovery Simulation Interface Guide for HSPICE	Describes use of the Simulation Interface with other EDA tools for HSPICE.
AvanWaves User Guide	Describes the AvanWaves tool, which you can use to display waveforms generated during HSPICE circuit design simulation.

Searching Across the HSPICE Documentation Set

You can access the PDF format documentation from your install directory for the current release by entering `-docs` on the terminal command line when the HSPICE tool is open.

Synopsys includes an index with your HSPICE documentation that lets you search the entire HSPICE documentation set for a particular topic or keyword. In a single operation, you can instantly generate a list of hits that are hyperlinked to the occurrences of your search term. For information on how to perform searches across multiple PDF documents, see the HSPICE release notes.

Note:

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

You can also invoke HSPICE and RF documentation in a browser-based help system by entering `-help` on your terminal command line when the HSPICE tool is open. This provides access to all the HSPICE manuals with the exception of the *AvanWaves User Guide* which is available in PDF format only.

Known Limitations and Resolved STARs

You can find information about known problems and limitations and resolved Synopsys Technical Action Requests (STARs) in the *HSPICE Release Notes* shipped with this release. For updates, go to SolvNet.

To access the *HSPICE Release Notes*:

1. Go to https://solvnet.synopsys.com/ReleaseNotes. (If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
2. Select Download Center> HSPICE> version number> Release Notes.

Conventions

The following typographical conventions are used in Synopsys HSPICE documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Italic`	Indicates a user-defined value, such as *`object_name`*.
`Bold`	Indicates user input—text you type verbatim—in syntax and examples.

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

Customer Support

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

Accessing SolvNet

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

To access SolvNet:

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

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

Contacting the Synopsys Technical Support Center

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

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

Part: 1 Introduction to HSPICE

This manual is organized according to the following five Parts:

- Introduction to HSPICE
- Elements and Devices
- Analyses
- Simulation Applications
- Demonstration Files/Errors-Warnings

Part 1 presents the following chapters /topics:

1 Overview

Describes HSPICE features and the simulation process.

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

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

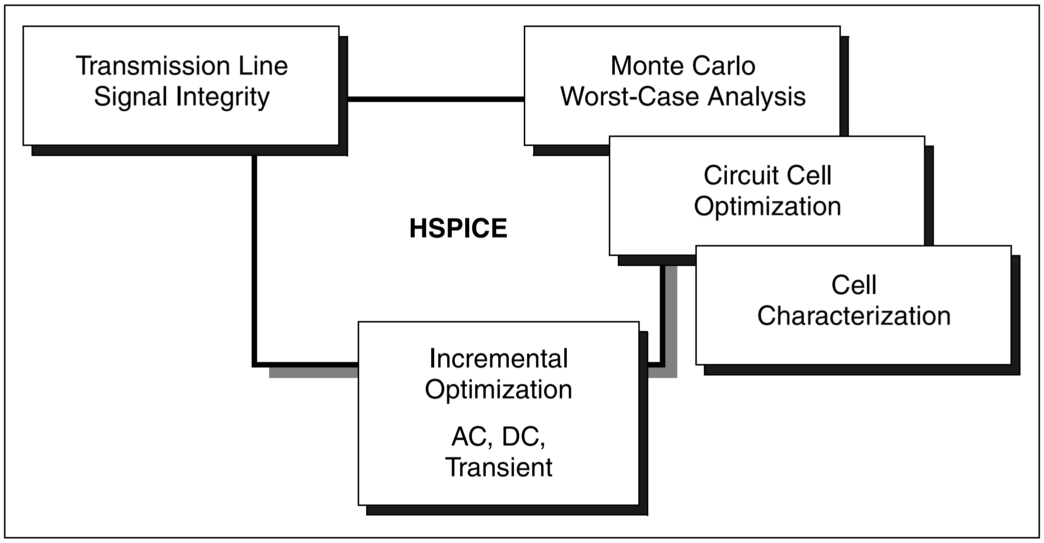


Figure 1 Synopsys HSPICE Design Features

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

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

For a description of commands that you can include in your HSPICE netlist, see the HSPICE and HSPICE RF Netlist Commands chapter in the *HSPICE Reference Manual: Commands and Control Options*.

These topics are covered in the following sections:

- HSPICE Varieties
- Features
- HSPICE Features for Running Higher-Level Simulations
- Simulation Structure
- Experimental Methods Supported by HSPICE
- Simulation Process Overview

HSPICE Varieties

Synopsys HSPICE is available in two varieties:

- HSPICE
- HSPICE RF

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

HSPICE RF is newer and offers many (but not all) HSPICE simulation capabilities and HSPICE RF simulations of radio-frequency (RF) devices, which HSPICE does *not* support.

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

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 characterize standard cell libraries
- Geometric lossy-coupled transmission lines for PCB, multi-chip, package, and IC technologies
- Discrete component, pin, package, and vendor IC libraries
- Interactive graphing and analysis of multiple simulation waveforms by using with waveform viewers such as WaveView Analyzer and CosmosScope
- Flexible license manager that allocates licenses intelligently based on run status and user-specified job priorities

  If you suspend a simulation job (Ctrl-Z), the load sharing facility (LSF) license manager signals HSPICE to release that job's license. This frees the license for another simulation job, or so the stopped job can reclaim the license and resume. You can also prioritize simulation jobs you submit; LSF automatically suspends low-priority simulation jobs to run high-priority jobs. When the high-priority job completes, LSF releases the license back to the lower-priority job, which resumes from where it was suspended. To resume the LSF job, on the same terminal, type either **fg** or **bg**.
- A number of circuit analysis types (see Figure 2 on page 6) and device modeling technologies.

- Ability to integrate models with the Custom CMI for which HSPICE or HSPICE RF uses a dynamically-linked shared library. Consult your HSPICE technical support team for access to the HSPICE CMI application note and source code.
- Ability to invoke the TMI flow using proprietary TSMC model files and compiled libraries. Jointly developed by Synopsys and TSMC the TMI technology and API provides a compact model with additional instance parameters and equations for an advanced modeling approach to support TSMC's extension of the standard BSIM4 model. Modeling API code is written in C and available in a compiled format for HSPICE and HSIM to link to during the simulation. TMI-required settings to invoke the flow and the location of a *.so* file are set by TSMC. The API also performs automatic platform selection on the *.so* file. Both HSPICE and HSIM provide the tool binaries and support the same **.so* file.

  Use the existing HSPICE and HSIM commands to run the simulation. (Contact Synopsys Technical Support for further information.) See also the *HSPICE Reference Manual: Commands and Control Options* for .OPTION TMI FLAG and .OPTION TMIPATH.

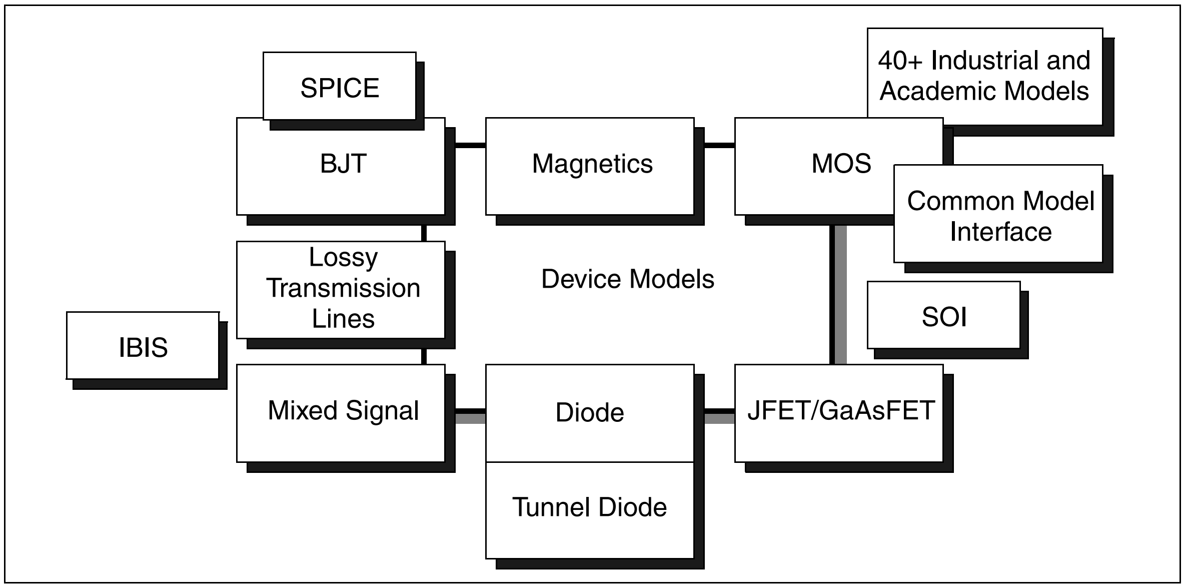


Figure 2 Synopsys HSPICE Modeling Technologies

HSPICE Features for Running Higher-Level Simulations

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

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

- Explicit include files – `.INCLUDE` 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 statements.
- Statistical analysis – sweep monte analysis statements.
- Multiple alternative –`.ALTER` statement.
- Automatic measurements – `.MEASURE` statement.
- Condition-controlled netlists (`IF`-`ELSEIF`-`ELSE`-`ENDIF` statements).

Simulation Structure

Experimental Methods Supported by HSPICE

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

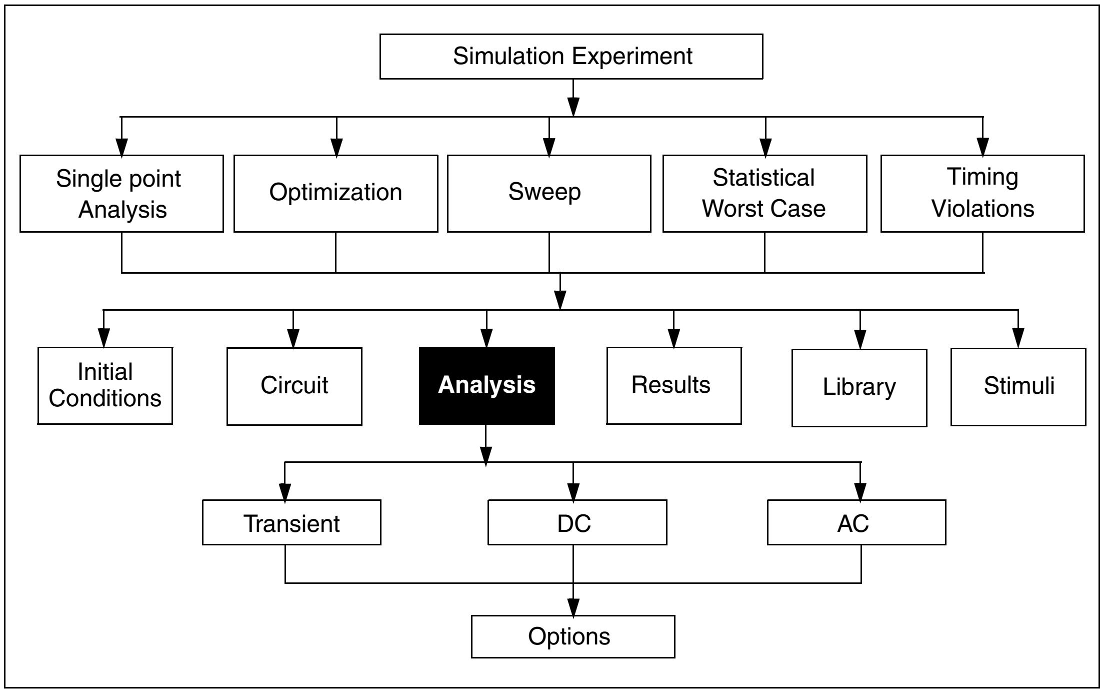


Figure 3 Simulation Program Structure

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

- process
- voltage
- temperature
- parasitics

HSPICE 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).

Simulation Process Overview

Figure 4 shows the HSPICE simulation process.

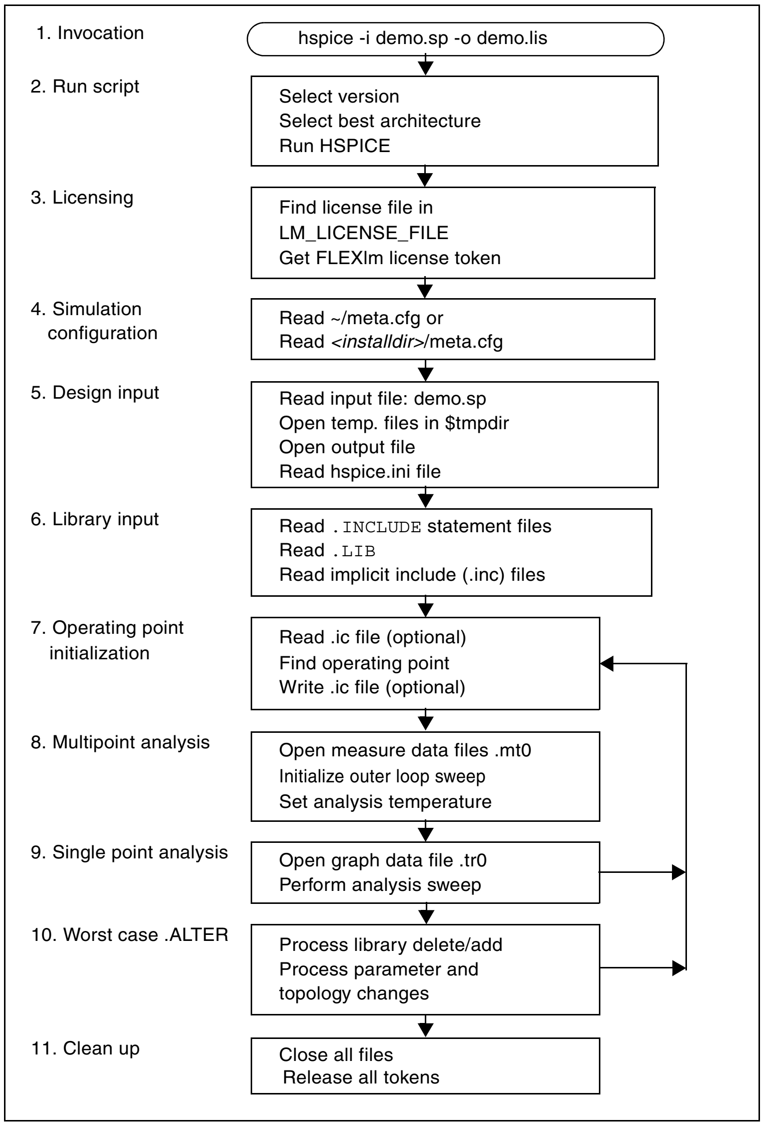


Figure 4 Simulation Process

2

Setup

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

For descriptions of individual HSPICE commands mentioned in this chapter, see the HSPICE Reference Manual: Commands and Control Options.

These topics are discussed in the following sections:

- Setting Environment Variables
- Standard Input Files
- Standard Output Files
- Working Directory Path Character Limit

Setting Environment Variables

The following sections describe procedures for setting the environment variables needed to run HSPICE.

- License Variable
- License Queuing Variable
- Temporary Directory Variable
- Windows Variable

License Variable

HSPICE or HSPICE RF requires you to set the `LM_LICENSE_FILE` environment variable. This variable specifies the location of the license.dat license file. Set the `LM_LICENSE_FILE` environment variable to `port@hostname` to point to a license file on a server.

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

  ```
  setenv LM_LICENSE_FILE port@hostname
  ```

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

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

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

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

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

Temporary Directory Variable

Specify the location to deposit scratch files by setting the `tmpdir` (UNIX/Linux), `TEMP` or `TMP` (Windows) environment variable. HSPICE opens three scratch files in the /tmp directory. To change this directory, reset the `tmpdir` environment variable in the HSPICE command script.

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

License Queuing Variable

The optional `META_QUEUE` environment variable is a useful feature that causes HSPICE or HSPICE RF to wait for an available license. It is particularly helpful in environments where the tool is run sequentially from batch files and a license checkout failure could result in the loss of important data. (AvanWaves also supports use of this environment variable.) `META_QUEUE`, however, does not queue across license “pools” (which are disallowed in FLEXlm).

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

```
setenv META_QUEUE 1
```

A pool of licenses is created by an INCREMENT line that contains one or more license tokens.

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

Additional pools can be created by having multiple INCREMENT lines in the same license file, such as exist in the case of separate tool purchases or off-maintenance keys.

Another way multiple pools can exist is with the addition of multiple server entries in the LM_LICENSE_FILE variable. For example:LM_LICENSE_FILE = 27000@server1:27000@server2:27000@server3

Without queuing enabled, the tool tries the INCREMENT lines in server1, then server2, and so on, until all servers in the list are exhausted. When you turn on queuing, however, the first INCREMENT in the first pool that is queried (server1) queues the request, and the application does not continue to look in other license pools for available tokens.

Windows Variable

Setting the HSPWIN_KEY environment variable to 1 checks out the hspicewin license token first when an HSPICE simulation is run. If not set to 1, an hspice token is checked out first. The HSPWIN_KEY environment variable is only available on the Windows platform.

Note:

When installing the HSPICE program on Windows, the ADMINISTRATOR priority is essential for successful installation.

Standard Input Files

This section describes the standard input files to 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. You can also use WaveView Analyzer.

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 stdout on UNIX platforms, which you can redirect.

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

Table 1 Input Files

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

Output Configuration File (*meta.cfg*)

You use the output configuration file to set up the printer, plotter, and terminal. The HSPICE default search order for the *meta.cfg* file is as follows:

1. `cwd/meta.cfg`	— current working directory
2. `-i input.sp/meta.cfg`	— same directory as input case
3. `$HOME/meta.cfg`	— user HOME directory
4. `$installdir/meta.cfg`	— HSPICE installation directory

Besides the *meta.cfg* file, HSPICE also reads one `cwd/casename_prefix.cfg` configuration file for current simulation case output.

The configuration file can contain one line (`default_include=$path>/filename`) for HSPICE to read one default-include file. The default-include filename is case-sensitive (except for the Windows versions of HSPICE). The default-include file can be overridden by setting the environment variable `'default_include'`(note lower case).

Initialization File (hspice.ini)

The initialization file enables you to specify user defaults. If HSPICE reads one *hspice.ini* file, HSPICE includes its contents at the top of the input file. (This does not apply to HSPICE RF). All HSPICE simulations will look for ONE implicit hspice.ini file. The HSPICE default search order for the *hspice.ini* file is:

1. `cwd/hspice.ini`	—current working directory
2. `$HOME/hspice.ini`	—user HOME directory
3. `$installdir/hspice.ini`	—HSPICE installation directory

You can use an initialization file to set options (for `.OPTION` statements) and to access libraries. To include customized initialization files, you can define default_include=*filename* in a *command.inc* or *meta.cfg* file.

DC Operating Point Initial Conditions File

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

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

Input Netlist File

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

Library Input File

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

Analog Transition Data File

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

Standard Output Files

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

Table 2 HSPICE Output Files and Extensions

Output File Type	**Extension**
AC analysis measurement results	.ma#[a]
AC analysis results (from `.POST` statement)	.ac#
DC analysis measurement results	.ms#
DC analysis results (from `.POST` statement)	.sw#
Digital output	.a2d
FFT analysis graph data (from `FFT` statement)	.ft#
Hardcopy graph data (from meta.cfg PRTDEFAULT)	.gr#[b]
Operating point information (from .OPTION OPFILE statement)	.dp#
Operating point node voltages (initial conditions)	.ic#
Output listing	.lis, or user-specified

Table 2 HSPICE Output Files and Extensions

Output File Type	Extension
Output status	.st#
Output tables (from .DCMATCH OUTVAR statement)	.dm#
Subcircuit cross-listing	.pa#
Transient analysis measurement results	.mt#
Transient analysis results (from .POST statement)	.tr#
Waveform viewing files from .OPTION WDF argument for use with Synopsys WaveView/SX tools	**_wdf.tr#, *_wdf.sw#, or *_wdf.ac#*

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

b. Requires a .GRAPH statement, or a pointer to a file in the meta.cfg file. The Windows and Linux versions of HSPICE do not generate this file.

AC Analysis Results File

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

AC Analysis Measurement Results File

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

DC Analysis Results File

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

DC Analysis Measurement Results File

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

Digital Output File

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

FFT Analysis Graph Data File

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

Hardcopy Graph Data File

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

Operating Point Information File

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

Operating Point Node Voltages File

HSPICE writes operating point node voltages to file *output_file*.ic*#*, where *#* is 0 to 9999, when the input file includes a `.SAVE` statement. These node voltages are the DC operating point initial conditions.

Output Listing File

The output listing is a text file. It can be named <*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.

The output file includes the following information:

- Name of the simulator used.
- Version of the HSPICE simulator used.
- Synopsys message block.
- Input filename.
- User name.
- License details.
- Copy of the input netlist file.
- Node count.
- Operating point parameters.
- Actual control option values that HSPICE uses for the present simulation (useful when options such as RUNLVL override user-set values.)
- Details of the volt drop, current, and power for each source and subcircuit.
- Low-resolution ASCII plots, originating from a `.PLOT` statement.
- Results of a `.PRINT` statement.
- Results of the `.OPTION` statements.
- Total CPU time (the sum of op point, transient, readin, errchk, setup, and output times).
- In the following snippet of a **.lis* file, you can see that the total cpu time is the sum of op point, transient, readin, errchk, setup and output analysis times.

```
analysis           time    # points   tot. iter  conv.iter
op point           0.00           1          4
transient          0.07      446328         64      32 rev= 3
readin             0.01
errchk             0.01
setup              0.00
output             0.00
total cpu time     0.09 seconds
```

The different analyses stand for time required to:

- Op point: Do operating point analysis.
- Transient: Do transient analysis.
- Readin: Read the user data file and any additional library files, and generate an internal representation of the information.
- Errchk: Check the errors and evaluate the models.
- Output: Prepare the output files and to process all prints and plots.
- Setup: Construct a sparse matrix pointer system.
- Total CPU time is the time taken for the simulation only. It will differ slightly from run to run, even though runs are identical. It will not include memory/disk allocation time or disk I/O time. You can calculate this by subtracting job ended time from job started time.

Output Status File

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

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

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

Output Tables

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

Subcircuit Cross-Listing File

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

Transient Analysis Measurement Results File

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

Transient Analysis Results File

Both HSPICE and HSPICE RF place the results of transient analysis in file `*output_file*.tr#`, where `#` is `0-9999`, as set forth in the `-n` command-line argument. This file lists the numerical results of transient analysis. A `.TRAN` statement in the input file, together with an `.OPTION POST` statement, creates this post-analysis file.

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

Waveform Viewing File

Following use of .OPTION WDF for transient, DC, or AC analyses, for the WDF waveform file, HSPICE automatically appends *_wdf* into the output file root name to specify that it is in WDF format. The file names appear as: **_wdf.tr#, *_wdf.sw#, or *_wdf.ac#*.

For example, the WDF waveform output file will be named: *design_wdf.tr0*.

Working Directory Path Character Limit

HSPICE has a limitation on the number of characters in a working directory path. HSPICE accepts a working directory path of less than 236 characters.

Paths of greater length result in HSPICE exiting with a signal 11 error. To check the length of the working directory path, you can use the UNIX command:

```
% pwd | wc -c
```

3 Startup and Simulation

Describes the invocation commands, and simulation modes.

For descriptions of individual HSPICE commands mentioned in this chapter, see the HSPICE Reference Manual: Commands and Control Options.

These topics are discussed in the following sections:

Running HSPICE Simulations

Use the following syntax to start HSPICE:

```
hspice [-i path/input_file] [-o path/output_file]
   [-n number] [-html path/html_file] [-d]
   [-C path/input_file] [-CC path/input_file] [-I] [-K]
   [-L command_file] [-S] [-mp [number]] [-mt number]
   [-meas measure_file] [-top subcktname] [-hdl filename]
   [-hdlpath pathname] [-vamodel name] [-vamodel name2...]
   [-help] [-doc] [-h] [-v] [-x]
```

For a description of the `hspice` command syntax and arguments, see “HSPICE Command Syntax” in the *HSPICE Reference Manual: Commands and Control Options*.

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

1. Invocation.

   For example, at the shell prompt, enter:

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

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

2. Script execution.

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

3. Licensing.

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

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

4. Simulation configuration.

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

5. Design input.

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

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

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

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

The following is an example of a simple HSPICE input netlist:

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

6. Library input.

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

7. Operating point initialization.

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

8. Analysis.

   HSPICE can perform a single or multipoint sweep of the design and produce one set of output files. In the example above, the `.TRAN` statement causes HSPICE to perform a multipoint transient sweep analysis for 20ns for temperatures ranging from -55° C to 75° C, in steps of 10° C.

9. 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. You can activate multi-processing while running .ALTER cases by entering `hspice -mp` on the command line.

10. Suspending a simulation

Suspend a simulation job by pressing Ctrl-Z. The load sharing facility (LSF) frees up the license for another simulation job. To resume the job, on the same terminal, type either `fg` or `bg`to access a license and continue the simulation.

11. Normal termination.

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

Running HSPICE Simulations on Windows

You can use the MS-DOS command window to run HSPICE in command line mode, similar to UNIX/Linux.

For example:

1. Open an MS-DOS command window (Run > cmd).
2. Enter your case directory.
3. Type the following to invoke HSPICE and view command line help:

   ```
   c:\synopsys\Hspice_release_version\bin\hspice
   ```
4. Or type the following command to run a simulation:

   ```
   C:\synopsys\Hspice_release_version\bin\hspice filename.sp -o
   ```

Running HSPICE RF Simulations

Use the following syntax to invoke HSPICE RF:

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

For a description of the `hspicerf` command syntax and arguments, see "HSPICE RF Command Syntax" in the *HSPICE Reference Manual: Commands and Control Options*.

Running HSPICE Interactively

For a full discussion, refer to Chapter 5, Using Interactive Mode. Interactive mode enables you to use these HSPICE commands at the HSPICE prompt to help you simulate circuits interactively:

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

Starting Interactive Mode

To invoke the interactive mode, enter:

```
hspice -I
```

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

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

```
hspice> save command filename
```

Running a Command File in Interactive Mode

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

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

Quitting Interactive Mode

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

```
hspice> quit
```

Running Multithreading or Multiprocessing HSPICE Simulations

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

By using multithreading, you can speed up simulations with no loss of accuracy. Multithreading gives the best results for circuit designs that contain many voltage- and current-controlled sources such as: VCCS(G), VCVS(E), CCCS(F), CCVS(H), and MOSFET, diode, or BJT models in the netlist.

The requirements for when HSPICE can use more than 1 thread are:

- 512 behavioral elements (VCCSs, VCVSs, CCCSs, CCVSs)
- 512 transistor devices (MOSFETs, BJTs)
- 1024 diodes

If the circuit lacks the required number of active devices, HSPICE automatically uses a single thread. To reduce the threshold number of devices, you can use .OPTION MTTHRESH to set the number of active devices. `MTTHRESH` must = 2 or more. Otherwise, HSPICE MT defaults to 512.

Note:

In the case of too few active devices, a multithreaded simulation may run slower than single-threaded. Whether or not performance degrades depends on many facets of the circuit topology. The default threshold of 512 can be lowered in case the circuit under simulation has less than 512 active elements and still accelerates well with multithreading.

To Run Multithreading

One license is required per two CPUs. To run multithreading on the UNIX/Linux platforms, enter:

```
% hspice -mt number -i input_file -o output_file
```

- Omitting the *number* parameter results in an error message.
- If you specify a *number* larger than the number of available CPUs, HSPICE sets the number of threads equal to the number of available CPUs.

For additional information about command-line options, see "HSPICE Command Syntax" in the *HSPICE Reference Manual: Commands and Control Options*.

In Windows NT Explorer:

1. Click Start > Run, type `cmd`, then press Enter.
2. In the cmd.exe window, `cd` to the case directory.
3. Type:

```
hspice.exe -mt number -i input_file -o output_file
```

In the Synopsys HSPICE User Interface (HSPUI):

1. Select the correct hspice.exe version in the Version combo box.
2. Select MultiCpu Option and select the mt number.
3. Select the input case and run.

Performance Improvement Estimations

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

Where:

`Tserial` represents HSPICE calculations that are not threaded.

`Tparallel` represents threaded HSPICE calculations.

`Ncpu` is the number of CPUs used.

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

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

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

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

Multiprocessing .ALTER Cases, Transient Sweeps, Monte Carlo

While multithreading (`-mt`) is suitable for netlists with huge numbers of active elements (MOSFETs, BJTs or diodes), if you specify the `-mp` *`number`* (multiprocessing) switch, then HSPICE can also take advantage of multiple cores and processors for simulations containing .ALTER cases, transient sweeps, and Monte Carlo analyses. Here Monte Carlo analysis includes DC and transient. You can limit the number of CPUs by specifying the number. If the number is not specified, then HSPICE auto-determines the child processes by the number of available CPUs. This information is printed to the *.st0* file.

.ALTER Cases

In `.ALTER` cases, MP works by creating (forking) child HSPICE processes for each ALTER case up to the number of processors. Each of these HSPICE child processes checks out a license. Additional ALTER cases are divided among the HSPICE processes and run in a serial fashion. For example:

If you start a simulation with a main case and 11 ALTER cases on a 4-CPU system, for example, HSPICE will fork four child processes and divide the main case and ALTER cases among the 4 CPUs, which are then performed serially. The number of available processors and cores can be determined by examining the processor, physical ID, and core ID flags in the `/proc/cpuinfo` file.

HSPICE waits for all the cases to be run then combines the resulting listing files together. *TR0*, *IC0*, *PA0*, *MT0* and other files that are normally generated, one for each ALTER, will still remain separate.

The wall clock time of a multiprocess `.alter` analysis is printed in the *.lis* file.

For example:

```
******  Runtime Statistics for Alter Analysis with MP  ******
        total elapsed time          13 seconds
        job started at     17:21:59 05/16/2008
        job ended   at     17:22:12 05/16/2008
```

Note:

Elapsed time may differ slightly from the total elapsed time of all child processes due to the overhead time cost of output merging.

Known Limitation

The `-mp` feature only operates with `.alter` commands in the top netlist; it does not operate with `.alters` embedded in *.include* files.

Transient Sweeps and Monte Carlo Trials

In transient sweep and Monte Carlo analyses, MP works by creating (forking) child HSPICE processes for multiple jobs based on the partition of a sweep or Monte Carlo loop. Each of these HSPICE processes checks out a license.

For example:

If you start a simulation with 100 trials of transient Monte Carlo on a 4-CPU system. HSPICE forks four child processes and divides the 100 trials among the 4 CPUs, so each child process runs 25 trials. Since one license is checked out at the beginning of the simulation, three additional licenses are checked out and the related information is printed in the *.lis* file.

Multiprocessing Notes

When using `-mp`, note that:

- Unlike `-mt`, the `-mp` option is not available on the Windows platform.
- MP only works on single systems with multiple physical processors and does not distribute jobs across the LAN.
- HSPICE MP will only assign one job to each processor at a time.
- MP takes advantage of multiple processor cores.

Note:

Using MP with MT and Client/Server mode: If both `-mp` and `-mt` are used, then `-mp` takes over and `-mt` is ignored. If both `-mp` and `-CC` are used, then `-CC` takes over and `-mp` is ignored. See Launching the Advanced Client/Server (C/S) Mode.

Using HSPICE in Client/Server Mode

When you run many small simulation cases, you can use the client/server mode to improve performance. This performance improvement occurs because you check out and check in an HSPICE license only once. This is an effective measure when you characterize cells. (For an advanced procedure see Launching the Advanced Client/Server (C/S) Mode.)

To Start Client/Server Mode

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

```
hspice> -C
```

Server

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

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

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

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

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

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

Client

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

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

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

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

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

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

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

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

To Simulate a Netlist in Client/Server Mode

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

```
hspice -C path/input_file
```

Note:

This mode also supports other HSPICE command line options. For a description of the options shown, see “HSPICE Command Syntax” in the *HSPICE Reference Manual: Commands and Control Options*.

To Quit Client/Server Mode

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

```
hspice -C -K
```

Launching the Advanced Client/Server (C/S) Mode

The Advanced Client/Server Mode provides an efficient interface for cell characterization applications. The advanced C/S mode facilitates the Client/Server mode as follows:

- Checks out an HSPICE license once and locks it to do multiple simulations in sequence.
- Reads in the common file only once in multiple simulations with different circuits, when they include a common file, which may a contain subcircuit or model definition.
- Provides an easy-to-use interface.

Command Syntax

Three commands start the HSPICE server, do simulations, and stop the server. The following tables describe the arguments.

1. To start the server, enter:
   ```
   hspice -CC [-port hostname:port_num] [-share inc_file]
   + [-stop stop_time]
   ```
2. To begin a simulation, enter:
   ```
   hspice -CC input_file [-port hostname:port_num] [-o
   output_file]
   ```
3. To stop the HSPICE server, enter:
   ```
   hspice -CC -K [-port hostname:port_num]
   ```

Argument	Description
-CC	Launches the advanced HSPICE C/S mode. After the server starts, it will run in background.
-port hostname: port_num	Starts server on the designated `port hostname:port_num`. If you do not specify this argument, the default port number (25001) is used. If the default port is not available, HSPICE chooses any free port. When you start the server or run a case with `-port hostname:port_num`, the hostname will be ignored. But you can stop the server with `-port hostname:portnum` to implement remote control. HSPICE will print out port information to the screen.

Argument	Description
-share inc_file	Specifies a common file name shared by different circuits.
-stop stop_time	If you do not specify a “stop_time” the server will stop automatically after the default stop_time of 12 hours. The maximum “stop_time” that can be specified is 12 hours.
-o output_file	Specifies the output file name. If an “output_file” is not specified, HSPICE uses the input root filename as the output file root filename.
-K	Shuts down the client server.

Application Instances

In the following instances, assume there are 5 netlist files (t1.sp, t2.sp, … t5.sp) to be run, and this group includes a common file. Also assume that all 5 files are located in the same directory: /home1/user1/test/testcase.

t1.sp

```
V1 1 0 dc 1.05
V2 2 0 dc 0
.temp 125
.inc "/home1/user1/test/model/model_file"
.inc "101.spc"
.end
```

t2.sp

```
V1 1 0 dc 1.05
V2 2 0 dc 0
.temp 125
.inc "/home1/user1/test/model/model_file"
.inc "102.spc"
.end
```

```
...
```

t5.sp

```
V1 1 0 dc 1.05
V2 2 0 dc 0
.temp 125
.inc "/home1/user1/test/model/model_file"
.inc "105.spc"
.end
```

Using the following commands, you can invoke the HSPICE C/S mode to run this group of cases. The work path is: /home1/user1/test/testcase

1. Start the HSPICE server on the default port and read in the common file:
   `hspice -CC -share /home1/user1/test/model/model_file`
2. Run a simulation on the default port without reading in the common file again.
   `hspice -CC t1.sp`

   Explanation: Since the `.inc "/home1/user1/test/model/model_file"` statement appears in each netlist, it is not read in again because the server has already processed the information.
3. Repeat Step 2 until all cases are simulated.
4. Exit the HSPICE Client/Server mode.
   `hspice -CC -K`

The sequence of commands is:

```
hspice -CC -share  /home1/user1/test/model/model_file
hspice -CC t1.sp
hspice -CC t2.sp
hspice -CC t3.sp
hspice -CC t4.sp
hspice -CC t5.sp
hspice -CC -K
```

Notes

- If you start the server and run simulations by Perl script, use the system ($cmd) instead of '$cmd' to avoid hanging the server. For example,

  ```
  #!/depot/perl-5.8.3/bin/perl -w
  ...
   $cmd = "hspice -CC -share inc_file ";
  ...
  ```

- To use multiple servers, you need to specify multiple ports. If you submit several scripts to start multiple servers, you need to specify multiple ports. If you do not designate port numbers to a multiple-cpu machine or to a machine in computer farm environment, only one server will start on the default port number. If the default port is not available, HSPICE chooses any free port. HSPICE also prints out port information. The printed message is similar to `"Server is started on port=port_num"`. To assure that

the simulation is run successfully in a different script, add `-port port_num`.
For example,

```
#!/depot/perl-5.8.3/bin/perl -w
##start server without designated port, redirect output
#information
$cmd = "hspice -CC >& log ";
system($cmd) ;
##get the port_num on which server is started
$portnum=`grep port= log|awk {{print $6}}`;
##do simulation
$cmd1 = "hspice -CC test1.sp -port $portnum";
system($cmd1) ;
...
##stop server
$cmdn ="hspice -CC -K -port $portnum";
system($cmdn) ;
```

- To avoid redefinition errors, verify that the common file both in "`-share` *`inc_file`*" and in "`.inc` *`inc_file`*" of every netlist has the same absolute path and file name. For example, there are 5 netlist files, t1.sp, t2.sp, t3.sp, t4.sp, and t5.sp to be run and this group of netlists includes a common file. Assume that all these 5 files are in the same directory /home1/user1/test/testcase. The following is the correct usage.

```
hspice -CC -share /home1/user1/test/model/model_file
hspice -CC t1.sp
hspice -CC t2.sp
hspice -CC t3.sp
hspice -CC t4.sp
hspice -CC t5.sp
hspice -CC -K
```

Each of the netlists includes `.inc "/home1/tom/test/model/model_file"` In every case, the absolute path name of the common file in `.inc "/home1/user1/test/model/model_file"` is the same as the absolute path name of the common file specified by `-share /home1/user1/test/model/model_file`. The common file `/home1/user1/test/model/model_file` will only be read in once and `.inc "/home1/user1/test/model/model_file"` will be ignored in every case.

Running HSPICE to Calculate New Measurements

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

To Calculate New Measurements

To get new measurements from a previous simulation, enter:

```
hspice -meas measure_file -i wavefile [-o outputfile]
```

For a description of the options shown, see HSPICE Command Syntax in the *HSPICE Reference Manual: Commands and Control Options*.

4
Input Netlist and Data Entry

Describes the input netlist file and methods of entering data.

For descriptions of individual HSPICE commands referenced in this chapter, see Chapter 2, HSPICE and HSPICE RF Netlist Commands, in the *HSPICE Reference Manual: Commands and Control Options*.

The following topics are discussed in this chapter:

- Input Netlist File Guidelines
- Input Netlist File Composition
- Using Subcircuits
- Subcircuit Call Statement Discrete Device Libraries
- Post-Layout Back-Annotation

Input Netlist File Guidelines

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

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

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

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

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

Note:

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

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

Input Line Format

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

    Any valid string of characters between two token delimiters is a token. You can use a character string as a parameter value in HSPICE, but not in HSPICE RF. See Delimiters on page 46.
- An input statement, or equation can be up to 1024 characters long.
- HSPICE or HSPICE RF ignores differences between upper and lower case in input lines, except in quoted filenames.
- To continue a statement on the next line, enter a plus (+) sign as the first non-numeric, non-blank character in the next line.
- To indicate “to the power of” in your netlist, use two asterisks (`**`). For example, `2**5` represents two to the fifth power (2^5).
- To continue all HSPICE or HSPICE RF statements, including quoted strings (such as paths and algebraics), use a single whitespace followed by a backslash (\) or a double backslash (\\) at the end of the line that you want to continue.
  - A single backslash preserves white space.
- All characters after the listed statement lines will be ignored:
  - `.include 'filename'`
  - `.lib 'filename' corner`

   - `.enddata`, `.end`, `.endl`, `.ends` and `.eom`

     For example:

```
.include 'biasckt.inc';    $ semicolon ignored
.lib 'mos25l.l' tt,        $ comma ignored
```

- Parameter names must begin with an alphabetic character, but thereafter can contain numbers and some special characters. See Special Characters.
   - When you use an asterisk (`*`) or a question mark (`?`) with a `.PRINT`, `.PROBE`, `.LPRINT` (HSPICE RF), or `.CHECK` (HSPICE RF) statement, HSPICE or HSPICE RF uses the character as a wildcard. For additional information, see Using Wildcards on Node Names on page 61.
   - When you use curly braces (`{ }`), HSPICE converts them to square brackets (`[ ]`) automatically.
   - Names are input tokens. Token delimiters must precede and follow names. See Delimiters.
   - Names can be up to 1024 characters long and are not case-sensitive.
   - Do not use any of the time keywords as a parameter name or node name in your netlist.
   - The following symbols are reserved operator keywords:

     `( ) = " ‘`

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

Special Characters

The following table lists the special characters that can be used as part of node names, element parameter names, and element instance names. For detailed discussion, see the appropriate sections in this chapter.

Note:

To avoid unexpected results or error messages, do not use the following mathematical characters in a parameter name in HSPICE: *** - + ^** and **/**.

Table 3 HSPICE/ HSPICE RF Netlist Special Characters

Special Character **Note: P = character is legal anywhere in the string, first or included**		**Node Name**	**Instance Name (cannot be the first character; element key letter only)**	**Parameter Name (cannot be the first character, element key letter only)**	**Delimiters**
~	tilde	HSPICE P, Included only for HSPICE RF	Included only	Included only	n/a
!	exclamation point	P	Included only	Included only	n/a
@	at sign	P	included only	Included only	n/a
#	pound sign	P	Included only	Included only	n/a
$	dollar sign	Included only (avoid if after a number in node name)	Included only	Included only	In-line comment character
%	percent	HSPICE P, Included only for HSPICE RF	Included only	HSPICE: included only, Illegal in HSPICE RF	n/a
^	caret	HSPICE P, included only for HSPICE RF	Included only	HSPICE: included only (avoid usage), Illegal in HSPICE RF	“To the power of”, i.e., 2^5, two raised to the fifth power
&	ampersand	HSPICE P, Included only for HSPICE RF	Included only	Included only	n/a

Table 3 HSPICE/ HSPICE RF Netlist Special Characters

Special Character **Note: P = character is legal anywhere in the string, first or included**		**Node Name**	**Instance Name (cannot be the first character; element key letter only)**	**Parameter Name (cannot be the first character, element key letter only)**	**Delimiters**
*	asterisk	HSPICE: included only (avoid using * in node names), Illegal for HSPICE RF	Included only	HSPICE: included only (avoid using in parameter names), Illegal in HSPICE RF	Comment in both HSPICE/HSPICE RF. Wildcard character. Double asterisk (**) is “To the power of”.
( )	parentheses	Illegal	Illegal	Illegal	Token delimiter
-	minus	HSPICE: included only HSPICE RF P	Included only	Included only (avoid usage)	n/a
_	underscore	P	Included only	Included only	n/a
+	plus sign	HSPICE: included only HSPICE RF P	Included only	HSPICE: included only (avoid usage); Illegal in HSPICE RF	Continues previous line, except for quoted strings (expressions, paths, algebraics)
=	equals	Illegal	Illegal	optional in .PARAM statements	Token delimiter
< >	less/more than	HSPICE P, included only for HSPICE RF	Included only	Included only	n/a
?	question mark	HSPICE P, Illegal for HSPICE RF	Included only	Included only	Wildcard in character in both HSPICE and HSPICE RF
/	forward slash	P	Included only	Illegal	n/a

Table 3 HSPICE/ HSPICE RF Netlist Special Characters

Special Character **Note:** P = character is legal anywhere in the string, first or included		Node Name	Instance Name (cannot be the first character; element key letter only)	Parameter Name (cannot be the first character, element key letter only)	Delimiters
{ }	curly braces	HSPICE: included only, converts { } to [ ] No conversion for HSPICE RF	Included only	Included only	Auto-converts to square brackets ([ ]) Single ({) or (}) can be used in Variation Blocks
[ ]	square brackets	Included only	Included only	Included only	n/a
\	backslash (requires a whitespace before to use as a continuation)	HSPICE: included only, HSPICE RF P	Included only	Illegal in HSPICE, Included only in HSPICE RF	Continuation character for quoted strings (preserves whitespace)
\\	double backslash (requires a whitespace before to use as a continuation)	HSPICE: included only, HSPICE RF P	Illegal	Illegal	Continuation character for quoted strings (preserves whitespace)
|	pipe	HSPICE P, Included only, HSPICE RF	Included only	Included only	n/a
,	comma	Illegal	Illegal	Illegal	Token delimiter
.	period	Illegal	Included only	Included only	Netlist keyword, (i.e., .TRAN, .DC, etc.). Hierarchy delimiter when used in node names

Table 3 HSPICE/ HSPICE RF Netlist Special Characters

Special Character **Note: P = character is legal anywhere in the string, first or included**		**Node Name**	**Instance Name (cannot be the first character; element key letter only)**	**Parameter Name (cannot be the first character, element key letter only)**	**Delimiters**
:	colon	Included only	Included only	Included only	Delimiter for element attributes
;	semi-colon	Included only	Included only	Included only	n/a
" "	double-quotes	Illegal	Illegal	Illegal	Expression and filename delimiter
‘ ’	single quotes	Illegal	Illegal	Illegal	Expression and filename delimiter
	Blank (whitespace)	Use before \ or \\ line continuations			Token delimiter
Tab	Tab				Token delimiter

First Character

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

Table 4 First Character Descriptions

Line	**If the First Character is...**	**Indicates**
First line of a netlist	Any character	Title or comment line. The first line of an included file is a normal line and not a comment.
Subsequent lines of netlist, and all lines of included files	. (period)	Netlist keyword. For example, .TRAN 0.5ns 20ns

Table 4 First Character Descriptions (Continued)

Line	If the First Character is...	Indicates
	`c, C, d, D, e, E, f, F, g, G, h, H, i, I, j, J, k, K, l, L, m, M, q, Q, r, R, s, S, v, V, w, W`	Element instantiation
	`*` (asterisk)	Comment line
	`+` (plus)	Continues previous line

Delimiters

- An input token is any item in the input file that HSPICE or HSPICE RF recognizes. Input token delimiters are: tab, blank (whitespace), 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.

Instance Names

The names of element instances begin with the element key letter (see Table 5), except in subcircuits where instance names begin with X. (Subcircuits are sometimes called macros or modules.) Instance names can be up to 1024 characters long. In HSPICE, the `.OPTION LENNAM` defines the length of names in printouts (default=16).

Table 5 Element Identifiers

Letter (First Char)	Element	Example Line
`B`	IBIS buffer	`b_io_0 nd_pu0 nd_pd0 nd_out nd_in0 nd_en0 nd_outofin0 nd_pc0 nd_gc0`
`C`	Capacitor	`Cbypass 1 0 10pf`

Table 5 Element Identifiers (Continued)

Letter (First Char)	Element	Example Line
`D`	Diode	`D7 3 9 D1`
`E`	Voltage-controlled voltage source	`Ea 1 2 3 4 K`
`F`	Current-controlled current source	`Fsub n1 n2 vin 2.0`
`G`	Voltage-controlled current source	`G12 4 0 3 0 10`
`H`	Current-controlled voltage source	`H3 4 5 Vout 2.0`
`I`	Current source	`I A 2 6 1e-6`
`J`	JFET or MESFET	`J1 7 2 3 GAASFET`
`K`	Linear mutual inductor (general form)	`K1 L1 L2 1`
`L`	Linear inductor	`LX a b 1e-9`
`M`	MOS transistor	`M834 1 2 3 4 N1`
`P`	Port	`P1 in gnd port=1 z0=50`
`Q`	Bipolar transistor	`Q5 3 6 7 8 pnp1`
`R`	Resistor	`R10 21 10 1000`
`S`	S parameter element	`S1 nd1 nd2 s_model2`
`V`	Voltage source	`V1 8 0 5`
`T,U,W`	Transmission Line	`W1 in1 0 out1 0 N=1 L=1`
`X`	Subcircuit call	`X1 2 4 17 31 MULTI WN=100 LN=5`

Hierarchy Paths

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

Numbers

You can enter numbers as integer, floating point, floating point with an integer exponent, or integer or floating point with one of the scale factors listed in Table 6.

Table 6 Scale Factors

Scale Factor	Prefix	Symbol	Multiplying Factor
T	tera	T	1e+12
G	giga	G	1e+9
MEG or X	mega	M	1e+6
K	kilo	k	1e+3
MIL	n/a	none	25.4e-6
M	milli	m	1e-3
U	micro	μ	1e-6
N	nano	n	1e-9
P	pico	p	1e-12
F	femto	f	1e-15
A	atto	a	1e-18

Note:

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

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

Parameters and Expressions

- Parameter names in HSPICE or HSPICE RF use HSPICE name syntax rules, except that names must begin with an alphabetic character. The other characters must be either a number, or a special character. See Table 3 on page 42 in the Special Characters section for a listing of legal parameter names. For example, a “`%`” is legal if included in HSPICE, but illegal in HSPICE RF.
- To define parameter hierarchy overrides and defaults, use the `.OPTION PARHIER=global | local` statement.
- If you create multiple definitions for the same parameter or option, HSPICE or HSPICE RF uses the last parameter definition or `.OPTION` statement, even if that definition occurs later in the input than a reference to the parameter or option. HSPICE or HSPICE RF does not warn you when you redefine a parameter.
- You must define a parameter before you use that parameter to define another parameter.

- When you select design parameter names, be careful to avoid conflicts with parameterized libraries.
- To delimit expressions, use single or double quotes.
- Expressions cannot exceed 1024 characters.
- For improved readability, use a double backslash preceded by a whitespace (\\) 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.
- Limitation 1: If a parameter is defined as an expression containing output signals such as v(node) or i(element), this parameter only can be used in an element value expression directly, and can not be evaluated to another parameter.

  For example, the following is correct:

  ```
  .param a='2*sqrt(V(p,n))'
    r1 p n '1k+a'
  ```

  The following definition is correct, but this definition points up the limitation and is not permitted because HSPICE generates an incorrect result.

  ```
  .param a='2*sqrt(v(p,n))'
  .param b='a+1'
   r1 p n '1k+b'
  ```

  It is best to use a user-defined function to replace the previous example, so that all of r1 and r2 are correct.

  ```
  .param a(x)='2*sqrt(x)'
  .param b(x)='a(x)+1'
  r1 p n '1k+a(V(p,n))'
  r2 p n '1k+b(V(p,n))'
  ```

- Limitation 2: If an expression containing output signals such as v(node) or i(element) is used in an element value directly, the element only can be R, C, L, E, or G.

Correct

```
G1 1 0 cur='((1-(a0*v(gate)))/b0)'
```

Incorrect

```
I1 1 0 cur='((1-(a0*v(gate)))/b0)'
```

Input Netlist File Structure

An input netlist file should consist of one main program. In HSPICE, an input netlist can contain one or more optional submodules. HSPICE uses 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`,and `.LIB`) to structure the input netlist file modules. These statements can call netlists, model parameters, test vectors, analysis, and option macros into a file from library files or other files. The input netlist file also can call an external data file, which contains parameterized data for element sources and models. You must enclose the names of included or internally-specified files in single or double quotation when they begin with a number (0-9).

Schematic Netlists

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

The process of creating a schematic involves:

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

Table 7 Input Netlist File Sections

Sections	Examples	Definition
Title	.TITLE	The first line in the netlist is the title of the input netlist file. (HSPICE)

Table 7 Input Netlist File Sections (Continued)

Sections	Examples	Definition
Set-up	.OPTION .IC or .NODESET, .PARAM, .GLOBAL	Sets conditions for simulation. Initial values in circuit and subcircuit. Set parameter values in the netlist. Set node name globally in netlist.
Sources	Sources and digital inputs	Sets input stimuli (I or V element).
Netlist	Circuit elements .SUBKCT, .ENDS, or .MACRO, .EOM	Circuit for simulation. Subcircuit definitions.
Analysis	.DC, .TRAN, .AC, and so on. .SAVE and .LOAD .DATA, .TEMP	Statements to perform analyses. Save and load operating point information. (HSPICE) Create table for data-driven analysis. Set temperature analysis.
Output	.PRINT, .PROBE, .MEASURE	Statements to output variables. Statement to evaluate and report user-defined functions of a circuit.
Library, Model and File Inclusion	.INCLUDE	General include files.
	.MALIAS	Assigns an alias to a diode, BJT, JFET, or MOSFET. (HSPICE)
	.MODEL	Element model descriptions.
	.LIB	Library.
	.OPTION SEARCH	Search path for libraries and included files. (HSPICE)
	.OPTION BRIEF= 1 and .OPTION BRIEF= 0	Control printback to output listing. (HSPICE)

Table 7 Input Netlist File Sections (Continued)

Sections	Examples	Definition
Alter blocks (HSPICE Only)	.ALIAS, .ALTER, .DEL LIB	Renames a previous model. Sequence for in-line case analysis. Removes previous library selection.
End of netlist	.END	Required statement; end of netlist.

Input Netlist File Composition

The HSPICE and HSPICE RF circuit description syntax is compatible with the SPICE input netlist format. Figure 5 shows the basic structure of an input netlist.

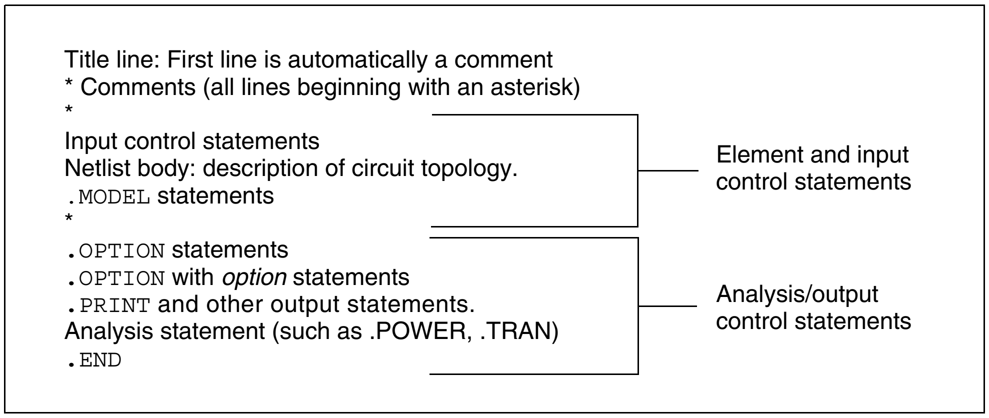

Title line: First line is automatically a comment
* Comments (all lines beginning with an asterisk)
*
Input control statements
Netlist body: description of circuit topology.
.MODEL statements
*
.OPTION statements
.OPTION with option statements
.PRINT and other output statements.
Analysis statement (such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

Figure 5 Basic Netlist Structure

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

To create the circuit:

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

Insert the pulse source to the inverter input.

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

```
* Sample inverter circuit
* **** MOS models *****
.MODEL n1 NMOS LEVEL=3 THETA=0.4 ...
.MODEL p1 PMOS LEVEL=3 ...
* ***** Define power supplies and sources *****
VDD VDD 0 5
VPULSE VIN 0 PULSE 0 5 2N 2N 2N 98N 200N
VGND GND 0 0
* ***** Actual circuit topology *****
M1 VOUT VIN VDD VDD p1
M2 VOUT VIN GND GND n1
* ***** Analysis statement *****
.TRAN 1n 300n
* ***** Output control statements *****
.OPTION POST PROBE
.PROBE V(VIN) V(VOUT)
.END
```

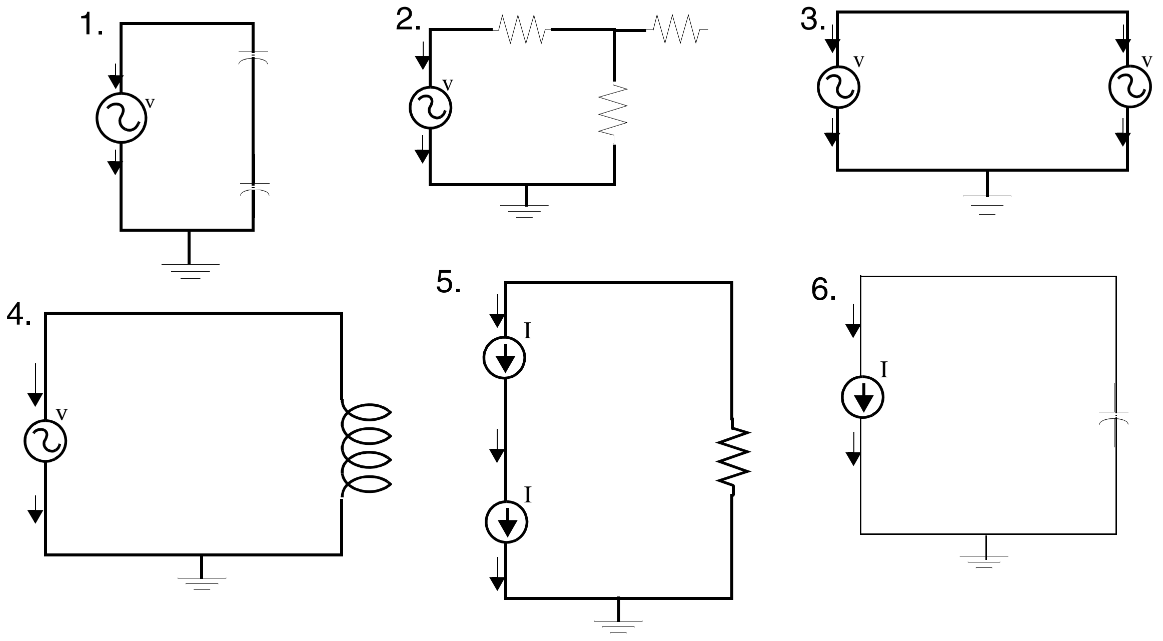
For a description of individual commands used in HSPICE or HSPICE RF netlists, see the HSPICE and HSPICE RF Netlist Commands chapter in the *HSPICE Reference Manual: Commands and Control Options*.

HSPICE Topology Rules

When constructing the circuit description HSPICE does not allow certain topologies. The following rules apply:

1. Every node must have a DC path to ground (all circuits must a have at least one ground not in series with a capacitor).
2. No dangling nodes (all nodes must have at least two connections).
3. No voltage loops (no voltage sources in parallel with no other elements).
4. No ideal voltage source in closed inductor loop.
5. No stacked current sources (no current sources in series).

6. No ideal current source in closed capacitor loop.



Title of Simulation

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

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

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

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

Comments and Line Continuation

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

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

You can place comment statements anywhere in the circuit description.

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

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

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

For example,

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

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

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

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

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

Element and Source Statements

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

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

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

Table 8 lists the parameters of an element statements.

Table 8 Element Parameters

Parameter	Description
elname	Element name that cannot exceed 1023 characters, and must begin with a specific letter for each element type: B IBIS buffer (HSPICE Only) C Capacitor D Diode E,F,G,H Dependent current and voltage sources I Current (inductance) source J JFET or MESFET K Mutual inductor L Inductor model or magnetic core mutual inductor model M MOSFET Q BJT P Port R Resistor S, S-parameter mode T, U, W Transmission line V Voltage source X Subcircuit call
node1 ...	Node names identify the nodes that connect to the element. The node name begins with a letter and can contain a maximum of 1023 characters. For a listing of legal and illegal special characters that can be used in node names, see the Special Characters section, Table 3 on page 42.

Table 8 Element Parameters (Continued)

Parameter	Description
mname	HSPICE or HSPICE RF requires a model reference name for all elements, except passive devices.
pname1 ...	An element parameter name identifies the parameter value that follows this name.
expression	Any mathematical expression containing values or parameters, such as param1 * val2
val1 ...	Value of the *pname1* parameter, or of the corresponding model node. The value can be a number or an algebraic expression.
M=val	Element multiplier. Replicates *val* element times, in parallel. Do not assign a negative value or zero as the M value.

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

Example 1

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

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

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

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

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

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

Example 2

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

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

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

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

Defining Subcircuits

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

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

Node Name (or Node Identifier) Conventions

Nodes are the points of connection between elements in the input netlist. You can use either names or numbers to designate nodes. Node numbers can be from 1 to 999999999999999 (1-1e16); node number 0 is always ground. HSPICE or HSPICE RF ignores letters that follow numbers in node names. When the node name begins with a letter or a valid special character, the node name can contain a maximum of 1024 characters.

The following are conventions regarding node names:

- In addition to letters and digits, node names can include, but NOT always begin with some special characters. See the Special Characters section, Table 3 on page 42. Note that use of special characters in node names varies between HSPICE and HSPICE RF.
- If you use braces { } in node names, HSPICE or HSPICE RF changes them to brackets [ ].
- You cannot use the following characters in node names: `() ,=,‘, <blank>`
- You should avoid using the dollar sign ($) after a numerical digit in a node name because HSPICE assumes whatever follows the “$” symbol is an in-line comment (see Comments and Line Continuation on page 55 for additional information). It can cause error and warning messages depending on where the node containing the “$” is located. For example, HSPICE generates an error indicating that a resistor node is missing:

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

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

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

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

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

- To make node names global across all subcircuits, use a `.GLOBAL` statement (see Global Node Names on page 65).
- The `0`, `GND`, `GND!`, and `GROUND` node names all refer to the global HSPICE or HSPICE RF ground. Simulation treats nodes with any of these names as a ground node, and produces v(0) into the output files. Besides these ground nodes, all node names are regarded as separate nodes.For example, 0 and 0.3 are different nodes, 1A and 1 are different nodes, 2~ and 2 are different nodes.

Using Wildcards on Node Names

You can use wildcards to match node names.

- 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).
  - And `.PRINT v(o*)` prints the voltages for all nodes whose names start with `o`; if your netlist contains nodes named `in` and `out`, this example prints only the `v(out)` voltage. For example, the following prints the results of a transient analysis for the voltage at the matched node name.

    ```
    .PRINT TRAN V(9?t*u)
    ```
- Wildcards must begin with a letter or a number; for example,

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

Here are some practical applications for these wildcards:

- If your netlist includes a resistor named `r1` and a voltage source named `vin`, then `.PRINT i(*)` prints the current for both elements `i(r1)` and `i(vin).`
- The statement `.PRINT v(o*)` prints the voltages for all nodes whose names start with o; if your netlist contains nodes named `in` and `out`, this example prints only the `v(out)` voltage.
- If your netlist contains nodes named `0`, `1`, 2, and `3`, then `.PRINT v(0,*)` or `.PRINT v(0 *)` prints the voltage between node `0` and each of the other nodes: `v(0,1)`, `v(0,2)`, and `v(0,3).`
- Wildcards can be used to set initial conditions in `.IC` and `.NODESET` statements. Node names including wildcards in `.IC` and `.NODESET` must start with “v()”. For example, `.NODESET v(a*)=5`.

Examples

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

- Probe node voltages for nodes at all levels.

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

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

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

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

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

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

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

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

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

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

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

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

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

- Set transient initial conditions to all nodes. For example:

  ```
   .ic v(*)=val
  ```

  initializes nodal voltages for DC operating point analysis to nodes whose names start with “a”, such as: a1, a2, a11, a21.

  ```
     .nodeset v(a*)=val
  ```

  If one node is specified and it also matches a wildcard in `.IC` or `.NODESET`, then the specified value will override the wildcard's value. For example, if `.ic v(a*)=5 v(a1)=10`, then `v(a1)` will be 10.

Element, Instance, and Subcircuit Naming Conventions

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

Element names in HSPICE or HSPICE RF begin with a letter designating the element type, followed by up to 1023 alphanumeric characters. Element type letters are R for resistor, C for capacitor, M for a MOSFET device, and so on (see Element and Source Statements on page 57).

Subcircuit Node Names

HSPICE assigns two subcircuit node names.

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

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

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

  Note:

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

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

Path Names of Subcircuit Nodes

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

You can use path names in `.PRINT, .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 6 on page 64.

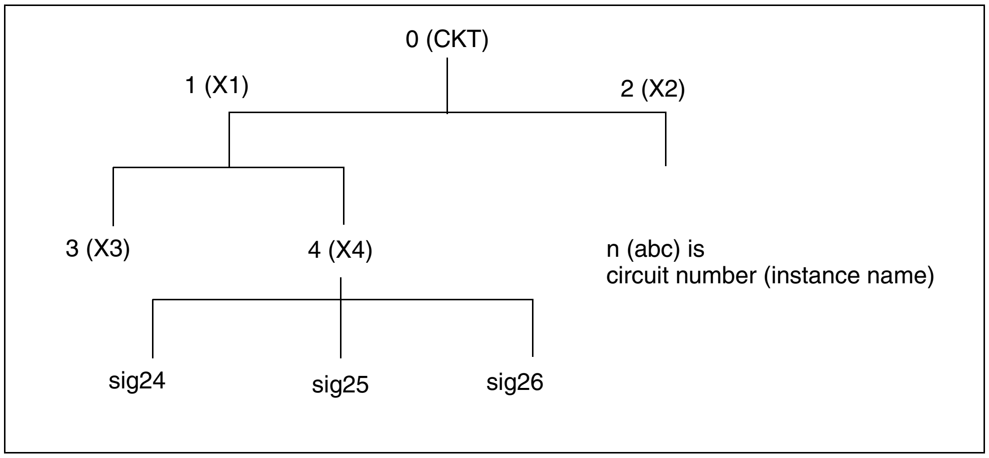


Figure 6 Subcircuit Calling Tree, with Circuit Numbers and Instance Names

In Figure 6, the path name of the `sig25` node in the `X4` subcircuit is `X1.X4.sig25`. You can use this path in HSPICE or HSPICE RF statements, such as:

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

Abbreviated Subcircuit Node Names

In HSPICE, you can use circuit numbers as an alternative to path names, to reference nodes or elements in `.PRINT`, `.NODESET`, or `.IC` statements. Compiling the circuit assigns a circuit number to all subcircuits, creating an abbreviated path name:

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

Note:

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

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

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

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

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

Automatic Node Name Generation

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

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

Global Node Names

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

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

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

Circuit Temperature

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

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

Data-Driven Analysis

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

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

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

For more details about using the `.DATA` statement in different types of analysis, see Chapter 12, Initializing DC/Operating Point Analysis and Chapter 14, Transient Analysis.

Library Calls and Definitions

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

In HSPICE, you can also place a `.LIB` call statement in an `.ALTER` block.

Library Building Rules

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

Automatic Library Selection (HSPICE)

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 or output buffer subcircuit in a file named iobuf.inc, you can create three copies of the file, to simulate fast, slow and typical corner cases. Each file contains different HSPICE transistor models, representing the different process corners. Store these files (all named iobuf.inc) in separate directories.

Defining Parameters

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

- Predefined Analysis
- Measurement Parameters

Predefined Analysis

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

HSPICE or HSPICE RF supports the following predefined analysis parameters:

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

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

- frequency
- time

Measurement Parameters

A `.MEASURE` statement produces a measurement parameter. In general, the rules for measurement parameters are the same as those for standard parameters. However, measurement parameters are not defined in a `.PARAM` statement, but directly in the `.MEASURE` statement. For more information, see .MEASURE Parameter Types on page 318.

Outputting Pass/Fail Measure Data

You can use `.measure` to create a logic equation that describes the pass/fail condition. It will output a 0/1 value for pass/fail in the *.mt0* file. For example:

```
.meas m1 find v(1) at 10n
.meas m2 find v(2) at 10n
.meas pass param="(m1 > 1) && (m2 < 2)"
```

Altering Design Variables and Subcircuits

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

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

Using Multiple .ALTER Blocks

This section does not apply to HSPICE RF.

- For the first simulation run, HSPICE reads the input file, up to the first `.ALTER` statement, and performs the analyses up to that `.ALTER` statement.
- After it completes the first simulation, HSPICE reads the input between the first `.ALTER` statement, and either the next `.ALTER` statement or the `.END` statement.
- HSPICE then uses these statements to modify the input netlist file.

- HSPICE then resimulates the circuit.
- For each additional `.ALTER` statement, HSPICE performs the simulation that precedes the first `.ALTER` statement.
- HSPICE then performs another simulation, using the input between the current `.ALTER` statement, and either the next `.ALTER` statement or the `.END` statement.

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

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

Connecting Nodes

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

Deleting a Library

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

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

Ending a Netlist

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

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

Condition-Controlled Netlists (IF-ELSE)

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

```
.if (condition1)
   <statement_block1>

# The following statement block in {braces} is
# optional, and you can repeat it multiple times:
{ .elseif (condition2)
   <statement_block2>
}

# The following statement block in [brackets]
# is optional, and you cannot repeat it:
[ .else
   <statement_block3>
]
.endif
```

The following example provides a quick overview of using the IF-ELSE construct:

```
*
.tran 1n 100ns
.option post
.param  variable = 2
vin1 source 0 0 pwl 0ns 0v 20ns 0v 21ns 5v 30ns 5v 31ns 0v 40ns
+ 0v 41ns 5v 50ns 5v 70ns 5v 71ns 0v 80ns 0v 81ns 5v
+ 90ns 5v 91ns 0v 100ns 0v
Rin source 1 1000
x1 1 2 resistor
r1 2 0 1k
.subckt resistor 1 2
.if (variable==1)
r22 1 2 1k
.elseif (variable==2)
r22 1 2 2k
.else
r22 1 2 3k
.endif
.ends
.print v(2) i(1)
.alter
.param variable=1
.alter
.param variable=3
.end
```

Guidelines for Using IF-ELSE Blocks

The following guidelines aid in usage of the .IF, .ELSE-IF, or .ELSE.

- 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 (not HSPICE RF) analysis and output statements. The exceptions are:
  - `.END`, `.ALTER`, `.GLOBAL`, `.DEL LIB`, `.MALIAS`, `.ALIAS`, `.LIST`, `.NOLIST`, and `.CONNECT` statements.
  - `search`, `d_ibis`, `d_imic`, `d_lv56`, `biasfi`, `modsrh`, `cmiflag`, `nxx`, and `brief` options.
- You can include `IF-ELSEIF-ELSE` statements in subcircuits and subcircuits in `IF-ELSEIF-ELSE` statements.
- You can use IF-ELSEIF-ELSE blocks to select different submodules to structure the netlist (using `.INC`, `.LIB`, and `.VEC` statements).

- If two or more models in an IF-ELSE block have the same model name and model type, they must also be the same revision level.
- Parameters in an IF-ELSE block do not affect the parameter value within the condition expression. HSPICE or HSPICE RF updates the parameter value only after it selects the IF-ELSE block.
- You can nest IF-ELSE blocks.
- You can include `.SUBCKT` and `.MACRO` statements within an IF-ELSE block.
- You can include an unlimited number of `ELSEIF` statements within an IF-ELSE block.
- You cannot use an IF-ELSE block within another statement. In the following example, HSPICE or HSPICE RF does not recognize the IF-ELSE block as part of the resistor definition:

```
r 1 0
   .if (r_val>10k)
   + 10k
   .else
   + r_val
   .endif
```

Using Subcircuits

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

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

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

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

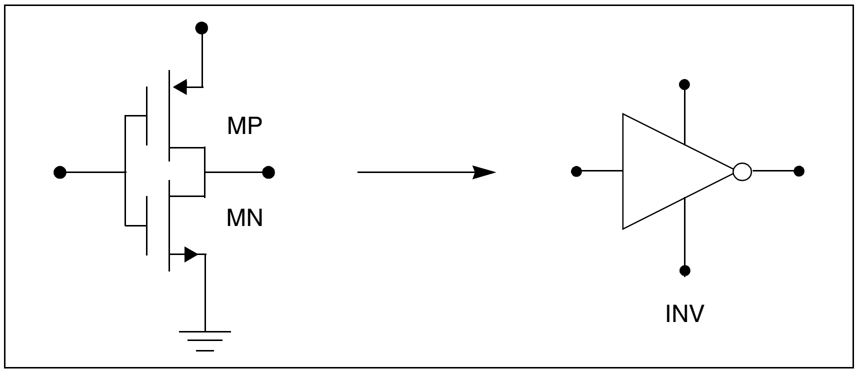


Figure 7 Subcircuit Representation

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

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

Note:

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

Hierarchical Parameters

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

M (Multiply) Parameter

The most basic HSPICE or HSPICE RF 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. This, in effect, creates parallel copies of the element.

For example, if you have an invertor and specify M=2, then HSPICE multiplies the internal component by 2. The M parameter multiplies the internal component values, which, in effect, creates parallel copies of the element. To simulate 32 output buffers switching simultaneously, you need to place only one subcircuit; for example,

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

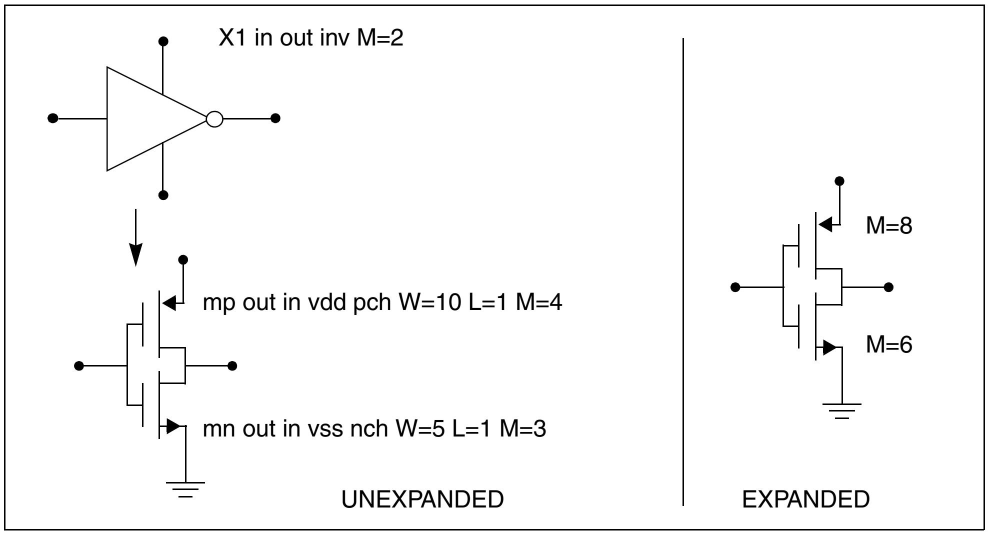


Figure 8 How Hierarchical Multiply Works

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

S (Scale) Parameter

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

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

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

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

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

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

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

In this example:

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

Using Hierarchical Parameters to Simplify Simulation

You can use the hierarchical parameter to simplify simulations. An example is shown in the following listing and Figure 9 on page 77.

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

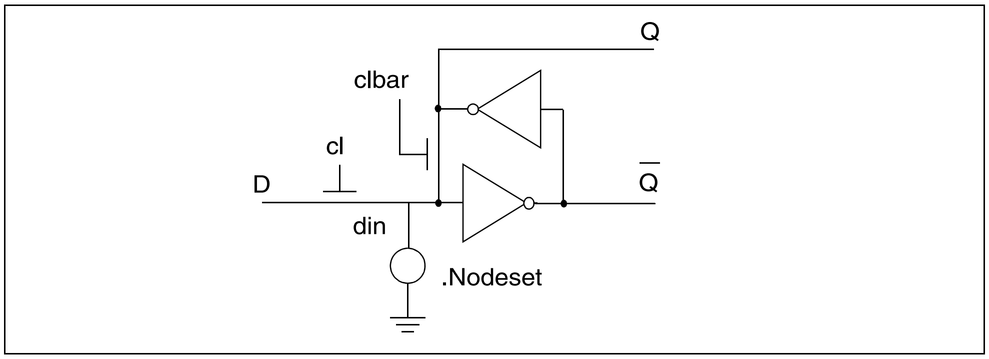


Figure 9 D Latch with Nodeset

HSPICE does not limit the size or complexity of subcircuits; they can contain subcircuit references, and any model or element statement. However, in HSPICE RF, you cannot replicate output commands within subcircuit definitions.

To specify subcircuit nodes in `.PRINT` statements, specify the full subcircuit path and node name.

You can print the hierarchical parameter using a `.print`/`.probe`/`.measure` statement. For example:

```
x1 1 0 sub1
.subckt sub1 n1 n2
.param p1=1
...
.ends
.tran ...
.print tran par('x1.p1')
.measure tram m1 param='x1.p1'
```

Undefined Subcircuit Search (HSPICE)

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

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

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

Troubleshooting Subcircuit Node Issues

Duplicate Node Message

In the `.SUBCKT` definition, if HSPICE finds two or more node names that are similar, it will issue the following error:

```
**error** subcircuit definition duplicates node x1234        }

**error**  .ends  card missing at readin
```

In the following example, for the .SUBCKT definition, the node “in” is defined twice. The second definition of the node “in” is a duplicate of the first node, which is not allowed in HSPICE.:

```
.subckt ABC in in out
.
.
.ends
```

Duplicating Ports

To create duplicate ports in HSPICE you can define them in the instance definition of the subcircuit.

For example:

```
.subckt DUP A B C D
.
.
.ends
```

If you want to make node “B” a duplicate of node “A” then the instance definition should look like:

```
XDUP A A C D DUP
```

Using Assigned Circuit Numbers instead of Full Path Node Names

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

HSPICE assigns a circuit number to all subcircuits in the netlist. The circuit number can be used to abbreviate the path name, as follows:
`<subckt-num>:<name>`

You can find the abbreviated path name information in the **.pa0* file. Nodes that are not in a subcircuit have a 0 prefix which references the main or top level circuit.

Subcircuit Call Statement Discrete Device Libraries

The Synopsys Discrete Device Library (DDL) is a collection of HSPICE device models of discrete components, which you can use with HSPICE. 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 pathname 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 license files.

DDL Library Access

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

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

```
X1 2 1 D1N4004
```

Where `D1N4004` is the model name.

See Element and Source Statements on page 57 and the *HSPICE Elements and Device Models Manual* for descriptions of element statements.

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

HSPICE or HSPICE RF accesses DDL models in several ways:

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

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

  Use this method to list the personal libraries to search. HSPICE first searches the default libraries referenced in the hspice.ini file, then searches libraries in the order listed in the input file.
- Directly include a specific model, using the `.INCLUDE` statement. For example, to use a model named T2N2211, store the model in a file named T2N2211.inc, and put the following statement in the input file:

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

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

Vendor Libraries

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

- ASIC vendors provide comprehensive cells, corresponding to inverters, gates, latches, and output buffers.
- Memory and microprocessor vendors supply input and output buffers.

- Interface vendors supply complete cells for simple functions and output buffers, to use in generic family output.
- Analog vendors supply behavioral models.

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

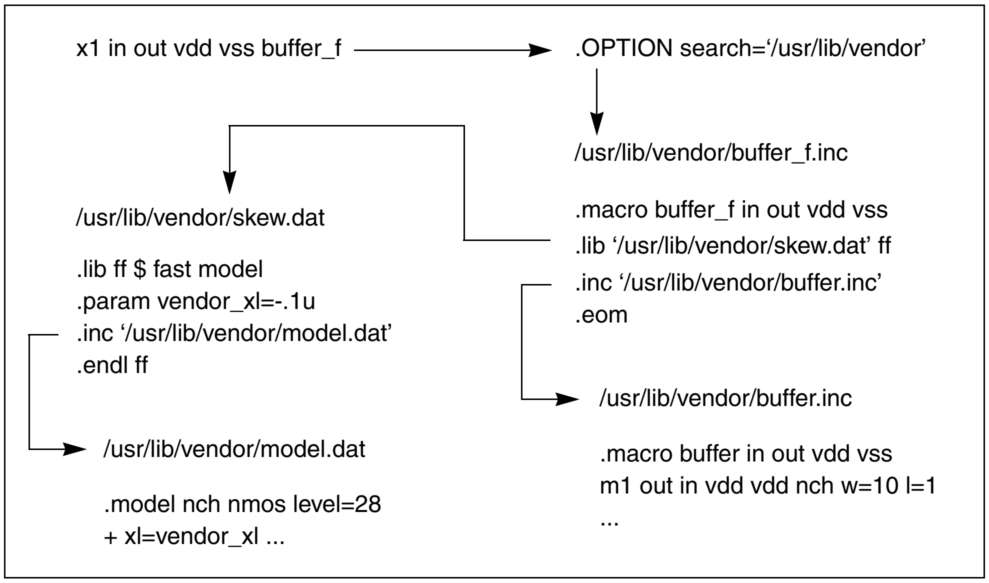

x1 in out vdd vss buffer_f
.OPTION search='/usr/lib/vendor'
/usr/lib/vendor/buffer_f.inc
.macro buffer_f in out vdd vss
.lib '/usr/lib/vendor/skew.dat' ff
.inc '/usr/lib/vendor/buffer.inc'
.eom
/usr/lib/vendor/skew.dat
.lib ff $ fast model
.param vendor_xl=-.1u
.inc '/usr/lib/vendor/model.dat'
.endl 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 ...

Figure 10 Vendor Library Usage

Subcircuit Library Structure

The `.OPTION SEARCH` command is an implicit include command. The include file can contain `.lib` calls in addition to the subcircuit definition. (See Figure 10 for a graphic view.) Your library structure must adhere to the `.INCLUDE` statement specification in the implicit subcircuit. You can use this statement to specify the directory that contains the *<subname>.inc* subcircuit file, and then reference the *<subname>* in each subcircuit call.

The component naming conventions for each subcircuit is:

`<`*`subname`*`>.inc`

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

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

Post-Layout Back-Annotation

The HSPICE parser enables the parsing and annotating of parasitic elements. HSPICE supports back-annotation of two types of parasitic netlist formats:

- Standard Parasitic Format (SPF)
- Standard Parasitic Exchange Format (SPEF)

The parasitic netlist describes the interconnect delay and load, due to parasitic resistance and capacitance. Parasitics can be represented on a net-by-net basis from a simple lumped capacitance to a fully distributed resistance capacitance tree.

Invoking Post-Layout Back-Annotation

The enhanced parser has different format for warning/error message. It is similar to RF or Fast-Spice and stricter on the syntax than the original HSPICE parser but produces the same result once the net-list is accepted. To annotate circuits, HSPICE should be launched in the following way:

```
hspice -i inputfile -o output -x
```

The following is a list of back-annotation options that you can use in HSPICE:

- .OPTION BA_FILE = “filename”: Full parasitic expansion. Filename is the name of the file that contains parasitic information in SPEF or DSPF format.
- .OPTION BA_ACTIVE = “active_net_filename”: Conducts selective parasitic expansion. active_net_filename is the name of the file that contains information about active nets. You must use this option with BA_FILE, or it has no effect.
- .OPTION BA_ERROR: the mode of error handling on nets.
  - 1. EXIT: terminate simulation with error
  - 2. NO: don't expand anything on the net
  - 3. LUMPCAP (default): only add the total lumped net capacitance

- 4. YES: expand whatever can be expanded

- .OPTION BA_TERMINAL: the terminal mapping with the format "terminal_name_in_file recognized_name", e.g., "ba_terminal = Cmy_gate SRC".

Table 9 Default rules for element terminal names

Terminal Index	M, J Elements	Q Elements	R,C,D Elements
1	D*	C*	A*, P*
2	G*	B*	M*, B* C*, N*
3	S*	E*	S*
4	B*	S*	n/a

DSPF and SPEF File Structures

DSPF File Structure

The DSPF standard is published by Open Verilog International (OVI).

```
DSPF_file ::=
*|DSPF{version}
{*|DESIGN design_name}
{*|DATE date}
{*|VENDOR vendor}
{*|PROGRAM program_name}
{*|VERSION program_version}
{*|DIVIDER divider}
{*|DELIMITER delimiter}
.SUBCKT
*|GROUND_NET
{path divider} net_name
*|NET {path divider} net_name ||
{path divider} instance_name ||
pin_name
net_capacitance
*|P (pin_name pin_type
pinCap
{resistance {unit} {O}
capacitance {unit} {F}}
{x_coordinate y_coordinate})
||
*|I {path divider} instance_name
delimiter pin_name
{path divider} instance_name
pin_name pin_type
pinCap
{resistance {unit} {O}
capacitance {unit}{F}}
{x_coordinate y_coordinate}
*|S ({path divider} net_name ||
{path divider} instance_name
delimiter pin_name ||
pin_name
instance_number
{x_coordinate y_coordinate})
capacitor_statements
resistor_statements
subcircuit_call_statements
.ENDS
{.END}
```

SPEF File Structure

The IEEE-1481 specification requires the following file structure in a SPEF file. For the current release, only the only typical set (triple value SPEF file) is annotated. Parameters in [brackets] are optional:

```
SPEF_file ::=
*SPEF version
*DESIGN design_name
*DATE date
*VENDOR vendor
*PROGRAM program_name
*VERSION program_version
*DESIGN_FLOW flow_type {flow_type}
*DIVIDER divider
*DELIMITER delimiter
*BUS_DELIMITER bus_prefix bus_suffix
*T_UNIT time_unit NS|PS
*C_UNIT capacitance_unit FF|PF
*R_UNIT resistance_unit OHM|KOHM
*L_UNIT inductance_unit HENRY|MH|UH
[*NAME_MAP name_index
name_id|bit|path|name|physical_ref]
[*POWER_NETS logical_power_net physical_power_net ...]
[*GROUND_NETS ground_net ...]
[*PORTS logical_port I|B|O
*C coordinate ...
*L par_value
*S rising_slew falling_slew [low_threshold high_threshold]
*D cell_type]
[*PHYSICAL_PORTS [physical_instance delimiter]
physical_port I|B|O
*C coordinate ...
*L par_value
*S rising_slew falling_slew [low_threshold high_threshold]
*D cell_type]
[*DEFINE logical_instance design_name |
*PDEFINE physical_instance design_name]
*D_NET net_path total_capacitance
[*V routing_confidence]
[*CONN
*P [logical_instance delimiter]
logical_port|physical_port
I|B|O
*C coordinate ...
*L par_value
*S rising_slew falling_slew
[low_threshold high_threshold]
*D cell_type
```

```
|
*I [physical_instance delimiter]
logical_pin|physical_node
I|B|O
*C coordinate ...
*L par_value
*S rising_slew falling_slew
[low_threshold high_threshold]
*D cell_type
*N net_name delimiter net_number coordinate
[*CAP cap_id node1 [node2] capacitance]
[*RES res_id node1 node2 resistance]
[*INDUC induc_id node1 node2 inductance]
*END
```

5 Using Interactive Mode

This chapter describes how to use the interactive mode in HSPICE.

Note:

Interactive mode is not supported in HSPICE RF.

Invoking Interactive Mode

You start interactive mode using the `hspice` executable in the $*installdir*/<*arch*> directory (for example, $*installdir*/sparcOS5).

To start the interactive mode, from the command prompt, type:

```
% hspice -I
HSPICE >
```

The interactive environment functions from a special HSPICE-shell. You can use many commands while in this environment to simplify your design work.

Quitting Interactive Mode

You quit an interactive mode session by entering:

```
HSPICE > quit
```

Executing an Interactive Script

You can also use interactive commands in a command script file. To execute an interactive script, from the command prompt, type:

```
% hspice -I -L command_script
```

Where <*command_script*> is the name of the file containing the interactive commands.

Examples

The examples in this section examples show you how to use the interactive environment commands.

Getting Help

You use the `help` command to show the interactive mode syntax; for example,

```
% hspice -I
HSPICE > help

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

Creating a Netlist

You use the `input` command to create a netlist by using the vi text editor; for example,

```
% hspice -I
HSPICE > input
R1 1 0 2
V1 1 0 3
.print I(R1)
.end
```

Save the file and exit vi.

Specifying an Analysis

You use the `ac`, `dc`, or `tran` command to specify an analysis; for example,

```
HSPICE > dc v1 -5v 5v 0.5v
```

Running an Analysis

You use the `run` command to simulate a netlist; for example,

```
HSPICE > run
> info:  ***** hspice job concluded
```

HSPICE outputs the simulation results. This output is equivalent to a *.lis* file.

Viewing a Netlist

You use the `list` command to view a netlist; for example,

```
HSPICE > list
1  * this is an interactive mode example
2
3  R1 1 0 2
4  V1 1 0 3
5  .print I(R1)
6  .end
```

Loading and Running an Existing Netlist

Use the `load` command to read an existing netlist; for example, to load a netlist named “tt1.sp“:

```
% hspice -I
HSPICE > load tt1
```

Use the `list` command to view a netlist; for example, to view the tt1.sp netlist:

```
HSPICE > list
1  * this is an interactive mode example
2
3  R1 1 0 2
4  V1 1 0 3
5  .print I(R1)
6  .end
```

Use the `dc` command to specify an analysis and then run HSPICE:

```
HSPICE > dc v1 -5v 5v 0.5v
HSPICE > run
> info:  ***** hspice job concluded
```

Use the `list` command to view a netlist again. Notice that the DC analysis is in the interactive mode netlist. The original netlist, tt1.sp, is unchanged.

```
HSPICE > list
1  * this is an interactive mode example
2
3  R1 1 0 2
4  V1 1 0 3
5  .print I(R1)
6  .dc v1 -5v 5v 0.5v
7  .end
```

Use the `run` command to simulate the netlist:

```
HSPICE > run
> info:  ***** hspice job concluded
```

Use the `cd` command to change the current working directory to /home/usr:

```
HSPICE > cd /home/usr
```

Use the `pwd` command to print the full pathname of the current directory:

```
HSPICE > pwd
> /home/usr
```

Use the `quit` command to terminate the interactive mode:

```
HSPICE > quit
%
```

Using Environment Commands

Use the `load` command to read netlist “tt3.sp” and the list command to view it:

```
% hspice -I
HSPICE > load tt3.sp
HSPICE > list
R1 1 0 2
V1 1 0 3
.print I(R1)
.param a=10, t=24
.dc v(1) -5v 5v 0.5v
.end
```

Use the `ls` command to list the files in the current working directory:

```
HSPICE > ls
tt.sp
tt.sw0
:
```

Use the `set outflag` command to prevent printing simulation results to stdout (the default is `true`):

```
HSPICE > set outflag false
HSPICE > run
```

Use the `info outflag` command to view the current setting of outflag:

```
HSPICE > info outflag
false
HSPICE > quit
%
```

Recording and Saving Interactive Commands to a File

Use the `load` command to read netlist “tt6.sp” and print the full pathname of the current directory:

```
% hspice -I
HSPICE > load tt6.sp
HSPICE > pwd
```

Write all interactive commands entered to file int1.cmd:

```
HSPICE > save command int1
HSPICE > ls
int1.cmd
tt6.sp
tt6.sw0
HSPICE > input
R1 1 0 2
V1 1 0 3
.print I(R1)
.end
HSPICE > save netlist ex.sp
HSPICE > ls
int1.cmd
tt6.sp
tt6.sw0
ex.sp
HSPICE > quit
%
```

Quit interactive mode and return to the system prompt:

Printing a Voltage Value During Simulation

Here's an example where a voltage value is printed during simulation:

```
% hspice -I
HSPICE > load tt2.sp
HSPICE > list
1  * this is an interactive mode example
2
3  R1 1 0 2
4  V1 1 0 3
5  .print I(R1)
6  .dc  v1  -5v  5v  0.5v
HSPICE > print dc v(1,0)
HSPICE > run
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
v(1, 0)  0.000000
>info:        ***** hspice job concluded
HSPICE > quit
%
```

Using a Command File to Run HSPICE in Interactive Mode

You use the `-I -L` arguments to invoke interactive mode on a command file and run it; for example,

% **hspice -I -L** int1.cmd

Running Multiple Testcases

If you want to run multiple testcases and save the license check in and check out times, you can use the command file similar the following example.

Using any text editor, create a command file named `bat.cmd` containing these entries:

```
load tt.sp
run
load qq.sp
run
quit
```

Run HSPICE in interactive mode to execute bat.cmd:

```
% hspice -I -L bat.cmd
...
% ls
tt.sp
tt.sw0
qq.sp
qq.sw0
```

You will find that a license is checked out only one time and HSPICE simulates both tt.sp and qq.sp netlists.

6

HSPICE GUI for Windows

Describes how to use the HSPICE GUI for Windows.

To open and install the GUI, click on the HSPUI icon. Figure 11 shows the directory structure for the HSPICE GUI for Windows.

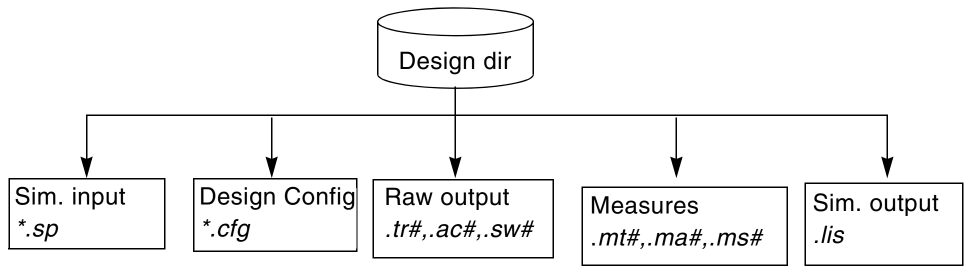


Figure 11 Directory Structure

This appendix is a organized as follows:

- Working with Designs
- Configuring the HSPICE GUI for Windows
- Running Multiple Simulations
- Metaencrypt and Converter Utilities, Client/Server Operation
- Troubleshooting Guide

Working with Designs

A new design can be created in several ways. The Launcher allows you to browse for an input file for HSPICE, which has the default file suffix .sp. Click the Launcher Open button to display a standard file browser.

Selecting a file of the type <design>.sp causes the Launcher to display the main form, which contains the following items:

- input filename
- design title (the first line of the file *design*.sp)
- output filename
- HSPICE and AvanWaves version

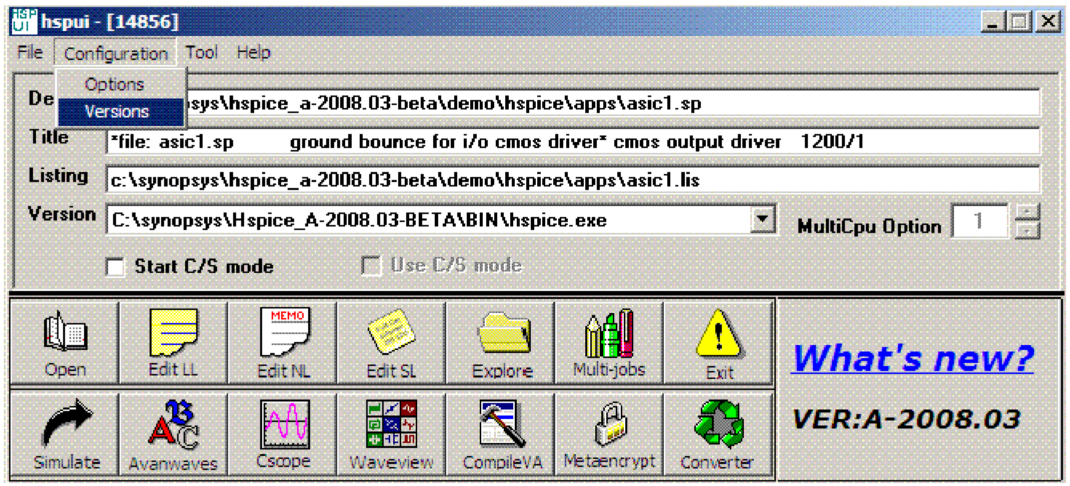
Table 10 Design Commands in the Launcher

Command	**Description**
File > Save Configuration	Saves the current design launcher configuration
<LastDesigns>	Lists the last five designs opened
File > Exit	Exits the Launcher

The Launcher checks on the status of a given design when it is opened. If the input file exists, the Simulate button is active. If the listing file exists for the design, the Edit Listing button is active. The Edit Netlist and AvanWaves buttons are always active.

You do not need to save a design to Simulate or view the results of a simulation with AvanWaves.

Figure 12 on page 97 shows the main window of the Launcher.



sys\hspice_a-2008.03-beta\demo\hspice\apps\asic1.sp

file: asic1.sp ground bounce for i/o cmos driver cmos output driver 1200/1

c:\synopsys\hspice_a-2008.03-beta\demo\hspice\apps\asic1.lis

C:\synopsys\Hspice_A-2008.03-BETA\BIN\hspice.exe

1

Start C/S mode

Use C/S mode

Figure 12 Launcher Main Window

Configuring the HSPICE GUI for Windows

Customize configurations using the Configuration menu of the Launcher as shown in Figure 13.

The start-up directory defaults to the value of the *installdir* environment variable set up during HSPICE installation.

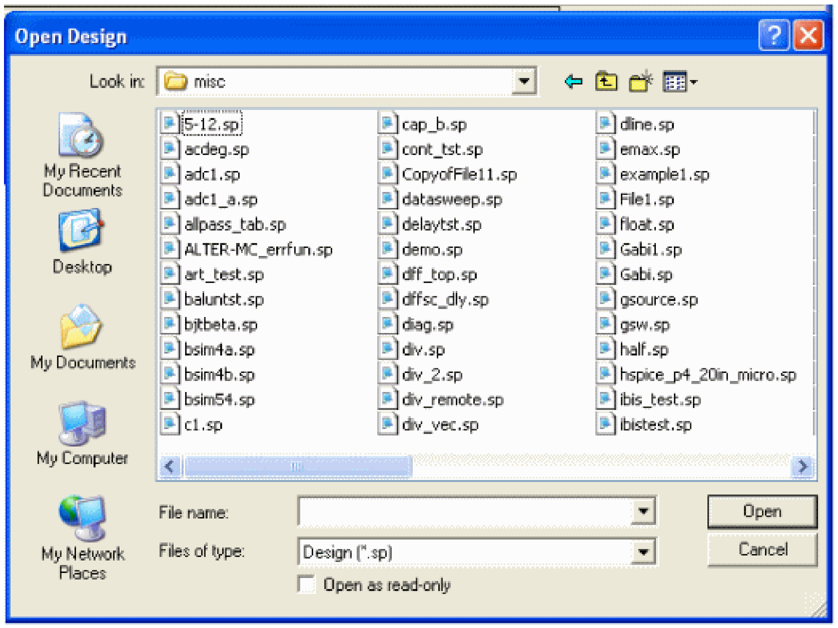
- The input file suffix defaults to .sp.
- The output file suffix defaults to .lis.
- The editor defaults to notepad.exe.

If you change a value, the Launcher updates the *installdir*/hspui.cfg file. The next Launcher run provides the new values.

The Configuration > Versions item lists current executables and their paths for the Launcher (HSPUI), HSPICE, and WaveView.

Note:

The Configuration > Version strings change from the main window Versions combo box. You cannot change them here.



5-12.sp, acdeg.sp, adc1.sp, adc1_a.sp, allpass_tab.sp, ALTER-MC_errfun.sp, art_test.sp, baluntst.sp, bjtbeta.sp, bsim4a.sp, bsim4b.sp, bsim54.sp, c1.sp, cap_b.sp, cont_tst.sp, CopyofFile11.sp, datasweep.sp, delaytst.sp, demo.sp, dff_top.sp, dffsc_dly.sp, diag.sp, div.sp, div_2.sp, div_remote.sp, div_vec.sp, dline.sp, emax.sp, example1.sp, File1.sp, float.sp, Gabi1.sp, Gabi.sp, gsource.sp, gsw.sp, half.sp, hspice_p4_20in_micro.sp, ibis_test.sp, ibistest.sp

File name:
Files of type: Design (*.sp)

Figure 13   Launcher Options > Open Design

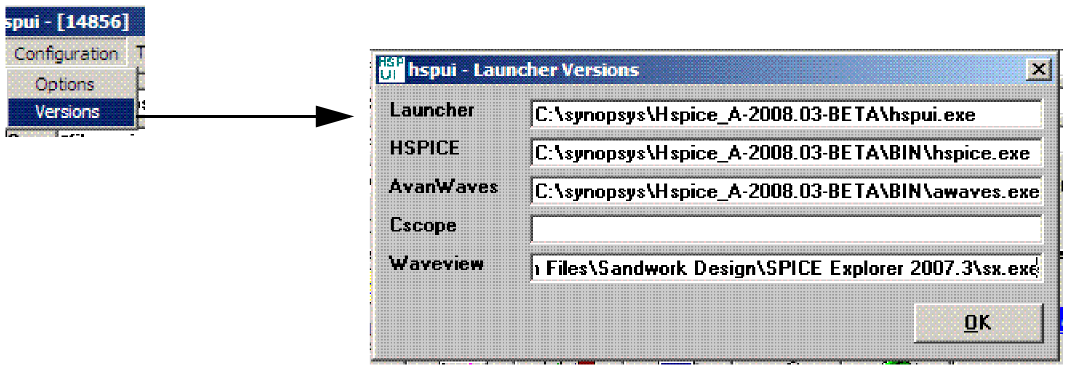
To associate your *design*.sp file with the Launcher, use the File > Associate command in the Windows File Manager. You can double-click an *.sp* file in the File Manager window to automatically invoke the HSPICE/Win Launcher. Refer to your Windows documentation for details on how to do this.

For further configuration tips see Setting the hspui.cfg File Values.

Launching Waveview in HSPUI

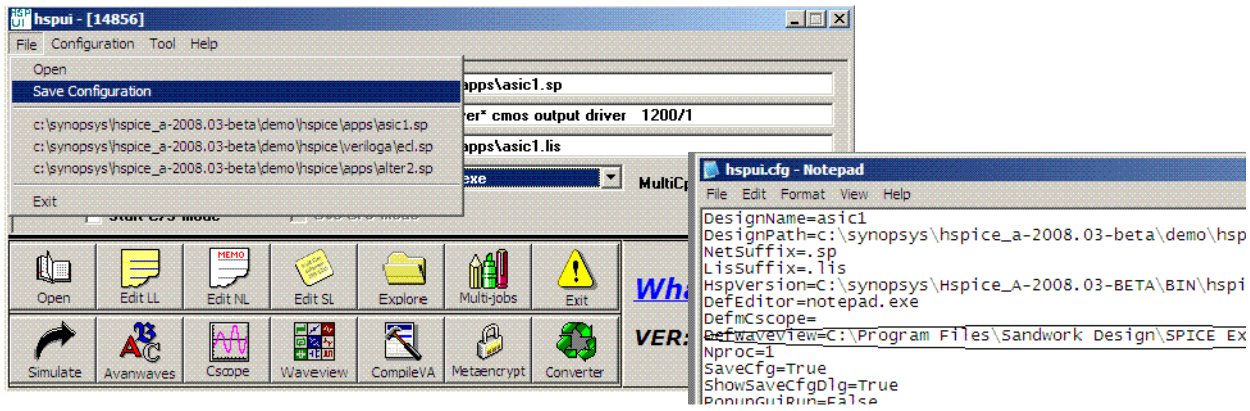
You can activate the WaveView Analyzer add-on through the UI.

1. Open the Versions form in the HSPUI.



Launcher	C:\synopsys\Hspice_A-2008.03-BETA\hspui.exe
HSPICE	C:\synopsys\Hspice_A-2008.03-BETA\BIN\hspice.exe
AvanWaves	C:\synopsys\Hspice_A-2008.03-BETA\BIN\awaves.exe
Cscope	
Waveview	ı Files\Sandwork Design\SPICE Explorer 2007.3\sx.exe

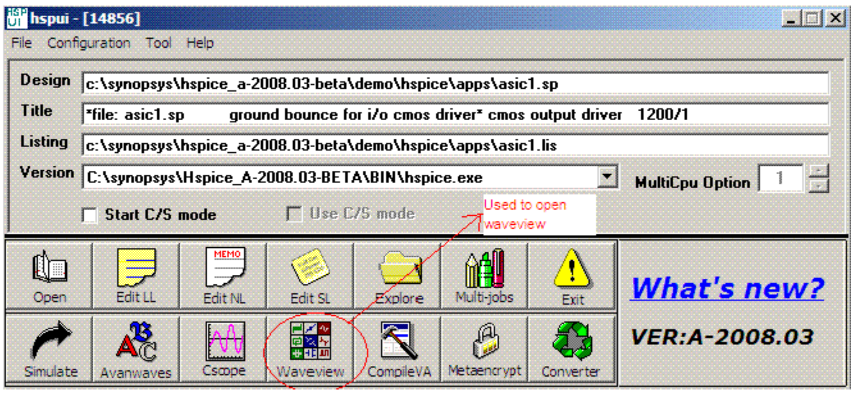
2. Configure the sx executable in the Waveview field.



```
DesignName=asic1
DesignPath=C:\synopsys\hspice_a-2008.03-beta\demo\hsp
NetSuffix=.sp
LisSuffix=.lis
Hspversion=C:\synopsys\Hspice_A-2008.03-BETA\BIN\hspi
DefEditor=notepad.exe
DefmCscope=
DefWaveView=C:\Program Files\Sandwork Design\SPICE Ex
Nproc=1
SaveCfg=True
ShowSaveCfgDlg=True
PopupGuiRun=False
```

3. Save configuration to hspui.cfg

4. Click the **Waveview** button in HSPUI main form to launch WaveView Analyzer.



Design	c:\synopsys\hspice_a-2008.03-beta\demo\hspice\apps\asic1.sp
Title	*file: asic1.sp ground bounce for i/o cmos driver* cmos output driver 1200/1
Listing	c:\synopsys\hspice_a-2008.03-beta\demo\hspice\apps\asic1.lis
Version	C:\synopsys\Hspice_A-2008.03-BETA\BIN\hspice.exe
MultiCpu Option	1
Start C/S mode	☐
Use C/S mode	☐

Setting up Windows for Verilog-A

The following setup is required to properly compile and simulate Verilog-A models.

1. Select My Computer > Properties > Advanced > Environment Variables
2. Click **System Variables**.
3. Define a new variable name as “HSP_HOME” and set the value to the actual installation path. For example, C:\synopsys\Hspice_B-2008.09
4. Edit the “PATH” variable to include:

```
%PATH%;%HSP_HOME%\bin;%HSP_HOME%\win2k\veriloga_utils;%HSP_H
OME%\win2k\
        veriloga_utils\include
```

5. Click **OK** to save.

Running Multiple Simulations

Use the HSPICE/Launcher file browser to build a list of simulations from different directories for consecutive HSPICE processing.

Click Multi-jobs in the main window to open the HSPICE Multi-jobs window (Figure 14). Simulation files are chosen from the Drive/Directory list box and placed in the Files list box.

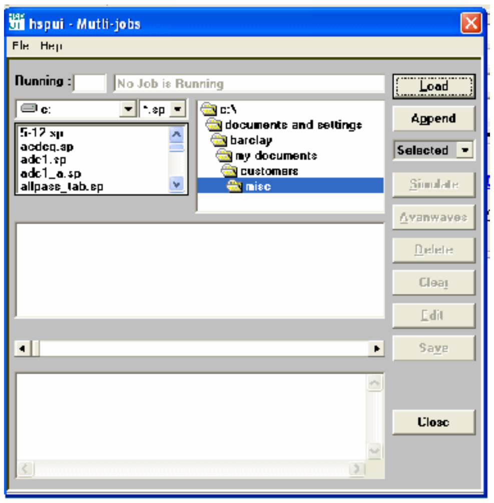


Figure 14   HSPUI Multi-jobs Window

Building the Batch Job List

To build a batch job list:

1. Click Multi-jobs in the main window.
2. Using the Drive/Directory boxes, locate the directory of files that you wish to simulate.
3. To load all files in the directory, click the **Load** button on the right side of the Multi-jobs window.

   Note that any file names already in the list will be replaced.
4. To add additional files from other directories, repeat Step 2 and use the Append button.

Simulating a Batch Job

To simulate a batch job use either of the following two methods:

- Open the Multi-Jobs window, click **Load** to load netlists and then click **Simulate**.

  or

- Open the Multi-Jobs window, click **Load** to load netlists, then click the **Save** button to create a batch simulation file (**.bat* file), then run the **.bat* file on Explorer or by using the MS-DOS `cmd` window.

To simulate a batch job list:

1. To simulate all of the files in the Batch Job list, set the pull-down menu to **All** and click the **Simulate** button.
2. To run simulation on a single file or a group of files, set the pulldown menu to **Selected** and select those files you wish to simulate from the Batch Job list box.

   Use the left mouse button to select a single file.

   - Press and hold the Ctrl key and select another file with the left mouse button to add to the selected list.
   - Press and hold the Shift key to select all files between the current file and the last selected file.
3. Click the Simulate button to start the consecutive simulations.

Sample Batch Work-Flow

Once in Multi-jobs, an example work flow might be:

1. Navigate to your source files: You can select the source drive letter from the pull-down menu in the upper left. Next, if needed, change the file extension filter from .sp to *.* to pick up .hsp or .spi files. Then, you can navigate to a new folder using the file system mini-browser.

   Note:

   If you set the filter to *.*, file types other than HSPICE input decks are likely to be included.
2. Add files to the run list using the **Load** and **Append** buttons: When you navigate to the folder containing the SPICE decks, they are displayed in the left-hand list window. Although you can select files in this window, clicking

the **Load** button adds all the files in the left-hand list window, as well as those in subdirectories. These files go into a run list and are numbered sequentially. **Load** clears the contents of the run list, but you can navigate to another folder containing HSPICE sources files and click the **Append** button to add additional files to the run list.

3. Edit the run list: Select files one at a time (CTRL-click) or a range of files (SHIFT-click) and click the **Delete** button to remove them from the run list.

   Note:

   > **Delete** does not remove the file from your hard-drive, just the run list. **Clear** removes the entire contents of the run list and allows you to start over again.

4. Perform text edits on individual HSPICE jobs: Selecting a job and clicking **Edit** displays that HSPICE deck in the text editor you designated in the HSPICE UI Configuration > Options dialog. Save and close it to continue.
5. Simulate the jobs: Only selected jobs in the run list can be simulated. Select (highlight) files by clicking on them, or by choosing **All Files** from the pull-down menu below the **Append** button. Click **Simulate** to run the jobs in order, one at a time.
6. Save the run list for later use: Using the File > Save or File > Save As pull-down menu items, you can save the contents of the run list to a file and open it with File > Open to begin work at a later time.
7. Invoke a waveform viewer graphical waveform analysis.
8. Create a batch (**.bat*) file with a list of jobs to be run without using the UI: Click the **Save** button (not File > Save) to create a DOS batch file containing the full path to the HSPICE executable and the design name for each job in the run list. This *.bat* file can be executed from a DOS command prompt or by double-clicking on it in Windows.

Running Multi-Threading

To run multi-threading:

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

Metaencrypt and Converter Utilities, Client/Server Operation

For information about use of the `metaencrypt` utility for encryption of files, refer to Chapter 7, Library and Data Encryption.

For information about use of the Converter utility for the conversion of output by HSPICE, refer to Using the HSPICE Output Converter Utility.

The C/S mode check boxes allow you to start and use the HSPICE traditional client/server mode. For information about Client/Server usage, refer to Using HSPICE in Client/Server Mode.

CMI Directory Structure

For information on the Custom Common Model Interface (CMI) directory structure for Windows platforms, contact your Synopsys technical support team to access the application note for the HSPICE CMI.

Troubleshooting Guide

Setting the hspui.cfg File Values

If, for example, your netlists do not have an **.sp* suffix, you can edit your *hspui.cfg* file to accept the file extension that you use. Locate the hspui.cfg file in the HSPICE installation directory. Use this file to configure the HSPUI buttons and settings, including the netlist extension.

If you do not find this file, create one as follows.

- Open the HSPUI.
- Save the configuration, **File > Save Configuration**.

In the *hspui.cfg* file you will find a listing similar to the following:

```
DesignName=
DesignPath=
NetSuffix=.sp
LisSuffix=.lis
HspVersion=C:\synopsys\Hspice_Z-2007.03\BIN\hspice.exe
DefEditor=C:\Program Files\Windows NT\Accessories\wordpad.exe
DefmCscope=
Nproc=1
LastFile(0)=
LastFile(1)=
LastFile(2)=
LastFile(3)=
LastFile(4)=
```

The lines for `DesignName`, `DesignPath`, `HspVersion`, and `LastFile(#)` are informational and should *not* be edited.

- To change the netlist extension, edit the `NetSuffix` line. Multiple file extension support is not available.
- To change the output listing file to have a different extension, edit the `LisSuffix` line.
- To use an editor other than Notepad, enter the path on the `DefEditor` line.
- If you want to use CosmosScope as the waveform viewer, add the path to the CosmosScope (cscope.exe) executable on the `DefmCscope` line.
- You may also change the default number of processors used when running HSPICE; edit the `Nproc` line.

Text Editor Issues

If you click the Edit LL button in the HSPUI, and the listing file (**.lis*) comes up with extra carriage returns and is hard to read, solve this issue using one of the following solutions:

- With Notepad open, click Format on the tool bar and uncheck Word Wrap.
- Configure the HSPUI to use another text editor to view the files:
- From the HSPUI click Configuration > Options.
- For the Default Editor field, click the Browse button.
- Navigate to the *.exe* for desired text editor, i.e., C:\Program Files\Windows NT\Accessories\wordpad.exe

Simulating a UNIX Netlist File

If the netlist file is a UNIX text file (no ^M at the end of each line), then HSPUI will not read it correctly and the simulate and edit netlist buttons will be grayed out. However, if the file is saved as a DOS text file (^M at the end of each line), then HSPUI will read the file correctly and the simulate and edit netlist buttons are enabled. (HSPICE will simulate a netlist file in either text format.)

7

Library and Data Encryption

Describes the Synopsys library encryption methods and how they are used to protect your intellectual property.

Organization

These sections present the HSPICE encryption methods according to the following topics:

Library Encryption

Using HSPICE, you can encrypt your own proprietary HSPICE custom models, parameters, and circuits and distribute to others without revealing your company’s sensitive information. Recipients of an encrypted library can run simulations using HSPICE and your libraries, so that encrypted parameters, encrypted circuit netlists, and internal node voltages do not appear in output files. Your library user sees the devices and circuits as black boxes that provide terminal functions only.

Encrypting a Model Library Using the metaencrypt Utility

A typical model library from a foundry has the following structure:

```
* model library mylib.lib
.lib tt
.param toxn=
...
.inc mymodel.mdl
.endl
```

If you encrypt both mylib.lib and mymodel.mdl, then you will get the error: `Command exited with non-zero status 1` during the HSPICE simulation. This is because HSPICE does not support the nesting of encrypted files.

To correctly encrypt the model file, you need to change the library structure. The model parameters and the models need to be encrypted separately as shown in the following steps:

1. The modified structure should be as follows:

   ```
   .* model library mylib.lib
   .lib tt
   .inc myparam.par   $ put parameter definitions into myparam.par
   .inc mymodel.mdl   $ original model file
   .endl
   ```

2. Encrypt the parameter file, model file and netlist as follows:

   ```
   metaencrypt -i myparam.par -o myparam.par.enc -t randkey
   metaencrypt -i mymodel.mdl -o mymodel.mdl.enc -t randkey
   metaencrypt -i mynetlist.sp -o mynetlist.sp.enc -t randkey
   ```

3. To simulate the circuit, include the encrypted files and call the library file, mylib.lib in the top level netlist:

```
* top level netlist
.inc mynetlist.sp.enc
.lib mylib.lib tt
...
.end

* modified library mylib.lib
.lib tt
.inc myparam.par.enc
.inc mymodel.mdl.enc
.end
```

Three Encryption Methods

HSPICE and HSPICE RF support three types of encryption through the `metaencrypt` utility:

- Traditional Library Encryption (Freelib)
- 8-Byte Key Encryption
- Triple DES Public and Random Keys

The `metaencrypt` utility can encrypt files with lines up to 254 characters or shorter. You can include multiple types of encrypted files in a HSPICE simulation.

Installing and Running metaencrypt

This section describes how to install and run `metaencrypt`.

Installing metaencrypt

The metaencrypt utility is part of the general HSPICE distribution and is located in the <$installdir>/bin directory.

If you have not installed HSPICE on your system, first install HSPICE according to the *Installation Guide* and the *HSPICE Release Notes*. Verify that the license file contains license tokens `encrypt` and `metaencrypt3des` for 3DES.

Running metaencrypt

Syntax

```
metaencrypt -i input_file -o encrypted_output_file \
   -t encrypt_type -d encrypt_dir
   -r synopsys_tool[:access_control] [0|1]
   -r synopsys_tool[:access_control] [0|1]...
```

Argument	Description
-i *infileName*	Unencrypted input filename.
-o *outfileName*	Encrypted output filename.
-t *encrypt_type*	Encryption method: ▪ Freelib—weak (low security) -t [ddl1 | ddl2 | custom | freelib] ▪ 8-byte—strong (medium security) -t <8-byte-string> ▪ Triple DES—strongest (high security) -t [privkey <key or file> | randkey]
-d *encrypt_dir*	Valid only when using tripleDES to encrypt file. Data between .PROT and .UNPROT commands are encrypted and written to file infile.enc# in directory encrypt_dir.

Argument	Description

-r *synopsys...tool...*	synopsys_tool(s) can be: HSPICE, NanoSim, HSIM or Synopsys. access_control can be 0 or 1. The default is 0.
	0: tools suppress any warning information related to encrypted block. The default value is ON for all, i.e., if no –r option, or no value specified after –r, it allows all possible for all Synopsys tools (NanoSim, HSIM, HSPICE) to read.
	1: Tools do not suppress warning messages from encrypted block.
	Notes:
	HSPICE can always read an encrypted netlist file, even if no -r hspice has been specified.
	If there are multiple settings to same tool, the last setting will be enabled.
	
	Example 1: In this invocation, the access_control to NanoSim is 0 while others is 1.
	`.metaencrypt -i test.sp -o test.spe -t randkey`
	`+ -r synopsys:1 -r nanosim:0`
	Example 2: Shows how you can limit the parsing of the encrypted files to HSPICE only, not other tools.
	`metaencrypt -i test.sp -o test.spe -t randkey`
	`+ -r hspice`

Encryption Guidelines

Before encrypting, you must test out any circuits and device parameters, as you will not be able to see what is wrong after encryption because HSPICE does not let you read the encrypted data.

You can use any legal HSPICE command inside subcircuits that you encrypt. Refer to Using Subcircuits in Chapter 4, Input Netlist and Data Entry for more information about how to construct subcircuits. The structure of your libraries can affect how you encrypt them. If your library requires that you change the name of a subcircuit, you must encrypt that circuit again.

To encrypt more than one file in a directory, use the following shell script, which encrypts the files as a group. In this example, the script uses the traditional encryption method. The script produces a *.inc* encrypted file, for each .dat file in the current directory. The `metaencrypt` command assumes that unencrypted files have a .dat suffix.

```
#!/bin/sh
for i in *.dat
do
Base=‘basename $i .dat‘
metaencrypt -i $Base.dat -o $Base.inc -t Freelib
done
```

An encrypted file is used in much the same way as before encryption. The name of the file may be different, however, and so the `.include` and `.lib` commands may need to be updated.

Note:

HSPICE encryption does not apply to Verilog-A module files. Instead, you can use the standalone HSPICE Verilog-A compiler, hsp-vacomp, to generate a compiled library file (**.cml*) from your Verilog-A module file, then ship the **.cml* file instead of your text format Verilog-A file. Note that the **.cml* file is both platform- and HSPICE-version specific. For more information, refer to Chapter 31, Using Verilog-A.

You can probe any specified encrypted nodes using `.OPTION PROBE`.

General Example

The requirements for encrypted libraries of subcircuits are the same as the requirements for regular subcircuit libraries, as described in the HSPICE Simulation and Analysis User Guide. To refer to an encrypted subcircuit, use its subcircuit name in a subcircuit element line of the HSPICE netlist.

The following example describes an encrypted I/O buffer library subcircuit. This subcircuit consists of several subcircuits and model commands that you need to protect with encryption. Figure 15 on page 113 shows the organization of subcircuits and models, in the libraries used in this example.

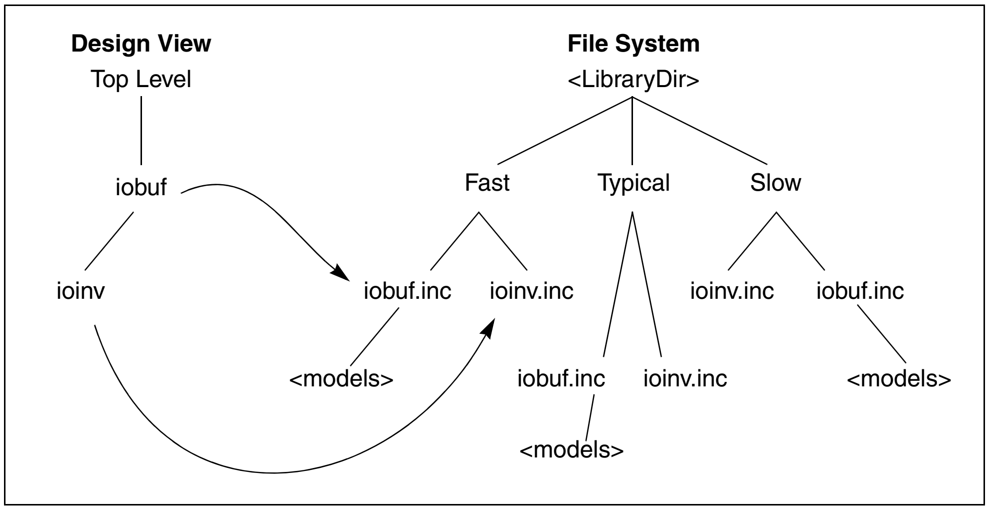


Figure 15 Encrypted Library Structure

The following input file fragment from the main circuit level selects the Fast library. It also creates two instances of the iobuf circuit.

```
...
.Option Search='<LibraryDir>/Fast'   $ Corner Spec
x1 drvin drvout iobuf   Cload=2pF   $ Driver
u1 drvout 0 recvin 0 PCBModel  ...   $ Trace
x2 recvin recvout iobuf   $ Receiver
...
```

The <LibraryDir>/Fast/iobuf.inc file contains:

```
.Subckt iobuf Pin1 Pin2 Cload=1pF
*
* iobuf.inc - model 2001 improved iobuf
*
cPin1 Pin1 0 1pF   $ Users cannot change this!
x1 Pin1 Pin2 ioinv
.Model pMod pmos Level=28 Vto=...      $ <FastModels>
.Model nMod nmos Level=28 Vto=...      $ <FastModels>
cPin2 Pin2 0 Cload   $ gives you some control
.Ends
```

The <LibraryDir>/Fast/ioinv.inc file contains:

```
.Subckt ioinv Pin1 Pin2
mp1 Vcc    Pin1 Pin2 Vcc pMod...
mn1 Pin2   Pin1 Gnd Gnd
nMod...
.Ends
```

The encrypted file looks similar to the following:

```
.SUBCKT ioinv Pin1 Pin2
.PROT FREELIB   $ Encryption starts here ...
X34%43*27@#^3rx*34&%^#1
^(*^!^HJHD(*@H$!:&*$
dFE2341&*&)(@@3   $ ... and stops here

.ENDS
```

Note:

The `.UNPROTECT` statement is encrypted during the encryption process.

After encryption, the basic layout of the subcircuits is the same. However, you cannot read the file. Only HSPICE can read this file.

Encryption also suppresses printouts of encrypted model information from HSPICE. Only HSPICE can decrypt the model.

Traditional Library Encryption

The traditional library encryption algorithm (Freelib) is based on a 5-rotor Enigma machine. You can specify which portions of subcircuits to encrypt. Encrypted portions are designated only within the `.PROT/.UNPROT` commands. Anything outside is left unencrypted. Library encryption uses a key value, which HSPICE reconstructs for decryption.

Note:

If you divide a data line into more than one line, and use the line continuation character (+) to link the lines, you cannot add `.PROT` or `.UNPROT` commands among these lines. The following example fails:

```
.prot
.Model N1 NMOS Level= 57
+TNOM = 27 TOX = 4.5E-09 TSI = .0000001 TBOX = 8E-08
+MOBMOD = 0 CAPMOD = 2 SHMOD =0
.unprot
+PARAMCHK=0 WINT = 0 LINT = -2E-08
```

When `metaencrypt` reads the input file, it looks for `.PROT` and `.UNPROT` pairs, and encrypts the text between them. You can encrypt only one file at a time.

Example

```
metaencrypt -i newmos.lib -o newmos.inc -t freelib
```

Note:

The netlist needs to be “complete”, i.e,- have an `.end` statement.

Creating Files Using Traditional Encryption

The following sections describe:

- Non-Library Encrypted Portions
- *.lib File Encryption

Non-Library Encrypted Portions

You can encrypt the data between `.PROT` and `.UNPROT` commands, in a .sp file, so that HSPICE can recognize it.

Note:

If you use .sp encryption, the encrypted data must not use `.INC`, `.LIB`, or `.LOAD`, to include another file.

The following is an example of an .sp file:

```
*sample.sp*
......

.lib 'cmos.lib' TT
.prot
.... $ data to be encrypted
.... $ do not include .inc .lib .load in encrypted data
.unpr
.inc sample2.inc
......
.end
```

*.lib File Encryption

You can place any important information into a *.lib* file, and encrypt it.

You can place parallel `..lib` commands into one library file, and encrypt each **.lib* separately. However, you must place `.PROT` and `.UNPROT` commands between each pair of `..lib` and `.endl` commands.

Note:

You must place `.PROT` and `.UNPROT` commands between `.lib` and `.endl` commands. To find the library name, HSPICE searches the **.lib* file first.

The following is an example of a **.lib* file:

```
*sample.lib*
.lib test1 $ .prot , .unpr should be put between
* .lib and .endl
.prot
...... $ data to be encrypted
.unpr
...... $ data not to be encrypted
.endl test1

.lib test2 $ .prot , .unpr should be put between
* .lib and .endl
.prot
...... $ data to be encrypted
.unpr
.endl test2

.lib test3
...... $ data
.endl test3
```

You use the above encrypted .lib file as you would any unencrypted one.

```
.lib './sample.lib' test1
.lib './sample.lib' test2
.lib './sample.lib' test3
......
.end
```

Example: Traditional (freelib) Encryption in an HSPICE Netlist

The following complete example illustrates the `metaencrypt` encryption structure. This example `enc.sp` netlist has three encrypted files: the `mm.spe`, the `xx.ic`, and the `kk.lib`.

```
file enc.sp:
*test .inc .lib .load encryption
.inc "mm.spe"
.load "xx.ic"
.lib ’kk.lib’ pch
.OPTION post list
.tran 2ns 400ns
.end

file mm.spe:
.prot CUSTOM
-hs#ylB]*7[
+t’Y=O$S[t0]ajL
+C :Nx:$.$=<*X:$<#pP=020#ZWP=020x\K:[1:898
y[-x:$-#tRr0($x#4:/[U$<\K:I[U$<J <9 :P#ZQ
6%P2V7D6:]4l/0#+:IXj0#ZWP=020#ZWP/[U$=J++bZ
3[7D6:BxHpg8
/C902P73+26
mh$y#D:bX/$\KwI)U-0R#=-ib+\[
a$o) :P.#$<) :P.#to)V:\7*K-I1M$#’;-[Xz:9qpy
eMDv0%wUoxZ>mzwF*-(3_;W6x.*P!uW.]a+P0.h:n=O>1q+H(J0
o.H#-/B+($;W Me*0x<6#9[UqpH/2h97%;-/B+T35Q
$\m;’_-he[uE$%H) 5a:ZxRW9x=*77w$2]=*P!RW%.ahT3VQ
H0[I:[

file xx.ic:

.prot FREELIB
59yUH\$=’x.3k77*<]8AT]8
<:7-(:9CV+7x15Xj+h’x=5Xj+(2 +4]8
<:7_D:\[2x9Y>/.7q
59y3\#D$ *y2k=u]PIq:97jH=u1w5Xj+x6
92k#<2FW0’k772<xBU677Q
59y3\#s# r21$],29b72[4’/RW72wd#$:O.U
+ 0sW%5$;[4sv;9=zV7[WFW[(g8#/’]=AH%T5:7Z
[$%C999A2P!8
<:X2o60’$ 06($_#upe1:pX8
<5ax/toC n90;<0dw0]23G%C z9$Dh#Sw5a90
ZM*2!M[0
o729!=PAy73x(/1:6[
+ 0%2UT%8
_:-x*$X+q
$9P2y73x(/1:L
T#;*9A27!j+(/z
$$o#(:/b0
o7ZW-9 -PxJ+y
a9[$0\;n90;<0dw0]23G%C z9$Dh#Sw5a90
Zr   ;6
```

```
file kk.lib:

.LIB NCH
.prot FREELIB
HO. T,# %fXz>MZWf*-(3_;w6X.*p!Uw.]A+p0.H:N=o>1Q+h(j0
o.H#-/B+($;W Me*0x<6#9[UqpH/2h97%;-/B+T35Q
$\m;'_-he[uE$%H) 5a:ZxRW9x=*77w$2]=*P!RW%.ahT3VQ
H0[I:[
.ENDL

.LIB PCH
.prot FREELIB
HO. T,#t%fXz>MZWf*-(3_;w6X.*p!Uw.]A+p0.H:N=o>1Q+h(j0
o.H#-/B+($;W Me*0x<6#9[UqpH/2h97%;-/B+T35Q
$\m;'_-he[uE$%H) 5a:ZxRW9x=*77w$2]=*P!RW%.ahT3VQH0[I:[
.ENDL
```

8-Byte Key Encryption

An 8-byte key encryption feature, based on a 56-bit DES, is available in `metaencrypt`. When using 8-byte encryption you can encrypt files with line lengths of 254 characters or shorter. The encrypted data is in binary format.

To encrypt a file, you provide a keyname that can contain alphabetic characters and numbers, and which is no longer than 8 bytes. To use the encrypted file, you must use the `.inc` command in the main netlist.

HSPICE supports, include file (*.inc*) encryption, when you use 8-byte key encryption. To use this encryption:

1. Insert the data to encrypt, into an include file.
2. Encrypt this file.

Follow these rules when you use 8-byte key encryption:

- 8-byte key encryption supports only *.inc* encryption.
- 8-byte encryption does NOT support `.LIB`, `.LOAD`, or `.OPTION SEARCH` encryption. Choose another form of encryption for these types of files.
- Do not include `.PROT` and `.UNPROT` when you perform 8-byte encryption.

- If keyname is an 8-byte string (combination of characters and numbers), then `metaencrypt` performs the 8-byte key encryption.
- In a .sp file, you cannot encrypt the first line because it is the title. You also cannot encrypt the last line because it marks the end of the file.

Creating 8-byte key Encryption

Use the following syntax to create 8-byte key encryption:

```
metaencrypt -i example.dat -o example.inc -t fGi85H9b
```

In the following example, example.dat contains the data to encrypt.

```
* DFF subckt netlist
$notice no .prot or .unprot used for this method
.subckt XGATE  control in n_control out
m0 in n_control out vdd pmos l=0.90u w=3.4u
m1 in control out gnd nmos l=0.90u w=3.4u .ends
.....
v14 vdd gnd dc=5
Xi3  net25 net31 net27 dff_nq DFF l=1u wn=3.8u wp=10u
Xi2  dff_nq d_output INV wp=26.4u wn=10.6u
.ends XGATE
```

Placing an 8-byte key Encrypted File into a HSPICE Netlist

The following fragment is an example of placing an 8-byte key encrypted file into an HSPICE Netlist:

```
* example.sp file using encrypted example.dat
.Options Post Brief NoMod
.Global vdd gnd
.lib 'demo.lib' TT
.inc 'example.inc'  $ this is the encrypted file
...
*
.Tran 1n 8n Sweep Optimize = Opt1
+                 Result   = MaxVout    $ Look at measure
+                 Model    = OptMod
.end
```

Triple DES Public and Random Keys

When using 3DES technology, the `metaencrypt` command checks for both the `encrypt` and `metaencrypt3des` license tokens. During simulation of a 3DES-encrypted netlist, HSPICE requires a `hspice3des` license token, for which you must meet export compliance guidelines to use HSPICE 3DES. Check with your local sales representative for this special license.

Note:

Files encrypted with the Triple DES algorithm cannot be read by previous releases of HSPICE.

The HSPICE triple DES encryption uses a 192-bit key to achieve a maximum level of security. You can generate the encryption keys for a new algorithm using one of the following options:

- 256-bit public key

  With this option, HSPICE generates a 256-bit public key during the encryption process. You need to distribute this key to the customer in order to run a simulation. The encrypted file and the generated public key is different every time the encryption is performed, even with the same private key and input file. This allows you to generate different encrypted file and public key combinations for different customers.

Your customers cannot access the key used for encryption, but they can run a simulation on a circuit by putting this public key file in the directory from which the simulation is run.

Multiple encrypted files are supported. You have to put all relevant public key files in the directory from which the simulation is run. You can also generate these encrypted files with the same private key or different private keys. The actual “key” needs to be a user-generated 192-bit string. For example, the following two files would be encrypted with the same key:

`metaencrypt -i a.dat -o a.inc -t` *<privkey>*

`metaencrypt -i b.dat -o b.inc -t` *<privkey>*

Two public key files are created: a.inc.key, and b.inc.key. These files are different even if the same private key is used in the encryption. A simulation run with a netlist file containing `.include` a.inc and `.include` b.inc commands requires that both key files be in the simulation directory.

- 192-bit random key

  With this option, HSPICE does not need an additional public key. The usage model for this option is consistent with other encryption options in previous releases of HSPICE. The encrypted file is different every time encryption is run.

Commands Not Supported by 3DES

- Embedded `.INC` encryption, which causes confusion in decryption.
- The file that you are encrypting cannot contain an `.inc`, `.lib`, or `.load` command that calls another file.

Creating 3DES Encrypted Files

Observe these rules when creating TripleDES encrypted files:

- You can use embedded `.LIB` encryption only if you set it up using `.prot` and `.unprot` inside of the `.lib` plus use the `-d` option.
- Do not use `.OPTION SEARCH`, when you encrypt models and subcircuits. (The old metaencryption functionality supported this method.) To directly encrypt subcircuits and model libraries, use the traditional `.INC` and `.LIB` encryption method.

Random Key Example

For files *without* embedded `.lib`, `.inc`, or `.load` commands:

```
metaencrypt -i dff.sp -o dff_rand.spe -t randkey
```

Note:

This netlist file has no `.prot` or `.unprot` commands in it, similar to 8-byte encryption.

For files *with* embedded `.lib` commands:

```
metaencrypt -i  demo.lib -o demo_rand.lib -t randkey
+ -d ./lib_rand
```

Note:

The *demo.lib* for this has the same `.prot/.unprot` setup as for traditional freelib.

Public Key Example

For files without embedded `.lib`, `.inc`, or `.load` commands:

```
metaencrypt -i dff.sp -o dff_priv.spe -t privkey
0123456789ABCDEF9876543210FEDCBA1357924680ACEBDF
```

Note:

This netlist file has no `.prot` or `.unprot` commands in it, similar to 8-byte encryption.

For a file that has an embedded `.lib` command:

```
metaencrypt -i  demo.lib -o demo_priv.lib -t privkey
0123456789ABCDEF9876543210FEDCBA1357924680ACEBDF -d ./lib_priv
```

Note:

The *demo.lib* for this has the same `.prot/.unprot` setup as for traditional freelib.

Placing 3DES Encryption Files into a HSPICE Netlist

While using a random or private key method for tripleDES may look similar, the private one requires that the key be included in the same directory.

Example 1 Random TripleDES

```
* Example netlist for including random TripleDES
.Options Post Brief NoMod
.Global vdd gnd
.lib 'demo_rand.lib' TT   $ bring in the random 3DES lib file,
*that looks at the ./lib_rand directory for files
.inc 'dff_rand.spe'      $ bring in random 3DES encrypted design
*file
....
.end
```

Example 2 Private TripleDES

```
* Example netlist for including private TripleDES
.Options Post Brief NoMod
.Global vdd gnd
.lib 'demo_priv.lib' TT   $ bring in private key lib file; the
*keys are in the ./lib_priv directory along with the files
.inc 'dff_priv.spe'      $ bring in private key encrypted file;
*key must be in this same directory (dff_priv.spe.key)
.....
.end
```

Troubleshooting Issues

‘Bad encryption format’ or ‘version check failed’ error

If you receive new encrypted models from a vendor and are getting a “Bad encryption format’ or “version check failed” error, your models were encrypted with the Triple DES (3DES) encryption technology. Because of export restrictions on the strong encryption technology, a separate license is required. To verify, view the HSPICE listing file to see a failed license checkout attempt:

```
lic: Unable to checkout hspice3des
lic:
lic: Release hspice3des token(s)
lic: feature hspice3des not found
```

3DES licenses are provided at no charge. Have your site contact call your Synopsys sales representative and request that they add a $0 triple DES line item, part #4279 to your existing PO. The number of 3DES licenses requested should equal the number of simulations (up to the quantity of HSPICE licenses you own) that you simultaneously simulate using triple DES encrypted models.

warning parameters... as an expression containing output signals

This warning occurs even when there are no explicit encrypted blocks in the netlist. There are two reasons for this warning message.

- `.protect` and `.unprotect` commands are in the netlist.
- The results of parameter expressions which contain output signals are not correct. For example:

```
.param myfunc (one,two)='abs (one - two)'
.param test=myfunc(v(1),v(2))
.protect
.if ( test <= 1 )
.param k='2*1p'
.elseif ( test <= 4 )
.param k='8*1p'
.else
.param k='1*test*1p'
.endif
.unprotect
c1 2 0 c=k
```

Part: 2 Elements and Devices

The following chapters/topics are included in this Part:

8 Elements

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

Elements are local and sometimes customized instances of a device model specified in your design netlist. Element names can be up to 1024 characters.

For descriptions of the standard device models on which elements (instances) are based, see the *HSPICE Reference Manual: Elements and Device Models* and the *HSPICE Reference Manual: MOSFET Models*. For signal integrity applications see the *HSPICE User Guide: Signal Integrity*.

These topics are covered in the following sections:

- Passive Elements
- Multi-Terminal Linear Elements
- Active Elements
- IBIS Buffers (HSPICE Only)

Passive Elements

This section describes the passive elements: resistors, capacitors, and inductors. See Multi-Terminal Linear Elements for discussion of the W-, U-, and S-elements. See also, T-element (Ideal Transmission Lines) in the *HSPICE User Guide: Signal Integrity*.

The content of this section includes:

- Values for Elements
- Resistor Elements in a HSPICE or HSPICE RF Netlist
- Linear Resistors

- Behavioral Resistors in HSPICE or HSPICE RF
- Frequency-Dependent Resistors
- Skin Effect Resistors
- Capacitors
- Linear Resistors
- Frequency-Dependent Capacitors
- Behavioral Resistors in HSPICE or HSPICE RF
- DC Block Capacitors
- Charge-Conserved Capacitors
- Inductors
- Mutual Inductors
- Ideal Transformer
- Linear Inductors
- Frequency-Dependent Inductors
- AC Choke Inductors
- Reluctors

Values for Elements

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

Resistor Elements in a HSPICE or HSPICE RF Netlist

```
Rxxx n1 n2 <mname> Rval [TC1 TC2 TC3] [SCALE=val] [M=val]
+ [AC=val] [DTEMP=val] [L=val] [W=val] [C=val]
+ [NOISE=val]

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

```
Rxxx n1 n2 R='equation' …
```

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

Parameter	Description
W	Resistor width. Default=0.0, if you did not specify W in the model.
C	Capacitance connected from node n2 to bulk. Default=0.0, if you did not specify C in a resistor model
user-defined equation	Can be a function of any node voltages, element currents, temperature, frequency, or time
NOISE	▪ NOISE=0, do not evaluate resistor noise. ▪ NOISE=1, evaluate resistor noise (default).

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

HSPICE Examples

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

```
R1 Rnode1 Rnode2 100
```

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

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

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

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

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

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

HSPICE RF Examples

Some basic examples for HSPICE RF include:

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

  ```
  R1 1 0 ‘v(c)’
  ```

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

  ```
  R2 1 0 ‘abs(v(c)) + abs(v(d))’
  ```

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

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

Linear Resistors

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

Parameter	Description
Rxxx	Name of a resistor
node1 and node2	Names or numbers of the connecting nodes
modelname	Name of the resistor model
value	Nominal resistance value, in ohms
R	Resistance, in ohms, at room temperature
TC1, TC2	Temperature coefficient
W	Resistor width
L	Resistor length
M	Parallel multiplier
C	Parasitic capacitance between node2 and the substrate
DTEMP	Temperature difference between element and circuit
SCALE	Scaling factor

Example

```
R1 1 2 10.0
Rload 1 GND RVAL

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

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

Note:

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

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

Behavioral Resistors in HSPICE or HSPICE RF

```
Rxxx n1 n2 . . . [R=] 'equation' . . .
```

Note:

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

Example

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

Frequency-Dependent Resistors

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

```
+ <FMAX=value>>
```

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

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

Example

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

Skin Effect Resistors

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

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

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

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

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

Capacitors

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

Polynomial form:

```
Cxxx n1 n2 POLY c0 c1... <IC=val> <M=val>
```

Parameter	Description
Cxxx	Capacitor element name. Must begin with C, followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name.
n2	Negative terminal node name.
mname	Capacitance model name. Elements use this name to reference a capacitor.
C=capacitance	Capacitance at room temperature—a numeric value or a parameter in farads.
TC1	First-order temperature coefficient for the capacitor. See the Passive Device Models chapter in the *HSPICE Elements and Device Models Manual* for temperature-dependent relations.
TC2	Second-order temperature coefficient for the capacitor.
SCALE	Element scale parameter, scales capacitance by its value. Default=1.0.

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

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

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

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

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

Example 1

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

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

Example 2

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

```
C1 1 2 20p
```

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

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

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

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

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

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

Linear Capacitors

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

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

Example

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

In this example:

- Cbypass is a straightforward, 10-picofarad (PF) capacitor.
- C1, which calls the CBX model, does not have a constant capacitance.
- CB is a 10 PF capacitor, with an initial voltage of 4V across it.
- CP is a 0.1 PF capacitor.

Frequency-Dependent Capacitors

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

Syntax

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

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

Example

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

Behavioral Capacitors in HSPICE or HSPICE RF

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

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

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

Example

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

DC Block Capacitors

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

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

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

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

Charge-Conserved Capacitors

```
Cxxx node1 node2 q='expression'
```

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

The `expression` supports the following parameters and variables:

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

    Note:

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

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

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

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

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

Capacitor equations are not implicitly converted to charge equations.

Example 1: Capacitance-based Capacitor

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

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

Example 2: Charge-based Capacitor

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

Example 3: Capacitance-based Capacitor

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

Example 4: Charge-based Capacitor

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

Inductors

General form:

```
Lxxx n1 n2 <L=>inductance <<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...>
```

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

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

Parameter	Description
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.

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

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

Example 1

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

```
L1 coilin coilout 100n
```

Example 2

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

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

Example 3

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

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

Example 4

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

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

Example 5

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

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

Mutual Inductors

General form:

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

Mutual core form:

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

Parameter	Description
Kxxx	Mutual inductor element name. Must begin with K, followed by up to 1023 alphanumeric characters.
Lyyy	Name of the first of two coupled inductors.
Lzzz	Name of the second of two coupled inductors.
K=coupling	Coefficient of mutual coupling. K is a unitless number, with magnitude > 0. If K is negative, the direction of coupling reverses. This is equivalent to reversing the polarity of either of the coupled inductors. Use the K=coupling syntax when using a parameter value or an equation, and the keyword “k=” can be omitted.
Kaaa	Saturable core element name. Must begin with K, followed by up to 1023 alphanumeric characters.
Lbbb, Lccc, Lddd	Names of the windings about the Kaaa core. One winding element is required, and each winding element must use the magnetic winding syntax. All winding elements with the same magnetic core model should be written in one mutual inductor statement in the netlist.
mname	Saturable core model name. (See the Passive Device Models chapter in the *HSPICE Elements and Device Models Manual* for more information.)

Parameter	Description
MAG= magnetization	Initial magnetization of the saturable core. You can set this to +1, 0, or -1, where +/- 1 refer to positive and negative values of the BS model parameter. (See the Passive Device Models chapter in the *HSPICE Elements and Device Models Manual* for more information.)

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

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

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

Note:

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

Example 1

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

```
K1 Lin Lout 0.9
```

Example 2

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

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

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

Example 3

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

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

Ideal Transformer

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

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

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

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

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

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

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

Example 1

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

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

Example 2

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

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

Example 3

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

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

Linear Inductors

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

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

Example:

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

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

Frequency-Dependent Inductors

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

Syntax

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

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

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

Example

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

AC Choke Inductors

Syntax

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

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

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

Reluctors

Syntax

Reluctance Inline Form

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

Reluctance External File Form

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

Parameter	Description
Lxxx	Name of a reluctor. Must begin with L, followed by up to 1023 alphanumeric characters
n1p n1n ... nNp nNn	Names of the connecting terminal nodes. The number of terminals must be even. Each pair of ports represents the location of an inductor.
RELUCTANCE	Keyword to specify reluctance (inverse inductance).
r1, c1, val1, r2, c2, val2, ... rm, cm, valm	Reluctance matrix data. In general, K will be sparse and only non-zero values in the matrix need be given. Each matrix entry is represented by a triplet (r,c,val). The value r and c are integers referring to a pair of inductors from the list of terminal nodes. If there are 2*N terminal nodes, there will be N inductors, and the r and c values must be in the range [1,N]. The val value is a reluctance value for the (r,c) matrix location, and the unit for reluctance is the inverse Henry (H^{-1}). Only terms along and above the diagonal are specified for the reluctance_matrix. The simulator fills in the lower triangle to ensure symmetry. If you specify lower diagonal terms, the simulator converts that entry to the appropriate upper diagonal term. If multiple entries are supplied for the same (r,c) location, then only the first one is used, and a warning will be issued indicating that some entries are ignored. All diagonal entries of the reluctance matrix must be assigned a positive value. The reluctance matrix should be positive definite.
FILE=”<filename1>”	For the external file format, the data files should contain three columns of data. Each row should contain an (r,c,val) triplet separated by white space. The r, c, and val values may be expressions surrounded by single quotes. Multiple files may be specified to allow the reluctance data to be spread over several files if necessary.

Parameter	Description
SHORTALL	▪ SHORTALL=yes, all inductors in this model are converted to short circuits, and all reluctance matrix values are ignored. ▪ SHORTALL=no (default), inductors are not converted to short circuits, and reluctance matrix values are not ignored.
IGNORE_COUPLING	▪ IGNORE_COUPLING=yes, all off-diagonal terms are ignored (that is, set to zero). ▪ IGNORE_COUPLING=no (default), off-diagonal terms are not ignored.

Example

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

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

Alternatively, the same element could be specified by using:

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

Where reluctance.dat contains:

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

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

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

Multi-Terminal Linear Elements

A multi-terminal linear element such as 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 is voltage between the conductors that is transmitted from one end of the pair to the other end.

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

The following sections discuss:

- W-element (Distributed Transmission Lines)
- U-element (Lumped Transmission Lines)
- S-element (Generic Multiport)

W-element (Distributed Transmission Lines)

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

- Model 1: RLGC-Model specification
  - Internally specified in a .model statement
  - Externally specified in a different file
- Model 2: U-Model specification
  - RLGC input for up to five coupled conductors
  - Geometric input (planer, coax, twin-lead)
  - Measured-parameter input
  - Skin effect
- Model 3: Built-in field solver model
  - Standard format (using geometric data with the W-element)
  - Tabular format
- Model 4: Frequency-dependent tabular model
- Model 5: S-parameter Model

W-element Statement

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

RLGC file form:

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

U Model form:

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

Field solver form:

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

Table Model form:

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

S Model form:

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

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 xth transmission line (in1 is required).
refin	Ground reference for input signal
outx	Signal output node for the xth transmission line (each input port must have a corresponding output port).
refout	Ground reference for output signal.
N	Number of conductors (excluding the reference conductor).
L	Physical length of the transmission line, in units of meters.
RLGCfile=filename	File name reference for the file containing the RLGC information for the transmission lines (for syntax, see Using the W-element in the *HSPICE Signal Integrity Guide*).
Umodel=modelname	U-model lossy transmission-line model reference name. A lossy transmission line model, used to represent the characteristics of the W-element transmission line.
FSmodel= modelname	Internal field solver model name. References the PETL internal field solver as the source of the transmission-line characteristics (for syntax, see Using the Field Solver Model section in the *HSPICE Signal Integrity Guide*).

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

The number of ports on a single transmission line is 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.

Example 1

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

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

Where,

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

Example 2

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

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

Where,

- in1 and in2 input nodes connect to the out1 and out2 output node
- Both signal references are grounded.

- umod_1 references the U-model.
- The transmission line is 10 meters long.

Example 3

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

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

Where,

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

Example 4: S Model Example

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

Where,

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

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

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

Figure 16 shows node numbering for the element syntax.

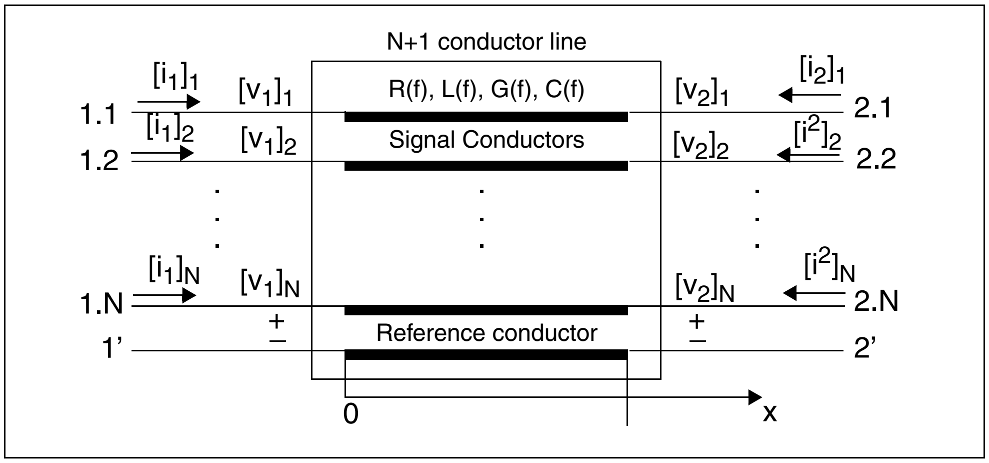


Figure 16 Terminal Node Numbering for the W-element

For additional information about the W-element, see the W-element Modeling of Coupled Transmission Lines chapter in the *HSPICE User Guide: Signal Integrity*.

U-element (Lumped Transmission Lines)

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

Parameter	**Description**
Uxxx	Lossy (U-element) transmission line element name. Must begin with U, followed by up to 1023 alphanumeric characters.
inx	Signal input node for the x^{th} transmission line (in1 is required).
refin	Ground reference for the input signal.
outx	Signal output node for the x^{th} transmission line (each input port must have a corresponding output port).
refout	Ground reference for the output signal.
mname	Model reference name for the U-model lossy transmission-line.

Parameter	Description
L	Physical length of the transmission line, in units of meters.
LUMPS	Number of lumped-parameter sections used to simulate the element.

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

Example 1

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

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

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

Example 2

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

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

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

Example 3

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

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

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

S-element (Generic Multiport)

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

- S (scattering)
- Y (admittance)

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

- DC
- AC
- Transient
- Small Signal

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

Group Delay Handler in Time Domain Analysis

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

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

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

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

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

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

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

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

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

Preconditioning S-parameters

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

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

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

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

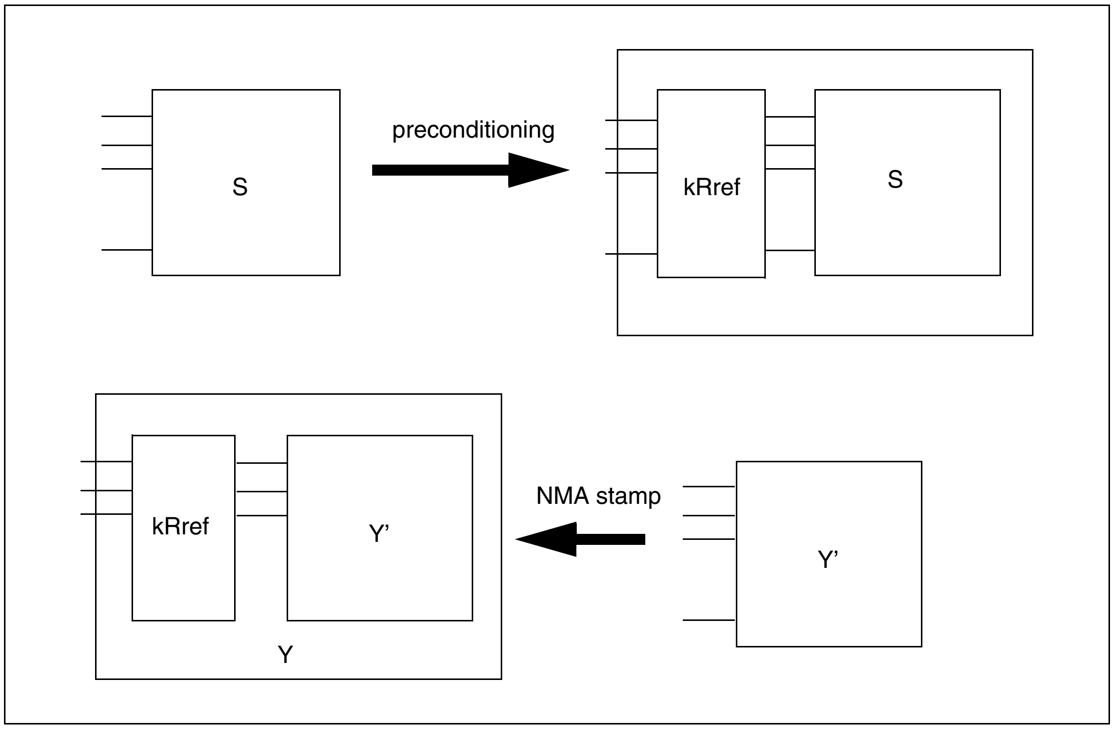


Figure 17 Preconditioning S-parameters

Active Elements

This section describes the active elements: diodes and transistors.

Diode Element

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

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

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

Fowler-Nordheim (`LEVEL=2`) form:

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

Parameter	Description
Dxxx	Diode element name. Must begin with D, followed by up to 1023 alphanumeric characters.
nplus	Positive terminal (anode) node name. The series resistor for the equivalent circuit is attached to this terminal.
nminus	Negative terminal (cathode) node name.
mname	Diode model name reference.
AREA	Area of the diode (unitless for LEVEL=1 diode, and square meters for LEVEL=3 diode). This affects saturation currents, capacitances, and resistances (diode model parameters are IK, IKR, JS, CJO, and RS). The SCALE option does not affect the area factor for the LEVEL=1 diode. Default=1.0. Overrides AREA from the diode model. If you do not specify the AREA, HSPICE or HSPICE RF calculates it from the width and length.
PJ	Periphery of junction (unitless for LEVEL=1 diode, and meters for LEVEL=3 diode). Overrides PJ from the diode model. If you do not specify PJ, HSPICE or HSPICE RF calculates it from the width and length specifications.
WP	Width of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides WP in the diode model. Default=0.0.
LP	Length of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides LP in the diode model. Default=0.0.
WM	Width of metal capacitor, in meters (for LEVEL=3 diode only). Overrides WM in the diode model. Default=0.0.
LM	Length of metal capacitor, in meters (for LEVEL=3 diode only). Overrides LM in the diode model. Default=0.0.
OFF	Sets the initial condition for this element to OFF, in DC analysis. Default=ON.

Parameter	Description
IC=vd	Initial voltage, across the diode element. Use this value when you specify the UIC option in the .TRAN statement. The .IC statement overrides this value.
M	Multiplier, to simulate multiple diodes in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
W	Width of the diode, in meters (LEVEL=3 diode model only)
L	Length of the diode, in meters (LEVEL=3 diode model only)

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

Example 1

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

```
D1 1 2 diode1
```

Example 2

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

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

Example 3

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

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

Bipolar Junction Transistor (BJT) Element

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

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

Parameter	Description
Qxxx	BJT element name. Must begin with Q, then up to 1023 alphanumeric characters.
nc	Collector terminal node name.
nb	Base terminal node name.
ne	Emitter terminal node name.
ns	Substrate terminal node name, which is optional. You can also use the BULK parameter to set this name in the BJT model.
mname	BJT model name reference.
area, AREA=area	Emitter area multiplying factor, which affects currents, resistances, and capacitances. Default=1.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON.
IC=vbeval, vceval, VBE, VCE	Initial internal base-emitter voltage (vbeval) and collector-emitter voltage (vceval). HSPICE or HSPICE RF uses this value when the .TRAN statement includes UIC. The .IC statement overrides it.
M	Multiplier, to simulate multiple BJTs in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
AREAB	Base area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.
AREAC	Collector area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.

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

Example 1

In the Q1 BJT element:

```
Q1 1 2 3 model_1
```

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

Example 2

In the following Qopamp1 BJT element:

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

- The collector connects to the c1 node.
- The base connects to the b3 node.
- The emitter connects to the e2 node.
- The substrate connects to the s node.
- 1stagepnp references the BJT model.
- The `AREA` area factor is 1.5.
- The `AREAB` area factor is 2.5.
- The `AREAC` area factor is 3.0.

Example 3

In the Qdrive BJT element:

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

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

JFETs and MESFETs

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

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

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

Parameter	Description
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.

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

Example 1

In the J1 JFET element:

```
J1 1 2 3 model_1
```

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

Example 2

In the following Jopamp1 JFET element:

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

- The drain connects to the d1 node.
- The source connects to the g3 node.
- The gate connects to the s2 node.
- 1stage references the JFET model.
- The area is 100 microns.

Example 3

In the Jdrive JFET element:

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

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

MOSFETs

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

Parameter	Description
Mxxx	MOSFET element name. Must begin with M, followed by up to 1023 alphanumeric characters.
nd	Drain terminal node name.
ng	Gate terminal node name.
ns	Source terminal node name.
nb	Bulk terminal node name, which is optional. To set this argument in the MOSFET model, use the BULK parameter.
mname	MOSFET model name reference or subckt name if .OPTION MACMOD is set.
L	MOSFET channel length, in meters. This parameter overrides .OPTION DEFL, with a maximum value of 0.1m. Default=DEFL.
W	MOSFET channel width, in meters. This parameter overrides .OPTION DEFW. Default=DEFW.
AD	Drain diffusion area. Overrides .OPTION DEFAD. Default=DEFAD, if you set the ACM=0 model parameter.
AS	Source diffusion area. Overrides .OPTION DEFAS. Default=DEFAS, if you set the ACM=0 model parameter.
PD	Perimeter of drain junction, including channel edge. Overrides.OPTION DEFPD. Default=DEFAD, if you set the ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.

Parameter	Description
PS	Perimeter of source junction, including channel edge. Overrides .OPTION DEFPS. Default=DEFAS, if you set the ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
NRD	Number of squares of drain diffusion for resistance calculations. Overrides .OPTION DEFNRD. Default=DEFNRD, if you set ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
NRS	Number of squares of source diffusion for resistance calculations. Overrides .OPTION DEFNRS. Default=DEFNRS when you set the MOSFET model parameter ACM=0 or 1. Default=0.0, when you set ACM=2 or 3.
RDC	Additional drain resistance due to contact resistance, in units of ohms. This value overrides the RDC setting in the MOSFET model specification. Default=0.0.
RSC	Additional source resistance due to contact resistance, in units of ohms. This value overrides the RSC setting in the MOSFET model specification. Default=0.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON. This command does not work for depletion devices.
IC=vds, vgs, vbs	Initial voltage across external drain and source (vds), gate and source (vgs), and bulk and source terminals (vbs). Use these arguments with `.TRAN UIC`. `.IC` statements override these values.
M	Multiplier, to simulate multiple MOSFETs in parallel. Affects all channel widths, diode leakages, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
GEO	Source/drain sharing selector for a MOSFET model parameter value of ACM=3. Default=0.0.
DELVTO	Zero-bias threshold voltage shift. Default=0.0.

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

Example

In the following M1 MOSFET element:

```
M1 1 2 3 model_1
```

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

In the following Mopamp1 MOSFET element:

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

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

In the following Mdrive MOSFET element:

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

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

Extended MOSFET Element Support Using .OPTION MACMOD

Use option MACMOD to enable HSPICE MOSFET to access a subckt definition when no model reference exists. .OPTION MACMOD syntax is:

```
.OPTION MACMOD= [1|2|3|0]
```

When `macmod=1`, HSPICE seeks a subckt definition for the M*** element if no model reference exists. The desired subckt name must match (case insensitive) the `mname` field in the M*** instance statement. In addition, the number of terminals of the subckt must match with the M*** element referencing it; otherwise HSPICE aborts the simulation based on no definition for the M*** element.

The following limitations apply when `macmod=1`:

1. Element template output does not support MOSFET elements which use subckt definitions.
2. This feature will not support a MOSFET element whose `mname` is defined by a string parameter.
3. The number of terminals for a HSPICE MOSFET element must be within the range of 3-7; any number of terminals that is out of this range will cause the simulation to fail.

When `macmod=2`, HSPICE seeks a MOSFET model definition when it cannot find matching subckt or Verilog-A definition for an X-element. The targeted MOSFET MODEL card could be either HSPICE built-in MOSFET model or CMI MOSFET model. If the model card that matched with the X-element reference name is not a type of MOSFET models, simulator errors out with message of reference not found.

The following limitations apply when `macmod=2`:

1. The feature of “string parameter supported in MOSFET model name” is not applied to X-elements that are mapped to MOSFET model cards; i.e., reference name of the X-element must be constant string characters.
2. Subckt direct port probing command, `isub()` is not supported on X-elements mapped to MOSFET model cards.
3. HSPICE MOSRA analysis may not be performed on the X-elements, even when they direct map to MOSFET model cards.

When `macmod=3`, HSPICE enables both of the above features; HSPICE seeks a `.subckt` definition for an M-element if there is no matching model reference; HSPICE seeks a `.model` MOSFET definition for an X-element if there is no matching `.subckt` or Verilog-A definition. Usage considerations and limitations remain the same for both features, respectively.

Note:

> When MACMOD=2 or 3, for the X-element that maps to an M-element, if it has an instance parameter named 'Multi' (case insensitive), then 'Multi' is aliased to the 'M' factor, the M (multiply) parameter.

When macmod=0, if there is no .option MACMOD in the input files or MACMOD=0, then neither of the above two features is enabled; HSPICE ignores the MACMOD option when any value other than 1|2|3|0 is set.

The MACMOD option is a global option; if there are multiple MACMOD options in one simulation, HSPICE uses the value of the last MACMOD option.

Example 1

```
**
.option MACMOD=1
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=1
.model nch nmos level=49 ….
.subckt nch d g s b w=1 l=1 p1=gp1
.if (p1 > 0)
Mnch d g s b model_1 w=w l=l
.else
Mnch d g s b model_2 w=w l=l
.endif
.ends
```

In Example 1, extended MOSFET is turned on. However, because the mname in a .MODEL statement matches the mname of the M1 element, element M1uses model nch rather than the subckt definition. The extra instance parameter p1 is ignored.

Example 2

```
**
.option MACMOD
.param gp1=1 gp2=2 gp3=3
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=gp1 p2=gp2 p3=gp3
.subckt nch d g s b w=1 l=1 p1=gp1 p2=gp2 p3=gp3
.if (p1 > 0 && p2==1 && p3 ==1)
Mnch d g s b model_1 w=w l=l
.else if ( p1 == 0 && p2 ==1 && p3 ==1)
Mnch d g s b model_2 w=w l=l
.else
Mnch d g s b model_3 w=w l=l
.endif
.ends
.model model_1 nmos level=49 ...
.model model_2 nmos level=53 ...
.model model_3 nmos level=54 ...
```

In Example 2, extended MOSFET element support is turned on; since there is no matching `.MODEL` statement, `M1` uses subckt definition `nch`; after evaluation, `M1` results in a MOSFET element using MOSFET model `model_3`.

Example 3

```
**
.option MACMOD
.param gp1=1 gp2=2 gp3=3
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=gp1 p2=gp2 p3=gp3
.subckt nch d g s b w=1 l=1 p1=gp1 p2=gp2 p3=gp3
.if (p1 > 0 && p2==1 && p3 ==1)
Mnch d g s b model_1 w=w l=l
.else if ( p1 == 0 && p2 ==1 && p3 ==1)
Mnch d g s b model_2 w=w l=l
.else
Mnch d g s b model_3 w=w l=l
.endif
C1 g 0 1p
.ends
```

Example 3 shows extended MOSFET element support turned on. Instance `M1` uses macro model `nch`, which is a subckt definition that consists of one MOSFET device and one capacitor.

MACMOD Option Limitations

- The number of terminals for M*** must be within the range of 3 to 7. A number of terminals outside of that range cause the simulation to fail.
- This feature does not support a MOSFET element whose `mname` is defined by a string parameter.
- The MACMOD option only applies to HSPICE MOSFET elements.
- Element template output does not support MOSFET elements which use subckt definitions.

For example, if Example 3 includes the output command:

```
.PRINT LX8(M1) LV9(M1)
```

—then HSPICE ignores the above output command. Because `M1` is using a subckt definition, it is no longer a HSPICE primitive MOSFET device.

- The desired subckt name must match the `mname` field in the M*** instance statement. The match of subckt name and the `mname` field is case insensitive.
- The .MODEL definition will always be used over a subckt definition even when .OPTION MACMOD is on.

Direct X-Element Mapping to a MOSFET Model Card

HSPICE will look for a MOSFET model definition when it cannot find a matching `subckt` or Verilog-A definition for an X-element. This applies mainly to a CMI model, which is turned on only when the `.option CMIFLAG` is set; Subckt or Verilog-A module definitions will always take preference over the CMI model card.

The X-elements that are mapped to model cards are treated as HSPICE MOSFET devices with certain usage considerations and limitations described in the following sections.

For information about the HSPICE CMI, contact your Synopsys support team.

Considerations

Syntax check rules of MOSFET devices will be applied to the X-elements directly mapping to MOSFET model cards, such as:

1. The valid instance parameter list of that X-element will then be the valid instance parameter list of the particular MOSFET model defined in the mapping model card, any invalid instance parameter of that X-element will be ignored; any undefined instance parameter comparing with the corresponding MOSFET model, is set to default values of the mapping MOSFET model; implicit instance parameters of the mapping MOSFET model are applied to such X-elements instead of the implicit parameter set of a conventional X-element.
2. MOSFET instance parameters are device native parameters, not the netlist parameter, so they cannot be overridden by the `.PARAM` netlist commands; such instance parameter rules are applied to X-elements with direct mapping to MOSFET model cards, i.e., instance parameters of such an X-element become native device parameters, thus cannot be overridden by netlist `.PARAM` commands.
3. The number of terminals of the X-element must be in the valid range of its mapping MOSFET model; or simulator exits with an error message.

Limitations

1. The feature of “string parameter supported on MOSFET model name” is not applied to X-elements that are mapped to MOSFET model cards. In other words, a reference name of the X-element must be written as constant string characters.
2. The Subckt direct port probing command, `Isub()` is not supported on X-elements mapped to MOSFET model cards.
3. HSPICE MOSRA analysis may not be performed on the X-elements, even if they directly map to MOSFET model cards.

IBIS Buffers (HSPICE Only)

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

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

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

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 the Modeling Input/Output Buffers Using IBIS Files chapter in the *HSPICE User Guide: Signal Integrity*.

9
Sources and Stimuli

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

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

These topics are discussed in the following sections:

- Independent Source Elements
- Independent Source Functions
- Voltage and Current Controlled Elements
- Voltage-Dependent Voltage Sources — E-elements
- Current-Dependent Current Sources — F-elements
- Voltage-Dependent Current Sources — G-elements
- Current-Dependent Voltage Sources — H-elements
- Specifying a Digital Vector File and Mixed Mode Stimuli

Independent Source Elements

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

Source Element Conventions

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

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

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

Independent Source Element Syntax

```
Vxxx n+ n- [[DC=] dcval tranfun [AC=acmag acphase]]
```

```
Ixxx n+ n- [[DC=] dcval tranfun [AC=acmag acphase]]
+ [M=val]
```

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

Parameter	Description
M	Multiplier, to simulate multiple parallel current sources. HSPICE or HSPICE RF multiplies source current by M. Default=1.0.

Example 1

```
VX 1 0 5V
```

Where,

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

Example 2

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

Where,

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

Example 3

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

Where,

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

Example 4

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

Where,

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

Example 5

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

Where,

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

Example 6

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

Where,

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

Example 7

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

Where,

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

Example 8

```
VMEAS 12 9
```

Where,

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

DC Sources

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

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

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

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

AC Sources

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

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

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

Transient Sources

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

- Trapezoidal pulse (`PULSE` function)
- Sinusoidal (`SIN` function)
- Exponential (`EXP` function)

- Piecewise linear (`PWL` function)
- Single-frequency FM (`SFFM` function)
- Single-frequency AM (`AM` function)
- Pattern (`PAT` function)
-

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

Mixed Sources

Mixed sources specify source values for more than one type of analysis. For example, you can specify a DC source, an AC source, and a transient source, all of which connect to the same nodes. In this case, when you run specific analyses, HSPICE or HSPICE RF selects the appropriate DC, AC, or transient source. If the mixed source DC value is missing, the zero-time value of its transient source will be used as the DC value by default. Otherwise, for DC analysis, the DC source value is used by the program, and is selected for operating point calculation for all analysis except TRAN; for TRAN analysis, an additional operating point is calculation with the zero-time source transient value.

Example

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

Where,

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

Each source connects between nodes 13 and 2.

For DC analysis, the program uses its dc value 0.5v, and this operating point is selected for the coming AC analysis. For transient analysis, another operating point is calculated by using zero source value because the sinusoidal source is zero at time zero.

Port Element

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

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

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

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

The port element accepts transient waveforms AM, EXP, PULSE, PWL, SFFM, SIN, and, for signal integrity usage, the PAT source.

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

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

Parameter	Description
HB *mag* p*hase harm tone modharm modtone*	(HSPICE RF) HB voltage, current, or power source value. Multiple HB specifications with different harm, tone, modharm, and modtone values are allowed. ■ phase is in degrees ■ harm and tone are indices corresponding to the tones specified in the .HB statement. Indexing starts at 1 (corresponding to the first harmonic of a tone). ■ modtone and modharm specify sources for multi-tone simulation. A source specifies a tone and a harmonic, and up to 1 offset tone and harmonic (modtone for tones and modharm for harmonics). The signal is then described as: V(or I)=mag*cos(2*pi* (harm*tone+modharm*modtone)*t + phase)
transient_waveform	(Transient analysis) Voltage or power source waveform. Any one of waveforms: AM, EXP, PULSE, PWL, SFFM, SIN, or PRBS. Multiple transient descriptions are not allowed.
TRANFORHB=[0|1]	■ (HSPICE RF) 0 (default): The transient description is ignored if an HB value is given or a DC value is given. If no DC or HB value is given and TRANFORHB=0, then HB analysis treats the source as a DC source, and the DC source value is the time=0 value. ■ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, then the time=infinity value is used as a DC analysis source value. For example, the following statement is treated as a DC source with value=1 for HB analysis: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=1 In contrast, the following statement is a 0V DC source: v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=0 The following statement is treated as a periodic source with a 1us period that uses PWL values: v1 1 0 PWL (0 0 1n 1 0.999u 1 1u 0) R + TRANFORHB=1 To override the global TRANFORHB option, explicitly set TRANFORHB for a voltage or current source.

Parameter	Description
DCOPEN	Switch for open DC connection when DC mag is not set. ▪ 0 (default): P element behaves as an impedance termination. ▪ 1 : P element is considered an open circuit in DC operating point analysis. DCOPEN=1 is mainly used in .LIN analysis so the P element will not affect the self-biasing device under test by opening the termination at the operating point.
z0=*val*>	(LIN analysis) System impedance used when converting to a power source, inserted in series with the voltage source. Currently, this only supports real impedance. ▪ When power=0, z0 defaults to 0. ▪ When power=1, z0 defaults to 50 ohms. You can also enter zo=val.
RDC=*val*	(DC analysis) Series resistance (overrides `z0`).
RAC=*val*	(AC analysis) Series resistance (overrides `z0`).
RHBAC=*val*	(HSPICE RF HBAC analysis) Series resistance (overrides `z0`).
RHB=*val*	(HSPICE RF HB analysis) Series resistance (overrides `z0`).
RTRAN=*val*	(Transient analysis) Series resistance (overrides `z0`).
power=[0 | 1 | 2 | W | dbm]	(HSPICE RF) power switch ▪ When 0 (default), element treated as a voltage or current source. ▪ When 1 or W, element treated as a power source, realized as a voltage source with a series impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ▪ When 2 or dbm, element treated as a power source in series with the port impedance. Values are in dbms. You can use this parameter for transient analysis if the power source is either DC or SIN.

Example

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

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

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

Independent Source Functions

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

- Trapezoidal pulse (`PULSE` function)
- Sinusoidal (`SIN` function)
- Exponential (`EXP` function)
- Piecewise linear (`PWL` function)
- Single-frequency FM (`SFFM` function)
- Single-frequency AM (`AM` function)
- Pattern (`PAT` function)
- Pseudo Random-Bit Generator Source (`PRBS` function)

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

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

Trapezoidal Pulse Source

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

```
Vxxx n+ n- PU[LSE] [(]v1 v2 [td [tr [tf [pw [per]]]]] [)]
+ [PERJITTER=val [SEED=val]]

Ixxx n+ n- PU[LSE] [(]v1 v2 [td [tr [tf [pw [per]]]]] [)]
+ [PERJITTER=val [SEED=val]]
```

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 voltage or current before the pulse onset (units: volts/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
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.
perjitter	RMS value for period jitter, adjusts the magnitude of the random time.
seed	Used to generate random number sequences with different seed value. The value is a negative integer, defaults to -1.

Table 11 Time-Value Relationship for a PULSE Source

Time	Value
0	v1

Table 11 Time-Value Relationship for a PULSE Source (Continued)

Time	Value
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.

Effect of Jitter on the PULSE Source

The effect of jitter on the PULSE source results in random shifts of the rise and fall transitions that normally take place at:

RISE edge: $td + n \cdot T_0 \le t \le td + n \cdot T_0$

FALL edge: $td + pw + n \cdot T_0 \le t \le td + tr + pw + tf + n \cdot T_0$

The jitter effect is equivalent to introducing random shifts in the period $0T$ consistent with the 1st order jitter model based on Period Jitter, also according to the expression for period variations $T_0 \rightarrow T_j = T_0 + \Delta T(t)$. The first period of the PULSE source is also guaranteed to be that specified in the syntax, yet all subsequent pulse periods are random according to $T_0 + \Delta T(t)$.

The PULSE source with jitter will still maintain constant rise and fall times. This creates some uncertainty in how the pulse width (pw) varies as the period varies due to jitter.

HSPICE RF uses a special calculation that holds the rise and fall times (tr and tf) constant, and also holds the *50% Duty Cycle* constant. Note that the 50% duty cycle is defined based on the halfway points on the rise and fall times. The result is that the pulse width (pw) change due to jitter variations is given by:

$$pw_j = pw \cdot \left(\frac{T_j}{T_0}\right) + \frac{1}{2} \cdot (tr + tf) \cdot \left(\frac{T_j}{T_0} - 1\right)$$

Also, this sets the minimum period for a PULSE source with jitter to be $tr + tf$, resulting in the extreme case of a sawtooth waveform.

A Gaussian random number generator is used to compute the random $\Delta T(t)$ variations after each leading edge of the clock sources. For flexibility, the `SEED` parameter (integer) is supported for generating different random number sequences when different `SEED` integers are used for initialization.

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.
- The RMS value for period jitter is 10ns.
- The seed is -1.

```
    VIN 3 0 PULSE (-1 1 2NS 2NS 2NS 50NS 100NS) perjitter=10ns
seed=-1
```

Example 2

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

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

Example 3

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

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

- The pulse width is 20 nanoseconds.
- The pulse period is 50 nanoseconds.

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

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

Figure 18 shows the result of simulating this netlist, in HSPICE or HSPICE RF.

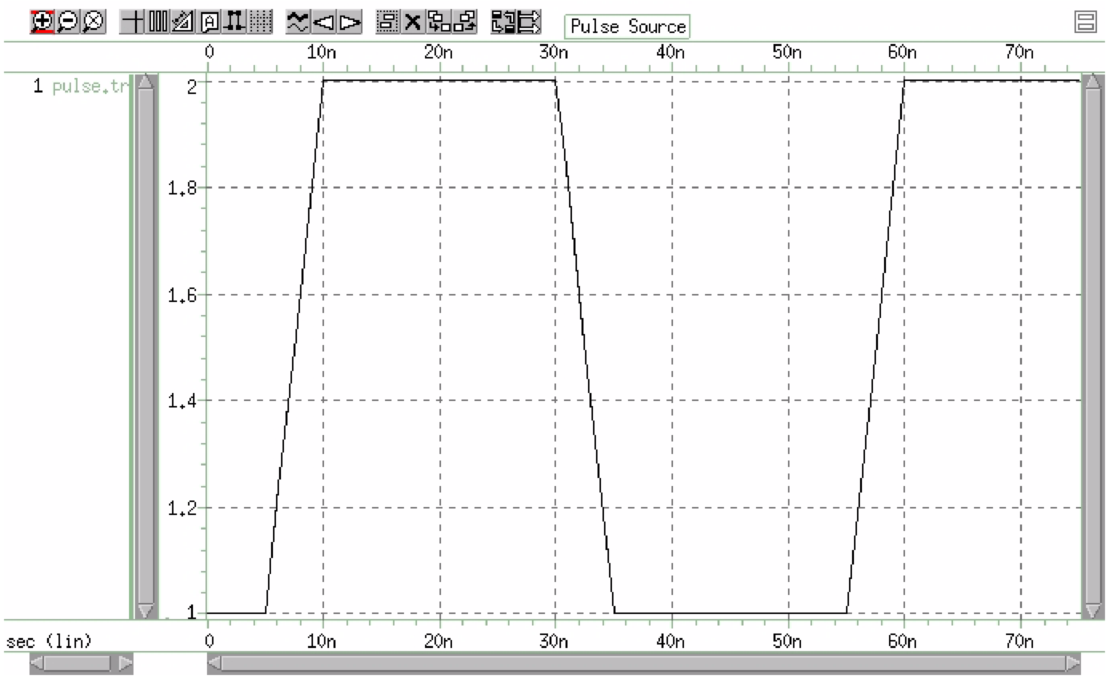


Figure 18 Pulse Source Function

Sinusoidal Source Function

HSPICE or HSPICE RF provides a damped sinusoidal source function, 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

```
Vxxx n+ n- SIN [(] vo va [freq [td [q [j]]]] [)]
+ [[PERJITTER=val] [SEED=val]]

Ixxx n+ n- SIN [(] vo va [freq [td [q [j]]]] [)]
+ [[PERJITTER=val] [SEED=val]]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits the sinusoidal response.
SIN	Keyword for a sinusoidal time-varying source.
vo	Voltage or current offset, in volts or amps.
va	Voltage or current peak value (vpeak), in volts or amps.
freq	Source frequency in Hz. Default=1/TSTOP.
td	Time (propagation) delay before beginning the sinusoidal variation, in seconds. Default=0.0. Response is 0 volts or amps, until HSPICE or HSPICE RF reaches the delay value, even with a non-zero DC voltage.
q	Damping factor, in units of 1/seconds. Default=0.0.
j	Phase delay, in units of degrees. Default=0.0.
perjitter	RMS value for period jitter, used to adjust the magnitude of the random time.
seed	Used to generate random number sequences with different seed value. The value is a negative integer, defaults to -1.

The following table of expressions defines the waveform shape:

Table 12 Waveform Shape Expressions

Time	Value
0 to td	$vo + va \cdot SIN\left(\frac{2 \cdot \Pi \cdot \varphi}{360}\right)$
td to tstop	$vo + va \cdot Exp[-(Time - td) \cdot q] \cdot SIN\left\{2 \cdot \Pi \cdot \left[freq \cdot (time - td + x(t)) + \frac{j}{360}\right]\right\}$
	Where q and j are the damping factor and phase delay in the syntax.

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 19 on page 195 for a plot of the source output.

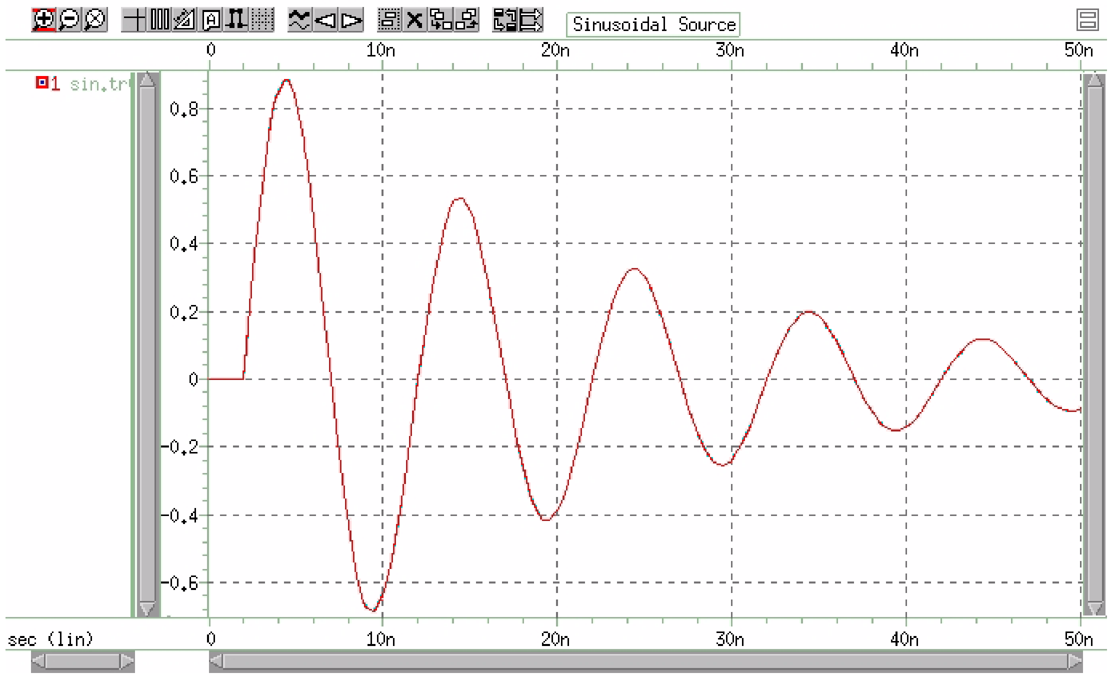


Figure 19 Sinusoidal Source Function

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

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

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

Table 13 SIN Voltage Source

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

Table 13 SIN Voltage Source

Parameter	Value
damping factor	50 MHz

Exponential Source Function

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

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

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

Parameter	Description
Vxxx, Ixxx	Independent voltage source, exhibiting an exponential response.
EXP	Keyword for an exponential time-varying source.
v1	Initial value of voltage or current, in volts or amps.
v2	Pulsed value of voltage or current, in volts or amps.
td1	Rise delay time, in seconds. Default=0.0.
td2	Fall delay time, in seconds. Default=td1+TSTEP.
t1	Rise time constant, in seconds. Default=TSTEP.
t2	Fall time constant, in seconds. Default=TSTEP.

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

The following table of expressions defines the waveform shape:

Table 14 Waveform Shape Definitions

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

Example

```
VIN 3 0 EXP (-4 -1 2NS 30NS 60NS 40NS)
```

The above example describes an exponential transient source, which connects between nodes 3 and 0. In this source:

- Initial t=0 voltage is -4 V.
- Final voltage is -1 V.
- Waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns.
- At 60 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

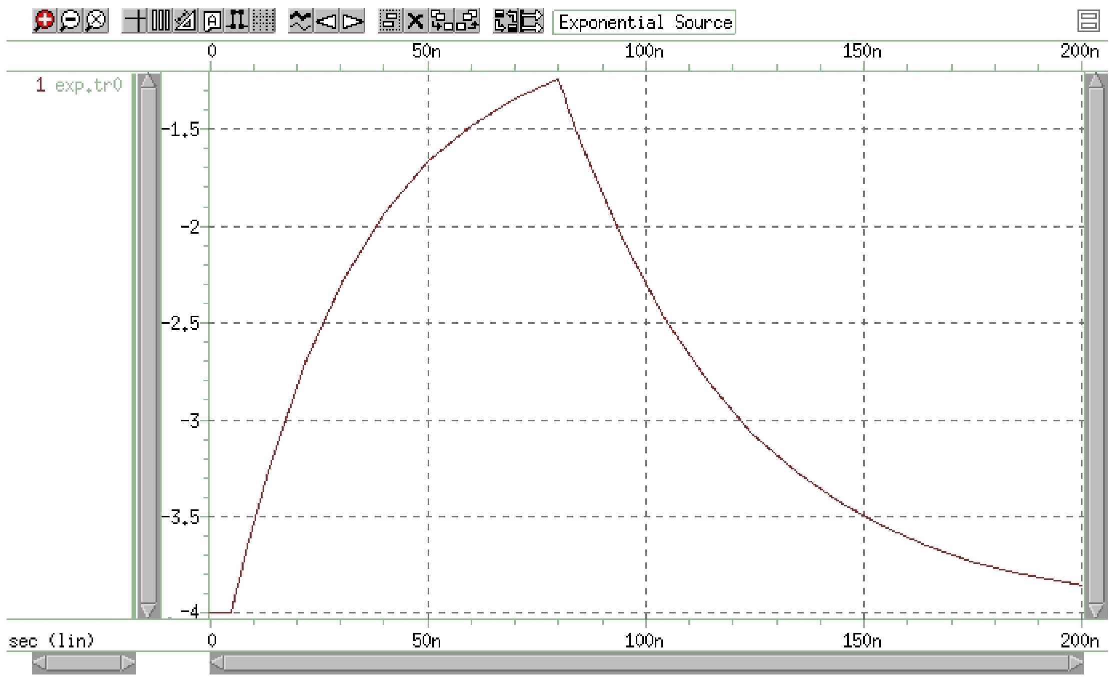


Figure 20 Exponential Source Function

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

```
*file: exp.spexponential independant source
.options post
.param v0=-4 va=-1 td1=5n tau1=30n tau2=40n td2=80n
v 1 0 exp(v0 va td1 tau1 td2 tau2)
r 1 0 1
.tran .05n 200n
.end
```

This example shows an entire netlist, which contains an `EXP` voltage source. In this source:

- Initial t=0 voltage is -4 V.
- Final voltage is -1 V.
- Waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns.
- At 80 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

Piecewise Linear Source

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

General Form

```
Vxxx n+ n- PWL <(> t1 v1 <t2 v2 t3 v3…> <R <=repeat>>
+ <TD=delay> <)>

Ixxx n+ n- PWL <(> t1 v1 <t2 v2 t3 v3…> <R <=repeat>>
+ <TD=delay> <)>
```

MSINC and ASPEC Form

```
Vxxx n+ n- PL <(> v1 t1 <v2 t2 v3 t3…> <R <=repeat>>
+ <TD=delay> <)>

Ixxx n+ n- PL <(> v1 t1 <v2 t2 v3 t3…> <R <=repeat>>
+ <TD=delay> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source; uses a piecewise linear response.
PWL	Keyword for a piecewise linear time-varying source.
v1 v2 … vn	Current or voltage values at the corresponding timepoint.
t1 t2 … tn	Timepoint values, where the corresponding current or voltage value is valid.
R=repeat	Keyword and time value to specify a repeating function. With no argument, the source repeats from the beginning of the function. *repeat* is the time, in units of seconds, which specifies the start point of the waveform to repeat. This time needs to be less than the greatest time point, *tn*.
TD=delay	Time, in units of seconds, which specifies the length of time to delay (propagation delay) the piecewise linear function.

- Each pair of values (*t1*, *v1*) specifies that the value of the source is *v1* (in volts or amps), at time *t1*.
- Linear interpolation between the time points determines the value of the source, at intermediate values of time.
- The PL form of the function accommodates ASPEC style formats, and reverses the order of the time-voltage pairs to voltage-time pairs.
- If you do not specify a time-zero point, HSPICE or HSPICE RF uses the DC value of the source, as the time-zero source value.

HSPICE or HSPICE RF does not force the source to terminate at the `TSTOP` value, specified in the `.TRAN` statement.

If the slope of the piecewise linear function changes below a specified tolerance, the timestep algorithm might not choose the specified time points as simulation time points. To obtain a value for the source voltage or current, HSPICE or HSPICE RF extrapolates neighboring values. As a result, the simulated voltage might deviate slightly from the voltage specified in the `PWL` list. To force HSPICE or HSPICE RF to use the specified values, use `.OPTION SLOPETOL`, which reduces the slope change tolerance.

R causes the function to repeat. You can specify a value after this *R*, to indicate the beginning of the function to repeat. The repeat time must equal a breakpoint in the function. For example, if *t1*=1, *t2*=2, *t3*=3, and *t4*=4, then the repeat value can be 1, 2, or 3.

Specify TD=*val* to cause a delay at the beginning of the function. You can use `TD` with or without the repeat function.

Example

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

```
file pwl.sp repeated piecewise linear source
.option post
.tran 5n 500n
v1 1 0 pwl 60n 0v, 120n 0v, 130n 5v, 170n 5v, 180n 0v, r
r1 1 0 1

v2 2 0 pl 0v 60n, 0v 120n, 5v 130n, 5v 170n, 0v 180n, r 60n
r2 2 0 1
.end
```

This example shows an entire netlist, which contains two piecewise linear voltage sources. The two sources have the same function:

- First is in normal format. The repeat starts at the beginning of the function.
- Second is in ASPEC format. The repeat starts at the first timepoint.

See Figure 21 for the difference in responses.

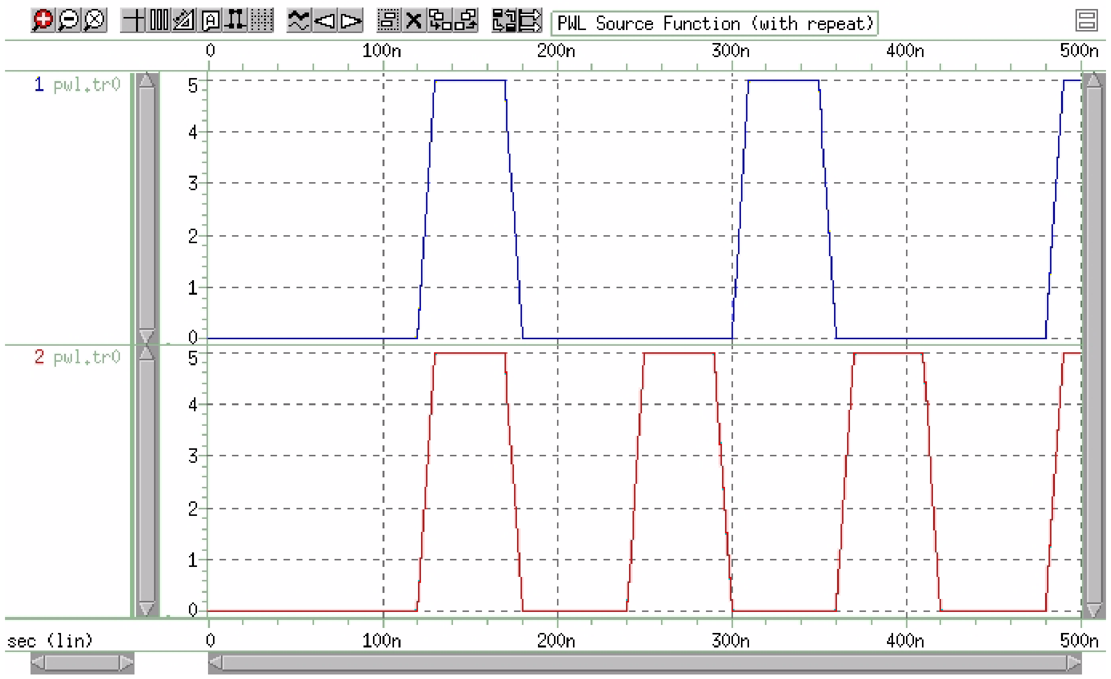


Figure 21 Results of Using the Repeat Function

Data-Driven Piecewise Linear Source

HSPICE provides a data-driven piecewise linear source function in an independent voltage or current source.

```
Vxxx n+ n- PWL (TIME, PV)
Ixxx n+ n- PWL (TIME, PV)
.DATA dataname
TIME PV
t1 v1
t2 v2
t3 v3
t4 v4
. . . .
.ENDDATA
```

```
.TRAN DATA=datanam
```

Parameter	Description
TIME	Parameter name for time value, provided in a .DATA statement.
PV	Parameter name for amplitude value, provided in a .DATA statement.

Use 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. This source enables simulation results reuse as an input source in another simulation. Transient analysis must be data-driven.

Example

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

```
*DATA DRIVEN PIECEWISE LINEAR SOURCE
.options list node post
V1 1 0 PWL(TIME, pv1)
R1 1 0 1
V2 2 0 PWL(TIME, pv2)
R2 2 0 1
.DATA dsrc
TIME pv1 pv2
0n 5v 0v
5n 0v 5v
10n 0v 5v
.ENDDATA
.TRAN DATA=dsrc
.print v(1) v(2)
.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; there should be no time in `.TRAN` because time has been included in `DATA`.

Single-Frequency FM Source

HSPICE or HSPICE RF provides a single-frequency FM source function, in an independent voltage or current source.

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

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

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the frequency-modulated response.
SFFM	Keyword for a single-frequency, frequency-modulated, time-varying source.
vo	Output voltage or current offset, in volts or amps.
va	Output voltage or current amplitude, in volts or amps.
fc	Carrier frequency, in Hz. Default=1/TSTOP.
mdi	Modulation index, which determines the magnitude of deviation from the carrier frequency. Values normally lie between 1 and 10. Default=0.0.
fs	Signal frequency, in Hz. Default=1/TSTOP.

The following expression defines the waveform shape:

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

Example

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

```
*file: sffm.spfrequency modulation source
.options post
vsff1 15 0 dc 3v sffm(0v 1v 20k 10 5k)
rssf1 15 0 1
.tran .001ms .5ms
.probe tran v(15)
.end
```

This example shows an entire netlist, which contains a single-frequency, frequency-modulated voltage source. In this source.

- The offset voltage is 0 volts.
- The maximum voltage is 1 millivolt.

- The carrier frequency is 20 kHz.
- The signal is 5 kHz, with a modulation index of 10 (the maximum wavelength is roughly 10 times as long as the minimum).

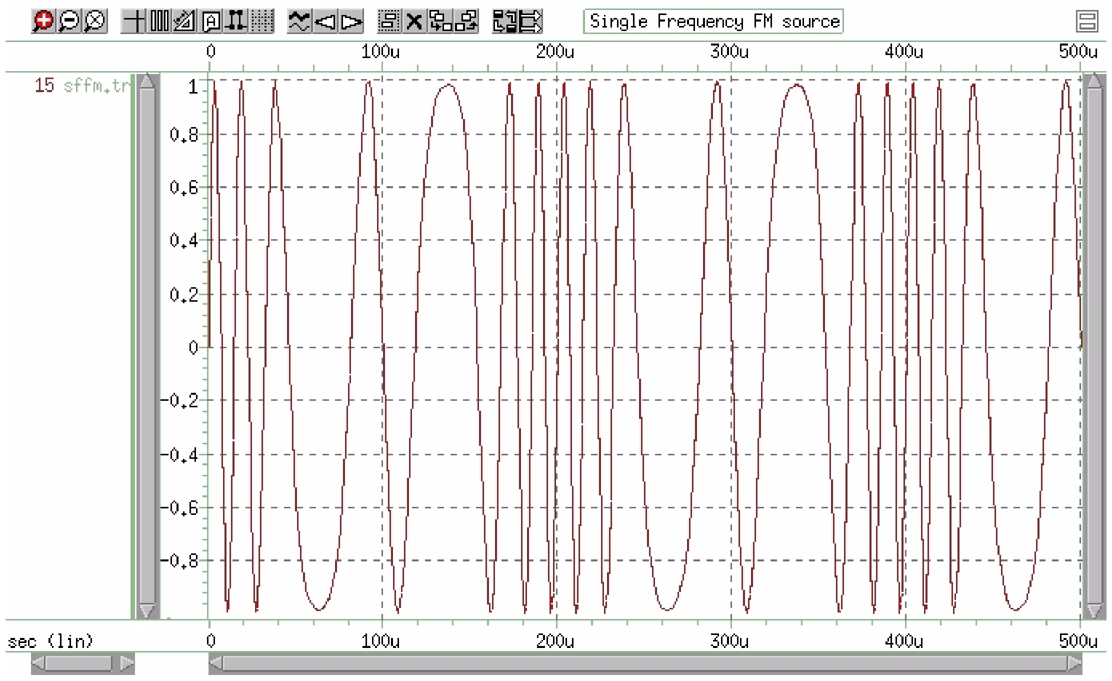


Figure 22 Single Frequency FM Source

Single-Frequency AM Source

HSPICE or HSPICE RF provides a single-frequency AM source function in an independent voltage or current source.

```
Vxxx n+ n- AM < (> sa oc fm fc <td> <)>
Ixxx n+ n- AM < (> sa oc fm fc <td> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the amplitude-modulated response.
AM	Keyword for an amplitude-modulated, time-varying source.

sa	Signal amplitude, in volts or amps. Default=0.0.
fc	Carrier frequency, in hertz. Default=0.0.
fm	Modulation frequency, in hertz. Default=1/TSTOP.
oc	Offset constant, a unitless constant that determines the absolute magnitude of the modulation. Default=0.0.
td	Delay time (propagation delay) before the start of the signal, in seconds. Default=0.0.

The following expression defines the waveform shape:

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

Example

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

```
*file amsrc.sp amplitude modulation
.option post
.tran .01m 20m

v1 1 0 am(10 1 100 1k 1m)
r1 1 0 1

v2 2 0 am(2.5 4 100 1k 1m)
r2 2 0 1

v3 3 0 am(10 1 1k 100 1m)
r3 3 0 1
.end
```

This example shows an entire netlist, which contains three amplitude-modulated voltage sources.

- In the first source:
  - Amplitude is 10.
  - Offset constant is 1.
  - Carrier frequency is 1 kHz.
  - Modulation frequency of 100 Hz.
  - Delay is 1 millisecond.

- In the second source, only the amplitude and offset constant differ from the first source:
    - Amplitude is 2.5.
    - Offset constant is 4.
    - Carrier frequency is 1 kHz.
    - Modulation frequency of 100 Hz.
    - Delay is 1 millisecond.
- The third source exchanges the carrier and modulation frequencies, compared to the first source:
    - Amplitude is 10.
    - Offset constant is 1.
    - Carrier frequency is 100 Hz.
    - Modulation frequency of 1 kHz.
    - Delay is 1 millisecond.

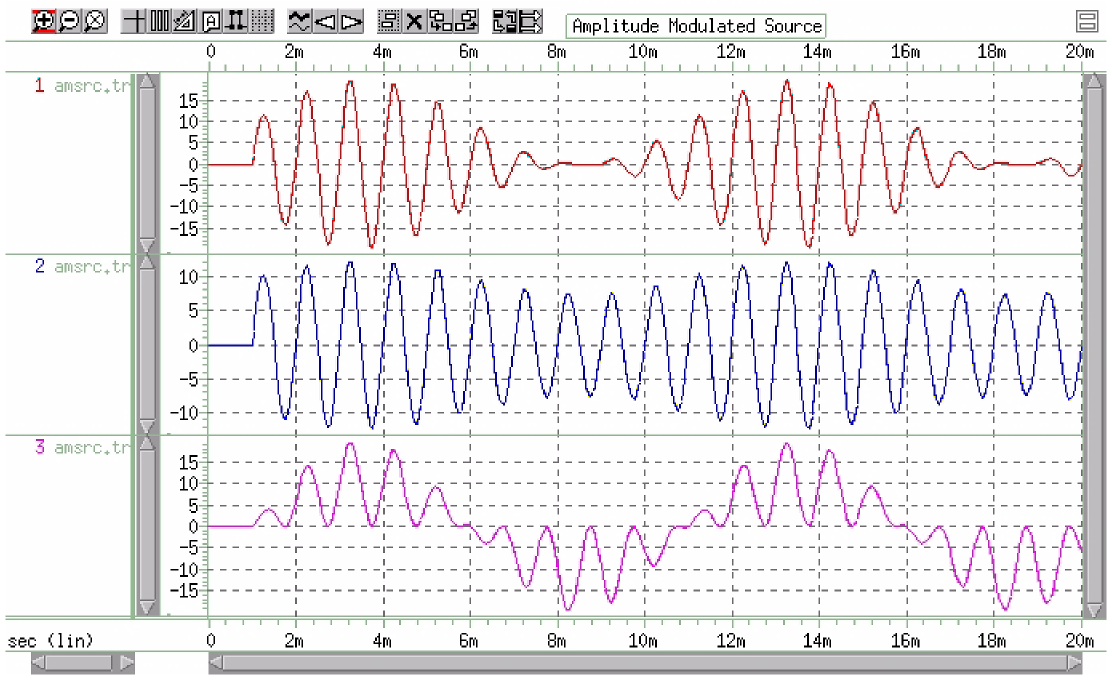


Figure 23 Amplitude Modulation Plot

Pattern Source

HSPICE or HSPICE RF provides a pattern source function, in an independent voltage or current source. The pattern source function uses four states, '1','0','m', and 'z', which represent the high, low, middle voltage, or current and high impedance state respectively. The series of these four states is called a “b-string.”

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf  tsample data <RB=val>
+ <R=repeat> <)>

Ixxx n+ n- PAT <(> vhi vlo td tr tf  tsample data <RB=val>
+ <R=repeat> <)>
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits a pattern response.
PAT	Keyword for a pattern time-varying source.
vhi	High voltage or current value for pattern sources (units of volts or amps).
vlo	Low voltage or current value for pattern sources (units of volts or amps).
td	Delay (propagation) time in seconds from the beginning of the transient interval to the first onset ramp. It can be negative. The state in the delay time is the same as the first state specified in data.
tr	Duration of the onset ramp (in seconds) from the low value to the high value (reverse transit time).
tf	Duration of the recovery ramp (in seconds) from the high value back to the low value (forward transit time).
tsample	Time spent at '0' or '1' or 'M' or 'Z' pattern value (in seconds).

Parameter	Description
data	String of '1' ,'0','M', 'Z' representing a pattern source. The first alphabet must be 'B', which represents it is a binary bit stream. This series is called b-string. '1' represents the high voltage or current value, '0' is the low voltage or current value, 'M' represents the value which is equal to 0.5*(vhi+vlo).'Z' represents the high impedance state (only for voltage source).
RB	Keyword to specify the starting bit when repeating. The repeat data starts from the bit indicated by RB. RB must be an integer. If the value is larger than the length of the b-string, an error is reported. If the value is less than 1, it is set to 1 automatically.
R=repeat	Keyword to specify how many times to execute the repeating operation be executed. With no argument, the source repeats from the beginning of the b-string. If R=-1, it means the repeating operation will continue forever. R must be an integer and if it is less than -1, it will be set to 0 automatically.

The time from 0 to the first transition is:

```
tdelay+N*tsample-tr(tf)/2
```

- `N` is the number of the same bit from the beginning.
- If the first transition is rising, this equation uses `tr`.
- If the first transition is falling, it uses `tf`.

Example

The following example shows a pattern source with two b-strings:

```
*FILE: pattern source gereral form
v1 1 0 pat (5 0 0n 1n 1n 5n b1011 r=1 rb=2 b0m1z)
r1 1 0 1
```

In this pattern:

- High voltage is 5 v
- Low voltage is 0 v
- Time delay is 0 n
- Rise time is 1 n
- Fall time is 1 n
- Sample time is 5 n

The first b-string is `1011`, which repeats once and then repeats from the second bit, which is `0`. The second b-string is `0m1z`. Since neither `R` and `RB` is specified here, they are set to the default value, which is `R=0`, `RB=1`.

Example

The following b-string and its repeat time R and repeating start bit `RB` cannot use a parameter—it is considered as a undivided unit in HSPICE and can only be defined in a `.PAT` command.

```
*FILE:pattern source using parameter
.param td=40ps tr=20ps tf=80ps tsample=400ps
VIN 1 0 PAT (2 0 td tr tf tsample b1010110 r=2)
r1 1 0 1
```

In this pattern:

- High voltage is 2 V.
- Low voltage is 0 V.
- Time delay is 40 ps.
- Rise time is 20 ps.
- Fall time is 80 ps.
- Sample time is 400 ps.
- Data is 1010110.

Nested-Structure Pattern Source HSPICE provides Nested Structure (NS) for the pattern source function to construct complex waveforms. NS is a combination of a b-string and other nested structures defined in a `.PAT` command, which is explained later in this section.

The following general syntax is for an NS pattern source.

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf  tsample
+ [component 1 ... component n] <RB=val> <R=repeat> <)>
Ixxx n+ n- PAT <(> vhi vlo td tr tf  tsample
+ [component 1 ... component n] <RB=val> <R=repeat> <) >
```

Parameter	Description
component	Component is the element that makes up NS, which can be a b-string or a patname defined in other `PAT` commands. Brackets ([ ]) must be used.

Parameter	Description
RB=val	Keyword to specify the starting component when repeating. The repeat data starts from the component indicated by RB. RB must be an integer. If RB is larger than the length of the NS, an error is reported. If RB is less than 1, it is automatically set to 1.
R=repeat	Keyword to specify how many times the repeating operation is executed. With no argument, the source repeats from the beginning of the NS. If R=-1, the repeating operation continues indefinitely. R must be an integer, and if it is less than -1, it is automatically set to 0.

If the component is a b-string, it can also be followed by `R=repeat` and `RB=val` to specify the repeat time and repeating start bit.

Example

```
*FILE: Pattern source using nested structure
v1 1 0 pat (5 0 0n 1n 1n 5n [b1011 r=1 rb=2 b0m1z] r=2 rb=2)
r1 1 0 1
```

When expanding the nested structure, you get the pattern source like this:

```
'b1011 r=1 rb=2 b0m1z b0m1z b0m1z'
```

The whole `NS` repeats twice, and each time it repeats from the second `b0m1z` component.

Pattern-Command Driven Pattern Source The following general syntax is for including a pattern-command driven pattern source in an independent voltage or current source. The `RB` and `R` of a b-string or `NS` can be reset in an independent source. With no argument, the `R` and `RB` are the same when defined in the pattern command.

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf  tsample PatName <RB=val>
+ <R=repeat> <)>
Ixxx n+ n- PAT <(> vhi vlo td tr tf  tsample Patname <RB=val>
+ <R=repeat> <)>
```

Additional syntax applies to the `.PAT`-command driven pattern source:

```
.PAT <PatName>=data <RB=val> <R=repeat>
.PAT <patName>=[component 1 ... component n] <RB=val>
   <R=repeat>
```

The PatName is the pattern name that has an associated b-string or nested structure.

Example 1

```
v1 1 0 pat (5 0 0n 1n 1n 5n a1 a2 r=2 rb=2)
.PAT a1=b1010 r=1 rb=1
.PAT a2=b0101 r=1 rb=1
```

The final pattern source is:

```
b1010 r=1 rb=1 b0101 r=2 rb=2
```

When the independent source uses the pattern command to specify its pattern source, `r` and `rb` can be reset.

Example 2

```
*FILE 2: Pattern source driven by pattern command
v1 1 0 pat (5 0 0n 1n 1n 5n [a1 b0011] r=1 rb=1)
.PAT a1=[b1010 b0101] r=0 rb=1
```

The final pattern source is:

```
b1010 b0101 b0011 b1010 b0101 b0011
```

The `a1` is a predefined NS, and it can be referenced by pattern source.

Pseudo Random-Bit Generator Source

HSPICE or HSPICE RF Pseudo Random Bit Generator Source (PRBS) function, in an independent voltage or current source. This function can be used in several applications from cryptography and bit-error-rate measurement, to wireless communication systems employing spread spectrum or CDMA techniques. In general, PRBS uses a Linear Feedback Shift Register (LFSR) to generate a pseudo random bit sequence.

```
Vxxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate seed <[>
+ taps <]> <rout=val> <)>

Ixxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate seed <[>
+ taps <]> <rout=val> <)>
```

Parameter	Description
LFSR	Specifies the voltage or current source as PRBS.

Parameter	Description
vlow	The minimum voltage or current level.
vhigh	The maximum voltage or current level.
tdelay	Specifies the initial time delay to the first transition.
trise	Specifies the duration of the onset ramp (in seconds) from the initial value to the pulse plateau value (reverse transit time).
tfall	Specifies the duration of the recovery ramp (in seconds) from the pulse plateau back to the initial value (forward transit time).
rate	The bit rate.
seed	The initial value loaded into the shift register.
taps	The bits used to generate feedback.
rout	The output resistance.

Example 1

The following example shows the pattern source that is connected between node in and node gnd:

```
vin in gnd LFSR (0 1 1m 1n 1n 10meg 1 [5, 2] rout=10)
```

Where,

- The output low voltage is `0` , and the output high voltage is `1` v.
- The delay time is `1` ms.
- The rise and fall times are each `1` ns.
- The bit rate is `10meg` bits/s.
- The seed is `1`.
- The taps are `[5, 2]`.
- The output resistance is `10` ohm.
- The output from the LFSR is: 1000010101110110001111100110100...

Example 2

The following example shows the pattern source connected between node 1 and node 0:

```
.PARAM td1=2.5m tr1=2n
vin 1 0 LFSR (2 4 td1 tr1 1n 6meg 2 [10, 5, 3, 2])
```

Where,

- The output low voltage is `2` v, and the output high voltage is `4` v.
- The delay is `2.5` ms.
- The rise time is `2` ns, and the fall time is `1` ns.
- The bit rate is `6meg` bits/s.
- The seed is `2`.
- The taps are `[10, 5, 3, 2]`.
- The output resistance is `0` ohm.

Example 3

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

```
* prbs.sp
.OPTION POST
.TRAN 0.5n 50u
V1 1 0 LFSR (0 1 1u 1n 1n 10meg 1 [5, 2] rout=10)
R1 1 0 1
.END
```

Example 4

To generate a PRBS source that includes jitter, use the following steps:

1. Construct your usual linear feedback shift register (LFSR) generator.
2. Construct a matching (T,tr,tf) PULSE source as a clock, but add jitter to it with the PERJITTER keyword.
3. Use the PULSE source to gate (buffer) the LFSR output (through an ideal AND gate, VCCS, or similar function).

Linear Feedback Shift Register

A LFSR consists of several simple-shift registers in which a binary-weighted modulo-2 sum of the taps is fed back to the input. The modulo-2 sum of two1-bit binary numbers yields 0 if the two numbers are identical and 1 if the differ is 0+0=0, 0+1=1, or 1+1=0.

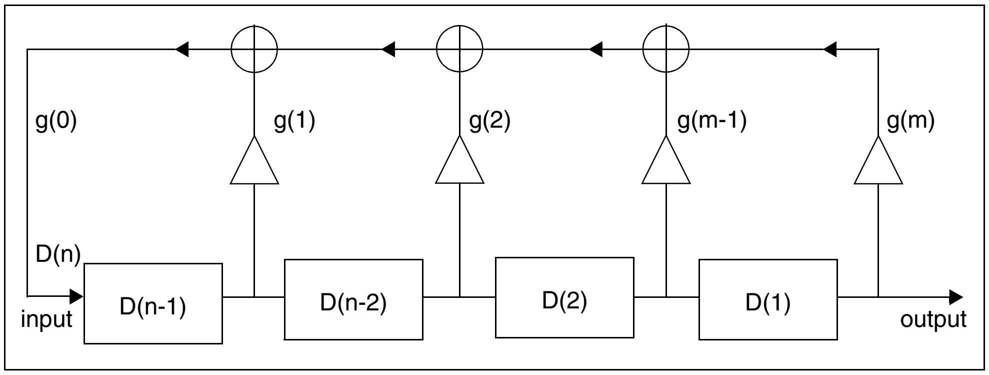


Figure 24 LFSR Diagram

For any given tap, the weight “gi” is either 0, (meaning “no connection”), or 1, (meaning it is fed back). Two exceptions are g0 and gm, which are always 1 and therefore always connected. The gm is not really a feedback connection, but rather an input of the shift register that is assigned a feedback weight for mathematical purposes.

The maximum number of bits is defined by the first number in your TAPS definition. For example [23, 22, 21, 20, 19, 7] denotes a 23 stage LFSR. The TAPS definition is a specific feedback tap sequence that generates an M-Sequence PRB. The LFSR stages limit is between 2 and 30. The seed cannot be set to zero; HSPICE reports an error and exits the simulation if you set the seed to zero.

Conventions for Feedback Tap Specification

A given set of feedback connections can be expressed in a convenient and easy-to-use shorthand form with the connection numbers listed within a pair of brackets. The g0 connection is implied and not listed since it is always connected. Although gm is also always connected, it is listed in order to convey the shift register size (number of registers).

The following line is a set of feedback taps where j is the total number of feedback taps (not including g0), f(1)=m is the highest-order feedback tap (and the size of the LFSR), and f(j) are the remaining feedback taps:

```
[f(1), f(2), f(3), ..., f(j)]
```

Example

The following line shows that the number of registers is 7 and the total number of feedback taps is 4:

```
[7, 3, 2, 1]
```

The following feedback input applies for this specification:

```
D(n)=[D(n-7)+D(n-3)+D(n-2)+D(n-1)] mod 2
```

Voltage and Current Controlled Elements

HSPICE or HSPICE RF provides two voltage-controlled and two current-controlled elements, known as E, G, H, and F Elements. You can use these controlled elements to model:

- MOS transistors
- bipolar transistors
- tunnel diodes
- SCRs
- analog functions, such as:
  - operational amplifiers
  - summers
  - comparators
  - voltage-controlled oscillators
  - modulators
  - switched capacitor circuits

Depending on whether you used the polynomial or piecewise linear functions, the controlled elements can be:

- Linear functions of controlling-node voltages.
- Non-linear functions of controlling-node voltages.
- Linear functions of branch currents.
- Non-linear functions of branch currents.

The functions of the E, F, G, and H controlled elements are different.

- The E-element can be:
  - A voltage-controlled voltage source
  - A behavioral voltage source

- An ideal op-amp.
- An ideal transformer.
- An ideal delay element.
- A piecewise linear, voltage-controlled, multi-input AND, NAND, OR, or NOR gate.

- The F-element can be:
  - A current-controlled current source.
  - An ideal delay element.
  - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.
- The G-element can be:
  - A voltage-controlled current source.
  - A behavioral current source.
  - A voltage-controlled resistor.
  - A piecewise linear, voltage-controlled capacitor.
  - An ideal delay element.
  - A piecewise linear, multi-input AND, NAND, OR, or NOR gate.
- The H-element can be:
  - A current-controlled voltage source.
  - An ideal delay element.
  - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.

The next section describes polynomial and piecewise linear functions. Later sections describe element statements for linear or nonlinear functions. For detailed PWL examples, see PWL/DATA/VEC Converter on page 338.

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 several polynomial equations, using the `POLY(NDIM)` parameter in the E, F, G, or H-element statement.

Syntax can be either `POLY=`*`INTEGER_NUMBER`* or `POLY(`*`INTEGER_NUMBER`*`)`

For example, either of the following are legitimate statements for an E-element instance with the `POLY` function:

```
E1 e1 0 POLY=2 e11 0 e12 0  1 2 3
```

or

```
E1 e1 0 POLY(2) e11 0 e12 0  1 2 3
```

Polynomial values can be:

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).
POLY(n)	Multi-dimensional equation (function of n controlling variables).

Each polynomial equation includes polynomial coefficient parameters (P0, P1 … Pn), which you can set to explicitly define the equation.

One-Dimensional Function

If the function is one-dimensional (a function of one branch current or node voltage), the following expression determines the `FV` function value:

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

Parameter	Description
FV	Controlled voltage or current from the controlled source.
P0. . .PN	Coefficients of a polynomial equation.
FA	Controlling branch current, or nodal voltage.

Note:

If you specify one coefficient in a one-dimensional polynomial, HSPICE or HSPICE RF assumes that the coefficient is P1 (P0=0.0). Use this as input for linear controlled sources.

The following controlled source statement is a one-dimensional function. This voltage-controlled voltage source connects to nodes 5 and 0.

```
E1 5 0 POLY(1) 3 2 1 2.5
```

In the above source statement, the single-dimension polynomial function parameter, `POLY(1)`, informs HSPICE or HSPICE RF that E1 is a function of the difference of one nodal voltage pair. In this example, the voltage difference is between nodes 3 and 2, so `FA=V(3,2)`.

The dependent source statement then specifies that P0=1 and P1=2.5. From the one-dimensional polynomial equation above, the defining equation for V(5,0) is:

$$V(5,\ 0)\ =\ 1 + 2.5 \cdot\ V(3{,}2)$$

You can also express V(5,0) as *E1*:

$$E1\ =\ 1 + 2.5 \cdot\ V(3{,}2)$$

Two-Dimensional Function

If the function is two-dimensional (that is, a function of two node voltages or two branch currents), the following expression determines `FV`:

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

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:

```
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.

N-Dimensional Function

An N-dimensional polynomial function can be expressed as:

$$FV = p_0 + \sum_{j-1}^{k}(p_{ij}Fx_1 + p_{2j}Fx_2 + \_\_\_\_ + p_{ij}Fx_k)^j$$

where, $Fx_1, Fx_2, \ldots\ldots F_k$,represent the *k* independent controlling branch current, or nodal voltage, and $p_{ij}, i = 1, 2, \ldots.k = 1, 2, \ldots.n$ are the coefficients.

Piecewise Linear Function

You can use the one-dimensional piecewise linear (`PWL`) function to model special element characteristics, such as those of:

- tunnel diodes
- silicon-controlled rectifiers
- diode breakdown regions

To describe the piecewise linear function, specify measured data points. Although data points describe the device characteristic, HSPICE or HSPICE RF automatically smooths the corners, to ensure derivative continuity. This, in turn, results in better convergence.

The `DELTA` parameter controls the curvature of the characteristic at the corners. The smaller the `DELTA`, the sharper the corners are. The maximum `DELTA` is limited to half of the smallest breakpoint distance. If the breakpoints are sufficiently separated, specify the `DELTA` to a proper value.

- You can specify up to 100 point pairs.
- You must specify at least two point pairs (each point consists of an *x* and a *y* coefficient).

To model bidirectional switch or transfer gates, G-elements use the `NPWL` and `PPWL` functions, which behave the same way as NMOS and PMOS transistors.

You can also use the piecewise linear function to model multi-input AND, NAND,OR, and NOR gates. In this usage, only one input determines the state of the output.

- In AND and NAND gates, the input with the smallest value determines the corresponding output of the gates.
- In OR and NOR gates, the input with the largest value determines the corresponding output of the gates.

Power Sources

This section describes independent sources and controlled sources.

Independent Sources

A power source is a special kind of voltage or current source, which supplies the network with a pre-defined power that varies by time or frequency. The source produces a specific input impedance.

To apply a power source to a network, you can use either:

- A Norton-equivalent circuit (if you specify this circuit and a current source)—the I (current source) element, or
- A Thevenin-equivalent circuit (if you specify this circuit and a voltage source)—the V (voltage source) element.

As with other independent sources, simulation assumes that positive current flows from the positive node, through the source, to the negative node. A power source is a time-variant or frequency-dependent utility source; therefore, the value/phase can be a function of either time or frequency.

A power source is a subclass 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.

Using the Keyword POWER

If you use the `POWER`keyword in the netlist, then a simulation recognizes a current/voltage source as a power source:

```
Vxxx node+ node- power=<powerVal <powerFun>> imp=value1
+ imp_ac=value2,value3 powerFun=<FREQ <TIME>>(...)
Ixxx node+ node- power=<powerVal <powerFun>> imp=value1
```

```
+ imp_ac=value2,value3 powerFun=<FREQ <TIME>>(...)
```

Parameter	Description
powerVal	A constant power source supplies the available power. If you specify POWER_DB, then the value is in decibels; otherwise, it is in Watts*POWER_SCAL, where POWER_SCAL is a scaling factor that you specify in a SCALE option (default=1).
powerFun	This function name indicates the time-variant or frequency-variant power source. In this equation, *powerFun* defines the functional dependence on time or frequency. ■ If the function name for *powerFun* is FREQ, then it is a frequency power source: FREQ(freq1, val1, freq2, val2,...) ■ If the function name for *powerFun* is TIME, then it is a piece-wise time variant function: TIME(t1, val1, t2, val2...)
imp=	DC impedance value.
imp_ac=	Magnitude and phase offset (in degrees) of AC impedance.

Example 1

```
V11 10 20 power=5 imp=5K
```

This example applies a 5-decibel/unit power source to node 10 and node 20, in a Thevenin-equivalent manner. The impedance of this power source is 5k Ohms.

Example 2

```
Iname 1 0 power=20 imp=9MEG
```

This example applies a 20-decibel/unit power source to node 1 and to ground, in a Norton-equivalent manner. The source impedance is 9 mega-ohms.

Example 3

```
V5 6 0 power=FREQ(10HZ, 2, 10KHZ, 0.01) imp=2MEG imp_ac=(100K, 60)
V5 6 0 power=func1 imp=2MEG imp_ac=(100K, 60DEC)
+ func1=FREQ(10HZ, 2, 10KHZ, 0.01)
```

In the two preceding examples, a power source operates at two different frequencies, with two different values:

- At 10 Hz, the power value is 2 decibel/unit.
- At 10 kHz, the power value is 0.01 decibel/unit.

Also in these examples:

- The DC impedance is 2 mega-ohms.
- The AC impedance is 100 kilo-ohms.
- The phase offset is 60 degrees.

Power Calculation

The `POWER` keyword is used to calculate the total dissipated power of the circuit.

For example, if you have a top level circuit as follows:

```
********************************
vvdd vdd 0 5
vin in 0 pulse(0 5 0 1n 1n 4n 10n)
rout  out  0 10k
cout  out  0 cload
x1 in out inv
.subckt inv in out
          mn out in 0 0 nch W=10u L=1u
          mp out in vdd vdd pch W=10u L=1u
.ends
********************************
```

...you need to use `.PRINT TRAN POWER` to calculate the total circuit power.

The total circuit POWER will be `p(rout)+p(cout)+p(x1)`. This can also be verified in HSPICE with the following statement,

```
.print tran tot_pwr=par('(p(rout)+p(cout)+p(x1))')
```

Note that HSPICE does not add the power of each independent source (V & I).

Subcircuit Power Calculation

To print or probe an element or subcircuit power, use the variables, `P(element_name)` or `P(instance_name_of_subckt)`. For example, if you have the following subcircuit:

```
x1 in out inv
.subckt inv in out
       mn out in 0 0 nch W=10u L=1u
       mp out in vdd vdd pch W=10u L=1u
.ends
```

...you can use `.PRINT TRAN P(x1)` to calculate the subcircuit power. The subcircuit power of `P(x1)` will be the sum of `p(x1.mn)` and `p(x1.mp)`. This can be verified by using the following statement:

```
.print tran p_x1=par('p(x1.mn)+p(x1.mp)')
```

Measuring Leakage Power

When you probe `P(Instance_name)`, beginning with HSPICE version A-2007.12, HSPICE includes the gate tunneling current in the power function calculations. In prior releases, the power calculation function in HSPICE did not include gate tunneling induced power dissipation.

Troubleshooting Differences in Rise/Fall Power Input Signals

You may note a difference between the AVG power calculations for the rise time and fall time for input signals. There should not be a difference, but if you measure the AVG subcircuit power by using `P(Instance_name)` for an inverter subcircuit, for example, you may note that HSPICE excludes the energy stored in the output and includes the energy discharged from the capacitor for fall time.

If there is any difference in the measured results, then try running the simulation with the most accurate settings by setting `.OPTION RUNLVL=6`. In addition, you can set `.OPTION DELMAX` to the minimum rise or fall time of your circuit.

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.

```
Exxx n+ n- [VCCS|LEVEL=0] in+ in- ...
```

- `LEVEL=0` is an interchangeable function keyword, such as `VCAP` or `VCCS`, etc.
- `LEVEL=1` is an OpAmp.
- `LEVEL=2` is a Transformer.

See also Using G- and E-elements in Chapter 29, Modeling Filters and Networks.

Voltage-Controlled Voltage Source (VCVS)

Linear

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

For a description of these parameters, see E-element Parameters on page 233.

Polynomial (POLY)

```
Exxx n+ n- [VCVS] POLY(NDIM) in1+ in1- ...
+ inndim+ inndim-TC1=val [TC2=val] [SCALE=val]
+ [MAX=val] [MIN=val] [ABS=1] p0 p1… [IC=val]
```

In this syntax, *dim* (*dimensions*) ≤3. For a description of these parameters, see E-element Parameters on page 233. For a description of possible POLY syntaxes, see Polynomial Functions on page 216.

Piecewise Linear (PWL)

```
Exxx n+ n- [VCVS] PWL(1) in+ in- [DELTA=val]
+ [SCALE=val] [TC1=val] [TC2=val] x1,y1 x2,y2  ...
+ x100,y100 [IC=val]
```

For a description of these parameters, see E-element Parameters on page 233.

Multi-Input Gates

```
Exxx n+ n- [VCVS] gatetype(k) in1+ in1- ... inj+ inj-
+ [DELTA=val] [TC1=val] [TC2=val] [SCALE=val]
+ x1,y1  ...   x100,y100 [IC=val]
```

In this syntax, *gatetype(k)* can be AND, NAND, OR, or NOR gates. For a description of these parameters, see E-element Parameters on page 233.

Delay Element

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

For a description of these parameters, see E-element Parameters on page 233.

Laplace Transform

Voltage Gain H(s):

```
Exxx n+ n- LAPLACE in+ in-  k0, k1, ..., kn / d0, d1, ..., dm
+ [SCALE=val] [TC1=val] [TC2=val]
```

For a description of these parameters, see E-element Parameters on page 233.

Transconductance H(s):

```
Gxxx n+ n- LAPLACE in+ in-  k0, k1, ..., kn / d0, d1, ..., dm
+ [SCALE=val] [TC1=val] [TC2=val] [M=val]
```

H(s) is a rational function, in the following form:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

You can use parameters to define the values of all coefficients (k_0, k_1, ..., d_0, d_1, ...).

For a description of the G-element parameters, see G-element Parameters on page 250.

Example

```
Glowpass 0 out LAPLACE in 0   1.0 / 1.0  2.0  2.0  1.0
Ehipass out 0 LAPLACE in 0  0.0,0.0,0.0,1.0 / 1.0,2.0,2.0,1.0
```

The `Glowpass` element statement describes a third-order low-pass filter, with the transfer function:

$$H(s) = \frac{1}{1 + 2s + 2s^2 + s^3}$$

The `Ehipass` element statement describes a third-order high-pass filter, with the transfer function:

$$H(s) = \frac{s^3}{1 + 2s + 2s^2 + s^3}$$

Pole-Zero Function

Voltage Gain H(s):

```
Exxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm <SCALE=val> <TC1=val>
+ <TC2=val>
```

For a description of these parameters, see E-element Parameters on page 233.

Transconductance H(s):

```
Gxxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm <SCALE=val> <TC1=val>
+ <TC2=val> <M=val>
```

The following equation defines H(s) in terms of poles and zeros:

$$H(s) = \frac{a \cdot (s + \alpha_{z1} - j2\pi f_{z1})..(s + \alpha_{zn} - j2\pi f_{zn})(s + \alpha_{zn} + j2\pi f_{zn})}{b \cdot (s + \alpha_{p1} - j2\pi f_{p1})..(s + \alpha_{pm} - j2\pi f_{pm})(s + \alpha_{pm} + j2\pi f_{pm})}$$

The complex poles or zeros are in conjugate pairs. The element description specifies only one of them, and the program includes the conjugate. You can use parameters to specify the a, b, α, and f values.

For a description of the G-element parameters, see G-element Parameters on page 250.

Example

```
Ghigh_pass 0 out POLE in 0 1.0 0.0,0.0 / 1.0 0.001,0.0
Elow_pass out 0 POLE in 0 1.0 / 1.0, 1.0,0.0 0.5,0.1379
```

The `Ghigh_pass` statement describes a high-pass filter, with the transfer function:

$$H(s) = \frac{1.0 \cdot (s + 0.0 + j \cdot 0.0)}{1.0 \cdot (s + 0.001 + j \cdot 0.0)}$$

The `Elow_pass` statement describes a low-pass filter, with the transfer function:

$$H(s) = \frac{1.0}{1.0 \cdot (s + 1)(s + 0.5 + j2\pi \cdot 0.1379)(s + 0.5 - (j2\pi \cdot 0.1379))}$$

Frequency Response Table

Voltage Gain H(s):

```
Exxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val> <LEVEL=val> <ACCURACY=val>
```

For a description of these parameters, see E-element Parameters on page 233

Transconductance H(s):

```
Gxxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val> <M=val> <LEVEL=val> <ACCURACY=val>
```

Where,

- Each `fi` is a frequency point, in hertz.
- `ai` is the magnitude, in dB.
- `f1` is the phase, in degrees.

At each frequency, HSPICE or HSPICE RF uses interpolation to calculate the network response, magnitude, and phase. HSPICE or HSPICE RF interpolates the magnitude (in dB) logarithmically, as a function of frequency. It also interpolates the phase (in degrees) linearly, as a function of frequency.

$$|H(j2\pi f)| = \left(\frac{a_i - a_k}{\log f_i - \log f_k}\right)(\log f - \log f_i) + a_i$$

$$\angle H(j2\pi f) = \left(\frac{\phi_i - \phi_k}{f_i - f_k}\right)(f - f_i) + \phi_i$$

For a description of the G-element parameters, see G-element Parameters on page 250.

Example

```
Eftable output   0 FREQ input   0
+ 1.0k   -3.97m   293.7
+ 2.0k   -2.00m   211.0
+ 3.0k   17.80m   82.45
+  ...... ...
+ 10.0k -53.20    -1125.5
```

- The first column is frequency, in hertz.
- The second column is magnitude, in dB.
- The third column is phase, in degrees.

Set the `LEVEL` to `1` for a high-pass filter.

Set the last frequency point to the highest frequency response value that is a real number, with zero phase.

You can use parameters to set the frequency, magnitude, and phase, in the table.

Foster Pole-Residue Form

Gain E(s) form

```
Exxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1})/ (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2})/ (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3})/ (Re{p3}, Im{p3})
+ ...
```

For a description of these parameters, see E-element Parameters on page 233.

Transconductance G(s) form

```
Gxxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1})/ (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2})/ (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3})/ (Re{p3}, Im{p3})
+ ...
```

In the above syntax, parenthesis , commas, and slashes are separators—they have the same meaning as a space. A pole-residue pair is represented by four numbers (real and imaginary part of the residue, then real and imaginary part of the pole).

You must make sure that Re[pi]<0; otherwise, the simulations will certainly diverge. Also, it is a good idea to assure passivity of the model (for an N-port admittance matrix Y, Re{Y} should be positive-definite), or the simulation is likely to diverge).

For a description of the G-element parameters, see G-element Parameters on page 250.

Example

To represent a G(s) in the form,

$$G(s) = 0.001 + 1\times 10^{-12}s + \frac{0.0008}{s + 1\times 10^{10}} + \frac{(0.001 - j0.006)}{s - (-1\times 10^{8} + j1.8\times 10^{10})} + \frac{(0.001 + j0.006)}{s - (-1\times 10^{8} - j1.8\times 10^{10})}$$

You would input:

```
G1 1 0 FOSTER 2 0 0.001 1e-12
+(0.0004, 0)/(-1e10, 0) (0.001, -0.006)/(-1e8, 1.8e10)
```

Note:

In the case of a real poles, half the residue value is entered because it is essentially applied twice. In the above example, the first pole-residue pair is real, but is written as “A1/(s-p1)+A1/(s-p1)”; therefore, 0.0004 is entered rather than 0.0008.

Behavioral Voltage Source (Noise Model)

You can implement a behavioral voltage noise source with an E-element. As noise elements, these are two-terminal elements that represent a noise source connected between two specified nodes.

```
Exxx n+ n- noise='expression'
```

Where,
Exxx is the voltage-controlled element name, which must being with “E”, followed by up to 1023 alphanumeric characters.
n+ is the positive node.
n- is the negative node.
noise='expression' can contain the bias, frequency, or other parameters.

Data form

```
Exxx n+ n- noise data=dataname
.DATA dataname
+ pname1 pname2
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is V^2/Hz.

Ideal Op-Amp

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

You can also substitute `LEVEL=1` in place of `OPAMP`:

```
Exxx n+ n- in+ in- level=1
```

For a description of these parameters, see E-element Parameters.

Ideal Transformer

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

You can also substitute `LEVEL=2` in place of `TRANSFORMER`:

```
Exxx n+ n- in+ in- level=2 k
```

For a description of these parameters, see E-element Parameters.

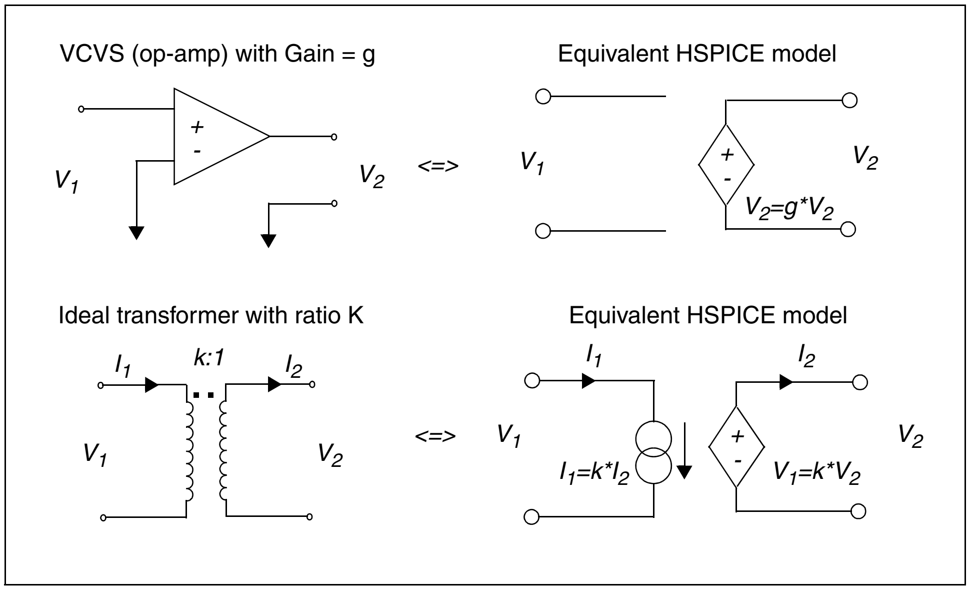


Figure 25 Equivalent VCVS and Ideal Transformer HSPICE Models

E-element Parameters

The E-element parameters are described in the following list.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
DELAY	Keyword for the delay element. Same as for the voltage-controlled voltage source, except it has an associated propagation delay, TD. This element adjusts propagation delay in macro (subcircuit) modeling. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of the piecewise linear corners. This parameter defaults to one-fourth of the smallest distance between breakpoints. The maximum is one-half of the smallest distance between breakpoints.
Exxx	Voltage-controlled element name. Must begin with E, followed by up to 1023 alphanumeric characters.
gain	Voltage gain.
gatetype(k)	Can be AND, NAND, OR, or NOR. *k* represents the number of inputs of the gate. *x* and *y* represent the piecewise linear variation of output, as a function of input. In multi-input gates, only one input determines the state of the output.
IC	Initial condition: initial estimate of controlling voltage value(s). If you do not specify IC, default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
k	Ideal transformer turn ratio: $\mathrm{V(in+,in-)} = k \cdot \mathrm{V(n+,n-)}$ 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.

Parameter	Description
n+/-	Positive or negative node of a controlled element.
NDIM	Number of polynomial dimensions. If you do not set POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle\ TD,\ tstop\rangle}{tstep},\ 10\right]$ The .TRAN statement specifies tstep and tstop values.
LEVEL=<x>	Interchangeable function keyword such as `VCCS`, `VCAP`, etc.
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 or HSPICE RF assumes that it is P1 (P0=0.0), and that the element is linear. If you specify more than one polynomial coefficient, the element is nonlinear, and P0, P1, P2 ... represent them (see Polynomial Functions on page 216).
POLY	Keyword for the polynomial function. If you do not specify `POLY(`*`ndim`*`)`, *HSPICE assumes a one-dimensional polynomial.* *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.

Parameter	Description
TRANSFORMER or LEVEL=2	Keyword for an ideal transformer. TRANSFORMER is a reserved word; do not use it as a node name.
VOL	Voltage output that flows from n+ to n-. The expression that you define can be a function of: ▪ node voltages ▪ branch currents ▪ time (time variable) ▪ temperature (temper variable) ▪ frequency (hertz variable)
VCVS	Keyword for a voltage-controlled voltage source. VCVS is a reserved word; do not use it as a node name.
x1,...	Controlling voltage across the *in*+ and *in*- nodes. The *x* values must be in increasing order.
y1,...	Corresponding element values of *x*.

E-element Examples

Ideal OpAmp

You can use the voltage-controlled voltage source to build a voltage amplifier, with supply limits.

- The output voltage across nodes 2,3 is v(14,1) * 2.
- The value of the voltage gain parameter is 2.
- The `MAX` parameter sets a maximum E1 voltage of 5 V.
- The `MIN` parameter sets a minimum E1 voltage output of -5 V.

Example

If V(14,1)=-4V, then HSPICE or HSPICE RF sets E1 to -5V, and not -8V as the equation suggests.

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

To specify a value for polynomial coefficient parameters, use the following format:

```
.PARAM CU=2.0
E1 2 3 14 1 MAX=+5 MIN=-5 CU
```

Voltage Summer

An ideal voltage summer specifies the source voltage, as a function of three controlling voltage(s):

- V(13,0)
- V(15,0)
- V(17,0)

To describe a voltage source, the voltage summer uses this value:

$$V(13,0) + V(15,0) + V(17,0)$$

This example represents an ideal voltage summer. It initializes the three controlling voltages for a DC operating point analysis, to 1.5, 2.0, and 17.25 V.

```
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.25
```

Polynomial Function

A voltage-controlled source can also output a non-linear function, using a one-dimensional polynomial. This example does not specify the `POLY` parameter, so HSPICE or HSPICE RF assumes it is a one-dimensional polynomial—that is, a function of one controlling voltage. The equation corresponds to the element syntax. Behavioral equations replace this older method.

```
V (3,4)=10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)2"
E2 3 4 POLY 21 17 10.5 2.1 1.75
E2 3 4 VOLT="10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)2"
E2 3 4 POLY 21 17 10.5 2.1 1.75
```

Zero-Delay Inverter Gate

Use a piecewise linear transfer function to build a simple inverter, with no delay.

```
Einv out 0 PWL(1) in 0 .7v,5v 1v,0v
```

Delayed and Inverted Signal

You can use an E-element to invert a signal to generate a signal which is delayed and the inverse of another signal. Use the following syntax to generate a signal which is delayed (`TD`) and is the inverse of the input signal (`in`).

```
.option list node post
Vin in 0 pwl(0 0 10n 0 13n 2v 23n 2v 24n 0v)      $$ input signal
       to be inverted and delayed
Edelay in_delay 0 DELAY in 0 TD=2n               $$ signal
       "in_delay" is the "in" signal delayed by TD
Edelay_inv out_inv 0  PWL(1) in1 0 0v 2v 2v 0v    $$ signal
       "out_inv" is the inverted and delayed "in" signal
.tran .1n 30n
.print v(in_delay) v(out_inv) v(in)
.end
```

Differential Amplifiers and Opamp Signals

Defining a differential voltage source to use with differential amplifiers or opamps can be achieved with E-elements. E-elements can be used to provide differential signals to drive circuits requiring differential signals.

```
******spice definition*****
VID 7 0 DC 0 ac=1 SIN (0 0.70 1MEG)
E+ in+ 10 7 0 0.5   $ differential signal 1, level can be varied
by changing gain
E- in- 10 7 0 -0.5  $ differential signal 2
VIC 10 0 DC 0.3V  $VIC is the common mode signal source
```

For a full netlist differential block analysis example, see also: $installdir/demo/hspice/behave/diff1.sp.

Ideal Transformer

If the turn ratio is 10 to 1, the voltage relationship is V(out)=V(in)/10.

```
Etrans out 0 TRANSFORMER in 0 10
```

You can also substitute `LEVEL=2` in place of `TRANSFORMER`:

```
Etrans out 0 in 0 level=2 10
```

Voltage-Controlled Oscillator (VCO)

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

Example 1

In this example, the voltage at the control node controls the frequency of the sinusoidal output voltage at the out node. v0 is the DC offset voltage, and gain is the amplitude. The output is a sinusoidal voltage, whose frequency is specified in *freq · control*.

```
Evco out 0 VOL='v0+gain*SIN(6.28 freq*v(control)*TIME)'
```

Note:

This equation is valid only for a steady-state VCO (fixed voltage). If you sweep the control voltage, this equation does not apply.

Example 2

In this example, a Verilog-A module is used to control VCO output by tracking phase to ensure continuity.

```
`include "disciplines.vams"

module vco(vin, vout);
 inout vin, vout;
 electrical vin, vout;
 parameter real amp = 1.0;
 parameter real offset = 1.0;
 parameter real center_freq = 1G;
 parameter real vco_gain = 1G;
 real phase;

 analog begin
  phase = idt(center_freq + vco_gain*V(vin), 0.0);
  V(vout) <+ offset+amp*sin(6.2831853*phase);
 end
endmodule
```

Example 3

This example is a controlled-source equivalent of the Verilog-A module shown in the previous example. Like the previous example, it establishes a continuous phase and therefore, a continuous output voltage.

```
.subckt vco in out amp=1 offset=1 center_freq=1 vco_gain=1
.ic v(phase)=0
cphase phase 0 1e-10
g1 0 phase cur='1e-10*(center_freq+vco_gain*v(in))'
eout out 0 vol='offset+amp*sin(6.2831853*v(phase))'
.ends
```

Example 4

In this example, controlled-sources are used to control VCO output.

```
.param pi=3.1415926
.param twopi='2*pi'
.subckt vco in inb out outb f0=100k kf=50k out_off=0.0 out_amp=1.0
gs 0 s poly(2) c 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
gc c 0 poly(2) s 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
cs s 0 1e-9 ic=0
cc c 0 1e-9 ic=1
eout out 0 vol='out_off+(out_amp*v(s))'
eoutb outb 0 vol='out_off+(out_amp*v(c))'
.ic v(c)=1 v(s)=0
.ends
```

Using the E-element for AC Analysis

The following equation describes an E-element:

```
E1 ee 0 vol=f(v(1), v(2))
```

In an AC analysis, voltage is computed as follows:

v(ee)=A * delta_v1 + B * delta_v2

Where,

- A is the derivative of f(v(1), v(2)) to v(1) at the operating point
- B is the derivative of f(v(1), v(2)) to v(2) at the operating point
- delta_v1 is the AC voltage variation of v(1)
- delta_v2 is the AC voltage variation of v(2)

Example

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

```
*****************************************************
****** E element for AC analysis
.option post
.op
*CASE1-Mixed and zero time unit has zero value(tran)
v_n1 n1 gnd dc=6.0 pwl 0.0 6.0 1.0n 6.0 ac 5.0
v_n2 n2 gnd dc=4.0 pwl 0.0 4.0 1.0n 6.0 ac 2.0
e1 n3 gnd vol='v(n1)+v(n2)'
e2 n4 gnd vol='v(n1)*v(n2)'
r1 n1 gnd 1
r2 n2 gnd 1
r3 n3 gnd 1
r4 n4 gnd 1
.tran 10p 3n
.ac dec 1 1 100meg
.print ac v(n?)
.end
*****************************************************
```

The AC voltage of node n3 is:

```
v(n3)=1.0 *v(n1)(ac)  +  1.0 * v(n2)(ac)
   =  1.0 * 5.0    +  1.0  *  2.0
   =  7.0  (v)
```

The AC voltage of node n4 is:

```
v(n4)=v(n2)(op) * v(n1)(ac) + v(n1)(op) * v(n2)(ac)
   =  4.0   *   5.0  +  6.0   *  2.0
   =  32.0 (v)
```

Current-Dependent Current Sources — F-elements

This section explains the F-element syntax and parameters.

Note:

G-elements with algebraics make F-elements obsolete. You can still use F-elements for backward-compatibility with existing designs.

Current-Controlled Current Source (CCCS) Syntax

Linear

```
Fxxx n+ n- <CCCS> vn1 gain <MAX=val> <MIN=val> <SCALE=val>
```

```
+ <TC1=val> <TC2=val> <M=val> <ABS=1> <IC=val>
```

Polynomial (POLY)

```
Fxxx n+ n- <CCCS> POLY(ndim) vn1 <... vnndim> <MAX=val>
+ <MIN=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ <ABS=1> p0 <p1…> <IC=val>
```

In this syntax, *dim* (*dimensions*) ≤3.

Piecewise Linear (PWL)

```
Fxxx n+ n- <CCCS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> <M=val> x1,y1  ... x100,y100 <IC=val>
```

Multi-Input Gates

```
Fxxx n+ n- <CCCS> gatetype(k) vn1, ... vnk <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <ABS=1>
+ x1,y1  ...   x100,y100 <IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

Note:

G-elements with algebraics make F-elements obsolete. You can still use F-elements for backward-compatibility with existing designs.

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

F-element Parameters

The F-element parameters are described in the following list.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCCS	Keyword for current-controlled current source. CCCS is a HSPICE reserved keyword; do not use it as a node name.

Parameter	Description
DELAY	Keyword for the delay element. Same as for a current-controlled current source, but has an associated propagation delay, TD. Adjusts the propagation delay in the macro model (subcircuit) process. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. The maximum is 1/2 of the smallest distance between breakpoints.
Fxxx	Element name of the current-controlled current source. Must begin with F, followed by up to 1023 alphanumeric characters.
gain	Current gain.
gatetype(k)	AND, NAND, OR, or NOR. *k* is the number of inputs for the gate. *x* and *y* are the piecewise linear variation of the output, as a function of input. In multi-input gates, only one input determines the output state. Do not use the above keywords as node names.
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.
M	Number of replications of the element, in parallel.
MAX	Maximum output current. Default=undefined; sets no maximum.
MIN	Minimum output current. Default=undefined; sets no minimum.
n+/-	Connecting nodes for a positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$ The `.TRAN` statement specifies the tstep and tstop values.

Parameter	Description
P0, P1 …	The polynomial coefficients. If you specify one coefficient, HSPICE or HSPICE RF assumes it is P1 (P0=0.0), and the source element is linear. If you specify more than one polynomial coefficient, then the source is non-linear, and HSPICE or HSPICE RF assumes that the polynomials are P0, P1, P2 … See Polynomial Functions on page 216.
POLY	Keyword for the polynomial function. If you do not specify POLY(*ndim*), HSPICE assumes that this is a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
vn1 …	Names of voltage sources, through which the controlling current flows. Specify one name for each dimension.
x1,...	Controlling current, through the *vn1* source. Specify the *x* values in increasing order.
y1,...	Corresponding output current values of *x*.

F-element Examples

Example 1

This example describes a current-controlled current source, connected between nodes 13 and 5. The current, which controls the value of the controlled source, flows through the voltage source named `VSENS`.

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

Note:

To use a current-controlled current source, you can place a dummy independent voltage source into the path of the controlling current.

The defining equation is:

$I(F1) = 5 \cdot\ I(VSENS)$

- Current gain is 5.
- Maximum current flow through F1 is 3 A.
- Minimum current flow is -3 A.

If `I(VSENS)=2 A`, then this examples sets `I(F1)` to 3 amps, not 10 amps (as the equation suggests). You can define a parameter for the polynomial coefficient(s):

```
.PARAM VU=5
F1 13 5 VSENS MAX=+3 MIN=-3 VU
```

Example 2

This example is a current-controlled current source, with the value:

I(F2)=1e-3 + 1.3e-3 · I(VCC)

Current flows from the positive node, through the source, to the negative node. The positive controlling-current flows from the positive node, through the source, to the negative node of `vnam` (linear), or to the negative node of each voltage source (nonlinear).

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

Example 3

This example is a delayed, current-controlled current source.

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

Example 4

This example is a piecewise-linear, current-controlled current source.

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

Voltage-Dependent Current Sources — G-elements

This section explains G-element syntax statements, and their parameters.

```
 Gxxx n+ n- <VCCS|LEVEL=0> in+ in- ...
```

- `LEVEL=0` is a Voltage-Controlled Current Source (VCCS).
- `LEVEL=1` is a Voltage-Controlled Resistor (VCR).

- `LEVEL=2` is a Voltage-Controlled Capacitor (VCCAP), Negative Piece-Wise Linear (NPWL).
- `LEVEL=3` is a VCCAP, Positive Piece-Wise Linear (PPWL).

See also Using G- and E-elements.

Voltage-Controlled Current Source (VCCS)

Linear

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

For a description of the G-element parameters, see G-element Parameters on page 250.

Polynomial (POLY)

```
Gxxx n+ n- <VCCS> POLY(NDIM) in1+ in1- ... <inndim+ inndim->
+ <MAX=val> <MIN=val> <SCALE=val> <M=val> <TC1=val>
+ <TC2=val> <ABS=1> P0<P1…> <IC=vals>
```

For a description of the G-element parameters, see G-element Parameters on page 250. For a description of possible POLY syntaxes, see Polynomial Functions on page 216.

Piecewise Linear (PWL)

```
Gxxx n+ n- <VCCS> PWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
Gxxx n+ n- <VCCS> NPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val><TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
Gxxx n+ n- <VCCS> PPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

For a description of the G-element parameters, see G-element Parameters on page 250.

Multi-Input Gate

```
Gxxx n+ n- <VCCS> gatetype(k) in1+ in1- ...
+ ink+ ink- <DELTA=val> <TC1=val> <TC2=val> <SCALE=val>
+ <M=val> x1,y1 ... x100,y100<IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates. For a description of the G-element parameters, see G-element Parameters on page 250.

Delay Element

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

For a description of the G-element parameters, see G-element Parameters on page 250.

Laplace Transform

For details, see Laplace Transform on page 227.

Pole-Zero Function

For details, see Pole-Zero Function on page 228.

Frequency Response Table

For details, see Frequency Response Table on page 229.

Foster Pole-Residue Form

For details, see Foster Pole-Residue Form on page 230.

Behavioral Current Source (Noise Model)

Expression form

```
gxxx node1 node2 noise='noise_expression'
```

The *xxx* parameter can be set with a value up to 1024 characters. The *node1* and *node2* are the positive and negative nodes that connect to the noise source. The noise expression can contain the bias, frequency, or other parameters, and involve node voltages and currents through voltage sources.

For a description of the G-element parameters, see G-element Parameters on page 250.

This syntax creates a simple two-terminal current noise source, whose value is described in A^2/Hz. The output noise generated from this noise source is:

noise_expression*H

H is the transfer function from the terminal pair (node1,node2) to the circuit output, where the output noise is measured.

You can also implement a behavioral noise source with an E-element. As noise elements, they are two-terminal elements that represent a noise source connected between two specified nodes.

```
gname node1 node2 node3 node4 noise='expression'
```

This syntax produces a noise source correlation between the terminal pairs (`node1 node2`) and (`node3 node4`). The resulting output noise is computed as:

noise_expression*sqrt(H1*H2*)

- H1 is the transfer function from (node1,node2) to the output.
- H2 is the transfer function from (node3,node4) to the output.

The noise expression can involve node voltages and currents through voltage sources.

Data form

```
Gxxx node1 node2 noise data=dataname
.DATA dataname
+ pname1 pname2
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is A^2/Hz.

For a description of the G-element parameters, see G-element Parameters on page 250.

Example

The following netlist shows a 1000 ohm resistor (`g1`) using a G-element. The `g1noise` element, placed in parallel with the `g1` resistor, delivers the thermal noise expected from a resistor. The `r1` resistor is included for comparison: The noise due to `r1` should be the same as the noise due to `g1noise`.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2
+ noise='4*1.3806266e-23*(TEMPER+273.15)*0.001'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Voltage-Controlled Resistor (VCR)

Linear

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

For a description of the G-element parameters, see G-element Parameters on page 250.

Polynomial (POLY)

```
Gxxx n+ n- VCR POLY(NDIM) in1+ in1- ...
+ <inndim+ inndim-> <MAX=val> <MIN=val><SCALE=val>
+ <M=val> <TC1=val> <TC2=val>    P0 <P1…> <IC=vals>
```

For a description of the G-element parameters, see G-element Parameters on page 250.

Piecewise Linear (PWL)

```
Gxxx n+ n- VCR PWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR NPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR PPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

For a description of the G-element parameters, see G-element Parameters on page 250.

Multi-Input Gates

```
Gxxx n+ n- VCR gatetype(k) in1+ in1- ... ink+ ink-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ x1,y1 ... x100,y100 <IC=val>
```

For a description of the G-element parameters, see G-element Parameters on page 250.

Voltage-Controlled Capacitor (VCCAP)

```
Gxxx n+ n- VCCAP PWL(1) in+ in-   <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

HSPICE or HSPICE RF uses either `LEVEL=2` (`NPWL`) or `LEVEL=3` (`PPWL`), based on the relationship of the (n+, n-) and (in+, in-) nodes. For a description of the G-element parameters, see G-element Parameters on page 250.

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+*,*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+*).

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

G-element Parameters

The G-element parameters described in the following list.

Parameter	**Description**
ABS	Output is an absolute value, if ABS=1.
CUR, VALUE, NOISE	Current output that flows from n+ to n-. The expression that you define can be a function of: ■ node voltages ■ branch currents ■ time (`time` variable) ■ temperature (`temper` variable) ■ frequency (`hertz` variable)
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.

Parameter	Description
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.
LEVEL=<x>	Function keyword such as VCCS, VCAP, etc.
IC	Initial condition. Initial estimate of the value(s) of controlling voltage(s). If you do not specify IC, the default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
M	Number of replications of the elements in parallel.
MAX	Maximum value of the current or resistance. The default is undefined, and sets no maximum value.
MIN	Minimum value of the current or resistance. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of the controlled element.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = max\left[\frac{min\langle\ TD,\ tstop\rangle}{tstep},\ 10\right].$ The .TRAN statement specifies the tstep and tstop values.
NPWL	Models symmetrical bidirectional switch/transfer gate, NMOS.

Parameter	Description
P0, P1 …	The polynomial coefficients. ▪ If you specify one coefficient, HSPICE or HSPICE RF assumes that it is P1 (P0=0.0), and 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 216).
POLY	Keyword for the polynomial dimension function. If you do not specify *POLY*(*ndim*), HSPICE assumes that it is a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for the piecewise linear function.
PPWL	Models symmetrical bidirectional switch/transfer gate, PMOS.
SCALE	Multiplier for the element value.
SMOOTH	For piecewise-linear, dependent-source elements, SMOOTH selects the curve-smoothing method. A curve-smoothing method simulates exact data points that you provide. You can use this method to make HSPICE or HSPICE RF simulate specific data points, which correspond to either measured data or data sheets. Choices for SMOOTH are 1 or 2: ▪ Selects the smoothing method used in Hspice versions before release H93A. Use this method to maintain compatibility with simulations that you ran, using releases older than H93A. ▪ Selects the smoothing method, which uses data points that you provide. This is the default for Hspice versions starting with release H93A.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$.
TD	Keyword for the time (propagation) delay.
transconductance	Voltage-to-current conversion factor.
transfactor	Voltage-to-resistance conversion factor.

Parameter	**Description**
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

Modeling Switches

You can model a switch to be open or closed based on simulation time or a pair of controlling nodes.

Switch Example 1: Time-varying switch—use the built-in function TIME to change the value of R from 0 (closed) to 100g ohm (open) when the simulator reaches time value T1:

```
R1 n1 n2 '100g*(TIME <= T1)'
```

As long as TIME $\leq$T1, the expression evaluates to zero and so does the resistor (switch) value. The resistor could easily be rewritten to switch from closed to open:

```
R1 n1 n2 '100g*(TIME >= T1)'
```

Switch example 2: Voltage-controlled switch—use a voltage-controlled resistor and the PWL (piece-wise linear) function. The point-value pair represents the controlling input and output resistance respectively

```
G_Switch n1 n2 VCR PWL(1) c1 c2 0v,100g 1v,1p
```

where: `n1` and `n2` are the poles of the switch, and `c1` and `c2` are the control nodes. In the following sample netlist, the switch is controlled by the PWL voltage source to switch at `1us`:

```
* g-element switch
.option post
V_ctrl c1 0 PWL (0 0v .99u .001v 1u 1v)
G_Switch  n1 n2 VCR PWL(1) c1 0 0v,100g 1v,1p
V_ref n1 0 10v
R_load n2 0 100
.tran .1u 2u
.end
```

Switch example 3—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
```

Runtime Current Source with Equation Containing Output Variable

HSPICE does not support a runtime output variable such as `v(gate)` in the example equation that follows. If the .DATA block has a runtime current source (I-element) where an equation contains runtime output variable such as `v(gate)`, as in this example equation,

```
I0 1 0 '(1-a0*v(gate))/b0'
vg  gate  0 '(gt_vl)'  $ (gt_vl)
```

—then the recommended method is to use the G-element:

```
g0 1 0 cur='((1-(a0*v(gate)))/b0)'
```

Voltage-Controlled Capacitor

The capacitance value across the (*out,0*) nodes varies linearly (from 1 p to 5 p), when voltage across the ctrl,0 nodes varies between 2 v and 2.5 v. The capacitance value remains constant at 1 picofarad when below the lower voltage limit, and at 5 picofarads when above the upper voltage limit.

```
Gcap out 0 VCCAP PWL(1) ctrl 0 2v,1p 2.5v,5p
```

Zero-Delay Gate

To implement a two-input AND gate, use an expression and a piecewise linear table.

- The inputs are voltages at the a and b nodes.
- The output is the current flow from the out to 0 node.
- HSPICE or HSPICE RF multiplies the current by the `SCALE` value—which in this example, is the inverse of the load resistance, connected across the out,0 nodes.

  ```
  Gand out 0 AND(2) a 0 b 0 SCALE='1/rload' 0v,0a 1v,.5a
  + 4v,4.5a 5v,5a
  ```

Delay Element

A delay is a low-pass filter type delay, similar to that of an opamp. In contrast, a transmission line has an infinite frequency response. A glitch input to a G delay attenuates in a way that is similar to a buffer circuit. In this example, the output of the delay element is the current flow from the *out* node to the *1* node with a value equal to the voltage across the (*in*, *0*) nodes, multiplied by the `SCALE` value, and delayed by the `TD` value.

```
Gdel out 0 DELAY in 0 TD=5ns SCALE=2 NPDELAY=25
```

Diode Equation

To model forward-bias diode characteristics from node 5 to ground use a runtime expression. The saturation current is 1e-14 amp and the thermal voltage is 0.025 v.

```
Gdio 5 0 CUR='1e-14*(EXP(V(5)/0.025)-1.0)'
```

Diode Breakdown

You can model the diode breakdown region to a forward region. When voltage across a diode is above or below the piecewise linear limit values (-2.2v, 2v), the diode current remains at the corresponding limit values (-1a, 1.2a).

```
Gdiode 1 0 PWL(1) 1 0 -2.2v,-1a -2v,-1pa .3v,.15pa
+.6v,10ua 1v,1a 2v,1.2a
```

Triodes

Both of the following voltage-controlled current sources implement a basic triode.

- The first example uses the poly(2) operator, to multiply the anode and grid voltages together, and to scale by .02.
- The second example uses the explicit behavioral algebraic description.

```
gt i_anode cathode poly(2) anode,cathode
+ grid,cathode 0 0 0 0 .02

gt i_anode cathode
+ cur='20m*v(anode,cathode)*v(grid,cathode)'
```

Behavioral Noise Model

The following netlist shows a 1000 Ohm resistor (g1) implemented using a G-element. The g1noise element, placed in parallel with the g1 resistor, delivers the thermal noise expected from a resistor. The r1 resistor is included for comparison: the noise due to r1 should be the same as the noise due to g1noise.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2 noise='sqrt(4*1.3806266e-23*(TEMPER+273.15)*0.001)'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Current-Dependent Voltage Sources — H-elements

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

Note:

E-elements with algebraics make H-elements obsolete. You can still use H-elements for backward-compatibility with existing designs.

Current-Controlled Voltage Source (CCVS)

Linear

```
Hxxx n+ n- <CCVS> vn1 transresistance <MAX=val> <MIN=val>
+ <SCALE=val> <TC1=val><TC2=val> <ABS=1> <IC=val>
```

Polynomial (POLY)

```
Hxxx n+ n- <CCVS> POLY(NDIM) vn1 <... vnndim>
+ <MAX=val><MIN=val> <TC1=val> <TC2=val> <SCALE=val>
+ <ABS=1> P0  <P1…> <IC=val>
```

Piecewise Linear (PWL)

```
Hxxx n+ n- <CCVS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1  ...   x100,y100 <IC=val>
```

Multi-Input Gate

```
Hxxx n+ n- gatetype(k) vn1, ...vnk <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1 ...   x100,y100 <IC=val>
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

Note:

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

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

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCVS	Keyword for the current-controlled voltage source. CCVS is a HSPICE reserved keyword; do not use it as a node name.
DELAY	Keyword for the delay element. Same as for a current-controlled voltage source, but has an associated propagation delay, TD. Use this element to adjust the propagation delay in the macro (subcircuit) model process. DELAY is a HSPICE reserved keyword; do not use it as a node name.
DELTA	Controls curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. Maximum is 1/2 of the smallest distance between breakpoints.
gatetype(k)	Can be AND, NAND, OR, or NOR. The *k* value is the number of inputs of the gate. The *x* and *y* terms are the piecewise linear variation of the output, as a function of the input. In multi-input gates, one input determines the output state.
Hxxx	Element name of current-controlled voltage source. Must start with H, followed by up to 1023 alphanumeric characters.
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.
MAX	Maximum voltage. Default is undefined; sets no maximum.
MIN	Minimum voltage. Default is undefined; sets no minimum.
n+/-	Connecting nodes for positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE or HSPICE RF assumes a one-dimensional polynomial. NDIM must be a positive number.

Parameter	**Description**
NPDELAY	Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is: $NPDELAY_{default} = max\left[\frac{min\langle TD, tstop\rangle}{tstep}, 10\right]$. The .TRAN statement specifies the tstep and tstop values.
P0, P1 . . .	Polynomial coefficients. ▪ If you specify one polynomial coefficient, the source is linear, and HSPICE or HSPICE RF assumes that the polynomial is P1 (P0=0.0). ▪ If you specify more than one polynomial coefficient, the source is non-linear. HSPICE assumes the polynomials are P0, P1, P2 … See Polynomial Functions on page 216.
POLY	Keyword for polynomial dimension function. If you do not specify *POLY*(*ndim*), HSPICE assumes a one-dimensional polynomial. *Ndim* must be a positive number.
PWL	Keyword for a piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
transresistance	Current-to-voltage conversion factor.
vn1 …	Names of voltage sources, through which controlling current flows. You must specify one name for each dimension.
x1,...	Controlling current, through the *vn1* source. Specify the *x* values in increasing order.
y1,...	Corresponding output voltage values of *x*.

Example 1

```
HX 20 10 VCUR MAX=+10 MIN=-10 1000
```

The example above selects a linear current-controlled voltage source. The controlling current flows through the dependent voltage source, called VCUR.

Example 2

The defining equation of the CCVS is:

$$HX = 1000 \cdot I(VCUR)$$

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

- If the equation produces a value of HX greater than +10 V, then the `MAX` parameter sets HX to 10 V.
- If the equation produces a value of HX less than -10 V, then the `MIN` parameter sets HX to -10 V.

VCUR is the name of the independent voltage source, through which the controlling current flows. If the controlling current does not flow through an independent voltage source, you must insert a dummy independent voltage source.

Example 3

```
.PARAM CT=1000
HX 20 10 VCUR MAX=+10 MIN=-10 CT
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5, 1.3
```

The example above describes a dependent voltage source, with the value:

$$V = I(VIN1) \cdot I(VIN2)$$

This two-dimensional polynomial equation specifies:

- FA1=VIN1
- FA2=VIN2
- P0=0
- P1=0
- P2=0
- P3=0
- P4=1

The initial controlling current is .5 mA through VIN1, and 1.3 mA for VIN2.

Positive controlling current flows from the positive node, through the source, to the negative node of vnam (linear). The (non-linear) polynomial specifies the source voltage, as a function of the controlling current(s).

Specifying a Digital Vector File and Mixed Mode Stimuli

HSPICE and HSPICE RF input netlists support digital vector files. A VEC file consists of three parts:

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

To incorporate this information into your simulation, include the `.VEC` command in your netlist.

Commands in a Digital Vector File

For descriptions of all commands that you can use in a VEC file, see Digital Vector File Commands in the *HSPICE Reference Manual: Commands and Control Options*.

Vector Patterns

The *Vector Pattern Definition* section defines the vectors, their names, sizes, signal direction, sequence or order for each vector stimulus, and so on. A RADIX line must occur first and the other lines can appear in any order in this section. All keywords are case-insensitive.

Here is an example Vector Pattern Definition section:

```
; start of Vector Pattern Definition section
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
```

These four lines are required and appear in the first lines of a VEC file:

- `RADIX` defines eight single-bit vectors.
- `VNAME` gives each vector a name.

- `IO` determines which vectors are inputs, outputs, or bidirectional signals. In this example, all eight are input signals.
- `TUNIT` indicates that the time unit for the tabular data to follow is in units of nanoseconds.

For additional information about these keywords, see Defining Tabular Data on page 262.

Defining Tabular Data

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

The *Tabular Data* section defines (in tabular format) the values of the signals at specified times. Rows in the Tabular Data section must appear in chronological order because row placement carries sequential timing information. Its general format is:

```
time1 signal1_value1 signal2_value1 signal3_value1...
time2 signal1_value2 signal2_value2 signal3_value2...
time3 signal1_value3 signal2_value3 signal3_value3...
.
.
```

Where *timex* is the specified time, and *signaln_valuen* is the values of specific signals at specific points in time. The set of values for a particular signal (over all times) is a vector, which appears as a vertical column in the tabular data and vector table. The set of all *signal1_valuen* constitutes one vector.

For example,

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

This example shows that:

- At 11.0 time units, the value for the first and fifth vectors is 1.
- At 20.0 time units, the first, second, fifth, and sixth vectors are 1.
- At 33.0 time units, the first, third, fifth, and eighth vectors are 1.

Input Stimuli

HSPICE or HSPICE RF converts each input signal into a PWL (piecewise linear) voltage source, and a series resistance. Table 15 shows the legal states for an input signal. Signal values can have any of these legal states.

Table 15 Legal States for an Input Signal

State	Description
0	Drive to ZERO (gnd). Resistance set to 0.
1	Drive to ONE (vdd). Resistance set to 0.
Z, z	Floating to HIGH IMPEDANCE. A `TRIZ` statement defines resistance value.
X, x	Drive to ZERO (gnd). Resistance set to 0.
L	Resistive drive to ZERO (gnd). An `OUT` or `OUTZ` statement defines resistance value.
H	Resistive drive to ONE (vdd). An `OUT` or `OUTZ` statement defines resistance value.
U, u	Drive to ZERO (gnd). Resistance set to 0.

Expected Output

HSPICE or HSPICE RF converts each output signal into a `.DOUT` statement in the netlist. During simulation, HSPICE or HSPICE RF compares the actual results with the expected output vector(s). If the states are different, an error message appears. The legal states for expected outputs include the values listed in Table 16.

Table 16 Legal States for an Output Signal

State	Description
0	Expect ZERO.
1	Expect ONE.
X, x	Don’t care.

Table 16 Legal States for an Output Signal

U, u	Don't care.
Z, z	Expect HIGH IMPEDANCE (don't care). Simulation evaluates Z, z as "don't care" because HSPICE or HSPICE RF cannot detect a high impedance state.

For example,

```
...
IO OOOO
; start of tabular section data
11.0 1001
20.0 1100
30.0 1000
35.0 xx00
```

Where,

- The first line is a comment line because of the semicolon character.
- The second line expects the output to be 1 for the first and fourth vectors, while all others are expected to be low.
- At 20 time units, HSPICE or HSPICE RF expects the first and second vectors to be high, and the third and fourth to be low.
- At 30 time units, HSPICE or HSPICE RF expects only the first vector to be high, and all others low.
- At 35 time units, HSPICE or HSPICE RF expects the output of the first two vectors to be "don't care"; it expects vectors 3 and 4 to be low.

Verilog Value Format

HSPICE or HSPICE RF accepts Verilog-sized format to specify numbers; for example,

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

Where:

- <*size*> specifies the number of bits, in decimal format.
- *<base format>* indicates:
  - binary ('b or 'B)

- octal ('o or 'O)
- hexadecimal ('h or 'H).

- <*number*> values are combinations of the *0*, *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *A*, *B*, *C*, *D*, *E*, and *F* characters. Depending on what base format you choose, only a subset of these characters might be legal.

  You can also use unknown values (X) and high-impedance (Z) in the <*number*> field. An X or Z sets four bits in the hexadecimal base, three bits in the octal base, or one bit in the binary base.

  If the most significant bit of a number is 0, X, or Z, HSPICE or HSPICE RF automatically extends the number (if necessary), to fill the remaining bits with 0, X, or Z, respectively. If the most significant bit is 1, HSPICE or HSPICE RF uses 0 to extend it.

  For example,

```
4'b1111
12'hABx
32'bZ
8'h1
```

  This example specifies values for:

  - 4-bit signal in binary
  - 12-bit signal in hexadecimal
  - 32-bit signal in binary
  - 8-bit signal in hexadecimal

    Equivalents of these lines in non-Verilog format, are:

```
1111
AB xxxx
ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ
0000 0001
```

Periodic Tabular Data

Tabular data is often periodic, so you do not need to specify the absolute time at every time point. When you specify the `PERIOD` statement, the Tabular Data section omits the absolute times. For more information, see Defining Tabular Data on page 262.

For example, the `PERIOD` statement in the following sets the time interval to 10ns between successive lines in the tabular data. This is a shortcut when you use vectors in regular intervals throughout the entire simulation.

```
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
PERIOD 10
; start of vector data section
1000 1000
1100 1100
1010 1001
```

Waveform Characteristics

The *Waveform Characteristics* section defines various attributes for signals, such as the rise or fall time, the thresholds for logic high or low, and so on. For example,

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

The waveform characteristics are based on a bit-mask. Where:

- The `TRISE` (signal rise time) setting of 0.3ns applies to the first four vectors, but not to the last four.
- The example does not show how many bits are in each of the first four vectors, although the first vector is at least one bit.
- The fourth vector is four bits because F is hexadecimal for binary 1111.
- All bits of the fourth vector have a rise time of 0.3ns for the constant you defined in `TUNIT`. This also applies to `TFALL` (fall time), `VIH` (voltage for logic-high inputs), and `VIL` (voltage for logic-low inputs).

Modifying Waveform Characteristics

The `TDELAY`, `IDELAY`, and `ODELAY` statements define the delay time of the signal, relative to the absolute time of each row in the Tabular Data section.

- `TDELAY` applies to the input and output delay time of input, output, and bidirectional signals.
- `IDELAY` applies to the input delay time of bidirectional signals.
- `ODELAY` applies to the output delay time of bidirectional signals.

The `SLOPE` statement specifies the rise and fall times for the input signal. To specify the signals to which the *slope* applies, use a mask.

The `TFALL` statement sets an input fall time for specific vectors.

The `TRISE` statement sets an input rise time for specific vectors.

The `TUNIT` statement defines the time unit.

The `OUT` and `OUTZ` keywords are equivalent, and specify output resistance for each signal (for which the mask applies); `OUT` (or `OUTZ`) applies only to input signals.

The `TRIZ` statement specifies the output impedance, when the signal (for which the mask applies) is in tristate; `TRIZ` applies only to the input signals.

The `VIH` statement specifies the logic-high voltage for each input signal to which the mask applies.

The `VIL` statement specifies the logic-low voltage for each input signal to which the mask applies.

Similar to the `TDELAY` statement, the `VREF` statement specifies the name of the reference voltage for each input vector to which the mask applies. `VREF` applies only to input signals.

Similar to the `TDELAY` statement, the `VTH` statement specifies the logic threshold voltage for each output signal to which the mask applies. The threshold voltage determines the logic state of output signals for comparison with the expected output signals.

The `VOH` statement specifies the logic-high voltage for each output signal to which the mask applies.

The `VOL` statement specifies the logic-low voltage for each output signal to which the mask applies.

Using the Context-Based Control Option

The `OPTION CBC` (Context-Based Control) specifies the direction of bidirectional signals. A bidirectional signal is an input if its value is `0`, `1`, or `Z`; conversely, a bidirectional signal is an output if its value is `H`, `L`, `U`, or `X`.

For example,

```
RADIX 1 1 1
IO I O B
VNAME A Z B
OPTION CBC
10.0 0 X L
20.0 1 1 H
30.0 1 0 Z
```

This example sets up three vectors, named A, Z, and B. Vector A is an input, vector Z is an output, and vector B is a bidirectional signal (defined in the IO statement).

The OPTION CBC line turns on context-based control. The next line sets vector A to a logic-low at 10.0 ns, and vector Z is care.”do not care.” Because the L value is under vector B, HSPICE expects a logic-low output.

At 20 ns, vector A transitions high, and the expected outputs at vectors Z and B are high. Finally, at 30 ns, HSPICE expects vector Z to be low, vector B changes from an output to a high-impedance input, and vector the A signal does not change.

Comment Lines and Line Continuations

Any line in a VEC file that begins with a semicolon (;) is a comment line. Comments can also start at any point along a line. HSPICE or HSPICE RF ignores characters after a semicolon. For example,

```
; This is a comment line
radix 1 1 4 1234 ; This is a radix line
```

As in netlists, any line in a VEC file that starts with a plus sign (+) is a continuation from the previous line.

Parameter Usage

You can use .PARAM statements with some VEC statements when you run HSPICE. These VEC statements fall into the three groups, which are described in the following sections. No other VEC statements but those identified here support .PARAM statements.

First Group

- `PERIOD`
- `TDELAY`
- `IDELAY`
- `ODELAY`
- `SLOPE`
- `TRISE`
- `TFALL`

For these statements, the `TUNIT` statement defines the time unit. If you do not include a `TUNIT` statement, the default time unit value is ns.

Do not specify absolute unit values in a `.PARAM` statement. For example, if in your netlist:

```
.param myperiod=10ns            $ ‘ns’ makes this incorrect
```

And in your VEC file:

```
tunit ns
period myperiod
```

What you wanted for the time period is 10ns; however, because you specified absolute units, 1e-8ns is the value used. In this example, the correct form is:

```
.param myperiod=10
```

Second Group

- `OUT` or `OUTZ`
- `TRIZ`

In these statements, the unit is ohms.

- If you do not include an `OUT` (or `OUTZ`) statement, the default is 0.
- If you do not include a `TRIZ` statement, the default is 1000M.

The `.PARAM` definition for this group follows the HSPICE syntax.

For example, if in your netlist:

```
.param myout=10                $ means 10 ohm
.param mytriz= 10Meg           $ means 10,000,000 ohm, don't
$ confuse Meg with M, M means 0.001
```

And in your VEC file:

```
out myout
triz mytriz
```

Then, HSPICE returns 10 ohm for `OUT` and 10,000,000 ohm for `TRIZ`.

Third Group

- `VIH`
- `VIL`
- `VOH`
- `VOL`
- `VTH`

In these statements, the unit is volts.

- If you do not include an `VIH` statement, the default is 3.3.
- If you do not include a `VIL` statement, the default is 0.0.
- If you do not include a `VOH` statement, the default is 2.64.
- If you do not include an `VOL` statement, the default is 0.66.
- If you do not include an `VTH` statement, the default is 1.65.

Digital Vector File Example

```
; specifies # of bits associated with each vector
radix 1 2 444
;*******************************************************
; defines name for each vector. For multi-bit vectors,
; innermost [] provide the bit index range, MSB:LSB
vname v1 va[[1:0]] vb[12:1]
;actual signal names: v1, va[0], va[1], vb1, vb2, ... vb12
;*******************************************************
; defines vector as input, output, or bi-directional
io i o bbb
; defines time unit
tunit ns
;*******************************************************
; vb12-vb5 are output when 'v1' is 'high'
enable v1 0 0 FF0
; vb4-vb1 are output when 'v1' is 'low'
enable ~v1 0 0 00F
;*******************************************************
; all signals have a delay of 1 ns
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
tdelay 1.0
; va1 and va0 signals have 1.5ns delays
tdelay 1.5 0 3 000
;*******************************************************
; specify input rise/fall times (if you want different
; rise/fall times, use the trise/tfall statement.)
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
slope 1.2
;*******************************************************
; specify the logic 'high' voltage for input signals
vih 3.3 1 0 000
vih 5.0 0 0 FFF
; to specify logic low, use 'vil'
;*******************************************************
; va & vb switch from 'lo' to 'hi' at 1.75 volts
vth 1.75 0 1 FFF

;***************************************************
; tabular data section
10.0 1 3 FFF
20.0 0 2 AFF
30.0 1 0 888
```

10

Parameters and Functions

Describes how to use parameters within HSPICE and HSPICE RF netlists.

Parameters are similar to the variables used in most programming languages. Parameters hold a value that you assign when you create your circuit design or that the simulation calculates based on circuit solution values. Parameters can store static values for a variety of quantities (resistance, source voltage, rise time, and so on). You can also use them in sweep or statistical analysis.

For descriptions of individual commands referenced in this chapter, see Chapter 2, HSPICE and HSPICE RF Netlist Commands in the *HSPICE Reference Manual: Commands and Control Options*.

These topics are covered in the following sections:

- Using Parameters in Simulation (.PARAM)
- Using Algebraic Expressions
- Built-In Functions and Variables
- Parameter Scoping and Passing

Using Parameters in Simulation (.PARAM)

Defining Parameters

Parameters in HSPICE are names that you associate with numeric values. (See Assigning Parameters on page 275.) You can use any of the methods described in Table 17 to define parameters.

Table 17 .PARAM Statement Syntax

Parameter	Description
Simple assignment	.PARAM <SimpleParam>=1e-12
Algebraic definition	.PARAM <AlgebraicParam>='SimpleParam*8.2' SimpleParam excludes the output variable. You can also use algebraic parameters in .PRINT and .PROBE statements. For example: .PRINT AlgebraicParam=par('algebraic expression') You can use the same syntax for .PROBE statements. See Using Algebraic Expressions on page 278.
User-defined function	.PARAM <*MyFunc*(*x, y*)>='Sqrt((*x***x*)+(*y***y*))'
Character string definition	.PARAM <*paramname*>=str('*string*')
Subcircuit default	.SUBCKT <SubName> <ParamDefName>=<Value> str('string') .MACRO <SubName> <ParamDefName>=<Value> str('string')
Predefined analysis function	.PARAM <mcVar>=Agauss(1.0,0.1)
.MEASURE statement	.MEASURE <DC | AC | TRAN> result TRIG ... + TARG ... <GOAL=val> <MINVAL=val> + <WEIGHT=val> <MeasType> <MeasParam> (See Specifying User-Defined Analysis (.MEASURE) on page 317.)
.PRINT | .PROBE	.PRINT | .PROBE + 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). These definitions assign a value of 3 to the *DupParam* parameter.

```
.PARAM DupParam=1
...
.PARAM DupParam=3
```

HSPICE assigns `3` as the value for all instances of `DupParam`, including instances that are earlier in the input than the `.PARAM DupParam=3` statement.

All parameter values in HSPICE are IEEE double floating point numbers. The parameter resolution order is:

1. Resolve all literal assignments.
2. Resolve all expressions.
3. Resolve all function calls.

Table 18 shows the parameter passing order.

Table 18 Parameter Passing Order

.OPTION PARHIER=GLOBAL	.OPTION PARHIER=LOCAL
Analysis sweep parameters	Analysis sweep parameters
.PARAM statement (library)	.SUBCKT call (instance)
.SUBCKT call (instance)	.SUBCKT definition (symbol)
.SUBCKT definition (symbol)	.PARAM statement (library)

Assigning Parameters

You can assign the following types of values to parameters:

- Constant real number
- Algebraic expression of real values
- Predefined function
- Function that you define
- Circuit value
- Model value

To invoke the algebraic processor, enclose a complex expression in single quotes. A simple expression consists of one parameter name.

The parameter keeps the assigned value, unless:

- A later definition changes its value, or
- An algebraic expression assigns a new value during simulation.

HSPICE does not warn you, if it reassigns a parameter.

Example: Modeling an eFuse

You can model an electrically programmable eFUSE device as follows. Instantiate an eFUSE as a subcircuit and pass a parameter that determines whether the eFUSE is “blown” or intact:

```
.subckt efuse in out blown=0
 Rfuse in out r='2*(1-blown)+100e6*blown'
.ends efuse
```

If `blown=0`, then the fuse is intact (2 ohms). If `blown=1` then the fuse is blown and you get the much higher resistance of 100 meg. To use the eFUSE, instantiate it with a subcircuit call:

```
xefuse1 in out efuse blown=0
```

Alternately, you can control the eFUSE with a parameter setting:

```
.param blown=1
x1 in out efuse
```

Inline Parameter Assignments

To define circuit values, use 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`, `.PROBE` or `.MEASURE` statement, use the PAR keyword. (See Chapter 11, Simulation Output, for more information.)

Example

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

User-Defined Function Parameters

You can define a function that is similar to the parameter assignment, but you cannot nest the functions more than two deep.

- An expression can contain parameters that you did not define.
- A function must have at least one argument, and can have up to 20 (and in many cases, more than 20) arguments.
- You can redefine functions.

The format of a function is:

```
funcname1(arg1[,arg2...])=expression1
+ [funcname2(arg1[,arg2...])=expression2] off
```

Parameter	Description
funcname	Specifies the function name. This parameter must be distinct from array names and built-in functions. In subsequently defined functions, all embedded functions must be previously defined.
arg1, arg2	Specifies variables used in the expression.
off	Voids all user-defined functions.

Example

```
.PARAM f(a,b)=POW(a,2)+a*b g(d)=SQRT(d)
+ h(e)=e*f(1,2)-g(3)
```

Predefined Analysis Function

HSPICE includes specialized analysis types, such as Optimization and Monte Carlo, that require a way to control the analysis.

Measurement Parameters

`.MEASURE` statements produce a *measurement* parameter. The rules for measurement parameters are the same as for standard parameters, except that measurement parameters are defined in a `.MEASURE` statement, not in a `.PARAM` statement. For a description of the `.MEASURE` statement, see Specifying User-Defined Analysis (.MEASURE) on page 317.

.PRINT and .PROBE Parameters

`.PRINT,and.PROBE` 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` or `.PROBE` statement, not in a `.PARAM` statement

For more information about the `.PRINT` or `.PROBE` statements, see Displaying Simulation Results on page 296.

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 74.

Using Algebraic Expressions

Note:

Synopsys HSPICE uses double-precision numbers (15 digits) for expressions, user-defined parameters, and sweep variables. For better precision, use parameters (instead of constants) in algebraic expressions because constants are only single-precision numbers (7 digits).

In HSPICE, an algebraic expression, with quoted strings, can replace any parameter in the netlist.

In HSPICE, you can then use these expressions as output variables in `.PRINT`, statements. Algebraic expressions can expand your options in an input netlist file.

Some uses of algebraic expressions are:

- Parameters:

```
.PARAM x='y+3'
```

- Functions:

```
.PARAM rho(leff,weff)='2+*leff*weff-2u'
```

- Algebra in elements:

```
R1 1 0 r='ABS(v(1)/i(m1))+10'
```

- Algebra in `.MEASURE` statements:

```
.MEAS vmax MAX V(1)
.MEAS imax MAX I(q2)
.MEAS ivmax PARAM='vmax*imax'
```

- Algebra in output statements:

```
.PRINT conductance=PAR('i(m1)/v(22)')
```

The basic syntax for using algebraic expressions for output is:

`PAR('`*`algebraic expression`*`')`

In addition to using quotations, you must define the expression inside the `PAR( )` statement for output.The continuation character for quoted parameter strings, in HSPICE, is a double backslash (\\). (Outside of quoted strings, the single backslash (\) is the continuation character.)

Built-In Functions and Variables

In addition to simple arithmetic operations (+, -, *, /), use the built-in functions listed in Table 19 and the variables listed in Table 18 on page 275 in HSPICE expressions.

Table 19 Synopsys HSPICE Built-in Functions

HSPICE Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x (radians)
cos(x)	cosine	trig	Returns the cosine of x (radians)
tan(x)	tangent	trig	Returns the tangent of x (radians)
asin(x)	arc sine	trig	Returns the inverse sine of x (radians)
acos(x)	arc cosine	trig	Returns the inverse cosine of x (radians)
atan(x)	arc tangent	trig	Returns the inverse tangent of x (radians)
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x (radians)
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic cosine of x (radians)

Table 19 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
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)$\|x\|^y$
x**y	power		If x<0, returns the value of x raised to the integer part of y. If x=0, returns 0. If x>0, returns the value of x raised to the y power.
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(|x|)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)$\log_{10}$(|x|)
exp(x)	exponential	math	Returns e, raised to the power x: e^x
db(x)	decibels	math	Returns the base 10 logarithm of the absolute value of x, multiplied by 20, with the sign of x: (sign of x)20$\log_{10}$(|x|)
int(x)	integer	math	Returns the integer portion of x. The fractional portion of the number is lost.
nint(x)	integer	math	Rounds x up or down, to the nearest integer.

Table 19 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
sgn(x)	return sign	math	Returns -1 if x is less than 0. Returns 0 if x is equal to 0. Returns 1 if x is greater than 0
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y: (sign of y)|x|
def(x)	parameter defined	control	Returns 1 if parameter x is defined. Returns 0 if parameter x is not defined.
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. Not supported for CMI models (Level 54 and greater).
lv(<Element>) or lx(<Element>)	element templates	various	Returns various element values during simulation. See Element Template Output (HSPICE Only) on page 316 for more information.
v(<Node>), i(<Element>)...	circuit output variables	various	Returns various circuit values during simulation. See DC and Transient Output Variables on page 301 for more information.

Table 19 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
<cond> ?x : y	ternary operator		Returns *x* if *cond* is not zero. Otherwise, returns *y*. .param z= ‘condition ? x:y’
<	relational operator (less than)		Returns 1 if the left operand is less than the right operand. Otherwise, returns 0. .para x=y<z (y less than z)
<=	relational operator (less than or equal)		Returns 1 if the left operand is less than or equal to the right operand. Otherwise, returns 0. .para x=y<=z (y less than or equal to z)
>	relational operator (greater than)		Returns 1 if the left operand is greater than the right operand. Otherwise, returns 0. .para x=y>z (y greater than z)
>=	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 20 on page 283 for use in elements, such as E, G, R, C, and L. You can use them in expressions, but you cannot redefine them; for example, this statement would be illegal:

```
.param temper=100
```

Table 20 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. You can use the HSPICE simulation temperature in an equation by using the temper variable parameter. For example: .temp 20 50 100 .par x="temper/2" v0 1 0 1 r0 1 0 r=x
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 subcircuits, you need to create fewer similar cells, to provide enough functionality in your library. You can pass circuit parameters into hierarchical designs, and assign different values to the same parameter within individual cells, when you run simulation.

For example, if you use parameters to set the initial state of a latch in its subcircuit definition, then you can override this initial default in the instance call. You need to create only one cell, to handle both initial state versions of the latch.

You can also use parameters to define the cell layout. For example, you can use parameters in a MOS inverter, to simulate a range of inverter sizes, with only one cell definition. Local instances of the cell can assign different values to the size parameter for the inverter.

In HSPICE, you can also perform Monte Carlo analysis or optimization on a cell that uses parameters.

How you handle hierarchical parameters depends on how you construct and analyze your cells. You can construct a design in which information flows from the top of the design, down into the lowest hierarchical levels.

- To centralize the control at the top of the design hierarchy, set *global* parameters.
- To construct a library of small cells that are individually controlled from within, set *local* parameters and build up to the block level.

This section describes the scope of parameter names, and how HSPICE resolves naming conflicts between levels of hierarchy.

Library Integrity

Integrity is a fundamental requirement for any symbol library. Library integrity can be as simple as a consistent, intuitive name scheme, or as complex as libraries with built-in range checking.

Library integrity might be poor if you use libraries from different vendors in a circuit design. Because names of circuit parameters are not standardized between vendors, two components can include the same parameter name for different functions. For example, one vendor might build a library that uses the name `Tau` as a parameter to control one or more subcircuits in their library. Another vendor might use `Tau` to control a different aspect of their library. If you set a global parameter named `Tau` to control one library, you also modify the behavior of the second library, which might not be the intent.

If the scope of a higher-level parameter is global to all subcircuits at lower levels of the design hierarchy, higher-level definitions override lower-level parameter values with the same names. The scope of a lower-level parameter is local to the subcircuit where you define the parameter (but global to all subcircuits that are even lower in the design hierarchy). Local scoping rules in HSPICE prevent higher-level parameters from overriding lower-level parameters of the same name, when that is not desired.

Reusing Cells

Parameter name problems also occur if different groups collaborate on a design. Global parameters prevail over local parameters, so all circuit designers must know the names of all parameters, even those used in sections of the design for which they are not responsible. This can lead to a large investment in standard libraries. To avoid this situation, use local parameter scoping, to encapsulate all information about a section of a design, within that section.

Creating Parameters in a Library

To ensure that the input netlist includes critical, user-supplied parameters when you run simulation, you can use “illegal defaults”—that is, defaults that cause the simulator to abort if you do not supply overrides for the defaults.

If a library cell includes illegal defaults, you must provide a value for each instance of those cells. If you do not, the simulation aborts.

For example, you might define a default MOSFET width of 0.0. HSPICE aborts because MOSFET models require this parameter.

Example 1

```
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0 $ Inherit illegal values by default
mp1 <NodeList> <Model> L=1u W='Wid*2'
mn1 <NodeList> <Model> L=1u W=Wid
.ENDS

* Invoke symbols in a design
x1 A Y1 Inv        $ Bad! No widths specified
x2 A Y2 Inv Wid=1u $ Overrides illegal value for Width
```

This simulation aborts on the `x1` subcircuit instance because you never set the required `Wid` parameter on the subcircuit instance line. The `x2` subcircuit simulates correctly. Additionally, the instances of the `Inv` cell are subject to accidental interference because the `Wid` global parameter is exposed outside the domain of the library. Anyone can specify an alternative value for the parameter, in another section of the library or the circuit design. This might prevent the simulation from catching the condition on `x1`.

Example 2

In this example, the name of a global parameter conflicts with the internal library parameter named `Wid`. Another user might specify such a global parameter, in a different library. In this example, the user of the library has specified a different meaning for the `Wid` parameter, to define an independent source.

```
.Param Wid=5u         $ Default Pulse Width for source
v1 Pulsed 0 Pulse ( 0v 5v 0u 0.1u 0.1u Wid 10u )
...
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0      $ Inherit illegals by default
mp1 <NodeList> <Model> L=1u W='Wid*2'
mn1 <NodeList> <Model> L=1u W=Wid
.Ends
* Invoke symbols in a design
x1 A Y1 Inv        $ Incorrect width!
x2 A Y2 Inv Wid=1u   $ Incorrect! Both x1 and x2
$ simulate with mp1=10u and
$ mn1=5u instead of 2u and 1u.
```

Under global parameter scoping rules, simulation succeeds, but incorrectly. HSPICE does not warn 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.

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 21 compares the three primary methods for configuring libraries, to achieve required parameter checking for default MOS transistor widths.

Table 21 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.	
Global	At the global level and on .SUBCKT definition lines	Works with all 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 .OPTION DEFW.

String Parameter (HSPICE Only)

HSPICE uses a special delimiter to identify string and double parameter types. The single quotes (‘), double quotes (“), or curly brackets ({}) do not work for these kinds of delimiters. Instead, use the sp1=str('string') keyword for an sp1 parameter definition and use the str(sp1) keyword for a string parameter instance.

Example

The following sample netlist shows an example of how you can use these definitions for various commands, keywords, parameters, and elements:

```
xibis1 vccq vss out in IBIS
+ IBIS_FILE=str('file1.ibs') IBIS_MODEL=str('model1')
xibis2 vccq vss out in IBIS
+ IBIS_FILE=str('file2.ibs') IBIS_MODEL=str('model2')

.subckt IBIS vccq vss out in
+ IBIS_FILE=str('file.ibs')
+ IBIS_MODEL=str('ibis_model')
ven en 0 vcc
BMCH vccq vss out in en v0dq0 vccq vss buffer=3
+ file= str(IBIS_FILE) model=str(IBIS_MODEL)
+ typ=typ ramp_rwf=2 ramp_fwf=2 power=on
.ends
```

HSPICE supports these kinds of definitions and instances with the following netlist components:

- `.PARAM` statements
- `.SUBCKT` statements
- S-parameter `FQMODEL` in both the S-parameter instance and S-parameter model and the `TSTONEFILE` keyword in the S-element
- `FILE` and `MODEL` keywords
- B-elements
- W-element `keywords RLGCFILE`, `UMODEL`, `FSMODEL`, `RLGCMODEL`, `TABLEMODEL`, and `SMODEL`

Note:

The keywords `MNAME` and `CITIFILE` are not supported.

String Parameters in Passive and Active Component Keywords

You can include string parameters in all HSPICE passive and active component model name keywords. When defining a parameter that is a character string, the keyword `str('string')` is used to define the parameter. When an instance of the parameter is used, the parameter name is called as `str(parameter_name).`

Syntax

For passive elements:

```
Rxxx n1 n2 <mname <str(mname)>> Rval <TC1 <TC2><TC>> <SCALE=val>
+ <M=val> <AC=val> <DTEMP=val> <L=val> <W=val> <C=val>
+<NOISE = val>
Cxxx n1 n2 <mname <str(mname)>> <C = >capacitance <<TC1 = >val>
+ <<TC2 = >val> <SCALE = val> <IC = val> <M = val>
+ <W = val> <L = val> <DTEMP = val>
Lxxx n1 n2 <L = >inductance <mname <str(mname)>> <<TC1 = >val>
+ <<TC2 = >val> <SCALE = val> <IC = val> <M = val>
+ <DTEMP = val> <R = val>
```

For active elements, the model name can be defined using the original syntax, or string parameter model name syntax

```
Dxxx nplus nminus str(mname) <<AREA = >area> <<PJ = >val>
+ <WP = val> <LP = val> <WM = val> <LM = val> <OFF>
+ <IC = vd> <M = val> <DTEMP = val>
Qxxx nc nb ne <ns> str(mname) <area> <OFF>
+ <IC = vbeval,vceval> <M = val> <DTEMP = val>
Jxxx nd ng ns <nb> str(mname) <<<AREA> = area | <W = val>
+ <L = val>> <OFF> <IC = vdsval,vgsval> <M = val>
+ <DTEMP = val>
Mxxx nd ng ns <nb> str(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>
```

Example

```
.param mypmos=str('p')
.param mynmos=str('n')
.lib 'ltst.lib' TT

.subckt circuit vout vin vdd nmod=str('nch')pmod=str('pch')
m1 vout vin vdd vdd str(pmod) w=4u l=5u
m2 vout vin 0 0 str(nmod) w=2u l=5u
.ends circuit

x1 vout vin vdd circuit dtemp=11 nmod=str(mynmos)pmod=str(mypmos)
```

Parameter Defaults and Inheritance

Use the `.OPTION PARHIER` parameter to specify scoping rules.

Syntax:

```
.OPTION PARHIER=< GLOBAL | LOCAL >
```

The default setting is GLOBAL.

Example

This example explicitly shows the difference between local and global scoping for using parameters in subcircuits.

The input netlist includes the following:

```
.OPTION parhier=<global | local>
.PARAM DefPwid=1u
.SUBCKT Inv a y DefPwid=2u DefNwid=1u
Mp1 <MosPinList> pMosMod L=1.2u W=DefPwid
Mn1 <MosPinList> nMosMod L=1.2u W=DefNwid
.ENDS
```

Set the .OPTION PARHIER=parameter scoping option to GLOBAL. The netlist also includes the following input statements:

```
xInv0 a y0 Inv            $ override DefPwid default,
$ xInv0.Mp1 width=1u
xInv1 a y1 Inv DefPwid=5u $ override DefPwid=5u,
$ xInv1.Mp1 width=1u

.measure tran Wid0 param='lv2(xInv0.Mp1)' $ lv2 is the
               $ template for
.measure tran Wid1 param='lv2(xInv1.Mp1)' $ the channel
               $ width
           $ 'lv2(xInv1.Mp1)'
.ENDS
```

Simulating this netlist produces the following results in the listing file:

```
wid0=1.0000E-06
wid1=1.0000E-06
```

If you change the .OPTION PARHIER=parameter scoping option to LOCAL:

```
xInv0 a y0 Inv            $ not override .param
  $ DefPwid=2u,
  $ xInv0.Mp1 width=2u
xInv1 a y1 Inv DefPwid=5u      $ override .param
         $ DefPwid=2u,
         $ xInv1.Mp1 width=5u:
.measure tran Wid0 param='lv2(xInv0.Mp1)'$ override the
.measure tran Wid1 param='lv2(xInv1.Mp1)'$ global .PARAM
...
```

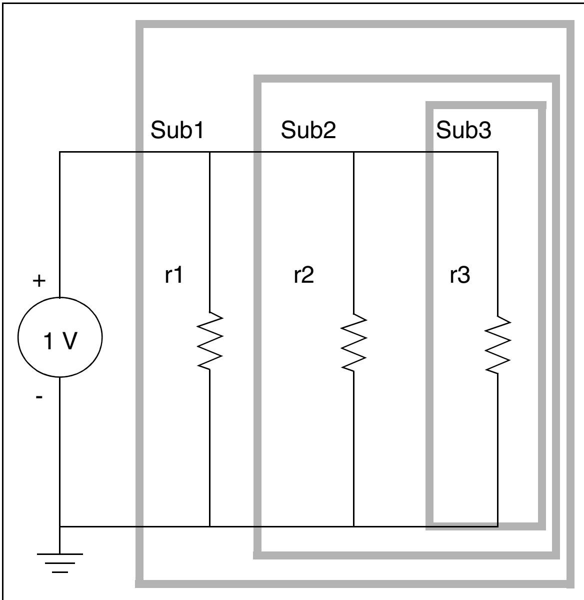
Simulation produces the following results in the listing file:

```
wid0=2.0000E-06
wid1=5.0000E-06
```

Parameter Passing

Figure 26 on page 291 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.



```
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
```

Figure 26 Hierarchical Parameter Passing Problem

The total resistance of the chain has two possible solutions: 0.3333Ω and 0.5455Ω.

You can use `.OPTION PARHIER` to specify which parameter value prevails, when you define parameters with the same name at different levels of the design hierarchy.

Under global scoping rules, if names conflict, the top-level assignment `.PARAM Val=1` overrides the subcircuit defaults, and the total is 0.3333Ω. Under local scoping rules, the lower level assignments prevail, and the total is 0.5455Ω (one, two, and three ohms in parallel).

The example in Figure 26 produces the results in Table 22, based on how you set `.OPTION PARHIER` to local/global:

Table 22 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

The following checklist determines whether you will see simulation differences when you use the default scoping rules. These checks are especially important if your netlists contain devices from multiple vendor libraries.

- Check your subcircuits for parameter defaults, on the `.SUBCKT` or `.MACRO` line.
- Check your subcircuits 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.

11
Simulation Output

Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.

You can also use output variables in behavioral circuit analysis, modeling, and simulation techniques. To display electrical specifications such as rise time, slew rate, amplifier gain, and current density, use the output format features.

For descriptions of individual HSPICE commands referenced in this chapter, see the HSPICE Reference Manual: Commands and Control Options.

These topics are organized in the following sections:

Note:

Parameter Storage Format (PSF) output is supported by all HSPICE analyses in the HSPICE integration to the Cadence^{TM} Virtuoso® Analog Design Environment.

Note that the PSF and SDA writers used in HSPICE rely on libraries that are not currently available in 64 bit versions, and 64 bit HSPICE cannot link the 32-bit libraries. If you inspect the log file, you will see the message "`**warning** 64bit cannot support option psf, artist or sda`".

Overview of Output Statements

Output Commands

The input netlist file contains output statements, including `.PRINT`, `PROBE`, `.MEASURE`, `.DOUT`, and `.STIM`. Each statement specifies the output variables, and the type of simulation result, to display—such as `.DC`, `.AC`, or `.TRAN`. When you specify `.OPTION POST`, HSPICE puts all output variables, referenced in `.PRINT`, `.PROBE`, `.MEASURE`, `.DOUT`, and `.STIM` statements into HSPICE output files.

HSPICE RF supports only `.OPTION POST`, `.OPTION PROBE`, `.PRINT`, `.PROBE`, and `.MEASURE` statements. It does not support `.DOUT` or `.STIM` statements. Refer to the *HSPICE Reference Manual: Commands and Control Options* for information on all listed statements.

CosmosScope provides high-resolution, post-simulation, and interactive display of waveforms.

Table 23 Output Statements

Output Statement	**Description**
.PRINT	Prints numeric analysis results in the output listing file (and post-processor data, if you specify .OPTION POST). See .PRINT.
.PROBE	Outputs data to post-processor output files, but not to the output listing (used with .OPTION PROBE, to limit output). See .PROBE.
.MEASURE	Prints the results of specific user-defined analyses (and post-processor data, if you specify .OPTION POST), to the output listing file or HSPICE RF. See .MEASURE (or) .MEAS.
.DOUT (HSPICE only	Specifies the expected final state of an output signal. See .DOUT or Expected State of Digital Output Signal (.DOUT).

Table 23 Output Statements (Continued)

Output Statement	Description
.STIM (HSPICE only)	Specifies simulation results to transform to PWL, Data Card, or Digital Vector File format. See .STIM.

Output Variables

The output format statements require special output variables, to print or plot analysis results for nodal voltages and branch currents. HSPICE or HSPICE RF uses the following output variables:

- DC and transient analysis
- AC analysis
- element template (HSPICE only)
- `.MEASURE` statement
- parametric analysis

For HSPICE or HSPICE RF, DC and transient analysis displays:

- individual nodal voltages: *V*(*n1* [,*n2*])
- branch currents: *I*(*Vxx*)
- element power dissipation: *In*(*element*)

AC analysis displays imaginary and real components of a nodal voltage or branch current, and the magnitude and phase of a nodal voltage or branch current. AC analysis results also print impedance parameters, and input and output noise.

Element template analysis displays element-specific nodal voltages, branch currents, element parameters, and the derivatives of the element's node voltage, current, or charge.

The `.MEASURE` statement variables define the electrical characteristics to measure in a `.MEASURE` statement analysis.

Parametric analysis variables are mathematical expressions, which operate on nodal voltages, branch currents, element template variables (HSPICE only), or other parameters that you specify. Use these variables when you run behavioral analysis of simulation results. See Using Algebraic Expressions on page 278.

Displaying Simulation Results

The following section describes the statements that you can use to display simulation results for your specific requirements.

.PRINT Statement

The `.PRINT` statement specifies output variables for which HSPICE or HSPICE RF prints values.

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.

HSPICE RF prints the `.PRINT` output data to a separate file.

You can include wildcards in `.PRINT` statements.

You can also use the iall keyword in a `.PRINT` statement, to print all branch currents of all diode, BJT, JFET, or MOSFET elements in your circuit design.

Example

If your circuit contains four MOSFET elements (named m1, m2, m3, m4), then `.PRINT` iall (m*) is equivalent to `.PRINT` i(m1) i(m2) i(m3) i(m4). It prints the output currents of all four MOSFET elements.

Statement Order

HSPICE or HSPICE RF creates different .sw0 and .tr0 files, based on the order of the `.PRINT` and `.DC` statements. If you do not specify an analysis type for a `.PRINT` command, the type matches the last analysis command in the netlist, before the `.PRINT` statement.

.PROBE Statement

HSPICE or HSPICE RF usually saves all voltages, supply currents, and output variables. Set `.OPTION PROBE`, to save output variables only. Use the `.PROBE` statement to specify the quantities to print in the output listing.

If you are interested only in the output data file, and you do not want tabular or plot data in your listing file, set `.OPTION PROBE` and use `.PROBE` to select the values to save in the output listing.

You can include wildcards in `.PROBE` statements.

Using Wildcards in PRINT and PROBE Statements

You can include wildcards in `.PRINT` and `.PROBE` statements. Refer to this example netlist in the discussion that follows:

```
* test wildcard
.option post
v1 1 0 10
r1 1 n20 10
r20 n20 n21 10
r21 n21 0 10
.dc v1 1 10 1
***Wildcard equivalent for:
*.print i(r1) i(r20) i(r21) i(v1)
.print i(*)
***Wildcard equivalent for:
*.probe v(0) v(1)
.probe v(?)
***Wildcard equivalent for:
*.print v(n20) v(n21)
.print v(n2?)
***Wildcard equivalent for:
*.probe v(n20, 1) v(n21, 1)
.probe v(n2*, 1)
.end
```

Supported Wildcard Templates (HSPICE only)

```
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 isub
```

For detailed information about the templates, see `.PRINT` statement (see Selecting Simulation Output Parameters on page 300).

When you use the wildcard `i(*)` in a `.print` or `.probe` statement, HSPICE will output all branch currents.

For `.AC` analysis, to plot all currents for each valid AC output variable type, you can also use the following in statements:

```
im(*) ir(*) ip(*) idb(*) it(*)
```

In the preceding test case (named test wildcard), if you use an `.AC` statement instead of a `.DC` statement, any valid AC output variable types can be used with the wildcards `v(n2?)` and `v(n2*,1)`. For example:

```
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)
```

To output the branch current at all terminals of a diode, BJT, JFET or MOSFET, use the output template iall. For example, iall(m*) is equivalent to:

```
i1(m*)  i2(m*)  i3(m*)  i4(m*)
```

Print Control Options

The codes that you can use to specify the element templates for output in HSPICE or HSPICE RF are:

- `.OPTION INGOLD` for output in exponential form.
- `.OPTION POST` to display plots using an interactive waveform viewer.
- `.OPTION ACCT` to generate a detailed accounting report.

HSPICE supports the following plot file formats: **.tr#*, **.ac#*, and **.sw#*. If a plot fails to open, it is due to one of the following reasons:

- The waveform file format is not supported.
- The file format is not understood.
- The file is not found.
- The file is larger than max size of *(x)*.

Changing the File Descriptor Limit (HSPICE Only)

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`.

Note:

In the following example, you can substitute `.PROBE`, instead of `.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 preceding `.PRINT` statement specifies two ways to print the voltage on the A input of the X1 instance.

```
.PRINT v(3) v(X1.Y)   v(X2.A) $ Print statement 3
```

The preceding `.PRINT` statement specifies three different ways to print the voltage at the Y output of the X1 instance (or the A input of the X2 instance).

```
.PRINT v(X2.N1) $ Print statement 4
```

The preceding `.PRINT` statement prints the voltage on the N1 internal node of the X2 instance.

```
.PRINT i(X1.M1) $ Print statement 5
```

The preceding `.PRINT` statement prints out the drain-to-source current, through the M1 MOSFET in the X1 instance.

Example 2

```
X1 5 6 YYY
 .SUBCKT YYY 15 16
  X2 16 36 ZZZ
  R1 15 25 1
  R2 25 16 1
 .ENDS
 .SUBCKT ZZZ 16 36
  C1 16 0 10P
  R3 36 56 10K
  C2 56 0 1P
 .ENDS
 .PRINT V(X1.25) V(X1.X2.56) V(6)
```

Value	Description
V(X1.25)	Local node to the YYY subcircuit definition, which the X1 subcircuit calls.
V(X1.X2.56)	Local node to the ZZZ subcircuit. The X2 subcircuit calls this node; X1 calls X2.
V(6)	Voltage of node 16, in the X1 instance of the YYY subcircuit.

This example prints voltage analysis results at node 56, within the X2 and X1 subcircuits. The full path, X1.X2.56, specifies that node 56 is within the X2 subcircuit, which in turn is within the X1 subcircuit.

Selecting Simulation Output Parameters

Parameters provide the appropriate simulation output. To define simulation parameters, use the `.OPTION` and `.MEASURE` statements, and define specific variable elements.

DC and Transient Output Variables

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for any element.
- Current output for a subcircuit pin.
- Element templates (HSPICE only). For each device type, the templates contain:
  - values of variables that you set
  - state variables
  - element charges
  - capacitance currents
  - capacitances
  - derivatives

Print Control Options on page 298 summarizes the codes that you can use, to specify the element templates for output in HSPICE or HSPICE RF.

Nodal Capacitance Output

Syntax

Cap(*nxxx*)

For nodal capacitance output, HSPICE prints or plots the capacitance of the specified node nxxxx.

Example

```
.print dc Cap(5) Cap(6)
```

Nodal Voltage

Syntax

```
V(n1<,n2>)
```

Parameter	Description
n1, n2	HSPICE or HSPICE RF prints or plots the voltage difference (*n1*-*n2*) between the specified nodes. If you omit *n2*, HSPICE or HSPICE RF prints or plots the voltage difference between *n1* and ground (node 0).

Current: Independent Voltage Sources

Syntax

```
I(Vxxx)
```

Parameter	Description
Vxxx	Voltage source element name. If an independent power supply is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, I(X*1*.V*xxx*).

Example

```
.PRINT TRAN I(VIN)
.PRINT DC I(X1.VSRC)
.PRINT DC I(XSUB.XSUBSUB.VY)
```

Current: Element Branches

Syntax

```
In(Wwww)
Iall(Wwww)
```

Parameter	Description
n	Node position number, in the element statement. For example, if the element contains four nodes, I3 is the branch current output for the third node. If you do not specify *n*, HSPICE or HSPICE RF assumes the first node.

Parameter	**Description**
Wwww	Element name. To access current output for an element in a subcircuit, append a dot and the subcircuit name to the element name. For example, I3(X1.Wwww).
Iall (Wwww)	An alias just for diode, BJT, JFET, and MOSFET devices. ■ If *Wwww* is a diode, it is equivalent to: I1(Wwww) I2(Wwww). ■ If *Wwww* is one of the other device types, it is equivalent to: I1(Wwww) I2(Wwww) I3(Wwww) I4(Wwww)

Example 1

```
I1(R1)
```

This example specifies the current through the first R1 resistor node.

Example 2

```
I4(X1.M1)
```

This example specifies the current, through the fourth node (the substrate node) of the M1 MOSFET, defined in the X1 subcircuit.

Example 3

```
I2(Q1)
```

The last example specifies the current, through the second node (the base node) of the Q1 bipolar transistor.

To define each branch circuit, use a single element statement. When HSPICE or HSPICE RF evaluates branch currents, it inserts a zero-volt power supply, in series with branch elements.

If HSPICE cannot interpret a `.PRINT` statement that contains a branch current, it generates a warning.

Branch current direction for the elements in Figure 27 through Figure 32 is defined in terms of arrow notation (current direction), and node position number (terminal type).

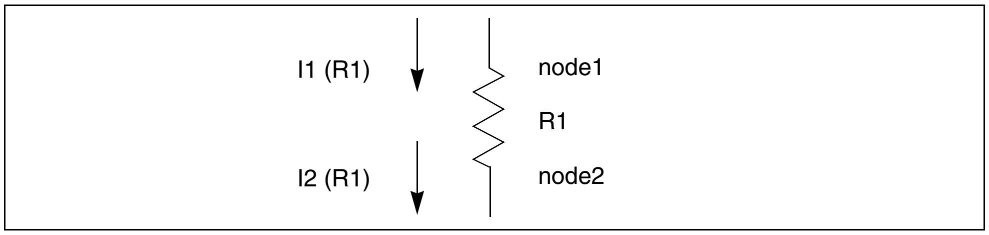


Figure 27 Resistor (node1, node2)

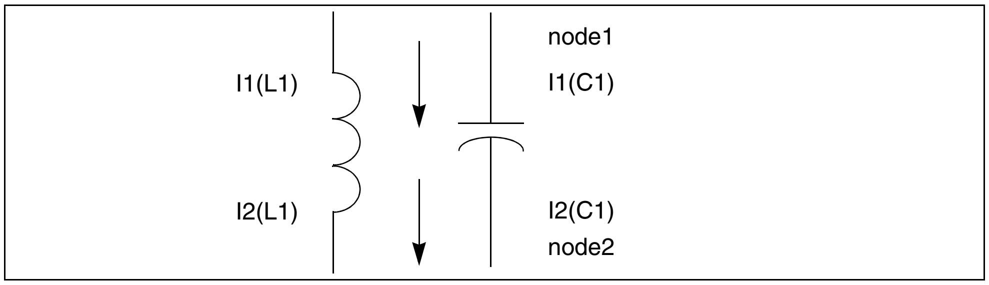


Figure 28 Inductor (node1, node2); capacitor (node 1, node2)

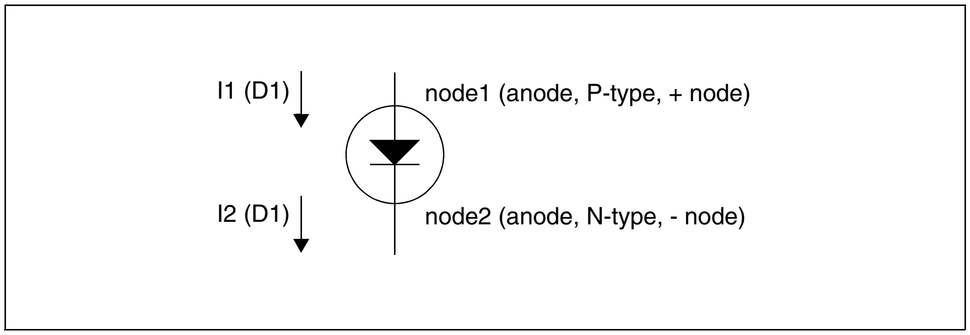


Figure 29 Diode (node1, node2)

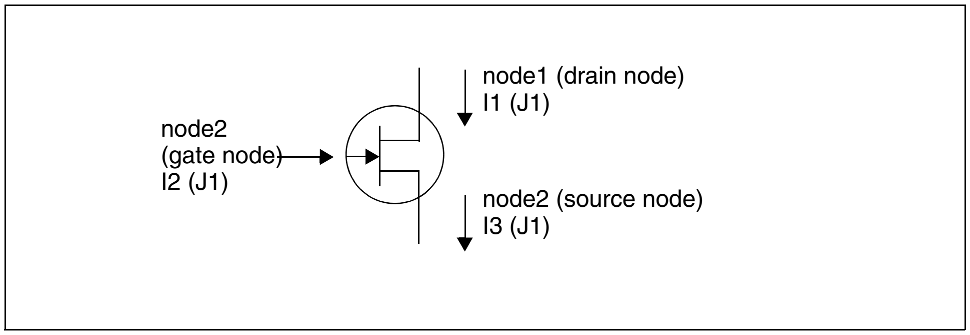


Figure 30   JFET (node1, node2, node3) - n-channel

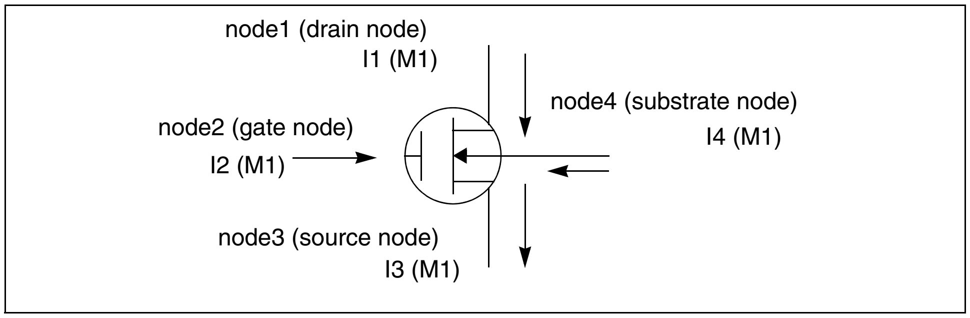


Figure 31   MOSFET (node1, node2, node3, node4) - n-channel

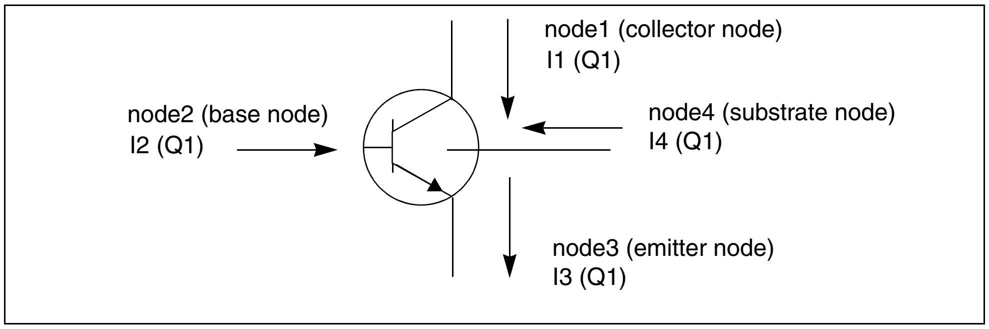


Figure 32   BJT (node1, node2, node3, node4) - npn

Current: Subcircuit Pin

Syntax

```
ISUB(X****.****)
```

Example

```
.PROBE ISUB(X1.PIN1)
```

Power Output

For power calculations, HSPICE or HSPICE RF computes dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H). To compute this power, HSPICE or HSPICE RF multiplies the voltage across an element, and its corresponding branch current.

However, for semiconductor devices, HSPICE or HSPICE RF calculates only the dissipated power. It excludes the power stored in the device junction or parasitic capacitances from the device power computation. The following sections show equations for calculating the power that different types of devices dissipate.

HSPICE or HSPICE RF also computes the total power dissipated in the circuit, which is the sum of the power dissipated in:

- Devices
- Resistors
- Independent current sources
- All dependent sources

For hierarchical designs, HSPICE or HSPICE RF also computes the power dissipation for each subcircuit.

Note:

For the total power (dissipated power + stored power), HSPICE or HSPICE RF does not add the power of each independent source (voltage and current sources).

Wildcard Support

Wildcard support is available for current subcircuit pins in single and multiple hierarchies using asterisk (*) and question mark (?) characters. For example:

Single Hierarchy

```
.print isub(x1.*) isub(x1.a?)
```

Multi-level Hierarchy

```
.print isub(x1.x2.*) isub(x1.x?.a?)
```

Print Power

```
.PRINT <DC | TRAN> P(element_or_subcircuit_name)POWER
```

HSPICE calculates power only for transient and DC sweep analyses. Use the `.MEASURE` statement to compute the average, RMS, minimum, maximum, and peak-to-peak value of the power. The `POWER` keyword invokes the total power dissipation output.

HSPICE RF supports p(instance) but not the `POWER` variable in DC/transient analysis.

Example

```
.PRINT TRAN   P(M1)   P(VIN)   P(CLOAD)   POWER
.PRINT TRAN   P(Q1)   P(DIO)   P(J10)   POWER
.PRINT TRAN   POWER   $ Total transient analysis
* power dissipation
.PRINT DC POWER   P(IIN)   P(RLOAD)   P(R1)
.PRINT DC POWER   P(V1)   P(RLOAD)   P(VS)
.PRINT TRAN P(Xf1) P(Xf1.Xh1)
```

Diode Power Dissipation

$$Pd = Vpp' \cdot (Ido + Icap) + Vp'n \cdot Ido$$

Parameter	Description
Pd	Power dissipated in the diode.
Ido	DC component of the diode current.
Icap	Capacitive component of the diode current.
Vp'n	Voltage across the junction.
Vpp'	Voltage across the series resistance, RS.

BJT Power Dissipation

- Vertical

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vee' \cdot Ietot + Vsc' \cdot Iso - Vcc' \cdot Istot$$

- Lateral

$$\begin{aligned} Pd = {} & Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vbb' \cdot Ibtot + Vee' \cdot Ietot \\ & Vsb' \cdot Iso - Vbb' \cdot Istot \end{aligned}$$

Parameter	Description
Ibo	DC component of the base current.
Ico	DC component of the collector current.
Iso	DC component of the substrate current.
Pd	Power dissipated in a BJT.
Ibtot	Total base current (excluding the substrate current).
Ictot	Total collector current (excluding the substrate current).
Ietot	Total emitter current.
Istot	Total substrate current.
Vb'e'	Voltage across the base-emitter junction.
Vbb'	Voltage across the series base resistance, RB.
Vc'e'	Voltage across the collector-emitter terminals.
Vcc'	Voltage across the series collector resistance, RC.
Vee'	Voltage across the series emitter resistance, RE.
Vsb'	Voltage across the substrate-base junction.
Vsc'	Voltage across the substrate-collector junction.

JFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vgd' \cdot Igdo + Vgs' \cdot Igso + Vs's \cdot (Ido + Igso + Icgs) + Vdd' \cdot (Ido - Igdo - Icgd)$$

Parameter	Description
Icgd	Capacitive component of the gate-drain junction current.
Icgs	Capacitive component of the gate-source junction current.
Ido	DC component of the drain current.
Igdo	DC component of the gate-drain junction current.
Igso	DC component of the gate-source junction current.
Pd	Power dissipated in a JFET.
Vd's'	Voltage across the internal drain-source terminals.
Vdd'	Voltage across the series drain resistance, RD.
Vgd'	Voltage across the gate-drain junction.
Vgs'	Voltage across the gate-source junction.
Vs's	Voltage across the series source resistance, RS.

MOSFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vbd' \cdot Ibdo + Vbs' \cdot Ibso + Vs's \cdot (Ido + Ibso + Icbs + Icgs) + Vdd' \cdot (Ido - Ibdo - Icbd - Icgd)$$

Parameter	Description
Ibdo	DC component of the bulk-drain junction current.
Ibso	DC component of the bulk-source junction current.
Icbd	Capacitive component of the bulk-drain junction current.
Icbs	Capacitive component of the bulk-source junction current.

Parameter	Description
Icgd	Capacitive component of the gate-drain current.
Icgs	Capacitive component of the gate-source current.
Ido	DC component of the drain current.
Pd	Power dissipated in the MOSFET.
Vbd'	Voltage across the bulk-drain junction.
Vbs'	Voltage across the bulk-source junction.
Vd's'	Voltage across the internal drain-source terminals.
Vdd'	Voltage across the series drain resistance, RD.
Vs's	Voltage across the series source resistance, RS.

AC Analysis Output Variables

Output variables for AC analysis include:

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for a subcircuit pin.
- Element branch current.
- Impedance (Z), admittance (Y), hybrid (H), and scattering (S) parameters.
- Input and output impedance, and admittance.

Table 24 lists AC output variable types. In this table, the type symbol is appended to the variable symbol, to form the output variable name. For example, VI is the imaginary part of the voltage, or IM is the magnitude of the current.

Table 24 AC Output Variable Types

Type Symbol	Variable Type
DB	decibel
I	imaginary part
M	magnitude
P	phase
R	real part
T	group delay

Specify real or imaginary parts, magnitude, phase, decibels, and group delay for voltages and currents.

Nodal Capacitance Output

Syntax

```
Cap(nxxx)
```

For nodal capacitance output, HSPICE prints the capacitance of the specified node nxxxx.

Example

```
.print ac Cap(5) Cap(6)
```

Nodal Voltage

Syntax

```
Vz(n1<,n2>)
```

Parameter	Description
z	Specifies the voltage output type (see Table 24 on page 311)
n1, n2	Specifies node names. If you omit *n2*, HSPICE or HSPICE RF assumes ground (node 0).

Example

This example applies to HSPICE, but not HSPICE RF. It prints the magnitude of the AC voltage of node 5, using the VM output variable. HSPICE uses the VDB output variable to print the voltage at node 5, and uses the VP output variable to print the phase of the nodal voltage at node 5.

```
.PRINT AC VM(5) VDB(5) VP(5)
```

HSPICE and SPICE Methods for Producing Complex Results

To produce complex results, an AC analysis uses either the SPICE or HSPICE method, and the `.OPTION ACOUT` control option, to calculate the values of real or imaginary parts for complex voltages of AC analysis, and their magnitude, phase, decibel, and group delay values. The default for HSPICE is `ACOUT=1`. To use the SPICE method, set `ACOUT=0`.

A typical use of the SPICE method is to calculate the nodal vector difference, when comparing adjacent nodes in a circuit. You can use this method to find the phase or magnitude across a capacitor, inductor, or semiconductor device.

Use the HSPICE method to calculate an inter-stage gain in a circuit (such as an amplifier circuit), and to compare its gain, phase, and magnitude.

The following examples define the AC analysis output variables for the HSPICE method, and then for the SPICE method.

HSPICE Method Example:

Real and imaginary:

```
VR(N1,N2)= REAL [V(N1,0)] - REAL [V(N2,0)]
VI(N1,N2)= IMAG [V(N1,0)] - IMAG [V(N2,0)]
```

Magnitude:

$$VM(N1,0)= [VR(N1,0)^2 + VI(N1,0)^2]^{0.5}$$
$$VM(N2,0)= [VR(N2,0)^2 + VI(N2,0)^2]^{0.5}$$
```
VM(N1,N2)= VM(N1,0) - VM(N2,0)
```

Phase:

```
VP(N1,0)= ARCTAN[VI(N1,0)/VR(N1,0)]
VP(N2,0)= ARCTAN[VI(N2,0)/VR(N2,0)]
VP(N1,N2)= VP(N1,0) - VP(N2,0)
```

Decibel:

```
VDB(N1,0)=20 · LOG10[VM(N1,0)]
VDB(N1,N2)= 20 · LOG10(VM(N1,0)/VM(N2,0))
```

SPICE Method Example:

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:

$VDB(N1,N2)=20 \cdot LOG10[VM(N1,N2)]$

Current: Independent Voltage Sources

Syntax

I*z*(*Vxxx*)

Parameter	Description
z	Current output type (see Table 24 on page 311).
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

```
.PRINT AC IR(V1) IM(VN2B) IP(X1.X2.VSRC)
```

Current: Element Branches

Syntax

I*zn*(*Wwww*)

Parameter	Description
z	Current output type (see Table 24 on page 311).

Parameter	Description
n	Node position number, in the element statement. For example, if the element contains four nodes, IM3 denotes the magnitude of the branch current output for the third node.
Wwww	Element name. If the element is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, IM3(X1.Wwww).

```
.PRINT AC IP1(Q5) IM1(Q5) IDB4(X1.M1)
```

If you use the form In(Xxxx) for AC analysis output, then HSPICE or HSPICE RF prints the magnitude value, IMn(Xxxx).

Current: Subcircuit Pin

Syntax

```
ISUB(X****.****)
```

Example

```
.PROBE ISUB(X1.PIN1)
```

Group Time Delay

The TD group time delay is associated with AC analysis. TD is the negative derivative of the phase in radians, with respect to radian frequency. HSPICE or HSPICE RF uses the difference method to compute TD:

$$TD = -\frac{1}{360} \cdot \frac{(phase2 - phase1)}{(f2 - f1)}$$

phase1 and *phase2* are the phases (in degrees) of the specified signal, at the f1 and f2 frequencies (in hertz).

Syntax

```
.PRINT AC VT(10) VT(2,25) IT(RL)
.PRINT 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   AC   1
R1   1   2      2K
C1   2   3      5NF
E3   3   0      2 0 -1000.0
.AC DEC      15   1K   100K
.PRINT AC      VT(3)   (0,4U)   VP(3)
.END
```

Network

Syntax

```
Xij (z), ZIN(z), ZOUT(z), YIN(z), YOUT(z)
```

Parameter	Description
X	Specifies Z (impedance), Y (admittance), H (hybrid), or S (scattering).
ij	i and j can be 1 or 2. They identify the matrix parameter to print.
z	Output type (see Table 24 on page 311). If you omit z, HSPICE or HSPICE RF prints the magnitude of the output variable.
ZIN	Input impedance. For a one-port network, ZIN, Z11, and H11 are the same.
ZOUT	Output impedance.
YIN	Input admittance. For a one-port network, YIN and Y11 are the same.
YOUT	Output admittance.

Example

```
.PRINT   AC   Z11(R)   Z12(R)   Y21(I)   Y22   S11   S11(DB)
.PRINT   AC   ZIN(R)   ZIN(I)   YOUT(M)   YOUT(P)   H11(M)
.PRINT   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)>
```

Parameter	Description
ovar	Noise and distortion analysis parameter. It can be ONOISE (output noise), INOISE (equivalent input noise), or any of the distortion analysis parameters (HD2, HD3, SIM2, DIM2, DIM3).
z	Output type (only for distortion). If you omit z, HSPICE or HSPICE RF outputs the magnitude of the output variable.

Example

```
.PRINT DISTO HD2(M) HD2(DB)
```

Prints the magnitude and decibel values of the second harmonic distortion component, through the load resistor that you specified in the `.DISTO` statement (not shown). You cannot use the `.DISTO` statement in HSPICE RF.

```
.PRINT NOISE INOISE ONOISE
```

Note:

You can specify the noise and distortion output variable, and other AC output variables, in the `.PRINT AC` statements.

Element Template Output (HSPICE Only)

The `.PRINT`, and `.PROBE` 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
```

Parameter	Description
Elname	Name of the element.
Property	Property name of an element, such as a user-input parameter, state variable, stored charge, capacitance current, capacitance, or derivative of a variable.

The alias is:

```
LVnn(Elname)
LXnn(Elname)
```

Parameter	Description
LV	Form to obtain output of user-input parameters, and state variables.
LX	Form to obtain output of stored charges, capacitor currents, capacitances, and derivatives of variables.
nn	Code number for the desired parameter (listed in tables in this section).
Elname	Name of the element.

Example

```
.PRINT 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 or HSPICE RF simulations.

Computing the measurement results is based on postprocessing output. If you use the `INTERP` option to reduce the size of the postprocessing output, then the measurement results can contain interpolation errors. For more information, see .OPTION INTERP in the *HSPICE Reference Manual: Commands and Control Options*.

Fundamental measurement modes in HSPICE are:

- Rise, fall, and delay
- Find-when
- Equation evaluation
- Average, RMS, min, max, and peak-to-peak
- Integral evaluation
- Derivative evaluation
- Relative error

If a `.MEASURE` statement does not execute, then HSPICE or HSPICE RF writes 0.0e0 in the *.mt#* file as the `.MEASURE` result, and writes FAILED in the output listing file. Use `.OPTION MEASFAIL` to write results to the *.mt#*, *.ms#*, or *.ma#* files. For more information, see .OPTION MEASFAIL in the *HSPICE Reference Manual: Commands and Control Options*.

Note:

The *.mt#* format consists of 72 characters in a line and fields that contain 16 characters each.

To control the output variables, listed in `.MEASURE` statements, use the `.PUTMEAS` option. For more information, see the .OPTION PUTMEAS option in the *HSPICE Reference Manual: Commands and Control Options*.

Note:

If a `.measure` statement uses the result of previous `.meas` statement, then the calculation starts when the previous result is found. Until the previous result is found, it outputs zero.

.MEASURE Statement Order

The `.MEASURE` statement matches the last analysis command in the netlist before the `.MEASURE` statement.

Example

```
.tran 20p 1.0n sweep sigma -3 3 0.5
.tran 20p 1.0n sweep monte=20
.meas mover max v(2,1)
```

In this example, `.meas` matches the second `.tran` statement and generates only one measure output file.

.MEASURE Parameter Types

You cannot use measurement parameter results that the `.PARAM` statements in `.SUBCKT` blocks produce, outside of the subcircuit. That is, you cannot pass any measurement parameters defined in `.SUBCKT` statements, as bottom-up parameters in hierarchical designs.

Measurement parameter names must not conflict with standard parameter names. HSPICE or HSPICE RF issues an error message, if it encounters a measurement parameter with the same name as a standard parameter definition.

To prevent `.MEASURE` statement parameters from overwriting parameter values in other statements, HSPICE or HSPICE RF keeps track of parameter types. If you use the same parameter name in both a `.MEASURE` statement and a `.PARAM` statement at the same hierarchical level, simulation terminates and reports an error.

No error occurs if parameter assignments are at different hierarchical levels. `.PRINT` statements that occur at different levels, do not print hierarchical information for parameter name headings.

Example

In HSPICE RF simulation output, you cannot apply `.MEASURE` to waveforms generated from another `.MEASURE` statement in a parameter sweep.

The following example illustrates how HSPICE or HSPICE RF handles `.MEASURE` statement parameters.

```
...
.MEASURE tran length TRIG v(clk) VAL=1.4
+ TD=11ns RISE=1 TARGv(neq) VAL=1.4 TD=11ns
+ RISE=1
.SUBCKT path out in width=0.9u length=600u
+ rm1 in m1 m2mg w='width' l='length/6'
...
.ENDS
```

In the above listing, the *length* in the resistor statement:

```
rm1 in m1 m2mg w='width' l='length/6'
```

does not inherit its value from *length* in the `.MEASURE` statement:

```
.MEASURE tran length ...
```

because they are of different types.

The correct value of l in rm1 should be:

```
l=length/6=100u
```

The value should not be derived from a measured value in transient analysis.

FIND and WHEN Functions

The `FIND` and `WHEN` functions of the `.MEASURE` statement specify to measure:

- Any independent variables (time, frequency, parameter).
- Any dependent variables (voltage or current for example).
- Derivative of a dependent variable, if a specific event occurs.

You can use these measure statements in unity gain frequency or phase measurements. You can also use these statements to measure the time, frequency, or any parameter value:

- When two signals cross each other.
- When a signal crosses a constant value.

The measurement starts after a specified time delay, `TD`. To find a specific event, set `RISE`, `FALL`, or `CROSS` to a value (or parameter), or specify `LAST` for the last event.

`LAST` is a reserved word; you cannot use it as a parameter name in the above measure statements. For definitions of parameters of the measure statement, see Displaying Simulation Results on page 296.

Equation Evaluation

Use the Equation Evaluation form of the `.MEASURE` statement to evaluate an equation, that is a function of the results of previous `.MEASURE` statements. The equation must not be a function of node voltages or branch currents.

The `expression` option is an arithmetic expression that uses results from other prior `.MEASURE` statements. If `equation` or `expression` includes node voltages or branch currents, Unexpected results may incur.

Average, EM_AVG, RMS, MIN, MAX, INTEG, and PP

Average (`AVG`), `RMS`, `MIN`, `MAX`, and peak-to-peak (`PP`) measurement modes report statistical functions of the output variable, rather than analysis values.

- `AVG` calculates the area under an output variable, divided by the periods of interest.
- `RMS` divides the square root of the area under the output variable square, by the period of interest.
- `MIN` reports the minimum value of the output function, over the specified interval.

- `MAX` reports the maximum value of the output function, over the specified interval.
- `PP` (peak-to-peak) reports the maximum value, minus the minimum value, over the specified interval.

  `AVG`, `RMS`, and `INTEG` have no meaning in a DC data sweep, so if you use them, HSPICE or HSPICE RF issues a warning message.
- `EM_AVG` Calculates the average electromigration current. For a symmetric bipolar waveform, the current is:
  I_avg (0, T/2) - R*Iavg (T/2, T), where R is the recovery factor specified using `.option em_recovery`. Wildcards are also supported during this measurement.

Measuring Recovered Electromigration

The `.MEAS` keyword,`EM_AVG`, enables you to calculate “recovered” average current due to electromigration. Recovered average current is specially meaningful for bipolar currents (such as output of the inverter), as the mathematical average for such a waveform will be zero.The keyword uses the From-To measurement function to provide a range to measure. For example:

```
.measure tran em em_avg I(out) from=5n to=50n
```

where `out` is the node at which the measurement is taken.

The example does the following operations:

1. Finds the average of all the positive currents (Ipos_avg) from 5ns to 50ns.
2. Finds the average (absolute value) of all the negative currents (Ineg_avg) from 5ns to 50ns.
3. Does the operation “Ipos_avg - R*Ineg_avg”.

   Where R is a user-provided coefficient using `.option em_recovery=`*`value`*. The default value of `em_recovery` is `1`. See .OPTION EM_RECOVERY in the *HSPICE Reference Manual: Commands and Control Options*.

For this feature HSPICE also supports wildcards (*) during `em_avg` measurement. For example:

```
.meas tran em em_avg I(m*) from=10n to=100n
```

INTEGRAL Function

The `INTEGRAL` function reports the integral of an output variable, over a specified period.

DERIVATIVE Function

The `DERIVATIVE` function provides the derivative of:

- An output variable, at a specified time or frequency.
- Any sweep variable, depending on the type of analysis.
- A specified output variable, when some specific event occurs.

In the following HSPICE RF example, the `SLEW` measurement provides the slope of V(OUT) during the first time, when V(1) is 90% of VDD.

```
.MEAS TRAN SLEW DERIV V(OUT) WHEN V(1)='0.90*VDD'
```

ERROR Function

The relative error function reports the relative difference between two output variables. You can use this format in optimization and curve-fitting of measured data. The relative error format specifies the variable to measure and calculate from the `.PARAM` variable. To calculate the relative error between the two, HSPICE or HSPICE RF uses the `ERR`, `ERR1`, `ERR2`, or `ERR3` function. With this format, you can specify a group of parameters to vary, to match the calculated value and the measured data.

Error Equations

ERR

1. *ERR* sums the squares of (M-C)/max (`M`, `MINVAL`) for each point.
2. It then divides by the number of points.
3. Finally, it calculates the square root of the result.
   - M (meas_var) is the measured value of the device or circuit response.
   - C (calc_var) is the calculated value of the device or circuit response.
   - NPTS is the number of data points.

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} \left(\frac{M_i - C_i}{\max(MINVAL, M_i)}\right)^2\right]^{1/2}$$

ERR1

`ERR1` computes the relative error at each point. For `NPTS` points, HSPICE or HSPICE RF calculates `NPTS ERR1` error functions. For device characterization, the `ERR1` approach is more efficient than the other error functions (`ERR`, `ERR2`, `ERR3`).

$$ERR1_i = \frac{M_i - C_i}{\max(MINVAL, M_i)}$$, i=1,NPTS

HSPICE or HSPICE RF does not print out each calculated `ERR1` value. When you set the `ERR1` option, HSPICE or HSPICE RF calculates an `ERR` value, as follows:

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR1_i^2\right]^{1/2}$$

ERR2

This option computes the absolute relative error, at each point. For `NPTS` points, HSPICE or HSPICE RF calls `NPTS` error functions.

$$ERR2_i = \left|\frac{M_i - C_i}{\max(MINVAL, M_i)}\right|$$, i=1,NPTS

The returned value printed for `ERR2` is:

$$ERR = \frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR2_i$$

ERR3

$$ERR3_i = \frac{\pm\log\left|\frac{M_i}{C_i}\right|}{\left|\log[\max(MINVAL, |M_i|)]\right|}$$, i=1,NPTS

The + and - signs correspond to a positive and negative `M/C` ratio.

Note:

If the `M` measured value is less than `MINVAL`, HSPICE or HSPICE RF uses `MINVAL` instead. If the absolute value of `M` is less than the `IGNOR` or `YMIN` value, or greater than the `YMAX` value, the error calculation does not consider this point.

Expected State of Digital Output Signal (.DOUT)

The digital output (`.DOUT`) statement specifies the expected final state of an output signal (HSPICE only). During simulation, HSPICE compares the simulated results with the expected output vector. An error results if states are different. The `.DOUT` statement uses either of two syntaxes. In both syntaxes, the time and state parameters define the expected output of the *nd* node.

- The first syntax specifies a single threshold voltage, VTH. Any voltage level above VTH is high; any level below VTH is low.

  ```
  .DOUT nd VTH (time state time state)
  ```

  where:

  `nd` is the node name

  `VTH` is the single voltage threshold

  `time` is an absolute time-point

  `state` is one of the following expected conditions of the nd node, at the specified time:

`0` expect	ZERO, LOW
`1` expect	ONE, HIGH
`else`	Don’t care

- The second syntax defines a threshold for both a logic high (`VHI`) and low (`VLO`).

  ```
  .DOUT nd VLO VHI (time state <time state>)
  ```

where:

`nd` is the node name

`VLO` is the voltage of the logic low state

`VHI` is the voltage of the logic high state

`time` is an absolute time-point

`state` is one of the following expected conditions of the nd node, at the specified time:

`0`	expect ZERO, LOW
`1`	expect ONE, HIGH
`else`	Don't care

Note:

If you specify both syntaxes (VTH, plus VHI and VLO), then HSPICE processes only `VTH`, and ignores `VHI` and `VLO`.

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.

The legal values for `state` are:

- `0` Expect ZERO.
- `1` Expect ONE.
- `X, x` Do not care.
- `U, u` Do not care
- `Z, z` Expect HIGH IMPEDANCE (do not care).

Example

The `.PARAM` statement in the following example sets the value of the `VTH` variable 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 considered 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 (HSPICE Only)

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. To control the precision and data format, you can use the same options as you would in a normal simulation. For example:

```
.option numdgt=6    $ sets precision, range is 0 to 10, numdgt=4
        is the default
.option ingold=0    $ sets format, 0=eng 1=combined 2=exponential
```

Note that these settings these affect how data is printed out for your entire testcase. There is no way to only affect the `.STIM` command because the output of the `.STIM` command is taken from the simulation data.

For the syntax and description of the `.STIM` statement, see the .STIM command in the *HSPICE Reference Manual: Commands and Control Options.*

Reusing the PAR(...) output as input to other elements

You can use the par(...) output as the input voltage to another elements.

For example:

```
.print tran v(5) par('5*cos(6.28*v(10)*v(5)*k/360)')
```

You can use the signal either as the input to another element in the same simulation (see Example 1), or you can save the signal and use it as input in another simulation (see Examples 2 and 3). The definitions in print or probe output statements can only be used in output statements. They cannot be referred by any other definitions.You can use the E-element:

```
e1 in 0 vol='5*cos(6.28*v(10)*v(5)*k/360)'
```

Then, the node 'in' can be used as the input to the other element in the same netlist, as shown in this example:

```
M1 dr in src subr pch w=2u l=1u
.subckt inv vdd 0 A B
   M1  A B  vdd  vdd  pch  w=6u  l=1u
   M2  A B  0    0    nch  w=3u  l=1u
   .ends inv
   x1 vdd  0  in out inv
```

You can use `.STIM` statements to save the signal from the first simulation in order to create a PWL source.

```
.stim tran pwl filename=test1 vsrc[0]=v(in) node1=A node2=B
from=0.0ns to=10ns
+ npoints=100
.stim tran pwl filename=test2 vsrc[1]=v(in) node1=C node2=D
from=0.0ns to=10ns
+ npoints=50
```

In this example, the .STIM command creates two stimulus files named test1.pwl0_tr0 and test2.pwl2_tr0. Each has a voltage source: one namedvsrc[0], applied between nodes A and B, and one named vsrc[1], applied between nodes C and D. The stimulus files have a PWL source function based on the voltage of node 'in' during the time 0.0ns to 10ns with 50 points (for vsrc[0]) and 100 points (for vsrc[1]).

Contents of the test1.pwl0_tr0 file:

```
vsrc[0] A B PWL
+  0.           5.0000
+  200.00000p   2.2114
+  400.00000p   2.4666
+  600.00000p   -362.1421m
 .........................
```

Contents of the test2.pwl1_tr0 file:

```
vsrc[1] C D PWL
+  0.            5.0000
+  100.00000p   -1.8008
+  200.00000p   -3.2748
+  300.00000p   -1.3264
.........................
```

The PWL files generated from the .STIM commands can be used as inputs to another simulation.

Load the signal in CosmosScope and then export the (x,y) data of the signal in text format by selecting File > Save > Plotfiles. You will have to edit the data so it looks like a PWL source by adding a source definition and line continuation characters.

Output Files

The `.STIM` statement generates the following output files:

Output File Type	Extension
PWL Source	`.pwl$_tr#`The .STIM statement writes PWL source results to output_file.pwl$_tr#. This output file results from a `.STIM` <tran> pwl statement in the input file.
Data Card	`.dat$_tr#, .dat$_ac#, or .dat$_sw#`The .STIM statement writes DATA Card results to output_file.dat$_sw#, output_file.dat$_ac#, or output_file.dat$_tr#. This output file is the result of a .stim <tran| ac|dc> data statement in the input file.
Digital Vector File	`.vec$_tr#`The .STIM statement writes Digital Vector File results to output_file.vec$_tr#. This output file is the result of a .stim <tran> vec statement in the input file.

Symbol	Description
tr | ac | sw	▪ tr=transient analysis. ▪ ac=AC analysis. ▪ sw=DC sweep analysis.
#	Either a sweep number, or a hard-copy file number. For a single sweep run, the default number is `0`.
$	Serial number of the current .STIM statement, within statements of the same stimulus type (pwl, data, or vec). $=0 ~ n-1 (n is the number of the .STIM statement of that type). The initial $ value is 0. For example, if you specify three .STIM pwl statements, HSPICE generates three PWL output files, with the suffix names pwl0_tr#, pwl1_tr#, and pwl2_tr#.

Element Template Listings (HSPICE Only)

A full and extensive listing of MOSFET output templates, is found in the *HSPICE Reference Manual: MOSFET Models,* MOSFET Output Templates.

Table 25 Resistor (R Element)

Name	**Alias**	**Description**
G	LV1	Conductance at analysis temperature.
R	LV2	Resistance at analysis temperature.
TC1	LV3	First temperature coefficient.
TC2	LV4	Second temperature coefficient.

Table 26 Capacitor (C Element)

Name	**Alias**	**Description**
CEFF	LV1	Computed effective capacitance.
IC	LV2	Initial condition.
Q	LX0	Charge, stored in capacitor.
CURR	LX1	Current, flowing through capacitor.
VOLT	LX2	Voltage, across capacitor.

Table 27 Inductor (L Element)

Name	**Alias**	**Description**
LEFF	LV1	Computed effective inductance.
IC	LV2	Initial condition.
FLUX	LX0	Flux, in the inductor.
VOLT	LX1	Voltage, across inductor.

Table 27 Inductor (L Element) (Continued)

Name	Alias	Description
CURR	LX2	Current, flowing through inductor.

Table 28 Mutual Inductor (K Element)

Name	Alias	Description
K	LV1	Mutual inductance.

Table 29 Voltage-Controlled Current Source (G Element)

Name	Alias	Description
CURR	LX0	Current, through the source, if VCCS.
R	LX0	Resistance value, if VCR.
C	LX0	Capacitance value, if VCCAP.
CV	LX1	Controlling voltage.
CQ	LX1	Capacitance charge, if VCCAP.
DI	LX2	Derivative of the source current, relative to the control voltage.
ICAP	LX2	Capacitance current, if VCCAP.
VCAP	LX3	Voltage, across capacitance, if VCCAP.

Table 30 Voltage-Controlled Voltage Source (E Element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Current, through source.
CV	LX2	Controlling voltage.

Table 30 Voltage-Controlled Voltage Source (E Element) (Continued)

Name	Alias	Description
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 31 Current-Controlled Current Source (F Element)

Name	Alias	Description
CURR	LX0	Current, through source.
CI	LX1	Controlling current.
DI	LX2	Derivative of the source current, relative to the control current.

Table 32 Current-Controlled Voltage Source (H Element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Source current.
CI	LX2	Controlling current.
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 33 Independent Voltage Source (V Element)

Name	Alias	Description
VOLT	LV1	DC/transient voltage.
VOLTM	LV2	AC voltage magnitude.
VOLTP	LV3	AC voltage phase.

Table 34 Independent Current Source (I Element)

Name	Alias	Description
CURR	LV1	DC/transient current.
CURRM	LV2	AC current magnitude.
CURRP	LV3	AC current phase.

Table 35 Diode (D Element)

Name	Alias	Description
AREA	LV1	Diode area factor.
AREAX	LV23	Area, after scaling.
IC	LV2	Initial voltage, across diode.
VD	LX0	Voltage, across diode (VD), excluding RS (series resistance).
IDC	LX1	DC current, through diode (ID), excluding RS. Total diode current is the sum of IDC and ICAP.
GD	LX2	Equivalent conductance (GD).
QD	LX3	Charge of diode capacitor (QD).
ICAP	LX4	Current, through the diode capacitor. Total diode current is the sum of IDC and ICAP.
C	LX5	Total diode capacitance.
PID	LX7	Photo current, in diode.

Table 36 BJT (Q Element)

Name	Alias	Description
AREA	LV1	Area factor.

Table 36 BJT (Q Element) (Continued)

Name	Alias	Description
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.
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).

Table 36 BJT (Q Element) (Continued)

Name	Alias	Description
GPI	LX4	gπ=^{1}ib /^{1}vbe, constant vbc.
GU	LX5	gμ=^{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).
CAP_BE	LX19	cbe capacitance (Cπ).
CAP_IBC	LX20	cbc internal base-collector capacitance (Cμ).
CAP_SCB	LX21	csc substrate-collector capacitance for vertical transistors. csb substrate-base capacitance for lateral transistors.
CAP_XBC	LX22	cbcx external base-collector capacitance.
CMCMO	LX23	^{1}(TF*IBE) /^{1}vbc.
VSUB	LX24	Substrate voltage.

Table 37 JFET (J Element)

Name	Alias	Description
AREA	LV1	JFET area factor.
VDS	LV2	Initial condition for drain-source voltage.
VGS	LV3	Initial condition for gate-source voltage.
PIGD	LV16	Photo current, gate-drain in JFET.
PIGS	LV17	Photo current, gate-source in JFET.
VGS	LX0	VGS.
VGD	LX1	Gate-drain voltage (VGD).
CGSO	LX2	Gate-to-source (CGSO).
CDO	LX3	Drain current (CDO).
CGDO	LX4	Gate-to-drain current (CGDO).
GMO	LX5	Transconductance (GMO).
GDSO	LX6	Drain-source transconductance (GDSO).
GGSO	LX7	Gate-source transconductance (GGSO).
GGDO	LX8	Gate-drain transconductance (GGDO).
QGS	LX9	Gate-source charge (QGS).
CQGS	LX10	Gate-source charge current (CQGS).
QGD	LX11	Gate-drain charge (QGD).
CQGD	LX12	Gate-drain charge current (CQGD).
CAP_GS	LX13	Gate-source capacitance.
CAP_GD	LX14	Gate-drain capacitance.
QDS	LX16	Drain-source charge (QDS).

Table 37 JFET (J Element) (Continued)

Name	Alias	Description
CQDS	LX17	Drain-source charge current (CQDS).
GMBS	LX18	Drain-body (backgate) transconductance (GMBS).

Table 38 Saturable Core Element (K Element)

Name	Alias	Description
MU	LX0	Dynamic permeability (mu), Weber/(amp-turn-meter).
H	LX1	Magnetizing force (H), Ampere-turns/meter.
B	LX2	Magnetic flux density (B), Webers/meter^2.

Table 39 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).

Redirecting the Simulation Output Results Files to a Different Directory

If you need to I redirect the simulation output result files to a directory other than the current working directory., use either of the following two options. At a a command line prompt, enter either:

```
% hspice -i test.sp -o /root/user/hspice/result/test.lis
```

HSPICE redirects the simulation results to the specified location “/root/user/hspice/result”.

or

```
% hspice -i test.sp -o results/test.lis
```

In the second example, HSPICE redirects the simulation results to the

specified folder with respect to the current working directory. However, the destination folder should be created before starting the simulation. Otherwise, HSPICE returns an error message and aborts.

Using the HSPICE Output Converter Utility

This section describes how to convert output generated by HSPICE.

The converter utility is a post-process tool used to convert the output files (*.tr#, *.ac#, and *.sw#) generated by HSPICE. You can use converter to get the Parameter Storage Format (PSF) output files directly from the .tr#, .ac#, or .sw# files generated by HSPICE with the `POST` output control option. You can also use the converter to get the PWL Source, DATA Card, and Digital Vector File (VEC) from the *.tr#, *.ac#, and *.sw# files generated by HSPICE with the `POST` or `CSDF` control options. You can reuse these stimuli in a new simulation.

Note:

The converter utility is not supported in HSPICE RF.

Converter Features

The converter utility converts *.tr#, *.ac#, and *.sw# files to PSF and PWL/DATA/VEC files, which are described in the following sections.

PSF Converter

The PSF Converter runs on the following platforms:

- Sun/Solaris 2.5 and 2.8
- HP/UX10.20 and 11.0
- IBM AIX5.1

Syntax

```
converter -t PSF -i input_file <-o output_file> <-a/-b>
```

Table 40 PSV Converter Parameters

Argument	Description
-t	Specifies the file type (must be psf).
-i	Specifies input file name. The input file must be the output file generated by HSPICE with the POST output control option.
-o	Specifies output file name. The converter assigns a .psf as the extension of the output file. If you do not specify the output file name, the converter appends _psf to the root name of the input file, and it remains the extension of the input file.
-a	Specifies the ASCII format for the output file.
-b	Specifies the binary format for the output file. By default, the output file is in binary format. The content included in angled brackets (< >) is optional.

Example

```
converter -t PSF -i testpost.tr0 -o testpsf
```

The input file is `testpost.tr0`, which HSPICE generates with the POST option. The output file name is `testpsf`. After running, HSPICE generates two new files: testpsf.psf and logFile. The testpsf.psf file is a PSF file that you can view with the Analog Waveform Display (AWD). The logFile is necessary for the AWD to load the waveform.

PWL/DATA/VEC Converter

The PWL/DATA/VEC Converter runs on the following platforms:

- Sun/Solaris 2.5 and 2.8
- HP/UX10.20 and 11.0
- IBM AIX5.1
- DEC Alpha

- PC Linux/RedHat 6.2, 7.0, 7.1 and 7.2
- PC Windows NT4.0, 2000, and XP-Professional

The PWL/DATA/VEC Converter is mainly for reusing previous simulation results directly from the *.tr#, *.ac#, and *.sw# files produced by HSPICE. The converter is in accordance with the `.STIM` statement in the HSPICE netlist.

Syntax

```
converter -t PWL/DATA/VEC -i input_file <-o output_file>
```

Table 41 PWL/DATA/VEC Converter Parameters

Argument	Description
-t	Specifies the type of the stimulus (PWL).
-i	Specifies the input file name. The input file are the output files generated by HSPICE with the POST=x or CSDF=x output control options.
-o	Specifies the output file name. If you do not specify the output file name, the converter automatically assigns the following file names: ■ input_filename.tr#_PWL# ■ input_filename.ac#_DAT# ■ input_filename.tr#_DAT# ■ input_filename.sw#_DAT# ■ input_filename.tr#_VEC# The content included by angled brackets (< >) is optional.

Note:

For PWL and VEC, the input file must be generated by transient analysis.

Prompt User Mode

The PWL/DATA/VEC Converter is a prompt user mode. The converter displays corresponding prompts and asks you to input some data after you start it successfully.

Input the following at the command line and press the Enter key:

```
Converter -t PWL -i sample.tr0
```

The following input prompts appear one at a time and require your specified entries on the command line.

1. `Enter the number of output variables(>0):`

Specify the number of output variables from the waveform file to be converted.

2. `Enter output variables reused:`

   Specify the name of the node(s) in the design to be converted. The node names must match a node name in the waveform file that you are converting.

3. `Enter name of the source (optional):`

   If no source is specified, the source name will be vm<node_name>.

4. `Enter positive node name (optional):`

   If no positive node is specified, the positive node name will be the same as node name(s) specified for the output variable(s).

5. `Enter negative node name (optional):`

   If no negative node is specified, 0 (ground) will be used as the negative node for each node name specified.

6. `Enter independent variable type [1--from/to, 2--dispersed]:`

   This input line is optional. If nothing is entered and you press the Enter key, the input prompts end, the executable automatically runs and generates the <*design_name*>.tr0_PWL0 stimuli file that will contain all time points from the original waveform file.

   If an independent variable type is specified, the following prompts are shown according to the type. A value is required for each prompt.

7. `For 1-- from/to,`

   `Enter the start point:`

   Starting point of the output file.

8. `Enter the end point:`

   Ending point of the output file.

9. `Enter the number of output points:`

   Number of output points.

10. `For 2-- :dispersed`

    `Enter the dispersed points:`

    Enter a list of time points that will be written to the file.

Once you enter the necessary information at the last prompt and press the Return key, the executable automatically runs and generates the *design_name*.tr0_PWL0 stimuli files.

Input Line Dependencies

The input lines you use must adhere to the following conditions:

- Variables used in a PWL source must be voltage or current signals.
- Variables used in a VEC file can only be voltage signals.
- PWL Source Names must begin with V or I.
- Dispersed time points must be increasing in value when the stimulus type is PWL or VEC.
- For the optional items, you can enter the Return Key directly to adopt the default value.

Running the Converter Utility in Batch Mode

While the converter is interactive, prompting you with a series of questions, it can be run in batch mode by redirecting input from an “answer” file.

The command to run the converter in batch mode has two parts and requires two files. The first file (see the following batch file), invokes the converter and tells it the waveform file to use. The second file is the “answer file” containing the answers to the conversion questions. The questions asked by the converter utility are listed in the section titled Prompt User Mode. You can create sample batch files using the following syntax:

```
converter -t pwl -i file1.tr0 < answers1.txt
converter -t pwl -i file2.tr0 < answers2.txt
```

where, `file1.tr0` and `file2.tr0` are HSPICE generated transient output files. The above will create *file1.tr0_PWL0* and *file2.tr0_PWL0*.

Examples of input files with answers:

```
//  single PWL created :
      1          // # of signals
      v(nd)      // names of signals
      vsig1      // name of PWL source
      sig1       // + node of PWL
      0          // - node of PWL
      1          // choose 1 for from/to , 2 to define each point
      0          // start time
      600p       // end time
      100        // # of points
//  Answer file to create multiple PWL signals in one tr0_PWL0 file
//  after first answer, an answer is needed for each
//  signal even if they are the same
             2
             v(sig1) v(sig2)
             vsig1 vsig2
             n1 n2
             0 0
             1 1
             0 0
             100n 100n
             100 100
```

Troubleshooting Issues

Resolving Inductor/Voltage Source Loop Errors

HSPICE issues an inductor/voltage source loop error when:

- Two or more voltage sources are connected to the same nodes.
- A voltage source with an inductor is connected directly across its nodes.
- Two or more inductors are connected in a loop and the current is not limited.

The use of these topologies should be avoided.

However, if HSPICE reports this error, then follow these steps to correct the error:

1. Find out where the topology exists and correct it.
2. Combine multiple voltage sources into a single equivalent voltage source.
3. Limit the current by connecting a small series resistance (1n ohm or smaller) to the voltage source loop.

Voltage Source Missing Rising and Falling Edges

If you have defined rise and fall times in an independent voltage source, and the rise and fall times are missing when you look at the waveform of the source.

The source is defined as:

```
V1 in 0 pulse 0 5 10n 1n 1n 200n 333n
.option post=2
```

See check.jpg for the resulting waveform.

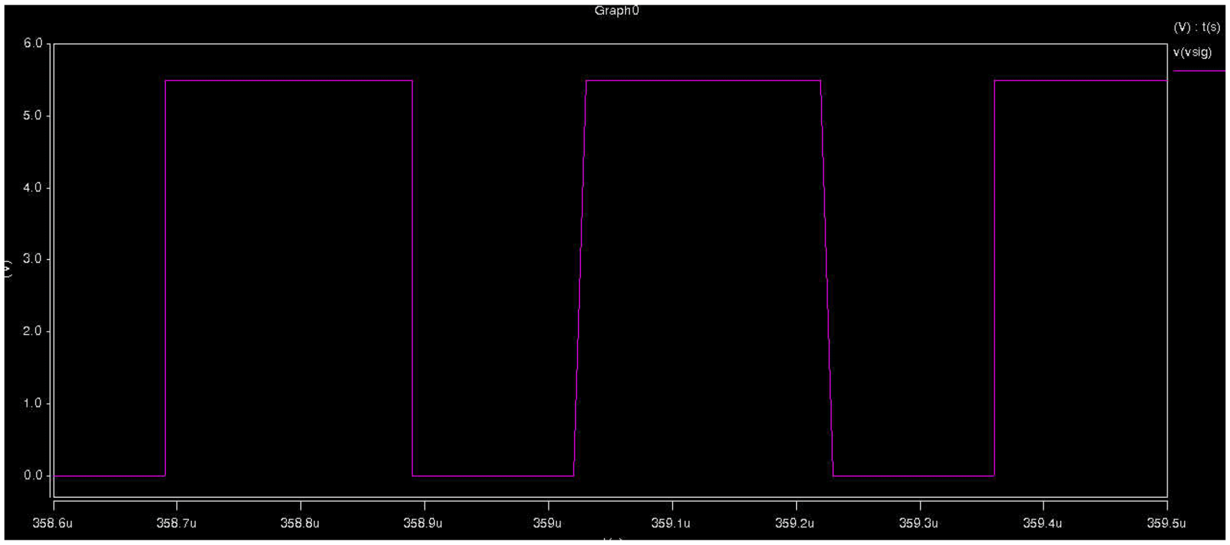


Figure 33

When you set `.option POST=2,` HSPICE prints the waveform file as ASCII data.

When used with the default post-processing output version, `POST_VERSION=9601`, the number of significant digits is limited. This can cause a loss of resolution in the waveforms.

If you set .option `POST_VERSION=2001` in addition to .option `POST=2`, then the ASCII waveform data will contain more significant digits and the resolution will be increased and the rising and falling edges will be present (see check1.jpg).

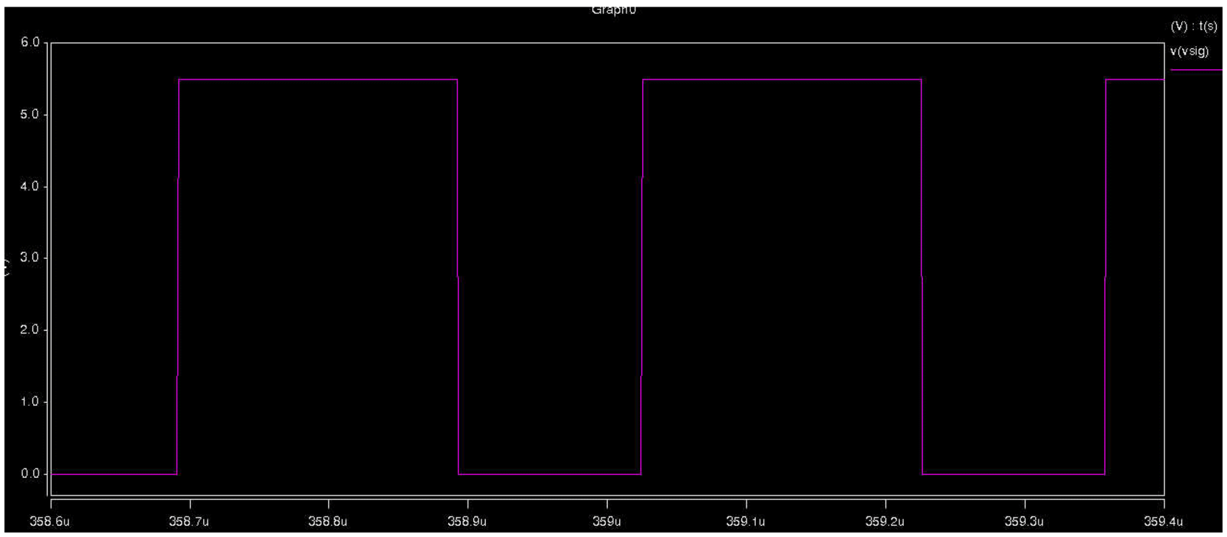


Figure 34

Using `POST_VERSION=2001` will also cause the output file header to display the correct number of output variables when the number of output variables exceeds

9999.

Part: 3 Analyses

This Part contains the following chapters/topics:

12

Initializing DC/Operating Point Analysis

Describes DC initialization and operating point analysis.

For descriptions of individual HSPICE commands referenced in this chapter, see the HSPICE Reference Manual: Commands and Control Options.

These topics are covered in the following sections:

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.

The following describes how this is done:

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 the seed value for the iterative dc algorithm for 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.

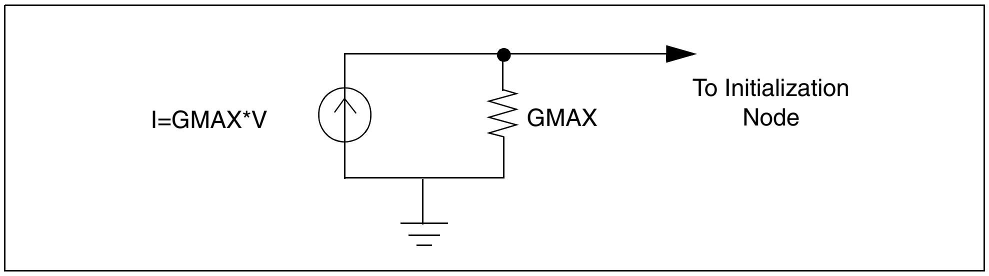


Figure 35 Equivalent Voltage Source: NODESET and .IC

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.

`.IC` and `.NODESET` statements can be used in a `.DC` analysis, in addition to `.TRAN` statements, unless `.OPTION DCIC=0` is set. 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, 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.

Figure 36 shows the simulation flow for DC analysis in HSPICE.

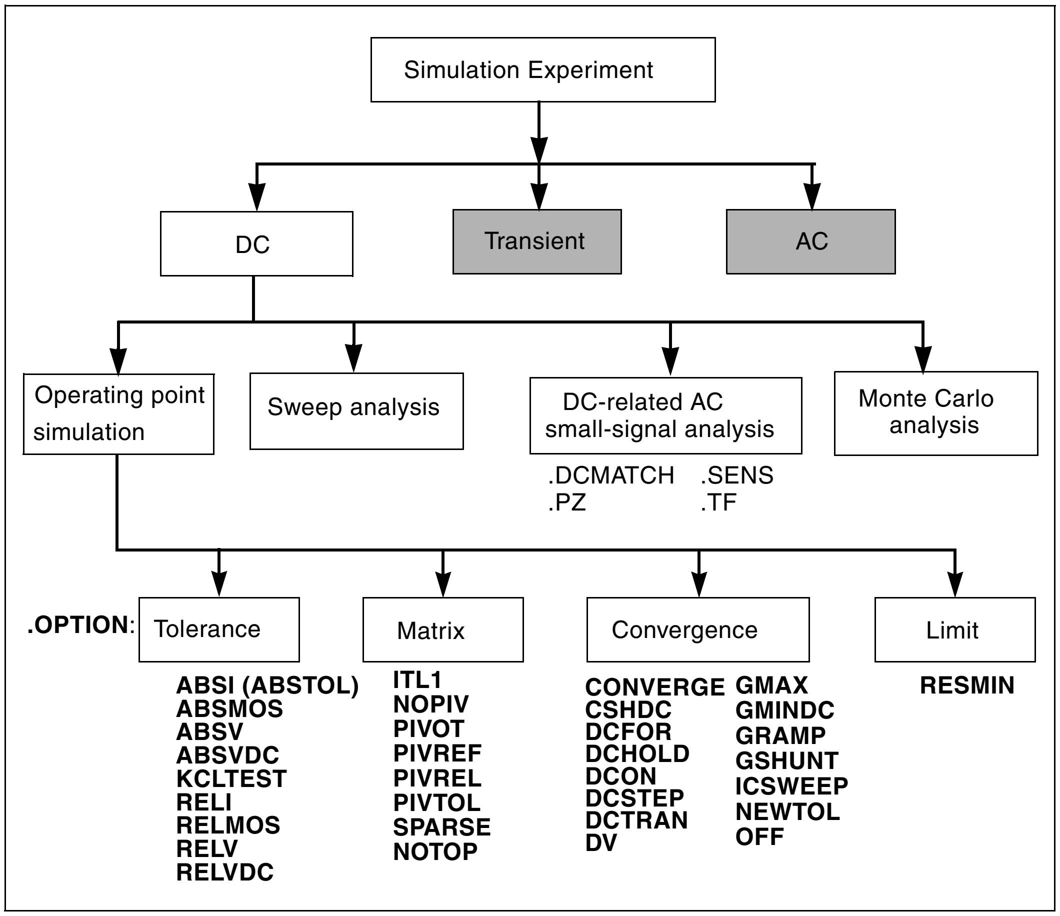


Figure 36 DC Initialization and Operating Point Analysis Simulation Flow

DC Initialization and Operating Point Calculation

Use a `.OP` statement in HSPICE to:

- Calculate the DC operating point of a circuit
- Produce an operating point during a transient analysis

A simulation can only have one `.OP` statement.

.OP Statement — Operating Point

When you include an `.OP` statement in an input file, HSPICE calculates the DC operating point of the circuit. You can also use the `.OP` statement to produce an operating point, during a transient analysis.

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.

Output

```
***** OPERATING POINT INFORMATION  TNOM=25.000 TEMP=25.000
***** OPERATING POINT STATUS IS ALL  SIMULATION TIME IS 0.
NODE     VOLTAGE  NODE     VOLTAGE  NODE      VOLTAGE
+ 0:2=0  0:3=437.3258M  0:4=455.1343M
+ 0:5=478.6763M  0:6=496.4858M  0:7=537.8452M
+ 0:8=555.6659M  0:10=5.0000  0:11=234.3306M

 **** VOLTAGE SOURCES
SUBCKT
ELEMENT  0:VNCE  0:VN7  0:VPCE  0:VP7
VOLTS  0  5.00000  0  -5.00000
AMPS  -2.07407U  -405.41294P  2.07407U  405.41294P
POWER  0.  2.02706N  0.   2.02706N
 TOTAL VOLTAGE SOURCE POWER DISSIPATION=4.0541 N WATTS
**** BIPOLAR JUNCTION TRANSISTORS
SUBCKT
  ELEMENT   0:QN1  0:QN2   0:QN3  0:QN4
* Note: HSPICE RF does not support qn(element)
* charge output.
  MODEL      0:N1     0:N1    0:N1    0:N1
  IB  999.99912N  2.00000U  5.00000U  10.00000U
  IC  -987.65345N  -1.97530U  -4.93827U  -9.87654U
  VBE  437.32588M  455.13437M  478.67632M  496.48580M
  VCE  437.32588M  17.80849M  23.54195M  17.80948M
  VBC  437.32588M  455.13437M  478.67632M  496.48580M
  VS  0.  0.  0.  0.
  POWER  5.39908N  875.09107N  2.27712U  4.78896U
  BETAD -987.65432M -987.65432M -987.65432M -987.65432M
  GM  0.  0.  0.  0.
  RPI  2.0810E+06  1.0405E+06  416.20796K  208.10396K
  RX  250.00000M   250.00000M  250.00000M  250.00000M
  RO  2.0810E+06  1.0405E+06  416.20796K  208.10396K
  CPI  1.43092N  1.44033N  1.45279N  1.46225N
  CMU  954.16927P  960.66843P  969.64689P  977.06866P
  CCS  800.00000P  800.00000P  800.00000P  800.00000P
  BETAAC  0.  0.  0.  0.
  FT   0.  0.  0.  0.
```

Element Statement IC Parameter

Use the element statement parameter, `IC=<val>`, to set DC terminal voltages for selected active devices.

HSPICE uses the value, set in `IC=<val>`, as the DC operating point value.

Example

This example describes an H element dependent-voltage source:

```
HXCC 13 20 VIN1 VIN2 IC=0.5, 1.3
```

The current, through VIN1, initializes to 0.5 mA. The current, through VIN2, initializes to 1.3 mA.

Initial Conditions and UIC Parameters

Use the `.IC` (or `.DCVOLT`), the IC parameter on an element statement, and the `.NODESET` commands to set transient initial conditions in HSPICE. How it initializes depends on whether the `.TRAN` analysis statement includes the UIC parameter. 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.

The node voltages that you specify in the .NODESET statement are used only in the first iteration to set an initial guess for the DC operating point analysis. Transient analysis releases the initialized nodes to calculate the second and later time points.

If you specify the UIC parameter in the `.TRAN` statement, HSPICE does not calculate the initial DC operating point, but directly enters transient analysis.

When you use `.TRAN` with UIC, the `.TRAN` node values (at time zero) are determined by searching for the first value found in this order: from .IC value, then IC parameter on an element statement, then `.NODESET` value, otherwise use a voltage of zero. Note that forcing a node value of the dc operating point may not satisfy KVL and KCL. In this event you may likely see activity during the initial part of the simulation. This may happen if UIC is used and some node values left unspecified, when too many (conflicting) `.IC` values are specified, or when node values are forced and the topology changes. In this event you may likely see activity during the initial part of the simulation. Forcing a node voltage applies a fixed equivalent voltage source during DC analysis and transient analysis removes the voltage sources to calculate the second and later time points.

Therefore to correct DC convergence problems use `.NODESET` commands (without `.TRAN` with UIC) liberally (when a good guess can be provided) and use `.IC`s sparingly (when the exact node voltage is known).

.SAVE and .LOAD Statements

HSPICE saves the operating point, unless you use the `.SAVE LEVEL=NONE` statement. HSPICE restores the saved operating-point file, only if the input file contains a `.LOAD` statement.

The `.SAVE` statement in HSPICE stores the operating point of a circuit, in a file that you specify. You can save the operating point data as either an `.IC` or a `.NODESET` statement. For quick DC convergence in subsequent simulations, use the `.LOAD` statement to input the contents of this file. HSPICE saves the operating point by default, even if the HSPICE input file does not contain a `.SAVE` statement. To not save the operating point, specify `.SAVE LEVEL=NONE`.

A parameter or temperature sweep saves only the first operating point.

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. Do not use the `.LOAD` command for concatenated netlist files.

.DC Statement—DC Sweeps

You can use the `.DC` statement in DC analysis to:

- Sweep any parameter value.
- 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 `.DC` statement format depends on the application that uses it. For syntax details, refer to the .DC command in the *HSPICE Reference Manual: Commands and Control Options*.

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.

Statement	Description
.DCMATCH	A technique for computing the effects of local variations in device characteristics on the DC solution of a circuit.
.PZ	Performs pole/zero analysis.
.SENS	Obtains DC small-signal sensitivities of output variables for circuit parameters.
.TF	Calculates DC small-signal values for transfer functions (ratio of output variable, to input source).

HSPICE includes DC control options, and DC initialization statements, to model resistive parasitics and initialize nodes. These statements enhance convergence properties and accuracy of simulation. This section describes how to perform DC-related, small-signal analysis.

DC Initialization Control Options

Use control options in a DC operating-point analysis, to control DC convergence properties and simulation algorithms. Many of these options also affect transient analysis because DC convergence is an integral part of transient convergence. Include the following options for *both* DC and transient convergence:

- Absolute and relative voltages.
- Current tolerances.
- Matrix options.

Use `.OPTION` statements to specify the following options, which control DC analysis:

ABSTOL	DV	ITL2	PIVREL
CAPTAB	GDCPATH	KCLTEST	PIVTOL
CSHDC	GRAMP	MAXAMP	RESMIN
DCCAP	GSHDC	NEWTOL	SPARSE
DCFOR	GSHUNT	NOPIV	SYMB
DCHOLD	ICSWEEP	OFF	
DCIC	ITLPTRAN	PIVOT	
DCSTEP	ITL1	PIVREF	

DC and AC analysis also use some of these options. Many of these options also affect the transient analysis because DC convergence is an integral part of transient convergence. For a description of transient analysis, see Chapter 14, Transient Analysis.

Accuracy and Convergence

Convergence is the ability to solve a set of circuit equations, within specified tolerances, and within a specified number of iterations. In numerical circuit simulation, a designer specifies a relative and absolute accuracy for the circuit solution. The simulator iteration algorithm then attempts to converge to a solution that is within these set tolerances. That is, if consecutive simulations achieve results within the specified accuracy tolerances, circuit simulation has converged. How quickly the simulator converges, is often a primary concern to a designer—especially for preliminary design trials. So designers willingly sacrifice some accuracy for simulations that converge quickly.

Accuracy Tolerances

HSPICE 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 42 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 359 describes the options in Table 42.

Table 42 Absolute and Relative Accuracy Tolerances

Type	.OPTION	Default
Nodal Voltage Tolerances	ABSVDC	50 µv
	RELVDC	.001

Table 42 Absolute and Relative Accuracy Tolerances

Type	.OPTION	Default
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 43 shows an example of the relationship between the `RELVDC` value, and the number of iterations.

Table 43 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

Table 43 RELV vs. Accuracy and Simulation Time for 2 Bit Adder

RELVDC	Iteration	Delay (ns)	Period (ns)	Fall time (ns)
.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 Appendix D, Transient Simulation with RUNLVL=0.

ABSH	DCON	RELH
ABSI	DCTRAN	RELI
ABSMOS	DI	RELMOS
ABSVDC	GMAX	RELV
CONVERGE	GMINDC	RELVDC

Autoconverge Process

If a circuit does not converge in the number of iterations that `ITL1` specifies, HSPICE initiates an autoconvergence process. This process manipulates `DCON`, `GRAMP`, and `GMINDC`, and even `CONVERGE` in some cases. Figure 37 on page 362 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.

In the process flow shown in Figure 37 on page 362:

- Setting `.OPTION DCON=-1` disables steps 2 and 3.
- Setting `.OPTION CONVERGE=-1` disables steps 4 and 5.
- Setting `.OPTION DCON=-1 CONVERGE=-1` disables steps 2, 3, 4, and 5.
- If you set the `DV` option to a value other than the default, step 2 uses the value you set for `DV`, but step 3 changes `DV` to 1e6.
- Setting `.OPTION GRAMP` has no effect on autoconverge. Autoconverge sets `GRAMP` independently.
- If you set `.OPTION GMINDC`, then `GMINDC` ramps to the value you set, instead of to 1e-12, in steps 2 and 3.

DCON and GMINDC

The `GMINDC` option helps stabilize the circuit, during DC operating-point analysis. For MOSFETs, `GMINDC` helps stabilize the device in the vicinity of the threshold region. HSPICE inserts `GMINDC` between:

- Drain and bulk.
- Source and bulk.
- Drain and source.

The drain-to-source `GMINDC` helps to:

- Linearize the transition from cutoff to weakly-on.
- Smooth-out model discontinuities.
- Compensate for the effects of negative conductances.

The pn junction insertion of `GMINDC` in junction diodes linearizes the low conductance region. As a result, the device behaves like a resistor in the low-conductance region. This prevents the occurrence of zero conductance, and improves the convergence of the circuit.If a circuit does not converge, HSPICE automatically sets the `DCON` option. This option invokes `GMINDC` ramping, in steps 2 and 3 of Figure 37 on page 362.

`GMINDC` for various elements is shown in Figure 38 on page 363.

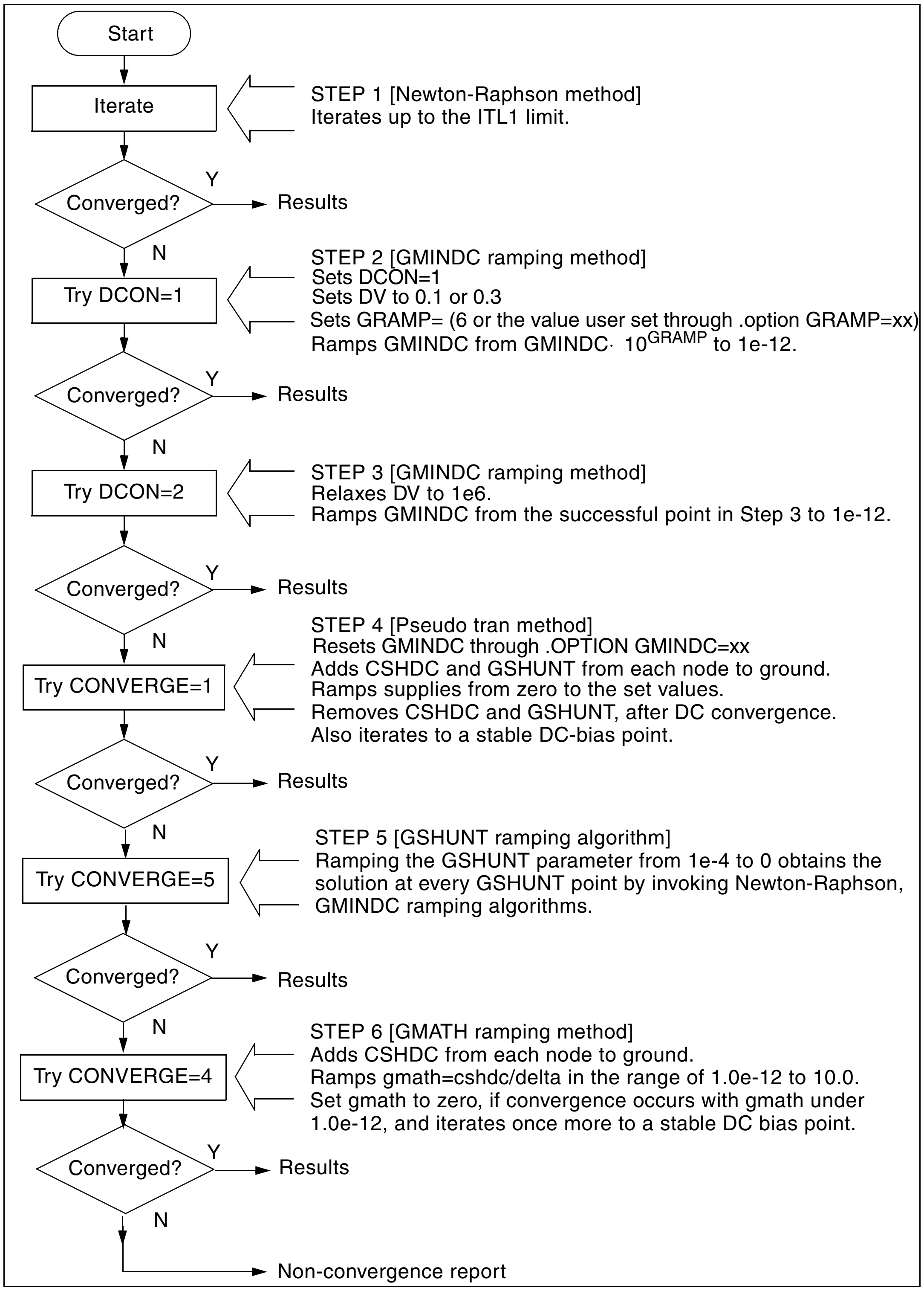

Start
Iterate
STEP 1 [Newton-Raphson method]
Iterates up to the ITL1 limit.
Converged?
Y
Results
N
Try DCON=1
STEP 2 [GMINDC ramping method]
Sets DCON=1
Sets DV to 0.1 or 0.3
Sets GRAMP= (6 or the value user set through .option GRAMP=xx)
Ramps GMINDC from GMINDC· 10^GRAMP to 1e-12.
Converged?
Y
Results
N
Try DCON=2
STEP 3 [GMINDC ramping method]
Relaxes DV to 1e6.
Ramps GMINDC from the successful point in Step 3 to 1e-12.
Converged?
Y
Results
N
Try CONVERGE=1
STEP 4 [Pseudo tran method]
Resets GMINDC through .OPTION GMINDC=xx
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 to a stable DC-bias point.
Converged?
Y
Results
N
Try CONVERGE=5
STEP 5 [GSHUNT ramping algorithm]
Ramping the GSHUNT parameter from 1e-4 to 0 obtains the solution at every GSHUNT point by invoking Newton-Raphson, GMINDC ramping algorithms.
Converged?
Y
Results
N
Try CONVERGE=4
STEP 6 [GMATH ramping method]
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 once more to a stable DC bias point.
Converged?
Y
Results
N
Non-convergence report

Figure 37 Autoconvergence Process Flow Diagram

In the Auto Convergence Process Flow diagram (Figure 37 on page 362):

- Setting .OPTION DCON=-1 disables Steps 2 and 3.
- Setting .OPTION CONVERGE=-1 disables Steps 4 and 5.
- Setting .OPTION DCON=-1 CONVERGE=-1 disables Steps 2, 3, 4, and 5.
- Setting the DV option to a value other than the default, Step 2 uses the value you set for DV, but Step 3 changes DV to 1e6.
- Setting .OPTION GRAMP has no effect on autoconverge. Autoconverge sets GRAMP independently.
- Setting .OPTION GMINDC, then GMINDC ramps to the value you set, instead of to 1e-12, in Steps 2 and 3.
- Setting .OPTION CONVERGE=5 invokes GSHUNT ramping which is tried before GMATH ramping due to a more robust algorithm than GMATH. The GSHUNT algorithm provides enhancements to the autoconvergence flow current probe, and numerical stability.
- Setting .OPTION CONVERGE=4 invokes the GMATH ramping method.

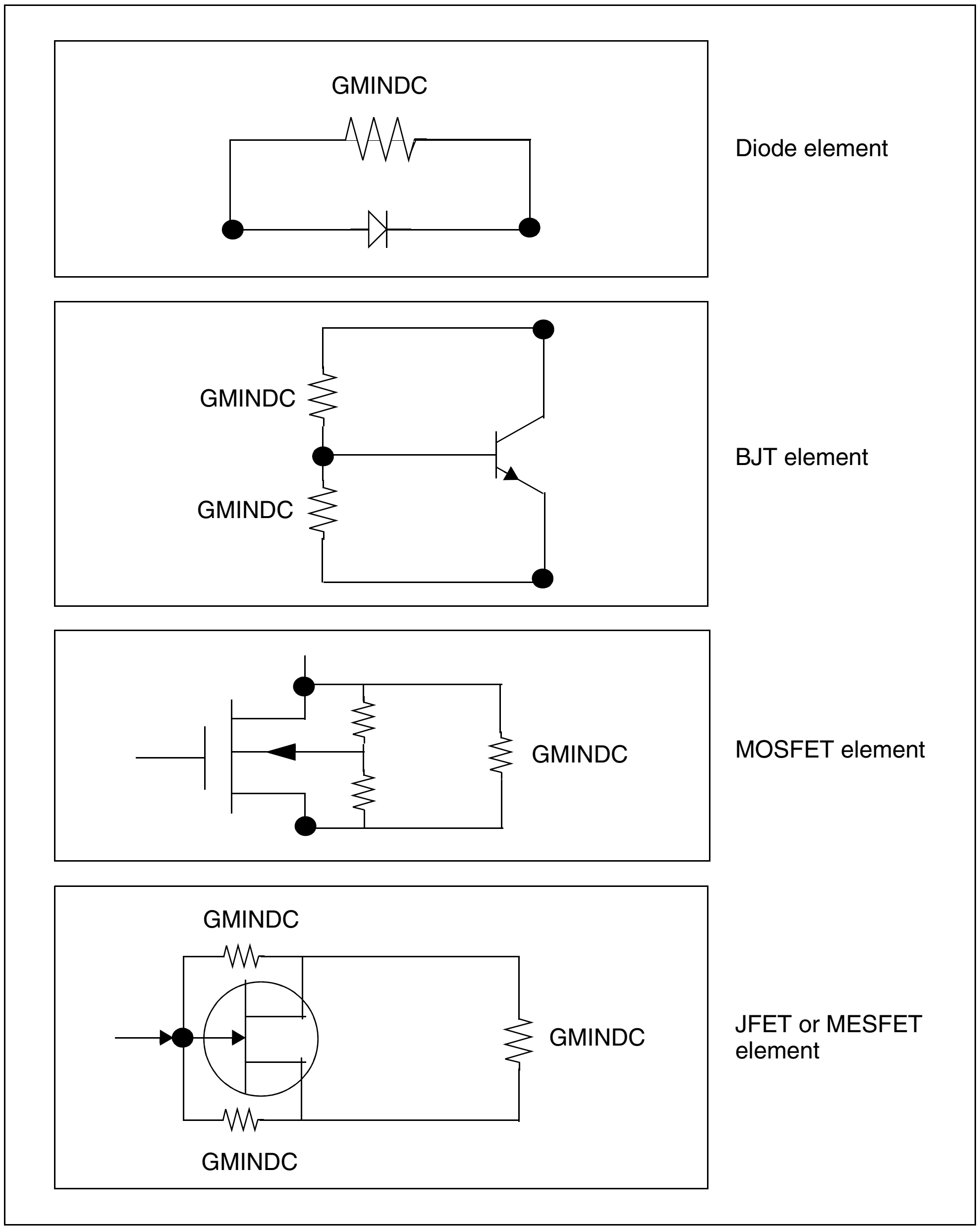


Figure 38 GMINDC Insertion

Reducing DC Errors

To reduce DC errors, perform the following steps:

1. To check topology, set `.OPTION NODE`, to list nodal cross-references.
   - Do all MOS p-channel substrates connect to either VCC or positive supplies?
   - Do all MOS n-channel substrates connect to either GND or negative supplies?
   - Do all vertical NPN substrates connect to either GND or negative supplies?
   - Do all lateral PNP substrates connect to negative supplies?
   - Do all latches have either an OFF transistor, a `.NODESET`, or an `.IC`, on one side?
   - Do all series capacitors have a parallel resistance, or is `.OPTION DCSTEP` set?
2. Verify 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 subthreshold parameters of MOS models at reasonable values?
   - Are JS and JSW set in the MOS model for the DC portion of a diode model? A typical JS value is 1e-4A/M^2.
   - Are CJ and CJSW set in MOS diode models?
   - Is weak-inversion NG and ND set in JFET/MESFET models?
   - Make sure that DIODE models have non-zero values for saturation current, junction capacitance, and series resistance.
   - Use MOS ACM=1, ACM=2, or ACM=3 source and drain diode calculations to automatically generate parasitics.
3. General remarks:
   - Ideal current sources require large values of `.OPTION GRAMP`, especially for BJT and MESFET circuits. Such circuits do not ramp up with the supply voltages, and can force reverse-bias conditions, leading to excessive nodal voltages.

- Schmitt triggers are unpredictable for DC sweep analysis, and sometimes for operating points for the same reasons that oscillators and flip-flops are unpredictable. Use slow transient.
- Large circuits tend to have more convergence problems because they have a higher probability of uncovering a modeling problem.
- Circuits that converge individually, but fail when combined, are almost guaranteed to have a modeling problem.
- Open-loop op-amps have high gain, which can lead to difficulties in converging. Start op-amps in unity-gain configuration, and open them up in transient analysis, using a voltage-variable resistor, or a resistor with a large AC value (for AC analysis).

4. Check your options:
   - Remove all convergence-related options, and try first with no special `.OPTION` settings.
   - Check non-convergence diagnostic tables for non-convergent nodes. Look up non-convergent nodes in the circuit schematic. They are usually latches, Schmitt triggers, or oscillating nodes.
   - For stubborn convergence failures, bypass DC all together, and use `.TRAN` with UIC set. Continue transient analysis until transients settle out, then specify the `.OP` time, to obtain an operating point during the transient analysis. To specify an AC analysis during the transient analysis, add an `.AC` statement to the `.OP` time statement.
   - `SCALE` and `SCALM` scaling options have a significant effect on parameter values in both elements and models. Be careful with units.

Shorted Element Nodes

HSPICE disregards any capacitor, resistor, inductor, diode, BJT, or MOSFET if all of its leads connect together. The simulation ignores it 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 39 on page 366, HSPICE inserts conductance (G), in parallel with capacitance (C_g). This provides current paths around capacitances, in DC analysis.

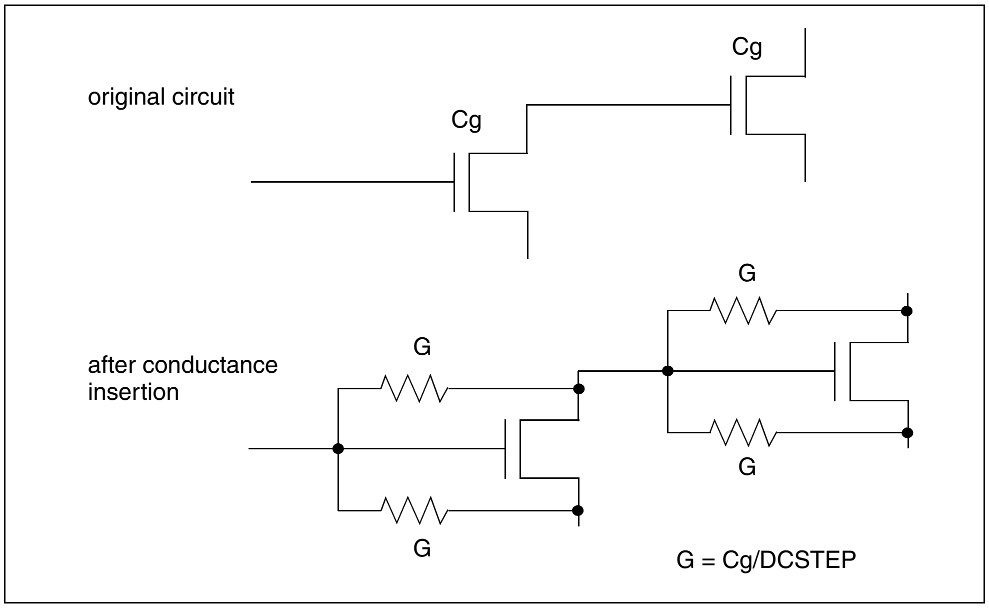


Figure 39 Conductance Insertion

Floating-Point Overflow

If MOS conductance is negative or zero, HSPICE 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: Lists all non-convergent elements, and their associated element currents, element voltages, model parameters, and current error tolerances (tol).

To locate the branch current or nodal voltage that causes non-convergence, use the following steps:

1. Analyze the diagnostic tables. Look for unusually large values of branch currents, nodal voltages or tolerances.
2. After you locate the cause, use the `.NODESET` or `.IC` statements, to initialize the node or branch.

   If circuit simulation does not converge, HSPICE automatically generates a non-convergence diagnostic table, indicating:

   - The quantity of recorded voltage failures.
   - The quantity of recorded branch element failures.

     Any node in a circuit can create voltage failures, including *hidden* nodes (such as extra nodes that parasitic resistors create).

3. Check the element printout for the subcircuit, model, and element name for all parts of the circuit where node voltages or currents do not converge.

For example, Table 44 on page 368 identifies the xinv21, xinv22, xinv23, and xinv24 inverters, as problem subcircuits in a ring oscillator. It also indicates that the p-channel transistors, in the xinv21, xinv22, xinv24 subcircuits, 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 44, 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 44   Subcircuit Voltage, Current, and Tolerance

subckt element model	**xinv21 21:mphc1 0:p1**	**xinv22 22:mphc1 0:p1**	**xinv23 23:mphc1 0:p1**	**xinv23 23:mnch1 0:n1**	**xinv24 24: mphc1 0:p1**
id	27.5809f	140.5646u	1.8123p	1.7017m	5.5132u
ibs	205.9804f	3.1881f	31.2989f	0.	200.0000f
ibd	0.	0.	0.	-168.7011f	0.
vgs	4.9994	-4.9992	69.9223	4.9998	-67.8955
vds	4.9994	206.6633u	69.9225	-64.9225	2.0269
vbs	4.9994	206.6633u	69.9225	0.	2.0269
vth	-653.8030m	-745.5860m	-732.8632m	549.4114m	-656.5097m
tolds	114.8609	82.5624	155.9508	104.5004	5.3653
tolbd	0.	0.	0.	0.	0.
tolbs	3.534e-19	107.1528m	0.	0.	0.

Traceback of Non-Convergence Source

To locate a non-convergence source, trace the circuit path for error tolerance. For example, in an inverter chain, the last inverter can have a very high error tolerance. If this is the case, examine the error tolerance of the elements that drive the inverter. If the driving tolerance is high, the driving element could be the source of non-convergence. However, if the tolerance is low, check the driven element as the source of non-convergence.

Examine the voltages and current levels of a non-convergent MOSFET to discover the operating region of the MOSFET. This information can flow to the location of the discontinuity in the model—for example, subthreshold-to-linear, or linear-to-saturation.

When considering error tolerances, check the current and nodal voltage values. If these values are extremely low, a relatively large number is divided by a very small number. This produces a large calculation result, which can cause the non-convergence errors. To solve this, increase the value of the absolute-accuracy options.

Use the diagnostic table, with the DC iteration limit (`ITL1` option), to find the sources of non-convergence. When you increase or decrease `ITL1`, HSPICE prints output for the problem nodes and elements for a new iteration—that is, the last iteration of the analysis that you set in `ITL1`.

Solutions for Non-Convergent Circuits

Non-convergent circuits generally result from:

- Poor Initial Conditions
- Inappropriate Model Parameters
- PN Junctions (Diodes, MOSFETs, BJTs)

The following sections explain these conditions.

Poor Initial Conditions

Multi-stable circuits need state information, to guide the DC solution. You must initialize ring oscillators and flip-flops. These multi-stable circuits can either produce an intermediate forbidden state, or cause a DC convergence problem. To initialize a circuit, use the `.IC` statement, which forces a node to the requested voltage. Ring oscillators usually require you to set only one stage.

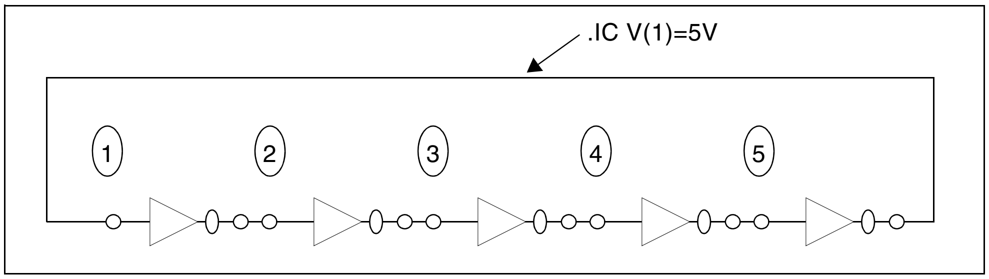


Figure 40 Ring Oscillator

The best way to set up the flip-flop is to use an `.IC` statement in the subcircuit definition.

Example

The following example sets the local `Qset` parameter to 0, and uses this value for the `.IC` statement, to initialize the Q latch output node. As a result, all latches have a default state of Q low. Setting `Qset` to vdd calls a latch, which overrides this state.

```
.subckt latch in Q Q/ d Qset=0
.ic Q=Qset
...
.ends
Xff data_in[1] out[1] out[1]/ strobe LATCH Qset=vdd
```

Inappropriate Model Parameters

If you impose non-physical model parameters, you might create a discontinuous IDS or capacitance model. This can cause an *internal timestep too small* error, during the transient simulation. The mosivcv.sp demonstration file shows IDS, VGS, GM, GDS, GMB, and CV plots for MOS devices. A sweep near threshold from *Vth-0.5* V to *Vth+0.5* V (using a delta of 0.01 V), sometimes discloses a possible discontinuity in the curves.

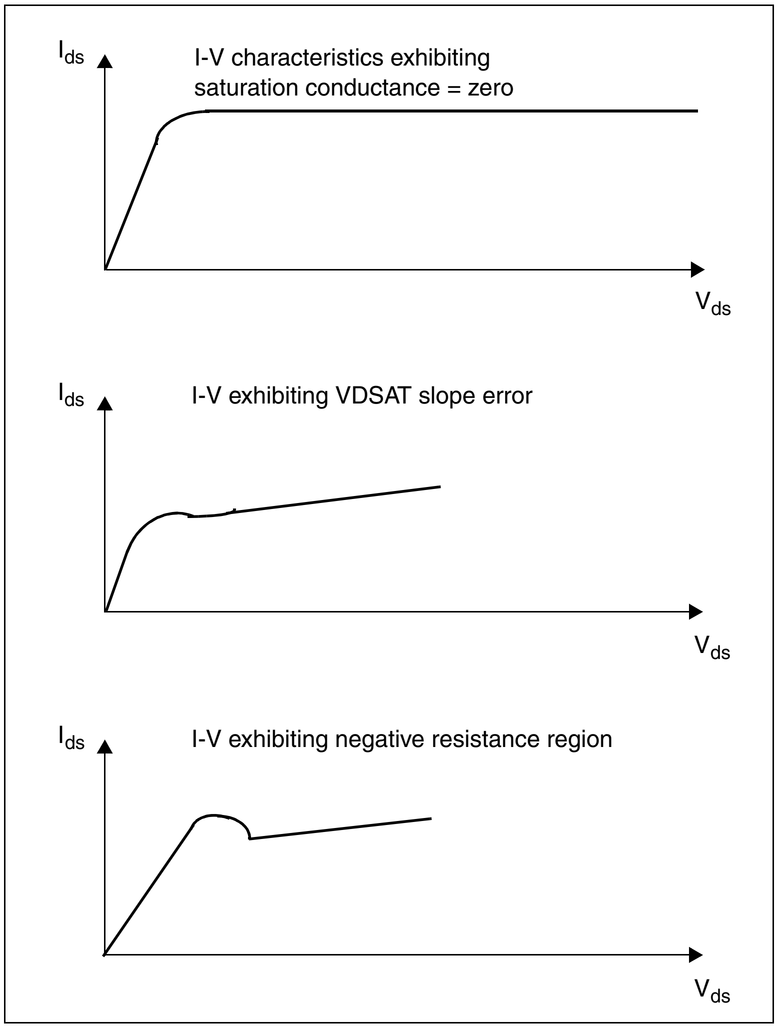


Figure 41 Discontinuous I-V Characteristics

If simulation does not converge when you add a component or change a component value, then the model parameters are not appropriate or do not correspond to physical values they represent.

To locate the problem, follow these steps:

1. Check the input netlist file for non-convergent elements.

   Devices with a *TOL* value greater than 1, are non-convergent.

2. Find the devices at the beginning of the combined-logic string of gates that seem to start the non-convergent string.

3. Check the operating point of these devices very closely, to see what region they operate in.

   Model parameters associated with this region are probably inappropriate.

Circuit simulation uses single-transistor characterization, to simulate a large collection of devices. If a circuit fails to converge, the cause can be a single transistor, anywhere in the circuit.

PN Junctions (Diodes, MOSFETs, BJTs)

PN junctions found in diode, BJT, and MOSFET models, might exhibit non-convergent behavior, in both DC and transient analysis.

Example

PN junctions often have a high *off* resistance, resulting in an ill-conditioned matrix. To overcome this, use `.OPTION GMINDC` and `.OPTION GMIN` to automatically parallel every PN junction in a design, with a conductance.

Non-convergence can occur if you overdrive the PN junction. This happens if you omit a current-limiting resistor, or if the resistor has a very small value. In transient analysis, protection diodes are often temporarily forward-biased (due to the inductive switching effect). This overdrives the diode, and can result in non-convergence, if you omit a current-limiting resistor.

Troubleshooting DC Bias Point and DC Sweep Non-Convergence The following procedures are designed to trade runtime performance and loosen certain tolerance bounds to overcome DC non-convergence. HSPICE steps from one DC convergence algorithm to another other to find a solution. You can assist this process as follows (in the same order).

1. Remove or comment out all simulation control options from your HSPICE testbench/netlists to allow the default auto-convergence procedure to work.
2. For circuits with feedback or multiple bias states (FF and latches), it is important to provide HSPICE with an initial guess that is close to the final solution. Use the `.NODESET` command to set initial voltage guesses. In particular, focus on those nodes that are listed as nonconvergent in the output *.lis* file.

   ```
   .nodeset v(in)=0 v(out)=3.3
   ```

3. Use the symbolic (`.OPTION SYMB`) operating point algorithm which finds initial guesses before calculating the operating point.

   ```
   .option SYMB=1
   ```

Explanation: When the SYMB option is used, HSPICE assumes the circuit is digital and assigns a low/high state to all nodes as a reasonable initial voltage guess. This option improves DC convergence for oscillators, logic, and mixed-signal circuits.

4. Increase ITL1 from default value of 200 up to 500 in steps of 100. To further help DC sweep analysis, you may increase ITL2 in the same manor which increases the number of iterations HSPICE will take at each DC sweep point.

```
.option ITL1=300 ITL2=300
```

Explanation: If increasing ITL2 doesn't help DC sweep analysis, the problem likely lies in model discontinuities. In other words, if you set ITL2=400 and the convergence problem is not solved, it is unlikely that any further increase of the value of ITL2 will help convergence. As a workaround, you may try to increase and offset the sweep size in an attempt to miss model problems.

Original: `.dc vin 0v 3.3v .1v`
Increase: `.dc vin 0v 3.3v .2v`
Offset: `.dc vin .01v 3.31v .1v`

5. HSPICE will try various convergence algorithms as it struggles to achieve DC convergence. Read the *.lis* file to see where HSPICE was when the job aborted. HSPICE first tries DCON=1,2, then converge=1. If this isn't enough, try the other two converge choices along with larger gmindc values (converge=3 is the source stepping method listed in “Inside SPICE”). However, gmindc should not be set larger than 1e-9.

```
.option converge=2 gmindc=1e-11
.option converge=3 gmindc=1e-11
```

6. If certain active element nodes seem to be non-convergent, you may have HSPICE perform 2 DC bias point calculations. The first calculation is performed with the active elements turned off. Then, this solution is used as the first guess for the DC solution with the elements turned on. You may choose one or more elements to turn off, declared on the element line.

```
Diode n1 n2 diode_model off
Qbjt n1 n2 n3 bjt_model off
Mosfet n1 n2 n3 n4 mos_model off
```

Convergence Failure: Too Many Current Probes in Netlist

Probing current in HSPICE is accomplished by the insertion of a zero-volt source. When large numbers of current probes are added explicitly or use of wildcard syntax such as `.probe i(*)`, the size of the solution matrix increases significantly which can lead to convergence failures. These failures are usually accompanied by the error message: `**diagnostic** rebuilding matrix with pivot option for special current probe process` which is then followed by `**error** no convergence in operating point.`

Workarounds:

- Reduce the number of current probes by only probing specific nodes of interest, or adding a qualification to the wildcard.
- Create a saved operating point and tell HSPICE to use those initial conditions in the transient analysis.

The basic steps are:

- Run HSPICE without all the current probes, but include a .OP statement to create an initial conditions (*.ic0*) file.
- Include that file in your netlist. Example:
  ```
  .include my_design.ic0
  ```
- Add “uic” for Use Initial Conditions on the .TRAN line. Example:
  ```
  .tran 1n 100n uic
  ```

Then, it is possible that the design will run to completion even with the large number of current probes.

For more information on non-convergence, refer to Autoconverge Process and Reducing DC Errors in this chapter.

13

Pole/Zero Analysis

Describes how to use pole/zero analysis in HSPICE or HSPICE RF.

You can use pole/zero analysis in HSPICE or HSPICE RF to study the behavior of linear, time-invariant networks. You can apply the results to the design of analog circuits, such as amplifiers and filters. Use pole/zero analysis to determine the stability of a design, or to calculate the poles and zeroes to specify in a `POLE` function (see Using Pole/Zero Analysis on page 376).

Pole/zero analysis uses the `.PZ` (Pole/Zero) statement, instead of pole/zero (POLE function) and Laplace (`LAPLACE` function) transfer function modeling, which are also described in Using Pole/Zero Analysis on page 376.

Overview of Pole/Zero Analysis

In pole/zero analysis, a network transfer function describes a network. For any linear time-invariant network, you can use this general form to write the function:

$$H(s) = \frac{N(s)}{D(s)} = \frac{a_0 s^m + a_1 \cdot s^{(m-1)} + \ldots + a_m}{b_0 s^n + b_1 \cdot s^{(n-1)} + \ldots + b_n}$$

In the factorized form, the general function is:

$$H(s) = \frac{a_0}{b_0} \cdot \frac{(s+z_1)(s+z_2)\ldots(s+z_i)\ldots(s+z_m)}{(s+p_1)(s+p_2)\ldots(s+p_j)\ldots(s+p_m)}$$

- The roots of the numerator N(s) (that is, z_i) are the zeros of the network function.
- The roots of the denominator D(s) (that is, p_j) are the poles of the network function.
- S is a complex frequency.

The dynamic behavior of the network depends on the location of the poles and zeros, on the network function curve (complex plane). The (real) poles are the natural frequencies of the network. You can graphically deduce the magnitude and phase curve of most network functions from the location of its poles and zeros (reference 2).

References on page 394 lists a variety of source materials that address:

- Transfer functions of physical systems.
- Design of systems and physical modeling.
- Interconnect transfer function modeling.

Using Pole/Zero Analysis

HSPICE RF uses the exact matrix approach and the Muller method, while HSPICE uses only the Muller method to calculate the roots of the N(s) and D(s) polynomials.

Matrix Approach

The matrix approach is based on the singular-value matrix decomposition algorithm. It applies primarily to a network that has no frequency-dependent elements. In this case, HSPICE RF writes the `D(s)` function as the determinant of the network matrices, `D(s) = det(G + sC)`, where G is the frequency-independent conductance matrix and C is the capacitance matrix. The poles can be the *eigen* values of the matrix equation `(G + sC) X = 0`, where X is the *eigen* vector.

Similarly, following Cramer's rule, the roots of the N(s) function can also be the *eigen* values of a matrix. HSPICE RF supports two different kinds of singular values deposition (*svd*):

- `OPTION=HQR` requires that the G matrix is non-singular.
- `OPTION=SVD` requires only that G and C are real matrices.

The later `SVD` requires twice as much memory as the first one, and is approximately three times slower.

Muller Method

You can apply the Muller method if the network contains frequency-dependent elements (such as S or W Elements).

The Muller method approximates the polynomial, using a quadratic equation that fits through three points in the vicinity of a root. To obtain successive iterations toward a particular root, HSPICE or HSPICE RF finds the nearer root of a quadratic, whose curve passes through the last three points.

In Muller's method, selecting the three initial points affects both the convergence of the process, and the accuracy of the roots obtained:

1. If the poles or zeros occupy a wide frequency range, then choose (X0R, X0I) close to the origin, to find poles or zeros at the zero frequency first.
2. Find the remaining poles or zeros, in increasing order.

   The (X1R, X1I) and (X2R, X2I) values can be orders of magnitude larger than (X0R, X0I). If any poles or zeros occur at high frequencies, adjust X1I and X2I accordingly.

Pole/zero analysis results are based on the circuit's DC operating point, so the operating point solution must be accurate. Use the `.NODESET` statement (not `.IC`) for initialization, to avoid DC convergence problems.

For the syntax of the `.PZ` statement, see the HSPICE Reference Manual: Commands and Control Options.

How HSPICE Calculates Poles and Zeros

HSPICE calculates poles and zeros independently from the denominator and numerator polynomials of the transfer function respectively. It is possible that there exist common multipliers, such as the pole and zero appearing at exactly the same location. In this case, the pole and zero will just factor out. HSPICE only presents the roots of the two polynomials without doing the factorization.

For example, if you use the netlist *fkerwin.sp* from the demo folder, from the transfer function there should be two poles and zeros:

```
*       = (s**2 + 2) / (s**2 + 0.1*s + 1)
*    poles --- (-0.05004,+0.9987) , (-0.05004,-0.9987)
*    zeros --- ( 0.0    ,+1.4142) , ( 0.0    ,-1.4142)
```

But HSPICE reports three poles and zeros:

```
poles (rad/sec)                 poles (hertz)
***************************************************************
 real             imag     real            imag
zeros (rad/sec)                    zeros (hertz)
-50.0394m       998.7214m        -7.9640m       158.9515m
-50.0394m      -998.7214m        -7.9640m      -158.9515m
***************************************************************
 real             imag             real             imag
 0.               -1.4142           0.           -225.0812m
 0.                1.4142 0.                      225.0812m
-1.4142            0.            -225.0812m         0.
-1.4142            0.            -225.0812m         0.
```

The common pole and zero at -1.4142 0 is cancelled in the transfer function. Since HSPICE solves the denominator and numerator separately, these are reported in the results.

Pole/Zero Analysis Examples

The following are examples of different types of pole/zero analysis.

Example 1 – Low-Pass Filter

This example is based on HSPICE demonstration netlist flp5th.sp, which is available in directory $<installdir>/demo/hspice/filters:

```
file:flp5th.sp lowpass 5th order filter
*****
* reference: gabor c. temes and sanjit k. mitra, "modern fiter
theory
* and design", j. wiley, 1973, page 74.
* t = v(3) / iin
* =0.113*(s**2 + 1.6543)*(s**2 + 0.2632) /
* (s**5 + 0.9206*s**4 + 1.26123*s**3 + 0.74556*s**2
* + 0.2705*s + 0.09836)
*****
* pole zero, ac(.001hz-10hz) analysis
*
.options post
.pz v(3) iin
.ac dec 100 .001hz 10hz
.probe ac vdb(3) vp(3)
*
iin 1 0 1.00 ac 1
r1 1 0 1.0
c3 1 0 1.52
c4 2 0 1.50
c5 3 0 0.83
c1 1 2 0.93
l1 1 2 0.65
c2 2 3 3.80
l2 2 3 1.00
r2 3 0 1.0
.end
```

The following is an equivalent example in HSPICE RF:

```
* HSPICE RF example:
5TH-ORDER LOW_PASS FILTER
*
.OPTION POST
.PZ I(R2) IN
.AC DEC 100 .001HZ 10HZ
IN 0 1 1.00 AC 1
R1 1 0 1.0
C3 1 0 1.52
C4 2 0 1.50
C5 3 0 0.83
C1 1 2 0.93
L1 1 2 0.65
C2 2 3 3.80
L2 2 3 1.00
R2 3 0 1.00
.END
```

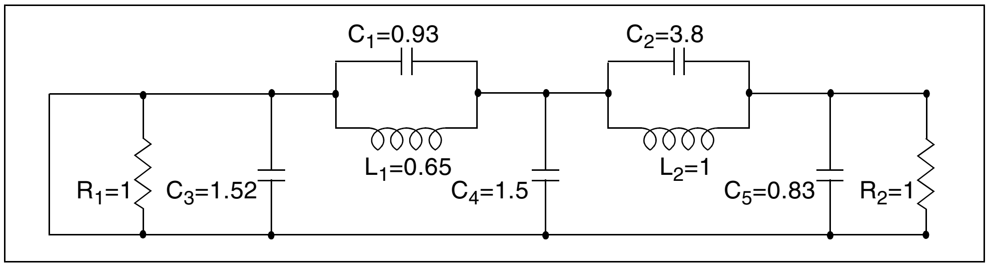


Figure 42 Low-Pass Prototype Filter

Table 45 shows the magnitude and phase variation of the current output, resulting from AC analysis. These results are consistent with pole/zero analysis. The pole/zero unit is radians per second, or hertz. The X-axis unit, in the plot, is in hertz.

Table 45 Pole/Zero Analysis Results for Low-Pass Filter

Poles (rad/sec)		**Poles (Hertz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-6.948473e-02	-4.671778e-01	-1.105884e-02	-7.435365e-02
-6.948473e-02	4.671778e-01	-1.105884e-02	7.435365e-02
-1.182742e-01	-8.914907e-01	-1.882392e-02	-1.418852e-01
-1.182742e-01	8.914907e-01	-1.882392e-02	1.418852e-01
-5.450890e-01	0.000000e+00	-8.675361e-02	0.000000e+00
0.000000e+00	-1.286180e+00	0.000000e+00	-2.047019e-01
0.000000e+00	-5.129892e-01	0.000000e+00	-8.164476e-02
0.000000e+00	5.129892e-01	0.000000e+00	8.164476e-02
0.000000e+00	1.286180e+00	0.000000e+00	2.047019e-01
Constant Factor = 1.129524e-01			

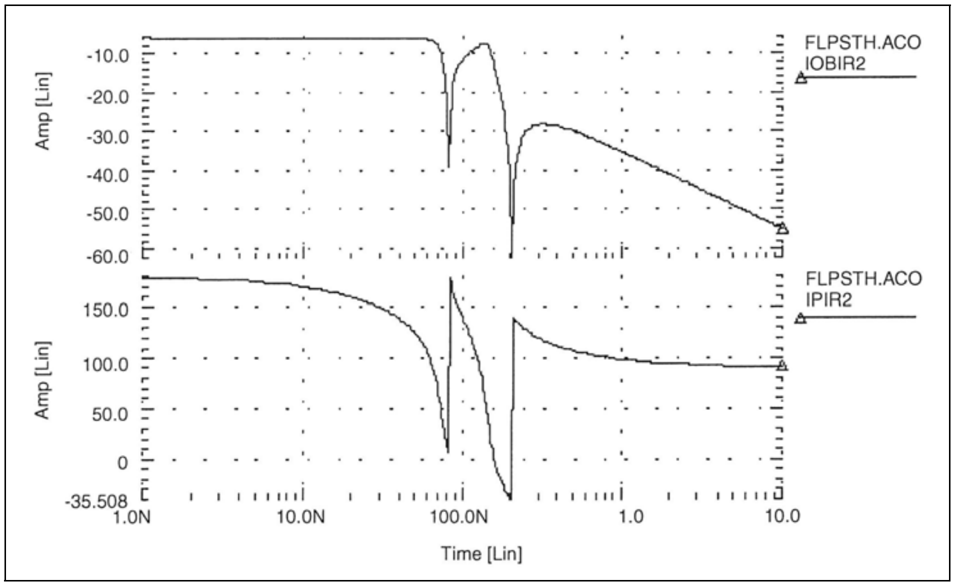


Figure 43 Fifth-Order Low-Pass Filter Response

Example 2 – Kerwin’s Circuit

This example is a HSPICE input file for pole/zero analysis of Kerwin’s circuit, which is located in the following directory:

$installdir/demo/hspice/filters/fkerwin.sp

Table 46 lists analysis results.

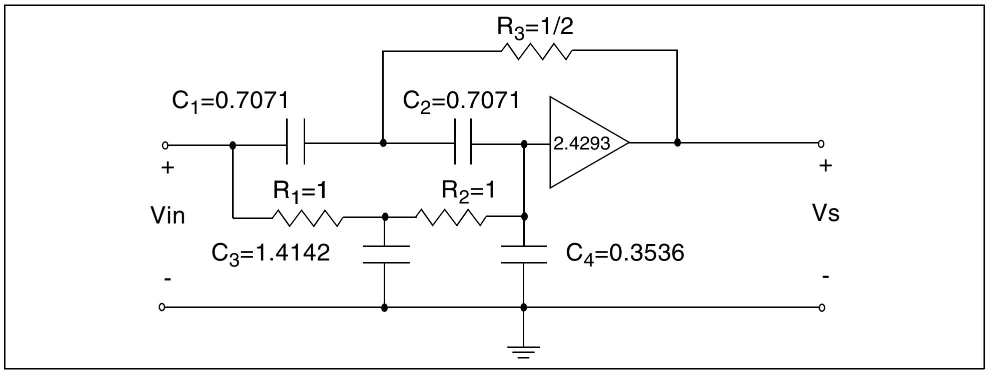


Figure 44 Design Example for Kerwin's Circuit

Table 46 Pole/Zero Analysis Results for Kerwin's Circuit

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-5.003939e-02	9.987214e-01	-7.964016e-03	1.589515e-01
-5.003939e-02	-9.987214e-01	-7.964016e-03	-1.589515e-01
-1.414227e+00	0.000000e+00	-2.250812e-01	0.000000e+00
0.000000e+00	-1.414227e+00	0.000000e+00	-2.250812e-01
0.000000e+00	1.414227e+00	0.000000e+00	2.250812e-01
-1.414227e+00	0.000000e+00	-2.250812e-01	0.000000e+00
Constant Factor = 1.214564e+00			

Example 3 – High-Pass Butterworth Filter

This example is a HSPICE input file, for pole/zero analysis of a fourth-order high-pass Butterworth filter. This file is in <$installdir>/demo/hspice/filters/fhp4th.sp. Table 47 on page 383 shows the analysis results.

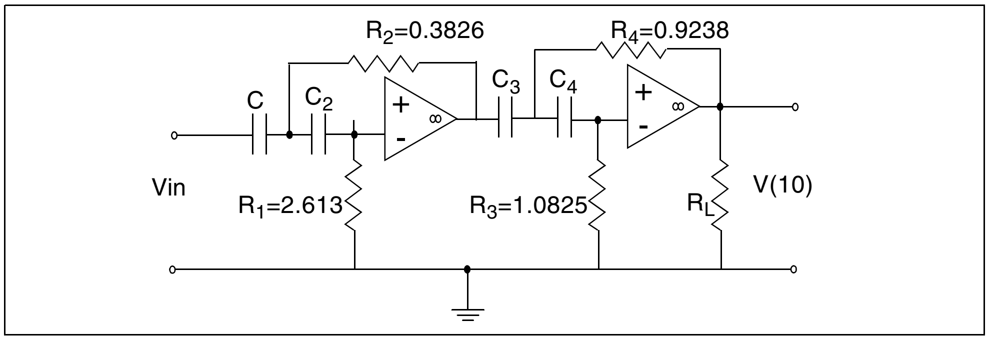


Figure 45 Fourth-Order High-Pass Butterworth Filter

Table 47 Pole/Zero Analysis Results for High-Pass Butterworth Filter

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-3.827019e-01	-9.240160e-01	-6.090889e-02	1.470617e-01
-3.827019e-01	9.240160e-01	-6.090890e-02	-1.470617e-01
-9.237875e-01	3.828878e-01	-1.470254e-01	6.093849e-02
-9.237875e-01	-3.828878e-01	-1.470254e-01	-6.093849e-02
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
Constant Factor = 1.000000e+00			

Example 4 – CMOS Differential Amplifier

This example is based on HSPICE demonstration netlist mcdiff.sp, which is available in directory $<installdir>/demo/hspice/apps. Table 48 on page 385 shows the analysis results.

```
*file: mcdiff.sp cmos differential amplifier
* analysis : ac(20khz-500mhz), pole-zero
* mos level=5
*
.options acct scale=1e-6 scalm=1e-6 wl opts post
.pz v(5) vin
vin 7 0 0 ac 1
.ac dec 10 20k 500meg
.probe ac vdb(5) vp(5)
m1 4 0 6 6 mn 100 10 2 2
m2 5 7 6 6 mn 100 10 2 2
m3 4 4 1 1 mp 60 10 1.5 1.5
m4 5 4 1 1 mp 60 10 1.5 1.5
m5 6 3 2 2 mn 50 10 1.0 1.0
vdd 1 0 5
vss 2 0 -5
vgg 3 0 -3
rin 7 0 1
.model mn nmos level=5 vt=1 ub=700 frc=0.05 tox=800 dnb=1.6e16
+ xj=1.2 latd=0.7 cj=0.13 phi=1.2 tcv=0.003
.model mp pmos level=5 vt=-1 ub=245 frc=0.25 tox=800 dnb=1.3e15
+ xj=1.2 latd=0.9 cj=0.09 phi=0.5 tcv=0.002
.end
```

The following is an equivalent example in HSPICE RF:

```
* HSPICE RF example
CMOS DIFFERENTIAL AMPLIFIER
.OPTION PIVOT SCALE=1.e-6 SCALM=1.e-6 HQR
.PZ V(5) VIN
VIN 7 0 0 AC 1
.AC DEC 10 20K 500MEG
M1 4 0 6 6 MN 100 10 2 2
M2 5 7 6 6 MN 100 10 2 2
M3 4 4 1 1 MP 60 10 1.5 1.5
M4 5 4 1 1 MP 60 10 1.5 1.5
M5 6 3 2 2 MN 50 10 1.0 1.0
VDD 1 0 5
VSS 2 0 -5
VGG 3 0 -3
RIN 7 0 1
.MODEL MN NMOS LEVEL=5 VT=1 UB=700 FRC=0.05 DNB=1.6E16
+ XJ=1.2 LATD=0.7 CJ=0.13 PHI=1.2 TCV=0.003 TOX=800
.MODEL MP PMOS LEVEL=5 VT=-1 UB=245 FRC=0.25 TOX=800
+ DNB=1.3E15 XJ=1.2 LATD=0.9 CJ=0.09 PHI=0.5 TCV=0.002
.END
```

Table 48   Pole/Zero Analysis Results for CMOS Differential Amplifier

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-1.798766e+06	0.000000e+00	-2.862825e+05	0.000000e+00
-1.126313e+08	-6.822910e+07	-1.792583e+07	-1.085900e+07
-1.126313e+08	6.822910e+07	-1.792583e+07	1.085900e+07
-1.315386e+08	7.679633e+07	-2.093502e+07	1.222251e+07
-1.315386e+08	-7.679633e+07	-2.093502e+07	-1.222251e+07
7.999613e+08	0.000000e+00	1.273178e+08	0.000000e+00
Constant Factor = 3.103553e-01			

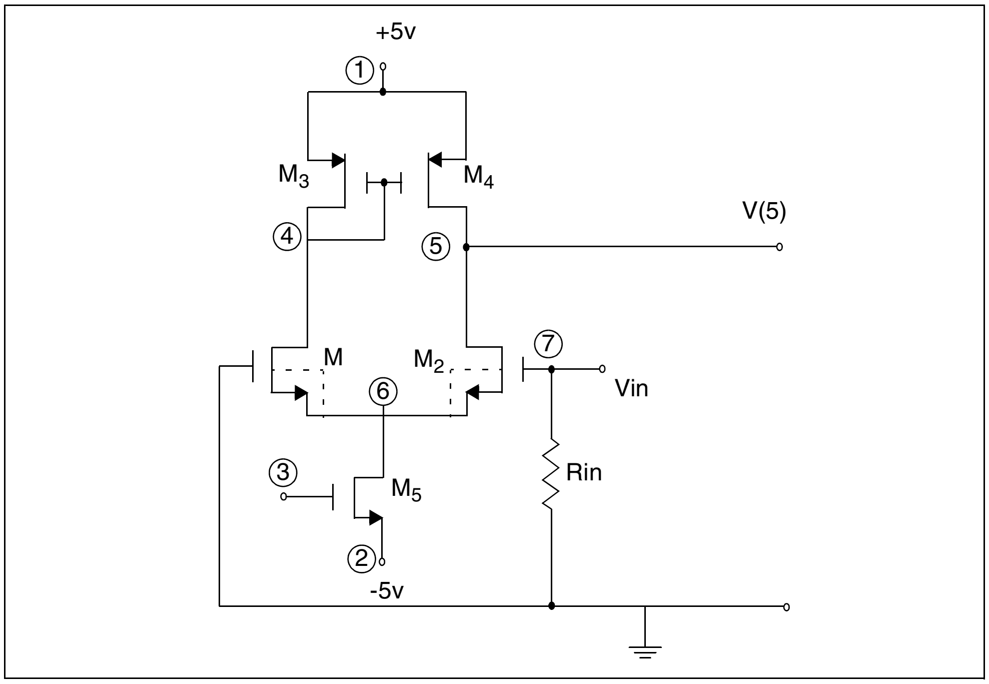


Figure 46 CMOS Differential Amplifier

Example 5 – Simple Amplifier

This example is a HSPICE input file for pole/zero analysis of an equivalent circuit for a simple amplifier with:

- RS¼=RPI=RL=1000 ohms
- gm=0.04 mho
- CMU=1.0e-11 farad
- CPI¼=1.0e-9 farad

The file is in <$installdir>/demo/hspice/apps/ampg.sp. Table 49 shows the analysis results.

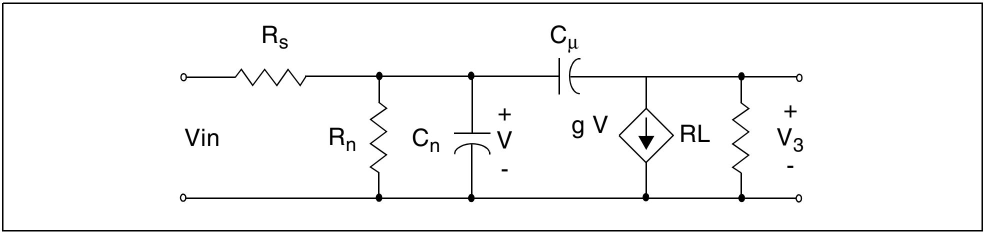


Figure 47 Simple Amplifier

Table 49 Pole/Zero Analysis Results for Amplifier

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-1.412555+06	0.000000e+00	-2.248151e+05	0.000000e+00
-1.415874+08	0.000000e+00	-2.253434e+07	0.000000e+00
4.000000e+09	0.000000e+00	6.366198e+08	0.000000e+00
Constant Factor = 1.000000e+06			

Example 6— Active Low-Pass Filter

This example is based on demonstration netlist flp9th.sp, which is available in directory $<installdir>/demo/hspice/filters. It is for a pole/zero analysis of an active ninth-order low-pass filter by using the ideal operational amplifier element. This example performs an AC analysis. Table 50 is the analysis results.

```
* file flp9th.sp----9th order low-pass filter
*
* reference: jiri vlach and kishore singhal, 'computer
* methods for circuit analysis and design',
* van nostrand reinhold co., 1983, pages 142
* and 494 to 496.
*
* pole/zero analysis and using vcvs as an ideal op-amp.
* for just pole/zero analysis .ac statement is not required.
vin in 0 ac 1
.ac dec 100 1 100k
.print vm(out) vm(in) vp(out)
.probe ac vdb(out,in) par('db(vm(out)/vm(in))')
.pz v(out) vin
.options post dcstep=1e3
+ x0r=-1.23456e+3 x1r=-1.23456e+2 x2r=1.23456e+3
+ fscal=1e-6 gscal=1e3 cscal=1e9 lscal=1e3
.subckt fdnr 1 r1=2k c1=12n r4=4.5k
r1 1 2 r1
c1 2 3 c1
r2 3 4 3.3k
r3 4 5 3.3k
r4 5 6 r4
c2 6 0 10n
eop1 5 0 2 4 level=1
eop2 3 0 6 4 level=1
.ends
*
rs in 1 5.4779k
r12 1 2 4.44k
r23 2 3 3.2201k
r34 3 4 3.63678k
r45 4 out 1.2201k
c5 out 0 10n
x1 1 fdnr r1=2.0076k c1=12n r4=4.5898k
x2 2 fdnr r1=5.9999k c1=6.8n r4=4.25725k
x3 3 fdnr r1=5.88327k c1=4.7n r4=5.62599k
x4 4 fdnr r1=1.0301k c1=6.8n r4=5.808498k
.end
```

The following is an equivalent example in HSPICE RF:

```
* HSPICE RF example
VIN IN 0 AC 1

.PZ V(OUT) VIN
.AC DEC 50 .1K 100K
.OPTION PLST DCSTEP=1E3 XOR=-1.23456E+3 X1R=-1.23456E+2
+ X2R=1.23456E+3 FSCAL=1E-6 GSCAL=1E3 CSCAL=1E9 LSCAL=1E3
.PRINT AC VDB(OUT)

.SUBCKT OPAMP IN+ IN- OUT GM1=1 RI=1K CI=26.6U GM2=1.33333 RL=75
RII IN+ IN- 2MEG
RI1 IN+ 0 500MEG
RI2 IN- 0 500MEG
G1 1 0 IN+ IN- GM1
C1 1 0 CI
R1 1 0 RI
G2 OUT 0 1 0 GM2
RLD OUT 0 RL
.ENDS
.SUBCKT FDNR 1 R1=2K C1=12N R4=4.5K RLX=75
R1 1 2 R1
C1 2 3 C1
R2 3 4 3.3K
R3 4 5 3.3K
R4 5 6 R4
C2 6 0 10N
XOP1 2 4 5 OPAMP
XOP2 6 4 3 OPAMP
.ENDS
$
$
RS IN 1 5.4779K
R12 1 2 4.44K
R23 2 3 3.2201K
R34 3 4 3.63678K
R45 4 OUT 1.2201K
C5 OUT 0 10N

X1 1 FDNR R1=2.0076K C1=12N R4=4.5898K
X2 2 FDNR R1=5.9999K C1=6.8N R4=4.25725K
X3 3 FDNR R1=5.88327K C1=4.7N R4=5.62599K
X4 4 FDNR R1=1.0301K C1=6.8N R4=5.808498K
.END
```

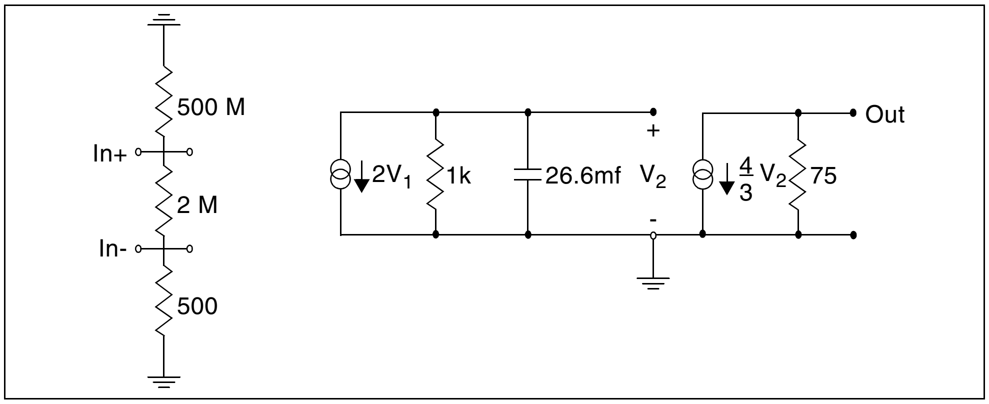


Figure 48 Linear Model of the 741C Operational Amplifier

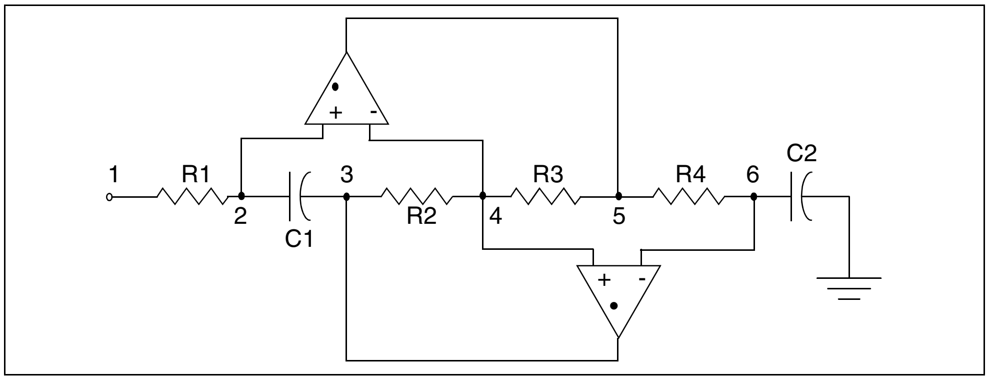


Figure 49 FDNR Subcircuit

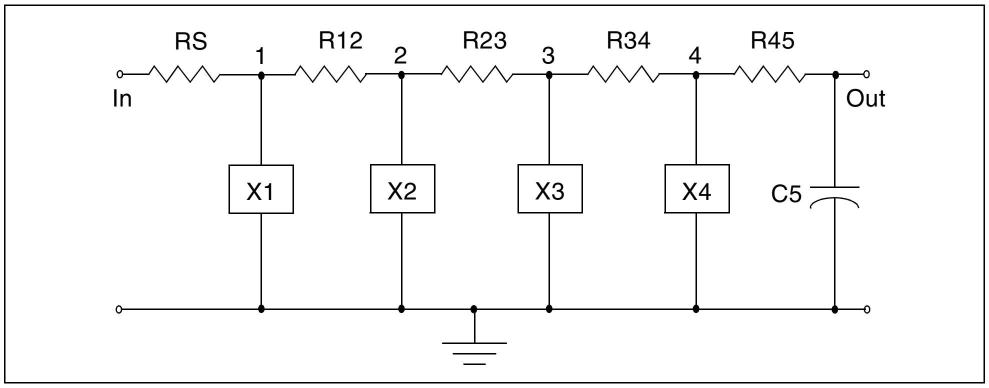


Figure 50 Active Realization of the Low-Pass Filter

Table 50 Pole/Zero Analysis Results for the Active Low-Pass Filter

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-4.505616e+02	-2.210451e+04	-7.170911e+01	-3.518042e+03
-4.505616e+02	2.210451e+04	-7.170911e+01	3.518042e+03
-1.835284e+03	2.148369e+04	-2.920944e+02	3.419236e+03
-1.835284e+03	-2.148369e+04	-2.920944e+02	-3.419236e+03
-4.580172e+03	1.944579e+04	-7.289571e+02	3.094894e+03
-4.580172e+03	-1.944579e+04	-7.289571e+02	-3.094894e+03
-9.701962e+03	1.304893e+04	-1.544115e+03	2.076802e+03
-9.701962e+03	-1.304893e+04	-1.544115e+03	-2.076802e+03
-1.353908e+04	0.000000e+00	-2.154811e+03	0.000000e+00
-3.668995e+06	-3.669793e+06	-5.839386e+05	-5.840657e+05
-3.668995e+06	3.669793e+06	-5.839386e+05	5.840657e+05
-3.676439e+06	-3.676184e+06	-5.851234e+05	-5.850828e+05
-3.676439e+06	3.676184e+06	-5.851234e+05	5.850828e+05

Table 50 Pole/Zero Analysis Results for the Active Low-Pass Filter

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
-3.687870e+06	3.687391e+06	-5.869428e+05	5.868665e+05
-3.687870e+06	-3.687391e+06	-5.869428e+05	-5.868665e+05
-3.695817e+06	-3.695434e+06	-5.882075e+05	-5.881466e+05
-3.695817e+06	+3.695434e+06	-5.882075e+05	5.881466e+05
-3.220467e-02	-2.516970e+04	-5.125532e-03	-4.005882e+03
-3.220467e-02	2.516970e+04	-5.125533e-03	4.005882e+03
2.524420e-01	-2.383956e+04	4.017739e-02	-3.794184e+03
2.524420e-01	2.383956e+04	4.017739e-02	3.794184e+03
1.637164e+00	2.981593e+04	2.605627e-01	4.745353e+03
1.637164e+00	-2.981593e+04	2.605627e-01	-4.745353e+03
4.888484e+00	4.852376e+04	7.780265e-01	7.722796e+03
4.888484e+00	-4.852376e+04	7.780265e-01	-7.722796e+03
-3.641366e+06	-3.642634e+06	-5.795413e+05	-5.797432e+05
-3.641366e+06	3.642634e+06	-5.795413e+05	5.797432e+05
-3.649508e+06	-3.649610e+06	-5.808372e+05	-5.808535e+05
-3.649508e+06	3.649610e+06	-5.808372e+05	5.808535e+05
-3.683700e+06	3.683412e+06	-5.862790e+05	5.862333e+05
-3.683700e+06	-3.683412e+06	-5.862790e+05	-5.862333e+05
-3.693882e+06	3.693739e+06	5.878995e+05	5.878768e+05
-3.693882e+06	-3.693739e+06	-5.878995e+05	-5.878768e+05

Table 50 Pole/Zero Analysis Results for the Active Low-Pass Filter

Poles (rad/sec)		**Poles (Hz)**	
Real	**Imaginary**	**Real**	**Imaginary**
Constant Factor = 4.451586e+02			

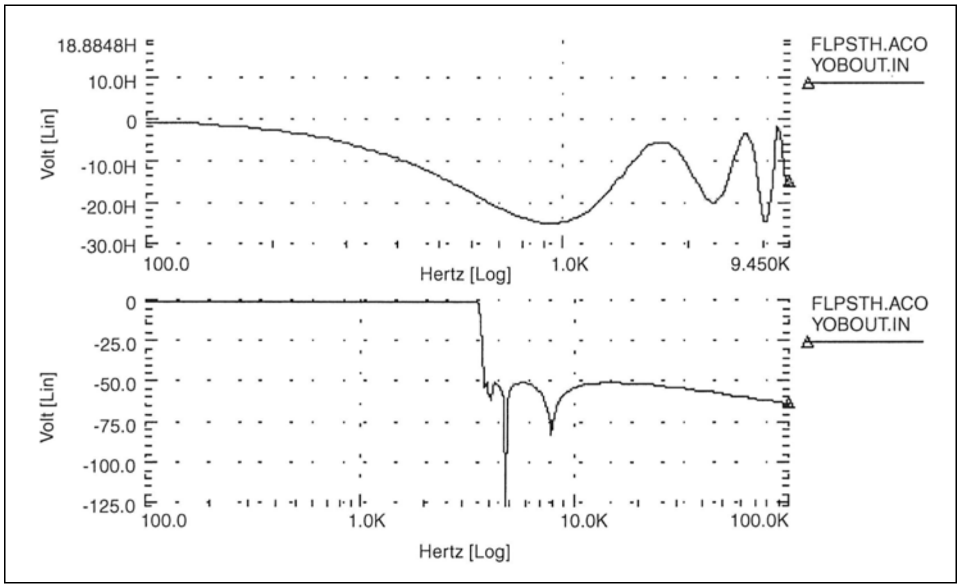


Figure 51 9th Order Low-Pass Filter Response

The graph in Figure 51 shows overall response of the low-pass filter.

References

[1] Desoer, Charles A. and Kuh, Ernest S. *Basic Circuit Theory*. New York: McGraw-Hill.1969. Chapter 15.

[2] Van Valkenburg, M. E. *Network Analysis*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1974, Chapters 10 & 13.

[3] R.H. Canon, Jr. *Dynamics of Physical Systems*. New York: McGraw-Hill, 1967. This text describes electrical, mechanical, pneumatic, hydraulic, and mixed systems.

[4] B.C. Kuo. *Automatic Control Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1975. This source discusses control system design, and provides background material about physical modeling.

[5] L.T. Pillage, and R.A. Rohrer. "Asymptotic Waveform Evaluation for Timing Analysis", *IEEE Trans CAD.* Apr. 1990, pp. 352 - 366. This paper is a good references about interconnect transfer function modeling, and discusses extracting transfer functions for timing analysis.

[6] S. Lin, and E.S. Kuh. "Transient Simulation of Lossy Interconnects Based on the Recursive Convolution Formulation", *IEEE Trans CAS*. Nov. 1992, pp. 879 - 892. This paper is another source for how to model interconnect transfer functions.

[7] Muller, D. E., *A Method for Solving Algebraic Equations Using a Computer, Mathematical Tables, and Other Aids to Computation (MTAC)*. 1956, Vol. 10,. pp. 208-215.

[8] Temes, Gabor C. and Mitra, Sanjit K. *Modern Filter Theory And Design.* J. Wiley, 1973, page 74.

[9] Temes, Gabor C. and Lapatra, Jack W. *Circuit Synthesis And Design*, McGraw-Hill. 1977, page 301, example 7-6.

[10] Temes, Gabor C. and Mitra, Sanjit K., *Modern Filter Theory And Design.* J. Wiley, 1973, page 348, example 8-3.

[11] Desoer, Charles A. and Kuh, Ernest S. *Basic Circuit Theory.* McGraw-Hill, 1969, page 613, example 3.

[12] Vlach, Jiri and Singhal, Kishore. *Computer Methods For Circuit Analysis and Design.* Van Nostrand Reinhold Co., 1983, pages 142, 494-496.

14 Transient Analysis

Describes how to use transient analysis to compute the circuit solution.

Transient analysis computes the circuit solution, as a function of time, over a time range specified in the `.TRAN` statement.

For descriptions of individual HSPICE commands referenced in this chapter, see the *HSPICE Reference Manual: Commands and Control Options*.

These topics are presented in the following sections:

Simulation Flow

Figure 52 on page 396 illustrates the simulation flow for transient analysis in HSPICE.

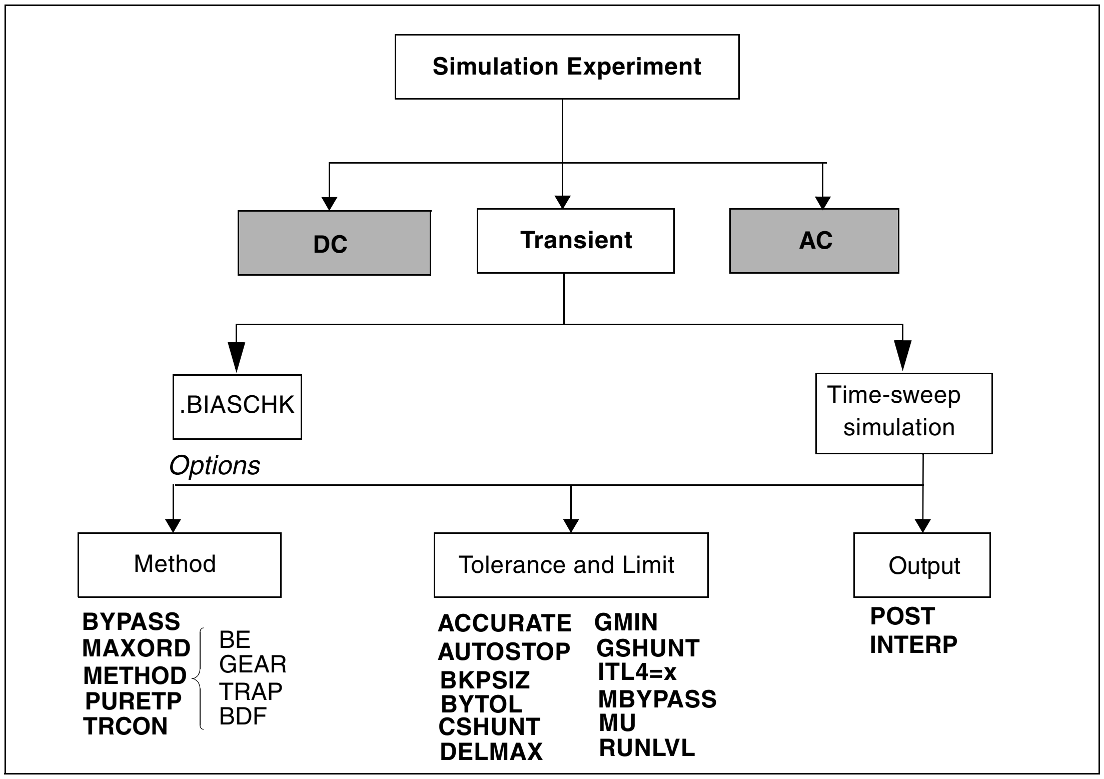


Figure 52 Transient Analysis Simulation Flow

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.

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.

For setting initial conditions, see Initial Conditions and UIC Parameters.

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
```

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. See also: Troubleshooting: Internal Timestep, Measurement Errors at the end of this chapter.

Data-Driven vs. Outer Parameter Sweeps

The following defines the differences between a data-driven sweep and an outer parameter sweep.

Data-Driven Sweep

The use of a data set allows the sweeping of both nonuniform values and multiple parameters. You will need to specify each value varied in the simulation. This method generates one output file for the entire simulation. When viewing signals, the traces correspond to each parameter sweep.

Example

```
.tran 1n 100n sweep data=mydata
.data mydata param1 param2 ...
val1  vala ...
val2  valb ...
 ....
.enddata
```

Parameter Sweep

When the values of a parameter can be expressed using decade, octave, linear, or point-of-interest variation, you can use the `sweep` keyword to control the parameter. This method does not allow for multiple parameters to be swept. Similar to the data-driven sweep, only one output file is created, with the signals having multiple traces. Be sure to sequence the *var* (`param`) before the *type* (`DEC`, `LIN`).

Examples

In this example, `param` will be varied 10 times for each decade from 1u to 10u and a transient analysis will be run for each value.

```
.tran 1n 100n sweep param DEC 10 1u 10u
```

In this example, `param` will be varied 5 equal times from 1u to 10u with a transient analysis for each value.

```
.tran 1n 100n sweep param LIN 5 1u 10u
```

Similar to the data-driven sweep, only one output file will be created. The signals will also be multi-membered.

Sweeping Multiple Parameters

Although HSPICE does not directly provide the facility to sweep multiple parameters, it does offer the `.DATA` table structure. A linked perl script is available to allow you to specify lists of parameters and values at https://solvnet.synopsys.com/retrieve/021478.html

This script will create a `.DATA` table with all permutations of the listed values. It also allows you to create `.ALTER`s instead of a `.DATA` table, if preferred. For usage details, run `hspice_param_sweeper -h`.

The script's output goes to STDOUT, so redirect it to a file, for example: hspice_param_sweeper > *param_sweep.sp*), and then `.INCLUDE` the file into your HSPICE netlist. If you choose to create `.ALTER`s, make sure you `.INCLUDE` them at the very end of your netlist.

If you create a `.DATA` table, you can invoke it as follows:

```
.TRAN 10p 100n SWEEP DATA=sweeper_params
```

Here is a sample input to the script.

```
  vddr: 1.1, 1.0, 0.9
  vssr: 0.0
  temp: 0, 55, 100
```

Note that `temp` is a special parameter that will sweep the simulation temperature. This example will produce a `.DATA` table with 9 rows (3*1*3) containing all combinations of the listed parameter values, or 1 base sim + 8 `.ALTER`s if the `-alter` option is used.

Here is the output produced by the sample input.

```
 .DATA sweeper_params temp vddr vssr
  0 1.1 0.0
 55 1.1 0.0
100 1.1 0.0
  0 1.0 0.0
 55 1.0 0.0
100 1.0 0.0
  0 0.9 0.0
 55 0.9 0.0
100 0.9 0.0
 .ENDDATA
```

After you download the script named *hspice_param_sweeper.gz* (right-click and select "Save Target As..."), be sure to modify the first line of the script to point to your local installation of perl. The default path should work on most systems.

Specifying Data Driven Timesteps

Instead of using a constant timestep in a `.TRAN` statement, you can specify the timesteps using an inline data statement for the transient simulation.

The data defined in the `.DATA` statement should define the time point and current value for a PWL current source. In the following example, the `.DATA` statement *tstep_val* defines the time step, *step_val* and the current value, *ival*. HSPICE uses the timesteps defined in the `.DATA` statement during the transient simulation.

```
Ipwl nd1 0 PWL (step_val ival)
.tran DATA = tstep_val
.DATA tstep_val step_val ival
+ 10p 1m
+ 30p 10m
+ 70P 10m
+ 100p 100m
.ENDDATA
```

The timestep value specified in the data table (`DATA=tstep_val`) controls the print intervals.

Transient Analysis Output

```
.print tran ov1 [ov2 ... ovN]
.probe tran ov1 [ov2 ... ovN]
.measure tran measspec
```

The *ov1, ... ovN* output variables can include the following:

- V(n): voltage at node *n*.
- V(n1<,n2>): voltage between the *n1* and *n2* nodes.
- Vn(d1): voltage at *n*th terminal of the *d1* device.
- In(d1): current into *n*th terminal of the *d1* device.
- '*expression*': expression, involving the plot variables above

You can use wildcards to specify multiple output variables in a single command. Output is affected by `.OPTION POST` or `.OPTION PROBE`.

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, or .MEASURE statement to a *.tr# file.

Transient Analysis of an RC Network

Follow these steps to run a transient analysis of a RC network with a pulse source, a DC source, and an AC source:

1. Type the following netlist into a file named quickTRAN.sp.

```
A SIMPLE TRANSIENT RUN
.OPTION LIST NODE POST
.OP
.TRAN 10N 2U
.PRINT TRAN V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1 PULSE 0 5 10N 20N 20N 500N 2U
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

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

Note:

The V1 source specification includes a pulse source. For the syntax of pulse sources and other types of sources, see Chapter 9, Sources and Stimuli.

2. To run HSPICE, type the following:

   ```
   hspice quickTRAN.sp > quickTRAN.lis
   ```

3. To examine the simulation results and status, use an editor and view the .lis and .st0 files.
4. Run WaveView and open the .sp file.
5. From the File menu, select File > Import Waveform > File.
6. Select the *quickTRAN.tr0* file from the Open: Waveform Files window.
7. Display the voltage at nodes 1 and 2 on the x-axis.

Figure 53 shows the waveforms.

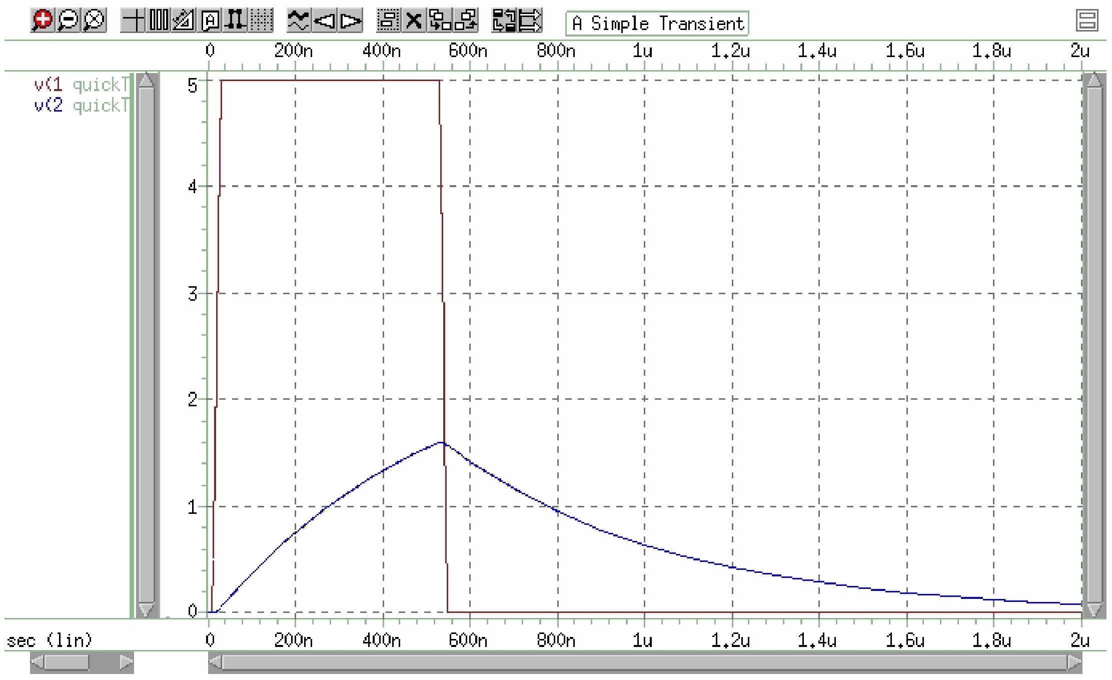


Figure 53 Voltages at RC Network Circuit Node 1 and Node 2

Transient Analysis of an Inverter

As a final example, you can analyze the behavior of the simple MOS inverter shown in Figure 54.

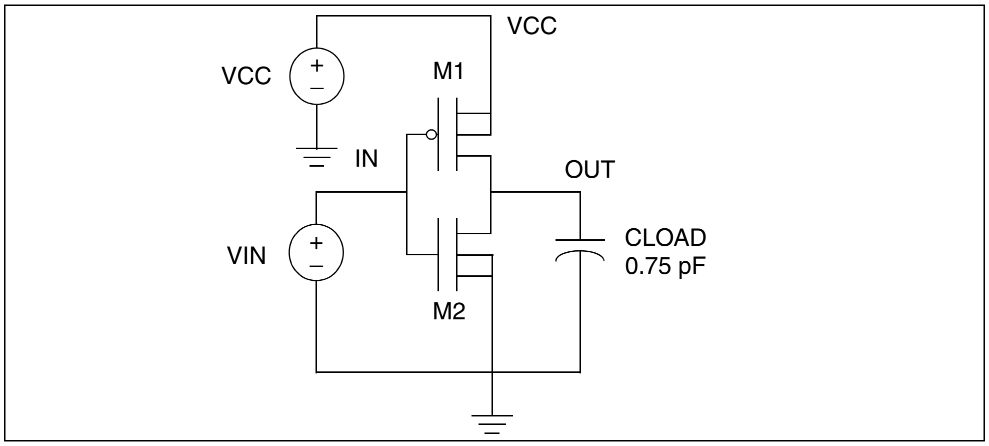


Figure 54 MOS Inverter Circuit

Follow these steps to analyze this behavior:

1. Type the following netlist data into a file named quickINV.sp.

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

   You can find the complete netlist for this example in directory $<installdir>/demo/hspice/apps/quickINV.sp.

2. To run HSPICE, type the following:

```
hspice quickINV.sp > quickINV.lis
```

3. Use WaveView to examine the voltage waveforms, at the inverter IN and OUT nodes.

Figure 55 on page 403 shows the waveforms.

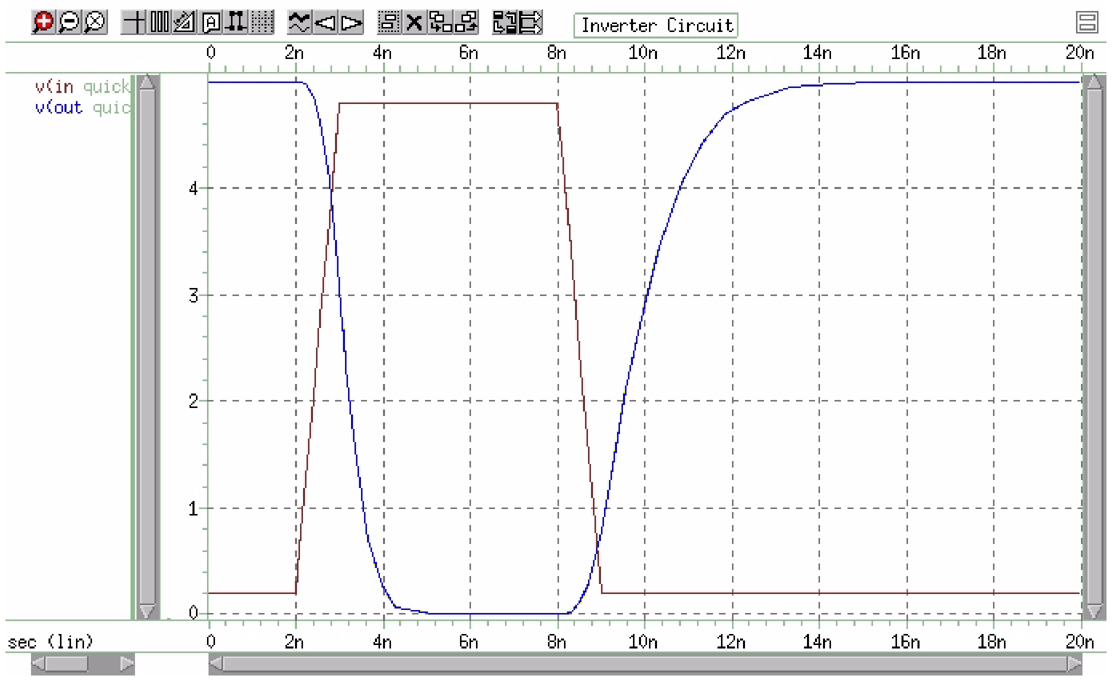


Figure 55 Voltage at MOS Inverter Node 1 and Node 2

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 51 lists the options for RUNLVL.

Table 51 Transient Control Options with RUNLVL Turned On, by Category

Method	Tolerance and Limit		Output
BYPASS	ACCURATE	GMIN	POST
MAXORD	AUTOSTOP	GSHUNT	INTERP
METHOD=	BKPSIZ	ITL4	
Backward-Euler (BE)	BYTOL	MBYPASS	
GEAR	CSHUNT	MAXAMP	
TRAP	DELMAX	MU	
BDF		RUNLVL	
PURETP			
TRCON			

For discussion of METHOD options, see Numerical Integration Algorithm Controls on page 408.

Simulation Speed and Accuracy Using the RUNLVL Option

Convergence is the ability to solve a set of circuit equations within specified tolerances and within a specified number of iterations. How quickly the simulator converges, is often a primary concern to a designer—especially for preliminary design trials. So designers may willingly sacrifice some accuracy for simulations that converge quickly.

The RUNLVL algorithm, which is on by default in HSPICE, focuses on a balance between speed and accuracy. The RUNLVL algorithm:

1. Uses an enhanced Local Truncation Error (LTE) method based on nodal voltage for timestep control. This is advantageous because voltage is the target result users want from a simulation, and there is a clear mathematical relation between error tolerance and time step.
2. Adopts a new Newton-Raphson (NR) iteration method for transient analysis. It not only improves the convergence but also makes the convergence faster.
3. Improves the BYPASS algorithm, as well.

Transient analysis performance may be improved up to 20X without any loss of accuracy by using the RUNLVL option compared to RUNLVL=0. For discussion of RUNLVL=0, see Appendix D, Transient Simulation with RUNLVL=0.

RUNLVL Features

The RUNLVL algorithm provides the following characteristics:

- Simplifies accuracy control by setting RUNLVL values between 1 and 6 with 6 discrete settings (1=fastest, 6=most accurate).
- Avoids interpolation error in .MEASURE statements by using the interpolating polynomial used by the time integration method.
- Dynamically checks for correct handling of input signals and controlled sources between computed time steps to avoid small time step being set before transient simulation start.
- Allows HSPICE to take time steps no larger than (Tstop-Tstart)/20. DELMAX is automatically set to (Tstop-Tstart)/20 if there is no specific setting of DELMAX. The effect is that, for example, HSPICE can take larger time steps for flat regions.

The RUNLVL algorithm scales all simulation tolerances simultaneously and affects time step control, convergence, and model bypass all at once.

This algorithm is activated only by `.option RUNLVL=`*`value`*. Higher values of RUNLVL result in smaller time steps (more Newton-Raphson iterations) to meet stricter error tolerances, and higher simulation accuracy.

A valid value for .OPTION RUNLVL is an integer from 1 to 6. Values outside of this range will cause an error. The default value for RUNLVL is 3. This is the recommended starting setting. Higher values are for simulations requiring high accuracy. Lower values are used for simulating pure digital or mostly digital circuits. Setting RUNLVL=0 will turn off the option. If there are multiple settings of RUNLVL options, the last one set is used.

The .OPTION RUNLVL invokes the advanced simulation algorithm, with the default value of RUNLVL=3. This is the recommended starting setting. However, you can set it to a higher value if the circuit type is pure analog and/or the simulation needs high accuracy.

Table 52 Guidelines for RUNLVL Settings

Circuit type	RUNLVL Setting
Digital	RUNLVL=1-3
Analog or mixed signal accuracy	RUNLVL=3-5
Cell characterization	RUNLVL=5-6

The RUNLVL flag and its effective value is reported in a **.lis* file. The RUNLVL option is automatically set in the `$install_dir/hspice/hspice.ini` file, which is generated during the installation process.

All HSPICE simulations first try to find ONE implicit hspice.ini file and take it as the first include file; the search order for *hspice.ini* is:

1. Current working directory
2. User's home directory
3. $install_dir/hspice directory

Interactions Between .OPTION RUNLVL and Other Options

Refer to Table 53 for information on how RUNLVL affects the values of other options. Since the latest algorithm invoked by RUNLVL sets the timestep and error tolerance internally, many transient error tolerance and timestep control options are no longer valid; furthermore, to assure the greatest efficiency of the RUNLVL algorithm, you should let the new engine manage everything itself. Options that are recommended not to tune are listed in the table, as well.

Note:

If RUNLV is set without value, its value defaults to =3. RNLVL=0 must be set explicitly. (See Appendix D, Transient Simulation with RUNLVL=0.)

Table 53 Options and Interactions

Option	Default value without RUNLVL	Default value with RUNLVL=3	User definition ignored	Recommend not to tune
ABSV/VNTOL	50u	50u		x
ABSVAR	500m	500m	x	
ACCURATE* [a]	0	0		
BYPASS* [a]	2	2		
CHGTOL	1.0f	1.0f	x	
DI	100	100		x
DVDT	4	4	x	
DVTR	1.0k	1.0k	x	

Table 53 Options and Interactions (Continued)

Option	**Default value without RUNLVL**	**Default value with RUNLVL=3**	**User definition ignored**	**Recommend not to tune**
FAST** [b]	0	0	x	
FS	250m	250m		x
FT	250m	250m	x	
IMIN/ITL3	3	3	x	
LVLTIM	1	4	x	
METHOD*** [c]	TRAP	TRAP		
RELQ	10m	10m	x	
RELTOL	1.0m	1.0m		x
RELV	1.0m	1.0m		x
RELVAR	300.0m	300.0m	x	
RMAX	5	5	x	
RMIN	1.0n	1.0n		x

*a. * ACCURATE and BYPASS notes:*
1. If .option ACCURATE is set, then the RUNLVL value is limited to 5 or 6. Specifying a RUNLVL less than 5 results in a simulation at RUNLVL=5. When both ACCURATE and RUNLVL are set, the RUNLVL algorithm will be used.

2. When RUNLVL is set, BYPASS is set to 2. Users can re-define the BYPASS value by setting .option BYPASS=<value>; this behavior is independent of the order of RUNLVL and BYPASS;

*b. **The FAST option is disabled by the RUNLVL option; setting the RUNLVL value to 1 is comparable to setting the FAST option.*

*c. ***RUNLVL can work with METHOD=GEAR; in cases where GEAR only determines the numeric integration method during transient analysis, the other options that were previously set by GEAR (when there is no RUNLVL) now are determined by the RUNLVL mode. This behavior is independent of the order of RUNLVL and METHOD. See the following table.*

The interactions of RUNLVL and GEAR are shown in Table 54.

Table 54 RUNLVL option and GEAR method interactions

Option	GEAR without RUNLVL	GEAR with RUNLVL=3
BYPASS	0	2
BYTOL	50u	100u
LVLTIM	2	Disabled by runlvl
MAXORD	2	3 for RUNLVL=6 2 for RUNLVL=1-5
MBYPASS	1	2
RMAX	2	Disabled by runlvl

HSPICE maintains all the algorithms of RUNLVL=0 for backward compatibility consideration. See Appendix D, Transient Simulation with RUNLVL=0.

Numerical Integration Algorithm Controls

In HSPICE transient analysis, you can select one of several options solve the circuit differential algebraic equations:

- Backward-Euler
- Gear
- Trapezoidal
- BDF

Table 55 Integration Method

Integration Algorithm	Option Settings	Comments

Backward-Euler (BE)	METHOD=GEAR MAXORD=1 or
	METHOD=GEAR MU=0
		Backward-Euler only

GEAR	METHOD=GEAR
	METHOD=GEAR MAXORD=2|3
		Combines GEAR and BE
		2nd/3rd order increases accuracy

Table 55 Integration Method (Continued)

Integration Algorithm	Option Settings	Comments

TRAP	METHOD=TRAP
	METHOD=TRAP PURETP
		Combines Trapezoidal and BE
		Trapezoidal only

BDF	METHOD=BDF	Higher order integration (Backward Differentiation Formulae)

Each algorithm has advantages and disadvantages.

The trapezoidal is often the preferred algorithm because of its high accuracy level and low simulation time. The pure trapezoidal (PURETP) is recommended for oscillators.

The Gear method is an appropriate algorithm for convergence. 2nd-order GEAR is more accurate than Backward-Euler and 3rd-order GEAR is more accurate than 2nd-order GEAR. The GEAR is recommended for those circuit simulations that require high accuracy on current such as leakage current measurement.

If the circuit fails to converge using the Trapezoidal integration method, HSPICE uses the GEAR method to run the transient analysis again from time=0. This process is called autoconvergence.

The BDF method is an high order integration method based on the backward differentiation formulae. The key features include: variable order, variable step size, and high order polynomial interpolation. Two tolerance options are available to the user for the BDF method: `.OPTIONS BDFRTOL` (relative) and `BDFATOL` (absolute); each has a default of 1e-3. The BDF limitations are listed below.

`METHOD=BDF` supports the following models only:

- MOSFET, levels 1-54
- BJT, level 1 only
- Diodes, all
- Resistors, all
- Capacitors, excludes DC block
- Independent sources: V and I
- Dependent sources: E/F/G/H

- L, excludes AC choke
- K, excludes magnetic core, ideal transformer

Note:

BDF issues a warning in the *.lis* file if it encounters an unsupported model. The message is similar to: `WARNING!!!, netlist contains 'unsupported models', HSP-BDF is disabled.`

Dynamic Check Using the .BIASCHK Statement

The `.BIASCHK` statement can monitor the voltage bias, current, device-size, expression and region during transient analysis, and reports:

- Element name
- Time
- Terminals
- Bias that exceeds the limit
- Number of times the bias exceeds the limit for an element

For the syntax and description of this statement, see the .BIASCHK command in the *HSPICE Reference Manual: Commands and Control Options*.

HSPICE saves the information as both a warning and a BIASCHK summary in the **.lis* file. You can use this command only for active elements and capacitors.

You can also use `.OPTION BIASFILE` and `.OPTION BIAWARN` with a `.BIASCHK` statement.

The following limitations apply to the `.BIASCHK` statement:

- `.BIASCHK` is only supported for diode, jfet, nmos, pmos, bjt, and c models, as well as subcircuits.
- For a device-size check, only W and L MOSFET models are supported.
- Wildcards in element and model names, and `except` definitions are supported but not in expressions.

Four methods are available to check the data with the `.BIASCHK` command:

- Limit and noise method
- Maximum method

- Minimum method
- Region method

  Note:

  The region method of data checking is only supported in MOSFET models.

Limit and Noise Method

For a transient simulation using the limit and noise method to check the data, use the following syntax:

For local_max

`v(tn-1) > *limit_value*`

The bias corresponds anyone of the following two conditions:

- `v(tn-1) > v(tn) && v(tn-1) >= v(tn-2)`
- `v(tn-1) >= v(tn) && v(tn-1) > v(tn-2)`

`local_min`: The minimum bias after the time last local max occurs.

During a transient analysis, the `local_max` is recorded if it is greater than the limit. In the summary reported after transient analysis, the `local_max(current)` is replaced with the `local_max(next)` when the following comparison is true:

local_max(current) - local_min < noise && local_max(next) - local_min < noise && local_max(current) < local_max(next)

At the end of the simulation, all `local_max` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is greater than the `limit_value` you specify.

Maximum Method

For a transient simulation using the maximum method, use this syntax:

For `local_max`:

`v(tn-1) > *max_value*`

The bias corresponds any one of the following two conditions:

- `v(tn-1) > v(tn) && v(tn-1) >= v(tn-2)`
- `v(tn-1) >= v(tn) && v(tn-1) > v(tn-2)`

During a transient analysis, all `local_max` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is greater than `max_value` you specify.

Minimum Method

For a transient simulation using the minimum method to check the data, use the following syntax:

For `local_min`:

`v(tn) <` *`min_value`*

The bias corresponds any one of the following two conditions:

- `v(tn-1) < v(tn) && v(tn-1) <= v(tn-2)`
- `v(tn-1) <= v(tn) && v(tn-1) < v(tn-2)`

During a transient analysis, all `local_min` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is smaller than `min_value` you specify.

Region Method

This method is only for MOSFET models. Three regions exist:

- cutoff
- linear
- saturation

When the specified transistor enters and exits during transient analysis, the specified region is reported.

The biaschk.sp demo example is a netlist that uses the `.BIASCHK` command for a transient simulation. You can find the sample netlist for this example in:

$installdir/demo/hspice/apps/biaschk.sp

Troubleshooting: Internal Timestep, Measurement Errors

This section provides the following tips for error recovery:

- Troubleshooting ‘Time step Too Small’ Errors
- Troubleshooting .MEASUREMENT Issues

Troubleshooting ‘Time step Too Small’ Errors

These are the usual steps to follow when you get an “internal time step too small” in transient analysis errors. The best approach is to incrementally change the values of these options, one at a time. Note the time immediately following the timestep error in the list file. If your simulation gets further into the run, then the option is beneficial and you may wish to try higher or lower values as appropriate.

1. Be sure you are using the latest version of HSPICE if you can. Improvements are continuously made to convergence algorithms.
2. Comment out all timestep and convergence options you already have and try increasing the value of .OPTION RUNLVL as a first step.

   The RUNLVL option is turned on by default starting with version 2006.09 to level=3. It implements improved convergence techniques. If a higher RUNLVL such as 5 or 6 is set, try a lower RUNLVL to get convergence.

   Note:

   Remove any other convergence options when you use RUNLVL.
3. Increase .OPTION ITL4

   This is the number of iterations HSPICE will try at one time point, before giving up and taking a smaller time step. The default is 8.

   Suggested values:
   ```
   .option itl4 = 50
   .option itl4 = 100
   ```

   Note:

   .OPTION ITL4 is the same as .OPTION IMAX. 100 is the maximum value.
4. Use GSHUNT and CSHUNT to add small amounts of conductance and or capacitance from each node to ground. Together or alone, these options can help solve timestep too small problems caused by either high-frequency oscillations or numerical noise.

   ```
   .option gshunt=1e-13 cshunt=1e-17
   .option gshunt=1e-12 cshunt=1e-16
   .option gshunt=1e-11 cshunt=5e-15
   .option gshunt=1e-10 cshunt=1e-15
   .option gshunt=1e-9  cshunt=1e-14
   ```

5. Increase the timestep value, to step over possible model discontinuities.

   From original timestep settings, change the `.TRAN` statement to incrementally increase TSTEP:
   ```
   .tran (2)*tstep tstop
   .tran (2.5)*tstep tstop
   .tran (3)*tstep tstop
   ```
6. Using `.OPTION METHOD=GEAR` may help certain high gain analog (such as op-amps) and/or oscillatory circuits (such as a ring oscillators) during transient analysis by changing integration methods.
   ```
   .option method=gear
   ```
7. Investigate the device models used. Be sure the version of the models was developed for or qualified with the version of HSPICE you are using. For CMOS devices, make sure you have finite terminal capacitances and resistances. For level 49, be sure you have the model parameters as specified in the following example: (these are samples, not defaults)
   ```
   .model mname nmos level=49 version=3.2
    + cj=5e-4 cjsw=1e-10 cgd0=1e-10 cgs0=1e-10 rs=1e-9 rd=1e-9
   ```
   In the case of BJT device, be sure to have the following model parameters set (again, examples, not defaults).
   ```
   .model mname npn rb=50 r c=.4 re=1e-3
   ```

Troubleshooting .MEASUREMENT Issues

If `.MEASURE` results are incorrect compared to the waveforms, the differences you see may be due to one or more of the following issues.

- You are not comparing the same point. Make sure that the proper nodes and sweep points are being used for each comparison.
- Do you have `.option INTERP` set in your netlist? If so, remove it. HSPICE will only save data points at the interval defined by the tstep parameter in the `.TRAN` statement. For example, for the `.TRAN` statement:
  ```
  .tran 1n 100n
  ```
  HSPICE will save 100 points at 1ns intervals to the tr0 file. The lack of precision can cause issues with your measurements.

- In your `.TRAN` statement, did you use the `START` keyword to delay output generation? If so, it should be removed. This option interferes with the `.measure` calculations.
- If a `.measure` statement uses the result of previous `.meas` statement, then the calculation starts when the previous result is found. Until the previous result is found, it outputs zero.

15

Spectrum Analysis

Describes HSPICE implementation of spectrum analysis based on the Fourier transforms.

Spectrum analysis represents a time-domain signal within the frequency domain. It most commonly uses the Fourier transform. A Discrete Fourier Transform (DFT) uses sequences of time values to determine the frequency content of analog signals in circuit simulation.

The following topics are covered in these sections:

- Spectrum Analysis (Fourier Transform)
- .FFT Analysis
- Examining the FFT Output
- AM Modulation
- Graphical Output
- Balanced Modulator and Demodulator
- Signal Detection Test Circuit

Spectrum Analysis (Fourier Transform)

This section describes the Fourier and FFT Analysis flow for HSPICE.

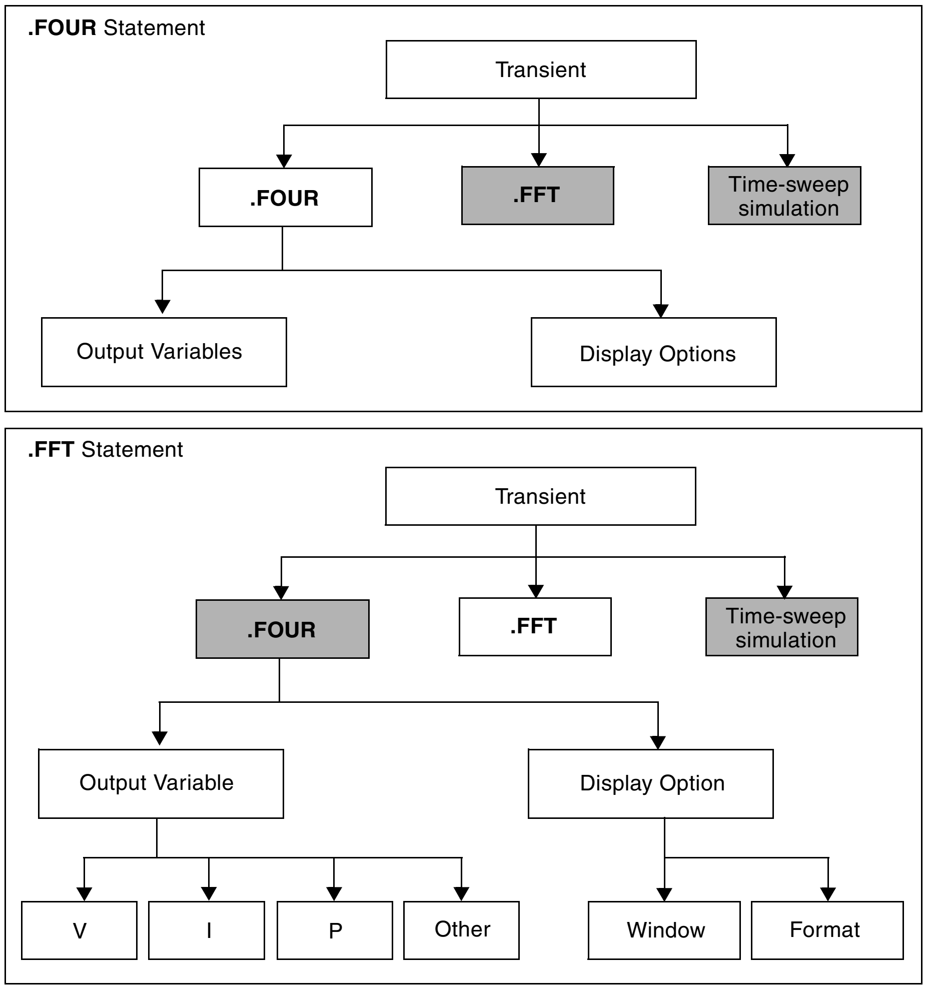


Figure 56 Fourier and FFT Analysis

HSPICE provides two different Fourier analyses.

- `.FOUR` is the same as is available in SPICE 2G6: a standard, fixed-window analysis tool. The `.FOUR` statement performs a Fourier analysis, as part of the transient analysis.
- `.FFT` is a much more flexible Fourier analysis tool. Use it for analysis tasks that require more detail and precision.

Using the Fourier-Related Statements and Options

For syntax and examples, see the following commands and control options in the *HSPICE Reference Manual: Commands and Control Options*:

- .FFT
- .FOUR
- .MEASURE FFT
- .OPTION FFT_ACCURATE
- .OPTION ACCURATE
- .OPTION FFTOUT
- .ENVFFT

Fourier Accuracy

Fourier Accuracy is dependent on transient simulation accuracy. For best accuracy, set small values for `.OPTION RMAX` or `.OPTION DELMAX`. For maximum accuracy, set `.OPTION DELMAX` to `1/(500*frequency)`. For circuits with very high resonance factors (high-Q circuits, such as crystal oscillators, tank circuits, and active filters), set `DELMAX` to less than
`1/(500*frequency)`
where, frequency refers to fundamental frequency.

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:

$$THD = \frac{1}{R1} \cdot \left(\sum_{m=2}^{9} R_m^2 \right)^{1/2} \cdot 100\%$$

The following equation calculates the Fourier coefficients:

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(mt) + \sum_{m=0}^{9} D_m \cdot \sin(mt)$$

The following equations calculate values for the preceding equation:

$$C_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \cos(m \cdot t) \cdot dt$$

$$D_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \sin(m \cdot t) \cdot dt$$

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(m \cdot t) + \sum_{m=0}^{9} D_m \cdot \sin(m \cdot t)$$

The following equations approximate the C and D values:

$$C_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \cos\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

$$D_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \sin\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

The following equations calculate the magnitude and phase:

$$R_m = (C_m^2 + D_m^2)^{1/2}$$

$$\Phi_m = \arctan\left(\frac{C_m}{D_m}\right)$$

Example 1

The following is input content for an `.OP`, `.TRAN`, or `.FOUR` analysis. This example is based on demonstration netlist four.sp, which is available in the directory $*installdir*/demo/hspice/apps.

```
CMOS INVERTER
*
M1 2 1 0 0 NMOS W=20U L=5U
M2 2 1 3 3 PMOS W=40U L=5U
VDD 3 0 5
VIN 1 0 SIN 2.5 2.5 20MEG
*
.MODEL NMOS NMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.MODEL PMOS PMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.OP
.TRAN 1N 500N
.FOUR 20MEG V(2)
.PRINT TRAN V(2) V(1)
.END
```

Example 2

```
******
cmos inverter
**** fourier analysis   tnom = 25.000 temp = 25.000 ****
fourier components of transient response v(2)
dc component=2.430D+00
harmonic   frequency  fourier    normalized  phase    normalized
   no        (hz)   component    component   (deg)    phase (deg)
 1       20.0000x    3.0462      1.0000   176.5386      0.
 2       40.0000x  115.7006m   37.9817m -106.2672  -282.8057
 3       60.0000x  753.0446m  247.2061m  170.7288    -5.8098
 4       80.0000x   77.8910m   25.5697m -125.9511  -302.4897
 5      100.0000x  296.5549m   97.3517m  164.5430   -11.9956
 6      120.0000x   50.0994m   16.4464m -148.1115   -324.6501
 7      140.0000x  125.2127m   41.1043m  157.7399   -18.7987
 8      160.0000x   25.6916m    8.4339m  172.9579    -3.5807
 9      180.0000x   47.7347m   15.6701m  154.1858   -22.3528
    total harmonic distortion=   27.3791   percent
```

.FFT Analysis

The `.FFT` statement uses the internal time point values. By default, `.FFT` uses a second-order interpolation to obtain waveform samples, based on the number of points that you specify.

You can use windowing functions to reduce the effects of waveform truncation on the spectral content. You can also use the `.FFT` command to specify:

- Output format
- Output frequency range

- Start and stop time point
- Fundamental frequency
- Window type
- Number of sampling time points

Using Windows in FFT Analysis

One problem with spectrum analysis in circuit simulators is that the duration of the signals is finite, although adjustable. Applying the FFT method to finite-duration sequences can produce inadequate results. This occurs because DFT assumes periodic extensions, causing spectral leakage.

The effect occurs when the finite duration of the signal does not result in a sequence that contains a whole number of periods. This is especially true when you use FFT to detect or estimate signals – that is, to detect weak signals in the presence of strong signals, or to resolve a cluster of equal-strength frequencies.

In FFT analysis, windows are frequency-weighting functions, applied to the time-domain data, to reduce the spectral leakage associated with finite-duration time signals. Windows are smoothing functions, which peak in the middle frequencies, and decrease to zero at the edges. Windows reduce the effects of discontinuities, as a result of finite duration. Figure 57 shows the windows available in HSPICE or HSPICE RF. Table 56 on page 423 lists the common performance parameters, for FFT windows.

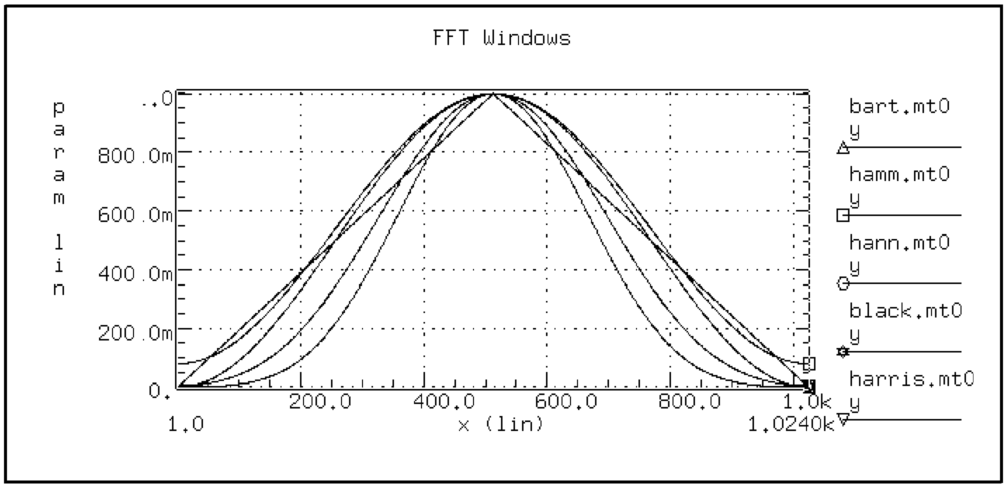


Figure 57 FFT Windows

The most important parameters in Table 56 are:

- Highest side-lobe level (to reduce bias, the lower the better).
- Worst-case processing loss (to increase detectability, the lower the better).

Table 56 Window Weighting Characteristics in FFT Analysis

Window	**Equation**	**Highest Side-Lobe (dB)**	**Side-Lobe Roll-Off (dB/ octave)**	**3.0-dB Bandwidth (1.0/T)**	**Worst-Case Process Loss (dB)**
Rectangular	W(*n*)=1, $0 \le n <$ NP[a]	-13	-6	0.89	3.92
Bartlett	W(*n*)=2*n*/(NP-1), $0 \le n \le$(NP/2)-1 W(*n*)=2-2*n*/(NP-1), NP/2 $\le n <$ NP	-27	-12	1.28	3.07
Hanning	W(*n*)=0.5-0.5[cos(2π*n*/(NP-1))], $0 \le n <$ NP	-32	-18	1.44	3.18
Hamming	W(*n*)=0.54-0.46[cos(2π*n*/(NP-1))], $0 \le v <$ NP	-43	-6	1.30	3.10
Blackman	W(*n*)=0.42323 -0.49755[cos(2π*n*/(NP-1))] +0.07922cos[cos(4π*n*/(NP-1))], $0 \le n <$ NP	-58	-18	1.68	3.47
Blackman-Harris	W(*n*)=0.35875 -0.48829[cos(2π*n*/(NP-1))] +0.14128[cos(4π*n*/(NP-1))] -0.01168[cos(6π*n*/(NP-1))], $0 \le n <$ NP	-92	-6	1.90	3.85
Gaussian	W(n)=exp[-0.5a2(NP/2-1-n)2/(NP)2],				
a=2.5	$0 \le n \le$(NP/2)-1	-42	-6	1.33	3.14
a=3.0	W(n)=exp[-0.5a2(n-NP/2)2/(NP)2],	-55	-6	1.55	3.40
a=3.5	NP/2 $\le n <$ NP	-69	-6	1.79	3.73

Table 56 Window Weighting Characteristics in FFT Analysis (Continued)

Window	Equation	Highest Side-Lobe (dB)	Side-Lobe Roll-Off (dB/ octave)	3.0-dB Bandwidth (1.0/T)	Worst-Case Process Loss (dB)
Kaiser-	W(n)=I0(x2)/I0(x1)				
Bessel	x1=pa	-46	-6	1.43	3.20
a=2.0	x2=x1*sqrt[1-(2(NP/2-1-n)/NP)2],	-57	-6	1.57	3.38
a=2.5	0 ≤n ≤(NP/2)-1	-69	-6	1.71	3.56
a=3.0	x2=x1*sqrt[1-(2(n-NP/2)/NP)2],	-82	-6	0.89	3.74
a=3.5	NP/2 ≤n < NP				
	I0 is the zero-order modified				
	Bessel function				

a. NP is the number of points used for the FFT analysis.

Some compromise usually is necessary, to find a suitable window filtering for each application. As a rule, window performance improves with functions of higher complexity (those listed lower in the table).

- The Kaiser window has an `ALFA` parameter, which adjusts the compromise between different figures of merit for the window.
- The simple rectangular window produces a simple bandpass truncation, in the classical Gibbs phenomenon.
- The Bartlett or triangular window has good processing loss, and good side-lobe roll-off, but lacks sufficient bias reduction.
- The Hanning, Hamming, Blackman, and Blackman-Harris windows use progressively more complicated cosine functions. These functions provide smooth truncation, and a wide range of side-lobe level and processing loss.
- The last two windows in the table are parameterized windows. Use these windows to adjust the side-lobe level, the 3 dB bandwidth, and the processing loss.

Figure 58 and Figure 59 show the characteristics of two typical windows.

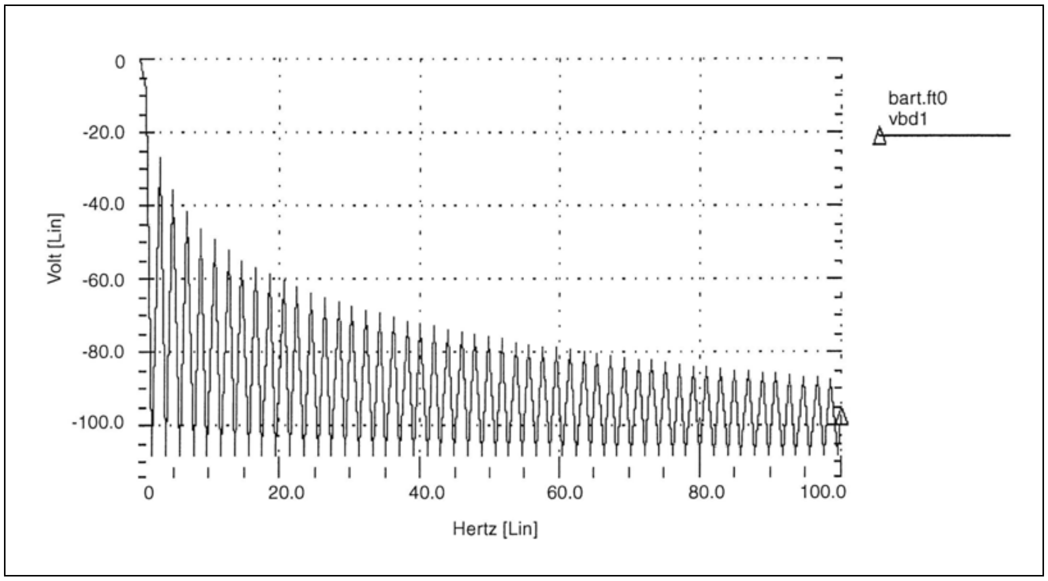


Figure 58 Bartlett Window Characteristics

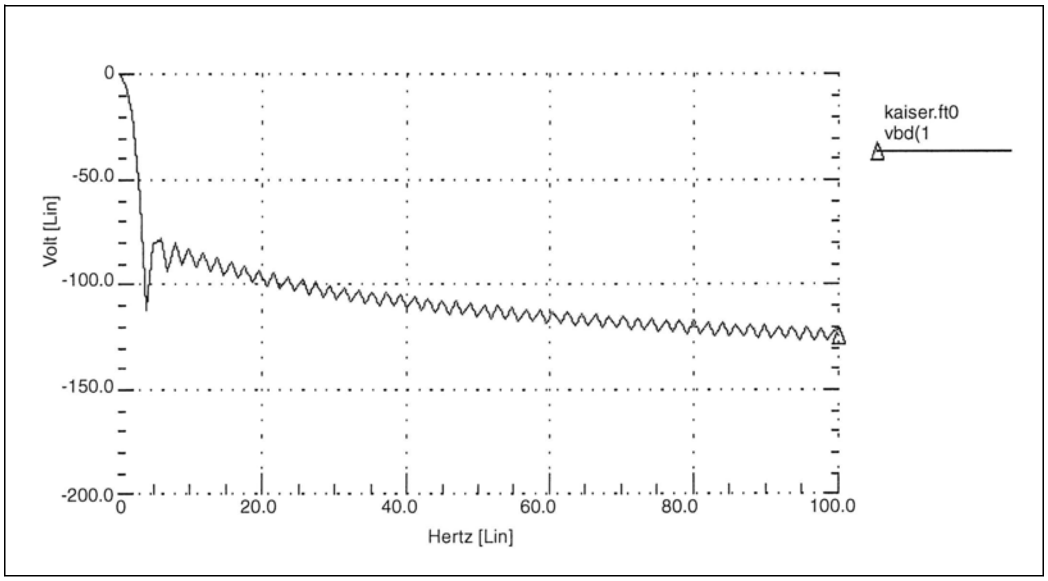


Figure 59 Kaiser-Bessel Window Characteristics, ALFA=3.0

Examining the FFT Output

HSPICE or HSPICE RF prints FFT analysis results in tabular format, in the *.lis* file. These results are based on parameters in the `.FFT` statement. HSPICE prints normalized magnitude values, unless you specify `FORMAT=UNORM`, in which case it prints unnormalized magnitude values. The number of printed frequencies is half the number of points (NP) specified in the `.FFT` statement.

- If you use `FMIN` to specify a minimum frequency, or `FMAX` to specify a maximum frequency, HSPICE or HSPICE RF prints only the specified frequency range.
- If you use `FREQ` to specify a frequency, HSPICE or HSPICE RF outputs harmonics of this frequency, and the percent of total harmonic distortion.

HSPICE or HSPICE RF generates a *.ft#* file and the listing file for each FFT output variable that contains data to display in FFT analysis waveforms (such as in WaveView Analyzer). You can view the magnitude in dB, and the phase in degrees.

In the following sample FFT analysis *.lis* file output, the header defines parameters in the FFT analysis.

```
****** Sample FFT output extracted from the .lis file
fft test ... sine
****** fft analysis   tnom= 25.000   temp= 25.000
****** fft components of transient response v(1)
Window:   Rectangular
First Harmonic:   1.0000k
Start Freq:   1.0000k
Stop Freq:   10.0000k
dc component: mag(db)= -1.132D+02 mag= 2.191D-06 phase= 1.800D+02

frequency   frequency   fft_mag   fft_mag   fft_phase
index      (hz)   (db)      (deg)
2   1.0000k   0.   1.0000   -3.8093m
4   2.0000k   -125.5914   525.3264n   -5.2406
6   3.0000k   -106.3740   4.8007u   -98.5448
8   4.0000k   -113.5753   2.0952u   -5.5966
10   5.0000k   -112.6689   2.3257u   -103.4041
12   6.0000k   -118.3365   1.2111u   167.2651
14   7.0000k   -109.8888   3.2030u   -100.7151
16   8.0000k   -117.4413   1.3426u   161.1255
18   9.0000k   -97.5293   13.2903u   70.0515
20   10.0000k   -114.3693   1.9122u   -12.5492
total harmonic distortion =     1.5065m percent
```

The preceding example specifies a frequency of 1 kHz, and a `THD` up to 10 kHz, which corresponds to the first ten harmonics.

The highest FFT output frequency might not match the specified `FMAX`, due to adjustments that HSPICE or HSPICE RF makes.

Table 57 describes the output of the FFT analysis.

Table 57 .FFT Output Description

Column Heading	Description
Frequency Index	Runs from 1 to NP/2, or the corresponding index for FMIN and FMAX. The DC component, corresponding to the 0 index, displays independently.
Frequency	The actual frequency, associated with the index.
fft_mag (dB), fft_mag	The first FFT magnitude column is in dB. The second FFT magnitude column is in units of the output variable. HSPICE or HSPICE RF normalizes the magnitude, unless you specify UNORM format.
fft_phase	Associated phase, in degrees.

Notes:

- Use the following formula as a guideline to specify a frequency range for FFT output:

  frequency increment = 1.0/(STOP - START)

  Each frequency index is a multiple of this increment. To obtain a finer frequency resolution, maximize the duration of the time window or specify more time points (larger NP).
- `FMIN` and `FMAX` have no effect on the .ft0, .ft1, ..., .ftn files.
- If `FFTOUT` is specified in a `.OPTION` statement, HSPICE can print results of `THD`, `SNR`, `SFDR`, `SNDR`, and `ENOB` and then sort the harmonics of fundamental by magnitude size.

  Assume that freq=f_0 is the fundamental frequency.

  - `THD` is total harmonic distortion, which can be computed with the following formula:

$$THD = \frac{\sqrt{(sum(mag(n \cdot f_0) \cdot mag(n \cdot f_0)))}}{mag(f_0)}$$

- `SNR` is the ratio of signal to noise, which can be computed with the following formula:

$$SNR = 10\log\left[\frac{(mag(f_0) \cdot mag(f_0))}{sum(mag(f) \cdot mag(f))}\right]$$

The f loop over the whole spectrum except f_0 and its harmonics. If FMIN/FMAX is specified, f loops over the spectrum between FMIN and FMAX except f_o and its harmonics, as well as all the frequency components above FMAX.

- `SFDR` is the spurious-free dynamic range and is defined as the distance from the fundamental input signal to the highest spur (in dB). `SFDR` involves both magnitude distance and frequency separation.
- `SNDR` is the signal to noise and distortion ratio, which is the level of the fundamental divided by the square root of the sum of squares of all frequency components other than the fundamental frequency. It can be computed with the following formula:

$$SNDR = 10\log\left[\frac{mag(f_0) \cdot mag(f_0)}{sum(mag(f) \cdot mag(f))}\right]$$

- `ENOB` is the effective number of bits, which can be computed with the following formula:

$$ENOB = \frac{(SNDR - 1.76\text{db})}{6.02}$$

The f loops over the whole spectrum except f_0. If FMIN/FMAX is specified, f loops over the spectrum between FMIN and FMAX except f_o. All the frequency components above FMAX are also looped by f.

Measuring FFT Output Information

All of the FFT output information above can be measured using the following syntax:

Measuring frequency component at certain frequency point:

```
.MEASURE FFT result
+ Find [vm|vp|vr|vi|vdb|im|ip|ir|ii|idb](signal) AT=freq
```

Measuring THD of a signal spectrum up to a certain harmonic:

```
.MEASURE FFT result THD signal_name [NBHARM=num]
```

Default: `NBHARM`=maximum harmonic in FFT result.

Measuring SNR/SNDR/ENOB of a signal up to a certain frequency point:

```
.MEASURE FFT result [SNR|SNDR|ENOB] signal_name
+ [NBHARM=num|MAXFREQ=val]
```

Default: `NBHARM`=maximum harmonic in FFT result. All the frequency components above NBHARM are considered to be noise.

`MAXFREQ`=maximum frequency in FFT result. All the frequency components above MAXFREQ are considered to be noise.

Measuring SFDR of a signal from `minfreq` to `maxfreq`:

```
.MEASURE FFT result SFDR signal_name [MAXFREQ=val][MINFREQ=val]
```

Default: `MINFREQ=0`.

`MAXFREQ`=Maximum frequency in FFT result.

Syntax to perform FFT measurements from previous simulation results:

```
hspice -i *.tr0 -meas measure_file
```

For more information, see .MEASURE FFT in the *HSPICE Reference Manual: Commands and Control Options*.

AM Modulation

This example is based on demonstration netlist exam1.sp, which is available in directory $*installdir*/demo/hspice/fft, shows a 1 kHz carrier (FC), which a 100 Hz signal (FM) modulates.

```
AM Modulation

.OPTION post
.PARAM sa=10 offset=1 fm=100 fc=1k td=1m
VX 1 0 AM(sa offset fm fc td)
Rx 1 0 1

.TRAN 2u 50m
.FFT V(1) START=10m STOP=40m FMIN=833 FMAX=1.16K
.END
```

The following equation describes the voltage at node 1, which is an AM signal:

$$v(1) = sa \cdot (offset + \sin(\omega_m(Time - td))) \cdot \sin(\omega_c(Time - td))$$

You can expand the preceding equation, as follows.

$$v(1) = (sa \cdot offset \cdot \sin(\omega_c(Time - td)) + 0.5 \cdot sa \cdot \cos((\omega_c - \omega_m)(Time - td)))$$

$$- 0.5 \cdot sa \cdot \cos((\omega_c + \omega_m)(Time - td))$$

where

$$\omega_c = 2\pi f_c$$

$$\omega_f = 2\pi f_m$$

The preceding equations indicate that v(1) is a summation of three signals, with the following frequency:

f_c, $(f_c - f_m)$, and $(f_c + f_m)$

This is the carrier frequency and the two sidebands.

See also Behavioral Amplitude Modulator on page 679.

Graphical Output

Figure 60 and Figure 61 on page 431 display the results.

- Figure 60 shows the time-domain curve for node 1.
- Figure 61 shows the frequency-domain components, for the magnitude of node 1.

The carrier frequency is 1 kHz, with two sideband frequencies 100 Hz apart.

The third, fifth, and seventh harmonics are more than 100 dB below the fundamental, indicating excellent numerical accuracy. The time-domain data contains an integer multiple of the period, so you do not need to use windowing.

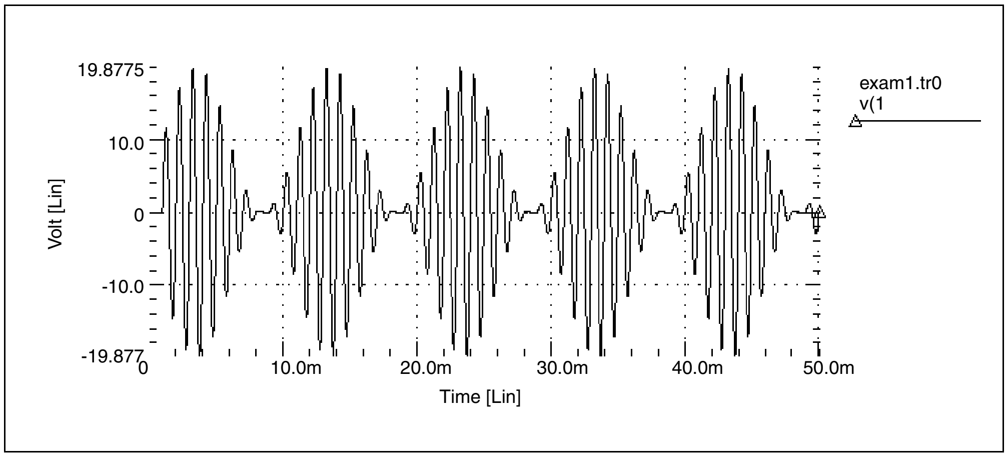


Figure 60 AM Modulation

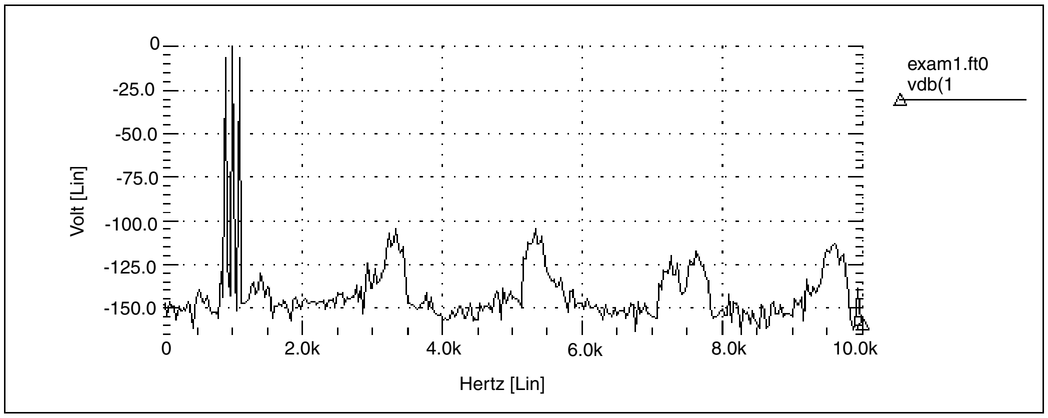


Figure 61 AM Modulation Spectrum

Balanced Modulator and Demodulator

Demodulation, or detection, recovers a modulating signal from the modulated output voltage. The netlist, in the Input and Output Listing section that follows, shows this process. This example uses HSPICE or HSPICE RF behavioral models, and FFT analysis, to confirm the validity of the process, in the frequency domain.

The low-pass filter uses the Laplace element. This filter introduces delay in the output signal, which causes spectral leakage if you do not use FFT windowing. However, if you use window weighting to perform FFT, you eliminate most of the spectral leakage. The THD of the two outputs, shown in “Input and Output Listing,” verifies this. HSPICE or HSPICE RF expects a 1 kHz output signal, so specify a 1 kHz frequency in the `.FFT` command. Also specify FMAX, to provide the first few harmonics in the output listing, for THD calculations.

Input and Output Listing

The sample input and output listing files are located in the following directory: $installdir/demo/hspice/fft/balance.sp

Figure 62 through Figure 70 show the signals, and their spectral content. The modulated signal contains only the sum, and the difference of the carrier frequency and the modulating signal (1 kHz and 10 kHz). At the receiver end, this example recovers the carrier frequency, in the demodulated signal. This example also shows a 10 kHz frequency shift, in the above signals (to 19 kHz and 21 kHz).

A low-pass filter uses a second-order Butterworth filter, to extract the carrier frequency. A Harris window significantly improves the noise floor in the filtered output spectrum, and reduces `THD` in the output listing (from 9.23% to 0.047%). However, this example needs a filter with a steeper transition region, and better delay characteristics, to suppress modulating frequencies below -60 dB. Figure 65 is a normalized filtered output signal waveform.

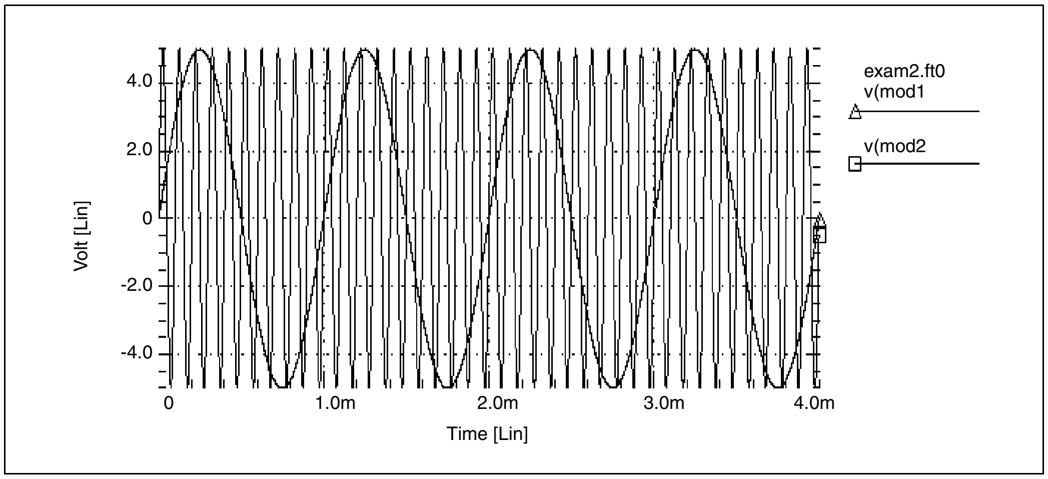


Figure 62 Modulating and Modulated Signals

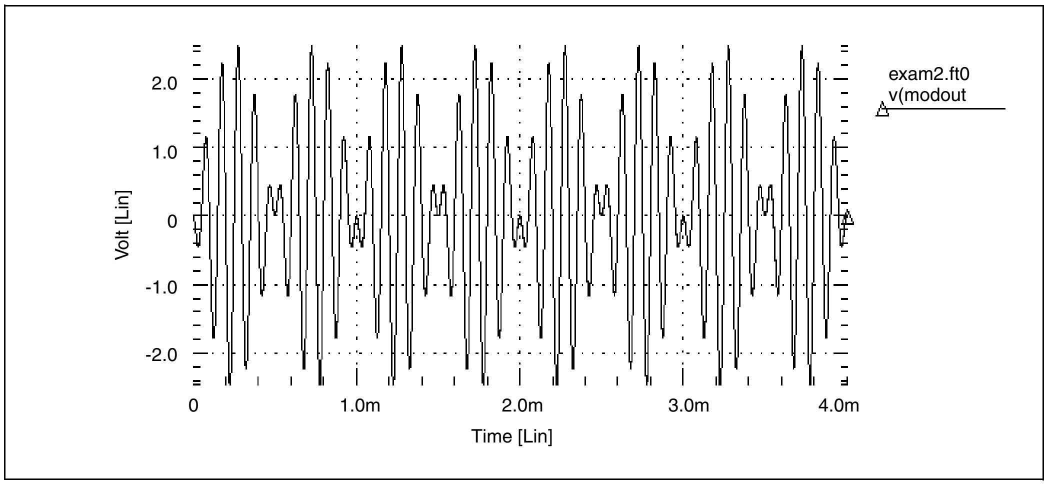


Figure 63 Modulated Signal

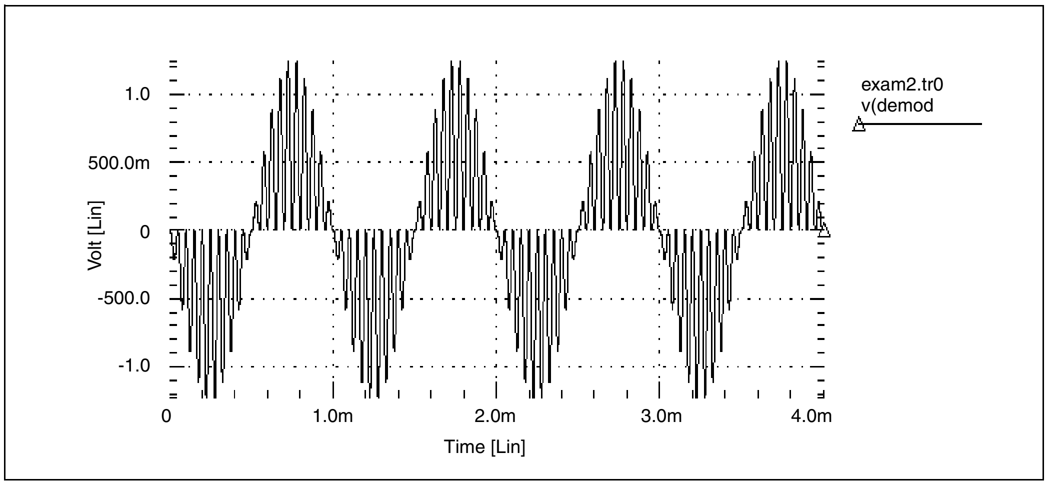


Figure 64 Demodulated Signal

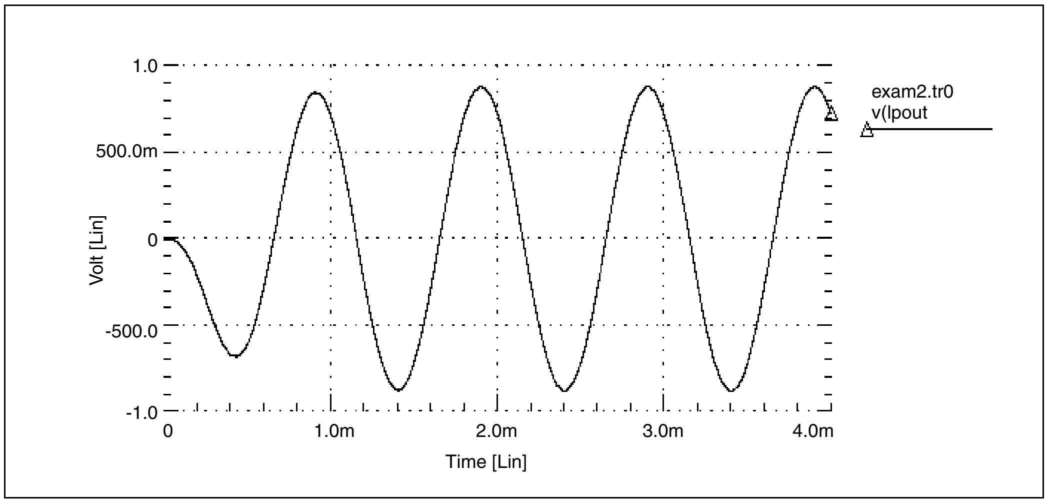


Figure 65 Filtered Output Signal

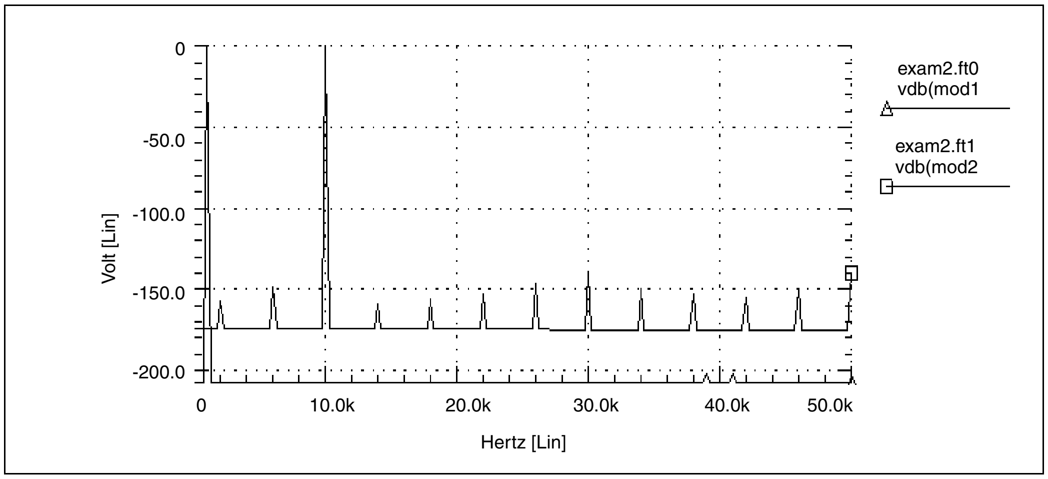


Figure 66 Modulating and Modulated Signal Spectrum

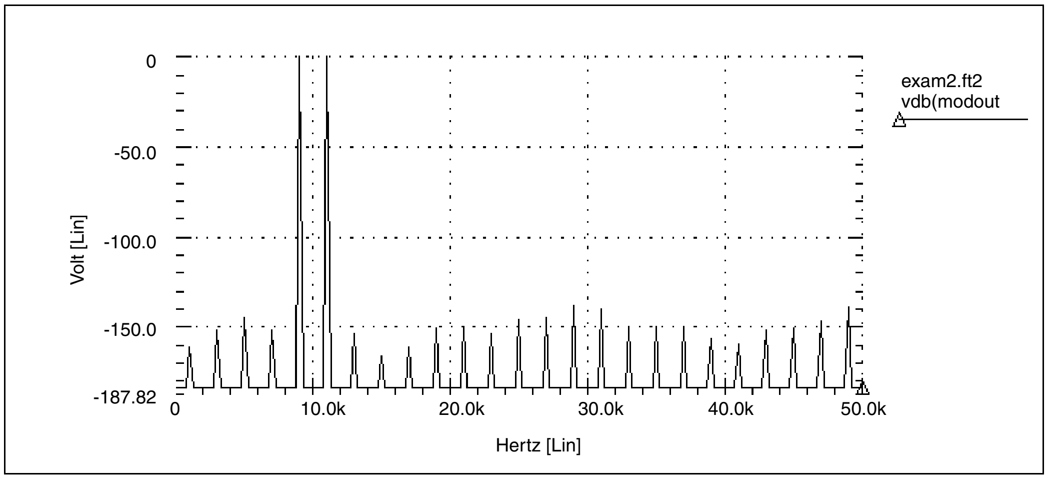


Figure 67 Modulated Signal Spectrum

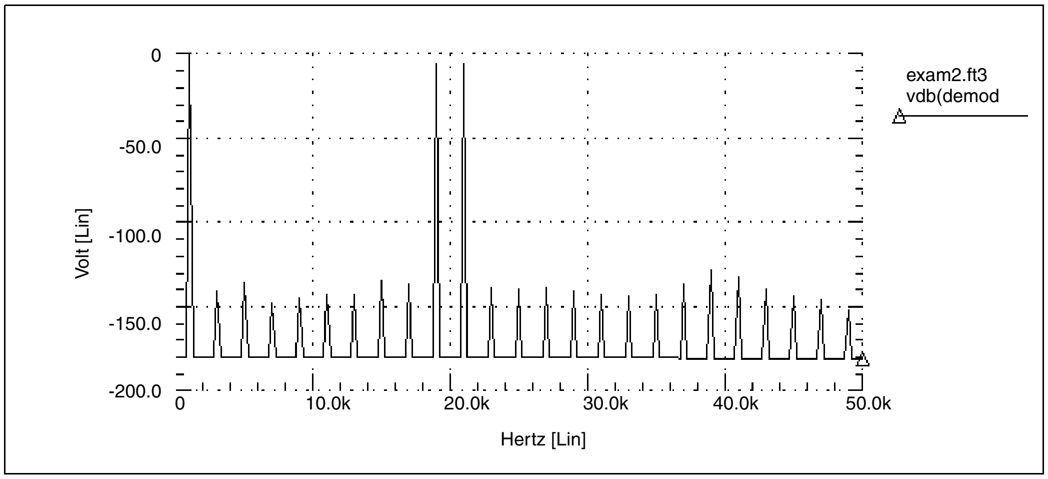


Figure 68 Demodulated Signal Spectrum

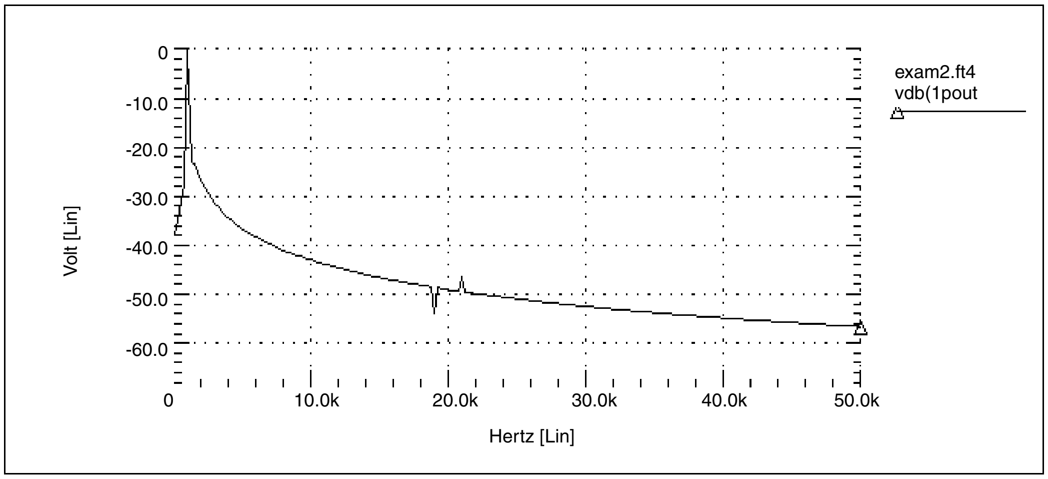


Figure 69 Filtered Output Signal (no window)

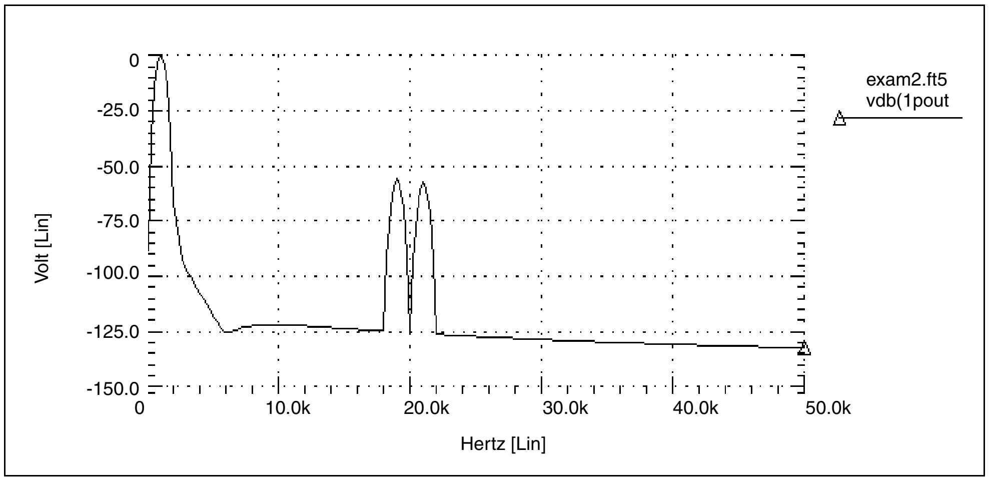


Figure 70   Filtered Output Signal (Blackman-Harris window)

Signal Detection Test Circuit

This example is a high-frequency mixer test circuit. It illustrates the effect of using a window to detect a weak signal, in the presence of a strong signal that is at a nearby frequency. This example adds two high-frequency signals, with a 40 dB separation (amplitudes are 1.0 and 0.01).

Input Listing

The sample input listing file is located in the following directory:
$installdir/demo/hspice/fft/exam3.sp

Output

Figure 71 shows the rectangular window. Compare this with the spectra of the output for all FFT window types, as shown in Figure 72 through Figure 78. Without windowing, HSPICE or HSPICE RF does not detect the weak signal because of spectral leakage.

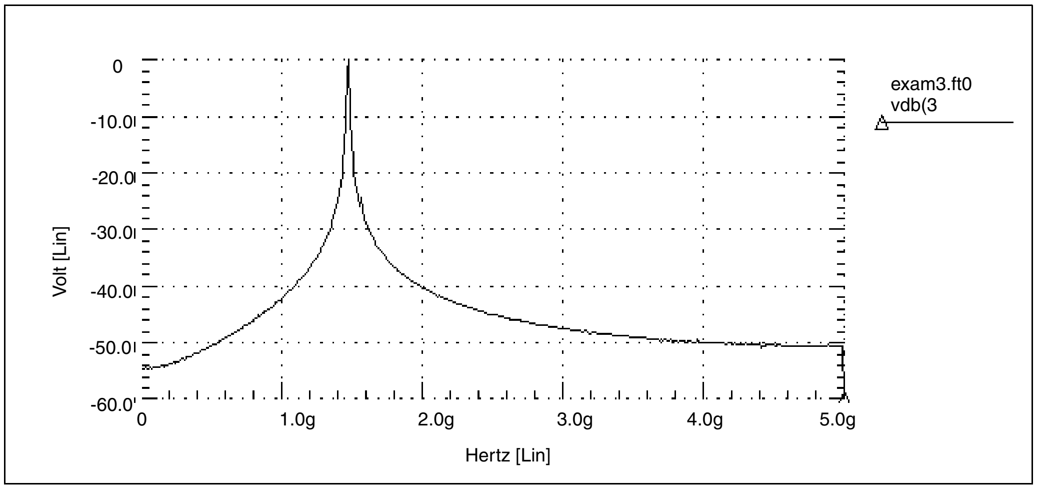


Figure 71 Mixer Output Spectrum, Rectangular Window

- In the Bartlett window (Figure 72), the noise floor increases dramatically, compared to the rectangular window (from -55, to more than -90 dB).
- The cosine windows (Hanning, Hamming, Blackman, and Blackman-Harris) all produce better results than the Bartlett window. However, the Blackman-Harris window provides the highest degree of separation for the two tones, and the lowest noise floor.
- The final two windows (Figure 77 and Figure 78) use the `ALFA=3.0` parameter, which is the default value in HSPICE or HSPICE RF. These two windows also produce acceptable results, especially the Kaiser-Bessel window, which sharply separates the two tones, and has a noise floor of almost -100-dB.

Processing such high frequencies, as demonstrated in this example, shows the numerical stability and accuracy of the FFT spectrum analysis algorithms, in HSPICE or HSPICE RF.

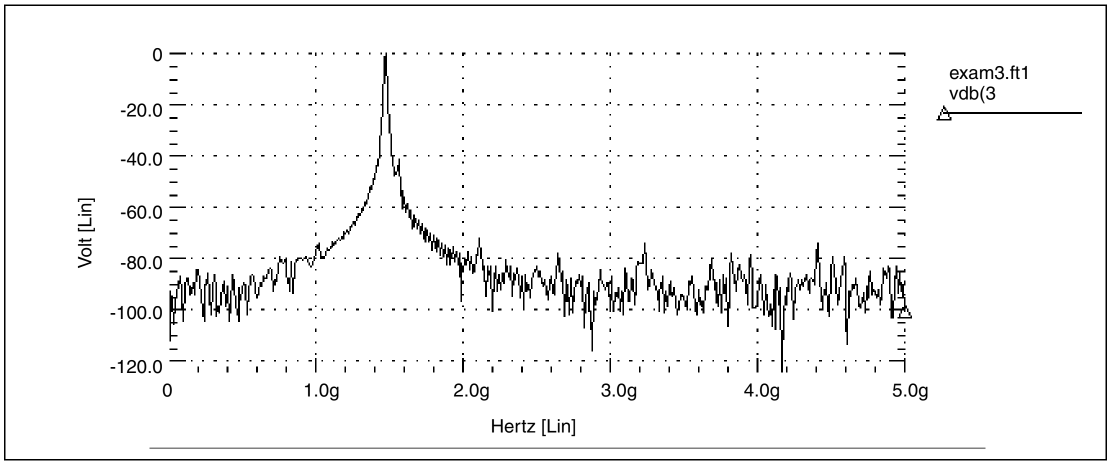


Figure 72   Mixer Output Spectrum, Bartlett Window

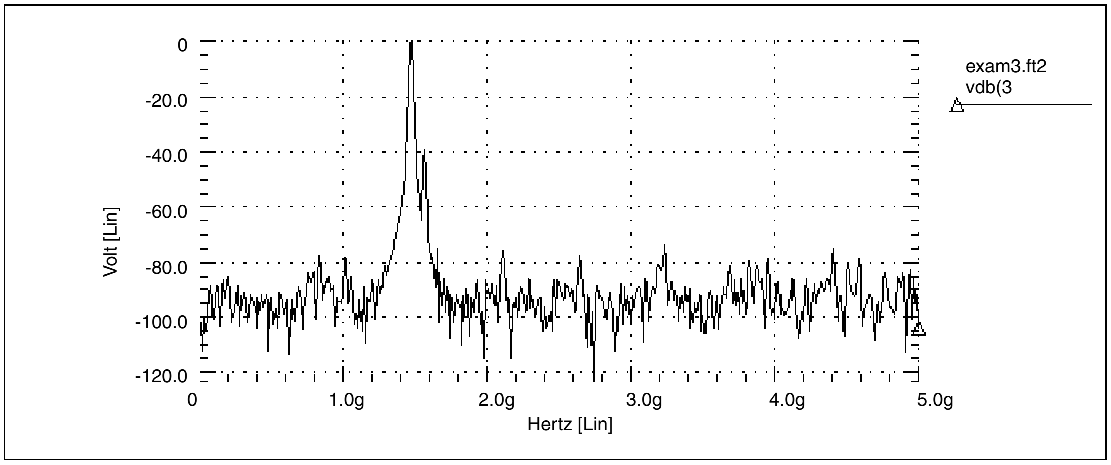


Figure 73   Mixer Output Spectrum, Hanning Window

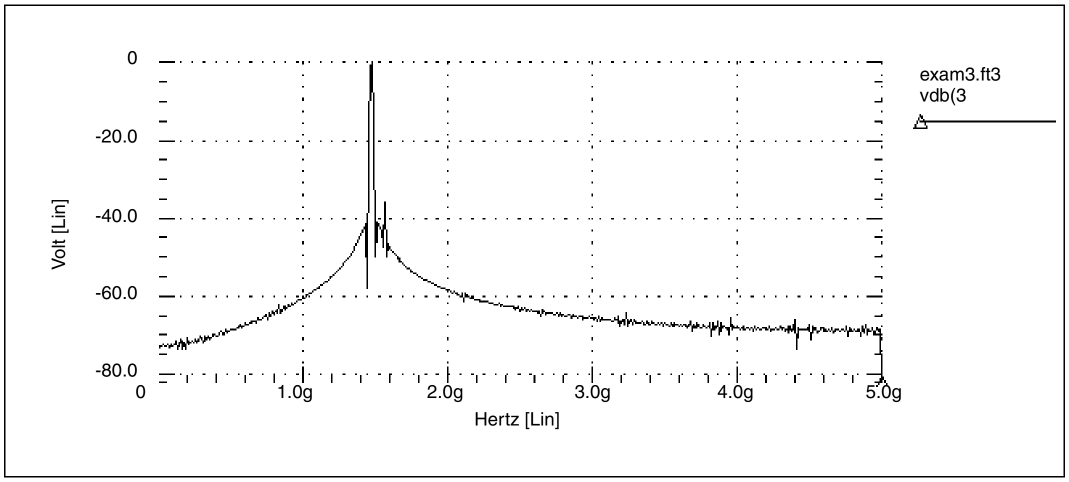


Figure 74   Mixer Output Spectrum, Hamming Window

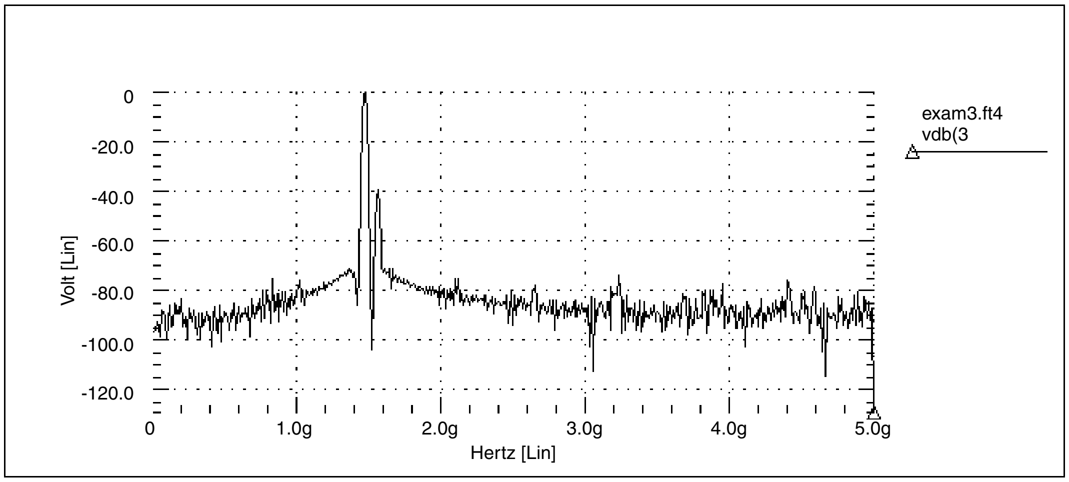


Figure 75   Mixer Output Spectrum, Blackman Window

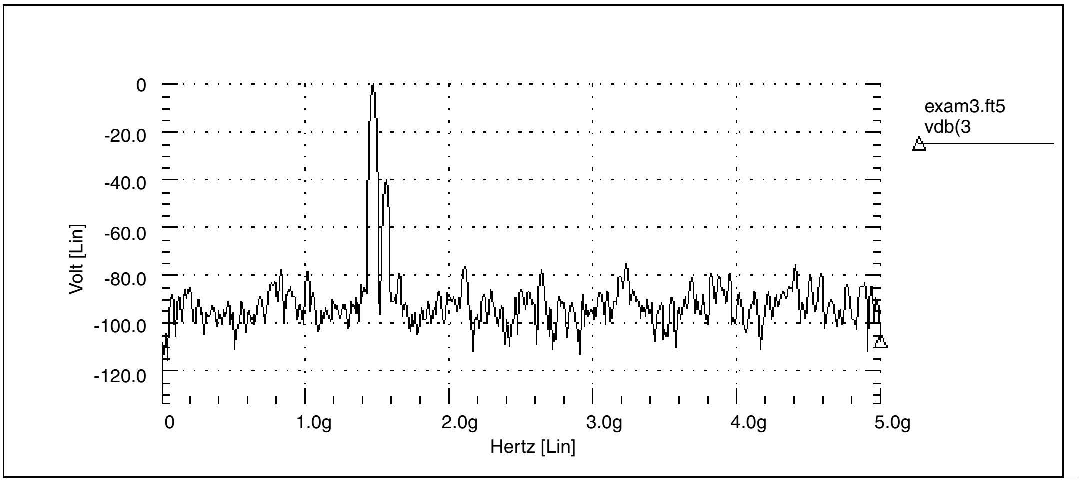


Figure 76 Mixer Output Spectrum, Blackman-Harris Window

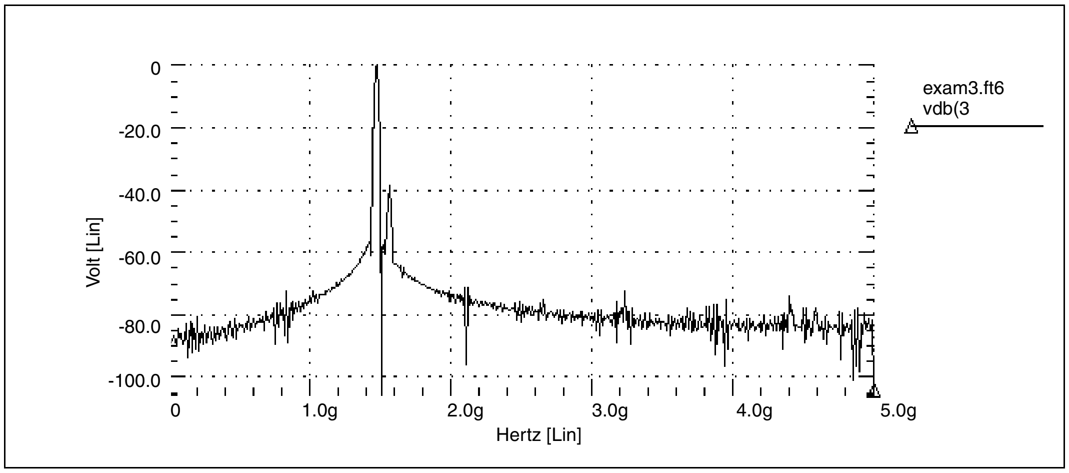


Figure 77 Mixer Output Spectrum, Gaussian Window

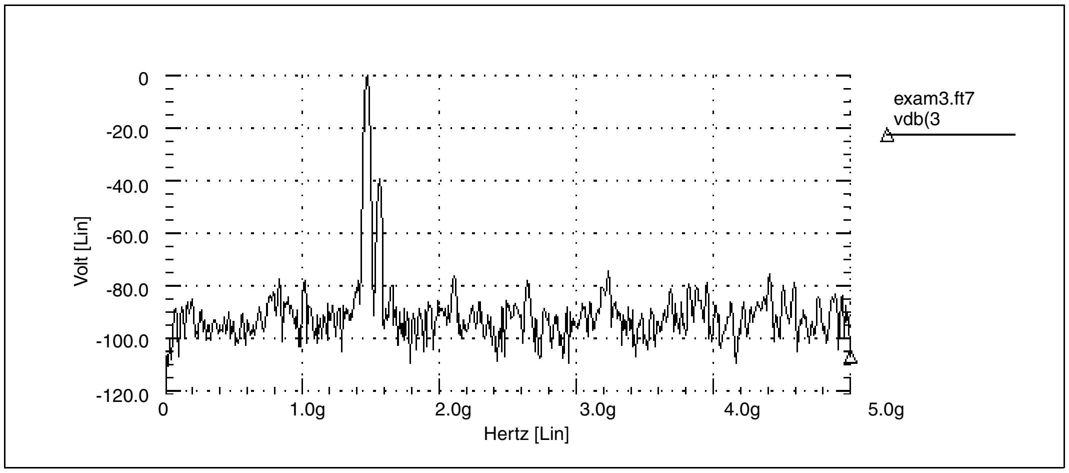


Figure 78 Mixer Output Spectrum, Kaiser-Bessel Window

References

[1] For an excellent discussion of DFT windows, see Fredric J. Harris, “On the Use of Windows for Harmonic Analysis with Discrete Fourier Transform,” *Proceedings of the IEEE*, Vol. 66, No. 1, Jan. 1978.

16

AC Small-Signal and Noise Analysis

Describes how to perform AC and noise small signal analyses in HSPICE.

This chapter covers AC small signal analysis, AC analysis of an RC network, noise analysis, and other AC analysis statements. For information on output variables, see AC Analysis Output Variables on page 310.

For descriptions of individual HSPICE commands referenced in this chapter, see Chapter 2, HSPICE and HSPICE RF Netlist Commands in the *HSPICE Reference Manual: Commands and Control Options*.

These topics are covered in the following sections:

Using the .AC Statement

You can use the `.AC` statement for the following applications:

- Single/double sweeps
- Sweeps using parameters
- `.AC` analysis optimization
- Random/Monte Carlo analyses

For `.AC` command syntax, see the .AC command in the *HSPICE Reference Manual: Commands and Control Options*.

.AC Control Options

You can use the following `.AC` control options when performing an AC analysis:

ABSH	ACOUT	DI
MAXAMP	RELH	UNWRAP

For syntax descriptions for these options, see Chapter 3, HSPICE and RF Netlist Simulation Control Options in the *HSPICE Reference Manual: Commands and Control Options*.

.AC Command Examples

Example 1

```
.AC DEC 10 1K 100MEG
```

This example performs a frequency sweep by 10 points per decade from 1kHz to 100MHz.

Example 2

```
.AC LIN 100 1 100HZ
```

This example runs a 100-point frequency sweep from 1- to 100-Hz.

Example 3

```
.AC DEC 10 1 10K SWEEP cload LIN 20 1pf 10pf
```

This example performs an AC analysis for each value of `cload`. This results from a linear sweep of `cload` between 1- and 10-pF (20 points), sweeping the frequency by 10 points per decade from 1- to 10-kHz.

Example 4

```
.AC DEC 10 1 10K SWEEP rx POI 2 5k 15k
```

This example performs an AC analysis for each value of `rx`, `5k` and `15k`, sweeping the frequency by 10 points per decade from 1- to 10-kHz.

Example 5

```
.AC DEC 10 1 10K SWEEP DATA=datanm
```

This example uses the `.DATA` statement to perform a series of AC analyses, modifying more than one parameter. The `datanm` file contains the parameters.

Example 6

```
.AC DEC 10 1 10K SWEEP MONTE=30
```

This example illustrates a frequency sweep and a Monte Carlo analysis with 30 trials.

Example 7

```
AC DEC 10 1 10K SWEEP MONTE=10 firstrun=15
```

This example illustrates a frequency sweep and a Monte Carlo analysis from the 15th to the 24th trials.

Example 8

```
.AC DEC 10 1 10K SWEEP MONTE=list(10 20:30 35:40 50)
```

This example illustrates a frequency sweep and a Monte Carlo analysis at 10th trial and then from the 20th to 30th trial, followed by the 35th to 40th trial and finally at 50th trial.

AC Small Signal Analysis

AC small signal analysis in HSPICE computes AC output variables as a function of frequency (see Figure 79 on page 448). 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.

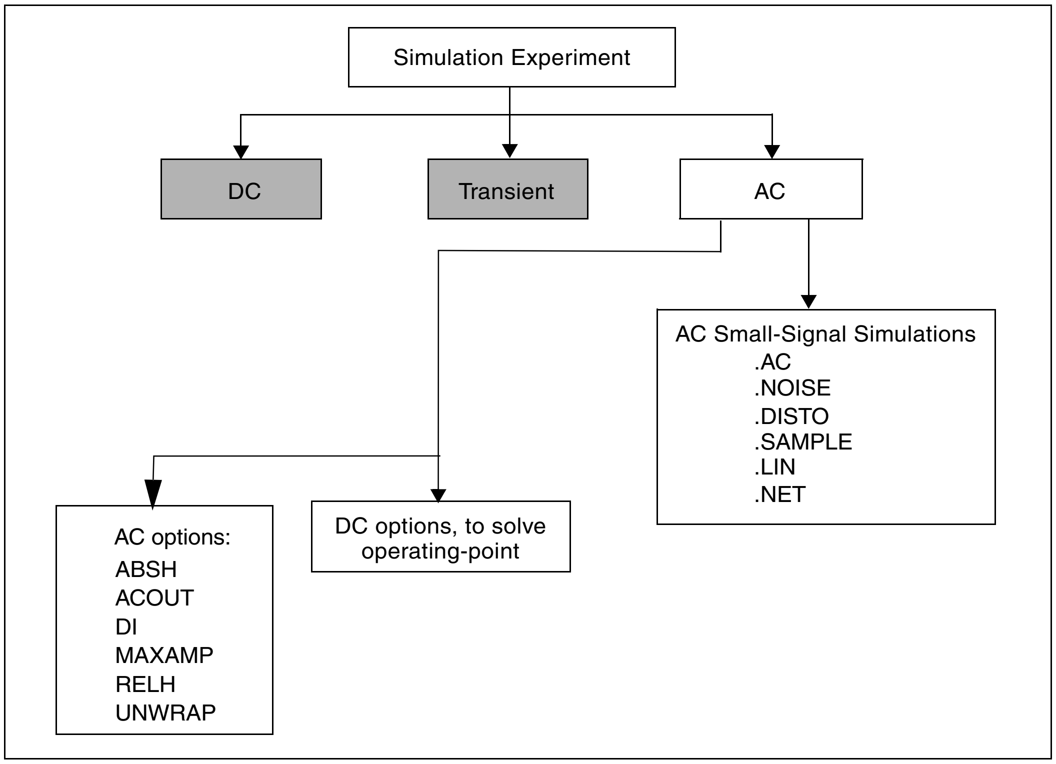


Figure 79 AC Small Signal Analysis Flow

In HSPICE, the output of AC Analysis includes voltages and currents.

HSPICE converts capacitor and inductor values to their corresponding admittances:

$y_C = j\omega C$ for capacitors

$y_L = \frac{1}{j\omega L}$ for inductors

Resistors can have different DC and AC values. If you specify `AC=<value>` in a resistor statement, HSPICE 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 the *HSPICE Elements and Device Models Manual*. MOSFET

AC-equivalent circuit models are described in the *HSPICE Elements and Device Models Manual*.

The AC analysis statement can sweep values for:

- Frequency.
- Element.
- Temperature.
- Model parameter.
- 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.

You can use the `.AC` statement in several different formats, depending on the application. You can also use the `.AC` statement to perform data-driven analysis in HSPICE.

AC Analysis of an RC Network

Figure 80 on page 450 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.

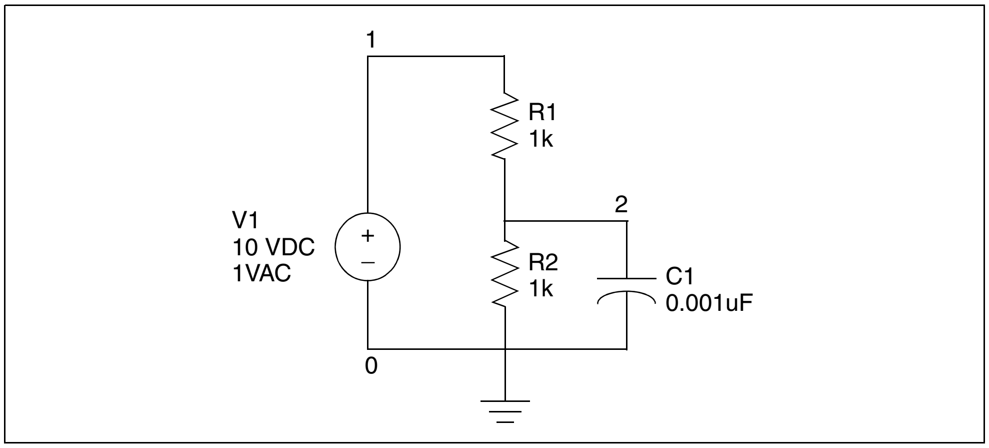


Figure 80 RC Network Circuit

The netlist for this RC network is based on demonstration netlist quickAC.sp, which is available in directory $<installdir>/demo/hspice/apps:

```
A SIMPLE AC RUN
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 1MEG
.PRINT AC V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

Follow this procedure 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 WaveView.
5. From the File menu, select File > Import > Waveform File.
6. Select the *quickAC.ac0* file from the Open: Waveform Files window.
7. Display the voltage at node 2 by using a log scale on the x-axis.

Figure 81 on page 451 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.

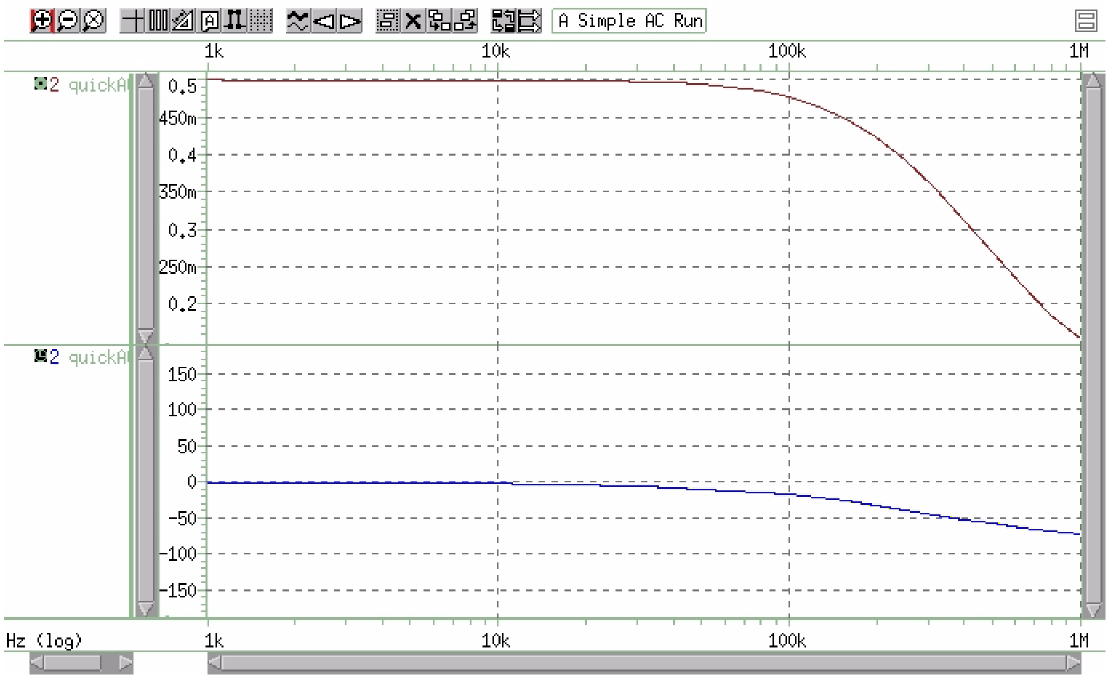


Figure 81 RC Network Node 2 Frequency Response

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.

Using .NOISE for Small-Signal Noise Analysis

A circuit noise analysis can be performed associated with a small-signal `.AC` analysis. The `.NOISE` command will activate a noise analysis that calculates the output noise generated based on the contributions from all noise sources within the circuit. This noise may be from passive elements, such as thermal (Johnson) noise in resistors, or from sources such as shot, channel, and flicker noise present within transistors. Most transistors will have several noise sources. For descriptions of noise models for each device type, see the HSPICE Reference Manual: Elements and Device Models. In most cases, the individual noise sources in HSPICE lack statistical correlation, and this allows their contributions to output noise to be computed independently. The total output noise voltage is the RMS sum of the individual noise contributions:

$$onoise = \sqrt{\sum_{k=0}^{N} |Z_k|^2 \overline{|i_{nk}|^2}}$$

Where,

$onoise$ is the output noise spectral density ($V/\sqrt{Hz}$) at the AC analysis frequency.

$\overline{|i_{nk}|^2}$ is the mean-squared noise spectral density (A^2/Hz) for each noise current source due to thermal, shot, flicker, or other noise.

Z_k is the equivalent transimpedance between each noise current source and the output.

N is the number of noise sources associated with all circuit elements.

This analysis will be performed for every frequency specified with the `.AC` command. The output for noise analysis is specified in the `.NOISE` syntax:

Basic Syntax:

```
.NOISE v(out <,ref>) src <interval>
```

The output noise (`onoise`) voltage is computed at the out node specified; if the (optional) `ref` node is also given, the output is taken as the differential output noise voltage `v(out,ref)`. Noise analysis requires the specification of an independent input source (`src`). This allows the calculation of the equivalent input noise given by

$$inoise = \frac{onoise}{|G|}$$

Where,

$inoise$ is the equivalent input noise spectral density at the input source. G is the gain between the input source (*src*) and the output.

The .NOISE analysis can also generate a summary for how each noise generator within the circuit will contribute to output noise. Specify an integer value for *interval* to include a device noise summary for every *interval* frequency points in the HSPICE output listing. No summary is included unless *interval* is specified. The .NOISE analysis will also compute the total integrated noise over the AC frequency range, which will also be included in the output listing. The output summary will include values for the device noise sources given in Table 58 on page 454 — Table 60 on page 455.

To request `.NOISE` analysis results (magnitude and decibel) with `.print/ .probe` use:

```
.probe noise onoise onoise(m) onoise(db)
.probe noise inoise inoise(m) inoise(db)
```

Results will be included in the **.ac0* file. Output noise voltage or current units are either $V/\sqrt{Hz}$ or $A/\sqrt{Hz}$, respectively. Device-level noise source contributions will also be included in the **.ac0* file (unless you have set `.option probe=1`). The naming convention and units for device level noise contributions is also shown in the following tables.

To ensure that device noise models will be included in the analysis, verify that noise parameters are being set in your transistor models. Include values for AF and KF, for example, if you wish to activate flicker noise models for your devices.

See also, Using Noise Analysis Results as Input Noise Sources in the *HSPICE User Guide: RF Analysis*.

For a complete description of the `.NOISE` command syntax and examples, see the .NOISE command in the *HSPICE Reference Manual: Commands and Control Options*. Note that the .NOISE analysis requires an `.AC` statement, and that if more than one `.NOISE` statement is included, HSPICE will run only the last statement.

Table 58 .NOISE Measurements Available for MOSFETs

.ac	.lis	Unit	Description
nd	rd	$\frac{V^2}{Hz}$	Output thermal noise due to drain resistor
ns	rs	$\frac{V^2}{Hz}$	Output thermal noise due to source resistor
ni	id	$\frac{V^2}{Hz}$	Output channel thermal noise
nf	fn	$\frac{V^2}{Hz}$	Output flicker noise
ntg	total	$\frac{V^2}{Hz}$	Total output noise: TOT=RD + RS + ID + FN

Table 59 .NOISE Measurements Available for BJTs

.ac	.lis	Unit	Description
rb	rb	$\frac{V^2}{Hz}$	Output thermal noise due to base resistor
rc	rc	$\frac{V^2}{Hz}$	Output thermal noise due to collector resistor
re	re	$\frac{V^2}{Hz}$	Output thermal noise due to emitter resistor

Table 59 .NOISE Measurements Available for BJTs (Continued)

.ac	.lis	Unit	Description
nb	ib	$\frac{V^2}{Hz}$	Output noise due to base shot noise source
nc	ic	$\frac{V^2}{Hz}$	Output thermal noise due to collector shot noise source
nf	fn	$\frac{V^2}{Hz}$	Output noise due to flicker noise source
nt	total	$\frac{V^2}{Hz}$	Total output noise: TOT=RB + RC + RE + IB + IC + FN

Table 60 .NOISE Measurements Available for Diodes

.ac	.lis	Unit	Description
nr	rs	$\frac{V^2}{Hz}$	Output noise due to diode series resistance
ni	id	$\frac{V^2}{Hz}$	Output noise due to shot noise
nf	fn	$\frac{V^2}{Hz}$	Output noise due to flicker noise
nt	total	$\frac{V^2}{Hz}$	Total output noise: TOT=RS + ID + FN

Other AC Analysis Statements

The following sections describe the commands you can use to perform other types of AC analyses:

- Using .DISTO for Small-Signal Distortion Analysis on page 456
- Using .SAMPLE for Noise Folding Analysis on page 456

Use the `.NOISE` and `.AC` statements to control the noise analysis of the circuit.

Using .DISTO for Small-Signal Distortion Analysis

The `.DISTO` statement computes the distortion characteristics of the circuit in an AC small-signal, sinusoidal, steady-state analysis. HSPICE computes and reports five distortion measures at the specified load resistor. The analysis is performed assuming that one or two signal frequencies are imposed at the input. The first frequency, F1 (used to calculate harmonic distortion), is the nominal analysis frequency set by the `.AC` statement frequency sweep. The optional second input frequency, F2 (used to calculate intermodulation distortion), is set implicitly by specifying the `skw2` parameter, which is the ratio F2/F1.

For command syntax and examples, see the .DISTO command in the *HSPICE Reference Manual: Commands and Control Options*.

Using .SAMPLE for Noise Folding Analysis

For data acquisition of analog signals, data sampling noise often needs to be analyzed. This is accomplished with the `.SAMPLE` statement used in conjunction with the `.NOISE` and `.AC` statements. The SAMPLE analysis performs a simple noise folding analysis at the output node.

For the syntax and description of the `.SAMPLE` statement, see the .SAMPLE command in the *HSPICE Reference Manual: Commands and Control Options*.

17

Linear Network Parameter Analysis

Describes how to perform an AC sweep to extract small-signal linear network parameters in HSPICE and HSPICE RF.

The chapter covers `.LIN` analysis, RF measurements from `.LIN`, extracting mixed-mode S (scattering) and noise parameters, and additional measurements.

For descriptions of individual commands referenced in this chapter, see the *HSPICE Reference Manual: Commands and Control Options*.

These topics are covered in the following sections:

- .LIN Analysis
- Additional Measurements From .LIN
- Extracting Mixed-Mode Scattering (S) Parameters

.LIN Analysis

The `.LIN` command extracts noise and linear transfer parameters for a general multi-port network.

When used with the `.AC` command, `.LIN` makes available a broad set of linear port-wise measurements:

- Multi-port scattering [S] parameters
- Noise parameters
- Stability factors
- Gain factors
- Matching coefficients

The `.LIN` analysis is similar to basic small-signal, swept-frequency .AC analysis, but it also automatically calculates a series of noise and small-signal transfer parameters between the terminals identified using port (P) elements.

HSPICE can output the result of group delay extraction and two-port noise analysis to either a *.sc** file, a Touchstone file, or a CITIfile.

The `.PRINT`/`.PROBE`/`.MEAS` output syntax for `.LIN` supports H (hybrid) parameters and S/Y/Z/H group delay.

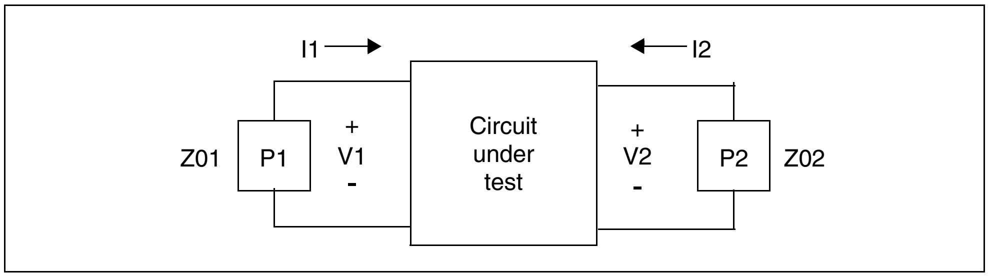


Figure 82 Basic Circuit in .LIN Analysis

Identifying Ports with the P-element

The `.LIN` command computes the S (scattering), Y (admittance), and Z (impedance) parameters directly based on the location of the port (P) elements in your circuit, and the specified values for their reference impedances.

The port element identifies the ports used in `.LIN` analysis.

A full description of the port element and syntax may be found in the Port Element section of the chapter on Sources and Stimuli in this user guide.

Using the P-element for Mixed-Mode Measurement

You can use a port element with three terminals as the port element for measuring the mixed mode S-parameters. Except for the number of external terminals, the syntax of the port element remains the same. The `.LIN` analysis function internally sets the necessary drive mode (common/differential) of these mixed mode port elements. For analyses other than the `.LIN` analysis (such as DC, AC, TRAN, and so on), the mixed-mode P-element acts as a differential driver that drives positive nodes with half of their specified voltage and the negative nodes with a negated half of the specified voltage. Figure 83 shows the block diagram of the mixed mode port element.

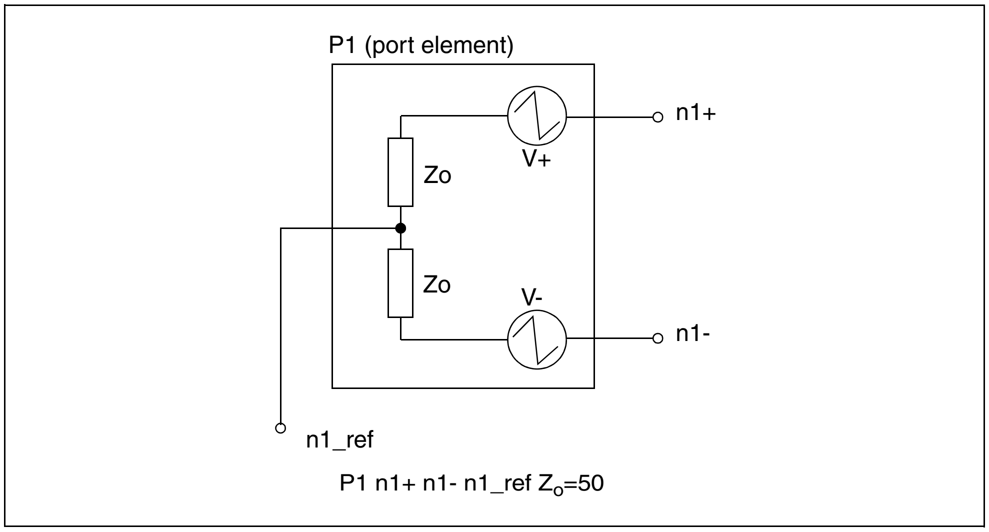


Figure 83   Mixed Mode Port Element

The port element can also be used as a signal source with a built in reference impedance. For further information on its use as a signal source, see Chapter 9, Sources and Stimuli.

.LIN Input Syntax

```
.LIN [sparcalc=1|0 [modelname = modelname]]
+ [filename = filename] [format=selem|citi|touchstone]
+ [noisecalc=1|0] [gdcalc=1|0]
+ [mixedmode2port=dd|dc|ds|cd|cc|cs|sd|sc|ss]
+ [dataformat=ri|ma|db]
+ [listfreq=(frequencies|none|all)]
+ [listcount=num] [listfloor=val] [listsources=1|0|on|off]
```

For argument descriptions, see the .LIN command in the *HSPICE Reference Manual: Commands and Control Options*.

.LIN Output Syntax

This section describes the syntax for the `.PRINT` and `.PROBE` statements used for LIN analysis.

.PRINT and .PROBE Statements

```
.PRINT AC Xmn | Xmn LINPARAM(TYPE) | X(m,n) |
+ X(m,n) LINPARAM(TYPE)> Hmn | Hmn(TYPE) |
+ H(m,n) | H(m,n)(TYPE)>
.PROBE AC Xmn | Xmn LINPARAM(TYPE) | X(m,n) |
+ X(m,n) LINPARAM(TYPE)> Hmn | Hmn(TYPE) |
+ H(m,n) | H(m,n)(TYPE)>
```

Argument	Description
Xmn, X(m,n)	One of these parameter types: ▪ S (scattering parameters) ▪ Y (admittance parameters) ▪ Z (impedance parameters) ▪ H (hybrid parameters) mn refers to a pair of port numbers, where m can be 1 or 2, and n can be 1 or 2.
Hmn, H(m,n)	Complex hybrid (H-) parameters. mn refers to a pair of port numbers, where m can be 1 or 2, and n can be 1 or 2. If m,n=0 or m,n>2, HSPICE issues a warning and ignores the output request. ▪ To calculate a one-port H parameter, you must specify at least one port (P) element. ▪ To calculate a two-port H parameter, you must specify two or more port (P) elements. For additional information, see Hybrid Parameter Calculations on page 462.

Argument	Description
LINPARAM	Two-port noise parameters: ▪ NFMIN (noise figure minimum) ▪ NF (Noise figure) ▪ VN2 (Equivalent input noise voltage squared ▪ IN2 (Equivalent input noise current squared) ▪ RHON (Correlation coefficient between input noise voltage and input noise current) ▪ RN (Noise equivalent resistance) ▪ GN (Noise equivalent conductance) ▪ ZCOR (Noise correlation impedance) ▪ YCOR (Noise correlation admittance) ▪ ZOPT (Optimum source impedance for noise) ▪ YOPT (Optimum source admittance for noise) ▪ GAMMA_OPT (source reflection coefficient that achieves the minimum noise figure) ▪ ZOPT (source impedance that achieves minimum noise) ▪ RN (noise equivalent resistance) ▪ K_STABILITY_FACTOR (Rollett stability factor) ▪ MU_STABILITY_FACTOR (Edwards & Sinsky stability factor) ▪ G_MAX (maximum available/operating power gain) ▪ G_MSG (Maximum stable gain) ▪ G_TUMAX (Maximum unilateral transducer power gain) ▪ G_U (Unilateral power gain) ▪ G_MAX_GAMMA1 (source reflection coefficient that achieves maximum available power gain) ▪ G_MAX_GAMMA2 (load reflection coefficient that achieves maximum operating power gain) ▪ G_MAX_Z1=Source impedance needed to realize G_MAX (complex, Ohms) ▪ G_MAX_Z2=Load impedance needed to realize G_MAX (complex, Ohms) ▪ G_MAX_Y1=Source admittance needed to realize G_MAX (complex, Siemens) ▪ G_MAX_Y2=Load admittance needed to realize G_MAX (complex, Siemens) ▪ G_AS (associate gain—maximum gain at the minimum noise figure) ▪ VSWR(n) (voltage standing-wave ratio at the n port) ▪ GD (group delay from port=1 to port=2) ▪ G_MSG (maximum stable gain) ▪ G_TUMAX (maximum unilateral transducer power gain) ▪ G_U (unilateral power gain)

Argument	Description
TYPE	Data type definitions: ■ R=Real ■ I=Imaginary ■ M=Magnitude ■ P=PD=Phase in degrees ■ PR=Phase in radians ■ DB=decibels

Examples

```
.print AC S11 S21(DB) S(2,3)(D) S(2,1)(I)
.print AC NFMIN GAMMA_OPT G_AS
.probe AC RN G_MAX ZOPT Y(3,1)(M) Y31(P)
```

Hybrid Parameter Calculations

The hybrid parameters are transformed from S-parameters:

- For a one-port circuit, the calculation is:

$$H_{11} = Z_{01}\frac{(1+S_{11})}{(1-S_{11})}$$

- For a two-port circuit, the calculation is:

$$H_{11} = Z_{01}\frac{(1+S_{11})(1+S_n)-S_{12}S_{21}}{(1-S_{11})(1+S_n)+S_{12}S_{21}}$$

$$H_{12} = \sqrt{\frac{Z_{02}}{Z_{01}}}\frac{2S_{12}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

$$H_{21} = \sqrt{\frac{Z_{02}}{Z_{01}}}\frac{-S_{21}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

$$H_{22} = \frac{1}{Z_{02}}\frac{(1-S_{11})(1-S_{22})-S_{12}S_{21}}{(1-S_{11})(1+S_{22})+S_{12}S_{21}}$$

For networks with more than two ports when computing the 1,2 H index parameters, HSPICE assumes that ports numbered 3 and above terminate in their port reference impedance (z0). The above two-port calculations therefore remain appropriate because S11, S12, S21, and S22 remain valid, and simulation can ignore higher order S-parameters.

Multi-Port Scattering (S) Parameters

S-parameters represent the ratio of incident and scattered (or forward and reflected) normalized voltage waves. Figure 84 shows a two-port network.

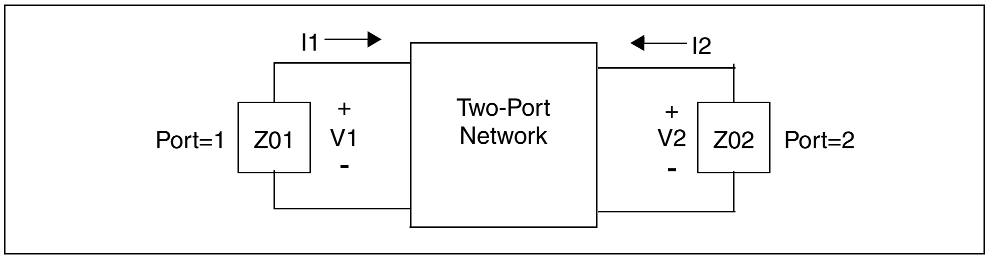


Figure 84 Two-Port Network

The following equations define the incident (forward) waves for this two-port network:

$$a_1 = \frac{v_1 + Z_{01} I_1}{2 \cdot \sqrt{Z_{01}}} \qquad a_2 = \frac{v_2 + Z_{02} I_2}{2 \cdot \sqrt{Z_{02}}}$$

The following equations define the scattered (reflected) waves for this two-port network:

$$b_1 = \frac{v_1 - Z_{01} I_1}{2 \cdot \sqrt{Z_{01}}} \qquad b_2 = \frac{v_2 - Z_{02} I_2}{2 \cdot \sqrt{Z_{02}}}$$

The following equations define the S-parameters:

$$S_{11} = \left.\frac{b_1}{a_1}\right|_{a_2 = 0} \qquad S_{12} = \left.\frac{b_1}{a_2}\right|_{a_1 = 0}$$

$$S_{21} = \left.\frac{b_2}{a_1}\right|_{a_2 = 0} \qquad S_{22} = \left.\frac{b_2}{a_2}\right|_{a_1 = 0}$$

Each S-parameter is a complex number, which can represent gain, isolation, or a reflection coefficient.

Example

The following examples show how you can represent a gain, isolation, or reflection coefficient:

```
.PRINT AC S11(DB) $ Input return loss
.PRINT AC S21(DB) $ Gain
.PRINT AC S12(DB) $ Isolation
.PRINT AC S22(DB) $ Output return loss
```

Two-Port Transfer and Noise Calculations

Two-port noise analysis is a linear AC noise analysis method that determines the noise figure of a linear two-port for an arbitrary source impedance.

Several output parameter measurements are specific to two-port networks. `.LIN` analysis supports two-port calculations for 3 or more ports if port=1 is the input and port=2 the output. All other ports terminate in their characteristic impedance. This is equivalent to operating on the two-port [S] submatrix extracted from the multi-port [S] matrix. This occurs for both signal and noise calculations. A warning appears if N<2 and you specified two-port quantities.

Noise and signal port-wise calculations do not require that port elements use a ground reference. You can therefore measure fully-differential circuits.

`.LIN` generates a set of noise parameters. The analysis assumes a noise model consisting of:

- A shunt current noise source, called In, at the input of a noiseless two-port linear network.
- A series voltage noise source, called Vn, at the input of a noiseless two-port linear network.
- A source with impedance, called Zs, that drives this two-port network.
- The two-port network drives a noiseless load, called Zl.

Equivalent Input Noise Voltage and Current

For each analysis frequency, HSPICE computes a noise equivalent circuit for a linear two-port. The noise equivalent circuit calculation results in an equivalent noise voltage and current, and their correlation coefficient.

- `VN2`: Equivalent input noise voltage squared (Real, V^2).
- `IN2`: Equivalent input noise current squared (Real, A^2).
- `RHON`: Correlation coefficient between the input noise voltage and the input noise current (complex, unitless).

Equivalent Noise Resistance and Conductance

These measurements are the equivalent resistance and conductance, which generate the equivalent noise voltage and current values at a temperature of T=290K in a 1Hz bandwidth.

- `RN`: Noise equivalent resistance (Real, Ohms)
- `GN`: Noise equivalent conductance (Real, Siemens)

Noise Correlation Impedance and Admittance

These measurements represent the equivalent impedance and admittance that you can insert at the input of the noise equivalent circuit to account for the correlation between the equivalent noise voltage and the current values.

- `ZCOR`: Noise correlation impedance (Complex, Ohms)
- `YCOR`: Noise correlation admittance (Complex, Siemens)

Optimum Matching for Noise

These measurements represent the optimum impedance, admittance, and reflection coefficient value that result in the best noise performance (minimum noise figure).

- `ZOPT`: Optimum source impedance for noise (Complex, Ohms)
- `YOPT`: Optimum source admittance for noise (Complex, Siemens)
- `GAMMA_OPT`: Optimum source reflection coefficient (Complex, unitless)

Because `ZOPT` and `YOPT` can commonly take on infinite values when computing optimum noise conditions, calculations for optimum noise loading reflect the `GAMMA_OPT` coefficient.

Noise Figure and Minimum Noise Figure

Noise figure represents the ratio of the SNR (signal to noise ratio) at the input to the SNR at the output. You can set the input source impedance to `ZOPT` to obtain the minimum noise figure.

- `NFMIN`: Minimum noise figure (source at `ZOPT`) (real, unitless, power ratio)
- `NF`: Noise figure (value obtained with source impedance at Zc[1]) (real, unitless, power ratio)

Associated Gain

This measurement assumes that the input impedance matches the minimum noise figure, and the output matches the maximum gain.

`G_AS` is the associated gain—maximum power gain at `NFMIN` (real, power ratio)

Output Format for Group Delay in .sc* Files

All of the S/Y/Z/H parameters support a group delay calculation. The output syntax of `.PRINT` and `.PROBE` statement for group delay is:

```
Xmn(T) | Xmn(TD) | X(m,n)(T) | X(m,n)(TD)
```

- `X`=S, Y, Z, or H
- `m, n`=port number (1 or 2 for H parameter)

The output of group delay matrices in .sc* files lets HSPICE directly read back the group delay information, the tabulated data uses the regular HSPICE model syntax with the SP keyword:

```
*| group delay parameters
.MODEL SMODEL_GD SP N=2 SPACING=POI INTERPOLATION=LINEAR
+ MATRIX=NONSYMMETRIC VALTYPE=REAL
+ DATA=3
+         1e+08
+             0        5e-09
+         5e-09            0
+    {...data...}
```

`model name` is the model name of the S-parameters, plus `_GD`.

`GROUPDELAY=[0|1]` in the top line indicates group delay data:

```
*| N=2 DATA=3 NOISE=0 GROUPDELAY=1
*| NumOfBlock=1 NumOfParam=0
```

Output Format for Two-Port Noise Parameters in .sc* Files

Output of two-port noise parameter data in .sc* files shows the tabulated data with the following quantities in the following order:

```
*| 2-port noise parameters
*|   frequency Fmin[dB] GammaOpt(M) GammaOpt(P) RN/Z0
*|     {...data...}
```

In this syntax:

- `Fmin[dB]`=minimum noise figure (dB).
- `GammaOpt(M)`=magnitude of the reflection coefficient needed to realize Fmin.
- `GammaOpt(P)`=phase (degrees) of the reflection coefficient needed to realize Fmin.
- `RN/Z0`=normalized noise resistance.

Both GammaOpt and RN/Z0 values are normalized with respect to the characteristic impedance of the port=1 element (that is, Z01).

Noise Parameters in 2-Port and N-Port Networks

You can use the `.LIN` analysis to compute the equivalent two-port noise parameters or multi-port noise parameters for a network. The `noisecalc=1` option automatically calculates the following equivalent circuit values.

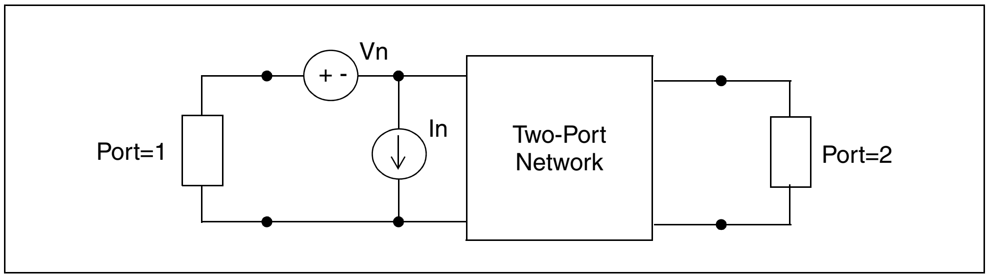


Figure 85 Noise Equivalent Circuit

- V_n is the equivalent input-referred noise voltage source.
- I_n is the equivalent input-referred noise current source.
- I_nV_n is their correlation.

The `noisecalc=2` option activates N-port noise analysis:

```
.LIN [sparcalc=[1|0] [modelname = modelname]]
+ [filename = filename] [format=selem|citi|touchstone]
+ [noisecalc=[2|1|0] [gdcalc=[1|0]]
+ [mixedmode2port=dd|dc|ds|cd|cc|cs|sd|sc|ss]>
+ [dataformat=ri|ma|db]
```

Invoking an N-port noise analysis produces an **.nc#* file to output the N-port noise correlation matrix in the following format, which is similar to a **.sc#* file.

```
*| N=2 DATA=1 NOISE=0 GROUPDELAY=0 COMPLEX_DATAFORMAT=RI
*| NumOfBlock=2 NumOfParam=1 ParamNames=res
*| ParamSweep res:50 100
*|------------------------------------
*| res=50
.MODEL NFQMODEL SP N=2 SPACING=POI INTERPOLATION=LINEAR
MATRIX=NONSYMMETRIC
+ DATA=1
+           100
+       0.457141             0     -0.457141            0
+      -0.457141             0      0.457141            0
*|------------------------------------
*| res=100
.MODEL NFQMODEL1 SP N=2 SPACING=POI INTERPOLATION=LINEAR
MATRIX=NONSYMMETRIC
+ DATA=1
+           100
+       0.514284             0     -0.514284            0
+      -0.514284             0      0.514284            0
```

The noise analysis data is output in the *.lis* file in the following format:

```
 Frequency = 100.000  Hz
 NF (mag) = 1.0+1.02857
 NF (dB) = 3.07189
 Element Contribution to NF (mag)    (dB)
 r1      TOT                1.02857  122.327m
```

All the 2-port noise parameters are deduced and output in the *.lis* file with `.PRINT/.PROBE` commands.

HSPICE can output the result of `.LIN` noise analysis to a *.sc**, Touchstone, or CITIfile. Note this limitation for Touchstone files: because Touchstone files currently provide only two-port noise parameters, this type of noise model only supports two-port S-parameter noise analysis for both passive and active systems.

Hybrid (H) Parameters

`.LIN` analysis can calculate the complex two-port H (hybrid) parameter of a multi-terminal network.

The H parameters of a two-port network relate the voltages and currents at input and output ports:

$$V1 = h11 \cdot I1 + h12 \cdot V2$$

$$I2 = h21 \cdot I1 + h22 \cdot V2$$

In the preceding equations:

- $H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$ Hybrid matrix
- V1=Voltage at input port
- I2=Current at output port
- V2=Voltage at output port
- I1=Current at input port

You can add the hybrid H parameter matrixes of two networks to describe networks that are in series at their input and in parallel at their output.

`.LIN` can calculate H parameters based on the scattering parameters of the networks. `.LIN` analysis can extract one-port and two-port network H parameters. For networks with more than two ports, `.LIN` assumes that the ports numbered 3 and above terminate in their port characteristic impedance (Zc[i], i>2).

Group Delay

Group delay measures the transit time of a signal through a network versus frequency. It reduces the linear portion of the phase response to a constant value, and transforms the deviations from linear phase into deviations from constant group delay (which causes phase distortion in communications systems). The average delay represents the average signal transit time through a network system.

HSPICE can output the result of `.LIN` group delay measurement to a .sc*, Touchstone, or CITIfile.

Group delay is a function of frequency:

$$gd(w) = \frac{-d(phase)}{d(w)}$$

Where,

- gd=Group delay at the *f* frequency, $2\pi f = w$
- phase=phase response at the *f* frequency
- w=radians frequency

All complex S, Y, Z, and H parameters support a group delay calculation.

Syntax

```
Xmn(T) | Xmn(TD) | X(m,n)(T) | X(m,n)(TD)
X=S, Y, Z, or H (parameters)
m,n=port number (1 or 2 for H-parameters)
```

The results of the group delay calculation are scalar real numbers in units of seconds. For `.LIN`, group delay values are a function of frequency. The calculation is:

$$r_{ij} = |r_{ij}|e^{jf_{ij}(w)}$$

Differentiating the complex logarithm with respect to omega results in:

$$\frac{1}{r_{ij}}\frac{dr_{ij}}{dw} = \frac{1}{|r_{ij}|}\frac{d|r_{ij}|}{dw} + j\frac{df}{dw}$$

The group delay is the negative derivative of the phase. Simulation can compute it from the imaginary component of the derivative w.r.t. frequency of the measurement:

$$\tau(w) = -\frac{df}{dw} = -Im\left[\frac{1}{r_{ij}}\frac{dr_{ij}}{dw}\right]$$

Additional Measurements From .LIN

In addition to S, Y, Z, and H parameters, a LIN analysis can include the output measurements in the following sections.

Impedance Characterizations

- `VSWR(i)`=Voltage standing wave ratio at port i (real, unit-less)
- `ZIN(i)`=Input impedance at port i (complex, Ohms)
- `YIN(i)`=Input admittance at port i (complex, Siemens)

Stability Measurements

- `K_STABILITY_FACTOR`=Rollett stability factor (real, unit-less)
- `MU_STABILITY_FACTOR`=Edwards & Sinsky stability factor (real, unit-less)

Gain Measurements

- `G_MAX`=Maximum available/operating power gain (real, power ratio)
- `G_MSG`=Maximum stable gain (real, power ratio)
- `G_TUMAX`=Maximum unilateral transducer power gain (real, power ratio)
- `G_U`=Unilateral power gain (real, power ratio)

Matching for Optimal Gain

- `G_MAX_GAMMA1`=Source reflection coefficient needed to realize `G_MAX` (complex, unit-less)
- `G_MAX_GAMMA2`=Load reflection coefficient needed to realize `G_MAX` (complex, unit-less)
- `G_MAX_Z1`=Source impedance needed to realize `G_MAX` (complex, Ohms)
- `G_MAX_Z2`=Load impedance needed to realize `G_MAX` (complex, Ohms)
- `G_MAX_Y1`=Source admittance needed to realize `G_MAX` (complex, Siemens)
- `G_MAX_Y2`=Load admittance needed to realize `G_MAX` (complex, Siemens)

Noise Measurements

- `VN2`=Equivalent input noise voltage squared (real, V2)
- `IN2`=Equivalent input noise current squared (real, A2)
- `RHON`=Correlation coefficient between input noise voltage and input noise current (complex, unit-less)
- `RN`=Noise equivalent resistance (real, Ohms)
- `GN`=Noise equivalent conductance (real, Siemens)
- `ZCOR`=Noise correlation impedance (complex, Ohms)

- `YCOR`=Noise correlation admittance (complex, Siemens)
- `ZOPT`=Optimum source impedance for noise (complex, Ohms)
- `YOPT`=Optimum source admittance for noise (complex, Siemens)
- `GAMMA_OPT`=Optimum source reflection coefficient (complex, unit-less)
- `NFMIN`=Noise figure minimum (source at Zopt) (real, unit-less power ratio)
- `NF`=Noise figure (value obtained with source impedance at Z01) (real, unit-less power ratio)
- `G_AS`=Associated gain -- maximum power gain at NFMIN (real, power ratio)

Two-Port Transfer and Noise Measurements

Several of the output parameter measurements are assumed to be for two-port networks. When the network has 3 or more ports, the measurements are still carried out by assuming that `port=1` is the input and `port=2` is the output. All other ports are assumed terminated in their (noiseless) characteristic (z0) impedances. Note that this assumption is equivalent to operating on the two-port [S] submatrix extracted from the multi-port [S] matrix. This is true for both signal and noise calculations. A warning message is issued in cases where N>2 when two-port quantities are requested.

Signal and noise port-wise calculations do not require that port elements use a ground reference. Measurements are therefore possible; for example, for fully differential circuits.

Since Zopt and Yopt can commonly take on infinite values when computing optimum noise conditions, calculations for optimum noise loading is performed in terms of the reflection coefficient GammaOpt, and is made as robust as possible.

Output Format for Two-Port Noise Parameters in .sc* Files

The output of two-port noise parameter data in *.sc** files are slightly modified. The tabulated data appears with the following quantities in the following order:

```
*| 2-port noise parameters
*|  frequency  Fmin[dB]  GammaOpt(M) GammaOpt(P)  RN/Z0
*|    {...data...}
```

Where

- `Fmin[dB]` is the minimum noise figure (dB)
- `GammaOpt(M)` is the magnitude of reflection coefficient needed to realize `Fmin`
- `GammaOpt(P)` is the phase (degrees) of reflection coefficient needed to realize `Fmin`
- `RN/Z0` is the normalized noise resistance

Both `GammaOpt` and `RN/Z0` values are normalized with respect to the characteristic impedance of the port=1 element; for example, Z01.

VSWR

The *Voltage Standing Wave Ratio* represents the ratio of maximum to minimum voltages along a standing wave pattern due to a port's impedance mismatch. All ports other than the port of interest terminate in their characteristic impedances. VSWR is a real number related to that port's scattering parameter:

$$VSWR[i]= \frac{1 + |s_{ii}|}{1 - |s_{ii}|}$$

ZIN(i)

The Input Impedance at the *i* port is the complex impedance into a port with all other ports terminated in their appropriate characteristic impedance. It is related to that port's scattering parameter:

$$ZIN[i] = Z_{0i}\frac{1 + S_{ii}}{1 - S_{ii}}$$

YIN(i)

The Input Admittance at the *i* port is the complex admittance into a port with all other ports terminated in their appropriate characteristic impedance. It is related to that port's scattering parameter:

$$YIN[i] = \frac{1}{Z_{0i}}\frac{1 - S_{ii}}{1 + S_{ii}}$$

K_STABILITY_FACTOR (Rollett Stability Factor)

The *Rollett* stability factor is:

$$K = \frac{1 - |s_{11}|^2 - |s_{22}|^2 + |\Delta|^2}{2|s_{12}||s_{21}|}$$

Δ determines the two-port *S* matrix calculated from this equation:

$$\Delta = s_{11}s_{22} - s_{12}s_{21}$$

An amplifier where K>1 is unconditionally stable at the selected frequency.

MU_STABILITY_FACTOR (Edwards-Sinsky Stability Factor)

The following equation defines the Edwards-Sinsky stability factor.

$$\mu = \frac{1 - |S_{11}|^2}{|S_{22} - \Delta S_{11}^*| + |S_{21}S_{12}|}$$

$$\Delta = S_{11}S_{22} - S_{12}S_{21}$$

An amplifier with μ>1 is considered unconditionally stable at the specified frequency.

Maximum Available Power Gain—G_MAX

This is the gain value that can be realized if the two-port is simultaneously conjugate-matched at both input and output (with no additional feedback):

$$G_{max} = \frac{|s_{21}|}{|s_{12}|}\left(K - \sqrt{K^2 - 1}\right)$$

K is the Rollett stability factor. Special cases of `G_MAX` are handled in the following manner:

- If $|S_{12}|$=0 and ($|S_{11}|$=1 or $|S_{22}|$=1), G_MAX=$|S_{21}|^2$
- If $|S_{12}|$=0 and $|S_{11}|\neq1$ and $|S_{22}|\neq1$, G_MAX=G_TUMAX
- If $|S_{12}|\neq0$ and K≤1, G_MAX=G_MSG

When values for K≤1, the Maximum Available Power Gain is undefined, and HSPICE RF returns the Maximum Stable Gain.

Maximum Stable Gain - G_MSG

For a two-port that is conditionally stable (K<1), the following equation calculates the maximum stable gain:

$$G_{MSG} = \frac{|s_{21}|}{|s_{12}|}$$

To achieve this gain, resistively load the unstable two-port so that K=1, and then simultaneously conjugately match the input and output ports. `G_MSG` is therefore equivalent to `G_MAX` with K=1. In terms of admittance parameters:

$$G_{MSG} = \frac{|y_{21}|}{|y_{12}|}$$

MSG is equivalent to the Maximum Available Power Gain if K=1.

Maximum Unilateral Transducer Power Gain —G_TUMAX

This is the highest possible gain that a two-port with no feedback (that is, S_{12}=0) can achieve.

$$G_{tumax} = \frac{|s_{21}|^2}{(1-|s_{11}|^2)(1-|s_{22}|^2)}$$

Unilateral Power Gain—GU

This is the highest gain that the active two-port can ever achieve by embedding in a matching network that includes feedback. The frequency where the unilateral gain becomes unity defines the boundary between an active and a passive circuit. The frequency is usually referred to as f_{max}, the maximum frequency of oscillation.

To realize this gain, HSPICE RF neutralizes the feedback of the two-port, and simultaneously conjugate-matches both input and output:

$$G_U = \frac{\left|\frac{s_{21}}{s_{12}} - 1\right|^2}{2K\left|\frac{s_{21}}{s_{12}}\right| - 2Re\left\{\frac{s_{21}}{s_{12}}\right\}}$$

Simultaneous Conjugate Match for G_MAX

A simultaneous conjugate match is required at the source and load to realize the G_{max} gain value. The source reflection coefficient at the input must be:

$$\Gamma_1 = \frac{C_1^*}{|C_1|}\left[\frac{B_1}{2|C_1|} - \sqrt{\frac{B_1^2}{|2C_1|^2} - 1}\right]$$

$$B_1 = 1 - |s_{22}|^2 + |s_{11}|^2 - |\Delta|^2$$

$$C_1 = s_{11} - \Delta s_{22}^* \qquad \Delta = s_{11}s_{22} - s_{12}s_{21}$$

The load reflection coefficient (`G_MAX_GAMMA_2`) is:

$$\Gamma_2 = \frac{C_2^*}{|C_2|}\left[\frac{B_2}{2|C_2|} - \sqrt{\frac{B_2^2}{|2C_2|^2} - 1}\right]$$

In the preceding equation:

$$B_2 = 1 - |s_{11}|^2 + |s_{22}|^2 - |\Delta|^2$$

$$C_2 = s_{22} - \Delta s_{11}^* \qquad \Delta = s_{11}s_{22} - s_{12}s_{21}$$

You can obtain useful solutions only when:

$$\frac{B_1}{|2C_1|} > 1 \qquad \frac{B_2}{|2C_2|} > 1$$

These equations also imply that K>1.

HSPICE RF derives calculations for the related impedances and admittances from the preceding values.

For G_MAX_Z1:

$$Z_1 = Z_{01}\frac{1+\Gamma_1}{1-\Gamma_1}$$

For G_MAX_Z2:

$$Z_2 = Z_{02}\frac{1+\Gamma_2}{1-\Gamma_2}$$

For G_MAX_Y1

$$Y_1 = \frac{1}{Z_{01}}\frac{1-\Gamma_1}{1+\Gamma_1}$$

For G_MAX_Y2

$$Y_2 = \frac{1}{Z_{02}}\frac{1-\Gamma_2}{1+\Gamma_2}$$

Equivalent Input Noise Voltage and Current—IN2, VN2, RHON

For each analysis frequency, HSPICE RF computes a noise-equivalent circuit for a linear two-port. The noise analysis assumes that all ports terminate in noise-less resistances. For circuits with more than two ports, ports identified as 3 and above terminate, and the analysis considers only ports 1 and 2. The noise-equivalent circuit calculation results in an equivalent noise voltage and current, and their correlation coefficient. These values are:

$$VN2 = \overline{|\mathrm{v}_n|^2} \qquad IN2 = \overline{|\mathrm{i}_n|^2}$$

$$RHON = \rho_n = \frac{\overline{i_n v_n^*}}{\sqrt{|i_n|^2 |v_n|^2}}$$

Equivalent Noise Resistance and Conductance—RN, GN

These measurements are the equivalent resistance and conductance that would generate the equivalent noise voltage and current values at a temperature of $T_0 = 290k$ in a 1 Hz bandwidth (that is, $\Delta f = 1Hz$).

$$RN = R_n = \frac{\overline{|v_n|^2}}{4kT_0\Delta f} \qquad GN = G_n = \frac{\overline{|i_n|^2}}{4kT_0\Delta f}$$

Noise Correlation Impedance and Admittance—ZCOR, YCOR

These measurements represent the equivalent impedance and admittance that you can insert at the input of the noise-equivalent circuit to account for the correlation between the equivalent noise voltage and the current values.

$$ZCOR = \rho_n \sqrt{\frac{\overline{|v_n|^2}}{\overline{|i_n|^2}}} = \frac{\overline{i_n^* v_n}}{\overline{|i_n|^2}} = R_{cor} + jX_{cor}$$

$$YCOR = \rho_n \sqrt{\frac{\overline{|i_n|^2}}{\overline{|v_n|^2}}} = \frac{\overline{i_n v_n{}^*}}{\overline{|v_n|^2}} = G_{cor} + jB_{cor}$$

ZOPT, YOPT, GAMMA_OPT – Optimum Matching for Noise

The equivalent input noise sources and their correlation make it possible to compute the impedance, admittance, and reflection coefficient values that, if presented at the input of the noisy two-port, result in the best noise performance. These values are:

$$Z_{opt} = \sqrt{\frac{R_n}{G_n} - X_{cor}{}^2} - jX_{cor} \qquad Y_{opt} = \frac{1}{Z_{opt}} = \sqrt{\frac{G_n}{R_n} - B_{cor}{}^2} - jB_{cor}$$

$$GAMMA\_OPT = \Gamma_{opt} = \frac{Z_{opt} - Z_{01}}{Z_{opt} + Z_{01}}$$

Noise Figure and Noise Figure Minimum—NF, NFMIN

If you set the input source impedance to ZOPT, the two-port operates with the minimum Noise Figure. The definition of the Noise Figure (F) is unusual because it involves the available gain of the two-port and not its transducer gain. You can express it in the following form:

$$F = 1 + \frac{N_a}{G_a k T_0 \Delta f}$$

- G_a is the available power gain.
- N_a is the available noise power at the output of the two-port (due solely to the two-port's noise and not to the input impedance).
- k is Boltzmann's constant.
- T_0 is the 290 Kelvin reference temperature.

The NMIN minimum noise figure value is computed as:

$$NFMIN = F_{min} = 1 + 2G_n\left(R_{cor} + \sqrt{\frac{R_n}{G_n} - X_{\mathrm{cor}}^2}\right)$$

where NFMIN≥1. For input source impedance values other than `ZOPT`, the Noise Figure varies as a function of the input source reflection coefficient, according to:

$$F = F_{min} + \frac{R_n|\Gamma_S - \Gamma_{opt}|^2}{2Z_{01}|1+\Gamma_{opt}|^2}$$

The HSPICE RF Noise Figure measurement (NF) returns the noise figure value if the input terminates in the port characteristic impedance (that is, $\Gamma_s = 0$). This value is:

$$NF = F_{min} + \frac{R_n|\Gamma_{opt}|}{2Z_{01}|1+\Gamma_{opt}|^2} = F_{min} + \frac{G_n}{Z_{01}}|Z_{01} - Z_{opt}|^2$$

Associated Gain—G_As

HSPICE RF also includes a measurement named Associated Gain, which assumes that the Γ_S in-out impedance is matched for the minimum noise figure (that is, $\Gamma_s = \Gamma_{opt}$), while the output is matched for the maximum gain.

$$G_{AS} = \frac{|s_{21}|^2(1 - |\Gamma_{Opt}|^2)}{|1 - s_{11}\Gamma_{Opt}|^2(1 - |s'_{22}|^2)}$$

In the preceding equation: $s'_{22} = s_{22} + \frac{s_{12}s_{21}\Gamma_{Opt}}{1 - s_{11}\Gamma_{Opt}}$

Extracting Mixed-Mode Scattering (S) Parameters

Note:

An easily-adapted time domain reflectometry (TDR) example is available in Performing a TDR Simulation in HSPICE in the *HSPICE User Guide: Signal Integrity*.

In HSPICE RF, the `.LIN` analysis includes a keyword for extracting mixed-mode scattering (S) parameters.

Syntax

```
.LIN … [ mixedmode2port= dd|dc|ds|cd|cc|cs|sd|sc|ss ]
```

The following keywords in a `.PRINT` and `.PROBE` statements specifies the elements in the mixed mode S parameter matrices:

```
S|Y|Z<xy>nm<(t)>
```

Argument	Description
x, y	One of the following: ■ D (differential) ■ C (common) ■ S (single-ended) For example: ■ SCC=common mode S-parameters ■ SDC or SCD=cross mode S-parameters If you omit x,y, then HSPICE uses the value set for the mixedmode2port.
S_{cc}	Common-mode S-parameters
S_{cd} and S_{dc}	Mode-conversion or cross-mode S-parameters
m, n	port number
type	One of the following: ■ DB: magnitude in decibels ■ I: imaginary part ■ M: magnitude (default) ■ P: phase in degree ■ R: real part

Defaults

Availability and default value for the `mixedmode2port` keyword depends on the port configuration.

Example 1

```
p1=p2=single
```

Where,

- Available: ss
- Default: ss

Example 2

```
p1=p2=balanced
```

Where,

- Available: dd,cd,dc,cc
- Default: dd

Example 3

```
p1=balanced p2=single
```

Where,

- Available: ds,cs
- Default: ds

Example 4

```
p1=single p2=balanced
```

Where,

- Available: sd,sc
- Default: sd

Output File Formats

An *sc#* file format for the mixed mode:

- The S-element model has additional keywords, such as `mixedmodei` and `idatatype`, if the netlist includes one or more balanced ports.
- The `mixedmode2port` keyword prints in the header line.
- The other S-element keywords also appear in the header lines.

Touchstone format for the mixed mode:

The following lines for data mapping are added to the head of the Touchstone output file if the netlist includes one or more balanced ports.

```
! S11=SDD11
! S12=SDD12
! S13=SDC11
! S14=SDC12
```

Two-Port Parameter Measurement

Two-port parameter measurement function takes the first 2 ports, then reads the corresponding parameter with the drive condition specified by the `mixedmode2port` keyword.

Output Format and Description

File Type	Description
*.ac#	Output from both the .PROBE and .PRINT statements.
*.printac#	Output from the .PRINT statement. This is available in HSPICE RF only.
*.sc#	The extracted S-parameters/2-port noise parameters are written to a *.sc# file by using the S-element format. If you want to simulate the S element, you can reference the *.sc# file in your netlist.

```
* N=numOfPorts DATA=numOfFreq NOISE=[0|1] GROUPDELAY=[0|1]
* NumOfBlock=numOfSweepBlocks NumOfParam=numOfSweptParameters
* MIXEDMODE=[0|1] DATATYPE=mixedModeDataTypeString
.MODEL mname S
+ N=numOfPorts FQMODEL=SFQMODEL TYPE=S Zo=*** *** ...
.MODEL SFQMODEL SP N=numberOfPorts SPACING=POI
   INTERPOLATION=LINEAR MATRIX=NONSYMMETRIC
+ DATA= numberOfData
+ freq1
+ s11real s11imag s12real s12imag ... s1Nreal s1Nimag
...
+ sN1real sN1imag ... sNNreal sNNimag
...
...
+ freqNumberOfData
+ s11real s11imag s12real s12imag ...s1Nreal s1Nimag
...
+ sN1real sN1imag ... sNNreal sNNimag

* 2-port noise parameter
* frequency   Fmin [dB]   GammaOpt(M)   GammaOpt(P)   RN/Zo
* 0.10000E+09   0.   1.0000    0.    1.0281
* ...
```

The 2-port noise section starts with “*” so that you can include this file in your HSPICE or HSPICE RF input netlists.

Features Supported

`.LIN` analysis in HSPICE and HSPICE RF supports the following features:

- Automatic calculation of bias-dependent S, Y, and Z parameters. No additional sources required.
- Automatic calculation of noise parameters.
- Automatic calculation of group delay matrices.

In addition, HSPICE RF supports all existing HSPICE RF models. For noise analysis, HSPICE and HSPICE RF view port 1 as the input and port 2 as the output.

Prerequisites and Limitations

The following prerequisites and limitations apply to `.LIN` analysis in HSPICE RF:

- Requires one `.LIN` statement to specify calculation options.
- Requires one `.AC` statement to specify frequency sweep and parameter sweep.
- Requires at least one P element, numbered from port 1 to N.
- For noise analysis, HSPICE RF views port 1 as the input and port 2 as the output.

Reported Statistics for the Performance Log (HSPICE RF Only)

- Simulation time
  - DC op time
  - Total simulation time
- Memory used
  - Total memory

Errors and Warnings

- If the circuit contains fewer than two P-elements and noisecalc=1, then the 2-port noise calculation is skipped.
- If the circuit contains fewer than two P-elements, does not let you cannot use the `.PRINT`, `.PROBE`, or `.MEAS` command with any two-port noise or gain parameters.
- If the circuit contains more than two P-elements, all two-port parameters are computed. By default, `port=1` is the input and `port=2` is the output. All other ports terminate in their reference impedances.

Example

```
.OPTION POST=2
.AC DEC 1 20MEG 20G
.LIN noisecalc=1
Pout outs gnd port=2 RDC=50 RAC=50 DC=0 AC=1 0
Pin ins gnd port=1 RDC=50 RAC=50 DC=0.5 AC=0.5 0
xlna_2_ ins outs lna
.subckt lna in out
rhspr5 in _n481 50
rhspr6 _n523 out 100
vvdd _n523 gnd dc=1.8
qhspnpn3 out _n481 gnd gnd bjtm1 area=3
.ends lna
.global gnd
.END
```

Via Modeling for PCBs in HSPICE

You can do modeling of vias for PCBs in pre-layout signal integrity phase of HSPICE simulations. While A 3D field solver is required to accurately model vias, you can use the built-in HSPICE 2D field solver to approximate a via model using the following method:

1. Create a SPICE netlist for a via. The via can be modeled as a lumped pi-model if the via delay is less than 0.1 of the signal edge.

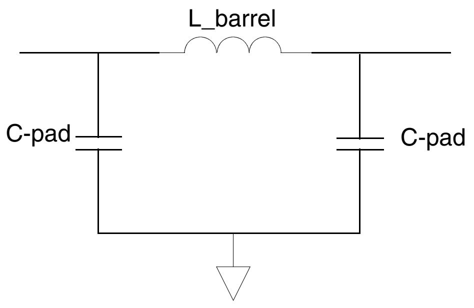


Figure 86 Where: L_barrel is the inductance of the barrel of the via and C_pad is the capacitance of the pad.

2. Use the .LIN command to extract an S-parameter model file in Touchstone format.
3. Use the S-element to use the model file.

Other Considerations

If you are working with a PCB via, note that it can also be modeled as a Pi Lumped Element Model. The pi-model circuit approximates the behavior of short transmission lines (lumped RLC circuits). But the exact values for the RLC parameters of a via depend on the size and geometry of the via. To determine the exact values, you need to use a 3D field solver, such as the Synopsys Raphael product to extract exact parameter values.

Equivalent RLCs for GHz boards require a 3D EM analysis for accurate modeling, such as Raphael provides to include the magnetic flux in the conductor at DC or for modeling edge effects accurately. If Raphael can produce the RLGC table model, you can simulate with the model in HSPICE and then compare the result with field solvers to evaluate the difference.

Note that modeling IC vias is more complex as you need to consider stress effects (since process-induced mechanical stress can cause electromigration and formation of voids or cracks in vias or metal lines), thermal effects (since vias have much higher thermal conductivity than the dielectric materials (ILD), and modeling via plug resistance, etc.

References

[1] Goyal, Ravender. “S-Parameter Output From SPICE Program”, *MSN & CT,* February 1988, pp. 63 and 66.

[2] Robert J. Weber “Introduction to Microwave Circuits”, IEEE Press.

[3] Behzad Razavi, “Design of Analog CMOS Integrated Circuits”, McGraw Hill.

[4] Reinhold Ludwig, Pavel Bretchko, “RF Circuit Design Theory and Applications”.

[5] G.D. Vendelin, Design of Amplifiers and Oscillators by the S-Parameter Method, John Wiley & Sons, 1982.

[6] R.S. Carson, High-Frequency Amplifiers, 2nd Edition, John Wiley & Sons, 1982.

[7] G. Gonzalez, Microwave Transistor Amplifiers: Analysis and Design, 2nd Edition, Prentice-Hall, 1997.

[8] M.L. Edwards and J.H. Sinsky, “A single stability parameter for linear 2-port networks,” IEEE 1992 MTT-S Symposium Digest, pages 885-888.

[9] H. Rothe and W. Dahlke, “Theory of noisy fourpoles”, Proc. IRE, volume 44, pages 811-818, June 1956.

[10] David E. Bockeman, “Combined Differential and Common-Mode Scattering Parameters: Theory and Simulation,” IEEE trans. on MTT Volume 43, Number 7, Jul. 1995.

[11] “Understanding Mixed Mode S-parameters,” http://www.si-list.org/files/tech_files/Understandmm.pdf

[12] Robert J. Weber “Introduction to Microwave Circuits”, IEEE Press.

[13] Behzad Razavi, “Design of Analog CMOS Integrated Circuits”, McGraw Hill.

[14] Reinhold Ludwig, Pavel Bretchko, “RF Circuit Design Theory and Applications.”

18

Statistical Eye Analysis

Describes statistical eye diagram analysis using the .STATEYE command.

Analysis of high-speed serial interfaces requires processing of millions of bits of data making analysis by traditional analysis tools computationally expensive. Eye diagrams are used extensively in the evaluation of high-speed serial interfaces and as a fundamental performance metric for high-speed serial interfaces in the bit error rate (BER). Statistical eye diagram techniques allow eye diagrams and BER to be evaluated quickly and accurately.

These topics are discussed in the following sections:

Important:

The `.STATEYE` command is invoked using the `hspicerf` executable on the command line for this release (not `hspice`). However, running a statistical eye analysis only requires an HSPICE license token.

Statistical Eye Analysis Setup

The port element is used to designate the incident (input) and probe (output) ports for the system to be analyzed. Ports can be specified as single ended or mixed mode. Random jitter can be applied to each incident and probe point in the system.

Each incident port acts as random bit pattern source with specified voltage magnitude. If an incident port element does not have a time domain voltage magnitude specification, the default values, V_high=1.0, V_low=–1.0 are used.

Probe ports are used as observation points where `.PRINT`, `.PROBE` and `.MEASURE` statements can be defined.

To perform the statistical eye diagram analysis, use the analysis command `.STATEYE`.

Input Syntax

The syntax for the `.STATEYE` command is:

```
.STATEYE T=time_interval Trf=rise_fall_time
+ Incident_port=idx1, [idx2, … idxN]
+ Probe_port=idx1, [idx2, … idxN]
+ [Rj=Rj1, [Rj2, … RjN]]
+ [tran_init=n_periods]
+ [V_low=val] [V_high=val]
+ [T_resolution=n] [V_resolution=n]
```

Parameter	Description
T	Time of one period of the incident pulse signal (in seconds)
Trf	Single value (in seconds) to set both the rise and fall times of the incident pulse
Incident_port	An array of the index numbers of the incident port elements
Probe_port	An array of the index numbers of the probing port elements
Rj	An array of the real numbers to specify the standard deviation of the Gaussian random jitter. The array must be in the order of the port element index. No random jitter is added by default.
V_low	Low voltage level of the incident pulse. The value is used when the voltage level is not specified in the incident port(s).
V_high	High voltage level of the incident pulse. The value is used when the voltage level is not specified in the incident port(s).

Parameter	Description
Tran_init	An integer number that specifies the numbers of periods (T) that is used by the initial transient analysis to determine the pulse response of the system. Default value is 30.
T_resolution	An integer number used to specify the probability density function (PDF) image resolution of the time axis. Default value is 200.
V_resolution	An integer number used to specify the probability density function (PDF) image resolution of the voltage axis. Default value is 200.

Port Element Configuration

Incident ports act as pure random bit stream generators in `.STATEYE` analysis unless either LFSR or PAT source keywords are specified.

When either a LFSR or PAT keyword is specified, the incident port will act as a specific bit pattern generator.

Port elements that are defined but not specified as an incident or probe port will be assumed to be a DC voltage source with no time dependence.

The following statistical eye diagram analysis results can be printed or probed:

- Eye diagram at a port
- Bit error rate map at a port
- Bathtub curve of the bit error rate at a port for specified voltage
- Bathtub curve of the bit error rate at a port for a specified time

.Print and .Probe Syntax

```
.print stateye eye(port_idx)
.print stateye eyeBW(port_idx)
.print stateye eyeV(port_idx,v)
.print stateye eyeT(port_idx,t)
.print stateye ber(port_idx)
.print stateye bathtubV(port_idx,v)
.print stateye bathtubT(port_idx,t)

.probe stateye eye(port_idx)
.probe stateye eyeBW(port_idx)
.probe stateye eyeV(port_idx,v)
.probe stateye eyeT(port_idx,t)
```

```
.probe stateye ber(port_idx)
.probe stateye bathtubV(port_idx,v)
.probe stateye bathtubT(port_idx,t)
```

Where:

- `eye(port_idx)` specifies the port to output the eye diagram
- `eyeBW(port_idx)` specifies the port to output the eye diagram as black and white data
- `eyeV(port_idx,v)` specifies the port to output as cross-section of the eye diagram at specified port at specified voltage
- `eyeT(port_idx, t)` specifies cross section of the eye diagram at specified port at specified time
- `ber(port_idx)` specifies the port to output the bit error rate
- `bathtubv(port_idx, v)` specifies the port and voltage to output a bathtub curve
- `bathtubt(port_idx, t)` specifies the port and time to output a bathtub curve

.Measure Syntax

The following statistical eye diagram analysis results can be measured:

- Vertical eye opening at a specified time
- Horizontal eye opening at a specified voltage
- Measure a bit pattern that produces the worst eye opening at a specified time in a high or low state

```
.measure stateye result_name Veye port_idx time=val
+ [tol=val]
.measure stateye result_name Heye port_idx volt=val
+ [tol=val]
.measure stateye result_name WorstBits port_idx time=val
+ [state=low|high]
```

Where,

- `Veye` *`port_idx`* `time=`*`val`* `[tol=`*`val`*`]` specifies the port, time and tolerance where the vertical eye opening is to be measured.
- `Heye` *`port_idx`* `volt=`*`val`* `[tol=`*`val`*`]` specifies the port, voltage and tolerance where the horizontal eye opening is to be measured.
- `WorstBits` *`port_idx`* `time=`*`val`* `[state=`*`low|high`*`]` specifies the port, time and state of the bit pattern that produces the worst eye opening.

The default value of tolerance (`tol`) and logic state (`state`) are `0` and `high`, respectively.

Output Data

Probing Output

The `.probe stateye` command generates *netlist.stet#* and *netlist.stev#* for following purposes:

- netlist.stet#: `eye(t,v)`, `eyeBW(t,v)`, `eyeV(t)`, `ber(t,v)` and `bathtubV(t)`
- *netlist.stev#:* `bathtubT(v)`

Printing Output

The `.print stateye` command generates the data directory *netlist.printSte#/* then stores a convenient gnuplot script for each `.print stateye` target with necessary data files. For example:

- The `.print stateye eye(2)` statement creates the `netlist.printSte0/eye_2` script file.
- The `.print stateye bathtubV(2,0.1)` creates the `netlist.printSte0/bathtub_2_0.1000e+00` script file.

On the `gnuplot` command shell, these scripts can be directly loaded to display the target data. For example,

```
% gnuplot
gnuplot> load "printSte0/eye_2"
gnuplot> load "netlist.printSte0/bathtub_2_0.1000e+00"
```

Measurement Output

The `.measure stateye` command generates the *netlist.mste#* file and stores the measurement results.

Examples, Statistical Analysis Setup

Incident port definitions

```
p1 tx_in+ tx_in- 0 port=1
p2 in 0 port=2
```

Probe port definitions

```
p3 rxout+ rxout- 0 port=3
p4 out 0 port=4
```

Analysis statement

```
.stateye T = 400p trf=20p
+ incident_Port= 1, 2
+ probe_port = 3, 4
+ Rj = 5p, 5p, 2p, 2p tran_init = 50
+ T_resolution = 300 V_resolution = 300
```

Print, probe, and measure statements

```
.print stateye eye(4)
.print stateye ber(3)
.print stateye bathtubV(3, 0.9)
.print stateye bathtubT(4, 1n)

.probe stateye eye(4)
.probe stateye ber(3)
.probe stateye bathtubV(3, 0.9)
.probe stateye bathtubT(4, 1n)

.measure stateye eyevert Veye 4 time=1n tol=0.1n
.measure stateye eyehorz Heye 4 volt=0.9 tol=0.05
.measure stateye badbithigh WorstBits 3 time=1n state=high
```

Known Limitation

For the current version, a linear time invariant system is assumed for the superposition of the pulse responses even though non-linearities will be taken into account when computing the single bit pulse response.

19

Timing Analysis Using Bisection

Describes how to use the bisection function in timing optimization.

To analyze circuit timing violations, a typical methodology is to generate a set of operational parameters that 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. You must perform an iterative analysis to define the violation specification.

Typical types of timing constraint violations include:

- Data setup time, before the clock.
- Data hold time, after the clock.
- Minimum pulse width required for a signal to propagate to the output.
- Maximum toggle frequency of the component(s).

For more information about optimization, see the HSPICE Simulation and Analysis User Guide

Overview of Bisection

Before bisection methods were developed, engineers built external drivers to submit multiple parameterized simulations to SPICE-type simulators. Each simulation explored a region of the operating envelope for the circuit. To provide part of the analysis, the driver also post-processed the simulation results, to deduce the limiting conditions.

If you characterize small circuits this way analysis times are relatively small, compared with the overall job time. This method is inefficient, due to overhead of submitting the job, reading and checking the netlist, and setting up the matrix. The newer bisection methods increase efficiency when you analyze timing violations, to find the causes of timing failure. Bisection optimization is an efficient cell-characterization method, in Synopsys HSPICE or HSPICE RF.

The bisection methodology saves time in three ways:

- Reduces multiple jobs to a single characterization job.
- Removes post-processing requirements.
- Uses accuracy-driven iterations.

Figure 87 on page 497 shows a typical analysis of setup-time constraints. Clock and data input waveforms drive a cell. Two input transitions (rise and fall) occur at times T_1 and T_2. The result is an output transition, when V(out) changes from low to high. The following relationship between the T_1(data) and T_2 (clock) times must be true for the V(out) transition to occur: T_2>(T_1+setup time).

Characterization or violation analysis determines the setup time. To do this, HSPICE or HSPICE RF keeps T_2 fixed and repeats the simulation with different T_1 values. It then observes which T_1 values produce an output transition and which do not.

Before bisection, users had to run tight sweeps of the delay between the data setup and clock edge, and look for the value at which no transition occurs. To do this, you swept a value that specifies how far the data edge precedes a fixed clock edge. This method is time consuming, and is accurate only if the sweep step is very small. Linear search methods cannot accurately determine the setup time value, unless you use extremely small steps from T_1 to T_2 to simulate the circuit at each point, and monitor the outcome.

For example, even if you know that the desired transition occurs during a particular 5ns period, you might need to run 50 simulations to search for the setup time to within 0.1ns over that 5ns period. But the error in the result can be as large as 0.05ns.

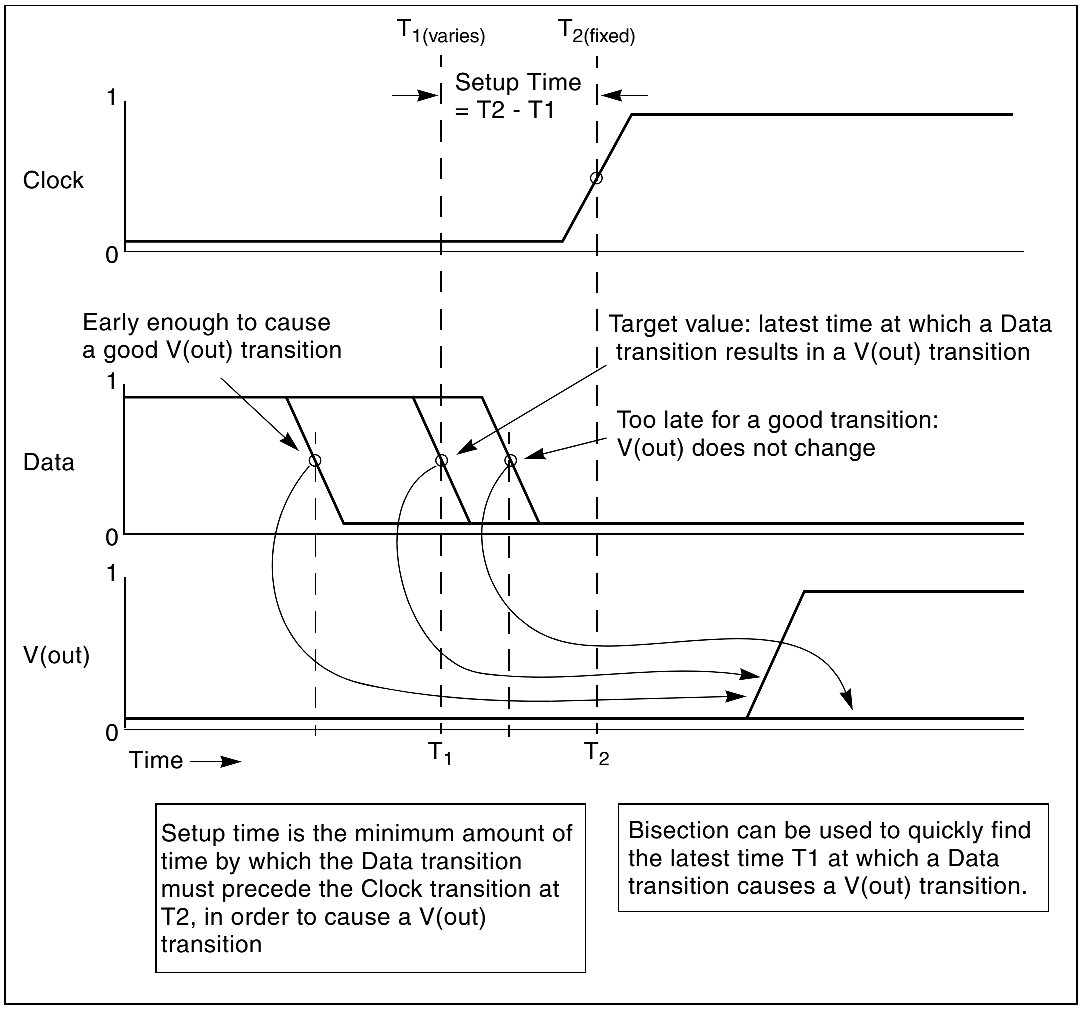

T1(varies)
T2(fixed)
Setup Time
= T2 - T1
1
Clock
0
Early enough to cause
a good V(out) transition
Target value: latest time at which a Data transition results in a V(out) transition
1
Too late for a good transition:
V(out) does not change
Data
0
1
V(out)
0
Time
T1
T2
Setup time is the minimum amount of time by which the Data transition must precede the Clock transition at T2, in order to cause a V(out) transition
Bisection can be used to quickly find the latest time T1 at which a Data transition causes a V(out) transition.

Figure 87 Determining Setup Time with Bisection Violation Analysis

The bisection feature greatly reduces the amount of work and computational time required to find an accurate solution for this type of problem. The following sections show examples of using this feature to identify timing violations for the setup, hold, and minimum clock pulse width.

Bisection Methodology

Bisection is an optimization method that uses a binary search method, to find the value of an input variable (target value). This variable is associated with a goal value of an output variable.

The type of the input and output variables can be voltage, current, delay time, or gain, related by some transfer function. In general, use a binary search to locate the goal value of the output variable within a search range of the input variable. Then, iteratively halve that range to rapidly converge on the target value. At each iteration, HSPICE or HSPICE RF compares the measured value of the output variable with the goal value. Both the pass/fail method and the bisection method use bisection (see Using Bisection on page 498).

The bisection procedure consists of two measurement and optimization steps, when solving the timing violation problem:

- Detecting whether the output transition occurred.
- Automatically varying the input parameter (T_1 in Figure 87 on page 497) to find the value for which the transition barely occurs.

Measurement

Use the `MAX` measurement function to detect the success or failure of an output transition. For a low-to-high output transition, a `MAX` measurement produces zero on failure, or approximately the V_{dd} supply voltage on success. This measurement, using a goal of V_{dd} (minus a suitable small value to ensure a solution), is sufficient to drive the optimization.

Optimization

The bisection method is straightforward if you specify a single measurement with a goal, and known upper and lower boundary values for the input parameter. The characterization engineer must specify acceptable upper and lower boundary values.

Using Bisection

Before you can use bisection, you must specify the following:

- A pair of values, for the upper and lower boundaries of the input variables. To find a solution, one of these values must result in an output variable >|`goal value`| and the other must result in <|`goal value`|.
- A goal value. If there is no goal keyword in the statement, the goal value will not be defaulted to zero, and HSPICE considers the measure result as a relative error expression.
- Error tolerance value. The bisection process stops when the difference between successive test values is $\leq$ error tolerance. If the other criteria are also met, see the following steps.
- Related variables. Use a monotonic transfer function to relate variables where a steadily-progressing time (increase or decrease) results in a single occurrence of the goal value at the target input variable value.

HSPICE or HSPICE RF includes the error tolerance in a relation, used as a process-termination criterion.

Figure 88 on page 504 shows an example of the binary search process that the bisection algorithm uses. This example is the pass/fail type, and is appropriate for a setup-time analysis that tests for the presence of an output transition. In the example depicted in Figure 87 on page 497 note that:

1. A long setup time TS (= T2 - T1) results in a `VOUT` transition (a pass).
2. A too-short setup time (where the latch has not stabilized the input data, before the clock transition) results in a fail.

   Explanation: For example, you might define a pass time value as any setup time, TS, that produces a `VOUT` output minimum high logic output level of 2.7V, which is the goal value.
3. The target value is a setup time that just produces the `VOUT` value of 2.7V. Finding the exact value is impractical, if not impossible, so you need to specify an error tolerance to calculate a solution arbitrarily close to the target value.
4. The bisection algorithm performs tests for each specified boundary value to determine the direction in which to pursue the target value, after the first bisection. In this example shown in Figure 87 on page 497, the upper boundary has a pass value and the lower boundary has a fail value.
5. To start the binary search you specify the lower and upper boundaries.

   The program tests the point midway between the lower and upper boundaries (see Figure 88 on page 504).

- If the initial value passes the test, the target value must be less than the tested value (in this example). The bisection algorithm moves the upper search limit to the value that it just tested.
- If the test fails, the target value must be greater than the tested value. Bisection moves the lower limit to the value that it just tested.

6. The algorithm tests a value midway between the new limits.
7. The search continues in this manner, moving one limit or the other to the last midpoint, and testing the value midway between the new limits.
8. The process stops when the difference between the latest test values is less than or equal to the error tolerance that you specified. To normalize this value, multiply by the initial boundary range.

For more information about using the `.MODEL` statement for bisection, see the HSPICE and HSPICE RF Command Reference.

Examining the Command Syntax

The following syntax is used for bisection:

```
.MODEL <OptModelName> OPT METHOD=BISECTION ...
```

-or-

```
.MODEL <OptModelName> OPT METHOD=PASSFAIL ...
```

`OptModelName` is the model to be used. Refer to the HSPICE Simulation and Analysis User Guide for name information on specifying optimization models in HSPICE. The `METHOD` keyword specifies which optimization method to use. The `OPT` keyword indicates that optimization is to be performed.

For bisection, the method can be one of the following:

- `BISECTION`

  When the difference between the two latest test input values is within the error tolerance and the latest measured value exceeds the goal, bisection has succeeded and then ends. This process reports the optimized parameter that corresponded to the test value that satisfies this error tolerance and this goal (passes).

- `PASSFAIL`

  When the difference between the last two optimization parameter test values is < the error tolerance and the associated goal measurement fails for one of the values and passes for the other, bisection has succeeded and then ends. The process reports the optimization parameter test value associated with the last passing measurement. “Pass” is defined as a condition in which the associated goal measurement can produce a valid result. “Fail” is defined as a condition in which the associated goal measurement is unable to produce a valid result. For example, if the measurement is of TRIG/TARG form, and the TARG event is not found, then this optimization parameter test value is deemed a failure. When using PUSHOUT bisection, the definition of a failure is modified to also include any goal measurement result that is valid and > the push-out specification.

The parameters are passed in a normal optimization specification:

```
.PARAM <ParamName>=<OptParFun> (<Initial>, <Lower>, <Upper>)
```

In the `BISECTION` method, the measure results for `<Lower>` and `<Upper>` limits of `<ParamName>` must be on opposite sides of the goal value in the `.MEASURE` statement. In the `PASSFAIL` method, the measure must pass for one limit and fail for the other limit. The process ignores the value of the `<Initial>` field. The error tolerance is a parameter in the model which is being optimized. Using the `BISECTION` method, a bisectional search is applied to multiple parameters. The logical relationship of these parameters is based on 'AND'. In the `PASSFAIL` method, a bisectional search is applied to only one parameter.

When the `OPTLST` option is set (`.OPTION OPTLST=1`), the process outputs the following information for the `BISECTION` method:

```
bisec-opt iter = <num_iterations> xlo = <low_val> xhi = <high_val>
x = <result_low_val> xnew = <result_high_val>
err = <error_tolerance>
```

The `x` is the old parameter value and `xnew` is the new parameter value.

When `.OPTION OPTLST=1`, the process outputs the following information for the `PASSFAIL` method:

```
bisec-opt iter = <num_iterations> xlo = <low_val> xhi = <high_val>
x = <result_low_val> xnew = <result_high_val>
measfail = 1
```

In this syntax, `measfail=0` for a test failure for the `x` value.

Performing Transient Analyses with Bisections

When performing transient analysis bisection with the `.TRAN` statement, use the following syntax:

```
.TRAN <TranStep> <TranTime> SWEEP OPTIMIZE=<OptParFun>
+ RESULTS=<MeasureNames> MODEL=<OptModelName>
```

When performing a transient analysis bisection with the `.MEASURE` statement, use the following syntax:

```
.MEASURE TRAN <MeasureName> <MeasureClause> GOAL=<GoalValue>
```

Setup Time Analysis

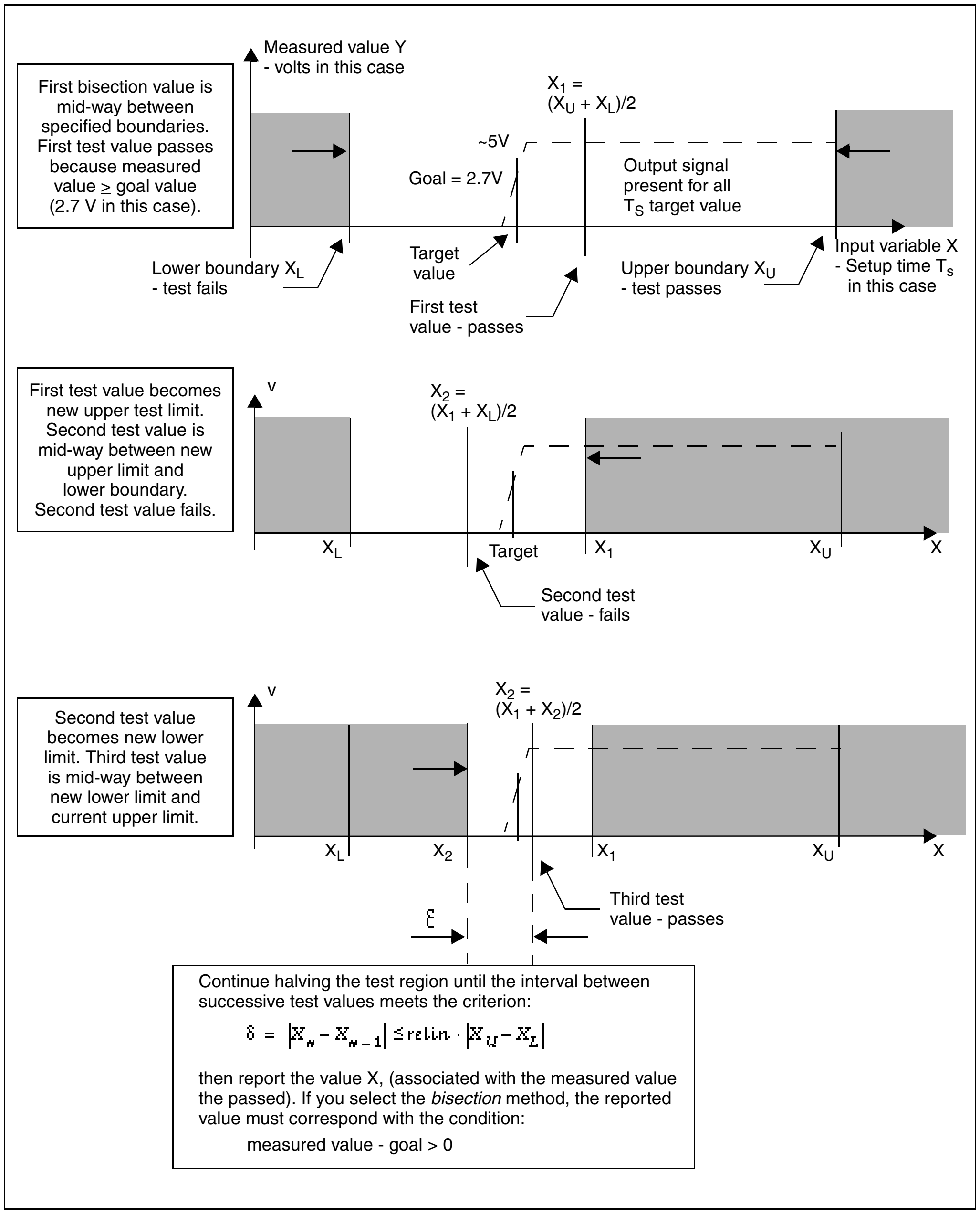
This example uses a bisectional search to find the minimum setup time for a D flip-flop. The circuit for this example is dff_top.sp, which is located in directory <$*installdir*>/demo/hspice/bisec.

Figure 88 on page 504 and Figure 89 on page 505 show the results of this demo. HSPICE or HSPICE RF does not directly optimize the setup time, but extracts it from its relationship with the `DelayTime` parameter (the time before the data signal), which is the parameter to optimize.

Input Listing

The following portion of the input listing shows how `.TRAN` analysis, the `DelayTime` parameter, and `.MEASURE` statements are used in bisection:

```
* DFF_top Bisection Search for Setup Time
* PWL Stimulus
v28 data gnd PWL
+ 0s 5v
+ 1n 5v
+ 2n 0v
+ Td = "DelayTime" $ Offsets Data from time by DelayTime
v27 clock gnd PWL
+ 0s 0v
+ 3n 0v
+ 4n 5v
* Specify DelayTime as the search parameter and provide
* the lower and upper limits.
.PARAM DelayTime= Opt1 ( 0.0n, 0.0n, 5.0n )
* Transient simulation with Bisection Optimization
.TRAN 0.1n 8n SweepOptimize = Opt1
+ Result = MaxVout$ Look at measure
+ Model = OptMod
* This measure finds the transition if it exists
.MEASURE Tran MaxVout Max v(D_Output) Goal = ‘v(Vdd)’
* This measure calculates the setup time value
.MEASURE Tran SetupTimeTrig v(Data)Val = ‘v(Vdd)/2’
+ Fall = 1
+ Targ v(Clock)Val = ‘v(Vdd)/2’
+ Rise = 1
* Optimization Model
.MODEL OptMod Opt
+ Method = Bisection
.OPTION Post Brief NoMod
```



First bisection value is mid-way between specified boundaries. First test value passes because measured value ≥ goal value (2.7 V in this case).

First test value becomes new upper test limit. Second test value is mid-way between new upper limit and lower boundary. Second test value fails.

Second test value becomes new lower limit. Third test value is mid-way between new lower limit and current upper limit.

Continue halving the test region until the interval between successive test values meets the criterion:

$$\delta = |X_n - X_{n-1}| \le relin \cdot |X_U - X_L|$$

then report the value X, (associated with the measured value the passed). If you select the *bisection* method, the reported value must correspond with the condition:

measured value - goal > 0

Figure 88 Bisection Example for Three Iterations

Results

The top plot in Figure 89 shows the relationship between the clock and the data pulses that determine the setup time. The bottom plot is the output transition.

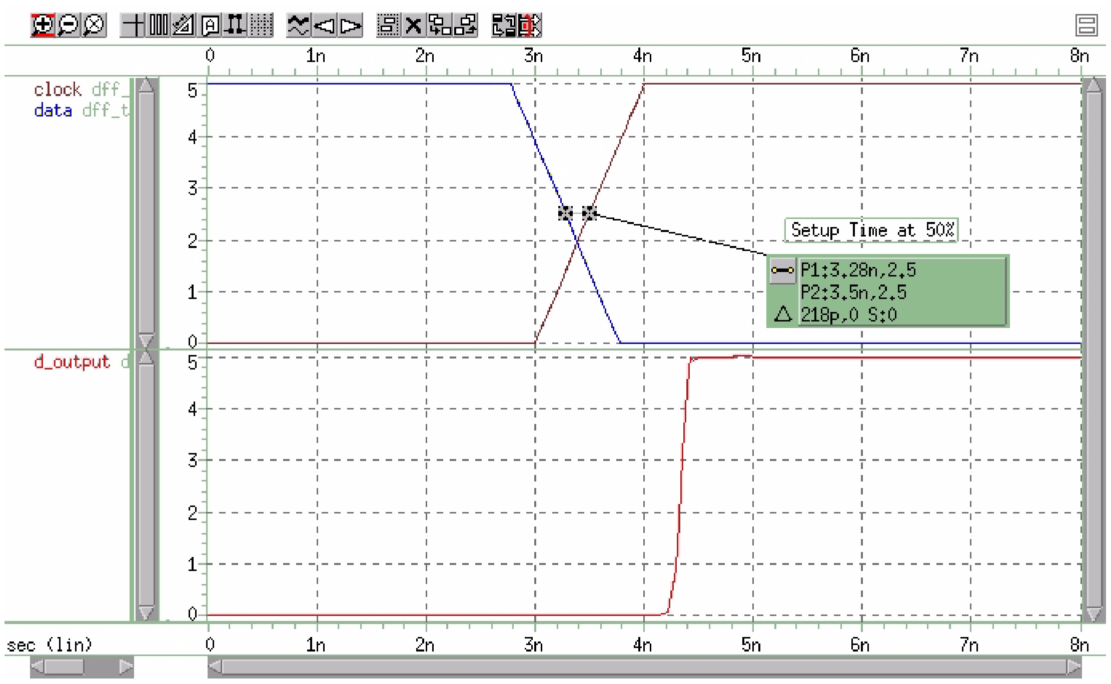


Figure 89   Transition at Minimum Setup Time

Find the actual value for the setup time in the “Optimization, Results” section of the input listing file:

```
optimization completed, the condition
relin = 1.0000E-03 is satisfied
**** optimized parameters opt1
.PARAM DelayTime = 1.7822n
...
maxvout = 5.0001E+00 at= 4.8984E-09
from     = .0000E+00    to= 8.0000E-09
setuptime= 2.1777E-10 targ= 3.5000E-09 trig= 3.2822E-09
```

This listing file excerpt shows that the optimal value for the setup time is 0.21777ns.

The top plot in Figure 90 on page 506 shows examples of early and late data transitions, and the transition at the minimum setup time. The bottom plot shows how the timing of the data transition affects the output transition. The following analysis statement produces these results:

```
* Sweep 3 values for DelayTime   Early   Optim   Late
.TRAN 0.1n 8n Sweep DelayTime Poi 3   0.0n    1.7822   5.0n
```

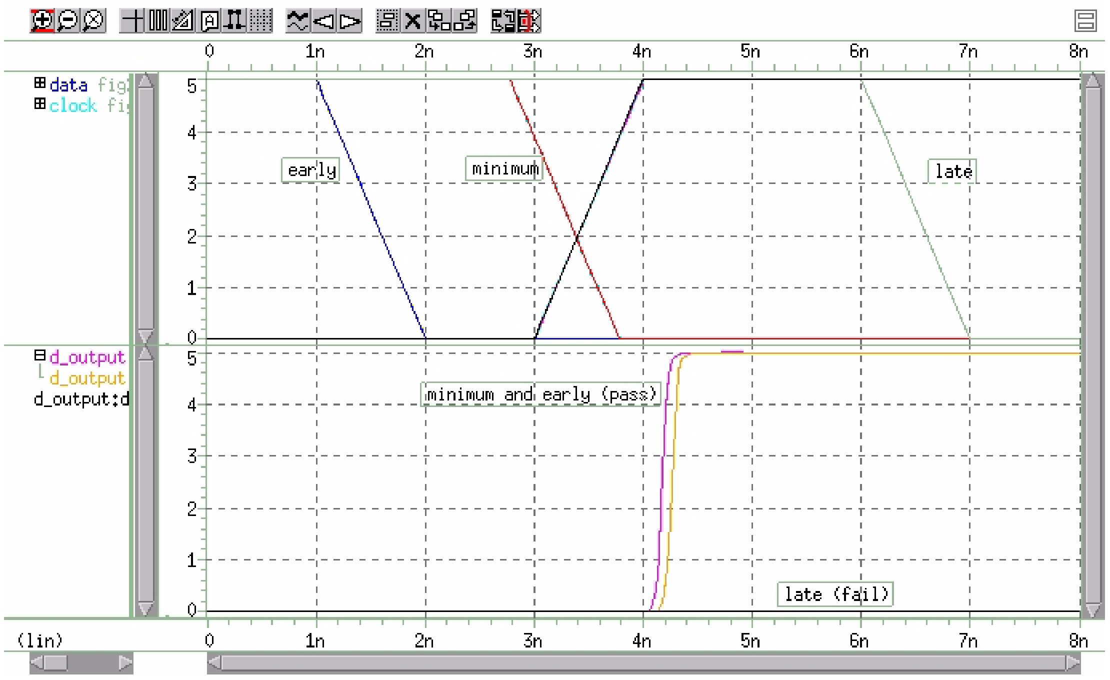


Figure 90   Early, Minimum, and Late Setup and Hold Times

This analysis produces the following results:

```
*** parameter DelayTime = .000E+00 *** $ Early
setuptime= 2.0000E-09 targ= 3.5000E-09   trig= 1.5000E-09
*** parameter DelayTime = 1.7822E-09 *** $ Optimal
setuptime= 2.1780E-10 targ= 3.5000E-09   trig= 3.2822E-09
*** parameter DelayTime = 5.000E-09 *** $ Late
setuptime= -3.0000E-09 targ= 3.5000E-09   trig= 6.5000E-09
```

Minimum Pulse Width Analysis

This example uses a pass/fail bisectional search to find a minimum pulse width required so the input pulse can propagate to the output of an inverter. It is based on demonstration netlist iva_a.sp, which is available in directory $<*installdir*>/demo/hspice/bisect. Figure 91 on page 507 shows the results of this demo.

Input Listing Directory

This input listing file is located in: $installdir/demo/hspice/bisect/inv_a.sp.

Results

Figure 91 shows results of pass/fail search, for two different capacitive loads.

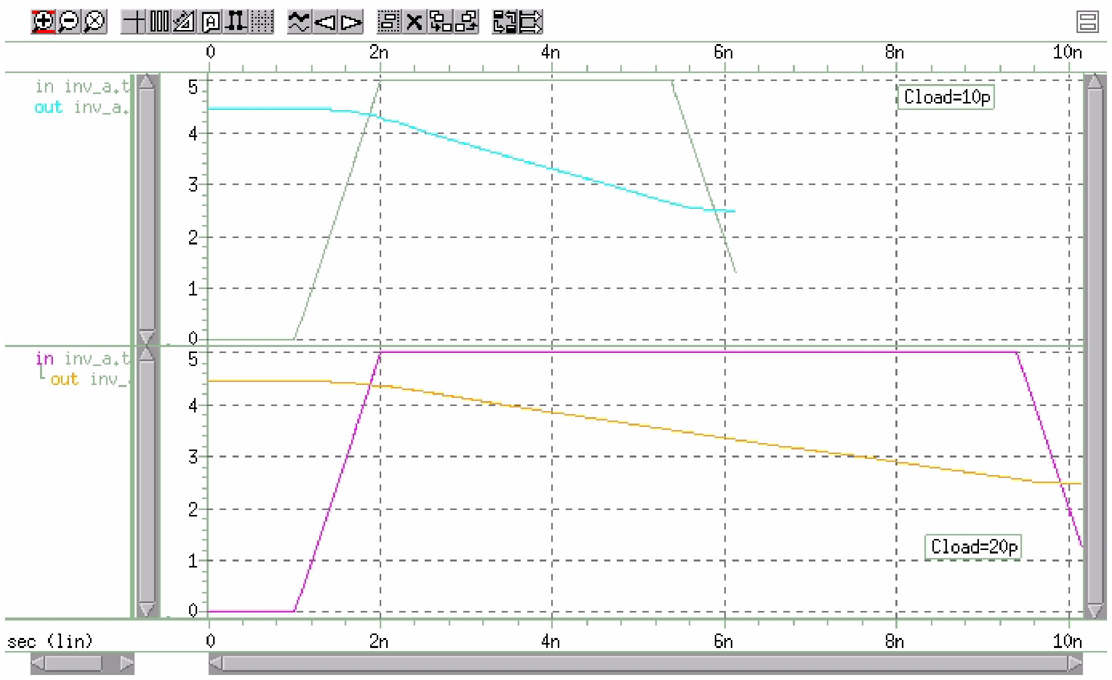


Figure 91 Results of Bisectional Pass/Fail Search

Pushout Bisection Methodology

For setup- or hold-time optimization analysis, a normal bisection method varies the input timing to find the point just before failure. At this point, delaying the input longer results in failure, and the output does not transition. In pushout analysis, instead of finding the last point just before failure, the first successful output transition is used as the golden target. You can then apply a maximum allowed pushout time to decide if the subsequent results are classified as passes or failures. Finding the optimized pushout result is similar to a normal bisection because both use a binary search to approach the desired solution. The main difference is the goal or the optimization criteria.

The following example (Figure 92 on page 508) shows a transition of Vin with a varying delay during a Vclck transition. When the “lower” input transitions, it indicates that the device being tested is functioning. The “upper” input does not transition, which indicates that the device is not functioning.

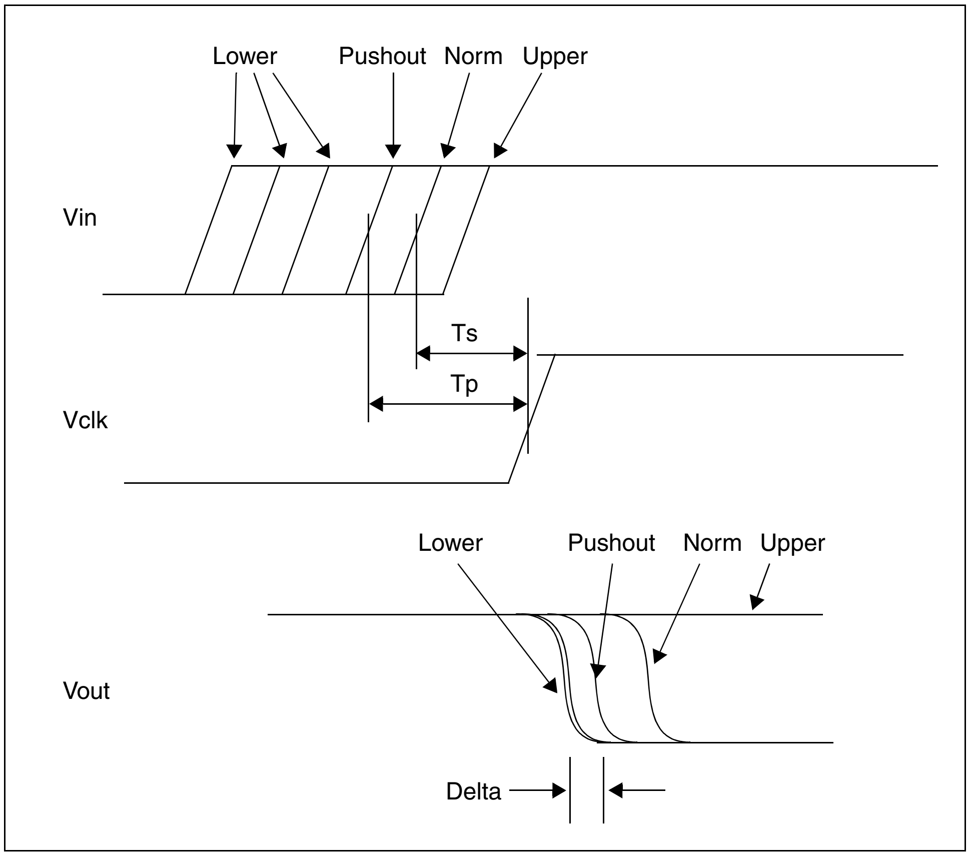


Figure 92 Pushout Bisection Example

Consider the pushout bisection example located in the following directory:

$*installdir*/demo/hspice/bisect/dff_push.sp

The transition of Vin is delayed by varying amounts with respect to a Vclk transition. For the Lower input transition, the output transitions and indicates that the device under test is functioning. For the Upper input transition, the output does not transition and indicates that the device is not functioning.

Normal bisection varies the input timing to find the point just before failure (called Norm here). At this point, a failure occurs when the device is delayed longer and the output does not transition. The circuit works at points between Lower and Norm, but the output transition is delayed from the lower conditions by setting Delta. This is called the Pushout. The pushout can also lie between Norm and Upper, which depends on your use of the lower or upper option.

In you use normal bisection in this example, the resulting gain is delaytime=7.374e-10 pushout=-1.672e-09. Instead, when setting pushout=0.01, the result is delaytime=3.918e-11 pushout=3.970e-09.

20

Analyzing Variability and Using the Variation Block

Introduces variability, describes how it can be defined in HSPICE, and introduces the Variation Block.

The topics are covered in the following sections:

Overview of Variation on Silicon

As semiconductor technologies migrate to ever-smaller geometries, larger relative variations in device characteristics are being observed. These fluctuations in device characteristics have been analyzed and classified for the purpose of dealing with the variations in manufacturing during the design phase. The following types of variations can be identified at the wafer level:

- Global variations from foundry, lot, or wafer processing
- Across wafer variations due to materials and gas flow, generally linear across the area of a chip
- Across wafer variations due to thermal, optical, and spin processes, generally linear across the area of a chip

As a result of microscopic random processes, local variations are observed between closely spaced identically designed devices. Microscopic variations include line edge roughness, finite number of dopant atoms in the channel, and atomic level oxide thickness changes.

In analog design, certain circuit characteristics can be made insensitive to global variations by applying the concept of matched devices; however these characteristics are then affected by the local variations. In digital designs for nanometer technologies, large local variations can cause unacceptable variations in path delays and signal slopes.

Large circuits suffer from spatial or position dependent variations, which create problems with clock skew for devices that are far apart. Finally device characteristics are affected by features in proximity (metal coverage, fill patterns, mechanical strain, shape variation due to lithography, etc.) and orientation. Some of these variations are systematic and can be accounted for in design or post-layout verification.

Historically, only the effects of variation on device characteristics (transistors, resistors, and capacitors) have been considered. In nanometer technologies, the variations on the interconnect also need to be taken into account because the relative variation in the resistance and capacitance has increased due to smaller wire width and inter conductor spacing.

These variations combined, summarized as parametric variability, dominate yield loss in the nanometer technologies. The circuits function in terms of connectivity, but do not meet specifications on metrics such as speed, leakage, or offset. For example while the threshold of MOS devices gets smaller, approaching 200mV, the variation in threshold gets larger, with standard deviation up to 30mV for short devices. Due to the low supply voltages, in combination with requirements for high speed, the circuits stop working with these large spreads in device characteristics. Therefore, simulating (or predicting) the effects of these variations on circuit response is increasingly important, in particular when considering the high mask costs and time to market constraints for the majority of today's products.

To be able to simulate the effects of the variations in device characteristics due to materials and manufacturing, they need to be described in a way that the simulator can handle. Traditionally, the global variations were specified through process corner files, and the other types of variations mentioned above were either guessed or ignored. In recent years, statistics blocks were added to the model files. They describe variations in terms of distributions on device model parameters. An even newer approach for defining variations is the Variation Block, described later in this chapter.

The following analyses are available in HSPICE to simulate the effects of variations on circuit response:

- Monte Carlo analysis is the traditional method for finding the variation in circuit response resulting from the parameter variations.
- DC and AC mismatch analyses are efficient methods for simulating the effects of local variations on a circuit's DC or AC response.

  To get satisfactory answers from these analyses, the variation definitions must have been generated for the target technology of the design, similar to device models.

Defining Variability in HSPICE

Three approaches are available for defining variability in HSPICE:

- Defining a Variation Block; for example,

  ```
  .Variation
     global and local variation definitions
  .End_Variation
  ```

- Defining variations on parameters; for example,

  ```
  .param var=agauss(20,1.2,3)
  ```

  For a discussion of this topic, see Appendix A, Statistical Analysis.

- Defining variations on models using `lot` and `dev` parameters in the model file; for example,

  ```
  vth0=0.6 lot/0.1 dev/0.02
  ```

  For a discussion of this topic, see Appendix A, Statistical Analysis.

The Variation Block approach replaces the older methods of defining variations on parameters and models in HSPICE because it best fulfills the requirements for simulating nanometer technology devices.

The advantages of the Variation Block over previous solutions are:

- The Variation Block consolidates variation definitions in single records.
- A clear distinction exists between Global, Local, and Spatial Variations.
- A subset of variation types can be selected in a dependent simulation.

- The syntax allows for defining Local and Global Variation as a function of device geometry, and Spatial Variation as a function of device location.
- Monte Carlo results derived from the Variation Block are consistent with those from DCMatch or ACMatch analyses.

The following sections present these topics:

Overview of the Variation Block

The characteristics of circuits produced in semiconductor processing are subject to variability, as is the case for any manufactured product. For a given target technology, the nominal device characteristics are described with a set of parameters, which applies to a certain device model (for example, BSIM4). In HSPICE, the variability of the model parameters is described through a *Variation Block*. A Variation Block is a container for specifying variations introduced by the effects in manufacturing on geometry and model parameters.

For the purpose of dealing with variations in HSPICE, they are modeled as Global, Local, or Spatial variations.

- Global Variations are defined as variations in device characteristics from lot to lot, wafer to wafer and chip to chip; they are caused by variations in starting material and differences between equipment and procedures. Global Variations affect all devices with the same model name in the same way.
- Local Variations are defined as variations between devices in proximity, or with common centroid layout on the same chip; they are caused by the microscopic variations in materials and geometry, and affect different devices differently.
- Spatial Variations are defined as variations due to the physical arrangement of the devices on the layout; they are caused by gradients from material properties, imperfections of lenses, and spin processes. The dependence on distance means that large or distributed circuits are more affected by Spatial Variations.

All three classes can be described in the Variation Block in a very flexible way by user-defined expressions. Since there are currently no industry-wide standards for specifying process variability, this feature allows each company to implement their own proprietary model for variability. The Variation Block is generally provided by a modeling group, very similar to device models (for example, BSIM) because it must be created specifically for each technology from test circuits.

Like a model, the Variation Block can be part of a library which is encrypted; therefore, the content is not accessible to the designers. They can add a separate Variation Block in their netlist, to define options and variations on generic elements. See Control Options and Syntax and Variations of Element Parameters.

The structure of the Variation Block allows for building expressions to model interdependence and hierarchy of the variations. For example, one random variable can control the variation in oxide thickness of both PMOS and NMOS devices, as it is generally the same for both types of devices.

Note that the earlier methods for specifying variation are not compatible with the Variation Block. For controlling the behavior of Variation Blocks, see section on options for controlling the behavior. The Variation Block is currently used for Monte Carlo, DC and AC mismatch analyses; for a description of these analyses, see Chapter 21, Monte Carlo Analysis Using the Variation Block Flow and Chapter 22, Mismatch Analyses, respectively.

For the functions available to build expressions as presented in the next sections, see Parameters and Functions on page 273.

Variation Block Structure

A Variation Block is structured in four sections:

- general section
- subblock for Global Variations
- subblock for Local Variations
- subblock for Spatial Variations

This discussion presents the syntax of a Variation Block, followed by discussion of the contents of the four sections.

```
.Variation
   Define options
   Define common parameters that apply to all subblocks
   .Global_Variation
      Define the univariate independent random variables
      Define additional random variables through tranformations
      Define variations of model parameters
   .End_Global_Variation
   .Local_Variation
      Define the univariate independent random variables
      Define additional random variables through transformations
      Define variations of model parameters
      .Element_Variation
         Define variations of element parameters
      .End_Element_Variation
   .End_Local_Variation
   .Spatial_Variation
      Define the univariate independent random variables
      Define additional random variables through tranformations
      Define variation of model parameters
   .End_Spatial_Variation
.End_Variation
```

General Section

In the general section, options can be defined that control the variability analyses which use the content of the Variation Block. Options can be specified, one per logical record.

Note:

> ".Option" (with a leading period) does not work for options specified in the Variation Block.

The correct syntax is:

```
Option OptionName = value
```

See Control Options and Syntax at the end of this chapter for a listing and description of Variation Block control options.

Parameters, also, can be defined that apply to all subblocks in which variations are specified; however, these cannot contain any distribution related functions. Parameters defined within a Variation Block are completely independent of parameters defined outside of it.

Example: `parameter PI=3.1416`

Subblocks for Global, Local and Spatial Variations

Within the variation subblocks, univariate independent random variables can be defined. These are random variables with specific distributions over a certain sample space. Additional random variables can be generated through transformations. These random variables form the basis for correlations and complicated distributions.

A basic rule of the Variation Block approach is to have the model definition on the top level, instead of inside of a subcircuit, as necessary in the old approach.In all three subblocks, variations on model parameters can be defined. This is where global or Local Variations on the parameters of semiconductor devices are specified.

A special section within the subblock for Local Variations allows for defining Local Variations on elements. This is either for specifying local temperature variations or variations on generic elements that do not have a model, as used early in the pre-layout design phase, or for off-chip components; for example, resistors and capacitors. Local and Global variation support the block operator brackets described in Group Operator {...} and Subexpressions on page 531.

Independent Random Variables

When describing variations, a standard normal (Gaussian) distribution is assumed, unless otherwise specified explicitly. This default behavior is explained in later sections. Other types of distributions or correlations are modeled by applying transformations to the independent random variables. These independent random variables come from three basic distributions:

- Uniform distribution: defined over the range from -0.5 to 0.5: `U()`
- Normal distribution: with mean=0 and variance=1, default range +/-4: `N()`
- User-defined cumulative distribution function: CDF(xyPairs)

  If f(x) is the probability density of a random variable x, then the cumulative distribution function is the integral of f(x). A cumulative distribution function can be approximated by a piecewise linear function, which can be described as a sequence of pairs of points [xi, yi]. The following rules apply:

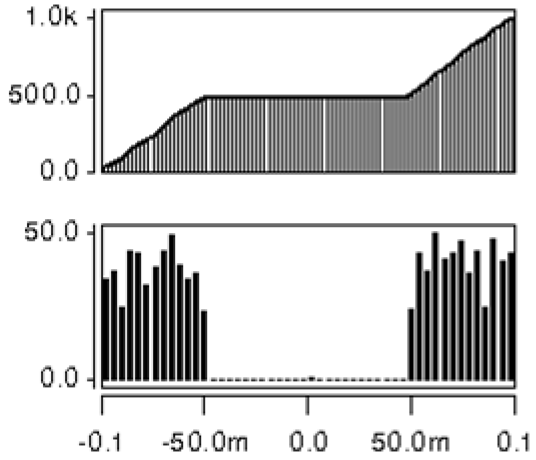
  1. at least two pairs are required
  2. white space or a comma is required between each number
  3. the CDF starts at zero: y1=0
  4. CDF ends at one: yn=1

5. xi values must be monotonically increasing xi+1 > xi
6. yi values must be monotonically non-decreasing yi+1 $\geq$ yi
7. $\int_{-\infty}^{\infty} x \cdot f(x) \cdot dx = 0$

Example

```
Parameter var=CDF(-0.1 0 -0.05 0.5 0.05 0.5 0.1 1.0)
```

This CDF produces for 1000 samples:



The distributions N() and U() do not accept any arguments.

The syntax for defining independent random variables is:

```
Parameter a=U() b=N()   c=CDF(x1,y1,......xn,yn)
```

These distributions cannot be defined within expressions; variables must be assigned and then the variables can be used within expressions.

Dependent Random Variables

To model distributions which are more complicated than the ones which are available through the predefined independent random variables, transformations can be applied by using expressions on independent random variables. A dependent variable can also be created as a function of more than one independent random variable to express correlation.

Example 1

This example creates a random variable with normal distribution, with mean A and standard deviation B.

```
Parameter var=N()  Y='A + B * var'
```

Example 2

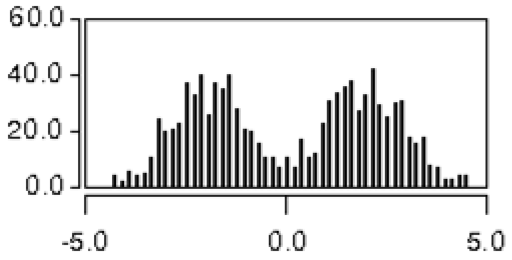
This example creates a random variable with a uniform distribution from D to E, where D and E are arbitrary constants.

```
Parameter var=U()  Y='0.5*(D+E) + (E-D) * var'
```

Example 3

This example creates a random variable with two peaks.

```
Parameter a=N() b=N() c='a+2*sgn(b)'
```



Absolute Versus Relative Variation

By default, the specified variation is absolute, which means additive to the original model or element parameter; however, sometimes it is more appropriate to specify relative variations that are defined by appending a space and a “%” sign to the expression. The simulator divides the result of the expression by 100, and multiplies by the original parameter value and the random number from the appropriate generator to calculate the change.

Example

In the following example, the variation on the threshold parameter `vth0` is specified as absolute (sigma of 80 or 70mV), the variation on the mobility `u0` as relative (15 or 13 percent).

```
.Global_Variation
   Nmos snps20N vth0=0.08 u0=15 %
   Pmos snps20P vth0=0.07 u0=13 %
.End_Global_Variation
```

Variations on Model Parameters

Variations on model parameters can be defined in subblocks for Global, Local, and Spatial Variation. In the course of the simulation, these variations are then applied to the specified device model parameters. Model parameter variations are described with the following syntax:

```
Model_Type Model_Name Model_Parameter=Expression for Sigma
```

The syntax `Model_Parameter=Expression for Sigma` is a shorthand notation for `Variation_in_Model_Parameter='Expression for Sigma`'.

If the expression references only constants and parameters that evaluate to constants, then a Gaussian variation with zero mean and a sigma equal to the expression is automatically implied. To describe variation as a function of previously defined independent random variables, use the construct `'Perturb()'`, with the following syntax:

```
Model_Type Model_Name Model_Parameter=Perturb('Expression')
```

The expression for sigma may need to be enclosed in quotes, see the general HSPICE rules for Using Algebraic Expressions on page 278.

The following lines define a global Variation, with implicit normal distribution, with zero mean and sigma of 10, on the parameter `rsh` of resistors with model `Rpoly`.

```
.Global_Variation
   R Rpoly rsh=10
.End_Global_Variation
```

In the next example, the independent variable `Toxvar` is used to model global Variations on oxide thickness. `Toxvar` is an independent random variable with a normal distribution, with mean=0 and sigma=1. In the device models `nch` and `pch`, `Toxvar` is applied to the parameters tox with a different multiplier. The oxide thicknesses in the two models vary in parallel; they are correlated.

```
.Global_Variation
   Parameter Toxvar=N()
   Nmos nch tox=Perturb('7e-10*Toxvar')
   Pmos pch tox=Perturb('8e-10*Toxvar')
.End_Global_Variation
```

HSPICE supports the following model types: NMOS, PMOS, R, Q, D, and C.

Variations can only be defined on parameters that are explicitly specified in the associated device model.

For a list of supported models and parameters see “Supported Models and Parameters for HSPICE variability” on SolvNet. For binned models, variations can be defined separately by specifying the model name with the bin extension; for example, devices from bins 1 and 2 receive different variation on the parameter `lint`, which models length variation:

```
Nmos snps20N.1 lint=10n
Nmos snps20N.2 lint=12n
```

Variations on Subcircuit Parameters

The Variation Block allows for defining variation on parameters, which are specified in the subcircuit definition record, with a default value. The default parameter values can be overwritten by specifying them at subcircuit instantiation.

The syntax is:

```
Subckt SubcktName Parameter='expression for sigma'
```

The following rules apply for these types of definitions:

- Only parameters that are defined as formal numeric arguments on the subcircuit definition record can be subject to variation. (This is the line which starts with `.SUBCKT` and possibly has continuation lines.)
- The subcircuit must not be defined within another subcircuit.
- If the subcircuit contains a model, then variations on the model parameters, as described in section, Variations on Model Parameters, are not supported. Instead, variations need to be defined on a subcircuit parameter and the parameter used inside of the model.

Subcircuit Parameters Variation Use Models

The subckt parameters variation feature addresses the following three needs:

- A component is defined with an expression, which is not available in a model:

  ```
  r1 1 0 'Rsh*l/w*(1+b1*(tanh(b2*abs(v(1,0)/l))))'
  ```

This expression models a voltage dependent resistor, with non-linear dependence not available in a traditional model. If this resistor is called within a subcircuit, and parameters are specified on the subcircuit definition record, then variation can be modeled, for example, on `Rsh`, `l` and `w`:

Example 1

```
i1 0 1 1m
x1 1 0 rtanh
.subckt rtanh a b rsh=1k w=1u l=1u
.param b1 = -0.4   b2 = 8u
r1 a b 'Rsh*l/w*(1+b1*(tanh(b2*abs(v(a,b)/l))))'
.ends
.Variation
   .Global_Variation
      subckt rtanh rsh=10 %
   .End_Global_Variation
   .Local_Variation
      subckt rtanh rsh=3 %
   .End_Local_Variation
.End_Variation
```

- A component is represented by a network, and the subcomponents must have the same value when local variations are specified:

Example 2

```
r1 end1 middle rmodel w='w+dw' l='l/2+dl'
r2 end2 middle rmodel w='w+dw' l='l/2+dl'
c1 middle sub cmodel w=w l=l
```

In this example, the resistor has a center tap, where a capacitor is connected. If these three components are defined within a subcircuit and parameters dw, dl, and rsh are defined on the subcircuit definition record, then the two resistors of the same network always have the same value; local variations only cause different instantiations of the subcircuit to have different equivalent resistance between the network terminals.

- Users need to calculate the value of a device model parameter through an equation because the built-in equations are not adequate:

Example 3

```
.subckt nch n1 n2 n3 n4 dvth0_glob=0 dvth0_loc=0 du0_glob=0
du0_loc=0
+ dtox=0  dlint=0  dwint=0 l=60n w=120n as='w*90n' ad='w*90n'
...
.param vth0_base=0.345 u0_base=0.015
.param vth0_geo=function1(w,l,temper,vth0_base)
.param u0_geo=function2(w,l,temper,u0_base)
.param dvth0_geo=function3(w,l,dvth0_loc)
.param du0_geo=function4(w,l,temper,du0_loc)
M1 n1 n2 n3 n4 nch25 w=w l=l as=as ad=ad ...
.MODEL nch25 NMOS LEVEL = 54
+ vth0='vth0_geo+dvth0_glob+dvth0_geo'
u0='u0_geo*(1+du0_glob)*(1+du0_geo)'
+ tox='2.6n+dtox'   lint='2.1n+dlint'   wint='5.3n+dwint' ...
.ends
X1 d1 g1 s1 b1 nch l=60n w=150n
X2 d2 g2 s2 b2 nch l=80n w=120n
.Variation
   .Global_Variation
       subckt nch dvth0_glob=0.03 dtox=0.12n du0_glob=0.2
dlint=2n dwint=3n
   .End_Global_Variation
   .Local_Variation
       subckt nch dvth0_loc=2.0m dtox=0.03n du0_loc=0.03
dlint=21p dwint=47p
   .End_Local_Variation
.End_Variation
```

The values of model parameters vth0 and u0 are defined through user defined equations (*function1* and *function2*), with dependency on device size and temperature. This necessitates a local model (nch25). The parameters with variations are declared on the subcircuit definition line. In this example, global and local variations are processed differently through the subcircuit, therefore the respective variations have to be specified on separate parameters. Global variations (dvth0_glob and du0_glob) are applied to the model parameters vth0 and u0 directly. Local variation definitions (dvth0_loc and du0_loc) are adjusted for device size using *function3* and *function4*, and result then in the variations dvth0_geo and du0_geo applied to the model parameters vth0 and u0.

While this use model is supported, it is not desirable because it leads to one model per device, which is inefficient in terms of memory.

Variations of Element Parameters

Devices are not only affected by variations in the underlying model parameters, but also through variations of properties specified at instantiation of an element, or variations on implied properties, such as local temperature. Also, early in the design phase, passive devices sometimes have only a nominal value, but no model as yet because no decision has been made on a specific implementation. For these elements, variations can be specified on the implicit value parameter; for example: R1 1 0 **1k**.

Variations on element parameters are only supported for Local Variations; they are defined in a section within the Local Variation subblock.

Element parameter Variations are described with the following syntax:

```
Element_Type Element_Parameter = 'Expression for Sigma'
```

The syntax `Element_Parameter = 'Expression for Sigma'` is shorthand notation for:

`Variation_in_Element_Parameter =` *`Expression for Sigma`*

If the expression references only constants and parameters that evaluate to constants, then a Gaussian variation with zero mean and a sigma equal to the expression is automatically implied. To describe variation as a function of previously defined independent random variables, use the construct `Perturb()`, with the following syntax:
`Element_Type Element_Parameter = Perturb(`*`Expression`*`)`

The expressions may need to be enclosed in quotes, see the general HSPICE rules for Using Algebraic Expressions on page 278. See Parameters and Expressions on page 49 for limitations.

The following lines define a normal distribution with sigma of 10 on the resistors without model:

```
.Element_Variation
   R R=10
.End_Element_Variation
```

In the following example, only resistor `ra2` is affected by the temperature variation specified with a uniform distribution from 0 to 10 degrees (the resistor is located next to a power device).

```
ra1 1 0 1k
rb1 2 0 1k
ra2 3 0 rpoly l=10u w=1u
rb2 4 0 rpoly l=10u w=1u
.model rpoly r rsh=100 tc1=0.01
.Variation
   .Local_Variation
     .Element_Variation
        Parameter tempvar=U()
        R(Element_Name~='ra*' && Model_Name~='rpoly')
        + dtemp=perturb('10*tempvar+5')
     .End_Element_Variation
   .End_Local_Variation
.End_Variation
```

Because different classes of devices might be affected differently, a selection mechanism based on element name and model name is provided by using a condition clause:

```
Element_Type(condition_clause) Element_Parameter= 'Expression
for Sigma'
```

The condition clause allows for specifying variations on selected elements, according to their name or associated model. Wildcard substitutions can be indicated as “?” for single character and “*” for multiple characters.

Examples for condition clause syntax are:

```
Element_Type(model_name~='modelNameA')
Element_Type(element_name~='elNameB')
Element_Type(model_name~='modelNameC' OPERATOR
   element_name~='elNameD') par='exp'
Element_Type(model_name~='modelNameE' OPERATOR
   model_name~='modelNameF') par='exp'
Element_Type(element_name~='elNameG' OPERATOR
   element_name~='elNameH')  par='exp'
```

Where `OPERATOR` can be `&& (AND)`, `|| (OR)`. The operator “`~=`” stands for “matches”.

All pattern matching operations are case-insensitive. A leading subcircuit prefix is ignored when matching the element name.

Example

In this example, only resistor `ra1` varies.

```
ra1 1 0 1k
rb1 2 0 1k
.Variation
   .Local_Variation
   .Element_Variation
      R(element_name~='ra*') R=20
   .End_Element_Variation
   .End_Local_Variation
.End_Variation
```

Supported element types and their parameters are:

Table 61 Supported elements and parameters

Element	Parameter
M	DTEMP
R	Rval* DTEMP
C	Cval* DTEM
Q	AREA DTEMP
D	DTEMP
L	Lval* DTEMP
I	DCval* mag phase
V	DCval* mag phase

The asterisk "*" denotes implicit value parameter. The DTEMP parameter is implicit; it does not have to be specified on the element instantiation line.

Example for Voltage Source

Netlist element:	V1 1 0 0.1 AC
Variation definition:	V DC=5% (5% of 0.1) V MAG=5% (5% of 1) V PHASE=5 (5 degrees)

Access Functions

Certain variations depend on element geometry, as defined with parameters at instantiation. The access function (only supported for the Variation Block) `Get_E()` allows for accessing these parameters in expressions by using the following syntax:

```
Get_E(Element_Parameter)
```

Where `Element_Parameter` is the name of an element parameter, which must be defined on the instantiation line (except for the `DTEMP` parameter and the multiplier `M`). This access function is generally used for specifying variations as a function of device geometry. The `Get_E()` access function reports the effective device geometry, after resolving parameters, scales and adjustments by process parameters, such as, xw, xl, wint, lint. Refer to the HSPICE Reference Manual: MOSFET Models for equations which depend on the LEVEL of interest and Geometric Scaling for Diode Models in the *HSPICE Reference Manual: Device and Element Models* for scaling equations.

For example, the local variation on the threshold is often specified as inversely proportional to the square root of the total area of the device, as calculated from the product of the element parameters `W`, `L`, and `M`.

```
Nmos nch vth0='1.234e-9/sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
```

Another function allows for accessing the values of global parameters by using the following syntax:

```
Get_P(Global_Parameter)
```

The parameter value is taken from the circuit context: from the subcircuit, if defined inside, otherwise from the top level. In the following example, the resistor variation is determined by sweep parameter “tol”:

```
.param tol=1
ra1 1 0 1k
i1 0 1 1m
.Variation
   .Local_Variation
     .Element_Variation
        R R='get_P(tol)' %
     .End_Element_Variation
   .End_Local_Variation
.End_Variation
.dc tol 1 5 1 monte=100
```

If a variation must be expressed as a function of a simulator option (specified as `.option=optionval` outside of the Variation Block), the access function `Get_O` is available, using the construct `Get_O(option_name)`. For example, if the element parameter `nf` (number of fingers) is used with some advanced models, the device width reported by the `Get_E` function depends on the value of `.OPTION WNFLAG`. For variation as a function of total device area, the following definition produces the expected results, independent of the settings of `WNFLAG`:

```
        vth0 = `6.0621e-9/sqrt(Get_E(W)*Get_E(L)*Get_E(M)*\\
               (1-Get_O(WNFLAG)+Get_O(WNFLAG)*get_E(NF)))'
```

Spatial Variation

To make the Spatial Variation useful, the simulator needs to know the coordinate of a particular device. The element instantiation must therefore be extended to include placement information. For example, for an MOS device:

```
Mid Dn Gn Sn Bn ModelName w=width l=length x=xcoor y=ycoor
```

In the Spatial Variation definition, the element coordinates are accessed using the `Get_E()` function.

For the current release, HSPICE supports only netlists with a single subcircuit, with devices on the top level and/or in the subcircuit. All devices of the model which has Spatial Variation defined, must have coordinates, and these must be specified with numbers (no parameters allowed).

Special Rules Regarding Variation Block Usage

The contents of the Variation Block are generally created by a foundry. To safeguard against unintentional overwriting of these variation definitions, two features are in place:

- The name-space of the Variation Block is separate from the netlist contents.
- Once a variation is specified on a parameter, it can not be redefined later, even in `.ALTER` statements. For example, if a user wants to change the corners defined in a model library file with an `.ALTER` statement, then the Variation Block must be specified in a separate **.lib* section.

Variation Block Examples

This simple Variation Block is used in the example netlists *opampdcm.sp* and *opampmc.sp* which are available in the HSPICE demo directory:

$*installdir*/demo/hspice/variability

The following variations are defined in the example:

Global Variations on vth0 (absolute)
Global Variations on u0 (relative)
Local Variations on vth0 (absolute), as a function of device area
Local Variations on u0 (relative), as a function of device area
Local Variation on the implicit value of resistors (relative)

```
.Variation
  .Global_Variation
     Nmos snps20N vth0=0.07 u0=10 %
     Pmos snps20P vth0=0.08 u0=8 %
  .End_Global_Variation
  .Local_Variation
     Nmos snps20N vth0='1.234e-9/
          sqrt(Get_E(W)*get_E(L)*Get_E(M))'
     + u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
     Pmos snps20P vth0='1.234e-9/
          sqrt(Get_E(W)*get_E(L)*Get_E(M))'
     + u0='2.345e-6/sqrt(Get_E(W)*get_E(L)*get_E(M))' %
     .Element_Variation
       R r=10 %
     .End_Element_Variation
  .End_Local_Variation
.End_Variation
```

Principal Component-based Global Variation Modeling

In this example, the independent random variables `A1`, `A2`, and `A3` are the principal components, on which all variations (`nmos` and `pmos`) are modeled. See [1] for details.

```
.Global_Variation
      Parameter A1=N() A2=N() A3=N()
      Nmos nch
+  tox =Perturb('-6.2E-12*A1-8.1E-12*A2-2.7E-12*A3')
+  vth0=Perturb('-3.6E-03*A1+8.9E-03*A2-1.5E-03*A3')
+  cjn =Perturb('-3.2E-06*A1+6.7E-06*A2-4.3E-06*A3')
+  u0  =Perturb(' 5.6E-04*A1-9.7E-04*A2+7.6E-04*A3')
+  ....
      Pmos pch
+  tox =Perturb('-7.5E-12*A1-6.9E-12*A2-8.8E-12*A3')
+  vth0=Perturb('-7.4E-03*A1+3.3E-03*A2-7.2E-03*A3')
+  cjn =Perturb('-5.0E-06*A1+8.9E-06*A2-3.2E-06*A3')
+  u0  =Perturb(' 7.6E-04*A1-4.3E-04*A2+4.8E-04*A3')
+  ....
.End_Global_Variation
```

Local Variation Example for Submicron Technology

This Local Variation data was created using the methodology outlined in [2]. Note the different dependencies on `w` and `l` for the different parameters.

```
.Local_Variation
    Nmos nch
+  tox ='3.1e-07/sqrt(Get_E(L)*Get_E(W)*Get_E(M))' %
+  wint ='6.2e-12/sqrt(Get_E(L)*Get_E(M))'
+  lint ='2.0e-12/sqrt(Get_E(W)*Get_E(M))'
+  nch ='1.9e-06/sqrt(Get_E(L)*Get_E(W)*Get_E(M))' %
.End_Local_Variation
```

Spatial Variation Example

```
.Variation
   .Spatial_Variation
      Parameter a = N( )
      Parameter b = U( )
      Parameter Pi = 3.14159265
      Parameter Angle = 'Pi*2*b'
      NMOS snps20n
+ vth0 = Perturb('20*a*sqrt(Get_E(x)* Get_E(x)+ Get_E(y)*
Get_E(y)) \\
         *cos(Angle-atan(Get_E(y)/Get_E(x))-Get_E(x)<0?Pi:0))')
    .End_Spatial_Variation
.End_Variation
```

The Spatial Variation is specified as a plane with a slope sigma of 20mV/mm, and arbitrary rotation on the chip surface.

Group Operator {...} and Subexpressions

To improve readability of complex variation specifications, a group operator {...} and subexpressions are available. Used within a defined group, subexpressions can reference element and global parameters.

(As an efficient alternative, consider using the Common Model Interface (CMI). Consult your HSPICE technical support team for application note and source code access.)

Syntax

```
ModelType ModelName {
Parameter ...
ModelParameter= ...
}
```

The group operator `{...}` separates variation definitions group by group. Each group is based on one model, which means all parameters defined inside a group operator are specific to this model. A group definition starts after the Model Name, and must end after the last model parameter specification of the same Model Name.

`Parameter` definitions support expressions with `Get_E()` or `Get_P()`. `ModelParameter` definitions have no leading `Parameter` key.

Syntax Extension with Bins

```
ModelType ModelName {
Parameter ...
ModelParameter= ...
ModelName.1 ModelParameter= ...
........
ModelName.m ModelParameter= ...
}
```

Model parameter definitions within a group before the first bin name (`ModelName.1` in the example) apply to all bins; whereas the following definition is bin specific:

```
ModelName.1 ModelParameter= ...
```

Example

In this example, note that the expressions before `NMOS` apply to all bins, whereas those for `mn.12` are bin specific.

```
.Global_Variation
   Parameter PG1=N() PG2=N() PG3=N()
+ dxl=' 4.3e-9 * PG1 '
+ dvth0='0.02 * (-0.29 * PG1 + 0.95 * PG2)'
+ dtoxe='1.3e-10 * (0.39 * PG1 -0.87 * PG2 + 0.28 * PG3)'
+ F1='1.0/(2*SQRT(dvth0*dvth0))'
+ F_FF='(-dvth0+SQRT(dvth0*dvth0))*F1'
+ F_SS='(dvth0+SQRT(dvth0*dvth0))*F1'
NMOS mn.12 {
   parameter u0varg='-dvth0*(F_FF*2.1+F_SS*0.6)'
   xl= perturb(dxl)
   vth0=perturb(dvth0)
   lvth0=perturb(dvth0*(F_FF*0.097+F_SS*0.054))
   u0=perturb('u0varg') % wu0=perturb('u0varg') %
   lu0=perturb('u0varg') % pu0=perturb('u0varg') %
   toxe=perturb(dtoxe) toxp=perturb(dtoxe)
}
.End_Global_Variation
.Local_Variation
NMOS mn.12 {
parameter sqrtarea='SQRT(get_E(W)*get_E(L)*get_E(M)'
vth0='1.2e-9/sqrtarea'   u0='2.3e-6/sqrtarea'
}
.End_Local_Variation
```

Rules for Using the Group Operator

The following rules apply when using the block operator:

- Independent random variables can not be defined inside of a group.
- Condition clauses are not supported inside of a group.
- Any specifications that appear at the same line and after the opening '{' are ignored; a parameter definition should begin at a new line after the bracket.
- Group operators only support model parameter, *not* subcircuit parameter definitions.
- A group with the same `ModelType` and `ModelName` can be defined only once; HSPICE will abort if any duplicated group definitions are found.

Parameter Hierarchies

Succession relations between parameters defined inside and outside of a group have the following restrictions: parameters defined inside a group can not conflict with those defined outside of it. However, the same parameter can be redefined inside another group, and these are invisible to each other.

The parameter hierarchies are as follows:

```
.Param a=2                      Defined outside of Variation Block; can be
                                referenced using get_P() syntax

.Variation
    parameter a=3
.Global_Variation               Global parameter within Variation Block;
                                can be used in all subblocks
   parameter b='a*get_P(a)'     b = 6; valid in global variation subblock
   NMOS nch {
   parameter c=0.4*b            c = 2.4; only visible in this group;
   ....                         can be redefined in other groups
   }
   PMOS pch {
   parameter c='-0.3*b'         c = -1.8; c is only visible in this group;
   ....                         can be redefined in other groups
   }
.End_Global_Variation
.Local_Variation
parameter b='2*a*get_P(a)'      b=12; valid in local variation subblock
.Element_Variation
R r='0.1*b' %                   Relative sigma of r is 0.1*2*2*3*0.01=0.012
.End_Element_Variation
.End_Local_Variation
.End_Variation
```

Interconnect Variation in Star-RCXT with the HSPICE Flow

In its Z-2006.12 release, the Synopsys layout extraction tool, Star-RCXT, established a Sensitivity-Based Extraction Flow, which can generate a variation-aware netlist to interpret and produce simulation results based on the probability distribution of interconnect variations. The currently available methodology of running worst case corners produces pessimistic results, as opposed to the new method, which calculates the actual distribution, and which then allows for selecting design limits based on yield.

Using the Sensitivity-Based Extraction Flow, Star-RCXT extracts resistors and capacitors associated with the interconnect. HSPICE then works as a post-processor to do statistical analysis with the output file from Star-RCXT which contains sensitivity information that is required to support Variation Block-based ACMatch, DCMatch, and Monte Carlo analyses.

Refer to the *Star-RCXT User Guide*, Chapter 11: Variation-Aware Extraction for more information.

Variation Block and Statistical Sensitivity Coefficients

Statistical sensitivity coefficients corresponding to each parasitic value are generated and provided by Star-RCXT. These coefficients measure the expected change in the capacitance/resistance due to the variation of an interconnect process parameter. By definition, the fractional change of capacitance/resistance value due to a unit variation in a specific parameter is the statistical sensitivity of the capacitance/resistance in question with respect to that parameter.

The combination of the nominal capacitance/resistance tables, and the corresponding statistical sensitivity coefficients, provides the necessary and sufficient coverage for all possible effects of parameter variations on capacitance/resistance. This eliminates the need for using extensive sets of capacitance tables, and provides a realistic coverage of all possible ranges of random variation.

Application of statistical sensitivity coefficients requires that the parameter variations be small. This restriction is acceptable for nanometer semiconductor processes since a large part of the process variation tends to be systematic and is considered and modeled under the scope of deterministic process variation.

Given the distribution of parameter variations, based on statistical sensitivity information, you can get the statistical effects on capacitance and resistance values in Monte Carlo, DCMatch, and ACMatch analyses. Interconnect variability is defined in its own block, the `Interconnect_Variation` subblock. Currently, the variation is restricted to the global level. The Interconnect_Variation subblock is created by Star-RCXT and is included as part of the post-layout netlist.

Usage Example and Input Syntax

1: Interconnect Variation Block

The information in the Variation Parameters section is re-coded as follows:

```
.Variation
   .Interconnect_Variation
     .Global_Variation
        ID= param_id Name = param_name [R_Sensitivity_Type =
        + param_type] [C_Sensitivity_Type = param_type]
        [L_Sensitivity_Type = param_type] [K_Sensitivity_Type =
        + param_type] [CV= coeff_of_var]
      ...
     .End_Global_Variation
   .End_Interconnect_Variation
.End_Variation
```

Argument	Description
param_id	Is a nonnegative integer to uniquely identify the parameter. In this way, every parameter is associated with a different integer. These unique identifiers are used in the parasitic section to represent the sensitivity information.
param_name	Are alphanumeric characters without any spaces or meta characters.
param_type	Valid values are N, D, or X. These refer to the form of the sensitivity expression and indicate if the particular parameter variation appears in the numerator, the denominator, or does not influence the element value. If not specified, the default is X.
coeff_of_var	This argument is numeric and optional. The default value is 1.

Variation blocks have global scope and the above definition should appear outside any subcircuit definitions. `R_Sensitivity_Type`, `L_Sensitivity_Type`, and `K_Sensitivity_Type` help to define the form of the sensitivity expression. This is a generalization of the Taylor series-based variation form, $1 + \sum_{i \varepsilon I} s_i \Delta p_i$, to the more general Pade′ approximation: $\dfrac{1 + \sum s_i \Delta p_i}{1 + \sum_{j \varepsilon J} s_j \Delta p_j}$.

The index sets *I* and *J* are disjoint, for example, a parameter can influence either the numerator or the denominator, but not both.

In the Star-RCXT-VX and HSPICE releases, only resistors have the more general Pade′ form, capacitors have the Taylor series form, and inductors (normal and K-Matrix style) have no variation.

Example of the extended NETNAME-style information

```
.Variation
  .Interconnect_Variation
    .Global_Variation
     ID=0  Name=ME1_T   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.06
     ID=1  Name=ME1_W   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.04
     ID=2  Name=ME1_R   R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.05
     ID=3  Name=ME12_T  R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.06
     ID=4  Name=ME12_ER R_Sensitvity_Type=X C_Sensitvity_Type=N
CV=0.02
     ID=5  Name=ME2_T   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.08
     ID=6  Name=ME2_W   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.07
     ID=7  Name=ME2_R   R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.04
     ID=8  Name=ME23_T   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.054
     ID=9  Name=ME23_ER R_Sensitvity_Type=X C_Sensitvity_Type=N
CV=0.02
     ID=10 Name=ME3_T   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.08
     ID=11 Name=ME3_W   R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.07
     ID=12 Name=ME3_R   R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.04
    .End_Global_Variation
  .End_Interconnect_Variation
.End_Variation
```

In the example above,

- The ID field must be a non-negative integer. HSPICE uses the ID field to link variation information to the sensitivity in the C and R records.
- The Name field is alphanumeric and should not contain any white space or meta characters. The Name field is used only for output annotation.
- The CV field is numeric and the CV field is interpreted as the standard deviation for a (default) normal distribution.

2: Model Card in the Header Section

The purpose of the model card in the header section is to communicate to HSPICE the model name used in the parasitic section for the resistors as well as the reference temperature. The reference temperature is equal to the GLOBAL_TEMPERATURE in ITF with units in celsius.

Syntax

```
.model <model_name> R Tref=<global_temperature>
```

Example

```
.model resStar R Tref=25
```

3: Parasitic Section

The resistance and capacitance records take the form:

```
Cxxx <node1> <node2> Value SENS [param_id, param_id, …]=
    [sens_coeff, sens_coeff, …]
 Rxxx <node1> <node2> <model_name> R=<value> TC1=<val> TC2=<val>
    SENS [param_id, param_id, …] = [sens_coeff, sens_coeff, …]
........
```

Examples

The following is an example of a C record in NETNAME format:

```
C1 G2[21]:F12 Y2:897 0.699 Sens [0,1,5,6] =
    [0.009,0.001,0.006,0.010]
C2 X3:962 RX[12]:F74 0.324 Sens [0,1,5,8] =
    [0.010,0.006,0.017,-0.003]
```

The following is an example of an R record in NETNAME format:

```
R1 G2[21]:F12 G2[21]:8 resStar R=0.699 TC1=0.0023 TC2=4e-7
    Sens [5,6,7] = [0.51,0.64,0.86]
```

The `Sens` key defines the start of sensitivity information and the two vectors are the sparse sensitivity indices and the corresponding values. The first vector may contain only ordered non-negative integers that map to the `Interconnect_Variation` section, while the second vector of real numbers is interpreted as the sensitivities. The lengths of the two vectors must match. There must be one blank space between the `Sens` keyword and the sensitivity indices.

Note:

For interconnect output, see Interconnect Output Formats in Chapter 21, Monte Carlo Analysis Using the Variation Block Flow.

Control Options and Syntax

Options can be specified, one per logical record in a Variation Block. Several of the listed options are useful if a Variation Block is part of a model file that a designer cannot edit. However, you can add a Variation Block with options to control how the contents of all Variation Blocks are used in the analysis. For Monte Carlo-specific options: see Chapter 21, Monte Carlo Analysis Using the Variation Block Flow.

Note:

No period is required before the word Option in the Variation Block, and is, in fact, illegal.

The following options can be specified within the Variation Block:

- `Option Normal_Limit=Value`
  Limits the range of the Normal distributions. The default value is 4, i.e., numbers in the range +/- 4 σ are generated. The range allowed is 0.1 to 6.0.

  This option allows a foundry to limit the perturbations to parameter ranges where a model is still valid.

- `Option Ignore_Variation_Block=Yes`
  Ignores the Variation Block and executes earlier style variations (traditional Monte Carlo analysis). By default, the contents of the variation block are executed and other definitions (`AGAUSS`, `GAUSS`, `AUNIF`, `UNIF`, `LOT`, and `DEV`) are ignored. Previous methods of specifying variations on parameters and models are not compatible with the Variation Block. By default, the contents of the Variation Block are used and all other specifications are ignored. Thus no changes are required in existing netlists other than adding the Variation Block.

- `Option Ignore_Local_Variation=Yes`
  Excludes effects of local variations in simulation. Default is `No`.
- `Option Ignore_Global_Variation=Yes`
  Excludes effects of global variations in simulation. Default is `No`.
- `Option Ignore_Spatial_Variation=Yes`
  Excludes effects of spatial variations in simulation. Default is `No`.
- `Option Ignore_Interconnect_Variation=Yes`
  Excludes effects of interconnect variations in simulation. Default is `No`. (See Interconnect Variation in Star-RCXT with the HSPICE Flow.)
- `Option Output_Sigma_Value=Value`
  Use to specify the sigma value of the results of Monte Carlo, DCMatch, and ACMatch analyses. Default is 1, range is 1 to 10. Note that this option only changes the output listings and that the input sigma is not affected.
- `Option Vary_Only Subckts=SubcktList`
  Use either this option to limit variation to the specified subcircuits or the following option, but not both. Actual subcircuit names are specified here (not the hierarchical names).
- `Option Do_Not_Vary Subckts=SubcktList`
  Excludes variation on the specified subcircuits. Use either this option to limit variation to the specified subcircuits or the one above, *but not both*. Actual subcircuit names are specified here (not the hierarchical names).

References

[1] K. Singhal and V. Visvanathan: Statistical device models from worst case files and electrical test data. IEEE Trans. Semiconductor Manufacturing, November 1999. (Global variation modeling by principal components)

[2] P.G. Drennan and C.C. McAndrew: Understanding MOSFET mismatch for analog design. IEEE J. Solid-State Circuits, March 2003. (Modeling mismatch in nanometer technologies)

21

Monte Carlo Analysis Using the Variation Block Flow

Describes enhanced Monte Carlo analysis in HSPICE.

These topics are covered in the following sections:

Overview

Monte Carlo analysis is the generic tool for simulating the effects of variations in device characteristics on circuit performance. The variations in device characteristics are expressed as distributions on the underlying model parameters. For each sample of the Monte Carlo analysis, random values are assigned to these parameters and a complete simulation is executed, producing one or more measurement results. The series of results from a particular measurement represent a distribution, which can be characterized by statistical terms; for example, mean value and standard deviation (σ). With increasing number of samples, the shape of the distribution gets better defined with the effect that the two quantities converge to their final values.

A standard way of analyzing the results is by arranging them in bins. Each bin represents how many results fall into a certain range (slice) of the overall distribution. A plot of these bins is a histogram, which shows the shape of the distribution as the number of results versus slice. As the number of samples increases, the shape of the histogram gets smoother.

The ultimate interest of Monte Carlo simulation is to find out how the distribution in circuit response relates to the specification. The aspect of yield is considered here:

- What is the percentage of devices which meet the specification?
- Is the design centered with respect to the specification?

Closely related is the aspect of over-design. This is when the circuit characteristics are within specification with a wide margin, which could be at the expense of area or power and ultimately cost.

A typical design process is iterative, first for finding a solution which meets the nominal specification, and then moving on to a solution that meets yield and economic constraints, including the effects of variations in device characteristics. In this optimization process, it helps to understand the relationship of the design parameters to the circuit response, and the relationships of the different types of circuit response. This information is available after running Monte Carlo analysis and can best be presented by Pairs Plots. This is a matrix of two-dimensional plots for investigating pair-wise relationships and exploring the data interactively. HSPICE does not produce such plots, but makes the necessary data available from Monte Carlo simulation. Figure 93 shows an example of a Pairs Plot from a simple resistive divider:

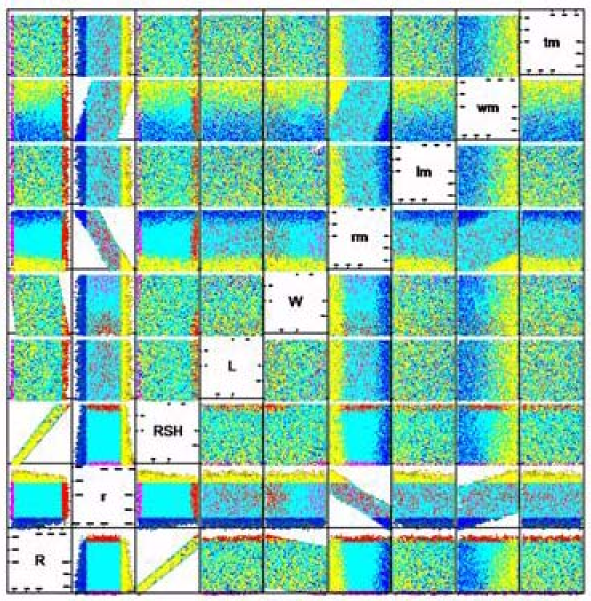


Figure 93 Pairs Plot example

An application note, “Pairs Plots from HSPICE Monte Carlo,” describes the basic ideas and includes a MATLAB® script to create such a plot. Contact the Synopsys Support Center for a copy of the application note.

Monte Carlo analysis is computationally expensive; therefore, other types of analysis have been created that produce certain results more efficiently. For cases where only the effects of variations on the DC or AC response of a circuit is of interest, methods called DCMatch and ACMatch analyses can be used; for a description, see Chapter 22, Mismatch Analyses.

Monte Carlo Analysis in HSPICE

Monte Carlo analysis has been available in HSPICE for some time and is based on two approaches:

- defining distributions on global parameters (using `AGAUSS`, `GAUSS`, `UNIF`, and `AUNIF`) in a netlist; for example,

  ```
  .param var=agauss(20,1.2,3)
  ```

- defining distributions on model parameters using `DEV` and `LOT` constructs in a model file; for example,

  ```
  vth0=0.6 lot/0.1 dev/0.02
  ```

The above two methods are documented in Appendix A, Statistical Analysis.

To satisfy some key requirements for modern semiconductor technologies, a new approach is available based on the Variation Block, which is described in detail in Chapter 20, Analyzing Variability and Using the Variation Block. This new approach is not compatible with the earlier ones; see the first option in Monte Carlo-Specific Variation Block Options on page 546 for ways to select one or the other method.

Figure 94 on page 544 shows the Monte Carlo simulation flow when global and local variations are specified. Sample number 1 of a Monte Carlo analysis is always executed with nominal values and no variation. For subsequent samples, HSPICE updates the parameters specified for variation in the variation block with random values. For global variations, a specified parameter is changed by the same random value for all elements that share a common model. For local variation, the specified parameter is changed by a different random value for each element. The changes due to global and local variations are additive and are saved in a file for post-processing. When all the elements have been updated, the simulation is executed and the measurement results are saved. When all the requested samples have been simulated, HSPICE calculates the statistics of the measurement results and includes them in the run listing.

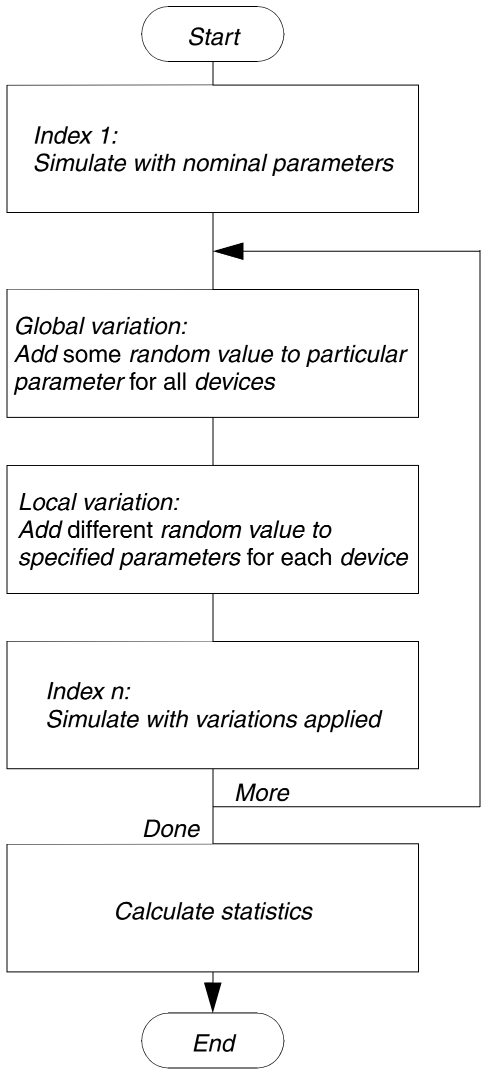

Start
Index 1:
Simulate with nominal parameters
Global variation:
Add some random value to particular parameter for all devices
Local variation:
Add different random value to specified parameters for each device
Index n:
Simulate with variations applied
More
Done
Calculate statistics
End

Figure 94 Monte Carlo analysis flow in HSPICE

Input Syntax

Monte Carlo analysis is always executed in conjunction with another analysis (see Traditional Monte Carlo Analysis in Appendix A for full discussion):

```
.DC sweepVar start stop step [SWEEP MONTE=MCCommand]
.AC type step start stop [SWEEP MONTE=MCcommand]
.TRAN step start stop [SWEEP MONTE=MCCommand]
```

Syntax for *MCcommand*:

```
MONTE=val|list(num)|valfirstrun=num|
+ list(<num1:num2><num3><num4:num5>)
```

Parameter	Description
val	Specifies the number of random samples to produce.
val firstrun=num	Specifies the sample number on which the simulation starts.
list (num)	Specifies the sample number to execute.
list(<num1:num2><num3> <num4:num5>)	Samples from num1 to num2, sample num3, and samples from num4 to num5 are executed (parentheses are optional).

The parameter values and results are always the same for a particular sample, whether generated in one pass or using `firstrun` or the `list` syntax. Therefore, Monte Carlo analyses can be split or distributed and the results spliced together with scripts.

DC Sweep Examples

In these examples a DC sweep is applied to a parameter `k`. In the first case, 10 samples are produced. In the second case, five samples are produced, starting with sample number 6. In the last two examples, samples 5, 6, 7 and 10 are simulated.

```
.dc k start=2 stop=4 step=0.5 monte=10
.dc k start=2 stop=4 step=0.5 monte=5 firstrun=6
.dc k start=2 stop=4 step=0.5 monte=list (5:7 10)
```

Monte Carlo-Specific Variation Block Options

Options for the deprecated Monte Carlo style are ignored when simulations based on the Variation Block are executed.

The options described in Chapter 19 (Control Options and Syntax) control Monte Carlo analysis. In addition, the following Monte Carlo-specific options can be specified in the first section of the Variation Block:

- `Option Random_Generator = [Default | MSG]`

  Specifies the random number generator used in Variation Block based Monte Carlo analysis. `Random_Generator=MSG` invokes the generator from previous releases. `Random_Generator=Default` uses a longer cycle generator.

- `Option Stream =[x | Random | Default]`

  Specifies an integer stream number for random number generator (only for Variation Block). The minimum value of x is 1, the maximum value of x is 20; If `Stream=Random`, HSPICE creates a random stream number between 1 and 20 according to the system clock, and prints it in the *.lis* file for the user to debug. If `Stream=Default`, then it is equivalent to `Stream=1`.

- Option `Normal_Limit=Value`

  Limits the range for the numbers generated by the random number generator for Normal distributions. The default value is 4, i.e., numbers in the range +/- 4 σ are generated. The range allowed is 0.1 to 6.0.

- `Option Output_Sigma_Value=Value`

  Specifies the sigma value of the results. Default is 1, range is 1 to 10. Note that the input sigma is not affected.

- `Option Print_Only Subckts=SubcktList`
  Use either this option to limit output in the *.mc#* file to the specified subcircuits or the following one, *but not both*. Actual subcircuit names are specified here (not the hierarchical names).

- `Option Do_Not_Print Subckts=SubcktList`

  Use either this option to exclude output from the specified subcircuits to the *.mc#* file or the one above, but not both. Actual subcircuit names are specified here (not the hierarchical names).

Example for Ignore_Global and Normal_Limit Options

In the following example, global variations are not simulated, and the Normal distributions are exercised to 6σ. For information regarding Local and Global Variations, see Subblocks for Global, Local and Spatial Variations in the chapter Analyzing Variability and Using the Variation Block.

```
.Variation
      Option Ignore_Global_Variation=Yes
      Option Normal_limit=6
      .Global_Variation
      Definitions for global variations
   .End_Global_Variation
      .Local_Variation
      Definitions for local variations
   .End_Local_Variation
.End_Variation
```

Output

Simulation Listing

The output listing file contains a summary of the names of all input parameters that are subject to global or local variations. For each sample, the measured results are printed. Finally, the statistics for the measured data are reported.

Example: Output Listing

Partial printout of an output listing:

```
MONTE CARLO DEFINITIONS
Random number generator is default, and stream = 1
Global variations:    model               parameter
                      snps20n             vth0
                      snps20n             u0

Local variations:     model               parameter
                      snps20n             vth0
                      snps20n             u0

Element variations:   element             parameter
                       r1                 r

 *** monte carlo  index =     1 ***
   systoffset1=  1.3997E-03
 *** monte carlo  index =     2 ***
   systoffset1= -9.2694E-04

MONTE CARLO STATISTICS
   meas_variable = systoffset
   mean  =   1.4398m      varian =   1.2391u
   sigma =   1.1132m      avgdev = 893.3815u
   max   =   5.3035m      min    =  -1.4532m
 1-sigma =   1.1132m      median =   1.4184m
```

Measurement Output File

Measure commands cause simulation results to be saved for each sample, along with its index number. Depending on the analysis type, the name of the result file has an extension of ms#, ma#, or mt#, where the # denotes the regular sequence number for HSPICE output files. The changes in all parameter values subject to variation are saved in a file with an extension of *.mcs#*, *.mca#*, or *.mct#*, depending on the analysis type.

Parameter File

The structure of this file is the same as for regular measure files. Therefore, the data can be read with the same post-processing tools (for example, CosmosScope). In the header section, the names of the parameters and independent variables are presented as follows:

- for global variation on model parameter:
  `Model_Name@Parameter_Name@ID`
- for local variation on model parameter:
  `Element_Name@Model_Name@Parameter_Name@ID`
- for local variation on element parameter:
  `Element_Name@Parameter_Name@ID`

- for interconnect variation:
  `Param_Name@ID` and `Element_Name@ID`
- for independent variables:
  `Variable_Name@ID`

where: `ID` is a three-character string for identifying the type of the parameter as follows.

Table 62 and Table 63 on page 549 list the independent and dependent parameter types, respectively.

Table 62 Independent Parameter Type Identifier

First character		**Second character**		**Third character**	
I	Independent variable	G	Global	N	Normal distribution
		L	Local	U	Uniform distribution
		S	Spatial	C	CDF distribution

Table 63 Dependent Parameter Type Identifier

First character		**Second character**		**Third character**	
M	Model	G	Global	R	Relative
E	Element	L	Local	A	Absolute
S	Subcircuit Variables	S	Spatial		
T	Interconnect				

The independent variables include explicitly specified random variables [for example: A=N()], and the internally generated random variables for implicit definitions in the expressions for sigma (for example: Nmos snps20 vth0=0.07). Values for parameters that have absolute variation specified in the variation block are reported as absolute deviation from the nominal value. Values for parameters that have relative variation specified are reported as a relative deviation in percent. The printed value for parameter “status” is “1” for a successful simulation, and “0” for a failed simulation. If the netlist or the model or both are encrypted, hash codes are printed in the appropriate places, which are meaningful to HSPICE for External Sampling.

Example: *mcs#* File

```
index       snps20n@vth0@IGN             snps20n@u0@IGN
            snps20n@vth0@MGA             snps20n@u0@MGR
            xi82.mn6@snps20n@vth0@MLA  xi82.mn6@snps20n@u0@MLR
            xi82.mn1@vth0@ILN            xi82.mn1@u0@ILN
            xi82.mn1@snps20n@vth0@MLA  xi82.mn1@snps20n@u0@MLR
            xi82.mn2@vth0@ILN            xi82.mn2@u0@ILN
            xi82.mn2@snps20n@vth0@MLA  xi82.mn2@snps20n@u0@MLR
            xi82.rcomp@r@ILN             xi82.rcomp@r@ELR
            status           alter#
  1.0000         0.                0.              0.
                 0.                0.              0.
                 0.                0.              0.
                 0.                0.              0.
                 0.                0.
                 1.0               1.0000
  2.0000         0.6141            0.6284          4.299e-02
                 6.284e-02         2.1837          0.2184
                 1.7554            0.1755          1.6017
                 0.1602            0.4769          4.769e-02
                -1.0088            0.5350
```

In this example, the changes due to the global variations on parameters vth0 (absolute) and u0 (relative) are reported first, then the changes on each device due to local variations on the same parameters are reported next, and finally, the local variation on the parameter r of the element rcomp are reported. Note that the parameter value applied to the device for a particular sample is the nominal value, plus the reported change due to global variations, plus the reported change due to local variations, and so on.

The contents of this parameter file are useful for data mining. In connection with the measured data in the regular output file, the relationship of circuit response variation to parameter variation can be investigated by using, for example, a Pairs Plot as shown in Figure 93 on page 542.

Note:

The contents of this file are subject to change.

Interconnect Output Formats

The following is example output for interconnect variation testing. In the Monte Carlo sampling output file **.mc?#*, one identifier keyword for interconnect variation parameter is used. In the following, `IGN` is the extension for independent variables. `TGA` is the extension for dependent variables.The `t` is present for interconnect parameters:

```
$ DATA1 SOURCE='HSPICE' VERSION='Y-2006.09'
$ This file format is subject to change
.TITLE '* two capacitors for model parameter variation testing'
 index         fox_c_t@IGN      fox_a_t@IGN      ild_b_t@IGN
               imd1c_t@IGN      imd1d_t@IGN      imd2a_t@IGN
               r1@TGA           r2@TGA           r11@TGA
               r22@TGA          r211@TGA         r222@TGA
               c1@TGA           c2@TGA           c11@TGA
               c22@TGA
status           alter#
1.0000       0            0.           0.
             0.           0.           0.
             0.           0.           0.
             0.           0.           0.
             0.           0.           0.
             0.
             1.0          1.0000

2.0000       0.6141       0.6284       0.8866
             5.198e-02    1.2452      -1.5600
            -1.031e-02   -3.537e-04   -5.191e-02
             6.133e-05    2.464e-03    2.964e-03
             2.464e-04    3.037e-04    5.073e-04
             5.796e-04
             1.0          1.0000

3.0000      -0.1087       -0.6694       3.363e-02
             1.6842       -1.0088       0.5350
             3.551e-03     9.223e-05    1.782e-02
            -2.440e-04    -7.813e-04    1.220e-03
            -7.813e-05    -2.212e-04    5.374e-05
             1.055e-04
```

Sampling Options

This section presents the advanced sampling methods can be defined in the Variation Block.

Factorial Sampling

```
Option Sampling_Method=Factorial
```

Use this option to evaluate the circuit response at the extremes of variable ranges to get an idea of the worst and best case behavior. The response is evaluated at the center of the hypercube (nominal) and at all its corners in the full factorial sampling method (see Figure 95). There are $1+2^m$ samples for a space with *m* independent variables. To prevent large runaway jobs, the problem dimension is restricted to $m \leq 12$ which results in ~4K simulations. If the size constraint is violated, then the command is ignored and an error message is generated.

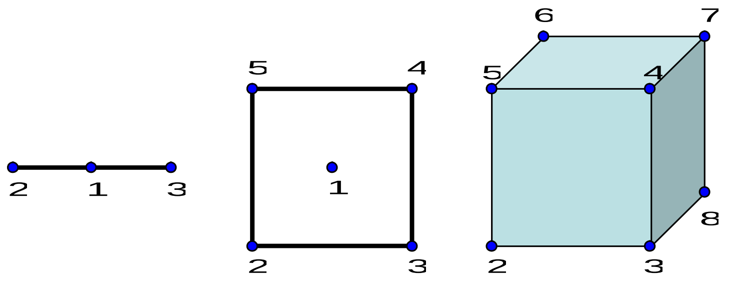


Figure 95 Factorial Hypercube Evaluation at Center and Corners

One-Factor-at-a-Time Sampling

```
Option Sampling_Method=OFAT
```

This sampling method varies One-Factor-At-a-Time, a Design of Experiments feature. [1] The number of samples is `2m+1` with `m` independent variables. The number specified on the Monte Carlo command is ignored, and `m` must be less than 2500. Sampling starts with no perturbation (nominal), then negative and positive perturbation only on the first parameter, negative and positive perturbation only on the second parameter, etc. The amounts of perturbation are the extreme values for a uniform distribution, and the Normal_Limit values for a normal distribution. Figure 96 illustrates OFAT examples.

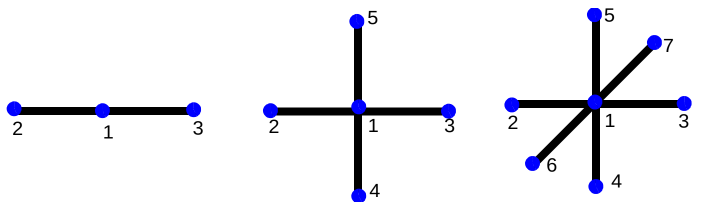


Figure 96 One-Factor-at-a-Time sampling

A suboption, `intervals=n`, creates 2mn+1 intervals with equal probability. The full syntax is then:

```
Option Sampling_Method=OFAT Intervals=2
```

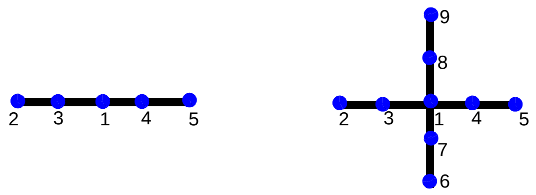


Figure 97   suboption intervals

Latin Hypercube Sampling

```
Option Sampling_Method=LHS
```

Latin Hypercube Sampling is an efficient sampling technique for Monte Carlo analysis of systems which are modeled on a computer and that have large numbers of variable parameters. [2] [3] Advantages of LHS are:

- The estimation error is smaller on most real world problems and a smaller sample size can be used to get the same precision in the results.
- The sample points are evenly spread over the entire range of variation of each parameter.
- The circuit is exercised over a wide range of parameter values and will often detect weak spots in the design.
- You can replicate the sampling using `Option Replicates=`*`Value`*

  This option runs replicates of the Latin Hypercube samples. The sample with Nominal conditions is simulated once. HSPICE repeats the LHS run the number of times specified by `Value`. For example, if, in a regular run, you have 10+1 (including nominal value) iterations, if you set `Replicates=2`, you generate 21 iterations (or 2x +1) Latin Hypercube Sampling.

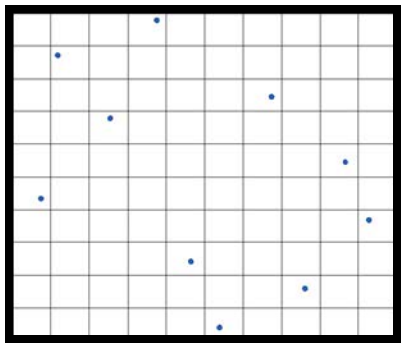


Figure 98 Example of the distribution of 10 sampling points in two dimensions

Note:

Monte Carlo options `firstrun` and `list` are not supported with LHS.

External Sampling

You can also execute a dataset of externally created perturbations instead of relying on one of the built-in sampling methods. External sampling allows design and process exploration tools to run statistical experiments, with the variables for each sample under their full control.

Usage Model for External Sampling

Use the following procedure

1. Execute HSPICE with a regular simulation command (AC, DC, TRAN) and monte=1 to produce a **.mc?0* file, which lists all the independent and dependent variables.
2. Create a data block outside of HSPICE with the desired perturbations on the independent variables, for global and local variations.
3. Run an HSPICE simulation using this externally generated data block content.
4. Repeat Steps 2 and 3, depending on the outcome of the previous experiments.

Syntax

The external sampling feature is defined in two parts in the Variation Block, a data block and the option:

```
.Variation
.Data test1
 Index p1@IGN   snps20p@vth0@ILN  x1.mn1@q1@ILN   x2.mn1@vth0@ILN
1     .0            .0                 .0                .0
2     .0          0.1086               .0                .0
3     .3            .0                 .0                .0
4     .0            .0                1.0                .0
5     .0            .0                 .0               1.0
6     .0            .0                1.0              -1.0
.Enddata

Option Sampling_Method=External Block_Name=test1
.End_Variation
```

The data block syntax is the same as for the regular HSPICE data block from `.Data` to `.Enddata`. The first variable is always the index. All identifiers for the variables start with “I” because this is the only variable type which can be set externally. The feature itself is invoked by specifying the external sampling method, with the appropriate block name.

To run a particular sample from the data block use the following `MCcommand`:

```
monte = list(num) or monte = list(<num1:num2> <num3> <num4:num5>)
```

If the netlist or the model or both are encrypted, the hash codes printed in the parameter file are recognized by HSPICE when reading in the data block.

Additional rules:

1. HSPICE does not check the range of the values in the supplied data block against option value Normal_Limit.
2. Independent random variables which are not specified in the data block are set to zero (no variation).

Application Considerations

If the variations that are applied are too large, generally caused by the combinations of variation specified in the variation block and the value of `Normal_Limit`, some circuits can show abnormal behavior and produce unrealistic results for certain samples. This can distort the summary statistics reported by HSPICE at the end of the Monte Carlo simulation. Therefore, you should review the individual measurement results for outliers and analyze them properly. For example, when plotting measurement results as a function of index in CosmosScope, the outliers are readily apparent.

Known Limitations

One Dimensional Monte Carlo Simulation

Running a Monte Carlo simulation with variation block does not support a one dimensional sweep such as `.DC sweep MONTE=50`. The run fails to converge and HSPICE issues an error message.

Use the following workaround:

Insert a dummy one point sweep, for example:

```
.param dummy = 1
.dc dummy 1 1 1 monte=50
.measure dc v1 find v(node1) at = 1
```

The Monte Carlo analysis runs properly and reports all the data points.

Troubleshooting Monte Carlo Issues

Independent Random Variable Assignments

If you add additional independent random variables to a Monte Carlo run you might see that none of these variables have any impact on the simulation run. But you might see differences in statistical results between simulations with and without these additional independent random variables. This may raise skepticism, as there is no difference in the results if you run the case more than once with the same set of independent random variables.

The difference is not due to varying the number independent random variables but because of the way random values are assigned to them. Consider the following two cases two cases where the only difference is the number of independent random variables.

Let two random variables be defined in case 1:

```
.variation
  .spatial_variation
        Parameter a = N( )
        Parameter b = N( )
        Parameter Slope = 'a/50u'
        Parameter Pi = 3.14159265
        Parameter Angle = 'Pi*2*b'
R rmodel rsh=Perturb('Slope*sqrt(Get_E(x)* Get_E(x)+ Get_E(y)*
Get_E(y)) \\
       *cos(Angle-atan(get_e(y)/get_e(x))-(get_E(x)<0?Pi:0))')
  .end_spatial_variation
.end_variation
```

Random variables `a` and `b` are used to calculate variations in sheet resistivity as a function of a resistor's coordinates.

Let case 2 have four random variables defined:

```
.variation
  .spatial_variation
        Parameter a = N( )
        Parameter b = N( )
        Parameter c = N( )
        Parameter d = N( )
        Parameter Slope = 'a/50u'
        Parameter Pi = 3.14159265
        Parameter Angle = 'Pi*2*b'
R rmodel rsh=Perturb('Slope*sqrt(Get_E(x)* Get_E(x)+ Get_E(y)*
Get_E(y))\\
       *cos(Angle-atan(get_e(y)/get_e(x))-(get_E(x)<0?Pi:0))')
  .end_spatial_variation
.end_variation
```

Random variables `a` and ' are used in the Variation Block; `c` and `d` are not used.

During the Monte Carlo sweep, a pseudo random number generator creates an array of random values (`Random1, Random2,...,RandomN`) and assigns them to each an independent random variable.

In case 1, the random number assignment is:

```
Monte=1 -- a=Random1 b=Random2
Monte=2 -- a=Random3 b=Random4
 ...
Monte=N -- a=Random2N-1 b=Random2N
```

In case 2, the random number assignment is:

```
Monte=1 -- a=Random1 b=Random2 c=Random3 d=Random4
Monte=2 -- a=Random5 b=Random6 c=Random7 d=Random8
  ...
Monte=N -- a=Random4N-3 b= Random4N-2 c=Random4N-1 d=Random4N
```

Here, although `Random1` through `RandomN` are the same in both cases, the sequence of assignment to independent random variables differs. Hence, the individual samples of `a` (or `b`) differ between the two simulations. As a consequence, at low sample numbers the difference in the standard deviation (sigma) of the distributions of `a` (or `b`) might be quite large. For higher sample numbers, the differences get smaller, according to the general convergence rate of Monte Carlo results of

$$1\sqrt{n}$$

where n is the number of samples.

Because a pseudo random number generator is used in HSPICE, repeated simulations generate the same set of statistical results for a given set of independent random variables. The user can change the random number sequences at each run by defining in the Variation Block:

$$\text{Option Stream} = val \leq , \; val \leq$$

In the obsolete traditional Monte Carlo style, this is the equivalent of setting:

```
.option seed=val
```

When using Monte Carlo simulation, you should keep in mind that there is always uncertainty associated with this method in the relationship of one sample to the overall population.

References

[1] V. Czitrom: One-Factor-at-a-Time Versus Designed Experiments. The American Statistician, pp.126-131, May 1999.

[2] M.D. McKay, R.J. Beckman, and W.J. Conover: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics, pp. 239-245, 1979.

[3] M. Stein: Large Sample Properties of Simulations Using Latin Hypercube Sampling. Technometrics, pp. 143-151, 1987.

22 Mismatch Analyses

Describes the use of DC and AC mismatch analyses in HSPICE.

DCMatch and ACMatch analyses are efficient techniques for computing the effects of variations on a circuit's DC or AC response. The variation definitions are taken from the Variation Block. Both methods are small signal analyses, similar to noise analysis. Unlike the traditional Monte Carlo analysis, these new methods do not rely on sampling, and are therefore significantly faster. The Monte Carlo results converge to those from DCMatch or ACMatch analysis for a large number of samples, provided that the circuit characteristics are close to linear in Monte Carlo analysis (for example no clipping or hysteresis).

DCMatch and ACMatch analyses are affected by the control options specified in the Variation Block, see Control Options and Syntax in the chapter Analyzing Variability and Using the Variation Block.

These topics are covered in the following sections:

Mismatch

Variations in materials and processing steps are the source of differences in the characteristics of identically designed devices in close proximity on the same integrated circuit. These are random time-independent variations by nature and are collectively called *mismatch*.

Mismatch is one of the key limiting factors in analog signal processing. It affects more and more circuit types as device dimensions and signal swings are reduced. Mismatch is a function of the geometry of the devices involved, their spatial relationship (distance and orientation) and their environment.

This chapter discusses the following mismatch analyses:

- DCMatch Analysis
- ACMatch Analysis

DCMatch Analysis

When the effects of variation on DC response of a circuit are of interest, a method called DC mismatch (DCMatch) analysis can be used.

In DCMatch analysis, the combined effects of variations of all devices on a specified node voltage or branch current are determined. The primary purpose is to consider the effects of Local variations (that is, for devices in close proximity). DCMatch analysis also allows for identifying groups of matched devices (that is, devices that should be implemented on the layout according to special rules). A secondary set of results is calculated from the influences of Global and Spatial Variations, which is useful for investigating whether their effects on circuit response are much smaller than the effects of Local variations, when optimizing a design.

DCMatch analysis is based on the following dependencies and assumptions:

- variations in device characteristics are modeled through variations in the underlying model parameters.
- effects on a circuit's DC solution are small and can be modeled as a linear combination of the variations in the random variables.

In HSPICE, the variations in model parameters are defined in the Variation Block (see Chapter 20, Analyzing Variability and Using the Variation Block). Those definitions are used to calculate the variation in DC response. DCMatch analysis runs either from a default operating point or for each value of the independent variable in a DC sweep. The default output is in the form of tables containing the sorted contributions of the relevant devices to the total variation, as well as information on matched devices. In the current implementation, a heuristic algorithm makes a best guess effort to identify matched devices. This means that the results are suggestions only. In addition to the table, the total variation and contributions of selected devices can be output using `.PROBE` and `.MEASURE` commands.

Input Syntax

```
.DCMatch OUTVAR [THRESHOLD=T] [FILE=string] [INTERVAL=Int]
```

Parameter	Description
OUTVAR	One or more node voltages, voltage differences for a node pair, or currents through an independent voltage source.
THRESHOLD	Report devices with a relative variance contribution above Threshold in the summary table. ■ T=0: reports results for all devices ■ T<0: suppresses table output; however, individual results are still available through .PROBE or .MEASURE statements. The upper limit for `T` is 1, but at least 10 devices are reported, or all if there are less than 10. Default value is 0.01.
FILE	Valid file name for the output tables. Default is `basename.dm#` where “`#`” is the usual sequence number for HSPICE output files.
INTERVAL	Applies only if a DC sweep is specified. *Int* is a positive integer. A summary is printed at the first sweep point, then for each subsequent increment of *Int*, and then, if not already printed, at the final sweep point.

Note:

If more than one DCMatch analysis is specified per simulation, only the last statement is used.

Example 1

In this example, HSPICE reports DCMatch variations on the voltage of node 9, the voltage difference between nodes 4 and 2, and on the current through the source VCC.

```
.DCMatch V(9) V(4,2) I(VCC)
```

Example 2

In this example, the variable `XVal` is being swept in the DC command from 1k to 9k in increments of 1k. DCMatch variations are calculated for the voltage on node `out`. Tables with DCMatch results are generated for the set XVal={1K, 4K, 7K, 9K}.

```
.DC XVal Start=1K Stop=9K Step=1K
.DCMatch V(out) Interval=3
```

DCMatch Table Output

For each output variable and sweep point, HSPICE generates a result record that includes setup information, total variations, and a table with the sorted contributions of the relevant devices. The individual entries are:

- Sweep or operating points for which the table is generated
- Name of the output variable
- DC value of this output variable
- Values used for DCMatch options
- Output sigma due to combined Global, Local, and Spatial variations

$$\sqrt{\sigma_{\text{global}}{}^2 + \sigma_{\text{local}}{}^2 + \sigma_{\text{spatial}}{}^2}$$

- Results for Global variations which are similar to the specifics of Local Variation
- Results for Local variations:
  - Number of devices that had no local variability specified
  - Output sigma due to Local variations
  - Number of devices with local variance contributions below the threshold value and not included in the table
  - Table with sorted device contributions

    Contribution sigma (in volts or amperes). Values below 100nV or 1pA are rounded to zero to avoid reporting numerical noise.
  - Contribution variance for ith parameter (in percent)

    $$\frac{\sigma(i)^2}{\sum_{1}^{n}\sigma(k)^2} \times 100$$

    The parameter “Threshold” applies to this column.
  - Cumulative variance through ith parameter (in percent)

$$\frac{\sum_{1}^{i} \sigma(k)^2}{\sum_{1}^{n} \sigma(k)^2} \times 100$$

The table also includes a suggestion on matched devices that should be verified independently. Devices with the same number in the column “Matched pair” are likely to be matched. Their layout should be reviewed for conformity to established matching rules.

- Results for Spatial variations are similar to the previous item, Local Variation.

Example

```
sweep point = operating point
===============================================================
output = v(out) node voltage =  1.25V  threshold = 1.000E-02
 perturbation =  2.00     interval = 1
 Output  1-sigma due to total variations = 548.32uV
 DCMatch GLOBAL VARIATION
  10 Devices had no Global Variability specified.
 Output 1-sigma due to Global variations = 29.11uV
 --------------------------------------------------------------
 Contribution    Contribution    Cumulative      Independent
 1-Sigma(V)       Variance (%)    Variance (%)    Variable
  20.11u          47.75           47.75         snps20p@vth0
  18.90u          42.18           89.94         snps20p@u0
   8.97u           9.49           99.43         snps20n@u0
   2.20u         571.20m         100.00         snps20n@vth0
 DCMatch LOCAL VARIATION
   10 Devices had no Local Variability specified
 Output 1-sigma due to Local variations = 547.74uV
 4 Devices with Contribution Variance larger than Threshold
---------------------------------------------------------------
Contribution   Contribution   Cumulative     Matched    Device
1-Sigma(V)      Variance (%)   Variance (%)   pair       Name
295.72u        29.17          29.17        1        xi82.mn1
295.49u        29.12          58.29        1        xi82.mn2
251.96u        21.17          79.47        2        xi82.mp4
247.81u        20.48          99.95        2        xi82.mp3
 10.02u        33.52m         99.98        0        r1
  6.48u        14.02m        100.00        0        xi82.mp5
  3.15u         3.31m        100.00        0        xi82.mp5
  1.72u       984.01u        100.00        0        xi82.r0
657.79n       144.32u        100.00        0        xi82.mn6
  0.            0.           100.00        0        xi82.mn8
```

Output Using .PROBE and .MEASURE Commands

Depending on the output variable specified on the `.DCMatch` command, results produced by DCMatch analysis can be saved by using `.PROBE` and `.MEASURE` commands (see syntax and examples that follow). If multiple output variables are specified, a result is produced for the last variable only. A DC sweep needs to be specified to produce these kinds of outputs; a single point sweep is sufficient.

The keywords available for saving specific results from DCMatch analysis are:

Table 64 Keyword descriptions from DCMatch Analysis

Keyword	Description
`DCM_Total`	Output sigma due to Global and Local variations.
`DCM_Global`	Output sigma due to Global variations.
`DCM_Global(par)`	Contribution of parameter “par” to output sigma due to Global variations. Here, 'par' can be an independent variable or a model parameter.
`DCM_Local`	Output sigma due to Local variations.
`DCM_Local(dev)`	Contribution of device “dev” to output sigma due to Local variations.
`DCM_Spatial`	Output sigma due to Spatial Variations.
`DCM_Spatial(var)`	Contribution of independent variable “var” to output sigma due to Spatial Variations.

Syntax for .PROBE Command for DCMatch

A `.PROBE` statement in conjunction with `.OPTION POST` creates a data file with waveforms that can be displayed in CosmosScope.

```
.PROBE DC DCM_Total
.PROBE DC DCM_Global
.PROBE DC DCM_Local
.PROBE DC DCM_Global(VariableName)
.PROBE DC DCM_Global(ModelType,ModelName,ParameterName)
.PROBE DC DCM_Local(InstanceName)
.PROBE DC DCM_Spatial
.PROBE DC DCM_Spatial(VariableName)
```

This type of output is useful for plotting the effects of mismatch as a function of bias current, temperature, or a circuit parameter.

Examples

In the first example, the contribution of the variations on vth0 (threshold) of the nmos devices with model SNPS20N is saved. In the second example, the contribution of device mn1 in subcircuit X8 is saved.

```
.Probe DCM_Global(nmos,SNPS20N,vth0)
.Probe DCM_Local(X8.mn1)
```

Syntax for .MEASURE Command

With `.MEASURE` statements, HSPICE performs measurements on the simulation results and saves them in a file with a .ms# extension.

```
.MEAS DC res1 max DCM_Total
.MEAS DC res2 max DCM_Global
.MEAS DC res3 max DCM_Local
.MEAS DC res4 max DCM_Global(VariableName)
.MEAS DC res5 max DCM_Global(ModelType,ModelName,ParameterName)
.MEAS DC res6 max DCM_Local(InstanceName)
.MEAS DC res7 find DCM_Local at=SweepValue
.MEAS DC res8 find DCM_Local(InstanceName) at=SweepValue
.MEAS DC res9 max DCM_Spatial
.MEAS DC res10 find DCM_Spatial(VariableName) at=SweepValue
```

Example

In this example, the result *systoffset* reports the systematic offset of the amplifier; the result *matchoffset* reports the variation due to local mismatch; and the result *maxoffset* reports the maximum (3-sigma) offset of the amplifier.

```
.MEAS DC systoffset avg V(inp,inn)
.MEAS DC matchoffset avg DCM_Local
.MEAS DC maxoffset param='abs(systoffset)+3.0*matchoffset'
```

DCMatch Example Netlist

An example netlist for running DCMatch analysis using a classic 7-transistor CMOS operational amplifier is available in the HSPICE demo directory as $<*installdir*>/demo/hspice/variability/opampdcm.sp

In this netlist, device sizes are set up as a function of a parameter k, which allows for investigating the effects of the Global and Local Variations as a function of device size. The following lines relate to DCMatch analysis:

```
...
.param k=2
...
mn1 net031 inn net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mn2 net18 inp net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mp3 net031 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
mp4 net18 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
...
.Variation
    .Global_Variation
         Nmos snps20N vth0=0.07 u0=10 %
         Pmos snps20P vth0=0.08 u0=8 %
    .End_Global_Variation
    .Local_Variation
         Nmos snps20N vth0='1.234e-9/
sqrt(get_E(W)*get_E(L)*get_E(M))'
         + u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
         Pmos snps20P vth0='1.234e-9/
sqrt(get_E(W)*get_E(L)*get_E(M))'
         + u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
          .Element_Variation
                R r=10 %
          .End_Element_Variation
     .End_Local_Variation
.End_Variation
...
.DCMatch v(out)
.dc k start=1 stop=4 step=0.5
...
.meas DC systoffset find V(in_pos,in_neg)  at=2
.meas DC dcmoffset find DCM_Local at=2
.meas DC maxoffset param='abs(systoffset)+3.0*dcmoffset'
.meas DC dcm_mn2 find DCM_Local(xi82.mn2) at=2
.meas DC gloffset find DCM_Global at=2
.option post
...
```

The DCMatch analysis produces four types of output from this netlist:

- table from operating point with k=2 in the output listing
- table from DC sweep for k=1 to 4 in file *opampdcm.dm0*
- waveform for output variation as a function of `k` in file opampdcm.sw0
- in file *opampdcm.sw0* for `k=2:`
  - values for systematic offset
  - output sigma due to Local Variation
  - 3-sigma amplifier offset
  - contribution of device mn2 to output sigma due to Local Variation
  - output sigma due to Global Variation

ACMatch Analysis

In ACMatch analysis, the combined effects of variations of device characteristics on the frequency response of a circuit are determined. The variation definitions are taken from the Variation Block.

The main application for ACMatch analysis is in the simulation of circuits which are sensitive to parasitics or require matching of parasitics, for characteristics such as delays and power supply rejection.

ACMatch analysis takes the changes in frequency response due to variations in DC parameters (which affect operating point and low frequency response, as well as bias dependent capacitors) and due to variations in AC parameters. Note that variation on the stimuli (voltage and current sources) can be specified on the DC and AC parameters, and both types are considered in the ACMatch analysis.

ACMatch analysis is similar to DCMatch analysis in that:

- It is efficient compared to Monte Carlo analysis because there is no sampling involved.
- Variations in component characteristics are modeled through variations in the underlying model parameters.
- Effects on a circuit's DC solution are small, and can be modeled as a linear combination of the variation in independent random variables. This is relevant for ACMatch analysis because the changes in the DC solution affect the circuit's AC characteristics.

ACMatch analysis is specified with an AC analysis, which defines the frequencies for which the circuit is analyzed; this can be a single or multiple sweep points. At least one measure or other output statement is required for this AC analysis, and subsequently ACMatch analysis, to run. The primary output of ACMatch analysis is a table with the sorted parameter and device contributions.

Input Syntax

```
.ACMatch OUTVAR <THRESHOLD=T> <FILE=string> <INTERVAL=Int>
```

Parameter	Description
OUTVAR	OutputVariable can be one or several output voltages, difference voltages or branch current through an independent voltage source. The voltage or current specifier is followed by an identifier of the AC quantity of interest: M magnitude P phase R real part I imaginary part
THRESHOLD	Report devices with a relative variance contribution above Threshold in the summary table. ▪ T=0: reports results for all devices ▪ T<0: suppresses table output; however, individual results are still available through .PROBE or .MEASURE statements. The upper limit for `T` is 1, but at least 10 devices are reported, or all if there are less than 10. Default value is 0.01.
FILE	Valid file name for the output tables. Default is `basename.am#` where “#” is the usual sequence number for HSPICE output files.
INTERVAL	Relates to the associated AC sweep. *Int* is a positive integer. A summary is printed at the first sweep point, then for each subsequent increment of *Int*, and then, if not already printed, at the final sweep point.

If more than one ACMatch analysis is specified per simulation, only the last statement is executed.

Example

```
.ACMatch VM(out) VP(out)
.AC dec 10 1k 10Meg interval=10
```

ACMatch Table Output

For each output variable and sweep point, HSPICE generates a result record in a file with default extension .am#, which includes setup information, a main result with the total variations, and a table with the sorted contributions of the relevant devices and parameters.

The individual entries include:

- frequency sweep value
- name of the output variable
- AC magnitude of this output variable
- ACMatch option values
- number of devices which had no variability specified
- output sigma values due to combined Global and Local Variations
- result for Global Variations
- contributions of parameters to Global Variations
- results for Local Variations
- contributions of devices to Local Variations

The entries in the different columns correspond to those described in the section on DCMatch Table Output.

To avoid printing unreliable results due to precision issues, phase output is not available in the table if the associated magnitude of the same variation type is less than 1uV for voltage or 1nA for current output. A warning is printed instead.

Example

```
frequency =  1.00000D+06
===============================================================
=======
output = v(out) node voltage = 1.87 V threshold = 1.000E-02
 perturbation =  2.00     interval = 1

 Output  1-sigma due to Global and local Variations = 48.68mV

 ACMatch GLOBAL VARIATION
   10 Devices had no Global Variability specified
 Output  1-sigma due to Global Variations =  46.97mV
---------------------------------------------------------------
-------
 Contribution        Contribution          Cumulative
Independent
 1Sigma(V)       Variance (%)      Variance (%)      Variable
 38.89m           68.57             68.57          snps20n@vth0
 15.78m           11.28             79.85          snps20p@vth0
 15.08m           10.31              90.16          snps20n@tox
 14.56m            9.61              99.77          snps20n@u0
  1.80m          146.80m             99.91          snps20p@u0
  1.38m           86.49m           100.00          snps20p@tox


 ACMatch LOCAL VARIATION
    6 Devices had no Local Variability specified
 Output  1-sigma due to Local Variations =    12.79mV
   7 Devices with Local Contribution Variance larger than Threshold
 --------------------------------------------------------------
-------
 Contribution     Contribution     Cumulative       Device
 1Sigma(V)       Variance (%)     Variance (%)      Name
   7.43m            33.77             33.77            xi82.mn7
   6.26m            23.97             57.73            xi82.mp4
   6.20m            23.53             81.26            xi82.mp3
   4.87m            14.49             95.75            xi82.mn8
   1.53m             1.43             97.18            xi82.mn2
   1.49m             1.36             98.54            xi82.mn1
   1.40m              1.20              99.74              r1
 563.27u          193.90m             99.93            xi82.mp5
 239.10u           34.94m           99.97          xi82.rcomp
 184.24u            20.75m            99.99            xi82.mn6
```

Output from .PROBE and .MEASURE Commands for ACMatch

The syntax of `.MEASURE` and `.PROBE` commands for ACMatch analysis is similar to the syntax for DCMatch analysis:

Syntax for .PROBE Command

A `.PROBE` statement in conjunction with `.OPTION POST` creates a data file with waveforms that can be displayed in CosmosScope.

```
.PROBE AC ACM_Total
.PROBE AC ACM_Global
.PROBE AC ACM_Local
.PROBE AC ACM_Global(VariableName)
.PROBE AC ACM_Global(ModelType,ModelName,ParameterName)
.PROBE AC ACM_Local(InstanceName)
```

Syntax for .MEASURE Command

With `.MEASURE` statements, HSPICE performs measurements on the simulation results and saves them in a file with a *.ma#* extension.

```
.MEAS AC res1 max ACM_Total
.MEAS AC res2 max ACM_Global
.MEAS AC res3 max ACM_Local
.MEAS AC res5 max ACM_Global(VariableName)
.MEAS AC res6 max ACM_Global(ModelType,ModelName,ParameterName)
.MEAS AC res7 max ACM_Local(InstanceName)
.MEAS AC res8 find ACM_Local at=SweepValue
.MEAS AC res9 find ACM_Local(InstanceName) at=SweepValue
```

Example

An example netlist for running ACMatch analysis using a classic 7-transistor CMOS operational amplifier is available in the HSPICE demo directory as $<*installdir*>/demo/hspice/variability/opampacm.sp. The following lines relate to ACMatch analysis:

```
.Variation
  .Global_Variation
      Nmos snps20N vth0=0.07 u0=10 %  tox=3 %
      Pmos snps20P vth0=0.08 u0=8 %   tox=3 %
  .End_Global_Variation
  .Local_Variation
    Nmos snps20N vth0='1.234e-9 sqrt(get_E(W)*get_E(L)*get_E(M))'
          +     u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))'
%
      +       tox='3.456e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
      Pmos snps20P vth0='1.234e-9/
sqrt(get_E(W)*get_E(L)*get_E(M))'
      +       u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
      +       tox='3.456e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
      .Element_Variation
          R r=10 %
      .End_Element_Variation
  .End_Local_Variation
.End_Variation
.ACMatch v(out)
.ac dec 1 1k 10Meg
.meas ac res1 find acm_local at=1k
```

In this example, ACMatch analysis runs at 1kHz, 10kHz, 100kHz, 1MHz, and 10MHz.

After simulation, the results in *opampacm.am0* show the contributions of devices and parameters, and their different relative importance for the different frequencies. In addition, the measurement result is printed in *opampacm.ma0*.

Application Considerations

ACMatch analysis results match those of a small signal Monte Carlo Analysis. Discrepancies arise with certain test setups, if the operating point in Monte Carlo analysis varies by a large amount. For example, the output of an amplifier might saturate at one of the supply rails for some samples, if it is configured for high gain at DC. If such conditions exist, and the amplifier is used with the same gain configuration in the real application, then they point to issues which need to be investigated with DCMatch analysis and resolved first. Otherwise, the DC gain configuration of the amplifier needs to be changed in the test setup.

Mismatch Compared to Monte Carlo Analysis

DCMatch and ACMatch analyses use calculus of probability instead of sampling. The following table shows a comparison of the two types.

Feature	**Mismatch**	**Monte Carlo**
Analysis type	DC or AC	AC, DC, Transient
Device/parameter contribution report	Yes	No
Relative run time	Fast	Slow
Accuracy	Good	Dependent on number of samples
Distributions	Normal preferred	Any
Variation result report	Global, Local, Spatial and Interconnect separate	Global, Local, Spatial and Interconnect combined

References

[1] M.Pelgrom, A.Duinmaijer, and A.Welbers: “Matching Properties of MOS Transistors”, IEEE J. Solid-State Circuits, pp. 1433-1439, May 1989.

[2] P.R.Kinget: “Device Mismatch and Tradeoffs in the Design of Analog Circuits”, IEEE J. Solid-State Circuits, pp. 1212-1224, June 2005.

23
Exploration Block

Describes the use of the Exploration Block in HSPICE.

The Exploration Block addresses the designer's need to study the behavior and sensitivities of circuits to come up with an optimum design. During this early design phase, the designer may want to explore ranges of device sizes for a given circuit topology. The Exploration Block feature allows the designer to describe a set of experiments with different geometries, without changes to the original netlist.

The Exploration Block is closely related to the Variation Block with external sampling (see Chapter 20, Analyzing Variability and Using the Variation Block).

These topics are covered in the following sections:

Exploration Block Description

The Exploration Block extracts the parameters suitable for exploration from a netlist, and thus eliminates parsing by the Exploration tool. The parameters are presented in a normalized format. This solution eliminates the exploration tool's need to rewrite the netlist with new parameter values. Using the Exploration Block, the Exploration tool returns only the updated values of the parameters which need to be changed in the course of a set of exploration simulations. HSPICE finds all the places where they need to be applied.

The Exploration Block contains a section with options, and may include a data block with instructions on how to change certain parameters on individual devices or device groups. The data block must be created outside the simulator, based on information from the simulator and considerations specific to the particular design, possibly from an optimization program.

Usage Model

To accommodate time restriction, exploration needs to be applied in an organized manner, with the smallest number of unrelated variables. A strategy which has proven useful is to take into account that integrated circuits are built with hierarchy, and that there are known relationships between devices for best results (partial and full matching). In essence, the designer's knowledge about the circuit is encapsulated in the way exploration is carried out. Experience with optimization tools has shown that this information is crucial for success, but also difficult to set up correctly.

Multiple Instantiations of the Same Cell or Subcircuit

In a typical design process, a large circuit is assembled from cells out of existing libraries. Each cell has different descriptions, for different applications: subcircuit, layout, behavioral model, etc. The circuit netlist describes how the cells are connected with other cells, and contains a description of their content. Exploration of HSPICE is restricted to the hierarchical mode only with this release.

In hierarchical mode, exploration is cell-oriented, meaning that all instantiations of a particular cell are affected the same way. With this usage model, if designers wish to explore separate instantiations of the same cell in a different manner, then they need to create new cell names, with their content definitions repeated, before exploration can start. This renaming procedure needs to be done anyhow for the final design, if the designer accepts new device sizes coming out of the exploration exercise because a basic rule of a circuit description is that multiple cell definitions with same name (and possibly different content) are not allowed.

Specifying Relationships between Devices

The following relationships between different devices can be specified in the Exploration Block to force matching:

```
device1Property1=expression of device2Property1
```

Such expressions reduce the number of variables for exploration because derived properties are processed inside of HSPICE. These relationship rules will be applied to all the devices subject to exploration. Therefore even if no change is requested from the Exploration tool, HSPICE executes these rules. So, if for example the lengths of devices opamp.mn1 and opamp.mn2 are different in the netlist, they will be the same in a simulation which contains an Exploration block with the rule that they should be the same.

A simplified syntax expresses relationships of a whole set of device properties.

Examples:

`length(opamp.mn1)=length(opamp.mn2)`	
`opamp.mp4=opamp.mp3`	`mp4` will be identical to `mp3`, in all properties
`bias.mn5=2*bias.mn6`	`mn5` consists of 2 devices in parallel, which are identical to `mn6`

Specifying Relationships between Properties

To cover appropriate scaling of secondary properties on the same device, the following relationships can be specified:

```
deviceProperty2=expression of deviceProperty1.
```

Example:

```
ad='120n*W' as='120n*W' ps='240n+W' pd='240n+W'
```

Subcircuits and Elements Supported for Exploration

The exploration feature is primarily designed for design work on integrated circuits in CMOS technology. In a first implementation, Exploration is supported for

- Independent sources: DC value
- MOS devices: W, L, M, dtemp
- Resistors: R or W, L, M, dtemp
- Capacitors: C or W, L, M, dtemp

When designing circuits, the multiplicity factor `M` is always a positive integer, but the Exploration tool can request arbitrary positive values.

To preserve relationships which have been previously defined through expressions, exploration can only be applied to parameters which are defined with numerical values.

Example1

```
     m1  out  in1 vdd vdd pch w=wp  l=100n m=3
```

Exploration can be applied to element parameters `l` and `m`, but not to `w` directly.

Example 2

```
subckt nand  in1  in2 out   wp=100n wn=50n len=100n
 m1     out  in1 vdd vdd pch w=wp  l=len
 m2     out  in2 vdd vdd pch w=wp  l=len
 m3     out  in1 mid gnd nch w=wn  l=len
 m4     mid  in2 gnd gnd nch w=wn  l=len
.ends nand
```

Exploration can be applied to subcircuit parameters `wp`, `wn` and `len`. The application envisioned here is for leaf cells with programmable layout: separate width and common length of pmos and nmos devices.

Example 3

```
.subckt onebit  in1  in2  carry-in  out  carry-out
 x1  in1   in2   7  nand
 x2  in1   7     8  nand  wp=100n  wn=100n
 x3  in2   7     9  nand  wp=300n  wn=150n
.ends onebit
subckt nand  in1  in2  out   wp=200n wn=100n
 m1  out  in1 vdd vdd pch w=wp  l=100n
 m2  out  in2 vdd vdd pch w=wp  l=100n
 m3  out  in1 mid gnd nch w=wn  l=100n
 m4  mid  in2 gnd gnd nch w=wn  l=100n
.ends nand
```

The subcircuit named `onebit` can be used for exploration because it instantiates other subcircuits using parameters with numerical values: `wn` and `wp` of `nand` gates `x2` and `x3`. The subcircuit called `nand` can be used for exploration on the default values `wn` and `wp` (exploration only affects instantiation `x1` because `x2` and `x3` parameters override the default values). The devices `m1` to `m4` can be used for exploration on their length but not on the width. This preserves the imposed relationship of equal width for `m1` and `m2`, and for `m3` and `m4`.

Exploration supports variation in temperature, in addition to element and subcircuit parameters. Encrypted sections of a netlist are not available for exploration.

Flow Using an External Exploration Tool

The design flow consists of an information extraction and export phase in HSPICE, a Definition phase for the Exploration Block outside of HSPICE, and an Exploration phase in HSPICE.

Export Phase

HSPICE creates an output file with the Export Block. It contains in the first section the names of the variables suitable for exploration, parameters, element and subcircuit parameters, along with appropriate identifiers, which include subcircuit name, device instance names, models and properties. In the second section, the corresponding values are listed.

From Example 3 on page 581:

- Subcircuit `onebit` with properties `wn` and `wp` for instantiations `x2` and `x3`
- Subcircuit `nand` with properties `wp` and `wn` (only useful if an instantiation exists where `wp` and `wn` are not defined, as in `onebit.x1`)
- `m1` to `m4` with properties `l`

In the export phase, HSPICE runs a simulation from the originally supplied netlist, ignoring any Exploration Block content other than options.

Definition Phase

Note:

You must create or adapt the external utility described in the following sections; it is not provided by HSPICE.

- An external utility reads the files created by HSPICE with the device information.
- Supplemental information to the external utility consists of technology and details of the experiment.
- The utility creates a set of experiments and formulates them as a data block, with some or all variables contained in the Export Block, and one or more sets of exploration data.
- The utility submits the netlist with Exploration Block to HSPICE.

Exploration Phase

- HSPICE applies the content of the data block, calculates the secondary parameters, and runs a set of simulations with the updated device geometries as specified in the Exploration Block. HSPICE produces measurement results and a file with all the parameter values used for each exploration simulation.
- The external utility analyzes and combines the simulation results.
- Based on the results, the utility might specify another set of experiments, with a new set of simulations, and run through these steps until some predefined goal is reached.

Refer to the Figure 99 on page 583, and notice the flow difference before and after adding exploration block.

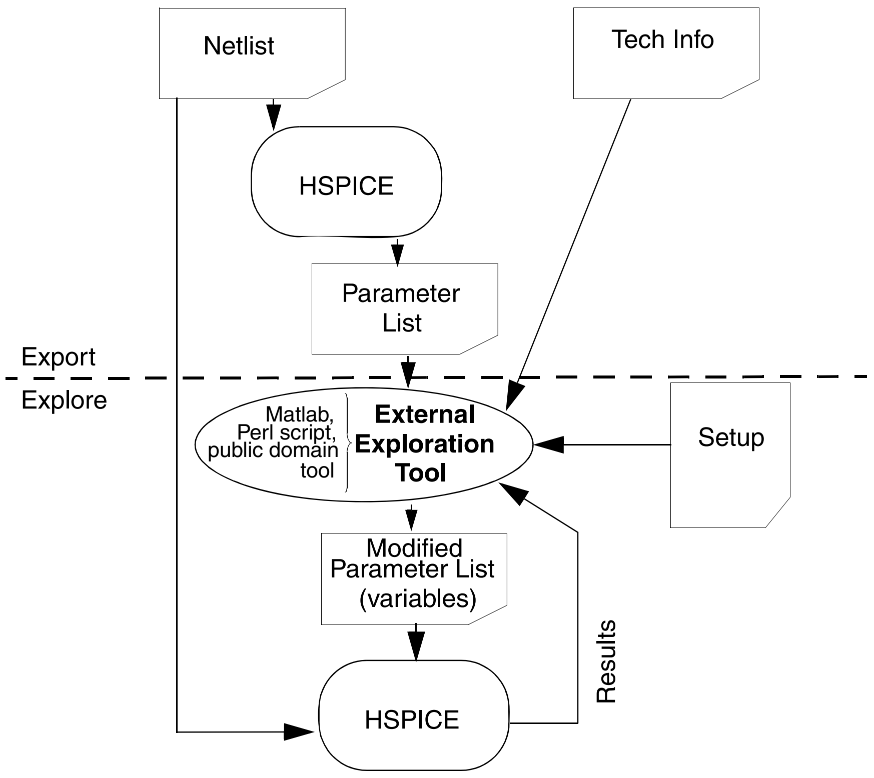

Netlist
Tech Info
HSPICE
Parameter List
Export
Explore
Matlab, Perl script, public domain tool
External Exploration Tool
Setup
Modified Parameter List (variables)
Results
HSPICE

Figure 99 Exploration Block Flow

Netlist Export

At the end of the exploration procedure, a valid netlist needs to be generated which reflects the final choices for the device sizes, in order to be able to drive a layout tool and run a successful LVS (layout versus schematic verification).

Exploration Block Syntax

Because the Exploration Block is closely related to the Variation Block, the internal structure is similar, in particular when compared with external sampling. Major differences include:

- The Variation Block specifies variation on the final device size, whereas the Exploration Block deals with the values as defined in the netlist.
- Perturbations specified in the Variation Block are applied in a flat manner, whereas those from Exploration Block apply to subcircuits, or cells.

In the following sections, a period is used as separator between a subcircuit name and an element name. For example:

`opamp.rbias` refers to the resistor **rbias** instantiated in subcircuit **opamp**.

A period is also used as separator between subcircuit names, if one subcircuit is defined within another. For example:

`opamp.bias.rbias` refers to the resistor **rbias** in subcircuit **bias**, nested within subcircuit **opamp**.

Syntax Structure

The current implementation of the Exploration Block is structured with options, parameters, relationships between devices, relationships between properties, area calculation, and data block characteristics.

```
.Design_Exploration
        Options
          Parameter Parameter_Name = value
          Parameter Parameter_Name = expression

           .Data BlockName
             Index    Name   Name, …
            …
            .EndData
.End_Design_Exploration
```

Exploration Block Options

The options include:

- `Option Explore only|Do_not_explore`
- `Option Export`

- `Option Exploration_method`
- `Option Ignore_exploration`
- `Option Secondary_param`

If you want to explore only certain cells or subcircuits use:

Option	**Description**
`Option Explore_only Subckts= SubcktList`	This command is executed hierarchically—the specified subcircuits and all instantiated subcircuits and elements underneath are affected. Thus, if an inverter with name `INV1` is placed in a digital control block called `DIGITAL` and in an analog block `ANALOG`, and `Option Explore_only Subckts = ANALOG`, then the perturbations only affect the `INV1` in the analog block. You must create a new inverter `INV1analog`, with the new device sizes.
`Option Do_not_explore Subckts= SubcktList`	Excludes listed subcircuits.

The two modes of exploration are distinguished by setting either:

Option	**Description**
`Option Export=yes`	Exports extraction data and runs one simulation with the original netlist
`Option Export=no`	(Default) Runs a simulation with Exploration data

The perturbation types are selected by setting either:

Option	**Description**
`Option Exploration_method= external Block_name= Block_name`	The Block_name is the same as the name specified in the .DATA block; HSPICE will sweep the row content with the EXCommand (see the section EXCommand Option: Export Data Block Action).
`Option Ignore_exploration= yes|no`	(Default=no) HSPICE ignores the content in the design_exploration block, when `Ignore_exploration=yes`.

Option	Description
`Option Secondary_param= yes|no`	(Default = no) If `Secondary_param= yes`, HSPICE exports the MOSFET secondary instance parameters to a **.mex* file (created when `option export=yes`), and also permits the secondary parameters to be imported as a column header in the .DATA block (`option export=no`). See the example, following this section.

Example: Option secondary_param=yes

```
.design_exploration
  option secondary_param=yes
  parameter asad = '1.5e-7'
  option exploration_method=external block_name=dat2
  *nmos snps20n design_area='(get_e(l)+1u)*(get_e(w)+0.8u)'
   pmos snps20p as = 'asad*get_e(W)' ad ='2*asad*get_E(W)'
  .data dat2
      index vdd@p k@p opamp.rcomp@r@e opamp.r0@r@e r1@r@e
      r0@r@e opamp.ccomp@c@e c0@c@e c1@c@e
      opamp.mn1@snps20n@m@e
      opamp.mn1@snps20n@ad@e
      opamp.mn2@snps20n@m@e
      opamp.mn2@snps20n@ad@e
      opamp.mp3@snps20p@m@e
      opamp.mp4@snps20p@m@e
      opamp.mp5@snps20p@l@e    opamp.mp5@snps20p@w@e
      opamp.mp5@snps20p@m@e    opamp.mn8@snps20n@l@e
      opamp.mn8@snps20n@w@e    opamp.mn8@snps20n@m@e
      opamp.mn7@snps20n@l@e    opamp.mn7@snps20n@w@e
      opamp.mn7@snps20n@m@e    opamp.mn6@snps20n@l@e
      opamp.mn6@snps20n@w@e    opamp.mn6@snps20n@m@e
      v2@v@e
      1.000   2.5000  2.0000  7.000e+03  1.000e+06  1.000e+06
      1.000e+07  9.000e-13  1.000e-03  5.000e-12
      4.0000  1.000e-08    4.0000  1.000e-08    4.0000
      4.0000  4.000e-07  1.000e-05    3.0000  6.000e-06
      3.600e-05    10.0000  6.000e-06  3.600e-05    4.0000
      6.000e-06  3.600e-05   6.0000    0.
   .enddata
.end_design_exploration
```

Notice that column header `opamp.mn1@snps20n@ad@e` can be recognized by HSPICE only if `Option Secondary_param=yes`. In the netlist for the opamp (not shown above), only the devices mn1 and mn2 have secondary element parameter AD defined.

The supported MOSFET secondary parameters are: AS, AD, PS, PD, NRD, NRS, RDC, RSC.

Parameters Section

Parameters can be defined here, which are used in subsequent definitions within the Exploration Block. The name space is separate from the netlist. Parameters specified with numerical values are exported; derived parameters are not exported, thus are not available for exploration.

Device Relationships

Relationships between element properties exist, which must be respected when changing device size. To reduce the amount of time required by the Exploration Tool to calculate these dependencies, such relationships can be defined directly in the Exploration Block.

To force a relationship between two different elements, use the syntax:

```
Element subcircuitName.ElementName parameterName=
'expression of
get_E([subcircuitName.]ElementName@parameterName)'
```

The element parameter names here include W and L for NMOS and PMOS devices. The subcircuit name on the right side of the definition is optional, if it is the same as the one on the left side.

```
Element opamp.mn1 l='get_E(mn2@l)'
Element inv4.mp1 w='2*get_E(inv2.mp1@w)'
```

These relationships are enforced on all instantiations of subcircuits opamp and inv4 (unless specifically excluded from exploration). Also, properties L of opamp.mn1 and W of inv4.mp1 are not exported, and are not available for exploration.

Secondary Element Parameters

To calculate secondary element parameters on a single device:

```
Element subcircuitName.ElementName parameterName=
  'expression of parameterName'
```

The element parameters here include AD, AS, PD, PS, NRS, and NRD on the left side and expressions of L and W of the same element on the right hand side. For example:

```
Element opamp.nm1 AD='1e-7*get_E(W)'
```

This relationship is enforced on all instantiations of subcircuit opamp (unless specifically excluded from exploration). Also, the property AD of opamp.mn1 is not exported, and it is not available for exploration.

Secondary Device Parameters

Expressions for calculating the values of secondary device parameters for all devices with a certain model can be defined. Default values for AD, AS, PD, PS, NRS and NRD are often specific for devices which share the same model, as a function of W and L.

```
ModelType ModelName instanceParameterName='expression of
parameterName'
```

For example:

```
nmos snps65n as='asad*get_e(W)' ad='asad*get_e(W)'
```

This directive means that all nmos devices subject to exploration, with model snps65n, and have AS and/or AD specified, have their source and drain areas re-calculated by this equation prior to simulation. If the secondary parameter is not specified on the device, then it is not added.

Note that HSPICE simulation results can change when such a definition is added to the Exploration Block, if the original values for AS and AD are different from the values calculated using the expression. While the secondary parameters are not exported, they are available for exploration when defined in the data block (expressions are not supported):

```
Opamp.mn1 AD=1e-12
```

Same-Circuit Parameters

To force relationship between parameters of the same subcircuit, use the syntax

```
Subckt subcircuitName parameterName='expression of
parameterName'
```

Note that this function supports only relationship within the same subcircuit.

Derived Device Properties

Derived device properties, as defined in the Exploration Block, are not exported. While specifying device relationship in a direct way is not supported, you can do this through parameter transformations.

Example 1:

```
Element opamp.mn1@L = ‘Get_E(opamp.mn2@L)’
```

Only opamp.mn2@L from the netlist is exported; here, the device property of opamp.mn1@l is not exported.

Example 2:

```
nmos nch ad='120n*get_E(W)'
+ as='120n*get_E(W)'
+ ps='240n+get_E(W)'
+ pd='240n+get_E(W)'
```

The properties AD, AS, PS, and PD are not exported.

Parameters Defined Outside the Exploration Block

The parameters defined outside the Exploration Block can be referenced using the syntax:

```
get_P(parameterName)
```

Area Measurement

One of the tradeoffs of integrated circuit design is design area. While an exact number is only available after layout, certain rules can be defined to give a good estimate.

The complete measurement consists of three steps:

1. Calculate area of each device, according to model specific expressions
2. Calculate total area of top circuit or specified subcircuit.
3. Make results available to built-in measurement processor for output.

The calculation is performed as part of the operating point for AC and TRAN, but executed for each step of a DC transfer characteristics. This allows for reporting area at a certain value of a design parameter, which affects circuit area. However, area is not recalculated if it changes during an AC or transient sweep.

Syntax for device area calculation:

```
Modeltype Modelname design_area = expression
```

Example

```
nmos nch design_area='(get_E(L)+1u)*(get_E(W)+0.8u)'
```

The measurement syntax allows for reporting the area of the whole circuit, or a subcircuit, and has the usual structure:

```
.measure analysisType measName Function
design_area(total|subcircuitName)
```

Where, `analysisType` is `DC|AC|Tran`, `Function` can be `min, max, find -at`.

For example:

```
          .measure top_area max design_area(total)
```

Rules for Area Measurement

The following rules apply for area measurement.

1. Area computation only supports R, C and MOSFET currently. There is no geometry parameter for L, so this element will be ignored in area computation.
2. Typical syntax examples:

   Compute total circuit area:

   ```
   .measure |dc|ac|tran output_name1 find design_area at=val
   ```

Hierarchical based, compute the sum area of subcircuit x1:

   ```
   .measure |dc|ac|tran output_name2 find design_area(x1) at=val
   ```

Compute area of x1.mn1

   ```
   .measure |dc|ac|tran output_name3 find design_area(x1.mn1)
       at=val
   ```

   For the equations defining area inside `.design_exploration block`, only model specific expressions are currently supported:
   `Modeltype Modelname design_area` `= expression`
3. The priority of computing one device area is
   - For R and C:

(i) Expressions defined inside `exploration_block`

(ii) `W*L*M` (`W` and `L` is defined as instance parameter)

(iii) `Wmodel*Lmodel *M` (`Wmodel` and `Lmodel` are the model geometry values)

(iv) Otherwise, their area is zero.

- For MOSFETs:

(i) Expressions defined inside `exploration_block`

(ii) `W*L*M` (`W` and `L` are defined as instance parameters)

(iii) `Wdefault*Ldefault *Mdefault` (`Wdefault` and `Ldefault` are the default geometry values for MOSFET)

(iv) `M` is the multiplier parameter.

4. If '`scale`' is defined, then `design_area = area*scale*scale`
5. Such measurement works with the `.DC |.AC |.TRAN` command, whether `.design_exploration` block is defined or not.

Processing Netlist Parameters

As shown in the section “Flow Using an External Exploration Tool,” you can have a special case of passing parameter values down one level of hierarchy. In a general case, when HSPICE finds a parameter definition with numerical value (`.param paramName=value`), it is exported with its name and value in the appropriate section. Parameters which are defined with other parameters instead of numerical values, or expressions of other parameters and numerical values, are not included in the Export file. This preserves relationships between devices, which have been set up by the designer in the original netlist.

Example of `diffpair` in netlist:

```
.subckt diffamp in1 in2 out lpair=2u  wpair=2u  mpair=4
   mn1 d1 in1 s b modelName l=lpair w=wpair m=mpair
   mn2 d2 in2 s b modelName l=lpair w=wpair m=mpair
....
.ends diffamp
```

The subcircuit `diffamp` with its parameters `lpair`, wpair and `mpair` will be in the Export file, with their local values. The devices `mn1` and `mn2` are not available for exploration.

Export File Syntax

HSPICE writes the extracted data from the circuit to a file with the extension “*.mex?*” with syntax similar to the **.mcx?* file, which lists the perturbations created from the Variation Block content. The option settings are reported first, followed by the names of all requested subcircuits and devices with their respective parameter names.

Separators are used as follows:

- A single period is used as hierarchy separator between a subcircuit and an instance or device name, and for separating one or more subcircuit names, if their definitions are nested
- The @ character is used for separating model and parameter names.
- Additionally, identifiers are appended as follows to identify the proper owner if an element and a nested subcircuit have the same name:
  - E for element parameters
  - P for global parameters, and parameters used in subcircuit definitions and instantiations

Syntax Structure

The following constructs are provided:

- For primitives (R,C, without model):

  ```
  [SubcircuitName.]InstanceName@ParamName@E
  ```

Example:

```
    Opamp.rbias@r@E
```

- For devices with model (in NMOS, PMOS)

  ```
  [SubcircuitName.]InstanceName@ModelType@ModelName@ParamName@
  E
  ```

Example:

```
Opamp.mn1@snps65n@L@E
```

- For standalone parameters:

  ```
  [SubcircuitName@]=ParamName@P
  ```

Example:

```
(.param factor) Opamp@factor@P
```

- For parameters declared on subcircuit definition line:

  `[`*`SubcircuitName@`*`]` *`ParamName@`*`P`

Example:

```
nand@wp@P
```

- For parameters appended to subcircuit instantiation:

  `[`*`SubcircuitName.`*`]` *`InstanceName@ParamName@`*`P`

Example:

```
 onebit.x2@wp@P
```

Whenever the optional *`SubcircuitName`* is not specified, the top level is assumed (implicit definition). For nested subcircuits, several *`SubcircuitName`* entries separated with a period are used.

Example Export File

```
onebit.x2@wp@P        onebit.x2@wn@P
onebit.x3@wp@P        onebit.x3@wn@P
opamp.mn1@snps65n@L@E opamp.bias.rbias@r@E
diffamp@lpair@P       diffamp@wpair@P      diffamp@mpair@P
index1 value1 value2 value3 value4 value5 etc
```

If option `export=yes` is set, then the output file contains a single data set with the original design values from the netlist. If `option export=no` (or default) then one data set is written per exploration step, with all the parameters suitable for exploration, not only the ones which were changed through an Exploration Data Block (see Exploration Block Syntax).

Executing Exploration in HSPICE

Exploration is considered a second sweep, therefore the syntax of the sweep with data block command is used:

```
.DC|.TRAN|.AC analysisDetail sweep EXCommand
```

...with `EXCommand` using the keyword `explore`, otherwise having the same syntax as `MCCommand` for Monte Carlo. The sample number is optional (and ignored if specified) when data export is requested. The following table shows the tasks performed by the simulator with the different combinations of `EXCommand`, `option Export`, and Data Block definition (valid meaning here: defined and having at least one set). Simulation with relationships means that the relationships described in the section Device Relationships are enforced.

EXCommand Option: Export Data Block Action

EXCommand	Option Export	Data Block	Action
Ignored	Yes	ignored	Export and run simulation with original netlist
`explore`	No or undefined	valid	Simulate all sets in Data Block, with relationships
`explore=5`	No or undefined	valid	Simulate sets 1 to 5 from Data Block, with relationships
`explore_list=3`	No or undefined	valid	Simulate set 3 from Data Block, with relationships
Ignored	No or undefined	not defined	Simulate with relationships enabled

Exploration Data Block Syntax

The exploration tool output back into HSPICE is a data block, which is referenced in the Exploration Block as “Exploration_Data”.

The content is as follows:

```
variableName1 variableName2 variableName3
index1 value11 value12  value13
index2 value21 value22  value23
......
```

`index` is an integer, monotonically increasing.

It is sufficient here to include only the cumulative set of parameters which change for the exploration run. Parameter names and values not specified here are left at their original values.

Exploration Block Interactions

The following rules apply per the described circumstances.

When an Exploration Block and Variation Block are both present, HSPICE can currently handle sweeps in up to two dimensions. Provided that a Variation Block and an Exploration Block are present, then the following rules apply:

- There is a `monte` command only: Exploration Block is ignored
- There is an `explore` command only: Variation Block is ignored
- There is neither `monte` nor `explore` command: Variation and Exploration blocks ignored
- When there is both a `monte` and `explore` command and there is/are:
    - Single Monte Carlo sample specified: execute Exploration Block content, according to the `explore` command and option settings.
    - Several Monte Carlo samples specified, single Exploration request: execute the requested Exploration Data set, with specified Monte Carlo samples.
    - Several Monte Carlo samples and several `explore` requests: abort, with appropriate message.
- For multiple Exploration Blocks:
    - Options are cumulative, the last definition prevails.
    - Only one named data block can be executed.

Limitations

The following feature is not implemented in this release.

Netlist Export

At the end of the exploration procedure, a valid netlist needs to be generated which reflects the final choices for the device sizes, in order to be able to drive a layout tool and run a successful LVS (layout versus schematic verification).

24
Optimization

Describes optimization in HSPICE for optimizing electrical yield.

These topics are covered in the following sections:

Overview

Optimization automatically generates model parameters and component values from a set of electrical specifications or measured data. When you define an optimization program and a circuit topology, HSPICE automatically selects the design components and model parameters to meet your DC, AC, and transient electrical specifications.

The circuit-result targets are part of the `.MEASURE` command structure and you use a `.MODEL` statement to set up the optimization.

Note:

HSPICE uses post-processing output to compute the `.MEASURE` statements. If you set `INTERP=1` to reduce the post-processing output, the measurement results might contain interpolation errors. See the HSPICE Reference Manual: Commands and Control Options for more information about these options.

HSPICE employs an incremental optimization technique. This technique solves the DC parameters first, then the AC parameters, and finally the transient parameters. A set of optimizer measurement functions not only makes transistor optimization easy, but significantly improves cell and circuit optimization.

To perform optimization, create an input netlist file that specifies:

- Minimum and maximum parameter and component limits.
- Variable parameters and components.
- An initial estimate of the selected parameter and component values.
- Circuit performance goals or a model-versus-data error function.

If you provide the input netlist file, optimization specifications, component limits, and initial guess, then the optimizer reiterates the circuit simulation until it either meets the target electrical specification, or finds an optimized solution.

For improved optimization, reduced simulation time, and increased likelihood of a convergent solution, the initial estimate of component values should produce a circuit whose specifications are near those of the original target. This reduces the number of times the optimizer reselects component values and resimulates the circuit.

Optimization Control

How much time an optimization requires before it completes depends on:

- Number of iterations allowed.
- Relative input tolerance.
- Output tolerance.
- Gradient tolerance.

The default values are satisfactory for most applications. Generally, 10 to 30 iterations are sufficient to obtain accurate optimizations.

Simulation Accuracy

For optimization, set the simulator with tighter convergence options than normal. The following are suggested options:

For DC MOS model optimizations:

```
absmos=1e-8
relmos=1e-5
relv=1e-4
```

For DC JFET, BJT, and diode model optimizations:

```
absi=1e-10
reli=1e-5
relv=1e-4
```

For transient optimizations:

```
relv=1e-4
relvar=1e-2
```

Curve Fit Optimization

Use optimization to curve-fit DC, AC, or transient data:

1. Use the `.DATA` statement to store the numeric data for curves in the data file as in-line data.
2. Use the `.PARAM xxx=OPTxxx` statement to specify the variable circuit components and the parameter values for the netlist.

   The optimization analysis statements use the `DATA` keyword to call the in-line data.
3. Use the `.MEASURE` statement to compare the simulation result to the values in the data file

   In this statement, use the `ERR1` keyword to control the comparison.

If the calculated value is not within the error tolerances specified in the optimization model, HSPICE selects a new set of component values. HSPICE then simulates the circuit again and repeats this process until it obtains the closest fit to the curve or until the set of error tolerances is satisfied.

Goal Optimization

Goal optimization differs from curve-fit optimization because it usually optimizes only a particular electrical specification, such as rise time or power dissipation.

To specify goal optimizations, do the following:

1. Use the GOAL keyword.
2. In the `.MEASURE` statement, select a relational operator where `GOAL` is the target electrical specification to measure.

For example, you can choose a relational operator in multiple-constraint optimizations when the absolute accuracy of some criteria is less important than for others.

Timing Analysis

To analyze circuit timing violation, HSPICE uses a binary search algorithm. This algorithm generate a set of operational parameters, which produce a failure in the required behavior of the circuit. When a circuit timing failure occurs, you can identify a timing constraint, which can lead to a design guideline. Typical types of timing constraint violations include:

- Data setup time before a clock.
- Data hold time after a clock.
- Minimum pulse width required to allow a signal to propagate to the output.
- Maximum toggle frequency of the component(s).

Bisection Optimization finds the value of an input variable (target value) associated with a goal value for an output variable. To relate them, you can use various types of input and output variables, such as voltage, current, delay time, or gain, and a transfer function.

You can use the bisection feature in either a pass-fail mode or a bisection mode. In each case, the process is largely the same.

Optimization Statements

Optimization requires several statements:

- `.MODEL modname OPT ...`
- `.PARAM parameter=OPTxxx (init, min, max)`

  Use `.PARAM` statements to define initial, lower, and upper bounds.
- A `.DC`, `.AC`, or `.TRAN` analysis statement, with:

  ```
  MODEL=modname
  OPTIMIZE=OPTxxx
  RESULTS=measurename
  ```

  Use `.PRINT` and `.probe` output statements, with the `.DC`, `.AC`, or `.TRAN` analysis statements.

Only use an analysis statement with the `OPTIMIZE` keyword for optimization. To generate output for the optimized circuit, specify another analysis statement (`.DC`, `.AC`, or `.TRAN`), and the output statements.

- `.MEASURE measurename ... <GOAL=| < | > val>`

  Include a space on either side of the relational operator:

  `= <space> < <space> > <space>`

  For a description of the types of `.MEASURE` statements that you can use in optimization, see Chapter 11, Simulation Output

  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.

Optimizing Analysis (.DC, .TRAN, .AC)

The following syntax optimizes HSPICE simulation for a DC, AC, and Transient analysis.

```
.DC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.AC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.TRAN <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

Argument	Description
DATA	Specifies an in-line file of parameter data to use in optimization.
MODEL	The optimization reference name, which you also specify in the .MODEL optimization statement.
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.

Argument	Description
RESULTS	The measurement reference name. You also specify this name in the .MEASURE optimization statement. RESULTS passes the analysis data to the .MEASURE optimization statement.

Optimization Examples

This section contains examples of HSPICE optimizations (for HSPICE RF optimization, see “Optimization” in the *HSPICE RF Manual*):

- MOS Level 3 Model DC Optimization
- MOS Level 13 Model DC Optimization
- RC Network Optimization
- Optimizing CMOS Tristate Buffer
- BJT S-parameters Optimization
- BJT Model DC Optimization
- Optimizing GaAsFET Model DC
- Optimizing MOS Op-amp

MOS Level 3 Model DC Optimization

This example shows an optimization of I-V data to a Level 3 MOS model. The data consists of gate curves (ids versus vgs) and drain curves (ids versus vds).

This example optimizes the Level 3 parameters:

- `VTO`
- `GAMMA`
- `UO`
- `VMAX`
- `THETA`
- `KAPPA`

After optimization, HSPICE compares the model to the data for the gate, and then to the drain curves. `.OPTION POST` generates AvanWaves files for comparing the model to the data.

Input Netlist File for Level 3 Model DC Optimization

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/devopt/ml3opt.sp`

The HSPICE input netlist shows:

- Using `.OPTION` to tighten tolerances, which increases the accuracy of the simulation. Use this method for I-V optimization.
- `.MODEL optmod OPT itropt=30` limits the number of iterations to 30.
- The circuit is one transistor. The `VDS`, `VGS`, and `VBS` parameter names, match names used in the data statements.
- `.PARAM` statements specify `XL`, `XW`, `TOX`, and `RSH` process variation parameters, as constants. The device characterizes these measured parameters.
- The model references parameters. In `GAMMA= GAMMA`, the left side is a Level 3 model parameter name; the right side is a `.PARAM` parameter name.
- The long `.PARAM` statement specifies initial, min and max values for the optimized parameters. Optimization initializes UO at 480, and maintains it within the range 400 to 1000.
- The first `.DC` statement indicates that:
  - Data is in the in-line `.DATA` all block, which contains merged gate and drain curve data.
  - Parameters that you declared as `OPT1` (in this example, all optimized parameters) are optimized.
  - The `COMP1` error function matches the name of a `.MEASURE` statement.
  - The `OPTMOD` model sets the iteration limit.
- The `.MEASURE` statement specifies least-squares relative error. HSPICE divides the difference between data par(ids) and model i(m1) by the larger of:
  - the absolute value of `par(ids)`, or
  - `minval=10e-6`

If you use `minval`, low current data does not dominate the error.

- Use the remaining `.DC` and `.PRINT` statements for print-back after optimization. You can place them anywhere in the netlist input file because parsing the file correctly assigns them.
- The `.PARAM VDS=0 VGS=0 VBS=0 IDS=0` statements declare these data column names as parameters.

  The `.DATA` statements contain data for `IDS` versus `VDS`, `VGS`, and `VBS`. Select data that matches the model parameters to optimize.

Example

To optimize GAMMA, use data with back bias (VBS= -2 in this case). To optimize KAPPA, the saturation region must contain data. In this example, the all data set contains:

- Gate curves: vds=0.1 vbs=0,-2 vgs=1 to 5 in steps of 0.25.
- Drain curves: vbs=0 vgs=2,3,4,5 vds=0.25 to 5 in steps of 0.25.

Figure 100 shows the results.

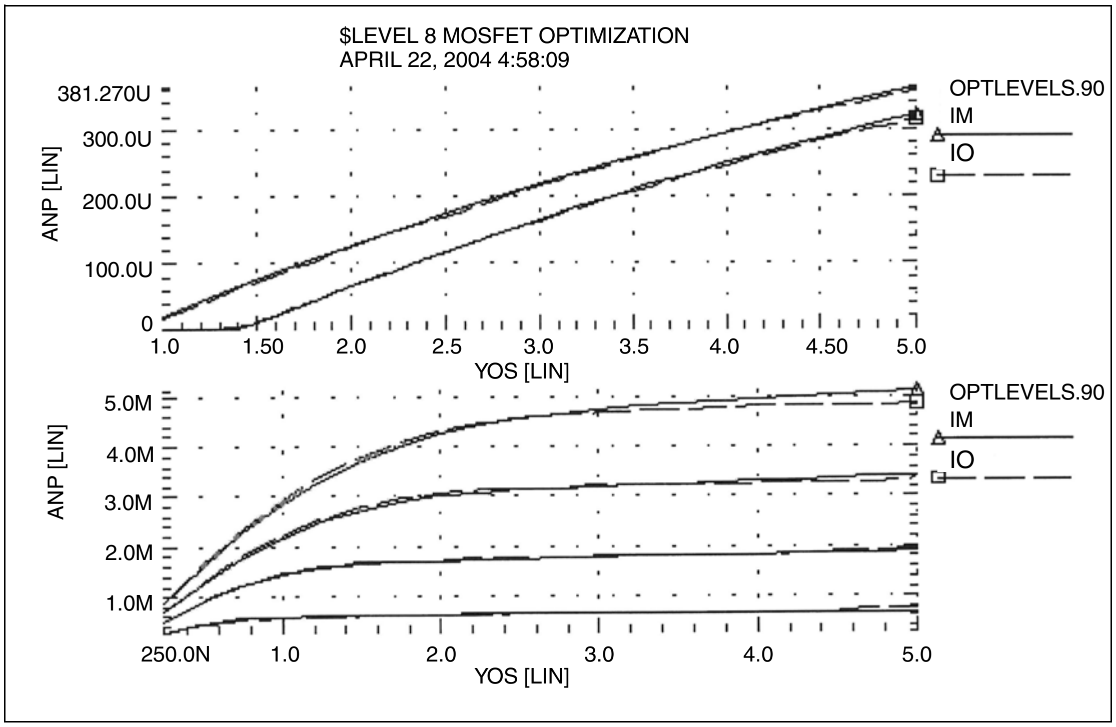


Figure 100 Level 3 MOSFET Optimization

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 101 on page 605 shows the results.

DC Optimization Input Netlist File for Level 13 Model

You can find the sample netlist for this example in the following directory: $*installdir*/demo/hspice/mos/ml13opt.sp.

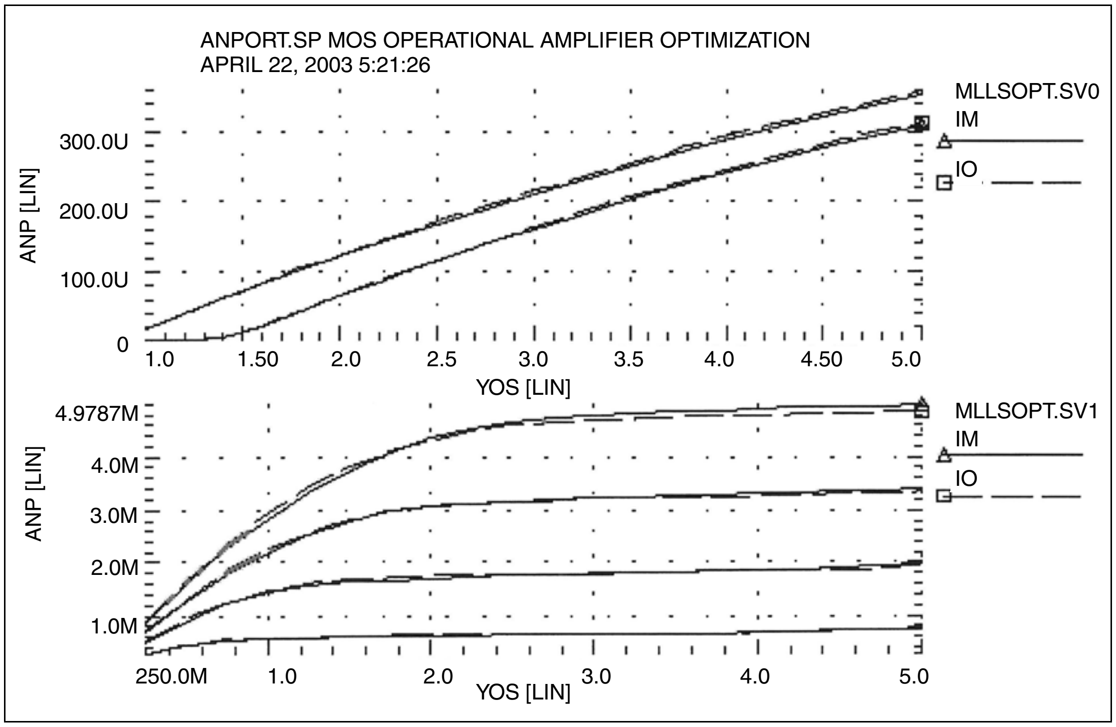


Figure 101 Level 13 MOSFET Optimization

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.
- $\mathrm{OPT}_{\mathrm{RC}}$ identifies `RX` and `CX` as optimization parameters, and sets their starting, minimum, and maximum values.

Network optimization uses these HSPICE features:

- Measure voltages and report times that are subject to a goal.
- Measure device power dissipation subject to a goal.
- Measure statements replace the tabular or plot output.
- Parameters used as element values.
- Parameter optimizing function.
- Transient analysis with SWEEP optimizing.

Optimization Results

```
RESIDUAL SUM OF SQUARES       = 4.291583E-16
NORM OF THE GRADIENT          = 5.083346E-04
MARQUARDT SCALING PARAMETER   = 2.297208E-04
NO. OF FUNCTION EVALUATIONS   =  20
NO. OF ITERATIONS  =  9
```

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.

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:

$$G_j = \sum_{i=1}^{ne} E_i \cdot (\Delta E_i / \Delta P_j)$$

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

```
.param rx=  7.4823   $   55.6965  5.7945m
.param cx=133.9934m  $   44.3035  5.1872m
```

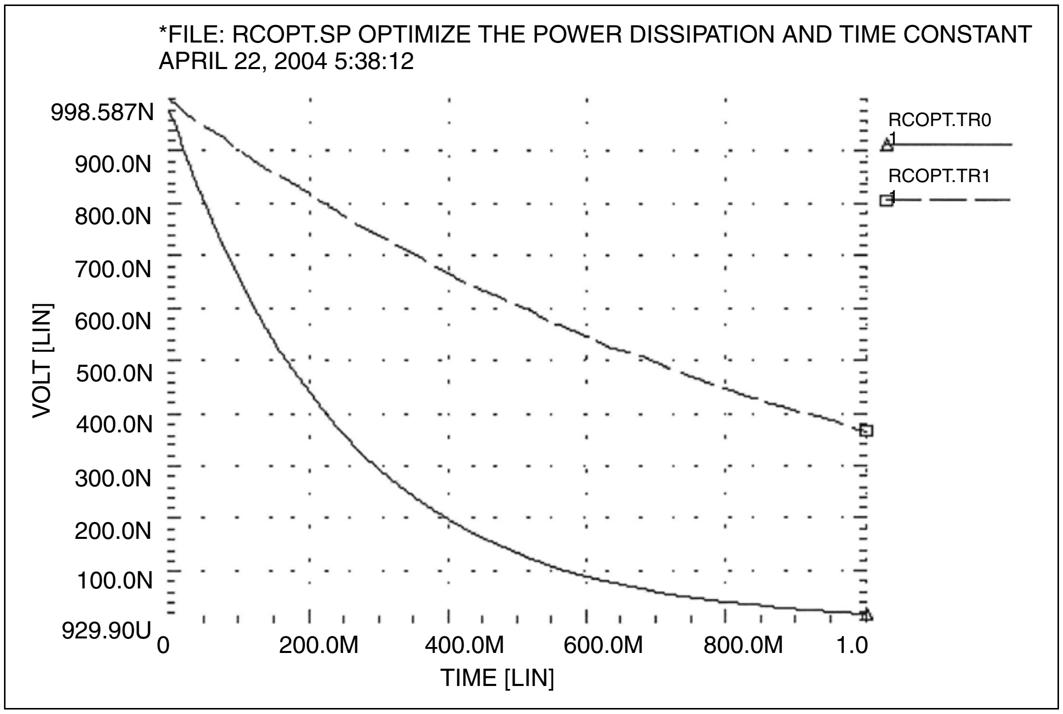


Figure 102 Power Dissipation and Time Constant (VOLT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result

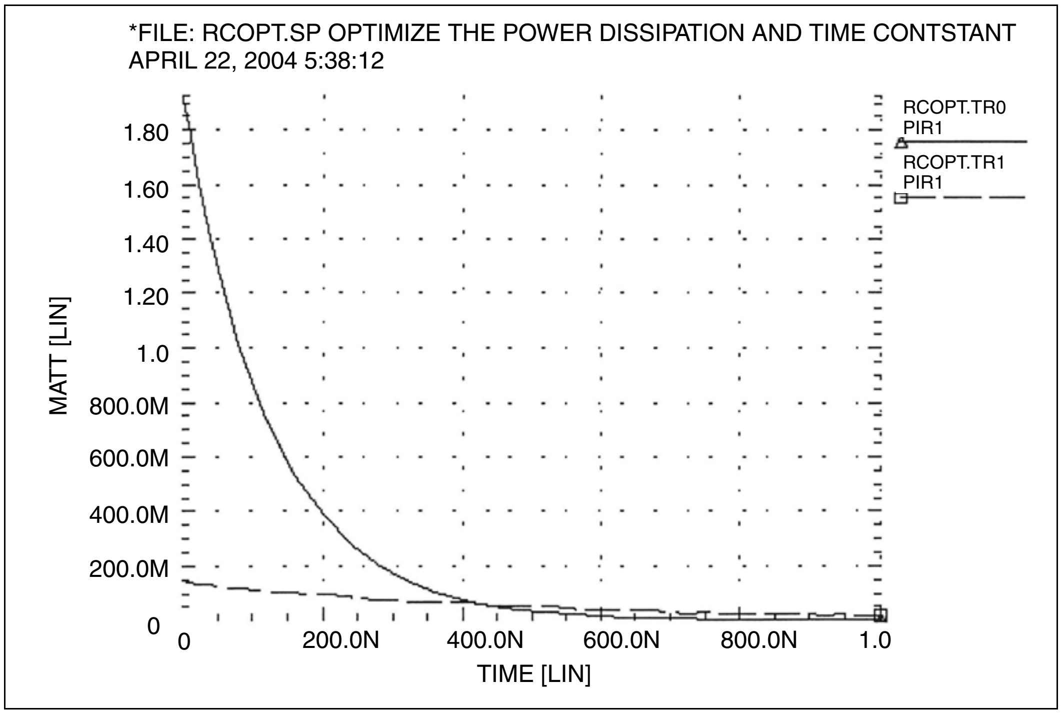


Figure 103 Power Dissipation and Time Constant (WATT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result

Optimizing CMOS Tristate Buffer

The example circuit is an inverting CMOS tristate buffer. The design targets are:

- Rising edge delay of 5 ns (input 50% voltage to output 50% voltage).
- Falling edge delay of 5 ns (input 50% voltage to output 50% voltage).
- RMS power dissipation should be as low as possible.
- Output load consists of:
  - pad capacitance
  - leadframe inductance
  - 50 pF capacitive load

The HSPICE strategy is:

- Simultaneously optimize both the rising and falling delay buffer.
- Set up the internal power supplies, and the tristate enable as global nodes.
- Optimize all device widths except:
    - Initial inverter (assumed to be standard size).
    - Tristate inverter and part of the tristate control (optimizing is not sensitive to this path).
- Perform an initial transient analysis for plotting purposes. Then optimize and perform a final transient analysis for plotting.
- To use a weighted RMS power measure, specify unrealistically-low power goals. Then use MINVAL to attenuate the error.

Input Netlist File to Optimize a CMOS Tristate Buffer

You can find the sample netlist for this example in the following directory:
$installdir/demo/hspice/apps/trist_buf_opt.sp

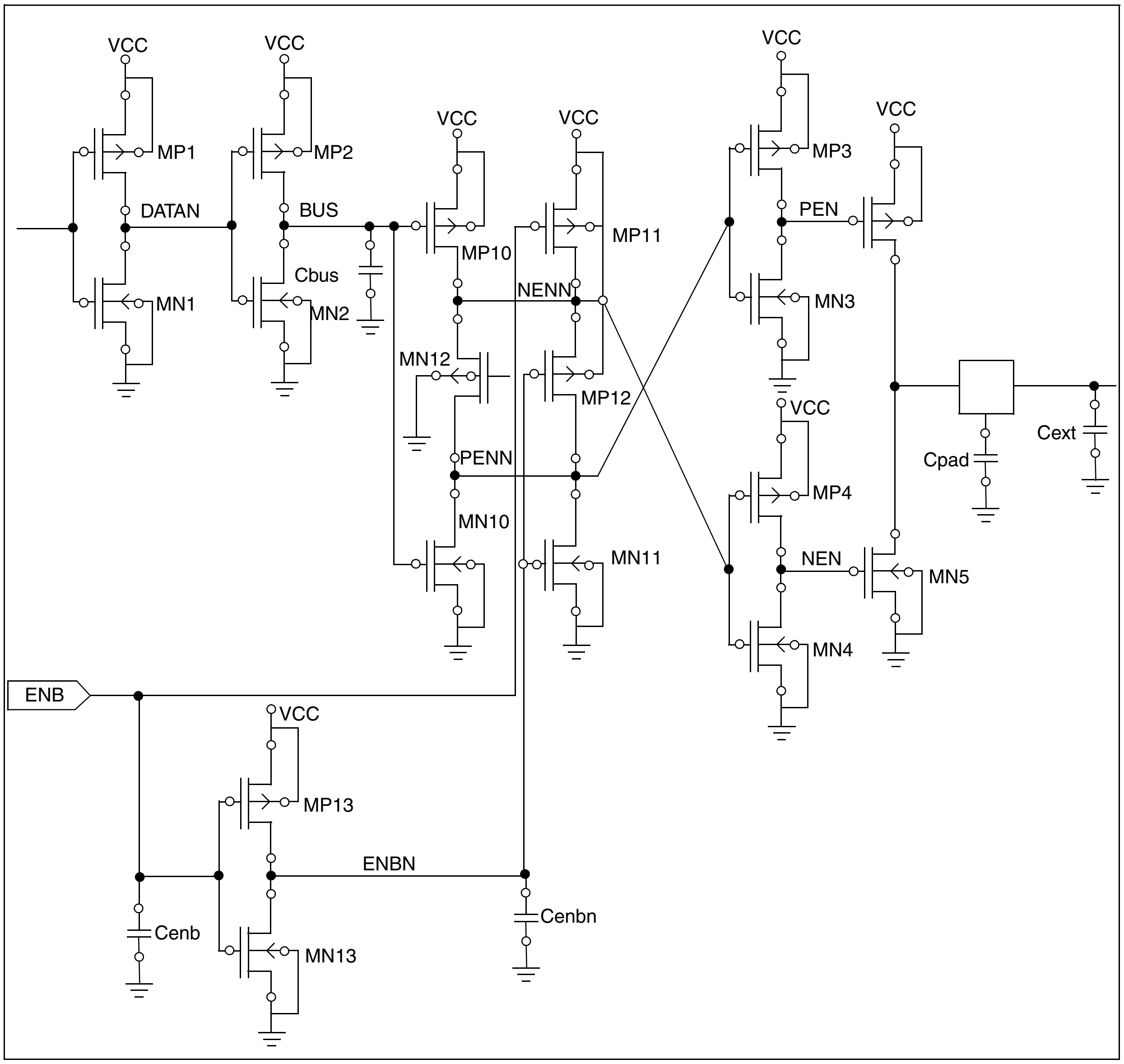


Figure 104 Tristate Buffer Optimization Circuit

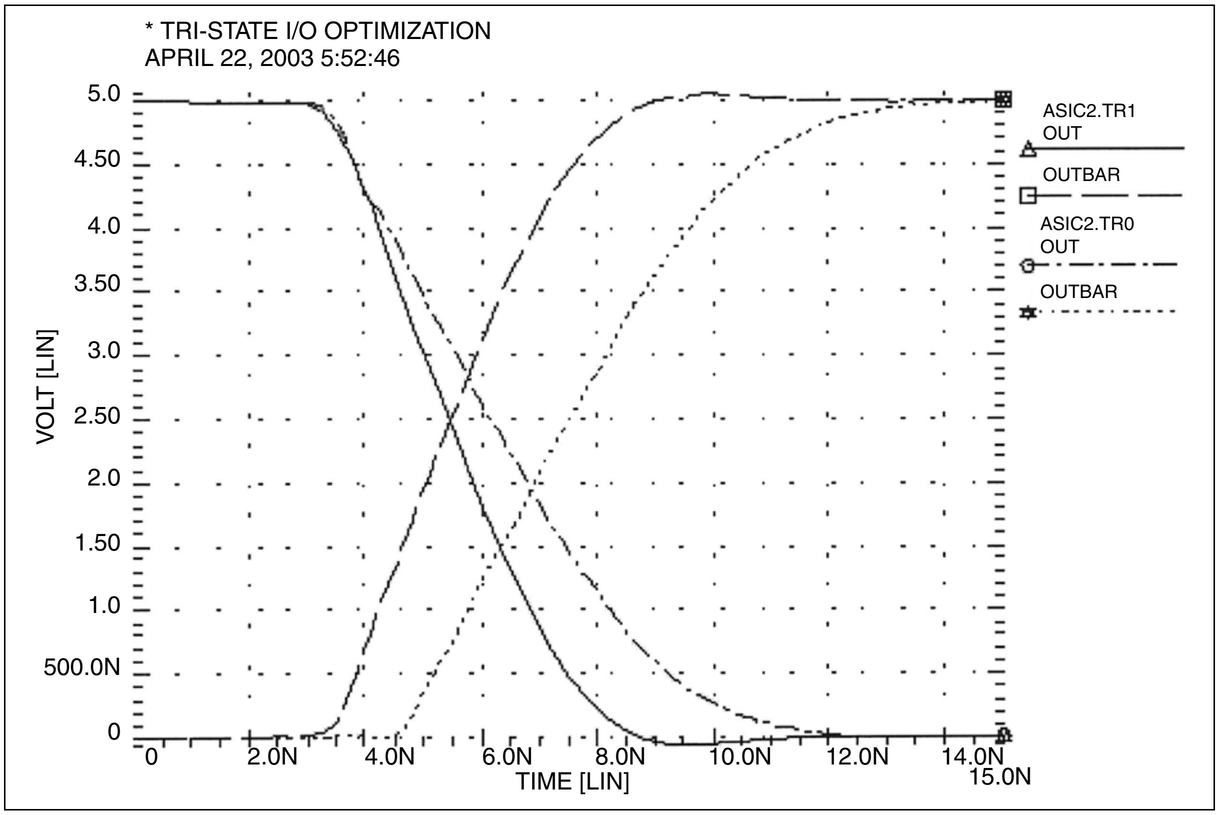


Figure 105 Tristate Input/Output Optimization ACIC2B.TR0 = Before Optimization, ACIC2B.TR1=Optimized Result

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

BJT Equivalent Circuit Input

Use the bjtopt.sp netlist file located in your $<installdir>/demo/hspice/devopt directory for optimizing BJT S-parameters.

Optimization Results

```
RESIDUAL SUM OF SQUARES    =5.142639e-02
NORM OF THE GRADIENT       =6.068882e-02
MARQUARDT SCALING PARAMETER=0.340303
CO. OF FUNCTION EVALUATIONS=170
NO. OF ITERATIONS          =35
```

The maximum number of iterations (25) was exceeded. However, the results probably are accurate. Increase ITROPT accordingly.

```
Optimized Parameters OPT1- Final Values
***OPTIMIZED PARAMETERS OPT1 SENS  %NORM-SEN
.PARAM  LBB = 1.5834N  $ 27.3566X  2.4368
.PARAM  LCC = 2.1334N  $ 12.5835X  1.5138
.PARAM  LEE  =723.0995P  $254.2312X 12.3262
.PARAM  TF  =12.7611P  $  7.4344G 10.0532
.PARAM  CBE  =620.5195F  $ 23.0855G  1.5300
.PARAM  CBC = 1.0263P  $346.0167G 44.5016
.PARAM  RB   =  2.0582   $ 12.8257M  2.3084
.PARAM  RE   =869.8714M  $ 66.8123M  4.5597
.PARAM  RC  =54.2262   $  3.1427M 20.7359
.PARAM  IS  =99.9900P  $  3.6533X 34.4463M
```

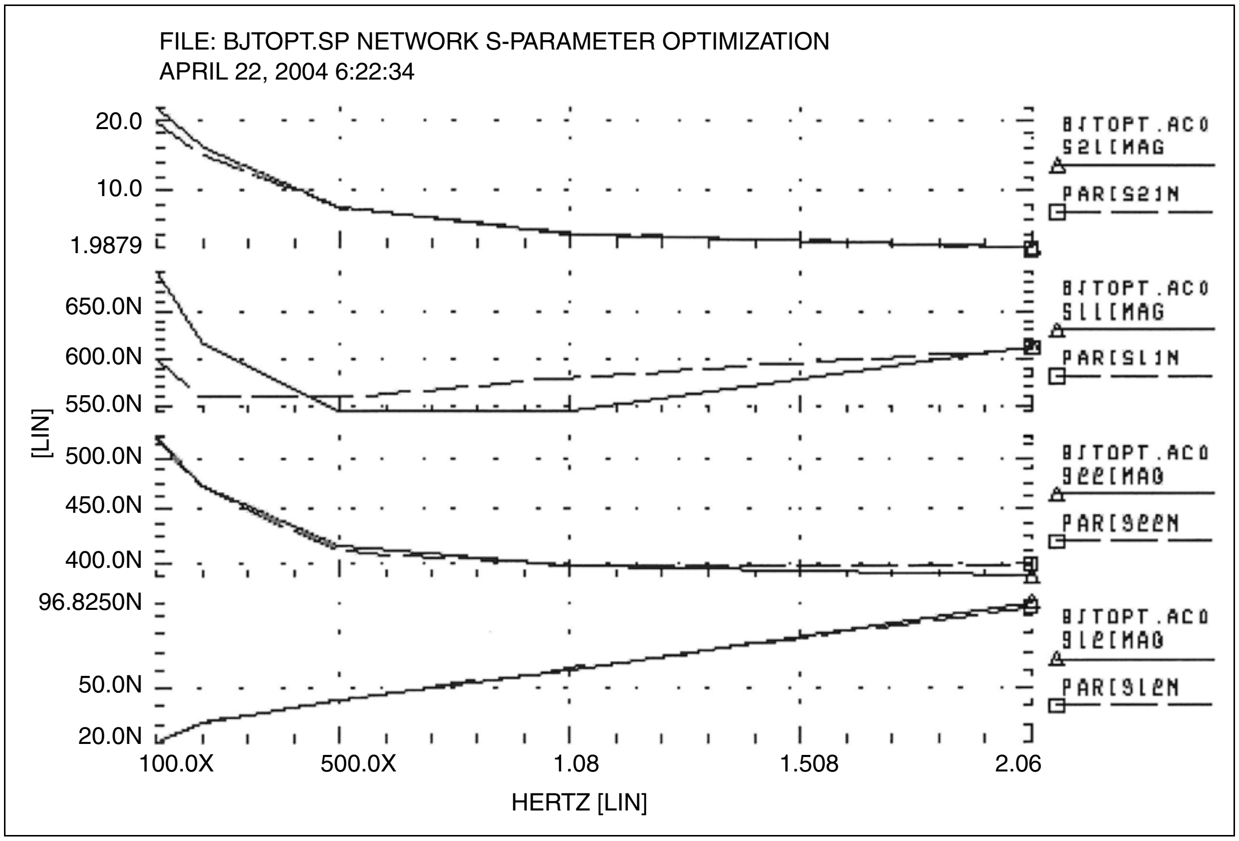


Figure 106 BJT-S Parameter Optimization

BJT Model DC Optimization

The goal is to match forward and reverse Gummel plots obtained from a HP4145 semiconductor analyzer by using the HSPICE `LEVEL=1` Gummel-Poon BJT model. Because Gummel plots are at low base currents, HSPICE does not optimize the base resistance. HSPICE also does not optimize forward and reverse Early voltages (VAF and VAR) because simulation does not measure VCE data.

The key feature in this optimization is incremental optimization.

1. HSPICE first optimizes the forward-Gummel data points.
2. HSPICE updates forward-optimized parameters into the model.

After updating, you cannot change these parameters.

3. HSPICE next optimizes the reverse-Gummel data points.

BJT Model DC Optimization Input Netlist File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/devopt/opt_bjt.sp`

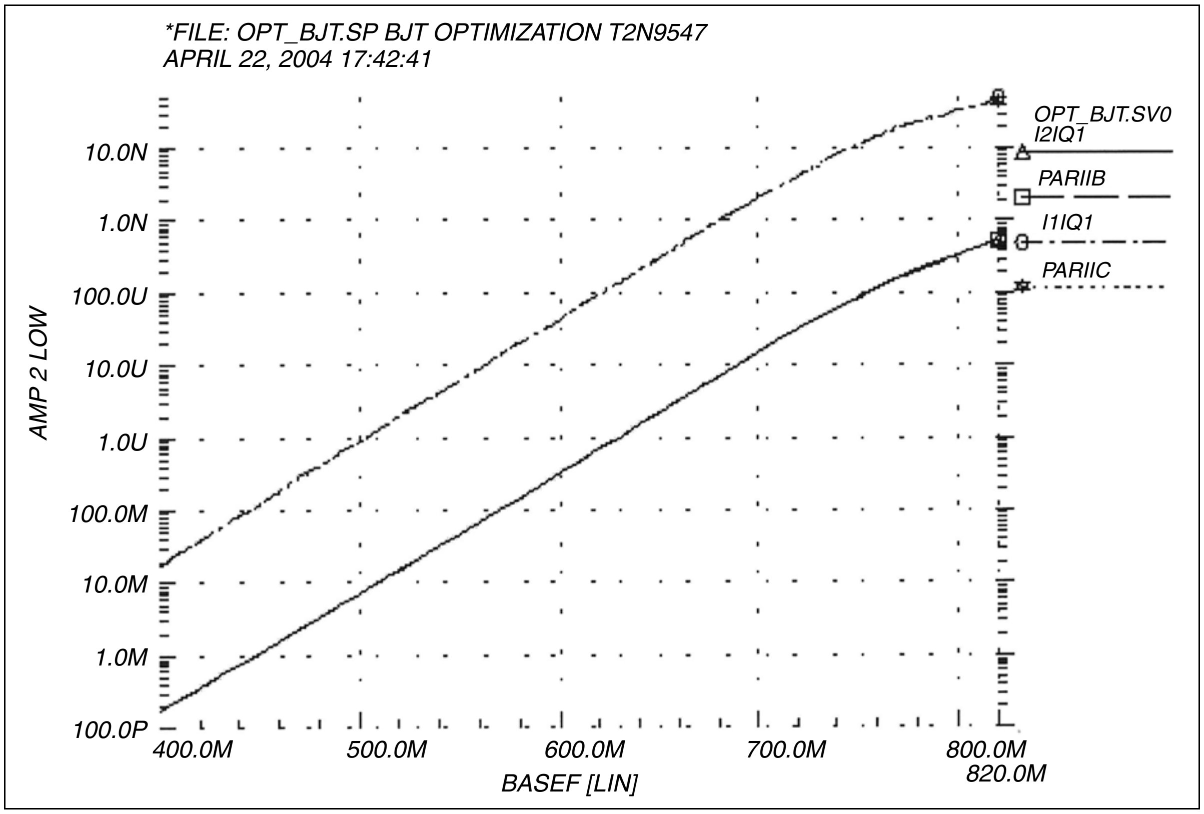


Figure 107 BJT Optimization Forward Gummel Plots

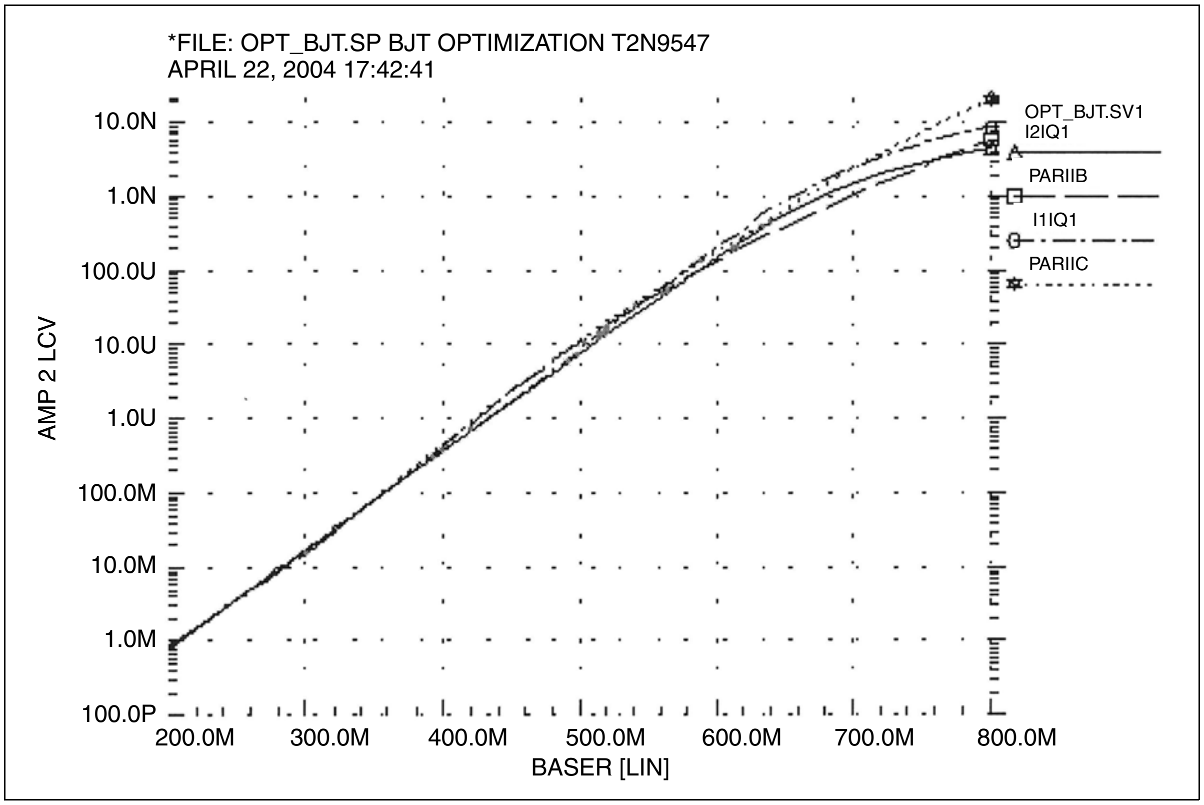


Figure 108 BJT Optimization Reverse Gummel Plots

Optimizing GaAsFET Model DC

This example circuit is a high-performance, GaAsFET transistor. The design target is to match HP4145 DC measured data to the HSPICE LEVEL=3 JFET model.

The HSPICE strategy is:

- `.MEASURE IDSERR` is an `ERR1` type function. It provides linear attenuation of the error results starting at 20 mA. This function ignores all currents below 1 mA. The high-current fit is the most important for this model.
- The `OPT1` function simultaneously optimizes all DC parameters.

- The `.DATA` statement merges TD1.dat and TD2.dat data files.
- The graph plot model sets the `MONO=1` parameter to remove the retrace lines from the family of curves.

GaAsFET Model DC Optimization Input Netlist File

You can find the sample netlist for this example in the following directory: $installdir/demo/hspice/devopt/jopt.sp

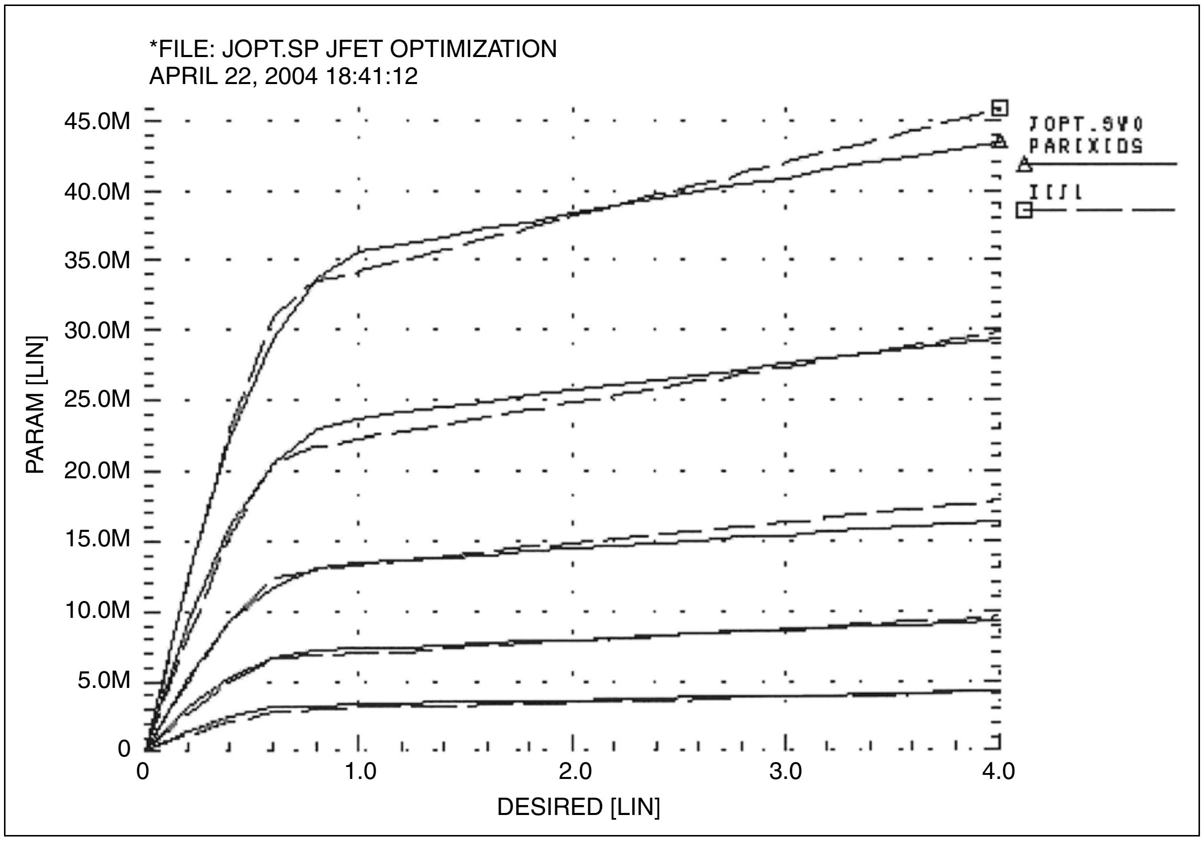


Figure 109 JFET Optimization

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.

Example: MOS Op-amp Optimization Input Netlist File

You can find the sample netlist for this example in the following directory: $installdir/demo/hspice/ciropt/ampopt.sp

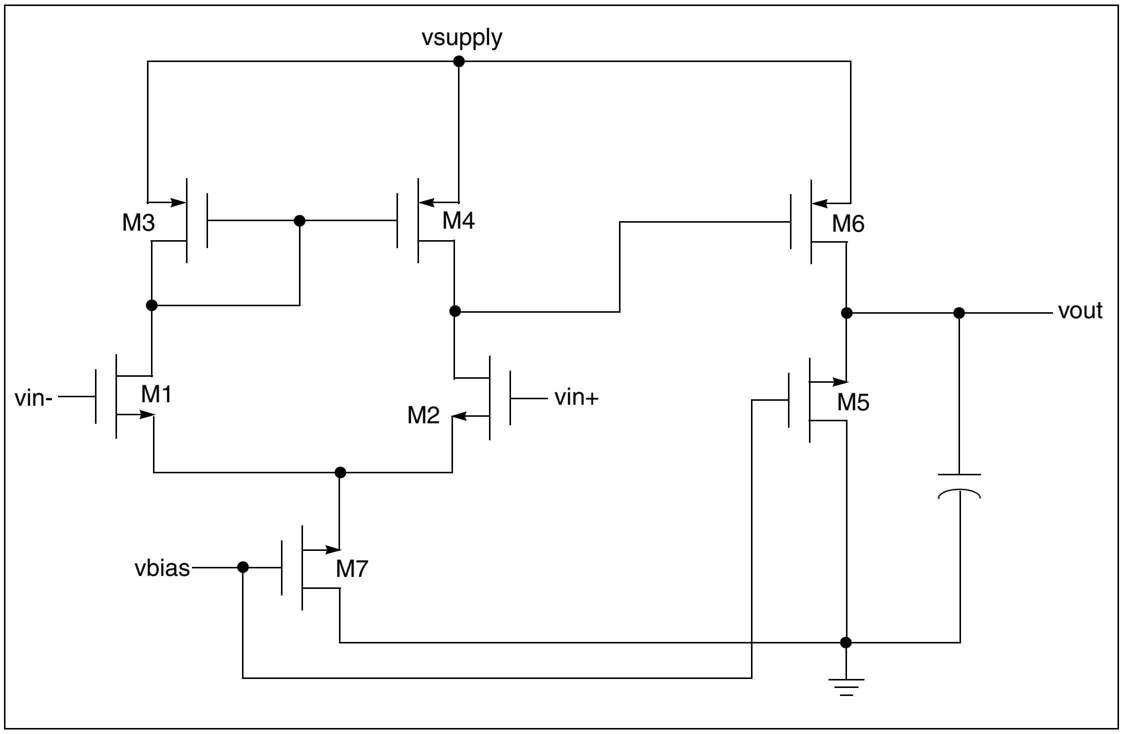


Figure 110 CMOS Op-amp

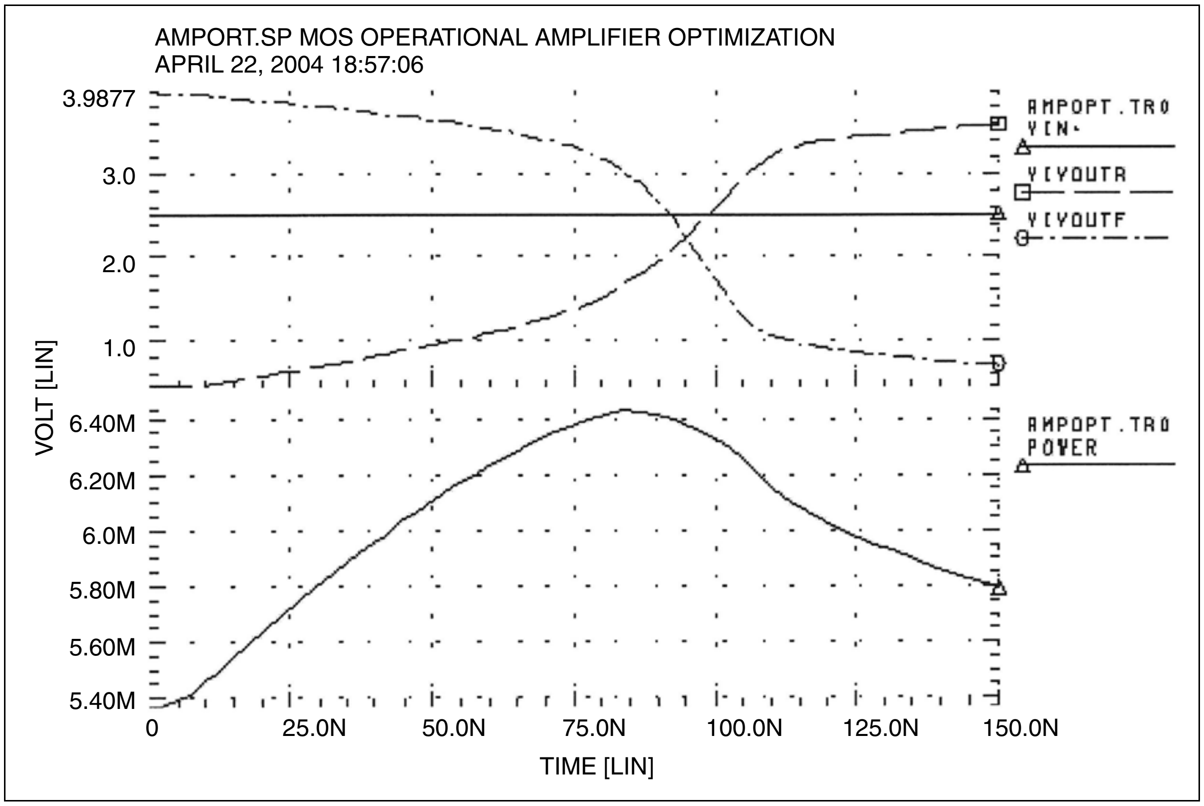


Figure 111 Operational Amplifier Optimization

25

RC Reduction and Post-Layout Simulation

Describes RC network reduction in HSPICE and post-layout simulation containing a large number of parasitic elements.

In HSPICE, the post-layout simulation is similar to pre-layout simulation. You can do the post-layout simulation with DSPF only if it is a fully extracted netlist with instances. The DSPF file can be included to the pre-layout netlist. You will need to replace the ideal .SUBCKT blocks from your original netlist with .SUBCKT blocks containing the extracted parasitics. Remember to verify that the port order in the extracted .SUBCKT blocks match the port order in the ideal netlist.

If your extracted netlist is not too large (approximately 100,000 elements or fewer), then HSPICE can give you very results. Otherwise, you can also employ an RC reduction.

These topics are covered in the following sections:

- Linear Acceleration
- Linear Acceleration Control Options Summary

Linear Acceleration

By using the `SIM_LA` option, you can accelerate the simulation of circuits that include large linear RC networks. To achieve this acceleration, HSPICE linearly reduces all matrices that represent RC networks. The result is a smaller matrix that maintains the original port behavior, yet achieves significant savings in memory and computation. Thus, the `SIM_LA` option is ideal for circuits with large numbers of resistors and capacitors, such as clock trees, power lines, or substrate networks.

In general, the RC elements are separated into their own network. The nodes shared by both main circuit elements (including `.PRINT`, `.PROBE`, and

`.MEASURE` statements), and RC elements are the port nodes of the RC network. All other RC nodes are internal nodes. The currents flowing into the port nodes are a frequency-dependent function of the voltages at those nodes. The multiport admittance of a network represents this relationship.

- The `SIM_LA` option formulates matrices to represent multiport admittance.
- Then, to eliminate as many internal nodes as possible, it reduces the size of these matrices, while preserving the admittance, otherwise known as port node behavior.
- The amount of reduction depends on the *f*0 upper frequency, the threshold frequency where `SIM_LA` preserves the admittance. This is shown graphically in Figure 112.

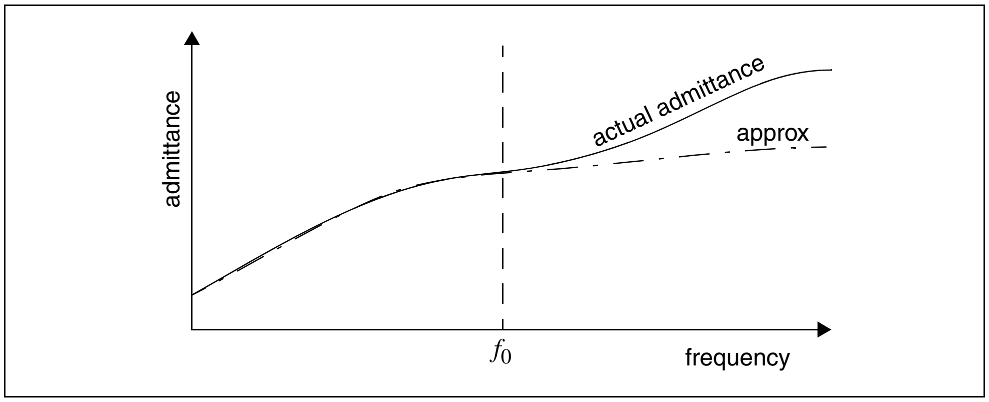


Figure 112 Multiport Admittance vs. Frequency

The `SIM_LA` option is very effective for post-layout simulation because of the volume of parasitics. For frequencies below *f*0, the *approx* signal matches that of the original admittance. Above f_0, the two waveforms diverge, but the higher frequencies are not of interest. The lower the *f*0 frequency, the greater the amount of reduction.

For the syntax and description of this control option, see .OPTION SIM_LA in the *HSPICE Reference Manual: Commands and Control Options.*

You can choose one of two algorithms, explained in the following sections:

- PACT Algorithm
- PI Algorithm

PACT Algorithm

The `PACT` (Pole Analysis via Congruence Transforms) algorithm reduces the RC networks in a well-conditioned manner, while preserving network stability.

- The transform preserves the first two moments of admittance at DC (slope and offset), so that DC behavior is correct (see Figure 113).
- The algorithm preserves enough low-frequency poles from the original network to maintain the circuit behavior up to a specified maximum frequency *f*0, within the specified tolerance.

This approach is more accurate between these two algorithms, and is the default.

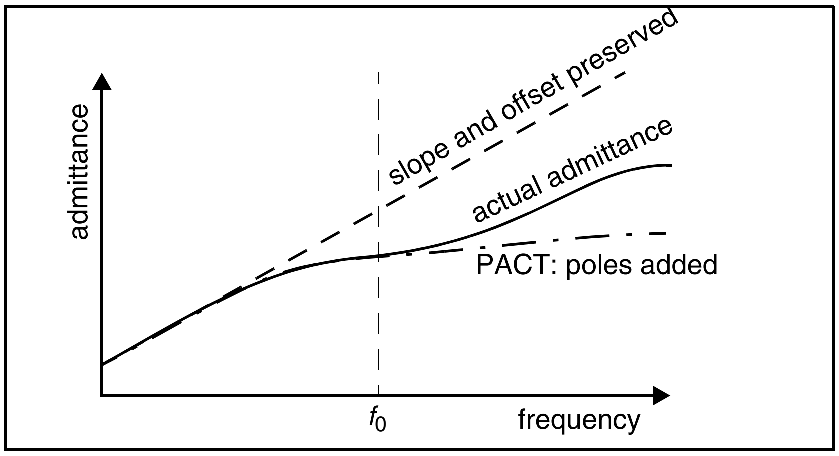


Figure 113 PACT Algorithm

PI Algorithm

This algorithm creates a *pi* model of the RC network.

- For a two-port, the *pi* model reduced network consists of:
  - a resistor connecting the two ports, and
  - a capacitor connecting each port to ground

The result resembles the Greek letter pi.

- For a general multiport, `SIM_LA` preserves the DC admittance between the ports, and the total capacitance that connects the ports to ground. All floating capacitances are lumped to ground.

Linear Acceleration Control Options Summary

In addition to `SIM_LA`, other options are available to control the maximum resistance and minimum capacitance values to preserve, and to limit the operating parameters of the PACT algorithm. Table 65 contains a summary of these control options. For their syntax and descriptions, see the respective option descriptions in Chapter 3, HSPICE and RF Netlist Simulation Control Options, in the *HSPICE Reference Manual: Command and Control Options*.

Table 65 PACT Options

Syntax	Description
.OPTION SIM_LA=PACT | PI	Activates linear matrix reduction and selects between two methods.
.OPTION LA_FREQ=*value*	Upper frequency where you need accuracy preserved. *value* is the upper frequency for which the PACT algorithm preserves accuracy. If *value* is 0, PACT drops all capacitors because only DC is of interest. The maximum frequency required for accurate reduction depends on both the technology of the circuit and the time scale of interest. In general, the faster the circuit, the higher the maximum frequency. The default is 1GHz.
.OPTION LA_MAXR=*value*	Maximum resistance for linear matrix reduction. *value* is the maximum resistance preserved in the reduction. SIM_LA assumes that any resistor greater than *value* has an infinite resistance, and drops the resistor after reduction finishes. The default is 1e15 ohms.
.OPTION LA_MINC=*value*	Minimum capacitance for linear matrix reduction. *value* is the minimum capacitance preserved in the reduction. After reduction completes, SIM_LA lumps any capacitor smaller than *value* to ground. The default is 1e-16 farads.

Table 65 PACT Options (Continued)

Syntax	**Description**
.OPTION LA_TIME=*value*	Minimum time for which accuracy must be preserved. *value* is the minimum switching time for which the PACT algorithm preserves accuracy. HSPICE does not accurately represent waveforms that occur more rapidly than this time. LA_TIME is simply the dual of LA_FREQ. The default is 1ns, equivalent to setting LA_FREQ=1GHz.
.OPTION LA_TOL=*value*	Error tolerance for the PACT algorithm. *value* is the error tolerance for the PACT algorithm, is between 0.0 and 1.0. The default is 0.05.

Example

In this example, the circuit has a typical risetime of 1ns. Set the maximum frequency to 1 GHz, or set the minimum switching time to 1ns.

```
.OPTION LA_FREQ = 1GHz
-or-
.OPTION LA_TIME = 1ns
```

However, if spikes occur in 0.1ns, HSPICE will not accurately simulate them. To capture the behavior of the spikes, use:

```
.OPTION LA_FREQ = 10GHz
-or-
.OPTION LA_TIME = 0.1ns
```

Note:

Higher frequencies (smaller times) increase accuracy, but only up to the minimum time step used in HSPICE.

Supporting Parasitic L- and K-elements

HSPICE supports simulation with parasitic L- and K-elements. You need to set the minimum value of mutual inductance using the KLIM option. The default value of KLIM is 10mH. The second-order mutual inductance is not calculated for values less than KLIM, but parasitic mutual inductance values can be many orders smaller than the default value.

Also, note that RC reduction will not be very effective with respect to L- and K-elements. If you increase the simulation speed of your netlist having a huge number of parasitic elements, you need to properly understand the accuracy versus speed trade-off. For more information about the KLIM option, see .OPTION KLIM in the *HSPICE Reference Manual: Commands and Control Options*.

26

MOSFET Model Reliability Analysis (MOSRA)

Describes the procedures for HSPICE MOSFET reliability analysis (MOSRA).

These topics are covered in the following sections:

Overview

As CMOS technology is scaled down, reliability requirements become both more challenging and more important in maintaining the long-term reliability of these devices. Two of the most critical reliability issues, effects due to the hot carrier injection (HCI) and the negative bias temperature instability (NBTI) can change the characteristics of MOS devices. HSPICE reliability analysis allows circuit designers to predict the reliability of their design to allow enough margin for their circuits to function correctly over their lifetime.

A unified custom reliability modeling MOSRA API is available. Contact your Synopsys technical support team for more information.

Usage Model

HSPICE reliability analysis (or HCI and NBTI analysis), is always a two-phase simulation: the fresh simulation phase and the post-stress simulation phase.

- Fresh simulation phase: HSPICE computes the electron age/stress of selected MOS transistors in the circuit based on circuit behavior and HSPICE built-in stress model including HCI and/or NBTI effect.
- Post-stress simulation phase: HSPICE simulates the degradation effect on circuit performance, based on the stress information produced during the fresh simulation phase.

Beginning with the A-2007.09 release, the HSPICE MOSRA feature supports the following HSPICE MOSFET levels: Level 49, Level 53, Level 54, Level 57, Level 66, Level 70, and Level 71, and external CMI MOSFET models.

The HSPICE reliability analysis process is presented in Figure 114.

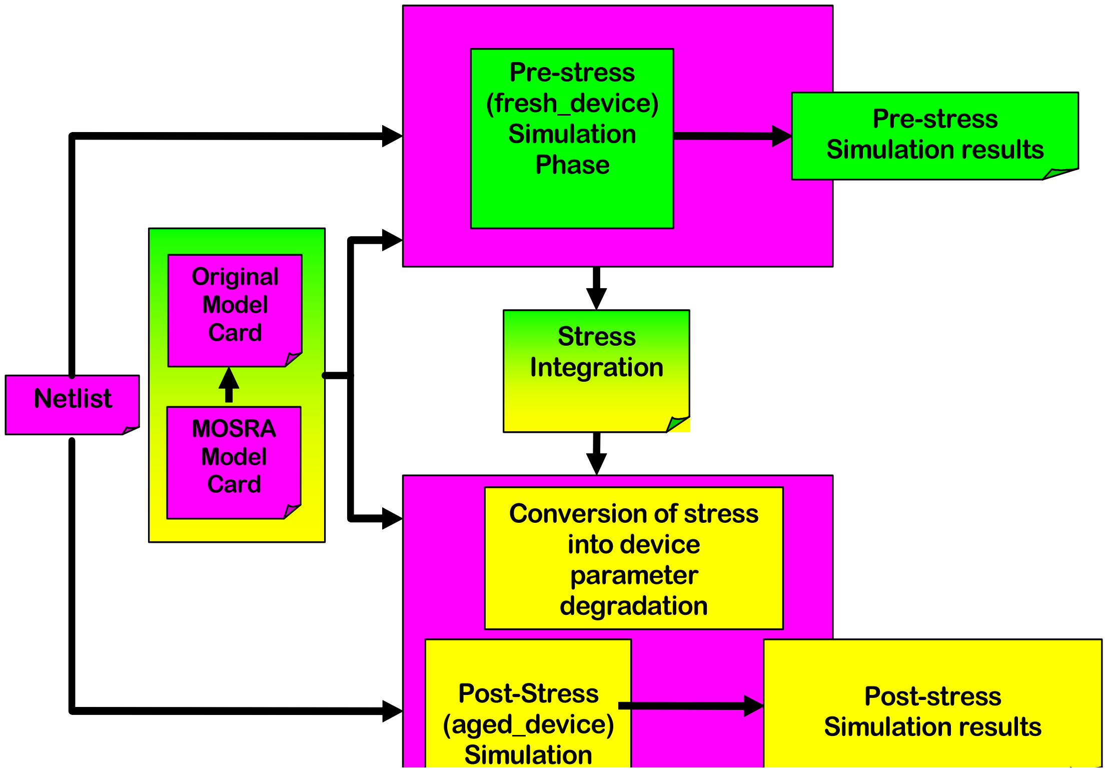

Pre-stress (fresh_device) Simulation Phase
Pre-stress Simulation results
Original Model Card
MOSRA Model Card
Netlist
Stress Integration
Conversion of stress into device parameter degradation
Post-Stress (aged_device) Simulation
Post-stress Simulation results

Figure 114 HSPICE Reliability Simulation Flow

Example Setup

The following example file demonstrates how to set up a HSPICE reliability reliability analysis.

```
* MOSRA TEST
vdd 1 0 2
mp1 3 2 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn1 3 2 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
mp2 4 3 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn2 4 3 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
mp3 2 4 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn3 2 4 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
c1 2 0 .1p

.model p1 pmos level=54 version=4.5
.model n1 nmos level=54 version=4.5

.model p1_ra mosra level=1
+tit0 = 5e-8  titfd = 7.5e-10  tittd = 1.45e-20
+tn = 0.25

.appendmodel p1_ra mosra p1 pmos
.mosra reltotaltime=1e8

.ic v(2)=2
.tran .1ps 5ns
.options post
.end
```

.MOSRA, .MOSRAPRINT, .MODEL, and .APPENDMODEL Commands

The following are descriptions of the command syntax and arguments for the `.MOSRA`, `MOSRAPRINT`, `. MODEL`, and `.APPENDMODEL` statements.

.MOSRA

Starts HSPICE HCI and/or NBTI reliability analysis.

Syntax

```
.MOSRA RelTotalTime=time_value
+ [RelStartTime=time_value] [DEC=value] [LIN=value]
+ [RelStep=time_value] [RelMode=0|1|2] SimMode=[0|1|2]
+ [AgingStart=time_value] [AgingStop=time_value]
+ [AgingPeriod=time_value] [AgingWidth=time_value]
+ [AgingInst="inst_name"]
+ [HciThreshold=value] [NbtiThreshold=value]
+ [Integmod=0|1|2] [Xpolatemod=0|1|2]
+ [Tsample1=value] [Tsample2=value]
```

Argument	Description
RelTotalTime	Defines final reliability test time to use in post stress simulation phase. Required argument.
RelStartTime	Time point of the first post-stress simulation. Default is 0.
DEC	Specifies number of post-stress time points simulated per decade.
LIN	Specifies the linear post-stress time points from RelStartTme to RelTotalTime.
RelStep	Performs post-stress simulation phase on time= RelStep, 2* RelStep, 3* RelStep, … until it achieves the RelTotalTime; the default is equal to RelTotalTime. Value is ignored if DEC or LIN value is set.
RelMode	Selects whether a simulation accounts for both HCI and NBTI effects or either one of them. If RelMode in the `.MOSRA` command is not set or set to 0, then the RelMode inside individual MOSRA models takes precedence for that MOSRA model only; the rest of the MOSRA models take the RelMode value from the `.MOSRA` command. If any other value except 0, 1, or 2 is set, a warning message is issued, and RelMode is set to default value 0. The RelMode parameter is also available in the MOSRA model card with additional restrictions. ■ 0: both HCI and NBTI, Default. ■ 1: HCI only ■ 2: NBTI only
SimMode	■ 0: Select pre-stress simulation only ■ 1: Select post-stress simulation only ■ 2: Select both pre- and post-stress simulation If set to 1, HSPICE reads in the **.radeg0* file for post-stress simulation and uses it to update the input for reliability analysis; the transient output is in a **.tr1* waveform file. The **.radeg* file should be in the same directory as the input netlist. When SimMode =1, the netlist stimuli can be different from the SimMode=0 netlist that generated the **.radeg* file.
AgingStart	Optionally defines time when HSPICE starts stress effect calculation during transient simulation. Default is 0.0.

Argument	Description
AgingStop	Optionally defines time when HSPICE stops stress effect calculation during transient simulation. Default is tstop in `.TRAN` statement.
AgingPeriod	Specifies the stress period; scales the total degradation over time.
AgingWidth	The AgingWidth (circuit time “on”) argument works with the AgingPeriod argument. For example: if you specify AgingPeriod=1.0s and AgingWidth=0.5s, then the circuit will be turned on for 0.5s, then be turned off for 0.5s. (The period is 1.0s.)
HciThreshold	Optionally used in post stress simulation; this argument allows you to define whether the HCI effect is accounted for in a particular transistor, based on the specified HCI threshold value. Default is 0.0
NbtiThreshold	Optionally used in post stress simulation; this argument allows you to define whether the NBTI effect will be accounted for in a particular transistor, based on this threshold value. Default is 0.0
AgingInst	Selected MOSFET devices to which HSPICE applies HCI and/or NBTI analysis. The default is all MOSFET devices with reliability model appended. The name must be surrounded by quotes. Multiple names are allowed, and wildcards are supported.
Integmod	The flag is used to select the integration method and function. ■ 0 (default): call the MOSRAAPI integration function ■ 1: True derivation and integration method ■ 2: Linearization and integration method (support non-constant n coefficient)
Xpolatemod	The flag is used to select the extrapolation method and function. ■ 0 (default): call the MOSRAAPI extrapolation function ■ 1: Linearization extrapolation method (support non-constant n coefficient) ■ 2: Aeff/Neff extraction and extrapolation method
Tsample1	First simulation time point of stress_total sampling for Aeff/Neff extraction
Tsample2	Second simulation time point of stress_total sampling for Aeff/Neff extraction

Description

Use the `.MOSRA` command to initiate HCI and NBTI analysis. This is a two-phase simulation, the fresh simulation phase and the post stress simulation phase. During the fresh simulation phase, HSPICE computes the electron age/stress of selected MOS transistors in the circuit based on circuit behavior and the HSPICE built-in stress model including HCI and/or NBTI effect. During the post stress simulation phase, HSPICE simulates the degradation effect on circuit performance, based on the stress information produced during the fresh simulation phase.

If either DEC or LIN is specified, the RelStep value is ignored. See Figure 115 on page 633 for an illustration of the `.MOSRA` command/syntax.

Example

```
.mosra reltotaltime=6.3e+8 relstep=6.3e+7
+ agingstart=5n agingstop=100n
+ hcithreshold=0 nbtithreshold=0
+ aginginst="x1.*"
```

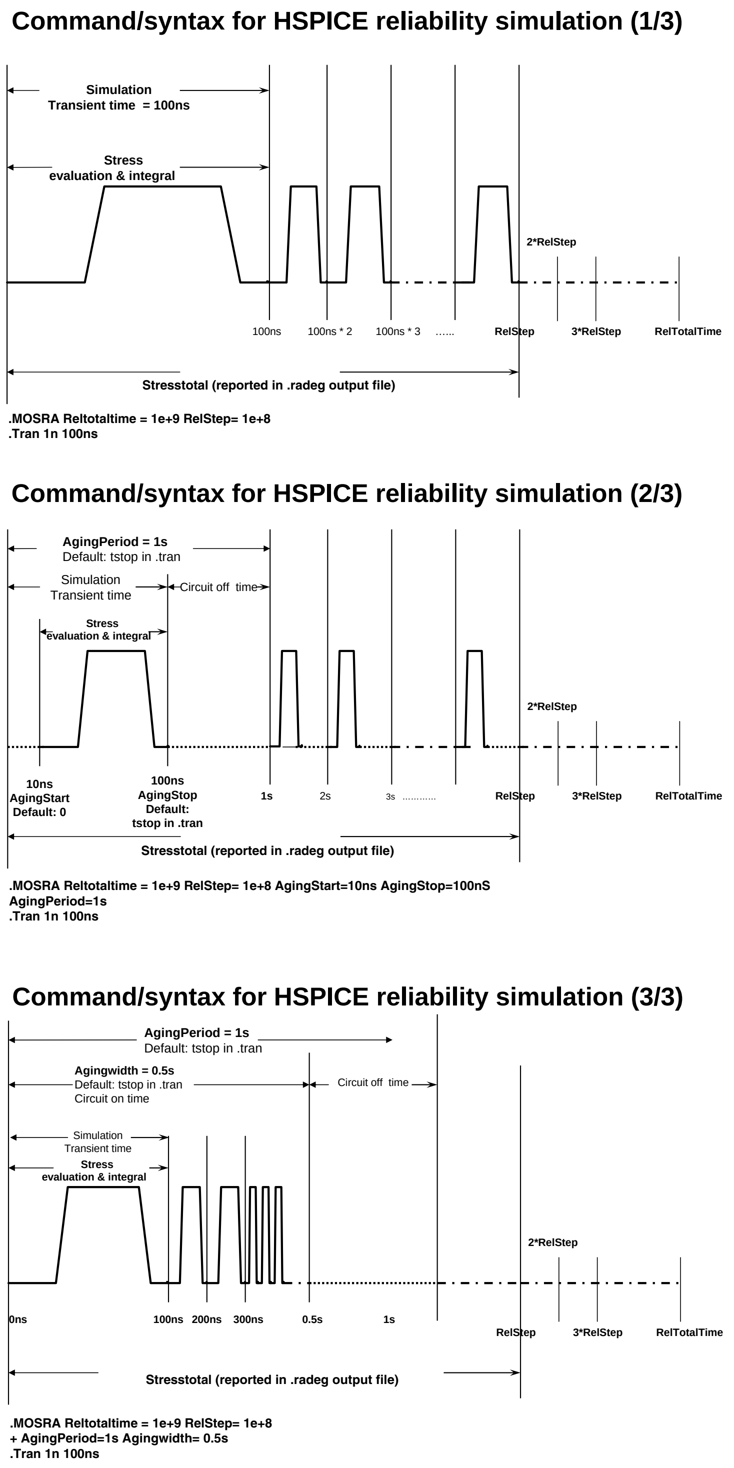


Figure 115 Graphic Illustration of MOSRA Command/Syntax

.MOSRAPRINT

Provides `.PRINT/.PROBE` usage to the `.MOSRA` command.

Syntax

```
.MOSRAPRINT output_name output_type(element_name, vds=exp1,
   vgs=exp2, vbs=exp3)
```

Argument	Description
output_nam	User-defined output variable; this `output_name@element_name` is used as the as output variable name in the output file.
output_type	Can be one of output variable types `vth`, `gm`, `gds`, or `ids`.
element_name	Output variable name in the output file.

Definition

This command provides measurement functionary to the `.MOSRA` command. This syntax prints the Ids value of the mosfet at the final reliability test time. There is no order requirement for vds, vgs, and vbs. The wildcards '`?`' and '`*`' can be used in element_name.

The output file format is the same as the measurement file format. The extension file name for this file is **.ra#*.

Example

The following syntax prints the Ids value of the mosfet m1, when vds = 5 vgs= 5, vbs = 0, at the reltime point.

```
.MOSRA reltotaltime=5e+7 relstep=1e+7
.MOSRAPRINT ids(m1, vds=5, vgs=5, vbs=0)
```

.MODEL

Syntax

```
.model mname mosra
+ level=value
+ [RelMode=0|1|2]
+ <relmodelparam>
```

Argument	Description
mname	User-defined MOSFET reliability model name
mosra	HSPICE model type name for MOSFET reliability model

level (alias: mosralevel)	To use the Synopsys MOSRA model, set LEVEL=1. For compatibility with HSIM, in the `.MODEL` statement, 'LEVEL=' can be replaced with 'MOSRALEVEL='. HSPICE will consider them equivalent. Example: The following two lines will be interpreted the same by HSPICE.
	`.MODEL my_mod MOSRA LEVEL=1`
	`.MODEL my_mod MOSRA MOSRALEVEL=1`
	To use custom MOSRA models and for discussion of LEVEL values, refer to the HSPICE Application Note: Unified Custom Reliability Modeling API (MOSRA API). Contact your Synopsys technical support team for more information.

RelMode	HSPICE reliability mode level; selects whether a simulation accounts for both HCI and NBTI effects or either one of them. If the RelMode in the `.MOSRA` command is defined as 1 or 2, it takes higher priority and applies to all MOSRA models. If RelMode in the `.MOSRA` command is not set or set to 0, then the RelMode inside individual MOSRA models take precedence for that MOSRA model only; the rest of the MOSRA models take the RelMode value from the `.MOSRA` command. If any other value is set, except 0, 1, or 2, a warning is issued, and RelMode is automatically set to the default value 0. ■ 0: both HCI and NBTI, Default. ■ 1: HCI only ■ 2: NBTI only
relmodelparam	Reliability model parameter for HCI and NBTI. See the following sections for HCI and NBTI parameters.

Synopsys LEVEL1 mosra model, NBTI for PMOS

Vth degradation

Table 66 NBTI for PMOS Vth degradation

Parameter	Default	Description
tit0	5e-7	First parameter for interface trap inducing threshold voltage degradation
titce	0	Inversion charge exponent for interface trap inducing threshold voltage degradation
titfd	7.5e-10	Oxide electric field dependence for interface trap inducing threshold voltage degradation
titlc	0	Channel length coefficient for oxide trap inducing threshold voltage degradation
title	0	Channel length exponent for oxide trap inducing threshold voltage degradation
tittd	1.45e-20	Temperature dependent component of interface trap inducing threshold voltage degradation
titwc	0	Channel width coefficient for interface trap inducing threshold voltage degradation
titwe	0	Channel width exponent for interface trap inducing threshold voltage degradation
tn	0.25	Stress time exponent for interface trap inducing threshold voltage degradation
tot0	0	First parameter for oxide trap inducing threshold voltage
totdd	0	Drain voltage dependent coefficient for oxide electric field in threshold voltage degradation
totde	1	Drain voltage exponent for oxide electric field in threshold voltage degradation
totfd	0	Oxide electric field dependent component for oxide trap inducing threshold voltage degradation

Table 66 NBTI for PMOS Vth degradation (Continued)

Parameter	Default	Description
tottd	0	Temperature dependent component for oxide trap inducing threshold voltage degradation
totwc	0	Channel width coefficient of oxide trap inducing threshold voltage degradation
totwe	0	Channel width exponent of oxide trap inducing threshold voltage degradation
totlc	0	Channel length coefficient of oxide trap inducing threshold voltage degradation
totle	0	Channel length component of oxide trap inducing threshold voltage degradation
tk	0.5	Stress time exponent for oxide trap inducing threshold voltage degradation
tfd0	0	First parameter for transient degradation of threshold voltage
tdcd	0	Duty cycle dependent exponent for transient degradation of threshold voltage

Mobility degradation

Table 67 NBTI for PMOS Mobility degradation

Parameter	Default	Description
udcd	0	Duty cycle dependent coefficient for transient mobility degradation
uit0	0	First parameter for interface trap inducing mobility degradation
uitce	0	Inversion charge exponent for interface trap inducing mobility degradation
uitfd	0	Oxide electric field dependence for interface trap inducing mobility degradation
uitlc	0	Channel length dependent coefficient for interface trap inducing mobility degradation
uitle	0	Channel length exponent for interface trap inducing mobility degradation
uittd	0	Temperature dependent coefficient of interface trap inducing mobility degradation
uitwc	0	Channel width dependent coefficient for interface trap inducing mobility degradation
uitwe	0	Channel width exponent for interface trap inducing mobility degradation
uk	0.5	Stress time exponent for oxide trap inducing mobility degradation
un	0.25	Stress time exponent for interface trap inducing mobility degradation
uot0	0	First parameter for transient degradation
uotdd	0	Drain voltage dependent coefficient for oxide electric field in mobility degradation
uotde	1	Drain voltage exponent for oxide electric field in mobility degradation
uotfd	0	Frequency exponent for transient degradation

Table 67 NBTI for PMOS Mobility degradation (Continued)

Parameter	Default	Description
uotlc	0	Channel length coefficient for oxide trap inducing mobility degradation
uotle	0	Channel length exponent for oxide trap inducing mobility degradation
uottd	0	Duty cycle dependent exponent for transient degradation
uotwc	0	Channel width coefficient for oxide trap inducing mobility degradation
uotwe	0	Channel width exponent for oxide trap inducing mobility degradation
utd0	0	First parameter transient mobility degradation

Synopsys LEVEL1 mosra model, HCI for NMOS and PMOS

Table 68 HCI for NMOS and PMOS

Parameter	Default	Description
hiie	0	Saturation channel electric field for the impact ionization current
hiii	0	First parameter for impact ionization
hiil	0	Channel length dependent parameter for the impact ionization current
hiit	0	Temperature dependent parameter for the impact ionization current
hiivd0	0	First vds dependent parameter of the impact ionization current
hiivd1	0	Second vds dependent parameter of the impact ionization current
hiivd2	0	Third vds dependent parameter of the impact ionization current
hiivg0	0	First vgs dependent parameter for the impact ionization current
hiivg1	0	Second vgs dependent parameter for the impact ionization current
hiivg2	0	Third vgs dependent parameter for the impact ionization current
hiivgd	0	vds dependent parameter for the impact ionization current
hk	0.5	Time exponent for mobility degradation induced by HCI
hn	0.5	Time exponent for threshold voltage degradation induced by HCI
Isubmode	1	Selection method for calculation of HCI degradation. ■ 0: selects isub (substrate current) to calculate HCI degradation ■ 1: selects Iii (impact ionization current)
tdce	0	Channel current exponent for threshold voltage degradation induced by HCI
tdii	0	Impact ionization current exponent for threshold voltage degradation induced by HCI

Table 68 HCI for NMOS and PMOS (Continued)

Parameter	Default	Description
tdle	0	Channel length exponent for threshold voltage degradation induced by HCI
thci0	0	First parameter for threshold voltage degradation induced by HCI
udce	0	Channel current exponent for mobility degradation induced by HCI
udii	0	Impact ionization current exponent for mobility degradation induced by HCI
udle	0	Channel length exponent for mobility degradation induced by HCI
uhci0	0	First parameter for mobility degradation induced by HCI
vdsat0	0	Nominal drain saturation voltage of the impact ionization current

.APPENDMODEL

This command appends the parameter values from the source model card (SrcModel) to the destination model card (DestModel). All arguments are required. Wildcards are supported for the `.APPENDMODEL` command. In addition, the `.OPTION APPENDALL` enables the top hierarchical level to use the `.APPENDMODEL` command even if the MOSFET model is embedded in a subcircuit. See .OPTION APPENDALL in the *HSPICE Reference Manual: Commands and Control Options*.

Syntax

```
.appendmodel SrcModel ModelKeyword1 DestModel ModelKeyword2
```

Argument	Description
SrcModel	Source model name, e.g., the name of the MOSRA model.
ModelKeyword1	Model type for SrcModel. For example, the keyword “mosra”.
DestModel	Destination model name, e.g., the original model in the model library.
ModelKeyword2	Model type for DestModel. For example, 'nmos'.

Example

This example appends the content of the model card hci_1 to the b3_nch BSIM3 model card.

```
.appendmodel hci_1 mosra b3_nch nmos
```

Wildcard Examples

In this example, the `mosra` model `p1_ra` is appended to all of the pmos models. Note that you need quotation marks if the model name is defined only by a wildcard.

```
.appendmodel p1_ra mosra  "*"   pmos
```

In the following example, the `mosra` model `p1_ra` is appended to all of the `pmos` models that are named `pch*` (`pch1`, `pch2`, `pch_tt`, etc.).

```
.appendmodel p1_ra mosra   pch*   pmos
```

.OPTION APPENDALL

With this option, `.APPENDMODEL` at the main (uppermost) circuit level hierarchy can be used even if the MOSFET model is embedded in a subcircuit. If there is `.APPENDMODEL` both in the main circuit and in a subcircuit, the `.APPENDMODEL` in the subcircuit will have higher priority. For example:

```
.option appendall
.appendmodel n_ra mosra nch nmos

.SUBCKT mosra_test 1 2 3 4
M1 1 2 3 4 nch L=PL W=PW
.model nch nmos level= ...
.ENDS
```

The `.APPENDMODEL` command in the main circuit will be used.

```
.option appendall
.appendmodel n_ra mosra nch nmos

.SUBCKT mosra_test 1 2 3 4
M1 1 2 3 4 nch L=PL W=PW
.model nch nmos level= ...
.appendmodel n_ra1 mosra nch nmos
.ENDS
```

The `.APPENDMODEL` command in the subcircuit will be used.

Simulation Output File

For each post-stress circuit time point, HSPICE generates a set of reliability data containing the stress information for the selected transistors. HSPICE then back-annotates the degradations to these transistors (aged_device) and performs the post-stress simulation. For example:

```
Degraded_device_parameter_Vth0 = Fresh_vth0 + delvth (NMOS)
Degraded_device_parameter_Vth0 = Fresh_vth0 -  delvth (PMOS)
Degraded_device_paramter_U0 = Fresh_u0 * mulu0
```

HSPICE combines reliability data for all post-stress points in the following output **.radeg* file.

```
delvth0  = 0.154229E-03
mulu0    =  99.9985%

** $DATA1 SOURCE='HSPICE' VERSION='A-2008.03-BETA3 32-BIT'
 ******Result of Reliability Analysis******
mosrasort: delvth0

Circuit time 0.100000E+05

mn1
Device Type: NMOS
L= 0.100000E-06
W= 0.500000E-05
Bias Direction: forward
delvth0   = 0.000000E+00
mulu0     =     100.0000%
mulua     =     100.0000%
mulub     =     100.0000%
muluc     =     100.0000%
delnfactor= 0.000000E+00

mn2
Device Type: NMOS
L= 0.100000E-06
W= 0.500000E-05
Bias Direction: forward
delvth0   = 0.000000E+00
mulu0     =     100.0000%
mulua     =     100.0000%
mulub     =     100.0000%
muluc     =     100.0000%
delnfactor= 0.000000E+00
```

```
mp2
Device Type: PMOS
L= 0.100000E-06
W= 0.100000E-04
Bias Direction: forward
delvth0   = 0.169531E-01
mulu0     =     100.0000%
mulua     =     100.0000%
mulub     =     100.0000%
muluc     =     100.0000%
delnfactor= 0.000000E+00

mp1
Device Type: PMOS
L= 0.100000E-06
W= 0.100000E-04
Bias Direction: forward
delvth0   = 0.167418E-01
mulu0     =     100.0000%
mulua     =     100.0000%
mulub     =     100.0000%
muluc     =     100.0000%
delnfactor= 0.000000E+00
```

RADEG Output Sorting (.OPTION MOSRASORT)

HSPICE uses the option `mosrasort` to enable the descending sort for reliability degradation (RADEG) output.

`.option mosrasort=`*`degradation_type_keyword`*

The degradation type could be any degradation keyword in the RADEG output. For example, `.option mosrasort=delvth0`

HSPICE does a descending sort for RADEG output on `delvth0`'s value.

If the `mosrasort` option is not specified, or the degradation type keyword is not recognized, HSPICE will not do the sorting. (Degradation type keywords are listed in the *HSPICE Application Note: Unified Custom Reliability Modeling API (MOSRA API)*, available by contacting SPICE technical support.) If you only specify the option `mosrasort`, and do not specify the degradation type keyword, HSPICE sorts RADEG by the `delvth0` keyword. HSPICE sorts the output separately in lists, one for NMOS, one for PMOS. HSPICE prints the NMOS list first, and then the PMOS list.

Known Limitation

HSPICE MOSRA currently does not work with Monte Carlo analysis.

Part: 4 Simulation Applications

This Part contains the following chapters/topics.

27

Performing Digital Cell Characterization

Describes how to characterize cells in data-driven analysis. Also shows you some typical data sheet parameters.

Most ASIC vendors use the basic capabilities of the `.MEASURE` statement in Synopsys HSPICE or HSPICE RF to characterize standard cell libraries, and to prepare data sheets.

HSPICE or HSPICE RF stores input sweep parameters and measure output parameter, in measure output data files (design.mt0, design.sw0, and design.ac0). These files store multiple sweep data. You can use WaveView Analyzer to plot this data; for example, to generate fanout plots of delay versus load. You can also use the slope and intercept of the loading curves to calibrate VHDL, Verilog, Lsim, TimeMill, and Synopsys models.These topics are covered in the following sections:

- Performing Basic Cell Measurements
- Performing Advanced Cell Characterization
- Cell Examples

This chapter shows you some typical data sheet parameters. By looking at a series of typical data sheet examples, you can see the flexibility of the `.MEASURE` statement.

This chapter also shows you how to characterize cells in data-driven analysis. Data-driven analysis automates cell characterization, including calculating the delay coefficient for the timing-simulator polynomial. You can simultaneously vary an unlimited number of parameters, or the number of analyses to perform. Cell characterization uses an ASCII file format for automated parameter input to HSPICE or HSPICE RF.

Performing Basic Cell Measurements

This section describes how to perform basic cell measurements.

Rise, Fall, and Delay Calculations

The following example does the following:

- Uses the `MAX` function to calculate `vmax`, over the time region of interest.
- Uses the `MIN` function to calculate `vmin`.
- Uses the measured parameters in subsequent calculations, for accurate 10% and 90% points, when determining the rise and fall time.

  `RISE=1` is relative to the time window that the `TDval` delay forms.
- Uses a fixed value for the measure threshold, to calculate the `Tdelay` delay.

```
.MEAS TRAN vmax MAX V(out) FROM=TDval TO=Tstop
.MEAS TRAN vmin MIN V(out) FROM=TDval TO=Tstop
.MEAS TRAN Trise TRIG V(out) val='vmin+0.1*vmax'
+ TD=TDval RISE=1 TARG V(out) val='0.9*vmax' RISE=1
.MEAS TRAN Tfall TRIG V(out) val='0.9*vmax' TD=TDval
+ FALL=2 TARG V(out) val='vmin+0.1*vmax' FALL=2
.MEAS TRAN Tdelay TRIG V(in) val=2.5 TD=TDval FALL=1
+ TARG V(out) val=2.5 FALL=2
```

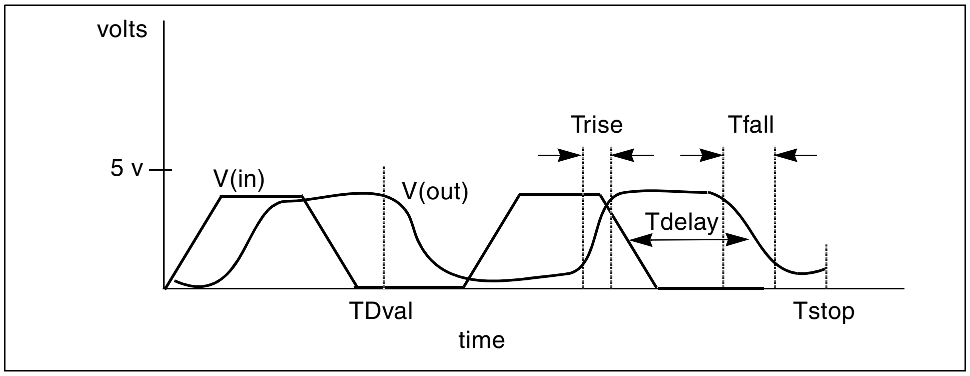


Figure 116 Rise, Fall, and Delay Time Demonstration

Ripple calculation performs the following:

- Delimits the wave at the 50% of VCC points
- Finds the `Tmid` midpoint
- Defines a bounded region by finding the pedestal voltage (`Vmid`) and then finding the first time that the signal crossed this value, `Tfrom`
- Measures the ripple in the defined region using the peak-to-peak (`PP`) measure function from `Tfrom` to `Tmid`

The following is an example:

```
.MEAS TRAN Th1 WHEN V(out)='0.5*vcc' CROSS=1
.MEAS TRAN Th2 WHEN V(out)='0.5*vcc' CROSS=2
.MEAS TRAN Tmid PARAM='(Th1+Th2)/2'
.MEAS TRAN Vmid FIND V(out) AT='Tmid'
.MEAS TRAN Tfrom WHEN V(out)='Vmid' RISE=1
.MEAS TRAN Ripple PP V(out) FROM='Tfrom' TO='Tmid'
```

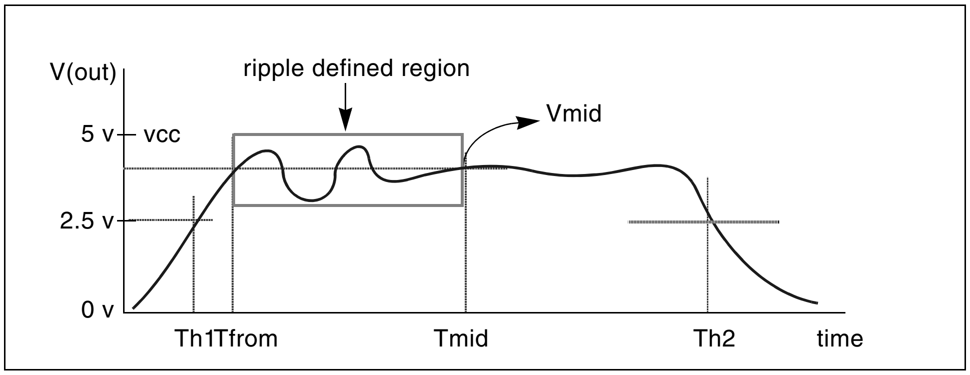


Figure 117 Waveform to Demonstrate Ripple Calculation

This file sweeps the sigma of the model parameter distribution, while it examines the delay. It shows you the delay derating curve, for the worst cases in the model. This example is based on demonstration netlist sigma.sp, which is available in directory $<installdir>/demo/hspice/cchar.

For a description of this technique for building a worst-case sigma library, see the HSPICE Simulation and Analysis User Guide.

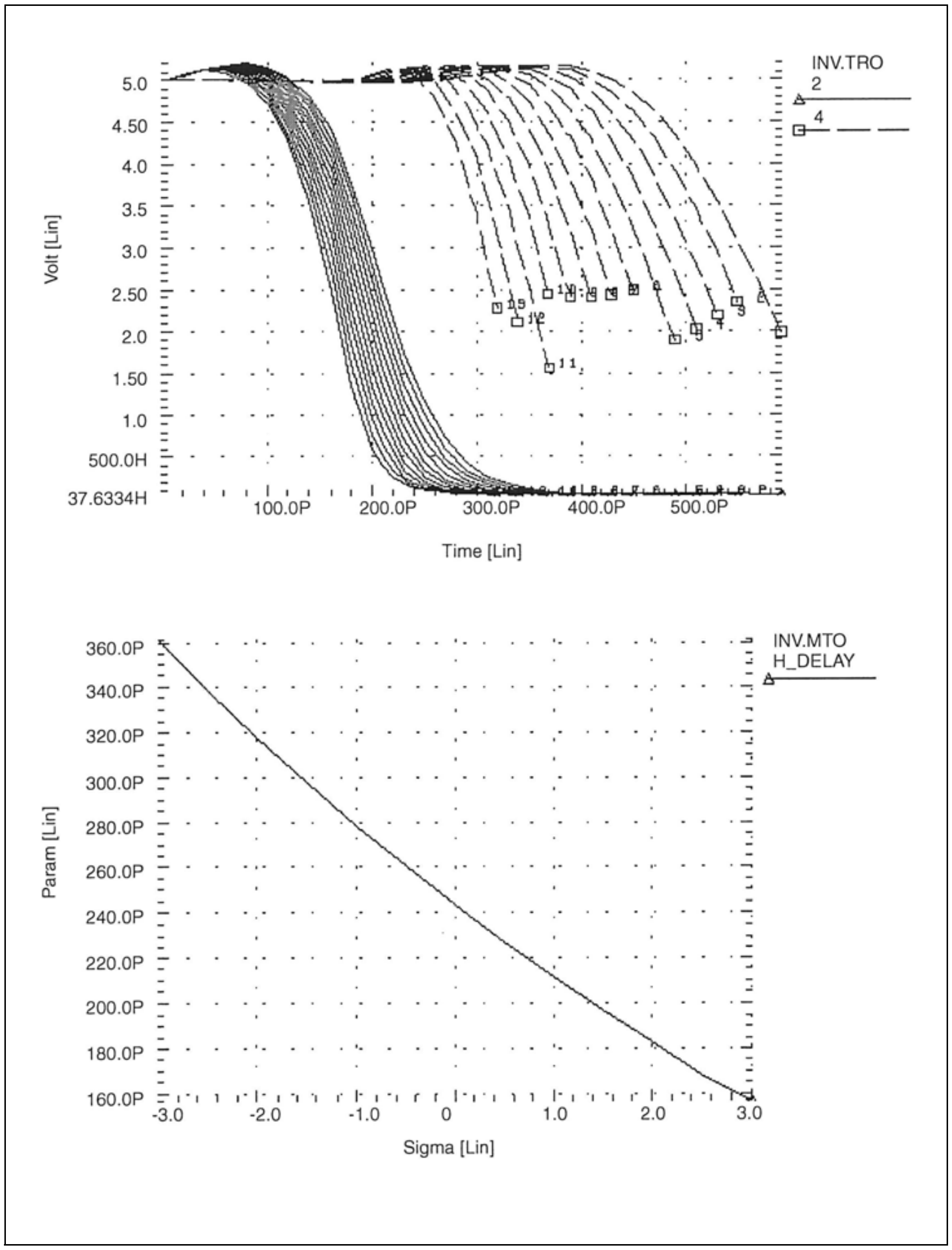


Figure 118 Inverter Pair Transfer Curves and Sigma Sweep vs. Delay

Delay versus Fanout

The example sweeps the subcircuit multiplier to quickly generate five load curves. To obtain more accurate results, buffer the input source with one stage.

For each second-sweep variable (`m_delay` and `rms_power`), the example calculates:

- mean
- variance
- sigma
- average deviance

This example is based on the demonstration netlist load1.sp, which is available in directory $<installdir>/demo/hspice/cchar.

This example outputs the following results:

```
meas_variable = m_delay
mean = 273.8560p varian = 1.968e-20
sigma = 140.2711p avgdev = 106.5685p

meas_variable = rms_power
mean = 5.2544m varian = 8.7044u
sigma = 2.9503m avgdev = 2.2945m
```

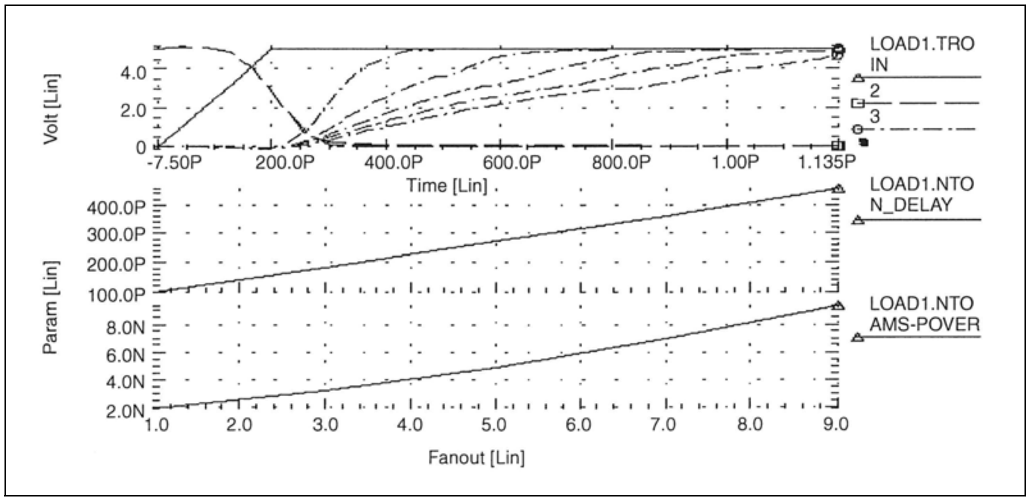


Figure 119 Inverter Delay and Power, versus Fanout

Pin Capacitance Measurement

This example does the following:

- Shows the effect of dynamic capacitance, at the switch point.
- Sweeps the DC input voltage (`pdcin`) to the inverter.
- Performs an AC analysis, at each 0.1 V increment.
- Calculates the `incap` measure parameter from the imaginary current through the voltage source at 10 kHz in the AC curve (not shown).

The peak capacitance (at the switch point) occurs when the voltage at the output side changes, in the direction opposite the input side of the Miller capacitor. This adds the Miller capacitance, times the inverter gain, to the effective capacitance.

```
mp out in 1 1 mp w=10u l=3u
mn out in 0 0 mn w=5u l=3u
vin in 0 DC= pdcin AC 1 0
.ac lin 2 10k 100k sweep pdcin 0 5 .1
.measure ac incap find par( '-1 * ii(vin)/
+ (hertz*twopi)' ) AT=10000hertz
```

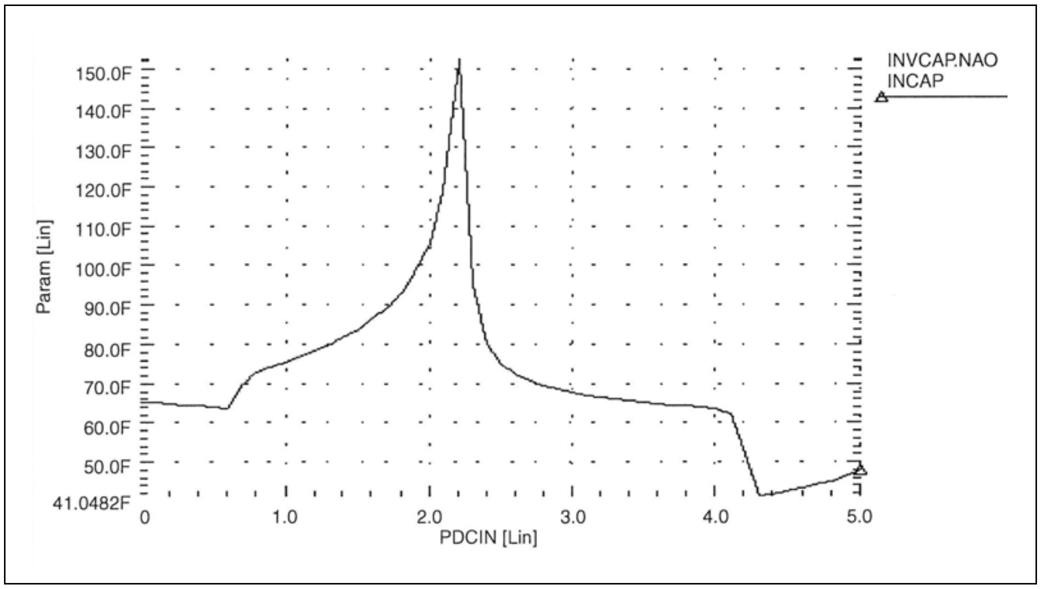


Figure 120 Graph of Pin Capacitance versus Inverter Input Voltage

Op-amp Characterization of LM124

This example analyzes op-amps. This example uses:

- `.MEASURE` statements to present a very complete data sheet.
- Four `.MEASURE` statements, to reference the `out0` output node of an op-amp circuit. These statements use output variable operators for parameters:
  - decibels `vdb(out0)`
  - voltage magnitude `vm(out0)`
  - phase `vp(out0)`

This example is based on the demonstration file alm124.sp, which is located in $<installdir>/demo/apps.

This example outputs the following results:

```
unitfreq = 9.0786E+05 targ= 9.0786E+05 trig= 1.0000E+00
phasemargin = 6.6403E+01

gain(db) = 9.9663E+01 at= 1.0000E+00 from= 1.0000E+00
+ to= 1.0000E+07

gain(mag)= 9.6192E+04 at= 1.0000E+00 from= 1.0000E+00
+ to= 1.0000E+07
```

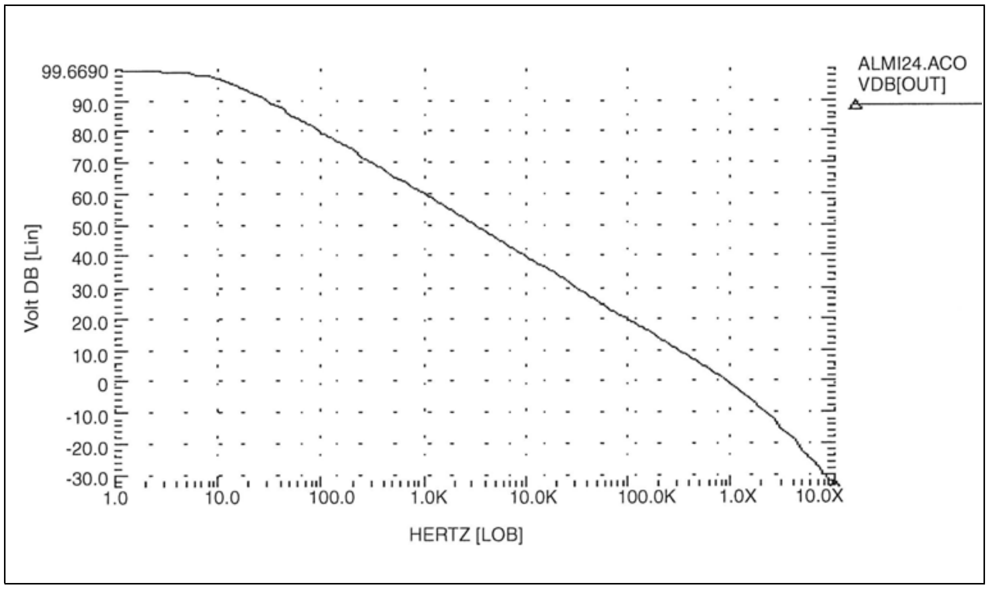


Figure 121 Magnitude Plot of Op-Amp Gain

Performing Advanced Cell Characterization

This section provides example input files, which characterize cells for an inverter, based on 3-micron MOSFET technology. The program finds the propagation delay, and the rise and fall times, for the inverter, using best, worst, and typical cases for different fanouts. You can use this library data for digital-based simulators, such as those used to simulate gate arrays and standard cells.

The example is based on the demonstration file cellchar.sp, which is located in $<*installdir*>/demo/hspice/apps. It demonstrates how to use the following to characterize a CMOS inverter:

- `.MEASURE` statement
- `.DATA` statement
- `AUTOSTOP` option
- `SUBCKT` definition
- `SUBCKT` call
- models

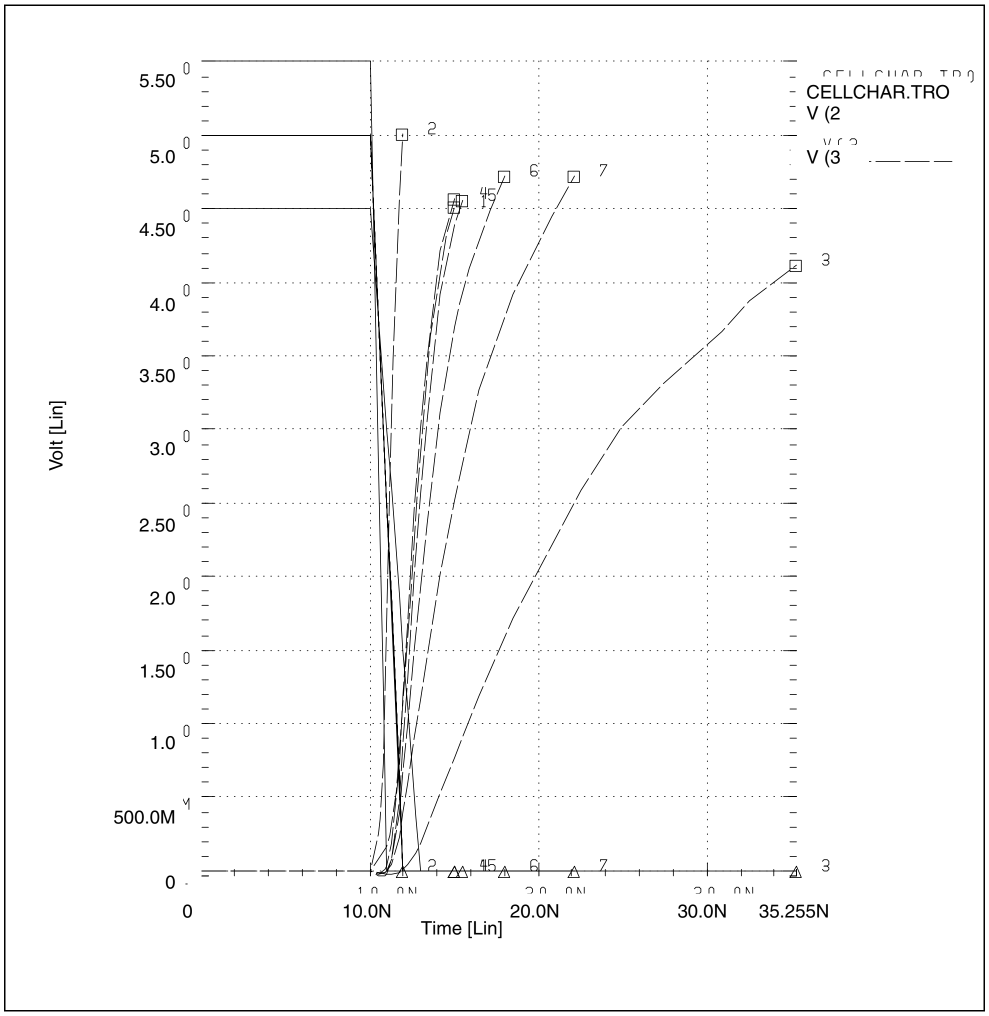


Figure 122 Plotting the Simulation Outputs

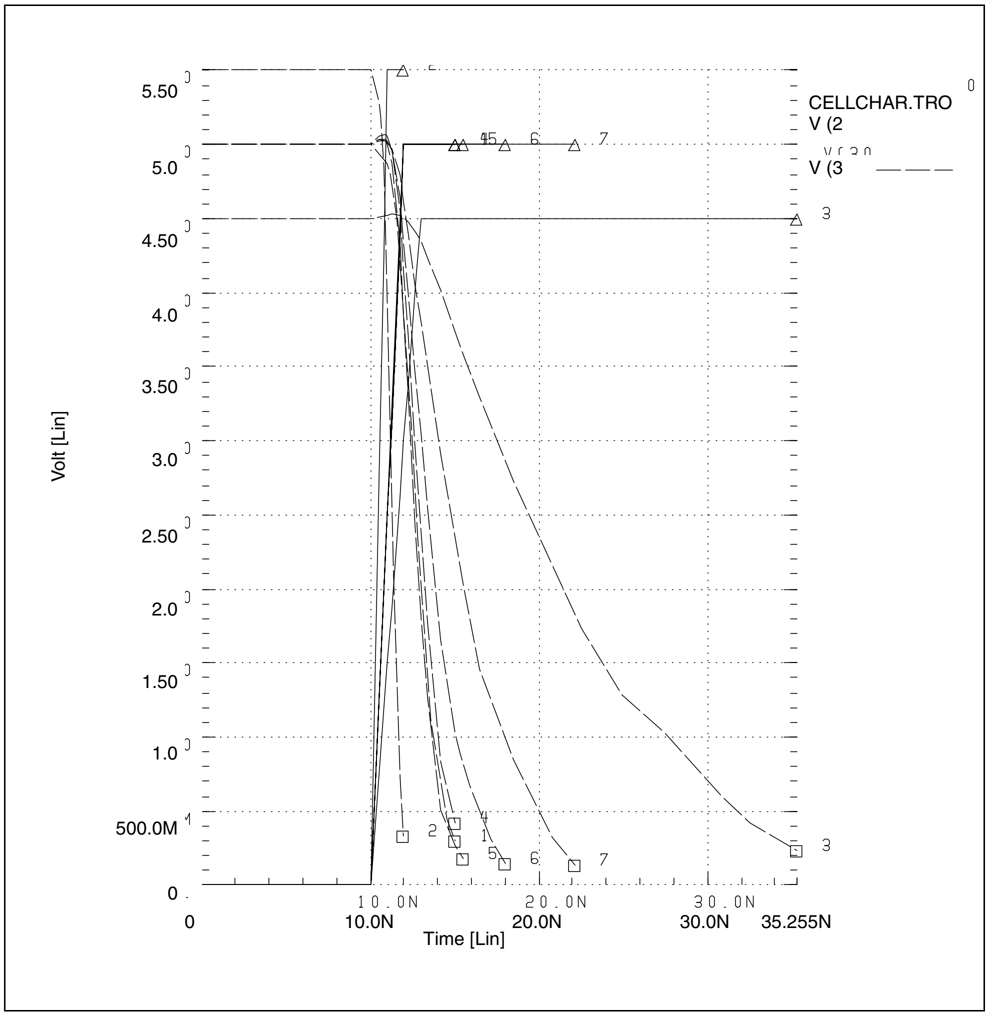


Figure 123 Verifying the Measure Statement Results by the Plots

Cell Examples

Figure 124 and Figure 125 are identical, except that their input signals are complementary.

- The circuit in Figure 124 calculates the rise time and the low-to-high propagation delay time.
- The circuit in Figure 125 calculates the fall time and the high-to-low propagation delay time.

If you use only one circuit, CPU time increases because analysis time increases when HSPICE or HSPICE RF calculates both rise and fall times.

The XOUTL or XOUTH subcircuit represents the fanout of the cell (inverter). To modify fanout, specify different multipliers (m) in the subcircuit calls.

You can also specify local and global temperatures. This example characterizes the cell at a global temperature of 27, but the temperature of the M1 and M2 devices is (27+`DTEMP`). The `.DATA` statement specifies the DTEMP value.

The example uses a transient parameterized sweep, with `.DATA` and `.MEASURE` statements, to determine the inverter timing, for best, typical, and worst cases.

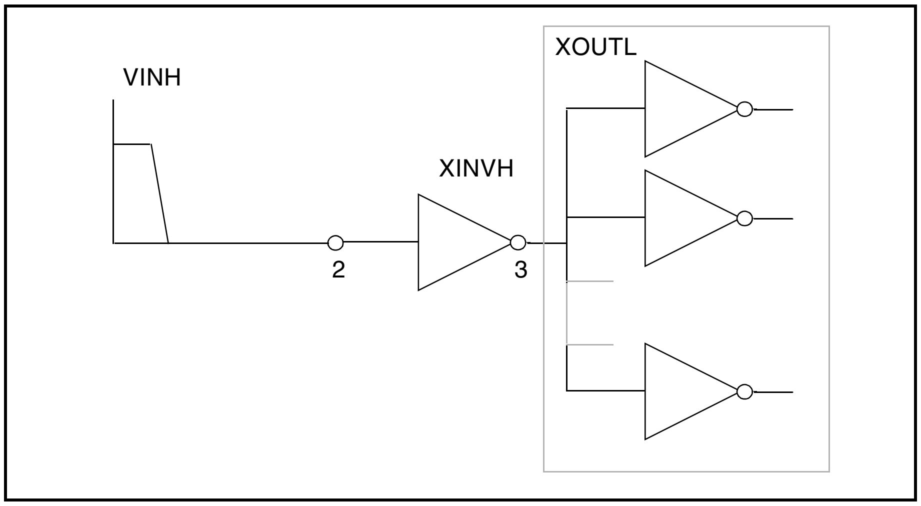


Figure 124 Cell Characterization Circuit 1

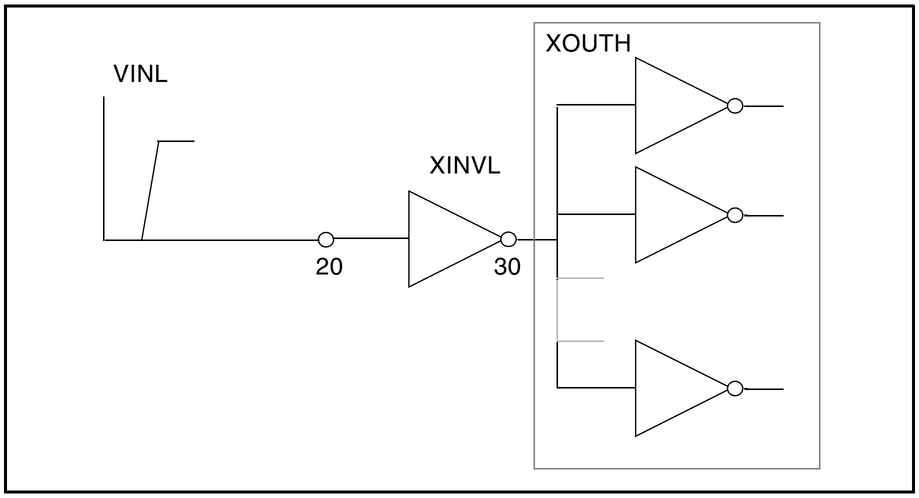


Figure 125 Cell Characterization Circuit 2

This example varies the following parameters:

- power supply
- input rise and fall time
- fanout
- MOSFET temperature
- n-channel and p-channel threshold
- drawn width and length of the MOSFET

Use the `.MEASURE` statement to specify a parameter to measure.

Use the `AUTOSTOP` option, to speed simulation time.The `AUTOSTOP` option terminates the transient sweep, although it has not completely swept the specified transient sweep range.

The `.MEASURE` statement uses quoted string parameter variables to measure the rise time, fall time, and propagation delays.

Note:

Do not use character strings as parameter values in HSPICE RF.

Rise time starts when the voltage at node 3 (the output of the inverter) is equal to 0.1 · VDD (that is, V(3) = 0.1VDD).Rise time ends when the voltage at node 3 is equal to 0.9 · VDD (that is, V(3) = 0.9VDD).

For more accurate results, start the `.MEASURE` calculation after either:

- A time delay, or
- A simulation cycle, specifying delay time in the `.MEASURE` statement, or
- An input pulse statement.

The following example features:

- `AUTOSTOP` option and `.MEASURE` statements.
- Mean, variance, sigma, and avgdev calculations.
- Circuit and element temperature.
- Algebraic equation handling.
- PAR( ) as an output variable, in the `.MEASURE` statement.
- Subcircuit parameter passing, and subcircuit multiplier.
- `.DATA` statement.

28

Behavioral Modeling

Describes how to create behavioral models.

Behavioral modeling substitutes more abstract, less computationally intensive, circuit models for lower-level descriptions of analog functions. These simpler models emulate the transfer characteristics of the circuit elements that they replace, but with increased efficiency. Behavioral modeling substantially reduces the actual simulation time per circuit. At the level of an entire design and simulation cycle, design efficiency greatly increases, and you can complete a design (from concept to marketable product) in substantially less time.

Note:

You can use the Verilog-A behavioral modeling language in Synopsys HSPICE and HSPICE RF.

These topics are presented in the following sections:

Behavioral Design Process

HSPICE provides specific modeling elements that promote the use of behavioral and mixed signal techniques. These models include controllable sources that you can configure, to emulate op-amps, single-input or multi-input logic gates, or any system with a continuous algebraic transfer function.

- You can create these functions in algebraic form, or in the form of coordinate pairs.
- You can use digital stimulus files, to enter logic waveforms into the simulation deck, rather than using piecewise linear sources to enter digital waveforms.
- You can define clock rise times, fall times, periods, and voltage levels.

With HSPICE behavioral models, the typical design cycle for a circuit or system is:

1. Fully simulate a subcircuit, with pertinent inputs, characterizing its transfer functions.
2. Determine which HSPICE elements, singularly or in combination, accurately describe the transfer function.
3. Reconfigure the subcircuit appropriately.
4. After you verify the behavioral model, substitute the model into the larger system, in place of the lower-level subcircuit.

Using Behavioral Elements

Behavioral elements offer a higher level of abstraction, and faster processing, compared to a lower-level description of an analog function.

- System-level designers can use function libraries of subcircuits, containing these elements, to describe parts such as:
  - op-amps
  - vendor specific output buffer drivers
  - TTL drivers
  - logic-to-analog converters
  - analog-to-logic simulator converters

- Integrated Circuit designers can use these elements to reduce design time, especially when designing filters and signal processors.

Behavioral elements use an arbitrary algebraic equation, as a transfer function to either a voltage (E) or current (G) source. This function can include:

- nodal voltages
- element currents
- time
- other parameters, which you define

A good example of this is a VCO, where `control` is the input voltage node, and `osc` is the oscillator output:

```
Evco osc 0 VOL='voff+gain*\\
     SIN(6.28*freq*(1+V(control))*TIME)'
```

You can use subcircuits to encapsulate a function.

- If you split the function definition from the use, you create a hierarchy.
- If you pass parameters into the subcircuit, you create a parameterized cell.
- If you create a full transistor cell library, and a behavioral representation library, you can include mixed-signal functions within HSPICE.

You can use the built-in `OPTIMIZE` function to calibrate the behavioral elements from a full transistor circuit.

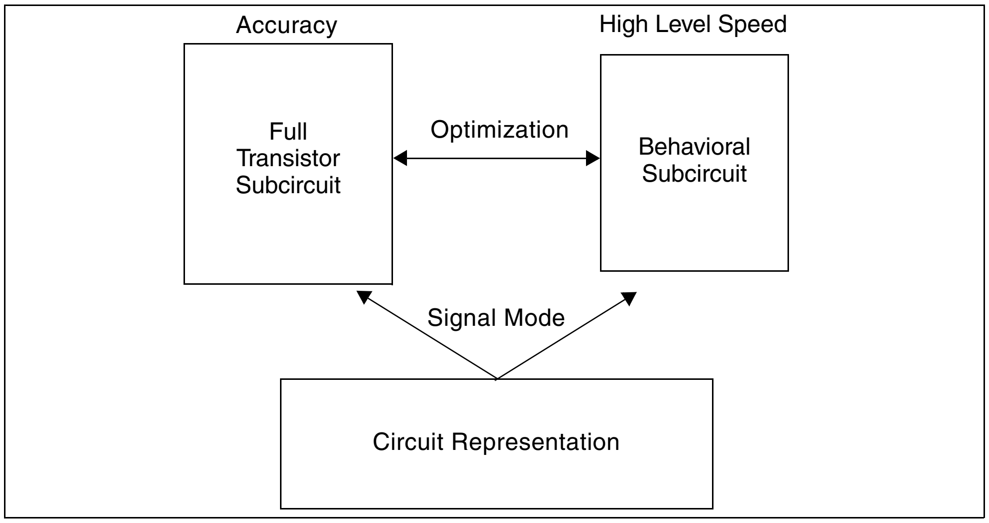


Figure 126 Netlisting by Signal Mode

Controlled Sources

Controlled sources model both analog and digital circuits, at the behavioral level. This reduces simulation times for mixed signals, and models system-level operations. Controlled sources also model gate-switching action, for behavioral modeling of digital circuits. For analog behavioral modeling, you can program the controlled sources as mathematical functions. These functions can be either linear or non-linear, depending on other nodal voltages and branch currents.

Libraries

The Discrete Device Library contains standard industry IC components. You can use this library to model board-level designs that contain any of the following:

- transistors
- diodes
- opamps
- comparators
- converters
- IC pins
- printed circuit board traces
- coaxial cables

You can also model drivers and receivers, to analyze transmission line effects, power line noise, and signal line noise.

Voltage and Current Controlled Elements

HSPICE or HSPICE RF provides two voltage-controlled and two current-controlled elements, known as E, F, G, and H-elements. For a description of these elements, see Chapter 9, Sources and Stimuli.

Modeling with Digital Behavioral Components

This section shows how to model, using digital behavioral components.

Behavioral AND and NAND Gates

The following example uses a G Element to model a 2-input AND gate. An E Element models a two-input NAND gate. Figure 127 shows the resulting waveforms. This example is located in the following directory:

$installdir/demo/hspice/behave/behave.sp

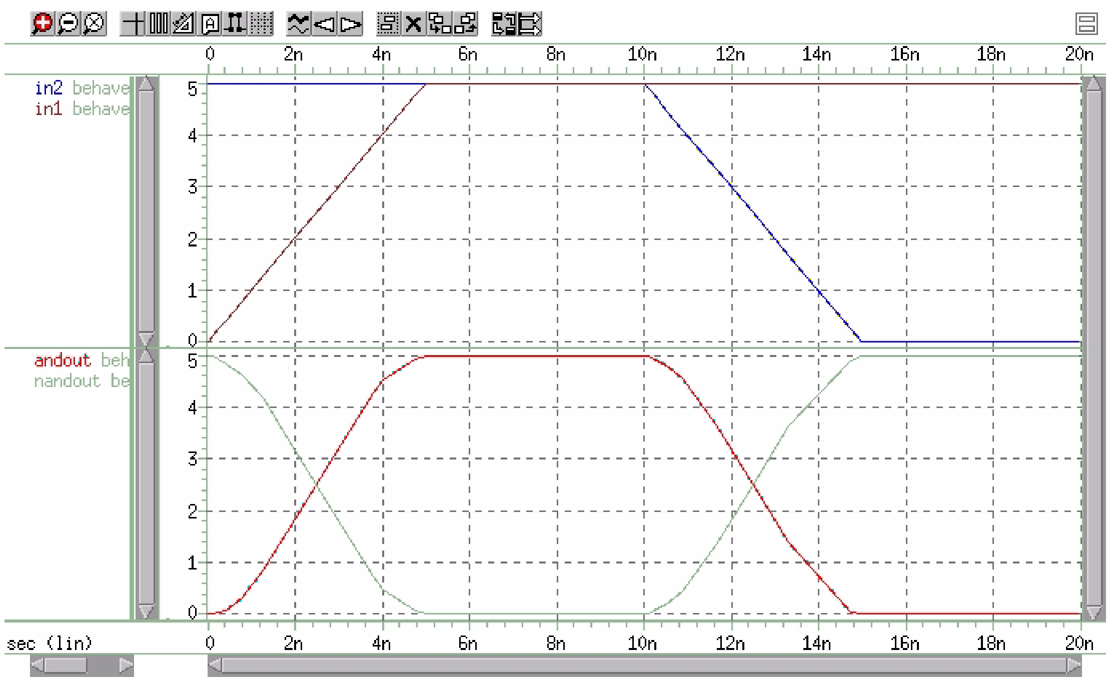


Figure 127 NAND/AND Gates

Behavioral D-Latch

This example uses one input NAND gates, and `NPWL`/`PPWL` functions, to model a D flip-flop.

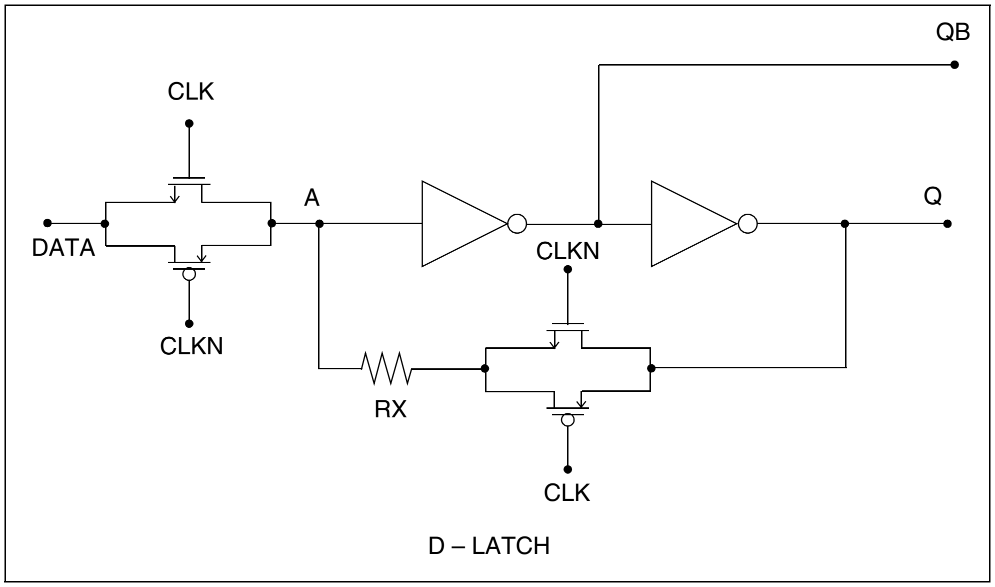


Figure 128 D-Latch

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/dlatch.sp

The file contains the following examples:

- Waveforms
- Subcircuit Definitions for Behavioral N-Channel MOSFET
- Behavioral P-Channel MOSFET

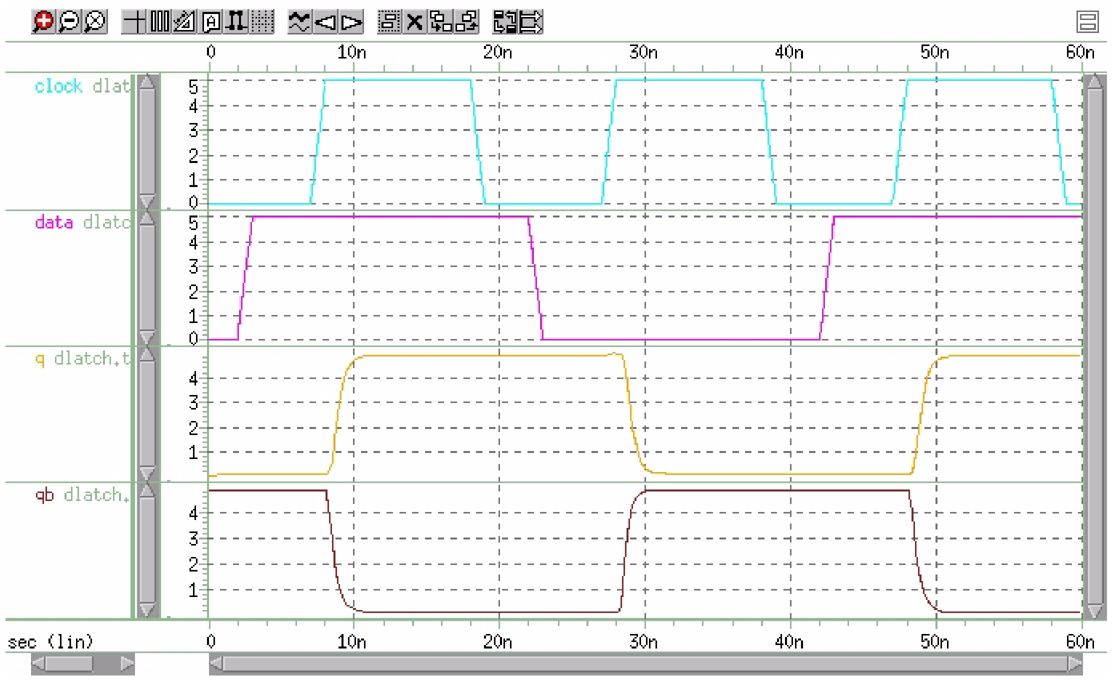


Figure 129 D-Latch Response

Behavioral Double-Edge Triggered Flip-Flop

This example uses the D_LATCH subcircuit from the previous example, and several NAND gates, to model a double-edged, triggered flip-flop.

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/det_dff.sp

- Main Circuit
- Subcircuit Definitions

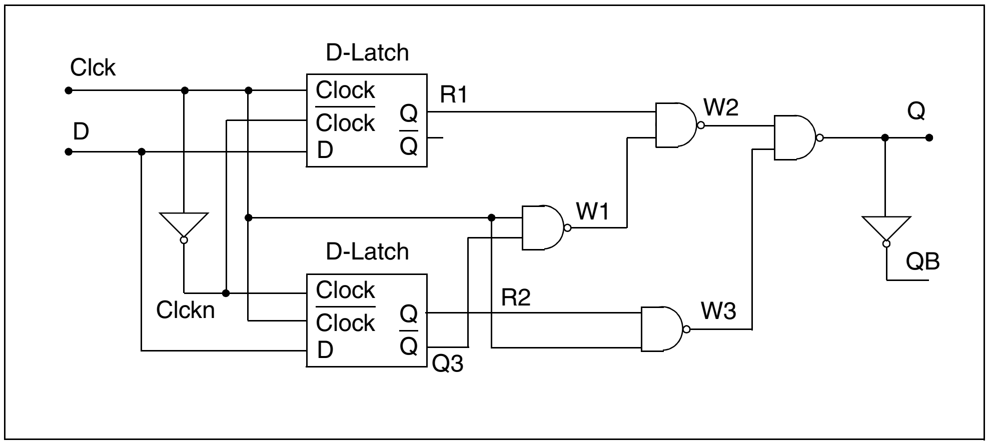


Figure 130 Double-Edge Triggered Flip-Flop

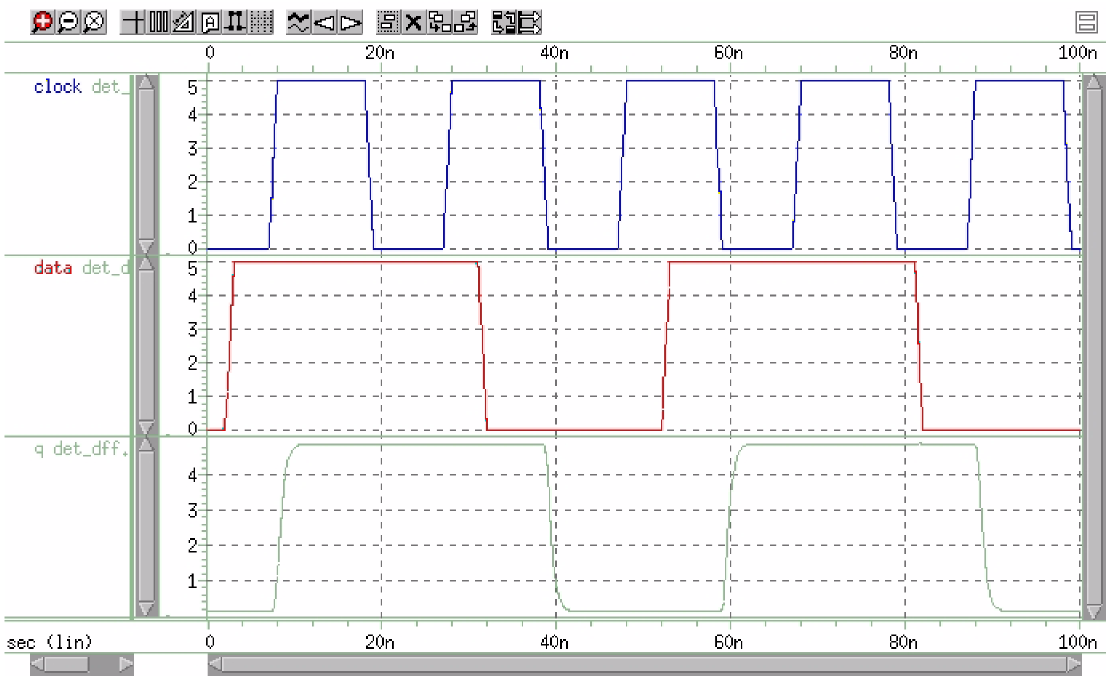


Figure 131 Double Edge Triggered Flip-Flop Response

Calibrating Digital Behavioral Components

This section describes how to calibrate, using digital behavioral components.

Building Behavioral Lookup Tables

The following simulation demonstrates an ACL family output buffer, with 2ns delay, and 1.8ns rise and fall time. It also shows ground and VDD supply currents, and internal ground bounce due to the package.

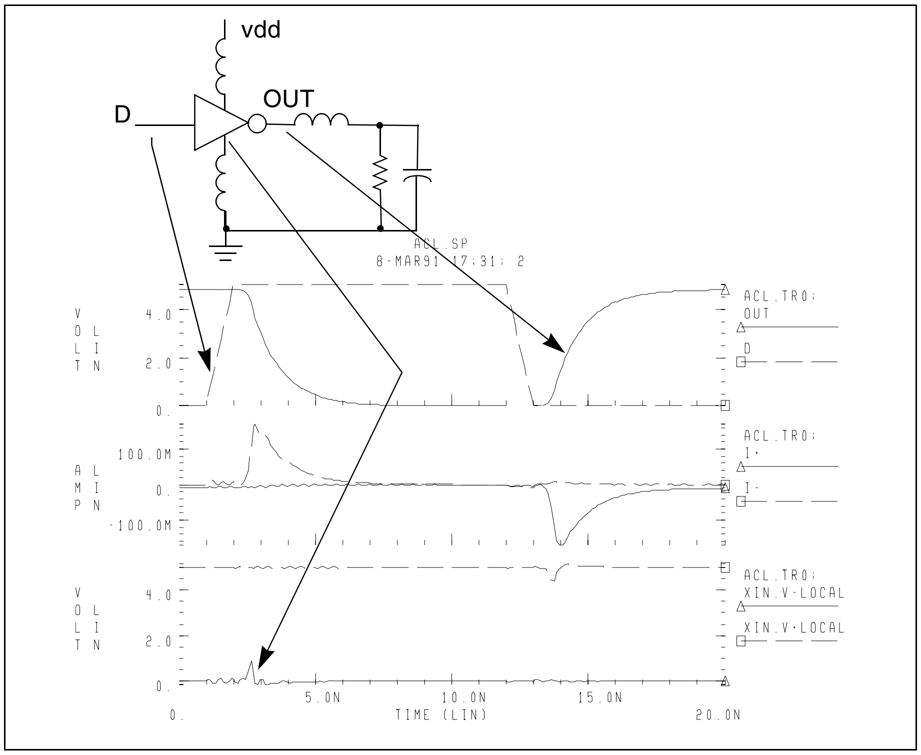


Figure 132 ACL Family Output Buffer

The following commands automatically measure the datasheet quantities, such as TPHL, risetime, maximum power dissipation, and ground bounce.

```
.MEAS tphl trig v(D) val='.5*vdd' rise=1
+ targ v(out) val='.5*vdd' fall=1
.MEAS risetime trig v(out) val='.1*vdd' rise=1
+ targ v(out) val='.9*vdd' rise=1
.MEAS max_power max power
.MEAS bounce max v(xin.v_local)
```

The inverter consists of capacitors, diodes, one-dimensional lookup table MOSFETs, and a special low-pass delay element. A property of the low-pass delay element, attenuates pulses that are narrower than the delay value.

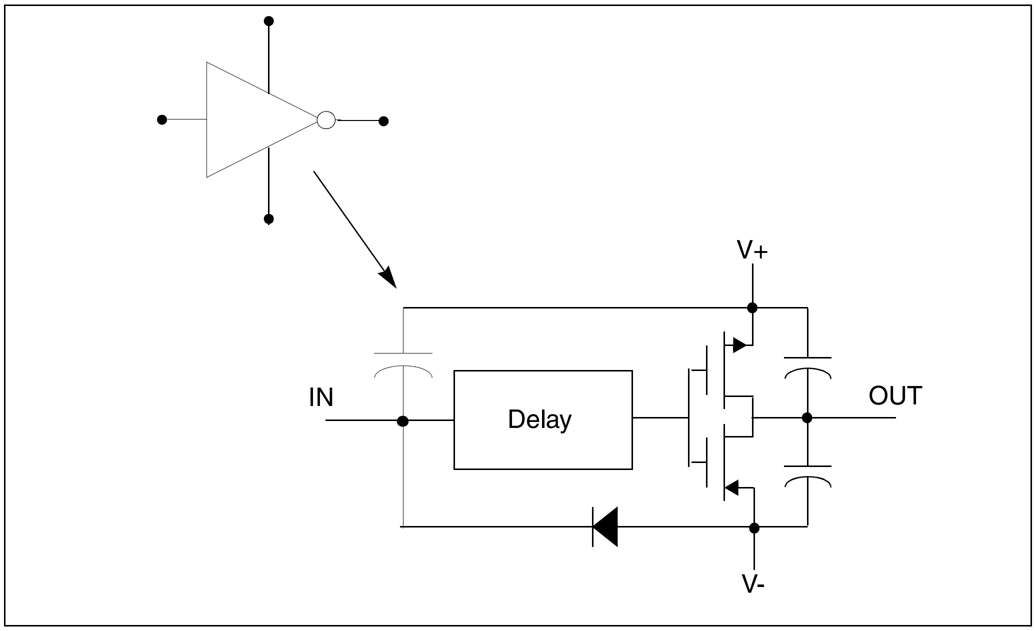


Figure 133 Inverter

Subcircuit Definition

```
.subckt inv in out v+ v-
cout+ out_l v+ 2p
cout- out_l v- 2p
xmp out_l inx v+ pmos
xmn out_l inx v- nmos
e inx v- delay in v- td=1n
din v- in dx
.model dx d cjo=2pf
chi in v+ .5pf
.ends inv
```

One-dimensional lookup tables represent the behavioral MOSFETs.

Behavioral N-Channel MOSFET

The following example is a Drain Gate source.

```
.subckt nmos 1 2 3
gn 3 1 VCR npwl(1) 2 3 scale=0.008
* VOLTAGE   RESISTANCE
+ 0.    495.8840g
+ 200.00000m    456.0938g
+ 400.00000m    141.6902g
+ 600.00000m    7.0624g
+ 800.00000m    258.9313meg
+ 1.00000    6.4866meg
+ 1.20000    842.9467k
+ 1.40000    21.6882k
+ 1.60000   170.8367k
+ 1.80000    106.4944k
+ 2.00000    72.7598k
+ 2.20000    52.4632k
+ 2.40000    38.5634k
+ 2.60000    8.8056k
+ 2.80000    5.2543k
+ 3.00000    4.3553k
+ 3.40000    3.4950k
+ 3.80000    2.0534k
+ 4.20000    2.7852k
+ 4.60000    2.5k
+ 5.0    2.3k
.ends nmos
```

The preceding example is a voltage-versus-resistance table. It shows, for example, that the resistance at 5 V is 2.3 kohms.

Creating a Behavioral Inverter Lookup Table

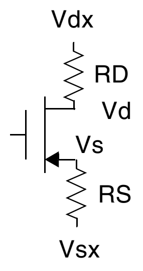
You can create an inverter lookup table in three simple steps:

1. Simulate an actual transistor level inverter, using a DC sweep of the input.
2. Print the V/I output, for the output pullup and pulldown transistors.
3. Copy the printed output into the volt lookup table element, for the controlled resistor.

The following test file, inv_vin_vout.sp, calculates `RN` (the effective pulldown resistor transfer function) and `RP` (the pullup transfer function).

- `RN` is calculated as Vout/I(mn), where mn is the pulldown transistor.
- `RP` is calculated as (VCC-Vout)/I(mp), where mp is the pullup transfer function.

The actual calculation uses a more accurate method, to obtain the series resistance of the transistor, as in Figure 134.



Rtot= (Vds-Vsx)/Ids

For greater accuracy:

Rtot= RD + RS + (vd-vs)/Ids
RD = 1/LV16(mn)
RS = 1/LV17(mn)
(vd-vs) = LX3(mn)
Ids = LX4(mn)

Figure 134 VIN versus VOUT

The first graph in Figure 135 shows `VIN` versus `VOUT`.

The second graph shows the computed transfer resistances (`RP` and `RN`), as a function of `VIN`.

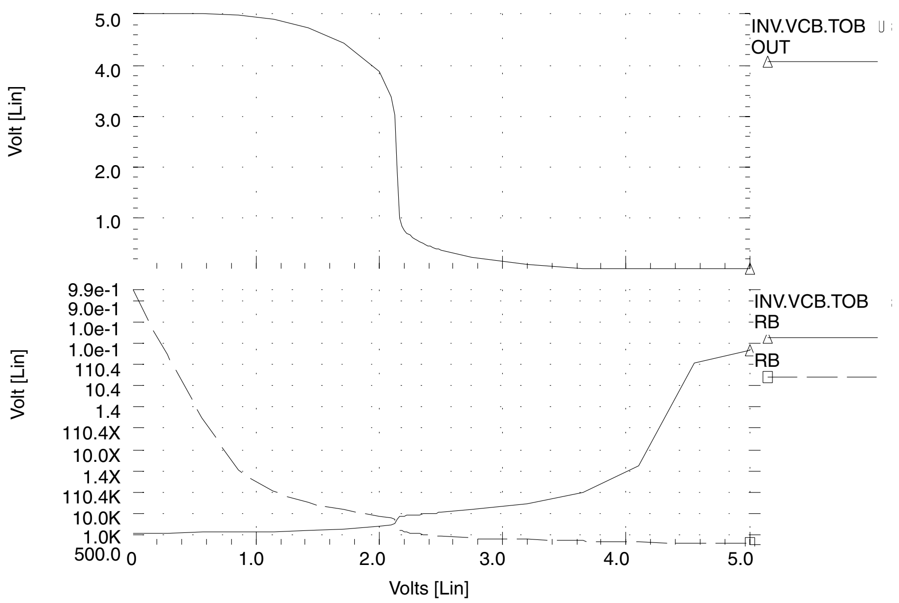


Figure 135 RP and RN as a Function of VIN

The HSPICE file used to calculate RP and RN is located in the following directory:

$installdir/demo/hspice/behave/inv_vin_vout.sp

Optimizing Behavioral CMOS Inverters

To calibrate behavioral models, run HSPICE on the full transistor version of a cell. Then optimize the behavioral model to this data.

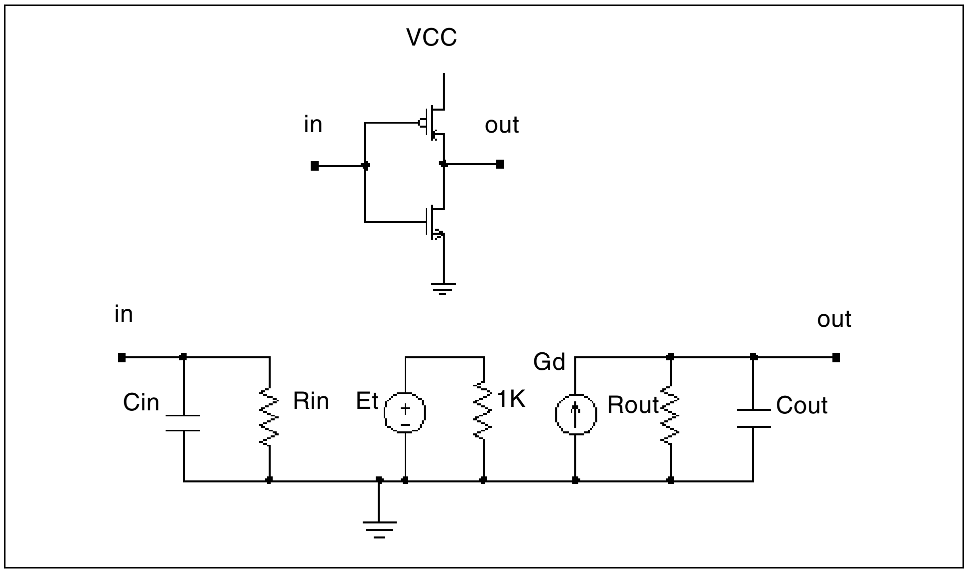


Figure 136 CMOS Inverter and its Equivalent Circuit

In this example, HSPICE uses the LEVEL 3 MOSFET model to simulate the CMOS inverter.

1. To obtain the input and output resistances, HSPICE performs a `.TF` transfer function analysis (`.TF V(out) Vin`).
2. To obtain the transfer function table of the inverter, HSPICE performs the DC analysis, and sweeps the input voltage (`.DC Vin 0 5 .1`).
3. HSPICE uses this table, in the PWL element, to represent the transfer function of the inverter.
4. A voltage-controlled PWL capacitance adjusts the rise and fall time of the inverter, in the equivalent circuit, across the output resistance.

5. The delay element obtains the propagation delay, across the output RC circuit.
6. HSPICE uses the inverter in a ring oscillator, to adjust the input capacitance.
7. HSPICE uses optimization analysis for all adjustments in this example. The data file and the results are shown.

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/invb_op.sp

The `invb_op.sp` file contains the following sections:

- Subcircuit Definition
- Inverter Using Model
- Optimization Results
- Optimization Completed
- Optimized Parameters OPTINV
- Optimize Results Measure Names and Values

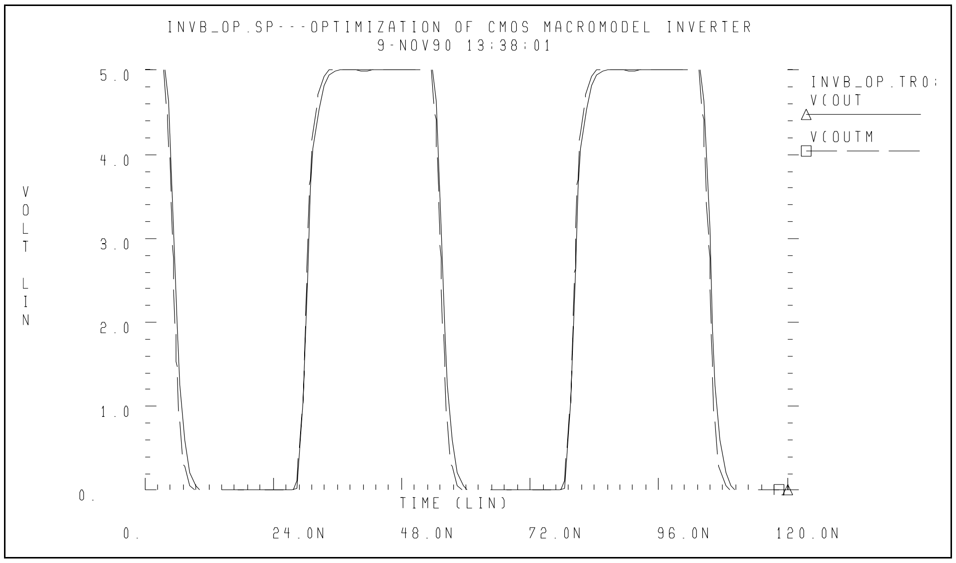


Figure 137 CMOS Inverter Response

Optimizing Behavioral Ring Oscillators

To optimize behavioral ring oscillator performance, review the examples in this section.

Example Five-Stage Ring Oscillator

This example is located in the following directory:

$installdir/demo/hspice/behave/ring5bm.sp

The ring5bm.sp file also contains the results of the five-stage ring oscillator example.

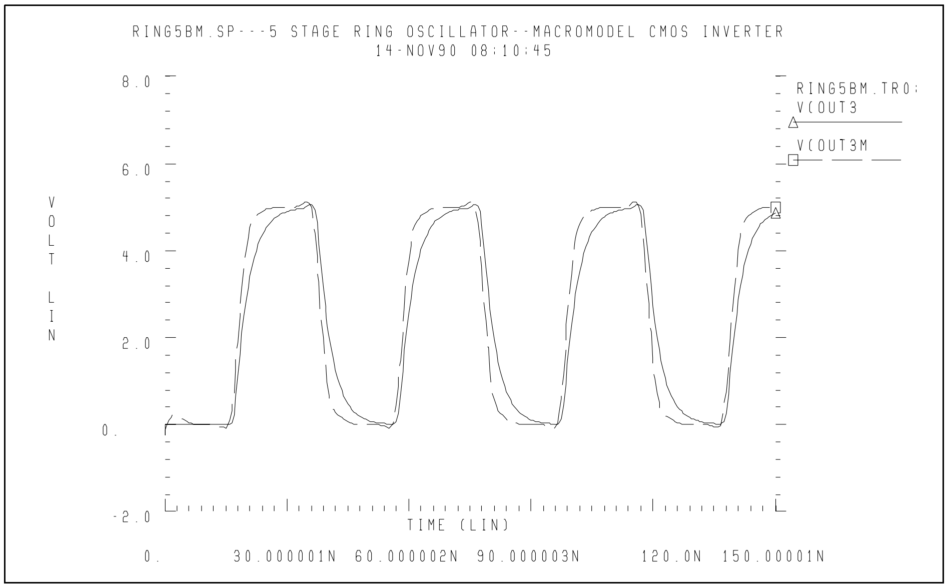


Figure 138 Ring Oscillator Response

Analog Behavioral Elements

The following components are examples of analog behavioral building blocks. Each component demonstrates a basic HSPICE feature:

- integrator: ideal op-amp E-element source
- differentiator: ideal op-amp E-element source
- ideal transformer: ideal transformer E-element source

- AM modulator: algebraic G-element source
- data sampler: algebraic E-element source

HSPICE uses an ideal op-amp to model the integrator circuit, and a VCVS to adjust output voltage. The following equation calculates output of the integrator:

$$Vout = -\frac{gain}{Ri \cdot Ci} \cdot \int_0^t Vin \cdot dt + Vout(0)$$

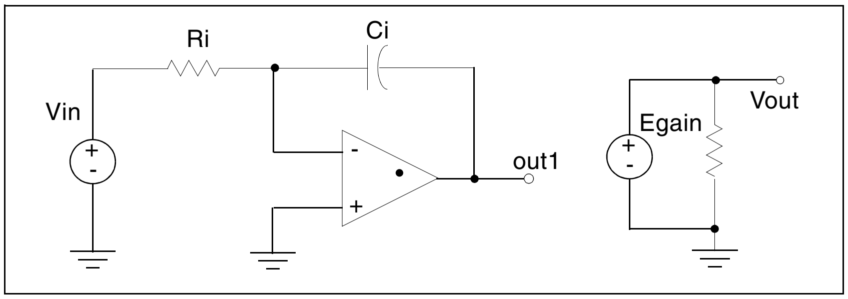


Figure 139 Integrator

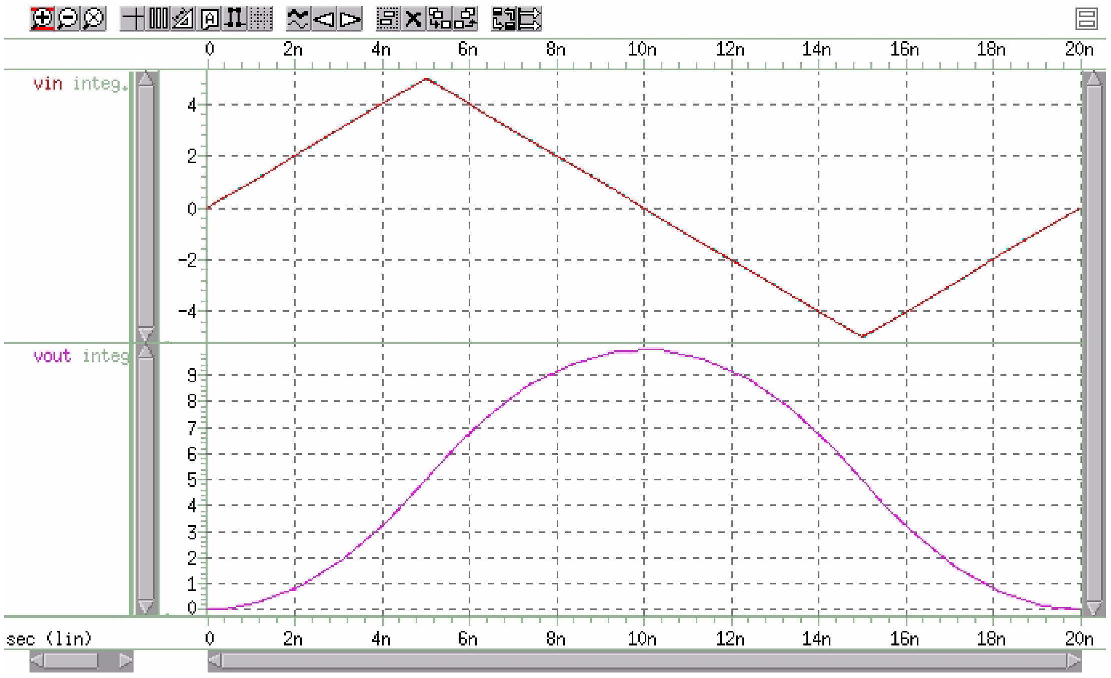


Figure 140 Response of Integrator to a Triangle Waveform

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/integ.sp

The integ.sp file also contains the following sections:

- Control and options
- Subcircuit definition
- Circuit

Behavioral Differentiator

HSPICE uses an ideal op-amp to model a differentiator, and a VCVS to adjust the magnitude and polarity of the output. The following equation calculates the differentiator response:

$$Vout = -gain \cdot Rd \cdot Cd \cdot \frac{d}{dt}Vin$$

For a high-frequency signal, the output of a differentiator can overshoot the edges. To smooth this out, you can use a simple RC filter.

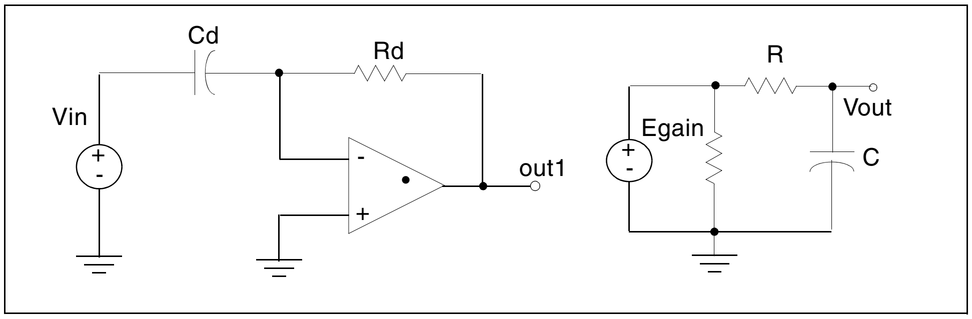


Figure 141 Differentiator

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/diff.sp

The diff.sp file also contains the following sections:

- Control and Options
- Subcircuit Definition
- Circuit

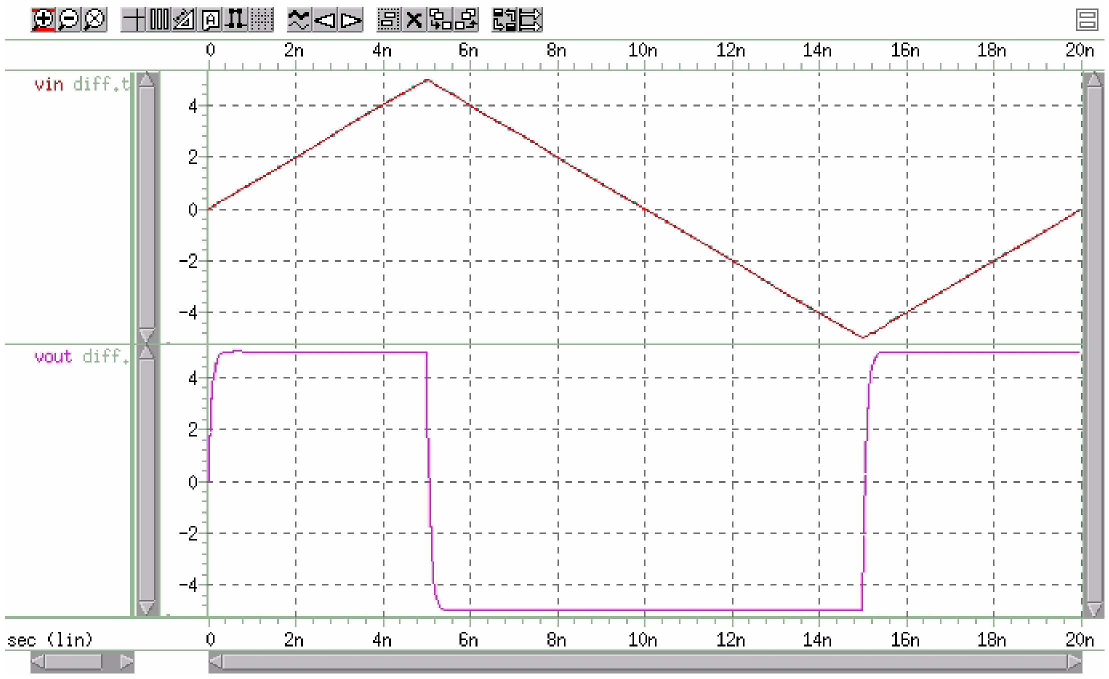


Figure 142 Response of a Differentiator to a Triangle Waveform

Ideal Transformer

The following example uses the ideal transformer to convert 8-ohms impedance of a loudspeaker, to 800 ohms impedance. This is a proper load value for a power amplifier, $Rin = n^2 \bullet RL$.

```
MATCHING IMPEDANCE BY USING IDEAL TRANSFORMER
E OUT 0 TRANSFORMER IN 0 10
RL OUT 0 8
VIN IN 0 1
.OP
.END
```

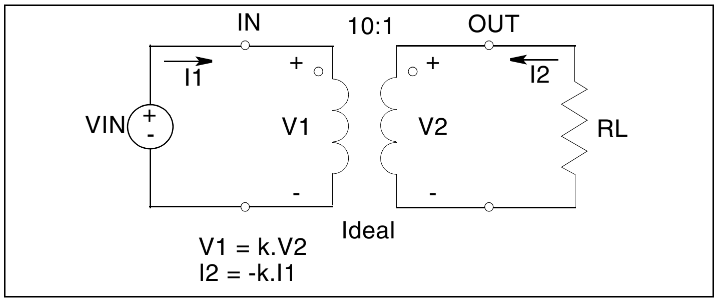


Figure 143 Ideal Transformer Example

Behavioral Amplitude Modulator

This example, which uses a G-element as an amplitude modulator with a pulse waveform carrier, is located in the following directory:

$*installdir*/demo/hspice/behave/amp_mod.sp

See also AM Modulation on page 429.

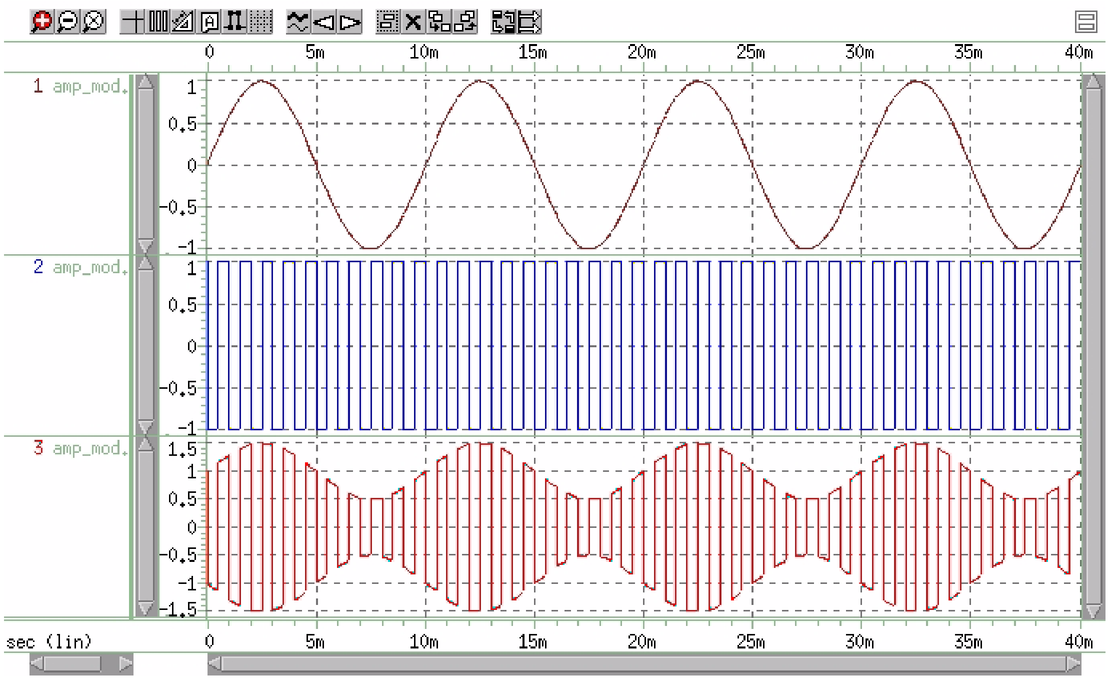


Figure 144 Amplitude Modulator Waveforms

Behavioral Data Sampler

An example of sampling behavioral data is located in the following directory:

$installdir/demo/hspice/behave/sampling.sp:

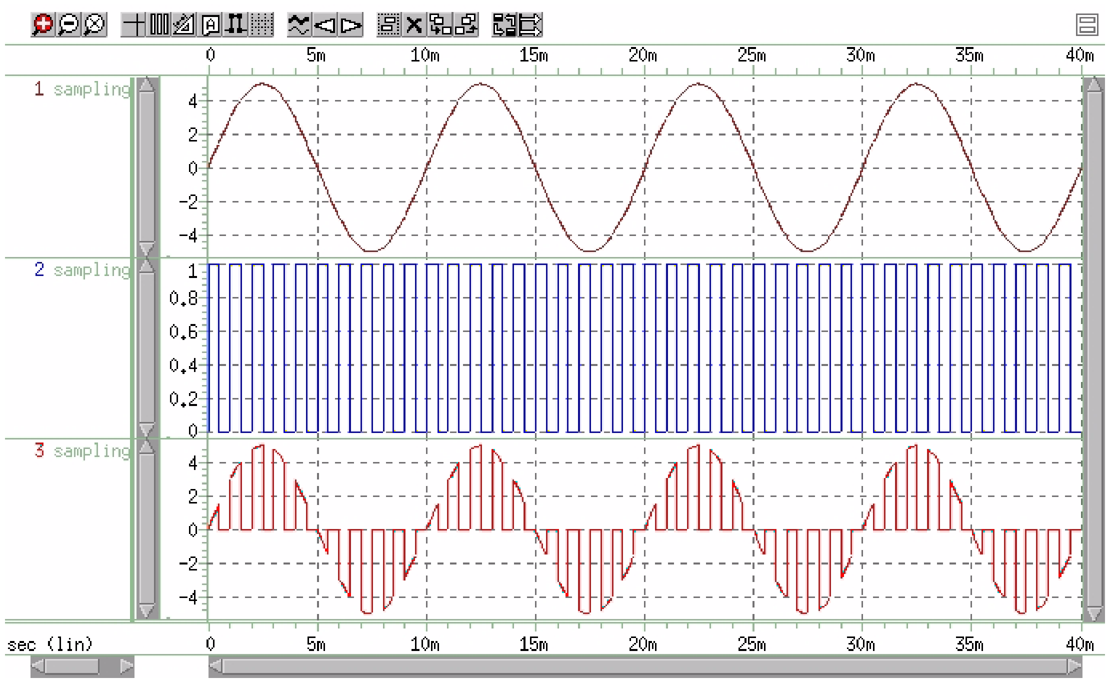


Figure 145 Sampled Data

Op-Amps, Comparators, and Oscillators

This section describes the use of op-amps, comparators, and oscillators, when you perform HSPICE simulation.

741 Op-Amp from Controlled Sources

To model the µA741 operational amplifier, use PWL controlled sources. A piecewise linear CCVS (source “h”) limits the output to ±15 V.

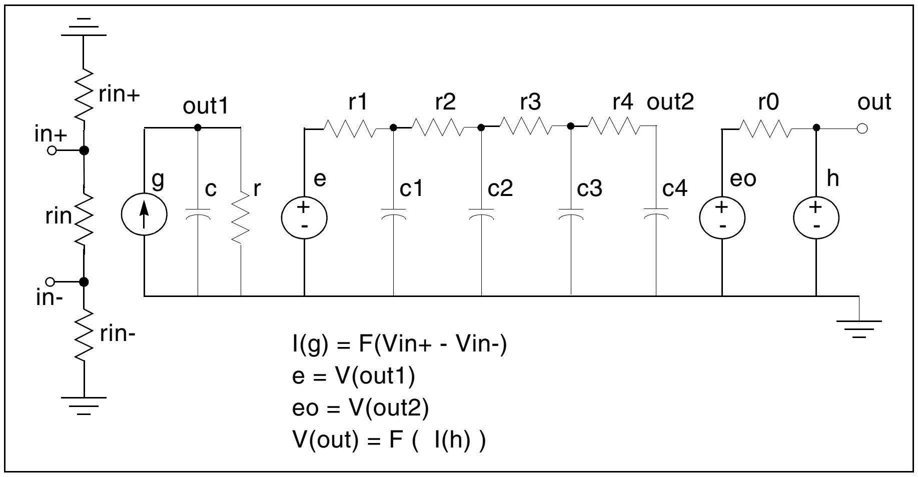


Figure 146 Operational Amplifier

Example

This example is located in the following directory:

$*installdir*/demo/hspice/behave/op_amp.sp

The op_amp.sp file also contains the following sections:

- Main Circuit
- RC Circuit With Pole At 9 MHz
- Output Limiter to 15 V

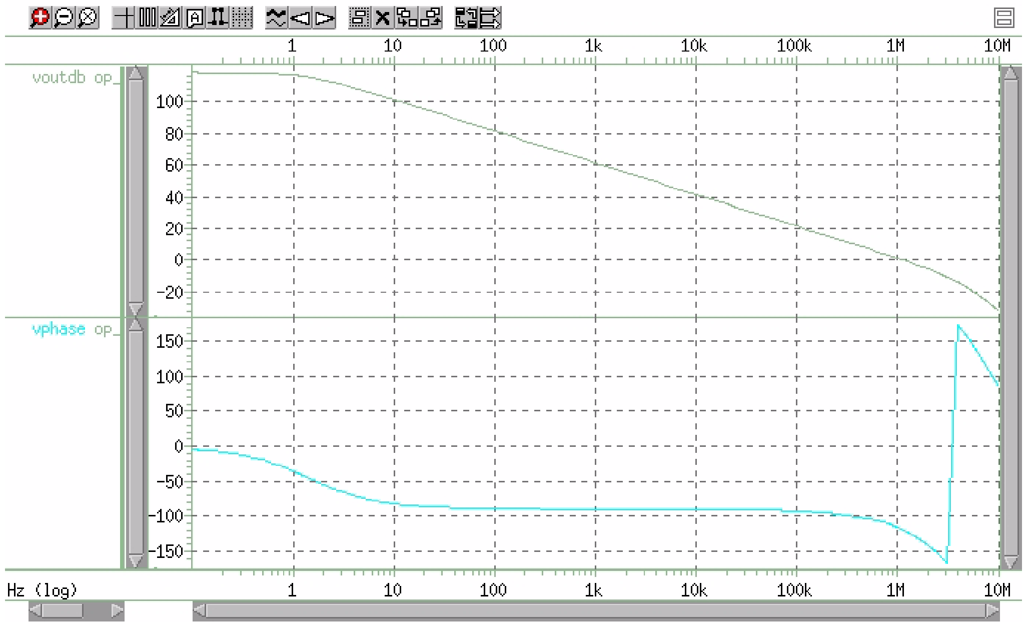


Figure 147 AC Analysis Response

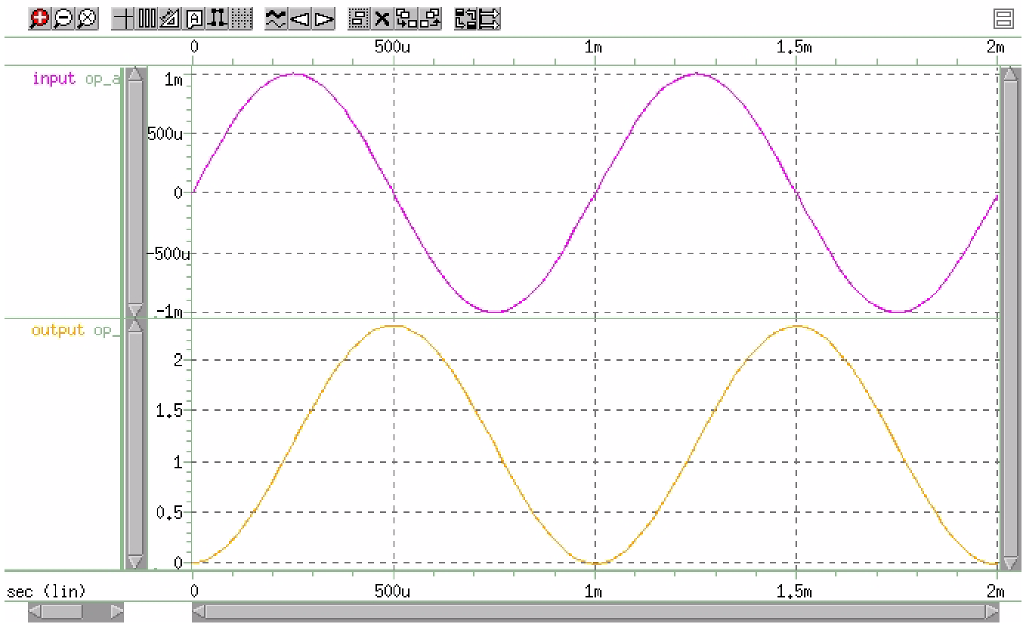


Figure 148 Transient Analysis Response

Inverting Comparator with Hysteresis

A piecewise-linear VCVS models an inverting comparator.

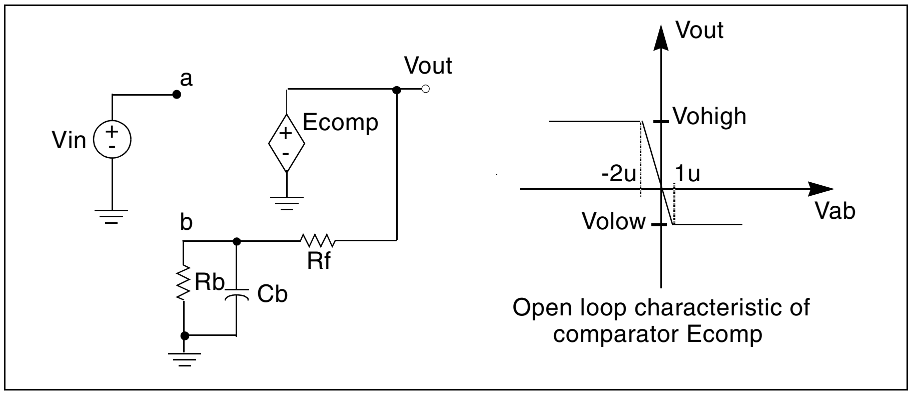


Figure 149 Inverting Comparator with Hysteresis

Two reference voltages correspond to the volow and vohigh voltages of Ecomp:

$$Vreflow = \frac{Volow \cdot Rb}{Rb + Rf} \qquad Vrefhigh = \frac{Vohigh \cdot Rb}{Rb + Rf}$$

When Vin exceeds Vrefhigh, the Vout output changes to Volow. For Vin values less than Vreflow, the output changes to Vohigh.

An example is located in the following directory:

$installdir/demo/hspice/behave/compar.sp

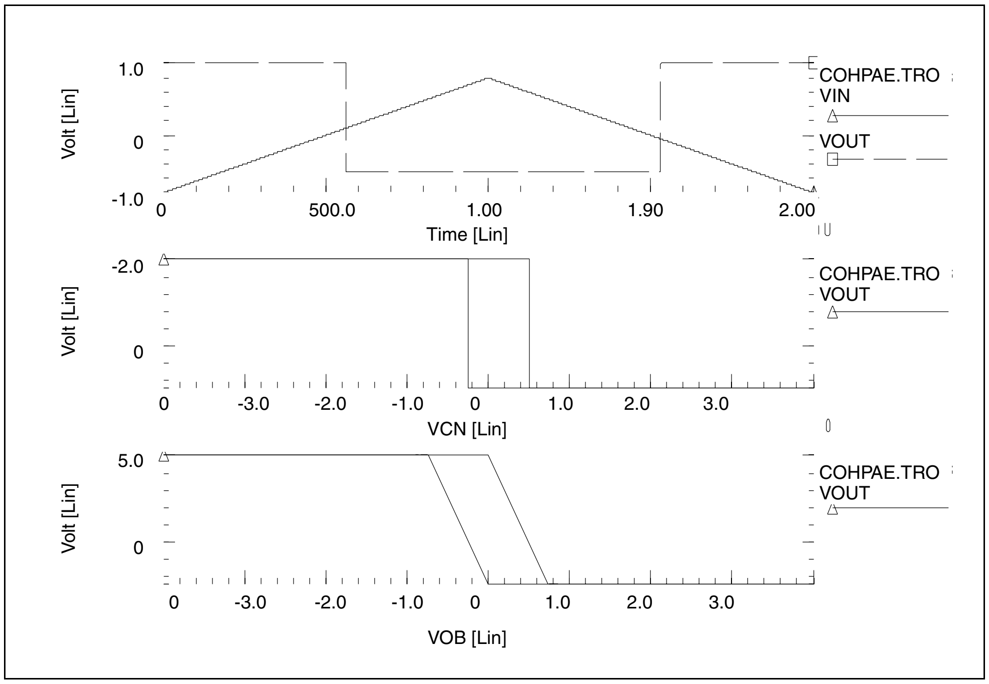


Figure 150 Response of Comparator

Voltage-Controlled Oscillator (VCO)

In this example, a one-input NAND (functioning as an inverter) models a five-stage ring oscillator. PWL capacitance switches the load capacitance of this inverter from 1pF to 3 pF. As the simulation results indicate, the oscillation frequency decreases, as the load capacitance increases.

Example

This example is based on demonstration netlist vcob.sp, which is available in directory $<installdir>/demo/hspice/behave. This file also contains a sample subcircuit definition.

```
vcob.sp voltage controlled oscillator using pwl functions
.option post
.global ctrl
.tran 1n 100n
.ic v(in)=0 v(out1)=5
.probe tran v(in)
x1 in out1 inv
x2 out1 out2 inv
x3 out2 out3 inv
x4 out3 out4 inv
x5 out4 in inv
vctrl ctrl 0 pwl(0,0 35n,0 40n,5)
*
* macro definitions
*
.subckt inv in out rout=1k
gcout out 0 pwl(1) ctrl 0 level=2 delta=.01
+ 4.5 1p
+ 4.6 3p
rout out 0 rout
gn 0 out nand(1) in 0 scale='1.0k/rout'
+ 0. 5.00ma
+ 0.25 4.95ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.20ma
+ 5.0 0.05ma
.ends inv
*

.end
```

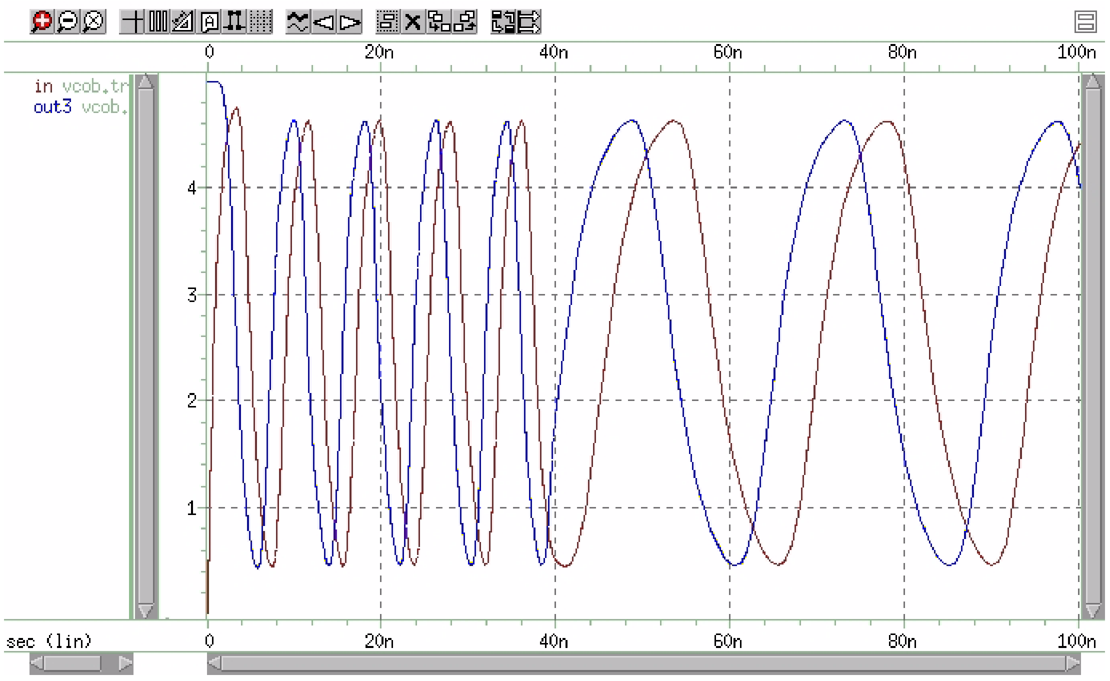


Figure 151 Voltage Controlled Oscillator Response

LC Oscillator

The initial capacitor charge is 5 V. The value of capacitance is the function of voltage, at node 10. The capacitance value becomes four times higher, at the t2 time. The following equation calculates the frequency of this LC circuit:

$$freq = \frac{1}{6.28 \cdot \sqrt{L \cdot C}}$$

At the t2 time, the frequency must be halved. The amplitude of oscillation depends on the condition of the circuit, when the capacitance value changes.

The stored energy is:

$$E = (0.5 \cdot C \cdot V^2) + (0.5 \cdot L \cdot I^2)$$

$$E = 0.5 \cdot C \cdot Vm^2,\ I = 0 \qquad E = 0.5 \cdot L \cdot Im^2,\ V = 0$$

At the t2 time, when V=0, if C changes to A $\cdot$ C, then:

$$0.5 \cdot L \cdot Im^2 = 0.5 \cdot Vm^2 = 0.5 \cdot (A \cdot C) \cdot Vm'^2$$

and from the above equation:

$$Vm' = \frac{Vm}{\sqrt{A}}$$

$$Qm' = \sqrt{A} \cdot Vm$$

The second condition that HSPICE considers is when V=Vin, if C changes to A· C, then:

$$Qm = Qm'$$

$$C \cdot Vm = A \cdot C \cdot Vm'$$

$$Vm' = \frac{Vm}{A}$$

Therefore, HSPICE modifies the voltage amplitude, between Vm/sqrt(A) and Vm/A, depending on the circuit condition when the circuit switches. This example tests the `CTYPE=0` and `1` results. The result for `CTYPE=1` must be correct because capacitance is a function of voltage at node 10, not a function of the voltage across the capacitor itself.

Example

This example is based on demonstration netlist *calg2.sp*, which is available in directory $*installdir*/demo/hspice/behave:

```
* in this example the ctype 0 and 1 is tested. the result for
* ctype=1 must be correct because capacitance is function of
* voltage at node 10, not voltage across itself.
*
.option post
.ic v(1)=5 v(2)=5
c1 1 0 c='1e-9*v(10)' ctype=1
l1 1 0 1m
*
c2 2 0 c='1e-9*v(10)' ctype=0
l2 2 0 1m
*
v10 10 0 pwl(0sec,1v t1,1v t2,4v)
r10 10 0 1
```

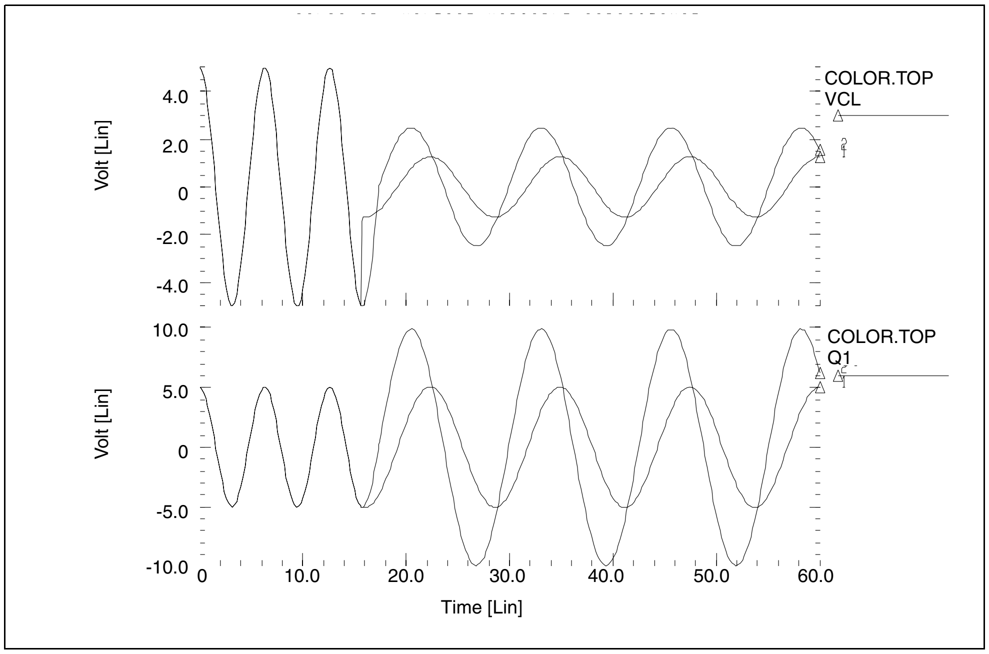


Figure 152 Correct Result Corresponding to CTYPE=1

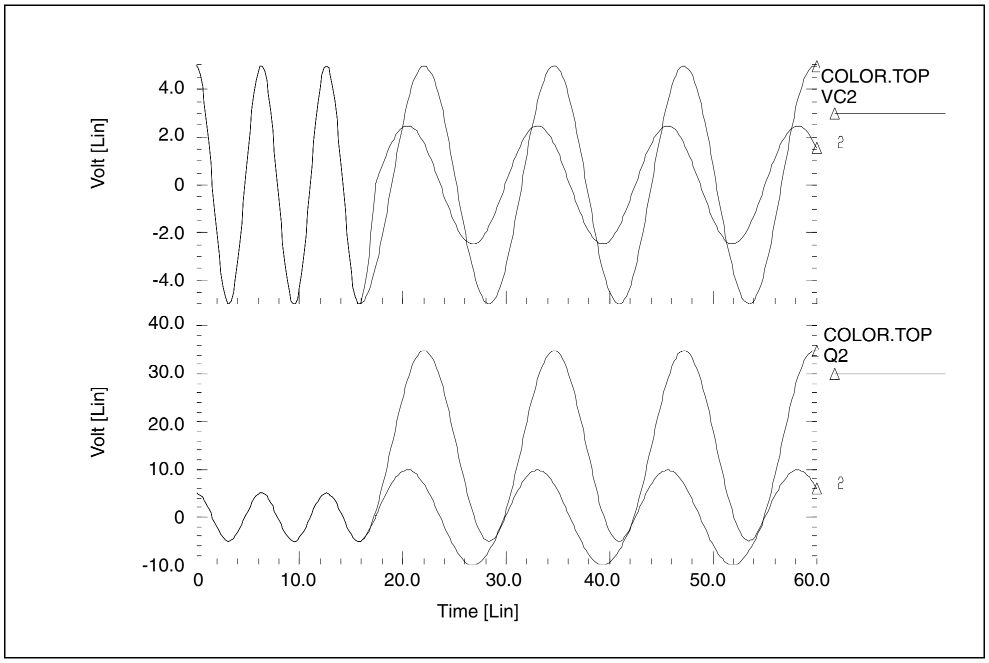


Figure 153 Incorrect Result Corresponding to CTYPE=0

Phase-Locked Loops (PLL)

The following sections explain material having to do with phase-locked loops.

Phase Detector, with Multi-Input NAND Gates

This circuit uses behavioral elements, to implement the inverters, with 2, 3, and 4 input NAND gates.

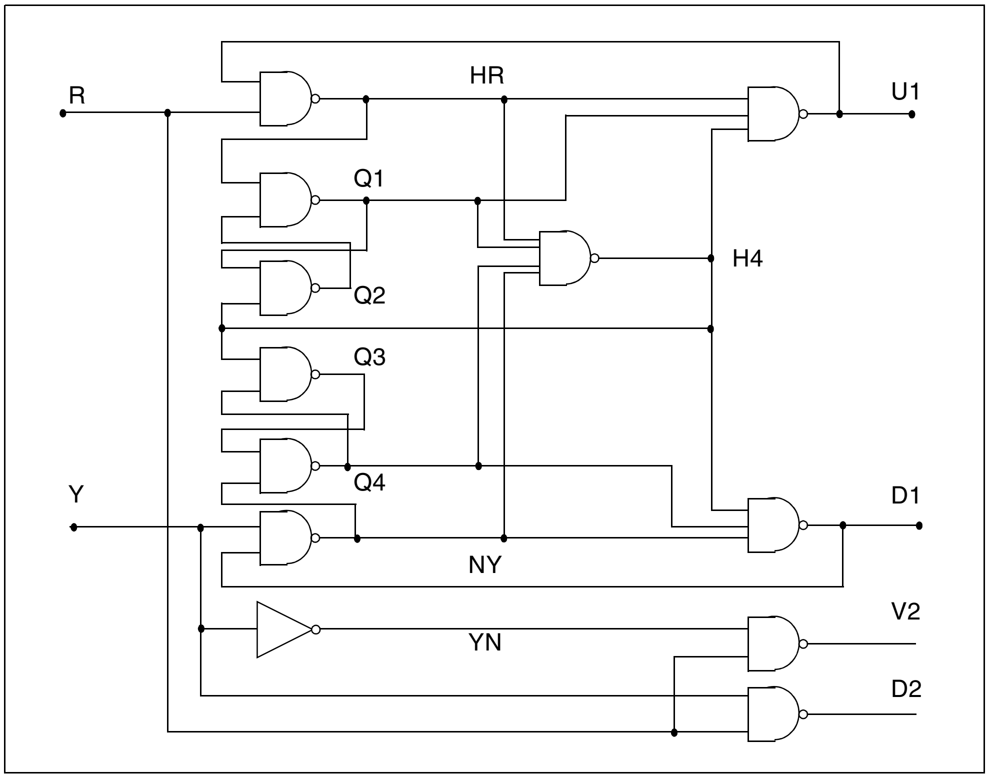


Figure 154 Phase Detector

Example

An example is located in the following directory:

$installdir/demo/hspice/behave/pdb.sp

This file also contains sample subcircuit definitions.

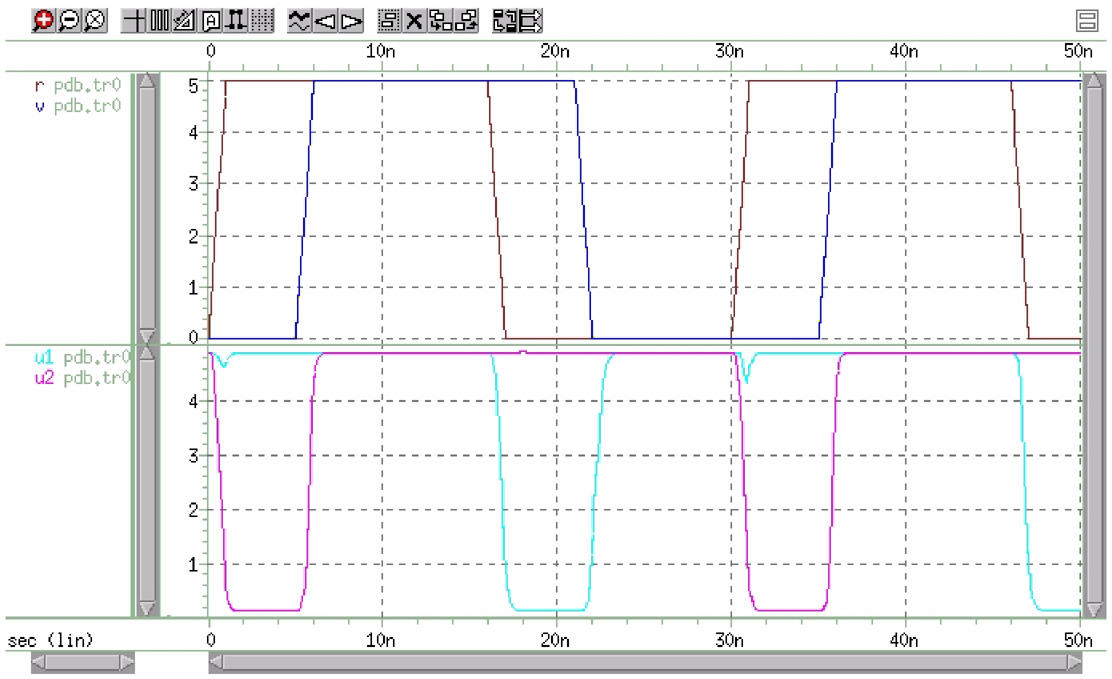


Figure 155 Phase Detector Response

Phase Locked Loop Modeling

A Phase-locked Loop (PLL) circuit synchronizes to an input waveform, within a selected frequency range. This returns an output voltage that is proportional to variations in the input frequency. It has three basic components:

- A voltage-controlled oscillator (VCO), which returns an output waveform that is proportional to its input voltage.
- A phase detector, which compares the VCO output to the input waveform, and returns an output voltage, depending on their phase difference.
- A loop filter, which filters phase detector voltage. Returns output voltage, which forms the VCO input (and external voltage output) of the PLL.

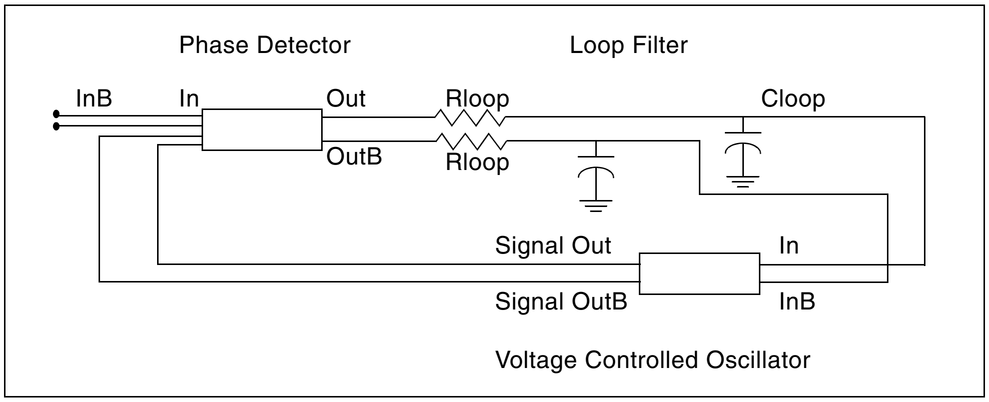


Figure 156 Behavioral Phase-Locked Loop

The PLL can be implemented using behavioral elements (Figure 156) or using bipolar transistors (Figure 157 on page 692 and Figure 158 on page 693).

The netlist for the behavioral PLL is the *pll_bvp.sp* file and the netlist for the full bipolar PLL is *pll.sp* file. The netlist for the full bipolar PLL contains the loop filter and the output circuit. Both netlist files are available in directory: $*installdir*/demo/hspice/behave.

The PLL transfer function shows a linear region of voltage vs. (periodic) time which is defined as the “lock” range.

The results of transient simulations (Figure 157) show minimal difference between implementations. However, run time statistics show that the behavioral model reduces simulation time, to one-third that of the full circuit.

If you use this PLL in a larger system simulation (for example, an AM tracking system), include the behavioral model. This model substantially reduces simulation run time, and still accurately represents the subcircuit.

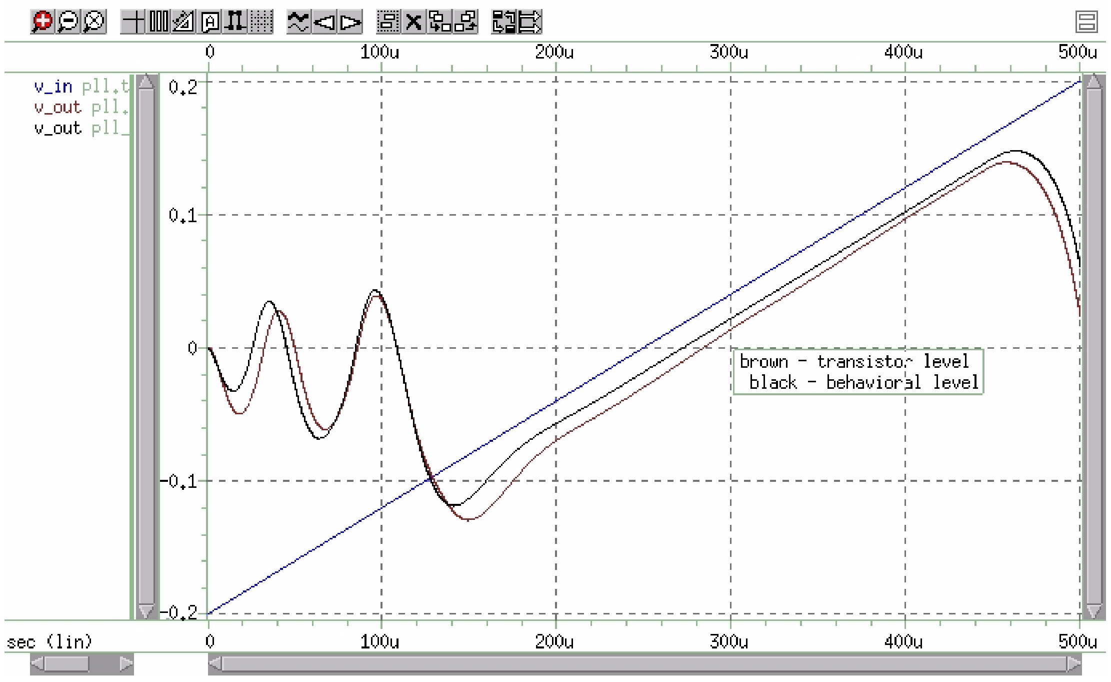


Figure 157 Behavioral (PLL_BVP Curve) vs. Bipolar (PLL_Curve) Simulation

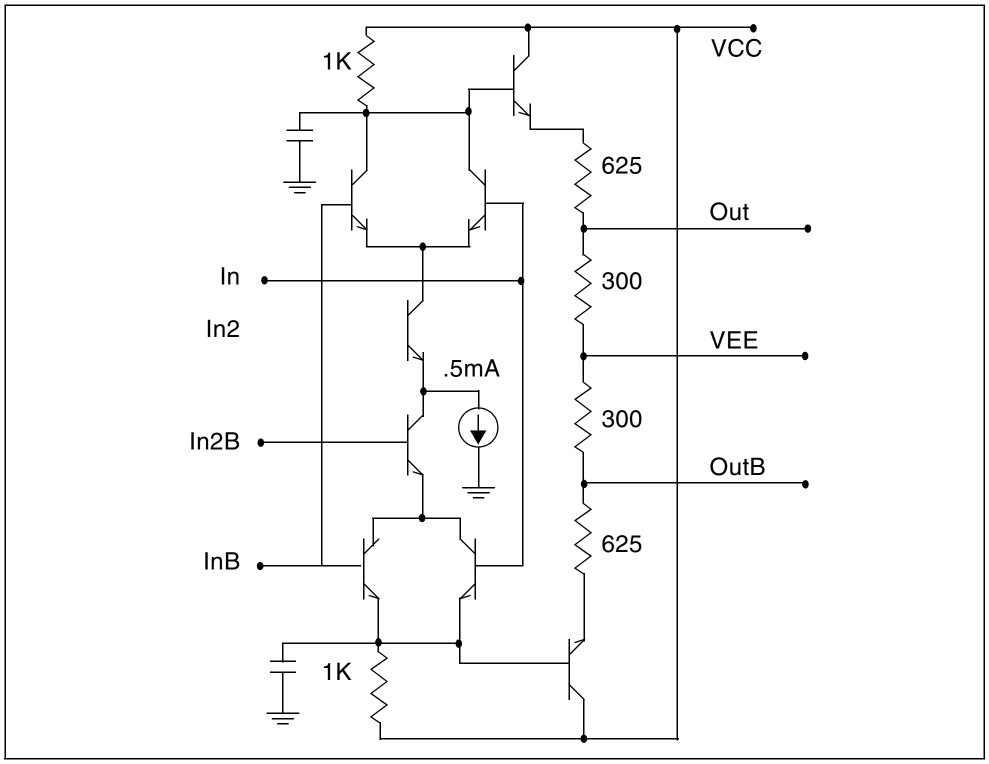


Figure 158 Bipolar Phase Detector

References

[1] Chua & Lin. *Computer Aided Analysis of Electronic Circuits*. Englewood Cliffs: Prentice-Hall, 1975, page 117. See also “SPICE2 Application Notes for Dependent Sources,” by Bert Epler, *IEEE Circuits & Devices Magazine*, September 1987.

29

Modeling Filters and Networks

Describes modeling filters and networks, including Laplace transforms.

When you apply Kirchhoff’s laws to circuits that contain energy storage elements, the result is simultaneous differential equations, in the time domain. A simulator must solve these equations, to analyze the circuit’s behavior. Solving any equation that is higher than first order can be difficult, and classical methods cannot easily solve some driving functions.

In both cases, to simplify the solution, you can use Laplace transforms. These transforms convert time domain equations, containing integral and differential terms, into algebraic equations in the frequency domain.

Transient Modeling

The Laplace transform method provides an easy way to relate a circuit’s behavior, in time and frequency-domains. This facilitates simultaneous work in those domains.

The algorithm that Synopsys HSPICE or HSPICE RF uses for Laplace and pole/zero transient modeling, offers better performance than the Fast Fourier Transform (FFT) algorithm. To invoke Laplace and pole/zero transient modeling, use a `LAPLACE` or `POLE` function call in a source element statement.

Laplace transfer functions are especially useful in top-down system design, when you use ideal transfer functions instead of detailed circuit designs. In HSPICE or HSPICE RF, you can also mix Laplace transfer functions, with transistors and passive components. Using this capability, you can model a system as the sum of the contributing ideal transfer functions. You can then progressively replace these functions with detailed circuit models, as they become available. Conventional uses of Laplace transfer functions include control systems, and behavioral models that contain non-linear elements.

Laplace transforms reduce the time needed to design and simulate large interconnect systems, such as clock distribution networks. You can use asymptotic waveform evaluation (AWE) and other methods, to create a Laplace transfer function model. The AWE model can use only a few poles to represent the large circuit. You can input these poles through a Laplace transform model, to closely approximate the delay and overshoot characteristics of many networks, in a fraction of the original simulation time.

You can use pole/zero analysis to help determine the stability of the design. You can use the `POLE` function in HSPICE or HSPICE RF when the poles and zeros of the circuit are specified, or you can use the `.PZ` statement (see the HSPICE Reference Manual: Commands and Control Options) to derive the poles and zeros from the transfer function.

Frequency response is an important analog circuit property. It is normally the ratio of two complex polynomials (functions of complex frequencies), with positive real coefficients. The form of frequency response can be either the locations of poles and zeros, or a frequency table.

The usual way to design complex circuits is to interconnect smaller functional blocks of known frequency responses, either in pole/zero or frequency table form. For example, to design a band-reject filter, you can interconnect a low-pass filter, a high-pass filter, and an adder. Study the function of the complex circuit, in terms of its component blocks, before you design the actual circuit. After you test the functionality of the component blocks, you can use these blocks as a reference in optimization techniques, to determine the value of the complex element.

Using G- and E-elements

This section describes how to use G- and E-elements (controlled behavioral sources).

Laplace Transform Function Call

Use the G- and E-elements as linear functional blocks, or as elements with specific frequency responses.

In the following equations, *H*(s) denotes the frequency response (also called the impulse response), where s is a complex frequency variable ($s = j2\pi f$). To obtain the frequency response, perform an AC analysis, and set `AC=1` in the input source (the Laplace transform of an impulse is 1). The following expression relates the input and output of the G- and E-elements, with specified frequency response:

$$Y(j2\pi f) = Hj(2\pi f) \cdot Xj(2\pi f)$$

where X is the input, Y is the output, and H is the transfer function, at the f frequency.

AC analysis uses the above relation, at any frequency, to determine the frequency response. For operating point and DC sweep analysis, the relation is the same, but the frequency is zero.

The transient analysis is more complicated than the frequency response. The output is a convolution of the input waveform, with the impulse response h(t):

$$y(t) = \int_{-\infty}^{t} x(\tau) \cdot h(t-\tau) \cdot d\tau$$

In discrete form, the output is:

$$y(k\Delta) = \Delta \sum_{m=0}^{k} x(m\Delta) \cdot h[(k-m) \cdot \Delta]\text{, k = 0, 1, 2, ...}$$

where you can obtain h(t) from H(f), using the inverse Fourier integral:

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{j2\pi ft} \cdot df$$

The following equation calculates the inverse discrete Fourier transform:

$$h(m\Delta) = \frac{1}{N \cdot \Delta} \sum_{n=0}^{N-1} H(f_n) \cdot e^{\frac{j2\pi nm}{N}}\text{, m = 0, 1, 2, ..., N-1}$$

where N is the number of equally-spaced time points, and Δ is the time interval or time resolution.

For the frequency response table form (FREQ) of the `LAPLACE` function, HSPICE uses a performance-enhanced algorithm, to convert H(f) to h(t). This algorithm requires N to be a power of 2. The following equation determines the f_n frequency point:

$$f_n = \frac{n}{N \cdot \Delta}, \quad \text{n = 0, 1, 2, ..., N-1}$$

where n > N/2 represents the negative frequencies. The following equation determines the Nyquist critical frequency:

$$f_c = f_{N/2} = \frac{1}{2 \cdot \Delta}$$

Because the negative frequency responses are the image of the positive responses, you need to specify only N/2 frequency points, to evaluate N time points of h(t). The larger the value of f_c is, the more accurate the transient analysis results are. However, for large f_c values, the Δ becomes smaller, and computation time increases.

The maximum frequency of interest depends on the functionality of the linear network. For example, in a low-pass filter, you can set f_c to the frequency at which the response drops by 60 dB (a factor of 1000).

$$|H(f_c)| = \frac{|H_{max}|}{1000}$$

After you select or calculate f_c, the following equation can determine Δ:

$$\Delta = \frac{1}{2 \cdot f_c}$$

The following equation calculates the frequency resolution:

$$\Delta f = f_1 = \frac{1}{N \cdot \Delta}$$

which is inversely proportional to the maximum time (NΔ), over which HSPICE evaluates h(t). Therefore, the transient analysis accuracy also depends on the frequency resolution, or the number of points (N).

You can specify the frequency resolution (`DELF`) and the maximum frequency (`MAXF`) in the G- or E-element statement. To calculate N, HSPICE or HSPICE RF uses 2 · `MAXF`/`DELF`. Next, HSPICE or HSPICE RF modifies N as a power of 2. The effective `DELF` is `2 ·` MAXF/N, to reflect the changes in N.

Element Statement Parameters

These keywords are common to the three forms described above:

- Laplace
- Pole-zero
- Frequency response table

Table 69 Element Statement Parameters

Parameter	Description
ACCURACY	Used only in G-element, with the frequency response table. 0: Default. This method generates more accurate results and achieves better performance. 1: The same as ACCURACY=0 2: The method used in release 2000.4 and before.
DELF, DELTA	Frequency resolution Δf. The inverse of `DELF` is the time window, over which HSPICE or HSPICE RF calculates h(t) from H(s). A smaller DELF value means more accurate transient analysis, and longer CPU time. The number of points (N) used to convert H(s) to h(t) is $N=2\cdot$ MAXF/DELF. Because N must be a power of 2, HSPICE or HSPICE RF adjusts the DELF value. The default is 1/TSTOP. In the G-element, with FREQ and ACCURACY = 0 or 1, to perform circular convolution for periodic input, HSPICE or HSPICE RF limits the period. To do this, HSPICE or HSPICE RF sets DELF = 1/T:T < TSTOP.
FREQ	Keyword to indicate that a frequency response table describes the transfer function. Do not use FREQ as a node name in a G- or E-element. This is not the same as the `FREQ` model parameter that plots symbol frequency in `.PRINT`/`.PROBE` statements (or the deprecated `.PLOT`/`.GRAPH` statements.
LAPLACE	Keyword to indicate that a Laplace transform function describes the transfer function. Do not use LAPLACE as a node name, on a G- or E-element.
LEVEL	Used only in elements with a frequency-response table. Set this parameter to 1, if the element is a high-pass filter.
M	G-element multiplier. This parameter represents M G-elements in parallel. Default is 1.

Table 69 Element Statement Parameters (Continued)

Parameter	Description
MAXF, MAX	Maximum, or the Nyquist critical frequency. The larger the MAXF value, the more accurate the transient results are, and the longer the CPU time. The default is $1024 \cdot DELF$. These parameters apply only when you also use the FREQ parameter.
POLE	Keyword to indicate that the pole and zero location describes the transfer function. Do not use POLE as a node name, on a G- or E-element.
SCALE	Element value multiplier.
TC1,TC2	First-order and second-order temperature coefficients. The default is zero. The temperature updates the SCALE:$SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$

Z Transform Function Call

Z transform functions are used for G- and E-elements (controlled

behavioral sources), which is similar to the Laplace form function (see Laplace Transform Function Call for more details).

H(z) denotes the frequency response, where z is a complex frequency variable.

Its value is: $z = e^{(j \cdot w)}$

where, $w = 2 \cdot Pl \cdot f/ fs$ and $fs = 2 \cdot \mathrm{MAXF}$
MAXF denotes the Nyquist critical frequency.

Z Transform Syntax

Transconductance H(z):

```
Gxxx n+ n- ZTRANS in+ in- k0, k1, ..., kn
+ / d0, d1, ..., dm
+ [MAXF=val][SCALE=] [TC1=val] [TC2=val] [M=val]
```

Voltage Gain H(z):

```
Exxx n+ n- ZTRANS in+ in- k0, k1, ..., kn
+ / d0, d1, ..., dm
+ [MAXF=val][SCALE=] [TC1=val] [TC2=val]
```

H(z) is a rational function, in the following form:

$$H(z) = \frac{k0 + k1 \cdot z^{-1} + k2 \cdot z^{-2} + ... + kn \cdot z^{-n}}{d0 + d1 \cdot z^{-1} + d2 \cdot z^{-2} + ... + dm \cdot z^{-m}}$$

You can use parameters to define the values of all coefficients

$(k0,\ k1,\ ...,d0,\ d1,\ ...)$

Example:

```
Glow_pass 0 out ZTRANS in 0 0.0317 0.0951
+ 0.0951 0.0317 / 1.0 -1.459 0.9104 -0.1978
Ehigh_pass out 0 ZTRANS in 0 -0.0082 -0.1793
+ 0.6579 -0.1793 -0.0082 / 1
```

The `Glow_pass` element statement describes a third-order low-pass filter, with the transfer function:

$$H(z) = \frac{0.0317 + 0.0951^{-1} + 0.0951^{-2} + 0.0317z^{-3}}{1.0 - 1.459z^{-1} + 0.9104z^{-(2)} + 0.1978z^{-3}}$$

The `Ehigh_pass` element statement describes a fourth-order highpass filter, with the transfer function:

$$H(z) = 0.0082 - 0.1793z^{-1} + 0.06579z^{-2} - 0.1793z^{-3}$$

G- and E-element Notes

- If elements in the data file specify frequency responses, do not use pole/zero analysis. If you specify `MAXF=<`*par*`>` in a G- or E-element Statement, HSPICE or HSPICE RF warns that it is ignoring `MAXF`. This is normal.
- HSPICE or HSPICE RF performs circular convolution, when G-element `ACCURACY = 0` or `1` (see Figure 165 on page 707).

Laplace Band-Reject Filter

This example models an active band-reject filter, with 3-dB points at 100 and 400 Hz, and <35 dB of attenuation, between 175 and 225 Hz. The band-reject filter contains low-pass and high-pass filters, and an adder. The low-pass and high-pass filters are fifth-order Chebyshev, with 0.5-dB ripple.

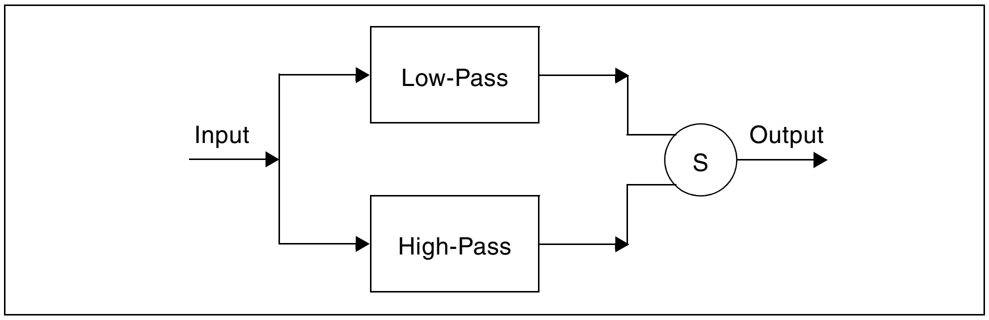


Figure 159 Band-Reject Filter

Example

This example is located in the following directory:
$*installdir*/demo/hspice/filters/BandstopL.sp

The BandstopL.sp file also contains a sample band-reject filter circuit.

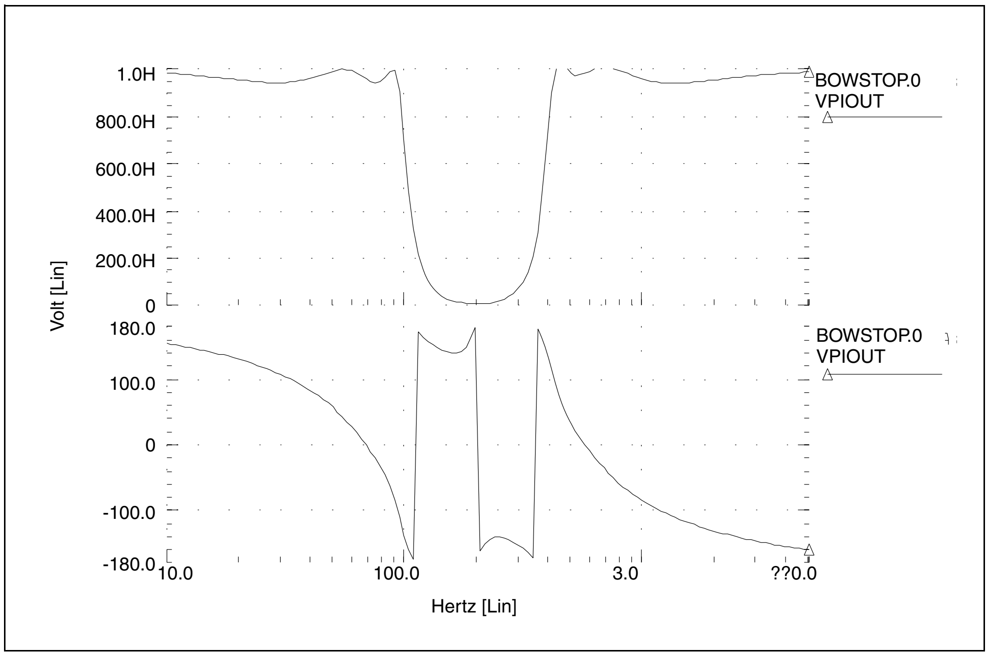


Figure 160 Frequency Response of the Band-Reject Filter

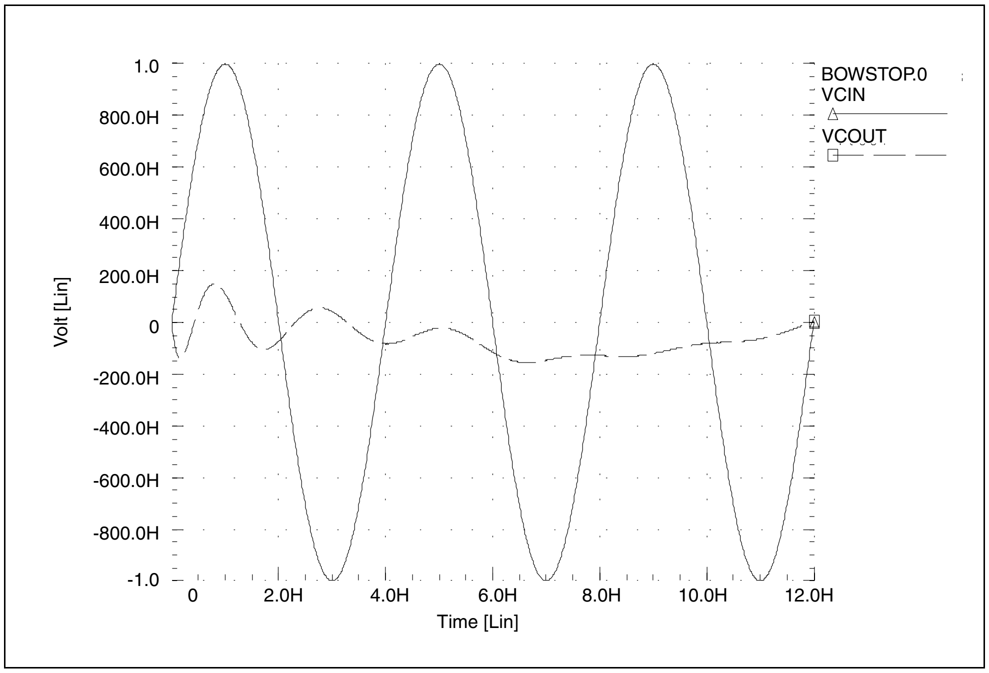


Figure 161 Transient Response of the Band-Reject Filter to a 250 Hz Sine Wave

Laplace Low-Pass Filter

This example simulates a third-order low-pass filter, with a Butterworth transfer function. It also compares the results of both the actual circuit and the functional G Element, with the third-order Butterworth transfer function, for AC and transient analysis.

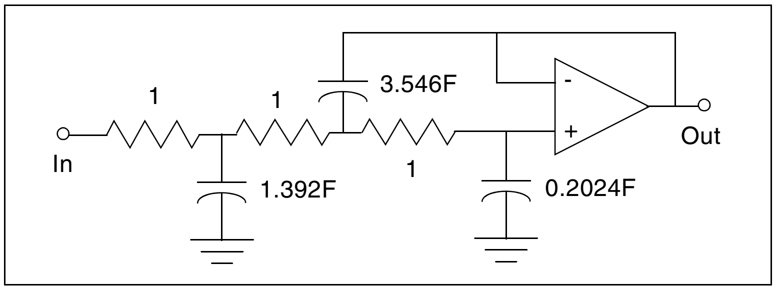


Figure 162 Third-Order Active Low-Pass Filter

The third-order Butterworth transfer function that describes the above circuit is:

$$H(s) = \frac{1.0}{1.0 \cdot (s+1)(s+0.5+j2\pi \cdot 0.1379)(s+0.5-(j2\pi \cdot 0.1379))}$$

The following example is the input listing for the above filter. Parameters set the pole locations for the G Element. Also, this listing specifies only one of the complex poles. The program derives the conjugate pole. The output of the circuit is the `out` node, and the output of the functional element is `outg`.

Example

An example of a third-order low-pass Butterworth filter is located in the following directory:
$installdir/demo/hspice/filters/Low_Pass.sp

The Low_Pass.sp file also contains a sample circuit description.

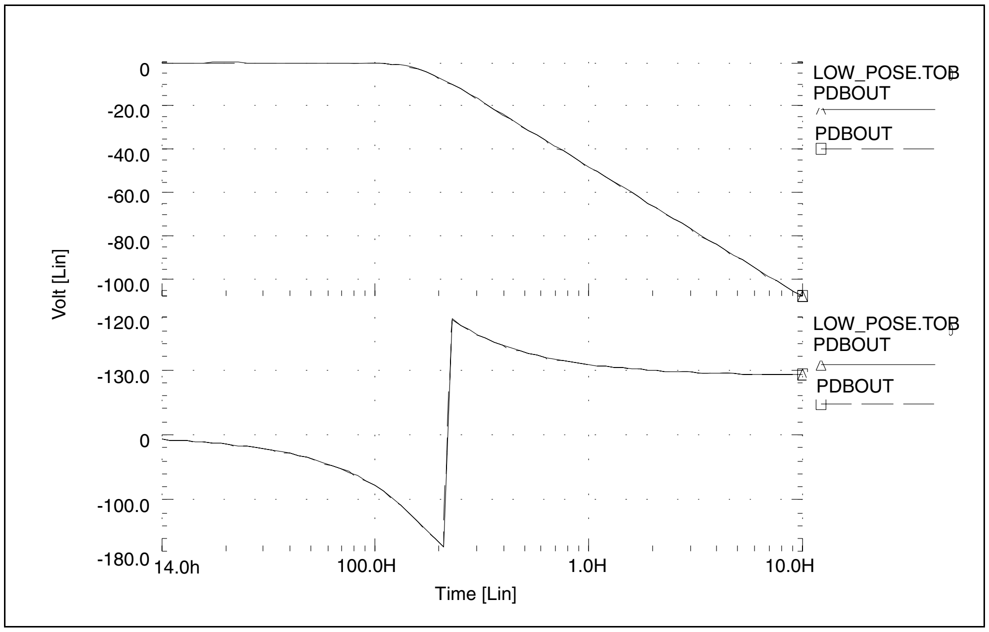


Figure 163 Frequency Response of Circuit and Functional Element

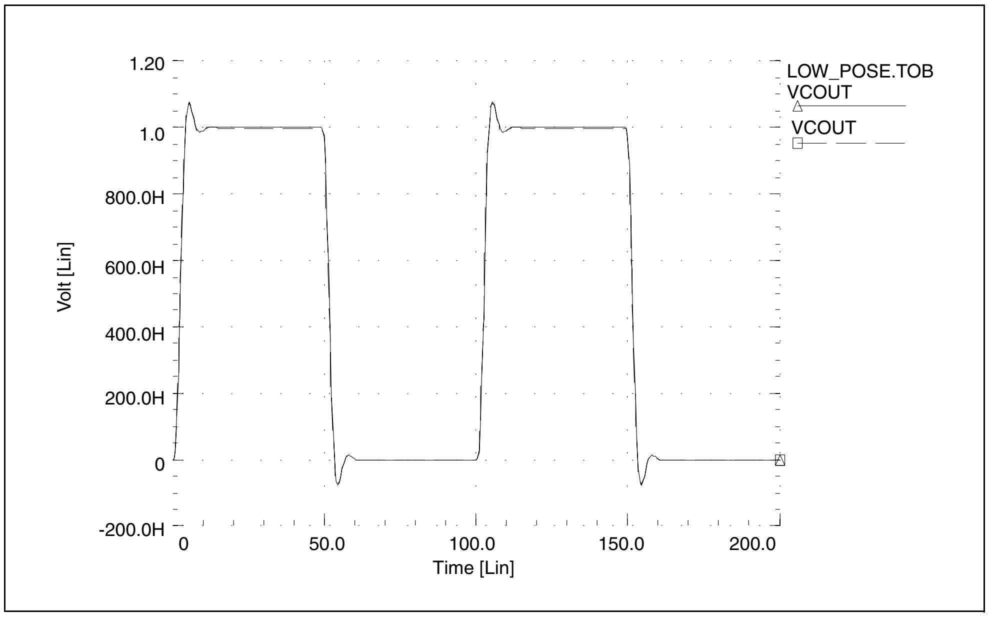


Figure 164 Transient Response of Circuit and Functional Element to a Pulse

Circular Convolution Example

This 30-degree phase-shift filter uses circular convolution. `If DELF=10 MHz`, HSPICE or HSPICE RF uses inverse fast Fourier transform (IFFT) to obtain the period of time domain response for the G-element. This value is based on the input frequency table, and is 100 ns. The `FREQ` G-element performs the convolution integral from t - T to t, assuming that all control voltages at t<0 are zero. t is the target time point, and T is the period of the time domain response, for the G-element.

In this example, during time points from 0 to 100 ns, HSPICE or HSPICE RF uses harmonic components higher than 10 MHz, due to the input transition at t=0. So the circuit does not behave as a phase shift filter. After one period (t>100 ns), HSPICE or HSPICE RF performs circular convolution, based on a period of 100 ns. The transient result represents a 30-degree phase shift, for continuous periodic control voltage.

Notes

- V(ctrl): control voltage input.
- V(expected): node. Represents an ideal 30-degree shifted wave for the input.
- V(test): output of the G Element.

30-Degree Phase Shift Circuit File

This example illustrates a 30-degree phase-shift filter. It is based on demonstration netlist phaseshift.sp, which is available in directory $<installdir>/demo/hspice/filters.

```
****
.tran 0.1n 300n
.OPTION post ingold=2 accurate
Vctrl ctrl gnd sin (0 1 10e6)
Gtest gnd test freq ctrl gnd
+ 1.0e00 0 30
+ 1.0e01 0 30
+ 1.0e02 0 30
+ 1.0e03 0 30
+ 1.0e04 0 30
+ 1.0e05 0 30
+ 1.0e06 0 30
+ 1.0e07 0 30
+ 1.0e08 0 30
+ 1.0e09 0 30
+ 1.0e10 0 30
+ MAXF=1.0e9 DELF=10e6
Rtest test gnd 1
Iexpected gnd 3 sin (0 1 10e6 0 0 30)
Vmes 3 expected 0v
Rexpected expected gnd 1
.end
```

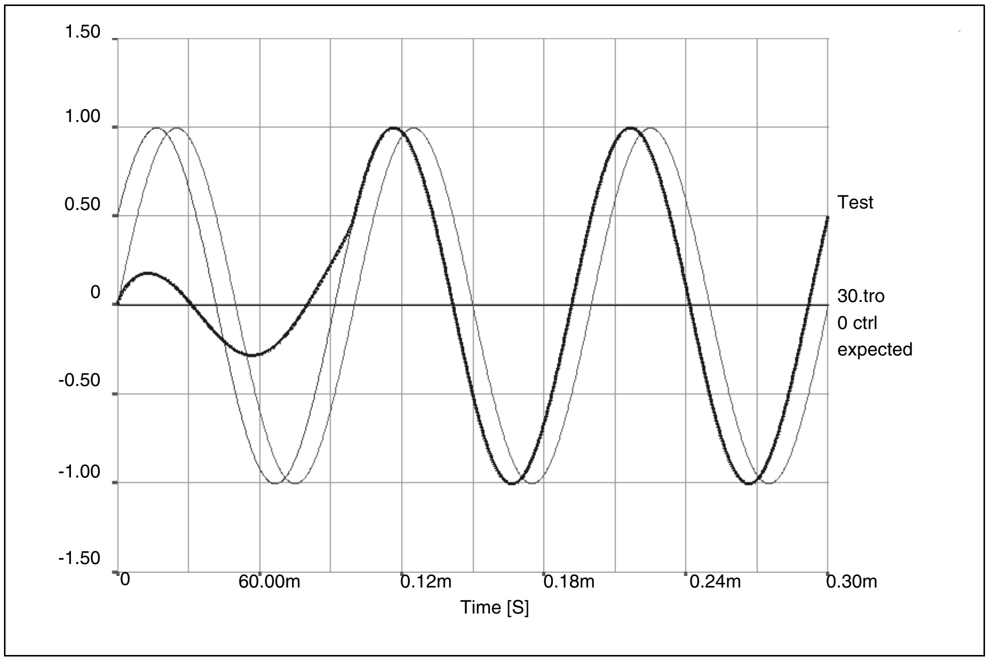


Figure 165 Transient Response of the 30 Degree Phase Shift Filter

Laplace and Pole-Zero Modeling

Laplace Transform (LAPLACE) Function

HSPICE or HSPICE RF provides two types of `LAPLACE` function calls: one for transconductance, and one for voltage gain transfer functions. See Using G- and E-elements on page 696 for the general forms, and Element Statement Parameters on page 699 for descriptions of the parameters.

General Form of the Transfer Function

To use `LAPLACE` modeling function, you must find the k_0, ..., k_n and d_0, ..., d_m coefficients of the transfer function. The transfer function is the s-domain (frequency domain) ratio, of the output for a single-source circuit, to the input, with initial conditions set to zero. The following equation represents the Laplace transfer function:

$$H(s) = \frac{Y(s)}{X(s)}$$

where:

- s is the complex frequency, j2πf.
- Y(s) is the Laplace transform of the output signal.
- X(s) is the Laplace transform of the input signal.

  Note:

  To obtain the impulse response H(s), HSPICE or HSPICE RF performs AC analysis, where `AC=1` represents the input source. The Laplace transform of an impulse is 1. For an element with an infinite response at DC (such as a unit step function H(s)=1/s), HSPICE or HSPICE RF calculations use the value of the `EPSMIN` option (the smallest number possible on the platform) as the transfer function.

The general form of the transfer function H(s) in the frequency domain, is:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

The order of the numerator for the transfer function cannot be greater than the order of the denominator. The exception is differentiators, for which the transfer function H(s) = ks. You can use parameter values for all k and d coefficients of the transfer function, in the circuit descriptions.

Finding the Transfer Function

The first step in determining the transfer function of a circuit is to convert the circuit to the s-domain. To do this, transform the value for each element, into its s-domain equivalent form.

Table 70 on page 708 and Table 71 on page 710 show transforms that convert some common functions to the s-domain. The next section provides examples of using transforms, to determine transfer functions.

Table 70 Laplace Transforms for Common Source Functions

f(t), t>0	Source Type	L{f(t)}= F(s)
σ(t)	impulse	1

Table 70 Laplace Transforms for Common Source Functions (Continued)

f(t), t>0	**Source Type**	**L{f(t)}= F(s)**
u(t)	step	$\frac{1}{s}$
t	ramp	$\frac{1}{s^2}$
e-at	exponential	$\frac{1}{s+a}$
sin ωt	sine	$\frac{w}{s^2+w^2}$
cos ωt	cosine	$\frac{s}{s^2+w^2}$
$\sin(wt+\theta)$	sine	$\frac{s\sin(\theta)+\omega\cos(\theta)}{s^2+\omega^2}$
$\cos(wt+\theta)$	cosine	$\frac{s\cos(\theta)-\omega\sin(\theta)}{s^2+\omega^2}$
$\sinh wt$	hyperbolic sine	$\frac{\omega}{s^2-\omega^2}$
$\cosh wt$	hyperbolic cosine	$\frac{s}{s^2-\omega^2}$
te-at	damped ramp	$\frac{1}{(s+a)^2}$
$e^{-at}\sin wt$	damped sine	$\frac{\omega}{(s+a)^2+\omega^2}$

Table 70 Laplace Transforms for Common Source Functions (Continued)

f(t), t>0	Source Type	L{f(t)}= F(s)
$e^{-at}\cos wt$	damped cosine	$\frac{s+a}{(s+a)^2+\omega^2}$

Table 71 Laplace Transforms for Common Operations

f(t)	L{f(t)} = F(s)
$Kf(t)$	$KF(s)$
$f_1(t)+f_2(t)-f_3(t)+..$	$F_1(s)+F_2(s)-F_3(s)+..$
$\frac{d}{dt}f(t)$	$sF(s)-f(0-\ )$
$\frac{d^2}{dt^2}f(t)$	$s^2F(s)-sf(0)-\frac{d}{dt}f(0-\ )$
$\frac{d^n}{dt^n}f(t)$	$s^nF(s)-s^{n-1}f(0)-s^{n-2}\frac{d}{dt}f(0)$
$\int_{-\infty}^{t} f(t)dt$	$\frac{F(s)}{s}+\frac{f^{-1}(0)}{s}$
$f(t-a)u(t-a),\ a>0$ (u is the step function)	$e^{-as}F(s)$
$e^{-at}f(t)$	$F(s+a)$
$f(at),\ a>0$	$\frac{1}{a}F\left(\frac{s}{a}\right)$

Table 71 Laplace Transforms for Common Operations (Continued)

f(t)	L{f(t)} = F(s)
$tf(t)$	$-\frac{d}{ds}(F(s))$
$t^n f(t)$	$-(1)^n \frac{d^n}{ds^n} F(s)$
$\frac{f(t)}{t}$	$\int_s^\infty F(u)du$ (u is the step function)
$f(t-t_1)$	$e^{-t_1 s} F(s)$

Determining the Laplace Coefficients

The following examples describe how to determine the appropriate coefficients, for the Laplace modeling function call.

Laplace Example 1 – Voltage Gain Transfer Function To find the voltage-gain transfer function for the circuit in Figure 166 on page 711, convert the circuit to its equivalent s-domain circuit, and solve for v_o / v_g.

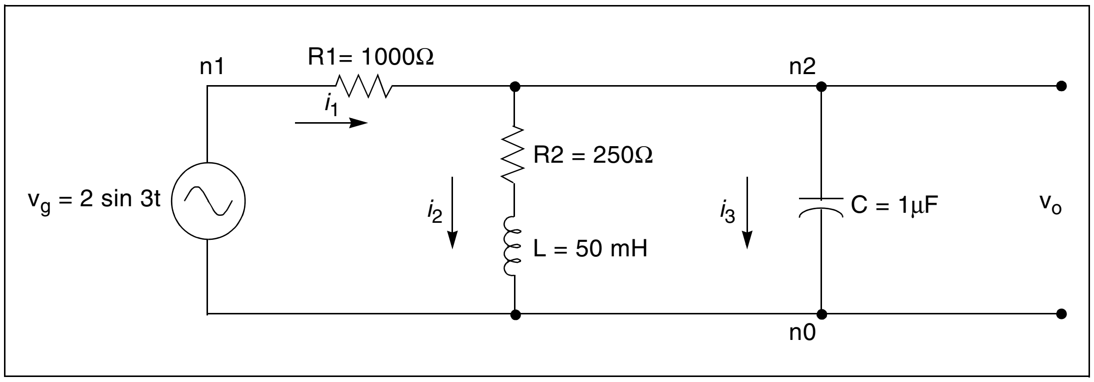


Figure 166 LAPLACE Example 1 Circuit

Use transforms fromTable 71 on page 710 to convert the inductor, capacitor, and resistors. L{f(t)} represents the Laplace transform of f(t):

$$L\left\{L\frac{d}{dt}f(t)\right\} = L\cdot\ (sF(s)-f(0)) = 50\times 10^{-3}\cdot\ (s-0) = 0.05s$$

$$L\left\{\frac{1}{C}\int_0^t f(t)dt\right\} = \frac{1}{C}\cdot\ \left(\frac{F(s)}{s}+\frac{f^{-1}(0)}{s}\right) = \frac{1}{10^{-6}}\cdot\ \left(\frac{1}{s}+0\right) = \frac{10^6}{s}$$

$$L\{R1\cdot\ f(t)\} = R1\cdot\ F(s) = R1 = 1000\ \ W$$

$$L\{R2\cdot\ f(t)\} = R2\cdot\ F(s) = R2 = 250\ \ W$$

To convert the voltage source to the s-domain, use the sin ωt transform from Table 70 on page 708:

$$L\{2\sin 3t\} = 2\cdot\ \frac{3}{s^2+3^2} = \frac{6}{s^2+9}$$

Figure 167 on page 712 displays the s-domain equivalent circuit.

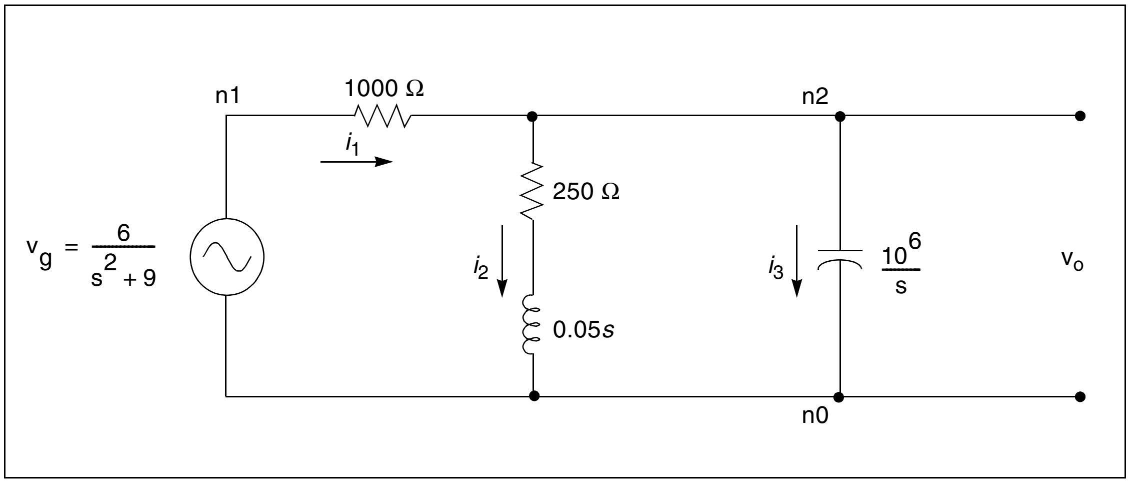


Figure 167 S-Domain Equivalent of the LAPLACE Example 1 Circuit

Summing the output currents from the n2 node:

$$\frac{v_o - v_g}{1000} + \frac{v_o}{250+0.05s} + \frac{v_o s}{10^6} = 0$$

Solve for v_o:

$$v_o = \frac{1000(s+5000)v_g}{s^2 + 6000s + 25 \times 10^6}$$

The voltage-gain transfer function is:

$$H(s) = \frac{v_o}{v_g} = \frac{1000(s+5000)}{s^2 + 6000s + 25 \times 10^6} = \frac{5 \times 10^6 + 1000s}{25 \times 10^6 + 6000s + s^2}$$

For the `Laplace` function call, use the k_n and d_m coefficients for the transfer function, in the form:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

The coefficients from the above voltage-gain transfer function are:

$k_0 = 5 \times 10^6$ $k_1 = 1000$

$d_0 = 25 \times 10^6 d_1 = 6000$ $d_2 = 1$

Using these coefficients, the following is a Laplace modeling function call, for the voltage-gain transfer function of the circuit in Figure 166 on page 711:

LAPLACE Example 2 – Differentiator To model a differentiator, use either G or E elements, as shown in the following example.

In the frequency domain:

E-element: $V_{out} = ksV_{in}$

G-element: $I_{out} = ksV_{in}$

In the time domain:

E-element: $v_{out} = k\frac{dV_{in}}{dt}$

G-element: $i_{out} = k\frac{dV_{in}}{dt}$

For a differentiator, the voltage gain transfer function is:

$$H(s) = \frac{V_{out}}{V_{in}} = ks$$

In the general form of the transfer function:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

If you set k_1 = k and d_0 = 1, and the remaining coefficients are zero, then the equation becomes:

$$H(s) = \frac{ks}{1} = ks$$

Using the k_1 = k and d_0 = 1 coefficients, in the Laplace modeling, the circuit descriptions for the differentiator are:

```
Edif out GND LAPLACE in GND 0 k / 1
Gdif out GND LAPLACE in GND 0 k / 1
```

LAPLACE Example 3 – Integrator You can use G or E Elements to model an integrator, as follows:

In the frequency domain:

E Element: $V_{out} = \frac{k}{s} V_{in}$

G Element: $I_{out} = \frac{k}{s} V_{in}$

In the time domain:

E Element: $v_{out} = k \int V_{in} dt$

G Element: $i_{out} = k \int V_{in} dt$

For an integrator, the voltage gain transfer function is:

$$H(s) = \frac{V_{out}}{V_{in}} = \frac{k}{s}$$

In the general form of the transfer function:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

As in the previous example, if you set k_0 = k and d_1 = 1, then the equation becomes:

$$H(s) = \frac{k + 0 + \ldots + 0}{0 + s + \ldots + 0} = \frac{k}{s}$$

Laplace Transform POLE (Pole/Zero) Function

This section describes the general form of the pole/zero transfer function. It also provides examples of converting specific transfer functions, into pole/zero circuit descriptions.

POLE Function Call

You can use the `POLE` function if the poles and zeros of the circuit are available. You can derive the poles and zeros from the transfer function, as described in this chapter, or you can use the `.PZ` statement to find them, as described in the HSPICE and HSPICE RF Command Reference.

HSPICE or HSPICE RF provides two forms of the `LAPLACE` function call: one for transconductance, and one for voltage gain transfer functions. See Using G- and E-elements on page 696 for the general forms, and for optional parameters.

To use the `POLE` pole/zero modeling function, find the a, b, f, and α coefficients of the transfer function. The transfer function is the s-domain (frequency domain) ratio of the output, for a single-source circuit to the input, with initial conditions set to zero.

General Form of the Transfer Function

The general expanded form of the pole/zero transfer function H(s) is:

$$H(s) = \frac{a(s + \alpha_{z1} + j2\pi f_{z1})(s + \alpha_{z1} - j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})(s + \alpha_{zn} - j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})(s + \alpha_{p1} - j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})(s + \alpha_{pm} - j2\pi f_{pm})}$$

You can use parameters to set the a, b, α, and f values.

The following is an example:

```
Ghigh_pass 0 out   POLE in 0 1.0   0.0,0.0 / 1.0  0.001,0.0
Elow_pass out 0    POLE in 0   1.0 / 1.0, 1.0,0.0   0.5,0.1379
```

The `Ghigh_pass` statement describes a high pass filter, with the transfer function:

$$H(s) = \frac{1.0 \cdot (s + 0.0 + j \cdot 0.0)}{1.0 \cdot (s + 0.001 + j \cdot 0.0)}$$

The `Elow_pass` statement describes a low-pass filter, with the transfer function:

$$H(s) = \frac{1.0}{1.0 \cdot (s+1)(s + 0.5 + j2\pi \cdot 0.1379)(s + 0.5 - (j2\pi \cdot 0.1379))}$$

To write a pole/zero circuit description for an element, you need to know H(s) transfer function of the element, in terms of the a, b, f, and α coefficients.

Before you use the values of these coefficients in `POLE` function calls (in the circuit description), you must simplify the transfer function, as described in the next section.

Reduced Form of the Transfer Function

Complex poles and zeros occur in conjugate pairs (a set of complex numbers differ only in the signs of their imaginary parts):

$(s + \alpha_{pm} + j2\pi f_{pm})(s + \alpha_{pm} - j2\pi f_{pm})$, for poles.

$(s + \alpha_{zn} + j2\pi f_{zn})(s + \alpha_{zn} - j2\pi f_{zn})$, for zeros.

To write the transfer function in pole/zero format, supply coefficients for one term of each conjugate pair. HSPICE or HSPICE RF provides the coefficients for the other term. If you omit the negative complex roots, the result is the reduced form of the transfer function, Reduced{H(s)}.

To find the reduced form, collect all general-form terms that have negative complex roots:

$$H(s) = \frac{a(s + \alpha_{z1} + j2\pi f_{z1})..(s + \alpha_{zn} + j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})..(s + \alpha_{pm} + j2\pi f_{pm})} \cdot \frac{a(s + \alpha_{z1} - j2\pi f_{z1})..(s + \alpha_{zn} - j2\pi f_{zn})}{b(s + \alpha_{p1} - j2\pi f_{p1})..(s + \alpha_{pm} - j2\pi f_{pm})}$$

Then discard the right-hand term, which contains all terms with negative roots. What remains is the reduced form:

$$Reduced\{H(s)\} = \frac{a(s+\alpha_{z1}+j2\pi f_{z1})..(s+\alpha_{zn}+j2\pi f_{zn})}{b(s+\alpha_{p1}+j2\pi f_{p1})..(s+\alpha_{pm}+j2\pi f_{pm})}$$

For this function, find the a, b, f, and α coefficients to use in a `POLE` function, for a voltage-gain transfer function. The following examples show how to determine the coefficients, and write `POLE` function calls for a high-pass filter and a low-pass filter.

POLE Example 1 – Highpass Filter For a high-pass filter with a transconductance transfer function, such as:

$$H(s) = \frac{s}{(s+0.001)}$$

Find the a, b, α , and f coefficients needed to write the transfer function in the general form shown previously. You can then see the conjugate pairs of complex roots. You need to supply only one of each conjugate pair of roots, in the `Laplace` function call. HSPICE or HSPICE RF automatically inserts the other root.

To transform the function into a form that is more similar to the general form of the transfer function, rewrite the transconductance transfer function as:

$$H(s) = \frac{1.0(s+0.0)}{1.0(s+0.001)}$$

Because this function has no negative imaginary parts, it is already in the HSPICE or HSPICE RF reduced form (reference number 2) shown previously.

You can now identify the a, b, f, and α coefficients, so that the H(s) transfer function matches the reduced form. This matching process obtains the values:

n = 1, m = 1

a = 1.0 α_{z1} = 0.0 f_{z1} = 0.0

b = 1.0 α_{p1} = 0.001 f_{p1} = 0.0

Using these coefficients in the reduced form, provides the transfer function:

$$\frac{s}{(s+0.001)}$$

So the general transconductance transfer function `POLE` function call:

```
Gxxx n+ n- POLE in+ in- a α z1,fz1... α zn,fzn /
b  α p1,fp1... α pm,fpm
```

for an element named Ghigh_pass, becomes:

```
Ghigh_pass gnd out POLE in gnd 1.0 0.0,0.0 /
+ 1.0 0.001,0.0
```

POLE Example 2 – Low-Pass Filter For a low-pass filter, with the following voltage-gain transfer function:

$$H(s) = \frac{1.0}{1.0(s+1.0+j2\pi\cdot\ 0.0)(s+0.5+j2\pi\cdot\ 0.15)(s+0.5-j2\pi\cdot\ 0.15)}$$

you need to find the a, b, α, and f coefficients, to write the transfer function in the general form, so that you can identify the complex roots with negative imaginary parts.

To separate the reduced form, Reduced{H(s)}, from the terms with negative imaginary parts, rewrite the voltage-gain transfer function as:

$$H(s) = \frac{1.0}{1.0(s+1.0+j2\pi\cdot\ 0.0)(s+0.5+j2\pi\cdot\ 0.15)}\cdot\frac{1.0}{(s+0.5-j2\pi\cdot\ 0.15)}$$
$$= Reduced\{H(s)\}\cdot\frac{1.0}{(s+0.5-j2\pi\cdot\ 0.15)}$$

So:

$$Reduced\{H(s)\} = \frac{1.0}{1.0(s+1.0)(s+0.5+j2\pi\cdot\ 0.15)}$$

or:

$$\frac{a(s+\alpha_{z1}+j2\pi f_{z1})..(s+\alpha_{zn}+j2\pi f_{zn})}{b(s+\alpha_{p1}+j2\pi f_{p1})..(s+\alpha_{pm}+j2\pi f_{pm})} = \frac{1.0}{1.0(s+1.0+j2\pi\cdot\ 0.0)(s+0.5+j2\pi\cdot\ 0.15)}$$

Now assign coefficients in the reduced form, to match the specified voltage transfer function. The following coefficient values produce the transfer function:

```
n = 0, m = 2,
a =1.0   b = 1.0   $\alpha_{p1}$ = 1.0   $f_{p1}$ = 0   $\alpha_{p2}$ = 0.5   $f_{p2}$ = 0.15
```

You can substitute these coefficients in the `POLE` function-call, for a voltage-gain transfer function:

```
Exxx n+ n- POLE in+ in- a $\alpha_{z1}$,$f_{z1}$...$\alpha_{zn}$,$f_{zn}$ /
b $\alpha_{p1}$,$f_{p1}$...$\alpha_{pm}$,$f_{pm}$
```

for an element named `Elow_pass`, to obtain the following statement:

```
Elow_pass out GND POLE in 1.0 / 1.0 1.0,0.0 0.5,0.15
```

RC Line Modeling

Most RC lines can use very simple models, with only a single dominant pole. You can use AWE methods to find the dominant pole, computed based on the total series resistance and capacitance, or determined using the Elmore delay.

The Elmore delay uses the (d1-k1) value as the time constant, for a single-pole approximation to the complete H(s), where H(s) is the transfer function of the RC network for a specified output. The inverse Laplace transform of h(t) is H(s):

$$\tau_{DE} = \int_0^{\infty} t \cdot h(t)dt$$

Actually, the Elmore delay is the first moment of the impulse response, and so corresponds to a first-order AWE result.

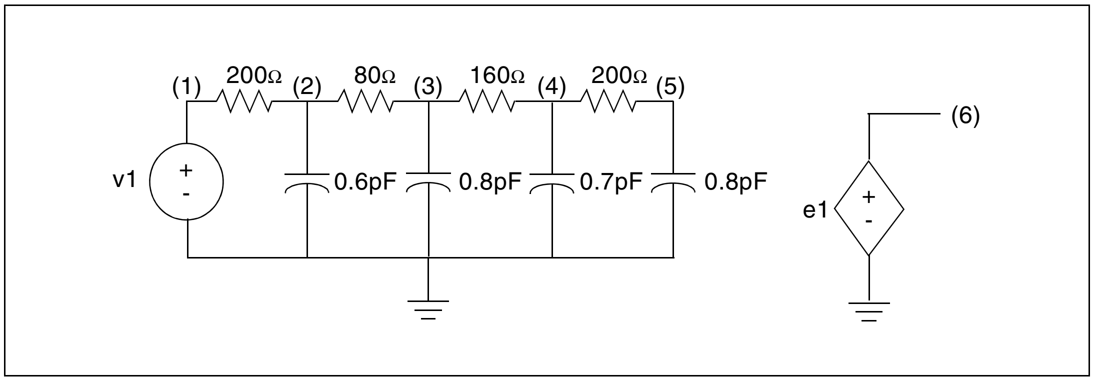


Figure 168 Circuits for a RC Line

Example

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

```
****
.Tran 0.02ns 3ns
.OPTION Post Accurate List Probe
v1 1 0 PWL 0ns 0 0.1ns 0 0.3ns 5 1.3ns 5 1.5ns 0
r1 1 2 200
c1 2 0 0.6pF
r2 2 3 80
c2 3 0 0.8pF
r3 3 4 160
c3 4 0 0.7pF
r4 4 5 200
c4 5 0 0.8pF
e1 6 0 LAPLACE 1 0 1 / 1 1.16n
.Probe v(1) v(5) v(6)
.Print v(1) v(5) v(6)
.End
```

To closely approximate the output of the RC circuit (shown in Figure 168), you can use a single-pole response, as shown in Figure 169.

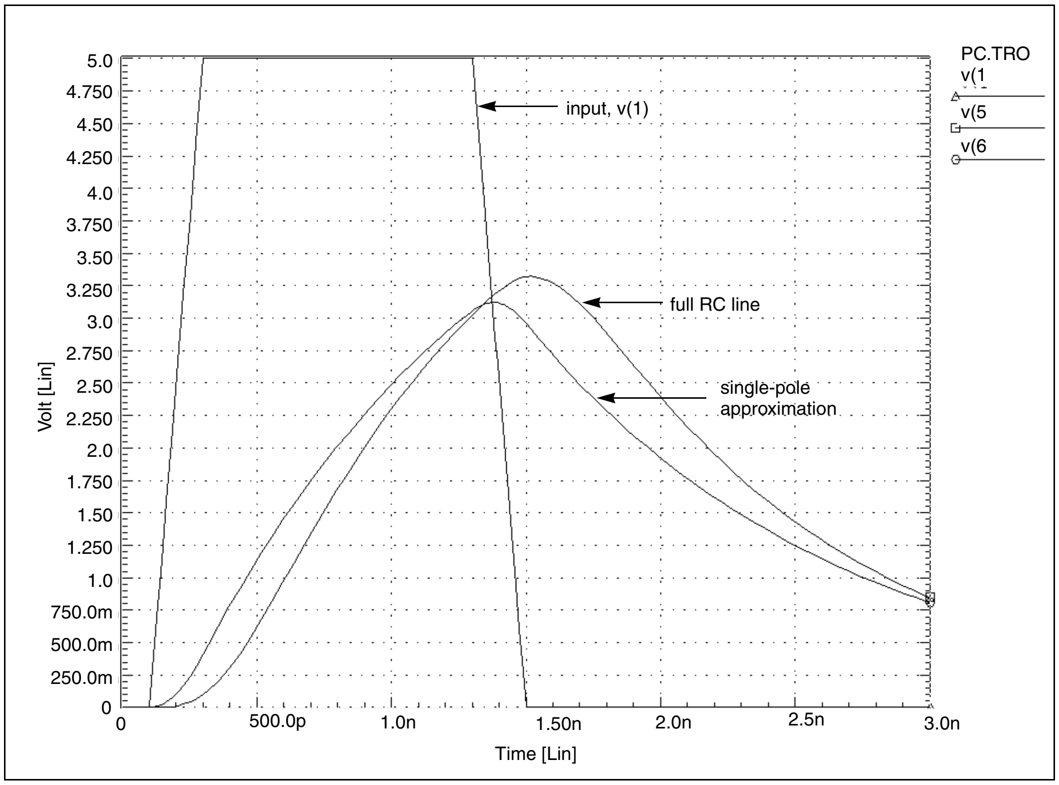


Figure 169 Transient Response of the RC Line and Single-Pole Approximation

In Figure 169, the single-pole approximation has less delay: 1 ns, compared to 1.1 ns for the full RC line model, at 2.5 V. The single-pole approximation also has a lower peak value than the RC line model. All other things being equal, a circuit with a shorter time constant results in less filtering, and allows a higher maximum voltage value. The single-pole approximation produces a lower amplitude (and less delay) than the RC line because the single pole neglects the other three poles in the actual circuit. However, a single-pole approximation still provides very good results for many problems.

AWE Transfer Function Modeling

Approximations, using single-pole transfer functions, can cause larger errors for low-loss lines than for RC lines because lower resistance allows ringing. Circuit ringing creates complex pole pairs in transfer function approximation. You need at least one complex pole pair, to represent low-loss line response. Figure 170 is a typical low-loss line, and the transfer function sources used to test various approximations. To obtain the transfer functions, HSPICE or HSPICE RF uses asymptotic waveform evaluation.

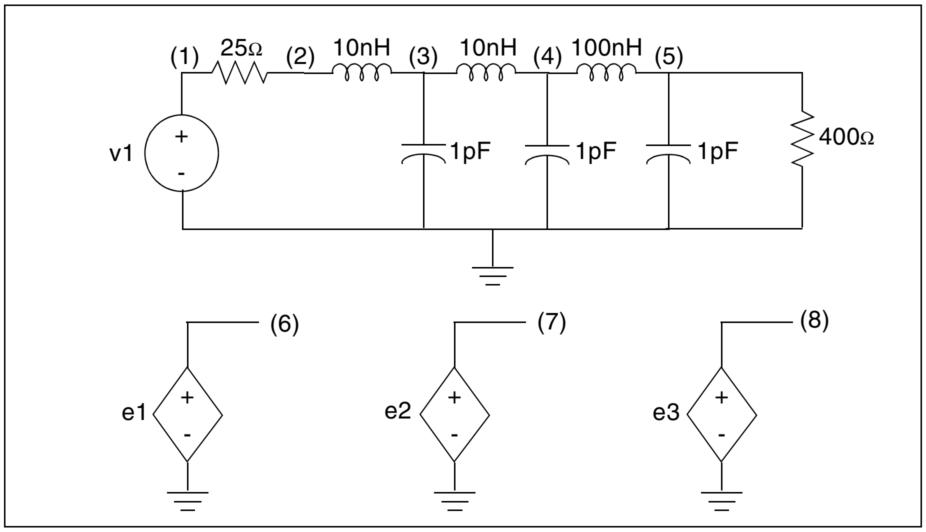


Figure 170 Circuits for a Low-Loss Line

The sample file located in the following directory is a Low-Loss Line circuit file: $installdir/demo/hspice/filters/lowloss.sp

Figure 171 shows a transient response of a low-loss line. It also shows E Element Laplace models, using one, two, and four poles. The single-pole model shows none of the ringing of the higher-order models. Also, all E models must adjust the gain of their response, for the finite load resistance, so the models are not independent of the load impedance. The 0.94-gain multiplier in the models take care of the 25-ohm source, and the 400-ohm load-voltage divider. These approximations are good delay estimations.

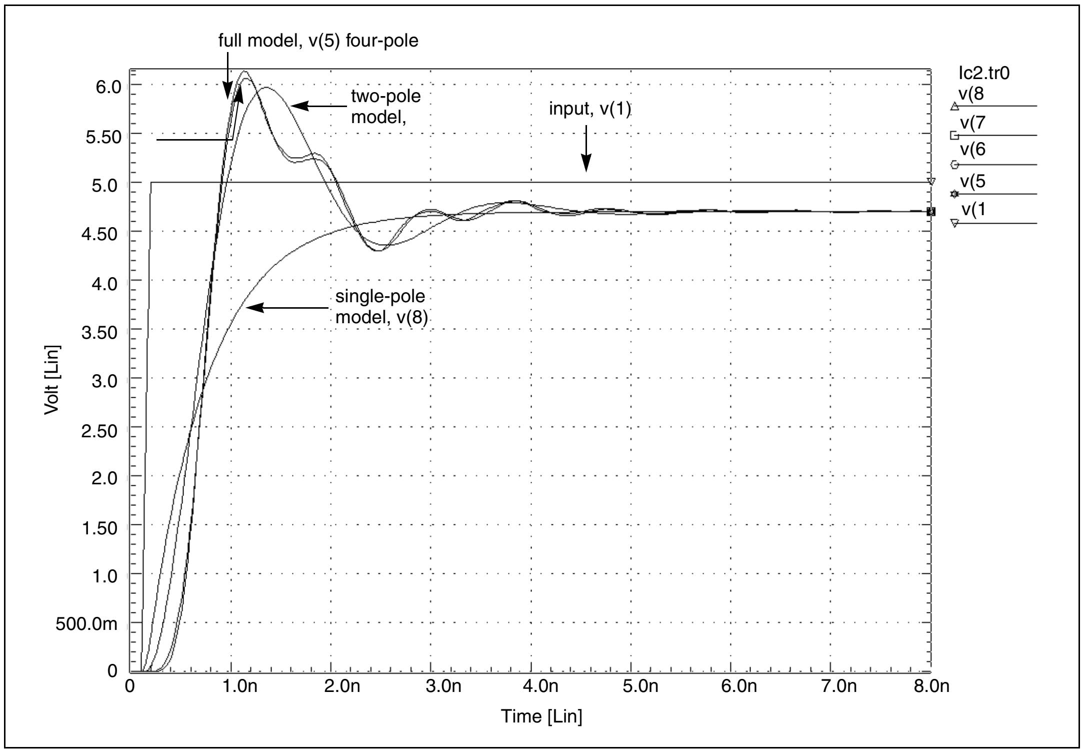


Figure 171 Transient Response of the Low-Loss Line

Although the two-pole approximation provides reasonable agreement with the transient overshoot, the four-pole model offers almost perfect agreement. The actual circuit has six poles. You can use scaling to bring some of the very small numbers in the Laplace model, above the 1e-28 limit of HSPICE or HSPICE RF. The `SCALE` parameter multiplies ever Y-parameter in the LAPLACE specification, by the same value (in this case 1.0E-20).

A low-loss line allows reflections between the load and source, compared to the loss of an RC line, which usually isolates the source from the load. So you can either incorporate the load into the AWE transfer function approximation, or create a HSPICE device model that allows source/load interaction. If you allow source/load interaction, you do not need to perform the AWE expansions each time that you change load impedances. This allows HSPICE or HSPICE RF to handle non-linear loads, and removes the need for a gain multiplier, as in the circuit file shown. You can use four voltage-controlled current sources, or G Elements, to create a Y-parameter model for a transmission line. The Y-parameter network provides the needed source/load interaction. The next example shows such a Y-parameter model, for a low-loss line.

Y-parameter Line Modeling

A model that is independent of load impedance, is more complicated. You can still use AWE techniques, but you need a way for the load voltage and current to interact with the source impedance. For a transmission line of 100 ohms and 0.4 ns total delay (as shown in Figure 172), to compare the response of the line, use a Y-parameter model and a single-pole model.

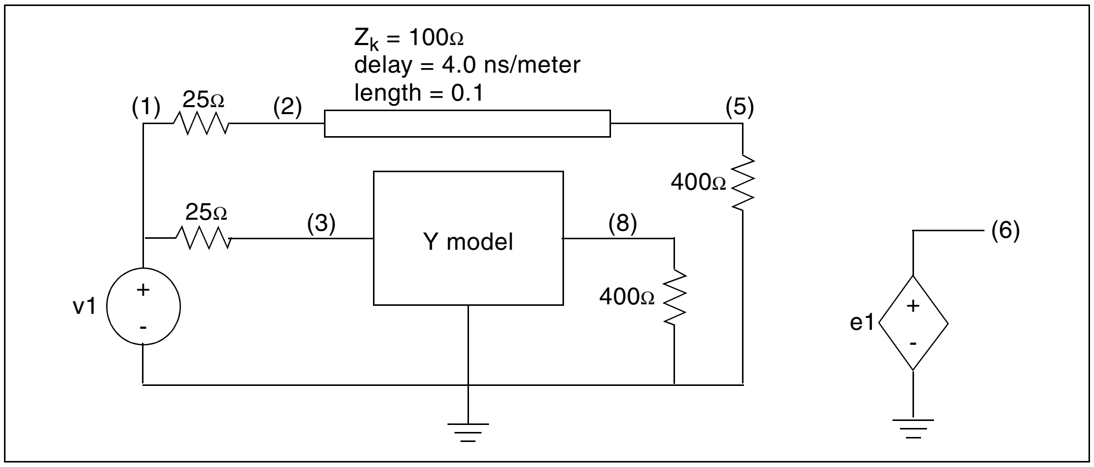


Figure 172 Line and Y-parameter Modeling

Figure 173 shows the voltage and current definitions for a Y-parameter model.

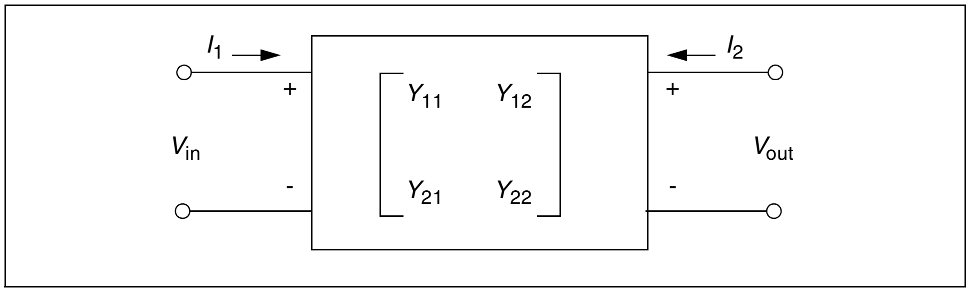


Figure 173 Y Matrix for the Two-Port Network

The following equations describe the general network in Figure 173, which you can translate into G Elements:

$$I_1 = Y_{11}V_{in} + Y_{12}V_{out}$$

$$I_2 = Y_{21}V_{in} + Y_{22}V_{out}$$

Figure 174 shows a schematic for a set of two-port Y-parameters. The circuit consists mostly of G Elements.

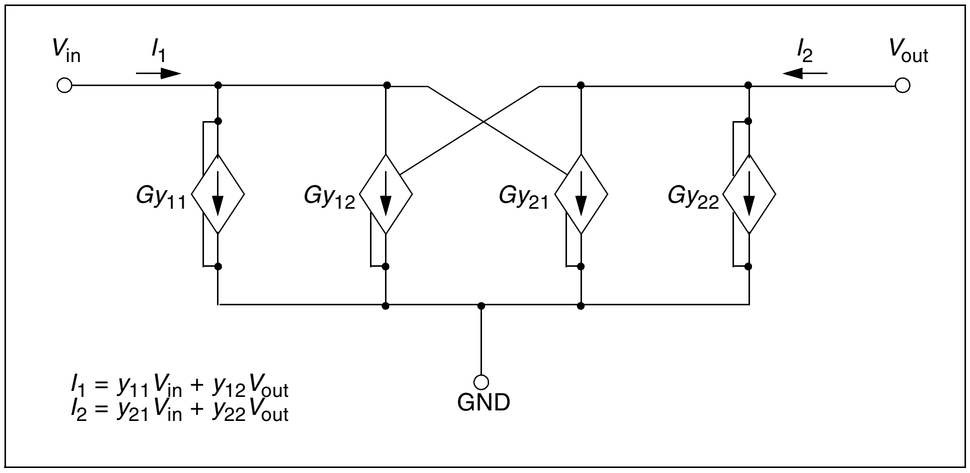

V_{in}
I_1
I_2
V_{out}
Gy_{11}
Gy_{12}
Gy_{21}
Gy_{22}
$I_1 = y_{11}V_{in} + y_{12}V_{out}$
$I_2 = y_{21}V_{in} + y_{22}V_{out}$
GND

Figure 174 Schematic for the Y-parameter Network

A Pade expansion of the Y-parameters for a transmission line, determines the Laplace parameters for the Y-parameter model, as shown in matrix form in the following equation:

$$Y = \frac{1}{Z_o} \cdot \begin{bmatrix} \coth(p) & -\operatorname{csch}(p) \\ -\operatorname{csch}(p) & \coth(p) \end{bmatrix}$$

where *p* is the product of the propagation constant, times the line length.

A Pade approximation contains polynomials, in both the numerator and the denominator. A Pade approximation can model both poles and zeros, and coth and csch functions also contain both poles and zeros, so a Pade approximation provides a better low-order model, than a series approximation does.

The following equation calculates the Pade expansion of coth(p) and csch(p), with a second-order numerator and a third-order denominator:

$$coth(p) \rightarrow \frac{\left(1 + \frac{2}{5} \cdot p^2\right)}{\left(p + \frac{1}{15} \cdot p^3\right)} \qquad csch(p) \rightarrow \frac{\left(1 - \frac{1}{20} \cdot p^2\right)}{\left(p + \frac{7}{60} \cdot p^3\right)}$$

When you substitute ($s \cdot length \cdot \sqrt{LC}$) for p, HSPICE or HSPICE RF generates polynomial expressions for each G Element. When you substitute 400 nH for L, 40 pF for C, 0.1 meter for length, and 100 for Z_o, ($Z_o = \sqrt{L/C}$) in the matrix equation above, you can use the resulting values in a circuit file.

The circuit file shown in the following sections uses all of the equation substitutions. The Pade approximations have different denominators for csch and coth, but the circuit file contains identical denominators. Although the actual denominators for csch and coth are only slightly different, using them can cause oscillations in the HSPICE response. To avoid this problem, use the same denominator in the coth and csch functions in the example. The simulation results might vary, depending on which denominator you use as the common denominator because the coefficient of the third-order term changes (but by less than a factor of 2).

This sample LC Line circuit file is located in the following directory:
$installdir/demo/hspice/filters/lcline.sp

Figure 175 compares the output of the Y-parameter model, with that of a full transmission line simulation, and with that obtained for a single-pole transfer function. In the latter case, the gain for the load impedance is incorrect, so the function produces an incorrect final voltage level. As expected, the Y parameter model provides the correct final voltage level. Although the Y parameter model provides a good approximation of the circuit delay, it contains too few poles to model all of the transient details. However, the Y-parameter model provides excellent agreement with the overshoot and settling times.

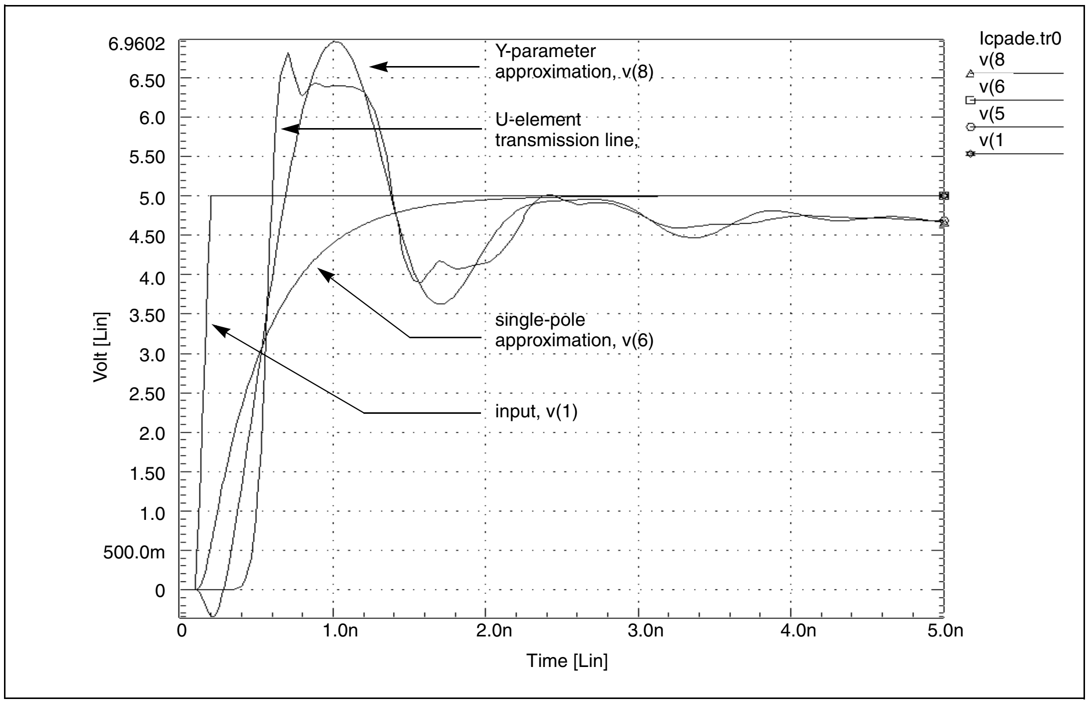


Figure 175 Transient Response of the Y-parameter Line Model

Comparison of Circuit and Pole/Zero Models

This example simulates a ninth-order, low-pass filter circuit, and compares the results with its equivalent pole/zero description, using an E Element. The results are identical, but the pole/zero model runs about 40% faster.The example shown in Simulation Time Summary on page 726 shows the total CPU times for the two methods. For larger circuits, the computation time saving can be much higher.

Figure 176 on page 728 and Figure 177 on page 729 display the transient and frequency response comparisons, resulting from the two modeling methods.

Simulation Time Summary

Circuit model simulation times:

```
analysis   time   # points   # iter   conv.iter
op point   0.23   1   3
ac analysis   0.47   151   151
transient   0.75   201   226   113  rev=0
readin   0.22
errchk   0.13
setup   0.10
output   0.00
total cpu time 1.98 seconds
```

Pole/zero model simulation times:

```
analysis   time   # points   # iter   conv.iter

op point   0.12   1   3

ac analysis   0.22   151   151

transient   0.40   201   222   111  rev=0

readin   0.23

errchk   0.13

setup   0.02

output   0.00

total cpu time 1.23 seconds
```

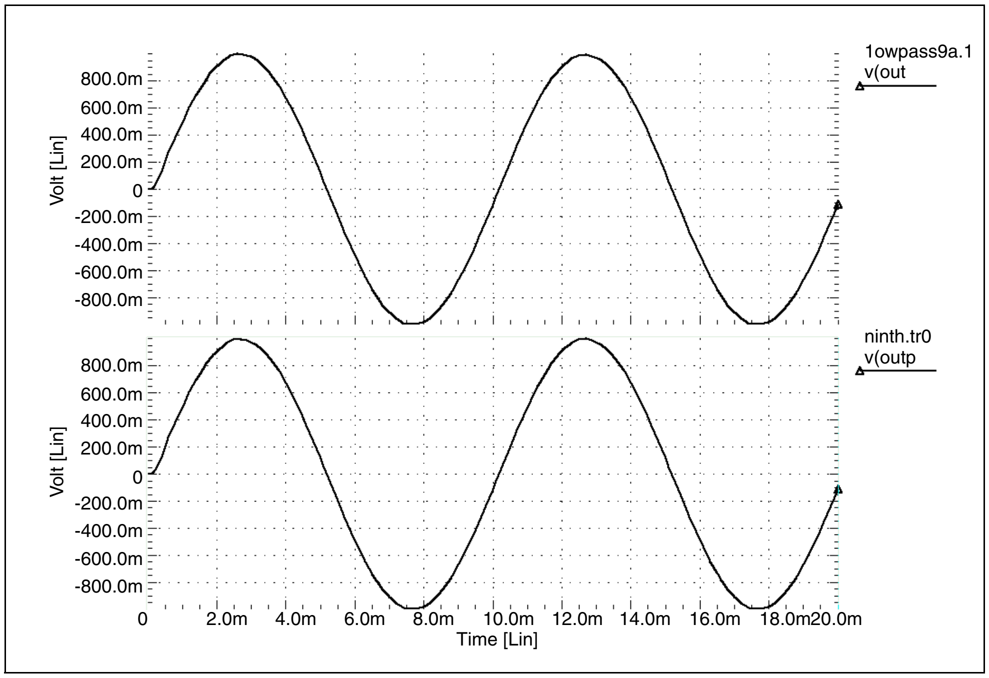


Figure 176 Transient Responses of the Circuit and Pole/Zero Models

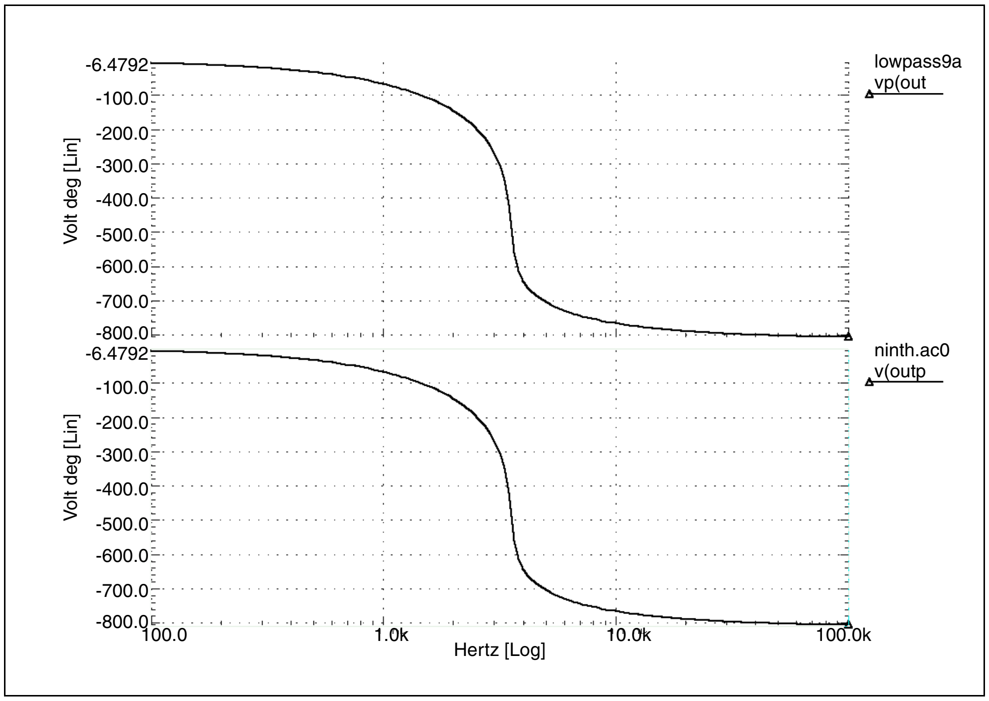


Figure 177 AC Analysis Responses of the Circuit and Pole/Zero Models

Modeling Switched Capacitor Filters

Switched Capacitor Network

You can model a resistor as a capacitor and switch combination. The value of the equivalent is proportional to the frequency of the switch, divided by capacitance.

Construct a filter from MOSFETs and capacitors, where the filter characteristics are a function of the switching frequency of the MOSFETs.

To quickly determine the filter characteristics, use ideal switches (voltage controlled resistors), instead of MOSFETs. The resulting simulation speed-up can be as great as 7 to 10 times faster than a circuit using MOSFETs.

To construct an RC network, the model uses a resistor and a capacitor, along with a switched-capacitor equivalent network. The `RCOUT` node is the resistor/capacitor output, and `VCROUT` is the switched-capacitor output.

The `GVCR1` and `GVCR2` switches, and the `C3` capacitance, model the resistor. The following equation calculates the resistor value:

$$Res = \frac{Tswitch}{C3}$$

where Tswitch is the period of the PHI1 and PHI2 pulses.

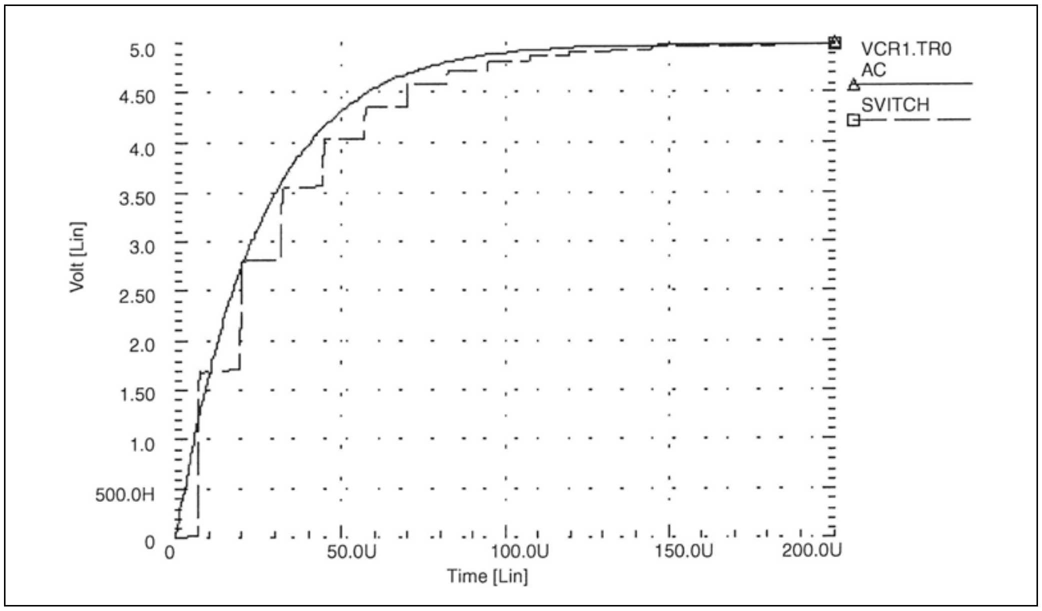


Figure 178 VCR1.SP Switched Capacitor RC Circuit

Switched Capacitor Filter Example

This example is a fifth-order elliptic, switched-capacitor filter. The passband is 0-1 kHz, with loss less than 0.05 dB. This results from cascading models of the switches. The resistance is1 ohm when the switch is closed, and 100 Megohm when it is open. The E Element models op-amps as an ideal op-amp. This example provides the transient response of the filter, for 1 kHz and 2 kHz sinusoidal input signals.

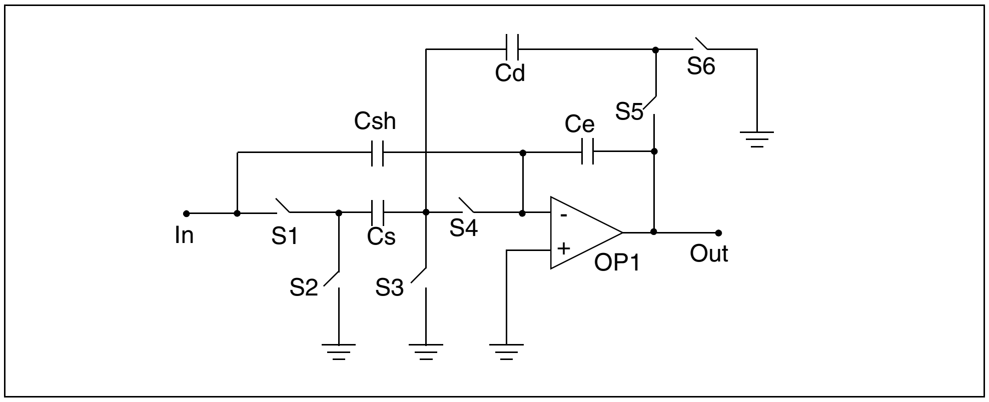


Figure 179 Linear Section

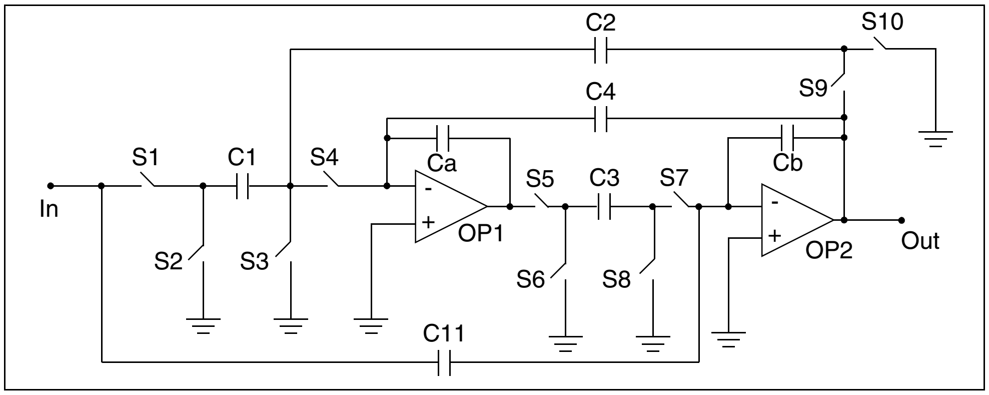


Figure 180 High_Q Biquad Section

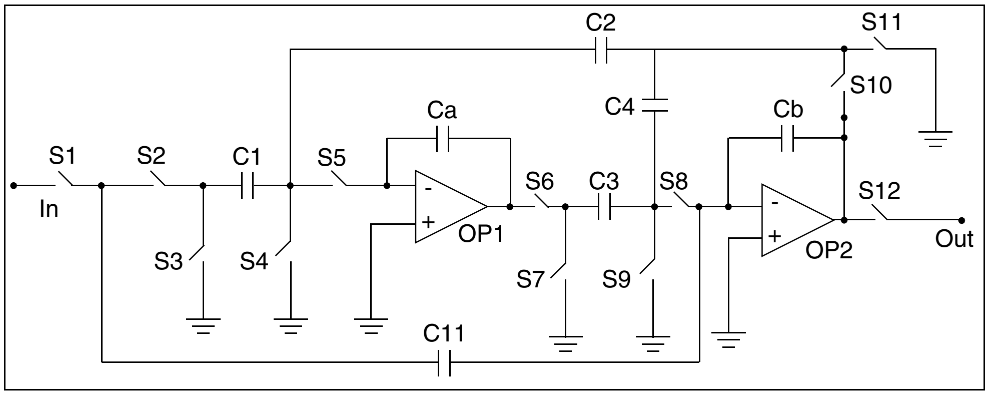


Figure 181 Low_Q Biquad Section

Input File for Switched Capacitor Filter

This sample input file for a switched capacitor filter is located in the following directory:

$installdir/demo/hspice/behave/swcap5.sp

This file also contains the following examples:

- Sample and Hold
- Linear Section
- High_Q Biquad Section
- Low_Q Biquad Section

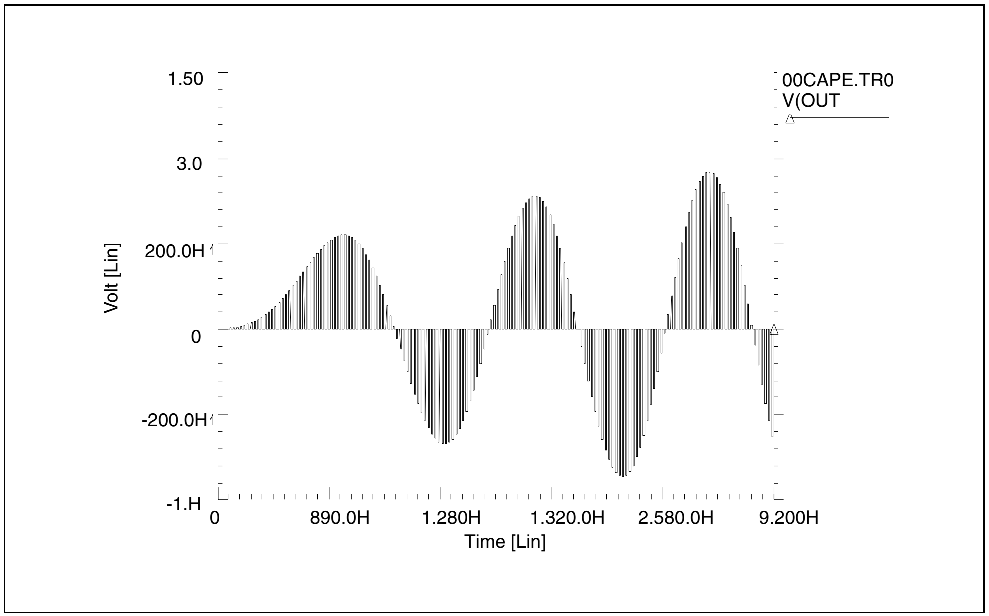


Figure 182 Response to 1-kHz Sinusoidal Input

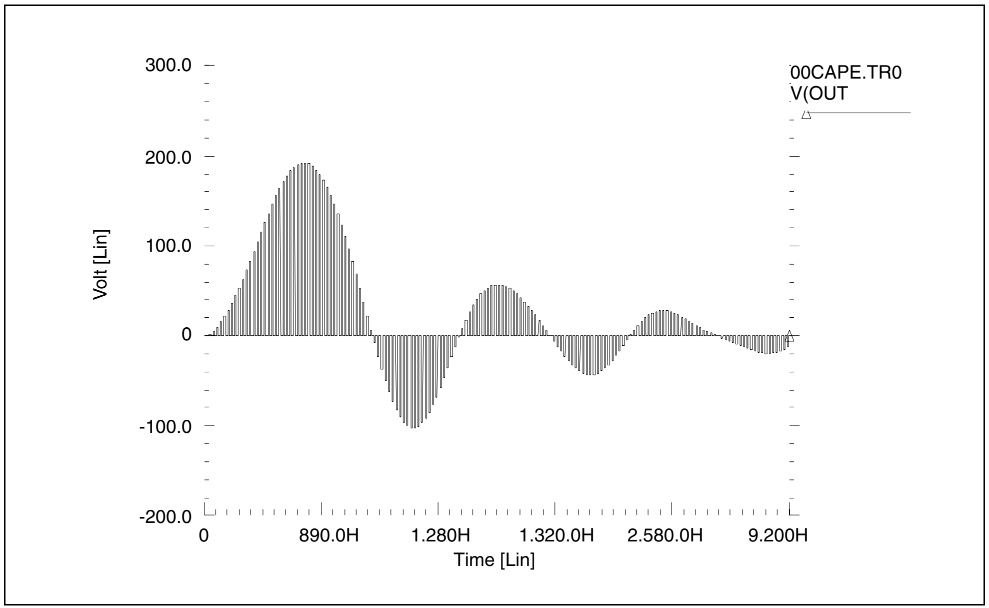


Figure 183 Response to 2-kHz Sinusoidal Input

References

[1] Williams, Arthur B., and Taylor, Fred J. *Electronic Filter Design Handbook*. New York: McGraw-Hill, 1988, pp. 6-20 to 6-23.

[2] Nillson, James W. *Electric Circuits*, 4th Edition. Reading, Massachusetts: Addison-Wesley, 1993.

[3] Edminister, Joseph A. *Electric Circuits*. New York: McGraw-Hill, 1965.

[4] Ghausi, Kelly, and M.S. “On the Effective Dominant Pole of the Distributed RC Networks”, *Jour. Franklin Inst.*, June 1965, pp. 417- 429.

[5] Elmore, W.C. and Sands, M. *Electronics, National Nuclear Energy Series*, New York: McGraw-Hill, 1949.

[6] Pillage, L.T. and Rohrer, R.A. “Asymptotic Waveform Evaluation for Timing Analysis”, *IEEE Trans. CAD*, Apr. 1990, pp. 352 - 366.

[7] Kuo, F. F. *Network Analysis and Synthesis*. John Wiley and Sons, 1966.

[8] Gregorian, Roubik & Temes, Gabor C. *Analog MOS Integrated Circuits*. J. Wiley, 1986, page 354.

30

Simulation of Random Noise

Describes the characteristics of random signals, types of noise, component noise models, and noise simulation in HSPICE and HSPICE RF.

Introduction

One of the essential techniques in designing an analog circuit is to be able to identify major noise sources in the circuit and know how to minimize their affect to an acceptable level. Noise can be loosely defined as a disturbance signal with random amplitude, which is usually an unwanted phenomenon in a circuit.

It is important to limit the power of noise to a low level relative to that of signal; otherwise, the signal can be greatly distorted or completely indistinguishable. They rely on simulation tools to help them identify the dominant sources of noise in the circuit, measure the noise levels (in power, current, or voltage), and eventually make changes in the design to reduce the noise levels if necessary.

Two types of noise are commonly defined: inherent and interference.

- Inherent noise is the noise created by the elements internal to the circuit (resistors, diodes, and transistors) caused by movement of electrons in the circuit.
- Interference noise refers to the unwanted signals from the environment absorbed by the circuit.

Eliminating interfering noise can be done by different methods of signal shielding or code modulation so the signal can be differentiated from the noise. Inherent noise, however, can only be affected by changes in the circuit topology and the bias currents.

This chapter discusses only inherent noise and the HSPICE simulations related to it. Whenever the term “noise” is used, it refers to “inherent noise”.

Presented first in the following sections are some theoretical definitions for noise sources in an analog circuit. These are followed by a discussion on how HSPICE and HSPICE RF performs noise simulation and how to interpret outputs from noise simulation.

Characteristics of Random Signals

This section describes noise sources in the time and frequency domains, and methods to deal with multitude noise sources in a circuit.

Probability Distribution Function versus Power Spectral Density

Noise is a signal with random amplitude. Figure 184 illustrates a sample noise signal in the time domain. The vertical axis could be either a branch current or a node voltage.

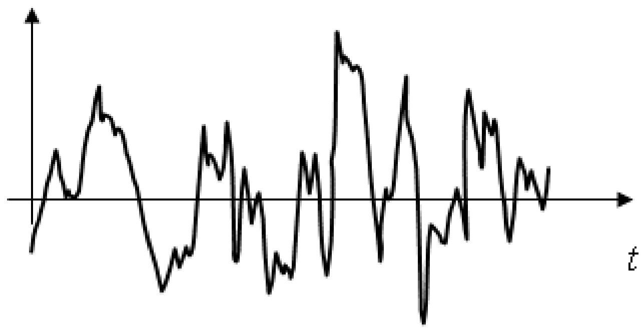


Figure 184 Typical noise signal (voltage or current) in the time domain

To simulate a noise source in the time domain, you have to know the Probability Distribution Function (PDF) of that signal and then create a voltage or current source that produces a random output signal conforming to the PDF. Figure 185 demonstrates a sample PDF for a random signal with Gaussian distribution. Extracting PDF parameters out of circuit components is not straightforward and therefore not common in analyzing noise.

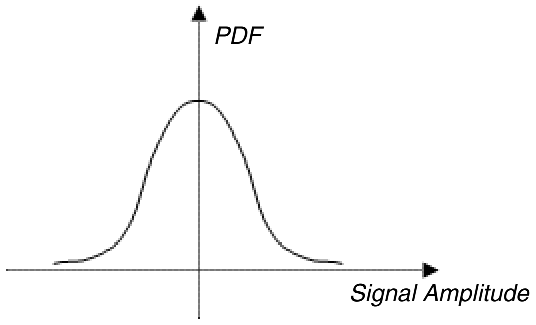


Figure 185 PDF for a random signal with Gaussian probability distribution

However, noise can be easily modeled and measured in the frequency domain. The noise models in the frequency domain also relate directly to the component and circuit characteristics. This is the reason noise analyses are usually done in the frequency domain.

While we cannot obtain a deterministic value for the noise amplitude in the time domain because it is a random variable, it is quite possible to determine the power of the noise if the noise is bandwidth limited. In other words, it is true that the amplitude of a noise signal changes randomly in the time domain. However, its power, as defined by Eq. is a fixed and finite value if and only if the signal is bandwidth limited.

Definition of Power for a noise voltage (a) and current (b), normalized for a 1-ohm load:

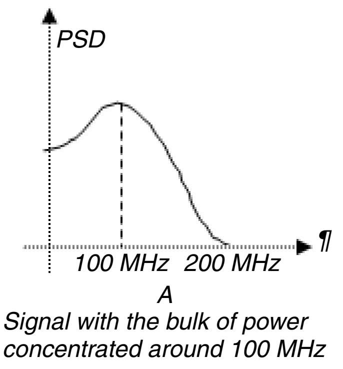
$$P = lim_{T \to \infty} \cdot\ 1/\ T\int_{-T/\ 2}^{T/\ 2} |v(t)|^2 dt \qquad \text{Eq. a}$$

$$P = lim_{T \to \infty} \cdot\ 1/\ T\int_{-T/\ 2}^{T/\ 2} |i(t)|^2 dt \qquad \text{Eq. b}$$

The power of a signal can also be described in the frequency domain; by means of a Power Spectral Density (PSD). A PSD identifies the portion of signal power present as a function of signal frequency. Figure 186 illustrates two PSD graphs.

Graph A resembles the PSD of a typical data signal in high speed and RF communication; a signal with a limited bandwidth. It can be seen from the graph that the composing frequencies of the signal are between DC (0 Hz) and 200 MHz, with the bulk of power being concentrated around 100 MHz.

Graph B shows a signal with a flat power distribution; meaning that the power is equally distributed across all frequencies. Such a signal is referred to as “white noise”. Resistor thermal noise possesses such a PSD.



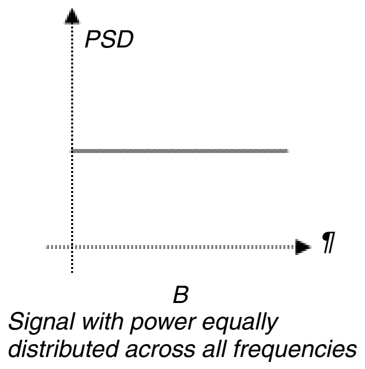


Figure 186 Sample PSD graphs

To obtain the total power of a signal within a certain bandwidth, you must calculate the area under the PSD curve within the given frequency range, as defined in Eq. 1.

Definition of power based in the frequency domain, normalized for a 1 ohm load:

$$P = \int_{f\_1}^{f\_2} PSD(f)df \tag{1}$$

An example of power calculation in the frequency domain for a noise signal is shown in Figure 187. In this example, the PSD is a flat curve with a value of 5 x 10_{-20} V_2/Hz.

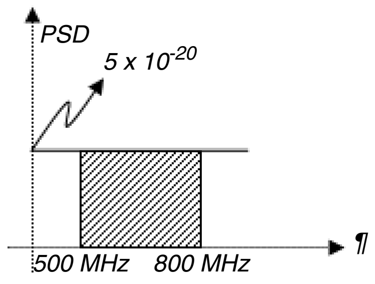


Figure 187 Sample PSD graphs

The power of the noise between the frequencies 500 MHz and 800 MHz is equal to the shaded area under the curve and is given by:

$$P_n = \int_{500M}^{800M} 5\times 10^{-20} df = (800M - 500M)5\times 10^{-20} = 1.5\times 10^{-11} V^2$$

The root mean square (RMS) value of the noise signal is given by Eq. 2. The RMS value of a signal is defined as a DC value that would render the same power as the signal in question. For the general sinusoidal signal Acos(ωt), the RMS value is $A/\sqrt{2}$.

RMS value of a signal is the root of the 1 ohm normalized signal power:

$$v_{rms} = \sqrt{P} \tag{2}$$

Which means the RMS voltage value of the noise signal in the above example is:

$$v_{rms} = \sqrt{1.5\times 10^{-11}} = 5.48\mu V$$

This means that the power of the noise is identical to that of a sinusoidal signal with a RMS value of 5.48 uV.

Multiple Noise Sources in a Circuit

In the usual case of dealing with a multitude of noise sources in a circuit, it is important to know how to add up the effects of noise to obtain the accumulated noise value.

If the noise sources are completely independent (that is, un-correlated), such as the noises generated by two separate resistors, their powers add up to their RMS values. Figure 188 shows how two independent noise sources, in this case both voltage sources, add up.

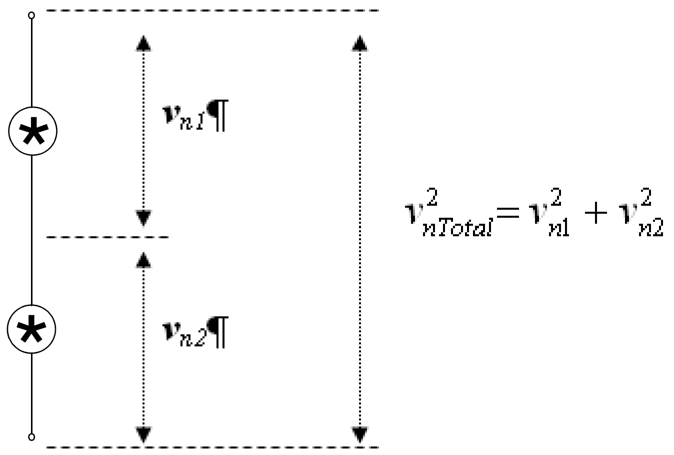


Figure 188 Noise summation: independent noise signals add in squared values

It might not be evident from the equation for V_{nTotal} in Figure 188 that when there are several noise sources in a circuit, only reducing the biggest noise contributors will reduce the total noise significantly. Reducing the smaller sources, even by large amounts does not have a considerable affect on the overall noise value because the power of two of signals magnifies the dominance of bigger sources over smaller ones.

To clarify this point, consider the case where two noise sources are present, with RMS values of v_1 = 2 uV and v_2 = 9 uV. The total noise is given by:

$$v_{nTotal} = \sqrt{(2)^2 + (9)^2} = 9.21\mu V$$

Now if in an attempt to lower the total noise the designer of the circuit cuts the 2 uV source by 50%, the total noise will be:

$$v_{nTotal} = \sqrt{(1)^2 + (9)^2} = 9.06\mu V$$

Now, if instead we cut the 9 uV source by only 10%, the total noise will be:

$$v_{nTotal} = \sqrt{(2^2) + (8.1^2)} = 8.34\mu V$$

This emphasizes the fact that to reduce the total noise, we should first concentrate on the strongest source of noise.

Summary

Important points from this section are:

- Inherent noise is best modeled in the frequency domain by its PSD.
- The power of noise depends on the bandwidth of the system and is defined as the area under the PSD curve between two given frequencies.
- When multiple noise sources are present, the square of their voltage (or current) values add.
- When multiple sources of noise are present in a circuit, the most effective way of reducing the total noise is to focus on reducing the most dominant noise contributor.

Noise Types

This section identifies and describes three common types of noise in analog circuits.

Thermal Noise

Thermal noise is generated by random movements of electrons in a resistive conductor. Any circuit element with resistive characteristics, whether a resistor or the base junction series resistor in a BJT’s small signal model, disseminates thermal noise. The models for thermal noise voltage source and their equivalent current source are shown in Figure 189.

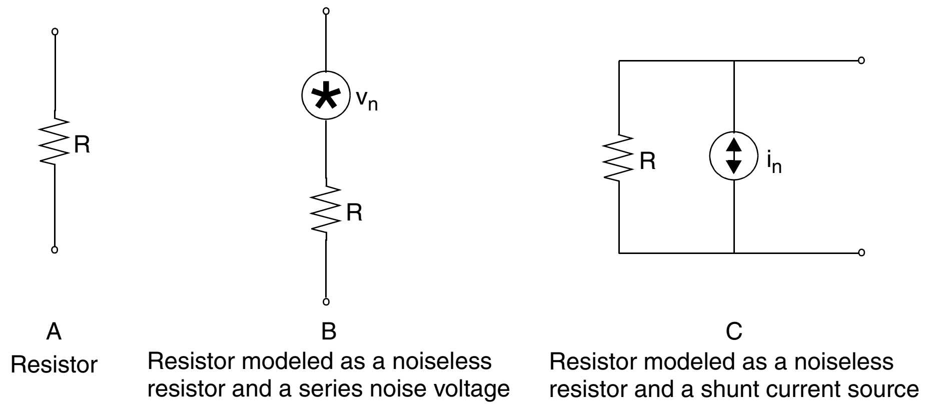

R
vn
R
R
in
A
Resistor
B
Resistor modeled as a noiseless resistor and a series noise voltage
C
Resistor modeled as a noiseless resistor and a shunt current source

Figure 189 Thermal noise voltage sources and their equivalent circuits

The voltage and current PSD functions for resistor thermal noise are:

$$V_n^2(f) = 4kTR \qquad v^2/\ Hz$$

$$I_n^2(f) = 4kT/\ R \qquad A^2/\ Hz$$

Where,

k is Boltzmann's constant equal to 1.38 x 10^{-23}.
T is the temperature in Kelvin degrees: 1 °K = °C + 273. For a room temperature of 25 °C (88 °F), the default temperature setting in HSPICE and HSPICE RF, T will be 298 °K.
R is the resistance in ohms.

It should be noted that the two voltage and current PSD models are equivalent, they both produce the same amount of open circuit voltage or short circuit current and can be used interchangeably during noise analysis.

Notice that thermal noise voltage PSD is proportional to the resistance while the current PSD is inversely proportional to the resistance. By looking at the PSD models, a general conclusion can be made; presence of bigger resistors in a circuit usually translates to a bigger thermal noise voltage.

The PSD models have been explicitly described as functions of frequency to re-emphasize the fact that the PSD is only meaningful in the frequency domain context.

The above models suggest that PSDs are flat across all frequencies. This assumption will no longer be true for frequencies above a few hundred gigahertz.

Example 1

What will be the RMS voltage value of thermal noise generated by a 10 kohm resistor operating in the 1- to 1.2-GHz frequency range? Assuming a room temperature of 25 °C or 298 °K, the PSD function will be (see also Figure 190):

$$(f) = 4kTR = 4(1.38\times 10^{-23})(298)(10K) = 1.64\times 10^{-16} \qquad V^2/$$

$$_{ms} = \int PSD(f)df = \int (1.64\times 10^{-16})df = (1.64\times 10^{-16})\times(0.2G) = 3.28\times 10^{-8} \qquad V^2$$

$$_{rms} = \sqrt{3.28\times 10^{-8}} = 181.11\mu V$$

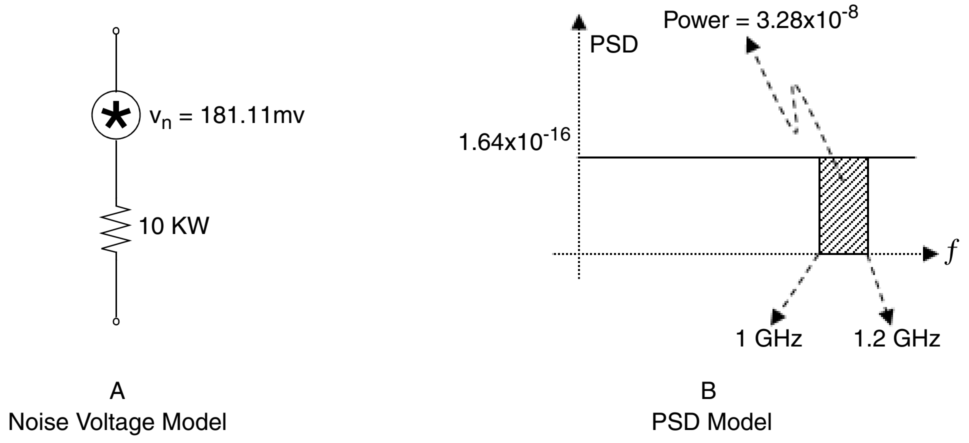


Figure 190 Noise models for 10 kohm resistor

Flicker Noise

Flicker noise is noise generated by fluctuations in the average current travel time in a conductor. As electrons take different random paths to get from one end of a resistor to the other at any given time, the average current will experience different resistance along the way, leading to fluctuations in current, hence the generation of flicker noise. Figure 191 illustrates how electrons can fall into paths with different lengths while flowing through a resistor.

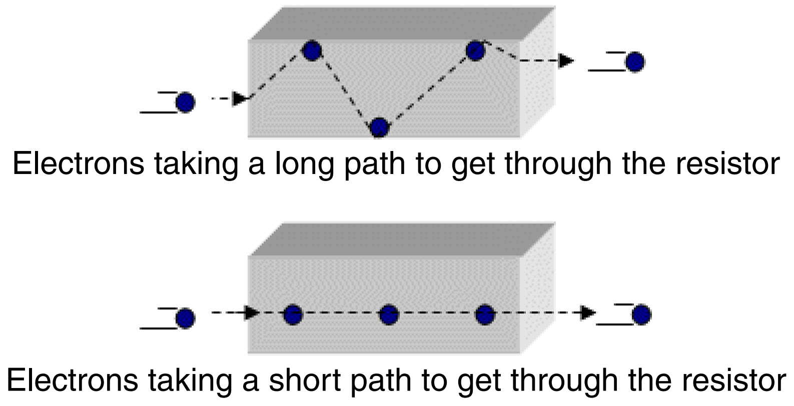

Electrons taking a long path to get through the resistor
Electrons taking a short path to get through the resistor

Figure 191 Paths for current through a resistor

By shrinking the cross-section area of the conducting material, there will be smaller differences between different current paths and the flicker noise will shrink. That is why thin film resistors have a much smaller flicker noise than carbon rod resistors, which have a much larger conducting cross-section.

In addition, at high frequencies, the current tends to travel through the outer surface of the conductor, a phenomenon know as “skin effect”, effectively shrinking the conducting cross-section to a very thin layer. This results in smaller differences in the current path lengths and consequently smaller flicker noise. Because of stronger flicker noise effects at higher frequencies, flicker noise is inversely proportional to frequency. That is why it is also called “1/f noise”. Flicker noise is only significant at lower frequency values, but it can have a major impact on nonlinear circuits at RF frequencies.

The major sources of flicker noise in analog circuits are semiconductor components. Resistors can be manufactured to produce small amount of flicker noise and are not usually taken into account for calculation of flicker noise in a circuit.

Flicker noise is usually represented as a current PSD for resistors and semiconductors with this equation:

$$I_n^2(f) = K_f I / f \qquad A^2 / Hz$$

Where,
I is the current through the resistor.
f is frequency in hertz.
K_f is a constant which depends on the characteristics and geometry of the conductor.

The PSD for flicker noise is depicted in Figure 192.

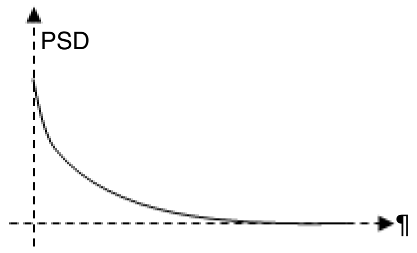


Figure 192 PSD for flicker noise

HSPICE and HSPICE RF use more general models for CMOS flicker noise PSDs. One of the models used for CMOS transistor is the SPICE2/Berkeley noise model:

$$I_n^2(f) = K_f I^{A_f} / (C_{ox} L^2 f^{E_f}) \qquad A^2/Hz$$

Where,
K_f is the flicker noise parameter.
A_f is the current exponent.
C_{ox} is the Gate oxide capacitance.
L is the effective length of the transistor.
E_f is the frequency exponent.

Another model used by HSPICE for flicker noise calculation is the BSIM model, which looks very similar to the above equation. It is important to note that you usually must not be concerned with the value settings of these parameters. They must be defined and set to proper values in the transistor models defined in the technology library.

The combined PSD for flicker and thermal noise is shown in Figure 193.

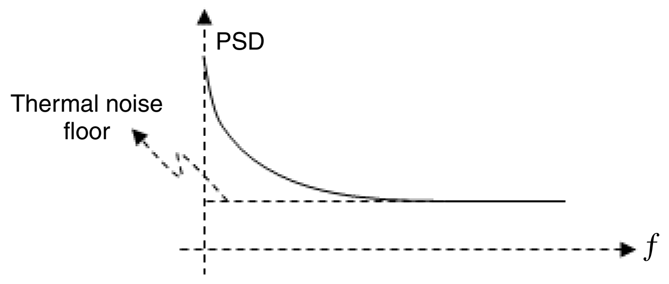


Figure 193 PSD of combined thermal and flicker noise

Note that flicker noise is the dominant source of noise at lower frequencies and as its magnitude shrinks at higher frequencies, the thermal noise will become the dominant source of noise. Thermal noise is also referred to as “the noise floor” because that is the minimum possible amount of noise in a circuit.

Shot Noise

Shot noise is only generated by semiconductor elements and is caused by random passage of electrons and holes across a potential barrier, such as a P-N junction. Shot noise is often represented by a current PSD, known as “Schottky formula”:

$$I_n^2(f) = 2qI \qquad A^2/\ Hz$$

Where,

q = 1.6 x 10^{-19} C is the charge of an electron.
I is the bias current of the junction and can be I_c, collector bias currents for a BJT transistor.

Shot noise PSD curves are flat across all frequencies.

Summary

The three major types of noise in an analog circuit are:

- Thermal noise, also known as white noise, is generated by resistors in the circuit. Thermal noise is a function of conductor resistance.
- Flicker noise, also called 1/f noise, is mainly generated by transistors in a circuit. It is a function of component geometry and its magnitude drops as frequency increases.
- Shot noise is caused by bias currents in the base and collector of BJT transistors.

Component Noise Models and HSPICE/HSPICE RF Noise Simulation

This section describes how HSPICE and HSPICE RF models noisy elements, calculates the total output noise PSD, obtains the total output noise voltage PSD, and how the RMS output noise voltage can be deduced from the total output noise PSD.

Element Noise Models

Each resistor, diode, and transistor in a circuit generates at lease some type of inherent noise. HSPICE and HSPICE RF model the noise generating elements in a circuit as a noiseless element combined with a noise current or voltage source with a known PSD. Figure 194 includes the listing of the four noise generating elements in analog circuits and their equivalent noise models.

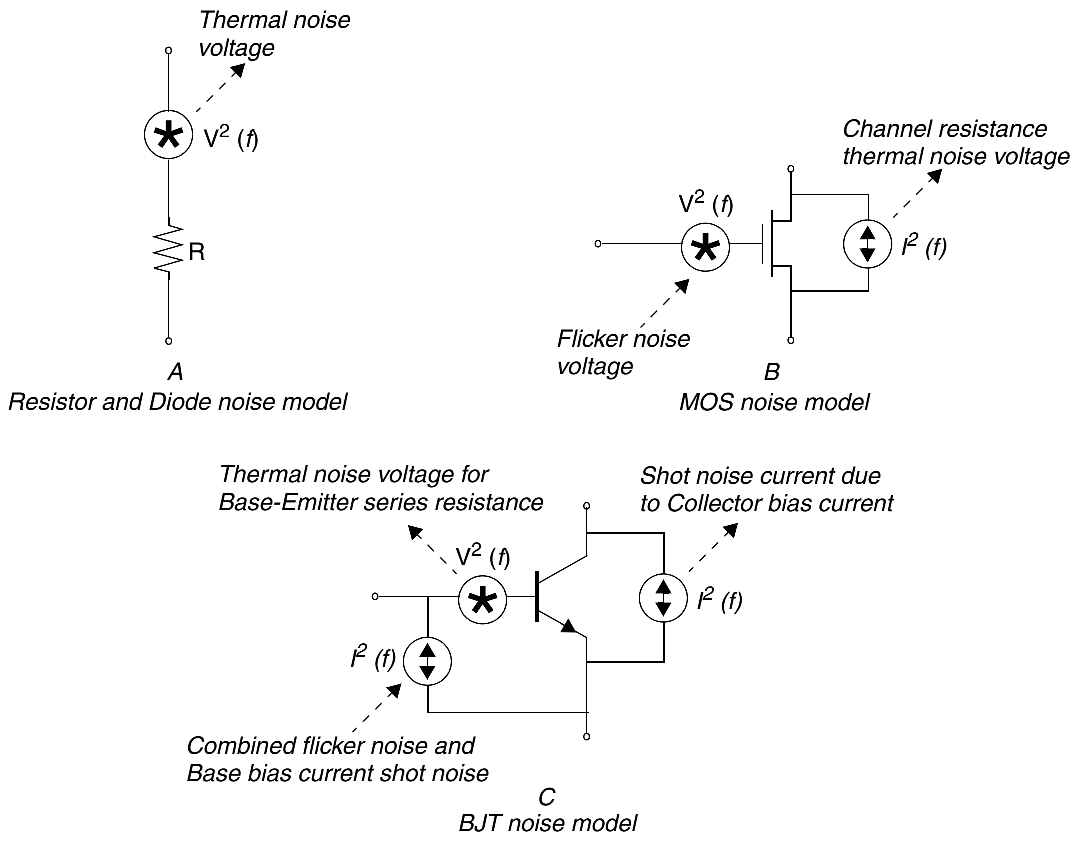

Thermal noise voltage
V2 (f)
R
A
Resistor and Diode noise model
Channel resistance thermal noise voltage
V2 (f)
I2 (f)
Flicker noise voltage
B
MOS noise model
Thermal noise voltage for Base-Emitter series resistance
Shot noise current due to Collector bias current
V2 (f)
I2 (f)
I2 (f)
Combined flicker noise and Base bias current shot noise
C
BJT noise model

Figure 194 Noise generating circuit elements and their noise models

The noise model for a diode is identical to that of a resistor with a resistance of:

$$R_d = kT/\ qI_d$$

Where,
R_d is the equivalent resistance of the diode.
k is Boltzmann's constant.
q is the charge of an electron.
I_d is the bias current of the diode.

HSPICE and HSPICE RF Noise Simulation

To perform noise analysis in HSPICE and HSPICE RF, a `.LIN` statement along with an `.AC` statement must be used. The `.LIN` syntax for the noise analysis is:

```
.LIN noisecalc=1
```

The `.LIN` statement requires that the circuit ports be identified using port elements. The port element will behave either as a noiseless impedance or as a voltage source in series with the port impedance. By default, the port impedance is 50 ohms.

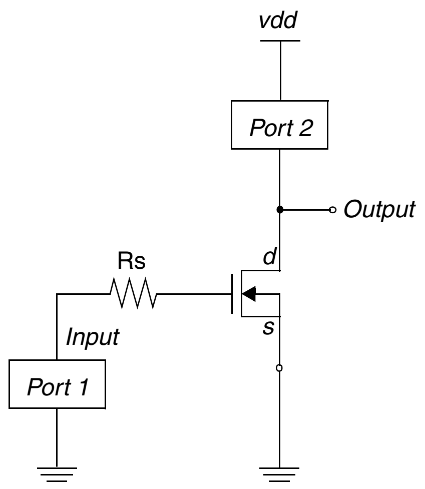
The frequency points at which the noise calculations are performed are the same points defined by the `.AC` statement. The noise calculations for each frequency point will be output to the listing file.

RX Transfer Function

RX is the transfer function from the noise source to the circuit output. Since the noise sources for built-in devices are all current sources with PSDs (power spectral densities) in A^2/Hz, and the output is usually a voltage (specified on `.NOISE` statements), RX is a transimpedance (V/A).

The total output noise voltage is the integrated output noise. The output noise at a given frequency is a PSD in V^2/Hz. This can be integrated over the frequency range (using the `.AC` command) to get a total output noise in V^2, if you take the square root, you get the total output noise in V.

For example (see Example 195 on page 749), take a simple common source NMOS amplifier to show how HSPICE and HSPICE RF calculate the output noise. Note that in this example, the output port element (P2) is used as a pull-up resistor in the circuit.



```
* A Common Source NMOS amplifier
.options list post
.model n_tran nmos level=49 version=3.22 AF=.826 KF=4e-29

vdd vdd 0 DC=5
p1 in 0 port=1 ac=0.1 dc=2.1 z0=50
p2 out vdd port=2 z0=20k
rs in g1 50
m1 out g1 0 0 n_tran l=1.5u w=40u
.ac dec 10 10Meg 10G
.lin noisecalc=1
.print ac v(out) onoise

.end
```

Figure 195 A common source NMOS amplifier and its netlist

The first step in the noise analysis is to set all the signal voltage and current sources to 0. The equivalent circuit for our example, after setting `vdd=0`, is shown in Figure 196.

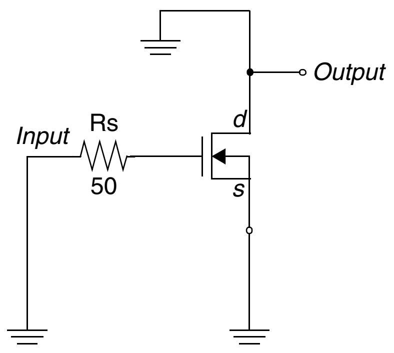


Figure 196 Equivalent circuit after independent V and I sources are set 0

Next, HSPICE pr HSPICE RF models each resistor, diode, and transistor with its noise model, and then calculates the output voltage resulting from the noise signal, one element at a time.

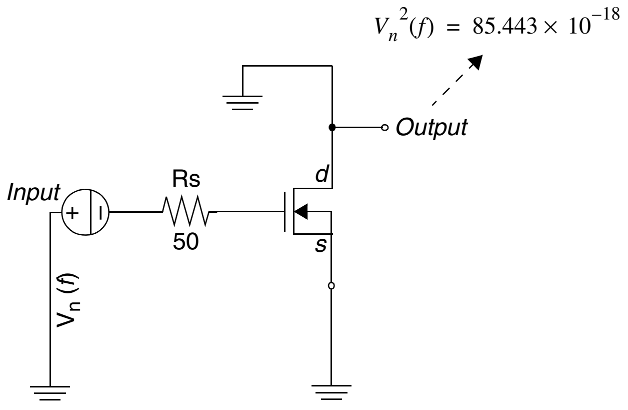
To start, it replaces Rs with its noise model, as shown in Figure 197, and calculates the PSD of the noise voltage as seen at the output port. HSPICE or HSPICE RF reports an output voltage PSD of 85.4443x10^{-18} V^2/Hz, caused by the thermal noise model of Rs. The value rx shown is used to obtain the voltage transfer function from the Rs noise source to the output port:

$$\text{Voltage Transfer Function} = Rs/\ \ rx$$

HSPICE and HSPICE RF use rx for its internal calculations and you need not pay particular attention to it.

Figure 197 Circuit noise model for Rs and analysis output at 100 MHz

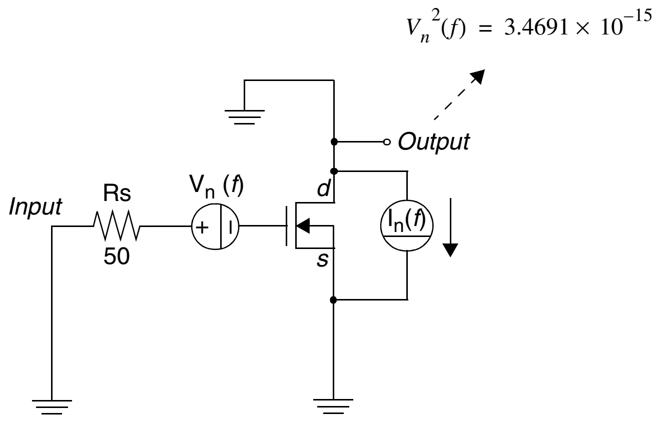
The circuit for calculating the PSD of output noise generated by the NMOS is given in Figure 198.



```
frequency = 100.0000x     hz

 *** resistor squared noise voltages (sq v/hz)

 element     0:rs
    total   85.4443a
       rx 509.3796
```



```
frequency = 100.0000x     hz

 ** mosfet squared noise voltages (sq v/hz)

 element     0:m1
      rd    0.
      rs    0.
      id    3.4544f
      rx   18.4636k
      fn   14.7854a
   total   3.4691f
```

Figure 198 Circuit noise model for NMOS and analysis output

The total PSD at 100 MHz will be the sum of all individual PSDs (the square of noise signals add):

$$V_{nTotal}^2\ (f = 100MHz) = 85.4443 \times 10^{-18} + 3.4691 \times 10^{-15} = 3.5546 \times 10^{-15} \qquad V^2/\ Hz$$

$$V_{nTotal}(f = 100MHz) = \sqrt{3.5546 \times 10^{-15}} = 59.6204 \times 10^{-9} \qquad V/\ \sqrt{Hz}$$

It is common to represent the total noise generated by a circuit as the equivalent input noise (otherwise called *input referred noise*). The input referred noise voltage/current is the voltage/current that if applied at the input of the noise-free circuit, would have generated the same output voltage as the total output noise voltage.

To calculate the equivalent input noise signal, you have to obtain the transfer function from input to output. For the example circuit above, the transfer function will be:

$$TF = \frac{\text{Output Voltage}}{\text{Input Voltage}} = \frac{V(out)}{VS}$$

In the case where the input source is a current source instead of a voltage source, the TF would be the ratio of output voltage over input current.

The equivalent input noise voltage (or current in case of the input current source) is given by:

$$\text{Input Referred Noise} = \frac{\text{Total Output Noise Voltage}}{TF}$$

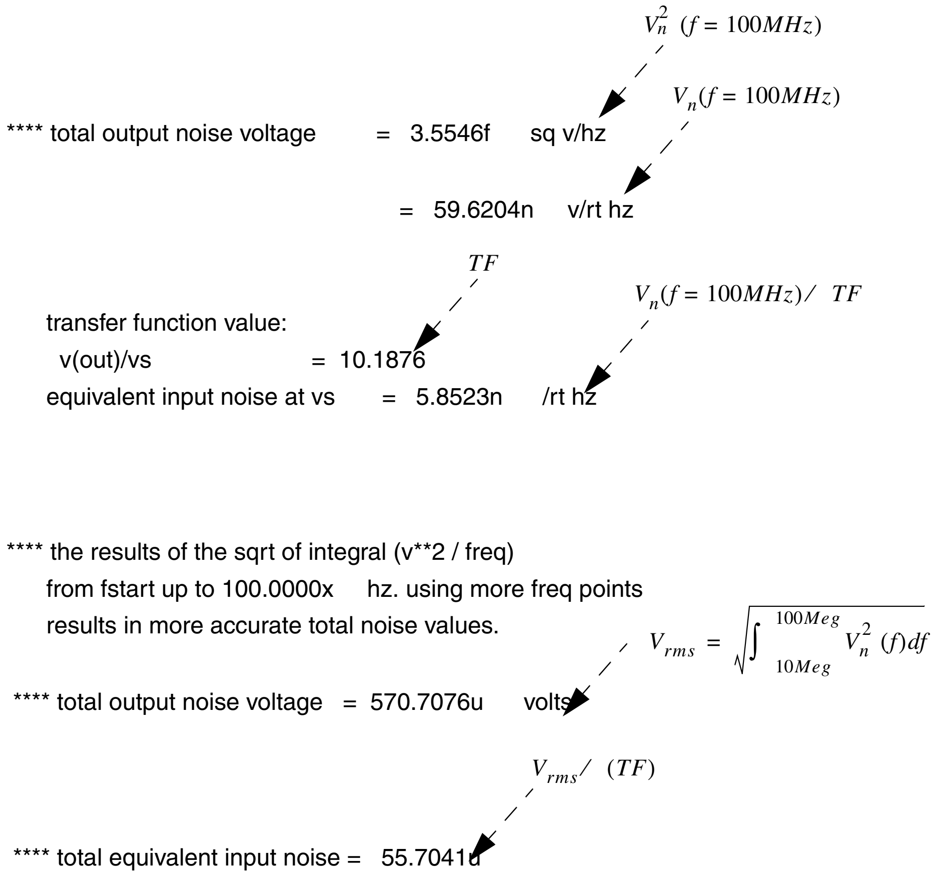
The input referred noise is useful to calculate the circuit's “noise figure,” which is a measure of how much a circuit is generating inherent noise, a large noise figure indicates a high level of noise generation by the circuit. Noise figure is defined as:

$$\text{Noise Figure} = 10 \cdot \log \cdot (\text{Noise Factor})$$

Where Noise Factor is given by:

$$\text{Noise Factor} = 1 + \frac{\text{Input Referred Inherent Noise Power}}{\text{Power of External Noise at the Input}}$$

Example 199 on page 753 shows the HSPICE noise analysis summary output for the above circuit.



```
**** total output noise voltage        =  3.5546f      sq v/hz

                                          =  59.6204n     v/rt hz

     transfer function value:
       v(out)/vs                    =  10.1876
     equivalent input noise at vs        =  5.8523n      /rt hz


**** the results of the sqrt of integral (v**2 / freq)
     from fstart up to 100.0000x     hz. using more freq points
     results in more accurate total noise values.

 **** total output noise voltage   =  570.7076u      volts

 **** total equivalent input noise =   55.7041u
```

Figure 199 Noise summary for noise performed at 100 MHz

Summary

To summarize,

- HSPICE and HSPICE RF model noisy resistors, diodes, and transistors as noiseless elements connected to noise generating voltage and current sources.
- To calculate the total output noise PSD, HSPICE and HSPICE RF set all signal and supply sources to zero. It then, one by one, replaces the noise generating elements with their equivalent noise model, calculates the noise

voltage PSD at the output caused by the element's noise source, which is similar to an .AC analysis. It then repeats the same process for the next noise-generating element.

- Once the output noise PSDs are calculated for all elements, HSPICE adds the PSDs together to obtain the total output noise voltage PSD.
- The RMS output noise voltage and the input referred noise can be deduced from the total output noise PSD.

31

Using Verilog-A

Describes how to use Verilog-A in HSPICE and HSPICE RF simulations.

Verilog-A is used to create and use analog behavioral descriptions that encapsulate high-level behavioral and structural descriptions of systems and components.

The language allows the behavior of each model, or module, to be described mathematically in terms of its ports and parameters applied to an instance of the module. A module can be defined at a level of abstraction appropriate for the model and analysis, including architectural design, and verification. Verilog-A supports both a top-down design as well as a bottom-up verification methodology.

Verilog-A was derived from the IEEE 1364 Verilog Hardware Description Language (HDL) specification and is intended for describing behavior in analog systems. The Verilog-A language that HSPICE supports is compliant with *Verilog-AMS Language Reference Manual, Version 2.2,* with limitations listed in Unsupported Language Features on page 798.

The Verilog-A implementation in HSPICE supports a mixed design of Verilog-A descriptions and transistor-level SPICE netlists with a simple use model. Most analysis features available in HSPICE are supported for Verilog-A based devices, including AC, DC, transient analysis, statistical analysis, and optimization.

The HSPICE RF supported analysis types are HB, HBOSC, HBAC, HBNOISE, HBXF, PHASENOISE, SN, SNOSC and ENV.

Note:

Some restrictions for Verilog-A models used with RF simulation algorithms apply. For example:

Verilog-A modules that are time dependent cannot be used for HB or SN unless the time dependence is periodic with a period that matches the HB or SN setup.

Verilog-A modules with “internal states” are not guaranteed to work correctly in HB or SN because the internal state cannot be tracked by the engine, so HB or SN may think it is converged to a periodic steady-state even though the internal state may not be in periodic steady state.

Some event-driven constructs in Verilog-A may not be compatible with HB.

These are standard restrictions for Verilog-A in periodic steady-state analysis and are the same as other RF simulators that use Verilog-A.

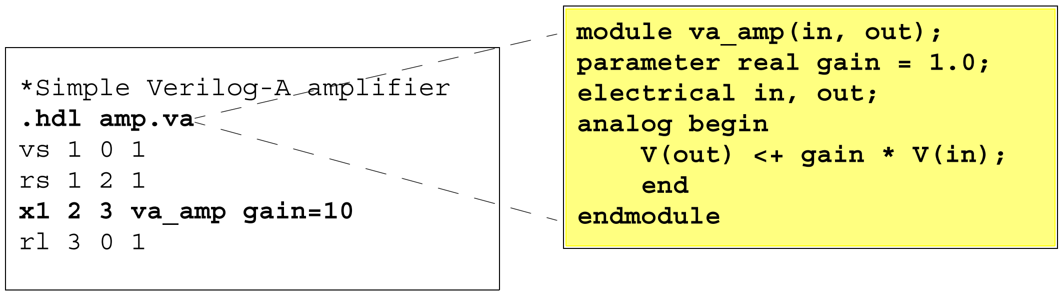
These topics are covered in the following sections:

Note:

In the context of this chapter, “HSPICE” refers to both HSPICE and HSPICE RF unless noted otherwise.

Getting Started

This section explains how to get started using a compact device model written in Verilog-A in HSPICE.



```
*Simple Verilog-A amplifier
.hdl amp.va
vs 1 0 1
rs 1 2 1
x1 2 3 va_amp gain=10
rl 3 0 1
```

```
module va_amp(in, out);
parameter real gain = 1.0;
electrical in, out;
analog begin
    V(out) <+ gain * V(in);
    end
endmodule
```

Figure 200 HSPICE and Verilog-A

Verilog-A devices use the following conventions:

- Modules are loaded into the simulator with either the `.HDL` netlist command or the `-hdl` command-line option. The command-line is not supported in HSPICE RF.
- Modules are instantiated in the same manner as HSPICE subcircuits. The first character for the name of instance should be “X”.
- Instance and model parameters can be modified in the same way as other HSPICE instances.
- Module names should not conflict with any HSPICE built-in device keyword (see Using Model Cards with Verilog-A Modules on page 779). If this happens, HSPICE issues a warning message and ignores the Verilog-A module definition.
- Node voltages and branch currents can be output using conventional output commands.

To run an HSPICE Verilog-A simulation, you need to run the “hspice” script, which is located in the $<*installdir*>/hspice_2008.03/bin/hspice, regardless of the platform. For example,

```
/installed_hspice/hspice_2008.03/bin/hspice
```

The following example illustrates how a compact device model written in Verilog-A can be analyzed with HSPICE.

Example: JFET Compact Device Model

HSPICE contains a large number of compact device models coded natively in the simulator. Verilog-A provides a convenient method to introduce new compact models. The JFET device model uses a simple expression to relate the source-drain current to the gate voltage.

The simplified Verilog-A description of this model is shown.

```
`include "constants.vams"
`include "disciplines.vams"
module jfet(d, g, s);
parameter real Vto = -2.0 from (-inf:inf);  // Threshold voltage
parameter real Beta = 1.0e-4 from [0:inf);// Transconductance
parameter real Lambda = 0.0 from [0:inf); // Channel modulation
  electrical d, g, s;
  real Id, Vgs, Vds;
  analog begin
      Vgs = V(g,s);
      Vds = V(d,s);
      if (Vds <= Vgs-Vto)
          Id = Beta*(1+Lambda*Vds)*Vds*(2*(Vgs-Vto)- Vds);
      else if (Vgs-Vto < Vds)
          Id = Beta*(1+Lambda*Vds)*(Vgs-Vto)*(Vgs-Vto);
      I(d,s) <+ Id;
  end
endmodule
```

In this example the module name is `jfet` and the module has three ports, named `d`, `g`, and `s`. Three parameters, `Vto`, `Beta`, and `Lambda`, can be passed in from the netlist. The electrical behavior is defined between the `analog begin` and `end` statements. The node voltages across the gate to source and drain to source is accessed and assigned to the variables `Vgs` and `Vgd`. These values are used to determine the drain-source current, `Id`. The calculated current is contributed to the branch from `d` to `s` in the final statement using the contribution operator, `<+`.

This Verilog-A module is loaded into HSPICE with an `.HDL` command in the netlist. The device is then instantiated using the X prefix for the device name. The connectivity, module name, and parameter assignments follow the format of a subcircuit device. The following instantiation line in the netlist is for this device:

```
x1 drain gate source jfet Beta=1.1e-4 lambda=0.01
```

The nodes drain, gate, and source are mapped to the ports d, g, s in the same order as defined in the module definition. Any parameters in the instantiation line are passed to the module; otherwise, the default value defined on the parameter declaration line is used. The parameter declaration allows ranges and exclusions to be easily defined.

The parameter passed in from the netlist is tested during the simulation and a run- time error occurs if the parameter is out of the allowed range.

The device is used in the HSPICE netlist in exactly the same manner that a built-in device is used. The netlist example shown performs a simple DC-IV analysis.

```
Verilog-A version of the SPICE JFET
.hdl jfet.va
.options post=1

VCC Drain  0  3.0
VG  Gate   0  0.5
VS  Source 0  0.0

X1 Drain Gate Source jfet Vto=-2.0 Beta=1.1e-4 Lambda=0.01
.dc VCC 0.0 4.0 0.01  VG -2.0 0.0 0.5
.print I(VCC)
.end
```

When the simulation is performed, the program compiles the Verilog-A source file into a compiled object file. This object file is automatically cached and subsequent simulations do not require the compile step unless the Verilog-A source file is modified. After simulation, HSPICE outputs the data in the same fashion as other devices. In this example the drain-source current is plotted as a function of Vds and parameterized by Vgs. Figure 201 displays the plot of the drain-source current for this model.

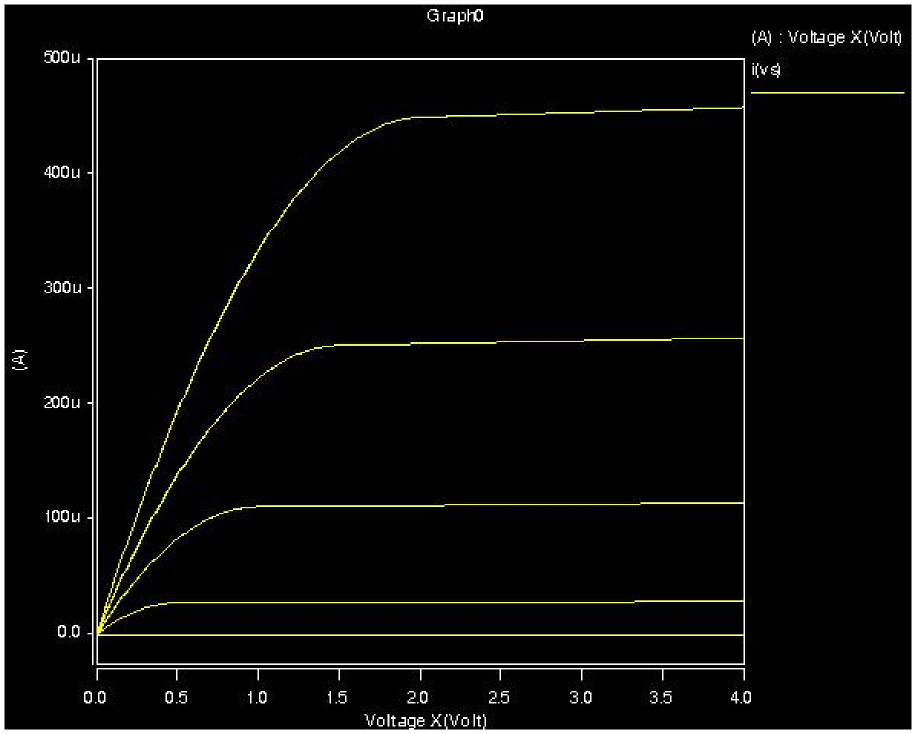


Figure 201 IV Characteristics of a Verilog-A JFET

Introduction to Verilog-A

The following is a Verilog-A module example provides an overview of the language. See the Verilog-AMS LRM 2.2 from Accellera for syntax and usage details.

Verilog-A Module Template

The following basic template illustrates basic of the language's features.

```
`include "disciplines.vams" // Natures and disciplines
`include "constants.vams"   // Common physical and math
                            // constants
module my_model(port1, bus);
  electrical port1;
  electrical [0:1] bus;
  parameter real real_param = 1.0 from [0:inf);
  parameter integer int_param = 1 from [-1:1] exclude 0;
  real real_var;
  analog begin
    @ ( initial_step ) begin
    /*    Code inside an initial_step block is executed
      at the first step of each analysis */
     end

     real_var = I(port1); // Current port1 to ground
     V(bus[0], bus[1]) <+ real_var * real_param * int_param;

     @ ( final_step ) begin
     /* Code inside an final_step block is executed
      at the last step of each analysis */
      end
   end
endmodule
```

Data Types

Several Verilog-A data types are available. The parameter type is used to pass values from the netlist to the module.

Table 72 Verilog-A Data Types

Data Type	Description
attribute	A mechanism for specifying properties about objects, statements, and groups of statements that may be used to control the operation or behavior of the tool. `(* attr_spec {, attr_spec } *)`
genvar	Special integer-valued variable for behavioral looping constructs `genvar genvar_name {, genvar_name};`
integer	Discrete numerical type `integer integer_name {, integer_name};`

Table 72 Verilog-A Data Types (Continued)

Data Type	Description
local parameters	Identified by the localparam keyword, local parameters are identical to parameters except that they cannot directly be modified with the defparam statement or by the ordered or named parameter value assignment. Local parameters can be assigned to a constant expression containing a parameter that can be modified with the defparam statement or by the ordered or named parameter value assignment.
parameter	Attribute that indicates data type is determined at module instantiation. `parameter [{integer | real | string}] param_name = default_value [ from[ range_begin:range_end ] [ exclude exclude_value_or_range ] ] ;`
parameter aliases	Aliases can be defined for parameters. This allows an alternate name to be used when overriding module parameter values; for example, `parameter real dtemp = 0 from [-‘P_CELSIUS0:inf);` `aliasparam trise = dtemp;` Then the following two instantiations of the module are valid: `nmos #(.trise(5)) m1(.d(d), .g(g), .s(s), .b(b));` `nmos #(.dtemp(5)) m2(.d(d), .g(g), .s(s), .b(b));` And the value of the parameter dtemp, as used in the module equations for both instances, is 5.
real	Continuous numerical type `real real_name {, real_name};`
string parameters	String parameters can be declared. Strings are useful for parameters that act as flags, where the correspondence between numerical values and the flag values may not be obvious. The set of allowed values for the string can be specified as a comma-separated list of strings inside curly braces.

Analog Operators and Filters

Analog operators and filters maintain memory states of past behavior. They can not be used in an analog function.

Table 73 Verilog-A Analog Operators and Filters

Operator	**Description**
Time derivative	The ddt operator computes the time derivative of its argument. `ddt(expr)`
Time integral	The idt operator computes the time-integral of its argument. `idt(expr, [ic [ , assert [ , abstol ] ] ] )` See the example following this table.
Derivative	The ddx operator provides access to symbolically-computed partial derivatives of expressions in the analog block. `ddx(expr, V(a)))`
Linear time delay	absdelay() implements the absolute transport delay for continuous waveform. `absdelay(input, time_delay [, maxdelay ])`

Discrete waveform filters	The transition filter smooths out piecewise linear waveforms.
	`transition(expr[,td[,rise_time[,fall_time [,time_tol]]]])`
	The slew analog operator bounds the rate of change (slope) of the waveform.
	`slew(expr[,max_pos_slew_rate [,max_neg_slew_rate]])`
	The last_crossing() function returns a real value representing the simulation time when a signal expression last crossed zero.
	`last_crossing(expr, direction)`

Laplace transform filters	laplace_zd() implements the zero-denominator form of the Laplace transform filter. The laplace_np() implements the numerator-pole form of the Laplace transform filter. laplace_nd() implements the numerator- denominator form of the Laplace transform filter. laplace_zp() implements the zero-pole form of the Laplace transform filter. `laplace_*(expr, u, v)`

Table 73 Verilog-A Analog Operators and Filters (Continued)

Operator	Description
Z-transform filters	The Z-transform filters implement linear discrete-time filters. All Z-transform filters share three common arguments: T, t, and t0. T specifies the period of the filter, is mandatory, and must be positive. t specifies the transition time, is optional, and must be nonnegative. ▪ zi_zd() implements the zero-denominator form of the Z-transform filter. ▪ zi_np() implements the numerator-pole form of the Z-transform filter. ▪ zi_nd() implements the numerator-denominator form of the Z-transform filter. ▪ zi_zp() implements the zero-pole form of the Z-transform filter. `zi_*( expr , u , v , T [ , t [ , t0 ] ] )`
Limited Exponential	Limits exponential argument change from one iteration to the next. `limexp(arg)`

Example: Using idt to Perform Time Integral of an Argument

You can set up real time integration (instantaneous time integral) by using the “time integral operator” in Verilog-A and simulating the Verilog-A block in HSPICE. The syntax is:
`idt(expr, [ic])`

The `ic` term is an initial condition and it aids in convergence. For example:

```
 v(out) <+ K*idt(i(node_name), 0.0);
```

This example calculates the integral of the specified node_name, with an initial condition of 0.0. The initial condition allows the “idt” operator to return the value of the initial condition during the operating point analysis. To avoid convergence problems, be sure to specify an initial condition. If an initial condition is not supplied, the idt function multiplies the argument by infinity during the operating point analysis. An initial condition is not necessary *only* when you have a system in which the feedback forces the argument of idt to be zero.

Mathematical Functions

The following mathematical functions are available when using HSPICE with Verilog-A.

Table 74 Verilog-A Mathematical Functions

Function	Description	Domain	Return Value
ln()	natural log	x>0	real
log(x)	log base 10	x>0	real
exp(x)	exponential	x<80	real
sqrt(x)	square root	x>=0	real
min(x,y)	Minimum of x and y	all x, y	If either is real, returns real, otherwise returns the type of x,y.
max(x,y)	Maximum of x and y	all x, y	If either is real, returns real, otherwise returns the type of x,y.
abs(x)	absolute value	all x	same as x
pow(x,y)	x**y	if x>=0, all y; if x<0, int(y)	real
floor(x)	Floor	all x	real
ceil(x)	Ceiling	all x	real

Transcendental Functions

The following mathematical functions are available when using HSPICE with Verilog-A.

Table 75 Verilog-A Transcendental Function

Function	Description	Domain
sin(x)	sine	all x

Table 75 Verilog-A Transcendental Function (Continued)

Function	**Description**	**Domain**
cos(x)	cosine	all x
tan(x)	tangent	x != n (pi/2), n is odd
asin(x)	arc-sine	-1<= x <= 1
acos(x)	arc-cosine	-1<= x <= 1
atan(x)	arc-tangent	all x
atan2(x,y)	arc-tangent of x/y	all x, all y
hypot(x,y)	sqrt(x**2 + y**2)	all x, all y
sinh(x)	hyperbolic sine	x < 80
cosh(x)	hyperbolic cosine	x < 80
tanh(x)	hyperbolic tangent	all x
asinh(x)	arc-hyperbolic sine	all x
acosh(x)	arc-hyperbolic cosine	x >= 1
atanh(x)	arch-hyperbolic tangent	-1 <= x <= 1

AC Analysis Stimuli

The AC stimulus function produces a sinusoidal stimulus for use during a small-signal analysis:

```
ac_stim([analysis_name [, mag [, phase]]])
```

Noise Functions

The noise functions contribute noise during small-signal analyses. The functions have an optional name, which the simulator uses to tabularize the contributions.

Table 76 Verilog-A Noise Functions

Function	Description
White Noise	Generates a frequency-independent noise of power pwr. `white_noise(pwr[,name])`
Flicker Noise	Generates a frequency-dependent noise of power pwr at 1 Hz which varies in proportion to the expression 1/fexp. `flicker_noise(pwr,exp[,name])`
Noise Table	Defines noise via a piecewise-linear function of frequency. Vector is frequency, power pairs in ascending frequencies. `Noise_table(vector[,name])`

Analog Events

The following analog events are available when using HSPICE with Verilog-A.

Table 77 Verilog-A Analog Event-Controlled Statements

Function	Description
Initial Step	Event trigger at first step of an analysis. `@(initial_step[(list_of_analyses)])`
Final Step	Event trigger at last step of an analysis. `@(final_step[(list_of_analyses)])`
Cross	Zero crossing threshold detection. `cross(expr[,dir[,time_tol[,expr_tol]]])`
Timer	Generates analog event at specific time. `timer(start_time[,period[,time_tol]])`

Table 77 Verilog-A Analog Event-Controlled Statements (Continued)

Function	Description
Above	Generates an event when a specified expression becomes greater than or equal to zero. `above(expr[,time_tol[,expr_tol]])`

Timestep and Simulator Control

These functions provide a mechanism to alert the simulator to discontinuities or to limit the time step.

Table 78 Verilog-A Discontinuity and Time Step Limit Functions

Function	Description
Bound time step	Controls the maximum time step the simulator takes during a transient simulation. `$bound_step( expression );`
Announce discontinuity	Provides the simulator information about known discontinuities to provide help for simulator convergence algorithms. `$discontinuity [ ( constant_expression ) ] ;`

Limiting functions	The `$limit()` function provides a method to indicate device m model nonlinearities besides the exponential to the simulator and, if necessary, recommend a function to use to limit the change of its output from iteration to nitration.
	`$limit(access_function_reference`
	`   [, function_identifier [, argument_list]]);`

System Tasks and I/O Functions

System functions provide access to system-level tasks as well as access to simulator information.

Table 79 Verilog-A System Tasks and I/O Functions

Function	Description
`$param_given()`	Returns 1 if the parameter was overridden by a module instance parameter value assignment and 0, otherwise.
`$port_connected()`	Determines whether a connection was specified for a port. `$port_connected ( port_scalar_expression )`
`$table_model()`	Models behavior of a system by interpolating between data points that are samples of that system's behavior. `$table_model(`*`table_inputs`*`, `*`table_data_source`*`,` *`table_control_string`*`);` where: ▪ `table_inputs` is an (optionally multi-dimensional) expression. ▪ `table_data_source` is either a string indicating the name of the file holding the table data or an array literal or the name of an array. ▪ `table_control_string` is a two part string. The first character is an integer indicating the degrees of the spline interpolation (either 1 | 2| 3). The second part of the control string consists of one or two characters (either C | L | E) indicating the type of extrapolation mode at the beginning and end of the data. ▪ Table interpolation character degree: 1 = Linear spline (degree 1) 2 = Quadratic spline (degree 2) 3 = Cubic spline (degree 3) Table extrapolation character type: C = Clamp extrapolation L = Linear extrapolation (default) E = Error condition Note: Quadratic and cubic spline interpolation or clamped extrapolation are not part of the LRM 2.2.

Table 79 Verilog-A System Tasks and I/O Functions (Continued)

Function	Description

`$strobe()`
`$display()`
`$write()`
	Displays simulation data when the simulator has converged on a solution for all nodes using a `printf()` style format.
	`$strobe(args);`

`$fopen()`	Opens a file for writing and assigns it to an associated channel.`multi_channel_desc = $fopen("file");`
	Starting with the 2006.09 release, `$fopen()` accepts a second argument, a mode string. The mode string can be either:
	`"w"` - overwrite contents if the file exists. `"w"` is the default mode;
	or `"a"` - append to contents if the file exists. For example,
	`hdl1 = $fopen("append_mode.txt", "a");`
	`hdl2 = $fopen("overwrite_mode.txt", "w");`
	
	The following expansions are not part of the LRM2.2.
	File names can use an escape sequence `"%I"` to represent the name of the instance in the file name. For example,
	hdl1=$fopen("%I_data.txt")
	When called from an instance, X2 will look for a file called *X2_data.txt*. Also, the tilde "~" can be used to reference the value of the environment variable $HOME (or %HOMEDRIVE%%HOMEPATH% on win32). The tilde will be expanded to the full path of the home directory, so if `$HOME=/user/default`, then
	hdl1=$fopen("~/datafiles/%I_data.txt")
	called from instance X1 will look for a file "`/user/default/datafiles/X1_data.txt`".

`$fclose()`	Closes a file from a previously-opened channel(s). `$fclose(multi_channel_descriptor);`

`$fstrobe()`
`$fdisplay()`
`$fwrite()`
	Writes simulation data to an opened channel(s) when the simulator has converged. Follows format for $strobe.
	`$fstrobe(multi_channel_descriptor, "information to be written");`

`$dist_functions()`	Probabilistic distribution functions
`$debug()`	Provides the capability to display simulation data while the analog simulator is solving the equations.

Table 79 Verilog-A System Tasks and I/O Functions (Continued)

Function	Description

`$random()`	Provides a mechanism for generating random numbers.
	`random_function`
	`$random [ ( seed [, type_string] ) ] ;`

`$mfactor()`
`$xposition()`
`$yposition()`
`$angle()`
`$hflip()`
`$vflip()`
	These functions return hierarchical information about the instance of a module or paramset.
	`$mfactor` is the shunt multiplicity factor of the instance, that is, the number of identical devices that should be combined in parallel and modeled.
	`$xposition` and `$yposition` are the offsets, in meters, of the location of the center of the instance.
	`$angle` indicates that the instance has been rotated some number of degrees in the counter-clockwise directions
	`$hflip` and `$vflip` are used to indicate that the instance has been mirrored about its center.

Table 79 Verilog-A System Tasks and I/O Functions (Continued)

Function	Description

`$fscanf()`	Non-LRM 2.2 function. Provides a means to read data from files.
	Syntax:
	[integer_return_value =] $fscanf
	(multi_channel_descriptor, format_string
	[, list_of_arguments]);
	where:
	`multi_channel_descriptor` is the multichannel descriptor returned by the `$fopen` command when the file was opened;
	`format_string` is a string describing how the data will be matched;
	`list_of_arguments` is optional and comma-separated, where the read data matching the list of arguments will be stored.
	
	The allowed commands for the `format_string` are the same as those available for the `$strobe()` function argument. Each data value read will be sequentially matched to the corresponding argument in the `list_of_arguments`.
	
	The `list_of_arguments` must have the correct number of variables to match the data value types in the `format_string`. The optional return value of the function is set to the number of valid arguments read during the operation; if the return value is not used, a warning is issued.
	
	The channel specified in the `multi_channel_descriptor` must be assigned to an open file by using the `$fopen()` function.
	Example:
	The following example reads an integer, real, and character value from the file *data.txt* and puts the values in `int_value`, `real_value`, and `char_value`, respectively. The integer valid is set to the number of valid reads, in this case, 3.
	`integer multi_ch_desc, valid, int_value,`
	`   char_value;`
	`real real_value;`
	`@(initial_step)`
	`   multi_ch_desc = $fopen ("data.txt", "r");`
	`valid = $fscanf (multi_ch_desc, "%d %e %c",`
	`   int_value, real_value, char_value);`

Table 79 Verilog-A System Tasks and I/O Functions (Continued)

Function	Description

`$fflush[()]`	Non-LRM 2.2 function. Writes any buffered output to the file(s) specified by the optional file descriptor. If no argument is supplied, it writes any buffered output to all open files.
	Syntax:
	`$fflush [( multi_channel_descriptor ) ] ;`
	where: `multi_channel_descriptor` is an integer that represents opened file(s).
	Examples:
	The following examples illustrate typical uses for the `$fflush` command.
	`$fflush (multi_ch_desc);`
	`$fflush ( );`
	`$fflush;`

Simulator Environment Functions

The environment parameter functions return simulator environment information to the module. Return circuit ambient temperature in Kelvin.

Table 80 Verilog-A Environment Parameter Functions

Function	Description
Circuit temperature	Returns circuit ambient temperature in Kelvin. `$temperature`
Time	Returns absolute time in seconds. `$abstime`
Thermal voltage	$vt can optionally have Temperature (in Kelvin) as an input argument and returns the thermal voltage (kT/q) at the given temperature. $vt without the optional input temperature argument returns the thermal voltage using $temperature. `$vt [ (Temperature) ]`
Analysis flag	Returns true (1) if current analysis matches any one of the passed arguments. `$analysis(str {, str } )`

Table 80 Verilog-A Environment Parameter Functions (Continued)

Function	Description
Simulation parameter	Returns the value of the named specified simulation parameter. `gmin = $simparam("gmin", 1.0);` Returns simulator version information. `$simparam("simulatorVersion"));`

Module Hierarchy

Modules can instantiate other modules so that networks of modules can be constructed. Structural statements are used inside the module block but cannot be used inside the analog block.

```
module_name #({.param1(expr){, .param2(expr})}
instance_name ({node {, node});
```

Example

```
my_src #(.fstart(100), .ramp(z)) u1 (plus, minus);
```

Parameter Sets

Parameter sets (paramsets) bring the concept of model cards directly into Verilog-A. Paramsets allow sharing of a set of parameters among several modules. They may also be chained allowing a common parameter set to be used.

Example

```
paramset nch my_nmos; // default paramset
parameter real l=1u from [0.25u:inf);
parameter real w=1u from [0.2u:inf);
.l=l; .w=w; .ad=w*0.5u; .as=w*0.5u;
.level=3; .kp=5e-5; .tox=3e-8; .u0=650; .nsub=1.3e17;
.vmax=0; .tpg=1; .nfs=0.8e12;
endparamset
```

Simulation with Verilog-A Modules

When simulating with Verilog-A in HSPICE, you need to have the following basic input files:

- HSPICE netlist/model card (Mandatory)
- Verilog-A model file (for example, **.va* or **.vams* file) or Compiled Model Library file (**.cml*) (Mandatory)
- HSPICE Verilog-A feature setup options (Optional, but mandatory under certain conditions)

Basic output files:

- HSPICE standard output files
- The **.val* file, Verilog-A log file, which contains Verilog-A specific message from compiling and simulating phase. The contents of **.val* file is also echoed to the **.lis* file.
- Compiled Verilog-A code (*.cml* file) (when Verilog-A modules are compiled manually).

Loading Verilog-A Devices

This section describes loading Verilog-A modules into HSPICE and specifying cell names for Verilog-A definitions. A module must be loaded before it can be instantiated.

Verilog modules are loaded into HSPICE in one of two ways:

- by including an `.HDL` statement in an HSPICE netlist
- by using the `-hdl` command-line option (not supported in HSPICE RF).

Files can be in the current directory or specified via an absolute or relative path. The Verilog-A file is assumed to have a **.va* extension when only a prefix is provided. For example, `.hdl "model"` looks for a model.va file and *not* a file named “model”.

Use the `-vamodel` command-line option to specify cell names for Verilog-A definitions (not supported in HSPICE RF).

For a description of the `.hdl` statement, see the .HDL command in the *HSPICE Reference Manual: Commands and Control Options*. For a description of the `-hdl` and `-vamodel` command-line options, see “HSPICE Command Syntax” in Chapter 1 of the *HSPICE Reference Manual: Commands and Control Options*.

.hdl Command Module Selection and Module Alias

The `.hdl` command uses the following syntax,

```
.hdl file_name [module_name] [module_alias]
```

The optional `module_name` and `module_alias` parameters were introduced in HSPICE version Z-2007.03 and do not apply to HSPICE RF.

If a module is specified, then only that module is loaded from the specified Verilog-A or CML file. If the module is not found or if the module specification is not case unique, then an error is generated.

If a `module_alias` is specified (in addition to a module name), then that module is loaded into the system using the alias in place of the module name defined in the Verilog-A source file. Thereafter, any reference to the module is made using its alias. The system behaves as if the module had the alias as its module name. A module may be loaded with any number of aliases in addition to being loaded without an alias.

The `module_alias` is useful when loading modules of the same name from different files. For example, if you have a module `res` in two libraries, such as`'fast.va'` and `'slow.va'`, then you can write,

```
.hdl 'fast.va' 'res' 'fast_res'
.hdl 'slow.va' 'res' 'slow_res'
...
x1 1 2 fast_res r=1
x2 2 0 slow_res r =1
...
```

The `module_name` and `module_alias` arguments can be specified without quotes and single or double quotes. Tokens after the module alias are ignored.

The same Verilog-A case insensitivity rules used for module and parameter names apply to both the `module_name` and `module_alias` arguments, and the same module override logic applies.

Verilog-A File Search Path

During a simulation, HSPICE searches in the current directory for Verilog-A files. You can also provide a search path via either the `-hdlpath` command-line option (not supported in HSPICE RF) or the `HSP_HDL_PATH` environment variable to have HSPICE search other directories for the files. The `-hdlpath`

HSPICE command-line option is provided for HSPICE Verilog-A use only, which defines the search path specifically for Verilog-A files.

For a description of the `-hdlpath` command-line option, see "HSPICE Command Syntax" in the *HSPICE Reference Manual: Commands and Control Options*.

When a Verilog-A file cannot be found in the current working directory or the directory defined by `-hdlpath`, or there is no `-hdlpath` defined, HSPICE searches directory defined by `HSP_HDL_PATH` for the Verilog-A file.

The directory search order for Verilog-A files is:

1. Current working directory
2. Path defined by `-hdlpath`
3. Path defined by `HSP_HDL_PATH`

The path defined by either `-hdlpath` or `HSP_HDL_PATH` can consist a set of directory names. The path separator must follow HSPICE conventions or platform conventions (";" on UNIX). Path entries that do not exist are ignored and no error or warning messages are issued.

Example

This example assumes the c-shell is used.

```
setenv HSP_HDL_PATH  /lib_path/veriloga
```

Verilog-A File Loading Considerations

Several restrictions and issues must be considered when loading Verilog-A modules:

- You can place an `.HDL` statement anywhere in the top-level circuit. All Verilog-A modules are loaded into the system prior to any device instantiation.
- An `.HDL` statement is not allowed inside a `.subckt` or `IF-ELSEIF-ELSE` block; otherwise, the simulation will exit with an error message.
- When a module to be loaded has the same name as a previously-loaded module, or the names differ in case only, the latter one is ignored and the simulator issues a warning message.
- If a Verilog-A module file is not found or the Compiled Model Library (CML) file has an incompatible version, the simulation exits and an error message is issued.

Note:

There is no method to encrypt Verilog-A models in HSPICE.

You could use the hsp-vacomp utility to compile the Verilog-A into a *.cml* file and send that to the customer. However, the *.cml* file is platform and HSPICE version-specific. Therefore, you will probably need to generate multiple .cml files for your customer.

In addition due to the high level of abstraction, it is very difficult to reverse engineer the actual circuit

topology.

Instantiating Verilog-A Devices

Verilog-A devices are X elements. A Verilog-A device can have zero or more nodes and can accept zero or more parameter assignments. Verilog-A devices also support the concept of a model card.

Syntax

```
X<inst> <nodes> moduleName | ModelName param=value |
   param=str('strvalue') | param=str(strparm)
```

Where, `strparm` can be defined by a `.PARAM` command.

Verilog-A module definitions are unique in each HSPICE simulation. A Verilog-A module that matches the name, or differs only in case of a previously loaded module is ignored. A Verilog-A module definition is ignored if its name conflicts with HSPICE built-in models. Verilog-A devices parameters can be re-assigned In either instance statements or model card statements (see “Using Model Cards with Verilog-A Modules” in this chapter) invalid parameters that are not predefined in the Verilog-A module file are ignored. HSPICE issues warning messages on those invalid parameters.

Parameters that are defined in a Verilog-A module file are treated as device parameters to that particular module, they are not netlist parameters and thus can not be overridden by the `.PARAM` command directly. For example:

```
*
.param w=1u
.hdl "nmos_va.va"
**  nmos_va is the Verilog-A module name
**  'w', 'l' are nmos_va module parameters.

x1 d g s b nmos_va w=0.35u l=0.35u
x2 d g s b nmos_va w=w l=0.35u
...
```

The 'w' for x1 is 0.35u, it is not overridden by the `.param` command. Only 'w' of x2 is re-assigned to '1u' because its device parameter 'w' has been pre-assigned to a netlist parameter 'w', and thus can be overridden by a `.param` command.

Using Model Cards with Verilog-A Modules

The HSPICE Verilog-A device supports the concept of model cards, with similar usage to HSPICE standard built-in devices. The Verilog-A module name should not conflict with the following built-in device keywords. In the event of a conflict, HSPICE issues a warning message and ignores the module definition.

AMP, C, CORE, D, L, NJF, NMOS, NPN, OPT, PJF, PLOT, PMOS, PNP, R, U, W, SP

The model card type should be the same as the Verilog-A module name. Every Verilog-A module can have one or more associated model cards.

Unlike built-in device model cards and instances, you can specify any module parameter in Verilog-A model cards, instance statements, or inherited parameter values from module definitions. Instance parameters always override model parameters. If the model card includes parameters that are not predefined in its associated module file, HSPICE issues a warning message, ignores the definition, and continues with the simulation.

Syntax

.model *mname type pname1= pname2= pname3=* …

Argument	Description
mname	User-defined model name reference. Elements must use this name to refer to this model card.
type	Model type, it must be the same as Verilog-A module name.

Argument	Description
pname#	Parameter name. Every parameter must be predefined in its associated Verilog-A module with default parameter value set. For legibility, use either blanks or commas to separate each assignment. Use a plus sign (+) to start a continuation line.

Example

For the following examples, assume the following Verilog-A module is used:

```
module va_amp(in, out);
electrical in, out;
input in;
output out;
parameter real gain=1.0;
parameter real fc=100e6;
...
analog begin
```

Its associated model cards can then be:

```
.model myamp va_amp gain=2 fc=200e6
.model myamp2 va_amp gain=10
```

The instantiations of Verilog-A module va_amp are:

```
x1 n1 n2 myamp
x2 n3 n4 myamp gain=3.0
x3 n5 n6 myamp gain=2.0 fc=150e6
x4 n7 n8 myamp2 fc=300e6
x5 n9 n10 va_amp
```

- Instance x1 inherits model myamp parameters (that is, `gain=2`, `fc=200e6`).
- Instance x2 inherits `fc=200e6` from model `myamp`, but overrides `gain` with the value 3.0.
- Instance x3 overrides all model myamp parameters.
- Instance x4 inherits parameter `gain=10` from model `myamp2`, and overrides parameter `fc`, which is an implicit parameter in `myamp2`.
- Instance x5 does not use a model card and directly instantiates the Verilog-A module `va_amp` and inherits all module `va_amp` default parameters, which are `gain=1` and `fc=100e6`.

For any X element, the default search order to find the cell definition is:

1. HSPICE subcircuit definition
2. Verilog-A model card
3. Verilog-A module definition

Example

Suppose you have the following HSPICE netlist fragment:

```
.hdl "mydiode"
X1 a b mydiode
.model mydiode D ...
```

In this example, the simulation fails even though the Verilog-A module `mydiode` is loaded. The reason is that the simulator finds the model card mydiode first, which is an HSPICE built-in 'D' model—not the Verilog-A model the X1 statement is trying to locate.

Restrictions on Verilog-A Module Names

Verilog-A module name cannot conflict with certain HSPICE built-in device keywords. If a conflict occurs, HSPICE issues a warning message and the Verilog-A module definition is ignored.

The following built-in device keywords cannot be used as Verilog-A module names: AMP, C, CORE, D, L, NJF, NMOS, NPN, OPT, PJF, PLOT, PMOS, PNP, R, U, W, SP

Overriding Subcircuits with Verilog-A Modules

If both a subcircuit and a Verilog-A module have the same case-insensitive name, by default, HSPICE uses the subcircuit definition. This behavior can be changed by setting `vamodel` options, either at the command line or in a `.OPTION` statement. The `vamodel` options are not supported in HSPICE RF.

The `VAMODEL` option works on cell-based definitions only. Instance-based overriding is not supported.

Netlist Option

Syntax

```
.OPTION vamodel[=name]
```

This option is not supported in HSPICE RF. The `name` is the cell name that uses a Verilog-A definition rather than the subcircuit when both exist. Each `vamodel` option can take no more than one name. Multiple names need multiple `vamodel` options.

If no name is provided for the `vamodel` option, HSPICE uses the Verilog-A definition whenever it is available.

Example 1

```
.option vamodel=vco
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco`.

Example 2

```
.option vamodel=vco vamodel=chargepump
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco` and cell `chargepump`.

Example 3

```
.option vamodel
```

This example instructs HSPICE to always use the Verilog-A definition whenever it is available.

Command-line Option

Syntax

```
-vamodel <name> -vamodel <name2> …
```

This command-line option is not supported in HSPICE RF. The `name` is the cell name that uses a Verilog-A definition rather than subcircuit when both are exist. Each command-line `-vamodel` option can take no more than one name. Repeat `-vamodel` if multiple Verilog-A modules are defined.

If no `name` after `-vamodel` is supplied, then in any case the Verilog-A definition, whenever it is available, overrides the subcircuit.

The following examples show various ways to set the option and the resulting HSPICE behavior.

Example 1

```
hspice pll.sp -vamodel vco
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell vco.

Example 2

```
hspice pll.sp -vamodel vco -vamodel chargepump
```

This example instructs HSPICE to use Verilog-A definition for all instantiations of cell `vco` and cell `chargepump`.

Example 3

```
hspice pll.sp -vamodel
```

This example instructs HSPICE to always use a Verilog-A definition whenever it is available.

Disabling .OPTION vamodel with .OPTION spmodel (HSPICE Only)

Use the `.OPTION spmodel` netlist option to switch back to the HSPICE definition. For example, if you override the HSPICE definition with the Verilog-A definition using `.OPTION vamodel`, use `.OPTION spmodel` during `.ALTER` analysis to revert to the HSPICE definition, which is the same as the `VAMODEL` option. The `SPMODEL` option works on cell-based definitions only. Instance-based overriding is not supported.

Syntax

```
.OPTION spmodel[=name]
```

The name is the cell name that will use spice definition. Each `spmodel` option can take no more than one name; multiple names need multiple `spmodel` options.

Example 1

```
.option spmodel
```

This example disables the previous `.OPTION vamodel`, but has no effect on the other `vamodel` options if they are specified for the individual cells. For example, if `.option vamodel=vco` is set, the cell of vco uses the Verilog-A definition whenever it is available.

Example 2

```
.option spmodel=chargepump
```

This example disables the previous `.option vamodel=chargepump`, which causes all instantiations of `chargepump` to re-use the subcircuit definition.

Using Vector Buses or "Ports"

The Verilog-A language supports the concept of buses (vector ports), whereas HSPICE does not. If you instantiate a module that has a vector port, the connections to individual bus signals in the HSPICE netlist must be specified. The Verilog-A module internally expands the vector port and connects them to the signals inside the Verilog-A module.

Example

Given a Verilog-A module with a vector port defined:

```
module d2a(in, out);
   electrical [1:4] in;
   electrical out;
      analog ...
```

Its instantiation in HSPICE could be:

```
x1 in1 in2 in3 in4 o1 d2a
```

In this case, the nodes in1 through in4 are mapped to ports in[1] -> in[4], respectively. If the bus in Verilog-A module is specified as electrical [4:1], then the signals would be connected as in1 -> in4 to in[4] -> in[1], respectively.

Using Integer Parameters

HSPICE netlist parameters are all of type real. When an integer Verilog-A parameter is assigned a real value, it is rounded to the nearest integer value.

Implicit Parameter M Support

Verilog-A supports the multiplicity factor. A Verilog-A device can have parameter that is not device specific:

```
   M         Multiplicity factor
```

If a loaded Verilog-A module has parameter named either "M" or "m", then that module parameter cannot be set in the instance line. The "M" or "m" parameter in the instance line always means the "Multiplicity factor" parameter and the appropriate multiplicity factor is applied to the Verilog-A device during the simulation. The implicit device parameter scaling factor S and the temperature difference between the element and circuit, `DTEMP`, are not supported.

Module and Parameter Name Case Sensitivity

Verilog-A is case-sensitive, whereas HSPICE is case-insensitive. This places certain restrictions on use in terms of module and parameter names and output control.

Module Names When an attempt to load a second module into the system with a module name that differs from a previously loaded module by case only, then the second module is ignored and a warning message is issued.

Module Parameters Parameters in the same module with names that only differ by case cannot be redefined in either Verilog-A instance line or Verilog-A `.MODEL` cards. HSPICE issues an error message and exits the simulation.

Example

In this example a simple amplifier accepts two parameters, gain and Gain, as input to the module.

```
module my_amp(in, out);
   electrical in, out;
   parameter real gain = 1.0;
   parameter real Gain = 1.0;
   analog V(out) <+ (Gain+gain)*V(in);
endmodule
```

If you instantiate this module as:

```
x1 n1 n2 my_amp Gain=1
```

HSPICE cannot uniquely define the Gain parameter, so a warning message is issued and the definition of Gain is ignored. This module can be instantiated as is, provided neither the Gain nor gain parameter is assigned in the netlist.

Instantiating Primitive Devices inside Verilog-A Modules

Beginning with the A-2007.09 release, HSPICE Verilog-A supports instantiation of HSPICE many primitive devices inside Verilog-A modules. You can instantiate primitive devices with the same feature set as Verilog-A modules. For example, the following is part of an HSPICE netlist, which defines one `pmos` model, one `nmos` model, and calls a Verilog-A module, `'inva'`.

```
*
.param ww=50u
.hdl 'inv.va'
.model n nmos level=49 version = 3.0
.model p pmos level=49 version = 3.0
 vvdd vdd 0 3.3
vgnd gnd 0 0
vin in 0 pulse 0 3.3 0 1p 1p  100n 200n
x1 in out vdd gnd inva parm_w=ww
...
```

The following is the Verilog-A module file, *inv.va*, which instantiates an HSPICE primitive `nmos` device and a `pmos` device, whose model definitions are in the `.model` statements of the HSPICE netlist above. Note that the Verilog-A function `$mfactor` is used to define the M (multiplicity) factor in the hierarchical instance of the HSPICE primitive MOSFET device.

```
//
`include "discipline.h"
module inva(in, out,vdd, gnd);
input in,vdd, gnd;
output out;
electrical in, vdd, gnd, out;
parameter  parm_w=5.0u, parm_l=0.35u;
n # (.w(parm_w), .l(parm_l), .$mfactor(30), .dtemp(100))
m1(out,in, gnd, gnd);
p # (.w(10u),.l(parm_l), .$mfactor(30), .dtemp(100))  p1(out,in,
vdd, vdd);

endmodule
```

Table 81 Supported Native Devices

Devices	HSPICE Model Levels and Comments
MOSFET	1,2,3,4,5,49,50,53,54,55,57,58,59,61,62,63,64,68,72,73
BJT	1
JFET	1
Diode	1, 2,3
Resistor Capacitor	The resistor and capacitor can be instantiated via a modelcard or directly. For example: `resistor #(.r(r)) my_dev2(p,n);`

Table 81 Supported Native Devices

Devices	HSPICE Model Levels and Comments
Inductor	The inductor must be instantiated directly. For example: `inductor #(.l(l)) my_dev1(p,n);`

Instantiating HSPICE subcircuits inside Verilog-A Modules

Beginning with the A-2008.09 release, HSPICE Verilog-A supports instantiation of HSPICE subcircuits from within Verilog-A modules. Subcircuits are instantiated using the format of a Verilog-A hierarchical instantiation, using the subcircuit and parameter names defined in the HSPICE netlist.

For example, the following is part of an HSPICE netlist, which defines a subcircuit named `my_rcnetwork` with one node and two parameters. The netlist instantiates a Verilog-A module called `va_amp`. This module will in turn instantiate the `my_rcnetwork` subcircuit.

```
.hdl 'modules.va'
.param local_r=50 local_c=1p
.subckt my_rcnetwork(n1) r=local_r p=local_c
R1 n1 0 r
C1 n1 0 c
.ends
vin in 0 pulse 0 3.3 0 1p 1p 100n 200n
x1 in out va_amp gain=2 r_filter=25 c_filter=2p
...
```

The following is the Verilog-A module file, `modules.va`, which defines the module `va_amp`.

```
`include "discipline.h"
module va_amp(in, out);
input in;
output out;
electrical in, out;
parameter real gain=1.0;
parameter real r_filter=50;
parameter real c_filter=1p;
// Instantiate the HSPICE subcircuit:
my_rcnetwork #(.r(r_filter), .c(c_filter))
 my_filter1(in);
my_rcnetwork #(.r(r_filter), .c(c_filter)) my_filter2(in);
analog
   V(out) <+ gain * V(in);
endmodule
```

The subcircuit `my_rcnetwork` is instantiated twice, at node `in` and node `out`. The input parameters to the Verilog-A module, `r_filter` and `c_filter`, are passed to the subcircuit `my_rcnetwork` parameters `r` and `c` using the Verilog-A parameter passing syntax.

Output Simulation Data

Verilog-A devices support the same output capabilities as built-in devices. You can access the following Verilog-A device quantities via any of these HSPICE output statements: `.PRINT`,`.PROBE`, `.DOUT`, and so forth.

- Port current
- Port voltage
- Internal node voltage (HSPICE only)
- Internal named branch current (HSPICE only)
- Internal module variables (HSPICE only)
- Module parameters (HSPICE only)

V() and I() Access Functions

You can access port voltage and internal node voltage of Verilog-A devices via the V() function. Port current and internal branch currents can be accessed via the I() function.

The internal nodes of Verilog-A devices are accessible via the V() function when the full hierarchical name is provided. The port current and named branches (on the instance base only) can be accessible via the I() function.

Examples:

In the following examples, assume the Verilog-A module definition fragment is:

```
module va_fnc(plus, minus);
inout plus, minus;
electrical plus, minus;
electrical int1, int2;
branch (int1, int2) br1;
  //creates an internal branch br1 between internal
  //nodes int1 and int2;

analog begin
```

And the Verilog-A module may be instantiated in the netlist as:

```
x1 1 2 va_fnc
```

To print the current on Verilog-A device port name plus for the instance x1:

```
.print I(x1.plus)
```

The `plus` is the port name defined in Verilog-A module, not the netlist node name.

To print the Verilog-A module internal node named int1 for the instance x1:

```
.print V(x1.int1)
```

If the `va_fnc` module is hierarchical and has a child instance called c1 with an internal node int1 then the node int1 can be output as

```
.print V(x1.c1.int1)
```

That is, the full HSPICE instance name is concatenated with the full internal Verilog-A instance name to form the complete name.

During compilation of Verilog-A modules, the compiler optimizes some internal branches out of the system such that these branches are not available for output. HSPICE Verilog-A provides a compilation environment variable, `HSP_VACOMP_OPTIONS`, with `-B` option being set, all internal named branches in Verilog-A modules become accessible. However, making all internal branches accessible may have negative impact on simulation performance; turn on the option only when necessary.

Refer to section Setting Options for the HSPICE Verilog-A Compiler for examples of setting `HSP_VACOMP_OPTIONS`.

After `HSP_VACOMP_OPTIONS -B` is set, you can probe branch current with HSPICE output commands. In the previous Verilog-A module, there is an internal branch name br1 declared. To probe the branch current

```
.print I(x1.br1)
```

Output Bus Signals

Verilog-A bus signals can be accessed with HSPICE output commands using the Verilog-A naming and accessing conventions.

Example

Given an example Verilog-A module:

```
module my_bus(in, out);
electrical in;
electrical [1:4] out;
...
```

And instantiated in the netlist as

```
x1 1 2 3 4 5 my_bus
```

...then values at vector port out can be output by explicitly listing each position.

```
.print v(x1.out[1]), v(x1.out[2]), v(x1.out[3]), v(x1.out[4])
```

Bus elements can also be specified using wildcards, as described in the section Using Wildcards in Verilog-A (HSPICE only) on page 792.

Output Internal Module Variables (HSPICE only)

Verilog-A internal variables, by default, are hidden from output. However, module variables with a description or units attribute, or both, are known as output variables, and HSPICE provides access to their values; for example, suppose a module for a MOS transistor with the following declaration at module scope provides the output variable `cgs`:

```
(* desc="gate-source capacitance", units="F" *) real cgs;
```

The cgs module variable can be printed just like a normal parameter variable. In addition, HSPICE Verilog-A provides a compilation environment variable `HSP_VACOMP_OPTIONS`, with `-G` option being set, you can use HSPICE output command to access internal module variables of Verilog-A instances.

Alternatively, the compiler options can be set with `.option VAOPTS` in your netlist. For example:

```
.option vaopts=str('-G')
```

Syntax

`Instance:`*`internal_variable`*

Example

```
.print xva_vco:freq
```

This example outputs internal variable frequency value of Verilog-A instance xva_vco.

Output Module Parameters (HSPICE only)

You can apply HSPICE element templates to HSPICE primitives that are instantiated in Verilog-A modules. The element templates are found in Chapter 11, section: Element Template Listings (HSPICE Only).

Example

For a netlist instantiation of a Verilog-A device:

```
. model mod1 npn beta=222
X1 c b 0 0 va_wrapper area=0.25 m=2
...
.PRINT TRAN X1.Q1:BETA LV2(X1.Q1)
```

where the Verilog-A source code definition of a module name `va_wrapper`:

```
`include "disciplines.vams"
module va_wrapper(c,b,e,s);
electrical c,b,e,s;
parameter real area=1.0;
mod1 #(.area(area)) q1(c,b,e,s);
endmodule
```

This example outputs the `BETA` value and the `icvbe` (template element alias `lv2`) for the Verilog-A instance `X1` that in turn instantiated a primitive BJT of the model mod1 in an instance `Q1`.

Case Sensitivity in Simulation Data Output

When Verilog-A information is output via the HSPICE output commands, the case of the node names associated with the quantities to be output is ignored. Contributions from the Verilog-A noise sources that have the same name when case is ignored are combined.

Example

```
I(d,s) <+ white_noise(4*k*T/R1, "thermalnoise");
I(d2,s2) <+ white_noise(4*k*T/R2, "ThermalNoise");
```

The two noise contributions are combined into one contribution called `thermalnoise` in the output files.

Using Wildcards in Verilog-A (HSPICE only)

Verilog-A names support the use of wildcards to simplify using the output commands.

Examples:

Given the Verilog-A module,

```
module test(p,n);
electrical p,n;
electrical int1, int2;
…
```

instantiated as

```
x1 1 2 test
```

then all of the internal nodes (in this case `int1` and `int2`) can be printed using the command:

```
.print v(x1.*)
```

All indices of a bus in the module:

```
module my_bus(in, out);
electrical in;
electrical [1:4] out;
…
```

Can be specified as:

```
x1 1 2 3 4 5 my_bus
.print v(x1.out[*])
.print v(x1.*)
```

Both of the internal nodes, `int1` and `int2` for the child `ch1` in the instance `x_par1` can be specified using

```
.print v(x_par1.ch1.int*)
```

The HSPICE `.OPTION POST` command does not output internal nodes from Verilog-A modules. Use the wildcard feature to specify a Verilog-A instance if you need to output all internal nodes.

Port Probing and Branch Current Reporting Conventions

When printing and reporting currents for Verilog-A devices, HSPICE follows the same conventions when specifying the direction of current flow as in built-in devices. A positive branch current implies that current is flowing into the device terminal or internal branch.

Unsupported Output Function Features

The following output functions are not supported in this release:

- Port probing: `In( )`, where `n` is the node number). Instead, you can use `I(instance.port_name_in_module).`
- `Iall()`: Instead, you can output all the terminal currents using a wild card.
- `Isub()`: This is not applicable to Verilog-A components.
- `P()` and `Power()`: Instead, you can use the `$strobe` Verilog-A function.
- Nodal capacitance
- Group delay

- Direct current probing on MOSFET elements that are instantiated inside Verilog-A modules.
- Element template output on MOSFET elements that are instantiated inside Verilog-A modules.

Running 32-bit HSPICE Verilog-A on Linux x86_64

HSPICE Verilog-A CML files are shared libraries created by the Verilog-A module compiler, hsp-vacomp. On a 64-bit system, a 32-bit executable must load 32-bit shared libraries. When the 32-bit HSPICE is run on a 64-bit system, hsp-vacomp must create 32-bit CML files.The system needs to support building both 32 and 64 bit exes/libs (i.e., it must have multilib support).

If your 32-bit HSPICE has problems compiling Verilog-A modules on a 64-bit system, your system may not have multilib support. You may see the following error when running 32-bit HSPICE Verilog-A:

```
/usr/bin/ld: cannot open crti.o: No such file or directory
  collect2: ld returned 1 exit status
```

On a 64-bit system that is set up to build both 32- and 64-bit libraries you will find two sets of library/object files in these directories,

```
/usr/lib
/usr/lib64
```

On a system that does not have 32-bit build support you will find only

```
/usr/lib64
```

To fix the problem, first find which package installs `/usr/lib64/crti.o`

```
prompt>rpm -qf /usr/lib64/crti.o
+ --qf "%{name}-%{version}-%{release}.%{arch}\n"
glibc-devel-2.3.4-2.9.x86_64
```

It is a reasonable guess, then, that the rpm:

```
glibc-devel-2.3.4-2.9.i386
```

should be installed to get 32-bit build support. For example, on a 64-bit system that is set up for multilib support, what is returned is:

```
$ rpm -q glibc-devel --qf "%{name}-%{version}-
%{release}.%{arch}\n"
  glibc-devel-2.3.4-2.9.i386
  glibc-devel-2.3.4-2.9.x86_64
```

Use any package manager to install `glibc-devel-2.3.4-2.9.i386`.
For example, rpm -i glibc-devel.386.

You will then see `/usr/lib/crti.o` and other files on your system and you can build 32-bit CMLs on the 64-bit architecture.

To use the standalone utility hsp-vacomp to compile Verilog-A for a 64-bit workstation, you need to set the environment variable as follows:

```
setenv bitflag 64
```

When the hsp-vacomp utility is run, a 64-bit CML file is generated.

Compiling Verilog-A Files on Demand

Under default conditions the compiler will compile any files found in `.hdl` commands only when modules in the file are instantiated in the circuit. Note that if a file contains multiple modules and only one module is instantiated, all of the modules in the file will be compiled. While this is typically not an issue, compilation times can be noticeable for extremely large modules.

To always compile files even if no modules in the file are used in the circuit, set the environment variable, HSP_DISABLE_DEMAND_COMPILE:

```
setenv HSP_DISABLE_DEMAND_COMPILE 1
```

Setting Options for the HSPICE Verilog-A Compiler

While Verilog-A modules are automatically compiled in HSPICE simulation, you can set the environment variableto control compiler options from the default setting.

```
 HSP_VACOMP_OPTIONS
```

Alternatively, the compiler options can be set with `.option VAOPTS` in your netlist.

Note:

Multiple settings of the compiler options will be concatenated.

Example 1

```
setenv HSP_VACOMP_OPTIONS -G
```

When `-G` is set, all internal variables are accessible for output. For example, `-G` allows HSPICE to access Verilog-A options in `.MEASURE` statements, or you can print the values of non-electrical Verilog-A variables to a waveform file if you add a `.PROBE` statement to the top level SPICE netlist.

Example 2

```
setenv HSP_VACOMP_OPTIONS -B
```

When `-B` is set, all internal named branches are accessible for output.

Example 3

```
setenv HSP_VACOMP_OPTIONS "-B -G"
```

All internal variables and named branches are accessible for output.

The Compiled Model Library Cache

The HSPICE Verilog-A solution provides the performance of a compiled solution without the need for user intervention. The first time a Verilog-A source file is loaded, or after a Verilog-A source file is modified, the system automatically invokes the compiler. The Compiled Model Library (CML) is automatically cached and subsequent simulations use this cached file and bypass the compilation process. Although for the most part you do not need to be concerned with the cache mechanism, you can control some aspects. You can change the cache location, prevent caching, or delete the cache.

Cache Location

By default the cache directory is located in your $HOME directory under the hidden directory .hsp-model-cache

This directory holds a directory structure that indicates compiler version, platform, and model directory.

Example

Given the load command

```
.hdl "/users/finn/modules/amp.va"
```

the compiler generates a CML file in

```
/users/finn/.hsp-model-
cache/1.30/users/finn/modules/lib.<arch>/amp.cml
```

Where `<arch>` is one of hpux, sun, linux, and so on.

The location of the cache can be changed from the default value by setting the environment variable `HSP_CML_CACHE` to an appropriate location.

Example

The following sets the environment variable `HSP_CML_CACHE` so that the model cache is created under the my_local_cache directory.

```
setenv HSP_CML_CACHE /users/finn/my_local_cache
```

If the previous example were now simulated, the CML file would be

```
/users/finn/my_local_cache/1.30/users/finn/modules/lib.<arch>
/amp.cml
```

Deleting the Cache

The cache structure is maintained unless you choose to delete it manually. You can do this any time; HSPICE automatically recreates the cache when needed. One reason to delete the cache is if a newer version of the HSPICE Verilog-A compiler is used and the previous cache is no longer necessary. The cache can be deleted using conventional operating system commands.

Example

To delete the default cache from the operating system command prompt, type:

```
rm -r ~/.hsp-model-cache
```

Using the Standalone Compiler

Verilog-A modules used in HSPICE simulations are automatically compiled and cached by the simulator. You can compile files manually if you wish (to check syntax for example). The Verilog-A compiler takes a Verilog-A file as an input and produces a Compiled Model Library (CML) file, which is a platform-specific shared library.

Example 1

```
hsp-vacomp resistor.va
```

The Verilog-A compiler, `hsp-vacomp`, compiles the Verilog-A module file resistor.va, and produces a CML file resistor.cml in the same directory.

You can include the CML file in the same manner as the Verilog-A file in an HSPICE netlist.

Example 2

```
.hdl "resistor.cml"
```

Note:

When a CML file is specified in the load command the compiler is never invoked, even if the source file is modified.

To launch this standalone compiler, use the 'hsp-vacomp' wrapper located in the $installdir/bin directory, rather than the platform-specific executable, e.g., $installdir/linux/hsp-vacomp. The wrapper takes care of all the required environment settings, and picks up the executable from the appropriate platform directory.

Troubleshooting Environment Setting Issues

If you cannot run netlists containing Verilog-A modules in the latest version of HSPICE, verify that your license key has the license token `hspiceva` in your license file. This token is part of all HSPICE packages and is available on all supported platforms. The following errors indicate that the correct environment variables have not been set.

Message on the terminal:

```
The environment variable 'HSP_HOME' must be set to use hsp-vacomp
```

Error message in the output *.lis* file:

```
hsp-vacomp: Error: File not found 'disciplines.vams' or
'discpline.h' or 'constants.h' in search path '.'=
```

To correct the errors, source the *cshrc.meta* file found in the HSPICE bin directory. This sets the environment variables 'HSP_HOME' and 'SHLIB_PATH'.

For the Windows platform, see SolvNet article:
https://solvnet.synopsys.com/retrieve/020458.html

Unsupported Language Features

The following Verilog-A LRM 2.2 Language Features are not supported.

- Derived natures described in LRM 2.2 section 3.4.1.1

  For example, the following deriving the nature `New_curr` from `Ttl_curr` is not supported.

```
nature Ttl_curr
   units = "A" ;
   access = I ;
abstol = 1u ;
endnature
// The derived nature is not supported:
nature New_curr : Ttl_curr
   abstol = 1m ;
   maxval = 12.3 ;
endnature
```

- Input, output, and inout enforcement described in LRM 2.2, section 7.1.

```
module test(in,out);
   electrical in,out;
   input in;
   output out;
   real out_value;
   analog begin
      out_value = 1.0;
      V(in) <+ out_value;   // Input node used as output
         // is not prevented, V(in) will be
         // assigned to out_value
   end
endmodule
```

- The `defparam` statement as described in LRM 2.2, section 7.2.1

  For example:

```
module rc(n1, n2);
   electrical n1, n2;
   my_res r1 (n1, n2);
   my_cap c1 (n1, n2);
endmodule
module  my_res(n1, n2);
   electrical n1, n2;
   parameter dev_temp = 27;
   parameter res = 50;
   parameter tcr = 1;
   analog
      V(n1,n2) <+ I(n1,n2)*res*tcr*($temperature-dev_temp);
endmodule


module  my_cap(n1, n2);
   electrical n1, n2;
   parameter dev_temp = 25;
   parameter cap = 1;
   parameter tcc = 1;
   analog
      I(n1,n2) <+ cap*ddt(V(n1,n2))*tcc*($temperature
      dev_temp);
endmodule
// defparam statement not supported
module annotate;
defparam
   rc.r1.dev_temp = 30;
   rc.c1.dev_temp = 25;
endmodule
```

- Ordered parameter lists in hierarchical instantiation as described in LRM 2.2, section 7.2.2.

  For example:

```
module module_a(out,out2);
   electrical out,out2;
   parameter real value1 = -10.0;
   parameter real value2 = -20.0;
   analog begin
      V(out) <+ value1;
      V(out2) <+ value2;
   end
endmodule
module test_param_by_order(out,out2);
   electrical out,out2;
   parameter real value1 = -1.0;
   parameter real value2 = -2.0;
   // Ordered parameter lists are not supported:
   module_a #(1,2) A1(out,out2);
   // instead use:
   module_a #(.value1(1),.value2(2)) A1(out,out2);
endmodule
```

- Out-of-module-references as described in the LRM 2.2, section 7.

  In this example, the reference to `example2.net` inside the `example1` module is not supported.

```
module example1;
   electrical example2.net; // Feature not supported
endmodule
module example2;
   electrical net;
endmodule
```

- Vector ports, where the port expression defining the size of a port is a parameter expression, as described in LRM 2.2 section 7.4.1.

  In this example, the vector port range size must be a constant—not a parameter value.

```
module test(out);
   parameter integer size = 7 from [1:16];
   electrical [0:size] out; // Feature not supported
   analog begin
      V(out[0]) <+ 0.0;
   end
endmodule
```

- The `$monitor` function, as described in LRM 2.2, section 10.6.

```
$monitor("\nEvent occurred."); // Feature not supported
```

- Parameters used to specify ranges for the `generate` statement, as described in LRM 2.2, section C.19.3.

  ```
  generate indexr_identifier (start_expr,end_expr[,incr_expr ])
  ```

- Time tolerances on transition() functions, as described in LRM 2.2, section 4.4.9.1, respectively.

  ```
  transition(expr[,td [,rise_time [,fall_time [,time_tol ] ] ] ])
  ```

- “random type-string” feature (LRM 2.2, Sections 10.2 and 10.3).
- reg-strings as described in Section 2.6 of LRM 2.2.
- Output variables and string parameters on paramsets.
- The following are limitations in HSPICE RF Verilog-A only:
  - `$strobe` in DC analysis
  - `$simparam` simulation parameter
  - @(final_step)
  - 0 port module
  - Delays (`absdelay()`), event-controlled constructs, memory states (variables that hold their value between timesteps), and explicit time-dependent functions are not supported in RF analyses.

Known Limitations

Known limitations for Verilog-A with HSPICE and HSPICE RF include:

analysis() Function Behavior

The `analysis()` function definition assumes that the operating point (OP) analysis associated with any user-specified analysis is unique to that user-specified analysis. For example, when you specify the following function, it must return 1 for AC analysis and 1 for its underlying operation point (OP) analysis.

```
analysis("ac")
```

Similarly, `analysis("tran")` must return 1 for transient analysis and 1 for its underlying OP analysis. In HSPICE, a single “common” OP analysis is performed in the setup that is outside the context of AC, transient, or other analyses. Since that OP is outside the context of the user-specified analysis, the analysis() function does not know the parent analysis type (during the OP analysis). The analysis(“ac”), analysis(“tran”), and so on, returns 0 during this “common” OP analysis. You can ensure that the analysis function returns true (1) during these analyses by adding “static” to the list of functions.

Example

```
if ( analysis("ac") )
begin
   // do something
end
```

should be written as:

```
if( analysis("ac", "static") )
begin
   // do something
end
```

The same is true for the “tran” and “noise” analysis names.

Part: 5 Demonstration Files/Errors-Warnings

The section contains the following chapters:

- Chapter 31, Running Demonstration Files
- Chapter 32, Warning/Error Messages

Chapter 30 contains descriptions and locations of the demo files:

- HSPICE Integration to Cadence^{TM} Virtuoso® Analog Design Environment Tutorial Examples
- Applications of General Interest Examples
- Behavioral Application Examples
- Benchmark Examples
- Bisection-Timing Analysis Examples
- BJT and Diode Examples
- Cell Characterization Examples
- Circuit Optimization Examples
- Device Optimization Examples
- Encryption Examples
- Fourier Analysis Examples
- Filters Examples
- IBIS / Examples
- Magnetics Examples
- MOSFET Device Examples
- Signal Integrity Examples
- Sources Examples
- S-parameter Examples
- Transmission Lines Examples
- Transmission (W-element) Line Examples
- Variability Examples
- Verilog-A Examples

31 Running Demonstration Files

Contains examples of basic file construction techniques, advanced features, and simulation hints. Lists and describes numerous HSPICE input files. For HSPICE RF-specific input files see the Tutorial chapter of the HSPICE User Guide: RF Analysis.

These topics are covered in the following sections:

- Using the Demo Directory Tree
- Two-Bit Adder Demo
- MOS I-V and C-V Plot Example Input File
- Temperature Coefficients Demo
- Modeling Wide-Channel MOS Transistors
- Listing of Demonstration Input Files

Using the Demo Directory Tree

To run demonstration files: go to your HSPICE installed version location e.g.: <*path_to_hspice_version*>/hspice/demo/hspice. In this directory you will find multiple demo files. After a proper hspice path has been set up, you can execute Linux/Solaris/HP:

```
%> hspice -i your_spice_file -o output_file
```

On Windows, execute Start > All > Programs > *your_hspice_installed_version* > *HSPICE-installed_version* > file > simulate and proceed.

The tables in the section Listing of Demonstration Input Files on page 824 list demonstration files, which are designed as training examples. All HSPICE distributions include these examples in the demo directory tree, where `$installdir` is the installation directory environment variable:

Table 82 Demo Directories

Directory Path	**File Directory**	**Description**
$*installdir*/demo/hspice	/aa_integ	HSPICE integration tutorial, Analog Artist
	/apps	general applications
	/behave	analog behavioral components
	/bench	standard benchmarks
	/bjt	bipolar components
	/bisect	bisection optimization
	/cchar	characteristics of cell prototypes
	/ciropt	circuit level optimization
	/ddl	Discrete Device Library
	/devopt	device level optimization
	/fft	Fourier analysis
	/encrypt	Traditional, 8-bit, and 3DES encryption
	/filters	filters
	/ibis	IBIS examples
	/mag	transformers, magnetic core components
	/mos	MOS components
	/si	Signal Integrity applications
	/sources	dependent and independent sources
	/sparam	S-parameter applications
$*installdir*/demo/hspice	/tline	filters and transmission lines

Table 82 Demo Directories (Continued)

Directory Path	File Directory	Description
	/twline	W-element transmission lines and field solvers
	/variability	Variation Block, Monte Carlo, and AC/DC Mismatch examples
	/veriloga	Verilog-A examples

Two-Bit Adder Demo

This two-bit adder shows how to improve efficiency, accuracy, and productivity in circuit simulation. The adder is in the $*installdir*/demo/hspice/apps/mos2bit.sp demonstration file. It consists of two-input NAND gates, defined using the NAND subcircuit. CMOS devices include length, width, and output loading parameters. Descriptive names enhance the readability of this circuit.

One-Bit Subcircuit

The ONEBIT subcircuit defines the two half adders, with carry in and carry out. To create the two-bit adder, HSPICE or HSPICE RF uses two calls to ONEBIT. Independent piecewise linear voltage sources provide the input stimuli. The *R* repeat function creates complex waveforms.

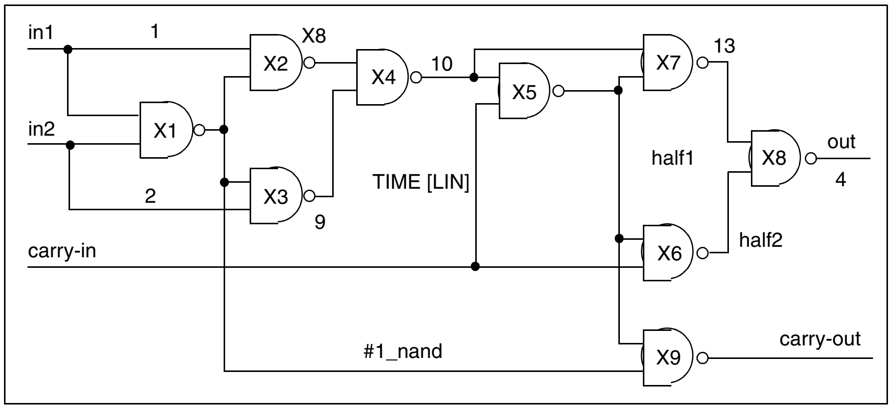


Figure 202 One-bit Adder subcircuit

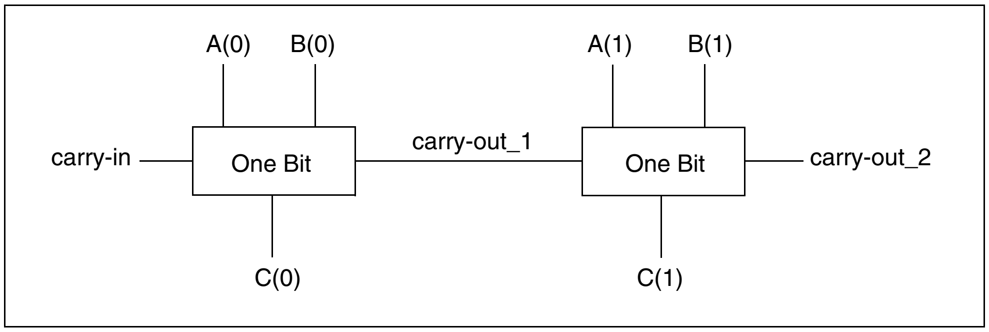


Figure 203 Two-bit Adder Circuit

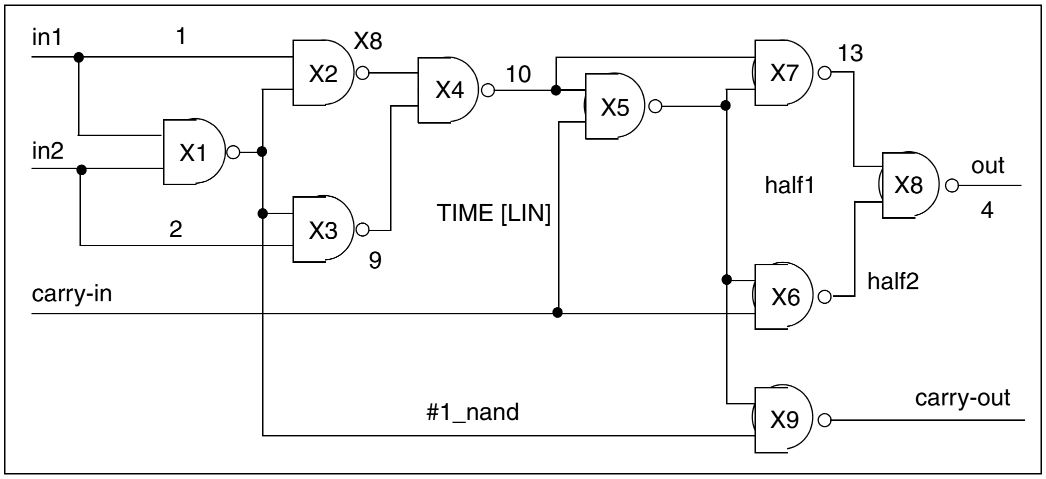


Figure 204 1-bit NAND Gate Binary Adder

MOS Two-Bit Adder Input File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/apps/mos2bit.sp`

MOS I-V and C-V Plotting Demo

To diagnose a simulation or modeling problem, you usually need to review the basic characteristics of the transistors. You can use this demonstration template file, $*installdir*/demo/hspice/mos/mosivcv.sp, 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 83 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	Eases circuit clarity. Symbolic name contains up to 16 characters.
.PRINT	.PRINT statements print internal variables.

Printing Variables

Use this template to print internal variables, such as:

Table 84 Demo Printing Variables

Variable	Description
i(mn1)	i1, i2, i3, or i4 specifies true branch currents for each transistor node.
LV18(mn6)	Total gate capacitance (C-V plot).
LX7(mn1)	GM gate transconductance. (LX8 specifies GDS; LX9 specifies GMB).

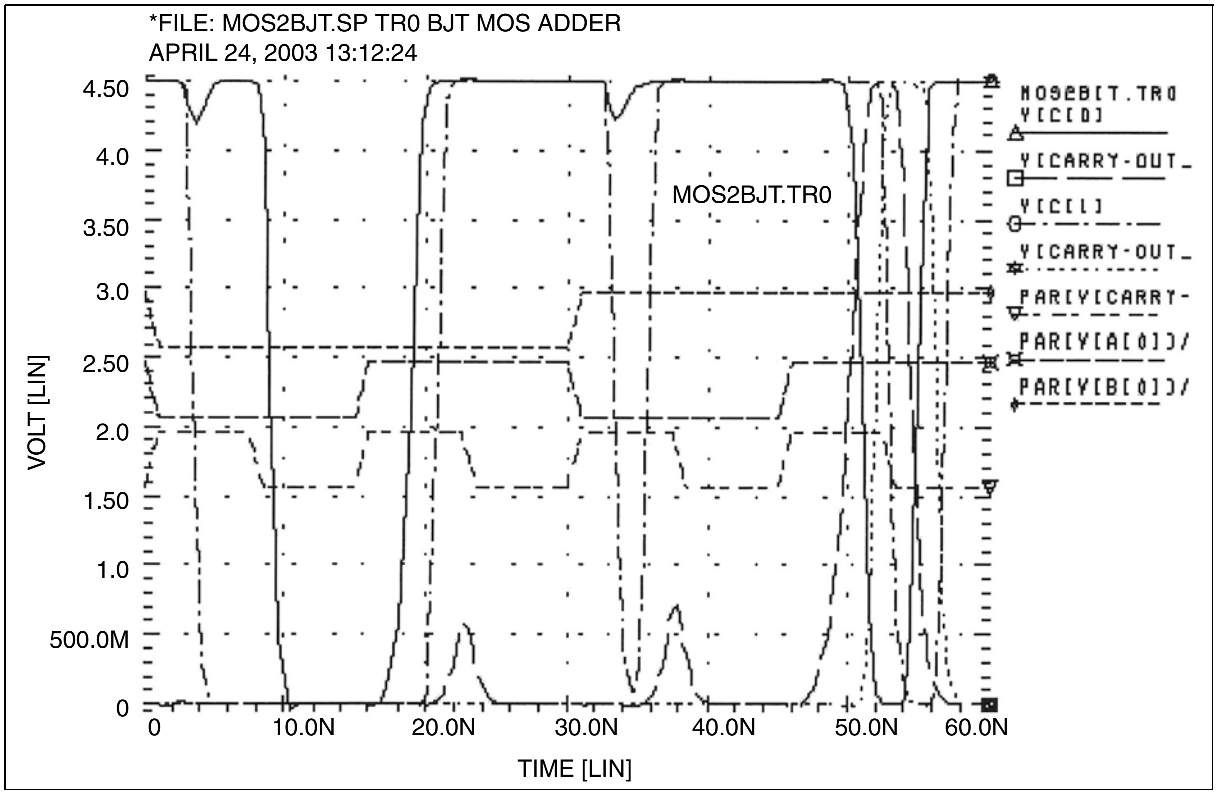


Figure 205 MOS IDS Plot

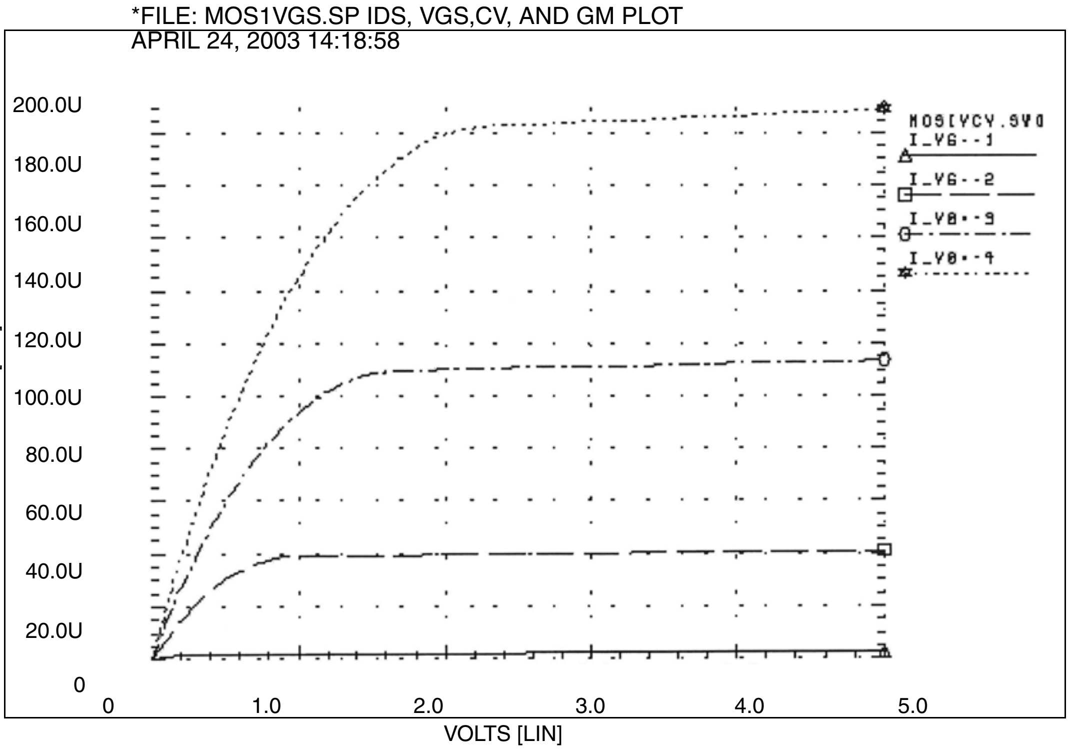


Figure 206 MOS VGS Plot

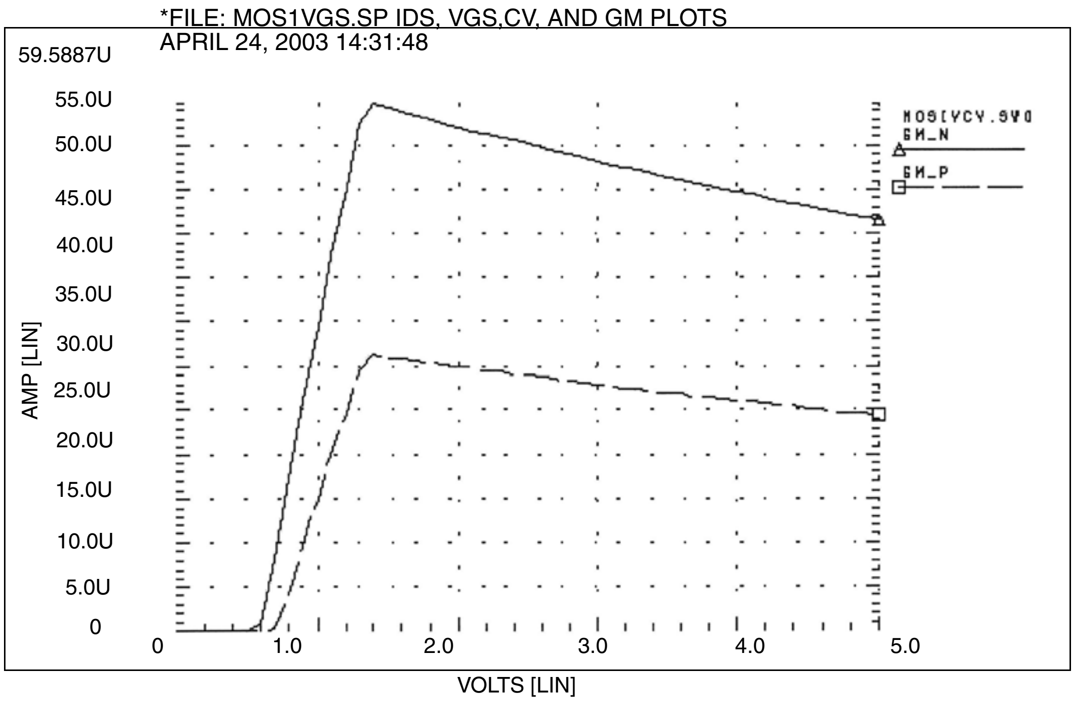


Figure 207 MOS GM Plot

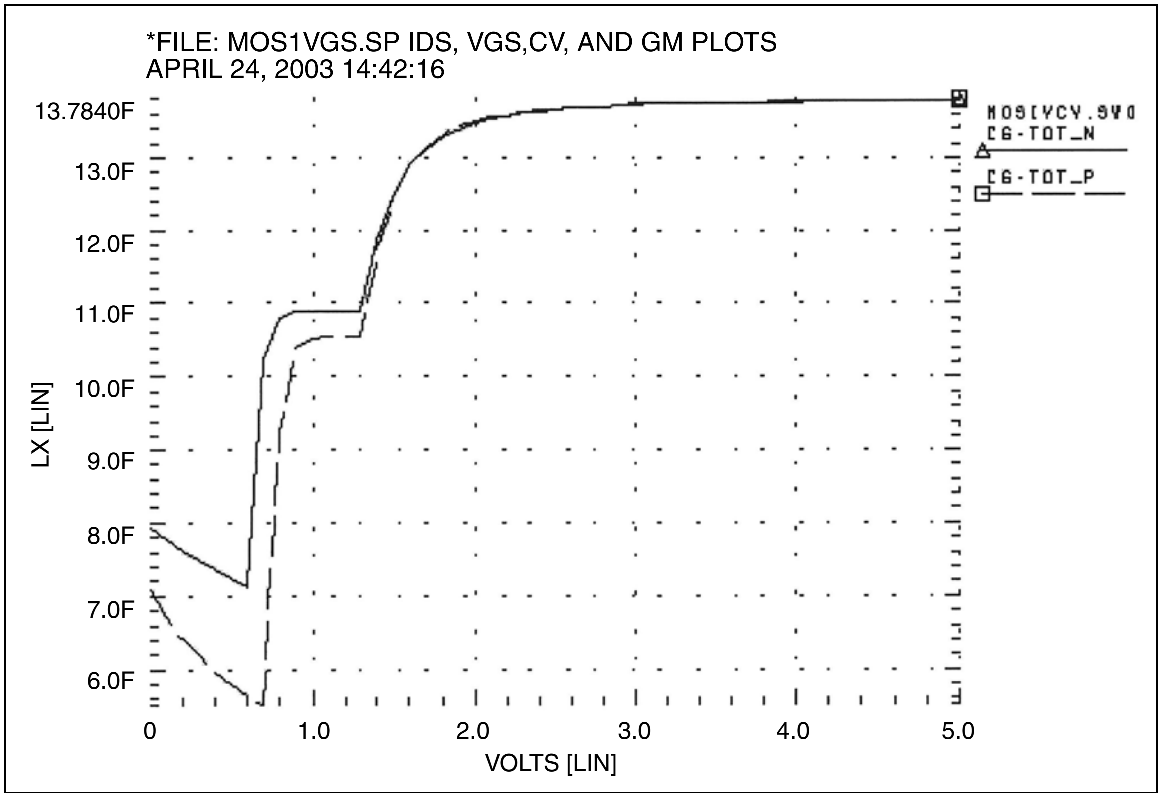


Figure 208 MOS C-V Plot

MOS I-V and C-V Plot Example Input File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/mos/mosivcv.sp`

CMOS Output Driver Demo

ASIC designers need to integrate high-performance IC parts onto a printed circuit board (PCB). The output driver circuit is critical to system performance. The $*installdir*/demo/hspice/apps/asic1.sp demonstration file shows models for an output driver, the bond wire and leadframe, and a six-inch length of copper transmission line.

This simulation demonstrates how to:

- Define parameters, and measure test outputs.
- Use the LUMP5 macro to input geometric units, and convert them to electrical units.
- Use `.MEASURE` statements to calculate the peak local supply current, voltage drop, and power.
- Measure RMS power, delay, rise times, and fall times.
- Simulate and measure an output driver under load. The load consists of:
  - Bondwire and leadframe inductance.
  - Bondwire and leadframe resistance.
  - Leadframe capacitance.
  - Six inches of 6-mil copper, on an FR-4 printed circuit board.
  - Capacitive load, at the end of the copper wire.

Strategy

The HSPICE strategy is to:

- Create a five-lump transmission line model for the copper wire.
- Create single lumped models for leadframe loads.

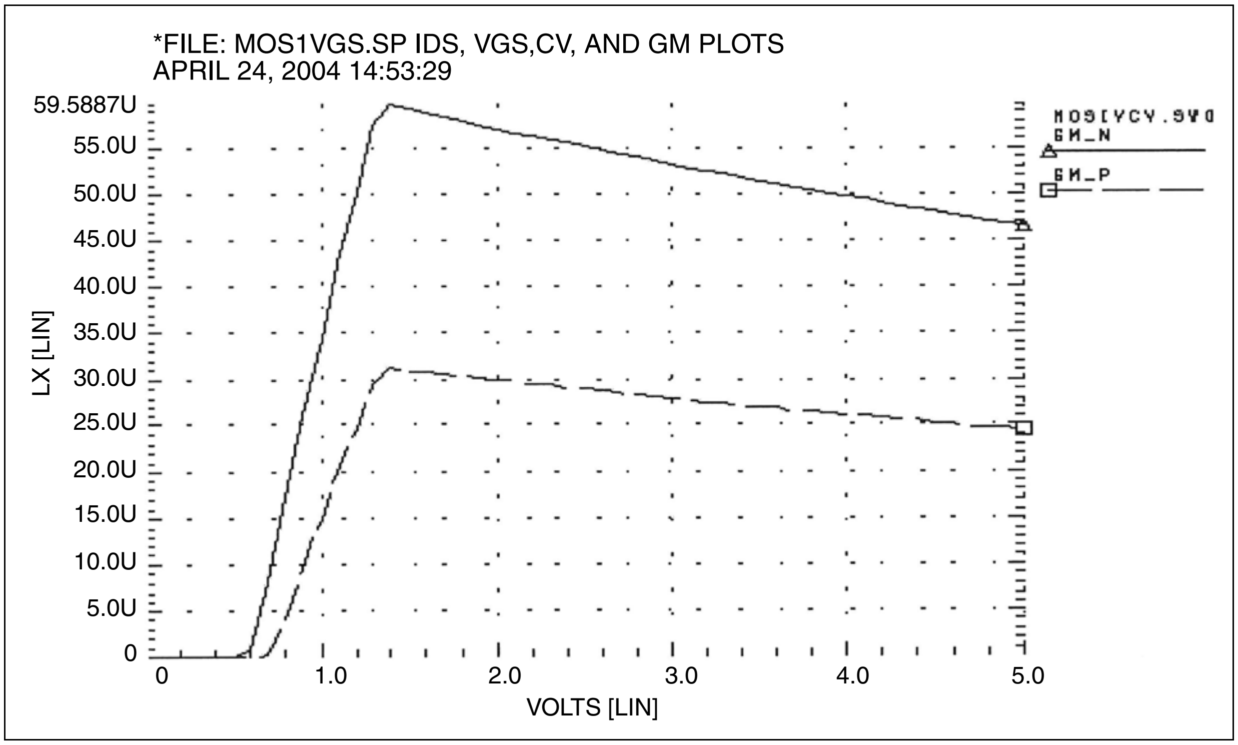


Figure 209 Noise Bounce

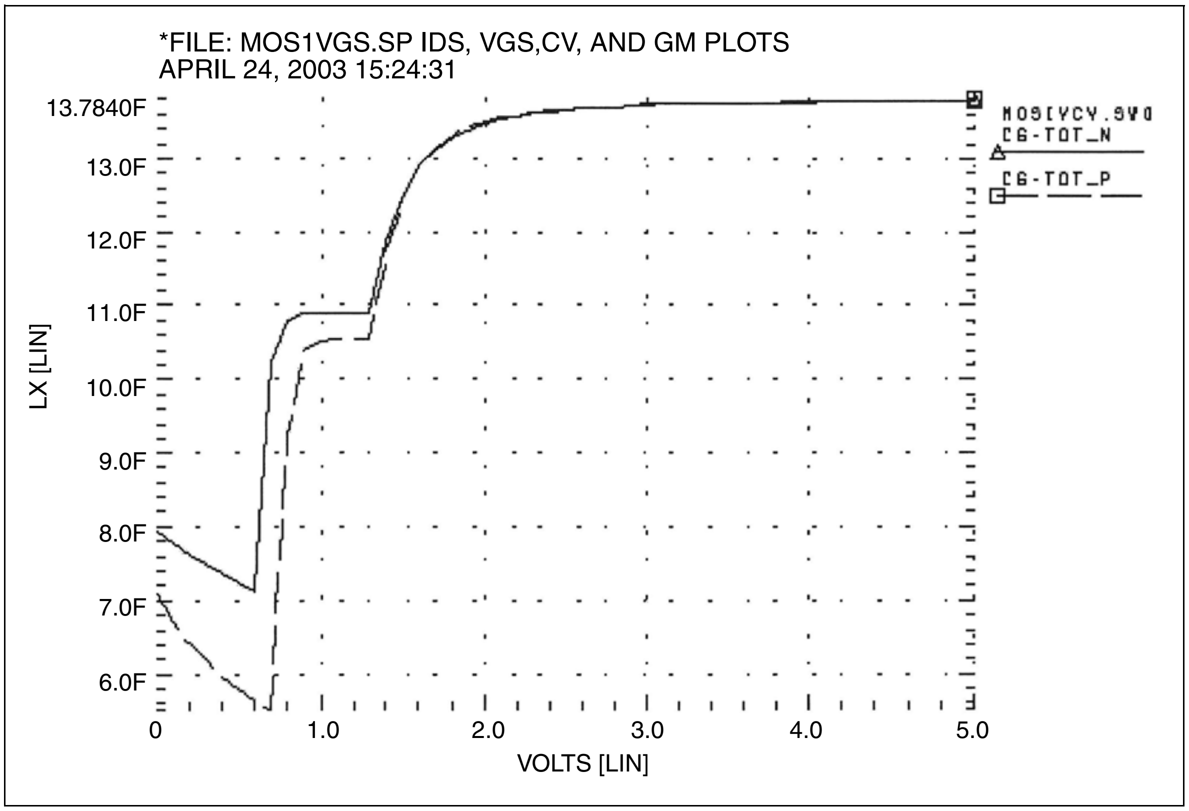


Figure 210 Asic1.sp Demo Local Supply Voltage

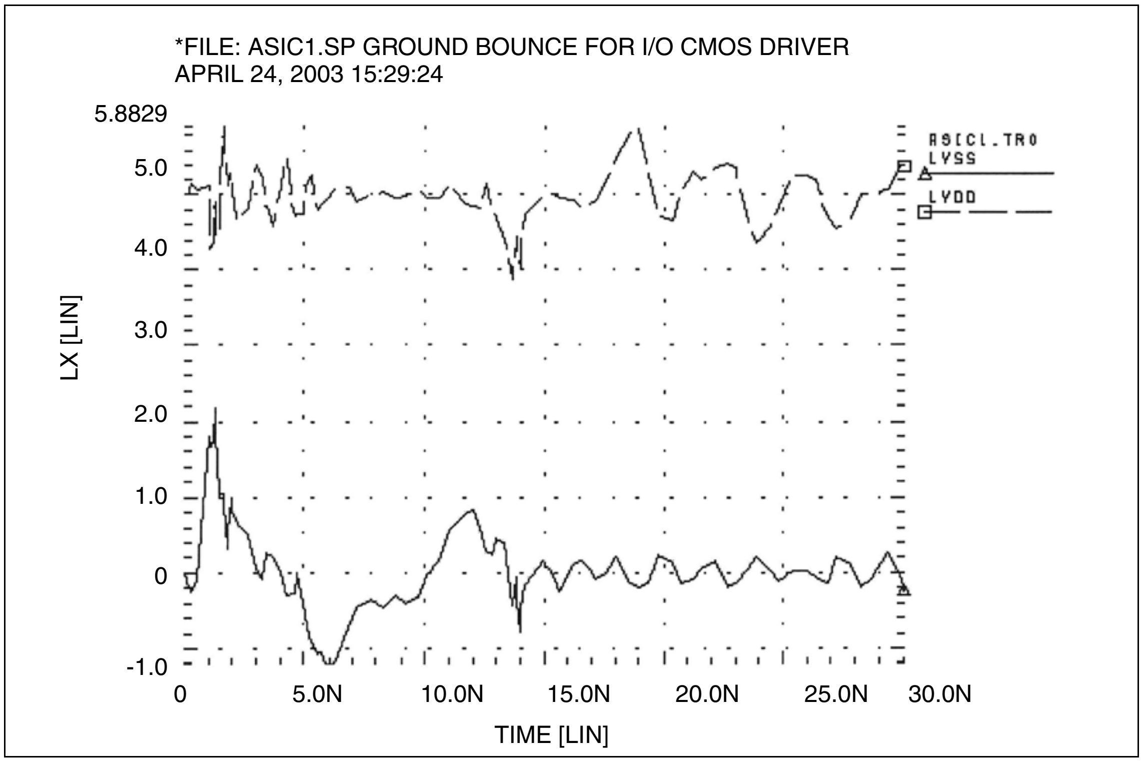


Figure 211 Asic1.sp Demo Local Supply Current

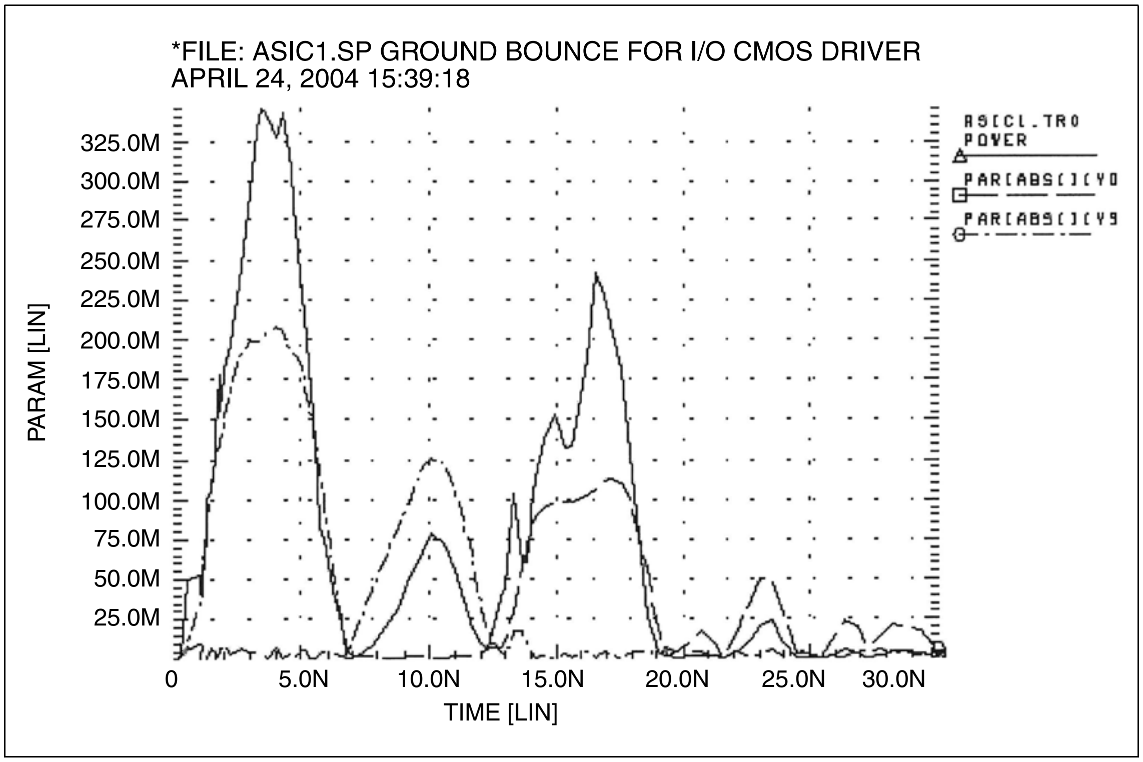


Figure 212 Asic1.sp Demo Input and Output Signals

CMOS Output Driver Example Input File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/asic1.sp
```

Temperature Coefficients Demo

SPICE-type simulators do not always automatically compensate for variations in temperature. The simulators make many assumptions that are not valid for all technologies. Many of the critical model parameters in HSPICE 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

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/ciropt/opttemp.sp
```

Optimization Section

```
.model optmod opt
.dc data=RES_TEMP optimize=opt1
+           results=r@temp1,r@temp2,r@temp3
+           model=optmod
.param tc1r_opt=opt1(.001,-.1,.1)
.param tc2r_opt=opt1(1u,-1m,1m)
.meas r@temp1 err2 par(R_meas_t1) par('1.0 / lv1(r-25)')
.meas r@temp2 err2 par(R_meas_t2) par('1.0 / lv1(r0) ')
.meas r@temp3 err2 par(R_meas_t3) par('1.0 / lv1(r+25) ')
* * Output section *
.dc data=RES_TEMP
.print 'r1_diff'=par('1.0/lv1(r-25)')
+      'r2_diff'=par('1.0/lv1(r0) ')
+      'r3_diff'=par('1.0/lv1(r+25)')
.data RES_TEMP R_meas_t1 R_meas_t2 R_meas_t3
950 1000 1010
950 1000 1010
.enddata
.end
```

Modeling Wide-Channel MOS Transistors

If you select an appropriate model for I/O cell transistors, simulation accuracy improves. For wide-channel devices, model the transistor as a *group* of transistors, connected in parallel, with appropriate RC delay networks. If you model the device as only *one* transistor, the polysilicon gate introduces delay.

When you scale to higher-speed technologies, the area of the polysilicon gate decreases, reducing the gate capacitance. However, if you scale the gate oxide thickness, the capacitance per unit area increases, which also increases the RC product.

Example

The following example illustrates how scaling affects the delay. For example, for a device with:

- Channel width=100 microns.
- Channel length=5 microns.
- Gate oxide thickness=800 Angstroms.

The resulting RC product for the polysilicon gate is:

$$\text{Rpoly} = \frac{W}{L} \cdot 40 \qquad \text{poly} = \frac{Esio \cdot nsi}{tox} \cdot L \cdot W$$

$$\text{Rpoly} = \frac{100}{5} \cdot 40 = 800, \quad \text{Co} = \frac{3.9 \cdot 8.86}{800} \cdot 100 \cdot 5 = 215fF \quad \text{RC=138 ps}$$

For a transistor with:

- Channel width=100 microns.
- Channel length=1.2 microns.
- Gate oxide thickness=250 Angstroms.

The resulting RC product for the polysilicon gate is:

$$\text{Rpoly} = \frac{channel\ width}{channel\ length} \cdot 40$$

$$\text{Co} = \frac{3.9 \cdot 8.86}{Tox} \cdot channel\ width \cdot channel\ length \quad \text{RC=546 ps}$$

You can use a nine-stage ladder model to model the RC delay in CMOS devices.

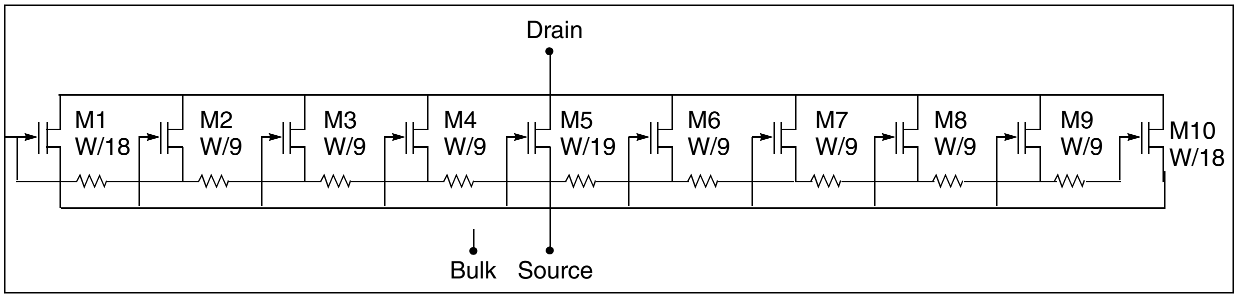


Figure 213 Nine-stage Ladder Model

In this example, the nine-stage ladder model is in data file $*installdir*/demo/hspice/apps /asic3.sp. To optimize this model, HSPICE uses measured data from a wide channel transistor as the target data\. Optimization produces a nine-stage ladder model, which matches the timing characteristics of the physical data (HSPICE RF does not support optimization). HSPICE compares the simulation results for the nine-stage ladder model, and the one-stage model by using the nine-stage ladder model as the reference. The one-stage model results are about 10% faster than actual physical data indicates.

Example

You can find the sample Nine-Stage Ladder model netlist for this example in the following directory:

`$installdir/demo/hspice/apps/asic3.sp`

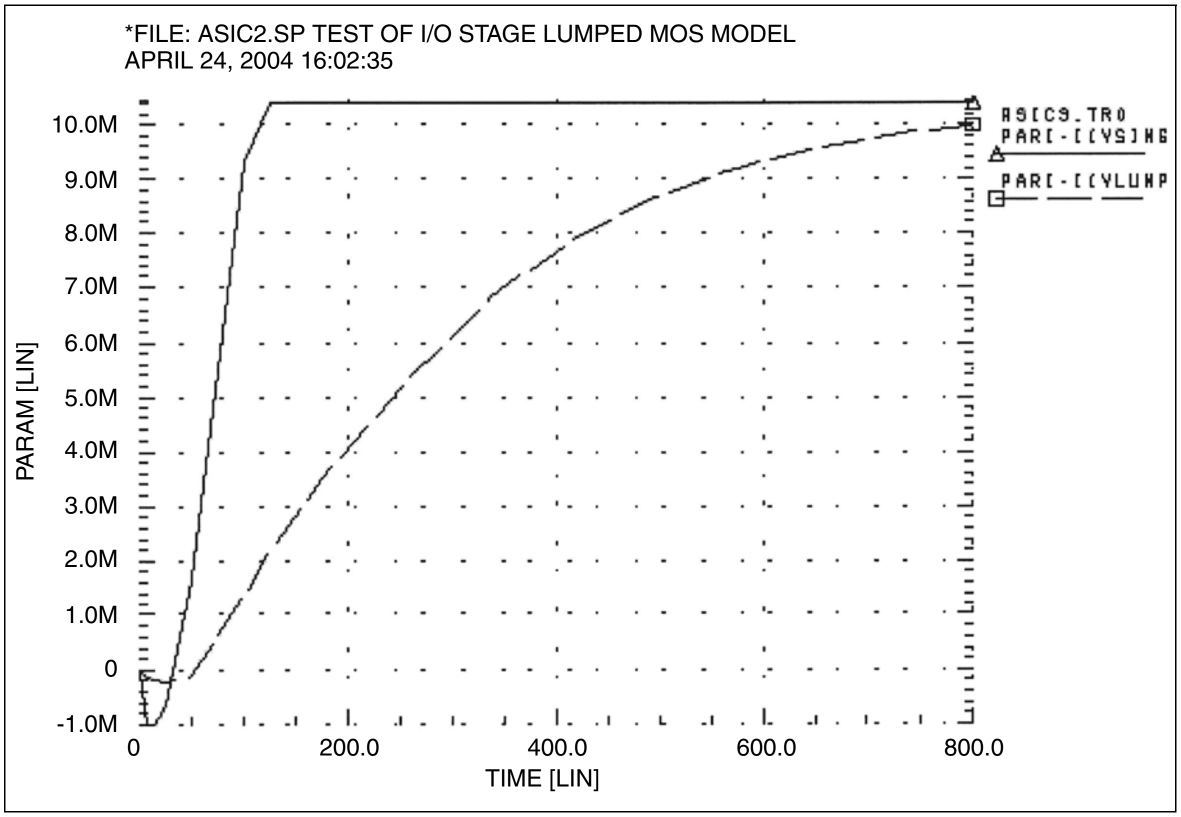


Figure 214 Asic3 Single vs. Lumped Model

Listing of Demonstration Input Files

The following section are listings of shipped demonstration files for illustrating HSPICE functionality.

The demo file groups include:

- HSPICE Integration to Cadence^{TM} Virtuoso® Analog Design Environment Tutorial Examples
- Applications of General Interest Examples

- Behavioral Application Examples
- Benchmark Examples
- Bisection-Timing Analysis Examples
- BJT and Diode Examples
- Cell Characterization Examples
- Circuit Optimization Examples
- Device Optimization Examples
- Encryption Examples
- Fourier Analysis Examples
- Filters Examples
- IBIS / Examples
- Magnetics Examples
- MOSFET Device Examples
- Signal Integrity Examples
- Sources Examples
- S-parameter Examples
- Transmission Lines Examples
- Transmission (W-element) Line Examples
- Variability Examples
- Verilog-A Examples

HSPICE Integration to Cadence^{TM} Virtuoso® Analog Design Environment Tutorial Examples

Table 85 HSPICE Integration Tutorial Examples

File Name	Location: $installdir/demo/hspice/aa_integ
PLL_Demo_5/ PLL_Demo_61	This directory contains two directory suites of files for ADE versions 5.1xx and 6.1xx (required to run a guided tutorial found in the first chapter of the HSPICE Integration User Guide, including a Verilog-A example). See Quick-Start Tutorial in the *HSPICE Integration to Cadence^{TM} Virtuoso® Analog Design Environment User Guide*

Applications of General Interest Examples

Table 86 Applications of General Interest Examples

File Name	Location: $installdir/demo/hspice/apps
alm124.sp	AC, noise, and transient op-amp analysis
alm124.inc	macro model
alter2.sp	ALTER example
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
biaschk.sp	Apply bias check analysis on D flip flop
bjtdiff.sp	BJT diff amp with every analysis type
bjtschmt.sp	Bipolar Schmidt trigger
bjtsense.sp	Bipolar sense amplifier

Table 86 Applications of General Interest Examples (Continued)

File Name	Location: $installdir/demo/hspice/apps
cellchar.sp	Characteristics of ASIC inverter cell
four.sp	CMOS inverter applied with Fourier analysis
gaasamp.sp	Simple GaAsFET amplifier
gen28.inc	Model library file
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
monte_test.tar	Suite of 20 DC test files named test1.sp through test20.sp to test combinations of resistors, subskts, model/instance parameters, etc.
mos2bit.sp	Two-bit MOS adder
noise.sp	Noise analysis on two parallel MOS
opampdcm.sp	DCmatch analysis, opamp
quickAC.sp	AC analysis on a RC network
quickINV.sp	Transient analysis on a inverter
quickTRAN.sp	Tran on a resistor divider
rc_monte.sp	Transient Monte Carlo on Resistor
sclopass.sp	Switched-capacitor low-pass filter
tlib1	model library file
tlib2	model library file

Table 86 Applications of General Interest Examples (Continued)

File Name	Location: $installdir/demo/hspice/apps
tlib3	model library file
tlib4	model library file
trist_buf_opt.sp	Tri-State buffer optimization
wildchar.sp	Wildcard print and probe example
worst.sp	Worst case skew models by using .ALTER
xbjt2bit.sp	BJT NAND gate two-bit binary adder

Behavioral Application Examples

Table 87 Behavioral Application Examples

File Name	Location: $installdir/demo/hspice/behave
acl.sp	Acl gate
amp_mod.sp	Amplitude modulator with pulse waveform carrier
behave.sp	AND/NAND gates by using G-, E-elements AND/NAND gates by using G, E Elements
calg2.sp	Voltage variable capacitance
compar.sp	Behavioral comparator with hysteresis
det_dff.sp	Double edge-triggered flip-flop
diff.sp	Differentiator circuit
diff1.sp	Differential block analysis
diode.sp	Behavioral diode by using a PWL VCCS
dlatch.sp	CMOS D-latch by using behaviorals

Table 87 Behavioral Application Examples

File Name	Location: $installdir/demo/hspice/behave
galg1.sp	Sampling a sine wave
idealop.sp	Ninth-order low-pass filter
integ.sp	Integrator circuit
inv_vin_vout.sp	DC sweep of a INV
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
pdb.sp	Phase detector by using behavioral NAND gates
pll.sp	PLL build with BJT
pll_bvp.sp	PLL build with behavioral source
pwl10.sp	Operational amplifier used as a voltage follower
pwl2.sp	PPW-VCCS with a gain of 1 amp/volt
pwl4.sp	Eight-input NAND gate
pwl7.sp	Modeling inverter by using a PWL VCVS
pwl8.sp	Smoothing the triangle waveform by using the PWL CCCS
ring5bm.sp	Five-stage ring oscillator – macromodel CMOS inverter
ringb.sp	Ring oscillator by using behavioral model
sampling.sp	Sampling a sine wave
swcap5.sp	Fifth-order elliptic switched capacitor filter
switch.sp	Test for PWL switch element
swrc.sp	Switched capacitor RC circuit

Table 87 Behavioral Application Examples

File Name	Location: $installdir/demo/hspice/behave
vcob.sp	Voltage-controlled oscillator by using PWL functions

Benchmark Examples

Table 88 Benchmark Examples

File Name	Location: $installdir/demo/hspice/bench
bigmos1.sp	Large MOS simulation
demo.sp	Quick demo file to test installation
example.sp	CMOS amplifier
digstim.vec	Vector stimulus file for m2bit_v.sp
m2bit.sp	72-transistor two-bit adder – typical cell simulation
m2bit_v.sp	Same as m2bit.sp except uses vector stimulus file
senseamp.sp	Bipolar analog test case

Bisection-Timing Analysis Examples

Table 89 Bisection-Timing Examples

File Name	Location: $installdir/demo/hspice/alge
dff_push.sp	DFF pushout bisection search for setup time
dff_top.sp	DFF bisection search for setup time
fig26_4.sp	Early, Optimal and Late Setup Times of DFF
inv_a.sp	inverter bisection (pass-fail)

Table 89 Bisection-Timing Examples (Continued)

File Name	**Location: $installdir/demo/hspice/alge**
tsmc018.m	TSMC model file used by dff_push.sp

BJT and Diode Examples

Table 90 BJT and Diode Device Examples

File Name	Location: $installdir/demo/hspice/bjt
bjtbeta.sp	plot BJT beta
bjtft.sp	plot BJT FT by using s-parameters
bjtgm.sp	plot BJT Gm, Gpi
dpntun.sp	junction tunnel diode
hicum.sp	HICUM BJT MOS terminal characterization
mextram.sp	I-V characteristics of a MEXTRAM BJT
mextram_ac.sp	AC analysis of a MEXTRAM BJT
mextram_dc.sp	DC analysis of a MEXTRAM BJT
mextram_tran.sp	Tran analysis of a MEXTRAM BJT
net_ana.sp	NET analysis of a BJT
quasisat.sp	quasisat.sp comparison of bjt Level1 and Level2
self-heat.sp	VBIC BJT with self heating feature
tun.sp	tunnel oxide diode
vbic.sp	DC of a VBIC BJT
vbic99_ac.sp	NET analysis of a VBIC99 BJT
vbic99_dc.sp	DC analysis of a VBIC99 BJT

Table 90 BJT and Diode Device Examples

vbic99_tran.sp	TRAN analysis of a VBIC99 BJT

Cell Characterization Examples

Table 91 Cell Characterization Examples

File Name	Location: /$installdir/demo/hspice/cchar
digin.sp	U-element with digital output
gen28.inc	level 28 model library used by netlists
inv3.sp	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	sigma.sp sweep MOSFET -3 sigma to +3 sigma by using measure output
tdgtl.a2d	Viewsim A2D HSPICE input file
tdgtl.d2a	Viewsim D2A HSPICE input file
tdgtl.sp	two-bit adder by using D2A Elements

Circuit Optimization Examples

Table 92 Circuit Optimization Examples

File Name	Location: $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

Device Optimization Examples

Table 93 Device Optimization

File name	Location: *$installdir/demo/hspice/devopt*
beta.sp	LEVEL=2 beta optimization
bjtopt.sp	s-parameter optimization of a 2n6604 BJT
bjtopt1.sp	2n2222 DC optimization
bjtopt2.sp	2n2222 Hfe optimization
d.sp	diode, multiple temperatures
dcopt1.sp	1n3019 diode, I-V and C-V optimization

Table 93 Device Optimization

File name	**Location: *$installdir/demo/hspice/devopt***
jopt.sp	300u/1u GaAs FET, DC optimization
ml13opt.sp	MOS LEVEL=2 I-V optimization
ml2opt.sp	MOS LEVEL=3 I-V optimization
opt_bjt.sp	T2N9547 BJT Optimization

Encryption Examples

File name	**Location: $installdir/demo/hspice/encryption**
8-byte_key.tar	Suite of files demonstrating how to set up an 8-byte_key encryption file.
traditional.tar	Suite of files demonstrating how to set up a traditional (free_lib) encryption file.
triple_DES.tar	Suite of files plus a directory of 2 lib_DES libraries demonstrating how to set up a 3DES encryption file.
README and auxiliary files	

Fourier Analysis Examples

Table 94 Fast Fourier Transform Examples

File Name	**Location: $installdir/demo/hspice/fft**
fft5.sp	FFT analysis, data-driven transient analysis
fft6.sp	FFT analysis, sinusoidal source

Table 94 Fast Fourier Transform Examples

File Name	Location: $installdir/demo/hspice/fft
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

Filters Examples

Table 95 Filters

File Name	Location: $installdir/demo/hspice/filters
bandstopl.sp	band reject filter, AC and transient analysis
fbp_1.sp	bandpass LCR filter, measurement
fbp_2.sp	bandpass LCR filter, pole/zero
fbpnet.sp	bandpass LCR filter, using .NET
fbprlc.sp	LCR AC analysis for resonance
fhp4th.sp	high-pass LCR, fourth-order Butterworth filter, pole-zero analysis
fkerwin.sp	pole/zero analysis of Kerwin’s circuit
flp5th.sp	low-pass, fifth-order filter, pole-zero analysis
flp9th.sp	low-pass, ninth-order FNDR, with ideal op-amps, pole-zero analysis
lcline.sp	LC line model using Laplace behavioral elements
low_pass.sp	behavioral model using E and G elements
low_pass9a.sp	active low pass filter using behavioral opamp models
lowloss.sp	RL line model using Laplace behavioral elements
ninth.sp	active low pass filter using Laplace elements
phaseshift.sp	Behavioral model using G table element
rcline.sp	RC line model using Laplace elements

IBIS / Examples

Table 96 IBIS Modeling Files

File name	**Location: $installdir/demo/hspice/ibis**
at16245.ibs	IBIS model file, used in iob_ex1.sp example file
iob_ex1.sp	Using IBIS buffer example
cmpt1.ibs	IBIS model file
ebd.ebd	IBIS EBD file example
ebd.sp	Using EBD files example
pinmap.ebd	IBIS EBD file example, uses pin mapping
pinmap.sp	Using EBD files example
pinmap.ibs	IBIS model file
readme	readme file for ICM examples
bga_1.sp	Using ICM with nodepath description example
bga_example.icm	ICM example file
s_w_test_GHz_db.s4p	TouchStone file for ICM example, called by test1.icm
sect2_s_2.inc	S-parameter model call, used by test1.sp
test1.icm	ICM example file
test1.sp	Using ICM with nodepath description and S-element
sect3_rlgc_4.inc	RLGC file used by test1.sp
sect_w_4.inc	RLGC file used by test1.sp
test1.icm	ICM example file
test1.sp	Using ICM with treepath and rlgc data example
complex.icm	ICM example file
complex.sp	Using ICM with swath matrix expansion

Magnetics Examples

Table 97 Magnetics

File Name	Location: Magnetics $installdir/demo/hspice/mag
aircore.sp	Air-core transformer circuit
bhloop.sp	Magnetic core model, plot B-H loop characteristics
jiles.sp	Effects of core model parameters on B-H loop characteristics
mag2.sp	Three primary, two secondary, magnetic-core transformer
magcore.sp	Magnetic-core transformer circuit
royerosc.sp	Royer magnetic-core oscillator
tj2b.sp	Hysteresis effects in magnetic cores
tj_opt.sp	Optimizing magnetic core parameters

MOSFET Device Examples

Table 98 MOSFET Devices

File Name	Location: $installdir/demo/hspice/mos
calcap.sp	Calculate AC gate capacitance
calcap.ic0	Results file from calcap.sp
calcap.ic1	Results file from calcap.sp
calcap.lis	Results file from calcap.sp
calcap.results	Results file from calcap.sp
calcap.st0	Results file from calcap.sp

Table 98 MOSFET Devices

File Name	Location: $installdir/demo/hspice/mos
capop0.sp	Plot MOS capacitances, LEVEL=2
capop1.sp	Plot MOS capacitances, LEVEL=2
capop2.sp	Plot MOS capacitances, LEVEL=2
cascode.sp	MOS Cascode amplifier example, show effect of level=3 impact ionization parameter
chrgpump.sp	Charge-conservation test, charge pump using LEVEL=3 MOS
gatecap.sp	DC gate capacitance calculation
mcap2_a.sp	MOS charge conservation capacitances
mcap3.sp	MOS charge conservation capacitances
ml13iv.sp	Plot I-V for LEVEL=13
ml13opt.sp	Optimizing MOS LEVEL=13 model parameter
ml27iv.sp	Plot I-V for LEVEL=27 SOSFET
ml5iv.sp	MOS LEVEL=5 example
mosiv.sp	Plot I-V for files that you include
mosivcv.sp	Example of plotting I-V and C-V curves, uses LEVEL=3 model
nch0.inc	MOS model for mosiv.sp and cap_m.sp
selector.sp	Automatic model selector for width and length
ssoi.sp	Floating bulk model
t1.sp	MOS LEVEL=13 TOX calculation test
tempdep.sp	MOS LEVEL=3 temperature dependence
tgam2.sp	LEVEL=6, gamma model

Signal Integrity Examples

See also IBIS / Examples,S-parameter Examples, Transmission Lines Examples, and Transmission (W-element) Line Examples.

Table 99 Signal Integrity

File Name	Location: $installdir/demo/spice/si
iotran.sp	Signetics I/O buffer with transmission lines example
ipopt.sp	TDR Optimization Example
qa8.sp	Xilinx I/O buffer with transmission lines example
qabounce.sp	Ground bounce example

Sources Examples

Table 100 Sources

File Name	Location: $installdir/demo/hspice/sources
amsrc.sp	Amplitude modulation source example
datadriven_pwl.sp	Data driven PWL source example
eelm.sp	E-element AC source example
exp.sp	Exponential independent source example
prbs.sp	PRBS source example
pulse.sp	Pulse source example
pwl.sp	Repeated piecewise-linear source example
rtest.sp	Voltage-controlled resistor, inverter chain
sffm.sp	Single-frequency, FM modulation source example
sin.sp	Sinusoidal source, waveform example

Table 100 Sources (Continued)

File Name	Location: $installdir/demo/hspice/sources
uelm.sp	Digital U-element source example
uelm.d2a	Part of uelem.sp example
vcr1.sp	Switched-capacitor network by using G-switch

S-parameter Examples

Table 101 S-Parameter Examples

File Name	Location: $installdir/demo/hspice/sparam
diffamp_s.sp	Mixed mode S-parameter example, differential amplifier
mixed2p.s4p	Mixed mode S-parameters in Touchstone format, used by mixedmode_s.sp
mixedmode_s.sp	Mixed mode S-parameter example, transmission line
sparam.sp	Using S-parameter model in SP model format
spciti.sp	S-element example, calling CITI format S-parameter file
spmod.sp	S-element example, calling Touchstone format S-parameter file
ss_citi.citi	CITI format S-parameter file example
ss_ts.s2p	TouchStone format S-parameter file example

Transmission Lines Examples

Table 102 Transmission Lines (tline) Example Files

File name	Location: $installdir/demo/hspice/tline
rcfilt.inc	RC filter macro model
strip1.sp	U-element, two microstrips, in series (8 mil and 16 mil wide)
strip2.sp	U-element, two microstrips, coupled together
stripline.sp	U-element strip line example
uele.sp	U-element, three coupled lines

Transmission (W-element) Line Examples

Table 103 twline Demo Files

File Name	Location: $installdir/demo/hspice/twline
ex1.sp	4 conductor RLGC model W-element example
ex2.sp	4 conductor RLGC file W-element example
ex3.sp	4 conductor W-element using U-element parameters
example.rlc	RLGC file used by ex2.sp
fs_ex1.sp	Field solver, conductor above ground plane
fs_ex2.sp	Field solver, three trace example
fs_ex3.sp	Field solver, coupled line example
fs_ex4.sp	Field solver, Monte Carlo example
petl_ex1.sp	Field solver, 1 conductor coax example

Table 103 twline Demo Files (Continued)

File Name	Location: $installdir/demo/hspice/twline
petl_ex2.sp	Field solver, 2 conductor coax example
rlgc.rlc	RLGC file used by rlgc.sp
rlgc.sp	W-element using RLGC file
umodel.sp	4 conductor W-element using U-element parameters

Variability Examples

Table 104 Variation Block, Monte Carlo, and Mismatch Demo Files

File name	Location: $installdir/demo/hspice/variability
matrix.sp	Matrix of 9 resistors for testing spatial variation with Monte Carlo
monte_test.tar	test1.sp through test20.sp: An array of circuits for Monte Carlo
opampacm.sp	Operational amplifier for ACMAtch testing with Variation Block
opampdcm.sp	Operational amplifier for DCMatch testing
opampmc.sp	Operational amplifier for Monte Carlo testing with Variation Block

Verilog-A Examples

Table 105 HSPICE Verilog-A: Netlist and Verilog-A Files

File name	Location: $installdir/demo/hspice/veriloga
biterrorrate.sp	Bit error rate counter
biterrorrate.va	
bjt.sp	BJT model

Table 105 HSPICE Verilog-A: Netlist and Verilog-A Files (Continued)

File name	Location: $installdir/demo/hspice/veriloga
bjt.va	
colpitts.sp	Colpitts BJT oscillator
colpitts.va	
dac.sp	DAC and ADC
dac.va	
deadband.sp	Deadband amplifier
deadband.va	
ecl.sp	ECL inverter
opamp.sp	Opamp
opamp.va	
pll.sp	Behavioral model of PLL
pll.va	
psfet.sp	Parker Skellern FET model
psfet.va	
resistor.sp	Very simple Verilog-A resistor model
resistor.va	
sample_hold.sp	Sample and hold
sample_hold.va	
sinev.sp	Simple voltage source
sinev.va	

32

Warning/Error Messages

Provides an overview of the type of warnings and error messages that HSPICE prints and troubleshooting measures to take when possible.

This chapter briefly describes:

- Warning Messages
- Error Messages
- Exit Codes

Warning Messages

The following sections present these topics:

- Topology Warnings
- Model Warnings
- Control Option Warnings
- Device Warnings
- Analysis Warnings

Topology Warnings

Topology Integrity

When HSPICE encounters topology integrity issues, it reports warning messages similar to the four types shown:

```
**warning**  only 1 connection at node 1:net0107 defined in subckt
bg: called in element 12:mn0 defined in subckt bg at line 161
within the hspice source, library or include file

**warning** both nodes of resistor 1:rinp defined in subckt opa350
are connected together

**warning** 2:r11 defined in subckt pwdr resistance limited to
1.000E-05

**warning** the following singular supplies were terminated to 1
meg resistor
 supply   node1                         node2
 vdd18    0:dvdd18 defined in subckt 0  0:0 defined in subckt0
 vdd1p8   0:dvdd1p8 defined in subckt 0  0:0 defined in subckt 0
```

No DC Path to Ground

The warning for no dc path to ground, effective from 2007.09, is:

```
**warning**  no dc path to ground from node 13:fl defined in
subckt d****01 now it is connected with gdcpath
```

Duplicate Initialization

If a node is initialized using a .ic or .nodeset more than once, the following warning is issued:

```
**warning** a duplicate initialization for node=1620:ram***,
keeping last value 0.900 only.
```

Model Warnings

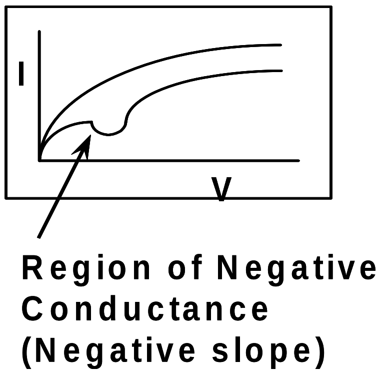
Zero or Negative Conductance

The following two examples show sample warning messages for negative or zero conductance:

```
**warning** negative-mos conductance = 0:m1 iter= 2
vds,vgs,vbs =      4.22          2.12         0.925
gm,gds,gmbs,ids=  1.707E-03   9.366E-05  -1.380E-04  5.040E-04

**warning** conductance of 0. on node 0:net4823 iter= 1
```

The typical causes for these warnings are modeling problems in the subthreshold equations (if in cutoff) or channel length modulation equations (if in saturation). The magnitude reported in the warnings indicates the magnitude of the conductance (leakage) that must be placed across the drain and source to offset the effect of the negative conductance. Typically, .option GMINDC and GMIN can be used to do this.



Encryption-Related Warnings

```
**warning** Data associated with encrypted blocks were suppressed
due to encrypted content

 **warning** Some parameters in  encrypted block are defined as
an expression containing output signals. which may cause incorrect
result. Suggest to use user defined functions to replace.
```

Model Binning Warnings

```
**warning** (L65***.mdl:2)  model n_10_rvl    device geometries
will not be checked against the limits set by lmin, lmax, wmin
and wmax. To enable this check, add a period(.) to the model name
(i.e. enable model selector).
```

Key Model Parameter Checking

```
*** warning ***: area for diode can not be 0.0, reset to 1e-12
(default value)
```

Parameter Expression Warning

```
**warning** parameter pdt is defined as an expression containing
output signals, which may cause incorrect result. Suggest to use
user defined functions to replace.
```

For example:

```
.Param pdt = “rs*pi*i(node2)+v(out)”
```

Control Option Warnings

RUNLVL

The following is an informational warning about the default RUNLVL option setting.

```
** runlvl is invoked, you can disable it by:
 a) Add option runlvl=0 to your current simulation job.
 b) Copy $installdir/hspice.ini to your HOME directory and
customize it by adding option runlvl=0, which disables it for all
of your simulation jobs.
 c) Re-invoke $installdir/bin/config program and unselect the
option runlvl setting in box 'hspice.ini' which disables it for
whole group simulation jobs.

 ** runlvl is invoked, some options are ignored or automatically
set:
 Options below are automatically set(user setting will overwrite
them):
      if runlvl=6,             .option bypass=0
      if runlvl=[1|2|3|4|5],  .option bypass=2
Options below are ignored, they are replaced by automated
algorithms:
 lvltim    dvdt      ft    fast   trtol  absvar  relvar
 relq  chgtol    dvtr    imin    itl3    rmax

 ** runlvl is invoked, actual option value used by HSPICE are:

     runlvl= 3     bypass= 2    mbypass=  2.00  bytol= 100.00u
```

ACCURATE

```
***accurate option sets default value of the options:
lvltim= 3 dvdt= 2 relvar= 200.00m absvar= 200.00m ft= 200.00m
relmos=  10.00m
 (used for FFT control) fft_accurate= 1
```

FAST

```
**warning** the fast option set the bypass on and the following
options:
dvdt= 3 bytol=  50.00u
```

GMIN, GMINDC

```
**warning**  pivtol too large ** reset to half minimum value of
(gmindc,gmin)
```

Device Warnings

Device warnings are specific to each model.

Device Geometry Check

```
Warning: Pd = 1.36e-06 is less than W.
Model:    0:nch
W = 4.444e-06, L = 5.3e-07
```

Device Parameter Check

```
Warning: Moin = 1568.2 is too large.
Warning: Acde = 0.0350921 is too small.
```

Analysis Warnings

Transient

```
**warning** the third value 0.00000D+00 and the fourth value
1.00000D-12 are both smaller than the second value 5.00000D-10,
so the transient statement is interpreted as'.tran tstep tstop
tstart delmax'.
```

Example:

```
.tran 1n 1u 0 1p
.tran 1n 1u 1p 2u
```

Bisection

With option OPTCON=1

```
**warning** endpoints have same sign in bisection
For x             =   0.0000    , y             =   0.0000    .
For x             =   1.0000    , y             =   1.0000    .
Both of these are on the same side of the goal value y        =
0.30000   .
```

Multiple Results:

```
**warning** multiple results used in bisection
```

Example:

```
.tran 1.0e-9 8.0e-9 sweep optimize=opt1 results=y,z
model=opt_model
```

Pass/Fail

```
**warning** passfail does not support more than one result, only
first one is validated
```

Example:

```
.model opt_model opt method=passfail relin=0.01 relout=0.01
.tran 1.0e-9 8.0e-9 sweep optimize=opt1 results=y,z
model=opt_model
```

Measure

```
**warning** measure results may be incorrect since initial start
time is non-zero.
 vin_pp=  1.4855E-01  from=  1.5000E-05     to=  2.0000E-05
```

```
**warning** the Equation Evaluation form of the .MEASURE statement
must not be a function of node voltages or branch currents.
Unexpected results may incur.
```

Example:

```
.MEAS VARG PARAM=‘(V(2) + V(3))/2’
```

.DC and .OP Analysis Warnings

When both DC and TRAN source are defined:

```
**warning** dc voltage reset to initial transient source value
in source 0:vclk new dc=  0.0000D+00
```

Example:

```
vlo2 in gnd dc vhaf sin(0 '(pwr(10,((toin)/20)))*(1e-6)*SQ2' fq
0 0 180)
```

Character line limit warning

The HSPICE line limit is 1024 characters.

```
**warning** node full pathname length in .ic file greater than
limit, node NOT initialized in the save file   nodeset.ic
```

Autoconverge overflow message

```
**warning** Due to a floating point overflow problem, the damped
pseudo-tran method was used. Also, gmindc was set to 1.0000E-11
```

Difficult operating point calculation warning message

```
**warning** This was a difficult operating point. You can speed
up your simulation by specifying:.OPTION CONVERGE=4
```

Auto-convergence flow messages

```
convergence problems in dc sweep curves at 15.894 resimulating
with dc convergence controls

**diagnostic** dc convergence failure, resetting dcon option to
1 and retrying with dcon=1, it converged for gmindc=   5.500E-14

**diagnostic** dc convergence failure,
 resetting dcon option to 2 and retrying.

**diagnostic** although this circuit has failed to converge to
gmindc= 1.000E-15, it did converge to a gmindc= 7.662E-15 for
most circuits a value of gmindc 1e-7 or less, is acceptable
```

Operating point diagnostic failure messages

```
**diagnostic** number of iteration exceeds min (7000,
20*itl1)=7000 in pseudo tran process (converge=1 process). Usually
this happens when the models are discontinuous, or there are
uninitialized bi-stable cells (flip-flop) in the circuit. By
setting options dcon=-1 and converge=-1 you can disable auto
convergence process. Retry the run, non-convergence diagnostics
will provide useful information about the nodes and devices which
can be used to work around the non-convergence problems.
```

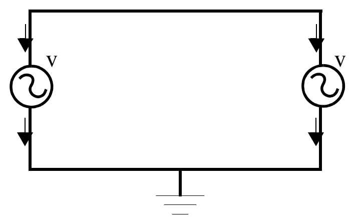
Error Messages

The following sections present these topics:

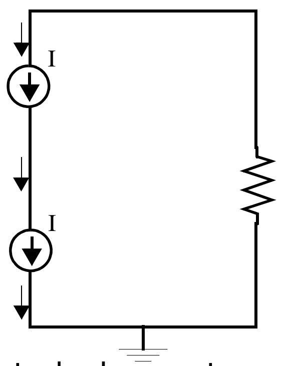
- Topology Errors
- Model Errors
- Analysis Errors

Topology Errors

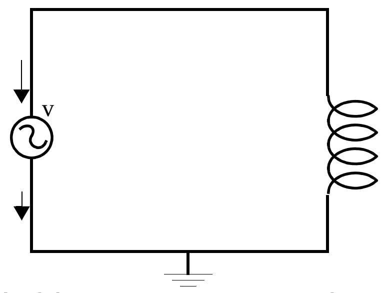
When constructing the circuit description HSPICE does not allow certain topologies. Topology errors will be reported according the following circumstances:



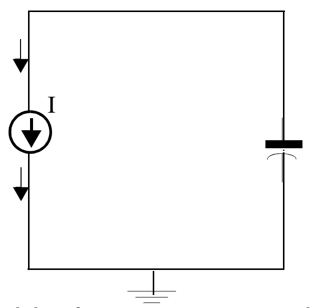
No voltage loops: no voltage sources in parallel with no other elements



No stacked current sources: no current sources in series



No ideal voltage source in closed inductor loop



No ideal current source in closed capacitor loop

Negative or 0 multiplier is not allowed

```
**error** Value of multiplier parameter in 1:x1 is less than or
equal to 0
```

Model Errors

Undefined Model

```
**error** model name pch in the element 0:mp is not defined.
```

Redundant Model Definition

```
**error**  above line attempts to redefine tnl
**error** (../models/res2:4) difficulty in reading input
```

Undefined parameter

```
**error** no definition for 0:rsit was called by      0:rin
**error** no definition for 0:toxn it was called by      0:n
```

Analysis Errors

.DC and Operating Convergence Errors

No convergence error

```
**error** no convergence in operating point
```

No convergence at a .DC sweep point

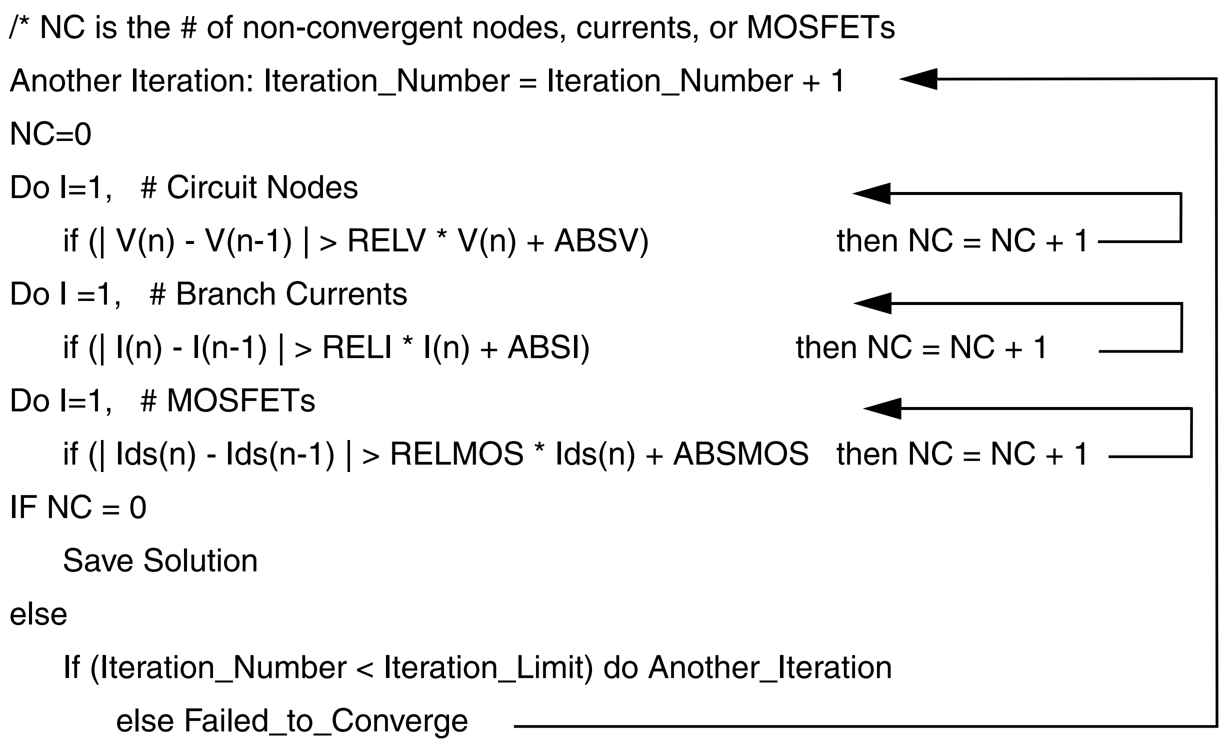
```
**error** no convergence in dc sweep curves at   15.851
```

Operating Point Debugging Information

```
*** hspice diagnostic *** nonconvergent voltage failures= 33803
nonconvergent element current failures= 1
```

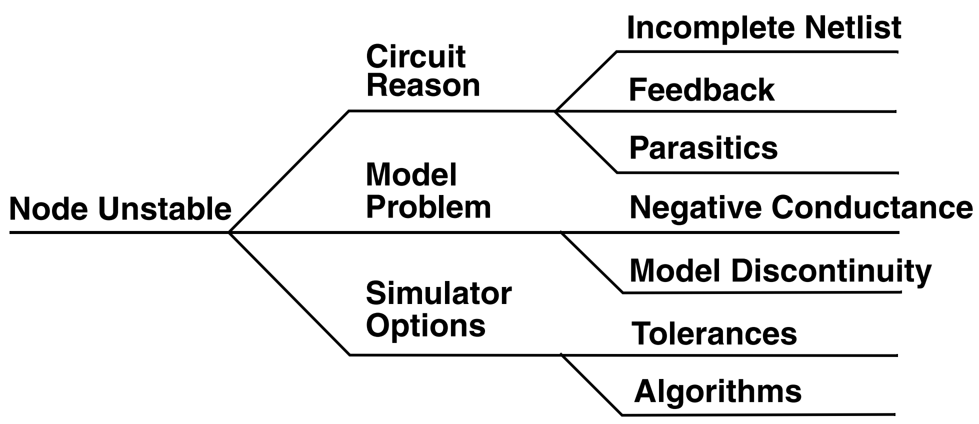
...

Convergence Termination Criteria



```
/* NC is the # of non-convergent nodes, currents, or MOSFETs
Another Iteration: Iteration_Number = Iteration_Number + 1
NC=0
Do I=1,   # Circuit Nodes
     if (| V(n) - V(n-1) | > RELV * V(n) + ABSV)              then NC = NC + 1
Do I =1,   # Branch Currents
     if (| I(n) - I(n-1) | > RELI * I(n) + ABSI)             then NC = NC + 1
Do I=1,   # MOSFETs
     if (| Ids(n) - Ids(n-1) | > RELMOS * Ids(n) + ABSMOS   then NC = NC + 1
IF NC = 0
     Save Solution
else
     If (Iteration_Number < Iteration_Limit) do Another_Iteration
          else Failed_to_Converge
```

Non-Convergence: Possible Causes



DC/OP Convergence Aids

Aids to Remedy DC Bias Non-Convergence

- Auto-convergence process
- .NODESET/.IC Commands (see the following sections)
- Model-related solutions
- Others, with less impact
  - DCSTEP and GMINDC ramping
  - Source stepping/ramping
  - GSHUNT/CSHDC
  - DV

DC Bias Point Convergence Actions

Take the following actions to resolve issues dealing with DC bias point non-convergence:

- Remove all options *except* acct, node, list, post, and opts
- Allow the auto converge process to proceed
- Review the *.lis* file for convergence hints
- Search for “warning” and “error” messages
- Rerun the simulation

DC Bias Point Troubleshooting with .NODESET and .IC

Non-convergence can occur due to poor initial conditions. Set initial conditions and/or nodesets. For example:

```
.IC v(1)=5v v(abc)=0v v(12)=VDD
```

```
.NODESET v(x1.87)=5v
```

Identify the problem nodes by:

- Reviewing the non-convergent diagnostic table in the listing file
- Identifying non-convergent nodes with unusually high voltages, branch currents, or high error tolerances
- Initializing these nodes
- Reviewing the circuit for un-initialized feedback paths (flip-flops, oscillators, etc.)

Because it is inefficient to manually add .NODESET and/or .IC for a large number of nodes, to set a large number of nodes:

1. Comment out all analysis commands except `.TRAN`.
2. Add `UIC` to the end of the `.TRAN` command.
3. Disable auto-convergence process by setting:

   ```
   .option DCON=-1 CONVERGE=-1
   ```

4. Use `.SAVE [TYPE=<nodeset|ic>] [TIME=<x>]` to store the calculated operating point as a .ic or .nodeset file.
5. Simulate the circuit.
6. Use `.LOAD` for loading the file from the `.SAVE` command.
7. Enable the auto-converge process by removing the `DCON` and `CONVERGE` options.
8. Remove `UIC` from the `.TRAN` command.
9. Re-simulate the circuit.

Troubleshooting Model-Related DC Bias Point Issues

Inappropriate model parameters are usually the cause having to do with units or negative/zero conductance:

- If the issue is units:
  - .OPTION SCALM (global)

- .MODEL SCALM factor (local value within .MODEL statements)

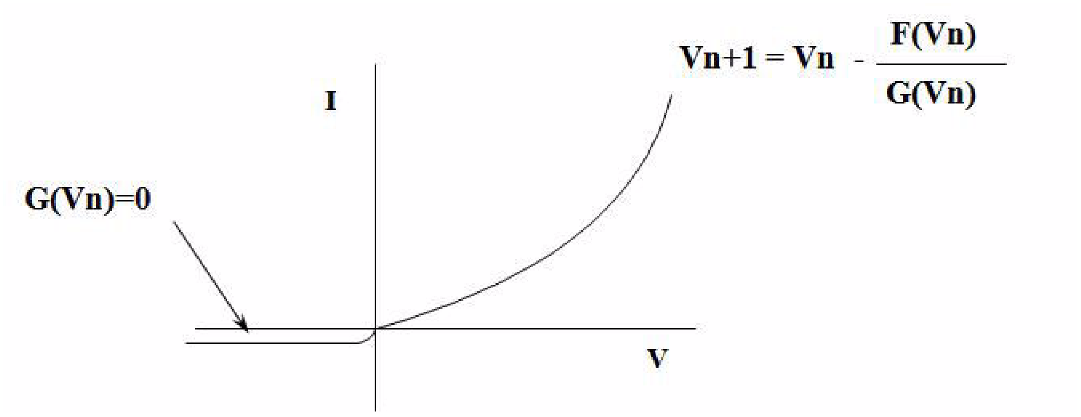
Is a global .OPTION SCALM needed?

- Look at the listing file and review element values.
- Swap in a known good model.

Convergence/Conductance

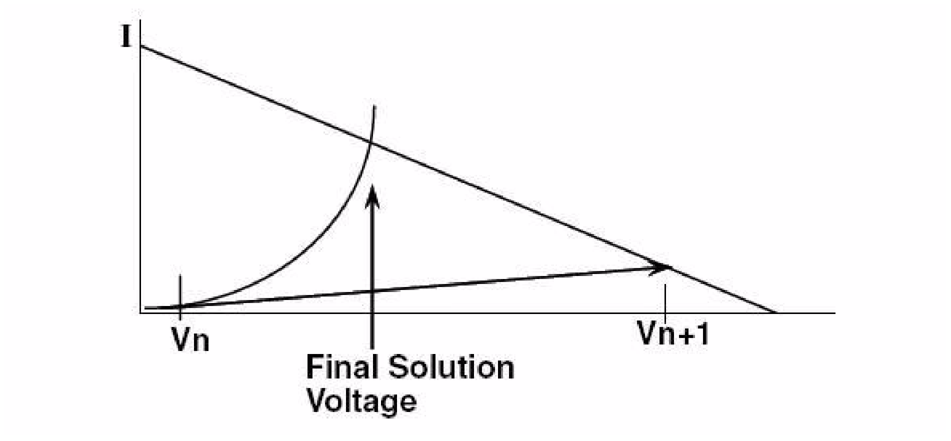
This section describes issues and possible solutions when conductance values impact on convergence. For example: Using a conductance term to predict the next voltage value, can create the problem that if conductance becomes small, the 2nd term becomes large:

- Next voltage value unrealistic
- Causes extra iterations
- Worse: Conductance of zero!



- Since all semiconductor device models contain regions of zero conductance:
  - Shunt R placed in parallel with every PN junction and drain to source
  - Determine smallest parasitic Rp that can be placed across any 2 nodes without influencing circuit behavior
    — G=1/Rp
    — Try setting .OPTION GMINDC=1e-9 GMIN=1e-9
  - Default for both GMIN and GMINDC is 1e-12
- You must ask, “How much leakage is acceptable?”
  - Typically, a setting GMIN=1e-10 does not affect CMOS circuit accuracy

- Larger values of GMIN will affect accuracy and indicate that there may be a model problem



Convergence/Diode Resistance

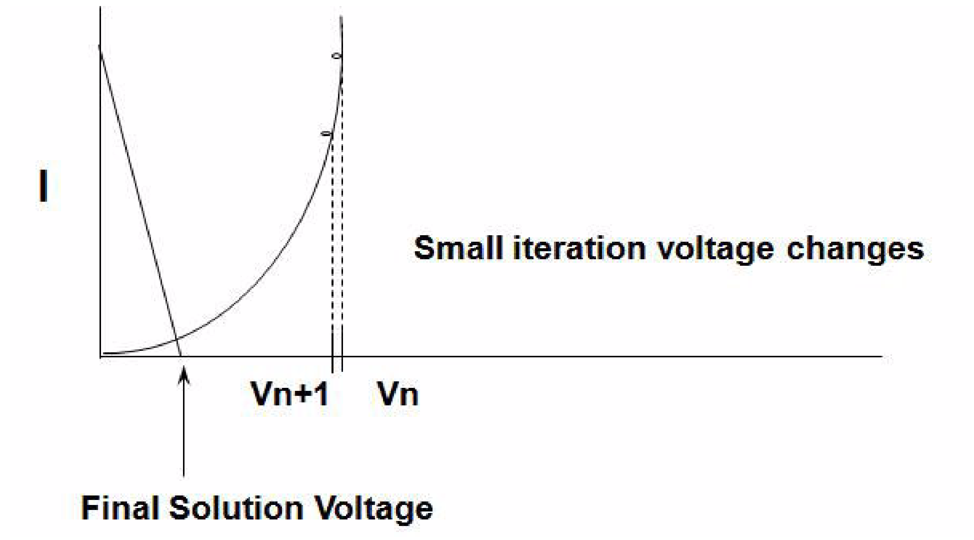
High conductance is troublesome to the algorithm:

Problem: Highly forward-biased diodes (greater than 0.8V)

- Lead to very small iteration-to-iteration voltage changes
- Can cause HSPICE to reach iteration limit before reaching the proper solution voltage

Solution:

- Always specify the series-resistance model parameter for all diodes, bipolar devices, and MOSFETs in the circuit (Default is ZERO ohms).
- At high forward bias, the series resistance dominates the conductance of the device and helps reduce the occurrence of non-convergence.



- Series resistance model parameters for active devices:

Diode	RS
Bipolar transistor	RE and RC
JFET	RD and RS
MOSFET	RD and RS

Analysis Options: DIAGNOSTIC

HSPICE automatically prints out the first occurrence of “negative-mos conductance” in the *.lis* file.

.option DIAGNOSTIC

- Causes all occurrences of negative model conductances to be printed in the *.lis* file
- If the magnitude of the negative conductance is > -1e-8, consult your modeling department or foundry

Transient Analysis Errors and Solutions

Transient Analysis Error

- Most frequent error message:

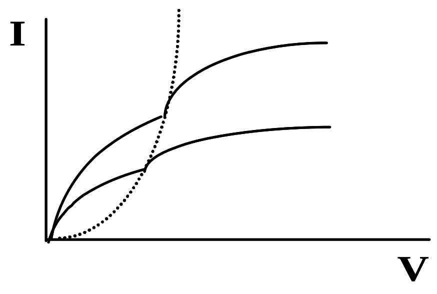
```
**error**  internal timestep too small in transient analysis
```

- Occurs when: Internal timestep < RMIN * TSTEP>
  TSTEP is from .TRAN statement.

Transient Non-Convergence Rapid Voltage Transitions:

- Dynamic timestep control automatically reduces the timestep size
- As the circuit approaches a voltage transition, two potentially conflicting events occur:
  - Semiconductor devices are switching from one region of operation to another.
  - Timestep is reduced.

Model Discontinuities:



- Separate equations used for the different operating regions of an active device
- Most common discontinuities are at the intersection of the linear and saturation regions
- Failure Mechanism
  - Newton-Raphson can oscillate back and forth across the discontinuity
  - Oscillations use up iterations without progressing toward a solution
  - A sweep increases likelihood of hitting model discontinuities

Transient Convergence Aids Corrective actions include:

- Device model capacitance
- GEAR integration
- Use RUNLVL option

Device model capacitance—Transient non-convergence is primarily caused by a combination of model discontinuities and a reduced step size brought on by voltage transitions within the circuit:

- All simulation models should have their associated capacitance terms set to a non-zero value.
- Real models have real capacitances.
- Capacitive Model Parameters:

• Diode	CJO
• Bipolar	CJE, CJC, CJS
• JFET	CGD, CGS
• MOSFET	CGDO, CGSO, CGDO, CBD, CBS, CJ, CJSW

GEAR Integration:

- Numeric integration of time varying currents and voltages are accomplished through Trapezoidal and Gear linearization.
- GEAR integration acts as a filter, removing oscillations that can occur due to the trapezoidal algorithm.
- Circuits that are non-convergent with TRAP will often converge with GEAR.

Why GEAR sometimes converges where TRAP fails:

- Gear integration uses a “weighted average” of past timesteps to determine the next time step.
- This past history helps to project over model discontinuities that exist.

Use the RUNLVL option:

- This option has an enhanced convergence algorithms providing less chance of encountering “time step too small” error.
- RUNLVL=3 similar to default HSPICE setting. This is the default setting.
- RUNLVL=5 similar to set ACCURATE option

Exit Codes

HSPICE prints the these exit codes. The corresponding meanings are as follows.

- 0: Simulation succeeded
- 1: Simulation failed due to errors, e.g., syntax error, non-convergence, etc.
- 2: HSPICE stopped by control+C
- 8: Floating-point exception
- 11: Segmentation fault (invalid memory address access)
- 15: HSPICE stopped by a UNIX `kill` command
- 24: CPU limit exceeded

A

Statistical Analysis

Describes the features available in HSPICE for statistical analysis before the Y-2006.03 release.

Overview

Described in this appendix are the features available in HSPICE for statistical analysis before the Y-2006.03 release. These features are still supported; however, the advanced features described in Chapter 20, Analyzing Variability and Using the Variation Block, and Chapter 21, Monte Carlo Analysis Using the Variation Block Flow represent a significant enhancement over prior approaches.

The previously available documentation on statistical analysis has been reviewed and enhanced for the benefit of those users who are not yet ready to migrate to the new approach. In particular, the last section was added to explain the setup for simulating the effects of global and local variations on silicon with Monte Carlo.

The following subjects are described in this appendix:

- Application of Statistical Analysis
- Analytical Model Types
- Simulating Circuit and Model Temperatures
- Worst Case Analysis
- Getting Started with Traditional Monte Carlo Simulations
- Traditional Monte Carlo Analysis
- Worst Case and Monte Carlo Sweep Example
- Simulating the Effects of Global and Local Variations with Monte Carlo

Application of Statistical Analysis

When you design an electrical circuit, it must meet tolerances for the specific manufacturing process. The electrical yield is the number of parts that meet the electrical test specifications. Overall process efficiency requires maximum yield. To analyze and optimize the yield, HSPICE RF supports statistical techniques and observes the effects of variations in element and model parameters.

Analytical Model Types

To model parametric and statistical variation in circuit behavior, use:

- `.PARAM` statement to investigate the performance of a circuit as you change circuit parameters. For details about the `.PARAM` statement, see the .PARAM statement in the *HSPICE and HSPICE RF Command Reference*.
- Temperature variation analysis to vary the circuit and component temperatures, and compare the circuit responses. You can study the temperature-dependent effects of the circuit, in detail.
- Monte Carlo analysis when you know the statistical standard deviations of component values to center a design. This provides maximum process yield, and determines component tolerances.
- Worst-case corner analysis when you know the component value limit to automate quality assurance for:
  - basic circuit function
  - process extremes
  - quick estimation of speed and power tradeoffs
  - best-case and worst-case model selection
  - parameter corners
  - library files
- Data-driven analysis for cell characterization, response surface, or Taguchi analysis (see Performing Digital Cell Characterization), which automates characterization of cells and calculates the coefficient of polynomial delay for timing simulation. You can simultaneously vary any number of parameters and perform an unlimited number of analyses. This analysis

uses an ASCII file format so HSPICE can automatically generate parameter values. This analysis can replace hundreds or thousands of HSPICE simulation runs.

Use yield analyses to modify:

- DC operating points
- DC sweeps
- AC sweeps
- Transient analysis.

CosmosScope can generate scatter plots from the operating point analysis or a family of curve plots for DC, AC, and transient analysis.

Use `.MEASURE` statements to save results for delay times, power, or any other characteristic extracted in a `.MEASURE` statement. HSPICE generates a table of results in an .mt# file in ASCII format. You can analyze the numbers directly or read this file into CosmosScope to view the distributions. Also, if you use `.MEASURE` statements in a Monte Carlo or data-driven analysis, then the HSPICE output file includes the following statistical results in the listing:

Mean $\frac{x_1 + x_2 + \ldots + x_n}{N}$

Variance $\frac{(x_1 - Mean)^2 + ..(x_n - Mean)^2}{N-1}$

Sigma $\sqrt{Variance}$

Average Deviation $\frac{|x_1 - Mean| + \ldots + |x_n - Mean|}{N-1}$

Simulating Circuit and Model Temperatures

Temperature affects *all* electrical circuits. Figure 215 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.

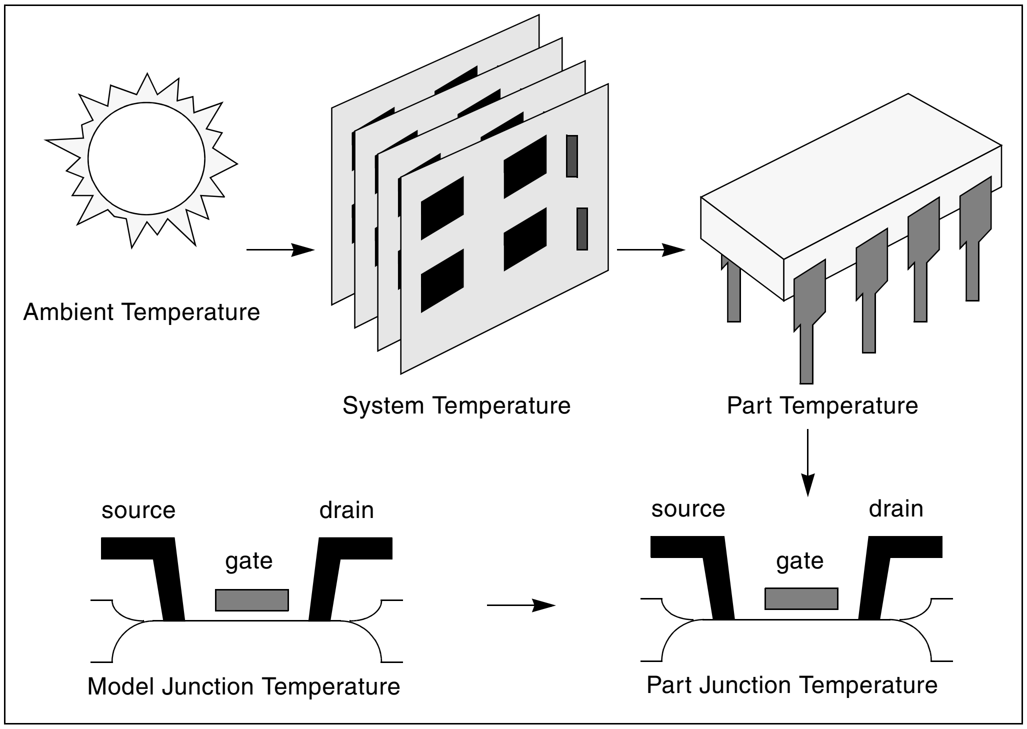


Figure 215 Part Junction Temperature Sets System Performance

HSPICE or HSPICE RF calculates temperatures as differences from the ambient temperature:

$$Tambient + \Delta system + \Delta part + \Delta junction = Tjunction$$

$$Ids = f(Tjunction, Tmodel)$$

Every element includes a DTEMP keyword, which defines the difference between junction and ambient temperature.

Example

The following example uses DTEMP in a MOSFET element statement:

```
M1 drain gate source bulk Model_name W=10u L=1u DTEMP=+20
```

Temperature Analysis

You can specify three temperatures:

- Model reference temperature specified in a `.MODEL` statement. The temperature parameter is usually `TREF`, but can be `TEMP` or `TNOM` in some models. This parameter specifies the temperature, in °C, at which HSPICE or HSPICE RF measures and extracts the model parameters. Set the value of `TNOM` in an `.OPTION` statement. Its default value is 25°C.
- Circuit temperature that you specify using a `.TEMP` statement or the `TEMP` parameter. This is the temperature, in °C, at which HSPICE or HSPICE RF simulates all elements. To modify the temperature for a particular element, use the `DTEMP` parameter. The default circuit temperature is the value of `TNOM`.
- Individual element temperature, which is the circuit temperature, plus an optional amount that you specify in the `DTEMP` parameter.

To specify the temperature of a circuit in a simulation run, use either the `.TEMP` statement, or the `TEMP` parameter in the `.DC`, `.AC`, or `.TRAN` statements. HSPICE or HSPICE RF compares the circuit simulation temperature that one of these statements sets against the reference temperature that the `TNOM` option sets. `TNOM` defaults to 25°C, unless you use the `SPICE` option, which defaults to 27°C. To calculate the derating of component values and model parameters, HSPICE or HSPICE RF uses the difference between the circuit simulation temperature, and the `TNOM` reference temperature.

Elements and models within a circuit can operate at different temperatures. For example, a high-speed input/output buffer that switches at 50 MHz is much hotter than a low-drive NAND gate that switches at 1 MHz). To simulate this temperature difference, specify both an element temperature parameter (`DTEMP`), and a model reference parameter (`TREF`). If you specify `DTEMP` in an element statement, the element temperature for the simulation is:

```
element temperature=circuit temperature + DTEMP
```

Specify the `DTEMP` value in the element statement (resistor, capacitor, inductor, diode, BJT, JFET, or MOSFET statement), or in a subcircuit element. Assign a parameter to `DTEMP`, then use the `.DC` statement to sweep the parameter. The `DTEMP` value defaults to zero.

If you specify TREF in the model statement, the model reference temperature changes (TREF overrides TNOM). Derating the model parameters is based on the difference between circuit simulator temperature and TREF (instead of TNOM).

.TEMP Statement

To specify the temperature of a circuit for a HSPICE or HSPICE RF simulation, use the .TEMP statement.

Worst Case Analysis

Circuit designers often use worst-case analysis when designing and analyzing MOS and BJT IC circuits. To simulate the worst case, set all variables to their 2- or 3-sigma worst-case values. Because several independent variables rarely attain their worst-case values simultaneously, this technique tends to be overly pessimistic and can lead to over-designing the circuit. However, this analysis is useful as a fast check.

Model Skew Parameters

The HSPICE device models include physically-measurable model parameters. The circuit simulator uses parameter variations to predict how an actual circuit responds to extremes in the manufacturing process. Physically-measurable model parameters are called *skew* parameters because they skew from a statistical mean to obtain predicted performance variations.

Examples of skew parameters are the difference between the drawn and physical dimension of metal, postillion, or active layers, on an integrated circuit.

Generally, you specify skew parameters independently of each other, so you can use combinations of skew parameters to represent worst cases. Typical skew parameters for CMOS technology include:

- XL – polysilicon CD (critical dimension of the poly layer, representing the difference between drawn and actual size).
- XW_n, XW_p – active CD (critical dimension of the active layer, representing the difference between drawn and actual size).
- TOX – thickness of the gate oxide.

- `RSH`$_n$, `RSH`$_p$ – resistivity of the active layer.
- `DELVTO`$_n$, `DELVTO`$_p$– variation in threshold voltage.

You can use these parameters in any level of MOS model, within the HSPICE device models. The `DELVTO` parameter shifts the threshold value. HSPICE adds this value to `VTO` for the Level 3 model, and adds or subtracts it from `VFB0` for the BSIM model. Table 106 shows whether HSPICE adds or subtracts deviations from the average.

Table 106 Sigma Deviations

Type	Parameter	Slow	Fast
NMOS	XL	+	-
	RSH	+	-
	DELVTO	+	-
	TOX	+	-
	XW	-	+
PMOS	XL	+	-
	RSH	+	-
	DELVTO	-	+
	TOX	+	-
	XW	-	+

HSPICE selects skew parameters based on the available historical data that it collects either during fabrication or electrical test. For example, HSPICE collects the *XL skew* parameter for poly CD during fabrication. This parameter is usually the most important skew parameter for a MOS process.

Figure 216 is an example of data that historical records produce.

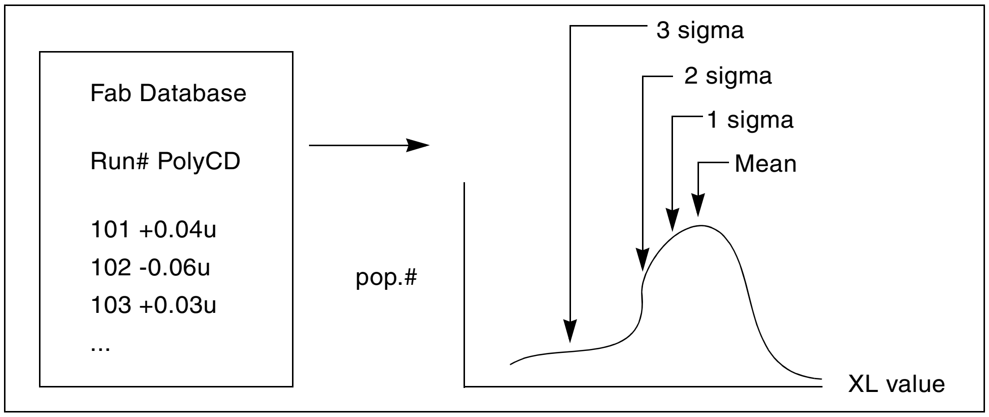


Figure 216 Historical Records for Skew Parameters in a MOS Process

Using Skew Parameters

Figure 217 shows how to create a worst-case corners library file for a CMOS process model. 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.

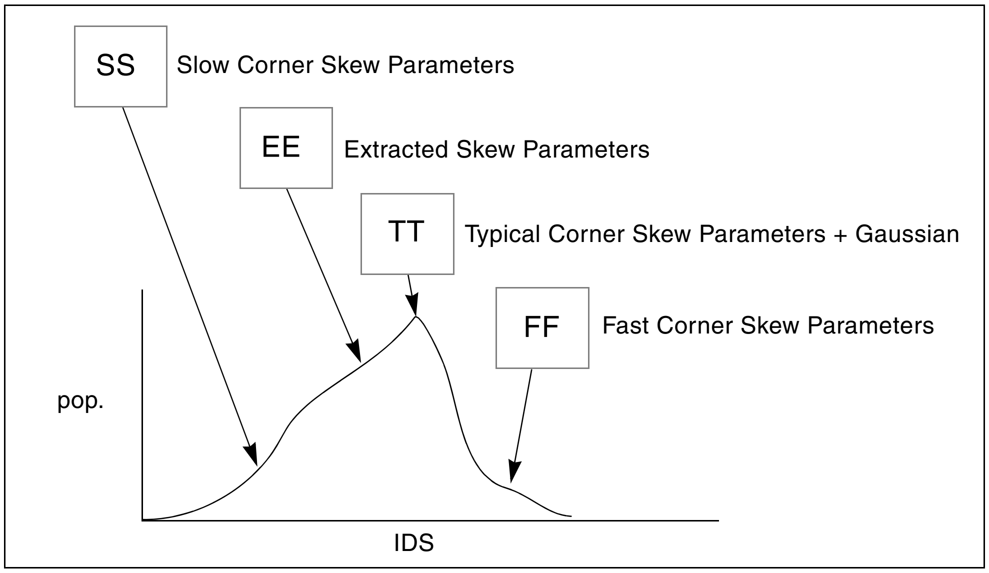

SS
Slow Corner Skew Parameters
EE
Extracted Skew Parameters
TT
Typical Corner Skew Parameters + Gaussian
FF
Fast Corner Skew Parameters
pop.
IDS

Figure 217 Worst Case Corners Library File for a CMOS Process Model

The `.LIB` (library) statement, and the `.INCLUDE` (include file) statement, access the models and skew. The library contains parameters that modify `.MODEL` statements. The following example of `.LIB` features both worst-case and statistical-distribution data by using model skew parameters. In statistical distribution, the median value is the default for all non-Monte Carlo analysis.

Example

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY DATE:3/4/91
$ PROCESS: 1.0U CMOS, FAB22, STATISTICS COLLECTED 3/90-2/91
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN

.PARAM
$ polysilicon Critical Dimensions
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
$ Active layer Critical Dimensions
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
$ Gate Oxide Critical Dimensions (200 angstrom +/- 10a at 1
$ sigma)
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'

$ Threshold voltage variation
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'


.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file
.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES

.PARAM TOX=230 XL=-0.18u DELVTON=-.15V DELVTOP= 0.15V
.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file

.ENDL FF
```

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

```
.MODEL NCH NMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTON . .
.MODEL PCH PMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTOP . .
```

Note:

The model keyname (left) equals the skew parameter (right). Model keys and skew parameters can use the same names.

Skew File Interface to Device Models

Skew parameters are model parameters for transistor models or passive components. A typical device model set includes:

- MOSFET models for all device sizes by using an automatic model selector.
- RC wire models for polysilicon, metal1, and metal2 layers in the drawn dimension. Models include temperature coefficients and fringe capacitance.
- Single-diode and distributed-diode models for N+, P+, and well (includes temperature, leakage, and capacitance based on the drawn dimension).
- BJT models for parasitic bipolar transistors. You can also use these for any special BJTs, such as a BiCMOS for ECL BJT process (includes current and capacitance as a function of temperature).
- Metal1 and metal2 transmission line models for long metal lines.
- Models must accept elements. Sizes are based on a drawn dimension. If you draw a cell at 2µ dimension and shrink it to 1µ, the physical size is 0.9µ. The effective electrical size is 0.8µ. Account for the four dimension levels:
  - drawn size
  - shrunken size
  - physical size
  - electrical size

Most simulator models scale directly from *drawn* to *electrical* size. HSPICE MOS models support all four size levels as in Figure 218.

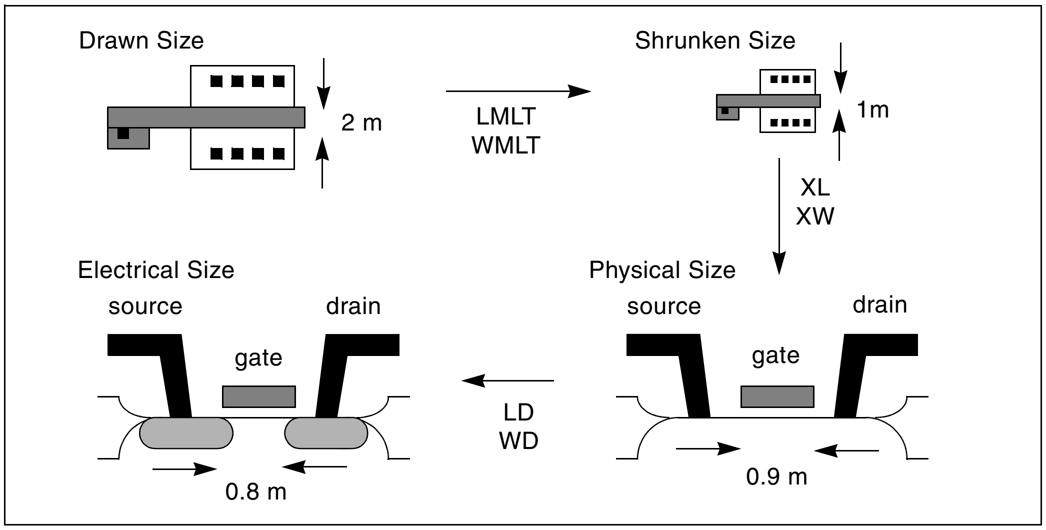


Figure 218 Device Model from Drawn to Electrical Size

Getting Started with Traditional Monte Carlo Simulations

The following is a high-level overview of HSPICE traditional Monte Carlo analysis. The sections that follow provide more in-depth information. For the most effective and efficient methods it is suggestion that your explore Variation Block-based Monte Carlo simulation (see Chapter 21, Monte Carlo Analysis Using the Variation Block Flow).

The basic premise of a Monte Carlo analysis is that you are going to parameterize one or more circuit variables, vary those values by a randomized amount from the norm and run HSPICE a pre-determined number of times. Each run is called a sweep and will generate tabular or plot data as specified by the user. Measurements are also typically used to look at circuit operating conditions from run to run.

You can randomize anything that can be set with a parameter, or variable. Examples include things as diverse as a simple resistor value, a model parameter for a MOSFET, or the length of a transmission line.

Values can be varied using three basic statistical variations; uniform, limit and Gaussian. Using those methods, you choose the nominal value and the absolute or relative variation. You can optionally supply the standard deviation and a multiplier.

Basic Syntax

The basic syntax of a Monte Carlo analysis includes three elements:

1. Defining a parameter with one of the distribution keywords
2. Using the parameter in your netlist as the value for an element or model parameter
3. Including the keywords SWEEP and MONTE keyword in the analysis statement

Consider the following example. In this simple RC charging circuit, the value of r1 has a nominal value of 1K and is varied by 400 ohms for 10 iterations.

RC charging circuit

```
.option post probe
*define a parameter called "resval" with an absolute, uniform
distribution
.param resval=aunif(1000,400)
 vsrc_one 1 0 5v
 r_one 1 2 resval
 c_one 2 0 1u
.ic 2=0
*specify 10 Monte Carlo iterations
.tran 1e-5 5e-3 sweep monte=10
*measure to find when 1 time constant (.632*vdd) occurs
.meas tran tc when v(2)='.632*5'
*create plots of  the charging curve and resistor values
.probe v(2) par(resval)
.end
```

The resulting waveforms are called “multi-member”. Plotting the one signal will display the curves from all the runs.

Local and Global Parameter Variation

A common source of confusion is local and global parameter variation. The key is that each time you use a parameter, it gets assigned a new random value. Take the following examples:

```
.param resval=aunif(1000,400)
r_one 1 2 resval
r_two 2 3 resval
r_three 3 4 resval
```

In this case, all three resistors will get unique, random values. If you want to set a group of components to the same random value, assign an intermediate parameter first:

```
.param resval=aunif(1000,400)
.param my_resval=resval
r_one 1 2 my_resval
r_two 2 3 my_resval
r_three 3 4 my_resval
```

In the second example, the assignment of a random value is only done once, then used three times. The exception to this rule is for model parameters.

Exception for Model Parameters

Since a model definition is only done once, the behavior described above would assign the same parameter value to all devices referencing that model. To overcome this, `.OPTION MODMONTE` lets the user decide if all instances of a device should get the same or unique model parameters.

Starting Values and Seeds

Another source of confusion is the starting value. If you run the same Monte Carlo simulation twice, the results will be identical because HSPICE always uses the same “seed” value for the first run. If it randomized the seed by default, it would be difficult, if not impossible, to tell whether changes you made to the circuit and topology were the result of your changes or the new random values. You can specify a seed or tell HSPICE to pick a random seed with `.OPTION SEED` if that behavior is desired.

Other Monte Carlo Control Options

- `.OPTION MONTECON`—some random parameter assignments can cause HSPICE not to converge. This parameter is used to decide whether to terminate a simulation or move to the next run if convergence fails.
- `.OPTION RANDGEN`—use this option to specify the type of random number generator used.
- `.OPTION MCBRIEF`—controls how HSPICE outputs Monte Carlo parameters.

Traditional Monte Carlo Analysis

Monte Carlo analysis uses a random number generator to create the following types of functions.

- Gaussian parameter distribution
  - Relative variation—variation is a ratio of the average.
  - Absolute variation—adds variation to the average.

  - Bimodal–multiplies distribution to statistically reduce nominal parameters.
- Uniform parameter distribution
  - Relative variation—variation is a ratio of the average.
  - Absolute variation—adds variation to the average.
  - Bimodal–multiplies distribution to statistically reduce nominal parameters.
- Random limit parameter distribution
  - Absolute variation—adds variation to the average.
  - Monte Carlo analysis randomly selects the *min* or *max* variation.

The value of the `MONTE` analysis keyword determines how many times to perform operating point, DC sweep, AC sweep, or transient analysis.

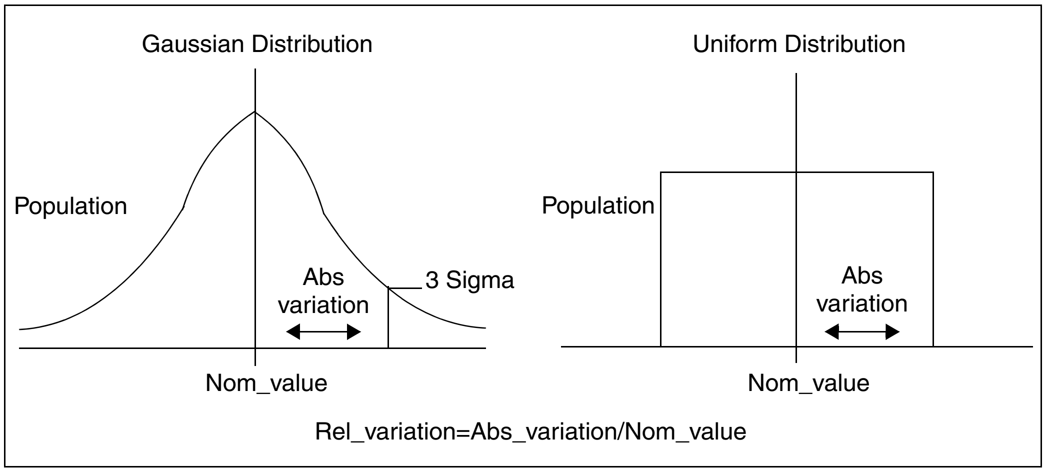


Figure 219 Monte Carlo Distribution

Monte Carlo Setup

To set up a Monte Carlo analysis, use the following HSPICE statements:

- `.PARAM` statement—sets a model or element parameter to a Gaussian, Uniform, or Limit function distribution.
- `.DC`, `.AC`, or `.TRAN` analysis—enables `MONTE`.

- `.MEASURE` statement—calculates the output mean, variance, sigma, and standard deviation.
- `.MODEL` statement—sets model parameters to a Gaussian, Uniform, or Limit function distribution.

Select the type of analysis to run, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating Point

```
.DC MONTE=<firstrun=num1>
```

-or-

```
.DC MONTE=list <(> <num1:num2> <num3> <num5:num6> <num7> <)>
```

DC Sweep

```
.DC vin 1 5 0.25 sweep MONTE=val <firstrun=num1>
```

-or-

```
.DC vin 1 5 0.25 sweep MONTE=list<(> <num1:num2> <num3>
+ <num5:num6> <num7> <)>
```

AC Sweep

```
.AC dec 10 100 1meg sweep MONTE=val <firstrun=num1>
```

-or-

```
.AC dec 10 100 1meg sweep MONTE=list<(> <num1:num2>
+ <num3> <num5:num6> <num7> <)>
```

TRAN Sweep

```
.TRAN 1n 10n sweep MONTE=val <firstrun=num1>
```

-or-

```
.TRAN 1n 10n sweep MONTE=list<(> <num1:num2> <num3>
+ <num5:num6> <num7> <)>
```

The *val* value specifies the number of Monte Carlo iterations to perform. A reasonable number is 30. The statistical significance of 30 iterations is quite high. If the circuit operates correctly for all 30 iterations, there is a 99% probability that over 80% of all possible component values operate correctly. The relative error of a quantity, determined through Monte Carlo analysis, is proportional to val$^{-1/2}$.

The *firstrun* values specify the desired number of iterations. HSPICE runs from num1 to num1+val-1. The number after *firstrun* can be a parameter. You can write only one number after *list*. The colon represents “from ... to ...". Specifying only one number makes HSPICE runs only a the one specified point.

Example 1

In this example, HSPICE runs from the 90th to 99th Monte Carlo iterations:

```
.tran 1n 10 sweep monte=10 firstrun=90
```

You can write more than one number after *list*. The colon represents “from ... to ...". Specifying only one number makes HSPICE run only at that single point.

Example 1

In this example, HSPICE begins running at the 10th iteration, then continues from the 20th to the 30th, at the 40th, and finally from the 46th to 72nd Monte Carlo iteration. The numbers after list can not be parameter.

```
.tran 1n 10n sweep monte=list(10 20:30 40 46:72)
```

Monte Carlo Output

- `.MEASURE` statements are the most convenient way to summarize the results.
- `.PRINT` statements generate tabular results, and print the values of all Monte Carlo parameters.
- `.MCBRIEF` determines the output types of the random parameters during Monte Carlo analysis to improve output performance.
- If one iteration is out of specification, you can obtain the component values from the tabular listing. A detailed resimulation of that iteration might help identify the problem.
- AvanWaves superimposes all iterations as a single plot so you can analyze each iteration individually.

.PARAM Distribution Function

This section describes how to use assign a `.PARAM` parameter in Monte Carlo analysis. For a general description of the `.PARAM` statement, see the .PARAM command in the *HSPICE and HSPICE RF Command Reference*.

You can assign a `.PARAM` parameter to the keywords of elements and models, and assign a distribution function to each `.PARAM` parameter. HSPICE recalculates the distribution function each time that and element or model keyword uses a parameter. When you use this feature, Monte Carlo analysis can use a parameterized schematic netlist without additional modifications.

Syntax

```
.PARAM xx=UNIF(nominal_val, rel_variation
+ <, multiplier>)

.PARAM xx=AUNIF(nominal_val, abs_variation <,
+ multiplier>)

.PARAM xx=GAUSS(nominal_val, rel_variation, sigma <,
+ multiplier>)

.PARAM xx=AGAUSS(nominal_val, abs_variation, sigma <,
+ multiplier>)

.PARAM xx=LIMIT(nominal_val, abs_variation)
```

Argument	Description
xx	Distribution function calculates the value of this parameter.
UNIF	Uniform distribution function by using relative variation.
AUNIF	Uniform distribution function by using absolute variation.
GAUSS	Gaussian distribution function by using relative variation.
AGAUSS	Gaussian distribution function by using absolute variation
LIMIT	Random-limit distribution function by using absolute variation. Adds +/- *abs_variation* to *nominal_val* based on whether the random outcome of a -1 to 1 distribution is greater than or less than 0.
nominal_val	Nominal value in Monte Carlo analysis and default value in all other analyses.
abs_variation	AUNIF and AGAUSS vary the nominal_val by +/- *abs_variation*.

Argument	Description
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.

Example 1

In this example, each R has an unique variation.

```
.param mc_var=agauss(0,1,3)   $ +/-1 absolute swing or
                              $ +/-100% relative swing
.param val='1000*(1+mc_var)'
v_vin vin 0 dc=1 ac=.1
r1 vin 0  '1000*(1+mc_var)'
r2 vin 0  '1000*(1+mc_var)'
```

Example 2

In this example, each R has an identical variation.

```
.param mc_var=agauss(0,1,3)   $ +/- 20% swing
.param val='1+mc_var'
v_vin vin 0 dc=1 ac=.1
r1 vin 0  '1000*val'
r2 vin 0  '1000*val'
```

Example 3

In this example, local variations to an instance parameter are applied by assigning randomly-generated variations directly to each instance parameter. Each resistor r1 through r3 receives randomly different resistance values during each Monte Carlo run.

```
.param r_local=agauss(...)
r1 1 2 r=r_local
r2 3 4 r=r_local
r3 5 6 r=r_local
```

Example 4

In this example, global variations to an instance parameter are applied by assigning the variation to an intermediate parameter before assigning it to each instance parameter. Each resistor r1 through r3 receives the same random resistance value during each Monte Carlo run.

```
.param r_random=agauss(...)
.param r_global=r_random
r1 1 2 r=r_global
r2 3 4 r=r_global
r3 5 6 r=r_global
```

Monte Carlo Parameter Distribution

Each time you use a parameter, Monte Carlo calculates a new random variable.

- If you do not specify a Monte Carlo distribution, then HSPICE assumes the nominal value.
- If you specify a Monte Carlo distribution for only one analysis, HSPICE uses the nominal value for all other analyses.

You can assign a Monte Carlo distribution to all elements that share a common model. The actual element value varies according to the element distribution. If you assign a Monte Carlo distribution to a model keyword, then all elements that share the model, use the same keyword value. You can use this feature to create double element and model distributions.

For example, the MOSFET channel length varies from transistor to transistor by a small amount that corresponds to the die distribution. The die distribution is responsible for offset voltages in operational amplifiers, and for the tendency of flip-flops to settle into random states. However, all transistors on a die site vary according to the wafer or fabrication run distribution. This value is much larger than the die distribution, but affects all transistors the same way. You can specify the wafer distribution in the MOSFET model to set the speed and power dissipation characteristics.

Monte Carlo Examples

Gaussian, Uniform, and Limit Functions

You can find the sample netlist for this example in the following directory:
$installdir/demo/hspice/apps/mont1.sp

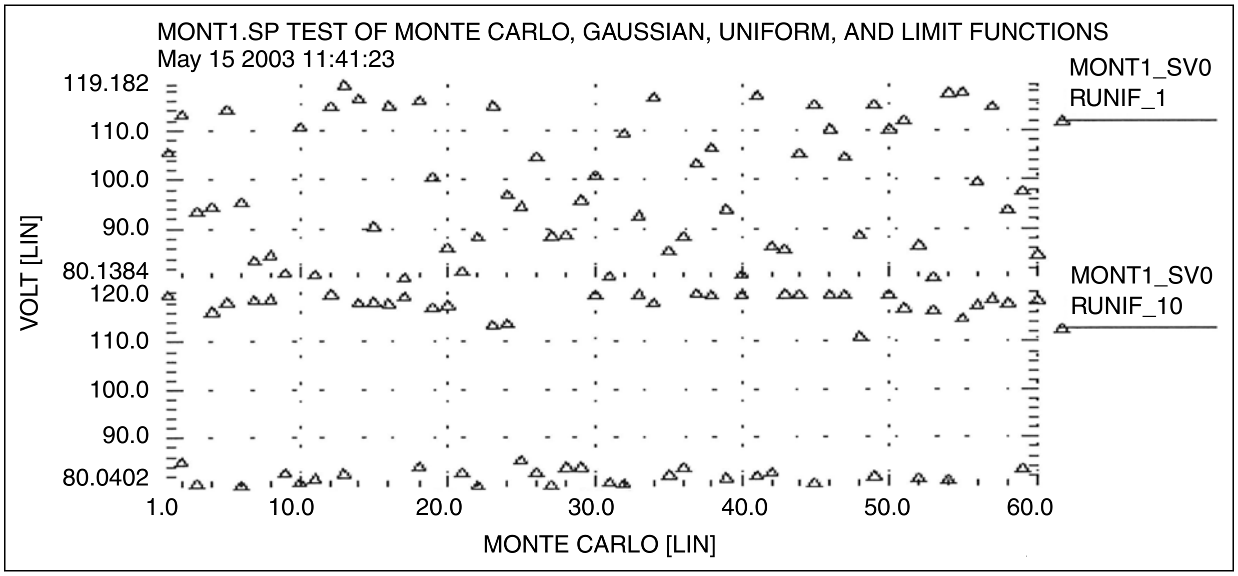


Figure 220 Uniform Functions

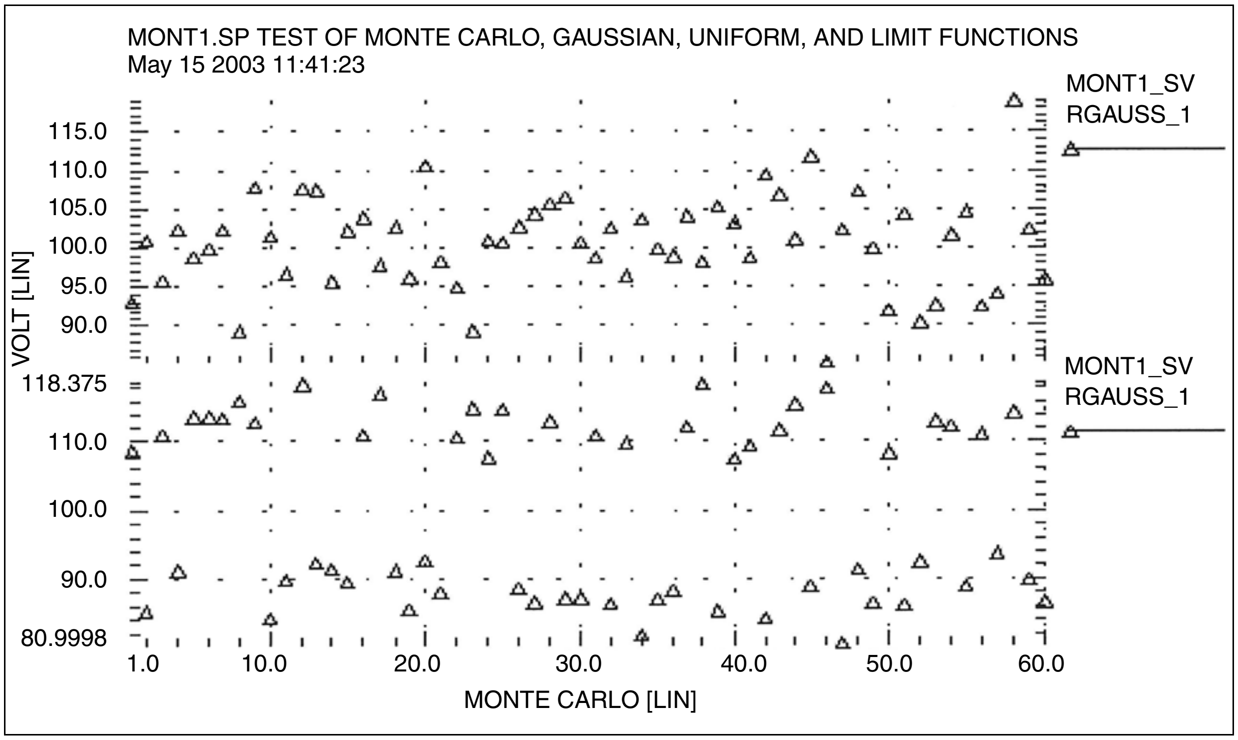


Figure 221 Gaussian Functions

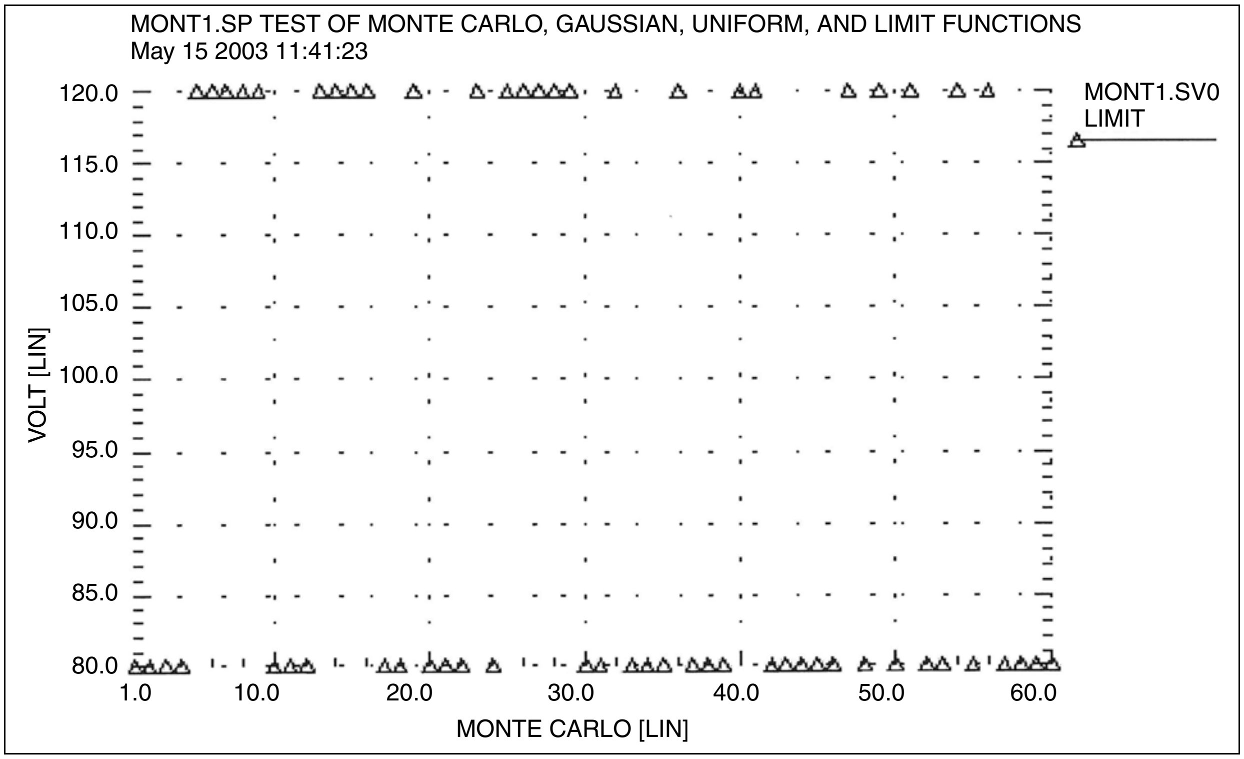


Figure 222 Limit Functions

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.

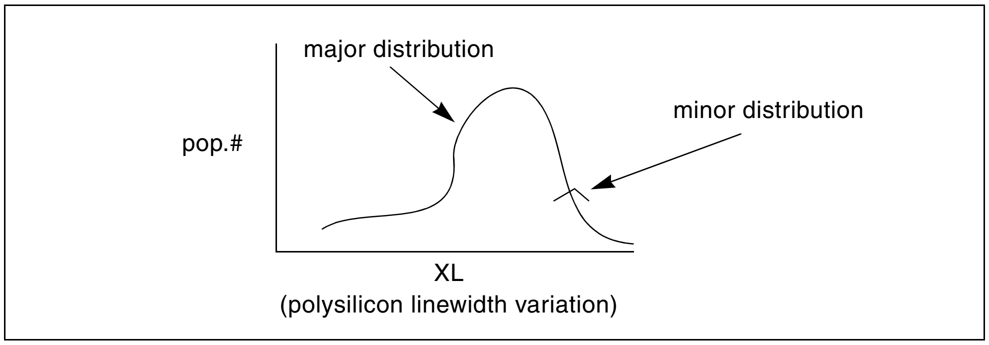


Figure 223 Major and Minor Distribution of Manufacturing Variations

The following example 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.

You can find the sample netlist for this example in the following directory: $installdir/demo/hspice/apps/mondc_a.sp

- 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.

You can find the sample netlist for this example in the following directory: $installdir/demo/hspice/apps/rc_monte.sp

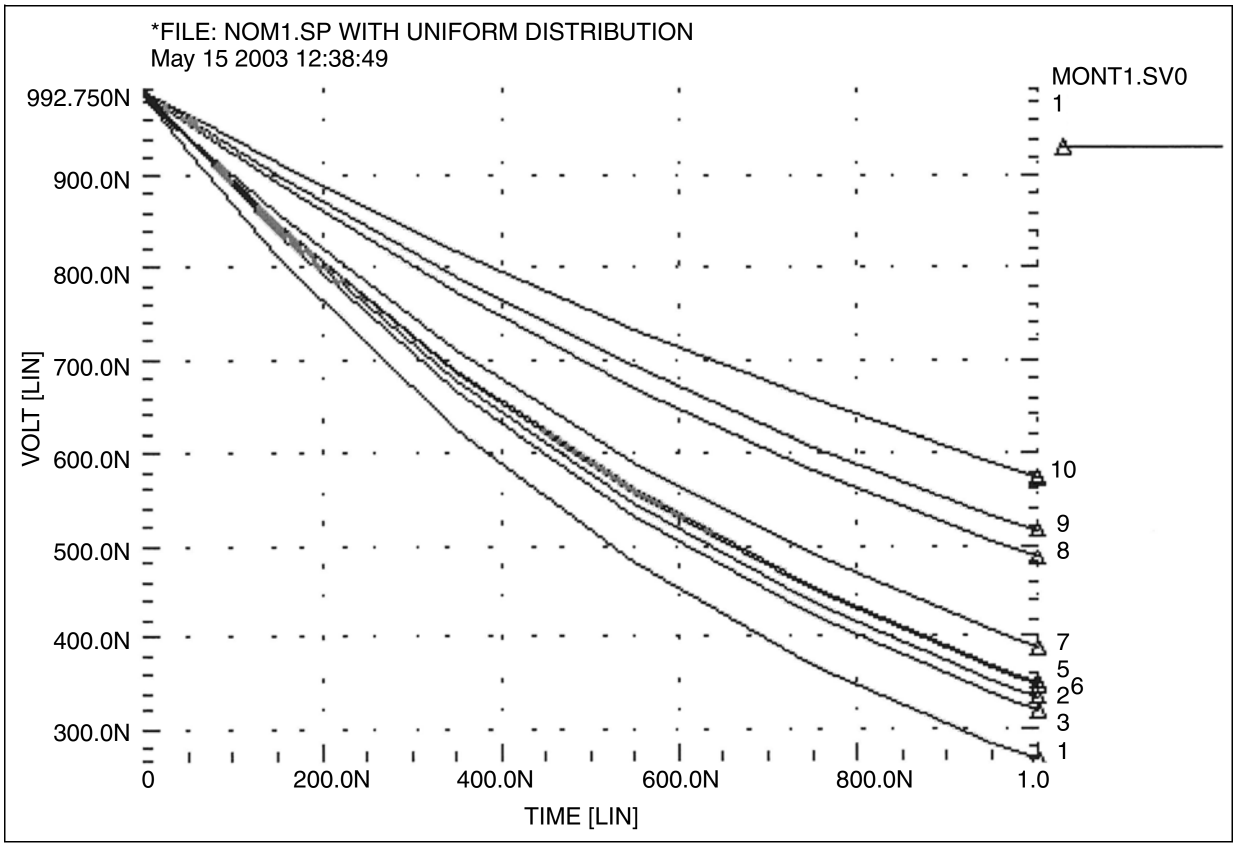


Figure 224 Monte Carlo Analysis of RC Time Constant

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.

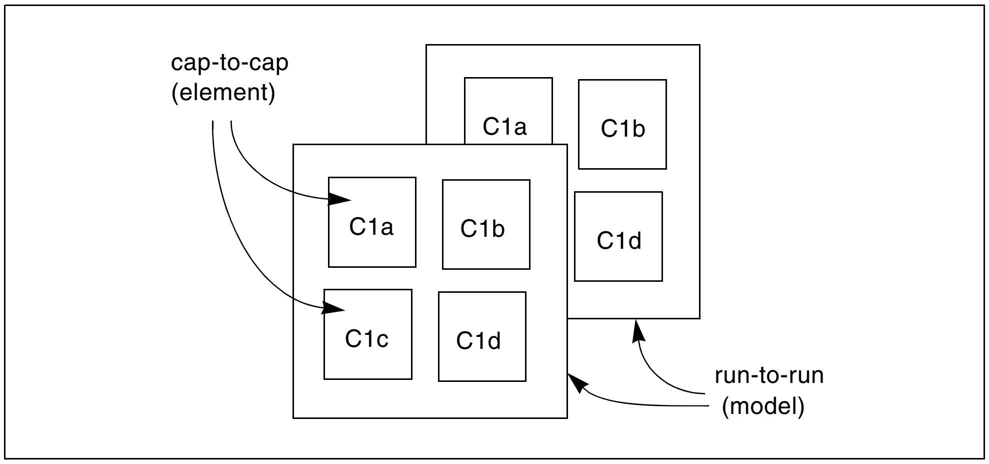


Figure 225 Monte Carlo Distribution

You can approach this problem from physical or electrical levels.

- The physical level relies on physical distributions, such as oxide thickness and polysilicon line width control.
- The electrical level relies on actual capacitor measurements.

Physical Approach:

1. Since oxide thickness control is excellent for small areas on a single wafer, you can use a local variation in polysilicon to control the variation in capacitance for adjacent cells.
2. Next, define a local poly line-width variation and a global (model-level) poly line-width variation. In this example:
   - The local polysilicon line width control for a line 10 m wide, manufactured with process A, is ±0.02 m for a 1-sigma distribution.
   - The global (model level) polysilicon line-width control is much wider; use 0.1 m for this example.
3. The global oxide thickness is 200 angstroms with a ±5 angstrom variation at 1 sigma.
4. The cap element is square with local poly variation in both directions.
5. The cap model has two distributions:
   - poly line-width distribution
   - oxide thickness distribution.

The effective length is:

```
Leff=Ldrawn - 2 · DEL
```

The model poly distribution is half the physical per-side values:

```
C1a 1 0 CMOD W=ELPOLY L=ELPOLY
C1b 1 0 CMOD W=ELPOLY L=ELPOLY
C1C 1 0 CMOD W=ELPOLY L=ELPOLY
C1D 1 0 CMOD W=ELPOLY L=ELPOLY
$ 10U POLYWIDTH,0.05U=1SIGMA
$ CAP MODEL USES 2*MODPOLY  .05u= 1 sigma
$ 5angstrom oxide thickness AT 1SIGMA
.PARAM ELPOLY=AGAUSS(10U,0.02U,1)
+ MODPOLY=AGAUSS(0,.05U,1)
+ POLYCAP=AGAUSS(200e-10,5e-10,1)
.MODEL CMOD C THICK=POLYCAP DEL=MODPOLY
```

Electrical Approach:

The electrical approach assumes no physical interpretation, but requires a local (element) distribution and a global (model) distribution. In this example:

- You can match the capacitors to ±1% for the 2-sigma population.
- The process can maintain a ±10% variation from run to run for a 2-sigma distribution.

```
C1a 1 0 CMOD SCALE=ELCAP
C1b 1 0 CMOD SCALE=ELCAP
C1C 1 0 CMOD SCALE=ELCAP
C1D 1 0 CMOD SCALE=ELCAP
.PARAM ELCAP=Gauss(1,.01,2) $ 1% at 2 sigma
+ MODCAP=Gauss(.25p,.1,2) $10% at 2 sigma
.MODEL CMOD C CAP=MODCAP
```

Worst Case and Monte Carlo Sweep Example

The following example measures the delay and the power consumption of two inverters. Additional inverters buffer the input and load the output.

This netlist contains commands for two sets of transient analysis: parameter sweep from -3 to +3-sigma, and a Monte Carlo analysis. It creates one set of output files (mt0 and tr0) for the sigma sweep, and one set (mt1 and tr1) for Monte Carlo.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma, use measure output
.param vref=2.5 sigma=0
.global 1
vcc  1 0  5.0
vin  in 0 pwl 0,0 0.2n,5
x1 in 2 inv
x2 2 3 inv
x3 3 out inv
x4 out 4 inv
.macro inv in out
  mn out in 0 0 nch w=10u l=1u
  mp out in 1 1 pch w=10u l=1u
.eom
.param mult1=1
+ polycd=agauss(0,0.06u,1)   xl='polycd-sigma*0.06u'
+ nactcd=agauss(0,0.3u,1)  xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1)  xwp='pactcd+sigma*0.3u'
+ toxcd=agauss(200,10,1)   tox='toxcd-sigma*10'
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtoncd+sigma*0.05'
+ rshncd=agauss(50,8,1)   rshn='rshncd-sigma*8'
+ rshpcd=agauss(150,20,1)   rshp='rshpcd-sigma*20'
* level=28 example model
.model nch nmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwn tox=tox delvto=delvton rsh=rshn
...
.model pch pmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp js=3e-04 jsw=9e-10
...
* transient with sweep
.tran 20p 1.0n   sweep sigma -3 3 .5
.meas s_delay trig v(2) val=vref fall=1
+           targ v(out) val=vref fall=1
.meas s_power rms power
* transient with Monte Carlo
.tran 20p 1.0n   sweep monte=100
.meas m_delay trig v(2) val=vref fall=1
+           targ v(out) val=vref fall=1
.meas m_power rms power
.probe tran v(in) v(1) v(2) v(3) v(4)
.end
```

Transient Sigma Sweep Results

The plot in Figure 226 shows the family of transient analysis curves for the transient sweep of the sigma parameter from -3 to +3 from the file inv.tr0. In the sweep, HSPICE uses the values of sigma to update the skew parameters, which in turn modify the actual NMOS and PMOS models.

Operating-Point Results in Transient Analysis

If you want to get OP results after every Monte Carlo simulation in transient analysis, you can add the option `opfile` to the netlist. OP results will all output to the file *.dp0.

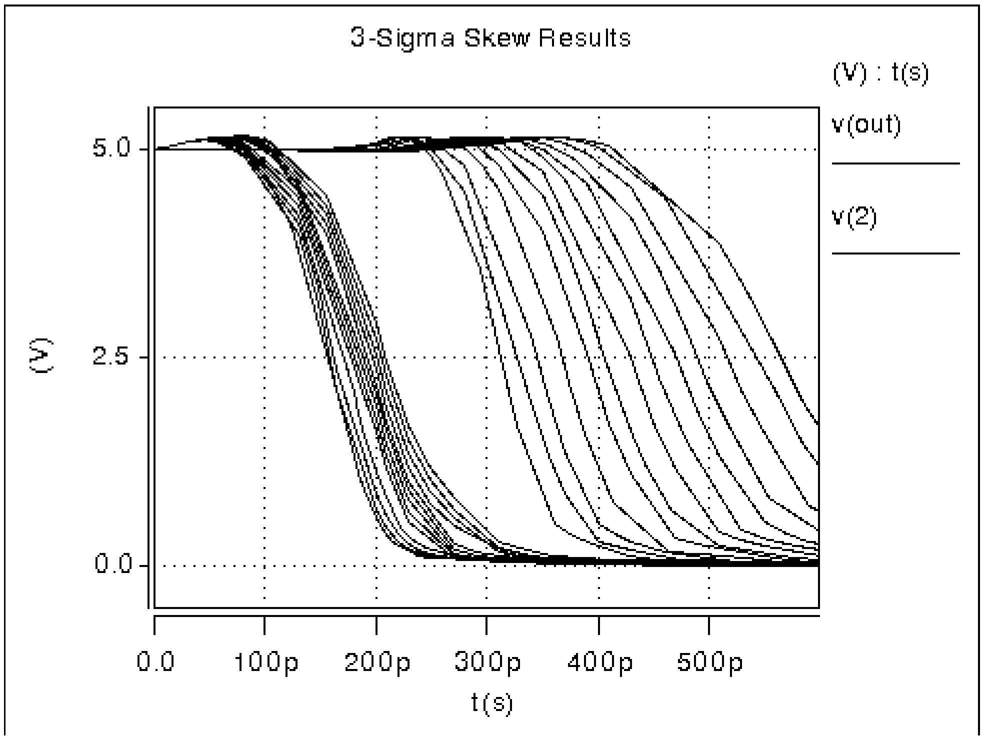


Figure 226 Sweep of Skew Parameters from -3 Sigma to +3 Sigma

To view the measured results, plot the inv.mt0 output file. The plot in Figure 227 shows the measured pair delay and the total dissipative power, as a function of the parameter sigma. To get the specific operating point information of each Monte Carlo run, use `opfile=1`.

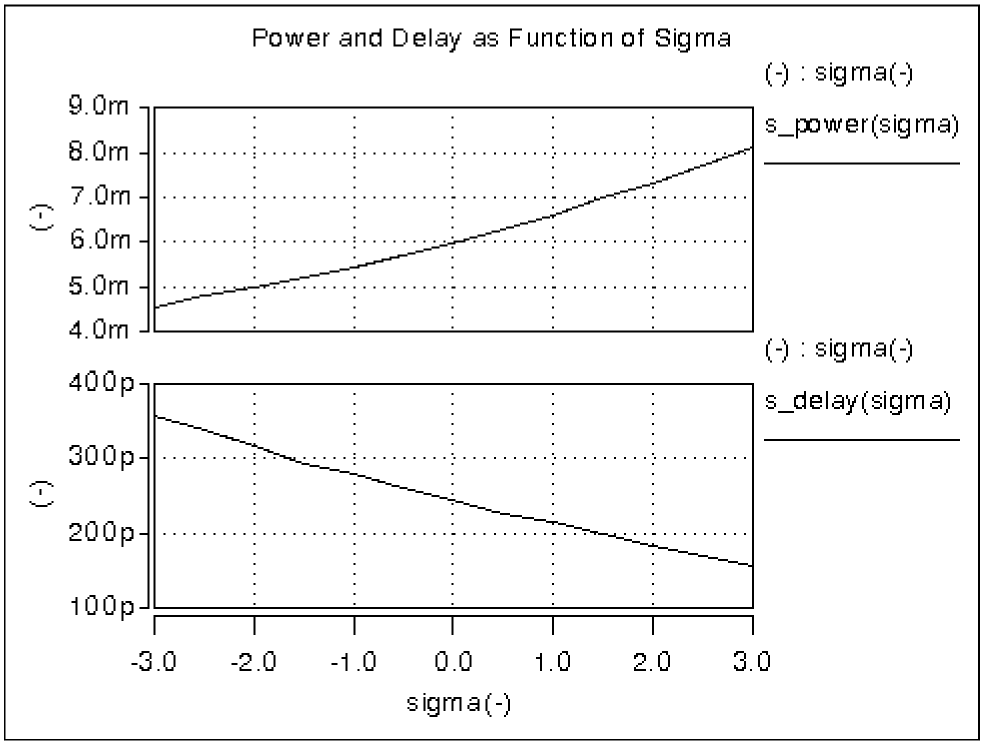


Figure 227 Sweep MOS Inverter, Pair Delay and Power: -3 Sigma to 3 Sigma

Monte Carlo Results

This section describes the output of the Monte Carlo analysis in HSPICE. The plot in Figure 228 shows that the relationship between `TOX` against `XL` (polysilicon width=transistor length)) is completely random, as set up in the input file.

To generate this plot in CosmosScope:

1. Read in the file inv.mt1.
2. Open the Calculator, select TOX (left mouse button), transfer to calculator (middle mouse button), and then select and transfer XL.
3. On the WAVE pulldown in the calculator, select f(x), and then click the plot icon.
4. Using the right mouse button on the plotted waveform, select Attributes to change from the line plot to symbols.

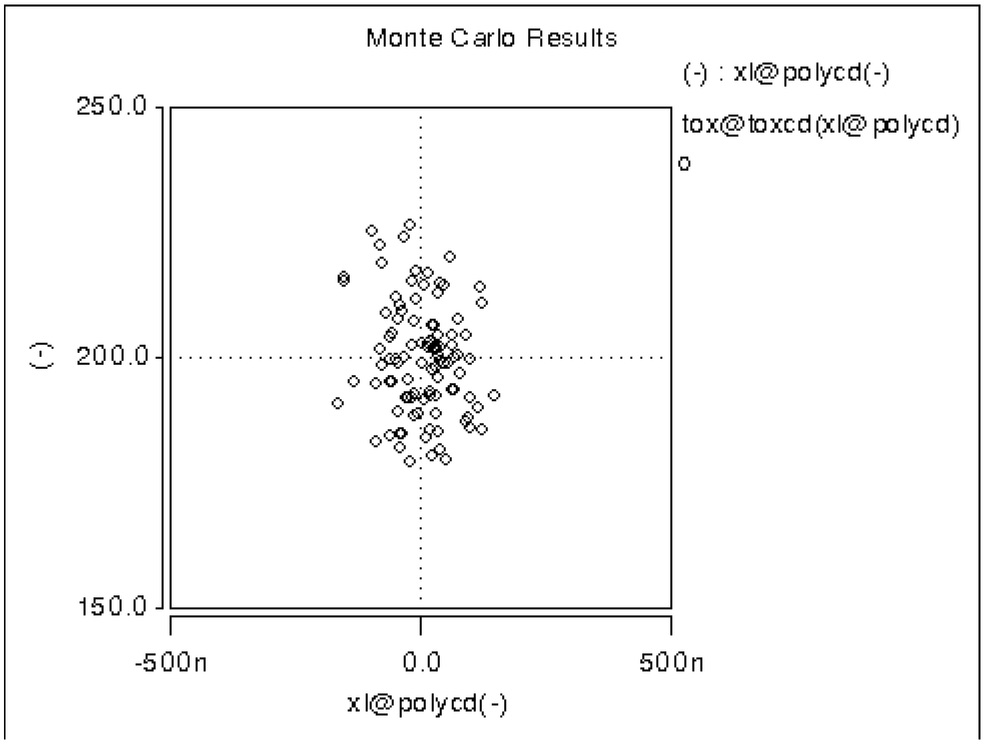


Figure 228 Scatter Plot, XL and TOX

The next graph (see Figure 229) is a standard scatter plot showing the measured delay for the inverter pair against the Monte Carlo index number.

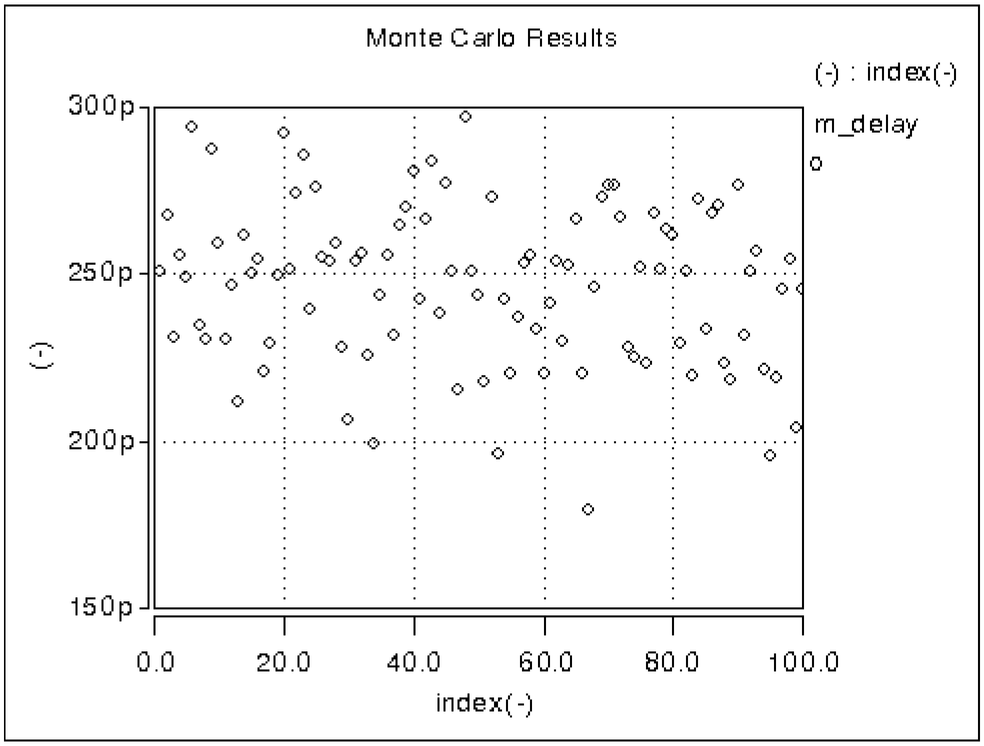


Figure 229 Scatter Plot of Inverter Pair Delay

If a particular result looks interesting; for example, if the simulation 68 (monte carlo index=68) produces the smallest delay, then you can obtain the Monte Carlo parameters for that simulation.

```
*** monte carlo  index =    68 ***
   MONTE CARLO PARAMETER DEFINITIONS
 polycd  xl              = -1.6245E-07
 nactcd  xwn             =  3.4997E-08
 pactcd  xwp             =  3.6255E-08
 toxcd   tox             =   191.0
 vtoncd  delvton         = -2.2821E-02
         delvtop         =  4.1776E-02
 vtopcd
 rshncd  rshn            =   45.16
 rshpcd  rshp            =   166.2
 m_delay=  1.7929E-10  targ=  3.4539E-10   trig=  1.6610E-10
 m_power=  6.6384E-03  from=  0.0000E+00     to=  1.0000E-09
```

In the preceding listing, the m_delay value of 1.79e-10 seconds is the fastest pair delay. You can also examine the Monte Carlo parameters that produced this result.

The information on shortest delay and so forth is also available from the statistics section at the end of the output listing. While this information is useful to determine whether the circuit meets specification, it is often desirable to understand the relationship of the parameters to circuit performance. Plotting the results against the Monte Carlo index number does not help for this purpose. You need to generate plots that display a Monte Carlo result as a function of a parameter. For example, Figure 230 shows the inverter pair delay to channel as a function of poly width, which relates directly to device length.

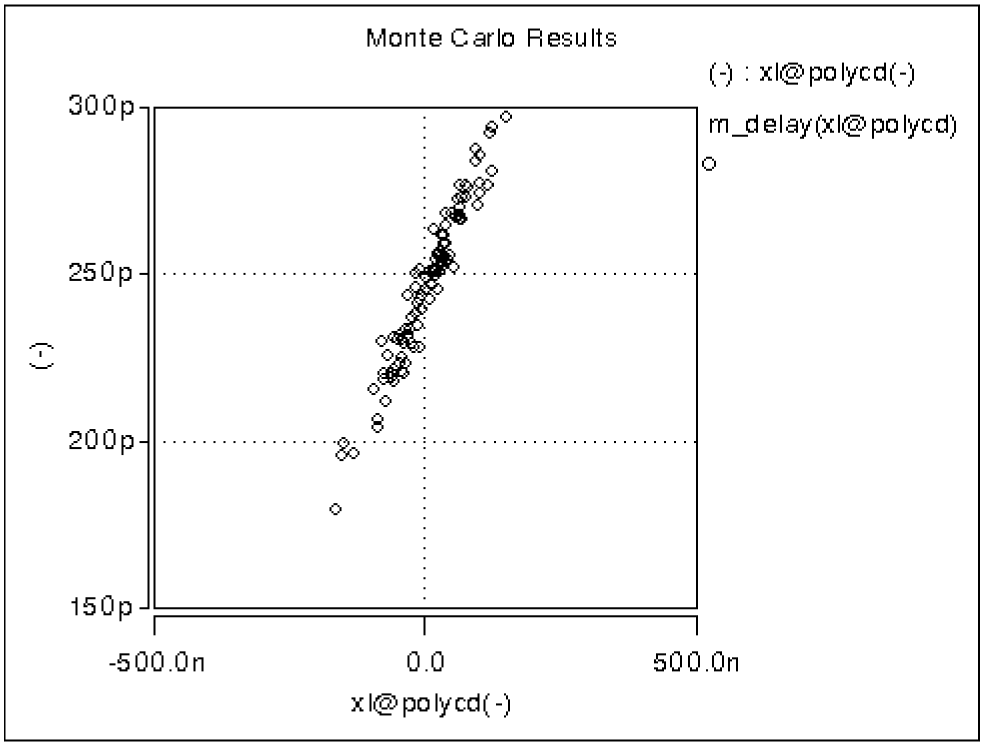


Figure 230 Delay as a function of Poly width (XL)

Figure 231 shows the pair delay against the `TOX` parameter. The scatter plot shows no obvious dependence, which means that the effect of `TOX` is much smaller than `XL`. To explore this in more detail, set the `XL` skew parameter to a constant and run a simulation.

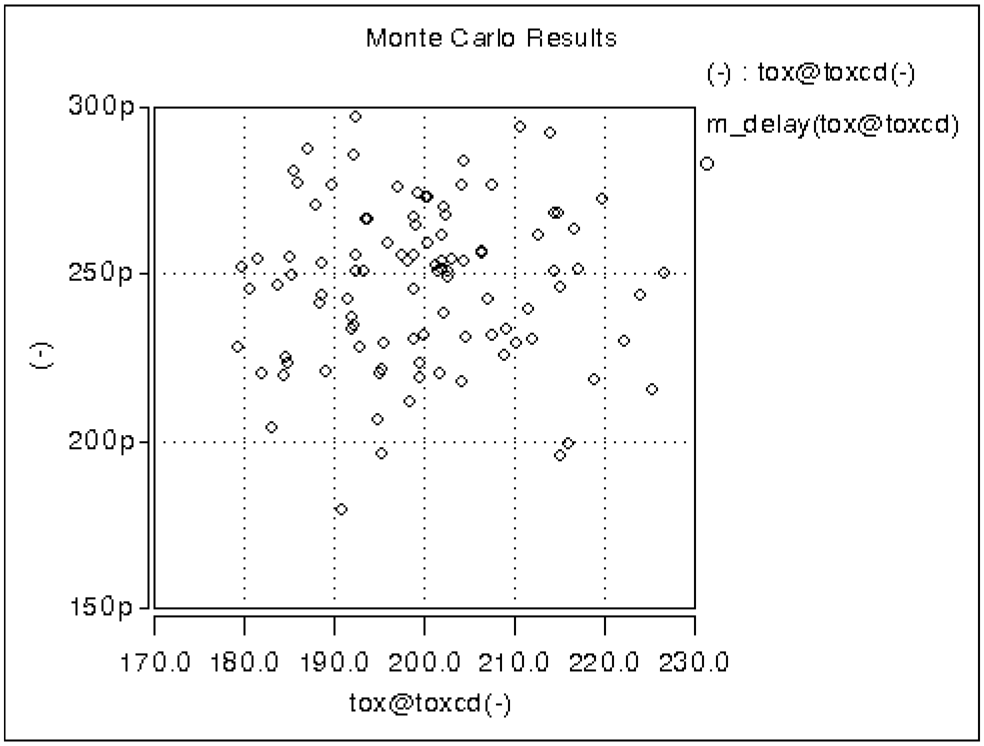


Figure 231   Sensitivity of Delay with TOX

The plot in Figure 232 overlays the skew result with the ones from Monte Carlo. The skew simulation traverses the design space with all parameters changing in parallel and then produces a relationship between power and delay, which shows as a single line. Monte Carlo exercises a variety of independent parameter combinations, and shows that there is no simple relationship between the two results. Since the distributions were defined as Gaussian in the netlist, parameter values close to the nominal are more often exercised than the ones far away. With the relatively small number of samples, the chance of hitting a combination at the extremes is very small. In other words, designing for 3-sigma extreme for every parameter is probably not a good solution from the point of view of economy.

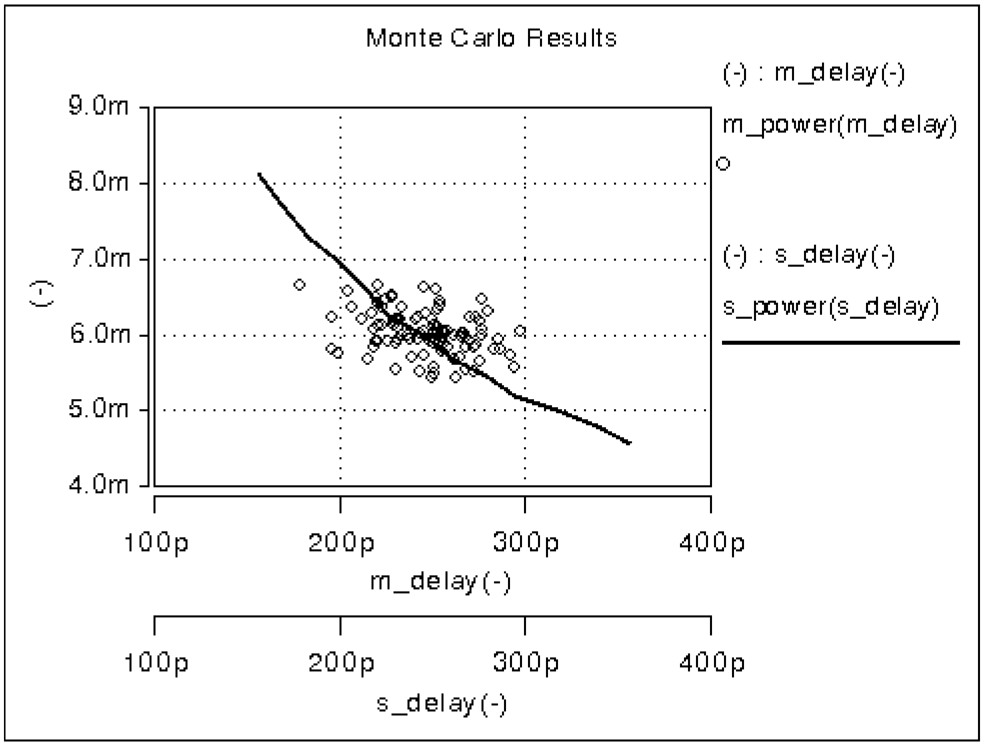


Figure 232 Superimposing Sigma Sweep Over Monte Carlo

Figure 233 superimposes the required part grades for product sales onto the Monte Carlo plot. This example uses a 250 ps delay and 6.0 mW power dissipation to determine the four binning grades.

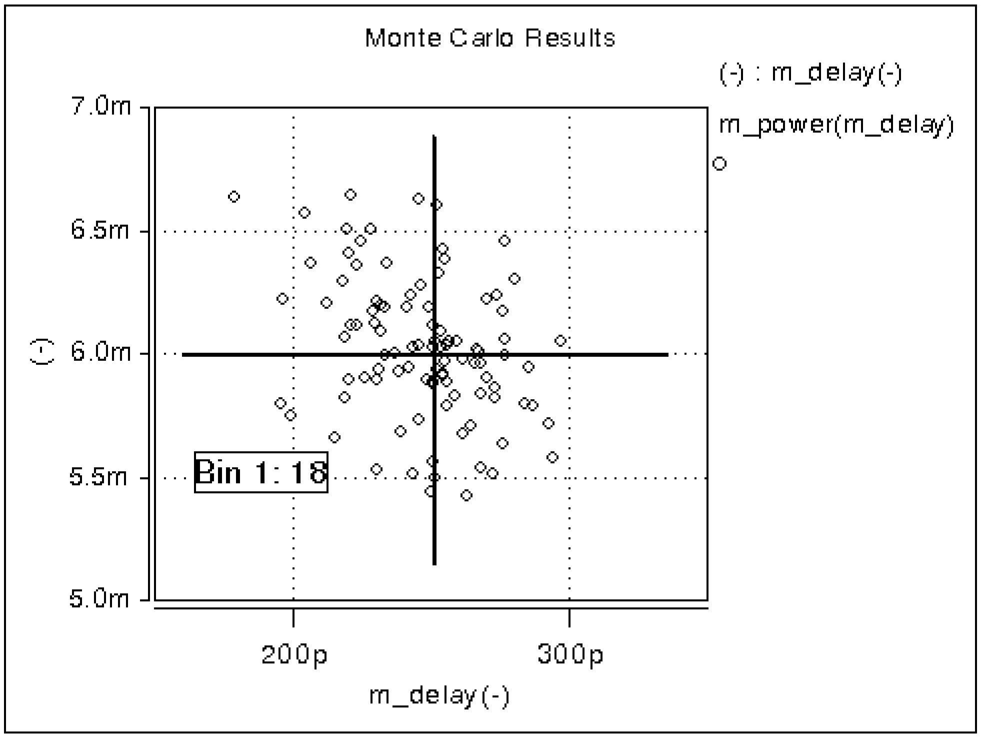


Figure 233 Speed/Power Yield Estimation

Sorting the results from inv.mt1 yields:

- Bin1 - 18%
- Bin2 - 30%
- Bin3 - 31%
- Bin4 - 21%

If this circuit is representative of the entire chip, then the present yield should be 18% for the premium Bin 1 parts, assuming variations in process parameters as specified in the netlist. Of course this example only shows the principle on how to analyze the Monte Carlo results; there is no market for a device with two of these inverters.

Simulating the Effects of Global and Local Variations with Monte Carlo

Monte Carlo analysis is dependent on a method to describe variability. Four different approaches are available in HSPICE:

1. Specify distributions on parameters and apply these to instance parameters.
2. Specify distributions on parameters and apply these to model parameters.
3. Specify distributions on model parameters using `DEV`/`LOT` construct.
4. Specify distributions on model parameters in a Variation Block.

While the first three methods are still supported in HSPICE, the method based on the Variation Block is preferred due to its improvements and future development. The Variation Block is described in Chapter 20, Analyzing Variability and Using the Variation Block, and Monte Carlo analysis controlled by the Variation Block is described in Chapter 21, Monte Carlo Analysis Using the Variation Block Flow.

In the following sections, the first three methods are described. The description relies on test cases, which can be found in the tar file monte_test.tar in directory $<installdir>/demo/hspice/variability.

Variations Specified on Geometrical Instance Parameters

This method consists of defining parameters with variation using the distribution functions `UNIF`, `AUINF`, `GAUSS`, `AGAUSS`, and `LIMIT`. These parameters are then used to generate dependent parameters or in the place of instance parameters. In a Monte Carlo simulation, at the beginning of each sample, new random values are calculated for these parameters. For each reference, a new random value is generated; however, no new value is generated for a derived parameter. Therefore, it is possible to apply independent variations to parameters of different devices, as well as the same variation to parameters of a group of devices. Parameters that describe distributions can be used in expressions, thus it is possible to create combinations of variations (correlations).

These concepts are best explained with circuit examples. In the three following examples, variation is defined on the width of a physical resistor, which has a model. If this device was a polysilicon resistor for example, then the variations describe essentially the effects of photoresist exposure and etching on the width of the poly layer.

- test1.sp has a distribution parameter defined called globw. A parameter called globwidth is assigned the value of globw. The parameter globwidth is assigned a different random value for each Monte Carlo sample. The parameter globwidth is used to define the width of the physical resistors r1, r2, r3, and r4, with model “resistor”. Since parameter globwidth does not have its own distribution defined, but rather gets its value from the parameter globw, the value for globwidth is the same wherever it is used; thus the resistors have the same width for each Monte Carlo sample, and therefore the same resistance. When plotting the simulation results v1, v2, v3, and v4 from the `.meas` file, the waveforms overlay perfectly. This type of setup is typically used to model global variations, which means variations that affect all devices the same way.
- test2.sp has a distribution parameter defined called locwidth. This parameter is used to define the width of the physical resistors r1, r2, r3, and r4, with model “resistor”. Since the parameter has its own distribution defined, its value will be different for each reference, and of course for each Monte Carlo sample. Therefore, the resistors will always have different values, and the voltages will be different. This type of setup is typically used to model local variations, which means variations that affect devices in a different way.
- test3.sp has two kinds of distributions defined: globw/globwidth as in the first example, and locwidth as in the second example. The sum of the two is used to define the width of the resistors. Therefore, the resistors will always have different widths: a common variation due to globwidth and a separate variation due to locwidth. In the example, the distribution for locwidth was chosen as narrower than for globwidth. When overlaying the measurement results, the large common variation can easily be seen; however, all voltages are different.

In summary, each reference to a parameter with a specified distribution causes a new random variable to be generated for each Monte Carlo sample. When referencing the parameter on an instance, the effect of a local variation is created. When referencing the parameter on an expression for a second parameter and using the second parameter on an instance, then the effect of a global variation is created.

Variations Specified in the Context of Subcircuits

The concept explained in the previous section applies also to subcircuits as instances, and instances within subcircuits. Here we again use the example of a physical resistor, with variation of its width.

- test4.sp uses a subcircuit for each resistor instead of the top-level resistors in test3.sp. On each subcircuit, a parameter “width” is assigned a value by an expression, which is the same for all of them. This value is then passed into the subcircuit and the resistor width gets this value. Because the expression is the same for all subcircuits, the value of parameter “width” will be the same for all subcircuits, thus it expresses a global variation. Therefore all resistors have the same width, and the terminal voltages are the same.
- In test5.sp, if a different “width” is used for the subcircuits, then the expressions are treated separately, get local variation assigned, and different values are passed into the subcircuit. In test5.sp, the differences inside of the expressions are kept numerically very small, thus the differences from the different values of “locwidth” are dominant and the results look almost identical to the ones from test3.sp.
- In test6.sp, the resistor width is assigned inside of the subcircuit. The variations get picked up from the top level. Because each subcircuit is a separate entity, the parameter “w” is treated as a separate reference, thus each resistor will have its own value, partly defined through the common value of “globwidth” and partly through the separate value of “locwidth”.
- test7.sp has two resistors in the subcircuit. Each device in each subcircuit has a separate reference to the variation, therefore each device gets its own value.
- In test8.sp, the variation definition for “locwidth” has been moved from the top level into the subcircuit. Each resistor has a common global variation and its own local variation.
- test9.sp assigns the top level variation to a local parameter, which in turn is applied to the width definition of the resistor. This happens independently within each subcircuit, thus we end up with the same values for the resistor pair in each subcircuit, but different values for the different pairs. This technique can be applied to long resistors when a middle terminal is required for connecting capacitance to the substrate. The resulting two resistor pieces will have the same resistance, but it will be different from other resistor pairs.

In summary, each subcircuit has its own parameter space, therefore it is possible to put groups of identical components into a subcircuit, and within each group all devices have the same parameter values, but between the groups, parameters are different. When specifying variations on these parameters, the effects of local variations between the groups are created.

Variations on a Model Parameter Using a Local Model in Subcircuit

If a model is specified within a subcircuit, then the specified parameter values apply only to the devices in the same subcircuit. Therefore, it is possible to calculate the value of a model parameter within the subcircuit; for example, as a function of geometry information.

When specifying variations on these parameters, the effects of local variations between subcircuits are created. If this method is used at the extreme with one device per subcircuit, then each device has its own model. This approach leads to a substantial overhead in the simulator and is therefore not recommended.

Indirect Variations on a Model Parameter

In sections Variations Specified on Geometrical Instance Parameters and Variations Specified in the Context of Subcircuits, variations on geometrical parameters were presented. If we want to specify variations on a model parameter; for example, the threshold of a MOS device, then the approach explained in the previous section with one model per device in a subcircuit could be used. However, this is impractical because the netlist needs to be created to call each device as a subcircuit, and because of the overhead. Since variations are of interest only on a few model parameters, an indirect method of varying model parameters can be used. Some special instance parameters are available for this purpose. For example, for MOS devices, the parameter delvt0 defines a shift in threshold.

Referencing a parameter with a distribution as value for `delvt0` creates the effect of local threshold variations. A significant number of parameters of this type are available in HSPICE for BSIM3 and BSIM4 models. The variations can be tailored for each device depending on its size for example. A disadvantage of this method is that the netlist needs to be parameterized properly to get the correct variations. The process of preparing a basic netlist for Monte Carlo simulations with this approach is tedious and error prone, therefore it is best handled with scripts.

For a listing of supported BSIM3 and BSIM4 instance parameters, see the *HSPICE Reference Manual: MOSFET Models*, Supported Instance Parameters, BSIM3, BSIM4, BSIM3SOI and BSIM4SOI.

Variations Specified on Model Parameters

This section discusses the method of specifying distributions on parameters and using these parameters to define values of model parameters. With this approach, the netlist does not have to be parameterized. The `modmonte` option can be used to distinguish between global variations (all devices of a particular model have the same parameter set) or local variations (every device has a unique random value for the specified parameters).

- test10.sp shows a simple case where the model parameter for sheet resistivity is assigned a distribution defined on the parameter `rsheet`. The results show that all resistors have the same value for each Monte Carlo sample, but a different one for different samples. This setup is useful for studying global variations.
- test11.sp has `.option modmonte=1` added. Now every resistor has a different value.

Note that `.option modmonte` has no effect on any other approach presented here.

In summary, assigning parameters with specified distributions to model parameters allows for investigating the effects of global or local variations, but not both. The possibility of selecting one or the other with a simple option is misleading in the sense that the underlying definitions for global and local variations are not the same for a realistic semiconductor technology.

Variations Specified Using DEV and LOT

The two limitations of the approach described in section Variations Specified on Model Parameters are resolved in this method by specifying global and local variations directly on a model parameter with the syntax:

```
parameterName=parameterValue LOT/distribution LotDist
+ DEV/distribution DevDist
```

Where,

`LOT` keyword for global distribution
`DEV` keyword for local distribution

distribution is as explained in section Variations Specified on Geometrical Instance Parameters
`LotDist`, `DevDist` characteristic number for the distribution. 3-sigma value for Gaussian distributions.

- test12.sp has large global and small local variation, similar to the setup in the file test3.sp The result shows four different curves, with a large common part and small separate parts. The amount of variation defined in the two files is the same. The curves look different from the test3.sp results because different random sequences are used. However the statistical results (sigma) converge for a large number of samples.

There is no option available to select only local or only global variations. This can be an obstacle if the file is read-only or encrypted.

Combinations of Variation Specifications

Specifying distributions on parameters and applying them to model parameters can be used on some models and the `DEV`/`LOT` approach on others in the same simulation.

- test13.sp has `DEV`/`LOT` specified for model res1, and the parameter “width” for model res2. The values for the resistors with model res1 are different, and the values for resistors with model res2 are the same.
- test14.sp is similar to test7.sp and has `modmonte=1` specified. All four resistors have different values. However, note that in reality, the sigma for width would be different when simulating local or global variations.
- test15.sp has instance parameter variations specified on two resistors and `DEV`/`LOT` on two others. From the waveforms, v3 and v4 form a first pair, and v1 and v2 a second pair.

It is also possible to mix variations on instance parameters and model parameters in the same setup.

- test16.sp has small instance parameter variations specified on width and relatively large model parameter variations on the sheet resistivity, `rsh`. The results show four different waveforms, with a common behavior.
- test17.sp shows instance and model parameter variations as in the previous test case, but `.option modmonte` is set to 1, thus the model variations affect every device in a different way. The results show completely independent behavior of all four resistors.

If an instance parameter or instance parameter variations and model parameter variations are specified on the same parameter, then the instance parameter always overrides the model parameter. Because only few parameters can be used in both domains, this case is rather seldom, but it needs to be considered to avoid unexpected results.

- test18.sp has model variation specified on width with a parameter. Two resistors have width also defined on instance. The resistors with instance parameter do not vary at all. The other two resistors vary independently, as expected because `.option modmonte` is set to 1.
- test19.sp is similar to test18.sp with `.option modmonte` set to 0. The two resistors that do not have width defined on the instance line vary together.
- test20.sp has `DEV`/`LOT` specified. Instance parameters override variations on selected resistors.

Variation on Model Parameters as a Function of Device Geometry

For local variations (see Chapter 22, Mismatch Analyses), it is a common requirement to specify variation on a model parameter as a function of device geometry. For example, the MOS device threshold was observed to vary with the total device area.

The approach explained in the section Indirect Variations on a Model Parameter can be used. While this allows for specifying local variations on each device, it does not include the capability of using expressions based on element parameters. Thus, variation cannot be described with an expression that includes the device's geometry. Conceptually, a netlist processor could be written that inserts the appropriate values for the parameters as a function of device size. (Synopsys does not make such a tool available).

The `DEV`/`LOT` approach has no mechanism to describe variation as a function of an element parameter.

Conclusion

The three approaches described above for specifying variations are not well suited for semiconductor technologies because of one or more of the following issues:

- require changes to netlist
- difficult to recognize whether a variation is global or local
- no way to describe variability as a function of device size
- no way to run only global or only local variation.

To overcome these issues, a new approach was introduced in HSPICE. This approach is based on a so called *variation block*. See Chapter 20, Analyzing Variability and Using the Variation Block, and for details on how Monte Carlo analysis is processed with this advanced approach, see Chapter 21, Monte Carlo Analysis Using the Variation Block Flow.

B

Full Simulation Example

Contains information and sample input netlist for a full simulation example in HSPICE.

The example in this appendix shows the basic text and post-processor output for two sample input netlists. The example uses WaveView to view the results.

Simulation Example Using WaveView

This example demonstrates the basic steps to perform a simulation and to view the waveform results by using the Synopsys WaveView waveform viewer.

Input Netlist and Circuit

This example is based on demonstration netlist example.sp, which is available in directory $*installdir*/demo/hspice/bench. This example is an input netlist for a linear CMOS amplifier. Comment lines indicate the individual sections of the netlist. See the HSPICE Reference Manual: Commands and Control Options for information about individual commands.

Input netlist

```
* Example HSPICE netlist, using a linear CMOS amplifier
* netlist options
.option post probe brief nomod
* defined parameters
.param analog_voltage=1.0
* global definitions
.global vdd
* source statements
Vinput in gnd SIN ( 0.0v analog_voltage 10x )
Vsupply vdd gnd DC=5.0v
* circuit statements
Rinterm in gnd 51
Cincap in infilt 0.001
Rdamp infilt clamp 100
Dlow gnd clamp diode_mod
Dhigh clamp vdd diode_mod
Xinv1 clamp inv1out inverter
Rpull clamp inv1out 1x
Xinv2 inv1out inv2out inverter
Routterm inv2out gnd 100x
* subcircuit definitions
.subckt inverter in out
Mpmos out in vdd vdd pmos_mod l=1u w=6u
Mnmos out in gnd gnd nmos_mod l=1u w=2u
.ends
* model definitions
.model pmos_mod pmos level=3
.model nmos_mod nmos level=3
.model diode_mod d
* analysis specifications
.TRAN 10n 1u sweep analog_voltage lin 5 1.0 5.0
* output specifications
.probe TRAN v(in) v(clamp) v(inv1out) v(inv2out) i(dlow)
.measure TRAN falltime TRIG v(inv2out) VAL=4.5v FALL=1
+ TARG V(inv2out) VAL=0.5v FALL=1
.end
```

Figure 234 on page 909 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. The diode and MOSFET models in the netlist do not have non-default parameter values, except to specify Level 3 MOSFET models (empirical model).

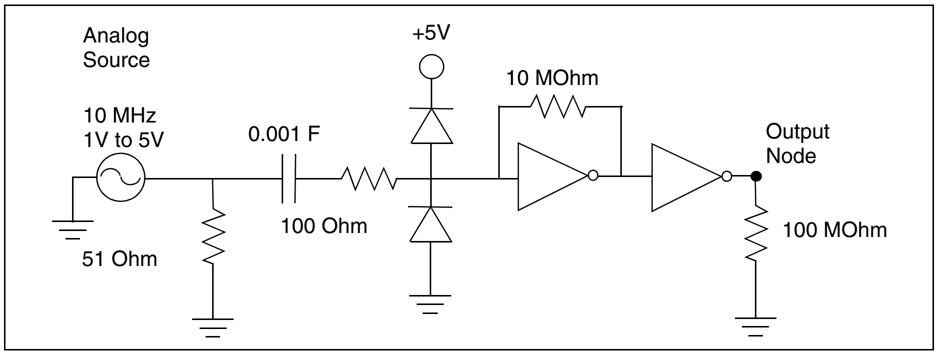


Figure 234 Circuit Diagram for Linear CMOS Inverter

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 107 HSPICE Output Files

Filename	Description
example.ic	Initial conditions for the circuit.
example.lis	Text simulation output listing.
example.mt0	Post-processor output for .MEASURE statements.
example.pa0	Subcircuit path table.
example.st0	Run-time statistics.

Table 107 HSPICE Output Files (Continued)

Filename	Description
example.tr0	Post-processor output for transient analysis.

The following subsections show text files to simulate the amplifier by using HSPICE on a UNIX workstation. The example does not show the two post-processor output files, which are in binary format.

Example.ic0

Initial conditions for the circuit

```
* "simulator" "HSPICE"
* "version" "A-2008.03 32-BIT"
* "format" "HSP"
* "rundate" "15:39:17 04/07/2008"
* "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.mt0

Post-processor output for .MEASURE statements

```
$DATA1 SOURCE='HSPICE' VERSION='A-2008.03 32-BIT'
.TITLE '* example hspice netlist, using a linear cmos amplifier'
 analog_voltage   falltime         temper           alter#
 1.0000         3.788e-08        25.0000          1.0000
 2.0000         1.639e-08        25.0000          1.0000
 3.0000         1.085e-08        25.0000          1.0000
 4.0000         7.843e-09        25.0000          1.0000
 5.0000         6.572e-09        25.0000          1.0000
```

Example.lis

Simulation output listing text file

```
Using: /usr/bin/time -p /remote/cktcae/HSPICE/A-2008.03/hspice/
linux/hspice example.sp

****** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux ******
Copyright (C) 2008 by Synopsys Inc. All rights reserved.
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:

 /remote/cktcae/HSPICE/A-2008.03/hspice/license.warn

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: v8.5b
lic: USER:   hspiceuser           HOSTNAME: hspiceamd1
lic: HOSTID: 0015605ff6da         PID:      18077
lic: Using FLEXlm license file:
lic: 26585@us01_lic4
lic: Checkout 1 hspice
lic: License/Maintenance for hspice will expire on 31-jan-2009/
2009.12
lic: FLOATING license(s) on SERVER us01_lic4
lic:
Init: hspice initialization file: hspice.ini
 * netlist options
 .option post probe brief nomod

***************************************************************
 ** runlvl is invoked, you can disable it by:
      a) Add option runlvl=0 to your current simulation job.
      b) Copy $installdir/hspice.ini to your HOME directory and
         customize it by adding option runlvl=0, which disables
         it for all of your simulation jobs.
      c) Re-invoke $installdir/bin/config program and unselect the
         option runlvl setting in box 'hspice.ini' which disables
         it for whole group simulation jobs.
 ** runlvl is invoked, some options are ignored or automatically
    set:
      Options below are automatically set (user setting will
      overwrite them):
      if runlvl=6,            .option bypass=0
      if runlvl=[1|2|3|4|5],  .option bypass=2
      Options below are ignored, they are replaced by automated
      algorithms:
        lvltim    dvdt      ft    fast   trtol  absvar  relvar
          relq  chgtol    dvtr    imin    itl3    rmax
 ** runlvl is invoked, actual option value used by HSPICE are:
     runlvl= 3     bypass= 2    mbypass=  2.00  bytol= 100.00u

***************************************************************

**warning** dc voltage reset to initial transient source value
             in source        0:vinput    new dc=  0.0000D+00

   Opening plot unit= 15
 file=example.pa0

 *** parameter analog_voltage   =    1.000E+00 ***

******
* example hspice netlist, using a linear cmos amplifier
******  transient analysis tnom=  25.000 temp=  25.000 ******
 falltime=  3.7885E-08  targ=  7.0408E-08   trig=  3.2524E-08

          ***** job concluded

*** parameter analog_voltage   =    2.000E+00 ***

******
 * example hspice netlist, using a linear cmos amplifier

 ******  transient analysis tnom=  25.000 temp=  25.000    ******
   falltime=  1.6394E-08  targ=  5.8128E-08   trig=  4.1734E-08

          ***** job concluded

 *** parameter analog_voltage   =    3.000E+00 ***
 ******

 * example hspice netlist, using a linear cmos amplifier

 ******  transient analysis         tnom=  25.000 temp=  25.000

 ******
   falltime=  1.0848E-08  targ=  5.5187E-08   trig=  4.4339E-08

          ***** job concluded

 *** parameter analog_voltage   =    4.000E+00 ***
 ******
```

```
* example hspice netlist, using a linear cmos amplifier

 ******  transient analysis tnom=  25.000 temp=  25.000 ******
  falltime=  7.8434E-09  targ=  5.4334E-08   trig=  4.6490E-08

         ***** job concluded

*** parameter analog_voltage   =    5.000E+00 ***

******
* example hspice netlist, using a linear cmos amplifier

 ******  transient analysis tnom=  25.000 temp=  25.000 ******
  falltime=  6.5718E-09  targ=  5.3069E-08   trig=  4.6497E-08

  meas_variable = falltime
  mean  =  15.9083n      varian = 165.2538a
  sigma =  12.8551n      avgdev =   8.9848n
  max   =  37.8846n      min    =   6.5718n
1-sigma =  12.8551n      median =  16.3940n

         ***** job concluded

****** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux ******
******
* example hspice netlist, using a linear cmos amplifier
****** job statistics summary     tnom= 25.000 temp= 25.000
******

          total memory used        159 kbytes
# nodes =       8 # elements=      14
# diodes=     2 # bjts   =     0 # jfets  =     0 # mosfets =     4
# va device =       0

  analysis time    # points   tot. iter  conv.iter
  op point  0.00     1          55
  transient 0.06     505       6881        2043 rev=        58
  readin             0.00
  errchk             0.00
  setup              0.00
  output             0.00
          total cpu time          0.06 seconds
          total elapsed time         1 seconds
              job started at  15:39:17 04/07/2008
              job ended   at  15:39:18 04/07/2008

Init: hspice initialization file: hspice.ini
lic: Release hspice token(s)
```

Example.pa0

Subcircuit path output

```
1 xinv1.
2 xinv2.
```

Example.st0

Run-time statistics

```
***** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux ******
Input File: example.sp
 lic:
 lic: FLEXlm: v8.5b
 lic: USER:   hspiceuser           HOSTNAME: hspiceamd1
 lic: HOSTID: 0015605ff6da         PID:      18077
 lic: Using FLEXlm license file:
 lic: 26585@us01_lic4
 lic: Checkout 1 hspice
 lic: License/Maintenance for hspice will expire on 31-jan-2009/
2009.12
 lic: FLOATING license(s) on SERVER us01_lic4
 lic:
 Init: hspice initialization file: hspice.ini
 init: begin read circuit files,  cpu clock=  0.00E+00
       option post
       option probe
       option brief
 init: end read circuit files,  cpu clock=  0.00E+00 memory=    145 kb
 init: begin check errors,  cpu clock=  0.00E+00
 init: end check errors,  cpu clock=  0.00E+00 memory=    145 kb
 init: begin setup matrix, pivot=    10 cpu clock=  0.00E+00
 establish matrix -- done,  cpu clock=  0.00E+00 memory=    146 kb
 re-order matrix -- done,  cpu clock=  0.00E+00 memory=    147 kb
 init: end setup matrix,  cpu clock=  0.00E+00 memory=    158 kb
 sweep: parameter       parameter1       begin, #sweeps =   5
 parameter: analog_voltage  =    1.00E+00
 dcop: begin dcop,  cpu clock=  0.00E+00
 dcop: end dcop,  cpu clock=  0.00E+00 memory=   158 kb  tot_iter=
11
 output: example.mt0
 sweep: tran tran1    begin, stop_t=  1.00E-06 #sweeps= 101 cpu
clock=  0.00E+00
 tran: time= 1.0000E-07 tot_iter=      97 conv_iter=      31 cpu
clock= 1.00E-02
 tran: time= 2.0000E-07 tot_iter=     195 conv_iter=      62 cpu
clock= 1.00E-02
 tran: time= 3.0000E-07 tot_iter=     296 conv_iter=      93 cpu
clock= 1.00E-02
 tran: time= 4.0000E-07 tot_iter=     392 conv_iter=     124 cpu
clock= 1.00E-02
 tran: time= 5.0000E-07 tot_iter=     484 conv_iter=     154 cpu
clock= 1.00E-02
 tran: time= 6.0000E-07 tot_iter=     580 conv_iter=     184 cpu
clock= 1.00E-02
 tran: time= 7.0000E-07 tot_iter=     677 conv_iter=     215 cpu
```

```
clock= 1.00E-02
 tran: time= 8.0000E-07 tot_iter=     778 conv_iter=     246 cpu
clock= 1.00E-02
 tran: time= 9.0000E-07 tot_iter=     874 conv_iter=     277 cpu
clock= 1.00E-02
 tran: time= 1.0000E-06 tot_iter=     966 conv_iter=     307 cpu
clock= 1.00E-02
 sweep: tran tran1    end,  cpu clock=  1.00E-02 memory=    159 kb
 parameter: analog_voltage  =    2.00E+00
 dcop: begin dcop,  cpu clock=  1.00E-02
 dcop: end dcop, cpu clock= 1.00E-02 memory=   158 kb  tot_iter=
22
 output: example.mt0
 sweep: tran tran2    begin, stop_t=  1.00E-06 #sweeps= 101 cpu
clock=  1.00E-02
 tran: time= 1.0000E-07 tot_iter=     101 conv_iter=      34 cpu
clock= 1.00E-02
 tran: time= 2.0000E-07 tot_iter=     200 conv_iter=      68 cpu
clock= 2.00E-02
 tran: time= 3.0000E-07 tot_iter=     305 conv_iter=     101 cpu
clock= 2.00E-02
 tran: time= 4.0000E-07 tot_iter=     415 conv_iter=     136 cpu
clock= 2.00E-02
 tran: time= 5.0000E-07 tot_iter=     535 conv_iter=     174 cpu
clock= 2.00E-02
 tran: time= 6.0000E-07 tot_iter=     652 conv_iter=     210 cpu
clock= 2.00E-02
 tran: time= 7.0000E-07 tot_iter=     769 conv_iter=     246 cpu
clock= 2.00E-02
 tran: time= 8.0000E-07 tot_iter=     886 conv_iter=     282 cpu
clock= 2.00E-02
 tran: time= 9.0000E-07 tot_iter=    1003 conv_iter=     318 cpu
clock= 2.00E-02
 tran: time= 1.0000E-06 tot_iter=    1120 conv_iter=     354 cpu
clock= 2.00E-02
 sweep: tran tran2    end,  cpu clock=  2.00E-02 memory=    159 kb
 parameter: analog_voltage  =    3.00E+00
 dcop: begin dcop,  cpu clock=  2.00E-02
 dcop: end dcop, cpu clock= 2.00E-02 memory=   158 kb  tot_iter=
33
 output: example.mt0
 sweep: tran tran3    begin, stop_t=  1.00E-06 #sweeps= 101 cpu
clock=  2.00E-02
 tran: time= 1.0000E-07 tot_iter=     130 conv_iter=      40 cpu
clock= 3.00E-02
 tran: time= 2.0000E-07 tot_iter=     253 conv_iter=      76 cpu
clock= 3.00E-02
 tran: time= 3.0000E-07 tot_iter=     376 conv_iter=     112 cpu
clock= 3.00E-02
 tran: time= 4.0000E-07 tot_iter=     499 conv_iter=     148 cpu
```

```
clock= 3.00E-02
 tran: time= 5.0000E-07 tot_iter=     622 conv_iter=     184 cpu
clock= 3.00E-02
 tran: time= 6.0000E-07 tot_iter=     745 conv_iter=     220 cpu
clock= 3.00E-02
 tran: time= 7.0000E-07 tot_iter=     868 conv_iter=     256 cpu
clock= 3.00E-02
 tran: time= 8.0000E-07 tot_iter=     991 conv_iter=     292 cpu
clock= 3.00E-02
 tran: time= 9.0000E-07 tot_iter=    1114 conv_iter=     328 cpu
clock= 3.00E-02
 tran: time= 1.0000E-06 tot_iter=    1237 conv_iter=     364 cpu
clock= 3.00E-02
 sweep: tran tran3    end,  cpu clock=  3.00E-02 memory=    159 kb
 parameter: analog_voltage  =    4.00E+00
 dcop: begin dcop,  cpu clock=  3.00E-02
 dcop: end dcop, cpu clock= 3.00E-02 memory=  158 kb tot_iter=
44
 output: example.mt0
 sweep: tran tran4    begin, stop_t=  1.00E-06 #sweeps= 101 cpu
clock=  3.00E-02
 tran: time= 1.0000E-07 tot_iter=     159 conv_iter=      45 cpu
clock= 3.00E-02
 tran: time= 2.0000E-07 tot_iter=     322 conv_iter=      92 cpu
clock= 4.00E-02
 tran: time= 3.0000E-07 tot_iter=     485 conv_iter=     139 cpu
clock= 4.00E-02
 tran: time= 4.0000E-07 tot_iter=     648 conv_iter=     186 cpu
clock= 4.00E-02
 tran: time= 5.0000E-07 tot_iter=     811 conv_iter=     233 cpu
clock= 4.00E-02
 tran: time= 6.0000E-07 tot_iter=     974 conv_iter=     280 cpu
clock= 4.00E-02
 tran: time= 7.0000E-07 tot_iter=    1137 conv_iter=     327 cpu
clock= 4.00E-02
 tran: time= 8.0000E-07 tot_iter=    1300 conv_iter=     374 cpu
clock= 4.00E-02
 tran: time= 9.0000E-07 tot_iter=    1463 conv_iter=     421 cpu
clock= 5.00E-02
 tran: time= 1.0000E-06 tot_iter=    1626 conv_iter=     468 cpu
clock= 5.00E-02
 sweep: tran tran4    end,  cpu clock=  5.00E-02 memory=    159 kb
 parameter: analog_voltage  =    5.00E+00
 dcop: begin dcop,  cpu clock=  5.00E-02
 dcop: end dcop, cpu clock= 5.00E-02 memory=  158 kb tot_iter=
55
 output: example.mt0
 sweep: tran tran5    begin, stop_t=  1.00E-06 #sweeps= 101 cpu
clock=  5.00E-02
 tran: time= 1.0000E-07 tot_iter=     173 conv_iter=      48 cpu
```

```
clock= 5.00E-02
 tran: time= 2.0000E-07 tot_iter=    378 conv_iter=    104 cpu
clock= 5.00E-02
 tran: time= 3.0000E-07 tot_iter=    573 conv_iter=    163 cpu
clock= 5.00E-02
 tran: time= 4.0000E-07 tot_iter=    786 conv_iter=    222 cpu
clock= 5.00E-02
 tran: time= 5.0000E-07 tot_iter=    991 conv_iter=    278 cpu
clock= 5.00E-02
 tran: time= 6.0000E-07 tot_iter=   1184 conv_iter=    334 cpu
clock= 6.00E-02
 tran: time= 7.0000E-07 tot_iter=   1379 conv_iter=    388 cpu
clock= 6.00E-02
 tran: time= 8.0000E-07 tot_iter=   1569 conv_iter=    443 cpu
clock= 6.00E-02
 tran: time= 9.0000E-07 tot_iter=   1741 conv_iter=    494 cpu
clock= 6.00E-02
 tran: time= 1.0000E-06 tot_iter=   1912 conv_iter=    545 cpu
clock= 6.00E-02
 sweep: tran tran5   end,  cpu clock=  6.00E-02 memory=   159 kb
 sweep: parameter       parameter       1 end
>info:         ***** hspice job concluded
 Init: hspice initialization file: hspice.ini
 lic: Release hspice token(s)
```

View HSPICE Results in WaveView

These steps show how to use the Synopsys WaveView Waveform Viewer to view the results of the transient analysis from the linear CMOS amplifier simulation.

To view HSPICE transient analysis waveforms, do the following:

1. Invoke SPICE Explorer—From a UNIX command line, type:

   ```
   %  sx
   ```

   On a Windows-NT system, choose the menu command:

   Programs > (*user_install_location*) > SPICE Explorer

2. In the menu bar, select File > Import Waveform file… (Ctrl-O)
3. In the Open: Waveform Files dialog box, click on *example.tr0* and click **OK**. This opens the transient analysis waveform file.
4. Expand the hierarchy in the output view browser to show the available output signals.

5. Left-click on the signal *v(in*, then drag and drop the selected signal into the WaveView panel to display a plot similar to the one shown in Figure 235 on page 919.

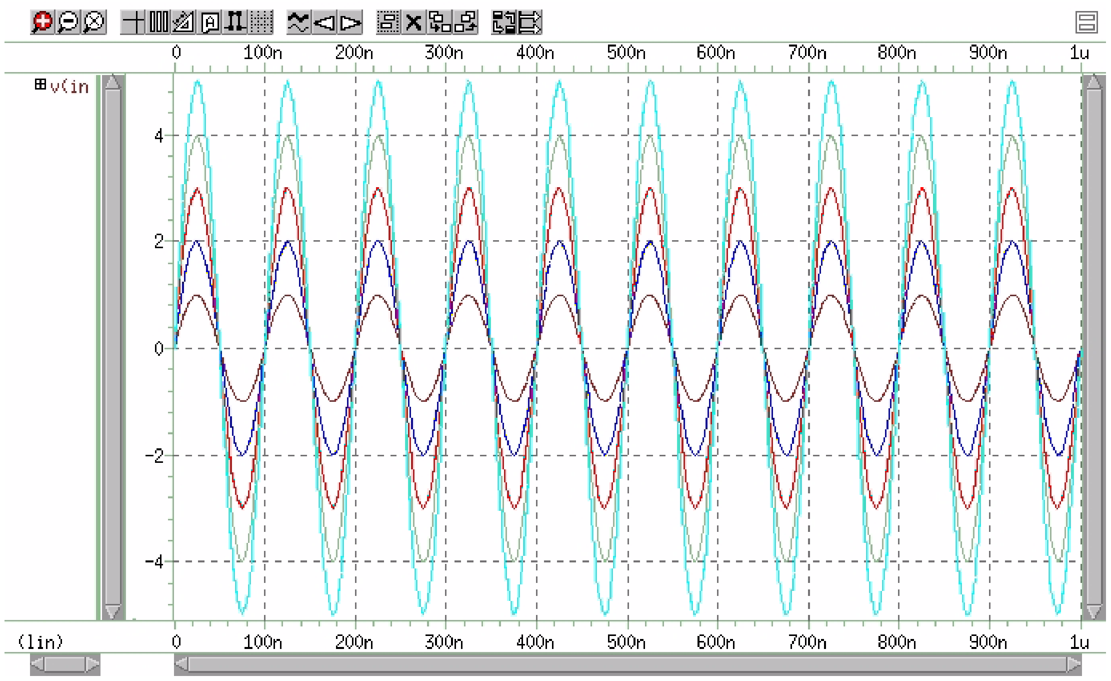


Figure 235 Plot of Voltage on Node in

6. To delete this plot, click on the **X** icon in the WaveView panel tool bar.
7. Repeat Step 5 to plot *v(clamp*. You should see a plot similar to the one shown in Figure 240.

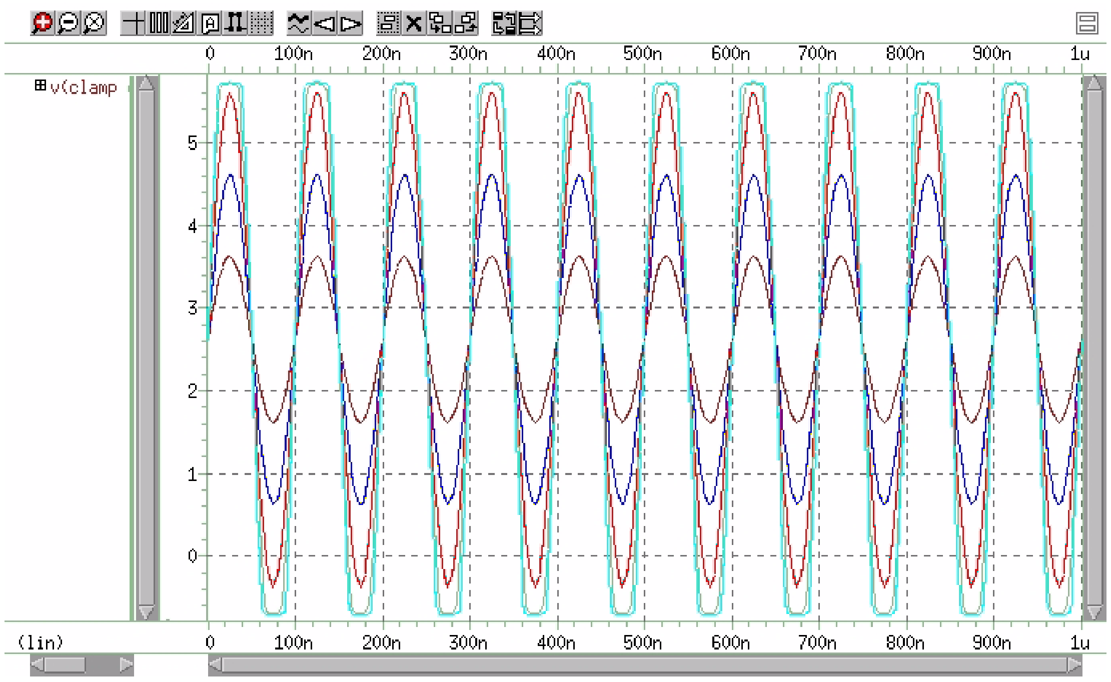


Figure 236 Plot of Voltage on Node clamp vs. Time

8. Repeat Step 6 and Step 7 to view the signals *v(inv1out* and *v(inv2out* (Figure 237 and Figure 238 on page 922).

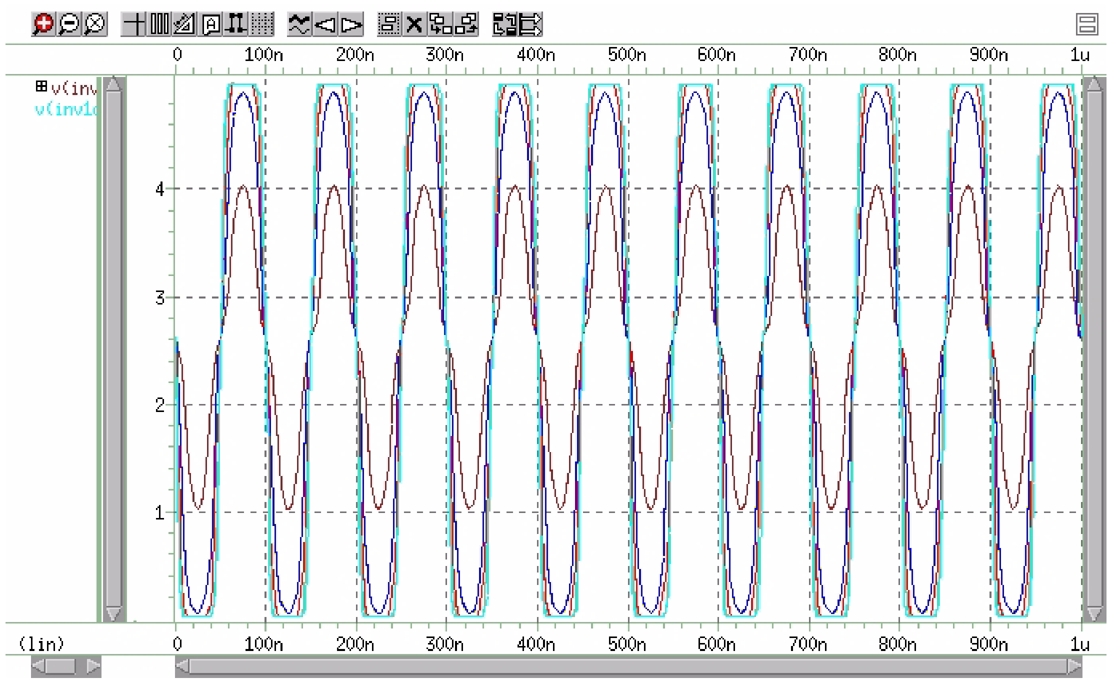


Figure 237   Plot of Voltage on Node inv1out vs.Time

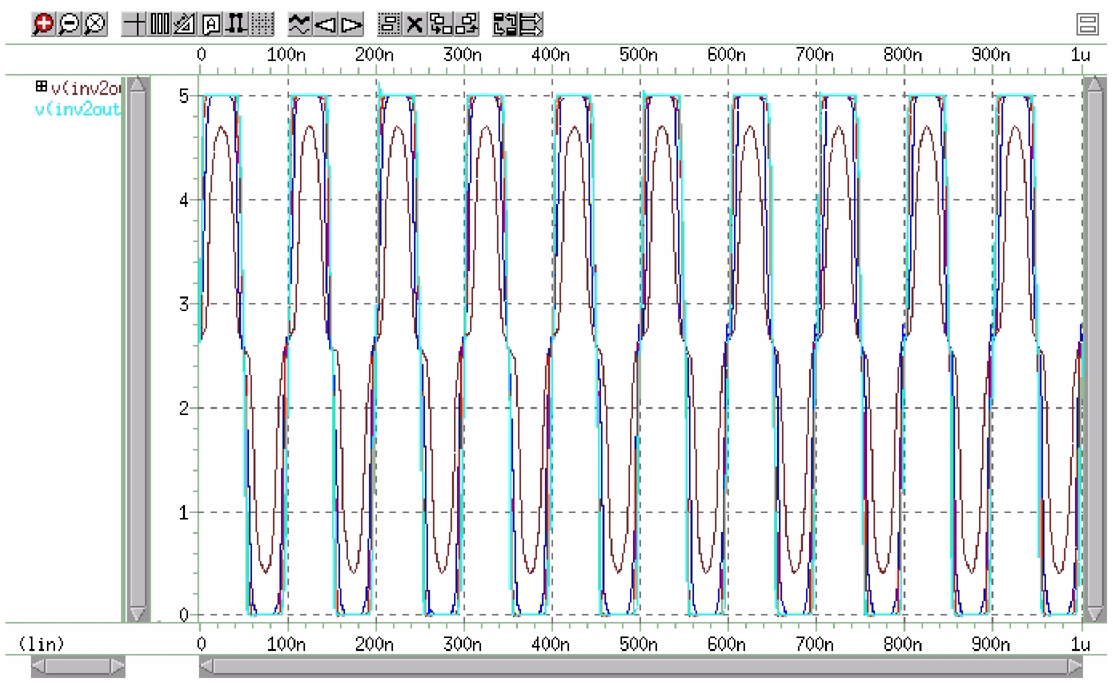


Figure 238 Plot of Voltage on Node inv2out vs. Time

9. To delete this plot, click the **X** icon in the WaveView panel tool bar.
10. Left-click on the signal *i(dlow*, then drag and drop the selected signal into the WaveView panel.
11. To show the individual members from the voltage sweep, expand the signal hierarchy by clicking on the '+' sign near the signal name in the WaveView panel.
12. Click the ungroup panels icon ( )on the WaveView panel tool bar to display a plot similar to the one shown in Figure 239 on page 923.

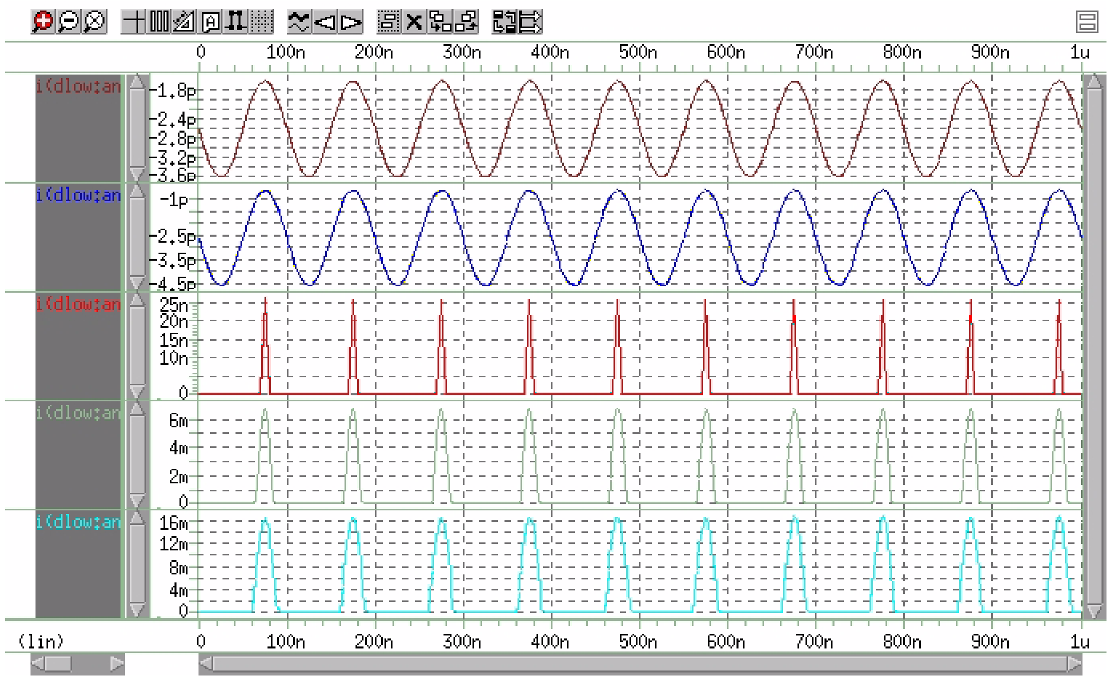


Figure 239 Plot of Current through Diode dlow vs. Time

13. To delete this plot, click the **X** icon in the WaveView panel tool bar.
14. In the menu bar, select File > Import Waveform file (Ctrl-O).
15. In the Open: Waveform Files dialog box, click on *example.mt0* and click **OK** to open the measure file.
16. Expand the hierarchy in the output view browser to show the available output signals.
17. Left-click on the signal *falltime*. Drag and drop the selected signal into the WaveView panel. You should see a plot similar to the one shown in Figure 240 on page 924.

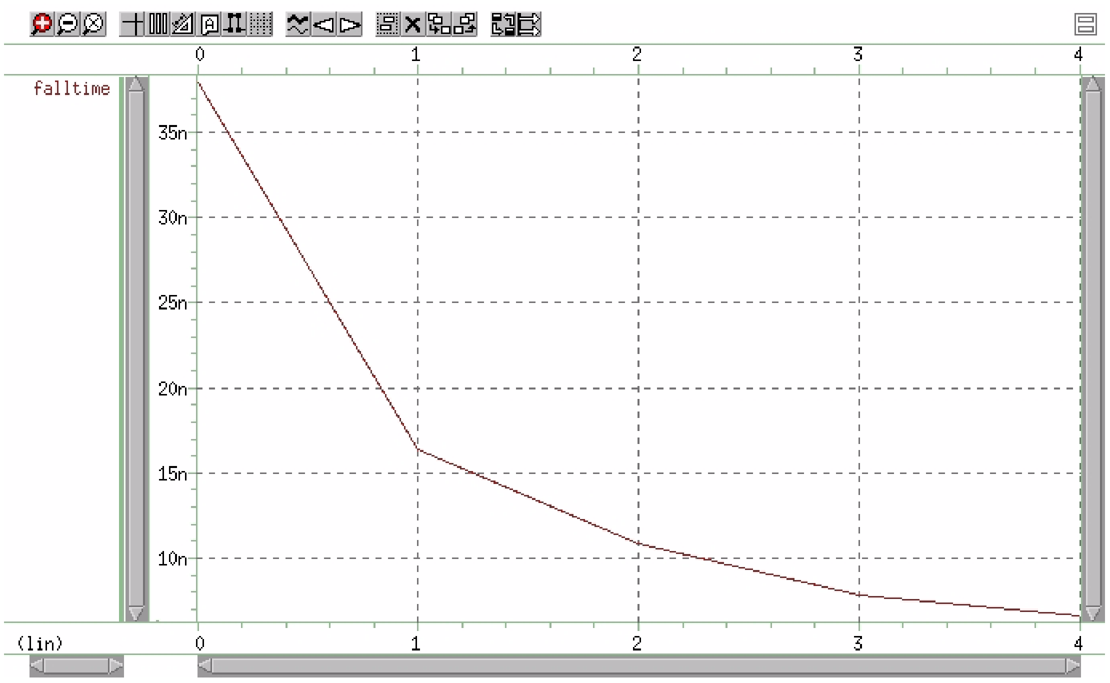


Figure 240 Plot of Measured Variable falltime vs. Amplifier Input Voltage

This concludes the simulation example using WaveView.

The *SPICE Explorer and WaveView User's Manual* includes a full tutorial, information about the various SPICE Explorer tools, and reference information. You can also find more information on the Synopsys website: http:// www.synopsys.com.

C
Obsolete HSPICE Functionality

Describes out-of-date, rarely used, or de-emphasized functionality.

The following sections are included in the HSPICE documentation set for completeness only. Most of this material has been replaced by more accurate, efficient, and useful methodologies.

U-element Digital and Mixed Mode Stimuli

HSPICE input netlists support two types of digital stimuli: digital vector files (described in the Chapter 9 section, Specifying a Digital Vector File and Mixed Mode Stimuli) and U-element digital input files, described in the following sections.

U-element Digital Input Elements and Models

This section describes the input file format for the digital input U Element.

In HSPICE, the U-element can reference digital input and digital output models for mixed-mode simulation. If you run HSPICE in standalone mode, the state information originates from a digital file. Digital outputs are handled in a similar fashion. In digital input file mode, the input file is named <design>.d2a, and the output file is named <design>.a2d.

A2D and D2A functions accept the terminal “\” backslash character as a line-continuation character, to allow more than 255 characters in a line. Use line continuation if the first line of a digital file, which contains the signal name list, is longer than the maximum line length that your text editor accepts.

Do not put a blank first line in a digital D2A file. If the first line of a digital file is blank, HSPICE issues an error message.

Example

The following example demonstrates how to use the “\” line continuation character, to format an input file for text editing. The example file contains a signal list for a 64-bit bus.

```
...
a00 a01 a02 a03 a04 a05 a06 a07 \
a08 a09 a10 a11 a12 a13 a14 a15 \
... * Continuation of signal names
a56 a57 a58 a59 a60 a61 a62 a63 End of signal names
... Remainder of file
```

General Form

```
Uxxx interface nlo nhi mname SIGNAME=sname IS=val
```

Parameter	Description
Uxxx	Digital input element name. Must begin with U, followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit, to which the digital input attaches.
nlo	Node connected to the low-level reference.
nhi	Node connected to the high-level reference.
mname	Digital input model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. Can be a string of up to eight alphanumeric characters.
IS	Initial state of the input element. Must be a state that the model defines.

Model Syntax

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

Digital-to-Analog Input Model Parameters

Table 108 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 241 is the circuit representation of the element.

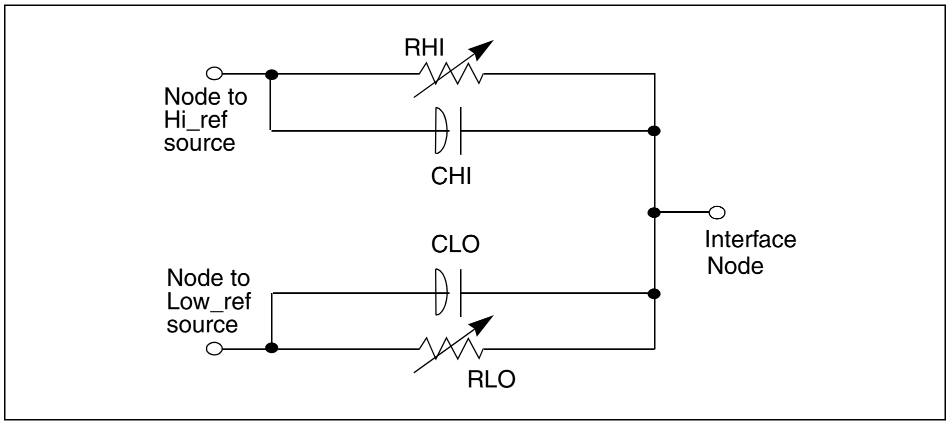


Figure 241 Digital-to-Analog Converter Element

Example

The following example shows how to use the U element and model, as a digital input for a HSPICE netlist.

You can find the sample netlist for this example in the following directory:

$installdir/demo/hspice/sources/uelm.sp

The associated digital input file is:

```
1
00 1:1
09 z:1
10 0:1
11 z:1
20 1:1
30 0:1
39 x:1
40 1:1
41 x:1
50 0:1
60 1:1
70 0:1
80 1:1
```

U Element Digital Outputs

Digital output (not supported in HSPICE RF).

Syntax

```
Uxxx interface reference mname SIGNAME=sname
```

Parameter	Description
Uxxx	Digital output element name. Must begin with U, followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit, at which HSPICE measures the digital output.
reference	Node to use as a reference for the output.
mname	Digital output model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. A string of up to eight alphanumeric characters.

Model Syntax

```
.MODEL mname U LEVEL=4 <parameters...>
```

Analog-to-Digital Output Model Parameters

Table 109 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.

Table 109 Analog-to-Digital Parameters (Continued)

Name (Alias)	Units	Default	Description
S1VLO	volt		State 1 low-level voltage.
S1VHI	volt		State 1 high-level voltage.
S19NAME			State 19 character abbreviation. A string of up to four alphanumerical characters.
S19VLO	volt		State 19 low-level voltage.
S19VHI	volt		State 19 high-level voltage.
TIMESTEP	sec	1E-9	Step size for digital input file.
TIMESCALE			Scale factor for time.

To define up to 20 different states in the model definition, use the S*n*NAME, S*n*VLO and S*n*VHI parameters, where *n* ranges from 0 to 19. Figure 242 shows the circuit representation of the element.

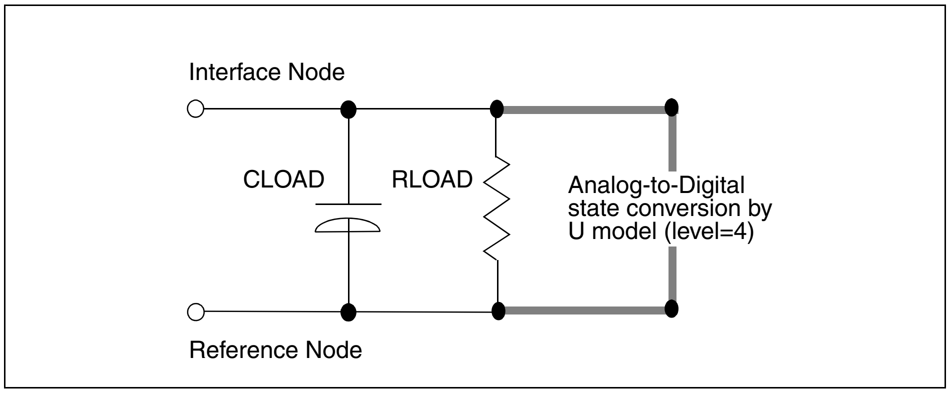


Figure 242 Analog-to-Digital Converter Element

Replacing Sources With Digital Inputs

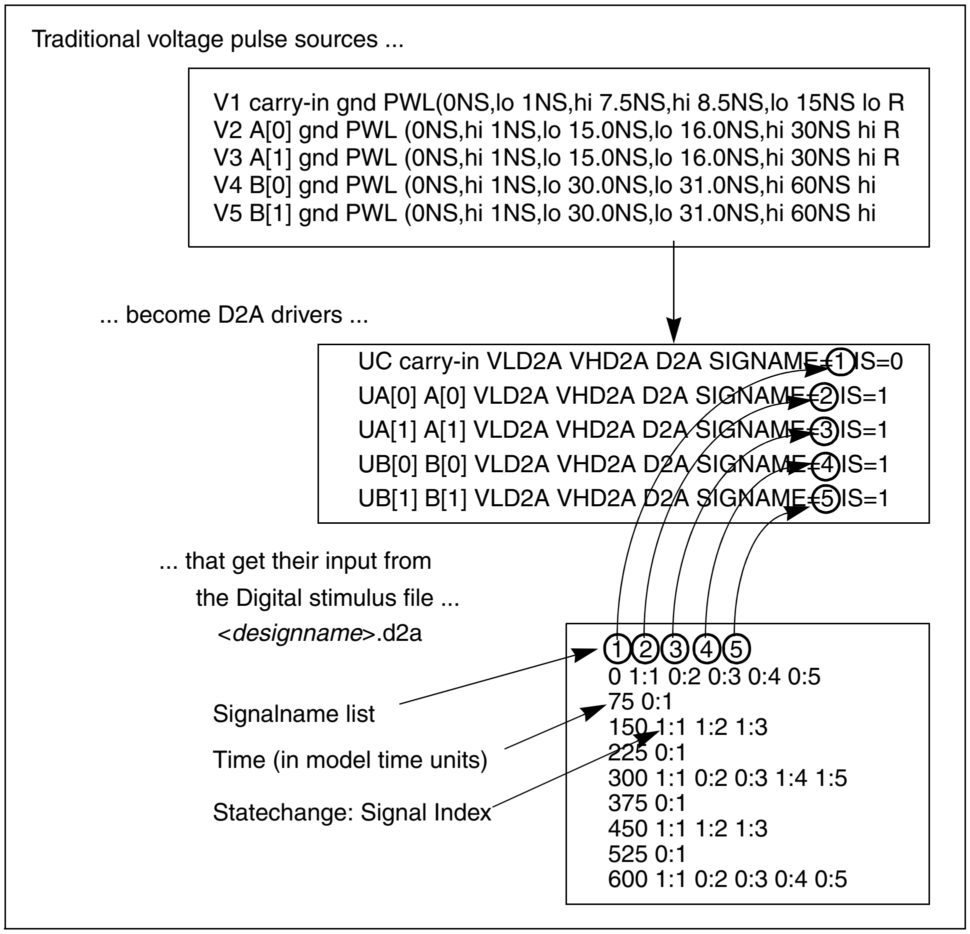

Traditional voltage pulse sources ...
V1 carry-in gnd PWL(0NS,lo 1NS,hi 7.5NS,hi 8.5NS,lo 15NS lo R
V2 A[0] gnd PWL (0NS,hi 1NS,lo 15.0NS,lo 16.0NS,hi 30NS hi R
V3 A[1] gnd PWL (0NS,hi 1NS,lo 15.0NS,lo 16.0NS,hi 30NS hi R
V4 B[0] gnd PWL (0NS,hi 1NS,lo 30.0NS,lo 31.0NS,hi 60NS hi
V5 B[1] gnd PWL (0NS,hi 1NS,lo 30.0NS,lo 31.0NS,hi 60NS hi
... become D2A drivers ...
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
... that get their input from
the Digital stimulus file ...
<*designname*>.d2a
Signalname list
Time (in model time units)
Statechange: Signal Index
1 2 3 4 5
0 1:1 0:2 0:3 0:4 0:5
75 0:1
150 1:1 1:2 1:3
225 0:1
300 1:1 0:2 0:3 1:4 1:5
375 0:1
450 1:1 1:2 1:3
525 0:1
600 1:1 0:2 0:3 0:4 0:5

Figure 243 Digital File Signal Correspondence

Example

The following is an example of replacing sources with digital inputs. This example is based on demonstration netlist digin.sp, which is available in directory $*installdir*/demo/hspice/cchar:

```
* EXAMPLE OF U-ELEMENT DIGITAL OUTPUT
.OPTION POST
VOUT carry_out GND PWL 0N 0V 10N 0V 11N 5V 19N 5V 20N 0V
+ 30N 0V 31N 5V 39N 5V 40N 0V
VREF REF GND DC 0.0V
UCO carry_out REF A2D SIGNAME=12
R1 REF 0 1k

* DEFAULT DIGITAL OUTPUT MODEL (no "X" value)

.MODEL A2D U LEVEL=4 TIMESTEP=0.1NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI= 2.7
+ S4NAME=1 S4VLO= 1.4 S4VHI=9.0
+ CLOAD=0.05pf
.TRAN 1N 500N
.END
```

The digital output file should look something like this:

```
12
0        0:1
105      1:1
197      0:1
305      1:1
397      0:1
```

- *12* represents the signal name
- The first column is the time, in units of 0.1 nanoseconds.
- The second column has the signal value: signal index pairs. Signal index is corresponds to the position of signal name in the signal name list.
- This file uses more columns to represent subsequent outputs.

For another example, see the file identified and the plot in Figure 27.

$installdir/demo/hspice/cchar/mos2bit.sp

See the plot in Figure 244 on page 933.

In this example, a 2-bit MOS adder uses a digital input file. In the plot, the `a[0]`, `a[1]`, `b[0]`, `b[1]`, and `carry-in` nodes all originate from a digital file input similar to Figure 243 on page 931. HSPICE outputs a digital file.

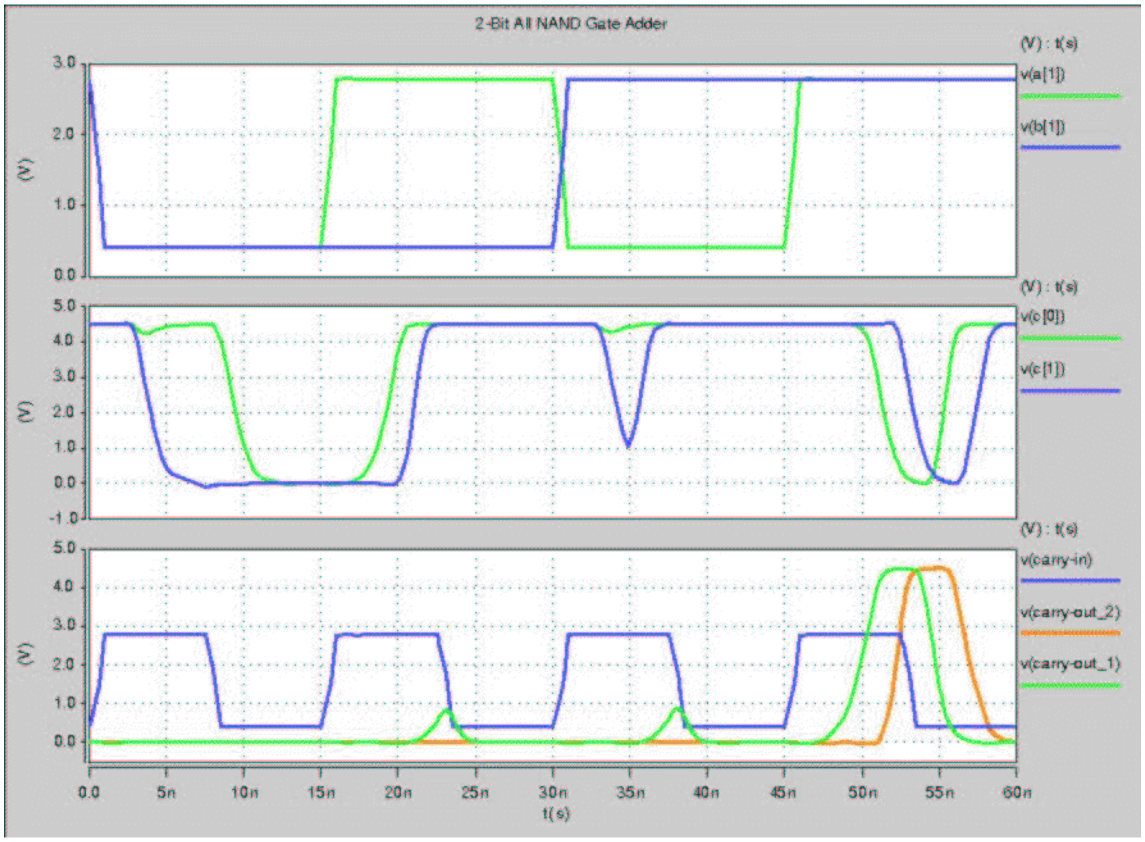


Figure 244 Digital Stimulus File Input

.NET Parameter Analysis

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 or HSPICE RF calculates:

```
.NET Vout Isrc   V  =  [Z]     [I]
.NET Iout Vsrc   I  =  [Y]     [V]
.NET Iout Isrc   [V1 I2]T  =  [H]    [I1 V2]T
.NET Vout Vsrc   [I1 V2]T  =  [S]    [V1 I2]T
([M]T represents the transpose of the M matrix).
```

Note:

The preceding list does not mean that you must use combination (1) to calculate Z parameters. However, if you specify `.NET Vout Isrc`, HSPICE or HSPICE RF initially evaluates the Z matrix parameters. It then uses standard conversion equations to determine S-parameters or any other requested parameters.

Figure 245 shows the importance of variables in the `.NET` statement. Here, `Isrc` and `Vce` are the DC biases, applied to the BJT.

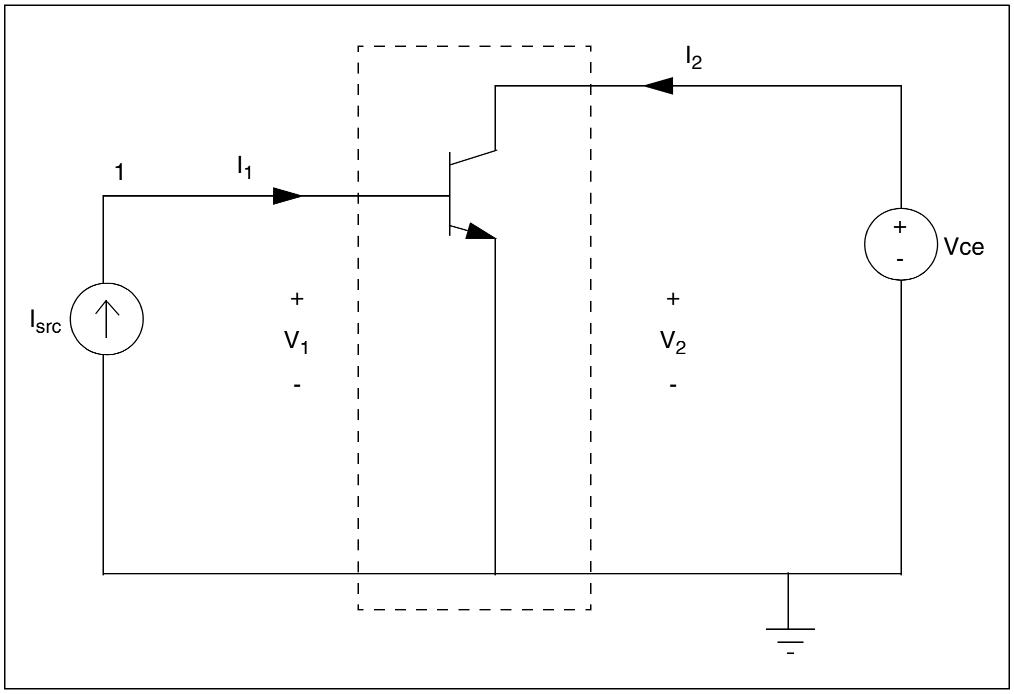


Figure 245 Parameters with .NET V(2) Isrc

This `.NET` statement provides an incorrect result for the Z parameter calculation:

```
.NET V(2) Isrc
```

When HSPICE or HSPICE RF runs AC analysis, it shorts all DC voltage sources; all DC current sources are open-circuited. As a result, V(2) shorts to ground and its value is zero in AC analysis. This affects the results of the network analysis.

In this example, HSPICE or HSPICE RF 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 246 shows the correct biasing configurations for performing network analysis for the Z, Y, H, and S-parameters.

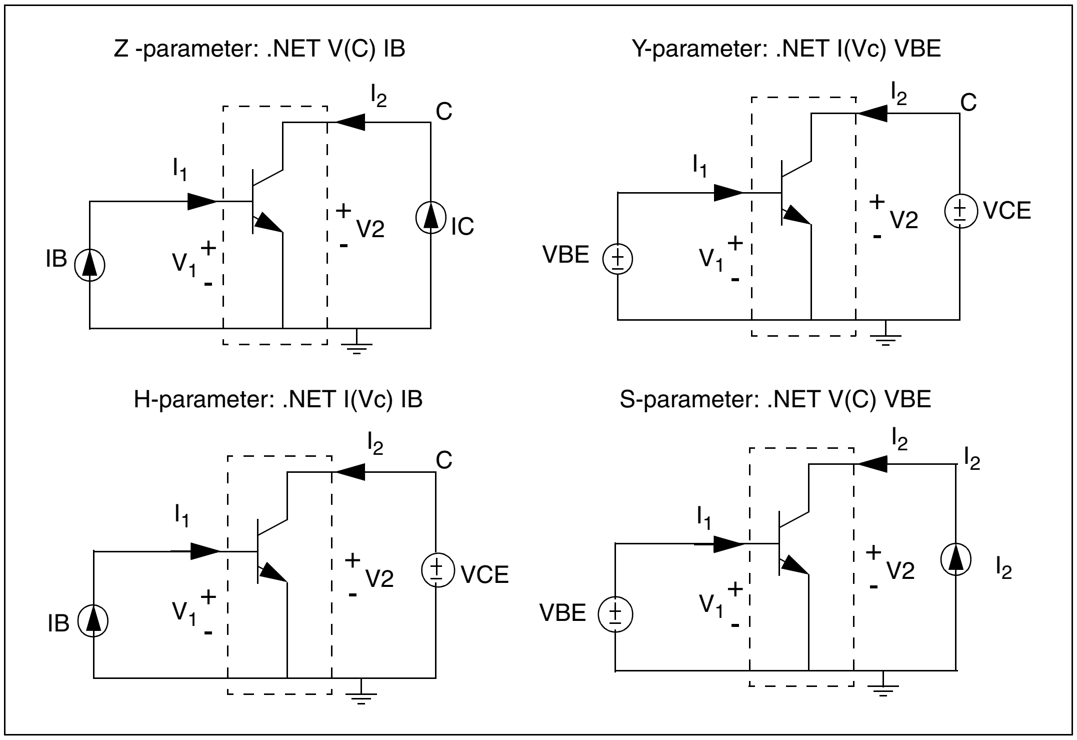

Z -parameter: .NET V(C) IB
Y-parameter: .NET I(Vc) VBE
H-parameter: .NET I(Vc) IB
S-parameter: .NET V(C) VBE
I1
I2
C
V1
V2
IB
IC
VBE
VCE

Figure 246 Network Parameter Configurations

Example

To calculate the H parameters, HSPICE or HSPICE RF uses the `.NET` statement.

```
.NET I(VC) IB
```

VC denotes the voltage at the C node, which is the collector of the BJT. With this statement, HSPICE or HSPICE RF uses the following equations to calculate H parameters immediately after AC analysis:

$$V1 = H11 \cdot I1 + H12 \cdot V2$$

$$I2 = H21 \cdot I1 + H22 \cdot V2$$

To calculate Hybrid parameters (H11 and H21), the DC voltage source (V_{CE}) sets V2 to zero, and the DC current source (IB) sets I1 to zero. Setting I1 and V2 to zero, precisely meets the conditions of the circuit under examination: the input current source is open-circuited, and the output voltage source shorts to ground.

A data file containing measured results can drive external DC biases applied to a BJT. Not all DC currents and voltages (at input and output ports) might be available. When you run a network analysis, examine the circuit and select suitable input and output variables. This helps you to obtain correctly-calculated results. The following example demonstrates HSPICE network analysis of a BJT or HSPICE RF.

Network Analysis Example: Bipolar Transistor

You can find the sample netlist for this example in the following directory:
$installdir/demo/hspice/bjt/net_ana.sp

.NET Parameter Analysis

```
Xij(z), ZIN(z), ZOUT(z), YIN(z), YOUT(z)
```

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.

Parameter	Description
z	Output type (HSPICE or HSPICE RF): ▪ R: real part. ▪ I: imaginary part. ▪ M: magnitude. ▪ P: phase. ▪ DB: decibel. ▪ T: group time delay (HSPICE RF does not support group time delays in AC analysis output).
YIN	Input admittance. For a one-port network, YIN is the same as Y11.
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)
.PRINT AC S22(M) S22(P) S21(R) H21(P) H12(R) S22(T)
```

Bandpass Netlist: Network Analysis Results

You can find the sample netlist for this example in the following directory:
$installdir/demo/hspice/filters/fbpnet.sp

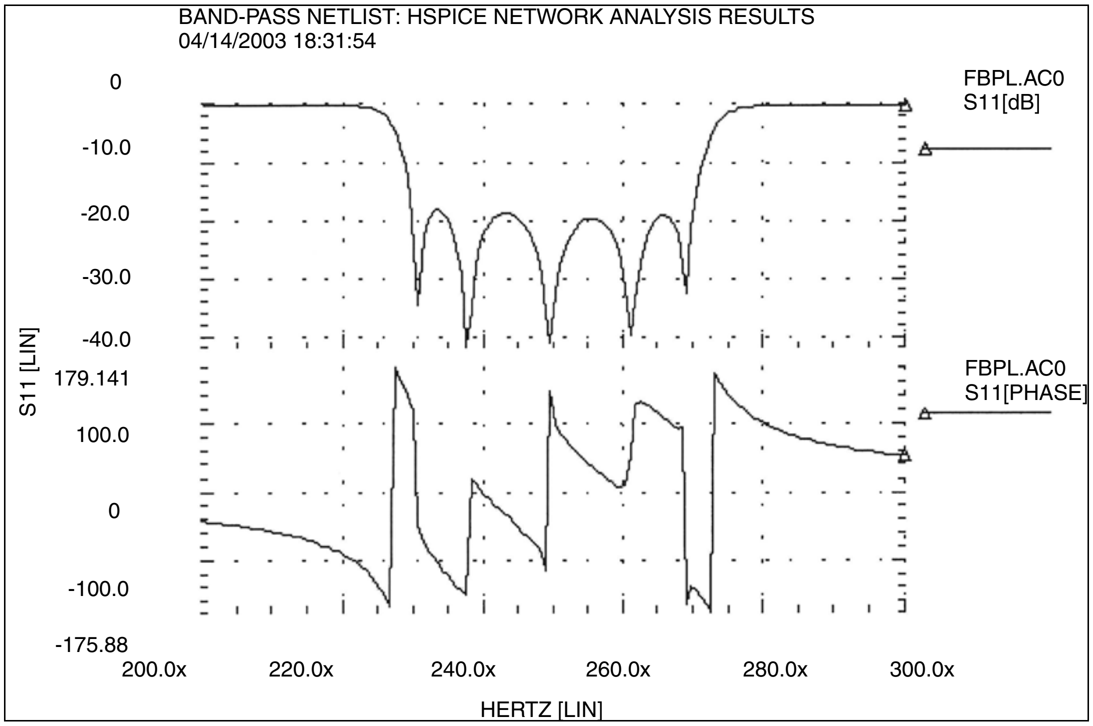


Figure 247 S11 Magnitude and Phase Plots

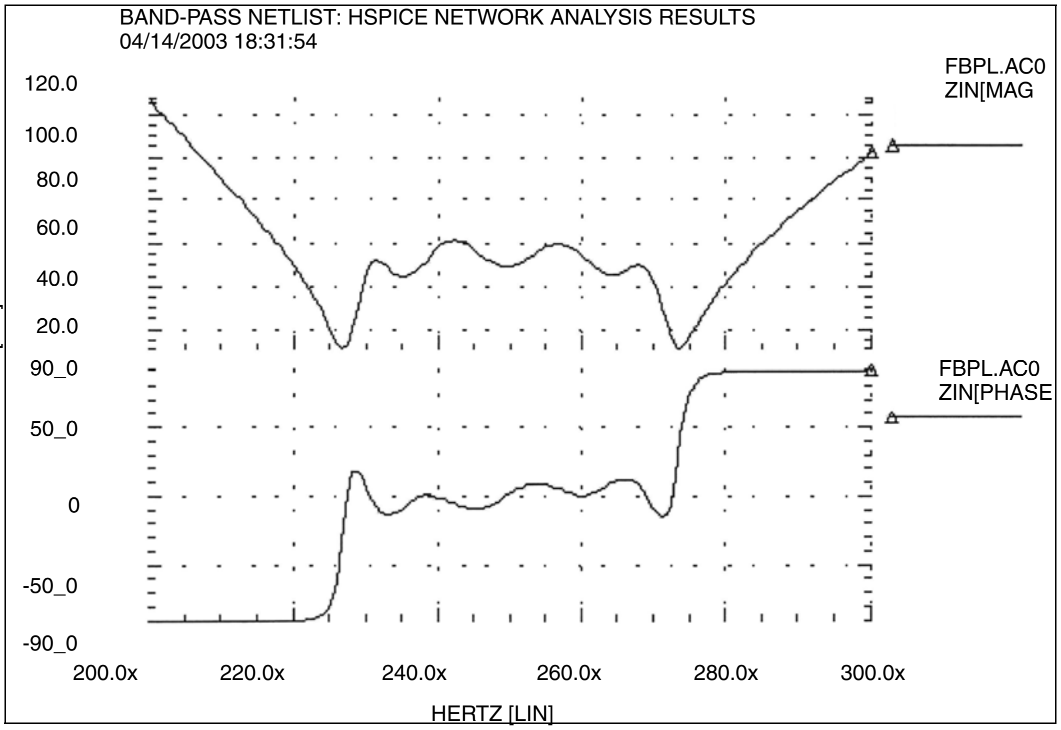


Figure 248 ZIN Magnitude and Phase Plots

Behavioral Modeling, Obsolete Functionality

Digital Stimulus Files

Complex transition files are difficult to process, if you use piecewise linear sources. You can use the `A2D` and `D2A` conversion functions, in the U-element, to simplify processing of transition files.

- The `A2D` function converts analog output to digital data.
- The `D2A` function converts digital input data to analog.

You can also export the output to either logic or VHDL simulators.

Op-Amp Subcircuit Generators (Behavioral Modeling

The subcircuit generator automatically designs operational amplifiers, to meet electrical specifications (such as PSRR, CMRR, and Vos). The generator produces component values, for each element in the design. When HSPICE combines these values, the resulting subcircuits simulate faster than conventional circuit-level implementations.

Op-Amp Model Generator

HSPICE uses the model generator to automatically design and simulate both board-level and IC op-amp designs. Here's how:

1. Start from the existing electrical specifications for a standard industrial operational amplifier.
2. Enter the specifications in the op-amp model statement.

   HSPICE automatically generates the internal components of the op-amp, to meet the specifications.
3. You can then call the design from a library, for a board-level simulation.

The HSPICE op-amp model is a subcircuit. It is about 20 times faster to simulate, than an actual transistor level op-amp. You can adjust the AC gain and phase to within 20 percent of the actual measured values, and set the transient slew rates accurately. This model does not contain high-order frequency response poles and zeros, so it can significantly differ from actual amplifiers, in predicting high-frequency instabilities.

You can use this model to represent normal amplifier characteristics, including:

- input offsets
- small signal gain
- transient effects

The op-amp subcircuit generator consists of a model, and one or more elements. Each element is in the form of a subcircuit call.

1. The model generates an output file of the op-amp equivalent circuit, which you can collect into libraries.

   The file name is the name of the model (mname), with an *.inc* extension.
2. After you generate the output file, other HSPICE input files can reference this subcircuit, using a `.SUBCKT` call to the model name.

3. The `.SUBCKT` call automatically searches for the file in the present directory.
4. It then searches the directories specified in any `.OPTION SEARCH ='directory_path_name'`.
5. Finally, it searches the directory where the DDL (Discrete Device Library) is located.

The amplifier element references the amplifier model.

If the model generator creates op-amp model that do not converge in DC analysis, use the `.IC` or `.NODESET` statement, to set the input nodes to the voltage that is halfway between the VCC and VEE. This balances the input nodes, and stabilizes the model.

Op-Amp Element Statement Format

`COMP=0` (internal compensation)

```
xa1 in- in+ out vcc vee modelname AV=val
```

`COMP=1` (external compensation)

```
xa1 in- in+ out comp1 comp2 vcc vee modelname AV=val
```

Parameter	Description
in-	Inverting input
in+	Non-inverting input
out	Output, single ended
vcc	Positive voltage supply
vee	Negative voltage supply
modelname	Subcircuit reference name

Op-Amp .MODEL Statement Format

```
.MODEL mname AMP parameter=value …
```

Parameter	Description
mname	Model name. Elements use this name to reference the model.
AMP	Identifies an amplifier model (HSPICE only).
parameter	Any model parameter, as described following this table.
value	Value assigned to a parameter.

Example

```
X0 IN- IN+ OUT0 VCC VEE ALM124
.MODEL ALM124 AMP
+ C2=   30.00P   SRPOS=   .5MEG   SRNEG=   .5MEG
+ IB=   45N   IBOS=   3N    VOS=   4M
+ FREQ=   1MEG   DELPHS=   25   CMRR=   85
+ ROUT=   50   AV=   100K   ISC=   40M
+ VOPOS=   14.5   VONEG=   -14.5   PWR=   142M
+ VCC=   16   VEE=   -16   TEMP=   25.00
+ PSRR=   100   DIS=   8.00E-16   JIS=   8.00E-16
```

Op-Amp Model Parameters

Table 110 shows the model parameters for op-amps. The defaults for these parameters depend on the DEF parameter setting. Table 111 on page 948 shows defaults for each of the three DEF settings.

Table 110 Op-Amp Model Parameters

Names (Alias)	Units	Default	Description
AV (AVD)	volt/volt		Amplifier gain, in volts out, per volt in. The DC ratio of the voltage in, to the voltage out. Typical gains are from 25k to 250k. If the frequency is too low, increase the negative and positive slew rates, or decrease DELPHS.

Table 110 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
CMRR	volt/volt		Common mode rejection ratio. This is usually between 80 and 110 dB. You can enter this value as 100 dB, or as 100000.
AV1K	volt/volt		Amplifier gain, at 1 kHz. Estimates the unity-gain bandwidth. You can express gain as actual voltage gain, or in decibels (a standard unit conversion). If you set AV1K, HSPICE ignores FREQ. A typical value for AV1K is AV1K=(unity gain freq)/1000.
C2	farad		Internal feedback compensation capacitance.For an internally-compensated amplifier, if you do not specify a capacitance value, the default is 30 pF. If the gain is high (above 500k), the internal compensation capacitor is probably different (typically 10 pF). For an externally-compensated amplifier (COMP=1), set C2 to 0.5 pF, as the residual internal capacitance.
COMP			Compensation level selector. If set to 1, it modifies the number of equivalent nodes, to include external compensation nodes. See `C2` for external compensation settings. COMP=0 internal compensation (default). COMP=1 external compensation.
DEF			Default model selector. Choose one of these: 0= generic (0.6 MHz bandwidth) (default) 1= ua741 (1.2 MHz bandwidth) 2= mc4560 (3 MHz bandwidth).

Table 110 Op-Amp Model Parameters (Continued)

Names (Alias)	**Units**	**Default**	**Description**
DELPHS	deg		Excess phase, at the unity gain frequency. Also called the *phase margin*. HSPICE measures DELPHS in degrees. Typical excess phases range from 5° to 50°. To determine DELPHS, subtract the phase at unity gain from 90°. The result is the phase margin. Use the same chart as used for the FREQ determination above. DELPHS interacts with FREQ (or AV1K). Values of DELPHS tend to lower the unity gain bandwidth, especially values greater than 20°. Pick the DELPHS closest to measured value, that does not reduce unity gain bandwidth more than 20%. Otherwise, the model might not have enough poles, to always return correct phase and frequency responses.
DIS	amp	1e-16	Saturation current, for diodes and BJTs.
FREQ (GBW, BW)	Hz		Unity gain frequency, measured in hertz. Typical frequencies are from 100 kHz to 3 MHz. If you do not specify this parameter, measure the open-loop frequency response at 0 dB voltage gain, and measure the actual compensation capacitance. Typical compensation is 30 pF, and single-pole compensation configuration. If AV1K > zero, HSPICE calculates unity gain frequency from AV1K; it ignores FREQ.

Table 110 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
IB	amp		Amount of current required to bias the input differential transistors. This is usually a fundamental electrical characteristic. Typical values are between 20 and 400 nA.
IBOS	amp		Input bias offset current, or *input offset current*. Amount of unbalanced current, between input differential transistors. Usually a fundamental electrical characteristic. Typical values are 10% to 20% of the IB.
ISC	amp		Input short circuit current – not always specified. Typical values are 5 to 25 mA. HSPICE can determine ISC from output characteristics (current sinking), as the maximum output sink current. ISC and ROUT interact with each other. If ROUT is too large for the ISC value, HSPICE reduces ROUT.
JIS	amp		JFET saturation current. Default=1e-16. You do not need to change this value.
LEVIN			Input level type selector. You can create only a BJT differential pair. LEVIN=1 BJT differential input stage.
LEVOUT			Output level type selector. You can create only a single-ended output stage. LEVOUT=1 single-ended output stage.
MANU			Manufacturer’s name. Add this to the model parameter list, to identify the source of model parameters. HSPICE prints the name in the final equivalent circuit.

Table 110 Op-Amp Model Parameters (Continued)

Names (Alias)	**Units**	**Default**	**Description**
PWR (PD)	watt		Total power dissipation, for the amplifier. Includes a calculated value for the op-amp input differential pair. If you set a high slew rate, and very low power, HSPICE issues a warning, and shows power dissipation only for an input differential pair.
RAC (r0ac, roac)	ohm		High-frequency output resistance. This typically is about 60% of ROUT. RAC usually ranges between 40 to 70 ohms, for op-amps with video drive capabilities.
ROUT	ohm		Low-frequency output resistance. To find this value, use the closed-loop output impedance graph. The impedance at about 1kHz, using the maximum gain, is close to ROUT. Gains of 1,000 and above show effective DC impedance, generally in the frequency region between 1k and 10 kHz. Typical ROUT values are 50 to 100 ohms.
SRNEG (SRN)	volt		Negative output slew rate. HSPICE extracts this value from a graph that shows the response for the voltage follower pulse. This is usually a 4 V or 5 V output change, with 10 to 20 V supplies. Measures the negative change in voltage, and the amount of time for the change.
SRPOS (SRP)	volt		Positive output slew rate. HSPICE extracts this value from a graph that shows the response for a voltage follower pulse. This is usually a 4 V or 5 V output change, with 10 to 20 V supplies. Measures the positive change in voltage, and the amount of time for the change. Typical slew rates are from 70 k to 700 k.

Table 110 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
TEMP	°C		Temperature, in degrees Celsius. Usually the temperature at which HSPICE measured model parameters, which is typically 25 °C.
VCC	volt		Positive power-supply reference voltage, for VOPOS. HSPICE measures the VOPOS amplifier, with respect to VCC.
VEE	volt		Negative power-supply voltage. HSPICE measures the VONEG amplifier, with respect to VCC.
VONEG (VON)	volt		Maximum negative output voltage. This is less than VEE (the negative power-supply voltage), by the internal voltage drop.
VOPOS (VOP)	volt		Maximum positive output voltage. This is less than VCC (the positive power supply voltage), by the internal voltage drop.
VOS	volt		Required input offset voltage, between input differential transistors, which zeroes output voltage. Usually a fundamental electrical characteristic. Typical values for bipolar amplifiers range from 0.1 mV to 10 mV. HSPICE measures VOS in volts. In some amplifiers, VOS can cause a failure to converge. If this occurs, set VOS to 0, or use the initial conditions for convergence.

Op-Amp Model Parameter Defaults

Table 111 Op-Amp Model Parameter Defaults

		Defaults		
Parameter	**Description**	**DEF=0**	**DEF=1**	**DEF=2**
AV	Amplifier voltage gain	160k	417k	200k
AV1K	Amplifier voltage gain, at 1 kHz	-	1.2 k	3 k
C2	Feedback capacitance	30 p	30 p	10 p
CMRR	Common-mode rejection ratio	96 db 63.1k	106 db 199.5k	90 db 31.63k
COMP	Compensation level selector	0	0	0
DEF	Default level selector	0	1	2
DELPHS	Delta phase, at unity gain	25×	17×	52×
DIS	Diode saturation current	8e-16	8e-16	8e-16
FREQ	Frequency, for unity gain	600 k	-	-
IB	Current, for input bias	30 n	250 n	40 n
IBOS	Current, for input bias offset	1.5 n	0.7 n	5 n
ISC	Current, for output short circuit	25 mA	25 mA	25 mA
LEVIN	Circuit-level selector, for input	1	1	1
LEVOUT	Circuit-level selector, for output	1	1	1
MANU	Manufacturer’s name	-	-	-
PWR	Power dissipation	72 mW	60 mW	50 mW
RAC	AC output resistance	0	75	70
ROUT	DC output resistance	200	550	100
SRPOS	Positive output slew rate	450 k	1 meg	1 meg

Table 111 Op-Amp Model Parameter Defaults (Continued)

		Defaults		
Parameter	**Description**	**DEF=0**	**DEF=1**	**DEF=2**
SRNEG	Negative output slew rate	450 k	800 k	800 k
TEMP	Temperature of model	25 deg	25 deg	25 deg
VCC	Positive supply voltage, for VOPOS	20	15	15
VEE	Negative supply voltage, for VONEG	-20	-15	-15
VONEG	Maximum negative output	-14	-14	-14
VOPOS	Maximum positive output	14	14	14
VOS	Input offset voltage	0	0.3 m	0.5 m

Simulation Results

The simulation results include the DC operating point analysis, for an input voltage of 0 v, and power supply voltages of ±15 v.

- The DC offset voltage is 3.3021 mV, which is less than that specified for the original `VOS` specification, in the op-amp `.MODEL` statement.
- The unity-gain frequency is 907.885 kHz, which is within 10% of the 1 MHz that the FREQ parameter (in the `.MODEL` statement) specifies.
- The required time rate, for a 1 V change in the output (from the `.MEASURE` statement), is 2.3 µs (from the `SRPOS` simulation result listing). This provides a slew rate of 0.434 mV/s, which is within about 12% of the 0.5 mV/s, specified in the `SRPOS` parameter of the `.MODEL` statement.
- The negative slew rate is almost exactly 0.5 mV/s, which is within 1% of the slew rate specified in the `.MODEL` statement.

Example

```
$$ FILE ALM124.SP
.OPTION NOMOD AUTOSTOP SEARCH=' '
.OP VOL
.AC DEC 10 1HZ 10MEG
.MODEL PLOTDB   PLOT XSCAL=2 YSCAL=3
.MODEL PLOTLOGX PLOT XSCAL=2
.GRAPH AC MODEL=PLOTDB VM(OUT0)
.GRAPH AC MODEL=PLOTLOGX VP(OUT0)
.TRAN 1U 40US 5US .15MS
.GRAPH V(IN) V(OUT0)
.MEASURE TRAN   'SRPOS'TRIG V(OUT0) VAL=2V RISE=1
+ TARG V(OUT0) VAL=3V RISE=1
.MEASURE TRAN   'SRNEG'TRIG V(OUT0) VAL=-2V FALL=1
+ TARG V(OUT0) VAL=-3V FALL=1
.MEASURE AC   'UNITFREQ'TRIG AT=1
+ TARG VDB(OUT0) VAL=0 FALL=1
.MEASURE AC   'PHASEMARGIN' FIND VP(OUT0)
+ WHEN VDB(OUT0)=0
.MEASURE AC   'GAIN(DB)'MAX VDB(OUT0)
.MEASURE AC   'GAIN(MAG)'MAX VM(OUT0)
VCC  VCC GND +15V
VEE  VEE GND -15V
VIN IN GND AC=1 PWL 0US   0V 1US   0V 1.1US +10V 15US +10V
+ 15.2US -10V 100US -10V
.MODEL ALM124 AMP
+ C2=   30.00P   SRPOS=   .5MEG   SRNEG=   .5MEG
+ IB=   45N   IBOS=   3N    VOS=   4M
+ FREQ=   1MEG   DELPHS=   25   CMRR=   85
+ ROUT=   50   AV=   100K   ISC=   40M
+ VOPOS=   14.5   VONEG=   -14.5   PWR=   142M
+ VCC=   16   VEE=   -16   TEMP=   25.00
+ PSRR=   100   DIS=   8.00E-16   JIS=   8.00E-16
*
```

Unity Gain Resistor Divider Mode

```
*
Rfeed   OUT0   IN-   10K   AC=10000G
RIN   IN   IN-   10K
RIN+   IN+   GND   10K
X0   IN-   IN+   OUT0   VCC VEE ALM124
ROUT0   OUT0   GND   2K
COUT0   OUT0   GND   100P
.END

***** OPERATING POINT STATUS IS VOLTAGE
***** SIMULATION TIME IS 0.
  NODE    =VOLTAGE   NODE   =VOLTAGE   NODE   =VOLTAGE
+ 0:IN   = 0.   0:IN+   =-433.4007U   0:IN-   = 3.3021M
+ 0:OUT0   = 7.0678M   0:VCC   = 15.0000   0:VEE   = -15.0000
unitfreq   = 907.855K   TARG   = 907.856K   TRIG   =   1.000
PHASEMARGIN   = 66.403
gain(db)   = 99.663   AT   =  1.000
FROM   = 1.000   TO   = 10.000X
gain(mag)   = 96.192K   AT   =  1.000
FROM   =  1.000    TO   = 10.000X
srpos   = 2.030U   TARG   = 35.471U   TRIG   = 33.442U
srneg   = 1.990U   TARG    =  7.064U   TRIG   =  5.074U
```

Behavioral Tunnel Diode

The following example uses a PWL VCCS to model a tunnel diode. HSPICE obtains the current characteristics for two `DELTA` values (50 μV and 10 μV). The IV characteristics, corresponding to `DELTA=10mv`, have sharper corners. HSPICE also displays the derivative of the current, with respect to voltage (GD). The GD value, around the breakpoints, changes in a linear fashion.

An example of a behavior tunnel diode is located in the following directory:

$*installdir*/demo/hspice/behave/tunnel.sp

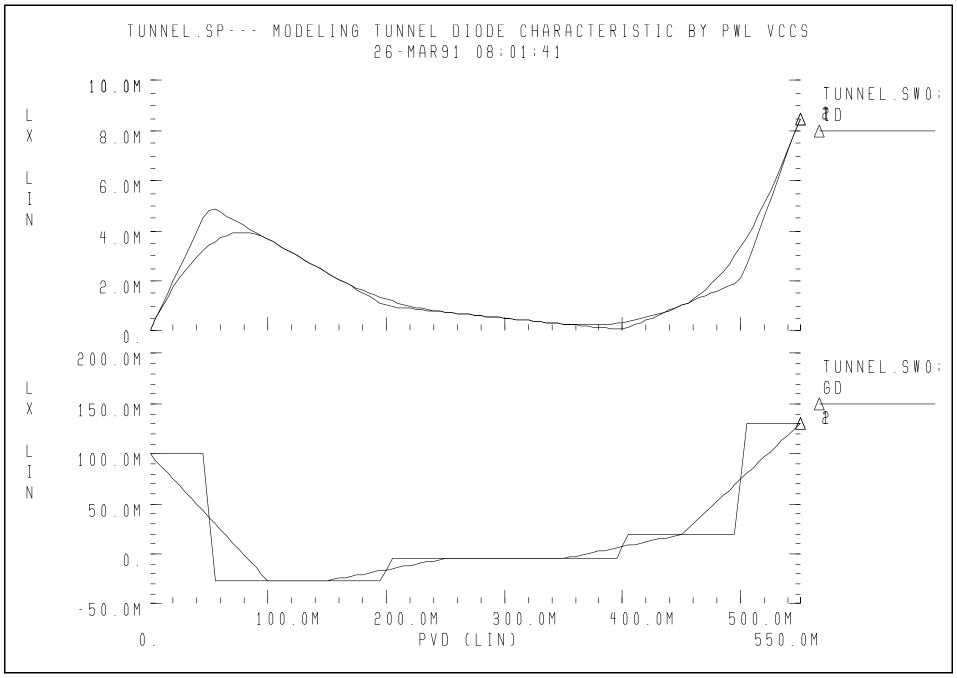


Figure 249 Tunnel Diode Characteristic

Behavioral Silicon-Controlled Rectifier (SCR)

To model the silicon controlled rectifier (SCR) characteristic, use a PWL CCVS, which provides a unique voltage value for any current value.

An example of a behavioral silicon-controlled rectifier is located in the following directory:

$installdir/demo/hspice/behave/pwl6.sp

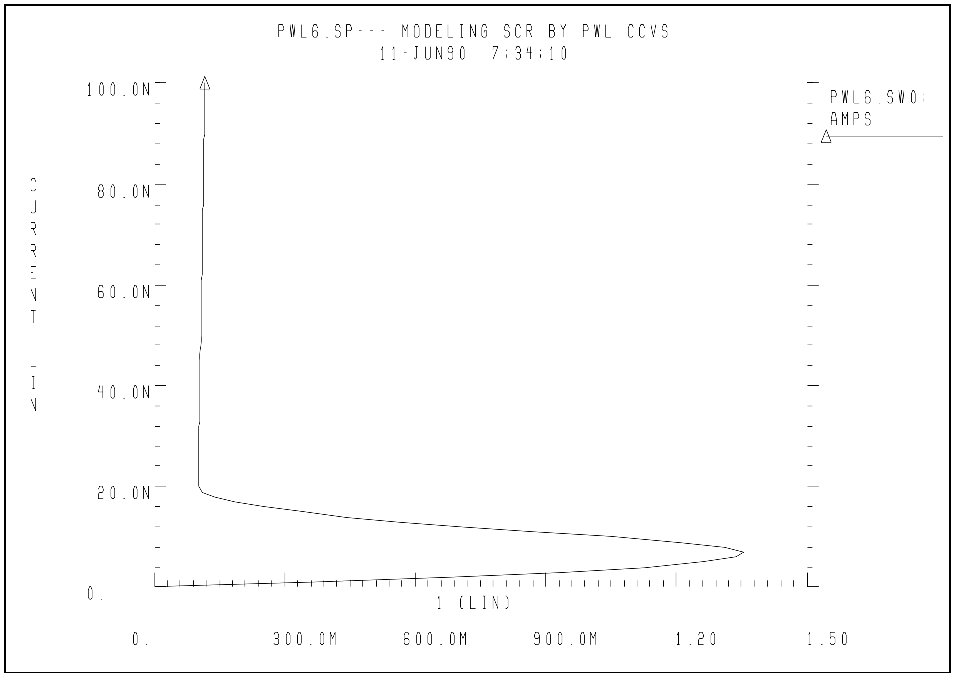


Figure 250 Silicon Controlled Rectifier

Behavioral Triode Vacuum Tube Subcircuit

The following example shows how to include the behavioral elements in a subcircuit, for very good triode tube action. The EA voltage source modifies the basic power law equation (current source GT), for better response in saturation.

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/triode.sp

he triode.sp file also contains the following sections:

- Control and Options
- Circuit
- Subcircuit Definition

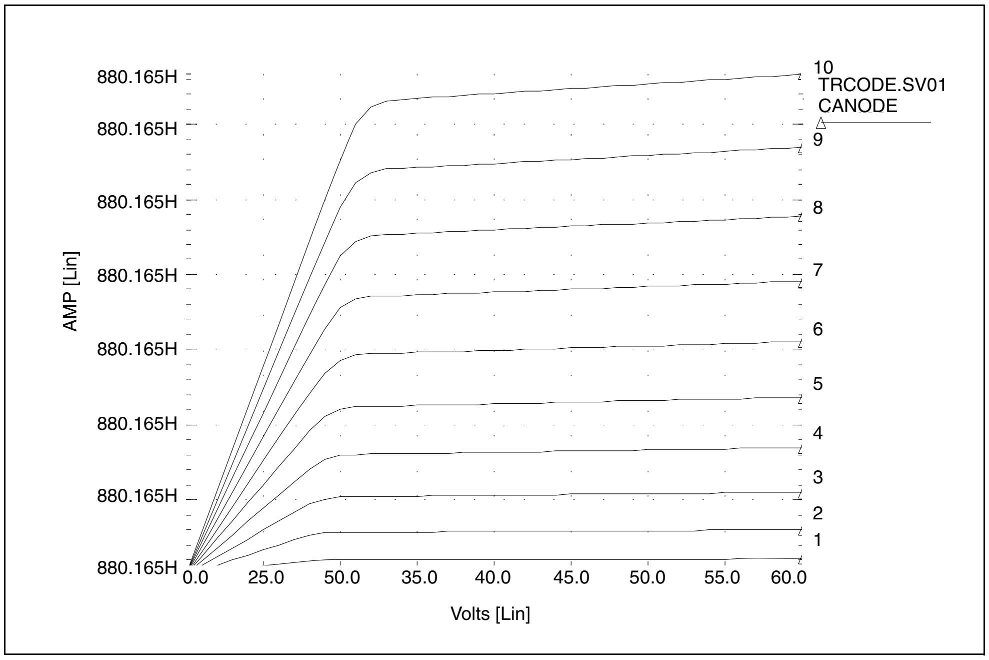


Figure 251 Triode Family of Curves

D

Transient Simulation with RUNLVL=0

Describes running a transient simulation with .OPTION RUNLVL=0.

The following sections discuss effects of changing the .OPTION RUNLVL setting from its default to 0. For full discussion of the transient simulation, see Chapter 14, Transient Analysis.

Controlling the Simulation with RUNLVL=0

The transient simulation runtime with .option RUNLVL=0 is much slower than with the default RUNLVL=3 control setting. For backward compatibility, all of the original HSPICE transient analysis algorithms are enabled when RUNLVL=0.

Without RUNLVL control (RUNLVL=0), 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.

Table 112 Transient Control Options with RUNLVL Turned Off, by Category

Method		Tolerance		Limit	
BYPASS	MAXORD	ABSH	RELH	AUTOSTOP	ITL3
CSHUNT	METHOD	ABSV	RELI	DELMAX	ITL4
DVDT	POST	ABSVAR	RELQ	DVTR	ITL5
GSHUNT	PROBE	ACCURAT	RELTOL	FS	RMAX
INTERP	PURETP	E BYTOL	RELV	FT	RMIN
ITRPRT	TRCON	CHGTOL	RELVAR	GMIN	VFLOOR
LVLTIM		DI	SLOPETOL		
		MBYPASS	TIMERES		
		MAXAMP	TRTOL		
		MU	VNTOL		

Simulation Accuracy

When .OPTION RUNLVL=0 is set, HSPICE uses the traditional control option values. These values aim for superior accuracy, within an acceptable amount of simulation time. The control options and their default settings (to maximize accuracy) are:

DVDT=4	LVLTIM=1	RMAX=5	SLOPETOL=0.75
FT=FS=0.25	BYPASS=1	BYTOL=MBYPASS x VNTOL=0.100m	

Note:

> `BYPASS` is on (set to 1), only when `DVDT=4`. For other `DVDT` settings, `BYPASS` is off (0). The `SLOPETOL` value is 0.75, only if `DVDT=4` and `LVLTIM=1`. For all other values of `DVDT` or `LVLTIM`, `SLOPETOL` defaults to 0.5.

Timestep Control for Accuracy

The `DVDT` control option selects the timestep control algorithm. For a description of the relationships between `DVDT` and other control options, see Selecting Timestep Control Algorithms on page 957.

The `DELMAX` control option also affects simulation accuracy. `DELMAX` specifies the maximum timestep size. If you do not set `.OPTION DELMAX`, HSPICE computes a `DELMAX` value. Factors that determine the computed `DELMAX` value are:

- `.OPTION RMAX` and `.OPTION FS`.
- Breakpoint locations for a PWL source.
- Breakpoint locations for a PULSE source.
- Smallest period for a SIN source.
- Smallest delay for a transmission line component.
- Smallest ideal delay for a transmission line component.
- `TSTEP` value, in a `.TRAN` analysis.
- Number of points, in an FFT analysis.

Use the `FS` and `RMAX` control options, to control the `DELMAX` value.

- `.OPTION FS`, which defaults to 0.25, scales the breakpoint interval in the `DELMAX` calculation.
- `.OPTION RMAX` defaults to 5 (if `DVDT`=4 and `LVLTIM`=1), and scales the `TSTEP` (timestep) size in the `DELMAX` calculation.

For circuits that contain oscillators or ideal delay elements, use `.OPTION DELMAX`, to set `DELMAX` to one-hundredth of the period or less.

`.OPTION ACCURATE` tightens the simulation options to output the most accurate set of simulation algorithms and tolerances. If you set `ACCURATE` to 1, HSPICE uses these control options:

Table 113 Control Option Settings When ACCURATE=1

DVDT=2	RELVAR=0.2	BYPASS=0	FT=FS=0.2	RELMOS=0.01
BYTOL=0	TIM=3	ABSVAR=0.2	RMAX=2	SLOPETOL=0.5

Guidelines for Choosing Accuracy Options

Use `.OPTION ACCURATE` for:

- Analog or mixed signal circuits.
- Circuits with long time constants, such as RC networks.
- Circuits with ground bounce.

Use the default options 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` or `RUNLVL=5`
- `RUNLVL=0` and `DVDT=3`.
- `RUNLVL=0`, `DVDT=4`, or `RUNLVL=3` (default). 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.

Selecting Timestep Control Algorithms

In HSPICE, you can select one of three dynamic timestep-control algorithms:

- Iteration Count Dynamic Timestep
- Local Truncation Error Dynamic Timestep
- DVDT Dynamic Timestep

Iteration Count Dynamic Timestep

The simplest dynamic timestep algorithm is the iteration count algorithm. The control options that control this algorithm are `.OPTION IMAX` and `.OPTION IMIN`.

Local Truncation Error Dynamic Timestep

The local truncation error (LTE) timestep algorithm uses a Taylor-series approximation to calculate the next timestep for a transient analysis. This algorithm uses the allowed LTE to calculate an internal timestep.

- If the calculated timestep is smaller than the current timestep, HSPICE 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 LTE timestep algorithm, set `LVLTIM=2` or `METHOD=GEAR`. The control options available with the algorithm for local truncation error, are:

```
TRTOL (default=7)
CHGTOL (default=1e-15)
RELQ (default=0.01)
```

For some circuits (such as magnetic core circuits), GEAR and LTE provide more accurate result than TRAP. You can use this method with circuits containing inductors, diodes, BJTs (even Level 4 and above), MOSFETs, or JFETs.

DVDT Dynamic Timestep

To select DVDT dynamic timestep algorithm, set the `LVLTIM` option to 1 or 3.

- If you set `LVLTIM=1`, the `DVDT` algorithm does not use timestep reversal. HSPICE 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`.
- 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 `.OPTION RELVAR` and `.OPTION ABSVAR` with the `DVDT` option to improve simulation time or accuracy. For faster simulation time, increase `RELVAR` and `ABSVAR` (but this might decrease accuracy).

Note:

If you need backward compatibility, use these options. Setting `.OPTION DVDT=3` automatically sets all of these values.

```
LVLTIM=1  RMAX=2  SLOPETOL=0.5
FT=FS=0.25  BYPASS=0  BYTOL=0.050
```

Index